Élan™SC520 Microcontroller

User's Manual

Order #22004B

The contents of this document are provided in connection with Advanced Micro Devices, Inc. ("AMD") products. AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to make changes to specifications and product descriptions at any time without notice. No license, whether express, implied, arising by estoppel or otherwise, to any intellectual property rights is granted by this publication. Except as set forth in AMD's Standard Terms and Conditions of Sale, AMD assumes no liability whatsoever, and disclaims any express or implied warranty, relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or infringement of any intellectual property right.

AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur. AMD reserves the right to discontinue or make changes to its products at any time without notice.

Trademarks

AMD, the AMD logo and combinations thereof, Am186, AMDebug, AMD Athlon, E86, K86, and Élan are trademarks; Am486 and Am5$_x$86 are registered trademarks; and FusionE86 is a service mark of Advanced Micro Devices, Inc.

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corp.

Product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

IF YOU HAVE QUESTIONS, WE'RE HERE TO HELP YOU.

The AMD customer service network includes U.S. offices, international offices, and a customer training center. Expert technical assistance is available from the AMD worldwide staff of field application engineers and factory support staff to answer E86™ family hardware and software development questions.

Frequently accessed numbers are listed below. Additional contact information is listed on the back of this manual. AMD's WWW site lists the latest phone numbers.

Technical Support

Answers to technical questions are available online, through e-mail, and by telephone.

Go to AMD's home page at **www.amd.com** and follow the Support link for the latest AMD technical support phone numbers, software, and Frequently Asked Questions.

For technical support questions on all E86 products, send e-mail to **epd.support@amd.com** (in the US and Canada) or **euro.tech@amd.com** (in Europe and the UK).

You can also call the AMD Corporate Applications Hotline at:

(800) 222-9323	Toll-free for U.S. and Canada
44-(0) 1276-803-299	U.K. and Europe hotline

WWW Support

For specific information on E86 products, access the AMD home page at **www.amd.com** and follow the Embedded Processors link. These pages provide information on upcoming product releases, overviews of existing products, information on product support and tools, and a list of technical documentation. Support tools include online benchmarking tools and CodeKit software—tested source code example applications. Many of the technical documents are available online in PDF form.

Questions, requests, and input concerning AMD's WWW pages can be sent via e-mail to **webfeedback@amd.com**.

Documentation and Literature Support

Data books, user's manuals, data sheets, application notes, and product CDs are free with a simple phone call. Internationally, contact your local AMD sales office for product literature.

To order literature, go to **www.amd.com/support/literature.html** or, in the US and Canada, call (800) 222-9323.

Third-Party Support

AMD FusionE86^{SM} program partners provide an array of products designed to meet critical time-to-market needs. Products and solutions available include emulators, hardware and software debuggers, board-level products, and software development tools, among others. The WWW site and the *E86™ Family Products Development Tools CD*, order #21058, describe these solutions. In addition, mature development tools and applications for the x86 platform are widely available in the general marketplace.

TABLE OF CONTENTS

PREFACE INTRODUCTION XXIII
Élan™SC520 Microcontroller xxiii
Purpose of this Manual xxiii
Intended Audience xxiii
Overview of this Manual xxiii
Related Documents xxiv
AMD Documentation xxiv
Additional Information xxv
Documentation Conventions xxv

CHAPTER 1 ARCHITECTURAL OVERVIEW 1-1
1.1 Élan™SC520 Microcontroller 1-1
1.1.1 Distinctive Characteristics 1-1
1.2 Block Diagram 1-2
1.3 Architectural Overview 1-4
1.3.1 Industry-Standard x86 Architecture 1-4
1.3.2 AMDebug™ Technology for Advanced Debugging 1-4
1.3.3 Industry-Standard PCI Bus Interface 1-5
1.3.4 High-Performance SDRAM Controller 1-5
1.3.5 ROM/Flash Controller 1-5
1.3.6 Flexible Address-Mapping 1-5
1.3.7 General-Purpose (GP) Bus Interface 1-6
1.3.8 Clock Generation 1-6
1.3.9 Integrated Peripherals 1-7
1.3.10 JTAG Boundary Scan Test Interface 1-7
1.3.11 System Testing and Debugging Features 1-8
1.4 Applications 1-8
1.4.1 Smart Residential Gateway 1-8
1.4.2 Thin Client 1-8
1.4.3 Digital Set Top Box 1-9
1.4.4 Telephone Line Concentrator 1-9

CHAPTER 2 PIN INFORMATION 2-1
2.1 Overview. 2-1
2.2 Logic Symbols 2-1
2.3 Signal Descriptions. 2-4

CHAPTER 3 SYSTEM INITIALIZATION 3-1
3.1 Overview. 3-1
3.1.1 Native Embedded Initialization Sequence 3-1
3.1.2 BIOS Initialization Sequence 3-3
3.1.3 Memory-Mapped Configuration Region (MMCR) 3-3
3.1.4 Reset Event 3-4
3.1.5 Reset Vector and Reset Segment 3-5
3.2 Configuring the SDRAM Controller 3-6
3.3 Identifying the CPU Core 3-7
3.4 Setting the CPU Speed 3-7
3.5 Configuring External GP Bus Devices 3-7
3.6 Configuring the Pin Multiplexing 3-8
3.7 Configuring the Programmable Address Region (PAR) Registers 3-8
3.7.1 Specifying Pages and Regions 3-9

3.7.2 Address Region Attributes . . . 3-12
3.7.2.1 Write-Protect Attribute . . . 3-12
3.7.2.2 Cacheability Control Attribute . . . 3-12
3.7.2.3 Code Execution Attribute . . . 3-12
3.7.2.4 Performance Considerations . . . 3-12
3.7.3 PAR Register Priority . . . 3-13
3.7.4 External GP Bus Devices . . . 3-13
3.7.4.1 Single Device (an A/D Converter) Using One Chip Select . . . 3-14
3.7.4.2 Single Device That Performs Its Own Decode . . . 3-14
3.7.4.3 Multiple Devices On One Chip Select . . . 3-14
3.7.5 PCI Bus Devices . . . 3-15
3.7.5.1 VGA Controller on the PCI Bus . . . 3-15
3.7.5.2 Network Adapter for Remote Program Loading . . . 3-16
3.7.6 External ROM Devices . . . 3-17
3.7.6.1 Boot ROM Device Mapping for BIOS Shadowing . . . 3-17
3.7.6.2 Two Banks of Flash for an Execute-In-Place (XIP) Operating System . . . 3-17
3.7.7 SDRAM Regions . . . 3-18
3.7.7.1 Setting Up DMA Buffers . . . 3-18
3.7.7.2 Write-Protected Code Segments . . . 3-18
3.8 Configuring the Interrupt Mapping . . . 3-19
3.8.1 Edge-Sensitive or Level-Triggered Interrupts . . . 3-19
3.8.2 Interrupt Mapping . . . 3-19
3.8.3 Interrupt Polarity . . . 3-20
3.9 Configuring the Programmable I/O Pins . . . 3-20
3.10 Configuring the PCI Host Bridge and Arbitration . . . 3-20
3.11 Disabling Internal Peripherals . . . 3-21

CHAPTER 4 SYSTEM ADDRESS MAPPING 4-1
4.1 Overview . . . 4-1
4.2 Registers . . . 4-2
4.3 Operation . . . 4-3
4.3.1 Programming External Memory, Buses, and Chip Selects . . . 4-4
4.3.2 Programmable Address Region (PAR) Registers . . . 4-5
4.3.3 Memory Space . . . 4-7
4.3.3.1 SDRAM Space . . . 4-8
4.3.3.2 ROM/Flash Space . . . 4-8
4.3.3.3 GP Bus Memory Space . . . 4-9
4.3.3.4 PCI Bus Memory Space . . . 4-9
4.3.3.5 Memory-Mapped Configuration Region (MMCR) Registers Space . . . 4-9
4.3.4 I/O Space . . . 4-11
4.3.4.1 Configuration Base Address (CBAR) Register . . . 4-11
4.3.4.2 PCI Configuration Space . . . 4-12
4.3.4.3 PCI I/O Space . . . 4-12
4.3.4.4 PC/AT-Compatible I/O Peripherals Region . . . 4-13
4.3.4.5 GP Bus I/O Region . . . 4-15
4.3.5 Configuration Information . . . 4-15
4.3.5.1 Configuring ROM/Flash Space . . . 4-15
4.3.5.2 Configuring SDRAM Address Space . . . 4-15
4.3.5.3 Configuring GP Bus Peripheral Space . . . 4-16
4.3.5.4 Configuring the Élan™SC520 Microcontroller for Windows® Compatibility . . . 4-17
4.3.5.5 Configuring PCI Bus Devices . . . 4-18
4.3.6 Interrupts . . . 4-18
4.3.7 Software Considerations . . . 4-18
4.4 Initialization . . . 4-21

CHAPTER 5 CLOCK GENERATION AND CONTROL 5-1

5.1 Overview. 5-1
5.2 Block Diagram 5-2
5.3 System Design 5-3
5.3.1 Clock Pin Loading 5-4
5.3.2 Selecting a Crystal 5-4
5.3.2.1 Running the Élan™SC520 Microcontroller at 33.333 MHz 5-5
5.3.3 Bypassing Internal Oscillators 5-5
5.4 Registers 5-6
5.5 Operation 5-7
5.5.1 Internal Clocks 5-7
5.5.1.1 CPU 5-7
5.5.1.2 PCI Bus 5-7
5.5.1.3 SDRAM Controller 5-7
5.5.1.4 ROM/Flash Interface 5-7
5.5.1.5 GP Bus. 5-7
5.5.1.6 GP-DMA Controller 5-8
5.5.1.7 Programmable Interval Timer. 5-8
5.5.1.8 General-Purpose Timers 5-8
5.5.1.9 Software Timer. 5-8
5.5.1.10 Watchdog Timer. 5-8
5.5.1.11 Real-Time Clock 5-8
5.5.1.12 UART Serial Ports 5-8
5.5.1.13 Synchronous Serial Interface. 5-8
5.5.2 Using the CLKTIMER[CLKTEST] Pin 5-9
5.6 Initialization. 5-9

CHAPTER 6 RESET GENERATION 6-1

6.1 Overview. 6-1
6.2 Block Diagram 6-1
6.3 System Design 6-2
6.4 Registers 6-3
6.5 Operation 6-3
6.5.1 System Reset 6-4
6.5.2 System Reset with SDRAM Retention 6-6
6.5.3 Soft CPU Reset 6-7
6.5.4 GP Bus Reset 6-7
6.5.5 PCI Reset 6-7
6.5.6 RTC Reset 6-7
6.5.7 Determining Reset Sources 6-8
6.5.8 CPU A20 Gate Support 6-8
6.5.9 Clocking Considerations 6-8
6.5.10 Software Considerations 6-8
6.5.11 Latency 6-9
6.6 Initialization. 6-9

CHAPTER 7 Am5$_X$86® CPU 7-1
7.1 Overview. 7-1
7.2 Block Diagram 7-1
7.3 Registers 7-1
7.4 Operation 7-3
7.4.1 Floating Point Unit (FPU) 7-3
7.4.2 Cache Memory Management 7-4
7.4.3 Clocking Considerations 7-4
7.4.4 Interrupts 7-5
7.4.5 Latency 7-5
7.5 Initialization. 7-5
7.5.1 Hard CPU Reset 7-5
7.5.2 Soft CPU Reset 7-5

CHAPTER 8 SYSTEM ARBITRATION 8-1
8.1 Overview. 8-1
8.2 Block Diagram 8-1
8.3 Registers 8-2
8.4 Operation 8-3
8.4.1 Operating Modes 8-3
8.4.1.1 Nonconcurrent Arbitration Mode 8-3
8.4.1.2 Concurrent Arbitration Mode 8-4
8.4.2 CPU Bus Arbiter 8-5
8.4.2.1 CPU Arbitration Protocol 8-5
8.4.2.2 CPU Cache Snooping 8-6
8.4.2.3 Accessing the PCI Host Bridge Target. 8-6
8.4.2.4 GP Bus DMA Arbitration 8-7
8.4.2.5 Arbitration During Clock Speed Changes 8-7
8.4.3 PCI Bus Arbiter 8-7
8.4.3.1 PCI Bus Arbitration Protocol 8-8
8.4.3.2 Bus Parking 8-10
8.4.3.3 Rearbitration 8-10
8.4.4 Bus Cycles 8-11
8.4.4.1 CPU Bus Arbitration. 8-11
8.4.4.2 CPU Bus Cache Write-Back 8-12
8.4.4.3 CPU-to-PCI Cycle 8-14
8.4.4.4 PCI Bus Arbitration 8-15
8.4.4.5 PCI Bus Arbitration Parking 8-16
8.4.4.6 Nonconcurrent Mode Arbitration 8-18
8.4.5 Interrupts 8-19
8.4.6 Software Considerations 8-19
8.4.7 Latency 8-20
8.4.7.1 Simple Rotating Priority Latency 8-20
8.4.7.2 High-Priority Queue Latency 8-21
8.4.7.3 Low-Priority Queue Latency. 8-21
8.4.7.4 CPU Latency 8-21
8.4.7.5 Nonconcurrent Arbitration Mode Latency 8-21
8.4.7.6 Concurrent Arbitration Mode Latency 8-22
8.4.7.7 Concurrent Arbitration Mode Bus Parking Latency . . . 8-22
8.5 Initialization. 8-22

CHAPTER 9 PCI BUS HOST BRIDGE 9-1
9.1 Overview. 9-1
9.2 Block Diagram 9-1
9.3 System Design 9-2
9.3.1 PCI Clocking 9-5
9.3.1.1 Running the Élan™SC520 Microcontroller
at 33.333 MHz 9-6

9.4 Registers 9-7
9.5 Operation 9-8
9.5.1 Unsupported PCI Bus Functions 9-8
9.5.1.1 Unsupported PCI Bus Configuration Registers 9-9
9.5.2 Configuration Information 9-9
9.5.2.1 Generating PCI Bus Configuration Cycles. 9-10
9.5.3 Élan™SC520 Microcontroller’s Host Bridge as PCI Bus Master . 9-11
9.5.3.1 Write Posting 9-11
9.5.3.2 Read Cycles. 9-12
9.5.3.3 Delayed Transaction Support. 9-12
9.5.3.4 Host Bridge Master Bus Cycles 9-12
9.5.4 Élan™SC520 Microcontroller’s Host Bridge as PCI Bus Target . . 9-18
9.5.4.1 PCI Host Bridge Target Address Space. 9-18
9.5.4.2 PCI Bus Command Support 9-19
9.5.4.3 $\overline{\text{DEVSEL}}$ Timing. 9-19
9.5.4.4 Delayed Transaction Support. 9-19
9.5.4.5 Address FIFO. 9-20
9.5.4.6 PCI Host Bridge FIFOs and Prefetching 9-20
9.5.4.7 Burst Ordering 9-21
9.5.4.8 Maintaining Data Coherency 9-21
9.5.4.9 PCI Host Bridge Target Bus Cycles 9-22
9.5.5 Interrupts 9-27
9.5.6 Latency 9-28
9.5.6.1 Master Latency 9-28
9.5.6.2 Target Latency 9-28
9.6 Initialization. 9-29

CHAPTER 10 SDRAM CONTROLLER 10-1

10.1 Overview. 10-1
10.2 Block Diagram 10-1
10.3 System Design 10-1
10.3.1 SDRAM Pins 10-5
10.3.2 SDRAM Clocking 10-6
10.3.3 SDRAM Loading 10-8
10.4 Registers 10-10
10.5 Operation 10-11
10.5.1 SDRAM Support 10-11
10.5.2 SDRAM Addressing 10-12
10.5.2.1 Supported SDRAM Devices. 10-13
10.5.2.2 Page Size. 10-16
10.5.3 Error Correction Code (ECC) 10-16
10.5.4 Buffering 10-17
10.5.5 SDRAM Control Configuration 10-18
10.5.5.1 Refresh Control 10-18
10.5.5.2 Drive-Strength Selection 10-19
10.5.5.3 Write Buffer Test Mode 10-19
10.5.5.4 Operation Mode Select 10-20
10.5.6 SDRAM Timing Configuration 10-20
10.5.6.1 $\overline{\text{CAS}}$ Latency (C_L) 10-20
10.5.6.2 $\overline{\text{RAS}}$ Precharge (T_{RP}) 10-21
10.5.6.3 $\overline{\text{RAS}}$-to-$\overline{\text{CAS}}$ Delay (T_{RCD}). 10-21
10.5.6.4 $\overline{\text{RAS}}$-to-$\overline{\text{RAS}}$ or Auto-Refresh-to-$\overline{\text{RAS}}$ (T_{RC}) 10-21
10.5.6.5 Minimum $\overline{\text{RAS}}$ (T_{RAS}). 10-22
10.5.7 Bus Cycles 10-22
10.5.7.1 SDRAM Burst Read Cycle. 10-22
10.5.7.2 SDRAM Write Cycle. 10-23
10.5.7.3 ECC SDRAM Cycles 10-24
10.5.7.4 SDRAM Auto Refresh Cycle 10-26

10.5.7.5 SDRAM Mode Register Access Cycles 10-27
10.5.8 Interrupts 10-27
10.5.9 Software Considerations 10-28
10.5.9.1 ECC Errors. 10-28
10.5.9.2 Buffer Disabling During SDRAM Configuration 10-28
10.5.9.3 Write Protection 10-28
10.5.10 Latency 10-28
10.6 Initialization. 10-29
10.6.1 Programmable Reset 10-29
10.6.2 SDRAM Device Initialization 10-30
10.6.2.1 Operation Mode Select 10-30
10.6.2.2 NOP Command 10-31
10.6.2.3 Precharge Command. 10-31
10.6.2.4 Auto Refresh Command 10-31
10.6.2.5 Mode Register Programming. 10-31
10.6.3 Boot Process 10-32
10.6.4 SDRAM Sizing Algorithm 10-32
10.6.4.1 Determining the Number of Columns
for an External Bank 10-33
10.6.4.2 Determining the Number of Internal Banks 10-34
10.6.4.3 Determining the True External Bank Ending Address 10-35

CHAPTER 11 WRITE BUFFER AND READ BUFFER 11-1
11.1 Overview. 11-1
11.2 Block Diagram 11-2
11.3 System Design 11-3
11.4 Registers 11-4
11.5 Operation 11-4
11.5.1 Write Buffer 11-5
11.5.1.1 Write Buffer Disabled. 11-5
11.5.1.2 Write Buffer Enabled 11-5
11.5.1.3 Write Buffer Watermark 11-9
11.5.2 Read Buffer and the Read-Ahead Feature 11-10
11.5.2.1 Read-Ahead Feature Disabled. 11-10
11.5.2.2 Read-Ahead Feature Enabled 11-10
11.5.3 DMA Considerations 11-11
11.5.4 PCI Considerations 11-12
11.5.4.1 Write Cycles. 11-12
11.5.4.2 Read Cycles. 11-12
11.5.5 Software Considerations 11-13
11.5.6 SDRAM Bandwidth Improvements 11-13
11.6 Initialization. 11-15

CHAPTER 12 ROM/FLASH CONTROLLER 12-1
12.1 Overview. 12-1
12.2 Block Diagram 12-2
12.3 System Design 12-2
12.3.1 Voltage Isolation 12-3
12.4 Registers 12-5
12.5 Operation 12-5
12.5.1 ROM Support 12-5
12.5.1.1 Supported Device Types 12-6
12.5.2 ROM Control and Timing Configuration 12-7
12.5.2.1 ROM Location 12-7
12.5.2.2 ROM Width 12-7
12.5.2.3 Operating Mode 12-7
12.5.2.4 Access Timing 12-8

12.5.3 Bus Cycles 12-9
12.5.3.1 Single CPU Read Access 12-9
12.5.3.2 Page-Mode Read Access 12-10
12.5.3.3 Cache-Line Fill. 12-11
12.5.3.4 Writing to Flash Devices 12-11
12.5.4 Software Considerations 12-12
12.5.4.1 Address Decoding 12-12
12.5.4.2 Programming Flash Memory 12-12
12.5.5 Latency 12-13
12.6 Initialization. 12-14

CHAPTER 13 GENERAL-PURPOSE BUS CONTROLLER 13-1
13.1 Overview. 13-1
13.2 Block Diagram 13-1
13.3 System Design 13-1
13.3.1 GP Bus Loading 13-4
13.3.2 Voltage Translation 13-4
13.4 Registers 13-5
13.5 Operation 13-6
13.5.1 Programmable Bus Interface Timing 13-7
13.5.1.1 Timing Requirements. 13-7
13.5.1.2 Using GPRDY with Programmable Timing. 13-8
13.5.1.3 Using GP Bus Echo Mode with Programmable Timing 13-8
13.5.2 I/O-Mapped and Memory-Mapped Device Support 13-9
13.5.3 Chip Select Qualification 13-9
13.5.4 Data Sizing and Unaligned Accesses 13-9
13.5.5 Sharing the Address and Data Bus
with the ROM/Flash Controller 13-10
13.5.6 GP Bus Echo Mode 13-10
13.5.7 DMA Interface 13-11
13.5.8 Usage Scenarios 13-11
13.5.8.1 Compatibility with Common ISA Devices. 13-11
13.5.8.2 Interfacing with a Super I/O Controller. 13-13
13.5.8.3 Interfacing with an AMD Enhanced
Serial Communications Controller (8 MHz) 13-14
13.5.9 Bus Cycles 13-16
13.5.9.1 8-Bit Data Access of an 8-Bit I/O Device 13-16
13.5.9.2 16-Bit Data Access of a 16-Bit I/O Device 13-17
13.5.9.3 16-Bit Data Access of an 8-Bit I/O Device 13-17
13.5.9.4 32-Bit Data Access of an 8-Bit I/O Device 13-18
13.5.9.5 32-Bit Data Access of a 16-Bit I/O Device 13-18
13.5.9.6 8-Bit Data Access of a 16-Bit I/O Device 13-19
13.5.9.7 $\overline{\text{GPIOCS16}}$ and $\overline{\text{GPMEMCS16}}$ Timing. 13-19
13.5.9.8 Wait States. 13-20
13.5.10 Interrupts 13-21
13.5.11 Latency 13-21
13.5.11.1 8/16-Bit GP Bus Width 13-21
13.5.11.2 Slow GP Bus Cycles 13-21
13.5.11.3 Noncacheable GP Bus. 13-21
13.6 Initialization. 13-22

CHAPTER 14 GP BUS DMA CONTROLLER 14-1

14.1 Overview 14-1

14.2 Block Diagram 14-1

14.3 System Design 14-3

14.4 Registers 14-4

14.4.1 Memory-Mapped Registers 14-4

14.4.2 Direct-Mapped Registers 14-6

14.5 Operation 14-8

14.5.1 GP-DMA Transfers 14-8

14.5.1.1 GP-DMA Initiators 14-9

14.5.1.2 GP-DMA Channel Mapping 14-10

14.5.2 Operating Modes 14-10

14.5.2.1 Normal GP-DMA Mode 14-10

14.5.2.2 Enhanced GP-DMA Mode 14-11

14.5.3 Addressing GP-DMA Channels 14-11

14.5.3.1 Addressing In Normal GP-DMA Mode 14-11

14.5.3.2 Addressing In Enhanced GP-DMA Mode 14-12

14.5.4 GP-DMA Transfer Modes 14-12

14.5.4.1 Single Transfer Mode 14-12

14.5.4.2 Demand Transfer Mode 14-12

14.5.4.3 Block Transfer Mode 14-13

14.5.4.4 Transfer Types 14-13

14.5.4.5 Automatic Initialization Control 14-14

14.5.4.6 Priority 14-15

14.5.4.7 Buffer Chaining 14-15

14.5.5 Bus Cycles 14-16

14.5.5.1 GP Bus I/O to SDRAM 14-16

14.5.5.2 GP-DMA Read with Cache Hit 14-17

14.5.6 GP Bus Echo Mode 14-17

14.5.7 Clocking Considerations 14-18

14.5.8 Interrupts 14-18

14.5.9 Software Considerations 14-18

14.5.10 Latency 14-18

14.5.10.1 Nonpreemptive Latency 14-18

14.5.10.2 Preemptive Latency 14-19

14.6 Initialization 14-19

14.6.1 Example Configurations 14-19

14.6.1.1 Configuring an 8-Bit Channel in Normal GP-DMA Mode 14-19

14.6.1.2 Configuring a 16-Bit Channel in Normal GP-DMA Mode 14-20

14.6.1.3 Configuring an 8-Bit Channel in Enhanced GP-DMA Mode 14-20

14.6.1.4 Configuring a 16-Bit Channel in Enhanced GP-DMA Mode 14-21

CHAPTER 15 PROGRAMMABLE INTERRUPT CONTROLLER 15-1

15.1 Overview 15-1

15.2 Block Diagram 15-2

15.3 System Design 15-2

15.4 Registers 15-4

15.5 Operation 15-7

15.5.1 Interrupt Flow Sequence 15-7

15.5.2 Interrupt Sources 15-8

15.5.2.1 Hardware-Generated Interrupts 15-8

15.5.3 Interrupt Source Routing 15-10

15.5.3.1 Polarity Inversion of Interrupt Requests 15-10

15.5.3.2 PC/AT Compatibility 15-12
15.5.3.3 Floating Point Errors 15-12
15.5.3.4 Disabling the Slave Controllers 15-13
15.5.4 Edge-Triggered or Level-Sensitive Interrupts 15-13
15.5.5 Interrupt Sharing 15-13
15.5.6 Non-Maskable Interrupts and Routing 15-14
15.5.6.1 Sharing NMIs 15-14
15.5.7 Priority Types 15-16
15.5.8 Configuration Information 15-16
15.5.8.1 Programming 15-16
15.5.8.2 PC/AT Configuration 15-18
15.5.9 Software Considerations 15-18
15.5.9.1 Interrupt Sharing 15-18
15.5.9.2 Disabling the Slave Controllers 15-19
15.5.9.3 Detecting Invalid Interrupt Requests 15-19
15.5.9.4 Floating Point Unit Error Handling 15-19
15.6 Initialization. 15-20

CHAPTER 16 PROGRAMMABLE INTERVAL TIMER 16-1
16.1 Overview. 16-1
16.2 Block Diagram 16-1
16.3 System Design 16-1
16.4 Registers 16-2
16.5 Operation 16-3
16.5.1 PIT Channel 0 16-3
16.5.2 PIT Channel 1 16-3
16.5.3 PIT Channel 2 16-4
16.5.4 Operating Modes 16-4
16.5.4.1 Mode 0: Interrupt on Terminal Count 16-4
16.5.4.2 Mode 1: Hardware-Retriggerable One-Shot 16-4
16.5.4.3 Mode 2: Rate Generator 16-5
16.5.4.4 Mode 3: Square Wave Mode 16-5
16.5.4.5 Mode 4: Software-Triggered Strobe 16-5
16.5.4.6 Mode 5: Hardware-Triggered Strobe 16-5
16.5.5 Clocking Considerations 16-6
16.5.5.1 Internal Clock. 16-6
16.5.5.2 External Clock 16-6
16.5.6 Interrupts 16-6
16.5.7 Software Considerations 16-6
16.5.7.1 Using the PIT Clock Source in PC/AT-Compatible Systems 16-6
16.6 Initialization. 16-7

CHAPTER 17 GENERAL-PURPOSE TIMERS 17-1
17.1 Overview. 17-1
17.2 Block Diagram 17-1
17.3 System Design 17-1
17.4 Registers 17-2
17.5 Operation 17-3
17.5.1 GP Timer 0 and GP Timer 1 17-3
17.5.2 GP Timer 2 17-4
17.5.3 Operating Modes 17-4
17.5.3.1 Interrupt on Terminal Count Mode 17-4
17.5.3.2 Hardware Retrigger Mode 17-4
17.5.3.3 Alternate Compare Mode. 17-4
17.5.3.4 Square Wave Mode 17-4
17.5.3.5 Continuous Mode. 17-4
17.5.3.6 Prescaler Mode 17-4

17.5.4 Configuration Information 17-5
17.5.5 Clocking Considerations 17-5
17.5.5.1 Internal Clock. 17-5
17.5.5.2 External Clock 17-6
17.5.6 Interrupts 17-6
17.5.7 Software Considerations 17-6
17.5.7.1 Combining GP Timer Count Elements. 17-6
17.5.7.2 Reading the Cascaded 32-Bit Timer 17-6
17.6 Initialization. 17-8

CHAPTER 18 SOFTWARE TIMER 18-1
18.1 Overview. 18-1
18.2 Block Diagram 18-1
18.3 Registers 18-2
18.4 Operation 18-2
18.4.1 Configuration Information 18-3
18.5 Initialization. 18-3

CHAPTER 19 WATCHDOG TIMER 19-1
19.1 Overview. 19-1
19.2 Block Diagram 19-1
19.3 Registers 19-2
19.4 Operation 19-3
19.4.1 Configuration Information 19-3
19.4.1.1 Keyed Sequences 19-3
19.4.1.2 Interrupt Request Generation 19-4
19.4.1.3 System Reset Generation 19-4
19.4.1.4 Time-Out Duration 19-4
19.4.2 Interrupts 19-5
19.4.3 AMDebug™ Technology Interface 19-5
19.4.4 Software Considerations 19-5
19.5 Initialization. 19-6

CHAPTER 20 REAL-TIME CLOCK 20-1
20.1 Overview. 20-1
20.2 Block Diagram 20-1
20.3 System Design 20-3
20.3.1 Backup Battery Considerations 20-3
20.3.1.1 System with an External Backup Battery. 20-3
20.3.1.2 System without an External Backup Battery 20-4
20.3.2 Selecting and Interfacing a 32.768-kHz Crystal 20-5
20.3.3 Using an External RTC 20-5
20.4 Registers 20-6
20.5 Operation 20-7
20.5.1 Configuration Information 20-7
20.5.1.1 Configuring the Hour Format 20-7
20.5.1.2 Programming the Date and Time. 20-8
20.5.1.3 Generating Periodic Interrupts. 20-8
20.5.1.4 Using the Alarm Function 20-9
20.5.1.5 Handling Year 2000 Issues 20-9
20.5.2 Clocking Considerations 20-9
20.5.3 Interrupts 20-9
20.5.4 Software Considerations 20-10
20.5.4.1 Initializing the RTC Divider Chain 20-10
20.5.4.2 Accessing the CMOS Memory. 20-10
20.5.4.3 Legacy NMI Enable Bit Moved. 20-10
20.6 Initialization. 20-10
20.6.1 RTC Reset 20-11

CHAPTER 21 UART SERIAL PORTS 21-1
21.1 Overview. 21-1
21.2 Block Diagram 21-1
21.3 System Design 21-2
21.4 Registers 21-3
21.5 Operation 21-5
21.5.1 Data Transmission 21-6
21.5.1.1 16450-Compatible UART Mode 21-6
21.5.1.2 16550-Compatible UART Mode 21-7
21.5.2 Data Reception 21-7
21.5.2.1 16450-Compatible UART Mode 21-7
21.5.2.2 16550-Compatible UART Mode 21-7
21.5.3 Error Handling 21-8
21.5.3.1 Parity Error. 21-8
21.5.3.2 Framing Error 21-8
21.5.3.3 Break Indication 21-8
21.5.3.4 Error Reporting 21-8
21.5.4 Configuration Information 21-9
21.5.4.1 Baud Rate 21-9
21.5.4.2 Hardware Flow Control 21-9
21.5.4.3 Operating Modes 21-9
21.5.5 DMA Interface 21-10
21.5.5.1 Transmit DMA 21-10
21.5.5.2 Receive DMA. 21-10
21.5.6 Clocking Considerations 21-10
21.5.7 Interrupts 21-10
21.5.7.1 Serial Port Interrupts 21-12
21.5.7.2 DMA Interrupts. 21-12
21.5.7.3 Interrupt Disable. 21-13
21.6 Initialization. 21-13

CHAPTER 22 SYNCHRONOUS SERIAL INTERFACE 22-1
22.1 Overview. 22-1
22.2 Block Diagram 22-1
22.3 System Design 22-1
22.4 Registers 22-2
22.5 Operation 22-3
22.5.1 Usage Scenarios 22-3
22.5.1.1 Four-Pin Interface 22-3
22.5.1.2 Three-Pin Interface 22-3
22.5.2 Configuration Information 22-5
22.5.2.1 Bit Order. 22-5
22.5.2.2 Clock Idle State 22-5
22.5.2.3 Clock Phase. 22-5
22.5.3 Bus Cycles 22-5
22.5.3.1 4-Bit Read Cycle 22-6
22.5.3.2 Burst, 16-Bit, and 32-Bit Cycles. 22-7
22.5.4 Clocking Considerations 22-7
22.5.5 Interrupts 22-7
22.5.6 Software Considerations 22-8
22.6 Initialization. 22-8

CHAPTER 23 PROGRAMMABLE INPUT/OUTPUT 23-1
23.1 Overview. 23-1
23.2 Block Diagram 23-1
23.3 System Design 23-2
23.4 Registers 23-4
23.5 Operation 23-4
23.5.1 Configuration Information 23-5
23.5.1.1 PIO Pins and Simple Input. 23-5
23.5.1.2 PIO Pins and Simple Output 23-5
23.5.2 Software Considerations 23-5
23.6 Initialization. 23-6

CHAPTER 24 SYSTEM TEST AND DEBUGGING 24-1
24.1 Overview. 24-1
24.2 System Design 24-1
24.2.1 Loading 24-2
24.3 Registers 24-2
24.4 Operation 24-3
24.4.1 System Test Mode 24-3
24.4.1.1 Pin Functions in System Test Mode. 24-3
24.4.1.2 Using the System Test Mode Interface 24-4
24.4.1.3 SDRAM Write Cycle in System Test Mode 24-4
24.4.1.4 SDRAM Read Cycle in System Test Mode 24-5
24.4.1.5 Tracing Transactions on the ROM Interface 24-5
24.4.1.6 Tracing Transactions on the GP Bus Interface 24-6
24.4.2 Write Buffer Test Mode 24-7
24.4.2.1 Using the Write Buffer Test Mode Interface 24-7
24.4.2.2 SDRAM Write Cycle in Write Buffer Test Mode 24-8
24.4.2.3 SDRAM Read Cycle in Write Buffer Test Mode 24-8
24.4.3 Other Debugging Features 24-10
24.4.3.1 Nonconcurrent Arbitration Mode 24-10
24.4.3.2 Echoing Integrated Peripheral Accesses
on the GP Bus 24-10
24.4.3.3 Summary of Additional System Debugging Features. 24-10
24.4.4 Software Considerations 24-11
24.4.5 Latency 24-11
24.5 Initialization. 24-12

CHAPTER 25 BOUNDARY SCAN TEST INTERFACE 25-1
25.1 Overview. 25-1
25.2 Block Diagram 25-1
25.3 System Design 25-2
25.3.1 JTAG Pin Strapping 25-2
25.4 Registers 25-2
25.5 Operation 25-2
25.5.1 Instruction Register 25-3
25.5.1.1 Implemented Instructions. 25-3
25.5.2 Configuration Information 25-5
25.5.2.1 Instruction Path 25-5
25.5.2.2 Bypass Path. 25-5
25.5.2.3 Main Data Scan Path 25-5
25.5.2.4 Serial Debug Port Data Register 25-14
25.5.2.5 Device Identification Register 25-14
25.5.3 Test Access Port (TAP) Controller 25-15
25.5.3.1 TAP Controller States. 25-15
25.5.4 Bus Cycles 25-19
25.5.5 Clocking Considerations 25-20
25.6 Initialization. 25-20

CHAPTER 26 AMDebug™ TECHNOLOGY 26-1

26.1 Overview. 26-1

26.2 Block Diagram 26-2

26.3 System Design 26-2

26.3.1 Connecting the AMDebug™ Port 26-3

26.3.2 Mechanical Specifications for the Target Connector 26-5

26.3.3 Locating the Connector on the Target System 26-5

26.4 Operation 26-6

26.4.1 On-Chip Trace Cache 26-7

26.4.2 Software Performance Profiling 26-7

INDEX Index-1

LIST OF FIGURES

Figure 1-1 Élan™SC520 Microcontroller Block Diagram . . . 1-3
Figure 1-2 Élan™SC520 Microcontroller-Based Smart Residential Gateway Reference Design . . . 1-10
Figure 1-3 Élan™SC520 Microcontroller-Based Thin Client Reference Design . . . 1-11
Figure 1-4 Élan™SC520 Microcontroller-Based Digital Set Top Box Reference Design . . . 1-12
Figure 1-5 Élan™SC520 Microcontroller-Based Telephone Line Concentrator Reference Design . . . 1-13
Figure 2-1 Logic Diagram by Interface . . . 2-2
Figure 2-2 Logic Diagram by Default Pin Function . . . 2-3
Figure 3-1 Initial Near Jump Example . . . 3-6
Figure 3-2 Programmable Address Region (PAR) Register Format . . . 3-10
Figure 3-3 Programmable Address Region (PAR) Register Worksheet . . . 3-11
Figure 4-1 Programmable Address Region (PAR) Register Format . . . 4-6
Figure 4-2 System Memory Map . . . 4-7
Figure 4-3 System I/O Map . . . 4-11
Figure 5-1 Clock Source Block Diagram . . . 5-2
Figure 5-2 System Clock Distribution Block Diagram . . . 5-3
Figure 5-3 Bypassing the 32.768-kHz Oscillator . . . 5-5
Figure 5-4 Bypassing the 33-MHz Oscillator . . . 5-6
Figure 5-5 Clock Routing for the CLKTEST Pin . . . 5-9
Figure 6-1 Reset Controller Block Diagram . . . 6-2
Figure 6-2 PRGRESET Timing . . . 6-6
Figure 6-3 Power-On Reset Sequence of Events . . . 6-9
Figure 7-1 Am5$_x$86® CPU Block Diagram . . . 7-2
Figure 8-1 System Arbitration Block Diagram . . . 8-2
Figure 8-2 Skipped Master Example . . . 8-5
Figure 8-3 CPU Bus Rotating Priority Queue . . . 8-6
Figure 8-4 External PCI Master Arbitration Queues . . . 8-9
Figure 8-5 Host Bridge Master Arbitration Queue . . . 8-9
Figure 8-6 CPU Bus Arbitration . . . 8-11
Figure 8-7 CPU Bus Cache Write-Back . . . 8-13
Figure 8-8 CPU-to-PCI Cycle . . . 8-14
Figure 8-9 PCI Bus Arbitration . . . 8-15
Figure 8-10 PCI Bus Concurrent Mode Arbitration Parking . . . 8-16
Figure 8-11 Nonconcurrent Mode Arbitration . . . 8-18
Figure 8-12 Simple Rotating Priority Queue . . . 8-20
Figure 9-1 PCI Interface Block Diagram . . . 9-2
Figure 9-2 Élan™SC520 Microcontroller Connection to an External PCI Bus Target . . . 9-3
Figure 9-3 Élan™SC520 Microcontroller Connection to an External PCI Bus Master . . . 9-4
Figure 9-4 Élan™SC520 Microcontroller $\overline{\text{SERR}}$ and $\overline{\text{PERR}}$ Connection . . . 9-5
Figure 9-5 PCI Bus Clocking Example 1: Lightly Loaded System . . . 9-6
Figure 9-6 PCI Bus Clocking Example 2: Heavily Loaded System . . . 9-6
Figure 9-7 PCI Configuration Address (PCICFGADR) Register . . . 9-10
Figure 9-8 CPU Read Cycle to the PCI Bus . . . 9-13
Figure 9-9 CPU Read Cycle to the PCI Bus with External Target Retry . . . 9-14
Figure 9-10 CPU Posted Write Cycle to the PCI Bus . . . 9-15
Figure 9-11 Am5$_x$86 CPU Non-Posted Write Cycle to the PCI Bus . . . 9-16
Figure 9-12 CPU Write Cycles to Internal PCI Bus Configuration Registers . . . 9-17
Figure 9-13 CPU Read Cycles from Internal PCI Bus Configuration Registers . . . 9-18
Figure 9-14 External PCI Bus Master Posted Write to SDRAM . . . 9-23
Figure 9-15 External PCI Master SDRAM Read (Delayed Transaction) . . . 9-24
Figure 9-16 PCI Host Bridge Target Disconnect . . . 9-26
Figure 10-1 SDRAM Controller Block Diagram . . . 10-2
Figure 10-2 Detailed Block Diagram of SDRAM Controller . . . 10-3
Figure 10-3 SDRAM Bank Configuration . . . 10-4
Figure 10-4 Example Configuration of a 168-Pin SDRAM DIMM . . . 10-5

Figure 10-5 SDRAM Clock Generation. 10-7
Figure 10-6 Alternate SDRAM Clock Generation with External Clock Driver 10-7
Figure 10-7 SDRAM Burst Read Cycle (Read-Ahead Feature Disabled) (Page Miss/Page Hit). . 10-22
Figure 10-8 SDRAM Write Cycle (Write Buffer and ECC Disabled) (Page Miss/page Hit) 10-23
Figure 10-9 SDRAM CPU Burst Write (Write Buffer and ECC Disabled) (Page Miss/Page Hit) . . 10-24
Figure 10-10 SDRAM Burst Read Cycle with ECC Enabled. 10-25
Figure 10-11 SDRAM Read-Modify-Write Cycle (for Data Write) with ECC Enabled (Page Hit). . . 10-26
Figure 10-12 SDRAM Auto Refresh Cycle 10-27
Figure 10-13 SDRAM Mode Register Access. 10-27
Figure 11-1 Write Buffer and Read Buffer Block Diagram (SDRAM Subsystem) 11-2
Figure 11-2 Write Buffer and Read Buffer Block Diagram 11-3
Figure 11-3 Write Buffer Merging Example. 11-7
Figure 11-4 Write Buffer Collapsing Example. 11-8
Figure 11-5 Write Buffer Read-Merging Example. 11-9
Figure 11-6 Bus Thrashing with Write Buffer Disabled and Enabled 11-14
Figure 12-1 ROM Controller Block Diagram 12-2
Figure 12-2 Voltage Isolation Examples 12-4
Figure 12-3 Page-Mode ROM: Fetching Four Words from a 16-Bit ROM. 12-6
Figure 12-4 Non-Page-Mode ROM: Fetching Four Words from a 16-Bit ROM 12-8
Figure 12-5 Page-Mode ROM: Fetching Four Doublewords (Aligned) from a 32-Bit ROM. 12-8
Figure 12-6 Page-Mode ROM: Fetching Four Doublewords (Unaligned) from an 8-Bit ROM. . . . 12-8
Figure 12-7 Multiple Accesses: Data Amounts Smaller than One Doubleword (2 Bytes) from an 8-Bit ROM. 12-10
Figure 12-8 Page Access for Fetching Four Doublewords from a 32-Bit ROM (Burst Sequence: 2-1-1-1) 12-10
Figure 12-9 Page Access for Fetching Two Doublewords from a 16-Bit ROM 12-11
Figure 12-10 Cache-Line Fill (Fetching Four Doublewords from a 32-Bit ROM). 12-11
Figure 12-11 Word Write Cycle to Flash Memory. 12-12
Figure 13-1 GP Bus Controller System Block Diagram 13-2
Figure 13-2 Example: Using an External Data Buffer to Address Excess Loading 13-4
Figure 13-3 Example: Using a Voltage Translator. 13-5
Figure 13-4 GP Bus Timing Format 13-8
Figure 13-5 Élan™SC520 Microcontroller Interfacing with a Super I/O Controller 13-13
Figure 13-6 Timing Diagram of a Super I/O Interface 13-14
Figure 13-7 Élan™SC520 Microcontroller Interfacing with an Am85C30 13-15
Figure 13-8 Timing Diagram of an Am85C30 Interface 13-16
Figure 13-9 8-Bit Data Access of an 8-Bit I/O Device. 13-16
Figure 13-10 16-Bit Data Access of a 16-Bit I/O Device. 13-17
Figure 13-11 16-Bit Data Access of an 8-Bit I/O Device. 13-17
Figure 13-12 32-Bit Data Access of an 8-Bit I/O Device. 13-18
Figure 13-13 32-Bit Data Access of a 16-Bit I/O Device. 13-18
Figure 13-14 8-Bit Data Access of a 16-Bit I/O Device. 13-19
Figure 13-15 16-Bit Access of a 16-Bit I/O Device 13-20
Figure 13-16 GPRDY Timing 13-21
Figure 14-1 GP-DMA Controller Block Diagram 14-2
Figure 14-2 Master and Slave Core Cascading Diagram 14-3
Figure 14-3 GP-DMA Read Transfer. 14-13
Figure 14-4 GP-DMA Write Transfer. 14-14
Figure 14-5 GP-DMA Verify Transfer 14-14
Figure 14-6 GP-DMA Read in Demand Transfer Mode 14-16
Figure 14-7 GP-DMA Read Transfer with Cache Hit (Write-Back Cache) 14-17
Figure 15-1 Programmable Interrupt Controller (PIC) Block Diagram 15-3
Figure 15-2 Interrupt Sources. 15-9
Figure 15-3 Interrupt Source Routing 15-11
Figure 15-4 NMI Routing. 15-15
Figure 16-1 Programmable Interval Timer Block Diagram 16-2
Figure 17-1 General-Purpose Timers Block Diagram. 17-2
Figure 18-1 Software Timer Block Diagram 18-1

Figure 19-1 Watchdog Timer Block Diagram 19-2
Figure 20-1 Real-Time Clock Block Diagram 20-2
Figure 20-2 RTC Voltage Monitor Block Diagram 20-3
Figure 20-3 Circuit with Backup Battery 20-4
Figure 20-4 Circuit without Backup Battery. 20-5
Figure 21-1 UART Block Diagram. 21-2
Figure 21-2 UART Frame Configuration 21-5
Figure 21-3 UART Frame Transmission 21-5
Figure 22-1 SSI Block Diagram. 22-2
Figure 22-2 SSI Four-Pin Interface 22-4
Figure 22-3 SSI Simultaneous Transmit and Receive. 22-4
Figure 22-4 SSI Three-Pin Interface 22-4
Figure 22-5 SSI Typical Half-Duplex Communication, Non-Inverted Phase and Clock Modes. . . 22-4
Figure 22-6 SSI Clock Phase and Clock Idle State: Effects on Data 22-6
Figure 22-7 SSI 4-Bit Read Cycle: Full-Duplex, Non-Inverted Phase, Non-Inverted Clock 22-6
Figure 22-8 SSI Back-to-Back Transactions for Full-duplex,
Microwire-Compatible Configuration 22-7
Figure 22-9 SSI Timing: TC_INT and BSY_STA Bits 22-8
Figure 23-1 PIO Signal Block Diagram 23-2
Figure 24-1 System Test Mode Timing During a SDRAM Write Cycle (Page Hit) 24-5
Figure 24-2 System Test Mode Timing During an SDRAM Read Cycle (Page Miss). 24-5
Figure 24-3 Write Buffer Test Mode Timing During an SDRAM Write Cycle (Page Hit) 24-8
Figure 24-4 Write Buffer Test Mode Timing During a SDRAM Read Cycle (Page Miss) 24-9
Figure 25-1 Logical Structure of Boundary Scan Register 25-1
Figure 25-2 Serial Debug Port Data Register Format. 25-14
Figure 25-3 Device Identification Register Format 25-14
Figure 25-4 Test Access Port Controller State Diagram 25-15
Figure 25-5 Test Logic Operation: Data Scan. 25-19
Figure 25-6 Test Logic Operation: Instruction Scan 25-20
Figure 26-1 AMDebug™ Technology Software Architecture. 26-2
Figure 26-2 12-Pin Connector Format 26-4
Figure 26-3 20-Pin Serial Connector Format 26-4
Figure 26-4 Mechanical Specifications for AMDebug™ Technology Target Connector 26-5
Figure 26-5 Locating the Target Connector 26-6

LIST OF TABLES

Table 0-1 Documentation Notation xxv
Table 2-1 Signal Descriptions Table Definitions 2-4
Table 2-2 Signal Descriptions 2-5
Table 3-1 CPUID Codes 3-7
Table 3-2 Example PAR Programming: Single Device Using One Chip Select 3-14
Table 3-3 Example PAR Programming: Single Device That Performs Its Own Decode 3-14
Table 3-4 Example PAR Programming: Multiple Devices on One Chip Select 3-14
Table 3-5 Example PAR Programming: VGA Controller on the PCI Bus 3-15
Table 3-6 Example PAR Programming: COM3 with VGA Present on the PCI Bus 3-16
Table 3-7 Example PAR Programming: Network Adapter for Remote Program Loading 3-16
Table 3-8 Example PAR Programming: Boot ROM Device Mapping for BIOS Shadowing 3-17
Table 3-9 Example PAR Programming: First Bank of Flash for XIP Operating System. 3-17
Table 3-10 Example PAR Programming: Second Bank of Flash for XIP Operating System 3-18
Table 3-11 Example PAR Programming: Setting Up DMA Buffers 3-18
Table 3-12 Example PAR Programming: Write-Protected Code Segments 3-19
Table 4-1 Address Decoding Registers—Memory-Mapped 4-2
Table 4-2 Address Decoding Registers—Direct-Mapped 4-2
Table 4-3 Bus Master Address Spaces 4-3
Table 4-4 Memory and I/O Space Summary 4-4
Table 4-5 PC/AT Peripherals I/O Map 4-14
Table 5-1 Clock Start-up and Lock Times 5-2
Table 5-2 Clock Signals Shared with Other Interfaces 5-3
Table 5-3 Timing Error as It Translates to Clock Accuracy 5-5
Table 5-4 Clock Control Registers—Memory-Mapped 5-6
Table 6-1 Reset Generation Registers—Memory-Mapped 6-3
Table 6-2 Reset Generation Registers—Direct-Mapped 6-3
Table 6-3 Élan™SC520 Microcontroller Reset Sources 6-4
Table 6-4 States of Cores after System Reset 6-5
Table 7-1 Am5$_x$86® CPU Registers—Memory-Mapped 7-1
Table 7-2 Am5$_x$86® CPU Registers—Direct-Mapped 7-1
Table 7-3 Cache Configuration Options 7-4
Table 8-1 System Arbitration Registers—Memory-Mapped 8-2
Table 9-1 PCI Host Bridge Registers—Memory-Mapped 9-7
Table 9-2 PCI Host Bridge Registers—Direct-Mapped 9-8
Table 9-3 PCI Host Bridge Registers—PCI Indexed 9-8
Table 10-1 SDRAM Clock Loading Estimates Based on Device Width 10-6
Table 10-2 Estimated Capacitance (4-Bit SDRAM Devices) 10-8
Table 10-3 Estimated Capacitance (8-Bit SDRAM Devices) 10-8
Table 10-4 Estimated Capacitance (16-Bit SDRAM Devices) 10-9
Table 10-5 Estimated Capacitance (32-Bit SDRAM Devices) 10-9
Table 10-6 SDRAM Controller Registers—Memory-Mapped 10-10
Table 10-7 Address Mapping to MAx Signals for SDRAM Devices 10-12
Table 10-8 SDRAM Devices Supported with Column Boundary Specification 10-13
Table 10-9 Column Address Configuration Settings for SDRAM. 10-15
Table 10-10 SDRAM Page Sizes. 10-16
Table 10-11 SDRAM Refresh Rates 10-18
Table 10-12 Load Mode Register Settings 10-31
Table 11-1 SDRAM Signals Shared with Other Interfaces 11-4
Table 11-2 SDRAM Buffer Control Registers—Memory-Mapped 11-4
Table 12-1 ROM/Flash Data Bus Connection Options 12-1
Table 12-2 ROM Signals Shared with Other Interfaces. 12-3
Table 12-3 ROM Controller Registers—Memory-Mapped. 12-5
Table 12-4 Example: ROM Access Timing and Wait States 12-9
Table 12-5 Accesses and ROM Width. 12-9
Table 12-6 CFGx Pinstrap Configuration Options for $\overline{\text{BOOTCS}}$ 12-14

Table 13-1 GP Bus Signals Shared with Other Interfaces . . . 13-3
Table 13-2 GP Bus Registers—Memory-Mapped . . . 13-5
Table 13-3 GP Bus Echo Mode Minimum Timing . . . 13-9
Table 13-4 Cross-Reference Table of ISA Signals and GP Bus Signals . . . 13-12
Table 13-5 Example Super I/O Controller Interface Timing . . . 13-13
Table 13-6 Example AMD Enhanced Serial Communications Controller Interface Timing . . . 13-15
Table 13-7 Differentiating Upper/Lower Byte Access of 16-Bit Devices . . . 13-19
Table 13-8 Dynamic Bus Sizing Override of Programmed Data Width . . . 13-20
Table 14-1 GP-DMA Signals Shared with Other Interfaces . . . 14-4
Table 14-2 GP-DMA Controller Registers—Memory-Mapped . . . 14-4
Table 14-3 GP-DMA Controller Registers—Direct-Mapped . . . 14-7
Table 14-4 Supported GP-DMA Initiator/Target Combinations . . . 14-9
Table 14-5 GP-DMA Channel Mapping . . . 14-10
Table 14-6 8-Bit GP-DMA Channel Address Generation . . . 14-12
Table 14-7 16-Bit GP-DMA Channel Address Generation . . . 14-12
Table 14-8 GP-DMA Cycle Types . . . 14-16
Table 15-1 Programmable Interrupt Controller Signals Shared with Other Interfaces . . . 15-2
Table 15-2 Programmable Interrupt Controller Registers—Memory-Mapped . . . 15-4
Table 15-3 Programmable Interrupt Controller Registers—Direct-Mapped . . . 15-6
Table 15-4 PC/AT Interrupt Channel Mapping . . . 15-12
Table 16-1 Programmable Interval Timer Signals Shared with Other Interfaces . . . 16-1
Table 16-2 Programmable Interval Timer Configuration Registers—Memory-Mapped . . . 16-2
Table 16-3 Programmable Interval Timer Configuration Registers—Direct-Mapped . . . 16-3
Table 16-4 PIT Internal Clock Source . . . 16-6
Table 16-5 PIT External Clock Source . . . 16-6
Table 17-1 General-Purpose Timer Signals Shared with Other Interfaces . . . 17-1
Table 17-2 General-Purpose Timer Registers—Memory-Mapped . . . 17-2
Table 17-3 GP Timers Internal Clock Sources . . . 17-5
Table 17-4 GP Timers External Clock Sources (Using a 33.333 MHz Crystal) . . . 17-6
Table 18-1 Software Timer Configuration Registers—Memory-Mapped . . . 18-2
Table 19-1 Watchdog Timer Registers—Memory-Mapped . . . 19-2
Table 19-2 Watchdog Timer Time-Out Duration . . . 19-4
Table 20-1 Real-Time Clock Registers—Memory-Mapped . . . 20-6
Table 20-2 Real-Time Clock Registers—Direct-Mapped . . . 20-6
Table 20-3 Real-Time Clock Registers—RTC Indexed . . . 20-6
Table 20-4 Using RATE_SEL to Specify a Periodic Interrupt Rate . . . 20-8
Table 21-1 UART Signals Shared with Other Interfaces . . . 21-2
Table 21-2 Connection of DTE to DTE . . . 21-3
Table 21-3 UART Registers—Memory-Mapped . . . 21-3
Table 21-4 UART Registers—Direct-Mapped . . . 21-4
Table 21-5 Baud Rates, Divisors, and Clock Source . . . 21-9
Table 21-6 UART Interrupt Programming Summary . . . 21-11
Table 21-7 Serial Port Interrupt and Interrupt Priority . . . 21-12
Table 22-1 Synchronous Serial Interface Registers—Memory-Mapped . . . 22-2
Table 23-1 PIO Signals Shared with Other Interfaces . . . 23-3
Table 23-2 PIO Registers—Memory-Mapped . . . 23-4
Table 23-3 PIO Configuration Summary . . . 23-5
Table 24-1 System Test and Debugging Signals Shared with Other Interfaces . . . 24-2
Table 24-2 System Test and Debugging Registers—Memory-Mapped . . . 24-2
Table 24-3 WBMSTR2–WBMSTR0 Pin Definition During Write Buffer Write Cycles . . . 24-8
Table 24-4 WBMSTR2–WBMSTR0 Pin Definition During SDRAM Read Cycles . . . 24-9
Table 25-1 Chip Test and Debugging Registers . . . 25-2
Table 25-2 Test Access Port Instruction Set . . . 25-3
Table 25-3 Main Data Scan Path . . . 25-5
Table 26-1 AMDebug™ Technology Connector Pins . . . 26-3

INTRODUCTION

Élan™SC520 MICROCONTROLLER

The Élan™SC520 microcontroller is a full-featured microcontroller developed for the general embedded market. The ÉlanSC520 microcontroller combines a 32-bit, low-voltage Am5$_x$86® CPU with a complete set of integrated peripherals suitable for both real-time and PC/AT-compatible embedded applications.

PURPOSE OF THIS MANUAL

This manual describes the technical features and programming interface of the ÉlanSC520 microcontroller.

Intended Audience

The *Élan™SC520 Microcontroller User's Manual*, order #22004, is intended for computer software and hardware engineers and system architects who are designing or are considering designing systems based on the ÉlanSC520 microcontroller.

Overview of this Manual

The manual is organized into the following chapters:

- Chapter 1 includes an **architectural overview** of the ÉlanSC520 microcontroller, along with **applications diagrams**.
- Chapter 2 describes the **signals and pins** of the ÉlanSC520 microcontroller. **Logic diagrams** showing defaults and pins with shared signals are also found in this chapter. Detailed pin state information is available in the *Élan™SC520 Microcontroller Data Sheet*.
- Chapter 3 provides an overview of **system initialization** and shows **example configurations**.
- Chapter 4 describes the **system address mapping** on the ÉlanSC520 microcontroller.
- Chapter 5 provides information on **clock generation and control**.
- Chapter 6 describes the **reset sources and states** of the ÉlanSC520 microcontroller.
- Chapter 7 includes an overview of the integrated Am5$_x$86 CPU. For additional information about the CPU, consult the references provided in this chapter.
- Chapter 8 describes the **system arbiter** on the ÉlanSC520 microcontroller, which includes a CPU bus arbiter and a PCI bus arbiter.
- Chapter 9 describes the **PCI bus host bridge** implemented on the ÉlanSC520 microcontroller.
- Chapter 10 describes the **synchronous DRAM (SDRAM) controller**.
- Chapter 11 describes the **SDRAM write buffer and read buffer** with read-ahead feature.
- Chapter 12 describes the **ROM/Flash controller**.

- Chapter 13 describes the programmable **general-purpose (GP) bus interface** included on the ÉlanSC520 microcontroller.
- Chapter 14 describes the **GP bus DMA controller**.
- Chapter 15 describes the **programmable interrupt controller (PIC)**, which includes three interrupt controllers.
- Chapter 16 describes the **programmable interval timer (PIT)**, which includes three timers.
- Chapter 17 describes the three **general-purpose (GP) timers** included on the ÉlanSC520 microcontroller.
- Chapter 18 describes the **software timer** that eases the task of keeping system time.
- Chapter 19 describes the **watchdog timer** used to guard against runaway software.
- Chapter 20 describes the **real-time clock (RTC) and RTC voltage monitor** included on the ÉlanSC520 microcontroller.
- Chapter 21 describes the two **UART serial ports**.
- Chapter 22 describes the **synchronous serial interface (SSI)**.
- Chapter 23 describes the 32 **programmable input/output (PIO) pins** on the ÉlanSC520 microcontroller.
- Chapter 24 is a summary of the **system test features** found on the ÉlanSC520 microcontroller.
- Chapter 25 describes the Joint Test Action Group (JTAG) (IEEE Std. 1149.1-1990) **boundary scan test interface** features of the ÉlanSC520 microcontroller.
- Chapter 26 provides an overview of **AMDebug™ technology** and the board specifications necessary to utilize this capability, which is supported by third-party FusionE86 vendors.

RELATED DOCUMENTS

The following documents contain additional information that will be useful in designing an embedded application based on the ÉlanSC520 microcontroller.

AMD Documentation

In addition to this manual, the documentation set for the ÉlanSC520 microcontroller includes the following documents:

- *Élan™SC520 Microcontroller Register Set Manual*, order #22005, fully describes all the configuration registers required to program the microcontroller.
- *Élan™SC520 Microcontroller Data Sheet*, order #22003, includes complete pin lists, pin state tables, timing and thermal characteristics, and package dimensions for the ÉlanSC520 microcontroller.

Other information of interest:

- The *Am486® Microprocessor Software User's Manual*, order #18497, includes the complete instruction set for the integrated Am5$_X$86 CPU.
- *Am5$_X$86® Microprocessor Family Data Sheet*, order #19751
- *Am486® DX/DX2 Microprocessor Hardware Reference Manual*, order #17965

- *E86 Family Products and Development Tools CD*, order #21058, provides a single-source multimedia tool for customer evaluation of AMD products, as well as FusionE86 partner tools and technologies that support the E86 family. Technical documentation is included on the CD in PDF format.

To order literature, contact the nearest AMD sales office or call the literature center at one of the numbers listed on the back cover of this manual. In addition, these documents are available in PDF form on the AMD web site. To access the web site, go to www.amd.com and follow the Embedded Processor link for information about the E86 family.

Additional Information

The following non-AMD documents and sources provide additional information that may be of interest to ÉlanSC520 microcontroller users:

- *PCI Local Bus Specification,* Revision 2.2, December 18, 1998, PCI Special Interest Group, 800-433-5177 (US), 503-693-6360 (International), www.pcisig.com.
- *IEEE Std 1149.1-1990 Standard Test Access Port and Boundary-Scan Architecture,* (order #SH16626-NYF), Institute of Electrical and Electronic Engineers, Inc., 800-678-4333, www.ieee.org.
- *PCI System Architecture*, Mindshare, Inc., Reading, MA: Addison-Wesley, 1995, ISBN 0-201-40993-3.
- *ISA System Architecture*, Mindshare, Inc., Reading, MA: Addison-Wesley, 1995, ISBN 0-201-40996-8.
- *80486 System Architecture*, Mindshare, Inc., Reading, MA: Addison-Wesley, 1995, ISBN 0-201-40994-1.
- *The Indispensable PC Hardware Book*, Hans-Peter Messmer, Wokingham, England: Addison-Wesley, 1995, ISBN 0-201-87697-3.

DOCUMENTATION CONVENTIONS

Table 0-1 lists the documentation conventions used throughout this manual.

Table 0-1 Documentation Notation

Notation	Meaning
Reset Default Values	
Default	Value after a system reset
0	Low
1	Active or High
x	No value is guaranteed
?	Determined by sources external to the ÉlanSC520 microcontroller
Read/Write Attributes	
R	The bit field is read-only. A write to the register at this bit field has no effect. The contents may or may not be changed by hardware.

Table 0-1 Documentation Notation (Continued)

Notation	Meaning
W	The bit field is write-only. Reading this register at this bit field does not return a meaningful value and has no side effects.
R/W	The bit field is read/write. Reading the register at this bit field always returns the last value written. Reads have no side effects.
R/W!	The bit field is read/write with conditions. The "!" indicates that there are side effects to using this bit. For example, reading a bit or register might not always return the last value written. Note that both reads and writes can have side effects. If you see a "!", be sure to read the bit description and programming notes.
RSV	The bit field is reserved for internal test/debug or future expansion. This bit field should be written to 0 for normal system operation. This bit field always returns 0 when read.
RSV!	The bit field is reserved for compatibility purposes. For example, the bit field might be ignored during writes to maintain software compatibility. If you see a "!", be sure to read the bit description and programming notes.
Reference Notation	
MMCR offset 00h	ÉlanSC520 microcontroller Memory-Mapped Configuration Region (MMCR) offset register 00h
PCI index 00h	PCI indexed register 00h
Port 00h	Direct-mapped I/O register 00h
RTC index 00h	RTC and configuration RAM indexed register 00h
Pin Naming	
{ }	Pin function during hardware reset
[ ]	Alternative pin function selected by software configuration
$\overline{\text{ROMCS1}}$	An overbar indicates that the signal assumes the logic Low state when asserted.
GPRESET	The absence of an overbar indicates that the signal assumes the logic High state when asserted.
$\overline{\text{ads}}$, hold	A signal name in all lowercase indicates an internal signal.
$\overline{\text{ROMCS2}}$–$\overline{\text{ROMCS1}}$	Two ROM chip select signals
$\overline{\text{ROMCSx}}$	Any of the two ROM chip select signals
Numbers	
b	Binary number
d	Decimal number Decimal is the default radix

Table 0-1 Documentation Notation (Continued)

Notation	Meaning
h	Hexadecimal number
x in register address	Any of several legal values; e.g., using 0xF8h for the UART Transmit Holding register is either 02F8h or 03F8h, depending on the UART
[X–Y]	The bit field that consists of bits X through Y. Example: The SB_ADDR[23–16] bit field.
33 MHz	Refers to the system clock frequency being used. This can be either 33.000 MHz or 33.333 MHz. See the *Élan™SC520 Microcontroller User's Manual* for more information about clock generation.
General	
field	Bit field in a register (one or more consecutive and related bits)
can	It is possible to perform an action if properly configured
will	A certain action is going to occur
Set the ENB bit.	Write the ENB bit to 1. ***Note:*** *The bit referred to is either in the register being described, or the register is referred to explicitly in the surrounding text.*
Clear the ENB bit.	Change the ENB bit to 0. Usually a bit is cleared by writing a 0 to it; however, some bits are cleared by writing a 1.
Reset the ENB bit.	Context-sensitive. Can refer either to resetting the bit to its default value or to clearing the bit.

CHAPTER

1 ARCHITECTURAL OVERVIEW

1.1 Élan™SC520 MICROCONTROLLER

The Élan™SC520 microcontroller is a full-featured microcontroller developed for the general embedded market. The ÉlanSC520 microcontroller combines a 32-bit, low-voltage Am5$_x$86 CPU with a complete set of integrated peripherals suitable for both real-time and PC/AT-compatible embedded applications.

An integrated PCI host bridge, SDRAM controller, enhanced PC/AT-compatible peripherals, and advanced debugging features provide the system designer with a wide range of on-chip resources, allowing support for legacy devices as well as new devices available in the current PC marketplace.

Designed for medium- to high-performance applications in the telecommunications, data communications, and information appliance markets, the ÉlanSC520 microcontroller is particularly well suited for applications requiring high throughput combined with low latency.

1.1.1 Distinctive Characteristics

- Industry-standard Am5$_x$86® CPU with floating point unit (FPU) and 16-Kbyte write-back cache
  - 100-MHz and 133-MHz operating frequencies
  - Low-voltage operation (core V_{CC} = 2.5 V)
  - 5-V tolerant I/O (3.3-V output levels)
- E86™ family of x86 embedded processors
  - Part of a software-compatible family of microprocessors and microcontrollers well supported by a wide variety of development tools
- Integrated PCI host bridge controller leverages standard peripherals and software
  - 33 MHz, 32-bit PCI bus Revision 2.2-compliant
  - High-throughput 132-Mbyte/s peak transfer
  - Supports up to five external PCI masters
  - Integrated write-posting and read-buffering for high-throughput applications
- Synchronous DRAM (SDRAM) controller
  - Supports 16-, 64-, 128-, and 256-Mbit SDRAM.
  - Supports 4 banks for a total of 256 Mbytes.
  - Error Correction Code provides system reliability.
  - Buffers improve read and write performance.
- AMDebug™ technology offers a low-cost solution for the advanced debugging capabilities required by embedded designers.
  - Allows instruction tracing during execution from the Am5$_x$86 CPU's internal cache
  - Uses an enhanced JTAG port for low-cost debugging

  - Parallel debug port for high-speed data exchange during in-circuit emulation
- General-purpose (GP) bus with programmable timing for 8- and 16-bit devices provides good performance at very low cost.
- ROM/Flash controller for 8-, 16-, and 32-bit devices
- Enhanced PC/AT-compatible peripherals provide improved performance.
  - Enhanced programmable interrupt controller (PIC) prioritizes 22 interrupt levels (up to 15 external sources) with flexible routing.
  - Enhanced DMA controller includes double buffer chaining, extended address and transfer counts, and flexible channel routing.
  - Two 16550-compatible UARTs operate at baud rates up to 1.15 Mbit/s with optional DMA interface.
- Standard PC/AT-compatible peripherals
  - Programmable interval timer (PIT)
  - Real-time clock (RTC) with battery backup capability and 114 bytes of RAM
- Additional integrated peripherals
  - Three general-purpose 16-bit timers provide flexible cascading for 32-bit operation.
  - Watchdog timer guards against runaway software.
  - Software timer
  - Synchronous serial interface (SSI) offers full-duplex or half-duplex operation.
  - Flexible address decoding for programmable memory and I/O mapping and system addressing configuration
- 32 programmable input/output (PIO) pins
- Native support for pSOS, QNX, RTXC, VxWorks, and Windows® CE operating systems
- Industry-standard BIOS support

1.2 BLOCK DIAGRAM

Figure 1-1 on page 1-3 illustrates the integrated Am5$_X$86 CPU, bus structure, and on-chip peripherals of the ÉlanSC520 microcontroller. Three primary interfaces are provided:

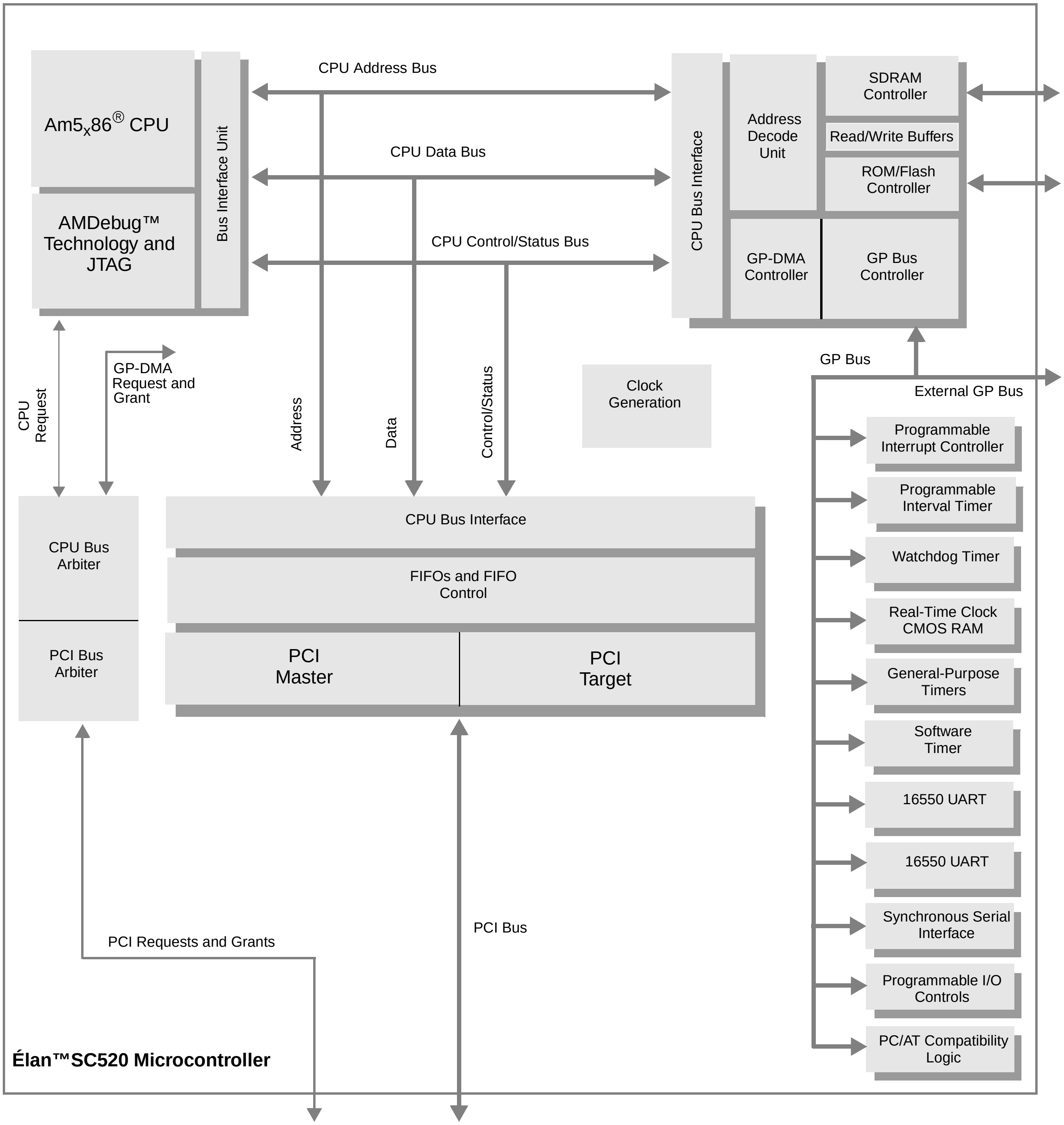
- A high-performance, 66-MHz 32-bit synchronous DRAM (SDRAM) interface of up to 256 Mbytes is used for Am5$_X$86 CPU code execution, as well as buffer storage of external PCI bus masters and GP bus DMA initiators. A high-performance ROM/Flash interface can also be connected to the SDRAM interface.
- An industry-standard, 32-bit PCI bus is provided for high bandwidth I/O peripherals such as local area network controllers, synchronous communications controllers, and disk storage controllers.
- A simple 8/16-bit, 33-MHz general-purpose bus (GP bus) provides a glueless connection to lower bandwidth peripherals, and NVRAM, SRAM, ROM, or custom ASICs; supports dynamic bus sizing and compatibility with many common ISA devices.

These three buses listed above are provided in all operating modes of the ÉlanSC520 microcontroller.

In addition to these three primary interfaces, the ÉlanSC520 microcontroller also contains internal oscillator circuitry and phase locked loop (PLL) circuitry, requiring only two simple crystals for virtually all system clock generation.

Diagrams showing how the ÉlanSC520 microcontroller can be used in various system designs are included in “Applications” on page 1-8.

Figure 1-1 Élan™SC520 Microcontroller Block Diagram



1.3 ARCHITECTURAL OVERVIEW

The ÉlanSC520 microcontroller was designed to provide:

- A balanced mix of high performance and low-cost interface mechanisms
- A high-performance, industry-standard 32-bit PCI bus
- Glueless interfacing to many 8- and 16-bit I/O peripherals and an 8- and 16-bit bus with programmable timing
- A cost-effective system architecture that meets a wide range of performance criteria while retaining the lower cost of a 32-bit system
- A high degree of leverage from present day hardware and software technologies

1.3.1 Industry-Standard x86 Architecture (Chapter 7)

The Am5$_X$86 CPU in the ÉlanSC520 microcontroller utilizes the industry-standard x86 microprocessor instruction set that enables compatibility across a variety of performance levels from the 16-bit Am186™ processors to the high-end AMD Athlon™ processor. Software written for the x86 architecture family is compatible with the ÉlanSC520 microcontroller.

Other benefits of the Am5$_X$86 CPU include:

- Improved time-to-market and easy software migration
- Existing availability of multiple operating systems that directly support the x86 architecture. Whether the application requires a real-time operating system (RTOS) or one of the popular Microsoft® operating systems, the ÉlanSC520 microcontroller provides consistent compatibility with many off-the-shelf operating systems.
- Multiple sources of field-proven development tools
- Integrated floating point unit (FPU) (compliant with ANSI/IEEE 754 standard)
- 16-KByte unified cache configurable for either write-back or write-through cache mode

The Am5$_X$86 CPU is described in Chapter 7.

1.3.2 AMDebug™ Technology for Advanced Debugging (Chapter 26)

The ÉlanSC520 microcontroller provides support for low-cost, full-featured, in-circuit emulation capability. This in-circuit emulation support was developed at AMD specifically to enable users to test and debug their software earlier in the design cycle. Utilizing this capability, the software can be more extensively exercised, and at full execution speeds. It also allows tracing during execution from the Am5$_X$86 CPU's internal cache.

AMDebug support provides the product design team with two different communication paths on the ÉlanSC520 microcontroller, each of which is supported by powerful debug tools from third-party vendors in AMD's FusionE86 program.

- Serial AMDebug technology uses a serial connection based on an enhanced JTAG protocol and an inexpensive 12-pin connector that can be placed on each board design. This low-cost solution satisfies the requirement of a large number of software developers.
- Parallel AMDebug technology uses a parallel debug port to exchange commands and data between the ÉlanSC520 microcontroller and the host. The higher pin count requires that the extra signal pins be provided on a special bond-out package of the ÉlanSC520 microcontroller, which is only made available to tool developers, such as in-circuit emulator manufacturers. The parallel AMDebug port greatly simplifies the task of supporting high speed data exchange.

1.3.3 Industry-Standard PCI Bus Interface (Chapter 9)

The ÉlanSC520 microcontroller provides a 33-MHz, 32-bit PCI bus Revision 2.2-compliant host bridge interface, including integrated write-posting and read-buffering capabilities suitable for high-throughput applications. The PCI host bridge leverages standard peripherals and software. It also provides:

- High throughput (132 Mbytes/s peak transfer rate)
- Deep buffering and support for burst transactions from PCI bus masters to SDRAM
- Flexible arbitration mechanism
- Support for up to five external PCI masters

1.3.4 High-Performance SDRAM Controller (Chapter 10)

The ÉlanSC520 microcontroller provides an integrated SDRAM controller that supports popular industry-standard synchronous DRAMs (SDRAM).

- The SDRAM controller interfaces with SDRAM chips as well as with most standard DIMMs to enable use of standard off-the-shelf memory components.
- The SDRAM controller supports programmable timing options and provides the required external clock.
- Up to four 32-bit banks of SDRAM are supported with a maximum capacity of 256 Mbytes.
- An important reliability-enhancing Error Correction Code (ECC) feature is built into the SDRAM controller. The resultant increase in the memory content reliability enables the ÉlanSC520 microcontroller to be effectively utilized in applications that require more reliable operation, such as communications environments.
- The SDRAM controller contains a write buffer and read ahead buffer subsystem that improves both write and read performance.
- SDRAM refresh options allow the SDRAM contents to be maintained during reset.

1.3.5 ROM/Flash Controller (Chapter 12)

The ÉlanSC520 microcontroller provides an integrated ROM controller for glueless interfacing to ROM and Flash devices. The ÉlanSC520 microcontroller supports two types of interfaces to such devices—a simple interface via the GP bus for 8- and 16-bit devices, and an interface to the SDRAM memory data bus for higher performance 8-, 16-, and 32-bit devices.

The ROM/Flash controller:

- Reduces system cost by gluelessly interfacing static memory with up to three ROM/Flash chip selects
- Supports execute-in-place (XIP) operating systems for applications that require executing out of ROM or Flash memory instead of DRAM
- Supports high-performance page-mode devices

1.3.6 Flexible Address-Mapping (Chapter 4)

In addition to the memory management unit (MMU) within the Am5$_X$86 CPU core, the ÉlanSC520 microcontroller provides 16 Programmable Address Region (PAR) registers that enable flexible placement of memory (SDRAM, ROM, Flash, SRAM, etc.) and peripherals into the two address spaces of the Am5$_X$86 CPU (memory address space and I/O address space). The PAR hardware allows designers to flexibly configure both address spaces and place memory and/or external peripherals, as required by the application. The internal memory-mapped configuration registers space can also be remapped to accommodate system requirements. PAR registers also allow control of important attributes, such as cacheability, write protection, and code execution protection for memory resources.

1.3.7 General-Purpose (GP) Bus Interface (Chapter 13)

The ÉlanSC520 microcontroller includes a simple general-purpose (GP) bus that provides programmable bus timing and allows the connection of 8/16-bit peripheral devices and memory to the ÉlanSC520 microcontroller. The GP bus operates at 33 MHz, which offers good performance at a very low interface cost.

The ÉlanSC520 microcontroller provides up to eight chip selects for external GP bus devices such as off-the-shelf I/O peripherals, custom ASICs, and SRAM or NVRAM. The GP bus interface supports programmable timing and dynamic bus width and cycle stretching to accommodate a wide variety of standard peripherals, such as UARTs, 10-Mbit LAN controller chips and serial communications controllers. Up to four external DMA channels provide fly-by DMA transfers between peripheral devices on the GP bus and system SDRAM.

Internally, the GP bus is used to provide a full complement of integrated peripherals, such as a DMA controller, programmable interrupt controller, timers, and UARTs, as described in “Integrated Peripherals” on page 1-7. These internal peripherals are designed to operate at the full clock rate of the GP bus. The internal peripherals can also be configured to operate in PC/AT-compatible configuration, but are generally not restricted to this configuration.

The ÉlanSC520 microcontroller provides a way to view accesses to the internal peripherals on the external GP bus for debugging purposes.

1.3.8 Clock Generation (Chapter 5)

The ÉlanSC520 microcontroller offers user-configurable CPU core clock speed operation at 100 or 133 MHz for different power/performance points depending on the application.

Not all ÉlanSC520 microcontroller devices support all CPU clock rates. The maximum supported clock rate for a device is indicated by the part number printed on the package. The clocking circuitry can be programmed to run the device at higher than the rated speeds. However, if an ÉlanSC520 microcontroller is programmed to run at a higher clock speed than that for which it is rated, then erroneous operation can result, and physical damage to the device may occur.

The ÉlanSC520 microcontroller includes on-chip oscillators and PLLs, as well as most of the required PLL loop filter components. The ÉlanSC520 microcontroller requires two standard crystals, one for 32.768 kHz and one for 33 MHz. All the clocks required inside the ÉlanSC520 microcontroller are generated from these crystals. The ÉlanSC520 microcontroller also supplies the clocks for the SDRAM and PCI bus; however, external clock buffering may be required in some systems.

Note: *The ÉlanSC520 microcontroller supports either a 33.000-MHz or 33.333-MHz crystal. In this document, the generic term “33 MHz” refers to the system clock derived from whichever 33-MHz crystal frequency is being used in the system.*

1.3.9 Integrated Peripherals

The ÉlanSC520 microcontroller is a highly integrated single-chip CPU with a complete set of integrated peripherals that are a superset of common PC/AT peripherals, plus a set of memory-mapped peripherals that enhance its usability in various applications.

- A programmable interrupt controller (PIC) (see Chapter 15) that provides the capability to prioritize 22 interrupt levels, up to 15 of these being external sources. The PIC can be programmed to operate in PC/AT-compatible mode, but also contains extended features, including support for more sources and flexible routing that allows any interrupt request to be steered to any PIC input. Interrupt requests can be programmed to generate either non-maskable interrupt (NMI) or maskable interrupt requests.
- An integrated DMA controller (see Chapter 14) is included for transferring data between SDRAM and GP bus peripherals. The GP-DMA controller operates in single-cycle (fly-by) mode for more efficient transfers. The GP-DMA controller can be programmed for PC/AT compatibility, but also contains enhanced features:
  - A double buffer-chaining mode provides a more efficient software interface.
  - Extended address and transfer counts
  - Flexible routing of DMA channels
- Three general-purpose 16-bit timers (see Chapter 17) that provide flexible cascading for extension to 32-bit operation. These timers provide the ability to configure down to the resolution of four clock periods where the clock period is the 33-MHz clock. Timer input and output pins provide the ability to interface with off-chip hardware.
- A standard PC/AT-compatible programmable interval timer (PIT) (see Chapter 16) that consists of three 16-bit timers.
- A software timer (see Chapter 18) that eases the task of keeping system time. It provides 1-µs resolution and can also be used for performance monitoring.
- A watchdog timer (see Chapter 19) to guard against runaway software.
- A real-time clock (RTC) with battery backup capability (see Chapter 20). The RTC also provides 114 bytes of battery-backed RAM for storage of configuration parameters.
- Two integrated 16550-compatible UARTs (see Chapter 21) that provide full handshaking capability with eight pins each. Enhancements enable the UARTs to operate at baud rates up to 1.152 Mbits/s. The UARTs can be configured to use the integrated GP bus DMA controller to transfer data between the serial ports and SDRAM.
- A synchronous serial interface (SSI) that is compatible with SCP, SPI, and Microwire slave devices (see Chapter 22). The SSI interface can be configured for either full-duplex or half-duplex operation using a 4-wire or 3-wire interface.
- 32 programmable I/O pins are provided (see Chapter 23). These pins are multiplexed with other peripherals and interface functions.
- The ÉlanSC520 microcontroller also provides PC/AT-compatible functions for control of the a20 gate and the soft CPU reset (Ports 0060h, 0064h, 0092h).

1.3.10 JTAG Boundary Scan Test Interface (Chapter 25)

The ÉlanSC520 microcontroller provides a full JTAG test port that is compliant with IEEE Std 1149.1-1990 for use during board testing.

1.3.11 System Testing and Debugging Features (Chapter 24)

To facilitate debugging, the ÉlanSC520 microcontroller provides observability of many portions of its internal operation, including:

- A three-pin interface that can be used in either system test mode or write buffer test mode, to aid in determining internal bus initiators of SDRAM cycles, and determining when SDRAM data is valid on the interface. An additional mode provides observability of integrated peripheral accesses.
- A nonconcurrent arbitration mode to reduce debug complexity when PCI bus masters and GP bus DMA initiators are also accessing SDRAM.
- CPU cache control and dynamic core clock speed control under program control.
- Ability to disable write posting and read prefetching in the SDRAM controller to simplify tracing of SDRAM cycles.
- Notification of memory write protection and non-executable memory region violations.

1.4 APPLICATIONS

The figures on the following pages show the ÉlanSC520 microcontroller as it might be used in several reference design applications in the data communications, information appliances, and telecommunication markets.

1.4.1 Smart Residential Gateway

Figure 1-2 on page 1-10 shows an ÉlanSC520 microcontroller-based Smart Resident Gateway (SRG), which is a router for a home network between the wide area network (WAN) (the internet) and a local area network (LAN) (an intranet of computers and information appliances in the home). The SRG provides firewall protection of the LAN from unauthorized access through the internet. A common internet access medium is shared by all users on the LAN.

A variety of connections are possible for both the WAN and the LAN. For example, the WAN connection can be a V.90 modem, cable modem, ISDN, ADSL, or Ethernet.

The LAN connection can be:

- HomePNA—Home Phoneline Networking Alliance, an alliance with a widely endorsed home networking specification
- Bluetooth—a computing and telecommunications industry specification that describes how computing devices can easily interconnect with each other and with home and business phones and computers using a short-range wireless connection)
- Home RF—a standard competing with Bluetooth for the interconnection of computing devices in a LAN using radio frequency
- Ethernet—local area network technology
- Power line—a LAN using the AC power distribution network in a home or business to interconnect devices. Digital information is transmitted on a high-frequency carrier signal on top of the AC power.

1.4.2 Thin Client

Figure 1-3 on page 1-11 shows an ÉlanSC520 microcontroller-based “thin client,” which is the modern replacement for the traditional terminal in a remote computing paradigm. Application programs run remotely on a server, and data is warehoused on centrally managed disks at the “server farm.” An efficient communications protocol transmits keyboard and mouse commands upstream and transmits video BIOS calls downstream. The thin client renders and displays the graphics for the user.

The thin client is typically connected to an Ethernet LAN, although a remote location can connect to a server via a WAN connection such as a modem. A minimum speed of 24 kbaud is required for the communication protocol, unless the application is graphics-intensive, in which case a faster connection is required.

1.4.3 Digital Set Top Box

Figure 1-4 on page 1-12 shows an ÉlanSC520 microcontroller-based digital set top box (DSTB), which is a consumer client device that uses a television set as the display. Common applications for the DSTB are internet access, e-mail, and streaming audio and video content.

The minimal system includes a connection to the WAN via a modem, ADSL, or cable modem; an output to a TV; and an InfraRed (IR) link to a remote control or wireless keyboard. Expanded systems include DVD drives and MPEG2 decoders to deliver digital video content. A hard drive may be employed to store video data for future replay. Keyboard, mouse, printer, or a video camera are options that can be included.

1.4.4 Telephone Line Concentrator

Figure 1-5 on page 1-13 shows an ÉlanSC520 microcontroller-based telephone line concentrator located in the neighborhood that converts multiple analog subscriber loops into a high-speed digitally multiplexed line for connection to the central office switching network.

Figure 1-2 Élan™SC520 Microcontroller-Based Smart Residential Gateway Reference Design

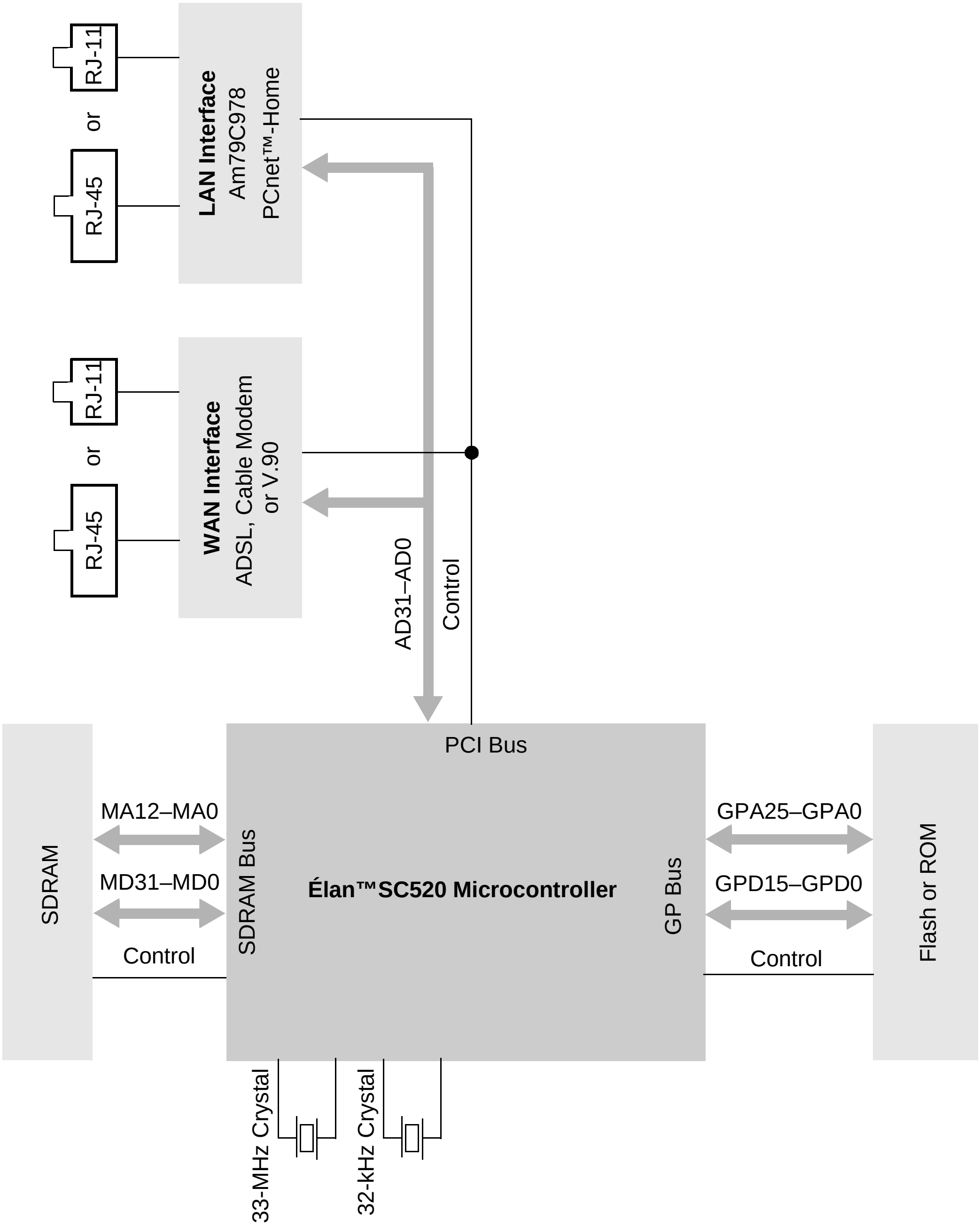


Figure 1-3 Élan™SC520 Microcontroller-Based Thin Client Reference Design

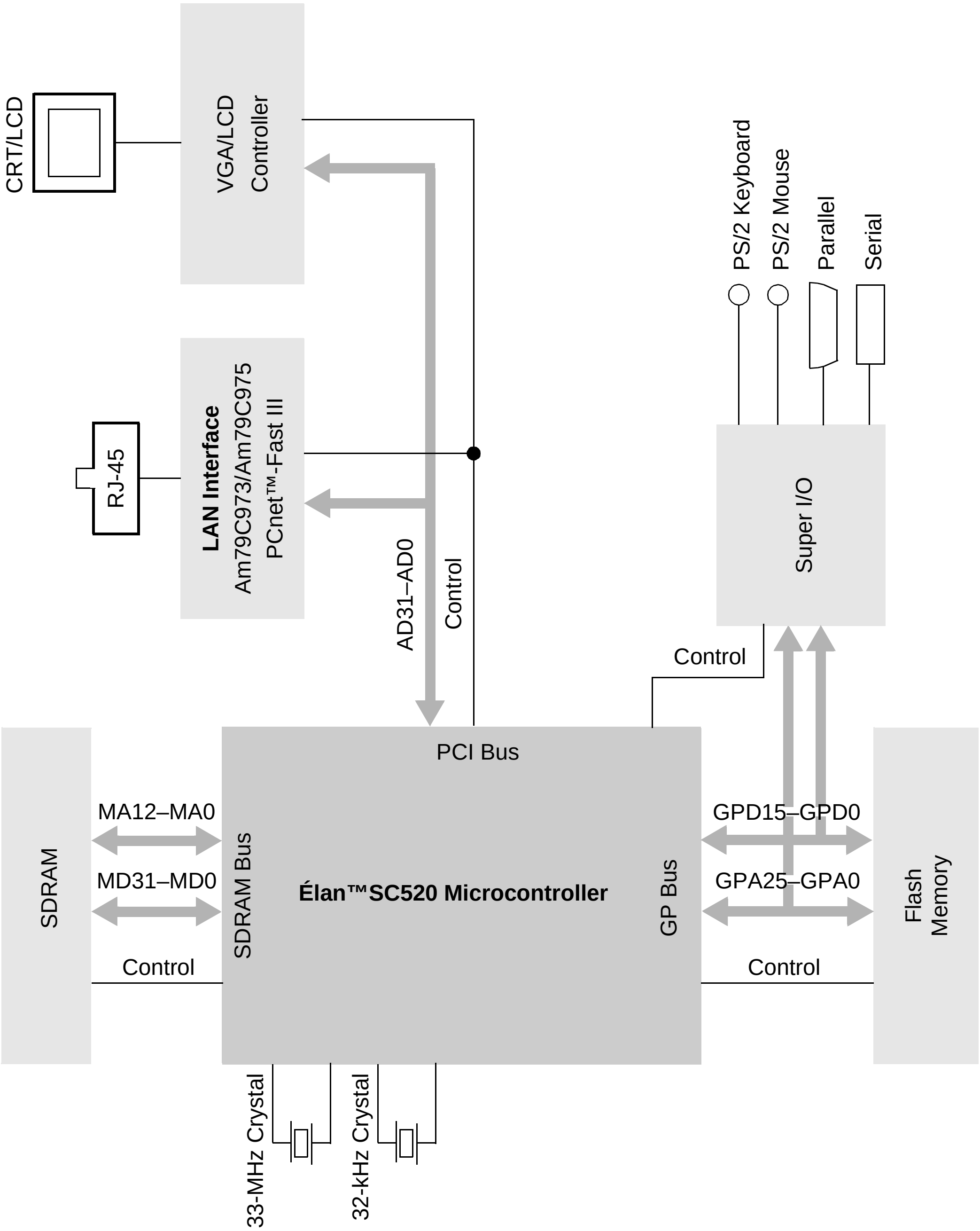


Figure 1-4 Élan™SC520 Microcontroller-Based Digital Set Top Box Reference Design

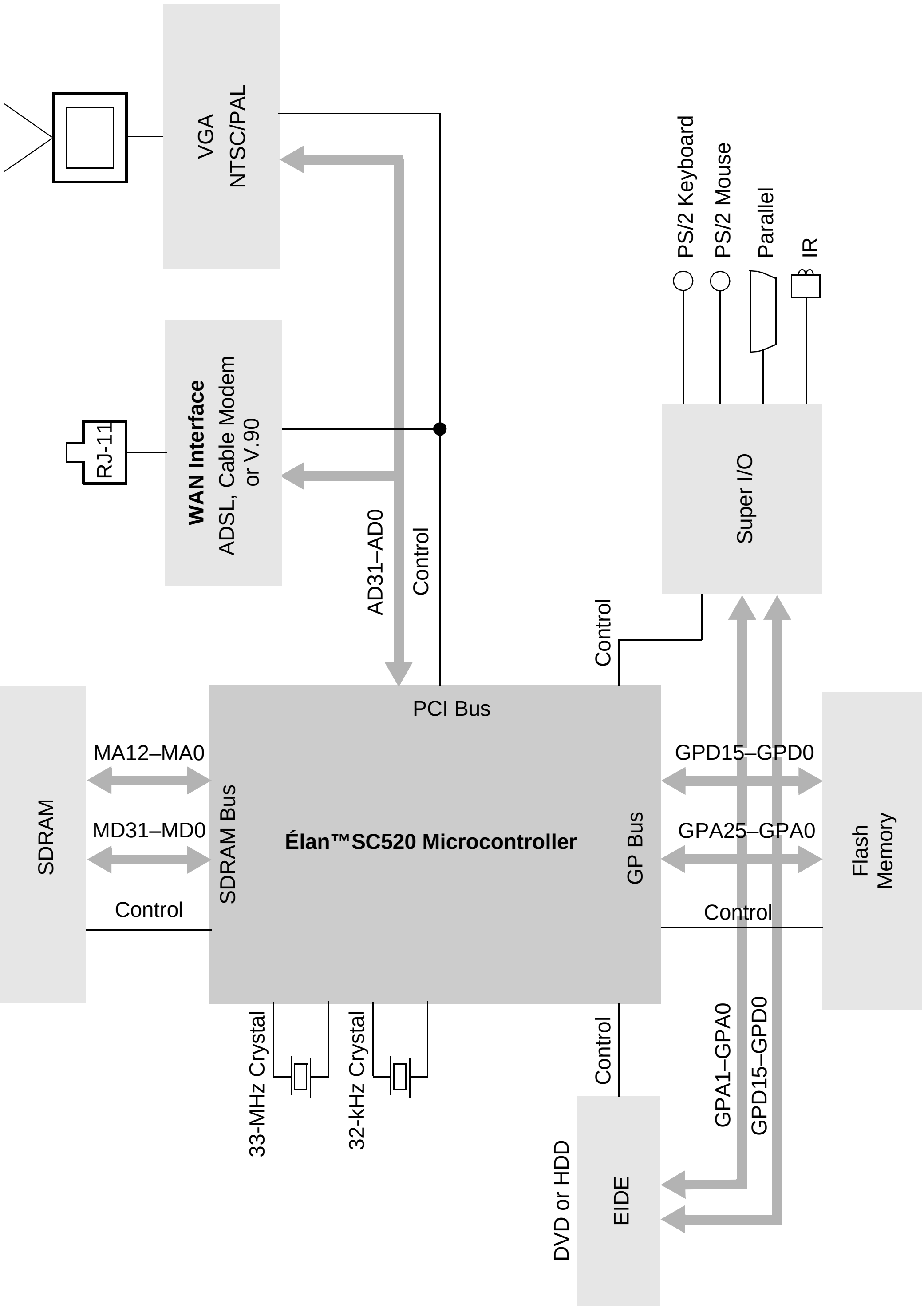
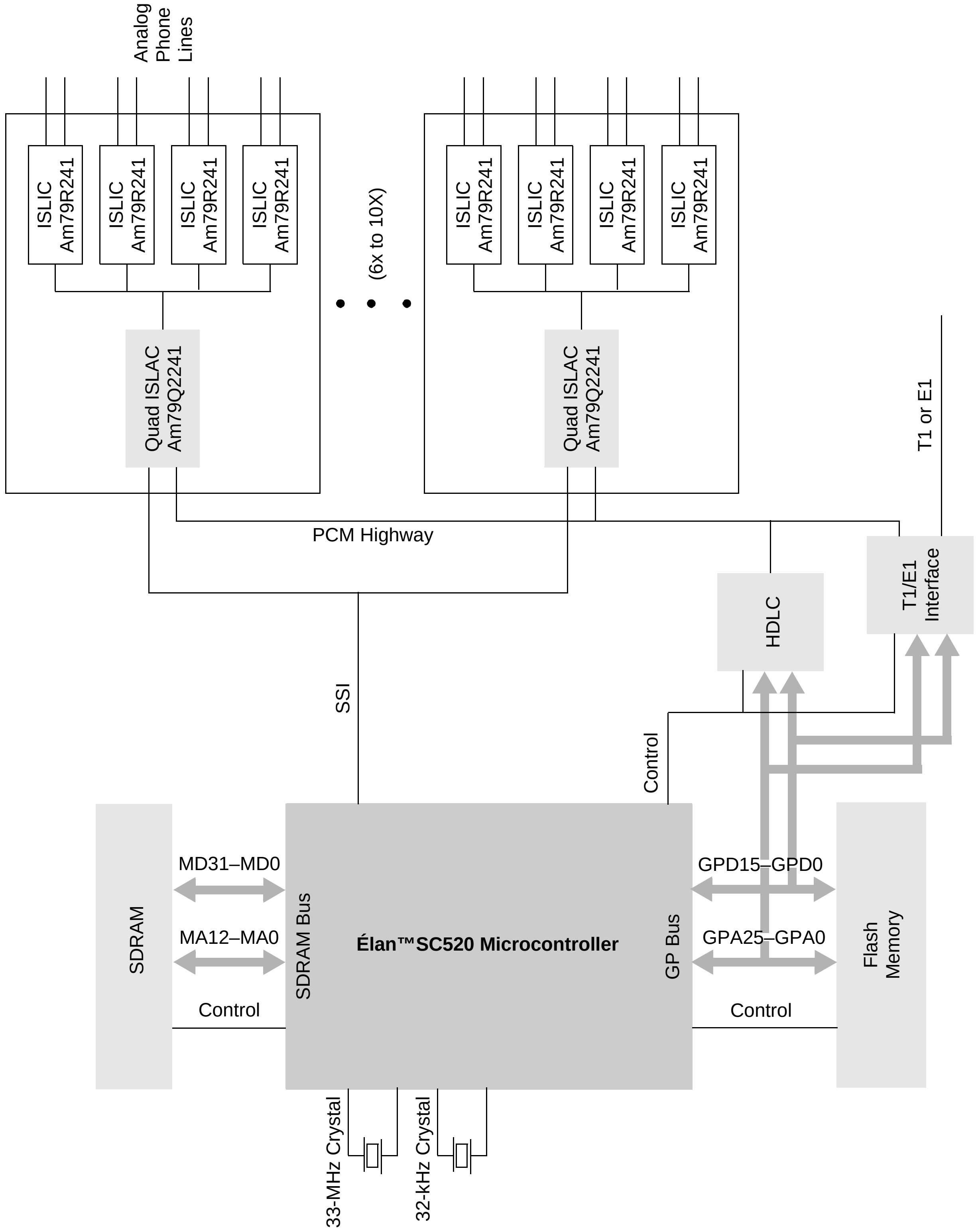


Figure 1-5 Élan™SC520 Microcontroller-Based Telephone Line Concentrator Reference Design



CHAPTER

2 PIN INFORMATION

2.1 OVERVIEW

The ÉlanSC520 microcontroller contains 258 signal pins plus power and ground signals. A minimal number of signals are shared with others.

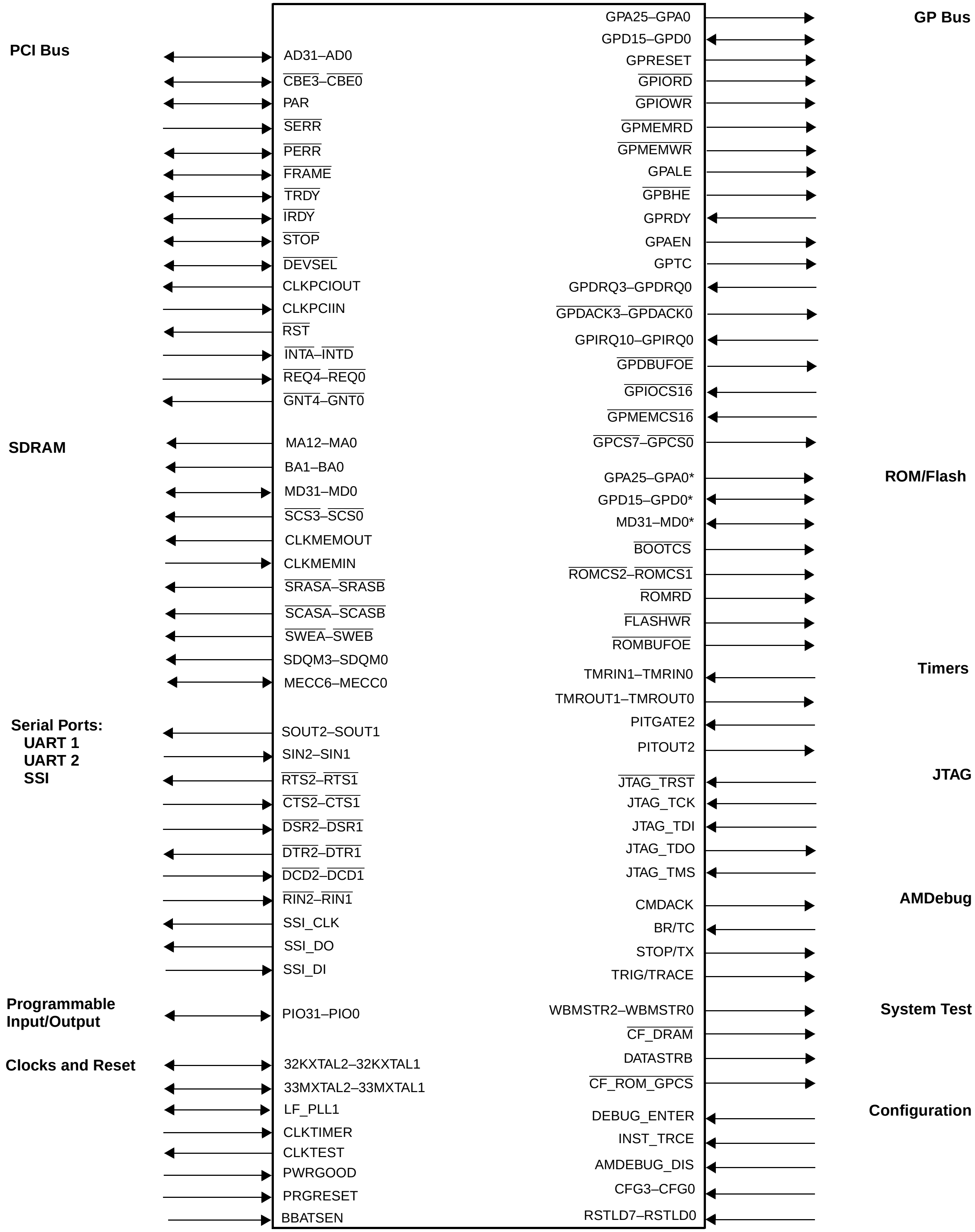
The signals are organized alphabetically within the following functional groups:

- Synchronous DRAM controller (page 2-5)
- ROM/Flash controller (page 2-6)
- PCI bus (page 2-6)
- General-purpose (GP) bus (page 2-7)
- Serial ports (page 2-9)
- Timers (page 2-10)
- Clocks and reset (page 2-10)
- Chip selects (page 2-11)
- Programmable I/O (PIO) (page 2-11)
- JTAG boundary scan test interface (page 2-12)
- AMDebug interface (page 2-12)
- System test (page 2-12)
- Configuration (page 2-13)
- Power (page 2-14)

2.2 LOGIC SYMBOLS

Figure 2-1 shows a logical symbol of the device, with pins grouped by function or interface. Figure 2-2 shows a logical symbol with pins grouped by default function. Figure 2-2 also shows pin multiplexing on the ÉlanSC520 microcontroller.

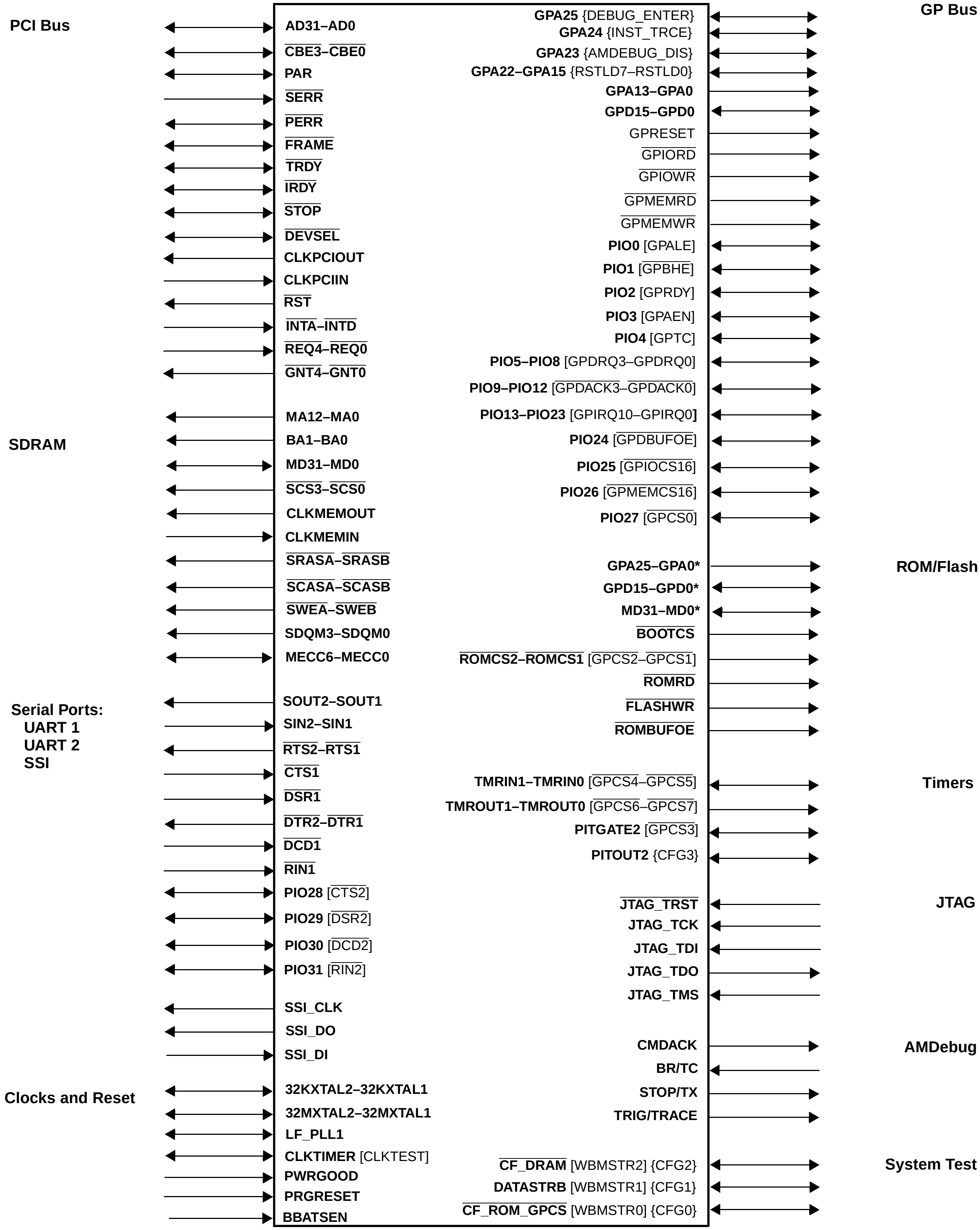
Figure 2-1 Logic Diagram by Interface[1]



Notes:

1. Pins noted with asterisks are duplicated in this diagram to clarify which signals are used for each interface.

Figure 2-2 Logic Diagram by Default Pin Function[1]



Notes:

1. Pin names in ***bold*** *indicate the default pin function. Brackets, [ ], indicate alternate, multiplexed functions. Braces, { }, indicate pinstrap pins. Pins noted with asterisks are duplicated in this diagram to clarify which signals are used for each interface.*

2.3 SIGNAL DESCRIPTIONS

Table 2-1 describes the terms used in the signal description table. In general, the brackets, [ ], indicate alternate, multiplexed functions, and braces, { }, indicate reset configuration pins (pinstraps). The line over a pin name indicates an active Low signal. The word pin refers to the physical wire; the word signal refers to the electrical signal that flows through it.

Table 2-2, “Signal Descriptions” on page 5 contains a description of the ÉlanSC520 microcontroller signals. The descriptions in Table 2-2 are organized by functional group. Table 2-2 describes the signals that are available for each interface and which signals are shared with others. Signal sharing is also shown in Figure 2-2.

Detailed information on pin state, including maximum load values, power-on reset default function, reset state, power-on reset default operation, hold state, and voltage, is available in the *Élan™SC520 Microcontroller Data Sheet*, order #22003. Connection and package diagrams, as well as pin number assignments, are also included in that document.

Table 2-1 Signal Descriptions Table Definitions

Term	Definition
General Terms	
[ ]	Indicates the pin alternate function; a pin defaults to the signal named without the brackets.
{ }	Indicates the reset configuration pin (pinstrap).
pin	Refers to the physical wire.
signal	Refers to the electrical signal that flows across a pin.
$\overline{\text{SIGNAL}}$	A line over a signal name indicates that the signal is active Low; a signal name without a line is active High.
Signal Types	
Analog	Analog voltage
B	Bidirectional
H	High
I	Input
LS	Programmable to hold last state of pin
O	Totem pole output
O/TS	Totem pole output/three-state output
OD	Open-drain output
OD-O	Open-drain output or totem pole output
Osc	Oscillator
PD	Internal pulldown resistor (~100–150 kΩ)
Power	Power pins
PU	Internal pullup resistor (~100–150 kΩ)
STI	Schmitt trigger input
STI-OD	Schmitt trigger input or open-drain output
TS	Three-state output

Table 2-2 Signal Descriptions

Signal	Multiplexed Signal	Type	Description
Synchronous DRAM Controller			
BA1–BA0	—	O	**Bank Address** is the SDRAM bank address bus.
CLKMEMIN	—	I	**SDRAM Clock Input** is the SDRAM clock return signal used to minimize skew between the internal SDRAM clock and the CLKMEMOUT signal provided to the SDRAM devices. This signal compensates for buffer and load delays introduced by the board design.
CLKMEMOUT	—	O	**SDRAM Clock Output** is the 66-MHz clock that provides clock signalling for the synchronous DRAM devices. This clock may require an external Low skew buffer for system implementations that result in heavy loading on the SDRAM clock signal.
MA12–MA0	—	O	**SDRAM Address** is the SDRAM multiplexed address bus.
MD31–MD0	—	B	**SDRAM Data Bus** inputs data during SDRAM read cycles and outputs data during SDRAM write cycles.
MECC6–MECC0	—	B	**Memory Error Correction Code** contains the ECC checksum (syndrome) bits used to validate and correct data errors.
$\overline{\text{SCASA}}$–$\overline{\text{SCASB}}$	—	O	**Column Address Strobes** are used in combination with the $\overline{\text{SRASA}}$–$\overline{\text{SRASB}}$ and $\overline{\text{SWEA}}$–$\overline{\text{SWEB}}$ to encode the SDRAM command type. $\overline{\text{SCASA}}$ and $\overline{\text{SCASB}}$ are the same signal provided on two different pins to reduce the total load connected to $\overline{\text{CAS}}$. Suggested system connection: $\overline{\text{SCASA}}$ for SDRAM banks 0 and 1 $\overline{\text{SCASB}}$ for SDRAM banks 2 and 3
$\overline{\text{SCS3}}$–$\overline{\text{SCS0}}$	—	O	**SDRAM Chip Selects** are the SDRAM chip-select outputs. These signals are asserted to select a bank of SDRAM devices. The chip-select signals enable the SDRAM devices to decode the commands asserted via $\overline{\text{SRASA}}$–$\overline{\text{SRASB}}$, $\overline{\text{SCASA}}$–$\overline{\text{SCASB}}$, and $\overline{\text{SWEA}}$–$\overline{\text{SWEB}}$.
SDQM3–SDQM0	—	O	**Data Input/Output Masks** make SDRAM data output high-impedance and blocks data input on SDRAM while active. Each of the four SDQM3–SDQM0 signals is associated with one byte of four throughout the array. Each SDQMx signal provides an input mask signal for write accesses and an output enable signal for read accesses.
$\overline{\text{SRASA}}$–$\overline{\text{SRASB}}$	—	O	**Row Address Strobes** are used in combination with the $\overline{\text{SCASA}}$–$\overline{\text{SCASB}}$ and $\overline{\text{SWEA}}$–$\overline{\text{SWEB}}$ to encode the SDRAM command type. $\overline{\text{SRASA}}$ and $\overline{\text{SRASB}}$ are the same signal provided on two different pins to reduce the total load connected to $\overline{\text{RAS}}$. Suggested system connection: $\overline{\text{SRASA}}$ for SDRAM banks 0 and 1 $\overline{\text{SRASB}}$ for SDRAM banks 2 and 3
$\overline{\text{SWEA}}$–$\overline{\text{SWEB}}$	—	O	**SDRAM Memory Write Enables** are used in combination with the $\overline{\text{SRASA}}$–$\overline{\text{SRASB}}$ and $\overline{\text{SCASA}}$–$\overline{\text{SCASB}}$ to encode the SDRAM command type. $\overline{\text{SWEA}}$ and $\overline{\text{SWEB}}$ are the same signal provided on two different pins to reduce the total load connected to WE. Suggested system connection: $\overline{\text{SWEA}}$ for SDRAM banks 0 and 1 $\overline{\text{SWEB}}$ for SDRAM banks 2 and 3

Table 2-2 Signal Descriptions (Continued)

Signal	Multiplexed Signal	Type	Description
ROM/Flash Controller			
$\overline{\text{BOOTCS}}$	—	O	**ROM/Flash Boot Chip Select** is an active Low output that provides the chip select for the startup ROM and/or the ROM/Flash array (BIOS, HAL, O/S, etc.). The $\overline{\text{BOOTCS}}$ signal asserts for accesses made to the 64-Kbyte segment that contains the Am5$_x$86 CPU boot vector: addresses 3FF0000h–3FFFFFFh. In addition to this linear decode region, $\overline{\text{BOOTCS}}$ asserts in response to accesses to user-programmable address regions.
$\overline{\text{FLASHWR}}$	—	O	**Flash Write** indicates that the current cycle is a write of the selected Flash device. When this signal is asserted, the selected Flash device can latch data from the data bus.
GPA25–GPA0	—	O	**General-Purpose Address Bus** provides the address to the system's ROM/Flash devices. It is also the address bus for the GP bus devices. Twenty-six address lines provide a maximum addressable space of 64 Mbytes for each ROM chip select.
GPD15–GPD0	—	B	**General-Purpose Data Bus** inputs data during memory and I/O read cycles and outputs data during memory and I/O write cycles. A reset configuration pin (CFG2) allows the GP bus to be used for the boot chip-select ROM interface. Configuration registers are used to select whether $\overline{\text{ROMCS2}}$ and $\overline{\text{ROMCS1}}$ use the GP bus data bus or the MD data bus. The GP data bus supports 16-bit or 8-bit ROM interfaces. Two data buses are selectable to facilitate the use of ROM in a mixed voltage system.
MD31–MD0	—	B	**Memory Data Bus** inputs data during SDRAM read cycles and outputs data during SDRAM write cycles. Configuration registers are used to select whether $\overline{\text{ROMCS2}}$ and $\overline{\text{ROMCS1}}$ use the GP bus data bus or the MD data bus. A reset configuration pin (CFG2) allows the GP data bus to be used for $\overline{\text{BOOTCS}}$. The memory data bus supports an 8-, 16-, or 32-bit ROM interface.
$\overline{\text{ROMBUFOE}}$	—	O	**ROM Buffer Output Enable** is an optional signal used to enable a buffer to the ROM/Flash devices if they need to be isolated from the ÉlanSC520 microcontroller, other **GP bus** devices, or SDRAM system for voltage or loading considerations. This signal asserts for all accesses through the ROM controller. The buffer direction is controlled by the $\overline{\text{ROMRD}}$ or $\overline{\text{FLASHWR}}$ signal.
$\overline{\text{ROMCS2}}$	[$\overline{\text{GPCS2}}$]	O	**ROM/Flash Chip Selects** are signals that can be programmed to be asserted for accesses to user-programmable address regions.
$\overline{\text{ROMCS1}}$	[$\overline{\text{GPCS1}}$]	O	
$\overline{\text{ROMRD}}$	—	O	**ROM/Flash Read** indicates that the current cycle is a read of the selected ROM/Flash device. When this signal is asserted, the selected ROM device can drive data onto the data bus.
Peripheral Component Interconnect (PCI) Bus			
AD31–AD0	—	B	**PCI Address Data Bus** is the PCI time-multiplexed address/data bus.
$\overline{\text{CBE3}}$–$\overline{\text{CBE0}}$	—	B	**Command or Byte-Enable Bus** functions 1) as a time-multiplexed bus command that defines the type of transaction on the AD bus, or 2) as byte enables: $\overline{\text{CBE0}}$ for AD7–AD0 $\overline{\text{CBE1}}$ for AD15–AD8 $\overline{\text{CBE2}}$ for AD23–AD16 $\overline{\text{CBE3}}$ for AD31–AD24
CLKPCIIN	—	I	**PCI Bus Clock Input** is the 33-MHz PCI bus clock. This pin can be connected to the CLKPCIOUT pin for systems where the ÉlanSC520 microcontroller is the source of the PCI bus clock.

Table 2-2 Signal Descriptions (Continued)

Signal	Multiplexed Signal	Type	Description
CLKPCIOUT	—	O	**PCI Bus Clock Output** is a 33-MHz clock output for the PCI bus devices. This signal is derived from the 33MXTAL2–33MXTAL1 interface.
$\overline{\text{DEVSEL}}$	—	B	**Device Select** is asserted by the target when it has decoded its address as the target of the current transaction.
$\overline{\text{FRAME}}$	—	B	**Frame** is driven by the transaction initiator to indicate the start and duration of the transaction.
$\overline{\text{GNT4}}$–$\overline{\text{GNT0}}$	—	O	**Bus Grants** are asserted by the ÉlanSC520 microcontroller to grant access to the bus.
$\overline{\text{INTA}}$–$\overline{\text{INTD}}$	—	I	**Interrupt Requests** are asserted to request an interrupt. These four interrupts are the same type of interrupt as the GPIRQ10–GPIRQ0 signals, and they go to the same interrupt controller. They are named $\overline{\text{INTx}}$ to match the common PCI interrupt naming convention. Configuration registers allow inversion of these interrupt requests to recognize active low interrupt requests. These interrupt requests can be routed to generate NMI.
$\overline{\text{IRDY}}$	—	B	**Initiator Ready** is asserted by the current bus master to indicate that data is ready on the bus (write) or that the master is ready to accept data (read).
PAR	—	B	**PCI Parity** is driven by the initiator or target to indicate parity on the AD31–AD0 and $\overline{\text{CBE3}}$–$\overline{\text{CBE0}}$ buses.
$\overline{\text{PERR}}$	—	B	**Parity Error** is asserted to indicate a PCI bus data parity error in the previous clock cycle.
$\overline{\text{REQ4}}$–$\overline{\text{REQ0}}$	—	I	**Bus Requests** are asserted by the master to request access to the bus.
$\overline{\text{RST}}$	—	O	**Reset** is asserted to reset the PCI devices.
$\overline{\text{SERR}}$	—	I	**System Error** is used for reporting address parity errors or any other system error where the result is catastrophic.
$\overline{\text{STOP}}$	—	B	**Stop** is asserted by the target to request that the current bus transaction be stopped.
$\overline{\text{TRDY}}$	—	B	**Target Ready** is asserted by the currently addressed target to indicate its ability to complete the current data phase of a transaction.
General-Purpose (GP) Bus			
GPA14–GPA0	—	O	**General-Purpose Address Bus** outputs the physical memory or I/O port address. Twenty-six address lines provide a maximum addressable space of 64 Mbytes. This bus also provides the address to the system's ROM/Flash devices.
GPA15	{RSTLD0}	O{I}	
GPA16	{RSTLD1}	O{I}	
GPA17	{RSTLD2}	O{I}	
GPA18	{RSTLD3}	O{I}	
GPA19	{RSTLD4}	O{I}	
GPA20	{RSTLD5}	O{I}	
GPA21	{RSTLD6}	O{I}	
GPA22	{RSTLD7}	O{I}	
GPA23	{AMDEBUG_DIS}	O{I}	
GPA24	{INST_TRCE}	O{I}	
GPA25	{DEBUG_ENTER}	O{I}	

Table 2-2 Signal Descriptions (Continued)

Signal	Multiplexed Signal	Type	Description
[GPAEN]	PIO3	O	**GP Bus Address Enable** indicates that the current address on the GPA25–GPA0 address bus is a memory address, and that the current cycle is a DMA cycle. All I/O devices should use this signal in decoding their I/O addresses and should not respond when this signal is asserted. When GPAEN is asserted, the GPDACKx signals are used to select the appropriate I/O device for the DMA transfer. GPAEN also asserts when a DMA cycle is occurring internally.
[GPALE]	PIO0	O	**GP Bus Address Latch Enable** is driven at the beginning of a **GP bus** cycle with valid address. This signal can be used by external devices to latch the GP address for the current cycle.
[GPBHE]	PIO1	O	**GP Bus Byte High Enable** is driven active when data is to be transferred on the upper 8 bits of the GP data bus.
GPD15–GPD0	—	B	**General-Purpose Data Bus** inputs data during memory and I/O read cycles, and outputs data during memory and I/O write cycles.
[GPDACK0]	PIO12	O	**GP Bus DMA Acknowledge** can each be mapped to one of the seven available DMA channels. They are asserted active Low to acknowledge the corresponding DMA requests.
[GPDACK1]	PIO11	O	
[GPDACK2]	PIO10	O	
[GPDACK3]	PIO9	O	
[GPDBUFOE]	PIO24	O	**GP Bus Data Bus Buffer Output Enable** is used to control the output enable on an external transceiver that may be on the GP data bus. Using this transceiver is optional in the system design and is necessary only to alleviate loading or voltage issues. This pin is asserted for all external **GP bus** accesses. It is not asserted during accesses to the internal peripherals even if **GP bus** echo mode is enabled. Note that if the ROM is configured to use the GP data bus, then its bytes are not controlled by this buffer enable; they are controlled by the ROMBUFOE signal.
[GPDRQ0]	PIO8	I	**GP Bus DMA Request** can each be mapped to one of the seven available DMA channels. They are asserted active High to request DMA service.
[GPDRQ1]	PIO7	I	
[GPDRQ2]	PIO6	I	
[GPDRQ3]	PIO5	I	
[GPIOCS16]	PIO25	STI	**GP Bus I/O Chip-Select 16** is driven active early in the cycle by the targeted I/O device on the **GP bus** to request a 16-bit I/O transfer.
GPIORD	—	O	**GP Bus I/O Read** indicates that the current cycle is a read of the currently addressed I/O device on the **GP bus**. When this signal is asserted, the selected I/O device can drive data onto the data bus.
GPIOWR	—	O	**GP Bus I/O Write** indicates that the current cycle is a write of the currently addressed I/O device on the **GP bus**. When this signal is asserted, the selected I/O device can latch data from the data bus.

Table 2-2 Signal Descriptions (Continued)

Signal	Multiplexed Signal	Type	Description
[GPIRQ0]	PIO23	I	**GP Bus Interrupt Request** can each be mapped to one of the available interrupt channels or NMI. They are asserted when a peripheral requires interrupt service. Configuration registers allow inversion of these interrupt requests to recognize active low interrupt requests. These interrupt requests can be routed to generate NMI.
[GPIRQ1]	PIO22	I	
[GPIRQ2]	PIO21	I	
[GPIRQ3]	PIO20	I	
[GPIRQ4]	PIO19	I	
[GPIRQ5]	PIO18	I	
[GPIRQ6]	PIO17	I	
[GPIRQ7]	PIO16	I	
[GPIRQ8]	PIO15	I	
[GPIRQ9]	PIO14	I	
[GPIRQ10]	PIO13	I	
[$\overline{\text{GPMEMCS16}}$]	PIO26	STI	**GP Bus Memory Chip-Select 16** is driven active early in the cycle by the targeted memory device on the **GP bus** to request a 16-bit memory transfer.
[$\overline{\text{GPMEMRD}}$]	—	O	**GP Bus Memory Read** indicates that the current **GP bus** cycle is a read of the selected memory device. When this signal is asserted, the selected memory device can drive data onto the data bus.
[$\overline{\text{GPMEMWR}}$]	—	O	**GP Bus Memory Write** indicates that the current **GP bus** cycle is a write of the selected memory device. When this signal is asserted, the selected memory device can latch data from the data bus.
[GPRDY]	PIO2	STI	**GP Bus Ready** can be driven by open-drain devices. When pulled Low during a **GP bus** access, wait states are inserted in the current cycle. This pin has an internal weak pullup that should be supplemented by a stronger external pullup for faster rise time.
GPRESET	—	O	**GP Bus Reset**, when asserted, re-initializes to reset state all devices connected to the **GP bus**.
[GPTC]	PIO4	O	**GP Bus Terminal Count** is driven from the internal DMA controller to indicate that the transfer count for the currently active DMA channel has reached zero, and that the current DMA cycle is the last transfer.
Serial Ports			
$\overline{\text{CTS1}}$ $\overline{\text{CTS2}}$	— PIO28	I I	**Clear To Send** is driven back to the serial port to indicate that the external data carrier equipment (DCE) is ready to accept data.
$\overline{\text{DCD1}}$ [$\overline{\text{DCD2}}$]	— PIO30	I I	**Data Carrier Detect** is driven back to the serial port from a piece of DCE when it has detected a carrier signal from a communications target.
$\overline{\text{DSR1}}$ [$\overline{\text{DSR2}}$]	— PIO29	I I	**Data Set Ready** is used to indicate that the external DCE is ready to establish a communication link with the internal serial port controller.
$\overline{\text{DTR2}}$–$\overline{\text{DTR1}}$	—	O	**Data Terminal Ready** indicates to the external DCE that the internal serial port controller is ready to communicate.
$\overline{\text{RIN1}}$ [$\overline{\text{RIN2}}$]	— PIO31	I I	**Ring Indicate** is used by an external modem to inform the serial port that a ring signal was detected.
$\overline{\text{RTS2}}$–$\overline{\text{RTS1}}$	—	O	**Request To Send** indicates to the external DCE that the internal serial port controller is ready to send data.
SIN2–SIN1	—	I	**Serial Data In** is used to receive the serial data from the external serial device or DCE into the internal serial port controller.
SOUT2–SOUT1	—	O	**Serial Data Out** is used to transmit the serial data from the internal serial port controller to the external serial device or DCE.

Table 2-2 Signal Descriptions (Continued)

Signal	Multiplexed Signal	Type	Description
SSI_CLK	—	O	**SSI Clock** is driven by the ÉlanSC520 microcontroller SSI port during active SSI transmit or receive transactions. The idle state of the clock and the assertion/sample edge are configurable.
SSI_DI	—	STI	**SSI Data Input** receives incoming data from a peripheral device SSI port. Data is shifted in on the opposite SSI_CLK signal edge in which SSI_DO drives data. SSI_DO and SSI_DI can be tied together to interface to a three-pin SSI peripheral.
SSI_DO	—	OD	**SSI Data Output** drives data to a peripheral device SSI port. Data is driven on the opposite SSI_CLK signal edge in which SSI_DI latches data. The DO signal is normally at high-impedance when no transmit transaction is active on the SSI port.
Timers			
PITGATE2 PITOUT2	[$\overline{\text{GPCS3}}$] {CFG3}	I O{I}	**Programmable Interval Timer 2 Gate** provides control for the PIT Channel 2. **Programmable Interval Timer 2 Output** is output from the PIT Channel 2. This signal is typically used as the PC speaker signal.
TMRIN0 TMRIN1	[$\overline{\text{GPCS5}}$] [$\overline{\text{GPCS4}}$]	I I	**Timer Inputs 0 and 1** can be programmed to be the control or clock for the general-purpose (GP) timers 0 and 1.
TMROUT0 TMROUT1	[$\overline{\text{GPCS7}}$] [$\overline{\text{GPCS6}}$]	O O	**Timer Outputs 0 and 1** are outputs from two of the GP timers. These outputs can be used as pulse-width modulation signals.
Clocks and Reset			
32KXTAL2– 32KXTAL1	—	osc	**32.768-kHz Crystal Interface** is used for connecting an external crystal or oscillator to the ÉlanSC520 microcontroller. This clock source is used to clock the real-time clock (RTC). In addition, internal PLLs generate clocks for the timers and UARTs based on this clock source. When an external oscillator is used, 32KXTAL1 should be grounded and the clock source driven on 32KXTAL2.
33MXTAL2– 33MXTAL1	—	osc	**33-MHz Crystal Interface** is the main system clock for the chip. This clock source is used to derive the SDRAM, CPU, and PCI clocks. When an external oscillator is used, 33MXTAL1 should be unconnected and the clock source driven on 33MXTAL2.
[CLKTEST]	CLKTIMER	O	**Test Clock Output** is a shared pin that allows many of the internal clocks to be driven externally. CLKTEST can drive the internal clocks of the UARTs, PLL1, PLL2, the programmable interval timer (PIT), or the real-time clock (RTC) for testing or for driving an external device.
CLKTIMER	[CLKTEST]	I	**Timer Clock Input** is a shared clock pin that can be used to input a frequency to the programmable interval timer (PIT).
LF_PLL1	—	I	**Loop Filter Interface** is used for connecting external loop filter components. Component values and circuit descriptions are contained in the *Élan™SC520 Microcontroller Data Sheet*, order #22003.
PRGRESET	—	STI	**Programmable Reset** can be programmed to reset the ÉlanSC520 microcontroller, but allow SDRAM refresh to continue during the reset. This allows the system to be reset without losing the information stored in SDRAM. On power-up, PRGRESET is disabled and must be programmed to be operational. When disabled, this pin has no effect on the ÉlanSC520 microcontroller.
PWRGOOD	—	STI	**Power Good** is a reset signal that indicates to the ÉlanSC520 microcontroller that the V_{CC} levels are within the normal operation range. It is used to reset the entire chip and must be held Low for one second after all V_{CC} signals (except VCC_RTC) on the chip are High. This signal must be returned Low before the V_{CC} signals degrade to put the RTC into the correct state for operation in RTC-only mode.

Table 2-2 Signal Descriptions (Continued)

Signal	Multiplexed Signal	Type	Description
Chip Selects			
[$\overline{\text{GPCS0}}$]	PIO27	O	**General-Purpose Chip Select** signals are for the GP bus. They can be used for either memory or I/O accesses. These chip selects are asserted for Am5$_x$86 CPU accesses to the corresponding regions set up in the Programmable Address Region (PAR) registers.
[$\overline{\text{GPCS1}}$]	$\overline{\text{ROMCS1}}$	O	
[$\overline{\text{GPCS2}}$]	$\overline{\text{ROMCS2}}$	O	
[$\overline{\text{GPCS3}}$]	PITGATE2	O	
[$\overline{\text{GPCS4}}$]	TMRIN1	O	
[$\overline{\text{GPCS5}}$]	TMRIN0	O	
[$\overline{\text{GPCS6}}$]	TMROUT1	O	
[$\overline{\text{GPCS7}}$]	TMROUT0	O	
Programmable I/O (PIO)			
PIO0	[GPALE]	B	**Programmable Input/Output** signals can be programmed as inputs or outputs. When they are outputs, they can be driven High or Low by programming bits in registers.
PIO1	[$\overline{\text{GPBHE}}$]	B	
PIO2	[GPRDY]	B	
PIO3	[GPAEN]	B	
PIO4	[GPTC]	B	
PIO5	[GPDRQ3]	B	
PIO6	[GPDRQ2]	B	
PIO7	[GPDRQ1]	B	
PIO8	[GPDRQ0]	B	
PIO9	[$\overline{\text{GPDACK3}}$]	B	
PIO10	[$\overline{\text{GPDACK2}}$]	B	
PIO11	[$\overline{\text{GPDACK1}}$]	B	
PIO12	[$\overline{\text{GPDACK0}}$]	B	
PIO13	[GPIRQ10]	B	
PIO14	[GPIRQ9]	B	
PIO15	[GPIRQ8]	B	
PIO16	[GPIRQ7]	B	
PIO17	[GPIRQ6]	B	
PIO18	[GPIRQ5]	B	
PIO19	[GPIRQ4]	B	
PIO20	[GPIRQ3]	B	
PIO21	[GPIRQ2]	B	
PIO22	[GPIRQ1]	B	
PIO23	[GPIRQ0]	B	
PIO24	[$\overline{\text{GPDBUFOE}}$]	B	
PIO25	[$\overline{\text{GPIOCS16}}$]	B	
PIO26	[$\overline{\text{GPMEMCS16}}$]	B	
PIO27	[$\overline{\text{GPCS0}}$]	B	
PIO28	[$\overline{\text{CTS2}}$]	B	
PIO29	[$\overline{\text{DSR2}}$]	B	
PIO30	[$\overline{\text{DCD2}}$]	B	
PIO31	[$\overline{\text{RIN2}}$]	B	

Table 2-2 Signal Descriptions (Continued)

Signal	Multiplexed Signal	Type	Description
JTAG Boundary Scan Test Interface			
JTAG_TCK	—	I	**Test Clock** is the input clock for test access port.
JTAG_TDI	—	I	**Test Data Input** is the serial input stream for input data. This pin has a weak internal pullup resistor. It is sampled on the rising edge of JTAG_TCK. If not driven, this input is sampled High internally.
JTAG_TDO	—	O/TS	**Test Data Output** is the serial output stream for result data. It is in the high-impedance state except when scanning is in progress.
JTAG_TMS	—	I	**Test Mode Select** is an input for controlling the test access port. This pin has a weak internal pullup resistor. If it is not driven, it is sampled High internally.
$\overline{\mathrm{JTAG\_TRST}}$	—	I	**JTAG Reset** is the test access port (TAP) reset. This pin has a weak internal pulldown resistor. If not driven, this input is sampled Low internally and causes the TAP controller logic to remain in the reset state.
AMDebug Interface			
BR/TC	—	I	**Break Request/Trace Capture** requests entry to AMDebug technology mode. The AMDebug technology serial/parallel interface can reconfigure this pin to turn instruction trace capture on or off.
CMDACK	—	O	**Command Acknowledge** indicates command completion status. It is asserted High when the in-circuit emulator logic is ready to receive new commands from the host. It is driven Low when the in-circuit emulator core is executing a command from the host and remains Low until the command is completed.
STOP/TX	—	O	**Stop/Transmit** is asserted High on entry to AMDebug mode. During normal mode, this is set High when there is data to be transmitted to the host (during operating system/application communication).
TRIG/TRACE	—	O	**Trigger/Trace** triggers event to logic analyzer (optional, from Am5$_x$86 CPU debug registers).The AMDebug technology serial/parallel interface can reconfigure this pin to indicate the trace on or off status.
System Test			
$\overline{\mathrm{CF\_DRAM}}$	[WBMSTR2] {CFG2}	O{I}	**Code Fetch SDRAM,** during SDRAM reads, provides code fetch status. When Low, this indicates that the current SDRAM read is a CPU code fetch demanded by the CPU, or a read prefetch initiated due to a demand code fetch by the CPU. When High during reads, this indicates that the SDRAM read is not a code fetch, and it could have been initiated by the CPU, PCI master, or the GP bus GP-DMA controller, either demand or prefetch. During SDRAM write cycles this pin provides an indication of the source of the data, either GP-DMA controller/PCI bus master or CPU. When High, this indicates that either a GP bus DMA initiator or an external PCI bus master contributed to the current SDRAM write cycle (the CPU may also have contributed). A Low indicates that the CPU is the only master that contributed to this write cycle.
$\overline{\mathrm{CF\_ROM\_GPCS}}$	[WBMSTR0] {CFG0}	O{I}	**Code Fetch ROM/GPCS** provides an indication that the CPU is performing a code fetch from ROM (on either the GP bus or SDRAM data bus), or from any $\overline{\mathrm{GPCSx}}$ pin. When Low during a read cycle (as indicated by either $\overline{\mathrm{GPMEMRD}}$ or $\overline{\mathrm{ROMRD}}$), the CPU is performing a code fetch from ROM or a GP bus chip select. At all other times (including writes), this signal is High.
DATASTRB	[WBMSTR1] {CFG1}	O{I}	**Data Strobe** is a debug signal that is asserted to allow the external system to latch SDRAM data. This can be used to trace data on the SDRAM interface with an in-circuit emulator probe or logic analyzer.

Table 2-2 Signal Descriptions (Continued)

<table>
<tr><th>Signal</th><th>Multiplexed Signal</th><th>Type</th><th>Description</th></tr>
<tr><td>[WBMSTR0]</td><td>$\overline{\text{CF\_ROM\_GPCS}}$
{CFG0}</td><td>O{I}</td><td><b>Write Buffer Master</b> indicates which block(s) wrote to a rank in the write buffer (during SDRAM write cycles) and which block is reading from SDRAM (during SDRAM read cycles).
<b>WBMSTR0</b>, when a logical 1, indicates that the internal GP bus DMA controller has contributed to the write buffer rank (write cycles) or is reading from SDRAM (read cycles).</td></tr>
<tr><td>[WBMSTR1]</td><td>DATASTRB
{CFG1}</td><td>O{I}</td><td><b>WBMSTR1</b>, when a logical 1, indicates that the PCI master has contributed to the write buffer rank (write cycles) or is reading from SDRAM (read cycles).</td></tr>
<tr><td>[WBMSTR2]</td><td>$\overline{\text{CF\_DRAM}}$
{CFG2}</td><td>O{I}</td><td><b>WBMSTR2</b>, when a logical 1, it indicates that the CPU has contributed to the write buffer rank (write cycles) or is reading from SDRAM (read cycles).</td></tr>
<tr><td colspan="4"><b>Configuration</b></td></tr>
<tr><td>{AMDEBUG_DIS}</td><td>GPA23</td><td>I</td><td><b>AMDebug Disable</b> is an active High configuration signal latched at the assertion of Power Good (PWRGOOD). This pin has a built-in pulldown resistor.
At Power Good assertion:
Low = Normal operation, mode can be enabled by software.
High = AMDebug mode is disabled and cannot be enabled by software.</td></tr>
<tr><td>{CFG0}</td><td>$\overline{\text{CF\_ROM\_GPCS}}$
[WBMSTR0]</td><td>I</td><td><b>Configuration Inputs 3–0</b> are latched into the chip when PWRGOOD is asserted. These signals are all shared with other features. These signals have built-in pulldown resistors.
<b>CFG0:</b> Choose 8-, 16-, or 32-bit ROM/Flash interface for $\overline{\text{BOOTCS}}$.</td></tr>
<tr><td>{CFG1}</td><td>DATASTRB
[WBMSTR1]</td><td>I</td><td><b>CFG1:</b> Choose 8-, 16-, or 32-bit ROM/Flash interface for $\overline{\text{BOOTCS}}$.
<table>
<tr><th>CFG1</th><th>CFG0</th><th>$\overline{\text{BOOTCS}}$ Data Width</th></tr>
<tr><td>0</td><td>0</td><td>8-bit</td></tr>
<tr><td>0</td><td>1</td><td>16-bit</td></tr>
<tr><td>1</td><td>x (don’t care)</td><td>32-bit</td></tr>
</table></td></tr>
<tr><td>{CFG2}</td><td>$\overline{\text{CF\_DRAM}}$
[WBMSTR2]</td><td>I</td><td><b>CFG2</b>: When Low when PWRGOOD is asserted, the ÉlanSC520 microcontroller uses the GP data bus for $\overline{\text{BOOTCS}}$. When seen as High during PWRGOOD assertion, the $\overline{\text{BOOTCS}}$ access is across the SDRAM data bus. Default is Low (by a built-in pulldown resistor).</td></tr>
<tr><td>{CFG3}</td><td>PITOUT2</td><td>I</td><td><b>CFG3 (Internal AMD test mode enable)</b>: <i>For normal ÉlanSC520 microcontroller operation, do not pull High during reset.</i></td></tr>
<tr><td>{DEBUG_ENTER}</td><td>GPA25</td><td>I</td><td><b>Enter AMDebug Mode</b> is an active High configuration signal latched at the assertion of Power Good (PWRGOOD). This pin enables the AMDebug mode, which causes the processor to fetch and execute one instruction from the $\overline{\text{BOOTCS}}$ device, and then enter AMDebug mode where the CPU waits for debug commands to be delivered by the JTAG port. This pin has a built-in pulldown resistor.
At PWRGOOD assertion:
High = AMDebug mode enabled
Low = Normal operation</td></tr>
<tr><td>{INST_TRCE}</td><td>GPA24</td><td>I</td><td><b>Instruction Trace</b> is an active High configuration signal latched at the assertion of Power Good (PWRGOOD). Enables trace record generation from Power Good assertion. This pin has a built-in pulldown resistor.
At PWRGOOD assertion:
High = Trace controller enabled to output trace records
Low = Normal operation</td></tr>
</table>

Table 2-2 Signal Descriptions (Continued)

Signal	Multiplexed Signal	Type	Description
{RSTLD0}	GPA15	I	**Reset Latched Inputs** are shared signals that are latched into a register when PWRGOOD is asserted. They are used to input static information to software (i.e., board revision). These signals have built-in pulldown resistors.
{RSTLD1}	GPA16	I	
{RSTLD2}	GPA17	I	
{RSTLD3}	GPA18	I	
{RSTLD4}	GPA19	I	
{RSTLD5}	GPA20	I	
{RSTLD6}	GPA21	I	
{RSTLD7}	GPA22	I	
Power			
BBATSEN	—	Analog	**Backup Battery Sense** is a pin on which real-time clock (RTC) backup battery voltage is sampled each time PWRGOOD is asserted. If this pin samples below 2.0 V, the Valid RAM and Time (VRT) bit in RTC index 0Dh is cleared until read. After the read, the VRT bit is set until BBATSEN is sensed via a subsequent PWRGOOD assertion. BBATSEN also provides a power-on-reset signal for the RTC when an RTC backup battery is applied for the first time.
VCC_ANLG	—	Power	**Analog Power Supply** for the analog circuits (PLLs).
VCC_CORE	—	Power	**Power Supply** for the ÉlanSC520 microcontroller core logic.
VCC_I/O	—	Power	**Power Supply** to the I/O pad ring.
VCC_RTC	—	Power	**Power Supply** for the real-time clock and 32-kHz oscillator.
GND	—	Power	**Digital Ground** for the remaining ÉlanSC520 microcontroller core logic.
GND_ANLG	—	Power	**Analog Ground** for the analog circuits.

CHAPTER

3 SYSTEM INITIALIZATION

3.1 OVERVIEW

This chapter provides information and guidelines for initializing the ÉlanSC520 microcontroller. Several source code examples of information described in this chapter are available on the AMD web site. This CodeKit software is tested source code for example applications. To obtain this software, as well as other product information and tools, access the AMD home page at **www.amd.com** and follow the Embedded Processors link.

From a software perspective, the types of systems that can be developed with the ÉlanSC520 microcontroller fall into two broad categories, native embedded systems and systems that use a BIOS[1].

Of course, these are not the only types of systems that can be built with the ÉlanSC520 microcontroller. It is quite possible to develop hybrid systems that have a BIOS but do not run a “desktop” operating system like Windows®, DOS, Unix, or Linux. While there are many possible ways to initialize the ÉlanSC520 microcontroller, any initialization sequence can be derived from the following two techniques.

- System initialization with a BIOS
- System initialization for a native embedded system without a BIOS

For systems with a BIOS, most, or all, of the system initialization is done by the BIOS while the system is running in real mode. After initialization, the BIOS loads an operating system or application from nonvolatile media, which is generally a disk drive, but could be Flash memory or other media. The operating system or application begins operating in real mode and then may make its own transition into protected mode. Windows 95 and Windows NT® are examples of such operating systems. Real-time operating systems can also operate in this manner.

BIOS initialization can be complex. Some BIOS products may make a temporary transition into protected mode to perform certain operations and then revert back to real mode, before passing execution to an operating system or application. Such behavior is dependent on how the BIOS is written and the features provided and are beyond the scope of this discussion.

For embedded systems, the initialization sequence is usually much simpler and generally occurs primarily in protected mode. In this scenario, the processor comes up from a reset and transitions into protected mode as soon as possible. The only real-mode code in the system is the code required to jump from the reset vector and the execute code that causes the ÉlanSC520 microcontroller to transition into protected mode.

3.1.1 Native Embedded Initialization Sequence

Many systems designed with the ÉlanSC520 microcontroller are native embedded systems that do not have a BIOS. The software architecture for such systems can take many forms.

[1] A BIOS is a PC software component. It is a set of real-mode code that is responsible for initializing the system and providing a standard set of I/O and system services used by an operating system and application level software. These services are provided via a standard interface.

Some use a commercial real-time operating system (RTOS), a custom RTOS, or a simple ‘main loop’ or non-preemptive executive. In general, the executive or RTOS generally interfaces to the hardware using a hardware dependent layer called a *board support package (BSP)*[1].

In general, the system initialization flow for a native embedded system follows this sequence:

```
1         < Reset event >
2         Near Jump to reset handler from the reset vector
3         Switch to simple protected mode
4         Determine the cause of the reset
5         Initialize the DRAM controller and DRAM. Size the DRAM
6         Setup a Stack and begin execution from “C” code
7         if (NOT Execute-In-Place) then
8                   Copy the Operating System to DRAM
9                   Jump to the operating system’s entry point
10        Set up the Global Descriptor Table (GDT), Local Descriptor Table (LDT),
          Interrupt Descriptor Table (IDT), fault handlers, page tables, and a
          Task State Segment (TSS) for the operating system, application or
          executive.
11        Set the processor speed
12        Configure the GP bus timings
13        Configure the pin multiplexing
14        Configure the GP bus chip selects
15        Configure the Programmable Address Region (PAR) registers
16        Configure the interrupt mappings
17        Configure the programmable I/O (PIO) pins
18        Configure the PCI bus controller and arbitration mode
19        Initialize a periodic timer interrupt (if necessary)
20        Now, the BSP can initialize devices external to the ÉlanSC520
          microcontroller and otherwise continue to start the operating system,
          I/O drivers and application.
```

In the above example, the switch to simple protected mode (line 3) sets the processor CS register and the CS descriptor cache. This disables the redirection of the reset region to the reset segment (see “Reset Vector and Reset Segment” on page 3-5 for more information).

In line 3 above, the term *simple protected mode* means that the protected mode environment (GDT, LDT, IDT, and TSS) is the simplest kind possible. For example, both the LDT and IDT can be empty and the TSS and GDT can contain minimal information. Or, alternatively, the IDT can be empty. This means that exceptions cannot be handled, but this should not be a problem for the short period that the initialization code runs. More importantly, the TSS and GDT for simple protected mode can be contained in read-only memory (usually Flash) and do not have to be created at runtime. Once the DRAM is operational, then more extensive GDT, LDT, and IDT tables and one or more appropriate TSS can be setup in DRAM.

[1] There is no standard term for this component. Other terms for BSP are OEM Adaptation Layer (OAL), Hardware Adaptation Layer (HAL), or Porting Layer. A BSP is like a BIOS, but is almost always unique to a specific executive or RTOS. This is especially true for comercially available RTOS products. A BSP for one vendor’s RTOS generally does not work with products from another vendor. Also, where a BIOS is most often a 16-bit real-mode entity, a BSP is usually a 32-bit protected mode entity. Lastly, operating systems and applications always communicate with a BIOS using software interrupts (or other run-time mechanisms), but a BSP is often linked directly to an executive or application to form a single executable and is called directly using the CALL instruction.

Some embedded systems execute from read-only memory (usually Flash) and only use DRAM for data storage. This style of system architecture is supported by most RTOS products. This is reflected in line 7. Systems that execute out of Flash memory do not need to copy the operating system and/or application to DRAM.

Another interesting point is that once the DRAM controller is initialized, then the initialization code can setup a stack and finish the reset of its work in a high-level language (usually C).

3.1.2 BIOS Initialization Sequence

In contrast to a native embedded system, the flow of system initialization with a BIOS generally follows this sequence:

```
1         < Reset event >
2         Near Jump to reset handler from the reset vector
3         Map the Memory-Mapped Configuration Region (MMCR) to an address below
          0010FFEFh (real-mode address limit)
4         Determine the cause of the reset
5         Initialize the DRAM controller and DRAM. Size the DRAM, record in CMOS
6         Copy the BIOS into DRAM (shadowing)
7         Execute a Far Jump within the BIOS code to start execution out of the
          shadowed BIOS copy instead of the copy in ROM
8         Set up basic interrupt handlers for processor faults
9         Detect the CPU ID and display on the console
10        Set the processor speed
11        Configure the GP bus timings
12        Configure the pin multiplexing
13        Configure the GP bus chip selects
14        Configure the Programmable Address Region (PAR) registers
15        Configure the interrupt mappings
16        Configure the programmable I/O (PIO) pins
17        Configure the PCI bus controller and arbitration
18        Now, the BIOS can continue with standard PC-style system initialization
```

There are some important contrasts between the steps for a system with a PC BIOS and those for a native embedded system.

- Steps 1 through 6 are done in real mode while executing from the reset segment before executing the first Far Jump (JMP) instruction. This is in contrast to the initialization for a native embedded system, which transitions to simple protected mode before these steps.
- The Memory-Mapped Configuration Region (MMCR) needs to be mapped to a region below 00100000h so it is accessible by real-mode software. 32-bit protected-mode native embedded systems do not need to move the MMCR.
- The remainder of the system initialization is done in real mode from the BIOS image running from DRAM. This is in contrast to an embedded system, which does all of its initialization from 32-bit protected mode (running either from DRAM or Flash).

3.1.3 Memory-Mapped Configuration Region (MMCR)

The Memory-Mapped Configuration Region (MMCR) is a 4-Kbyte area located at physical address FFFEF000h and contains various configuration and control registers for the ÉlanSC520 microcontroller. Configuring and controlling many of the device's features requires accessing the MMCR registers. System initialization code for a native embedded system can access this region directly because most (or all) initialization takes place from 32-bit protected mode.

In contrast, real-mode code cannot access physical memory above 0010FFEFh (the real-mode addressing limit), and thus cannot access the default location of the MMCR. This problem is easily resolved by programming the Configuration Base Address (CBAR) register (Port FFFCh) to place the MMCR at an address somewhere below the real-mode addressing limit. This allows real-mode initialization code to directly access the MMCR. This is done in step 3 of the BIOS initialization sequence.

Note: *Programming the* Configuration Base Address (CBAR) register *can place the MMCR at an address other than its default. However, the MMCR region is always accessible at its default location of FFFEF000h, regardless of how the CBAR register is programmed.*

3.1.4 Reset Event

The ÉlanSC520 microcontroller has three primary classes of resets.

- System reset (often called a hard reset or power-on reset)
- System reset with SDRAM retention (called programmable reset)
- Soft reset (often called warm start)

For more information on resetting the ÉlanSC520 microcontroller, see Chapter 6, “Reset Generation”, and “Initialization” on page 7-5.

Often, systems have a hardware reset button or other external devices that can cause a reset. For the ÉlanSC520 microcontroller, all of these cause a system reset. However, there are many ways to implement external reset logic. After a reset (of any kind), boot software can determine what caused the reset by examining various status bits.

A common and effective method of handling a reset is to determine the cause of the reset and record the event in the CMOS memory, or in some other non-volatile memory such as an EEPROM, non-volatile DRAM, or Flash. Debugging or diagnostic software could then examine and report the causes of the last few resets. This can be very helpful when trying to determine the cause of system problems. Note that the system could record other information as well; the time and date of the reset event is a good example.

When a system reset occurs (regardless of the source) internal registers and logic blocks are set to their power-on reset state. Therefore, if a system reset occurs, the boot software must initialize the system from scratch.

There is one exception to this, called *programmable reset*. This function is enabled via the PRG_RST_ENB bit in the Reset Configuration (RESCFG) register (MMCR offset D72h). If this bit is set, assertion of the PRGRESET pin, SYS_RST bit, watchdog timer system reset event, or AMDebug technology system reset event while PWRGOOD is asserted will result in a system reset in which the SDRAM configuration (SDRAM type, number of banks, refresh rate, etc.) is maintained so that the contents of SDRAM are preserved. SDRAM controller parameters retained include the SDRAM type, number of banks, refresh rate, and signal drive strength. This feature allows the system to be reset while guaranteeing that the contents of SDRAM are not disturbed. This can be very valuable for system debugging or for systems that require minimal startup time. This reset condition can be detected by software. Note that, once programmable reset has been enabled, all system resets other than PRWGOOD deassertion are converted to this type.

When a soft reset occurs, the system may be able to restart if the operating system saved enough state information. For example, an old 80286-style operating system (e.g., OS/2) causes a processor reset in order to return to real mode and call 16-bit BIOS routines.

Note: *It is important to understand that, for most systems, a soft reset does not need to be handled much differently than a system reset. For example, a system that does not need* *to explicitly perform a soft restart will simply cause a system reset when a soft reset is detected.*

Note that the watchdog timer can generate an interrupt (maskable or non-maskable) or a system reset, or both. Handling watchdog timer time-outs can be complex. For more information on how the WDT operates, see Chapter 19, “Watchdog Timer”.

3.1.5 Reset Vector and Reset Segment

Immediately after a hard or soft reset, the Am5$_X$86 CPU core begins execution in real mode at the address F000:FFF0. This real-mode address is called the *reset vector*. While the reset vector is a real-mode address, it is a redirection of the physical address FFFFFFF0h, which is located at the top physical address of the memory device selected by $\overline{\text{BOOTCS}}$. This device is called the *boot ROM device*.

After a hard or soft reset, the 64-Kbyte physical address space from FFFF0000 to FFFFFFFFh (resident in the boot ROM device) is redirected into real-mode address space from F000:0000 to F000:FFFF. This real-mode region is called the *reset segment*. The region in the boot ROM device is called the *reset region*. The code that resides in this area is called the *reset handler*.

This redirection is not performed by the addressing unit, but is an artifact of the values programmed into the CS descriptor cache by the CPU at reset time. After any reset, the CPU core sets the base value of CS Descriptor Cache register to FFFF0000h with a limit of 0000FFFFh (64 Kbytes). The processor CS:EIP register pair is set to F000:0000FFF0.

The redirection works because, in real mode, linear addresses for code fetches are generated by taking the offset in EIP and adding it to the contents of the base register in the CS descriptor cache. Since the paging unit is disabled at reset, these linear addresses map directly to physical addresses.

This simple mechanism causes both the redirection of the reset code region to the reset segment and the first instruction fetch to occur from the reset vector.

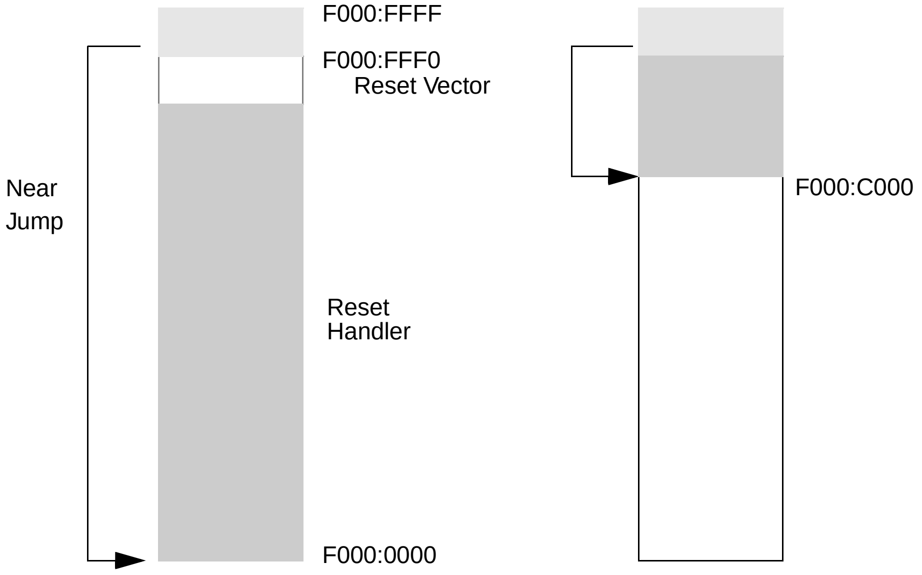
Note that none of the other segment registers (and internal descriptor registers) have this behavior. This behavior is *only* applicable to the CS Segment register and its internal descriptor cache. For more information on the configuration of the processor registers at reset, see the *Am486® DX/DX2 Microprocessor Hardware Reference Manual*, 1994 (order #17965).

What this means is that the artificial reset segment redirection is only active until the CPU executes a Far Jump (JMP) instruction. This is because a Far Jump instruction causes the CS Segment register to be reloaded. When a segment register is loaded in real mode, the processor sets the value of the corresponding descriptor cache base register to 16 times the new value of the segment register. Since the processor is running in real mode, the internal CS Descriptor registers are set to their normal real-mode values.

Since the reset vector is at F000:FFF0, there are only 16 bytes before the end of the segment. That is only enough for a few instructions. So, regardless of how much (or how little) the reset code does, the instruction at the reset vector must be a *Near Jump* into the reset region.

For example, as shown in Figure 3-1, if the reset handler is large, then the initial Near Jump could be to F000:0000.

Figure 3-1 Initial Near Jump Example



The reset vector Near Jump is not required to jump to F000:0000. It can jump anywhere into the reset segment. For example, if the reset handler code is only 16 Kbytes in size, it could jump to F000:C000, leaving more room on the boot ROM device for other code. This allows the reset handler to be placed right up against the reset vector, thus using the space in the boot ROM device more efficiently.

Note: *For debugging using AMDebug technology, not only should this first Jump instruction be a Near Jump, it should be a Jump Near Indirect instruction, which is opcode FF/4. In-circuit emulation and debug software that uses the internal trace cache searches for this opcode to aid in determining when the reset event occurred.*

As much or as little of the system initialization code can take place in the reset handler while the system is executing from the reset segment (i.e., before the first Far Jump instruction). For example, a native embedded system using a 32-bit only RTOS will merely setup the protected mode data structures, switch to protected mode, and jump directly into system boot code (the boot ROM device is the device selected by $\overline{\text{BOOTCS}}$).

In contrast, a system with a PC-style BIOS would initialize the SDRAM controller, shadow the BIOS to SDRAM, and then jump to the BIOS.

3.2 CONFIGURING THE SDRAM CONTROLLER

After a system reset, the SDRAM controller configuration registers are reset to their default states. All the SDRAM controller banks and SDRAM refresh are disabled by default. For details on how to enable the SDRAM controller and the SDRAM configuration, see “Initialization” on page 10-29.

Note that the ÉlanSC520 microcontroller can be reset in a manner that preserves the operation of the SDRAM controller. This condition can be detected and handled properly by the SDRAM initialization code.

If the Error Correction Code (ECC) logic for SDRAM is enabled, ECC operation requires that SDRAM and its associated ECC memory be initialized. This is accomplished by the boot code, which must write to every location in SDRAM. This process initializes the ECC SDRAM to reflect the proper error-checking codes. If this procedure is not performed, false errors will occur when writing data smaller than a 32-bit doubleword. For a more detailed discussion of ECC, see “Error Correction Code (ECC)” on page 10-16.

3.3 IDENTIFYING THE CPU CORE

Information about the integrated Am5$_X$86 CPU core is available by reading the processor DX register after a system reset and by using the CPUID instruction at any time. The CPUID instruction is available on later model 32-bit processors from all leading x86 vendors and allows programs to determine information about the CPU, including the manufacturer, cache type, and availability of a floating point unit (FPU). By using the CPUID instruction, software can determine the type of CPU running the system. For example, software could detect that it is running on an Am5$_X$86 CPU and perform the appropriate action.

The ÉlanSC520 Microcontroller Revision ID (REVID) register (MMCR offset 00h) can be used to identify the revision of the device itself.

A user-modifiable bit in the CPU’s Flags register called the ID bit indicates support of the CPUID instruction. The ID bit is reset to 0 at CPU hard or soft reset for compatibility with existing processor designs.

The results reported by the CPUID instruction reflect the state of the processor at the last CPU hard or soft reset. If the CPU cache write mode or core clock speed is changed, and if the CPU encounters a soft reset following the change, then a subsequent CPUID instruction will report the altered condition of the processor (i.e., the state at the time the soft reset occurred). After a hard CPU reset, the ÉlanSC520 microcontroller always reports the cache mode as write-back and the clock speed as 100 MHz.

The CPUID instruction returns encodings shown in Table 3-1.

Table 3-1 CPUID Codes

CPU	Clock Speed	Write-Back Mode	Write-Through Mode
Am5$_X$86 CPU	100 MHz	0494h	0484h
Am5$_X$86 CPU	133 MHz	04F4h	04E4h

3.4 SETTING THE CPU SPEED

The ÉlanSC520 microcontroller is available at multiple clock speeds. By default, the ÉlanSC520 microcontroller core comes up from a system reset running at 100 MHz. See Chapter 7, “Am5x86® CPU”, for more information.

Note: *Not all ÉlanSC520 microcontroller devices support all Am5$_X$86 CPU clock rates. The maximum supported clock rate for a device is indicated by the part number printed on the package. The clocking circuitry can be programmed to run the device at higher than rated speeds. However, if an ÉlanSC520 microcontroller is programmed to run at a higher clock speed than that for which it is rated, then erroneous operation will result, and physical damage to the device may occur.*

3.5 CONFIGURING EXTERNAL GP BUS DEVICES

Programming the ÉlanSC520 microcontroller to support external peripherals on the GP bus requires three steps.

1. Program the GP bus timing mechanism to control the bus timings for the device. This is done first so that the initial access to the device (after the chip selects and PARs are programmed) will function properly. The GP bus timings and bus cycles are discussed in “Bus Cycles” on page 13-16.

2. If needed, program the PIO pin logic to map the GP bus chip select signal and other control signals to a physical pin.
3. Program a PAR register to map the external peripheral into physical address space and to configure a chip select for the device.

For peripherals connected externally to the GP bus, the Programmable Address Region registers control where they are mapped into the I/O or memory address space. Programming and using these registers is discussed in Section 3.7.

3.6 CONFIGURING THE PIN MULTIPLEXING

The ÉlanSC520 microcontroller has several pins that are multiplexed to two functions. There are no pins that have three functions. Most of the pins that are multiplexed are programmable input/output pins (PIOs).

To program a pin that is multiplexed with a PIO, its corresponding function bit must be set in the PIO31–PIO16 Pin Function Select (PIOPFS31_16) register (MMCR offset C22h) or the PIO15–PIO0 Pin Function Select (PIOPFS15_0) register (MMCR offset C20h).

Other pins with multiple programmable functions are all noted in Figure 2-2 on page 2-3.

3.7 CONFIGURING THE PROGRAMMABLE ADDRESS REGION (PAR) REGISTERS

The PAR registers provide a common programming interface to configure physical memory and I/O regions in an ÉlanSC520 microcontroller system. PAR registers are programmed by atomically writing 32-bit values. See “Programmable Address Region (PAR) Registers” on page 4-5 for more information on using the PAR registers. “Software Considerations” on page 4-18 provides other important details.

The PAR registers are used to define four characteristics.

- Target device
- Attributes for the address region
- Size of the address region
- Start address for the region

It is important to note that the PAR registers are used to define *physical* address regions. PAR registers are not used to define effective address regions or linear address regions. For example, an effective address (often called a logical or virtual address) gets translated into a linear address by the Am5$_X$86 CPU’s segmentation unit. If the paging unit is enabled, then linear addresses get translated into physical addresses and placed on the CPU’s bus. If the paging unit is not enabled, then the mapping from linear address to physical address is direct (one-to-one).

Depending on how your system is set up, driver software, system software and other software that must be aware of physical addresses should be written to take the Am5$_X$86 CPU addressing modes into account. This can be an extremely complex topic and is beyond the scope of this chapter.

The general format of the PAR registers is shown in Figure 3-2 on page 3-10. Provided as a programming aid, Figure 3-3 on page 3-11 is a blank worksheet for calculating PAR register values.

3.7.1 Specifying Pages and Regions

For memory-mapped address regions, the Region Size/Start Address (SZ_ST_ADR) bit field in the PAR registers specifies the number of 64-Kbyte or 4-Kbyte pages for the region.

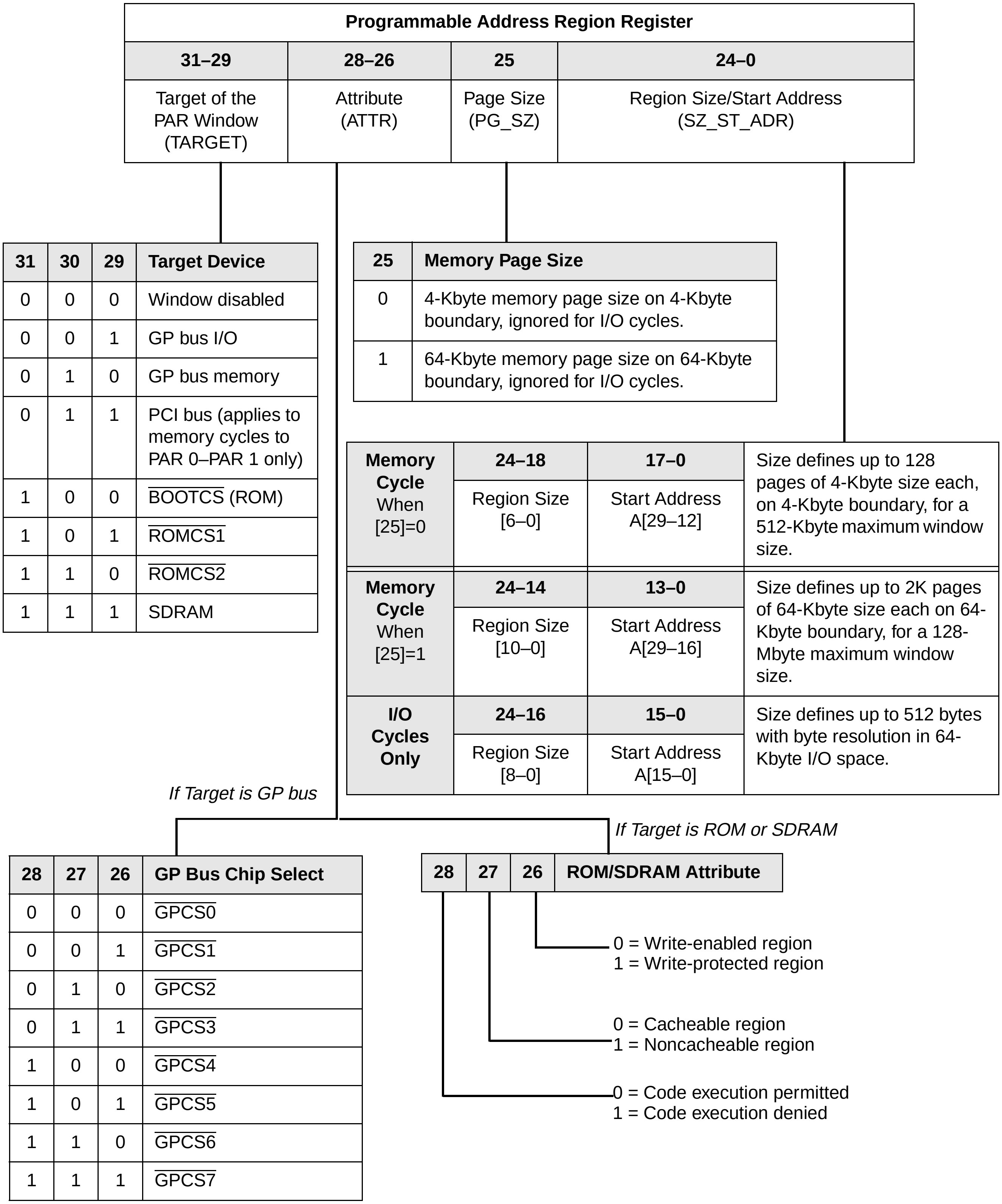
Regions using a 64-Kbyte page size can have up to 2048 pages, for a maximum size of 128 Mbytes. Regions using a 4-Kbyte page size can have up to 128 pages, for a maximum size of 512 Kbytes.

- To specify the number of pages for a region, the value (page count minus 1) is programmed into the SZ_ST_ADR field of the PAR register.
  - For example, to specify a 16-Kbyte region using a 4-Kbyte page size, the value 03h (0000011b) would be programmed into bits 24–18 of a PAR register, i.e., one less than the required number of pages.
  - To specify a page count of one, all the bits in the SZ_ST_ADR field for a PAR register should be cleared to 0.
  - To specify the maximum number of pages, either 2048 or 128, all the bits in the SZ_ST_ADR field should be set to 1.
- To specify the 4-Kbyte page size, the Page Size (PG_SZ) bit should be cleared to 0. For a 64-Kbyte page size, it should be set to 1.

The same holds true for GP bus I/O-mapped regions. The region size field specifies the number of bytes in the addressable region. For example, to specify a region size of 8 bytes, the value 07h (0111b) should be programmed into the SZ_ST_ADR field of the PAR register.

Note: *For GP bus I/O-mapped regions, the PAR registers' PG_SZ bit is ignored. In general, it should be cleared to 0 for GP bus I/O regions.*

Figure 3-2 Programmable Address Region (PAR) Register Format



Programmable Address Region Register			
31–29	**28–26**	**25**	**24–0**
Target of the PAR Window (TARGET)	Attribute (ATTR)	Page Size (PG_SZ)	Region Size/Start Address (SZ_ST_ADR)

31	30	29	Target Device
0	0	0	Window disabled
0	0	1	GP bus I/O
0	1	0	GP bus memory
0	1	1	PCI bus (applies to memory cycles to PAR 0–PAR 1 only)
1	0	0	$\overline{\text{BOOTCS}}$ (ROM)
1	0	1	$\overline{\text{ROMCS1}}$
1	1	0	$\overline{\text{ROMCS2}}$
1	1	1	SDRAM

25	Memory Page Size
0	4-Kbyte memory page size on 4-Kbyte boundary, ignored for I/O cycles.
1	64-Kbyte memory page size on 64-Kbyte boundary, ignored for I/O cycles.

Memory Cycle When [25]=0	**24–18**	**17–0**	Size defines up to 128 pages of 4-Kbyte size each, on 4-Kbyte boundary, for a 512-Kbyte maximum window size.
	Region Size [6–0]	Start Address A[29–12]	
Memory Cycle When [25]=1	**24–14**	**13–0**	Size defines up to 2K pages of 64-Kbyte size each on 64-Kbyte boundary, for a 128-Mbyte maximum window size.
	Region Size [10–0]	Start Address A[29–16]	
I/O Cycles Only	**24–16**	**15–0**	Size defines up to 512 bytes with byte resolution in 64-Kbyte I/O space.
	Region Size [8–0]	Start Address A[15–0]	

If Target is GP bus

28	27	26	GP Bus Chip Select
0	0	0	$\overline{\text{GPCS0}}$
0	0	1	$\overline{\text{GPCS1}}$
0	1	0	$\overline{\text{GPCS2}}$
0	1	1	$\overline{\text{GPCS3}}$
1	0	0	$\overline{\text{GPCS4}}$
1	0	1	$\overline{\text{GPCS5}}$
1	1	0	$\overline{\text{GPCS6}}$
1	1	1	$\overline{\text{GPCS7}}$

If Target is ROM or SDRAM

28	27	26	ROM/SDRAM Attribute

Bit 26:
0 = Write-enabled region
1 = Write-protected region

Bit 27:
0 = Cacheable region
1 = Noncacheable region

Bit 28:
0 = Code execution permitted
1 = Code execution denied

Figure 3-3 Programmable Address Region (PAR) Register Worksheet

<table>
<tr><td colspan="3" rowspan="3">Target Device</td><td colspan="3" rowspan="3">Attribute</td><td rowspan="3">Page Size</td><td colspan="11">Region Size, 64-Kbyte Pages</td><td colspan="14">Start Address (on 64-Kbyte Boundary)</td><td rowspan="3">Fields</td></tr>
<tr><td colspan="7">Region Size, 4-Kbyte Pages</td><td colspan="18">Start Address (on 4-Kbyte Boundary)</td></tr>
<tr><td colspan="9">Region Size, I/O Bytes</td><td colspan="16">I/O Location Base</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>Bits</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>Binary</td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td>Hex</td></tr>
</table>

<table>
<tr><td colspan="3" rowspan="3">Target Device</td><td colspan="3" rowspan="3">Attribute</td><td rowspan="3">Page Size</td><td colspan="11">Region size, 64-Kbyte Pages</td><td colspan="14">Start Address (on 64-Byte Boundary)</td><td rowspan="3">Fields</td></tr>
<tr><td colspan="7">Region Size, 4-Kbyte Pages</td><td colspan="18">Start Address (on 4-Kbyte Boundary)</td></tr>
<tr><td colspan="9">Region Size, I/O Bytes</td><td colspan="16">I/O Location Base</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>Bits</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>Binary</td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td>Hex</td></tr>
</table>

<table>
<tr><td colspan="3" rowspan="3">Target Device</td><td colspan="3" rowspan="3">Attribute</td><td rowspan="3">Page Size</td><td colspan="11">Region size, 64-Kbyte Pages</td><td colspan="14">Start Address (on 64-Kbyte Boundary)</td><td rowspan="3">Fields</td></tr>
<tr><td colspan="7">Region Size, 4-Kbyte Pages</td><td colspan="18">Start Address (on 4-Kbyte Boundary)</td></tr>
<tr><td colspan="9">Region Size, I/O Bytes</td><td colspan="16">I/O Location Base</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>Bits</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>Binary</td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td>Hex</td></tr>
</table>

<table>
<tr><td colspan="3" rowspan="3">Target Device</td><td colspan="3" rowspan="3">Attribute</td><td rowspan="3">Page Size</td><td colspan="11">Region size, 64-Kbyte Pages</td><td colspan="14">Start Address (on 64-Kbyte Boundary)</td><td rowspan="3">Fields</td></tr>
<tr><td colspan="7">Region Size, 4-Kbyte Pages</td><td colspan="18">Start Address (on 4-Kbyte Boundary)</td></tr>
<tr><td colspan="9">Region Size, I/O Bytes</td><td colspan="16">I/O Location Base</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>Bits</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>Binary</td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td>Hex</td></tr>
</table>

3.7.2 Address Region Attributes

The address region attributes (as specified in the ATTR bit field of a PAR register) can be used with ROM or SDRAM regions to control how the regions can be accessed. This section includes some examples of how the attributes can be used with SDRAM and ROM regions.

3.7.2.1 Write-Protect Attribute

When this feature is enabled for an address region in SDRAM or ROM, an interrupt is generated when a write is performed to the region. This interrupt can be used to find problems with errant software or to help debug Flash programming code.

3.7.2.2 Cacheability Control Attribute

The Cacheability Control Attribute bit in the PAR registers provides a simple mechanism for controlling the caching of memory regions. This mechanism is much easier to use than the Am5$_X$86 CPU's paging unit.

For SDRAM regions, turning off caching can be useful for regions that contain buffers used for DMA or for PCI bus mastering devices.

This feature is also useful for Flash regions. For some operations, it is necessary to turn off caching for a Flash region. An example is when a Flash device needs to be erased or programmed. Any time a Flash device's internal registers need to be read or written, caching should be disabled for the device. For example, the Flash sector erasing code needs to poll the device to see when erases and other operations are complete. If caching is not turned off, then the software will merely continue to read the value from the processor's cache and not the correct value from the device. This is also true during the Flash programming write/verify cycle. For more information, see page 12-12.

3.7.2.3 Code Execution Attribute

Execution control works in a similar manner to the Write-Protect Attribute bit. The difference is that when this bit is set, any code fetches by the CPU to the defined region will cause an invalid opcode fetch fault to be generated. This is accomplished by returning an invalid opcode to the CPU, instead of the data resident in the device at the requested address.

This is very useful for debugging problems. Large areas of the address space can be execute-protected. For example, the Flash for a file system could be protected from code execution. Data reads and writes for the Flash file system would happen normally. But, if a code erroneously jumped into this data area, an invalid opcode fetch fault would be generated immediately.

3.7.2.4 Performance Considerations

It is possible to control the same attributes that the PAR registers provide using the native mechanisms in the Am5$_X$86 CPU core. For example, 4-Kbyte pages can be write-protected using the paging unit and paging tables. Noncached regions can also be created using this mechanism. Execution protection can also be performed using a segmented code model and descriptor attributes.

Using the native x86 mechanisms will work, but using the address region attributes in a PAR register is easier and provides higher performance. If the CPU's paging unit is enabled, the entire system takes a small performance hit because all linear address must be translated to physical address. Also, defining nonexecutable regions is very difficult to do and requires 48-bit code pointers (huge pointers) and a fully segmented 32-bit code model. This is a high price to pay to obtain execute-only regions. These performance penalties are not incurred when using the ÉlanSC520 microcontroller's address region attribute mechanism.

3.7.3 PAR Register Priority

The PAR register mechanism is a very flexible and useful one. It is designed to allow the system programmer to easily program the address decoding and set attributes for addressable regions. One feature of the PAR register system that may not be obvious from the examples included in this chapter is that the PAR registers have a priority mechanism. The highest priority PAR register is PAR 0 and the lowest priority register is PAR 15. This feature is not relevant unless two (or more) PAR regions overlap. If they do overlap, then the higher priority PAR register takes precedence.

The PAR registers are used to modify and add to the default system addressing (see Table 4-4 on page 4-4). Note that the system can function quite well with all of the PAR registers disabled. For example, a system could start-up, use a PAR register to copy the contents of Flash to SDRAM[1], jump to the code in SDRAM, and then disable the PAR register used for the copy. With all the PAR registers disabled, the normal address resolution priorities in the system govern addressing of physical devices.

3.7.4 External GP Bus Devices

Devices on the GP bus can be addressed in two ways. Each is controlled by programming the PAR registers.

- By chip select, mapping the device into memory or I/O space
- Devices can do their own memory or I/O address decoding.

Programming a PAR register with the GP bus as the target is required to cause memory or I/O cycles to be forwarded to the external GP bus. This is true for devices that use chip selects and devices that decode their own address (generate their own chip selects). Programming a PAR register is necessary because, by default, memory and I/O cycles generated by the Am5$_X$86 CPU that are not decoded by an internal GP bus peripheral or memory resources (like SDRAM, ROM, and the MMCR registers) go to the PCI bus.

For a device on the external GP bus, programming a PAR register configures the following characteristics:

- Target device field—For either a GP bus memory-mapped cycle or an I/O cycle
- Attribute field—For the particular GP bus chip select to which the device is attached
- Memory page size field— Most peripherals use a 4-Kbyte granularity. Peripherals that have very large memory address spaces, such as SDRAM or ROM, might need to use a 64-Kbyte granularity.
- Region size and start address

For a device that requires a chip select from the ÉlanSC520 microcontroller, the chip select must be mapped to a physical pin using the PIO registers. For devices that do their own address decoding, the PAR register must still be programmed, and the chip select should be chosen; however, the chip select from the PAR register does not need to be mapped to a physical pin.

Note: *All of the internal peripherals on the GP bus are decoded at fixed locations. The locations for these peripherals cannot be changed by programming a PAR register. For example, the internal real-time clock cannot be moved to a different location. No PAR registers are required to access any of the internal peripheral devices on the ÉlanSC520 microcontroller.*

[1] This is one way to shadow a BIOS to DRAM.

3.7.4.1 Single Device (an A/D Converter) Using One Chip Select

In this example, an A/D converter has four 16-bit registers that need to be mapped into I/O space on GPCS5 at I/O address 0500h. As shown in Table 3-2, the value to program into a PAR register in this case is 34070500h.

Table 3-2 Example PAR Programming: Single Device Using One Chip Select

Bit Field	Value	Meaning
Target Device	001b	GP bus I/O space
Attribute Field	101b	GPCS5
Page Size	0b	Clear to 0 (this bit not applicable to I/O space)
Region Size	7h	Specifies an 8-byte region size
Start Address	0500h	Physical address 0500h

3.7.4.2 Single Device That Performs Its Own Decode

In this example, an external memory-mapped 16-color 480 x 320 pixel LCD controller performs its own address decoding. It needs a 128-Kbyte window mapped at 000C0000h. A chip select must be used (specified in the ATTR bit field of the PAR register), but it does not need to be mapped to an external pin. GPCS7 is used here. As shown in Table 3-3, the value to program into a PAR register in this case is 5E00400Ch.

Table 3-3 Example PAR Programming: Single Device That Performs Its Own Decode

Bit Field	Value	Meaning
Target Device	010b	GP bus memory space
Attribute Field	111b	GPCS7
Page Size	1b	64-Kbyte granularity
Region Size	1h	Specifies two 64-Kbyte pages for a 128-Kbyte region size
Start Address	000Ch	Physical address 000C0000h

3.7.4.3 Multiple Devices On One Chip Select

A single PAR register can be programmed for a larger range than is needed by a single peripheral. For example, consider a bank of 16 memory-mapped A/D converters, each of which has four 16-bit registers. An external PAL is programmed to do the address decoding for each individual A/D converter. The converters will be memory-mapped to a range of 00020000–0002003Fh. The PAL generates the chip selects for each of the four converters by watching for the appropriate memory read and write cycles and is qualified from GPCS2 from the ÉlanSC520 microcontroller. As shown in Table 3-4, the value to program into a PAR register in this case is 48000020h.

Table 3-4 Example PAR Programming: Multiple Devices on One Chip Select

Bit Field	Value	Meaning
Target Device	010b	GP bus memory space
Attribute Field	010b	GPCS2
Page Size	0b	4-Kbyte granularity
Region Size	0h	One 4-Kbyte page
Start Address	20h	Physical address 00020000h

3.7.5 PCI Bus Devices

Normally, devices on the PCI bus are mapped into memory space that is above the configured amount of DRAM and just under 4 Gbytes (FFFEFFFFh). The ÉlanSC520 microcontroller's address decode logic forwards all access to these memory locations to the PCI bus.

Normally, memory cycles below the top address used by SDRAM are forwarded only to the SDRAM controller, or to the GP bus if a PAR register is appropriately programmed. However, for Windows and DOS compatibility, some PCI peripherals need to be mapped into SDRAM space. These regions usually fall below the real-mode address limit (physical address 0010FFEFh). Devices that can require this include PCI-based VGA video cards and PCI-based network adapters. To allow this, the first two PAR registers support the PCI bus as a target. Note PCI as a target can only be specified in PAR 0 and PAR 1.

For such devices, a PAR register must be programmed that allows addresses lower than the highest SDRAM address to be forwarded to the PCI bus. This is in addition to the normal PCI bus device configuration. The VGA controller example in Section 3.7.5.1 illustrates this.

Typically, all I/O space accesses above the 1-Kbyte boundary are forwarded to the PCI bus, and all I/O space accesses below the 1-Kbyte boundary are forwarded to the GP bus.

- With some minor exceptions for the CBAR and PCI configuration registers, the I/O space above the 1-Kbyte boundary can be redirected from the PCI to the GP bus using PAR registers.
- The IO_HOLE_DEST bit in the Address Decode Control (ADDDECCTL) register (MMCR offset 80h) can be programmed to allow *all* I/O space addresses below the 1-Kbyte boundary that are not assigned to internal peripherals to be forwarded to the PCI bus.
- Note that PAR registers can still be mapped in the lower 1-Kbyte I/O space to override the IO_HOLE_DEST bit. This way, I/O devices in the lower 1-Kbyte space can reside internally to the ÉlanSC520 microcontroller, on the external GP-Bus, and on the PCI bus.

3.7.5.1 VGA Controller on the PCI Bus

A VGA video controller's 128 Kbytes of memory is normally mapped from 000A0000–000BFFFFh (physical addresses). So, to support a PCI-based video controller, PAR 0 or PAR 1 would need to be programmed to 7200400Ah. This configures PAR 0 or PAR 1 with the characteristics shown in Table 3-5. The attribute fields are ignored for the PCI bus target. PCI regions are always writable, executable, and noncached.

Table 3-5 Example PAR Programming: VGA Controller on the PCI Bus

Bit Field	Value	Meaning
Target Device	011b	PCI bus
Attribute Field	000b	Not applicable
Page Size	1b	64-Kbyte granularity
Region Size	1h	Specifies two 64-Kbyte pages for a 128-Kbyte region size
Start Address	Ah	Physical address 000A0000h

A PCI VGA video adapter also requires PCI I/O from addresses 03B0–03BBh and 03C0–03CFh. A PAR register is not required to map these I/O locations to PCI space, but instead the IO_HOLES_DEST bit must be set in the Address Decode Control (ADDDECCTL) register (MMCR offset 80h). This has the effect of mapping *all* external I/O accesses to PCI space rather than to the GP bus. If there are no external GP bus I/O devices, then no further

PAR programming is required to support this configuration. Note that the internal I/O devices will still be correctly accessed when the IO_HOLES_DEST bit is set.

However, if any external GP bus device requires I/O addresses, then a PAR register will be required to allow access to this device. As an example, assume an external 16550 UART is used to implement a COM3 port.

The standard I/O locations for COM3 are 03E8–03EFh. As shown in Table 3-6, a PAR register will be required with a setting of 340703E8hto enable external GP bus accesses to this I/O range. In this example, $\overline{\text{GPCS5}}$ is used as a chip enable for the external device. If another $\overline{\text{GPCSx}}$ is required, then appropriate changes should be made to the PAR register setting.

Table 3-6 Example PAR Programming: COM3 with VGA Present on the PCI Bus

Bit Field	Value	Meaning
Target Device	001b	GP bus I/O space
Attribute Field	101b	$\overline{\text{GPCS5}}$
Page Size	0b	Clear to 0 (this bit not applicable to I/O space)
Region Size	7h	Specifies an 8-byte region size
Start Address	03E8h	Physical address 03E8h

3.7.5.2 Network Adapter for Remote Program Loading

A memory-mapped network adapter will usually reside in PCI space that is far above the real-mode address limit. However, to perform Remote Program Loading (RPL), often called network boot, over a network, the 16-bit BIOS needs to use the network adapter. To avoid writing 32-bit protected-mode BIOS code, PAR 0 or PAR 1 can be used to place a memory-mapped network adapter above the real-mode address limit. For this example, it is assumed that the network adapter has 16 Kbytes of address space that needs to be placed at 000B0000h. This area is noncacheable because it is PCI address space. As shown in Table 3-7, the value to configure PAR 0 or PAR 1 for this configuration is 600C00B0h.

Table 3-7 Example PAR Programming: Network Adapter for Remote Program Loading

Bit Field	Value	Meaning
Target Device	011b	PCI bus
Attribute Field	000b	Not applicable
Page Size	0b	4-Kbyte granularity
Region Size	03h	Specifies four 4-Kbyte pages for a 16-Kbyte region size
Start Address	B0h	Physical address 000B0000h

Note that most network adapters will also require a small amount of PCI I/O space. The location of this I/O space can usually be changed through a PCI configuration register on the adapter and can be assigned by an operating system through plug and play functionality. Usually, this address can be set to any value and is typically above the 1-Kbyte I/O boundary affected by the IO_HOLES_DEST bit. Since I/O accesses above 400h are always sent to PCI space (unless overridden by a PAR register to go to the GP bus), no special programming is needed to allow I/O accesses for a typical PCI network adapter.

3.7.6 External ROM Devices

The PAR registers can also be used to define the addressing for ROM devices selected by $\overline{\text{BOOTCS}}$, $\overline{\text{ROMCS1}}$, and $\overline{\text{ROMCS2}}$. ROM devices include true ROMs, EEPROM, Flash devices, and other similar devices.

It is important to note that the top 64 Kbytes of the ROM device selected by $\overline{\text{BOOTCS}}$ (the boot device chip select) is *always* mapped to the physical addresses from FFFF0000–FFFFFFFFh. This area is called the reset region. The reset region is cached, executable, and not write-protected. This 64-Kbyte mapping is fixed and always active, even if the boot ROM device is mapped to another address using a PAR register. ROM devices attached to $\overline{\text{BOOTCS}}$, $\overline{\text{ROMCS1}}$, or $\overline{\text{ROMCS2}}$ can be mapped anywhere in physical address space below 40000000h (1 Gbyte).

3.7.6.1 Boot ROM Device Mapping for BIOS Shadowing

A 512-Kbyte Flash device is a common boot ROM device for systems with a BIOS. One way to shadow the BIOS is to map it below 00100000h so that it can be accessed by real-mode code. This is easily done with a single PAR register. For shadowing purposes, a good place to park the boot ROM device is at 00001000h, which is just above the interrupt vector table. The value 89FC0001h configures the PAR register as shown in Table 3-8.

Table 3-8 Example PAR Programming: Boot ROM Device Mapping for BIOS Shadowing

Bit Field	Value	Meaning
Target Device	100b	$\overline{\text{BOOTCS}}$
Attribute Field	010b	Write enable, noncacheable, code execution permitted
Page Size	0b	4-Kbyte granularity
Region Size	7Fh	Specifies 128 4-Kbyte pages for a 512-Kbyte region size
Start Address	1h	Physical address 00001000h

3.7.6.2 Two Banks of Flash for an Execute-In-Place (XIP) Operating System

A system has eight 8-Mbit byte-wide Flash devices. Four are on $\overline{\text{ROMCS1}}$ and four on $\overline{\text{ROMCS2}}$. These devices will be mapped into eight Mbytes of contiguous 32-bit address space from 00400000–00BFFFFFh. This requires two PAR registers because two ROM chip selects need to be used. This example uses PAR 4 and PAR 5. Note that in addition to programming the PAR registers, the ROM chip selects need to be mapped to physical pins.

The value A20FC040h for PAR 4 would setup $\overline{\text{ROMCS1}}$ for the first bank of Flash. This configures the PAR register with the characteristics shown in Table 3-9. The value C20FC080h for PAR 5 would setup $\overline{\text{ROMCS2}}$ for the first bank of Flash. This configures the PAR register with the characteristics shown in Table 3-10.

Table 3-9 Example PAR Programming: First Bank of Flash for XIP Operating System

Bit Field	Value	Meaning
Target Device	101b	$\overline{\text{ROMCS1}}$
Attribute Field	000b	Write enable, cacheable, code execution allowed
Page Size	1b	64-Kbyte granularity
Region Size	3Fh	Specifies sixty-four 64-Kbyte pages for a 4-Mbyte region size
Start Address	40h	Physical address 00400000h

Table 3-10 Example PAR Programming: Second Bank of Flash for XIP Operating System

Bit Field	Value	Meaning
Target Device	110b	$\overline{\text{ROMCS2}}$
Attribute Field	000b	Write enable, cacheable, code execution allowed
Page Size	1b	64-Kbyte granularity
Region Size	3Fh	Specifies sixty-four 64-Kbyte pages for a 4-Mbyte region size
Start Address	80h	Physical address 00800000h

3.7.7 SDRAM Regions

The PAR registers can also be used to define regions of SDRAM and control the read/write, cacheability, and execution attributes.

3.7.7.1 Setting Up DMA Buffers

Often PCI and GP bus devices use GP-DMA or PCI bus mastering to read and write data directly from buffers in SDRAM. It is often useful to mark such buffers as noncached. This can be done using the CPU's paging unit, but doing so is complex and may conflict with how an operating system uses the page tables.

In any case, disabling caching for a region is quite simple. Setting the Cacheability Control Attribute (bit 27) in a PAR register defines a buffer region. For example, a 512-Kbyte region can be defined to store transmit and receive buffers for a fast Ethernet PCI controller. Since this is a data-only area, the Code Execution Attribute (bit 28) is set.

Assuming that the region is located at physical address 00020000h, a PAR register would be programmed with the value F9FC0020h. This configures the PAR register with the characteristics shown in Table 3-11.

Table 3-11 Example PAR Programming: Setting Up DMA Buffers

Bit Field	Value	Meaning
Target Device	111b	SDRAM
Attribute Field	110b	Write enable, noncacheable, code execution denied
Page Size	0b	4-Kbyte granularity
Region Size	7Fh	Specifies 128 4-Kbyte pages for a 512-Kbyte region size
Start Address	20h	Physical address 00200000h

Of course, this is not absolutely necessary. The cache controller in the ÉlanSC520 microcontroller always maintains the coherency between the cache and SDRAM. For buffer regions used by GP-DMA channels or PCI bus masters, disabling caching with a PAR register is more efficient and provides better bus performance than allowing the CPU to cache the buffer. This avoids the bus activity (and latency) involved with keeping the cache and the SDRAM coherent.

3.7.7.2 Write-Protected Code Segments

In many embedded systems, all (or most) of the applications and operating system code is contiguous in memory. In such cases, a single PAR register can be used to write-protect most (or all) of the code in a system. If errant code attempted to write to the protected region, then an interrupt would be generated. Note that the CPU completes the write cycle, but the SDRAM or ROM controller (as appropriate) prevents the write from occurring at the device.

Several actions could be taken, from merely preventing the write from taking place, to killing the offending thread, or even restarting the system. Also, the event could be recorded and/ or reported to a debugging or diagnostic interface or console port. During debugging, a breakpoint could be set at the front of the write-protect interrupt service routine.

Assuming the system code resides in the first 768 Kbytes of SDRAM at address 0, the value E602C000h configures a PAR register with the values shown in Table 3-12.

Table 3-12 Example PAR Programming: Write-Protected Code Segments

Bit Field	Value	Meaning
Target Device	111b	SDRAM
Attribute Field	001b	Write disable, cacheable, code execution permitted
Page Size	1b	64-Kbyte granularity
Region Size	Bh	Specifies twelve 64-Kbyte pages for a 768-Kbyte region size
Start Address	0h	Physical address 00000000h

3.8 CONFIGURING THE INTERRUPT MAPPING

The ÉlanSC520 microcontroller has very flexible interrupt routing and control capability. Each of the hardware interrupt sources can be mapped to any of the different interrupt priority levels in the programmable interrupt controller (PIC).

In contrast to a basic PC, which has fixed interrupt mappings and operation, the ÉlanSC520 microcontroller has a very flexible interrupt management architecture. For full details on this system, see Chapter 15, “Programmable Interrupt Controller”. The information in “Interrupt Sources” on page 15-8 is of particular importance.

The following sections discuss options to be considered for the software that configures interrupts.

3.8.1 Edge-Sensitive or Level-Triggered Interrupts

Edge- and level-triggering can be programmed for each PIC or on an interrupt-by-interrupt basis.

For example, all of the interrupts on the Slave 2 interrupt controller could be programmed for edge-triggered operation.

- Setting the S2_GINT_MODE bit in the Interrupt Control (PICICR) register (MMCR offset D00h) allows the LTIM bit in the Slave 2 PIC Initialization Control Word 1 (S2PICICW1) register (Port 0024h) to control how interrupts are triggered for that controller.
- If the S2_GINT_MODE bit is cleared, then the edge- or level-triggered nature is controlled for each interrupt input to the PIC individually using the Slave 2 PIC Interrupt Mode (SL2PICMODE) register (MMCR offset D04h).

3.8.2 Interrupt Mapping

Using the Interrupt Mapping registers, each interrupt source can be mapped to one of the interrupt channels in the PIC block, the NMI interrupt, or can be disabled as an interrupt input. The flexibility of the ÉlanSC520 microcontroller allows any interrupt source in the system to trigger either a regular interrupt or an NMI.

3.8.3 Interrupt Polarity

Each of the interrupt controllers can recognize either a Low-to-High edge-triggered or an active High level-sensitive interrupt request. To support external devices that generate active Low interrupt requests (either edge or level), a programmable inversion of each of the external interrupt requests is available.

Many devices generate a Low-going interrupt signal using an open-collector output. These devices are easily supported on the ÉlanSC520 microcontroller by setting the appropriate bit in the Interrupt Pin Polarity (INTPINPOL) register (MMCR offset D10h). For example, if such a device were connected to GPIRQ8, then setting GPINT8_POL in the Interrupt Pin Polarity (INTPINPOL) register would program the interrupt for a Low-going interrupt input.

It is important to ensure that the polarity values for all internal interrupt sources are programmed correctly at reset time.

3.9 CONFIGURING THE PROGRAMMABLE I/O PINS

An important part of the ÉlanSC520 microcontroller initialization is configuration of the programmable I/O (PIO) pins. These are general-purpose I/O pins that can be programmed as inputs or outputs. When configured as an input, the state of the input can be read using the PIOx_DATA bit in the PIOx Data register.

The PIO pins can also be configured as outputs by setting their corresponding direction bits in the PIOx Direction registers.

3.10 CONFIGURING THE PCI HOST BRIDGE AND ARBITRATION

The PCI Host Bridge must be configured and initialized *before* PCI operation such as enumeration and device configuration take place. There are two parts to the PCI host bridge configuration: ÉlanSC520 microcontroller-specific configuration and normal PCI bus configuration.

1. Configure the PCI host bridge.
   a. Program the desired ÉlanSC520 microcontroller arbitration mode, including concurrency mode and PCI bus master arbitration priorities, etc. See “Initialization” on page 8-22, for more detailed information on arbitration.
   b. Program the Programmable Address Region (PAR) registers, if required. If there are one or two VGA video controllers, PAR 0 and PAR 1 may need to be programmed to place the VGA graphics memory in SDRAM space at PC-compatible locations. PAR 0 and PAR 1 could also be used for other PCI peripherals (such as a network card) that require mapping below physical address 00100000h. See Chapter 4, “System Address Mapping”, for details on programming PCI bus memory space.
   c. Program the ÉlanSC520 microcontroller-specific PCI host bridge configuration (write posting, retry time-out counter, interrupts, etc.). Note that write-posting must be disabled while operating in nonconcurrent arbitration mode. See Chapter 8, “System Arbitration”, for further details on nonconcurrent mode arbitration.
   d. Program the standard PCI bus configuration registers. See “Configuration Information” on page 9-9 for more information.
2. Configure the external PCI bus devices.

In general, PCI host bridge configuration bits should not be changed except during a PCI bus initialization after a system or programmable reset.

3.11 DISABLING INTERNAL PERIPHERALS

Most applications will use the ÉlanSC520 microcontroller’s internal UART devices and its internal real-time clock (RTC). However, some applications might need to use external devices mapped to these same I/O locations. To use external devices, the corresponding internal device must be disabled. This is necessary because these internal peripherals are at fixed I/O locations and cannot be re-mapped. If any internal devices are disabled, accesses to the I/O addresses for these peripherals are forwarded to the external GP bus.

Disabling these peripherals turns off their address decoding, so that externally connected peripherals can be used in their place. If the addresses cannot be externally decoded without a chip select, a PAR register must be mapped to allow a chip select to be asserted for these addresses.

Using external devices in place of the internal ones might be necessary for several reasons. A common reason would be to use a multifunction external chip that has parallel ports, serial ports, floppy disk controller, an RTC, and other devices.

- The internal RTC can be disabled by setting the RTC_DIS bit in the Address Decode Control (ADDDECCTL) register (MMCR offset 80h).
- UART 1 and UART 2 can be disabled by setting the UART1_DIS and UART2_DIS bits in the Address Decode Control (ADDDECCTL) register.

Note that, if the internal peripherals are disabled, the external peripheral’s interrupt signals will need to be connected to external interrupt lines, which then need to be routed to the appropriate interrupt channel. For example, if an external UART is used to replace UART 2 (as COM2), then its interrupt could be connected to GPIRQ8, which would then need to be routed to interrupt priority P3.

Also, in this scenario, the pin used for GPIRQ8 would need to be configured as a general-purpose IRQ (the interface function for the pin, not its default PIO function) by setting the PIO15_FNC bit in the PIO15–PIO0 Pin Function Select (PIOPFS15_0) register (MMCR offset C20h).

Note: *When the internal peripherals are disabled, they are still fully functional. Disabling the peripherals disables the address decoding* only *for that device. For example, if the RTC is programmed to generate interrupts and then subsequently disabled, it will continue to generate interrupts but will no longer be accessible. Before disabling an internal peripheral, be sure to turn off its interrupts.*

CHAPTER

4 SYSTEM ADDRESS MAPPING

4.1 OVERVIEW

The ÉlanSC520 microcontroller includes flexible memory and I/O address decoding with features for both real-time operating systems (RTOS) and systems requiring PC/AT functionality for Windows compatibility. Address decoding is distributed between the memory controllers, GP bus controller, and PCI host bridge controller. The ÉlanSC520 microcontroller provides the following memory and I/O address mapping options.

- The default SDRAM map is linear space starting at 00000000h through the top of SDRAM (defined by the total size of the SDRAM array, up to a maximum of 256 Mbytes).
- The default boot ROM/Flash chip select ($\overline{\text{BOOTCS}}$ pin) is mapped in a 64-Kbyte linear region at the top of CPU memory space from FFFF0000–FFFFFFFFh, and this entire ROM space can be redirected through configuration registers (address translation is not supported).
- All configuration registers that do not reside in PC/AT I/O space or PCI configuration space are memory-mapped and are located in a 4-Kbyte region in memory address space from FFFEF000–FFFEFFFFh.
  - This 4-Kbyte region is called the *memory-mapped configuration region (MMCR)*.
  - The MMCR can optionally be relocated on any 4-Kbyte boundary in the lower 1-Gbyte region via an I/O mapped register called the Configuration Base Address (CBAR) register (Port FFFCh).
  - The default MMCR region in high memory (below the boot space) is visible even if it is aliased via the Configuration Base Address (CBAR) register.
- The default PCI bus map is contiguous space starting directly above the top of SDRAM through 4 Gbytes, minus the 68 Kbytes for the boot ROM/Flash region and the MMCR.
- 16 general-purpose Programmable Address Region (PAR) windows allow address mapping for a variety of applications, including operating systems requiring x86 real mode support. Each window allows any memory region in the lower 1-Gbyte region to be directed to the following resources:
  - Any of three ROM chip-selects with the ability to apply cacheability, write-protection, and nonexecutable region attributes
  - Any of eight GP bus chip-selects for external memory or I/O peripherals on the GP bus
  - Two PAR registers allow cycles to be forwarded to the PCI bus for applications that require PCI space to be overlaid on top of SDRAM. All accesses above the top of SDRAM to the top of 32-bit memory space are *automatically* forwarded to PCI bus (with the exception of the ROM boot space and memory-mapped configuration space).
  - Accesses in normal SDRAM space (lower 256 Mbytes) can also be redirected to ROM, the GP bus, or the PCI bus.
  - PAR windows can be created in the SDRAM region to allow noncacheable, write-protected, and/or nonexecutable buffers.

- Integrated PC/AT compatible peripherals are direct-mapped in normal PC I/O space (i.e., the programmable interrupt controller, programmable interval timer, GP bus DMA controller, RTC, and UARTs). All remaining integrated peripherals are memory-mapped (the watchdog timer, software timer, GP timers, and SSI).
- As a PCI target, the PCI bus host bridge decodes normal SDRAM address space, allowing external PCI bus master access of the entire SDRAM space. PCI bus I/O accesses from PCI masters are not decoded by the PCI host bridge.

4.2 REGISTERS

Address decoding is controlled by the configuration registers listed in Table 4-1 and Table 4-2.

Table 4-1 Address Decoding Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
Address Decode Control	ADDDECCTL	80h	RTC disable, UART 1 and UART 2 disables, write protect violation interrupt enable, I/O hole access destination
Write-Protect Violation Status	WPVSTA	82h	Write-protect violation interrupt status, master, window number
Programmable Address Region 0	PAR0	88h	General-purpose resource decoding
Programmable Address Region 1	PAR1	8Ch	General-purpose resource decoding
Programmable Address Region 2	PAR2	90h	General-purpose resource decoding
Programmable Address Region 3	PAR3	94h	General-purpose resource decoding
Programmable Address Region 4	PAR4	98h	General-purpose resource decoding
Programmable Address Region 5	PAR5	9Ch	General-purpose resource decoding
Programmable Address Region 6	PAR6	A0h	General-purpose resource decoding
Programmable Address Region 7	PAR7	A4h	General-purpose resource decoding
Programmable Address Region 8	PAR8	A8h	General-purpose resource decoding
Programmable Address Region 9	PAR9	ACh	General-purpose resource decoding
Programmable Address Region 10	PAR10	B0h	General-purpose resource decoding
Programmable Address Region 11	PAR11	B4h	General-purpose resource decoding
Programmable Address Region 12	PAR12	B8h	General-purpose resource decoding
Programmable Address Region 13	PAR13	BCh	General-purpose resource decoding
Programmable Address Region 14	PAR14	C0h	General-purpose resource decoding
Programmable Address Region 15	PAR15	C4h	General-purpose resource decoding

Table 4-2 Address Decoding Registers—Direct-Mapped

Register	Mnemonic	I/O Address	Function
Configuration Base Address	CBAR	FFFCh	Base address for the alias of the MMCR registers

4.3 OPERATION

There are three types of system bus masters supported on the ÉlanSC520 microcontroller: the Am5$_x$86 CPU, the PCI bus, and the GP bus DMA controller.

As shown in Table 4-3, each of the three bus masters can access specific types of address space.

- The Am5$_x$86 CPU and the PCI bus each implement separate memory and I/O address space.
- The PCI bus further specifies a separate space for device configuration registers.
- The GP bus DMA controller supports fly-by transfers between GP bus devices and SDRAM; therefore, as a bus master, it supports memory space only.

Table 4-3 Bus Master Address Spaces

Bus Master and Address Space		SDRAM	ROM	GP Bus	PCI Bus	Integrated PC/AT Peripherals	Integrated Non-PC/AT Peripherals	Memory-Mapped Registers	CBAR Register
CPU	Memory	✔	✔	✔	✔		✔	✔	
	I/O			✔	✔	✔			✔
PCI Bus	Memory	✔			✔				
	I/O				✔				
	Configuration[1]								
GP-DMA	Memory	✔							

Notes:

[1] Accessed indirectly by the CPU via the PCI configuration registers in I/O space.

The Am5$_x$86 CPU and PCI bus definition support separate memory and I/O address spaces (I/O space is limited to 64 Kbytes on the CPU). The *PCI Local Bus Specification,* Revision 2.2, further defines a separate space for configuration registers.

The ÉlanSC520 microcontroller divides these address spaces as follows:

- Memory space
  - ROM/Flash space for data and code storage using up to three chip selects (accessible only by the CPU)
  - SDRAM space for data and code storage
  - GP bus memory space (accessible only by the CPU)
  - PCI bus memory space (accessible only by the CPU and PCI bus masters)
  - Internal memory-mapped configuration region (MMCR) registers (accessible only by the CPU)
- I/O space
  - Integrated PC/AT-compatible peripherals (accessible only by the CPU)
  - Configuration Base Address (CBAR) register (Port FFFCh) to set the MMCR’s base address (accessible only by the CPU)

- GP bus I/O space (accessible only by the CPU)
- PCI bus I/O space (accessible by the CPU and PCI masters)
- PCI bus configuration space (accessible only by the CPU)

Table 4-4 summarizes the organization of memory and I/O address regions in the ÉlanSC520 microcontroller.

Table 4-4 Memory and I/O Space Summary

Device	Memory Space	I/O Space
SDRAM	• Linear space starting at 00000000h to top of SDRAM (maximum 256 Mbytes) • PAR registers define noncacheable, write-protected, nonexecutable regions	N/A
ROM/Flash	• $\overline{\text{BOOTCS}}$ mapped to CPU boot space from FFFF0000–FFFFFFFFh (64 Kbytes) • PAR registers define noncacheable, write-protected, nonexecutable regions	N/A
PCI Bus Normal Space	• Default above SDRAM to top of memory address space (4 Gbytes), minus boot space (64 Kbytes) and MMCR (4 Kbytes) • Two PAR registers can define any region that overlays SDRAM space	Any space not claimed by CBAR, PC/AT peripherals, GP bus (via PAR registers), or PCI configuration registers (0CF8–0CFFh)
PCI Bus Configuration Space	N/A	0CF8–0CFFh
GP Bus	Defined via PAR registers in lower 1 Gbyte	Defined via PAR registers in lower 64 Kbytes, except for integrated peripherals' I/O space
Integrated PC/AT Peripherals	N/A	0000h-03FFh
MMCR Registers	• Defaults to 4-Kbyte region starting at FFFEF000h • CBAR can alias this to any 4-Kbyte boundary in lower 1 Gbyte	N/A
Configuration Base Address (CBAR) Register	N/A	FFFC–FFFFh

4.3.1 Programming External Memory, Buses, and Chip Selects

Programming the external memory, buses, and chip selects on the ÉlanSC520 microcontroller is accomplished in three steps:

1. Configure the address space and any required attributes for the specified region.
2. Configure the timing, when applicable, and any required attributes of the interface.
3. For chip selects, enable the function on the desired pin by programming the pin multiplexing in the PIO registers.

This chapter describes how to complete step 1. Programming the required timing and attributes of the external interface (i.e., SDRAM, ROM, GP bus, or PCI bus) is accomplished by writing to registers that control these interfaces. Finally, for chip selects, see Chapter 23, “Programmable Input/Output”, which describes enabling the actual programmable I/O (PIO) pins that can be shared with other functions.

4.3.2 Programmable Address Region (PAR) Registers

Programmable Address Region (PAR) registers provide a common programming interface to configure memory space and I/O space regions in an ÉlanSC520 microcontroller system. As referenced in Table 4-4, the PAR registers are primarily used to define the address regions of ROM and GP bus, as well as to set attributes for ROM and SDRAM regions.

The first two PAR registers (PAR 0 and PAR 1) also allow the user to redirect CPU accesses that normally fall into SDRAM space to the PCI bus, for special cases that require this functionality. The ÉlanSC520 microcontroller provides a total of 16 PAR registers to provide the user with flexibility in organizing memory space and I/O space in the system. They are organized in a priority scheme starting with the lowest register (PAR 0). Thus, if overlapping regions are programmed, the lowest number PAR register takes priority. The PAR registers are 32 bits each and reside in the MMCR space.

Since the ÉlanSC520 microcontroller supports PC/AT-compatible peripherals, the regions required for these peripherals are fixed in I/O space and are not relocatable via PAR registers. This includes the GP bus DMA controller, the programmable interval timer (PIT), the programmable interrupt controller (PIC), the two 16550-compatible UARTs, the real-time clock (RTC), and the PC/AT port logic.

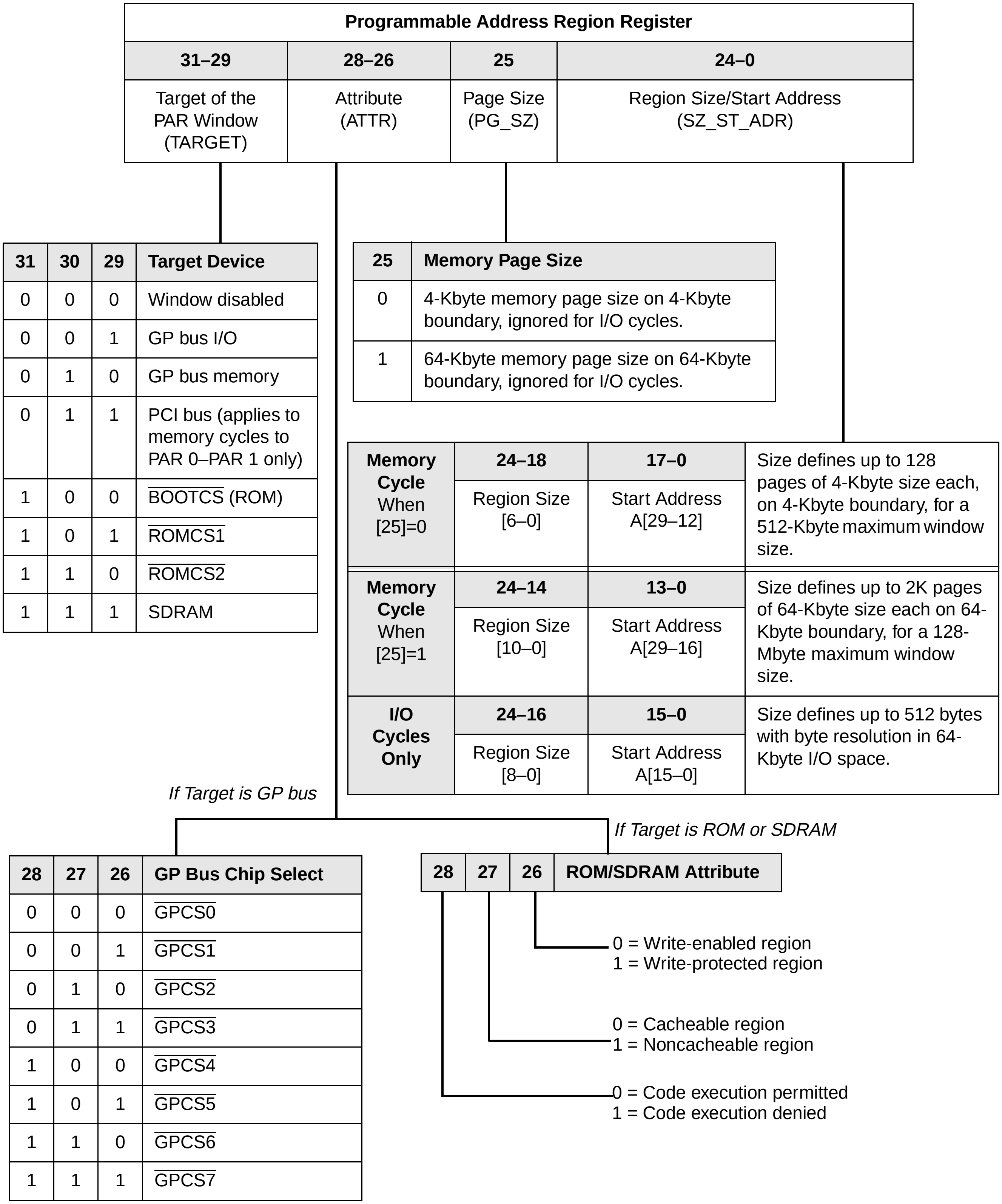
Figure 4-1 illustrates the layout of the 32-bit PAR register. Note that the registers are organized in four sections, as follows:

- The Target (TARGET) bit field defines the destination of the cycle (i.e., ROM, GP bus, etc.).
- The Attribute (ATTR) bit field allows memory regions to be programmed with special conditions such as write-protection and noncacheability for ROM or SDRAM access or selects a specific chip select for GP bus accesses.
- The Page Size (PG_SZ) bit defines the size of each memory page within the regions.
- The Region Size/Start Address (SZ_ST_ADR) bit field is used to define both the beginning of the region and the total size of the region (in conjunction with the Page Size bit).

The PAR register is used to define only the actual address space for the targets; it does not control the parameters for timing and bus width required for ROM and GP bus devices. Those controls must be programmed independently in the ROM controller and GP bus controller configuration registers.

Note: *If a PAR window is configured for PCI, AND the CBAR register is programmed to overlap with this PAR window, AND the PAR window is placed below the top of DRAM, the MMCR is not given priority over the PCI access. This configuration could result in system errors due to concurrence of both PCI and internal MMCR accesses.*

Figure 4-1 Programmable Address Region (PAR) Register Format



Programmable Address Region Register			
31–29	**28–26**	**25**	**24–0**
Target of the PAR Window (TARGET)	Attribute (ATTR)	Page Size (PG_SZ)	Region Size/Start Address (SZ_ST_ADR)

31	30	29	Target Device
0	0	0	Window disabled
0	0	1	GP bus I/O
0	1	0	GP bus memory
0	1	1	PCI bus (applies to memory cycles to PAR 0–PAR 1 only)
1	0	0	$\overline{\text{BOOTCS}}$ (ROM)
1	0	1	$\overline{\text{ROMCS1}}$
1	1	0	$\overline{\text{ROMCS2}}$
1	1	1	SDRAM

25	Memory Page Size
0	4-Kbyte memory page size on 4-Kbyte boundary, ignored for I/O cycles.
1	64-Kbyte memory page size on 64-Kbyte boundary, ignored for I/O cycles.

Memory Cycle When [25]=0	**24–18**	**17–0**	Size defines up to 128 pages of 4-Kbyte size each, on 4-Kbyte boundary, for a 512-Kbyte maximum window size.
	Region Size [6–0]	Start Address A[29–12]	
Memory Cycle When [25]=1	**24–14**	**13–0**	Size defines up to 2K pages of 64-Kbyte size each on 64-Kbyte boundary, for a 128-Mbyte maximum window size.
	Region Size [10–0]	Start Address A[29–16]	
I/O Cycles Only	**24–16**	**15–0**	Size defines up to 512 bytes with byte resolution in 64-Kbyte I/O space.
	Region Size [8–0]	Start Address A[15–0]	

If Target is GP bus

28	27	26	GP Bus Chip Select
0	0	0	$\overline{\text{GPCS0}}$
0	0	1	$\overline{\text{GPCS1}}$
0	1	0	$\overline{\text{GPCS2}}$
0	1	1	$\overline{\text{GPCS3}}$
1	0	0	$\overline{\text{GPCS4}}$
1	0	1	$\overline{\text{GPCS5}}$
1	1	0	$\overline{\text{GPCS6}}$
1	1	1	$\overline{\text{GPCS7}}$

If Target is ROM or SDRAM

28	27	26	ROM/SDRAM Attribute

Bit 26:
0 = Write-enabled region
1 = Write-protected region

Bit 27:
0 = Cacheable region
1 = Noncacheable region

Bit 28:
0 = Code execution permitted
1 = Code execution denied

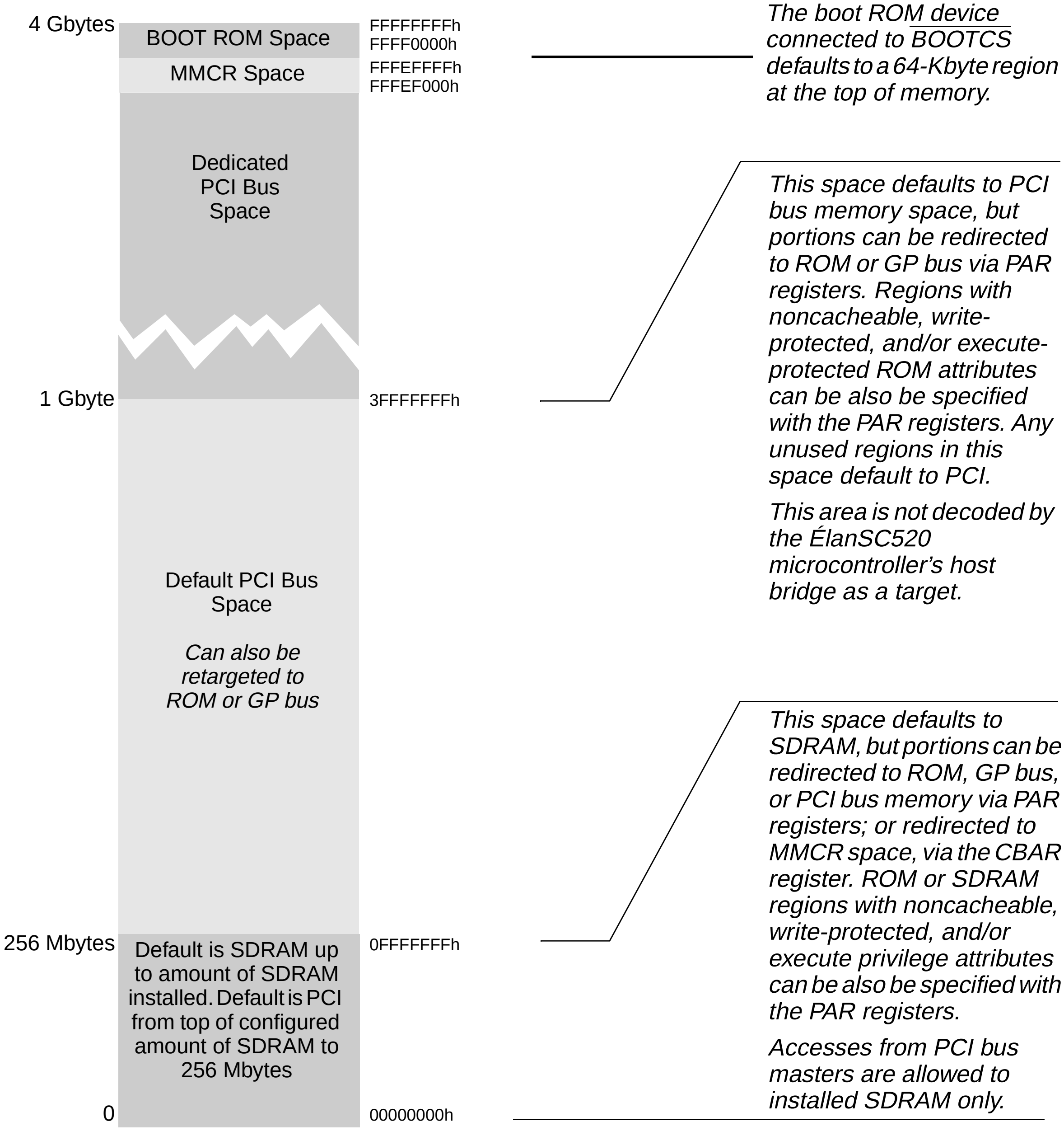
4.3.3 Memory Space

Memory space in the ÉlanSC520 microcontroller includes SDRAM, ROM, PCI bus, GP bus, and the MMCR registers. A system memory map is shown in Figure 4-2.

- The CPU has access to the entire memory space.
- PCI bus masters and the GP bus DMA controller have access to SDRAM space only.

Characteristics of these memory spaces are defined in subsequent sections.

Figure 4-2 System Memory Map



Notes:

The boot ROM device connected to BOOTCS defaults to a 64-Kbyte region at the top of memory.

This space defaults to PCI bus memory space, but portions can be redirected to ROM or GP bus via PAR registers. Regions with noncacheable, write-protected, and/or execute-protected ROM attributes can be also be specified with the PAR registers. Any unused regions in this space default to PCI.

This area is not decoded by the ÉlanSC520 microcontroller's host bridge as a target.

This space defaults to SDRAM, but portions can be redirected to ROM, GP bus, or PCI bus memory via PAR registers; or redirected to MMCR space, via the CBAR register. ROM or SDRAM regions with noncacheable, write-protected, and/or execute privilege attributes can be also be specified with the PAR registers.

Accesses from PCI bus masters are allowed to installed SDRAM only.

4.3.3.1 SDRAM Space

SDRAM space in an ÉlanSC520 microcontroller system defaults to a linear region starting at the lowest 32-bit memory address (00000000h) and ending at the top of SDRAM, which is defined by the amount of SDRAM populated in the system and programmed in the SDRAM controller's configuration registers.

The maximum amount of SDRAM supported in an ÉlanSC520 microcontroller system is 256 Mbytes, in various configurations between one and four physical banks. Once the SDRAM configuration registers are programmed and the individual banks are enabled, SDRAM is immediately accessible.

The ÉlanSC520 microcontroller allows special attributes to be applied to any region within SDRAM space. These attributes are not required for normal operation, however some applications can benefit from their use. Programming PAR registers for SDRAM access is required *only* if special attributes must be applied to specific SDRAM regions, as described below. There are three attributes that can be applied to any SDRAM region:

- Noncacheable regions
- Write-protected regions
- Code execution control

In a typical system configuration, an external PCI bus master has full access to the entire SDRAM region. The address decoding logic in the ÉlanSC520 microcontroller's PCI host bridge automatically claims cycles to this address space on the PCI bus generated by external PCI bus masters and causes them to be directed to SDRAM. PCI bus master cycles that are forwarded to the memory controller always result in an SDRAM cycle, even if a PAR register has been programmed to redirect the address to the GP bus or ROM. Also, if a PCI bus master generates a memory write cycle that is forwarded to the memory controller and a PAR has been programmed to write-protect the region, an SDRAM write cycle will occur with the SDQM signals inactive, the data will be discarded, and the data written into the PCI bridge FIFOs will be purged. The ÉlanSC520 microcontroller can be programmed to generate an interrupt in this case to notify the CPU of such write protection violations, and that a PCI bus master caused the violations. Any data written to the write buffer prior to enabling write-protection will be successfully written to SDRAM.

4.3.3.2 ROM/Flash Space

The ÉlanSC520 microcontroller supports three separate address regions for ROM/Flash, which are selected by the PAR registers. The $\overline{\text{BOOTCS}}$ ROM chip select must be used for the boot device and defaults to a 64-Kbyte linear region at the top of the 4-Gbyte CPU space. During the boot process, the ROM code can configure PAR registers to enable the entire $\overline{\text{BOOTCS}}$ ROM space and redirect it to the desired region. The default 64-Kbyte region is always enabled, however. The PAR register accepts separate TARGET values for each of the three ROM chip select regions ($\overline{\text{BOOTCS}}$, $\overline{\text{ROMCS1}}$, and $\overline{\text{ROMCS2}}$). ROM space is accessible by the CPU only, regardless of PAR register programming.

ROM space is normally cacheable and writes to these regions are allowed (this is useful for Flash devices). However, PAR registers can also be used to enable specific attributes, such as defining noncacheability and write-protected regions.

The ÉlanSC520 microcontroller supports multiple data widths in the ROM array, as well as programmable timing. These characteristics are configured independently of the address space in the ÉlanSC520 microcontroller. See Chapter 12, “ROM/Flash Controller”, for a description of these features and instructions for configuring the ROM chip select timing and data widths.

4.3.3.3 GP Bus Memory Space

GP bus memory space is enabled only through PAR registers and is accessible only by the CPU. There are eight chip selects that can be selected by the PAR registers. Note that the PAR registers do not allow any attributes to be defined in GP bus memory space regions, and GP bus memory space is always noncacheable.

The PAR registers are used to select GP bus space and the specific chip select, but separate configuration registers within the GP bus controller block must be programmed to control the width of the data bus and the timing of the bus. There is no restriction on the mapping of memory address space to GP bus chip selects. For example, if a noncontiguous memory region is required for a specific chip select, then multiple PAR registers can be programmed with the same chip select as the target, but with different address ranges.

Positive address decoding is also supported on the GP bus for devices that perform their own address decoding and therefore do not require a chip select to be generated by the ÉlanSC520 microcontroller. This is accomplished simply by not choosing the corresponding chip select in the pin multiplexing registers when the PAR register is set up (see step 3 in “Programming External Memory, Buses, and Chip Selects” on page 4-4). The address and control signals are still generated on the GP bus.

PCI bus masters are not permitted to access the GP bus in an ÉlanSC520 microcontroller system. If a PCI bus master generates an address in normal SDRAM space that is claimed by the ÉlanSC520 microcontroller, but the region has been redirected to the GP bus via a PAR register, the cycle will still be sent to SDRAM and will be write-protected, regardless of the cycle type, and the resultant data will be discarded.

4.3.3.4 PCI Bus Memory Space

The ÉlanSC520 microcontroller’s address decoding logic automatically defaults all memory space above configured SDRAM to the PCI bus, with the exception of the 4-Kbyte memory-mapped configuration space and the 64-Kbyte boot space. All CPU memory space accesses in this address region are redirected to the PCI bus, and the ÉlanSC520 microcontroller does not claim accesses in this address region that are generated by PCI bus masters. The GP bus DMA controller cannot access this region.

The CPU can allocate space within the lower 1 Gbyte for GP bus or ROM, overlaying and effectively eliminating parts of this PCI bus region. For example, a ROM device could be mapped in memory between the top of SDRAM and 1 Gbyte, a region that would normally default to PCI bus. In this case, only this particular region would be redirected to ROM, but the remaining region within the 4-Gbyte space would continue to be directed to the PCI bus.

Some system applications may require a region below the top of SDRAM to be redirected to the PCI bus. An example of this is a PCI bus video card mapped to the 000A0000h-000BFFFFh region in a PC/AT application. In this case, a PAR register must be used to redirect the address from the CPU to the PCI bus instead of the SDRAM. Note that only PAR 0 or PAR 1 can be used to select PCI as a target.

Note: *If a PAR window is configured for PCI, AND the CBAR register is programmed to overlap with this PAR window, AND the PAR window is placed below the top of DRAM, the MMCR is not given priority over the PCI access. This configuration could result in system errors due to concurrence of both PCI and internal MMCR accesses.*

4.3.3.5 Memory-Mapped Configuration Region (MMCR) Registers Space

All integrated peripherals and configuration registers in the ÉlanSC520 microcontroller that are not defined as PCI bus configuration space, PC/AT peripheral configuration registers, or the Configuration Base Address (CBAR) register are memory-mapped in the ÉlanSC520 microcontroller. These registers are accessed in a 4-Kbyte region near the top of CPU address space at location FFFEF000h after reset, but can be additionally aliased to any 4-Kbyte boundary within the first 1-Gbyte of memory space (between 00000000h and 3FFFFFFFh) by performing an I/O write to the Configuration Base Address (CBAR) register. MMCR register space has a higher priority than the Programmable Address Region (PAR) registers.

See Section 4.3.4.1 for details on programming the CBAR register.

Reading unimplemented registers in this 4-Kbyte region returns indeterminate data values. Writing to unimplemented registers in this region has no effect.

Note: *If a PAR window is configured for PCI, AND the CBAR register is programmed to overlap with this PAR window, AND the PAR window is placed below the top of DRAM, the MMCR is not given priority over the PCI access. This configuration could result in system errors due to concurrence of both PCI and internal MMCR accesses.*

4.3.3.5.1 Integrated Memory-Mapped Peripherals

The ÉlanSC520 microcontroller's non-PC/AT integrated peripherals are located within the MMCR region, instead of being I/O mapped as are the integrated PC/AT peripherals. The peripherals located in the memory-mapped configuration region include:

- Am5$_X$86 CPU extension registers
- SDRAM controller and SDRAM buffering
- ROM controller
- PCI host bridge
- System arbitration
- Memory and I/O space control
- GP bus controller
- PIO, pin multiplexing and clock control
- Software timer
- General-purpose timers 0, 1 and 2
- Watchdog timer
- Synchronous serial interface (SSI)
- Feature enhancements to PC/AT-compatible peripherals
  - Programmable interval timer (PIT) extension registers in the programmable input/output (PIO) and programmable interrupt controller (PIC) blocks
  - UART extensions
  - Programmable interrupt controller (PIC) extensions
  - Reset control
  - GP-DMA controller extensions

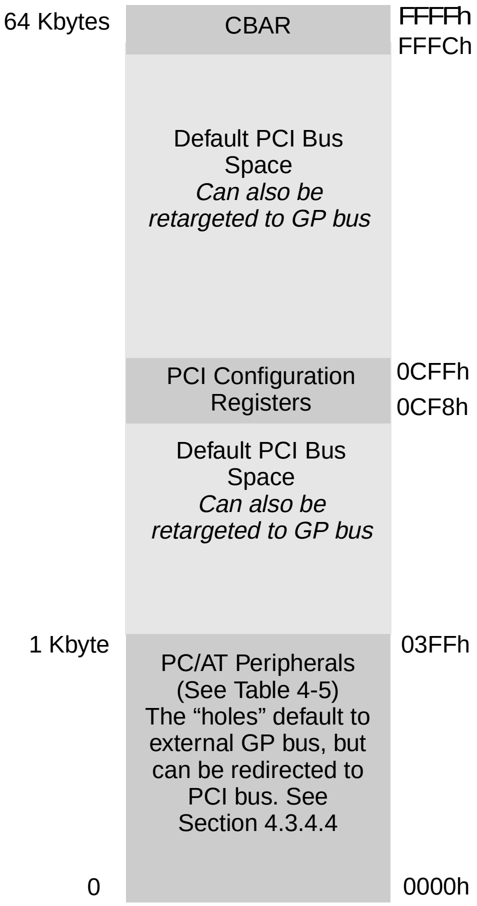
4.3.4 I/O Space

The ÉlanSC520 microcontroller’s I/O space is partitioned into five regions:

- Configuration Base Address (CBAR) register
- PCI bus configuration space
- External PCI bus I/O devices
- Integrated PC/AT-compatible peripherals
- External GP bus I/O devices

Figure 4-3 shows the system I/O address space map for the ÉlanSC520 microcontroller. Each of the regions is described in the following sections.

Figure 4-3 System I/O Map


64 Kbytes
CBAR
FFFFh
FFFCh
Default PCI Bus Space
Can also be retargeted to GP bus
PCI Configuration Registers
0CFFh
0CF8h
Default PCI Bus Space
Can also be retargeted to GP bus
1 Kbyte
03FFh
PC/AT Peripherals (See Table 4-5) The “holes” default to external GP bus, but can be redirected to PCI bus. See Section 4.3.4.4
0
0000h

4.3.4.1 Configuration Base Address (CBAR) Register

The Configuration Base Address (CBAR) register (Port FFFCh) is a 32-bit register that is used to relocate the integrated memory-mapped peripherals and MMCR registers, thus allowing a more flexible system memory map. The CBAR is fixed in I/O space at FFFCh and is “keyed” to prevent accidental programming.

The CBAR allows an alias of the memory-mapped configuration registers (MMCR) to be aliased anywhere in the first 1 Gbyte of address space on a 4-Kbyte boundary. The MMCR is always available in the memory space directly below the boot ROM space at FFFEF000h, but the CBAR can be programmed to optionally allow a copy of this region anywhere in the lower 1-Gbyte space (on a 4-Kbyte boundary).

4.3.4.2 PCI Configuration Space

PCI Local Bus Specification, Revision 2.2, defines an indirect-mapped configuration space that occupies only eight bytes in I/O space from 0CF8–0CFFh, and this mechanism is supported in the ÉlanSC520 microcontroller. The PCI bus configuration scheme uses two 32-bit I/O locations:

- PCI Configuration Address (PCICFGADR) register (Port 0CF8h) is the *address* register where the actual address of the device's register and the bus number is located.
- PCI Configuration Data (PCICFGDATA) register (Port 0CFCh) is the *data* register where the data of the specific register is written to or read from.

This PCI configuration space is accessible *only* by the CPU in the ÉlanSC520 microcontroller, and the I/O cycle is claimed by the PCI bus configuration register block.

As a target, the ÉlanSC520 microcontroller does not accept any PCI bus configuration space accesses from other PCI bus masters.

Host-bridge-specific PCI configuration registers are described in the *Élan™SC520 Microcontroller Register Set Manual*, order #22005. See also the *PCI Local Bus Specification,* Revision 2.2, for details on PCI bus device configuration register programming.

4.3.4.3 PCI I/O Space

The CPU's I/O cycles can be directed to the PCI bus for normal direct-mapped access of devices, with the following restrictions:

- I/O addresses claimed by the integrated PC/AT peripherals and the CBAR cannot be forwarded to the PCI bus under any conditions. See the I/O map in Figure 4-3 on page 4-11 and Table 4-5 on page 4-14 for details of the I/O addresses that are claimed by the integrated peripherals.
- By default, the "holes" in this portion of the I/O address space (0000–03FFh) are forwarded to the external GP bus. The Address Decode Control (ADDDECCTL) register (MMCR offset 80h) can be configured to forward accesses to these holes to the PCI bus. A PAR register is not required for this.
- I/O addresses implemented by PCI bus configuration space (0CFC–0CFFh) are only forwarded to the PCI bus as an I/O cycle when the ENABLE bit in the PCI Configuration Address (PCICFGADR) register is cleared to 0. Otherwise, they are forwarded as a PCI configuration cycle. Ports 0CF8–0CFBh are forwarded to the PCI bus as I/O transactions only for non-doubleword accesses to this region; otherwise, they are claimed by the host bridge as a PCI configuration cycle.

All other CPU I/O cycles are, by default, forwarded to the PCI bus as normal PCI I/O transactions. PAR registers can be enabled to direct portions of this region to the GP bus.

As a target, the ÉlanSC520 microcontroller does not accept any I/O space accesses from PCI bus masters.

4.3.4.4 PC/AT-Compatible I/O Peripherals Region

The ÉlanSC520 microcontroller includes several integrated peripheral cores that are PC/AT compatible, including the DMA controller, programmable interrupt controller (PIC), programmable interval timer (PIT), UARTs, real-time clock, and various control/status registers. These I/O addresses are automatically decoded by the ÉlanSC520 microcontroller's address decoding logic and require no special setup or PAR registers. Table 4-5 summarizes the I/O map for these integrated peripherals.

There are *holes* in this region, which are I/O transactions in the lower 1-Kbyte region that not claimed by the ÉlanSC520 microcontroller's internal peripherals. These addresses can be decoded externally, or, if a chip select is required, a PAR register can be programmed for these addresses.

- By default, all of the accesses to holes in this portion of the I/O address space (0000h to 03FFh) are forwarded to the external GP bus.
- To forward all accesses to the PCI bus, the IO_HOLE_DEST bit in the Address Decode Control (ADDDECCTL) register (MMCR offset 80h) can be set.
- If necessary, PARx registers can be used to override sending accesses to the PCI bus on an individual peripheral basis. In this way, accesses for individual peripherals can be directed back to the external GP bus.

For example, some PCI cards (notably VGA cards) use legacy I/O locations. The IO_HOLE_DEST bit allows the holes to be directed to either the PCI or to the GP bus. For a system requiring legacy GP bus peripherals along with legacy PCI peripherals (for instance, a PCI VGA card and a GP bus keyboard controller), the IO_HOLE_DEST bit would be set to 1 to direct all accesses to the PCI bus. The legacy GP bus keyboard controller would then be configured via PAR registers to override this setting. See "VGA Controller on the PCI Bus" on page 3-15 for another discussion of this topic.

Note: *If a PARx register is configured to address GP bus I/O space within a hole, accesses in the defined region are forwarded to the GP bus regardless of the IO_HOLE_DEST bit value. It is the programmer's responsibility to ensure that external peripherals are not mapped over any of the ÉlanSC520 microcontroller's internal peripherals. Normal operation is not guaranteed in this case. See "Disabling Internal Peripherals" on page 3-21 for more information about this topic.*

Table 4-5 PC/AT Peripherals I/O Map

Peripheral Core	I/O Address Range
Slave GP-DMA Controller	0000–000Fh
Master Interrupt Controller	0020–0021h
Slave 2 Interrupt Controller • This controller is not defined in standard PC/AT architecture, but has been included in the ÉlanSC520 microcontroller to provide additional interrupt request sources	0024–0025h
Programmable Interval Timer (PIT)	0040–0043h
Keyboard Control A20M and Fast Reset (SCP) • Accesses to these locations are always directed to the external GP bus, but are also snooped internally for PC/AT functions.	0060h, 0064h
System Control Port B/NMI Status • Reads and writes to this location are directed to this register only and are not seen on the external GP bus	0061h
Real-Time Clock (RTC) Index/Data	0070h, 0071h
General-Purpose Scratch Registers • These are unused locations from the original DMA Page Register file and are maintained for PC/AT compatibility. Writes to these locations update the corresponding register and are also seen on the external GP bus. Reads to the locations return the data from the corresponding register, but do not initiate a cycle on the external GP bus.	0080h 0084–0086h 0088h 008C–008Eh
General-Purpose Scratch Register • This is an unused location from the original DMA Page Register file and is maintained for PC/AT compatibility. Reads and writes to this location are directed to this register only and are not seen on the external GP bus.	008Fh
GP-DMA Page Registers • Reads and writes to these locations are directed to these registers only and are not seen on the external GP bus.	0081–0083h 0087h 0089h-008Bh
System Control Port A	0092h
Slave 1 Interrupt Controller	00A0–00A1h
Master GP Bus DMA Controller	00C0–00DEh (even addresses only)
Floating Point Error Interrupt Clear	00F0h
UART 2	02F8–02FFh
UART 1	03F8–03FFh

The ÉlanSC520 microcontroller also allows the internal UARTs and the real-time clock (RTC) to be disabled, for applications when an external device is preferred. This is controlled by configuration register bits in the Address Decode Control (ADDDECCTL) register (MMCR offset 80h). When these peripherals are disabled, the I/O cycle is forwarded externally to the GP bus. This allows connection of external devices such as a standard Super I/O chip.

Integrated PC/AT peripherals are not accessible by PCI bus masters.

4.3.4.5 GP Bus I/O Region

The PAR registers must be used to address external I/O devices on the GP bus. GP bus addressing is implemented with byte granularity, to accommodate devices with very few registers and very fragmented I/O maps that are typically found in PC/AT-compatible systems.

When programming PAR registers for GP bus I/O space, it is best to configure the space on doubleword boundaries. Note that when specifying unaligned byte regions for I/O access, the software that accesses the regions must directly address the correct byte or bytes. For example, if a PAR is programmed with an I/O region, and the start address is xxx1h (i.e., byte #1), when the CPU performs a word or doubleword access starting at xxx0h (i.e., byte #0), the entire doubleword access is redirected to the PCI bus (byte #1 will not be accessed on the GP bus as programmed). In this case, the byte requested *must* be directly accessed by the CPU at I/O address xxx1h.

This region is not accessible by PCI bus masters.

4.3.5 Configuration Information

4.3.5.1 Configuring ROM/Flash Space

There are three ROM address regions that can be defined in the ÉlanSC520 microcontroller, but only the $\overline{\text{BOOTCS}}$ region is absolutely required for system boot up from reset. The optional two regions, $\overline{\text{ROMCS1}}$ and $\overline{\text{ROMCS2}}$ are configured via PAR registers. $\overline{\text{BOOTCS}}$ configuration is described in Chapter 3, “System Initialization”. See “Programmable Address Region (PAR) Registers” on page 4-5 for details on PAR register programming.

4.3.5.2 Configuring SDRAM Address Space

SDRAM space is determined at boot time when the SDRAM controller’s configuration registers are programmed and individual banks are enabled. A typical design can perform an SDRAM sizing routine to determine the amount of memory installed in the system and write the appropriate values to the configuration registers. For example, in a system that contains 16 megabytes of SDRAM, initialization software defines the SDRAM address region from 00000000–00FFFFFFh, and all accesses to this region are forwarded to the SDRAM controller unless a PAR register has been programmed to overlay the region with MMCR, ROM, PCI bus, or GP bus space.

4.3.5.2.1 Noncacheable, Write-Protected, or Nonexecutable SDRAM Regions

In the default condition, the entire SDRAM region is cacheable and executable by the CPU, and read/writable by the CPU, PCI bus master, and GP bus DMA controller cycles. There may be some system configurations in which specific portions of SDRAM require restricted access which can be accomplished by enabling specific attributes. A few common examples follow:

- An SDRAM region that contains only code can be marked as write-protected with an attribute in the PAR register. This prevents the CPU and any bus master from illegally writing over the code in SDRAM due to faulty programming. In addition, an interrupt can be generated to the CPU when a violation occurs to assist in debug of the illegal write condition.
- An SDRAM region that contains only data can be marked as nonexecutable with an attribute in the PAR register. If a software task attempts to branch to that location and resume execution due to a software bug, the CPU will read an illegal opcode, forcing an exception. The exception handler will then facilitate debugging the program that caused the illegal condition.

4.3.5.3 Configuring GP Bus Peripheral Space

Configuring space for GP bus peripherals is accomplished via PAR register programming. This section describes a few system configuration examples beyond the normal programming of chip select regions.

4.3.5.3.1 Configuring a Chip Select for Noncontiguous Memory or I/O Space

Some peripheral subsystems may require a *single* chip select that must be asserted in noncontiguous address locations. For example, an I/O device can contain multiple integrated functions that are each addressed at separate, noncontiguous I/O addresses (such as a custom ASIC). In this case multiple PAR registers can be used to define each individual address region, but all can be mapped to the same chip select by programming the TARGET field to GP bus and the ATTR field to the same chip select. This is most useful when working with a highly fragmented I/O map such as defined in PC/AT systems, where there is little unused I/O address space.

This can also be accomplished by programming a *single* PAR register to cover the entire range of addresses, which results in some wasted address space.

4.3.5.3.2 Positive Decoding Example

Some peripherals connected to the GP bus may perform their own address decoding from the GP bus addresses and do not require a chip select. In this case, the same steps are followed for programming the configuration registers, but the pin multiplexing registers do not need to be programmed to allow the actual chip select to be driven on a pin, thus allowing the pin to be used for other functions.

If multiple positive decoding regions are required in an application, the PAR registers for each reason can be programmed to map to the same unused chip select, to conserve pin functions.

4.3.5.3.3 Configuring the Élan™SC520 Microcontroller to Use an External Super I/O Chip

It may be desirable to connect a commercially available Super I/O chip on the GP bus in an ÉlanSC520 microcontroller system (for example, systems requiring a keyboard or IDE drive can implement this device).

In this case, since the Super I/O implements two UARTs programmed at the same address as the ÉlanSC520 microcontroller’s integrated UARTs, the internal UARTs can be disabled to support the COM1 and COM2 ports in the Super I/O chip, if desired. In this case, when the CPU performs I/O accesses to the UART address regions, the cycles will be forwarded out to the external GP bus. Also, the Super I/O is a positive decoding device, i.e., it does not require a chip select because it performs the address decoding from the GP bus addresses.

The I/O map for the Super I/O device is fragmented and may require the use of multiple PAR registers for noncontiguous addressing, as described in Section 4.3.5.3.1. If the fragmented I/O space unused by the Super I/O chip is not required elsewhere in the system, then a single PAR register can be used to map the entire range of peripherals. In this case, the UART address spaces would be the highest used I/O space internally in the ÉlanSC520 microcontroller, so the Super I/O peripherals would not be in conflict, allowing a single PAR register to define the entire range of Super I/O peripherals from 01F0–07BEh.

See “Interfacing with a Super I/O Controller” on page 13-13, for an example of connecting the Super I/O chip to the ÉlanSC520 microcontroller’s GP bus.

4.3.5.4 Configuring the Élan™SC520 Microcontroller for Windows® Compatibility

The ÉlanSC520 microcontroller can be configured to operate as a Windows compatible microcontroller. This section describes some of the steps that may be required to configure the memory and I/O addressing; however, this will vary depending on the requirements of the system.

4.3.5.4.1 Memory Regions Above DOS 640-Kbyte Application Space

The ÉlanSC520 microcontroller can be programmed to accommodate the legacy PC/AT-compatible region above the DOS 640-Kbyte application space at 000A0000h area ending at 000FFFFFh (1 Mbyte). This space defaults to SDRAM once the SDRAM banks are enabled, but the PAR registers can be programmed to support the various requirements of systems requiring Windows compatibility. The list below outlines some of the steps to consider when building a memory map in the ÉlanSC520 microcontroller system for such compatibility.

- Two 64-Kbyte video regions from 000A0000–000AFFFFh and 000B0000–000BFFFFh default to SDRAM, but can be enabled as PCI bus space for PC/AT compatible video cards on the PCI bus, via one of the PAR registers. The ÉlanSC520 microcontroller's PCI bus host bridge (as a target) will automatically ignore accesses in this space when either PAR 0 or PAR 1 are programmed to overlay SDRAM regions with the PCI bus.
- The remaining area from 000C0000–000FFFFFh is normally sub-divided in a PC/AT system into several different address regions for BIOS, and accesses to these regions can be redirected to either ROM, the GP bus, or the PCI bus by programming PAR registers. Most systems will not require the use of all BIOS regions defined, since many are for expansion ROMs intended for various plug-in cards (such as network interface cards). The following regions are normally defined:
  - One BIOS region with 64-Kbyte granularity from 000F0000–000FFFFFh
  - Four extended system BIOS regions, each with 16-Kbyte granularity from 000E00000–000EFFFFFh
  - 8 Expansion ROM regions, each with 16-Kbyte granularity, from 000C0000–000DFFFFFh

4.3.5.4.2 Integrated Peripheral Mapping

Because the ÉlanSC520 microcontroller already provides standard PC/AT-compatible peripherals that use direct I/O address mapping, there are no I/O address conflicts with these devices. See Table 4-5 on page 4-14 for a summary of this I/O map.

The Configuration Base Address (CBAR) register (Port FFFCh) can be used to alias the internal memory-mapped registers and peripherals to a convenient location. For example, they could be mapped between 640 Kbytes and 1 Mbyte for real mode operation. The memory-mapped configuration region is always available in the upper CPU space (4 Gbytes), but the aliased location is only accessible when the CBAR is programmed and the ENABLE bit has been set.

4.3.5.4.3 DMA Channel and Interrupt Request Steering

The ÉlanSC520 microcontroller provides a method to route interrupt request sources and DMA request pins to the appropriate channels on the programmable interrupt controller (PIC) and the GP-DMA controller, respectively.

See Chapter 15, “Programmable Interrupt Controller”, for further information on interrupt request routing.

See Chapter 14, “GP Bus DMA Controller”, for further information on DMA request routing.

4.3.5.5 Configuring PCI Bus Devices

PCI bus device configuration is accomplished in the ÉlanSC520 microcontroller with the standard PCI Configuration Mechanism #1, as defined in the *PCI Local Bus Specification,* Revision 2.1. This configuration requires an indirect mapped I/O scheme in which the address of the device is written to the PCI Configuration Address (PCICFGADR) register (Port 0CF8h), and the data is accessed via the PCI Configuration Data (PCICFGDATA) register (Port 0CFCh). See“Configuration Information” on page 9-9 for more information. See also the *PCI Local Bus Specification,* Revision 2.2.

4.3.6 Interrupts

The ÉlanSC520 microcontroller can be programmed to generate an interrupt request when a write protection violation occurs, providing software with a debugging mechanism to determine which task illegally attempted to write to the memory region marked with this attribute. In this case, an interrupt request is generated to the programmable interrupt controller (PIC) block, where the request is routed to the appropriate type of interrupt (maskable or non-maskable) and level, based on the programming of the configuration registers. The PAR window that contains the address region where the write protect violation occurred is latched into a register, as well as which bus owner caused the violation (CPU, GP-DMA controller, or PCI bus master).

See Chapter 15, “Programmable Interrupt Controller”, for details of PIC programming.

4.3.7 Software Considerations

Since the ÉlanSC520 microcontroller provides some flexibility in defining the system memory and I/O map, there are a number of software considerations that must be analyzed. The list below describes some of the issues that must be considered when programming the configuration registers to define the memory and I/O space in an ÉlanSC520 microcontroller system.

- The Configuration Base Address (CBAR) register must be accessed as a 32-bit I/O register to guarantee that all bits are written at the same time. The MATCH field of the CBAR must be written with the correct pattern to enable *or* disable the MMCR alias.
- MMCR register space has higher priority than the Programmable Address Region (PAR) registers.
- The PAR registers are organized such that the lowest register (PAR 0) is the highest priority and the last PAR register (PAR15) is lowest priority. Therefore, if two PAR registers are overlaid due to programming, the lowest numbered PAR takes priority.
- PAR registers should not be programmed to conflict with any of the fixed I/O regions, such as the Configuration Base Address (CBAR) register or the PCI bus configuration space.The ÉlanSC520 microcontroller’s address decoding does not permit PAR registers to overlay the integrated PC/AT peripherals.
- In general, the PAR register start address and region size should not be programmed to conflict with each other. It is possible to program the PAR registers such that the region size is greater than the start address allows. For example, if the region size is defined as 64 Kbytes, but the start address is programmed to be the top of the 1-Gbyte region (maximum address allowed by PAR registers) minus 4 Kbytes, then the address space available will be the 4-Kbyte region starting at the start address.
    - Subsequent access past the 1-Gbyte boundary will still be to the PCI bus
    - The remaining 60-Kbyte region will *not* qualify as a PAR hit.

- When programming the PAR registers for an SDRAM region, the PAR register start address and region size should not conflict with the programmed value that defines the top of SDRAM in the system. For example, if a PAR is setup for SDRAM and the region size is defined as 8 Kbytes, but the start address is programmed to be the top of the SDRAM minus 4 Kbytes, then addresses above the top of SDRAM will *not* result in a hit for this PAR.
- If the TARGET field of any PAR register is defined as SDRAM, but no SDRAM has been enabled via the SDRAM controller configuration registers, the memory space defaults to the PCI bus.
- Systems that configure another memory space resource to be overlaid on top of SDRAM space do not have access to the SDRAM that was overlaid, since address translation is not supported in the ÉlanSC520 microcontroller. For example, if a PCI bus video card is used in the 000A0000–000AFFFFh region (as in typical PC/AT operation), the system will lose the 64 Kbytes of SDRAM in that region as long as the PAR register is enabled.
- Any region that is overlaid on default SDRAM space through a PAR register or CBAR takes priority over the SDRAM region in the decoding block. In effect, a portion of SDRAM becomes inaccessible when this is done. If a PCI bus master generates an address to this overlaid address region, the cycles will be forwarded to SDRAM and will be write-protected.
- Code execution from memory on the GP bus or the PCI bus is discouraged (after boot code has executed), since accesses to these spaces are not cacheable and may result in unacceptable latencies under some conditions. Code execution is more efficient when executing from SDRAM or from ROM devices that use $\overline{\text{BOOTCS}}$, $\overline{\text{ROMCS1}}$, or $\overline{\text{ROMCS2}}$, because accesses to these resources are cacheable.
- The ÉlanSC520 microcontroller guarantees coherency with SDRAM buffers that are shared between the CPU and other bus masters, but it may be beneficial to mark these regions as noncacheable to avoid the overhead with cache write-backs upon every access by the bus master. This can be accomplished by programming a PAR register and setting the noncacheable attribute. Cache snooping will continue; however, the performance impact is negligible, since there will be no write-back cycles.
- Care must be taken when programming configuration registers that affect address decoding during normal system operation when either PCI bus master or GP bus DMA activity is occurring.
  - When writing to PAR registers, verify that the ÉlanSC520 microcontroller's PCI host bridge target FIFOs have been flushed and disable PCI bus master access of SDRAM to prevent unexpected forwarding of accesses from other masters. An example of a potential problem is modifying a PAR register to redirect normal SDRAM region accesses to the PCI bus, while a PCI bus master has already been granted the PCI bus. In this case, when the CPU completes the write to the PAR register, the posted PCI bus master access is forwarded to the SDRAM controller because the bus was already granted to the PCI bus master. This problem can be alleviated by disabling PCI bus master access to SDRAM (the default mode after reset) via the System Arbiter Master Enable (SYSARBMENB) register (MMCR offset 72h), and performing a read from an external PCI agent to flush the ÉlanSC520 microcontroller's target FIFOs, before writing to configuration registers that affect address decoding.
  - The CPU cache should always be flushed after the cacheability attribute is changed from cacheable to noncacheable for any memory region (by programming the PAR register), or when the cache write policy is changed from write-back to write-through.

- Programming the PAR register maximum region size and a page size of 64 Kbytes allows a space up to 128 Mbytes to be defined; however, the GP bus/ROM address pins support a maximum of 64 Mbytes per chip select. If a 128-Mbyte space is programmed for a GP bus or ROM chip select, the upper 64 Mbytes will be aliased with the lower 64-Mbyte region.
- When programming PAR registers for GP bus I/O space, it is best to configure the space on doubleword boundaries. Note that when specifying unaligned byte regions for I/O access, the software that accesses the regions must directly address the correct byte or bytes. For example, if a PAR is programmed with an I/O region, and the start address is xxx1h (i.e., byte 1), when the CPU performs a word or doubleword access starting at xxx0h (i.e., byte 0), the entire doubleword access is redirected to the PCI bus (byte 1 will not be accessed on the GP bus as programmed). In this case the byte requested *must* be directly accessed by the CPU at I/O address xxx1h.
- A write-protection violation occurs when the CPU, any PCI bus master, or the GP-DMA controller attempts to write to any memory region that has been marked as write-protected by a PAR register attribute. When this occurs, the cycle is always forwarded to SDRAM as a write cycle with the SDQM signals inactive, and the original data is discarded. Any data that was written to the write buffer prior to enabling write-protection is successfully written to SDRAM.
- Software must include proper interrupt service routines and exception handlers when enabling write-protection violation interrupts and nonexecutable region attributes in the Address Decode Control (ADDDECCTL) register (MMCR offset 80h). Note that in the case of the write protection violation, the PAR register number that contains the address region of the violation is latched in the WPV_WINDOW bit field in the Write-Protect Violation Status (WPVSTA) register (MMCR offset 82h) and retained until it is cleared by software. The PARx window number is latched when a write-protect violation occurs. Subsequent write-protect violations are not captured until software clears the interrupt by writing a 1 to the WPV_STAT bit in the same register.
- If two or more PAR registers are overlapping (programmed to have some address range in common), the write-protection exception is generated only if the higher priority PAR has the attribute enabled. If the lower priority PAR has the write-protect attribute enabled but the higher priority PAR has it disabled, then writes into the common address range shared by the two PAR registers will *not* generate an exception. This discussion applies to the cacheability control and code execution attributes, as well.
- Access of ÉlanSC520 microcontroller internal configuration registers:
  - All integrated PC/AT peripherals mapped to I/O space must be accessed only as 8 bits unless otherwise specified.
  - All memory-mapped integrated peripherals and configuration registers for PC/AT peripherals must be accessed as specified in the *Élan™SC520 Microcontroller Register Set Manual*, order #22005.
  - PCI configuration registers should be accessed as 32 bits unless otherwise specified in the *Élan™SC520 Microcontroller Register Set Manual*, order #22005.

4.4 INITIALIZATION

The ÉlanSC520 microcontroller’s address decoding is cleared to the default state by a system reset.

- The $\overline{\text{BOOTCS}}$ decoding is enabled for the 64-Kbyte region from FFFF0000–FFFFFFFFh
- SDRAM address space is disabled.
- All PAR registers are disabled and cleared to zeros, which means there are no external GP bus address spaces enabled. Note that I/O holes below 1 Kbyte will be directed to the external GP bus. However, no chip selects are enabled, and positive decodes would be required.
- Integrated PC/AT peripheral I/O space is enabled as defined in Table 4-5 on page 4-14.
- The Configuration Base Address (CBAR) register is addressed in I/O space at FFFCh. Memory-mapped configuration register space is enabled at FFFEF000–FFFEFFFFh (below CPU boot space address).
- The PCI bus is disabled, and the configuration registers are defaulted to the values specified in *PCI Local Bus Specification,* Revision 2.2. PCI configuration space is enabled in I/O space at ports 0CF8h and 0CFCh (PCICFGADR and PCICFGDATA).

See “Programmable Address Region (PAR) Registers” on page 4-5 for information on configuring these registers. See “Configuration Information” on page 4-15 for additional detail on configuring the various address spaces included on the ÉlanSC520 microcontroller.

CHAPTER

5 CLOCK GENERATION AND CONTROL

5.1 OVERVIEW

The ÉlanSC520 microcontroller is designed to generate all of the internal and system clocks it requires. The ÉlanSC520 microcontroller includes on-chip oscillators and PLLs, as well as most of the required PLL loop filter components.

The ÉlanSC520 microcontroller requires two standard crystals, one for 32.768 kHz and one for 33 MHz. All the clocks required inside the ÉlanSC520 microcontroller are generated from these crystals. Output clock pins are provided for selected clocks, providing up to 24 mA of sink or source current.

The ÉlanSC520 microcontroller also supplies the clocks for SDRAM and the PCI host bridge; however external clock buffering may be required in some systems.

The clocking generation and control features include:

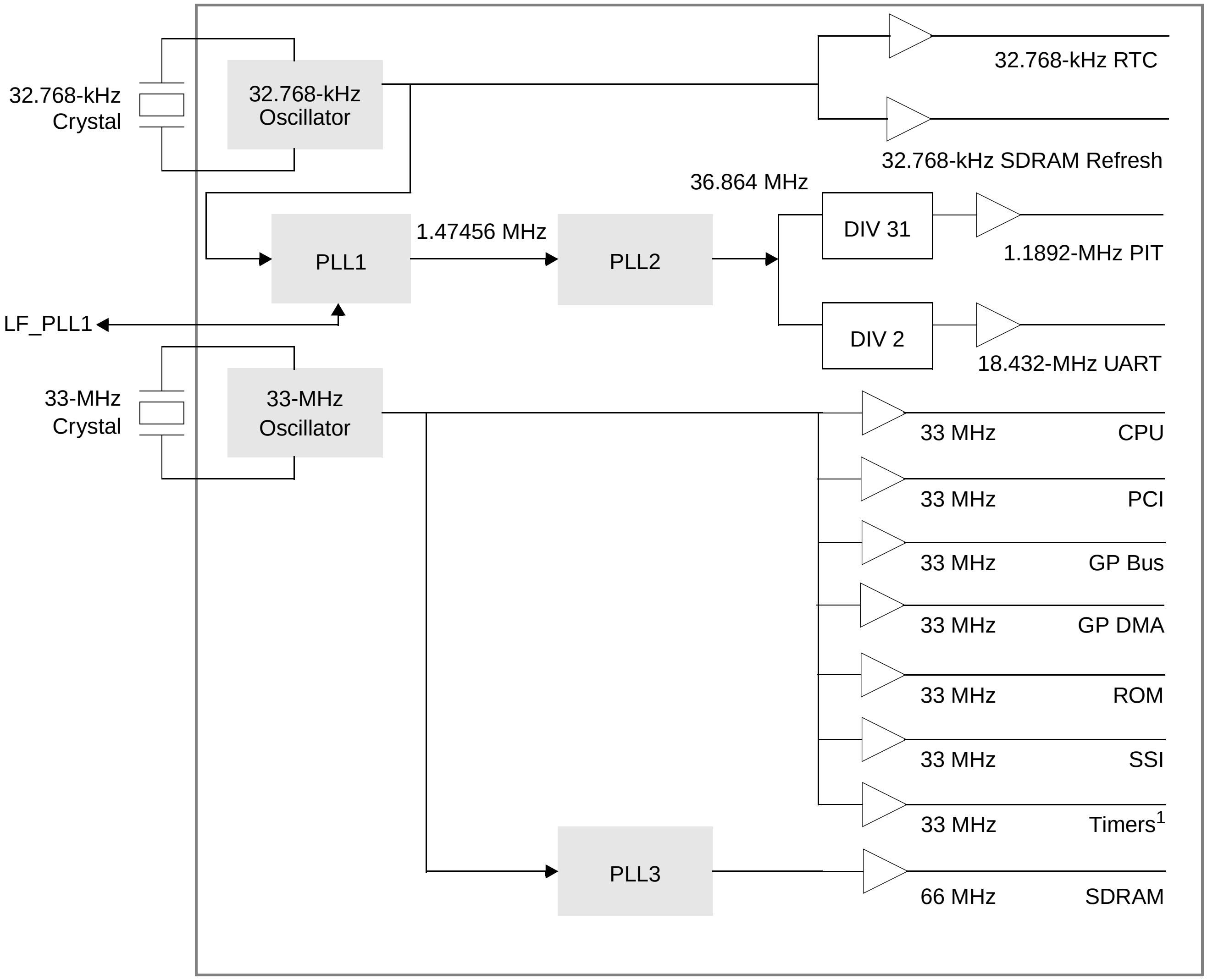
- RTC low-current oscillator using standard off-the-shelf 32.768-kHz crystal
- 33-MHz oscillator using standard off-the-shelf 33-MHz crystal (33.000 or 33.333 MHz)
- 33-MHz clock provides clocks for the integrated Am5$_x$86 CPU and external PCI bus
- Integrated 66-MHz PLL provides clocks for external SDRAM
- Integrated PLLs for generating 1.1892-MHz PIT clock and 18.432-MHz UART clock
- Integrated on-chip PLL loop filters for the 66-MHz and 36.864-MHz PLLs, eliminating the need for external capacitors
- 33.333-MHz/30.000-MHz PCI Clock Output Pin, CLKPCIOUT
- 66-MHz SDRAM Clock Output Pin, CLKMEMOUT
- 33-MHz and 32.768-kHz oscillators bypass option

Note: *The ÉlanSC520 microcontroller supports either a 33.000-MHz or 33.333-MHz crystal. In this document, the term “33 MHz” refers to the system clock derived from whichever 33-MHz crystal frequency is being used in the system.*

5.2 BLOCK DIAGRAM

Figure 5-1 shows a block diagram of the ÉlanSC520 microcontroller's internal clocks. Table 5-1 shows PLL lock times and oscillator start-up times. See the *Élan™SC520 Microcontroller Data Sheet*, order #22003, for timing diagrams and additional clocking specifications.

Figure 5-1 Clock Source Block Diagram



Notes:

1. Includes the programmable interval timer (PIT), general-purpose timers, watchdog timer, and the software timer.

Table 5-1 Clock Start-up and Lock Times

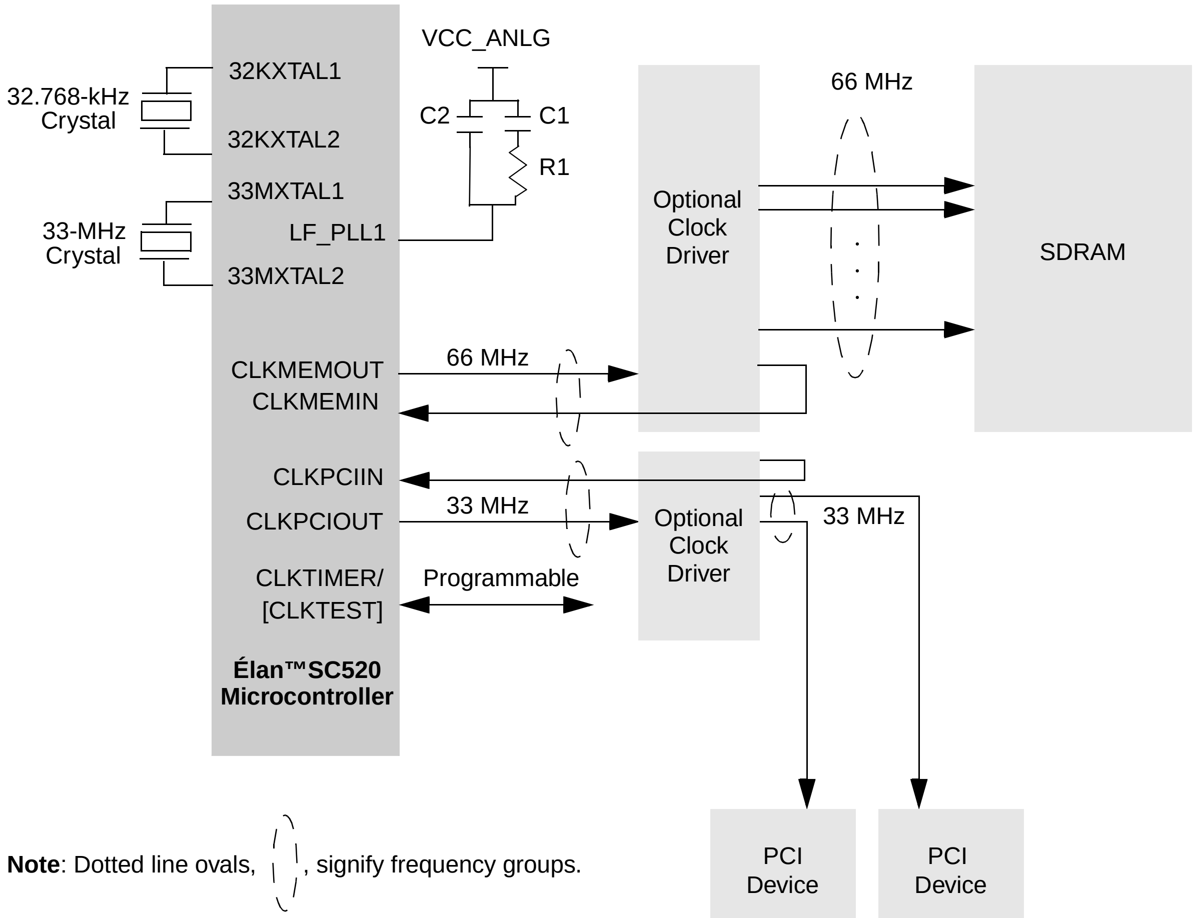
Clock Source	Max
32.768-kHz Oscillator	1 s
33-MHz Oscillator	10 ms
PLL1 (1.47456 MHz)	10 ms
PLL2 (36.864 MHz)	100 µs
PLL3 (66 MHz)	50 µs

5.3 SYSTEM DESIGN

Figure 5-2 shows a system block diagram of the ÉlanSC520 microcontroller's external clocks. As shown in Figure 5-2, external clock drivers may be necessary when the system presents a large capacitive load.

Table 5-2 lists the shared clock signals of the ÉlanSC520 microcontroller.

Figure 5-2 System Clock Distribution Block Diagram



Note: Dotted line ovals, signify frequency groups.

Table 5-2 Clock Signals Shared with Other Interfaces

Default Function	Alternate Function	Control
CLKTIMER	CLKTEST	CLK_PIN_DIR bit in Clock Select (CLKSEL) register (MMCR offset C26h)

5.3.1 Clock Pin Loading

Clock pins are designed to either source or sink 24 mA. The maximum amount of capacitive load that can be placed on a clock pin is determined by the required rise/fall times.

Use the following equation to determine the maximum capacitive loading.

$$C = I/(dV/dt)$$

where:

I = Current
dV = Voltage change
dt = Time change

As an example, suppose that the system requires a rise/fall time of 1 ns, with a voltage swing of 2.5 V. Then, the maximum capacitive load is:

$$C_{max} = 24\ \text{mA}/(2.5\ \text{V}/1\ \text{ns}) = 9.6\ \text{pF}$$

Derating curves for the device are provided in the *Élan™SC520 Microcontroller Data Sheet*, order #22003.

5.3.2 Selecting a Crystal

The accuracy of the real-time clock (RTC) depends on several factors relating to crystal selection and board design. A clock timing budget determines the clock accuracy. The designer should determine the timing budget before selecting a crystal.

There are four major contributors to a clock timing budget.

- **Frequency Tolerance**—This is the crystal calibration frequency. It states how far off the actual crystal frequency is from the nominal frequency. For a typical 32.768-kHz crystal (watch crystal), the frequency tolerance is ± 20 parts per million (ppm). Frequency tolerance is specified at room temperature.
- **Frequency Stability**—This parameter is a measure of how much the crystal resonant frequency is influenced by operating temperature. For watch crystals, typical numbers are around –30 ppm over the temperature range.
- **Aging**—This parameter is how much the crystal resonant frequency changes with time. Typical aging numbers are ± 3 ppm per year.
- **Load Capacitance**—The crystal is calibrated with a specific load capacitance. If the system load capacitance does not equal the crystal load capacitance, a timing error is introduced. The timing error is calculated by the following equation.

$$\text{Error} = \{[1 + C1/(CL_{xtal}+Co)]^{1/2} - [1 + C1/(CL_{system}+Co)]^{1/2}\}/\ [1 + C1/(CL_{xtal}+Co)]^{1/2}$$

where:

C1 is the crystal motional capacitance
Co is the crystal static capacitance
CL_{xtal} is the crystal load capacitance
CL_{system} is the system load capacitance

For the error in ppm, multiply Error by 10^6.

Once the complete timing error has been calculated by adding all of the errors together, compare it to the initial timing budget. Table 5-3 provides a convenient translation of ppm to seconds per month.

Table 5-3 Timing Error as It Translates to Clock Accuracy

Timing Error (Parts per Million)	Seconds/Month
± 10	± 25.9
± 20	± 51.8
± 30	± 77.8
± 40	± 103.7
± 50	± 129.6

Detailed crystal specifications and further information on crystal selection can be found in the *Élan™SC520 Microcontroller Data Sheet*, order #22003.

5.3.2.1 Running the Élan™SC520 Microcontroller at 33.333 MHz

The clock that is supplied to the PCI bus (CLKPCIOUT) is exactly the same as the frequency of the crystal. The ÉlanSC520 microcontroller simply buffers the 33-MHz crystal input and provides it to the CLKPCIOUT pin. Since crystals have inaccuracies, it is possible that these inaccuracies cause the period of CLKPCIOUT to become marginally less than 30 ns.

It is up to the system designer to choose the accuracy of the crystal used with the ÉlanSC520 microcontroller. The 33.000-MHz frequency provides a better guard band than the 33.333-MHz crystal. In practice, most PCI devices can tolerate both frequencies, but it is important to be aware of the impact of choosing the crystal on this potential violation of the PCI bus specification. The *PCI Local Bus Specification,* Revision 2.2 requires that the minimum clock period be 30 ns.

5.3.3 Bypassing Internal Oscillators

The 32.768-kHz and the 33-MHz ÉlanSC520 microcontroller oscillators can be bypassed by connecting an external clock to the crystal pins. See Figure 5-3 and Figure 5-4 for suggested circuitry.

Figure 5-3 shows two resistors in series with their common node connected to 32KXTAL2. The value of the resistor connected to ground (R2) is 100 kΩ. The value of R1 depends on the voltage level of the external oscillator, according to the following formula:

V(32KXTAL2) = 2.5 V = R2 / (R2 + R1) * V(External Oscillator)

Figure 5-3 Bypassing the 32.768-kHz Oscillator

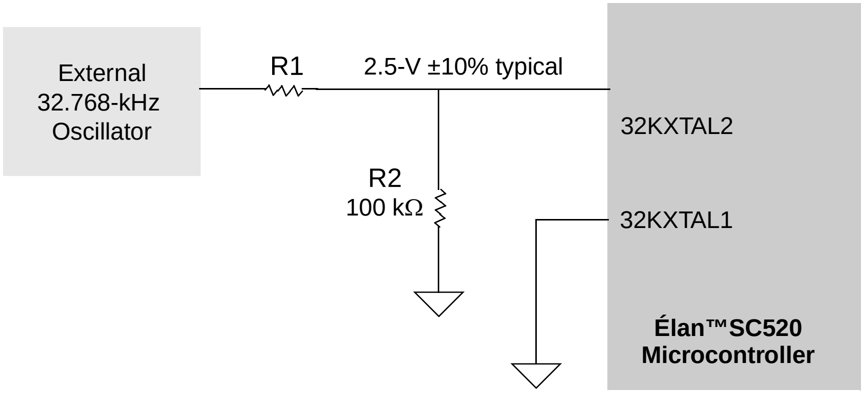
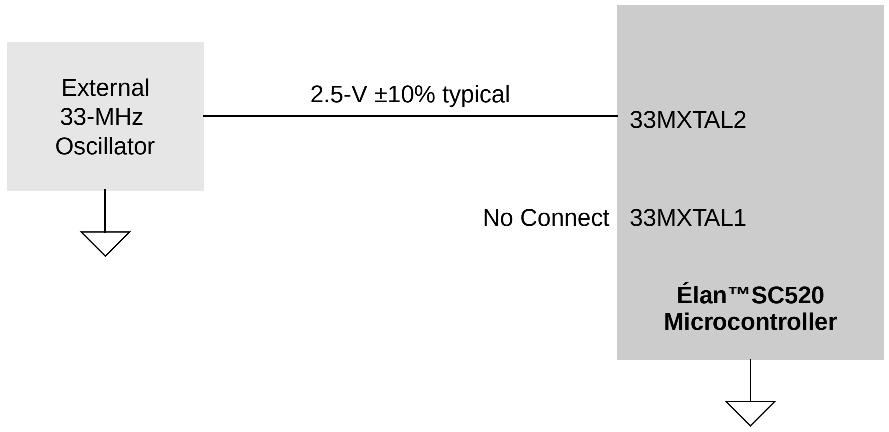


Figure 5-4 Bypassing the 33-MHz Oscillator



5.4 REGISTERS

A summary listing of the memory-mapped configuration registers used to control the clocks on the ÉlanSC520 microcontroller is shown in Table 5-4.

Table 5-4 Clock Control Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
Am5$_x$86 CPU Control	CPUCTL	02h	CPU clock speed control
Software Timer Configuration	SWTMRCFG	C64h	Crystal frequency (33.000 MHz or 33.333 MHz) for software timer
Clock Select	CLKSEL	C26h	CLKTIMER[CLKTEST] pin enable, clock output select options (18.432 MHz or 1.8432 MHz UART, PLL1, PLL2, PIT, and RTC), CLKTIMER or CLKTEST select
GP Timer 0 Mode/Control	GPTMR0CTL	C72h	GP Timer 0: internal clock source prescaler, external clock source
GP Timer 1 Mode/Control	GPTMR1CTL	C7Ah	GP Timer 1: internal clock source prescaler, external clock source
UART 1 General Control UART 2 General Control	UART1CTL UART2CTL	CC0h, CC4h	UARTx clock source: 1.8432 MHz or 18.432 MHz
SSI Control	SSICTL	CD0h	SSI clock speed
GP-DMA Control	GPDMACTL	D80h	GP-DMA clock frequency: 4 MHz, 8 MHz, or 16 MHz

5.5 OPERATION

The clocks on the ÉlanSC520 microcontroller are generated from two local oscillators.

The 32.768-kHz oscillator is used to drive PLL1 (1.47456-MHz PLL), which in turn drives PLL2 (36.864-MHz PLL). The 36.864-MHz clock is divided by 2 to produce the 18.432-MHz UART clock. It is divided by 31 to produce the 1.1892-MHz PIT clock.

The 33-MHz oscillator produces the 33-MHz PCI and CPU clocks. The 33-MHz oscillator is also used to drive PLL3 (66-MHz PLL) to produce the SDRAM clock.

5.5.1 Internal Clocks

5.5.1.1 CPU

The Am5$_X$86 CPU bus frequency in the ÉlanSC520 microcontroller is always 33 MHz; however, the Am5$_X$86 CPU core frequency is programmable to be 100 MHz or 133 MHz. The clock speed of the Am5$_X$86 CPU core defaults to 100 MHz, but can be changed dynamically via the Am5$_X$86 CPU Control (CPUCTL) register (MMCR offset 02h). Clocking considerations for the Am5$_X$86 CPU are described in “Clocking Considerations” on page 7-4.

The ÉlanSC520 microcontroller supports either a 33.000-MHz or 33.333-MHz crystal as the 33-MHz clock source.

5.5.1.2 PCI Bus

The PCI bus system clock on the ÉlanSC520 microcontroller runs at 33 MHz. The PCI bus system clock (CLK) is described in “PCI Clocking” on page 9-5, as is usage of the two PCI bus clock pins, CLKPCIIN and CLKPCIOUT.

The CLKPCIOUT pin is a 33-MHz clock output for the PCI bus devices. This signal is derived from the 33MXTAL2–33MXTAL1 interface.

Note that the ÉlanSC520 microcontroller supports either a 33.000-MHz or 33.333-MHz crystal. “Running the Élan™SC520 Microcontroller at 33.333 MHz” on page 5-5 details some important considerations in choosing a crystal for a PCI system.

5.5.1.3 SDRAM Controller

The SDRAM clock runs at 66 MHz, twice the frequency of the 33-MHz oscillator. The refresh rate of the SDRAM controller is derived from the 32.768-kHz clock. The flexible refresh rate supports a wide variety of devices.

Clocking considerations for the SDRAM controller, including the CLKMEMIN and CLKMEMOUT pins, are described in “SDRAM Clocking” on page 10-6.

5.5.1.4 ROM/Flash Interface

The ROM/Flash controller is clocked from the internal Am5$_X$86 CPU bus and operates at 33 MHz.

5.5.1.5 GP Bus

The GP-bus interfaces internally to the Am5$_X$86 CPU and operates at 33 MHz.

5.5.1.6 GP-DMA Controller

The GP-DMA controller can be programmed to operate at 4 MHz, 8 MHz, or 16 MHz. This option is specified in the GP-DMA Control (GPDMACTL) register (MMCR offset D80h). Note that these frequencies are derived from the 33-MHz clock. The exact frequency is an even fraction of the crystal (33.000-MHz or 33.333-MHz) being used in the system.

5.5.1.7 Programmable Interval Timer

The programmable interval timer (PIT) clock source can be either the derived 1.1892-MHz PIT clock or the CLKTIMER pin.

Note: *Since the PIT clock does not run at the industry-standard 1.19318 MHz, modifications in software must be made to allow for this difference. See “Using the PIT Clock Source in PC/AT-Compatible Systems” on page 16-6 for more information.*

5.5.1.8 General-Purpose Timers

The clock source for the three general-purpose timers is the 33-MHz clock. For Timer 0 and Timer 1, the clock source can also be an external pin or a derived prescale clock. This option is specified in the GP Timer 0 Mode/Control (GPTMR0CTL) register (MMCR offset C72h) and the GP Timer 1 Mode/Control (GPTMR1CTL) register (MMCR offset C7Ah). Clocking considerations for the general-purpose timers are described in “Clocking Considerations” on page 17-5.

5.5.1.9 Software Timer

The software timer uses the 33-MHz clock. Proper configuration of the software timer requires the programmer to specify in the Software Timer Configuration (SWTMRCFG) register (MMCR offset C64h) whether a 33.000-MHz or 33.333-MHz crystal is being used in the system.

5.5.1.10 Watchdog Timer

The watchdog timer uses the 33-MHz clock. It supports up to a 30-second time-out period. The EXP_SEL field in the Watchdog Timer Control (WDTMRCTL) register (MMCR offset CB0h) indicates the exponent value used to calculate the time-out duration.

5.5.1.11 Real-Time Clock

The 32KXTAL2–32KXTAL1 pins are used to connect the external 32.768-kHz crystal or oscillator to the ÉlanSC520 microcontroller. This clock source is then used to clock the internal real-time clock (RTC) included on the ÉlanSC520 microcontroller.

5.5.1.12 UART Serial Ports

The UARTs each support an internal baud-rate clock of either 18.432 MHz or 1.8432 MHz. This frequency is programmed in the CLK_SRC bit in the UART 1 General Control (UART1CTL) register (MMCR offset CC0h) or the UART 2 General Control (UART2CTL) register (MMCR offset CC4h).

5.5.1.13 Synchronous Serial Interface

The synchronous serial interface (SSI) clock is derived from the 33-MHz clock. The CLK_SEL bit in the SSI Control (SSICTL) register (MMCR offset CD0h) is used to configure the frequency of the SSI clock (the SSI_CLK pin). The actual bit rate will vary, depending on whether the system is using a 33.000-MHz or a 33.333-MHz crystal.

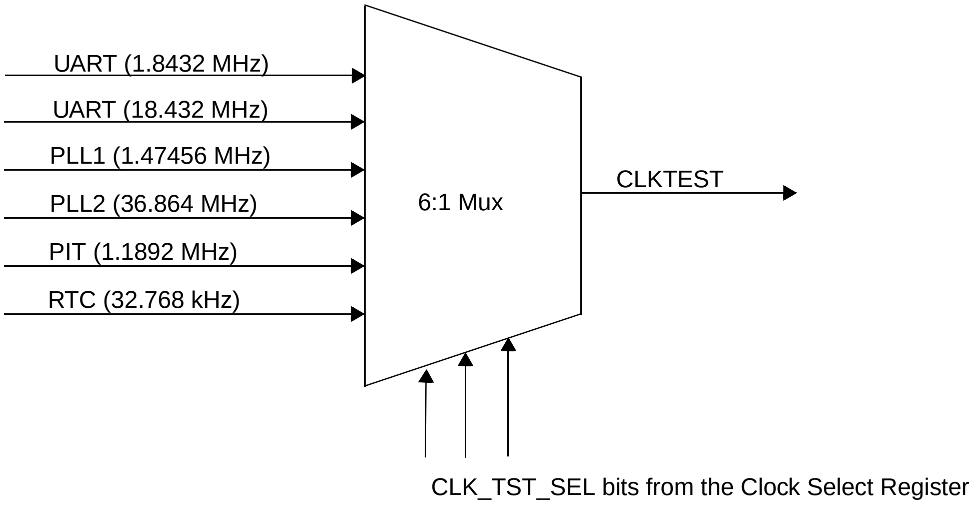
5.5.2 Using the CLKTIMER[CLKTEST] Pin

The CLKTIMER[CLKTEST] pin can be programmed as an input (CLKTIMER) or as an output (CLKTEST) in the Clock Select (CLKSEL) register (MMCR offset C26h).

- When programmed as an input (default), this pin can be used to provide the clock for the programmable interval timer (PIT) core. See “Using the PIT Clock Source in PC/AT-Compatible Systems” on page 16-6 for more information. While the pin is being enabled as an input, it is synchronized to the CPU clock to prevent spurious pulses from occurring in the PIT core.
- When programmed as an output, this pin, as CLKTEST, can drive one of several of the internal clocks outside the microcontroller for testing or drive an external device. Figure 5-5 shows the available clocks that can be directed to the CLKTEST pin by programming the Clock Select (CLKSEL) register (MMCR offset C26h).

Note: *Caution should be exercised when programming the CLKTIMER[CLKTEST] pin as an output, since there is no logic to avoid spurious pulses while enabling or changing clock frequencies. The target device should be held in reset, the CLK_TST_SEL bit field programmed to the correct frequency, the CLK_PIN_DIR bit set to 1 (output), and the CLK_PIN_ENB bit set to 1 (enabled). Then, the target device can be released from reset.*

Figure 5-5 Clock Routing for the CLKTEST Pin



5.6 INITIALIZATION

The Am5$_x$86 CPU core is reset during a system reset, and the CPU core clock frequency defaults to 100 MHz. A soft reset does not affect the CPU core clock frequency.

The CLKTIMER[CLKTEST] pin is disabled on reset and must be enabled via the Clock Select (CLKSEL) register (MMCR offset C26h) before it will function.

See Figure 5-1 on page 5-2 and Table 5-1 on page 5-2 for start-up information. See also Figure 6-3 on page 6-9 and the reset timing diagrams in the *Élan™SC520 Microcontroller Data Sheet*, order #22003.

CHAPTER 6 RESET GENERATION

6.1 OVERVIEW

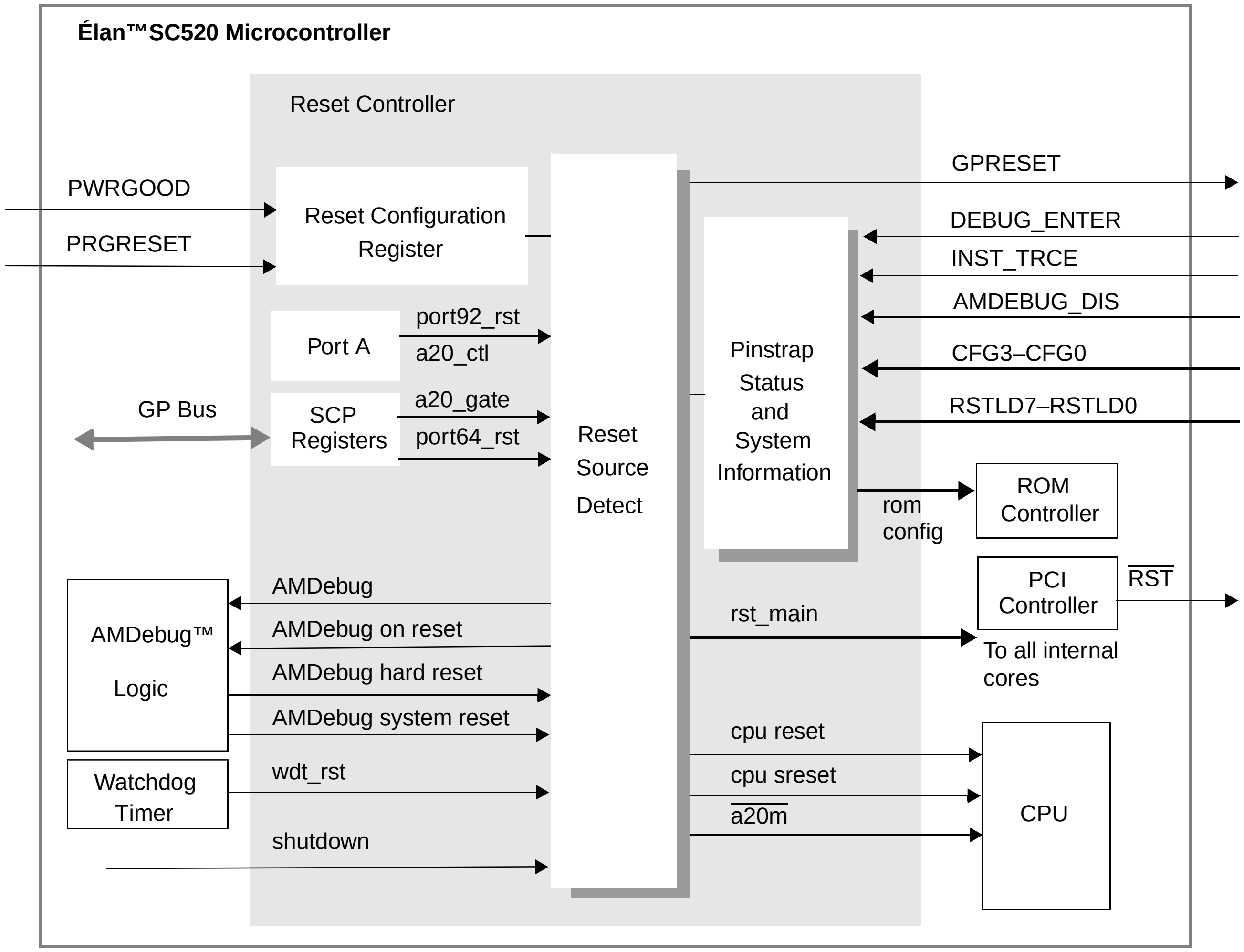
Reset features of the ÉlanSC520 microcontroller include:

- ÉlanSC520 microcontroller system reset generation via PWRGOOD pin, software writes, watchdog timer, and AMDebug system reset
- ÉlanSC520 microcontroller system reset with SDRAM interface contents maintained (called *programmable reset*)
- Hard CPU reset generation via system reset
- Soft CPU reset generation via software writes and detection of the CPU special cycle type “shutdown”
- GP bus reset generation via system reset and software writes
- PCI bus reset generation via system reset and software writes. See Chapter 9, “PCI Bus Host Bridge”
- Reset sources can be determined by software
- Latches system-configuration data presented on the shared CFG3–CFG0 pins and static system board information presented on the shared RSTLD7–RSTLD0 pins at the assertion of the PWRGOOD pin. See Chapter 12, “ROM/Flash Controller”, for information in the CFGx pins.
- System Control Processor (SCP) A20 gate and reset CPU command emulation
- Control bit to enable AMDebug mode after the CPU has been reset

6.2 BLOCK DIAGRAM

Figure 6-1 shows a block diagram of the reset controller.

Figure 6-1 Reset Controller Block Diagram



6.3 SYSTEM DESIGN

The POWERGOOD signal from the system board is connected to the PWRGOOD pin on the ÉlanSC520 microcontroller to produce CPU reset and system reset events. During the period required for stabilization of the power supplies and the internal oscillators, which is typically not less than 1 second, the POWERGOOD signal is kept deasserted. The start-up time of the internal PLLs is typically 10 ms from the assertion of the PWRGOOD pin. The power-on reset waveform diagram is shown in Figure 6-3 on page 6-9.

All system resets, aside from PWRGOOD pin, are on the order of 10 ms, while soft resets take 16 CPU clocks.

See the *Élan™SC520 Microcontroller Data Sheet*, order #22003, for timing tables and additional timing diagrams.

6.4 REGISTERS

The reset generation on the ÉlanSC520 microcontroller is controlled by the memory-mapped registers listed in Table 6-1 and the direct-mapped registers listed in Table 6-2.

Table 6-1 Reset Generation Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
Host Bridge Control	HBCTL	60h	PCI reset ($\overline{\text{RST}}$)
Watchdog Timer Control	WDTMRCTL	CB0h	Watchdog timer enable, WDT reset enable, interrupt flag, duration of the WDT time-out interval
System Board Information	SYSINFO	D70h	System configuration data latched on RSTLD7–RSTLD0 pins at assertion of PWRGOOD
Reset Configuration	RESCFG	D72h	Control bits for system reset, GP bus reset (GPRESET), programmable SDRAM retention reset (PRGRESET pin enable), and AMDebug mode enable
Reset Status	RESSTA	D74h	Reset source status: SCP reset, AMDebug hard reset detect, AMDebug system reset, watchdog timer time-out, CPU shutdown (soft reset), PRGRESET pin, and PWRGOOD pin

Table 6-2 Reset Generation Registers—Direct-Mapped

Register	Mnemonic	I/O Address	Function
SCP Data Port	SCPDATA	60h	System Control Processor (SCP) data write, a20 gate command emulation
SCP Command Port	SCPCMD	64h	SCP command write, a20 gate command emulation, CPU reset command emulation
System Control Port A	SYSCTLA	92h	Soft CPU reset generation, alternate a20 control

6.5 OPERATION

There are several different types of reset supported on the ÉlanSC520 microcontroller:

- System reset
- System reset with SDRAM retention, called *programmable reset*
- Soft CPU reset
- GP bus reset
- PCI reset
- RTC reset

System reset is the primary reference reset on the ÉlanSC520 microcontroller. It is described in “System Reset” on page 6-4.

Table 6-3 shows the ÉlanSC520 microcontroller reset sources and the functions affected.

Table 6-3 Élan™SC520 Microcontroller Reset Sources

Source	CPU (Hard/Soft)	GPRESET Pin	$\overline{\text{RST}}$ Pin (PCI)	Internal Registers	Notes
PWRGOOD pin	Hard	✔	✔	✔	
PRGRESET pin	Hard	✔	✔	✔	1,2
SYS_RST bit, RESCFG register	Hard	✔	✔	✔	2
Watchdog timer reset event	Hard	✔	✔	✔	2
AMDebug system reset	Hard	✔	✔	✔	2
CPU_RST bit, SYSCTLA register (Port 0092h)	Soft				3
SCP soft reset, SCPCMD register (Port 0064h)	Soft				
CPU shutdown (typically caused by a triple fault)	Soft				
GP_RST bit, RESCFG register		✔			
PCI_RST bit, HBCTL register			✔		

Notes:

1. The PRG_RST_ENB bit must be set to enable the reset function on this pin.

2. If the PRG_RST_ENB bit is set, the SDRAM controller configuration is maintained to support system reset in which SDRAM contents are also maintained.

3. Any write of a 1 to the CPU_RST bit will cause a soft reset, regardless if the bit was previously a 1 or 0.

6.5.1 System Reset

System reset on the ÉlanSC520 microcontroller can be initiated by any of the following *reset events*:

- PWRGOOD pin assertion
- Software writes to the SYS_RST bit in the Reset Configuration (RESCFG) register (MMCR offset D72h)
- AMDebug system reset event
- Watchdog timer time-out event that is enabled to generate a system reset

On system reset, the following sequence of events occurs.

1. A system reset event is asserted.
2. Internal CPU, ÉlanSC520 microcontroller internal registers, system GP bus, and PCI bus resets are asserted.
3. The system reset event is deasserted. If PWRGOOD was the source of the reset, configuration and system board data are latched on the CFG3–CFG0 and RSTLD7–RSTLD0 pins, respectively.
4. An RTC reset is generated if the RTC voltage monitor has detected a low RTC battery condition *and* the system reset source was PWRGOOD.
5. Internal PLL start-up time is allowed to pass.
6. Internal CPU, system GP bus, and PCI bus resets are deasserted.

The duration of the system reset is on the order of 10 ms, the start-up time of the internal PLLs. The GPRESET and $\overline{\text{RST}}$ pins are asserted for the 10-ms interval.

In response to the hard CPU reset, all internal Am5$_x$86 CPU registers return to their reset state, and the contents of the CPU cache are discarded. For further information on hard CPU reset, see the *Am486® DX/DX2 Microprocessor Hardware Reference Manual*, 1994 (order #17965).

Note: *The CFG3–CFG0 and RSTLD7–RSTLD0 pins are latched only as a result of the assertion of the PWRGOOD signal, and not as a result of the SYS_RST bit, AMDebug system reset event, or watchdog timer event.*

If the ICE_ON_RST bit in the Reset Configuration (RESCFG) register is set to a 1, the AMDebug utility enters into AMDebug mode after system reset.

The states of the ÉlanSC520 microcontroller cores after system reset are shown in Table 6-4. See the “Initialization” section at the end of each chapter for more detailed information.

Table 6-4 States of Cores after System Reset

Core	State	Comment
Am5$_x$86 CPU	Enabled	CPU clock frequency is set to 100 MHz. Internal registers and internal cache are reset. The FPU must be initialized with an FNINIT instruction.
System arbiter	Enabled	Default is nonconcurrent arbitration mode. All bus masters are disabled except the CPU as PCI and internal Am5$_x$86 CPU bus master.
PCI host bridge master controller	Enabled	
PCI host bridge target controller	Disabled	
SDRAM controller	Disabled	No banks are enabled.
Write buffer and read buffer	Disabled	
ROM controller	Enabled	$\overline{\text{BOOTCS}}$ (only) is enabled at system reset
GP bus controller	Enabled	External GP bus is disabled until PAR registers are initialized.
GP-DMA controller	Enabled	All channels are masked off.
Programmable interrupt controller (PIC)	Enabled	Interrupts are masked at the CPU. NMIs are disabled.
Software timer	Enabled	
General-purpose (GP) timers	Disabled	All GP timer registers are reset to 0. Each timer must be programmed before it can be used.
Programmable interval timer (PIT)	Disabled	Each PIT channel must be programmed before it can be used.
Watchdog timer (WDT)	Disabled	
Real-time clock (RTC)	Enabled	
UARTs	Disabled	
Synchronous serial interface (SSI)	Disabled	Inactive until an SSI command is written.
Programmable input/output (PIO) pins	Enabled	All PIO pins default to inputs and to their PIO function.
JTAG test access port (TAP)	Enabled	$\overline{\text{JTAG\_TRST}}$ should be asserted active Low to ensure normal operation.
AMDebug mode	Enabled	If the ICE_ON_RST bit in the Reset Configuration (RESCFG) register is set.

System reset is a subset of the power-on reset sequence described in “Initialization” on page 6-9.The only real difference between the two is that, for power-on reset, power is being applied to the part in addition to the reset, and the stabilization of power supplies to deassertion of the reset is specified. The two terms are otherwise synonymous in this document.

6.5.2 System Reset with SDRAM Retention

The ÉlanSC520 microcontroller is capable of performing a system reset in which the contents of the SDRAM system are maintained.

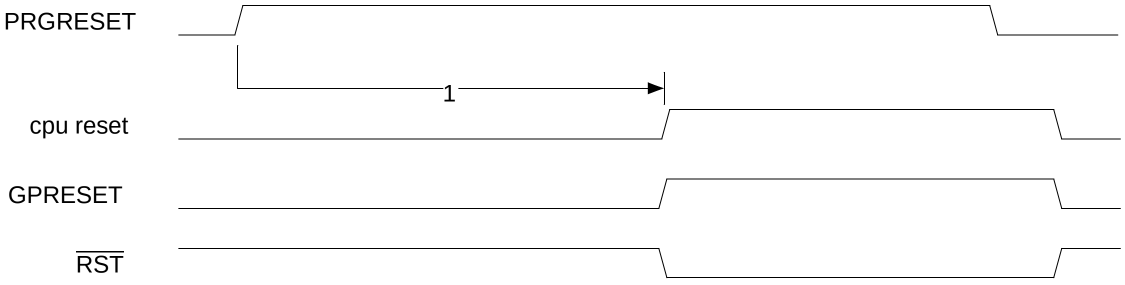
This function, called *programmable reset*, is enabled via the PRG_RST_ENB bit in the Reset Configuration (RESCFG) register (MMCR offset D72h). If this bit is set, assertion of the PRGRESET pin, SYS_RST bit, watchdog timer system reset event, or AMDebug system reset event while PWRGOOD is asserted will result in a system reset in which the SDRAM configuration (SDRAM type, number of banks, refresh rate, etc.) is maintained so that the contents of SDRAM are preserved.

Although the CFG3–CFG0 and RSTLD7–RSTLD0 pins are not latched, all other aspects of this type of reset are the same as a system reset.

The system reset request is arbitrated with the internal SDRAM controller to ensure that all SDRAM banks are idle prior to assertion of the reset. In addition, this arbitration allows the SDRAM controller to complete the current SDRAM cycle. Figure 6-2 shows the sequence of events following a PRGRESET assertion with the PRG_RST_ENB bit enabled.

Note: *If a system reset request is not acknowledged by the SDRAM controller when the PRG_RST_ENB configuration bit is set, a normal system reset occurs. In this event, the PRG_RST_ENB bit is cleared. Clearing of the PRG_RST_ENB bit indicates that the contents of the SDRAM were not maintained.*

Figure 6-2 PRGRESET Timing



Notes:

1. *Reset assertion from PRGRESET assertion is approximately 32 CPU clocks. All SDRAM banks are idle.*
2. *The PRG_RST_ENB bit in the Reset Configuration (RESCFG) register must be enabled.*
3. *The signal “cpu reset” is an internal signal, shown here for reference only. It is not available as an external pin.*

6.5.3 Soft CPU Reset

A soft CPU reset is differentiated from a hard CPU reset in that soft CPU reset does not affect the CPU's cache state. See "Initialization" on page 7-5 for more information about the differences between hard and soft CPU reset.

A soft CPU reset does not reset the ÉlanSC520 microcontroller's internal register bits, with the exception of the NMI_ENB bit in the Interrupt Control (PICICR) register (MMCR offset D00h). A soft CPU reset does not assert the GPRESET or $\overline{\text{RST}}$ pins. For a soft CPU reset, the CPU's internal sreset signal is asserted for 16 clock cycles.

There are four ways a soft CPU reset is generated on the ÉlanSC520 microcontroller:

- A software write to the CPU_RST bit of System Control Port A (SYSCTLA) register (Port 0092h)—Writing a 1 to this bit generates a soft reset event. Following this reset, the CPU_RST bit remains set until software clears it. This feature can be used by software as an indication that the System Control Port A (SYSCTLA) register was used to generate the reset. Writing a 1 to the CPU_RST bit always generates a soft reset, even if the bit was not cleared after a previous reset.
- SCP Reset CPU command—A soft reset event is asserted when the CPU issues the standard command write of FEh to the SCP Command Port (SCPCMD) register (Port 0064h).
- Triple bus fault—A soft reset event is asserted in response to a CPU shutdown cycle due to a triple bus fault.
- Entering AMDebug mode—A soft reset event is also asserted in response to a soft reset command from the AMDebug utility. If the ICE_ON_RST bit in the Reset Configuration (RESCFG) register (MMCR offset D72h) is set to a 1, the AMDebug utility enters into AMDebug mode after a soft CPU reset.

6.5.4 GP Bus Reset

GP bus reset can be generated via a system reset or a software write. Writing a 1 to the GP_RST bit in the Reset Configuration (RESCFG) register (MMCR offset D72h) asserts the GPRESET pin. Clearing this bit to 0 deasserts the GPRESET pin.

6.5.5 PCI Reset

The PCI reset signal, $\overline{\text{RST}}$, is generated via a system reset or software writes. Writing a 1 to the PCI_RST bit in the Host Bridge Control (HBCTL) register (MMCR offset 60h) asserts the PCI $\overline{\text{RST}}$ pin. Clearing this bit to 0 deasserts the PCI $\overline{\text{RST}}$ pin.

6.5.6 RTC Reset

RTC reset occurs anytime the BBATSEN input is sampled below 2.0 V during a power-on reset or during a system reset where the reset source was PWRGOOD. RTC Status D (RTCSTAD) register (RTC index 0Dh) includes a status bit that indicates the validity of the contents of the RAM, time registers, and the calendar. The RTC_VRT bit is set based on the assertion of the internal RTC reset.

Note that this RTC reset may or may not occur when a system reset occurs, depending on the reset source and the state of BBATSEN. BBATSEN also provides a reset signal for the RTC when an RTC backup battery is applied for the first time.

6.5.7 Determining Reset Sources

Status bits are available in the Reset Status (RESSTA) register (MMCR offset D74h) for software to determine the source of reset. These bits are set when the associated event is detected and cleared by writing a 1. They include:

- ICE_HRST_DET—Hard CPU reset from AMDebug logic
- ICE_SRST_DET—AMDebug system reset
- WDT_RST_DET—Watchdog timer time-out system reset
- SD_RST_DET—Soft CPU reset resulting from a detection of the CPU shutdown cycle due to triple fault
- PRGRST_DET—System reset from PRGRESET pin that resets the ÉlanSC520 microcontroller, allows SDRAM refresh, and maintains SDRAM configuration
- PWRGOOD_DET—System reset from PWRGOOD pin

6.5.8 CPU A20 Gate Support

The ÉlanSC520 microcontroller does not support an a20 gate input pin. In a typical PC/AT system, this input was driven by the external System Control Processor (SCP) in response to a command request that is issued by the main CPU. In the ÉlanSC520 microcontroller, this a20 gate command sequence is detected by internal logic, and the appropriate action is taken.The ÉlanSC520 microcontroller provides an additional a20 gate source in the System Control Port A (SYSCTLA) register (Port 0092h). These two a20 gate sources are logically ORed such that both sources must be deasserted to cause the CPU’s a20 output to be gated Low.

The SCP a20 gate command is detected when the CPU issues the standard command write of D1h to the SCP Command Port (SCPCMD) register (Port 0064h), followed by a data write to the SCP Data Port (SCPDATA) register (Port 0060h). Bit 1 of the write to the SCP Data Port (SCPDATA) register drives the a20 control logic. A value of 1 allows the CPU’s a20 signal to propagate to the core logic, while a value of 0 allows the CPU’s a20 signal to be driven Low, as long as no other a20 gate control sources are forcing the CPU’s a20 signal to propagate.

In addition to the SCP a20 gate command emulation, the A20G_CTL bit in the System Control Port A (SYSCTLA) register (Port 0092h) can also be used for alternate a20 signal control. Setting the A20G_CTL bit allows the CPU’s a20 signal to be propagated to the system logic. Clearing this bit (default state) allows the a20 signal to be driven Low as long as no other a20 gate control sources are forcing the a20 signal to propagate.

6.5.9 Clocking Considerations

As a result of an ÉlanSC520 microcontroller system reset event, the internal PLLs are re-started. The PLL start-up time from the deassertion of the system reset source is 10 ms.

6.5.10 Software Considerations

The CPU cache, SDRAM controller write buffer, and PCI transaction queues are discarded as a result of a system reset.

6.5.11 Latency

PRGRESET events must be arbitrated in the SDRAM controller to ensure that the SDRAM devices are in a state in which data is not lost when the PRGRESET event is propagated. This arbitration causes the PRGRESET event to be delayed by no more than 32 CPU clock periods prior to assertion of the internal and external reset signals.

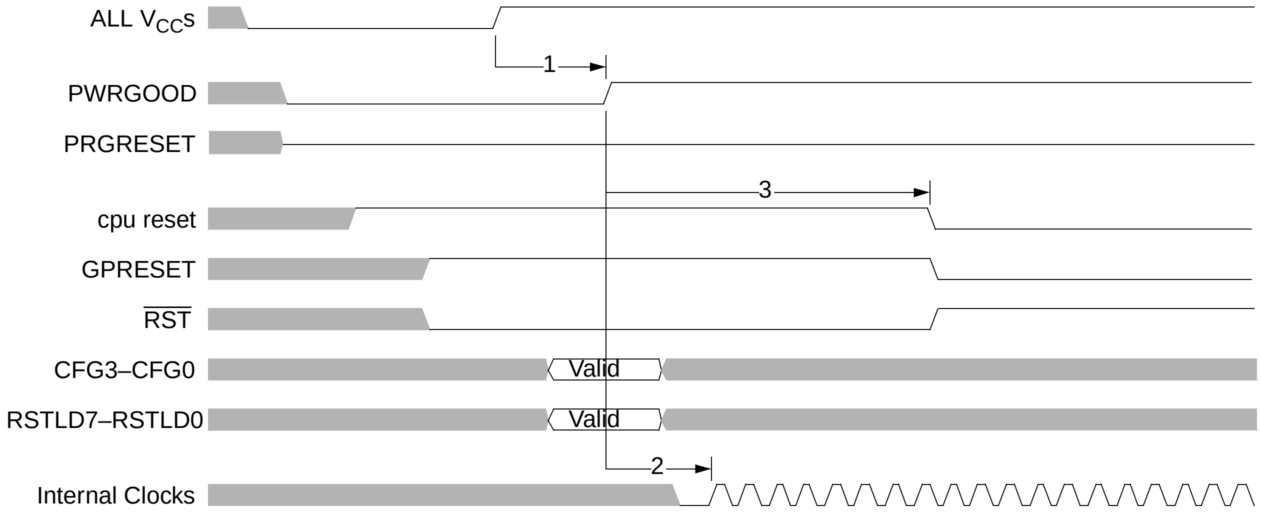
6.6 INITIALIZATION

At power-on reset for the ÉlanSC520 microcontroller, the following sequence of events occurs.

1. The PWRGOOD pin is deasserted.
2. The power planes come up.
3. Internal CPU, ÉlanSC520 microcontroller internal registers, system GP bus, and PCI bus resets are asserted.
4. PWRGOOD is asserted. Configuration and system board data are latched on the CFG3–CFG0 and RSTLD7–RSTLD0 pins, respectively.
5. RTC reset event is generated if the RTC voltage monitor has detected a low RTC battery condition.
6. Internal PLLs are enabled and clocks become stable (internal PLL startup time is allowed to pass).
7. Internal CPU, system GP bus, and PCI bus resets are deasserted.

Figure 6-3 shows this sequence. For power-on reset, the PWRGOOD pin must be held deasserted for the duration of time it takes for the stabilization of the system board power supply output voltages and the start-up time of the internal 32-kHz and 33-MHz oscillators. This time is typically on the order of 1 second.

Figure 6-3 Power-On Reset Sequence of Events



Notes:

1. PWRGOOD *valid from all V_{CC} valid (except VCC_RTC) is typically 1 second.*
2. *PLL start-up time from* PWRGOOD *valid is less than 10 ms.*
3. *CPU reset and external resets deasserted from PWRGOOD are 10 ms.*
4. *Internal signals are shown for reference only; they are not available on external pins.*

CHAPTER

7 Am5$_X$86® CPU

7.1 OVERVIEW

The ÉlanSC520 microcontroller has an integrated Am5$_X$86 CPU core. The features of the Am5$_X$86 CPU include:

- Operation at 100 MHz or 133 MHz, with a 33-MHz bus interface
- 16-Kbyte unified cache configurable for either write-back or write-through cache mode
- Integrated floating point unit (ANSI/IEEE 754 compliant)
- On-chip debug support. See Chapter 26, “AMDebug™ Technology”, for more information.

7.2 BLOCK DIAGRAM

Figure 7-1 shows a block diagram of the Am5$_X$86 CPU.

7.3 REGISTERS

The Am5$_X$86 CPU is controlled by the registers listed in Table 7-1 and Table 7-2.

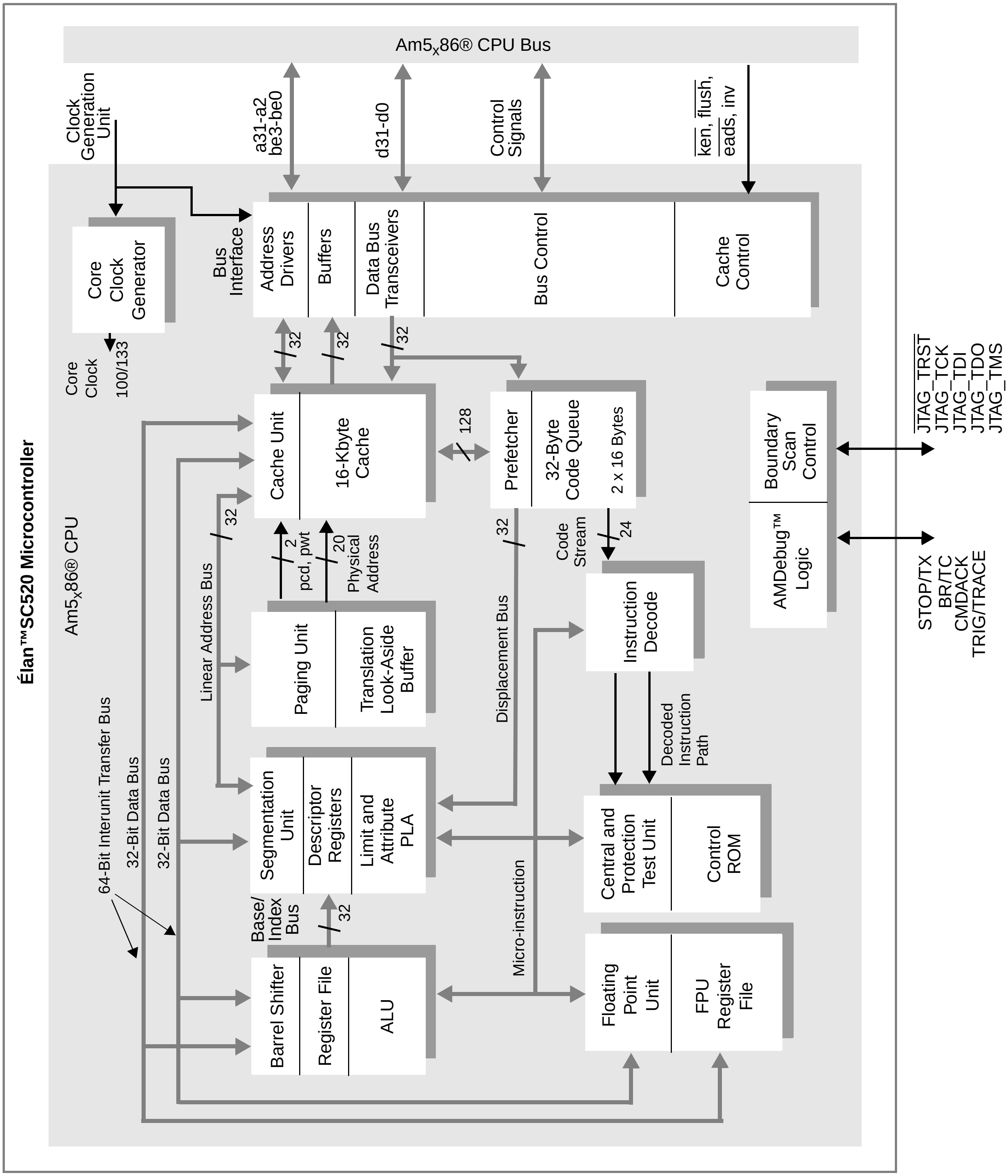
Table 7-1 Am5$_X$86® CPU Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
ÉlanSC520 Microcontroller Revision ID	REVID	00h	Product identification, major and minor stepping level
Am5$_X$86 CPU Control	CPUCTL	02h	CPU cache mode (write-through or write-back), CPU clock speed control
Floating Point Error Interrupt Mapping	FERRMAP	D46h	Floating point error interrupt mapping
Reset Status	RESSTA	D74h	Reset source status: CPU shutdown (soft reset)

Table 7-2 Am5$_X$86® CPU Registers—Direct-Mapped

Register	Mnemonic	I/O Address	Function
SCP Data Port	SCPDATA	60h	System Control Processor (SCP) data write, a20 gate command emulation
SCP Command Port	SCPCMD	64h	SCP command write, a20 gate command emulation, CPU reset command emulation
System Control Port A	SYSCTLA	92h	CPU soft reset generation, alternate a20 control
Floating Point Error Interrupt Clear	FPUERRCLR	F0h	Clear FPU error interrupt

Figure 7-1 Am5_x86® CPU Block Diagram



7.4 OPERATION

The ÉlanSC520 microcontroller is a highly integrated system in silicon, and the $Am5_X86$ CPU is central to this integration. The $Am5_X86$ CPU is a high-performance CPU that is fully software-compatible with the Am486 microprocessor family. Most of the details of the communication between the $Am5_X86$ CPU core and the peripherals are transparent to the user and are not documented here.

A full description of the operation of the $Am5_X86$ CPU is well beyond the scope of this chapter. The following AMD publications are a good starting point for learning about the $Am5_X86$ CPU as it has evolved over time. The oldest publication is listed first. The later publications enhance the original functional descriptions.

- *Am486® DX/DX2 Microprocessor Hardware Reference Manual*, 1994 (order #17965)
- *Enhanced Am486® Microprocessor Family Data Sheet*, 1995, (order #19225)
- *$Am5_X86$® Microprocessor Family Data Sheet*, 1996 (order #19751)

The $Am5_X86$ CPU core in the ÉlanSC520 microcontroller is derived from the Enhanced Am486 family (as described in order #19225). The $Am5_X86$ CPU enhances system performance by raising the maximum CPU operating frequency to 133 MHz, while maintaining complete compatibility with the standard Am486 CPU architecture. The following differences may be relevant to the user:

- There is no provision for an L2 cache. The signals that would be needed are not brought out of the ÉlanSC520 microcontroller.
- System management mode (SMM) is not supported on the ÉlanSC520 microcontroller.
- From an $Am5_X86$ CPU-core perspective only, the cache defaults to the write-back cache mode and reports this state in response to the CPUID instruction. The cache mode can be reconfigured to write-through mode via the $Am5_X86$ CPU Control (CPUCTL) register (MMCR offset 02h).

Programs sometimes require the ability to determine the hardware on which they are running. The ÉlanSC520 microcontroller can be identified via the CPUID instruction and the ÉlanSC520 Microcontroller Revision ID (REVID) register (MMCR offset 00h). This is discussed in “Identifying the CPU Core” on page 3-7.

7.4.1 Floating Point Unit (FPU)

The $Am5_X86$ CPU provides an integrated floating point unit (FPU) that operates in parallel with the Arithmetic Logic Unit (ALU). The FPU is useful in applications that involve more intensive computational complexity. The major features of the integrated FPU are:

- Compliant with ANSI/IEEE 754 standard
- Provides arithmetic instructions to handle various numeric data types and formats
- Provides built-in transcendental functions for functions like sine, cosine, tangent, logarithms, etc.
- Software-compatible with the 80387 (and previous) math co-processors

The FPU must be initialized with an FNINIT instruction after any system reset.

7.4.2 Cache Memory Management

The ÉlanSC520 microcontroller contains a 16-Kbyte unified code and data cache. Cache operation defaults to write-back cache mode. However, this mode can be disabled by setting the Cache Write Mode (CACHE_WR_MODE) bit in the Am5$_X$86 CPU Control register (MMCR offset 02h). Note that the cache should be flushed when switching this bit from write-back to write-through cache mode.

The cache that is internal to the CPU is historically referred to as the *level 1 (L1) cache*. Cache that is located external to the CPU is called *level 2 (L2)*. The ÉlanSC520 microcontroller does not have the control mechanism or the pins to support an L2 cache. The L1 cache can be configured through the standard cache configuration bits in the CPU's machine status (CR0) register. The Cache Disable (CD) and Not Write-Through (NW) bits are decoded as shown in Table 7-3.

Table 7-3 Cache Configuration Options

CD	NW	Operating Mode
1	1	Cache line fills, cache write-throughs, and cache invalidations are disabled. To completely disable the cache, set both CD and NW to 1 and flush the cache by executing a WBINVD or INVD instruction.
1	0	Cache line fills are disabled. Cache write-throughs and cache invalidations are enabled. This configuration allows software to disable the cache for a short time, then re-enable it without flushing the original contents.
0	1	Invalid setting. A general-protection exception with an error code of 0 is generated.
0	0	Cache line fills, cache write-throughs, and cache invalidations are enabled. This is the normal operating configuration.

If paging is enabled in the CPU, then cacheability as well as cache write policy can be controlled on a per-page basis via control bits in the page tables. Note that the CACHE_WR_MODE bit in the Am5$_X$86 CPU Control (CPUCTL) register must be set to write-back cache mode for write-back behavior to occur.

Caching is controlled by the memory management subsystem on a per-access basis. For example, GP bus and PCI bus accesses are not cached. The programmer has control over which regions of memory (SDRAM and ROM) are cacheable and which are not. This is described in detail in Chapter 4, "System Address Mapping".

7.4.3 Clocking Considerations

The Am5$_X$86 CPU bus frequency in the ÉlanSC520 microcontroller is always 33 MHz. However, the Am5$_X$86 CPU core frequency is programmable to be 100 MHz or 133 MHz. The clock speed of the Am5$_X$86 CPU core defaults to 100 MHz, but can be changed dynamically via the Am5$_X$86 CPU Control (CPUCTL) register (MMCR offset 02h). Systems that maintain relatively high cache hit rates benefit more from the higher core speeds, because they are not dependent on external bus activity for accessing ROM or SDRAM.

The clock speed change is transparent to the system, with the exception that there is approximately 1-ms delay to allow the Am5$_X$86 CPU's clock PLLs to stabilize. Following the clock speed configuration, the ÉlanSC520 microcontroller's clock control logic automatically forces the Am5$_X$86 CPU's cache to be flushed, and waits for the completion of the flush before changing the PLLs' frequency select (caching is also disabled for any subsequent memory read cycles during the flush operation). Since the CPU PLLs require approximately 1 ms to stabilize following the speed change, all Am5$_X$86 CPU cache snooping is suspended. However, since the cache was previously flushed, there are no coherency issues, PCI bus master cycles, or GP-DMA controller operations during this period. Interrupts generated to the Am5$_x$86 CPU will be honored only after the Am5$_x$86 CPU is operating again.

Once the CPU PLLs have stabilized and the new core frequency has been established, caching is once again enabled in the same mode as it was prior to the clock speed change. There are no special requirements by external system hardware or software to support clock speed switching.

Note: *Not all ÉlanSC520 microcontroller devices support all CPU clock rates. The maximum supported clock rate for a device is indicated by the part number printed on the package. The clocking circuitry can be programmed to run the device at higher than rated speeds. However, if an ÉlanSC520 microcontroller is programmed to run at a higher clock speed than that for which it is rated, then erroneous operation will result and physical damage to the device may occur.*

7.4.4 Interrupts

The Am5$_x$86 CPU receives a maskable interrupt from the programmable interrupt controller (PIC). The Am5$_x$86 CPU also supports a non-maskable interrupt (NMI) input that can be disabled. See Chapter 15, “Programmable Interrupt Controller”, for details of both maskable and non-maskable interrupt sources and routing.

7.4.5 Latency

The clock speed change is transparent to the system with the exception that there is approximately a 1-ms delay to allow the Am5$_x$86 CPU’s clock PLLs to stabilize. Interrupts generated to the Am5$_x$86 CPU will be honored only after the Am5$_x$86 CPU is operating again.

7.5 INITIALIZATION

The Am5$_x$86 CPU included on the ÉlanSC520 microcontroller supports two different types of CPU reset: hard CPU reset and soft CPU reset. Chapter 6, “Reset Generation” provides details of the various sources of hard and soft reset to the Am5$_x$86 CPU. For additional information on Am5$_x$86 CPU initialization, see Chapter 3, “System Initialization” and the references provided in “Operation” on page 7-3.

7.5.1 Hard CPU Reset

The Am5$_x$86 CPU is reset during a hard CPU reset, and the Am5$_x$86 CPU core clock frequency defaults to 100 MHz. Hard CPU reset is used to initialize the Am5$_x$86 CPU due to deassertion of the PWRGOOD signal, as well as other reset sources (see Table 6-3 on page 6-4). Hard CPU reset resets Am5$_x$86 CPU registers and the internal cache.

Hard CPU reset forces the microprocessor to terminate all execution and local bus activity. All entries into the cache are invalidated, the cache is disabled, and the FPU is reset. The Am5$_x$86 CPU begins executing from the boot vector at FFFFFFF0h after system reset is deasserted. The core clock frequency is 100 MHz.

7.5.2 Soft CPU Reset

Soft CPU reset does not affect the CPU’s write buffers, cache, or cache mode (write-back or write-through). The Am5$_x$86 CPU core clock frequency remains the same, and cache snooping continues unaffected during soft reset.

Soft reset provides a method to switch from protected to real operating mode. After a soft CPU reset, the Am5$_x$86 CPU begins initialization at location FFFFFFF0h. The processor state is the same as it is after a hard reset, except that the internal cache, the CD and NW bits in the Am5$_x$86 CPU’s machine status (CR0) register, and the Am5$_x$86 CPU’s write buffers retain the values they had prior to the soft reset.

A soft reset event clears the NMI_ENB bit in the Interrupt Control (PICICR) register, disabling NMIs. This allows software to initialize the stack pointer before setting the NMI_ENB bit again after a soft reset.

CHAPTER

8 SYSTEM ARBITRATION

8.1 OVERVIEW

The ÉlanSC520 microcontroller includes two arbiters. A CPU bus arbiter arbitrates between the Am5$_x$86 CPU, the PCI host bridge, and the GP-DMA controller on the internal CPU bus. A PCI bus arbiter arbitrates between the Am5$_x$86 CPU and up to five external PCI masters on the external PCI bus. The system arbiter complies with *PCI Local Bus Specification,* Revision 2.2, and complies with PCI bus transaction ordering rules.

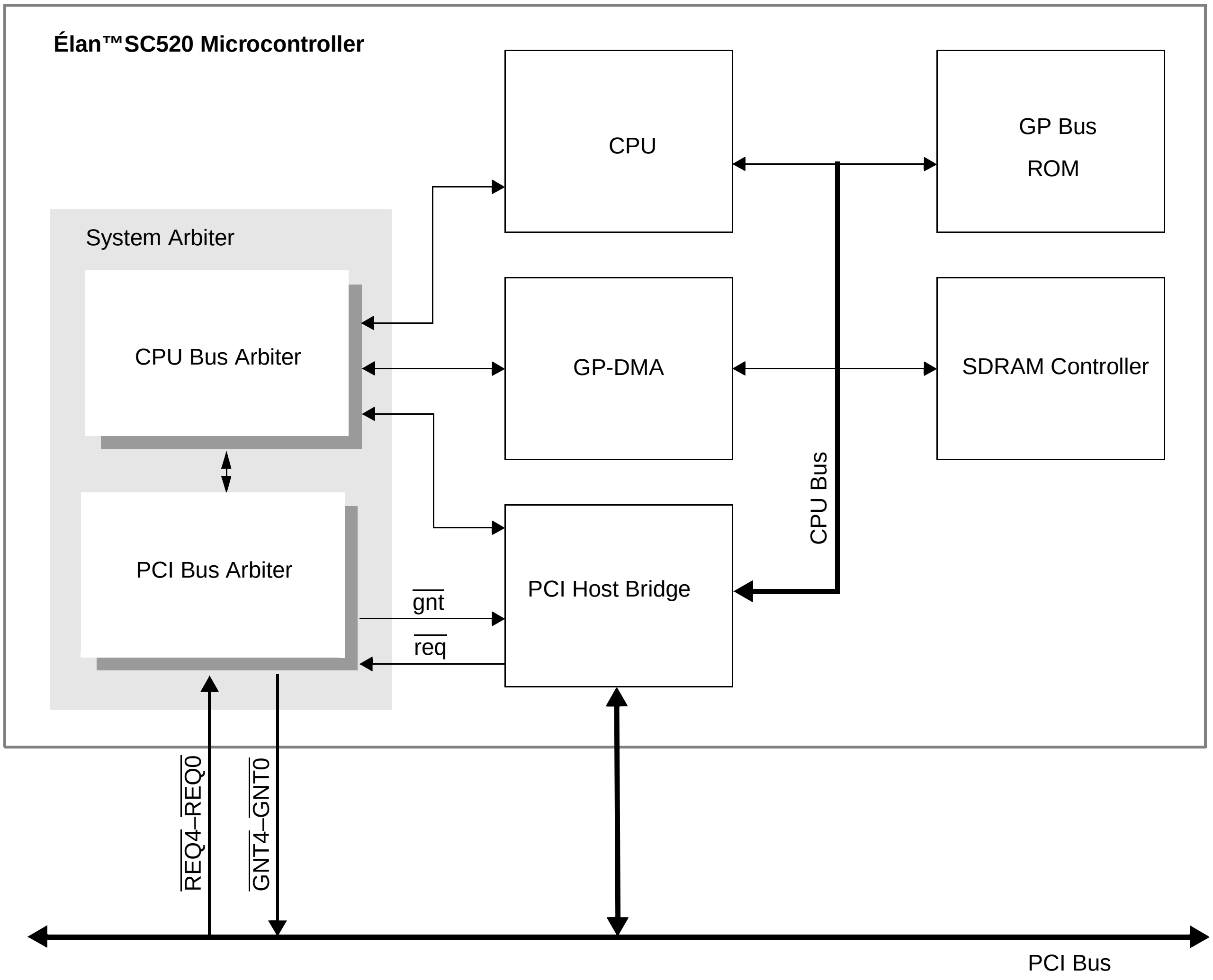
Features of the arbitration subsystem include:

- Supports up to five external PCI bus masters
- Supports concurrent and nonconcurrent operating modes:
  - Concurrent arbitration mode allows PCI bus arbitration to occur independently of CPU bus arbitration, supporting peer-to-peer operation on PCI bus simultaneously with CPU access of memory and GP bus.
  - Nonconcurrent arbitration mode forces all masters to automatically acquire ownership of both PCI and CPU buses, regardless of destination of the cycles.
- PCI bus arbiter supports two queues with rotating priority for bus mastership:
  - High-priority queue supports two bus masters maximum, any masters can be programmed to the high-priority queue.
  - Low-priority queue contains all masters not assigned to the high-priority queue.
- CPU priority is programmable to automatically achieve bus ownership following every one, two, or three PCI-bus-master tenures.
- Option for PCI bus parking on CPU or on last master in concurrent arbitration mode
- PCI bus master request/grant pairs can be individually masked in a separate control register.
- CPU bus arbiter provides an automatic Am5$_x$86 CPU bypass that allows continued PCI bus and GP-DMA access of SDRAM during Am5$_x$86 CPU clock changes and PLL stabilization periods.

8.2 BLOCK DIAGRAM

Figure 8-1 shows a block diagram of the system arbiter.

Figure 8-1 System Arbitration Block Diagram



8.3 REGISTERS

The arbitration subsystem is controlled by the memory-mapped registers listed in Table 8-1.

Table 8-1 System Arbitration Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
System Arbiter Control	SYSARBCTL	70h	PCI bus parking select, concurrent arbitration mode enable, PCI bus grant time-out interrupt enable
PCI Bus Arbiter Status	PCIARBSTA	71h	PCI bus arbiter grant time-out identification and status
System Arbiter Master Enable	SYSARBMENB	72h	Enables for PCI bus $\overline{\text{REQ4}}$–$\overline{\text{REQ0}}$ signals
Arbiter Priority Control	ARBPRICTL	74h	PCI bus arbiter rotating priority queue control

Table 8-1 System Arbitration Registers—Memory-Mapped (Continued)

Register	Mnemonic	MMCR Offset Address	Function
PCI Host Bridge Interrupt Mapping	PCIHOSTMAP	D14h	System arbiter and PCI host bridge interrupt mapping to any of 22 available interrupt channels or NMI, PCI NMI enable control

8.4 OPERATION

The ÉlanSC520 microcontroller's arbitration subsystem consists of two separate bus arbitration units for the CPU bus and the PCI bus.

- The CPU bus arbiter arbitrates between the Am5$_X$86 CPU, the PCI host bridge, and the GP-DMA controller on the internal CPU bus.
- The PCI bus arbiter arbitrates between the Am5$_X$86 CPU and up to five external PCI masters on the external PCI bus.

8.4.1 Operating Modes

The system arbiter can operate in two modes for maximum flexibility:

- Nonconcurrent arbitration mode
- Concurrent arbitration mode

The two bus arbiters operate completely independently when the system is configured for concurrent arbitration mode, but they are interlocked when the system is configured for nonconcurrent arbitration mode.

Maximum performance is typically achieved in concurrent arbitration mode, because this allows simultaneous PCI bus and CPU bus operation. However, some systems may benefit from nonconcurrent arbitration mode, especially if the system experiences data coherency problems due to older, non-compliant bus bridges.

The arbitration mode is specified with the CNCR_MODE_ENB bit in the System Arbiter Control (SYSARBCTL) register (MMCR offset 70h). System arbitration defaults to nonconcurrent arbitration mode after reset.

8.4.1.1 Nonconcurrent Arbitration Mode

Nonconcurrent arbitration mode forces all masters to automatically acquire ownership of both PCI and CPU buses, regardless of destination of the cycles. In this mode, no concurrency between the CPU bus and the PCI bus is allowed. External PCI masters are only granted the PCI bus when the host bridge has been granted the CPU bus, even for peer-to-peer transfers.

When an external PCI bus master requests the PCI bus, the following occurs:

1. The PCI bus arbiter samples an external PCI request asserted and asserts the host bridge request to the CPU bus arbiter. The PCI bus arbiter is parked on the CPU by default and should not be programmed to park on the last master in this mode.
2. The CPU bus arbiter samples the host bridge request asserted and grants the CPU bus to the host bridge at the completion of the next Am5$_X$86 CPU cycle. The CPU bus is owned by the Am5$_X$86 CPU by default, so a request to the CPU must be asserted to gain ownership of this bus.

3. The PCI bus arbiter sees that the host bridge has been granted the CPU bus and grants the PCI bus to the external PCI master requesting the PCI bus. Note that now the external PCI master owns both the PCI bus and the CPU bus.

In nonconcurrent arbitration mode, the PCI bus and CPU bus essentially become one bus where only one master is allowed on the bus at any time. Note that write-posting from the CPU to the PCI bus should be disabled while the arbiter is configured for nonconcurrent arbitration mode.

Note that there is an exception to the normal rules of non-concurrency in this mode, as listed in the following steps:

1. The CPU acquires both buses and performs a memory or I/O read/write of an external PCI target. The target issues a retry to the CPU. The PCI bus is idle due to the retry, but the CPU still remains active (in a wait state) on the CPU bus.
2. An external PCI bus master now asserts a request to perform a memory write to SDRAM. In normal nonconcurrent arbitration mode, this request would not be granted, because the PCI bus arbiter would be waiting to acquire ownership of the CPU bus (but the CPU is in a wait state waiting to retry the PCI target read). PCI bus transaction ordering specifies that a PCI agent cannot base the acceptance of a memory write as a target on the completion of a read as a master. Therefore the ÉlanSC520 microcontroller's host bridge must force the CPU off the bus and allow the external master write to complete.
3. After asserting $\overline{\text{boff}}$ to the CPU, the arbiter grants the PCI bus to the external master, and the master completes its write. When the PCI bus master completes the write, the $\overline{\text{boff}}$ signal is deasserted and the CPU is back on the CPU bus. The original CPU-to-PCI transaction is now retried by the ÉlanSC520 microcontroller's host bridge master controller.

8.4.1.2 Concurrent Arbitration Mode

Concurrent arbitration mode allows PCI bus arbitration to occur independently of CPU bus arbitration, supporting peer-to-peer operation on PCI bus simultaneous with CPU access of memory and the GP bus. In this mode, the CPU bus arbiter and PCI bus arbiter operate independently. Default bus ownership for each of the two arbiters is the same as nonconcurrent arbitration mode. External PCI masters are granted the PCI bus without the host bridge being granted the CPU bus. This allows concurrent CPU bus and PCI bus operation.

A few examples of concurrency are:

- The Am5$_x$86 CPU accessing SDRAM concurrently with an external PCI bus master writing data to the host bridge's target FIFOs
- The Am5$_x$86 CPU or GP-DMA controller accessing SDRAM concurrently with an external PCI bus master accessing an external PCI bus target (peer-to-peer transfer)
- The ÉlanSC520 microcontroller's host bridge target controller accessing SDRAM concurrently with the master controller writing posted data to an external PCI target

8.4.2 CPU Bus Arbiter

The CPU bus arbiter controls access to the internal CPU bus. This internal bus allows for:

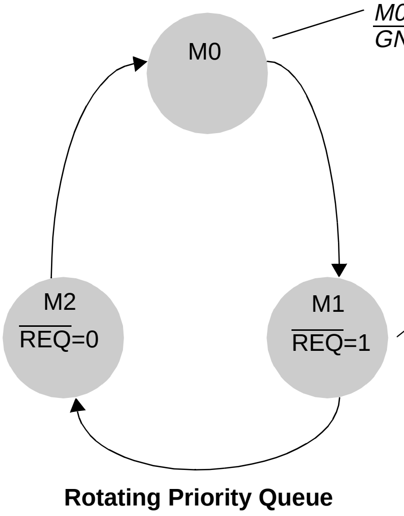
- Am5_x86 CPU access of SDRAM, GP bus, PCI, or ROM
- GP-DMA access of SDRAM
- PCI host bridge access of SDRAM for external PCI master cycles

No concurrent operation is allowed on the CPU bus (e.g., Am5_x86 CPU accessing the GP bus while the PCI host bridge is accessing SDRAM). At any time, only one master is granted access to the CPU bus.

8.4.2.1 CPU Arbitration Protocol

The CPU bus arbiter implements a rotating priority algorithm that guarantees each bus master a place in the arbitration rotation. A master becomes lowest priority in the queue when it receives a bus grant. A master is skipped in the rotation if its request is not asserted, but a lower priority master request is asserted. In this case, the skipped master becomes lowest priority as if it had been serviced (see Figure 8-2).

Figure 8-2 Skipped Master Example



M0 is finishing its transaction; therefore its $\overline{REQ}$ and $\overline{GNT}$ are being deasserted

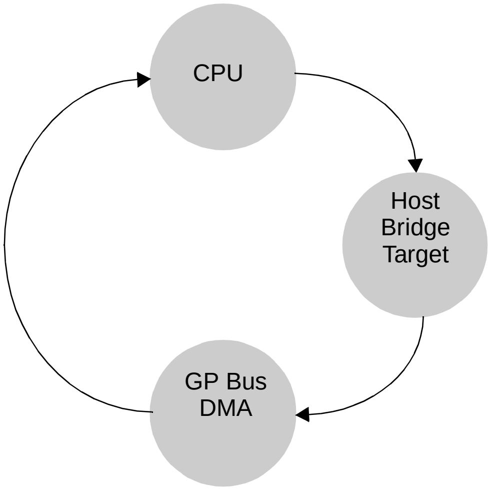
M1 not requesting the bus at the end of the M0 transaction; thus it is skipped, and M2 gets $\overline{GNT}$ asserted instead

Notes:
Priority: M0, M2, M0, M1, M2, M0, M1, M2, ...

In the example shown in Figure 8-2, assume that M0 has just finished a transaction. In this case, the next master in the rotating priority queue would be M1. M1, however, is not requesting the bus, and M2 (a lower priority master at this time) is requesting the bus. In this case, M1 is skipped and the bus is granted to M2. M1 is the lowest priority master in the rotation after being skipped, as if it had been granted the bus. After M2 finishes its transaction, M0 becomes the highest priority master.

The rotating queue for the CPU bus can be seen in Figure 8-3. The Am5_x86 CPU is the default owner when no master is requesting the CPU bus and following reset. The host bridge becomes a bus requestor when it has posted write data from a PCI bus master, or it needs to perform a SDRAM read for a PCI bus master.

Figure 8-3 CPU Bus Rotating Priority Queue



8.4.2.2 CPU Cache Snooping

The Am5$_X$86 CPU includes a write-back cache that updates only the internal cache on memory writes from the CPU (if configured for write-back mode). When only the internal cache memory is updated for a memory write, the external SDRAM contains invalid data. Thus, snooping is required to maintain coherency when other bus masters are accessing SDRAM. Any time another master (GP-DMA or PCI host bridge) is accessing a SDRAM location that contains stale data (valid data is in Am5$_X$86 CPU cache), the valid cache data must be written back to SDRAM before the other master is allowed access to the SDRAM. Therefore, all non-Am5$_X$86 CPU accesses to SDRAM (both reads and writes) are snooped by the Am5$_X$86 CPU.

The Am5$_X$86 CPU cache can be optionally configured to operate in write-through cache mode by setting the CACHE_WR_MODE bit in the Am5$_X$86 CPU Control (CPUCTL) register (MMCR offset 02h). In this mode, both the internal cache and external memory are updated on memory writes. Because the external memory is updated, there are no cache data concurrency issues due to Am5$_X$86 CPU memory writes. Other master write cycles are still snooped, however, to keep the Am5$_X$86 CPU's cache coherent with external memory. In this case, the external memory is updated, and the cache contains invalid data. The snoop invalidates this internal cache location to maintain coherency. There is no overhead involved with snooping when the cache is configured for write-through cache mode. The snoop happens during the cycle (no preemption, write-back, or additional wait-states are inserted).

The ÉlanSC520 microcontroller does not support dynamic cache-write policy changes.

8.4.2.3 Accessing the PCI Host Bridge Target

The PCI host bridge allows external PCI bus masters to read and write the ÉlanSC520 microcontroller's SDRAM. Two 64 doubleword FIFOs (one read, one write) in the ÉlanSC520 microcontroller's host bridge are used to increase PCI bus performance. Once granted the bus by the CPU bus arbiter, the PCI host bridge target controller is allowed to prefetch up to 64 DWORDs (for a memory-read-multiple command), or write (memory-write or memory-write-and-invalidate commands) up to 64 doublewords before the bus is granted to another master. During this time, no other master is granted the CPU bus. The Am5$_X$86 CPU, however, is granted the bus during this time to write back a cache location if necessary.

8.4.2.4 GP Bus DMA Arbitration

The GP-DMA controller allows internal and external GP bus peripherals to have DMA access to SDRAM. There is no preemption mechanism for GP-DMA. Therefore, once a DMA transaction begins, no other master is granted the CPU bus until the DMA controller deasserts its bus request, which varies according to whether the channel is programmed for a single cycle transfer or a block mode transfer. See Chapter 14, “GP Bus DMA Controller”, for information on the various DMA modes and transactions. However, the Am5$_X$86 CPU is granted the bus during this time to write back a cache location, if necessary.

8.4.2.5 Arbitration During Clock Speed Changes

The Am5$_X$86 CPU’s internal core clock speed can be changed dynamically during operation, for systems that require thermal management. While the clock is changing, there is a period where the Am5$_X$86 CPU cannot generate any bus cycles; therefore, cache snooping cannot be performed.

To allow bus masters continued access of SDRAM during the long PLL recovery times, the CPU bus arbiter masks the Am5$_X$86 CPU bus requests and allows only the PCI host bridge and GP-DMA controller access to the CPU bus. If no master is requesting the CPU bus, the CPU bus arbiter is the default owner (no master is granted the bus).

Note that during normal operation when the Am5$_X$86 CPU core clock is not changing, the Am5$_X$86 CPU is the default owner of the CPU bus.

8.4.3 PCI Bus Arbiter

The *PCI Local Bus Specification,* Revision 2.2, defines a central resource known as the *arbiter*. This resource controls PCI master access to the PCI bus. The arbitration approach is *access-based*, which means a PCI master is only granted the bus when it needs (requests) the bus (except in the case of bus parking, discussed in “Bus Parking” on page 8-10).

A simple request/grant handshake is used where each PCI master has a unique request ($\overline{\text{REQ}}$) and grant ($\overline{\text{GNT}}$) signal. PCI bus arbitration is *hidden*, which means arbitration for the next cycle occurs during the current cycle, so that no cycles are wasted due to arbitration (except when the bus is in the idle state and no other requests/grants are active).

The PCI bus is *parked* on a PCI master when the bus is idle to prevent floating signals on the bus. This is done by asserting a PCI master’s $\overline{\text{GNT}}$ signal, even though the PCI master is not requesting the bus. In turn, the PCI master turns on its output drivers, which prevents the bus from floating.

The ÉlanSC520 microcontroller includes the PCI bus arbiter central resource. The integrated PCI bus arbiter arbitrates between the PCI host bridge (Am5$_X$86 CPU as PCI master) and up to five external masters. The $\overline{\text{req}}$/$\overline{\text{gnt}}$ signal pair for the PCI host bridge on the ÉlanSC520 microcontroller is internally connected to the PCI bus arbiter. Five external $\overline{\text{REQ}}$/$\overline{\text{GNT}}$ pin pairs ($\overline{\text{REQ4}}$–$\overline{\text{REQ0}}$, $\overline{\text{GNT4}}$–$\overline{\text{GNT0}}$) are provided to connect external PCI masters to the ÉlanSC520 microcontroller’s PCI bus arbiter. In the following descriptions in this chapter, the term *PCI bus arbiter* refers to the ÉlanSC520 microcontroller’s integrated PCI bus arbiter.

Because the Am5$_X$86 CPU does not burst memory-write cycles (except cache write-backs, which do not apply here because PCI bus memory is noncacheable in the ÉlanSC520 microcontroller), the ÉlanSC520 microcontroller will not burst more than two consecutive doublewords during a CPU write to the PCI bus. Therefore, the PCI bus master latency timer is not provided in the ÉlanSC520 microcontroller.

8.4.3.1 PCI Bus Arbitration Protocol

The *PCI Local Bus Specification,* Revision 2.2, states that the central arbiter must implement a fairness algorithm, which means that each potential bus master must be granted access to the bus independently of other requests. The PCI bus arbiter satisfies this requirement by implementing a rotating priority arbitration scheme that guarantees each bus master a place in the arbitration rotation (see Figure 8-3 on page 8-6 for information on rotating priority arbitration).

Rotating priority mode alone may not provide adequate arbitration in a system where it is known that some PCI bus masters require more bandwidth than others. Therefore, the ÉlanSC520 microcontroller's PCI bus arbiter has two rotating priority queues to accommodate this requirement: a high-priority queue and a low-priority queue.

The masters in the high-priority queue are granted more bandwidth than masters in the low-priority queue. The high-priority queue can contain up to two PCI masters, and the low-priority queue contains all masters that are not in the high-priority queue. The HI_PRI_0_SEL and HI_PRI_1_SEL bit fields in the Arbiter Priority Control (ARBPRICTL) register (MMCR offset 74h) are used to specify the position of each PCI master in the high-priority queue.

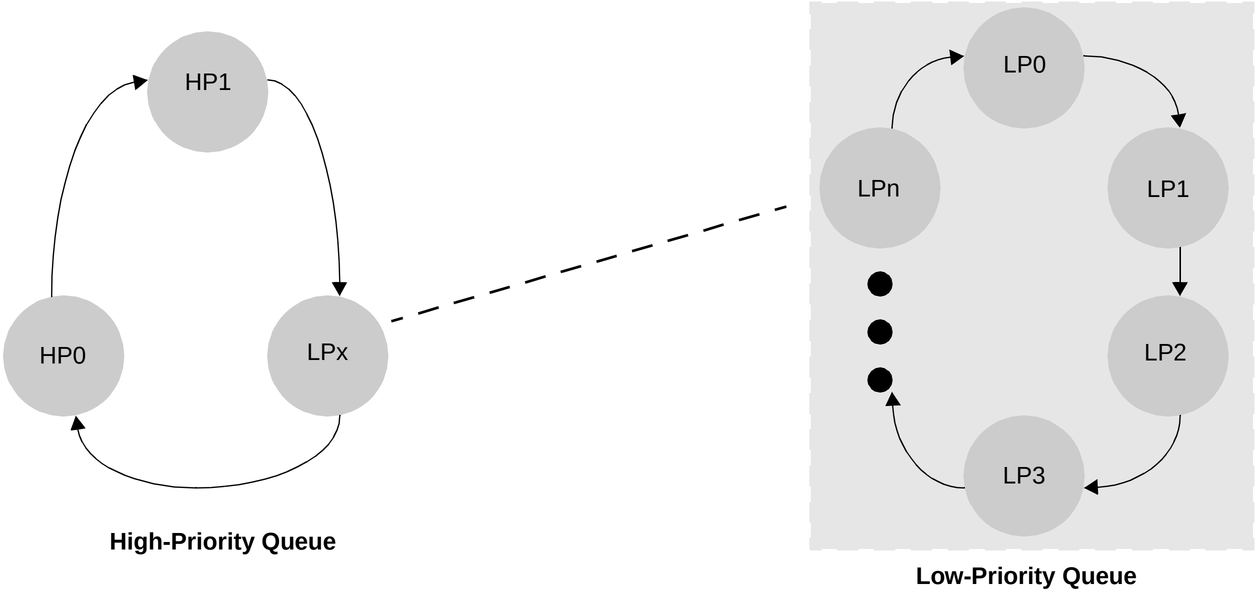
Both queues have rotating priority, and one low-priority master is granted the bus for every rotation of the high-priority queue. After the low-priority master is granted the bus, the low-priority queue rotates to the next low-priority master (see Figure 8-4).

Any one or two (or none) of the ÉlanSC520 microcontroller's PCI bus masters can be placed in the high-priority queue. Note that programming the same bus master for both slots in the high-priority queue *does* provide additional performance for that master. The net result of programming the same master in both slots of the high-priority queue is that the master is given tenure during both slots. If no masters are in the high-priority queue, then there is one rotating priority queue where each master has equal priority.

The high and low-priority queues are for external PCI bus masters, and the Am5$_X$86 CPU PCI master adds an additional level of arbitration. The PCI bus arbiter can be configured with the CPU_PRI bit field in the Arbiter Priority Control (ARBPRICTL) register to grant the bus to the Am5$_X$86 CPU after every one, two, or three external PCI transactions (where the external PCI master to be granted the bus is determined from the high and low-priority queues). This implements another rotating priority queue (see Figure 8-5).

See the *PCI Local Bus Specification,* Revision 2.2, for detailed requirements of PCI bus arbitration.

Figure 8-4 External PCI Master Arbitration Queues



Notes:

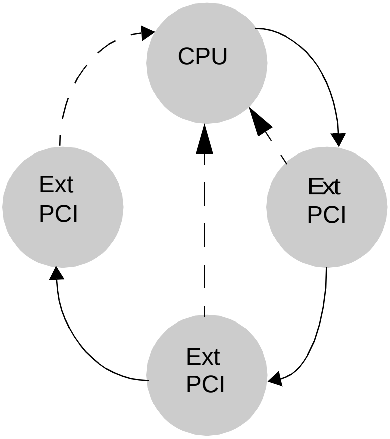
HP0, HP1: High-priority masters

LP0, LP1, LP2, LP3, ..., LPn: Low-priority masters

LPx: Current low-priority master selected

Priority: HP0, HP1, LP0, HP0, HP1, LP1, HP0, HP1, LP2, HP0, HP1, LP3, ..., HP0, HP1, LPn

Figure 8-5 Host Bridge Master Arbitration Queue



Notes:

The PCI bus arbiter is configurable to grant the bus to the host bridge after every 1, 2, or 3 external PCI transactions.

Priority configured for 1: CPU, Ext PCI, CPU, Ext PCI, ...

Priority configured for 2: CPU, Ext PCI, Ext PCI, CPU, Ext PCI, Ext PCI, ...

Priority configured for 3: CPU, Ext PCI, Ext PCI, Ext PCI, CPU, Ext PCI, Ext PCI, Ext PCI, ...

8.4.3.2 Bus Parking

The PCI bus arbiter parks the bus on a PCI bus master when the bus is idle (no master is requesting the bus). This is required on the PCI bus to guarantee that the bus is properly terminated at all times. The PCI bus arbiter arbitrates for the next transaction as soon as the current PCI master that is granted the bus begins its transaction.

Bus parking is configured with the BUS_PARK_SEL bit in the System Arbiter Control (SYSARBCTL) register (MMCR offset 70h). Note that the BUS_PARK_SEL bit must not be changed except during PCI bus arbiter initialization after a system of programmable reset.

8.4.3.2.1 Nonconcurrent Arbitration Mode Bus Parking

The bus should always be parked on the CPU in nonconcurrent arbitration mode. This is necessary to guarantee adequate CPU performance. Otherwise, the CPU would be required to acquire ownership of both the CPU bus and the PCI bus for each external access (including code fetches).

8.4.3.2.2 Concurrent Arbitration Mode Bus Parking

In concurrent arbitration mode, the PCI bus arbiter can be configured to park on the last master that was granted the bus or configured to always park on the Am5$_x$86 CPU. If no other PCI masters are requesting the bus, the $\overline{\text{GNT}}$ to the current PCI master remains asserted until the current PCI master transaction completes.

A bus master that is parked can start a transaction without asserting its $\overline{\text{REQ}}$ pin (PCI bus protocol allows a master to start a cycle when its $\overline{\text{GNT}}$ is asserted and the bus is idle), but it must assert $\overline{\text{REQ}}$ if it requires multiple transactions.

When no PCI bus requests or grants are active, the arbiter retains priority established from the last tenure. For example, if the bus is idle and no requests or grants are active and all masters simultaneously request the bus, the arbiter services the master that is next in rotation.

8.4.3.3 Rearbitration

A PCI bus master that is granted the bus and has not started a transaction within 16 clocks after the bus becomes idle can be assumed to be “broken.” In this case, the PCI bus arbiter automatically re-arbitrates and grants the bus to the next PCI master.

An interrupt can be generated when a PCI bus master that has acquired bus ownership has not started a transaction within 16 clocks, and the $\overline{\text{REQ}}/\overline{\text{GNT}}$ number of the “broken” PCI master is reported in the PCI Bus Arbiter Status (PCIARBSTA) register (MMCR offset 71h). This allows software to disable the broken master and modify the bus parking such that the PCI bus is parked on the CPU.

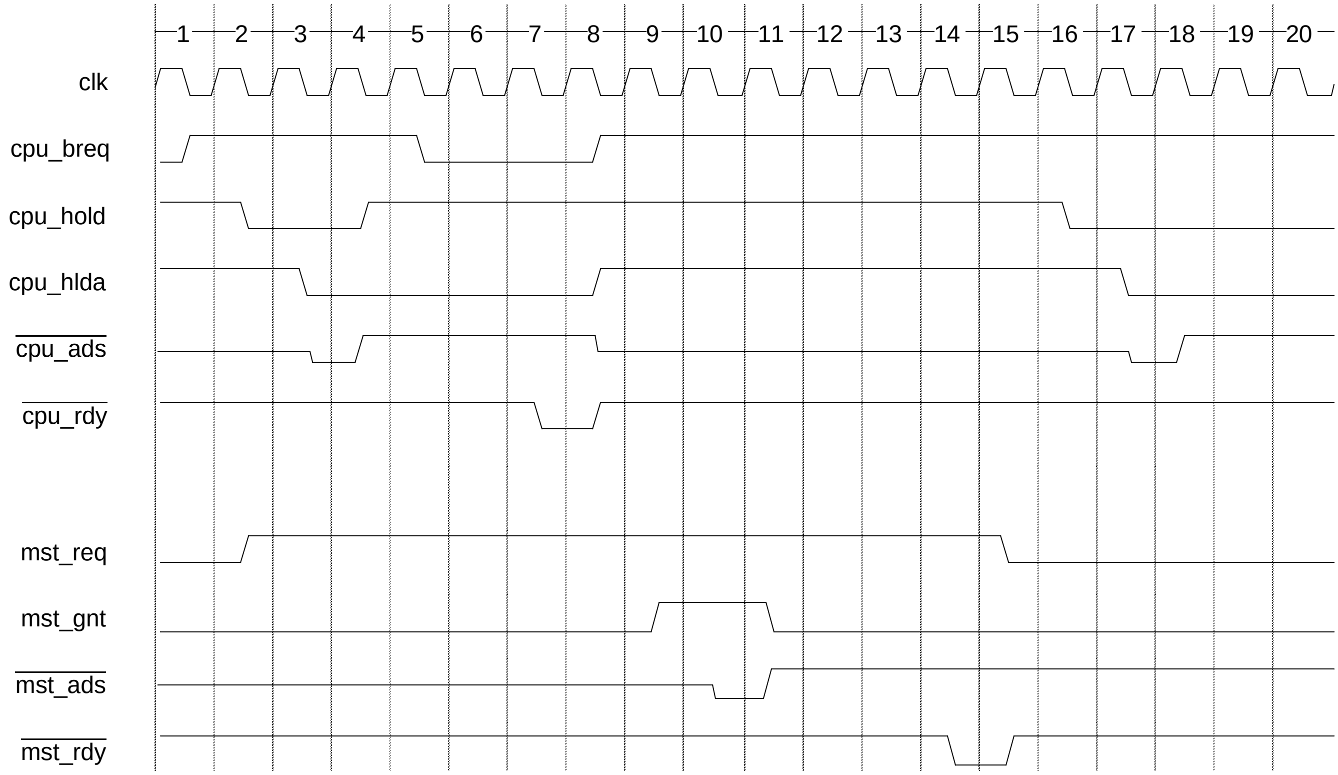
8.4.4 Bus Cycles

This section includes example timing diagrams showing various types of arbitration that may occur in the ÉlanSC520 microcontroller. Note that these are example cases only, and not all cases are shown. The diagrams are functionally representative in nature, and should not be used to infer detailed timing information. Note also that the synchronization between the CPU and PCI clock domains is not shown in detail.

8.4.4.1 CPU Bus Arbitration

Figure 8-6 shows CPU bus arbitration between two CPU bus masters (for clarity, this diagram shows only two bus masters). For additional CPU bus masters, there would be more arbitration signal groups and more than one CPU bus transaction could take place before an individual CPU bus master would be granted the bus.

Figure 8-6 CPU Bus Arbitration


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
clk
cpu_breq
cpu_hold
cpu_hlda
$\overline{\text{cpu\_ads}}$
$\overline{\text{cpu\_rdy}}$
mst_req
mst_gnt
$\overline{\text{mst\_ads}}$
$\overline{\text{mst\_rdy}}$

Notes:

In Figure 8-6, the CPU bus master signals are labeled mst_xxx and the Am5$_X$86 CPU signals are labeled cpu_xxx.

Snooping is not shown in this figure.

The clk signal denotes the 33-MHz clock source and represents both the CPU clock and the PCI clock. This diagram does not represent the full synchronization of signals between these clock domains.

The following sequence annotates the CPU bus arbitration cycle shown in Figure 8-6.

- **Clock #1**: The Am5$_X$86 CPU requests the bus by asserting cpu_breq. Note at this time that the bus is granted to some other master because cpu_hlda is asserted.
- **Clock #2**: The CPU bus arbiter samples the Am5$_X$86 CPU's request asserted and begins arbitration. The CPU bus arbiter determines that the bus is free and that the Am5$_X$86 CPU is the next master to receive the bus, so it deasserts cpu_hold to the Am5$_X$86 CPU. If the bus was not free or the Am5$_X$86 CPU was not the next master to receive the bus,

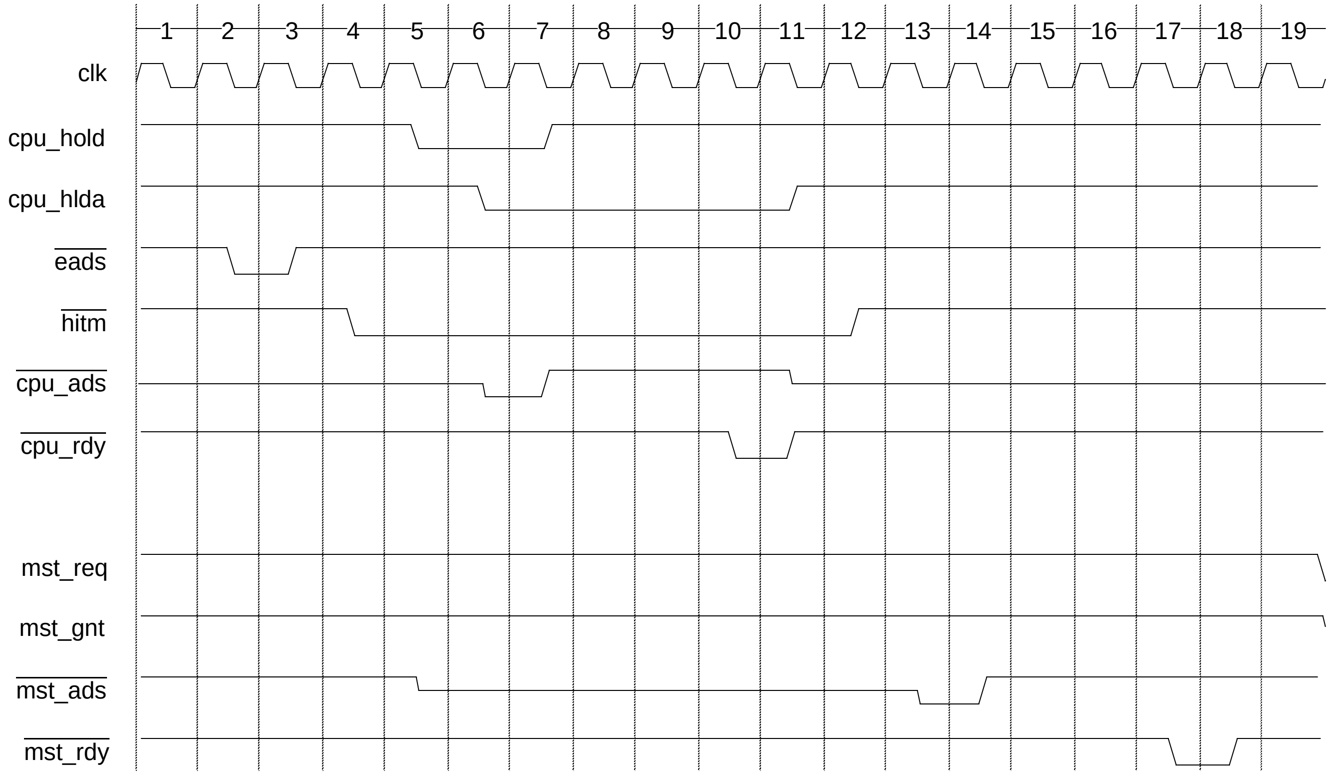
cpu_hold to the Am5$_x$86 CPU would remain asserted. In this example, another CPU bus master also requests the bus by asserting mst_req.

- **Clock #3**: The Am5$_x$86 CPU samples cpu_hold deasserted and deasserts cpu_hlda to take ownership of the bus. The Am5$_x$86 CPU begins a cycle by asserting $\overline{\text{cpu\_ads}}$.
- **Clock #4**: The CPU bus arbiter samples $\overline{\text{cpu\_ads}}$ asserted and rearbitrates. The CPU bus arbiter determines that the bus will be granted to another master (CPU bus master) when the current cycle is done, so it asserts cpu_hold to the Am5$_x$86 CPU. The Am5$_x$86 CPU will maintain ownership of the bus until it asserts cpu_hlda.
- **Clock #8**: The Am5$_x$86 CPU samples $\overline{\text{cpu\_rdy}}$ asserted, which ends the current cycle. The Am5$_x$86 CPU has also sampled cpu_hold asserted and surrenders the bus by asserting cpu_hlda. The Am5$_x$86 CPU has another cycle pending, so it asserts cpu_breq to request access to the CPU bus.
- **Clock #9**: The CPU bus arbiter samples cpu_hlda asserted from the Am5$_x$86 CPU and grants the bus to the CPU bus master (the next master in the queue) by asserting mst_gnt to the CPU bus master.
- **Clock #10**: The CPU bus master samples mst_gnt asserted and begins a cycle by asserting $\overline{\text{mst\_ads}}$.
- **Clock #11**: The CPU bus arbiter samples $\overline{\text{mst\_ads}}$ asserted and rearbitrates. The CPU bus arbiter determines that the bus will be granted to the Am5$_x$86 CPU when the current cycle is done, so it deasserts mst_gnt to the CPU bus master. The CPU bus master will maintain ownership of the bus until it deasserts mst_req.
- **Clock #15**: The CPU bus master samples $\overline{\text{mst\_rdy}}$ asserted, which ends the current cycle. The CPU bus master also samples mst_gnt deasserted and surrenders the bus by deasserting mst_req.
- **Clock #16**: The CPU bus arbiter samples mst_req deasserted from the CPU bus master, and grants the bus to the Am5$_x$86 CPU by deasserting cpu_hold.

8.4.4.2 CPU Bus Cache Write-Back

Figure 8-7 shows an Am5$_x$86 CPU cache write-back cycle. The cache must be written back when another CPU bus master accesses a memory location that has been modified in the internal Am5$_x$86 CPU cache only (the external memory contains invalid data).

Figure 8-7 CPU Bus Cache Write-Back


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
clk
cpu_hold
cpu_hlda
$\overline{\text{eads}}$
$\overline{\text{hitm}}$
$\overline{\text{cpu\_ads}}$
$\overline{\text{cpu\_rdy}}$
mst_req
mst_gnt
$\overline{\text{mst\_ads}}$
$\overline{\text{mst\_rdy}}$

Notes:

In Figure 8-7, the CPU bus master signals are labeled mst_xxxx and the Am5$_X$86 CPU signals are labeled cpu_xxxx.

The additional internal CPU bus interface signals shown in Figure 8-7 for write-back cycles are

- *$\overline{\text{eads}}$: External Address Strobe—Asserted by the CPU bus master to initiate a snoop by the Am5$_X$86 CPU.*
- *$\overline{\text{hitm}}$: Hit Modified Line—CPU must write back cache line to maintain coherency.*

The clk signal denotes the 33-MHz clock source and represents both the CPU clock and the PCI clock. This diagram does not represent the full synchronization of signals between these clock domains.

The following sequence annotates the CPU bus cache write-back cycle shown in Figure 8-7.

- **Clock #1**: The CPU bus master owns the bus (CPU bus master mst_gnt is asserted, Am5$_X$86 CPU cpu_hold/cpu_hlda are asserted).
- **Clock #2**: The CPU bus master initiates an inquire cycle by asserting $\overline{\text{eads}}$ to the Am5$_X$86 CPU.
- **Clock #4**: The Am5$_X$86 CPU asserts $\overline{\text{hitm}}$ to signal that the snoop resulted in a hit to a modified line. The Am5$_X$86 CPU must perform a write-back cycle to maintain coherency.
- **Clock #5**: The CPU bus master samples $\overline{\text{hitm}}$ asserted and relinquishes the bus on the next clock. The CPU bus arbiter deasserts cpu_hold to the Am5$_X$86 CPU to allow the Am5$_X$86 CPU to perform the write-back cycle.
- **Clock #6**: The Am5$_X$86 CPU samples cpu_hold deasserted and deasserts cpu_hlda to take ownership of the bus. The $\overline{\text{cpu\_ads}}$ signal is asserted to begin the write-back cycle.
- **Clock #7**: The CPU bus arbiter samples $\overline{\text{cpu\_ads}}$ asserted and asserts cpu_hold to the Am5$_X$86 CPU so that no additional cycles are generated after the write-back cycle.

- **Clock #11**: The Am5$_x$86 CPU samples $\overline{\text{cpu\_rdy}}$, which ends the write-back cycle. The Am5$_x$86 CPU has also sampled cpu_hold asserted and surrenders the bus by asserting cpu_hlda.

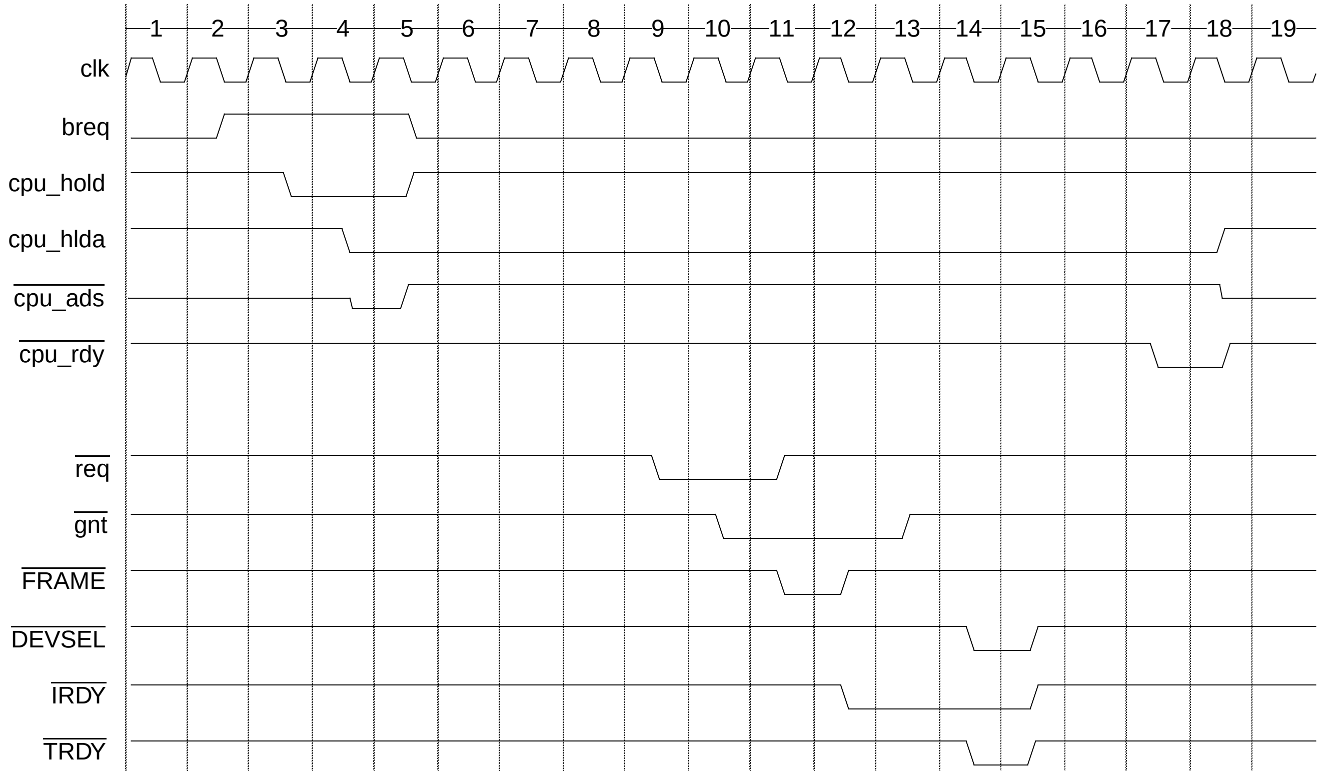
Note: *This write-back cycle is for illustration purposes only; the actual write-back cycle would consist of multiple data phases.*

- **Clock #12**: The Am5$_x$86 CPU deasserts $\overline{\text{hitm}}$ one clock after $\overline{\text{cpu\_rdy}}$ ends the write-back cycle.
- **Clock #13**: The CPU bus master samples $\overline{\text{hitm}}$ deasserted and starts the bus cycle.

8.4.4.3 CPU-to-PCI Cycle

Figure 8-8 shows an Am5$_x$86 CPU-to-PCI bus cycle. The Am5$_x$86 CPU cycle is either a read cycle or a write cycle with write posting disabled.

Figure 8-8 CPU-to-PCI Cycle



Notes:

The clk signal denotes the 33-MHz clock source and represents both the CPU clock and the PCI clock. This diagram does not represent the full synchronization of signals between these clock domains.

The following sequence annotates the Am5$_x$86 CPU-to-PCI cycle shown in Figure 8-8.

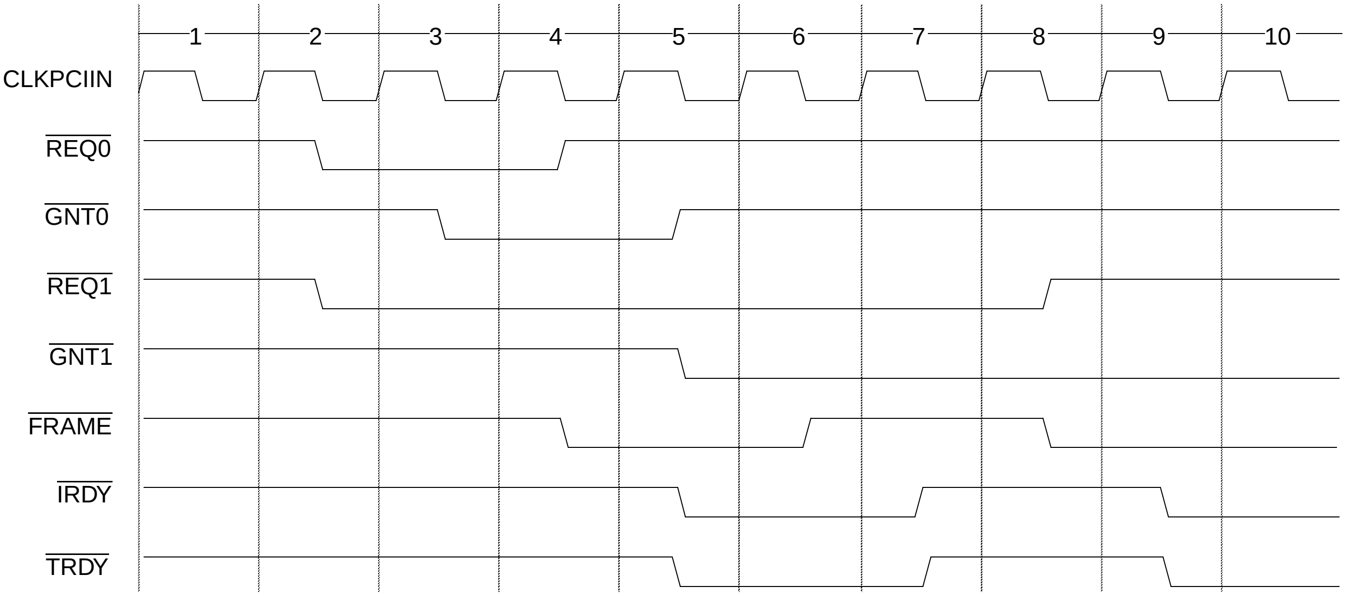
- **Clock #2**: The Am5$_x$86 CPU asserts breq to request the CPU bus. The CPU bus arbiter will grant the bus to the CPU when the current bus owner's cycle is completed. The pending Am5$_x$86 CPU cycle is to PCI.
- **Clock #3**: The CPU bus arbiter deasserts cpu_hold to the Am5$_x$86 CPU to grant the bus to the Am5$_x$86 CPU. The deassertion of cpu_hold would be delayed if the CPU bus was not idle or if another higher priority master was requesting the CPU bus.

- **Clock #4**: The cpu_hlda signal is deasserted by the Am5$_X$86 CPU to take ownership of the CPU bus, and $\overline{\text{cpu\_ads}}$ is asserted to begin a cycle to PCI.
- **Clock #5**: The CPU bus arbiter samples $\overline{\text{cpu\_ads}}$ asserted and rearbitrates. In this example, a higher priority master is requesting the bus, so cpu_hold is asserted to the Am5$_X$86 CPU. The Am5$_X$86 CPU maintains ownership of the CPU bus until it completes its cycle and asserts cpu_hlda.
- **Clock #9**: The host bridge asserts its $\overline{\text{req}}$ to the PCI bus arbiter in response to the Am5$_X$86 CPU bus cycle to PCI.
- **Clock #10**: The PCI bus arbiter asserts $\overline{\text{gnt}}$ to the host bridge. The assertion of $\overline{\text{gnt}}$ would be delayed if the bus was not idle or if another higher priority master was requesting the PCI bus.
- **Clock #11**: The host bridge samples $\overline{\text{gnt}}$ asserted and begins the PCI transaction.
- **Clock #17**: The PCI transaction is complete and the host bridge returns $\overline{\text{cpu\_rdy}}$ to the Am5$_X$86 CPU ending the Am5$_X$86 CPU-to-PCI cycle.
- **Clock #18**: The Am5$_X$86 CPU samples $\overline{\text{cpu\_rdy}}$ asserted ending the current cycle and asserts cpu_hlda to allow the next CPU bus master access to the CPU bus.

8.4.4.4 PCI Bus Arbitration

Figure 8-9 shows how the PCI bus arbiter arbitrates between two masters. Although there are only two PCI masters in this example, the mechanism is the same when there are more PCI masters. The differences are that there would be more $\overline{\text{REQ}}$/$\overline{\text{GNT}}$ signal pairs and more than one PCI bus transaction could take place before an individual PCI master is granted the bus.

Figure 8-9 PCI Bus Arbitration



The following sequence annotates the PCI bus arbitration cycle shown in Figure 8-9.

- **Clock #2**: Master 0 and master 1 simultaneously request access to the bus.

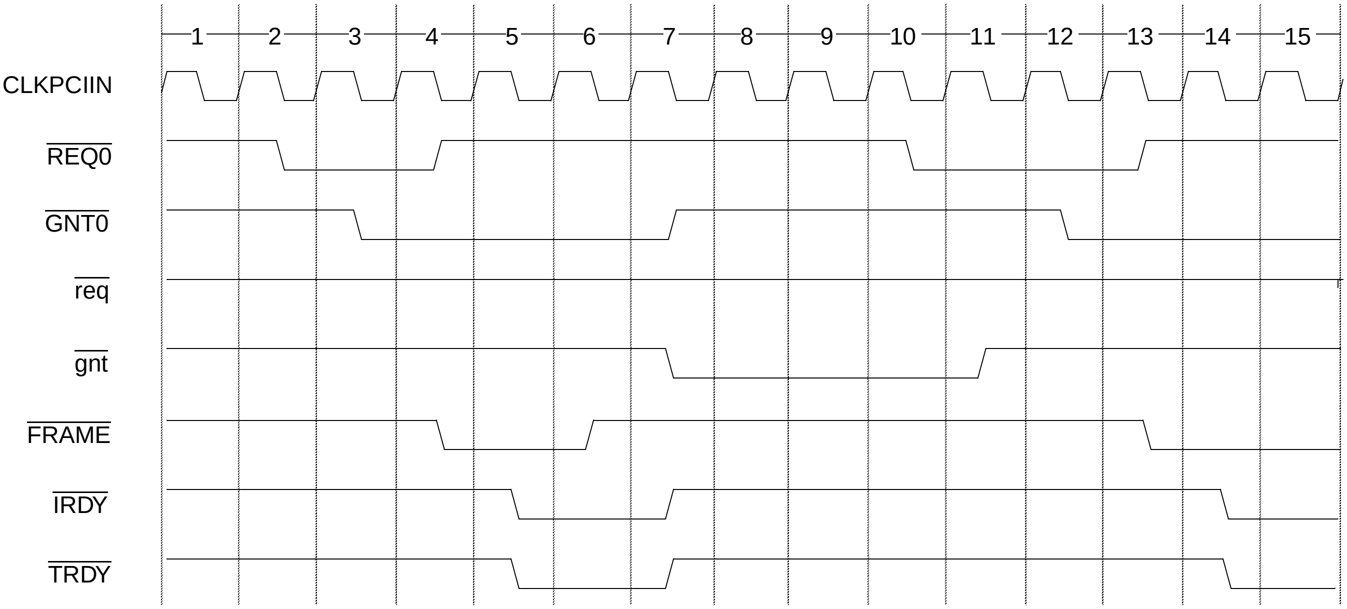
- **Clock #3**: The PCI bus arbiter samples $\overline{\text{REQ}}$ asserted and begins arbitration. Master 0 has higher priority at this time than master 1 so the PCI bus arbiter grants the PCI bus to master 0.
- **Clock #4**: Master 0 samples the bus idle and its $\overline{\text{GNT0}}$ signal asserted and begins a transaction by asserting $\overline{\text{FRAME}}$. Master 0 now becomes the lowest priority master in the rotating priority queue.
- **Clock #5**: The PCI bus arbiter detects a transaction has started and rearbitrates for the next master. Master 1 is the now the highest priority master in the rotating priority queue, so the PCI bus arbiter deasserts the $\overline{\text{GNT0}}$ for master 0 and asserts the $\overline{\text{GNT1}}$ for master 1.
- **Clock #8**: Master 1 samples the bus idle and its $\overline{\text{GNT1}}$ asserted and begins a transaction by asserting $\overline{\text{FRAME}}$. Master 1 now becomes the lowest priority master in the rotating priority queue.
- **Clock #9**: No other masters are requesting the bus, so the PCI bus arbiter keeps asserting the $\overline{\text{GNT1}}$ for master 1. This allows master 1 to continue the transaction, even after its master latency timer has expired. If another master were requesting the bus, the PCI bus arbiter would rearbitrate, deassert the $\overline{\text{GNT1}}$ for master 1, and assert the $\overline{\text{GNT}}$ for the next master to be granted the bus.

8.4.4.5 PCI Bus Arbitration Parking

Figure 8-10 shows an example of bus parking in concurrent arbitration mode when no master is requesting access to the PCI bus.

In this example, the PCI bus arbiter is configured to park on the Am5$_x$86 CPU. If the PCI bus arbiter is configured to park on the last master that acquired the bus, then the PCI bus arbiter would continue to assert the $\overline{\text{GNT}}$ to the master that had just completed a transaction.

Figure 8-10 PCI Bus Concurrent Mode Arbitration Parking



Notes:

In Figure 8-10, $\overline{\text{req}}$/$\overline{\text{gnt}}$ are for the Am5$_x$86 CPU.

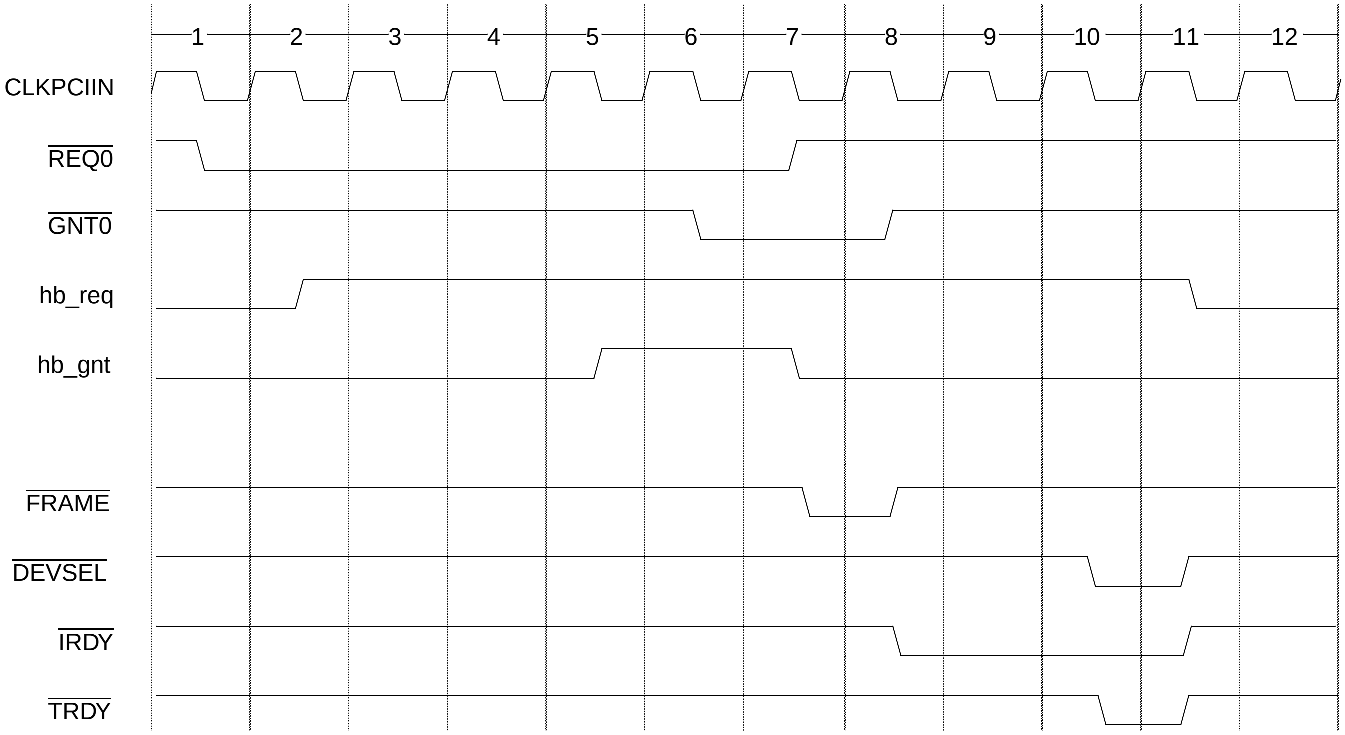
The following sequence annotates the PCI bus concurrent mode arbitration parking cycle shown in Figure 8-10.

- **Clock #2**: Master 0 requests access to the bus.
- **Clock #3**: The PCI bus arbiter samples REQ asserted and begins arbitration. Master 0 is the only master requesting the bus, so the PCI bus arbiter grants the bus to master 0 by asserting GNT0.
- **Clock #4**: Master 0 samples the bus idle and its GNT0 asserted, and begins a transaction by asserting FRAME. Master 0 now becomes the lowest priority master in the rotating priority queue.
- **Clock #5**: The PCI bus arbiter detects a transaction has started and begins to rearbitrate for the next master. Because no other masters are requesting the bus, the PCI bus arbiter keeps asserting the GNT0 for master 0. This allows master 0 to continue a transaction even after its master latency timer has expired. If another master were requesting the bus, the PCI bus arbiter would rearbitrate, deassert the GNT0 for master 0, and assert the GNT for the next master to be granted the bus.
- **Clock #7**: Master 0 samples the end of the transaction. The PCI bus arbiter samples FRAME deasserted, signaling that this is the last data phase of the transaction. Because no other masters are requesting the bus, the PCI bus arbiter will now park the bus on the configured master (Am5$_x$86 CPU). The PCI bus arbiter deasserts GNT0 to master 0 and asserts gnt to the Am5$_x$86 CPU. Note that req is not asserted. If the PCI bus arbiter was configured to park on the last master that acquired the bus, it would keep GNT0 asserted and park on master 0.
- **Clock #8**: The Am5$_x$86 CPU samples the bus idle and its gnt asserted. Note the Am5$_x$86 CPU does not have to start a transaction, but it does need to drive the shared PCI bus signals to stable values. If the Am5$_x$86 CPU wants to start a transaction, it does not have to assert req and wait for gnt. It can assert FRAME and begin a transaction on any clock it samples gnt asserted. The master on which the PCI bus is parked has no arbitration latency.
- **Clock #10**: Master 0 requests the bus by asserting REQ0.
- **Clock #11**: The PCI bus arbiter samples REQ asserted and begins arbitration. Master 0 is the only master requesting the bus, so the PCI bus arbiter determines that master 0 will be the next master to be granted the bus. The PCI bus arbiter then deasserts gnt to the Am5$_x$86 CPU.
- **Clock #12**: The PCI bus arbiter asserts GNT0. Note the PCI bus arbiter cannot simultaneously deassert one master's GNT and assert another master's GNT when the bus is idle. Doing so could cause contention on the shared PCI bus signals.
- **Clock #13**: Master 0 samples the bus idle and its GNT0 signal asserted and begins a transaction by asserting FRAME. Master 0 now becomes the lowest priority master in the rotating priority queue. Note that there is a two-clock arbitration latency for masters that are not parked on the bus when the bus is idle. This is because, when the bus is idle, one GNT cannot be asserted on the same clock when another GNT is deasserted. Therefore, GNT to the master the bus is parked on will be deasserted in one clock, and the GNT to the next master granted the bus will be asserted one clock later, resulting in a two-clock arbitration latency.

8.4.4.6 Nonconcurrent Mode Arbitration

Figure 8-11 shows external PCI master arbitration in nonconcurrent mode. In nonconcurrent arbitration mode, both the CPU bus and the PCI bus are granted to the PCI master, regardless of the destination of the PCI transaction.

Figure 8-11 Nonconcurrent Mode Arbitration



Notes:

The diagram includes the following internal signals:

- *hb_req: PCI host bridge requesting the Am5$_X$86 CPU bus.*
- *hb_gnt: PCI host bridge has been granted Am5$_X$86 CPU bus.*

The following sequence annotates the nonconcurrent mode arbitration cycle shown in Figure 8-11.

- **Clock #1**: An external PCI master requests the PCI bus.
- **Clock #2**: The PCI bus arbiter samples an external PCI request asserted and asserts the host bridge request to the CPU bus arbiter. The external PCI master $\overline{\text{GNT0}}$ cannot be asserted until the host bridge is granted the CPU bus. If the system arbiter were operating in concurrent arbitration mode, the external PCI master $\overline{\text{GNT0}}$ could be asserted in clock #2 because the PCI bus and the CPU bus would be operating independently.
- **Clock #5**: The CPU bus arbiter has determined the host bridge will be granted the CPU bus and asserts hb_gnt to the host bridge. The assertion of hb_gnt could be delayed if a higher priority master was requesting the CPU bus.
- **Clock #6**: The PCI bus arbiter detects the host bridge has been granted the CPU bus and asserts $\overline{\text{GNT0}}$ to the external PCI master.
- **Clock #7**: The CPU bus arbiter rearbitrates and determines another CPU bus master will be granted the bus and deasserts hb_gnt to the host bridge. The host bridge will

maintain ownership of the CPU bus until it deasserts hb_req. The external PCI master samples GNT0 asserted and asserts FRAME to begin the PCI transaction.

- **Clock #8**: GNT0 is deasserted because either the external master is parked on the CPU or another master has requested the bus.
- **Clock #11**: The host bridge samples the end of the PCI transaction and has sampled hb_gnt deasserted, so it deasserts hb_req to allow the next CPU bus master access to the CPU bus.

8.4.5 Interrupts

The system arbiter has one interrupt signal routed to the ÉlanSC520 microcontroller's PCI host bridge. This interrupt source shares the interrupt controller input used by any PCI host bridge interrupts that are enabled in the Host Bridge Master Interrupt Control (HBMSTIRQCTL) register (MMCR offset 66h) register and the Host Bridge Target Interrupt Control (HBTGTIRQCTL) (MMCR offset 62h) register.

The following condition can be programmed to generate an interrupt from the system arbiter.

- When the PCI bus arbiter has asserted a GNT in response to a request (the bus is not parked) and a PCI transaction was not started within 16 clocks after the bus became idle, per the *PCI Local Bus Specification,* Revision 2.2.

The GNT_TO_INT_ENB bit in the System Arbiter Control (SYSARBCTL) register (MMCR offset 70h) is used to enable interrupts that are generated when the PCI bus arbiter detects a grant time-out. Before the GNT_TO_INT_ENB bit is set, the PCI Host Bridge Interrupt Mapping (PCIHOSTMAP) register (MMCR offset D14h) must be configured to route the interrupt to the appropriate interrupt request level and priority.

The REQ/GNT number of the PCI master that did not start a transaction is reported in the GNT_TO_STA bit of the PCI Bus Arbiter Status (PCIARBSTA) register (MMCR offset 71h). Note that the GNT_TO_STA bit is set on PCI bus arbiter grant time-outs regardless of the GNT_TO_INT_ENB bit value.

8.4.6 Software Considerations

The system arbiter can operate in concurrent or nonconcurrent arbitration mode (see "Operating Modes" on page 8-3). Write posting from the CPU to the PCI bus should be disabled while configured in nonconcurrent arbitration mode. When changing between nonconcurrent and concurrent arbitration mode, all system arbiter requests should be disabled, as follows:

- GP-DMA channels should be disabled to prevent the DMA controller from requesting the CPU bus.
- External PCI bus master requests should be inhibited.
- The Am5$_x$86 CPU should not attempt to access the PCI bus.

A PCI bus master that does not start a transaction within 16 clocks after the bus is idle can be considered broken. The PCI bus arbiter checks for this condition and provides status on which PCI bus master GNT was asserted when this condition was detected. Software can read this status and disable the broken master's REQ to the PCI bus arbiter through the System Arbiter Master Enable (SYSARBMENB) register (MMCR offset 72h). This prevents the broken master from wasting PCI bandwidth.

Note that the PCI bus arbiter does not automatically disable the broken master's REQ signal.

8.4.7 Latency

Because the PCI bus is shared by many masters, each master incurs a latency accessing the bus due to other masters. This latency is determined by each master in the system and the arbitration algorithm. The latency contributed by each master is controlled through its associated master latency timer, which limits the amount of time a master is allowed for each transaction. When this timer expires, the current master must end its transaction and allow another master access to the bus.

The ÉlanSC520 microcontroller PCI bus arbiter has two rotating priority queues and an $Am5_x86$ CPU relative priority. The $Am5_x86$ CPU does not burst on PCI, and therefore does not have a master latency timer. The longest transaction for the $Am5_x86$ CPU is 16 PCI clocks.

The latency contributed by the ÉlanSC520 microcontroller PCI bus arbiter can be controlled in the Arbiter Priority Control (ARBPRICTL) register (MMCR offset 74h) through the use of the high-priority queue and the relative $Am5_x86$ CPU priority configuration.

8.4.7.1 Simple Rotating Priority Latency

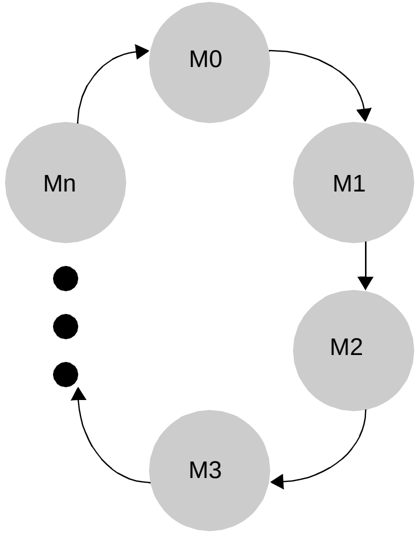
In a simple one-level rotating priority queue, the maximum latency for each master would be the sum of all the other master latency timers in the system.

In Figure 8-12, the maximum latency for master M0 would be the sum of the longest possible transactions for masters M1, M2, M3, ..., Mn. The longest transaction for each master is limited by its associated master latency timer, so the maximum latency for M0 would be:

> master latency timer for M1 + master latency timer for M2 + master latency timer for M3 + ... + master latency timer for Mn

This latency would be seen by M0 when it had just completed a transaction, all other masters were requesting access to the bus, and each master required the bus for the entire duration of its associated master latency timer.

Figure 8-12 Simple Rotating Priority Queue



8.4.7.2 High-Priority Queue Latency

The maximum latency for a master in the high-priority queue is the sum of:

- Master latency timer of other master in high-priority queue—This time can be decreased by decreasing the master latency timer of the other master in the high-priority queue, or this time can be eliminated by programming only one master in the high-priority queue.
- Longest master latency timer of all masters in the low-priority queue—This can be decreased by decreasing the master latency timer of all masters in the low-priority queue.
- 3 * (Am5$_X$86 CPU maximum transaction time)

8.4.7.3 Low-Priority Queue Latency

The maximum latency for a master in the low-priority queue (note that after a low-priority master has completed a transaction, every PCI master will be granted the bus before the low-priority master will be granted the bus again) is the sum of:

- Number of external masters * (Am5$_X$86 CPU maximum transaction time)—The Am5$_X$86 CPU maximum transaction time is multiplied by the number of external masters, because the Am5$_X$86 CPU is granted the bus after every external PCI transaction if the Am5$_X$86 CPU relative priority is configured for one external PCI master cycle. This can be decreased by decreasing the Am5$_X$86 CPU relative priority (configure the relative priority to allow more external PCI cycles for every Am5$_X$86 CPU PCI cycle).
- Number of masters in the low-priority queue * (master latency timers of all masters in the high-priority queue)—The master latency timers of all masters in the high-priority queue is multiplied by the number of masters in the low-priority queue, because the high-priority masters are granted the bus after each low-priority master grant. This time can be decreased by decreasing the number of masters in the high-priority queue or by decreasing the master latency timers of the masters in the high-priority queue.
- Master latency timers of all masters in the low-priority queue—This time can be decreased by decreasing the master latency timers of the masters in the low-priority queue.

8.4.7.4 CPU Latency

The maximum latency for the Am5$_X$86 CPU is:

- 3 * (longest master latency timer of all external masters)—The master latency timer is multiplied by 3 because the worst case is when the Am5$_X$86 CPU relative priority is configured for three external PCI master cycles for every Am5$_X$86 CPU PCI cycle. This time can be decreased by decreasing the master latency timers of external masters or by increasing the Am5$_X$86 CPU relative priority.

8.4.7.5 Nonconcurrent Arbitration Mode Latency

Operating in nonconcurrent arbitration mode adds to the PCI bus latency. In nonconcurrent arbitration mode, all PCI masters must be granted the CPU bus in addition to the PCI bus before a transaction can proceed. The time associated with being granted the CPU bus adds to each PCI master's latency.

The maximum latency is:

(time for the longest Am5$_X$86 CPU transfer) + (time for the longest GP-DMA transfer)

The longest Am5$_X$86 CPU transfer is one cache line, and the longest GP-DMA transfer is programmable. This additional latency is added to the latency of each external PCI master as calculated in the high-priority and low-priority queues. This latency is incurred for all PCI transactions, not only transactions where the ÉlanSC520 microcontroller is the PCI target. Note that this includes PCI bus transactions where both the master and the target are external PCI bus agents.

8.4.7.6 Concurrent Arbitration Mode Latency

The CPU bus adds to the PCI bus latency even when operating in concurrent arbitration mode. Buffering in the host bridge, however, decreases the amount of latency on the PCI bus due to the CPU bus. PCI transactions where the ÉlanSC520 microcontroller is not the target do not have any added latency due to the CPU bus.

PCI write transactions where the ÉlanSC520 microcontroller is the target are posted in the host bridge. The data is not immediately written to SDRAM, but have some latency due to CPU bus arbitration. The external PCI master transaction, however, will be completed, and so the external PCI master will not see this additional latency.

PCI read transactions where the ÉlanSC520 microcontroller is the target can be delayed transactions. In this case, the external PCI master requesting the data sees the latency added by the CPU bus arbitration.

Other PCI transactions are allowed on the PCI bus while the host bridge is arbitrating for the CPU bus, and so only the external PCI master requesting the data incurs the CPU bus latency, not the whole PCI bus. Note that CPU bus latency is added only to external PCI master read transactions where the ÉlanSC520 microcontroller is the target.

8.4.7.7 Concurrent Arbitration Mode Bus Parking Latency

There is some latency associated with bus parking. The master that is parked on the bus is able to begin a transaction immediately (without having to assert $\overline{\text{REQ}}$), because its $\overline{\text{GNT}}$ is already asserted. All other masters have to arbitrate for the bus by asserting $\overline{\text{REQ}}$ and waiting for $\overline{\text{GNT}}$. This arbitration takes two PCI clocks (see "PCI Bus Arbitration Parking" on page 8-16). This applies to concurrent mode arbitration only.

8.5 INITIALIZATION

The system arbiter logic and configuration is reset in response to system reset.

After reset, the system arbiter operates in nonconcurrent arbitration mode. The priority queue is defaulted such that $\overline{\text{REQ0}}$ is the highest priority and $\overline{\text{REQ4}}$ is the lowest priority, because no masters are configured in the high-priority queue at this time. All masters are disabled at reset, with the exception of the CPU as a PCI and CPU bus master.

After reset, the following initialization steps are required:

1. Enable concurrent operating mode, if desired, by setting the CNCR_MODE_ENB bit in the System Arbiter Control (SYSARBCTL) register (MMCR offset 70h). System arbitration defaults to nonconcurrent arbitration mode after reset. Note that changing the CNCR_MODE_ENB bit should only be done when all bus master requests are disabled.
2. Configure PCI bus parking with the BUS_PARK_SEL bit in the System Arbiter Control (SYSARBCTL) register. Note that the BUS_PARK_SEL bit should only be changed when the PCI bus is currently parked on the CPU. By default, the PCI bus arbiter parks on the Am5$_X$86 CPU, but the arbiter can be programmed to park on the last active PCI bus master if operating in concurrent arbitration mode.
3. Configure PCI bus arbiter priority in the Arbiter Priority Control (ARBPRICTL) register (MMCR offset 74h) if any external PCI masters are to be configured in the high-priority queue. By default, all external masters are configured to be in the low-priority queue.

4. Enable external PCI requests to the PCI bus arbiter in the System Arbiter Master Enable (SYSARBMENB) register (MMCR offset 72h). By default, all external PCI bus master requests are disabled.
5. Enable/Clear the PCI bus $\overline{\text{GNT}}$ time-out interrupt with the GNT_TO_INT_ENB bit in the System Arbiter Control (SYSARBCTL) register, if desired. By default, this interrupt source is disabled, but the GNT_TO_ID status bit is set in the PCI Bus Arbiter Status (PCIARBSTA) register (MMCR offset 71h) if a PCI bus $\overline{\text{GNT}}$ time-out is detected.

CHAPTER

9 PCI BUS HOST BRIDGE

9.1 OVERVIEW

The ÉlanSC520 microcontroller includes an integrated PCI bus host bridge, which allows the microcontroller to interface with any PCI bus Revision 2.2-compliant master or target device.

The PCI host bridge includes the following features:

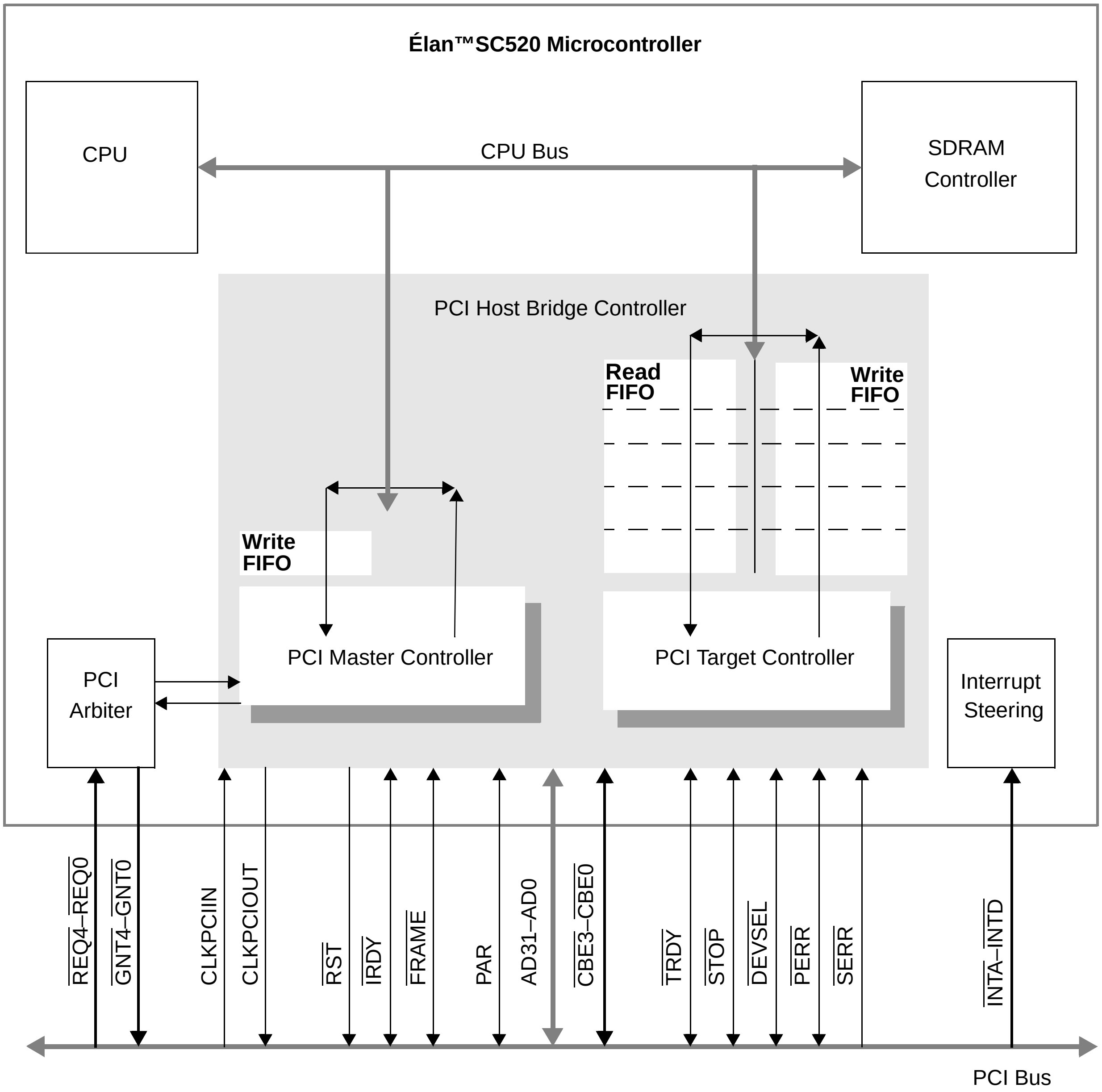
- 33 MHz, 32-bit PCI bus Revision 2.2-compliant
- Peak transfer rate of 132 Mbytes/s
- Support for delayed transactions improves PCI bus utilization
- Support for long bursts without disconnect when the ÉlanSC520 microcontroller is a target (64 doublewords for both reads and writes)
- Capable of zero wait state burst transfers as a target
- Support for advanced PCI bus commands as a target: memory-read-line, memory-read-multiple
- Flexible PCI bus interrupt steering logic
- Supports fast back-to-back transactions as a PCI bus target

According to the *PCI Local Bus Specification,* Revision 2.2, the initiator, or *master*, is the device that initiates the PCI transfer. The slave, or *target*, is the device being addressed by the master for the data transfer.

9.2 BLOCK DIAGRAM

The ÉlanSC520 microcontroller PCI host bridge interface is shown in Figure 9-1.

Figure 9-1 PCI Interface Block Diagram



9.3 SYSTEM DESIGN

Figure 9-2 shows how the ÉlanSC520 microcontroller can be connected to an external PCI bus *target* device.

Figure 9-3 on page 9-4 shows how the ÉlanSC520 microcontroller can be connected to an external PCI bus *master* device.

In each configuration, the PCI bus clock is driven from the ÉlanSC520 microcontroller on the CLKPCIOUT pin and may require external buffering due to system loading (see "PCI Clocking" on page 9-5). $\overline{\text{RST}}$, the PCI bus reset signal, is driven from the ÉlanSC520 microcontroller.

The optional PCI bus target device interrupts can be connected to the PCI bus interrupt pins on the ÉlanSC520 microcontroller ($\overline{\text{INTA}}$, $\overline{\text{INTB}}$, $\overline{\text{INTC}}$, $\overline{\text{INTD}}$) or any of the GPIRQ10–

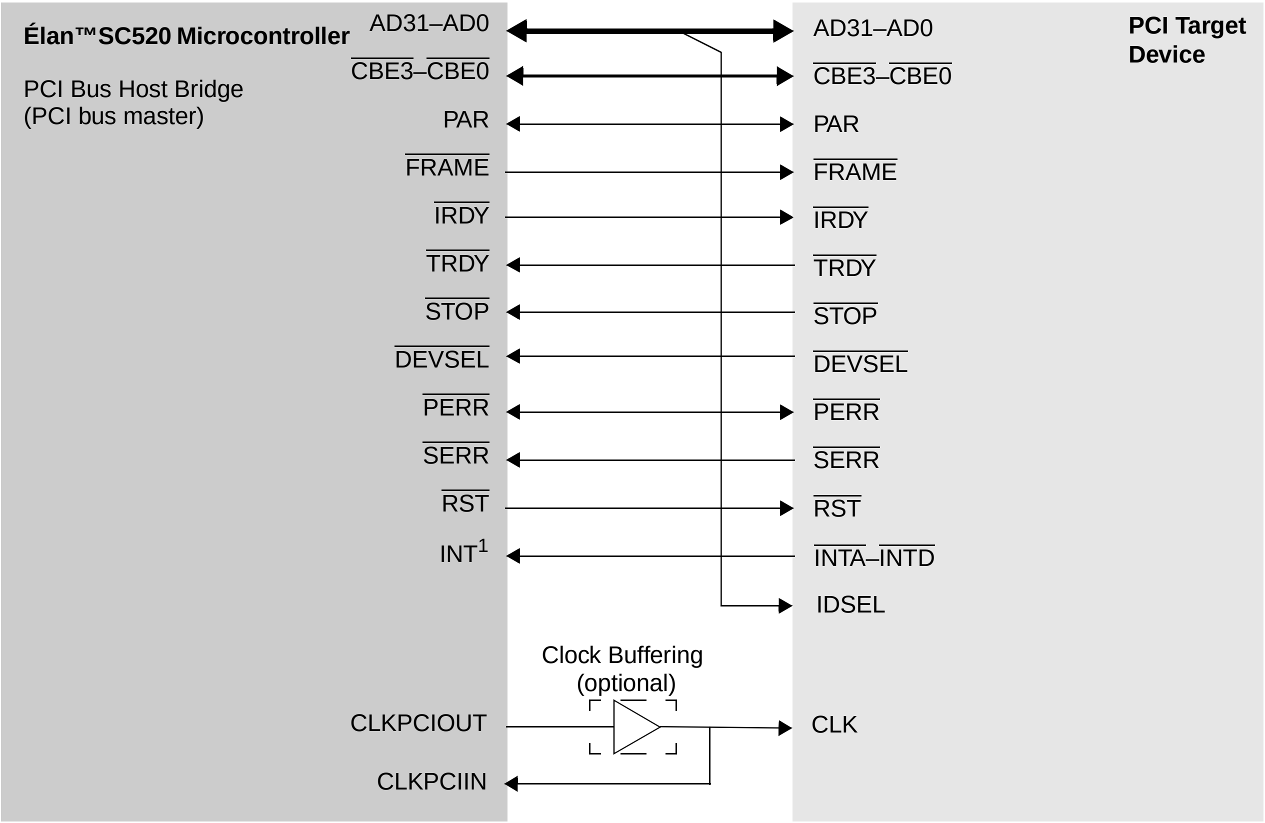
GPIRQ0 pins on the GP bus. See Chapter 15, “Programmable Interrupt Controller”, for further information on connecting interrupt requests to the ÉlanSC520 microcontroller.

Figure 9-4 on page 9-5 shows how the PERR and SERR signals are connected to the ÉlanSC520 microcontroller. PERR is driven by the PCI bus device (including the host bridge) that is receiving data (sampling the AD31–AD0 bus during data phases). SERR is driven by external PCI bus devices that detect a system error. External pullups must be provided for PERR and SERR.

The PCI bus input and output pins of the ÉlanSC520 microcontroller are PCI bus revision 2.2 compliant. See the PCI bus specification for information on physical loading and routing. The following PCI signals require pullups: FRAME, IRDY, TRDY, STOP, DEVSEL, PERR, and SERR. These pullups must be provided externally to the ÉlanSC520 microcontroller (the ÉlanSC520 microcontroller PCI bus pins do not have any termination).

The system PCI bus reset (RST) signal is sourced from the ÉlanSC520 microcontroller and is asynchronous to the PCI bus clock. See “Initialization” on page 9-29 for more information on reset.

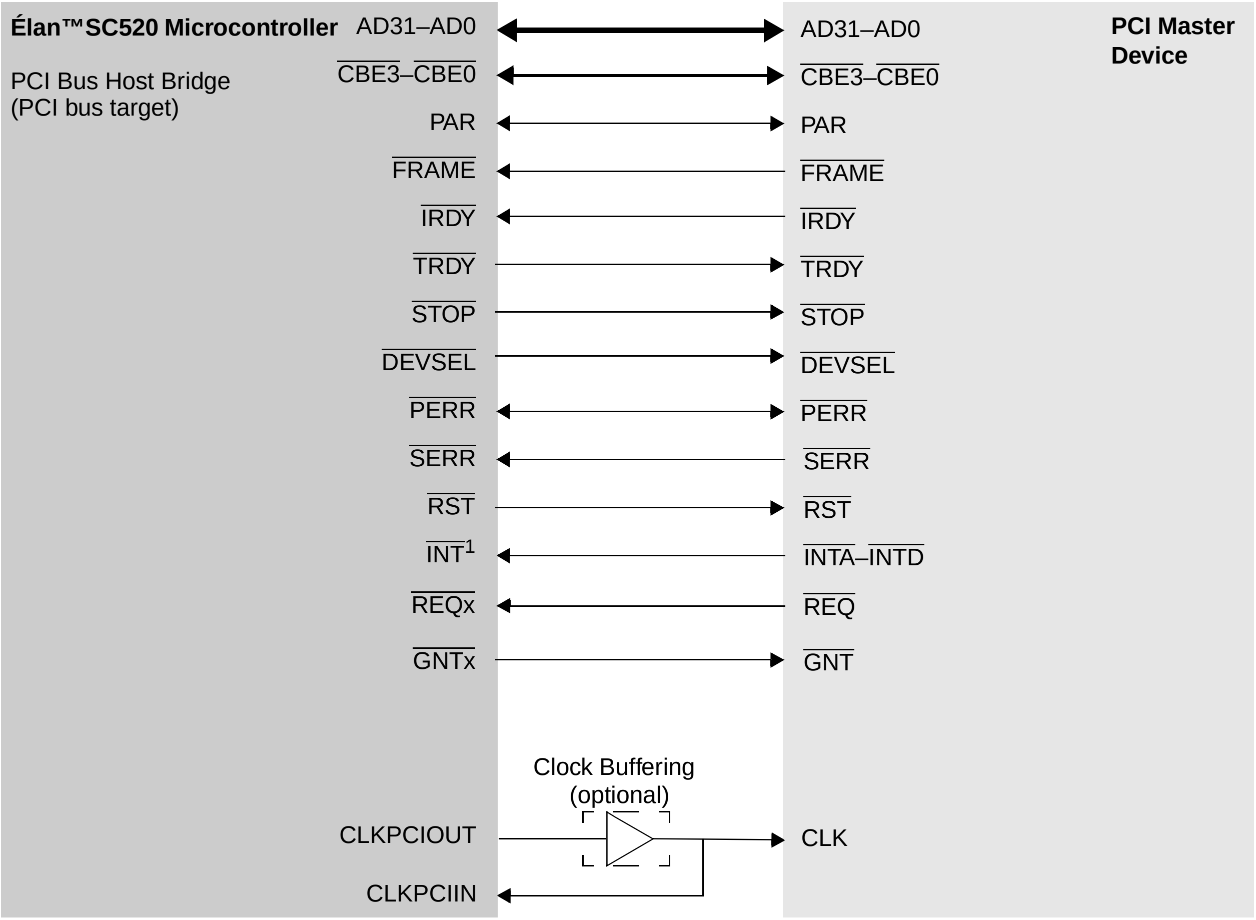
Figure 9-2 Élan™SC520 Microcontroller Connection to an External PCI Bus Target



Notes:

1. INT implies any of the following pins: INTA–INTD or GPIRQ10–GPIRQ0

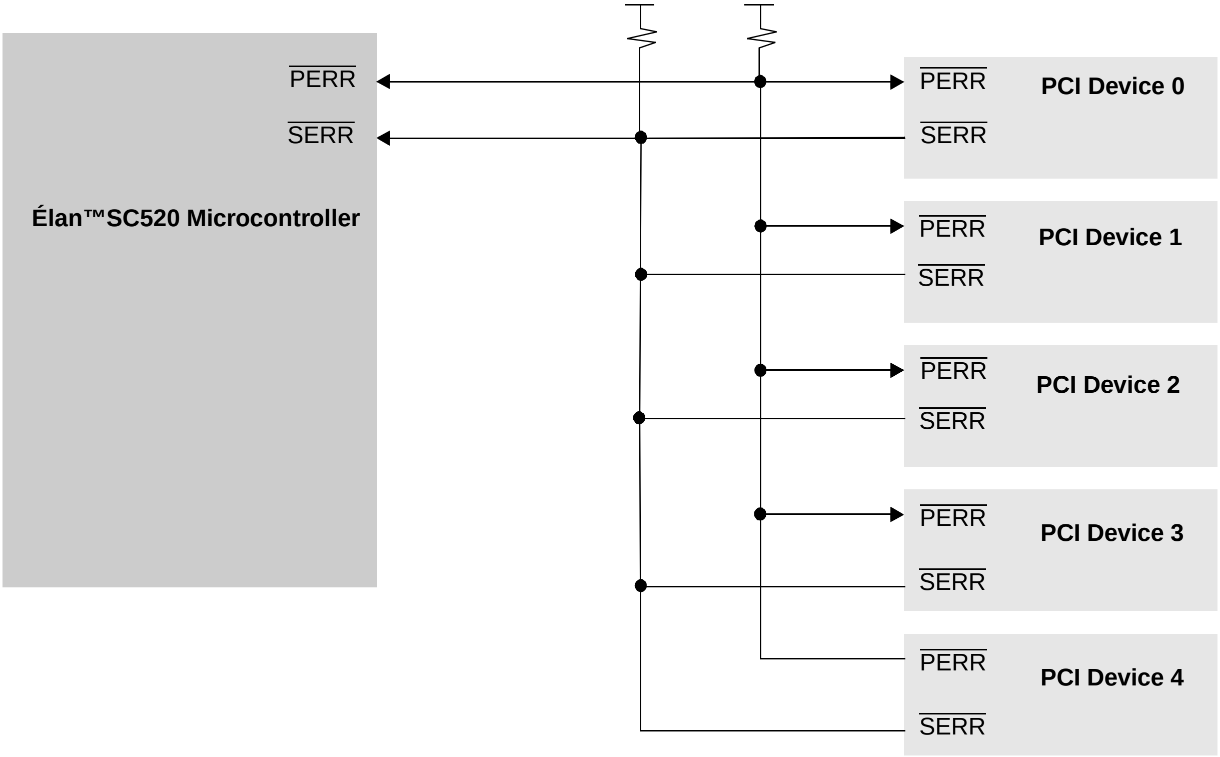
Figure 9-3 Élan™SC520 Microcontroller Connection to an External PCI Bus Master



Notes:

1. *INT implies any of the following pins: INTA–INTD or GPIRQ10–GPIRQ0*

Figure 9-4 Élan™SC520 Microcontroller $\overline{\text{SERR}}$ and $\overline{\text{PERR}}$ Connection



9.3.1 PCI Clocking

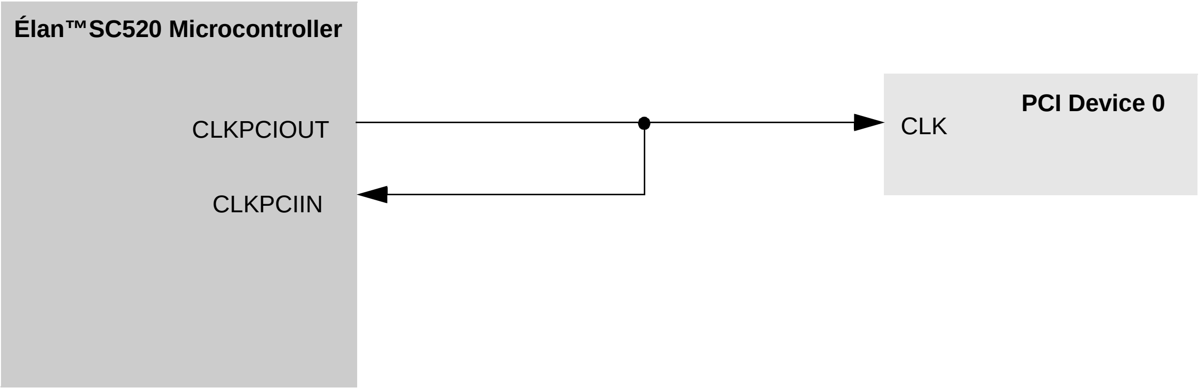
The system PCI bus clock (CLK) is sourced from the ÉlanSC520 microcontroller. There are two PCI bus clock pins on the ÉlanSC520 microcontroller: CLKPCIIN and CLKPCIOUT. The CLKPCIOUT output pin drives a 33-MHz clock that is used as the system PCI bus clock. However, the PCI host bridge logic is clocked from the CLKPCIIN input pin. The two pins are provided for the PCI bus clock to minimize clock skew between the PCI host bridge and external PCI bus devices.

The CLKPCIIN input pin guarantees that the PCI host bridge is driven with the same clock as the external PCI bus devices. Otherwise, external buffering and loading of the CLKPCIOUT pin could delay the clock, so that the skew between the PCI host bridge and external PCI bus devices would not meet the PCI bus specification.

External buffering of CLKPCIOUT may or may not be required, depending on the system loading (see Figure 9-5 and Figure 9-6). The ÉlanSC520 microcontroller does not dynamically slow down or stop the output CLKPCIOUT clock; therefore the PCI bus CLKRUN pin is not supported.

The CLKPCIIN pin is specifically intended for addressing the clock skew problem. It is *not* intended to enable running the PCI host bridge with a clock that is asynchronous to the CLKPCIOUT pin. Driving the CLKPCIIN pin from an external source that is of a different frequency is also not supported.

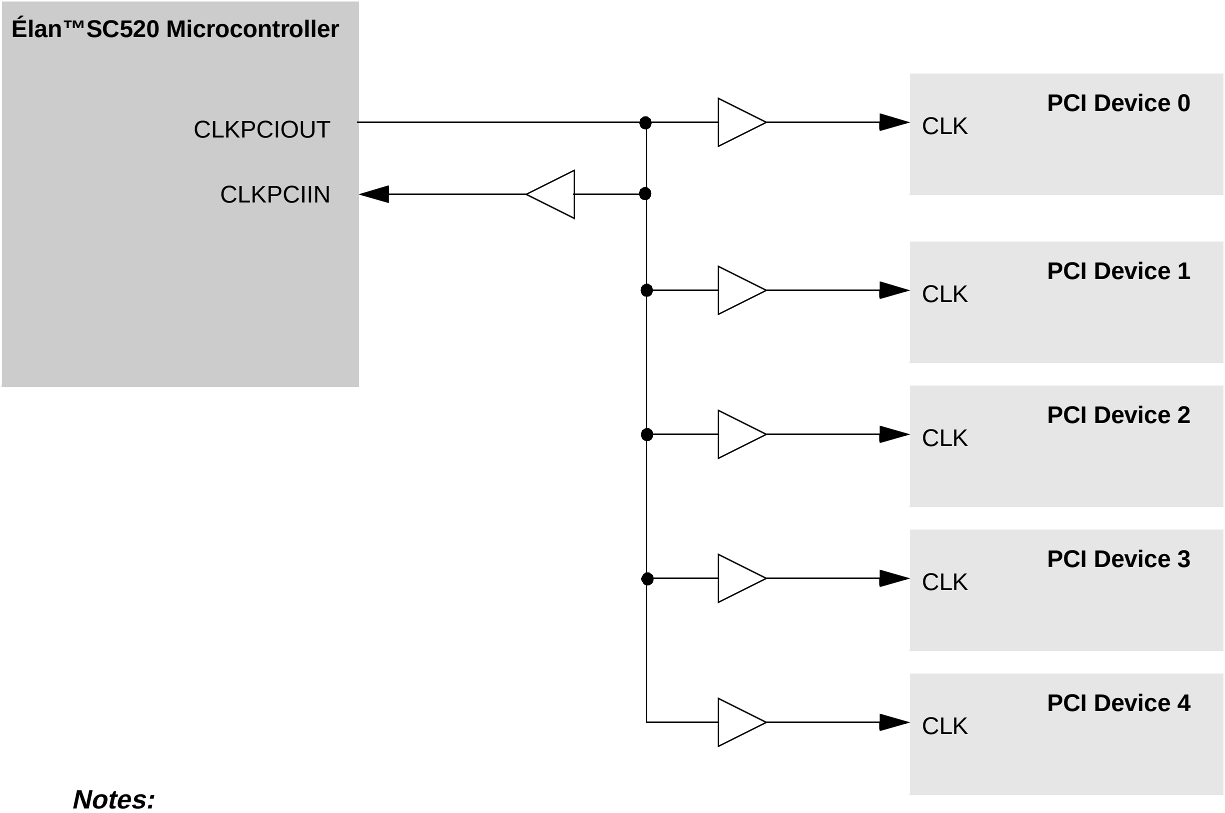
Figure 9-5 PCI Bus Clocking Example 1: Lightly Loaded System



Notes:

In this lightly loaded system, no clock buffering is required.

Figure 9-6 PCI Bus Clocking Example 2: Heavily Loaded System



Notes:

In this heavily loaded system, clock buffering is required.

9.3.1.1 Running the Élan™SC520 Microcontroller at 33.333 MHz

The clock that is supplied to the PCI bus (CLKPCIOUT) is exactly the same as the frequency of the crystal. The ÉlanSC520 microcontroller simply buffers the 33-MHz crystal input and provides it to the CLKPCIOUT pin. Because crystals have inaccuracies, it is possible that these inaccuracies cause the period of CLKPCIOUT to become marginally less than 30 ns.

It is up to the system designer to choose the accuracy of the crystal used with the ÉlanSC520 microcontroller. The 33.000-MHz frequency provides a better guard band than the 33.333-MHz crystal. In practice, most PCI devices tolerate both frequencies, but it is important to be aware of the impact of choosing the crystal on this potential violation of the PCI bus specifications. The PCI bus specification requires that the minimum clock period be 30 ns.

9.4 REGISTERS

The PCI host bridge configuration registers specific to the ÉlanSC520 microcontroller are memory-mapped in ÉlanSC520 microcontroller configuration space. These registers are listed in Table 9-1. Table 9-2 lists the direct-mapped registers used to configure the PCI bus host bridge. The standard PCI configuration space header registers supported on the ÉlanSC520 microcontroller are shown in Table 9-3 as PCI indexed registers.

Table 9-1 PCI Host Bridge Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
Host Bridge Control	HBCTL	60h	PCI reset, target FIFO purge enable, automatic delayed transaction enable, and master write posting enable
Host Bridge Target Interrupt Control	HBTGTIRQCTL	62h	Target interrupt or NMI select and interrupt enables: delayed transaction time-out, address parity, and data parity
Host Bridge Target Interrupt Status	HBTGTIRQSTA	64h	Target interrupt status: delayed transaction time-out, address parity, data parity; target interrupt identification
Host Bridge Master Interrupt Control	HBMSTIRQCTL	66h	Master interrupt or NMI select and interrupt enables: retry time-out, target abort, master abort, system error, received parity error, detected parity error
Host Bridge Master Interrupt Status	HBMSTIRQSTA	68h	Master interrupt status: retry time-out, target abort, master abort, system error, received parity error, detected parity error; master command interrupt identification
Host Bridge Master Interrupt Address	MSTINTADD	6Ch	Master address interrupt identification
Interrupt Pin Polarity	INTPINPOL	D10h	Polarity of external interrupt sources ($\overline{\text{INTA}}$–$\overline{\text{INTD}}$ and GPIRQ10–GPIRQ0)
PCI Host Bridge Interrupt Mapping	PCIHOSTMAP	D14h	System arbiter and PCI Host Bridge interrupt mapping to any of 22 available interrupt channels or NMI, PCI NMI enable control
PCI Interrupt A Mapping	PCIINTAMAP	D30h	PCI $\overline{\text{INTA}}$ mapping
PCI Interrupt B Mapping	PCIINTBMAP	D31h	PCI $\overline{\text{INTB}}$ mapping
PCI Interrupt C Mapping	PCIINTCMAP	D32h	PCI $\overline{\text{INTC}}$ mapping
PCI Interrupt D Mapping	PCIINTDMAP	D33h	PCI $\overline{\text{INTD}}$ mapping

Table 9-2 PCI Host Bridge Registers—Direct-Mapped

Register	Mnemonic	I/O Address	Function
PCI Configuration Address	PCICFGADR	0CF8h	PCI configuration space enable, bus number, device number, function number, register number
PCI Configuration Data	PCICFGDATA	0CFCh	PCI configuration data

Table 9-3 PCI Host Bridge Registers—PCI Indexed

Register	Mnemonic	I/O Address	Function
Device/Vendor ID	PCIDEVID	CF8h/CFCh Index 00h	Device identification, vendor identification
Status/Command	PCISTACMD	CF8h/CFCh Index 04h	Parity error detected, signalled system error, received master abort, received target abort, signalled target abort, $\overline{\text{DEVSEL}}$ timing, data parity reported, fast back-to-back capable, $\overline{\text{SERR}}$ enable, parity error response, master enable, memory access enable, I/O space enable
Class Code/Revision ID	PCICCREVID	CF8h/CFCh Index 08h	Base class code, sub-class code, program interface type, revision identification
Header Type	PCIHEADTYPE	CF8h/CFCh Index 0Eh	PCI configuration space header format
Master Retry Time-Out	PCIMRETRYTO	CF8h/CFCh Index 41h	PCI master retry time-out value

9.5 OPERATION

The PCI host bridge on the ÉlanSC520 microcontroller has the following functionality:

- **Master controller**—Allows the Am5$_X$86 CPU to be a master on the PCI bus. The Am5$_X$86 CPU can generate configuration transactions to configure the PCI host bridge, as well as all external devices on the PCI bus. The Am5$_X$86 CPU can also generate memory and I/O read and write transactions on the PCI bus.
- **Target controller**—Allows external PCI bus masters to access the ÉlanSC520 microcontroller's SDRAM.

9.5.1 Unsupported PCI Bus Functions

The following list summarizes some of the PCI bus functionality that is not supported in the ÉlanSC520 microcontroller's PCI host bridge. These functions are listed as optional in the PCI bus specification.

- 66 MHz is not supported.
- 64-bit data is not supported.
- 64-bit addressing (dual address cycles) is not supported due to the maximum 32-bit address space of the Am5$_X$86 CPU.
- Cacheable PCI bus memory (SBDONE, $\overline{\text{SBO}}$) is not supported.
- The optional $\overline{\text{CLKRUN}}$ pin is not supported.

- The $\overline{\text{LOCK}}$ pin is an optional pin not required in most systems, because other mechanisms are typically employed for coherency.
- Address/data stepping is not supported as a master due to the performance implications.
- The ÉlanSC520 microcontroller does not support a downstream “Southbridge” device, because most peripherals normally included in a Southbridge are integrated into the ÉlanSC520 microcontroller.
- The optional message-signalled interrupt feature described in the *PCI Local Bus Specification*, Revision 2.2, is not supported in the ÉlanSC520 microcontroller.

9.5.1.1 Unsupported PCI Bus Configuration Registers

Some standard PCI bus configuration registers are not implemented, because the ÉlanSC520 microcontroller is a host-to-PCI bridge and does not support some optional PCI functionality.

- Base Address registers are not implemented, because the ÉlanSC520 microcontroller is the host PCI device. Target address space configuration is done through ÉlanSC520 microcontroller-specific configuration (see “PCI Host Bridge Target Address Space” on page 9-18).
- Latency timer and MAX_LAT, MIN_GNT are not implemented, because the ÉlanSC520 microcontroller’s PCI host bridge does not support multiple data phase transactions as a master.
- Cache line size is not implemented, because the ÉlanSC520 microcontroller PCI host bridge does not support cacheable PCI memory.

9.5.2 Configuration Information

The PCI host bridge can generate configuration cycles on the PCI bus.

The Configuration Mechanism #1, as defined in the *PCI Local Bus Specification,* Revision 2.1, is used. The PCI Configuration Address (PCICFGADR) register resides at I/O address 0CF8h, and the PCI Configuration Data (PCICFGDATA) register resides at I/O address 0CFCh. The Am5$_x$86 CPU accesses these two I/O ports to generate PCI configuration cycles.

The PCI host bridge pre-drives the AD31–AD0 pins for five clocks before asserting $\overline{\text{FRAME}}$ when performing configuration cycles. This allows IDSEL to settle before the transaction starts (IDSEL signals may have a slow rise time).

External PCI bus devices require an IDSEL pin to allow configuration from the ÉlanSC520 microcontroller’s PCI bus host bridge. The method implemented for IDSEL generation is system-specific; however, the ÉlanSC520 microcontroller implements the commonly used practice in which the AD31–AD11 pins are asserted for IDSEL generation during the configuration cycles (the host bridge uses AD11). In this scheme, the AD12 is IDSEL for device number 1, AD13 is IDSEL for device number 2, etc. The AD pins are asserted during configuration cycles according to the decode of the PCI bus device; thus, this scheme is limited to 20 devices on the PCI bus.

The ÉlanSC520 microcontroller’s PCI bus host bridge is hardwired to device number 0 (AD11), and the host bridge PCI bus configuration registers are accessed through the PCI Configuration Address (PCICFGADR) register (Port 0CF8h) and PCI Configuration Data (PCICFGDATA) register (Port 0CFCh), like any external PCI device. An external PCI bus configuration cycle is not generated when the Am5$_x$86 CPU configures the internal PCI host bridge registers.

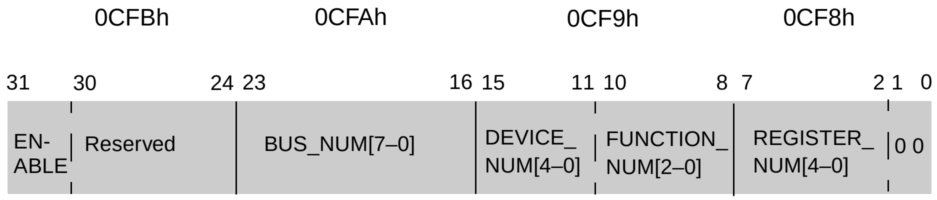
The host bridge PCI bus configuration space contains only PCI bus device configuration header registers, as defined in the PCI bus specification. ÉlanSC520 microcontroller-specific host bridge configuration registers are memory-mapped in ÉlanSC520 microcontroller configuration space. See Chapter 4, “System Address Mapping”, for further details on memory-mapped configuration space.

9.5.2.1 Generating PCI Bus Configuration Cycles

A two-step process is required to generate a PCI bus configuration cycle.

1. First, the Am5$_x$86 CPU must perform a 32-bit I/O write to the PCI Configuration Address (PCICFGADR) register (Port 0CF8h) with the following information: bus number, device number, function, and register number (doubleword) to be accessed (see Figure 9-7).
2. Then, the Am5$_x$86 CPU can perform an I/O cycle (read or write) to the PCI Configuration Data (PCICFGDATA) register (Port 0CFCh) to access the desired configuration register.

Figure 9-7 PCI Configuration Address (PCICFGADR) Register



Bit	Name	Function
31	ENABLE	This bit must be set to 1 to enable configuration space mapping.
30–24	Reserved	
23–16	BUS_NUM[7–0]	Bus number
15–11	DEVICE_NUM[4–0]	Device number
10–8	FUNCTION_NUM[2–0]	Function number
7–2	REGISTER_NUM[4–0]	Register number
1–0	Reserved	These bits must always be written to 00.

For example, to access the Status/Command (PCISTACMD) register (PCI index 04h) (doubleword 1) of the PCI host bridge, the following cycles are generated by the Am5$_x$86 CPU:

1. 32-bit I/O write to Port 0CF8h: 80000004h
   - ENABLE = 1 to enable configuration space mapping
   - BUS_NUM = 0 (PCI host bridge is on bus number 0)
   - DEVICE_NUM = 0 (PCI host bridge is hardwired to device number 0)
   - FUNCTION_NUM = 0 (PCI host bridge has only one function)
   - REGISTER_NUM = 1
   - Bits 1–0 must be written 00
2. 8/16/24/32-bit I/O read/write to/from Port 0CFCh to access configuration register bytes

The Master Enable (BUS_MAS) bit in the Status/Command (PCISTACMD) register (PCI index 04h) is always forced active. Thus, the PCI host bridge can always generate memory, I/O, and configuration transactions on the PCI bus to configure external PCI devices.

To enable the host bridge as a PCI bus target device, the Memory Access Enable (MEM_ENB) bit in the Status/Command (PCISTACMD) register must be set. When this bit is set, the host bridge responds to external PCI bus master cycles that access the ÉlanSC520 microcontroller's SDRAM.

No configuration bits need to be set to access the PCI host bridge's configuration registers from the Am5$_X$86 CPU.

Note that any write access to the PCI Configuration Data (PCICFGDATA) register (Port 0CFCh) in which the ENABLE bit of the PCI Configuration Address (PCICFGADR) register (Port 0CF8h) is not set is forwarded to the PCI bus as an I/O transaction.

Any non-doubleword access to Port 0CF8h is also forwarded to the PCI bus as an I/O transaction.

9.5.3 Élan™SC520 Microcontroller's Host Bridge as PCI Bus Master

The PCI host bridge allows the Am5$_X$86 CPU to be a master on the PCI bus. The Am5$_X$86 CPU can generate configuration transactions to configure the host bridge, as well as all external devices on the PCI bus (internal PCI host bridge configuration cycles are not seen on the external PCI bus). The Am5$_X$86 CPU can also generate memory and I/O read and write transactions on the PCI bus.

As a PCI bus master, the ÉlanSC520 microcontroller does not generate the following cycles:

- Dual address cycles for 64-bit addressing
- Memory-write-and-invalidate cycles (cacheable memory on the PCI bus is not supported)
- Memory-read-multiple or memory-read-line cycles (the Am5$_X$86 CPU does not generate long read burst transactions that may benefit from these commands)
- Fast back-to-back cycles
- Lock cycles (the $\overline{\text{LOCK}}$ pin is not supported)
- Multiple data phase cycles
- Special cycles and interrupt acknowledge cycles (these Am5$_X$86 CPU cycles are not echoed on the PCI bus)

9.5.3.1 Write Posting

To increase Am5$_X$86 CPU bandwidth utilization, memory writes to the PCI bus can be posted by setting the M_WPOST_ENB bit in the Host Bridge Control (HBCTL) register (MMCR offset 60h). This allows the Am5$_X$86 CPU cycle to complete without incurring the PCI bus transaction latency. The $\overline{\text{rdy}}$ signal is returned immediately to the Am5$_X$86 CPU, and the cycle completes sometime later on the PCI bus. The PCI host bridge posts only one Am5$_X$86 CPU write cycle to the PCI bus. Am5$_X$86 CPU-to-PCI bus-cycle ordering is maintained, which means additional Am5$_X$86 CPU cycles (both read and write) to the PCI bus incur wait states until a posted write cycle completes on the PCI bus.

I/O and configuration write cycles are not posted. However, write cycles to memory-mapped I/O regions are not detected by the PCI host bridge, so write posting must be disabled to prevent the posting of memory-mapped I/O cycles. If write posting is disabled, the PCI host bridge waits until the write cycle has completed on the PCI bus before returning ready to the Am5$_X$86 CPU.

Write posting should not be enabled while operating in nonconcurrent arbitration mode. See Chapter 8, “System Arbitration”, for further details on nonconcurrent mode arbitration.

9.5.3.2 Read Cycles

The PCI host bridge does not read ahead PCI bus memory for Am5$_X$86 CPU read cycles. Each Am5$_X$86 CPU read cycle generates a single data phase read cycle on the PCI bus, with only the data requested by the Am5$_X$86 CPU being read. The PCI host bridge does not burst Am5$_X$86 CPU-to-PCI-bus read cycles, because the Am5$_X$86 CPU typically performs burst reads only during cache-line fills, and PCI bus memory is noncacheable. There are a few cases when the Am5$_X$86 CPU may burst two doublewords (i.e., misaligned transfer). In this case, the PCI host bridge breaks the transfer up into single cycles on the PCI bus.

9.5.3.3 Delayed Transaction Support

The PCI host bridge as a PCI master supports delayed transactions. A transaction that was retried repeats until completed on the PCI bus. The PCI host bridge does not make any distinction between a transaction that was retried and a transaction that was disconnected. Both types of transactions are repeated until they complete on the PCI bus.

A programmable retry time-out counter prevents a deadlock condition due to a broken target on the PCI bus. The Master Retry Time-Out (M_RETRY_TO) field in the Master Retry Time-Out (PCIMRETRYTO) register (PCI index 41h) controls this feature. When the time-out counter expires (a cycle was retried unsuccessfully n times on the PCI bus), the cycle is discarded and an interrupt can be generated. For a read cycle, the data returned is all ones. The Host Bridge Master Interrupt Address (MSTINTADD) register (MMCR offset 6Ch) contains the address of the transaction that was retried unsuccessfully. Note that the master retry count configuration must not be changed except during PCI bus initialization after a system or programmable reset.

Transaction ordering is maintained during delayed transactions. A transaction that is retried by an external PCI bus target must complete before any subsequent Am5$_X$86 CPU-to-PCI bus transactions are generated.

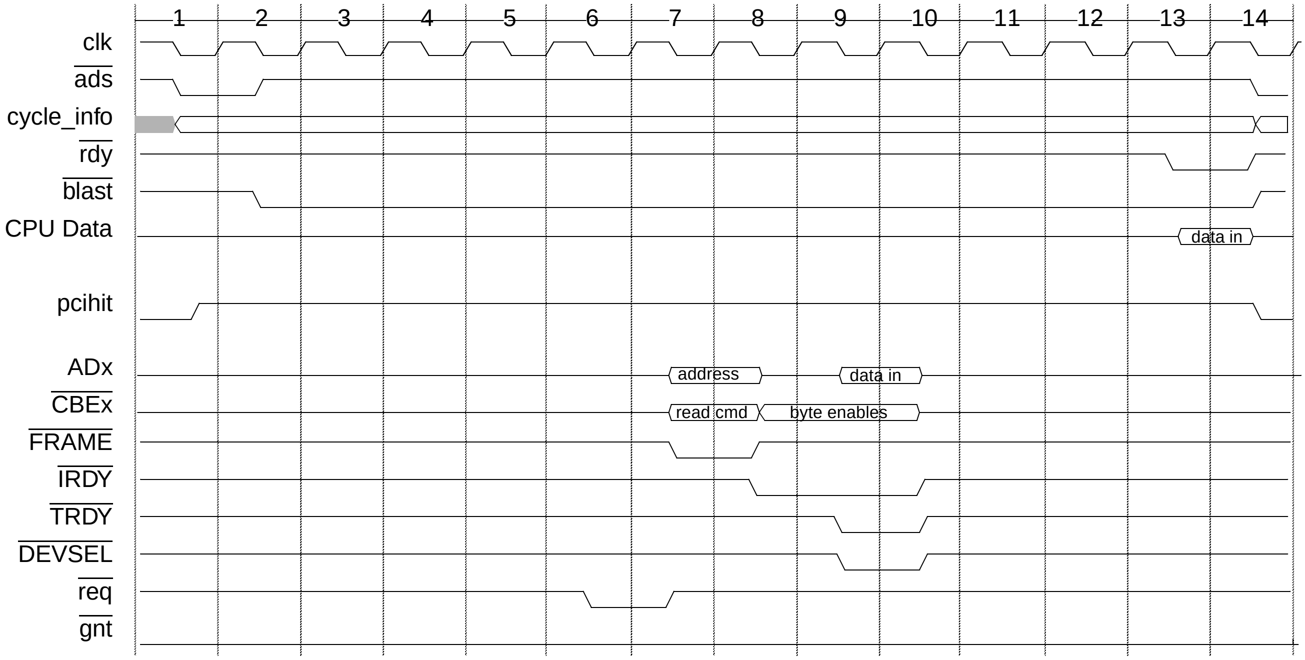
9.5.3.4 Host Bridge Master Bus Cycles

This section describes in detail the cycles generated by the ÉlanSC520 microcontroller acting as PCI host bridge master and includes both the PCI bus and the internal Am5$_X$86 CPU bus. Note that these are example cases only, and not all cases are shown. The diagrams are functionally representative in nature, and should not be used to infer detailed timing information. Note also that the synchronization between the CPU and PCI clock domains is not shown in detail.

9.5.3.4.1 CPU Read Cycle to the PCI Bus

Figure 9-8 shows an Am5$_X$86 CPU read cycle to the PCI bus. Figure 9-8 could also represent a memory, I/O or external PCI bus device configuration cycles. The first group of signals includes the internal Am5$_X$86 CPU signals, the second group includes additional ÉlanSC520 microcontroller internal signals, and the third group includes the PCI bus signals. Note that the PCI bus request and grant signals are shown for convenience, but these are not seen externally when the Am5$_X$86 CPU is the initiator of PCI bus transactions.

Figure 9-8 CPU Read Cycle to the PCI Bus



Notes:

The diagram includes the following internal signals:

- *pcihit: Address decode signal that the current Am5$_{X}$86 CPU cycle is a PCI cycle.*

The clk signal denotes the 33-MHz clock source and represents both the CPU clock and the PCI clock. This diagram does not represent the full synchronization of signals between these clock domains.

The following sequence annotates the Am5$_{X}$86 CPU read cycle to the PCI bus shown in Figure 9-8.

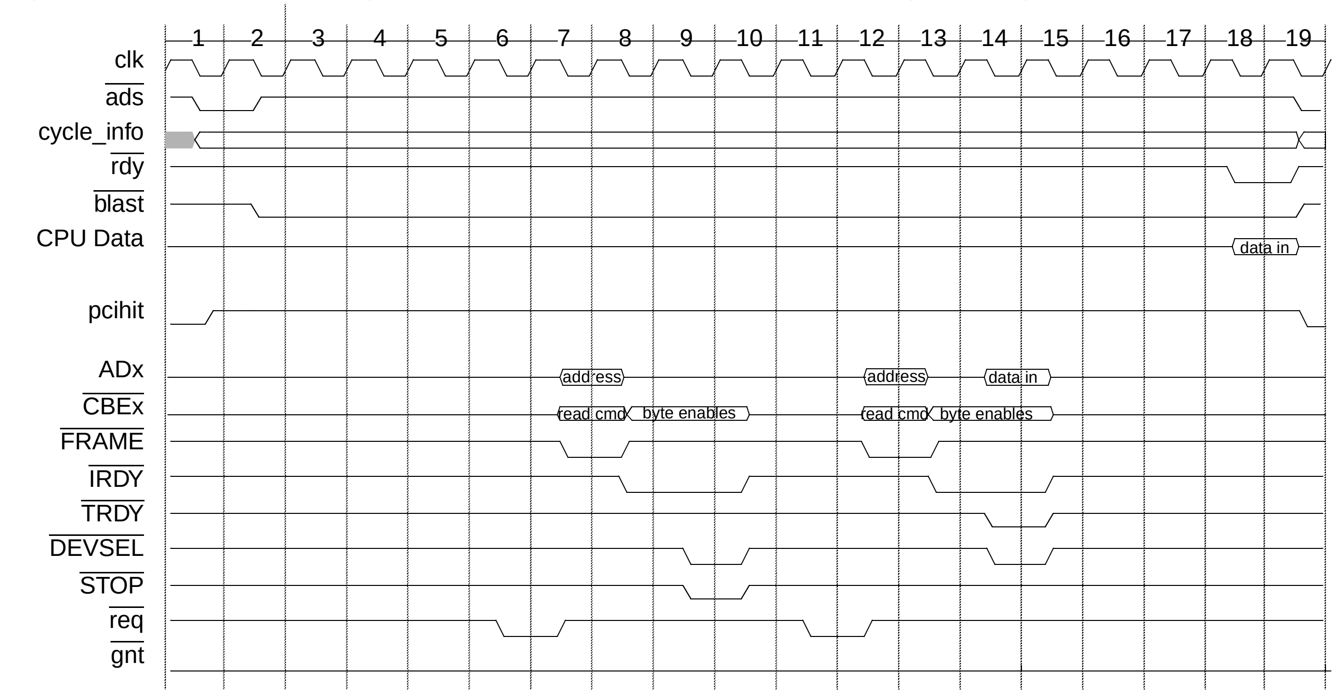
- **Clock #1:** The Am5$_{X}$86 CPU starts a read cycle to the PCI bus.
- **Clock #2:** Note that blast is asserted by the Am5$_{X}$86 CPU signaling a non-burst transfer. If this were a burst read cycle, the Am5$_{X}$86 CPU would deassert blast, but because the PCI host bridge returns rdy to the Am5$_{X}$86 CPU instead of brdy, the Am5$_{X}$86 CPU would break up the burst into single cycles. A posted write cycle pending in the master posted write buffer would delay the completion of the Am5$_{X}$86 CPU read cycle.
- **Clock #6:** The PCI host bridge master controller has synchronized the Am5$_{X}$86 CPU bus request and asserts req to gain access to the PCI bus. Because the Am5$_{X}$86 CPU is the initiator of the cycle, the bus request signal is not seen externally.
- **Clock #7:** The PCI host bridge gnt signal is sampled asserted, and the PCI bus is idle, so FRAME is asserted to begin the PCI bus transaction. In this example, there is no arbitration delay (the arbiter is parked on the host bridge). If another external PCI bus master was granted the bus, or the bus was not idle, FRAME assertion would be delayed until the host bridge's gnt was asserted and the bus was idle.
- **Clock #9:** The external PCI bus target asserts TRDY, indicating that the requested data is available. In this example, the PCI bus target did not add any wait states to the transaction. A PCI bus Revision 2.2-compliant target can add up to 16 wait states that would delay the PCI bus transaction and subsequent Am5$_{X}$86 CPU cycle completion. An external PCI bus target can also issue a retry that would delay the PCI bus transaction and subsequent Am5$_{X}$86 CPU cycle completion (see Section 9.5.3.4.2).

- **Clock #10:** The PCI host bridge samples $\overline{\text{TRDY}}$ asserted and latches the data from the PCI bus.
- **Clock #13:** The Am5$_\text{X}$86 CPU bus synchronizes the end of the PCI bus cycle and asserts $\overline{\text{rdy}}$ to the Am5$_\text{X}$86 CPU with the requested read data.

9.5.3.4.2 CPU Read Cycle to the PCI Bus with External Target Retry

Figure 9-9 shows an Am5$_\text{X}$86 CPU read cycle to the PCI bus that was retried by the external PCI bus target. An external PCI bus target can issue a retry if it is currently busy or if the transaction will be completed as a delayed transaction.

Figure 9-9 CPU Read Cycle to the PCI Bus with External Target Retry



Notes:

The clk signal denotes the 33-MHz clock source and represents both the CPU clock and the PCI clock. This diagram does not represent the full synchronization of signals between these clock domains.

The following sequence annotates the Am5$_\text{X}$86 CPU read cycle to the PCI bus with external target retry shown in Figure 9-9. This example is the same as a regular read (see Section 9.5.3.4.1) until Clock #9.

- **Clock #9:** The target asserts $\overline{\text{STOP}}$ with $\overline{\text{TRDY}}$ deasserted, signaling a retry. The target may add up to 16 waitstates before asserting $\overline{\text{STOP}}$, which would delay the PCI transaction and Am5$_\text{X}$86 CPU cycle completion.
- **Clock #10:** The PCI host bridge master controller deasserts $\overline{\text{IRDY}}$ and ends the current transaction. The data requested by the Am5$_\text{X}$86 CPU was not read because of the delayed transaction, so $\overline{\text{rdy}}$ is not returned to the Am5$_\text{X}$86 CPU. The host bridge will retry the current transaction until data is read from the target.
- **Clock #11:** The PCI host bridge asserts $\overline{\text{req}}$ to re-gain access to the PCI bus. Because the Am5$_\text{X}$86 CPU is the initiator of the cycle, the bus request signal is not seen externally.
- **Clock #12:** The PCI host bridge $\overline{\text{gnt}}$ signal is sampled asserted, and the PCI bus is idle, so $\overline{\text{FRAME}}$ is asserted to retry the PCI transaction. In this example, there is no arbitration

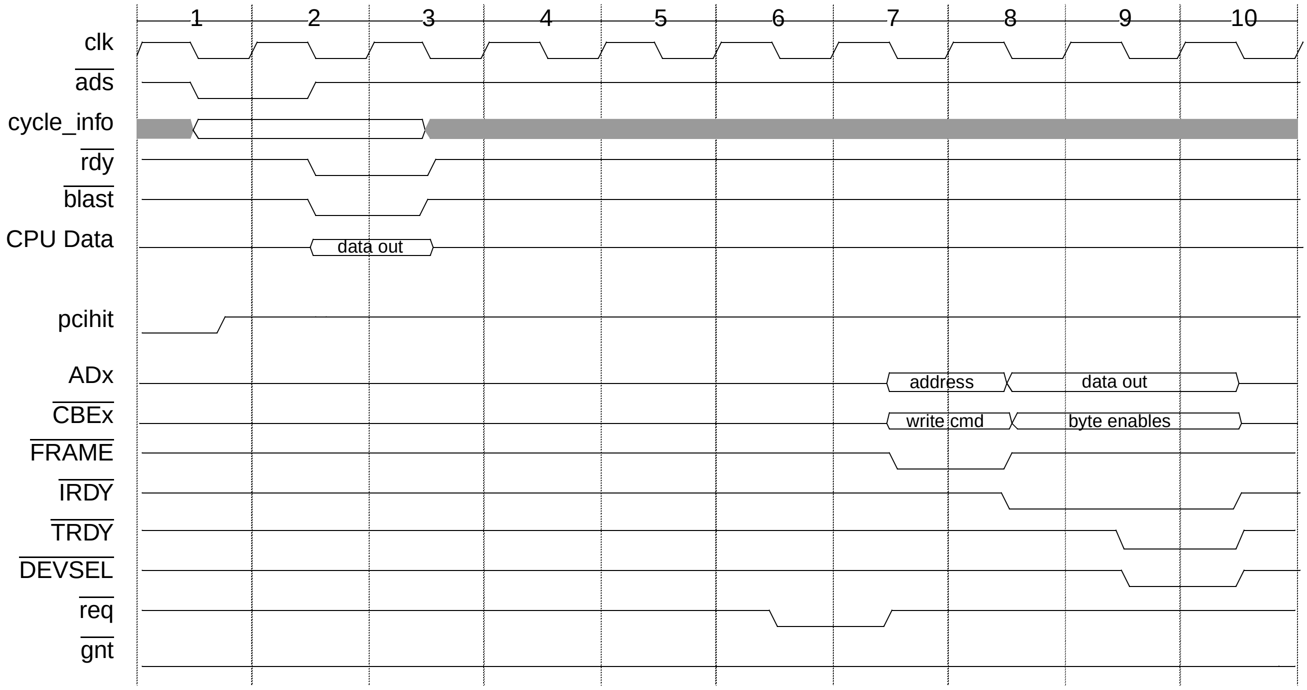
delay (the arbiter is parked on the host bridge). If another external PCI bus master was granted the bus or the bus was not idle, $\overline{\text{FRAME}}$ assertion would be delayed until the host bridge’s $\overline{\text{gnt}}$ was asserted and the bus was idle.

- **Clock #14:** The PCI bus target asserts $\overline{\text{TRDY}}$ indicating the data is available.
- **Clock #15:** The PCI host bridge samples $\overline{\text{TRDY}}$ asserted and latches the data from the PCI bus.
- **Clock #18:** The Am5$_{x}$86 CPU bus synchronizes the end of the PCI bus cycle and asserts $\overline{\text{rdy}}$ to the Am5$_{x}$86 CPU with the requested read data.

9.5.3.4.3 CPU Posted Write Cycle to the PCI Bus

Figure 9-10 shows an Am5$_{x}$86 CPU write cycle to the PCI bus that is posted by the PCI host bridge. This can only be a memory-write cycle to the PCI bus; I/O and configuration writes are not posted.

Figure 9-10 CPU Posted Write Cycle to the PCI Bus



Notes:

The clk signal denotes the 33-MHz clock source and represents both the CPU clock and the PCI clock. This diagram does not represent the full synchronization of signals between these clock domains.

The following sequence annotates the Am5$_{x}$86 CPU posted write cycle to the PCI bus shown in Figure 9-10.

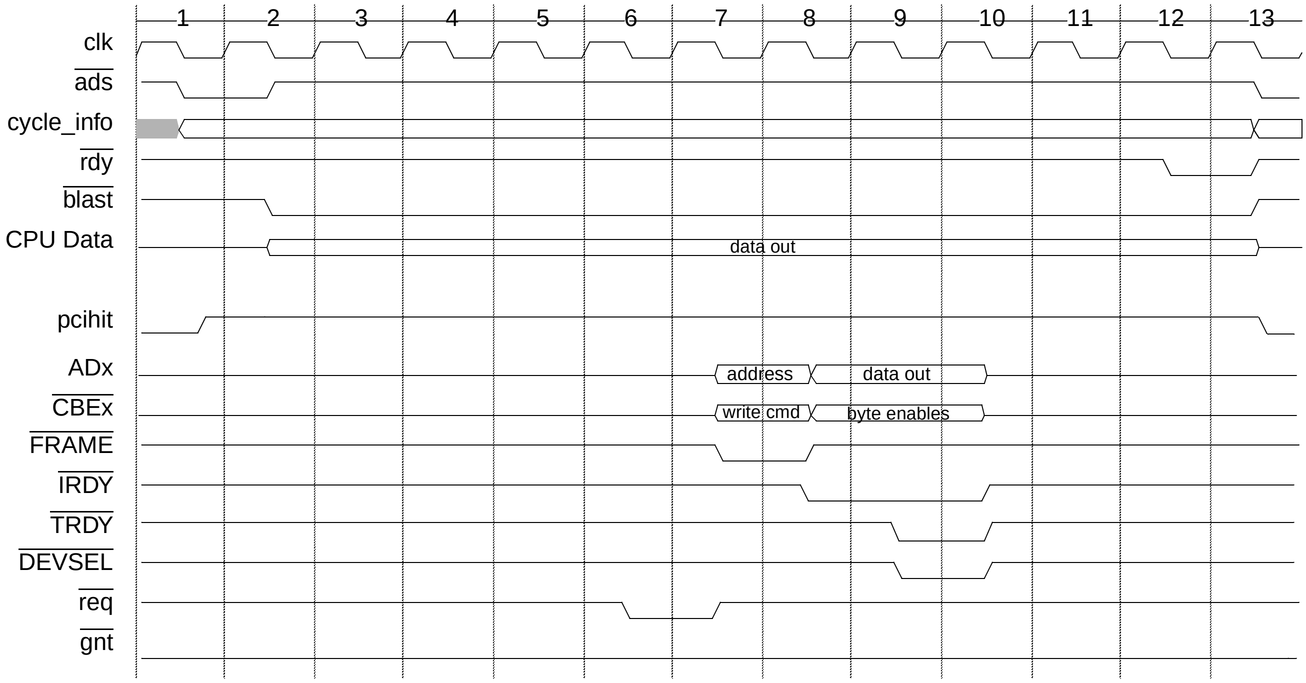
- **Clock #1:** The Am5$_{x}$86 CPU starts a write cycle to the PCI bus.
- **Clock #2:** The PCI host bridge also asserts $\overline{\text{rdy}}$ to the Am5$_{x}$86 CPU, which ends the Am5$_{x}$86 CPU write cycle. The PCI bus transaction has been posted in the host bridge and will complete sometime later. If another write cycle is already pending in the posted write buffer, $\overline{\text{rdy}}$ will be delayed to the Am5$_{x}$86 CPU until the preceding posted write has completed.

- **Clock #6:** The PCI host bridge master controller has synchronized the Am5_x86 CPU bus request and asserts $\overline{\text{req}}$ to gain access to the PCI bus.
- **Clock #7:** The PCI host bridge $\overline{\text{gnt}}$ signal is sampled asserted, and the PCI bus is idle, so $\overline{\text{FRAME}}$ is asserted to begin the PCI transaction. In this example, there is no arbitration delay (the arbiter is parked on the host bridge). If another external PCI master was granted the bus or the bus was not idle, $\overline{\text{FRAME}}$ assertion would be delayed until the host bridge's $\overline{\text{gnt}}$ was asserted and the bus was idle. Because the Am5_x86 CPU is the initiator of the cycle, the bus request signal is not seen externally.
- **Clock #9:** The external PCI target asserts $\overline{\text{TRDY}}$ indicating it can accept the write data. In this example, the PCI target did not add any wait states to the transaction. A PCI bus Revision 2.2 compliant target can add up to 16 wait states that would delay the transaction completion. A PCI bus target can also retry the PCI transaction. In this case, the host bridge continues to generate the same transaction until the target returns $\overline{\text{TRDY}}$ to complete the transaction. See Section 9.5.3.4.2 for information on retried transactions.
- **Clock #10:** The PCI host bridge samples $\overline{\text{TRDY}}$ asserted, which ends the PCI bus transaction.

9.5.3.4.4 CPU Non-Posted Write Cycle to the PCI Bus

Figure 9-11 shows an Am5_x86 CPU memory write cycle to the PCI bus with write posting disabled. Figure 9-11 could represent any I/O or configuration write cycle.

Figure 9-11 Am5_x86 CPU Non-Posted Write Cycle to the PCI Bus



Notes:

The clk signal denotes the 33-MHz clock source and represents both the CPU clock and the PCI clock. This diagram does not represent the full synchronization of signals between these clock domains.

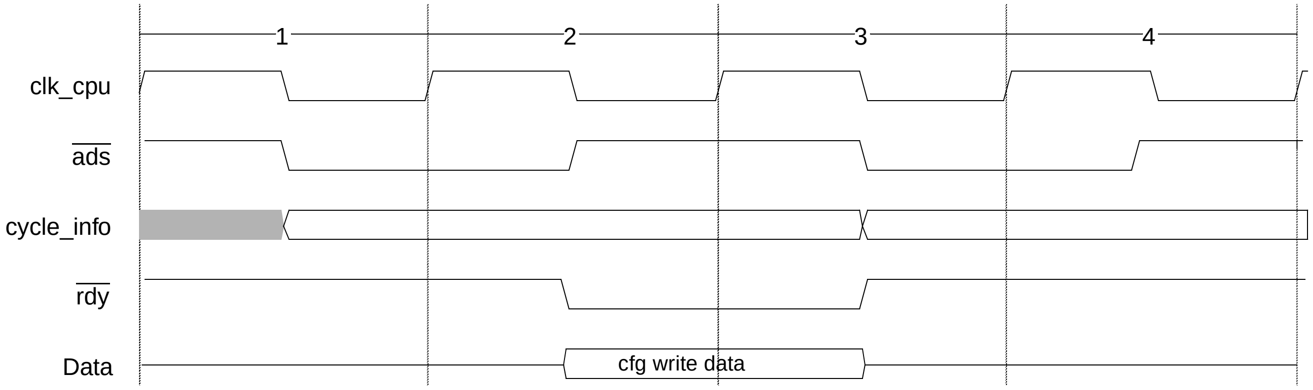
The following sequence annotates the Am5$_x$86 CPU non-posted write cycle to the PCI bus shown in Figure 9-11.

- **Clock #1:** The Am5$_x$86 CPU starts a write cycle to the PCI bus.
- **Clock #6:** The PCI host bridge master controller has synchronized the Am5$_x$86 CPU bus request and asserts $\overline{\text{req}}$ to gain access to the PCI bus. Because the Am5$_x$86 CPU is the initiator of the cycle, the bus request signal is not seen externally.
- **Clock #7:** The PCI host bridge $\overline{\text{gnt}}$ signal is sampled asserted, and the PCI bus is idle, so $\overline{\text{FRAME}}$ is asserted to begin the PCI bus transaction. In this example, there is no arbitration delay (the arbiter is parked on the host bridge). If another external PCI bus master was granted the bus, or the bus was not idle $\overline{\text{FRAME}}$ assertion would be delayed until the host bridge's $\overline{\text{gnt}}$ was asserted and the bus was idle. Because the Am5$_x$86 CPU is the initiator of the cycle, the bus request signal is not seen externally.
- **Clock #9:** The PCI target asserts $\overline{\text{TRDY}}$, indicating it can accept the write data. In this example, the PCI bus target did not add any wait states to the transaction. A PCI bus Revision 2.2 compliant target can add up to 16 wait states that would delay the transaction completion. A PCI bus target can also retry the transaction. In this case, the host bridge continues to generate the same transaction until the target returns $\overline{\text{TRDY}}$ to complete the transaction. The $\overline{\text{rdy}}$ signal is not returned to the Am5$_x$86 CPU until the PCI bus transaction completes. See Section 9.5.3.4.2 for information on retried transactions.
- **Clock #10:** The PCI host bridge samples $\overline{\text{TRDY}}$ asserted, which ends the transaction.
- **Clock #12:** The Am5$_x$86 CPU bus synchronizes the end of the PCI bus cycle and asserts $\overline{\text{rdy}}$ to the Am5$_x$86 CPU, which ends the write cycle.

9.5.3.4.5 PCI Bus Configuration Read/Write

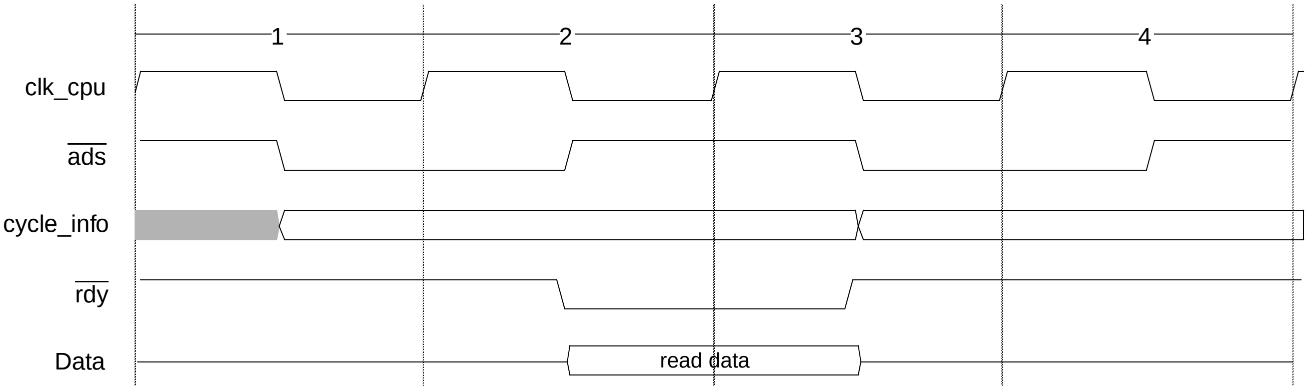
Am5$_x$86 CPU write cycles to the PCI Configuration Address (PCICFGADR) register (Port 0CF8h) or the PCI Configuration Data (PCICFGDATA) register (Port 0CFCh) for internal PCI host bridge configuration complete with zero Am5$_x$86 CPU cycle wait states (see Figure 9-12).

Figure 9-12 CPU Write Cycles to Internal PCI Bus Configuration Registers



Am5$_x$86 CPU read cycles from the PCI Configuration Address (PCICFGADR) register or PCI Configuration Data (PCICFGDATA) register for internal PCI host bridge configuration registers also complete with zero wait states (see Figure 9-13). See the read and write timing diagrams in Figure 9-8 through Figure 9-11 for Am5$_x$86 CPU read and writes cycles to the PCI Configuration Data (PCICFGDATA) register that access external PCI bus device configuration registers.

Figure 9-13 CPU Read Cycles from Internal PCI Bus Configuration Registers



9.5.4 Élan™SC520 Microcontroller's Host Bridge as PCI Bus Target

As a target, the integrated PCI host bridge only accepts memory cycles from external PCI bus masters to allow accesses to the ÉlanSC520 microcontroller's SDRAM.

To enable the host bridge as a PCI bus target device, the Memory Access Enable (MEM_ENB) bit in the Status/Command (PCISTACMD) register must be set. When this bit is set, the PCI host bridge ignores all I/O and configuration cycles on the PCI bus and responds to memory cycles within the address space, as defined in Section 9.5.4.1.

9.5.4.1 PCI Host Bridge Target Address Space

Under normal conditions, the ÉlanSC520 microcontroller's PCI host bridge responds to PCI bus master memory cycles in the entire SDRAM address space to allow full access of SDRAM from external PCI bus masters. This space is defined as a linear region, starting at the lowest address (00000000h) and ending at the top of SDRAM, depending on the amount populated in the system (a maximum of 256 Mbytes). The SDRAM controller's configuration registers are programmed with the amount of SDRAM in the system during the initial boot process.

Some systems may require specific CPU address space that is normally defined as an SDRAM region to be redirected to the PCI bus. An example application is a PCI-bus-based VGA video card for PC/AT compatibility. In ÉlanSC520 microcontroller, this redirection is programmed via the first two Programmable Address Region (PAR) registers (PAR 0 and PAR 1). When this feature is used in a system, the ÉlanSC520 microcontroller's PCI host bridge target shadows PAR 0 and PAR 1 and ignores accesses by external PCI bus masters in the programmed address space if they are programmed for PCI bus in the TARGET field.

See Chapter 4, "System Address Mapping", for further details of PCI host bridge target address space.

Because the ÉlanSC520 microcontroller is configured as a PCI host bridge, the PCI bus Base Address registers normally found in the PCI bus configuration space are not implemented.

9.5.4.2 PCI Bus Command Support

As a PCI bus target, the ÉlanSC520 microcontroller's PCI host bridge treats the memory-write-and-invalidate command the same as a memory-write cycle. When either of these commands is issued by a PCI bus master, the PCI host bridge and system arbitration blocks force the Am5$_X$86 CPU's integrated cache to snoop the addresses prior to writing the data to SDRAM. If the cache detects a modified cache line at the same address, it writes back and invalidates the line. If the CPU is operating in write-through cache mode, the line is simply invalidated and the data is written to SDRAM.

The PCI host bridge does not respond to configuration cycles or special cycles issued by external PCI bus masters. Interrupt acknowledge cycles and special cycles are not forwarded to the PCI bus.

9.5.4.3 $\overline{\text{DEVSEL}}$ Timing

When an external PCI bus master accesses the ÉlanSC520 microcontroller's SDRAM, the PCI host bridge always asserts $\overline{\text{DEVSEL}}$ with medium timing (two clocks after $\overline{\text{FRAME}}$ is asserted). The ÉlanSC520 microcontroller does not serve as a subtractive decode agent on the PCI bus.

9.5.4.4 Delayed Transaction Support

External PCI bus master reads of the ÉlanSC520 microcontroller's SDRAM can be configured to be delayed transactions This maximizes PCI bus efficiency by freeing up the bus while the initial SDRAM read request is issued to the SDRAM controller.

When the Automatic Delayed Transaction Enable (T_DLYTR_ENB) field is set in the Host Bridge Control (HBCTL) register (MMCR offset 60h), the PCI host bridge immediately issues a retry to the external PCI bus master read cycle and begins requesting the data from the SDRAM controller. The external PCI bus master read cycle is retried until *any* of the requested data has been read into the target read FIFO. Only the first doubleword requested needs to be read into the target read FIFO before the PCI host bridge completes the delayed transaction instead of retrying it again. After the PCI host bridge responds to the delayed transaction, it continues to prefetch data and provides all the data requested (up to 64 doublewords maximum) by the external PCI bus master without disconnecting.

When a delayed transaction read cycle is pending (waiting for the originating external PCI bus master to retry the transaction), all other read transactions are terminated with a retry. The PCI host bridge supports one outstanding delayed transaction, so these retried transactions are not latched. Write transactions, however, are allowed to complete and are placed in the PCI host bridge target write FIFO. A delayed transaction discard timer is provided so that a broken master does not deadlock the system. If, after 2^{15} PCI clocks, a master has not retried a delayed transaction, the transaction is discarded and an interrupt can be optionally generated. The delayed transaction discard timer is fixed at 2^{15} PCI clocks.

When external PCI bus master reads of ÉlanSC520 microcontroller's SDRAM are not configured as automatic delayed transactions, the PCI host bridge tries to return the requested data to the PCI bus master without issuing a retry. Wait states are inserted into the transaction until the data is read from SDRAM. If the initial data cannot be returned in 32 clocks, the PCI host bridge terminates the transaction with a retry and latches the read transaction as a delayed transaction to comply with the *PCI Local Bus Specification,* Revision 2.2. Note that if any data is pending in the Am5$_X$86 CPU-to-PCI posted-write latch, it must be flushed before read data can be returned to an external PCI master by the PCI host bridge target controller. In this case, the PCI host bridge immediately retries the external PCI master read transaction and latches the request as a delayed transaction.

The PCI host bridge retries any external PCI bus master write cycle when the write FIFO is full. The PCI host bridge retries all external PCI bus master cycles (write and read) if the address FIFO is full (see the Section 9.5.4.5).The PCI host bridge always disconnects after 64 consecutive doublewords are transferred to prevent any one PCI bus master from monopolizing the bus and to guarantee sufficient CPU bus bandwidth.

9.5.4.5 Address FIFO

The PCI host bridge's target controller includes an address FIFO that keeps track of address and command requests made to the target controller. The address FIFO allows one outstanding delayed read transaction and up to four posted writes, depending on the ordering of the transactions.

- If the address FIFO is empty (no latched transactions in the target controller) and a read transaction is received prior to any posted writes, the read is latched and a delayed transaction retry is issued. After this, up to four posted writes can be latched following the read (for a total of five latched transactions in the FIFO).
- If the address FIFO contains any posted write transaction (before a read transaction is received), only a total of four transactions can be latched into the address FIFO. That is, if the first posted transaction is a write, up to four transactions can be latched into the address FIFO (three writes and one read, or four writes).
- If four posted writes reside in the address FIFO, no delayed read transactions can be latched. In this case, all read requests are retried (not latched into the address FIFO) until one of the posted writes has completed internally.
- In all cases, only a maximum of one delayed read transaction can be latched into the address FIFO. If two read transactions are received, the target controller only latches the first one. The second (and subsequent) reads are not latched into the target controller, even if the address FIFO is not full.
- Note that, even if the address FIFO is not full, but the data FIFO is already full, further posted writes are not accepted.

The ÉlanSC520 microcontroller's PCI host bridge complies to the *PCI Local Bus Specification,* Revision 2.2, rules for transaction ordering to prevent deadlock conditions.

9.5.4.6 PCI Host Bridge FIFOs and Prefetching

The PCI host bridge target controller has a 64-doubleword write FIFO and posts writes from external PCI bus masters to SDRAM. The PCI host bridge does not insert wait states into an external PCI bus master write cycle by deasserting $\overline{\text{TRDY}}$. If the write FIFO becomes full during an external PCI bus master write transaction, the PCI host bridge issues a disconnect to end the cycle. A maximum of four transfers can be posted (each transfer can burst multiple data phases, but the ÉlanSC520 microcontroller's target FIFOs store a maximum number of 64 doublewords for all the posted writes).

The SDRAM controller's write buffer can byte-merge, combine, and collapse data if enabled, yielding additional performance of SDRAM writes from PCI bus masters. See Chapter 11, "Write Buffer and Read Buffer", for further details. However, the PCI host bridge does *not* byte-merge, combine, or collapse data in the target write FIFO.

The PCI host bridge as a target prefetches data from SDRAM in response to an external PCI bus master read transaction. The read buffer in the SDRAM controller should be enabled for optimal performance, especially during memory-read-multiple commands by external PCI bus masters.

- For memory-read and memory-read-line commands, the PCI host bridge prefetches data up to the next cache line (a cache line is four doublewords).
- Memory-read-multiple commands fill the target FIFO (64 doublewords).

Once the PCI host bridge has been granted access to the CPU bus, it will hold the bus until it has prefetched up to the next cache-line boundary for memory-read and memory-read-line commands, and 64 doublewords for memory-read-multiple commands. The PCI host bridge may insert wait states before asserting $\overline{\text{TRDY}}$ for the first data phase. The PCI host bridge can then burst one cache line with zero wait states. After each cache line, the PCI host bridge can insert wait states by deasserting $\overline{\text{TRDY}}$ if the target read FIFO becomes empty.

Note that, if the target read FIFO becomes empty after a cache-line boundary for memory-read and memory-read-line commands or after 64 doublewords for a memory-read-multiple command, the PCI host bridge issues a disconnect to end the transaction.

9.5.4.7 Burst Ordering

To provide optimal CPU performance during SDRAM accesses, the ÉlanSC520 microcontroller’s SDRAM controller is designed to support Am5$_X$86 CPU cache-line burst ordering, but the PCI bus specifies linear burst ordering. Therefore, all PCI host bridge accesses to SDRAM are cache-line-aligned (start on a four-doubleword boundary). If the external PCI bus master read cycle was not cache-line-aligned, the PCI host bridge starts requesting the SDRAM read from the address that the master issued and generates single-phase data cycles until it becomes cache-line-aligned.

For example, if the external PCI bus master started a write with address 10008h and wrote ten doublewords, the PCI host bridge would generate single, non-burst write cycles to address 10008h and 1000Ch. After these two write cycles, the transaction would be cache-line-aligned, so the PCI host bridge would complete the transaction with burst cycles.

9.5.4.8 Maintaining Data Coherency

All external PCI bus master accesses to SDRAM are snooped by the Am5$_X$86 CPU’s cache, which writes back and invalidates a cache line as appropriate. If the CPU detects a hit to a modified line in its cache, the arbitration unit forces the PCI host bridge to relinquish the Am5$_X$86 CPU bus to allow the cache line to be written back to SDRAM. If the cache is configured in write-through cache mode, the line is simply invalidated and the PCI host bridge is not forced off the bus for a write-back cycle.

In many systems that employ posting buffers, a potential data coherency problem exists because of the delay between an external master write transaction and when SDRAM is actually updated due to the write posting FIFO. The PCI bus complicates this potential problem when PCI-to-PCI bridges are implemented in the system.

In ÉlanSC520 microcontroller, for example, if an external master writes a block of data into SDRAM and then generates an interrupt request to the Am5$_X$86 CPU to process the data, it is important to prevent the Am5$_X$86 CPU from attempting to read SDRAM before the posted data has actually been written to SDRAM by the PCI host bridge’s posting-write FIFO. The PCI bus specification recommends that the CPU perform a read to the interrupting PCI bus device, to force all system posted write buffers to flush (including PCI bus bridges).

If the PCI host bridge target read FIFOs contain data from a previous memory-read command that was obtained as part of a delayed transaction while a write to the same memory address region occurs, the read FIFOs can optionally be purged to maintain coherency by setting the T_PURGE_RD_ENB bit in the Host Bridge Control (HBCTL) register (MMCR offset 60h). The T_PURGE_RD_ENB bit must not be changed except during PCI bus initialization after a system or programmable reset.

- Memory-read and memory-read-line commands generate a purge when the write address is within the same cache line as the prefetched data. Note that the addresses do not necessarily overlap in this case. For example, a memory-read command to 5008h will prefetch 5008h and 500Ch. A memory-write command to 5000h will then cause a purge because it is in the same cache line, even though the addresses do not overlap.
- Memory-read-multiple commands generate a purge if the write is in the same 64-doubleword region as the prefetched data. In this case, exact addresses are compared. Note that a write to the same 64-doubleword region causes a purge even if the prefetch is not complete. If, for example, the host bridge is prefetching the 32nd doubleword on the Am5$_x$86 CPU bus, and a write comes into the 53rd doubleword (or any number greater than 32 and less than 64, in this case), this write will cause a purge.

9.5.4.9 PCI Host Bridge Target Bus Cycles

This section describes in detail the cycles generated by an external PCI bus master for which the ÉlanSC520 microcontroller PCI host bridge responds, and includes both the PCI bus and the internal Am5$_x$86 CPU bus. The PCI host bridge forwards cycles that are destined to SDRAM from the PCI bus to the Am5$_x$86 CPU bus.

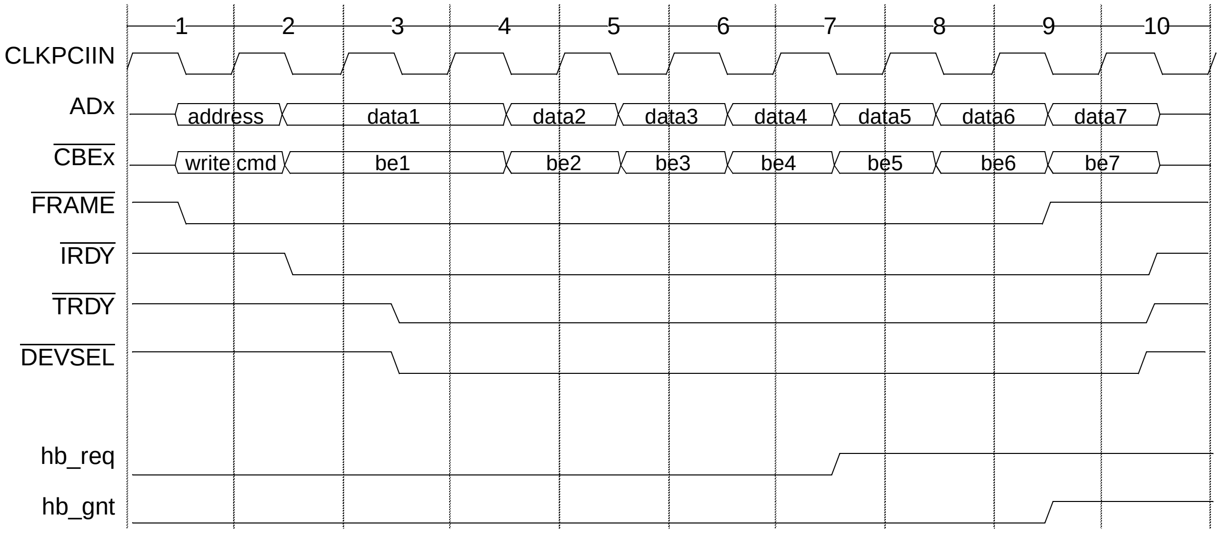
The examples shown apply primarily to concurrent arbitration mode; there are a few differences when operating in nonconcurrent arbitration mode. See Chapter 8, “System Arbitration”, for further details on the arbitration modes.

Note that these are example cases only, and not all cases are shown. The diagrams are functionally representative in nature, and should not be used to infer detailed timing information. Note also that the synchronization between the CPU and PCI clock domains is not shown in detail.

9.5.4.9.1 External PCI Bus Master Posted Write to SDRAM

Figure 9-14 shows an external PCI bus master writing seven doublewords to the ÉlanSC520 microcontroller’s SDRAM. The first group of signals are the PCI bus signals, and the second group are internal signals.

Figure 9-14 External PCI Bus Master Posted Write to SDRAM



Notes:

The diagram includes the following internal signals:

- *hb_req: PCI host bridge requesting the Am5$_X$86 CPU bus to access the SDRAM controller.*
- *hb_gnt: PCI host bridge has been granted Am5$_X$86 CPU bus and can access the SDRAM controller.*

See Chapter 8, “System Arbitration”, for information on Am5$_X$86 CPU bus arbitration.

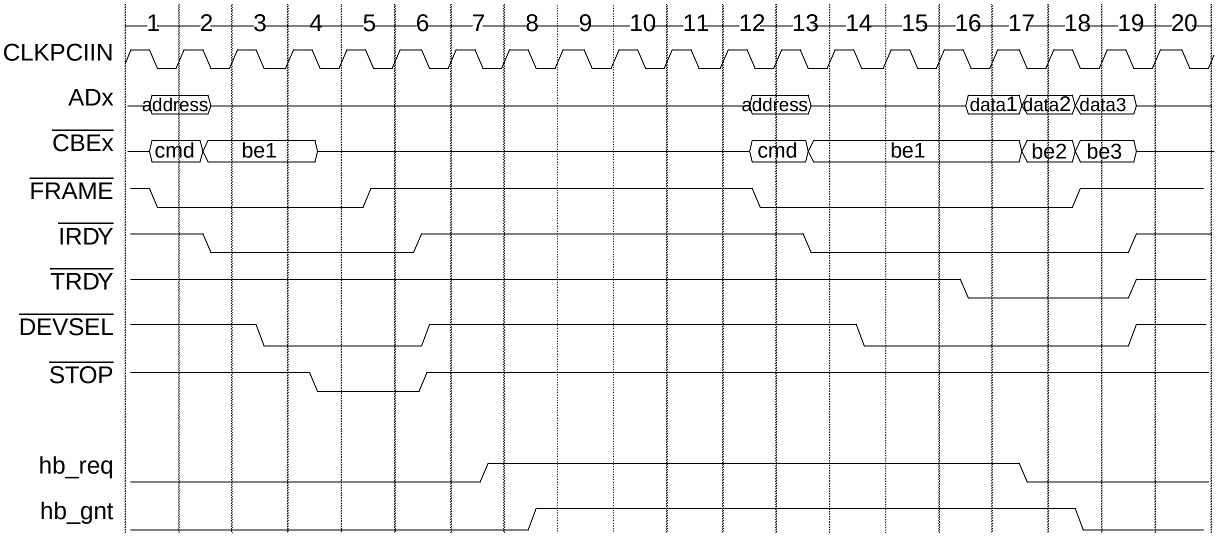
The following sequence annotates the external PCI bus master posted write to SDRAM shown in Figure 9-14.

- **Clock #1:** An external PCI master initiates a write transaction to the ÉlanSC520 microcontroller’s SDRAM.
- **Clock #3:** The PCI host bridge always asserts $\overline{\text{DEVSEL}}$ with medium timing. In this example, the write FIFO is not full, so $\overline{\text{TRDY}}$ is also asserted to accept the write data. If either the write FIFO or the address FIFO had been full, then the PCI host bridge would immediately issue a retry to the external master by asserting $\overline{\text{STOP}}$ instead of $\overline{\text{TRDY}}$.
- **Clocks #4–#10:** The write FIFO is not full, so $\overline{\text{TRDY}}$ remains asserted to accept the write data. The PCI host bridge does not insert wait states into the PCI transaction by deasserting $\overline{\text{TRDY}}$. If the FIFO becomes full during the transaction but the external PCI master indicates it is willing to burst more data (by keeping $\overline{\text{FRAME}}$ asserted), the host bridge issues a disconnect by deasserting $\overline{\text{TRDY}}$ and asserting $\overline{\text{STOP}}$ (see Section 9.5.4.9.3). The external master can insert wait states into the PCI transaction by deasserting $\overline{\text{IRDY}}$. The host bridge is posting the write data (it will be written to SDRAM sometime later).
- **Clock #7:** The PCI host bridge has synchronized the first PCI data phase (Clock #4) and requests access to the SDRAM controller.
- **Clock #9:** The SDRAM controller is granted to the PCI host bridge and the PCI bus data can be written to SDRAM. The hb_gnt signal may be delayed if the Am5$_X$86 CPU or GP-DMA is accessing SDRAM.

9.5.4.9.2 External PCI Master SDRAM Read (Delayed Transaction)

Figure 9-15 shows an external PCI bus master read transaction to the ÉlanSC520 microcontroller's SDRAM.

Figure 9-15 External PCI Master SDRAM Read (Delayed Transaction)



The following sequence annotates the external PCI master SDRAM read shown in Figure 9-15.

- **Clock #1:** An external PCI bus master initiates a read transaction to ÉlanSC520 microcontroller's SDRAM.
- **Clock #3:** The PCI host bridge target controller accepts the transaction by asserting $\overline{\text{DEVSEL}}$. $\overline{\text{TRDY}}$ is not asserted, because there is no data in the target read FIFO (this is a new transaction).
- **Clock #4:** The PCI host bridge target controller asserts $\overline{\text{STOP}}$, signaling a retry to the external PCI bus master. Because no data was transferred, the external PCI bus master is required to retry the transaction. (This figure assumes that the ÉlanSC520 microcontroller is configured for automatic delayed transactions.) The host bridge latches the transaction information and will prefetch the requested read data. This is now a delayed transaction, and the PCI bus master is required to relinquish bus ownership and re-arbitrate to retry the cycle. If there is already a previous delayed transaction pending, the current transaction will not be latched. Note that, in this example, $\overline{\text{STOP}}$ is asserted for two clock periods, because a target is required to keep this signal asserted until $\overline{\text{FRAME}}$ is deasserted.
- **Clock #7:** The PCI host bridge has synchronized the delayed transaction request and requests access to the SDRAM controller to prefetch the data requested by the external PCI master.
- **Clock #8:** The CPU bus is granted to the PCI host bridge, and the PCI bus data can be read from SDRAM. The hb_gnt signal may be delayed if the $Am5_x86$ CPU or GP-DMA controller is accessing SDRAM. The host bridge prefetches up to the next cache line in response to a memory-read or memory-read-line command and up to 64 doublewords in response to a memory-read-multiple command.

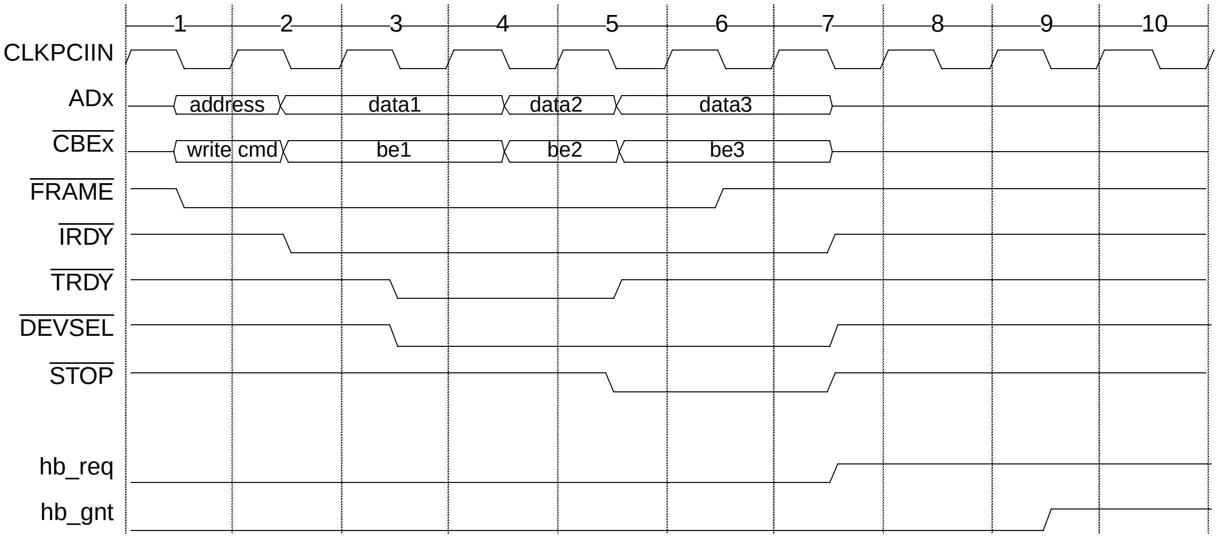
- **Clock #12:** The external PCI bus master retries the delayed transaction. While a delayed transaction is pending, all other read transactions are retried by the host bridge (these are not latched as delayed transactions). Write transactions, however, are allowed to complete and are put into the write FIFO. If the external PCI master retries the delayed transaction before the host bridge has read the first doubleword of data into the target read FIFO, the host bridge issues another retry to the external PCI bus master (and keeps issuing retries until the first doubleword of data has been read into the target read FIFO).
- **Clock #14:** By now, the PCI host bridge has read in the first doubleword of data into the target read FIFO and recognizes this transaction as the pending delayed transaction. The host bridge asserts $\overline{\text{DEVSEL}}$ to claim the transaction.
- **Clock #16:** The PCI host bridge asserts $\overline{\text{TRDY}}$ for the first data phase of the transaction. After the first data phase, the host bridge can burst up to the next cache-line boundary without adding anymore wait states. After each cache line, the PCI host bridge may insert wait states if the target read FIFO becomes empty.
- **Clocks #17–#19:** The external PCI master reads the data from the PCI host bridge. (Although the figure shows it this way, note that SDRAM having the data by Clock #17 is quite optimistic.) The external PCI bus master can insert wait states into the transaction by deasserting $\overline{\text{IRDY}}$. Clock #19 is the last data requested by the external PCI bus master ($\overline{\text{FRAME}}$ deasserted, $\overline{\text{IRDY}}$ asserted).

9.5.4.9.3 PCI Host Bridge Target Disconnect

Figure 9-16 shows the PCI host bridge target controller issuing a disconnect to an external PCI bus master. This example shows a disconnect during an external PCI bus master write cycle, but the mechanism is the same for external PCI bus master read cycles. The only difference is that Clock #2 is a turnaround cycle on AD31–AD0 bus. The PCI host bridge issues a disconnect if:

- During an external PCI bus master write cycle, the write FIFO becomes full or 64 consecutive doublewords have been written by the bus master.
- During an external PCI bus master read cycle, the target read FIFO becomes empty—Note that for memory-read and memory-read-line commands, the PCI host bridge can burst up to the next cache-line boundary without disconnecting; for memory-read-multiple commands, the PCI host bridge can burst 64 doublewords without disconnecting. If the external PCI bus master wishes to burst beyond these limits, then the PCI host bridge may issue a disconnect.

Figure 9-16 PCI Host Bridge Target Disconnect



The following sequence annotates the PCI host bridge target disconnect shown in Figure 9-16.

- **Clock #1:** An external PCI bus master initiates a write transaction to ÉlanSC520 microcontroller SDRAM.
- **Clock #3:** The PCI host bridge always asserts DEVSEL with medium timing and asserts TRDY, signaling it is ready to accept data (provide data for external PCI bus master reads).
- **Clocks #3–#4:** Both TRDY and IRDY are sampled asserted, signaling a valid data phase. External master write data will be accepted by the PCI host bridge (or the external master will read data for external PCI bus master read cycles).
- **Clock #5:** The PCI host bridge write FIFO is full (or the target read FIFO is empty for external PCI bus master read cycles), so TRDY is deasserted and STOP is asserted, signaling a disconnect. Because TDRY is deasserted, Clock #5 is the last valid data phase. Note that FRAME is still asserted, signaling that the external PCI bus master is requesting to burst more data.
- **Clock #6:** The external PCI bus master deasserts FRAME in response to STOP being sampled asserted. Because TRDY is deasserted, this is not a valid data phase and no data will be transferred.
- **Clock #7:** The external PCI bus master deasserts IRDY and the PCI host bridge deasserts STOP and DEVSEL, ending the PCI bus transaction. The host bridge has synchronized the first PCI bus data phase (Clock #3) and requests access to the SDRAM controller.
- **Clock #9:** The CPU bus is granted to the PCI host bridge, and the PCI bus data can be written to SDRAM. The hb_gnt signal may be delayed if the Am5$_x$86 CPU or GP-DMA controller is accessing SDRAM.

9.5.5 Interrupts

The PCI host bridge has one maskable interrupt request signal and one NMI signal routed to the ÉlanSC520 microcontroller's interrupt controller. These interrupt signals are shared by the arbiter, and PCI master and target controllers of the host bridge. Each interrupt source (both master and target sources) can be individually programmed to generate a maskable interrupt instead of a non-maskable interrupt request.

The following conditions can be programmed to generate an interrupt by the PCI host bridge *master* controller:

- Detected parity error during a read cycle
- Received parity during a write cycle or during the address phase of a read cycle
- Retry time-out counter expired
- Cycle was terminated with master abort
- Cycle was terminated with target abort
- System error ($\overline{\text{SERR}}$) pin asserted by PCI bus device

When an interrupt is generated, the *address of the cycle* during which the interrupt condition was detected is stored in the Host Bridge Master Interrupt Address (MSTINTADD) register (MMCR offset 6Ch), and the *command* is stored in the Host Bridge Master Interrupt Status (HBMSTIRQSTA) register (MMCR offset 68h). If multiple interrupt conditions are pending, the registers store the information for the first interrupt condition only. If multiple interrupts are pending, there is no indication to which interrupts the Master Interrupt Command Identification (M_CMD_IRQ_ID) and Master Interrupt Address Identification (M_AD_IRQ_ID) fields correspond. Status bits in the Status/Command (PCISTACMD) register (PCI index 04h) are also set when error conditions are detected. These bits are set whenever the error condition is detected, regardless of the interrupt enable bits.

The following conditions can be programmed to generate an interrupt by the host bridge *target* controller:

- Detected parity error during a data phase of a write cycle
- Detected parity error during an address phase
- Delayed transaction time-out—2^{15} clocks have expired without an external PCI master retrying a delayed transaction

When an interrupt is generated, the *REQ/GNT number of the PCI bus master* that caused the error is stored in the Host Bridge Target Interrupt Status (HBTGTIRQSTA) register (MMCR offset 64h). If multiple interrupt conditions are pending, the Target Interrupt Identification (T_IRQ_ID) field stores only the information for the first interrupt condition. If multiple interrupts are pending, there is no indication to which interrupt the T_IRQ_ID field corresponds. The appropriate status bits in the Status/Command (PCISTACMD) register (PCI index 04h) are also set when error conditions are detected. These bits are set whenever the error condition is detected, regardless of the interrupt enable bits.

See Chapter 15, "Programmable Interrupt Controller", for further details on the programming and routing of interrupt requests. See Chapter 8, "System Arbitration", for further details on arbitration.

9.5.6 Latency

PCI bus latency issues are described separately for the CPU and external PCI bus masters.

- *Master latency* refers to the case when the ÉlanSC520 microcontroller's Am5$_X$86 CPU is the master on the PCI bus.
- *Target latency* refers to the case when the ÉlanSC520 microcontroller is a PCI bus target accessed by external PCI bus masters.

9.5.6.1 Master Latency

The posted write buffer allows Am5$_X$86 CPU memory-write cycles to complete without incurring the PCI bus transaction latency. Any other cycle between the CPU and the PCI bus (memory read, I/O write, I/O read) must complete on the PCI bus before ready is returned to the Am5$_X$86 CPU. Note that write posting must be disabled while the ÉlanSC520 microcontroller is operating in nonconcurrent arbitration mode. See Chapter 8, "System Arbitration", for details on nonconcurrent mode arbitration.

The target being accessed may retry the Am5$_X$86 CPU cycle (target busy) multiple times, which would delay the Am5$_X$86 CPU. This performance penalty can be limited by configuration of the Am5$_X$86 CPU using the Master Retry Time-Out (M_RETRY_TO) field in the Master Retry Time-Out (PCIMRETRYTO) register (PCI index 41h), which limits the number of times the PCI host bridge retries a transaction before returning the $\overline{\text{rdy}}$ signal to the Am5$_X$86 CPU. Note that the master retry count configuration must not be changed except during PCI bus initialization after a system or programmable reset.

The Am5$_X$86 CPU typically performs non-burst read transactions to the PCI bus, because PCI bus memory is noncacheable (write transactions to PCI are *always* non-burst). There are a few cases when the CPU bursts up to two doublewords on a read transaction. For simplicity, in these cases, the PCI host bridge breaks up any Am5$_X$86 CPU burst read cycles into single doubleword read transactions on the PCI bus, which also slows down the Am5$_X$86 CPU read performance to the PCI bus. Because the PCI host bridge master controller performs single data phase transactions only, the master latency timer is not implemented.

9.5.6.2 Target Latency

Write posting and delayed transactions in the PCI host bridge target controller allow external PCI bus master cycles to complete without incurring SDRAM access latency. Without write posting and delayed transactions, the PCI host bridge target controller would insert wait states, while arbitrating for use of the SDRAM controller.

Delayed transaction support allows this time spent arbitrating for the CPU bus and the SDRAM controller transaction to be reallocated to another bus master, rather than forcing the first bus master to remain in a long wait state period. Instead, the first bus master's request is latched and placed in the delayed transaction queue for processing by the PCI host bridge, and the bus master is forced off of the PCI bus with a retry, at which point the PCI bus arbiter may grant the bus to another PCI bus master. The second PCI bus master could perform a peer-to-peer transfer or memory write to SDRAM while the PCI host bridge continues to process the first bus master's request.

Delayed transactions avoid the wasted bus bandwidth that may occur if the PCI host bridge's response to the transaction exceeded the specified 32 PCI bus clocks (16 for non-host bridge devices), at which point the PCI bus master would be retried anyway (thus wasting 16–32 PCI bus clocks).

The concurrent nature of ÉlanSC520 microcontroller's system architecture is such that a SDRAM read request from an external PCI master may be delayed. The reasons for this delay are:

- The Am5_x86 CPU may be currently accessing ROM, GP bus, or SDRAM.
- The SDRAM controller may be currently servicing a SDRAM refresh.
- A DMA transaction may be in progress between a GP-DMA initiator and SDRAM. Such transactions are variable in length and subject to the programmed DMA transfer mode. For example, in block or demand mode, the DMA transfer cannot be preempted.

Note: *Large GP Bus DMA transfers in demand or block mode, or very slow GP bus cycles (initiated via programmable GP bus timing, or by deasserting the GPRDY signal) can cause the PCI host bridge target controller to violate the 10 µs memory write maximum completion time limit set in the PCI Local Bus Specification, Revision 2.2. In PCI bus 2.2-compliant designs, software must limit the length of GP bus cycles and GP bus DMA demand- or block-mode transfers.*

Delayed transactions *can* increase Am5_x86 CPU and GP-DMA latency to SDRAM because of prefetching in response to memory-read-multiple commands. For example, when a prefetch of 64 doublewords occurs during a PCI bus master memory-read-multiple cycle of the ÉlanSC520 microcontroller's SDRAM, neither the Am5_x86 CPU or the GP-DMA controller has access to the CPU bus. After the initial prefetch of 64 doublewords, the PCI host bridge relinquishes ownership of the CPU bus.

9.6 INITIALIZATION

The PCI bus $\overline{\text{RST}}$ signal, when asserted, resets the ÉlanSC520 microcontroller's PCI host bridge, as well as any external PCI bus devices.

The $\overline{\text{RST}}$ signal is asserted in response to a system reset (see "System Reset" on page 6-4) or by setting the PCI_RST bit in the Host Bridge Control (HBCTL) register (MMCR offset 60h). These reset sources assert and deassert the $\overline{\text{RST}}$ signal asynchronously to the PCI bus clock.

When the $\overline{\text{RST}}$ signal is asserted, the PCI host bridge master controller and target controller state machines go to their idle states, and the host bridge FIFOs are purged. The PCI host bridge register bits are reset to their default states due to system reset, but the PCI_RST bit does not reset the PCI host bridge configuration registers or the host bridge status bits (see the register descriptions in the *Élan™SC520 Microcontroller Register Set Manual*, order #22005).

After reset, the PCI host bridge target controller is disabled, but the host bridge responds to configuration transactions from the Am5_x86 CPU. Note that the PCI host bridge master controller is always enabled.

After reset the following steps should be taken to configure the PCI host bridge. Configure the PCI host bridge first; then, configure the external PCI bus devices.

1. Configure the PCI host bridge.
   a. Program the desired ÉlanSC520 microcontroller arbitration mode, including concurrency mode and PCI bus master arbitration priorities, etc. See "Initialization" on page 8-22, for more detailed information on arbitration.
   b. Program the Programmable Address Region (PAR) registers, if required. See Chapter 4, "System Address Mapping", for details on programming PCI bus memory space.

c. Program the ÉlanSC520 microcontroller-specific PCI host bridge configuration (write posting, retry time-out counter, interrupts, etc.). Note that write-posting must be disabled while operating in nonconcurrent arbitration mode. See Chapter 8, “System Arbitration”, for further details on nonconcurrent mode arbitration.

d. Program the standard PCI bus configuration registers. See “Configuration Information” on page 9-9 for more information.

2. Configure the external PCI bus devices.

In general, PCI host bridge configuration bits should not be changed except during a PCI bus initialization after a system or programmable reset.

A PCI bus 2.2-compliant target is not required to meet the normal initial latency time limit if it is accessed during the 2^{25} clock periods (about one second) following $\overline{\text{RST}}$ signal deassertion. During this time, an addressed target is permitted to do any of the following:

- Initiate a retry.
- Claim the access and hold in wait states until ready to respond.
- Ignore the access.

A device that ignores the access is essentially not recognized if the initialization software tries to configure it too soon after $\overline{\text{RST}}$ is deasserted, resulting in an incomplete system configuration. To support such devices, the initialization software might need to include a delay to ensure that 2^{25} clock periods pass before PCI devices are configured.

CHAPTER

10 SDRAM CONTROLLER

10.1 OVERVIEW

The ÉlanSC520 microcontroller includes an integrated SDRAM controller.

Features include:

- SDRAM (synchronous DRAM) support
- 3.3-V DC 66-MHz SDRAM or faster (16 Mbit through 256 Mbit)
- Achieves 3-1-1-1 read bursts on SDRAM (page hit for all device speed grades with $\overline{\text{CAS}}$ latency (C_L) = 2)
- Support for up to four banks, each bank independently programmed for size and symmetry (symmetric and asymmetric SDRAMs)
- Up to 256 Mbytes of SDRAM
- Optional SDRAM refresh during reset
- SDRAM auto refresh
- Error Correction Code (ECC) support (single-bit correct/multi-bit detect)
- SDRAM write buffering that supports write-merging, write-collapsing, and read-merging
- Read buffer with read-ahead feature for SDRAM read prefetching
- Read-around-write support that gives read priority over posted writes when the write buffer is enabled

10.2 BLOCK DIAGRAM

The SDRAM controller and its interface to the system SDRAM, along with the write buffer and the read buffer, are shown in Figure 10-1. (The write buffer and read buffer are described in Chapter 11.) Figure 10-2 shows a more detailed block diagram of the SDRAM controller subsystem.

10.3 SYSTEM DESIGN

The SDRAM controller of the ÉlanSC520 microcontroller supports SDRAM devices only. Figure 10-3 illustrates the connection of the SDRAM signals from the ÉlanSC520 microcontroller to the SDRAM banks.

Although the data bus width is only 32-bits in the ÉlanSC520 microcontroller, 64-bit (168-pin DIMMs) memory modules can be used. Each 168-pin DIMM can be used as a pair of banks. By appropriately connecting the $\overline{\text{SCS3}}$–$\overline{\text{SCS0}}$ signals to the SDRAM DIMM module, 168-pin modules can be used in an ÉlanSC520 microcontroller system.

Figure 10-4 shows an example configuration of a 168-pin SDRAM DIMM used as two banks. For the DIMM in this example, 8-bit devices are used. A DIMM configured for ECC is not shown.

Figure 10-1 SDRAM Controller Block Diagram

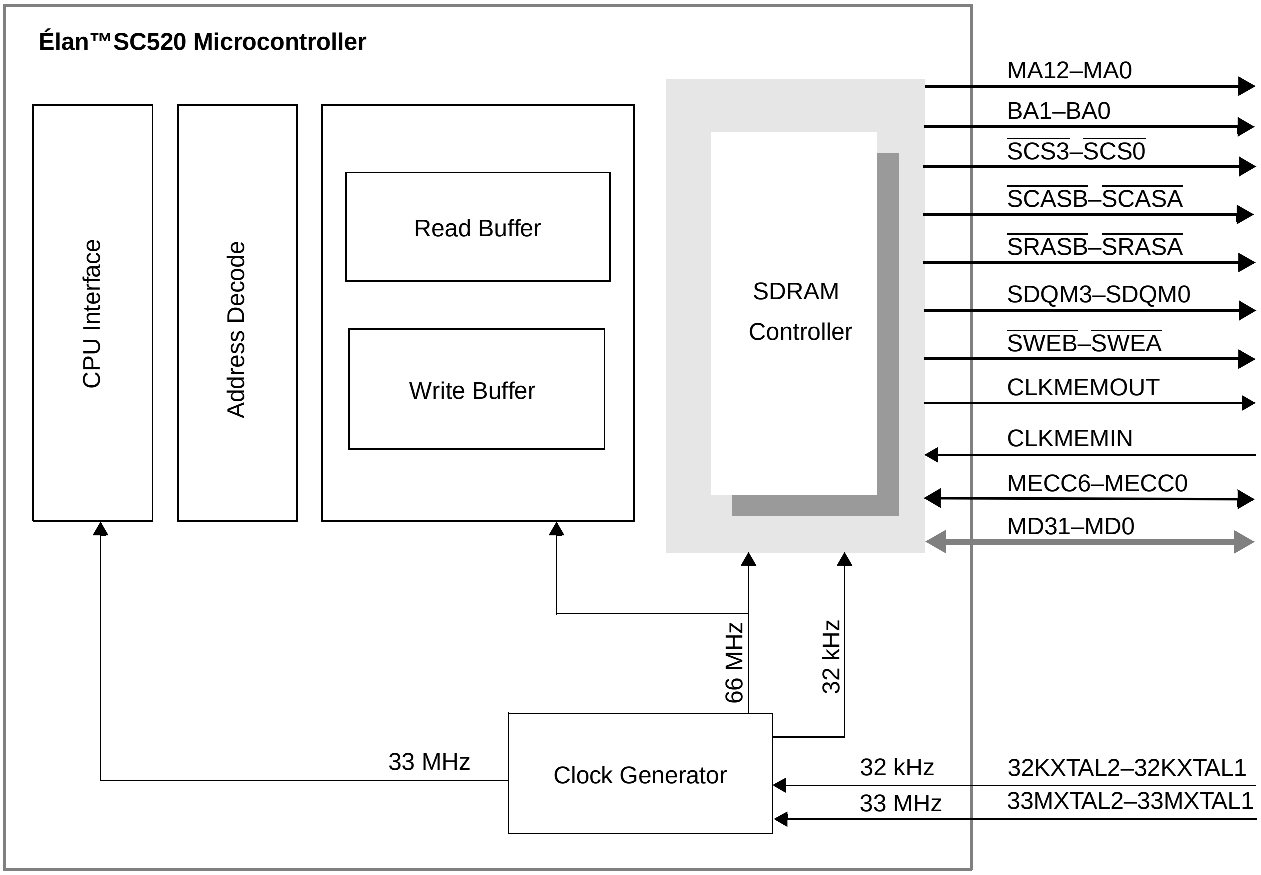
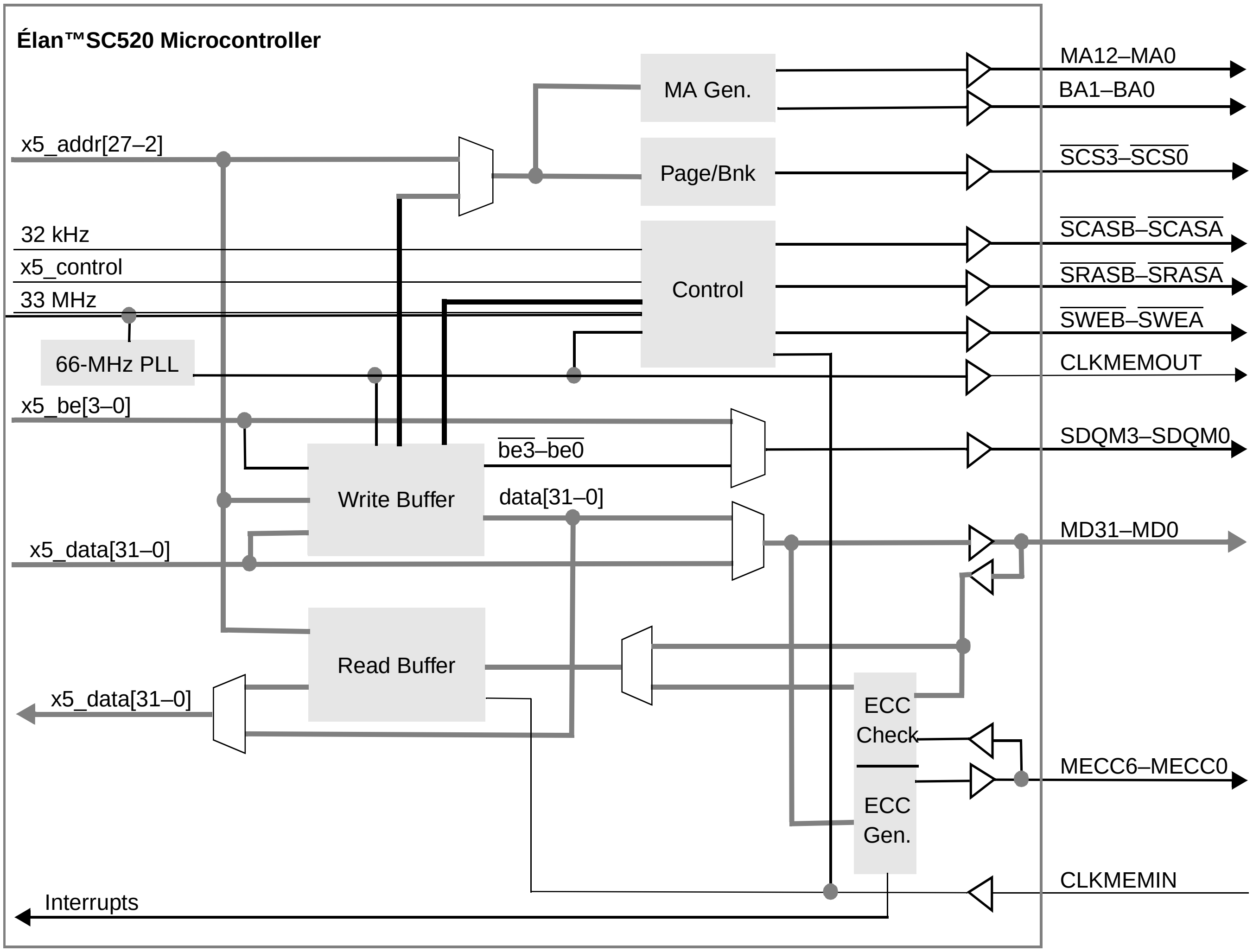


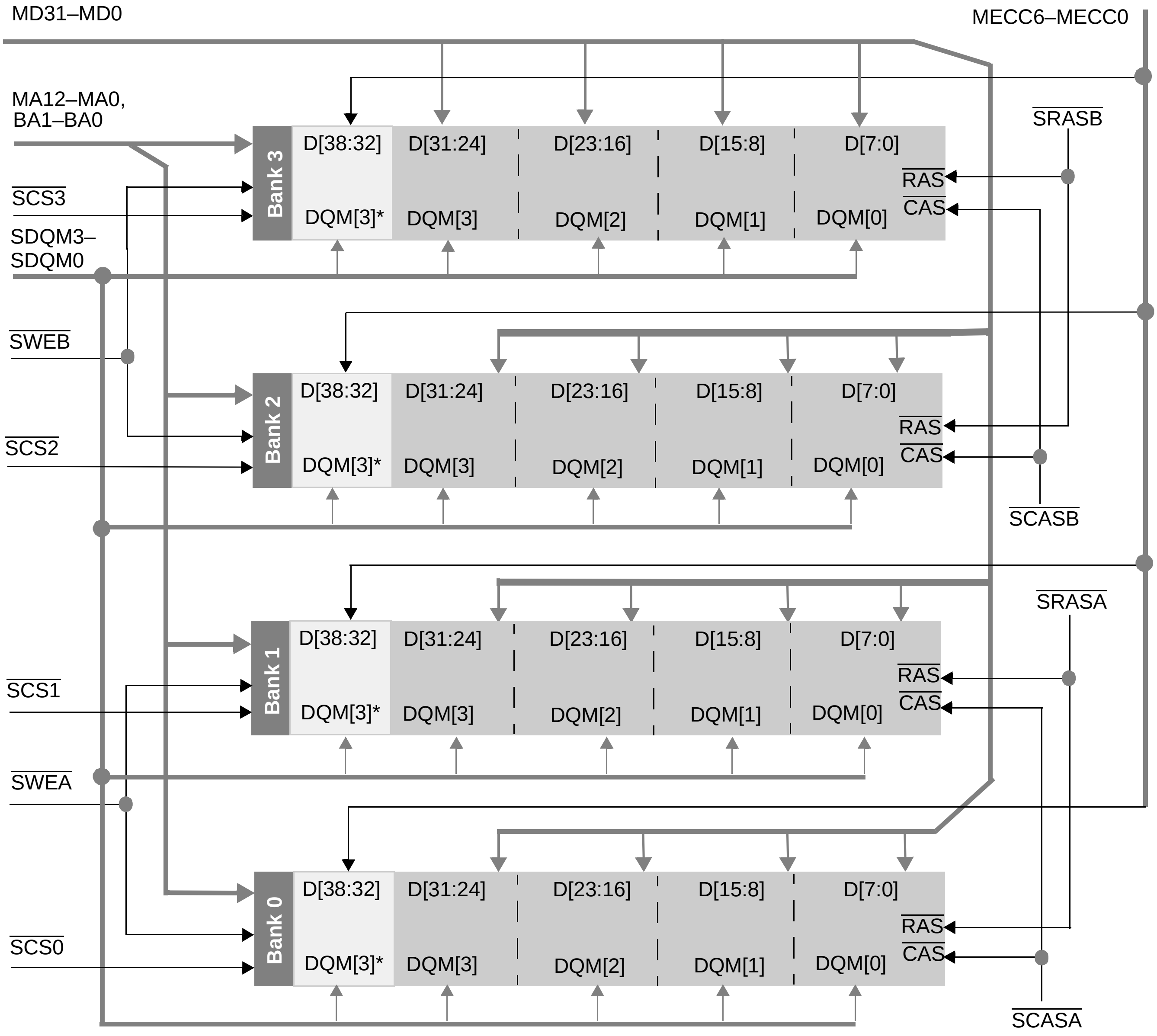
Figure 10-2 Detailed Block Diagram of SDRAM Controller



Notes:

SDRAM controller trace and test logic is not shown.

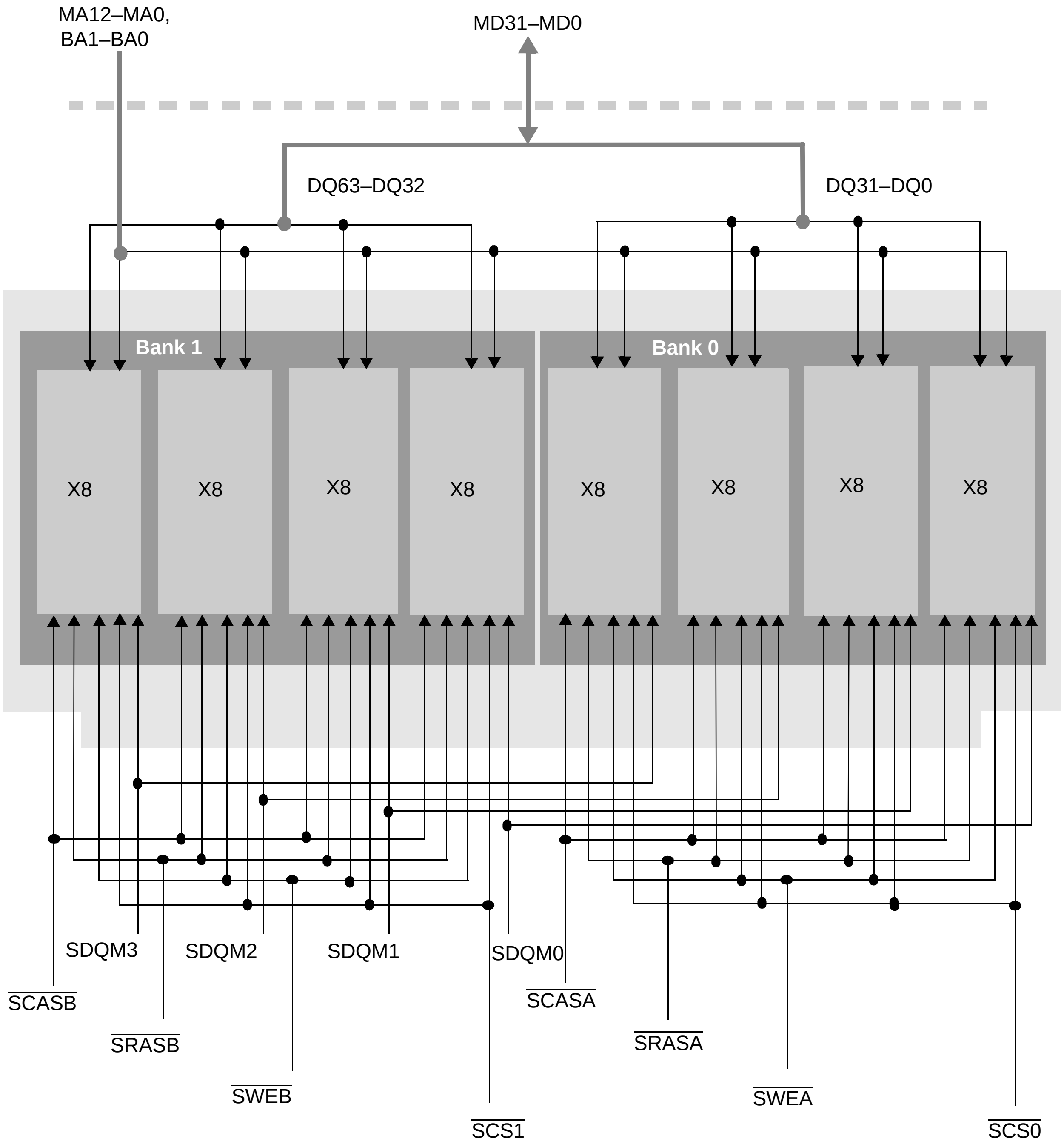
Figure 10-3 SDRAM Bank Configuration



Notes:

** ECC is optional. Since the entire doubleword is always written to the SDRAM during a read-modify-write operation (see “Error Correction Code (ECC)” on page 10-16), any one of the four SDQM signals can be connected to the DQM of the device that stores the 7-bit check word.*

Figure 10-4 Example Configuration of a 168-Pin SDRAM DIMM



10.3.1 SDRAM Pins

The SDRAM interface pins are dedicated to supporting SDRAM devices only.

Four chip select signals, $\overline{\text{SCS3}}$–$\overline{\text{SCS0}}$, are provided for independent bank selection.

The $\overline{\text{SRASA}}$–$\overline{\text{SRASB}}$, $\overline{\text{SCASA}}$–$\overline{\text{SCASB}}$, and $\overline{\text{SWEA}}$–$\overline{\text{SWEB}}$ signals are device command signals that are encoded by the SDRAM controller to send a command to the SDRAM devices. Each device in the array must sample these signals.

- Since this may result in heavy loading, two $\overline{\text{SRAS}}$ and two $\overline{\text{SCAS}}$ signals are provided to allow splitting load capacitance on these pins among the banks.

  - For example, banks 0 and 1 can share the $\overline{\text{SRASA}}$ and $\overline{\text{SCASA}}$ signal.
  - Likewise, banks 2 and 3 can share the $\overline{\text{SRASB}}$ and the $\overline{\text{SCASB}}$ signal.
- Two $\overline{\text{SWE}}$ signals are also provided to alleviate single pin loading.
  - For example, banks 0 and 1 can share the $\overline{\text{SWEA}}$ signal, and banks 2 and 3 can share the $\overline{\text{SWEB}}$ signal.

The four SDQM lines, SDQM3–SDQM0, provide byte masking.

- Each of the four SDQM3–SDQM0 signals is associated with one byte of four throughout the array. Each SDQMx signal provides an input mask signal for write accesses and an output enable signal for read accesses.

See Figure 10-3 on page 10-4, which illustrates the connection of SDRAM signals from the ÉlanSC520 microcontroller to the external SDRAM banks. Since the SDRAM controller shares the MD31–MD0 data bus with the ROM/Flash controller, the SDRAM controller guarantees the SDQM3–SDQM0 signals are forced inactive to make sure the SDRAM devices do not contend with the ROM or Flash devices that may share the data bus.

10.3.2 SDRAM Clocking

The SDRAM device's clock is sourced from the SDRAM controller interface of the ÉlanSC520 microcontroller. As shown in Figure 10-1 on page 10-2, there are two clock pins dedicated for the SDRAM interface.

- CLKMEMOUT is a 66-MHz clock.
- CLKMEMIN must be a direct feedback version of CLKMEMOUT.

The SDRAM controller's data buffers use CLKMEMIN to latch read data coming from the SDRAM devices. CLKMEMIN is used to compensate for delays associated with board loading and external buffering (to allow for read data flight time from the SDRAM device). The allowable delay between CLKMEMOUT and CLKMEMIN is –0.5 to +6.0 ns.

The following describes a typical scenario for SDRAM systems used with the ÉlanSC520 microcontroller. These are general guidelines to demonstrate system considerations and are not intended for use as system implementations.

The CLKMEMOUT pin has a 24-mA driver and is capable of driving a 50-pF load directly, without requiring an external clock driver/buffer and still remain under the maximum allowable delay of 6 ns. A CLKMEMOUT load above 50 pF may result in delays greater than 6 ns that could jeopardize data integrity. The 50-pF load includes all loads presented to the CLKMEMOUT pin such as board routing (between CLKMEMOUT and CLKMEMIN), DIMM connector load, and SDRAM device load.

Table 10-1 shows estimated bank loads as they pertain to SDRAM device data widths. As shown in Table 10-1, a bank composed of 4-bit devices presents a greater load to the CLKMEMOUT pin than a bank composed of 8-bit devices. This table does not include board or connector loads.

Table 10-1 SDRAM Clock Loading Estimates Based on Device Width

	Device Width			
	4-Bit	**8-Bit**	**16-Bit**	**32-Bit**
Device count (per bank)	8	4	2	1
Total SDRAM clock loading (pF)	32	16	8	4

Figure 10-5 shows a lightly loaded system. Typically, this delay can be implemented as fast buffers, capacitors, series resistors, etc. or as a short.

Figure 10-5 SDRAM Clock Generation

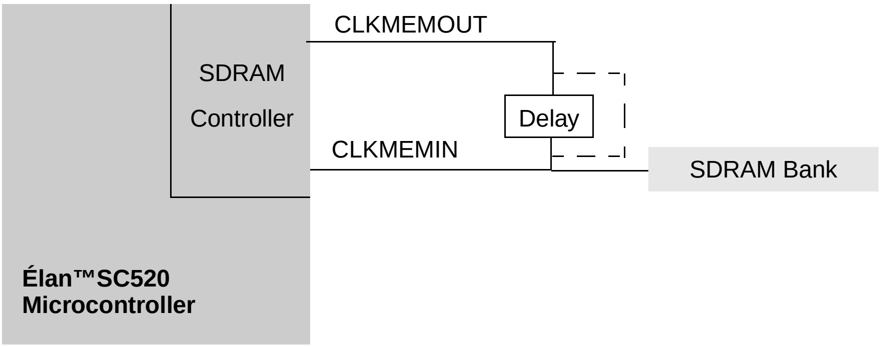
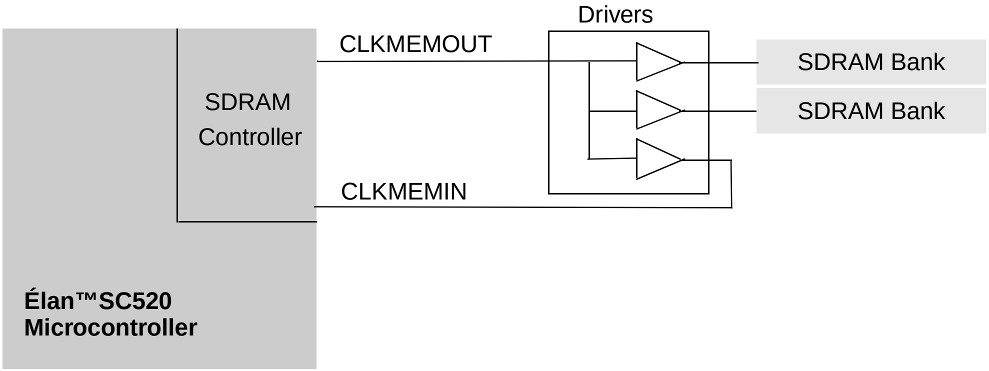


Figure 10-6 shows an example of a two-bank SDRAM system that uses an external clock driver. The clock driver is used to buffer CLKMEMOUT to support the load of multiple banks of SDRAM. A buffered version of CLKMEMOUT is returned on CLKMEMIN to compensate for the clock skew presented by the clock driver.

Figure 10-6 Alternate SDRAM Clock Generation with External Clock Driver



The delays that the system designer must take into consideration are identified by this equation:

$$T_{AC} + T_{SKEW} + T_{CK\_LD} + T_{D\_LD} <= T_{CK}$$

where:

T_{AC} : Access time of SDRAM device (not impacted by board design)

T_{SKEW}: The delay between CLKMEMOUT to CLKMEMIN

T_{CK_LD}: Additional clock delay due to loading

T_{D_LD}: Data delay due to loading

T_{CK}: SDRAM memory clock

See the *Élan™SC520 Microcontroller Data Sheet*, order #22003, for timing tables and additional timing diagrams.

10.3.3 SDRAM Loading

Table 10-2 through Table 10-5 show estimated capacitances for the SDRAM devices that the ÉlanSC520 microcontroller can support. (See Table 10-8 on page 10-13 for a listing of the SDRAM devices supported by ÉlanSC520 microcontroller.) The tables are broken up for SDRAM device data width for clarity. The purpose of these tables is to identify SDRAM loading as it applies to various bank configurations. The ÉlanSC520 microcontroller provides some flexibility in signal drive strength to allow the user to optimize performance, depending on the SDRAM array configuration.

In the estimated capacitance tables, the input capacitance of SRASx, SCASx, SWEx, MAx, BAx, SDQMx, and SCSx for a single device was assumed to be 5 pF. 4 pF was used for the CLK signal. The MDx signals are assumed to be 6 pF. These tables do not account for board trace capacitance. It is assumed in these tables that both pins provided for a control signal, e.g., SRASA–SRASB, SCASA–SCASB, and SWEA–SWEB are split across banks evenly.

As can be seen in the tables, a 4-bank configuration of 16-bit devices has a loading of less than 50 pF for any signal, but for a 4-bank configuration of 4-bit devices, the capacitance of the interface increases. The ÉlanSC520 microcontroller provides programmable drive strength buffers on all address, data, and control signals to support varying SDRAM device loads. See “SDRAM Control Configuration” on page 10-18 for more details.

Table 10-2 Estimated Capacitance (4-Bit SDRAM Devices)

Number of 32-Bit Banks	CLK Loading (pF)	SRASx Loading (pF)	SCASx Loading (pF)	SCSx Loading (pF)	SWEx Loading (pF)	SDQMx Loading (pF)	MAx/BAx Loading (pF)	MDx Loading (pF)
1	32	40	40	40	40	10	40	6
2	64	40	40	40	40	20	80	12
3	96	80	80	40	80	30	120	18
4	128	80	80	40	80	40	160	24

Notes:

Capacitive loads shown in the table above are derived from an estimated SDRAM pin capacitance value of 5 pF for SRASx, SCASx, SWEx, MAx, BAx, SDQMx, and SCSx; 4 pF for the CLK signal; and 6 pF for the MDx signals, per device.

Table 10-3 Estimated Capacitance (8-Bit SDRAM Devices)

Number of 32-Bit Banks	CLK Loading (pF)	SRASx Loading (pF)	SCASx Loading (pF)	SCSx Loading (pF)	SWEx Loading (pF)	SDQMx Loading (pF)	MAx/BAx Loading (pF)	MDx Loading (pF)
1	16	20	20	20	20	5	20	6
2	32	20	20	20	20	10	40	12
3	48	40	40	20	40	15	60	18
4	64	40	40	20	40	20	80	24

Notes:

Capacitive loads shown in the table above are derived from an estimated SDRAM pin capacitance value of 5 pF for SRASx, SCASx, SWEx, MAx, BAx, SDQMx, and SCSx; 4 pF for the CLK signal; and 6 pF for the MDx signals, per device.

Table 10-4 Estimated Capacitance (16-Bit SDRAM Devices)

Number of 32-Bit Banks	CLK Loading (pF)	$\overline{\text{SRASx}}$ Loading (pF)	$\overline{\text{SCASx}}$ Loading (pF)	$\overline{\text{SCSx}}$ Loading (pF)	$\overline{\text{SWEx}}$ Loading (pF)	SDQMx Loading (pF)	MAx/BAx Loading (pF)	MDx Loading (pF)
1	8	10	10	10	10	5	10	6
2	16	10	10	10	10	10	20	12
3	24	20	20	10	20	15	30	18
4	32	20	20	10	20	20	40	24

Notes:

Capacitive loads shown in the table above are derived from an estimated SDRAM pin capacitance value of 5 pF for SRASx, SCASx, SWEx, MAx, BAx, SDQMx, and SCSx; 4 pF for the CLK signal; and 6 pF for the MDx signals, per device.

Table 10-5 Estimated Capacitance (32-Bit SDRAM Devices)

Number of 32-Bit Banks	CLK Loading (pF)	$\overline{\text{SRASx}}$ Loading (pF)	$\overline{\text{SCASx}}$ Loading (pF)	$\overline{\text{SCSx}}$ Loading (pF)	$\overline{\text{SWEx}}$ Loading (pF)	SDQMx Loading (pF)	MAx/BAx Loading (pF)	MDx Loading (pF)
1	4	5	5	5	5	5	5	6
2	8	5	5	5	5	10	10	12
3	12	10	10	5	10	15	15	18
4	16	10	10	5	10	20	20	24

Notes:

Capacitive loads shown in the table above are derived from an estimated SDRAM pin capacitance value of 5 pF for SRASx, SCASx, SWEx, MAx, BAx, SDQMx, and SCSx; 4 pF for the CLK signal; and 6 pF for the MDx signals, per device.

As can be seen clearly from the capacitance tables, as more SDRAM devices are connected to the SDRAM controller interface signals on the ÉlanSC520 microcontroller, loading on all these signals increases. Note that the numbers reflect only the actual device capacitance, and not circuit board trace or buffer capacitance.

The SDRAM controller’s data bus (MD31–MD0) is shared with the ROM/Flash controller. It is advisable to consider loading issues on the MD31–MD0 bus when both SDRAM and ROM/Flash devices are installed. Heavy loading by SDRAM and ROM/Flash devices may slow down the SDRAM timings and cause data corruption.

When ECC devices are not installed, it is advisable to add individual 10-Kohm pulldown resistors on the MECC6–MECC0 bus to prevent the bus from floating during read access.

10.4 REGISTERS

A summary listing of the registers used to control the SDRAM configuration are shown in Table 10-6.

Table 10-6 SDRAM Controller Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
SDRAM Control	DRCCTL	10h	Operation mode select, refresh enable, refresh rate select, SDRAM write buffer test mode enable
SDRAM Timing Control	DRCTMCTL	12h	$\overline{\text{RAS}}$-to-$\overline{\text{CAS}}$ delay, $\overline{\text{RAS}}$ precharge, $\overline{\text{CAS}}$ latency
SDRAM Bank Configuration	DRCCFG	14h	Bank count select, address column width requirements for each bank
SDRAM Bank 0–3 Ending Address	DRCBENDADR	18h	Independent bank ending configurations and enables for banks 0, 1, 2 and 3
ECC Control	ECCCTL	20h	ECC enable, interrupt enable for single-bit and multi-bit error detection
ECC Status	ECCSTA	21h	Single-bit and multi-bit error status
ECC Check Bit Position	ECCCKBPOS	22h	ECC data bit position in check bit or data bit fields
ECC Check Code Test	ECCCKTEST	23h	ECC check code override for test and error handler development
ECC Single-Bit Error Address	ECCSBADD	24h	Address where single-bit ECC error occurred
ECC Multi-Bit Error Address	ECCMBADD	28h	Address where multi-bit ECC error occurred
Drive Strength Control	DSCTL	C28h	I/O pad drive strength for $\overline{\text{SCS3}}$–$\overline{\text{SCS0}}$, $\overline{\text{SRASA}}$–$\overline{\text{SRASB}}$, $\overline{\text{SCASA}}$–$\overline{\text{SCASB}}$, $\overline{\text{SWEA}}$–$\overline{\text{SWEB}}$, SDQM3–SDQM0, MA12–MA0, BA1–BA0, MD31–MD0, MECC6–MECC0
ECC Interrupt Mapping	ECCMAP	D18h	ECC interrupt mapping to any of 22 available interrupt channels or NMI, ECC NMI enable control
Reset Configuration	RESCFG	D72h	Programmable SDRAM retention reset (PRGRESET pin enable)
Reset Status	RESSTA	D74h	Reset source status: PRGRESET pin

10.5 OPERATION

The ÉlanSC520 microcontroller supports up to four 32-bit banks of SDRAM, with a maximum capacity of 256 Mbytes. This integrated SDRAM controller interfaces gluelessly to most commodity synchronous DRAM (SDRAM) devices. Mixed symmetries are supported across all four banks.

The ÉlanSC520 microcontroller supports a column boundary method to accept a wide variety of SDRAM devices. The *column boundary* method requires only the device's column address width to define the device's page size and symmetry.

The symmetry of a device refers to its organization as defined by the number of columns and the number of rows.

- A device is termed *symmetric* if the number of columns and rows is equal (i.e., a square organization).
- A device is termed *asymmetric* if the number of rows exceeds the number of columns (i.e., a rectangular organization). No devices exist where the number of columns exceeds the number of rows.

The column boundary method allows the user to configure the ÉlanSC520 microcontroller to work with 16-Mbit, 64-Mbit, 128-Mbit, and 256-Mbit SDRAM densities (both 2-bank and 4-bank internal architectures) requiring 8-bit through 11-bit column address bits.

Error Correction Code (ECC) is also supported for SDRAM devices to ensure data integrity for these high-speed devices.

10.5.1 SDRAM Support

The ÉlanSC520 microcontroller sources a 66-MHz clock (CLKMEMOUT) to drive the SDRAM devices. An external clock driver can be used to buffer this clock output for heavily loaded systems. A return clock input (CLKMEMIN) is provided to control clock skew. See "SDRAM Clocking" on page 10-6 for detailed information on SDRAM clocking. Although the ÉlanSC520 microcontroller sources a 66-MHz clock, faster SDRAM devices are supported (83-MHz, 100-MHz, 125-MHz, etc.).

The SDRAM controller supports 16-Mbit, 64-Mbit, 128-Mbit, and 256-Mbit SDRAM densities with either 2-bank or 4-bank internal architectures.

- A $\overline{\text{CAS}}$ latency (C_L) option of either 2T or 3T is supported, where T refers to a 15-ns clock period when a 33.333-MHz crystal is used.
- SDRAM devices *must* be configured for a fixed *interleaved* burst length of four for reads and *single* writes.

See "SDRAM Control Configuration" on page 10-18 for detailed information on SDRAM configuration timing options.

The SDRAM controller services read and write requests on behalf of:

- Am5$_x$86 CPU
- PCI masters
- GP-DMA controller

With the exception of ECC read-modify-write cycles (due to SDRAM writes of less than a doubleword when ECC is enabled), all read requests to SDRAM occur as a read burst of four cycles at the interface, regardless of the amount of data requested by a master.

During read-modify-write cycles, the SDRAM burst read portion of the transaction is terminated early by the write cycle. This is independent of the enable state of the read-ahead feature of the read buffer, which is provided to increase read performance by prefetching data from SDRAM. See “Buffering” on page 10-17 for more information on the read buffer and associated read-ahead feature.

Write requests to SDRAM always occur as single data transfers, regardless of the amount of data written by a master. When the write buffer is enabled, all write transactions to SDRAM are initiated by the write buffer. The write buffer features write merging, write collapsing and read merging. See “Buffering” on page 10-17 for more information on the write buffer.

10.5.2 SDRAM Addressing

The ÉlanSC520 microcontroller asserts one of the four chip select signals, $\overline{SCS3}$–$\overline{SCS0}$, during access to one of the four memory banks. Table 10-7 shows the SDRAM memory address as a function of the system address for SDRAM devices.

The mapping of the system address into memory row and column addresses is influenced by the column address configuration provided for each bank.

- On page misses, a row address followed by a column address is generated during an SDRAM access.
- On page hits, only a column address is generated during an SDRAM access.

Table 10-7 shows the ÉlanSC520 microcontroller address mapping.

Table 10-7 Address Mapping to MAx Signals for SDRAM Devices

SDRAM (16 Mbit–256 Mbit)																	
SDRAM Configuration			**Bank Selection**		**MAx Pin Mapping**												
Column Address Width			**BA1**	**BA0**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
8	2-bank	Row	24	10	23	22	13	12	11	21	20	19	18	17	16	15	14
		Column	24	10			PC			9	8	7	6	5	4	3	2
	4-bank	Row	22	10	24	23	13	12	11	21	20	19	18	17	16	15	14
		Column	22	10			PC			9	8	7	6	5	4	3	2
9	2-bank	Row	25	11	24	23	13	12	22	21	20	19	18	17	16	15	14
		Column	25	11			PC	11	10	9	8	7	6	5	4	3	2
	4-bank	Row	23	11	25	24	13	12	22	21	20	19	18	17	16	15	14
		Column	23	11	25	24	PC		10	9	8	7	6	5	4	3	2
10	2-bank	Row	26	12	25	24	13	23	22	21	20	19	18	17	16	15	14
		Column	26	12			PC	11	10	9	8	7	6	5	4	3	2
	4-bank	Row	24	12	26	25	13	23	22	21	20	19	18	17	16	15	14
		Column	24	12			PC	11	10	9	8	7	6	5	4	3	2
11	2-bank	Row	27	13	26	25	24	23	22	21	20	19	18	17	16	15	14
		Column	27	13		12	PC	11	10	9	8	7	6	5	4	3	2
	4-bank	Row	25	13	27	26	24	23	22	21	20	19	18	17	16	15	14
		Column	25	13		12	PC	11	10	9	8	7	6	5	4	3	2

Notes: *PC refers to SDRAM precharge signaling. BA1–BA0 are the SDRAM Bank Address signals.*

10.5.2.1 Supported SDRAM Devices

The ÉlanSC520 microcontroller supports the SDRAM organizations listed in Table 10-8. (Note that SDRAM devices requiring less than 11 row address bits are not supported, and are not included in the table.)

This table includes *all* possible device organizations supported by the column boundary method, including those that may not be available at this time. As shown, the column boundary method allows the user to configure the ÉlanSC520 microcontroller to work with 16-Mbit, 64-Mbit, 128-Mbit, and 256-Mbit SDRAM densities (both 2-bank and 4-bank internal architectures), requiring 8-bit through 11-bit column address bits. Note that 14-bit (2 internal bank) devices can be supported by connecting the BA1 pin to the most significant address pin of the devices.

Note that *illegal* device symmetries have been omitted from Table 10-8. Illegal symmetries are those where the column width exceeds the row width dimension.

Table 10-8 SDRAM Devices Supported with Column Boundary Specification

Column Width	Density	Banks	Organization	Device Architecture	Device Count per Bank	Dimension Row: Col	MA/BA Width	Bank (32-Bit)
8-bit	16 Mbit	2	4M x 4	2M x 4 x 2-banks	8	13:8	14-bit	16 Mbytes
			2M x 8	1M x 8 x 2-banks	4	12:8	13-bit	8 Mbytes
			1M x 16	512K x 16 x 2-banks	2	11:8	12-bit	4 Mbytes
		4	4M x 4	1M x 4 x 4-banks	8	12:8	14-bit	16 Mbytes
			2M x 8	512K x 8 x 4-banks	4	11:8	13-bit	8 Mbytes
	64 Mbit	2	8M x 8	4M x 8 x 2-banks	4	14:8	15-bit	32 Mbytes
			4M x 16	2M x 16 x 2-banks	2	13:8	14-bit	16 Mbytes
			2M x 32	1M x 32 x 2-banks	1	12:8	13-bit	8 Mbytes
		4	8M x 8	2M x 8 x 4-banks	4	13:8	15-bit	32 Mbytes
			4M x 16	1M x 16 x 4-banks	2	12:8	14-bit	16 Mbytes
			2M x 32	512K x 32 x 4-banks	1	11:8	13-bit	8 Mbytes
	128 Mbit	2	8M x 16	4M x 16 x 2-banks	2	14:8	15-bit	32 Mbytes
			4M x 32	2M x 32 x 2-banks	1	13:8	14-bit	16 Mbytes
		4	8M x 16	2M x 16 x 4-banks	2	13:8	15-bit	32 Mbytes
			4M x 32	1M x 32 x 4-banks	1	12:8	14-bit	16 Mbytes
	256 Mbit	2	8M x 32	4M x 32 x 2-banks	1	14:8	15-bit	32 Mbytes
		4	8M x 32	2M x 32 x 4-banks	1	13:8	15-bit	32 Mbytes

Table 10-8 SDRAM Devices Supported with Column Boundary Specification (Continued)

Column Width	Density	Banks	Organization	Device Architecture	Device Count per Bank	Dimension Row: Col	MA/BA Width	Bank (32-Bit)
9-bit	16 Mbit	2	4M x 4	2M x 4 x 2-banks	8	12:9	13-bit	16 Mbytes
			2M x 8	1M x 8 x 2-banks	4	11:9	12-bit	8 Mbytes
		4	4M x 4	1M x 4 x 4-banks	8	11:9	13-bit	16 Mbytes
	64 Mbit	2	16M x 4	8M x 4 x 2-banks	8	14:9	15-bit	64 Mbytes
			8M x 8	4M x 8 x 2-banks	4	13:9	14-bit	32 Mbytes
			4M x 16	2M x 16 x 2-banks	2	12:9	13-bit	16 Mbytes
			2M x 32	1M x 32 x 2-banks	1	11:9	12-bit	8 Mbytes
		4	16M x 4	4M x 4 x 4-banks	8	13:9	15-bit	64 Mbytes
			8M x 8	2M x 8 x 4-banks	4	12:9	14-bit	32 Mbytes
			4M x 16	1M x 16 x 4-banks	2	11:9	13-bit	16 Mbytes
	128 Mbit	2	16M x 8	8M x 8 x 2-banks	4	14:9	15-bit	64 Mbytes
			8M x 16	4M x 16 x 2-banks	2	13:9	14-bit	32 Mbytes
			4M x 32	2M x 32 x 2-banks	1	12:9	13-bit	16 Mbytes
		4	16M x 8	4M x 8 x 4-banks	4	13:9	15-bit	64 Mbytes
			8M x 16	2M x 16 x 4-banks	2	12:9	14-bit	32 Mbytes
			4M x 32	1M x 32 x 4-banks	1	11:9	13-bit	16 Mbytes
	256 Mbit	2	16M x 16	8M x 16 x 2-banks	2	14:9	15-bit	64 Mbytes
			8M x 32	4M x 32 x 2-banks	1	13:9	14-bit	32 Mbytes
		4	16M x 16	4M x 16 x 4-banks	2	13:9	15-bit	64 Mbytes
			8M x 32	2M x 32 x 4-banks	1	12:9	14-bit	32 Mbytes
10-bit	16 Mbit	2	4M x 4	2M x 4 x 2-banks	8	11:10	12-bit	16 Mbytes
	64 Mbit	2	16M x 4	8M x 4 x 2-banks	8	13:10	14-bit	64 Mbytes
			8M x 8	4M x 8 x 2-banks	4	12:10	13-bit	32 Mbytes
			4M x 16	2M x 16 x 2-banks	2	11:10	12-bit	16 Mbytes
		4	16M x 4	4M x 4 x 4-banks	8	12:10	14-bit	64 Mbytes
			8M x 8	2M x 8 x 4-banks	4	11:10	13-bit	32 Mbytes
	128 Mbit	2	32M x 4	16M x 4 x 2-banks	8	14:10	15-bit	128 Mbytes
			16M x 8	8M x 8 x 2-banks	4	13:10	14-bit	64 Mbytes
			8M x 16	4M x 16 x 2-banks	2	12:10	13-bit	32 Mbytes
			4M x 32	2M x 32 x 2-banks	1	11:10	12-bit	16 Mbytes
		4	32M x 4	8M x 4 x 4-banks	8	13:10	15-bit	128 Mbytes
			16M x 8	4M x 8 x 4-banks	4	12:10	14-bit	64 Mbytes
			8M x 16	2M x 16 x 4-banks	2	11:10	13-bit	32 Mbytes
	256 Mbit	2	32M x 8	16M x 8 x 2-banks	4	14:10	15-bit	128 Mbytes
			16M x 16	8M x 16 x 2-banks	2	13:10	14-bit	64 Mbytes
			8M x 32	4M x 32 x 2-banks	1	12:10	13-bit	32 Mbytes
		4	32M x 8	8M x 8 x 4-banks	4	13:10	15-bit	128 Mbytes
			16M x 16	4M x 16 x 4-banks	2	12:10	14-bit	64 Mbytes
			8M x 32	2M x 32 x 4-banks	1	11:10	13-bit	32 Mbytes

Table 10-8 SDRAM Devices Supported with Column Boundary Specification (Continued)

Column Width	Density	Banks	Organization	Device Architecture	Device Count per Bank	Dimension Row: Col	MA/BA Width	Bank (32-Bit)
11-bit	64 Mbit	2	16M x 4	8M x 4 x 2-banks	8	12:11	13-bit	64 Mbytes
			8M x 8	4M x 8 x 2-banks	4	11:11	12-bit	32 Mbytes
		4	16M x 4	4M x 4 x 4-banks	8	11:11	13-bit	64 Mbytes
	128 Mbit	2	32M x 4	16M x 4 x 2-banks	8	13:11	14-bit	128 Mbytes
			16M x 8	8M x 8 x 2-banks	4	12:11	13-bit	64 Mbytes
			8M x 16	4M x 16 x 2-banks	2	11:11	12-bit	32 Mbytes
		4	32M x 4	8M x 4 x 4-banks	8	12:11	14-bit	128 Mbytes
			16M x 8	4M x 8 x 4-banks	4	11:11	13-bit	64 Mbytes
	256 Mbit	2	64M x 4	32M x 4 x 2-banks	8	14:11	15-bit	256 Mbytes
			32M x 8	16M x 8 x 2-banks	4	13:11	14-bit	128 Mbytes
			16M x 16	8M x 16 x 2-banks	2	12:11	13-bit	64 Mbytes
			8M x 32	4M x 32 x 2-banks	1	11:11	12-bit	32 Mbytes
		4	64M x 4	16M x 4 x 4-banks	8	13:11	15-bit	256 Mbytes
			32M x 8	8M x 8 x 4-banks	4	12:11	14-bit	128 Mbytes
			16M x 16	4M x 16 x 4-banks	2	11:11	13-bit	64 Mbytes

Notes:

Not all device organizations specified in this table are available at the time of this printing.

The SDRAM Bank Configuration (DRCCFG) register (MMCR offset 14h) has one bit (BNKx_BNK_CNT) to specify the internal number of banks and another bit field to specify the column address width (BNKx_COLWDTH) of the device. Table 10-9 shows suggested settings for these bit fields, given a device's column address width and internal bank count.

Table 10-9 Column Address Configuration Settings for SDRAM

Column Width	Banks	Internal Bank Count (BNKx_BNK_CNT)	Bank Column Address (BNKx_COLWDTH)
8-bit	2	0b	00b
	4	1b	00b
9-bit	2	0b	01b
	4	1b	01b
10-bit	2	0b	10b
	4	1b	10b
11-bit	2	0b	11b
	4	1b	11b

For example, if Bank 2 is composed of SDRAM devices organized as 2M x 8 x 4 banks (8 Mbyte x 8) with 4096 rows and 512 columns (9-bit), by using Table 10-9, the appropriate bank configuration for this 4-bank device is 1b for the BNK2_BNK_CNT field and 01b for the BNK2_COLWDTH field of the SDRAM Bank Configuration (DRCCFG) register.

10.5.2.2 Page Size

The page size of an SDRAM device is based on the column address width of the device. The ÉlanSC520 microcontroller address mapping takes advantage of the full page specified by the devices column address width. Table 10-10 lists the page size available based on the column address width specified. The page size in an SDRAM device applies for each internal bank.

Table 10-10 SDRAM Page Sizes

Column Width	Page Size for 32-Bit Banks
8-bit	1 Kbyte
9-bit	2 Kbytes
10-bit	4 Kbytes
11-bit	8 Kbytes

10.5.3 Error Correction Code (ECC)

The ÉlanSC520 microcontroller supports Error Correction Code (ECC) to check the integrity of transactions with the system SDRAM. ECC is implemented by a modified Hamming code. It corrects a single-bit error and detects all two-bit (called *multi-bit)* errors. The memory array must have check bits to implement ECC.

ECC operation requires that system memory be initialized. In this procedure, the boot code writes to every memory location, automatically generating valid ECC that is stored in the SDRAM check bits. If this procedure is not performed, errors will occur in the generation of the check bits when writing data smaller than a 32-bit doubleword or when reading uninitialized data.

The ECC circuit uses a modified Hamming code to generate a 7-bit check word from the 32-bit data word. This check word is stored along with the data word during the memory write cycle. During the memory read cycle, the 39-bit words from memory are processed by the ECC circuit to determine if errors have occurred in storing or retrieving data.

If there is a single-bit error in the 32-bit data word or check-bits, the ECC circuit flags an error, latches the error-generating address along with the bit position where the error was detected, and passes along the corrected data word to the requesting master. It does not write the corrected data back out to the SDRAM. It generates a maskable interrupt signal when a single-bit error is detected. This maskable interrupt signal is generated even if there is a single-bit error in the 7-bit check word.

Multi-bit errors are flagged but not corrected. These errors may occur in any two bits of the 39-bit word from memory (two errors in the 32-bit data word, two errors in the 7-bit check word, or one error in each word). A separate non-maskable interrupt is generated by the ECC logic for multi-bit errors.

These two interrupts are routed to the interrupt steering logic in the programmable interrupt controller. See Chapter 15, “Programmable Interrupt Controller”, for more details and further options.

If there is any write that is less than the full four bytes, there is a loss of performance due to ECC. The seven check-bits for any given ECC data field are generated over the entire field. In other words, all four bytes of data are taken into account in generating the seven check-bits associated with that data. If any changes were to occur to any of the data bytes, the check-bits would no longer be correct.

To avoid this, whenever a single byte is to be written to the SDRAM (or for that matter, any number of bytes that is less than the full doubleword), ECC first reads the whole data word, checks for any single- or multi-bit errors, and, if any are present, generates the corresponding interrupt and corrects the data (for a single-bit error), modifies the necessary bytes, and then generates the check-bits across the modified four bytes. Finally, the entire ECC word is stored back into memory. This process is called a *read-modify-write* operation. If a full doubleword is written, then there is no need for a read-modify-write cycle. Also, a partial doubleword write to a write-protected region does not generate a read-modify-write cycle.

Since seven check-bits are required for each bank of SDRAM if ECC is enabled, ECC cannot be supported if 168-pin (72-bit) SDRAM DIMMS are used. If a single 168-pin (72-bit) DIMM is used for supporting two banks, then ECC cannot be enabled due to lack of extra check bits in the DIMM. In this case, extra SDRAM devices must be used to store the check-bits.

To assist in the development of software to handle ECC single-bit and multi-bit errors, the ECC Check Code Test (ECCCKTEST) register (MMCR offset 23h) is provided. This register can be used to override the automatically-generated ECC check code with a user-provided check code for the following SDRAM write access.

10.5.4 Buffering

The ÉlanSC520 microcontroller includes two buffering techniques to optimize the memory system performance. These include the write buffer and read buffer.

When enabled, the write buffer effectively decouples master write activity from incurring the SDRAM latency penalty. This, in effect, also leaves SDRAM free to satisfy a higher demand in read activity by all masters. In addition, the write buffer provides write merge and write collapse functions to better utilize FIFO storage and reduce the number of transactions to SDRAM. The read merge function is also provided to reduce data coherency overhead by eliminating the need to flush the write buffer prior to a read access. During a read request, should the write buffer contain more recent data than SDRAM, the data from the write buffer is merged with data returned from SDRAM, eliminating the need to flush the write buffer.

The ÉlanSC520 microcontroller supports a Read-Around-Write feature when the write buffer is enabled. When the write buffer is enabled, the SDRAM controller’s arbiter favors read activity, effectively giving read priority to SDRAM over write data that has been posted in the write buffer. This feature is intended to increase master read performance.

The read buffer provides two cache lines (32 bytes total) of storage for read data returned from SDRAM. Read requests that can be retrieved from the read buffer can be provided in zero wait states to the requesting master. The SDRAM controller always fetches an entire cache line of data from the SDRAM and stores it in the read buffer, independently of the amount of data requested during the master access. For example, during a read request from a non-bursting master (i.e., single doubleword request), the SDRAM controller fetches the entire cache line of data from SDRAM and stores it in the read buffer.

The read buffer’s read-ahead function, when enabled, provides a mechanism to prefetch the cache line of information from SDRAM that immediately follows the requested cache line. This is in anticipation of future accesses to the prefetched line. The read-ahead feature of the read buffer enhances read burst activity by the Am5$_X$86 CPU and external PCI master burst read requests. Read prefetches, when enabled, occur only for read burst transfer requests of two or more doublewords. Single doubleword read requests do not cause a read-ahead buffer prefetch of the next cache line; they only cause the cache line of the demanded access to be read into the read buffer. GP-DMA read accesses are *always* single word accesses.

The read buffer is always enabled, however, the read-ahead feature and write buffer can be independently enabled and are disabled after a system reset or programmable reset. For more information on the SDRAM controller's buffering, see Chapter 11, "Write Buffer and Read Buffer".

10.5.5 SDRAM Control Configuration

The SDRAM controller provides the following control functions:

- Refresh rate
- Refresh enable
- SDRAM pin drive strength
- Write buffer test mode
- Operation mode select

10.5.5.1 Refresh Control

To refresh the SDRAM devices, the SDRAM controller issues the Auto Refresh command. Since the ÉlanSC520 microcontroller is intended to support a variety of vendors, the refresh rate at which this command is issued is a configurable parameter. It is specified in the DRAM Refresh Request Speed (RFSH_SPD) bit field in the SDRAM Control (DRCCTL) register (MMCR offset 10h) and offers either 7.8-µs, 15.6-µs, 31.2-µs or 62.5-µs periods.

Note: *Since the minimum refresh rate is 62.5 µs, which is below the maximum time between an Active command and a Precharge command (T_{RAS}), the SDRAM controller does not support a $\overline{\text{RAS}}$ time-out feature.*

The refresh rate is calculated from this equation:

Refresh Rate = Interval / Row

where:

Interval is how often a particular row must be refreshed
Row is the number of rows within the device that must be refreshed

Table 10-11 shows the SDRAM refresh rates and their corresponding intervals. SDRAM devices contain either two or four internal banks. During each refresh cycle, all internal SDRAM banks are refreshed simultaneously. This implies that a 2-bank architecture performs dual-row refresh and a 4-bank architecture performs a quad-row refresh, per refresh cycle.

Table 10-11 SDRAM Refresh Rates

Number of Rows	Refresh Rate			
	7.8 µs	**15.6 µs**	**31.2 µs**	**62.5 µs**
256	2 ms	4 ms	8 ms	16 ms
512	4 ms	8 ms	16 ms	32 ms
1024	8 ms	16 ms	32 ms	64 ms
2048	16 ms	32 ms	64 ms	128 ms
4096	32 ms	64 ms	128 ms	
8192	64 ms	128 ms		

For example, if an SDRAM device is organized as 2M x 8 x 4 banks (8Mb x 8) with 4096 rows and 512 columns and requires a 64-ms refresh interval, by using Table 10-11, the refresh rate is 15.6 μs.

During an SDRAM refresh period, all enabled banks are issued an Auto Refresh command. However, during a refresh cycle, SDRAM devices require a somewhat large amount of current, which could become quite large when considering a simultaneous refresh of multiple banks within the same clock period. To prevent this, the SDRAM controller staggers the bank refresh by selecting one bank at a time. This results in only one bank being issued an Auto Refresh command during any given clock, rather than all banks within the same clock. This method results in a slightly larger amount of overhead associated with refresh cycles, but prevents large current surges to the SDRAM banks on the system circuit board. Figure 10-12 on page 10-27 shows an SDRAM staggered refresh cycle.

SDRAM refresh cycles must be enabled only when the SDRAM Operation Mode Select specifier is in normal SDRAM mode. The Refresh Enable (RFSH_ENB) bit is located in the SDRAM Control (DRCCTL) register (MMCR offset 10h).

10.5.5.2 Drive-Strength Selection

The ÉlanSC520 microcontroller provides selectable drive strength options on all address, data and control signals to provide support for different SDRAM device loads presented by different system designs.

Pins with selectable drive strength options include:

- MA12–MA0 (memory address)
- BA1–BA0 (bank address)
- MD31–MD0 (memory data)
- MECC6–MECC0 (ECC data)
- $\overline{\text{SCS3}}$–$\overline{\text{SCS0}}$
- SDQM3–SDQM0
- $\overline{\text{SCASA}}$–$\overline{\text{SCASB}}$
- $\overline{\text{SRASA}}$–$\overline{\text{SRASB}}$
- $\overline{\text{SWEA}}$–$\overline{\text{SWEB}}$

With the exception of $\overline{\text{SCS3}}$–$\overline{\text{SCS0}}$, these pins are equipped to drive 12 mA, 18 mA or 24 mA of current. $\overline{\text{SCS3}}$–$\overline{\text{SCS0}}$ drive either 18 mA or 12 mA.

The SDRAM interface drive strength can be changed in the Drive Strength Control (DSCTL) register (MMCR offset C28h), which is described in the Programmable I/O section of the *Élan™SC520 Microcontroller Register Set Manual*, order #22005.

10.5.5.3 Write Buffer Test Mode

The ÉlanSC520 microcontroller supports a write buffer test mode, using the alternate function of the $\overline{\text{CF\_ROM\_GPCS}}$, DATASTRB, and $\overline{\text{CF\_DRAM}}$ pins that provide master contribution information. As WBMSTR2–WBMSTR0, these three pins indicate whether the Am5$_x$86 CPU, PCI bus master, GP-DMA, or a combination of these (because the write buffer may collapse or merge write data) has contributed into the rank of the write buffer currently in the process of being written to SDRAM. This option is specified with the WB_TST_ENB bit in the SDRAM Control (DRCCTL) register (MMCR offset 10h). See Chapter 24, "System Test and Debugging", for more information on the uses of these pins.

10.5.5.4 Operation Mode Select

The ÉlanSC520 microcontroller provides an SDRAM Operation Mode Select (OPMODE_SEL) bit field in the SDRAM Control (DRCCTL) register (MMCR offset 10h). These bits are used to select a particular mode of operation of the SDRAM controller.

- The default mode of operation is normal SDRAM mode. This is the mode in which the SDRAM controller must be configured for data access.
- The NOP, All Banks Precharge, Load Mode Register, and Auto Refresh commands specified by the OPMODE_SEL bit field are primarily used for SDRAM device initialization.

When specifying NOP, All Banks Precharge, Load Mode Register, or Auto Refresh commands, the command is not actually applied to the SDRAM devices until an Am5$_x$86 CPU access to SDRAM occurs (either a read or write cycle).

The write buffer must be disabled prior to utilizing the NOP, All Banks Precharge, Load Mode Register, or Auto Refresh OPMODE_SEL bit field if the Am5$_x$86 CPU cycle executed to generate these cycle types to the SDRAM devices is a write cycle.

The All Banks Precharge command should be issued prior to bank configuration changes. This places the SDRAM devices in an idle state and clears the SDRAM controller's page table entries.

See "SDRAM Device Initialization" on page 10-30 for more information.

10.5.6 SDRAM Timing Configuration

The ÉlanSC520 microcontroller provides independent timing configuration for SDRAM devices. The following timing parameters are configurable:

- $\overline{\text{CAS}}$ latency (C_L)
- $\overline{\text{RAS}}$ precharge (T_{RP})
- $\overline{\text{RAS}}$-to-$\overline{\text{CAS}}$ delay (T_{RCD})
- $\overline{\text{RAS}}$-to-$\overline{\text{RAS}}$ or auto-refresh-to-$\overline{\text{RAS}}$ (T_{RC})

Note that the write recovery time (T_{WR}) parameter is fixed to 2T (where T refers to a 15-ns clock period for a 33.333-MHz crystal).

10.5.6.1 $\overline{\text{CAS}}$ Latency (C_L)

The $\overline{\text{CAS}}$ latency (C_L) of an SDRAM device specifies the number of clocks between a read command being issued until the *first* piece of read data is available. After this delay, read data is returned on each subsequent clock.

The ÉlanSC520 microcontroller supports $\overline{\text{CAS}}$ latency options for either 2T or 3T (where T refers to a 15-ns clock period for a 33.333-MHz crystal). This parameter is a configuration option, since some SDRAM devices have slightly better access timing when configured for C_L = 3. The CAS_LAT bit in the SDRAM Timing Control (DRCTMCTL) register (MMCR offset 12h) is used to specify this value.

The C_L parameter is programmed into the device with the Load Mode Register command. See "SDRAM Device Initialization" on page 10-30 for more information.

10.5.6.2 $\overline{\text{RAS}}$ Precharge (T_{RP})

The $\overline{\text{RAS}}$ Precharge (T_{RP}) parameter of an SDRAM device refers to the minimum period of time that must be met following a Precharge command until a subsequent command to the same bank can be issued. After T_{RP} is met, the SDRAM device is considered to be in the idle state. T_{RP} varies between device vendors and device speed grades. Even though the ÉlanSC520 microcontroller provides a 66-MHz SDRAM device clock, faster devices are supported (83-MHz, 100-MHz, 125-MHz, etc.).

Since the ÉlanSC520 microcontroller is intended to support a variety of vendors and speed grades, T_{RP} is a configurable parameter and offers either 2T, 3T, 4T or 6T timing (where T refers to a 15-ns clock period for a 33.333-MHz crystal). It is specified in the RAS_PCHG_DLY bit field of the SDRAM Timing Control (DRCTMCTL) register (MMCR offset 12h).

10.5.6.3 $\overline{\text{RAS}}$-to-$\overline{\text{CAS}}$ Delay (T_{RCD})

The $\overline{\text{RAS}}$-to-$\overline{\text{CAS}}$ delay parameter of an SDRAM device refers to the minimum period of time between the time an Active command is issued to the time a read or write command may be issued. This is referred to the T_{RCD} parameter.

Since the ÉlanSC520 microcontroller is intended to support a variety of vendors and speed grades, the T_{RCD} parameter can be programmed for either 2T, 3T, or 4T timing (where T refers to a 15-ns clock period for a 33.333-MHz crystal). Most current SDRAM devices expect a minimum T_{RCD} of 30 ns (or greater), which may be violated with a 2T setting under heavy loading. This parameter is specified in the RAS_CAS_DLY bit field of the SDRAM Timing Control (DRCTMCTL) register (MMCR offset 12h).

10.5.6.4 $\overline{\text{RAS}}$-to-$\overline{\text{RAS}}$ or Auto-Refresh-to-$\overline{\text{RAS}}$ (T_{RC})

The $\overline{\text{RAS}}$-to-$\overline{\text{RAS}}$ or auto-refresh-to-$\overline{\text{RAS}}$ parameter (T_{RC}) of an SDRAM device refers to the minimum period of time between an Active command and another Active command to the *same* internal bank. It also pertains to the minimum amount of time between an Auto Refresh command and an Active command.

The ÉlanSC520 microcontroller does not provide a configuration for the T_{RC} parameter for the timing between an Active command and a following Active command to the *same* internal bank, since this is a function of the T_{RCD} and T_{RP} parameters. Two accesses to different rows of the same internal bank result in an Active command being issued for each access, but the Active command associated with the second access is always preceded by a Precharge Bank command. Because of the preceding Precharge Bank command for the second access, a combination of the T_{RCD} and T_{RP} parameters must provide adequate timing such that the T_{RC} parameter is not violated.

The minimum T_{RC} for an Active command to an Active command is calculated as:

> $T_{RC} = T_{RCD}$ (configuration setting in number of clocks) + T_{RP} (configuration setting in number of clocks) + 2T (where T refers to a 15-ns clock period for a 33.333-MHz crystal).

When a T_{RCD} of 2T is specified, 1T is added to the T_{RC} equation to enforce a minimum T_{RAS} of 5T.

T_{RC} also applies between an Auto Refresh command and an Active command. For this, the ÉlanSC520 microcontroller enforces a *fixed* 9T timing (where T refers to a 15-ns clock period for a 33.333-MHz crystal) following the *last* Auto Refresh command of a staggered refresh sequence.

10.5.6.5 Minimum $\overline{\text{RAS}}$ (T_{RAS})

The minimum $\overline{\text{RAS}}$ parameter of an SDRAM device refers to the minimum period of time that a row must remain open. This is the period of time between an Active command and a Precharge command to the *same* internal bank. This parameter is referred to as T_{RAS}.

Since the ÉlanSC520 microcontroller performs single write cycles, the minimum T_{RAS} occurs during write cycles. T_{RAS} is a function of T_{RCD}. This parameter is calculated as:

> T_{RAS} = T_{RCD} (configuration setting in number of clocks) + 2T (where T refers to a 15-ns clock period for a 33.333-MHz crystal).

A minimum T_{RAS} of 5T is enforced when a T_{RCD} of 2T is specified.

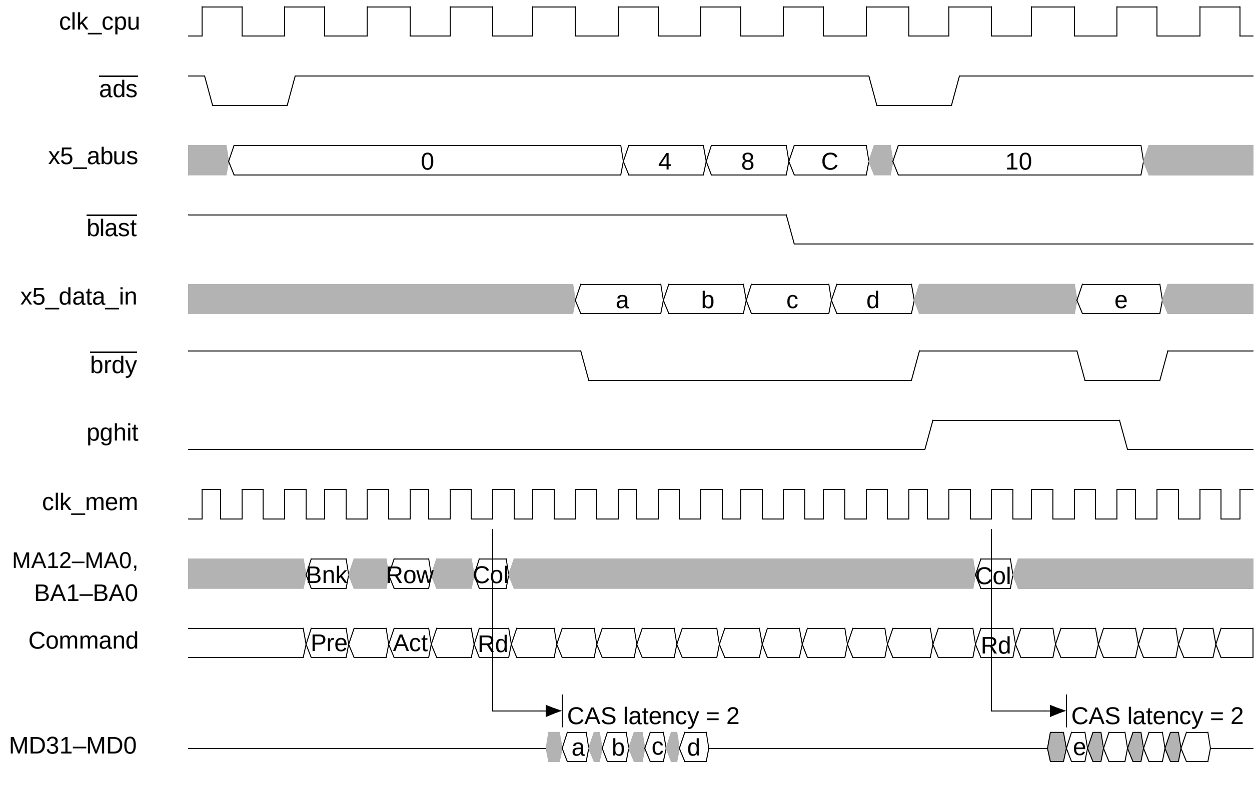
10.5.7 Bus Cycles

10.5.7.1 SDRAM Burst Read Cycle

The ÉlanSC520 microcontroller always bursts up to four doublewords on a read as shown in Figure 10-7. The burst read to the SDRAM could occur due to any of the following reasons:

- $Am5_x86$ CPU read
- Read buffer's read-ahead prefetch
- ÉlanSC520 microcontroller responding to PCI burst cycle as a target
- GP-DMA

Figure 10-7 SDRAM Burst Read Cycle (Read-Ahead Feature Disabled) (Page Miss/Page Hit)



Notes:

This timing diagram does not account for resynchronization of SDRAM signals with CLKMEMIN.

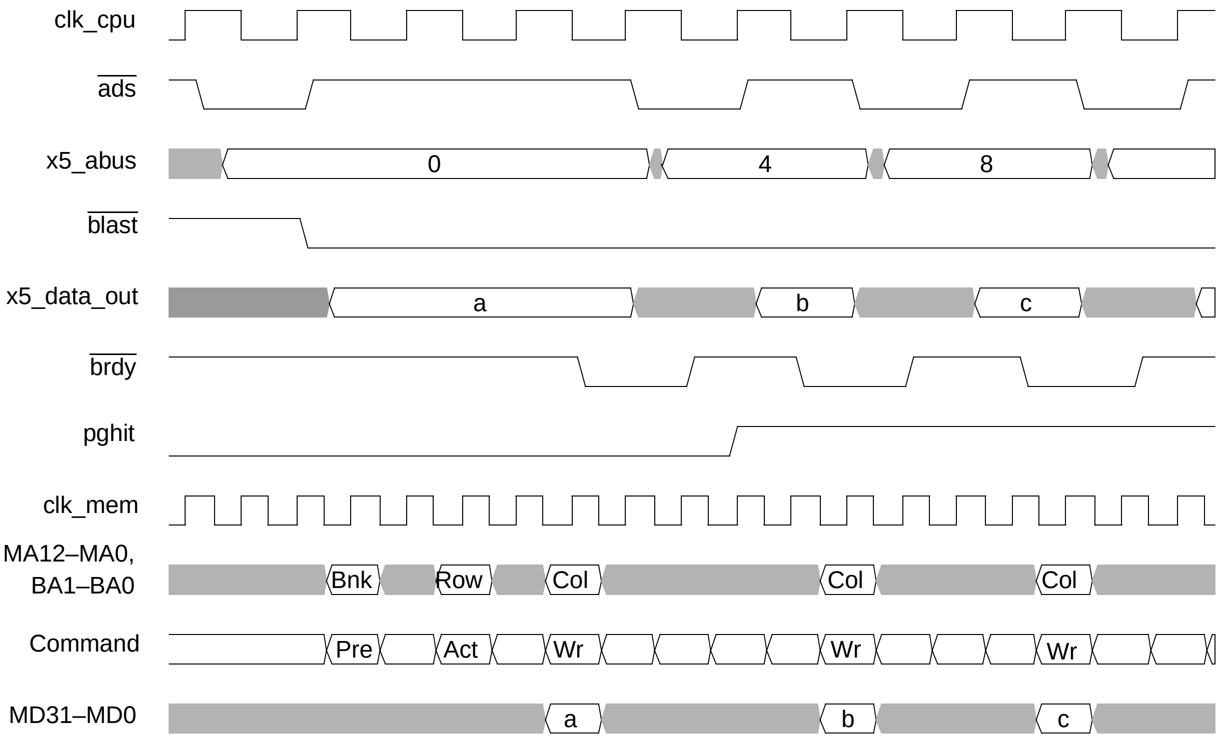
10.5.7.2 SDRAM Write Cycle

With the write buffer enabled, all writes to the SDRAM come from the write buffer. With the write buffer disabled, the SDRAM write cycle could occur due to any of the following reasons:

- Am5$_x$86 CPU
- ÉlanSC520 microcontroller responding to PCI burst cycle as target
- GP-DMA

All the writes are configured for single write mode, with each write occurring independently. Am5$_x$86 CPU non-burst write transfers are shown in Figure 10-8. An Am5$_x$86 CPU burst write cycle is shown in Figure 10-9.

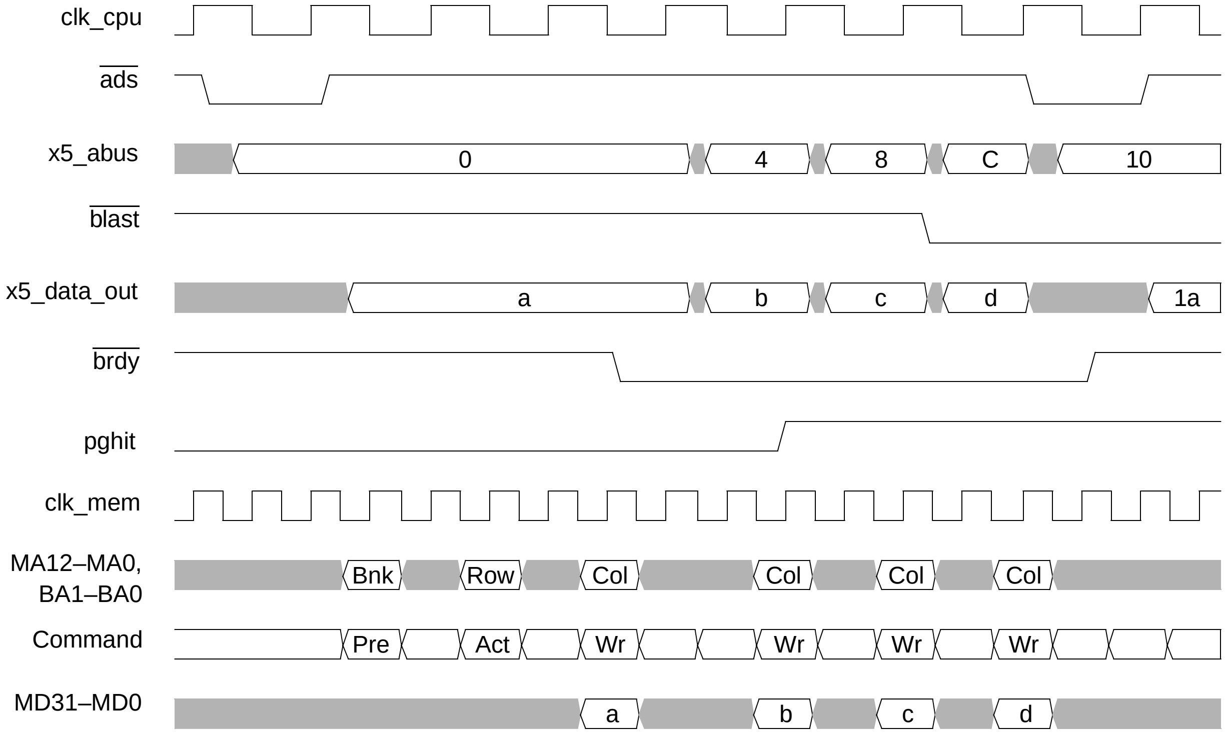
Figure 10-8 SDRAM Write Cycle (Write Buffer and ECC Disabled) (Page Miss/page Hit)



Notes:

This timing diagram does not account for resynchronization of SDRAM signals with CLKMEMIN.

Figure 10-9 SDRAM CPU Burst Write (Write Buffer and ECC Disabled) (Page Miss/Page Hit)



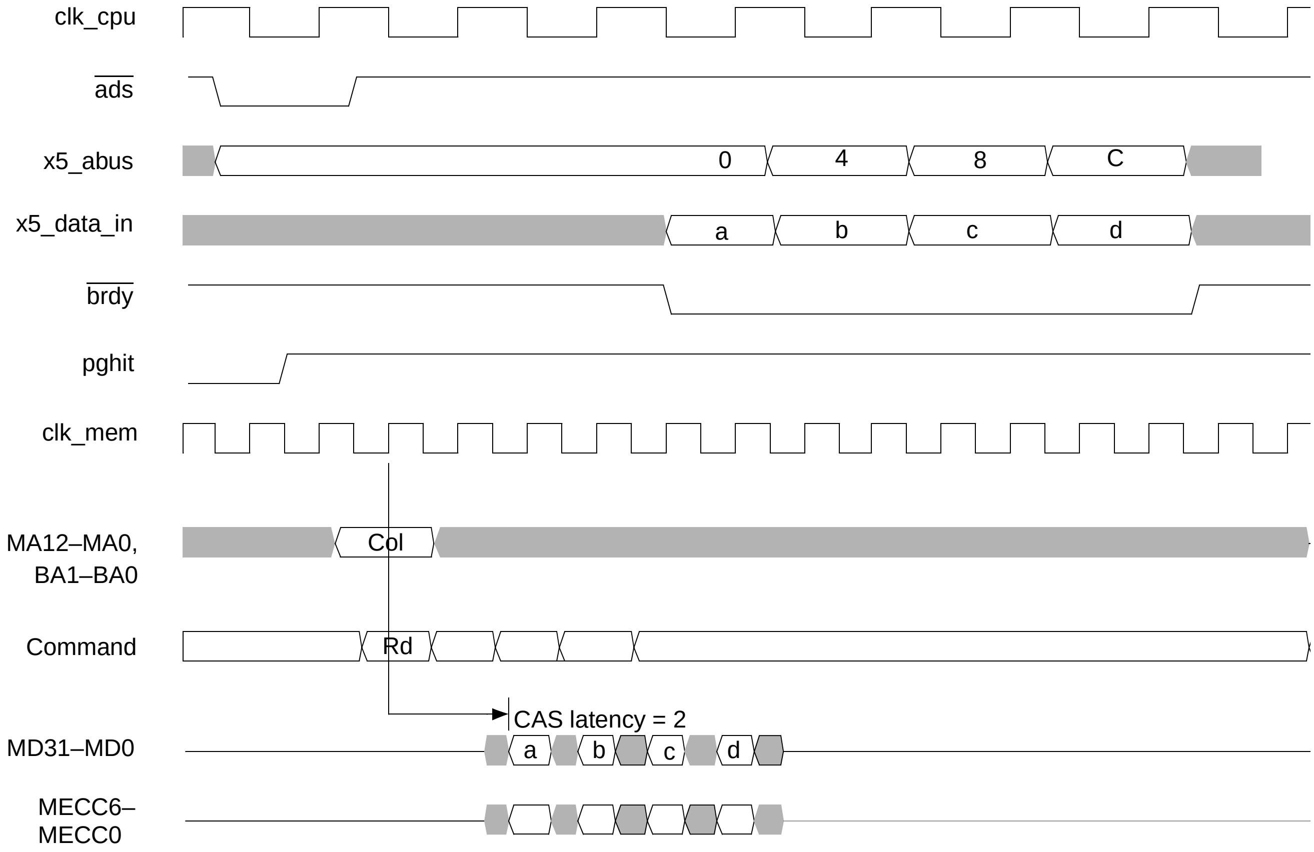
Notes:

This timing diagram does not account for resynchronization of SDRAM signals with CLKMEMIN.

10.5.7.3 ECC SDRAM Cycles

When ECC is enabled, additional overhead is necessary to compensate for ECC logic delays and read-modify-write cycles due to partial doubleword write cycles. The least amount of overhead occurs during a full doubleword write to the SDRAM. In the case of a read, however, the ECC has to generate the new check bits, check for any errors, and generate an interrupt if an error occurs. A delay of one CPU clock cycle is added for SDRAM read cycles with ECC enabled. With ECC enabled, read page hit burst timing of 4-1-1-1 (where C_L = 2) is achieved, compared to a 3-1-1-1 (where C_L = 2) burst with ECC disabled. See Figure 10-10 showing the read cycles with ECC enabled.

Figure 10-10 SDRAM Burst Read Cycle with ECC Enabled

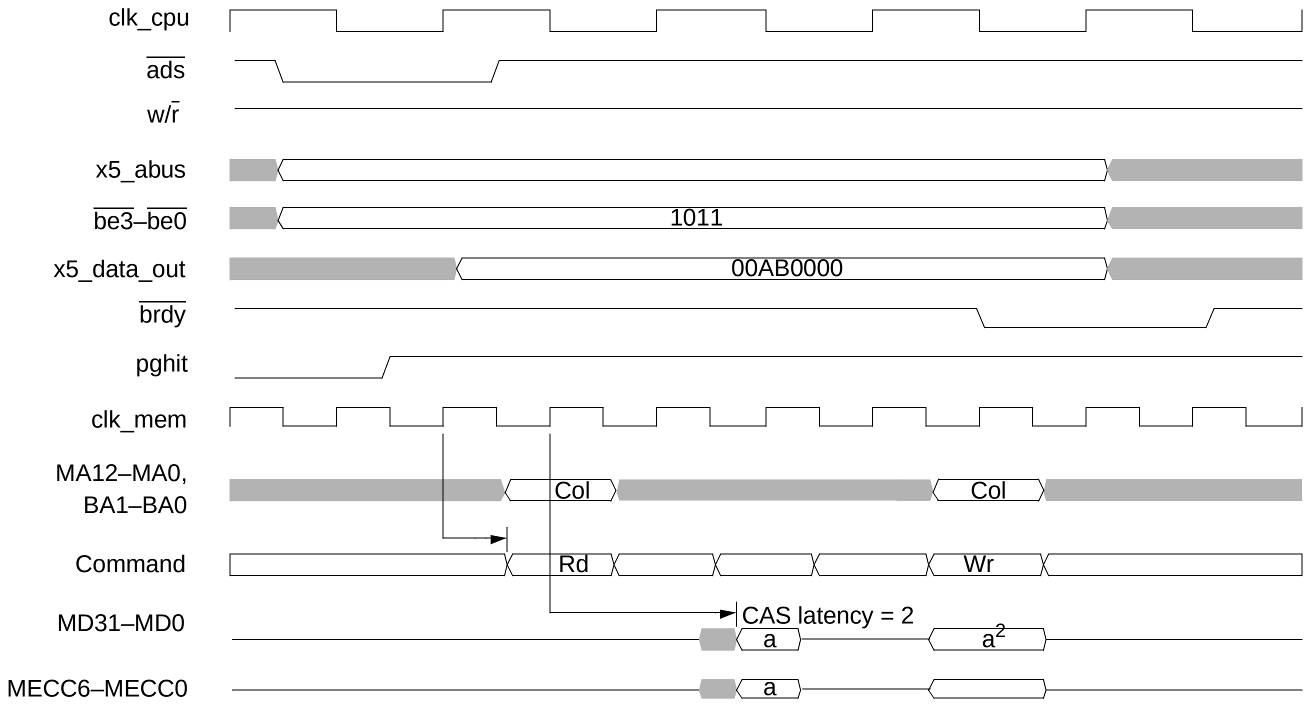


Notes:

This timing diagram does not account for resynchronization of SDRAM signals with CLKMEMIN.

The ECC overhead is even higher in the case of a read-modify-write cycle, as shown in Figure 10-11. As shown, a write cycle with a partial doubleword requires an SDRAM read cycle followed by a write cycle. Note that the SDRAM read burst is terminated early by the write cycle. See “Error Correction Code (ECC)” on page 10-16 for details of a read-modify-write cycle.

Figure 10-11 SDRAM Read-Modify-Write Cycle (for Data Write) with ECC Enabled (Page Hit)[1]



Notes:

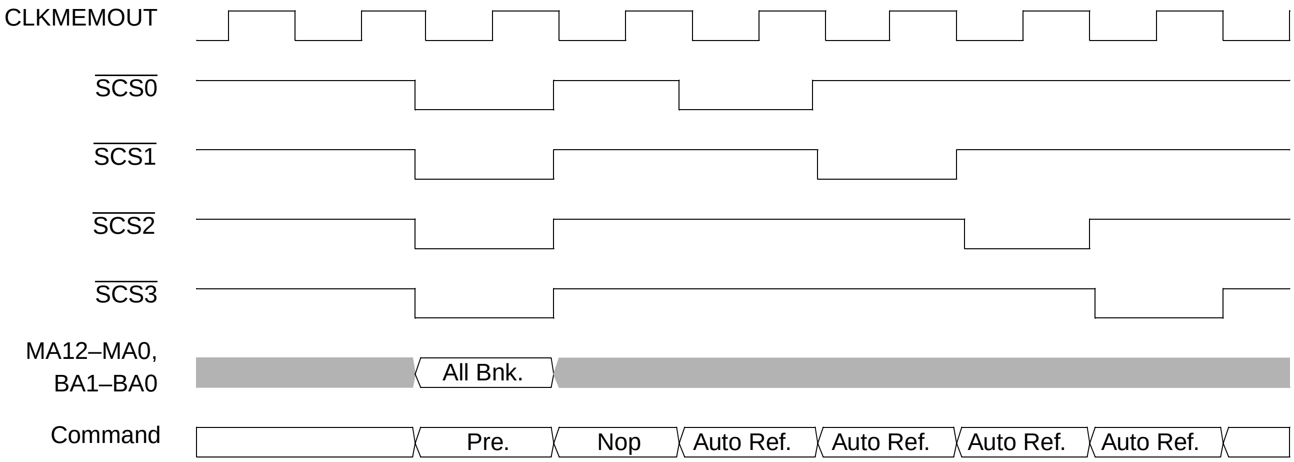
1. *This timing diagram does not account for resynchronization of SDRAM signals with CLKMEMIN.*
2. *Contents modified with the active bytes in the write word (00AB0000).*

10.5.7.4 SDRAM Auto Refresh Cycle

Auto refresh, as shown in Figure 10-12, is used during normal operation of the SDRAM and is analogous to the $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ refresh in EDO DRAMs. This command is nonpersistent, so it must be issued each time a refresh is required. The internal banks will be precharged and idle for a minimum of the Precharge time (T_{RP}) before the Auto Refresh command is applied. When the refresh cycle has completed, all the banks of the SDRAM will be in the precharged (idle) state. Note that this figure shows a staggered refresh cycle, as described in “Refresh Control” on page 10-18.

The purpose of the programmable reset in the memory controller is to maintain the state of the SDRAM during a reset. This allows SDRAM refreshes to occur during reset. See Chapter 6, “Reset Generation”, for more information.

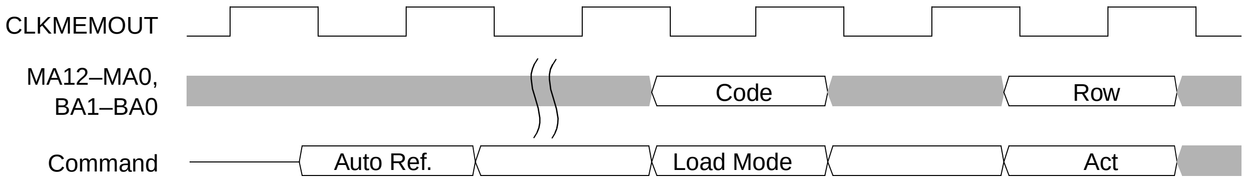
Figure 10-12 SDRAM Auto Refresh Cycle



10.5.7.5 SDRAM Mode Register Access Cycles

The mode register contained in the SDRAM devices is used to define the specific mode of operation of the SDRAM. This definition includes the selection of the burst length, burst type, $\overline{\text{CAS}}$ latency, operating mode, and write burst mode. An SDRAM Load Mode Command is shown in Figure 10-13. See “SDRAM Device Initialization” on page 10-30 for information on programming the mode register.

Figure 10-13 SDRAM Mode Register Access



10.5.8 Interrupts

The SDRAM controller implements Error Correction Code logic to detect and correct single-bit errors and detect multi-bit errors.

Separate interrupts can be generated for both single-bit error and multi-bit error detection. These two interrupts are routed from the SDRAM controller to the ÉlanSC520 microcontroller’s programmable interrupt controller (PIC).

- These two interrupts can be individually enabled by using the MULT_INT_ENB and SGL_INT_ENB bits in the ECC Control (ECCCTL) register (MMCR offset 20h).
- The interrupt signals remain asserted to the PIC until a write is performed to the MBIT_ERR and SBIT_ERR status bits in the ECC Status (ECCSTA) register (MMCR offset 21h). This write is typically performed by the interrupt handler associated with the interrupt.

Note: *The multi-bit error interrupt, when enabled, always generates a non-maskable interrupt (NMI).*

10.5.9 Software Considerations

10.5.9.1 ECC Errors

The ECC logic in the SDRAM controller detects single-bit error and multi-bit errors in the SDRAM data being accessed.

- When a single-bit error is detected, a maskable interrupt is generated. See Chapter 15, “Programmable Interrupt Controller”, for information on steering this interrupt.
- When a multi-bit error is detected, a non-maskable interrupt (NMI) is generated.

The interrupt handler should read the ECC Status (ECCSTA) register (MMCR offset 21h) logging the detection of a single-bit error (SBIT_ERR) or a multi-bit error (MBIT_ERR), depending on which interrupt signal is generated. The physical address where the error occurred is latched for both single-bit and multi-bit errors in the ECC Single-Bit Error Address (ECCSBADD) register (MMCR offset 24h) and ECC Multi-Bit Error Address (ECCMBADD) register (MMCR offset 28h), respectively. An encoded value of the data bit position where the single-bit error occurred is also latched in the ECC_CHK_POS bit field of the ECC Check Bit Position (ECCCKBPOS) register (MMCR offset 22h).

All latched information pertaining to an error is latched on the first occurrence and cleared when the latch is re-enabled. Information for errors that occur *after* the first occurrence, but before the latch is re-enabled, are lost.

10.5.9.2 Buffer Disabling During SDRAM Configuration

Prior to altering the SDRAM configuration, the write buffer and read-ahead feature of the read buffer must be disabled. This is to prevent SDRAM configuration changes while a write buffer or read-ahead prefetch to SDRAM is in progress. During bank configuration, it is important to not enable an SDRAM bank with the Bank Ending Address specified as 0.

10.5.9.3 Write Protection

Regions of SDRAM can be write-protected through the use of a Programmable Attribute Region (PAR) register. A write-protected region allows read cycle access, however, data is not written to the devices during a write cycle access. When writing to a region that is write-protected, an SDRAM write cycle still occurs; however, the SDQM3–SDQM0 data mask signals are active throughout the cycle to prevent the data from being written to the devices. If ECC is enabled and a noncomplete doubleword access is write-protected, the SDRAM controller does not generate a read-modify-write cycle.

10.5.10 Latency

The SDRAM controller’s write buffer and read buffer are designed to enhance the memory system’s bandwidth and performance. When enabled, the write buffer decouples master write or burst write activity from incurring the SDRAM access latency penalty along with the overhead associated with SDRAM refresh cycles. When enabled, the read-ahead feature of the read buffer decouples master read activity from incurring the SDRAM latency penalty on read buffer hits. For more information, see Chapter 11, “Write Buffer and Read Buffer”.

SDRAM devices require periodic refresh cycles to maintain data integrity within the device. This SDRAM activity must occur at fixed intervals as high priority requests. In the event that a data access request and a refresh cycle request occur at the same time, the data access request is stalled until the higher priority refresh cycle is complete. Devices that can tolerate a slower refresh period result in a system with less refresh overhead, leaving SDRAM free for data access requests. To support these devices, the ÉlanSC520 microcontroller provides an adjustable refresh rate of 7.8 µs, 15.6 µs, 31.2 µs or 62.5 µs.

When the write buffer is enabled, writes to SDRAM occur independently of any associated master activity until the write buffer is empty. Since the SDRAM data bus may be shared with the ROM/Flash controller, write-buffer writes may request concurrently with master requests to ROM/Flash. Should these two independent activities concurrently request access to the data bus, the ROM/Flash cycle takes precedence over the write-buffer write in favor of satisfying the requesting master. However, a ROM/Flash cycle may be temporarily delayed should a master request ROM/Flash access during a write-buffer write in progress. Furthermore, a ROM/Flash access that occurs during a read-ahead prefetch results in the ROM/Flash access being temporarily delayed until the read prefetch completes. See Chapter 12, “ROM/Flash Controller”, for information on ROM/Flash sharing the SDRAM data bus.

ECC results in additional latencies due to required read-modify-write cycles. The read-modify-write cycles are necessary when incomplete doublewords are written to the SDRAM devices (i.e., any writes less than four bytes). Read-modify-write is required to update the ECC code to include the information reflected in the partial doubleword to be written. However, a partial doubleword write to a write-protected region does not generate a read-modify-write cycle.

Prior to a write, the following sequence occurs:

1. The complete doubleword and ECC code is read from SDRAM and checked for errors (the respective interrupt is generated if an error is detected)
2. The new ECC code is generated to include the data just read and the new data to be written.
3. The complete modified doubleword and modified ECC code is written back into the SDRAM.

Should the write cycle be a complete doubleword, the ÉlanSC520 microcontroller does not require a read of the SDRAM first. This reduces the overhead associated with 32-bit writes to SDRAM. However, since a read is not performed prior to a doubleword write, the contents in SDRAM are not checked prior to the data being written.

10.6 INITIALIZATION

10.6.1 Programmable Reset

The ÉlanSC520 microcontroller’s SDRAM controller provides the capability to maintain the contents of the SDRAM during a reset event. In effect, two types of reset are supported:

- System reset—A complete reset where the entire SDRAM controller is reset and contents of the SDRAM devices are lost.
- Programmable reset—The SDRAM controller configuration is maintained and the contents of the SDRAM devices are also maintained by maintaining refresh cycles throughout the programmable reset duration.

Selection of the reset type is controlled by the PRG_RST_ENB bit in the Reset Configuration (RESCFG) register (MMCR offset D72h). With this bit, the PRGRESET pin can be programmed to reset the ÉlanSC520 microcontroller for a programmable reset. On power-up, the PRGRESET pin is disabled and must be programmed to be operational.

See “System Reset with SDRAM Retention” on page 6-6 for detailed information on the sources of these resets.

The purpose of the programmable reset in the memory controller is to maintain the state of the SDRAM during an ÉlanSC520 microcontroller reset. This requires SDRAM refreshes to occur throughout the entire duration of the programmable reset. Upon the assertion of the programmable reset, the SDRAM controller arbiter lets the current SDRAM access complete before returning the controller state machines to their idle states. This prevents data corruption in the SDRAM array should the programmable reset be asserted *during* an access to SDRAM. All SDRAM controller configuration is maintained.

Note: *The contents of the write buffer are discarded for both types of reset. Also, the enable states of the write buffer and read buffer are not maintained after a programmable reset. Therefore, if the write buffer and read buffer were enabled prior to the programmable reset, software must re-enable them after the programmable reset.*

10.6.2 SDRAM Device Initialization

Section 10.6.2– Section 10.6.4 provide details on enabling the core and SDRAM configuration. However, prior to altering the SDRAM configuration, the write buffer and read-ahead feature of the read buffer must be disabled. This is to prevent SDRAM configuration changes while a write buffer or read-ahead prefetch to SDRAM is in progress.

Refresh should be disabled anytime the SDRAM controller is not operating in normal SDRAM mode. SDRAM refresh cycles should only be enabled when the OPMODE_SEL bit field is configured for normal SDRAM mode. After the SDRAM devices are initialized (with refresh cycles remaining disabled), they can be reliably accessed.

If the Error Correction Code (ECC) logic for SDRAM is enabled, the ECC operation requires that SDRAM and its associated ECC memory be initialized. This is accomplished by the boot code that must write to every location in SDRAM. This process initializes the ECC SDRAM to reflect the proper Hamming code for its associated data. If this procedure is not performed, false errors will occur when reading or when writing data smaller than a 32-bit doubleword. See “Error Correction Code (ECC)” on page 10-16 for a more detailed discussion of ECC.

10.6.2.1 Operation Mode Select

SDRAM devices must be powered up and initialized in a predefined manner prior to access. The SDRAM controller’s SDRAM Control (DRCCTL) register (MMCR offset 10h) provides support for this procedure via the OPMODE_SEL field.

- By default, the OPMODE_SEL bit field reflects a normal SDRAM mode of operation. However, a *normal* SDRAM mode of operation refers to the mode the SDRAM controller must be configured in *after* SDRAM device initialization is complete. Normal SDRAM mode allows read and write accesses to occur as requested by a master. SDRAM refresh cycles should be enabled only when the OPMODE_SEL field is configured for normal SDRAM mode.
- The other settings for the OPMODE_SEL field force all SDRAM accesses to a specific SDRAM command type: NOP, Precharge, Load Command, or Refresh. Setting the OPMODE_SEL bits to *non*-normal SDRAM mode results in all banks being selected (i.e., $\overline{\text{SCS3}}$–$\overline{\text{SCS0}}$ are driven active), so that the command is applied to all SDRAM devices in the system.

To generate the command specified in the OPMODE_SEL field, an Am5$_X$86 CPU read or write cycle must be generated to the SDRAM region. The specified command occurs at the SDRAM interface rather than the actual read or write cycle requested by the Am5$_X$86 CPU.

10.6.2.2 NOP Command

Once power is applied and the clock is stable, most SDRAM devices require a 100-µs delay prior to applying an executable command. Therefore, boot code must guarantee that SDRAM is not accessed immediately after reset. During this period and continuing at least through the end of this period, the NOP command should be applied. During initialization, the NOP command is enabled, with a binary pattern of 001b being written to the Operation Mode Select bits. An Am5$_x$86 CPU read or write cycle must be generated to the SDRAM region to cause the generation of the specified command.

10.6.2.3 Precharge Command

Once the 100-µs delay has been satisfied with at least one NOP command having been applied, a Precharge command should be applied to all the internal banks within a device, thereby placing the device in the idle state. The All Banks Precharge command can be enabled during initialization, with a binary pattern of 010b being written to the Operation Mode Select bits. In this mode, MA10 (precharge) is held high during the precharge to enable the All Banks Precharge. Since all banks are selected, all banks will be enabled to interpret this command.

10.6.2.4 Auto Refresh Command

Once in the idle state, two Auto Refresh cycles must be performed. The Auto Refresh command can be enabled during initialization, with a binary pattern of 100b being written to the Operation Mode Select bits. The boot code must perform at least two accesses to SDRAM when in this mode.

10.6.2.5 Mode Register Programming

Once the Auto Refresh cycles are complete, the SDRAM is ready for mode register programming. The Load Mode Register command can be enabled during initialization with a binary pattern of 011b being written to the OPMODE_SEL field. Since all SDRAM banks are selected (i.e., $\overline{\text{SCS3}}$–$\overline{\text{SCS0}}$ are driven active), all banks will be configured to the same mode. The mode register is programmed to define the SDRAM devices burst length, burst type, $\overline{\text{CAS}}$ latency, operating mode, and write burst mode.

Of these five parameters, only the $\overline{\text{CAS}}$ latency parameter is configured by the user via the CAS_LAT bit in the SDRAM Timing Control (DRCTMCTL) register (MMCR offset 12h). The programmable options for $\overline{\text{CAS}}$ latency are 2T or 3T, where T = 15-ns clock period for a 33.333-MHz crystal. The other parameters are fixed by the ÉlanSC520 microcontroller.

Table 10-12 shows the parameters and their associated settings. All bits reflecting these configurations are driven on the MA12–MA0 signals during a Load Mode Register command. Since SDRAM devices require only 12 bits for the command width, MA12 is driven Low during this cycle.

Table 10-12 Load Mode Register Settings

Parameter	Setting	Description
Burst length	Four phases	Always read burst four
Burst type	Interleaved	Follow non-linear burst
$\overline{\text{CAS}}$ latency	Programmable	Select either 2T or 3T (see text)
Operating mode	Standard operation	Defined
Write burst mode	Single location	Single mode

10.6.3 Boot Process

In a closed embedded system, the designer may be able to simply choose the correct values to output to the configuration registers. Systems where the SDRAM parameters are not known at boot time present more issues. Many SDRAM considerations, such as signal loading, cannot be accurately determined by software. One way to deal with this issue is to have a staged boot process, as follows:

1. First, all timing registers are programmed to assume a worst-case system by default after reset.
2. Next, the SDRAM banks are tested for SDRAM existence, organization, and size. Banks that contain SDRAM are enabled with the correct parameters.
3. A system memory test is then performed to ensure that there are no problems. The user can be notified, and bad banks can be disabled, if any problems are encountered.

Since the user has control over SDRAM setup parameters, they must not be applied to the SDRAM array until late in the boot process, so that the setup program can always be used to recover the system if it becomes unbootable.

10.6.4 SDRAM Sizing Algorithm

The SDRAM sizing algorithm must alter the SDRAM configuration registers and write and read specific boundary SDRAM locations to determine where the SDRAM bank boundary exists. Data that is written and then returned on a read implies that valid SDRAM exists at that location.

However, prior to accessing the SDRAM devices, the mode register for the device must be programmed to configure the devices before they are functional. SDRAM device initialization is discussed in more detail in Section 10.6.2. Note that SDRAM refresh cycles should only be enabled when the OPMODE_SEL bit field is configured for normal SDRAM mode. After the SDRAM devices are initialized (with refresh cycles remaining disabled), they can be reliably accessed.

The SDRAM controller provides many configuration registers with control and timing configuration functions. However, only a subset of these registers is required to be accessed during the sizing procedure. In particular, the bits associated with specifying the column address width, the internal bank count specifier, and the bank ending address are the most critical for the sizing process.

- The column address width is used to specify the column width of the device.
- The internal bank count bit specifies if the device supports either two or four internal banks.
- The SDRAM Bank 0–3 Ending Address (DRCBENDADR) register (MMCR offset 18h) is used to specify the physical address bank boundary.

The column boundary method is used to accept a wide variety of SDRAM devices and symmetries. In configuring the symmetry of the device, this method requires only the column address width to be specified. Device addressing and symmetries are discussed in “SDRAM Addressing” on page 10-12.

It is important to point out that whenever the column address width, internal bank count, or bank ending address configuration is going to be changed, the All Banks Precharge command must be issued prior to the configuration update. The All Banks Precharge command can be enabled with a binary pattern of 010b being written to the OPMODE_SEL bit field. A cycle to SDRAM must be run for the command to take effect. The All Banks

Precharge command closes all open pages in the SDRAM devices, thus placing them in an idle state. This also forces the SDRAM controller's page table entries to be invalidated.

The column address requirement of the device specifies its symmetry (i.e., its usable number of columns, or page width, that the SDRAM controller can utilize), but does not specify the amount of addressable SDRAM in the 32-bit bank. The bank ending address is used to specify the physical address boundary of each bank. The bank ending address is independent of device density or device data width. During SDRAM sizing, a bank should never be enabled with a bank ending address of 0. The internal bank count specifier is used to inform the SDRAM controller of the internal bank architecture of the device, since SDRAM devices can contain either two or four internal banks.

- To dynamically determine the amount of SDRAM memory in the entire system, the sizing algorithm must first determine the amount of SDRAM installed per each external bank.
  - To do this, the algorithm must enable one external bank at a time and start with the largest possible configuration for that bank, which is 11 columns, 4 internal banks, and 13 rows.
  - If a smaller-sized SDRAM is installed in a given external bank, aliases will be created, and the sizing algorithm uses the aliasing to determine the actual size of the external SDRAM bank.
- Note that while SDRAM sizing is being performed, the Am5$_x$86 CPU cache, the SDRAM ECC, the SDRAM write buffer, and the SDRAM read-ahead feature should all be disabled.

For example, to setup external SDRAM Bank 3 to its largest possible SDRAM configuration setting, a value of A000h should be written into the SDRAM Bank Configuration (DRCCFG) register (MMCR offset 14h), and a value of FF000000h should be written into the SDRAM Bank 0–3 Ending Address (DRCBENDADR) register (MMCR offset 18h).

10.6.4.1 Determining the Number of Columns for an External Bank

Determining the correct number of columns for a given external bank of SDRAM can be accomplished by four writes and five reads of a given external bank.

Four unique data patterns must be selected.

An example is:

```
pattern1 = 0Bh
pattern2 = 0Ah
pattern3 = 09h
pattern4 = 08h
```

Four SDRAM memory addresses must be selected that all have the same internal bank and SDRAM row address bits (processor address bits 31–13 constant) and the same low order column address bits (processor address bits 9–0 constant), but with specially selected column addresses for processor address bits 12–10.

- The first address must have SDRAM column address bits 11, 9, and 8 (processor address bits 12–10) on.
- The second address must have SDRAM column address bit 11 (processor address bit 12) off and SDRAM column address bits 9–8 (processor address bits 11–10) on.
- The third address must have SDRAM column address bits 11 and 9 (processor address bits 12-11) off and SDRAM column address bit 8 (processor address bit 10) on.

- The final address must have SDRAM column address bits 11, 9, and 8 (processor address bits 12–10) off.

There are many addresses which meet this criteria, of which one example is:

address1 = 0E001E00h
address2 = 0E000E00h
address3 = 0E000600h
address4 = 0E000200h

Here is the sequence to determine the number of columns for a given external bank of SDRAM:

1. First, pattern1 is written and read back from address1.
2. Pattern2 is written and read back from address2.
3. Pattern3 is written and read back from address3.
4. Pattern4 is written and read back from address4.
5. If any of the four reads fail to produce the same pattern that was written, then either SDRAM does not exist for this external bank, or the SDRAM is nonfunctional, which, in either case, no memory is enabled and sizing continues with the next external bank.
6. If all four reads are correct, then address1 is read once again, and the pattern that is returned by this read determines the true number of columns.

Using the patterns given in the example, the value read *is* the number of real columns for the external bank.

10.6.4.2 Determining the Number of Internal Banks

Determining the correct number of internal banks and the true ending address of an external bank requires only five writes and seven reads of the external bank.

Five unique data patterns must be selected.

An example is:

pattern5 = 3Fh
pattern6 = 1Fh
pattern7 = 0Fh
pattern8 = 07h
pattern9 = AAh

Five SDRAM memory addresses must be selected which all have the same low-order SDRAM row address bits, the same least significant internal bank select bit (BA0), and the same SDRAM column address bits (processor address bits 31–28 and 23–0 constant), but with specially selected row addresses for processor address bits 27–24. Processor address bits 27–24 is where the SDRAM rows above ROW10 are mapped in this maximum SDRAM configuration.

- The first address must have processor address bits 27–24 all on.
- The second address must have processor address bit 27 off and processor address bits 26–24 on.
- The third address must have processor address bits 27–26 off and processor address bits 25-24 on.
- The fourth address must have processor address bits 27–25 off and processor address bit 24 on.

- The final address must have processor address bits 27–24 all off.

There are many addresses which meet this criteria, of which one example is:

address5 = 0F000000h
address6 = 07000000h
address7 = 03000000h
address8 = 01000000h
address9 = 00000000h

Here is the sequence to determine the correct number of internal banks:

1. First, pattern5 is written and read back from address5.
2. Pattern6 is written and read back from address6.
3. Pattern7 is written and read back from address7.
4. Pattern8 is written and read back from address8.
5. Pattern9 is written and read back from address9.
6. If any of these five reads fail to produce the same pattern that was written, then either SDRAM does not exist for this external bank, or the SDRAM is nonfunctional, which in either case no memory is enabled and sizing continues with the next external bank.
7. If all five reads are correct, then the correct number of internal banks can be determined by reading address7 once again.
8. If the pattern read from address7 is pattern9, then only two internal banks exist for this external bank.
9. If the pattern read from address7 is pattern7 or pattern8, then four internal banks exist.
10. If the pattern read from address7 is anything other than pattern7, pattern8, or pattern9, then there is no valid memory for this external bank.

The reason pattern7 is read back from a 2-internal-bank SDRAM is because the SDRAM controller thinks it has two open pages, and the SDRAM has only one open page, so the data is retrieved erroneously from the wrong page.

10.6.4.3 Determining the True External Bank Ending Address

The true ending address can now be determined by reading adress5 again. If any value other than pattern5, pattern6, pattern7, or pattern8 is read, then there is no valid memory for this external bank.

Here is the sequence to determine the true external bank ending address:

1. Using the values for these patterns as in the example, the value read represents the ending address for the external bank, if the device has 11 columns.
2. So, this value must be shifted right by the value 11, minus the actual number of columns determined to exist.
3. This value must then be incremented by 1 and ORed with 80h to be ready to be loaded into the appropriate byte of the SDRAM Bank 0–3 Ending Address (DRCBENDADR) register (MMCR offset 18h).

This process is continued until all four possible external banks have been checked.

CHAPTER

11 WRITE BUFFER AND READ BUFFER

11.1 OVERVIEW

The ÉlanSC520 microcontroller includes two buffering techniques to optimize the SDRAM system performance. These include a write buffer and a read buffer with a read-ahead feature.

The write buffer provides a mechanism for all masters (Am5$_X$86 CPU, PCI, or GP-DMA) to post write data with zero wait states. When enabled, the write buffer effectively decouples master write activity from incurring the SDRAM latency penalty. This, in effect, also allows SDRAM to satisfy a higher demand in read activity by all masters. In addition, the write buffer provides write-merging and write-collapsing functions to better utilize FIFO storage and reduce the total number of transactions to SDRAM. Data read-merging is also supported to efficiently maintain data coherency.

The read buffer provides two cache lines (32 bytes total) of storage for read data returned from SDRAM. The read buffer and its associated read-ahead function, when enabled, provide a mechanism to prefetch the cache line of information from SDRAM that immediately follows the requested cache line. This feature is provided in anticipation of future accesses to the prefetched line (spatial locality). The read buffer is always enabled; however, the read-ahead feature and write buffer are disabled after a system reset.

Although both the write buffer and read-ahead feature of the read buffer are tightly integrated, they can be independently enabled.

Features of the write buffer include:

- 32-level doubleword FIFO with random access capability
- Content addressable memory (CAM) provides snoop capability
- Zero wait state writes to non-full buffer
- Provides write-merging, write-collapsing, and read-merging functions
- Benefits Am5$_X$86 CPU, PCI, and GP-DMA SDRAM write transfers

Features of the read buffer include:

- Read buffer provides storage for two Am5$_X$86 CPU cache lines (32 bytes total)
- Zero wait state reads on read buffer hits
- Read-ahead feature that, when enabled, prefetches the next cache line of information from SDRAM for master read requests of two or more doublewords
- Demand doubleword start fetch
- Benefits Am5$_X$86 CPU, PCI, and GP-DMA SDRAM read transfers

The write buffer is expected to enhance individual write or burst write activity by all masters. It supplies zero wait state writes for all masters. However, the write buffer's write-merging and write-collapsing features greatly enhance Am5$_X$86 CPU, PCI, and GP-DMA 8-bit and

16-bit contiguous transfers, allowing multiple individual transfers to be merged into a single transaction to SDRAM.

The read-ahead feature of the read buffer enhances read burst activity by the Am5$_x$86 CPU and external PCI master burst read requests. SDRAM cache line fills by the Am5$_x$86 CPU are probably the most common read requests. These reads typically occur as cache-line bursts of four doubleword (32-bit) requests. PCI master burst read requests also benefit greatly.

Each feature can be independently configured. To maintain data coherency, the read buffer is invalidated during master write cycles or write buffer write cycles that hit an existing line in the read buffer. Data coherency during all configuration changes of the individual features is performed in hardware. A manual flush feature of the write buffer is provided.

11.2 BLOCK DIAGRAM

The write buffer and read buffer are integrated into the SDRAM controller's subsystem as shown in Figure 11-1. Each is capable of functioning independently. A more detailed view of the internal write buffer and read buffer architecture is shown in Figure 11-2.

Figure 11-1 Write Buffer and Read Buffer Block Diagram (SDRAM Subsystem)

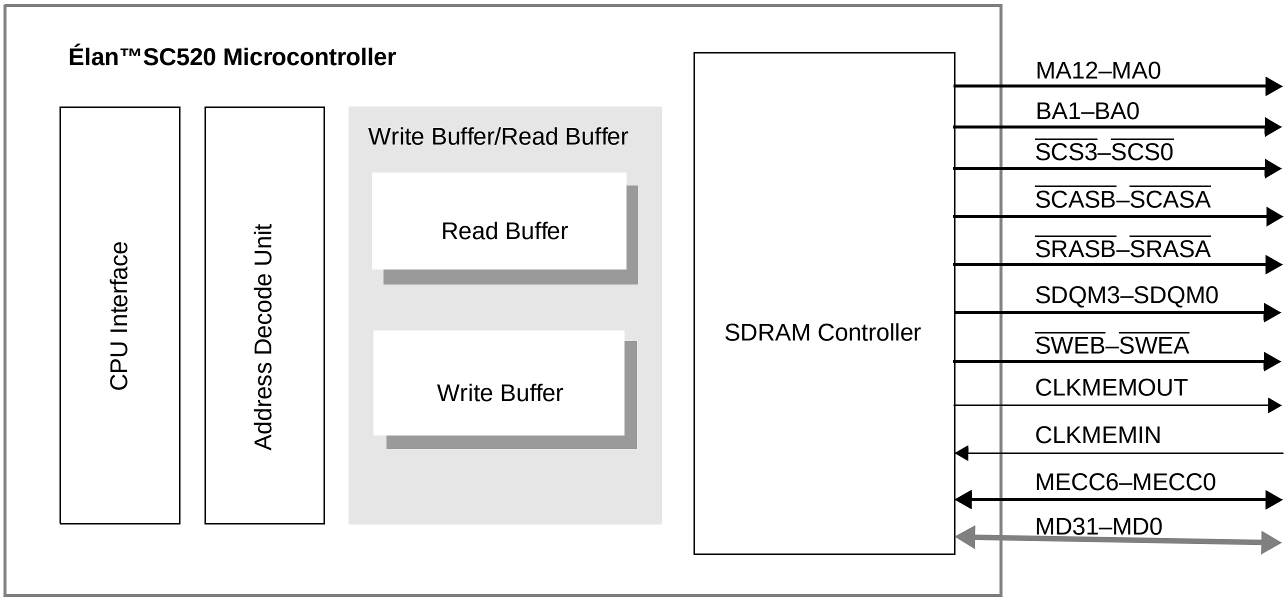
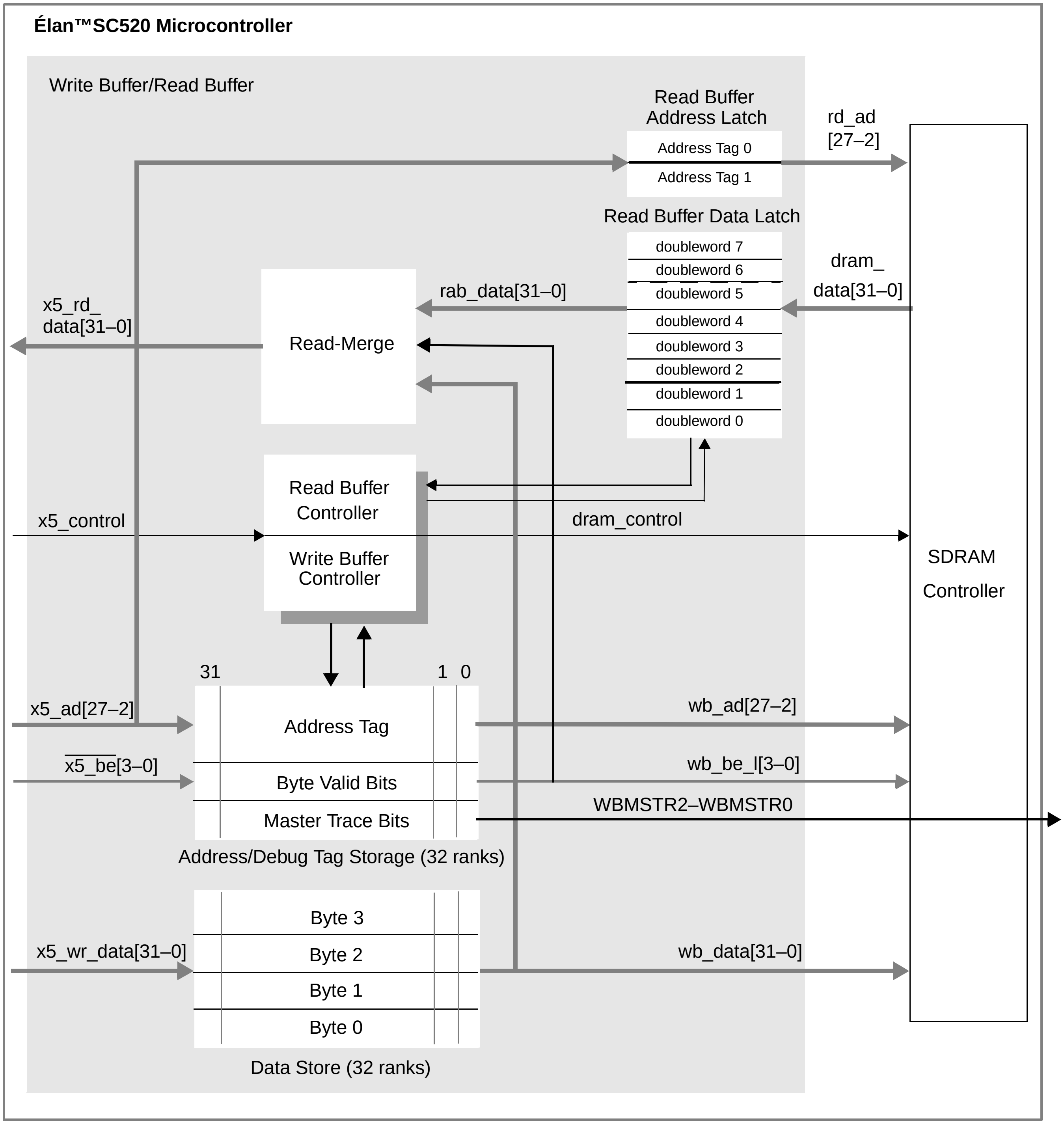

Élan™SC520 Microcontroller
CPU Interface
Address Decode Unit
Write Buffer/Read Buffer
Read Buffer
Write Buffer
SDRAM Controller
MA12–MA0
BA1–BA0
$\overline{SCS3}$–$\overline{SCS0}$
$\overline{SCASB}$–$\overline{SCASA}$
$\overline{SRASB}$–$\overline{SRASA}$
SDQM3–SDQM0
$\overline{SWEB}$–$\overline{SWEA}$
CLKMEMOUT
CLKMEMIN
MECC6–MECC0
MD31–MD0

Figure 11-2 Write Buffer and Read Buffer Block Diagram



11.3 SYSTEM DESIGN

Table 11-1 shows the multiplexing of signals that are used for SDRAM trace support and test. See Chapter 24, “System Test and Debugging”, for more information on the uses of these pins.

The CFG2–CFG0 pinstrap functions associated with these three pins are sampled only as a result of PWRGOOD assertion and do not affect the other functions of these pins, so they are not shown in this table. When enabled, the multiplexed signals shown in Table 11-1 either disable or alter any other function that uses the same pin.

Table 11-1 SDRAM Signals Shared with Other Interfaces

Default Signal	Alternate Function	Control Bit	Register
$\overline{\text{CF\_ROM\_GPCS}}$	WBMSTR0	WB_TST_ENB	SDRAM Control (DRCCTL) register (MMCR offset 10h)
DATASTRB	WBMSTR1		
$\overline{\text{CF\_DRAM}}$	WBMSTR2		

11.4 REGISTERS

The memory-mapped registers for SDRAM buffer control are shown in Table 11-2.

Table 11-2 SDRAM Buffer Control Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
SDRAM Control	DRCCTL	10h	SDRAM write buffer test mode enable
SDRAM Buffer Control	DBCTL	40h	Write buffer enable, read-ahead enable, write buffer watermark, write buffer flush

11.5 OPERATION

The write buffer and read buffer are two features implemented in the SDRAM controller to increase SDRAM performance.

The write buffer provides a mechanism for *all* masters (Am5$_X$86 CPU, PCI, or GP-DMA) to post write data with zero wait states, thus decoupling the master from experiencing the write latency penalty associated with the SDRAM. When the write buffer is enabled, all write activity to SDRAM is initiated by the write buffer.

The read-ahead feature of the read buffer is designed to increase SDRAM read performance by prefetching the cache line following the current access, thus possibly supplying data to the requester with zero wait states. The read-ahead feature takes advantage of the fetch-forward nature of the Am5$_X$86 CPU prefetch engine (which relies on spatial locality of program flow) and PCI read bursts. Read prefetching (when enabled) occurs only for master read accesses that result in a burst of two or more doublewords. A prefetch never occurs for a GP-DMA request since GP-DMA read requests are never burst. However, during a GP-DMA read request, the remainder of the cache line is always fetched.

The write buffer provides a debug feature that, when enabled, provides contributing master information on external pins (WBMSTR2–WBMSTR0) during a write buffer write cycle to SDRAM. These pins reflect which master contributed to the write buffer level in the process of being written back. The contributing masters reflected could be either: Am5$_X$86 CPU, PCI, or GP-DMA. Since the write buffer supports the write-merging and write-collapsing functions, it is possible that multiple masters contributed to the same level that is in the process of being written to SDRAM. See Chapter 24, “System Test and Debugging”, for more information on write buffer debug support.

11.5.1 Write Buffer

The ÉlanSC520 microcontroller's SDRAM controller contains 32 4-byte write data buffers. The write buffer provides benefits beyond that of a standard posting FIFO. A standard FIFO blindly posts data without knowledge of data that already exists within the FIFO. The write buffer is more efficient in that each write access is snooped.

The snoop function is used to determine if data associated with the current address already exists in the FIFO. This feature allows write data to be merged or collapsed with data that already exists in the write buffer. This results in a reduced number of overall writes to SDRAM for contiguous partial doubleword writes and also more efficiently utilizes the FIFO storage. The snoop capability also provides the read-merging function to more efficiently handle data coherency overhead. It does this by not requiring a total write buffer flush before servicing a read cycle (which would ordinarily be required by a standard FIFO that does not provide snooping).

The write buffer provides the following benefits:

- Zero wait state write data posting (to a non-full buffer), effectively decoupling master write activity from SDRAM latency
- Read-around-write support, enhanced by the read-merging function, effectively allowing the SDRAM controller to give read priority over buffered writes to SDRAM
- Write-merging and write-collapsing of write data

The read-around-write feature is provided when the write buffer is enabled. It allows read requests to SDRAM to occur in front of, or around, write buffer requests. Write buffer requests are due to write data that was posted during previous master write activity and is migrating to SDRAM. Read-around-write is only functional when the write buffer is enabled.

11.5.1.1 Write Buffer Disabled

When the write buffer is disabled, all write and read traffic generated by any master is directed around the write buffer directly to SDRAM. Write data is no longer posted, and read cycles no longer require snooping for data coherency. If the write buffer contained valid data when it was disabled, it is automatically flushed (by hardware) to SDRAM as a top priority before SDRAM is free to service any subsequent requests. This guarantees data coherency. Should any master try a read or write access to SDRAM at this time, the cycle is stalled (via wait states) until the write buffer flush is complete.

The write buffer can be manually flushed by setting the WB_FLUSH bit in the SDRAM Buffer Control (DBCTL) register (MMCR offset 40h). Write buffer flush complete status is available after a manual flush by reading the WB_FLUSH bit.

11.5.1.2 Write Buffer Enabled

When the write buffer is enabled, all write data by all masters are written into the write buffer. Data are written into SDRAM from the write buffer in FIFO fashion when the SDRAM controller is free to service the request.

The snoop capability is used to enhance performance for both read and write cycles.

- Through the use of the snoop feature on write cycles, the write buffer can determine if data already exists, and, if so, it either write-merges or write-collapses the data. This enhances write performance through a reduction in the total number of required write cycles to SDRAM for contiguous writes and also makes better utilization of the physical storage space of the buffer.

- For read cycles, the snoop feature is used to determine if data associated with the same address of the read request already exists in the write buffer. If data is already present, that data is read-merged with data being returned from SDRAM. This enhances SDRAM system performance by not requiring the write buffer to be flushed prior to satisfying a read cycle.

Write-merging, write-collapsing, and read-merging functions are described in Section 11.5.1.2.1 and Section 11.5.1.2.2.

Although the write buffer and read buffer service all master SDRAM memory requests, SDRAM reads that fill the Am5$_X$86 CPU cache are more common than SDRAM writes. To satisfy this demand and give priority read access to SDRAM, the write buffer works with the SDRAM controller to alleviate write overhead. This is accomplished by posting write data in zero wait states, effectively freeing the processor earlier to continue. Should a following read cycle occur, the read-around-write feature of the SDRAM controller gives priority to the read cycle to prevent the master from stalling. Without the snooping capability, the entire contents of the write buffer would have to be flushed prior to *any* read cycle in the event that more current data remains posted. Because of the snooping capability, needless flushes are not performed. This results in less overhead to maintain data coherency.

Should a read occur to an address contained in the write buffer, the write buffer merges its data with the data returned from SDRAM. The read-merging function maintains data coherency and eliminates the need to flush the write buffer.

11.5.1.2.1 Write-Merging and Write-Collapsing

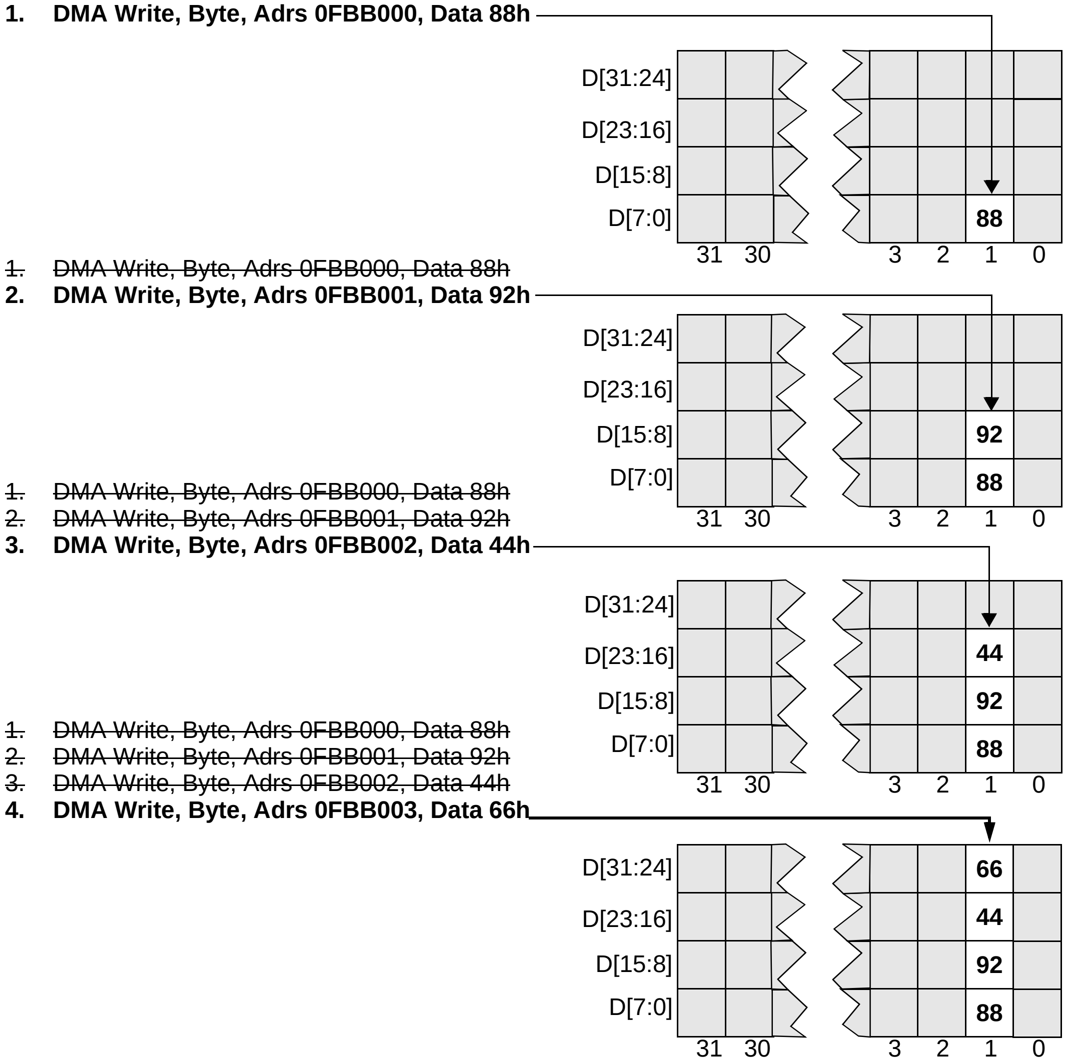
When enabled, the write buffer supports write-merging and write-collapsing.

- *Write-merging*, as illustrated in Figure 11-3 on page 11-7, occurs when a sequence of individual writes are merged into a single doubleword that hits in the write buffer level, or doubleword. However, write-merging implies that the same byte location is not written more than once.
- *Write-collapsing*, as illustrated in Figure 11-4 on page 11-8, is very similar to the write-merging function, with the exception that the same byte location *can* be written more than once. The write-collapsing function allows a sequence of individual writes to hit a single level in the write buffer, even though previous data in that doubleword can be overwritten.

These functions optimize SDRAM performance by minimizing individual writes to SDRAM. There are no dependencies between any doubleword in the write buffer and any of the masters that are capable of posting data to the write buffer. This implies that multiple masters may contribute to the merging or collapsing of any doubleword in the write buffer.

The terms *write-merging* and *write-collapsing* are intended to conform to the meaning as introduced in the *PCI Local Bus Specification,* Revision 2.2.

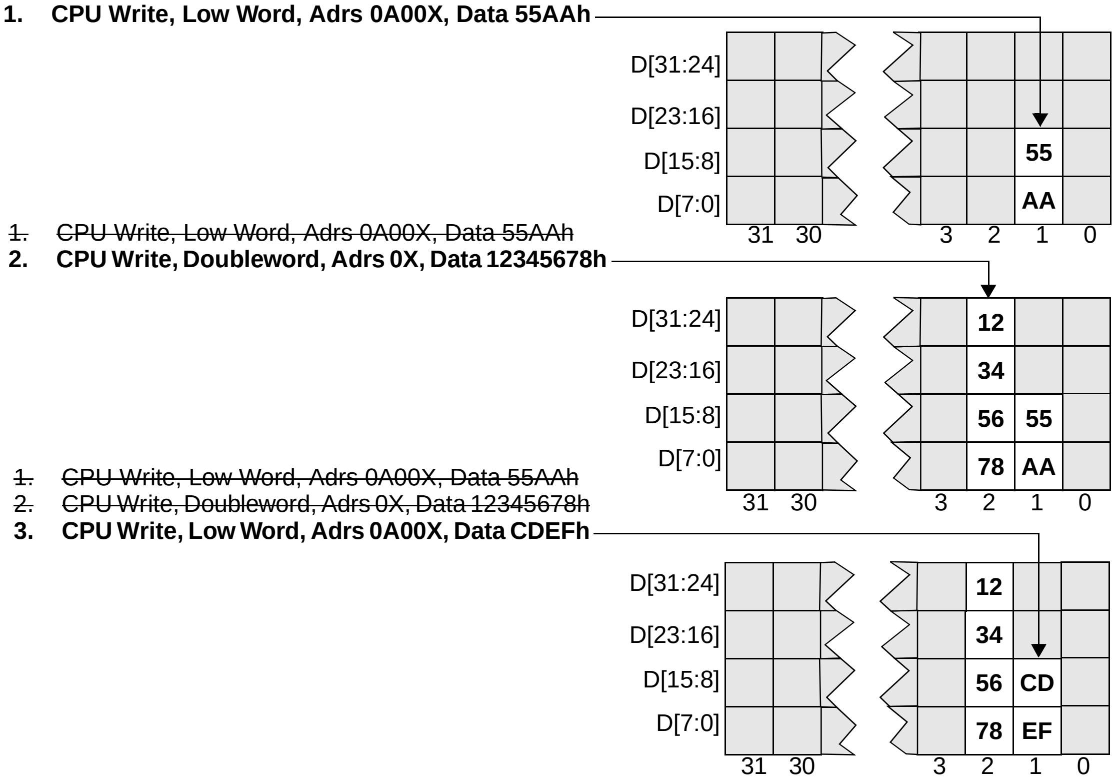
Figure 11-3 Write Buffer Merging Example


1. DMA Write, Byte, Adrs 0FBB000, Data 88h
D[31:24]
D[23:16]
D[15:8]
D[7:0]
88
31 30
3 2 1 0
~~1. DMA Write, Byte, Adrs 0FBB000, Data 88h~~
2. DMA Write, Byte, Adrs 0FBB001, Data 92h
D[31:24]
D[23:16]
D[15:8]
D[7:0]
92
88
31 30
3 2 1 0
~~1. DMA Write, Byte, Adrs 0FBB000, Data 88h~~
~~2. DMA Write, Byte, Adrs 0FBB001, Data 92h~~
3. DMA Write, Byte, Adrs 0FBB002, Data 44h
D[31:24]
D[23:16]
D[15:8]
D[7:0]
44
92
88
31 30
3 2 1 0
~~1. DMA Write, Byte, Adrs 0FBB000, Data 88h~~
~~2. DMA Write, Byte, Adrs 0FBB001, Data 92h~~
~~3. DMA Write, Byte, Adrs 0FBB002, Data 44h~~
4. DMA Write, Byte, Adrs 0FBB003, Data 66h
D[31:24]
D[23:16]
D[15:8]
D[7:0]
66
44
92
88
31 30
3 2 1 0

Notes:

This example illustrates how four separate write cycles can be “merged” and reduced to only one doubleword SDRAM write transaction.

Figure 11-4 Write Buffer Collapsing Example


1. CPU Write, Low Word, Adrs 0A00X, Data 55AAh
D[31:24]
D[23:16]
D[15:8]
D[7:0]
55
AA
31 30
3 2 1 0
~~1. CPU Write, Low Word, Adrs 0A00X, Data 55AAh~~
2. CPU Write, Doubleword, Adrs 0X, Data 12345678h
D[31:24]
D[23:16]
D[15:8]
D[7:0]
12
34
56 55
78 AA
31 30
3 2 1 0
~~1. CPU Write, Low Word, Adrs 0A00X, Data 55AAh~~
~~2. CPU Write, Doubleword, Adrs 0X, Data 12345678h~~
3. CPU Write, Low Word, Adrs 0A00X, Data CDEFh
D[31:24]
D[23:16]
D[15:8]
D[7:0]
12
34
56 CD
78 EF
31 30
3 2 1 0

Notes:

This example illustrates how existing data can be overwritten. Separate write cycles can be "collapsed" and reduced to only one doubleword SDRAM write transaction.

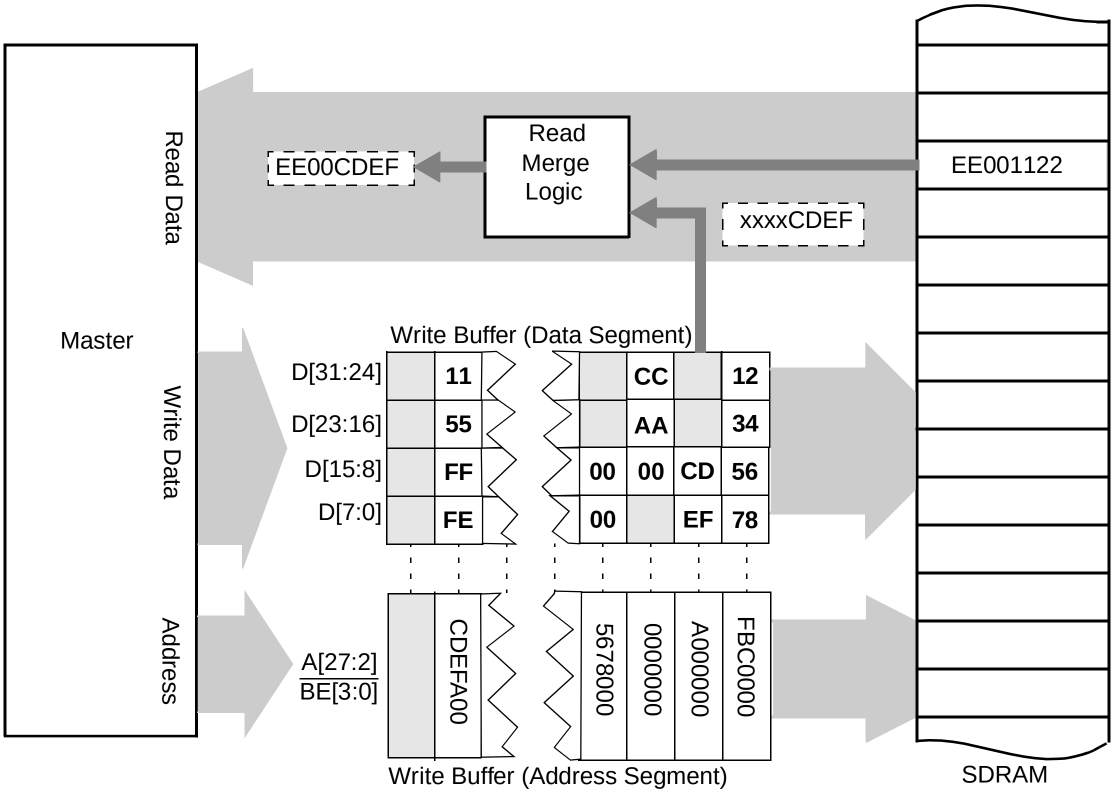
11.5.1.2.2 Read-Merging

The write buffer supports read-merging.

- *Read-merging*, as illustrated in Figure 11-5 on page 11-9, occurs when a read cycle hits a "dirty" doubleword that currently exists in the write buffer, and the read data returned from SDRAM is replaced, or *merged*, with existing bytes from the write buffer.

Read-merging does not negate the need for a SDRAM read cycle. Even during a read cycle that hits a complete dirty doubleword in the write buffer, a read cycle to SDRAM will still occur, but the entire doubleword from SDRAM will be replaced with the more recent doubleword in the write buffer. Read-merging maintains data coherency and enhances SDRAM performance by *not* requiring a flush of the write buffer contents to SDRAM before every read cycle.

Figure 11-5 Write Buffer Read-Merging Example



Notes:

This example illustrates a 32-bit master read of address A000000h, which causes a read hit in the write buffer. This causes the lower data word from the write buffer to be merged with the upper data word from SDRAM, to return the entire doubleword to the requesting master.

11.5.1.3 Write Buffer Watermark

The write buffer provides a watermark setting of either 8, 16, 24 or 28 doublewords. As data is written into the write buffer, a new rank of storage is allocated, provided that write-merging or collapsing is not taking place. When a write cycle resulting in a rank being allocated takes place that exceeds the watermark setting, the write buffer requests service from the SDRAM controller to initiate write transfers to SDRAM.

- A higher watermark setting (i.e., 28 doublewords) allows the write buffer to acquire more master write data prior to requesting SDRAM controller attention than a lower watermark setting. If a large amount of incomplete doubleword writes (i.e., byte, word, or three byte write transfers) is expected from either the Am5$_x$86 CPU, PCI, or GP-DMA, a higher watermark setting allows the write buffer to fill higher prior to requesting SDRAM service, resulting in a greater chance of write data merging or collapsing.
- A lower watermark setting can be used for applications that require more complete doublewords, and where merging/collapsing of data is less likely. This causes the write buffer to request SDRAM service at a lower threshold, thus reducing the chance of filling the write buffer.

The write buffer watermark setting can be configured with the WB_WM bit in the SDRAM Buffer Control (DBCTL) register (MMCR offset 40h). A waterrmark of 16 doublewords is recommended. Note that the write buffer must be disabled before changing the write buffer watermark.

The SDRAM controller's arbiter supports a write buffer *park* feature, such that after the write buffer's watermark is reached and requests SDRAM service, the SDRAM controller's arbiter continues to grant the write buffer SDRAM service, until either a master read cycle is requested to SDRAM or a SDRAM refresh occurs. After the write buffer's grant is removed, the write buffer's watermark will need to be exceeded prior to the write buffer requesting SDRAM service again. This park feature allows the write buffer to utilize SDRAM access until a higher priority master read or an SDRAM refresh cycle is requested.

11.5.2 Read Buffer and the Read-Ahead Feature

The SDRAM controller contains eight 4-byte read data buffers. Combined, these buffers make up the read buffer and are designed to hold two cache lines of data returned from SDRAM. The read buffer is designed to increase SDRAM read performance by storing previously read data from SDRAM and supplying this data in zero wait states to a requesting master.

The SDRAM controller always fetches an entire cache line of data from SDRAM and stores it in the read buffer, independently of the amount of data requested during the master access. For example, during a read request from a non-bursting master (i.e., single doubleword request), the SDRAM controller fetches the entire cache line of data from SDRAM and stores it in the read buffer. When the read-ahead feature of the read buffer is enabled and the master read access is a burst of two or more doublewords, not only is the requested cache line (i.e., the demanded line) of data retrieved from SDRAM, but also the cache line following it.

A demand fetch implies that the SDRAM controller will be servicing the read request from the master as it occurs. When the read-ahead feature is enabled, a read-ahead prefetch only occurs for master demand burst requests of two or more doublewords. The read-ahead feature takes advantage of the linear forward-fetch nature of the Am5$_X$86 CPU and PCI bursts. GP-DMA transfers are non-burst, and thus do not result in a prefetch. However, GP-DMA transfers can utilize the remainder of the cache line, since all read accesses result in a cache line access to SDRAM.

The read buffer provides storage for two cache lines of read data and cannot be disabled. The read-ahead feature of the read buffer can be disabled.

11.5.2.1 Read-Ahead Feature Disabled

When the read-ahead feature is disabled, the prefetch feature of the SDRAM controller is disabled. All master read requests that occur to SDRAM are demand fetches and always result in an entire cache line of data being read from SDRAM. Even when the read-ahead feature is disabled, both cache lines of storage of the read buffer are still utilized and contain the last two demand cache line fetches.

11.5.2.2 Read-Ahead Feature Enabled

When the read-ahead feature is enabled, following cache line prefetches from SDRAM will occur when the read access is a burst of two or more doublewords. The prefetched cache line always follows the demanded cache line. Should an access result in a read buffer hit, the read-ahead logic will request the cache line following the access that is currently being supplied from the read buffer.

The read buffer is organized as two cache lines of data and an associated address tag. On every read cycle these tags are compared to the read address being requested. If the compare results in a hit, this data is supplied to the requesting master in zero wait states. If, during this hit, the *next* cache line of data does not already exist in the read buffer, the prefetch logic will request it from SDRAM. Should a request result in a read buffer miss, the demanded read cycle request is satisfied by SDRAM, and the prefetch logic starts a request to acquire the next cache line. The demanded read cycle implies that the first doubleword request by the master will be serviced first, such that the master can continue while the remainder of the cache line is prefetched.

If the read-ahead feature of the read buffer is enabled, a prefetch occurs only for master read access that results in a burst of two or more doublewords. Single doubleword read requests do not result in a read-ahead prefetch and only result in the cache line of the demanded access being read into the read buffer. GP-DMA read accesses are always a single doubleword.

To maintain coherency in the system, each cache line of the read buffer has associated with it a valid bit that represents the validity of the cache line. Both cache-line valid bits are cleared on the occurrence of master write access to SDRAM or a write buffer write access to SDRAM that hits a cache line currently available in the read buffer.

11.5.3 DMA Considerations

The read buffer and its associated read-ahead feature provide optimum performance for burst-capable masters (during read cycles) that maintain long bus tenure (with burst transfers of two or more doublewords). Most masters with burst capability burst forward an entire cache line. For these masters, the read-ahead feature provides optimum performance, such that the anticipated data prefetch will result in a read buffer hit.

- The read-ahead feature performs well during Am5$_X$86 CPU burst reads (which usually result in full cache-line burst when the cache is enabled). During cache-line fills, the Am5$_X$86 CPU can maintain bus tenure for more than one burst transfer, such that successive bursts will be satisfied by read buffer prefetch hits.
- Also, during PCI master read burst requests, the read-ahead feature of the read buffer performs equally well for PCI master tenure to SDRAM that requests a cache line of data.
- However, since the GP-DMA controller supports multiple channels and is capable of operating in either single, demand or block transfer modes, it is possible that the read buffer performance during GP-DMA transfers becomes dependent on the GP-DMA channel configurations.

As mentioned earlier, the SDRAM controller always fetches an entire cache line from SDRAM during a read request, even if the read-ahead feature is disabled. Since DMA transfers are *non*-burst (i.e., single doubleword requests), even if the read-ahead feature is enabled, only the rest of the cache line is fetched, rather than the rest of the cache line and the following cache line, as would be seen during burst transfers of two or more doublewords.

- A DMA channel configured for incrementing order that starts at the beginning of a cache line takes full advantage of read buffer hit, since all following incrementing access should result in a read buffer hit up to the cache-line boundary, assuming demand or block transfer mode.
- DMA transfers that are configured for decrement mode will also see a read buffer benefit, since the remainder of the cache line is fetched. For DMA transfers that are configured for decrement mode, maximum read buffer performance is seen when the first access is at the end of a cache-line boundary.

DMA transfer mode types can have a direct impact on read buffer performance. It would be ideal for the same DMA channel to hit the read buffer as much as possible during its tenure.

In a system configured with multiple active DMA channels, read buffer misses will most likely occur for each change of channel tenure. This is because each DMA channel accesses different SDRAM regions that will most likely miss the read buffer, which still contains the cache line of data fetched during the previous channel's tenure. Therefore, it would be ideal for as many transfers to occur as possible while a particular DMA channel has access to SDRAM to utilize the rest of the cache line fetched during the DMA transfer's first doubleword request. This implies that, in a system with many active DMA channels configured for single transfer mode, read buffer misses will occur that do not utilize the cache line of data fetched during the previous channels tenure.

Demand and block DMA transfer modes will most likely take advantage of the rest of the cache-line fetches, since devices that use these modes typically have longer bus tenure, resulting in a higher utilization of the fetched data.

11.5.4 PCI Considerations

As a PCI target, the ÉlanSC520 microcontroller can respond to PCI master write and read requests to SDRAM. To facilitate large burst transfers as a PCI target, a 64-level write data FIFO and 64-level read data FIFO is available in the PCI target logic.

11.5.4.1 Write Cycles

For PCI master burst writes to SDRAM, the ÉlanSC520 microcontroller can sustain zero wait states until the PCI target write FIFO is filled. As the FIFO is filling at the PCI interface, data is being removed from the FIFO and written to SDRAM. When the SDRAM controller's write buffer is enabled, data can be quickly transferred from the PCI target write FIFO to the SDRAM write buffer in zero wait states (to a non-full write buffer), allowing the PCI target write FIFO to empty quickly. This prevents the PCI master from experiencing the SDRAM latencies, thus freeing up the PCI bus earlier.

During PCI target write transfers to SDRAM, the Am5$_X$86 CPU cache is snooped to maintain coherency. If the CPU cache is configured in write-back cache mode and a snoop results in a hit, the modified Am5$_X$86 CPU cache line must be written back to memory prior to allowing the PCI target write transfer to take place. When the write buffer is enabled, the Am5$_X$86 CPU cache-line write-back is posted to the write buffer, and the following PCI target write transaction collapses on top of the previously written cache-line write-back, resulting in a reduction in the overall number of transactions to memory.

11.5.4.2 Read Cycles

In most applications, a PCI master transfers data to SDRAM and then interrupts the processor when the transfer is complete. The processor then usually accesses this data in SDRAM. Since the write buffer supports read-merging, associated data that is still in the write buffer from the PCI transfer may be immediately read by the processor, without the overhead of first flushing the write buffer before allowing the read to occur. Also, since the SDRAM controller allows read-around-write activity when the write buffer is enabled, the processor reads are allowed to occur around writes that are posted in the write buffer, thus offering a performance increase to processor read requests.

During PCI master read transfers from SDRAM, the ÉlanSC520 microcontroller's PCI target read FIFO is filled with data read from SDRAM. This data is then supplied to the requesting PCI master directly from the target's read FIFO. Since PCI bursts are linear and forward in nature, the SDRAM controller's read-ahead feature prefetches data (from SDRAM) forward from the PCI master's start address. As the ÉlanSC520 microcontroller's PCI target read FIFO requests data from SDRAM, it is likely that these requests will result in read buffer hits due to prefetching, thus providing data quickly to the PCI target read FIFO.

Large PCI burst requests will benefit more from the read-ahead function than short, frequent independent PCI read transfers. Since the Am5$_X$86 CPU is a major requestor of SDRAM read accesses, short and frequent independent PCI transfers may result in read-ahead thrashing. For example, data prefetched for Am5$_X$86 CPU read requests may possibly not be used by PCI read requests and data prefetched for the PCI request may possibly not be used by the Am5$_X$86 CPU.

11.5.5 Software Considerations

The write buffer and read buffer require minimal configuration overhead.

Data coherency is maintained in hardware during write buffer configuration changes. This implies that when the write buffer is disabled, the contents are automatically flushed to SDRAM as a high priority, prior to allowing any master activity to occur to SDRAM. Even though a write buffer flush occurs automatically when it is disabled, a manual write buffer flush control is provided for software control via the WB_FLUSH bit in the SDRAM Buffer Control (DBCTL) register (MMCR offset 40h). If the read-ahead feature is disabled, the prefetched data remains in the read buffer.

Both the write buffer and read-ahead feature of the read buffer are disabled after a system reset or programmable reset. It is recommended that the write buffer be disabled prior to SDRAM sizing, SDRAM test, or other software activity that must have guaranteed write data delivery to the physical SDRAM array prior to reading. Failure to disable the write buffer for these usages may result in false SDRAM sizing or test indications.

Typically during SDRAM sizing or test, SDRAM is written and then read back to determine either if SDRAM exists at that location (during sizing) or if SDRAM is functional at that location (during test). Since the write buffer provides a read-merging function to reduce the overhead associated with maintaining data coherency, data is *not* forced from the write buffer to SDRAM prior to the read-back of the data. (This overhead would normally be required for *non*-snooping write buffers that do not support read-merging to maintain coherency.) Should the read occur while the associated write data is still in the write buffer, the correct data is read-merged with data from SDRAM, thus providing the correct read data even though the write data was not yet written to SDRAM. If, in this scenario, SDRAM was non-existent, it would appear as though it did exist, thus resulting in either an invalid SDRAM size or false “pass” status during a SDRAM test algorithm. If the write data migrated to SDRAM *before* the read-back, a correct indication would result.

The write buffer must be disabled only in these scenarios where software requires guaranteed delivery of write data to SDRAM prior to testing. Under normal program execution, the write buffer and read buffer “appear” as the SDRAM storage array.

11.5.6 SDRAM Bandwidth Improvements

When enabled, the performance benefit that the write buffer offers is its ability to effectively decouple the master write activity from incurring the SDRAM latency penalty. This in effect leaves the SDRAM free to satisfy a higher demand in read activity by all masters. To further optimize this, when the write buffer is enabled, it allows master read requests to occur around write data posted in the write buffer. In effect, read cycles are given priority to SDRAM when the write buffer is enabled. However, there are conditions that give the write buffer write priority to SDRAM over reads. These are:

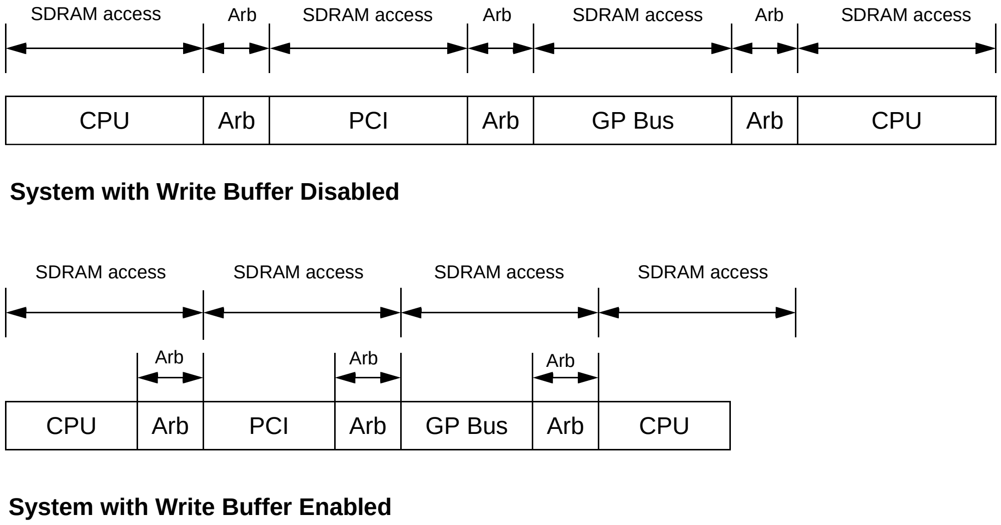
- Flush priority is given to the write buffer when the write buffer configuration changes to disabled.
- The user exercises the manual write buffer flush feature.

Since the write buffer supports data read-merging, data coherency overhead is kept to a minimum. The write buffer's read-merging capability is possible due to the write buffer's ability to snoop its own contents during read and write cycles. In the special case of a read to an address contained in the write buffer, the overhead associated with flushing the entire contents of the write buffer to maintain data coherency is eliminated. In this case, as data is returned from SDRAM during the read cycle, more current data in the write buffer is merged into the data stream, replacing older data bytes being returned from SDRAM. This greatly enhances the read-around-write behavior by eliminating the overhead associated with flushing the write buffer to maintain coherency.

The maximum write buffer performance is seen during individual contiguous byte writes to SDRAM. For example, suppose the GP-DMA was performing a 64-byte block transfer from an 8-bit device to SDRAM. Without the write buffer, this would require 64 individual byte-wide transfers to SDRAM. Because of the write buffer's write data-merging capability, each contiguous byte could be merged to form only 16 doubleword transfers to SDRAM. This would reduce the total number of SDRAM writes cycles from 64 to 16 in this example.

The write buffer also improves memory system performance during heavy SDRAM write data thrashing between multiple masters. Since the write buffer provides zero wait state posting of write data, the SDRAM interface is freed up earlier to service another master's request. While the next master is arbitrating for SDRAM, the write buffer can concurrently be writing back the data posted by previous masters. Therefore, during heavy SDRAM write thrashing periods by multiple masters, the write buffer can help to hide the arbitration overhead. This is shown in Figure 11-6.

Figure 11-6 Bus Thrashing with Write Buffer Disabled and Enabled



The maximum benefit of the read buffer's read-ahead feature is provided during consecutive prefetch hits. This will most likely occur during long master burst tenure or consecutive bursts by the same master. For example, suppose a PCI master requests a 256-byte (64-doubleword) read transfer from SDRAM. Since the read buffer prefetches a cache line forward and PCI burst transfers are linear and forward in nature, consecutive requests can be satisfied by data prefetched by the read-ahead feature.

11.6 INITIALIZATION

The write buffer and read buffer are reset during a system reset. As a result of this system reset event, the write buffer and read-ahead feature of the read buffer are both disabled, and all associated state machines are returned to their idle states.

During a programmable reset, the write buffer's contents are reset and not maintained. The contents of the read buffer are maintained during a programmable reset. The write buffer and read-prefetch configuration are *not* preserved during a programmable reset. See Chapter 6, "Reset Generation", for more detailed information on this type of reset.

It is recommended that, prior to SDRAM sizing and test, the write buffer be disabled to prevent false SDRAM sizing or test indications. It is also recommended that, during SDRAM sizing or test, the read-ahead feature is disabled. Having the read-ahead feature enabled will *not* result in false indications during sizing or test, but may result in a slight performance degradation during the SDRAM sizing or test algorithm, because read accesses are not consecutive in nature during sizing or test. After this period, the user is free to enable the write buffer and read-ahead feature when desired.

CHAPTER

12 ROM/FLASH CONTROLLER

12.1 OVERVIEW

The ÉlanSC520 microcontroller includes an integrated ROM controller that provides a high performance interface to ROMs, EPROMs, and Flash devices. Improved performance is achieved by supporting a full 32-bit data path and advanced page-mode devices.

Note that in this document the term *ROM* is used interchangeably with Flash and EPROM for simplicity. In addition, the term *ROM* is used to denote the entire bank of ROM devices connected to a chip select, e.g., a 32-bit ROM can be implemented as four discrete 8-bit ROM devices.

Features of the ROM controller include:

- Support for a wide variety of industry standard ROMs, EPROMs, and Flash devices, including advanced page-mode devices.
- Three chip selects are provided. Each chip select supports up to 64 Mbytes.
  - One chip select is dedicated to booting.
  - The remaining two chip selects are optional and are multiplexed with GP bus chip selects.
- Programmable timing for both non-page-mode and page-mode devices is supported.
- Programmable Address Region (PAR) register attributes provide code execution control, cacheabilitity control, and write protection for Flash devices

The ÉlanSC520 microcontroller supports 8-bit, 16-bit, and 32-bit ROM configurations.

- The GP address bus is always used for the ROM address, but the ROM data bus can be connected to either the GP bus data bus or the SDRAM data bus.
- For the boot device ($\overline{\text{BOOTCS}}$), a set of configuration pins latched into the chip when PWRGOOD is asserted is used to determine the width of the ROM array and which of the two buses (GP bus or SDRAM interface) is used for the ROM data bus.
  - The remaining two optional chip selects are configured via configuration registers in the ROM controller.
- 8-bit and 16-bit ROM configurations are supported when ROMs are connected to either the GP bus or the SDRAM data bus. 32-bit ROM configurations are supported only when ROMs are connected to the SDRAM data bus, as shown in Table 12-1.

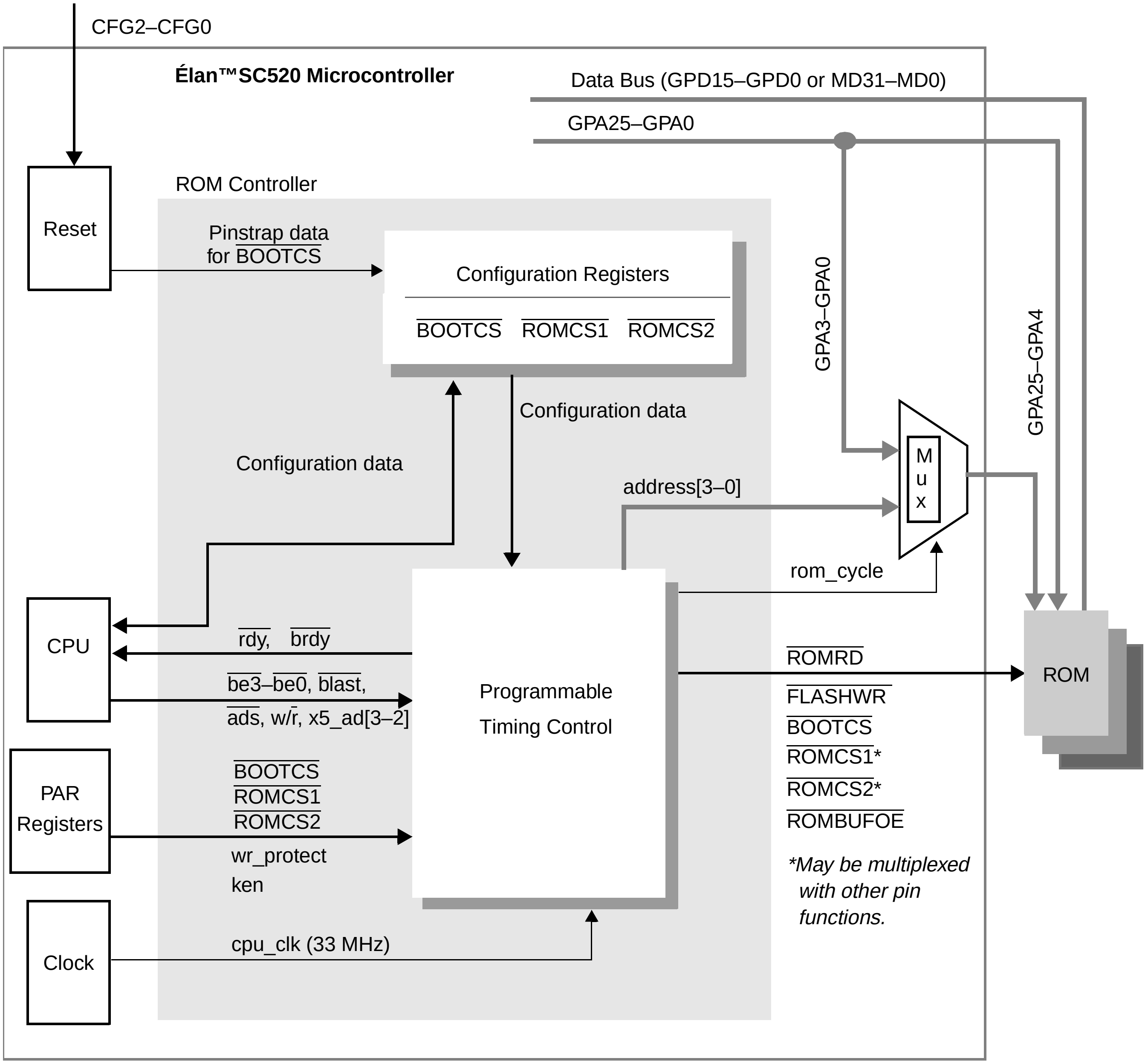
Table 12-1 ROM/Flash Data Bus Connection Options

Data Bus	8-Bit ROM	16-Bit ROM	32-Bit ROM
GP Bus data pins	Yes	Yes	No
SDRAM interface data pins	Yes	Yes	Yes

12.2 BLOCK DIAGRAM

Figure 12-1 shows a block diagram of the ROM controller.

Figure 12-1 ROM Controller Block Diagram



12.3 SYSTEM DESIGN

See the *Élan™SC520 Microcontroller Data Sheet*, order #22003, for timing tables and additional timing diagrams.

Configuration information for the boot device (BOOTCS), specifically the width of the ROM and the location of the ROM, is provided by external pinstrapping. The CFG2 pinstrapping defines the bus, either SDRAM or GP bus data bus, on which the ROM is located. The CFG1–CFG0 pins define the data width of the ROM devices. CFG2–CFG0 are latched when PWRGOOD is asserted. See “Initialization” on page 12-14 for more information.

The ROMCS1 and ROMCS2 signals are provided to support two additional ROM chip selects. These pins are shared with general-purpose chip selects, GPCS1 and GPCS2, respectively, as shown in Table 12-2. When enabled, the multiplexed signals shown in Table 12-2 either disable or alter any other function that uses the same pin.

Table 12-2 ROM Signals Shared with Other Interfaces

Default Signal	Alternate Function	Control Bit	Register
ROMCS2	GPCS2	GPCS2_SEL	Chip Select Pin Function Select (CSPFS) register (MMCR offset C24h)
ROMCS1	GPCS1	GPCS1_SEL	

The ROM controller can accommodate various performance and system voltage isolation requirements. Depending on the operation voltage required by the ROM and the voltage required by other devices sharing the same bus, the ROM data pins can be connected either to the GP bus or to the SDRAM interface (see Figure 12-2). Note that the ROM data pins must connect to only one interface per chip select (i.e., the ROM data pins may not straddle the two buses).

- Devices can be placed on the SDRAM bus to gain the advantage of a 32-bit data path. However, care must be taken by the system designer because of SDRAM loading and timing issues. See “System Design” on page 10-1.
- Alternately, the ROM devices can be implemented on the GP bus data pins. These devices are limited to 8- or 16-bits. See Table 12-1 for data width connection options.

Note that the addresses for ROM devices are always provided via GP bus, independently of whether the data pins of the ROM are connected to the GP bus or SDRAM bus.

12.3.1 Voltage Isolation

From the ÉlanSC520 microcontroller’s perspective, both the SDRAM bus and the GP bus are 5-V-tolerant and drive 3.3 V. However, an isolation buffer is necessary when using the same bus for 5-V ROM devices and 3.3-V SDRAM devices that are not 5-V-tolerant. For example, if the 3.3-V SDRAM devices are not 5-V-tolerant and share the data bus with 5-V ROM devices, the 3.3-V SDRAM devices could be damaged during ROM read access if an isolation buffer is not used.

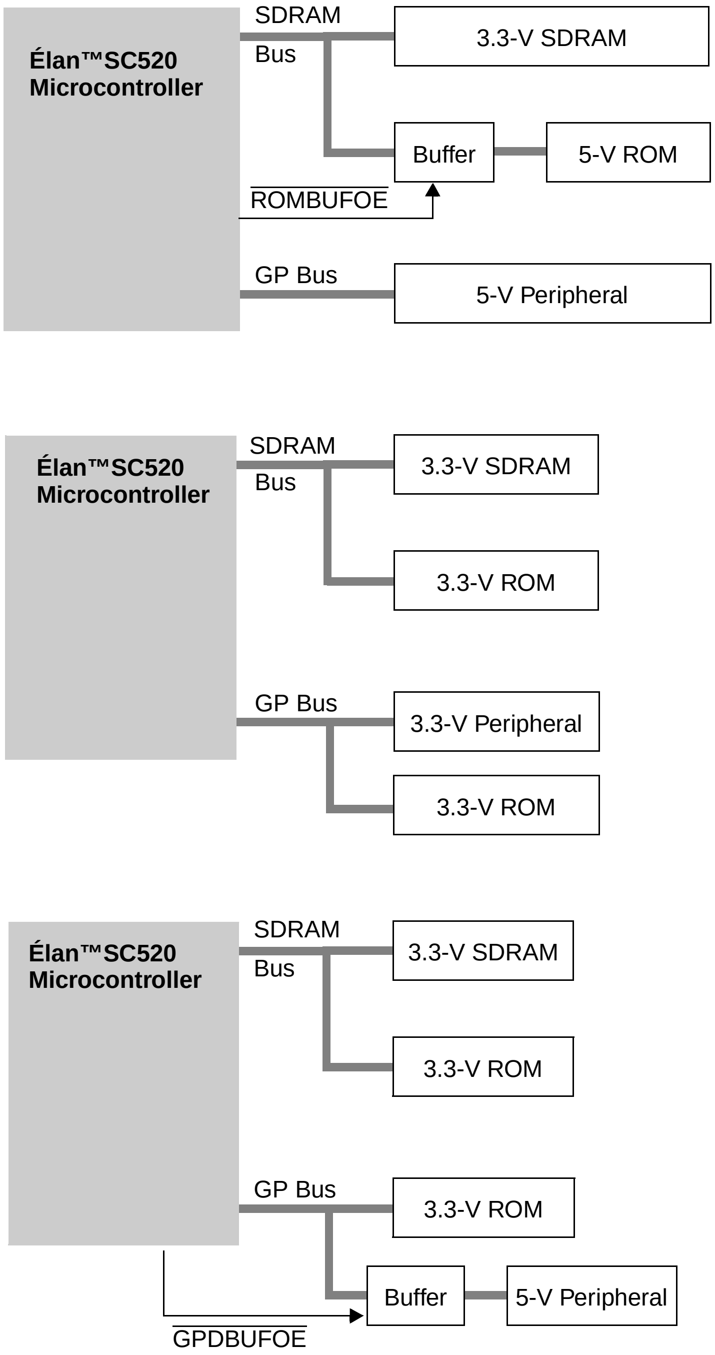
The ROMBUFOE signal is provided to support an isolation buffer, and this signal can be used for devices on the SDRAM bus or the GP bus. Some scenarios for such a situation are shown in Figure 12-2. The ROMBUFOE signal asserts during all accesses to ROM devices, whether the devices are located on the SDRAM bus or the GP bus.

Note that the SDRAM controller’s read and write buffers are not utilized during accesses to ROM devices. This is true even if a ROM device is located on the SDRAM bus. When the SDRAM buffering is enabled, the ROM devices connected to the SDRAM data bus (MD31– MD0), must use ROMRD to control the ROM device’s data pins. In this case, the system design should ensure that the external device does not drive data while ROMRD is asserted.

When sharing the SDRAM data bus with ROM devices, the loading of the data bus requires careful consideration. A buffer should be used on the data bus to prevent heavy loading by the ROM devices. In a system that utilizes buffering of these devices, the ROMBUFOE signal can be used to control the buffers. Similarly, data buffers can be used on the GP bus to control loading issues, and the ROMBUFOE pin should still be used to control buffers in front of these ROM devices.

If the system has ROM devices on both the SDRAM data bus and the GP bus and data bus buffers are used (on either bus), ROMBUFOE should be qualified with the appropriate ROM chip selects and ROMRD, as needed, to prevent bus conflicts. For example, when SDRAM buffering is enabled, the SDRAM controller could be attempting to complete posted writes to the SDRAM. During this time, if the Am5$_x$86 CPU performs a read from a ROM device that is on the GP bus (data bus), the buffer on the SDRAM bus (which isolates the ROM devices from the SDRAM) activates, unless its buffer control pins were also qualified with the ROMCSx pin.

Figure 12-2 Voltage Isolation Examples



Notes:

Both the GP bus and the SDRAM bus can be operated at either 5 V or 3.3 V.

12.4 REGISTERS

Table 12-3 shows the memory-mapped registers used to configure the ROM controller.

Table 12-3 ROM Controller Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
$\overline{\text{BOOTCS}}$ Control	BOOTCSCTL	50h	$\overline{\text{BOOTCS}}$ device select (SDRAM bus or GP bus), device data width, device operation mode, subsequent access delay, first access delay
$\overline{\text{ROMCS1}}$ Control	ROMCS1CTL	54h	$\overline{\text{ROMCS1}}$ device select (SDRAM bus or GP bus), device data width, device operation mode, subsequent access delay, first access delay
$\overline{\text{ROMCS2}}$ Control	ROMCS2CTL	56h	$\overline{\text{ROMCS2}}$ device select (SDRAM bus or GP bus), device data width, device operation mode, subsequent access delay, first access delay
Chip Select Pin Function Select	CSPFS	C24h	$\overline{\text{ROMCSx}}$ or $\overline{\text{GPCSx}}$ pin function select

12.5 OPERATION

ROM/Flash devices in a system are typically used to store two different kinds of information: system configuration data and program code. These applications impose different constraints on how to use the ROM/Flash memory in the system.

While it may be sufficient to load system configuration information from ROM/Flash at a low speed, this may be not acceptable for accessing ROM-resident code that has to be executed. In this case, this code has to be copied either to SDRAM or executed directly from ROM. (See Chapter 3, “System Initialization” for more information on shadowing.)

For copying code blocks, the ROM performance may not be critical, because it is only accessed once per copy operation. For the more critical situation of executing code directly from the ROM (e.g., an Execute-In-Place operating system), precautions have to be taken to ensure an accelerated ROM access even for ROM devices incapable of bursting.

This chapter discusses different configurations and operating modes that are appropriate for these varying situations.

12.5.1 ROM Support

Each of the three chip selects included on the ÉlanSC520 microcontroller supports up to 64 Mbytes. Some example configurations for each chip select are:

- Four 1-Mbit x 8 devices on the 32-bit SDRAM data bus for a total of 4 Mbytes
- Two banks of ROM, with each bank containing four 8-Mbit x 8 devices, providing a total of 64 Mbytes
- Two banks of ROM, with each bank containing two 8-Mbit x 16 devices, providing a total of 64 Mbytes
- Four banks of ROM, with each bank containing two 8-Mbit x 8 devices, providing a total of 64 Mbytes

ROM devices are accessible by the Am5$_x$86 CPU only. Normal operation of the ÉlanSC520 microcontroller is not guaranteed if an external PCI master or GP-DMA cycle results in a ROM access.

The addresses for ROM devices are always provided via the GP bus, independently of whether the data pins of the ROM are connected to the GP bus or SDRAM bus.

The ROM controller never bus-sizes read accesses to the Am5$_x$86 CPU. In other words, $\overline{\text{bs16}}$ and $\overline{\text{bs8}}$ are never asserted for a ROM read access. Rather, the ROM controller gathers as much data as the Am5$_x$86 CPU is requesting for read accesses. To accomplish this, the ROM controller monitors the internal byte enable signals, $\overline{\text{be3}}$–$\overline{\text{be0}}$, and the cacheability status of the access. Based on the byte enables, the ROM controller returns one to four bytes for non-burst Am5$_x$86 CPU cycles and up to an entire cache line, 16 bytes, for burst accesses.

The ROM controller does not support burst-write or multiple data operations during write cycles. Writes to ROM devices typically have no performance issues. The ROM controller returns $\overline{\text{rdy}}$, rather than $\overline{\text{brdy}}$, to the Am5$_x$86 CPU during write operations. In addition, the Am5$_x$86 CPU signals $\overline{\text{bs8}}$ and $\overline{\text{bs16}}$ are asserted based on data size of the selected ROM device.

12.5.1.1 Supported Device Types

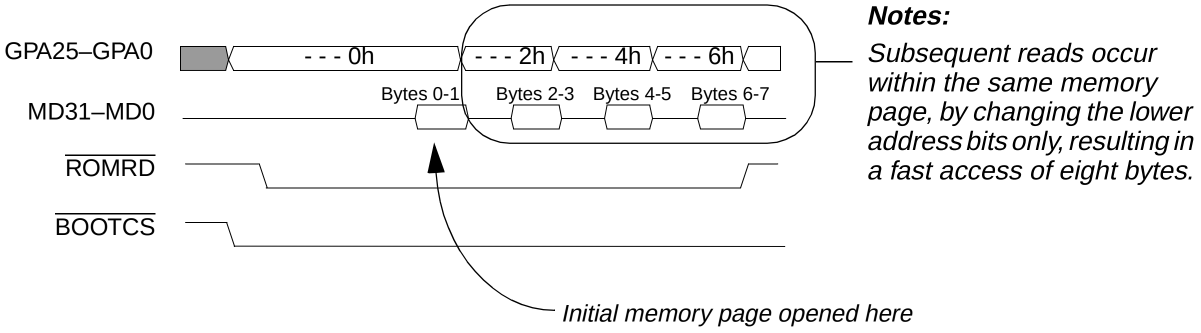
The ROM controller supports two ROM device types:

- Non-page-mode ROM—A ROM device that always has the same access delay, regardless of how much data is requested from the ROM.
- Advanced page-mode ROM—These devices improve performance by allowing fast multiple access of data within the same memory page. The ROM controller has no upper limit on the page size of the ROM device and works with any device that supports a page size of four. However, after the fourth entry in the page, the ROM controller issues a new initial access.The page is opened during the initial access, allowing faster data reads from subsequent locations within the page simply by strobing the lower address bits.

Non-page-mode and advanced page-mode ROMs do not require a clock signal.

Figure 12-3 illustrates a read of four words from a 16-bit advanced page-mode ROM. Note that the write buffer associated with the SDRAM controller has no relevance for the ROM controller, because it applies only to SDRAM accesses.

Figure 12-3 Page-Mode ROM: Fetching Four Words from a 16-Bit ROM



Notes:

Subsequent reads occur within the same memory page, by changing the lower address bits only, resulting in a fast access of eight bytes.

12.5.2 ROM Control and Timing Configuration

The ÉlanSC520 microcontroller provides ROM device configuration per chip select for the following:

- ROM location (on GP data bus or SDRAM data bus)
- ROM width (8, 16, or 32 bits)
- Operating mode (page-mode or non-page-mode)
- Access timing

12.5.2.1 ROM Location

The GP bus address is always used for the ROM address, but the ROM data bus can be connected to either the GP bus data bus or the SDRAM data bus.

For the boot device ($\overline{\text{BOOTCS}}$), the CFG2 pinstrap is used to determine which of the two buses is used for the ROM data bus. For all other ROM devices ($\overline{\text{ROMCS1}}$ and $\overline{\text{ROMCS2}}$), this configuration information must be programmed by the initialization software.

- The DGP bit in the $\overline{\text{BOOTCS}}$ Control (BOOTCSCTL) register (MMCR offset 50h) contains the value latched from the CFG2 pinstrap when the PWRGOOD pin is asserted.
- The DGP bit in the $\overline{\text{ROMCS1}}$ and $\overline{\text{ROMCS2}}$ control registers is used to configure the location of the ROM devices enabled by these two chip selects.

12.5.2.2 ROM Width

ROM device widths of 8 bits, 16 bits, and 32 bits are supported.

The CFG1–CFG0 pinstraps are used to determine the width of the boot device ($\overline{\text{BOOTCS}}$). For all other ROM devices ($\overline{\text{ROMCS1}}$ and $\overline{\text{ROMCS2}}$), this configuration information must be programmed by the initialization software.

- The WIDTH bit field in the $\overline{\text{BOOTCS}}$ Control (BOOTCSCTL) register contains the value latched from the CFG1–CFG0 pinstraps when the PWRGOOD pin is asserted.
- The WIDTH bit field in the $\overline{\text{ROMCS1}}$ and $\overline{\text{ROMCS2}}$ control registers is used to configure the width of the ROM devices enabled by these two chip selects.

12.5.2.3 Operating Mode

The MODE bit in the control registers provided for each chip select signal is used to program the operating mode of the associated device.

According to the different data delivery rates, the following operation modes are distinguished:

- Non-page mode—Characterized as having the same access time for all cycles. Figure 12-4 shows a ROM that is capable of three wait state operation.
- Page mode—Provides faster timing for subsequent data that falls within the page-size of the ROM device. Figure 12-5 shows an advanced page-mode ROM that is capable of one wait state for the first access and zero wait states for subsequent accesses.

If an unaligned access to a page-mode device is executed, i.e., when not all data are located in the same ROM page, a new page has to be opened, which imposes an additional delay (see Figure 12-6). Random access within a page is not supported.

Figure 12-4 Non-Page-Mode ROM: Fetching Four Words from a 16-Bit ROM

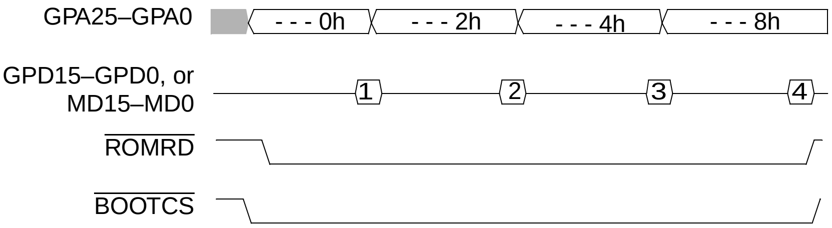


Figure 12-5 Page-Mode ROM: Fetching Four Doublewords (Aligned) from a 32-Bit ROM

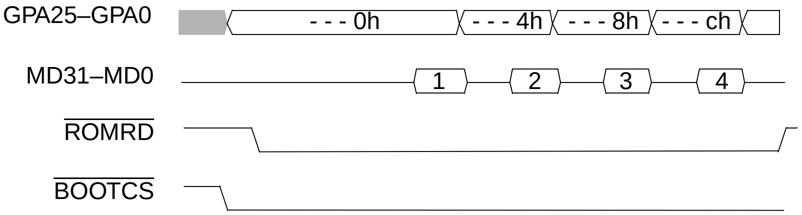
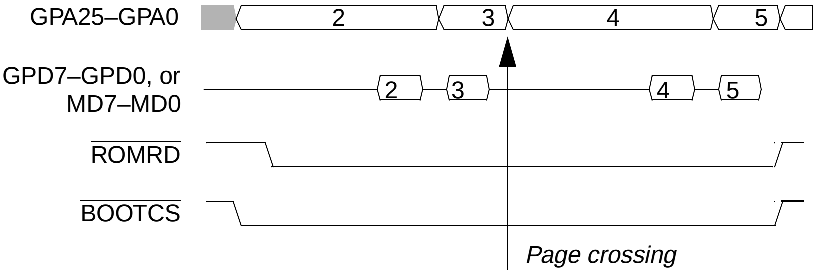


Figure 12-6 Page-Mode ROM: Fetching Four Doublewords (Unaligned) from an 8-Bit ROM



12.5.2.4 Access Timing

Access timing is controlled in the $\overline{\text{BOOTCS}}$ or $\overline{\text{ROMCSx}}$ Control registers.

– The delay for the first access, used for both non-page-mode and page-mode, and subsequent accesses for non-page-mode is specified in the FIRST_DLY bit field.

– The delay for subsequent accesses, for page-mode only, is specified in the SUB_DLY bit field.

Table 12-4 shows the access timing according to the programmed wait states. These values can be obtained using the following formula:

AccessTime = (NumberWaitstates + 1) * Period –Setup)

where:

Period is the clock period (assume 30 ns for a 33.333-MHz crystal)

Setup is assumed to be 20 ns. (It takes the actual setup time and the delay for address changes during subsequent ROM accesses into account.)

Table 12-4 Example: ROM Access Timing and Wait States[1]

Wait States	Access Timing (ns)
0	10
1	40
2	70
4	130

Notes:
1. This example assumes that a 33.333-MHz crystal is being used in the system.

12.5.3 Bus Cycles

The ROM controller always returns the amount of read data requested by the Am5$_{X}$86 CPU, i.e., $\overline{\text{brdy}}$ is returned for all read transfers from ROM. The actual number of ROM accesses is determined by the cacheability status of the Am5$_{X}$86 CPU transfer, the number of bytes requested, and the width of the ROM. The minimum number of data to be transferred is one byte. The maximum number of data to be delivered is 16 bytes (a cache-line fill). Depending on the ROM width, this leads to different numbers of accesses to fetch the requested data (see Table 12-5).

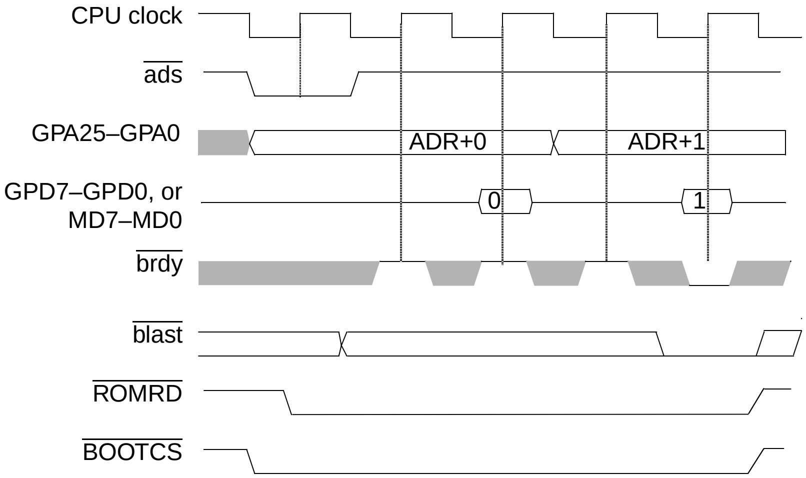
Table 12-5 Accesses and ROM Width

ROM Width	Minimum Number of Accesses	Maximum Number of Accesses (Cache-Line Fill)
8 bit	1	16
16 bit	1	8
32 bit	1	4

12.5.3.1 Single CPU Read Access

Figure 12-7 shows an example for the fetching of 16-bits of data, GPD15–GPD0, from an 8-bit non-page-mode ROM configured for one wait state. The transfer starts with a bus cycle initiation (i.e., $\overline{\text{ads}}$ asserted). The ROM controller then performs two ROM accesses and accumulates the amount of requested data prior to terminating the cycle. Note that only one ROM cycle would be performed had the ROM device been implemented as 16- or 32-bit.

Figure 12-7 Multiple Accesses: Data Amounts Smaller than One Doubleword (2 Bytes) from an 8-Bit ROM

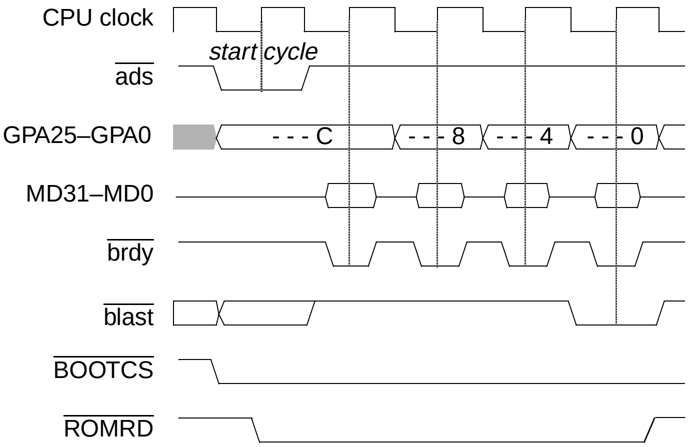


Notes: *An 8-bit ROM is attached to the 16-bit GP bus.*

12.5.3.2 Page-Mode Read Access

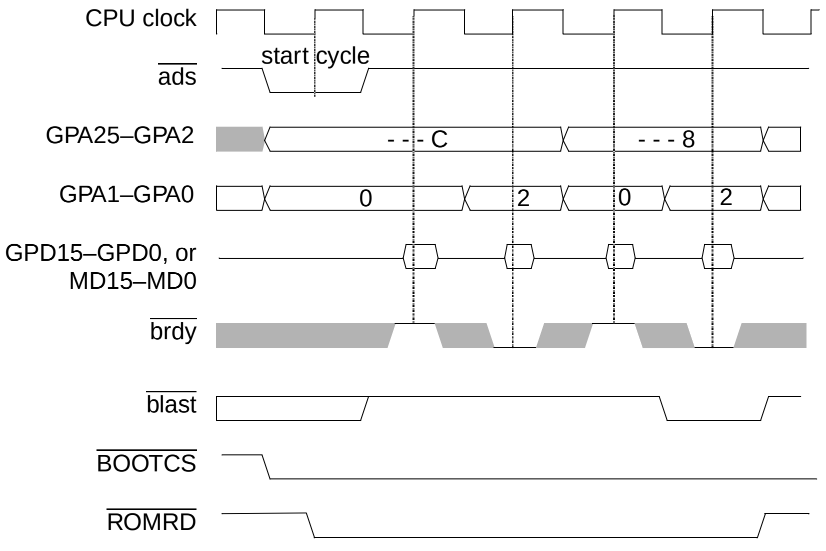
The ROM controller also provides performance advantages for Am5$_x$86 CPU burst operation. Further improvement can be achieved when using page-mode ROMs. An example is shown in Figure 12-8 for a 2-1-1-1 burst sequence, in which the first access requires two cycles and all subsequent accesses are performed within one cycle.

Figure 12-8 Page Access for Fetching Four Doublewords from a 32-Bit ROM (Burst Sequence: 2-1-1-1)



During burst transfers to ROM devices with a data width smaller than 32 bits, the ROM controller executes multiple cycles to gather the requested data. During a 32-bit request to a 16-bit device, the ROM controller executes two 16-bit cycles. During a 32-bit request to an 8-bit device, the ROM controller executes four 8-bit cycles. A 32-bit request to a 16-bit ROM device is shown in Figure 12-9.

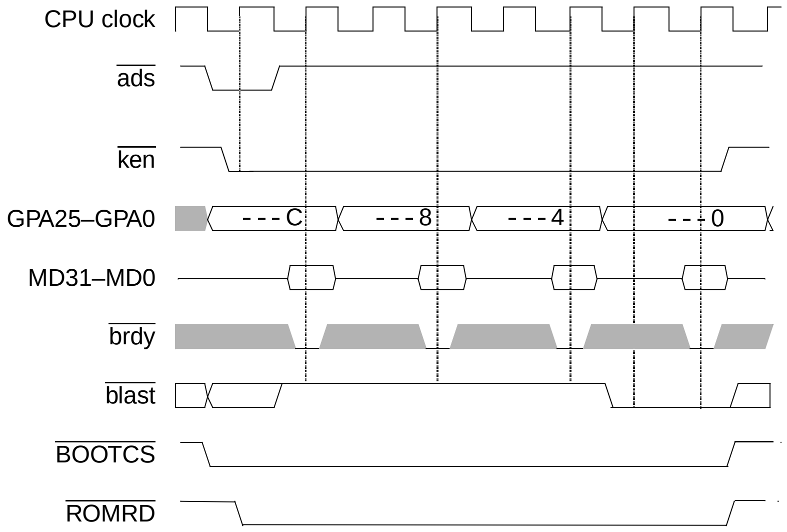
Figure 12-9 Page Access for Fetching Two Doublewords from a 16-Bit ROM



12.5.3.3 Cache-Line Fill

If a memory section is accessed that is cacheable, the $\overline{\text{ken}}$ signal is asserted to the Am5$_x$86 CPU indicating a cache-line fill operation. This causes the Am5$_x$86 CPU to read four doublewords (16 bytes) and leads to multiple ROM accesses. A cache-line fill to a 32-bit ROM is depicted in Figure 12-10.

Figure 12-10 Cache-Line Fill (Fetching Four Doublewords from a 32-Bit ROM)

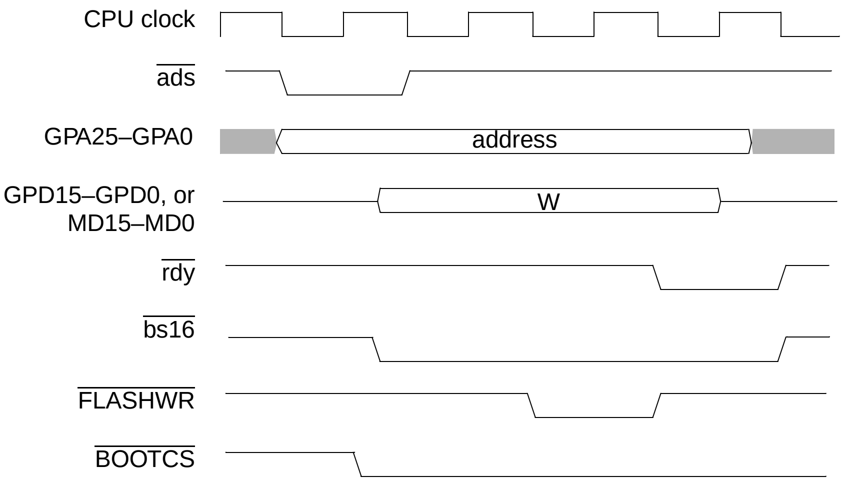


12.5.3.4 Writing to Flash Devices

The ÉlanSC520 microcontroller supports writable Flash devices. Since Flash devices are not intended for random write accesses, no burst-write operations are supported, i.e., $\overline{\text{rdy}}$ is returned to the Am5$_x$86 CPU. Figure 12-11 shows a write cycle to a Flash ROM. In addition, for write accesses, the ROM controller bus-sizes ROM accesses to indicate the width of the ROM device, e.g., if a 16-bit write is performed to an 8-bit ROM, two Am5$_X$86 CPU write cycles are generated to complete the operation.

All write access to Flash devices must occur in units no smaller than the data width of the device. For example, 8-bit writes to a 16-bit Flash device are not allowed. Care should be taken to also avoid 24-bit writes to 16-bit Flash devices, because this generates two Flash cycles, one with a complete 16-bit write and another with an 8-bit write to a 16-bit Flash device.

Figure 12-11 Word Write Cycle to Flash Memory



12.5.4 Software Considerations

12.5.4.1 Address Decoding

The ROM controller does not perform address decoding. Address decoding for chip select generation is provided by the Programmable Address Region (PAR) registers. In addition to the regions defined in the PAR registers, a default region from FFFF0000–FFFFFFFFh is defined at system reset to handle early code fetches from the boot ROM. See Chapter 3, “System Initialization”, and Chapter 4, “System Address Mapping”, for further details on configuring the address regions for ROM chip selects and the shadowing of ROM.

12.5.4.2 Programming Flash Memory

Flash is available in 8-bit and 16-bit versions and is organized into sectors. Sectors can be of fixed or variable size and range from 8–32 Kbytes. New, higher density Flash devices have sector sizes of up to 256 Kbytes.

Several programmable operations can be performed on Flash devices, including sector erase, sector protect, and programming of individual bytes.

- The erased value of a byte is 0FFh.
- Bits can be programmed from a 1 to a 0.
- If any bit in a sector needs to be changed from a 0 to a 1, the entire sector must be erased and reprogrammed.

Most Flash devices cannot be programmed while the Am5$_X$86 CPU is fetching data from it, requiring the programming code to reside in another device during programming. This is an easy restriction to overcome, because programming Flash is usually done during non-performance critical periods, such as during user configuration. However, new “dual boot” Flash allows fetching instructions from one portion of the device while programming or erasing a sector in another portion.

Typically, Flash is programmed (or erased) by writing a program command sequence to an address within the sector to be modified, followed by the erase command or the target address and data.

An example program command sequence is:

1. Write the byte AAh to address 555h within the sector.
2. Write the byte 55h to address 2AAh within the sector.
3. Write the byte A0h to address 555h within the sector.
4. Write the actual data to the actual address. If the base of a 1-MByte boot device is at 0FFF00000h, then a programming sequence for the first sector would start at address 0FFF00555h.

The actual values and addresses used vary by device.

After issuing the command, the programming code must wait until the embedded algorithm is complete before sending further programming requests to the Flash device. There are several ways to determine this.

- One way is to poll the status of a ready/busy hardware pin (which would be connected to a PIO pin).
- The second way is to continually read the address that was programmed, looking for one of several indications that the event is complete.

A typical waiting period is 16 ms. Sector erase can take from 1 s to up to 10 s near the end of the serviceable life of the device.

Both the program command sequence and the status read have implications for the use of the ÉlanSC520 microcontroller in Flash programming applications.

First, the area being programmed must set to be noncacheable. Writing the program command sequence does not actually change the physical addresses involved, meaning that caching this area would yield incorrect data the next time it is read. Also, the status read phase relies on the value of externally supplied bits to change from one read to another. Obviously, satisfying such a read from the cache would not work. Once the programming is complete, it is legitimate and desirable to enable caching on this region.

Another obvious implication is that programming Flash device requires a write strobe to be connected to the device. Devices are programmed in their natural word length, meaning that byte write enables are not required. During writes, there are minimum times for the write strobe pulse width. These follow naturally from the total chip enable cycle time, which would be used to determine the number of wait states to use when accessing the device for reads, requiring no special timing modifications. Flash requires a minimum reset pulse width of 500 ns, which is well within the ÉlanSC520 microcontroller’s minimum time.

12.5.5 Latency

ROM latency refers to the amount of time in which a ROM access can impact system performance. For example, during an Am5$_X$86 CPU access to ROM, no other master in the system will be granted access to the SDRAM resource. The latency time will be mainly affected by the width and the access time of the ROM device.

The lowest latency times can be achieved if fast 32-bit ROMs are implemented for Execute-In-Place (XIP) operating systems or for data structures that are accessed frequently. This ensures a rapid data transfer, which frees up the SDRAM resource for access by other masters.

For example, if four doublewords are accessed from a 2-1-1-1 advanced page-mode ROM, five clock cycles are required to load this data. However, loading the same amount of data from an 8-bit, non-page-mode ROM results in 48 clock cycles, assuming two wait states per ROM access. While the first approach promises reasonable performance, the latter imposes a latency that is possibly unacceptable.

12.6 INITIALIZATION

The ROM controller is connected to the ÉlanSC520 microcontroller's system reset.

The system designer must define the boot ROM configuration devices connected to $\overline{\text{BOOTCS}}$ using pinstrapping. The CFG2–CFG0 pins provided on the ÉlanSC520 microcontroller are latched at the assertion of PWRGOOD to define the location and data width of the boot device, as shown in Table 12-6.

- CFG2 defines whether the boot device is located on the SDRAM data bus or GP bus data bus.
- CFG1–CFG0 define the data width of the boot device.
- $\overline{\text{BOOTCS}}$ is forced active at system reset. Boot code must then initialize a Programmable Address Region (PAR) register to decode the required space for the boot ROM device. See “External ROM Devices” on page 3-17 for examples.

Table 12-6 CFGx Pinstrap Configuration Options for $\overline{\text{BOOTCS}}$

CFG2	CFG1	CFG0	$\overline{\text{BOOTCS}}$ Data Width	$\overline{\text{BOOTCS}}$ Location
0	0	0	8-bit	GP bus
0	0	1	16-bit	GP bus
1	0	0	8-bit	SDRAM bus
1	0	1	16-bit	SDRAM bus
1	1	x (don’t care)	32-bit	SDRAM bus

Non-boot devices that exist on $\overline{\text{ROMCS1}}$ and $\overline{\text{ROMCS2}}$ do not require pinstrapping and are configured with the ROM configuration registers.

At system reset, the ROM controller is enabled for $\overline{\text{BOOTCS}}$ only. The following steps should be taken to further configure $\overline{\text{BOOTCS}}$ and/or to enable other ROM devices.

1. Configure the ROM width, mode, access timing, and location in the $\overline{\text{BOOTCS}}$ Control (BOOTCSCTL) register (MMCR offset 50h), the $\overline{\text{ROMCS1}}$ Control (ROMCS1CTL) register (MMCR offset 54h), and/or the $\overline{\text{ROMCS2}}$ Control (ROMCS2CTL) register (MMCR offset 56h).
2. Set up the address range and the cacheability control, write protection, and code execution control attributes for the $\overline{\text{BOOTCS}}$ device or the $\overline{\text{ROMCSx}}$ device in the PAR registers.

CHAPTER

13 GENERAL-PURPOSE BUS CONTROLLER

13.1 OVERVIEW

The ÉlanSC520 microcontroller includes an integrated general-purpose bus (GP bus) controller. The GP bus is an internal and external bus that connects 8-bit or 16-bit peripheral devices and memory to the ÉlanSC520 microcontroller without glue logic.The GP bus operates at 33 MHz, which provides good performance at very low interface cost.

Features of the general-purpose bus controller include:

- Up to eight external chip selects ($\overline{\text{GPCS7}}$–$\overline{\text{GPCS0}}$)
- Supports 8- and 16-bit I/O and memory cycles
- Programmable bus interface timing
- Dynamic bus sizing using $\overline{\text{GPIOCS16}}$ and $\overline{\text{GPMEMCS16}}$
- Dynamic wait state support for external devices using GPRDY
- Up to 64 Mbytes of memory address space per chip select
- Supports 8- and 16-bit DMA initiators

13.2 BLOCK DIAGRAM

Figure 13-1 shows the block diagram of the GP bus controller.

13.3 SYSTEM DESIGN

Table 13-1 shows GP bus signals shared with other interfaces on the ÉlanSC520 microcontroller. The pinstrap functions associated with the GPA25–GPA14 pins are sampled only as a result of PWRGOOD assertion and do not affect the GP bus functions of these pins, so they are not shown in this table. When enabled, the multiplexed signals shown in Table 13-1 either disable or alter any other function that uses the same pin.

A ROM device’s data bus can be connected to either the GP bus data bus or the SDRAM data bus. However, the addresses for ROM devices are always provided via the GP bus, independently of whether the data pins of the ROM are connected to the GP bus or SDRAM bus. In either case, the ROM access shares GPA25–GPA0 with the GP bus.

For additional system diagrams using the GP bus, see “Interfacing with a Super I/O Controller” on page 13-13 and “Interfacing with an AMD Enhanced Serial Communications Controller (8 MHz)” on page 13-14.

See the *Élan™SC520 Microcontroller Data Sheet*, order #22003, for timing tables and additional timing diagrams.

Figure 13-1 GP Bus Controller System Block Diagram

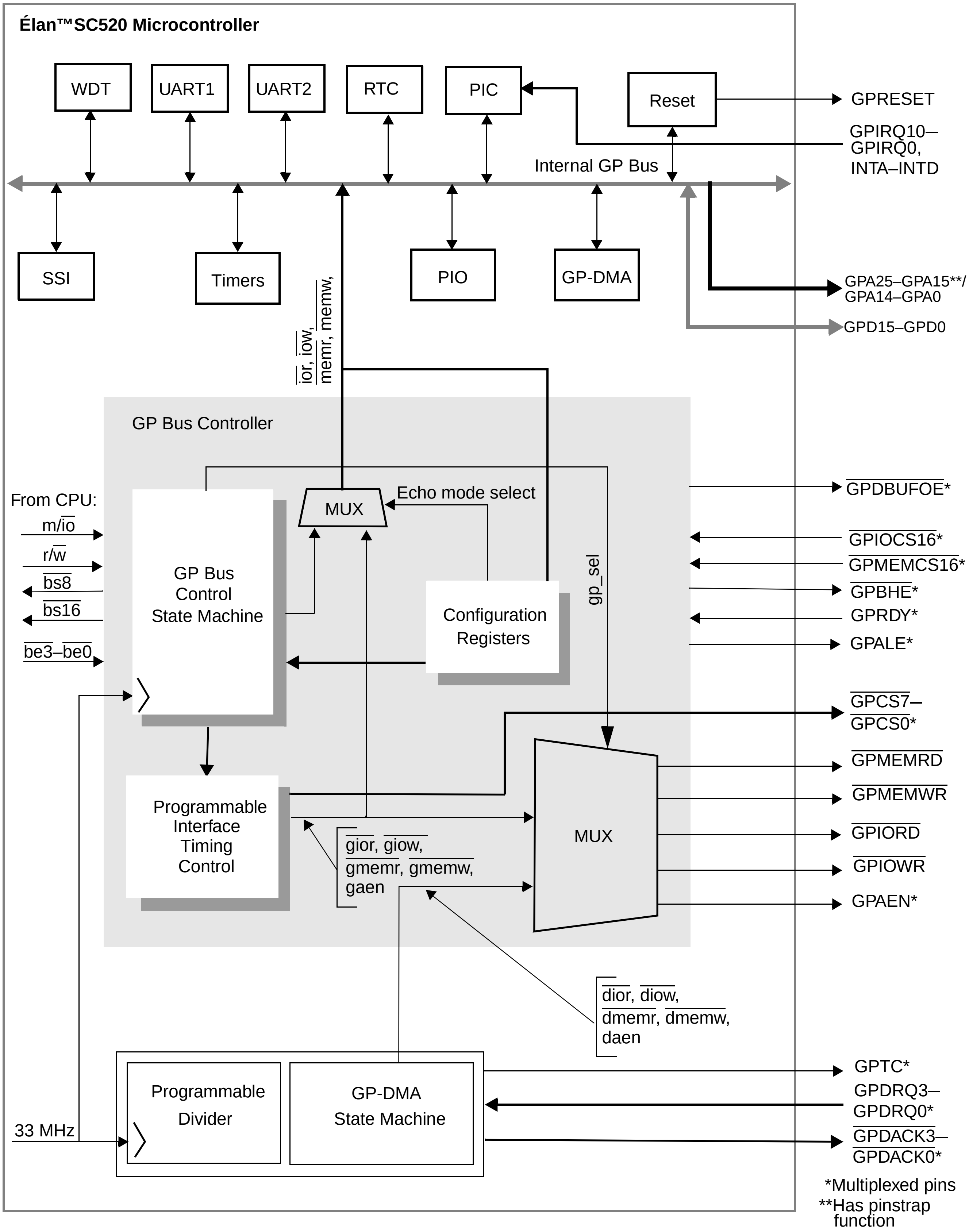


Table 13-1 GP Bus Signals Shared with Other Interfaces

Default Signal	Interface or Alternate Function	Control Bit	Register
TMROUT0	$\overline{\text{GPCS7}}$	GPCS7_SEL	Chip Select Pin Function Select (CSPFS) register (MMCR offset C24h)
TMROUT1	$\overline{\text{GPCS6}}$	GPCS6_SEL	
TMRIN0	$\overline{\text{GPCS5}}$	GPCS5_SEL	
TMRIN1	$\overline{\text{GPCS4}}$	GPCS4_SEL	
PITGATE2	$\overline{\text{GPCS3}}$	GPCS3_SEL	
$\overline{\text{ROMCS2}}$	$\overline{\text{GPCS2}}$	GPCS2_SEL	
$\overline{\text{ROMCS1}}$	$\overline{\text{GPCS1}}$	GPCS1_SEL	
PIO27	$\overline{\text{GPCS0}}$	PIO27_FNC	PIO31–PIO16 Pin Function Select (PIOPFS31_16) register (MMCR offset C22h)
PIO26	$\overline{\text{GPMEMCS16}}$	PIO26_FNC	
PIO25	$\overline{\text{GPIOCS16}}$	PIO25_FNC	
PIO24	$\overline{\text{GPDBUFOE}}$	PIO24_FNC	
PIO23	GPIRQ0	PIO23_FNC	
PIO22	GPIRQ1	PIO22_FNC	
PIO21	GPIRQ2	PIO21_FNC	
PIO20	GPIRQ3	PIO20_FNC	
PIO19	GPIRQ4	PIO19_FNC	
PIO18	GPIRQ5	PIO18_FNC	
PIO17	GPIRQ6	PIO17_FNC	
PIO16	GPIRQ7	PIO16_FNC	
PIO15	GPIRQ8	PIO15_FNC	PIO15–PIO0 Pin Function Select (PIOPFS15_0) register (MMCR offset C20h)
PIO14	GPIRQ9	PIO14_FNC	
PIO13	GPIRQ10	PIO13_FNC	
PIO12	$\overline{\text{GPDACK0}}$	PIO12_FNC	
PIO11	$\overline{\text{GPDACK1}}$	PIO11_FNC	
PIO10	$\overline{\text{GPDACK2}}$	PIO10_FNC	
PIO9	$\overline{\text{GPDACK3}}$	PIO9_FNC	
PIO8	GPDRQ0	PIO8_FNC	
PIO7	GPDRQ1	PIO7_FNC	
PIO6	GPDRQ2	PIO6_FNC	
PIO5	GPDRQ3	PIO5_FNC	
PIO4	GPTC	PIO4_FNC	
PIO3	GPAEN	PIO3_FNC	
PIO2	GPRDY	PIO2_FNC	
PIO1	$\overline{\text{GPBHE}}$	PIO1_FNC	
PIO0	GPALE	PIO0_FNC	

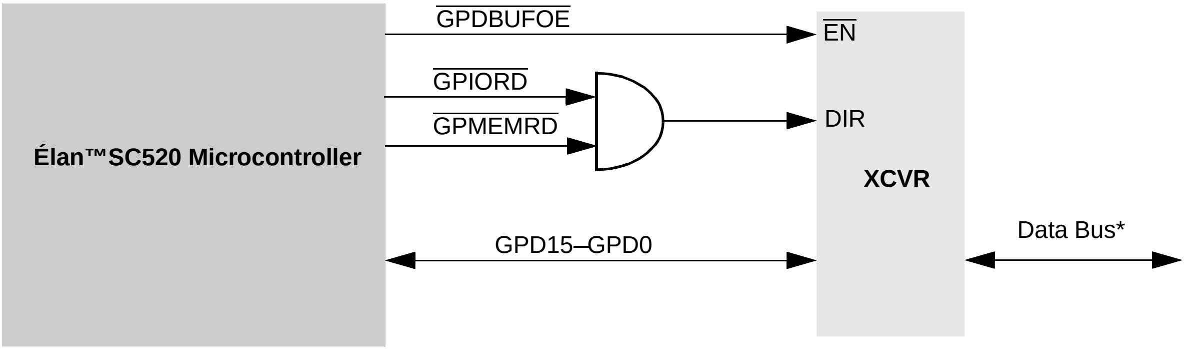
13.3.1 GP Bus Loading

As more external devices are connected to the GP bus, loading on GPA25–GPA0 and GPD15–GPD0 will increase. Therefore, the rise time and fall time of GPA25–GPA0 and GPD15–GPD0 will increase, and external buffers may be needed to reduce the loading.

The GP bus provides the $\overline{\text{GPDBUFOE}}$ pin for external buffer control to reduce the loading. This signal is asserted for all accesses to external GP bus peripherals. It is not asserted during accesses to the internal peripherals (regardless of the GP bus echo mode setting).

Figure 13-2 shows an example using an external data buffer. The $\overline{\text{GPDBUFOE}}$ pin can be used to enable the data buffer, and the $\overline{\text{GPIORD}}$ or $\overline{\text{GPMEMRD}}$ can be qualified together to select the direction of the data buffer. If all devices on the GP bus are only I/O-mapped devices, the AND gate in Figure 13-2 is not required. The $\overline{\text{GPIORD}}$ pin can be used to control the direction of the data transceiver. A similar simplification can be applied if all devices are memory-mapped using the $\overline{\text{GPMEMRD}}$ pin.

Figure 13-2 Example: Using an External Data Buffer to Address Excess Loading



Notes:

If the GP address bus must be buffered, ensure that the buffer is always enabled.

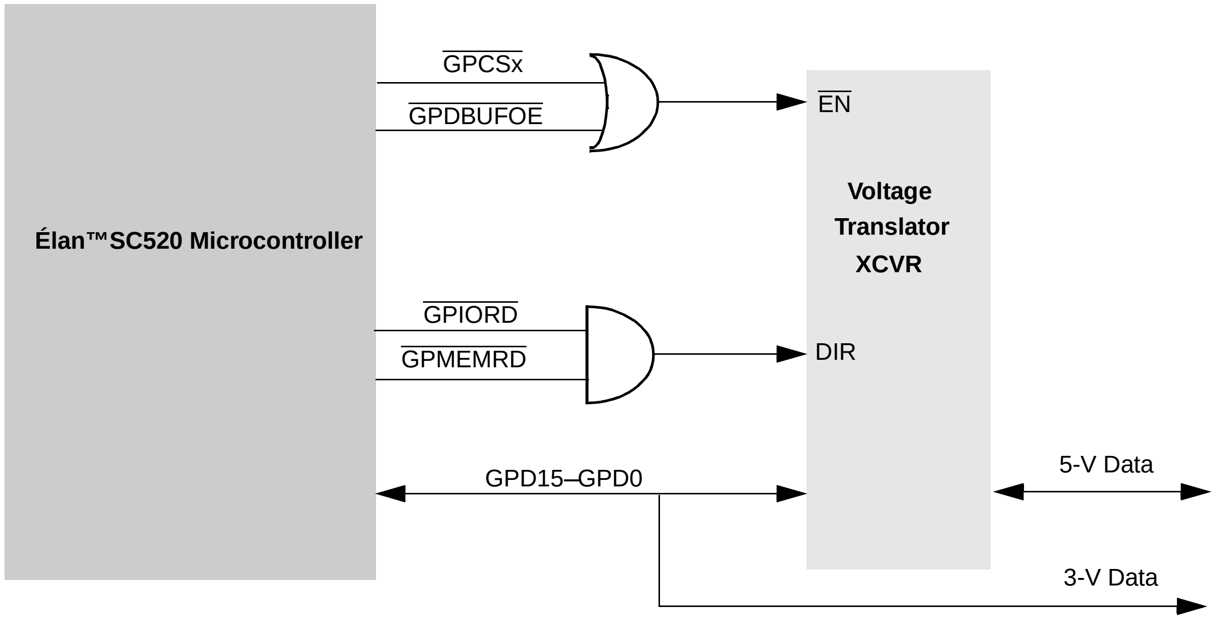
** All GP bus peripherals connect their data to this bus.*

The $\overline{\text{GPIOCS16}}$, $\overline{\text{GPMEMCS16}}$, and GPRDY pins are typically driven by open-drain outputs from external devices and require a strong pullup resistor (typically 1 Kohm) external to the ÉlanSC520 microcontroller. The GPIRQx pins also require pullup resistors (typically 1 Kohm).

13.3.2 Voltage Translation

The GP bus provides 5-V- tolerant inputs and 3-V outputs, but if the external devices contain both 3-V and 5-V devices, the $\overline{\text{GPDBUFOE}}$ pin qualified with a $\overline{\text{GPCSx}}$ signal can be used to control the voltage translator. Figure 13-3 shows one example of using a voltage translator.

Figure 13-3 Example: Using a Voltage Translator



Notes:

$\overline{\text{GPCSx}}$ is the chip select for the 5-V peripheral.

13.4 REGISTERS

Table 13-2 shows the memory-mapped registers used to configure the GP bus controller.

Table 13-2 GP Bus Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
GP Echo Mode	GPECHO	C00h	Echo mode enable
GP Chip Select Data Width	GPCSDW	C01h	Individual data width select for $\overline{\text{GPCS7}}$–$\overline{\text{GPCS0}}$
GP Chip Select Qualification	GPCSQUAL	C02h	Individual chip select qualification with $\overline{\text{GPIORD}}$, $\overline{\text{GPIOWR}}$, $\overline{\text{GPMEMRD}}$, or $\overline{\text{GPMEMWR}}$
GP Chip Select Recovery Time	GPCSRT	C08h	Global chip select recovery time for all GP bus cycles. Affects all GP bus chip selects.
GP Chip Select Pulse Width	GPCSPW	C09h	Global width selection for all chip select signals, measured from the offset
GP Chip Select Offset	GPCSOFF	C0Ah	Global offset time selection for all chip selects from the beginning of the bus cycle
GP Read Pulse Width	GPRDW	C0Bh	Width of the $\overline{\text{GPIORD}}$ and $\overline{\text{GPMEMRD}}$ signals from the offset
GP Read Offset	GPRDOFF	C0Ch	Offset from the beginning of the bus cycle for $\overline{\text{GPIORD}}$ and $\overline{\text{GPMEMRD}}$
GP Write Pulse Width	GPWRW	C0Dh	Width of the $\overline{\text{GPIOWR}}$ and $\overline{\text{GPMEMWR}}$ signals from the offset

Table 13-2 GP Bus Registers—Memory-Mapped (Continued)

Register	Mnemonic	MMCR Offset Address	Function
GP Write Offset	GPWROFF	C0Eh	Offset from the beginning of the bus cycle for GPIOWR and GPMEMWR
GP ALE Pulse Width	GPALEW	C0Fh	Width of the GPALE signal from the offset
GP ALE Offset	GPALEOFF	C10h	Offset from the beginning of the bus cycle for GPALE
PIO15–PIO0 Pin Function Select	PIOPFS15_0	C20h	PIO15–PIO0 or interface function select: GPIRQ10–GPIRQ8, GPDACK3–GPDACK0, GPDRQ3–GPDRQ3, GPTC, GPAEN, GPRDY, GPBHE, GPALE
PIO31–PIO16 Pin Function Select	PIOPFS31_16	C22h	PIO31–PIO16 or interface function select: RIN2, DCD2, DSR2, CTS2, GPCS0, GPMEMCS16, GPIOCS16, GPDBUFOE, GPIRQ7–GPIRQ0
Chip Select Pin Function Select	CSPFS	C24h	GPCS7–GPCS1 or alternate function select: TMROUT1–TMROUT0, TMRIN1–TMRIN0, PITGATE2, ROMCS2, ROMCS1
Reset Configuration	RESCFG	D72h	Control bit for GP bus reset (GPRESET)

13.5 OPERATION

The GP bus provides a simple interface to the integrated on-chip peripherals, as well as external peripherals. The GP bus operates at 33 MHz.

The GP bus controller provides one *fixed* timing set for the internal peripherals and one *programmable* timing set for the external peripherals.

Internal to the ÉlanSC520 microcontroller, the GP bus is used to provide a full complement of integrated peripherals such as a DMA controller, programmable interrupt controller PIC), programmable interval timer (PIT), UARTs, and real-time clock (RTC). The internal peripherals are designed to operate at the full clock rate of the GP bus. They can also be configured to operate in PC/AT-compatible configuration, but are generally not restricted to this configuration.

The GP bus interface can be programmed by software to control the interface timing between the GP bus and the external devices. The GP bus interface supports programmable timing, dynamic data width sizing, and cycle stretching to accommodate a wide variety of standard peripherals.

Eight chip selects are provided for external GP bus devices. They can be used for either memory or I/O accesses. These chip selects are asserted for Am5$_x$86 CPU accesses to the corresponding regions set up in the Programmable Address Region (PAR) registers.

Four external DMA channels provide fly-by DMA transfers between peripheral devices on the GP bus and system SDRAM.

GP bus accesses can be initiated only by the Am5$_x$86 CPU or by the integrated GP bus DMA controller. The devices on the GP bus are not cacheable from the Am5$_x$86 CPU's viewpoint, to enable a simple user view of devices (memory and peripherals) that are located on the GP bus.

The GP bus also provides an echo mode that is useful for debugging. If GP bus echo mode enabled, the internal GP bus cycle is echoed out on the external pins to enable visibility of internal cycles. Accesses to internal peripherals that are “echoed” out utilize the programmed timing set to ensure that there is no timing conflict with other external peripherals. Note that enabling echo mode does not affect the operation of GP-DMA accesses or GP bus external accesses.

13.5.1 Programmable Bus Interface Timing

The bus interface timing can be programmed for the following signals:

- Chip selects $\overline{\text{GPCS7}}$–$\overline{\text{GPCS0}}$
- Read strobes $\overline{\text{GPIORD}}$ and $\overline{\text{GPMEMRD}}$
- Write strobes $\overline{\text{GPIOWR}}$ and $\overline{\text{GPMEMWR}}$
- Address latch enable GPALE

For each of these signal types, the following parameters can be programmed:

- Offset from beginning of the bus cycle
- Pulse width from end of the offset
- Chip select recovery time

Figure 13-4 shows the shows the relationships between the various adjustable GP bus timing parameters. The actual time can be calculated with the following formula:

(REG_VAL + 1) * T_{CLK}

where:

REG_VAL = register content value

T_{CLK} = internal clock period

The minimum offset, pulse width and recovery time is 30 ns (for a 33.333-MHz crystal), resulting in a minimum bus cycle time of 90 ns. Since the offset, pulse width, and recovery parameters are each 8-bit values (maximum 255), the longest bus cycle in this case is 23 µs ($2^{(8\ \text{bits})}$ * 30 ns * 3 registers).

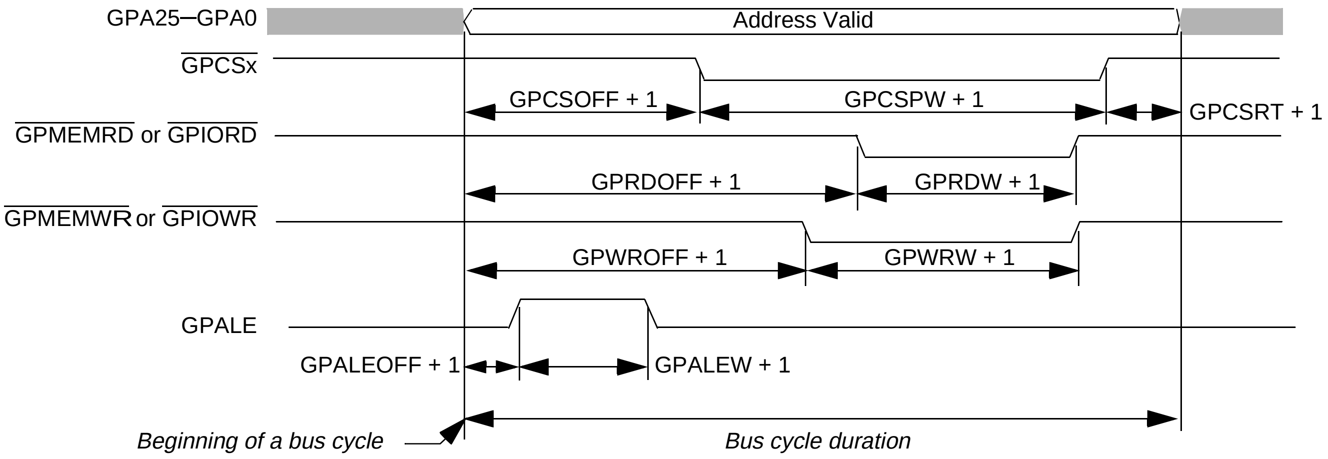
13.5.1.1 Timing Requirements

The programmed timing of the chip select determines the overall length of the GP bus cycle. Therefore, the timing parameters for the chip select must be appropriately programmed. This is required even if the external device does not require a connection to the $\overline{\text{GPCSx}}$ pin.

- To ensure that the command strobes (read or write) assert for the programmed time, the programmed Offset + Pulse Width + Recovery of the chip select must be programmed to be *longer* than the programmed Offset + Pulse Width of the command strobes.
- Similarly, to ensure that GPALE is asserted for the programmed time, the programmed Offset + Pulse Width + Recovery of the chip select must be programmed to be *longer* than the programmed Offset + Pulse Width of the GPALE.

Figure 13-4 on page 13-8 illustrates how the GP bus registers control this timing adjustment for GP bus signals.

Figure 13-4 GP Bus Timing Format



Notes:

1. *Timing parameter values are in units of one internal clock period.*
2. *Timing parameters in the diagram can be adjusted via the corresponding GP bus registers.*
3. *GPCSOFF + GPCSPW + GPCSRT must be greater than or equal to GPRDOFF + GPRDW, GPWROFF + GPWRW, or GPALEOFF + GPALEW.*
4. *The GPCSOFF, GPCSPW, and GPCSRT registers affect all $\overline{\text{GPCSx}}$ signals equally.*
5. *The abbreviations in the figure refer to these GP bus registers:*

Mnemonic	**Register**
GPCSRT	GP Chip Select Recovery Time
GPCSPW	GP Chip Select Pulse Width
GPCSOFF	GP Chip Select Offset
GPRDW	GP Read Pulse Width
GPRDOFF	GP Read Offset
GPWRW	GP Write Pulse Width
GPWROFF	GP Write Offset
GPALEW	GP ALE Pulse Width
GPALEOFF	GP ALE Offset

13.5.1.2 Using GPRDY with Programmable Timing

If the GPRDY signal is used, the bus cycle can be extended as long as required by the peripheral. GPRDY *cannot* be used to terminate any bus cycle earlier than programmed. More detailed information is provided in “Wait States” on page 13-20.

13.5.1.3 Using GP Bus Echo Mode with Programmable Timing

While GP bus echo mode is enabled, the system designer needs to ensure that the GP bus timing is not faster than that shown in Table 13-3. The minimum GP bus timing register values during the GP bus echo mode are shown in Table 13-3.

Table 13-3 GP Bus Echo Mode Minimum Timing

Signal Type	GPCSOFF, GPRDOFF, GPALEOFF (Offset) Register Value[1]	GPCSPW, GPRDW, GPALEW (Pulse Width) Register Value[1]	GPCSRT (Recovery Time) Register Value[1]
GP chip select	1	3	1
GP read	1	3	—
GP write	1	3	—
GPALE	0	0	—

Notes:
1. The actual time value is the register value plus 1. Times are in units of one internal clock period.

13.5.2 I/O-Mapped and Memory-Mapped Device Support

The GP bus controller supports any combination of 8-bit and 16-bit I/O and memory-mapped external devices.

- If the external device is an I/O-mapped device, $\overline{\text{GPIORD}}$ and $\overline{\text{GPIOWR}}$ are used to strobe the read and write accesses.
- If the external device is a memory-mapped device, $\overline{\text{GPMEMRD}}$ and $\overline{\text{GPMEMWR}}$ are used to strobe the read and write accesses.

To program I/O or memory-mapped address regions, see Chapter 4, “System Address Mapping”, and the examples in “External GP Bus Devices” on page 3-13.

13.5.3 Chip Select Qualification

All GP bus chip selects can be qualified with the command strobes, $\overline{\text{GPIORD}}$, $\overline{\text{GPIOWR}}$, $\overline{\text{GPMEMRD}}$, or $\overline{\text{GPMEMWR}}$, by programming the GP Chip Select Qualification (GPCSQUAL) register (MMCR offset C02h) and the Programmable Address Region (PAR) registers for memory or I/O device selection.

When chip select qualification is enabled, the internal chip selects are logically ANDed with one (or both) of these command strobes. If a single command is chosen for qualification, the corresponding chip select is not asserted for accesses of the other type. For example, if $\overline{\text{GPMEMWR}}$ is used to exclusively qualify a chip select, that chip select is not asserted for memory read accesses.

In a typical system environment, the command strobes are usually shorter than the chip selects, and, in such cases, the external chip selects have timing that is identical to the command strobes. Note that if the chip selects are internally qualified by commands, the timing relationships between the command and chip select assertion/deassertion cannot be guaranteed externally. For example, the chip select deassertion may lead the command deassertion.

The qualification feature is useful for interfacing with buffer chips and transceivers without requiring external gates or logic.

13.5.4 Data Sizing and Unaligned Accesses

The GP bus controller always operates in either 8-bit or 16-bit sizes. If the $\text{Am5}_{\text{X}}\text{86}$ CPU requests a 32-bit access from an 8-bit device or 16-bit device, the GP bus controller responds to the $\text{Am5}_{\text{X}}\text{86}$ CPU with $\overline{\text{bs8}}$, indicating 8-bit data width, or $\overline{\text{bs16}}$, indicating 16-bit data width, depending on the programming of the GP Chip Select Data Width (GPCSDW) register (MMCR offset C01h) and the state of the $\overline{\text{GPIOCS16}}$ and $\overline{\text{GPMEMCS16}}$ signals. The Am5$_X$86 CPU then generates multiple 8-bit or 16-bit bus cycles until all 32-bit data is accessed; thus, the size is transparent to software. This is true for read accesses and write accesses.

If the GP Chip Select Data Width (GPCSDW) register is programmed for 8-bit data width, assertion of external $\overline{\text{GPIOCS16}}$ (during an I/O access) or $\overline{\text{GPMEMCS16}}$ (during a memory access) overrides the data width specified in the GP Chip Select Data Width (GPCSDW) register, as discussed on page 13-19.

Unaligned address accesses (addresses that are not on the 16-bit address boundary) are supported through the Am5$_X$86 CPU. The Am5$_X$86 CPU breaks an unaligned address bus cycle into multiple bus cycles with appropriate byte enable signals ($\overline{\text{be3}}$–$\overline{\text{be0}}$). The GP bus controller simply takes one Am5$_X$86 CPU bus cycle at a time and generates one external bus cycle at a time.

13.5.5 Sharing the Address and Data Bus with the ROM/Flash Controller

A ROM device's data bus can be connected to either the GP bus data bus or the SDRAM data bus.

- When a ROM device is connected to the GP data bus, the ROM access shares both GPD15–GPD0 and GPA25–GPA0 with the GP bus.
- When a ROM device is connected to the SDRAM data bus, the ROM access shares only GPA25–GPA0 with the GP bus.

This does not cause bus contention, because only the Am5$_X$86 CPU can initiate an access to either ROM or to the GP bus. Since the Am5$_X$86 CPU can perform an access to only one controller at a time, no conflict is possible.

Note that the GP bus DMA controller can initiate an access on the GP bus. Since the GP bus DMA controller must already own the Am5$_X$86 CPU's bus before it can initiate an access, once again, there can be no conflict between bus cycles initiated by the GP bus DMA controller and ROM cycles initiated by the Am5$_X$86 CPU.

Note that the ROM devices are cacheable, but GP bus devices are noncacheable. This is because the ROM controller supports cacheability and has its own independent control signals (chip selects, read strobe, and write strobe).

13.5.6 GP Bus Echo Mode

In normal operation, the integrated peripheral accesses are not visible on the external pins. GP bus echo mode is provided to view accesses to the internal GP bus peripherals on the external pins. This feature aids in debugging system software and boot code. This applies to the integrated peripherals only (timers, GP-DMA controller, UARTs, SSI, RTC, etc.) and not to the memory or PCI bus controllers.

Accesses to internal peripherals that are “echoed” out utilize the programmable timing set to ensure that there is no timing conflict with other external peripherals. Typically, internal peripheral bus accesses are faster than external peripherals. Therefore, when using GP bus echo mode to debug the system, be aware that accesses to the integrated peripherals may be occurring at slower speeds to ensure compatibility with external devices, thus resulting in a slower system performance.

When GP bus echo mode is enabled, GPAEN is driven high during accesses from the Am5$_X$86 CPU to internal peripherals to prevent external devices from decoding (or responding to) these internal peripheral accesses. In normal operation (GP bus echo mode disabled), the GP bus controller never asserts GPAEN.

Note that accesses initiated by the GP bus DMA controller are not affected by enabling the GP bus echo mode, and therefore the GP bus DMA controller still asserts GPAEN as it does during normal operation.During an internal GPDMA access in GP bus echo mode, the external GP bus commands, $\overline{\mathrm{GPIORD}}$, $\overline{\mathrm{GPMEMRD}}$, $\overline{\mathrm{GPIOWR}}$, $\overline{\mathrm{GPMEMWR}}$, are not asserted. However, GPAEN is still asserted.

While GP bus echo mode is enabled, there are additional restrictions to the programmable timing parameters that must be taken into account. These are described in “Using GP Bus Echo Mode with Programmable Timing” on page 13-8.

13.5.7 DMA Interface

There are four DMA channels for external GP bus peripherals. The GPDRQ3–GPDRQ0 signals go directly to the GP-DMA controller, and their levels are programmable in the GP-DMA controller. All GP-DMA control signals and timing are generated by the GP-DMA controller, and the programmable timing in the GP bus controller does not affect the GP-DMA cycle timing. For more information, see Chapter 14, “GP Bus DMA Controller”.

13.5.8 Usage Scenarios

13.5.8.1 Compatibility with Common ISA Devices

The GP bus is compatible with most ISA devices, but the following ISA bus features are not supported.

- LA23–LA17 is supported through GPA23–GPA17, but note that because the Am5$_x$86 CPU itself does not support address pipelining, address pipelining is not supported on the GP bus.
- GPA25–GPA24 is added to increase the GP bus address space up to 64 Mbytes, instead of 16 Mbytes.
- External master access is not supported, and the ÉlanSC520 microcontroller is always the master on the GP bus (external masters can be accommodated by the PCI bus).
- $\overline{\mathrm{GPIOCS16}}$ and $\overline{\mathrm{GPMEMCS16}}$ do not cause the GP bus timings to change for the bus cycles during which these signals are asserted.
- IOCHRDY is supported via the GPRDY pin only as an input for the slave devices that require wait states. GPRDY as an output is not supported, since there is no external master support.
- $\overline{\mathrm{IOCHK}}$ is not supported, but a GPIRQx signal (mappable to a maskable or non-maskable interrupt) can be used to report errors.
- The $\overline{\mathrm{REFRESH}}$ pin is not supported, because the SDRAM refresh is not echoed out to the GP bus.
- $\overline{\mathrm{NOWS}}$ is not supported, due to the programmable interface timing on the GP bus.
- BCLK and OSC are not supported, because a typical ISA interface is asynchronous. External oscillators can be used, if needed.
- The GP bus provides programmable bus interface timing that can be configured to support most ISA bus devices. However, the GP bus does not support all legacy ISA timing. See the *Élan™SC520 Microcontroller Data Sheet*, order #22003, for information on the GP bus and GP-DMA timing supported by the ÉlanSC520 microcontroller.

Table 13-4 shows the cross-reference table of the ISA signals and the GP bus signals.

Table 13-4 Cross-Reference Table of ISA Signals and GP Bus Signals[1]

ISA Signal Name	GP Bus Signal Name
AEN	GPAEN
BALE	GPALE
BCLK	(Not Supported)
$\overline{\text{DACK}}$	$\overline{\text{GPDACK}}$
DRQ	GPDRQ
$\overline{\text{IOCHK}}$	Supported through GPIRQ
IOCHRDY	GPRDY
$\overline{\text{IOCS16}}$	$\overline{\text{GPIOCS16}}$
$\overline{\text{IOR}}$	$\overline{\text{GPIORD}}$
$\overline{\text{IOW}}$	$\overline{\text{GPIOWR}}$
IRQ	GPIRQ
LA23–LA17	GPA23–GPA17
$\overline{\text{MASTER}}$	(Not Supported)
$\overline{\text{MEMCS16}}$	$\overline{\text{GPMEMCS16}}$
$\overline{\text{MEMR}}$	$\overline{\text{GPMEMRD}}$
$\overline{\text{MEMW}}$	$\overline{\text{GPMEMWR}}$
OSC	(Not Supported)
$\overline{\text{REFRESH}}$	(Not Supported)
RSTDRV	GPRESET
SA19–SA0	GPA19–GPA0
$\overline{\text{SBHE}}$	$\overline{\text{GPBHE}}$
SD15–SD0	GPD15–GPD0
TC	GPTC
(Not Supported)	GPA25–GPA24

Notes:

[1] This table does not imply that the ÉlanSC520 microcontroller is fully compliant with all ISA timing specifications. See the Élan™SC520 Microcontroller Data Sheet, order #22003, for information on the GP bus and GP-DMA timing supported by the ÉlanSC520 microcontroller.

13.5.8.2 Interfacing with a Super I/O Controller

Figure 13-5 shows an example system diagram of the ÉlanSC520 microcontroller interfacing with a Super I/O controller. Figure 13-6 shows the interfacing timing example. In this example, the programmable interface timing registers can be programmed as shown in Table 13-5, using the equation from “Programmable Bus Interface Timing” on page 13-7:

Table 13-5 Example Super I/O Controller Interface Timing[1]

GP Bus Signal Type	Offset Register Value	Offset Time (ns)	Chip Require-ment (ns)	Pulse Width Register Value	Pulse Width (ns)	Chip Require-ment (ns)	Recovery Time Register Value	Recovery Timer (ns)	Chip Require-ment (ns)
GP chip selects	0	30	N/A	0	30	N/A	2	90	66
GP read	0	30	19	1	60	60	N/A	N/A	N/A
GP write	0	30	19	1	60	60	N/A	N/A	N/A
GPALE	0	30	N/A	0	30	N/A	N/A	N/A	N/A

Notes:

1. This example assumes that a 33.333-MHz crystal is being used in the system.

Note that the bus cycle can be stretched out by deasserting GPRDY; see “Wait States” on page 13-20 for more information.

Figure 13-5 Élan™SC520 Microcontroller Interfacing with a Super I/O Controller

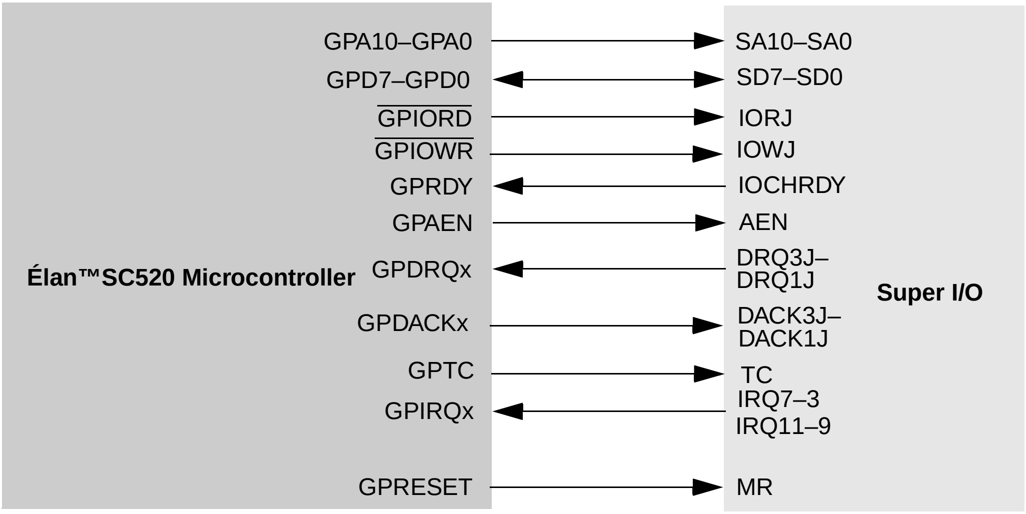
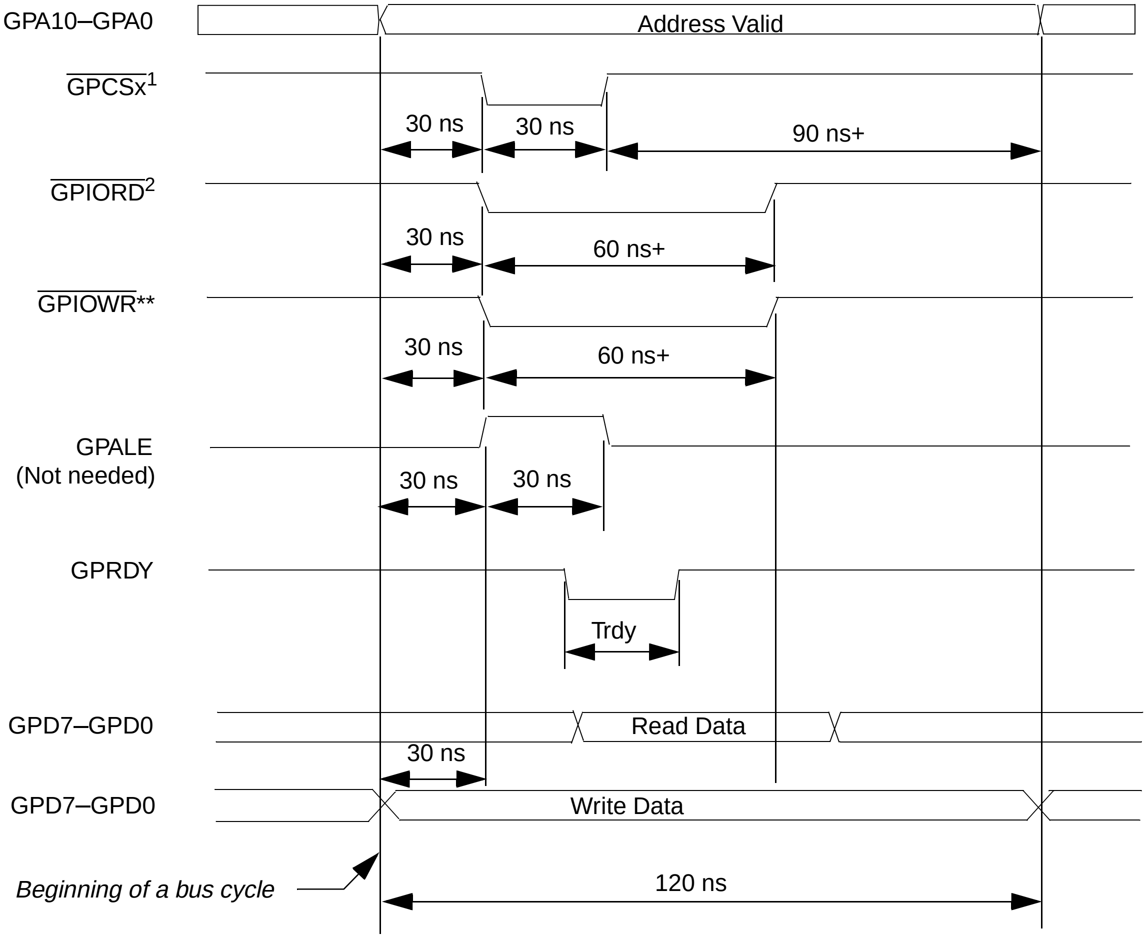


Figure 13-6 Timing Diagram of a Super I/O Interface



Notes:

1. *Although the chip selects are not used, the recovery time needs to be programmed.*
2. *GPIORD, GPIOWR, and the chip select recovery time are delayed when the GPRDY signal is deasserted.*
3. *This example assumes that a 33.333-MHz crystal is being used in the system.*

13.5.8.3 Interfacing with an AMD Enhanced Serial Communications Controller (8 MHz)

This slow version is depicted to illustrate an example of how the programmable timing can be used to function with various timing requirements. Figure 13-7 shows an example system diagram of the ÉlanSC520 microcontroller interfacing with an Am85C30 Enhanced Serial Communications controller. Table 13-6 and Figure 13-8 show the interfacing timing example. In this example, the programmable interface timing registers can be programmed using the equation from “Programmable Bus Interface Timing” on page 13-7.

Table 13-6 Example AMD Enhanced Serial Communications Controller Interface Timing[1]

GP Bus Signal Type	Offset Register Value	Offset Time (ns)	Chip Require-ment (ns)	Pulse Width Register Value	Pulse Width (ns)	Chip Require-ment (ns)	Recovery Time Register Value	Recovery Timer (ns)	Chip Require-ment (ns)
GP chip selects	2	90	0	4	150	150	0	30	3.5
GP read	2	90	70	4	150	150	N/A	N/A	N/A
GP write	2	90	70	4	150	150	N/A	N/A	N/A
GPALE	0	30	N/A	0	30	N/A	N/A	N/A	N/A

Notes:

1. This example assumes that a 33.333-MHz crystal is being used in the system.

Figure 13-7 Élan™SC520 Microcontroller Interfacing with an Am85C30

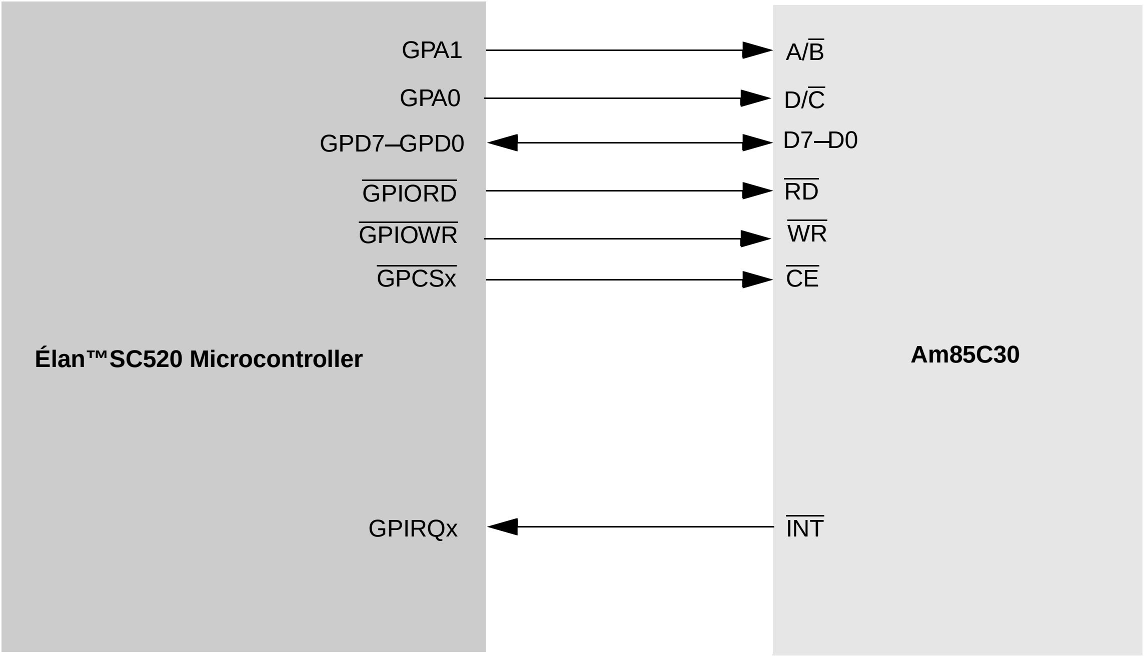
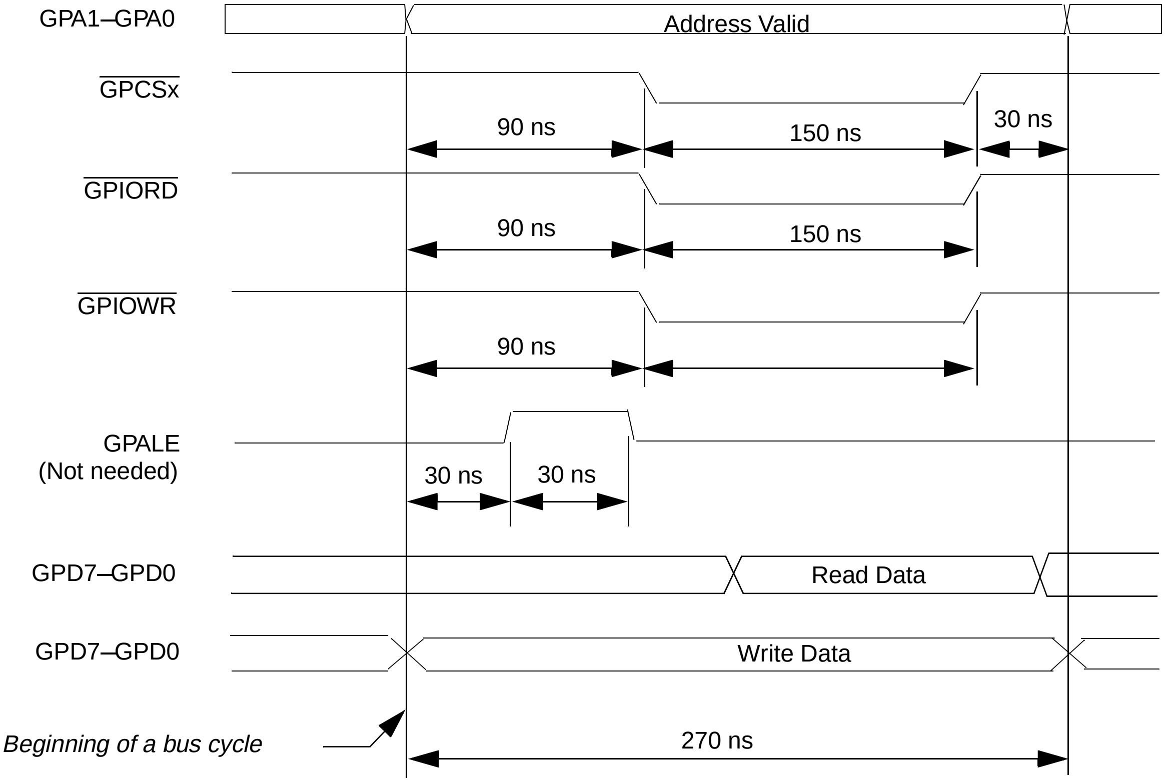


Figure 13-8 Timing Diagram of an Am85C30 Interface



Notes:

1. *This example assumes that a 33.333-MHz crystal is being used in the system.*

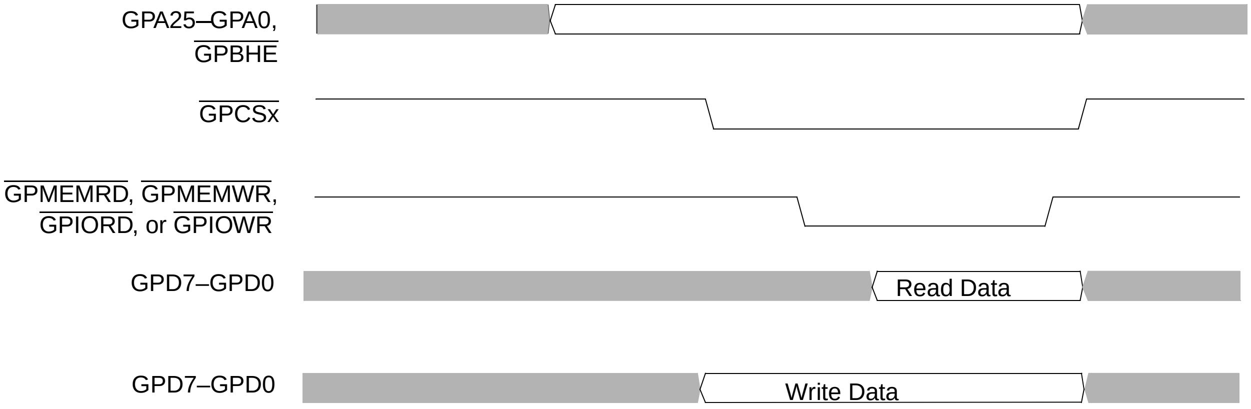
13.5.9 Bus Cycles

13.5.9.1 8-Bit Data Access of an 8-Bit I/O Device

During an 8-bit access to 8-bit I/O devices, GPD7–GPD0 is used to transfer data between the CPU and external devices. For an 8-bit memory-mapped I/O device, $\overline{\text{GPMEMWR}}$ and $\overline{\text{GPMEMRD}}$ are used instead of $\overline{\text{GPIOWR}}$ and $\overline{\text{GPIORD}}$.

Figure 13-9 shows the timing diagram of an 8-bit device access of an 8-bit I/O device.

Figure 13-9 8-Bit Data Access of an 8-Bit I/O Device

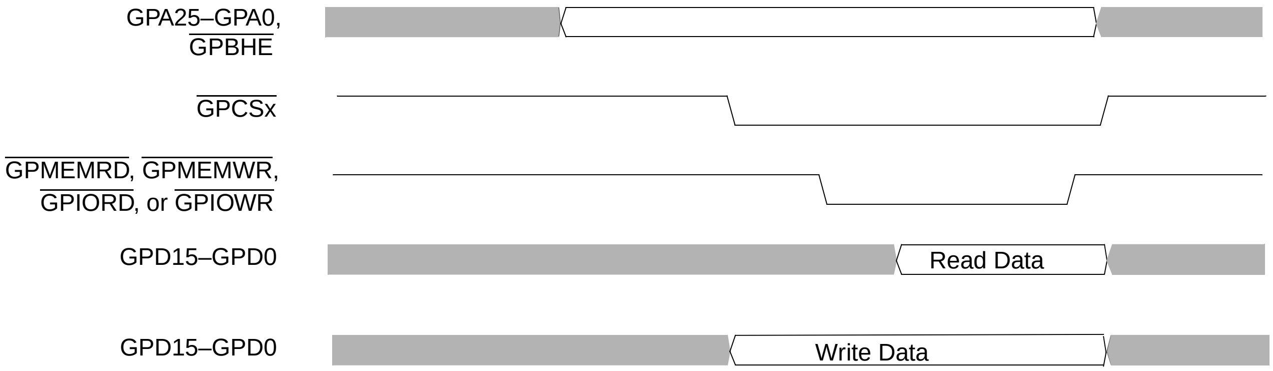


13.5.9.2 16-Bit Data Access of a 16-Bit I/O Device

A 16-bit data read/write access to 16-bit I/O devices are similar to the 8-bit I/O device accesses. In 16-bit accesses, all 16 bits of GPD are used. For memory-mapped I/O accesses, $\overline{\text{GPMEMRD}}$ and $\overline{\text{GPMEMWR}}$ are used instead of $\overline{\text{GPIORD}}$ and $\overline{\text{GPIOWR}}$.

Figure 13-10 shows the timing diagram of 16-bit accesses of a 16-bit I/O device.

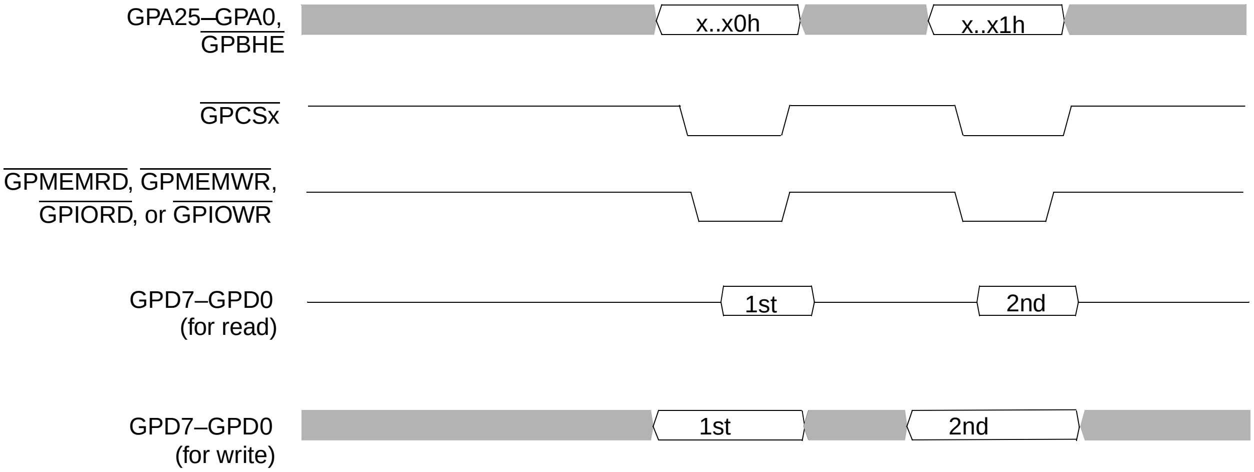
Figure 13-10 16-Bit Data Access of a 16-Bit I/O Device


GPA25–GPA0,
$\overline{\text{GPBHE}}$
$\overline{\text{GPCSx}}$
$\overline{\text{GPMEMRD}}$, $\overline{\text{GPMEMWR}}$,
$\overline{\text{GPIORD}}$, or $\overline{\text{GPIOWR}}$
GPD15–GPD0
Read Data
GPD15–GPD0
Write Data

13.5.9.3 16-Bit Data Access of an 8-Bit I/O Device

A 16-bit data access of an 8-bit I/O device requires two consecutive 8-bit data accesses of the I/O device, but the consecutive 8-bit data accesses are resolved by the Am5$_x$86 CPU transparent to software. For memory-mapped I/O accesses, $\overline{\text{GPMEMRD}}$ and $\overline{\text{GPMEMWR}}$ are used instead of $\overline{\text{GPIORD}}$ and $\overline{\text{GPIOWR}}$. When the Am5$_x$86 CPU requests a 16-bit data access, the GP bus controller responds to the Am5$_x$86 CPU with the $\overline{\text{bs8}}$ signal, indicating that the data width of the device is only 8 bits. The Am5$_x$86 CPU then generates two consecutive 8-bit bus cycles, and the 16-bit data access becomes two separate 8-bit data GP bus cycles. Figure 13-11 shows the timing diagram of a 16-bit access of an 8-bit I/O device.

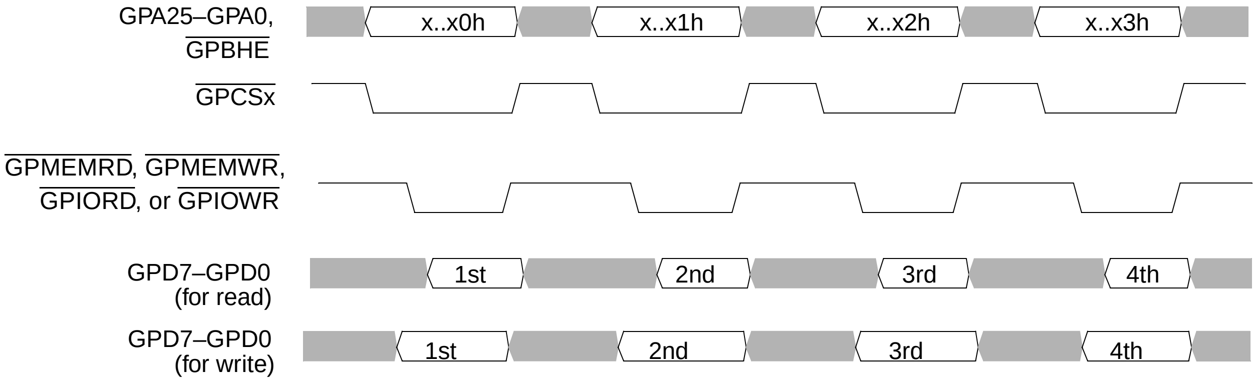
Figure 13-11 16-Bit Data Access of an 8-Bit I/O Device


GPA25–GPA0,
$\overline{\text{GPBHE}}$
x..x0h
x..x1h
$\overline{\text{GPCSx}}$
$\overline{\text{GPMEMRD}}$, $\overline{\text{GPMEMWR}}$,
$\overline{\text{GPIORD}}$, or $\overline{\text{GPIOWR}}$
GPD7–GPD0
(for read)
1st
2nd
GPD7–GPD0
(for write)
1st
2nd

13.5.9.4 32-Bit Data Access of an 8-Bit I/O Device

A 32-bit data access of an 8-bit I/O device requires four consecutive 8-bit data accesses of the 8-bit I/O device, but the consecutive 8-bit data accesses are resolved by the Am5$_x$86 CPU transparent to software. For memory-mapped I/O accesses, $\overline{\text{GPMEMRD}}$ and $\overline{\text{GPMEMWR}}$ are used instead of $\overline{\text{GPIORD}}$ and $\overline{\text{GPIOWR}}$. When the Am5$_x$86 CPU requests a 32-bit data access, the GP bus controller responds to the Am5$_x$86 CPU with the $\overline{\text{bs8}}$ signal, indicating that data width of the device is only 8 bits. The Am5$_x$86 CPU then generates four consecutive 8-bit bus cycles, and the 32-bit data access becomes four separate 8-bit data GP bus cycles. Figure 13-12 shows the timing diagram of a 32-bit access of an 8-bit I/O device.

Figure 13-12 32-Bit Data Access of an 8-Bit I/O Device



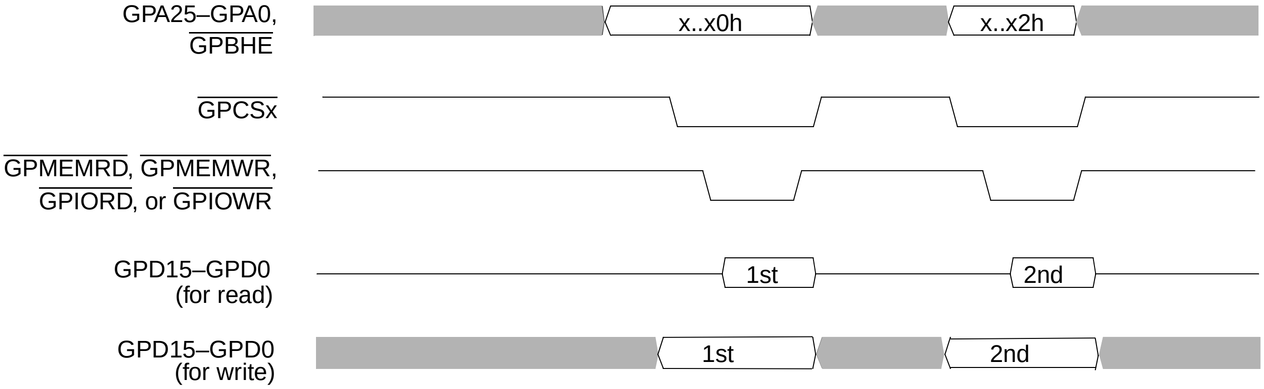
13.5.9.5 32-Bit Data Access of a 16-Bit I/O Device

A 32-bit data access of a 16-bit I/O device requires two consecutive 16-bit accesses of the device, but the consecutive 16-bit data accesses are resolved by the Am5$_x$86 CPU transparent to software. For memory-mapped I/O accesses, $\overline{\text{GPMEMRD}}$ and $\overline{\text{GPMEMWR}}$ are used instead of $\overline{\text{GPIORD}}$ and $\overline{\text{GPIOWR}}$.

When the Am5$_x$86 CPU requests a 32-bit data access, the GP bus controller responds to the Am5$_x$86 CPU with the $\overline{\text{bs16}}$ signal, indicating that the data width of the device is only 16 bits. The Am5$_x$86 CPU then generates two consecutive 16-bit bus cycles, and the 32-bit data access becomes two separate 16-bit cycles on the GP bus.

Figure 13-13 shows the timing diagram of a 32-bit access of a 16-bit I/O device.

Figure 13-13 32-Bit Data Access of a 16-Bit I/O Device



13.5.9.6 8-Bit Data Access of a 16-Bit I/O Device

The GPA0 and GPBHE signals are required to determine which byte of a 16-bit peripheral is accessed during byte read or write cycles. Table 13-7 describes how to determine which byte is accessed.

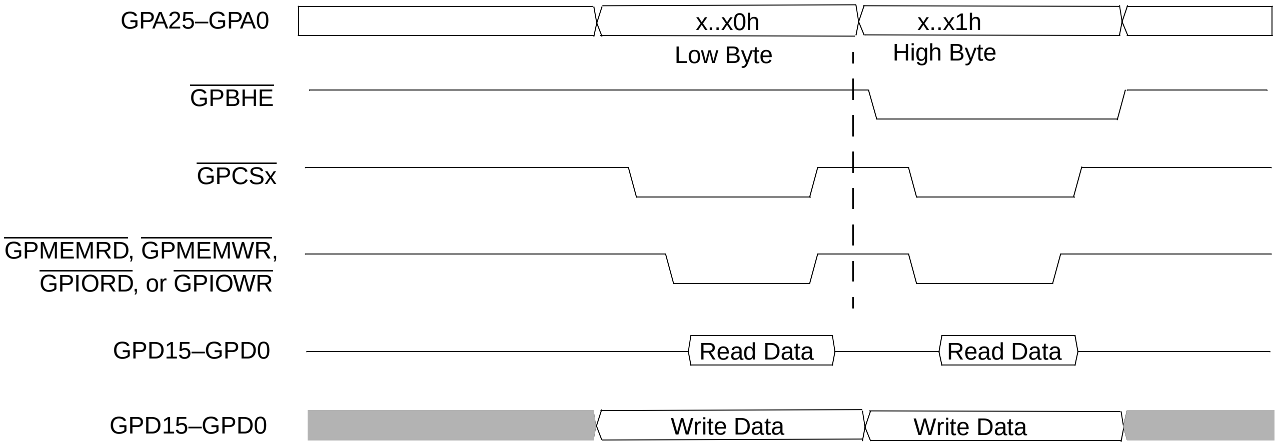
Table 13-7 Differentiating Upper/Lower Byte Access of 16-Bit Devices

GPBHE	GPA0	Cycle Description
0	0	16-bit access of 16-bit device
0	1	Upper byte access of 16-bit device
1	0	Lower byte access of either 8-bit or 16-bit device
1	1	Upper byte access of 8-bit device

For memory-mapped I/O accesses, GPMEMRD and GPMEMWR are used instead of GPIORD and GPIOWR.

Figure 13-14 shows the timing diagram of an 8-bit access of a 16-bit I/O device.

Figure 13-14 8-Bit Data Access of a 16-Bit I/O Device



13.5.9.7 GPIOCS16 and GPMEMCS16 Timing

The GP bus controller provides two methods for defining the data bus width.

- The GP Chip Select Data Width (GPCSDW) register (MMCR offset C01h) allows each chip select to be individually programmed for 8-bit or 16-bit data bus width.
- The GP bus controller also supports dynamic bus sizing through the GPIOCS16 and GPMEMCS16 pins. These pins can be used to override the programming of the data width for the current access, as described in Table 13-8.
  - The GPIOCS16 and GPMEMCS16 pins can be asserted after the address or chip select is valid and deasserted after the address or chip select invalid.
  - If one of these pins is asserted by the external devices, the GP bus controller asserts bs16 to the Am5$_x$86 CPU.
  - Assertion of these signals does not affect the programmable interface timing.

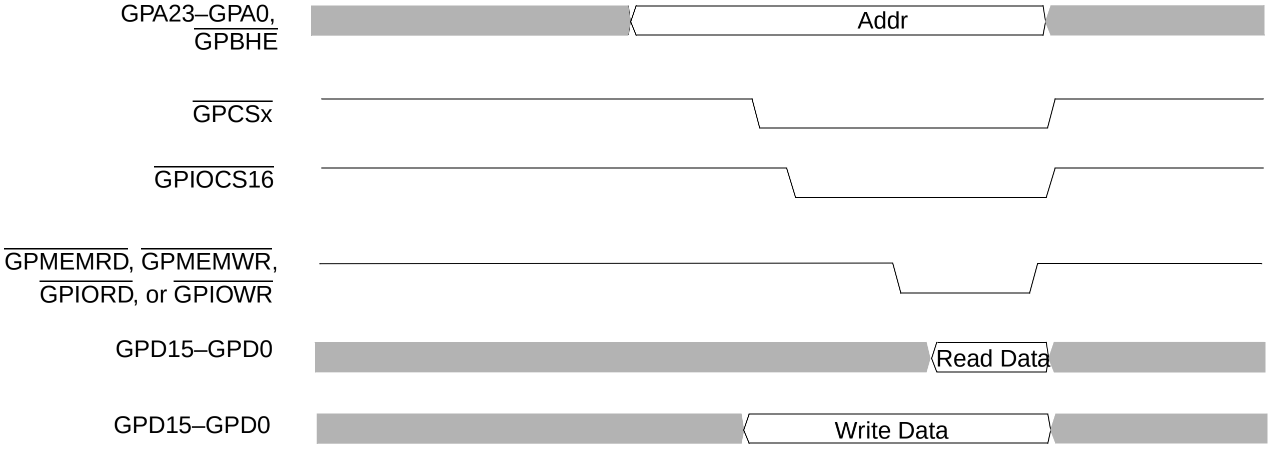
The latest assertion time for these two signals is the same as the timing for the GPRDY deassertion time (see “GPRDY Recognition” on page 13-20).

Table 13-8 Dynamic Bus Sizing Override of Programmed Data Width

GP Chip Select Data Width (GPCSDW) Register Setting	GPIOCS16 GPMEMCS16 Assertion	Resultant Bus Size
8-bit	Deasserted	8-bit
8-bit	Asserted	16-bit
16-bit	Deasserted	16-bit
16-bit	Asserted	16-bit

Figure 13-15 shows the GPIOCS16 timing for a 16-bit access and an 8-bit access.

Figure 13-15 16-Bit Access of a 16-Bit I/O Device



13.5.9.8 Wait States

The ÉlanSC520 microcontroller provides two ways to insert wait states in a GP bus cycle.

- The user can program the programmable interface timing registers to delay the timing of GPIORD, GPMEMRD, GPIOWR, or GPMEMWR for the required number of wait state cycles.
- GPRDY can also be used to insert wait states dynamically on a cycle basis.

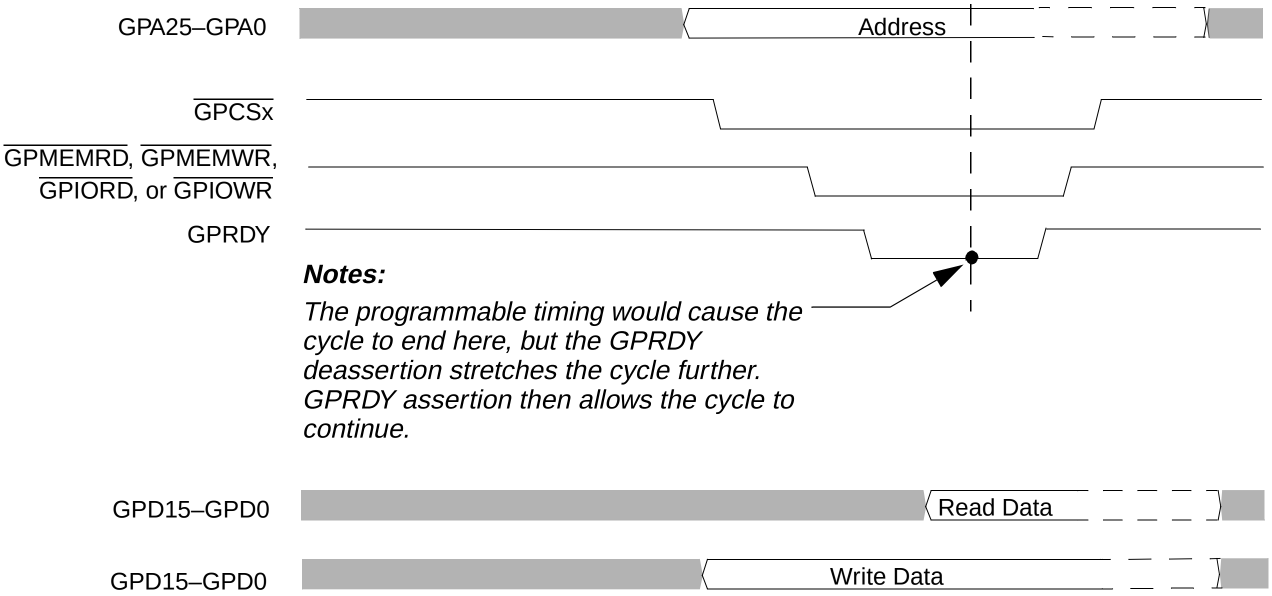
GPRDY can only be used to stretch GP bus cycles; it cannot be used to provide early termination of the cycle. The control signals are always asserted for a minimum of the entire period, as programmed in the timing control registers. Then, the additional delay can be inserted by the deassertion of GPRDY.

Figure 13-16 shows the timing of GPRDY.

13.5.9.8.1 GPRDY Recognition

Assuming a 33.333-MHz crystal, the GPRDY pin must be deasserted a minimum of 45 ns before the programmed deassertion of the command strobes and must have a minimum deassertion (Low) width of 30 ns to insert a wait state into a GP bus cycle. Additional wait states are inserted by extending the time in which the GPRDY pin is held deasserted. The command strobes will be deasserted after the GPRDY signal is internally synchronized and sampled asserted by the 33-MHz clock and after the programmed pulse width value for the strobe has expired.

Figure 13-16 GPRDY Timing



Notes:

The programmable timing would cause the cycle to end here, but the GPRDY deassertion stretches the cycle further. GPRDY assertion then allows the cycle to continue.

13.5.10 Interrupts

External devices that assert interrupts use the GPIRQ10–GPIRQ0 signals for this purpose. The GPIRQx interrupt signals bypass the GP bus controller and are routed to the programmable interrupt controller (PIC). See Chapter 15, “Programmable Interrupt Controller”, for more information.

13.5.11 Latency

13.5.11.1 8/16-Bit GP Bus Width

Due to the smaller data width of the GP bus, 32-bit accesses from the Am5$_X$86 CPU are broken up into separate 8-bit or 16-bit GP bus cycles. During this time, no other Am5$_X$86 CPU bus cycle can be generated, and neither the GP-DMA or an external PCI bus master can access SDRAM.

13.5.11.2 Slow GP Bus Cycles

If the interface timing is programmed to have slow GP bus cycles or if GPRDY is used to stretch cycles for long periods of time, the system performance can be affected because the CPU bus is monopolized.

Note: *Very long GP bus cycles can cause the PCI host bridge target controller to violate the 10 µs memory write maximum completion time limit set in the PCI Local Bus Specification, Revision 2.2. In PCI bus 2.2-compliant designs, software must limit the length of GP bus cycles and GP-DMA demand- or block-mode transfers.*

13.5.11.3 Noncacheable GP Bus

All GP bus accesses are noncacheable. Therefore, code execution out of this bus is not recommended.

13.6 INITIALIZATION

The GP bus controller is reset by a system reset. The internal GP bus is enabled, as are holes in the lower 1-Kbyte of I/O space; however, no chip selects are enabled. The external GP bus is disabled until the Programmable Address Region (PAR) registers are initialized.

GP bus reset can be generated via a system reset or software write. Writing a 1 to the GP_RST bit in the Reset Configuration (RESCFG) register (MMCR offset D72h) asserts the GPRESET pin. Clearing this bit to 0 deasserts the GPRESET pin. The GPRESET pin is only used for external GP bus peripherals. When this signal is asserted, all devices connected to the GP bus should re-initialize to their reset state.

To enable the GP bus controller:

1. Configure the address decoding region for each chip select in the PAR registers.
2. Configure the external chip select pins in the Chip Select Pin Function Select (CSPFS) register (MMCR offset C24h).
3. Configure the external GP bus timing in the programmable interface timing registers, as described in this chapter.
4. Configure the data width of each chip select in the GP Chip Select Data Width (GPCSDW) register (MMCR offset C01h).
5. Optionally, program the GP Chip Select Qualification (GPCSQUAL) register (MMCR offset C02h) to qualify the chip select with the read or write strobes, if needed.
6. Optionally, program the GP Echo Mode (GPECHO) register (MMCR offset C00h) to enable the GP bus echo mode, if needed.

CHAPTER

14 GP BUS DMA CONTROLLER

14.1 OVERVIEW

The ÉlanSC520 microcontroller includes an integrated GP bus DMA (GP-DMA) controller. The GP-DMA controller is designed to transfer data between external GP bus peripherals and SDRAM. Transfers between the internal UART serial ports and SDRAM are also supported. Throughout this document, the term *GP-DMA* refers to a DMA transaction on the GP bus.

Features of the GP bus DMA controller include:

- Fly-by transfers between GP bus peripherals and SDRAM
- Support for up to seven DMA request channels (with a maximum of four external requests)
- Two internal UART serial ports can initiate GP-DMA transfers
- GP-DMA controller can address all of the system SDRAM
- In enhanced GP-DMA mode:
  - Four channels are individually configurable for either 8 or 16 bits.
  - Maximum transfer count is 16 M (16,777,216) transfers (using 24-bit count register).
  - Channel widths default to PC/AT-compatible mode (three 16-bit, and four 8-bit).
  - Buffer chaining capability
- Variable clock modes: 4, 8, and 16 MHz
- Transfers to and from SDRAM only. No transfers are possible to PCI, ROM, or peer GP bus devices when using the GP-DMA controller.

Note: *The GP bus DMA controller is capable of supporting most ISA DMA applications and devices. However, not all of the legacy ISA timings are supported. See the Élan™SC520 Microcontroller Data Sheet, order #22003, for information on the GP bus and GP-DMA timing supported by the ÉlanSC520 microcontroller.*

14.2 BLOCK DIAGRAM

The GP-DMA controller consists of two DMA cores: the slave core and the master core.

- The slave core has four 8-bit channels by default: 0, 1, 2, and 3.
- The master core has three 16-bit channels by default: 5, 6, and 7.
- Channel 4 must be programmed to cascade mode and must be unmasked if any of the 8-bit channels 0–3 are to be used.
- In enhanced GP-DMA mode, Channels 3, 5, 6, and 7 are programmable to support either 8-bit or 16-bit mode.

Figure 14-1 shows a block diagram of the GP-DMA controller. Figure 14-2 shows how the master and slave cores are connected.

Figure 14-1 GP-DMA Controller Block Diagram

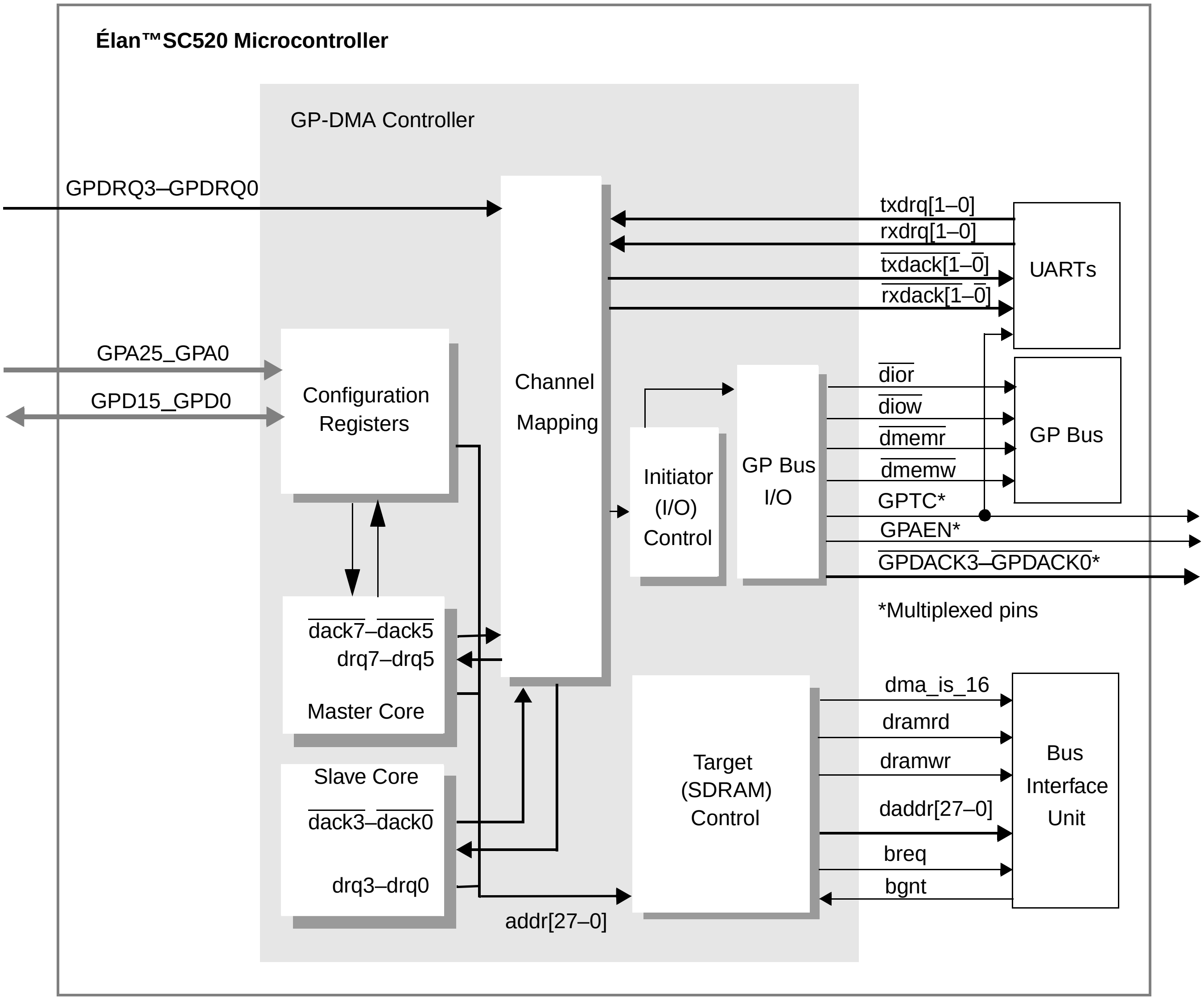
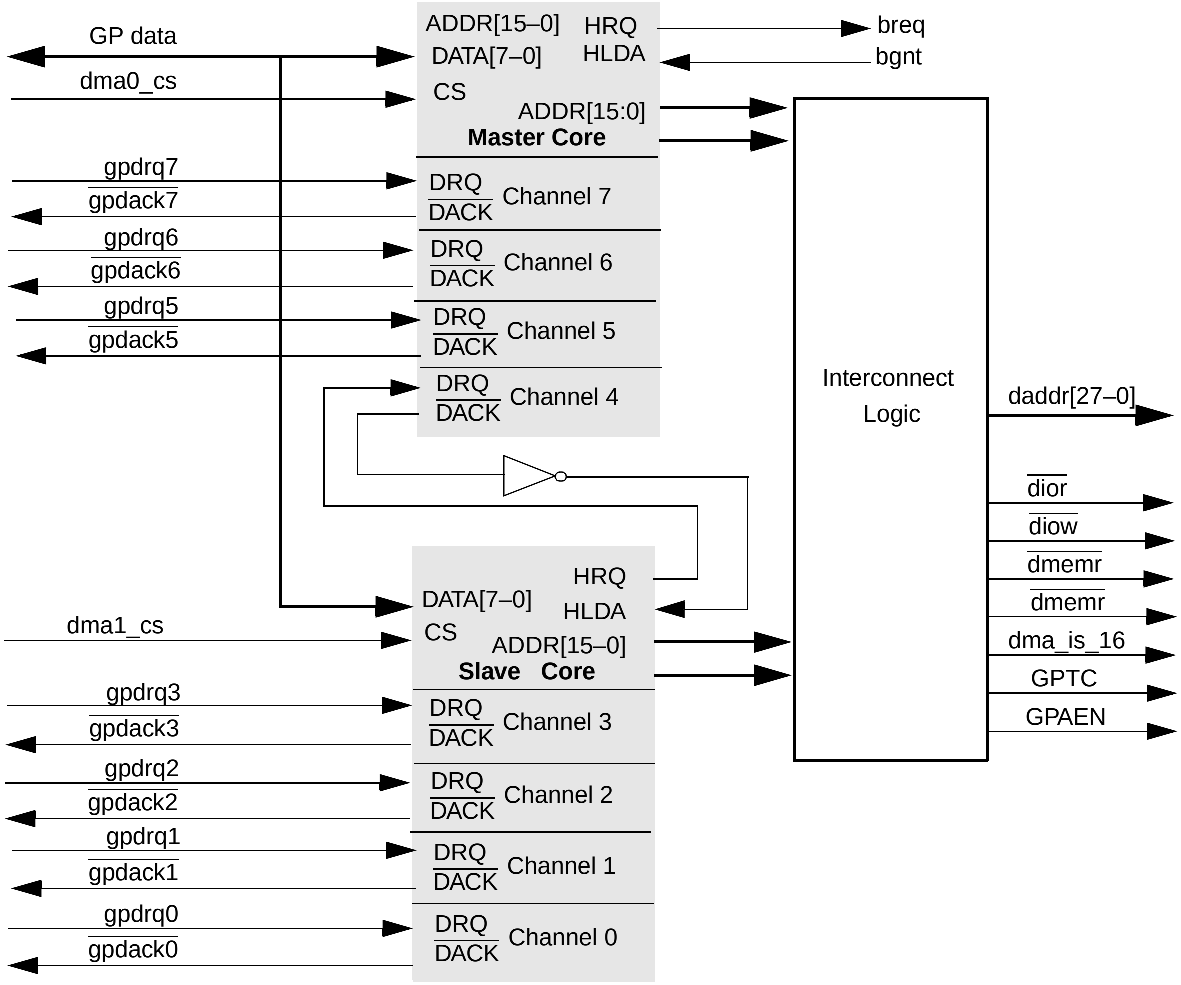


Figure 14-2 Master and Slave Core Cascading Diagram



14.3 SYSTEM DESIGN

Table 14-1 shows GP-DMA signals shared with other interfaces. When enabled, the multiplexed signals shown in Table 14-1 either disable or alter any other function that uses the same pin.

The GPDRQx and GPDACKx signals have programmable polarities. The default polarity is compatible to the ISA convention.

Since the GP-DMA controller does not generate an interrupt at the end of the transfer, system designers can externally connect GPTC to any GPIRQx to trigger an interrupt. Note that qualifying GPTC with a specific GPDACKx signal provides a more specific interrupt.

For an application that requires a DMA transfer every fixed interval of time, a timer output (TMROUT1 or TMROUT0) can be connected to the GPDRQx pin.

See the *Élan™SC520 Microcontroller Data Sheet*, order #22003, for timing tables and additional timing diagrams.

Table 14-1 GP-DMA Signals Shared with Other Interfaces

PIO (Default) Signal	Interface Function	Control Bit	Register
PIO12	GPDACK0	PIO12_FNC	PIO15–PIO0 Pin Function Select (PIOPFS15_0) register (MMCR offset C20h)
PIO11	GPDACK1	PIO11_FNC	
PIO10	GPDACK2	PIO10_FNC	
PIO9	GPDACK3	PIO9_FNC	
PIO8	GPDRQ0	PIO8_FNC	
PIO7	GPDRQ1	PIO7_FNC	
PIO6	GPDRQ2	PIO6_FNC	
PIO5	GPDRQ3	PIO5_FNC	
PIO4	GPTC	PIO4_FNC	
PIO3	GPAEN	PIO3_FNC	

14.4 REGISTERS

The GP bus DMA (GP-DMA) controller is configured using memory-mapped registers and direct-mapped registers.

14.4.1 Memory-Mapped Registers

A summary listing of the MMCR registers used to configure the GP-DMA controller is shown in Table 14-2. These registers provide functionality beyond the PC/AT compatibility, such as the extended page registers, the features in the enhanced GP-DMA mode, and the ability to route external GPDRQx and GPDACKx signals to a specific channel of the GP-DMA controller.

Table 14-2 GP-DMA Controller Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
PIO15–PIO0 Pin Function Select	PIOPFS15_0	C20h	PIO or interface function select: GPDACK3–GPDACK0, GPDRQ3–GPDRQ3, GPTC, GPAEN
DMA Buffer Chaining Interrupt Mapping	DMABCINTMAP	D40h	GP-DMA buffer chaining interrupt mapping
GP-DMA Control	GPDMACTL	D80h	GP-DMA enhanced mode enable, channel size, clock mode
GP-DMA Memory-Mapped I/O	GPDMAMMIO	D81h	I/O or memory-mapped I/O channel configuration
GP-DMA Resource Channel Map A	GPDMAEXTCHMAPA	D82h	Channel mapping for GPDRQ3–GPDRQ0
GP-DMA Resource Channel Map B	GPDMAEXTCHMAPB	D84h	Channel mapping for internal serial port GP-DMA requests
GP-DMA Channel 0 Extended Page	GPDMAEXTPG0	D86h	Bits 27–24 of the memory address for Channel 0

Table 14-2 GP-DMA Controller Registers—Memory-Mapped (Continued)

Register	Mnemonic	MMCR Offset Address	Function
GP-DMA Channel 1 Extended Page	GPDMAEXTPG1	D87h	Bits 27–24 of the memory address for Channel 1
GP-DMA Channel 2 Extended Page	GPDMAEXTPG2	D88h	Bits 27–24 of the memory address for Channel 2
GP-DMA Channel 3 Extended Page	GPDMAEXTPG3	D89h	Bits 27–24 of the memory address for Channel 3
GP-DMA Channel 5 Extended Page	GPDMAEXTPG5	D8Ah	Bits 27–24 of the memory address for Channel 5
GP-DMA Channel 6 Extended Page	GPDMAEXTPG6	D8Bh	Bits 27–24 of the memory address for Channel 6
GP-DMA Channel 7 Extended Page	GPDMAEXTPG7	D8Ch	Bits 27–24 of the memory address for Channel 7
GP-DMA Channel 3 Extended Transfer Count	GPDMAEXTTC3	D90h	Bits 23–16 of Channel 3 transfer count value (enhanced GP-DMA mode)
GP-DMA Channel 5 Extended Transfer Count	GPDMAEXTTC5	D91h	Bits 23–16 of Channel 5 transfer count value (enhanced GP-DMA mode)
GP-DMA Channel 6 Extended Transfer Count	GPDMAEXTTC6	D92h	Bits 23–16 of Channel 6 transfer count value (enhanced GP-DMA mode)
GP-DMA Channel 7 Extended Transfer Count	GPDMAEXTTC7	D93h	Bits 23–16 of Channel 7 transfer count value (enhanced GP-DMA mode)
Buffer Chaining Control	GPDMABCCTL	D98h	Buffer chaining enables for channels 7, 6, 5, and 3
Buffer Chaining Status	GPDMABCSTA	D99h	Buffer chaining status for channels 7, 6, 5, and 3
Buffer Chaining Interrupt Enable	GPDMABSINTENB	D9Ah	Buffer chaining interrupt enables for channels 7, 6, 5, and 3
Buffer Chaining Valid	GPDMABCVAL	D9Bh	Valid buffer of the buffer chaining operation
GP-DMA Channel 3 Next Address Low	GPDMANXTADDL3	DA0h	Address bits 0–15 of the next data buffer in memory used with Channel 3 (enhanced GP-DMA mode)
GP-DMA Channel 3 Next Address High	GPDMANXTADDH3	DA2h	Address bits 16–27 of the next data buffer in memory used with Channel 3 (enhanced GP-DMA mode)
GP-DMA Channel 5 Next Address Low	GPDMANXTADDL5	DA4h	Address bits 0–15 of the next data buffer in memory used with Channel 5 (enhanced GP-DMA mode)
GP-DMA Channel 5 Next Address High	GPDMANXTADDH5	DA6h	Address bits 16–27 of the next data buffer in memory used with Channel 5 (enhanced GP-DMA mode)

Table 14-2 GP-DMA Controller Registers—Memory-Mapped (Continued)

Register	Mnemonic	MMCR Offset Address	Function
GP-DMA Channel 6 Next Address Low	GPDMANXTADDL6	DA8h	Address bits 0–15 of the next data buffer in memory used with Channel 6 (enhanced GP-DMA mode)
GP-DMA Channel 6 Next Address High	GPDMANXTADDH6	DAAh	Address bits 16–27 of the next data buffer in memory used with Channel 6 (enhanced GP-DMA mode)
GP-DMA Channel 7 Next Address Low	GPDMANXTADDL7	DACh	Address bits 0–15 of the next data buffer in memory used with Channel 7 (enhanced GP-DMA mode)
GP-DMA Channel 7 Next Address High	GPDMANXTADDH7	DAEh	Address bits 16–27 of the next data buffer in memory used with Channel 7 (enhanced GP-DMA mode)
GP-DMA Channel 3 Next Transfer Count Low	GPDMANXTTCL3	DB0h	Bits 0–15 of the next transfer count for Channel 3 when using buffer chaining (enhanced GP-DMA mode)
GP-DMA Channel 3 Next Transfer Count High	GPDMANXTTCH3	DB2h	Bits 16–23 of the next transfer count for Channel 3 when using buffer chaining (enhanced GP-DMA mode)
GP-DMA Channel 5 Next Transfer Count Low	GPDMANXTTCL5	DB4h	Bits 0–15 of the next transfer count for Channel 5 when using buffer chaining (enhanced GP-DMA mode)
GP-DMA Channel 5 Next Transfer Count High	GPDMANXTTCH5	DB6h	Bits 16–23 of the next transfer count for Channel 5 when using buffer chaining (enhanced GP-DMA mode)
GP-DMA Channel 6 Next Transfer Count Low	GPDMANXTTCL6	DB8h	Bits 0–15 of the next transfer count for Channel 6 when using buffer chaining (enhanced GP-DMA mode)
GP-DMA Channel 6 Next Transfer Count High	GPDMANXTTCH6	DBAh	Bits 16–23 of the next transfer count for Channel 6 when using buffer chaining (enhanced GP-DMA mode)
GP-DMA Channel 7 Next Transfer Count Low	GPDMANXTTCL7	DBCh	Bits 0–15 of the next transfer count for Channel 7 when using buffer chaining (enhanced GP-DMA mode)
GP-DMA Channel 7 Next Transfer Count High	GPDMANXTTCH7	DBEh	Bits 16–23 of the next transfer count for Channel 7 when using buffer chaining (enhanced GP-DMA mode)

14.4.2 Direct-Mapped Registers

There are seven DMA channels in the GP-DMA controller. Table 14-3 shows the direct-mapped I/O registers that are available for each of the seven channels.

There are two DMA cores in the GP-DMA controller that are cascaded to provide the seven DMA channels. The cores are referred to as *master* and *slave*. Table 14-3 includes the set of the direct-mapped registers available in each of two cores. These registers program the function of the master or slave core.

In addition to the registers used to control GP-DMA, there is a set of general-purpose registers. These registers are decoded in the same chip select region with the page registers.

Table 14-3 GP-DMA Controller Registers—Direct-Mapped

Register	Mnemonic	I/O Address	Function
Registers for Each Channel			
Channel 0 Memory Address Channel 1 Memory Address Channel 2 Memory Address Channel 3 Memory Address Channel 4 Memory Address Channel 5 Memory Address Channel 6 Memory Address Channel 7 Memory Address	GPDMA0MAR GPDMA1MAR GPDMA2MAR GPDMA3MAR GPDMA4MAR GPDMA5MAR GPDMA6MAR GPDMA7MAR	0000h 0002h 0004h 0006h 00C0h 00C4h 00C8h 00CCh	Memory address bits 15–0 during GP-DMA transfers
Channel 0 Transfer Count Channel 1 Transfer Count Channel 2 Transfer Count Channel 3 Transfer Count Channel 4 Transfer Count Channel 5 Transfer Count Channel 6 Transfer Count Channel 7 Transfer Count	GPDMA0TC GPDMA1TC GPDMA2TC GPDMA3TC GPDMA4TC GPDMA5TC GPDMA6TC GPDMA7TC	0001h 0003h 0005h 0007h 00C2h 00C6h 00CAh 00CEh	Bits 15–0 of the transfer count for the GP-DMA transactions
Channel 2 Page Channel 3 Page Channel 1 Page Channel 0 Page Channel 6 Page Channel 7 Page Channel 5 Page	GPDMA2PG GPDMA3PG GPDMA1PG GPDMA0PG GPDMA6PG GPDMA7PG GPDMA5PG	0081h 0082h 0083h 0087h 0089h 008Ah 008Bh	Memory address bits 23–16 or 23–17 during GP-DMA transfers
Registers for Each DMA Core (Master and Slave)			
Master DMA Channel 4–7 Status Slave DMA Channel 0–3 Status	MSTDMASTA SLDMASTA	00D0h 0008h	GP-DMA request status and terminal count condition for each channel.
Master DMA Channel 4–7 Control Slave DMA Channel 0–3 Control	MSTDMACTL SLDMACTL	00D0h 0008h	DMA controller enable, arbitration mode, and timing control
Master Software DRQ(n) Request Slave Software DRQ(n) Request	MSTDMASWREQ SLDMASWREQ	00D2h 0009h	Software GP-DMA request initiated to a specific channel
Master DMA Channel 4–7 Mask Slave DMA Channel 0–3 Mask	MSTDMAMSK SLDMAMSK	00D4h 000Ah	GP-DMA channel mask
Master DMA Channel 4–7 Mode Slave DMA Channel 0–3 Mode	MSTLDMAMODE SLDMAMODE	00D6h 000Bh	Transfer mode, transfer type, automatic initialization, and address increment mode for each channel
Master DMA Clear Byte Pointer Slave DMA Clear Byte Pointer	MSTDMACBP SLDMACBP	00D8h 000Ch	Pointer to which byte will be accessed in the 16-bit GP-DMA registers

Table 14-3 GP-DMA Controller Registers—Direct-Mapped (Continued)

Register	Mnemonic	I/O Address	Function
Master DMA Controller Reset Slave DMA Controller Reset	MSTDMARST SLDMARST	00DAh 000Dh	GP-DMA controller reset
Master DMA Controller Temporary Slave DMA Controller Temporary	MSTDMATMP SLDMATMP	00DAh 000Dh	Preserves PC/AT compatibility
Master DMA Mask Reset Slave DMA Mask Reset	MSTDMAMSKRST SLDMAMSKRST	00DCh 000Eh	Mask register reset to activate the associated GP-DMA channels
Master DMA General Mask Slave DMA General Mask	MSTDMAGENMSK SLDMAGENMSK	00DEh 000Fh	GP-DMA channel masks
General-Purpose Registers			
General Registers	GPDMAGR0 GPDMAGR1 GPDMAGR2 GPDMAGR3 GPDMAGR4 GPDMAGR5 GPDMAGR6 GPDMAGR7 GPDMAGR8	0080h, 0084h–0086h, 0088h, 008Ch–008Fh	General-purpose R/W registers

14.5 OPERATION

The GP-DMA controller on the ÉlanSC520 microcontroller supports the following features.

- Only *fly-by* GP-DMA transfers are supported. A fly-by transfer is a transfer in which the data is moved from an I/O device or a memory-mapped I/O device to SDRAM (GP-DMA write), or from SDRAM to an I/O device or a memory-mapped I/O device (GP-DMA read) in a single transaction.
- Memory-to-memory (i.e., SDRAM-to-SDRAM) and I/O-to-I/O (peer-to-peer on the GP bus) transfers are not supported.
- Transfer modes supported: single, block, and demand
- Transfer types supported: read, write, and verify

14.5.1 GP-DMA Transfers

Because the ÉlanSC520 microcontroller also supports the standard PC/AT system architecture, the method for DMA transfer complies with the Industry Standard Architecture (ISA) specifications. The default polarities of GPDRQx and $\overline{\text{GPDACKx}}$ are active High and Low respectively, but they can be programmed differently.

The following general rules apply to GP-DMA transfers on the ÉlanSC520 microcontroller:

- The GP-DMA *initiator* is the I/O device that asserts GPDRQx. This is always an external I/O device (or memory -mapped I/O device) residing on the GP bus, or the internal UART serial ports, and can be either 8 bits or 16 bits. Note that the internal UARTs must be programmed as 8-bit channels.
- The GP-DMA *target* is always system memory (SDRAM). Table 14-4 on page 14-9 shows the possible GP-DMA initiators and targets.

- Since the GP-DMA target is always SDRAM, the relevant address range must be currently mapped to be system SDRAM. If that portion of the address space is not mapped to SDRAM, erroneous operation will result. See Chapter 4, “System Address Mapping”, for more details on how to set up the system address mapping.
- ÉlanSC520 microcontroller does not support peer-to-peer transfers between GP bus peripheral devices, or SDRAM-to-SDRAM.
- In PCI bus 2.2-compliant designs, software must limit the length of GP bus DMA demand- or block-mode transfers. Very large transfers could cause the PCI host bridge target controller to violate the 10 µs memory write maximum completion time limit set in the *PCI Local Bus Specification,* Revision 2.2.

Table 14-4 Supported GP-DMA Initiator/Target Combinations

GP-DMA Initiator	Channel Size	GP-DMA Target
UARTs	8 bits	SDRAM
GP Bus	8 or 16 bits	SDRAM

The GP-DMA controller provides the GPAEN signal to prevent other devices residing on the same external GP bus from decoding the address on the GPA bus. When the internal Transfer Count register rolls from 0h to FFFFh (FFFFFFh in enhanced GP-DMA mode), GP-DMA controller asserts GPTC to indicate the end of transfer.

14.5.1.1 GP-DMA Initiators

14.5.1.1.1 Internal UARTs

Each of the two UART serial ports on the ÉlanSC520 microcontroller can initiate DMA transfers from its transmit channel or receive channel, or both. Since the serial ports are 8-bit devices, their DMA requests can be mapped to any of the default 8-bit channels (channels 0–3).

- For a read transfer, the UART asserts its request from the transmit channel (txdrq), waits for the acknowledge (txdack), and latches the data from the low byte of the GPD15–GPD0 bus when the I/O command is asserted (GPIOWR).
- For a write transfer, the UART asserts its request from the receive channel (rxdrq), waits for the acknowledge (rxdack), and places the data on the low byte of the GPD15–GPD0 bus when the I/O command is asserted (GPIORD).

For the channel connected to the internal serial port, the drq sense level must be programmed as active High, the dack sense level must be programmed as active Low, the write mode must be programmed for late write using the WRTSEL bit, the timing mode must be configured for normal timing using the COMPTIM bit. This is the default configuration. These bits are found in the Slave and Master DMA Channel x Control (SLDMACTL and MSTDMACTL) registers. Note that internal requests from the UART serial ports cannot be mapped to a 16-bit channel, because the UARTs support 8-bit data transfer only.

14.5.1.1.2 External I/O Devices

An external I/O device can use any of the channels, depending on its size. Each I/O device uses one dedicated GPDRQ/GPDACK signal pair.

- During a read transfer, the external I/O device asserts its request (GPDRQx), waits for the acknowledge (GPDACKx), and latches the data from the GPD bus when the I/O command is asserted (GPIOWR).

- For a write transfer, the external I/O device asserts its request, waits for the acknowledge, and places the data on the GPD bus when the I/O command ($\overline{\text{GPIORD}}$) is asserted.

14.5.1.1.3 External Memory-Mapped I/O Devices

An external device on the GP bus can be mapped into memory address space. See Chapter 4, “System Address Mapping”, for more details. Such devices are referred to as *memory-mapped I/O devices*. GP-DMA transactions to a memory-mapped I/O device are handled in the same fashion as those to an I/O device, except that the commands used are $\overline{\text{GPMEMRD}}$ and $\overline{\text{GPMEMWR}}$, instead of $\overline{\text{GPIORD}}$ and $\overline{\text{GPIOWR}}$. The GP-DMA Memory-Mapped I/O (GPDMAMMIO) register (MMCR offset D81h) is used for this purpose.

14.5.1.2 GP-DMA Channel Mapping

GP-DMA requests can originate from the following sources:

- Transmit and receive channels from each of two internal UART serial ports (always 8-bit) for a total of four requests
- GP bus using GPDRQ3–GPDRQ0 and $\overline{\text{GPDACK3}}$–$\overline{\text{GPDACK0}}$ (8-bit or 16-bit).

Table 14-5 shows the ÉlanSC520 microcontroller resource and the GP-DMA channels to which the resource can be mapped.

All GP-DMA channel mapping in the ÉlanSC520 microcontroller is programmable using the two GP-DMA Resource Channel Map x (GPDMAEXTCHMAPx) registers.

Table 14-5 GP-DMA Channel Mapping

	GP-DMA Channel							
Microcontroller Resource	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
UART 1 transmit request	✔	✔	✔	✔				
UART 2 receive request	✔	✔	✔	✔				
UART 1 transmit request	✔	✔	✔	✔				
UART 2 receive request	✔	✔	✔	✔				
External request GPDRQ0	✔	✔	✔	✔		✔	✔	✔
External request GPDRQ1	✔	✔	✔	✔		✔	✔	✔
External request GPDRQ2	✔	✔	✔	✔		✔	✔	✔
External request GPDRQ3	✔	✔	✔	✔		✔	✔	✔

14.5.2 Operating Modes

The operating mode for the GP-DMA controller is configured using the ENH_MODE_ENB bit in the GP-DMA Control (GPDMACTL) register (MMCR offset D80h).

14.5.2.1 Normal GP-DMA Mode

Normal GP-DMA mode is the default operating mode of the GP-DMA controller. In this mode:

- Channels 0, 1, 2, and 3 are used for the internal UART serial ports and external 8-bit devices.
- Channel 5, 6, and 7 are used for any external 16-bit devices.

This mode is compatible with the PC/AT architecture.

14.5.2.2 Enhanced GP-DMA Mode

Only channels 3, 5, 6, and 7 support enhanced GP-DMA mode. In enhanced GP-DMA mode:

- Each of these four channels can be configured to be either 8-bit or 16-bit channel. The other channels (0, 1, and 2) can still be used as normal 8-bit channels in conjunction with the enhanced GP-DMA mode channels.
- The transfer count registers are increased to 24 bits in size, to allow counts up to 16 M (16,777,216) fransfers.
- The address adder is increased to 28 bits in size, eliminating the limitation of transferring within the 64 Kbytes boundaries (128 Kbytes for 16-bit devices) in normal GP-DMA mode.

This mode also offers the capability of chaining two noncontiguous memory buffers during DMA transfers, as described in “Buffer Chaining” on page 14-15.

14.5.3 Addressing GP-DMA Channels

14.5.3.1 Addressing In Normal GP-DMA Mode

GP-DMA Channel 4 is used to cascade channels 0–3 from the slave core through the master core to the CPU and is not available for data transfer. For proper operation, software must ensure that this setting is always configured for cascading only via the TRNMOD field in the Master DMA Channel 4–7 Mode (MSTDMAMODE) register (Port 00D6h).

14.5.3.1.1 8-Bit Transfers

Channels 0–3 support 8-bit data transfers between 8-bit I/O devices and system SDRAM. 8-bit GP-DMA can access any location within the system address space; however, the address adder is only 16 bits wide, so 8-bit GP-DMA requests cannot cross 64-Kbyte physical page boundaries. As shown in Table 14-6, during an 8-bit GP-DMA transfer:

- The Slave DMA Channel x Memory Address (GPDMAxMAR) registers provide address bits 15–0.
- The Slave DMA Channel x Page (GPDMAxPG) registers provide address bits 23–16.
- The GP-DMA Channel x Extended Page (GPDMAEXTPGx) registers provide bits 27–24 of the system memory address.

14.5.3.1.2 16-Bit Transfers

Channels 5–7 support 16-bit data transfers between 16-bit I/O devices and system SDRAM. 16-bit GP-DMA can access any even (word-aligned) location within the system address space; however, the address adder is only 16 bits wide, so 16-bit GP-DMA requests cannot cross 128-Kbyte physical page boundaries. During a 16-bit GP-DMA transfers:

- A0 is forced to 0.
- The Master DMA Channel x Memory Address (GPDMAxMAR) registers provide address bits 16–1.
- The Master DMA Channel x Page (GPDMAxPG) registers provide address bits 23–17.
- The GP-DMA Channel x Extended Page (GPDMAEXTPGx) registers provide bits 27–24 of the system memory address.

Table 14-6 8-Bit GP-DMA Channel Address Generation

Source	GP-DMA Channel x Extended Page Registers	Slave DMA Channel x Page Registers	Slave DMA Channel x Memory Address Register
Address	A27–A24	A23–A16	A15–A0

Table 14-7 16-Bit GP-DMA Channel Address Generation

Source	GP-DMA Channel x Extended Page Registers	Master DMA Channel x Page Registers	Master DMA Channel x Memory Address Register
Address	A27–A24	A23–A17	A16–A1, A0=0

14.5.3.2 Addressing In Enhanced GP-DMA Mode

In enhanced GP-DMA mode, channels 3, 5, 6 and 7 are programmable to support either 8-bit transfers or 16-bit transfers.

- When the channel is configured to be 8-bit, the address is generated as shown in Table 14-6.
- When the channel is configured to be 16-bit, the address is generated as shown in Table 14-7.
- However, when the buffer chaining feature is used, the memory address of the next data buffer is provided directly from the channel's Next Address register. This feature is described in "Buffer Chaining" on page 14-15.

The size of the address adder is increased to 28 bits wide to eliminate the limitation of 64-Kbyte physical page boundaries for 8-bit transfers and 128-Kbyte physical page boundaries for 16-bit transfers. This feature is available for channels 3, 5, 6, and 7 only.

14.5.4 GP-DMA Transfer Modes

The GP-DMA controller performs read, write, and verify operations in each of the three transfer modes: single, demand, or block. For all three modes, the GP-DMA initiator asserts GPDRQx and must hold it active until the assertion of $\overline{\text{GPDACKx}}$ in order to be recognized.

14.5.4.1 Single Transfer Mode

In *single transfer mode*, the GP-DMA controller performs one transfer each time it is granted the Am5$_X$86 CPU bus. The GP-DMA initiator asserts GPDRQx and holds it active as long as it has data to be transferred. The initiator must negate its DRQx relative to the I/O commands to ensure correct operation.

14.5.4.2 Demand Transfer Mode

In *demand transfer mode*, the GP-DMA initiator asserts GPDRQx and holds it active as long as it has data to be transferred. The GP-DMA controller continues to perform GP-DMA transfers until Terminal Count (TC) is reached or the GPDRQx is deasserted by the GP-DMA initiator. The initiator must negate its DRQx relative to the I/O commands to ensure correct operation.

When using demand transfer mode, if the transfer is configured for automatic initialization control mode, GPDRQx must be deasserted prior to the assertion of GPTC in the last DMA cycle to prevent another transfer. Otherwise, the channel is automatically masked and requires initialization before it will respond to subsequent requests.

14.5.4.3 Block Transfer Mode

In *block transfer mode*, the GP-DMA initiator asserts GPDRQ and holds it active until acknowledged by the assertion of GPDACKx. The GP-DMA controller performs GP-DMA transfers until TC is reached, indicating the programmed number of transfers has been completed.

14.5.4.4 Transfer Types

Three GP-DMA transfer types are supported: read, write, and verify.

- A *read transfer*, shown in Figure 14-3, consists of a memory read cycle from the address in the current address register (concatenation of the channel's Memory Address register, Page register, and Extended Page register), followed by an I/O write cycle to the associated device.
- A *write transfer*, shown in Figure 14-4, consists of an I/O read cycle followed by a memory write cycle to the address in the current address register. Depending on the GP-DMA channel selected, the data can be 8 bits or 16 bits in width.
- A *verify transfer*, shown in Figure 14-5, is either a read transfer or a write transfer, but without the generation of the I/O and memory control signals, such as GPIORD, GPIOWR, GPMEMRD, and GPMEMWR. A verify transfer is normally used for checking the GP-DMA core to determine whether the address generation and control logic are operating correctly. Data are not transferred in a verify cycle. ÉlanSC520 microcontroller does not drive the SDRAM address out on the MA address bus during a DMA verify cycle.

Figure 14-3 GP-DMA Read Transfer

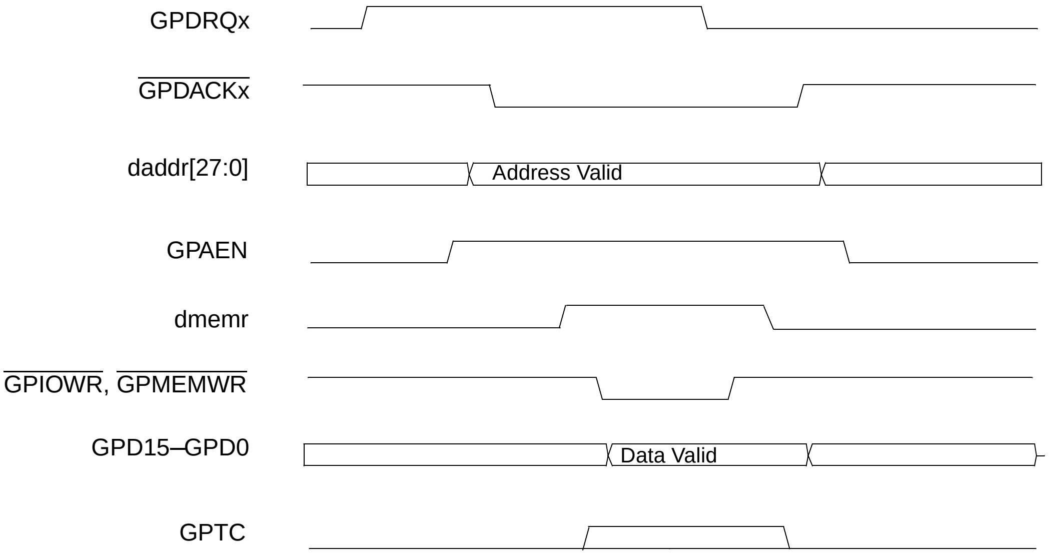


Figure 14-4 GP-DMA Write Transfer

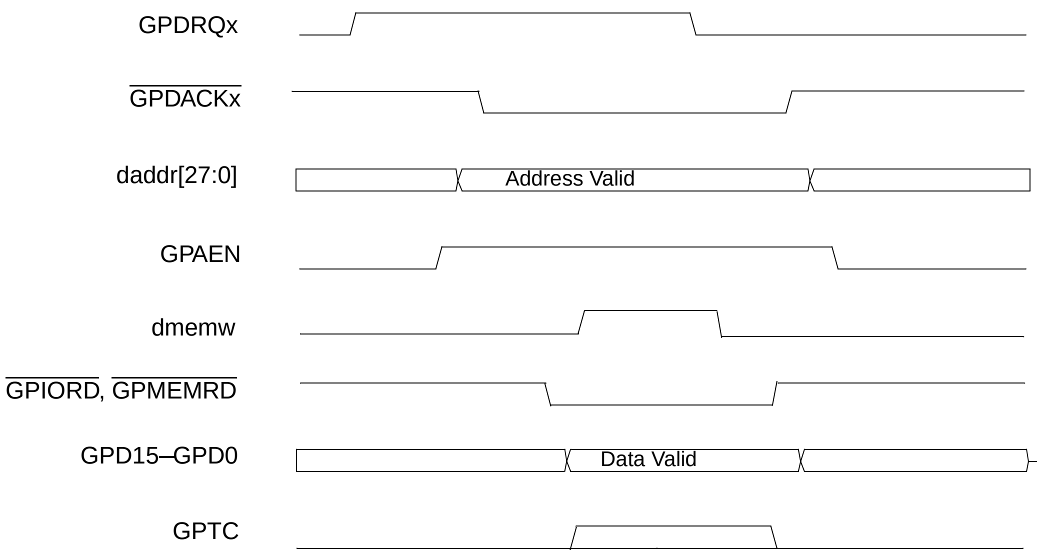
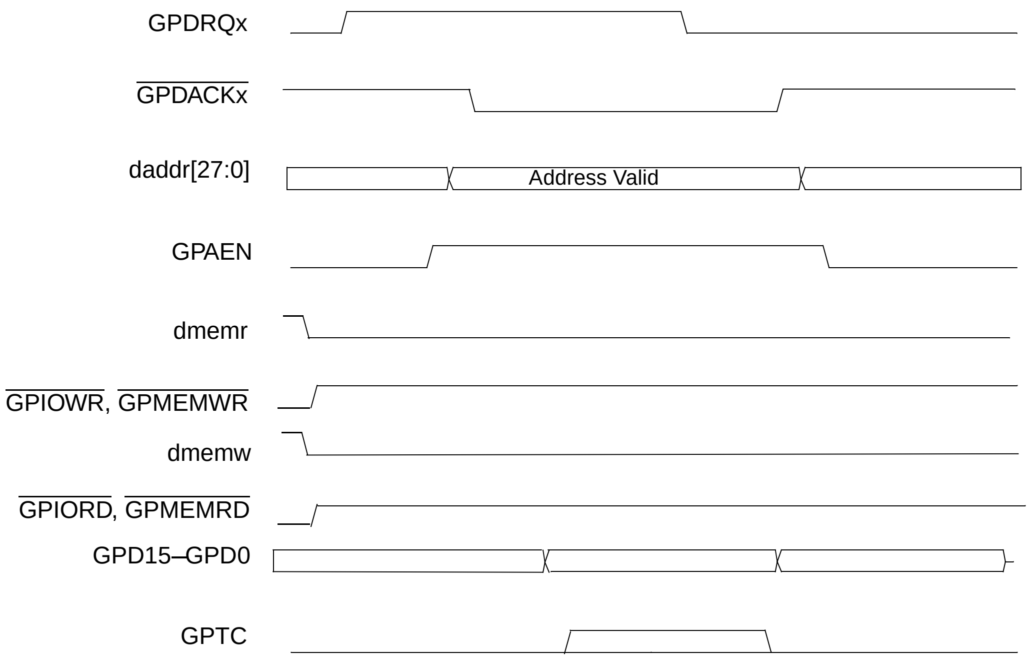


Figure 14-5 GP-DMA Verify Transfer



14.5.4.5 Automatic Initialization Control

When automatic initialization control mode is enabled via the AINIT bit in the Slave or Master Channel x Mode register, the original values of the current address and current count registers are automatically restored to the values in the base address and base count registers of the given channel following the terminal count.

This feature is useful when data quantities of the same size are transferred to or from a fixed buffer in SDRAM. This feature must be disabled when using buffer chaining mode; otherwise, unexpected results may occur.

14.5.4.6 Priority

The GP-DMA controller offers two priority schemes for servicing multiple requests. After the recognition of any one channel for service, the other channels are prevented from generating DMA cycles until the current transfer has completed (i.e., the current channel's $\overline{\text{DACKx}}$ has deasserted).

- The *fixed priority* scheme is based upon the value of channel numbers (Channel 0 is the highest priority, Channel 7 is the lowest priority). The higher priority channel prevents the lower priority channel from servicing the request.
- In the *rotating priority* scheme, the last channel serviced becomes the lowest priority, with the other channels rotating accordingly. This scheme is also known as the round robin scheme.

14.5.4.7 Buffer Chaining

In enhanced GP-DMA mode, channels 3, 5, 6, and 7 allow transfer to/from two or more data buffers in SDRAM for a single transfer request (fragmented data buffers). This feature is known as *buffer chaining*. The purpose of this feature is to facilitate GP-DMA transfers to or from non-contiguous buffers in SDRAM.

An example usage of this feature is to transfer a packet of data from SDRAM to the external device. The packet header and the packet data might be in two noncontiguous locations in SDRAM. By using the buffer chaining feature, users can transfer both the packet header and packet data in one DMA transfer. Similarly, the GP-DMA controller can be used to split up a packet header from the packet data into two SDRAM buffers when receiving packets.

Buffer chaining mode is enabled by setting the appropriate CHx_BCHN_ENB bit in the Buffer Chaining Control (GPDMABCCTL) register (MMCR offset D98h).

1. The Next Address registers and the Next Transfer Count registers should be programmed prior to the start of the GP-DMA cycle.
2. When the transfer count is reached, the GP-DMA controller checks the CHx_CBUF_VAL bits in the Buffer Chaining Valid (GPDMABCVAL) register (MMCR offset D9Bh).
3. If this bit is set, the contents of the Next Address and the Next Transfer Count registers are loaded into the internal current address and current transfer count registers, respectively.
4. The GP-DMA controller hardware then generates a maskable or non-maskable interrupt and clears the CHx_CBUF_VAL bits.
5. This bit indicates to software that another buffer can be set up in the chain by writing to the Next Address and Next Transfer Count registers with new values.
6. The DMA transfer then continues until the next terminal count.
7. If the CHx_CBUF_VAL bits were not set, GP-DMA controller generates the interrupt and also asserts GPTC to indicate the end of the chain.

Typically, buffer chaining should be used in single transfer mode, but block mode or demand mode operation is also supported.

When using block transfer mode, the GP-DMA controller holds the bus request active until the end of the last buffer in the chain. It is worth noting that only two buffers can be chained at a time when using block transfer mode. Because the GP-DMA controller does not release the GP bus during the transfer, the Next Address and Next Transfer Count cannot be reprogrammed to link in another buffer while a GP-DMA transfer is in progress.

The automatic initialization control mode cannot be used in conjunction with buffer chaining mode.

14.5.5 Bus Cycles

Table 14-8 shows the four GP-DMA cycle types and the command strobes generated in each cycle. The GP bus command strobes $\overline{\text{GPMEMRD}}$ and $\overline{\text{GPMEMWR}}$ are asserted for memory-mapped I/O devices on this bus. The internal memory commands are not shown in this table.

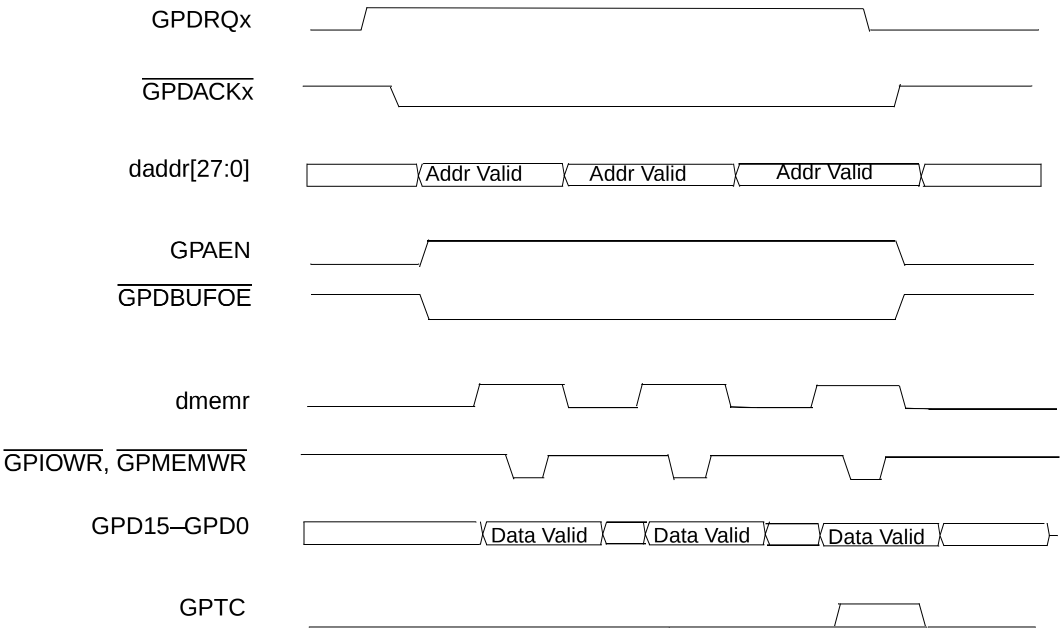
Table 14-8 GP-DMA Cycle Types

GP-DMA Initiator	GP-DMA Target	Data Transfer Direction (GP-DMA Cycle Type)	GP Bus Command Strobes Generated
I/O device	SDRAM	I/O to memory (GP-DMA write)	$\overline{\text{GPIORD}}$
I/O device	SDRAM	Memory to I/O (GP-DMA read)	$\overline{\text{GPIOWR}}$
Memory-mapped I/O device	SDRAM	Memory-Mapped I/O to memory (GP-DMA write)	$\overline{\text{GPMEMRD}}$
Memory-mapped I/O device	SDRAM	Memory to memory-mapped I/O (GP-DMA read)	$\overline{\text{GPMEMWR}}$

14.5.5.1 GP Bus I/O to SDRAM

Figure 14-6 shows a GP-DMA read cycle in demand transfer mode.

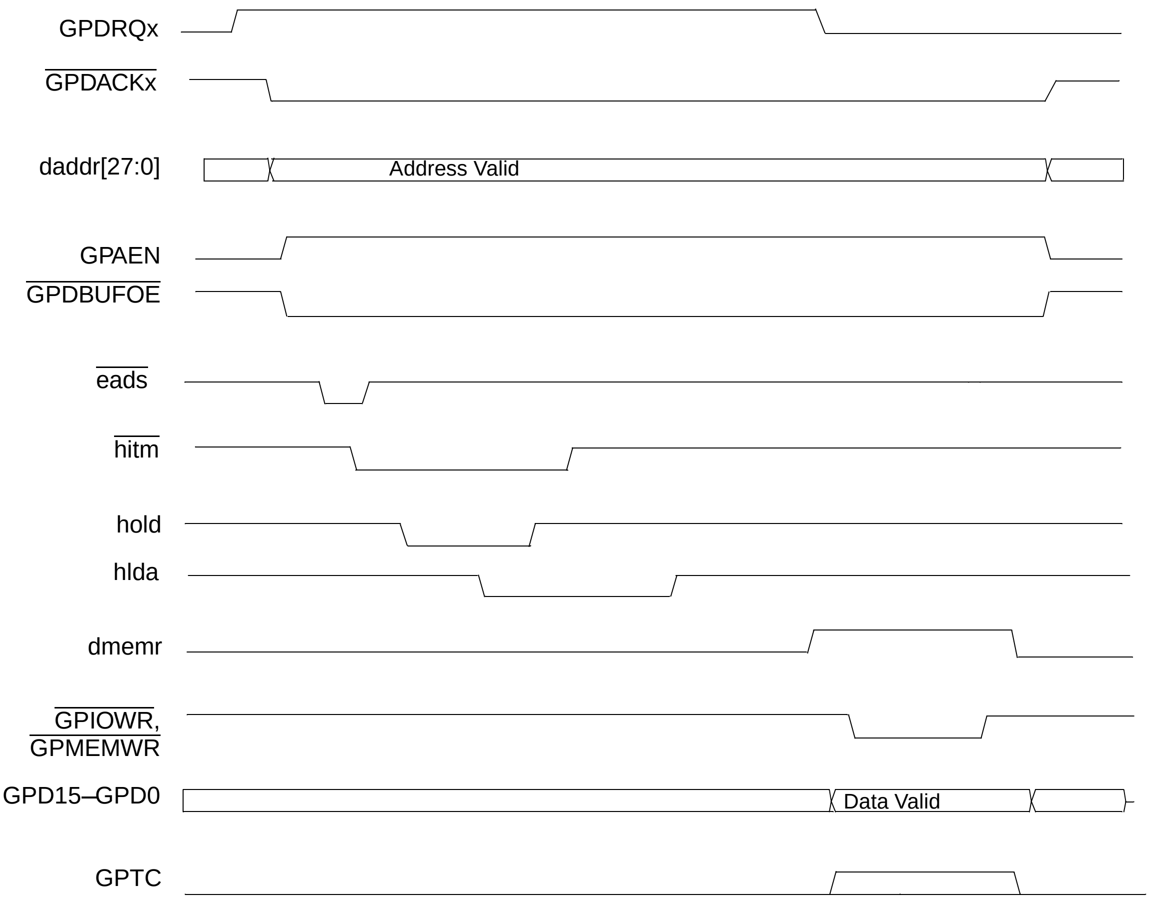
Figure 14-6 GP-DMA Read in Demand Transfer Mode


GPDRQx
$\overline{\text{GPDACKx}}$
daddr[27:0]
Addr Valid
Addr Valid
Addr Valid
GPAEN
$\overline{\text{GPDBUFOE}}$
dmemr
$\overline{\text{GPIOWR}}$, $\overline{\text{GPMEMWR}}$
GPD15–GPD0
Data Valid
Data Valid
Data Valid
GPTC

14.5.5.2 GP-DMA Read with Cache Hit

Figure 14-7 shows a read transfer with a cache hit (write-back cache).

Figure 14-7 GP-DMA Read Transfer with Cache Hit (Write-Back Cache)



14.5.6 GP Bus Echo Mode

When GP bus echo mode is enabled, GPAEN is driven high during accesses from the Am5$_x$86 CPU to internal peripherals to prevent external devices from decoding (or responding to) these internal peripheral accesses. In normal operation (GP bus echo mode disabled), the GP bus controller never asserts GPAEN.

However, accesses initiated by the GP bus DMA controller are not affected by enabling the GP bus echo mode, and therefore the GP bus DMA controller still asserts GPAEN as it does during normal operation. During an internal GPDMA access in GP bus echo mode, the external GP bus commands, $\overline{\text{GPIORD}}$, $\overline{\text{GPMEMRD}}$, $\overline{\text{GPIOWR}}$, $\overline{\text{GPMEMWR}}$, are not asserted. However, GPAEN is still asserted. For additional information about this mode, see “GP Bus Echo Mode” on page 13-10.

14.5.7 Clocking Considerations

The GP-DMA controller can be programmed to operate at 4 MHz, 8 MHz, or 16 MHz. This option is specified in the GP-DMA Control (GPDMACTL) register (MMCR offset D80h). Note that these frequencies are derived from the 33-MHz clock. The exact frequency is an even fraction of the crystal (33.000-MHz or 33.333-MHz) being used in the system.

14.5.8 Interrupts

In normal GP-DMA mode, the GP-DMA controller does not generate interrupts, but it does assert GPTC upon the completion of every transfer.

When buffer chaining mode is enabled, the GP-DMA controller generates a maskable or non-maskable interrupt every time a buffer is completely transferred. This interrupt is generated after the valid values of the Next Address and Next Transfer Count are loaded into the internal current address and current transfer count registers, respectively. GPTC is asserted only when there is no other buffer in the chain. When GPTC is asserted, the interrupt is still generated.

14.5.9 Software Considerations

Channel 4 must always be set to be in cascade mode; otherwise, erroneous operation may result. Only Channel 4 should be programmed for cascade mode. All other channels should be programmed to be in one of the other three modes (single, demand or block).

The Memory Address and Transfer Count registers of each channel are byte-accessed. Two consecutive byte reads or writes to the same I/O address are required when accessing the 16-bit values of these registers. In enhanced GP-DMA mode, although the Next Address registers and the Next Transfer Count registers are both split up into two 16-bit registers, the Low and High words have been placed so that they can be accessed using 32-bit instructions. Although the GP bus splits 32-bit accesses up into two 16-bit accesses (i.e., the setting of the low and high address will be nonatomic), this should not typically cause any problems.

When using the buffer chaining feature in block transfer mode, the GP-DMA controller continues to hold the bus request until the second buffer is finished. The interrupt generated after the first buffer finishes in this case is useless to software, because the interrupt handling routine is not able to get access to the Am5$_x$86 CPU bus (because the GP-DMA controller is programmed for block transfer mode).

Note that the GPDRQx signal must be deasserted before an active channel can be masked.

14.5.10 Latency

14.5.10.1 Nonpreemptive Latency

The ÉlanSC520 microcontroller implements a write buffer and a read buffer (with read-ahead feature) to optimize SDRAM performance. These buffers can improve GP-DMA latency during block transfer or demand transfer.

- During a write transfer, the write buffer collects bytes (or words) from the GP bus and writes back to SDRAM in a full doubleword. This mechanism effectively provides one-wait-state write accesses to SDRAM, as seen from the GP-DMA controller.
- During a read transfer, the read buffer reads the entire cache-line (16 bytes). This effectively provides zero-wait-state read accesses from SDRAM by the GP-DMA controller. However, since the read buffer fetches forward, GP-DMA channels that are configured in address decrement mode experience more read buffer misses. The read buffer does not prefetch for GP-DMA accesses because they are less than one doubleword.

The operations of these buffers are described in detail in Chapter 11, “Write Buffer and Read Buffer”.

14.5.10.2 Preemptive Latency

The following events could delay a GP-DMA acknowledgment.

- SDRAM refresh cycle (the acknowledgment is given; however, the transfer is delayed)
- PCI requests
- A higher priority GP-DMA request
- A cache write-back, if the GP-DMA target is in a dirty cache-line (the acknowledgment is given; however, the transfer is delayed)
- Slow transfers to ROM/GP bus

Once a demand transfer or block transfer has started, if the GP-DMA controller is trying to read from a SDRAM region that is in the cache, the transfer is paused while a cache snoop occurs. If the cache holds data in the cache line that the GP-DMA controller is accessing, a cache-line write-back cycle may also occur.

14.6 INITIALIZATION

The GP-DMA controller is reset by a system reset. In addition, the slave and the master controllers each have a software reset source, from the Slave DMA Controller Reset (SLDMARST) register (Port 000Dh and the Master DMA Controller Reset (MSTDMARST) register (Port 00DAh), respectively.

The GP-DMA controller is enabled after system reset, but all channels are masked off. This is also the state after the DMA Controller Reset registers are written to. All channels default to normal GP-DMA mode. The operating frequency defaults to 4 MHz.

14.6.1 Example Configurations

14.6.1.1 Configuring an 8-Bit Channel in Normal GP-DMA Mode

In normal GP-DMA mode, there are four 8-bit channels: 0, 1, 2, and 3. Any internal request from the serial ports or any external request can be mapped to one of these channels. The following steps configure an 8-bit channel.

1. Enable the DMA slave core.
2. Program Channel 4 to use cascade mode via the TRNMOD field in the Master DMA Channel 4–7 Mode (MSTDMAMODE) register (Port 00D6h) and unmask Channel 4.
3. Program operating frequency if not using the default 4 MHz.
4. Map the request to a specific channel.
5. Program the memory address, transfer count, page address, and extended page address of the associated channel.
6. Program DMA mode, type, address increment mode, and priority mode.
7. Unmask the channel request in the Slave DMA General Mask (SLDMAGENMSK) register (Port 000Fh). At this point, the GP-DMA controller is ready to accept the external request.

14.6.1.2 Configuring a 16-Bit Channel in Normal GP-DMA Mode

In normal GP-DMA mode, there are three 16-bit channels: 5, 6 and 7. Any external request can be mapped to one of these channels. The internal requests from the UART serial ports cannot be mapped to a 16-bit channel because they only support 8-bit data transfer. The following steps configure a 16-bit channel for an external request.

1. Enable the DMA master core.
2. Program the operating frequency if not using the default 4 MHz.
3. Map the external request to a specific channel.
4. Program the memory address, transfer count, page address, and extended page address of the associated channel.
5. Program DMA mode, type, address increment mode, and priority mode.
6. Unmask the channel request in the Master DMA General Mask (MSTDMAGENMSK) register (Port 00DEh). At this point, the GP-DMA controller is ready to accept the external request.

14.6.1.3 Configuring an 8-Bit Channel in Enhanced GP-DMA Mode

In enhanced GP-DMA mode, channels 5, 6, and 7 can be configured to be 8-bit channels. Any internal request from the UART serial ports can be mapped to Channel 3 for the enhanced GP-DMA mode features. The 8-bit external devices can be mapped to channels 3, 5, 6, and 7. The following steps configure an 8-bit channel for an external request.

1. Enable the DMA slave core if using Channel 3, otherwise enable the master core.
2. If using Channel 3, program Channel 4 to use cascade mode via the TRNMOD field in the Master DMA Channel 4–7 Mode (MSTDMAMODE) register (Port 00D6h) and unmask Channel 4. Also, if using channels 5, 6, or 7, set the corresponding CHx_ALT_SIZE bit in the GP-DMA Control (GPDMACTL) register (MMCR offset D80h).
3. Program the operating frequency if not using the default 4 MHz.
4. Enable enhanced GP-DMA mode.
5. Map the external request to a specific channel.
6. Program the memory address, transfer count, page address, and extended page address of the associated channel.
7. Program the extended transfer count for any transfer larger than 64 Kbytes (optional).
8. Program DMA mode, type, address increment mode, and priority mode.
9. Program the next address, next transfer count, and enable buffer chaining mode (optional).
10. Unmask the channel request in the General Mask register. At this point, the GP-DMA controller is ready to accept the external request.

14.6.1.4 Configuring a 16-Bit Channel in Enhanced GP-DMA Mode

In enhanced GP-DMA mode, Channel 3 can be configured to be a 16-bit channel. The 16-bit external devices can be mapped to channel 3, 5, 6, and 7. The following steps configure a 16-bit channel for an external request.

1. Enable the DMA slave core if using Channel 3, otherwise enable the master core.
2. If using Channel 3, program Channel 4 to use cascade mode via the TRNMOD field in the Master DMA Channel 4–7 Mode (MSTDMAMODE) register (Port 00D6h) and unmask Channel 4. Also, set the CH3_ALT_SIZE bit in the GP-DMA Control (GPDMACTL) register (MMCR offset D80h).
3. Program the operating frequency if not using the default 4 MHz.
4. Enable enhanced GP-DMA mode.
5. Map the external request to a specific channel.
6. Program the memory address, transfer count, page address, and extended page address of the associated channel.
7. Program the extended transfer count for any transfer larger than 128 Kbytes (optional).
8. Program DMA mode, type, address increment mode, and priority mode.
9. Program the next address, next transfer count, and enable buffer chaining mode (optional).
10. Unmask the channel request in the General Mask register. At this point, the GP-DMA controller is ready to accept the external request.

CHAPTER

15 PROGRAMMABLE INTERRUPT CONTROLLER

15.1 OVERVIEW

The ÉlanSC520 microcontroller's programmable interrupt controller (PIC) consists of three industry-standard controllers, integrated with a highly programmable interrupt router.

The programmable interrupt controller is configured so that two controllers are cascaded as slaves to a master controller that arbitrates interrupt requests from various sources to the Am5$_x$86 CPU. Interrupt channel 2 (IR2) and channel 5 (IR5) of the Master controller are hard-wired to the outputs of the Slave 1 and Slave 2 controller respectively. In this configuration, up to 22 maskable interrupt channels of different priorities are available to the programmer.

The programmable interrupt router handles routing of the various external and internal interrupt sources to the 22 interrupt channels of the three controllers. The interrupt router can also be programmed to handle routing of various NMI sources to generate a non-maskable interrupt to the CPU.

The ÉlanSC520 microcontroller's programmable interrupt controller is designed to support PC/AT-compatible features. Startup software can configure the programmable interrupt router to route the sources to be used as ISA interrupts to the appropriate interrupt channels of the Slave 1 and Master controllers.

PCI interrupts are level-sensitive, shareable, and typically implemented as open-drain inputs. To support this, the programmable interrupt controller optionally allows the selection of edge-triggered or level-sensitive interrupt detection on a per-channel basis, as an alternative to the standard global selection of edge-triggered or level-sensitive detection on all channels. This enhancement provides maximum flexibility in configuring a system environment where mixed interrupt types are used.

Features of the ÉlanSC520 microcontroller's programmable interrupt controller include:

- 22 interrupt priority levels plus NMI
- Programmable interrupt router capable of mapping interrupt sources (internal and external) to different priorities or NMI
- 15 general-purpose external interrupt requests (GPIRQ10–GPIRQ0 and $\overline{\text{INTA}}$–$\overline{\text{INTD}}$), programmable to be edge- or level-sensitive
- 19 internal interrupt requests programmable to be edge- or level-sensitive
- Ability to assert any of the interrupt priority levels, including NMI, via software
- Configurable to provide software compatibility with PC/AT interrupt controller
- Programmable interrupt polarity inversion for external sources
- Am5$_x$86 CPU floating point error ($\overline{\text{ferr}}$) interrupt clear, $\overline{\text{ignne}}$ function

15.2 BLOCK DIAGRAM

Figure 15-1 is a block diagram of the ÉlanSC520 microcontroller's programmable interrupt controller showing interrupt sources and routing.

The programmable interrupt controller consists of a system of three individual interrupt controllers (Master, Slave 1 and Slave 2), each of which has eight interrupt channels. Two of the interrupt channels on the Master controller are used to cascade the slave controllers. This allows a total of 22 interrupt priority levels in the ÉlanSC520 microcontroller. The priority levels are numbered from P1–P22 to indicate which priority levels are assigned to slave or master controllers, with P1 being the highest and P22 the lowest priority.

15.3 SYSTEM DESIGN

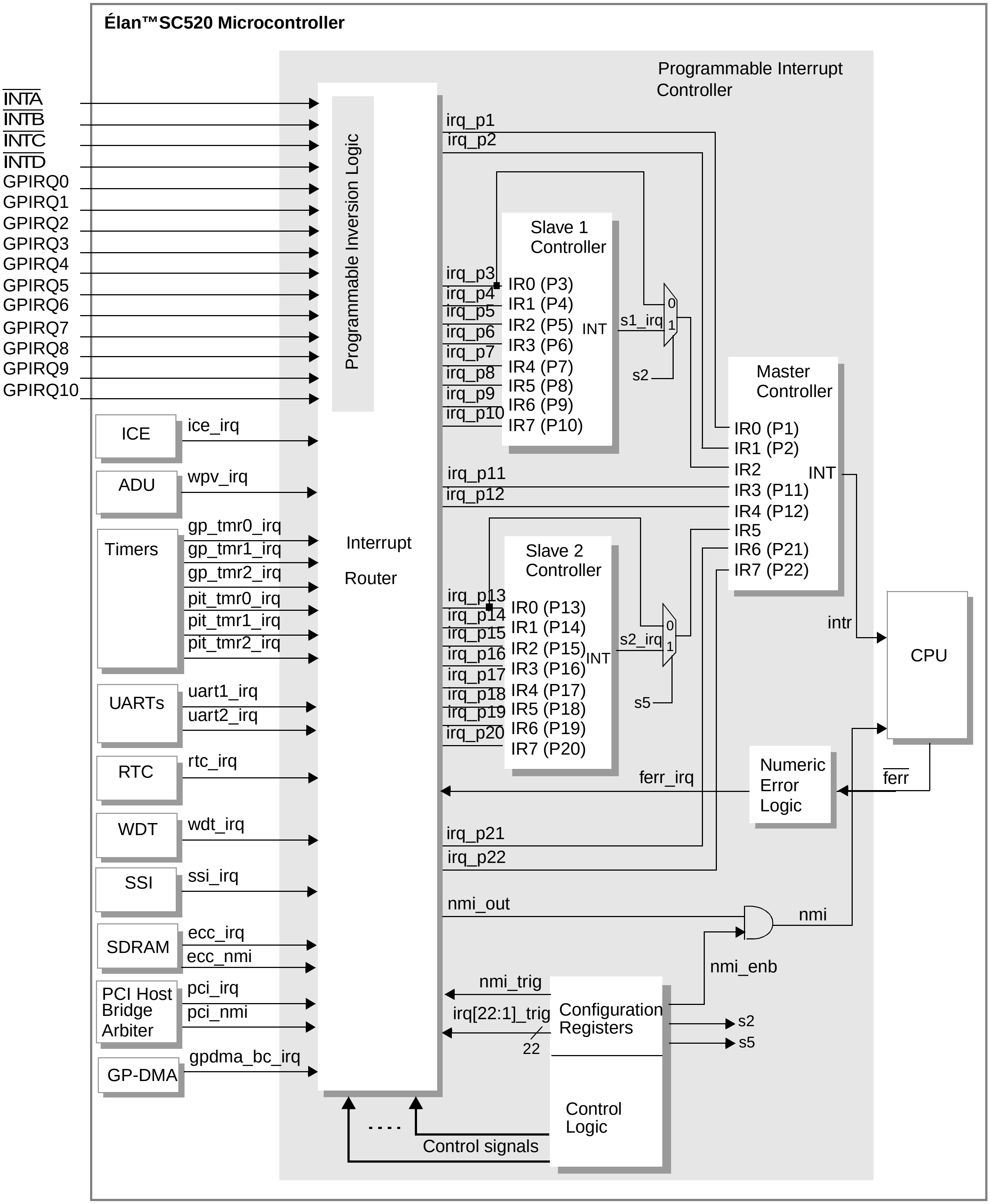
Table 15-1 shows PIC signals shared with other interfaces. When enabled, the multiplexed signals shown in Table 15-1 either disable or alter any other function that uses the same pin.

The GPIRQ10–GPIRQ0 and INTA–INTD signals are asserted when a peripheral requires interrupt service. The dedicated INTA–INTD pins are the same type of interrupt as the GPPIRQx signals. They are named INTx to match the common PCI interrupt naming convention.

Table 15-1 Programmable Interrupt Controller Signals Shared with Other Interfaces

PIO (Default) Function	Interface Function	Control Bit	Register
PIO23	GPIRQ0	PIO23_FNC	PIO31–PIO16 Pin Function Select (PIOPFS31_16) register (MMCR offset C22h)
PIO22	GPIRQ1	PIO22_FNC	
PIO21	GPIRQ2	PIO21_FNC	
PIO20	GPIRQ3	PIO20_FNC	
PIO19	GPIRQ4	PIO19_FNC	
PIO18	GPIRQ5	PIO18_FNC	
PIO17	GPIRQ6	PIO17_FNC	
PIO16	GPIRQ7	PIO16_FNC	
PIO15	GPIRQ8	PIO15_FNC	PIO15–PIO0 Pin Function Select (PIOPFS15_0) register (MMCR offset C20h)
PIO14	GPIRQ9	PIO14_FNC	
PIO13	GPIRQ10	PIO13_FNC	

Figure 15-1 Programmable Interrupt Controller (PIC) Block Diagram



Notes:

The priorities of the 22 channels are shown, with P1 being the highest and P22 the lowest priority.

15.4 REGISTERS

The programmable interrupt controller (PIC) is controlled by the registers listed in Table 15-2 and Table 15-3.

Table 15-2 Programmable Interrupt Controller Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
PIO15–PIO0 Pin Function Select	PIOPFS15_0	C20h	PIO or interface function select: GPIRQ10–GPIRQ8
PIO31–PIO16 Pin Function Select	PIOPFS31_16	C22h	PIO or interface function select: GPIRQ7–GPIRQ0
Interrupt Control	PICICR	D00h	Global interrupt mode enables, global NMI enable, NMI completion control
Master PIC Interrupt Mode	MPICMODE	D02h	Edge- or level-sensitive interrupt mode select per channel
Slave 1 PIC Interrupt Mode	SL1PICMODE	D03h	Edge- or level-sensitive interrupt mode select per channel
Slave 2 PIC Interrupt Mode	SL2PICMODE	D04h	Edge- or level-sensitive interrupt mode select per channel
Software Interrupt 16–1 Control	SWINT16_1	D08h	Software interrupt generation control (priority levels 1–16)
Software Interrupt 22–17/NMI Control	SWINT22_17	D0Ah	Software interrupt generation control (priority level 17–22), software NMI generation to the CPU
Interrupt Pin Polarity	INTPINPOL	D10h	Polarity of external interrupt sources ($\overline{\text{INTA}}$–$\overline{\text{INTD}}$ and GPIRQ10–GPIRQ0)
PCI Host Bridge Interrupt Mapping	PCIHOSTMAP	D14h	System arbiter and PCI Host Bridge interrupt mapping to any of 22 available interrupt channels or NMI, PCI NMI enable control
ECC Interrupt Mapping	ECCMAP	D18h	ECC interrupt mapping to any of 22 available interrupt channels or NMI, ECC NMI enable control
GP Timer 0 Interrupt Mapping	GPTMR0MAP	D1Ah	GP Timer 0 interrupt mapping to any of 22 available interrupt channels or NMI
GP Timer 1 Interrupt Mapping	GPTMR1MAP	D1Bh	GP Timer 1 interrupt mapping to any of 22 available interrupt channels or NMI
GP Timer 2 Interrupt Mapping	GPTMR2MAP	D1Ch	GP Timer 2 interrupt mapping to any of 22 available interrupt channels or NMI
PIT 0 Interrupt Mapping	PIT0MAP	D20h	PIT 0 interrupt mapping to any of 22 available interrupt channels or NMI
PIT 1 Interrupt Mapping	PIT1MAP	D21h	PIT 1 interrupt mapping to any of 22 available interrupt channels or NMI
PIT 2 Interrupt Mapping	PIT2MAP	D22h	PIT interrupt mapping to any of 22 available interrupt channels or NMI

Table 15-2 Programmable Interrupt Controller Registers—Memory-Mapped (Continued)

Register	Mnemonic	MMCR Offset Address	Function
UART 1 Interrupt Mapping	UART1MAP	D28h	UART 1 interrupt mapping to any of 22 available interrupt channels or NMI
UART 2 Interrupt Mapping	UART2MAP	D29h	UART 2 interrupt mapping to any of 22 available interrupt channels or NMI
PCI Interrupt A Mapping	PCIINTAMAP	D30h	PCI $\overline{\text{INTA}}$ mapping to any of 22 available interrupt channels or NMI
PCI Interrupt B Mapping	PCIINTBMAP	D31h	PCI $\overline{\text{INTB}}$ mapping to any of 22 available interrupt channels or NMI
PCI Interrupt C Mapping	PCIINTCMAP	D32h	PCI $\overline{\text{INTC}}$ mapping to any of 22 available interrupt channels or NMI
PCI Interrupt D Mapping	PCIINTDMAP	D33h	PCI $\overline{\text{INTD}}$ mapping to any of 22 available interrupt channels or NMI
DMA Buffer Chaining Interrupt Mapping	DMABCINTMAP	D40h	DMA buffer chain interrupt mapping to any of 22 available interrupt channels or NMI
SSI Interrupt Mapping	SSIMAP	D41h	SSI interrupt mapping to any of 22 available interrupt channels or NMI
Watchdog Timer Interrupt Mapping	WDTMAP	D42h	WDT interrupt mapping to any of 22 available interrupt channels or NMI
RTC Interrupt Mapping	RTCMAP	D43h	RTC interrupt mapping to any of 22 available interrupt channels or NMI
Write-Protect Violation Interrupt Mapping	WPVMAP	D44h	Write-protect violation to PAR interrupt mapping to any of 22 available interrupt channels or NMI
AMDebug Technology RX/TX Interrupt Mapping	ICEMAP	D45h	AMDebug technology JTAG port receive or transmit interrupt mapping to any of 22 available interrupt channels or NMI
Floating Point Error Interrupt Mapping	FERRMAP	D46h	Floating point error interrupt mapping to any of 22 available interrupt channels or NMI
GPIRQ0 Interrupt Mapping	GP0IMAP	D50h	GPIRQ0 interrupt mapping to any of 22 available interrupt channels or NMI
GPIRQ1 Interrupt Mapping	GP1IMAP	D51h	GPIRQ1 interrupt mapping to any of 22 available interrupt channels or NMI
GPIRQ2 Interrupt Mapping	GP2IMAP	D52h	GPIRQ2 interrupt mapping to any of 22 available interrupt channels or NMI
GPIRQ3 Interrupt Mapping	GP3IMAP	D53h	GPIRQ3 interrupt mapping to any of 22 available interrupt channels or NMI
GPIRQ4 Interrupt Mapping	GP4IMAP	D54h	GPIRQ4 interrupt mapping to any of 22 available interrupt channels or NMI
GPIRQ5 Interrupt Mapping	GP5IMAP	D55h	GPIRQ5 interrupt mapping to any of 22 available interrupt channels or NMI
GPIRQ6 Interrupt Mapping	GP6IMAP	D56h	GPIRQ6 interrupt mapping to any of 22 available interrupt channels or NMI

Table 15-2 Programmable Interrupt Controller Registers—Memory-Mapped (Continued)

Register	Mnemonic	MMCR Offset Address	Function
GPIRQ7 Interrupt Mapping	GP7IMAP	D57h	GPIRQ7 interrupt mapping to any of 22 available interrupt channels or NMI
GPIRQ8 Interrupt Mapping	GP8IMAP	D58h	GPIRQ8 interrupt mapping to any of 22 available interrupt channels or NMI
GPIRQ9 Interrupt Mapping	GP9IMAP	D59h	GPIRQ9 interrupt mapping to any of 22 available interrupt channels or NMI
GPIRQ10 Interrupt Mapping	GP10IMAP	D5Ah	GPIRQ10 interrupt mapping to any of 22 available interrupt channels or NMI

Table 15-3 Programmable Interrupt Controller Registers—Direct-Mapped

Register	Mnemonic	I/O Address	Function
Master PIC Interrupt Request Slave 2 PIC Interrupt Request Slave 1 PIC Interrupt Request	MPICIR S2PICIR S1PICIR	0020h 0024h 00A0h	Real-time status of interrupt request assertion
Master PIC In-Service Slave 2 PIC In-Service Slave 1 PIC In-Service	MPICISR S2PICISR S1PICISR	0020h 0024h 00A0h	Interrupt request service status
Master PIC Initialization Control Word 1 (ICW1) Slave 2 PIC Initialization Control Word 1 (ICW1) Slave 1 PIC Initialization Control Word 1 (ICW1)	MPICICW1 S2PICICW1 S1PICICW1	0020h 0024h 00A0h	Interrupt mode, address interval, cascade or single PIC configuration, ICW4 control
Master PIC Operation Control Word 2 (OCW2) Slave 2 PIC Operation Control Word 2 (OCW2) Slave 1 PIC Operation Control Word 2 (OCW2)	MPICOCW2 S2PICOCW2 S1PICOCW2	0020h 0024h 00A0h	Interrupt EOI, priority rotation control, EOI level select, control to access OCW2 and OCW3
Master PIC Operation Control Word 3 (OCW3) Slave 2 PIC Operation Control Word 3 (OCW3) Slave 1 PIC Operation Control Word 3 (OCW3)	MPICOCW3 S2PICOCW3 S1PICOCW3	0020h 0024h 00A0h	Poll command, read register command, special mask mode
Master PIC Initialization Control Word 2 (ICW2) Slave 2 PIC Initialization Control Word 2 (ICW2) Slave 1 PIC Initialization Control Word 2 (ICW2)	MPICICW2 S2PICICW2 S1PICICW2	0021h 0025h 00A1h	Base interrupt vector number

Table 15-3 Programmable Interrupt Controller Registers—Direct-Mapped (Continued)

Register	Mnemonic	I/O Address	Function
Master PIC Initialization Control Word 3 (ICW3) Slave 2 PIC Initialization Control Word 3 (ICW3) Slave 1 PIC Initialization Control Word 3 (ICW3)	MPICICW3 S2PICICW3 S1PICICW3	0021h 0025h 00A1h	Slave cascading channel select (MPICICW3)
Master PIC Initialization Control Word 4 (ICW4) Slave 2 PIC Initialization Control Word 4 (ICW4) Slave 1 PIC Initialization Control Word 4 (ICW4)	MPICICW4 S2PICICW4 S1PICICW4	0021h 0025h 00A1h	Nested mode, EOI mode
Master PIC Interrupt Mask (OCW1) Slave 2 PIC Interrupt Mask (OCW1) Slave 1 PIC Interrupt Mask (OCW1)	MPICINTMSK S2PICINTMSK S1PICINTMSK	0021h 0025h 00A1h	Channel interrupt mask
Floating Point Error Interrupt Clear	FPUERRCLR	F0h	Clear FPU error interrupt

15.5 OPERATION

15.5.1 Interrupt Flow Sequence

The following describes the typical interrupt flow sequence in a system that uses the ÉlanSC520 microcontroller's PIC.

1. When a device generates an interrupt request that translates to either a rising edge or level High at the mapped interrupt channel, the corresponding Interrupt Request (xIR) register bit is set.
2. The PIC performs a check on its internal Interrupt Mask (xINTMSK) register and In-Service (xISR) register. If this requesting interrupt is not masked off and if another interrupt of the same or higher priority is not in progress, the Master controller requests an interrupt from the CPU.
3. If the IF bit is set in the CPU's Flags register (via the STI instruction), the CPU acknowledges the interrupt. At this time, the PIC places the 8-bit interrupt vector of the currently active highest-priority interrupt request on the data bus, and the corresponding In-Service (xISR) register bit is set in the PIC. If the IF bit is disabled, the interrupt is ignored.

Note that the interrupt request must remain active at least until the first CPU acknowledge pulse occurs before it is considered as a valid interrupt request. If no interrupt request is active when the acknowledgement occurs, then the affected master or slave PIC returns the interrupt entry number associated with its IR7 input. However, in this circumstance no In-Service (xISR) register bit is set. This is known as the spurious interrupt condition and can be detected by the interrupt handler for priority level P22 (for the Master controller), P10 (for the Slave 1 controller), and P20 (for the Slave 2 controller). The Interrupt Request (xIR) register bit is always set for the duration of the interrupt request, regardless of whether it is a spurious or a valid interrupt request.

4. The CPU reads the interrupt vector and services the interrupt corresponding to the vector read during the acknowledgment.
5. Before further interrupts for the same priority level can be serviced, an EOI (end-of-interrupt must be issued to the PIC to reset the In-Service (xISR) register bit of the currently active interrupt. This can be done in one of two ways.
   - In automatic EOI (AEOI) mode, the In-Service (xISR) register bit is reset at the end of the acknowledgement cycle from the CPU. Note that AEOI mode does not support polling, and it can only be used in a master configuration, not in a slave configuration.
   - When AEOI is disabled, the interrupt handler must clear the In-Service (xISR) register bit by issuing a EOI command at the end of the interrupt service routine.

For an interrupt request coming from either one of the slave controllers, the slave controller generates an interrupt to the Master controller and asserts its corresponding Interrupt Request (xIR) register bit at the Master controller. The Master controller first determines if there is a higher priority interrupt that is currently being serviced. If there is not, it requests an interrupt from the CPU, as described in step 2. Otherwise, the higher priority interrupt service routine continues uninterrupted until another interrupt request is received from the PIC.

There are two ways in which an interrupt request from a slave controller differs from the interrupt sequence mentioned above. Steps 3–5 are similar in this case, but because the Interrupt Request (xIR) register bit set by the slave output is the highest priority interrupt, the Master controller now commands the slave controller to supply the interrupt vector to the CPU.

The other difference is that two EOIs are required: one to the Master controller to reset its highest priority In-Service (xISR) register bit (set by the interrupt request) and the other to the slave to reset its highest priority In-Service (xISR) register bit. The order of these two EOIs does not matter.

15.5.2 Interrupt Sources

The interrupt sources in the ÉlanSC520 microcontroller can be divided into four distinct categories:

- Externally-generated hardware interrupts from interrupt input pins
- Internally-generated hardware interrupts from peripherals
- Internally-generated hardware interrupts from interrupt trigger bits
- Software interrupts (generated with the INT instructions)

This section discusses all of these except software interrupts. Note that the first two hardware interrupt sources listed above can be mapped to the Am5$_x$86 CPU's NMI interrupt input. NMI is discussed in “Non-Maskable Interrupts and Routing” on page 15-14. Software interrupts work in the standard x86 fashion and are not discussed in this manual.

15.5.2.1 Hardware-Generated Interrupts

In the ÉlanSC520 microcontroller, there are 57 hardware interrupt sources:

- 23 can come from control bits in the Software Interrupt 16–1 Control (SWINT16_1) register (MMCR offset D08h) and the Software Interrupt 22–17/NMI Control (SWINT22_17) register (MMCR offset D0Ah).
- 15 can come from the 15 external interrupt pins (GPIRQ10–GPIRQ0 and $\overline{\text{INTA}}$–$\overline{\text{INTD}}$)

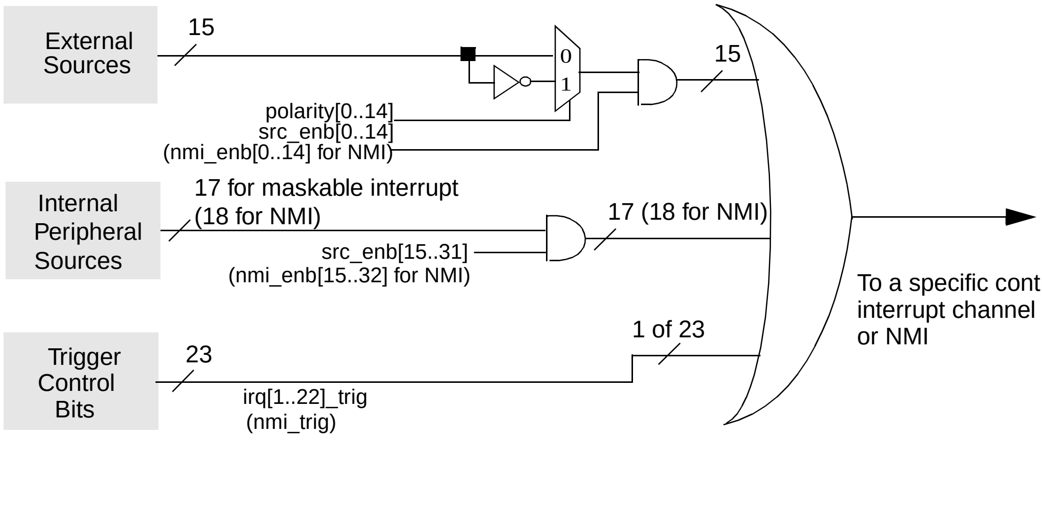
- 19 are generated from internal peripheral sources, including:
  - PCI host bridge/system arbiter (interrupt)
  - PCI host bridge (NMI)
  - SDRAM ECC single-bit error (interrupt)
  - SDRAM ECC multi-bit error (NMI)
  - Six timers (three GP timers and three PIT timers)
  - Two UARTS
  - GP-DMA buffer chaining
  - SSI
  - Watchdog timer
  - RTC
  - Write-protection violation in Programmable Address Region (PAR) register
  - AMDebug interface JTAG port receive or transmit activity
  - Floating point error

As shown in Figure 15-2 on page 15-9, of the19 internal peripheral sources:

- 17 can be used for maskable interrupts. The two sources that cannot be configured as a maskable interrupt are the SDRAM ECC multi-bit error NMI source and the PCI host bridge's separate NMI-only source.
- 18 can be routed to the Am5$_x$86 CPU's NMI input. The only source that cannot be used to generate an NMI is the SDRAM ECC single-bit error source.

The internal PCI host bridge and the SDRAM controller each generate a maskable interrupt source and an NMI interrupt source. However, only the internal PCI host bridge interrupt source can be mapped to generate either a maskable interrupt or an NMI. The SDRAM controller's maskable interrupt source cannot be mapped to generate an NMI.

Figure 15-2 Interrupt Sources



15.5.3 Interrupt Source Routing

Figure 15-3 on page 15-11 shows the implementation of the interrupt router. None of the interrupt enable signals are shared across the interrupt channels.

Each of the 32 hardware interrupt sources that come from peripherals (15 external and 17 internal) is fed into each of the 22 OR gates for the 22 interrupt channels. Each of the 22 OR gates also has an additional input from the one of the Software Interrupt x Control (SWINTx) registers.

When set, the interrupt trigger control bits cause their associated interrupt signals to be asserted at the PIC. These bits are under complete control of software. During normal operation, hardware does not set or clear these bits. A reset does clear these bits.

All incoming interrupt requests are arbitrated by the interrupt controllers based on the priority levels shown in Figure 15-1 on page 15-3, with the highest priority interrupt being serviced first. There is a mask bit associated with each of the 22 interrupt channels, providing a means for each interrupt channel to be masked individually.

Multiple interrupt requests can be shared on a common interrupt channel. This is discussed further in “Interrupt Sharing” on page 15-13.

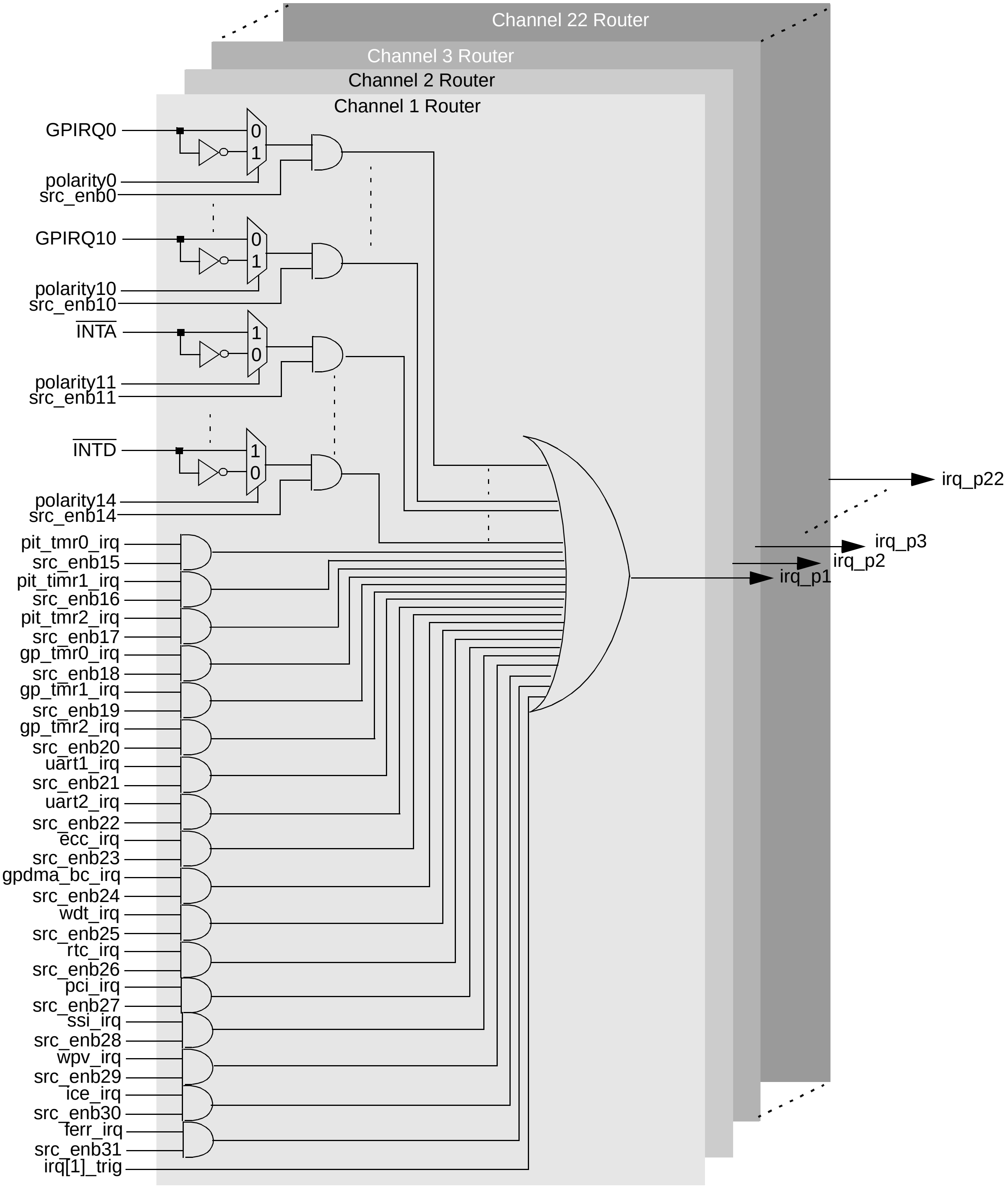
After reset, each of the interrupt sources must be mapped to the desired interrupt channel. This is usually done by the initialization software. It can be done during normal operation as well. The default power-on-reset state for these mapping bits is cleared; the programmer has to specifically map the individual interrupt requests to the desired interrupt channels.

15.5.3.1 Polarity Inversion of Interrupt Requests

Since each of the three individual interrupt controllers can only recognize either a Low-to-High edge-triggered or an active High level interrupt request, a programmable inversion is available for each of the 15 external interrupt requests to support active Low interrupt sources. For example, a PCI generated interrupt request that is active Low must be inverted within the ÉlanSC520 microcontroller prior to reaching the PIC channel to which it is mapped before the controller can recognize a valid interrupt request.

All internally-generated interrupt signals have the correct active High polarity and need no inversion via software. These internally-generated signals include those for the GP-DMA controller, PCI host bridge system arbiter, timers, UARTs, SSI, watchdog timer, SDRAM controller, RTC, AMDebug technology interface, floating-point error, and address mapping, as well as internally-generated NMI signals.

Figure 15-3 Interrupt Source Routing



Notes:

All the 32 hardware interrupt sources are common to all the 22 channel routers. The polarity control signal per external interrupt source is also common to all the 22 channel routers. The decoder for the enable signals is not shown; only the decoded representation of the signals is shown. Each channel router has its unique internally-generated hardware interrupt trigger, and only irq[1]_trig is shown for channel router 1.

15.5.3.2 PC/AT Compatibility

For PC/AT-compatible systems, the microcontroller hardware does not automatically map legacy ISA interrupt signals to their respective Slave 1 and Master controllers. The user's software must ensure that these interrupts are routed correctly to the appropriate PC/AT-compatible channels. Table 15-4 shows the interrupt channel assignment implemented in a PC/AT-compatible system.

Table 15-4 PC/AT Interrupt Channel Mapping

PC/AT-Compatible System		ÉlanSC520 Microcontroller	
IRQ	**I/O Device**	**Priority**	**Interrupt Source to Map**
IRQ0	System Timer 0	P1	Internal (PIT 0 interrupt)
IRQ1	Keyboard interface	P2	External via GPIRQx pin
IRQ2[1, 2]	Slave controller cascading	—	Cascaded from Slave 1 controller
IRQ3	UART 2	P11	Internal (UART 2 interrupt)
IRQ4	UART 1	P12	Internal (UART 1 interrupt)
IRQ5[1, 2]	Parallel port 2	P13	External via GPIRQx pin
IRQ6	Floppy disk controller	P21	External via GPIRQx pin
IRQ7	Parallel port 1	P22	External via GPIRQx pin
IRQ8	Real-time clock	P3	Internal (RTC interrupt)
IRQ9	Any 8- or 16-bit ISA device	P4	External via GPIRQx pin
IRQ10	Any 8- or 16-bit ISA device	P5	External via GPIRQx pin
IRQ11	Any 8- or 16-bit ISA device	P6	External via GPIRQx pin
IRQ12	Mouse interface	P7	External via GPIRQx pin
IRQ13	Numeric coprocessor	P8	Internal (floating point error interrupt)
IRQ14	Any 8- or 16-bit ISA device	P9	External via GPIRQx pin
IRQ15	Any 8- or 16-bit ISA device	P10	External via GPIRQx pin

Notes:

1. In the ÉlanSC520 microcontroller's PIC, interrupt channels 2 and 5 of the Master interrupt controller are hard-wired to the outputs of Slave 1 and Slave 2 interrupt controllers, respectively. The cascading of the slave controllers is fixed in order to simplify the system interrupt programming model.

2. When configured for PC/AT-compatible operation, the Slave 1 interrupt controller is cascaded and the Slave 2 controller is bypassed. In this configuration, IRQ2 is not available, and interrupt priority P13 acts as IRQ5. For configuration details see "PC/AT Compatibility" on page 15-12.

15.5.3.3 Floating Point Errors

The ÉlanSC520 microcontroller supports DOS-compatible floating point error handling via the standard Floating Point Error Interrupt Clear (FPUERRCLR) register (Port 00F0h), as in legacy PC/AT systems. PC/AT systems control floating point error reporting externally through the PC's interrupt controller, rather than through the internal CPU interrupt. In this case, an interrupt request is generated and typically routed to IRQ13 (although it is programmable via the ÉlanSC520 microcontroller's PIC). This allows an interrupt handler to write to the Floating Point Error Interrupt Clear (FPUERRCLR) register to clear the interrupt request and force the CPU's ignore numeric error ($\overline{\text{ignne}}$) signal active, thus enabling execution of floating-point instructions within the interrupt handler. Once the FPU error condition is cleared by the handler, the floating point error ($\overline{\text{ferr}}$) signal is deasserted, and the internal $\overline{\text{ignne}}$ signal is subsequently deasserted. The interrupt request and $\overline{\text{ignne}}$ signal are also cleared by a system reset.

15.5.3.4 Disabling the Slave Controllers

Each of the slave controllers can also be disabled via software, and interrupt requests can be easily routed to the associated interrupt channels of the Master controller. For example, if the Slave 1 controller is disabled, interrupt request irq_p3 that is hooked to the priority 3 input of the same controller is visible to the Master controller channel input IR2. Similarly, if the Slave 2 controller is also disabled, interrupt request irq_p13 is visible to the Master controller channel input IR5 (see Figure 15-1 on page 15-3). In other words, both of these interrupt requests would bypass the slave controllers. In this manner, a very simple interrupt configuration is realized via software, in which eight or fewer interrupt priorities can be implemented using just the Master controller. As such, only one EOI needs to be generated to minimize software overhead and improve latency of the interrupt cycle.

For more information about this topic, see “Software Considerations” on page 15-18.

15.5.4 Edge-Triggered or Level-Sensitive Interrupts

Each of the 22 interrupt priority levels can be configured as an edge-triggered or level-sensitive interrupt. This departs from the standard implementation of the individual interrupt controller, whereby a global bit for each controller determines the interrupt type for all the incoming interrupt requests.

In the ÉlanSC520 microcontroller, each individual interrupt controller is enhanced to provide this interrupt type recognition capability on a per channel basis. A bit is provided for each of the 22 interrupt channels for interrupt type programmability. The selection between global and per-channel interrupt mode is done via software. However, the original global bit is retained for the individual controllers, such that all of the interrupts for each device can be restored globally as either edge- or level-sensitive. This is useful for PC/AT compatibility, especially for the Master and Slave 1 controllers.

Regardless of whether the controller is programmed for edge-sensitive or level-sensitive mode, the interrupt request source must continue asserting the interrupt request until the CPU acknowledges the interrupt. Because this acknowledgment is not viewable externally to the ÉlanSC520 microcontroller, it is recommended that external interrupt sources provide a mechanism through which the interrupt service routine can deassert the interrupt request via software.

15.5.5 Interrupt Sharing

The controllers support sharing interrupt inputs from multiple interrupt sources. Interrupt sharing is applicable to all internal and external interrupt sources. To put it simply, since OR gates are used to map interrupt sources to interrupt channels, it is easy to map more than one interrupt source to a single interrupt channel. This is shown in Figure 15-3.

Level-sensitive interrupt sharing is typically implemented by tying multiple interrupt outputs using an open drain or open collector output to a single interrupt input pin. Of course, this can be done externally to the ÉlanSC520 microcontroller in the conventional manner.

However, interrupt sharing can also be easily configured internally to the microcontroller, merely by mapping multiple interrupt sources to the same interrupt channel. The channel’s OR gates inherently “share” the interrupt channel among multiple interrupts. In this scenario, an interrupt-pending status bit must be implemented in each device. All internal peripherals have interrupt status bits.

Since programmable inversion of the interrupt signal is available, the external device can generate an interrupt to the ÉlanSC520 microcontroller by either driving the interrupt request line Low and allowing a pullup resistor to generate the rising edge or by actively driving the line Low from its default High inactive state through a pullup resistor (as in PCI interrupt generation).

Sharing edge-triggered interrupts in the ÉlanSC520 microcontroller is not recommended.

For more information about this topic, see “Software Considerations” on page 15-18.

15.5.6 Non-Maskable Interrupts and Routing

A unique feature of the ÉlanSC520 microcontroller’s PIC is its ability to route most of its hardware interrupt sources via software to generate a non-maskable interrupt (NMI) to the CPU.

- With the exception of the internally-generated ECC interrupt from the SDRAM controller, all the other interrupt sources can be routed to the Am5$_X$86 CPU’s NMI input.
- The PCI host bridge and SDRAM controller each generate a separate and distinct NMI interrupt source to the PIC. The interrupt source can only generate an NMI and not a maskable interrupt to the CPU.

There are 34 interrupt sources for NMI generation to the CPU:

- 15 external interrupts
- 18 internally-generated interrupts
- 1 software NMI source

Figure 15-4 on page 15-15 shows the logical implementation of NMI generation in the ÉlanSC520 microcontroller.

15.5.6.1 Sharing NMIs

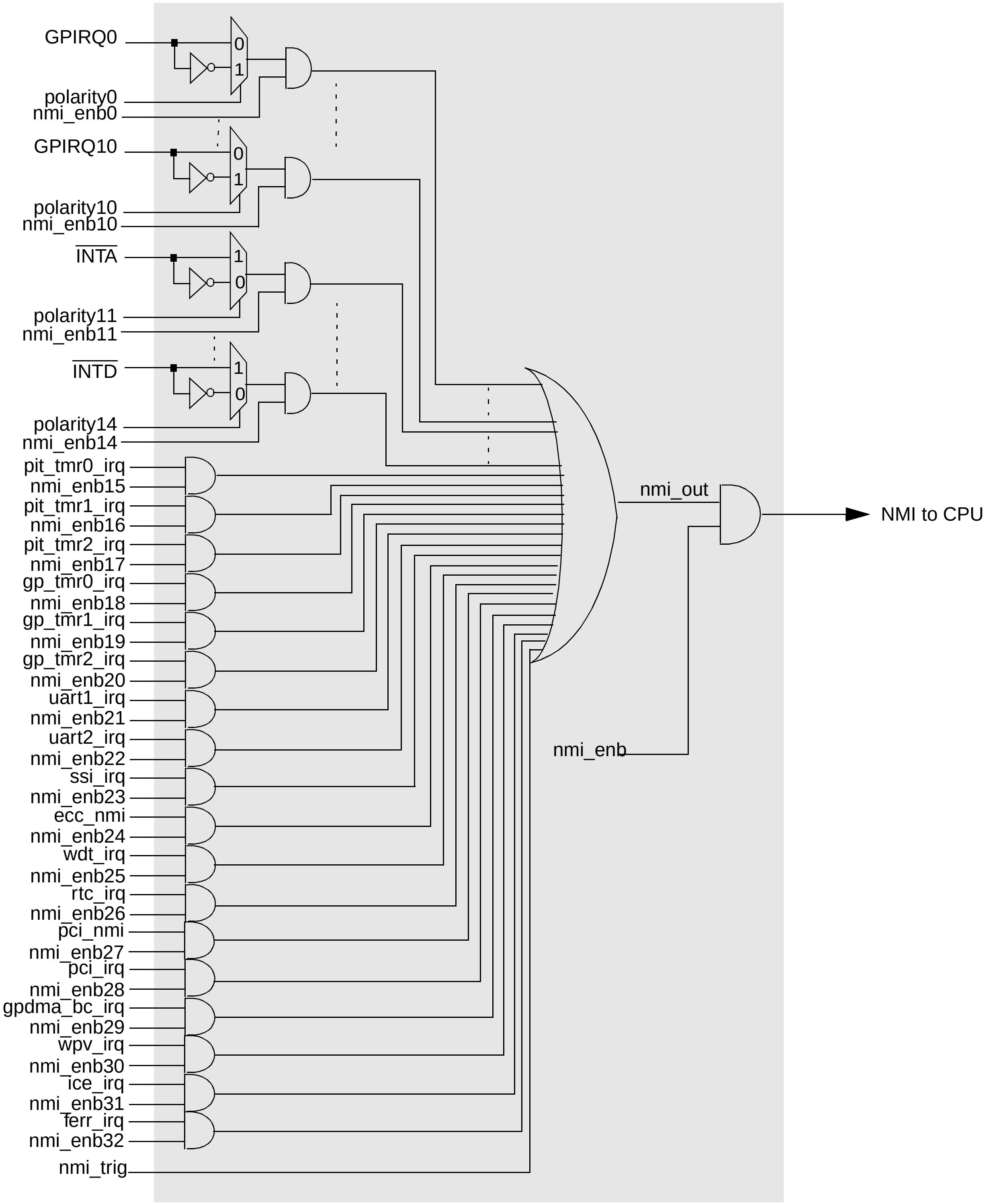
NMIs can be shared in the ÉlanSC520 microcontroller. NMI sources are routed logically to an OR gate, as shown in Figure 15-4 on page 15-15.

Each individual interrupt source is gated by an enable signal to selectively allow it to be shared with the other interrupt sources. Each of these enable signals is controlled via the Interrupt Mapping (xMAP) registers and is enabled by programming its interrupt routing bits to 11111b. An NMI Enable (NMI_ENB) bit in the Interrupt Control (PICICR) register (MMCR offset D00h) provides the mechanism to prevent all NMIs from reaching the CPU. This bit has been moved from the PC/AT-compatible location (see “Legacy NMI Enable Bit Moved” on page 20-10 for more details). NMIs are disabled on system and soft reset and must be enabled via setting the NMI_ENB bit before use.

It is recommended that sharing NMIs be done using level-sensitive NMIs only. All NMIs should be treated similarly to the maskable interrupt sources. All NMIs once asserted should remain asserted until cleared by software. The NMI_DONE bit located in the Interrupt Control (PICICR) register facilitates NMI sharing. This bit is visible to all NMI handlers, and the currently executing NMI handler should clear the NMI source prior to asserting the NMI_DONE bit. NMI handler software should write a 1 to the self-clearing NMI_DONE bit immediately before executing the IRET instruction to exit from the handler. Setting the NMI_DONE bit deasserts the NMI signal to the CPU for a brief time before allowing any other pending NMI requests to be serviced, in order to satisfy NMI timing requirements of the CPU.

Sharing edge-triggered NMIs in the ÉlanSC520 microcontroller is not recommended.

Figure 15-4 NMI Routing



Notes:

The polarity control signal per external interrupt source is common to those used across the channel routers. The gating NMI enable bits for each source are controlled via the interrupt mapping registers. The NMI conditioning logic to implement NMI sharing is not shown in this figure

15.5.7 Priority Types

Each individual interrupt controller prioritizes interrupt requests by their IR number, as shown in Figure 15-1 on page 15-3. This places IR0 as the highest priority and IR7 the lowest, which is the default ordering.

In a cascaded environment, the full 22 priority level is as shown in Figure 15-1, with P1 being the highest and P22 the lowest priority. As a result, if two or more interrupt requests appear simultaneously, the higher priority interrupt is serviced first and the lower priority interrupt is pending.

The interrupt controller supports nested interrupts. The depth level of nesting affects system performance, and the programmer must implement this with care.

The interrupt controller also supports specific and automatic rotation types.

- In *specific rotation*, the lowest priority can be programmed in the individual controller, thus fixing all the other priorities.
  - For example, in Figure 15-1, if P5 is programmed to be the lowest priority, then P6 of Slave 1 controller would be the highest priority within this controller.
  - In this case, the priority order starting with the highest priority level would follow as: P1–P2 (Master), P6–P10 (Slave 1), P3–P5 (Slave 1), P11–P12 (Master), P13–P20 (Slave 2), P21–P22 (Master). This is assuming that the Master and Slave 2 controllers are each programmed with IR7 as the lowest priority.
  - In fact, the implementation shown in Figure 15-1 is of a *fixed priority* scheme (with priority ordering of P1–P22) and is a variation of the specific rotation type.
- In *automatic rotation* scheme, all priority levels within the controller are treated as equal.
  - In this mode, an interrupt request after being serviced receives the lowest priority, so that the same device requesting an interrupt is queued.
  - In the worst scenario, the device would have to wait until each of the seven other devices is serviced at most one time.

15.5.8 Configuration Information

15.5.8.1 Programming

The initialization sequence of the PIC consists of writing a sequence of two to four bytes to each controller. The first initialization byte is written to the lower address of the controller, (020h for the Master, 0A0h for Slave 1, and 024h for Slave 2), and all subsequent initialization bytes are written to the upper address of the controller (021h for the Master, 0A1h for Slave 1, and 025h for Slave 2).

1. The first initialization byte, the Initialization Control Word 1 (xICW1) register, notifies the controller that an initialization sequence is starting. This register also controls the type of interrupt-triggering (edge- or level-sensitive), whether the controller is in a cascaded environment or alone, and whether the fourth initialization byte, the Initialization Control Word 4 (xICW4) register, is required or not.
2. The second byte, the Initialization Control Word 2 (xICW2) register, contains the vector offset for the controller. For PC/AT-compatible interrupts, xICW2 should be 08h for the Master controller and 70h for the Slave 1 controller (Slave 2 is not used in PC/AT-compatible systems).
3. The third byte, the Initialization Control Word 3 (xICW3) register is written only if xICW1 indicates that the controller is in a cascaded environment. For the Master controller, it

identifies which IR inputs are hooked up to slave controllers. For the slave controllers, it identifies the IR pin on the master to which that particular slave is connected.

It is important to note that the ÉlanSC520 microcontroller's PIC can be configured as a stand-alone master controller, one slave cascade (either Slave 1 or Slave 2), or cascading with both slave controllers.

– To configure it as a stand-alone Master controller where 8 or fewer interrupt requests are available to the user, bits 2 and 5 must be cleared to 0 in the Master PIC Initialization Control Word 3 (MPICICW3) register (Port 0021h).

– To configure it as a Slave 1 only cascade, the S2 and S5 bits must be set and cleared respectively in the Master PIC Initialization Control Word 3 (MPICICW3) register (Port 0021h).

– For Slave 2 cascade only configuration, the S2 and S5 bits must be cleared and set respectively in the Master PIC Initialization Control Word 3 (MPICICW3) register (Port 0021h).

– To configure cascading using both the slave controllers, the S2 and S5 bits must be set in the Master PIC Initialization Control Word 3 (MPICICW3) register (Port 0021h).

4. Finally, the Initialization Control Word 4 (xICW4) register (written only if indicated in the Initialization Control Word 1 (xICW1) register) controls whether EOIs are generated manually or automatically. It also contains some bits that must always be set in the ÉlanSC520 microcontroller.

Note that some parameters in the PIC configuration registers are fixed based on the way the controllers are arranged in the ÉlanSC520 microcontroller.

For example, the Slave 1 PIC Initialization Control Word 3 (S1PICICW3) register (Port 00A1h) always contains 2d to indicate that Slave 1 is hooked up to IR2 on the Master controller.

For those configuration parameters that are not fixed, software that initializes the controllers must be very careful to accurately reflect the correct arrangements of the controllers, as shown in Figure 15-1 on page 15-3.

For example, if neither Slave controller is being bypassed, the Master PIC Initialization Control Word 3 (MPICICW3) register (Port 0021h) should contain 24h (or 00100100b) to indicate that slave controllers are hooked up to its IR2 and IR5 signals.

After the interrupt controllers are initialized, any subsequent reads or writes to ports 021h, 0A1h, or 025h access the Interrupt Mask (xINTMSK) register of the Master, Slave 1, or Slave 2 controllers. The Operation Control Word 2 (xOCW2) and Operation Control Word 3 (xOCW3) registers are accessed by writing to the appropriate ports, 020h, 0A0h, or 024h. The controllers can be configured in various modes using these registers.

5. Initializing the Interrupt Mask (xINTMSK) register provides the masking of the interrupt requests on a per channel basis.
6. Writing to the Operation Control Word 2 (xOCW2) register configures the various rotation and EOI modes.
7. Finally, the Operation Control Word 3 (xOCW3) register configures the different mask modes, controls reading of the In-Service (xISR) register or the Interrupt Request (xIR) register, and whether the controller is to be used by software to perform polling.

The rest of the non-controller specific registers are programmed next. This includes programming the routing of the various interrupt sources to the appropriate priority level or

NMI (as indicated in Figure 15-1) polarity inversion of the interrupt sources if needed, different interrupt mode per channel, global interrupt mode enables, or master NMI enable. These registers are listed in Table 15-2 on page 15-4.

It is recommended that EOIs be issued for all the channels prior to using the Set Interrupt-Enable Flag (STI) instruction. This is to clear all spurious In-Service (xISR) register bits that are potentially set during the initialization phase before enabling the CPU to accept interrupt requests.

15.5.8.2 PC/AT Configuration

To configure the ÉlanSC520 microcontroller's PIC to be PC/AT-compatible, the same configuration sequence detailed in "Programming" on page 15-16 is observed with the following exceptions:

1. The SNGL bit must be cleared to 0 in the Master PIC Initialization Control Word 1 (MPICICW1) register (Port 0020h).
2. The S2 and S5 bits must be set to 1 and cleared to 0, respectively, in the Master PIC Initialization Control Word 3 (MPICICW3) register (Port 0021h).
3. The M_GINT_MODE and S1_GINT_MODE bits must be set to 1 in the Interrupt Control (PICICR) register (MMCR offset D00h).
4. The base interrupt vector numbers 08h and 70h must be written for the Master and Slave 1 PIC, respectively, to the Master PIC Initialization Control Word 2 (MPICICW2) register (Port 0021h) and the Slave 1 PIC Initialization Control Word 2 (S1PICICW2) register (Port 00A1h). This correctly programs the T7–T3 bit field in those registers, which corresponds to bits 7–3 of the 8-bit base interrupt vector number. This also clears the A10–A8 bit field (bits 2–0), which should be 0 for PC-AT-compatible interrupts.
5. The SFNM and AEOI bits must be cleared to 0 in the Master PIC Initialization Control Word 4 (MPICICW4) register (Port 0021h), and the SFNM bit must be cleared to 0 in the Slave 1 PIC Initialization Control Word 4 (S1PICICW4) register (Port 00A1h).
6. Any interrupt sources used in the system must be mapped to appropriate interrupt priorities via the interrupt mapping registers. Table 15-4 on page 15-12 correlates the PC/AT IRQs and I/O devices to the ÉlanSC520 microcontroller's interrupt priorities.

In this case, only the Slave 1 controller is cascaded to the Master controller via input IR2. The Slave 2 controller is logically removed from the Master controller, and the highest priority channel originally hooked to the former is now automatically routed to input IR5 of the latter, thereby preserving the architecture of the PC/AT interrupt controller.

15.5.9 Software Considerations

15.5.9.1 Interrupt Sharing

Interrupt sharing increases system complexity and involves more software overhead. Thorough understanding of performance implications to a system implementing interrupt sharing is needed. For multiple interrupt requests sharing a line, the system designer needs to be fully aware of the latency involved and the implications in interrupt sharing.

For example, in the worst case scenario, it may take an unacceptably long amount of time before the CPU is able to service the first interrupt request hooked at the very beginning of the interrupt chain (created during the interrupt hooking process). This problem is compounded further if one or more interrupt requests before it are still pending. This can be alleviated somewhat by prioritizing or re-ordering the more critical interrupt table entries later in the chain during the interrupt hooking process.

Although level-sensitive interrupt sharing generally works well, implementing edge-sensitive interrupt sharing is not recommended.

15.5.9.2 Disabling the Slave Controllers

The ÉlanSC520 microcontroller's PIC has the flexibility to allow removal of either or both the slave controllers logically from the cascade chain via software (see S2 and S5 bits in the Master PIC Initialization Control Word 3 (MPICICW3) register). Disabling one or more of the slave controllers allows configuring a system with fewer than 9 or 16 interrupt channels.

Although the slave controllers are hard-wired to the Master controller, bypassing the slave controllers via software during configuration could typically result in a more efficient interrupt system, whereby only the Master controller needs to be initialized and configured. With this configuration, only one non-specific EOI needs to be generated, instead of two, at the end of the interrupt service routine.

When either of the slave controllers is disabled, the highest priority interrupt hooked to the slave controllers is routed automatically to channels 2 and 5 of the Master controller, respectively. As such, the programmer needs to be aware that mapping interrupts to the other seven lower priority channels of the slave controller inhibits propagation of these interrupt requests to the Master controller. Figure 15-1 on page 15-3 shows this implementation in the ÉlanSC520 microcontroller's PIC.

15.5.9.3 Detecting Invalid Interrupt Requests

If an interrupt request does not remain active long enough for the corresponding In-Service (xISR) register bit to be set (a non-deterministic amount of time), the request is considered a spurious interrupt pulse.

Spurious pulses on any of the interrupt requests cause the interrupt handler associated with the IR7 input of the affected controller to be executed (priority level P22 for the Master controller, P10 for the Slave 1 controller, or P20 for the Slave 2 controller). The Interrupt Request (xIR) register bit is always set for the duration of the interrupt request, regardless of whether it is a spurious or a valid interrupt request.

The interrupt handler associated with IR7 is required to check the In-Service (xISR) register bit to determine if a valid interrupt request generated the interrupt. If the In-Service (xISR) register bit is set, then a valid interrupt request is generated, and the normal routine is executed. Otherwise, a spurious interrupt is identified and the interrupt routine exits.

In other words, spurious pulses on the interrupt requests that are shorter than a non-deterministic duration can be filtered out by software that checks the In-Service (xISR) register bit. Longer spurious pulses can only be detected if all interrupt sources hooked onto a given priority level provide their own status bits.

15.5.9.4 Floating Point Unit Error Handling

To implement DOS-compatible floating-point error handling, such as is used in legacy PC/AT systems, the Numeric Error (NE) bit in the CPU's Control 0 (CR0) register must be cleared. If the NE bit is set, an exception 16 will be generated instead of an external interrupt request via the ÉlanSC520 microcontroller's programmable interrupt controller. See the *Am486® DX/DX2 Microprocessor Hardware Reference Manual*, 1994 (order #17965), for further details on the floating point unit.

15.6 INITIALIZATION

The programmable interrupt controller responds only to system reset.

The Slave 1, Slave 2, and Master interrupt controllers are not affected by system reset. The interrupt controller direct-mapped registers, once configured, retain their values during a system reset. However, all other configuration registers default to their power-on reset states when a system reset occurs. The interrupt router is reset, such that the interrupt requests are gated off. This effectively disables all interrupt requests from reaching the CPU.

At system reset, the PIC is disabled.

1. Configure the Master, Slave 1, and Slave 2 controllers as described in “Configuration Information” on page 15-16. Mask all interrupts.
2. Place an interrupt service routine at the locations corresponding to the interrupt priority levels to be supported.
3. Enable the desired priority levels by mapping the interrupts sources to the interrupt levels in the interrupt router and unmasking the interrupt in the corresponding interrupt controllers. Set the IF bit in the CPU’s Flags register using the STI instruction. (NMIs are disabled on system and soft reset and must be enabled via NMI_ENB bit before use).

CHAPTER

16 PROGRAMMABLE INTERVAL TIMER

16.1 OVERVIEW

The ÉlanSC520 microcontroller includes four separate timer modules: a PC/AT-compatible programmable interval timer (PIT) with three timers, three general-purpose (GP) timers, a software timer, and a watchdog timer. The programmable interval timer is described in this chapter. The general-purpose timers are described in Chapter 17. The software timer is described in Chapter 18. The watchdog timer is described in Chapter 19.

The programmable interval timer (PIT) on the ÉlanSC520 microcontroller includes three separate timers, designed to provide PC/AT compatibility.

Features of the PIT include:

- Three 16-bit timers, or *channels*
- Clock source from either 1.1892-MHz source or an external pin. The same clock is routed to all three channels.
- One interrupt output for each channel
- One external output pin for PIT Channel 2
- Several modes of operation, including:
  - Interrupt on terminal count
  - Hardware-retriggerable one-shot
  - Rate and square wave generation
  - Hardware- and software-retriggerable strobe

16.2 BLOCK DIAGRAM

Figure 16-1 shows a block diagram of the programmable interval timer.

16.3 SYSTEM DESIGN

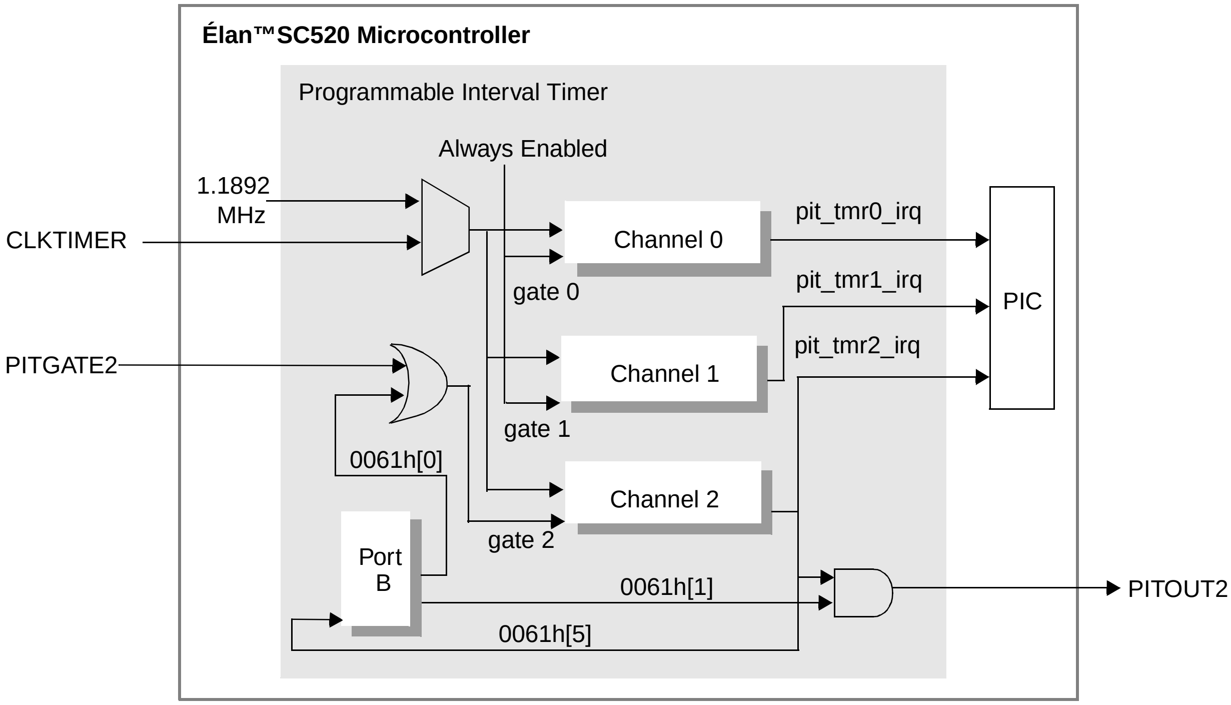
Table 16-1 shows the PIT signals shared with other interfaces. The pinstrap function associated with the PITOUT2 pin is sampled only as a result of PWRGOOD assertion and does not affect the PIT function of this pin, so it is not shown in this table. When enabled, the multiplexed signals shown in Table 16-1 either disable or alter any other function that uses the same pin.

Note: *The CFG3 pinstrap associated with PITOUT2 is used for an AMD internal test mode. Do not pull this pin High during reset.*

Table 16-1 Programmable Interval Timer Signals Shared with Other Interfaces

Default Signal	Alternate Function	Control	Register
CLKTIMER	CLKTEST	CLK_PIN_DIR	Clock Select (CLKSEL) register (MMCR offset C26h)
PITGATE2	$\overline{\text{GPCS3}}$	GPCS3_SEL	Chip Select Pin Function Select (CSPFS) register (MMCR offset C24h)

Figure 16-1 Programmable Interval Timer Block Diagram



Notes:

Port B is addressed at 0061h in I/O space.

16.4 REGISTERS

The programmable interval timer (PIT) is configured using the registers listed in Table 16-2 and Table 16-3. The direct-mapped System Control Port B register is used to provide PC/AT-compatible PIT functionality.

Table 16-2 Programmable Interval Timer Configuration Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
Chip Select Pin Function Select	CSPFS	C24h	$\overline{\text{GPCS3}}$ or PITGATE2 function select
Clock Select	CLKSEL	C26h	CLKTIMER[CLKTEST] pin enable, clock output select options (PIT), CLKTIMER select (input clock for PIT)
PIT 0 Interrupt Mapping	PIT0MAP	D20h	PIT 0 interrupt mapping
PIT 1 Interrupt Mapping	PIT1MAP	D21h	PIT 1 interrupt mapping
PIT 2 Interrupt Mapping	PIT2MAP	D22h	PIT 2 interrupt mapping

Table 16-3 Programmable Interval Timer Configuration Registers—Direct-Mapped

Register	Mnemonic	I/O Address	Function
PIT Channel 0 Count	PIT0CNT	0040h	Current count value for Channel 0
PIT Channel 1 Count	PIT1CNT	0041h	Current count value for Channel 1
PIT Channel 2 Count	PIT2CNT	0042h	Current count value for Channel 2
PIT 0 Status	PIT0STA	0040h	Counter mode status, null count, output state, latch command or read/write control setting, and BCD setting for Channel 0
PIT 1 Status	PIT1STA	0041h	Counter mode status, null count, output state, latch command or read/write control setting, and BCD setting for Channel 1
PIT 2 Status	PIT2STA	0042h	Counter mode status, null count, output state, latch command or read/write control setting, and BCD setting for Channel 2
PIT Mode Control	PITMODECTL	0043h	PIT counter select or read-back command, read/write control or counter latch command, counter mode, BCD select
PIT Counter Latch Command	PITCNTLAT	0043h	Control to latch current count of the selected channel for read-back
PIT Read-Back Command	PITRDBACK	0043h	Control to latch status and current count of each channel for read-back
System Control Port B	SYSCTLB	0061h	PITOUT2 signal enable, status, and Channel 2 gate input control

16.5 OPERATION

The programmable interval timer provides three different timers, or *channels*, and six modes of operation. Not all channels support every mode.

16.5.1 PIT Channel 0

PIT Channel 0 is used for generating interrupt requests. PIT Channel 0 can be configured to assert interrupt priority P1 (IRQ0) to allow it to operate in PC/AT-compatible mode. See Chapter 15, “Programmable Interrupt Controller”, for more information on interrupt steering.

PIT Channel 0 can be configured to assert IRQ0 to allow it to operate in PC/AT-compatible mode. The gate line is tied High such that PIT Channel 0 operates in four modes only, modes 0, 2, 3, and 4. Mode 0 is typically used for interrupts, because it remains in the High state until restarted.

16.5.2 PIT Channel 1

The PIT Channel 1 is used as a general-purpose timer. Its output is hardwired internally to drive an input of the programmable interrupt controller. See Chapter 15, “Programmable Interrupt Controller” for more information on interrupt steering.

The gate line is tied High such that PIT Channel 1 also operates in four modes only, modes 0, 2, 3, and 4. Mode 0 is typically used for interrupts, because it remains in the High state until restarted.

16.5.3 PIT Channel 2

The gate line for PIT Channel 2 is controlled by the PIT_GATE2 bit in the System Control Port B (SYSCTLB) register (Port 0061h) or the external input pin PITGATE2. PITGATE2 is a multiplexed pin; if it is disabled, the gate line is controlled only by the PIT_GATE2 bit in the System Control Port B (SYSCTLB) register.

The output of the PIT Channel 2 is hardwired internally on the ÉlanSC520 microcontroller to drive an input of the programmable interrupt controller and can be read in the PIT_OUT2_STA bit of the System Control Port B (SYSCTLB) register (Port 0061h). See Chapter 15 for more information on interrupt steering. The output goes to the external output pin PITOUT2 when the PIT_OUT2_ENB bit is set in the System Control Port B (SYSCTLB) register.

PIT Channel 2 works in all six modes.

16.5.4 Operating Modes

The modes for the each PIT channel are specified in Counter Mode (CTR_MODE) bit field in the PIT Mode Control (PITMODECTL) register (Port 0043h).

16.5.4.1 Mode 0: Interrupt on Terminal Count

In interrupt on terminal count mode,

1. When the initial count is loaded into the PIT Channel x Count (PITxCNT) register, the output of the counter goes Low.
2. The count value decrements by one for each input clock pulse if the gate input is held High.
3. If the gate input is held Low, count maintains its value until after a rising edge of clock after the Gate goes High again.
4. The output of the counter is initially Low and will remain Low until the counter reaches zero. The output then goes High until a new count or a new mode 0 control word is loaded into the Counter.

16.5.4.2 Mode 1: Hardware-Retriggerable One-Shot

In hardware-retriggerable one-shot mode:

1. After an initial count is loaded into the PIT Channel 2 Count (PIT2CNT) register, a rising edge on the gate signal causes the output of the counter to go Low.
2. The count value decrements with each successive clock pulse.
3. The gate trigger begins the one-shot pulse with the output going Low until the count reaches zero.
4. Output then goes High and remains High until the clock pulse after the next trigger.

The duration of the one-shot pulse is:

Duration = Initial count * Period of the clock input

This mode is called *hardware-retriggerable* because, once an output pulse has started, if a rising edge is experienced at the gate input, the counter is reloaded with the initial count and the pulse continues until the new count expires. This mode is supported on PIT Channel 2 only.

16.5.4.3 Mode 2: Rate Generator

When programmed in rate generator mode, the counters operate as divide by n counters, where n is the initial count.

1. The output signal starts off High until the initial count is decremented to one.
2. The output then goes Low for one clock pulse and goes High again.
3. The counter is reloaded with the initial count, and the counting sequence is repeated.

There appears one clock pulse at the output for every n clock cycles.

By default, PC/AT-compatible systems program PIT Channel 0 for this mode.

16.5.4.4 Mode 3: Square Wave Mode

In square wave mode:

1. The output of the counter has a 50% duty cycle whenever the counter is loaded with an even count. Initially the output is High.
2. The count decrements by two with each clock cycle when the gate is held High.
3. When the count reaches zero, the output toggles state, the initial count is reloaded, and the sequence is repeated.

The period of the output signal is:

Period = Input clock period * Initial count loaded into the counter

If the initial count is an odd number, the output is High for (n+1)/2 cycles and is Low for (n–1)/2 cycles.

By default, PC/AT-compatible systems program PIT channels 1 and 2 to use this mode to drive DRAM refresh and the speaker, respectively.

16.5.4.5 Mode 4: Software-Triggered Strobe

In software-triggered mode:

1. The counter automatically begins to decrement one clock pulse after it is loaded with the initial count through software. The output signal is initially High.
2. The count decrements at the rate set by the clock input signal.
3. At the moment the terminal count is reached, the counter generates a single strobe pulse on the output for one clock pulse duration.
4. If the counter is loaded with a count of n, then a strobe pulse is produced at the output after n+1 clock cycles.

16.5.4.6 Mode 5: Hardware-Triggered Strobe

In hardware-triggered mode:

1. Counting begins on a Low-to-High transition of the gate signal.
2. The output remains High until the count has expired.
3. The output goes Low for one clock cycle and goes High again.
4. After writing the control word and the initial count, the counter is loaded at the next clock pulse after the trigger.

The strobe pulse occurs n+ 1 clock pulses after the Low to High transition (trigger) on the Gate input. This count sequence is retriggerable.

In this mode, the counter output behaves just as in mode 4, except for the triggering mechanism. This mode is supported on PIT Channel 2 only.

16.5.5 Clocking Considerations

The PIT clock source can be either the derived 1.1892-MHz PIT clock or an external pin. This is configured in the CLK_PIN_DIR bit in the Clock Select (CLKSEL) register (MMCR offset C26h).

The PIT clock on the ÉlanSC520 microcontroller does not run at 1.19318 MHz, as in PC/AT-compatible systems. See Section 16.5.7.1 for more information.

16.5.5.1 Internal Clock

Table 16-4 PIT Internal Clock Source

Internal Clock Source	Resolution Range	Duration
1.1892 MHz	841.61 ns–55.1 ms	16-bit duration

16.5.5.2 External Clock

A separate external clock input pin, CLKTIMER, is provided to the PIT. Table 16-5 specifies the external clock source frequency range for the CLKTIMER input for the PIT.

Table 16-5 PIT External Clock Source

External Clock Source	Frequency Range
CLKTIMER	1.18125–1.20511 MHz

16.5.6 Interrupts

Each PIT channel provides its own interrupt to the programmable interrupt controller (PIC). See Chapter 15, for more information on interrupt steering.

For the PIT, the interrupt request is always generated on terminal count, and it is basically the output signal of the PIT channel. The pattern of the interrupt request signal depends on the programmed operation mode in the channel. Modes 0 and 1 generate a Low-to-High signal on terminal count, and they are usually used as interrupt sources.

16.5.7 Software Considerations

16.5.7.1 Using the PIT Clock Source in PC/AT-Compatible Systems

In PC/AT-compatible systems, system boot code usually programs the PIT Channel 0 Count (PIT0CNT) register (Port 0040h) to a value of FFFFh. It relies on this periodic interrupt in order to keep accurate time of day. Since the timer clock source is 1.1892 MHz in the ÉlanSC520 microcontroller, the priority P1 interrupt (IRQ0) is generated every 55.11 ms. Historically the PIT clock source has been 1.19318 MHz, and this translates into an interrupt generation rate of 54.93 ms. This interrupt generation rate difference causes the time-keeping function of a PC/AT-compatible system to be inaccurate.

There are two possible ways to address this issue. One method involves modifying the PIT Channel 0 Count (PIT0CNT) register via the system boot code. The second method involves driving the PIT from an external clock source.

- Modifying the PIT Channel 0 Count (PIT0CNT) register—If the system boot code programs this register to a value of FF2Bh, the desired interrupt generation rate of 54.93 ms can be achieved.

- Driving an external 1.19318-MHz clock on the CLKTIMER pin—A system designer can choose to supply an external clock source frequency of 1.19318 MHz on the CLKTIMER pin. This pin must be specifically configured for this functionality by the system boot code during the system boot process, prior to configuring the PIT. The CLK_PIN_DIR bit in the Clock Select (CLKSEL) register (MMCR offset C26h) is used for this purpose.

16.6 INITIALIZATION

At system reset, the state of the PIT itself is undefined. The mode, count value, and output of all channels are undefined. Each PIT channel must be programmed before it can be used. To prevent superfluous interrupts, each PIT channel must be configured prior to enabling interrupts on the Am5$_{X}$86 CPU.

1. Write a control word into the PIT Mode Control (PITMODECTL) register (Port 0043h).
2. Write an initial count into the PIT Channel x Count (PITxCNT) register of the PIT channel being programmed. The control word determines the format of the initial count.

CHAPTER

17 GENERAL-PURPOSE TIMERS

17.1 OVERVIEW

The general-purpose (GP) timers are intended for most generic timing or counting applications, such as generating periodic interrupts and measuring or counting external events.

Features of the general-purpose timers include:

- Three 16-bit timers
- Two-stage cascading of timers, to allow a maximum of two 32-bit timer/counter elements
- Clock source from the system clock (33 MHz), an external pin, or a derived prescale clock. The external pin and pre-scale clock are available for GP Timer 0 and GP Timer 1 only. The maximum clock is 33 MHz/4.
- One external input pin for each timer for GP Timer 0 and GP Timer 1, used for external event capture, pulse count, and counter reset/reload
- One external output pin for GP Timer 0 and GP Timer 1
- One interrupt output for each timer
- Several modes of operation, including:
  - Interrupt on terminal count
  - Hardware retrigger mode
  - Rate and square wave generation
  - Continuous mode

17.2 BLOCK DIAGRAM

Figure 17-1 shows a block diagram of the general-purpose timers.

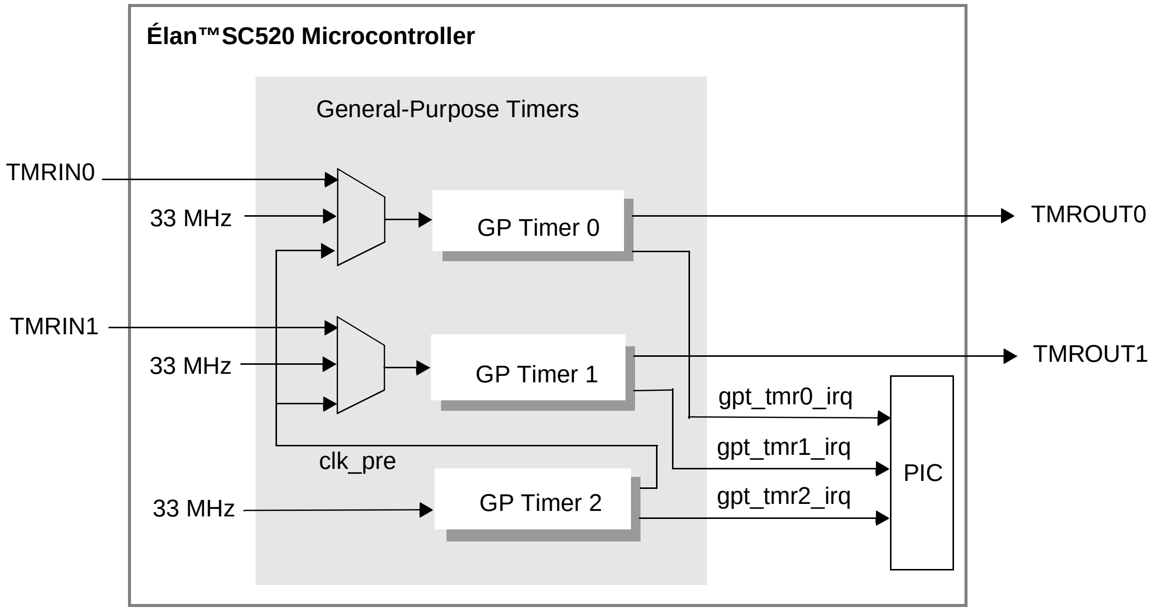
17.3 SYSTEM DESIGN

Table 17-1 shows the general-purpose timer signals shared with other interfaces. When enabled, the multiplexed signals shown in Table 17-1 either disable or alter any other function that uses the same pin.

Table 17-1 General-Purpose Timer Signals Shared with Other Interfaces

Default Signal	Alternate Function	Control Bit	Register
TMROUT0	$\overline{\text{GPCS7}}$	GPCS7_SEL	Chip Select Pin Function Select (CSPFS) register (MMCR offset C24h)
TMROUT1	$\overline{\text{GPCS6}}$	GPCS6_SEL	
TMRIN0	$\overline{\text{GPCS5}}$	GPCS5_SEL	
TMRIN1	$\overline{\text{GPCS4}}$	GPCS4_SEL	

Figure 17-1 General-Purpose Timers Block Diagram



17.4 REGISTERS

The general-purpose timers include the memory-mapped registers listed in Table 17-2.

Table 17-2 General-Purpose Timer Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
Chip Select Pin Function Select	CSPFS	C24h	TMROUTx, TMRINx, or $\overline{\text{GPCSx}}$ pin function select
GP Timers Status	GPTMRSTA	C70h	Interrupt status and clear for all three GP timers
GP Timer 0 Mode/Control	GPTMR0CTL	C72h	GP Timer 0 enable, permit Enable bit write, interrupt enable, maxcount register in use, maximum count, retrigger, internal clock source prescaler, external clock source, alternate compare mode, continuous mode
GP Timer 0 Count	GPTMR0CNT	C74h	Current count value
GP Timer 0 Maxcount Compare A	GPTMR0MAXCMPA	C76h	Maxcount value A to compare with current count
GP Timer 0 Maxcount Compare B	GPTMR0MAXCMPB	C78h	Maxcount value B, used in the alternate mode
GP Timer 1 Mode/Control	GPTMR1CTL	C7Ah	GP Timer 1 enable, permit Enable bit write, interrupt enable, maxcount register in use, maximum count, retrigger, internal clock source prescaler, external clock source, alternate compare mode, continuous mode
GP Timer 1 Count	GPTMR1CNT	C7Ch	Current count value

Table 17-2 General-Purpose Timer Registers—Memory-Mapped (Continued)

Register	Mnemonic	MMCR Offset Address	Function
GP Timer 1 Maxcount Compare A	GPTMR1MAXCMPA	C7Eh	Maxcount value A to compare with current count
GP Timer 1 Maxcount Compare B	GPTMR1MAXCMPB	C80h	Maxcount value B, used in the alternate mode
GP Timer 2 Mode/Control	GPTMR2CTL	C82h	GP Timer 2 enable, permit Enable bit write, interrupt enable, maxcount register in use, maximum count, continuous mode
GP Timer 2 Count	GPTMR2CNT	C84h	Current count value
GP Timer 2 Maxcount Compare A	GPTMR2MAXCMPA	C8Eh	Maxcount value to compare with current count
GP Timer 0 Interrupt Mapping	GPTMR0MAP	D1Ah	GP Timer 0 interrupt mapping to any of 22 available interrupt channels or NMI
GP Timer 1 Interrupt Mapping	GPTMR1MAP	D1Bh	GP Timer 1 interrupt mapping
GP Timer 2 Interrupt Mapping	GPTMR2MAP	D1Ch	GP Timer 2 interrupt mapping

17.5 OPERATION

The ÉlanSC520 microcontroller includes three GP timers, each of which supports several different operating modes.

17.5.1 GP Timer 0 and GP Timer 1

GP Timers 0 and 1 can be used to count or time external events that drive the timer input pins and to generate a variety of waveforms on the timer output pins.

The source clock for GP Timer 0 and GP Timer 1 can be configured to be one-fourth of the Am5$_x$86 CPU clock frequency, or it can be driven from the timer external input (TMRIN0 or TMRIN1) whose maximum clock frequency is one-fourth of the Am5$_x$86 CPU clock speed. When driven from the timer's external input pin, the timer counts the "event" of an input transition.

GP Timer 0 and GP Timer 1 are 16-bit timers. Each of these two timers can be cascaded as a 32-bit timer when GP Timer 2 is configured as a prescaler by setting the PSC_SEL bit in the GP Timer x Mode/Control (GPTMRxCTL) register. (See "Combining GP Timer Count Elements" on page 17-6.) When they are in 32-bit mode, GP timers 0 and 1 cannot be used as 16-bit timers.

The TMRIN0 and TMRIN1 pins can be configured to be one of many functions via the use of the configuration bits in the respective timer registers. These functions include:

- Clock input—Configured with the EXT_CLK bit in the GP Timer x Mode/Control (GPTMRxCTL) register
- Enable input—Configured with both the RTG bit and EXT_CLK bit cleared to 0 in the GP Timer x Mode/Control (GPTMRxCTL) register
- Reset input (hardware retrigger mode)—Configured with the RTG bit set to 1 and the EXT_CLK bit cleared to 0 in the GP Timer x Mode/Control (GPTMRxCTL) register

17.5.2 GP Timer 2

GP Timer 2 is a 16-bit timer that is not connected to any external pins. GP Timer 2 can be used by software to generate interrupts, or it can be polled for real-time coding and time-delay applications. It can also be enabled as a prescaler for GP Timer 0 and GP Timer 1. The source clock for GP Timer 2 is always one-fourth of the Am5$_x$86 CPU clock frequency.

17.5.3 Operating Modes

17.5.3.1 Interrupt on Terminal Count Mode

In this mode, an interrupt request is generated when the timer count value reaches a GP Timer Maxcount Compare register value. This is configured with the INT_ENB bit in the GP Timer x Mode/Control (GPTMRxCTL) register.

If continuous mode is enabled, the interrupt request pulse is generated continuously at a regular interval time, and the interval duration depends on the value in the GP Timer Maxcount Compare register.

17.5.3.2 Hardware Retrigger Mode

In hardware retrigger mode, a 0-to-1 edge transition on the TMRIN1 or TMRIN0 input pin resets the existing GP Timer x Count (GPTMRxCNT) register value, for their respective timers, and then counting continues. This mode is enabled by setting the RTG bit to 1 and clearing the EXT_CLK bit to 0 in the GP Timer x Mode/Control (GPTMRxCTL) register.

17.5.3.3 Alternate Compare Mode

Using both the primary GP Timer x Maxcount Compare A (GPTMRxMAXCMPA) register and the secondary GP Timer x Maxcount Compare B (GPTMRxMAXCMPB) register lets the timer alternate between two maximum values. This mode is enabled with the ALT_CMP bit in the GP Timer x Mode/Control (GPTMRxCTL) register.

In alternate compare mode, the TMROUT0 or TMROUT1 pin is High while the counter is counting and being compared to the GP Timer x Maxcount Compare A (GPTMRxMAXCMPA) register. The timer output pin is Low while the counter is counting and being compared to the GP Timer x Maxcount Compare B (GPTMRxMAXCMPB) register.

17.5.3.4 Square Wave Mode

In this mode, the TMROUT0 or TMROUT1 pin creates a waveform by indicating which of the two GP Timer Maxcount Compare registers is currently in control. The duty cycle and frequency of the waveform depend on the values in the alternating GP Timer Maxcount Compare register. This mode is enabled when both the ALT_CMP and the CONT_COMP bits are set in the GP Timer x Mode/Control (GPTMRxCTL) register.

17.5.3.5 Continuous Mode

In continuous mode, the GP Timer x Count (GPTMRxCNT) register is reset to 0 after it reaches the value in the GP Timer x Maxcount Compare register value (A or B), and the timer immediately begins counting again. Continuous mode is enabled by setting the CONT_CMP bit in the GP Timer x Mode/Control (GPTMRxCTL) register.

17.5.3.6 Prescaler Mode

The internal output of GP Timer 2 can be used as the input clock source for GP timers 0 and 1. When the PSC_SEL bit is set in the GP Timer x Mode/Control (GPTMRxCTL) register, timers 0 and 1 can be prescaled by GP Timer 2. This allows either or both GP Timer 0 and GP Timer 1 to be cascaded as a 32-bit timer. The PSC_SEL bit is ignored when external clocking is enabled (i.e., when the EXT_CLK bit is set).

17.5.4 Configuration Information

The GP Timer x Count (GPTMRxCNT) registers contain the current value of a timer. These registers can be read or written at any time, regardless of whether the corresponding timer is running. The timer increments the value of the corresponding GP Timer x Count (GPTMRxCNT) register each time a timer event occurs.

Each timer has a GP Timer x Maxcount Compare A (GPTMRxMAXCMPA) register that defines the maximum value of the timer.

- When the timer reaches the maximum value, it resets the GP Timer x Count (GPTMRxCNT) register value to 0 during the same clock cycle.
- The value in the GP Timer x Count (GPTMRxCNT) register never equals the GP Timer x Maxcount Compare A (GPTMRxMAXCMPA) register.

In addition, timers 0 and 1 have a secondary GP Timer x Maxcount Compare B (GPTMRxMAXCMPB) register.

- Using both the primary GP Timer x Maxcount Compare A (GPTMRxMAXCMPA) register and the secondary GP Timer x Maxcount Compare B (GPTMRxMAXCMPB) register lets the timer alternate between two maximum values. This is called *alternate compare mode*. It is controlled by the ALT_CMP bit in the GP Timer x Mode/Control (GPTMRxCTL) register.
  - If the timer is programmed to use both of its GP Timer Maxcount Compare registers, and the ALT_CMP and CONT_CMP bits are set in the GP Timer x Mode/Control (GPTMRxCTL) register, the timer output pin (TMROUT0 or TMROUT1) generates a square waveform.
  - The duty cycle and frequency of the waveform depend on the values in the alternating GP Timer Maxcount Compare registers.
- If the timer is programmed with the ALT_CMP bit to use only the primary GP Timer x Maxcount Compare A (GPTMRxMAXCMPA) register, the timer output pin (TMROUT0 or TMROUT1) switches Low for a single Am5$_x$86 CPU clock cycle after the maximum value is reached.

17.5.5 Clocking Considerations

The clock source for the three general-purpose timers is the 33-MHz system clock. For GP Timer 0 and GP Timer 1, the clock source can also be an external pin or a derived prescale clock. This option is specified in the GP Timer 0 Mode/Control (GPTMR0CTL) register (MMCR offset C72h) and the GP Timer 1 Mode/Control (GPTMR1CTL) register (MMCR offset C7Ah).

17.5.5.1 Internal Clock

The resolution range of the internal clock depends on which 33-MHz crystal is used in the system, as shown in Table 17-3.

Table 17-3 GP Timers Internal Clock Sources

Internal Clock Source	Resolution Range	Duration
33.000 MHz	121.20 ns–7.94 ms	16-bit duration
33.000 MHz	121.20 ns–520.55 seconds	32-bit duration
33.333 MHz	120.00 ns–7.86 ms	16-bit duration
33.333 MHz	120.00 ns–515.40 seconds	32-bit duration

17.5.5.2 External Clock

Separate external clock input pins, TMRIN0 and TMRIN1, are provided to each of the following two timers: GP Timer 0 and GP Timer 1, respectively. Table 17-4 specifies the external clock source frequency range for the TMRIN0 and TMRIN1 inputs for the general-purpose timers. The maximum frequency of the external clock is one-fourth the frequency of the crystal used.

Table 17-4 GP Timers External Clock Sources (Using a 33.333 MHz Crystal)

External Clock Source	Frequency Range
TMRIN0	0–8.33325 MHz
TMRIN1	0–8.33325 MHz

17.5.6 Interrupts

Each GP timer provides its own interrupt on the programmable interrupt controller (PIC). See Chapter 15, “Programmable Interrupt Controller”, for more information on interrupt steering.

The GP Timer x Mode/Control (GPTMRxCTL) registers for the general-purpose timers are used to enable the timer interrupt request generation. An interrupt request is generated when a maximum count is reached.

In the case where both Maximum Count Compare A and B registers are used, an interrupt request is generated when the GP Timer x Count (GPTMRxCNT) register is equal to either the value of the GP Timer x Maxcount Compare A (GPTMRxMAXCMPA) register or the value of the GP Timer x Maxcount Compare B (GPTMRxMAXCMPB) register.

The GP Timers Status (GPTMRSTA) register (MMCR offset C70h) contains the interrupt status information for the three general-purpose timers. A timer’s corresponding interrupt status bit is set when that timer’s interrupt request signal is asserted and remains set until cleared.

17.5.7 Software Considerations

17.5.7.1 Combining GP Timer Count Elements

Both GP Timer 0 and GP Timer 1 can be configured to be clocked by GP Timer 2 at the same time. This configuration provides a maximum of two 32-bit counters. GP Timer 2 is a common element between the two resulting 32-bit counters. The possible combinations of the timers include:

- GP Timer 2, GP Timer 1, and GP Timer 0 separate, resulting in three independent 16-bit counters
- GP Timer 2 + GP Timer 0 (as one 32-bit), with GP Timer 1 separate (as one 16-bit)
- GP Timer 2 + GP Timer 1 (as one 32-bit), with GP Timer 0 separate (as one 16-bit)
- GP Timer 2 + GP Timer 0 (as one 32-bit), with GP Timer 2 + GP Timer 1 (as second 32-bit), where GP Timer 2 is a common timebase

17.5.7.2 Reading the Cascaded 32-Bit Timer

When cascading GP Timer 0 or GP Timer 1 with GP Timer 2 to form a single 32-bit timer, caution must be exercised when reading the two counter outputs in order to properly handle rollover conditions. This is slightly complicated by the fact that there is no way to atomically read the contents of both counters. The goal is to develop an algorithm to return the 32-bit value of the cascaded timer at the time that the 16-bit “least significant” timer is read.

To test for rollover, software must read both timers two times in succession, reading the least significant timer value (i.e., GP Timer 2), followed by the most significant timer value. A very important assumption must be made that software is able to perform these four 16-bit reads in less one tick of the “most significant” timer. Software may have to disable interrupts in order to meet this qualification.

For example, suppose that on the first read, the value L1 is read from the least significant timer, and the value M1 is read from the most significant timer. Then, values of L2 and M2 are read from the least significant and most significant timers, respectively. There are three possibilities.

17.5.7.2.1 Case 1

(M1 = M2 = 0) and (L1 > L2)

This condition indicates that the most significant timer rolled over between reading L1 and M1. In this case, the correct value to be interpreted from the most significant timer value should be one less than the value programmed into the GP Timer x Maxcount Compare A (GPTMRxMAXCMPA) register (when the ALT_CMP bit is 0), or the maximum of GP Timer x Maxcount Compare A (GPTMRxMAXCMPA) register and GP Timer x Maxcount Compare B (GPTMRxMAXCMPB) register (when the ALT_CMP bit is 1).

17.5.7.2.2 Case 2

(M2 = M1 <> 0) and (L1 > L2)

This condition indicates that the least significant timer (but not the most significant timer) rolled over between reading M1 and L1. In this case, the correct value to be interpreted for the most significant timer is M1–1, which was the value of the most significant timer at the time that L1 was read.

17.5.7.2.3 Case 3

- In all other instances, if rollover occurred, then it occurred after L1 and M1 were read, and L1 and M1 can be used for the correct values.

17.5.7.2.4 Example 1

For example, suppose GP Timer 0 is programmed in continuous mode, clocked by the output of GP Timer 2, with the ALT_CMP bit cleared to 0 and a value of 2000h programmed for the GP Timer 0 Maxcount Compare A (GPTMR0MAXCMPA) register.

GP Timer 2 is programmed in continuous mode, clocked by the internal 33-MHz clock with the GP Timer 2 Maxcount Compare A (GPTMR2MAXCMPA) register set to 8000h.

The period of GP Timer 2 is:

8000h / 33 MHz * 4 = 4 ms

The cycle time of GP Timer 0 is:

2000h * 4 ms = 32.77 s

In the example, software reads the timers in the following order:

1. GP Timer 2 = 7997h
2. GP Timer 0 = 0h
3. GP Timer 2 = 14h
4. GP Timer 0 = 0h

In this example, the second value read for GP Timer 2 (14h) is less than the first value (7997h), and both values read for GP Timer 0 are 0. So this falls under case 1, and the correct 32-bit value of the cascaded timer is:

32764 ms + (7997h * 121.2 ns) = 32767.8 ms = 32.7678 s

17.5.7.2.5 Example 2

Suppose GP Timer 0 and GP Timer 2 are programmed as in Example 1, but the values returned from the timers are:

1. GP Timer 2 = 7997h
2. GP Timer 0 = 15h
3. GP Timer 2 = 5h
4. GP Timer 0 = 16h

In this example, the second value read for GP Timer 2 (5h) is less than the first value (7997h). However, because the first value read for GP Timer 0 (15h) is less than the second value (16h), case 3 applies and the correct 32-bit value of the cascaded timer is:

15h * 4ms + 7997 * 121.2 ns = 72.772 ms

17.6 INITIALIZATION

At system reset, all the general-purpose timer registers are reset to zero. Each timer must be programmed before it can be used.

1. Write the maximum compare count value into the GP Timer x Maxcount Compare (GPTMR0MAXCMPx) registers.
2. Enable the counting with the desired operation and mode in the GP Timer x Mode/Control (GPTMRxCTL) register.

CHAPTER

18 SOFTWARE TIMER

18.1 OVERVIEW

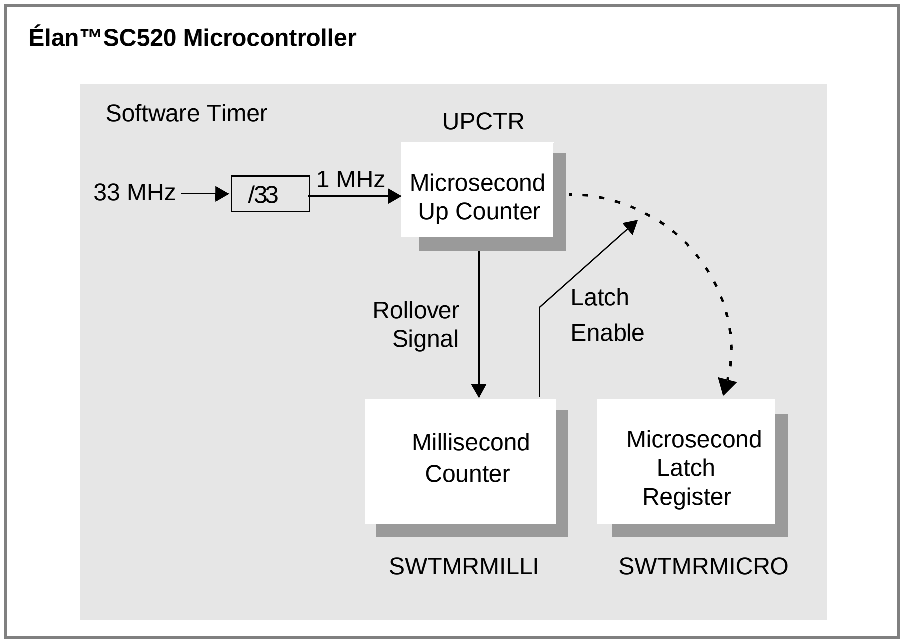
The software timer is intended to provide a millisecond timebase with microsecond resolution. Ideal applications for this function include providing a system wide software timebase, code profiling, and precise measurement of the time between events. Features of the software timer include:

- One 16-bit millisecond counter that increments with a period of one millisecond. This yields a maximum duration of 65.5 seconds. Note that this timer is accurate to the precision of the 33-MHz crystal used in the system.
- A microsecond latch register that provides the number of microseconds since the last time that the millisecond register was read.
- The 16-bit millisecond counter is reset to zero when it is read.
- The software timer can be configured to maintain an accurate time when either a 33.000-MHz or 33.333-MHz crystal is used in the system.

18.2 BLOCK DIAGRAM

Figure 18-1 shows a block diagram of the software timer.

Figure 18-1 Software Timer Block Diagram



18.3 REGISTERS

The software timer includes the registers listed in Table 18-1.

Table 18-1 Software Timer Configuration Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
Software Timer Millisecond Count	SWTMRMILLI	C60h	Current 10-bit count value (milliseconds)
Software Timer Microsecond Count	SWTMRMICRO	C62h	Current latched 16-bit count value (microseconds)
Software Timer Configuration	SWTMRCFG	C64h	Crystal frequency (33.000 MHz or 33.333 MHz)

18.4 OPERATION

The software timer provides a very efficient hardware timebase for use by software. It is designed to replace the traditional method of system timebase generation.

Traditionally, system timebase generation is accomplished by programming a timer to generate a periodic interrupt. The interrupt service routine for this interrupt then increments a counter each time the interrupt occurs. This value is often kept in a global variable, which can then be accessed by other code that needs to track time. Sometimes, a procedural interface (a function) is used to access the value of this counter. The counter maintained by the interrupt service routine is usually set to zero at system initialization time. Thus, it maintains the time since system boot.

The problem with this method is that it consumes a hardware timer resource. It also requires an interrupt service routine that executes very frequently. Often the requirement to have a higher resolution time is difficult to attain because the overhead of executing even a small interrupt service routine many times a second is too much. It is rarely practical to provide better than a 1-ms timebase with this technique. Also, in a system that makes extensive use of interrupts, the timer interrupt can sometimes be missed, causing the interrupt counter to becomes less accurate over time.

The software timer included on the ÉlanSC520 microcontroller can be used to resolve these problems. The software timer provides a 16-bit millisecond counter (the Software Timer Millisecond Count (SWTMRMILLI) register), a 10-bit microsecond-up counter (UPCTR), and a latch register for the UPCTR (the Software Timer Microsecond Count (SWTMRMICRO) register). Both counters reset to zero on system reset.

The microsecond-up counter increments at a rate of 1 MHz and rolls over on every 1000 counts (every 1 millisecond). When the microsecond-up counter rolls over, it signals the millisecond counter to increment. When the millisecond counter is read, three things happen:

1. The value in the Software Timer Millisecond Count (SWTMRMILLI) register (MMCR offset C60h) is returned to software.
2. The value in the microsecond up counter is latched into the Software Timer Microsecond Count (SWTMRMICRO) register (MMCR offset C62h).
3. The Software Timer Millisecond Count (SWTMRMILLI) register counter is reset to zero.

This operation allows software to keep track of time with no interrupt service routine.

For example, here is some example code that can be used to maintain a system timebase:

```
typedef unsigned long int DWORD;          // an unsigned 32-bit value
typedef unsigned short int WORD:         // an unsigned 16-bit value

static volatile WORD* SWTMRMILLI = 0xA0000200;   // volatile is essential
static volatile WORD* SWTMRMICRO = 0xA0000202;   // volatile is essential

static DWORD ticks;          // the number of 1-ms ticks since system boot
                             // that have passed since system reset

static DWORD mics;           // A running microsecond value

DWORD sys_ticks()

         {
         ticks += *SWTMRMILLI;
         mics = *SWTMRMICRO + (ticks * 1000);
         return ticks;
         }

DWORD sys_mics()

         {
         ticks += *SWTMRMILLI;
         mics  = *SWTMRMICRO + (ticks * 1000);
         return mics;
         }
```

This is all the code necessary to maintain both a 32-bit microsecond and a 32-bit millisecond timebase for an operating system or other timing needs.

18.4.1 Configuration Information

The software timer counter elements (millisecond and microsecond counts) are read-only. The software timer is always free-running, and it does not have any input or output (external pin or interrupt). The software timer can be configured to maintain an accurate time for either a 33.000-MHz or a 33.333-MHz crystal.

18.5 INITIALIZATION

At system reset, the software timer begins counting up from zero.

The software timer must be initialized for operation with either 33.000 MHz or 33.333 MHz, depending on the crystal being used in the system. This is configured with the XTAL_FREQ bit in Software Timer Configuration (SWTMRCFG) register (MMCR offset C64h).

CHAPTER

19 WATCHDOG TIMER

19.1 OVERVIEW

The ÉlanSC520 microcontroller includes an integrated watchdog timer (WDT).

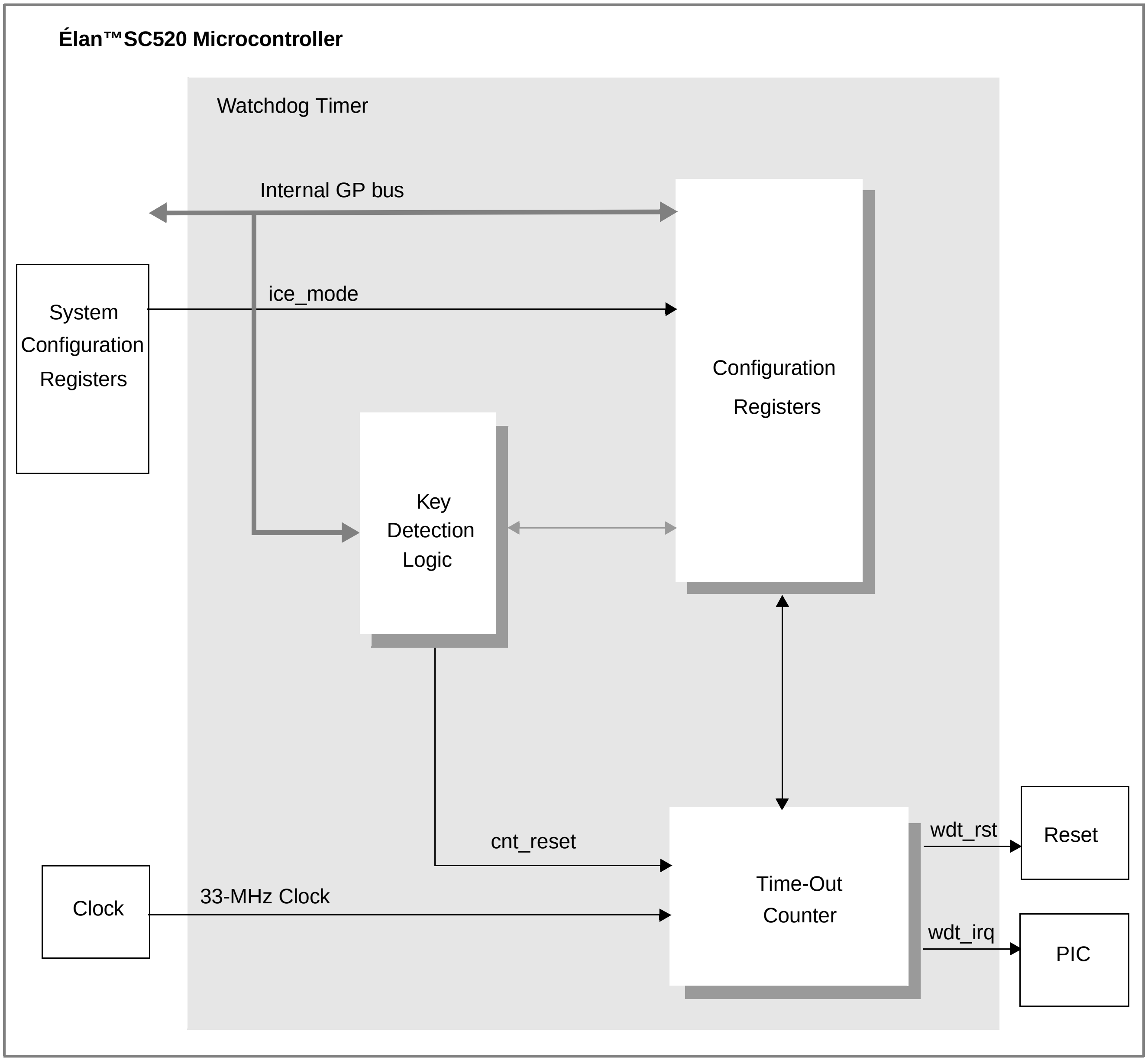
Features of the watchdog timer include:

- Distinct keyed write sequences are required to open the Watchdog Timer Control (WDTMRCTL) register for reconfiguration and to reset the current count.
- Supports up to a 30-second time-out period with a 33-MHz CPU clock
- Programmable to generate either a system reset or an interrupt request (maskable or non-maskable) on the first time-out. If software has not cleared an indicator bit by the second time-out, the watchdog timer always generates a system reset instead.
- The watchdog timer interrupt request can be programmed as maskable or non-maskable.
- A status flag for software to detect the watchdog timer's interrupt request
- ÉlanSC520 microcontroller input pins that are typically sampled at the initial power-on reset (i.e., with the PWRGOOD input) are not sampled for a system reset due to a watchdog timer time-out.
- The watchdog timer counters are automatically stopped in AMDebug technology mode.

19.2 BLOCK DIAGRAM

Figure 19-1 shows a block diagram of the watchdog timer.

Figure 19-1 Watchdog Timer Block Diagram



19.3 REGISTERS

The watchdog timer is controlled by the memory-mapped registers listed in Table 19-1.

Table 19-1 Watchdog Timer Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
Watchdog Timer Control	WDTMRCTL	CB0h	Watchdog timer enable, WDT reset enable, interrupt flag, duration of the WDT time-out interval
Watchdog Timer Count Low	WDTMRCNTL	CB2h	Bits 15–0 of the WDT current count

Table 19-1 Watchdog Timer Registers—Memory-Mapped (Continued)

Register	Mnemonic	MMCR Offset Address	Function
Watchdog Timer Count High	WDTMRCNTH	CB4h	Bits 30–16 of the WDT current count
Watchdog Timer Interrupt Mapping	WDTMAP	D42h	WDT interrupt mapping
Reset Status	RESSTA	D74h	Reset source status: watchdog timer time-out

19.4 OPERATION

The watchdog timer (WDT) can be used to regain control of the system when software fails to respond as expected. The watchdog timer should be used in systems that require a guaranteed recovery time from a software error.

When the watchdog timer is enabled, the counter is reset to zero automatically and starts counting. The count increments once for every 33-MHz clock cycle**.** While enabled, the software can reset the counter to zero at anytime by writing a clear keyed sequence as described in “Keyed Sequences” on page 19-3. If the software is unable to reset the counter before it reaches the time-out count, the watchdog timer generates an interrupt and/or a system reset.

- The watchdog timer can be configured to cause *either* an interrupt (maskable or non-maskable) or a system reset upon time-out.
- The watchdog timer can also be configured to generate *both* an interrupt and a system reset. In this mode, the watchdog timer generates an interrupt, then starts itself over. If it times out a second time, it generates a system reset.

A distinct keyed sequence is required to open up the Watchdog Timer Control (WDTMRCTL) register (MMCR offset CB0h) before writes. This prevents errant code from disabling or otherwise modifying the watchdog timer behavior. The same keyed sequence is always used for unlocking the watchdog timer control registers.

19.4.1 Configuration Information

19.4.1.1 Keyed Sequences

All writes to the Watchdog Timer Control (WDTMRCTL) register must be preceded by a distinct keyed sequence.

- A data pattern of 3333h, followed by a write of CCCCh, to the Watchdog Timer Control (WDTMRCTL) register opens up the register for a single write.

The value of the key is not written to the register but is used by internal logic to open the register for writing. Once the ENB bit is set in the Watchdog Timer Control (WDTMRCTL) register, a subsequent keyed sequence is required to allow any further writes to this register.

While enabled, the software can reset the counter to 0 at anytime by writing a keyed sequence to clear the counter.

- A data pattern of AAAAh, followed by a write of 5555h, to the Watchdog Timer Control (WDTMRCTL) register resets the counter.

The key itself resets the counter; no further writes are necessary. It should be noted that this clear-count key cannot be initiated while the write key is active. This would result in the value of AAAAh being written to the register.

Each individual write of these keyed sequences is not required to be written back-to-back as an atomic sequence. Any number of processor cycles, including memory and I/O reads and writes, can be inserted between the key and the writing of data, as long as they do not access the Watchdog Timer Control (WDTMRCTL) register.

19.4.1.2 Interrupt Request Generation

To configure interrupt request generation on the watchdog timer, software must first clear the ENB bit in the Watchdog Timer Control (WDTMRCTL) register and then clear the WRST_ENB bit. Once the watchdog timer times out, the interrupt request is generated.

The watchdog timer interrupt request can be programmed as maskable or non-maskable. See Chapter 15, “Programmable Interrupt Controller”, for details on selecting a maskable or non-maskable watchdog timer interrupt request.

If a second time-out event occurs and software has not cleared the IRQ_FLG bit asserted by the first time-out, the watchdog timer causes a system reset instead of an interrupt request, regardless of the setting of the WRST_ENB bit.

19.4.1.3 System Reset Generation

To configure the watchdog timer for system reset generation, software must first clear the ENB bit and then set the WRST_ENB bit. Once the watchdog timer times out, the system reset is generated.

19.4.1.4 Time-Out Duration

The Exponent Select (EXP_SEL) bit field in the Watchdog Timer Control (WDTMRCTL) register indicates the exponent value used to calculate the time-out duration in the following formula:

$$\text{Duration} = 2^{\text{Exponent}} / (\text{33 MHz crystal frequency})$$

where:

frequency is based on a 33-MHz incoming clock, as shown in Table 19-2.

Note that the ENB bit must be cleared to 0 before the EXP_SEL field can be written.

Table 19-2 Watchdog Timer Time-Out Duration

EXP_SEL Field	Exponent	33.000 MHz	33.333 MHz
00h	invalid value	infinity	infinity
01h	14	496 µs	492 µs
02h	24	508 ms	503 ms
04h	25	1.02 s	1.01 s
08h	26	2.03 s	2.01 s
10h	27	4.07 s	4.03 s
20h	28	8.13 s	8.05 s
40h	29	16.27 s	16.11 s
80h	30	32.54 s	32.21 s

Notes:

Only the least significant bit set in the EXP_SEL field determines the time-out duration. For example, setting the field to F0h results in an exponent of 27.

19.4.2 Interrupts

An interrupt is asserted upon time-out if the watchdog timer interrupt condition is configured accordingly in the Watchdog Timer Control (WDTMRCTL) register.

- If the watchdog timer is configured for interrupts, the IRQ_FLG bit in the Watchdog Timer Control (WDTMRCTL) register is set when the interrupt is generated.
- The interrupt service routine should examine this flag to determine if the interrupt was generated by the watchdog timer.
- If the IRQ_FLG is set, the interrupt service routine should clear the flag by writing the correct keyed sequence to the Watchdog Timer Control (WDTMRCTL) register and follow by writing a 1 to this bit.
- If the IRQ_FLG bit is not cleared when a second watchdog timer time-out occurs, a WDT system reset is generated, rather than a second interrupt event.

Note: *The IRQ_FLG bit is not cleared on a read. The bit must be cleared by writing the correct keyed sequence to the watchdog timer before writing a 1 to the corresponding bit position.*

19.4.3 AMDebug™ Technology Interface

The AMDebug technology interface allows emulator code to run without having to deal with possible watchdog timer time-outs. It also allows emulators to be used more effectively with applications that enable the watchdog timer. Entering AMDebug technology mode stops the watchdog timer counter from counting.

19.4.4 Software Considerations

If a watchdog timer time-out occurs when the timer is programmed with the WRST_ENB bit cleared, an interrupt is generated, the time-out counter is reset, and the IRQ_FLG bit in the Watchdog Timer Control (WDTMRCTL) register is set. If the IRQ_FLG bit is not cleared by software before a second watchdog timer time-out, a system reset is generated, regardless of the setting of the WRST_ENB bit. The IRQ_FLG bit can be cleared, but not set, by software.

Generation of the internal interrupt signal on the first watchdog timer time-out can be useful in systems where it may be possible to recover from spurious pulses, bad data, or incorrect code. This is especially true for the case where potential data recovery is important. Such systems should have the interrupt handler routine to ensure that it has not been corrupted by errant code. The watchdog timer must function in all cases where the software has failed to respond appropriately. The ÉlanSC520 microcontroller's watchdog timer has incorporated several features to ensure that this is the case.

- Once software enables the watchdog timer, the registers become read-only, except for the ENB and IRQ_FLG bits. This allows boot or monitor code to disable the watchdog timer until the system has been configured.
- All writes to the watchdog timer must be preceded by writes of a keyed sequence. Detection of the keyed sequence allows a single write to the Watchdog Timer Control (WDTMRCTL) register.
- The watchdog timer time-out counter can only be reset by setting the ENB bit or by writing a special clear key sequence to the Watchdog Timer Control (WDTMRCTL) register address.

These features guarantee that the watchdog timer is not affected by errant code.

Although both the Watchdog Timer Count High and Low registers can be read from a single 32-bit CPU instruction, 32-bit accesses are split into two 16-bit accesses. If it is necessary to read an accurate 32-bit value from the Watchdog Timer Counter, see Chapter 17, “General-Purpose Timers”, for suggestions on dealing with this issue.

19.5 INITIALIZATION

At power-on reset, the watchdog timer is disabled. Software must enable it by setting the ENB in the Watchdog Timer Control (WDTMRCTL) register. The watchdog timer time-out count defaults to the maximum value. The WRST_ENB bit is set for generation of system reset upon time-out. See “Configuration Information” on page 19-3.

Note that the processor does not resample external pins during a watchdog timer-generated system reset. This means that the System Board Information (SYSINFO) register (MMCR offset D70h), $\overline{\text{BOOTCS}}$ data bus width, and $\overline{\text{BOOTCS}}$ data bus select parameters do not change when a watchdog timer system reset occurs. All other activities are identical to those of a normal system reset.

CHAPTER

20 REAL-TIME CLOCK

20.1 OVERVIEW

The real-time clock (RTC) included on the ÉlanSC520 microcontroller is compatible with the MC146818A device used in PC/AT systems. The RTC consists of a time-of-day clock with alarm and a 100-year calendar. The clock/calendar has a programmable periodic interrupt and 114 bytes of static user RAM. The clock/calendar can be represented in either binary or binary-coded decimal (BCD).

The RTC includes the following features:

- PC/AT-compatible
- Counts seconds, minutes, and hours of the day
- Counts day of the week, date, month, and year
- Binary or BCD representation of time, calendar, and alarm
- 12- or 24-hour clock, with AM and PM indicator in 12-hour mode
- Daylight saving time option
- Automatic end-of-month recognition
- Automatic leap year compensation
- 14 bytes of clock and control registers
- 114 bytes of general-purpose RAM
- Three interrupt sources separately maskable with corresponding status bits
- Time-of-day alarm is programmable to occur from once-per-second to once-per-day
- Periodic interrupts can be programmed to occur at rates from 122 µs to 500 ms
- Update-ended interrupt provides cycle status
- Internal RTC reset signal can perform a reset on power-up

The RTC has its own power pin and reset separate from the rest of the other core supplies. When the chip is powered off, the RTC can remain powered up and in full functional mode, maintaining time, calendar, and user RAM data.

The RTC includes registers for time, calendar, and alarm data and four control/status registers. The RTC Status D (RTCSTAD) register (RTC index 0Dh) has a status bit (RTC_VRT) that indicates the validity of the contents of the RAM, time registers and the calendar. The RTC_VRT bit is set based on the assertion of the internal RTC reset.

The RTC interrupt request is connected internally to the programmable interrupt controller block.

20.2 BLOCK DIAGRAM

Figure 20-1 shows a block diagram of the real-time clock.

Figure 20-2 on page 20-3 shows a block diagram of the RTC voltage monitor. The ÉlanSC520 microcontroller's RTC voltage monitor is designed to signal the RTC core when the backup battery is not installed or is low. Additionally, the voltage monitor circuit signals the RTC core when the rest of the system is being powered down.

As shown in Figure 20-2, the voltage monitor includes a bandgap voltage generator for precision reference voltage and a high-gain amplifier for adjusting bandgap voltage to "low-battery" trip voltage. In addition to the backup battery monitor function, the voltage monitor also provides a power-down signal to the RTC. This signal is used to isolate the RTC core from the rest of the integrated peripherals. A timing diagram for this sequence is shown in the *Élan™SC520 Microcontroller Data Sheet*, order #22003.

Figure 20-1 Real-Time Clock Block Diagram

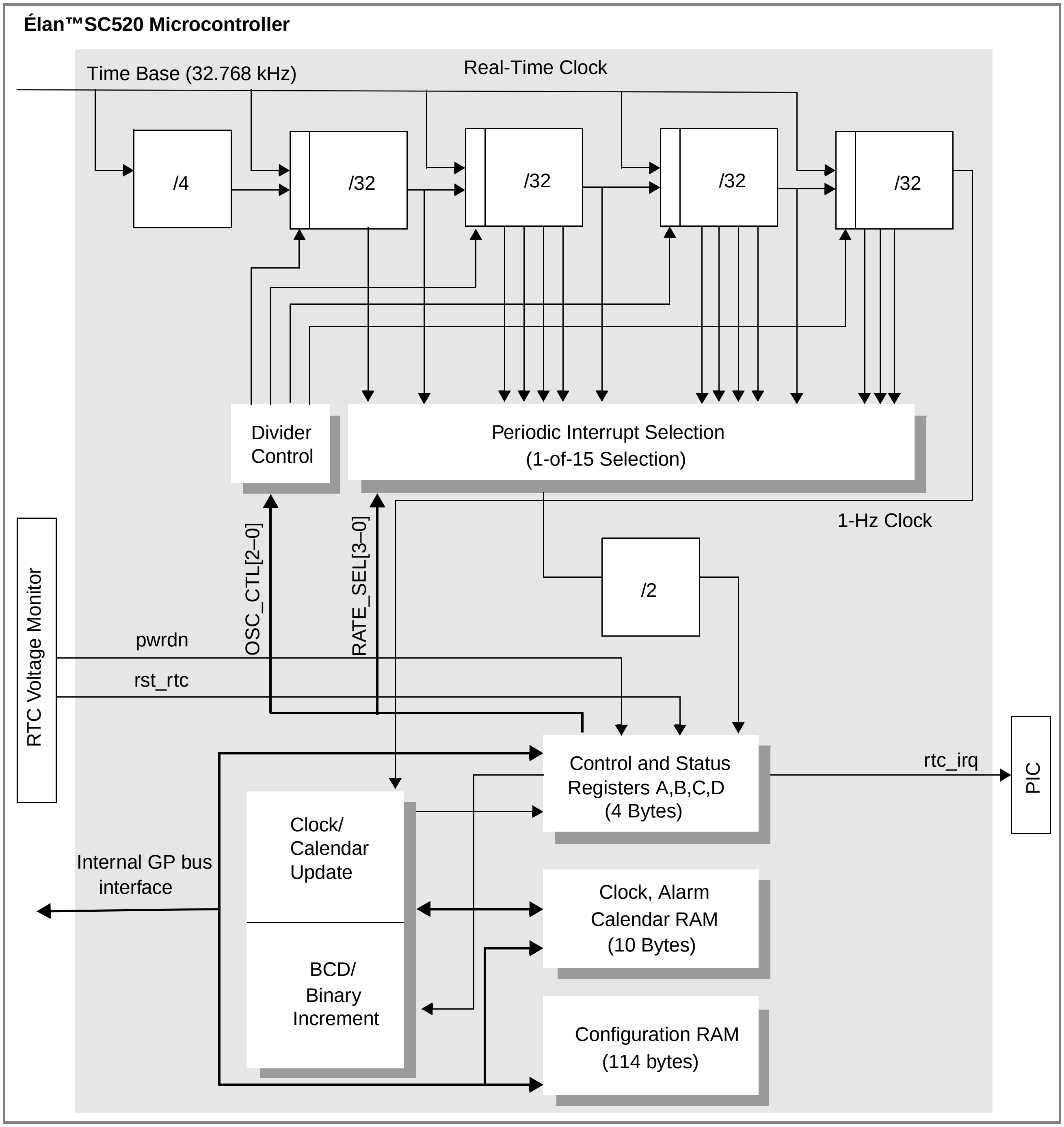
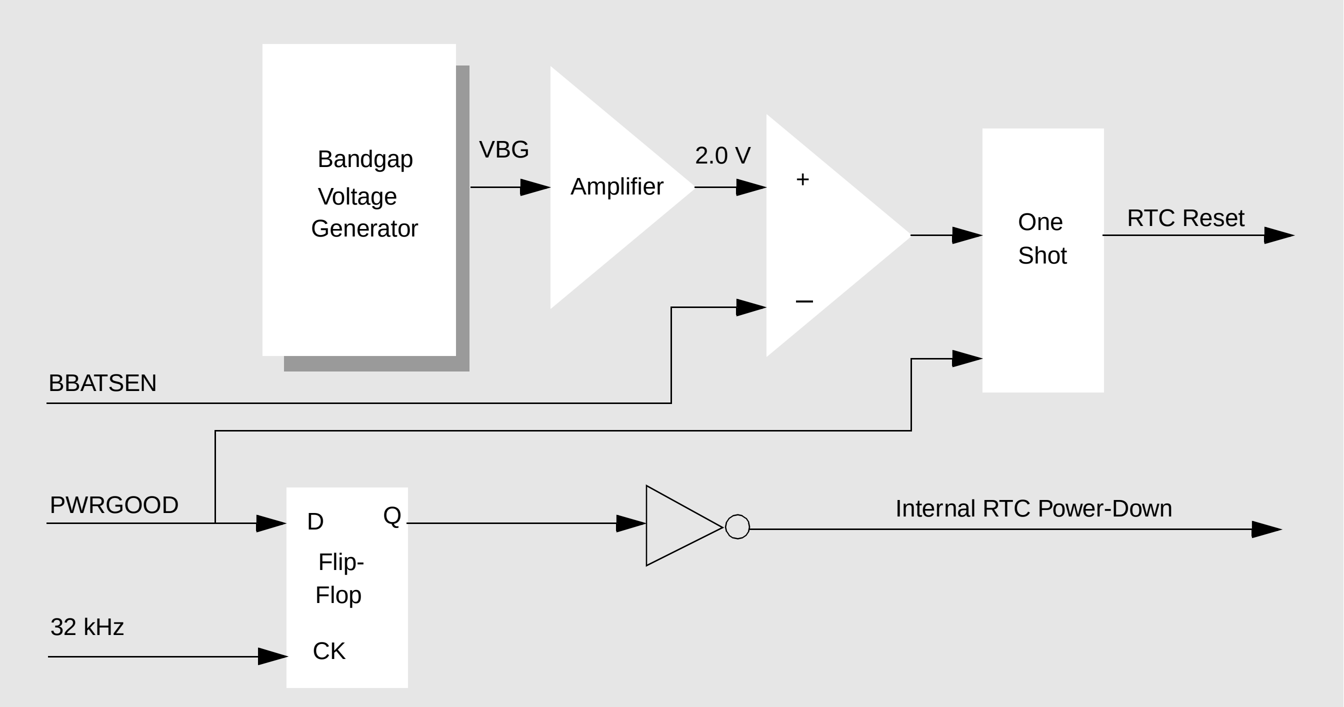

Élan™SC520 Microcontroller
Time Base (32.768 kHz)
Real-Time Clock
/4
/32
/32
/32
/32
Divider Control
Periodic Interrupt Selection (1-of-15 Selection)
1-Hz Clock
/2
OSC_CTL[2–0]
RATE_SEL[3–0]
RTC Voltage Monitor
pwrdn
rst_rtc
Control and Status Registers A,B,C,D (4 Bytes)
rtc_irq
PIC
Clock/ Calendar Update
BCD/ Binary Increment
Internal GP bus interface
Clock, Alarm Calendar RAM (10 Bytes)
Configuration RAM (114 bytes)

Figure 20-2 RTC Voltage Monitor Block Diagram



20.3 SYSTEM DESIGN

20.3.1 Backup Battery Considerations

The behavior of the RTC when the primary power supply is turned off depends on whether or not an external backup battery is included in the system design. The RTC can be connected to the main power plane if a backup battery is not needed in the system.

20.3.1.1 System with an External Backup Battery

If an external RTC backup battery is connected to the ÉlanSC520 microcontroller's VCC_RTC pin, the real-time clock (RTC) remains operational even if all the other power supplies are turned off.

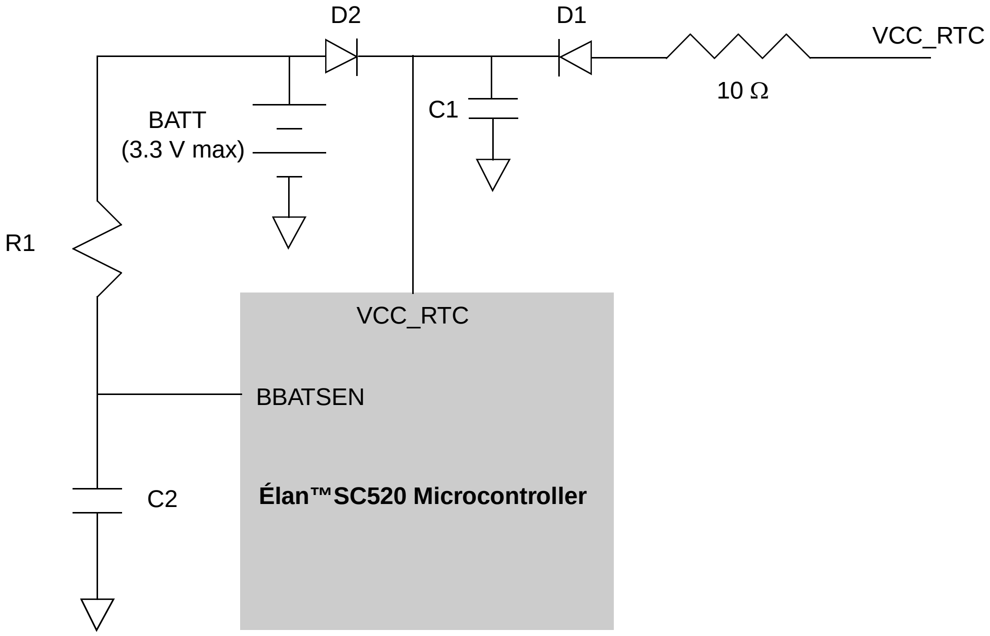
An implementation using a backup battery is shown in Figure 20-3. The primary power source for VCC_RTC is the main power plane (V_{CC}). D1 should be chosen so that the forward voltage drop is small, less than 0.25 V. D1 also prevents the backup battery from powering up the V_{CC} power plane when the main supply is turned off.

The backup battery voltage must not exceed 3.3 V (affects the BBATSEN and VCC_RTC pins); higher voltages may damage the ÉlanSC520 microcontroller.

The RC network composed of R1 and C2 provides a time delay for the internal circuit power-up sequence. C1 is for high-frequency filtering purposes.

See the *Élan™SC520 Microcontroller Data Sheet*, order #22003, for detailed component specifications.

Figure 20-3 Circuit with Backup Battery



Software can read the RTC_VRT bit in the RTC Status D (RTCSTAD) register (RTC index 0Dh) at system boot time to determine whether or not the RTC time, date, and user RAM are still valid since the last boot. This status bit is set based on the assertion of an internal RTC-only reset signal. In systems with external backup batteries (as shown in Figure 20-3), the RTC is reset when the main power supply is turned off and the backup battery is either low or is not installed.

In these systems, the VCC_RTC pin is a dedicated power supply pin for the 32.768-kHz oscillator and the RTC.

- When the primary system power supply is turned on, the main power plane (V_{CC}) drives the VCC_RTC pin through an external diode.
- When the primary power supply is turned off or nonfunctional, VCC_RTC is driven by the backup battery through a second external diode.
- The on-chip voltage monitor circuit monitors the voltage level of the backup battery through the BBATSEN pin each time the PWRGOOD signal is asserted.
- If the backup battery is sampled below 2.0 V, the RTC logic is reset. The read-only RTC_VRT bit is cleared and latched in this state until the bit is read. After this bit is initially read, it always reads back a value of 1 for all subsequent reads prior to an RTC reset.
- When the main system power supply is off and the backup battery is initially installed, the external RC circuit consisting of R1 and C2 causes a slow rising edge on the BBATSEN input, and the RTC is reset.

20.3.1.2 System without an External Backup Battery

For the system that is not using an external RTC backup battery, Figure 20-4 shows how the circuit should be designed. It uses the same RC that is needed by the battery system, but it is now connected to VCC_RTC.

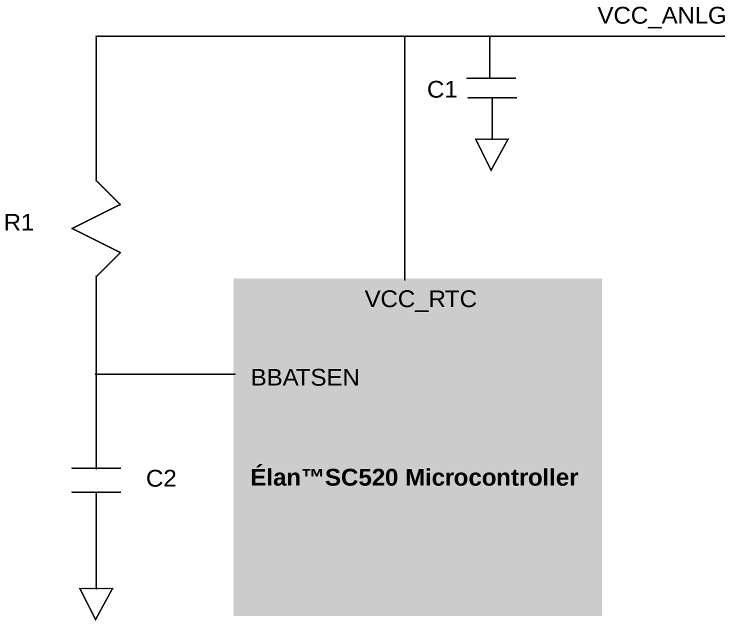
VCC_ANLG is selected as the power plane for VCC_RTC because it is a well-filtered power plane that is well below the VCC_RTC maximum of 3.3 V.

In this configuration, the RTC is reset after power-up, but is not reset by subsequent PWRGOOD assertions.

- The RTC is reset after a power-up—When power has been removed from the RTC, the contents are no longer valid. In this case, the RTC is reset.
- RTC is not reset—When a reset switch tied to PWRGOOD is pressed (V_{CC} remains High) and PWRGOOD reasserts with BBATSEN High, the RTC is not reset. In this case, power did not go away, the RTC contents are still valid, and no RTC reset occurs.

Detailed component specifications for the resistor and capacitor shown in Figure 20-4 can be found in the *Élan™SC520 Microcontroller Data Sheet*, order #22003.

Figure 20-4 Circuit without Backup Battery



20.3.2 Selecting and Interfacing a 32.768-kHz Crystal

See the *Élan™SC520 Microcontroller Data Sheet*, order #22003, for information and detailed specifications for selecting a 32.768-kHz crystal.

20.3.3 Using an External RTC

When the ÉlanSC520 microcontroller comes out of reset, the internal RTC is enabled. If the system application requires the use of an external RTC, the internal RTC should be disabled during the boot process and initialization to prevent potential conflicts.

The Address Decode Control (ADDDECCTL) register (MMCR offset 80h) includes a control bit for selecting between the internal RTC and an external RTC. Setting the RTC_DIS bit to 1 disables the internal RTC by disabling the internal I/O decode for addresses 0070h and 0071h. When the RTC_DIS bit is set, accesses to these addresses generate external bus cycles, allowing the use of an external RTC module.

Disabling the internal RTC does not automatically disable the interrupt connection to the programmable interrupt controller (PIC). If PC/AT-compatibility is required, the designer should connect the external RTC's interrupt request to one of the ÉlanSC520 microcontroller's programmable interrupt inputs and program the interrupt steering logic to route the request to interrupt priority P8.

Disabling the internal RTC does not disable or reset the core in any way.

See "Disabling Internal Peripherals" on page 3-21 for more information.

20.4 REGISTERS

The RTC is controlled by the configuration registers listed in Table 20-1, Table 20-2, and Table 20-3.

Table 20-1 Real-Time Clock Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
Address Decode Control	ADDDECCTL	80h	RTC disable
RTC Interrupt Mapping	RTCMAP	D43h	RTC interrupt mapping

Table 20-2 Real-Time Clock Registers—Direct-Mapped

Register	Mnemonic	I/O Address	Function
RTC/CMOS RAM Index	RTCIDX	0070h	RTC index to read or write
RTC/CMOS RAM Data Port	RTCDATA	0071h	Data to be read or written

Table 20-3 Real-Time Clock Registers—RTC Indexed

Register	Mnemonic	I/O Address	Function
RTC Current Second	RTCCURSEC	70h/71h Index 00h	Seconds
RTC Alarm Second	RTCALMSEC	70h/71h Index 01h	Seconds alarm
RTC Current Minute	RTCCURMIN	70h/71h Index 02h	Minutes
RTC Alarm Minute	RTCALMMIN	70h/71h Index 03h	Minutes alarm
RTC Current Hour	RTCCURHR	70h/71h Index 04h	Hours, 12- and 24-hour mode
RTC Alarm Hour	RTCALMHR	70h/71h Index 05h	Hours alarm, 12- and 24-hour mode
RTC Current Day of the Week	RTCCURDOW	70h/71h Index 06h	Day of the week
RTC Current Day of the Month	RTCCURDOM	70h/71h Index 07h	Day of the month
RTC Current Month	RTCCURMON	70h/71h Index 08h	Month
RTC Current Year	RTCCURYR	70h/71h Index 09h	Year

Table 20-3 Real-Time Clock Registers—RTC Indexed (Continued)

Register	Mnemonic	I/O Address	Function
RTC Control A	RTCCTLA	70h/71h Index 0Ah	Update status, divider chain control, and periodic interrupt rate selection
RTC Control B	RTCCTLB	70h/71h Index 0Bh	Update override (SET); periodic interrupt, alarm interrupt, and update-ended interrupt enables; date mode, 24/12 hour control, and daylight saving enable
RTC Status C	RTCSTAC	70h/71h Index 0Ch	Interrupt request, periodic interrupt, alarm interrupt, and update-ended interrupt flags
RTC Status D	RTCSTAD	70h/71h Index 0Dh	RTC power status (BBATSEN)
General-Purpose CMOS RAM (114 bytes)	RTCCMOS	70h/71h Index 0E–7Fh	General-purpose CMOS RAM bytes

20.5 OPERATION

Programs can retrieve time and calendar information from the RTC by reading the appropriate RTC index registers. Programs can also change the time, calendar, and alarm information in the RTC by writing to these registers.

The RTC executes an update cycle once per second, assuming that the OSC_CTL bit field in the RTC Control A (RTCCTLA) register (RTC index 0Ah) has been set to 010b and that the SET bit in RTC Control B (RTCCTLB) register has been cleared. When the SET bit is 1, all updates are disabled, and the program can initialize the time and date registers.

With a 32.768-kHz time base, the update cycle takes 1984 µs. If a program reads these RAM locations before the update is complete, the output is undefined. The Update-In-Progress (UIP) status bit is set in the RTC Control A (RTCCTLA) register during this time.

There are three ways to handle nonavailability during an RTC update.

- Use the update-ended interrupt—If enabled, this interrupt occurs after every update cycle. This means that over 998 ms are available to read the time and date registers.
- Use the Update-in-Progress bit (UIP) in RTC Control A (RTCCTLA) register—The UIP bit changes once per second. The update cycle begins 244 µs after the UIP bit goes high. This means that, if a 0 is read on the UIP bit, there are at least 244 µs before the time or calendar data will be changed. If a 1 is read in the UIP bit, the time or calendar data may not be valid. Note that the time allocated to read time or calendar data should not exceed 244 µs.
- Use a periodic interrupt to determine if an update cycle is occurring.

Note that, to ensure correct data, the time should not be set on the last day of the month within two seconds of the rollover to the next day.

20.5.1 Configuration Information

20.5.1.1 Configuring the Hour Format

The 12/24 Hour Mode Select (HOUR_MODE_SEL) bit in the RTC Control B (RTCCTLB) register (RTC index 0Bh) establishes whether the hour locations represent 1-to-12 or 0-to-23. The HOUR_MODE_SEL bit can not be changed without re-initializing the RTC Current Hour (RTCCURHR) register (RTC index 04h) and the RTC Alarm Hour (RTCALMHR) register (RTC index 05h). When the 12-hour format is selected, the AM_PM bit and ALM_AM_PM bit in these two respective registers represent PM when they are a 1.

20.5.1.2 Programming the Date and Time

A program can initialize the time, calendar, and alarm by writing to appropriate RAM location.

- Before initializing the internal registers, set the SET bit in the RTC Control B (RTCCTLB) register (RTC index 0Bh) to prevent time and calendar updates from occurring.
- Initialize the ten locations in the selected format (binary or BCD).
- Specify the format using the data mode (DATE_MODE) bit in the RTC Control B (RTCCTLB) register. All time, calendar, and alarm registers must use the same data mode, either binary or BCD.
- Clear the SET bit to allow updates.

20.5.1.3 Generating Periodic Interrupts

Different periodic interrupt rates can be specified by programming the RATE_SEL bit field in the RTC Control A (RTCCTLA) register, as shown in Table 20-4.

The periodic interrupt is enabled by the PER_INT_ENB bit field in the RTC Control B (RTCCTLB) register. The PER_INT_FLG bit in the RTC Status C (RTCSTAC) register (RTC index 0Ch) provides latched status for the RTC periodic interrupt event.

Note: *The first interrupt may not occur at the programmed rate due to synchronization.*

Table 20-4 Using RATE_SEL to Specify a Periodic Interrupt Rate

Periodic Interrupt Interval	Frequency	RATE_SEL3–0			
None	None	0	0	0	0
3.90625 ms	256 Hz	0	0	0	1
7.8125 ms	128 Hz	0	0	1	0
122.070 µs	8 kHz	0	0	1	1
244.14 µs	4 kHz	0	1	0	0
488.28 µs	2 kHz	0	1	0	1
976.562 µs	1 kHz	0	1	1	0
1.953125 ms	512 Hz	0	1	1	1
3.90625 ms	256 Hz	1	0	0	0
7.8125 ms	128 Hz	1	0	0	1
15.625 ms	64 Hz	1	0	1	0
31.25 ms	32 Hz	1	0	1	1
62.5 ms	16 Hz	1	1	0	0
125 ms	8 Hz	1	1	0	1
250 ms	4 Hz	1	1	1	0
500 ms	2 Hz	1	1	1	1

20.5.1.4 Using the Alarm Function

The three alarm bytes can be used in two different ways.

- If the Alarm Interrupt Enable (ALM_INT_ENB) bit is set in the RTC Control B (RTCCTLB) register, the alarm interrupt occurs at the time specified in the alarm registers.
- If a “don’t care” state (any hexadecimal byte from C0–FFh) is written to one or more of the three alarm registers, the alarm interrupt occurs from once per second to once per hour.
  - Setting the hour, minute, and second alarm registers with a value from C0–FFh causes an RTC alarm event to be generated once per second.
  - Setting the hours, and minutes alarm registers with a value from C0–FFh causes an RTC alarm event to be generated once per minute.
  - Setting the hours alarm registers with a value from C0–FFh causes an RTC alarm event to be generated once per hour.

20.5.1.5 Handling Year 2000 Issues

With appropriate software support, the ÉlanSC520 microcontroller is Y2K-compliant. The Y2K problem is handled by storing the century part of the year in the byte at 32h in the CMOS memory. The operating system software must handle rollover of the RTC Current Year (RTCCURYR) register (RTC index 09h).

To be Y2K-compliant, the software that sets the year must accept four-digit years. The routine that sets the RTC stores the lower portion of the year value in the RTC Current Year (RTCCURYR) register and the upper portion in the century CMOS memory location.

This operation is handled properly by PC-style BIOS software that supports the ÉlanSC520 microcontroller. For information on what BIOS products are supported, see the AMD web site.

For embedded systems, a simple set of software functions supports four-digit years with the RTC.

20.5.2 Clocking Considerations

The 32KXTAL2 and 32KXTAL1 pins are used to connect the external 32.768-kHz crystal or oscillator to the ÉlanSC520 microcontroller. This clock source is then used to clock the internal RTC.

For other details, see Chapter 5, “Clock Generation and Control”.

20.5.3 Interrupts

The RTC provides three different interrupt sources. All three interrupts are connected internally to the programmable interrupt controller and can be mapped to any interrupt channel. The three interrupt sources are:

- Periodic Interrupt—Can be set at rates from 122 µs to 500 ms.
- Alarm Interrupt—Can be set at rates from once-per-second to once-per-day.
- Update-Ended Interrupt—Provides update cycle status.

These three interrupts are enabled in RTC Control B (RTCCTLB) register. The RTC interrupt request is only active from low to high.

Before these interrupts can be used, they must be mapped to the programmable interrupt controller. For more information, see Table 15-4 on page 15-12.

20.5.4 Software Considerations

20.5.4.1 Initializing the RTC Divider Chain

An RTC reset event does not initialize either the divider chain or the Internal Oscillator Control (OSC_CTL) bit field in the RTC Control A (RTCCTLA) register (RTC index 0Ah). The internal RTC divider chain can be reset by writing a value of 110b or 111b to the OSC_CTL field. Writing either of these values resets the entire divider chain and disables the timebase updates. Resetting the divider chain is not required as part of the RTC initialization, but can be used to provide an accurate start time after initializing the timebase. To enable the divider chain and set the proper divisor, a value of 010b should be written to the OSC_CTL field.

20.5.4.2 Accessing the CMOS Memory

Access to CMOS RAM can be performed without any regard for RTC operations. However, if the RTC is disabled, the CMOS RAM will be unavailable, but not lost, unless both main and backup power to the RTC core is removed. Re-enabling the RTC will allow access to the CMOS RAM with its contents intact.

To access CMOS memory, first write the location of the desired byte to the RTC/CMOS RAM Index (RTCIDX) register (Port 0070h), then read the contents of that location from the RTC/CMOS RAM Data Port (RTCDATA) register (Port 0071h), or write the desired data byte to this data port.

20.5.4.3 Legacy NMI Enable Bit Moved

In PC/AT-compatible systems, bit 7 of the write-only RTC/CMOS RAM Index (RTCIDX) register (Port 0070h) is used to enable non-maskable interrupts (NMIs). On the ÉlanSC520 microcontroller, this NMI_ENB bit has been moved to bit 6 of the Interrupt Control (PICICR) register (MMCR offset D00h). Legacy software that needs to explicitly enable or disable interrupts should be modified accordingly. However, due to the difference in nature of the use of NMIs in legacy systems (memory parity errors and channel check) and in the ÉlanSC520 microcontroller (mappable to any interrupt source), compatibility issues are minimal. Writes to bit 7 of the RTC/CMOS RAM Index (RTCIDX) register (Port 0070h) on the ÉlanSC520 microcontroller have no effect and do not affect the index of the data accessed at the RTC/CMOS RAM Data Port (RTCDATA) register (Port 0071h).

For example:

```
-mov-al, 85h
-out-70h, al
```

and

```
-mov-al, 05h
-out-70h, al
```

Both sequences result in the contents of the RTC Alarm Hour (RTCALMHR) register (RTC index 05h) being accessed at the RTC/CMOS RAM Data Port (RTCDATA) register.

20.6 INITIALIZATION

The real-time clock is enabled at system reset.

1. Before initializing the internal registers, disable the time and calendar updates via the SET bit in RTC Control B (RTCCTLB) register (RTC index 0Bh).
2. Reset the RTC divider chain by writing a value of 11xb to the OSC_CTL field in the RTC Control A (RTCCTLA) register (RTC index 0Ah).

3. Initialize the ten time, calendar, and alarm registers in either binary or BCD data format.
4. Specify the format in the data mode via the DATE_MODE bit in the RTC Control B (RTCCTLB) register. All ten time, calendar, and alarm registers must use the same data mode, either binary or BCD.
5. Enable updates via the SET bit in RTC Control B (RTCCTLB) register.
6. Enable the divider chain by programming 010b in the OSC_CTL field. Time and update cycles will begin 500 ms after this write.

Steps 2 and 6 are necessary only if precision setting is required. Otherwise, the OSC_CTL field can be written to 010b in step 2, and step 6 can be skipped. The first update cycle, however, will occur at an undetermined time after updates are enabled.

When initialized, the RTC makes all updates in whatever data mode has been programmed. To change the data mode, the ten data bytes must be re-initialized.

20.6.1 RTC Reset

The RTC is not automatically reset by a system reset. There are three conditions that trigger an RTC reset:

- BBATSEN drops below 2.0 V (sampled when PWRGOOD asserts)—During operation from the main power supply, the backup battery voltage might drop below the trip voltage (2.0 V). The RTC is not reset until a PWRGOOD assertion occurs.
- Power is applied to VCC_RTC (at backup battery installation)—When the backup battery is plugged in, the RTC is immediately reset.
- No battery during power-up (sampled after PWRGOOD asserts)—If the system does not contain a backup battery and the BBATSEN line potential is below 2.0 V, the RTC is reset when PWRGOOD asserts.

Note that this RTC reset may or may not occur when a master power-on reset occurs, depending on the state of BBATSEN.

If the BBATSEN signal drops below the 2.0-V reference and PWRGOOD is Low, an internal RTC reset signal is generated to notify the user via the RTC_VRT bit (RTC index 0Dh) that the RTC contents are no longer valid.

CHAPTER

21 UART SERIAL PORTS

21.1 OVERVIEW

The ÉlanSC520 microcontroller includes two industry-standard 16550-compatible UARTs, both capable of running all existing 16450 and 16550 software.

The UARTs power up in 16450-compatible UART mode (also called *character mode* or *non-FIFO* mode). Each UART can be switched between the 16550-compatible mode (also called *FIFO mode*) and 16450-compatible mode under software control. In 16550-compatible mode, the receiver and the transmitter are each buffered with 16-byte FIFOs to offload the CPU from repetitive service routines.

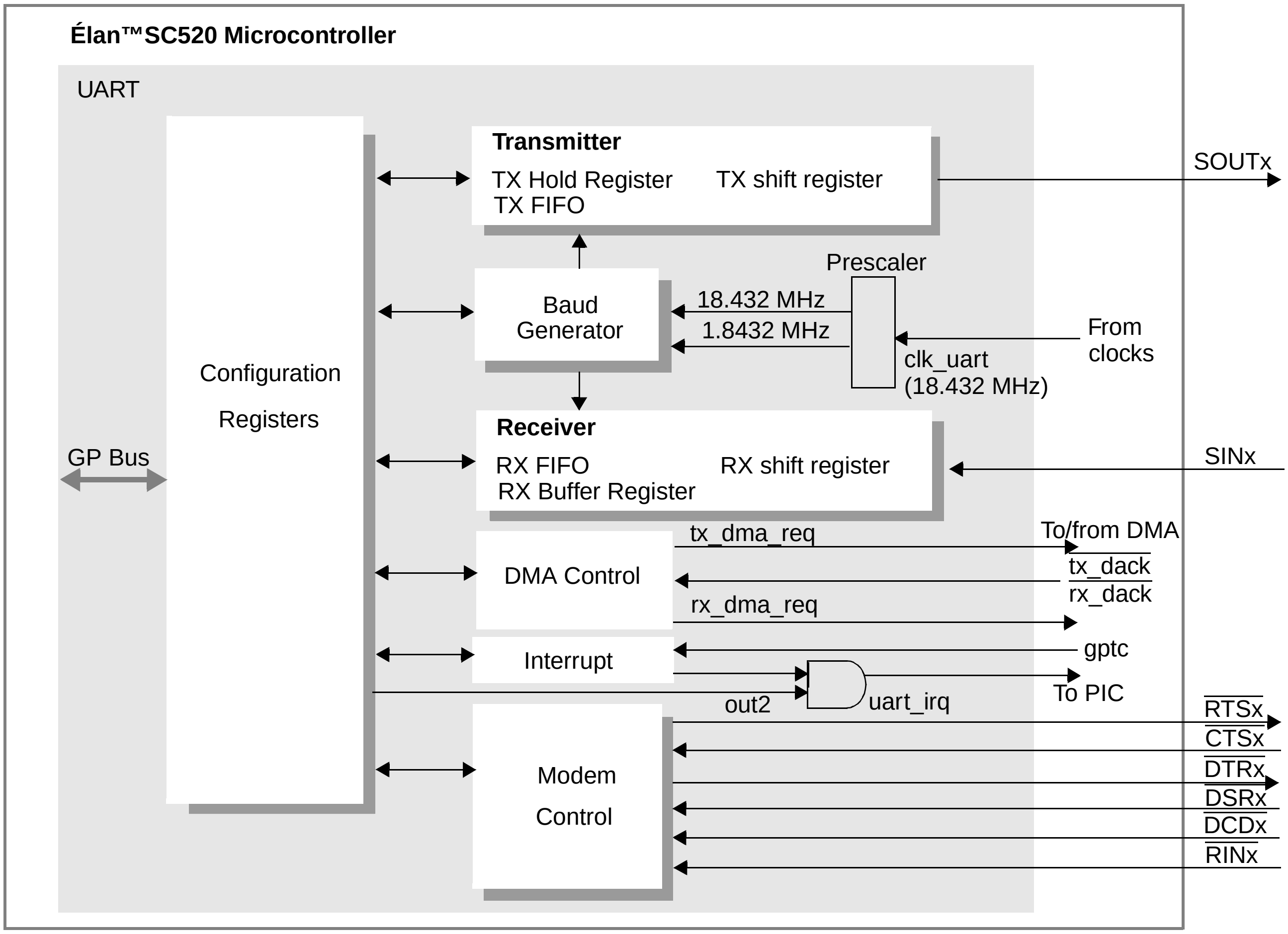
Features:

- Full UART pinout: SOUT, SIN, $\overline{\text{CTS}}$, $\overline{\text{RTS}}$, $\overline{\text{DSR}}$, $\overline{\text{DTR}}$, $\overline{\text{RIN}}$, and $\overline{\text{DCD}}$ for each UART
- In 16550-compatible mode, the transmitter and receiver are each buffered with 16-byte FIFOs
- Full-duplex (data can be sent in both directions simultaneously)
- DMA operation
- Internal baud-rate clock of 18.432 MHz or 1.8432 MHz
- Baud rates from DC to 1.152 Mbaud
  - Baud-rate generator provides input clock divisor from 1 to (2^{16}–1) to create 16x clock.
- Programmable serial interface:
  - 5-, 6-, 7-, and 8-bit character sizes
  - Even, odd, or no-parity bit generation and detection
  - 1, 1½, or 2 stop bits
  - Break generation and detection
  - Each UART's address decode can be individually disabled, allowing external devices to be used in their place.
- Internal Diagnostics:
  - False start bit detection
  - Complete status reporting capabilities
  - Break, parity error, overrun error, and framing error detection
  - Loopback controls for communications link fault isolation

21.2 BLOCK DIAGRAM

Figure 21-1 shows a block diagram of a single UART. The ÉlanSC520 microcontroller includes two UARTS that function identically to each other.

Figure 21-1 UART Block Diagram



21.3 SYSTEM DESIGN

UART 2 shares signals with the PIO31–PIO28 signals, as shown in Table 21-1. When enabled, the multiplexed signals shown in Table 21-1 either disable or alter any other function that uses the same pin.

Table 21-1 UART Signals Shared with Other Interfaces

PIO (Default) Function	Interface Function	Control Bit	Register
PIO31	$\overline{\text{RIN2}}$	PIO31_FNC	PIO31–PIO16 Pin Function Select (PIOPFS31_16) register (MMCR offset C22h)
PIO30	$\overline{\text{DCD2}}$	PIO30_FNC	
PIO29	$\overline{\text{DSR2}}$	PIO29_FNC	
PIO28	$\overline{\text{CTS2}}$	PIO28_FNC	

Both UARTs can work with full modem control signals (SOUTx, SINx, $\overline{\text{CTSx}}$, $\overline{\text{RTSx}}$, $\overline{\text{DSRx}}$, $\overline{\text{DTRx}}$, $\overline{\text{RINx}}$, and $\overline{\text{DCDx}}$) or with two wires only (SOUTx and SINx). If only two wires are used, the unused input port pins can be left unconnected. (There are internal pullup resistors on these signals.)

Each UART supports loopback mode. In this mode, the UART's transmitter output is internally connected with the receiver input. It is useful for testing the operation of a local UART channel without affecting the states of the UART output pins and independently of the state of the UART input pins.

When in loopback mode, $\overline{\text{RTS}}$ and $\overline{\text{DTR}}$ are internally connected to $\overline{\text{CTS}}$ and $\overline{\text{DSR}}$, respectively. In addition, the $\overline{\text{DTR}}$ and $\overline{\text{RTS}}$ signals are forced inactive. Therefore, hardware flow control is inactive.

UART interrupt requests are disabled while in loopback mode.

Table 21-2 lists the connection of DTE to DTE.

Table 21-2 Connection of DTE to DTE

DTE	DTE
SOUT	SIN
SIN	SOUT
$\overline{\text{CTS}}$	$\overline{\text{RTS}}$
$\overline{\text{RTS}}$	$\overline{\text{CTS}}$
$\overline{\text{DSR}}$	$\overline{\text{DTR}}$
$\overline{\text{DTR}}$	$\overline{\text{DSR}}$
$\overline{\text{RIN}}$	$\overline{\text{RIN}}$
$\overline{\text{DCD}}$	$\overline{\text{DCD}}$

21.4 REGISTERS

The UARTs are controlled by the memory-mapped registers listed in Table 21-3 and the direct-mapped registers listed in Table 21-4.

Table 21-3 UART Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
Address Decode Control	ADDDECCTL	80h	UART 1 and UART 2 disables
PIO31–PIO16 Pin Function Select	PIOPFS31_16	C22h	PIO or interface function select: $\overline{\text{RIN2}}$, $\overline{\text{DCD2}}$, $\overline{\text{DSR2}}$, $\overline{\text{CTS2}}$
Clock Select	CLKSEL	C26h	CLKTIMER[CLKTEST] pin enable, CLKTEST output select options (18.432 MHz or 1.8432 MHz UART), CLKTIMER or CLKTEST select
UART 1 General Control UART 2 General Control	UART1CTL UART2CTL	CC0h CC4h	Clock source; receive TC interrupt and transmit TC interrupt enables
UART 1 General Status UART 2 General Status	UART1STA UART2STA	CC1h CC5h	Receive TC and transmit TC interrupts status
UART 1 FIFO Control Shadow UART 2 FIFO Control Shadow	UART1FCRSHAD UART2FCRSHAD	CC2h CC6h	Information written to the direct-mapped UART x FIFO Control (UARTxFCR) register
UART 1 Interrupt Mapping	UART1MAP	D28h	UART 1 interrupt mapping

Table 21-3 UART Registers—Memory-Mapped (Continued)

Register	Mnemonic	MMCR Offset Address	Function
UART 2 Interrupt Mapping	UART2MAP	D29h	UART 2 interrupt mapping

Table 21-4 UART Registers—Direct-Mapped

Register	Mnemonic	I/O Address	Function
UART 1 Transmit Holding UART 2 Transmit Holding	UART1THR UART2THR	02F8h 03F8h	Byte to be transmitted
UART 1 Receive Buffer UART 2 Receive Buffer	UART1RBR UART2RBR	02F8h 03F8h	Received byte
UART 1 Baud Clock Divisor Latch LSB UART 2 Baud Clock Divisor Latch LSB	UART1BCDL UART2BCDL	02F8h 03F8h	Least significant byte of a 16-bit baud-rate clock divisor used to generate the 16x baud clock
UART 1 Baud Clock Divisor Latch MSB UART 2 Baud Clock Divisor Latch MSB	UART1BCDH UART2BCDH	02F9h 03F9h	Most significant byte of a 16-bit baud-rate clock divisor used to generate the 16x baud clock
UART 1 Interrupt Enable UART 2 Interrupt Enable	UART1INTENB UART2INTENB	02F9h 03F9h	Interrupt enables: modem status, receiver line status, transmit holding empty, received data available, and time-out
UART 1 Interrupt ID UART 2 Interrupt ID	UART1INTID UART2INTID	02FAh 03FAh	FIFO mode indication, interrupt identification, interrupt pending status
UART 1 FIFO Control UART 2 FIFO Control	UART1FCR UART2FCR	02FAh 03FAh	Trigger level for received data available interrupt, DMA mode, transmitter FIFO and receiver FIFO clear, FIFO enable for 16550-compatible mode
UART 1 Line Control UART 2 Line Control	UART1LCR UART2LCR	02FBh 03FBh	Divisor latch access (DLAB), break, stick parity, parity, asynchronous data parity, stop, transmit/receive word length
UART 1 Modem Control UART 2 Modem Control	UART1MCR UART2MCR	02FCh 03FCh	Loopback diagnostic mode, UARTx interrupt enable, RTSx and DTRx control
UART 1 Line Status UART 2 Line Status	UART1LSR UART2LSR	02FDh 03FDh	FIFO error, transmitter empty indicator, Transmitter Holding register or transmit FIFO empty, break indicator, framing error, parity error, overrun error, data ready
UART 1 Modem Status UART 2 Modem Status	UART1MSR UART2MSR	02FEh 033FEh	Real-time and latched status bits for DCDx, RINx, DSRx and CTSx
UART 1 Scratch Pad UART 2 Scratch Pad	UART1SCRATCH UART2SCRATCH	02FFh 03FFh	Temporary data storage

21.5 OPERATION

Each UART performs:

- *Serial-to-parallel* conversion on data characters received from a modem or a peripheral device
- *Parallel-to-serial* conversion on those data characters written by the CPU or DMA controller

During communication, data is transmitted and received in *frames*. The frame format, as well as the baud rate, must be the same on the transmitter and receiver. The frame format is determined by the settings in the UART x Line Control (UARTxLCR) register. Each frame begins with a start bit (Low) and ends with one, one and a half, or two stop bits (High). After the start bit is transmitted/received, the data bits, which can be programmed to a length of 5, 6, 7 or 8 bits, are transmitted/received serially with least significant bit first.The last data bit may be followed by an optional parity bit that is enabled using the PENB bit in the UART x Line Control (UARTxLCR) register. The line is always held High between frames (idle state).

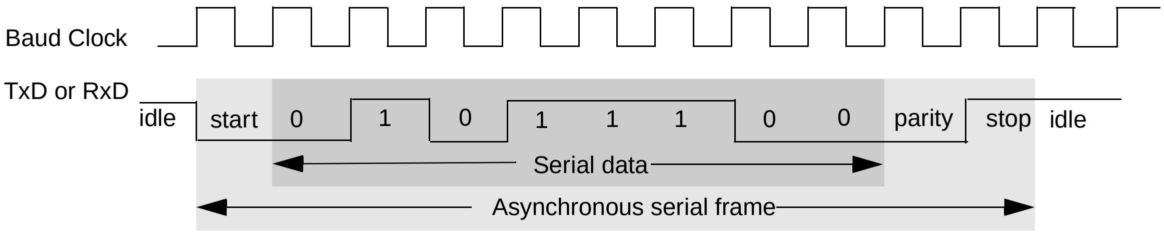
- *Transmission* of a frame is initiated when a byte is written to the UART x Transmit Holding (UARTxTHR) register.
- *Reception* of a frame is initiated when a start bit is received (the SIN input is driven Low for one baud-rate clock period).

Figure 21-2 shows the frame configurations supported and the bit stream sequence for a UART on the ÉlanSC520 microcontroller. Figure 21-3 shows an actual UART frame during transmission with configuration of even parity, one stop bit, and eight data bits.

Figure 21-2 UART Frame Configuration

Start Bit	5 Data Bits	Optional 6th Data Bit	Optional 7th Data Bit	Optional 8th Data Bit	Optional Parity Bit	Stop Bit	Optional 0.5 or 2nd Stop Bit

Figure 21-3 UART Frame Transmission



Asynchronous transmission of 03Ah as 8 bits of data, even parity, one stop bit

Each UART includes a programmable baud-rate generator that is capable of dividing the timing reference clock input by divisors of 1 to ($2^{16} - 1$), and producing a 16 x clock for driving the internal transmitter/receiver logic.

For each UART, six handshaking signals are provided:

- $\overline{\text{DTRx}}$ (Data Terminal Ready) output—When the signal is Low, it informs the modem set that the UART is ready to establish a communications link. The $\overline{\text{DTRx}}$ output signal can be asserted and deasserted by the UART x Modem Control (UARTxMCR) register. Loopback mode operation holds $\overline{\text{DTRx}}$ in its inactive state.
- $\overline{\text{DSRx}}$ (Data Set Ready) input—When the signal is Low, it indicates that the modem is ready to establish the communications link with the UART. The state of the $\overline{\text{DSRx}}$ pin can be tested in the UART x Modem Status (UARTxMSR) register. The DDSR status bit provided in the UART x Modem Status (UARTxMSR) register indicates if the $\overline{\text{DSRx}}$ signal has changed state since the register was last read. An interrupt can also be generated upon $\overline{\text{DSRx}}$ change.
- $\overline{\text{RTSx}}$ (Request-to-Send) output—When the signal is Low, it informs the modem that the UART is ready to exchange data. The $\overline{\text{RTSx}}$ output signal can be asserted and deasserted by the UART x Modem Control (UARTxMCR) register. Loopback mode operation holds $\overline{\text{RTSx}}$ in its inactive state.
- $\overline{\text{CTSx}}$ (Clear-to-Send) input—When the signal is Low, it indicates that the modem is ready to exchange data. The state of the $\overline{\text{CTSx}}$ pin can be tested in the UART x Modem Status (UARTxMSR) register. The DCTS status bit in the UART x Modem Status (UARTxMSR) register indicates if the $\overline{\text{CTSx}}$ signal has changed state since the register was last read. An interrupt can also be generated upon $\overline{\text{CTSx}}$ change.
- $\overline{\text{DCDx}}$ (Data Carrier Detect) input—When the signal is Low, it indicates that the data carrier has been detected by the modem and that contact between it and the other modem is established. The state of the $\overline{\text{DCDx}}$ pin can be tested in the UART x Modem Status (UARTxMSR) register. The DDCD status bit in the UART x Modem Status (UARTxMSR) register indicates if the $\overline{\text{DCDx}}$ signal has changed state since the register was last read. An interrupt can also be generated upon $\overline{\text{DCDx}}$ change.
- $\overline{\text{RINx}}$ (Ring Indicator) input—When the signal is Low, it indicates that a telephone ringing signal has been received by the modem. The state of the $\overline{\text{RINx}}$ pin can be tested in the UART x Modem Status (UARTxMSR) register. The TERI status bit is also provided in the UART x Modem Status (UARTxMSR) register indicates if the $\overline{\text{RINx}}$ signal has changed state from asserted to deasserted since the register was last read. An interrupt can also be generated upon $\overline{\text{RINx}}$ deassertion.

21.5.1 Data Transmission

21.5.1.1 16450-Compatible UART Mode

In 16450-compatible (non-FIFO or character) mode:

1. Data written to the UART x Transmit Holding (UARTxTHR) register is subsequently latched into the internal transmitter shift register when the transmitter shift register is empty.
2. Once data has been latched into the internal transmitter shift register, the Transmit Holding Register Empty (THRE) bit in the UART x Line Status (UARTxLSR) register goes to 1 (optionally generating a UART interrupt).
3. The application is once again permitted to write data to the UART x Transmit Holding (UARTxTHR) register.

Note that writing to the UART x Transmit Holding (UARTxTHR) register in this mode when the THRE bit is not set can result in incorrect data being transmitted.

The Transmitter Empty (TEMT) bit in the UART x Line Status (UARTxLSR) register is set in this mode if both the UART x Transmit Holding (UARTxTHR) register and internal transmitter shift register are empty. An application could write two bytes consecutively to the UART x Transmit Holding (UARTxTHR) register without checking THRE if TEMT is detected as set.

21.5.1.2 16550-Compatible UART Mode

In 16550-compatible (FIFO) mode:

1. Data written to the UART x Transmit Holding (UARTxTHR) register address is latched into the next available FIFO location.
2. The transmit data is shifted directly out of the first FIFO entry with valid data. There are a total of 16 bytes in the FIFO. Thus, if the TEMT bit is set, then software can safely write 16 bytes consecutively to the UART x Transmit Holding (UARTxTHR) register address for transmission.

The THRE (which can optionally generate an interrupt) and TEMT bits are set whenever the last character is shifted from the FIFO and the FIFO becomes empty. If the number of characters currently in the FIFO is unknown, software should wait for the THRE or TEMT bit to be set before writing additional data.

21.5.2 Data Reception

21.5.2.1 16450-Compatible UART Mode

In 16450-compatible mode:

1. Received data is shifted from the SIN pin into the internal receive shift register.
2. Once an entire UART frame has been received, the character is transferred from the internal receive shift register into the UART x Receive Buffer (UARTxRBR) register.
3. The Data Ready (DR) bit in the UART x Line Status (UARTxLSR) register is set to 1 (optionally generating an interrupt).

Note that the DR bit is cleared by a read of the UART x Receive Buffer (UARTxRBR) register. If a second character is transferred into the UART x Receive Buffer (UARTxRBR) register before software reads the first one (i.e., the DR bit is still set), then the Overrun Error (OE) bit in the UART x Line Status (UARTxLSR) register is set to 1, and the first character is destroyed. Subsequent received bytes continue to overwrite the UART x Receive Buffer (UARTxRBR) register until software reads the UART x Receive Buffer (UARTxRBR) register.

21.5.2.2 16550-Compatible UART Mode

In 16550-compatible mode:

1. Received data is shifted into the internal receive shift register.
2. Once an entire UART frame has been received, the character is transferred from the internal receive shift register into the FIFO.
   - If the FIFO was empty, the DR bit in the UART x Line Status (UARTxLSR) register is set and remains set until the FIFO is completely emptied by software.
   - If the received character places the FIFO above the limit indicated by the RFRT field in the UART x FIFO Control (UARTxFCR) register and received data available interrupts are enabled, then an interrupt is generated.
   - A receive FIFO time-out interrupt occurs when this interrupt is enabled if data is present in the FIFO and no received data have been placed into or read from the receive FIFO

in four character times. In 16550-compatible mode, the Overrun Error (OE) bit is set if a new character is completely received into the shift register when the FIFO is already 100% full. Data in the FIFO is not overwritten by this overrun. However, the data in the shift register is lost.

21.5.3 Error Handling

Received data can contain three types of abnormal conditions in addition to the overrun error:

- Parity errors
- Framing errors
- Break indications

21.5.3.1 Parity Error

A parity error indicates that the parity bit for the character in error did not match the parity indicated by the Even Parity Select (EPS) bit, Stick Parity Enable (SP) bit, and the Asynchronous Data Parity Enable (PENB) bit in the UART x Line Control (UARTxLCR) register.

21.5.3.2 Framing Error

A framing error indicates that the bit following the parity bit (if parity is enabled) or following the last data bit (if parity is not enabled) was detected to be a logic 0. To resynchronize following this type of error, the UART assumes that the framing error was due to the next start bit occurring too early. It samples the start bit twice (once as the erroneous stop bit of the first character and once as the start bit of the second character) before sampling the data for the second bit.

21.5.3.3 Break Indication

A break indication means that the received data input was detected to be 0 for a time longer than a full UART frame (including start bit, data bits, parity, and stop bits). The character loaded into the FIFO on a break indication is always 0, and subsequent characters are loaded normally once the receive input returns to its idle state (high).

21.5.3.4 Error Reporting

21.5.3.4.1 16450-Compatible UART Mode

In 16450-compatible mode, the error bits (OE, PE, FE, and BI) in the UART x Line Status (UARTxLSR) register indicate that the error was detected during reception of the current byte in the UART x Receive Buffer (UARTxRBR) register. If receiver line status Interrupts are enabled, any of the OE, PE, FE, or BI conditions trigger an interrupt.

21.5.3.4.2 16550-Compatible UART Mode

In 16550-compatible mode, the error bits (PE, FE, and BI) are set only when an error condition is detected in the character at the top of the FIFO. Since reading the UART x Receive Buffer (UARTxRBR) register causes the FIFO to advance to the next received character, the error bits must be read from the UART x Line Status (UARTxLSR) register prior to the data being read from the FIFO. The ERR_IN_FIFO bit in the UART x Line Status (UARTxLSR) register can be used to detect if *any* of the characters in the FIFO (not just the one at the top) had errors.

All of the error bits, with the exception of the ERR_IN_FIFO bit, are cleared when the UART x Line Status (UARTxLSR) register is read. The ERR_IN_FIFO bit is cleared when the UART x Line Status (UARTxLSR) register is read and all data present in the FIFO is error-free, or the FIFO becomes empty.

If receiver line status interrupts are enabled, any of the OE, PE, FE, or BI conditions trigger an interrupt. Note that the ERR_IN_FIFO cannot directly generate an interrupt.

21.5.4 Configuration Information

21.5.4.1 Baud Rate

To generate the baud rate of the transfer, the UART clock is divided by a divisor value chosen by the programmer. The UART's baud-rate generator automatically calculates the baud rate from the divisor value programmed into the two UART x Baud Clock Divisor Latch MSB and LSB registers. These registers are read at initialization to set the baud rate for the transfer. The baud rate is calculated according to the following equation:

Baudrate = clockfrequency / 16 * BAUDDIV

Here, clock frequency refers to the frequency of the main reference clock, 1.8432 MHz or 18.432 MHz. This frequency is determined by the CLKSRC bit in the UART x General Control (UARTxCTL) register. BAUDDIV is defined by the UART x Baud Clock Divisor Latch MSB and LSB registers. Table 21-5 lists the divisor value (in decimal and hexadecimal) to use with each clock frequency to achieve common baud rates.

Table 21-5 Baud Rates, Divisors, and Clock Source

	DIV[15–0] (Decimal)		DIV[15–0] (Hexadecimal)	
Baud Rate	**1.8432 MHz**	**18.432 MHz**	**1.8432 MHz**	**18.432 MHz**
300 baud	384d	3840d	0180h	0F00h
600 baud	192d	1920d	00C0h	0780h
2400 baud	48d	480d	0030h	01E0h
4800 baud	24d	240d	0018h	00F0h
7200 baud	16d	160d	000Fh	00A0h
9600 baud	12d	120d	000Ch	0078h
14.4 kbaud	8d	80d	0008h	0050h
19.2 kbaud	6d	60d	0006h	003Ch
57.6 kbaud	2d	20d	0002h	0014h
115.2 kbaud	1d	10d	0001h	000Ah
144 kbaud		8d		0008h
192 kbaud		6d		0006h
288 kbaud		4d		0004h
576 kbaud		2d		0002h
1.152 Mbaud		1d		0001h

21.5.4.2 Hardware Flow Control

When the EMSI bit of the UART x Interrupt Enable (UARTxINTENB) register is set, the modem status interrupt is enabled to facilitate the hardware flow control. The interrupts are triggered by changes in the following control lines: $\overline{\text{CTS}}$x, $\overline{\text{DTR}}$x, $\overline{\text{RIN}}$x, and $\overline{\text{DCD}}$x.

21.5.4.3 Operating Modes

16450-compatible UART mode and 16550-compatible UART mode can be setup by setting or clearing the FIFO_ENB bit of UART x FIFO Control (UARTxFCR) register. The ÉlanSC520 microcontroller UARTs can be switched between the 16550-compatible mode and 16450-compatible mode under software control.

When in 16550-compatible mode, the receiver and transmitter FIFO buffers can be cleared by the RF_CLR and TF_CLR bits in the UART x FIFO Control (UARTxFCR) register, respectively. The receiver FIFO trigger level can be programmed by RFRT field of the UART x FIFO Control (UARTxFCR) register.

21.5.5 DMA Interface

To support higher serial data transfer rates, both UARTs support DMA. For more detailed information on the operation of the GP-DMA controller, see Chapter 14, “GP Bus DMA Controller”.

The ÉlanSC520 microcontroller’s DMA interface provides up to four 8-bit DMA channels to support the two integrated UARTs. Each UART can use up to two DMA channels: one for receive and one for transmit. DMA transfers are supported for both 16450- and 16550-compatible modes.

The DMA controller can perform read and write operations in single cycle, demand, or block transfer mode. However, block transfer mode is not supported for UART transfers.

21.5.5.1 Transmit DMA

The internal tx_dma_req signal from the UART is asserted whenever there is room for another transmit character in the UART.

- For 16450-compatible UART mode, this means that either the internal transmitter shift register or the UART x Transmit Holding (UARTxTHR) register can accept a character.
- For 16550-compatible mode, this means that the transmit FIFO is not full.

21.5.5.2 Receive DMA

The internal rx_dma_req signal from the UART is asserted:

- For 16550-compatible mode, whenever the receive trigger level was reached or a time-out has occurred. The rx_dma_req signal is made inactive when the receive FIFO is completely empty.
- For 16450-compatible UART mode, whenever the UART x Receive Buffer (UARTxRBR) register contains a valid character.

For either mode, the internal rx_dma_req signal is deasserted whenever a character is received with an error condition. This allows software to inspect the error condition before the error status is cleared by a subsequent DMA transfer. Once software has cleared the error status by a read of the UART x Line Status (UARTxLSR) register, the rx_dma_req is asserted when another character is present in the UART.

21.5.6 Clocking Considerations

The clock input to the UARTs to support standard PC/AT baud selections is 1.8432 MHz. A clock of 18.432 MHz is also provided to the UARTs for fast serial communication. This frequency is determined by the CLK_SRC bit in the UART x General Control (UARTxCTL) register.

21.5.7 Interrupts

Each of the two UARTs on the ÉlanSC520 microcontroller provides its own interrupt to the programmable interrupt controller. Each ÉlanSC520 microcontroller serial port has an internal interrupt signal that can be mapped to uart1_irq or uart2_irq. For detailed information, see Chapter 15, “Programmable Interrupt Controller”.

Table 21-6 provides a summary of UART interrupt sources for both DMA and serial port interrupts.

Interrupts generated by the UARTs are cleared in a variety of ways, depending on the source event. For details about clearing a particular event, see the event's status bit description in the *Élan™SC520 Microcontroller Register Set Manual*, order #22005.

Table 21-6 UART Interrupt Programming Summary

Interrupt Description	Enable Register[1, 2]	Status Register[3]	Source Event	Polled Status Bit
Receive DMA transfer count	UART x General Control (UARTxCTL)	UART x General Status (UARTxSTA)	UART x Receive TC Detected	RXTC_DET
Transmit DMA transfer count			UART x Transmit TC Detected	TXTC_DET
Modem status change	UART x Interrupt Enable (UARTxINTENB)	UART x Modem Status (UARTxMSR)	Delta data carrier detect	DDCD
			Trailing edge ring indicator	TERI
			Delta data set ready	DDSR
			Delta clear to send	DCTS
Receiver line status		UART x Line Status (UARTxLSR)	Break indicator	BI
			Framing error	FE
			Parity error	PE
			Overrun error	OE
Transmitter holding register empty			Transmit holding register (16450-compatible mode) or transmitter FIFO (16550-compatible mode) empty	THRE
Received data available			Data ready (16450-compatible mode)	DR
		—[4]	FIFO trigger level reached (16550-compatible mode)	—
FIFO time-out[5]			FIFO time-out (16550-compatible mode)	—

Notes:

1. Before any UART interrupt is enabled, the corresponding UART x Interrupt Mapping (UARTxMAP) register must be configured to route the interrupt to the appropriate interrupt request level and priority.

2. The OUT2 bit in the UART x Modem Control (UARTxMCR) register is used as a master control for UART interrupts. The OUT2 bit must be set for UART interrupts to be generated. Status bits can be read even when interrupts are disabled.

3. If two of the interrupts enabled in the UART x Interrupt Enable (UARTxINTENB) register are pending simultaneously, the highest-priority interrupt is identified in the INT_ID bit field of the UART x Interrupt ID (UARTxINTID) register.

4. There are no polled-status bits for the FIFO trigger level and FIFO time-out events. These events are indicated by the INT_ID bit field only.

5. The FIFO time-out interrupt is enabled with the received data available interrupt by the ERDAI bit in the UART x Interrupt Enable (UARTxINTENB) register.

21.5.7.1 Serial Port Interrupts

Each serial port supports the standard UART interrupts. These include:

- Received data available or FIFO trigger level reached
- Transmit Holding register empty (THRE)
- Modem status change (including clear-to-send, data-set-ready, ring indicator, data carrier detect)
- Line Status register receiver interrupts (including overrun error, parity error, framing error and break interrupt)

In 16550-compatible mode, the FIFO time-out interrupt is also enabled when the received data available interrupt is enabled.

The UART interrupt sources and their priority are shown in Table 21-7. If two interrupt sources are pending simultaneously, the highest priority interrupt is indicated by the ID field of the UART x Interrupt ID (UARTxINTID) register. When the interrupt source is cleared, a subsequent read from this port returns the next highest priority interrupt source.

Table 21-7 Serial Port Interrupt and Interrupt Priority

INT_ID Bit Field	Description	Identification Priority
000b	Modem status change	Fourth (Lowest)
001b	Transmit holding register empty (16540-compatible mode)/Transmit FIFO empty (16550-compatible mode)	Third
010b	Received data available (16540-compatible mode)/ Receiver FIFO trigger (16550-compatible mode)	Second
011b	Receive line status	First (Highest)
100b	Not used	—
101b	Not used	—
110b	FIFO time-out	Second
111b	Not used	—

Note: *In 16450-compatible mode, the INT_ID2 bit always reads back 0. The INT_ID bit field is located in the UART x Interrupt ID (UARTxINTID) register.*

The UART interrupts are enabled by the Interrupt Enable register and read from the UART x Interrupt ID (UARTxINTID) register.

21.5.7.2 DMA Interrupts

Each UART can generate an interrupt when the Transfer Count (TC) signal associated with DMA transfers is asserted. Four enable bits and four status bits are available for these interrupts: transmit and receive Transfer Count reached for each UART. These bits are located in the UART x General Control (UARTxCTL) and UART x General Status (UARTxSTA) registers.

21.5.7.3 Interrupt Disable

Each UART interrupt request can be disabled (gated low) prior to the programmable interrupt controller by clearing to 0 the OUT2 bit in the UART x Modem Control (UARTxMCR) register. Note that setting the LOOP bit in the MCR also disables the UART interrupt request. Therefore, interrupts are not propagated to the PIC while in loopback mode.

21.6 INITIALIZATION

At system reset, the serial port is disabled. To be enabled, it must be configured by software.

1. Configure the UART by programming the desired baud rate, character length, stop-bits, and parity.
2. Enable interrupts and DMA operation as desired. Note that for UART interrupts to propagate to the programmable interrupt controller, the OUT2 bit in the UART x Modem Control (UARTxMCR) register must be set to 1.

After the UART is enabled, it powers up as a 16450-compatible device. It can be switched between 16550-compatible mode and 16450-compatible mode under software control.

3. Enable 16550-compatible mode by setting the FIFO_ENB bit of UART x FIFO Control (UARTxFCR) register. Note that the contents of this write-only register can be read back in the UART x FIFO Control Shadow (UARTxFCRSHAD) register.

CHAPTER

22 SYNCHRONOUS SERIAL INTERFACE

22.1 OVERVIEW

The ÉlanSC520 microcontroller includes a synchronous serial interface (SSI). The SSI provides efficient full-duplex and half-duplex, bidirectional communication to peripheral devices. The interface can be used to configure and monitor the status of devices such as ISDN transceivers, EEPROMs, SLACs, audio CODECs, LCD drivers, DSPs, etc. It can easily communicate with slave interfaces that are compatible to Motorola's Serial Peripheral Interface (SPI), Motorola's Serial Communication Port (SCP), National Semiconductor Corporation's Microwire, and other industry standards.

Features of the SSI include:

- Full or half-duplex operation
- Compatible with either four-pin or three-pin peripheral devices
- Multiple device enables through programmable I/O (PIO) pins
- Configurable clock idle state and phase
- Configurable bit shifting order, most significant bit or least significant bit first
- Programmable SSI clock speed, from 64 kHz to 8 MHz
- Transaction complete status, available as interrupt

22.2 BLOCK DIAGRAM

A block diagram of the SSI is shown in Figure 22-1. System diagrams, as well as timing diagrams, of a three-pin SSI interface and a four-pin SSI interface are shown on page 22-3.

22.3 SYSTEM DESIGN

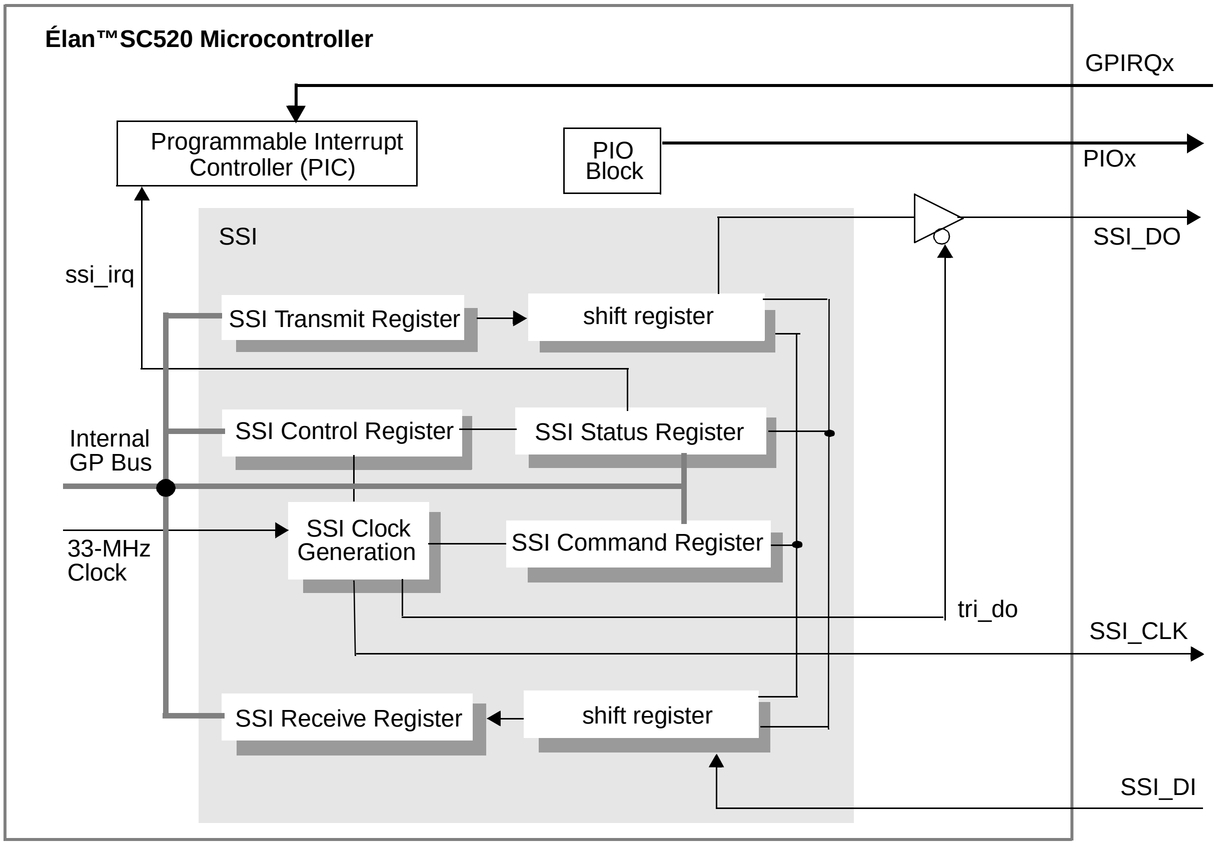
Three SSI pins are provided: clock out (SSI_CLK), data out (SSI_DO), and data in (SSI_DI). The SSI_DO signal is normally at high-impedance when no transmit transaction is active on the SSI. An external pullup or pulldown resistor can be added to this pin, if required by the slave device.

Most slave devices require an enable pin to be asserted during an operation and deasserted when not in operation. PIO pins on the ÉlanSC520 microcontroller can be used for this purpose.

Many slave SSI ports provide an interrupt output pin to the ÉlanSC520 microcontroller. These can be routed to one of the GPIRQx pins, which are multiplexed with PIOs. See Chapter 15, “Programmable Interrupt Controller”, and Chapter 2, “Pin Information”, for information on external interrupts.

See the *Élan™SC520 Microcontroller Data Sheet*, order #22003, for timing tables and additional timing diagrams.

Figure 22-1 SSI Block Diagram



22.4 REGISTERS

The memory-mapped registers shown in Table 22-1 are used to configure the SSI.

Table 22-1 Synchronous Serial Interface Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
SSI Control	SSICTL	CD0h	SSI clock speed, interrupt enable, clock phase, clock idle state, bit order
SSI Transmit	SSIXMIT	CD1h	Byte or data to be shifted out to SSI_DO pin
SSI Command	SSICMD	CD2h	Transfer command to be executed: transmit, receive, or simultaneous transmit/receive
SSI Status	SSISTA	CD3h	Busy status, transaction complete status
SSI Receive	SSIRCV	CD4h	Byte or data shifted in from SSI_DI pin
SSI Interrupt Mapping	SSIMAP	D41h	SSI interrupt mapping

22.5 OPERATION

Synchronous serial interface describes a port that can be implemented in several ways. Typically, the microcontroller port is called the *master* and one or more peripheral device ports are *slaves*.

- The master port (ÉlanSC520 microcontroller) configures a slave by serial transmission of slave commands, addresses, and data.
- A slave (peripheral device) can send requested status information or data, similarly.

Options in the SSI Control (SSICTL) register (MMCR offset CD0h), along with software-controlled device enable signals, can be used to customize the SSI port to emulate a variety of formats. Its flexibility allows simple communication with multiple devices, reducing software overhead.

Three commands are provided to initiate the transfer of data through the SSI. A write to the SSI Control (SSICTL) register selects the type of cycle to execute and initiates the cycle. The three SSI commands are:

- Transmit-only (half-duplex)—In a transmit transaction, or *cycle*, the contents of the SSI Transmit (SSIXMIT) register (MMCR offset CD1h) are serially shifted onto the SSI_DO pin.
- Receive-only (half-duplex)—A receive transaction shifts a byte from SSI_DI to the SSI Receive (SSIRCV) register (MMCR offset CD4h).
- Simultaneously transmit and receive (full-duplex)

The ÉlanSC520 microcontroller SSI is always the master and drives the clock when the SSI command is given. Slave devices cannot drive this clock. All transactions complete within eight clock cycles.

22.5.1 Usage Scenarios

22.5.1.1 Four-Pin Interface

A full-duplex, four-pin port has separate input and output data pins. Figure 22-2 is a block diagram of the SSI connected to multiple four-pin slave devices. A transmit and receive operation can take place within the same eight clocks, as shown in Figure 22-3. Many four-pin slave ports, however, operate in half-duplex. In that case, Figure 22-5 would apply.

22.5.1.2 Three-Pin Interface

SSI_DO and SSI_DI can be externally shorted to interface three-pin peripheral devices, as in Figure 22-4. This creates an I/O signal that matches slave I/O pins. Three-pin ports multiplex the data output and input for half-duplex communication. A typical half-duplex operation, as shown in Figure 22-5, is implemented with a two-byte protocol in non-inverted phase and clock modes. The first byte sends a command/address byte to the slave, which indicates that the data for the second byte will be transmitted or received. The slave should begin to transmit or receive when it detects an active clock.

Figure 22-2 SSI Four-Pin Interface

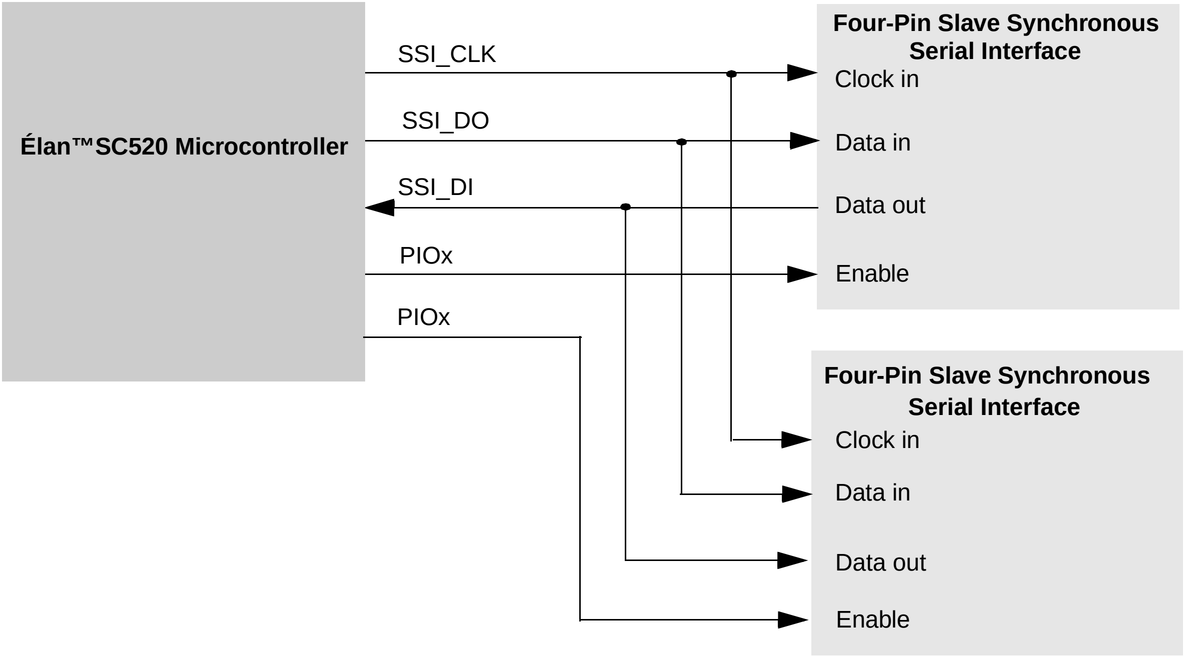

Élan™SC520 Microcontroller
SSI_CLK
SSI_DO
SSI_DI
PIOx
PIOx
Four-Pin Slave Synchronous Serial Interface
Clock in
Data in
Data out
Enable
Four-Pin Slave Synchronous Serial Interface
Clock in
Data in
Data out
Enable

Figure 22-3 SSI Simultaneous Transmit and Receive

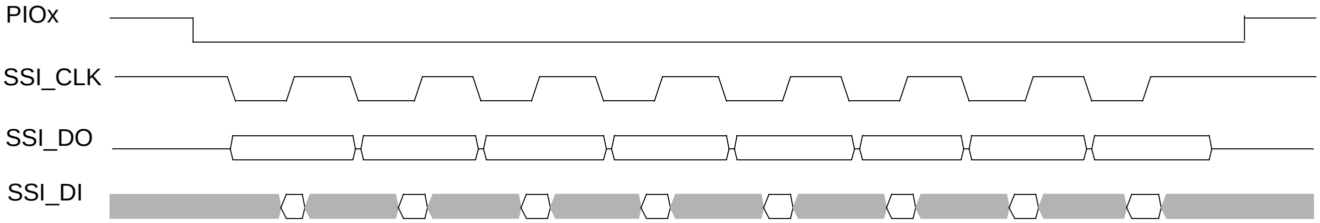


Figure 22-4 SSI Three-Pin Interface

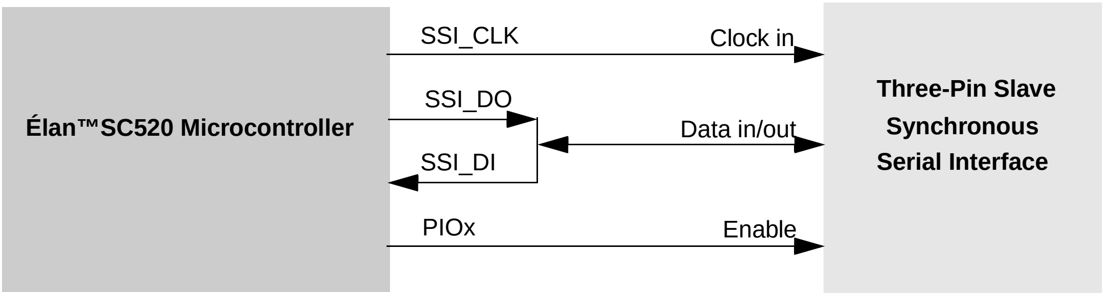
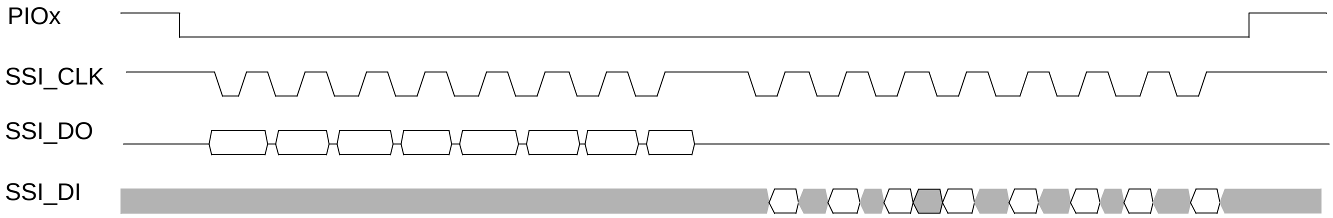

Élan™SC520 Microcontroller
SSI_CLK
SSI_DO
SSI_DI
PIOx
Clock in
Data in/out
Enable
Three-Pin Slave Synchronous Serial Interface

Figure 22-5 SSI Typical Half-Duplex Communication, Non-Inverted Phase and Clock Modes



22.5.2 Configuration Information

The MSBF_ENB, CLK_INV_ENB, and PHS_INV_ENB bits in the SSI Control (SSICTL) register (MMCR offset CD0h) define the order of the bits, the clock idle state, and the clock edge upon which data is transmitted/received (phase).

- The SSI should be configured to assert SSI_DO on the same clock edge that the slave uses to transmit.
- SSI_DI is sampled on the opposite clock edge.

22.5.2.1 Bit Order

The SSI bit order can be changed by the SSI Most Significant Bit First Mode Enable (MSBF_ENB) bit. A byte can be transferred with the least significant bit first (LSBF) or most significant bit first (MSBF). MSBF mode is enabled when this bit is written to a 1. This mode is common for input and output data.

22.5.2.2 Clock Idle State

The clock idle state is controlled by the SSI Inverted Clock Mode Enable (CLK_INV_ENB) bit. The absolute time to drive/sample is unchanged by the CLK_INV_ENB bit.

- When the CLK_INV_ENB bit has a value of 0, SSI_CLK idles High, then pulses Low during a transaction.
- If the CLK_INV_ENB bit is written to a 1, the clock idle state is Low.

22.5.2.3 Clock Phase

The clock phase, relative to the serial data, is determined by the SSI Inverted Phase Mode Enable (PHS_INV_ENB) bit.

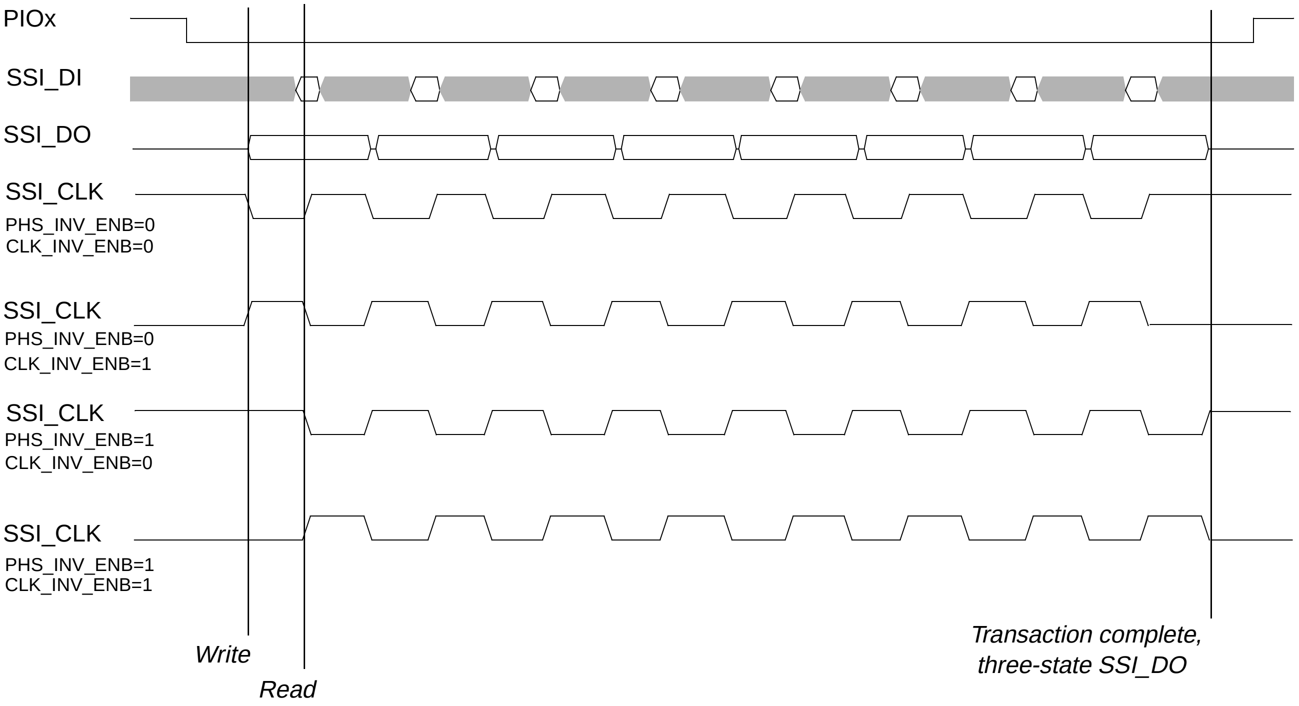
- In *non-inverted phase mode*, data is transmitted on odd edges of the SSI clock, and received on even edges.Therefore, the first SSI clock edge of a transaction shifts out the first bit on SSI_DO, if writing. SSI_DI data is latched, during a receive transaction, on even edges of the SSI clock.
- *Inverted phase mode* requires that the SSI_DI signal be sampled on the first (odd) clock edge(s). Consequently, the first bit is asserted on SSI_DO one-half an SSI clock cycle before the first edge of SSI_CLK, and even edges afterwards.

22.5.3 Bus Cycles

The four possible combinations of CLK_INV_ENB and PHS_INV_ENB are shown in Figure 22-6.

- Microwire compatibility is configured when the PHS_INV_ENB, CLK_INV_ENB, and MSBF_ENB bits are all set to 1.
- The SSI is compatible with an SCP interface when the PHS_INV_ENB and CLK_INV_ENB bits are cleared to 0, and the MSBF_ENB bit is set to 1.

Figure 22-6 SSI Clock Phase and Clock Idle State: Effects on Data

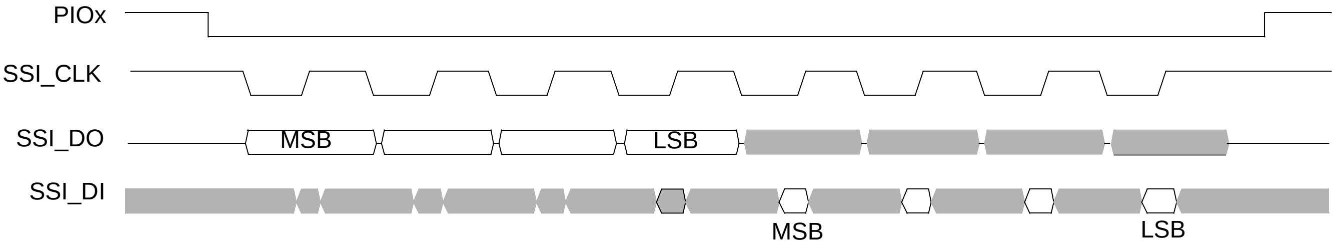


22.5.3.1 4-Bit Read Cycle

A 4-bit operation can be simulated by ignoring four of the eight bits transferred. Figure 22-7 shows an example of a 4-bit read operation.

1. A full-duplex SSI command is executed in non-inverted phase, non-inverted clock, and MSBF modes.
2. The first four bits on SSI_DO transmit a slave nibble read command.
3. The last four bits on SSI_DO can specify a four-bit NOP command, if they are not ignored by the slave.
4. The first four bits on SSI_DI are shifted in, but can be ignored by software.
5. The last four bits on SSI_DI are the requested nibble.
6. The SSI transaction is complete one-half the SSI_CLK period after the last read edge.

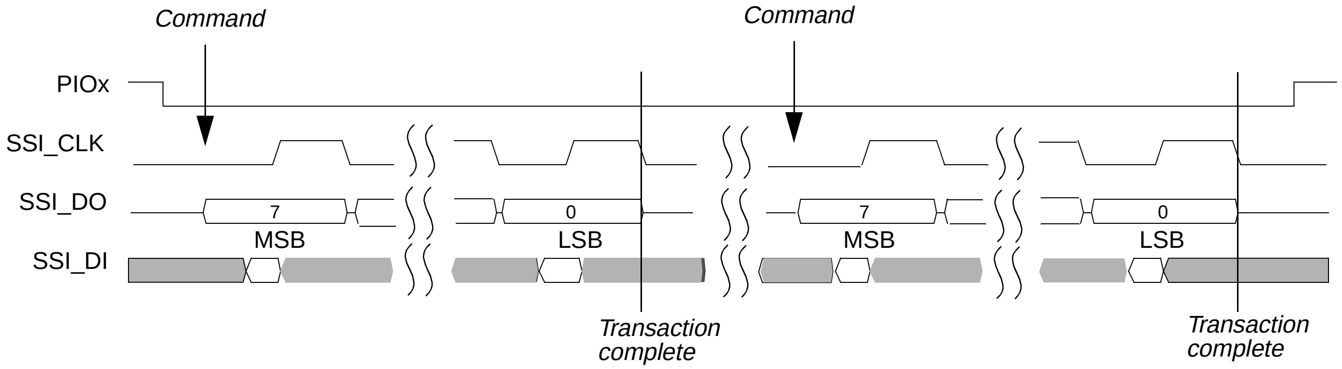
Figure 22-7 SSI 4-Bit Read Cycle: Full-Duplex, Non-Inverted Phase, Non-Inverted Clock



22.5.3.2 Burst, 16-Bit, and 32-Bit Cycles

Burst,16-bit, and 32-bit exchanges can be simulated by multiple 8-bit transactions. There is at least one CPU clock period idle time between transactions. Additional delay between each transaction is determined by software. Figure 22-8 shows an example of a 16-bit operation. Two full-duplex SSI commands are executed to a Microwire-compatible peripheral.

Figure 22-8 SSI Back-to-Back Transactions for Full-duplex, Microwire-Compatible Configuration



22.5.4 Clocking Considerations

The SSI clock is derived from the 33-MHz clock. The CLK_SEL bit in the SSI Control (SSICTL) register (MMCR offset CD0h) is used to configure the frequency of the SSI clock (the SSI_CLK pin). The actual bit rate will vary, depending on whether the system is using a 33.000-MHz or a 33.333-MHz crystal. See the *Élan™SC520 Microcontroller Register Set Manual*, order #22005, for frequency selection.

22.5.5 Interrupts

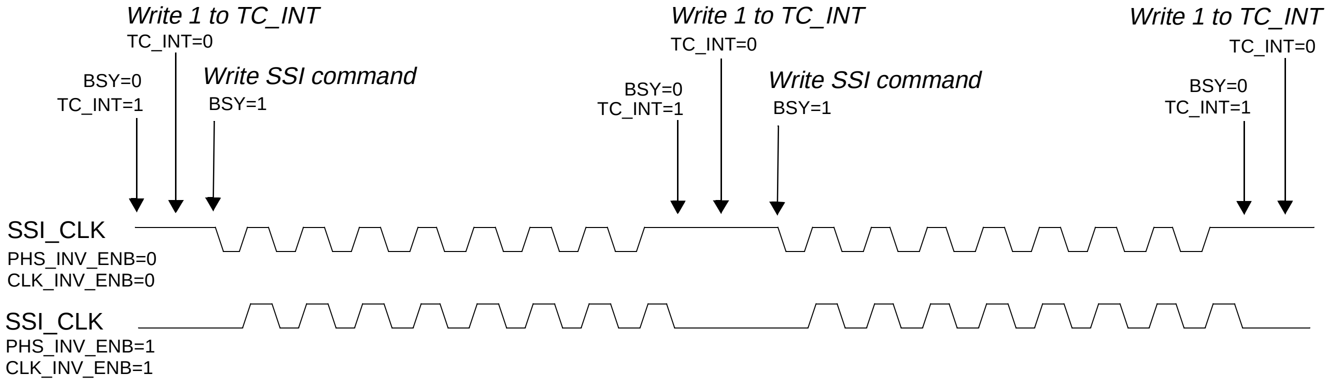
An interrupt can be generated by the SSI to alert the CPU that a transaction is complete.

1. The interrupt is enabled by writing the TC_INT_ENB bit to a 1 in the SSI Control (SSICTL) register.
2. When a transaction is complete, the BSY bit is cleared to a 0 in the SSI Status (SSISTA) register (MMCR offset CD3h), the SSI Transaction Complete Interrupt (TC_INT) bit is set to a 1 in the SSI Status (SSISTA) register, and an interrupt may be sent.
3. Hardware updates the SSI Status (SSISTA) register one-half an SSI clock period after the last receive edge of a transaction (or one full SSI clock period after the last transmit edge of a transaction, indicating that the SSI is again non-busy.
4. A 1 should be written back to the TC_INT bit to clear the bit and acknowledge the interrupt; writing a 0 has no effect.

If the interrupt is not enabled, the SSI Status (SSISTA) register can be polled to periodically read the BSY bit. BSY is set to a 1 when the SSI Command (SSICMD) register is loaded.

The TC_INT and BSY bit values for non-inverted and inverted phase modes are shown in Figure 22-9.

Figure 22-9 SSI Timing: TC_INT and BSY_STA Bits



22.5.6 Software Considerations

A slave should be enabled (if necessary) before a transmit or receive transaction is initiated and disabled after the transaction is complete. Software is responsible for controlling PIOs to implement chip enable signals, including setup and hold time specifications. These pins do not have associated SSI hardware functionality. See Chapter 23, “Programmable Input/Output”, for descriptions of these pins.

Unreliable operation will occur if the configuration is modified or a second SSI command is written during an active operation. Writes to the SSI Transmit (SSIXMIT) register, SSI Control (SSICTL) register, and SSI Command (SSICMD) register should not be performed while the SSI is busy. Software should load the SSI Transmit (SSIXMIT) register (if necessary) before writing an SSI command. The SSI Receive (SSIRCV) register (MMCR offset CD4h) should not be read until a receive transaction is complete.

22.6 INITIALIZATION

The SSI port is disabled during system reset, and all SSI register bits are initialized to 0. The SSI is enabled after reset, but inactive until an SSI command is executed. Some or all of the following steps should be taken to initiate an SSI transaction.

1. Enable/disable CPU transaction complete interrupt via the TC_INT_ENB bit in the SSI Control (SSICTL) register.
2. Enable/disable inverted phase mode via the PHS_INV_ENB bit in the SSI Control (SSICTL) register (MMCR offset CD0h).
3. Enable/disable inverted clock mode via the CLK_INV_ENB bit in the SSI Control (SSICTL) register.
4. Enable/disable MSBF mode via the MSBF_ENB bit in the SSI Control (SSICTL) register.
5. Select SSI clock speed via the CLK_SEL bits in the SSI Control (SSICTL) register.
6. Enable/disable device enable pins using PIOs.
7. Write output data to the SSI Transmit (SSIXMIT) register (MMCR offset CD1h).
8. Write an SSI command to the SSI Command (SSICMD) register (MMCR offset CD2h).
9. Wait for a transaction complete interrupt or poll the SSI Status (SSISTA) register (MMCR offset CD3h) to read BSY bit for port activity status, if the interrupt is disabled.
10. Read input data from the SSI Receive (SSIRCV) register (MMCR offset CD4h).
11. Write a 1 to the TC_INT bit in the SSI Status (SSISTA) register to clear bit and acknowledge the interrupt, if enabled.

CHAPTER

23 PROGRAMMABLE INPUT/OUTPUT

23.1 OVERVIEW

The ÉlanSC520 microcontroller supports 32 programmable I/O signals (PIOs) that can be used on the system board to monitor signals or control devices that are not handled by the other functions in the ÉlanSC520 microcontroller. These signals can be programmed to be inputs or to be driven out High or Low as outputs.

The PIO signals can be programmed for the following functions:

- Read as inputs (default condition after reset)
- Driven High or Low as an output

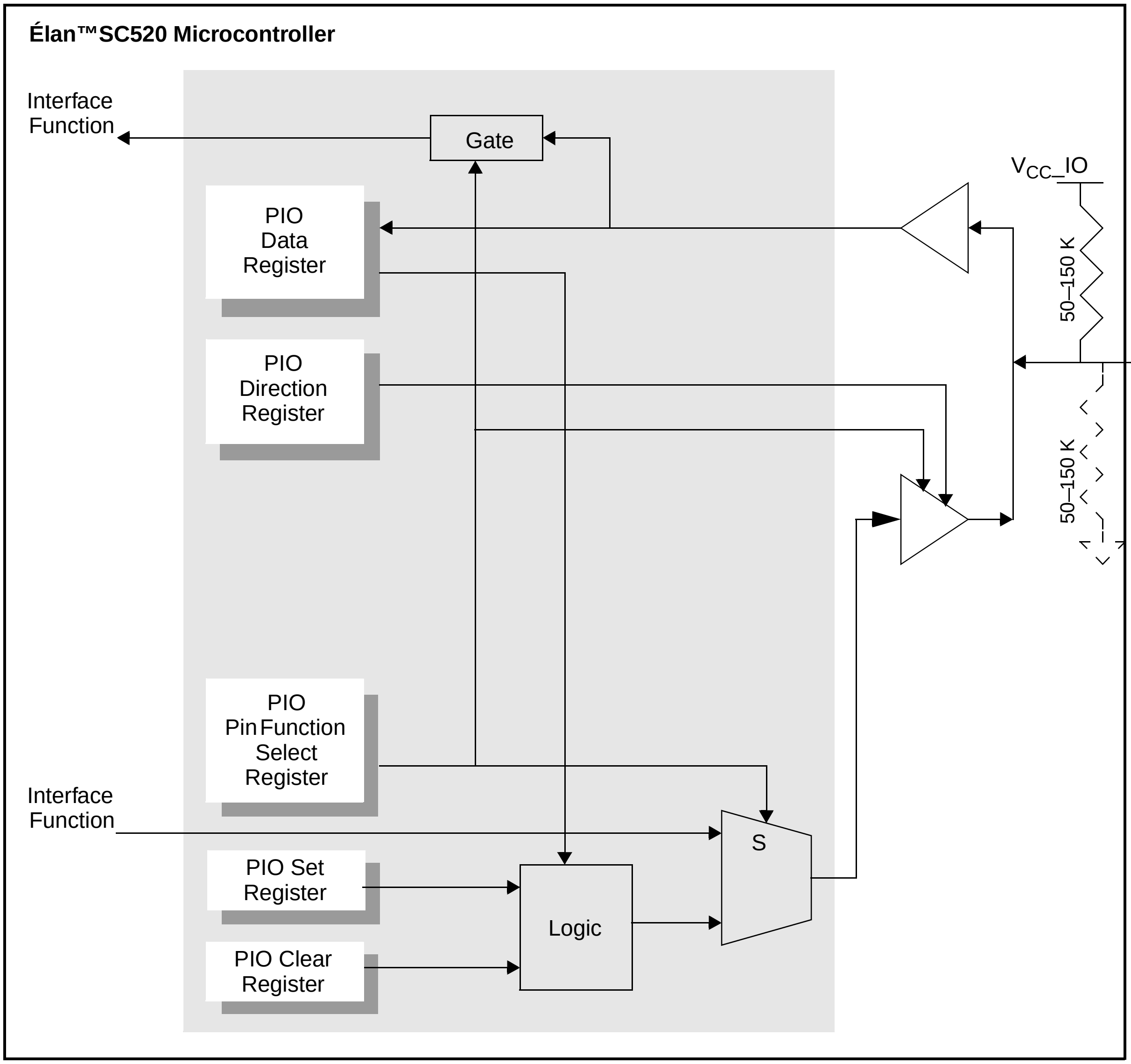
On the ÉlanSC520 microcontroller, all of the PIOs are shared with other functions that may not be needed in every system design, e.g., GP bus signals. This is done to give system designers the most flexibility. For clarity, throughout this document, the two functions available on the PIO pins are distinguished from each other as the *PIO function* and the *interface function*.

Each of the PIO signals is terminated within the ÉlanSC520 microcontroller with either a pullup or pulldown resistance. This feature makes system design easier by eliminating the need for termination on the board. Each PIO signal is terminated according to the pin's interface function, i.e., a normally active Low signal will usually have a pullup to make it inactive on reset. See the *Élan™SC520 Microcontroller Data Sheet*, order #22003, for the termination of each PIO signal.

23.2 BLOCK DIAGRAM

Figure 23-1 is a block diagram of the PIO feature. This structure is repeated for each of the PIOs; only one example PIO is shown in the diagram.

Figure 23-1 PIO Signal Block Diagram



Notes:

A PIO has either a pullup or pulldown resistor, but not both.

23.3 SYSTEM DESIGN

Because most of the PIOs share pins with other functions, designers are usually constrained in choosing which PIO pins to use in their system designs (i.e., they may need the interface function on their board). Choosing between PIOs and interface functions is done on a PIO basis in the two PIOx Pin Function Select registers, as shown in Table 23-1. When enabled, the multiplexed signals shown in Table 23-1 either disable or alter any other function that uses the same pin.

Note: *All PIOs are terminated by either pullup or pulldown resistors (depending on interface function's needs). The pullup and pulldown resistors are approximately 100–150 ohms. The termination of the pin should be considered when deciding which PIO to use. For example, if a PIO that is pulled down by default is to be used for a chip select, the internal pulldown will have to be overridden by a stronger external pullup resistor, or else the external device will have its chip select active at reset.*

After the assertion of PWRGOOD, all PIO signals default to be inputs with pullup or pulldown resistive termination, as shown in Table 23-1. The signals must be programmed before using them as outputs or the alternate interface function. See “Initialization” on page 23-6.

Table 23-1 PIO Signals Shared with Other Interfaces

PIO (Default) Function	Interface Function	Pin Configuration Following System Reset	Control Bit	Register
PIO31	$\overline{\text{RIN2}}$	input with pullup	PIO31_FNC	PIO31–PIO16 Pin Function Select (PIOPFS31_16) register (MMCR offset C22h)
PIO30	$\overline{\text{DCD2}}$	input with pullup	PIO30_FNC	
PIO29	$\overline{\text{DSR2}}$	input with pullup	PIO29_FNC	
PIO28	$\overline{\text{CTS2}}$	input with pullup	PIO28_FNC	
PIO27	$\overline{\text{GPCS0}}$	input with pullup	PIO27_FNC	
PIO26	$\overline{\text{GPMEMCS16}}$	input with pullup	PIO26_FNC	
PIO25	$\overline{\text{GPIOCS16}}$	input with pullup	PIO25_FNC	
PIO24	$\overline{\text{GPDBUFOE}}$	input with pullup	PIO24_FNC	
PIO23	GPIRQ0	input with pullup	PIO23_FNC	
PIO22	GPIRQ1	input with pullup	PIO22_FNC	
PIO21	GPIRQ2	input with pullup	PIO21_FNC	
PIO20	GPIRQ3	input with pullup	PIO20_FNC	
PIO19	GPIRQ4	input with pullup	PIO19_FNC	
PIO18	GPIRQ5	input with pullup	PIO18_FNC	
PIO17	GPIRQ6	input with pullup	PIO17_FNC	
PIO16	GPIRQ7	input with pullup	PIO16_FNC	
PIO15	GPIRQ8	input with pullup	PIO15_FNC	PIO15–PIO0 Pin Function Select (PIOPFS15_0) register (MMCR offset C20h)
PIO14	GPIRQ9	input with pullup	PIO14_FNC	
PIO13	GPIRQ10	input with pullup	PIO13_FNC	
PIO12	$\overline{\text{GPDACK0}}$	input with pullup	PIO12_FNC	
PIO11	$\overline{\text{GPDACK1}}$	input with pullup	PIO11_FNC	
PIO10	$\overline{\text{GPDACK2}}$	input with pullup	PIO10_FNC	
PIO9	$\overline{\text{GPDACK3}}$	input with pullup	PIO9_FNC	
PIO8	GPDRQ0	input with pulldown	PIO8_FNC	
PIO7	GPDRQ1	input with pulldown	PIO7_FNC	
PIO6	GPDRQ2	input with pulldown	PIO6_FNC	
PIO5	GPDRQ3	input with pulldown	PIO5_FNC	
PIO4	GPTC	input with pullup	PIO4_FNC	
PIO3	GPAEN	input with pullup	PIO3_FNC	
PIO2	GPRDY	input with pullup	PIO2_FNC	
PIO1	$\overline{\text{GPBHE}}$	input with pullup	PIO1_FNC	
PIO0	GPALE	input with pullup	PIO0_FNC	

23.4 REGISTERS

A summary listing of the memory-mapped configuration registers used to control the PIO signals is shown in Table 23-2.

Table 23-2 PIO Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
PIO15–PIO0 Pin Function Select	PIOPFS15_0	C20h	PIO15–PIO0 or interface function select: GPIRQ10–GPIRQ8, GPDACK3–GPDACK0, GPDRQ3–GPDRQ3, GPTC, GPAEN, GPRDY, GPBHE, GPALE
PIO31–PIO16 Pin Function Select	PIOPFS31_16	C22h	PIO31–PIO16 or interface function select: RIN2, DCD2, DSR2, CTS2, GPCS0, GPMEMCS16, GPIOCS16, GPDBUFOE, GPIRQ7–GPIRQ0
Chip Select Pin Function Select	CSPFS	C24h	GPCS7–GPCS1 or alternate function select: TMROUTx, TMRINx, PITGATE2, ROMCS2, ROMCS1
Clock Select	CLKSEL	C26h	CLKTIMER[CLKTEST] pin enable, clock output select options (18.432 MHz or 1.8432 MHz UART, PLL1, PLL2, PIT, and RTC), CLKTIMER or CLKTEST select
Drive Strength Control	DSCTL	C28h	I/O pad drive strength for SCS3–SCS0, SRASA–SRASB, SCASA–SCASB, SWEA–SWEB, SDQM3–SDQM0, MA12–MA0, MD31–MD0, MECC6–MECC0.
PIO15–PIO0 Direction	PIODIR15_0	C2Ah	PIO15–PIO0 as input or output
PIO31–PIO16 Direction	PIODIR31_16	C2Ch	PIO31–PIO16 as input or output
PIO15–PIO0 Data	PIODATA15_0	C30h	Read/write directly the state of the PIO15–PIO0 pin
PIO31–PIO16 Data	PIODATA31_16	C32h	Read/write directly the state of the PIO31–PIO16 pin
PIO15–PIO0 Set	PIOSET15_0	C34h	Drive PIO15–PIO0 output High
PIO31–PIO16 Set	PIOSET31_16	C36h	Drive PIO31–PIO16 output High
PIO15–PIO0 Clear	PIOCLR15_0	C38h	Drive PIO15–PIO0 output Low
PIO31–PIO16 Clear	PIOCLR31_16	C3Ah	Drive PIO31–PIO16 output Low

23.5 OPERATION

All PIO signal pins can be programmed as inputs, outputs, or to support their interface function (e.g., GP bus signals). They are enabled as PIO inputs at power-on reset, with built-in pullup or pulldown resistors.

As inputs, PIOs are used by software to monitor signals from other devices. They provide a path to bring signals into the chip that are not available through the other interfaces.

As outputs, the PIOs provide the ability for software to control external devices with signals that can be driven High or Low.

23.5.1 Configuration Information

23.5.1.1 PIO Pins and Simple Input

PIO pins are selected for simple input when the system powers up. The input value of the pins can be read using the PIOx Data registers.

Only two actions disable simple input on the PIO pin:

- Selecting the pin’s interface function
- Setting the PIO’s PIOx_DIR bit in the PIOx Direction register to configure the PIO as an output

23.5.1.2 PIO Pins and Simple Output

If the PIO pin's interface function has not been selected, and the PIOx_DIR is set, the PIO will be an output. The value of the pin can be set by writing to its bit in the PIOx Set and PIOx Clear registers, or by using the appropriate PIOx Data register.

23.5.2 Software Considerations

Table 23-3 summarizes the register settings required to configure the PIOs.

Table 23-3 PIO Configuration Summary

Function Select Register Bit	Direction Register Bit	Data Register Bit (Writes)	Set Register Bit	Clear Register Bit	Data Register Bit (Reads)[1]	Resulting Programmable I/O Pin Function
1	X[2]	X	X	X	?[3]	The pin is not a PIO; it uses its interface function. The value of the pin can be read at the Data bit, but writes to the Direction, Data, Set, and Clear bits have no effect.
0	0	X	X	X	?	The PIO is an input. The state of the pin can be read at the Data bit. Writes to the Data, Set and Clear bits have no effect.
0	1	X	X	1 [4]	0	The PIO is an output. The 1 that is written to the Clear bit causes this PIO pin to be driven Low. The state of the pin can be read at the Data bit, (in this case the pin is Low).
0	1	X	1	X	1	The PIO is an output. The 1 that is written to the Set bit causes this PIO pin to be driven High. The state of the pin can be read at the Data bit, (in this case the pin is High).
0	1	0	X	X	0	The PIO is an output. The 0 that is written to the Data bit causes this PIO pin to be driven Low. The state of the pin can be read at the Data bit, (in this case the pin is Low).
0	1	1	X	X	1	The PIO is an output. The 1 that is written to the Data bit causes this PIO pin to be driven High. The state of the pin can be read at the Data bit, (in this case the pin is High).

Notes:

1. The Data Register Bit (Reads) column shows the resulting state of the Data register bit and the corresponding PIO pin.

2. X = Not used in this operation.

3. ? = Input value. (The Data register bit state always reflects the corresponding pin state, whether input or output.)

4. For a particular PIO output operation, only one of the pin’s Data, Set, or Clear bits can be used. The state of the unused bits is not important, but subsequent writes to these bits can change the PIO pin state.

Note that although the registers to set, clear, and read the PIO pins can be accessed with 32-bit instructions, 32-bit accesses are split into two 16-bit accesses. This means, for example, that it is impossible to simultaneously set PIO5 and PIO18. Similarly, it is impossible to sample the state of PIO12 and PIO23 simultaneously; the 32-bit value returned by the instruction contains two 16-bit values sampled at different times. For 32-bit operations, the lower 16-bit word (for PIO31–PIO16) is always accessed before the upper 16-bit word (for PIO15–PIO0). The time between the two accesses is indeterminate and based on other masters besides the CPU trying to access the bus.

23.6 INITIALIZATION

After a system reset, all of the PIO31–PIO0 signals default to be inputs with pullup or pulldown resistive termination. The signals must be programmed before using them as outputs or the alternate interface function.

To initialize the PIOx signals, the following steps are required:

1. Based on the specific application, determine which ÉlanSC520 microcontroller pins can utilize the PIO function and which should be programmed as the interface function.
2. Program the PIOx Pin Function Select registers to select between the PIO function and the interface function of each of the PIO31–PIO0 pins.
3. For pins specified as using the PIO functionality, define the PIO direction by programming the PIOx Direction registers.
4. PIO pins that are defined as inputs can now be read via the PIOx Data registers.
5. PIO pins defined as outputs can now be written via the PIOx Data, PIOx Set, or PIOx Clear registers.

CHAPTER

24 SYSTEM TEST AND DEBUGGING

24.1 OVERVIEW

This chapter describes various system-level test features included in the ÉlanSC520 microcontroller. These features are useful for debugging hardware and software in an ÉlanSC520 microcontroller-based system. Some of the system-level debugging features are useful in conjunction with the AMDebug interface for software debugging. This functionality is described in Chapter 26, “AMDebug™ Technology”.

The list below summarizes the functionality that has been included in the ÉlanSC520 microcontroller to facilitate system-level debugging.

- A simple three-pin interface to aid in-circuit emulation tools with tracing external bus activity
- A write buffer test mode to assist in determining which bus masters contributed to the current active write buffer write cycle on the SDRAM interface
- A nonconcurrent arbitration mode that reduces the complexity of system transactions when the Am5$_X$86 CPU or PCI bus masters or GP-DMA cycles occur simultaneously
- Echoing internal cycles and read data on the GP bus during Am5$_X$86 CPU accesses of internal integrated peripherals
- Disabling the Am5$_X$86 CPU’s integrated cache controller, controlling the cache write policy, and specifying noncacheable memory regions
- Controlling the clock speed of the Am5$_X$86 CPU’s internal core
- Disabling the SDRAM read buffer and write buffer
- Ability to interrupt the Am5$_X$86 CPU when an illegal memory write occurs to a write-protected memory region, or to cause an exception when a code fetch occurs from data memory
- Ability to identify the source of a reset event
- Ability to trace Error Correcting Code (ECC) errors for testing
- Ability to override the ECC syndrome code

24.2 SYSTEM DESIGN

As shown in Table 24-1, three debugging pins on the ÉlanSC520 microcontroller operate as either $\overline{\text{CF\_DRAM}}$, DATASTRB, and $\overline{\text{CF\_ROM\_GPCS}}$, or WBMSTR2–WBMSTR0, depending on if the ÉlanSC520 microcontroller has been configured for system test mode (default) or write buffer test mode.

The CFG2–CFG0 pinstrap functions associated with these three pins are sampled only as a result of PWRGOOD assertion and do not affect the other functions of these pins, so they are not shown in this table. When enabled, the multiplexed signals shown in Table 24-1 either disable or alter any other function that uses the same pin.

Table 24-1 System Test and Debugging Signals Shared with Other Interfaces

Default Signal	Alternate Function	Control Bit	Register
CF_ROM_GPCS	WBMSTR0	WB_TST_ENB	SDRAM Control (DRCCTL) register (MMCR offset 10h)
DATASTRB	WBMSTR1		
CF_DRAM	WBMSTR2		

24.2.1 Loading

When a logic analyzer is connected to the ÉlanSC520 microcontroller pins, it presents an additional load that must be taken into consideration on critical buses, such as the SDRAM interface. Extreme care must be taken when connecting to either the SDRAM clock or the PCI bus clock. When external clock drivers are used on the system circuit board, it may be best to connect to the output of a lightly loaded or unused clock driver.

24.3 REGISTERS

Table 24-2 lists the memory-mapped registers that are used to control the system-level debugging features.

Table 24-2 System Test and Debugging Registers—Memory-Mapped

Register	Mnemonic	MMCR Offset Address	Function
Am5$_x$86 CPU Control	CPUCTL	02h	CPU cache mode select (write-through or write-back), CPU clock speed
SDRAM Control	DRCCTL	10h	System test mode (CF_DRAM, DATASTRB, and CF_ROM_GPCS), write buffer test mode (WBMSTR2–WBMSTR0) enable
ECC Check Code Test	ECCCKTEST	23h	ECC check code override for test and error handler development
ECC Single-Bit Error Address	ECCSBAD	24h	Physical address of the location in SDRAM that caused a single-bit ECC error
ECC Multi-Bit Error Address	ECCMBADD	28h	Physical address of the location in SDRAM that caused a multi-bit ECC error
SDRAM Buffer Control	DBCTL	40h	Write buffer functions: write buffer enable, read-ahead enable, write buffer watermark, write buffer flush.
System Arbiter Control	SYSARBCTL	70h	System arbitration concurrency mode enable
Address Decode Control	ADDDECCTL	80h	Write-protect violation interrupt enable
Programmable Address Region x	PAR0–PAR15	88–C4h	Set noncacheable, write-protected, and non-executable memory regions
GP Echo Mode	GPECHO	C00h	Echo mode enable for monitoring integrated peripheral accesses on GP bus
Reset Configuration	RESCFG	D72h	AMDebug mode enable

Table 24-2 System Test and Debugging Registers—Memory-Mapped (Continued)

Register	Mnemonic	MMCR Offset Address	Function
Reset Status	RESSTA	D74h	Reset source status: SCP reset, AMDebug hard reset detect, AMDebug system reset, watchdog timer time-out, CPU shutdown (soft reset), PRGRESET pin, and PWRGOOD pin

24.4 OPERATION

The ÉlanSC520 microcontroller provides several features that are useful in a lab environment for system-level debugging of both hardware and software. These features can be used in conjunction with an in-circuit emulation system, but can also be used independently to simplify some debugging activities. Many features are expected to be used with a logic analyzer to capture system transaction information. These distinct system-level debugging features are described in the separate sections of this chapter.

The three-pin debugging interface is a particularly useful feature of the ÉlanSC520 microcontroller. This interface operates in two different modes:

- System test mode
- Write buffer test mode

24.4.1 System Test Mode

System test mode is the primary use of the three-pin interface, which enables the pins to be monitored with a logic analyzer or external in-circuit emulation system hardware to gain important knowledge of current Am5$_x$86 CPU cycles.

System test mode is used primarily to differentiate Am5$_x$86 CPU code fetches from normal memory read cycles on the SDRAM and ROM/Flash or GP bus interface. A signal (DATASTRB) is also provided to identify when the data on the SDRAM data bus is valid. This signal is used primarily by in-circuit emulation tools for capturing SDRAM data when monitoring this interface.

System test mode is enabled by clearing the WB_TST_ENB bit in the SDRAM Control (DRCCTL) register (MMCR offset 10h). System test mode is the default test mode on the ÉlanSC520 microcontroller. The multiplexed debugging signals then operate as described in Section 24.4.1.1.

24.4.1.1 Pin Functions in System Test Mode

24.4.1.1.1 $\overline{\text{CF\_DRAM}}$

During SDRAM read cycles, the $\overline{\text{CF\_DRAM}}$ signal provides code fetch status.

- When Low, if DATASTRB is active in the current cycle, this signal indicates that the current SDRAM read is a CPU code fetch demanded by the CPU, or a read prefetch initiated due to a demand code fetch by the CPU.
- When High, this signal indicates that the SDRAM read is not a code fetch, and it could have been initiated by the CPU, PCI master, or the GP-DMA controller, either demand or prefetch.

During SDRAM write cycles, the $\overline{\text{CF\_DRAM}}$ signal provides an indication of the source of the data, either GP-DMA controller/PCI bus master, or CPU.

- When High, this signal indicates that either a GP-DMA initiator or an external PCI bus master contributed to the current SDRAM write cycle (the CPU may also have contributed).
- A Low indicates that the CPU is the only master that contributed to this write cycle.

24.4.1.1.2 DATASTRB

The DATASTRB signal is useful for the external in-circuit emulation system to latch data from the SDRAM interface, regardless of the programmed SDRAM timing.

- When Low, data on the SDRAM data bus is invalid.
- When High, data on the SDRAM data bus can be latched on the next rising edge of the CLKMEMIN signal.

24.4.1.1.3 CF_ROM_GPCS

The CF_ROM_GPCS signal can be sampled on the Low-to-High transition of the ROMRD signal during ROM/Flash cycles or during the Low-to-High transition of GPMEMRD for GPCS7–GPCS0 cycles.

- The CF_ROM_GPCD signal should be sampled only when either GPMEMRD or ROMRD is asserted.
- When Low under these conditions, this signal indicates that the CPU is performing a code fetch from ROM (on either the GP bus or SDRAM interface) or a GP bus memory device.

24.4.1.2 Using the System Test Mode Interface

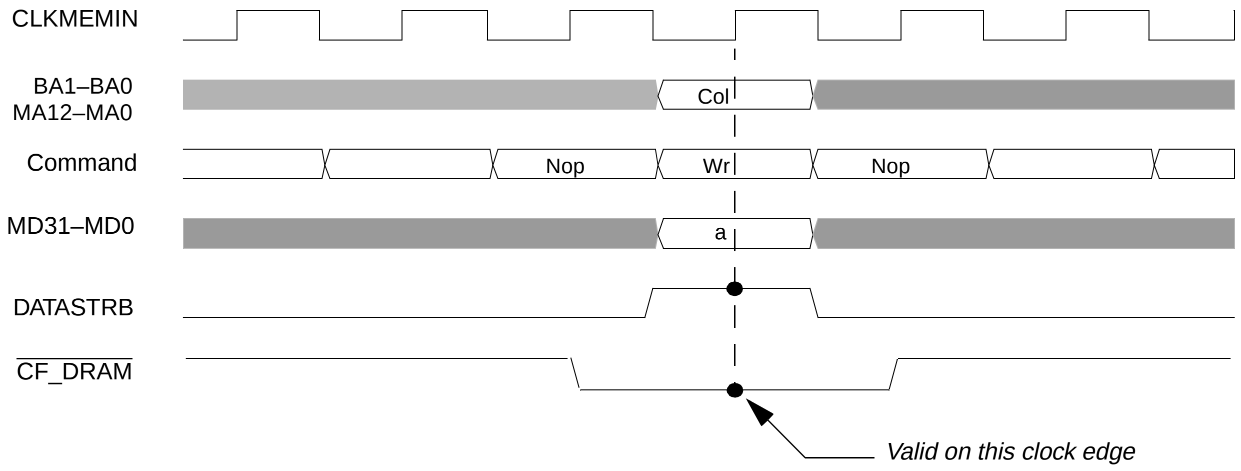
The system test mode interface is useful for tracing Am5$_X$86 CPU activity on the SDRAM and GP bus interfaces, including when the Am5$_X$86 CPU is the initiator, when the data is valid during SDRAM read and write cycles, and differentiating between code fetches and data accesses. This still requires demultiplexing the BA1–BA0 and MA12–MA0 SDRAM address bus to construct a full 28-bit address, which also requires knowledge of the programming of some of the SDRAM controller configuration registers for device size and symmetry. Since a data strobe is provided on the WBMSTR1 pin in this mode, detailed knowledge of the programming of the SDRAM timing is not required. See Chapter 10, “SDRAM Controller”, for details of SDRAM cycle timing and address multiplexing.

The CF_DRAM and CF_ROM_GPCS signals enable external determination of code fetches from SDRAM, ROM/Flash, or any GP bus memory device. Prefetches from the SDRAM controller’s read buffer can also be identified.

24.4.1.3 SDRAM Write Cycle in System Test Mode

Figure 24-1 illustrates the timing of a page hit SDRAM write cycle during system test mode. To capture the CF_DRAM, BA1–BA0, MA12–MA0, and MD31–MD0 signals, the logic analyzer or external in-circuit emulation system can use the DATASTRB signal to identify the appropriate time to latch the information. This information must be captured on the rising edge of CLKMEMIN when DATASTRB is sampled active. Note that DATASTRB is not asserted during the read portion of a read-modify-write cycle that occurs for sub-doubleword writes with ECC enabled.

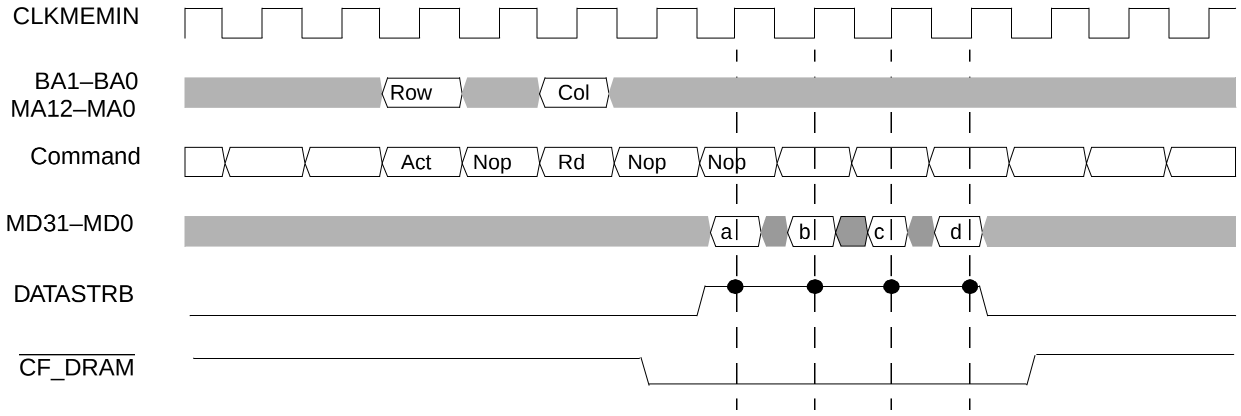
Figure 24-1 System Test Mode Timing During a SDRAM Write Cycle (Page Hit)



24.4.1.4 SDRAM Read Cycle in System Test Mode

Figure 24-2 illustrates the timing of a page miss SDRAM read cycle (with a $\overline{\text{CAS}}$ Latency of 2) during system test mode. To capture the $\overline{\text{CF\_DRAM}}$, BA1–BA0, MA12–MA0, and MD31–MD0 signals, the logic analyzer or external in-circuit emulation system can use the DATASTRB signal to identify the appropriate time to latch the information. This information must be captured on the rising edge of CLKMEMIN when DATASTRB is sampled active. The $\overline{\text{CAS}}$ latency timing is configured in the SDRAM Timing Control (DRCTMCTL) register (MMCR offset 12h). The BA1–BA0 and MA12–MA0 bus can be used to determine the physical address generated by the requesting master.

Figure 24-2 System Test Mode Timing During an SDRAM Read Cycle (Page Miss)



Notes:
$\overline{\text{CAS}}$ latency is 2.

24.4.1.5 Tracing Transactions on the ROM Interface

Tracing transactions on the ROM interface requires only the $\overline{\text{CF\_ROM\_GPCS}}$ signal if it is desired to differentiate code fetches from memory read cycles. Only the Am5$_x$86 CPU can be the initiator of ROM accesses. The address bus is non-multiplexed, and thus can be read directly from the GPA25–GPA0 pins during ROM/Flash cycles. The system configuration of the ROM array must be known, because the ROM data bus can be connected to either the SDRAM interface data pins (MD31–MD0), or the GP bus interface data pins (GPD15–GP0). Also, the timing of the ROM cycle will vary, depending on the device that has been connected to each of the ROM chip selects and the programming of the ROM controller configuration registers. The following pins can be monitored to trace transactions on the ROM interface:

- CF_ROM_GPCS if it is necessary to identify code fetches
- GPA25–GPA0 ROM non-multiplexed address bus
- GPD15–GPD0 ROM data bus, or MD31–MD0 SDRAM data bus, depending on the programming of the ROM controller configuration registers
- ROM chip selects BOOTCS, and optionally ROMCS1 and ROMCS2
- ROMRD, FLASHWR control signals

See Chapter 12, “ROM/Flash Controller”, for further details of ROM interface signals and timing to determine the appropriate time when the address and data pins are valid.

24.4.1.6 Tracing Transactions on the GP Bus Interface

Capturing transactions on the GP bus interface requires only the CF_ROM_GPCS signal if it is desired to differentiate code fetches from memory read cycles. However, some further signal qualification is required to filter out GP-DMA transactions from Am5$_X$86 CPU cycles. PCI bus masters are not permitted to initiate cycles on the GP bus. The signals required to trace cycles on the GP bus will vary depending on the type of slave devices connected externally.

Note that due to performance limitations of the GP bus, it is highly recommended that code execution from this bus be avoided.

The GPAEN signal must be monitored by the GP bus devices when GP-DMA initiators are connected on the GP bus to prevent address decoding during GP-DMA cycles. GP bus control signals asserted when the GPAEN signal is active (High) are controlling a read or write of a GP-DMA initiator, and the address on the GPA25–GPA0 pins are invalid. GPAEN is also driven active during internally echoed cycles to prevent address decoding by GP bus devices.

Since the GP bus supports several different cycle types, dynamic bus sizing, and timing control, there are numerous signals that may be required for adequate tracing of GP bus transactions. The following list summarizes the various signals that should be considered for such tracing.

- CF_ROM_GPCS if it is necessary to identify code fetches
- GPA25–GPA0 non-multiplexed address bus
- GPD7–GPD0 data bus for 8-bit cycles, or GPD15–GPD0 for 16-bit cycles
- GP bus chip selects, multiplexed on ROMCS1 or ROMCS2, or PIO pins
- GPALE, GPIORD/GPMEMRD, GPIOWR/GPMEMWR, GPAEN control signals
- GPRDY signal for devices that dynamically stretch GP bus cycles
- GPIOCS16 and GPMEMCS16 for devices that dynamically identify the bus width of the target device’s cycle

See Chapter 13, “General-Purpose Bus Controller”, for details of cycle timing.

24.4.2 Write Buffer Test Mode

Write buffer test mode identifies which bus owners (Am5$_X$86 CPU, PCI bus master, or GP-DMA controller) have contributed to the current SDRAM write cycle, and which bus owner is requesting the current SDRAM read cycle.

The ÉlanSC520 microcontroller implements a 32-rank First-In-First-Out (FIFO) write buffer for improved memory performance. The write buffer also supports write merging and write collapsing. Therefore, each of the 32-bit ranks and each byte within the rank can be written by either the Am5$_X$86 CPU, PCI bus masters, or the GP-DMA controller. For example, byte 0 and byte 1 of a write buffer rank can be written by the Am5$_X$86 CPU, byte 2 of the same rank can be written by a PCI bus master, and byte 3 of the same rank can be written by the GP-DMA controller.

Although this will result in improved performance of the SDRAM subsystem, it can be confusing when attempting system debugging with a logic analyzer, because it is impossible to identify the source of SDRAM write cycles from the normal SDRAM interface alone. (For more information on the write buffer, see Chapter 11, “Write Buffer and Read Buffer”.)

When write buffer test mode is enabled via the WB_TST_ENB bit in the SDRAM Control (DRCCTL) register (MMCR offset 10h), the WBMSTR2–WBMSTR0 pins indicate whether the Am5$_X$86 CPU, PCI bus master, GP-DMA controller, or a combination of these has written into a particular rank of the write buffer.

24.4.2.1 Using the Write Buffer Test Mode Interface

Sampling the WBMSTR2–WBMSTR0 pins for write buffer debugging requires external decoding of the SDRAM interface signals to determine when write cycles are occurring on the SDRAM interface. To provide useful information about the cycle, the BA1–BA0 and MA12–MA0 SDRAM address bus must be demultiplexed to provide the full 28-bit memory address, and the $\overline{\text{SRASx}}$, $\overline{\text{SCASx}}$, and $\overline{\text{SWEx}}$ command signals must be sampled to differentiate reads, writes, refresh cycles, etc.

Figure 24-3 shows WBMSTR2–WBMSTR0 timing during a SDRAM write cycle. The trace information is available one clock before the clock edge where the command is driven to the SDRAM. This guarantees sufficient setup so the trace information can be captured on the clock edge where the SDRAM command is sampled. It is the responsibility of the monitoring equipment to capture the WBMSTR2–WBMSTR0 trace signals information at the appropriate time and cycle type. This can be accomplished by monitoring the SDRAM interface pins and decoding the SDRAM cycle type for the programmed SDRAM timing. See Chapter 10, “SDRAM Controller”, for details on the ÉlanSC520 microcontroller’s address multiplexing scheme and SDRAM timing and signaling.

Determining when the data is valid during SDRAM read cycles requires knowledge of the SDRAM timing configuration, such as $\overline{\text{CAS}}$ latency, etc. See “SDRAM Read Cycle in Write Buffer Test Mode” on page 24-8. For writes, the data is available at the time of the write.

24.4.2.2 SDRAM Write Cycle in Write Buffer Test Mode

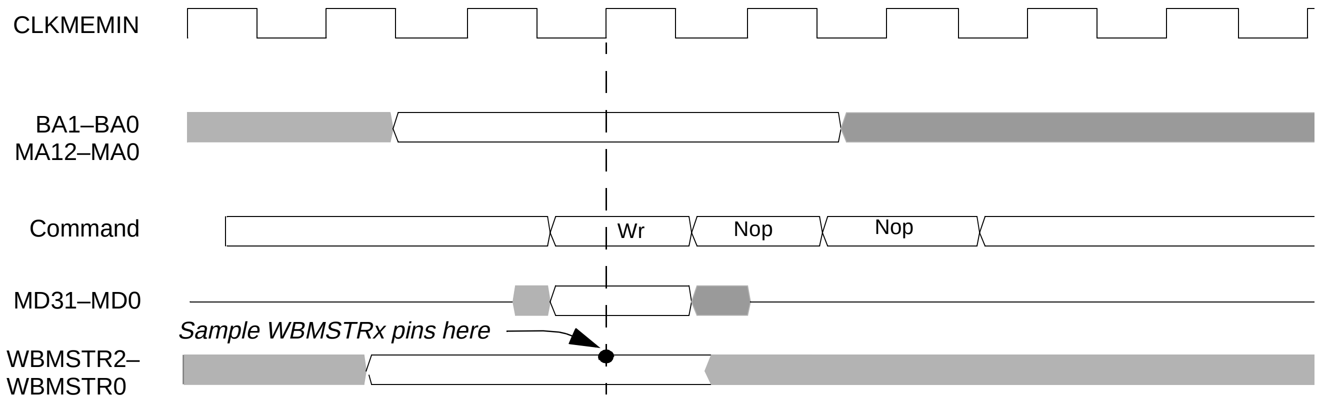
Table 24-3 describes the WBMSTR2–WBMSTR0 decoding during an SDRAM write operation.

Table 24-3 WBMSTR2–WBMSTR0 Pin Definition During Write Buffer Write Cycles

WBMSTR2–WBMSTR0 Pins			
Am5$_x$86 CPU	PCI Bus Master	GP-DMA Controller	Description
0	0	0	Reserved
0	0	1	GP-DMA contributed write data
0	1	0	PCI master contributed write data
0	1	1	PCI master and GP-DMA contributed write data
1	0	0	Am5$_x$86 CPU contributed write data
1	0	1	Am5$_x$86 CPU and GP-DMA contributed write data
1	1	0	Am5$_x$86 CPU and PCI master contributed write data
1	1	1	All masters contributed write data

Figure 24-3 illustrates the timing of an example of a page hit SDRAM write cycle during write buffer test mode. To capture the WBMSTR2–WBMSTR0 pins, the logic analyzer or external in-circuit emulation system must decode the SDRAM command and latch the WBMSTR2–WBMSTR0 pin on the rising edge of CLKMEMIN.

Figure 24-3 Write Buffer Test Mode Timing During an SDRAM Write Cycle (Page Hit)



24.4.2.3 SDRAM Read Cycle in Write Buffer Test Mode

During read operations, the WBMSTR2–WBMSTR0 pins can be used to determine which master is performing the current SDRAM read cycle. Although more than one of these sources may have written to a given rank in the write buffer, only one initiator can read a rank at any given time.

Table 24-4 describes the WBMSTR2–WBMSTR0 pins during a SDRAM read operation in write buffer test mode. Note that SDRAM read cycles can occur with more than one of the WBMSTR2–WBMST0 signals active during the read portion of a read-modify-write cycle.

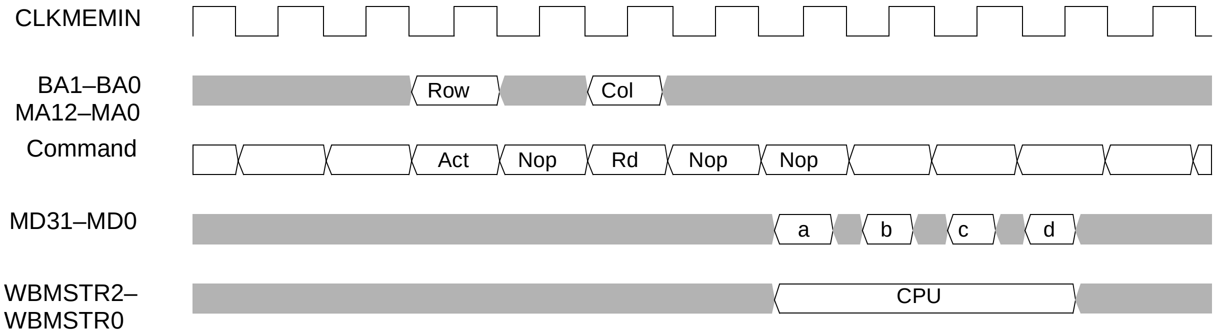
In this case, the WBMSTR2–WBMSTR0 pins represent which bus initiators contributed to the rank of the write buffer that is being written to SDRAM.

Table 24-4 WBMSTR2–WBMSTR0 Pin Definition During SDRAM Read Cycles

WBMSTR2–WBMSTR0 Pins			
Am5$_x$86 CPU	PCI Bus Master	GP-DMA Controller	Description
0	0	0	Read prefetch cycle (No master requested read cycle)
0	0	1	GP-DMA is current read master
0	1	0	PCI master is current read master
0	1	1	Reserved
1	0	0	Am5$_x$86 CPU is current read master
1	0	1	Reserved
1	1	0	Reserved
1	1	1	Reserved

Figure 24-4 illustrates the timing of a page miss SDRAM read cycle (with a $\overline{\text{CAS}}$ latency of 2) during write buffer test mode. To capture the WBMSTR2–WBMSTR0 pins during a read cycle, the logic analyzer or external in-circuit emulation system must decode the SDRAM read command and delay latching the WBMSTR2–WBMSTR0 pins until the appropriate $\overline{\text{CAS}}$ latency timing is met. WBMSTR2–WBMSTR0 are captured on the rising edge of CLKMEMIN. The $\overline{\text{CAS}}$ latency timing is configured in the SDRAM Timing Control (DRCTMCTL) register (MMCR offset 12h). The MA12–MA0 and BA1–BA0 signals can be used to determine the physical address generated by the requesting master.

Figure 24-4 Write Buffer Test Mode Timing During a SDRAM Read Cycle (Page Miss)



24.4.3 Other Debugging Features on the Élan™SC520 Microcontroller

24.4.3.1 Nonconcurrent Arbitration Mode

The ÉlanSC520 microcontroller's system arbitration is comprised of an Am5$_X$86 CPU bus arbiter and a PCI bus arbiter, which enables concurrent mode operation. In the concurrent arbitration mode, transactions on the Am5$_X$86 CPU bus and the PCI bus can occur simultaneously. For example, a peer-to-peer PCI bus transaction can occur simultaneously with an Am5$_X$86 CPU transaction. The advantage of this mode is the optimal utilization of the two buses. However, this can be confusing when attempting system debugging, because it is difficult to trace bus activity with concurrency. Also, some system bugs can be traced back to improper configuration during concurrent arbitration mode while both the Am5$_X$86 CPU and external PCI bus masters are active. This occurs, for example, when the Am5$_X$86 CPU is modifying configuration registers such as address decode registers that affect PCI bus master operation. In this case, using nonconcurrent arbitration mode instead can assist in tracing these problems.

At system initialization, the ÉlanSC520 microcontroller boots up in the nonconcurrent arbitration mode until the CNCR_MODE_ENB bit in the System Arbiter Control (SYSARBCTL) register (MMCR offset 70h) is set. For debugging purposes, it can be useful to omit this step and remain in nonconcurrent arbitration mode. For more details, see Chapter 8, "System Arbitration".

24.4.3.2 Echoing Integrated Peripheral Accesses on the GP Bus

All accesses from the Am5$_X$86 CPU to the ÉlanSC520 microcontroller's integrated peripherals are not externally visible, but can optionally be directly monitored on the GP bus using GP bus echo mode. If required, a logic analyzer can be connected to the GP bus to monitor and debug the transactions. When the GP_ECHO_ENB bit is set in the GP Echo Mode (GPECHO) register (MMCR offset C00h), accesses to the GP-DMA controller, RTC, internal timers, PIC, UARTs, and PIOs are echoed externally on the GP bus. During reads, the data from the peripheral is also driven on the GP bus data lines, GPD15–GPD0.

24.4.3.3 Summary of Additional System Debugging Features

There are additional features in the ÉlanSC520 microcontroller that are not included specifically for system debugging but can be useful during the debugging phase. These features are described in other chapters, but are summarized below for reference.

- The ÉlanSC520 microcontroller provides the ability to control the Am5$_X$86 CPU's cache write policy with the Am5$_X$86 CPU Control (CPUCTL) register (MMCR offset 02h) and to disable the cache using the CPU's machine status (CR0) register. This can be useful in debugging some system problems when cache coherency is a problem or when visibility of all Am5$_X$86 CPU memory cycles are required externally. See Chapter 7, "Am5x86® CPU", for details on cache control.
- The ÉlanSC520 microcontroller provides the ability to dynamically control the Am5$_X$86 CPU's internal clock speed in the Am5$_X$86 CPU Control (CPUCTL) register. This is primarily to allow thermal management, but there may be some cases when it is useful to adjust the clock speed for debugging purposes. See Chapter 7, "Am5x86® CPU", for details on clock speed control.
- The SDRAM controller's write buffer and read buffer can be disabled by resetting the WB_ENB bit in the SDRAM Buffer Control (DBCTL) register (MMCR offset 40h). This can be useful during system debugging, because it prevents queued SDRAM writes and prefetching on the SDRAM interface that can make it difficult to trace bus activity. See Chapter 11, "Write Buffer and Read Buffer", for details on disabling these features.

- The ÉlanSC520 microcontroller’s address decode logic allows notification of violations of write-protected memory regions, which is useful when debugging a software task that is illegally attempting to modify a portion of memory modified as write-protected. See Chapter 4, “System Address Mapping”, for further details on enabling this feature.
- The ÉlanSC520 microcontroller’s address decode logic also allows notification of violations of memory regions marked as non-executable address space. This is useful when debugging a software task that is attempting to execute code from a portion of memory designated for data only. See Chapter 4, “System Address Mapping”, for further details on enabling this feature.
- If the ICE_ON_RST bit is set in the Reset Configuration (RESCFG) register (MMCR offset D72h), the Am5$_X$86 CPU enters AMDebug mode whenever it is reset (immediately after the reset sequence). The debugging tool can read the Reset Status (RESSTA) register (MMCR offset D74h) to identify the source of the reset.
- The programmable interrupt controller (PIC) supports many features, such as the ability to mask specific interrupts and to force software interrupts, which can also be useful during the system debugging phase. See Chapter 15, “Programmable Interrupt Controller”, for details on configuring interrupts in a system.
- To assist in the development of software to handle ECC single-bit and multi-bit errors, the ECC Check Code Test (ECCCKTEST) register (MMCR offset 23h) is provided. This register can be used to override the automatically-generated ECC check code with a user-provided check code for the following SDRAM write access.

24.4.4 Software Considerations

The cache should always be flushed after the cacheability attribute for an address range is changed from cacheable to noncacheable for any memory region (by programming a PAR register), or when the cache write policy is changed from write-back to write-through.

Software must include proper interrupt service routines and exception handlers when enabling write-protection violation interrupts and non-executable region attributes in the Address Decode Control (ADDDECCTL) register (MMCR offset 80h). Note that in the case of the write-protect violation, the address of the violation is latched in a 32-bit register and retained until the register is cleared by software; any additional violations that occur before the register is read will not be seen.

A write-protection violation occurs when the Am5$_X$86 CPU, any PCI bus master, or the GP-DMA controller attempt to write to any memory region that has been marked as write-protected by a PAR register attribute. When this occurs, the cycle is always forwarded to SDRAM as a write-protected cycle (the SDQM3–SDQM0 pins are forced inactive), and the original data is discarded.

24.4.5 Latency

Some features described in this chapter to aid the debugging process may affect system performance, and these effects should therefore be considered when enabling or disabling. A brief list of the features and their direct affects on latency are listed.

- Write buffer and system test modes do not affect performance, unless the SDRAM timing has been programmed at slower speeds to accommodate external capturing of data.
- Nonconcurrent arbitration affects PCI bus latency, the Am5$_X$86 CPU’s latency, and the GP-DMA controller’s latency, since ownership of both buses must be negotiated before any transaction is allowed to begin. The effect in a system with no PCI bus masters or GP-DMA initiators is much less, because bus acquisition is immediate.

- The Am5$_x$86 CPU's internal cache can greatly affect system performance.
  - When disabled, all Am5$_x$86 CPU operations require an external bus cycle, which yields significantly less bus bandwidth for PCI bus masters and GP-DMA initiators.
  - When configured in write-through cache mode, all Am5$_x$86 CPU write cycles are forwarded to the Am5$_x$86 CPU bus, whereas in write-back cache mode, they are only forwarded out of the Am5$_x$86 CPU when a cache miss or write-back/copy-back cycle occurs. Although write-through cache mode takes much less of the bandwidth away from PCI bus masters and GP-DMA initiators, it is significantly more than when the cache is operating in write-back mode.
  - When areas of memory are marked as noncacheable in the PAR registers, the overhead of cache write-backs is reduced, yielding lower latency for all system bus owners.
- The internal Am5$_x$86 CPU core clock speed affects overall Am5$_x$86 CPU performance when the Am5$_x$86 CPU is able to execute from its internal cache. When the cache is disabled, the effect of a higher core speed is much less, because all operations require an external bus cycle at the fixed bus speed of 33 MHz.
- Disabling the write buffer and read buffer may significantly affect performance, depending on the ordering of reads and writes, and the number of PCI bus masters and the amount of GP-DMA activity in the system. It is difficult to predict the exact effect of these buffers on each system, because there are many dependencies. However, it should be noted that, in some cases, a notable change in system performance will occur. This also complicates the system debugging process, because the system bus activity profile may be much different in the two cases.
- Enabling interrupts for write-protect violation notification (as with all maskable interrupts), causes a context switch to occur, which naturally imposes a reload of the Interrupt Descriptor Table and saving the current state of the Am5$_x$86 CPU before servicing the interrupt. This should not be a long-term problem, because it is expected that the write violation protection would occur only during the initial debugging phases of system development.
- When GP bus echoing is enabled, the access times of the integrated peripherals is subject to the timing programmed for the external GP bus.

24.5 INITIALIZATION

The state of the ÉlanSC520 microcontroller debugging features after system reset is:

- The WBMSTR2–WBMSTR0 pins default to system test mode, in which they assume the function of $\overline{\text{CF\_DRAM}}$, DATASTRB, and $\overline{\text{CF\_ROM\_GPCS}}$ pins respectively.
- The system arbitration defaults to nonconcurrent arbitration mode operation.
- Echoing of integrated peripheral accesses is disabled.
- The Am5$_x$86 CPU's cache is disabled and configured for write-back cache mode.
- The Am5$_x$86 CPU default clock speed is 100 MHz.
- The write buffer and the read buffer are disabled.
- The write-protection violation interrupt is disabled, and the Programmable Address Region (PAR) registers are cleared; thus, no write-protect or non-executable memory regions are defined.

CHAPTER

25 BOUNDARY SCAN TEST INTERFACE

25.1 OVERVIEW

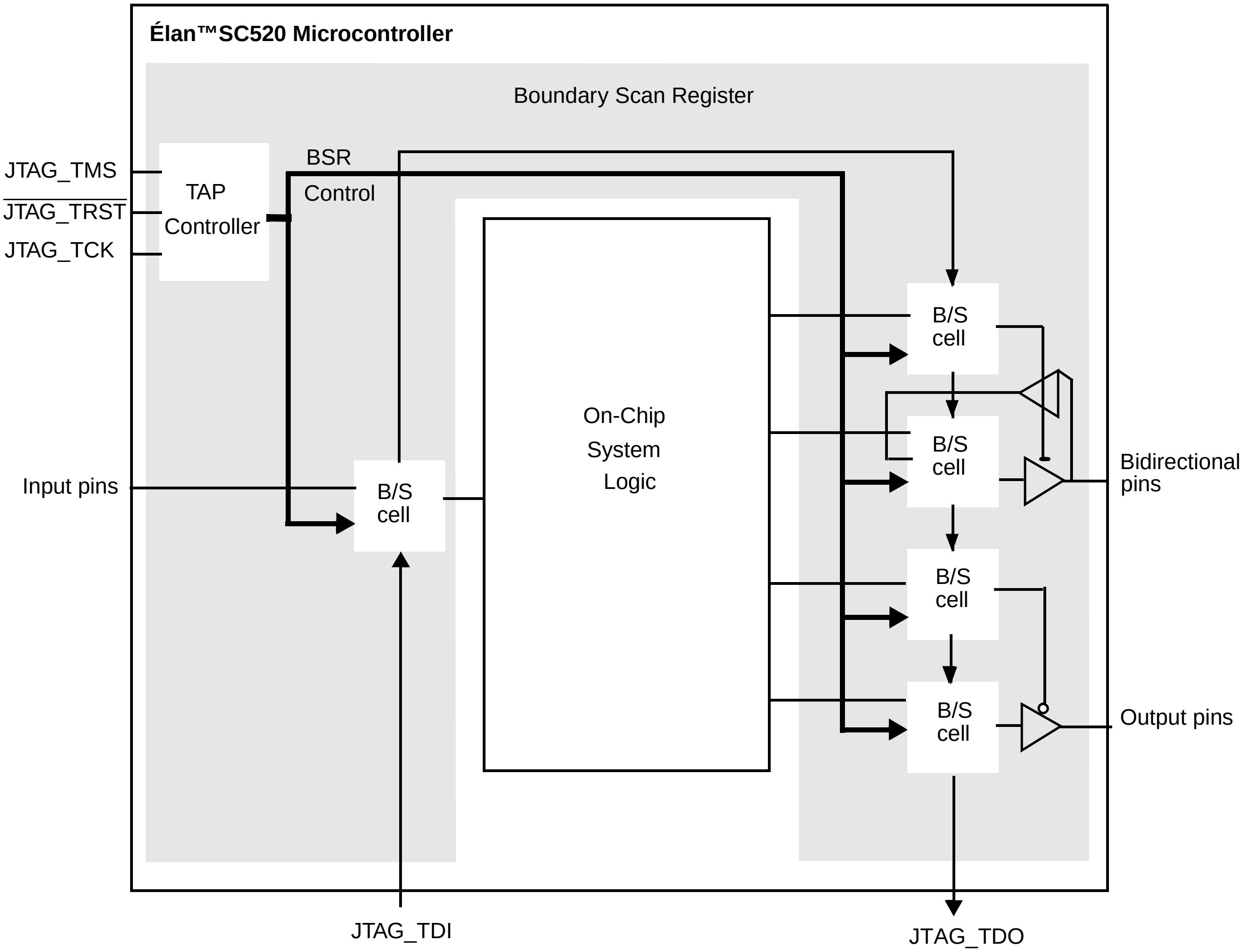
The ÉlanSC520 microcontroller provides test and debug features compliant with IEEE Standard Test Access Port (TAP) and Joint Test Action Group (JTAG) (IEEE Std 1149.1-1990). The test logic is provided to test and ensure that:

- Components function correctly
- Interconnections between various components are correct
- Various components interact correctly on the printed circuit board

25.2 BLOCK DIAGRAM

Figure 25-1 shows a block diagram of the Boundary Scan register of the ÉlanSC520 microcontroller.

Figure 25-1 Logical Structure of Boundary Scan Register



25.3 SYSTEM DESIGN

25.3.1 JTAG Pin Strapping

Designers using JTAG for board continuity testing commonly expect to exercise any pin in an arbitrary fashion. However, pinstrapping on the GPA25 pin could cause unexpected behavior. The pinstrap on the GPA25 pin is {DEBUG_ENTER}. If, at the assertion of PWRGOOD, {DEBUG_ENTER} is High, AMDebug mode will be enabled and the CPU will not perform as expected. The GPA25{DEBUG_ENTER} pin cannot be High at the assertion of PWRGOOD if the JTAG port is to be used for continuity testing.

25.4 REGISTERS

The ÉlanSC520 microcontroller contains four test data registers: Bypass register, Boundary Scan register, Device Identification register and Serial Debug Port Data register. A fifth register, the Instruction register, is used to specify the test to be executed and the data register to be accessed.

The Bypass register and Boundary Scan register are serially connected to JTAG_TDI and JTAG_TDO, with JTAG_TDI connected to the most significant bit and JTAG_TDO connected to the least significant bit of the test data register. Data is shifted one stage (bit position within the register) on each rising edge of the test clock (JTAG_TCK). Table 25-1 gives a description of each register. The Serial Debug Port Data register is part of the AMDebug utility and is physically located in the AMDebug logic. See Chapter 26, “AMDebug™ Technology”, for more information on the AMDebug interface.

Table 25-1 Chip Test and Debugging Registers

Register	Mnemonic	Function
Boundary Scan	BSR	A single shift register path containing the boundary scan cells that are connected to all input and output pins of the ÉlanSC520 microcontroller. Figure 25-1 shows the logical structure of the Boundary Scan register. Data is transferred without inversion from JTAG_TDI to JTAG_TDO through the Boundary Scan register during scanning. The Boundary Scan register is affected by the EXTEST and SAMPLE/PRELOAD instructions.
Bypass	BPR	Provides a path from JTAG_TDI to JTAG_TDO with one clock cycle latency.Used to bypass the chip completely while testing boards containing many chips.
Device Identification	DID	A 32-bit register that contains AMD’s ID code for the ÉlanSC520 microcontroller.
Serial Debug Port Data	SDPD	A 38 bit register that serves as a command/status/data interface with the Am5$_x$86 CPU processor. Figure 25-2 on page 25-14 shows the format.
Instruction	IR	Determines the test that has to be executed and the data register to access.

25.5 OPERATION

The test and debugging features on the ÉlanSC520 microcontroller include the following elements:

- **Pins**—JTAG_TDI, JTAG_TMS, JTAG_TDO, JTAG_TCK and $\overline{\text{JTAG\_TRST}}$. In addition, there are four pins for the AMDebug utility: CMDACK, BR/TC, STOP/TX, and TRIG/TRACE.
- **Instruction Register (IR)**—The instruction codes select the specific test or debug operation to be performed and the test data register to be accessed.

- **Test Data Registers**—Boundary Scan (BSR) register, Device Identification (DID) register, Bypass (BPR) register, and Serial Debug Port Data (SDPD) register.
- **Test Access Port (TAP) controller**—State-machine and control logic implementation.

The instruction and test data registers are separate shift-register paths connected in parallel that have a common serial data input and a common serial data output connected to the TAP signals, JTAG_TDI and JTAG_TDO, respectively.

25.5.1 Instruction Register

The Instruction register is a 4-bit register that allows instructions to be serially shifted into the device. The instruction determines the test to be executed and the data register to be accessed. The least significant bit is nearest the JTAG_TDO output. When the test access port (TAP) controller is reset, the Instruction register is loaded with the default instruction IDCODE.

25.5.1.1 Implemented Instructions

The ÉlanSC520 microcontroller supports all three mandatory boundary-scan instructions: BYPASS, SAMPLE/PRELOAD, and EXTEST, along with three additional instructions: IDCODE, HIGHZ and DEBUG.

Table 25-2 shows the test access port (TAP) instructions that are supported on the ÉlanSC520 microcontroller.

Table 25-2 Test Access Port Instruction Set

Instruction	IR3–IR0
EXTEST	0000
SAMPLE/PRELOAD	0001
IDCODE	0010
HIGHZ	0011
Reserved	0100
DEBUG	0101
Reserved	0110–1110
BYPASS	1111

25.5.1.1.1 EXTEST Instruction

The instruction code is 0000b. The EXTEST instruction allows testing of circuitry external to the component package, typically board interconnects. It does so by driving the values loaded into the microcontroller’s Boundary Scan register out on to the output pins corresponding to each boundary scan cell. It then captures the values on the microcontroller’s input pins to be loaded into their corresponding Boundary Scan register locations. I/O pins are selected as input or output, depending on the value loaded into their control setting locations in the Boundary Scan register. Values shifted into input latches in the Boundary Scan register are never used by the internal logic of the ÉlanSC520 microcontroller.

Note: *After using the EXTEST instruction, the ÉlanSC520 microcontroller should be reset before normal (non-boundary scan) use to ensure the state of the ÉlanSC520 microcontroller.*

25.5.1.1.2 SAMPLE/PRELOAD Instruction

The instruction code is 0001b. The SAMPLE/PRELOAD instruction performs two functions.

- When the TAP controller is in the Capture-DR state, the SAMPLE/PRELOAD instruction allows a “snapshot” of the normal operation of the ÉlanSC520 microcontroller without interfering with that normal operation. The instruction causes Boundary Scan register cells associated with outputs to sample the value being driven by the microcontroller. It causes the cells associated with inputs to sample the value being driven into the microcontroller. On both outputs and inputs, the sampling occurs on the rising edge of JTAG_TCK.
- When the TAP controller is in the Update-DR state, the SAMPLE/PRELOAD instruction preloads data to the device pins to be driven to the board by executing the EXTEST instruction. Data is preloaded to the pins from the Boundary Scan register on the falling edge of JTAG_TCK.

25.5.1.1.3 IDCODE Instruction

The instruction code is 0010b. The IDCODE instruction selects the Device Identification register to be connected to JTAG_TDI and JTAG_TDO, allowing the device identification code to be shifted out of the device on JTAG_TDO. Note that the Device Identification register is not altered by data being shifted in on JTAG_TDI.

25.5.1.1.4 HIGHZ Instruction

The instruction code is 0011b. The HIGHZ instruction connects the Bypass register between JTAG_TDI and JTAG_TDO. This instruction forces all outputs to a high-impedance state.

25.5.1.1.5 BYPASS Instruction

The instruction code is 1111b. The BYPASS instruction selects the Bypass register to be connected to JTAG_TDI or JTAG_TDO, effectively bypassing the test logic on the ÉlanSC520 microcontroller by reducing the shift length of the device to one bit.

Note that an open circuit fault in the board-level test data path causes the Bypass register to be selected following an instruction scan cycle due to the pullup resistor on the JTAG_TDI input. This has been done to prevent any unwanted interference with the proper operation of the system logic. The Instruction register can be accessed when this command is being executed, because only the Boundary Scan register is affected during this instruction.

25.5.1.1.6 DEBUG Instruction

The instruction code is 0101. The DEBUG instruction enables a 38-bit dedicated data register that serves as a command/status/data interface with Am5$_X$86 CPU processor. When the DEBUG instruction is written into the Instruction register, the serial debug shifter is connected to the JTAG TDI–TDO serial interface. The DEBUG command and data are loaded into and read from the serial debug shifter using the Capture-DR–Update-DR sequence in the TAP controller state machine.

Loading the DEBUG instruction enables additional AMDebug technology signals to provide pinpoint accuracy of external breakpoint assertion and elimination of status polling of the JTAG serial interface. These signals are: CMDACK, BR/TC, STOP/TX and TRIG/TRACE.

25.5.2 Configuration Information

There are five scan paths from JTAG_TDI to JTAG_TDO in the ÉlanSC520 microcontroller:

- Instruction path
- Bypass path
- Main data path through the Boundary Scan register
- Serial Debug Port Data register
- Device Identification register

25.5.2.1 Instruction Path

This four-cell path is used to scan into the Instruction register. This chain is loaded when the TAP controller is driven to the states Select-IR-Scan through Update-IR. See Figure 25-4 on page 25-15.

25.5.2.2 Bypass Path

This path bypasses the test logic on the microcontroller by reducing the shift length of the device to one bit. Commands can still be entered in the Instruction register during this operation.

25.5.2.3 Main Data Scan Path

Table 25-3 shows the main data scan path. The order shown is first-to-last; i.e., the first is closest to JTAG_TDI and the last is closest to JTAG_TDO. Control cells are used to control the enables of the three-state pads. If a 1 is shifted into the control cell, the associated pins are three-stated or selected as inputs.

Note: *Each of the shaded control cells shown in Table 25-3 contains the output enable control for the pads listed below the control cell and before the next control cell. For bidirectional pads, the output is listed first (closest to JTAG_TDI).*

Table 25-3 Main Data Scan Path

Pad Name	Scan Type	Boundary Scan Order
	Control	486
BA1	Output	485
BA0	Output	484
MA12	Output	483
MA11	Output	482
MA10	Output	481
MA9	Output	480
MA8	Output	479
MA7	Output	478
MA6	Output	477
MA5	Output	476
MA4	Output	475
MA3	Output	474
MA2	Output	473
MA1	Output	472
MA0	Output	471

Table 25-3 Main Data Scan Path (Continued)

Pad Name	Scan Type	Boundary Scan Order
	Control	470
MD31	Bidirectional	468, 469
MD30	Bidirectional	466, 467
MD29	Bidirectional	464, 465
MD28	Bidirectional	462, 463
MD27	Bidirectional	460, 461
MD26	Bidirectional	458, 459
MD25	Bidirectional	456, 457
MD24	Bidirectional	454, 455
MD23	Bidirectional	452, 453
MD22	Bidirectional	450, 451
MD21	Bidirectional	448, 449
MD20	Bidirectional	446, 447
MD19	Bidirectional	444, 445
MD18	Bidirectional	442, 443
MD17	Bidirectional	440, 441
MD16	Bidirectional	438, 439
MD15	Bidirectional	436, 437
MD14	Bidirectional	434, 435
MD13	Bidirectional	432, 433
MD12	Bidirectional	430, 431
MD11	Bidirectional	428, 429
MD10	Bidirectional	426, 427
MD9	Bidirectional	424, 425
MD8	Bidirectional	422, 423
MD7	Bidirectional	420, 421
MD6	Bidirectional	418, 419
MD5	Bidirectional	416, 417
MD4	Bidirectional	414, 415
MD3	Bidirectional	412, 413
MD2	Bidirectional	410, 411
MD1	Bidirectional	408, 409
MD0	Bidirectional	406, 407
MECC6	Bidirectional	404, 405
MECC5	Bidirectional	402, 403
MECC4	Bidirectional	400, 401
MECC3	Bidirectional	398, 399
MECC2	Bidirectional	396, 397
MECC1	Bidirectional	394, 395
MECC0	Bidirectional	392, 393
	Control	391
$\overline{\text{SCS3}}$	Output	390

Table 25-3 Main Data Scan Path (Continued)

Pad Name	Scan Type	Boundary Scan Order
$\overline{\text{SCS2}}$	Output	389
$\overline{\text{SCS1}}$	Output	388
$\overline{\text{SCS0}}$	Output	387
	Control	386
CLKMEMOUT	Output	385
CLKMEMIN	Input	384
	Control	383
$\overline{\text{SRASB}}$	Output	382
$\overline{\text{SRASA}}$	Output	381
	Control	380
$\overline{\text{SCASB}}$	Output	379
$\overline{\text{SCASA}}$	Output	378
	Control	377
$\overline{\text{SWEB}}$	Output	376
$\overline{\text{SWEA}}$	Output	375
	Control	374
SDQM3	Output	373
SDQM2	Output	372
SDQM1	Output	371
SDQM0	Output	370
	Control	369
$\overline{\text{BOOTCS}}$	Output	368
	Control	367
$\overline{\text{ROMRD}}$	Output	366
	Control	365
$\overline{\text{FLASHWR}}$	Output	364
	Control	363
$\overline{\text{ROMBUFOE}}$	Output	362
	Control	361
$\overline{\text{ROMCS1}}$	Output	360
	Control	359
$\overline{\text{ROMCS2}}$	Output	358
	Control	357
AD31	Bidirectional	355, 356
AD30	Bidirectional	353, 354
AD29	Bidirectional	351, 352
AD28	Bidirectional	349, 350
AD27	Bidirectional	347, 348
AD26	Bidirectional	345, 346
AD25	Bidirectional	343, 344
AD24	Bidirectional	341, 342
AD23	Bidirectional	339, 340

Table 25-3 Main Data Scan Path (Continued)

Pad Name	Scan Type	Boundary Scan Order
AD22	Bidirectional	337, 338
AD21	Bidirectional	335, 336
AD20	Bidirectional	333, 334
AD19	Bidirectional	331, 332
AD18	Bidirectional	329, 330
AD17	Bidirectional	327, 328
AD16	Bidirectional	325, 326
AD15	Bidirectional	323, 324
AD14	Bidirectional	321, 322
AD13	Bidirectional	319, 320
AD12	Bidirectional	317, 318
AD11	Bidirectional	315, 316
AD10	Bidirectional	313, 314
AD9	Bidirectional	311, 312
AD8	Bidirectional	309, 310
AD7	Bidirectional	307, 308
AD6	Bidirectional	305, 306
AD5	Bidirectional	303, 304
AD4	Bidirectional	301, 302
AD3	Bidirectional	299, 300
AD2	Bidirectional	297, 298
AD1	Bidirectional	295, 296
AD0	Bidirectional	293, 294
	Control	292
$\overline{\text{CBE3}}$	Bidirectional	290, 291
$\overline{\text{CBE2}}$	Bidirectional	288, 289
$\overline{\text{CBE1}}$	Bidirectional	286, 287
$\overline{\text{CBE0}}$	Bidirectional	284, 285
	Control	283
PAR	Bidirectional	281, 282
$\overline{\text{SERR}}$	Input	280
	Control	279
$\overline{\text{PERR}}$	Bidirectional	277, 278
	Control	276
$\overline{\text{FRAME}}$	Bidirectional	274, 275
	Control	273
$\overline{\text{TRDY}}$	Bidirectional	271, 272
	Control	270
$\overline{\text{IRDY}}$	Bidirectional	268, 269
	Control	267
$\overline{\text{STOP}}$	Bidirectional	265, 266
	Control	264

Table 25-3 Main Data Scan Path (Continued)

Pad Name	Scan Type	Boundary Scan Order
$\overline{\text{DEVSEL}}$	Bidirectional	262, 263
	Control	261
CLKPCIOUT	Output	260
CLKPCIIN	Input	259
	Control	258
$\overline{\text{RST}}$	Output	257
$\overline{\text{INTD}}$	Input	256
$\overline{\text{INTC}}$	Input	255
$\overline{\text{INTB}}$	Input	254
$\overline{\text{INTA}}$	Input	253
$\overline{\text{REQ4}}$	Input	252
$\overline{\text{REQ3}}$	Input	251
$\overline{\text{REQ2}}$	Input	250
$\overline{\text{REQ1}}$	Input	249
$\overline{\text{REQ0}}$	Input	248
	Control	247
$\overline{\text{GNT4}}$	Output	246
	Control	245
$\overline{\text{GNT3}}$	Output	244
	Control	243
$\overline{\text{GNT2}}$	Output	242
	Control	241
$\overline{\text{GNT1}}$	Output	240
	Control	239
$\overline{\text{GNT0}}$	Output	238
	Control	237
GPA25	Bidirectional	235, 236
GPA24	Bidirectional	233, 234
GPA23	Bidirectional	231, 232
GPA22	Bidirectional	229, 230
GPA21	Bidirectional	227, 228
GPA20	Bidirectional	225, 226
GPA19	Bidirectional	223, 224
GPA18	Bidirectional	221, 222
GPA17	Bidirectional	219, 220
GPA16	Bidirectional	217, 218
GPA15	Bidirectional	215, 216
GPA14	Output	214
GPA13	Output	213
GPA12	Output	212
GPA11	Output	211
GPA10	Output	210

Table 25-3 Main Data Scan Path (Continued)

Pad Name	Scan Type	Boundary Scan Order
GPA9	Output	209
GPA8	Output	208
GPA7	Output	207
GPA6	Output	206
GPA5	Output	205
GPA4	Output	204
GPA3	Output	203
GPA2	Output	202
GPA1	Output	201
GPA0	Output	200
	Control	199
GPD15	Bidirectional	197, 198
GPD14	Bidirectional	195, 196
GPD13	Bidirectional	193, 194
GPD12	Bidirectional	191, 192
GPD11	Bidirectional	189, 190
GPD10	Bidirectional	187, 188
GPD9	Bidirectional	185, 186
GPD8	Bidirectional	183, 184
	Control	182
GPD7	Bidirectional	180, 181
GPD6	Bidirectional	178, 179
GPD5	Bidirectional	176, 177
GPD4	Bidirectional	174, 175
GPD3	Bidirectional	172, 173
GPD2	Bidirectional	170, 171
GPD1	Bidirectional	168, 169
GPD0	Bidirectional	166, 167
	Control	165
GPRESET	Output	164
	Control	163
$\overline{\text{GPIORD}}$	Output	162
	Control	161
$\overline{\text{GPIOWR}}$	Output	160
	Control	159
$\overline{\text{GPMEMRD}}$	Output	158
	Control	157
$\overline{\text{GPMEMWR}}$	Output	156
	Control	155
PIO27	Bidirectional	153, 154
	Control	152
PIO26	Bidirectional	150, 151

Table 25-3 Main Data Scan Path (Continued)

Pad Name	Scan Type	Boundary Scan Order
	Control	149
PIO25	Bidirectional	147, 148
	Control	146
PIO24	Bidirectional	144, 145
	Control	143
PIO23	Bidirectional	141, 142
	Control	140
PIO22	Bidirectional	138, 139
	Control	137
PIO21	Bidirectional	135, 136
	Control	134
PIO20	Bidirectional	132, 133
	Control	131
PIO19	Bidirectional	129, 130
	Control	128
PIO18	Bidirectional	126, 127
	Control	125
PIO17	Bidirectional	123, 124
	Control	122
PIO16	Bidirectional	120, 121
	Control	119
PIO15	Bidirectional	117, 118
	Control	116
PIO14	Bidirectional	114, 115
	Control	113
PIO13	Bidirectional	111, 112
	Control	110
PIO12	Bidirectional	108, 109
	Control	107
PIO11	Bidirectional	105, 106
	Control	104
PIO10	Bidirectional	102, 103
	Control	101
PIO9	Bidirectional	99, 100
	Control	98
PIO8	Bidirectional	96, 97
	Control	95
PIO7	Bidirectional	93, 94
	Control	92
PIO6	Bidirectional	90, 91
	Control	89
PIO5	Bidirectional	87, 88

Table 25-3 Main Data Scan Path (Continued)

Pad Name	Scan Type	Boundary Scan Order
	Control	86
PIO4	Bidirectional	84, 85
	Control	83
PIO3	Bidirectional	81, 82
	Control	80
PIO2	Bidirectional	78, 79
	Control	77
PIO1	Bidirectional	75, 76
	Control	74
PIO0	Bidirectional	72, 73
	Control	71
SOUT1	Output	70
SIN1	Input	69
	Control	68
$\overline{\text{RTS1}}$	Output	67
$\overline{\text{CTS1}}$	Input	66
$\overline{\text{DSR1}}$	Input	65
	Control	64
$\overline{\text{DTR1}}$	Output	63
$\overline{\text{DCD1}}$	Input	62
$\overline{\text{RIN1}}$	Input	61
	Control	60
SOUT2	Output	59
SIN2	Input	58
	Control	57
$\overline{\text{RTS2}}$	Output	56
	Control	55
PIO28	Bidirectional	53, 54
	Control	52
PIO29	Bidirectional	50, 51
	Control	49
$\overline{\text{DTR2}}$	Output	48
	Control	47
PIO30	Bidirectional	45, 46
	Control	44
PIO31	Bidirectional	42, 43
	Control	41
SSI_CLK	Output	40
SSI_DI	Input	39
	Control	38
SSI_DO	Output	37
	Control	36

Table 25-3 Main Data Scan Path (Continued)

Pad Name	Scan Type	Boundary Scan Order
CLKTIMER	Bidirectional	34, 35
PWRGOOD	Input	33
PRGRESET	Input	32
	Control	31
CMDACK	Output	30
BR/TC	Input	29
	Control	28
STOP/TX	Output	27
	Control	26
TRIG/TRACE	Output	25
	Control	24
DC	Bidirectional	22, 23
	Control	21
DATASTRB	Bidirectional	19, 20
	Control	18
CPUACT	Bidirectional	16, 17
	Control	15
PITOUT2	Bidirectional	13, 14
	Control	12
PITGATE2	Bidirectional	10, 11
	Control	9
TMRIN1	Bidirectional	7, 8
	Control	6
TMRIN0	Bidirectional	4, 5
	Control	3
TMROUT1	Output	2
	Control	1
TMROUT0	Output	0

Notes:

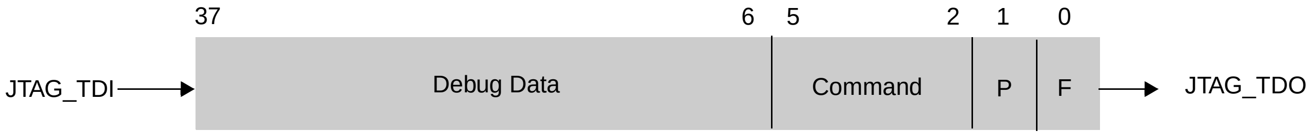
The control cell for the BA1–BA0 and MA12–MA0 pins is closest to the JTAG_TDI pin (beginning of the boundary scan chain), and TMROUT0 is closest to the JTAG_TDO pin (end of the boundary scan chain).

Each of the shaded control cells shown in Table 25-3 contains the output enable control for the pads listed below the control cell and before the next control cell. For bidirectional pads, the output is listed first (closest to JTAG_TDI).

25.5.2.4 Serial Debug Port Data Register

Figure 25-2 shows the format of the Serial Debug Port Data register. The 38-bit Serial Debug Port Data register serves as a command/status/data interface with the Am5$_x$86 CPU.

Figure 25-2 Serial Debug Port Data Register Format

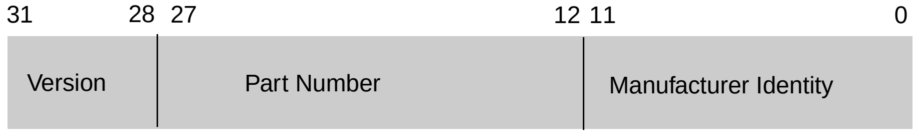


Bit	Name	Function
37–6	DEBUG_DATA[31–0]	Debug Data
5–2	COMMAND[3–0]	Command
1	P	Command pending flag status
0	F	Command finished flag status

25.5.2.5 Device Identification Register

Figure 25-3 shows the format of the Device Identification register. For the ÉlanSC520 microcontroller, the least significant 28-bits of the Device Identification register are hard-coded to a value of 0EFF003h. The VERSION field, represented by bits 31–28, reflects the value of the MINORSTEP field of the ÉlanSC520 Microcontroller Revision ID (REVID) register (MMCR offset 00h).

Figure 25-3 Device Identification Register Format



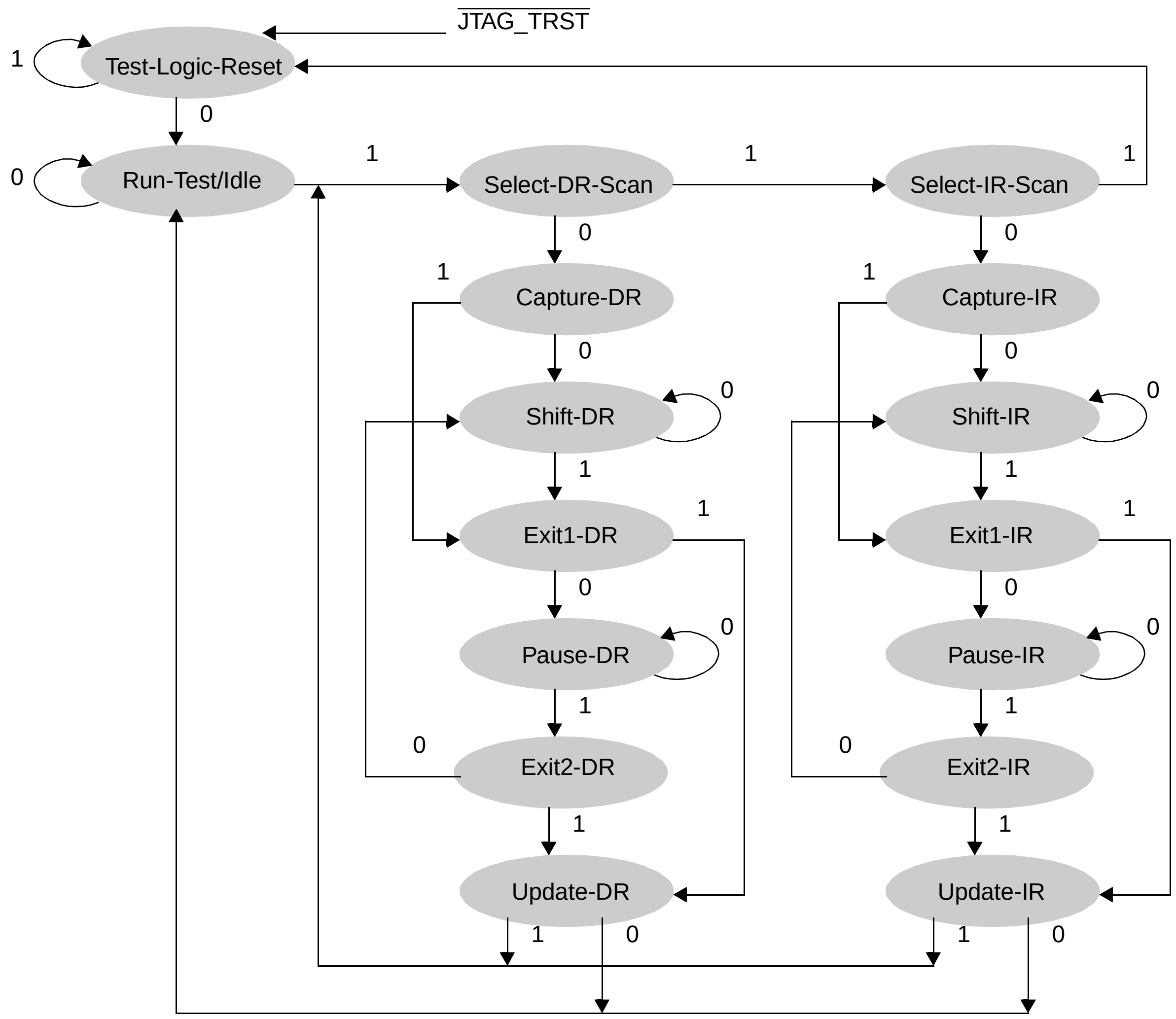
Bit	Name	Function
31–28	VERSION	Value of the MINORSTEP field of the ÉlanSC520 Microcontroller Revision ID (REVID) register
27–0	Part Number and Manufacturer Identity	Hardcoded to 0EFF003h

25.5.3 Test Access Port (TAP) Controller

The test access port (TAP) controller is a synchronous, finite state-machine that controls the sequence of operations of the test logic. The TAP controller changes state in response to the rising edge of JTAG_TCK. It can be reset to the Test-Logic-Reset state either by holding the $\overline{\text{JTAG\_TRST}}$ pin Low or by holding the JTAG_TMS pin High for five JTAG_TCK periods.

The TAP controller state-machine is shown in Figure 25-4.

Figure 25-4 Test Access Port Controller State Diagram



25.5.3.1 TAP Controller States

25.5.3.1.1 Test-Logic-Reset State

In this state, the test logic is disabled so that normal operation of the device can continue unhindered. This is achieved by initializing the Instruction register such that the IDCODE instruction is loaded. No matter what the original state of the controller, the controller enters Test-Logic-Reset state when the JTAG_TMS input is held High (1) for at least five rising edges of JTAG_TCK. The controller remains in this state while JTAG_TMS is High. The TAP controller is also forced to enter this state when $\overline{\text{JTAG\_TRST}}$ is asserted.

The JTAG TAP controller is not reset as a function of PWRGOOD when the system is powered up. Rather, $\overline{\text{JTAG\_TRST}}$ has an internal pulldown resistor which causes the TAP controller to reset.

25.5.3.1.2 Run-Test-Idle State

This is a controller state between scan operations. When in this state, the controller remains in this state as long as JTAG_TMS is held Low. For instructions not causing functions to execute during this state, no activity occurs in the test logic. The Instruction register and all test data registers retain their previous state. When JTAG_TMS is High and a rising edge is applied to JTAG_TCK, the controller moves to the Select-DR state.

25.5.3.1.3 Select-Data Register (DR)-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If JTAG_TMS is held Low and a rising edge is applied to JTAG_TCK when in this state, the controller moves into the Capture-DR state and a scan sequence for the selected test data register is initiated. If JTAG_TMS is held High and a rising edge is applied to JTAG_TCK, the controller moves to the Select-IR-Scan state.

The instruction does not change in this state.

25.5.3.1.4 Capture-DR State

In this state, the Boundary Scan register captures input pin data if the current instruction is EXTEST or SAMPLE/PRELOAD. The other test data registers, which do not have parallel input, are not changed.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to JTAG_TCK, the controller enters the Exit1-DR state if JTAG_TMS is High, or the Shift-DR state if JTAG_TMS is Low.

25.5.3.1.5 Shift-DR State

In this controller state, the test data register connected between JTAG_TDI and JTAG_TDO as a result of the current instruction shifts data one stage toward its serial output on each rising edge of JTAG_TCK.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to JTAG_TCK, the controller enters the Exit1-DR state if JTAG_TMS is High, or remains in the Shift-DR state if JTAG_TMS is Low.

25.5.3.1.6 Exit1-DR State

This is a temporary state. While in this state, if JTAG_TMS is held High, a rising edge applied to JTAG_TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If JTAG_TMS is held Low and a rising edge is applied to JTAG_TCK, the controller enters the Pause-DR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

25.5.3.1.7 Pause-DR State

The pause state allows the test controller to temporarily halt the shifting of data through the test data register in the serial path between JTAG_TDI and JTAG_TDO. An example of using this state could be to allow a tester to reload its pin memory from disk during application of a long test sequence.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as JTAG_TMS is Low. When JTAG_TMS goes High and a rising edge is applied to JTAG_TCK, the controller moves to the Exit2-DR state.

25.5.3.1.8 Exit2-DR State

This is a temporary state. While in this state, if JTAG_TMS is held High, a rising edge applied to JTAG_TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If JTAG_TMS is held Low and a rising edge is applied to JTAG_TCK, the controller enters the Shift-DR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

25.5.3.1.9 Update-DR State

The Boundary Scan register is provided with a latched parallel output to prevent changes at the parallel output while data is shifted in response to the EXTEST and SAMPLE/PRELOAD instructions. When the TAP controller is in this state and the Boundary Scan register is selected, data is latched onto the parallel output of this register from the shift-register path on the falling edge of JTAG_TCK. The data held at the latched parallel output does not change other than in this state.

All shift-register stages in a test data register selected by the current instruction retain their previous values during this state. The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to JTAG_TCK, the controller enters the Select-DR State if JTAG_TMS is held High or the Run-Test/Idle State if JTAG_TMS is held Low.

25.5.3.1.10 Select-Instruction Register (IR)-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If JTAG_TMS is held Low and a rising edge is applied to JTAG_TCK when in this state, the controller moves into the Capture-IR state and a scan sequence for the Instruction register is initiated. If JTAG_TMS is held High and a rising edge is applied to JTAG_TCK, the controller moves to the Test-Logic-Reset state.

The instruction does not change in this state.

25.5.3.1.11 Capture-IR State

In this controller state, the shift register contained in the Instruction register loads the fixed value 0001b on the rising edge of JTAG_TCK.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to JTAG_TCK, the controller enters the Exit1-IR state if JTAG_TMS is held High, or the Shift-IR state if JTAG_TMS is held Low.

25.5.3.1.12 Shift-IR State

In this state, the shift register contained in the Instruction register is connected between JTAG_TDI and JTAG_TDO and shifts data one stage towards its serial output on each rising edge of JTAG_TCK.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to JTAG_TCK, the controller enters the Exit1-IR state if JTAG_TMS is held High, or remains in the Shift-IR state if JTAG_TMS is held Low.

25.5.3.1.13 Exit1-IR State

This is a temporary state. In this state, if JTAG_TMS is held High, a rising edge applied to JTAG_TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If JTAG_TMS is held Low and a rising edge is applied to JTAG_TCK, the controller enters the Pause-IR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

25.5.3.1.14 Pause-IR State

The pause state allows the test controller to temporarily halt the shifting of data through the Instruction register.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as JTAG_TMS is Low. When JTAG_TMS goes High and a rising edge is applied to JTAG_TCK, the controller moves to the Exit2-IR state.

25.5.3.1.15 Exit2-IR State

This is a temporary state. While in this state, if JTAG_TMS is held High, a rising edge applied to JTAG_TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If JTAG_TMS is held Low and a rising edge is applied to JTAG_TCK, the controller enters the Shift-IR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

25.5.3.1.16 Update-IR State

The instruction shifted into the Instruction register is latched onto the parallel output from the shift-register path on the falling edge of JTAG_TCK. When the new instruction has been latched, it becomes the current instruction.

Test data registers selected by the current instruction retain their previous value.

When the TAP controller is in this state and a rising edge is applied to JTAG_TCK, the controller enters the Select-DR State if JTAG_TMS is held High or Run-Test Idle state if JTAG_TMS is held Low.

25.5.4 Bus Cycles

Figure 25-5 and Figure 25-6 on page 25-20 give the bus cycles information of the test logic operation in data scan mode and instruction scan mode, respectively.

Figure 25-5 Test Logic Operation: Data Scan

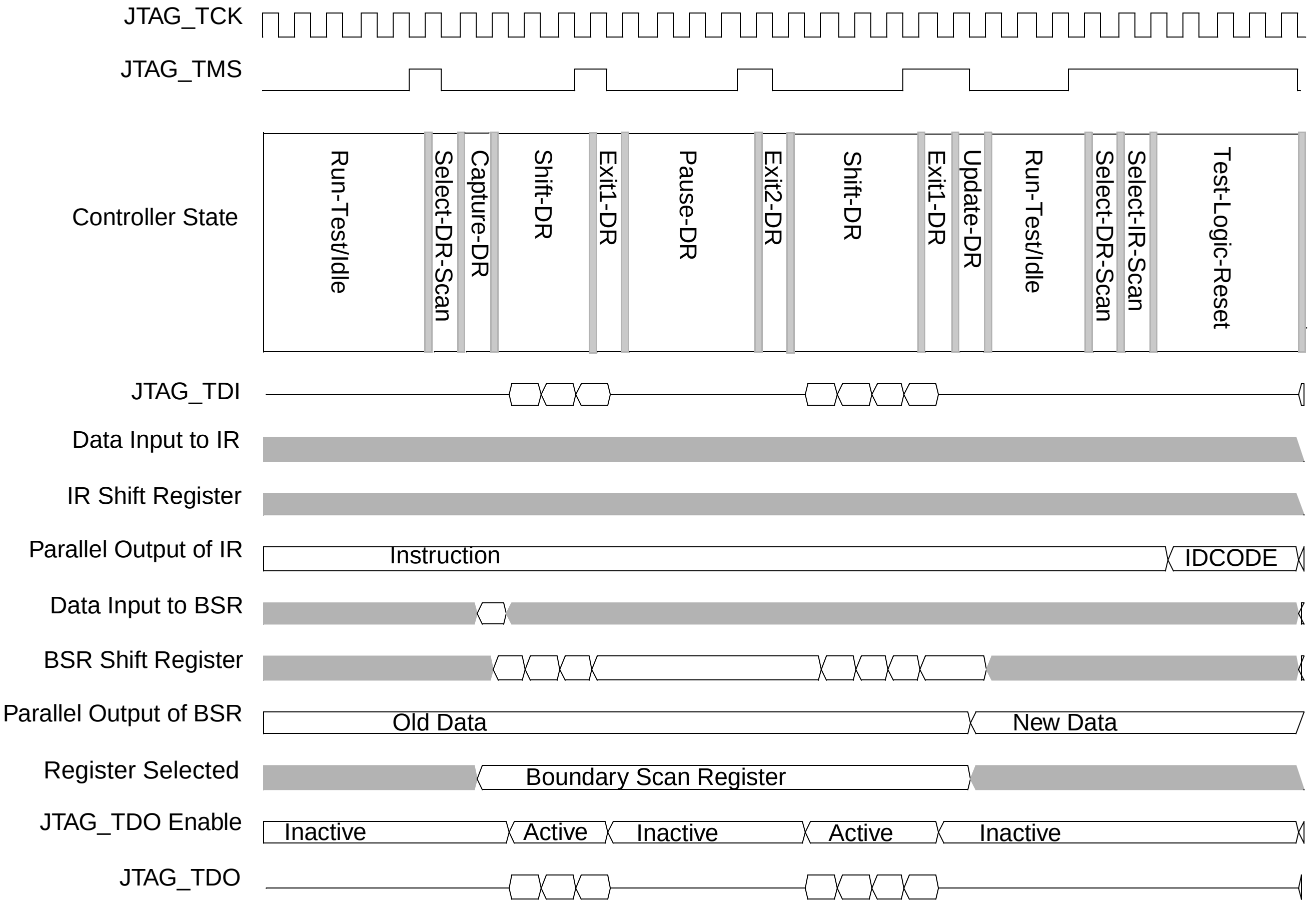
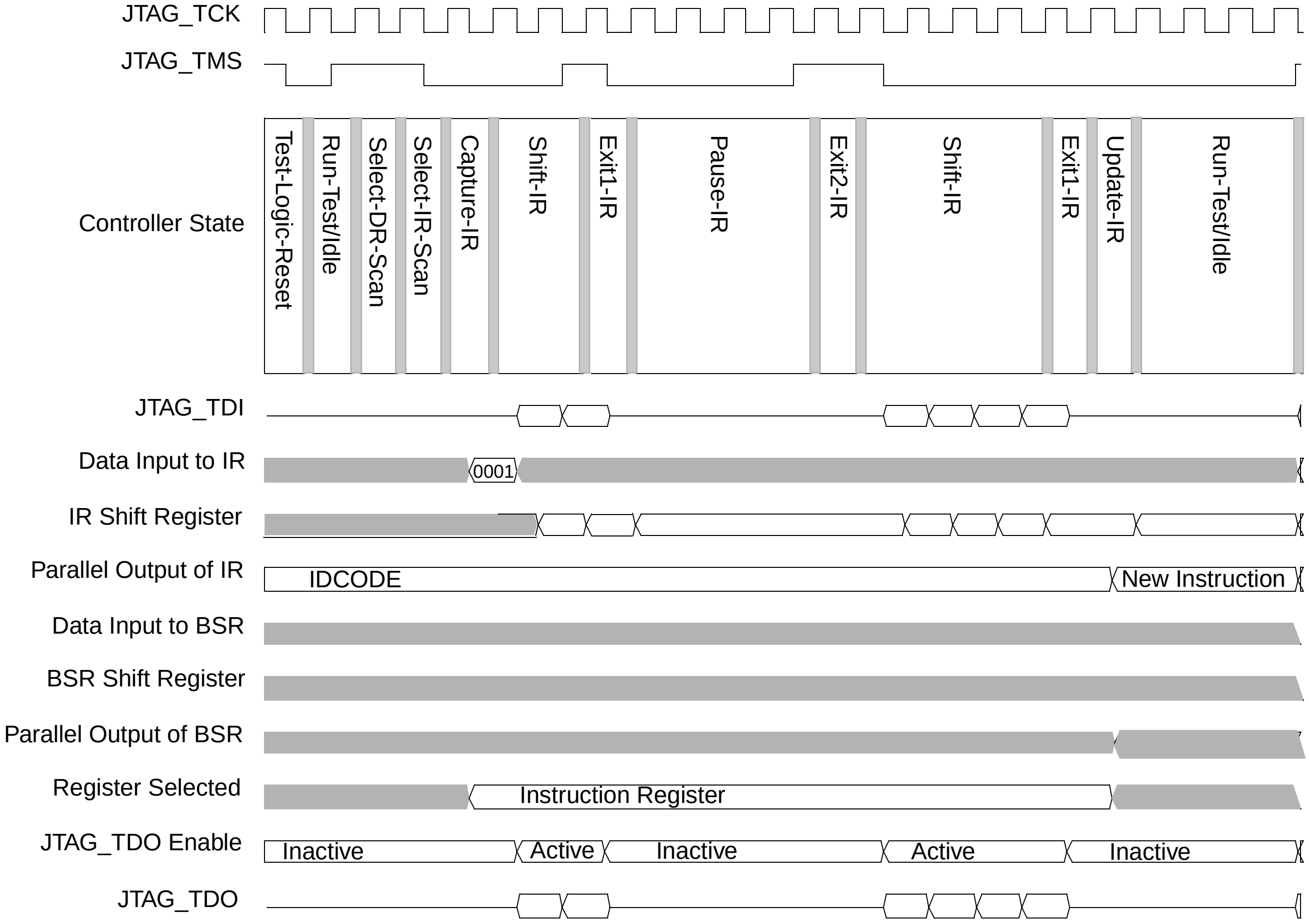


Figure 25-6 Test Logic Operation: Instruction Scan



25.5.5 Clocking Considerations

The targeted speed of operation for boundary scan is 25 MHz.

25.6 INITIALIZATION

The JTAG TAP controller is not reset as a function of PWRGOOD when the system is powered up.

The test access port controller can be reset in the following ways:

- When the $\overline{\text{JTAG\_TRST}}$ pin is driven Low (0)—This resets the entire JTAG subsystem including the Instruction register.
- When the JTAG_TMS pin is held High (1) for at least five rising edges of JTAG_TCK—It remains in this state while JTAG_TMS is High (1). If the TAP controller leaves the reset state owing to an erroneous Low (0) signal on the JTAG_TMS line at the time of a rising edge on JTAG_TCK, it returns to the reset state after JTAG_TMS is held High for three rising edges of JTAG_TCK.

In the Test-Logic-Reset State of the TAP controller, the test logic is disabled so that normal operation of the device can continue without any hindrance. See “Test-Logic-Reset State” on page 25-15.

CHAPTER

26 AMDebug™ TECHNOLOGY

26.1 OVERVIEW

The ÉlanSC520 microcontroller supports a full-featured, high-performance in-circuit emulation capability. This in-circuit emulation support was developed at AMD specifically to enable users to test and debug their software earlier in the design cycle. Utilizing this capability, the software can be more extensively exercised, and at full execution speeds. It also allows tracing during execution from the Am5$_X$86 CPU's internal cache.

The AMDebug interface included on the ÉlanSC520 microcontroller provides the product design team with two different communication paths, each of which is supported by powerful debug tools from third-party vendors in AMD's FusionE86 program. (See AMD FusionE86 partners documentation on p. iii under Third-Party Support.)

- Serial AMDebug technology uses a serial connection based on an enhanced JTAG protocol and an inexpensive 12-pin connector that can be placed on each board design. This low-cost solution satisfies the requirement of a large number of software developers.
- Parallel AMDebug technology uses a 25-pin parallel debug port to exchange commands and data between the ÉlanSC520 microcontroller and the host. The higher pin count requires that the extra signal pins be provided on a special bond-out package of the ÉlanSC520 microcontroller; this package is made available only to tool developers such as in-circuit emulator manufacturers. The parallel AMDebug port greatly simplifies the task of supporting high-speed data exchange.

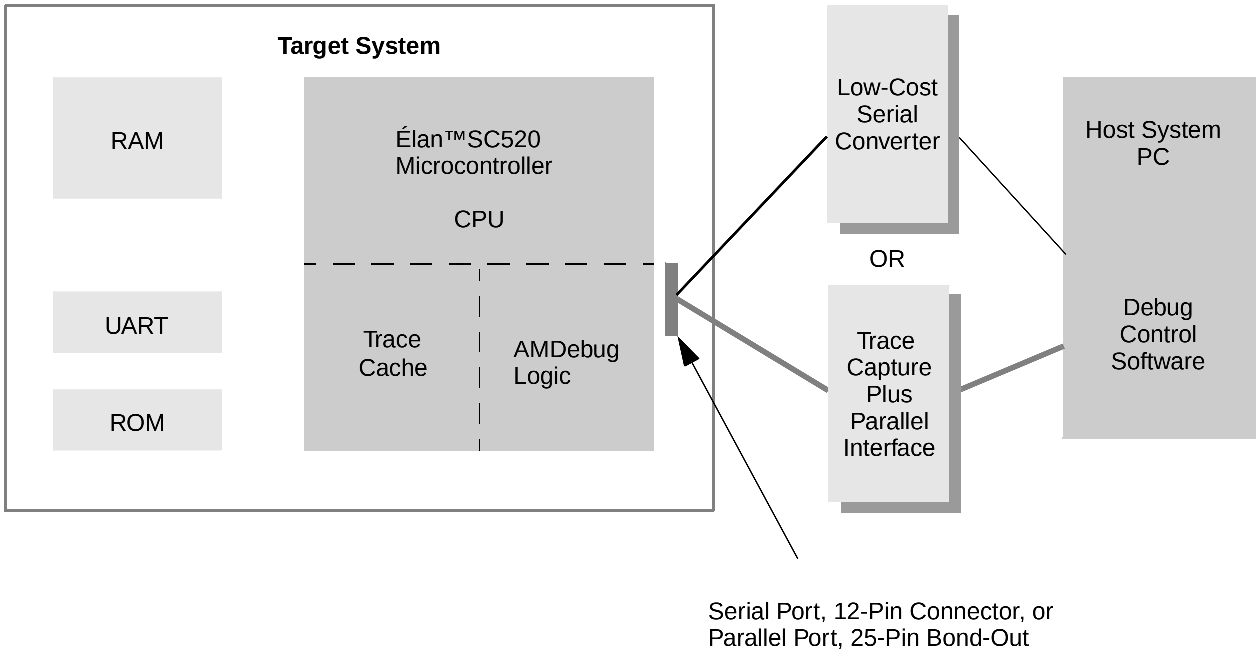
An on-chip trace controller provides trace information for reconstructing instruction execution flow in the processor. It supports tracing either to the serial AMDebug port, the bond-out parallel port, or to an internal trace buffer.

Use of JTAG technology for conventional boundary scan testing is described in Chapter 25, “Boundary Scan Test Interface”.

26.2 BLOCK DIAGRAM

Figure 26-1 shows a system diagram of AMDebug software architecture. Two different configurations are shown.

Figure 26-1 AMDebug™ Technology Software Architecture



26.3 SYSTEM DESIGN

The pinstrap functions associated with the GPA25–GPA23 pins, DEBUG_ENTER, INST_TRCE, and AMDEBUG_DIS, are sampled only as a result of PWRGOOD assertion and do not affect the GP bus functions of these pins.

Note that AMDebug technology does not function at 133 MHz. Debugging must be performed at 100 MHz.. Software tests and diagnostics can still be performed at 133 MHz if required, but the AMDebug port will not function reliably at frequencies above 100 MHz.

26.3.1 Connecting the AMDebug™ Port

There are multiple ways of connecting the host computer to the ÉlanSC520 microcontroller's AMDebug port, including through a host computer's serial port, parallel port, or via an Ethernet connection. For specific tool and connection types, refer to AMD FusionE86 partners documentation on p. iii under Third-Party Support.

At a minimum, AMDebug operation can be achieved with the four basic JTAG signal pins: JTAG_TCK, JTAG_TMS, JTAG_TDI, and JTAG_TDO. Using JTAG pins alone, without the advantages of additional support pins, the lowest possible cost is achieved in terms of processor pins, but with the cost of reduced functionality. No attempt is made to multiplex the function of the JTAG pins. Multiplexing would prevent ensuring their availability for communication with the processor at all times and under any operating condition.

An inexpensive connector that links the PC port to the AMDebug port can be acquired to satisfy the requirement of a large number of software developers. Connection to a target via this simple arrangement offers considerable advantages:

- There is no need to remove the processor to connect an in-circuit emulator-like umbilical.
- Connection is ensured no matter what the processor packaging technology.
- Debug communication is independent of processor or memory system clocking speeds.

There are two AMDebug connector formats specified: a 12-pin connector (Figure 26-2) and a 20-pin connector (Figure 26-3). They differ in maximum operating frequency and number of connector pins. They both have the same number of active signals, but the 20-pin version has a ground wire placed between each signal wire.

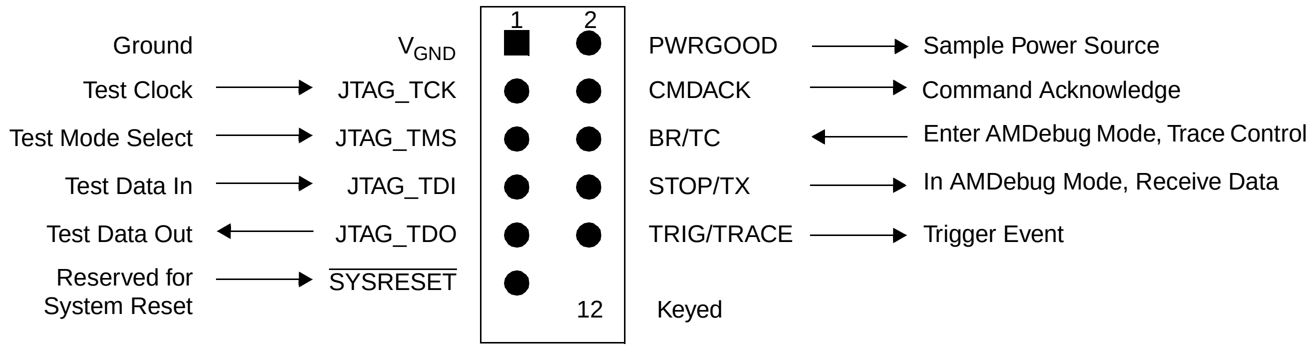
Table 26-1 AMDebug™ Technology Connector Pins

Name	I/O	External Resistor	Description
JTAG_TCK	Input	PU	Clock for the TAP controller and the debug serial/parallel interface
JTAG_TDI	Input	PU	Input test data and instructions
JTAG_TDO	Output	PU	Output data; three-stated when data is not driven
JTAG_TMS	Input	PU	Test functions and sequence of test changes
$\overline{\text{JTAG\_TRST}}$	Input	PU	Reset the JTAG controller
$\overline{\text{SYSRESET}}$	Input	PU	Reset all system logic. This pin should be held Low for at least four TCK clock cycles. $\overline{\text{SYSRESET}}$ can be ANDed directly with the PWRGOOD signal. This enables the AMDebug control unit to drive the ÉlanSC520 microcontroller's reset.
BR/TC	Input	PD	Request entry to AMDebug mode/Turn instruction trace capture on-off
CMDACK	Output	—	Acknowledge command
STOP/TX	Output	—	Asserted High on entry to AMDebug mode; during normal mode set High
TRIG/TRACE	Output	—	Trigger event to logic analyzer (optional, from Am486 debug registers)
PWRGOOD	Output	—	Sample power level used by the JTAG controller driving hardware

Software development systems based on the integrated debug technology should consider:

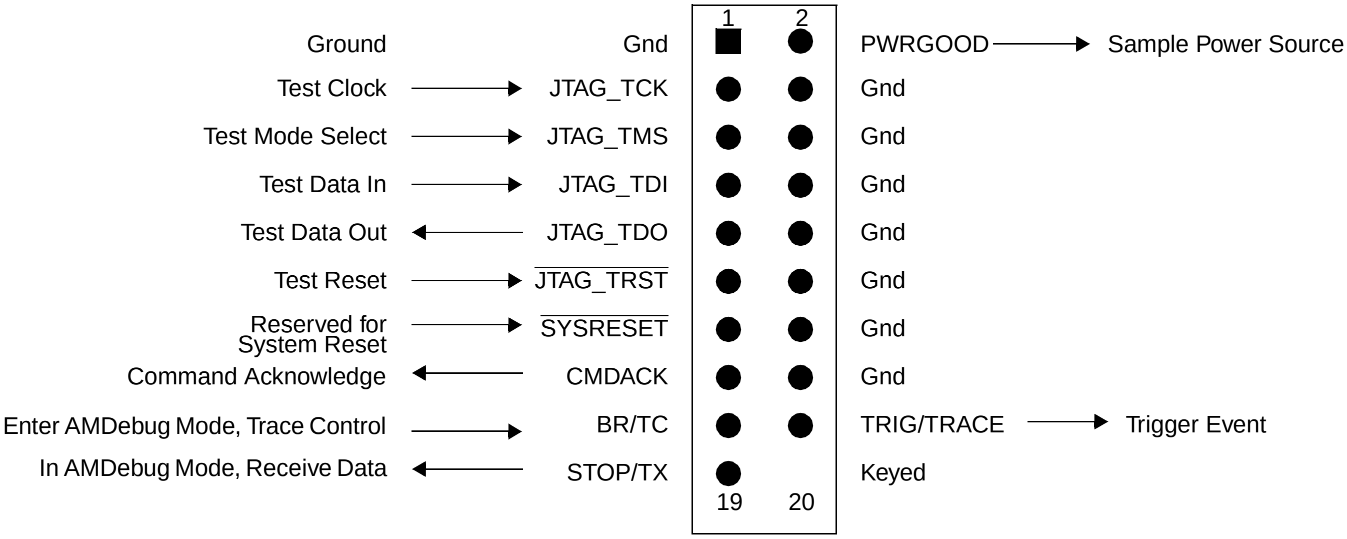
- Providing for at least a 12-pin connector on each board design
- Assigning the necessary tracking from the processor’s pins supporting the AMDebug port to the standard 12-pin connector
- Including the small connector on production systems to enable in-field debugging

Figure 26-2 12-Pin Connector Format



When the serial connector is clocked at high speeds, e.g., above 10 MHz, there is danger of signal cross talk. To alleviate this problem, a 20-pin serial connector format is also available, as shown in Figure 26-3. The arrangement places a ground wire between each signal wire. Low-cost tools based on AMDebug technology operate satisfactorily with the 12-pin connector shown in Figure 26-2, as long as cable lengths are not too long.

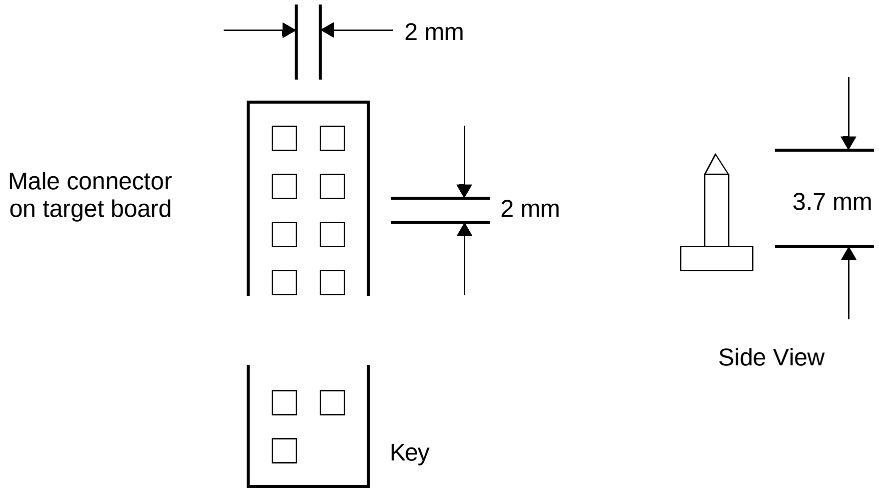
Figure 26-3 20-Pin Serial Connector Format



26.3.2 Mechanical Specifications for the Target Connector

A target board should contain a connector with male header pins. Pin spacing is 2 mm for both 12-pin and 20-pin formats, as shown in Figure 26-4. Debugging equipment should support a ribbon cable equipped with a female connector for attaching to the target. The appropriate last pin, pin 12 or pin 20, should not be installed, or, if necessary, removed at this location. At this location the female connector on the ribbon cable is populated with a post, which prevents the connector's insertion in the reverse position. Compatible connectors are available from Samtec, Inc. (model TMM-112-02-x-D for 12-pin), 3M, and other companies.

Figure 26-4 Mechanical Specifications for AMDebug™ Technology Target Connector

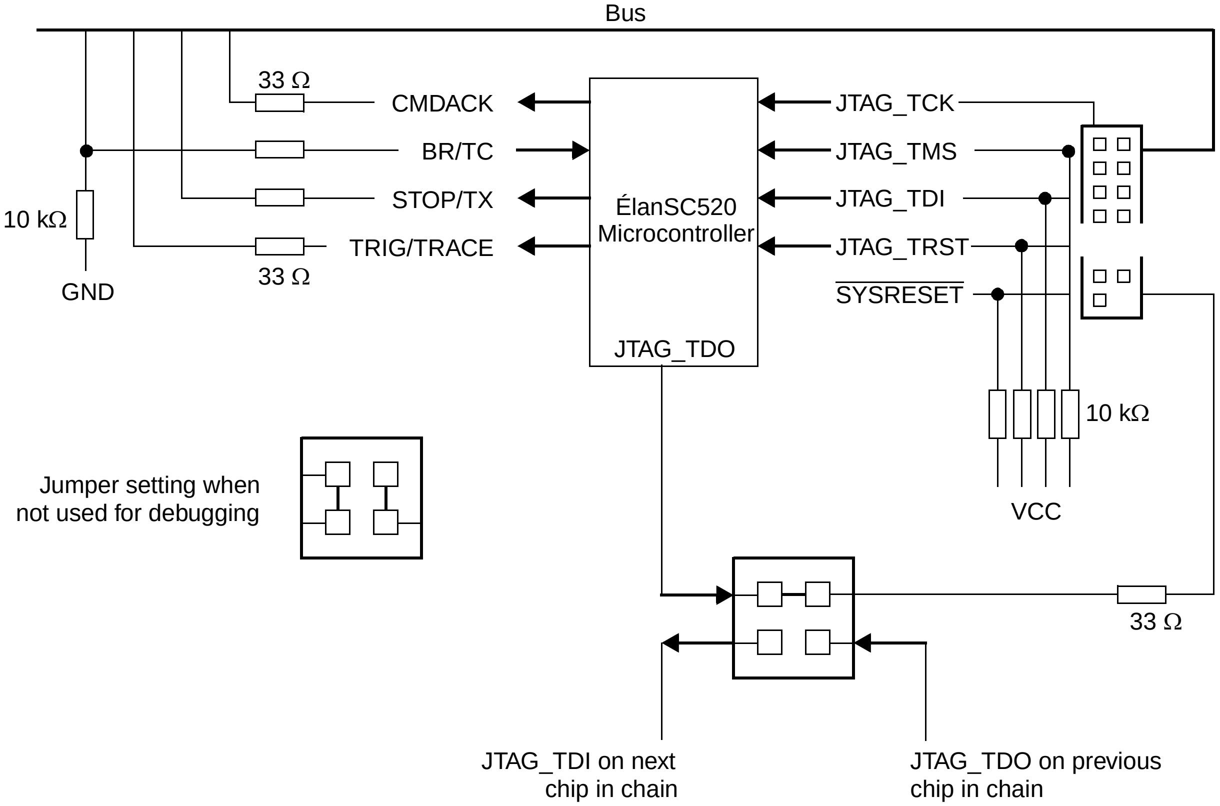


26.3.3 Locating the Connector on the Target System

Because the AMDebug port can contain high-frequency signals, position the connector as close to the processor as possible. However, allowances should be made for the physical requirements of the AMDebug control unit. For systems that support JTAG-based boundary scanning, a jumper block should be provided for isolating from the rest of the JTAG scan chain (see Figure 26-5) the connection from the AMDebug port to the processor. This jumper is not required by systems that only use JTAG to support AMDebug technology. When AMDebug technology debugging is not used, the jumpers can be set to connect the processor with the other devices forming the scan chain.

The target board should be equipped with pullup and pulldown resistors, as shown in Figure 26-5. The signal lines driven by the female cable connector should be three-stated before connection is established. The connection device can sense the required high voltage by sampling the VCC signal pin before driving the AMDebug port.

Figure 26-5 Locating the Target Connector



26.4 OPERATION

The AMD software debugging strategy enables a range of debugging tool solutions offered by tool providers. The AMDebug port provides for commands to be sent to the ÉlanSC520 microcontroller for processing by microcode. The AMDebug communication and data registers are used to exchange information between the target (ÉlanSC520 microcontroller) and a host system used to control the target.

The low-cost communication path, which meets the requirements of most software developers, uses the serial connection based on the enhanced JTAG protocol. This option requires very few signal pins to the processor and enables a 12-pin connector to be placed on each board design. A PC-port-to-AMDebug-port converter can be acquired inexpensively (see AMD FusionE86 partners documentation on p. iii under Third-Party Support).

The high-performance communication path, which is made available only to tool developers such as in-circuit emulator manufacturers, uses a parallel port connection that provides command and data exchange between the AMDebug port and the host. The higher pin count (25 instead of 8) requires that the extra signal pins be provided on a bond-out package. The die for both connection methods is the same. A standard parallel-interface format greatly simplifies the task of supporting high-speed data exchange with the target processor.

The parallel access also enables execution trace data to be provided on the bond-out parallel access pins. This is the same data that is gathered in the on-chip trace cache, described in Section 26.4.1, except that now trace depth is limited by the external hardware rather than the depth of the on-chip trace cache provided by the non bond-out processor.

26.4.1 On-Chip Trace Cache

An on-chip instruction cache makes it difficult to fully trace a program's execution path by merely observing the external pins. Software engineers need to know a program's address flow without turning off the cache or in any way intrusively monitoring the processor's operation. The use of clock scaling and high internal clock speed make it difficult to provide trace information to the outside world without the use of on-chip trace cache.

The AMD software debug strategy provides for a small on-chip trace cache that stores only critical information, such as the outcome of a branch decision. The compression techniques employed enable much of the execution path to be retained in the on-chip trace cache. The cache can also log other information, such as operating system activity or performance-critical parameters. On the bond-out package, the trace information can be continually provided off-chip at system bus speeds rather than the higher internal clock speeds. This is an advantage to the in-circuit emulator developer.

The trace cache is also useful when a multitasking operating system is employed. It is possible to unobtrusively trace the execution of a single task, thus extending the debugging capability beyond what is normally offered by debuggers incorporated with operating systems. This method overcomes the typically poor integration between operating systems and external trace capture hardware, such as a logic analyzer or in-circuit emulator.

26.4.2 Software Performance Profiling

Software profiling refers to examining the execution times, frequencies, and calling patterns of different software procedures within a complete program. A variety of techniques are currently used, some based on statistical analysis, others based on measurements achieved without statistical sampling. Execution times and call linkage are typically captured by external (off-chip) instrumentation watching the system buses.

Performance profiling is an exceptionally useful tool for the software engineer trying to optimize application execution times. When a bond-out package is used, an external hardware device can be constructed to capture the necessary data. When used for instruction tracing, the trace cache will contain more information than is necessary to perform only profile analysis. Typically, code first must be “instrumented” before it can be analyzed.

Alternatively, to support performance profiling, the breakpoint control registers can be further configured to start and stop a counter that measures elapsed time. When the counter is started, it is first set to zero, and when the counter is stopped, its value is placed in the trace cache. This scheme does not have all of the capabilities offered by a system assisted by external (off-chip) hardware; however, no instrumentation of code is required before profiling can commence.

In this case:

- A breakpoint/trace control register is set to trigger (start) the counter on the entry to a procedure.
- A second breakpoint control register is used to stop counting when the procedure prologue is entered.
- A similar scheme is used to measure other parameters, such as an interrupt handler execution times.
- When the counter is stopped, the 16-bit count value is placed in the trace cache.
- Post processing software is required to analyze the profile data.

The trace cache, when used to profile software, does not gather execution trace information. When profiling, the trace cache gathers information about the execution time spent, for example, in the selected procedure. Only one procedure can be profiled at one time. By examining the trace cache, the minimum, average, and maximum time spent in the procedure can be determined (within the limitations of the samples gathered).

A trace entry takes the form of a pair of time values. A second counter runs continually, but is reset to zero after it is placed in the trace cache. The second counter is used to obtain the frequency of occurrence of the procedure of interest; whereas, the first counter provides information about the procedures execution time (duration).

INDEX

Numerics

- 32KXTAL2–32KXTAL1 signals
  - description, 2-10
  - usage, 5-3, 5-8
- 33MXTAL2–33MXTAL1 signals
  - description, 2-10
  - usage, 2-7, 5-3

A

- A10–A8 bit field, 15-18
- a20 gate support, 6-8
- A20G_CTL bit field, 6-8
- GP bus. *See* general-purpose (GP) bus.
- AD31–AD0 signals
  - description, 2-6
  - usage, 2-7, 9-3, 9-9
- ADDDECCTL register, 4-2
- address buses
  - general-purpose (GP) bus address bus, 2-6
  - PCI address bus, 2-6
  - ROM/Flash controller address bus, 2-6
  - SDRAM controller address bus, 2-5
  - shared buses, 13-10
- Address Decode Control (ADDDECCTL) register
  - function, 4-2, 20-6, 21-3, 24-2
  - usage, 3-21, 4-12, 4-13, 4-14, 4-20, 20-5, 24-11
- address mapping
  - bus master address spaces (table), 4-3
  - cacheability control, 4-15, 4-20
  - code execution control, 4-15, 4-20
  - configuration, 4-15
    - buses, 4-4
    - chip select for noncontiguous memory or I/O, 4-16
    - chip selects, 4-4
    - external memory, 4-4
    - external super I/O chip, 4-16
    - GP bus peripheral space, 4-16
    - memory regions above DOS application space, 4-17
    - PCI bus devices, 4-18
    - ROM/Flash space, 4-15
    - Windows® compatibility, 4-17
  - configuration register access, 4-20
  - I/O map (figure), 4-11
  - I/O space, 4-11
    - Configuration Base Address register, 4-11
    - GP bus I/O region, 4-15
    - PC/AT-compatible I/O peripherals region, 4-13
    - PCI configuration space, 4-12
    - PCI I/O space, 4-12
  - initialization, 4-21
  - interrupts, 4-18
  - memory and I/O space summary (table), 4-4
  - memory map (figure), 4-7
  - memory space, 4-7
    - GP bus memory space, 4-9
    - integrated memory-mapped peripherals, 4-10
    - memory-mapped configuration region (MMCR) space, 4-9
    - PCI bus memory space, 4-9
    - SDRAM space, 4-8
  - operation, 4-3
  - PC/AT peripherals I/O map (table), 4-14
  - positive address decoding, 3-13, 4-9
  - positive address decoding (example), 3-14, 4-16
  - Programmable Address Region (PAR) registers, 4-5
    - PAR register format (figure), 4-6
  - registers, 4-2
  - software considerations, 4-18
  - write protection violation, 4-18, 4-20
  - write-protection violation, 4-15
- AEOI bit field, 15-18
- AINIT bit field, 14-14
- ALM_AM_PM bit field, 20-8
- ALM_INT_ENB bit field, 20-9
- ALT_CMP bit field, 17-4, 17-5
- AM_PM bit field, 20-8
- Am486® CPU
  - instruction set, xxiv
- Am5x86 CPU Control (CPUCTL) register
  - function, 5-6, 7-1, 24-2
  - usage, 5-7, 7-3–7-4
- Am5x86® CPU
  - block diagram (figure), 7-2
  - bus arbitration, 8-3
  - cache
    - behavior during clock speed changes, 7-4
    - configuration options (table), 7-4
    - flushing, 24-11
    - memory management, 7-4
    - performance considerations, 24-12
  - clocking considerations, 7-4, 8-7

CPU core identification, 3-7
CPU PLL stabilization time, 7-4
CPU speed, 3-7
documentation, xxiv
floating point unit (FPU), 7-3
initialization, 7-5
hard CPU reset, 7-5
soft CPU reset, 6-7, 7-5
interrupts, 7-5
latency, 7-5
operation, 7-3
registers, 7-1

AMDebug Disable signal. *See* AMDEBUG_DIS signal.

AMDebug Technology RX/TX Interrupt Mapping (ICEMAP) register
function, 15-5

AMDEBUG_DIS signal
description, 2-13
usage, 26-2

AMDebug™ technology
block diagram (figure), 26-2
on-chip trace cache, 26-7
operation, 26-6
signal descriptions, 2-12
software architecture (figure), 26-2
software performance profiling, 26-7
system design, 26-2
12-pin connector format (figure), 26-4
20-pin serial connector format (figure), 26-4
connecting the AMDebug port, 26-3
connector pins (table), 26-3
locating the target connector (figure), 26-6
target connector mechanical specifications, 26-5

Analog Ground signal. *See* GND_ANLG signal.

Analog Power Supply signal. *See* VCC_ANLG signal.

applications. *See* Élan™SC520 microcontroller.

Arbiter Priority Control (ARBPRICTL) register
function, 8-2
usage, 8-8, 8-20, 8-22

arbitration. *See* system arbitration.

ARBPRICTL register, 8-2

ATTR bit field, 3-10, 3-12, 4-5, 4-16

B

BA1–BA0 signals
control, 10-10, 10-19
description, 2-5

Backup Battery Sense signal. *See* BBATSEN signal.

Bank Address signals. *See* BA1–BA0 signals.

BBATSEN signal
description, 2-14
usage, 6-7, 20-3, 20-4, 20-7, 20-11

BI bit field, 21-8

BIOS. *See* system initialization.

bit fields
A10–A8, 15-18
A20G_CTL, 6-8
AEOI, 15-18
AINIT, 14-14
ALM_AM_PM, 20-8
ALM_INT_ENB, 20-9
ALT_CMP, 17-4, 17-5
AM_PM, 20-8
ATTR, 3-10, 3-12, 4-5, 4-16
BI, 21-8
BNKx_BNK_CNT, 10-15, 10-15
BNKx_COLWDTH, 10-15, 10-15
BSY, 22-7
BUS_MAS, 9-11
BUS_NUM, 9-10
BUS_PARK_SEL, 8-10, 8-22
CACHE_WR_MODE, 7-4, 8-6
CAS_LAT, 10-20, 10-31
CH3_ALT_SIZE, 14-21
CLK_INV_ENB, 22-5
CLK_PIN_DIR, 5-3, 5-9, 16-1, 16-6, 16-7
CLK_PIN_ENB, 5-9
CLK_SEL, 5-8, 22-7
CLK_SRC, 21-10
CLK_TST_SEL, 5-9
CNCR_MODE_ENB, 8-3, 8-22, 24-10
COMPTIM, 14-9
CONT_CMP, 17-4
CPU_PRI, 8-8
CPU_RST, 6-4, 6-7
CTR_MODE, 16-4
DCTS, 21-6
DDCD, 21-6
DDSR, 21-6
DEVICE_NUM, 9-10
DGP, 12-7
DR, 21-7
ECC_CHK_POS, 10-28
EMSI, 21-9
ENABLE, 4-12, 4-17, 9-10, 9-11
ENB, 19-3, 19-4, 19-6
ENH_MODE_ENB, 14-10
EPS, 21-8
ERR_IN_FIFO, 21-8
EXP_SEL, 5-8, 19-4
EXT_CLK, 17-3
FE, 21-8
FIFO_ENB, 21-9, 21-13
FIRST_DLY, 12-8
FUNCTION_NUM, 9-10
GNT_TO_ID, 8-23
GNT_TO_INT_ENB, 8-19, 8-23
GNT_TO_STA, 8-19
GP_ECHO_ENB, 24-10
GP_RST, 6-4, 6-7, 13-22

- bit fields (continued)
  - GPCSx_SEL, 12-3, 13-3, 16-1, 17-1
  - HI_PRI_0_SEL, 8-8
  - HI_PRI_1_SEL, 8-8
  - HOUR_MODE_SEL, 20-7
  - ICE_HRST_DET, 6-8
  - ICE_ON_RST, 6-5, 6-7, 24-11
  - ICE_SRST_DET, 6-8
  - INT_ENB, 17-4
  - IRQ_FLG, 19-4, 19-5
  - LOOP, 21-13
  - LTIM, 3-19
  - M_AD_IRQ_ID, 9-27
  - M_CMD_IRQ_ID, 9-27
  - M_GINT_MODE, 15-18
  - M_RETRY_TO, 9-12, 9-28
  - M_WPOST_ENB, 9-11
  - MATCH, 4-18
  - MBIT_ERR, 10-27
  - MEM_ENB, 9-11, 9-18
  - MODE, 12-7
  - MSBF_ENB, 22-5
  - MULT_INT_ENB, 10-27
  - NMI_DONE, 15-14
  - NMI_ENB, 6-7, 15-14, 15-20, 20-10
  - OE, 21-7, 21-8
  - OPMODE_SEL, 10-20, 10-30, 10-31, 10-32
  - OSC_CTL, 20-7, 20-10
  - OUT2, 21-13
  - PCI_RST, 6-4, 6-7, 9-29
  - PE, 21-8
  - PENB, 21-5, 21-8
  - PER_INT_ENB, 20-8
  - PER_INT_FLG, 20-8
  - PG_SZ, 3-9, 3-10, 4-5
  - PHS_INV_ENB, 22-5
  - PIOx_FNC, 13-3, 14-4, 15-2, 21-2, 23-3
  - PIT_GATE2, 16-4
  - PIT_OUT2_ENB, 16-4
  - PIT_OUT2_STA, 16-4
  - PRG_RST_ENB, 3-4, 6-4, 6-6, 10-29
  - PRGRST_DET, 6-8
  - PSC_SEL, 17-3
  - PWRGOOD_DET, 6-8
  - RAS_CAS_DLY, 10-21
  - RAS_PCHG_DLY, 10-21
  - RATE_SEL, 20-8
  - REGISTER_NUM, 9-10
  - RF_CLR, 21-10
  - RFRT, 21-7, 21-10
  - RFSH_ENB, 10-19
  - RFSH_SPD, 10-18
  - RTC_DIS, 3-21, 20-5
  - RTC_VRT, 6-7, 20-4
  - RTG, 17-3
  - S1_GINT_MODE, 15-18
  - S2, 15-17, 15-18, 15-19

- bit fields (continued)
  - S2_GINT_MODE, 3-19
  - S5, 15-17, 15-18, 15-19
  - SBIT_ERR, 10-27
  - SD_RST_DET, 6-8
  - SET, 20-7
  - SFNM, 15-18
  - SGL_INT_ENB, 10-27
  - SNGL, 15-18
  - SP, 21-8
  - SUB_DLY, 12-8
  - SYS_RST, 6-4, 6-5
  - SZ_ST_ADR, 3-9, 3-10, 4-5
  - T_DLYTR_ENB, 9-19
  - T_IRQ_ID, 9-27
  - T_PURGE_RD_ENB, 9-22
  - T7–T3, 15-18
  - TARGET, 3-10, 4-5, 4-8, 4-16, 4-19, 9-18
  - TC_INT, 22-7
  - TC_INT_ENB, 22-7
  - TEMT, 21-7
  - TERI, 21-6
  - TF_CLR, 21-10
  - THRE, 21-6
  - TRNMOD, 14-11, 14-19
  - UART1_DIS, 3-21
  - UART2_DIS, 3-21
  - UIP, 20-7
  - WB_ENB, 24-10
  - WB_FLUSH, 11-5, 11-13
  - WB_TST_ENB, 10-19, 24-2, 24-3, 24-7
  - WB_WM, 11-9
  - WDT_RST_DET, 6-8
  - WIDTH, 12-7
  - WRST_ENB, 19-4
  - WRTSEL, 14-9
  - XTAL_FREQ, 18-3
- BNKx_BNK_CNT bit field, 10-15, 10-15
- BNKx_COLWDTH bit field, 10-15, 10-15
- $\overline{\text{BOOTCS}}$ Control (BOOTCSCTL) register
  - function, 12-5
  - usage, 12-7, 12-14
- $\overline{\text{BOOTCS}}$ signal
  - configuration, 12-2, 12-14
  - control, 3-10
  - description, 2-6
  - usage, 3-5, 3-17, 4-4, 4-7, 4-8, 4-15, 4-21, 19-6, 24-6
- BOOTCSCTL register, 12-5
- bootstraps
  - signal descriptions, 2-13
- Boundary Scan (BSR) register
  - function, 25-2
  - usage, 25-3–25-5, 25-16–25-17

BR/TC signal
  control, 25-4
  description, 2-12
BSY bit field, 22-7
Buffer Chaining Control (GPDMABCCTL) register
  function, 14-5
  usage, 14-15
Buffer Chaining Interrupt Enable (GPDMABSINTENB) register
  function, 14-5
Buffer Chaining Status (GPDMABCSTA) register
  function, 14-5
Buffer Chaining Valid (GPDMABCVAL) register
  function, 14-5
  usage, 14-15
bus arbitration. *See* system arbitration.
Bus Grant signals. *See* $\overline{\text{GNT4}}$–$\overline{\text{GNT0}}$ signals.
Bus Request signals. *See* $\overline{\text{REQ4}}$–$\overline{\text{REQ0}}$ signals.
BUS_MAS bit field, 9-11
BUS_NUM bit field, 9-10
BUS_PARK_SEL bit field, 8-10, 8-22
Bypass (BPR) register
  function, 25-2
  usage, 25-4

C

cache. *See* Am5x86® CPU.
CACHE_WR_MODE bit field, 7-4, 8-6
cacheability control, 3-12
CAS_LAT bit field, 10-20, 10-31
CBAR register, 4-2
$\overline{\text{CBE3}}$–$\overline{\text{CBE0}}$ signals
  description, 2-6
  usage, 2-7
$\overline{\text{CF\_DRAM}}$ signal
  control, 24-2
  description, 2-12
  usage, 10-19, 24-1, 24-3, 24-12
$\overline{\text{CF\_ROM\_GPCS}}$ signal
  control, 24-2
  description, 2-12
  usage, 10-19, 24-1, 24-4, 24-5, 24-6, 24-12
CFG2–CFG0 signals
  description, 2-13
  usage, 2-6, 6-4, 6-6, 11-3, 12-2, 12-7, 12-14, 24-1
CFG3 signal
  description, 2-13
  usage, 16-1
CH3_ALT_SIZE bit field, 14-21
Chip Select Pin Function Select (CSPFS) register
  function, 12-5, 13-6, 16-2, 17-2, 23-4
  usage, 12-3, 13-3, 13-22, 16-1, 17-1, 23-5
chip selects. *See* $\overline{\text{GPCS7}}$–$\overline{\text{GPCS0}}$ signals.
Class Code/Revision ID (PCICCREVID) register
  function, 9-8
Clear To Send signals. *See* $\overline{\text{CTS2}}$–$\overline{\text{CTS1}}$ signals.
CLK_INV_ENB bit field, 22-5
CLK_PIN_DIR bit field, 5-3, 5-9, 16-1, 16-6, 16-7
CLK_PIN_ENB bit field, 5-9
CLK_SEL bit field, 5-8, 22-7
CLK_SRC bit field, 5-8, 21-10
CLK_TST_SEL bit field, 5-9
CLKMEMIN signal
  description, 2-5
  usage, 10-6, 10-7, 10-11
CLKMEMOUT signal
  description, 2-5
  usage, 10-6, 10-7, 10-11
CLKPCIIN signal
  description, 2-6
  usage, 9-5
CLKPCIOUT signal
  description, 2-7
  usage, 5-5, 9-2, 9-5, 9-6
CLKSEL register, 5-6, 23-4
CLKTEST signal
  control, 5-3, 5-6, 16-1, 16-2
  description, 2-10
  usage, 5-9
CLKTIMER signal
  control, 5-3, 5-6, 16-1, 16-2
  description, 2-10
  usage, 5-8, 5-9, 16-6, 16-7
clock generation and control
  block diagram (figure), 5-2
  CLKTEST clock routing (figure), 5-9
  CLKTEST signal, 5-9
  CLKTIMER signal, 5-9
  clock distribution (figure), 5-3
  clock sources (figure), 5-2
  clock startup and lock times (table), 5-2
  initialization, 5-9
  internal clocks, 5-7
    CPU, 5-7
    general-purpose (GP) bus, 5-7
    general-purpose (GP) timers, 5-8
    GP-DMA controller, 5-8
    PCI bus, 5-7
    programmable interval timer (PIT), 5-8
    real-time clock (RTC), 5-8
    ROM/Flash interface, 5-7
    SDRAM controller, 5-7
    software timer, 5-8
    synchronous serial interface (SSI), 5-8
    UART serial ports, 5-8
    watchdog timer, 5-8
  operation, 5-7

    PLL lock time (table), 5-2
    PLLs, 5-7
    registers, 5-6
    signal descriptions, 2-10
    system design, 5-3
        bypassing 32.768-kHz oscillator (figure), 5-5
        bypassing 33-MHz oscillator (figure), 5-6
        bypassing internal oscillators, 2-10, 5-5
        clock pin loading, 5-4
        crystal selection, 5-4
        shared signals (table), 5-3
    timing error and clock accuracy (table), 5-5
Clock Select (CLKSEL) register
    function, 5-6, 16-2, 21-3, 23-4
    usage, 5-3, 5-9, 16-1, 16-6, 16-7
CMDACK signal
    control, 25-4
    description, 2-12
CNCR_MODE_ENB bit field, 8-3, 8-22, 24-10
code execution control, 3-12
Code Fetch ROM/GPCS signal.
        See CF_ROM_GPCS signal.
Code Fetch SDRAM signal. *See* CF_DRAM signal.
Column Address Strobe signals.
        See SCASA–SCASB signals.
Command Acknowledge signal. *See* CMDACK signal.
Command or Byte-Enable Bus signals.
        See CBE3–CBE0 signals.
COMPTIM bit field, 14-9
configuration
    signal descriptions, 2-13
Configuration Base Address (CBAR) register
    function, 4-2
    usage, 3-4, 4-9, 4-11, 4-17, 4-18, 4-21
Configuration Input 3 signal. *See* CFG3 signal.
Configuration Input signals. *See* CFG2–CFG0 signals.
configuration RAM
    function, 20-7
CONT_CMP bit field, 17-4
CPU. *See* Am5x86® CPU.
CPU bus arbitration. *See* system arbitration.
CPU_PRI bit field, 8-8
CPU_RST bit field, 6-4, 6-7
CPUCTL register, 7-1
CSPFS register, 23-4
CTR_MODE bit field, 16-4
CTS2–CTS1 signals
    control, 13-6, 21-2, 21-3, 21-4
    description, 2-9, 21-6
    usage, 21-2, 21-9
customer service, iii

D

data buses
    boot device configuration, 12-7, 12-14
    CFG2 pinstrap, 2-13
    general-purpose (GP) bus data bus, 2-8
    loading, 10-9, 12-3, 13-4
    PCI data bus, 2-6
    ROM/Flash controller data bus, 2-6
    ROM/Flash controller data bus connection options
        (table), 12-1
    SDQMx signal behavior, 10-6
    SDRAM controller data bus, 2-5
    shared buses, 13-10
    voltage isolation, 12-3
Data Carrier Detect signals. *See* DCD2–DCD1 signals.
Data Input/Output Mask signals.
        See SDQM3–SDQM0 signals.
Data Set Ready signals. *See* DSR2–DSR1 signals.
data sheet, xxiv
Data Strobe signal. *See* DATASTRB signal.
Data Terminal Ready signals.
        See DTR2–DTR1 signals.
DATASTRB signal
    control, 24-2
    description, 2-12
    usage, 10-19, 24-1, 24-3, 24-4, 24-12
DBCTL register, 11-4
DCD2–DCD1 signals
    control, 13-6, 21-2, 21-3, 21-4
    description, 2-9, 21-6
    usage, 21-2, 21-9
DCTS bit field, 21-6
DDCD bit field, 21-6
DDSR bit field, 21-6
DEBUG_ENTER signal
    description, 2-13
    usage, 25-2, 26-2
debugging. *See* chip test and debugging.
        See also system test and debugging.
        See also AMDebug™ technology.
Device Identification (DID) register
    format, 25-14
    function, 25-2
    usage, 25-4, 25-14
Device Select signal. *See* DEVSEL signal.
Device/Vendor ID (PCIDEVID) register
    function, 9-8
DEVICE_NUM bit field, 9-10
DEVSEL signal
    control, 9-8
    description, 2-7
    timing, 9-19
    usage, 9-3, 9-19

DGP bit field, 12-7

DMA Buffer Chaining Interrupt Mapping (DMABCINTMAP) register
  function, 14-4, 15-5

DMA. *See* GP-DMA controller.

DMABCINTMAP register, 15-5

documentation, xxiv
  Élan™SC520 microcontroller documentation, xxiv
  literature ordering, iii, xxv
  world wide web site, iii, xxv

documentation conventions, xxv

Documentation Notation table, xxv

DR bit field, 21-7

DRAM. *See* SDRAM controller.

DRCBENDADR register, 10-10

DRCCFG register, 10-10

DRCCTL register, 10-10

DRCTMCTL register, 10-10

Drive Strength Control (DSCTL) register
  function, 10-10, 23-4
  usage, 10-19

DSCTL register, 23-4

$\overline{\text{DSR2}}$–$\overline{\text{DSR1}}$ signals
  control, 13-6, 21-2, 21-3, 21-4
  description, 2-9, 21-6
  usage, 21-2

$\overline{\text{DTR2}}$–$\overline{\text{DTR1}}$ signals
  control, 21-4
  description, 2-9, 21-6
  usage, 21-2, 21-9

E

ECC Check Bit Position (ECCCKBPOS) register
  function, 10-10
  usage, 10-28

ECC Check Code Test (ECCCKTEST) register
  function, 10-10, 24-2
  usage, 10-17, 24-11

ECC Control (ECCCTL) register
  function, 10-10
  usage, 10-27

ECC Interrupt Mapping (ECCMAP) register
  function, 10-10, 15-4

ECC Multi-Bit Error Address (ECCMBADD) register
  function, 10-10, 24-2
  usage, 10-28

ECC Single-Bit Error Address (ECCSBADD) register
  function, 10-10, 24-2
  usage, 10-28

ECC Status (ECCSTA) register
  function, 10-10
  usage, 10-27

ECC. *See* SDRAM controller.

ECC_CHK_POS bit field, 10-28

ECCCKBPOS register, 10-10

ECCCKTEST register, 10-10

ECCCTL register, 10-10

ECCMAP register, 15-4

ECCMBADD register, 10-10

ECCSBADD register, 10-10

ECCSTA register, 10-10

Élan™SC520 microcontroller
  applications, 1-8
    digital set top box, 1-9
    smart residential gateway, 1-8
    telephone line concentrator, 1-9
    thin client, 1-8
  architectural overview, 1-4
    address-mapping, 1-5
    AMDebug™ technology, 1-4
    clock generation, 1-6
    general-purpose (GP) bus interface, 1-6
    integrated peripherals, 1-7
    JTAG boundary scan test interface, 1-7
    PCI bus interface, 1-5
    ROM/Flash controller, 1-5
    SDRAM controller, 1-5
    system testing and debugging features, 1-8
    x86 architecture, 1-4
  block diagram, 1-2
  crystal specifications, xxiv
  customer support, iii
  distinctive characteristics, 1-1
  logic diagram by default pin function, 2-3
  logic diagram by interface, 2-2
  package dimensions, xxiv
  pin designations, xxiv
  register descriptions, xxiv
  thermal characteristics, xxiv
  timing, xxiv

ÉlanSC520 Microcontroller Revision ID (REVID) register
  function, 7-1
  usage, 25-14

EMSI bit field, 21-9

ENABLE bit field, 4-12, 4-17, 9-10, 9-11

ENB bit field, 19-3, 19-4, 19-6

ENH_MODE_ENB bit field, 14-10

Enter AMDebug Mode signal. *See* DEBUG_ENTER signal.

EPS bit field, 21-8

ERR_IN_FIFO bit field, 21-8

EXP_SEL bit field, 5-8, 19-4

EXT_CLK bit field, 17-3

external oscillator, 2-10, 5-5

F

- FE bit field, 21-8
- FERRMAP register, 15-5
- fields. *See* bit fields.
- FIFO_ENB bit field, 21-9, 21-13
- FIRST_DLY bit field, 12-8
- Flash memory. *See* ROM/Flash controller
- FLASHWR signal
  - description, 2-6
  - usage, 24-6
- Floating Point Error Interrupt Clear (FPUERRCLR) register
  - function, 7-1, 15-7
  - usage, 15-12
- Floating Point Error Interrupt Mapping (FERRMAP) register
  - function, 7-1, 15-5
- floating point unit (FPU), 7-3
  - error handling, 15-12, 15-19
  - system reset, 7-3
- FNINIT instruction, 7-3
- FPUERRCLR register, 15-7
- FRAME signal
  - description, 2-7
  - usage, 9-3, 9-9
- FUNCTION_NUM bit field, 9-10

G

- general-purpose (GP) bus
  - block diagram (figure), 13-2
  - bus cycles, 13-16
    - 8-bit access of 16-bit I/O device (figure), 13-19
    - 8-bit access of 8-bit I/O device (figure), 13-16
    - 16-bit access of 16-bit I/O device (figure), 13-17
    - 16-bit access of 8-bit I/O device (figure), 13-17
    - 32-bit access of 16-bit I/O device (figure), 13-18
    - 32-bit access of 8-bit I/O device (figure), 13-18
    - differentiating byte accesses of 16-bit devices (table), 13-19
    - GPIOCS16 and GPMEMCS16 timing, 13-19
    - GPRDY timing (figure), 13-21
    - wait states, 13-20
  - bus sizing, 13-19
    - dynamic bus sizing override (table), 13-20
  - chip select qualification, 13-9
  - configuration, 4-16
  - configuring external GP bus devices, 3-7, 3-13
    - multiple devices on one chip select, 3-14
    - single device performing its own decode, 3-14
    - single device using one chip select, 3-14
  - configuring Programmable Address Region x (PARx) registers, 3-8
  - data sizing, 13-9
  - DMA interface, 13-11
  - echo mode, 13-8, 13-10
    - echo mode minimum timing (table), 13-9
  - GP bus reset, 6-7
  - I/O space, 4-15
  - I/O-mapped device support, 13-9
  - initialization, 13-22
  - interrupts, 13-21
  - ISA bus compatibility, 13-11
  - ISA signals and GP bus signals (table), 13-12
  - latency, 13-21
    - 8/16-bit GP bus width, 13-21
    - noncacheable GP bus, 13-21
    - slow GP bus cycles, 13-21
  - memory space, 4-9
  - memory-mapped device support, 13-9
  - operation, 13-6
  - overview, 13-1
  - programmable timing, 13-7
    - GPRDY, 13-8
    - programmable timing format (figure), 13-8
    - timing requirements, 13-7
  - registers, 13-5
  - serial communications controller interface, 13-14
    - Am85C30 interface (figure), 13-15
    - Am85C30 interface timing (figure), 13-16
  - sharing address and data bus with ROM/Flash, 13-10
  - signal descriptions, 2-7, 2-11
  - Super I/O controller interface, 13-13
    - Super I/O controller interface (figure), 13-13
    - Super I/O interface timing (figure), 13-14
  - system design, 13-1
    - external data buffer (figure), 13-4
    - loading, 13-4
    - shared signals (table), 13-3
    - voltage translation, 13-4
    - voltage translation example (figure), 13-5
  - unaligned accesses, 13-9
  - usage scenarios, 13-11
- general-purpose (GP) timers
  - block diagram (figure), 17-2
  - cascaded 32-bit timer, 17-6
  - clocking considerations, 17-5
    - external clock sources (table), 17-6
    - internal clock sources (table), 17-5
  - configuration, 17-5
  - GP Timer 0, 17-3
  - GP Timer 1, 17-3
  - GP Timer 2, 17-4
  - initialization, 17-8
  - interrupts, 17-6
  - operating modes, 17-4
    - alternate compare mode, 17-4
    - continuous mode, 17-4
    - hardware retrigger mode, 17-4
    - interrupt on terminal count mode, 17-4

prescaler mode, 17-4
square wave mode, 17-4
operation, 17-3
registers, 17-2
signal descriptions, 2-10
software considerations, 17-6
combining GP Timer Count elements, 17-6
reading the cascaded 32-bit timer, 17-6
system design, 17-1
shared signals (table), 17-1

General-Purpose Address Bus signals. *See* GPA25–GPA0 signals.

General-Purpose Chip Select signals. *See* GPCS7–GPCS0 signals.

General-Purpose CMOS RAM (RTCCMOS) register
function, 20-7

General-Purpose Data Bus signals. *See* GPD15–GPD0 signals.

GND signals
description, 2-14

GND_ANLG signal
description, 2-14

GNT_TO_ID bit field, 8-23

GNT_TO_INT_ENB bit field, 8-19, 8-23

GNT_TO_STA bit field, 8-19

GNT4–GNT0 signals
description, 2-7
usage, 8-7

GP ALE Offset (GPALEOFF) register
function, 13-6
usage, 13-8

GP ALE Pulse Width (GPALEW) register
function, 13-6
usage, 13-8

GP Bus Address Enable signal. *See* GPAEN signal.

GP Bus Address Latch Enable signal. *See* GPALE signal.

GP Bus Byte High Enable signal. *See* GPBHE signal.

GP Bus Data Bus Buffer Output Enable signal. *See* GPDBUFOE signal.

GP Bus DMA Acknowledge signals. *See* GPDACK3–GPDACK0 signals.

GP Bus DMA Request signals. *See* GPDRQ3–GPDRQ0 signals.

GP Bus I/O Chip-Select 16 signal. *See* GPIOCS16 signal.

GP Bus I/O Read signal. *See* GPIORD signal.

GP Bus I/O Write signal. *See* GPIOWR signal.

GP Bus Interrupt Request signals. *See* GPIRQ10–GPIRQ0 signals.

GP Bus Memory Chip-Select 16 signal. *See* GPMEMCS16 signal.

GP Bus Memory Read signal. *See* GPMEMRD signal.

GP Bus Memory Write signal. *See* GPMEMWR signal.

GP Bus Ready signal. *See* GPRDY signal.

GP Bus Reset signal. *See* GPRESET signal.

GP Bus Terminal Count signal. *See* GPTC signal.

GP Chip Select Data Width (GPCSDW) register
function, 13-5
usage, 13-9, 13-19, 13-20, 13-22

GP Chip Select Offset (GPCSOFF) register
function, 13-5
usage, 13-8

GP Chip Select Pulse Width (GPCSPW) register
function, 13-5
usage, 13-8

GP Chip Select Qualification (GPCSQUAL) register
function, 13-5
usage, 13-9, 13-22

GP Chip Select Recovery Time (GPCSRT) register
function, 13-5
usage, 13-8

GP Echo Mode (GPECHO) register
function, 13-5, 24-2
usage, 13-22, 24-10

GP Read Offset (GPRDOFF) register
function, 13-5
usage, 13-8

GP Read Pulse Width (GPRDW) register
function, 13-5
usage, 13-8

GP Timer x Count (GPTMRxCNT) register
function, 17-2, 17-3
usage, 17-4

GP Timer x Interrupt Mapping (GPTMRxMAP) register
function, 15-4, 17-3

GP Timer x Maxcount Compare A (GPTMRxMAXCMPA) register
function, 17-2, 17-3
usage, 17-4, 17-5, 17-6, 17-7, 17-8

GP Timer x Maxcount Compare B (GPTMRxMAXCMPB) register
function, 17-2, 17-3
usage, 17-4, 17-5, 17-6, 17-8

GP Timer x Mode/Control (GPTMRxCTL) register
function, 5-6, 17-2, 17-3
usage, 5-8, 17-3, 17-4, 17-5, 17-6, 17-8

GP Timers Status (GPTMRSTA) register
function, 17-2
usage, 17-6

GP timers. *See* general-purpose (GP) timers.

GP Write Offset (GPWROFF) register
function, 13-6
usage, 13-8

GP Write Pulse Width (GPWRW) register
function, 13-5
usage, 13-8

GP_ECHO_ENB bit field, 24-10

GP_RST bit field, 6-4, 6-7, 13-22

GPA25–GPA0 signals
  description, 2-6, 2-7
  usage, 2-8, 13-1, 13-4, 13-10, 24-5, 24-6

GPAEN signal
  control, 13-3, 13-6, 14-4
  description, 2-8
  usage, 13-10, 14-9, 14-17, 24-6

GPALE signal
  control, 13-3, 13-6
  description, 2-8
  usage, 13-7, 24-6

GPALEOFF register, 13-6

GPALEW register, 13-6

$\overline{\text{GPBHE}}$ signal
  control, 13-3, 13-6
  description, 2-8
  usage, 13-19

$\overline{\text{GPCS7}}$–$\overline{\text{GPCS0}}$ signals
  configuration, 3-13, 4-5, 4-9, 4-16, 4-20
  control, 3-10, 12-3, 13-3, 13-5, 13-6, 16-2, 17-1
  description, 2-11
  usage, 3-10, 12-3, 13-7, 24-4

GPCSDW register, 13-5

GPCSOFF register, 13-5

GPCSPW register, 13-5

GPCSQUAL register, 13-5

GPCSRT register, 13-5

GPCSx_SEL bit field, 12-3, 13-3, 16-1, 17-1

GPD15–GPD0 signals
  description, 2-8
  usage, 12-9, 13-10, 14-9, 24-6, 24-10

$\overline{\text{GPDACK3}}$–$\overline{\text{GPDACK0}}$ signals
  control, 13-3, 13-6, 14-4
  description, 2-8
  usage, 14-3

$\overline{\text{GPDBUFOE}}$ signal
  control, 13-3, 13-6
  description, 2-8
  usage, 13-4

GP-DMA Channel x Extended Page (GPDMAEXTPGx) register
  function, 14-4
  usage, 14-11, 14-12

GP-DMA Channel x Extended Transfer Count (GPDMAEXTTCx) register
  function, 14-5

GP-DMA Channel x Next Address High (GPDMANXTADDHx) register
  function, 14-5, 14-6
  usage, 14-15, 14-18

GP-DMA Channel x Next Address Low (GPDMANXTADDLx) register
  function, 14-5, 14-6
  usage, 14-15, 14-18

GP-DMA Channel x Next Transfer Count High (GPDMANXTTCHx) register
  function, 14-6
  usage, 14-15, 14-18

GP-DMA Channel x Next Transfer Count Low (GPDMANXTTCLx) register
  function, 14-6
  usage, 14-15, 14-18

GP-DMA Control (GPDMACTL) register
  function, 5-6, 14-4
  usage, 5-8, 14-10, 14-18, 14-20

GP-DMA controller
  addressing GP-DMA channels, 14-11
    16-bit channel address generation (table), 14-12
    8-bit channel address generation (table), 14-12
    enhanced GP-DMA mode, 14-12
    normal GP-DMA mode, 14-11
  block diagram (figure), 14-2
  bus cycles, 14-16
    cycle types (table), 14-16
    GP bus I/O to SDRAM, 14-16
    read in demand transfer mode (figure), 14-16
    read transfer (figure), 14-13
    read transfer with cache hit (figure), 14-17
    verify transfer (figure), 14-14
    write transfer (figure), 14-14
  channel mapping, 14-10
  channel mapping (table), 14-10
  clocking considerations, 14-18
  example configurations, 14-19
    16-bit channel in enhanced mode, 14-21
    16-bit channel in normal mode, 14-20
    8-bit channel in enhanced mode, 14-20
    8-bit channel in normal mode, 14-19
  fly-by transfers, 14-8
  GP bus echo mode, 14-17
  GP bus timing, 13-11
  GP-DMA initiators, 14-9
    external I/O devices, 14-9
    external memory-mapped I/O devices, 14-10
    internal UARTs, 14-9
  GP-DMA transfer modes
    automatic initialization control, 14-14
    block transfer mode, 14-13
    buffer chaining, 14-15
    demand transfer mode, 14-12
    priority, 14-15
    single transfer mode, 14-12
    transfer types, 14-13
  GP-DMA transfers, 14-8
  initialization, 14-19
  initiator, 14-8
  initiator/target combinations supported (table), 14-9

  - interrupts, 14-18
  - latency, 14-18
    - nonpreemptive latency, 14-18
    - preemptive latency, 14-19
  - master and slave core cascading (figure), 14-3
  - operating modes, 14-10
    - enhanced GP-DMA mode, 14-11
    - normal GP-DMA mode, 14-10
  - operation, 14-8
  - overview, 14-1
  - PCI considerations, 14-9
  - peer-to-peer transfers, 14-9
  - registers, 14-4
  - signal descriptions, 2-7
  - software considerations, 14-18
  - system design, 14-3
    - shared signals (table), 14-4
  - target, 14-8
  - transfer modes, 14-12
- GP-DMA Memory-Mapped I/O (GPDMAMMIO) register
  - function, 14-4
  - usage, 14-10
- GP-DMA Resource Channel Map A (GPDMAEXTCHMAPA) register
  - function, 14-4
  - usage, 14-10
- GP-DMA Resource Channel Map B (GPDMAEXTCHMAPB) register
  - function, 14-4
  - usage, 14-10
- GPDMABCCTL register, 14-5
- GPDMABCSTA register, 14-5
- GPDMABCVAL register, 14-5
- GPDMABSINTENB register, 14-5
- GPDMACTL register, 14-4
- GPDMAEXTCHMAPA register, 14-4
- GPDMAEXTCHMAPB register, 14-4
- GPDMAEXTPGx register, 14-4, 14-5
- GPDMAEXTTCx register, 14-5
- GPDMAMMIO register, 14-4
- GPDMANXTADDHx register, 14-5, 14-6
- GPDMANXTADDLx register, 14-5, 14-6
- GPDMANXTTCHx register, 14-6
- GPDMANXTTCLx register, 14-5, 14-6
- GPDMAxMAR register, 14-7
- GPDMAxPG register, 14-7
- GPDMAxTC register, 14-7
- GPDRQ3–GPDRQ0 signals
  - control, 13-3, 13-6, 14-4
  - description, 2-8
  - usage, 14-3, 14-10
- GPECHO register, 13-5
- GPIOCS16 signal
  - control, 13-3, 13-6
  - description, 2-8
  - timing, 13-19
  - usage, 13-4, 13-10, 13-11, 24-6
- GPIORD signal
  - control, 13-5
  - description, 2-8
  - usage, 13-4, 13-7, 13-9, 13-11, 13-20, 14-9, 14-17, 24-6
- GPIOWR signal
  - control, 13-5
  - description, 2-8
  - usage, 13-7, 13-9, 13-11, 13-20, 14-9, 24-6
- GPIRQ10–GPIRQ0 signals
  - control, 9-7, 13-3, 13-6, 15-2, 15-4
  - description, 2-9
  - usage, 2-7, 9-2, 13-21, 15-2, 15-5, 15-8, 23-3
- GPIRQx Interrupt Mapping (GPxIMAP) register
  - function, 15-5
- GPMEMCS16 signal
  - control, 13-3, 13-6
  - description, 2-9
  - timing, 13-19
  - usage, 13-4, 13-10, 13-11, 24-6
- GPMEMRD signal
  - control, 13-5
  - description, 2-9
  - usage, 13-4, 13-7, 13-9, 13-11, 13-20, 14-10, 14-17, 24-4, 24-6
- GPMEMWR signal
  - control, 13-5
  - description, 2-9
  - usage, 13-7, 13-9, 13-11, 13-20, 14-10, 14-17, 24-6
- GPRDOFF register, 13-5
- GPRDW register, 13-5
- GPRDY signal
  - control, 13-3, 13-6
  - description, 2-9
  - timing, 13-20
  - usage, 13-4, 13-8, 13-11, 13-13, 13-20, 24-6
- GPRESET signal
  - control, 6-3, 13-6, 13-22
  - description, 2-9
  - usage, 6-4, 6-7
- GPTC signal
  - control, 13-3, 13-6, 14-4
  - description, 2-9
  - usage, 14-3, 14-9, 14-12, 14-18
- GPTMRSTA register, 17-2
- GPTMRxCNT register, 17-2, 17-3
- GPTMRxCTL register, 17-2, 17-3
- GPTMRxMAP register, 15-4
- GPTMRxMAXCMPA register, 17-2, 17-3

GPTMRxMAXCMPB register, 17-2, 17-3

GPWROFF register, 13-6

GPWRW register, 13-5

GPxIMAP register, 15-5

Ground signal. *See* GND signal.

H

HBCTL register, 9-7

HBMSTIRQCTL register, 9-7

HBMSTIRQSTA register, 9-7

HBTGTIRQCTL register, 9-7

HBTGTIRQSTA register, 9-7

Header Type (PCIHEADTYPE) register
  function, 9-8

HI_PRI_0_SEL bit field, 8-8

HI_PRI_1_SEL bit field, 8-8

Host Bridge Control (HBCTL) register
  function, 6-3, 9-7
  usage, 6-7, 9-11, 9-19, 9-22, 9-29

Host Bridge Master Interrupt Address (MSTINTADD) register
  function, 9-7
  usage, 9-12, 9-27

Host Bridge Master Interrupt Control (HBMSTIRQCTL) register
  function, 9-7
  usage, 8-19

Host Bridge Master Interrupt Status (HBMSTIRQSTA) register
  function, 9-7
  usage, 9-27

Host Bridge Target Interrupt Control (HBTGTIRQCTL) register
  function, 9-7
  usage, 8-19

Host Bridge Target Interrupt Status (HBTGTIRQSTA) register
  function, 9-7
  usage, 9-27

HOUR_MODE_SEL bit field, 20-7

I

I/O map. *See* address mapping.

ICE_HRST_DET bit field, 6-8

ICE_ON_RST bit field, 6-5, 6-7, 24-11

ICE_SRST_DET bit field, 6-8

ICEMAP register, 15-5

initialization
  See also system initialization.
  address mapping, 4-21
  Am5x86® CPU, 7-5
  clocks, 5-9
  general-purpose (GP) bus, 13-22
  general-purpose (GP) timers, 17-8
  GP-DMA controller, 14-19
  JTAG test access port (TAP) controller, 25-20
  PCI host bridge, 9-29
  power-on reset, 6-9
  programmable input/output (PIO), 23-6
  programmable interrupt controller (PIC), 15-20
  programmable interval timer (PIT), 16-7
  read buffer, 11-15
  real-time clock (RTC), 20-10
  reset types, 6-3
  ROM/Flash controller, 12-14
  SDRAM controller, 10-29
  software timer, 18-3
  synchronous serial interface (SSI), 22-8
  system arbiter, 8-22
  system reset, 6-4
  UART serial ports, 21-13
  watchdog timer (WDT), 19-6
  write buffer, 11-15

Initiator Ready signal. *See* $\overline{\text{IRDY}}$ signal.

INST_TRCE signal
  description, 2-13
  usage, 26-2

Instruction (IR) register
  function, 25-2
  usage, 25-3, 25-3, 25-4–25-5, 25-15, 25-17, 25-20

instruction set manual, xxiv

Instruction Trace signal. *See* INST_TRCE signal.

INT_ENB bit field, 17-4

$\overline{\text{INTA}}$–$\overline{\text{INTD}}$ signals
  control, 9-7
  description, 2-7, 15-4
  usage, 9-2, 15-2, 15-8

Interrupt Control (PICICR) register
  function, 15-4
  usage, 3-19, 7-6, 15-14, 15-18, 20-10

Interrupt Pin Polarity (INTPINPOL) register
  function, 9-7, 15-4
  usage, 3-20

Interrupt Request signals. *See* $\overline{\text{INTA}}$–$\overline{\text{INTD}}$ signals.

interrupts. *See* programmable interrupt controller (PIC).

INTPINPOL register, 15-4

$\overline{\text{IRDY}}$ signal
  description, 2-7
  usage, 9-3

IRQ_FLG bit field, 19-4, 19-5

ISA bus compatibility
  ISA features not supported, 13-11
  ISA signals and GP bus signals (table), 13-12

J

- JTAG boundary scan test interface
  - block diagram (figure), 25-1
  - board continuity testing, 25-2
  - Boundary Scan register (figure), 25-1
  - bus cycles, 25-19
    - data scan (figure), 25-19
    - instruction scan (figure), 25-20
  - clocking considerations, 25-20
  - configuration information, 25-5
    - bypass path, 25-5
    - instruction path, 25-5
    - main data scan path, 25-5
  - Device Identification register (figure), 25-14
  - initialization, 25-20
  - instruction register, 25-3
    - BYPASS instruction, 25-4
    - EXTEST instruction, 25-3
    - HIGHZ instruction, 25-4
    - IDCODE instruction, 25-4
    - SAMPLE/PRELOAD instruction, 25-4
  - main data scan path (table), 25-5
  - operation, 25-2
  - overview, 25-1
  - registers, 25-2
  - Serial Debug Port Data register (figure), 25-14
  - signal descriptions, 2-12
  - TAP controller state diagram (figure), 25-15
  - TAP instruction set (table), 25-3
  - test access port (TAP) controller, 25-15
    - capture-DR state, 25-16
    - capture-IR state, 25-17
    - exit1-DR state, 25-16
    - exit1-IR state, 25-18
    - exit2-DR state, 25-17
    - exit2-IR state, 25-18
    - pause-DR state, 25-16
    - pause-IR state, 25-18
    - run-test-idle state, 25-16
    - select-DR-scan state, 25-16
    - select-IR-scan state, 25-17
    - shift-DR state, 25-16
    - shift-IR state, 25-17
    - test-logic-reset state, 25-15
    - update-DR state, 25-17
    - update-IR state, 25-18
- JTAG_TCK signal
  - description, 2-12
  - usage, 25-2, 25-4, 26-3
- JTAG_TDI signal
  - description, 2-12
  - usage, 25-2, 25-4, 25-5, 26-3
- JTAG_TDO signal
  - description, 2-12
  - usage, 25-2, 25-3, 25-4, 25-5, 26-3
- JTAG_TMS signal
  - usage, 2-12, 25-15, 26-3
- JTAG_TRST signal
  - description, 2-12
  - usage, 25-15

L

- LF_PLL1 signal
  - description, 2-10
  - usage, 5-2
- logic diagram
  - default pin function, 2-3
  - interface, 2-2
- LOOP bit field, 21-13
- Loop Filter Interface signal. *See* LF_PLL1 signal.
- LTIM bit field, 3-19

M

- M_AD_IRQ_ID bit field, 9-27
- M_CMD_IRQ_ID bit field, 9-27
- M_GINT_MODE bit field, 15-18
- M_RETRY_TO bit field, 9-12, 9-28
- M_WPOST_ENB bit field, 9-11
- MA12–MA0 signals
  - control, 10-10, 10-19
  - description, 2-5
  - usage, 10-31, 24-4, 24-7, 24-9
- Master DMA Channel 4–7 Control (MSTDMACTL) register
  - function, 14-7
- Master DMA Channel 4–7 Mask (MSTDMAMSK) register
  - function, 14-7
- Master DMA Channel 4–7 Mode (MSTDMAMODE) register
  - function, 14-7
  - usage, 14-11, 14-14, 14-19
- Master DMA Channel 4–7 Status (MSTDMASTA) register
  - function, 14-7
- Master DMA Channel x Memory Address (GPDMAxMAR) register
  - function, 14-7
  - usage, 14-11, 14-12, 14-18
- Master DMA Channel x Page (GPDMAxPG) register
  - function, 14-7
  - usage, 14-11, 14-12
- Master DMA Channel x Transfer Count (GPDMAxTC) register
  - function, 14-7
  - usage, 14-9, 14-18

Master DMA Clear Byte Pointer (MSTDMACBP) register
- function, 14-7

Master DMA Controller Reset (MSTDMARST) register
- function, 14-8
- usage, 14-19

Master DMA Controller Temporary (MSTDMATMP) register
- function, 14-8

Master DMA General Mask (MSTDMAGENMSK) register
- function, 14-8
- usage, 14-20

Master DMA Mask Reset (MSTDMAMSKRST) register
- function, 14-8

Master PIC Initialization Control Word 1 (MPICICW1) register
- function, 15-6
- usage, 15-16, 15-18

Master PIC Initialization Control Word 2 (MPICICW2) register
- function, 15-6
- usage, 15-16, 15-18

Master PIC Initialization Control Word 3 (MPICICW3) register
- function, 15-7
- usage, 15-16, 15-17, 15-18, 15-19

Master PIC Initialization Control Word 4 (MPICICW4) register
- function, 15-7
- usage, 15-16, 15-17, 15-18

Master PIC In-Service (MPICISR) register
- function, 15-6
- usage, 15-17

Master PIC Interrupt Mask (MPICINTMSK) register
- function, 15-7
- usage, 15-17

Master PIC Interrupt Mode (MPICMODE) register
- function, 15-4

Master PIC Interrupt Request (MPICIR) register
- function, 15-6

Master PIC Operation Control Word 2 (MPICOCW2) register
- function, 15-6
- usage, 15-17

Master PIC Operation Control Word 3 (MPICOCW3) register
- function, 15-6
- usage, 15-17

Master Retry Time-Out (PCIMRETRYTO) register
- function, 9-8
- usage, 9-12, 9-28

Master Software DRQ(n) Request (MSTDMASWREQ) register
- function, 14-7

MATCH bit field, 4-18

MBIT_ERR bit field, 10-27

MD31–MD0 signals
- control, 10-10, 10-19, 23-4
- description, 2-5, 2-6
- usage, 10-6, 10-9, 24-4, 24-6

MECC6–MECC0 signals
- control, 10-10, 10-19, 23-4
- description, 2-5
- usage, 10-9

MEM_ENB bit field, 9-11, 9-18

Memory Data Bus signals. *See* MD31–MD0 signals.

Memory Error Correction Code signals. *See* MECC6–MECC0 signals.

memory map. *See* address mapping.

memory-mapped configuration region, 4-1, 4-9
- memory space, 4-4
- priority, 4-18
- relocating, 4-11, 4-18

memory-mapped configuration region (MMCR)
- integrated memory-mapped peripherals, 4-10

MMCR. *See* memory-mapped configuration region. *See also* address mapping.

MODE bit field, 12-7

MPICICW1 register, 15-6

MPICICW2 register, 15-6

MPICICW3 register, 15-7

MPICICW4 register, 15-7

MPICINTMSK register, 15-7

MPICIR register, 15-6

MPICISR register, 15-6

MPICMODE register, 15-4

MPICOCW2 register, 15-6

MPICOCW3 register, 15-6

MSBF_ENB bit field, 22-5

MSTDMACBP register, 14-7

MSTDMACTL register, 14-7

MSTDMAGENMSK register, 14-8

MSTDMAMODE register, 14-7

MSTDMAMSK register, 14-7

MSTDMAMSKRST register, 14-8

MSTDMARST register, 14-8

MSTDMASTA register, 14-7

MSTDMASWREQ register, 14-7

MSTDMATMP register, 14-8

MSTINTADD register, 9-7

MULT_INT_ENB bit field, 10-27

N

- NMI. *See* programmable interrupt controller (PIC).
- NMI_DONE bit field, 15-14
- NMI_ENB bit field, 6-7, 15-14, 15-20, 20-10

O

- OE bit field, 21-7, 21-8
- OPMODE_SEL bit field, 10-20, 10-30, 10-31, 10-32
- OSC_CTL bit field, 20-7, 20-10
- OUT2 bit field, 21-13

P

- PAR signal
  - description, 2-7
- Parity Error signal. *See* $\overline{\text{PERR}}$ signal.
- PARx register, 4-2
- PC/AT compatibility, 1-7
  - a20 gate support, 6-8
  - address mapping, 4-13
  - general-purpose (GP) bus configuration, 13-6
  - GP-DMA transfers, 14-1, 14-8
  - interrupt channel mapping (table), 15-12
  - ISA devices, 13-11
  - ISA signals and GP bus signals (table), 13-12
  - normal GP-DMA mode, 14-11
  - PC/AT peripherals I/O map (table), 4-14
  - PC/AT port logic, 6-8, 16-4
  - programmable interrupt controller (PIC) configuration, 15-18
  - programmable interval timer (PIT) clock source, 16-6
  - real-time clock (RTC), 20-5
  - Windows® compatibility, 4-17
- PC/AT port logic
  - a20 gate support, 6-8
  - SCP Command Port register (Port 0064h), 6-8
  - SCP Data Port register (Port 0060h), 6-8
  - System Control Port A register (Port 0092h), 6-8
  - System Control Port B register (Port 0061h), 16-4
- PCI Address Data Bus signals. *See* AD31–AD0 signals.
- PCI Bus Arbiter Status (PCIARBSTA) register
  - function, 8-2
  - usage, 8-10, 8-19, 8-23
- PCI bus arbitration. *See* system arbitration.
- PCI Bus Clock Input signal. *See* CLKPCIIN signal.
- PCI Bus Clock Output signal. *See* CLKPCIOUT signal.
- PCI bus. *See* PCI host bridge.
- PCI Configuration Address (PCICFGADR) register
  - format (figure), 9-10
  - function, 9-8
  - usage, 4-12, 9-9, 9-10, 9-17
- PCI Configuration Data (PCICFGDATA) register
  - function, 9-8
  - usage, 4-12, 9-9, 9-10, 9-11, 9-17
- PCI host bridge
  - arbitration, 8-3
  - block diagram (figure), 9-2
  - broken transactions, 8-19
  - bus arbitration, 8-3
  - configuration, 9-9
    - generating configuration cycles, 9-10
  - configuration space, 4-12
  - configuring PCI bus devices
    - network adapter, 3-16
    - VGA controller on PCI bus, 3-15
  - host bridge as PCI bus master, 9-11
    - bus cycles, 9-12
      - configuration read/write (figure), 9-17
      - CPU non-posted write cycle (figure), 9-16
      - CPU posted write cycle (figure), 9-15
      - CPU read cycle (figure), 9-12
      - CPU read with external target retry (figure), 9-14
    - delayed transaction support, 9-12
    - read cycles, 9-12
    - write posting, 9-11
  - host bridge as PCI bus target, 9-18
    - address FIFO, 9-20
    - burst ordering, 9-21
    - bus cycles, 9-22
      - external master SDRAM read (figure), 9-24
      - external master SDRAM write (figure), 9-22
      - target disconnect (figure), 9-25
    - command support, 9-19
    - data coherency, 9-21
    - delayed transaction support, 9-19
    - $\overline{\text{DEVSEL}}$ timing, 9-19
    - FIFOs and prefetching, 9-20
    - target address space, 9-18
  - I/O space, 4-4, 4-12
  - initialization, 9-29
  - interrupts, 8-19, 9-27
    - delayed transaction time-out, 9-27
    - master abort, 9-27
    - parity errors, 9-27
    - retry time-out counter expired, 9-27
    - system error, 9-27
    - target abort, 9-27
  - latency, 9-28
    - master latency, 9-28
    - target latency, 9-28
  - memory space, 4-4, 4-9
  - operation, 9-8
  - PCI bus arbiter
    - bus parking, 8-7
  - PCI reset, 6-7
  - registers, 9-7
  - SDRAM read buffer, 11-12

SDRAM write buffer, 11-12
signal descriptions, 2-6
system design, 9-2
clocking in heavily loaded system (figure), 9-6
clocking in lightly loaded system (figure), 9-6
crystal selection implications, 9-6
external PCI bus master connection (figure), 9-4
external PCI bus target connection (figure), 9-3
PCI clocking, 9-5
$\overline{\text{SERR}}$ and $\overline{\text{PERR}}$ connection (figure), 9-5
unsupported configuration registers, 9-9
unsupported functions, 9-8
PCI Host Bridge Interrupt Mapping (PCIHOSTMAP) register
function, 8-3, 9-7, 15-4
usage, 8-19
PCI Interrupt A Mapping (PCIINTAMAP) register
function, 9-7, 15-5
PCI Interrupt B Mapping (PCIINTBMAP) register
function, 9-7, 15-5
PCI Interrupt C Mapping (PCIINTCMAP) register
function, 9-7, 15-5
PCI Interrupt D Mapping (PCIINTDMAP) register
function, 9-7, 15-5
PCI Parity signal. *See* PAR signal.
PCI_RST bit field, 6-4, 6-7, 9-29
PCIARBSTA register, 8-2
PCICCREVID register, 9-8
PCICFGADR register, 9-8
PCICFGDATA register, 9-8
PCIDEVID register, 9-8
PCIHEADTYPE register, 9-8
PCIHOSTMAP register, 15-4
PCIINTAMAP register, 15-5
PCIINTBMAP register, 15-5
PCIINTCMAP register, 15-5
PCIINTDMAP register, 15-5
PCIMRETRYTO register, 9-8
PCISTACMD register, 9-8
PE bit field, 21-8
PENB bit field, 21-5, 21-8
PER_INT_ENB bit field, 20-8
PER_INT_FLG bit field, 20-8
$\overline{\text{PERR}}$ signal
description, 2-7
usage, 9-3
PG_SZ bit field, 3-9, 3-10, 4-5
PHS_INV_ENB bit field, 22-5
PIC. *See* programmable interrupt controller (PIC).
PICICR register, 15-4
pins. *See* signals.
pinstraps
signal descriptions, 2-13
PIO functions. *See* programmable input/output (PIO).
PIO15–PIO0 Clear (PIOCLR15_0) register
function, 23-4
usage, 23-5
PIO15–PIO0 Data (PIODATA15_0) register
function, 23-4
usage, 23-5
PIO15–PIO0 Direction (PIODIR15_0) register
function, 23-4
usage, 23-5
PIO15–PIO0 Pin Function Select (PIOPFS15_0) register
function, 13-6, 14-4, 15-4, 23-4
usage, 3-8, 13-3, 14-4, 15-2, 23-3, 23-5
PIO15–PIO0 Set (PIOSET15_0) register
function, 23-4
usage, 23-5
PIO31–PIO0 signals
control, 13-3, 14-4, 15-2, 21-2, 23-3
usage, 23-6
PIO31–PIO16 Clear (PIOCLR31_16) register
function, 23-4
usage, 23-5
PIO31–PIO16 Data (PIODATA31_16) register
function, 23-4
usage, 23-5
PIO31–PIO16 Direction (PIODIR31_16) register
function, 23-4
usage, 23-5
PIO31–PIO16 Pin Function Select (PIOPFS31_16) register
function, 13-6, 15-4, 21-3, 23-4
usage, 3-8, 13-3, 15-2, 21-2, 23-3, 23-5
PIO31–PIO16 Set (PIOSET31_16) register
function, 23-4
usage, 23-5
PIOCLR15_0 register, 23-4
PIOCLR31_16 register, 23-4
PIODATA15_0 register, 23-4
PIODATA31_16 register, 23-4
PIODIR15_0 register, 23-4
PIODIR31_16 register, 23-4
PIOPFS15_0 register, 23-4
PIOPFS31_16 register, 23-4
PIOSET31_16 register, 23-4
PIOx_FNC bit field, 13-3, 14-4, 15-2, 21-2, 23-3
PIT Channel x Count (PITxCNT) register
function, 16-3
usage, 16-4, 16-6
PIT Counter Latch Command (PITCNTLAT) register
function, 16-3

- PIT Mode Control (PITMODECTL) register
  - function, 16-3
  - usage, 16-4, 16-7
- PIT Read-Back Command (PITRDBACK) register
  - function, 16-3
- PIT x Interrupt Mapping (PITxMAP) register
  - function, 15-4, 16-2
- PIT x Status (PITxSTA) register
  - function, 16-3
- PIT. *See* programmable interval timer (PIT).
- PIT_GATE2 bit field, 16-4
- PIT_OUT2_ENB bit field, 16-4
- PIT_OUT2_STA bit field, 16-4
- PITCNTLAT register, 16-3
- PITGATE2 signal
  - control, 13-3, 13-6, 16-1, 16-2, 16-4
  - description, 2-10
- PITMODECTL register, 16-3
- PITOUT2 signal
  - control, 16-3
  - description, 2-10
  - usage, 16-1, 16-4
- PITRDBACK register, 16-3
- PITxCNT register, 16-3
- PITxMAP register, 15-4
- PITxSTA register, 16-3
- PLL. *See* clock generation and control. *See also* reset generation.
- power
  - signal descriptions, 2-14
- Power Good signal. *See* PWRGOOD signal.
- Power Supply signals. *See* VCC_CORE signal, VCC_I/O signal, and VCC_RTC signal.
- power-on reset, 6-9
- PRG_RST_ENB bit field, 3-4, 6-4, 6-6, 10-29
- PRGRESET signal
  - control, 6-3, 10-10
  - description, 2-10
  - timing (figure), 6-6
  - usage, 3-4, 6-4, 6-6, 6-8, 6-9, 10-10, 10-29
- PRGRST_DET bit field, 6-8
- Programmable Address Region x (PARx) registers
  - address region attributes, 3-12
  - cacheability control, 3-12
  - code execution control, 3-12
  - doubleword boundaries, 4-20
  - external GP bus devices, 3-13
  - external ROM devices, 3-17
  - format (figure), 3-10
  - function, 4-2, 24-2
  - maximum region size, 4-20
  - PAR register priority, 3-13
  - PCI bus devices, 3-15
  - performance considerations of attributes, 3-12
  - region size, 4-18
  - SDRAM regions, 3-18
  - software considerations, 4-18
  - specifying pages and regions, 3-9
  - start address, 4-18
  - usage, 2-11, 3-8, 3-10, 4-5, 12-14, 13-6, 13-9, 13-22, 15-9
  - worksheet (figure), 3-11
  - write-protection, 3-12
- programmable input/output (PIO)
  - block diagram (figure), 23-2
  - configuration
    - configuration summary (table), 23-5
    - input pins, 23-5
    - output pins, 23-5
  - initialization, 23-6
  - operation, 23-4
  - overview, 23-1
  - PIO31–PIO0 signals, 23-4
  - registers, 23-4
  - signal descriptions, 2-11
  - software considerations, 23-5
  - system design, 23-2
    - shared signals (table), 23-3
- Programmable Input/Output signals. *See* PIO31–PIO0 signals.
- programmable interrupt controller (PIC)
  - block diagram (figure), 15-3
  - configuration, 15-16
    - PC/AT configuration, 15-18
    - programming, 15-16
  - configuring interrupt mapping, 3-19
  - edge-triggered or level-sensitive interrupts, 15-13
  - initialization, 15-20
  - interrupt flow sequence, 15-7
  - interrupt sharing, 15-13
  - interrupt source routing, 15-10
    - disabling the slave controllers, 15-13
    - floating point error handling, 15-12
    - PC/AT compatibility, 15-12
    - polarity inversion of interrupt requests, 15-10
  - interrupt source routing (figure), 15-11
  - interrupt sources, 15-8
    - hardware-generated interrupts, 15-8
  - interrupt sources (figure), 15-9
  - non-maskable interrupts and routing, 15-14
    - NMI routing (figure), 15-15
    - NMI sharing, 15-14
  - operation, 15-7
  - overview, 15-1
  - PC/AT interrupt channel mapping (table), 15-12
  - priority types, 15-16
  - registers, 15-4
  - software considerations, 15-18
    - detecting invalid interrupt requests, 15-19
    - disabling the slave controllers, 15-19

- floating point unit error handling, 15-19
- interrupt sharing, 15-18
- system design, 15-2
  - shared signals (table), 15-2

programmable interval timer (PIT)
- block diagram (figure), 16-2
- clocking considerations, 16-6
  - external clock source (table), 16-6
  - internal clock source (table), 16-6
- initialization, 16-7
- interrupts, 16-6
- operating modes, 16-4
  - hardware-retriggerable one-shot, 16-4
  - hardware-triggered strobe, 16-5
  - interrupt on terminal count, 16-4
  - rate generator, 16-5
  - software-triggered strobe, 16-5
  - square wave mode, 16-5
- operation, 16-3
- overview, 16-1
- PIT Channel 0, 16-3
- PIT Channel 1, 16-3
- PIT Channel 2, 16-4
- registers, 16-2
- signal descriptions, 2-10
- software considerations, 16-6
  - PC/AT-compatible systems, 16-6
- system design, 16-1
  - shared signals (table), 16-1

Programmable Interval Timer 2 Gate signal. *See* PITGATE2 signal.

Programmable Interval Timer 2 Output signal. *See* PITOUT2 signal.

programmable reset, 6-6, 10-29

Programmable Reset signal. *See* PRGRESET signal.

PSC_SEL bit field, 17-3

pulldown resistors
- configuration signals, 2-13
- ECC devices not installed, 10-9
- internal value, 2-4
- JTAG boundary scan test interface, 25-16
- JTAG signals, 2-12
- PIO31–PIO0 signals, 23-1–23-2
- PIO31–PIO0 signals (table), 23-3
- SSI devices, 22-1

pullup resistors
- GP bus external pullups required, 13-4
- GPRDY signal, 2-9
- internal value, 2-4
- JTAG boundary scan test interface, 25-4
- JTAG signals, 2-12
- PCI external pullups required, 9-3
- PIO31–PIO0 signals, 23-1–23-2
- PIO31–PIO0 signals (table), 23-3
- SSI devices, 22-1
- UART serial port signals, 21-2

PWRGOOD signal
- description, 2-10
- timing, 6-9
- usage, 6-2, 6-4, 6-8, 20-5

PWRGOOD_DET bit field, 6-8

R

RAS_CAS_DLY bit field, 10-21

RAS_PCHG_DLY bit field, 10-21

RATE_SEL bit field, 20-8

read buffer. *See* write buffer and read buffer.

real-time clock (RTC)
- block diagram (figure), 20-2
- configuration, 20-7
  - alarm function, 20-9
  - date and time, 20-8
  - hour format, 20-7
  - periodic interrupts, 20-8
  - using RATE_SEL (table), 20-8
  - year 2000 issues, 20-9
- disabling, 3-21
- initialization, 20-10
  - RTC reset, 20-11
- interrupts, 20-9
- operation, 20-7
- overview, 20-1
- registers, 20-6
- RTC reset, 6-7
- software considerations, 20-10
  - accessing the CMOS memory, 20-10
  - initializing the RTC divider chain, 20-10
  - legacy NMI enable bit moved, 20-10
- system design, 20-3
  - backup battery considerations, 20-3
  - circuit with backup battery (figure), 20-4
  - circuit without backup battery (figure), 20-5
  - external RTC, 20-5
  - selecting and interfacing a 32.768-kHz crystal, 20-5
- voltage monitor, 20-2
- voltage monitor block diagram (figure), 20-3

register set manual, xxiv

REGISTER_NUM bit field, 9-10

registers
- Address Decode Control (ADDDECCTL), 4-2
- Am5x86 CPU Control (CPUCTL), 7-1
- AMDebug Technology RX/TX Interrupt Mapping (ICEMAP), 15-5
- Arbiter Priority Control (ARBPRICTL), 8-2
- $\overline{\text{BOOTCS}}$ Control (BOOTCSCTL), 12-5
- Buffer Chaining Control (GPDMABCCTL), 14-5
- Buffer Chaining Interrupt Enable (GPDMABSINTENB), 14-5
- Buffer Chaining Status (GPDMABCSTA), 14-5

registers (continued)
- Buffer Chaining Valid (GPDMABCVAL), 14-5
- Chip Select Pin Function Select (CSPFS), 23-4
- Class Code/Revision ID (PCICCREVID), 9-8
- Clock Select (CLKSEL), 5-6, 23-4
- Configuration Base Address (CBAR), 4-2
- Device/Vendor ID (PCIDEVID), 9-8
- DMA Buffer Chaining Interrupt Mapping (DMABCINTMAP), 14-4, 15-5
- Drive Strength Control (DSCTL), 23-4
- ECC Check Bit Position (ECCCKBPOS), 10-10
- ECC Check Code Test (ECCCKTEST), 10-10
- ECC Control (ECCCTL), 10-10
- ECC Interrupt Mapping (ECCMAP), 15-4
- ECC Multi-Bit Error Address (ECCMBADD), 10-10
- ECC Single-Bit Error Address (ECCSBADD), 10-10
- ECC Status (ECCSTA), 10-10
- ÉlanSC520 Microcontroller Revision ID (REVID), 7-1
- Floating Point Error Interrupt Clear (FPUERRCLR), 15-7
- Floating Point Error Interrupt Mapping (FERRMAP), 15-5
- General-Purpose CMOS RAM (RTCCMOS), 20-7
- GP ALE Offset (GPALEOFF), 13-6
- GP ALE Pulse Width (GPALEW), 13-6
- GP Chip Select Data Width (GPCSDW), 13-5
- GP Chip Select Offset (GPCSOFF), 13-5
- GP Chip Select Pulse Width (GPCSPW), 13-5
- GP Chip Select Qualification (GPCSQUAL), 13-5
- GP Chip Select Recovery Time (GPCSRT), 13-5
- GP Echo Mode (GPECHO), 13-5
- GP Read Offset (GPRDOFF), 13-5
- GP Read Pulse Width (GPRDW), 13-5
- GP Timer x Count (GPTMRxCNT), 17-2, 17-3
- GP Timer x Interrupt Mapping (GPTMRxMAP), 15-4
- GP Timer x Maxcount Compare A (GPTMRxMAXCMPA), 17-2, 17-3
- GP Timer x Maxcount Compare B (GPTMRxMAXCMPB), 17-2, 17-3
- GP Timer x Mode/Control (GPTMRxCTL), 17-2, 17-3
- GP Timers Status (GPTMRSTA), 17-2
- GP Write Offset (GPWROFF), 13-6
- GP Write Pulse Width (GPWRW), 13-5
- GP-DMA Channel x Extended Page (GPDMAEXTPGx), 14-4, 14-5
- GP-DMA Channel x Extended Transfer Count (GPDMAEXTTCx), 14-5
- GP-DMA Channel x Next Address High (GPDMANXTADDHx), 14-5, 14-6
- GP-DMA Channel x Next Address Low (GPDMANXTADDLx), 14-5, 14-6
- GP-DMA Channel x Next Transfer Count High (GPDMANXTTCHx), 14-6
- GP-DMA Channel x Next Transfer Count Low (GPDMANXTTCLx), 14-6

registers (continued)
- GP-DMA Control (GPDMACTL), 14-4
- GP-DMA Memory-Mapped I/O (GPDMAMMIO), 14-4
- GP-DMA Resource Channel Map A (GPDMAEXTCHMAPA), 14-4
- GP-DMA Resource Channel Map B (GPDMAEXTCHMAPB), 14-4
- GPIRQx Interrupt Mapping (GPxIMAP), 15-5
- Header Type (PCIHEADTYPE), 9-8
- Host Bridge Control (HBCTL), 9-7
- Host Bridge Master Interrupt Address (MSTINTADD), 9-7
- Host Bridge Master Interrupt Control (HBMSTIRQCTL), 9-7
- Host Bridge Master Interrupt Status (HBMSTIRQSTA), 9-7
- Host Bridge Target Interrupt Control (HBTGTIRQCTL), 9-7
- Host Bridge Target Interrupt Status (HBTGTIRQSTA), 9-7
- Interrupt Control (PICICR), 15-4
- Interrupt Pin Polarity (INTPINPOL), 15-4
- Master DMA Channel 4–7 Control (MSTDMACTL), 14-7
- Master DMA Channel 4–7 Mask (MSTDMAMSK), 14-7
- Master DMA Channel 4–7 Mode (MSTDMAMODE), 14-7
- Master DMA Channel 4–7 Status (MSTDMASTA), 14-7
- Master DMA Channel x Memory Address (GPDMAxMAR), 14-7
- Master DMA Channel x Page (GPDMAxPG), 14-7
- Master DMA Channel x Transfer Count (GPDMAxTC), 14-7
- Master DMA Clear Byte Pointer (MSTDMACBP), 14-7
- Master DMA Controller Reset (MSTDMARST), 14-8
- Master DMA Controller Temporary (MSTDMATMP), 14-8
- Master DMA General Mask (MSTDMAGENMSK), 14-8
- Master DMA Mask Reset (MSTDMAMSKRST), 14-8
- Master PIC Initialization Control Word 1 (MPICICW1), 15-6
- Master PIC Initialization Control Word 2 (MPICICW2), 15-6
- Master PIC Initialization Control Word 3 (MPICICW3), 15-7
- Master PIC Initialization Control Word 4 (MPICICW4), 15-7
- Master PIC In-Service (MPICISR), 15-6
- Master PIC Interrupt Mask (MPICINTMSK), 15-7
- Master PIC Interrupt Mode (MPICMODE), 15-4
- Master PIC Interrupt Request (MPICIR), 15-6

registers (continued)
- Master PIC Operation Control Word 2 (MPICOCW2), 15-6
- Master PIC Operation Control Word 3 (MPICOCW3), 15-6
- Master Retry Time-Out (PCIMRETRYTO), 9-8
- Master Software DRQ(n) Request (MSTDMASWREQ), 14-7
- PCI Bus Arbiter Status (PCIARBSTA), 8-2
- PCI Configuration Address (PCICFGADR), 9-8
- PCI Configuration Data (PCICFGDATA), 9-8
- PCI Host Bridge Interrupt Mapping (PCIHOSTMAP), 15-4
- PCI Interrupt A Mapping (PCIINTAMAP), 15-5
- PCI Interrupt B Mapping (PCIINTBMAP), 15-5
- PCI Interrupt C Mapping (PCIINTCMAP), 15-5
- PCI Interrupt D Mapping (PCIINTDMAP), 15-5
- PIO15–PIO0 Clear (PIOCLR15_0), 23-4
- PIO15–PIO0 Data (PIODATA15_0), 23-4
- PIO15–PIO0 Direction (PIODIR15_0), 23-4
- PIO15–PIO0 Pin Function Select (PIOPFS15_0), 23-4
- PIO15–PIO0 Set (PIOSET15_0), 23-4
- PIO31–PIO16 Clear (PIOCLR31_16), 23-4
- PIO31–PIO16 Data (PIODATA31_16), 23-4
- PIO31–PIO16 Direction (PIODIR31_16), 23-4
- PIO31–PIO16 Pin Function Select (PIOPFS31_16), 23-4
- PIO31–PIO16 Set (PIOSET31_16), 23-4
- PIT Channel x Count (PITxCNT), 16-3
- PIT Counter Latch Command (PITCNTLAT), 16-3
- PIT Mode Control (PITMODECTL), 16-3
- PIT Read-Back Command (PITRDBACK), 16-3
- PIT x Interrupt Mapping (PITxMAP), 15-4
- PIT x Status (PITxSTA), 16-3
- Programmable Address Region x (PARx), 4-2
- Reset Configuration (RESCFG), 6-3
- Reset Status (RESSTA), 6-3
- $\overline{\text{ROMCS1}}$ Control (ROMCS1CTL), 12-5
- $\overline{\text{ROMCS2}}$ Control (ROMCS2CTL), 12-5
- RTC Alarm Hour (RTCALMHR), 20-6
- RTC Alarm Minute (RTCALMMIN), 20-6
- RTC Alarm Second (RTCALMSEC), 20-6
- RTC Control A (RTCCTLA), 20-7
- RTC Control B (RTCCTLB), 20-7
- RTC Current Day of the Month (RTCCURDOM), 20-6
- RTC Current Day of the Week (RTCCURDOW), 20-6
- RTC Current Hour (RTCCURHR), 20-6
- RTC Current Minute (RTCCURMIN), 20-6
- RTC Current Month (RTCCURMON), 20-6
- RTC Current Second (RTCCURSEC), 20-6
- RTC Current Year (RTCCURYR), 20-6
- RTC Interrupt Mapping (RTCMAP), 15-5
- RTC Status C (RTCSTAC), 20-7
- RTC Status D (RTCSTAD), 20-7

registers (continued)
- RTC/CMOS RAM Data Port (RTCDATA), 20-6
- RTC/CMOS RAM Index (RTCIDX), 20-6
- SCP Command Port (SCPCMD), 6-3
- SCP Data Port (SCPDATA), 6-3
- SDRAM Bank 0–3 Ending Address (DRCBENDADR), 10-10
- SDRAM Bank Configuration (DRCCFG), 10-10
- SDRAM Buffer Control (DBCTL), 11-4
- SDRAM Control (DRCCTL), 10-10
- SDRAM Timing Control (DRCTMCTL), 10-10
- Slave DMA Channel 0–3 Control (SLDMACTL), 14-7
- Slave DMA Channel 0–3 Mask (SLDMAMSK), 14-7
- Slave DMA Channel 0–3 Mode (SLDMAMODE), 14-7
- Slave DMA Channel 0–3 Status (SLDMASTA), 14-7
- Slave DMA Channel x Memory Address (GPDMAxMAR), 14-7
- Slave DMA Channel x Page (GPDMAxPG), 14-7
- Slave DMA Channel x Transfer Count (GPDMAxTC), 14-7
- Slave DMA Clear Byte Pointer (SLDMACBP), 14-7
- Slave DMA Controller Reset (SLDMARST), 14-8
- Slave DMA Controller Temporary (SLDMATMP), 14-8
- Slave DMA General Mask (SLDMAGENMSK), 14-8
- Slave DMA Mask Reset (SLDMAMSKRST), 14-8
- Slave Software DRQ(n) Request (SLDMASWREQ), 14-7
- Slave x PIC Initialization Control Word 1 (SxPICICW1), 15-6
- Slave x PIC Initialization Control Word 2 (SxPICICW2), 15-6
- Slave x PIC Initialization Control Word 3 (SxPICICW3), 15-7
- Slave x PIC Initialization Control Word 4 (SxPICICW4), 15-7
- Slave x PIC In-Service (SxPICISR), 15-6
- Slave x PIC Interrupt Mask (SxPICINTMSK), 15-7
- Slave x PIC Interrupt Mode (SLxPICMODE), 15-4
- Slave x PIC Interrupt Request (SxPICIR), 15-6
- Slave x PIC Operation Control Word 2 (SxPICOCW2), 15-6
- Slave x PIC Operation Control Word 3 (SxPICOCW3), 15-6
- Software Interrupt 16–1 Control (SWINT16_1), 15-4
- Software Interrupt 22–17/NMI Control (SWINT22_17), 15-4
- Software Timer Configuration (SWTMRCFG), 18-2
- Software Timer Microsecond Count (SWTMRMICRO), 18-2
- Software Timer Millisecond Count (SWTMRMILLI), 18-2
- SSI Command (SSICMD), 22-2
- SSI Control (SSICTL), 22-2
- SSI Interrupt Mapping (SSIMAP), 15-5
- SSI Receive (SSIRCV), 22-2

registers (continued)
  SSI Status (SSISTA), 22-2
  SSI Transmit (SSIXMIT), 22-2
  Status/Command (PCISTACMD), 9-8
  System Arbiter Control (SYSARBCTL), 8-2
  System Arbiter Master Enable (SYSARBMENB), 8-2
  System Board Information (SYSINFO), 6-3
  System Control Port A (SYSCTLA), 6-3
  System Control Port B (SYSCTLB), 16-3
  UART x Baud Clock Divisor Latch LSB (UARTxBCDL), 21-4
  UART x Baud Clock Divisor Latch MSB (UARTxBCDH), 21-4
  UART x FIFO Control (UARTxFCR), 21-4
  UART x FIFO Control Shadow (UARTxFCRSHAD), 21-3
  UART x General Control (UARTxCTL), 21-3
  UART x General Status (UARTxSTA), 21-3
  UART x Interrupt Enable (UARTxINTENB), 21-4
  UART x Interrupt ID (UARTxINTID), 21-4
  UART x Interrupt Mapping (UARTxMAP), 15-5
  UART x Line Control (UARTxLCR), 21-4
  UART x Line Status (UARTxLSR), 21-4
  UART x Modem Control (UARTxMCR), 21-4
  UART x Modem Status (UARTxMSR), 21-4
  UART x Receive Buffer (UARTxRBR), 21-4
  UART x Scratch Pad (UARTxSCRATCH), 21-4
  UART x Transmit Holding (UARTxTHR), 21-4
  Watchdog Timer Control (WDTMRCTL), 19-2
  Watchdog Timer Count High (WDTMRCNTH), 19-3, 19-6
  Watchdog Timer Count Low (WDTMRCNTL), 19-2
  Watchdog Timer Interrupt Mapping (WDTMAP), 15-5
  Write-Protect Violation Interrupt Mapping (WPVMAP), 15-5
  Write-Protect Violation Status (WPVSTA), 4-2

$\overline{\text{REQ4}}$–$\overline{\text{REQ0}}$ signals
  control, 2-7, 8-2
  usage, 8-7

Request To Send signals. *See* $\overline{\text{RTS2}}$–$\overline{\text{RTS1}}$ signals.

RESCFG register, 6-3

Reset Configuration (RESCFG) register
  function, 6-3, 10-10, 13-6, 24-2
  usage, 6-4, 6-5, 6-6, 6-7, 10-29, 13-22, 24-11

reset generation
  a20 gate support, 6-8
  block diagram (figure), 6-2
  core states after system reset (table), 6-5
  determining reset sources, 6-8
  GP bus reset, 6-7
  hard CPU reset, 7-5
  initialization, 6-9
  latency, 6-9
  operation, 6-3
  overview, 6-1
  PCI reset, 6-7
  PLL start-up, 6-8
  PLL start-up timing (figure), 6-9
  power-on reset, 6-9
    power-on reset timing (figure), 6-9
  PRGRESET timing (figure), 6-6
  programmable reset, 6-6, 10-29
  registers, 6-3
  reset sources (table), 6-4
  reset types, 6-3
  reset vector and reset segment, 3-4
  RTC reset, 6-7
  signal descriptions, 2-10
  soft CPU reset, 6-7, 7-5
  software considerations, 6-8
  system design, 6-2
  system reset, 6-4
  system reset with SDRAM retention, 6-6

Reset Latched Input signals. *See* RSTLD7–RSTLD0 signals.

Reset signal. *See* $\overline{\text{RST}}$ signal.

Reset Status (RESSTA) register
  function, 6-3, 7-1, 10-10, 19-3, 24-3
  usage, 6-8, 24-11

RESSTA register, 6-3

REVID register, 7-1

RF_CLR bit field, 21-10

RFRT bit field, 21-7, 21-10

RFSH_ENB bit field, 10-19

RFSH_SPD bit field, 10-18

$\overline{\text{RIN2}}$–$\overline{\text{RIN1}}$ signals
  control, 13-6, 21-2, 21-3, 21-4
  description, 2-9, 21-6
  usage, 21-2, 21-9

Ring Indicate signals. *See* $\overline{\text{RIN2}}$–$\overline{\text{RIN1}}$ signals.

ROM Buffer Output Enable signal. *See* $\overline{\text{ROMBUFOE}}$ signal.

ROM/Flash Boot Chip Select signal. *See* $\overline{\text{BOOTCS}}$ signal.

ROM/Flash Chip Select signals. *See* $\overline{\text{ROMCS2}}$–$\overline{\text{ROMCS1}}$ signals.

ROM/Flash controller
  access timing and wait states example (table), 12-9
  accesses and ROM width (table), 12-9
  address decoding, 12-12
  block diagram (figure), 12-2
  bus cycles, 12-9
    2 doublewords from 16-bit ROM (figure), 12-11
    4 aligned doublewords from 32-bit ROM (figure), 12-8
    4 unaligned doublewords from 8-bit ROM (figure), 12-8
    4 words from 16-bit ROM (figures), 12-8
    burst access from 32-bit ROM (figure), 12-10

    - cache-line fill (figure), 12-11
    - multiple accesses from 8-bit ROM (figure), 12-10
    - page-mode read access, 12-10
    - single CPU read access, 12-9
    - word write cycle to Flash memory (figure), 12-12
    - writing to Flash, 12-11
  - cacheability control, 3-12
  - code execution control, 3-12
  - configuration, 12-7
    - access timing, 12-8
    - operating mode, 12-7
    - ROM location, 12-7
    - ROM width, 12-7
  - configuring external ROM devices
    - boot device mapping for BIOS shadowing, 3-17
    - two Flash banks for XIP operating system, 3-17
  - device types supported, 12-6
  - initialization, 12-14
    - CFGx options for BOOTCS (table), 12-14
  - latency, 12-13
  - memory space, 4-4
  - operation, 12-5
  - overview, 12-1
  - programming Flash, 12-12
  - registers, 12-5
  - sharing address and data bus with GP bus, 13-10
  - signal descriptions, 2-6
  - software considerations, 12-12
  - system design, 12-2
    - data bus connection options (table), 12-1
    - shared signals (table), 12-3
    - voltage isolation, 12-3
    - voltage isolation examples (figure), 12-4
  - write-protection, 3-12, 3-18
- ROM/Flash controller data bus, 2-8
- ROM/Flash Read signal. *See* ROMRD signal.
- ROMBUFOE signal
  - description, 2-6
  - usage, 2-8, 12-3
- ROMCS1 Control (ROMCS1CTL) register
  - function, 12-5
  - usage, 12-14
- ROMCS1CTL register, 12-5
- ROMCS2 Control (ROMCS2CTL) register
  - function, 12-5
  - usage, 12-14
- ROMCS2CTL register, 12-5
- ROMCS2–ROMCS1 signals
  - control, 3-10, 12-3, 13-3, 13-6
  - description, 2-6
  - usage, 3-17, 4-8, 4-15, 12-3, 12-14, 24-6
- ROMRD signal
  - description, 2-6
  - usage, 12-3, 24-4, 24-6
- Row Address Strobe signals. *See* SRASA–SRASB signals.
- RST signal
  - control, 9-29
  - description, 2-7
  - usage, 6-4, 6-7, 9-2, 9-3, 9-29
- RSTLD7–RSTLD0 signals
  - description, 2-14
  - usage, 6-3, 6-4, 6-5, 6-6
- RTC. *See* real-time clock (RTC).
- RTC Alarm Hour (RTCALMHR) register
  - function, 20-6
  - usage, 20-7, 20-10
- RTC Alarm Minute (RTCALMMIN) register
  - function, 20-6
- RTC Alarm Second (RTCALMSEC) register
  - function, 20-6
- RTC Control A (RTCCTLA) register
  - function, 20-7
  - usage, 20-7, 20-8, 20-10
- RTC Control B (RTCCTLB) register
  - function, 20-7
  - usage, 20-7, 20-9, 20-10
- RTC Current Day of the Month (RTCCURDOM) register
  - function, 20-6
- RTC Current Day of the Week (RTCCURDOW) register
  - function, 20-6
- RTC Current Hour (RTCCURHR) register
  - function, 20-6
  - usage, 20-7
- RTC Current Minute (RTCCURMIN) register
  - function, 20-6
- RTC Current Month (RTCCURMON) register
  - function, 20-6
- RTC Current Second (RTCCURSEC) register
  - function, 20-6
- RTC Current Year (RTCCURYR) register
  - function, 20-6
  - usage, 20-9
- RTC Interrupt Mapping (RTCMAP) register
  - function, 15-5
- RTC Status C (RTCSTAC) register
  - function, 20-7
- RTC Status D (RTCSTAD) register
  - function, 20-7
  - usage, 6-7, 20-4
- RTC/CMOS RAM Data Port (RTCDATA) register
  - function, 20-6
  - usage, 20-10
- RTC/CMOS RAM Index (RTCIDX) register
  - function, 20-6
  - usage, 20-10
- RTC_DIS bit field, 3-21, 20-5
- RTC_VRT bit field, 6-7, 20-4
- RTCALMHR register, 20-6

- RTCALMMIN register, 20-6
- RTCALMSEC register, 20-6
- RTCCMOS register, 20-7
- RTCCTLA register, 20-7
- RTCCTLB register, 20-7
- RTCCURDOM register, 20-6
- RTCCURDOW register, 20-6
- RTCCURHR register, 20-6
- RTCCURMIN register, 20-6
- RTCCURMON register, 20-6
- RTCCURSEC register, 20-6
- RTCCURYR register, 20-6
- RTCDATA register, 20-6
- RTCIDX register, 20-6
- RTCMAP register, 15-5
- RTCSTAC register, 20-7
- RTCSTAD register, 20-7
- RTG bit field, 17-3
- $\overline{\text{RTS2}}$–$\overline{\text{RTS1}}$ signals
  - control, 21-4
  - description, 2-9, 21-6
  - usage, 21-2

S

- S1_GINT_MODE bit field, 15-18
- S2 bit field, 15-17, 15-18, 15-19
- S2_GINT_MODE bit field, 3-19
- S5 bit field, 15-17, 15-18, 15-19
- SBIT_ERR bit field, 10-27
- $\overline{\text{SCASA}}$–$\overline{\text{SCASB}}$ signals
  - control, 10-10, 10-19, 23-4
  - description, 2-5
  - usage, 10-5
- SCP Command Port (SCPCMD) register
  - function, 6-3, 7-1
  - usage, 6-7, 6-8
- SCP Data Port (SCPDATA) register
  - function, 6-3, 7-1
  - usage, 6-8
- SCPCMD register, 6-3
- SCPDATA register, 6-3
- $\overline{\text{SCS3}}$–$\overline{\text{SCS0}}$ signals
  - control, 10-10, 10-19, 23-4
  - description, 2-5
  - usage, 10-1, 10-5, 10-12, 10-30
- SD_RST_DET bit field, 6-8
- SDQM3–SDQM0 signals
  - control, 10-10, 10-19, 23-4
  - description, 2-5
  - usage, 10-6, 10-28, 24-11
- SDRAM Address signals. *See* MA12–MA0 signals.
- SDRAM Bank 0–3 Ending Address (DRCBENDADR) register
  - function, 10-10
  - usage, 10-32, 10-33, 10-35
- SDRAM Bank Configuration (DRCCFG) register
  - function, 10-10
  - usage, 10-15, 10-33
- SDRAM Buffer Control (DBCTL) register
  - function, 11-4, 24-2
  - usage, 11-5, 11-9, 11-13, 24-10
- SDRAM Chip Select signals. *See* $\overline{\text{SCS3}}$–$\overline{\text{SCS0}}$ signals.
- SDRAM Clock Input signal. *See* CLKMEMIN signal.
- SDRAM Clock Output signal. *See* CLKMEMOUT signal.
- SDRAM Control (DRCCTL) register
  - function, 10-10, 11-4, 24-2
  - usage, 10-18, 10-19, 10-30, 24-2, 24-3, 24-7
- SDRAM controller
  - addressing, 10-12
    - address mapping to MA (table), 10-12
    - page sizes (table), 10-16
    - SDRAM devices supported (table), 10-13
    - supported SDRAM devices, 10-13
  - block diagram (figure), 10-2
  - block diagram detail (figure), 10-3
  - buffering, 10-17
  - bus cycles, 10-22
    - auto refresh cycle (figure), 10-27
    - burst read cycle (figure), 10-22
    - burst read cycle with ECC enabled (figure), 10-25
    - CPU burst write (figure), 10-24
    - ECC cycles, 10-24
    - mode register access (figure), 10-27
    - read-modify-write cycle with ECC (figure), 10-26
    - write cycle (figure), 10-23
  - cacheability control, 3-12
  - code execution control, 3-12
  - column address configuration (table), 10-15
  - configuring GP-DMA buffers with PAR registers, 3-18
  - control configuration, 10-18
    - drive-strength selection, 10-19
    - operation mode select, 10-30
    - refresh control, 10-18
    - refresh rates (table), 10-18
    - write buffer test mode, 10-19
  - error correction code (ECC), 10-16, 10-27, 10-28
  - initialization, 10-29
    - boot process, 10-32
    - programmable reset, 10-29
    - SDRAM device initialization, 10-30
      - auto refresh command, 10-31
      - load mode register (table), 10-31
      - mode register programming, 10-31
      - NOP command, 10-31

operation mode select, 10-20
precharge command, 10-31
sizing algorithm, 10-32
external bank column number, 10-33
internal bank number, 10-34
true external bank ending address, 10-35
interrupts, 10-27
multi-bit error, 10-27
single-bit error, 10-27
latency, 10-28
memory space, 4-4, 4-8
operation, 10-11
registers, 10-10
SDRAM support, 10-11
signal descriptions, 2-5
software considerations, 10-28
disabling buffers during configuration, 10-28
ECC errors, 10-28
write protection, 10-28
system design, 10-1
168-pin SDRAM DIMM configuration (figure), 10-5
bank configuration (figure), 10-4
clock generation (figure), 10-7
clock generation with external driver (figure), 10-7
clock loading estimates (table), 10-6
clocking, 10-6
delay calculation, 10-7
estimated capacitance (tables), 10-8
loading, 10-8
pins, 10-5
timing configuration, 10-20
auto-refresh-to-$\overline{\text{RAS}}$ (T_{RC}), 10-21
$\overline{\text{CAS}}$ latency (C_{L}), 10-20
minimum $\overline{\text{RAS}}$ (T_{RAS}), 10-22
$\overline{\text{RAS}}$ precharge (T_{RP}), 10-21
$\overline{\text{RAS}}$-to-$\overline{\text{CAS}}$ delay (T_{RCD}), 10-21
$\overline{\text{RAS}}$-to-$\overline{\text{RAS}}$, 10-21
write-protection, 3-12, 3-18
SDRAM Memory Write Enable signals. *See* $\overline{\text{SWEA}}$–$\overline{\text{SWEB}}$ signals.
SDRAM Timing Control (DRCTMCTL) register
function, 10-10
usage, 10-20, 10-31, 24-5, 24-9
Serial Data In signals. *See* SIN2–SIN1 signals.
Serial Data Out signals. *See* SOUT2–SOUT1 signals.
Serial Debug Port Data (SDPD) register
format, 25-14
function, 25-2
usage, 25-14
$\overline{\text{SERR}}$ signal
control, 9-8
description, 2-7
usage, 9-3, 9-27
SET bit field, 20-7
Set Interrupt-Enable Flag (STI) instruction, 15-7, 15-18, 15-20
SFNM bit field, 15-18
SGL_INT_ENB bit field, 10-27
signal descriptions
AMDebug™ technology, 2-12
configuration, 2-13
descriptions (table), 2-5
general-purpose (GP) bus, 2-7, 2-11
general-purpose (GP) timers, 2-10
GP-DMA controller, 2-7
JTAG, 2-12
PCI host bridge, 2-6
power, 2-14
programmable input/output (PIO), 2-11
programmable interval timer (PIT), 2-10
reset generation, 2-10
ROM/Flash controller, 2-6
SDRAM controller, 2-5
synchronous serial interface (SSI), 2-9
UART serial ports, 2-9
signals
32KXTAL2–32KXTAL1, 2-10
33MXTAL2–33MXTAL1, 2-10
AD31–AD0, 2-6
AMDEBUG_DIS, 2-13
BA1–BA0, 2-5
BBATSEN, 2-14
$\overline{\text{BOOTCS}}$, 2-6
BR/TC, 2-12
$\overline{\text{CBE3}}$–$\overline{\text{CBE0}}$, 2-6
$\overline{\text{CF\_DRAM}}$, 2-12
$\overline{\text{CF\_ROM\_GPCS}}$, 2-12
CFG2–CFG0, 2-13
CFG3, 2-13
CLKMEMIN, 2-5
CLKMEMOUT, 2-5
CLKPCIIN, 2-6
CLKPCIOUT, 2-7
CLKTEST, 2-10
CLKTIMER, 2-10
CMDACK, 2-12
$\overline{\text{CTS2}}$–$\overline{\text{CTS1}}$, 2-9
DATASTRB, 2-12
$\overline{\text{DCD2}}$–$\overline{\text{DCD1}}$, 2-9
DEBUG_ENTER, 2-13
$\overline{\text{DEVSEL}}$, 2-7
$\overline{\text{DSR2}}$–$\overline{\text{DSR1}}$, 2-9
$\overline{\text{DTR2}}$–$\overline{\text{DTR1}}$, 2-9
$\overline{\text{FLASHWR}}$, 2-6
$\overline{\text{FRAME}}$, 2-7
GND, 2-14
$\overline{\text{GNT4}}$–$\overline{\text{GNT0}}$, 2-7
GPA25–GPA0, 2-6, 2-7
GPAEN, 2-8
GPALE, 2-8
$\overline{\text{GPBHE}}$, 2-8

signals (continued)
- GPCS7–GPCS0, 2-11
- GPD15–GPD0, 2-6, 2-8
- GPDACK3–GPDACK0, 2-8
- GPDBUFOE, 2-8
- GPDRQ3–GPDRQ0, 2-8
- GPIOCS16, 2-8
- GPIORD, 2-8
- GPIOWR, 2-8
- GPIRQ10–GPIRQ0, 2-9
- GPMEMCS16, 2-9
- GPMEMRD, 2-9
- GPMEMWR, 2-9
- GPRDY, 2-9
- GPRESET, 2-9
- GPTC, 2-9
- INST_TRCE, 2-13
- INTA–INTD, 2-7
- IRDY, 2-7
- JTAG_TCK, 2-12
- JTAG_TDI, 2-12
- JTAG_TDO, 2-12
- JTAG_TMS, 2-12
- JTAG_TRST, 2-12
- LF_PLL1, 2-10
- MA12–MA0, 2-5
- MD31–MD0, 2-5, 2-6
- MECC6–MECC0, 2-5
- PAR, 2-7
- PERR, 2-7
- PIO31–PIO0, 2-11
- PITGATE2, 2-10
- PITOUT2, 2-10
- PRGRESET, 2-10
- PWRGOOD, 2-10
- REQ4–REQ0, 2-7
- RIN2–RIN1, 2-9
- ROMBUFOE, 2-6
- ROMCS2–ROMCS1, 2-6
- ROMRD, 2-6
- RST, 2-7
- RSTLD7–RSTLD0, 2-14
- RTS2–RTS1, 2-9
- SCASA–SCASB, 2-5
- SCS3–SCS0, 2-5
- SDQM3–SDQM0, 2-5
- SERR, 2-7
- SIN2–SIN1, 2-9
- SOUT2–SOUT1, 2-9
- SRASA–SRASB, 2-5
- SSI_CLK, 2-10
- SSI_DI, 2-10
- SSI_DO, 2-10
- STOP, 2-7
- STOP/TX, 2-12
- SWEA–SWEB, 2-5
- TMRIN1–TIMIN0, 2-10

signals (continued)
- TRDY, 2-7
- TRIG/TRACE, 2-12
- VCC_ANLG, 2-14
- VCC_CORE, 2-14
- VCC_I/O, 2-14
- VCC_RTC, 2-14
- WBMSTR2–WBMSTR0, 2-13

SIN2–SIN1 signals
- description, 2-9
- usage, 21-2

Slave DMA Channel 0–3 Control (SLDMACTL) register
- function, 14-7

Slave DMA Channel 0–3 Mask (SLDMAMSK) register
- function, 14-7

Slave DMA Channel 0–3 Mode (SLDMAMODE) register
- function, 14-7
- usage, 14-14

Slave DMA Channel 0–3 Status (SLDMASTA) register
- function, 14-7

Slave DMA Channel x Memory Address (GPDMAxMAR) register
- function, 14-7
- usage, 14-11, 14-12, 14-18

Slave DMA Channel x Page (GPDMAxPG) register
- function, 14-7
- usage, 14-11, 14-12

Slave DMA Channel x Transfer Count (GPDMAxTC) register
- function, 14-7
- usage, 14-9, 14-18

Slave DMA Clear Byte Pointer (SLDMACBP) register
- function, 14-7

Slave DMA Controller Reset (SLDMARST) register
- function, 14-8
- usage, 14-19

Slave DMA Controller Temporary (SLDMATMP) register
- function, 14-8

Slave DMA General Mask (SLDMAGENMSK) register
- function, 14-8
- usage, 14-19

Slave DMA Mask Reset (SLDMAMSKRST) register
- function, 14-8

Slave Software DRQ(n) Request (SLDMASWREQ) register
- function, 14-7

Slave x PIC Initialization Control Word 1 (SxPICICW1) register
- function, 15-6
- usage, 3-19, 15-16

Slave x PIC Initialization Control Word 2 (SxPICICW2) register
- function, 15-6
- usage, 15-16, 15-18

- Slave x PIC Initialization Control Word 3 (SxPICICW3) register
  - function, 15-7
  - usage, 15-16, 15-17
- Slave x PIC Initialization Control Word 4 (SxPICICW4) register
  - function, 15-7
  - usage, 15-17, 15-18
- Slave x PIC In-Service (SxPICISR) register
  - function, 15-6
  - usage, 15-17
- Slave x PIC Interrupt Mask (SxPICINTMSK) register
  - function, 15-7
  - usage, 15-17
- Slave x PIC Interrupt Mode (SLxPICMODE) register
  - function, 15-4
  - usage, 3-19
- Slave x PIC Interrupt Request (SxPICIR) register
  - function, 15-6
- Slave x PIC Operation Control Word 2 (SxPICOCW2) register
  - function, 15-6
  - usage, 15-17
- Slave x PIC Operation Control Word 3 (SxPICOCW3) register
  - function, 15-6
  - usage, 15-17
- SLDMACBP register, 14-7
- SLDMAGENMSK register, 14-8
- SLDMAMODE register, 14-7
- SLDMAMSK register, 14-7
- SLDMAMSKRST register, 14-8
- SLDMARST register, 14-8
- SLDMASTA register, 14-7
- SLDMASWREQ register, 14-7
- SLDMATMP register, 14-8
- SLxPICMODE register, 15-4
- SNGL bit field, 15-18
- Software Interrupt 16–1 Control (SWINT16_1) register
  - function, 15-4
  - usage, 15-8
- Software Interrupt 22–17/NMI Control (SWINT22_17) register
  - function, 15-4
  - usage, 15-8
- software timer
  - block diagram (figure), 18-1
  - configuration, 18-2
  - initialization, 18-3
  - operation, 18-2
  - overview, 18-1
  - registers, 18-2
- Software Timer Configuration (SWTMRCFG) register
  - function, 5-6, 18-2
  - usage, 5-8, 18-3
- Software Timer Microsecond Count (SWTMRMICRO) register
  - function, 18-2
  - usage, 18-2
- Software Timer Millisecond Count (SWTMRMILLI) register
  - function, 18-2
  - usage, 18-2
- SOUT2–SOUT1 signals
  - description, 2-9
  - usage, 21-2
- SP bit field, 21-8
- $\overline{\text{SRASA}}$–$\overline{\text{SRASB}}$ signals
  - control, 10-10, 10-19, 23-4
  - description, 2-5
  - usage, 10-5, 10-8
- sreset CPU signal, 6-7
- SSI. *See* synchronous serial interface (SSI).
- SSI Clock signal. *See* SSI_CLK signal.
- SSI Command (SSICMD) register
  - function, 22-2
  - usage, 22-7, 22-8
- SSI Control (SSICTL) register
  - function, 5-6, 22-2
  - usage, 5-8, 22-3, 22-5, 22-7, 22-8
- SSI Data Input signal. *See* SSI_DI signal.
- SSI Data Output signal. *See* SSI_DO signal.
- SSI Interrupt Mapping (SSIMAP) register
  - function, 15-5, 22-2
- SSI Receive (SSIRCV) register
  - function, 22-2
  - usage, 22-3, 22-8
- SSI Status (SSISTA) register
  - function, 22-2
  - usage, 22-7, 22-8
- SSI Transmit (SSIXMIT) register
  - function, 22-2
  - usage, 22-3, 22-8
- SSI_CLK signal
  - description, 2-10
  - usage, 5-8, 22-1, 22-5, 22-7
- SSI_DI signal
  - description, 2-10
  - usage, 22-1, 22-3, 22-5
- SSI_DO signal
  - description, 2-10
  - usage, 22-1, 22-3, 22-5
- SSICMD register, 22-2
- SSICTL register, 22-2
- SSIMAP register, 15-5

- SSIRCV register, 22-2
- SSISTA register, 22-2
- SSIXMIT register, 22-2
- Status/Command (PCISTACMD) register
  - function, 9-8
  - usage, 9-10, 9-18, 9-27
- $\overline{\text{STOP}}$ signal
  - description, 2-7
  - usage, 9-3
- Stop/Transmit signal. *See* STOP/TX signal.
- STOP/TX signal
  - control, 25-4
  - description, 2-12
- SUB_DLY bit field, 12-8
- $\overline{\text{SWEA}}$–$\overline{\text{SWEB}}$ signals
  - control, 10-10, 10-19, 23-4
  - description, 2-5
  - usage, 10-5
- SWINT16_1 register, 15-4
- SWINT22_17 register, 15-4
- SWT. *See* software timer.
- SWTMRCFG register, 18-2
- SWTMRMICRO register, 18-2
- SWTMRMILLI register, 18-2
- SxPICICW1 register, 15-6
- SxPICICW2 register, 15-6
- SxPICICW3 register, 15-7
- SxPICICW4 register, 15-7
- SxPICINTMSK register, 15-7
- SxPICIR register, 15-6
- SxPICISR register, 15-6
- SxPICOCW2 register, 15-6
- SxPICOCW3 register, 15-6
- synchronous DRAM. *See* SDRAM controller.
- synchronous serial interface (SSI)
  - block diagram (figure), 22-2
  - bus cycles, 22-5
    - 4-bit read cycle, 22-6
    - burst, 16-bit, and 32-bit cycles, 22-7
    - clock phase and clock idle state (figure), 22-6
    - full-duplex, back-to-back transactions (figure), 22-7
    - full-duplex, non-inverted phase, non-inverted clock (figure), 22-6
    - half-duplex, non-inverted phase and clock modes (figure), 22-4
    - simultaneous transmit and receive (figure), 22-4
    - TC_INT and BSY_STA timing (figure), 22-8
  - clocking considerations, 22-7
  - configuration, 22-5
    - bit order, 22-5
    - clock idle state, 22-5
    - clock phase, 22-5
  - initialization, 22-8
  - interrupts, 22-7
  - operation, 22-3
  - overview, 22-1
  - registers, 22-2
  - signal descriptions, 2-9
  - software considerations, 22-8
  - system design, 22-1
    - four-pin interface (figure), 22-4
    - three-pin interface (figure), 22-4
- SYS_RST bit field, 6-4, 6-5
- SYSARBCTL register, 8-2
- SYSARBMENB register, 8-2
- SYSCTLA register, 6-3
- SYSCTLB register, 16-3
- SYSINFO register, 6-3
- System Arbiter Control (SYSARBCTL) register
  - function, 8-2, 24-2
  - usage, 8-3, 8-10, 8-19, 8-22, 24-10
- System Arbiter Master Enable (SYSARBMENB) register
  - function, 8-2
  - usage, 8-19, 8-23
- system arbitration
  - arbitration mode changes, 8-19
  - block diagram (figure), 8-2
  - broken transactions, 8-19
  - bus cycles, 8-11
    - CPU bus arbitration (figure), 8-11
    - CPU bus cache write-back (figure), 8-12
    - CPU-to-PCI cycle (figure), 8-14
    - nonconcurrent mode arbitration (figure), 8-18
    - PCI bus arbitration (figure), 8-15
    - PCI bus arbitration parking (figure), 8-16
  - CPU bus arbiter, 8-5
    - accessing the PCI host bridge target, 8-6
    - arbitration protocol, 8-5
    - cache snooping, 8-6
    - clock speed changes, 8-7
    - GP-DMA arbitration, 8-7
    - rotating priority queue (figure), 8-6
    - skipped master example (figure), 8-5
  - initialization, 8-22
  - interrupts, 8-19
  - latency, 8-20
    - concurrent arbitration mode, 8-22
    - concurrent arbitration mode bus parking, 8-22
    - CPU, 8-21
    - high-priority queue, 8-21
    - low-priority queue, 8-21
    - nonconcurrent arbitration mode, 8-21
    - simple rotating priority, 8-20
    - simple rotating priority queue (figure), 8-20
  - operating modes, 8-3
    - concurrent arbitration mode, 8-4
    - nonconcurrent arbitration mode, 8-3

  - operation, 8-3
  - PCI bus arbiter, 8-7
    - arbitration protocol, 8-8
    - bus parking, 8-10
    - external PCI master queues (figure), 8-9
    - host bridge master queue (figure), 8-9
    - rearbitration, 8-10
  - registers, 8-2
  - software considerations, 8-19
  - write posting, 8-19

- System Board Information (SYSINFO) register
  - function, 6-3
  - usage, 19-6
- System Control Port A (SYSCTLA) register
  - function, 6-3, 7-1
  - usage, 6-7, 6-8
- System Control Port B (SYSCTLB) register
  - function, 16-3
  - usage, 16-4
- System Error signal. *See* $\overline{\text{SERR}}$ signal.
- system initialization
  - BIOS initialization sequence, 3-3
  - configuration register access, 4-20
  - CPU core identification, 3-7
  - CPU speed, 3-7
  - disabling internal peripherals, 3-21
  - external GP bus devices, 3-7
    - multiple devices on one chip select, 3-14
    - single device performing its own decode, 3-14
    - single device using one chip select, 3-14
  - external ROM devices
    - boot device mapping for BIOS shadowing, 3-17
    - two Flash banks for XIP operating system, 3-17
  - interrupt mapping, 3-19
    - edge-sensitive or level-triggered interrupts, 3-19
    - interrupt polarity, 3-20
  - memory-mapped configuration region (MMCR), 3-3
  - native embedded initialization sequence, 3-1
  - PCI bus devices
    - network adapter, 3-16
    - VGA controller on PCI bus, 3-15
  - PCI host bridge and arbitration, 3-20
  - pin multiplexing, 3-8
  - Programmable Address Region x (PARx) regions
    - specifying pages and regions, 3-9
  - Programmable Address Region x (PARx)
    - registers, 3-8
    - address region attributes, 3-12
    - cacheability control, 3-12
    - code execution control, 3-12
    - external GP bus devices, 3-13
    - external ROM devices, 3-17
    - format (figure), 3-10
    - PAR register priority, 3-13
    - PCI bus devices, 3-15
    - performance considerations of attributes, 3-12
    - SDRAM regions, 3-18
    - worksheet (figure), 3-11
    - write-protection, 3-12
  - programmable I/O pins, 3-20
  - reset event, 3-4
  - reset vector and reset segment, 3-5
    - initial near jump example (figure), 3-6
  - SDRAM regions
    - configuring DMA buffers, 3-18
    - write-protected code segments, 3-18
- system reset, 6-6
- system test and debugging
  - cache mode control, 24-10
  - CPU clock speed control, 24-10
  - disabling write buffer and read buffer, 24-10
  - ECC check code override, 24-11
  - echoing integrated peripheral accesses, 24-10
  - execution control violation notification, 24-11
  - forcing software interrupts, 24-11
  - initialization, 24-12
  - interrupt masking, 24-11
  - latency, 24-11
  - nonconcurrent arbitration mode, 24-10
  - operation, 24-3
  - overview, 24-1
  - registers, 24-2
  - reset source identification, 24-11
  - software considerations, 24-11
  - system design, 24-1
    - loading, 24-2
    - logic analyzer use, 24-2
    - shared signals (table), 24-2
  - system test mode, 24-3
    - pin functions, 24-3
    - SDRAM read cycle (figure), 24-5
    - SDRAM write cycle (figure), 24-4
    - tracing transactions on GP bus, 24-6
    - tracing transactions on ROM interface, 24-5
  - write buffer test mode, 24-7
    - SDRAM read cycle (figure), 24-8
    - SDRAM write cycle (figure), 24-8
    - WBMSTR2–WBMSTR0 during SDRAM read cycles (table), 24-9
    - WBMSTR2–WBMSTR0 during write buffer write cycles (table), 24-8
  - write protection violation notification, 24-11
- SZ_ST_ADR bit field, 3-9, 3-10, 4-5

T

- T_DLYTR_ENB bit field, 9-19
- T_IRQ_ID bit field, 9-27
- T_PURGE_RD_ENB bit field, 9-22
- T7–T3 bit field, 15-18
- TARGET bit field, 3-10, 4-5, 4-8, 4-16, 4-19, 9-18

Target Ready signal. *See* $\overline{\text{TRDY}}$ signal.
TC_INT bit field, 22-7
TC_INT_ENB bit field, 22-7
technical support, iii
TEMT bit field, 21-7
TERI bit field, 21-6
test access port (TAP) controller. *See* JTAG boundary scan test interface.
Test Clock Output signal. *See* CLKTEST signal.
Test Data Input signal. *See* JTAG_TDI signal.
Test Data Output signal. *See* JTAG_TDO signal.
Test Mode Select signal. *See* JTAG_TMS signal.
testing. *See* JTAG boundary scan test interface.
    See also system test and debugging.
    See also AMDebug™ technology.
TF_CLR bit field, 21-10
third-party support products, iii
THRE bit field, 21-6
Timer Clock Input signal. *See* CLKTIMER signal.
Timer Input 0 and 1 signals.
    See TMRIN1–TMRIN0 signals.
Timer Output 0 and 1 signals.
    See TMROUT1–TMROUT0 signals.
TMRIN1–TIMIN0 signals
  control, 13-3, 13-6, 17-1
  description, 2-10
  usage, 17-3, 17-4, 17-6
TMROUT1–TMROUT0 signals
  control, 13-3, 13-6, 17-1
  description, 2-10
  usage, 14-3, 17-5
$\overline{\text{TRDY}}$ signal
  description, 2-7
  usage, 9-3, 9-20, 9-21
TRIG/TRACE signal
  control, 25-4
  description, 2-12
Trigger/Trace signal. *See* TRIG/TRACE signal.
TRNMOD bit field, 14-11, 14-19

U

UART serial ports
  baud rates, divisors, and clock source (table), 21-9
  block diagram (figure), 21-2
  clocking considerations, 21-10
  configuration, 21-9
    baud rate, 21-9
    hardware flow control, 21-9
    operating modes, 21-9
  data reception, 21-7
  data transmission, 21-6
  disabling, 3-21
  DMA interface, 21-10
    receive DMA, 21-10
    transmit DMA, 21-10
    UART as GP-DMA initiator, 14-9
  error handling, 21-8
    break indication, 21-8
    error reporting, 21-8
    framing error, 21-8
    parity error, 21-8
  frame configuration (figure), 21-5
  frame transmission (figure), 21-5
  GP-DMA channel mapping (table), 14-10
  initialization, 21-13
  interrupts, 21-10
    DMA interrupts, 21-12
    interrupt disable, 21-13
    interrupt priority (table), 21-12
    interrupt programming summary (table), 21-11
    serial port interrupts, 21-12
  operation, 21-5
  overview, 21-1
  registers, 21-3
  signal descriptions, 2-9
  system design, 21-2
    connection of DTE to DTE (table), 21-3
    shared signals (table), 21-2
UART x Baud Clock Divisor Latch LSB (UARTxBCDL) register
  function, 21-4
  usage, 21-9
UART x Baud Clock Divisor Latch MSB (UARTxBCDH) register
  function, 21-4
  usage, 21-9
UART x FIFO Control (UARTxFCR) register
  function, 21-4
  usage, 21-3, 21-7, 21-9, 21-10, 21-13
UART x FIFO Control Shadow (UARTxFCRSHAD) register
  function, 21-3
  usage, 21-13
UART x General Control (UARTxCTL) register
  function, 5-6, 21-3
  usage, 5-8, 21-9, 21-10, 21-11, 21-12
UART x General Status (UARTxSTA) register
  function, 21-3
  usage, 21-11, 21-12
UART x Interrupt Enable (UARTxINTENB) register
  function, 21-4
  usage, 21-9, 21-11
UART x Interrupt ID (UARTxINTID) register
  function, 21-4
  usage, 21-11, 21-12
UART x Interrupt Mapping (UARTxMAP) register
  function, 15-5, 21-3, 21-11

UART x Line Control (UARTxLCR) register
  function, 21-4
  usage, 21-5, 21-8
UART x Line Status (UARTxLSR) register
  function, 21-4
  usage, 21-6, 21-7, 21-8, 21-10, 21-11
UART x Modem Control (UARTxMCR) register
  function, 21-4
  usage, 21-6, 21-11, 21-13
UART x Modem Status (UARTxMSR) register
  function, 21-4
  usage, 21-6, 21-11
UART x Receive Buffer (UARTxRBR) register
  function, 21-4
  usage, 21-7, 21-8, 21-10
UART x Scratch Pad (UARTxSCRATCH) register
  function, 21-4
UART x Transmit Holding (UARTxTHR) register
  function, 21-4
  usage, 21-5, 21-6, 21-7, 21-10
UART1_DIS bit field, 3-21
UART2_DIS bit field, 3-21
UARTxBCDH register, 21-4
UARTxBCDL register, 21-4
UARTxCTL register, 21-3
UARTxFCR register, 21-4
UARTxFCRSHAD register, 21-3
UARTxINTENB register, 21-4
UARTxINTID register, 21-4
UARTxLCR register, 21-4
UARTxLSR register, 21-4
UARTxMAP register, 15-5
UARTxMCR register, 21-4
UARTxMSR register, 21-4
UARTxRBR register, 21-4
UARTxSCRATCH register, 21-4
UARTxSTA register, 21-3
UARTxTHR register, 21-4
UIP bit field, 20-7
universal asynchronous receiver/transmitter (UART).
    See UART serial ports.

V

VCC_ANLG signal
  description, 2-14
  usage, 5-3, 20-4
VCC_CORE signal
  description, 2-14
VCC_I/O signal
  description, 2-14
VCC_RTC signal
  usage, 2-10, 2-14, 6-9, 20-3, 20-4, 20-11

W

watchdog timer (WDT)
  AMDebug™ technology interface, 19-5
  block diagram (figure), 19-2
  configuration, 19-3
    interrupt request generation, 19-4
    keyed sequences, 19-3
    system reset generation, 19-4
    time-out duration, 19-4
  initialization, 19-6
  interrupts, 19-5
  operation, 19-3
  overview, 19-1
  registers, 19-2
  software considerations, 19-5
  time-out duration (table), 19-4
Watchdog Timer Control (WDTMRCTL) register
  function, 6-3, 19-2
  usage, 5-8, 19-3, 19-4, 19-5, 19-6
Watchdog Timer Count High (WDTMRCNTH) register
  function, 19-3
  usage, 19-6
Watchdog Timer Count Low (WDTMRCNTL) register
  function, 19-2
  usage, 19-6
Watchdog Timer Interrupt Mapping (WDTMAP) register
  function, 15-5, 19-3
WB_ENB bit field, 24-10
WB_FLUSH bit field, 11-5, 11-13
WB_TST_ENB bit field, 10-19, 24-2, 24-3, 24-7
WB_WM bit field, 11-9
WBMSTR2–WBMSTR0 signals
  control, 24-2
  description, 2-13
  usage, 10-19, 11-4, 24-1, 24-7, 24-8, 24-12
WDT. *See* watchdog timer (WDT).
WDT_RST_DET bit field, 6-8
WDTMAP register, 15-5
WDTMRCNTH register, 19-3, 19-6
WDTMRCNTL register, 19-2
WDTMRCTL register, 19-2
web site, iii
WIDTH bit field, 12-7
WPVMAP register, 15-5
WPVSTA register, 4-2
write buffer and read buffer
  block diagram (figure), 11-2
  data coherency, 11-13
  disabling during SDRAM sizing or test, 11-13

DMA considerations, 11-11
initialization, 11-15
operation, 11-4
PCI considerations, 11-12
read cycles, 11-12
write cycles, 11-12
read buffer, 11-10
disabled, 11-10
enabled, 11-10
registers, 11-4
SDRAM bandwidth improvements, 11-13
bus thrashing (figure), 11-14
software considerations, 11-13
system design, 11-3
shared signals (table), 11-4
write buffer, 11-5
disabled, 11-5
enabled, 11-5
read-merging, 11-8
read-merging example (figure), 11-9
watermark, 11-9
write-collapsing, 11-6
write-collapsing example (figure), 11-8
write-merging, 11-6

Write Buffer Master signals.
See WBMSTR2–WBMSTR0 signals.

Write-Protect Violation Interrupt Mapping (WPVMAP) register
function, 15-5

Write-Protect Violation Status (WPVSTA) register
function, 4-2
usage, 4-20

write-protection, 3-12

WRST_ENB bit field, 19-4

WRTSEL bit field, 14-9

X

XTAL_FREQ bit field, 18-3