April 2000

National Semiconductor®

# Geode™ GXLV Processor Series Low Power Integrated x86 Solutions

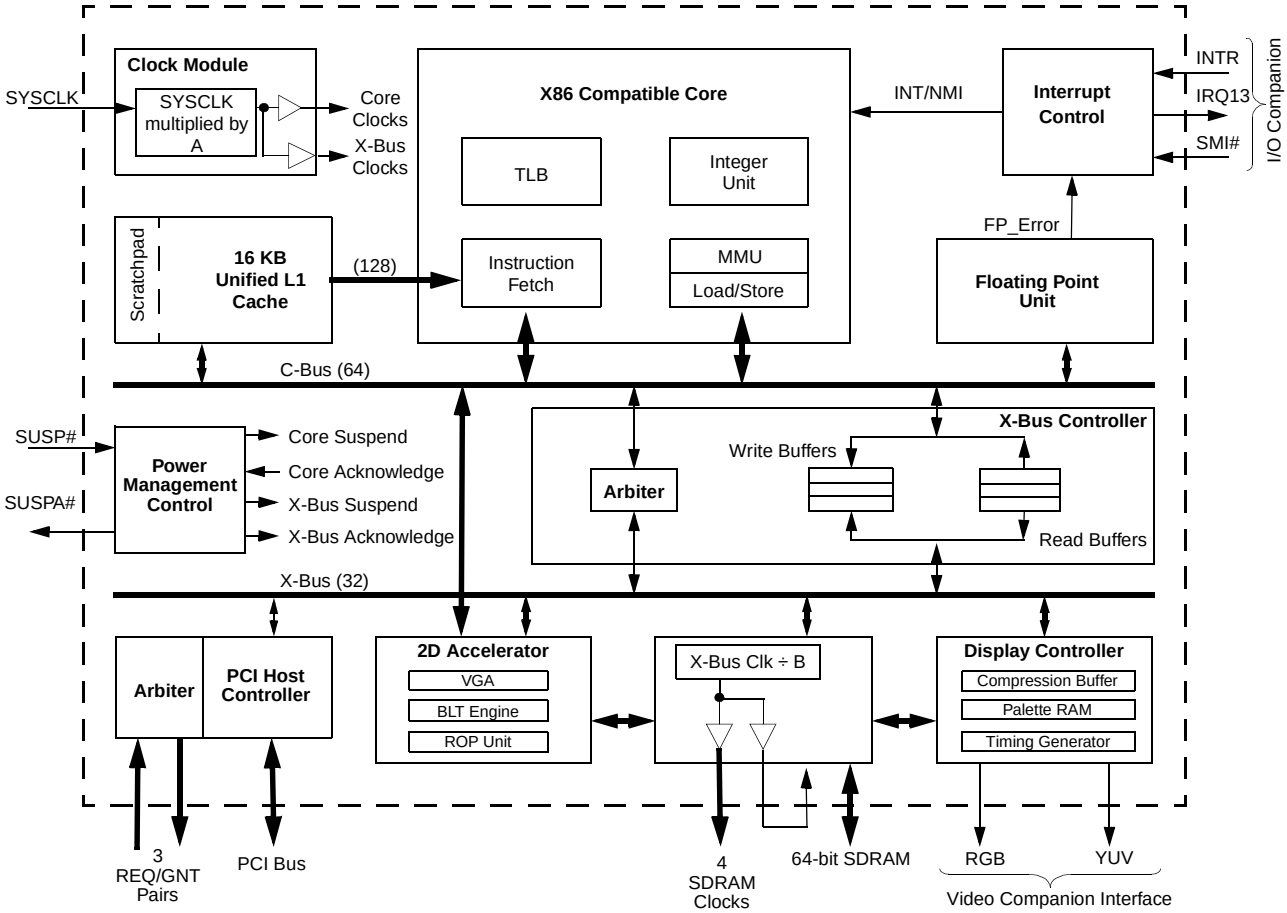
## General Description

The National Semiconductor® Geode™ GXLV processor series is a new line of integrated processors specifically designed to power information appliances for entertainment, education, and business. Serving the needs of consumers and business professionals alike, it is the perfect solution for information appliance applications such as thin clients, interactive set top boxes, and personal internet access devices.

The GXLV processor series is divided into three main categories as defined by the core operating voltage. Available with core voltages of 2.2V, 2.5V, and 2.9V, it offers extremely low typical power consumption (1.0W to 2.5W) leading to longer battery life and enabling small form-factor, fanless designs. Each core voltage is offered in frequencies that are enabled by specific system clock and multiplier settings. This allows the user to select the device(s) that best fit their power and performance requirements. This flexibility makes the GXLV processor series ideally suited for applications where power consumption and performance (speed) are equally important.

Typical power consumption is defined as an average, measured running Microsoft's Windows at 80% Active Idle (Suspend-on-Halt) with a display resolution of 800x600x8 bpp at 75 Hz.

## Internal Block Diagram




National Semiconductor is a registered trademark of National Semiconductor Corporation.
Geode and WebPAD are trademarks of National Semiconductor Corporation.
For a complete listing of National Semiconductor trademarks, please visit www.national.com/trademarks.

While the x86 core provides maximum compatibility with the vast amount of internet content available, the intelligent integration of several other functions, such as memory controller and graphics, offer a true system-level multimedia solution.

The GXLV processor core is a proven x86 design that offers competitive performance. It contains integer and floating point execution units based on sixth-generation technology. The integer core contains a single, five-stage execution pipeline and offers advanced features such as operand forwarding, branch target buffers, and extensive write buffering. Accesses to the 16 KB write-back L1 cache are dynamically reordered to eliminate pipeline stalls when fetching operands.

In addition to the advanced CPU features, the GXLV processor integrates a host of functions typically implemented with external components. A full function graphics accelerator contains a Video Graphics Array (VGA) controller, bitBLT engine, and a Raster Operations (ROP) unit for complete Graphical User Interface (GUI) acceleration under most operating systems. A display controller contains additional video buffering to enable >30 fps MPEG1 playback and video overlay when used with a National Semiconductor I/O Companion chip such as the CS5530. Graphics and system memory accesses are supported by a tightly coupled SDRAM controller which eliminates the need for an external L2 cache. A PCI host controller supports up to three bus masters for additional connectivity and multimedia capabilities.

The GXLV processor also incorporates Virtual System Architecture® (VSA™) technology. VSA technology enables the XpressGRAPHICS and XpressAUDIO subsystems. Software handlers are available that provide full compatibility for industry standard VGA and 16-bit audio functions that are transparent at the operating system level.

The GXLV processor is designed to be used with the CS5530 I/O Companion, also supplied by National Semiconductor. Together they provide a scalable, flexible, low-power, system-level solution well suited for a wide array of information appliances ranging from hand-held personal information access devices to digital set top boxes and thin clients.

## Features

### General Features

- Packaging:
  - — 352-Terminal Ball Grid Array (BGA) or
  - — 320-Pin Staggered Pin Grid Array (SPGA)
- 0.25-micron four layer metal CMOS process
- Split rail design:
  - — Available 2.2V, 2.5V, or 2.9V core
  - — 3.3V I/O interface (5V tolerant)
- Low typical power consumption:
  - — 1.0W @ 2.2V/166 MHz
  - — 2.5W @ 2.9V/266 MHz

**Note:** Typical power consumption is defined as an average, measured running Windows at 80% Active Idle (Suspend-on-Halt) with a display resolution of 800x600x8 bpp @ 75 Hz.

- Speeds offered up to 266 MHz
- Unified Memory Architecture:
  - — Frame buffer and video memory reside in main memory
  - — Minimizes Printed Circuit Board (PCB) area requirements
  - — Reduces system cost
- Compatible with multiple Geode I/O companion devices provided by National Semiconductor

### 32-Bit x86 Processor

- Supports Intel's MMX instruction set extension for the acceleration of multimedia applications
- 16 KB unified L1 cache
- Five-stage pipelined integer unit
- Integrated Floating Point Unit (FPU)
- Memory Management Unit (MMU) adheres to standard paging mechanisms and optimizes code fetch performance:
  - — Load-store reordering gives priority to memory reads
  - — Memory-read bypassing eliminates unnecessary or redundant memory reads
- Re-entrant System Management Mode (SMM) enhanced for VSA technology
- Fully Static Design

### Flexible Power Management

- Supports a wide variety of standards:
  - — APM for Legacy power management
  - — ACPI for Windows power management
    - – Direct support for all standard processor (C0-C4) states
  - — OnNOW specification compliant
- Supports a wide variety of hardware and software controlled modes:
  - — Fully Active
  - — Active Idle (core stopped, display active)
  - — Standby (core and all integrated functions halted)
  - — Sleep (core and integrated functions halted and all external clocks stopped)
  - — Suspend Modulation (automatic throttling of CPU core)
    - – Programmable duty cycle for optimal performance/thermal balancing
  - — Several dedicated and programmable wake-up events (via Geode I/O companion chip)

### PCI Host Controller

- Several arbitration schemes supported
- Supports up to three PCI bus masters
- Synchronous to CPU core
- Allows external PCI master accesses to main memory concurrent with CPU accesses to L1 cache

### Virtual Systems Architecture Technology

- Innovative architecture allowing OS independent (software) virtualization of hardware functions
- Provides XpressGRAPHICS subsystem:
  — High performance legacy VGA core compatibility

**Note:** Uses 2D Graphics Accelerator.

- Provides 16-bit XpressAUDIO subsystem:
  — 16-bit stereo FM synthesis
  — OPL3 emulation
  — Supports MPU-401 MIDI interface
  — Hardware assist provided via Geode I/O companion chip
- Additional hardware functions can be supported as needed

### 2D Graphics Accelerator

- Accelerates BitBLTs, line draw, text
- Bresenham vector engine
- Supports all 256 ROPs
- Supports transparent BLTs and page flipping for Microsoft's DirectDraw
- Runs at core clock frequency
- Full VGA and VESA mode support
- Special "driver level" instructions utilize internal scratchpad for enhanced performance

### Display Controller

- Display Compression Technology (DCT) architecture greatly reduces memory bandwidth consumption of display refresh
- Supports a separate video buffer and data path to enable video acceleration in Geode I/O companion devices
- Internal palette RAM for gamma correction
- Direct interface to Geode I/O companion devices for CRT and TFT flat panel support eliminates the need for an external RAMDAC
- Hardware cursor
- Supports up to 1280x1024x8 bpp and 1024x768x16 bpp

### XpressRAM Subsystem

- SDRAM interface tightly coupled to CPU core and graphics subsystem for maximum efficiency
- 64-Bit wide memory bus
- Support for:
  — Two 168-pin unbuffered DIMMs
  — Up to 16 simultaneously open banks
  — 16-byte reads (burst length of two)
  — Up to 256 MB total memory supported

### Diverse Operating System Support

- Microsoft's Windows 2000, 9X, NT, and CE
- Sun Microsystems' Java
- WindRiver Systems' VxWorks
- QNX Software Systems' QNX
- Linux

# Table of Contents


**1.0 Architecture Overview . . . 10**

1.1 INTEGER UNIT . . . 11
1.2 FLOATING POINT UNIT . . . 11
1.3 WRITE-BACK CACHE UNIT . . . 11
1.4 MEMORY MANAGEMENT UNIT . . . 11
1.5 INTERNAL BUS INTERFACE UNIT . . . 11
1.6 INTEGRATED FUNCTIONS . . . 12
1.6.1 Graphics Accelerator . . . 12
1.6.2 Display Controller . . . 12
1.6.3 XpressRAM Memory Subsystem . . . 12
1.6.4 PCI Controller . . . 12
1.7 GEODE GXLV/CS5530 SYSTEM DESIGNS . . . 13
1.7.1 Reference Designs . . . 16

**2.0 Signal Definitions . . . 19**

2.1 PIN ASSIGNMENTS . . . 20
2.2 SIGNAL DESCRIPTIONS . . . 31
2.2.1 System Interface Signals . . . 31
2.2.2 PCI Interface Signals . . . 33
2.2.3 Memory Controller Interface Signals . . . 36
2.2.4 Video Interface Signals . . . 37
2.2.5 Power, Ground, and No Connect Signals . . . 39
2.2.6 Internal Test and Measurement Signals . . . 39

**3.0 Processor Programming . . . 41**

3.1 CORE PROCESSOR INITIALIZATION . . . 41
3.2 INSTRUCTION SET OVERVIEW . . . 42
3.2.1 Lock Prefix . . . 42
3.3 REGISTER SETS . . . 43
3.3.1 Application Register Set . . . 43
3.3.1.1 General Purpose Registers . . . 43
3.3.1.2 Segment Registers . . . 45
3.3.1.3 Instruction Pointer Register . . . 45
3.3.1.4 EFLAGS Register . . . 46
3.3.2 System Register Set . . . 47
3.3.2.1 Control Registers . . . 48
3.3.2.2 Configuration Registers . . . 50
3.3.2.3 Debug Registers . . . 55
3.3.2.4 TLB Test Registers . . . 57
3.3.2.5 Cache Test Registers . . . 59
3.3.3 Model Specific Register Set . . . 62
3.3.4 Time Stamp Counter . . . 62
3.4 ADDRESS SPACES . . . 63
3.4.1 I/O Address Space . . . 63
3.4.2 Memory Address Space . . . 64

## Table of Contents (Continued)


3.5 OFFSET, SEGMENT, AND PAGING MECHANISMS . . . 64
3.5.1 Offset Mechanism . . . 64
3.5.2 Segment Mechanisms . . . 66
3.5.2.1 Real Mode Segment Mechanism . . . 66
3.5.2.2 Virtual 8086 Mode Segment Mechanism . . . 66
3.5.2.3 Segment Mechanism in Protected Mode . . . 67
3.5.2.4 Segment Selectors . . . 67
3.5.3 Descriptors . . . 70
3.5.3.1 Global and Local Descriptor Table Registers . . . 70
3.5.3.2 Segment Descriptors . . . 70
3.5.3.3 Task, Gate, Interrupt, and Application and System Descriptors . . . 71
3.5.4 Paging Mechanism . . . 77
3.6 INTERRUPTS AND EXCEPTIONS . . . 79
3.6.1 Interrupts . . . 79
3.6.2 Exceptions . . . 79
3.6.3 Interrupt Vectors . . . 80
3.6.3.1 Interrupt Vector Assignments . . . 80
3.6.3.2 Interrupt Descriptor Table . . . 80
3.6.4 Interrupt and Exception Priorities . . . 81
3.6.5 Exceptions in Real Mode . . . 82
3.6.6 Error Codes . . . 82
3.7 SYSTEM MANAGEMENT MODE . . . 83
3.7.1 SMM Operation . . . 84
3.7.2 SMI# Pin . . . 85
3.7.3 SMM Configuration Registers . . . 85
3.7.4 SMM Memory Space Header . . . 85
3.7.5 SMM Instructions . . . 87
3.7.6 SMM Memory Space . . . 88
3.7.7 SMI Generation for Virtual VGA . . . 88
3.7.8 SMM Service Routine Execution . . . 88
3.7.8.1 SMI Nesting . . . 88
3.7.8.2 CPU States Related to SMM and Suspend Mode . . . 90
3.8 HALT AND SHUTDOWN . . . 91
3.9 PROTECTION . . . 91
3.9.1 Privilege Levels . . . 91
3.9.2 I/O Privilege Levels . . . 91
3.9.3 Privilege Level Transfers . . . 92
3.9.3.1 Gates . . . 93
3.9.4 Initialization and Transition to Protected Mode . . . 93
3.10 VIRTUAL 8086 MODE . . . 93
3.10.1 Memory Addressing . . . 93
3.10.2 Protection . . . 93
3.10.3 Interrupt Handling . . . 93
3.10.4 Entering and Leaving Virtual 8086 Mode . . . 93
3.11 FLOATING POINT UNIT OPERATIONS . . . 94
3.11.1 FPU Register Set . . . 94
3.11.2 FPU Tag Word Register . . . 94
3.11.3 FPU Status Register . . . 94
3.11.4 FPU Mode Control Register . . . 94

## Table of Contents (Continued)


**4.0 Integrated Functions . . . . . 96**

4.1 INTEGRATED FUNCTIONS PROGRAMMING INTERFACE . . . . . 97
4.1.1 Graphics Control Register . . . . . 97
4.1.2 Control Registers . . . . . 99
4.1.3 Graphics Memory . . . . . 99
4.1.4 Scratchpad RAM . . . . . 100
4.1.4.1 Initialization of Scratchpad RAM . . . . . 100
4.1.4.2 Scratchpad RAM Utilization . . . . . 100
4.1.4.3 BLT Buffer . . . . . 100
4.1.5 Display Driver Instructions . . . . . 102
4.1.6 CPU_READ/CPU_WRITE Instructions . . . . . 102
4.2 INTERNAL BUS INTERFACE UNIT . . . . . 103
4.2.1 FPU Error Support . . . . . 103
4.2.2 A20M Support . . . . . 103
4.2.3 SMI Generation . . . . . 103
4.2.4 640 KB to 1 MB Region . . . . . 103
4.2.5 Internal Bus Interface Unit Registers . . . . . 104
4.3 MEMORY CONTROLLER . . . . . 107
4.3.1 Memory Array Configuration . . . . . 108
4.3.2 Memory Organizations . . . . . 109
4.3.3 SDRAM Commands . . . . . 110
4.3.3.1 SDRAM Initialization Sequence . . . . . 111
4.3.4 Memory Controller Register Description . . . . . 112
4.3.5 Address Translation . . . . . 117
4.3.5.1 High Order Interleaving . . . . . 117
4.3.5.2 Auto Low Order Interleaving . . . . . 117
4.3.5.3 Physical Address to DRAM Address Conversion . . . . . 117
4.3.6 Memory Cycles . . . . . 120
4.3.7 SDRAM Interface Clocking . . . . . 123
4.4 GRAPHICS PIPELINE . . . . . 125
4.4.1 BitBLT/Vector Engine . . . . . 125
4.4.2 Master/Slave Registers . . . . . 126
4.4.3 Pattern Generation . . . . . 126
4.4.3.1 Monochrome Patterns . . . . . 127
4.4.3.2 Dither Patterns . . . . . 127
4.4.3.3 Color Patterns . . . . . 128
4.4.4 Source Expansion . . . . . 128
4.4.5 Raster Operations . . . . . 128
4.4.6 Graphics Pipeline Register Descriptions . . . . . 129

## Table of Contents (Continued)


4.5 DISPLAY CONTROLLER . . . 134
4.5.1 Display FIFO . . . 135
4.5.2 Compression Technology . . . 135
4.5.3 Hardware Cursor . . . 136
4.5.4 Display Timing Generator . . . 136
4.5.5 Dither and Frame Rate Modulation . . . 136
4.5.6 Display Modes . . . 136
4.5.7 Graphics Memory Map . . . 140
4.5.7.1 DC Memory Organization Registers . . . 140
4.5.7.2 Frame Buffer and Compression Buffer Organization . . . 140
4.5.7.3 VGA Display Support . . . 141
4.5.8 Display Controller Registers . . . 141
4.5.8.1 Configuration and Status Registers . . . 144
4.5.9 Memory Organization Registers . . . 148
4.5.10 Timing Registers . . . 150
4.5.11 Cursor Position and Miscellaneous Registers . . . 152
4.5.12 Palette Access Registers . . . 153
4.5.13 FIFO Diagnostic Registers . . . 154
4.5.14 CS5530 Display Controller Interface . . . 155
4.5.14.1 CS5530 Video Port Data Transfer . . . 156
4.6 VIRTUAL VGA SUBSYSTEM . . . 157
4.6.1 Traditional VGA Hardware . . . 157
4.6.1.1 VGA Memory Organization . . . 158
4.6.1.2 VGA Front End . . . 158
4.6.1.3 Address Mapping . . . 158
4.6.1.4 Video Refresh . . . 159
4.6.1.5 VGA Video BIOS . . . 159
4.6.2 Virtual VGA . . . 159
4.6.2.1 Datapath Elements . . . 160
4.6.2.2 GXLV VGA Hardware . . . 160
4.6.2.3 SMI Generation . . . 161
4.6.2.4 VGA Range Detection . . . 161
4.6.2.5 VGA Sequencer . . . 161
4.6.2.6 VGA Write/Read Path . . . 161
4.6.2.7 VGA Address Generator . . . 161
4.6.2.8 VGA Memory . . . 161
4.6.3 VGA Configuration Registers . . . 162
4.6.4 Virtual VGA Register Descriptions . . . 164
4.7 PCI CONTROLLER . . . 166
4.7.1 X-Bus PCI Slave . . . 166
4.7.2 X-Bus PCI Master . . . 166
4.7.3 PCI Arbiter . . . 166
4.7.4 Generating Configuration Cycles . . . 166
4.7.5 Generating Special Cycles . . . 166
4.7.6 PCI Configuration Space Control Registers . . . 167
4.7.7 PCI Configuration Space Registers . . . 168
4.7.8 PCI Cycles . . . 173
4.7.8.1 PCI Read Transaction . . . 173
4.7.8.2 PCI Write Transaction . . . 174
4.7.8.3 PCI Arbitration . . . 175
4.7.8.4 PCI Halt Command . . . 175

## Table of Contents (Continued)


**5.0 Power Management . . . . . 176**

5.1 POWER MANAGEMENT FEATURES . . . . . 176
5.1.1 System Management Mode . . . . . 176
5.1.2 Suspend-on-Halt . . . . . 176
5.1.3 CPU Suspend . . . . . 176
5.1.3.1 Suspend Modulation for Thermal Management . . . . . 177
5.1.3.2 Suspend Modulation for Power Management . . . . . 177
5.1.4 3 Volt Suspend . . . . . 177
5.1.5 GXLV Processor Serial Bus . . . . . 177
5.1.6 Advanced Power Management (APM) Support . . . . . 177
5.2 SUSPEND MODES AND BUS CYCLES . . . . . 178
5.2.1 Timing Diagram for Suspend-on-Halt . . . . . 178
5.2.2 Initiating Suspend with SUSP# . . . . . 179
5.2.3 Stopping the Input Clock . . . . . 180
5.2.4 Serial Packet Transmission . . . . . 180
5.3 POWER MANAGEMENT REGISTERS . . . . . 181

**6.0 Electrical Specifications . . . . . 184**

6.1 PART NUMBERS/PERFORMANCE CHARACTERISTICS . . . . . 184
6.2 ELECTRICAL CONNECTIONS . . . . . 185
6.2.1 Power/Ground Connections and Decoupling . . . . . 185
6.2.1.1 Power Planes . . . . . 185
6.2.2 NC-Designated Pins . . . . . 187
6.2.3 Pull-Up and Pull-Down Resistors . . . . . 187
6.2.4 Unused Input Pins . . . . . 187
6.3 ABSOLUTE MAXIMUM RATINGS . . . . . 188
6.4 RECOMMENDED OPERATING CONDITIONS . . . . . 189
6.5 DC CHARACTERISTICS . . . . . 190
6.5.1 Input/Output DC Characteristics . . . . . 190
6.5.2 DC Current . . . . . 190
6.5.2.1 Definition of CPU Power States . . . . . 190
6.5.2.2 Definition and Measurement Techniques of CPU Current Parameters . . . . . 190
6.5.2.3 Definition of System Conditions for Measuring "On" Parameters . . . . . 191
6.5.2.4 DC Current Measurements . . . . . 192
6.6 I/O CURRENT DE-RATING CURVE . . . . . 195
6.6.1 Display Resolution . . . . . 195
6.6.2 Memory Speed . . . . . 195
6.6.3 I/O Current De-rating Curve . . . . . 195
6.7 AC CHARACTERISTICS . . . . . 196

**7.0 Package Specifications . . . . . 207**

7.1 THERMAL CHARACTERISTICS . . . . . 207
7.1.1 Heatsink Considerations . . . . . 208
7.2 MECHANICAL PACKAGE OUTLINES . . . . . 210

## Table of Contents (Continued)


**8.0 Instruction Set . . . . . 213**

8.1 GENERAL INSTRUCTION SET FORMAT . . . . . 213
8.1.1 Prefix (Optional) . . . . . 214
8.1.2 Opcode . . . . . 214
8.1.2.1 w Field (Operand Size) . . . . . 214
8.1.2.2 d Field (Operand Direction) . . . . . 214
8.1.2.3 s Field (Immediate Data Field Size) . . . . . 215
8.1.2.4 eee Field (MOV-Instruction Register Selection) . . . . . 215
8.1.3 mod and r/m Byte (Memory Addressing) . . . . . 215
8.1.4 reg Field . . . . . 216
8.1.4.1 sreg2 Field (ES, CS, SS, DS Register Selection) . . . . . 216
8.1.4.2 sreg3 Field (FS and GS Segment Register Selection) . . . . . 216
8.1.5 s-i-b Byte (Scale, Indexing, Base) . . . . . 216
8.1.5.1 ss Field (Scale Selection) . . . . . 216
8.1.5.2 index Field (Index Selection) . . . . . 217
8.1.5.3 Base Field (s-i-b Present) . . . . . 217
8.2 CPUID INSTRUCTION . . . . . 218
8.2.1 Standard CPUID Levels . . . . . 218
8.2.1.1 CPUID Instruction with EAX = 0000 0000h . . . . . 218
8.2.1.2 CPUID Instruction with EAX = 00000001h . . . . . 219
8.2.1.3 CPUID Instruction with EAX = 00000002h . . . . . 219
8.2.2 Extended CPUID Levels . . . . . 220
8.2.2.1 CPUID Instruction with EAX = 80000000h . . . . . 220
8.2.2.2 CPUID Instruction with EAX = 80000001h . . . . . 220
8.2.2.3 CPUID Instruction with EAX = 80000002h, 80000003h, 80000004h . . . . . 221
8.2.2.4 CPUID Instruction with EAX = 80000005h . . . . . 221
8.3 PROCESSOR CORE INSTRUCTION SET . . . . . 222
8.3.1 Opcodes . . . . . 222
8.3.2 Clock Counts . . . . . 222
8.3.3 Flags . . . . . 222
8.4 FPU INSTRUCTION SET . . . . . 234
8.5 MMX INSTRUCTION SET . . . . . 239
8.6 EXTENDED MMX INSTRUCTION SET . . . . . 244

**Appendix A Support Documentation . . . . . 246**

A.1 Order Information . . . . . 246
A.2 Data Book Revision History . . . . . 246

## 1.0 Architecture Overview

The Geode GXLV processor series represents the sixth generation of x86-compatible 32-bit processors with sixth-generation features. The decoupled load/store unit allows reordering of load/store traffic to achieve higher performance. Other features include single-cycle execution, single-cycle instruction decode, 16 KB write-back cache, and clock rates up to 266 MHz. These features are made possible by the use of advanced-process technologies and pipelining.

The GXLV processor has low power consumption at all clock frequencies. Where additional power savings are required, designers can make use of Suspend Mode, Stop Clock capability, and System Management Mode (SMM).

The GXLV processor is divided into major functional blocks (as shown in Figure 1-1):

- Integer Unit
- Floating Point Unit (FPU)
- Write-Back Cache Unit
- Memory Management Unit (MMU)
- Internal Bus Interface Unit
- Integrated Functions

Instructions are executed in the integer unit and in the floating point unit. The cache unit stores the most recently used data and instructions and provides fast access to this information for the integer and floating point units.

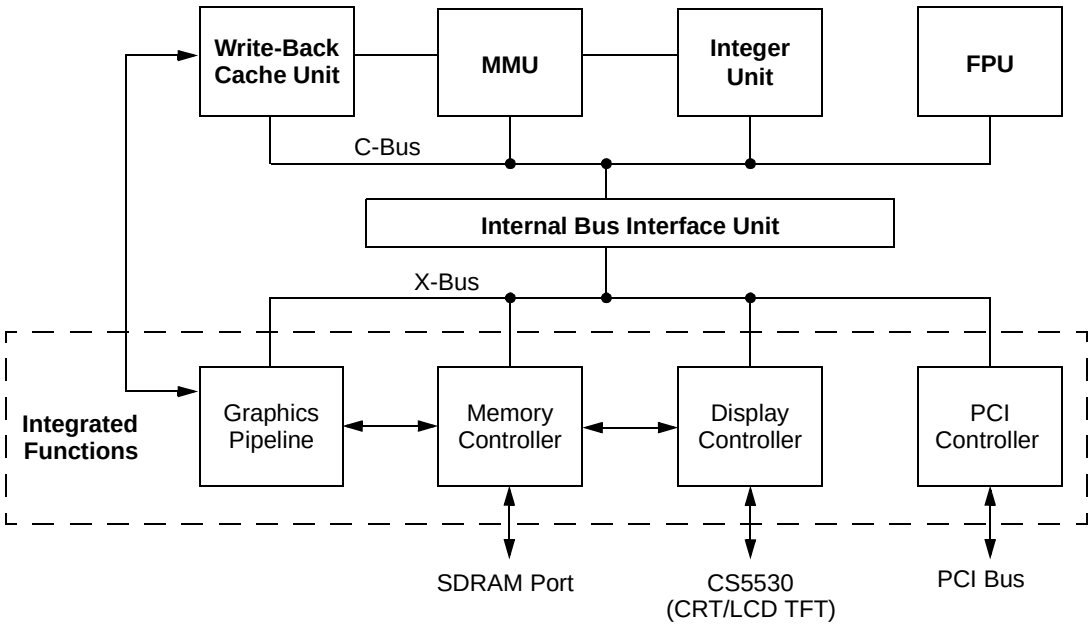



**Figure 1-1. Internal Block Diagram**

## Architecture Overview (Continued)

### 1.1 INTEGER UNIT

The integer unit consists of:

- Instruction Buffer
- Instruction Fetch
- Instruction Decoder and Execution

The pipelined integer unit fetches, decodes, and executes x86 instructions through the use of a five-stage integer pipeline.

The instruction fetch pipeline stage generates, from the on-chip cache, a continuous high-speed instruction stream for use by the processor. Up to 128 bits of code are read during a single clock cycle.

Branch prediction logic within the prefetch unit generates a predicted target address for unconditional or conditional branch instructions. When a branch instruction is detected, the instruction fetch stage starts loading instructions at the predicted address within a single clock cycle. Up to 48 bytes of code are queued prior to the instruction decode stage.

The instruction decode stage evaluates the code stream provided by the instruction fetch stage and determines the number of bytes in each instruction and the instruction type. Instructions are processed and decoded at a maximum rate of one instruction per clock.

The address calculation function is pipelined and contains two stages, AC1 and AC2. If the instruction refers to a memory operand, AC1 calculates a linear memory address for the instruction.

The AC2 stage performs any required memory management functions, cache accesses, and register file accesses. If a floating point instruction is detected by AC2, the instruction is sent to the floating point unit for processing.

The execution stage, under control of microcode, executes instructions using the operands provided by the address calculation stage.

Write-back, the last stage of the integer unit, updates the register file within the integer unit or writes to the load/store unit within the memory management unit.

### 1.2 FLOATING POINT UNIT

The floating point unit (FPU) interfaces to the integer unit and the cache unit through a 64-bit bus. The FPU is x87-instruction-set compatible and adheres to the IEEE-754 standard. Because almost all applications that contain FPU instructions also contain integer instructions, the GXLV processor's FPU achieves high performance by completing integer and FPU operations in parallel.

FPU instructions are dispatched to the pipeline within the integer unit. The address calculation stage of the pipeline checks for memory management exceptions and accesses memory operands for use by the FPU. Once the instructions and operands have been provided to the FPU, the FPU completes instruction execution independently of the integer unit.

### 1.3 WRITE-BACK CACHE UNIT

The 16 KB write-back unified (data/instruction) cache is configured as four-way set associative. The cache stores up to 16 KB of code and data in 1024 cache lines.

The GXLV processor provides the ability to allocate a portion of the L1 cache as a scratchpad, which is used to accelerate the Virtual Systems Architecture technology algorithms as well as for some graphics operations.

### 1.4 MEMORY MANAGEMENT UNIT

The memory management unit (MMU) translates the linear address supplied by the integer unit into a physical address to be used by the cache unit and the internal bus interface unit. Memory management procedures are x86-compatible, adhering to standard paging mechanisms.

The MMU also contains a load/store unit that is responsible for scheduling cache and external memory accesses. The load/store unit incorporates two performance-enhancing features:

- **Load-store reordering** that gives memory reads required by the integer unit a priority over writes to external memory.
- **Memory-read bypassing** that eliminates unnecessary memory reads by using valid data from the execution unit.

### 1.5 INTERNAL BUS INTERFACE UNIT

The internal bus interface unit provides a bridge from the GXLV processor to the integrated system functions (i.e., memory subsystem, display controller, graphics pipeline) and the PCI bus interface.

When external memory access is required, the physical address is calculated by the memory management unit and then passed to the internal bus interface unit, which translates the cycle to an X-Bus cycle (the X-Bus is a proprietary internal bus which provides a common interface for all of the integrated functions). The X-Bus memory cycle is arbitrated between other pending X-Bus memory requests to the SDRAM controller before completing.

In addition, the internal bus interface unit provides configuration control for up to 20 different regions within system memory with separate controls for read access, write access, cacheability, and PCI access.

## Architecture Overview (Continued)

### 1.6 INTEGRATED FUNCTIONS

The GXLV processor integrates the following functions traditionally implemented using external devices:

- High-performance 2D graphics accelerator
- Separate CRT and TFT control from the display controller
- SDRAM memory controller
- PCI bridge

The processor has also been enhanced to support VSA technology implementation.

The GXLV processor implements a Unified Memory Architecture (UMA). By using DCT (Display Compression Technology) architecture, the performance degradation inherent in traditional UMA systems is eliminated.

#### 1.6.1 Graphics Accelerator

The graphics accelerator is a full-featured GUI accelerator. The graphics pipeline implements a bitBLT engine for frame buffer bitBLTs and rectangular fills. Additional instructions in the integer unit may be processed, as the bitBLT engine assists the CPU in the bitBLT operations that take place between system memory and the frame buffer. This combination of hardware and software is used by the display driver to provide very fast bidirectional transfers between system memory and the frame buffer. The bitBLT engine also draws randomly oriented vectors, and scanlines for polygon fill. All of the pipeline operations described in the following list can be applied to any bitBLT operation.

- **Pattern Memory:** Render with 8x8 dither, 8x8 monochrome, or 8x1 color pattern.
- **Color Expansion:** Expand monochrome bitmaps to full depth 8- or 16-bit colors.
- **Transparency:** Suppresses drawing of background pixels for transparent text.
- **Raster Operations:** Boolean operation combines source, destination, and pattern bitmaps.

#### 1.6.2 Display Controller

The display port is a direct interface to the Geode I/O companion (i.e., CS5530, part number 25420-03) which drives a TFT flat panel display, LCD panel, or a CRT display.

The display controller (video generator) retrieves image data from the frame buffer, performs a color-look-up if required, inserts the cursor overlay into the pixel stream, generates display timing, and formats the pixel data for output to a variety of display devices. The display controller contains DCT architecture that allows the GXLV processor to refresh the display from a compressed copy of the frame buffer. DCT architecture typically decreases the screen refresh bandwidth requirement by a factor of 15 to 20, minimizing bandwidth contention.

#### 1.6.3 XpressRAM Memory Subsystem

The memory controller drives a 64-bit SDRAM port directly. The SDRAM memory array contains both the main system memory and the graphics frame buffer. Up to four module banks of SDRAM are supported. Each module bank can have two or four component banks depending on the memory size and organization. The maximum configuration is four module banks with four component banks, each providing a total of 16 open banks. The maximum memory size is 256 MB.

The memory controller handles multiple requests for memory data from the GXLV processor, the graphics accelerator and the display controller. The memory controller contains extensive buffering logic that helps minimize contention for memory bandwidth between graphics and CPU requests. The memory controller cooperates with the internal bus controller to determine the cacheability of all memory references.

#### 1.6.4 PCI Controller

The GXLV processor incorporates a full-function PCI interface module that includes the PCI arbiter. All accesses to external I/O devices are sent over the PCI bus, although most memory accesses are serviced by the SDRAM controller. The internal bus interface unit contains address mapping logic that determines if memory accesses are targeted for the SDRAM or for the PCI bus.

## Architecture Overview (Continued)

### 1.7 GEODE GXLV/CS5530 SYSTEM DESIGNS

A GXLV processor and Geode CS5530 I/O companion based design provides high performance using 32-bit x86 processing. The two chips integrate video, audio and memory interface functions normally performed by external hardware. The CS5530 enables the full features of the GXLV processor with MMX support. These features include full VGA and VESA video, 16-bit stereo sound, IDE interface, ISA interface, SMM power management, and IBM's AT compatibility logic. In addition, the CS5530 provides an Ultra DMA/33 interface, MPEG1 assist, and AC97 Version 2.0 compliant audio.

Figure 1-2 shows a basic block system diagram which also includes the Geode CS9210 graphics companion for designs that need to interface to a Dual Scan Super Twisted Pneumatic (DSTN) panel (instead of a TFT panel).

Figure 1-3 shows an example of a CS9210 interface in a typical GXLV/CS5530 based system design. The CS9210 converts the digital RGB output of the CS5530 to the digital output suitable for driving a color DSTN flat panel LCD. It can drive all standard color DSTN flat panels up to a 1024x768 resolution.

Figures 1-4 and 1-5 show the signal connections between the GXLV processor and the CS5530. For connections to the CS9210, refer to the CS9210 data book.

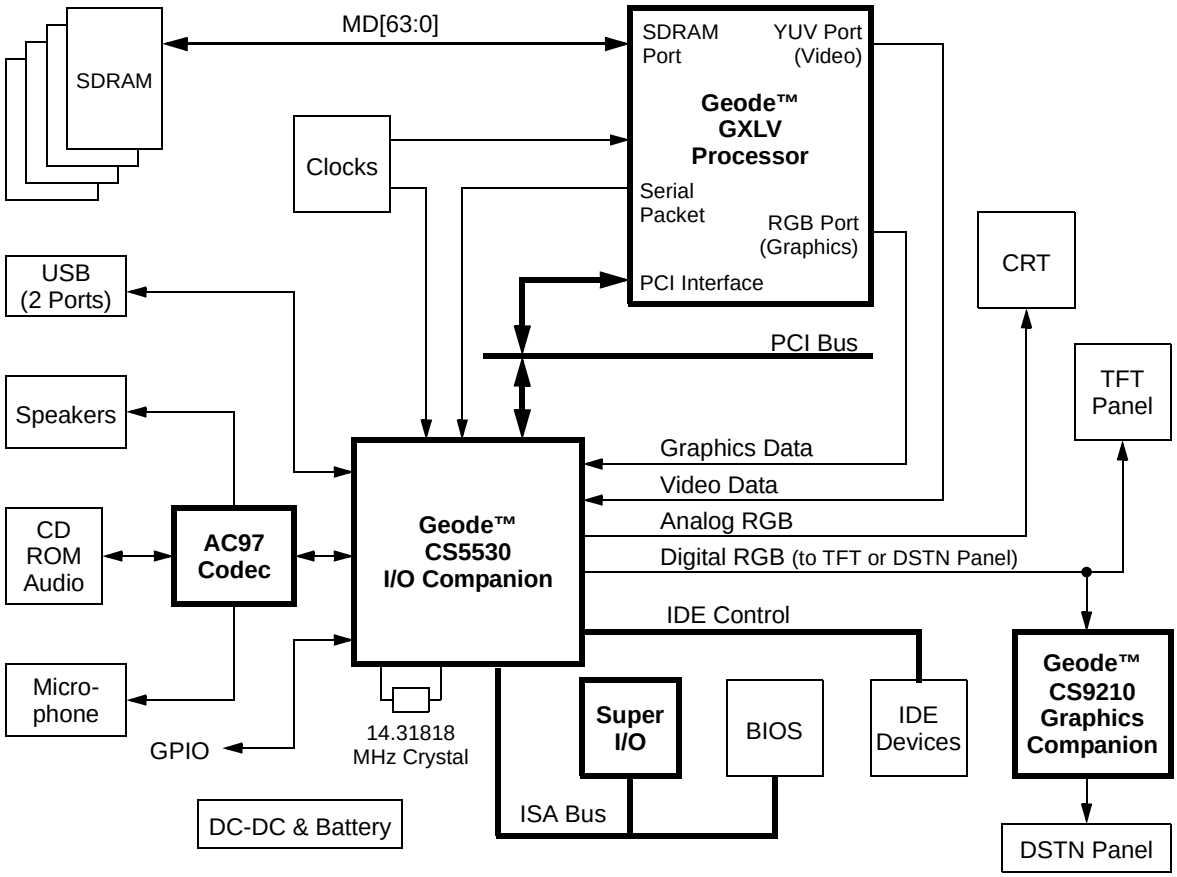



**Figure 1-2. Geode™ GXLV/CS5530 System Block Diagram**

## Architecture Overview (Continued)

.

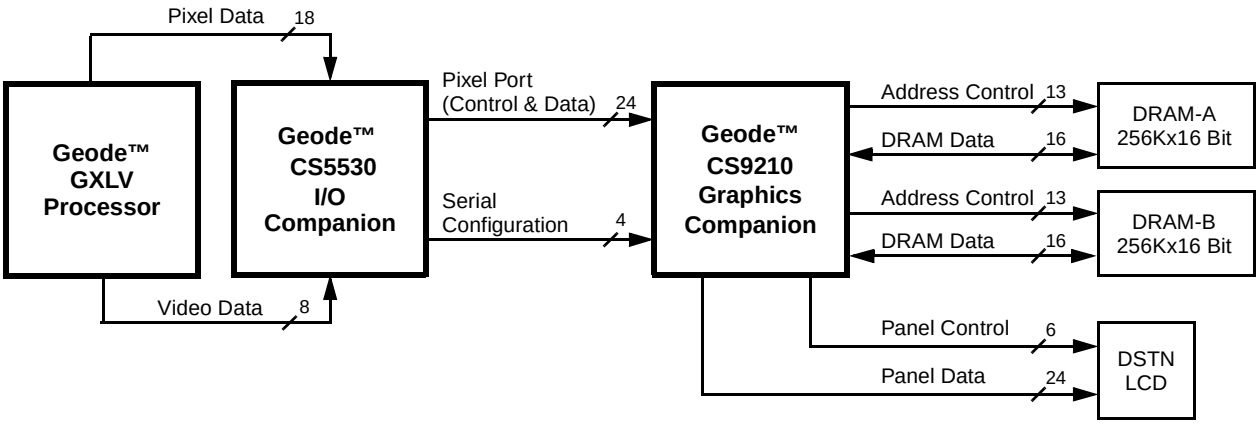
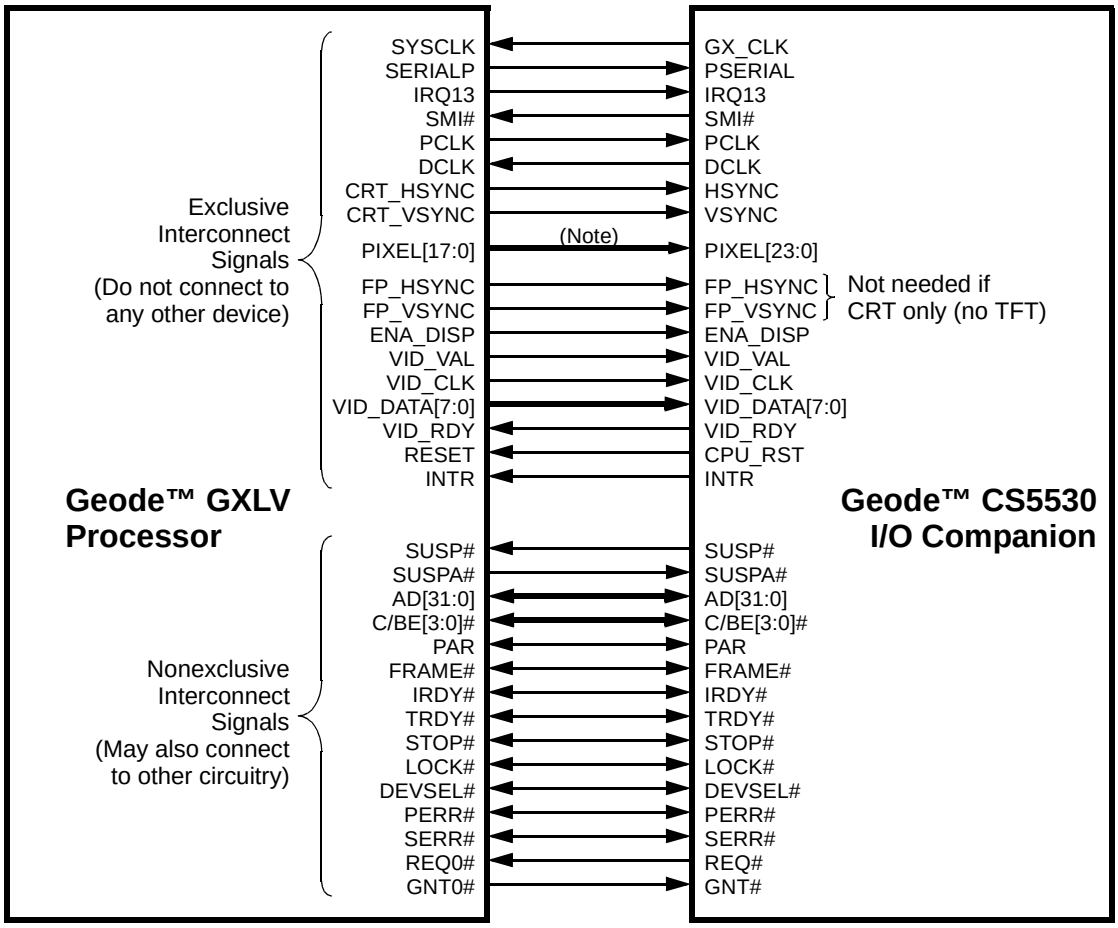



**Figure 1-3. Geode™ CS9210 Interface System Diagram**




**Note:** Refer to Figure 1-5 for interconnection of the pixel lines.

**Figure 1-4. Geode™ GXLV/CS5530 Signal Connections**

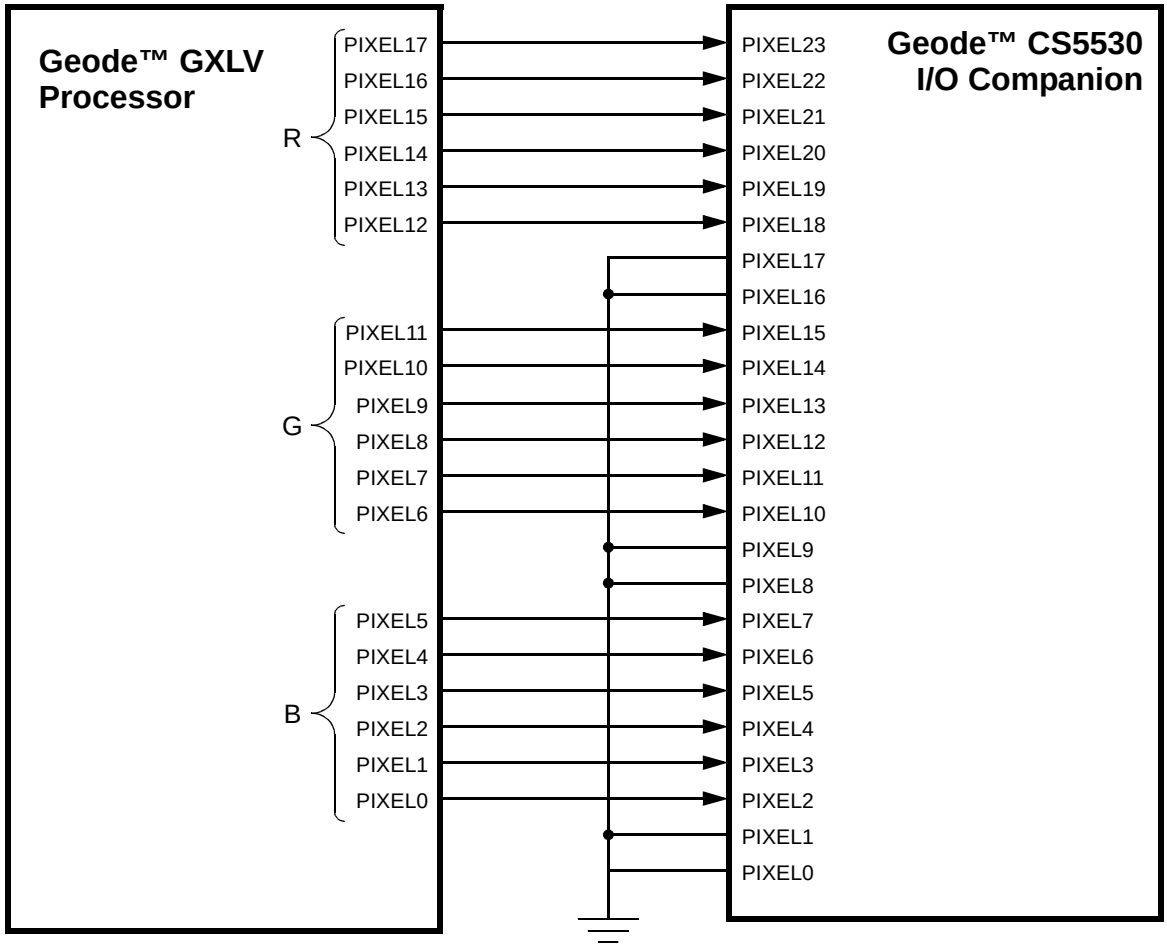



**Figure 1-5. PIXEL Signal Connections**

## Architecture Overview (Continued)

### 1.7.1 Reference Designs

As described previously, the GXLV series of integrated processors is designed specifically to work with National's Geode I/O and graphics companion devices. To help define and drive the emerging information appliance market, several reference systems have been developed by National Semiconductor. These GXLV processor based reference systems provide optimized and targeted solutions for three main segments of the information appliance market: Personal Internet Access, Thin Client, and Set-top Box. Contact your local National Semiconductor sales or field support representative for further information on reference designs for the information appliance market.

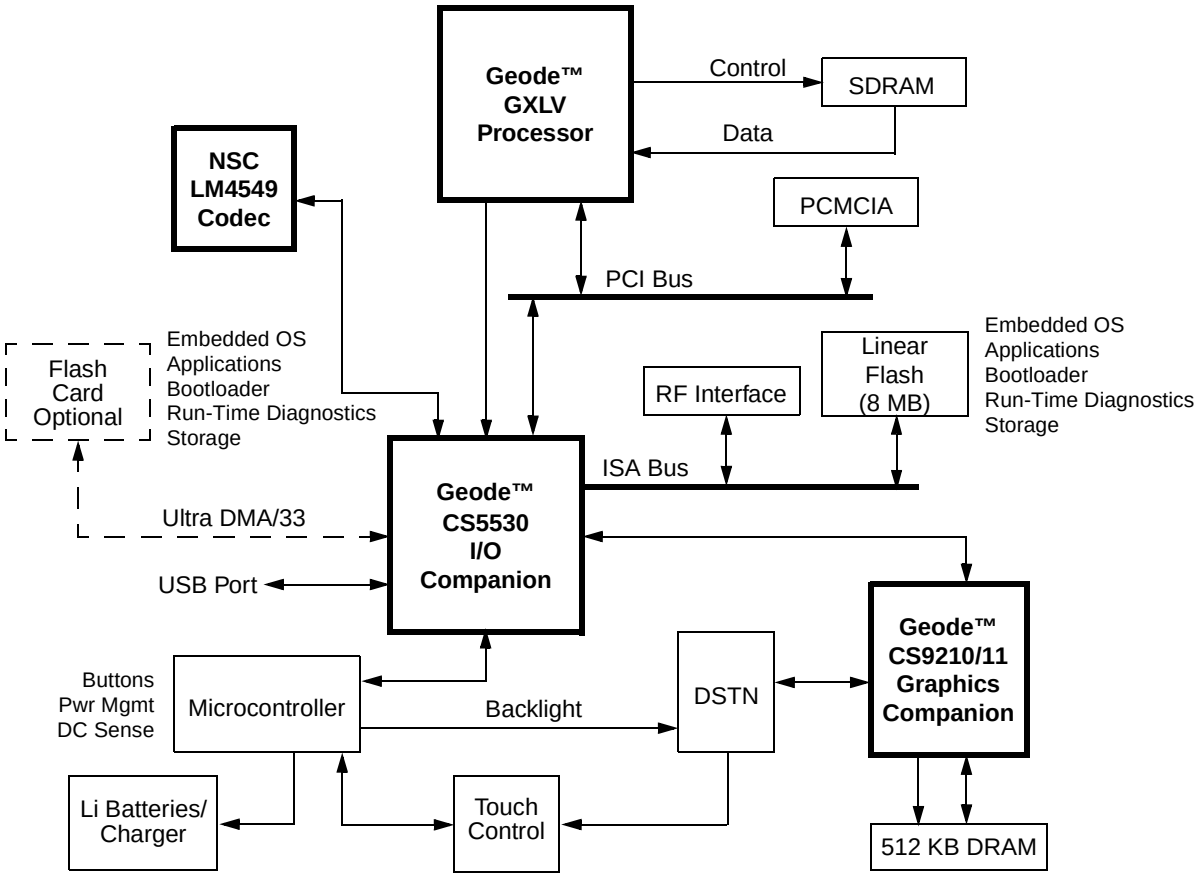



**Figure 1-6. Example WebPAD™ System Diagram**

## Architecture Overview (Continued)

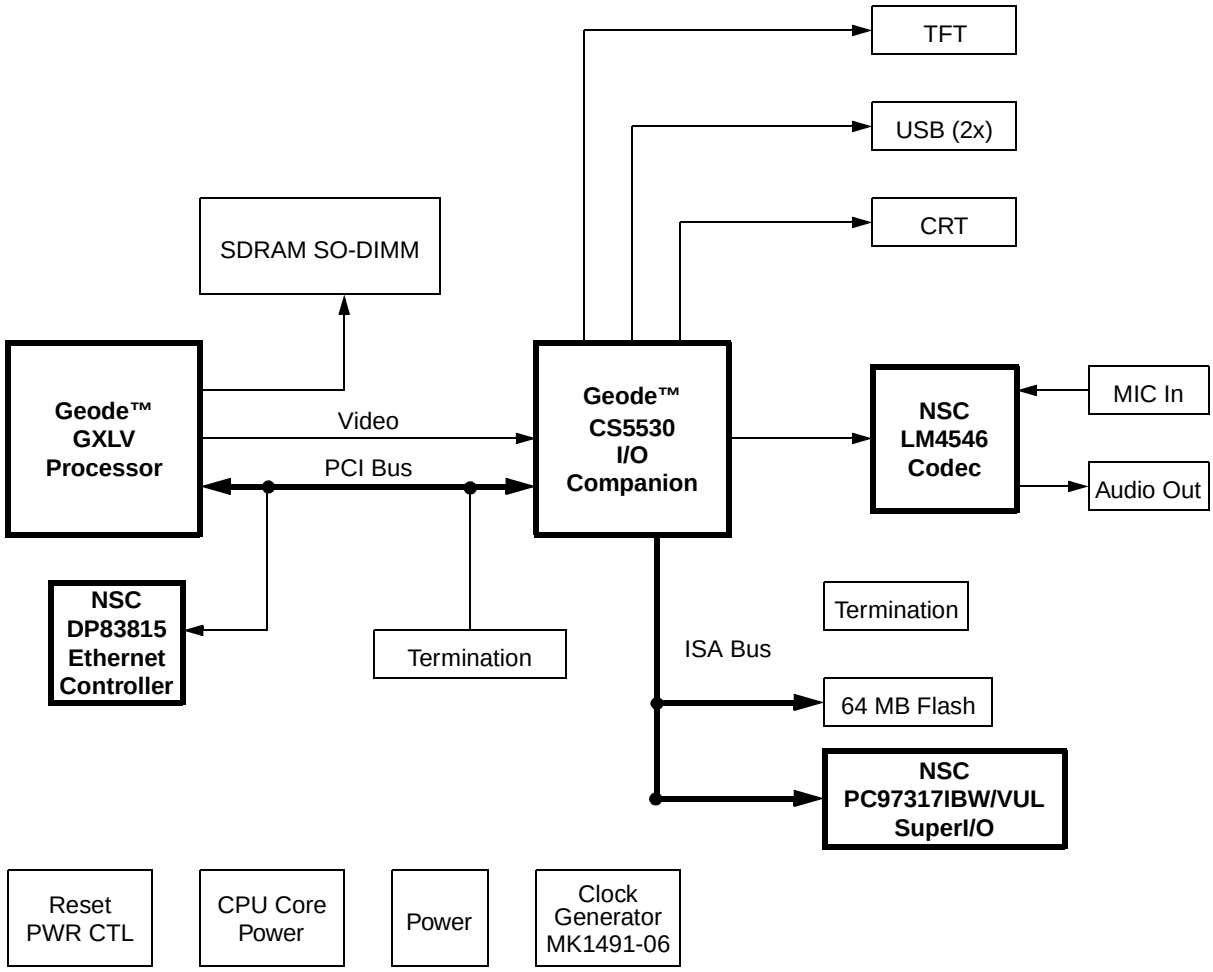



**Figure 1-7. Example Thin Client System Diagram**

## Architecture Overview (Continued)

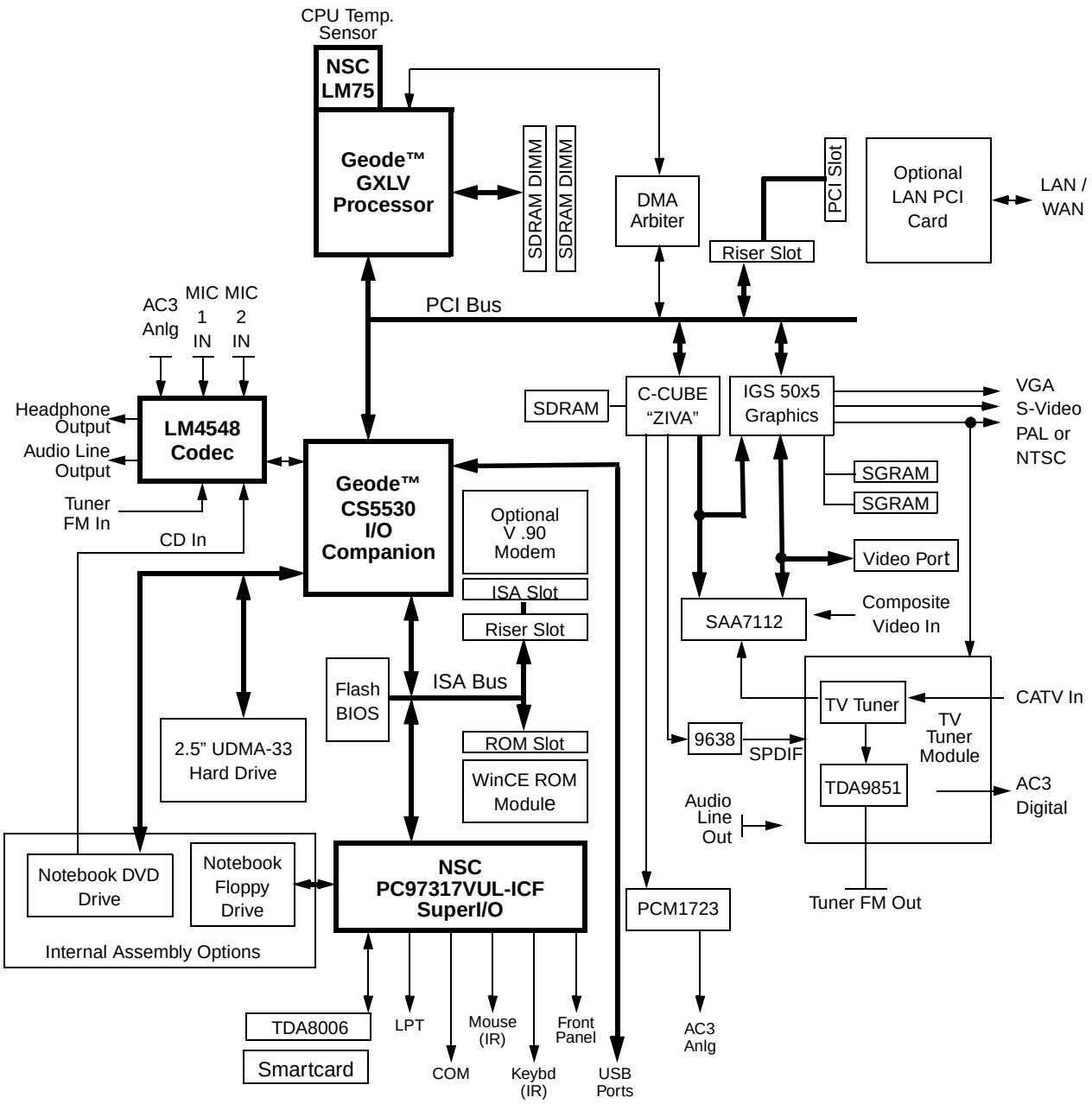



**Figure 1-8. Example Set-Top Box System Diagram**

## 2.0 Signal Definitions

This section describes the external interface of the Geode GXLV processor. Figure 2-1 shows the signals organized by their functional interface groups (internal test and electrical pins are not shown).

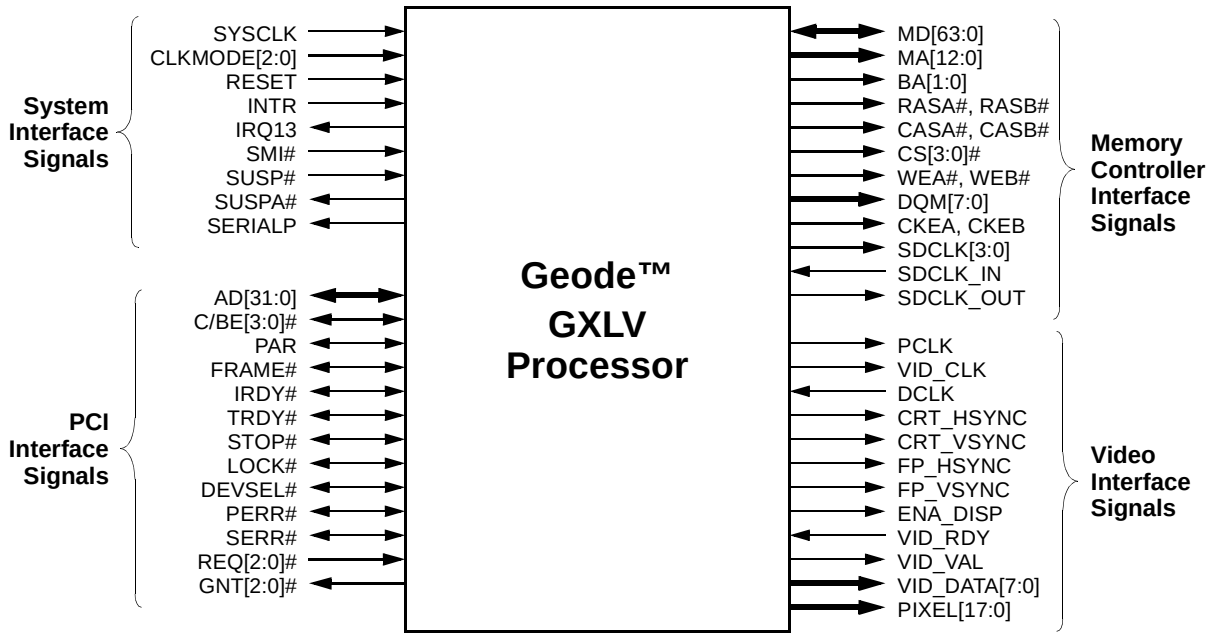



**Figure 2-1. Functional Block Diagram**

## Signal Definitions (Continued)

### 2.1 PIN ASSIGNMENTS

The tables in this section use several common abbreviations. Table 2-1 lists the mnemonics and their meanings.

Figure 2-2 shows the pin assignment for the 352 BGA with Table 2-2 and Table 2-3 listing the pin assignments sorted by pin number and alphabetically by signal name, respectively.

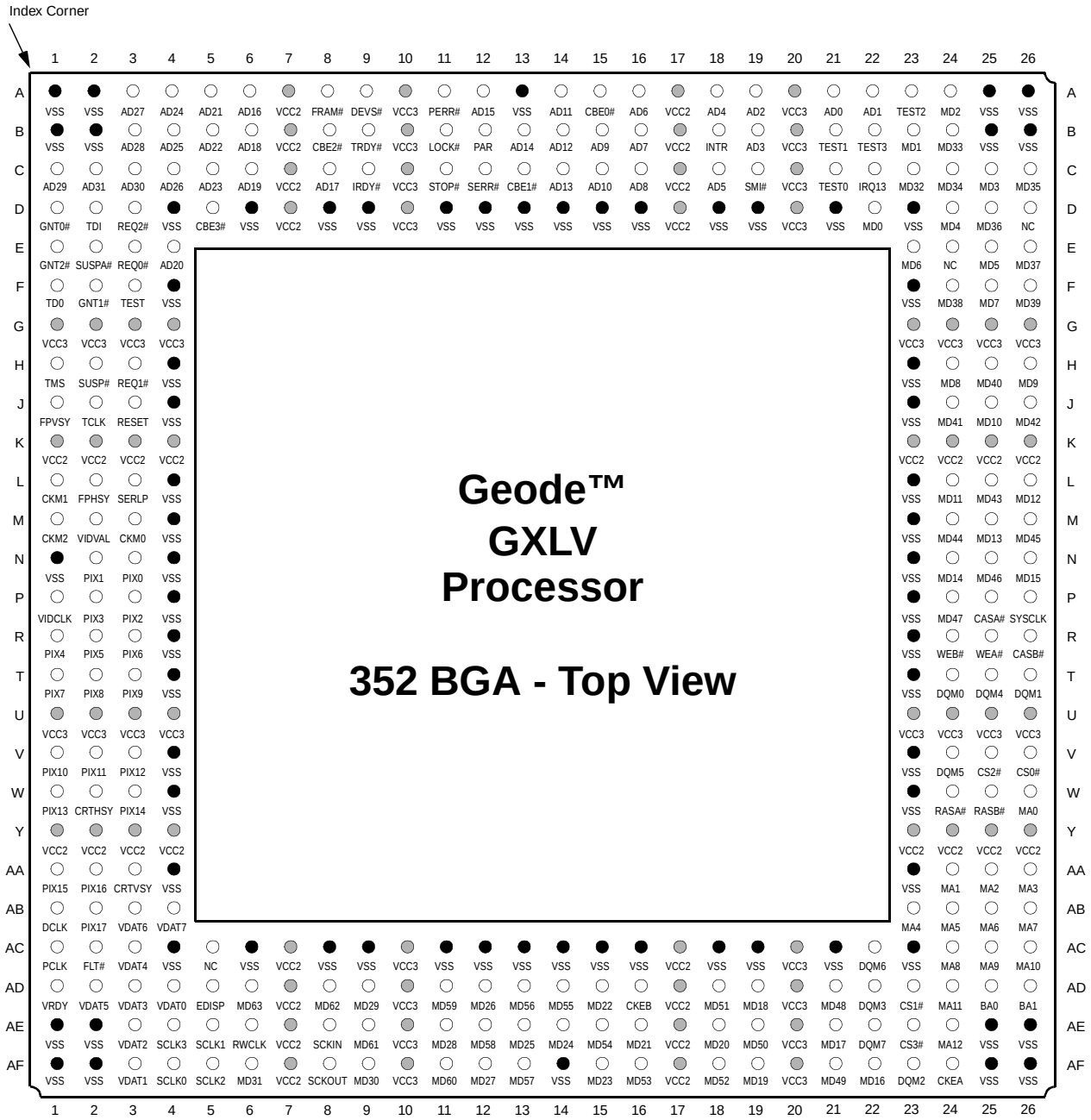
Figure 2-3 shows the pin assignment for the 320 SPGA with Table 2-4 and Table 2-5 listing the pin assignments sorted by pin number and alphabetically by signal name, respectively.

In Section 2.2 “Signal Descriptions” on page 31 a description of each signal is provided within its associated functional group.

**Table 2-1. Pin Type Definitions**

| Mnemonic | Definition |
|---|---|
| I | Standard input pin. |
| I/O | Bidirectional pin. |
| O | Totem-pole output. |
| OD | Open-drain output structure that allows multiple devices to share the pin in a wired-OR configuration. |
| PU | Pull-up resistor. |
| PD | Pull-down resistor. |
| s/t/s | Sustained tri-state an active-low tri-state signal owned and driven by one and only one agent at a time. The agent that drives an s/t/s pin low must drive it high for at least one clock before letting it float. A new agent cannot start driving an s/t/s signal any sooner than one clock after the previous owner lets it float. A pull-up resistor on the mother-board is required to sustain the inactive state until another agent drives it. |
| VCC (PWR) | Power pin. |
| VSS (GND) | Ground pin. |
| # | The "#" symbol at the end of a signal name indicates that the active, or asserted state occurs when the signal is at a low voltage level. When "#" is not present after the signal name, the signal is asserted when at a high voltage level. |
| t/s | Tri-state signal. |

## Signal Definitions (Continued)




**Note:** Signal names have been abbreviated in this figure due to space constraints.

● = GND terminal

◉ = PWR terminal (VCC2 = VCC_CORE; VCC3 = VCC_IO)

**Figure 2-2. 352 BGA Pin Assignment Diagram**

For order information, refer to Section A.1 “Order Information” on page 246.

## Signal Definitions (Continued)

### Table 2-2. 352 BGA Pin Assignments - Sorted by Pin Number

| Pin No. | Signal Name | Pin No. | Signal Name | Pin No. | Signal Name | Pin No. | Signal Name | Pin No. | Signal Name |
|---|---|---|---|---|---|---|---|---|---|
| A1 | VSS | B23 | MD1 | D19 | VSS | K1 | VCC2 | T1 | PIXEL7 |
| A2 | VSS | B24 | MD33 | D20 | VCC3 | K2 | VCC2 | T2 | PIXEL8 |
| A3 | AD27 | B25 | VSS | D21 | VSS | K3 | VCC2 | T3 | PIXEL9 |
| A4 | AD24 | B26 | VSS | D22 | MD0 | K4 | VCC2 | T4 | VSS |
| A5 | AD21 | C1 | AD29 | D23 | VSS | K23 | VCC2 | T23 | VSS |
| A6 | AD16 | C2 | AD31 | D24 | MD4 | K24 | VCC2 | T24 | DQM0 |
| A7 | VCC2 | C3 | AD30 | D25 | MD36 | K25 | VCC2 | T25 | DQM4 |
| A8 | FRAME# | C4 | AD26 | D26 | NC | K26 | VCC2 | T26 | DQM1 |
| A9 | DEVSEL# | C5 | AD23 | E1 | GNT2# | L1 | CLKMODE1 | U1 | VCC3 |
| A10 | VCC3 | C6 | AD19 | E2 | SUSPA# | L2 | FP_HSYNC | U2 | VCC3 |
| A11 | PERR# | C7 | VCC2 | E3 | REQ0# | L3 | SERIALP | U3 | VCC3 |
| A12 | AD15 | C8 | AD17 | E4 | AD20 | L4 | VSS | U4 | VCC3 |
| A13 | VSS | C9 | IRDY# | E23 | MD6 | L23 | VSS | U23 | VCC3 |
| A14 | AD11 | C10 | VCC3 | E24 | NC | L24 | MD11 | U24 | VCC3 |
| A15 | C/BE0# | C11 | STOP# | E25 | MD5 | L25 | MD43 | U25 | VCC3 |
| A16 | AD6 | C12 | SERR# | E26 | MD37 | L26 | MD12 | U26 | VCC3 |
| A17 | VCC2 | C13 | C/BE1# | F1 | TDO | M1 | CLKMODE2 | V1 | PIXEL10 |
| A18 | AD4 | C14 | AD13 | F2 | GNT1# | M2 | VID_VAL | V2 | PIXEL11 |
| A19 | AD2 | C15 | AD10 | F3 | TEST | M3 | CLKMODE0 | V3 | PIXEL12 |
| A20 | VCC3 | C16 | AD8 | F4 | VSS | M4 | VSS | V4 | VSS |
| A21 | AD0 | C17 | VCC2 | F23 | VSS | M23 | VSS | V23 | VSS |
| A22 | AD1 | C18 | AD5 | F24 | MD38 | M24 | MD44 | V24 | DQM5 |
| A23 | TEST2 | C19 | SMI# | F25 | MD7 | M25 | MD13 | V25 | CS2# |
| A24 | MD2 | C20 | VCC3 | F26 | MD39 | M26 | MD45 | V26 | CS0# |
| A25 | VSS | C21 | TEST0 | G1 | VCC3 | N1 | VSS | W1 | PIXEL13 |
| A26 | VSS | C22 | IRQ13 | G2 | VCC3 | N2 | PIXEL1 | W2 | CRT_HSYNC |
| B1 | VSS | C23 | MD32 | G3 | VCC3 | N3 | PIXEL0 | W3 | PIXEL14 |
| B2 | VSS | C24 | MD34 | G4 | VCC3 | N4 | VSS | W4 | VSS |
| B3 | AD28 | C25 | MD3 | G23 | VCC3 | N23 | VSS | W23 | VSS |
| B4 | AD25 | C26 | MD35 | G24 | VCC3 | N24 | MD14 | W24 | RASA# |
| B5 | AD22 | D1 | GNT0# | G25 | VCC3 | N25 | MD46 | W25 | RASB# |
| B6 | AD18 | D2 | TDI | G26 | VCC3 | N26 | MD15 | W26 | MA0 |
| B7 | VCC2 | D3 | REQ2# | H1 | TMS | P1 | VID_CLK | Y1 | VCC2 |
| B8 | C/BE2# | D4 | VSS | H2 | SUSP# | P2 | PIXEL3 | Y2 | VCC2 |
| B9 | TRDY# | D5 | C/BE3# | H3 | REQ1# | P3 | PIXEL2 | Y3 | VCC2 |
| B10 | VCC3 | D6 | VSS | H4 | VSS | P4 | VSS | Y4 | VCC2 |
| B11 | LOCK# | D7 | VCC2 | H23 | VSS | P23 | VSS | Y23 | VCC2 |
| B12 | PAR | D8 | VSS | H24 | MD8 | P24 | MD47 | Y24 | VCC2 |
| B13 | AD14 | D9 | VSS | H25 | MD40 | P25 | CASA# | Y25 | VCC2 |
| B14 | AD12 | D10 | VCC3 | H26 | MD9 | P26 | SYSCLK | Y26 | VCC2 |
| B15 | AD9 | D11 | VSS | J1 | FP_VSYNC | R1 | PIXEL4 | AA1 | PIXEL15 |
| B16 | AD7 | D12 | VSS | J2 | TCLK | R2 | PIXEL5 | AA2 | PIXEL16 |
| B17 | VCC2 | D13 | VSS | J3 | RESET | R3 | PIXEL6 | AA3 | CRT_VSYNC |
| B18 | INTR | D14 | VSS | J4 | VSS | R4 | VSS | AA4 | VSS |
| B19 | AD3 | D15 | VSS | J23 | VSS | R23 | VSS | AA23 | VSS |
| B20 | VCC3 | D16 | VSS | J24 | MD41 | R24 | WEB# | AA24 | MA1 |
| B21 | TEST1 | D17 | VCC2 | J25 | MD10 | R25 | WEA# | AA25 | MA2 |
| B22 | TEST3 | D18 | VSS | J26 | MD42 | R26 | CASB# | AA26 | MA3 |

## Signal Definitions (Continued)

**Table 2-2. 352 BGA Pin Assignments - Sorted by Pin Number (Continued)**

| Pin No. | Signal Name | Pin No. | Signal Name | Pin No. | Signal Name | Pin No. | Signal Name | Pin No. | Signal Name |
|---|---|---|---|---|---|---|---|---|---|
| AB1 | DCLK | AC16 | VSS | AD13 | MD56 | AE10 | VCC3 | AF7 | VCC2 |
| AB2 | PIXEL17 | AC17 | VCC2 | AD14 | MD55 | AE11 | MD28 | AF8 | SDCLK_OUT |
| AB3 | VID_DATA6 | AC18 | VSS | AD15 | MD22 | AE12 | MD58 | AF9 | MD30 |
| AB4 | VID_DATA7 | AC19 | VSS | AD16 | CKEB | AE13 | MD25 | AF10 | VCC3 |
| AB23 | MA4 | AC20 | VCC3 | AD17 | VCC2 | AE14 | MD24 | AF11 | MD60 |
| AB24 | MA5 | AC21 | VSS | AD18 | MD51 | AE15 | MD54 | AF12 | MD27 |
| AB25 | MA6 | AC22 | DQM6 | AD19 | MD18 | AE16 | MD21 | AF13 | MD57 |
| AB26 | MA7 | AC23 | VSS | AD20 | VCC3 | AE17 | VCC2 | AF14 | VSS |
| AC1 | PCLK | AC24 | MA8 | AD21 | MD48 | AE18 | MD20 | AF15 | MD23 |
| AC2 | FLT# | AC25 | MA9 | AD22 | DQM3 | AE19 | MD50 | AF16 | MD53 |
| AC3 | VID_DATA4 | AC26 | MA10 | AD23 | CS1# | AE20 | VCC3 | AF17 | VCC2 |
| AC4 | VSS | AD1 | VID_RDY | AD24 | MA11 | AE21 | MD17 | AF18 | MD52 |
| AC5 | NC | AD2 | VID_DATA5 | AD25 | BA0 | AE22 | DQM7 | AF19 | MD19 |
| AC6 | VSS | AD3 | VID_DATA3 | AD26 | BA1 | AE23 | CS3# | AF20 | VCC3 |
| AC7 | VCC2 | AD4 | VID_DATA0 | AE1 | VSS | AE24 | MA12 | AF21 | MD49 |
| AC8 | VSS | AD5 | ENA_DISP | AE2 | VSS | AE25 | VSS | AF22 | MD16 |
| AC9 | VSS | AD6 | MD63 | AE3 | VID_DATA2 | AE26 | VSS | AF23 | DQM2 |
| AC10 | VCC3 | AD7 | VCC2 | AE4 | SDCLK3 | AF1 | VSS | AF24 | CKEA |
| AC11 | VSS | AD8 | MD62 | AE5 | SDCLK1 | AF2 | VSS | AF25 | VSS |
| AC12 | VSS | AD9 | MD29 | AE6 | RW_CLK | AF3 | VID_DATA1 | AF26 | VSS |
| AC13 | VSS | AD10 | VCC3 | AE7 | VCC2 | AF4 | SDCLK0 | | |
| AC14 | VSS | AD11 | MD59 | AE8 | SDCLK_IN | AF5 | SDCLK2 | | |
| AC15 | VSS | AD12 | MD26 | AE9 | MD61 | AF6 | MD31 | | |

## Signal Definitions (Continued)

### Table 2-3. 352 BGA Pin Assignments - Sorted Alphabetically by Signal Name

| Signal Name | Type | Pin No. |
|---|---|---|
| AD0 | I/O | A21 |
| AD1 | I/O | A22 |
| AD2 | I/O | A19 |
| AD3 | I/O | B19 |
| AD4 | I/O | A18 |
| AD5 | I/O | C18 |
| AD6 | I/O | A16 |
| AD7 | I/O | B16 |
| AD8 | I/O | C16 |
| AD9 | I/O | B15 |
| AD10 | I/O | C15 |
| AD11 | I/O | A14 |
| AD12 | I/O | B14 |
| AD13 | I/O | C14 |
| AD14 | I/O | B13 |
| AD15 | I/O | A12 |
| AD16 | I/O | A6 |
| AD17 | I/O | C8 |
| AD18 | I/O | B6 |
| AD19 | I/O | C6 |
| AD20 | I/O | E4 |
| AD21 | I/O | A5 |
| AD22 | I/O | B5 |
| AD23 | I/O | C5 |
| AD24 | I/O | A4 |
| AD25 | I/O | B4 |
| AD26 | I/O | C4 |
| AD27 | I/O | A3 |
| AD28 | I/O | B3 |
| AD29 | I/O | C1 |
| AD30 | I/O | C3 |
| AD31 | I/O | C2 |
| BA0 | O | AD25 |
| BA1 | O | AD26 |
| CASA# | O | P25 |
| CASB# | O | R26 |
| C/BE0# | I/O | A15 |
| C/BE1# | I/O | C13 |
| C/BE2# | I/O | B8 |
| C/BE3# | I/O | D5 |
| CKEA | O | AF24 |
| CKEB | O | AD16 |
| CLKMODE0 | I | M3 |
| CLKMODE1 | I | L1 |
| CLKMODE2 | I | M1 |
| CRT_HSYNC | O | W2 |
| CRT_VSYNC | O | AA3 |
| CS0# | O | V26 |
| CS1# | O | AD23 |
| CS2# | O | V25 |
| CS3# | O | AE23 |
| DCLK | I | AB1 |
| DEVSEL# | s/t/s | A9 (PU) |

| Signal Name | Type | Pin No. |
|---|---|---|
| DQM0 | O | T24 |
| DQM1 | O | T26 |
| DQM2 | O | AF23 |
| DQM3 | O | AD22 |
| DQM4 | O | T25 |
| DQM5 | O | V24 |
| DQM6 | O | AC22 |
| DQM7 | O | AE22 |
| ENA_DISP | O | AD5 |
| FLT# | I | AC2 |
| FP_HSYNC | O | L2 |
| FP_VSYNC | O | J1 |
| FRAME# | s/t/s | A8 (PU) |
| GNT0# | O | D1 |
| GNT1# | O | F2 |
| GNT2# | O | E1 |
| INTR | I | B18 |
| IRDY# | s/t/s | C9 (PU) |
| IRQ13 | O | C22 |
| LOCK# | s/t/s | B11 (PU) |
| MA0 | O | W26 |
| MA1 | O | AA24 |
| MA2 | O | AA25 |
| MA3 | O | AA26 |
| MA4 | O | AB23 |
| MA5 | O | AB24 |
| MA6 | O | AB25 |
| MA7 | O | AB26 |
| MA8 | O | AC24 |
| MA9 | O | AC25 |
| MA10 | O | AC26 |
| MA11 | O | AD24 |
| MA12 | O | AE24 |
| MD0 | I/O | D22 |
| MD1 | I/O | B23 |
| MD2 | I/O | A24 |
| MD3 | I/O | C25 |
| MD4 | I/O | D24 |
| MD5 | I/O | E25 |
| MD6 | I/O | E23 |
| MD7 | I/O | F25 |
| MD8 | I/O | H24 |
| MD9 | I/O | H26 |
| MD10 | I/O | J25 |
| MD11 | I/O | L24 |
| MD12 | I/O | L26 |
| MD13 | I/O | M25 |
| MD14 | I/O | N24 |
| MD15 | I/O | N26 |
| MD16 | I/O | AF22 |
| MD17 | I/O | AE21 |
| MD18 | I/O | AD19 |
| MD19 | I/O | AF19 |

| Signal Name | Type | Pin No. |
|---|---|---|
| MD20 | I/O | AE18 |
| MD21 | I/O | AE16 |
| MD22 | I/O | AD15 |
| MD23 | I/O | AF15 |
| MD24 | I/O | AE14 |
| MD25 | I/O | AE13 |
| MD26 | I/O | AD12 |
| MD27 | I/O | AF12 |
| MD28 | I/O | AE11 |
| MD29 | I/O | AD9 |
| MD30 | I/O | AF9 |
| MD31 | I/O | AF6 |
| MD32 | I/O | C23 |
| MD33 | I/O | B24 |
| MD34 | I/O | C24 |
| MD35 | I/O | C26 |
| MD36 | I/O | D25 |
| MD37 | I/O | E26 |
| MD38 | I/O | F24 |
| MD39 | I/O | F26 |
| MD40 | I/O | H25 |
| MD41 | I/O | J24 |
| MD42 | I/O | J26 |
| MD43 | I/O | L25 |
| MD44 | I/O | M24 |
| MD45 | I/O | M26 |
| MD46 | I/O | N25 |
| MD47 | I/O | P24 |
| MD48 | I/O | AD21 |
| MD49 | I/O | AF21 |
| MD50 | I/O | AE19 |
| MD51 | I/O | AD18 |
| MD52 | I/O | AF18 |
| MD53 | I/O | AF16 |
| MD54 | I/O | AE15 |
| MD55 | I/O | AD14 |
| MD56 | I/O | AD13 |
| MD57 | I/O | AF13 |
| MD58 | I/O | AE12 |
| MD59 | I/O | AD11 |
| MD60 | I/O | AF11 |
| MD61 | I/O | AE9 |
| MD62 | I/O | AD8 |
| MD63 | I/O | AD6 |
| NC | -- | D26 |
| NC | -- | E24 |
| NC | -- | AC5 |
| PAR | I/O | B12 |
| PCLK | O | AC1 |
| PERR# | s/t/s | A11 (PU) |
| PIXEL0 | O | N3 |
| PIXEL1 | O | N2 |
| PIXEL2 | O | P3 |

| Signal Name | Type | Pin No. |
|---|---|---|
| PIXEL3 | O | P2 |
| PIXEL4 | O | R1 |
| PIXEL5 | O | R2 |
| PIXEL6 | O | R3 |
| PIXEL7 | O | T1 |
| PIXEL8 | O | T2 |
| PIXEL9 | O | T3 |
| PIXEL10 | O | V1 |
| PIXEL11 | O | V2 |
| PIXEL12 | O | V3 |
| PIXEL13 | O | W1 |
| PIXEL14 | O | W3 |
| PIXEL15 | O | AA1 |
| PIXEL16 | O | AA2 |
| PIXEL17 | O | AB2 |
| RASA# | O | W24 |
| RASB# | O | W25 |
| REQ0# | I | E3 (PU) |
| REQ1# | I | H3 (PU) |
| REQ2# | I | D3 (PU) |
| RESET | I | J3 |
| RW_CLK | O | AE6 |
| SDCLK_IN | I | AE8 |
| SDCLK_OUT | O | AF8 |
| SDCLK0 | O | AF4 |
| SDCLK1 | O | AE5 |
| SDCLK2 | O | AF5 |
| SDCLK3 | O | AE4 |
| SERIALP | O | L3 |
| SERR# | OD | C12 (PU) |
| SMI# | I | C19 |
| STOP# | s/t/s | C11 (PU) |
| SUSP# | I | H2 (PU) |
| SUSPA# | O | E2 |
| SYSCLK | I | P26 |
| TCLK | I | J2 (PU) |
| TDI | I | D2 (PU) |
| TDO | O | F1 |
| TEST | I | F3 (PD) |
| TEST0 | O | C21 |
| TEST1 | O | B21 |
| TEST2 | O | A23 |
| TEST3 | O | B22 |
| TMS | I | H1 (PU) |
| TRDY# | s/t/s | B9 (PU) |
| VCC2 | PWR | A7 |
| VCC2 | PWR | A17 |
| VCC2 | PWR | B7 |
| VCC2 | PWR | B17 |
| VCC2 | PWR | C7 |
| VCC2 | PWR | C17 |
| VCC2 | PWR | D7 |
| VCC2 | PWR | D17 |

## Signal Definitions (Continued)

**Table 2-3. 352 BGA Pin Assignments - Sorted Alphabetically by Signal Name (Continued)**

| Signal Name | Type | Pin No. |
|---|---|---|
| VCC2 | PWR | K1 |
| VCC2 | PWR | K2 |
| VCC2 | PWR | K3 |
| VCC2 | PWR | K4 |
| VCC2 | PWR | K23 |
| VCC2 | PWR | K24 |
| VCC2 | PWR | K25 |
| VCC2 | PWR | K26 |
| VCC2 | PWR | Y1 |
| VCC2 | PWR | Y2 |
| VCC2 | PWR | Y3 |
| VCC2 | PWR | Y4 |
| VCC2 | PWR | Y23 |
| VCC2 | PWR | Y24 |
| VCC2 | PWR | Y25 |
| VCC2 | PWR | Y26 |
| VCC2 | PWR | AC7 |
| VCC2 | PWR | AC17 |
| VCC2 | PWR | AD7 |
| VCC2 | PWR | AD17 |
| VCC2 | PWR | AE7 |
| VCC2 | PWR | AE17 |
| VCC2 | PWR | AF7 |
| VCC2 | PWR | AF17 |
| VCC3 | PWR | A10 |
| VCC3 | PWR | A20 |
| VCC3 | PWR | B10 |
| VCC3 | PWR | B20 |
| VCC3 | PWR | C10 |
| VCC3 | PWR | C20 |
| VCC3 | PWR | D10 |
| VCC3 | PWR | D20 |
| VCC3 | PWR | G1 |
| VCC3 | PWR | G2 |
| VCC3 | PWR | G3 |
| VCC3 | PWR | G4 |

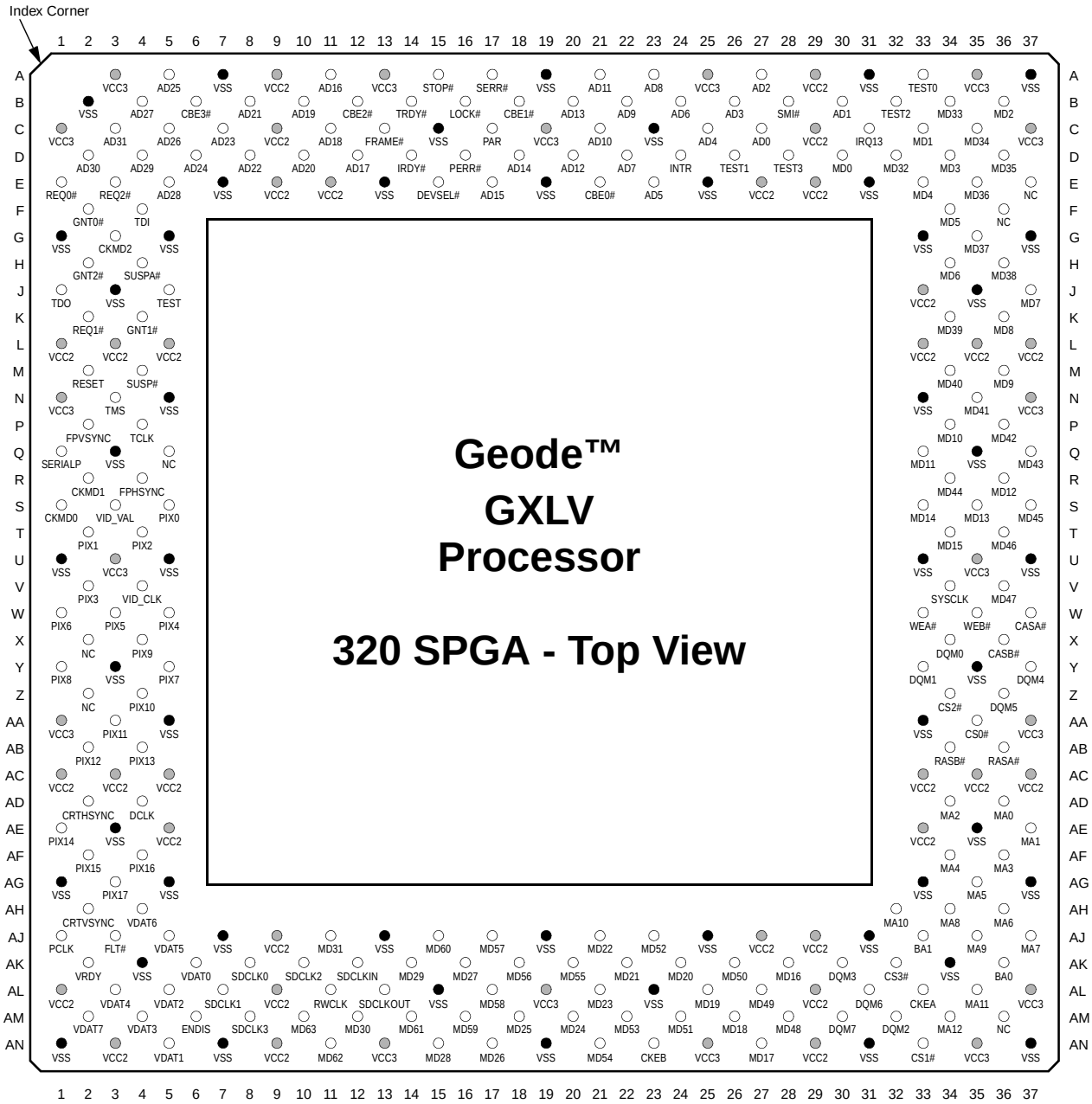
| Signal Name | Type | Pin No. |
|---|---|---|
| VCC3 | PWR | G23 |
| VCC3 | PWR | G24 |
| VCC3 | PWR | G25 |
| VCC3 | PWR | G26 |
| VCC3 | PWR | U1 |
| VCC3 | PWR | U2 |
| VCC3 | PWR | U3 |
| VCC3 | PWR | U4 |
| VCC3 | PWR | U23 |
| VCC3 | PWR | U24 |
| VCC3 | PWR | U25 |
| VCC3 | PWR | U26 |
| VCC3 | PWR | AC10 |
| VCC3 | PWR | AC20 |
| VCC3 | PWR | AD10 |
| VCC3 | PWR | AD20 |
| VCC3 | PWR | AE10 |
| VCC3 | PWR | AE20 |
| VCC3 | PWR | AF10 |
| VCC3 | PWR | AF20 |
| VID_CLK | O | P1 |
| VID_DATA0 | O | AD4 |
| VID_DATA1 | O | AF3 |
| VID_DATA2 | O | AE3 |
| VID_DATA3 | O | AD3 |
| VID_DATA4 | O | AC3 |
| VID_DATA5 | O | AD2 |
| VID_DATA6 | O | AB3 |
| VID_DATA7 | O | AB4 |
| VID_RDY | I | AD1 |
| VID_VAL | O | M2 |
| VSS | GND | A1 |
| VSS | GND | A2 |
| VSS | GND | A13 |
| VSS | GND | A25 |
| VSS | GND | A26 |

| Signal Name | Type | Pin No. |
|---|---|---|
| VSS | GND | B1 |
| VSS | GND | B2 |
| VSS | GND | B25 |
| VSS | GND | B26 |
| VSS | GND | D4 |
| VSS | GND | D6 |
| VSS | GND | D8 |
| VSS | GND | D9 |
| VSS | GND | D11 |
| VSS | GND | D12 |
| VSS | GND | D13 |
| VSS | GND | D14 |
| VSS | GND | D15 |
| VSS | GND | D16 |
| VSS | GND | D18 |
| VSS | GND | D19 |
| VSS | GND | D21 |
| VSS | GND | D23 |
| VSS | GND | F4 |
| VSS | GND | F23 |
| VSS | GND | H4 |
| VSS | GND | H23 |
| VSS | GND | J4 |
| VSS | GND | J23 |
| VSS | GND | L4 |
| VSS | GND | L23 |
| VSS | GND | M4 |
| VSS | GND | M23 |
| VSS | GND | N1 |
| VSS | GND | N4 |
| VSS | GND | N23 |
| VSS | GND | P4 |
| VSS | GND | P23 |
| VSS | GND | R4 |
| VSS | GND | R23 |
| VSS | GND | T4 |

| Signal Name | Type | Pin No. |
|---|---|---|
| VSS | GND | T23 |
| VSS | GND | V4 |
| VSS | GND | V23 |
| VSS | GND | W4 |
| VSS | GND | W23 |
| VSS | GND | AA4 |
| VSS | GND | AA23 |
| VSS | GND | AC4 |
| VSS | GND | AC6 |
| VSS | GND | AC8 |
| VSS | GND | AC9 |
| VSS | GND | AC11 |
| VSS | GND | AC12 |
| VSS | GND | AC13 |
| VSS | GND | AC14 |
| VSS | GND | AC15 |
| VSS | GND | AC16 |
| VSS | GND | AC18 |
| VSS | GND | AC19 |
| VSS | GND | AC21 |
| VSS | GND | AC23 |
| VSS | GND | AE1 |
| VSS | GND | AE2 |
| VSS | GND | AE25 |
| VSS | GND | AE26 |
| VSS | GND | AF1 |
| VSS | GND | AF2 |
| VSS | GND | AF14 |
| VSS | GND | AF25 |
| VSS | GND | AF26 |
| WEA# | O | R25 |
| WEB# | O | R24 |

**Note:** PU/PD indicates pin is internally connected to a weak (> 20-kohm) pull-up/-down resistor.

## Signal Definitions (Continued)




**Note:** Signal names have been abbreviated in this figure due to space constraints.

● = Denotes GND terminal

◉ = Denotes PWR terminal (VCC2 = VCC_CORE; VCC3 = VCC_IO)

**Figure 2-3. 320 SPGA Pin Assignment Diagram**

For order information, refer to Section A.1 “Order Information” on page 246.

## Signal Definitions (Continued)

### Table 2-4. 320 SPGA Pin Assignments - Sorted by Pin Number

| Pin No. | Signal Name | Pin No. | Signal Name | Pin No. | Signal Name | Pin No. | Signal Name | Pin No. | Signal Name |
|---|---|---|---|---|---|---|---|---|---|
| A3 | VCC3 | C25 | AD4 | G1 | VSS | R34 | MD44 | AB2 | PIXEL12 |
| A5 | AD25 | C27 | AD0 | G3 | CLKMODE2 | R36 | MD12 | AB4 | PIXEL13 |
| A7 | VSS | C29 | VCC2 | G5 | VSS | S1 | CLKMODE0 | AB34 | RASB# |
| A9 | VCC2 | C31 | IRQ13 | G33 | VSS | S3 | VID_VAL | AB36 | RASA# |
| A11 | AD16 | C33 | MD1 | G35 | MD37 | S5 | PIXEL0 | AC1 | VCC2 |
| A13 | VCC3 | C35 | MD34 | G37 | VSS | S33 | MD14 | AC3 | VCC2 |
| A15 | STOP# | C37 | VCC3 | H2 | GNT2# | S35 | MD13 | AC5 | VCC2 |
| A17 | SERR# | D2 | AD30 | H4 | SUSPA# | S37 | MD45 | AC33 | VCC2 |
| A19 | VSS | D4 | AD29 | H34 | MD6 | T2 | PIXEL1 | AC35 | VCC2 |
| A21 | AD11 | D6 | AD24 | H36 | MD38 | T4 | PIXEL2 | AC37 | VCC2 |
| A23 | AD8 | D8 | AD22 | J1 | TDO | T34 | MD15 | AD2 | CRT_HSYNC |
| A25 | VCC3 | D10 | AD20 | J3 | VSS | T36 | MD46 | AD4 | DCLK |
| A27 | AD2 | D12 | AD17 | J5 | TEST | U1 | VSS | AD34 | MA2 |
| A29 | VCC2 | D14 | IRDY# | J33 | VCC2 | U3 | VCC3 | AD36 | MA0 |
| A31 | VSS | D16 | PERR# | J35 | VSS | U5 | VSS | AE1 | PIXEL14 |
| A33 | TEST0 | D18 | AD14 | J37 | MD7 | U33 | VSS | AE3 | VSS |
| A35 | VCC3 | D20 | AD12 | K2 | REQ1# | U35 | VCC3 | AE5 | VCC2 |
| A37 | VSS | D22 | AD7 | K4 | GNT1# | U37 | VSS | AE33 | VCC2 |
| B2 | VSS | D24 | INTR | K34 | MD39 | V2 | PIXEL3 | AE35 | VSS |
| B4 | AD27 | D26 | TEST1 | K36 | MD8 | V4 | VID_CLK | AE37 | MA1 |
| B6 | C/BE3# | D28 | TEST3 | L1 | VCC2 | V34 | SYSCLK | AF2 | PIXEL15 |
| B8 | AD21 | D30 | MD0 | L3 | VCC2 | V36 | MD47 | AF4 | PIXEL16 |
| B10 | AD19 | D32 | MD32 | L5 | VCC2 | W1 | PIXEL6 | AF34 | MA4 |
| B12 | C/BE2# | D34 | MD3 | L33 | VCC2 | W3 | PIXEL5 | AF36 | MA3 |
| B14 | TRDY# | D36 | MD35 | L35 | VCC2 | W5 | PIXEL4 | AG1 | VSS |
| B16 | LOCK# | E1 | REQ0# | L37 | VCC2 | W33 | WEA# | AG3 | PIXEL17 |
| B18 | C/BE1# | E3 | REQ2# | M2 | RESET | W35 | WEB# | AG5 | VSS |
| B20 | AD13 | E5 | AD28 | M4 | SUSP# | W37 | CASA# | AG33 | VSS |
| B22 | AD9 | E7 | VSS | M34 | MD40 | X2 | NC | AG35 | MA5 |
| B24 | AD6 | E9 | VCC2 | M36 | MD9 | X4 | PIXEL9 | AG37 | VSS |
| B26 | AD3 | E11 | VCC2 | N1 | VCC3 | X34 | DQM0 | AH2 | CRT_VSYNC |
| B28 | SMI# | E13 | VSS | N3 | TMS | X36 | CASB# | AH4 | VID_DATA6 |
| B30 | AD1 | E15 | DEVSEL# | N5 | VSS | Y1 | PIXEL8 | AH32 | MA10 |
| B32 | TEST2 | E17 | AD15 | N33 | VSS | Y3 | VSS | AH34 | MA8 |
| B34 | MD33 | E19 | VSS | N35 | MD41 | Y5 | PIXEL7 | AH36 | MA6 |
| B36 | MD2 | E21 | C/BE0# | N37 | VCC3 | Y33 | DQM1 | AJ1 | PCLK |
| C1 | VCC3 | E23 | AD5 | P2 | FP_VSYNC | Y35 | VSS | AJ3 | FLT# |
| C3 | AD31 | E25 | VSS | P4 | TCLK | Y37 | DQM4 | AJ5 | VID_DATA5 |
| C5 | AD26 | E27 | VCC2 | P34 | MD10 | Z2 | NC | AJ7 | VSS |
| C7 | AD23 | E29 | VCC2 | P36 | MD42 | Z4 | PIXEL10 | AJ9 | VCC2 |
| C9 | VCC2 | E31 | VSS | Q1 | SERIALP | Z34 | CS2# | AJ11 | MD31 |
| C11 | AD18 | E33 | MD4 | Q3 | VSS | Z36 | DQM5 | AJ13 | VSS |
| C13 | FRAME# | E35 | MD36 | Q5 | NC | AA1 | VCC3 | AJ15 | MD60 |
| C15 | VSS | E37 | NC | Q33 | MD11 | AA3 | PIXEL11 | AJ17 | MD57 |
| C17 | PAR | F2 | GNT0# | Q35 | VSS | AA5 | VSS | AJ19 | VSS |
| C19 | VCC3 | F4 | TDI | Q37 | MD43 | AA33 | VSS | AJ21 | MD22 |
| C21 | AD10 | F34 | MD5 | R2 | CLKMODE1 | AA35 | CS0# | AJ23 | MD52 |
| C23 | VSS | F36 | NC | R4 | FP_HSYNC | AA37 | VCC3 | AJ25 | VSS |

## Signal Definitions (Continued)

**Table 2-4. 320 SPGA Pin Assignments - Sorted by Pin Number (Continued)**

| Pin No. | Signal Name | Pin No. | Signal Name | Pin No. | Signal Name | Pin No. | Signal Name | Pin No. | Signal Name |
|---|---|---|---|---|---|---|---|---|---|
| AJ27 | VCC2 | AK24 | MD20 | AL21 | MD23 | AM18 | MD25 | AN15 | MD28 |
| AJ29 | VCC2 | AK26 | MD50 | AL23 | VSS | AM20 | MD24 | AN17 | MD26 |
| AJ31 | VSS | AK28 | MD16 | AL25 | MD19 | AM22 | MD53 | AN19 | VSS |
| AJ33 | BA1 | AK30 | DQM3 | AL27 | MD49 | AM24 | MD51 | AN21 | MD54 |
| AJ35 | MA9 | AK32 | CS3# | AL29 | VCC2 | AM26 | MD18 | AN23 | CKEB |
| AJ37 | MA7 | AK34 | VSS | AL31 | DQM6 | AM28 | MD48 | AN25 | VCC3 |
| AK2 | VID_RDY | AK36 | BA0 | AL33 | CKEA | AM30 | DQM7 | AN27 | MD17 |
| AK4 | VSS | AL1 | VCC2 | AL35 | MA11 | AM32 | DQM2 | AN29 | VCC2 |
| AK6 | VID_DATA0 | AL3 | VID_DATA4 | AL37 | VCC3 | AM34 | MA12 | AN31 | VSS |
| AK8 | SDCLK0 | AL5 | VID_DATA2 | AM2 | VID_DATA7 | AM36 | NC | AN33 | CS1# |
| AK10 | SDCLK2 | AL7 | SDCLK1 | AM4 | VID_DATA3 | AN1 | VSS | AN35 | VCC3 |
| AK12 | SDCLK_IN | AL9 | VCC2 | AM6 | ENA_DISP | AN3 | VCC2 | AN37 | VSS |
| AK14 | MD29 | AL11 | RW_CLK | AM8 | SDCLK3 | AN5 | VID_DATA1 | | |
| AK16 | MD27 | AL13 | SDCLK_OUT | AM10 | MD63 | AN7 | VSS | | |
| AK18 | MD56 | AL15 | VSS | AM12 | MD30 | AN9 | VCC2 | | |
| AK20 | MD55 | AL17 | MD58 | AM14 | MD61 | AN11 | MD62 | | |
| AK22 | MD21 | AL19 | VCC3 | AM16 | MD59 | AN13 | VCC3 | | |

## Signal Definitions (Continued)

### Table 2-5. 320 SPGA Pin Assignments - Sorted Alphabetically by Signal Name

| Signal Name | Type | Pin. No. |
|---|---|---|
| AD0 | I/O | C27 |
| AD1 | I/O | B30 |
| AD2 | I/O | A27 |
| AD3 | I/O | B26 |
| AD4 | I/O | C25 |
| AD5 | I/O | E23 |
| AD6 | I/O | B24 |
| AD7 | I/O | D22 |
| AD8 | I/O | A23 |
| AD9 | I/O | B22 |
| AD10 | I/O | C21 |
| AD11 | I/O | A21 |
| AD12 | I/O | D20 |
| AD13 | I/O | B20 |
| AD14 | I/O | D18 |
| AD15 | I/O | E17 |
| AD16 | I/O | A11 |
| AD17 | I/O | D12 |
| AD18 | I/O | C11 |
| AD19 | I/O | B10 |
| AD20 | I/O | D10 |
| AD21 | I/O | B8 |
| AD22 | I/O | D8 |
| AD23 | I/O | C7 |
| AD24 | I/O | D6 |
| AD25 | I/O | A5 |
| AD26 | I/O | C5 |
| AD27 | I/O | B4 |
| AD28 | I/O | E5 |
| AD29 | I/O | D4 |
| AD30 | I/O | D2 |
| AD31 | I/O | C3 |
| BA0 | O | AK36 |
| BA1 | O | AJ33 |
| CASA# | O | W37 |
| CASB# | O | X36 |
| C/BE0# | I/O | E21 |
| C/BE1# | I/O | B18 |
| C/BE2# | I/O | B12 |
| C/BE3# | I/O | B6 |
| CKEA | O | AL33 |
| CKEB | O | AN23 |
| CLKMODE0 | I | S1 |
| CLKMODE1 | I | R2 |
| CLKMODE2 | I | G3 |
| CRT_HSYNC | O | AD2 |
| CRT_VSYNC | O | AH2 |
| CS0# | O | AA35 |
| CS1# | O | AN33 |
| CS2# | O | Z34 |
| CS3# | O | AK32 |
| DCLK | I | AD4 |
| DEVSEL# | s/t/s | E15 (PU) |

| Signal Name | Type | Pin. No. |
|---|---|---|
| DQM0 | O | X34 |
| DQM1 | O | Y33 |
| DQM2 | O | AM32 |
| DQM3 | O | AK30 |
| DQM4 | O | Y37 |
| DQM5 | O | Z36 |
| DQM6 | O | AL31 |
| DQM7 | O | AM30 |
| ENA_DISP | O | AM6 |
| FLT# | I | AJ3 |
| FP_HSYNC | O | R4 |
| FP_VSYNC | O | P2 |
| FRAME# | s/t/s | C13 (PU) |
| GNT0# | O | F2 |
| GNT1# | O | K4 |
| GNT2# | O | H2 |
| INTR | I | D24 |
| IRDY# | s/t/s | D14 (PU) |
| IRQ13 | O | C31 |
| LOCK# | s/t/s | B16 (PU) |
| MA0 | O | AD36 |
| MA1 | O | AE37 |
| MA2 | O | AD34 |
| MA3 | O | AF36 |
| MA4 | O | AF34 |
| MA5 | O | AG35 |
| MA6 | O | AH36 |
| MA7 | O | AJ37 |
| MA8 | O | AH34 |
| MA9 | O | AJ35 |
| MA10 | O | AH32 |
| MA11 | O | AL35 |
| MA12 | O | AM34 |
| MD0 | I/O | D30 |
| MD1 | I/O | C33 |
| MD2 | I/O | B36 |
| MD3 | I/O | D34 |
| MD4 | I/O | E33 |
| MD5 | I/O | F34 |
| MD6 | I/O | H34 |
| MD7 | I/O | J37 |
| MD8 | I/O | K36 |
| MD9 | I/O | M36 |
| MD10 | I/O | P34 |
| MD11 | I/O | Q33 |
| MD12 | I/O | R36 |
| MD13 | I/O | S35 |
| MD14 | I/O | S33 |
| MD15 | I/O | T34 |
| MD16 | I/O | AK28 |
| MD17 | I/O | AN27 |
| MD18 | I/O | AM26 |
| MD19 | I/O | AL25 |

| Signal Name | Type | Pin. No. |
|---|---|---|
| MD20 | I/O | AK24 |
| MD21 | I/O | AK22 |
| MD22 | I/O | AJ21 |
| MD23 | I/O | AL21 |
| MD24 | I/O | AM20 |
| MD25 | I/O | AM18 |
| MD26 | I/O | AN17 |
| MD27 | I/O | AK16 |
| MD28 | I/O | AN15 |
| MD29 | I/O | AK14 |
| MD30 | I/O | AM12 |
| MD31 | I/O | AJ11 |
| MD32 | I/O | D32 |
| MD33 | I/O | B34 |
| MD34 | I/O | C35 |
| MD35 | I/O | D36 |
| MD36 | I/O | E35 |
| MD37 | I/O | G35 |
| MD38 | I/O | H36 |
| MD39 | I/O | K34 |
| MD40 | I/O | M34 |
| MD41 | I/O | N35 |
| MD42 | I/O | P36 |
| MD43 | I/O | Q37 |
| MD44 | I/O | R34 |
| MD45 | I/O | S37 |
| MD46 | I/O | T36 |
| MD47 | I/O | V36 |
| MD48 | I/O | AM28 |
| MD49 | I/O | AL27 |
| MD50 | I/O | AK26 |
| MD51 | I/O | AM24 |
| MD52 | I/O | AJ23 |
| MD53 | I/O | AM22 |
| MD54 | I/O | AN21 |
| MD55 | I/O | AK20 |
| MD56 | I/O | AK18 |
| MD57 | I/O | AJ17 |
| MD58 | I/O | AL17 |
| MD59 | I/O | AM16 |
| MD60 | I/O | AJ15 |
| MD61 | I/O | AM14 |
| MD62 | I/O | AN11 |
| MD63 | I/O | AM10 |
| NC | -- | E37 |
| NC | -- | F36 |
| NC | -- | Q5 |
| NC | -- | X2 |
| NC | -- | Z2 |
| NC | -- | AM36 |
| PAR | I/O | C17 |
| PCLK | O | AJ1 |
| PERR# | s/t/s | D16 (PU) |

| Signal Name | Type | Pin. No. |
|---|---|---|
| PIXEL0 | O | S5 |
| PIXEL1 | O | T2 |
| PIXEL2 | O | T4 |
| PIXEL3 | O | V2 |
| PIXEL4 | O | W5 |
| PIXEL5 | O | W3 |
| PIXEL6 | O | W1 |
| PIXEL7 | O | Y5 |
| PIXEL8 | O | Y1 |
| PIXEL9 | O | X4 |
| PIXEL10 | O | Z4 |
| PIXEL11 | O | AA3 |
| PIXEL12 | O | AB2 |
| PIXEL13 | O | AB4 |
| PIXEL14 | O | AE1 |
| PIXEL15 | O | AF2 |
| PIXEL16 | O | AF4 |
| PIXEL17 | O | AG3 |
| RASA# | O | AB36 |
| RASB# | O | AB34 |
| REQ0# | I | E1 (PU) |
| REQ1# | I | K2 (PU) |
| REQ2# | I | E3 (PU) |
| RESET | I | M2 |
| RW_CLK | O | AL11 |
| SDCLK_IN | I | AK12 |
| SDCLK_OUT | O | AL13 |
| SDCLK0 | O | AK8 |
| SDCLK1 | O | AL7 |
| SDCLK2 | O | AK10 |
| SDCLK3 | O | AM8 |
| SERIALP | O | Q1 |
| SERR# | OD | A17 (PU) |
| SMI# | I | B28 |
| STOP# | s/t/s | A15 (PU) |
| SUSP# | I | M4 (PU) |
| SUSPA# | O | H4 |
| SYSCLK | I | V34 |
| TCLK | I | P4 (PU) |
| TDI | I | F4 (PU) |
| TDO | O | J1 |
| TEST | I | J5 (PD) |
| TEST0 | O | A33 |
| TEST1 | O | D26 |
| TEST2 | O | B32 |
| TEST3 | O | D28 |
| TMS | I | N3 (PU) |
| TRDY# | s/t/s | B14 (PU) |
| VCC2 | PWR | A9 |
| VCC2 | PWR | A29 |
| VCC2 | PWR | C9 |
| VCC2 | PWR | C29 |
| VCC2 | PWR | E9 |

## Signal Definitions (Continued)

**Table 2-5. 320 SPGA Pin Assignments - Sorted Alphabetically by Signal Name (Continued)**

| Signal Name | Type | Pin. No. |
|---|---|---|
| VCC2 | PWR | E11 |
| VCC2 | PWR | E27 |
| VCC2 | PWR | E29 |
| VCC2 | PWR | J33 |
| VCC2 | PWR | L1 |
| VCC2 | PWR | L3 |
| VCC2 | PWR | L5 |
| VCC2 | PWR | L33 |
| VCC2 | PWR | L35 |
| VCC2 | PWR | L37 |
| VCC2 | PWR | AC1 |
| VCC2 | PWR | AC3 |
| VCC2 | PWR | AC5 |
| VCC2 | PWR | AC33 |
| VCC2 | PWR | AC35 |
| VCC2 | PWR | AC37 |
| VCC2 | PWR | AE5 |
| VCC2 | PWR | AE33 |
| VCC2 | PWR | AJ9 |
| VCC2 | PWR | AJ27 |
| VCC2 | PWR | AJ29 |
| VCC2 | PWR | AL1 |
| VCC2 | PWR | AL9 |
| VCC2 | PWR | AL29 |
| VCC2 | PWR | AN3 |
| VCC2 | PWR | AN9 |
| VCC2 | PWR | AN29 |
| VCC3 | PWR | A3 |
| VCC3 | PWR | A13 |

| Signal Name | Type | Pin. No. |
|---|---|---|
| VCC3 | PWR | A25 |
| VCC3 | PWR | A35 |
| VCC3 | PWR | C1 |
| VCC3 | PWR | C19 |
| VCC3 | PWR | C37 |
| VCC3 | PWR | N1 |
| VCC3 | PWR | N37 |
| VCC3 | PWR | U3 |
| VCC3 | PWR | U35 |
| VCC3 | PWR | AA1 |
| VCC3 | PWR | AA37 |
| VCC3 | PWR | AL19 |
| VCC3 | PWR | AL37 |
| VCC3 | PWR | AN13 |
| VCC3 | PWR | AN25 |
| VCC3 | PWR | AN35 |
| VID_CLK | O | V4 |
| VID_DATA0 | O | AK6 |
| VID_DATA1 | O | AN5 |
| VID_DATA2 | O | AL5 |
| VID_DATA3 | O | AM4 |
| VID_DATA4 | O | AL3 |
| VID_DATA5 | O | AJ5 |
| VID_DATA6 | O | AH4 |
| VID_DATA7 | O | AM2 |
| VID_RDY | I | AK2 |
| VID_VAL | O | S3 |
| VSS | GND | A7 |
| VSS | GND | A19 |

| Signal Name | Type | Pin. No. |
|---|---|---|
| VSS | GND | A31 |
| VSS | GND | A37 |
| VSS | GND | B2 |
| VSS | GND | C15 |
| VSS | GND | C23 |
| VSS | GND | E7 |
| VSS | GND | E13 |
| VSS | GND | E19 |
| VSS | GND | E25 |
| VSS | GND | E31 |
| VSS | GND | G1 |
| VSS | GND | G5 |
| VSS | GND | G33 |
| VSS | GND | G37 |
| VSS | GND | J3 |
| VSS | GND | J35 |
| VSS | GND | N5 |
| VSS | GND | N33 |
| VSS | GND | Q3 |
| VSS | GND | Q35 |
| VSS | GND | U1 |
| VSS | GND | U5 |
| VSS | GND | U33 |
| VSS | GND | U37 |
| VSS | GND | Y3 |
| VSS | GND | Y35 |
| VSS | GND | AA5 |
| VSS | GND | AA33 |
| VSS | GND | AE3 |

| Signal Name | Type | Pin. No. |
|---|---|---|
| VSS | GND | AE35 |
| VSS | GND | AG1 |
| VSS | GND | AG5 |
| VSS | GND | AG33 |
| VSS | GND | AG37 |
| VSS | GND | AJ7 |
| VSS | GND | AJ13 |
| VSS | GND | AJ19 |
| VSS | GND | AJ25 |
| VSS | GND | AJ31 |
| VSS | GND | AK4 |
| VSS | GND | AK34 |
| VSS | GND | AL15 |
| VSS | GND | AL23 |
| VSS | GND | AN1 |
| VSS | GND | AN7 |
| VSS | GND | AN19 |
| VSS | GND | AN31 |
| VSS | GND | AN37 |
| WEA# | O | W33 |
| WEB# | O | W35 |

**Note:** PU/PD indicates pin is internally connected to a weak (> 20-kohm) pull-up/down resistor.

## 2.2 SIGNAL DESCRIPTIONS

### 2.2.1 System Interface Signals

| Signal Name | BGA Pin No. | SPGA Pin No. | Type | Description |
|---|---|---|---|---|
| SYSCLK | P26 | V34 | I | **System Clock**<br>PCI clock is connected to SYSCLK. The internal clock of the GXLV processor is generated by a proprietary patented frequency synthesis circuit which multiplies the SYSCLK input up to ten times. The SYSCLK to core clock multiplier is configured using the CLKMODE[2:0] inputs.<br>The SYSCLK input is a fixed frequency which can only be stopped or varied when the GXLV processor is in full 3V Suspend. (See Section 5.1.4 “3 Volt Suspend” on page 177 for details regarding this mode.) |
| CLKMODE[2:0] | M1, L1, M3 | G3, R2, S1 | I | **Clock Mode**<br>These signals are used to set the core clock multiplier. The PCI clock "SYSCLK" is multiplied by the value set by CLKMODE[2:0] to generate the GXLV processor’s core clock.<br>CLKMODE[2:0]:<br>000 = SYSCLK multiplied by 4 (Test mode only)<br>001 = SYSCLK multiplied by 10<br>010 = SYSCLK multiplied by 9<br>011 = SYSCLK multiplied by 5<br>100 = SYSCLK multiplied by 4<br>101 = SYSCLK multiplied by 6<br>110 = SYSCLK multiplied by 7<br>111 = SYSCLK multiplied by 8 |
| RESET | J3 | M2 | I | **Reset**<br>RESET aborts all operations in progress and places the GXLV processor into a reset state. RESET forces the CPU and peripheral functions to begin executing at a known state. All data in the on-chip cache is invalidated upon RESET.<br>RESET is an asynchronous input but must meet specified setup and hold times to guarantee recognition at a particular clock edge. This input is typically generated during the Power-On-Reset sequence. |
| INTR | B18 | D24 | I | **(Maskable) Interrupt Request**<br>INTR is a level-sensitive input that causes the GXLV processor to suspend execution of the current instruction stream and begin execution of an interrupt service routine. The INTR input can be masked through the EFlags Register IF bit. (See Table 3-4 on page 46 for bit definitions.) |
| IRQ13 | C22 | C31 | O | **Interrupt Request Level 13**<br>IRQ13 is asserted if an on-chip floating point error occurs.<br>When a floating point error occurs, the GXLV processor asserts the IRQ13 pin. The floating point interrupt handler then performs an OUT instruction to I/O address F0h or F1h. The GXLV processor accepts either of these cycles and clears the IRQ13 pin.<br>Refer to Section 3.4.1 “I/O Address Space” on page 63 for further information on IN/OUT instructions. |

## Signal Definitions (Continued)

### 2.2.1 System Interface Signals (Continued)

| Signal Name | BGA Pin No. | SPGA Pin No. | Type | Description |
|---|---|---|---|---|
| SMI# | C19 | B28 | I | **System Management Interrupt**<br>SMI# is a level-sensitive interrupt. SMI# puts the GXLV processor into System Management Mode (SMM). |
| SUSP# | H2 (PU) | M4 (PU) | I | **Suspend Request**<br>This signal is used to request that the GXLV processor enter Suspend mode. After recognition of an active SUSP# input, the processor completes execution of the current instruction, any pending decoded instructions and associated bus cycles. SUSP# is enabled by setting the SUSP bit in CCR2, and is ignored following RESET. (See Table 3-11 on page 52 for CCR2 bit definitions.)<br>Since the GXLV processor includes system logic functions as well as the CPU core, there are special modes designed to support the different power management states associated with APM, ACPI, and portable designs. The part can be configured to stop only the CPU core clocks, or all clocks. When all clocks are stopped, the external clock can also be stopped. (See Section 5.0 “Power Management” on page 176 for more details regarding power management states.)<br>This pin is internally connected to a weak (>20-kohm) pull-up resistor. |
| SUSPA# | E2 | H4 | O | **Suspend Acknowledge**<br>Suspend Acknowledge indicates that the GXLV processor has entered low-power Suspend mode as a result of SUSP# assertion or execution of a HALT instruction. SUSPA# floats following RESET and is enabled by setting the SUSP bit in CCR2. (See Table 3-11 on page 52 for CCR2 bit definitions.)<br>The SYSCLK input may be stopped after SUSPA# has been asserted to further reduce power consumption if the system is configured for 3V Suspend mode. (see Section 5.1.4 “3 Volt Suspend” on page 177 for details regarding this mode). |
| SERIALP | L3 | Q1 | O | **Serial Packet**<br>Serial Packet is the single wire serial-transmission signal to the CS5530 chip. The clock used for this interface is SYSCLK. This interface carries packets of miscellaneous information to the chipset to be used by the VSA technology software handlers. |

## Signal Definitions (Continued)

### 2.2.2 PCI Interface Signals

| Signal Name | BGA Pin No. | SPGA Pin No | Type | Description |
|---|---|---|---|---|
| FRAME# | A8 (PU) | C13 (PU) | s/t/s | **Frame**<br>FRAME# is driven by the current master to indicate the beginning and duration of an access. FRAME# is asserted to indicate a bus transaction is beginning. While FRAME# is asserted, data transfers continue. When FRAME# is deasserted, the transaction is in the final data phase.<br>This pin is internally connected to a weak (>20-kohm) pull-up resistor. |
| IRDY# | C9 (PU) | D14 (PU) | s/t/s | **Initiator Ready**<br>IRDY# is asserted to indicate that the bus master is able to complete the current data phase of the transaction. IRDY# is used in conjunction with TRDY#. A data phase is completed on any SYSCLK in which both IRDY# and TRDY# are sampled asserted. During a write, IRDY# indicates valid data is present on AD[31:0]. During a read, it indicates the master is prepared to accept data. Wait cycles are inserted until both IRDY# and TRDY# are asserted together.<br>This pin is internally connected to a weak (>20-kohm) pull-up resistor. |
| TRDY# | B9 (PU) | B14 (PU) | s/t/s | **Target Ready**<br>TRDY# is asserted to indicate that the target agent is able to complete the current data phase of the transaction. TRDY# is used in conjunction with IRDY#. A data phase is complete on any SYSCLK in which both TRDY# and IRDY# are sampled asserted. During a read, TRDY# indicates that valid data is present on AD[31:0]. During a write, it indicates the target is prepared to accept data. Wait cycles are inserted until both IRDY# and TRDY# are asserted together.<br>This pin is internally connected to a weak (>20-kohm) pull-up resistor. |
| STOP# | C11 (PU) | A15 (PU) | s/t/s | **Target Stop**<br>STOP# is asserted to indicate that the current target is requesting the master to stop the current transaction. This signal is used with DEVSEL# to indicate retry, disconnect or target abort. If STOP# is sampled active while a master, FRAME# will be deasserted and the cycle will be stopped within three SYSCLKs. STOP# can be asserted in the following cases:<br>• A PCI master tries to access memory that has been locked by another master. This condition is detected if FRAME# and LOCK# are asserted during an address phase.<br>• The PCI write buffers are full or a previously buffered cycle has not completed.<br>• Read cycles that cross cache line boundaries. This is conditional based upon the programming of bit 1 in the PCI Control Function 2 Register.<br>This pin is internally connected to a weak (>20-kohm) pull-up resistor. |

## Signal Definitions (Continued)

### 2.2.2 PCI Interface Signals (Continued)

| Signal Name | BGA Pin No. | SPGA Pin No | Type | Description |
|---|---|---|---|---|
| AD[31:0] | Refer to Table 2-3 | Refer to Table 2-5 | I/O | **Multiplexed Address and Data**<br>Addresses and data are multiplexed together on the same pins. A bus transaction consists of an address phase in the cycle in which FRAME# is asserted followed by one or more data phases. During the address phase, AD[31:0] contain a physical 32-bit address. During data phases, AD[7:0] contain the least significant byte (LSB) and AD[31:24] contain the most significant byte (MSB). Write data is stable and valid when IRDY# is asserted and read data is stable and valid when TRDY# is asserted. Data is transferred during the SYSCLK when both IRDY# and TRDY# are asserted. |
| C/BE[3:0]# | D5, B8, C13, A15 | B6, B12, B18, E21 | I/O | **Multiplexed Command and Byte Enables**<br>C/BE# are the bus commands and byte enables. They are multiplexed together on the same PCI pins. During the address phase of a transaction when FRAME# is active, C/BE[3:0]# define the bus command. During the data phase C/BE[3:0]# are used as byte enables. The byte enables are valid for the entire data phase and determine which byte lanes carry meaningful data. C/BE0# applies to byte 0 (LSB) and C/BE3# applies to byte 3 (MSB).<br>The command encoding and types are listed below.<br>0000 = Interrupt Acknowledge<br>0001 = Special Cycle<br>0010 = I/O Read<br>0011 = I/O Write<br>0100 = Reserved<br>0101 = Reserved<br>0110 = Memory Read<br>0111 = Memory Write<br>1000 = Reserved<br>1001 = Reserved<br>1010 = Configuration Read<br>1011 = Configuration Write<br>1100 = Memory Read Multiple<br>1101 = Dual Address Cycle (Reserved)<br>1110 = Memory Read Line<br>1111 = Memory Write and Invalidate |
| PAR | B12 | C17 | I/O | **Parity**<br>PAR is used with AD[31:0] and C/BE[3:0]# to generate even parity. Parity generation is required by all PCI agents: the master drives PAR for address and write-data phases, the target drives PAR for read-data phases.<br>For address phases, PAR is stable and valid one SYSCLK after the address phase.<br>For data phases, PAR is stable and valid one SYSCLK after either IRDY# is asserted on a write transaction or after TRDY# is asserted on a read transaction. Once PAR is valid, it remains valid until one SYSCLK after the completion of the data phase. (Also see PERR# description on page 35.) |

## Signal Definitions (Continued)

### 2.2.2 PCI Interface Signals (Continued)

| Signal Name | BGA Pin No. | SPGA Pin No | Type | Description |
|---|---|---|---|---|
| LOCK# | B11 (PU) | B16 (PU) | s/t/s | **Lock Operation**<br>LOCK# indicates an atomic operation that may require multiple transactions to complete. When LOCK# is asserted, nonexclusive transactions may proceed to an address that is not currently locked (at least 16 bytes must be locked). A grant to start a transaction on PCI does not guarantee control of LOCK#. Control of LOCK# is obtained under its own protocol in conjunction with GNT#. It is possible for different agents to use PCI while a single master retains ownership of LOCK#. The arbiter can implement a complete system lock. In this mode, if LOCK# is active, no other master can gain access to the system until the LOCK# is deasserted.<br>This pin is internally connected to a weak (>20-kohm) pull-up resistor. |
| DEVSEL# | A9 (PU) | E15 (PU) | s/t/s | **Device Select**<br>DEVSEL# indicates that the driving device has decoded its address as the target of the current access. As an input, DEVSEL# indicates whether any device on the bus has been selected. DEVSEL# will also be driven by any agent that has the ability to accept cycles on a subtractive decode basis. As a master, if no DEVSEL# is detected within and up to the subtractive decode clock, a master abort cycle will result except for special cycles which do not expect a DEVSEL# returned.<br>This pin is internally connected to a weak (>20-kohm) pull-up resistor. |
| PERR# | A11 (PU) | D16 (PU) | s/t/s | **Parity Error**<br>PERR# is used for the reporting of data parity errors during all PCI transactions except a Special Cycle. The PERR# line is driven two SYSCLKs after the data in which the error was detected, which is one SYSCLK after the PAR that was attached to the data. The minimum duration of PERR# is one SYSCLK for each data phase in which a data parity error is detected. PERR# must be driven high for one SYSCLK before going to TRI-STATE. A target asserts PERR# on write cycles if it has claimed the cycle with DEVSEL#. The master asserts PERR# on read cycles.<br>This pin is internally connected to a weak (>20-kohm) pull-up resistor. |
| SERR# | C12 (PU) | A17 (PU) | OD | **System Error**<br>SERR# may be asserted by any agent for reporting errors other than PCI parity. The intent is to have the PCI central agent assert NMI to the processor. When the Parity Enable bit is set in the Memory Controller Configuration register, SERR# will be asserted upon detecting a parity error on read operations from DRAM. |
| REQ[2:0]# | D3, H3, E3 (PU) | E3, K2, E1 (PU) | I | **Request Lines**<br>REQ# indicates to the arbiter that an agent desires use of the bus. Each master has its own REQ# line. REQ# priorities are based on the arbitration scheme chosen.<br>This pin is internally connected to a weak (>20-kohm) pull-up resistor. |

## Signal Definitions (Continued)

### 2.2.2 PCI Interface Signals (Continued)

| Signal Name | BGA Pin No. | SPGA Pin No | Type | Description |
|---|---|---|---|---|
| GNT[2:0]# | E1, F2, D1 | H2, K4, F2 | O | **Grant Lines**<br>GNT# indicates to the requesting master that it has been granted access to the bus. Each master has its own GNT# line. GNT# can be pulled away at any time a higher REQ# is received or if the master does not begin a cycle within a minimum period of time (16 SYSCLKs). |

### 2.2.3 Memory Controller Interface Signals

| Signal Name | BGA Pin No. | SPGA Pin No. | Type | Description |
|---|---|---|---|---|
| MD[63:0] | Refer to Table 2-3 | Refer to Table 2-5 | I/O | **Memory Data Bus**<br>The data bus lines driven to/from system memory. |
| MA[12:0] | Refer to Table 2-3 | Refer to Table 2-5 | O | **Memory Address Bus**<br>The multiplexed row/column address lines driven to the system memory.<br>Supports 256 MB SDRAM. |
| BA[1:0] | AD26, AD25 | AJ33, AK36 | O | **Bank Address Bits**<br>These bits are used to select the component bank within the SDRAM. |
| CS[3:0]# | AE23, V25, AD23, V26 | AK32, Z34, AN33, AA35 | O | **Chip Selects**<br>The chip selects are used to select the module bank within the system memory. Each chip select corresponds to a specific module bank.<br>If CS# is high, the bank(s) do not respond to RAS#, CAS#, WE# until the bank is selected again. |
| RASA#, RASB# | W24, W25 | AB36, AB34 | O | **Row Address Strobe**<br>RAS#, CAS#, WE# and CKE are encoded to support the different SDRAM commands. RASA# is used with CS[1:0]#. RASB# is used with CS[3:2]#. |
| CASA#, CASB# | P25, R26 | W37, X36 | O | **Column Address Strobe**<br>RAS#, CAS#, WE# and CKE are encoded to support the different SDRAM commands. CASA# is used with CS[1:0]#. CASB# is used with CS[3:2]#. |
| WEA#, WEB# | R25, R24 | W33, W35 | O | **Write Enable**<br>RAS#, CAS#, WE# and CKE are encoded to support the different SDRAM commands. WEA# is used with CS[1:0]#. WEB# is used with CS[3:2]#. |
| CKEA, CKEB | AF24, AD16 | AL33, AN23 | O | **Clock Enable**<br>For normal operation, CKE is held high. CKE goes low during SUSPEND. CKEA is used with CS[1:0]#. CKEB is used with CS[3:2]#. |

## Signal Definitions (Continued)

### 2.2.3 Memory Controller Interface Signals (Continued)

| Signal Name | BGA Pin No. | SPGA Pin No. | Type | Description |
|---|---|---|---|---|
| DQM[7:0] | Refer to Table 2-3 | Refer to Table 2-5 | O | **Data Mask Control Bits**<br>During memory read cycles, these outputs control whether the SDRAM output buffers are driven on the MD bus or not. All DQM signals are asserted during read cycles.<br>During memory write cycles, these outputs control whether or not MD data will be written into the SDRAM.<br>DQM[0] is associated with MD[7:0].<br>DQM[7] is associated with MD[63:56]. |
| SDCLK[3:0] | AE4, AF5, AE5, AF4 | AM8, AK10, AL7, AK8 | O | **SDRAM Clocks**<br>The SDRAM devices sample all the control, address, and data based on these clocks. |
| SDCLK_IN | AE8 | AK12 | I | **SDRAM Clock Input**<br>The GXLV processor samples the memory read data on this clock. Works in conjunction with the SDCLK_OUT signal. |
| SDCLK_OUT | AF8 | AL13 | O | **SDRAM Clock Output**<br>This output is routed back to SDCLK_IN. The board designer should vary the length of the board trace to control skew between SDCLK_IN and SDCLK. |

### 2.2.4 Video Interface Signals

| Signal Name | BGA Pin No | SPGA Pin No | Type | Description |
|---|---|---|---|---|
| PCLK | AC1 | AJ1 | O | **Pixel Port Clock**<br>PCLK is the pixel dot clock output. It clocks the pixel data from the GXLV processor to the CS5530. |
| VID_CLK | P1 | V4 | O | **Video Clock**<br>VID_CLK is the video port clock to the CS5530. |
| DCLK | AB1 | AD4 | I | **Dot Clock**<br>The DCLK input is driven from the CS5530 and is the pixel dot clock. In some cases this clock can be a 2x multiple of PCLK |
| CRT_HSYNC | W2 | AD2 | O | **CRT Horizontal Sync**<br>CRT Horizontal Sync establishes the line rate and horizontal retrace interval for an attached CRT. The polarity is programmable. See DC-Timing_CFG Register in Table 4-29 on page 146 for programming information. |
| CRT_VSYNC | AA3 | AH2 | O | **CRT Vertical Sync**<br>CRT Vertical Sync establishes the screen refresh rate and vertical retrace interval for an attached CRT. The polarity is programmable. See DC-Timing_CFG Register in Table 4-29 on page 147 for programming information. |

## Signal Definitions (Continued)

### 2.2.4 Video Interface Signals (Continued)

| Signal Name | BGA Pin No | SPGA Pin No | Type | Description |
|---|---|---|---|---|
| FP_HSYNC | L2 | R4 | O | **Flat Panel Horizontal Sync**<br>Flat Panel Horizontal Sync establishes the line rate and horizontal retrace interval for a TFT display. Polarity is programmable. (See Table 4-31 on page 146 for programming information.)<br>This signal is an input to the CS5530. The CS5530 re-drives this signal to the flat panel.<br>If no flat panel is used in the system, this signal is not connected. |
| FP_VSYNC | J1 | P2 | O | **Flat Panel Vertical Sync**<br>Flat Panel Vertical Sync establishes the screen refresh rate and vertical retrace interval for a TFT display. Polarity is programmable. (See Table 4-31 on page 146 for programming information.)<br>This signal is an input to the CS5530. The CS5530 re-drives this signal to the flat panel.<br>If no flat panel is used in the system, this signal is not connected. |
| ENA_DISP | AD5 | AM6 | O | **Display Enable**<br>Display Enable indicates the active display portion of a scan line to the CS5530.<br>In a CS5530-based system, this signal is required to be connected. |
| VID_RDY | AD1 | AK2 | I | **Video Ready**<br>This input signal indicates that the video FIFO in the CS5530 is ready to receive more data. |
| VID_VAL | M2 | S3 | O | **Video Valid**<br>VID_VAL indicates that video data to the CS5530 is valid. |
| VID_DATA[7:0] | Refer to Table 2-3 | Refer to Table 2-5 | O | **Video Data Bus**<br>When the Video Port is enabled, this bus drives Video (YUV or RGB 5:6:5) data synchronous to the VID_CLK output. |
| PIXEL[17:0] | Refer to Table 2-3 | Refer to Table 2-5 | O | **Graphics Pixel Data Bus**<br>This bus drives graphics pixel data synchronous to the PCLK output. |

## Signal Definitions (Continued)

### 2.2.5 Power, Ground, and No Connect Signals

| Signal Name | BGA Pin No. | SPGA Pin No. | Type | Description |
|---|---|---|---|---|
| VSS | Refer to Table 2-3 (Total of 71) | Refer to Table 2-5 (Total of 50) | GND | **Ground Connection** |
| VCC2 | Refer to Table 2-3 (Total of 32) | Refer to Table 2-5 (Total of 32) | PWR | **2.2V, 2.5V, or 2.9V (Nominal) Core Power Connection** |
| VCC3 | Refer to Table 2-3 (Total of 32) | Refer to Table 2-5 (Total of 18) | PWR | **3.3V (Nominal) I/O Power Connection** |
| NC | D26, E24, AC5 | E37, F36, Q5, X2, Z2, AM36 | | **No Connection**<br>A line designated as NC must be left disconnected. |

### 2.2.6 Internal Test and Measurement Signals

| Signal Name | BGA Pin No. | SPGA Pin No. | Type | Description |
|---|---|---|---|---|
| FLT# | AC2 | AJ3 | I | **Float**<br>Float forces the GXLV processor to float all outputs in the high-impedance state and to enter a power-down state. |
| RW_CLK | AE6 | AL11 | O | **Raw Clock**<br>This output is the GXLV processor clock. This debug signal can be used to verify clock operation. |
| TEST[3:0] | B22, A23, B21, C21 | D28, B32, D26, A33 | O | **SDRAM Test Outputs**<br>These outputs are used for internal debug only. |
| TCLK | J2 (PU) | P4 (PU) | I | **Test Clock**<br>JTAG test clock.<br>This pin is internally connected to a weak (>20-kohm) pull-up resistor. |
| TDI | D2 (PU) | F4 (PU) | I | **Test Data Input**<br>JTAG serial test-data input.<br>This pin is internally connected to a weak (>20-kohm) pull-up resistor. |
| TDO | F1 | J1 | O | **Test Data Output**<br>JTAG serial test-data output. |

## Signal Definitions (Continued)

### 2.2.6 Internal Test and Measurement Signals (Continued)

| Signal Name | BGA Pin No. | SPGA Pin No. | Type | Description |
|---|---|---|---|---|
| TMS | H1 (PU) | N3 (PU) | I | **Test Mode Select**<br>JTAG test-mode select.<br>This pin is internally connected to a weak (>20-kohm) pull-up resistor. |
| TEST | F3 (PD) | J5 (PD) | I | **Test**<br>Test-mode input.<br>This pin is internally connected to a weak (>20-kohm) pull-up resistor. |

# 3.0 Processor Programming

This section describes the internal operations of the Geode GXLV processor from a programmer's point of view. It includes a description of the traditional "core" processing and FPU operations. The integrated function registers are described at the end of this chapter.

The primary register sets within the processor core include:

- Application Register Set
- System Register Set
- Model Specific Register Set

The initialization of the major registers within the core are shown in Table 3-1.

The integrated function sets are located in main memory space and include:

- Internal Bus Interface Unit Register Set
- Graphics Pipeline Register Set
- Display Controller Register Set
- Memory Controller Register Set
- Power Management Register Set

## 3.1 CORE PROCESSOR INITIALIZATION

The GXLV processor is initialized when the RESET signal is asserted. The processor is placed in real mode and the registers listed in Table 3-1 are set to their initialized values. RESET invalidates and disables the CPU cache, and turns off paging. When RESET is asserted, the CPU terminates all local bus activity and all internal execution. While RESET is asserted the internal pipeline is flushed and no instruction execution or bus activity occurs.

Approximately 150 to 250 external clock cycles after RESET is deasserted, the processor begins executing instructions at the top of physical memory (address location FFFFFFF0h). The actual number of clock cycles depends on the clock scaling in use. Also, before execution begins, an additional $2^{20}$ clock cycles are needed when self-test is requested.

Typically, an intersegment jump is placed at FFFFFFF0h. This instruction will force the processor to begin execution in the lowest 1 MB of address space.

Table 3-1 lists the core registers and illustrates how they are initialized.

**Table 3-1. Initialized Core Register Controls**

| Register | Register Name | Initialized Contents | Comments |
|---|---|---|---|
| EAX | Accumulator | xxxxxxxxh | 0000 0000h indicates self-test passed. |
| EBX | Base | xxxxxxxxh | |
| ECX | Count | xxxxxxxxh | |
| EDX | Data | xxxx 04 [DIR0]h | DIR0 = Device ID |
| EBP | Base Pointer | xxxxxxxxh | |
| ESI | Source Index | xxxxxxxxh | |
| EDI | Destination Index | xxxxxxxxh | |
| ESP | Stack Pointer | xxxxxxxxh | |
| EFLAGS | Flags | 00000002h | See Table 3-4 on page 46 for bit definitions. |
| EIP | Instruction Pointer | 0000FFF0h | |
| ES | Extra Segment | 0000h | Base address set to 00000000h. Limit set to FFFFh. |
| CS | Code Segment | F000h | Base address set to FFFF0000h. Limit set to FFFFh. |
| SS | Stack Segment | 0000h | Base address set to 00000000h. Limit set to FFFFh. |
| DS | Data Segment | 0000h | Base address set to 00000000h. Limit set to FFFFh. |
| FS | Extra Segment | 0000h | Base address set to 00000000h. Limit set to FFFFh. |
| GS | Extra Segment | 0000h | Base address set to 00000000h. Limit set to FFFFh. |
| IDTR | Interrupt Descriptor Table Register | Base = 0, Limit = 3FFh | |
| GDTR | Global Descriptor Table Register | xxxxxxxxh | |
| LDTR | Local Descriptor Table Register | xxxxh | |
| TR | Task Register | xxxxh | |
| CR0 | Control Register 0 | 60000010h | See Table 3-7 on page 49 for bit definitions. |
| CR2 | Control Register 2 | xxxxxxxxh | See Table 3-7 on page 49 for bit definitions. |
| CR3 | Control Register 3 | xxxxxxxxh | See Table 3-7 on page 48 for bit definitions. |
| CR4 | Control Register 4 | 00000000h | See Table 3-7 on page 48 for bit definitions. |
| CCR1 | Configuration Control 1 | 00h | See Table 3-11 on page 52 for bit definitions. |
| CCR2 | Configuration Control 2 | 00h | See Table 3-11 on page 52 for bit definitions. |
| CCR3 | Configuration Control 3 | 00h | See Table 3-11 on page 52 for bit definitions. |
| CCR4 | Configuration Control 4 | 00h | See Table 3-11 on page 52 for bit definitions. |
| CCR7 | Configuration Control 7 | 00h | See Table 3-11 on page 52 for bit definitions. |

**Table 3-1. Initialized Core Register Controls (Continued)**

| Register | Register Name | Initialized Contents | Comments |
|---|---|---|---|
| SMHR | SMM Header Address | 000000h | See Table 3-11 on page 54 for bit definitions |
| SMAR | SMM Address 0 | 000000h | See Table 3-11 on page 54 for bit definitions. |
| DIR0 | Device Identification 0 | 4xh | Device ID and reads back initial CPU clock-speed setting. See Table 3-11 on page 54 for bit definitions. |
| DIR1 | Device Identification 1 | xxh | Stepping and Revision ID (RO). See Table 3-11 on page 54 for bit definitions. |
| DR7 | Debug Register 7 | 00000400h | See Table 3-13 on page 56 for bit definitions. |

**Note:** x = Undefined value

## 3.2 INSTRUCTION SET OVERVIEW

The GXLV processor instruction set can be divided into nine types of operations:

- Arithmetic
- Bit Manipulation
- Shift/Rotate
- String Manipulation
- Control Transfer
- Data Transfer
- Floating Point
- High-Level Language Support
- Operating System Support

The GXLV processor instructions operate on as few as zero operands and as many as three operands. A NOP (no operation) instruction is an example of a zero-operand instruction. Two-operand instructions allow the specification of an explicit source and destination pair as part of the instruction. These two-operand instructions can be divided into ten groups according to operand types:

- Register to Register
- Register to Memory
- Memory to Register
- Memory to Memory
- Register to I/O
- I/O to Register
- Memory to I/O
- I/O to Memory
- Immediate Data to Register
- Immediate Data to Memory

An operand can be held in the instruction itself (as in the case of an immediate operand), in one of the processor's registers or I/O ports, or in memory. An immediate operand is fetched as part of the opcode for the instruction.

Operand lengths of 8, 16, 32 or 48 bits are supported as well as 64 or 80 bits associated with floating-point instructions. Operand lengths of 8 or 32 bits are generally used when executing code written for 386- or 486-class (32-bit code) processors. Operand lengths of 8 or 16 bits are generally used when executing existing 8086 or 80286 code (16-bit code). The default length of an operand can be overridden by placing one or more instruction prefixes in front of the opcode. For example, the use of prefixes allows a 32-bit operand to be used with 16-bit code or a 16-bit operand to be used with 32-bit code.

Section 8.3 "Processor Core Instruction Set" on page 222 contains the clock count table that lists each instruction in the CPU instruction set. Included in the table are the associated opcodes, execution clock counts, and effects on the EFLAGS register.

### 3.2.1 Lock Prefix

The LOCK prefix may be placed before certain instructions that read, modify, then write back to memory. The PCI will not be granted access in the middle of locked instructions. The LOCK prefix can be used with the following instructions only when the result is a write operation to memory.

- Bit Test Instructions (BTS, BTR, BTC)
- Exchange Instructions (XADD, XCHG, CMPXCHG)
- One-Operand Arithmetic and Logical Instructions (DEC, INC, NEG, NOT)
- Two-Operand Arithmetic and Logical Instructions (ADC, ADD, AND, OR, SBB, SUB, XOR).

An invalid opcode exception is generated if the LOCK prefix is used with any other instruction or with one of the instructions above when no write operation to memory occurs (for example, when the destination is a register).

## 3.3 REGISTER SETS

The accessible registers in the processor are grouped into three sets:

1) The **Application Register Set** contains the registers frequently used by application programmers. Table 3-2 shows the General Purpose, Segment, the Instruction Pointer and the EFLAGS Registers.

2) The **System Register Set** contains the registers typically reserved for operating systems programmers: Control, System Address, Debug, Configuration, and Test Registers.

3) The **Model Specific Register (MSR) Set** is used to monitor the performance of the processor or a specific component within the processor. The Model Specific Register set has one 64-bit register called the Time Stamp Counter.

Each of these register sets are discussed in detail in the subsections that follow. Additional registers to support integrated GXLV processor subsystems are described in Section 4.1 “Integrated Functions Programming Interface” on page 97.

### 3.3.1 Application Register Set

The Application Register Set consists of the registers most often used by the applications programmer. These registers are generally accessible, although some bits in the EFLAGS register are protected.

The **General Purpose Register** contents are frequently modified by instructions and typically contain arithmetic and logical instruction operands.

In real mode, **Segment Registers** contain the base address for each segment. In protected mode, the segment registers contain segment selectors. The segment selectors provide indexing for tables (located in memory) that contain the base address for each segment, as well as other memory addressing information.

The **Instruction Pointer Register** points to the next instruction that the processor will execute. This register is automatically incremented by the processor as execution progresses.

The **EFLAGS Register** contains control bits used to reflect the status of previously executed instructions. This register also contains control bits that affect the operation of some instructions.

#### 3.3.1.1 General Purpose Registers

The General Purpose Registers are divided into four data registers, two pointer registers, and two index registers as shown in Table 3-2.

The **Data Registers** are used by the applications programmer to manipulate data structures and to hold the results of logical and arithmetic operations. Different portions of general data registers can be addressed by using different names.

An “E” prefix identifies the complete 32-bit register. An “X” suffix without the “E” prefix identifies the lower 16 bits of the register.

The lower two bytes of a data register are addressed with an “H” suffix (identifies the upper byte) or an “L” suffix (identifies the lower byte). These _L and _H portions of the data registers act as independent registers. For example, if the AH register is written to by an instruction, the AL register bits remain unchanged.

The **Pointer and Index Registers** are listed below.

| | |
|---|---|
| SI or ESI | Source Index |
| DI or EDI | Destination Index |
| SP or ESP | Stack Pointer |
| BP or EBP | Base Pointer |

These registers can be addressed as 16- or 32-bit registers, with the “E” prefix indicating 32 bits. The Pointer and Index Registers can be used as general purpose registers; however, some instructions use a fixed assignment of these registers. For example, repeated string operations always use ESI as the source pointer, EDI as the destination pointer, and ECX as a counter. The instructions that use fixed registers include multiply and divide, I/O access, string operations, stack operations, loop, variable shift and rotate, and translate instructions.

The GXLV processor implements a stack using the ESP Register. This stack is accessed during the PUSH and POP instructions, procedure calls, procedure returns, interrupts, exceptions, and interrupt/exception returns. The GXLV processor automatically adjusts the value of the ESP during operations that result from these instructions.

The EBP Register may be used to refer to data passed on the stack during procedure calls. Local data may also be placed on the stack and accessed with BP. This register provides a mechanism to access stack data in high-level languages.

## Processor Programming (Continued)

### Table 3-2. Application Register Set

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **General Purpose Registers** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | AX | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | AH | | | | | | | | AL | | | | | | | |
| EAX (Extended A Register) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | BX | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | BH | | | | | | | | BL | | | | | | | |
| EBX (Extended B Register) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | CX | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | CH | | | | | | | | CL | | | | | | | |
| ECX (Extended C Register) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | DX | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | DH | | | | | | | | DL | | | | | | | |
| EDX (Extended D Register) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | SI (Source Index) | | | | | | | | | | | | | | | |
| ESI (Extended Source Index) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | DI (Destination Index) | | | | | | | | | | | | | | | |
| EDI (Extended Destination Index) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | BP (Base Pointer) | | | | | | | | | | | | | | | |
| EBP (Extended Base Pointer) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | SP (Stack Pointer) | | | | | | | | | | | | | | | |
| ESP (Extended Stack Pointer) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **Segment (Selector) Registers** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | CS (Code Segment) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | SS (Stack Segment) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | DS (D Data Segment) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ES (E Data Segment) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | FS (F Data Segment) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | GS (G Data Segment) | | | | | | | | | | | | | | | |
| **Instruction Pointer and EFLAGS Registers** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| EIP (Extended Instruction Pointer) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ESP (Extended EFLAGS Register) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

#### 3.3.1.2 Segment Registers

The 16-bit segment registers, part of the main memory addressing mechanism, are described in Section 3.5 “Offset, Segment, and Paging Mechanisms” on page 64. The six segment registers are:

CS - Code Segment
DS - Data Segment
SS - Stack Segment
ES - Extra Segment
FS - Additional Data Segment
GS - Additional Data Segment

The segment registers are used to select segments in main memory. A segment acts as private memory for different elements of a program such as code space, data space and stack space.

There are two segment mechanisms, one for real and virtual 8086 operating modes and one for protected mode. Initialization and transition to protected mode is described in Section 3.9.4 “Initialization and Transition to Protected Mode” on page 93. The segment mechanisms are described in Section 3.5.2 “Segment Mechanisms” on page 66.

The active segment register is selected according to the rules listed in Table 3-3 and the type of instruction being currently processed. In general, the DS register selector is used for data references. Stack references use the SS register, and instruction fetches use the CS register. While some selections may be overridden, instruction fetches, stack operations, and the destination write operation of string operations cannot be overridden. Special segment-override instruction prefixes allow the use of alternate segment registers. These segment registers include the ES, FS, and GS registers.

#### 3.3.1.3 Instruction Pointer Register

The **Instruction Pointer (EIP) Register** contains the offset into the current code segment of the next instruction to be executed. The register is normally incremented by the length of the current instruction with each instruction execution unless it is implicitly modified through an interrupt, exception, or an instruction that changes the sequential execution flow (for example JMP and CALL).

Table 3-3 illustrates the code segment selection rules.

**Table 3-3. Segment Register Selection Rules**

| Type of Memory Reference | Implied (Default) Segment | Segment-Override Prefix |
|---|---|---|
| Code Fetch | CS | None |
| Destination of PUSH, PUSHF, INT, CALL, PUSHA instructions | SS | None |
| Source of POP, POPA, POPF, IRET, RET instructions | SS | None |
| Destination of STOS, MOVS, REP STOS, REP MOVS instructions | ES | None |
| Other data references with effective address using base registers of: EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP | DS<br>SS | CS, ES, FS, GS, SS<br>CS, DS, ES, FS, GS |

## Processor Programming (Continued)

### 3.3.1.4 EFLAGS Register

The EFLAGS Register contains status information and controls certain operations on the GXLV processor. The lower 16 bits of this register are referred to as the EFLAGS register that is used when executing 8086 or 80286 code. Table 3-4 gives the bit formats for the EFLAGS Register

**Table 3-4. EFLAGS Register**

| Bit | Name | Flag Type | Description |
|---|---|---|---|
| 31:22 | RSVD | -- | **Reserved:** Set to 0. |
| 21 | ID | System | **Identification Bit**: The ability to set and clear this bit indicates that the CPUID instruction is supported. The ID can be modified only if the CPUID bit in CCR4 (Index E8h[7]) is set. |
| 20:19 | RSVD | -- | **Reserved:** Set to 0. |
| 18 | AC | System | **Alignment Check Enable:** In conjunction with the AM flag (bit 18) in CR0, the AC flag determines whether or not misaligned accesses to memory cause a fault. If AC is set, alignment faults are enabled. |
| 17 | VM | System | **Virtual 8086 Mode:** If set while in protected mode, the processor switches to virtual 8086 operation handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes. The VM bit can be set by the IRET instruction (if current privilege level is 0) or by task switches at any privilege level. |
| 16 | RF | Debug | **Resume Flag:** Used in conjunction with debug register breakpoints. RF is checked at instruction boundaries before breakpoint exception processing. If set, any debug fault is ignored on the next instruction. |
| 15 | RSVD | -- | **Reserved:** Set to 0. |
| 14 | NT | System | **Nested Task:** While executing in protected mode, NT indicates that the execution of the current task is nested within another task. |
| 13:12 | IOPL | System | **I/O Privilege Level:** While executing in protected mode, IOPL indicates the maximum current privilege level (CPL) permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O permission bit map. IOPL also indicates the maximum CPL allowing alteration of the IF bit when new values are popped into the EFLAGS register. |
| 11 | OF | Arithmetic | **Overflow Flag:** Set if the operation resulted in a carry or borrow into the sign bit of the result but did not result in a carry or borrow out of the high-order bit. Also set if the operation resulted in a carry or borrow out of the high-order bit but did not result in a carry or borrow into the sign bit of the result. |
| 10 | DF | Control | **Direction Flag:** When cleared, DF causes string instructions to auto-increment (default) the appropriate index registers (ESI and/or EDI). Setting DF causes auto-decrement of the index registers to occur. |
| 9 | IF | System | **Interrupt Enable Flag:** When set, maskable interrupts (INTR input pin) are acknowledged and serviced by the CPU. |
| 8 | TF | Debug | **Trap Enable Flag:** Once set, a single-step interrupt occurs after the next instruction completes execution. TF is cleared by the single-step interrupt. |
| 7 | SF | Arithmetic | **Sign Flag:** Set equal to high-order bit of result (0 indicates positive, 1 indicates negative). |
| 6 | ZF | Arithmetic | **Zero Flag:** Set if result is zero; cleared otherwise. |
| 5 | RSVD | -- | **Reserved:** Set to 0. |
| 4 | AF | Arithmetic | **Auxiliary Carry Flag:** Set when a carry out of (addition) or borrow into (subtraction) bit position 3 of the result occurs; cleared otherwise. |
| 3 | RSVD | -- | **Reserved:** Set to 0. |
| 2 | PF | Arithmetic | **Parity Flag:** Set when the low-order 8 bits of the result contain an even number of ones; otherwise PF is cleared. |
| 1 | RSVD | | **Reserved:** Set to 1. |
| 0 | CF | Arithmetic | **Carry Flag:** Set when a carry out of (addition) or borrow into (subtraction) the most significant bit of the result occurs; cleared otherwise. |

### 3.3.2 System Register Set

The System Register Set, shown in Table 3-5, consists of registers not generally used by application programmers. These registers are typically employed by system level programmers who generate operating systems and memory management programs. Associated with the System Register Set are certain tables and segments which are listed in Table 3-5.

The **Control Registers** control certain aspects of the GXLV processor such as paging, coprocessor functions, and segment protection.

The **Configuration Registers** are used to define the GXLV CPU setup including cache management.

The **Debug Registers** provide debugging facilities for the GXLV processor and enable the use of data access breakpoints and code execution breakpoints.

The **Test Registers** provide a mechanism to test the contents of both the on-chip 16 KB cache and the Translation Lookaside Buffer (TLB).

The **Descriptor Table Register** hold descriptors that manage memory segments and tables, interrupts and task switching. The tables are defined by corresponding registers.

The two **Task State Segment Tables** defined by TSS register are used to save and load the computer state when switching tasks.

The **ID Registers** allow BIOS and other software to identify the specific CPU and stepping.

System Management Mode (SMM) control information is stored in the **SMM Registers**.

Table 3-5 lists the system register sets along with their size and function.

**Table 3-5. System Register Set**

| Group | Name | Function | Width (Bits) |
|---|---|---|---|
| Control Registers | CR0 | System Control Register | 32 |
| | CR2 | Page Fault Linear Address Register | 32 |
| | CR3 | Page Directory Base Register | 32 |
| | CR4 | Time Stamp Counter | 32 |
| Configuration Registers | CCRn | Configuration Control Registers | 8 |
| Debug Registers | DR0 | Linear Breakpoint Address 0 | 32 |
| | DR1 | Linear Breakpoint Address 1 | 32 |
| | DR2 | Linear Breakpoint Address 2 | 32 |
| | DR3 | Linear Breakpoint Address 3 | 32 |
| | DR6 | Breakpoint Status | 32 |
| | DR7 | Breakpoint Control | 32 |
| Test Registers | TR3 | Cache Test | 32 |
| | TR4 | Cache Test | 32 |
| | TR5 | Cache Test | 32 |
| | TR6 | TLB Test Control | 32 |
| | TR7 | TLB Test Data | 32 |
| Descriptor Tables | GDT | General Descriptor Table | 32 |
| | IDT | Interrupt Descriptor Table | 32 |
| | LDT | Local Descriptor Table | 16 |
| Descriptor Table Registers | GDTR | GDT Register | 32 |
| | IDTR | IDT Register | 32 |
| | LDTR | LDT Register | 16 |
| Task State Segment and Registers | TSS | Task State Segment Table | 16 |
| | TR | TSS Register Setup | 16 |
| ID Registers | DIRn | Device Identification Registers | 8 |
| SMM Registers | SMARn | SMM Address Region Registers | 8 |
| | SMHRn | SMM Header Addresses | 8 |
| Performance Registers | PCR0 | Performance Control Register | 8 |

## Processor Programming (Continued)

### 3.3.2.1 Control Registers

A map of the Control Registers (CR0, CR1, CR2, CR3, and CR4) is shown in Table 3-6 and the bit definitions are given in Table 3-7. (These registers should not be confused with the CRRn registers.) CR0 contains system control bits which configure operating modes and indicate the general state of the CPU. The lower 16 bits of CR0 are referred to as the Machine Status Word (MSW).

When operating in real mode, any program can read and write the control registers. In protected mode, however, only privilege level 0 (most-privileged) programs can read and write these registers.

### L1 Cache Controller

The GXLV processor contains an on-board 16 KB unified data/instruction write-back L1 cache. With the memory controller on-board, the L1 cache requires no external logic to maintain coherency. All DMA cycles automatically snoop the L1 cache.

The CD bit (Cache Disable, bit 30) in CR0 globally controls the operating mode of the L1 cache. LCD and LWT, Local Cache Disable and Local Write-through bits in the Translation Lookaside Buffer, control the mode on a page-by-page basis. Additionally, memory configuration control can specify certain memory regions as non-cacheable.

If the cache is disabled, no further cache line fills occur. However, data already present in the cache continues to be used. For the cache to be completely disabled, the cache must be invalidated with a WBINVD instruction after the cache has been disabled.

Write-back caching improves performance by relieving congestion on slower external buses. With four dirty bits, the cache marks dirty locations on a double-word (DWORD) basis. This further reduces the number of DWORD write operations needed during a replacement or flush operation.

The GXLV processor will cache SMM regions, reducing system management overhead to allow for hardware emulation such as VGA.

**Table 3-6. Control Registers Map**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **CR4 Register** | | | | | | | | | | | | | **Control Register 4 (R/W)** | | | | | | | | | | | | | | | | | | |
| RSVD | | | | | | | | | | | | | | | | | | | | | | | | | | | | | TSC | RSVD | |
| **CR3 Register** | | | | | | | | | | | | | **Control Register 3 (R/W)** | | | | | | | | | | | | | | | | | | |
| PDBR (Page Directory Base Register) | | | | | | | | | | | | | | | | | | | | RSVD | | | | | | | 0 | 0 | RSVD | | |
| **CR2 Register** | | | | | | | | | | | | | **Control Register 2 (R/W)** | | | | | | | | | | | | | | | | | | |
| PFLA (Page Fault Linear Address) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **CR1 Register** | | | | | | | | | | | | | **Control Register 1 (R/W)** | | | | | | | | | | | | | | | | | | |
| RSVD | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **CR0 Register** | | | | | | | | | | | | | **Control Register 0 (R/W)** | | | | | | | | | | | | | | | | | | |
| PG | CD | NW | RSVD | | | | | | | | | | AM | RSVD | WP | RSVD | | | | | | | | | | NE | RSVD | TS | EM | MP | PE |
| | | | | | | | | | | | | | | | | **Machine Status Word (MSW)** | | | | | | | | | | | | | | | |

**Table 3-7. CR4-CR0 Bit Definitions**

| Bit | Name | Description |
|---|---|---|
| **CR4 Register** | | **Control Register 4 (R/W)** |
| 31:3 | RSVD | **Reserved:** Set to 0 (always returns 0 when read). |
| 2 | TSC | **Time Stamp Counter Instruction:**<br>If = 1 RDTSC instruction enabled for CPL = 0 only; reset state.<br>If = 0 RDTSC instruction enabled for all CPL states. |
| 1:0 | RSVD | **Reserved:** Set to 0 (always returns 0 when read). |
| **CR3 Register** | | **Control Register 3 (R/W)** |
| 31:12 | PDBR | **Page Directory Base Register:** Identifies page directory base address on a 4 KB page boundary. |
| 11:0 | RSVD | **Reserved:** Set to 0. |

## Processor Programming (Continued)

### Table 3-7. CR4-CR0 Bit Definitions (Continued)

| Bit | Name | Description |
|---|---|---|
| **CR2 Register** | | **Control Register 2 (R/W)** |
| 31:0 | PFLA | **Page Fault Linear Address:** With paging enabled and after a page fault, PFLA contains the linear address of the address that caused the page fault. |
| **CR1 Register** | | **Control Register 1 (R/W)** |
| 31:0 | RSVD | **Reserved** |
| **CR0 Register** | | **Control Register 0 (R/W)** |
| 31 | PG | **Paging Enable Bit**: If PG = 1 and protected mode is enabled (PE = 1), paging is enabled. After changing the state of PG, software must execute an unconditional branch instruction (e.g., JMP, CALL) to have the change take effect. |
| 30 | CD | **Cache Disable**: If CD = 1, no further cache line fills occur. However, data already present in the cache continues to be used if the requested address hits in the cache. Writes continue to update the cache and cache invalidations due to inquiry cycles occur normally. The cache must also be invalidated with a WBINVD instruction to completely disable any cache activity. |
| 29 | NW | **Not Write-Through**: If NW = 1, the on-chip cache operates in write-back mode. In write-back mode, writes are issued to the external bus only for a cache miss, a line replacement of a modified line, execution of a locked instruction, or a line eviction as the result of a flush cycle. If NW = 0, the on-chip cache operates in write-through mode. In write-through mode, all writes (including cache hits) are issued to the external bus. This bit cannot be changed if LOCK_NW = 1 in CCR2. |
| 28:19 | RSVD | **Reserved** |
| 18 | AM | **Alignment Check Mask**: If AM = 1, the AC bit in the EFLAGS register is unmasked and allowed to enable alignment check faults. Setting AM = 0 prevents AC faults from occurring. |
| 17 | RSVD | **Reserved** |
| 16 | WP | **Write Protec**t: Protects read-only pages from supervisor write access. WP = 0 allows a read-only page to be written from privilege level 0-2. WP = 1 forces a fault on a write to a read-only page from any privilege level. |
| 15:6 | RSVD | **Reserved** |
| 5 | NE | **Numerics Exception**: NE = 1 to allow FPU exceptions to be handled by interrupt 16. NE = 0 if FPU exceptions are to be handled by external interrupts. |
| 4 | RSVD | **Reserved**: Do not attempt to modify, always 1. |
| 3 | TS | **Task Switched**: Set whenever a task switch operation is performed. Execution of a floating point instruction with TS = 1 causes a DNA fault. If MP = 1 and TS = 1, a WAIT instruction also causes a DNA fault. |
| 2 | EM | **Emulate Processor Extension**: If EM = 1, all floating point instructions cause a DNA fault 7. |
| 1 | MP | **Monitor Processor Extension**: If MP = 1 and TS = 1, a WAIT instruction causes Device Not Available (DNA) fault 7. The TS bit is set to 1 on task switches by the CPU. Floating point instructions are not affected by the state of the MP bit. The MP bit should be set to one during normal operations. |
| 0 | PE | **Protected Mode Enable**: Enables the segment based protection mechanism. If PE = 1, protected mode is enabled. If PE = 0, the CPU operates in real mode and addresses are formed as in an 8086-style CPU. Refer to Section 3.9 “Protection” on page 91. |

### Table 3-8. Effects of Various Combinations of EM, TS, and MP Bits

| CR0[3:1] | | | Instruction Type | |
|---|---|---|---|---|
| **TS** | **EM** | **MP** | **WAIT** | **ESC** |
| 0 | 0 | 0 | Execute | Execute |
| 0 | 0 | 1 | Execute | Execute |
| 1 | 0 | 0 | Execute | Fault 7 |
| 1 | 0 | 1 | Fault 7 | Fault 7 |
| 0 | 1 | 0 | Execute | Fault 7 |
| 0 | 1 | 1 | Execute | Fault 7 |
| 1 | 1 | 0 | Execute | Fault 7 |
| 1 | 1 | 1 | Fault 7 | Fault 7 |

## Processor Programming (Continued)

### 3.3.2.2 Configuration Registers

The Configuration Registers listed in Table 3-9 are CPU registers and are selected by register index numbers. The registers are accessed through I/O memory locations 22h and 23h. Registers are selected for access by writing an index number to I/O Port 22h using an OUT instruction prior to transferring data through I/O Port 23h. This operation must be atomic. The CLI instruction must be executed prior to accessing any of these registers.

Each data transfer through I/O Port 23h must be preceded by a register index selection through I/O Port 22h; otherwise, subsequent I/O Port 23h operations are directed off-chip and produce external I/O cycles.

If MAPEN, bit 4 of CCR3 (Index C3h[4]) = 0, external I/O cycles occur if the register index number is outside the range C0h-CFh, FEh, and FFh. The MAPEN bit should remain 0 during normal operation to allow system registers located at I/O Port 22h to be accessed (see Table 3-11 on page 52).

**Table 3-9. Configuration Register Summary**

| Index | Type | Name | Access Controlled By* | Default Value | Reference (Bit Formats) |
|---|---|---|---|---|---|
| C1h | R/W | CCR1 — Configuration Control 1 | SMI_LOCK | 00h | Table 3-11 on page 52 |
| C2h | R/W | CCR2 — Configuration Control 2 | -- | 00h | Table 3-11 on page 52 |
| C3h | R/W | CCR3 — Configuration Control 3 | SMI_LOCK | 00h | Table 3-11 on page 52 |
| E8h | R/W | CCR4 — Configuration Control 4 | MAPEN | 85h | Table 3-11 on page 53 |
| EBh | R/W | CCR7 — Configuration Control 7 | -- | 00h | Table 3-11 on page 53 |
| 20h | R/W | PCR — Performance Control | MAPEN | 07h | Table 3-11 on page 53 |
| B0h | R/W | SMHR0 — SMM Header Address 0 | MAPEN | xxh | Table 3-11 on page 54 |
| B1h | R/W | SMHR1 — SMM Header Address 1 | MAPEN | xxh | Table 3-11 on page 54 |
| B2h | R/W | SMHR2 — SMM Header Address 2 | MAPEN | xxh | Table 3-11 on page 54 |
| B3h | R/W | SMHR3 — SMM Header Address 3 | MAPEN | xxh | Table 3-11 on page 54 |
| B8h | R/W | GCR — Graphics Control Register | MAPEN | 00h | Table 4-1 on page 97 |
| B9h | R/W | VGACTL — VGA Control Register | -- | 00h | Table 4-37 on page 163 |
| BAh-BDh | R/W | VGAM0 — VGA Mask Register | -- | 00h | Table 4-37 on page 163 |
| CDh | R/W | SMAR0 — SMM Address 0 | SMI_LOCK | 00h | Table 3-11 on page 54 |
| CEh | R/W | SMAR1 — SMM Address 1 | SMI_LOCK | 00h | Table 3-11 on page 54 |
| CFh | R/W | SMAR2 — SMM Address 2 | SMI_LOCK | 00h | Table 3-11 on page 54 |
| FEh | RO | DIR0 — Device ID 0 | -- | 4xh | Table 3-11 on page 54 |
| FFh | RO | DIR1 — Device ID 1 | -- | xxh | Table 3-11 on page 54 |

**Note:** *MAPEN = Index C3h[4] (CCR3) and SMI_LOCK = Index C3h[0] (CCR3).

## Processor Programming (Continued)

### Table 3-10. Configuration Register Map

| Register (Index) | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| **Control Registers** | | | | | | | | |
| CCR1 (C1h) | RSVD | | | | | SMAC | USE_SMI | RSVD |
| CCR2 (C2h) | USE_SUSP | RSVD | | WT1 | SUSP_HLT | LOCK_NW | RSVD | |
| CCR3 (C3h) | LSS_34 | LSS_23 | LSS_12 | MAPEN | SUSP_SMM _EN | RSVD | NMI_EN | SMI_LOCK |
| CCR4 (E8h) | CPUID | SMI_NEST | FPU_FAST_ EN | DTE_EN | MEM_BYP | IORT2 | IORT1 | IORT0 |
| CCR7 (EBh) | RSVD | | | | | NMI | RSVD | EMMX |
| PCR (20h) | LSSER | RSVD | | | | | | |
| **SMM Base Header Address Registers** | | | | | | | | |
| SMHR0 (B0h) | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
| SMHR1 (B1h) | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A8 |
| SMHR2 (B2h) | A23 | A22 | A21 | A20 | A19 | A18 | A17 | A16 |
| SMHR3 (B3h) | A31 | A30 | A29 | A28 | A27 | A26 | A26 | A24 |
| SMAR0 (CDh) | A31 | A30 | A29 | A28 | A27 | A26 | A25 | A24 |
| SMAR1 (CEh) | A23 | A22 | A21 | A20 | A19 | A18 | A17 | A16 |
| SMAR2 (CFh) | A15 | A14 | A13 | A12 | SIZE3 | SIZE2 | SIZE1 | SIZE0 |
| **Device ID Registers** | | | | | | | | |
| DIR0 (FEh) | DID3 | DID2 | DID1 | DID0 | MULT3 | MULT2 | MULT1 | MULT0 |
| DIR1 (FFh) | SID3 | SID2 | SID1 | SID0 | RID3 | RID2 | RID1 | RID0 |
| **Graphics/VGA Related Registers** | | | | | | | | |
| GCR (B8h) | RSVD | | | | Scratchpad Size | | Base Address Code | |
| VGACTL (B9h) | RSVD | | | | | Enable SMI for VGA memory B8000h to BFFFFh | Enable SMI for VGA memory B0000h to B7FFFh | Enable SMI for VGA memory A0000h to AFFFFh |
| VGAM0 (BAh) | VGA Mask Register Bits [7:0] | | | | | | | |
| VGAM1 (BBh) | VGA Mask Register Bits [15:8] | | | | | | | |
| VGAM2 (BCh) | VGA Mask Register Bits [23:16] | | | | | | | |
| VGAM3 (BDh) | VGA Mask Register Bits [31:24] | | | | | | | |

## Processor Programming (Continued)

### Table 3-11. Configuration Registers

| Bit | Name | Description |
|---|---|---|
| **Index C1h** | | **CCR1 — Configuration Control Register 1 (R/W)** **Default Value = 00h** |
| 7:3 | RSVD | **Reserved:** Set to 0. |
| 2:1 | SMAC | **System Management Memory Access:**<br>If = 00: SMM is disabled.<br>If = 01: SMI# pin is active to enter SMM. SMINT instruction is inactive.<br>If = 10: SMM is disabled.<br>If = 11: SMINT instruction is active to enter SMM. SMI# pin is inactive.<br>**Note:** SMI_LOCK (CCR3[0]) must = 0, or the CPU must be in SMI mode, to write this bit. |
| 0 | RSVD | **Reserved:** Set to 0. |
| **Note:** Bits 1 and 2 are cleared to zero at reset. | | |
| **Index C2h** | | **CCR2 — Configuration Control Register 2 (R/W)** **Default Value = 00h** |
| 7 | USE_SUSP | **Enable Suspend Pins:**<br>If = 1: SUSP# input and SUSPA# output are enabled.<br>If = 0: SUSP# input is ignored. |
| 6 | RSVD | **Reserved:** This is a test bit that must be set to 0. |
| 5 | RSVD | **Reserved:** Set to 0. |
| 4 | WT1 | **Write-Through Region 1:**<br>If = 1: Forces all writes to the address region between 640 KB to 1 MB that hit in the on-chip cache to be issued on the external bus. |
| 3 | SUSP_HLT | **Suspend on HALT:**<br>If = 1: CPU enters Suspend mode following execution of a HALT instruction. |
| 2 | LOCK_NW | **Lock NW Bit:**<br>If = 1: Prohibits changing the state of the NW bit (CR0[29]) (refer to Table 3-7 on page 49).<br>Set to 1 after setting NW. |
| 1:0 | RSVD | **Reserved:** Set to 0. |
| **Note:** All bits are cleared to zero at reset. | | |
| **Index C3h** | | **CCR3 — Configuration Control Register 3 (R/W)** **Default Value = 00h** |
| 7 | LSS_34 | **Load/Store Serialize 3 GB to 4 GB:**<br>If = 1: Strong R/W ordering imposed in address range C0000000h to FFFFFFFFh: |
| 6 | LSS_23 | **Load/Store Serialize 2 GB to 3 GB:**<br>If = 1: Strong R/W ordering imposed in address range 80000000h to BFFFFFFFh: |
| 5 | LSS_12 | **Load/Store Serialize 1 GB to 2 GB:**<br>If = 1: Strong R/W ordering imposed in address range 40000000h to 7FFFFFFFh |
| 4 | MAPEN | **Map Enable:**<br>If = 1: All configuration registers are accessible. All accesses to I/O Port 22h are trapped.<br>If = 0: Only configuration registers Index C1h-C3h, CDh-CFh FEh, FFh (CCRn, SMAR, DIRn) are accessible. Other configuration registers (including PCR, SMHRn, GCR, VGACTL, VGAM0) are not accessible. |
| 3 | SUSP_SMM_EN | **Enable Suspend in SMM Mode:**<br>If 0 = SUSP# ignored in SMM mode.<br>If 1 = SUSP# recognized in SMM mode. |
| 2 | RSVD | **Reserved:** Set to 0. |
| 1 | NMI_EN | **NMI Enable:**<br>If = 1: NMI is enabled during SMM.<br>If = 0: NMI is not recognized during SMM.<br>**Note:** SMI_LOCK (CCR3[0]) must = 0 or the CPU must be in SMI mode to write to this bit. |
| 0 | SMI_LOCK | **SMM Register Lock:**<br>If = 1: SMM Address Region Register (SMAR[31:0]), SMAC (CCR1[2]), USE_SMI (CCR1[1]) cannot be modified unless in SMM routine. Once set, SMI_LOCK can only be cleared by asserting the RESET pin. |
| **Note:** All bits are cleared to zero at reset. | | |

## Processor Programming (Continued)

### Table 3-11. Configuration Registers (Continued)

| Bit | Name | Description |
|---|---|---|
| **Index E8h** | | **CCR4 — Configuration Control Register 4 (R/W)** **Default Value = 85h** |
| 7 | CPUID | **Enable CPUID Instruction:**<br>If = 1: The ID bit in the EFLAGS register to be modified and execution of the CPUID instruction occurs as documented in Section 8.2 “CPUID Instruction” on page 218.<br>If = 0: The ID bit can not be modified and execution of the CPUID instruction causes an invalid opcode exception. |
| 6 | SMI_NEST | **SMI Nest:**<br>If = 1: SMI interrupts can occur during SMM mode. SMM service routines can optionally set SMI_NEST high to allow higher-priority SMI interrupts while handling the current event |
| 5 | FPU_FAST_EN | **FPU Fast Mode Enable:**<br>If = 0: Disable FPU Fast Mode<br>If = 1: Enable FPU Fast Mode. |
| 4 | DTE_EN | **Directory Table Entry Cache:**<br>If = 1: Enables directory table entry to be cached.<br>Cleared to 0 at reset. |
| 3 | MEM_BYP | **Memory Read Bypassing:**<br>If = 1: Enables memory read bypassing.<br>Cleared to 0 at reset. |
| 2:0 | IORT(2:0) | **I/O Recovery Time:** Specifies the minimum number of bus clocks between I/O accesses:<br>000 = No clock delay — 100 = 16-clock delay<br>001 = 2-clock delay — 101 = 32-clock delay (default value after reset)<br>010 = 4-clock delay — 110 = 64-clock delay<br>011 = 8-clock delay — 111 = 128-clock delay |
| **Note:** MAPEN (CCR3[4]) must = 1 to read or write this register. | | |
| **Index EBh** | | **CCR7 — Configuration Control Register 7 (R/W)** **Default Value = 00h** |
| 7:3 | RSVD | **Reserved:** Set to 0. |
| 2 | NMI | **Generate NMI:**<br>If 0 = Do nothing<br>If 1 = Generate NMI<br>In order to generate multiple NMIs, this bit must be set to zero between each setting of 1. |
| 1 | RSVD | **Reserved:** Set to 0. |
| 0 | EMMX | **Extended MMX Instructions Enable:**<br>If = 1: Extended MMX instructions are enabled |
| **Index 20h** | | **PCR — Performance Control Register (R/W)** **Default Value = 07h** |
| 7 | LSSER | **Load/Store Serialize Enable (Reorder Disable):** LSSER should be set to ensure that memory mapped I/O devices operating outside of the address range 640 KB to 1 MB will operate correctly. For memory accesses above 1 GByte, refer to CCR3[7:5] (LSS_34, LSS_23, LSS_12.)<br>If = 1: All memory read and write operations will occur in execution order (load/store serializing enabled, reordering disabled).<br>If = 0: Memory reads and write can be reordered for optimum performance (load/store serializing disabled, reordering enabled).<br>Memory accesses in the address range 640 KB to 1 MB will always be issued in execution order. |
| 6 | RSVD | **Reserved:** Set to 0. |
| 5 | RSVD | **Reserved:** Set to 1. |
| 4:0 | RSVD | **Reserved:** Set to 0. |
| **Note:** MAPEN (CCR3[4]) must = 1 to read or write this register. | | |

## Processor Programming (Continued)

### Table 3-11. Configuration Registers (Continued)

| Bit | Name | Description |
|---|---|---|
| **Index B0h, B1h, B2h, B3h** | | **SMHR — SMM Header Address Register (R/W)** **Default Value = xxh** |
| Index | SMHR Bits | **SMM Header Address Bits [31:0]:** SMHR address bits [31:0] contain the physical base address for the SMM header space. For example, bits [31:24] correspond with Index B3h. Refer to Section 3.7.3 “SMM Configuration Registers” on page 85 for more information. |
| B3h | A[31:24] | |
| B2h | A[23:16] | |
| B1h | A[15:12] | |
| B0h | A[7:0] | |
| **Note:** MAPEN (CCR3[4]) must = 1 to read or write to this register. | | |
| **Index CDh, CEh, CFh** | | **SMAR — SMM Address Region/Size Register (R/W)** **Default Value = 00h** |
| Index | SMAR Bits | **SMM Address Region Bits [A31:A12]:** SMAR address bits [31:12] contain the base address for the SMM region. For example, bits [31:24] correspond with index CDh. Refer to Section 3.7.3 “SMM Configuration Registers” on page 85 for more information. |
| CDh | A[31:24] | |
| CEh | A[23:16] | |
| CFh[7:4] | A[15:12] | |
| CFh[3:0] | SIZE[3:0] | **SMM Region Size Bits [3:0]:** SIZE address bits contain the size code for the SMM region. During access the lower 4-bits of Port 23h hold SIZE[3:0]. Index CFh allows simultaneous access to SMAR address regions bits A[15:12] (see above) and size code bits SIZE[3:0].<br>0000 = SMM Disabled; 0100 = 32 KB; 1000 = 512 KB; 1100 = 8 MB<br>0001 = 4 KB; 0101 = 64 KB; 1001 = 1 MB; 1101 = 16 MB<br>0010 = 8 KB; 0110 = 128 KB; 1010 = 2 MB; 1110 = 32 MB<br>0011 = 16 KB; 0111 = 256 KB; 1011 = 4 MB; 1111 = 4 KB (same as 0001) |
| **Notes:** 1. SMI_LOCK (CCR3[0]) must = 0, or the CPU must be in SMI mode, to write these registers/bits.<br>2. Refer to Section 3.7.3 “SMM Configuration Registers” on page 85 for more information. | | |
| **Index FEh** | | **DIR0 — Device Identification Register 0 (RO)** **Default Value = 4xh** |
| 7:4 | DID[3:0] | **Device ID (Read Only):** Identifies device as GXLV processor. |
| 3:0 | MULT[3:0] | **Core Multiplier (Read Only):** Identifies the core multiplier set by the CLKMODE[2:0] pins (see signal descriptions page 31)<br>**MULT[3:0]:**<br>0000 = SYSCLK multiplied by 4 (Test mode only)<br>0001 = SYSCLK multiplied by 10<br>0010 = SYSCLK multiplied by 4<br>0011 = SYSCLK multiplied by 6<br>0100 = SYSCLK multiplied by 9<br>0101 = SYSCLK multiplied by 5<br>0110 = SYSCLK multiplied by 7<br>0111 = SYSCLK multiplied by 8<br>1xxx = Reserved |
| **Index FFh** | | **DIR1 — Device Identification Register 1 (RO)** **Default Value = xxh** |
| 7:0 | DIR1 | **Device Identification Revision (Read Only):** DIR1 indicates device revision number.<br>If DIR1 is 6xh = GXLV processor. |

## Processor Programming (Continued)

### 3.3.2.3 Debug Registers

Six debug registers (DR0-DR3, DR6 and DR7) support debugging on the GXLV processor. Memory addresses loaded in the debug registers, referred to as “breakpoints,” generate a debug exception when a memory access of the specified type occurs to the specified address. A breakpoint can be specified for a particular kind of memory access such as a read or write operation. Code and data breakpoints can also be set allowing debug exceptions to occur whenever a given data access (read or write operation) or code access (execute) occurs. The size of the debug target can be set to 1, 2, or 4 bytes. The debug registers are accessed through MOV instructions that can be executed only at privilege level 0 (real mode is always privilege level 0).

The Debug Address Registers (DR0-DR3) each contain the linear address for one of four possible breakpoints. Each breakpoint is further specified by bits in the Debug Control Register (DR7). For each breakpoint address in DR0-DR3, there are corresponding fields L, R/W, and LEN in DR7 that specify the type of memory access associated with the breakpoint. DR6 is read only and reports the results of the break.

The R/W field can be used to specify instruction execution as well as data access breakpoints. Instruction execution breakpoints are always acted upon before execution of the instruction that matches the breakpoint. The Debug Registers are mapped in Table 3-12, and the bit definitions are given in Table 3-13 on page 56.

**Table 3-12. Debug Registers**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **DR7 Register** | | | | | | | | | | | | **Debug Control Register 7 (R/W)** | | | | | | | | | | | | | | | | | | | |
| LEN3 | | R/W3 | | LEN2 | | R/W2 | | LEN1 | | R/W1 | | LEN0 | | R/W0 | | 0 | 0 | GD | 0 | 0 | 1 | 0 | 0 | G3 | L3 | G2 | L2 | G1 | L1 | G0 | L0 |
| **DR6 Register** | | | | | | | | | | | | **Debug Status Register 6 (R/O)** | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | BT | BS | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | B3 | B2 | B1 | B0 |
| **DR3 Register** | | | | | | | | | | | | **Debug Address Register 3 (R/W)** | | | | | | | | | | | | | | | | | | | |
| Breakpoint 3 Linear Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **DR2 Register** | | | | | | | | | | | | **Debug Address Register 2 (R/W)** | | | | | | | | | | | | | | | | | | | |
| Breakpoint 2 Linear Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **DR1 Register** | | | | | | | | | | | | **Debug Address Register 1 (R/W)** | | | | | | | | | | | | | | | | | | | |
| Breakpoint 1 Linear Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **DR0 Register** | | | | | | | | | | | | **Debug Address Register 0 (R/W)** | | | | | | | | | | | | | | | | | | | |
| Breakpoint 0 Linear Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

**Note:** All bits marked as 0 or 1 are reserved and should not be modified.

## Processor Programming (Continued)

The Debug Status Register (DR6) reflects conditions that were in effect at the time the debug exception occurred. The contents of the DR6 register are not automatically cleared by the processor after a debug exception occurs, and therefore should be cleared by software at the appropriate time. Code execution breakpoints may also be generated by placing the breakpoint instruction (INT3) at the location where control is to be regained. The single-step feature may be enabled by setting the TF flag (bit 8) in the EFLAGS register. This causes the processor to perform a debug exception after the execution of every instruction.

**Table 3-13. DR7 and DR6 Bit Definitions**

| Field(s) | Number of Bits | Description |
|---|---|---|
| **DR7 Register** | | **Debug Control Register (R/W)** |
| R/Wn | 2 | Applies to the DRn breakpoint address register:<br>00 = Break on instruction execution only<br>01 = Break on data write operations only<br>10 = Not used<br>11 = Break on data reads or write operations |
| LENn | 2 | Applies to the DRn breakpoint address register:<br>00 = One-byte length<br>01 = Two-byte length<br>10 = Not used<br>11 = Four-byte length |
| Gn | 1 | If = 1: Breakpoint in DRn is globally enabled for all tasks and is not cleared by the processor as the result of a task switch. |
| Ln | 1 | If = 1: Breakpoint in DRn is locally enabled for the current task and is cleared by the processor as the result of a task switch. |
| GD | 1 | Global disable of debug register access. GD bit is cleared whenever a debug exception occurs. |
| **DR6 Register** | | **Debug Status Register (RO)** |
| Bn | 1 | Bn is set by the processor if the conditions described by DRn, R/Wn, and LENn occurred when the debug exception occurred, even if the breakpoint is not enabled via the Gn or Ln bits. |
| BT | 1 | BT is set by the processor before entering the debug handler if a task switch has occurred to a task with the T bit in the TSS set. |
| BS | 1 | BS is set by the processor if the debug exception was triggered by the single-step execution mode (TF flag, bit 8, in EFLAGS set). |
| **Note:** n = 0, 1, 2, and 3 | | |

#### 3.3.2.4 TLB Test Registers

Two test registers are used in testing the processor's Translation Lookaside Buffer (TLB), TR6 and TR7. Table 3-14 is a register map for the TLB Test Registers with their bit definitions given in Table 3-15 on page 58. The test registers are accessed through MOV instructions that can be executed only at privilege level 0 (real mode is always privilege level 0).

The CPU TLB is a 32-entry, four-way set associative memory. Each TLB entry consists of a 24-bit tag and 20-bit data. The 24-bit tag represents the high-order 20 bits of the linear address, a valid bit, and three attribute bits. The 20-bit data portion represents the upper 20 bits of the physical address that corresponds to the linear address.

The TLB Test Control Register (TR6) contains a command bit, the upper 20 bits of a linear address, a valid bit and the attribute bits used in the test operation. The contents of TR6 are used to create the 24-bit TLB tag during both write and read (TLB lookup) test operations. The command bit defines whether the test operation is a read or a write.

The TLB Test Data Register (TR7) contains the upper 20 bits of the physical address (TLB data field), three LRU bits, two replacement (REP) bits, and a control bit (PL). During TLB write operations, the physical address in TR7 is written into the TLB entry selected by the contents of TR6. During TLB lookup operations, the TLB data selected by the contents of TR6 is loaded into TR7. Table 3-15 lists the bit definitions for TR7 and TR6.

**Table 3-14. TLB Test Registers**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **TR7 Register** | | | | | | | | | | | | **TLB Test Data Register (R/W)** | | | | | | | | | | | | | | | | | | | |
| Physical Address | | | | | | | | | | | | | | | | | | | | 0 | 0 | TLB LRU | | | 0 | 0 | P L | REP | | 0 | 0 |
| **TR6 Register** | | | | | | | | | | | | **TLB Test Control Register (R/W)** | | | | | | | | | | | | | | | | | | | |
| Linear Address | | | | | | | | | | | | | | | | | | | | V | D | D # | U | U # | R | R # | 0 | 0 | 0 | 0 | C |

## Processor Programming (Continued)

### Table 3-15. TR7-TR6 Bit Definitions

| Bit | Name | Description |
|---|---|---|
| **TR7 Register** | | **TLB Test Data Register (R/W)** |
| 31:12 | Physical Address | **Physical Address:**<br>TLB lookup: Data field from the TLB.<br>TLB write: Data field written into the TLB. |
| 11:10 | RSVD | **Reserved:** Set to 0. |
| 9:7 | TLB LRU | **LRU Bits:**<br>TLB lookup: LRU bits associated with the TLB entry before the TLB lookup.<br>TLB write: Ignored. |
| 4 | PL | **PL Bit:**<br>TLB lookup: If PL = 1, read hit occurred. If PL = 0, read miss occurred.<br>TLB write: If PL = 1, REP field is used to select the set. If PL = 0, the pseudo-LRU replacement algorithm is used to select the set. |
| 3:2 | REP | **Set Selection:**<br>TLB lookup: If PL = 1, this field indicates the set in which the tag was found. If PL = 0, undefined data.<br>TLB write: If PL = 1, this field selects one of the four sets for replacement. If PL = 0, ignored. |
| 1:0 | RSVD | **Reserved:** Set to 0. |
| **TR6 Register** | | **TLB Test Control Register (R/W)** |
| 31:12 | Linear Address | **Linear Address:**<br>TLB lookup: The TLB is interrogated per this address. If one and only one match occurs in the TLB, the rest of the fields in TR6 and TR7 are updated per the matching TLB entry.<br>TLB write: A TLB entry is allocated to this linear address. |
| 11 | V | **Valid Bit:**<br>TLB write: If V = 1, the TLB entry contains valid data. If V = 0, target entry is invalidated. |
| 10:9<br>8:7<br>6:5 | D, D#<br>U, U#<br>R, R# | **Dirty Attribute Bit and its Complement (D, D#):**<br>**User/Supervisor Attribute Bit and its Complement (U, U#):**<br>**Read/Write Attribute Bit and its Complement (R, R#):**<br>**Effect on TLB Lookup** / **Effect on TLB Write**<br>00 = Do not match / Undefined<br>01 = Match if D, U, or R bit is a 0 / Clear the bit<br>10 = Match if D, U, or R bit is a 1 / Set the bit<br>11 = Match if D, U, or R bit is either a 1 or 0 / Undefined |
| 4:1 | RSVD | **Reserved:** Set to 0. |
| 0 | C | **Command Bit:**<br>If C = 1: TLB lookup.<br>If C = 0: TLB write. |

## Processor Programming (Continued)

### 3.3.2.5 Cache Test Registers

Three test registers are used in testing the processor's on-chip cache, TR3-TR5. Table 3-16 is a register map for the Cache Test Registers with their bit definitions given in Table 3-17 on page 60. The test registers are accessed through MOV instructions that can be executed only at privilege level 0 (real mode is always privilege level 0).

The processor's 16 KB on-chip cache is a four-way set associative memory that is configured as write-back cache. Each cache set contains 256 entries. Each entry consists of a 20-bit tag address, a 16-byte data field, a valid bit, and four dirty bits.

The 20-bit tag represents the high-order 20 bits of the physical address. The 16-byte data represents the 16 bytes of data currently in memory at the physical address represented by the tag. The valid bit indicates whether the data bytes in the cache actually contain valid data. The four dirty bits indicate if the data bytes in the cache have been modified internally without updating external memory (write-back configuration). Each dirty bit indicates the status for one DWORD (4 bytes) within the 16-byte data field.

For each line in the cache, there are three LRU bits that indicate which of the four sets was most recently accessed. A line is selected using bits [11:4] of the physical address. Using a 16-byte cache fill buffer and a 16-byte cache flush buffer, cache reads and writes may be performed.

Figure 3-1 illustrates the internal cache architecture.

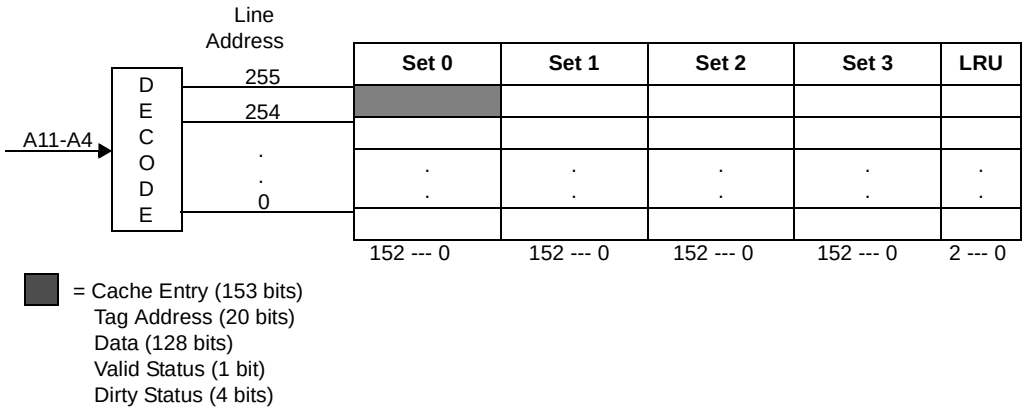



**Figure 3-1. Cache Architecture**

**Table 3-16. Cache Test Registers**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **TR5 Register (R/W)** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RSVD | | | | | | | | | | | | | | | | | | | | Line Selection | | | | | | | | Set/ DWORD | | Control Bits | |
| **TR4 Register - Cache (R/W)** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Cache Tag Address | | | | | | | | | | | | | | | | | | | | 0 | Valid | Cache LRU Bits | | | Dirty Bits | | | | 0 | 0 | 0 |
| **TR3 Register - Cache (R/W)** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Cache Data | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

## Processor Programming (Continued)

### Table 3-17. TR5-TR3 Bit Definitions

| Bit | Name | Description |
|---|---|---|
| **TR5 Register (R/W)** | | |
| 11:4 | Line Selection | **Line Selection:**<br>Physical address bits [11:4] used to select one of 256 lines. |
| 3:2 | Set/DWORD Selection | **Set/DWORD Selection:**<br>Cache read: Selects which of the four sets in the cache is used as the source for data transferred to the cache flush buffer.<br>Cache write: Selects which of the four sets in the cache is used as the destination for data transferred from the cache fill buffer.<br>Flush buffer read: Selects which of the four DWORDs in the flush buffer is used during a TR3 read.<br>Fill buffer write: Selects which of the four DWORDs in the fill buffer is written during a TR3 write. |
| 1:0 | Control Bits | **Control Bits:**<br>00 = Flush read or fill buffer write.<br>01 = Cache write.<br>10 = Cache read.<br>11 = Cache flush. |
| **TR4 Register (R/W)** | | |
| 31:12 | Upper Tag Address | **Upper Tag Address:**<br>Cache read: Upper 20 bits of tag address of the selected entry.<br>Cache write: Data written into the upper 20 bits of the tag address of the selected entry. |
| 10 | Valid Bit | **Valid Bit:**<br>Cache read: Valid bit for the selected entry.<br>Cache write: Data written into the valid bit for the selected entry. |
| 9:7 | LRU Bits | **LRU Bits:**<br>Cache read: The LRU bits for the selected line when scratchpad is disabled.<br>xx1 = Set 0 or Set 1 most recently accessed.<br>xx0 = Set 2 or Set 3 most recently accessed.<br>x1x = Most recent access to Set 0 or Set 1 was to Set 0.<br>x0x = Most recent access to Set 0 or Set 1 was to Set 1.<br>1xx = Most recent access to Set 2 or Set 3 was to Set 2.<br>0xx = Most recent access to Set 2 or Set 3 was to Set 3.<br>Cache write: Ignored. |
| 6:3 | Dirty Bits | **Dirty Bits:**<br>Cache read: The dirty bits for the selected entry (one bit per DWORD).<br>Cache write: Data written into the dirty bits for the selected entry. |
| 2:0 | RSVD | **Reserved:** Set to 0. |
| **TR3 Register (R/W)** | | |
| 31:0 | Cache Data | **Cache Data:**<br>Flush buffer read: Data accessed from the cache flush buffer.<br>Fill buffer write: Data to be written into the cache fill buffer. |

## Processor Programming (Continued)

There are five types of test operations that can be executed:

- Flush buffer read
- Fill buffer write
- Cache write
- Cache read
- Cache flush

These operations are described in detail in Table 3-18. To fill a cache line with data, the fill buffer must be written four times. Once the fill buffer holds a complete cache line of data (16 bytes), a cache write operation transfers the data from the fill buffer to the cache.

To read the contents of a cache line, a cache read operation transfers the data in the selected cache line to the flush buffer. Once the flush buffer is loaded, access the contents of the flush buffer with four flush buffer read operations.

**Table 3-18. Cache Test Operations**

| Test Operation | Code Sequence | Action Taken |
|---|---|---|
| Flush Buffer Read | MOV TR5, 0h<br>MOV dest,TR3 | Set DWORD = 0, control = 00 = flush buffer read.<br>Flush buffer (31:0) --> dest. |
| | MOV TR5, 4h<br>MOV dest,TR3 | Set DWORD = 1, control = 00 = flush buffer read.<br>Flush buffer (63:32) --> dest. |
| | MOV TR5, 8h<br>MOV dest,TR3 | Set DWORD = 2, control = 00 = flush buffer read.<br>Flush buffer (95:64) --> dest. |
| | MOV TR5, Ch<br>MOV dest,TR3 | Set DWORD = 3, control = 00 = flush buffer read.<br>Flush buffer (127:96) --> dest. |
| Fill Buffer Write | MOV TR5, 0h<br>MOV TR3, cache_data | Set DWORD = 0, control = 00 = fill buffer write.<br>Cache_data --> fill buffer (31:0). |
| | MOV TR5, 4h<br>MOV TR3, cache_data | Set DWORD = 1, control = 00 = fill buffer write.<br>Cache_data --> fill buffer (63:32). |
| | MOV TR5, 8h<br>MOV TR3, cache_data | Set DWORD = 2, control = 00 = fill buffer write.<br>Cache_data --> fill buffer (95:64). |
| | MOV TR5, Ch<br>MOV TR3, cache_data | Set DWORD = 3, control = 00 = fill buffer write.<br>Cache_data --> fill buffer (127:96). |
| Cache Write | MOV TR4, cache_tag | Cache_tag --> tag address, valid and dirty bits. |
| | MOV TR5, line+set+control=01 | Fill buffer (127:0) --> cache line (127:0). |
| Cache Read | MOV TR5, line+set+control=10<br>MOV dest, TR4 | Cache line (127:0) --> flush buffer (127:0).<br>Cache line tag address, valid/LRU/dirty bits --> dest. |
| Cache Flush | MOV TR5, 3h | Control = 11 = cache flush, all cache valid bits = 0. |

## Processor Programming (Continued)

### 3.3.3 Model Specific Register Set

The Model Specific Register (MSR) Set is used to monitor the performance of the processor or a specific component within the processor.

A MSR can be read using the RDMSR instruction, opcode 0F32h. During a MSR read, the contents of the particular MSR, specified by the ECX register, is loaded into the EDX:EAX registers.

A MSR can be written using the WRMSR instruction, opcode 0F30h. During a MSR write, the contents of EDX:EAX are loaded into the MSR specified in the ECX register.

The RDMSR and WRMSR instructions are privileged instructions.

The GXLV processor contains one 64-bit model specific register (MSR10) the Time Stamp Counter (TSC).

### 3.3.4 Time Stamp Counter

The TSC, (MSR[10]), is a 64-bit counter that counts the internal CPU clock cycles since the last reset. The TSC uses a continuous CPU core clock and will continue to count clock cycles unless the processor is in Suspend.

The TSC is read using a RDMSR instruction, opcode 0F32h, with the ECX register set to 10h. During a TSC read, the contents of the TSC is loaded into the EDX:EAX registers.

The TSC is written to using a WRMSR instruction, opcode 0F30h with the ECX register set to 10h. During a TSC write, the contents of EDX:EAX are loaded into the TSC.

The RDMSR and WRMSR instructions are privileged instructions.

In addition, the TSC can be read using the RDTSC instruction, opcode 0F31h. The RDTSC instruction loads the contents of the TSC into EDX:EAX. The use of the RDTSC instruction is restricted by the TSC flag (bit 2) in the CR4 register (refer to Tables 3-6 and 3-7 on page 48 for CR4 register information). When the TSC bit = 0, the RDTSC instruction can be executed at any privilege level. When the TSC bit = 1, the RDTSC instruction can only be executed at privilege level 0.

## 3.4 ADDRESS SPACES

The GXLV processor can directly address either memory or I/O space. Figure 3-2 illustrates the range of addresses available for memory address space and I/O address space. For the CPU, the addresses for physical memory range between 0000 0000h and FFFFFFFFh (4 GB). The accessible I/O address space ranges between 00000000h and 0000FFFFh (64 KB). The CPU does not use coprocessor communication space in upper I/O space between 800000F8h and 800000FFh as do the 386-style CPUs. The I/O locations 22h and 23h are used for GXLV processor configuration register access.

### 3.4.1 I/O Address Space

The CPU I/O address space is accessed using IN and OUT instructions to addresses referred to as "ports." The accessible I/O address space is 64 KB and can be accessed as 8-, 16- or 32-bit ports.

The GXLV processor configuration registers reside within the I/O address space at port addresses 22h and 23h and are accessed using the standard IN and OUT instructions.

The configuration registers are modified by writing the index of the configuration register to Port 22h, and then transferring the data through Port 23h. Accesses to the on-chip configuration registers do not generate external I/O cycles. However, each operation on Port 23h must be preceded by a write to Port 22h with a valid index value. Otherwise, subsequent Port 23h operations will communicate through the I/O port to produce external I/O cycles without modifying the on-chip configuration registers. Write operations to port 22h outside of the CPU index range (C0h-CFh and FEh-FFh) result in external I/O cycles and do not affect the on-chip configuration registers. Reading Port 22h generates external I/O cycles.

I/O accesses to port address range 3B0h through 3DFh can be trapped to SMI by the CPU if this option is enabled in the BC_XMAP_1 register (see SMIB, SMIC, and SMID bits in Table 4-9 on page 104). Figure 3-2 illustrates the I/O address space.

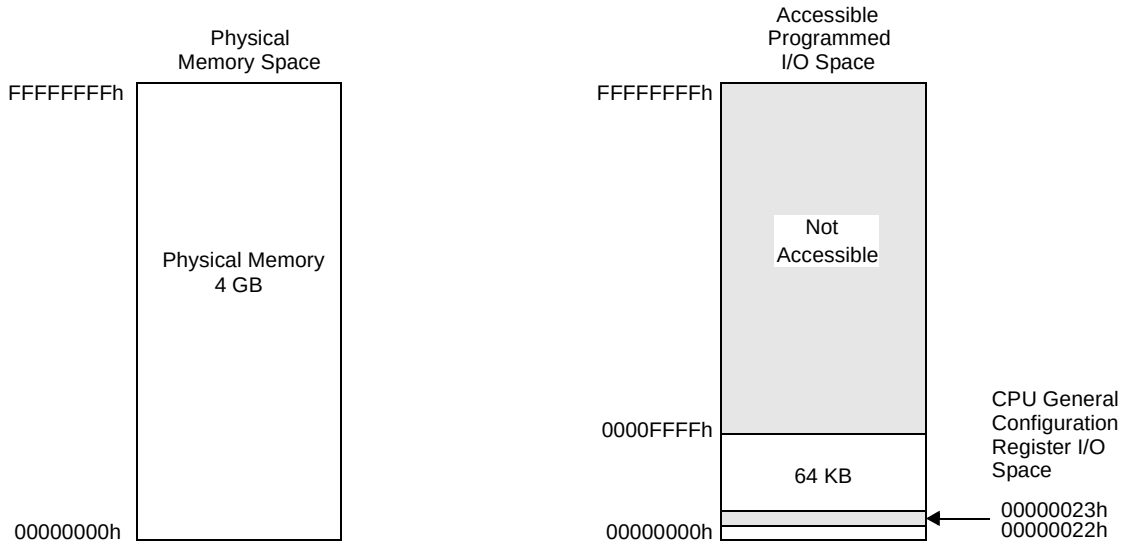



**Figure 3-2. Memory and I/O Address Spaces**

### 3.4.2 Memory Address Space

The processor directly addresses up to 4 GB of physical memory even though the memory controller addresses only 256 MB of DRAM. Memory address space is accessed as BYTES, WORDS (16 bits) or DWORDs (32 bits). WORDS and DWORDs are stored in consecutive memory bytes with the low-order byte located in the lowest address. The physical address of a WORD or DWORD is the byte address of the low-order byte.

The processor allows memory to be addressed using nine different addressing modes. These addressing modes are used to calculate an offset address, often referred to as an effective address. Depending on the operating mode of the CPU, the offset is then combined, using memory management mechanisms, into a physical address that is applied to the physical memory devices.

Memory management mechanisms consist of segmentation and paging. Segmentation allows each program to use several independent, protected address spaces. Paging translates a logical address into a physical address using translation lookup tables. Virtual memory is often implemented using paging. Either or both of these mechanisms can be used for management of the GXLV processor memory address space.

## 3.5 OFFSET, SEGMENT, AND PAGING MECHANISMS

The mapping of address space into a sequence of memory locations (often cached) is performed by the offset, segment, and paging mechanisms.

In general, the offset, segment and paging mechanisms work in tandem as shown below:

instruction offset ⇒ *offset mechanism* ⇒ offset address
offset address ⇒ *segment mechanism* ⇒ linear address
linear address ⇒ *paging mechanism* ⇒ physical page.

As will be explained, the actual operations depend on several factors such as the current operating mode and if paging is enabled.

**Note:** The paging mechanism uses part of the linear address as an offset on the physical page.

### 3.5.1 Offset Mechanism

In all operating modes, the offset mechanism computes an offset (effective) address by adding together up to three values: a base, an index and a displacement. The base, if present, is the value in one of eight general registers at the time of the execution of the instruction. The index, like the base, is a value that is contained in one of the general registers (except the ESP register) when the instruction is executed. The index differs from the base in that the index is first multiplied by a scale factor of 1, 2, 4 or 8 before the summation is made. The third component added to the memory address calculation is the displacement that is a value supplied as part of the instruction. Figure 3-3 illustrates the calculation of the offset address.

Nine valid combinations of the base, index, scale factor and displacement can be used with the CPU instruction set. These combinations are listed in Table 3-19. The base and index both refer to contents of a register as indicated by [Base] and [Index].

In real mode operation, the CPU only addresses the lowest 1 MB of memory and the offset contains 16-bits. In protected mode the offset contains 32 bits. Initialization and transition to protected mode is described in Section 3.9.4 “Initialization and Transition to Protected Mode” on page 93.

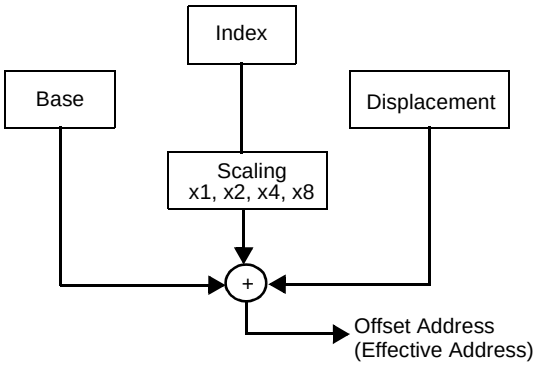



**Figure 3-3. Offset Address Calculation**

## Processor Programming (Continued)

**Table 3-19. Memory Addressing Modes**

| Addressing Mode | Base | Index | Scale Factor (SF) | Displacement (DP) | Offset Address (OA) Calculation |
|---|---|---|---|---|---|
| Direct | | | | x | OA = DP |
| Register Indirect | x | | | | OA = [BASE] |
| Based | x | | | x | OA = [BASE] + DP |
| Index | | x | | x | OA = [INDEX] + DP |
| Scaled Index | | x | x | x | OA = ([INDEX] * SF) + DP |
| Based Index | x | x | | | OA = [BASE] + [INDEX] |
| Based Scaled Index | x | x | x | | OA = [BASE] + ([INDEX] * SF) |
| Based Index with Displacement | x | x | | x | OA = [BASE] + [INDEX] + DP |
| Based Scaled Index with Displacement | x | x | x | x | OA = [BASE] + ([INDEX] * SF) + DP |

## Processor Programming (Continued)

### 3.5.2 Segment Mechanisms

Memory is divided into contiguous regions called “segments.” The segments allow the partitioning of individual elements of a program. Each segment provides a zero address-based private memory for such elements as code, data, and stack space.

The segment mechanisms select a segment in memory. Memory is divided into an arbitrary number of segments, each containing usually much less than the $2^{32}$ byte (4 GB) maximum.

There are two segment mechanisms, one for real and virtual 8086 operating modes, and one for protected mode.

#### 3.5.2.1 Real Mode Segment Mechanism

In real mode operation, the CPU addresses only the lowest 1 MB of memory. In this mode a selector located in one of the segment registers is used to locate a segment.

To calculate a physical memory address, the 16-bit segment base address located in the selected segment register is multiplied by 16 and then a 16-bit offset address is added. The resulting 20-bit address is then extended with twelve zeros in the upper address bits to create a 32-bit physical address.

The value of the selector (the INDEX field) is multiplied by 16 to produce a base address (see Figure 3-4). The base address is summed with the instruction offset value to produce a physical address.

#### 3.5.2.2 Virtual 8086 Mode Segment Mechanism

In virtual 8086 mode the operation is performed as in real mode except that a paging mechanism is added. When paging is enabled, the paging mechanism translates the linear address into a physical address using cached look-up tables (refer to Section 3.5.4 “Paging Mechanism” on page 77).

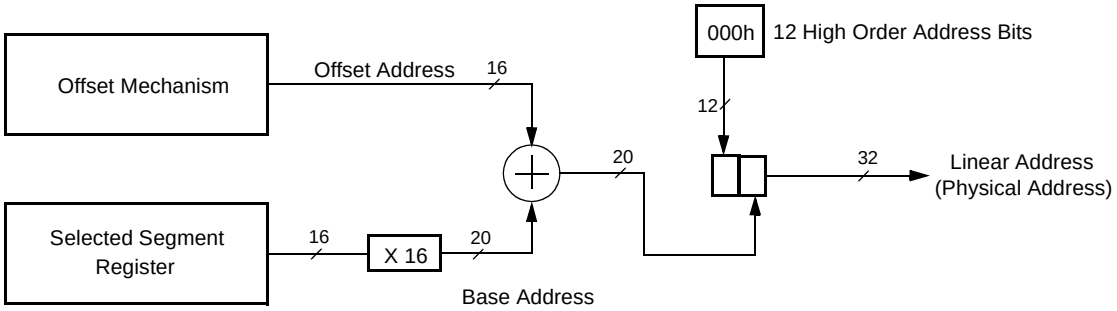



**Figure 3-4. Real Mode Address Calculation**

#### 3.5.2.3 Segment Mechanism in Protected Mode

The segment mechanism in protected mode is more complex. Basically as in real and virtual 8086 modes the offset address is added to the segment base address to produce a linear address (Figure 3-5). However, the calculation of the segment base address is based on the contents of descriptor tables.

If paging is enabled the linear address is further processed by the paging mechanism.

A more detailed look at the segment mechanisms for real and virtual 8086 modes and protected modes is illustrated in Figure 3-6 on page 68. In protected mode, the segment selector is cached. This is illustrated in Figure 3-7 on page 69.

#### 3.5.2.4 Segment Selectors

The segment registers are used to store segment selectors. In protected mode, the segment selectors are divided in to three fields: the RPL, TI and INDEX fields as shown in Figure 3-6 on page 68.

The segments are assigned permission levels to prevent application program errors from disrupting operating programs. The Requested Privilege Level (RPL) determines the effective privilege level of an instruction. RPL = 0 indicates the most privileged level, and RPL = 3 indicates the least privileged level. Refer to Section 3.9 “Protection” on page 91.

Descriptor tables hold descriptors that allow management of segments and tables in address space while in protected mode. The Table Indicator Bit (TI) in the selector selects either the General Descriptor Table (GDT) or one Local Descriptor Table (LDT). If TI = 0, GDT is selected; if TI =1, LDT is selected. The 13-bit INDEX field in the segment selector is used to index a GDT or LDT.

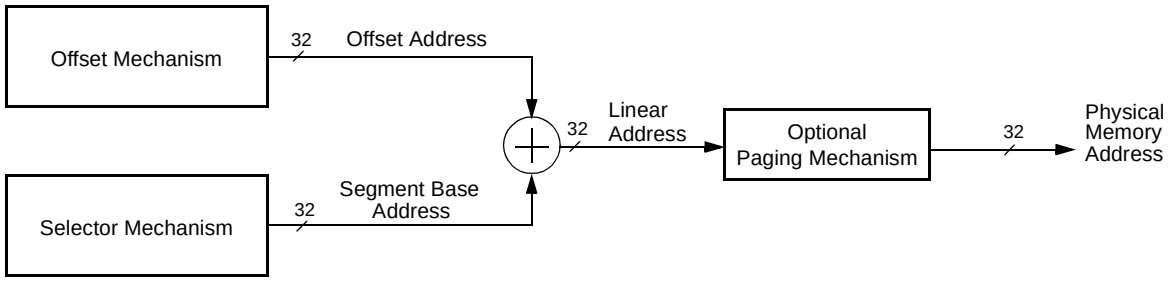



**Figure 3-5. Protected Mode Address Calculation**

## Processor Programming (Continued)

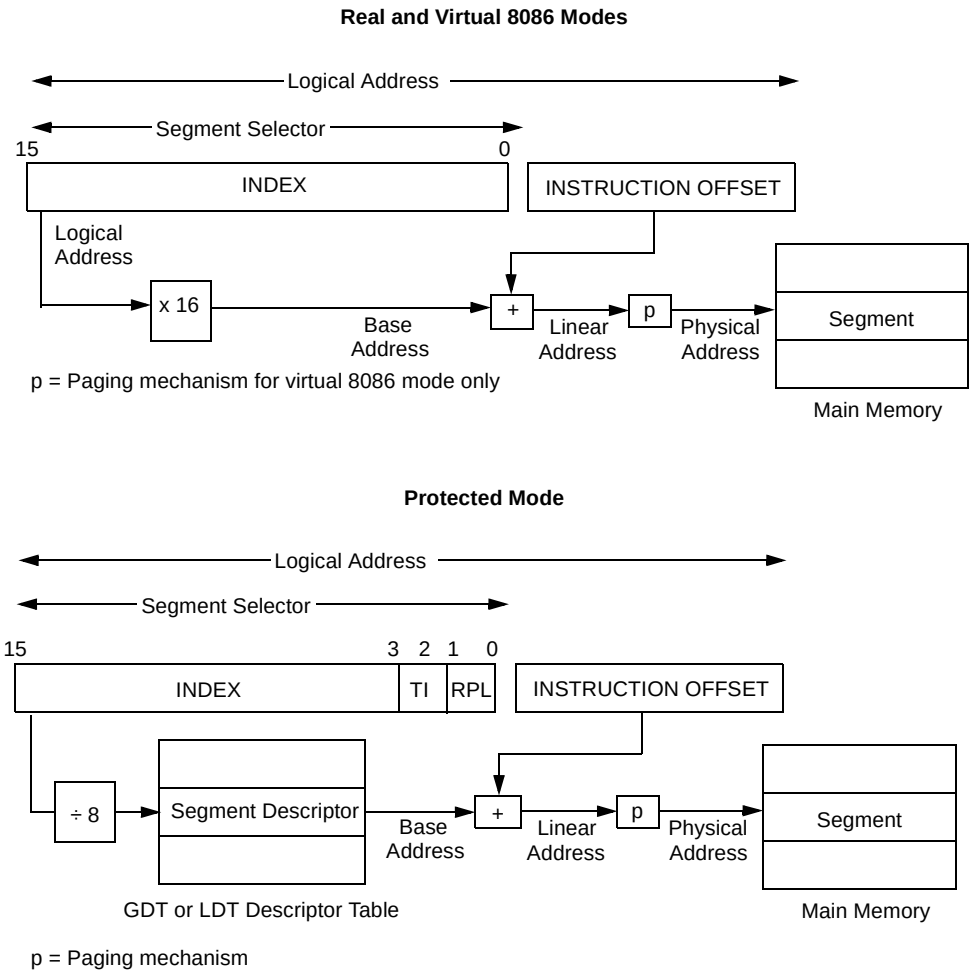



**Figure 3-6. Selector Mechanisms**

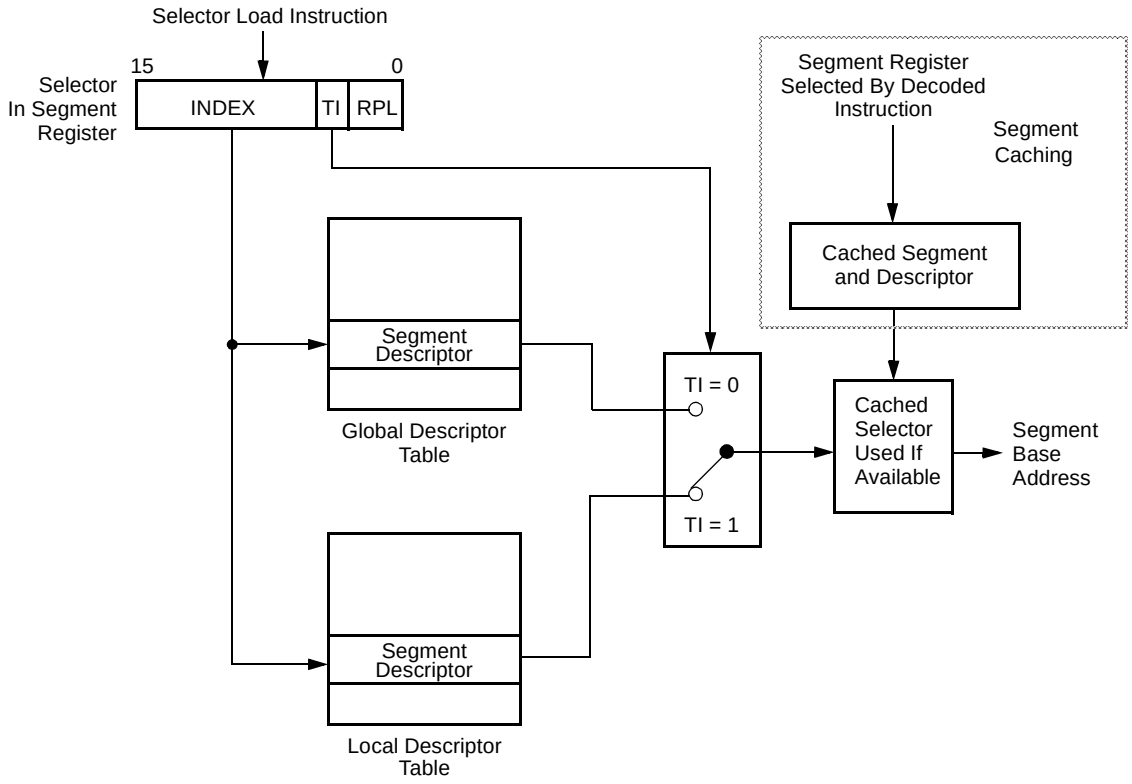



**Figure 3-7. Selector Mechanism Caching**

## Processor Programming (Continued)

### 3.5.3 Descriptors

#### 3.5.3.1 Global and Local Descriptor Table Registers

The GDT and LDT descriptor tables are defined by the Global Descriptor Table Register (GDTR) and the Local Descriptor Table Register (LDTR) respectively. Some texts refer to these registers as GDT and LDT descriptors.

The following instructions are used in conjunction with the GDT and LDT registers:

- LGDT - Load memory to GDTR
- LLDT - Load memory to LDTR
- SGDT - Store GDTR to memory
- SLDT - Store LDTR to memory

The GDTR is set up in real mode using the LGDT instruction. This is possible as the LGDT instruction is one of two instructions that directly load a linear address (instead of a segment relative address) in protected mode. (The other instruction is the Load Interrupt Descriptor Table [LIDT]).

As shown in Table 3-20, the GDTR contains a BASE field and a LIMIT field that define the GDTs. The Interrupt Descriptor Table Register (IDTR) is described in Section 3.5.3.3 “Task, Gate, Interrupt, and Application and System Descriptors” on page 71.

Also shown in Table 3-20, the LDTR is only two bytes wide as it contains only a SELECTOR field. The contents of the SELECTOR field point to a descriptor in the GDT.

#### 3.5.3.2 Segment Descriptors

There are several types of descriptors. A segment descriptor defines the base address, limit, and attributes of a memory segment.

The GDT or LDT can hold several types of descriptors. In particular, the segment descriptors are stored in either of two registers, the GDT or the LDT. Either of these tables can store as many as 8,192 ($2^{13}$) 8-byte selectors taking as much as 64 KB of memory.

The first descriptor in the GDT (location 0) is not used by the CPU and is referred to as the “null descriptor.”

**Types of Segment Descriptors**

The type of memory segments are defined by corresponding types of segment descriptors:

- Code Segment Descriptors
- Data Segment Descriptors
- Stack Segment Descriptors
- LDT Segment Descriptors

**Table 3-20. GDT, LDT and IDT Registers**

| 47 … 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **GDT Register** **Global Descriptor Table Register** | | | | | | | | | | | | | | | | |
| BASE | LIMIT | | | | | | | | | | | | | | | |
| **IDT Register** **Interrupt Descriptor Table Register** | | | | | | | | | | | | | | | | |
| BASE | LIMIT | | | | | | | | | | | | | | | |
| | **LDT Register** **Local Descriptor Table Register** | | | | | | | | | | | | | | | |
| | SELECTOR | | | | | | | | | | | | | | | |

#### 3.5.3.3 Task, Gate, Interrupt, and Application and System Descriptors

Besides segment descriptors there are descriptors used in task switching, switching between tasks with different priority and those used to control interrupt functions:

- Interrupt Descriptors
- Application and System Segment Descriptors
- Gate Descriptors
- Task State Segment Descriptors

All descriptors have some things in common. They are all eight bytes in length and have three fields (BASE, LIMIT, and TYPE). The BASE field defines the starting location for the table or segment. The LIMIT field defines the size and the TYPE field depends on the type of descriptor. One of the main functions of the TYPE field is to define the access rights to the associated segment or table.

**Interrupt Descriptors**

The Interrupt Descriptor Table is an array of 256 8-byte (4-byte for real mode) interrupt descriptors, each of which is used to point to an interrupt service routine. Every interrupt that may occur in the system must have an associated entry in the IDT. The contents of the IDTR are completely visible to the programmer through the use of the SIDT instruction.

The IDT is defined by the Interrupt Descriptor Table Register (IDTR). Some texts refer to this register as an IDT descriptor.

The following instructions are used in conjunction with the IDTR:

- LIDT - Load memory to IDTR
- SIDT - Store IDTR to memory

The IDTR is set up in real mode using the LIDT instruction. This is possible as the LIDT instruction is only one of two instructions that directly load a linear address (instead of a segment relative address) in protected mode (the other instructions is LGDT).

As previously shown in Table 3-20 on page 70, the IDTR contains a BASE ADDRESS field and a LIMIT field that define the IDT tables.

**Application and System Segment Descriptors**

The bit structure and bit definitions for segment descriptors are shown in Table 3-21 and Table 3-22 on page 72, respectively. The explanation of the TYPE field is shown in Table 3-23 on page 73.

**Table 3-21. Application and System Segment Descriptors**

| 31 | 31 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Memory Offset +4** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| BASE[31:24] | | | | | | | | G | D | 0 | AVL | LIMIT[19:16] | | | | P | DPL | | S | TYPE | | | | BASE[23:16] | | | | | | | |
| **Memory Offset +0** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| BASE[15:0] | | | | | | | | | | | | | | | | LIMIT[15:0] | | | | | | | | | | | | | | | |

## Processor Programming (Continued)

### Table 3-22. Descriptors Bit Definitions

| Bit | Memory Offset | Name | Description |
|---|---|---|---|
| 31:24 | +4 | BASE | **Segment Base Address:** Three fields which collectively define the base location for the segment in 4 GB physical address space. |
| 7:0 | +4 | | |
| 31:16 | +0 | | |
| 19:16 | +4 | LIMIT | **Segment Limit:** Two fields that define the size of the segment based on the Segment Limit Granularity Bit.<br>If G = 1: Limit value interpreted in units of 4 KB.<br>If G = 0: Limit value is interpreted in bytes. |
| 15:0 | +0 | | |
| 23 | +4 | G | **Segment Limit Granularity Bit:** Defines LIMIT multiplier.<br>If G = 1: Limit value interpreted in units of 4 KB. Segment size ranges from 4 KB to 4 GB.<br>If G = 0: Limit value is interpreted in bytes. Segment size ranges from 1 byte to 1 MB. |
| 22 | +4 | D | **Default Length for Operands and Effective Addresses:**<br>If D = 1: Code segment = 32-bit length for operands and effective addresses.<br>If D = 0: Code segment = 16-bit length for operands and effective addresses.<br>If D = 1: Data segment = Pushes, calls and pop instructions use 32-bit ESP register.<br>If D = 0: Data segment = Stack operations use 16-bit SP register. |
| 20 | +4 | AVL | **Segment Available:** This field is available for use by system software. |
| 15 | +4 | P | **Segment Present:**<br>If = 1: Segment is memory segment allocated.<br>If = 0: The BASE and LIMIT fields become available for use by the system. Also, If = 0, a segment-not-present exception generated when selector for the descriptor is loaded into a segment register allowing virtual memory management. |
| 14:13 | +4 | DPL | **Descriptor Privilege Level:**<br>If = 00: Highest privilege level<br>If = 11: Lowest privilege level |
| 12 | +4 | S | **Descriptor Type:**<br>If = 1: Code or data segment<br>If = 0: System segment |
| 11:8 | +4 | TYPE | **Segment Type: Refer to Table 3-23 for TYPE bit definitions.**<br>Bit 11 = Executable<br>Bit 10 = Conforming if Bit 12 = 1<br>Bit 10 = Expand Down if Bit 12 = 0<br>Bit 9 = Readable, if Bit 12 = 1<br>Bit 9 = Writable, if Bit 12 = 0<br>Bit 8 = Accessed |

## Processor Programming (Continued)

### Table 3-23. Application and System Segment Descriptors TYPE Bit Definitions

| TYPE Bits [11:8] | | System Segment and Gate Types Bit 12 = 0 | Application Segment Types Bit 12 = 1 | |
|---|---|---|---|---|
| Num | SEWA | TYPE (Data Segments) | | |
| 0 | 0000 | Reserved | Data | Read-Only |
| 1 | 0001 | Available 16-Bit TSS | Data | Read-Only, accessed |
| 2 | 0010 | LDT | Data | Read/Write |
| 3 | 0011 | Busy 16-Bit TSS | Data | Read/Write accessed |
| 4 | 0100 | 16-Bit Call Gate | Data | Read-Only, expand down |
| 5 | 0101 | Task Gate | Data | Read-Only, expand down, accessed |
| 6 | 0110 | 16-Bit Interrupt Gate | Data | Read/Write, expand down |
| 7 | 0111 | 16-Bit Trap Gate | Data | Read/Write, expand down, accessed |
| Num | SCRA | TYPE (Code Segments) | | |
| 8 | 1000 | Reserved | Code | Execute-Only |
| 9 | 1001 | Available 32-Bit TSS | Code | Execute-Only, accessed |
| A | 1010 | Reserved | Code | Execute/Read |
| B | 1011 | Busy 32-Bit TSS | Code | Execute/Read, accessed |
| C | 1100 | 32-Bit Call Gate | Code | Execute/Read, conforming |
| D | 1101 | Reserved | Code | Execute/Read, conforming, accessed |
| E | 1110 | 32-Bit Interrupt Gate | Code | Execute/Read-Only, conforming |
| F | 1111 | 32-Bit Trap Gate | Code | Execute/Read-Only, conforming accessed |

SEWA/SCRA:S = Code Segment (not Data Segment)
E = Expand Down
W = Write Enable
A = Accessed
C = Conforming Code Segment
R = Read Enable

## Processor Programming (Continued)

### Gate Descriptors

Four kinds of gate descriptors are used to provide protection during control transfers:

- Call gates
- Trap gates
- Interrupt gates
- Task gates

(For more information on protection refer to Section 3.9 “Protection” on page 91.)

**Call Gate Descriptor (CGD).** Call gates are used to define legal entry points to a procedure with a higher privilege level. The call gates are used by CALL and JUMP instructions in much the same manner as code segment descriptors. When a decoded instruction refers to a call gate descriptor in the GDT or LDT, the call gate is used to point to another descriptor in the table that defines the destination code segment.

The following privilege levels are tested during the transfer through the call gate:

- CPL = Current Privilege Level
- RPL = Segment Selector Field
- DPL = Descriptor Privilege Level in the call gate descriptor
- DPL = Descriptor Privilege Level in the destination code segment

The maximum value of the CPL and RPL must be equal or less than the gate DPL. For a JMP instruction the destination DPL equals the CPL. For a CALL instruction the destination DPL is less than or equal to the CPL.

**Conforming Code Segments.** Transfer to a procedure with a higher privilege level can also be accomplished by bypassing the use of call gates, if the requested procedure is to be executed in a conforming code segment. Conforming code segments have the C bit set in the TYPE field in their descriptor.

The bit structure and definitions for gate descriptors are shown in Tables 3-24 and 3-25.

**Table 3-24. Gate Descriptors**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Memory Offset +4** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| OFFSET[31:16] | | | | | | | | | | | | | | | | P | DPL | | 0 | TYPE | | | | 0 | 0 | 0 | PARAMETERS | | | | |
| **Memory Offset +0** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SELECTOR[15:0] | | | | | | | | | | | | | | | | OFFSET[15:0] | | | | | | | | | | | | | | | |

**Table 3-25. Gate Descriptors Bit Definitions**

| Bit | Memory Offset | Name | Description |
|---|---|---|---|
| 31:16 | +4 | OFFSET | **Offset:** Offset used during a call gate to calculate the branch target. |
| 15:0 | +0 | | |
| 31:16 | +0 | SELECTOR | **Segment Selector** |
| 15 | +4 | P | **Segment Present** |
| 14:13 | +4 | DPL | **Descriptor Privilege Level** |
| 11:8 | +4 | TYPE | **Segment Type:**<br>0100 = 16-bit call gate<br>0101 = Task gate<br>0110 = 16-bit interrupt gate<br>0111 = 16-bit trap gate<br>1100 = 32-bit call gate<br>1110 = 32-bit interrupt gate<br>1111 = 32-bit trap gate |
| 4:0 | +4 | PARAMETERS | **Parameters:** Number of parameters to copy from the caller’s stack to the called procedure’s stack. |

## Processor Programming (Continued)

### Task State Segments Descriptors

The CPU enables rapid task switching using JMP and CALL instructions that refer to Task State Segment (TSS) descriptors. During a switch, the complete task state of the current task is stored in its TSS, and the task state of the requested task is loaded from its TSS. The TSSs are defined through special segment descriptors and gates.

The **Task Register (TR)** holds 16-bit descriptors that contain the base address and segment limit for each task state segment. The TR is loaded and stored via the LTR and STR instructions, respectively. The TR can be accessed only during protected mode and can be loaded when the privilege level is 0 (most privileged). When the TR is loaded, the TR selector field indexes a TSS descriptor that must reside in the Global Descriptor Table (GDT).

Only the 16-bit selector of a TSS descriptor in the TR is accessible. The BASE, TSS LIMIT and ACCESS RIGHT fields are program invisible.

During task switching, the processor saves the current CPU state in the TSS before starting a new task. The TSS can be either a 386/486-type 32-bit TSS (see Table 3-26) or a 286-type 16-bit TSS (see Table 3-27).

**Task Gate Descriptors**. A task gate descriptor provides controlled access to the descriptor for a task switch. The DPL of the task gate is used to control access. The selector's RPL and the CPL of the procedure must be a higher level (numerically less) than the DPL of the descriptor. The RPL in the task gate is not used.

The I/O Map Base Address field in the 32-bit TSS points to an I/O permission bit map that often follows the TSS at location +68h.

**Table 3-26. 32-Bit Task State Segment (TSS) Table**

| 31 | | | | | | | | | | | | | | | 16 | 15 | | | | | | | | | | | | | | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I/O Map Base Address | | | | | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | T | +64h |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Selector for Task's LDT | | | | | | | | | | | | | | | | +60h |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | GS | | | | | | | | | | | | | | | | +5Ch |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | FS | | | | | | | | | | | | | | | | +58h |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | DS | | | | | | | | | | | | | | | | +54h |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | SS | | | | | | | | | | | | | | | | +50h |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | CS | | | | | | | | | | | | | | | | +4Ch |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ES | | | | | | | | | | | | | | | | +48h |
| EDI | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +44h |
| ESI | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +40h |
| EBP | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +3Ch |
| ESP | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +38h |
| EBX | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +34h |
| EDX | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +30h |
| ECX | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +2Ch |
| EAX | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +28h |
| EFLAGS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +24h |
| EIP | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +20h |
| CR3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +1Ch |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | SS for CPL = 2 | | | | | | | | | | | | | | | | +18h |
| ESP for CPL = 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +14h |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | SS for CPL = 1 | | | | | | | | | | | | | | | | +10h |
| ESP for CPL = 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +Ch |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | SS for CPL = 0 | | | | | | | | | | | | | | | | +8h |
| ESP for CPL = 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | +4h |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Back Link (Old TSS Selector) | | | | | | | | | | | | | | | | +0h |

**Note:** 0 = Reserved

## Processor Programming (Continued)

### Table 3-27. 16-Bit Task State Segment (TSS) Table

| 15 0 | |
|---|---|
| Selector for Task's LDT | +2Ah |
| DS | +28h |
| SS | +26h |
| CS | +24h |
| ES | +22h |
| DI | +20h |
| SI | +1Eh |
| BP | +1Ch |
| SP | +1Ah |
| BX | +18h |
| DX | +16h |
| CX | +14h |
| AX | +12h |
| FLAGS | +10h |
| IP | +Eh |
| SS for Privilege Level 0 | +Ch |
| SP for Privilege Level 1 | +Ah |
| SS for Privilege Level 1 | +8h |
| SP for Privilege Level 1 | +6h |
| SS for Privilege Level 0 | +4h |
| SP for Privilege Level 0 | +2h |
| Back Link (Old TSS Selector) | +0h |

### 3.5.4 Paging Mechanism

The paging mechanism translates a linear address to its corresponding physical address. If the required page is not currently present in RAM, an exception is generated. When the operating system services the exception, the required page can be loaded into memory and the instruction restarted. Pages are either 4 KB or 1 MB in size. The CPU defaults to 4 KB pages that are aligned to 4 KB boundaries.

A page is addressed by using two levels of tables as illustrated in Figure 3-8. Bits [31:22] of the 32-bit linear address, the Directory Table Index (DTI), are used to locate an entry in the page directory table. The page directory table acts as a 32-bit master index to up to 1 KB individual second-level page tables. The selected entry in the page directory table, referred to as the directory table entry (DTE), identifies the starting address of the second-level page table. The page directory table itself is a page and is therefore aligned to a 4 KB boundary. The physical address of the current page directory table is stored in the CR3 control register, also referred to as the Page Directory Base Register (PDBR).

Bits [21:12] of the 32-bit linear address, referred to as the Page Table Index (PTI), locate a 32-bit entry in the second-level page table. This page table entry (PTE) contains the base address of the desired page frame. The second-level page table addresses up to 1K individual page frames. A second-level page table is 4 KB in size and is itself a page. Bits [11:0] of the 32-bit linear address, the Page Frame Offset (PFO), locate the desired physical data within the page frame.

Since the page directory table can point to 1 KB page tables, and each page table can point to 1 KB page frames, a total of 1 MB page frames can be implemented. Each page frame contains 4 KB, therefore, up to 4 GB of virtual memory can be addressed by the CPU with a single page directory table.

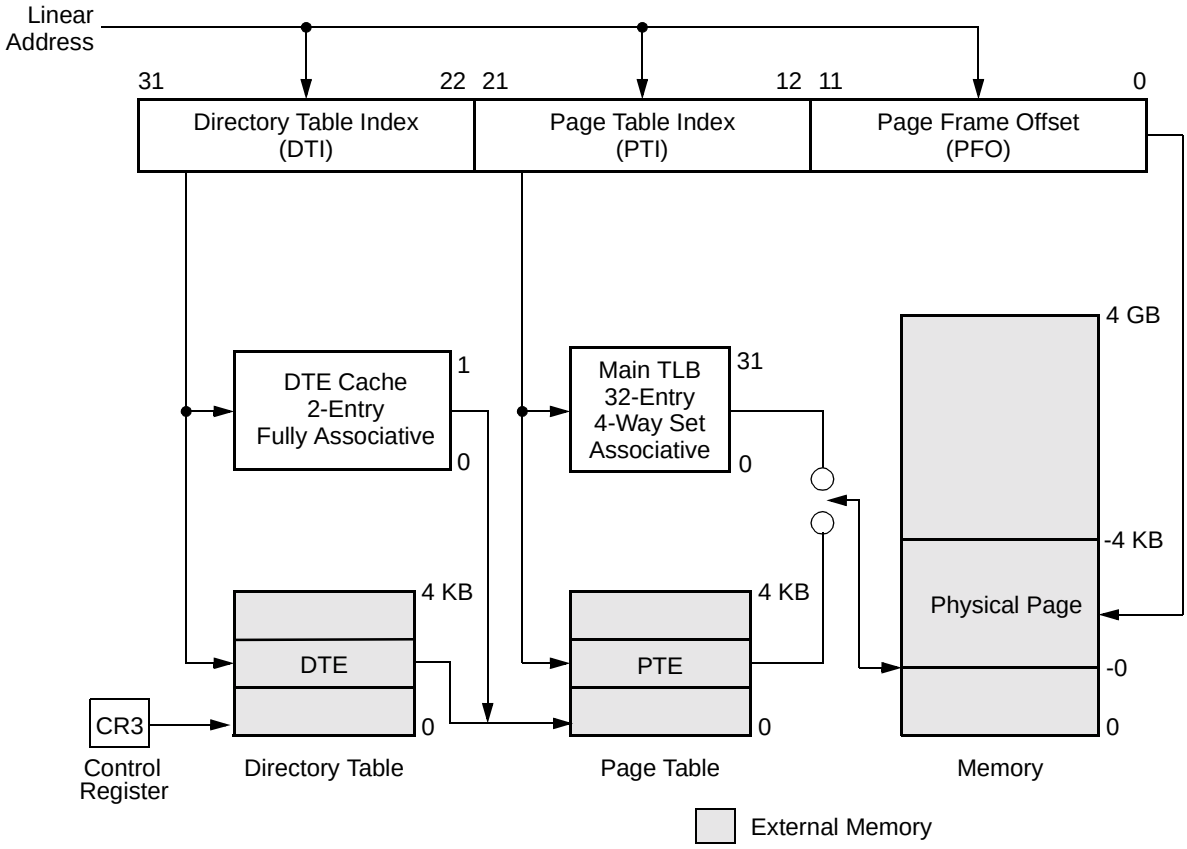



**Figure 3-8. Paging Mechanism**

## Processor Programming (Continued)

Along with the base address of the page table or the page frame, each DTE or PTE contains attribute bits and a present bit as illustrated in Table 3-28.

If the present bit (P) is set in the DTE, the page table is present and the appropriate page table entry is read. If P = 1 in the corresponding PTE (indicating that the page is in memory), the accessed and dirty bits are updated, if necessary, and the operand is fetched. Both accessed bits are set (DTE and PTE), if necessary, to indicate that the table and the page have been used to translate a linear address. The dirty bit (D) is set before the first write is made to a page.

The present bits must be set to validate the remaining bits in the DTE and PTE. If either of the present bits are not set, a page fault is generated when the DTE or PTE is accessed. If P = 0, the remaining DTE/PTE bits are available for use by the operating system. For example, the operating system can use these bits to record where on the hard disk the pages are located. A page fault is also generated if the memory reference violates the page protection attributes.

### Translation Look-Aside Buffer

The translation look-aside buffer (TLB) is a cache for the paging mechanism and replaces the two-level page table lookup procedure for TLB hits. The TLB is a four-way set associative 32-entry page table cache that automatically keeps the most commonly used page table entries in the processor. The 32-entry TLB, coupled with a 4 KB page size, results in coverage of 128 KB of memory addresses.

The TLB must be flushed when entries in the page tables are changed. The TLB is flushed whenever the CR3 register is loaded. An individual entry in the TLB can be flushed using the INVLPG instruction.

### DTE Cache

The DTE cache caches the two most recent DTEs so that future TLB misses only require a single page table read to calculate the physical address. The DTE cache is disabled following RESET and can be enabled by setting the DTE_EN bit in CCR4[4] (see CCR4 register on page 53).

**Table 3-28. Directory Table Entry (DTE) and Page Table Entry (PTE)**

| Bit | Name | Description |
|---|---|---|
| 31:12 | BASE ADDRESS | **Base Address:** Specifies the base address of the page or page table. |
| 11:9 | AVAILABLE | **Available:** Undefined and available to the programmer. |
| 8:7 | RSVD | **Reserved:** Unavailable to programmer. |
| 6 | D | **Dirty Bit:**<br>PTE format — If = 1: Indicates that a write access has occurred to the page.<br>DTE format — Reserved. |
| 5 | A | **Accessed Flag:** If set, indicates that a read access or write access has occurred to the page. |
| 4:3 | RSVD | **Reserved:** Set to 0. |
| 2 | U/S | **User/Supervisor Attribute:**<br>If = 1: Page is accessible by User at privilege level 3.<br>If = 0: Page is accessible by Supervisor only when CPL ≤ 2. |
| 1 | W/R | **Write/Read Attribute:**<br>If = 1: Page is writable.<br>If = 0: Page is read only. |
| 0 | P | **Present Flag:**<br>If = 1: The page is present in RAM and the remaining DTE/PTE bits are validated<br>If = 0: The page is not present in RAM and the remaining DTE/PTE bits are available for use by the programmer. |

## 3.6 INTERRUPTS AND EXCEPTIONS

The processing of either an interrupt or an exception changes the normal sequential flow of a program by transferring program control to a selected service routine. Except for SMM interrupts, the location of the selected service routine is determined by one of the interrupt vectors stored in the interrupt descriptor table.

True interrupts are hardware interrupts and are generated by signal sources external to the CPU. All exceptions (including so-called software interrupts) are produced internally by the CPU.

### 3.6.1 Interrupts

External events can interrupt normal program execution by using one of the three interrupt pins on the GXLV processor:

- Non-maskable Interrupt (No pin, see note)
- Maskable Interrupt (INTR pin)
- SMM Interrupt (SMI# pin)

**Note:** There is not an NMI pin on the GXLV processor. Generation of an NMI interrupt is not possible. However, software can generate an NMI by setting bit 2 of CCR7. (See the CCR7 register on page 53.)

For most interrupts, program transfer to the interrupt routine occurs after the current instruction has been completed. When the execution returns to the original program, it begins immediately following the interrupted instruction.

The **NMI interrupt** cannot be masked by software and always uses interrupt vector two to locate its service routine. Since the interrupt vector is fixed and is supplied internally, no interrupt acknowledge bus cycles are performed. This interrupt is normally reserved for unusual situations such as parity errors and has priority over INTR interrupts.

Once NMI processing has started, no additional NMIs are processed until an IRET instruction is executed, typically at the end of the NMI service routine. If the NMI is reasserted before execution of the IRET instruction, one and only one NMI rising edge is stored and then processed after execution of the next IRET.

During the NMI service routine, maskable interrupts may be enabled. If an unmasked INTR occurs during the NMI service routine, the INTR is serviced and execution returns to the NMI service routine following the next IRET. If a HALT instruction is executed within the NMI service routine, the CPU restarts execution only in response to RESET, an unmasked INTR or a System Management Mode (SMM) interrupt. NMI does not restart CPU execution under this condition.

The **INTR interrupt** is unmasked when the Interrupt Enable Flag (IF, bit 9) in the EFLAGS register is set to 1 (See the EFLAGS Register in Table 3-4 on page 46). Except for string operations, INTR interrupts are acknowledged between instructions. Long string operations have interrupt windows between memory moves that allow INTR interrupts to be acknowledged.

When an INTR interrupt occurs, the CPU performs an interrupt-acknowledge bus cycle. During this cycle, the CPU reads an 8-bit vector that is supplied by an external interrupt controller. This vector selects which of the 256 possible interrupt handlers will be executed in response to the interrupt.

The **SMM interrupt** has higher priority than either INTR or NMI. After SMI# is asserted, program execution is passed to an SMM service routine that runs in SMM address space reserved for this purpose. The remainder of this section does not apply to the SMM interrupts. SMM interrupts are described in greater detail later in Section 3.7 “System Management Mode” on page 83.

### 3.6.2 Exceptions

Exceptions are generated by an interrupt instruction or a program error. Exceptions are classified as traps, faults or aborts depending on the mechanism used to report them and the restartability of the instruction which first caused the exception.

A **Trap exception** is reported immediately following the instruction that generated the trap exception. Trap exceptions are generated by execution of a software interrupt instruction (INTO, INT3, INTn, BOUND), by a single-step operation or by a data breakpoint.

Software interrupts can be used to simulate hardware interrupts. For example, an INTn instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table. Execution of the interrupt service routine occurs regardless of the state of the IF flag (bit 9) in the EFLAGS register.

The one byte INT3, or breakpoint interrupt (vector 3), is a particular case of the INTn instruction. By inserting this one byte instruction in a program, the user can set breakpoints in the code that can be used during debug.

## Processor Programming (Continued)

Single-step operation is enabled by setting the TF bit (bit 8) in the EFLAGS register. When the TF is set, the CPU generates a debug exception (vector 1) after the execution of every instruction. Data breakpoints also generate a debug exception and are specified by loading the debug registers (DR0-DR3, see Table 3-12 on page 55) with the appropriate values.

A **Fault exception** is reported before completion of the instruction that generated the exception. By reporting the fault before instruction completion, the CPU is left in a state that allows the instruction to be restarted and the effects of the faulting instruction to be nullified. Fault exceptions include divide-by-zero errors, invalid opcodes, page faults and coprocessor errors. Debug exceptions (vector 1) are also handled as faults (except for data breakpoints and single-step operations). After execution of the fault service routine, the instruction pointer points to the instruction that caused the fault.

An **Abort exception** is a type of fault exception that is severe enough that the CPU cannot restart the program at the faulting instruction. The double fault (vector 8) is the only abort exception that occurs on the CPU.

### 3.6.3 Interrupt Vectors

When the CPU services an interrupt or exception, the current program's instruction pointer and flags are pushed onto the stack to allow resumption of execution of the interrupted program. In protected mode, the processor also saves an error code for some exceptions. Program control is then transferred to the interrupt handler (also called the interrupt service routine). Upon execution of an IRET at the end of the service routine, program execution resumes at the instruction pointer address saved on the stack when the interrupt was serviced.

#### 3.6.3.1 Interrupt Vector Assignments

Each interrupt (except SMI#) and exception are assigned one of 256 interrupt vector numbers as shown in Table 3-29. The first 32 interrupt vector assignments are defined or reserved. INT instructions acting as software interrupts may use any of interrupt vectors, 0 through 255.

The non-maskable hardware interrupt (NMI) is assigned vector 2. Illegal opcodes including faulty FPU instructions will cause an illegal opcode exception, interrupt vector 6. NMI interrupts are enabled by setting bit 2 of the CCR7 register (Index EBh[2] = 1, see Table 3-11 on page 52 for register format).

In response to a maskable hardware interrupt (INTR), the CPU issues interrupt acknowledge bus cycles used to read the vector number from external hardware. These vectors should be in the range 32 to 255 as vectors 0 to 31 are predefined.

#### 3.6.3.2 Interrupt Descriptor Table

The interrupt vector number is used by the CPU to locate an entry in the interrupt descriptor table (IDT). In real mode, each IDT entry consists of a 4-byte far pointer to the beginning of the corresponding interrupt service routine. In protected mode, each IDT entry is an 8-byte descriptor. The Interrupt Descriptor Table Register (IDTR) specifies the beginning address and limit of the IDT. Following RESET, the IDTR contains a base address of 00000000h with a limit of 3FFh.

The IDT can be located anywhere in physical memory as determined by the IDTR register. The IDT may contain different types of descriptors: interrupt gates, trap gates and task gates. Interrupt gates are used primarily to enter a hardware interrupt handler. Trap gates are generally used to enter an exception handler or software interrupt handler. If an interrupt gate is used, the Interrupt Enable Flag (IF) in the EFLAGS register is cleared before the interrupt handler is entered. Task gates are used to make the transition to a new task.

**Table 3-29. Interrupt Vector Assignments**

| Interrupt Vector | Function | Exception Type |
|---|---|---|
| 0 | Divide error | Fault |
| 1 | Debug exception | Trap/Fault* |
| 2 | NMI interrupt | --- |
| 3 | Breakpoint | Trap |
| 4 | Interrupt on overflow | Trap |
| 5 | BOUND range exceeded | Fault |
| 6 | Invalid opcode | Fault |
| 7 | Device not available | Fault |
| 8 | Double fault | Abort |
| 9 | Reserved | --- |
| 10 | Invalid TSS | Fault |
| 11 | Segment not present | Fault |
| 12 | Stack fault | Fault |
| 13 | General protection fault | Trap/Fault |
| 14 | Page fault | Fault |
| 15 | Reserved | --- |
| 16 | FPU error | Fault |
| 17 | Alignment check exception | Fault |
| 18:31 | Reserved | --- |
| 32:55 | Maskable hardware interrupts | Trap |
| 0:255 | Programmed interrupt | Trap |

**Note:** *Data breakpoints and single steps are traps. All other debug exceptions are faults.

### 3.6.4 Interrupt and Exception Priorities

As the CPU executes instructions, it follows a consistent policy for prioritizing exceptions and hardware interrupts. The priorities for competing interrupts and exceptions are listed in Table 3-30. SMM interrupts always take precedence. Debug traps for the previous instruction and next instructions are handled as the next priority. When NMI and maskable INTR interrupts are both detected at the same instruction boundary, the GXLV processor services the NMI interrupt first.

The CPU checks for exceptions in parallel with instruction decoding and execution. Several exceptions can result from a single instruction. However, only one exception is generated upon each attempt to execute the instruction. Each exception service routine should make the appropriate corrections to the instruction and then restart the instruction. In this way, exceptions can be serviced until the instruction executes properly.

The CPU supports instruction restart after all faults, except when an instruction causes a task switch to a task whose Task State Segment (TSS) is partially not present. A TSS can be partially not present if the TSS is not page aligned and one of the pages where the TSS resides is not currently in memory.

**Table 3-30. Interrupt and Exception Priorities**

| Priority | Description | Notes |
|---|---|---|
| 0 | Reset. | Caused by the assertion of RESET. |
| 1 | SMM hardware interrupt. | SMM interrupts are caused by SMI# asserted and always have highest priority. |
| 2 | Debug traps and faults from previous instruction. | Includes single-step trap and data breakpoints specified in the debug registers. |
| 3 | Debug traps for next instruction. | Includes instruction execution breakpoints specified in the debug registers. |
| 4 | Non-maskable hardware interrupt. | Caused by NMI asserted. |
| 5 | Maskable hardware interrupt. | Caused by INTR asserted and IF = 1. |
| 6 | Faults resulting from fetching the next instruction. | Includes segment not present, general protection fault and page fault. |
| 7 | Faults resulting from instruction decoding. | Includes illegal opcode, instruction too long, or privilege violation. |
| 8 | WAIT instruction and TS = 1 and MP = 1. | Device not available exception generated. |
| 9 | ESC instruction and EM = 1 or TS = 1. | Device not available exception generated. |
| 10 | Floating point error exception. | Caused by unmasked floating point exception with NE = 1. |
| 11 | Segmentation faults (for each memory reference required by the instruction) that prevent transferring the entire memory operand. | Includes segment not present, stack fault, and general protection fault. |
| 12 | Page Faults that prevent transferring the entire memory operand. | |
| 13 | Alignment check fault. | |

## Processor Programming (Continued)

### 3.6.5 Exceptions in Real Mode

Many of the exceptions described in Table 3-29 "Interrupt Vector Assignments" on page 80 are not applicable in real mode. Exceptions 10, 11, and 14 do not occur in real mode. Other exceptions have slightly different meanings in real mode as listed in Table 3-31.

**Table 3-31. Exception Changes in Real Mode**

| Vector Number | Protected Mode Function | Real Mode Function |
|---|---|---|
| 8 | Double fault. | Interrupt table limit overrun. |
| 10 | Invalid TSS. | Does not occur. |
| 11 | Segment not present. | Does not occur. |
| 12 | Stack fault. | SS segment limit overrun. |
| 13 | General protection fault. | CS, DS, ES, FS, GS segment limit overrun. In protected mode, an error code is pushed. In real mode, no error code is pushed. |
| 14 | Page fault. | Does not occur. |

### 3.6.6 Error Codes

When operating in protected mode, the following exceptions generate a 16-bit error code:

- Double Fault
- Alignment Check
- Invalid TSS
- Segment Not Present
- Stack Fault
- General Protection Fault
- Page Fault

The error code format and bit definitions are shown in Table 3-32. Bits [15:3] (selector index) are not meaningful if the error code was generated as the result of a page fault. The error code is always zero for double faults and alignment check exceptions.

**Table 3-32. Error Codes**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Selector Index | | | | | | | | | | | | | S2 | S1 | S0 |

**Table 3-33. Error Code Bit Definitions**

| Fault Type | Selector Index (Bits 15:3) | S2 (Bit 2) | S1 (Bit 1) | S0 (Bit 0) |
|---|---|---|---|---|
| Page Fault | Reserved. | Fault caused by:<br>0 = Not present page<br>1 = Page-level protection violation | Fault occurred during:<br>0 = Read access<br>1 = Write access | Fault occurred during<br>0 = Supervisor access<br>1 = User access. |
| IDT Fault | Index of faulty IDT selector. | Reserved | 1 | If = 1, exception occurred while trying to invoke exception or hardware interrupt handler. |
| Segment Fault | Index of faulty selector. | TI bit of faulty selector | 0 | If =1, exception occurred while trying to invoke exception or hardware interrupt handler. |

## 3.7 SYSTEM MANAGEMENT MODE

System Management Mode (SMM) is an enhancement of the standard x86 architecture. SMM is usually employed for system power management or software-transparent emulation of I/O peripherals. SMM is entered through a hardware signal “System Management Interrupt” (SMI# pin) that has a higher priority than any other interrupt, including NMI. An SMM interrupt can also be triggered from software using an SMINT instruction. Following an SMM interrupt, portions of the CPU state are automatically saved, SMM is entered, and program execution begins at the base of SMM address space (Figure 3-9).

The GXLV processor extends System Management Mode to support the virtualization of many devices, including VGA video. The SMM mechanism can be triggered not only by I/O activity, but by access to selected memory regions. For example, SMM interrupts are generated when VGA addresses are accessed. As will be described, other SMM enhancements have reduced SMM overhead and improved virtualization-software performance

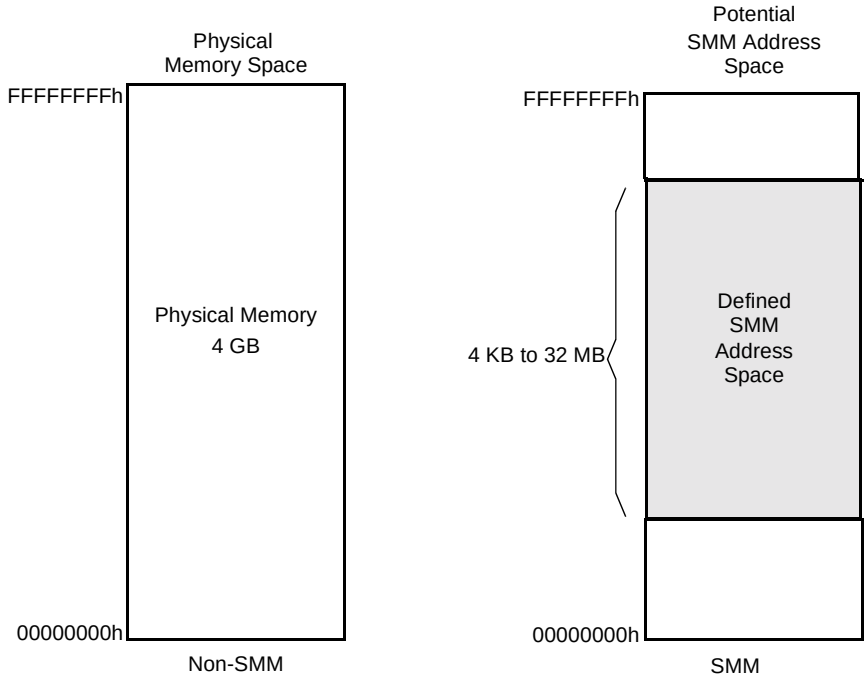



**Figure 3-9. System Management Memory Address Space**

## Processor Programming (Continued)

### 3.7.1 SMM Operation

SMM execution flow is summarized in Figure 3-10. Entering SMM requires the assertion of the SMI# pin for at least two SYSCLK periods or execution of the SMINT instruction. For the SMI# signal or SMINT instruction to be recognized, the following configuration registers must be programmed:

- SMAR (Index CDh-CFh) - The SMM Base address and size.
- CCR1 (Index C1) - SMAC bit and/or USE_SMI bit.

These registers formats are given in Table 3-11 on page 52.

After triggering an SMM through the SMI# pin or a SMINT instruction, selected CPU state information is automatically saved in the SMM memory space header located at the top of SMM memory space. After saving the header, the CPU enters real mode and begins executing the SMM service routine starting at the SMM memory region base address.

The SMM service routine is user definable and may contain system or power management software. If the power management software forces the CPU to power down or if the SMM service routine modifies more registers than are automatically saved, the complete CPU state information should be saved.

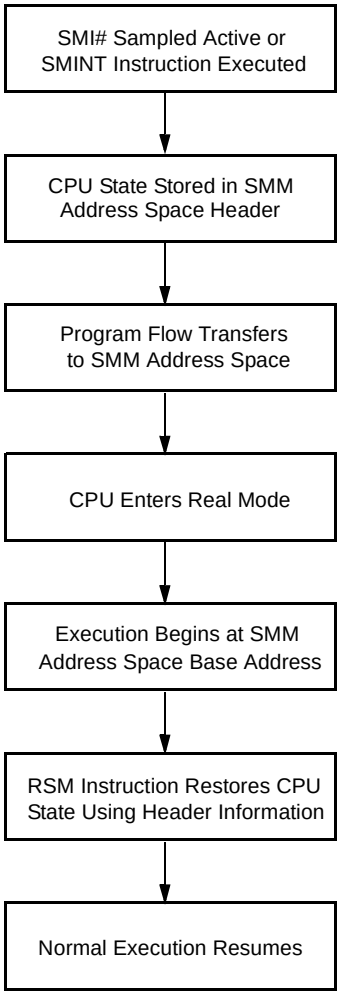



**Figure 3-10. SMM Execution Flow**

## Processor Programming (Continued)

### 3.7.2 SMI# Pin

External chipsets can generate an SMI based on numerous asynchronous events, including power management timers, I/O address trapping, external devices, audio FIFO events, and others. Since SMI# is edge sensitive, the chipset must generate an edge for each of the events above, requiring arbitration and storage of multiple SMM events. These functions are provided by the CS5530 I/O companion device. The processor generates an SMI when the external pin changes from high-to-low or when an Resume (RSM) occurs if SMI# has not remained low since the initiation of the previous SMI.

### 3.7.3 SMM Configuration Registers

The SMAR register specifies the base location of SMM code region and its size limit.

The SMHR register specifies the 32-bit physical address of the SMM header. The SMHR address must be 32-bit aligned as the bottom two bits are ignored by the microcode. Hardware will detect write operations to SMAR, and signal the microcode to recompute the header address. Access to the SMAR and SMHR registers is enabled by MAPEN (Index C3h[4] see bit details on page 52).

The SMAR register writes to the SMHR register when the SMAR register is changed. For this reason, changes to the SMAR register should be completed prior to setting up the SMHR register. The configuration registers bit formats are detailed in Table 3-11 beginning on page 52.

### 3.7.4 SMM Memory Space Header

Tables 3-34 and 3-35 show the SMM header. A memory address field has been added to the end (offset -40h) of the header for the GXLV processor. Memory data will be stored overlapping the I/O data, since these events cannot occur simultaneously. The I/O address is valid for both IN and OUT instructions, and I/O data is valid only for OUT. The memory address is valid for read and write operations, and memory data is valid only for write operations.

With every SMI interrupt or SMINT instruction, selected CPU state information is automatically saved in the SMM memory space header located at the top of SMM address space. The header contains CPU state information that is modified when servicing an SMM interrupt. Included in this information are two pointers. The Current IP points to the instruction executing when the SMI was detected, but it is valid only for an internal I/O SMI.

The Next IP points to the instruction that will be executed after exiting SMM. The contents of Debug Register 7 (DR7), the Extended Flags Register (EFLAGS), and Control Register 0 (CR0) are also saved. If SMM has been entered due to an I/O trap for a REP INSx or REP OUTSx instruction, the Current IP and Next IP fields contain the same addresses. In addition, the I and P fields contain valid information.

If entry into SMM is the result of an I/O trap, it is useful for the programmer to know the port address, data size and data value associated with that I/O operation. This information is also saved in the header and is valid only if SMI# is asserted during an I/O bus cycle. The I/O trap information is not restored within the CPU when executing a RSM instruction.

**Table 3-34. SMM Memory Space Header**

| Mem. Offset | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -04h | DR7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| –08h | EFLAGS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| –0Ch | CR0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| –10h | Current IP | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| –14h | Next IP | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| –18h | RSVD | | | | | | | | | | | | | | | | CS Selector | | | | | | | | | | | | | | | |
| –1Ch | CS Descriptor [63:32] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| –20h | CS Descriptor [31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| –24h | RSVD | | | | | | | | | | | | | | | | RSVD | | | | | | | N | V | X | M | H | S | P | I | C |
| –28h | I/O Data Size | | | | | | | | | | | | | | | | I/O Address [15:0] | | | | | | | | | | | | | | | |
| –2Ch | I/O or Memory Data [31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| –30h | Restored ESI or EDI | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| –34h | I/O or Memory Address [31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **Note:** | Check the M bit at offset 24 h to determine if the data is memory or I/O. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

## Processor Programming (Continued)

### Table 3-35. SMM Memory Space Header Description

| Name | Description | Size |
|---|---|---|
| DR7 | **Debug Register 7:** The contents of Debug Register 7. | 4 Bytes |
| EFLAGS | **Extended Flags Register:** The contents of Extended Flags Register. | 4 Bytes |
| CR0 | **Control Register 0:** The contents of Control Register 0. | 4 Bytes |
| Current IP | **Current Instruction Pointer:** The address of the instruction executed prior to servicing SMM interrupt. | 4 Bytes |
| Next IP | **Next Instruction Pointer:** The address of the next instruction that will be executed after exiting SMM. | 4 Bytes |
| CS Selector | **Code Segment Selector:** Code segment register selector for the current code segment. | 2 Bytes |
| CS Descriptor | **Code Segment Descriptor:** Encoded descriptor bits for the current code segment. | 8 Bytes |
| N | **Nested SMI Status:** Flag that determines whether an SMI occurred during SMM (i.e., nested). | 1 Bit |
| V | **SoftVGA SMI Status:** SMI was generated by an access to VGA region. | 1 Bit |
| X | **External SMI Status:**<br>If = 1: SMI generated by external SMI# pin.<br>If = 0: SMI internally generated by Internal Bus Interface Unit. | 1 Bit |
| M | **Memory or I/O Access:** 0 = I/O access; 1 = Memory access. | 1 Bit |
| H | **Halt Status:** Indicates that the processor was in a halt or shutdown prior to servicing the SMM interrupt. | 1 Bit |
| S | **Software SMM Entry Indicator:**<br>If = 1: Current SMM is the result of an SMINT instruction.<br>If = 0: Current SMM is not the result of an SMINT instruction. | 1 Bit |
| P | **REP INSx/OUTSx Indicator:**<br>If = 1: Current instruction has a REP prefix.<br>If = 0: Current instruction does not have a REP prefix. | 1 Bit |
| I | **IN, INSx, OUT, or OUTSx Indicator:**<br>If = 1: Current instruction performed is an I/O WRITE.<br>If = 0: Current instruction performed is an I/O READ. | 1 Bit |
| C | **CS Writable:** Code Segment Writable<br>If = 1: CS is writable<br>If = 0: CS is not writable | 1 Bit |
| I/O Data Size | **Indicates size of data for the trapped I/O cycle:**<br>01h = BYTE<br>03h = WORD<br>0Fh = DWORD | 2 Bytes |
| I/O Address | **Processor port used for the trapped I/O cycle** | 2 Bytes |
| I/O or Memory Data | **Data associated with the trapped I/O or memory cycle** | 4 Bytes |
| Restored ESI or EDI | **Restored ESI or EDI Value:** Used when it is necessary to repeat a REP OUTSx or REP INSx instruction when one of the I/O cycles caused an SMI# trap. | 4 Bytes |
| Memory Address | **Physical address of the operation that caused the SMI** | 4 Bytes |

**Note:** INSx = INS, INSB, INSW or INSD instruction.
OUTSx = OUTS, OUTSB, OUTSW and OUTSD instruction.

### 3.7.5 SMM Instructions

The GXLV processor core automatically saves a minimal amount of CPU state information when entering SMM which allows fast SMM service routine entry and exit. After entering the SMM service routine, the MOV, SVDC, SVLDT and SVTS instructions can be used to save the complete CPU state information. If the SMM service routine modifies more state information than is automatically saved or if it forces the CPU to power down, the complete CPU state information must be saved. Since the CPU is a static device, its internal state is retained when the input clock is stopped. Therefore, an entire CPU-state save is not necessary before stopping the input clock.

The SMM instructions, listed in Table 3-36, can be executed only if all the conditions listed below are met.

1. USE_SMI = 1.
2. SMAR size > 0.
3. Current Privilege Level = 0.
4. SMAC bit is high or the CPU is in an SMM service routine.

If any one of the conditions above is not met and an attempt is made to execute an SVDC, RSDC, SVLDT, RSLDT, SVTS, RSTS, or RSM instruction, an invalid opcode exception is generated. The SMM instructions can be executed outside of defined SMM space provided the conditions above are met.

The SMINT instruction can be used by software to enter SMM. The SMINT instruction can only be used outside an SMM routine if all the conditions listed below are true.

1. USE_SMI = 1
2. SMAR size > 0
3. Current Privilege Level = 0
4. SMAC = 1

If SMI# is asserted to the CPU during a software SMI, the hardware SMI# is serviced after the software SMI has been exited by execution of the RSM instruction.

All the SMM instructions (except RSM and SMINT) save or restore 80 bits of data, allowing the saved values to include the hidden portion of the register contents.

**Table 3-36. SMM Instruction Set**

| Instruction | Opcode | Format | Description |
|---|---|---|---|
| SVDC | 0F 78h [mod sreg3 r/m] | SVDC mem80, sreg3 | **Save Segment Register and Descriptor:**<br>Saves reg (DS, ES, FS, GS, or SS) to mem80. |
| RSDC | 0F 79h [mod sreg3 r/m] | RSDC sreg3, mem80 | **Restore Segment Register and Descriptor:**<br>Restores reg (DS, ES, FS, GS, or SS) from mem80. Use RSM to restore CS.<br>**Note:** Processing “RSDC CS, mem80” will produce an exception. |
| SVLDT | 0F 7Ah [mod 000 r/m] | SVLDT mem80 | **Save LDTR and Descriptor:**<br>Saves Local Descriptor Table (LDTR) to mem80. |
| RSLDT | 0F 7Bh [mod 000 r/m] | RSLDT mem80 | **Restore LDTR and Descriptor:**<br>Restores Local Descriptor Table (LDTR) from mem80. |
| SVTS | 0F 7Ch [mod 000 r/m] | SVTS mem80 | **Save TSR and Descriptor:**<br>Saves Task State Register (TSR) to mem80. |
| RSTS | 0F 7Dh [mod 000 r/m] | RSTS mem80 | **Restore TSR and Descriptor:**<br>Restores Task State Register (TSR) from mem80. |
| SMINT | 0F 38h | SMINT | **Software SMM Entry:**<br>CPU enters SMM. CPU state information is saved in SMM memory space header and execution begins at SMM base address. |
| RSM | 0F AAh | RSM | **Resume Normal Mode:**<br>Exits SMM. The CPU state is restored using the SMM memory space header and execution resumes at interrupted point. |

**Note:** mem80 = 80-bit memory location.

## Processor Programming (Continued)

### 3.7.6 SMM Memory Space

SMM memory space is defined by specifying the base address and size of the SMM memory space in the SMAR register. The base address must be a multiple of the SMM memory space size. For example, a 32 KB SMM memory space must be located at a 32 KB address boundary. The memory space size can range from 4 KB to 32 MB. Execution of the interrupt begins at the base of the SMM memory space.

SMM memory space accesses are always cacheable, which allows SMM routines to run faster.

### 3.7.7 SMI Generation for Virtual VGA

The GXLV processor implements SMI generation for VGA accesses. When enabled memory write operations in regions A0000h to AFFFFh, B0000h to B7FFFh, and B8000h to BFFFFh generate an SMI. Memory reads are not trapped by the GXLV processor. When enabled, the GXLV processor traps I/O addresses for VGA in the following regions: 3B0h to 3BFh, 3C0h to 3CFh, and 3D0h to 3DFh. Memory-write trapping is performed during instruction decode in the processor core. I/O read and write trapping is implemented in the Internal Bus Interface Unit of the GXLV processor.

The SMI-generation hardware requires two additional configuration registers to control and mask SMI interrupts in the VGA memory space: VGACTL and VGAM. The VGACTL register has a control bit for each address range shown above. The VGAM register has 32 bits that can selectively disable 2 KB regions within the VGA memory. The VGAM applies only to the A0000h to AFFFFh region. If this region is not enabled in VGA_CTL, then the contents of VGAM is ignored. The purpose of VGAM is to prevent an SMI from occurring when non-displayed VGA memory is accessed. This is an enhancement which improves performance for double-buffered applications. The format of each register is shown in Table 4-37 on page 163.

### 3.7.8 SMM Service Routine Execution

Upon entry into SMM, after the SMM header has been saved, the CR0, EFLAGS, and DR7 registers are set to their reset values. The Code Segment (CS) register is loaded with the base, as defined by the SMAR register, and a limit of 4 GB. The SMM service routine then begins execution at the SMM base address in real mode.

The programmer must save, restore the value of any registers not saved in the header that may be changed by the SMM service routine. For data accesses immediately after entering the SMM service routine, the programmer must use CS as a segment override. I/O port access is possible during the routine but care must be taken to save registers modified by the I/O instructions. Before using a segment register, the register and the register's descriptor cache contents should be saved using the SVDC instruction.

Hardware interrupts, INTRs and NMIs, may be serviced during an SMM service routine. If interrupts are to be serviced while executing in the SMM memory space, the SMM memory space must be within the address range of 0 to 1 MB to guarantee proper return to the SMM service routine after handling the interrupt.

INTRs are automatically disabled when entering SMM since the IF flag (EFLAGS register, bit 9) is set to its reset value. Once in SMM, the INTR can be enabled by setting the IF flag. An NMI event in SMM can be enabled by setting NMI_EN high in the CCR3 register (Index C3h[1]). If NMI is not enabled while in SMM, the CPU latches one NMI event and services the interrupt after NMI has been enabled or after exiting SMM through the RSM instruction. Upon entering SMM, the processor is in real mode, but it may exit to either real or protected mode depending on its state when SMM was initiated. The SMM header indicates to which state it will exit.

Within the SMM service routine, protected mode may be entered and exited as required, and real or protected mode device drivers may be called.

To exit the SMM service routine, an RSM instruction, rather than an IRET, is executed. The RSM instruction causes the GXLV processor core to restore the CPU state using the SMM header information and resume execution at the interrupted point. If the full CPU state was saved by the programmer, the stored values should be reloaded before executing the RSM instruction using the MOV, RSDC, RSLDT and RSTS instructions.

#### 3.7.8.1 SMI Nesting

The SMI mechanism supports nesting of SMI interrupts through the SMM service routine the SMI_NEST bit in the CCR4 register (Index E8h[6]), and the Nested SMI Status bit (bit N in the SMM header, see Table 3-35 "SMM Memory Space Header Description" on page 86). Nesting is an important capability in allowing high-priority events, such as audio virtualization, to interrupt lower-priority SMI code for VGA virtualization or power management. SMI_NEST controls whether SMI interrupts can occur during SMM. SMM service routines can optionally set SMI_NEST high to allow higher-priority SMI interrupts while handling the current event.

The SMM service routine is responsible for managing the SMM header data for nested SMI interrupts. The SMM header must be saved before SMI_NEST is set high, and SMI_NEST must be cleared and its header information restored before an RSM instruction is executed.

The Nested SMI Status bit has been added to the SMM header to show whether the current SMI is nested. The processor sets Nested SMI Status high if the processor was in SMM when the SMI was taken. The processor uses Nested SMI Status on exit to determine whether the processor should stay in SMM.

## Processor Programming (Continued)

When SMI nesting is disabled, the processor holds off external SMI interrupts until the currently executing SMM code exits. When SMI nesting is enabled, the processor can proceed with the SMI. The SMM service routine will guarantee that no internal SMIs are generated in SMM, so the processor ignores such events. If the internal and external SMI signals are received simultaneously, then the internal SMI is given priority to avoid losing the event.

The state diagram of the SMI_NEST and Nested SMI Status bits are shown in Figure 3-11 with each state explained next.

A. When the processor is outside of SMM, Nested SMI Status is always clear and SMI_NEST is set high.

B. The first-level SMI interrupt is received by the processor. The microcode clears SMI_NEST, sets Nested SMI Status high and saves the previous value of Nested SMI Status (0) in the SMM header.

C. The first-level SMM service routine saves the header and sets SMI_NEST high to re-enable SMI interrupts from SMM.

D. A second-level (nested) SMI interrupt is received by the processor. This SMI is taken even though the processor is in SMM because the SMI_NEST bit is set high. The microcode clears SMI_NEST, sets Nested SMI Status high and saves the previous value of Nested SMI Status (1) in the SMM header.

E. The second-level SMM service routine saves the header and sets SMI_NEST to re-enable SMI interrupts within SMM. Another level of nesting could occur during this period.

F. The second-level SMM service routine clears SMI_NEST to disable SMI interrupts, then restores its SMM header.

G. The second-level SMM service routine executes an RSM. The microcode sets SMI_NEST, and restores the Nested SMI Status (1) based on the SMM header.

H. The first-level SMM service routine clears SMI_NEST to disable SMI interrupts, then restores its SMM header.

I. The first-level SMM service routine executes an RSM. The microcode sets SMI_NEST high and restores the Nested SMI Status (0) based on the SMM header.

When the processor is outside of SMM, Nested SMI Status is always clear and SMI_NEST is set high.

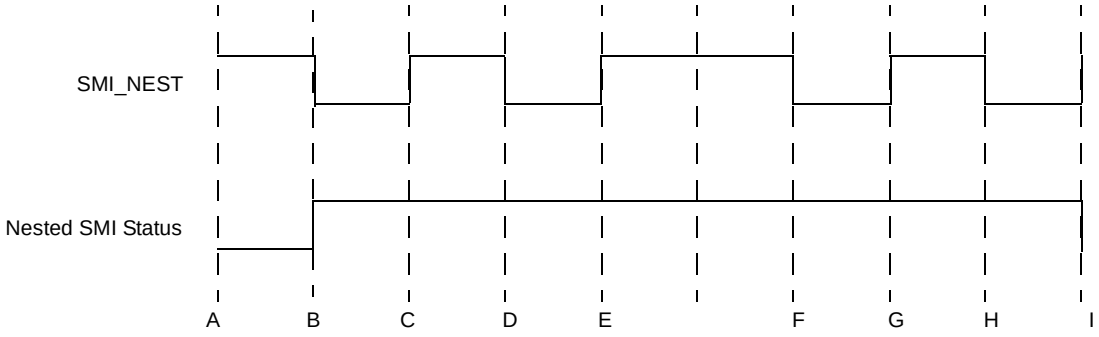



**Figure 3-11. SMI Nesting State Machine**

## Processor Programming (Continued)

### 3.7.8.2 CPU States Related to SMM and Suspend Mode

The state diagram shown in Figure 3-12 illustrates the various CPU states associated with SMM and Suspend mode. While in the SMM service routine, the GXLV processor core can enter Suspend mode either by (1) executing a halt (HLT) instruction or (2) by asserting the SUSP# input.

During SMM operations and while in SUSP#-initiated Suspend mode, an occurrence of either NMI or INTR is latched. (In order for INTR to be latched, the IF flag, EFLAGS register bit 9, must be set.) The INTR or NMI is serviced after exiting Suspend mode.

If Suspend mode is entered through a HLT instruction from the operating system or application software, the reception of an SMI# interrupt causes the CPU to exit Suspend mode and enter SMM. If Suspend mode is entered through the hardware (SUSP# = 0) while the operating system or application software is active, the CPU latches one occurrence of INTR, NMI, and SMI#.

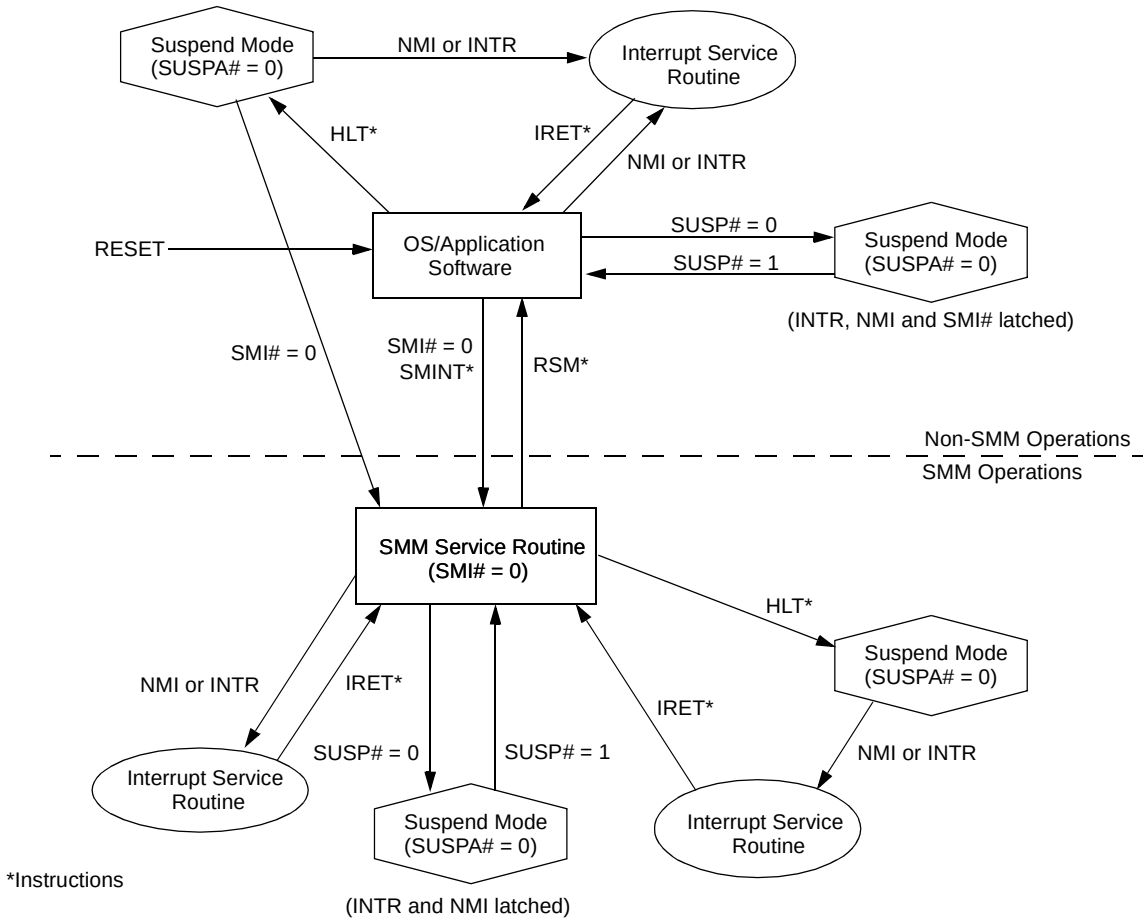



**Figure 3-12. SMM and Suspend Mode State Diagram**

## 3.8 HALT AND SHUTDOWN

The halt instruction (HLT) stops program execution and generates the Halt bus cycle on the PCI bus. The GXLV processor core then drives out a Stop Grant bus cycle and enters a low-power Suspend mode if the SUSP_HLT bit in CCR2 (Index C2h[3]) is set. SMI#, NMI, INTR with interrupts enabled (IF bit in EFLAGS = 1), or RESET forces the CPU out of the halt state. If the halt state is interrupted, the saved code segment and instruction pointer specify the instruction following the HLT.

Shutdown occurs when a severe error is detected that prevents further processing. The most common severe error is the triple fault, a fault event while handling a double fault. Setting the IDT limit to zero or the GDT limit to zero will cause a triple fault when in protected mode.

A RESET brings the processor out of shutdown. An NMI will work if the IDT limit is large enough, at least 000Fh, to contain the NMI interrupt vector and if the stack has enough room. The stack must be large enough to contain the vector and flag information (the stack pointer must be greater than 0005h).

## 3.9 PROTECTION

Segment protection and page protection are safeguards built into the GXLV processor's protected-mode architecture that deny unauthorized or incorrect access to selected memory addresses. These safeguards allow multitasking programs to be isolated from each other and from the operating system. This section concentrates on segment protection.

Selectors and descriptors are the key elements in the segment protection mechanism. The segment base address, size, and privilege level are established by a segment descriptor. Privilege levels control the use of privileged instructions, I/O instructions and access to segments and segment descriptors. Selectors are used to locate segment descriptors.

Segment accesses are divided into two basic types, those involving code segments (e.g., control transfers) and those involving data accesses. The ability of a task to access a segment depends on the:

- Segment type
- Instruction requesting access
- Type of descriptor used to define the segment
- Associated privilege levels (described next)

Data stored in a segment can be accessed only by code executing at the same or a more privileged level. A code segment or procedure can only be called by a task executing at the same or a less privileged level.

### 3.9.1 Privilege Levels

The values for privilege levels range between 0 and 3. Level 0 is the highest privilege level (most privileged), and level 3 is the lowest privilege level (least privileged). The privilege level in real mode is zero.

The **Descriptor Privilege Level** (DPL) is the privilege level defined for a segment in the segment descriptor. The DPL field specifies the minimum privilege level needed to access the memory segment pointed to by the descriptor.

The **Current Privilege Level** (CPL) is defined as the current task's privilege level. The CPL of an executing task is stored in the hidden portion of the code segment register and essentially is the DPL for the current code segment.

The **Requested Privilege Level** (RPL) specifies a selector's privilege level. RPL is used to distinguish between the privilege level of a routine actually accessing memory (the CPL), and the privilege level of the original requester (the RPL) of the memory access. The lesser of the RPL and CPL is called the Effective Privilege Level (EPL). Therefore, if RPL = 0 in a segment selector, the EPL is always determined by the CPL. If RPL = 3, the EPL is always 3 regardless of the CPL. If the level requested by RPL is less than the CPL, the RPL level is accepted and the EPL is changed to the RPL value. If the level requested by RPL is greater than CPL, the CPL overrides the requested RPL and EPL becomes the CPL value.

For a memory access to succeed, the EPL must be at least as privileged as the Descriptor Privilege Level (EPL ≤ DPL). If the EPL is less privileged than the DPL (EPL > DPL), a general protection fault is generated. For example, if a segment has a DPL = 2, an instruction accessing the segment only succeeds if executed with an EPL ≤ 2.

### 3.9.2 I/O Privilege Levels

The I/O Privilege Level (IOPL) allows the operating system executing at CPL = 0 to define the least privileged level at which IOPL-sensitive instructions can unconditionally be used. The IOPL-sensitive instructions include CLI, IN, OUT, INS, OUTS, REP INS, REP OUTS, and STI. Modification of the IF bit in the EFLAGS register is also sensitive to the I/O privilege level.

The IOPL is stored in the EFLAGS register (bits [31:12]). An I/O permission bit map is available as defined by the 32-bit Task State Segment (TSS). Since each task can have its own TSS, access to individual I/O ports can be granted through separate I/O permission bit maps.

If CPL ≤ IOPL, IOPL-sensitive operations can be performed. If CPL > IOPL, a general protection fault is generated if the current task is associated with a 16-bit TSS. If the current task is associated with a 32-bit TSS and CPL > IOPL, the CPU consults the I/O permission bitmap in the TSS to determine on a port-by-port basis whether or not I/O instructions (IN, OUT, INS, OUTS, REP INS, REP OUTS) are permitted. The remaining IOPL-sensitive operations generate a general protection fault.

## Processor Programming (Continued)

### 3.9.3 Privilege Level Transfers

A task's CPL can be changed only through intersegment control transfers using gates or task switches to a code segment with a different privilege level. Control transfers result from exception and interrupt servicing and from execution of the CALL, JMP, INT, IRET and RET instructions.

There are five types of control transfers that are summarized in Table 3-37. Control transfers can be made only when the operation causing the control transfer references the correct descriptor type. Any violation of these descriptor usage rules causes a general protection fault.

Any control transfer that changes the CPL within a task results in a change of stack. The initial values for the stack segment (SS) and stack pointer (ESP) for privilege levels 0, 1, and 2 are stored in the TSS. During a JMP or CALL control transfer, the SS and ESP are loaded with the new stack pointer and the previous stack pointer is saved on the new stack. When returning to the original privilege level, the RET or IRET instruction restores the SS and ESP of the less-privileged stack.

**Table 3-37. Descriptor Types Used for Control Transfer**

| Type of Control Transfer | Operation Types | Descriptor Referenced | Descriptor Table |
|---|---|---|---|
| Intersegment within the same privilege level. | JMP, CALL, RET, IRET* | Code Segment | GDT or LDT |
| Intersegment to the same or a more privileged level. Interrupt within task (could change CPL level). | CALL | Gate Call | GDT or LDT |
| | Interrupt Instruction, Exception, External Interrupt | Trap or Interrupt Gate | IDT |
| Intersegment to a less privileged level (changes task CPL). | RET, IRET* | Code Segment | GDT or LDT |
| Task Switch via TSS | CALL, JMP | Task State Segment | GDT |
| Task Switch via Task Gate | CALL, JMP | Task Gate | GDT or LDT |
| | IRET**, Interrupt Instruction, Exception, External Interrupt | Task Gate | IDT |

**Note:** *NT = 0 (Nested Task bit in EFLAGS, bit 14)
**NT =1 (Nested Task bit in EFLAGS, bit 14)

#### 3.9.3.1 Gates

Gate descriptors described in Section "Gate Descriptors" on page 74, provide protection for privilege transfers among executable segments. Gates are used to transition to routines of the same or a more privileged level. Call gates, interrupt gates and trap gates are used for privilege transfers within a task. Task gates are used to transfer between tasks.

Gates conform to the standard rules of privilege. In other words, gates can be accessed by a task if the effective privilege level (EPL) is the same or more privileged than the gate descriptor's privilege level (DPL).

### 3.9.4 Initialization and Transition to Protected Mode

The GXLV processor core switches to real mode immediately after RESET. While operating in real mode, the system tables and registers should be initialized. The GDTR and IDTR must point to a valid GDT and IDT, respectively. The size of the IDT should be at least 256 bytes, and the GDT must contain descriptors that describe the initial code and data segments.

The processor can be placed in protected mode by setting the PE bit (CR0 register bit 0). After enabling protected mode, the CS register should be loaded and the instruction decode queue should be flushed by executing an intersegment JMP. Finally, all data segment registers should be initialized with appropriate selector values.

## 3.10 VIRTUAL 8086 MODE

Both real mode and virtual 8086 (V86) modes are supported by the GXLV processor, allowing execution of 8086 application programs and 8086 operating systems. V86 mode allows the execution of 8086-type applications, yet still permits use of the paging and protection mechanisms. V86 tasks run at privilege level 3. Before entry, all segment limits must be set to FFFFh (64K) as in real mode.

### 3.10.1 Memory Addressing

While in V86 mode, segment registers are used in an identical fashion to real mode. The contents of the Segment register are multiplied by 16 and added to the offset to form the Segment Base Linear Address. The GXLV processor permits the operating system to select which programs use the V86 address mechanism and which programs use protected mode addressing for each task.

The GXLV processor also permits the use of paging when operating in V86 mode. Using paging, the 1 MB address space of the V86 task can be mapped to any region in the 4 GB linear address space.

The paging hardware allows multiple V86 tasks to run concurrently, and provides protection and operating system isolation. The paging hardware must be enabled to run multiple V86 tasks or to relocate the address space of a V86 task to physical address space other than 0.

### 3.10.2 Protection

All V86 tasks operate with the least amount of privilege (level 3) and are subject to all CPU protected mode protection checks. As a result, any attempt to execute a privileged instruction within a V86 task results in a general protection fault.

In V86 mode, a slightly different set of instructions are sensitive to the I/O privilege level (IOPL) than in protected mode. These instructions are: CLI, INT n, IRET, POPF, PUSHF, and STI. The INT3, INTO and BOUND variations of the INT instruction are not IOPL sensitive.

### 3.10.3 Interrupt Handling

To fully support the emulation of an 8086-type machine, interrupts in V86 mode are handled as follows. When an interrupt or exception is serviced in V86 mode, program execution transfers to the interrupt service routine at privilege level 0 (i.e., transition from V86 to protected mode occurs). The VM bit in the EFLAGS register (bit 17) is cleared. The protected mode interrupt service routine then determines if the interrupt came from a protected mode or V86 application by examining the VM bit in the EFLAGS image stored on the stack. The interrupt service routine may then choose to allow the 8086 operating system to handle the interrupt or may emulate the function of the interrupt handler. Following completion of the interrupt service routine, an IRET instruction restores the EFLAGS register (restores VM = 1) and segment selectors and control returns to the interrupted V86 task.

### 3.10.4 Entering and Leaving Virtual 8086 Mode

V86 mode is entered from protected mode by either executing an IRET instruction at CPL = 0 or by task switching. If an IRET is used, the stack must contain an EFLAGS image with VM = 1. If a task switch is used, the TSS must contain an EFLAGS image containing a 1 in the VM bit position. The POPF instruction cannot be used to enter V86 mode since the state of the VM bit is not affected. V86 mode can only be exited as the result of an interrupt or exception. The transition out must use a 32-bit trap or interrupt gate that must point to a non-conforming privilege level 0 segment (DPL = 0), or a 32-bit TSS. These restrictions are required to permit the trap handler to IRET back to the V86 program.

## 3.11 FLOATING POINT UNIT OPERATIONS

The GXLV processor contains an FPU that is x87 and MMX instruction-set compatible and adheres to the IEEE-754 standard. Because most applications that contain FPU instructions intermix with integer instructions, the GXLV processor's FPU achieves high performance by completing integer and FPU operations in parallel.

### 3.11.1 FPU Register Set

The FPU provides the user eight data registers, a control register, and a status register. The CPU also provides a data register tag word that improves context switching and stack performance by maintaining empty/non-empty status for each of the eight data registers. Two additional, registers contain pointers to (a) the memory location containing the current instruction word and (b) the memory location containing the operand associated with the current instruction word (if any).

### 3.11.2 FPU Tag Word Register

The FPU maintains a tag word register that is divided into eight tag word fields. These fields assume one of four values depending on the contents of their associated data registers: Valid (00), Zero (01), Special (10), and Empty (11). Note: Denormal, Infinity, QNaN, SNaN and unsupported formats are tagged as "Special." Tag values are maintained transparently by the CPU and are only available to the programmer indirectly through the FSTENV and FSAVE instructions. The tag word with TAG fields for each associated physical register, TAG(n), is shown in Table 3-38.

### 3.11.3 FPU Status Register

The FPU communicates status information and operation results to the CPU through the FPU status register, whose fields are detailed in Table 3-38. These fields include information related to exception status, operation execution status, register status, operand class, and comparison results. This register is continuously accessible to the CPU regardless of the state of the Control or Execution Units.

### 3.11.4 FPU Mode Control Register

The FPU Mode Control Register, shown in Table 3-38, is used by the GXLV processor to specify the operating mode of the FPU. The register fields include information related to the rounding mode selected, the amount of precision to be used in the calculations, and the exception conditions which should be reported to the GXLV processor using traps. The user controls precision, rounding, and exception reporting by setting or clearing appropriate bits.

### Table 3-38. FPU Registers

| Bit | Name | Description |
|---|---|---|
| **FPU Tag Word Register (R/W) (Note)** | | |
| 15:14 | TAG7 | **TAG7:** 00 = Valid; 01 = Zero; 10 = Special; 11 = Empty. |
| 13:12 | TAG6 | **TAG6:** 00 = Valid; 01 = Zero; 10 = Special; 11 = Empty. |
| 11:10 | TAG5 | **TAG5:** 00 = Valid; 01 = Zero; 10 = Special; 11 = Empty. |
| 9:8 | TAG4 | **TAG4:** 00 = Valid; 01 = Zero; 10 = Special; 11 = Empty. |
| 7:6 | TAG3 | **TAG3:** 00 = Valid; 01 = Zero; 10 = Special; 11 = Empty. |
| 5:4 | TAG2 | **TAG2:** 00 = Valid; 01 = Zero; 10 = Special; 11 = Empty. |
| 3:2 | TAG1 | **TAG1:** 00 = Valid; 01 = Zero; 10 = Special; 11 = Empty. |
| 1:0 | TAG0 | **TAG0:** 00 = Valid; 01 = Zero; 10 = Special; 11 = Empty. |
| **FPU Status Register (R/W) (Note)** | | |
| 15 | B | **Copy of ES bit** (bit 7 this register) |
| 14 | C3 | **Condition code bit 3** |
| 13:11 | S | **Top-of-Stack:** Register number that points to the current TOS. |
| 10:8 | C[2:0] | **Condition code bits [2:0]** |
| 7 | ES | **Error indicator:** Set to 1 if unmasked exception detected. |
| 6 | SF | **Stack Full:** FPU Status Register: or invalid register operation bit. |
| 5 | P | **Precision error exception bit** |
| 4 | U | **Underflow error exception bit** |
| 3 | O | **Overflow error exception bit** |
| 2 | Z | **Divide-by-zero exception bit** |
| 1 | D | **Denormalized-operand error exception bit** |
| 0 | I | **Invalid operation exception bit** |
| **FPU Mode Control Register (R/W) (Note)** | | |
| 15:12 | RSVD | **Reserved:** Set to 0 |
| 11:10 | RC | **Rounding control bits:**<br>00 = Round to nearest or even<br>01 = Round towards minus infinity<br>10 = Round towards plus infinity<br>11 = Truncate |
| 9:8 | PC | **Precision control bits:**<br>00 = 24-bit mantissa<br>01 = Reserved<br>10 = 53-bit mantissa<br>11 = 64-bit mantissa |
| 7:6 | RSVD | **Reserved:** Set to 0 |
| 5 | P | **Precision error exception bit** |
| 4 | U | **Underflow error exception bit** |
| 3 | O | **Overflow error exception bit** |
| 2 | Z | **Divide-by-zero exception bit** |
| 1 | D | **Denormalized-operand error exception bit** |
| 0 | I | **Invalid-operation exception bit** |
| **Note:** R/W only through the environment at store and restore commands. | | |

## 4.0 Integrated Functions

The integrated functions in the Geode GXLV processor are:

- Internal bus interface
- SDRAM memory controller
- High-performance 2D graphics accelerator
- Display controller with separate CRT and TFT data paths
- PCI bridge

The design organizes the memory controller, graphics pipeline and display controller into a Unified Memory Architecture (UMA). UMA simplifies system designs and significantly reduces overall system costs associated with high chip count, small footprint designs. Performance degradation in traditional UMA systems is reduced through the use of National Semiconductor's Display Compression Technology (DCT) architecture.

Figure 4-1 shows the major functional blocks of the GXLV processor and how the Internal Bus Interface Unit operates as the interface between the processor's core units and the integrated functions.

This section details how the integrated functions and Internal Bus Interface Unit operate and their respective registers.

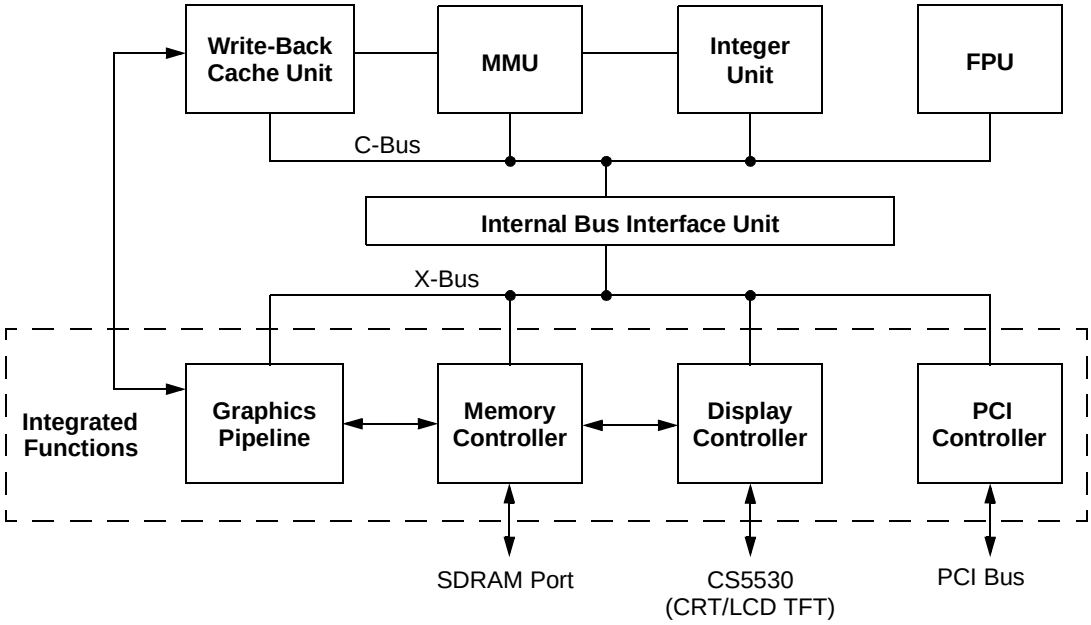



**Figure 4-1. Internal Block Diagram**

## 4.1 INTEGRATED FUNCTIONS PROGRAMMING INTERFACE

The GXLV processor's integrated functions programming interface is a memory mapped space. The control registers for the graphics pipeline, display controller, and memory controller are located in this space, as well as all the graphics memory: frame buffer, compression buffer etc. This memory address space is referred to as the GXLV processor memory space.

### 4.1.1 Graphics Control Register

The base address for these memory mapped registers is programmed in the Graphics Configuration Register (GCR, Index B8h, bits[1:0]), shown in Table 4-1. The GCR only specifies address bits [31:30] of physical memory. The remaining address bits [29:0] are fixed to zero. The GCR is I/O mapped because it must be accessed before memory mapping can be enabled. Refer to Section 3.3.2.2 "Configuration Registers" on page 50 for information on how to access this register.

The GXLV processor incorporates graphics functions that require registers to implement and control them. Most of these registers are memory mapped and physically located in the logical units they control. The mapping of these units is controlled by the GCR register.

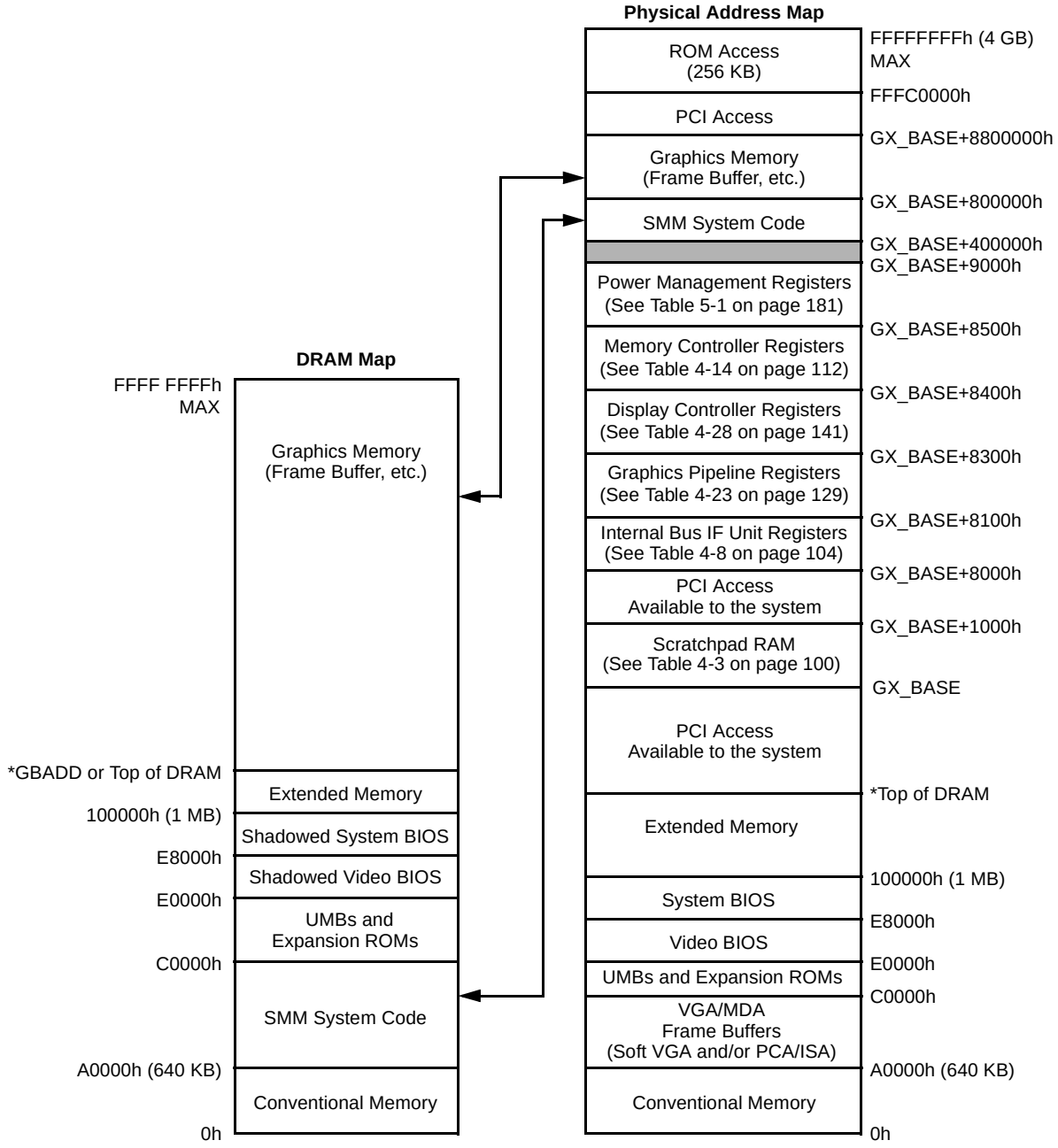
Figure 4-2 shows the complete memory address map for the GXLV processor. When accessing the GXLV processor memory space, address bits [29:24] must be zero. This means that the GXLV processor accesses a linear address space with a total of 16 MB. Address bit 23 divides this space into 8 MB for control (bit 23 = 0) and 8 MB for graphics memory (bit 23 = 1). In control space, bits [22:16] are not decoded, so the programmer should set them to zero. Address bit 15 divides the remaining 64 KB address space into scratchpad RAM and PCI access (bit 15 = 0) and control registers (bit 15 = 1). Note that scratchpad RAM is placed here by programming the tags appropriately.

Device drivers are responsible for performing physical-to-virtual memory-address translation, including allocation of selectors that point to the GXLV processor. All memory decoded by the processor may be accessed in protected mode by creating a selector with the physical address equal to the GXLV Base Address which is shown in Table 4-1, and a limit of 16 MB. Additionally, a selector with only a 64 KB limit is large enough to access all of the GXLV processor's registers and scratchpad RAM.

**Table 4-1. GCR Register**

| Bit | Name | Description |
|---|---|---|
| **Index B8h** | | **GCR Register (R/W)** **Default Value = 00h** |
| 7:4 | RSVD | **Reserved:** Set to 0. |
| 3:2 | SP | **Scratchpad Size:** Specifies the size of the scratchpad cache.<br>00 = 0 KB; Graphics instruction disabled (see Section 4.1.5 "Display Driver Instructions" on page 102).<br>01 = 2 KB<br>10 = 3 KB<br>11 = 4 KB |
| 1:0 | GX | **GXLV Base Address:** Specifies the physical address for the base (GX_BASE) of the scratchpad RAM, the graphics memory (frame buffer, compression buffer, etc.) and the other memory mapped registers.<br>00 = Scratchpad RAM, Graphics Subsystem, and memory-mapped configuration registers are disabled.<br>01 = Scratchpad RAM and control registers start at GX_BASE = 40000000h.<br>10 = Scratchpad RAM and control registers start at GX_BASE = 80000000h.<br>11 = Scratchpad RAM and control registers start at GX_BASE = C0000000h. |

## Integrated Functions (Continued)




* See BC_DRAM_TOP in Table 4-8 on page 104 or MC_GBASE_ADD in Table 4-15 on page 116.

**Figure 4-2. GXLV Processor Memory Space**

## Integrated Functions (Continued)

### 4.1.2 Control Registers

The control registers for the GXLV processor use 32 KB of the memory map, starting at GX_BASE+8000h (see Figure 4-2). This area is divided into internal bus interface unit, graphics pipeline, display controller, memory controller, and power management sections:

- The internal bus interface unit maps 100h locations starting at GX_BASE+8000h.
- The graphics pipeline maps 200h locations starting at GX_BASE+8100h.
- The display controller maps 100h locations starting at GX_BASE+8300h.
- The memory controller maps 100h locations starting at GX_BASE+8400h
- GX_BASE+8500h-8FFFh is dedicated to power management registers for the serial packet transmission control, the user-defined power management address space, Suspend Refresh, and SMI status for Suspend/Resume.

The register descriptions are contained in the individual subsections of this chapter. Accesses to undefined registers in the GXLV processor control register space will not cause a hardware error.

### 4.1.3 Graphics Memory

Graphics memory is allocated from system DRAM by the system BIOS. The GXLV processor's graphics memory is mapped into 4 MB starting at GX_BASE+800000h. This area includes the frame buffer memory and storage for internal display controller state. The size of the frame buffer is a linear map whose size depends on the user's requirements (i.e., resolution, color depth, video buffer, compression buffer, font caching, etc.). Frame buffer scan lines are not contiguous in many resolutions, so software that renders to the frame buffer must use a skip count to advance between scan lines. The display controller can use the graphics memory that lies between scan lines for the compression buffer. A*ccessing graphics memory between the end of a scan line and the start of another can cause display problems*. The skip count for all supported resolutions is shown in Table 4-2.

The graphics memory size is programmed by setting the graphics memory base address in the memory controller (see Table 4-15 on page 113). Display drivers communicate with system BIOS about resolution changes, to ensure that the correct amount of graphics memory is allocated. Since no mechanism exists to recover system DRAM from the operating system without rebooting *when a graphics resolution change requires an increased amount of graphics memory, the system must be rebooted!*

.

**Table 4-2. Display Resolution Skip Counts**

| Screen Resolution | Pixel Depth | Skip Count |
|---|---|---|
| 640x480 | 8 bits | 1024 |
| 640x480 | 16 bits | 2048 |
| 800x600 | 8 bits | 1024 |
| 800x600 | 16 bits | 2048 |
| 1024x768 | 8 bits | 1024 |
| 1024x768 | 16 bits | 2048 |
| 1280x1024 | 8 bits | 2048 |

## Integrated Functions (Continued)

### 4.1.4 Scratchpad RAM

To improve software performance for specific applications, part of the L1 cache (2, 3, or 4 KB) can be programed to operate as a scratchpad RAM. This scratchpad RAM operates at L1 speed which can speed up time-critical software operations. The scratchpad RAM is taken from set 0 of the L1 cache. Setting aside this RAM makes the L1 cache smaller by the scratchpad RAM size. The scratchpad RAM size is controlled by bits in the GCR register (Index B8h, bits[3:2]). See Table 4-1 on page 97.

The scratchpad RAM is usually memory mapped by BIOS to the upper memory region defined by the GCR register (Index B8h, bits [1:0]). Once enabled, the valid bits for the scratchpad RAM will always be true and the scratchpad RAM locations will never be flushed to external memory. The scratchpad RAM serves as a general purpose high speed RAM and as a BLT buffer for the graphics pipeline.

#### 4.1.4.1 Initialization of Scratchpad RAM

The scratchpad RAM must be initialized before the L1 cache is enabled. To initialize the scratchpad RAM after a cold boot:

1) Initialize the tags of the scratchpad RAM using the test registers TR4 and TR5 as outlined in Section 3.3.2.4 “TLB Test Registers”. The tags are normally programmed with an address value equivalent to GX_BASE (GCR register).
2) Enable the scratchpad RAM to the desired size (GCR register). This action will also lock down the tags.
3) Enable the L1 cache. Section 3.3.2.1 “Control Registers”.

#### 4.1.4.2 Scratchpad RAM Utilization

Use of scratchpad RAM by applications and drivers must be tightly controlled. To avoid conflicts, application software and third-party drivers should generally avoid accesses to the scratchpad RAM area. The scratchpad RAM is used by the graphics pipeline BLT buffers, and National-supplied display drivers and virtualization software. Table 4-3 describes the 2 KB, 3 KB, and 4 KB scratchpad RAM organization used by National developed software. The BLT buffers are programmed using CPU_READ/CPU_WRITE instructions described in Section 4.1.6 on page 102. If the graphics pipeline or National software is used, and it is desirable to use scratchpad RAM by software other than that supplied by National, please contact your local National Semiconductor technical support representative.

#### 4.1.4.3 BLT Buffer

Address registers, BitBLT, have been added to the front end of the L1 cache to enable the graphics pipeline to directly access a portion of the scratchpad RAM as a BLT buffer. Table 4-4 summarizes these registers. These registers do not have default values and must be initialized before use. Table 4-5 gives the register/bit formats. A 16-byte line buffer dedicated to the graphics pipeline BLT operations has been added to minimize accesses to the L1 cache.

When the BLT operation begins, the graphics pipeline generates a 32 bit data BLT request to the L1 cache. This request goes through the BitBLT registers to produce an address into the scratchpad RAM. The L1_BBx_POINTER register automatically increments after each access. A BLT operation generates many accesses to the BLT buffer to complete a BLT transfer. At the end of the BLT operation the graphics pipeline generates a signal to reload the L1_BBx_POINTER register with the L1_BBx_BASE register. This allows the BLT buffer to be used over and over again with a minimum of software overhead.

See Section 4.4 “Graphics Pipeline” on page 125 on programming the graphics pipeline to generate a BLT.

**Table 4-3. Scratchpad Organization**

| 2 KB Configuration | | 3 KB Configuration | | 4 KB Configuration | | |
|---|---|---|---|---|---|---|
| Offset | Size | Offset | Size | Offset | Size | Description |
| GX_BASE + 0EE0h | 288 bytes | GX_BASE + 0EE0h | 288 bytes | GX_BASE + 0EE0h | 288 bytes | SMM scratchpad |
| GX_BASE + 0E60h | 128 bytes | GX_BASE + 0E60h | 128 bytes | GX_BASE + 0E60h | 128 bytes | Driver scratchpad |
| GX_BASE + 0800h | 816 bytes | GX_BASE + 0400h | 1328 bytes | GX_BASE + 0h | 1840 bytes | BLT Buffer 0 |
| GX_BASE + 0B30h | 816 bytes | GX_BASE + 0930h | 1328 bytes | GX_BASE + 730h | 1840 bytes | BLT Buffer 1 |

## Integrated Functions (Continued)

**Table 4-4. L1 Cache BitBLT Register Summary**

| Mnemonic Name | Function |
|---|---|
| L1_BB0_BASE<br>L1 Cache BitBLT 0 Base Address | Contains the L1 set 0 address to the first byte of BLT Buffer 0. |
| L1_BB0_POINTER<br>L1 Cache BitBLT 0 Pointer | Contains the L1 set 0 address offset to the current line of BLT Buffer 0. |
| L1_BB1_BASE<br>L1 Cache BitBLT 1 Base Address | Contains the L1 set 0 address to the first byte of BLT Buffer 1. |
| L1_BB1_POINTER<br>L1 Cache BitBLT 1 Pointer | Contains the L1 set 0 address offset to the current line of BLT Buffer 1. |

**Notes:** 1. For information on accessing these registers, refer to Section 4.1.6 “CPU_READ/CPU_WRITE Instructions” on page 102.

2. The L1 cache locations accessed by the BitBLT registers must be enabled as scratchpad RAM prior to use.

**Table 4-5. L1 Cache BitBLT Registers**

| Bit | Name | Description |
|---|---|---|
| | | **L1_BB0_BASE Register (R/W)** **Default Value = None** |
| 15:12 | RSVD | **Reserved:** Set to 0. |
| 11:4 | INDEX | **BitBLT 0 Base Index:** The index to the starting cache line of set 0 in L1 of BLT Buffer 0. |
| 3:0 | BYTE | **BitBLT 0 Starting Byte:** Determines which byte of the starting line is the beginning of BLT Buffer 0. |
| | | **L1_BB0_POINTER Register (R/W)** **Default Value = None** |
| 15:12 | RSVD | **Reserved:** Set to 0. |
| 11:4 | INDEX | **BitBLT 0 Pointer Index:** The index to the current cache line of set 0 in L1 of BLT Buffer 0. |
| 3:0 | RSVD | **Reserved:** Set to 0. |
| | | **L1_BB1_Base Register (R/W)** **Default Value = None** |
| 15:12 | RSVD | **Reserved:** Set to 0. |
| 11:4 | INDEX | **BitBLT 1 Base Index:** The index to the starting cache line of set 0 in L1 of BLT Buffer 1. |
| 3:0 | BYTE | **BitBLT 1 Starting Byte:** Determines which byte of the starting line is the beginning of BLT Buffer 1. |
| | | **L1_BB1_POINTER Register (R/W)** **Default Value = None** |
| 15:12 | RSVD | **Reserved:** Set to 0. |
| 11:4 | INDEX | **BitBLT 1 Pointer Index:** The index to the current cache line of set 0 in L1 of BLT Buffer 1. |
| 3:0 | RSVD | **Reserved:** Set to 0. |

## Integrated Functions (Continued)

### 4.1.5 Display Driver Instructions

While the majority of the GXLV's integrated function interface is memory mapped, a few integrated function registers are accessed via four GXLV specific instructions. Table 4-6 shows these instructions.

Adding CPU instructions does not create a compatibility problem for applications that may depend on receiving illegal opcode traps. The solution is to make these instructions generate an illegal opcode trap unless a compatibility bit is explicitly set. The GXLV processor uses the scratchpad size field (bits [3:2] in GCR, Index B8h) to enable or disable all of the graphics instructions.

**Note:** If the scratchpad size bits are zero, meaning that none of the cache is defined as scratchpad, then hardware will assume that the graphics controller is not being used and the graphics instructions will be disabled.

Any other scratchpad size will enable all of the new instructions. Note that the base address of the memory map in the GCR register can still be set up to allow access to the memory controller registers

### 4.1.6 CPU_READ/CPU_WRITE Instructions

The GXLV processor has several internal registers that control the BLT buffer and power management circuitry in the dedicated cache subsystem. To avoid adding additional instructions to read and write these registers, the GXLV processor has a general mechanism to access internal CPU registers with reasonable performance. The GXLV processor has two special instructions to read and write CPU registers: CPU_READ and CPU_WRITE. Both instructions fetch a 32-bit register address from *EBX* as shown in Table 4-6 and Table 4-7. CPU_WRITE uses *EAX* for the source data, and CPU_READ uses *EAX* as the destination. Both instructions always transfer 32 bits of data.

These instructions work by initiating a special I/O transaction where the high address bit is set. This provides a very large address space for internal CPU registers.

The BLT buffer base registers define the starting physical addresses of the BLT buffers located within the dedicated L1 cache. The dedicated cache can be configured for up to 4 KB, so 12 address bits are required for each base address.

.

**Table 4-6. Display Driver Instructions**

| Syntax | Opcode | Registers | Description |
|---|---|---|---|
| BB0_RESET | 0F3A | N/A | Reset the BLT Buffer 0 pointer to the base. |
| BB1_RESET | 0F3B | N/A | Reset the BLT Buffer 1 pointer to the base. |
| CPU_WRITE | 0F3C | EBX = Register Address (see Table 4-7)<br>EAX = Source Data | Write data to CPU internal register. |
| CPU_READ | 0F3D | EBX = Register Address (see Table 4-7)<br>EAX = Destination Data | Read data from CPU internal register. |

**Table 4-7. Address Map for CPU-Access Registers**

| Register | EBX Address | Description |
|---|---|---|
| L1_BB0_BASE | FFFFFF0Ch | BLT Buffer 0 base address (see Table 4-5 on page 101). |
| L1_BB1_BASE | FFFFFF1Ch | BLT Buffer 1 base address (see Table 4-5 on page 101). |
| L1_BB0_POINTER | FFFFFF2Ch | BLT Buffer 0 pointer address (see Table 4-5 on page 101). |
| L1_BB1_POINTER | FFFFFF3Ch | BLT Buffer 1 pointer address (see Table 4-5 on page 101). |
| PM_BASE | FFFFFF6Ch | Power management base address (see Table 5-3 on page 183). |
| PM_MASK | FFFFFF7Ch | Power management address mask (see Table 5-3 on page 183). |

## 4.2 INTERNAL BUS INTERFACE UNIT

The GXLV processor's internal bus interface unit provides control and interface functions to the C-Bus and X-Bus. The functions on C-Bus include: processor core, FPU, graphics pipeline, and L1 cache. The functions on X-Bus include: PCI controller, display controller, memory controller, and graphics accelerator. It provides attribute control for several sections of memory, and plays an important part in the Virtual VGA function.

The internal bus interface unit performs functions which previously required the external pins IGNNE# and A20M#.

The internal bus interface unit provides configuration control for up to 20 different regions within system memory. This includes a top-of-memory register and 19 configurable memory regions in the address space between 640 KB and 1 MB. Each region has separate control for read access, write access, cacheability, and external PCI master access.

In support of VGA emulation, three of the memory regions are configurable for use by the graphics pipeline and three I/O ranges can be programmed to generate SMIs.

### 4.2.1 FPU Error Support

The FERR# (floating point error) and IGNNE# (ignore numeric error) pins of the 486 microprocessor have been replaced with an IRQ13 (interrupt request 13) pin. In DOS systems, FPU errors are reported by the external vector 13. Emulation of this mode of operation is specified by clearing the NE bit (bit 5) in the CR0 register. If the NE bit is active, the IRQ13 output of the GXLV processor is always driven inactive. If the NE bit is cleared, the GXLV processor drives IRQ13 active when the ES bit (bit 7) in the FPU Status Register is set high. Software must respond to this interrupt with an OUT instruction containing an 8-bit operand to F0h or F1h. When the OUT cycle occurs, the IRQ13 pin is driven inactive and the FPU starts ignoring numeric errors. When the ES bit is cleared, the FPU resumes monitoring numeric errors.

### 4.2.2 A20M Support

The GXLV processor provides an A20M bit in the BC_XMAP_1 Register (GX_BASE+ 8004h[21]) to replace the A20M# pin on the 486 microprocessor. When the A20M bit is set high, all non-SMI accesses will have address bit 20 forced to zero. External hardware must do an SMI trap on I/O locations that toggle the A20M# pin. The SMI software can then change the A20M bit as desired.

This will maintain compatibility with software that depends on wrapping the address at bit 20.

### 4.2.3 SMI Generation

The Internal Bus Interface Unit can generate SMI interrupts whenever an I/O cycle is in the VGA address ranges of 3B0h to 3BFh, 3C0h to 3CFh and/or 3D0h to 3DFh. If an external VGA card is present, the Internal Bus Interface reset values will not generate an interrupt on VGA accesses. (Refer to Section 4.6.3 "VGA Configuration Registers" on page 162 for instructions on how to configure the registers to enable the SMI interrupt.)

### 4.2.4 640 KB to 1 MB Region

There are 19 configurable memory regions located between 640 KB and 1 MB. Three of the regions, A0000h to AFFFFh, B0000h to B7FFFh, and B8000h to BFFFFh, are typically used by the graphics subsystem in VGA emulation mode. Each of the these regions has a VGA control bit that can cause the graphics pipeline to handle accesses to that section of memory (see Table 4-37 on page 163). The area between C0000h and FFFFFh is divided into 16 KB segments to form the remaining 16 regions. All 19 regions have four control bits to allow any combination of read-access, write-access, cache, and external PCI Bus Master access capabilities (see Table 4-10 on page 106).

## Integrated Functions (Continued)

### 4.2.5 Internal Bus Interface Unit Registers

The Internal Bus Interface Unit maps 100h bytes starting at GX_BASE+8000h. However only 16 bytes (four 32-bit registers) are defined. Refer to Section 4.1.2 “Control Registers” on page 99 for instructions on accessing these registers.

Table 4-8 summarizes the four 32-bit registers contained in the Internal Bus Interface Unit and Table 4-9 gives the register/bit formats.

**Table 4-8. Internal Bus Interface Unit Register Summary**

| GX_BASE+ Memory Offset | Type | Name/Function | Default Value |
|---|---|---|---|
| 8000h-8003h | R/W | **BC_DRAM_TOP**<br>Top of DRAM — Contains the highest available address of system memory not including the memory that is set aside for graphics memory, which corresponds to 1 GB of memory. The largest possible value for the register is 3FFFFFFFh. | 3FFFFFFFh |
| 8004h-8007h | R/W | **BC_XMAP_1**<br>Memory X-Bus Map Register 1 (A and B Region Control) — Contains the region control of the A and B regions and the SMI controls required for VGA emulation. PCI access to internal registers and the A20M function are also controlled by this register. | 00000000h |
| 8008h-800Bh | R/W | **BC_XMAP_2**<br>Memory X-Bus Map Register 2 (C and D Region Control) — Contains region control fields for eight regions in the address range C0h through DCh. | 00000000h |
| 800Ch-800Fh | R/W | **BC_XMAP_3**<br>Memory X-Bus Map Register 3 (E and F Region Control) — Contains the region control fields for memory regions in the address range E0h through FCh. | 00000000h |

**Table 4-9. Internal Bus Interface Unit Registers**

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+8000h-8003h** | | **BC_DRAM_TOP Register (R/W)** **Default Value = 3FFFFFFFh** |
| 31:28 | RSVD | **Reserved:** Set to 0. |
| 27:17 | TOP OF DRAM | **Top of DRAM:**<br>000h = Minimum top or 0001FFFFh (128 KB)<br>7FFh = Maximum top or 0FFFFFFFh (256 MB) |
| 16:0 | RSVD | **Reserved:** Set to 1. |
| **GX_BASE+8004h-8007h** | | **BC_XMAP_1 Register (R/W)** **Default Value = 00000000h** |
| 31:29 | RSVD | **Reserved:** Set to 0. |
| 28 | GEB8 | **Graphics Enable for B8 Region:** Allow memory R/W operations for address range B8000h to BFFFFh be directed to the graphics pipeline: 0 = Disable; 1 = Enable. If enabled, the GEB8 region is always non-cacheable. In the region control field (B8) the cache enable bit (bit 2) is ignored.<br>(Used for VGA emulation.) |
| 27:24 | B8 | **B8 Region:** Region control field for address range B8000h to BFFFFh.<br>**Note:** Refer to Table 4-10 on page 106 for decode. |
| 23 | RSVD | **Reserved:** Set to 0. |
| 22 | PRAE | **PCI Register Access Enable:** Allow PCI Slave to access internal registers on the X-Bus:<br>0 = Disable; 1 = Enable. |
| 21 | A20M | **Address Bit 20 Mask:** Address bit 20 is always forced to a zero except for SMI accesses:<br>0 = Disable; 1 = Enable. |
| 20 | GEB0 | **Graphics Enable for B0 Region:** Allow memory R/W operations for address range B8000h to BFFFFh be directed to the graphics pipeline: 0 = Disable; 1 = Enable. If enabled, the GEB0 region is always non-cacheable. In the region control field (B0) the cache enable bit (bit 2) is ignored.<br>(Used for VGA emulation.) |
| 19:16 | B0 | **B0 Region:** Region control field for address range B0000h to B7FFFh.<br>**Note:** Refer to Table 4-10 on page 106 for decode. |
| 15 | SMID | **SMID:** All I/O accesses for address range 3D0h to 3DFh generate an SMI: 0 = Disable; 1 = Enable.<br>(Used for VGA virtualization.) |

## Integrated Functions (Continued)

### Table 4-9. Internal Bus Interface Unit Registers

| Bit | Name | Description |
|---|---|---|
| 14 | SMIC | **SMIC:** All I/O accesses for address range 3C0h to 3CFh generate an SMI: 0 = Disable; 1 = Enable. (Used for VGA virtualization.) |
| 13 | SMIB | **SMIB:** All I/O accesses for address range 3B0h to 3BFh generate an SMI: 0 = Disable; 1 = Enable (Used for VGA virtualization.) |
| 12:8 | RSVD | **Reserved:** Set to 0. |
| 7 | XPD | **X-Bus Pipeline:** The address for the next cycle can be driven on the X-Bus before the completion of the data phase of the current cycle.<br>0 = Enable<br>1 = Disable |
| 6 | GNWS | **X-Bus Graphics Pipe No Wait State:** Data driven on the X-Bus from the graphics pipeline:<br>0 = 1 full clock before X_DSX is asserted<br>1 = On the same clock in which X_RDY is asserted |
| 5 | XNWS | **X-Bus No Wait State:** Data driven on the X-Bus from the internal bus interface unit:<br>0 = 1 full clock before X_DSX is asserted<br>1 = On the same clock in which X_RDY is asserted |
| 4 | GEA | **Graphics Enable for A Region:** Allow memory R/W operations for address range B8000h to BFFFFh be directed to the graphics pipeline: 0 = Disable; 1 = Enable. If enabled, the GEA region is always non-cacheable. In the region control field (A0) the cache enable bit (bit2) is ignored.<br>(Used for VGA emulation.) |
| 3:0 | A0 | **A0 Region:** Region control field for address range A0000h to AFFFFh.<br>**Note:** Refer to Table 4-10 on page 106 for decode. |
| **GX_BASE+8008h-800Bh** | | **BC_XMAP_2 Register (R/W)** — **Default Value = 00000000h** |
| 31:28 | DC | **DC Region:** Region control field for address range DC000h to DFFFFh. |
| 27:24 | D8 | **D8 Region:** Region control field for address range D8000h to DBFFFh. |
| 23:20 | D4 | **D4 Region:** Region control field for address range D4000h to D7FFFh. |
| 19:16 | D0 | **D0 Region:** Region control field for address range D0000h to D3FFFh. |
| 15:12 | CC | **CC Region:** Region control field for address range CC000h to CFFFFh. |
| 11:8 | C8 | **C8 Region:** Region control field for address range C8000h to CBFFF. |
| 7:4 | C4 | **C4 Region:** Region control field for address range C4000h to C7FFFh. |
| 3:0 | C0 | **C0 Region:** Region control field for address range C0000h to C3FFFh. |
| **Note:** Refer to Table 4-10 on page 106 for decode. | | |
| **GX_BASE+800Ch-800Fh** | | **BC_XMAP_3 Register (R/W)** — **Default Value = 00000000h** |
| 31:28 | FC | **FC Region:** Region control field for address range FC000h to FFFFFh. |
| 27:24 | F8 | **F8 Region:** Region control field for address range F8000h to FBFFFh. |
| 23:20 | F4 | **F4 Region:** Region control field for address range F4000h to F7FFFh. |
| 19:16 | F0 | **F0 Region:** Region control field for address range F0000h to F3FFFh. |
| 15:12 | EC | **EC Region:** Region control field for address range EC000h to EFFFFh. |
| 11:8 | E8 | **E8 Region:** Region control field for address range E8000h to EBFFFh. |
| 7:4 | E4 | **E4 Region:** Region control field for address range E4000h to E7FFFh. |
| 3:0 | E0 | **E0 Region:** Region control field for address range E0000h to E3FFFh. |
| **Note:** Refer to Table 4-10 on page 106 for decode. | | |

## Integrated Functions (Continued)

### Table 4-10. Region-Control-Field Bit Definitions

| Bit Position | Function |
|---|---|
| 3 | **PCI Accessible:** The PCI slave can access this memory if this bit is set high and if the appropriate Read or Write Enable bit is also set high. |
| 2 | **Cache Enable:** Caching this region of memory is inhibited if this bit is cleared. |
| 1 | **Write Enable:** Write operations to this region of memory are allowed if this bit is set high. If this bit is cleared, then write operations in this region are directed to the PCI master. |
| 0 | **Read Enable:** Read operations to this region of memory are allowed if this bit is set high. If this bit is cleared then read operations in this region are directed to the PCI master. |
| **Note:** If Cache Enable = 1 and Write Enable = 1, the Write Enable determination occurs after the data has passed the cache. Since the cache does write update, write data will change the cache if the address is cached. If a read then occurs to that address, the data will come from the written data that is in the cache even though the address is not writable. If this must be avoided then do not make the region cacheable. | |

## Integrated Functions (Continued)

### 4.3 MEMORY CONTROLLER

The memory controller arbitrates requests from the X-Bus (processor and PCI), display controller, and graphics pipeline.

The GXLV processor supports LVTTL (low voltage TTL) technology. LVTTL technology allows the SDRAM interface of the memory controller to run at frequencies up to 100 MHz.

The SDRAM clock is a function of the core clock. The SDRAM bus can be run at speeds that range between 66 MHz and 100 MHz. The core clock can be divided down from 2 to 5 in half clock increments to generate the SDRAM clock. SDRAM frequencies between 79 MHz and 100 MHz are only supported for certain types of closed systems and strict design rules must be adhered to. For further details, contact your local National Semiconductor technical support representative.

A basic block diagram of the memory controller is shown in Figure 4-3.

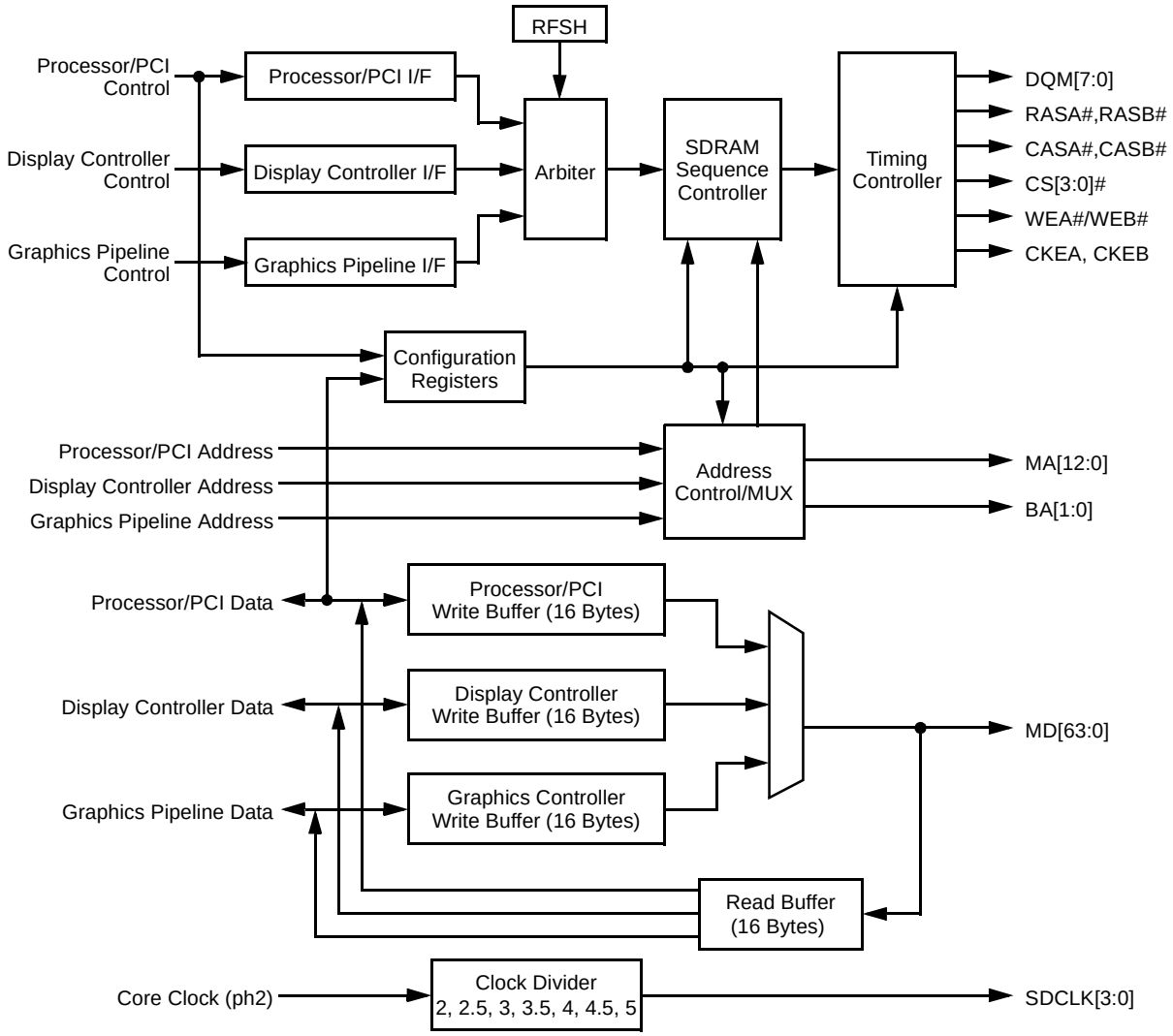



**Figure 4-3. Memory Controller Block Diagram**

## Integrated Functions (Continued)

### 4.3.1 Memory Array Configuration

The memory controller supports up to four 64-bit SDRAM banks, with maximum of eight physical devices per bank. Banks 0:1 and 2:3 must be identical configurations. Two 168-pin unbuffered SDRAM modules (DIMM) satisfy these requirements Though the following discussion is DIMM centric, DIMMs are not a system requirement. Each DIMM receives a unique set of RAS, CAS, WE, and CKE lines. Each DIMM can have one or two 64-bit DIMM banks. Each DIMM bank is selected by a unique chip select (CS). There are four chip select signals to choose between a total of four DIMM banks. Each DIMM bank also receives a unique SDCLK. Each DIMM bank can have two or four component banks. Component bank selection is done through the bank address (BA) lines.

For example, 16-Mbit SDRAM have two component banks and 64-Mbit SDRAM have two or four component banks. For single DIMM bank modules, the memory controller can support two DIMMS with a maximum of eight component banks. For dual DIMM bank modules, the memory controller can support two DIMMs with a maximum of 16 component banks. Up to 16 banks can be open at the same time. Refer to the SDRAM manufacturer's specification for more information on component banks.

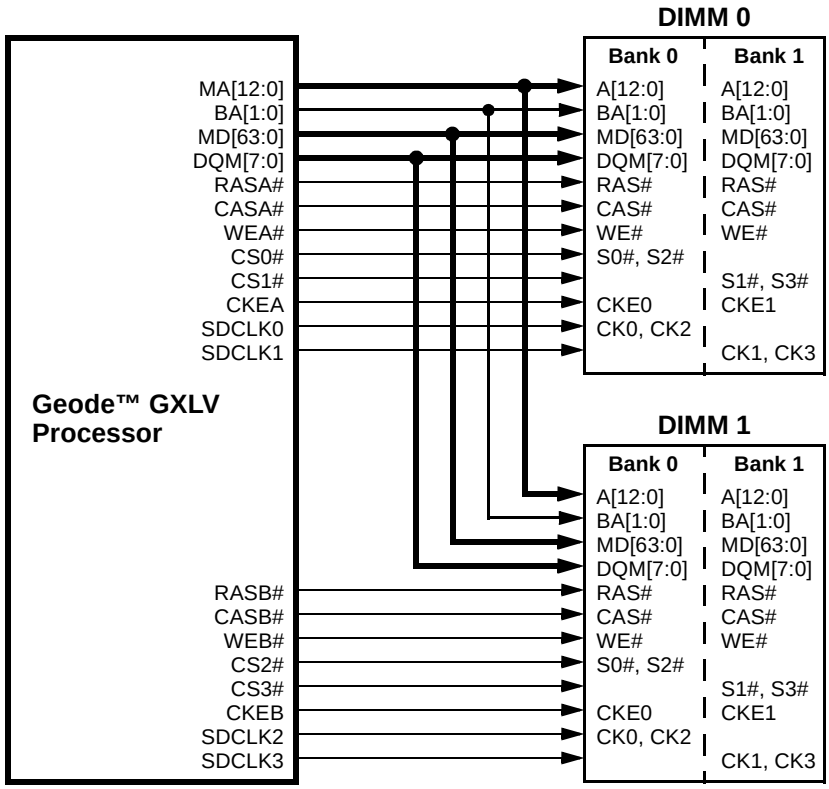



**Figure 4-4. Memory Array Configuration**

## Integrated Functions (Continued)

### 4.3.2 Memory Organizations

The memory controller supports JEDEC standard synchronous DRAMs in 16 Mbit and 64 Mbit configurations. Supported configurations are shown in Table 4-11. Note that when using x4 SDRAM, there are 16 devices per bank. The GXLV supports a total of 32 devices. There are only two banks total when x4 devices are used.

**Table 4-11. Synchronous DRAM Configurations**

| Depth | Organization | Row Address | Column Address | Bank Address | Total # of Address bits |
|---|---|---|---|---|---|
| 1 | 1 Mx16 | A10-A0 | A7-A0 | BA0 | 20 |
| 2 | 2 Mx8 | A10-A0 | A8-A0 | BA0 | 21 |
| | 2 Mx32 | A10-A0 | A7-A0 | BA1-BA0 | 21 |
| | 2 Mx32 | A10-A0 | A8-A0 | BA0 | 21 |
| | 2 Mx32 | A11-A0 | A6-A0 | BA1-BA0 | 21 |
| | 2 Mx32 | A12-A0 | A6-A0 | BA0 | 21 |
| 4 | 4 Mx4 | A10-A0 | A9-A0 | BA0 | 22 |
| | 4 Mx16 | A11-A0 | A7-A0 | BA1-BA0 | 22 |
| | 4 Mx16 | A12-A0 | A7-A0 | BA0 | 22 |
| | 4 Mx16 | A10-A0 | A9-A0 | BA0 | 22 |
| 8 | 8 Mx8 | A11-A0 | A8-A0 | BA1-BA0 | 23 |
| | 8 Mx8 | A12-A0 | A8-A0 | BA0 | 23 |
| | 8 Mx32 | A11-A0 | A8-A0 | BA1-BA0 | 23 |
| | 8 Mx32 | A12-A0 | A7-A0 | BA1-BA0 | 23 |
| 16 | 16 Mx4 | A11-A0 | A9-A0 | BA1-BA0 | 24 |
| | 16 Mx4 | A12-A0 | A9-A0 | BA0 | 24 |
| | 16 Mx16 | A12-A0 | A8-A0 | BA1-BA0 | 24 |
| | 16 Mx16 | A11-A0 | A9-A0 | BA1-BA0 | 24 |
| 32 | 32 Mx8 | A12-A0 | A9-A0 | BA1-BA0 | 25 |
| 64 | 64 Mx4 | A12-A0 | A9-A0,A11 | BA1-BA0 | 26 |

## Integrated Functions (Continued)

### 4.3.3 SDRAM Commands

This subsection discusses the SDRAM commands supported by the memory controller. Table 4-12 summarizes these commands followed by detailed operational information regarding each command. Refer to SDRAM device specifications available from SDRAM manufacturer's for more detailed information.

**Table 4-12. Basic Command Truth Table**

| Name | Command | CS | RAS | CAS | WE |
|---|---|---|---|---|---|
| MRS | Mode Register Set | L | L | L | L |
| PRE | Bank Precharge | L | L | H | L |
| ACT | Bank activate/row-address entry | L | L | H | H |
| WRT | Column address entry/Write operation | L | H | L | L |
| READ | Column address entry/Read operation | L | H | L | H |
| DESL | Control input inhibit/ No operation | H | X | X | X |
| RFSH* | CBR Refresh or Auto Refresh | L | L | L | H |

**Note:** *This command is CBR (CAS-before-RAS) refresh when CKE is high and self refresh when CKE is low.

**MRS** — The Mode Register command defines the specific mode of operation of the SDRAM. This definition includes the selection of burst length, burst type, and CAS latency. CAS latency is the delay, in clock cycles, between the registration of a read command and the availability of the first piece of output data.

The burst length is programmed by address bits MA[2:0], the burst type by address bit MA3 and the CAS latency by address bits MA[6:4].

The memory controller only supports a burst length of two and burst type of interleave.

The field value on MA[12:0] and BA[1:0] during the MRS cycle are as shown in Table 4-13.

**PRE** — The precharge command is used to deactivate the open row in a particular component bank or the open row in all (2 or 4, device dependent) component banks. Address pin MA10 determines whether one or all component banks are to be precharged. In the case where only one component bank is to be precharged, BA[1:0] selects which bank. Once a component bank has been precharged, it is in the Idle state and must be activated prior to any read or write commands.

**Table 4-13. Address Line Programming during MRS Cycles**

| BA[1:0] | MA[12:7] | MA[6:4] | MA3 | MA[2:0] |
|---|---|---|---|---|
| 00 | 000000 | CAS Latency:<br>000 = Reserved<br>010 = 2 CLK<br>100 = 4 CLK<br>110 = 6 CLK<br>001 = 1 CLK<br>011 = 3 CLK<br>101 = 5 CLK<br>111 = 7 CLK | 1<br>Burst type is always interleave. | 001<br>Burst length is always 2.<br>128-bit transfer. |

## Integrated Functions (Continued)

**ACT** — The activate command is used to open a row in a particular bank for a subsequent access. The value on the BA lines selects the bank, and the address on the MA lines selects the row. This row remains open for accesses until a precharge command is issued to that bank. A precharge command must be issued before opening a different row in the same bank.

**WRT** — The write command is used to initiate a burst write access to an active row. The value on the BA lines select the component bank, and the address provided by the MA lines select the starting column location. The memory controller does not perform auto precharge during write operations. This leaves the page open for subsequent accesses. Data appearing on the MD lines is written to the DQM logic level appearing coincident with the data. If the DQM signal is registered low, the corresponding data will be written to memory. If the DQM is driven high, the corresponding data will be ignored, and a write will not be executed to that location.

**READ** — The read command is used to initiate a burst read access to an active row. The value on the BA lines select the component bank, and the address provided by the MA lines select the starting column location. The memory controller does not perform auto precharge during read operations. Valid data-out from the starting column address is available following the CAS latency after the read command. The DQM signals are asserted low during read operations.

**RFSH** — Auto refresh is used during normal operation and is analogous to the CAS-before-RAS (CBR) refresh in conventional DRAMs. During auto refresh the address bits are “don’t care”. The memory controller precharges all banks prior to an auto refresh cycle. Auto refresh cycles are issued approximately 15 µs apart.

The self refresh command is used to retain data in the SDRAMs even when the rest of the system is powered down. The self refresh command is similar to an auto refresh command except CKE is disabled (low). The memory controller issues a self refresh command during 3V Suspend mode when all the internal clocks are stopped.

#### 4.3.3.1 SDRAM Initialization Sequence

After the clocks have started and stabilized, the memory controller SDRAM initialization sequence begins:

1) Precharge all component banks
2) Perform eight refresh cycles
3) Perform an MRS cycle
4) Perform eight refresh cycles

This sequence is compatible with the majority of SDRAMs available from the various vendors.

## Integrated Functions (Continued)

### 4.3.4 Memory Controller Register Description

The Memory Controller maps 100h locations starting at GX_BASE+8400h. Refer to Section 4.1.2 “Control Registers” on page 99 for instructions on accessing these registers.

Table 4-14 summarizes the 32-bit registers contained in the memory controller. Table 4-15 gives detailed register/bit formats.

**Table 4-14. Memory Controller Register Summary**

| GX_BASE+ Memory Offset | Type | Name/Function | Default Value |
|---|---|---|---|
| 8400h-8403h | R/W | **MC_MEM_CNTRL1**<br>Memory Controller Control Register 1: Memory controller configuration information (e.g., refresh interval, SDCLK ratio, etc.). BIOS must program this register based on the processor frequency and desired SDCLK divide ratio. | 248C0040h |
| 8404h-8407h | R/W | **MC_MEM_CNTRL2**<br>Memory Controller Control Register 2: Memory controller configuration information to control SDCLK. BIOS must program this register based on the processor frequency and the SDCLK divide ratio. | 00000801h |
| 8408h-840Bh | R/W | **MC_BANK_CFG**<br>Memory Controller Bank Configuration: Contains the configuration information for the each of the four SDRAM banks in the memory array. BIOS must program this register during boot by running an autosizing routine on the memory. | 41104110h |
| 840Ch-840Fh | R/W | **MC_SYNC_TIM1**<br>Memory Controller Synchronous Timing Register 1: SDRAM memory timing information - This register controls the memory timing of all four banks of DRAM. BIOS must program this register based on the processor frequency and the SDCLK divide ratio. | 2A733225h |
| 8414h-8417h | R/W | **MC_GBASE_ADD**<br>Memory Controller Graphics Base Address Register: This register sets the graphics memory base address, which is programmable on 512 KB boundaries. The display controller and the graphics pipeline generate a 20-bit DWORD offset that is added to the graphics memory base address to form the physical memory address. Typically, the graphics memory region is located at the top of physical memory. | 00000000h |
| 8418h-841Bh | R/W | **MC_DR_ADD**<br>Memory Controller Dirty RAM Address Register: This register is used to set the Dirty RAM address index for processor diagnostic access. This register should be initialized before accessing the MC_DR_ACC register | 00000000h |
| 841Ch-841Fh | R/W | **MC_DR_ACC**<br>Memory Controller Dirty RAM Access Register: This register is used to access the Dirty RAM. A read/write to this register will access the Dirty RAM at the address specified in the MC_DR_ADD register. | 0000000xh |

**Table 4-15. Memory Controller Registers**

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+ 8400h-8403h** | | **MC_MEM_CNTRL1 (R/W)** **Default Value = 248C0040h** |
| 31:29 | MDHDCTL | **MD High Drive Control:** Controls the drive strength and slew rate of the memory data bus (MD[63:0]) during a write cycle:<br>000 = TRI-STATE<br>001 = Smallest drive strength<br>010 -110 = Represents gradual drive strength increase<br>111 = Highest drive strength |
| 28:26 | MABAHDCTL | **MA/BA High Drive Control:** Controls the drive strength and slew rate of the memory address bus including the memory bank address bus (MA[12:0] and BA[1:0]):<br>000 = TRI-STATE<br>001 = Smallest drive strength<br>010 -110 = Represents gradual drive strength increase<br>111 = Highest drive strength |
| 25:23 | MEMHDCTL | **Control High Drive/Slew Control:** Controls the drive strength and slew rate of the memory control signals (CASA#, CASB#, RASA#, RASB#, CKEA, CKEB, WEA#, WEA#, DQM[7:0], and CS[3:0]#):<br>000 = TRI-STATE<br>001 = Smallest drive strength<br>010 -110 = Represents gradual drive strength increase<br>111 = Highest drive strength |
| 22 | RSVD | **Reserved:** Set to 0. |
| 21 | RSVD | **Reserved:** Must be set to 0. Wait state on the X-Bus x_data during read cycles - for debug only. |
| 20:18 | SDCLKRATE | **SDRAM Clock Ratio:** Selects SDRAM clock ratio:<br>000 = Reserved 100 = ÷ 3.5<br>001 = ÷ 2 101 = ÷ 4<br>010 = ÷ 2.5 110 = ÷ 4.5<br>011 = ÷ 3 (Default) 111 = ÷ 5<br>Ratio does not take effect until the SDCLKSTRT bit (bit 17 of this register) transitions from 0 to 1. |
| 17 | SDCLKSTRT | **Start SDCLK:** Start operating SDCLK using the new ratio and shift value (selected in bits [20:18] of this register): 0 = Clear; 1 = Enable.<br>This bit must transition from zero (written to zero) to one (written to one) in order to start SDCLK or to change the shift value. |
| 16:8 | RFSHRATE | **Refresh Interval:** This field determines the number of processor core clocks multiplied by 64 between refresh cycles to the DRAM. By default, the refresh interval is 00h. Refresh is turned off by default. |
| 7:6 | RFSHSTAG | **Refresh Staggering:** This field determines number of clocks between the RFSH commands to each of the four banks during refresh cycles:<br>00 = 0 SDRAM clocks 10 = 2 SDRAM clocks<br>01 = 1 SDRAM clocks (Default) 11 = 4 SDRAM clocks<br>Staggering is used to help reduce power spikes during refresh by refreshing one bank at a time. If only one bank is installed, this field must be set to 00. |
| 5 | 2CLKADDR | **Two Clock Address Setup:** Assert memory address for one extra clock before CS# is asserted: 0 = Disable; 1 = Enable.<br>This can be used to compensate for address setup at high frequencies and/or high loads. |
| 4 | RFSHTST | **Test Refresh:** This bit, when set high, generates a refresh request. This bit is only used for testing purposes. |
| 3 | XBUSARB | **X-Bus Round Robin:** When enabled, processor, graphics pipeline and non-critical display controller requests are arbitrated at the same priority level. When disabled, processor requests are arbitrated at a higher priority level. High priority display controller requests always have the highest arbitration priority: 0 = Enable; 1 = Disable. |
| 2 | SMM_MAP | **SMM Region Mapping:** Map the SMM memory region at GX_BASE+400000 to physical address A0000 to BFFFF in SDRAM: 0 = Disable; 1 = Enable. |
| 1 | RSVD | **Reserved:** Set to 0. |
| 0 | SDRAMPRG | **Program SDRAM:** When this bit is set the memory controller will program the SDRAM MRS register using LTMODE in MC_SYNC_TIM1.<br>This bit must transition from zero (written to zero) to one (written to one) in order to program the SDRAM devices. |

## Integrated Functions (Continued)

### Table 4-15. Memory Controller Registers (Continued)

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+8404h-8407h** | | **MC_MEM_CNTRL2 (R/W)** **Default Value = 00000801h** |
| 31:14 | RSVD | **Reserved:** Set to 0. |
| 13:11 | SDCLKHDCTL | **SDCLK High Drive/Slew Control:** Controls the high drive and slew rate of SDCLK[3:0] and SDCLK_OUT.<br>000 = Highest drive strength (no braking applied in the pads)<br>001 = Smallest drive strength<br>010 -110 = Represent gradual drive strength increase<br>111 = Highest drive strength |
| 10 | SDCLKOMSK# | **Enable SDCLK_OUT:** Turn on the output. 0 = Enabled; 1 = Disabled. |
| 9 | SDCLK3MSK# | **Enable SDCLK3:** Turn on the output. 0 = Enabled; 1 = Disabled. |
| 8 | SDCLK2MSK# | **Enable SDCLK2:** Turn on the output. 0 = Enabled; 1 = Disabled. |
| 7 | SDCLK1MSK# | **Enable SDCLK1:** Turn on the output. 0 = Enabled; 1 = Disabled. |
| 6 | SDCLK0MSK# | **Enable SDCLK0:** Turn on the output. 0 = Enabled; 1 = Disabled. |
| 5:3 | SHFTSDCLK | **Shift SDCLK:** This function allows shifting SDCLK to meet SDRAM setup and hold time requirements. The shift function will not take effect until the SDCLKSTRT bit (bit 17 of MC_MEM_CNTRL1) transitions from 0 to 1:<br>000 = No shift<br>001 = Shift 0.5 core clock<br>010 = Shift 1 core clock<br>011 = Shift 1.5 core clock<br>100 = Shift 2 core clocks<br>101 = Shift 2.5 core clocks<br>110 = Shift 3 core clocks<br>111 = Reserved<br>**Note:** Refer to Figure 4-10 on page 124 for an example of SDCLK shifting. |
| 2 | RSVD | **Reserved:** Set to 0. |
| 1 | RD | **Read Data Phase:** Selects if read data is latched one or two core clock after the rising edge of SDCLK: 0 = 1 core clock; 1 = 2 core clocks. |
| 0 | FSTRDMSK | **Fast Read Mask:** Do not allow core reads to bypass the request FIFO: 0 = Disable; 1 = Enable. |
| **GX_BASE+8408h-840Bh** | | **MC_BANK_CFG (R/W)** **Default Value = 41104110h** |
| 31 | RSVD | **Reserved:** Set to 0. |
| 30 | DIMM1_ MOD_BNK | **DIMM1 Module Banks (Banks 2 and 3):** Selects the number of module banks installed per DIMM for DIMM1:<br>0 = 1 Module bank (Bank 2 only)<br>1 = 2 Module banks (Bank 2 and 3) |
| 29 | RSVD | **Reserved:** Set to 0. |
| 28 | DIMM1_ COMP_BNK | **DIMM1 Component Banks (Banks 2 and 3):** Selects the number of component banks per module bank for DIMM1:<br>0 = 2 Component banks<br>1 = 4 Component banks<br>Banks 2 and 3 must have the same number of component banks. |
| 27 | RSVD | **Reserved:** Set to 0. |
| 26:24 | DIMM1_SZ | **DIMM1 Size (Banks 2 and 3):** Selects the size of DIMM1:<br>000 = 4 MB 010 = 16 MB 100 = 64 MB 110 = 256 MB<br>001 = 8 MB 011 = 32 MB 101 = 128 MB 111 = 512 MB (not supported)<br>This size is the total of both banks 2 and 3. Also, banks 2 and 3 must be the same size. |
| 23 | RSVD | **Reserved:** Set to 0. |
| 22:20 | DIMM1_PG_SZ | **DIMM1 Page Size (Banks 2 and 3):** Selects the page size of DIMM1:<br>000 = 1 KB 010 = 4 KB 1xx = 16 KB<br>001 = 2 KB 011 = 8 KB 111 = DIMM1 not installed<br>Both banks 2 and 3 must have the same page size. When DIMM1 (neither bank 2 or 3) is not installed, program all other DIMM1 fields to 0. |
| 19:15 | RSVD | **Reserved:** Set to 0. |
| 14 | DIMM0_ MOD_BNK | **DIMM0 Module Banks (Banks 0 and 1):** Selects number of module banks installed per DIMM for DIMM0:<br>0 = 1 Module bank (Bank 0 only)<br>1 = 2 Module banks (Bank 0 and 1) |

## Integrated Functions (Continued)

### Table 4-15. Memory Controller Registers (Continued)

| Bit | Name | Description |
|---|---|---|
| 13 | RSVD | **Reserved:** Set to 0. |
| 12 | DIMM0_ COMP_BNK | **DIMM0 Component Banks (Banks 0 and 1):** Selects the number of component banks per module bank for DIMM0:<br>0 = 2 Component banks<br>1 = 4 Component banks<br>Banks 0 and 1 must have the same number of component banks. |
| 11 | RSVD | **Reserved:** Set to 0. |
| 10:8 | DIMM0_SZ | **DIMM0 Size (Banks 0 and 1):** Selects the size of DIMM1:<br>000 = 4 MB 010 = 16 MB 100 = 64 MB 110 = 256 MB<br>001 = 8 MB 011 = 32 MB 101 = 128 MB 111 = 512 MB (not supported)<br>This size is the total of both banks 0 and 1. Also, banks 0 and 1 must be the same size. |
| 7 | RSVD | **Reserved:** Set to 0. |
| 6:4 | DIMM0_PG_SZ | **DIMM0 Page Size (Banks 0 and 1):** Selects the page size of DIMM0:<br>000 = 1 KB 010 = 4 KB 1xx = 16 KB<br>001 = 2 KB 011 = 8 KB 111 = DIMM0 not installed<br>Both banks 0 and 1 must have the same page size. When DIMM0 (neither bank 0 or 1) is not installed, program all other DIMM0 fields to 0. |
| 3:0 | RSVD | **Reserved:** Set to 0. |
| **GX_BASE+840Ch-840Fh** | **MC_SYNC_TIM1 (R/W)** | **Default Value = 2A733225h** |
| 31 | RSVD | **Reserved:** Set to 0. |
| 30:28 | LTMODE | **CAS Latency (LTMODE):** CAS latency is the delay, in SDRAM clock cycles, between the registration of a read command and the availability of the first piece of output data. This parameter significantly affects system performance. Optimal setting should be used. If DIMMs are used BIOS can interrogate EEPROM across the $I^2C$ interface to determine this value:<br>000 = Reserved 010 = 2 CLK 100 = 4 CLK 110 = 6 CLK<br>001 = Reserved 011 = 3 CLK 101 = 5 CLK 111 = 7 CLK<br>This field will not take effect until SDRAMPRG (bit 0 of MC_MEM_CNTRL1) transitions from 0 to 1. |
| 27:24 | RC | **RFSH to RFSH/ACT Command Period (tRC):** Minimum number of SDRAM clock between RFSH and RFSH/ACT commands:<br>0000 = Reserved 0100 = 5 CLK 1000 = 9 CLK 1100 = 13 CLK<br>0001 = 2 CLK 0101 = 6 CLK 1001 = 10 CLK 1101 = 14 CLK<br>0010 = 3 CLK 0110 = 7 CLK 1010 = 11 CLK 1110 = 15 CLK<br>0011 = 4 CLK 0111 = 8 CLK 1011 = 12 CLK 1111 = 16 CLK |
| 23:20 | RAS | **ACT to PRE Command Period (tRAS):** Minimum number of SDRAM clocks between ACT and PRE commands:<br>0000 = Reserved 0100 = 5 CLK 1000 = 9 CLK 1100 = 13 CLK<br>0001 = 2 CLK 0101 = 6 CLK 1001 = 10 CLK 1101 = 14 CLK<br>0010 = 3 CLK 0110 = 7 CLK 1010 = 11 CLK 1110 = 15 CLK<br>0011 = 4 CLK 0111 = 8 CLK 1011 = 12 CLK 1111 = 16 CLK |
| 19 | RSVD | **Reserved:** Set to 0. |
| 18:16 | RP | **PRE to ACT Command Period (tRP):** Minimum number of SDRAM clocks between PRE and ACT commands:<br>000 = Reserved 010 = 2 CLK 100 = 4 CLK 110 = 6 CLK<br>001 = 1 CLK 011 = 3 CLK 101 = 5 CLK 111 = 7 CLK |
| 15 | RSVD | **Reserved:** Set to 0. |
| 14:12 | RCD | **Delay Time ACT to READ/WRT Command (tRCD):** Minimum number of SDRAM clock between ACT and READ/WRT commands. This parameter significantly affects system performance. Optimal setting should be used:<br>000 = Reserved 010 = 2 CLK 100 = 4 CLK 110 = 6 CLK<br>001 = 1 CLK 011 = 3 CLK 101 = 5 CLK 111 = 7 CLK |
| 11 | RSVD | **Reserved:** Set to 0. |
| 10:8 | RRD | **ACT(0) to ACT(1) Command Period (tRRD):** Minimum number of SDRAM clocks between ACT and ACT command to two different component banks within the same module bank. The memory controller does not perform back-to-back Activate commands to two different component banks without a READ or WRT command between them. Hence, this field should be set to 001. |

## Integrated Functions (Continued)

### Table 4-15. Memory Controller Registers (Continued)

| Bit | Name | Description |
|---|---|---|
| 7 | RSVD | **Reserved:** Set to 0. |
| 6:4 | DPL | **Data-in to PRE command period (tDPL):** Minimum number of SDRAM clocks from the time the last write datum is sampled till the bank is precharged:<br>000 = Reserved 010 = 2 CLK 100 = 4 CLK 110 = 6 CLK<br>001 = 1 CLK 011 = 3 CLK 101 = 5 CLK 111 = 7 CLK |
| 3:0 | RSVD | **Reserved:** Leave unchanged. Always returns a 101h. |
| **Note:** Refer to SDRAM device specifications available from SDRAM manufacturer's for more detailed information | | |
| **GX_BASE+8414h-8417h** | **MC_GBASE_ADD (R/W)** | **Default Value = 00000000h** |
| 31:18 | RSVD | **Reserved:** Set to 0. |
| 17 | TE | **Test Enable TEST[3:0]:**<br>0 = TEST[3:0] are driven low (normal operation)<br>1 = TEST[3:0] pins are used to output test information |
| 16 | TECTL | **Test Enable Shared Control Pins:**<br>0 = RASB#, CASB#, CKEB, WEB# (normal operation)<br>1 = RASB#, CASB#, CKEB, WEB# are used to output test information |
| 15:12 | SEL | **Select:** This field is used for debug purposes only. Should be left at zero for normal operation. |
| 11 | RSVD | **Reserved:** Set to 0. |
| 10:0 | GBADD | **Graphics Base Address:** This field indicates the graphics memory base address, which is programmable on 512 KB boundaries. This field corresponds to address bits [29:19].<br>Note that BC_DRAM_TOP must be set to a value lower than the Graphics Base Address. |
| **GX_BASE+8418h-841Bh** | **MC_DR_ADD (R/W)** | **Default Value = 00000000h** |
| 31:10 | RSVD | **Reserved:** Set to 0. |
| 9:0 | DRADD | **Dirty RAM Address:** This field is the address index that is used to access the Dirty RAM with the MC_DR_ACC register. This field does not auto increment. |
| **GX_BASE+841Ch-841Fh** | **MC_DR_ACC (R/W)** | **Default Value = 0000000xh** |
| 31:2 | RSVD | **Reserved:** Set to 0. |
| 1 | D | **Dirty Bit:** This bit is read/write accessible. |
| 0 | V | **Valid Bit:** This bit is read/write accessible. |

### 4.3.5 Address Translation

The memory controller supports two address translations depending on the method used to interleave pages. The hardware automatically enables high order interleaving. Low order interleaving is automatically enabled only under specific memory configurations.

#### 4.3.5.1 High Order Interleaving

High Order Interleaving (HOI) uses the most significant address bits to select which bank the page is located in. This interleaving scheme works with any mixture of DIMM types. However, it spreads the pages over wide address ranges. For example, two 8 MB DIMMs contain a total of four component pages. Two pages are together in one DIMM separated from the other two pages by 8 MB.

#### 4.3.5.2 Auto Low Order Interleaving

The memory controller requires that banks 0:1 if both installed, be identical and banks 2:1 if both installed, be identical. When banks 0:1 are installed or banks 2,3 are installed Auto Low Order Interleaving (LOI) is in effect for those bank pairs. Therefore each DIMM (banks 0:1 or 2:3) must have the same number of DIMM banks, component banks, module sizes and page sizes.

LOI uses the least significant bits after the page bits to select which bank the page is located in. This requires that memory is a power of 2, that the number of banks is a power of 2, and that the page sizes are the same. As stated before, for LOI to work, the DIMMs have to be of the same type. LOI does give a good benefit by providing a moving page throughout memory. Using the same example as above, two banks would be on one DIMM and the next two banks would be on the second DIMM, but they would be linear in address space. For an eight bank system that has 1 KB address (8 KB data) pages, there would be an effective moving page of 64 KB of data.

#### 4.3.5.3 Physical Address to DRAM Address Conversion

Tables 4-16 and 4-17 give Auto LOI address conversion examples when two DIMMs of the same size are used in a system. Table 4-16 shows a one DIMM bank conversion example, while Table 4-17 shows a two DIMM bank example.

Tables 4-18 and 4-19 give Non-Auto LOI address conversion examples when either one or two DIMMs of different sizes are used in a system. Table 4-18 shows a one DIMM bank address conversion example, while Table 4-19 shows a two DIMM bank example. The addresses are computed on a per DIMM basis.

Since the DRAM interface is 64 bits wide, the lower three bits of the physical address get mapped onto the DQM[7:0] lines. Thus, the address conversion tables (Tables 4-16 through 4-19) show the physical address starting from A3.

## Integrated Functions (Continued)

### Table 4-16. Auto LOI -- 2 DIMMs, Same Size, 1 DIMM Bank

| | 1 KB Page Size | | 2 KB Page Size | | 4 KB Page Size | | | 1 KB Page Size | | 2 KB Page Size | | 4 KB Page Size | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Row | Col | Row | Col | Row | Col | | Row | Col | Row | Col | Row | Col |
| Address | 2 Component Banks | | | | | | | 4 Component Banks | | | | | |
| MA12 | A24 | -- | A25 | -- | A26 | | | A25 | -- | A26 | -- | A27 | |
| MA11 | A23 | -- | A24 | -- | A25 | | | A24 | -- | A25 | -- | A26 | |
| MA10 | A22 | -- | A23 | -- | A24 | | | A23 | -- | A24 | -- | A25 | |
| MA9 | A21 | -- | A22 | -- | A23 | | | A22 | -- | A23 | -- | A24 | |
| MA8 | A20 | -- | A21 | -- | A22 | A11 | | A21 | -- | A22 | -- | A23 | A11 |
| MA7 | A19 | -- | A20 | A10 | A21 | A10 | | A20 | -- | A21 | A10 | A22 | A10 |
| MA6 | A18 | A9 | A19 | A9 | A20 | A9 | | A19 | A9 | A20 | A9 | A21 | A9 |
| MA5 | A17 | A8 | A18 | A8 | A19 | A8 | | A18 | A8 | A19 | A8 | A20 | A8 |
| MA4 | A16 | A7 | A17 | A7 | A18 | A7 | | A17 | A7 | A18 | A7 | A19 | A7 |
| MA3 | A15 | A6 | A16 | A6 | A17 | A6 | | A16 | A6 | A17 | A6 | A18 | A6 |
| MA2 | A14 | A5 | A15 | A5 | A16 | A5 | | A15 | A5 | A16 | A5 | A17 | A5 |
| MA1 | A13 | A4 | A14 | A4 | A15 | A4 | | A14 | A4 | A15 | A4 | A16 | A4 |
| MA0 | A12 | A3 | A13 | A3 | A14 | A3 | | A13 | A3 | A14 | A3 | A15 | A3 |
| CS0#/CS1# | A11 | | A12 | | A13 | | | A12 | | A13 | | A14 | |
| CS2#/CS3# | -- | | -- | | -- | | | -- | | -- | | -- | |
| BA0/BA1 | A10 | | A11 | | A12 | | | A11/A10 | | A12/A11 | | A13/A12 | |

### Table 4-17. Auto LOI -- 2 DIMMs, Same Size, 2 DIMM Banks

| | 1 KB Page Size | | 2 KB Page Size | | 4 KB Page Size | | | 1 KB Page Size | | 2 KB Page Size | | 4 KB Page Size | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Row | Col | Row | Col | Row | Col | | Row | Col | Row | Col | Row | Col |
| Address | 2 Component Banks | | | | | | | 4 Component Banks | | | | | |
| MA12 | A25 | -- | A26 | -- | A27 | | | A26 | -- | A27 | -- | A28 | -- |
| MA11 | A24 | -- | A25 | -- | A26 | | | A25 | -- | A26 | -- | A27 | -- |
| MA10 | A23 | -- | A24 | -- | A25 | | | A24 | -- | A25 | -- | A26 | -- |
| MA9 | A22 | -- | A23 | -- | A24 | | | A23 | -- | A24 | -- | A25 | -- |
| MA8 | A21 | -- | A22 | -- | A23 | A11 | | A22 | -- | A23 | -- | A24 | A11 |
| MA7 | A20 | -- | A21 | A10 | A22 | A10 | | A21 | -- | A22 | A10 | A23 | A10 |
| MA6 | A19 | A9 | A20 | A9 | A21 | A9 | | A20 | A9 | A21 | A9 | A22 | A9 |
| MA5 | A18 | A8 | A19 | A8 | A20 | A8 | | A19 | A8 | A20 | A8 | A21 | A8 |
| MA4 | A17 | A7 | A18 | A7 | A19 | A7 | | A18 | A7 | A19 | A7 | A20 | A7 |
| MA3 | A16 | A6 | A17 | A6 | A18 | A6 | | A17 | A6 | A18 | A6 | A19 | A6 |
| MA2 | A15 | A5 | A16 | A5 | A17 | A5 | | A16 | A5 | A17 | A5 | A18 | A5 |
| MA1 | A14 | A4 | A15 | A4 | A16 | A4 | | A15 | A4 | A16 | A4 | A17 | A4 |
| MA0 | A13 | A3 | A14 | A3 | A15 | A3 | | A14 | A3 | A15 | A3 | A16 | A3 |
| CS0#/CS1# | A12 | | A13 | | A14 | | | A13 | | A14 | | A15 | |
| CS2#/CS3# | A11 | | A12 | | A13 | | | A12 | | A13 | | A14 | |
| BA0/BA1 | A10 | | A11 | | A12 | | | A11/A10 | | A12/A11 | | A13/A12 | |

## Integrated Functions (Continued)

### Table 4-18. Non-Auto LOI -- 1 or 2 DIMMs, Different Sizes, 1 DIMM Bank

| | 1 KB Page Size | | 2 KB Page Size | | 4 KB Page Size | | | 1 KB Page Size | | 2 KB Page Size | | 4 KB Page Size | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Row | Col | Row | Col | Row | Col | | Row | Col | Row | Col | Row | Col |
| Address | 2 Component Banks | | | | | | | 4 Component Banks | | | | | |
| MA12 | A23 | -- | A24 | -- | A25 | -- | | A24 | -- | A25 | -- | A26 | |
| MA11 | A22 | -- | A23 | -- | A24 | -- | | A23 | -- | A24 | -- | A25 | |
| MA10 | A21 | -- | A22 | -- | A23 | -- | | A22 | -- | A23 | -- | A24 | |
| MA9 | A20 | -- | A21 | -- | A22 | -- | | A21 | -- | A22 | -- | A23 | |
| MA8 | A19 | -- | A20 | -- | A21 | A11 | | A20 | -- | A21 | -- | A22 | A11 |
| MA7 | A18 | -- | A19 | A10 | A20 | A10 | | A19 | -- | A20 | A10 | A21 | A10 |
| MA6 | A17 | A9 | A18 | A9 | A19 | A9 | | A18 | A9 | A19 | A9 | A20 | A9 |
| MA5 | A16 | A8 | A17 | A8 | A18 | A8 | | A17 | A8 | A18 | A8 | A19 | A8 |
| MA4 | A15 | A7 | A16 | A7 | A17 | A7 | | A16 | A7 | A17 | A7 | A18 | A7 |
| MA3 | A14 | A6 | A15 | A6 | A16 | A6 | | A15 | A6 | A16 | A6 | A17 | A6 |
| MA2 | A13 | A5 | A14 | A5 | A15 | A5 | | A14 | A5 | A15 | A5 | A16 | A5 |
| MA1 | A12 | A4 | A13 | A4 | A14 | A4 | | A13 | A4 | A14 | A4 | A15 | A4 |
| MA0 | A11 | A3 | A12 | A3 | A13 | A3 | | A12 | A3 | A13 | A3 | A14 | A3 |
| CS0#/CS1# | -- | | -- | | -- | | | -- | | -- | | -- | |
| CS2#/CS3# | -- | | -- | | -- | | | -- | | -- | | -- | |
| BA0/BA1 | A10 | | A11 | | A12 | | | A11/A10 | | A12/A11 | | A13/A12 | |

### Table 4-19. Non-Auto LOI -- 1 or 2 DIMMs, Different Sizes, 2 DIMM Banks

| | 1 KB Page Size | | 2 KB Page Size | | 4 KB Page Size | | | 1 KB Page Size | | 2 KB Page Size | | 4 KB Page Size | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Row | Col | Row | Col | Row | Col | | Row | Col | Row | Col | Row | Col |
| Address | 2 Component Banks | | | | | | | 4 Component Banks | | | | | |
| MA12 | A24 | -- | A25 | -- | A26 | -- | | A25 | -- | A26 | -- | A27 | -- |
| MA11 | A23 | -- | A24 | -- | A25 | -- | | A24 | -- | A25 | -- | A26 | -- |
| MA10 | A22 | -- | A23 | -- | A24 | -- | | A23 | -- | A24 | -- | A25 | -- |
| MA9 | A21 | -- | A22 | -- | A23 | -- | | A22 | -- | A23 | -- | A24 | -- |
| MA8 | A20 | -- | A21 | -- | A22 | A11 | | A21 | -- | A22 | -- | A23 | A11 |
| MA7 | A19 | -- | A20 | A10 | A21 | A10 | | A20 | -- | A21 | A10 | A22 | A10 |
| MA6 | A18 | A9 | A19 | A9 | A20 | A9 | | A19 | A9 | A20 | A9 | A21 | A9 |
| MA5 | A17 | A8 | A18 | A8 | A19 | A8 | | A18 | A8 | A19 | A8 | A20 | A8 |
| MA4 | A16 | A7 | A17 | A7 | A18 | A7 | | A17 | A7 | A18 | A7 | A19 | A7 |
| MA3 | A15 | A6 | A16 | A6 | A17 | A6 | | A16 | A6 | A17 | A6 | A18 | A6 |
| MA2 | A14 | A5 | A15 | A5 | A16 | A5 | | A15 | A5 | A16 | A5 | A17 | A5 |
| MA1 | A13 | A4 | A14 | A4 | A15 | A4 | | A14 | A4 | A15 | A4 | A16 | A4 |
| MA0 | A12 | A3 | A13 | A3 | A14 | A3 | | A13 | A3 | A14 | A3 | A15 | A3 |
| CS0#/CS1# | A11 | | A12 | | A13 | | | A12 | | A13 | | A14 | |
| CS2#/CS3# | -- | | -- | | | | | | | | | | |
| BA0/BA1 | A10 | | A11 | | A12 | | | A11/A10 | | A12/A11 | | A13/A12 | |

## Integrated Functions (Continued)

### 4.3.6 Memory Cycles

Figures 4-5 through 4-8 illustrate various memory cycles that the memory controller supports. The following sub-sections describe some of the supported cycles.

**SDRAM Read Cycle**

Figure 4-5 shows a SDRAM read cycle. The figure assumes that a previous ACT command has presented the row address for the read operation. Note that the burst length for the READ command is always two.

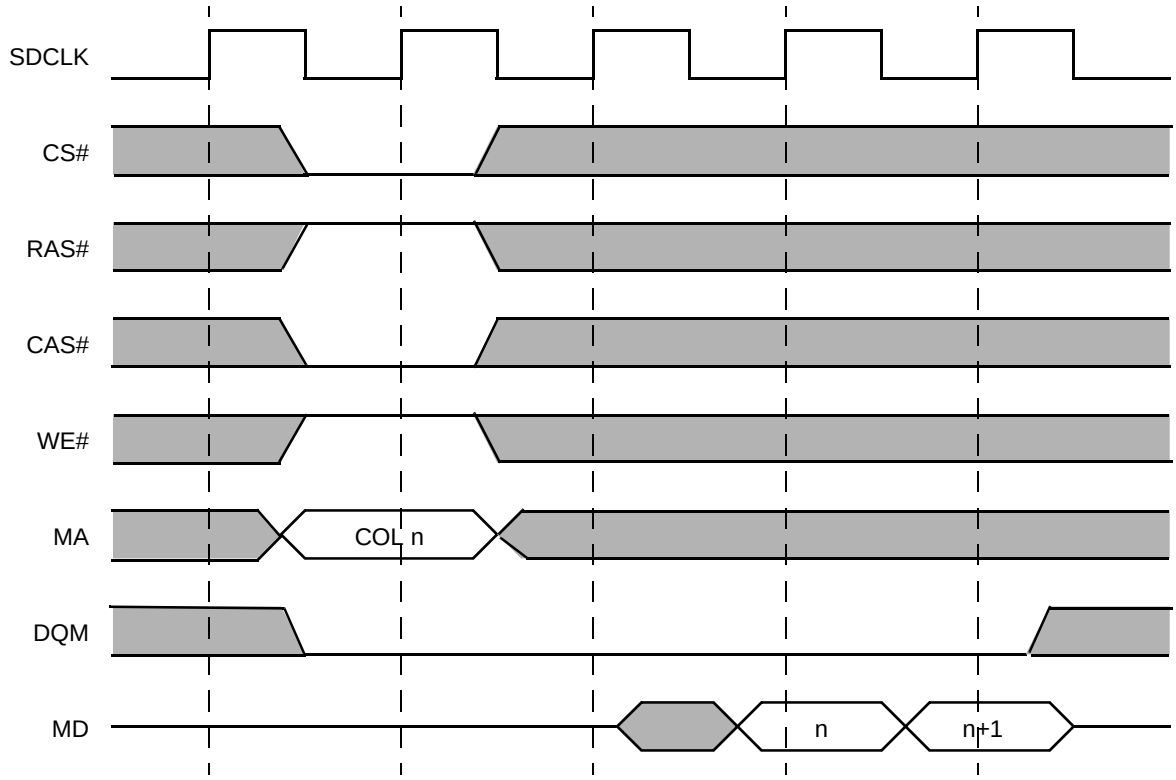



**Figure 4-5. Basic Read Cycle with a CAS Latency of Two**

## Integrated Functions (Continued)

**SDRAM Write Cycle**

Figure 4-6 shows a SDRAM write cycle. The burst length for the WRT command is two.

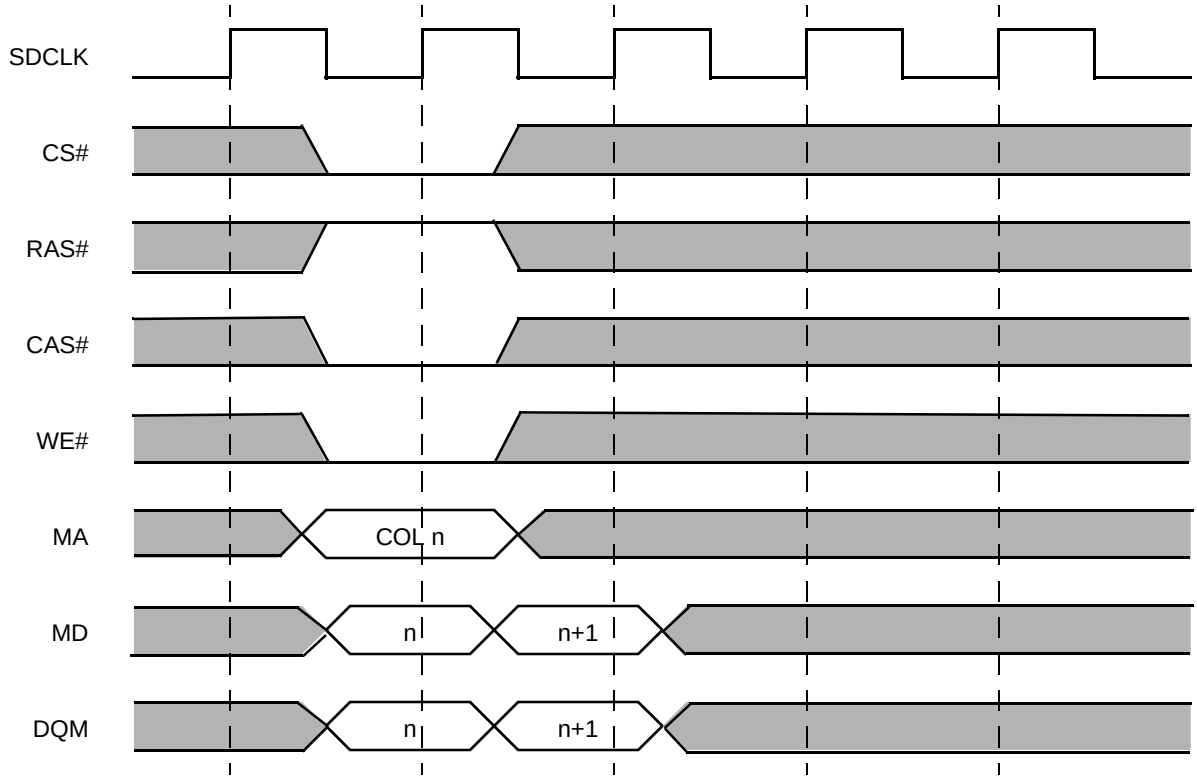



**Figure 4-6. Basic Write Cycle**

## Integrated Functions (Continued)

### SDRAM Refresh Cycle

Figure 4-7 shows a SDRAM auto refresh cycle. The memory controller always precedes the refresh cycle with a PRE command to all banks.

### Page Miss

Figure 4-8 shows a Read/WRT command after a page miss cycle. In order to program the new row address, a PRE command must be issued followed by an ACT command.

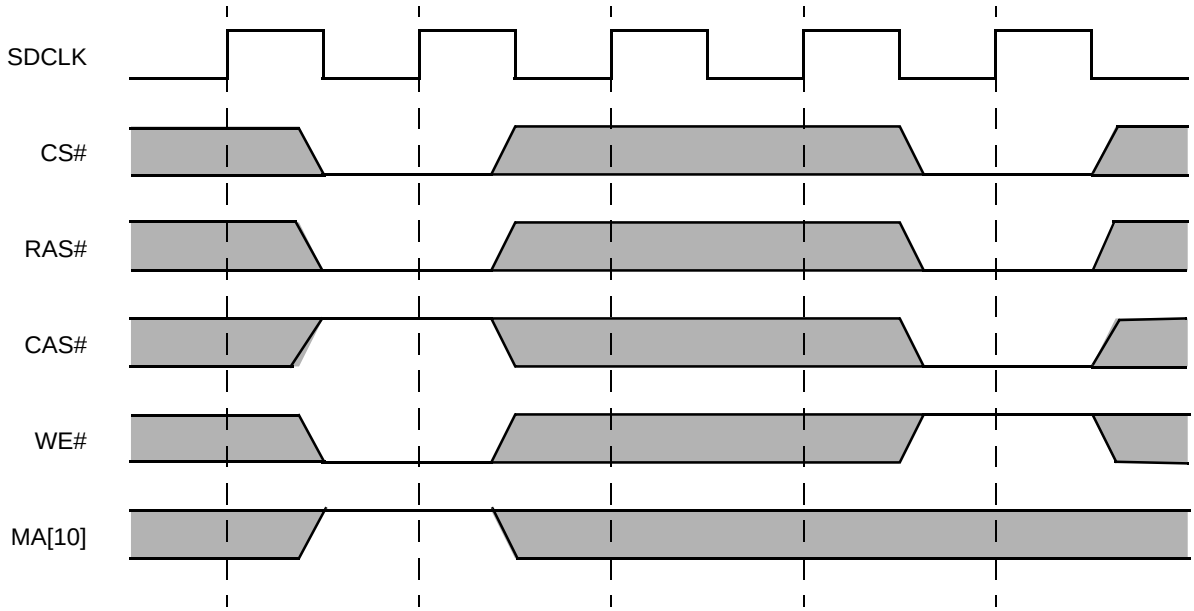



**Figure 4-7. Auto Refresh Cycle**

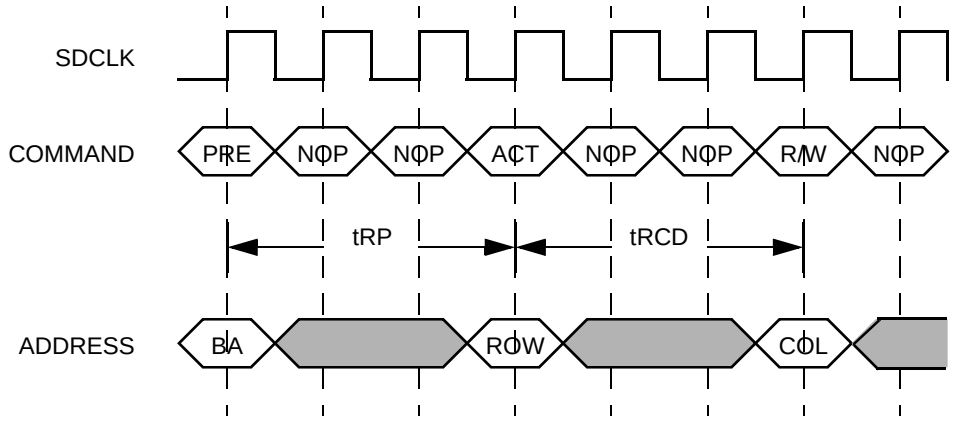



**Figure 4-8. Read/WRT Command to a New Row Address**

### 4.3.7 SDRAM Interface Clocking

The GXLV processor drives the SDCLK to the SDRAMs; one for each DIMM bank. All the control, data, and address signals driven by the memory controller are sampled by the SDRAM at the rising edge of SDCLK. SDCLKOUT is a reference signal used to generate SDCLKIN. Read data is sampled by the memory controller at the rising edge of SDCLKIN.

The delay for SDCLKIN from SDCLKOUT must be designed so that it lags the SDCLKs at the DRAM by approximately 1 ns (check application notes for additional information). The delay should also include the SDCLK transmission line delay. All four SDCLK traces on the board should be the same length, so there is no skew between them. These guidelines allow the memory interface to operate at a higher performance.

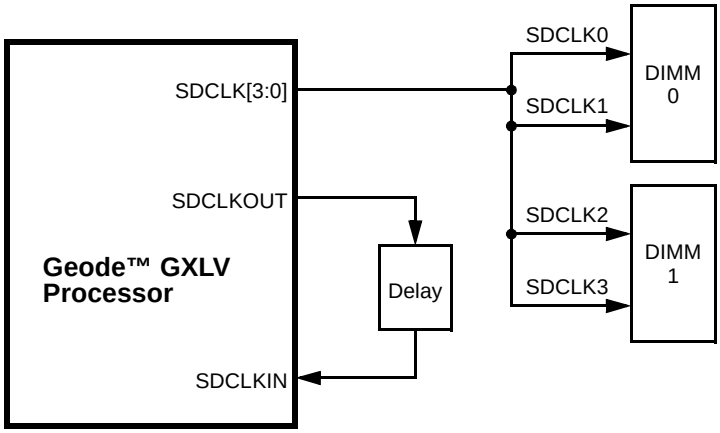


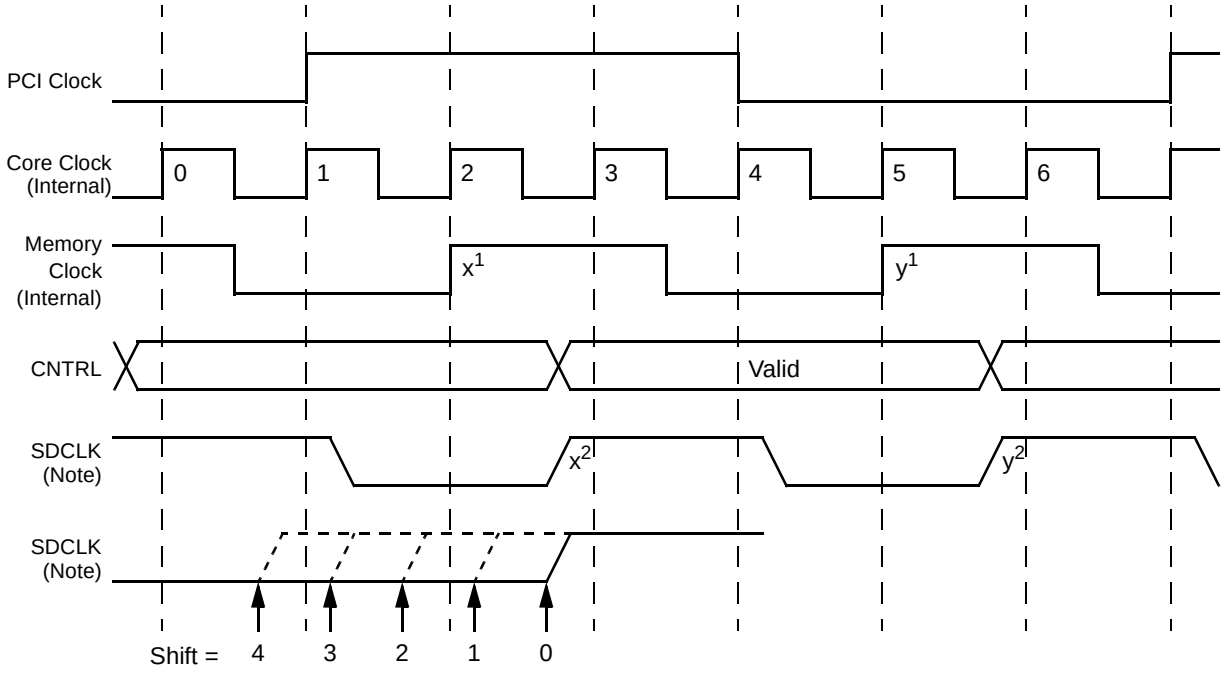

**Figure 4-9. SDCLKIN Clocking**

## Integrated Functions (Continued)

The SDRAM interface timings are programmable. The SHFTSDCLK bits in the MC_MEM_CNTRL2 register can be used to change the relationship between SDCLK and the control/address/data signals to meet setup and hold time requirements for SDRAM across different board layouts. SHFTSDCLK bit values are selected based upon the SDRAM signals loads and the core frequency (refer to Figures 6-9 and 6-10 on page 202).

Figure 4-10 shows an example of how the SHFTSDCLK bits setting affects SDCLK. The PCI clock is the input clock to the GXLV processor. The core clock is the internal processor clock that is multiplied up. The memory controller runs off this core clock. The memory clock is generated by dividing down the core clock. SDCLK is generated from the memory clock. In the example diagram, the processor clock is running 6X times the PCI clock and the memory clock is running in divide by 3 mode.

The SDRAM control, address, and data signals are driven off edge "$x^1$" of the memory clock to be setup before edge "$y^1$". With no shift applied, the control signals could end up being latched on edge "$x^2$" of the SDCLK. A shift value of two or three could be used so that SDCLK at the SDRAM is centered around when the control signals change.




**Note:** The first SDCLK shows how SDCLK operates with the SHFTSDCLK bits = 000, no shift.
The second SDCLK shows how SDCLK operates with the SHFTSDCLK bits = 001, shift 0.5 core clock.
(See MC_MEMCNTRL2 bits [5:3], Table 4-15 on page 114, for remaining decode values.)

**Figure 4-10. Effects of SHFTSDCLK Programming Bits Example**

## 4.4 GRAPHICS PIPELINE

The graphics pipeline of the GXLV processor contains a 2D graphics accelerator. This hardware accelerator has a BitBLT/vector engine which dramatically improves graphics performance when rendering and moving graphical objects. Overall operating system performance is improved as well. The accelerator hardware supports pattern generation, source expansion, pattern/source transparency, and 256 ternary raster operations. The block diagram of the graphics pipeline is shown in Figure 4-11.

### 4.4.1 BitBLT/Vector Engine

BLTs are initiated by writing to the GP_BLT_MODE register, which specifies the type of source data (none, frame buffer, or BLT buffer), the type of the destination data (none, frame buffer, or BLT buffer), and a source expansion flag.

Vectors are initiated by writing to the GP_VECTOR_MODE register (GX_BASE+8204h), which specifies the direction of the vector and a “read destination data” flag. If the flag is set, the hardware will read destination data along the vector and store it temporarily in the BLT Buffer 0.

The BLT buffers use a portion of the L1 cache, called “scratchpad RAM”, to temporarily store source and destination data, typically on a scan line basis. See Section 4.1.4.2 “Scratchpad RAM Utilization” for an explanation of scratchpad RAM. The hardware automatically loads frame-buffer data (source or destination) into the BLT buffers for each scan line. The driver is responsible for making sure that this does not overflow the memory allocated for the BLT buffers. When the source data is a bitmap, the hardware loads the data directly into the BLT buffer at the beginning of the BLT operation.

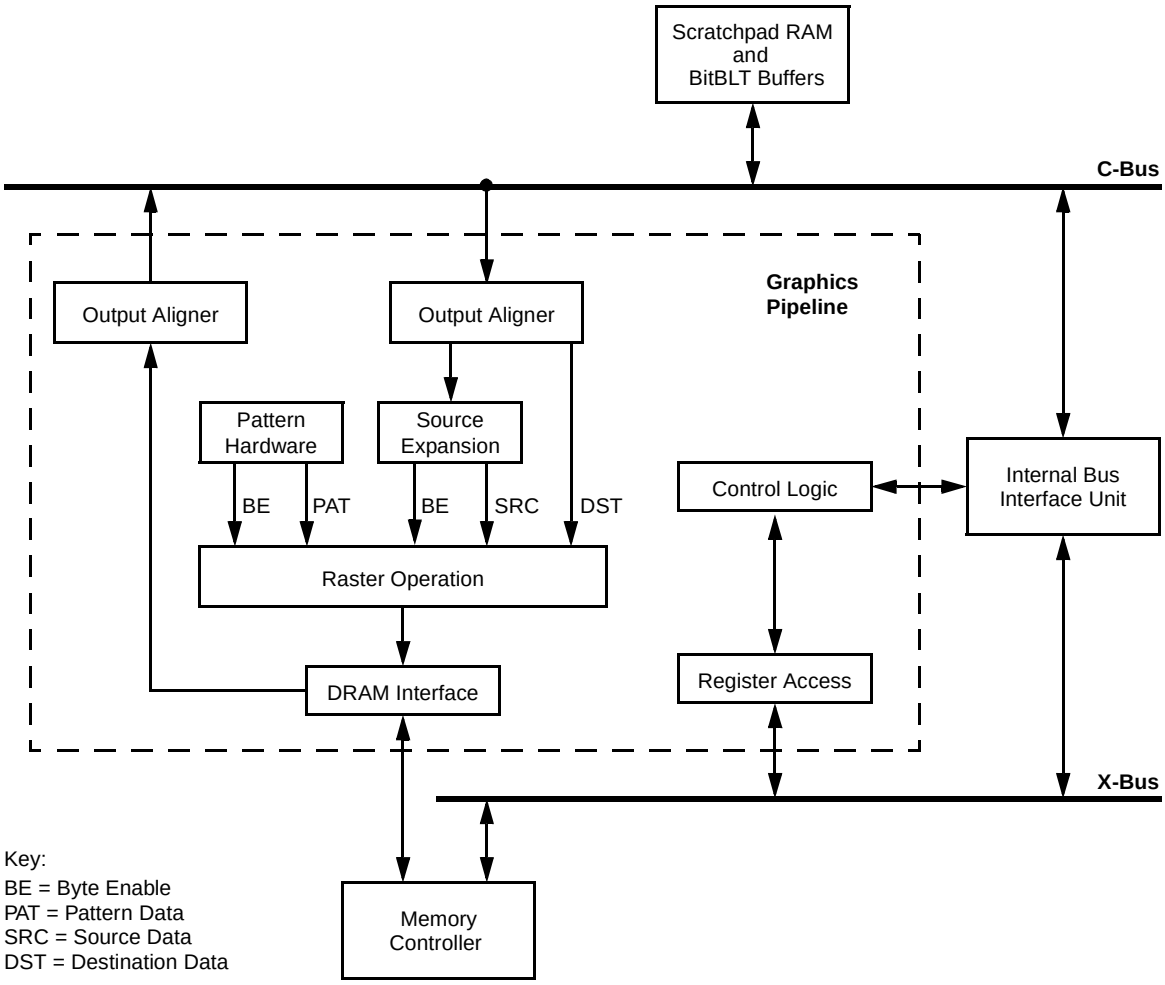



**Figure 4-11. Graphics Pipeline Block Diagram**

## Integrated Functions (Continued)

### 4.4.2 Master/Slave Registers

When starting a BitBLT or vector operation, the graphics pipeline registers are latched from the master registers to the slave registers. A second BitBLT or vector operation can then be loaded into the master registers while the first operation is rendered. If a second BLT is pending in the master registers, any write operations to the graphics pipeline registers will corrupt the values of the pending BLT. Software must prevent this from happening by checking the “BLT Pending” bit in the GP_BLT_STATUS register (GX_BASE+820Ch[2]).

Most of the graphics pipeline registers are latched directly from the master registers to the slave registers when starting a new BitBLT or vector operation. Some registers, however, use the updated slave values if the master registers have not been written, which allows software to render successive primitives without loading some of the registers as outlined in Table 4-20.

### 4.4.3 Pattern Generation

The graphics pipeline contains hardware support for 8x8 monochrome patterns (expanded to two colors), 8x8 dither patterns (expanded to four colors), and 8x1 color patterns. The pattern hardware, however, does not maintain a pattern origin, so the pattern data must be justified before it is loaded into the GXLV processor’s registers. For solid primitives, the pattern hardware is disabled and the pattern color is always sourced from the GP_PAT_COLOR_0 register (GX_BASE+8110h).

**Table 4-20. Graphics Pipeline Registers**

| Master | Function |
|---|---|
| GP_DST_XCOOR | Next X position along vector.<br>Master register if written, otherwise:<br>Unchanged slave if BLT, source mode = bitmap.<br>Slave + width if BLT, source mode = text glyph |
| GP_DST_YCOOR | Next Y position along vector.<br>Master register if written, otherwise:<br>Slave +/- height if BLT, source mode = bitmap.<br>Unchanged slave if BLT, source mode = text glyph. |
| GP_INIT_ERROR | Master register if written, otherwise:<br>Initial error for the next pixel along the vector. |
| GP_SRC_YCOOR | Master register if written, otherwise:<br>Slave +/- height if BLT, source mode = bitmap. |

## Integrated Functions (Continued)

### 4.4.3.1 Monochrome Patterns

Setting the pattern mode to 01b (GX_BASE+8200h[9:8] = 01b) in the GP_RASTER_MODE register selects the monochrome patterns (see bit details on page 131). Those pixels corresponding to a clear bit (0) in the pattern are rendered using the color specified in the GP_PAT_COLOR_0 (GX_BASE+8110h) register, and those pixels corresponding to a set bit (1) in the pattern are rendered using the color specified in the GP_PAT_COLOR_1 register (GX_BASE+8112h).

If the pattern transparency bit is set high in the GP_RASTER_MODE register, those pixels corresponding to a clear bit in the pattern data are not drawn.

Monochrome patterns use registers GP_PAT_DATA_0 (GX_BASE+ Memory Offset 8120h) and GP_PAT_DATA_1 (GX_BASE+ memory Offset 8124h) for the pattern data. Bits [7:0] of GP_PAT_DATA_0 correspond to the first row of the pattern, and bit 7 corresponds to the leftmost pixel on the screen. How the pattern and the registers fully relate is illustrated in Figure 4-12.

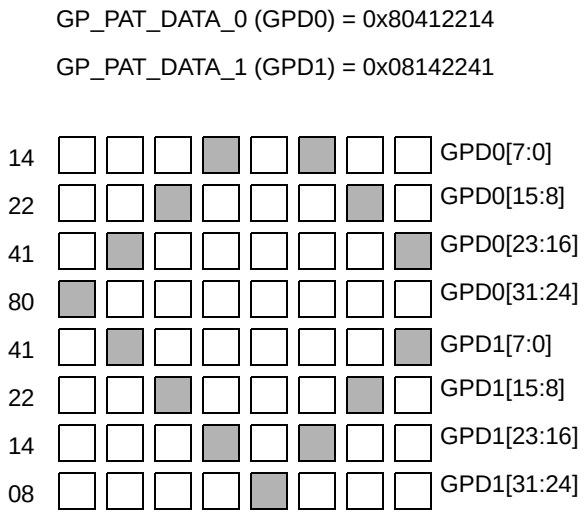



**Figure 4-12. Example of Monochrome Patterns**

### 4.4.3.2 Dither Patterns

Setting the pattern mode to 10b (GX_BASE+8200h[9:8] = 10b) in the GP_RASTER_MODE register selects the dither patterns. Two bits of pattern data are used for each pixel, allowing color expansion to four colors. The colors are specified in the GP_PAT_COLOR_0 through GP_PAT_COLOR_3 registers (Table 4-24 on page 130).

Dither patterns use all 128 bits of pattern data. Bits [15:0] of GP_PAT_DATA_0 correspond to the first row of the pattern (the lower byte contains the least significant bit of each pixel's pattern color and the upper byte contains the most significant bit of each pixel's pattern color). This is illustrated in Figure 4-13.

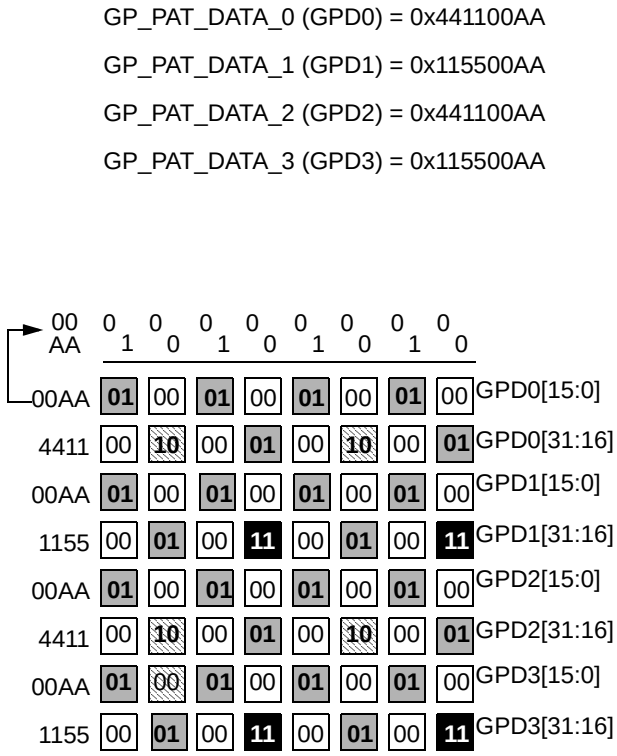



**Figure 4-13. Example of Dither Patterns**

## Integrated Functions (Continued)

#### 4.4.3.3 Color Patterns

Setting the pattern mode to 11b (GX_BASE+8200h[9:8] = 11b), in the GP_RASTER_MODE register selects the color patterns. Bits [63:0] are used to hold a row of pattern data for an 8-bpp pattern, with bits [7:0] corresponding to the leftmost pixel of the row. Likewise, bits [127:0] are used for a 16-bpp color pattern, with bits [15:0] corresponding to the leftmost pixel of the row.

To support an 8x8 color pattern, software must load the pattern data for each row.

### 4.4.4 Source Expansion

The graphics pipeline contains hardware support for color expansion of source data (primarily used for text). Those pixels corresponding to a clear bit (0) in the source data are rendered using the color specified in the GP_SRC_COLOR_0 register (GX_BASE+810Ch), and those pixels corresponding to a set bit (1) in the source data are rendered using the color specified in the GP_SRC_COLOR_1 register (GX_BASE+810Eh).

If the source transparency bit is set in the GP_RASTER_MODE register, those pixels corresponding to a clear bit (0) in the source data are not drawn.

### 4.4.5 Raster Operations

The GP_RASTER_MODE register specifies how the pattern data, source data (color-expanded if necessary), and destination data are combined to produce the output to the frame buffer. The definition of the ROP value matches that of the Microsoft API (application programming interface). This allows Windows display drivers to load the raster operation directly into hardware. Table 4-21 illustrates this definition. Some common raster operations are described in Table 4-22.

**Table 4-21. GP_RASTER_MODE Bit Patterns**

| Pattern (bit) | Source (bit) | Destination (bit) | Output (bit) |
|---|---|---|---|
| 0 | 0 | 0 | ROP[0] |
| 0 | 0 | 1 | ROP[1] |
| 0 | 1 | 0 | ROP[2] |
| 0 | 1 | 1 | ROP[3] |
| 1 | 0 | 0 | ROP[4] |
| 1 | 0 | 1 | ROP[5] |
| 1 | 1 | 0 | ROP[6] |
| 1 | 1 | 1 | ROP[7] |

**Table 4-22. Common Raster Operations**

| ROP | Description |
|---|---|
| F0h | Output = Pattern |
| CCh | Output = Source |
| 5Ah | Output = Pattern XOR destination |
| 66h | Output = Source XOR destination |
| 55h | Output = ~Destination |

### 4.4.6 Graphics Pipeline Register Descriptions

The graphics pipeline maps 200h locations starting at GX_BASE+8100h. Refer to Section 4.1.2 “Control Registers” on page 99 for instructions on accessing these registers. Table 4-23 summarizes the graphics pipeline registers and Table 4-24 gives detailed register/bit formats.

**Table 4-23. Graphics Pipeline Configuration Register Summary**

| GX_BASE+ Memory Offset | Type | Name / Function | Default Value |
|---|---|---|---|
| 8100h-8103h | R/W | **GP_DST/START_Y/XCOOR**<br>Destination/Starting Y and X Coordinates Register: In BLT mode this register specifies the destination Y and X positions for a BLT operation. In Vector mode it specifies the starting Y and X positions in a vector. | 00000000h |
| 8104-8107h | R/W | **GP_WIDTH/HEIGHT and GP_VECTOR_LENGTH/INIT_ERROR**<br>Width/Height or Vector Length/Initial Error Register: In BLT mode this register specifies the BLT width and height in pixels. In Vector mode it specifies the vector initial error and pixel length. | 00000000h |
| 8108h-810Bh | R/W | **GP_SRC_X/YCOOR and GP_AXIAL/DIAG_ERROR**<br>Source X/Y Coordinate Axial/Diagonal Error Register: In BLT mode this register specifies the BLT X and Y source. In Vector mode it specifies the axial and diagonal error for rendering a vector. | 00000000h |
| 810Ch-810Fh | R/W | **GP_SRC_COLOR_0 and GP_SRC_COLOR_1**<br>Source Color Register: Determines the colors used when expanding monochrome source data in either the 8-bpp mode or the 16-bpp mode. | 00000000h |
| 8110h-8113h | R/W | **GP_PAT_COLOR_0 and GP_PAT_COLOR_1**<br>Graphics Pipeline Pattern Color Registers 0 and1: These two registers determine the colors used when expanding pattern data. | 00000000h |
| 8114h-8117h | R/W | **GP_PAT_COLOR_2 and GP_PAT_COLOR_3**<br>Graphics Pipeline Pattern Color Registers 2 and 3: These two registers determine the colors used when expanding pattern data. | 00000000h |
| 8120h-8123h | R/W | **GP_PAT_DATA 0 through 3**<br>Graphics Pipeline Pattern Data Registers 0 through 3: Together these registers contain 128 bits of pattern data.<br>GP_PAT_DATA_0 corresponds to bits [31:0] of the pattern data.<br>GP_PAT_DATA_1 corresponds to bits [63:32] of the pattern data.<br>GP_PAT_DATA_2 corresponds to bits [95:64] of the pattern data.<br>GP_PAT_DATA_3 corresponds to bits [127:96] of the pattern data. | 00000000h |
| 8124h-8127h | R/W | | 00000000h |
| 8128h-812Bh | R/W | | 00000000h |
| 812Ch-812Fh | R/W | | 00000000h |
| 8140h-8143h (Note) | R/W | **GP_VGA_WRITE**<br>Graphics Pipeline VGA Write Patch Control Register: Controls the VGA memory write path in the graphics pipeline. | xxxxxxxxh |
| 8144h-8147h (Note) | R/W | **GP_VGA_READ**<br>Graphics Pipeline VGA Read Patch Control Register: Controls the VGA memory read path in the graphics pipeline. | 00000000h |
| 8200h-8203h | R/W | **GP_RASTER_MODE**<br>Graphics Pipeline Raster Mode Register: This register controls the manipulation of the pixel data through the graphics pipeline. Refer to Section 4.4.5 “Raster Operations” on page 128. | 00000000h |
| 8204h-8207h | R/W | **GP_VECTOR_MODE**<br>Graphics Pipeline Vector Mode Register: Writing to this register initiates the rendering of a vector. | 00000000h |
| 8208h-820Bh | R/W | **GP_BLT_MODE**<br>Graphics Pipeline BLT Mode Register: Writing to this initiates a BLT operation. | 00000000h |

**Note:** The registers at GX_BASE+8140, 8144h, 8210h, and 8214h are located in the area designated for the graphics pipeline but are used for VGA emulation purposes. Refer to Table 4-39 on page 165 for these register’s bit formats.

## Integrated Functions (Continued)

### Table 4-23. Graphics Pipeline Configuration Register Summary (Continued)

| GX_BASE+ Memory Offset | Type | Name / Function | Default Value |
|---|---|---|---|
| 820Ch-820Fh | R/W | **GP_BLT_STATUS**<br>Graphics Pipeline BLT Status Register: Contains configuration and status information for the BLT engine. The status bits are contained in the lower byte of the register. | 00000000h |
| 8210h-8213h (Note) | R/W | **GP_VGA_BASE**<br>Graphics Pipeline VGA Memory Base Address Register: Specifies the offset of the VGA memory, starting from the base of graphics memory. | xxxxxxxxh |
| 8214h-8217h (Note) | R/W | **GP_VGA_LATCH**<br>Graphics Pipeline VGA Display Latch Register: Provides a memory mapped way to read or write the VGA display latch. | xxxxxxxxh |

**Note:** The registers at GX_BASE+8140, 8144h, 8210h, and 8214h are located in the area designated for the graphics pipeline but are used for VGA emulation purposes. Refer to Table 4-39 on page 165 for these register's bit formats.

### Table 4-24. Graphics Pipeline Configuration Registers

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+8100h-8103h** | **GP_DST/START_X/YCOOR Register (R/W)** | **Default Value = 00000000h** |
| 31:16 | **DESTINATION/STARTING Y POSITION (SIGNED):** | BLT Mode: Specifies the destination Y position for a BLT operation.<br>Vector Mode: Specifies the starting Y position in a vector. |
| 15:0 | **DESTINATION/STARTING X POSITION (SIGNED):** | BLT Mode: Specifies the destination X position for a BLT operation.<br>Vector Mode: Specifies the starting X position in a vector. |
| **GX_BASE+8104h-8107h** | **GP_WIDTH/HEIGHT and GP_VECTOR_LENGTH/INIT_ERROR Register (R/W)** | **Default Value = 00000000h** |
| 31:16 | **PIXEL_WIDTH or VECTOR_LENGTH (UNSIGNED)**: | BLT Mode: Specifies the width, in pixels, of a BLT operation. No pixels are rendered for a width of zero.<br>Vector Mode: Bits [31:30] are reserved in this mode allowing this 14-bit field to specify the length, in pixels, of a vector. No pixels are rendered for a length of zero. This field is limited to 14 bits due to a lack of precision in the registers used to hold the error terms. |
| 15:0 | **PIXEL_HEIGHT or VECTOR_INITIAL_ERROR (UNSIGNED):** | BLT Mode: Specifies the height, in pixels, of a BLT operation. No pixels are rendered for a height of zero.<br>Vector Mode: Specifies the initial error for rendering a vector. |
| **GX_BASE+8108h-810Bh** | **GP_SCR_X/YCOOR and GP_AXIAL/DIAG_ERROR Register (R/W)** | **Default Value = 00000000h** |
| 31:16 | **SRC_X_POS or VECTOR_AXIAL_ERROR (SIGNED):** | BLT Mode: Specifies the source X position for a BLT operation.<br>Vector Mode: Specifies the axial error for rendering a vector. |
| 15:0 | **SRC_Y_POS or VECTOR_DIAG_ERROR (SIGNED):** | Source Y Position (Signed): Specifies the source Y position for a BLT operation.<br>Vector Mode: Specifies the diagonal error for rendering a vector. |
| **GX_BASE+810Ch-810Dh** | **GP_SRC_COLOR_0 Register (R/W)** | **Default Value = 0000h** |
| 15:0 | **8-bpp Mode:** 8-bpp color: The color index must be duplicated in the upper byte.<br>**16-bpp Mode**: 16-bpp color (RGB) | |
| **GX_BASE+810Eh-810Fh** | **GP_SRC_COLOR_1 Register (R/W)** | **Default Value = 0000h** |
| 15:0 | **8-bpp Mode:** 8-bpp color: The color index must be duplicated in the upper byte.<br>**16-bpp Mode**: 16-bpp color (RGB) | |

**Note:** The Graphics Pipeline Source Color Register specifies the colors used when expanding monochrome source data in either the 8-bpp mode or the 16-bpp mode. Those pixels corresponding to clear bits (0) in the source data are rendered using GP_SRC_COLOR_0 and those pixels corresponding to set bits (1) in the source data are rendered using GP_SRC_COLOR_1.

**Table 4-24. Graphics Pipeline Configuration Registers (Continued)**

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+8110h-8111h** | | **GP_PAT_COLOR_0 Register (R/W)** — **Default Value = 0000h** |
| 15:0 | | **8-bpp Mode:** 8-bpp color: The color index must be duplicated in the upper byte.<br>**16-bpp Mode:** 16-bpp color (RGB) |
| **Note:** The Graphics Pipeline Pattern Color 0-3 Registers specify the colors used when expanding pattern data. | | |
| **GX_BASE+8112h-8113h** | | **GP_PAT_COLOR_1 Register (R/W)** — **Default Value = 0000h** |
| 15:0 | | **8-bpp Mode:** 8-bpp color: The color index must be duplicated in the upper byte.<br>**16-bpp Mode:** 16-bpp color (RGB) |
| **Note:** The Graphics Pipeline Pattern Color 0-3 Registers specify the colors used when expanding pattern data. | | |
| **GX_BASE+8114h-8115h** | | **GP_PAT_COLOR_2 Register (R/W)** — **Default Value = 0000h** |
| 15:0 | | **8-bpp Mode:** 8-bpp color: The color index must be duplicated in the upper byte.<br>**16-bpp Mode:** 16-bpp color (RGB) |
| **Note:** The Graphics Pipeline Pattern Color 0-3 Registers specify the colors used when expanding pattern data. | | |
| **GX_BASE+8116h-8117h** | | **GP_PAT_COLOR_3 Register (R/W)** — **Default Value = 0000h** |
| 15:0 | | **8-bpp Mode:** 8-bpp color: The color index must be duplicated in the upper byte.<br>**16-bpp Mode:** 16-bpp color (RGB) |
| **Note:** The Graphics Pipeline Pattern Color 0-3 Registers specify the colors used when expanding pattern data. | | |
| **GX_BASE+8120h-8123h** | | **GP_PAT_DATA_0 Register (R/W)** — **Default Value = 00000000h** |
| 31:0 | | **GP Pattern Data Register 0:** The Graphics Pipeline Pattern Data Registers 0 through 3 together contain 128 bits of pattern data. The GP_PAT_DATA_0 register corresponds to bits [31:0] of the pattern data. |
| **GX_BASE+8124h-8127h** | | **GP_PAT_DATA_1 Register (R/W)** — **Default Value = 00000000h** |
| 31:0 | | **GP Pattern Data Register 1:** The Graphics Pipeline Pattern Data Registers 0 through 3 together contain 128 bits of pattern data. The GP_PAT_DATA_1 register corresponds to bits [63:32] of the pattern data. |
| **GX_BASE+8128h-812Bh** | | **GP_PAT_DATA_2 Register (R/W)** — **Default Value = 00000000h** |
| 31:0 | | **GP Pattern Data Register 2:** The Graphics Pipeline Pattern Data Registers 0 through 3 together contain 128 bits of pattern data. The GP_PAT_DATA_2 register corresponds to bits [95:64] of the pattern data. |
| **GX_BASE+812Ch-812Fh** | | **GP_PAT_DATA_3 Register (R/W)** — **Default Value = 00000000h** |
| 31:0 | | **GP Pattern Data Register 3:** The Graphics Pipeline Pattern Data Registers 0 through 3 together contain 128 bits of pattern data. The GP_PAT_DATA_3 register corresponds to bits [127:96] of the pattern data. |
| **GX_BASE+8140h-8143h** | | **GP_VGA_WRITE Register (R/W)** — **Default Value = xxxxxxxxh** |
| Note that the register at GX_BASE+82140h is located in the area designated for the graphics pipeline but is used for VGA emulation purposes. Refer to Table 4-39 on page 165 for this register's bit formats. | | |
| **GX_BASE+8144h-8147h** | | **GP_VGA_READ Register (R/W)** — **Default Value = 00000000h** |
| Note that the register at GX_BASE+8144h is located in the area designated for the graphics pipeline but is used for VGA emulation purposes. Refer to Table 4-39 on page 165 for this register's bit formats. | | |
| **GX_BASE+8200h-8203h** | | **GP_RASTER_MODE Register (R/W)** — **Default Value = 00000000h** |
| 31:13 | RSVD | **Reserved:** Set to 0. |
| 12 | TB | **Transparent BLT:** When set, this bit enables transparent BLT. The source color data will be compared to a color key and if it matches, that pixel will not be drawn. The color key value is stored in the BLT buffer as destination data. The raster operation must be set to C6h, and the pattern registers must be all F's for this mode to work properly. |
| 11 | ST | **Source Transparency:** Enables transparency for monochrome source data. Those pixels corresponding to clear bits in the source data are not drawn. |
| 10 | PT | **Pattern Transparency:** Enables transparency for monochrome pattern data. Those pixels corresponding to clear bits in the pattern data are not drawn. |

## Integrated Functions (Continued)

### Table 4-24. Graphics Pipeline Configuration Registers (Continued)

| Bit | Name | Description |
|---|---|---|
| 9:8 | PM | **Pattern Mode:** Specifies the format of the pattern data.<br>00 = Indicates a solid pattern. The pattern data is always sourced from the GP_PAT_COLOR_0 register.<br>01 = Indicates a monochrome pattern. The pattern data is sourced from the GP_PAT_COLOR_0 and GP_PAT_COLOR_1 registers.<br>10 = Indicates a dither pattern. All four pattern color registers are used.<br>11 = Indicates a color pattern. The pattern data is sourced directly from the pattern data registers. |
| 7:0 | ROP | **Raster Operation:** Specifies the raster operation for pattern, source, and destination data. |
| **Note:** Writing to this register launches a raster operation. | | |
| **GX_BASE+8204h-8207h** | **GP_VECTOR_MODE Register (R/W)** | **Default Value = 00000000h** |
| 31:4 | RSVD | **Reserved:** Set to 0. |
| 3 | DEST | **Read Destination Data:** Indicates that frame-buffer destination data is required. |
| 2 | DMIN | **Minor Direction:** Indicates a positive minor axis step. |
| 1 | DMAJ | **Major Direction:** Indicates a positive major axis step. |
| 0 | YMAJ | **Major Direction:** Indicates a Y major vector. |
| **GX_BASE+8208h-820Bh** | **GP_BLT_MODE Register (R/W)** | **Default Value = 00000000h** |
| 31:9 | RSVD | **Reserved:** Set to 0. |
| 8 | Y | **Reverse Y Direction:** Indicates a negative increment for the Y position. This bit is used to control the direction of screen to screen BLTs to prevent data corruption in overlapping windows. |
| 7:6 | SM | **Source Mode:** Specifies the format of the source data.<br>00 = Source is a color bitmap.<br>01 = Source is a monochrome bitmap (use source color expansion).<br>10 = Unused.<br>11 = Source is a text glyph (use source color expansion). This differs from a monochrome bitmap in that the X position is adjusted by the width of the BLT and the Y position remains the same. |
| 5 | RSVD | **Reserved:** Set to 0. |
| 4:2 | RD | **Destination Data:** Specifies the destination data location.<br>000 = No destination data is required. The destination data into the raster operation unit is all ones.<br>010 = Read destination data from BLT Buffer 0.<br>011 = Read destination data from BLT Buffer 1.<br>100 = Read destination data from the frame buffer (store temporarily in BLT Buffer 0).<br>101 = Read destination data from the frame buffer (store temporarily in BLT Buffer 1). |
| 1:0 | RS | **Source Data:** Specifies the source data location.<br>00 = No source data is required. The source data into the raster operation unit is all ones.<br>01 = Read source data from the frame buffer (temporarily stored in BLT Buffer 0).<br>10 = Read source data from BLT Buffer 0.<br>11 = Read source data from BLT Buffer 1. |
| **Note:** Writing to this register launches a BLT operation. | | |
| **GX_BASE+820Ch-820Fh** | **GP_BLT_STATUS Register (R/W)** | **Default Value = 00000000h** |
| 31:10 | RSVD | **Reserved:** Set to 0. |
| 9 | W | **Screen Width:** Selects a frame-buffer width of 2048 bytes (default is 1024 bytes). This register must be programmed correctly in order for compression to work. |
| 8 | M | **16-bpp Mode:** Selects a pixel data format of 16-bpp (default is 8-bpp). |
| 7:3 | RSVD | **Reserved:** Set to 0. |
| 2 | BP (RO) | **BLT Pending (Read Only):** Indicates that a BLT operation is pending in the master registers.<br>The “BLT Pending” bit must be clear before loading any of the graphics pipeline registers. Loading registers when this bit is set high will destroy the values for the pending BLT. |

## Integrated Functions (Continued)

### Table 4-24. Graphics Pipeline Configuration Registers (Continued)

| Bit | Name | Description |
|---|---|---|
| 1 | PB (RO) | **Pipeline Busy (Read Only):** Indicates that the graphics pipeline is processing data.<br>The “Pipeline Busy” bit differs from the “BLT Busy” bit in that the former only indicates that the graphics pipeline is processing data. The “BLT Busy” bit also indicates that the memory controller has not yet processed all of the requests for the current operation.<br>The “Pipeline Busy” bit must be clear before loading a BLT buffer if the previous BLT operation used the same BLT buffer. |
| 0 | BB (RO) | **BLT Busy (Read Only):** Indicates that a BLT / vector operation is in progress.<br>The “BLT Busy” bit must be clear before accessing the frame buffer directly. |

**GX_BASE+8210h-8213h** **GP_VGA_BASE (R/W)** **Default Value = xxxxxxxxh**

Note that the registers at GX_BASE+8210h is located in the area designated for the graphics pipeline but is used for VGA emulation purposes. Refer to Table 4-39 on page 165 for this register’s bit formats.

**GX_BASE+8214h-8217h** **GP_VGA_LATCH Register (R/W)** **Default Value = xxxxxxxxh**

Note that the registers at GX_BASE+8214h is located in the area designated for the graphics pipeline but is used for VGA emulation purposes. Refer to Table 4-39 on page 165 for this register’s bit formats.

## Integrated Functions (Continued)

### 4.5 DISPLAY CONTROLLER

The GXLV processor incorporates a display controller that retrieves display data from the memory controller and formats it for output on a variety of display devices. The GXLV processor connects directly to the graphics Geode I/O companion. The display controller includes a display FIFO, compression/decompression (codec) hardware, hardware cursor, a 256-entry-by-18-bit palette RAM (plus three extension colors), display timing generator, dither and frame-rate-modulation circuitry for TFT panels, and versatile output formatting logic. A diagram of the display controller subsystem is shown in Figure 4-14.

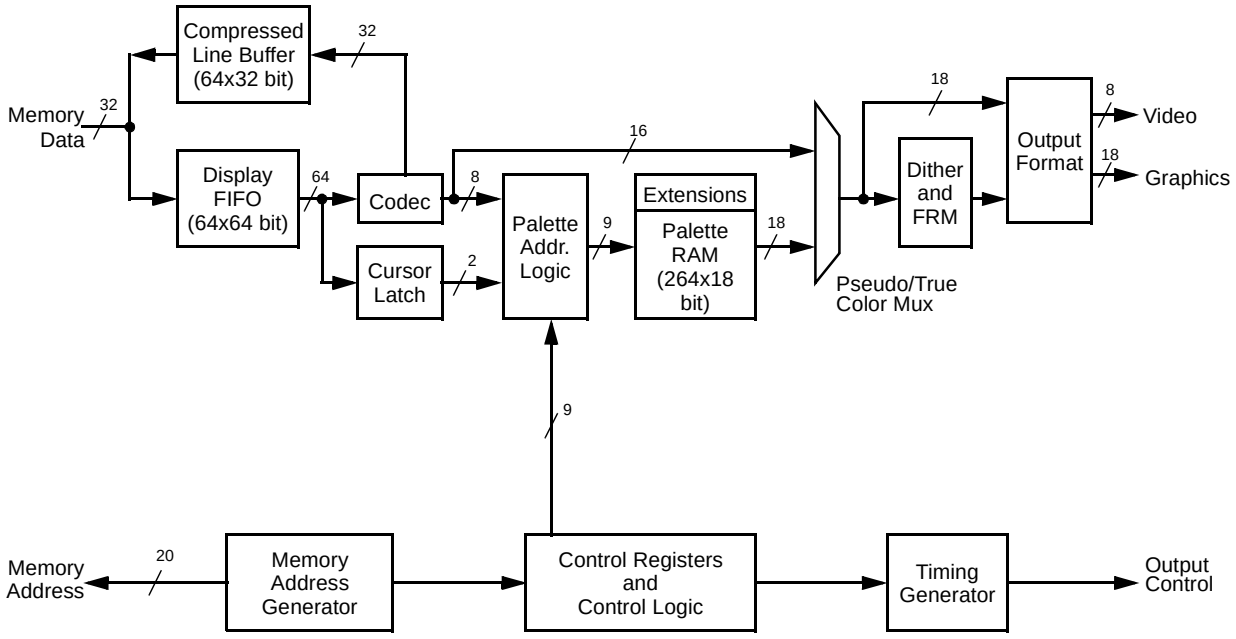



**Figure 4-14. Display Controller Block Diagram**

### 4.5.1 Display FIFO

The display controller contains a large (64x64 bit) FIFO for queuing up display data from the memory controller as required for output to the screen. The memory controller must arbitrate between display controller requests and other requests for memory access from the microprocessor core, L1 cache controller, and the graphics pipeline.

Display data is required in real time, making it the highest priority in the system. Without efficient memory management, system performance would suffer dramatically due to the constant display-refresh requests from the display controller. The large size of the display FIFO is desirable so that the FIFO may primarily be loaded during times when there is no other request pending to the DRAM controller which allows the memory controller to stay in page mode for a longer period of time when servicing the display FIFO. When a priority request from the cache or graphics pipeline occurs, if the display FIFO has enough data queued up, the DRAM controller can immediately service the request without concern that the display FIFO will underflow. If the display FIFO is below a programmable threshold, a high-priority request will be sent to the DRAM controller, which will take precedence over any other requests that are pending.

The display FIFO is 64 bits wide to accommodate high-speed burst read operations from the DRAM controller at maximum memory bandwidth. In addition to the normal pixel data stream, the display FIFO also queues up cursor patterns.

### 4.5.2 Compression Technology

To reduce the system memory contention caused by the display refresh, the display controller contains compression and decompression logic for compressing the frame buffer image in real time as it is sent to the display. It combines this compressed display buffer into the extra off-screen memory within the graphics memory aperture. Coherency of the compressed display buffer is maintained by use of dirty and valid bits for each line. The dirty and valid RAM is contained on-chip for maximum efficiency. Whenever a line has been validly compressed, it will be retrieved from the compressed display buffer for all future accesses until the line becomes dirty again. Dirty lines will be retrieved from the normal uncompressed frame buffer.

The compression logic has the ability to insert a programmable number of "static" frames, during which time dirty bits are ignored and the valid bits are read to determine whether a line should be retrieved from the frame buffer or compressed display buffer. The less frequently the dirty bits are sampled, the more frequently lines will be retrieved from the compressed display buffer. This allows a programmable screen image update rate (as opposed to refresh rate). Generally, an update rate of 30 frames per second is adequate for displaying most types of data, including real-time video. If a flat panel display is used that has a slow response time, such as 100 ms, the image need not be updated faster than ten frames per second, since the panel could not display changes beyond that rate.

The compression algorithm used in the GXLV processor commonly achieves compression ratios between 10:1 and 20:1, depending on the nature of the display data. This high level of compression provides higher system performance by reducing typical latency for normal system memory access, higher graphics performance by increasing available drawing bandwidth to the DRAM array, and much lower power consumption by significantly reducing the number of off-chip DRAM accesses required for refreshing the display. These advantages become even more pronounced as display resolution, color depth, and refresh rate are increased and as the size of the installed DRAM increases.

As uncompressed lines are fed to the display, they will be compressed and stored in an on-chip compressed line buffer (64x32 bits). Lines will not be written back to the compressed display buffer in the DRAM unless a valid compression has resulted, so there is no penalty for pathological frame buffer images where the compression algorithm breaks down.

## Integrated Functions (Continued)

### 4.5.3 Hardware Cursor

The display controller contains hardware cursor logic to allow overlay of the cursor image onto the pixel data stream. Overhead for updating this image on the screen is kept to a minimum by requiring that only the X and Y position be changed. This eliminates "submarining" effects commonly associated with software cursors. The cursor, 32x32 pixels with 2-bpp, is loaded into off-screen memory within the graphics memory aperture. The DC_CUR_ST_OFFSET programs the cursor start (see Table 4-30 on page 148). The 2-bit code selects color 0, color 1, transparent, or background-color inversion for each pixel in the cursor. The two cursor colors will be stored as extensions to the normal 256-entry palette at locations 100h and 101h.

The 2-bit cursor codes are as follows:

| AND | XOR | Displayed |
|---|---|---|
| 0 | 0 | Cursor Color 0 |
| 0 | 1 | Cursor Color 1 |
| 1 | 0 | Transparent – Background Pixel |
| 1 | 1 | Inverted – Bit-wise Inversion of Background Pixel |

The cursor overlay patterns are loaded to independent memory locations, usually mapped above the frame buffer and compressed display buffer (off-screen). The cursor buffer must start on a DWORD boundary. It is linearly mapped, and is always 256 bytes in size. If there is enough room (256 bytes) after the compression-buffer line but before the next frame-buffer line starts, the cursor pattern may be loaded into this area to make efficient use of the graphics memory.

Each pattern is a 32x32-pixel array of 2-bit codes. The codes are a combination of AND mask and XOR mask for a particular pixel. Each line of an overlay pattern is stored as two DWORDs, with each DWORD containing the AND masks for 16 pixels in the upper word and the XOR masks for 16 pixels in the lower word. DWORDs are arranged with the leftmost pixel block being least significant and the rightmost pixel block being most significant. Pixels within words are arranged with the leftmost pixels being most significant and the rightmost pixels being least significant. Multiple cursor patterns may be loaded into the off-screen memory. An application may simply change the cursor start offset to select a new cursor pattern. The new cursor pattern will become effective at the start of the next frame scan.

### 4.5.4 Display Timing Generator

The display controller features a fully programmable timing generator for generating all timing control signals for the display. The timing control signals include horizontal and vertical sync and blank signals in addition to timing for active and overscan regions of the display. The timing generator is similar in function to the CRTC of the original VGA, although programming is more straightforward. Programming of the timing registers are supported by National via a BIOS INT10 call during a mode set. When programming the timing registers directly, extreme care should be taken to ensure that all timing is compatible with the display device.

The timing generator supports overscan to maintain full backward compatibility with the VGA standard. This feature is supported primarily for CRT display devices since flat panel displays have fixed resolutions and do not provide for overscan. When a display mode is selected having a lower resolution than the panel resolution, the GXLV processor supports a mechanism to center the display by stretching the border to fill the remainder of the screen. The border color is at palette extension 104h.

### 4.5.5 Dither and Frame Rate Modulation

The display controller supports 2x2 dither and two-level frame rate modulation (FRM) to increase the apparent number of colors displayed on 9-bit or 12-bit TFT panels. Dither and FRM are individually programmable. With dithering and FRM enabled, 185,193 colors are possible on a 9-bit TFT panel, and 226,981 colors are possible on a 12-bit TFT panel.

### 4.5.6 Display Modes

The GXLV processor's display controller is programmable and supports resolutions up to 1024x768 at 16 bits per pixel and resolutions up to 1280x1024 at 8 bits per pixel. This means the GXLV processor supports the standard display resolutions of 640x480, 800x600, and 1024x768 display resolutions at both 8 and 16 bits per pixel and 1280x1024 resolution at 8 bits per pixel only. Two 16-bit display formats are supported: RGB 5-6-5 and RGB 5-5-5. Table 4-26 lists how the RGB data is mapped onto the pixel data bus for the CRT and various TFT interfaces. All CRT modes can have VESA-compatible timing. Table 4-25 lists some of the supported TFT panel display modes and Table 4-27 lists some of the supported CRT display modes.

## Integrated Functions (Continued)

.

**Table 4-25. TFT Panel Display Modes (Note 1)**

| Resolution | Simultaneous Colors | Refresh Rate (Hz) | DCLK Rate (MHz) (Note 2) | PCLK Rate (MHz) (Note 3) | Panel Type | Maximum Displayed Colors (Note 4) |
|---|---|---|---|---|---|---|
| 640x480 (Note 5) | 8-bpp 256 colors out of a palette of 256 | 60 | 50.35 | 25.175 | 9-bit | $57^3$ = 185,193 |
| | | | | | 12-bit | $61^3$ = 226,981 |
| | | | | | 18-bit | $4^3$ = 262,144 |
| | 16-bpp 64 KB colors 5-6-5 | 60 | 50.35 | 25.175 | 9-bit | 29x57x29 = 47,937 |
| | | | | | 12-bit | 31x61x31 = 58,621 |
| | | | | | 18-bit | 32x64x32 = 65,535 |
| 800x600 (Note 5) | 8-bpp 256 colors out of a palette of 256 | 60 | 80.0 | 40.0 | 9-bit | $57^3$ = 185,193 |
| | | | | | 12-bit | $61^3$ = 226,981 |
| | | | | | 18-bit | $64^3$ = 262,144 |
| | 16-bpp 64 KB Colors 5-6-5 | 60 | 80.0 | 40.0 | 9-bit | 29x57x29 = 47,937 |
| | | | | | 12-bit | 31x61x31 = 58,621 |
| | | | | | 18-bit | 32x64x32 = 65,535 |
| 1024x768 | 8-bpp 256 colors out of a palette of 256 | 60 | 65 | 65.0 | 9-bit/18-I/F | $57^3$ = 185,193 |
| | 16-bpp 64 KB colors 5-6-5 | 60 | 65 | 65.0 | 9-bit/18-I/F | 29x57x29 = 47,937 |

**Notes:**
1. This list is not meant to be an complete list of all the possible supported TFT display modes.
2. DCLK is the input clock from the Geode I/O companion. In some cases, DCLK is doubled to keep the Geode I/O companion's PLL in a desired operational range.
3. PCLK is the graphics output clock to the Geode I/O companion.
4. 9-bit and 12-bit panels use FRM and dither to increase displayed colors. (See Section 4.5.5 "Dither and Frame Rate Modulation" on page 136.)
5. All 640x480 and 800x600 modes can be run in simultaneous display with CRT

## Integrated Functions (Continued)

**Table 4-26. CRT and TFT Panel Data Bus Formats**

| Panel Data Bus Bit | CRT & 18-Bit TFT | 12-Bit TFT | 9-Bit TFT | | |
|---|---|---|---|---|---|
| | | | 640x480 | 1024x768 | |
| 17 | R5 | R5 | R5 | R5 | Even |
| 16 | R4 | R4 | R4 | R4 | |
| 15 | R3 | R3 | R3 | R3 | |
| 14 | R2 | R2 | | R5 | Odd |
| 13 | R1 | | | R4 | |
| 12 | R0 | | | R3 | |
| 11 | G5 | G5 | G5 | G5 | Even |
| 10 | G4 | G4 | G4 | G4 | |
| 9 | G3 | G3 | G3 | G3 | |
| 8 | G2 | G2 | | G5 | Odd |
| 7 | G1 | | | G4 | |
| 6 | G0 | | | G3 | |
| 5 | B5 | B5 | B5 | B5 | Even |
| 4 | B4 | B4 | B4 | B4 | |
| 3 | B3 | B3 | B3 | B3 | |
| 2 | B2 | B2 | | B5 | Odd |
| 1 | B1 | | | B4 | |
| 0 | B0 | | | B3 | |

## Integrated Functions (Continued)

**Table 4-27. CRT Display Modes (Note 1)**

| Resolution | Simultaneous Colors | Refresh Rate (Hz) | DCLK Rate (MHz) (Note 2) | PCLK Rate (MHz) (Note 3) |
|---|---|---|---|---|
| 640x480 | 8-bpp 256 colors out of a palette of 256 | 60 | 50.35 | 25.175 |
| | | 72 | 63.0 | 31.5 |
| | | 75 | 63.0 | 31.5 |
| | | 85 | 72.0 | 36.0 |
| | 16-bpp 64 KB colors RGB 5-6-5 | 60 | 50.35 | 25.175 |
| | | 72 | 63.0 | 31.5 |
| | | 75 | 63.0 | 31.5 |
| | | 85 | 72.0 | 36.0 |
| 800x600 | 8-bpp 256 colors out of a palette of 256 | 60 | 80.0 | 40.0 |
| | | 72 | 100.0 | 50.0 |
| | | 75 | 99.0 | 49.5 |
| | | 85 | 112.5 | 56.25 |
| | 16-bpp 64 KB colors RGB 5-6-5 | 60 | 80.0 | 40.0 |
| | | 72 | 100.0 | 50.0 |
| | | 75 | 99 | 49.9 |
| | | 85 | 112.5 | 56.25 |
| 1024x768 | 8-bpp 256 colors out of a palette of 256 | 60 | 65.0 | 65.0 |
| | | 70 | 75.0 | 75.0 |
| | | 75 | 78.5 | 78.5 |
| | | 85 | 94.5 | 94.5 |
| | 16-bpp 64 KB colors RGB 5-6-5 | 60 | 65.0 | 65.0 |
| | | 70 | 75.0 | 75.0 |
| | | 75 | 78.5 | 78.5 |
| | | 85 | 94.5 | 94.5 |
| 1280x1024 | 8-bpp 256 colors out of a palette of 256 | 60 | 108.0 | 108.0 |
| | | 75 | 135.0 | 135 |

**Notes:** 1. This list is not meant to be an complete list of all the possible supported CRT display modes.

2. DCLK is the input clock from the Geode I/O companion. In some cases, DCLK is doubled to keep the Geode I/O companion's PLL in a desired operational range.

3. PCLK is the graphics output clock to the Geode I/O companion.

## Integrated Functions (Continued)

### 4.5.7 Graphics Memory Map

The GXLV processor supports a maximum of 4 MB of graphics memory and will map it to an address space (see Figure 4-2 on page 98) higher than the maximum amount of installed RAM. The graphics memory aperture physically resides at the top of the installed system RAM. The start address and size of the graphics memory aperture are programmable on 512 KB boundaries. Typically, the system BIOS sets the size and start address of the graphics memory aperture during the boot process based on the amount of installed RAM, user defined CMOS settings, hard coded, etc. The graphics pipeline and display controller address the graphics memory with a 20-bit offset (address bits [21:2]) and four byte enables into the graphics memory aperture. The graphics memory stores several buffers that are used to generate the display: the frame buffer, compressed display buffer, VGA memory, and cursor pattern(s). Any remaining off-screen memory within the graphics aperture may be used by the display driver as desired or not at all.

#### 4.5.7.1 DC Memory Organization Registers

The display controller contains a number of registers that allow full programmability of the graphics memory organization. This includes starting offsets for each of the buffer regions described above, line delta parameters for the frame buffer and compression buffer, as well as compressed line-buffer size information. The starting offsets for the various buffers are programmable for a high degree of flexibility in memory organization.

#### 4.5.7.2 Frame Buffer and Compression Buffer Organization

The GXLV processor supports primary display modes 640x480, 800x600, and 1024x768 at both 8-bpp and 16-bpp, and 1280x1024 at 8-bpp. Pixels are packed into DWORDs as shown in Figure 4-15.

In order to simplify address calculations by the rendering hardware, the frame buffer is organized in an XY fashion where the offset is simply a concatenation of the X and Y pixel addresses. All 8-bpp display modes with the exception of the 1280x1024 resolution will use a 1024-byte line delta between the starting offsets of adjacent lines. All 16-bpp display modes and 1280x1024x8-bpp display modes will use a 2048-byte line delta between the starting offsets of adjacent lines. If there is room, the space between the end of a line and the start of the next line will be filled with the compressed display data for that line, thus allowing efficient memory utilization. For 1024x768 display modes, the frame-buffer line size is the same as the line delta, so no room is left for the compressed display data between lines. In this case, the compressed display buffer begins at the end of the frame buffer region and is linearly mapped.

**DWORD**

| Bit Position | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Address | 3h | | | | | | | | 2h | | | | | | | | 1h | | | | | | | | 0h | | | | | | | |
| Pixel Org - 8-bpp | (3,0) | | | | | | | | (2,0) | | | | | | | | (1,0) | | | | | | | | (0,0) | | | | | | | |
| Pixel Org - 16-bpp | (1,0) | | | | | | | | | | | | | | | | (0,0) | | | | | | | | | | | | | | | |

**Figure 4-15. Pixel Arrangement Within a DWORD**

#### 4.5.7.3 VGA Display Support

The graphics pipeline contains full hardware support for the VGA front end. The VGA data is stored in a 256 KB buffer located in graphics memory. The main task for Virtual VGA (see Section 4.6 “Virtual VGA Subsystem” on page 157) is converting the data in the VGA buffer to an 8-bpp frame buffer that can be displayed by the display controller.

For some modes, the display controller can display the VGA data directly and the data conversion is not necessary. This includes standard VGA mode 13h and the variations of that mode used in several games; the display controller can also directly display VGA planar graphics modes D, E, F, 10, 11, and 12. Likewise, the hardware can directly display all of the higher-resolution VESA modes. Since the frame buffer data is written directly to memory instead of travelling across an external bus, the GXLV processor often outperforms VGA cards for these modes.

The display controller, however, does not directly support text modes. SoftVGA must convert the characters and attributes in the VGA buffer to an 8-bpp frame buffer image the hardware uses for display refresh.

### 4.5.8 Display Controller Registers

The Display Controller maps 100h memory locations starting at GX_BASE+8300h for the display controller registers. Refer to Section 4.1.2 “Control Registers” on page 99 for instructions on accessing these registers.

The Display Controller Registers are divided into six categories:

- Configuration and Status Registers
- Memory Organization Registers
- Timing Registers
- Cursor and Line Compare Registers
- Color Registers
- Palette and RAM Diagnostic Registers

Table 4-28 summarizes these registers and locations, and the following subsections give detailed register/bit formats.

**Table 4-28. Display Controller Register Summary**

| GX_BASE+ Memory Offset | Type | Name/Function | Default Value |
|---|---|---|---|
| **Configuration and Status Registers** | | | |
| 8300h-8303h | R/W | **DC_UNLOCK**<br>Display Controller Unlock: This register is provided to lock the most critical memory-mapped display controller registers to prevent unwanted modification (write operations). Read operations are always allowed. | 00000000h |
| 8304h-8307h | R/W | **DC_GENERAL_CFG**<br>Display Controller General Configuration: General control bits for the display controller. | 00000000h |
| 8308h-830Bh | R/W | **DC_TIMING_CFG**<br>Display Controller Timing Configuration: Status and control bits for various display timing functions. | xx000000h |
| 830Ch-830Fh | R/W | **DC_OUTPUT_CFG**<br>Display Controller Output Configuration: Status and control bits for pixel output formatting functions. | xx000000h |
| **Memory Organization Registers** | | | |
| 8310h-8313h | R/W | **DC_FB_ST_OFFSET**<br>Display Controller Frame Buffer Start Address: Specifies offset at which the frame buffer starts. | xxxxxxxxh |
| 8314h-8317h | R/W | **DC_CB_ST_OFFSET**<br>Display Controller Compression Buffer Start Address: Specifies offset at which the compressed display buffer starts. | xxxxxxxxh |
| 8318h-831Bh | R/W | **DC_CUR_ST_OFFSET**<br>Display Controller Cursor Buffer Start Address: Specifies offset at which the cursor memory buffer starts. | xxxxxxxxh |
| 831Ch-831Fh | -- | **Reserved** | 00000000h |
| 8320h-8323h | R/W | **DC_VID_ST_OFFSET**<br>Display Controller Video Start Address: Specifies offset at which the video buffer starts. | xxxxxxxxh |
| 8324h-8327h | R/W | **DC_LINE_DELTA**<br>Display Controller Line Delta: Stores line delta for the graphics display buffers. | xxxxxxxxh |

## Integrated Functions (Continued)

### Table 4-28. Display Controller Register Summary (Continued)

| GX_BASE+ Memory Offset | Type | Name/Function | Default Value |
|---|---|---|---|
| 8328h-832Bh | R/W | **DC_BUF_SIZE**<br>Display Controller Buffer Size: Specifies the number of bytes to transfer for a line of frame buffer data and the size of the compressed line buffer. (The compressed line buffer will be invalidated if it exceeds the CB_LINE_SIZE, bits [15:9].) | xxxxxxxxh |
| 832Ch-832Fh | -- | **Reserved** | 00000000h |
| **Timing Registers** | | | |
| 8330h-8333h | R/W | **DC_H_TIMING_1**<br>Display Controller Horizontal and Total Timing: Horizontal active and total timing information. | xxxxxxxxh |
| 8334h-8337h | R/W | **DC_H_TIMING_2**<br>Display Controller CRT Horizontal Blanking Timin: CRT horizontal blank timing information. | xxxxxxxxh |
| 8338h-833Bh | R/W | **DC_H_TIMING_3**<br>Display Controller CRT Sync Timing: CRT horizontal sync timing information. Note, however, that this register should also be programmed appropriately for flat panel only display since the horizontal sync transition determines when to advance the vertical counter. | xxxxxxxxh |
| 833Ch-833Fh | R/W | **DC_FP_H_TIMING**<br>Display Controller Flat Panel Horizontal Sync Timing: Horizontal sync timing information for an attached flat panel display. | xxxxxxxxh |
| 8340h-8343h | R/W | **DC_V_TIMING_1**<br>Display Controller Vertical and Total Timing: Vertical active and total timing information. The parameters pertain to both CRT and flat panel display. | xxxxxxxxh |
| 8344h-8247h | R/W | **DC_V_TIMING_2**<br>Display Controller CRT Vertical Blank Timing: Vertical blank timing information. | xxxxxxxxh |
| 8348h-834Bh | R/W | **DC_V_TIMING_3**<br>Display Controller CRT Vertical Sync Timing: CRT vertical sync timing information. | xxxxxxxxh |
| 834Ch-834Fh | R/W | **DC_FP_V_TIMING**<br>Display Controller Flat Panel Vertical Sync Timing: Flat panel vertical sync timing information. | xxxxxxxxh |
| **Cursor and Line Compare Registers** | | | |
| 8350h-8353h | R/W | **DC_CURSOR_X**<br>Display Controller Cursor X Position: X position information of the hardware cursor. | xxxxxxxxh |
| 8354h-8357h | RO | **DC_V_LINE_CNT**<br>Display Controller Vertical Line Count: This read only register provides the current scanline for the display. It is used by software to time update of the frame buffer to avoid tearing artifacts. | xxxxxxxxh |
| 8358h-835Bh | R/W | **DC_CURSOR_Y**<br>Display Controller Cursor Y Position: Y position information of the hardware cursor. | xxxxxxxxh |
| 835Ch-835Fh | R/W | **DC_SS_LINE_CMP**<br>Display Controller Split-Screen Line Compare: Contains the line count at which the lower screen begins in a VGA split-screen mode. | xxxxxxxxh |
| 8360h-8363h | -- | **Reserved** | xxxxxxxxh |
| 8364h-8367h | -- | **Reserved** | xxxxxxxxh |
| 8368h-836Bh | -- | **Reserved** | xxxxxxxxh |
| 836Ch-836Fh | -- | **Reserved** | xxxxxxxxh |

**Table 4-28. Display Controller Register Summary (Continued)**

| GX_BASE+ Memory Offset | Type | Name/Function | Default Value |
|---|---|---|---|
| **Palette and RAM Diagnostic Registers** | | | |
| 8370h-8373h | R/W | **DC_PAL_ADDRESS**<br>Display Controller Palette Address: This register should be written with the address (index) location to be used for the next access to the DC_PAL_DATA register. | xxxxxxxxh |
| 8374h-8377h | R/W | **DC_PAL_DATA**<br>Display Controller Palette Data: Contains the data for a palette access cycle. | xxxxxxxxh |
| 8378h-837Bh | R/W | **DC_DFIFO_DIAG**<br>Display Controller Display FIFO Diagnostic: This register is provided to enable testability of the Display FIFO RAM. | xxxxxxxxh |
| 837Ch-837Fh | R/W | **DC_CFIFO_DIAG**<br>Display Controller Compression FIFO Diagnostic: This register is provided to enable testability of the Compressed Line Buffer (FIFO) RAM. | xxxxxxxxh |

## Integrated Functions (Continued)

### 4.5.8.1 Configuration and Status Registers

The Configuration and Status Registers group consists of four 32-bit registers located at GX_BASE+8300h-830Ch. These registers are described below and Table 4-29 gives their bit formats.

**Table 4-29. Display Controller Configuration and Status Registers**

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+8300h-8303h** | | **DC_UNLOCK Register (R/W)** **Default Value = 00000000h** |
| 31:16 | RSVD | **Reserved:** Set to 0. |
| 15:0 | UNLOCK_ CODE | **Unlock Code:** This register must be written with the value 4758h in order to write to the protected registers. The following registers are protected by the locking mechanism. Writing any other value enables the write lock function.<br>DC_GENERAL_CFG, DC_TIMING_CFG, DC_OUTPUT_CFG, DC_FB_ST_OFFSET, DC_CB_ST_OFFSET, DC_CUR_ST_OFFSET, DC_VID_ST_OFFSET, DC_LINE_DELTA, DC_BUF_SIZE, DC_H_TIMING_1, DC_H_TIMNG_2, DC_H_TIMING_3, DC_FP_H_TIMING, DC_V_TIMING_1, DC_V_TIMING_2, DC_V_TIMING_3, DC_FP_V_TIMING |
| **GX_BASE+8304h-8307h** | | **DC_GENERAL_CFG (R/W) (Locked)** **Default Value = 00000000h** |
| 31 | DDCK | **Divide Dot Clock:** Divide internal DCLK by two relative to PCLK:<br>0 = Disable; 1 = Enable. |
| 30 | DPCK | **Divide Pixel Clock:** Divide PCLK by two relative to internal DCLK:<br>0 = Disable; 1 = Enable. |
| 29 | VRDY | **Video Ready Protocol:** 0 = Low speed video port: 1 = High speed video port.<br>Always program to 1. |
| 28 | VIDE | **Video Enable:** Motion video port: 0 = Disable; 1 = Enable. |
| 27 | SSLC | **Split-screen Line Compare:** VGA line compare function: 0 = Disable; 1 = Enable.<br>When enabled, the internal line counter will be compared to the value programmed in the DC_SS _LINE_CMP register. If it matches, the frame buffer address will be reset to zero. This enables a split screen function. |
| 26 | CH4S | **Chain 4 Skip:** Allow display controller to read every 4th DWORD from the frame buffer for compatibility with the VGA: 0 = Disable; 1 = Enable. |
| 25 | DIAG | **FIFO Diagnostic Mode:** This bit allows testability of the on-chip Display FIFO and Compressed Line Buffer via the diagnostic access registers. A low-to-high transition will reset the Display FIFO's R/W pointers and the Compressed Line Buffer's read pointer. 0 = Normal operation; 1 = Enable. |
| 24 | LDBL | **Line Double:** Allow line doubling for emulated VGA modes: 0 = Disable; 1 = Enable.<br>If enabled, this will cause each odd line to be replicated from the previous line as the data is sent to the display. Timing parameters should be programmed as if pixel doubling is not used, however, the frame buffer should be loaded with half the normal number of lines. |
| 23:19 | RSVD | **Reserved:** Set to 0. |
| 18 | FDTY | **Frame Dirty Mode:** Allow entire frame to be flagged as dirty whenever a pixel write occurs to the frame buffer (this is provided for modes that use a linearly mapped frame buffer for which the line delta is not equal to 1024 or 2048 bytes): 0 = Disable; 1 = Enable.<br>When disabled, dirty bits are set according to the Y address of the pixel write. |
| 17 | RSVD | **Reserved:** Set to 0. |
| 16 | CMPI | **Compressor Insert Mode:** Insert one static frame between update frames: 0 = Disable; 1 = Enable.<br>An update frame is a frame in which dirty lines are updated. Conversely, a static frame is a frame in which dirty lines are not updated (the display image may not actually be static, because lines that are not compressed successfully must be retrieved from the uncompressed frame buffer). |
| 15:12 | DFIFO HI-PRI END LVL | **Display FIFO High Priority End Level:** This field specifies the depth of the display FIFO (in 64-bit entries x 4) at which a high-priority request previously issued to the memory controller will end. The value is dependent upon display mode.<br>This register should always be non-zero and should be larger than the start level. |

## Integrated Functions (Continued)

### Table 4-29. Display Controller Configuration and Status Registers (Continued)

| Bit | Name | Description |
|---|---|---|
| 11:8 | DFIFO HI-PRI START LVL | **Display FIFO High Priority Start Level:** This field specifies the depth of the display FIFO (in 64-bit entries x 4) at which a high-priority request will be sent to the memory controller to fill up the FIFO. The value is dependent upon display mode.<br>This register should always be nonzero and should be less than the high-priority end level. |
| 7:6 | DCLK_ MUL | **DCLK Multiplier:** This 2-bit field specifies the clock multiplier for the input DCLK pin. After the input clock is optionally multiplied, the internal DCLK and PCLK may be divided as necessary.<br>00 = Forced Low<br>01 = DCLK ÷ 2<br>10 = DCLK<br>11 = 2 x DCLK |
| 5 | DECE | **Decompression Enable:** Allow operation of internal decompression hardware:<br>0 = Disable; 1 = Enable. |
| 4 | CMPE | **Compression Enable:** Allow operation of internal compression hardware: 0 = Disable; 1 = Enable |
| 3 | PPC | **Pixel Panning Compatibility:** This bit has the same function as that found in the VGA.<br>Allow pixel alignment to change when crossing a split-screen boundary - it will force the pixel alignment to be 16-byte aligned: 0 = Disable; 1 = Enable.<br>If disabled, the previous alignment will be preserved when crossing a split-screen boundary. |
| 2 | DVCK | **Divide Video Clock:** Selects frequency of VID_CLK pin:<br>0 = VID_CLK pin frequency is equal to one-half (½) the frequency of the core clock.<br>1 = VID_CLK pin frequency is equal to one-fourth (¼) the frequency of the core clock.<br>**Note:** Bit 28 (VIDE) must be set to 1 for this bit to be valid. |
| 1 | CURE | **Cursor Enable:** Use internal hardware cursor: 0 = Disable; 1 = Enable. |
| 0 | DFLE | **Display FIFO Load Enable:** Allow the display FIFO to be loaded from memory:<br>0 = Disable; 1 = Enable.<br>If disabled, no write or read operations will occur to the display FIFO.<br>If enabled, a flat panel should be powered down prior to setting this bit low. Similarly, if active, a CRT should be blanked prior to setting this bit low. |
| **GX_BASE+8308h-830Bh** | | **DC_TIMING_CFG Register (R/W) (Locked)** **Default Value = xxx00000h** |
| 31 | VINT (RO) | **Vertical Interrupt (Read Only):** Is a vertical interrupt pending? 0 = No; 1 = Yes.<br>This bit is provided to maintain backward compatibility with the VGA. It corresponds to VGA port 3C2h bit 7. |
| 30 | VNA (RO) | **Vertical Not Active (Read Only):** Is the active part of a vertical scan is in progress (i.e., retrace, blanking, or border)? 0 = Yes; 1 = No.<br>This bit is provided to maintain backward compatibility with the VGA. It corresponds to VGA port 3BA/3DA bit 3. |
| 29 | DNA (RO) | **Display Not Active (Read Only):** Is the active part of a line is being displayed (i.e., retrace, blanking, or border)? 0 = Yes; 1 = No.<br>This bit is provided to maintain backward compatibility with the VGA. It corresponds to VGA port 3BA/3DA bit 0. |
| 28 | RSVD | **Reserved:** Set to 0. |
| 27 | DDCI (RO) | **DDC Input (Read Only):** This bit returns the value from the DDCIN pin that should reflect the value from pin 12 of the VGA connector. It is used to provide support for the VESA Display Data Channel standard level DDC1. |
| 26:20 | RSVD | **Reserved:** Set to 0. |
| 19:17 | RSVD | **Reserved:** Set to 0. |
| 16 | BKRT | **Blink Rate:**<br>0 = Cursor blinks on every 16 frames for a duration of 8 frames (approximately 4 times per second) and VGA text characters will blink on every 32 frames for a duration of 16 frames (approximately 2 times per second).<br>1 = Cursor blinks on every 32 frames for a duration of 16 frames (approximately 2 times per second) and VGA text characters blink on every 64 frames for a duration of 32 frames (approximately 1 time per second).<br>Blinking is enabled by BLNK bit 7. |

## Integrated Functions (Continued)

### Table 4-29. Display Controller Configuration and Status Registers (Continued)

| Bit | Name | Description |
|---|---|---|
| 15 | PXDB | **Pixel Double:** Allow pixel doubling to stretch the displayed image in the horizontal dimension:<br>0 = Disable; 1 = Enable.<br>If bit 15 is enabled, timing parameters should be programmed as if no pixel doubling is used, however, the frame buffer should be loaded with half the normal pixels per line. Also, the FB_LINE_SIZE parameter in DC_BUF_SIZE should be set for the number of bytes to be transferred for the line rather than the number displayed. |
| 14 | INTL | **Interlace Scan:** Allow interlaced scan mode:<br>0 = Disable (Non-interlaced scanning is supported.)<br>1 = Enable (If a flat panel is attached, it should be powered down before setting this bit.) |
| 13 | PLNR | **VGA Planar Mode:** This bit must be set high for all VGA planar display modes. |
| 12 | FCEN | **Flat Panel Center:** Allows the border and active portions of a scan line to be qualified as “active” to a flat panel display via the ENADISP signal. This allows the use of a large border region for centering the flat panel display. 0 = Disable; 1 = Enable.<br>When disabled, only the normal active portion of the scan line will be qualified as active. |
| 11 | FVSP | **Flat Panel Vertical Sync Polarity:**<br>0 = Causes TFT vertical sync signal to be normally low, generating a high pulse during sync interval.<br>1 = Causes TFT vertical sync signal to be normally high, generating a low pulse during sync interval. |
| 10 | FHSP | **Flat Panel Horizontal Sync Polarity:**<br>0 = Causes TFT horizontal sync signal to be normally low, generating a high pulse during sync interval.<br>1 = Causes TFT horizontal sync signal to be normally high, generating a low pulse during sync interval. |
| 9 | CVSP | **CRT Vertical Sync Polarity:**<br>0 = Causes CRT_VSYNC signal to be normally low, generating a high pulse during the retrace interval.<br>1 = Cause CRT_VSYNC signal to be normally high, generating a low pulse during the retrace interval. |
| 8 | CHSP | **CRT Horizontal Sync Polarity:**<br>0 = Causes CRT_HSYNC signal to be normally low, generating a high pulse during the retrace interval.<br>1 = Causes CRT_HSYNC signal to be normally high, generating a low pulse during the retrace interval. |
| 7 | BLNK | **Blink Enable:** Blink circuitry: 0 = Disable; 1 = Enable.<br>If enabled, the hardware cursor will blink as well as any pixels. This is provided to maintain compatibility with VGA text modes. The blink rate is determined by the bit 16 (BKRT). |
| 6 | VIEN | **Vertical Interrupt Enable:** Generate a vertical interrupt on the occurrence of the next vertical sync pulse:<br>0 = Disable, vertical interrupt is cleared;<br>1 = Enable.<br>This bit is provided to maintain backward compatibility with the VGA. |
| 5 | TGEN | **Timing Generator Enable:** Allow timing generator to generate the timing control signals for the display.<br>0 = Disable, the Timing Registers may be reprogrammed, and all circuitry operating on the DCLK will be reset.<br>1 = Enable, no write operations are permitted to the Timing Registers. |
| 4 | DDCK | **DDC Clock:** This bit is used to provide the serial clock for reading the DDC data pin. This bit is multiplexed onto the CRT_VSYNC pin, but in order for it to have an effect, the VSYE bit[2] must be set low to disable the normal vertical sync. Software should then pulse this bit high and low to clock data into the GXLV processor.<br>This feature is provided to allow support for the VESA Display Data Channel standard level DDC1. |
| 3 | BLKE | **Blank Enable:** Allow generation of the composite blank signal to the display device:<br>0 = Disable; 1 = Enable.<br>When disabled, the ENA_DISP output will be a static low level. This allows VESA DPMS compliance. |
| 2 | HSYE | **Horizontal Sync Enable:** Allow generation of the horizontal sync signal to a CRT display device:<br>0 = Disable; 1 = Enable.<br>When disabled, the HSYNC output will be a static low level. This allows VESA DPMS compliance.<br>Note that this bit only applies to the CRT; the flat panel HSYNC is controlled by the automatic power sequencing logic. |

**Table 4-29. Display Controller Configuration and Status Registers (Continued)**

| Bit | Name | Description |
|---|---|---|
| 1 | VSYE | **Vertical Sync Enable:** Allow generation of the vertical sync signal to a CRT display device:<br>0 = Disable; 1 = Enable.<br>When disabled, the VSYNC output will be a static low level. This allows VESA DPMS compliance.<br>Note that this bit only applies to the CRT; the flat panel VSYNC is controlled by the automatic power sequencing logic. |
| 0 | PPE | **Pixel Port Enable:** On a low-to-high transition this bit will enable the pixel port outputs.<br>On a high-to-low transition, this bit will disable the pixel port outputs. |
| **GX_BASE+830Ch-830Fh** | **DC_OUTPUT_CFG Register (R/W) (Locked)** | **Default Value = xxx00000h** |
| 31:16 | RSVD | **Reserved:** Set to 0. |
| 15 | DIAG | **Compressed Line Buffer Diagnostic Mode:** This bit allows testability of the Compressed Line Buffer via the diagnostic access registers. A low-to-high transition resets the Compressed Line Buffer write pointer. 0 = Disable (Normal operation); 1 = Enable. |
| 14 | CFRW | **Compressed Line Buffer Read/Write Select:** Enables the read/write address to the Compressed Line Buffer for use in diagnostic testing of the RAM.<br>0 = Write address enabled<br>1 = Read address enabled |
| 13 | PDEH | **Pixel Data Enable High:**<br>0 = The PIXEL [17:9] data bus to be driven to a logic low level. |
| 12 | PDEL | **Panel Data Enable Low:**<br>0 = This bit will cause the PIXEL[8:0] data bus to be driven to a logic low level. |
| 11:8 | RSVD | **Reserved:** Set to 0. |
| 7:5 | RSVD | **Reserved:** Set to 0. |
| 4:3 | RSVD | **Reserved:** Set to 0. |
| 2 | PCKE | **PCLK Enable:**<br>0 = PCLK is disabled and a low logic level is driven off-chip.<br>1 = Enable PCLK to be driven off-chip. |
| 1 | 16FMT | **16-bpp Format:** Selects RGB display mode:<br>0 = RGB 5-6-5 mode<br>1 = RGB 5-5-5 display mode<br>This bit is only significant if 8-bpp (OUTPUT_CONFIG, bit 0) is low, indicating 16-bpp mode. |
| 0 | 8-bpp | **8-bpp / 16-bpp Select:**<br>0 = 16-bpp display mode is selected. 16FMT (OUTPUT_CONFIG, bit 1) will indicate the format of the 16-bit data.)<br>1 = 8-bpp display mode is selected. Used in VGA emulation. |

## Integrated Functions (Continued)

### 4.5.9 Memory Organization Registers

The GXLV processor utilizes a graphics memory aperture that is up to 4 MB in size. The base address of the graphics memory aperture is stored in the DRAM controller Graphics Base Address register (see GBADD of MC_GBASE_ADD register, Table 4-15 on page 116 ). The graphics memory is made up of the normal uncompressed frame buffer, compressed display buffer, and cursor buffer. Each buffer begins at a programmable offset within the graphics memory aperture.

The various memory buffers are arranged so as to efficiently pack the data within the graphics memory aperture. The arrangement is programmable to efficiently accommodate different display modes. The cursor buffer is a linear block so addressing is straightforward. The frame buffer and compressed display buffer are arranged based upon scan lines. Each scan line has a maximum number of valid or active DWORDs, and a delta, which when added to the previous line offset, points to the next line. In this way, the buffers may either be stored as linear blocks, or as logical blocks as desired.

The Memory Organization Registers group consists of six 32-bit registers located at GX_BASE+8310h-8328h. These registers are summarized in Table 4-28 on page 141, and Table 4-30 gives their bit formats.

**Table 4-30. Display Controller Memory Organization Registers**

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+8310h-8313h** | | **DC_FB_ST_OFFSET Register (R/W) (Locked)** — **Default Value = xxxxxxxxh** |
| 31:22 | RSVD | **Reserved:** Set to 0. |
| 21:0 | FB_START _OFFSET | **Frame Buffer Start Offset:** This value represents the byte offset from the Graphics Base Address register (see GBADD of MC_GBASE_ADD register in Table 4-15 on page 116) of the starting location of the displayed frame buffer. This value may be changed to achieve panning across a virtual desktop or to allow multiple buffering.<br>When this register is programmed to a nonzero value, the compression logic should be disabled. The memory address defined by bits [21:4] will take effect at the start of the next frame scan. The pixel offset defined by bits [3:0] will take effect immediately (in general, it should only change during vertical blanking). |
| **GX_BASE+8314h-8317h** | | **DC_CB_ST_OFFSET Register (R/W) (Locked)** — **Default Value = xxxxxxxxh** |
| 31:22 | RSVD | **Reserved:** Set to 0. |
| 21:0 | CB_START _OFFSET | **Compressed Display Buffer Start Offset:** This value represents the byte offset from the Graphics Base Address register (see GBADD of MC_GBASE_ADD register in Table 4-15 on page 116) of the starting location of the compressed display buffer. Bits [3:0] must be programmed to zero so that the start offset is aligned to a 16-byte boundary. This value should change only when a new display mode is set due to a change in size of the frame buffer. |
| **GX_BASE+8318h-831Bh** | | **DC_CUR_ST_OFFSET Register (R/W) (Locked)** — **Default Value = xxxxxxxxh** |
| 31:22 | RSVD | **Reserved:** Set to 0. |
| 21:0 | CUR_START _OFFSET | **Cursor Start Offset:** This register contains the byte offset from the Graphics Base Address register (see GBADD of MC_GBASE_ADD register in Table 4-15 on page 116) of the starting location of the cursor display pattern. Bits [1:0] should always be programmed to zero so that the start offset is DWORD aligned. The cursor data will be stored as a linear block of data. |
| **GX_BASE+831Ch-831Fh** | | **Reserved** — **Default Value = 00000000h** |
| **GX_BASE+8320h-8323h** | | **DC_VID_ST_OFFSET Register (R/W) (Locked)** — **Default Value = xxxxxxxxh** |
| 31:22 | RSVD | **Reserved:** Set to 0. |
| 21:0 | VID_START _OFFSET | **Video Buffer Start Offset Value:** This register contains the byte offset from the Graphics Base Address register (see GBADD of MC_GBASE_ADD register in Table 4-15 on page 116) of the starting location of the Video Buffer Start. Bits [3:0] must be programmed as zero so that the start offset is aligned to a 16 byte boundary. |
| **GX_BASE+8324h-8327h** | | **DC_LINE_DELTA Register (R/W) (Locked)** — **Default Value = xxxxxxxxh** |
| 31:22 | RSVD | **Reserved:** Set to 0. |
| 21:12 | CB_LINE_ DELTA | **Compressed Display Buffer Line Delta:** This value represents number of DWORDs that, when added to the starting offset of the previous line, will point to the start of the next compressed line in memory. It is used to always maintain a pointer to the starting offset for the compressed display buffer line being loaded into the display FIFO. |
| 11:10 | RSVD | **Reserved:** Set to 0. |

**Table 4-30. Display Controller Memory Organization Registers (Continued)**

| Bit | Name | Description |
|---|---|---|
| 9:0 | FB_LINE_ DELTA | **Frame Buffer Line Delta:** This value represents number of DWORDs that, when added to the starting offset of the previous line, will point to the start of the next frame buffer line in memory. It is used to always maintain a pointer to the starting offset for the frame buffer line being loaded into the display FIFO. |
| **GX_BASE+8328h-832Bh** | **DC_BUF_SIZE Register (R/W) (Locked)** | **Default Value = xxxxxxxxh** |
| 31:30 | RSVD | **Reserved:** Set to 0. |
| 29:16 | VID_BUF_ SIZE | **Video Buffer Size:** These bits set the video buffer size, in 64-byte segments. The maximum size is 1 MB. |
| 15:9 | CB_LINE_ SIZE | **Compressed Display Buffer Line Size:** This value represents the number of DWORDs for a valid compressed line plus 1. It is used to detect an overflow of the compressed data FIFO. It should never be larger than 41h since the maximum size of the compressed data FIFO is 64 DWORDs. |
| 8:0 | FB_LINE_ SIZE | **Frame Buffer Line Size:** This value specifies the number of QWORDS (8-byte segments) to transfer for each display line from the frame buffer.<br>If panning is enabled, this value can generally be programmed to the displayed number of QWORDS + 2 so that enough data is transferred to handle any possible alignment. Extra pixel data in the FIFO at the end of a line will automatically be discarded. |
| **GX_BASE+832Ch-832Fh** | **Reserved** | **Default Value = 00000000h** |

## Integrated Functions (Continued)

### 4.5.10 Timing Registers

The Display Controller's timing registers control the generation of sync, blanking, and active display regions. They provide complete flexibility in interfacing to both CRT and flat panel displays. These registers will generally be programmed by the BIOS from an INT 10h call or by the extended mode driver from a display timing file. Note that the horizontal timing parameters are specified in character clocks, which actually means pixels divided by 8, since all characters are bit mapped. For interlaced display the vertical counter will be incremented twice during each display line, so vertical timing parameters should be programmed with reference to the total frame rather than a single field.

The Timing Registers group consists of six 32-bit registers located at GX_BASE+8330h-834Ch. These registers are summarized in Table 4-28 on page 141, and Table 4-31 gives their bit formats.

**Table 4-31. Display Controller Timing Registers**

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+8330h-8333h** | | **DC_H_TIMING_1 Register (R/W) (Locked)** **Default Value = xxxxxxxxh** |
| 31:27 | RSVD | **Reserved:** Set to 0. |
| 26:19 | H_TOTAL | **Horizontal Total:** The total number of character clocks for a given scan line minus 1. Note that the value is necessarily greater than the H_ACTIVE field because it includes border pixels and blanked pixels. For flat panels, this value will never change. The field [26:16] may be programmed with the pixel count minus 1, although bits [18:16] are ignored. The horizontal total is programmable on 8-pixel boundaries only. |
| 18:16 | IGRD | **Ignored** |
| 15:11 | RSVD | **Reserved:** Set to 0. |
| 10:3 | H_ACTIVE | **Horizontal Active:** The total number of character clocks for the displayed portion of a scan line minus 1. The field [10:0] may be programmed with the pixel count minus 1, although bits [2:0] are ignored. The active count is programmable on 8-pixel boundaries only. Note that for flat panels, if this value is less than the panel active horizontal resolution (H_PANEL), the parameters H_BLANK_START, H_BLANK_END, H_SYNC_START, and H_SYNC_END should be reduced by the value of H_ADJUST (or the value of H_PANEL - H_ACTIVE / 2)to achieve horizontal centering. |
| 2:0 | IGRD | **Ignored** |
| **Note:** For simultaneous CRT and flat panel display the H_ACTIVE and H_TOTAL parameters pertain to both. | | |
| **GX_BASE+8334h-8337h** | | **DC_H_TIMING_2 Register (R/W) (Locked)** **Default Value = xxxxxxxxh** |
| 31:27 | RSVD | **Reserved:** Set to 0. |
| 26:19 | H_BLK_END | **Horizontal Blank End:** The character clock count at which the horizontal blanking signal becomes inactive minus 1. The field [26:16] may be programmed with the pixel count minus 1, although bits [18:16] are ignored. The blank end position is programmable on 8-pixel boundaries only. |
| 18:16 | IGRD | **Ignored** |
| 15:11 | RSVD | **Reserved:** Set to 0. |
| 10:3 | H_BLK_START | **Horizontal Blank Start:** The character clock count at which the horizontal blanking signal becomes active minus 1. The field [10:0] may be programmed with the pixel count minus 1, although bits [2:0] are ignored. The blank start position is programmable on 8-pixel boundaries only. |
| 2:0 | IGRD | **Ignored** |
| **Note:** A minimum of four character clocks are required for the horizontal blanking portion of a line in order for the timing generator to function correctly. | | |
| **GX_BASE+8338h-833Bh** | | **DC_H_TIMING_3 Register (R/W) (Locked)** **Default Value = xxxxxxxxh** |
| 31:27 | RSVD | **Reserved:** Set to 0. |
| 26:19 | H_SYNC_END | **Horizontal Sync End:** The character clock count at which the CRT horizontal sync signal becomes inactive minus 1. The field [26:16] may be programmed with the pixel count minus 1, although bits [18:16] are ignored. The sync end position is programmable on 8-pixel boundaries only. |
| 18:16 | IGRD | **Ignored** |
| 15:11 | RSVD | **Reserved:** Set to 0. |
| 10:3 | H_SYNC_START | **Horizontal Sync Start:** The character clock count at which the CRT horizontal sync signal becomes active minus 1. The field [10:0] may be programmed with the pixel count minus 1, although bits [2:0] are ignored. The sync start position is programmable on 8-pixel boundaries only. |
| 2:0 | IGRD | **Ignored** |
| **Note:** This register should also be programmed appropriately for flat panel only display since the horizontal sync transition determines when to advance the vertical counter. | | |

## Integrated Functions (Continued)

### Table 4-31. Display Controller Timing Registers (Continued)

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+833Ch-833Fh** | **C_FP_H_TIMING Register (R/W) (Locked)** | **Default Value = xxxxxxxxh** |
| 31:27 | RSVD | **Reserved:** Set to 0. |
| 26:16 | FP_H_SYNC _END | **Flat Panel Horizontal Sync End:** The pixel count at which the flat panel horizontal sync signal becomes inactive minus 1. |
| 15:11 | RSVD | **Reserved:** Set to 0. |
| 10:0 | FP_H_SYNC _START | **Flat Panel Horizontal Sync Start:** The pixel count at which the flat panel horizontal sync signal becomes active minus 1. |
| **Note:** These values are specified in pixels rather than character clocks to allow precise control over sync position. For flat panels which combine two pixels per panel clock, these values should be odd numbers (even pixel boundary) to guarantee that the sync signal will meet proper setup and hold times. | | |
| **GX_BASE+8340h-8343h** | **DC_V_TIMING_1 Register (R/W) (Locked)** | **Default Value = xxxxxxxxh** |
| 31:27 | RSVD | **Reserved:** Set to 0. |
| 26:16 | V_TOTAL | **Vertical Total:** The total number of lines for a given frame scan minus 1. The value is necessarily greater than the V_ACTIVE field because it includes border lines and blanked lines. If the display is interlaced, the total number of lines must be odd, so this value should be an even number. |
| 15:11 | RSVD | **Reserved:** Set to 0. |
| 10:0 | V_ACTIVE | **Vertical Active:** The total number of lines for the displayed portion of a frame scan minus 1. For flat panels, if this value is less than the panel active vertical resolution (V_PANEL), the parameters V_BLANK_START, V_BLANK_END, V_SYNC_START, and V_SYNC_END should be reduced by the following value (V_ADJUST) to achieve vertical centering: V_ADJUST = (V_PANEL – V_ACTIVE) / 2<br>If the display is interlaced, the number of active lines should be even, so this value should be an odd number. |
| **Note:** These values are specified in lines. | | |
| **GX_BASE+8344h-8347h** | **DC_V_TIMING_2 Register (R/W) (Locked)** | **Default Value = xxxxxxxxh** |
| 31:27 | RSVD | **Reserved:** Set to 0. |
| 26:16 | V_BLANK_END | **Vertical Blank End:** The line at which the vertical blanking signal becomes inactive minus 1. If the display is interlaced, no border is supported, so this value should be identical to V_TOTAL. |
| 15:11 | RSVD | **Reserved:** Set to 0. |
| 10:0 | V_BLANK_ START | **Vertical Blank Start:** The line at which the vertical blanking signal becomes active minus 1. If the display is interlaced, this value should be programmed to V_ACTIVE plus 1. |
| **Note:** These values are specified in lines. For interlaced display, no border is supported, so blank timing is implied by the total/active timing. | | |
| **GX_BASE+8348h-834Bh** | **DC_V_TIMING_3 Register (R/W) (Locked)** | **Default Value = xxxxxxxxh** |
| 31:27 | RSVD | **Reserved:** Set to 0. |
| 26:16 | V_SYNC_END | **Vertical Sync End:** The line at which the CRT vertical sync signal becomes inactive minus 1. |
| 15:11 | RSVD | **Reserved:** Set to 0. |
| 10:0 | V_SYNC_START | **Vertical Sync Start:** The line at which the CRT vertical sync signal becomes active minus 1. For interlaced display, note that the vertical counter is incremented twice during each line and since there are an odd number of lines, the vertical sync pulse will trigger in the middle of a line for one field and at the end of a line for the subsequent field. |
| **Note:** These values are specified in lines. | | |
| **GX_BASE+834Ch-834Fh** | **DC_FP_V_TIMING Register (R/W) (Locked)** | **Default Value = xxxxxxxxh** |
| 31:27 | RSVD | **Reserved:** Set to 0. |
| 26:16 | FP_V_SYNC _END | **Flat Panel Vertical Sync End:** The line at which the flat panel vertical sync signal becomes inactive minus 2. Note that the internal flat panel vertical sync is latched by the flat panel horizontal sync prior to being output to the panel. |
| 15:11 | RSVD | **Reserved:** Set to 0. |
| 10:0 | FP_VSYNC _START | **Flat Panel Vertical Sync Start:** The line at which the internal flat panel vertical sync signal becomes active minus 2. Note that the internal flat panel vertical sync is latched by the flat panel horizontal sync prior to being output to the panel. |
| **Note:** These values are specified in lines. | | |

## Integrated Functions (Continued)

### 4.5.11 Cursor Position and Miscellaneous Registers

The Cursor Position Registers contain pixel coordinate information for the cursor. These values are not latched by the timing generator until the start of the frame to avoid tearing artifacts when moving the cursor.

The Cursor Position group consists of two 32-bit registers located at GX_BASE+8350h and GX_BASE+8358h. These registers are summarized in Table 4-28 on page 141, and Table 4-32 gives their bit formats.

**Table 4-32. Display Controller Cursor Position Registers**

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+8350h-8353h** | | **DC_CURSOR_X Register (R/W)** **Default Value = xxxxxxxxh** |
| 31:16 | RSVD | **Reserved:** Set to 0. |
| 15:11 | X_OFFSET | **X Offset:** The X pixel offset within the 32x32 cursor pattern at which the displayed portion of the cursor is to begin. Normally, this value is set to zero to display the entire cursor pattern, but for cursors for which the "hot spot" is not at the left edge of the pattern, it may be necessary to display the rightmost pixels of the cursor only as the cursor moves close to the left edge of the display. |
| 10:0 | CURSOR_X | **Cursor X:** The X coordinate of the pixel at which the upper left corner of the cursor is to be displayed. This value is referenced to the screen origin (0,0) which is the pixel in the upper left corner of the screen. |
| **GX_BASE+8354h-8357h** | | **DC_V_LINE_CNT Register (RO)** **Default Value = xxxxxxxxh** |
| 31:11 | RSVD | **Reserved (Read Only)** |
| 10:0 | V_LINE_CNT (RO) | **Vertical Line Count (Read Only):** This value is the current scanline of the display. |
| **Note:** The value in this register is driven directly off of the DCLK, and is not synchronized with the CPU clock. Software should read this register twice and compare the two results to ensure that the value is not in transition. | | |
| **GX_BASE+8358h-835Bh** | | **DC_CURSOR_Y Register (R/W)** **Default Value = xxxxxxxxh** |
| 31:16 | RSVD | **Reserved:** Set to 0. |
| 15:11 | Y_OFFSET | **Y Offset:** The Y line offset within the 32x32 cursor pattern at which the displayed portion of the cursor is to begin. Normally, this value is set to zero to display the entire cursor pattern, but for cursors for which the "hot spot" is not at the top edge of the pattern, it may be necessary to display the bottommost lines of the cursor only as the cursor moves close to the top edge of the display. If this value is nonzero, the CUR_START_OFFSET must be set to point to the first cursor line to be displayed. |
| 10 | RSVD | **Reserved:** Set to 0. |
| 9:0 | CURSOR_Y | **Cursor Y:** The Y coordinate of the line at which the upper left corner of the cursor is to be displayed. This value is referenced to the screen origin (0,0) which is the pixel in the upper left corner of the screen.<br>This field is alternately used as the line-compare value for a newly-programmed frame buffer start offset. This is necessary for VGA programs that change the start offset in the middle of a frame. In order to use this function, the hardware cursor function should be disabled. |
| **GX_BASE+835Ch-835Fh** | | **DC_SS_LINE_CMP Register (R/W)** **Default Value = xxxxxxxxh** |
| 31:11 | RSVD | **Reserved:** Set to 0. |
| 10:0 | SS_LINE_CMP | **Split-Screen Line Compare:** This is the line count at which the lower screen begins in a VGA split-screen mode. |
| **Note:** When the internal line counter hits this value, the frame buffer address is reset to 0. This function is enabled with the SSLC bit in the DC_GENERAL_CFG register (see Table 4-29). | | |

## Integrated Functions (Continued)

### 4.5.12 Palette Access Registers

These registers are used for accessing the internal palette RAM and extensions. In addition to the standard 256 entries for 8-bpp color translation, the GXLV processor palette has extensions for cursor colors and overscan (border) color.

The Palette Access Register group consists of two 32-bit registers located at GX_BASE+8370h and GX_BASE+8374h. These registers are summarized in Table 4-28 on page 141, and Table 4-33 gives their bit formats.

**Table 4-33. Display Controller Palette**

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+8370h-8373h** | | **DC_PAL_ADDRESS Register (R/W)** — **Default Value = xxxxxxxxh** |
| 31:9 | RSVD | **Reserved:** Set to 0. |
| 8:0 | PALETTE_ADDR | **Palette Address:** The address to be used for the next access to the DC_PAL_DATA register. Each access to the data register will automatically increment the palette address register. If non-sequential access is made to the palette, the address register must be loaded between each non-sequential data block. The address ranges are as follows.<br>Address — Color<br>0h - FFh — Standard Palette Colors<br>100h — Cursor Color 0<br>101h — Cursor Color 1<br>102h — Reserved<br>103h — Reserved<br>104h — Overscan (Color Border)<br>105h - 1FFh — Not Valid |
| **GX_BASE+8374h-8377h** | | **DC_PAL_DATA Register (R/W)** — **Default Value = xxxxxxxxh** |
| 31:18 | RSVD | **Reserved:** Set to 0. |
| 17:0 | PALETTE_DATA | **Palette Data:** The read or write data for a palette access. |

**Note:** When a read or write to the palette RAM occurs, the previous output value will be held for one additional DCLK period. This effect should go unnoticed and will provide for sparkle-free update. Prior to a read or write to this register, the DC_PAL_ADDRESS register should be loaded with the appropriate address. The address automatically increments after each access to this register, so for sequential access, the address register need only be loaded once

## Integrated Functions (Continued)

### 4.5.13 FIFO Diagnostic Registers

The FIFO Diagnostic Register group consists of two 32-bit registers located at GX_BASE+8378h and GX_BASE+837Ch. These registers are summarized in Table 4-28 on page 141, and Table 4-33 gives their bit formats

**Table 4-34. FIFO Diagnostic Registers**

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+8378h-837Bh** | | **DC_DFIFO_DIAG Register (R/W)** **Default Value = xxxxxxxxh** |
| 31:0 | DISPLAY FIFO DIAGNOSTIC DATA | **Display FIFO Diagnostic Read or Write Data:** Before this register is accessed, the DIAG bit in DC_GENERAL_CFG register (see Table 4-29 on page 144) should be set high and the DFLE bit should be set low. Since, each FIFO entry is 64 bits, an even number of write operations should be performed. Each pair of write operations will cause the FIFO write pointer to increment automatically. After all write operations have been performed, a single read of don't care data should be performed to load data into the output latch. Each subsequent read will contain the appropriate data which was previously written. Each pair of read operations will cause the FIFO read pointer to increment automatically. A pause of at least four core clocks should be allowed between subsequent read operations to allow adequate time for the shift to take place. |
| **GX_BASE+837Ch-837Fh** | | **DC_CFIFO_DIAG Register (R/W)** **Default Value = xxxxxxxxh** |
| 31:0 | COMPRESSED FIFO DIAGNOSTIC DATA | **Compressed Data FIFO Diagnostic Read or Write Data:** Before this register is accessed, the DIAG bit in DC_GENERAL_CFG (see Table 4-29 on page 144) register should be set high and the DFLE bit should be set low. Also, the DIAG bit in DC_OUTPUT_CFG (see Table 4-29) should be set high and the CFRW bit in DC_OUTPUT_CFG should be set low. After each write, the FIFO write pointer will automatically increment. After all write operations have been performed, the CFRW bit of DC_OUTPUT_CFG should be set high to enable read addresses to the FIFO and a single read of don't care data should be performed to load data into the output latch. Each subsequent read will contain the appropriate data which was previously written. After each read, the FIFO read pointer will automatically increment. |

## Integrated Functions (Continued)

### 4.5.14 CS5530 Display Controller Interface

As previously stated in Section 1.7 “Geode GXLV/CS5530 System Designs” on page 13, the GXLV processor interfaces with the Geode CS5530 I/O companion chip. This section will discuss the specifics on signal connections between the two devices with regards to the display controller.

Because the GXLV processor is used in a system with the CS5530 I/O companion chip, the need for an external RAMDAC is eliminated. The CS5530 contains the DACs, a video accelerator engine, and a TFT interface.

A GXLV processor and CS5530-based system supports both flat panel and CRT configurations. Figure 4-16 shows the signal connections for both types of systems.

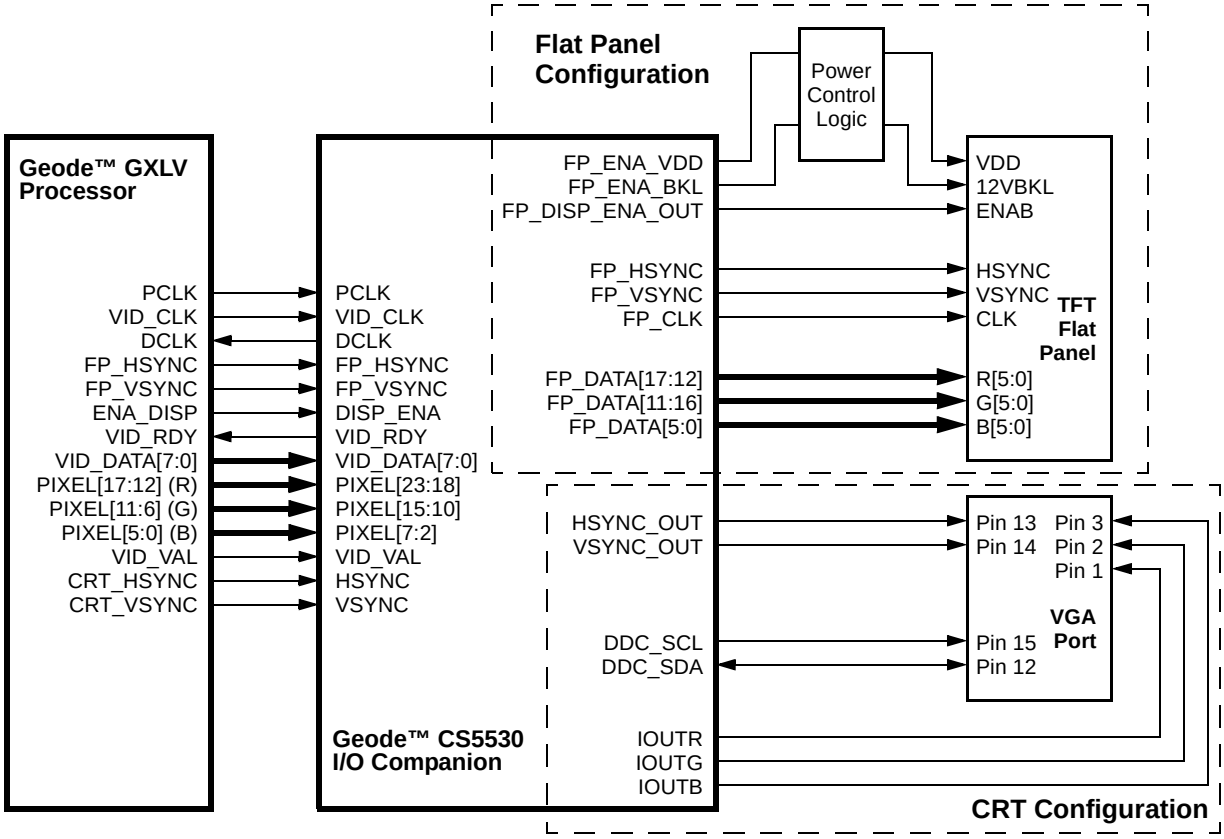


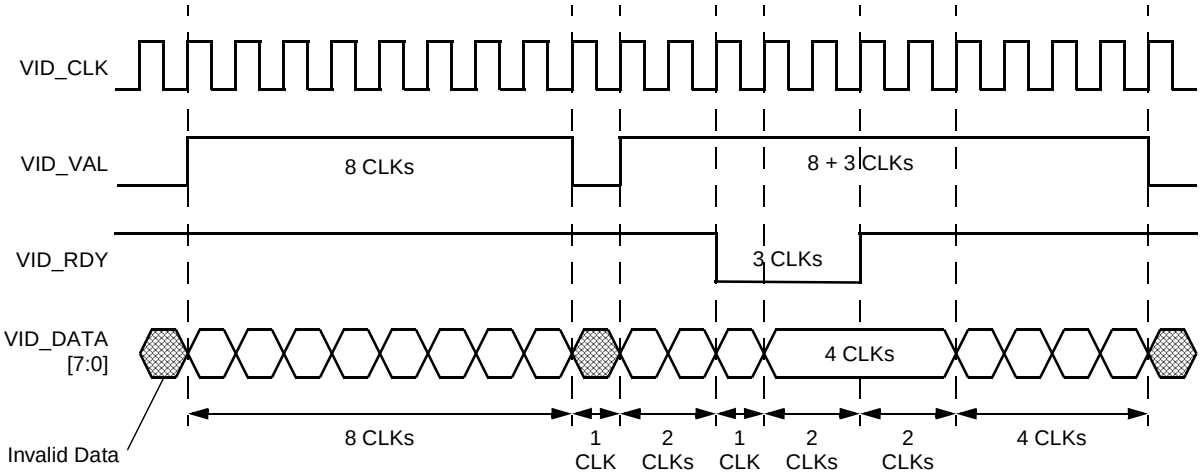

**Figure 4-16. Display Controller Signal Connections**

## Integrated Functions (Continued)

#### 4.5.14.1 CS5530 Video Port Data Transfer

VID_VAL indicates that the GXLV processor has placed valid data on VID_DATA[7:0]. VID_RDY indicates that the CS5530 is ready to accept the next byte of video data.

VID_DATA[7:0] is advanced when both VID_VAL and VID_RDY are asserted. VID_RDY is driven one clock early to the GXLV processor while VID_VAL is driven coincident with VID_DATA[7:0]. A sample interface functional timing diagram is shown in Figure 4-17.




**Note:** VID_CLK = CORE_CLK/2

**Figure 4-17. Video Port Data Transfer (CS5530)**

## 4.6 VIRTUAL VGA SUBSYSTEM

This section describes the Virtual System Architecture as implemented with the Geode GXLV processor(s) and VSA enhanced Geode I/O companion device(s). VSA provides a framework to enable software implementation of traditionally hardware-only components. VSA software executes in System Management Mode (SMM), enabling it to execute transparently to the operating system, drivers and applications.

The VSA design is based on a simple model for replacing hardware components with software. Hardware to be virtualized is merely replaced with simple access detection circuitry which asserts the processor's SMI# pin when hardware accesses are detected. The current execution stream is immediately preempted, and the processor enters SMM. The SMM system software then saves the processor state, initializes the VSA execution environment, decodes the SMI source and dispatches handler routines which have registered requests to service the decoded SMI source. Once all handler routines have completed, the processor state is restored and normal execution resumes. In this manner, hardware accesses are transparently replaced with the execution of SMM handler software.

Historically, SMM software was used primarily for the single purpose of facilitating active power management for notebook designs. That software's only function was to manage the power up and down of devices to save power. With high performance processors now available, it is feasible to implement, primarily in SMM software, PC capabilities traditionally provided by hardware. In contrast to power management code, this virtualization software generally has strict performance requirements to prevent application performance from being significantly impacted.

Several functions can be virtualized in a GXLV processor based design using the VSA environment. The VSA enhanced Geode I/O companions provide programmable resources to trap both memory and I/O accesses. However, specific hardware is included to support the virtualization of VGA core compatibility and audio functionality in the system.

The hardware support for VGA emulation resides completely inside the GXLV processor. Legacy VGA accesses do not generate off-chip bus cycles. However, the VSA support hardware for XpressAUDIO resides in an I/O Companion device such as the Geode CS5530.

### 4.6.1 Traditional VGA Hardware

A VGA card consists of display memory and control registers. The VGA display memory shows up in system memory between addresses A0000h and BFFFFh. It is possible to map this memory to three different ranges within this 128 KB block.

The first range is
- A0000h to AFFFFh for EGA and VGA modes,

the second range is
- B0000h to B7FFFh for MDA modes,

and the third range is
- B8000h to BFFFFh for CGA modes.

The VGA control registers are mapped to the I/O address range from 3B0h to 3DFh. The VGA registers are accessed with an indexing scheme that provides more registers than would normally fit into this range. Some registers are mapped at two locations, one for monochrome, and another for color.

The VGA hardware can be accessed by calling BIOS routines or by directly writing to VGA memory and control registers. DOS always calls BIOS to set up the display mode and render characters. Many other applications access the VGA memory and control registers directly. The VGA card can be set up to a virtually unlimited number of modes. However, many applications use one of the predefined modes specified by the BIOS routine which sets up the display mode. The predefined modes are translated into specific VGA control register setups by the BIOS. The standard modes supported by VGA cards are shown in Table 4-35.

**Table 4-35. Standard VGA Modes**

| Category | Mode | Text or Graphics | Resolution | Format | Type |
|---|---|---|---|---|---|
| Software | 0,1 | Text | 40x25 | Characters | CGA |
| | 2,3 | Text | 80x25 | Characters | CGA |
| | 4,5 | Graphics | 320x200 | 2 bpp | CGA |
| | 6 | Graphics | 640x200 | 1 bpp | CGA |
| | 7 | Text | 80x25 | Characters | MDA |
| Hardware | 0Dh | Graphics | 320x200 | 4 bpp | EGA |
| | 0Eh | Graphics | 640x200 | 4 bpp | EGA |
| | 0Fh | Graphics | 640x350 | 1 bpp | EGA |
| | 10h | Graphics | 640x350 | 4 bpp | EGA |
| | 11h | Graphics | 640x480 | 1 bpp | VGA |
| | 12h | Graphics | 640x480 | 4 bpp | VGA |
| | 13h | Graphics | 320x200 | 8 bpp | VGA |

## Integrated Functions (Continued)

A VGA is made up of several functional units.

- The **frame buffer** is 256 KB of memory that provides data for the video display. It is organized as 64 K 32-bit DWORDs.
- The **sequencer** decomposes word and DWORD CPU accesses into byte operations for the graphics controller. It also controls a number of miscellaneous functions, including reset and some clocking controls.
- The **graphics controller** provides most of the interface between CPU data and the frame buffer. It allows the programmer to read and write frame buffer data in different formats. Plus provides ROP (raster operation) and masking functions.
- The **CRT controller** provides video timing signals and address generation for video refresh. It also provides a text cursor.
- The **attribute controller** contains the video refresh datapath, including text rasterization and palette lookup.
- The **general registers** provide status information for the programmer as well as control over VGA-host address mapping and clock selection. This is all handled in hardware by the graphics pipeline.

It is important to understand that a VGA is constructed of numerous independent functions. Most of the register fields correspond to controls that were originally built out of discrete logic or were part of a dedicated controller such as the 6845. The notion of a VGA “mode” is a higher-level convention to denote a particular set of values for the registers. Many popular programs do not use standard modes, preferring instead to produce their own VGA set-ups that are optimal for their purposes.

#### 4.6.1.1 VGA Memory Organization

The VGA memory is organized as 64K 32-bit DWORDs. This organization is usually presented as four 64 KB “planes”. A plane consists of one byte out of every DWORD. Thus, plane 0 refers to the least significant byte from every one of the 64K DWORDs. The addressing granularity of this memory is a DWORD, not a byte; that is, consecutive addresses refer to consecutive DWORDs. The only provision for byte-granularity addressing is the four-byte enable signals used for writes. In C parlance,

```
single_plane_byte = (dword_fb[address] >>
(plane * 8)) & 0xFF;
```

When dealing with VGA, it is important to recognize the distinction between host addresses, frame buffer addresses, and the refresh address pipe. A VGA controller contains a lot of hardware to translate between these address spaces in different ways, and understanding these translations is critical to understanding the entire device. In standard four-plane graphics modes, a frame-buffer DWORD provides eight 4-bit pixels. The left-most pixel comes from bit 7 of each plane, with plane 3 providing the most significant bit.

```
pixel[i].bit[j] = dword_fb[address].bit[i*8 + (7-j)]
```

#### 4.6.1.2 VGA Front End

The VGA front end consists of address and data translations between the CPU and the frame buffer. This functionality is contained within the graphics controller and sequencer components. Most of the front end functionality is implemented in the VGA read and write hardware of the GXLV processor. An important axiom of the VGA is that the front end and back end are controlled independently. There are no register fields that control the behavior of both pieces. Terms like “VGA odd/even mode” are therefore somewhat misleading; there are two different controls for odd/even functionality in the front end, and two separate controls in the refresh path to cause “sensible” refresh behavior for frame buffer contents written in odd/even mode. Normally, all these fields would be set up together, but they don’t have to be. This sort of orthogonal behavior gives rise to the enormous number of possible VGA “modes”. The CPU end of the read and write pipelines is one byte wide. Word and DWORD accesses from the CPU to VGA memory are broken down into multiple byte accesses by the sequencer. For example, a word write to A0000h (in a VGA graphics mode) is processed as if it were two-byte write operations to A0000h and A0001h.

#### 4.6.1.3 Address Mapping

When a VGA card sees an address on the host bus, bits [31:15] determine whether the transaction is for the VGA. Depending on the mode, addresses 000AXXXX, 000B{0xxx}XXX, or 000B{1xxx}XXX can decode into VGA space. If the access is for the VGA, bits [15:0] provide the DWORD address into the frame buffer (see odd/even and Chain 4 modes, next paragraph). Thus, each byte address on the host bus addresses a DWORD in VGA memory.

On a write transaction, the byte enables are normally driven from the sequencer’s MapMask register. The VGA has two other write address mappings that modify this behavior. In odd/even (Chain 2) write mode, bit 0 of the address is used to enable bytes 0 and 2 (if zero) or bytes 1 and 3 (if one). In addition, the address presented to the frame buffer has bit 0 replaced with the PageBit field of the Miscellaneous Output register. Chain 4 write mode is similar; only one of the four byte enables is asserted, based on bits [1:0] of the address, and bits [1:0] of the frame buffer address are set to zero. In each of these modes, the MapMask enables are logically ANDed into the enables that result from the address.

#### 4.6.1.4 Video Refresh

VGA refresh is controlled by two units: the CRT controller (CRTC) and the attribute controller (ATTR). The CRTC provides refresh addresses and video control; the ATTR provides the refresh datapath, including pixel formatting and internal palette lookup.

The VGA back end contains two basic clocks: the dot clock (or pixel clock) and the character clock. The Clock-Select field of the Miscellaneous Output register selects a "master clock" of either 25 MHz or 28 MHz. This master clock, optionally divided by two, drives the dot clock. The character clock is simply the dot clock divided by eight or nine.

The VGA supports four basic pixel formats. Using text format, the VGA interprets frame buffer values as ASCII characters, foreground/background attributes, and font data. The other three formats are all "graphics modes", known as APA (All Points Addressable) modes. These formats could be called CGA-compatible (odd/even 4-bpp), EGA-compatible (4-plane 4-bpp), and VGA-compatible (pixel-per-byte 8-bpp). The format is chosen by the ShiftRegister field of the Graphics Controller Mode register.

The refresh address pipe is an integral part of the CRTC, and has many configuration options. Refresh can begin at any frame buffer address. The display width and the frame buffer pitch (scan-line delta) are set separately. Multiple scan lines can be refreshed from the same frame buffer addresses. The LineCompare register causes the refresh address to be reset to zero at a particular scan line, providing support for vertical split-screen.

Within the context of a single scan line, the refresh address increments by one on every character clock. Before being presented to the frame buffer, refresh addresses can be shifted by 0, 1, or 2 bits to the left. These options are often mis-named BYTE, WORD, and DWORD modes. Using this shifter, the refresh unit can be programmed to skip one out of two or three out of four DWORDs of refresh data. As an example of the utility of this function, consider Chain 4 mode, described in Section 4.6.1.3 "Address Mapping" on page 158. Pixels written in Chain 4 mode occupy one out of every four DWORDs in the frame buffer. If the refresh path is put into "Double-word" mode, the refresh will come only from those DWORDs writable in Chain 4. This is how VGA mode 13h works.

In text mode, the ATTR has a lot of work to do. At each character clock, it pulls a DWORD of data out of the frame buffer. In that DWORD, plane 0 contains the ASCII character code, and plane 1 contains an attribute byte. The ATTR uses plane 0 to generate a font lookup address and read another DWORD. In plane 2, this DWORD contains a bit-per-pixel representation of one scan line in the appropriate character glyph. The ATTR transforms these bits into eight pixels, obtaining foreground and background colors from the attribute byte. The CRTC must refresh from the same memory addresses for all scan lines that make up a character row; within that row, the ATTR must fetch successive scan lines from the glyph table so as to draw proper characters. Graphics modes are somewhat simpler. In CGA-compatible mode, a DWORD provides eight pixels. The first four pixels come from planes 0 and 2; each 4-bit pixel gets bits [3:2] from plane 2, and bits [1:0] from plane 0. The remaining four pixels come from planes 1 and 3. The EGA-compatible mode also gets eight pixels from a DWORD, but each pixel gets one bit from each plane, with plane 3 providing bit 3. Finally, VGA-compatible mode gets four pixels from each DWORD; plane 0 provides the first pixel, plane 1 the next, and so on. The 8 bpp mode uses an option to provide every pixel for two dot clocks, thus allowing the refresh pipe to keep up (it only increments on character clocks) and meaning that the 320-pixel-wide mode 13h really has 640 visible pixels per line. The VGA color model is unusual. The ATTR contains a 16-entry color palette with 6 bits per entry. Except for 8 bpp modes, all VGA configurations drive four bits of pixel data into the palette, which produces a 6-bit result. Based on various control registers, this value is then combined with other register contents to produce an 8-bit index into the DAC. There is a ColorPlaneEnable register to mask bits out of the pixel data before it goes to the palette; this is used to emulate four-color CGA modes by ignoring the top two bits of each pixel. In 8 bpp modes, the palette is bypassed and the pixel data goes directly to the DAC.

#### 4.6.1.5 VGA Video BIOS

The video BIOS supports the VESA BIOS Extensions (VBE) Version 1.2 and 2.0, as well as all standard VGA BIOS calls. It interacts with Virtual VGA through the use of several extended VGA registers. These are virtual registers contained in the VSA code for Virtual VGA. (These registers are defined in a separate document.)

### 4.6.2 Virtual VGA

The GXLV processor reduces the burden of legacy hardware by using a balanced mix of hardware and software to provide the same functionality. The graphics pipeline contains full hardware support for the VGA "front-end", the logic that controls read and write operations to the VGA frame buffer (located in graphics memory). For some modes, the hardware can also provide direct display of the data in the VGA buffer. Virtual VGA traps frame buffer accesses only when necessary, but it must trap all VGA I/O accesses to maintain the VGA state and properly program the graphics pipeline and display controller.

The processor core contains SMI generation hardware for VGA memory write operations. The bus controller contains SMI generation hardware for VGA I/O read and write operations. The graphics pipeline contains hardware to detect and process reads and writes to VGA memory. VGA memory is partitioned from system memory.

VGA functionality with the GXLV processor includes the standard VGA modes (VGA, EGA, CGA, and MDA) as well as the higher-resolution VESA modes. The CGA and MDA modes (modes 0 through 7) require that Virtual VGA

## Integrated Functions (Continued)

convert the data in the VGA buffer to a separate 8-bpp frame buffer that the hardware can use for display refresh.

The remaining modes, VGA, EGA, and VESA, can be displayed directly by the hardware, with no data conversion required. For these modes, Virtual VGA often outperforms typical VGA cards because the frame buffer data does not travel across an external bus.

Display drivers for popular GUI (graphical user interface) based operating systems are provided by National Semiconductor which enable a full featured 2D hardware accelerator to be used instead of the emulated VGA core.

#### 4.6.2.1 Datapath Elements

The graphics controller contains several elements that convert between host data and frame buffer data.

The rotator simply rotates the byte written from the host by 0 to 7 bits to the right, based on the RotateCount field of the DataRotate register. It has no effect in the read path.

The display latch is a 32-bit register that is loaded on every read access to the frame buffer. All 32 bits of the frame buffer DWORDs are loaded into the latch.

The **write-mode unit** converts a byte from the host into a 32-bit value. A VGA has four write modes:

- Write Mode 0:
  - Bit n of byte b comes from one of two places, depending on bit b of the EnableSetReset register. If that bit is zero, it comes from bit n of the host data. If that bit is one, it comes from bit b of the SetReset register. This mode allows the programmer to set some planes from the host data and the others from SetReset.
- Write Mode 1:
  - All 32 bits come directly out of the display latch; the host data is ignored. This mode is used for screen-to-screen copies.
- Write Mode 2:
  - Bit n of byte b comes from bit b of the host data; that is, the four LSBs of the host data are each replicated through a byte of the result. In conjunction with the BitMask register, this mode allows the programmer to directly write a 4-bit color to one or more pixels.
- Write Mode 3:
  - Bit n of byte b comes from bit b of the SetReset register. The host data is ANDed with the BitMask register to provide the bit mask for the write (see below).

The **read mode unit** converts a 32-bit value from the frame buffer into a byte. A VGA has two read modes:

- Read Mode 0:
  - One of the four bytes from the frame buffer is returned, based on the value of the ReadMapSelect register. In Chain 4 mode, bits [1:0] of the read address select a plane. In odd/even read mode, bit 0 of the read address replaces bit 0 of ReadMapSelect.
- Read Mode 1:
  - Bit n of the result is set to 1 if bit n in every byte b matches bit b of the ColorCompare register; otherwise it is set to 0. There is a ColorDon'tCare register that can exclude planes from this comparison. In four-plane graphics modes, this provides a conversion from 4 bpp to 1 bpp.

The ALU is a simple two-operand ROP unit that operates on writes. Its operating modes are COPY, AND, OR, and XOR. The 32-bit inputs are:

1) the output of the write-mode unit and

2) the display latch (not necessarily the value at the frame buffer address of the write).

An application that wishes to perform ROPs on the source and destination must first byte read the address (to load the latch) and then immediately write a byte to the same address. The ALU has no effect in Write Mode 1.

The bit mask unit does not provide a true bit mask. Instead, it selects between the ALU output and the display latch. The mask is an 8-bit value, and bit n of the mask makes the selection for bit n of all four bytes of the result (a zero selects the latch). No bit masking occurs in Write Mode 1.

The VGA hardware of the GXLV processor does not implement Write Mode 1 directly, but it can be indirectly implemented by setting the BitMask to zero and the ALU mode to COPY. This is done by the SMM code so there are no compatibility issues with applications.

#### 4.6.2.2 GXLV VGA Hardware

The GXLV processor core contains hardware to detect VGA accesses and generate SMI interrupts. The graphics pipeline contains hardware to detect and process reads and writes to VGA memory. The VGA memory on the GXLV processor is partitioned from system memory. The GXLV processor has the following hardware components to assist the VGA emulation software.

- SMI Generation
- VGA Range Detection
- VGA Sequencer
- VGA Write/Read Path
- VGA Address Generator
- VGA Memory

#### 4.6.2.3 SMI Generation

VGA emulation software is notified of VGA memory accesses by an SMI generated in dedicated circuitry in the processor core that detects and traps memory accesses. The SMI generation hardware for VGA memory addresses is in the second stage of instruction decoding on the processor core. This is the earliest stage of instruction decode where virtual addresses have been translated to physical addresses. Trapping after the execution stage is impractical, because memory write buffering will allow subsequent instructions to execute.

The VGA emulation code requires the SMI to be generated immediately when a VGA access occurs. The SMI generation hardware can optionally exclude areas of VGA memory, based on a 32-bit register which has a control bit for each 2 KB region of the VGA memory window. The control bit determines whether or not an SMI interrupt is generated for the corresponding region. The purpose of this hardware is to allow the VGA emulation software to disable SMI interrupts in VGA memory regions that are not currently displayed.

For direct display modes (8 bpp or 16 bpp) in the display controller, Virtual VGA can operate without SMI generation.

The SMI generation circuit on the GXLV processor has configuration registers to control and mask SMI interrupts in the VGA memory space.

#### 4.6.2.4 VGA Range Detection

The VGA range detection circuit is similar to the SMI generation hardware, however, it resides in the internal bus interface address mapping unit. The purpose of this hardware is to notify the graphics pipeline when accesses to the VGA memory range A0000h to BFFFFh are detected. The graphics pipeline has VGA read and write path hardware to process VGA memory accesses. The VGA range detection can be configured to trap VGA memory accesses in one or more of the following ranges: A0000h to AFFFFh (EGA,VGA), B0000h to B7FFFh (MDA), or B8000h to BFFFFh (CGA).

#### 4.6.2.5 VGA Sequencer

The VGA sequencer is located at the front end of the graphics pipeline. The purpose of the VGA sequencer is to divide up multiple-byte read and write operations into a sequence of single-byte read and write operations. 16-bit or 32-bit X-bus write operations to VGA memory are divided into 8-bit write operations and sent to the VGA write path. 16-bit or 32-bit X-bus read operations from VGA memory are accumulated from 8-bit read operations over the VGA read path. The sequencer generates the lower two bits of the address.

#### 4.6.2.6 VGA Write/Read Path

The VGA write path implements standard VGA write operations into VGA memory. No SMI is generated for write path operations when the VGA access is not displayed. When the VGA access is displayed, an SMI is generated so that the SMI emulation can update the frame buffer. The VGA write path converts 8-bit write operations from the sequencer into 32-bit VGA memory write operations. The operations performed by the VGA write path include data rotation, raster operation (ALU), bit masking, plane select, plane enable, and write modes.

The VGA read path implements standard VGA read operations from VGA memory. No SMI is needed for read-path operations. The VGA read path converts 32-bit read operations from VGA memory to 8-bit data back to the sequencer. The basic operations performed by the VGA read path include color compare, plane-read select, and read modes.

#### 4.6.2.7 VGA Address Generator

The VGA address generator translates VGA memory addresses up to the address where the VGA memory resides on the GXLV processor. The VGA address generator requires the address from the VGA access (A0000h to BFFFFh), the base of the VGA memory on the GXLV processor, and various control bits. The control bits are necessary because addressing is complicated by odd/even and Chain 4 addressing modes.

#### 4.6.2.8 VGA Memory

The VGA memory requires 256 KB of memory organized as 64 KB by 32 bits. The VGA memory is implemented as part of system memory. The GXLV processor partitions system memory into two areas, normal system memory and graphics memory. System memory is mapped to the normal physical address of the DRAM, starting at zero and ending at memory size. Graphics memory is mapped into high physical memory, contiguous to the registers and dedicated cache of the GXLV processor. The graphics memory includes the frame buffer, compression buffer, cursor memory, and VGA memory. The VGA memory is mapped on a 256 KB boundary to simplify the address generation

## Integrated Functions (Continued)

### 4.6.3 VGA Configuration Registers

SMI generation can be configured to trap VGA memory accesses in one of the following ranges:

A0000h to AFFFFh (EGA,VGA),
B0000h to B7FFFh (MDA),
or B8000h to BFFFFh (CGA).

Range selection is accomplished through programmable bits in the VGACTL register (Index B9h). Fine control can be exercised within the range selected to allow off-screen accesses to occur without generating SMIs.

SMI generation can also separately control the following I/O ranges: 3B0h to 3BFh, 3C0h to 3CFh, and 3D0h to 3DFh. The BC_XMAP_1 register (GX_BASE+8004h) in the Internal Bus Interface Unit has an enable/disable bit for each of the address ranges above.

The VGA control register (VGACTL) provides control for SMI generation through an enable bit for memory address ranges A0000h to BFFFFh. Each bit controls whether or not SMI is generated for accesses to the corresponding address range. The default value of this register is zero so that VGA accesses will not be trapped on systems with an external VGA card.

The VGA Mask register (VGAM) has 32 bits that can selectively mask 2 KB regions within the VGA memory region A0000h to AFFFFh. If none of the three regions is enabled in VGACTL, then the contents of VGAM are ignored. VGAM can be used to prevent the occurrence of SMI when non-displayed VGA memory is accessed. This is an enhancement that improves performance for double-buffered applications only.

## Integrated Functions (Continued)

Table 4-36 summarizes the VGA Configuration Registers. Detailed register/bit formats are given in Table 4-37. See Section 3.3.2.2 “Configuration Registers” on page 50 on how to access these registers.

**Table 4-36. VGA Configuration Registers Summary**

| Index | Type | Name/Function | Default Value |
|---|---|---|---|
| B9h | R/W | VGACTL: VGA Control Register | 00h (SMI generation disabled) |
| BAh-BDh | R/W | VGAM: VGA Mask Register | xxxxxxxxh |

**Table 4-37. VGA Configuration Registers**

| Bit | Description |
|---|---|
| **Index B9h** | **VGACTL Register (R/W)** **Default Value = 00h** |
| 7:3 | **Reserved**: Set to 0. |
| 2 | SMI generation for VGA memory range B8000h to BFFFFh: 0 = Disable; 1 = Enable. |
| 1 | SMI generation for VGA memory range B0000h to B7FFFh: 0 = Disable; 1 = Enable. |
| 0 | SMI generation for VGA memory range A0000h to AFFFFh: 0 = Disable; 1 = Enable. |
| **Index BAh-BDh** | **VGAM Register (R/W)** **Default Value = xxxxxxxxh** |
| 31 | SMI generation for address range AF800h to AFFFFh: 0 = Disable; 1 = Enable. |
| 30 | SMI generation for address range AF000h to AF7FFh: 0 = Disable; 1 = Enable. |
| 29 | SMI generation for address range AE800h to AEFFFh: 0 = Disable; 1 = Enable. |
| 28 | SMI generation for address range AE000h to AE7FFh: 0 = Disable; 1 = Enable. |
| 27 | SMI generation for address range AD800h to ADFFFh: 0 = Disable; 1 = Enable. |
| 26 | SMI generation for address range AD000h to AD7FFh: 0 = Disable; 1 = Enable. |
| 25 | SMI generation for address range AC800h to ACFFFh: 0 = Disable; 1 = Enable. |
| 24 | SMI generation for address range AC000h to AC7FFh: 0 = Disable; 1 = Enable. |
| 23 | SMI generation for address range AB800h to ABFFFh: 0 = Disable; 1 = Enable. |
| 22 | SMI generation for address range AB000h to AB7FFh: 0 = Disable; 1 = Enable. |
| 21 | SMI generation for address range AA800h to AAFFFh: 0 = Disable; 1 = Enable. |
| 20 | SMI generation for address range AA000h to AA7FFh: 0 = Disable; 1 = Enable. |
| 19 | SMI generation for address range A9800h to A9FFFh: 0 = Disable; 1 = Enable. |
| 18 | SMI generation for address range A9000h to A97FFh: 0 = Disable; 1 = Enable. |
| 17 | SMI generation for address range A8800h to A8FFFh: 0 = Disable; 1 = Enable. |
| 16 | SMI generation for address range A8000h to A87FFh: 0 = Disable; 1 = Enable. |
| 15 | SMI generation for address range A7800h to A7FFFh: 0 = Disable; 1 = Enable. |
| 14 | SMI generation for address range A7000h to A77FFh: 0 = Disable; 1 = Enable. |
| 13 | SMI generation for address range A6800h to A6FFFh: 0 = Disable; 1 = Enable. |
| 12 | SMI generation for address range A6000h to A67FFh: 0 = Disable; 1 = Enable. |
| 11 | SMI generation for address range A5800h to A5FFFh: 0 = Disable; 1 = Enable. |
| 10 | SMI generation for address range A5000h to A57FFh: 0 = Disable; 1 = Enable. |
| 9 | SMI generation for address range A4800h to A4FFFh: 0 = Disable; 1 = Enable. |
| 8 | SMI generation for address range A4000h to A47FFh: 0 = Disable; 1 = Enable. |
| 7 | SMI generation for address range A3800h to A3FFFh: 0 = Disable; 1 = Enable. |
| 6 | SMI generation for address range A3000h to A37FFh: 0 = Disable; 1 = Enable. |
| 5 | SMI generation for address range A2800h to A2FFFh: 0 = Disable; 1 = Enable. |
| 4 | SMI generation for address range A2000h to A27FFh: 0 = Disable; 1 = Enable. |
| 3 | SMI generation for address range A1800h to A1FFFh: 0 = Disable; 1 = Enable. |
| 2 | SMI generation for address range A1000h to A17FFh: 0 = Disable; 1 = Enable. |
| 1 | SMI generation for address range A0800h to A0FFFh: 0 = Disable; 1 = Enable. |
| 0 | SMI generation for address range A0000h to A07FFh: 0 = Disable; 1 = Enable. |

### 4.6.4 Virtual VGA Register Descriptions

This section describes the registers contained in the graphics pipeline used for VGA emulation. The graphics pipeline maps 200h locations starting at GX_BASE+8100h. Refer to Section 4.1.2 “Control Registers” on page 99 for instructions on accessing these registers.

The registers are summarized in Table 4-38, followed by detailed bit formats in Table 4-39.

**Table 4-38. Virtual VGA Register Summary**

| GX_BASE+ Memory Offset | Type | Name/Function | Default Value |
|---|---|---|---|
| 8140h-8143h | R/W | **GP_VGA_WRITE**<br>Graphics Pipeline VGA Write Patch Control Register: Controls the VGA memory write path in the graphics pipeline. | xxxxxxxxh |
| 8144h-8147h | R/W | **GP_VGA_READ**<br>Graphics Pipeline VGA Read Patch Control Register: Controls the VGA memory read path in the graphics pipeline. | 00000000h |
| 8210h-8213h | R/W | **GP_VGA_BASE VGA**<br>Graphics Pipeline VGA Memory Base Address Register: Specifies the offset of the VGA memory, starting from the base of graphics memory. | xxxxxxxxh |
| 8214h-8217h | R/W | **GP_VGA_LATCH**<br>Graphics Pipeline VGA Display Latch Register: Provides a memory mapped way to read or write the VGA display latch. | xxxxxxxxh |

## Integrated Functions (Continued)

### Table 4-39. Virtual VGA Registers

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+8140h-8143h** | **GP_VGA_WRITE Register (R/W)** | **Default Value = xxxxxxxxh** |
| 31:28 | RSVD | **Reserved:** Set to 0. |
| 27:24 | MAP_MASK | **Map Mask:** Enables planes 3 through 0 for writing. Combined with chain control to determine the final enables. |
| 23:21 | RSVD | **Reserved:** Set to 0. |
| 20 | W3 | **Write Mode 3:** Selects write mode 3 by using the bit mask with the rotated data. |
| 19 | W2 | **Write Mode 2:** Selects write mode 2 by controlling set/reset. |
| 18:16 | RC | **Rotate Count:** Controls the 8-bit rotator. |
| 15:12 | SRE | **Set/Reset Enable:** Enables the set/reset value for each plane. |
| 11:8 | SR | **Set/Reset:** Selects 1 or 0 for each plane if enabled. |
| 7:0 | BIT_MASK | **Bit Mask:** Selects data from the data latches (last read data). |
| **GX_BASE+8144h-8147h** | **GP_VGA_READ Register (R/W)** | **Default Value = 00000000h** |
| 31:18 | RSVD | **Reserved:** Set to 0. |
| 17:16 | RMS | **Read Map Select:** Selects which plane to read in read mode 0 (Chain 2 and Chain 4 inactive). |
| 15 | F15 | **Force Address Bit 15:** Forces address bit 15 to 0. |
| 14 | PC4 | **Packed Chain 4:** Provides 64 KB of packed pixel addressing when used with Chain 4 mode. This bit causes the VGA addresses to be shifted right by 2 bits. |
| 13 | C4 | **Chain 4 Mode:** Selects Chain 4 mode for both read operations and write operations. This overrides bits 10 and 9 of this register. |
| 12 | PB | **Page Bit:** Becomes LSB of address if COE is set high. |
| 11 | COE | **Chain Odd/Even:** Selects PB rather than A0 for least-significant VGA address bit. |
| 10 | W2 | **Write Chain 2 Mode:** Selects Chain 2 mode for write operations. Bit 13 overrides this bit. |
| 9 | R2 | **Read Chain 2 Mode:** Selects Chain 2 mode for read operations. Bit 13 overrides this bit. |
| 8 | RM | **Read Mode:** Selects between read mode 0 (normal) and read mode 1 (color compare). |
| 7:4 | CCM | **Color Compare Mask:** Selects planes to include in the color comparison (read mode 1). |
| 3:0 | CC | **Color Compare:** Specifies value of each plane for color comparison (read mode 1). |
| **GX_BASE+8210h-8213h** | **GP_VGA_BASE (R/W)** | **Default Value = xxxxxxxxh** |
| 31:14 | RSVD | **Reserved:** Set to 0. |
| 13:8 | VGA_RD_BASE | **Read Base Address:** The VGA base address is added to the graphics memory base to specify where VGA memory starts. The VGA base address provides address bits [19:14] when mapping VGA accesses into graphics memory. This allows the VGA base address to start on any 64 KB boundary within the 4 MB of graphics memory. This register is used for reads to the VGA trace buffer. |
| 7:6 | RSVD | **Reserved:** Set to 0. |
| 5:0 | VGA_WR_BASE | **Write Base Address:** The VGA base address is added to the graphics memory base to specify where VGA memory starts. The VGA base address provides address bits [19:14] when mapping VGA accesses into graphics memory. This allows the VGA base address to start on any 64 KB boundary within the 4 MB of graphics memory. This register is used for writes to the VGA trace buffer. |
| **GX_BASE+8214h-8217h** | **GP_VGA_LATCH Register (R/W)** | **Default Value = xxxxxxxxh** |
| 31:0 | LATCH | **Display Latch:** Specifies the value in the VGA display latch. VGA read operations cause VGA frame buffer data to be latched in the display latch. VGA write operations can use the display latch as a source of data for VGA frame buffer write operations. |

# Integrated Functions (Continued)

## 4.7 PCI CONTROLLER

The GXLV processor includes an integrated PCI controller with the following features.

### 4.7.1 X-Bus PCI Slave

- 16-byte PCI write buffer
- 16-byte PCI read buffer from X-bus
- Supports cache line bursting
- Write/Inv line support
- Pacing of data for read or write operations with X-bus
- No active byte enable transfers supported

### 4.7.2 X-Bus PCI Master

- 16 byte X-bus to PCI write buffer
- Configuration read/write Support
- Int Acknowledge support
- Lock conversion
- Support fast back-to-back cycles as slave

### 4.7.3 PCI Arbiter

- Fixed, rotating, hybrid, or ping-pong arbitration (programmable)
- Support four masters, three on PCI
- Internal REQ for CPU
- Master retry mask counter
- Master dead timer
- Resource or total system lock support

### 4.7.4 Generating Configuration Cycles

Configuration space is a physical address space unique to PCI. Configuration Mechanism #1 must be used by software to generate configuration cycles. Two DWORD I/O locations are used in this mechanism. The first DWORD location (CF8h) references a read/write register that is named CONFIG_ADDRESS. The second DWORD address (CFCh) references a register named CONFIG_DATA. The general method for accessing configuration space is to write a value into CONFIG_ADDRESS that specifies a PCI bus, a device on that bus, and a configuration register in that device being accessed. A read or write to CONFIG_DATA will then cause the bridge to translate that CONFIG_ADDRESS value to the requested configuration cycle on the PCI bus.

### 4.7.5 Generating Special Cycles

A special cycle is a broadcast message to the PCI bus. Two hardcoded special cycle messages are defined in the command encode: HALT and SHUTDOWN. Software can also generate special cycles by using special cycle generation for configuration mechanism #1 as described in the PCI Specification 2.1 and briefly described here. To initiate a special cycle from software, the host must write a value to CONFIG_ADDRESS encoded as shown in Table 4-40.

The next value written to CONFIG_DATA is the encoded special cycle. Type 0 or Type 1 conversion will be based on the Bus Bridge number matching the GXLV processor's bus number of 00h.

**Table 4-40. Special Cycle Code to CONFIG_ADDRESS**

| 31 | 30 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 0 0 0 0 0 0 | Bus No. = Bridge | | | | | | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| CONFIG ENABLE | RSVD | BUS NUMBER | | | | | | | | DEVICE NUMBER | | | | | FUNCTION NUMBER | | | REGISTER NUMBER | | | | | | TRANS LATION TYPE | |

**Note:** See Table 4-41 on page 167, bits [1:0] for translation type.

### 4.7.6 PCI Configuration Space Control Registers

There are two registers in this category: CONFIG_ADDRESS and CONFIG_DATA.

The CONFIG_ADDRESS register contains the address information for the next configuration space access to CONFIG_DATA. Only DWORD accesses are permitted to this register all others will be forwarded as normal I/O cycles to the PCI bus.

The CONFIG_DATA register contains the data that is sent or received during a PCI configuration space access.

Table 4-41 gives the bit formats for these two registers.

**Table 4-41. PCI Configuration Registers**

| Bit | Name | Description |
|---|---|---|
| **I/O Offset 0CF8h-0CFBh** | | **CONFIG_ADDRESS Register (R/W)** **Default Value = 00000000h** |
| 31 | GFC_EN | **CONFIG ENABLE:** Determines when accesses should be translated to configuration cycles on the PCI bus, or treated as a normal I/O operation. This register will be updated only on full DWORD I/O operations to the CONFIG_ADDRESS. Any other accesses are treated as normal I/O cycles in order to allow I/O devices to use BYTE or WORD registers at the same address and remain unaffected. Once bit 31 is set high, subsequent accesses to CONFIG_DATA are then translated to configuration cycles.<br>1 = Generate configuration cycles.<br>0 = Normal I/O cycles. |
| 30:24 | RSVD | **Reserved:** Set to 0. |
| 23:16 | BUS | **Bus:** Specifies a PCI bus number in the hierarchy of 1 to 256 buses. |
| 15:11 | DEVICE | **Device:** Selects a device on a specified bus. A device value of 00h will select the GXLV processor if the bus number is also 00h. DEVICE values of 01h to 15h will be mapped to AD[31:11], so only 21 of the 32 possible devices are supported. A DEVICE value of 00001b will map to AD[11] while a device of 10101b will map to AD[31]. |
| 10:8 | FUNCTION | **Function:** Selects a function in a multi-function device. |
| 7:2 | REGISTER | **Register:** Chooses a configuration DWORD space register in the selected device. |
| 1:0 | TT | **Translation Type Bits:** These bits indicate if the configuration access is local or one that requires translation through other bridges to another PCI bus. When an access occurs to the CONFIG_DATA address and the specified bus number matches the GXLV processor's bus number (00h), then a Type 0 translation takes place.<br>For a Type 0 translation, the CONFIG_ADDRESS register values are translated to AD lines on the PCI bus. Note that bits [10:2] are passed unchanged. The DEVICE value is mapped to one of 21 AD lines. The translation type bits are set to 00 to indicate a transaction on the local PCI bus.<br>When an access occurs to the CONFIG_DATA address and the specified bus number is not 00h (Type 1), the GXLV processor passes this cycle to the PCI bus by copying the contents of the CONFIG_ADDRESS register onto the AD lines during the address phase of the cycle while driving the translation type bits AD[1:0] to 01. |
| **I/O Offset 0CFCh-0CFFh** | | **CONFIG_DATA (R/W)** **Default Value = 00000000h** |
| 31:0 | CONFIG_DATA | **Configuration Data Register:** Contains the data that is sent or received during a PCI configuration space access. The register accessed is determined by the value in the CONFIG_ADDRESS register. The CONFIG_DATA register supports BYTE, WORD, or DWORD accesses. To access this register, bit 31 of the CONFIG_ADDRESS register must be set to 0 and a full DWORD I/O access must be done. Configuration cycles are performed when bit 31 of the CONFIG_ADDRESS register is set to 1 |

## Integrated Functions (Continued)

### 4.7.7 PCI Configuration Space Registers

To access the internal PCI configuration registers of the GXLV processor, the Configuration Address Register (CONFIG_ADDRESS) must be written as a DWORD using the format shown in Table 4-42. Any other size will be interpreted as an I/O write to Port 0CF8h. Also, when entering the Configuration Index, only the six most significant bits of the offset are used, and the two least significant bits must be 00b.

Table 4-43 summarizes the registers located within the Configuration Space. The tables that follow, give detailed register/bit formats.

**Table 4-42. Format for Accessing the Internal PCI Configuration Registers**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Reserved | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Configuration Index | | | | | | 0 | 0 |

**Table 4-43. PCI Configuration Space Register Summary**

| Index | Type | Name/Function | Default Value |
|---|---|---|---|
| 00h-01h | RO | Vendor Identification | 1078h |
| 02h-03h | RO | Device Identification | 0001h |
| 04h-05h | R/W | PCI Command | 0007h |
| 06h-07h | R/W | Device Status | 0280h |
| 08h | RO | Revision Identification | 00h |
| 09h-0Bh | RO | Class Code | 060000h |
| 0Ch | RO | Cache Line Size | 00h |
| 0Dh | R/W | Latency Timer | 00h |
| 0Eh-3Fh | -- | Reserved | 00h |
| 40h | R/W | PCI Control Function 1 | 00h |
| 41h | R/W | PCI Control Function 2 | 96h |
| 42h | -- | Reserved | 00h |
| 43h | R/W | PCI Arbitration Control 1 | 80h |
| 44h | R/W | PCI Arbitration Control 2 | 00h |
| 45h-FFh | -- | Reserved | 00h |

### Table 4-44. PCI Configuration Registers

| Bit | Name | Description |
|---|---|---|
| **Index 00h-01h** | | **Vendor Identification Register (RO)** — **Default Value = 1078h** |
| 31:0 | VID (RO) | **Vendor Identification Register (Read Only):** The combination of this value and the device ID uniquely identifies any PCI device. The Vendor ID is the ID given to National Semiconductor Corporation by the PCI SIG. |
| **Index 02h-03h** | | **Device Identification Register (RO)** — **Default Value = 0001h** |
| 31:0 | DIR (RO) | **Device Identification Register (Read Only):** This value along with the vendor ID uniquely identifies any PCI device. |
| **Index 04h-05h** | | **PCI Command Register (R/W)** — **Default Value = 0007h** |
| 15:10 | RSVD | **Reserved:** Set to 0. |
| 9 | FBE | **Fast Back-to-Back Enable (RO):** As a master, the GXLV processor does not support this function.<br>This bit returns 0. |
| 8 | SERR | **SERR# Enable:** This is used as an output enable gate for the SERR# driver. |
| 7 | WAT | **Wait Cycle Control:** GXLV processor does not do address/data stepping.<br>This bit is always set to 0. |
| 6 | PE | **Parity Error Response:**<br>0 = GXLV processor ignores parity errors on the PCI bus.<br>1 = GXLV processor checks for parity errors. |
| 5 | VPS | **VGA Palette Snoop:** GXLV processor does not support this function.<br>This bit is always set to 0. |
| 4 | MS | **Memory Write and Invalidate Enable:** As a master, the GXLV processor does not support this function.<br>This bit is always set to 0. |
| 3 | SPC | **Special Cycles:** GXLV processor does not respond to special cycles on the PCI bus.<br>This bit is always set to 0. |
| 2 | BM | **Bus Master:**<br>0 = GXLV processor does not perform master cycles on the PCI bus.<br>1 = GXLV processor can act as a bus master on the PCI bus. |
| 1 | MS | **Memory Space:** GXLV processor will always respond to memory cycles on the PCI bus.<br>This bit is always set to 1. |
| 0 | IOS | **I/O Space:** GXLV processor will not respond to I/O accesses from the PCI bus.<br>This bit is always set to 1. |
| **Index 06h-07h** | | **PCI Device Status Register (RO, R/W Clear)** — **Default Value = 0280h** |
| 15 | DPE | **Detected Parity Error:** When a parity error is detected, this bit is set to 1.<br>This bit can be cleared to 0 by writing a 1 to it. |
| 14 | SSE | **Signaled System Error:** This bit is set whenever SERR# is driven active. |
| 13 | RMA | **Received Master Abort:** This bit is set whenever a master abort cycle occurs. A master abort will occur whenever a PCI cycle is not claimed except for special cycles.<br>This bit can be cleared to 0 by writing a 1 to it. |
| 12 | RTA | **Received Target Abort:** This bit is set whenever a target abort is received while the GXLV processor is master of the cycle.<br>This bit can be cleared to 0 by writing a 1 to it. |
| 11 | STA | **Signaled Target Abort:** This bit is set whenever the GXLV processor signals a target abort. A target abort is signaled when an address parity occurs for an address that hits in the GXLV processor's address space.<br>This bit can be cleared to 0 by writing a 1 to it. |
| 10:9 | DT | **Device Timing:** The GXLV processor performs medium DEVSEL# active for addresses that hit into the GXLV processor address space. These two bits are always set to 01.<br>00 = Fast<br>01 = Medium<br>10 = Slow<br>11 = Reserved |

## Integrated Functions (Continued)

### Table 4-44. PCI Configuration Registers (Continued)

| Bit | Name | Description |
|---|---|---|
| 8 | DPD | **Data Parity Detected:** This bit is set when all three conditions are met.<br>1) GXLV processor asserted PERR# or observed PERR# asserted;<br>2) GXLV processor is the master for the cycle in which the PERR# occurred; and<br>3) PE (bit 6 of Command Register) is enabled.<br>This bit can be cleared to 0 by writing a 1 to it. |
| 7 | FBS | **Fast Back-to-Back Capable:** As a target, the processor is capable of accepting Fast Back-to-Back transactions.<br>This bit is always set to 1. |
| 6:0 | RSVD | **Reserved:** Set to 0. |
| **Index 08h** | | **Revision Identification Register (RO)** — **Default Value = 00h** |
| 7:0 | RID (RO) | **Revision ID (Read Only):** This register contains the revision number of the GXLV design. |
| **Index 09h-0Bh** | | **Class Code Register (RO)** — **Default Value = 060000h** |
| 23:16 | CLASS | **Class Code:** The class code register is used to identify the generic function of the device. The GXLV processor is classified as a host bridge device (06). |
| 15:0 | RSVD (RO) | **Reserved (Read Only)** |
| **Index 0Ch** | | **Cache Line Size Register (RO)** — **Default Value = 00h** |
| 7:0 | CACHELINE | **Cache Line Size (Read Only):** The cache line size register specifies the system cache line size in units of 32-bit words. This function is not supported in the GXLV processor. |
| **Index 0Dh** | | **Latency Timer Register (R/W)** — **Default Value = 00h** |
| 7:5 | RSVD | **Reserved:** Set to 0. |
| 4:0 | LAT_TIMER | **Latency Timer:** The latency timer as used in this implementation will prevent a system lockup resulting from a slave that does not respond to the master. If the register value is set to 00h, the timer is disabled. Otherwise, Timer represents the 5 MSBs of an 8-bit counter. The counter will reset on each valid data transfer. If the counter expires before the next TRDY# is received active, then the slave is considered to be incapable of responding, and the master will stop the transaction with a master abort and flag an SERR# active. This would also keep the master from being retried forever by a slave device that continues to issue retries. In these cases, the master will also stop the cycle with a master abort. |
| **Index 0Eh-3Fh** | | **Reserved** — **Default Value = 00h** |
| **Index 40h** | | **PCI Control Function 1 Register (R/W)** — **Default Value = 00h** |
| 7 | RSVD | **Reserved:** Set to 0. |
| 6 | SW | **Single Write Mode:** GXLV as a PCI slave supports:<br>0 = Multiple PCI write cycles<br>1 = Single cycle write transfers on the PCI bus. The slave will perform a target disconnect with the first data transferred. |
| 5 | SR | **Single Read Mode:** GXLV as a PCI slave supports:<br>0 = Multiple PCI read cycles.<br>1 = Single cycle read transfers on the PCI bus. The slave will perform a target disconnect with the first data transferred. |
| 4 | RXBNE | **Force Retry when X-Bus Buffers are Not Empty:** GXLV as a PCI slave:<br>0 = Accepts the PCI cycle with data in the PCI master write buffers. The data in the PCI master write buffers will not be affected or corrupted. The PCI master holds request active indicating the need to access the PCI bus.<br>1 = Retries cycles if the PCI master X-Bus write buffers contain buffered data. |
| 3 | SWBE | **PCI Slave Write Buffer Enable:** GXLV PCI slave write buffers: 0 = Disable; 1 = Enable. |
| 2 | CLRE | **PCI Cache Line Read Enable:** Read operations from the PCI into the GXLV processor:<br>0 = Single cycle unless a read multiple or memory read line command is used.<br>1 = Cause a cache line read to occur. |
| 1 | XBE | **X-Bus Burst Enable:** Enable X-Bus bursting when an external master performs PCI write/invalidate cycles. 0 = Disable; 1 = Enable.<br>(This bit does not control read bursting; bit 2 does.) |
| 0 | RSVD | **Reserved:** Should return a value of 0. |

## Integrated Functions (Continued)

### Table 4-44. PCI Configuration Registers (Continued)

| Bit | Name | Description |
|---|---|---|
| **Index 41h** | | **PCI Control Function 2 Register (R/W)** **Default Value = 96h** |
| 7 | RSVD | **Reserved:** Set to 0. |
| 6 | RW_CLK | **Raw Clock:** A debug signal used to view internal clock operation. 0 = Disable; 1 = Enable. |
| 5 | PFS | **PERR# forces SERR#:** PCI master drives an active SERR# anytime it also drives or receives an active PERR#: 0 = Disable; 1 = Enable. |
| 4 | XWB | **X-Bus to PCI Write Buffer:** Enable GXLV processor PCI master's X-Bus write buffers (non-locked memory cycles are buffered, I/O cycles and lock cycles are not buffered): 0 = Disable; 1 = Enable. |
| 3:2 | SDB | **Slave Disconnect Boundary:** GXLV as a PCI slave issues a disconnect with burst data when it crosses line boundary:<br>00 = 128 bytes<br>01 = 256 bytes<br>10 = 512 bytes<br>11 = 1024 bytes<br>Works in conjunction with bit 1. |
| 1 | SDBE | **Slave Disconnect Boundary Enable:** GXLV as a PCI slave:<br>0 = Disconnects on boundaries set by bits [3:2].<br>1 = Disconnects on cache line boundary which is 16 bytes. |
| 0 | XWS | **X-Bus Wait State Enable:** The PCI slave acting as a master on the X-Bus will insert wait states on write cycles for data setup time. 0 = Disable; 1 = Enable. |
| **Index 42h** | | **Reserved** **Default Value = 00h** |
| **Index 43h** | | **PCI Arbitration Control 1 Register (R/W)** **Default Value = 80h** |
| 7 | BG | **Bus Grant:**<br>0 = Grants bus regardless of X-Bus buffers.<br>1 = Grants bus only if X-Bus buffers are empty. |
| 6 | RSVD | **Reserved:** Set to 1. |
| 5 | RME2 | **REQ2# Retry Mask Enable:** Arbiter allows the REQ2# to be masked based on the master retry mask in bits [2:1]: 0 = Disable; 1 = Enable. |
| 4 | RME1 | **REQ1# Retry Mask Enable:** Arbiter allows the REQ1# to be masked based on the master retry mask in bits [2:1]: 0 = Disable; 1 = Enable. |
| 3 | RME0 | **REQ0# Retry Mask Enable:** Arbiter allows the REQ0# to be masked based on the master retry mask in bits [2:1]: 0 = Disable; 1 = Enable. |
| 2:1 | MRM | **Master Retry Mask:** When a target issues a retry to a master, the arbiter can mask the request from the retried master in order to allow other lower order masters to gain access to the PCI bus:<br>00 = No retry mask<br>01 = Mask for 16 PCI clocks<br>10 = Mask for 32 PCI clocks<br>11 = Mask for 64 PCI clocks |
| 0 | HXR | **Hold X-bus on Retries:** Arbiter holds the X-Bus X_HOLD for two additional clocks to see if the retried master will request the bus again: 0 = Disable; 1 = Enable<br>(This may prevent retry thrashing in some cases.) |

## Integrated Functions (Continued)

### Table 4-44. PCI Configuration Registers (Continued)

| Bit | Name | Description |
|---|---|---|
| **Index 44h** | | **PCI Arbitration Control 2 Register (R/W)** **Default Value = 00h** |
| 7 | PP | **Ping-Pong:**<br>0 = Arbiter grants the processor bus per the setting of bits [2:0].<br>1 = Arbiter grants the processor bus ownership of the PCI bus every other arbitration cycle. |
| 6:4 | FAC | **Fixed Arbitration Controls:** These bits control the priority under fixed arbitration. The priority table is as follows (priority listed highest to lowest):<br>000 = REQ0#, REQ1#, REQ2#<br>001 = REQ1#, REQ0#, REQ2#<br>010 = REQ0#, REQ2#,REQ1#<br>011 = Reserved<br>100 = REQ1#, REQ2#, REQ0#<br>101 = Reserved<br>110 = REQ2#, REQ1#, REQ0#<br>111 = REQ2#, REQ0#, REQ1#<br>**Note:** The rotation arbitration bits [2:0] must be set to 000 for full fixed arbitration. If rotation bits are not set to 000, then hybrid arbitration will occur. If Ping-Pong is enabled (bit 7 = 1), the processor will have priority every other arbitration. In this mode, the arbiter grants the PCI bus to a master and ignores all other requests. When the master finishes, the processor will be guaranteed access. At this point PCI requests will again be recognized. This will switch arbitration from CPU to PCI to CPU to PCI, etc. |
| 3 | RSVD | **Reserved:** Set to 0. |
| 2:0 | RAC | **Rotating Arbitration Controls:** These bits control the priority under rotating arbitration.<br>000 = Fixed arbitration will occur.<br>111 = Full rotating arbitration will occur.<br>When these bits are set to other values, hybrid arbitration will occur. |
| **Index 45h-FFh** | | **Reserved** **Default Value = 00h** |

## Integrated Functions (Continued)

### 4.7.8 PCI Cycles

The following sections and diagrams provide the functional relationships for PCI cycles.

#### 4.7.8.1 PCI Read Transaction

A PCI read transaction consists of an address phase and one or more data phases. Data phases may consist of wait cycles and a data transfer. Figure 4-18 illustrates a PCI read transaction. In this example, there are three data phases.

The address phase begins on clock 2 when FRAME# is asserted. During the address phase, AD[31:0] contains a valid address and C/BE[3:0]# contains a valid bus command. The first data phase begins on clock 3. During the data phase, AD[31:0] contains data and C/BE[3:0]# indicate which byte lanes of AD[31:0] carry valid data. The first data phase completes with zero delay cycles. However, the second phase is delayed one cycle because the target was not ready so it deasserted TRDY# on clock 5. The last data phase is delayed one cycle because the master deasserted IRDY# on clock 7.

For additional information refer to Chapter 3.3.1, Read Transaction, of the PCI Local Bus Specification, Revision 2.1.

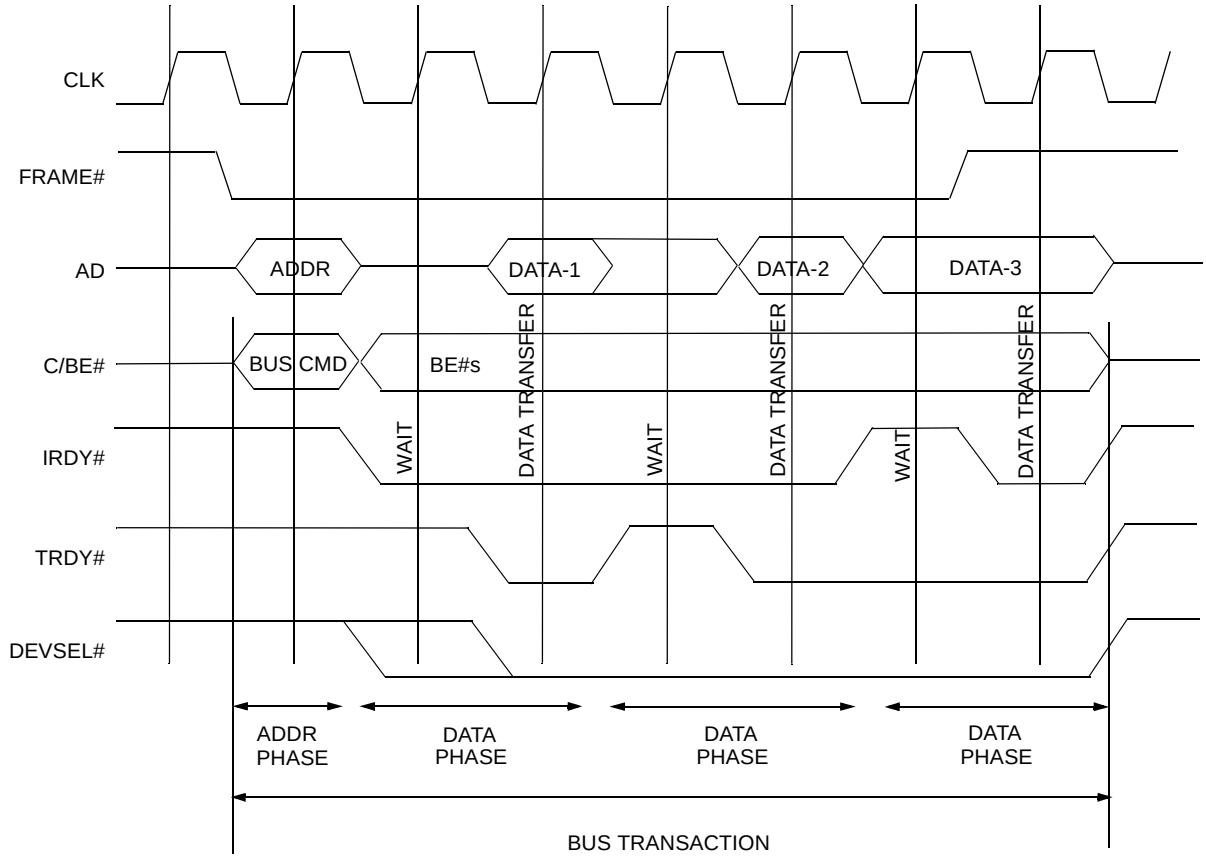



**Figure 4-18. Basic Read Operation**

## Integrated Functions (Continued)

### 4.7.8.2 PCI Write Transaction

A PCI write transaction is similar to a PCI read transaction, consisting of an address phase and one or more data phases. Since the master provides both address and data, no turnaround cycle is required following the address phase. The data phases work the same for both read and write transactions. Figure 4-19 illustrates a write transaction.

The address phase begins on clock 2 when FRAME# is asserted. The first and second data phases complete without delays. During data phase 3, the target inserts three wait cycles by deasserting TRDY#.

For additional information refer to Chapter 3.3.2, Write Transaction, of the PCI Local Bus Specification, Revision 2.1.

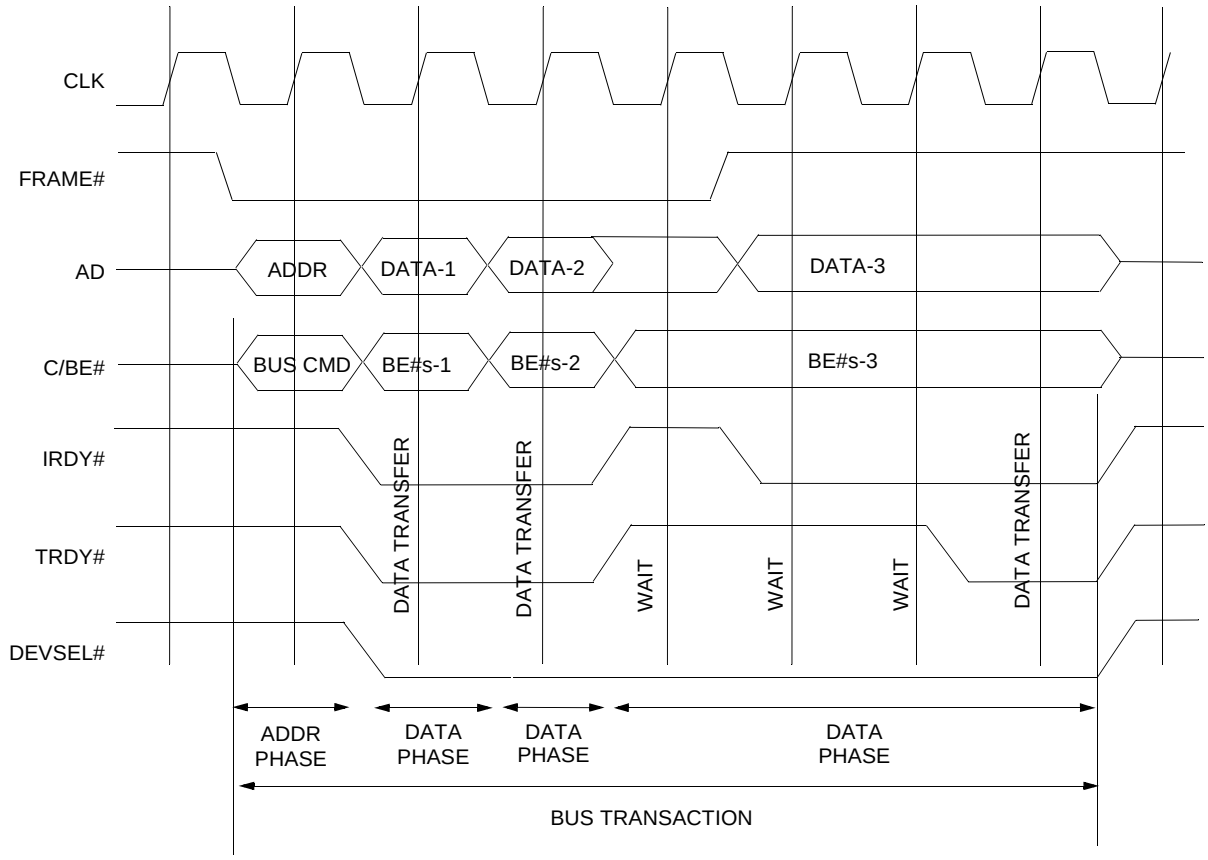



**Figure 4-19. Basic Write Operation**

#### 4.7.8.3 PCI Arbitration

An agent requests the bus by asserting its REQ#. Based on the arbitration scheme set in the PCI Arbitration Control 2 Register (Index 44h), the GXLV processor's PCI arbiter will grant the request by asserting GNT#. Figure 4-20 illustrates basic arbitration.

REQ#-a is asserted at CLK 1. The PCI arbiter grants access to Agent A by asserting GNT#-a on CLK 2. Agent A must begin a transaction by asserting FRAME# within 16 clocks, or the GXLV's PCI arbiter will remove GNT#. Also, it is possible for Agent A to lose bus ownership sooner if another agent with higher priority requests the bus. However, in this example, Agent B is of higher priority than Agent A. When Agent B requests the bus on CLK 2, Agent A is allowed to proceed per Specification. Agent A starts its transaction on CLK 3 by asserting FRAME# and completes its transaction. Since Agent A requests another transaction, REQ#-a remains asserted. When FRAME# is asserted on CLK 3, the PCI arbiter determines Agent B should go next, asserts GNT#-b and deasserts GNT#-a on CLK 4. Agent B requires only a single transaction. It completes the transaction, then deasserts FRAME# and REQ#-b on CLK 6. The PCI arbiter can then grant access to Agent A, and does so on CLK 7. Note that all buffers must flush before a grant is given to a new agent.

For additional information refer to Chapter 3.4.1, Arbitration Signaling Protocol, of the PCI Local Bus Specification, Revision 2.1.

#### 4.7.8.4 PCI Halt Command

Halt is a broadcast message from the GXLV processor indicating it has executed a HALT instruction. The PCI Special Cycle command is used to broadcast the message to all agents on the bus segment. During the address phase of the Halt Special cycle, C/BE[3:0]# = 0001 and AD[31:0] are driven to arbitrary values. During the data phase, C/BE[3:0]# = 1100 indicating bytes 1 and 0 are valid and AD[15:0] = 0001h.

For additional information, refer to Chapter 3.7.2, Special Cycle, and Appendix A, Special Cycle Messages, of the PCI Local Bus Specification, Revision 2.1.

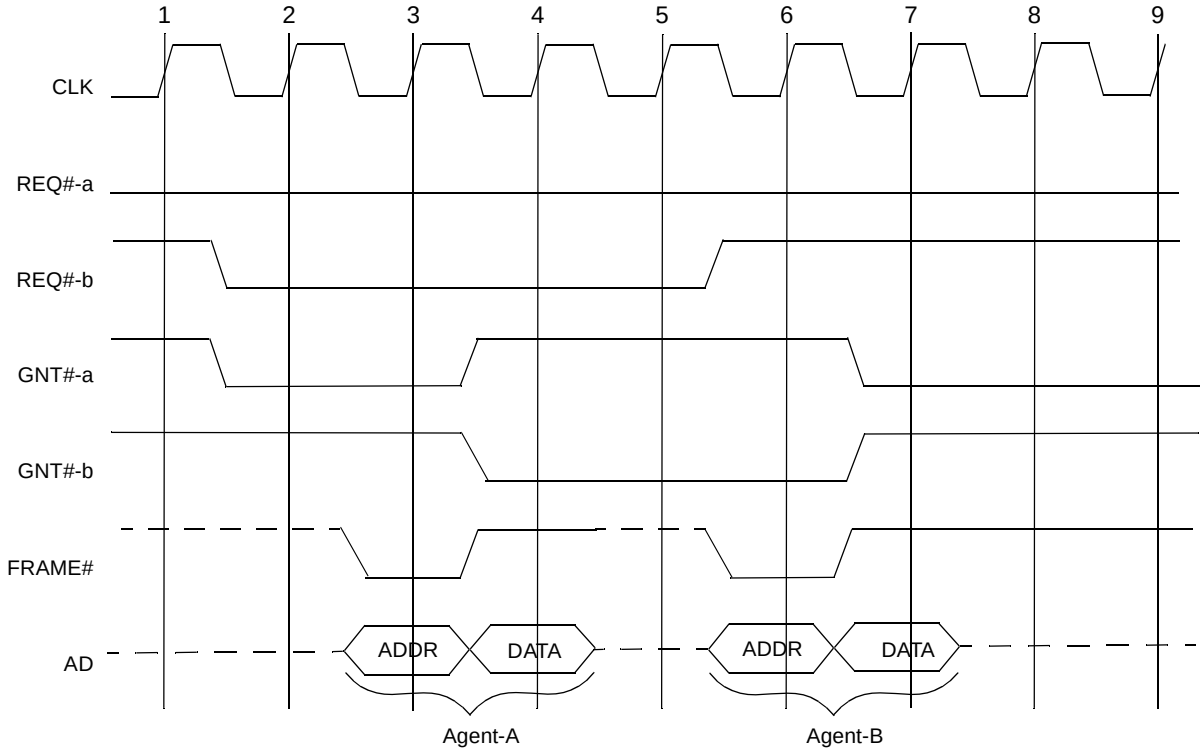



**Figure 4-20. Basic Arbitration**

# 5.0 Power Management

Power consumption in a GXLV processor based system is managed with the use of both of hardware and software. The complete hardware solution is provided for only when the GXLV processor is combined with a Geode I/O companion such as the CS5530.

The GXLV processor power consumption is managed primarily through a sophisticated clock stop management technology. The GXLV processor also provides the hardware enablers from which the complete power management solution depends on.

Typically the three greatest power consumers in a battery powered device are the display, the hard drive (if it has one) and the CPU. Managing power for the first two is relatively straightforward and is discussed in the CS5530 I/O companion data book. Managing CPU power is more difficult since effective use of the clock stop technology requires effective detection of inactivity, both at a system level and at a code processing level.

Basically two methods are supported to manage power during periods of inactivity. The first method, called activity based power management allows the hardware in the Geode I/O companion to monitor activity to certain devices in the system and if a period of inactivity occurs take some form of power conservation action. This method does not require OS support because this support is handled by SMM software. Simple monitoring of external activity is imperfect as well as inefficient. The second method, called passive power management, requires the OS to take the active role in managing power. National supports two application programming interfaces (APIs) to enable power management by the OS: Advanced Power Management (APM) and Advanced Configuration and Power Interface (ACPI). These two methods can be used independent of one another or they can be used together. The extent to which these resources are employed depends on the application and the discretion of the system designer.

The GXLV processor and Geode I/O companion chips contain advanced power management features for reducing the power consumption of the processor in the system.

## 5.1 POWER MANAGEMENT FEATURES

The GXLV processor based system supports the following power management features:

- GXLV processor hardware
  - System Management Mode (SMM)
  - Suspend-on-Halt
  - CPU Suspend
  - 3 Volt Suspend
  - GXLV Processor Serial Bus
- Geode I/O companion hardware:
  - I/O activity monitoring
    - SMI generation
  - CPU Suspend control
    - Suspend Modulation
    - 3 Volt Suspend
  - ACPI hardware
- Software:
  - API for APM aware OS
  - API for ACPI aware OS
  - PM VSA for not PM aware OS's

Geode I/O companion power management support is discussed in this specification only when necessary to better explain the GXLV processor's power management features.

Software support of power management is discussed in this specification only when necessary to better explain the GXLV processor's power management features.

### 5.1.1 System Management Mode

The GXLV processor has an operation mode called System Management Mode. This mode is generally entered when the SMI# pin goes active. SMM is explained in Section 3.7 "System Management Mode" on page 83. If active power management is desired, then the Geode I/O companion is programmed at boot time to activate SMM through the SMI# pin due to specific I/O inactivity.

SMM is also used in the passive power management method, however, it is limited to supporting specific API calls such as entering sleep modes.

### 5.1.2 Suspend-on-Halt

Suspend-on-Halt is the most effective power reducing feature of the GXLV processor with the system active. This feature allows the system to reduce power when the system's OS becomes idle without producing any delay when the system's OS becomes active.

When entered, Suspend-on-Halt stops the clock to the processor core while the intergrated functions (graphics, memory controller, PCI controller) are still active. There is absolutely no observational evidence that the processor has changed operational behavior except for two things. The GXLV draws significantly less core power and the SUSPA# pin is active while in this state.

### 5.1.3 CPU Suspend

CPU Suspend is a hardware initiated power management state. The SUSP# pin is asserted by external hardware such as an Geode I/O companion. The GXLV processor asserts the SUSPA# pin to indicate that the processor has entered CPU Suspend. This state is similar to Suspend-on-Halt except for its entry and exit method. SUSP# active causes the processor to enter the state and SUSP# inactive causes its exit. The power savings is identical to Suspend-on-Halt. Also, as in Suspend-on-Halt, the processor will temporally disable CPU Suspend when there is PCI master activity.

## Power Management (Continued)

CPU Suspend can be used for Suspend Modulation. The Geode I/O companion can be programmed to assert/deassert SUSP# at a programmable frequency and duty cycle. This has the effect of reducing the average frequency that the processor is running and thus reduces power consumption and performance. Certain processing activities (SMI#, Interrupts, and VGA activity) can be monitored by the Geode I/O companion to temporarily suspend, Suspend Modulation for a programmable amount of time. Suspend modulation programming is explained in detail in the Geode I/O companion data books such as the CS5530.

#### 5.1.3.1 Suspend Modulation for Thermal Management

The best use of Suspend Modulation is for thermal management. The Geode I/O companion monitors the temperature of the system and/or CPU and asserts the SMI# pin, if the system or CPU gets too hot. The power management SMM handler enables Suspend Modulation. When the temperature drops to a certain point the Geode I/O companion again asserts the SMI# pin. The power management SMM handler disables Suspend Modulation and normal operation resumes. A significant side effect of Suspend Modulation is a lowering of system performance while in this state. The system design must take this into account. If the system exceeds temperature limits only in extreme conditions then thermal management by use of Suspend Modulation can be easily and effectively used to reduce system cost by eliminating fans and possibility heatsinks. However, if maximum performance is required in all conditions then Suspend Modulation should not be used.

#### 5.1.3.2 Suspend Modulation for Power Management

Suspend modulation can also be used for a crude method of power management. The Geode I/O companion monitors I/O activity and when that monitoring indicates inactivity, the Geode /O companion asserts the SMI# pin. The power management SMM handler enables Suspend Modulation. When I/O activity picks up, the SMI# pin is asserted again and the power management SMM handler exits Suspend Modulation and normal operation resumes.

### 5.1.4 3 Volt Suspend

3 Volt Suspend is identical to CPU Suspend with the addition of setting CLK_STP in the PM_CNTRL_CSTP Register (Table 5-2 on page 181), and turning off the graphics pipeline (set GX_BASE+8304h[0] = 0) before the assertion of SUSP#. If CLK_STP is set and the graphics pipeline is still active then the SUSP# will be ignored and 3 Volt Suspend will not be entered. As 3 Volt Suspend is being entered, the memory controller  puts the SDRAMS in self refresh mode. At this point, all internal clocks in the GXLV processor are stopped. Once SUSPA# has gone active, SYSCLK input pin can be stopped. While in this state the GXLV processor will not respond to anything except the deassertion of SUSP# as long as SYSCLK has been restarted.

### 5.1.5 GXLV Processor Serial Bus

The power management logic of the GXLV processor provides the Geode I/O companion with information regarding the GXLV processor productivity. If the GXLV processor is determined to be relatively inactive, the GXLV processor power consumption can be greatly reduced by entering the Suspend Modulation mode.

Although the majority of the system power management logic is implemented in the Geode I/O companion, a small amount of logic is required within the GXLV processor to provide information from the graphics controller that is not externally visible otherwise. The GXLV processor implements a simple serial communications mechanism to transmit the CPU status to the Geode I/O companion. The GXLV processor accumulates CPU events in a 8-bit register, “PM Serial Packet Register” (GX_BASE+850Ch), that is serially transmitted out of the GXLV processor every 1 to 10 µs. The transmission frequency is set with bits [4:3] of the “PM Serial Packet Control Register”. These register formats are given in Table 5-2 starting on page 181.

### 5.1.6 Advanced Power Management (APM) Support

Many battery powered devices rely solely on the APM (Advanced Power Management) driver for DOS, Windows 95/98, and other operating systems to manage power to the CPU. APM provides several services that enhance the system power management by determining when the CPU is idle. For the CPU, APM is theoretically the best approach but there are some drawbacks.

- APM is an OS-specific driver which is not available for all operating systems.
- Application support is inconsistent. Some applications in foreground may prevent idle calls.

The components for APM support are:

- Software CPU Suspend control via the Geode I/O companion CPU Suspend Command Register.
- Software SMI entry via the Software SMI Register. This allows the APM BIOS to be part of the SMM handler.

## Power Management (Continued)

### 5.2 SUSPEND MODES AND BUS CYCLES

The following subsections describe the bus cycles of the various suspend states.

#### 5.2.1 Timing Diagram for Suspend-on-Halt

The CPU enters Suspend-on-Halt as a result of executing a halt (HLT) instruction if the SUSP_HALT bit in CCR2 (Index C2h[3]) is set. When the HLT instruction is executed, the halt PCI cycle is run on the PCI bus normally and then the SUSPA# pin will go active to indicate that the processor has entered the suspend state. This state is slightly is different from CPU Suspend because of how Suspend-on-Halt is entered and how it is exited. Suspend-on-Halt is exited upon recognition of an unmasked INTR or an SMI#. Normally SUSPA# is deactivated within six SYSCLKS from the detection of an active interrupt. However, the deactivation of SUSPA# may be delayed until the end of an active refresh cycle.

The CPU allows PCI master accesses during a HALT-initiated Suspend mode. The SUSPA# pin will go inactive during the duration of the PCI activity. If the CPU is in the middle of a PCI master access when the Halt instruction is executed, the assertion of SUSPA# will be delayed until the PCI access is completed. See Figure 5-1 for timing details.

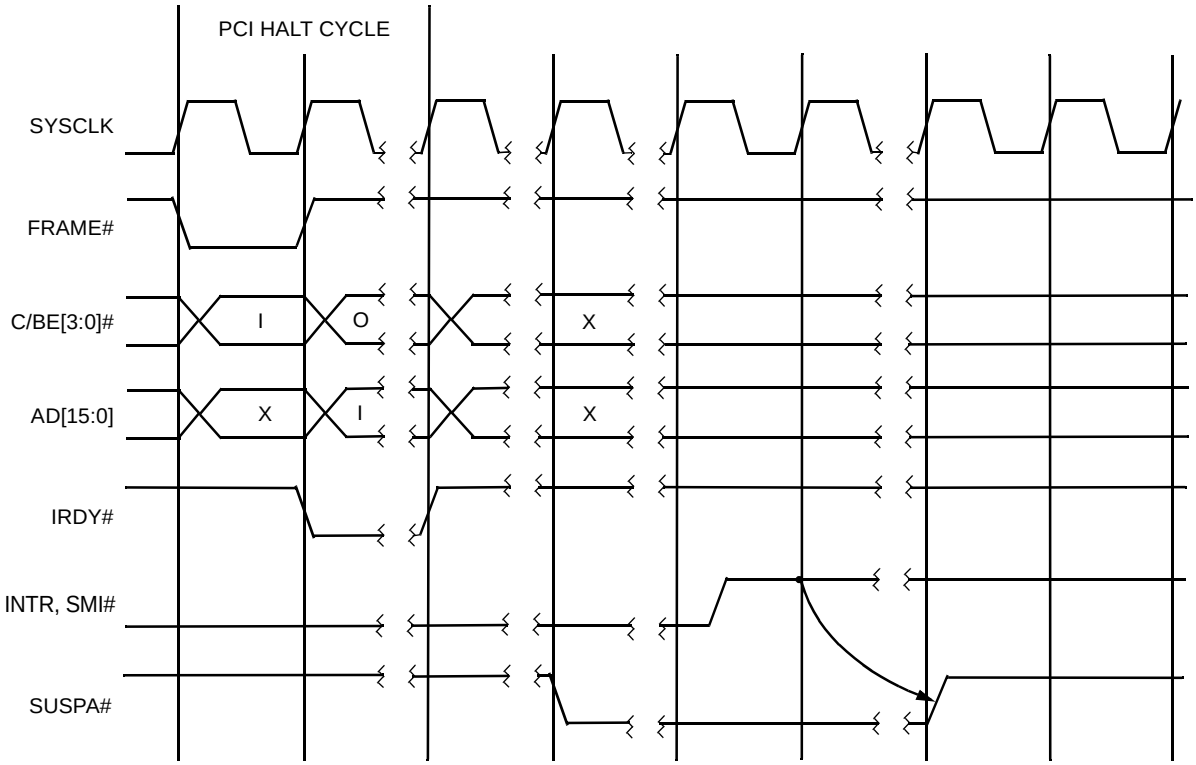



**Figure 5-1. HALT-Initiated Suspend Mode**

## Power Management (Continued)

### 5.2.2 Initiating Suspend with SUSP#

The GXLV processor enters the Suspend mode in response to SUSP# input assertion only when certain conditions are met. First, the USE_SUSP bit must be set in CCR2 (Index C2h[7]). In addition, execution of the current instructions and any pending decoded instructions and associated bus cycles must be completed. SUSP# is sampled on the rising edge of SYSCLK, and must meet specified setup and hold times to be recognized at a particular SYSCLK edge. See Figure 5-2 for timing details.

When all conditions are met, the SUSPA# output is asserted. The time from assertion of SUSP# to the activation of SUSPA# depends on which instructions were decoded prior to assertion of SUSP#. Normally, once SUSP# has been sampled inactive the SUSPA# output will be deactivated within two clocks. However, the deactivation of SUSPA# may be delayed until the end of an active refresh cycle.

If the CPU is already in a Suspend mode initiated by SUSP#, one occurrence of INTR and SMI# is stored for execution after Suspend mode is exited. The CPU also allows PCI master accesses during a SUSP#-initiated Suspend mode. See Figure 5-3 for timing details. If an unmasked REQx# is asserted, the GXLV processor will deassert SUSPA# and exit Suspend mode to respond to the PCI master access. If SUSP# is asserted when the PCI master access is completed, REQx# deasserted, the GXLV processor will reassert SUSPA# and return to a SUSP#-initiated Suspend mode. If the CPU is in the middle of a PCI master access when SUSP# is asserted, the assertion of SUSPA# will be delayed until the PCI access is completed.

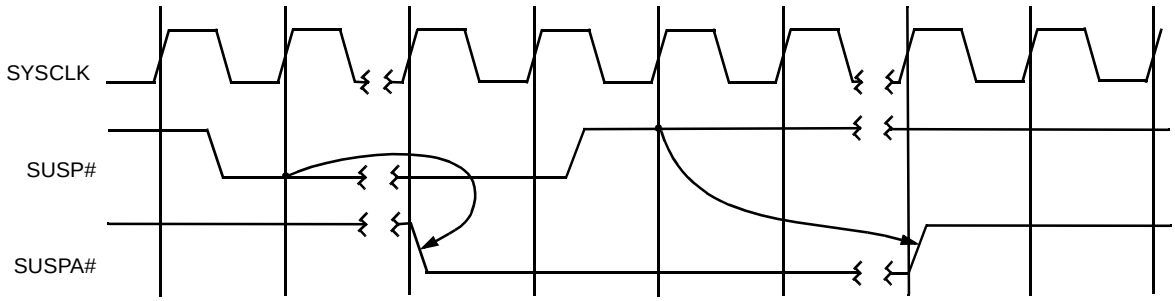



**Figure 5-2. SUSP#-Initiated Suspend Mode**

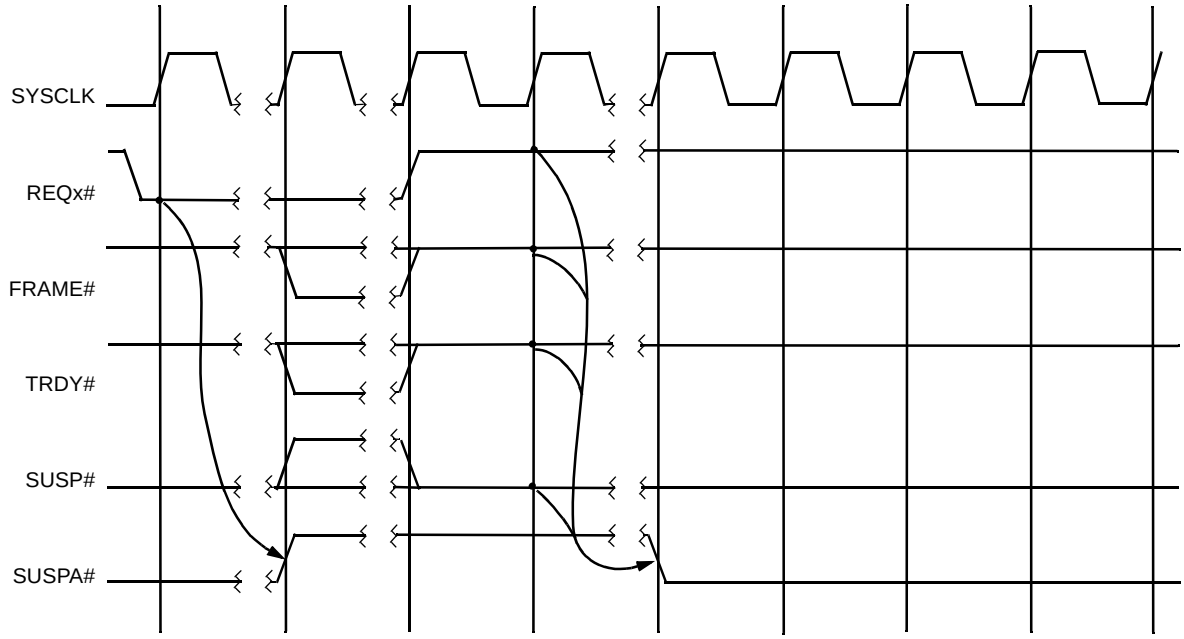



**Figure 5-3. PCI Access During Suspend Mode**

## Power Management (Continued)

### 5.2.3 Stopping the Input Clock

The GXLV processor is a static device, allowing the input clock (SYSCLK) to be stopped and restarted without any loss of internal CPU data. The SYSCLK input can be stopped at either a logic high or logic low state. The required sequence for stopping SYSCLK is to initiate 3 Volt Suspend, wait for the assertion of SUSPA# by the processor, and then stop the input clock.

The CPU remains suspended until SYSCLK is restarted and the Suspend mode is exited as described earlier. While SYSCLK is stopped, the processor can no longer sample and respond to any input stimulus including REQx#, NMI, SMI#, INTR, and RESET inputs.

Figure 5-4 illustrates the recommended sequence for stopping the SYSCLK using SUSP# to initiate 3 Volt Suspend. SYSCLK may be started prior to or following negation of the SUSP# input. The figure includes the SUSP_3V pin from the Geode I/O companion which is used to stop the external clocks.

### 5.2.4 Serial Packet Transmission

The GXLV processor transmits the contents of the “PM Serial Packet Register” on the SERIALP output pin to the PSERIAL input pin of the Geode I/O companion. The GXLV processor holds SERIALP low until the transmission interval counter (GX_BASE+8504h[4:3]) has elapsed. Once the counter has elapsed, PSERIAL is held high for two SYSCLKs to indicate the start of packet transmission.

The contents of the packet register are then shifted out starting from bit 7 down to bit 0. PSERIAL is held high for one SYSCLK to indicate the end of packet transmission and then remains low until the next transmission interval. After the packet transmission has completed, the packet contents are cleared.

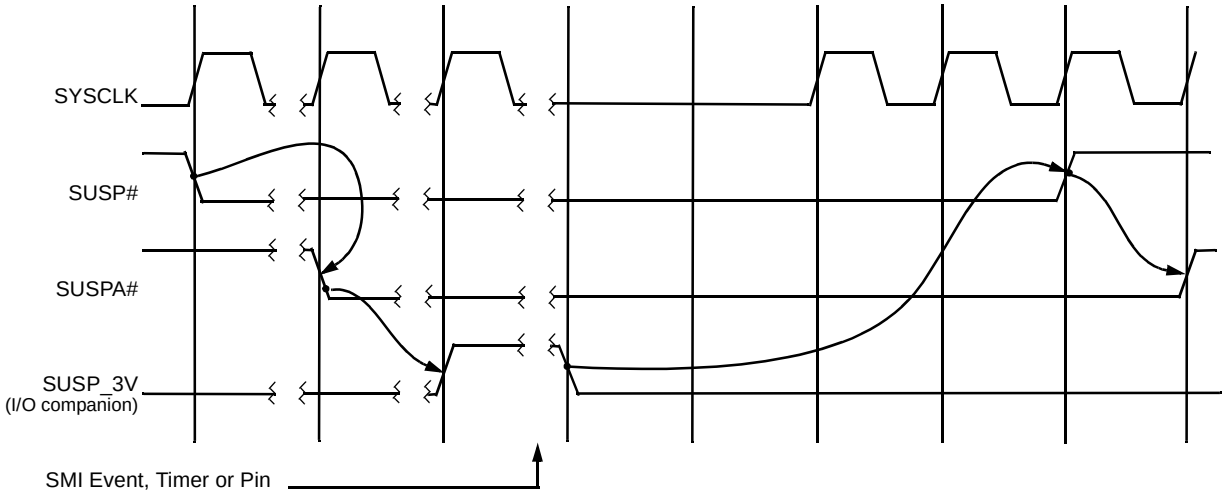



**Figure 5-4. Stopping SYSCLK During Suspend Mode**

## Power Management (Continued)

### 5.3 POWER MANAGEMENT REGISTERS

The GXLV processor contains the power management registers for the serial packet transmission control, the user-defined power management address space, Suspend Refresh, and SMI status for Suspend/Resume. These registers are memory mapped (GX_BASE+8500h-8FFFh) in the address space of the GXLV processor and are described in the following sections. Refer to Section 4.1.2 “Control Registers” on page 99 for instructions on accessing these registers.

Note, however, the PM_BASE and PM_MASK registers are accessed with the CPU_READ and CPU_WRITE instructions. Refer to Section 4.1.6 “CPU_READ/CPU_WRITE Instructions” on page 102 for more information regarding these instructions.

Table 5-1 summarizes the above mentioned registers. Tables 5-2 and 5-3 give these register’s bit formats.

**Table 5-1. Power Management Register Summary**

| GX_BASE+ Memory Offset | Type | Name/Function | Default Value |
|---|---|---|---|
| **Control and Status Registers** | | | |
| 8500h-8503h | R/W | **PM_STAT_SMI**<br>PM SMI Status Register: Contains System Management Mode (SMM) status information used by SoftVGA. | xxxxxx00h |
| 8504h-8507h | R/W | **PM_CNTRL_TEN**<br>PM Serial Packet Control Register: Sets the serial packet transmission frequency and enables specific CPU events to be recorded in the serial packet. | xxxxxx00h |
| 8508h-850Bh | R/W | **PM_CNTRL_CSTP**<br>PM Clock Stop Control Register: Enables the 3V Suspend Mode for the GXLV processor. | xxxxxx00h |
| 850Ch-850Fh | R/W | **PM_SER_PACK**<br>PM Serial Packet Register: Transmits the contents of the serial packet. | xxxxxx00h |
| **Programmable Address Region Registers** | | | |
| FFFFFF6Ch | R/W | **PM_BASE**<br>PM Base Register: Contains the base address for the programmable memory range decode. This register, in combination with the PM_MASK register, is used to generate a memory range decode which sets bit 1 in the serial transmission packet. | 00000000h |
| FFFFFF7Ch | R/W | **PM_MASK**<br>PM Mask Register: The address mask for the PM_BASE register | 00000000h |

**Table 5-2. Power Management Control and Status Registers**

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+8500h-8503h** | | **PM_STAT_SMI Register (R/W)** **Default Value = xxxxxx00h** |
| 31:8 | RSVD | **Reserved:** These bits are not used. Do not write to these bits. |
| 7:3 | RSVD | **Reserved:** Set to 0. |
| 2 | SMI_MEM | **SMI VGA Emulation Memory:** This bit is set high if a SMI was generated for VGA emulation in response to a VGA memory access. An SMI can be generated on a memory access to one of three regions in the A0000h to BFFFFh range as specified in the BC_XMAP_1 register. (See Table 4-9 on page 104) |
| 1 | SMI_IO | **SMI VGA Emulation I/O:** This bit is set high if a SMI was generated for VGA emulation in response to an I/O access. An SMI can be generated on a I/O access to one of three regions in the 3B0h to 3DFh range as specified in the BC_XMAP_1 register. (See Table 4-9 on page 104) |
| 0 | SMI_PIN | **SMI Pin:** When set high, this bit indicates that the SMI# input pin has been asserted to the GXLV processor. |
| **Note:** These bits are “sticky” bits and can only be cleared with a write of ‘1’ to the respective bit. | | |

## Power Management (Continued)

### Table 5-2. Power Management Control and Status Registers (Continued)

| Bit | Name | Description |
|---|---|---|
| **GX_BASE+8504h-8507h** | **PM_CNTRL_TEN Register (R/W)** | **Default Value = xxxxxx00h** |
| 31:8 | RSVD | **Reserved:** These bits are not used. Do not write to these bits. |
| 7:6 | RSVD | **Reserved:** Set to 0. |
| 5 | X_TEST (WO) | **Transmission Test (Write Only)** : Setting this bit causes the GXLV processor to immediately transmit the current contents of the serial packet. This bit is write only and is used primarily for test. This bit returns 0 on a read. |
| 4:3 | X_FREQ | **Transmission Frequency:** This field indicates the time between serial packet transmissions. Serial packet transmissions occur at the selected interval only if at least one of the packet bits is set high: 00 = Disable transmitter; 01 = 1 ms; 10 = 5 ms; 11 = 10 ms. |
| 2 | CPU_RD | **CPU Activity Read Enable:** Setting this bit high enables reporting of CPU level-1 cache read misses that are not a result of an instruction fetch. This bit is a don't-care if the CPU_EN bit is not set high. |
| 1 | CPU_EN | **CPU Activity Master Enable:** Setting this bit high enables reporting of CPU Level-1 cache misses in bit 6 of the serial transmission packet. When enabled, the CPU Level-1 cache miss activity is reported on any read (assuming the CPU_RD is set high) or write access excluding misses that resulted from an instruction fetch. |
| 0 | VID_EN | **Video Event Enable:** Setting this bit high enables video decode events to be reported in bit 0 of the serial transmission packet. CPU or graphics-pipeline accesses to the graphics memory and display-controller-register accesses are also reported. |
| **GX_BASE+8508h-850Bh** | **PM_CNTRL_CSTP Register (R/W)** | **Default Value = xxxxxx00h** |
| 31:8 | RSVD | **Reserved:** These bits are not used. Do not write to these bits. |
| 7:1 | RSVD | **Reserved:** Set to 0. |
| 0 | CLK_STP | **Clock Stop:** This bit configures the GXLV processor for Suspend Refresh Mode or 3 Volt Suspend Mode:<br>0 = Suspend Refresh Mode. The clocks to the memory and display controller remain active during Suspend.<br>1 = 3 Volt Suspend Mode. The external clock may be stopped during Suspend. |
| **Note:** When bit 0 is set high and the Suspend input pin (SUSP#) is asserted, the GXLV processor stops all it's internal clocks, and asserts the Suspend Acknowledge output pin (SUSPA#). Once SUSPA# is asserted the GXLV processor's SYSCLK input can be stopped. If bit 0 is cleared, the internal memory-controller and display-controller clocks are not stopped on the SUSP#/SUSPA# sequence, and the SYSCLK input can not be stopped. | | |
| **GX_BASE+850Ch-850Fh** | **PM_SER_PACK Register (R/O)** | **Default Value = xxxxxx00h** |
| 31:8 | RSVD | **Reserved:** These bits are not used. Do not write to these bits. |
| 7 | VID_IRQ | **Video IRQ:** This bit indicates the occurrence of a video vertical sync pulse. This bit is set at the same time that the VINT (Vertical Interrupt) bit is set in the DC_TIMING_CFG register. The VINT bit has a corresponding enable bit (VIEN) in the DC_TIM_CFG register (See Table 4-29 on page 145). |
| 6 | CPU_ACT | **CPU Activity:** This bit indicates the occurrence of a level 1 cache miss that was not a result of an instruction fetch. This bit has a corresponding enable bit in the PM_CNTL_TEN register. |
| 5:2 | RSVD | **Reserved:** Set to 0. |
| 1 | USR_DEF | **Programmable Address Decode:** This bit indicates the occurrence of a programmable memory address decode. This bit is set based on the values of the PM_BASE register and the PM_MASK register (see Table 5-3 on page 183). The PM_BASE register can be initialized to any address in the full 256 MB address range. |
| 0 | VID_DEC | **Video Decode:** This bit indicates that the CPU has accessed either the Display Controller registers or the graphics memory region. This bit has a corresponding enable bit in the PM_CNTRL_TEN. |
| **Note:** The GXLV processor transmits the contents of the serial packet only when a bit in the packet register is set and the interval counter has elapsed. The Geode I/O companion decodes the serial packet after each transmission. Once a bit in the packet is set, it will remain set until the completion of the next packet transmission. Successive events of the same type that occur between packet transmissions are ignored. Multiple unique events between packet transmissions will accumulate in this register. | | |

**Table 5-3. Power Management Programmable Address Region Registers**

| Bit | Name | Description |
|---|---|---|
| **Index FFFFFF6Ch** | | **PM_BASE Register (R/W)** **Default Value = 0000000h** |
| 31:28 | RSVD | **Reserved:** Set to 0. |
| 27:2 | BASE_ADDR | **Base Address:** This is the word-aligned base address for the programmable memory range compare. The actual address range is determined with this field and the PM_MASK register value. |
| 1:0 | RSVD | **Reserved:** Set to 0. |
| **Index FFFFFF7Ch** | | **PM_MASK Register (R/W)** **Default Value = 0000000h** |
| 31:28 | RSVD | **Reserved:** Set to 0. |
| 27:2 | ADR_MASK | **Address Mask:** This field is the address mask for the BASE_ADDR field in the PM_BASE register. If a bit in the ADR_MASK field is cleared the corresponding bit in the BASE_ADDR field must match the processor address. If a bit in the mask field is set high, the corresponding bit in the BASE_ADDR field always compares. If the processor cycle type matches the values of the WE and RE bits, and all bits in the BASE_ADDR field match the processor address based on the ADR_MASK field, bit 1 will be set high in the serial transmission packet. |
| 1 | WE | **Write Enable:** Compare memory write cycles with BASE_ADDR and ADR_MASK:<br>0 = Disable; 1 = Enable. |
| 0 | RE | **Read Enable:** Compare memory read cycles with BASE_ADDR and ADR_MASK:<br>0 = Disable; 1 = Enable |

# 6.0 Electrical Specifications

This section provides information on electrical connections, absolute maximum ratings, recommended operating conditions, DC characteristics, and AC characteristics for the Geode GXLV processor series. All voltage values in the electrical specifications are with respect to $V_{SS}$ unless otherwise noted. For detailed information on the PCI bus electrical specification refer to Chapter 4 of the PCI Bus Specification, Revision 2.1.

## 6.1 PART NUMBERS/PERFORMANCE CHARACTERISTICS

The GXLV series of processors is designated by three core voltage specifications: 2.9V, 2.5V, and 2.2V. Each core voltage is offered in frequencies that are enabled by specific system clock and internal multiplier settings. This allows the user to select the device(s) that best fit their power and performance requirements. This flexibility makes the GXLV processor series ideally suited for applications where power consumption and performance (speed) are equally important.

The part numbers in Table 6-1 designate the various combinations of speed and power consumption available. Note that while there are three $V_{CC2}$ (Core) voltages available, the $V_{CC3}$ (I/O) voltage remains constant at 3.3V (nominal) in order to maintain LVTTL compatibility with external devices.

**Table 6-1. Performance Characteristics**

| Part Marking | Core Voltage ($V_{CC2}$) | System Clock | Frequency Multiplier | Core Frequency | Maximum Power | Typical Power (Note) 80% Active Idle |
|---|---|---|---|---|---|---|
| GXLV-266P 2.9V 70C | 2.9V (Nominal) | 33 MHz | x8 | 266 MHz | 7.8W | 2.50W |
| GXLV-266P 2.9V 85C | | | | | | |
| GXLV-266B 2.9V 70C | | | | | | |
| GXLV-266B 2.9V 85C | | | | | | |
| GXLV-233P 2.5V 85C | 2.5V (Nominal) | 33 MHz | x7 | 233 MHz | 5.6W | 2.0W |
| GXLV-233B 2.5V 85C | | | | | | |
| GXLV-200P 2.2V 85C | 2.2V (Nominal) | 33 MHz | x6 | 200 MHz | 4.1W | 1.5W |
| GXLV-200B 2.2V 85C | | | | | | |
| GXLV-180P 2.2V 85C | | 30 MHz | x6 | 180 MHz | 3.9W | 1.25W |
| GXLV-180B 2.2V 85C | | | | | | |
| GXLV-166P 2.2V 85C | | 33 MHz | x5 | 166 MHz | 3.7W | 1.0W |
| GXLV-166B 2.2V 85C | | | | | | |

**Note:** Typical power consumption is defined as an average measured running Windows at 80% Active Idle (Suspend-on-Halt) with a display resolution of 800x600x8 bpp at 75 Hz.

# Electrical Specifications (Continued)

## 6.2 ELECTRICAL CONNECTIONS

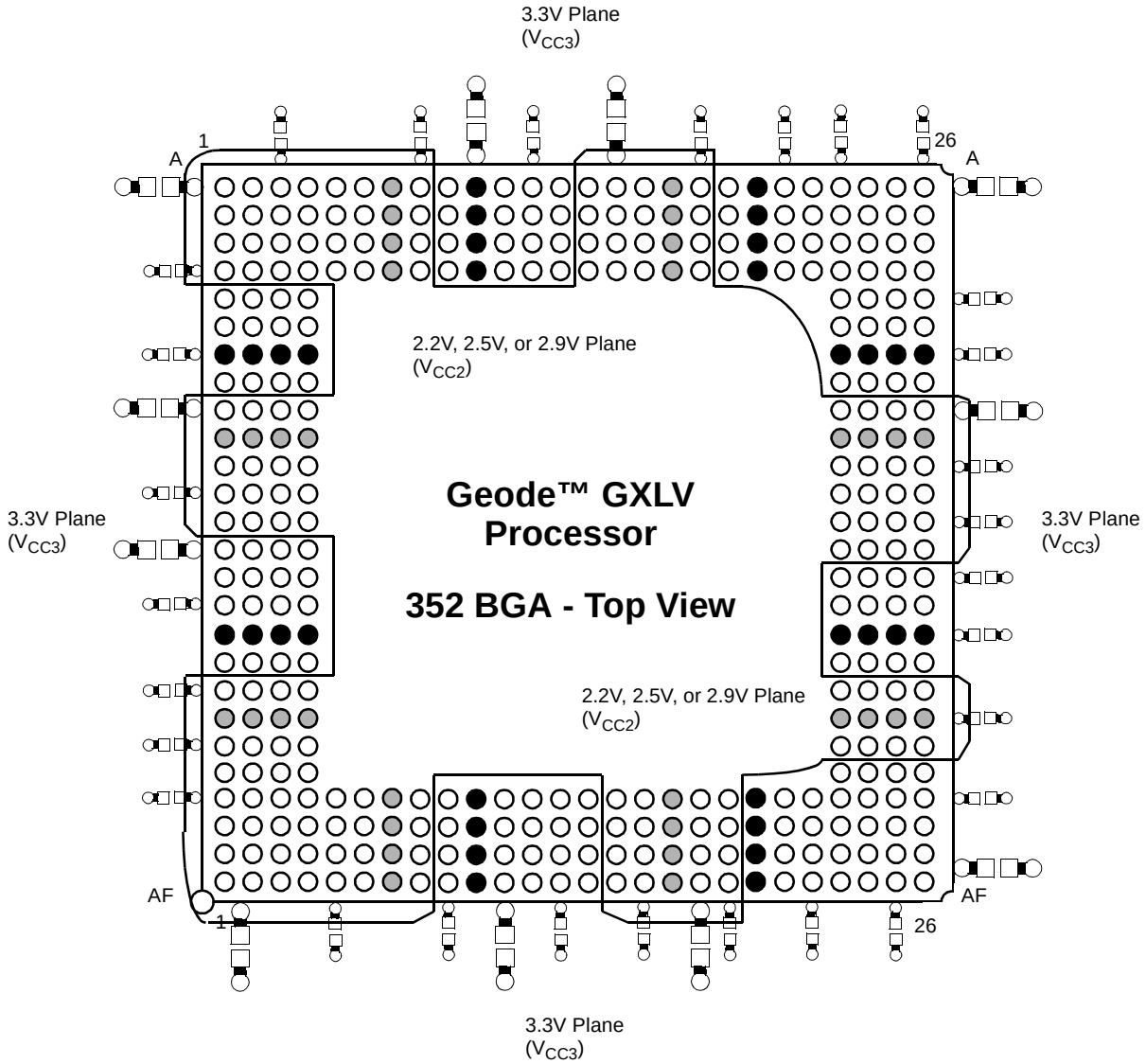
### 6.2.1 Power/Ground Connections and Decoupling

Testing and operating the GXLV processor requires the use of standard high frequency techniques to reduce parasitic effects. These effects can be minimized by filtering the DC power leads with low-inductance decoupling capacitors, using low-impedance wiring, and by connecting all $V_{CC2}$ and $V_{CC3}$ pins to the appropriate voltage levels.

#### 6.2.1.1 Power Planes

Figure 6-1 shows layout recommendations for splitting the power plane between $V_{CC2}$ (core: 2.2V, 2.5V, 2.9V) and $V_{CC3}$ (I/O: 3.3V) volts in the BGA package. The illustration assumes there is one power plane, and no components on the back of the board.

Figure 6-2 shows layout recommendations for splitting the power plane between $V_{CC2}$ (core: 2.2V, 2.5V, 2.9V) and $V_{CC3}$ (I/O: 3.3V) volts in the SPGA package.




**Legend**

= High frequency capacitor

= 220 µF, low ESR capacitor

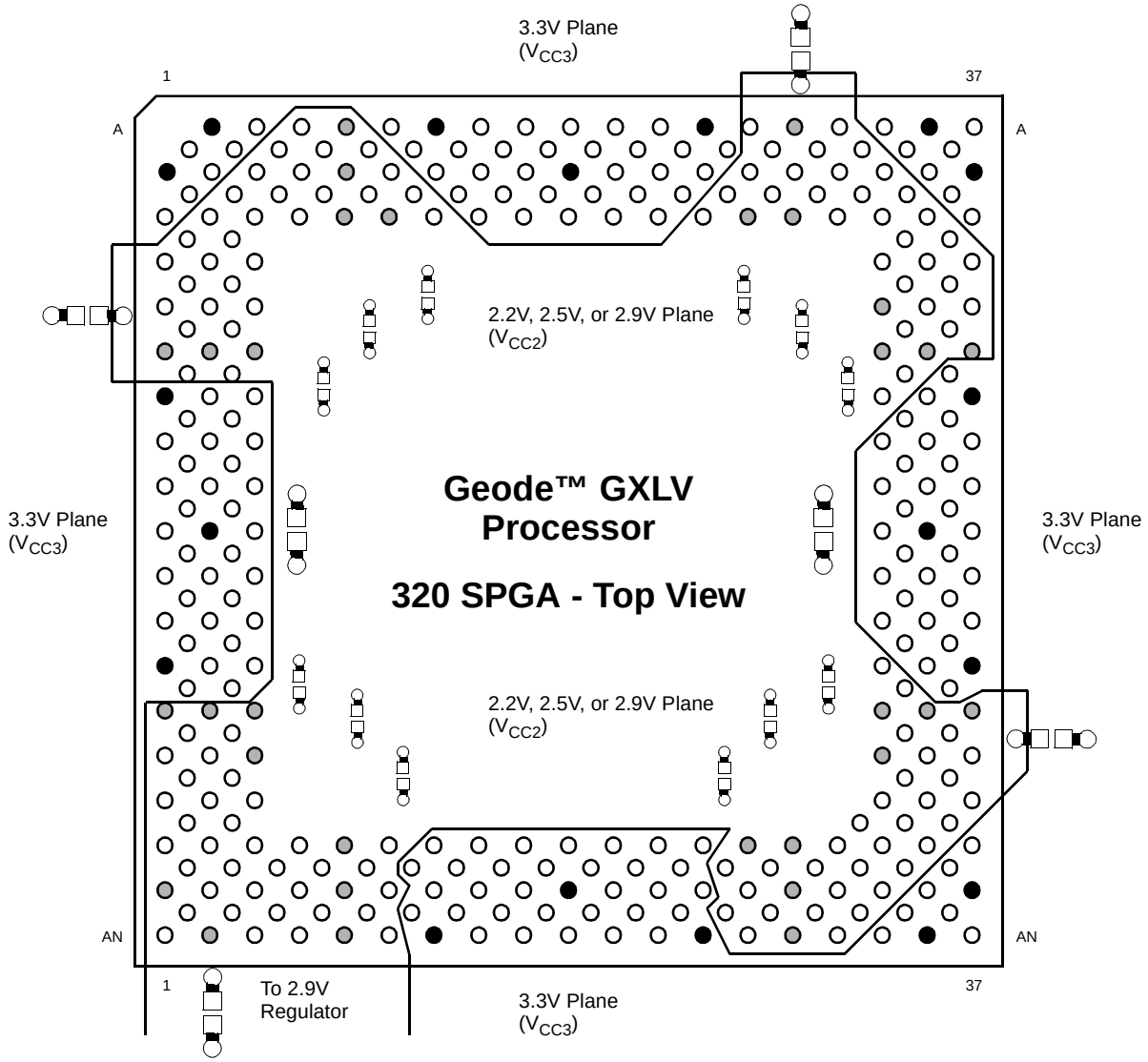
= 3.3V connection

= 2.2V, 2.5V, or 2.9V connection

**Note:** Where signals cross plane splits, it is recommended to include AC decoupling between planes with 47 pF capacitors.

**Figure 6-1. BGA Recommended Split Power Plane and Decoupling**

## Electrical Specifications (Continued)




**Legend**

= High frequency capacitor

= 220 µF, low ESR capacitor

= 3.3V connection

= 2.2V, 2.5V, or 2.9V connection

**Note:** Where signals cross plane splits, it is recommended to include AC decoupling between planes with 47 pF capacitors.

**Figure 6-2. SPGA Recommended Split Power Plane and Decoupling**

## Electrical Specifications (Continued)

### 6.2.2 NC-Designated Pins

Pins designated NC (No Connection) should be left disconnected. Connecting an NC pin to a pull-up/-down resistor, or an active signal could cause unexpected results and possible circuit malfunctions.

### 6.2.3 Pull-Up and Pull-Down Resistors

Table 6-2 lists the input pins that are internally connected to a weak (>20-kohm) pull-up/-down resistor. When unused, these inputs do not require connection to an external pull-up/-down resistor.

### 6.2.4 Unused Input Pins

All inputs not used by the system designer and not listed in Table 6-2 should be kept at either ground or $V_{CC3}$. To prevent possible spurious operation, connect active-high inputs to ground through a 20-kohm (±10%) pull-down resistor and active-low inputs to $V_{CC3}$ through a 20-kohm (±10%) pull-up resistor.

**Table 6-2. Pins with > 20-kohm Internal Resistor**

| Signal Name | BGA Ball No. | PU/PD |
|---|---|---|
| SUSP# | H2 | Pull-up |
| FRAME# | A8 | Pull-up |
| IRDY# | C9 | Pull-up |
| TRDY# | B9 | Pull-up |
| STOP# | C11 | Pull-up |
| LOCK# | B11 | Pull-up |
| DEVSEL# | A9 | Pull-up |
| PERR# | A11 | Pull-up |
| SERR# | C12 | Pull-up |
| REQ[2:0]# | D3, H3, E3 | Pull-up |
| TCLK | J2 | Pull-up |
| TMS | H1 | Pull-up |
| TDI | D2 | Pull-up |
| TEST | F3 | Pull-down |

# Electrical Specifications (Continued)

## 6.3 ABSOLUTE MAXIMUM RATINGS

Table 6-3 lists absolute maximum ratings for the GXLV processor. Stresses beyond the listed ratings may cause permanent damage to the device. Exposure to conditions beyond these limits may (1) reduce device reliability and (2) result in premature failure even when there is no immediately apparent sign of failure. Prolonged exposure to conditions at or near the absolute maximum ratings may also result in reduced useful life and reliability. These are stress ratings only and do not imply that operation under any conditions other than those listed under Table 6-4 on page 189 is possible.

**Table 6-3. Absolute Maximum Ratings**

| Symbol | Parameter | Min | Max | Units | Comments |
|---|---|---|---|---|---|
| $T_{CASE}$ | Operating Case Temperature | –65 | 110 | °C | Power Applied |
| $T_{STORAGE}$ | Storage Temperature | –65 | 150 | °C | No Bias |
| $V_{CC2}$ | Core Supply Voltage | | | | |
| | 2.2V (Nominal) | | 2.9 | V | |
| | 2.5V (Nominal) | | 3.2 | V | |
| | 2.9V (Nominal) | | 3.2 | V | |
| $V_{MAX}$ | Voltage On Any Pin | –0.5 | 6.0 | V | |
| $I_{IK}$ | Input Clamp Current | –0.5 | 10 | mA | Power Applied |
| $I_{OK}$ | Output Clamp Current | | 25 | mA | Power Applied |

# Electrical Specifications (Continued)

## 6.4 RECOMMENDED OPERATING CONDITIONS

Table 6-4 lists the operating conditions for the GXLV processor.

**Table 6-4. Operating Conditions**

| Symbol | Parameter | Min | Max | Units | Comments |
|---|---|---|---|---|---|
| $T_C$ | Operating Case Temperature | 0 | 85 | °C | |
| $V_{CC2}$ | Core Supply Voltage | | | | |
| | 2.2V (Nominal) | 2.09 | 2.31 | V | Note 1 |
| | 2.5V (Nominal) | 2.37 | 2.63 | V | |
| | 2.9V (Nominal) | 2.76 | 3.05 | V | |
| $V_{CC3}$ | Supply Voltage (3.3V Nominal) | 3.14 | 3.46 | V | Note 1 |
| $V_{IH}$ | Input High Voltage | | | | |
| | All except PCI bus and SYSCLK | 2.0 | $V_{CC3}$+0.5 | V | Note 3 |
| | PCI bus | 0.5 x $V_{CC3}$ | 5.5 | V | Note 2 |
| | SYSCLK | 2.7 | $V_{CC3}$+0.5 | V | Note 3 |
| $V_{IL}$ | Input Low Voltage | | | | |
| | All except PCI bus and SYSCLK | –0.5 | 0.8 | V | |
| | PCI bus | –0.5 | 0.3+$V_{CC3}$ | V | |
| | SYSCLK | –0.5 | 0.4 | V | |
| $I_{OH}$ | Output High Current | | –2 | mA | $V_O = V_{OH}$ (Min) |
| $I_{OL}$ | Output Low Current | | 5 | mA | $V_O = V_{OL}$ (Max) |

**Notes:** 1. This parameter is calculated as nominal ±5%.

2. Pin is tolerant to the PCI 5 Volt Signaling Environment DC specification.

3. Pin is not tolerant to the PCI 5 Volt Signaling Environment DC specification.

# Electrical Specifications (Continued)

## 6.5 DC CHARACTERISTICS

All DC parameters and current measurements in this section were measured under the operating conditions listed in Table 6-4 on page 189.

### 6.5.1 Input/Output DC Characteristics

Table 6-5 shows the input/output DC parameters for all the devices in the GXLV processor series.

**Table 6-5. DC Characteristics**

| Symbol | Parameter | Min | Typ | Max | Units | Comments |
|---|---|---|---|---|---|---|
| $V_{OL}$ | Output Low Voltage | | | 0.4 | V | $I_{OL}$ = 5 mA |
| $V_{OH}$ | Output High Voltage | 2.4 | | | V | $I_{OH}$ = –2 mA |
| $I_I$ | Input Leakage Current for all input pins except those with internal pull up/pull downs (PU/PDs). | | | ±10 | µA | $0 < V_{IN} < V_{CC3}$, See Table 6-2 |
| $I_{IH}$ | Input Leakage Current for all pins with internal PDs. | | | 200 | µA | $V_{IH}$ = 2.4 V, See Table 6-2 |
| $I_{IL}$ | Input Leakage Current for all pins with internal PUs. | | | –400 | µA | $V_{IL}$ = 0.35 V, See Table 6-2 |
| $C_{IN}$ | Input Capacitance | | | 16 | pF | f = 1 MHz, Note |
| $C_{OUT}$ | Output or I/O Capacitance | | | 16 | pF | f = 1 MHz, Note |
| $C_{CLK}$ | CLK Capacitance | | | 12 | pF | f = 1 MHz, Note |
| **Note:** Not 100% tested. | | | | | | |

### 6.5.2 DC Current

DC current is not a simple measurement. The CPU has four power states and two functional characteristics that determine how much current the processor uses at any given point in time.

#### 6.5.2.1 Definition of CPU Power States

The following DC characteristic tables list CPU core and I/O current for four distinct CPU power states:

- **On**: All internal and external clocks with respect to the processor are running and all functional blocks inside the processor (CPU core, memory controller, display controller, etc.) are actively generating cycles. This is equivalent to the ACPI specification's"S0" state.
- **Active Idle**: The CPU core has been halted, all other functional blocks (including the display controller for refreshing the display) are actively generating cycles. This state is entered when a HLT instruction is executed by the CPU core or the SUSP# pin is asserted. From a user's perspective, this state is indistinquishable from the "On" state and is equivalent to the ACPI specification's"S1" state.
- **Standby**: The CPU core has been halted and all internal clocks have been shut down. Externally, the SYSCLK input continues to be driven. This is equivalent to the ACPI specification's"S2" or "S3" state.
- **Sleep**: Very similar to "Standby" except that the SYSCLK input has been shut down as well. This is the lowest power state the processor can be in with voltage still applied to the device's core and I/O supply pins. This is equivalent to the ACPI specification's "S4BIOS" state.

#### 6.5.2.2 Definition and Measurement Techniques of CPU Current Parameters

The following two parameters indicate processor current while in the "On" state:

- **Typical Average**: Indicates the average current used by the processor while in the "On" state. This is measured by running typical Windows applications in a typical display mode. In this case, 800x600x8 bpp at 75 Hz, 50 MHz DCLK using a background image of vertical stripes (4-pixel wide) alternating between black and white with power management disabled (to guarantee that the processor never goes into the Active Idle state). This number is provided for reference only since it can vary greatly depending on the usage model of the system.

**Note:** This typical average should not be confused with the typical power numbers shown in Table 6-1 on page 184. The numbers in Table 6-1 are based on a combination of "On (Typical Average)" and "Active Idle" states.

- **Absolute Maximum**: Indicates the maximum instantaneous current used by the processor. CPU core current is measured by running the Landmark Speed 200™ benchmark test (with power management disabled) and measuring the peak current at any given instant during the test. I/O current is measured by running Microsoft Windows 98 and using a background image of vertical stripes (1-pixel wide) alternating between black and white at the maximum display resolution of 1280x1024x8 bpp at 75 Hz, 135 MHz DCLK.

## Electrical Specifications (Continued)

#### 6.5.2.3 Definition of System Conditions for Measuring "On" Parameters

Processor current is highly dependent two functional characteristics, DCLK (DOT clock) and SDRAM frequency. Table 6-6 shows how these factors are controlled when measuring the typical average and absolute maximum processor current parameters.

**Table 6-6. System Conditions Used to Determine CPU's Current Used During the "On" State**

| CPU Current Measurement | System Conditions | | | | Comments |
|---|---|---|---|---|---|
| | $V_{CC2}$ | $V_{CC3}$ | DCLK Freq | SDRAM Freq | |
| Typical Average | Nominal | Nominal | 50 MHz | Nominal Note 2 | Notes 1 and 4 |
| Absolute Maximum | Max | Max | 135 MHz | Max Note 5 | Notes 3, 4, 5, 7 |

**Notes:**

1. A DCLK frequency of 50 MHz is derived by setting the display mode to 800x600x8 bpp at 75 Hz, using a display image of vertical stripes (4-pixel wide) alternating between black and white with power management disabled.
2. SDRAM nominal frequency represents a single value that the memory controller can be configured for, between 66 MHz and 78 MHz, based on a given core clock frequency:

   166 MHz (5x) / 2.5 = 66.67 MHz
   180 MHz (6x) / 2.5 = 72.0 MHz
   200 MHz (6x) / 3.0 = 66.67 MHz
   233 MHz (7x) / 3.0 = 77.78 MHz
   266 MHz (8x) / 3.5 = 76.19 MHz
3. A DCLK frequency of 135 MHz is derived by setting the display mode to 1280x1024x8 bpp at 75 Hz, using a display image of vertical stripes (1-pixel wide) alternating between black and white with power management disabled.
4. See Table 6-4 on page 189 for nominal and maximum voltages.
5. SDRAM max frequency represents the highest frequency that the memory controller can be configured, up to 100 MHz, based on a given core clock frequency:

   166 MHz (5x) / 2.0 = 83.3 MHz
   180 MHz (6x) / 2.0 = 90.0 MHz
   200 MHz (6x) / 2.0 = 100.0 MHz
   233 MHz (7x) / 2.5 = 93.3 MHz
   266 MHz (8x) / 3.0 = 88.9 MHz
6. SDRAM speeds between 79 MHz and 100 MHz are only supported for particular types of closed system designs. Therefore, absolute maximum current will not be realized in most system designs. Refer to the de-rating curve in Figure 6-3 on page 195 to calculate absolute maximum current based on the system's parameters.
7. Not all system designs will support display modes that require a DCLK of 135 MHz. Therefore, absolute maximum current will not be realized in all system designs. Refer to the de-rating curve in Figure 6-3 on page 195 to calculate absolute maximum current based on the system's parameters.

## Electrical Specifications (Continued)

### 6.5.2.4 DC Current Measurements

The following tables show the DC current measurements for the 2.2V (Tables 6-7 and 6-8), 2.5V (Tables 6-9 and 6-10), and 2.9V (Tables 6-11 and 6-12) devices of the GXLV processor series.

**Table 6-7. 2.2V DC Characteristics for CPU Mode = “On”**

| Symbol | Parameter | Typ Avg | Abs Max | Units | Comments |
|---|---|---|---|---|---|
| $I_{CC3ON}$ | I/O Current @ $V_{CC3}$ = 3.3V (Nominal); CPU mode = "On" | | | | |
| | $I_{CC3}$ at $f_{CLK}$ = 166 MHz | 135 | 400 | mA | $I_{CC}$ for $V_{CC3}$, Note |
| | $I_{CC3}$ at $f_{CLK}$ = 180 MHz | 140 | 410 | | |
| | $I_{CC3}$ at $f_{CLK}$ = 200 MHz | 140 | 420 | | |
| $I_{CC2ON}$ | Core Current @ $V_{CC2}$ = 2.2V (Nominal); CPU mode = "On" | | | | |
| | $I_{CC2}$ at $f_{CLK}$ = 166 MHz | 775 | 1010 | mA | $I_{CC}$ for $V_{CC2}$, Note |
| | $I_{CC2}$ at $f_{CLK}$ = 180 MHz | 820 | 1060 | | |
| | $I_{CC2}$ at $f_{CLK}$ = 200 MHz | 900 | 1160 | | |
| **Note:** $f_{CLK}$ ratings refer to internal clock frequency. | | | | | |

**Table 6-8. 2.2V DC Characteristics for CPU Mode = “Active Idle”, “Standby”, and “Sleep”**

| Symbol | Parameter | Min | Typ | Max | Units | Comments |
|---|---|---|---|---|---|---|
| $I_{CC3IDLE}$ | I/O Current @ $V_{CC3}$ = 3.3V (Nominal); CPU mode = "Active Idle" | | | | | |
| | $I_{CC3IDLE}$ at $f_{CLK}$ = 166 MHz | | 130 | | mA | $I_{CC}$ for $V_{CC3}$, Note 1 |
| | $I_{CC3IDLE}$ at $f_{CLK}$ = 180 MHz | | 135 | | | |
| | $I_{CC3IDLE}$ at $f_{CLK}$ = 200 MHz | | 135 | | | |
| $I_{CC3STBY}$ | I/O Current @ $V_{CC3}$ = 3.3V (Nominal); CPU mode = "Standby" | | | 7 | mA | $I_{CC}$ for $V_{CC3}$, Note 2 |
| $I_{CC3SLP}$ | I/O Current @ $V_{CC3}$ = 3.3V (Nominal); CPU mode = "Sleep" | | | 3 | mA | $I_{CC}$ for $V_{CC3}$, Note 3 |
| $I_{CC2IDLE}$ | Core Current @ $V_{CC2}$ = 2.2V (Nominal); CPU mode = "Active Idle" | | | | | |
| | $I_{CC2IDLE}$ at $f_{CLK}$ = 166 MHz | | 175 | | mA | $I_{CC}$ for $V_{CC2}$, Note 1 |
| | $I_{CC2IDLE}$ at $f_{CLK}$ = 180 MHz | | 185 | | | |
| | $I_{CC2IDLE}$ at $f_{CLK}$ = 200 MHz | | 200 | | | |
| $I_{CC2STBY}$ | Core Current @ $V_{CC2}$ = 2.2V (Nominal); CPU mode = "Standby" | | | 16 | mA | $I_{CC}$ for $V_{CC2}$, Note 2 |
| $I_{CC2SLP}$ | Core Current @ $V_{CC2}$ = 2.2V (Nominal); CPU mode = "Sleep" | | | 6 | mA | $I_{CC}$ for $V_{CC2}$, Note 3 |

**Notes:**
1. $f_{CLK}$ ratings refer to internal clock frequency.
2. All inputs are at 0.2V or $V_{CC3}$ – 0.2 (CMOS levels). All inputs except clock are held static and all outputs are unloaded (static $I_{OUT}$ = 0 mA).
3. All inputs are at 0.2V or $V_{CC3}$ – 0.2 (CMOS levels). All inputs are held static and all outputs are unloaded (static $I_{OUT}$ = 0 mA).

## Electrical Specifications (Continued)

**Table 6-9. 2.5V DC Characteristics for CPU Mode = “On”**

| Symbol | Parameter | Typ Avg | Abs Max | Units | Comments |
|---|---|---|---|---|---|
| $I_{CC3ON}$ | I/O Current @ $V_{CC3}$ = 3.3V (Nominal); CPU mode = "On" $I_{CC3}$ at $f_{CLK}$ = 233 MHz | 160 | 420 | mA | $I_{CC}$ for $V_{CC3}$, Note |
| $I_{CC2ON}$ | Core Current @ $V_{CC2}$ = 2.5V (Nominal); CPU mode = "On" $I_{CC2}$ at $f_{CLK}$ = 233 MHz | 1200 | 1560 | mA | $I_{CC}$ for $V_{CC2}$, Note |

**Note:** $f_{CLK}$ ratings refer to internal clock frequency.

**Table 6-10. 2.5V DC Characteristics for CPU Mode = “Active Idle”, “Standby”, and “Sleep”**

| Symbol | Parameter | Min | Typ | Max | Units | Comments |
|---|---|---|---|---|---|---|
| $I_{CC3IDLE}$ | I/O Current @ $V_{CC3}$ = 3.3V (Nominal); CPU mode = "Active Idle" $I_{CC3IDLE}$ at $f_{CLK}$ = 233 MHz | | 150 | | mA | $I_{CC}$ for $V_{CC3}$, Note 1 |
| $I_{CC3STBY}$ | I/O Current @ $V_{CC3}$ = 3.3V (Nominal); CPU mode = "Standby" | | | 8 | mA | $I_{CC}$ for $V_{CC3}$, Note 2 |
| $I_{CC3SLP}$ | I/O Current @ $V_{CC3}$ = 3.3V (Nominal); CPU mode = "Sleep" | | | 4 | mA | $I_{CC}$ for $V_{CC3}$, Note 3 |
| $I_{CC2IDLE}$ | Core Current @ $V_{CC2}$ = 2.5V (Nominal); CPU mode = "Active Idle" $I_{CC2IDLE}$ at $f_{CLK}$ = 233 MHz | | 275 | | mA | $I_{CC}$ for $V_{CC2}$, Note 1 |
| $I_{CC2STBY}$ | Core Current @ $V_{CC2}$ = 2.5V (Nominal); CPU mode = "Standby" | | | 18 | mA | $I_{CC}$ for $V_{CC2}$, Note 2 |
| $I_{CC2SLP}$ | Core Current @ $V_{CC2}$ = 2.5V (Nominal); CPU mode = "Sleep" | | | 8 | mA | $I_{CC}$ for $V_{CC2}$, Note 3 |

**Notes:**
1. $f_{CLK}$ ratings refer to internal clock frequency.
2. All inputs are at 0.2V or $V_{CC3}$ – 0.2 (CMOS levels). All inputs except clock are held static, and all outputs are unloaded (static $I_{OUT}$ = 0 mA).
3. All inputs are at 0.2V or $V_{CC3}$ – 0.2 (CMOS levels). All inputs are held static, and all outputs are unloaded (static $I_{OUT}$ = 0 mA).

## Electrical Specifications (Continued)

**Table 6-11. 2.9V DC Characteristics for CPU Mode = "On"**

| Symbol | Parameter | Typ Avg | Abs Max | Units | Comments |
|---|---|---|---|---|---|
| $I_{CC3ON}$ | I/O Current @$V_{CC3}$ = 3.3V (Nominal); CPU mode = "On" $I_{CC3}$ at $f_{CLK}$ = 266 MHz | 160 | 415 | mA | $I_{CC}$ for $V_{CC3}$, Note |
| $I_{CC2ON}$ | Core Current @$V_{CC2}$ = 2.9V (Nominal); CPU mode = "On" $I_{CC2}$ at $f_{CLK}$ = 266 MHz | 1600 | 2100 | mA | $I_{CC}$ for $V_{CC2}$, Note |

**Note:** $f_{CLK}$ ratings refer to internal clock frequency.

**Table 6-12. 2.9V DC Characteristics for CPU Mode = "Active Idle", "Standby", and "Sleep"**

| Symbol | Parameter | Min | Typ | Max | Units | Comments |
|---|---|---|---|---|---|---|
| $I_{CC3IDLE}$ | I/O Current @$V_{CC3}$ = 3.3V (Nominal); CPU mode = "Active Idle" $I_{CC3IDLE}$ at $f_{CLK}$ = 266 MHz | | 150 | | mA | $I_{CC}$ for $V_{CC3}$, Note 1 |
| $I_{CC3STBY}$ | I/O Current @ $V_{CC3}$ = 3.3V (Nominal); CPU mode = "Standby" | | | 9 | mA | $I_{CC}$ for $V_{CC3}$, Note 2 |
| $I_{CC3SLP}$ | I/O Current @ $V_{CC3}$ = 3.3V (Nominal); CPU mode = "Sleep" | | | 5 | mA | $I_{CC}$ for $V_{CC3}$, Note 3 |
| $I_{CC2IDLE}$ | Core Current @$V_{CC2}$ = 2.9V (Nominal); CPU mode = "Active Idle" $I_{CC2IDLE}$ at $f_{CLK}$ = 266 MHz | | 380 | | mA | $I_{CC}$ for $V_{CC2}$, Note 1 |
| $I_{CC2STBY}$ | Core Current @ $V_{CC2}$ = 2.9V (Nominal); CPU mode = "Standby" | | | 22 | mA | $I_{CC}$ for $V_{CC2}$, Note 2 |
| $I_{CC2SLP}$ | Core Current @ $V_{CC2}$ = 2.9V (Nominal); CPU mode = "Sleep" | | | 10 | mA | $I_{CC}$ for $V_{CC2}$, Note 3 |

**Notes:**
1. $f_{CLK}$ ratings refer to internal clock frequency.
2. All inputs are at 0.2V or $V_{CC3}$ – 0.2 (CMOS levels). All inputs except clock are held static, and all outputs are unloaded (static $I_{OUT}$ = 0 mA)
3. All inputs are at 0.2V or $V_{CC3}$ – 0.2 (CMOS levels). All inputs are held static, and all outputs are unloaded (static $I_{OUT}$ = 0 mA).

## Electrical Specifications (Continued)

### 6.6 I/O CURRENT DE-RATING CURVE

As mentioned Section 6.5.2.3 “Definition of System Conditions for Measuring "On" Parameters” on page 191, the I/O current of the processor is affected by two system parameters, DCLK and SDRAM frequency. A de-rating curve (see Figure 6-3) is provided so that the system designer can determine the absolute maximum I/O current used by the processor for a particular design. Core current is not significantly affected by these two parameters, so a core current de-rating curve is not provided.

#### 6.6.1 Display Resolution

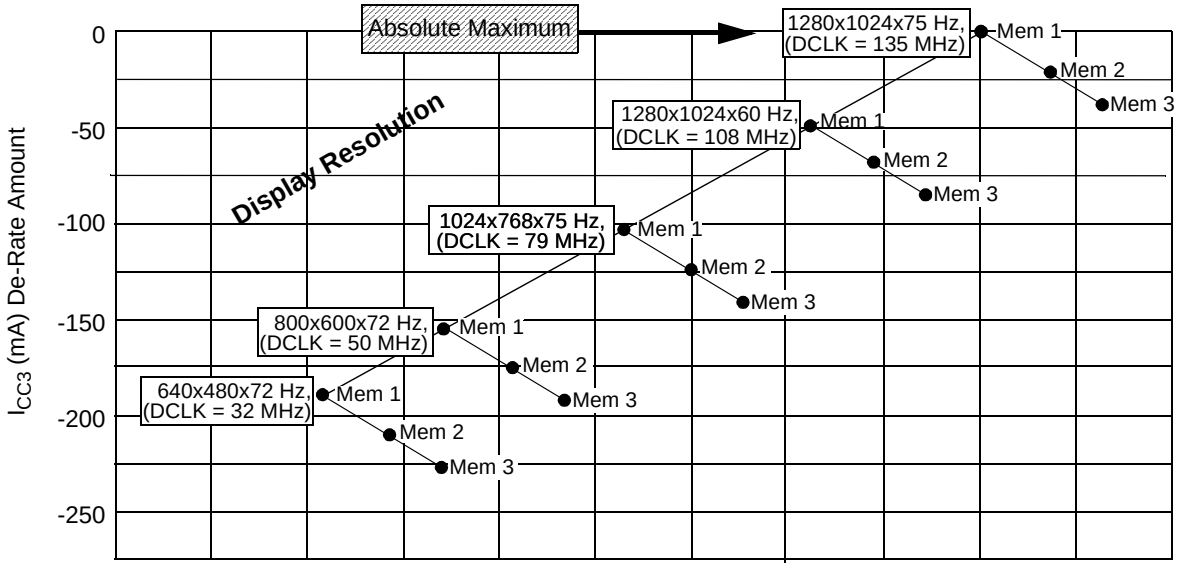
The change in current of five common display resolutions is used to extrapolate the de-rating curve. DCLK is derived from the display resolution, color depth, and refresh rate. The relationship between DCLK and I/O current is linear. The system designer must determine the maximum DCLK frequency required in the system based on the maximum display that will be supported.

#### 6.6.2 Memory Speed

Each device in the GXLV processor series is defined by a particular core voltage and core frequency. The SDRAM frequency is derived internally by a programmable divisor of the core frequency. Typically, there are three SDRAM frequencies between 55 and 100 MHz that can be derived from a single core frequency. These three frequencies are provided in the following de-rating curve so that their effect on current can be seen. Just as with the display resolution, current de-rating due to memory speed is linear. SDRAM frequencies between 79 and 100 MHz are only supported for certain types of closed systems and strict design rules must be adhered to. For further details, please contact your local National Semiconductor technical support representative.

#### 6.6.3 I/O Current De-rating Curve

The I/O current de-rating curve, shown in Figure 6-3, is the same for all devices in the GXLV series of processors. While the memory speeds for the various core frequencies are different, the three memory speeds for each device produce the same de-rating effect.




| MHz | Mem 1 MHz | | Mem 2 MHz | | Mem 3 MHz | |
|---|---|---|---|---|---|---|
| 166 | ÷2.0 | 83.3 | ÷2.5 | 66.7 | ÷3.0 | 55.6 |
| 180 | ÷2.0 | 90 | ÷2.5 | 72 | ÷3.0 | 60 |
| 200 | ÷2.0 | 100 | ÷2.5 | 80 | ÷3.0 | 66.7 |
| 233 | ÷2.5 | 93.3 | ÷3.0 | 77.8 | ÷3.5 | 66.7 |
| 266 | ÷3.0 | 88.9 | ÷3.5 | 76.2 | ÷4.0 | 66.7 |

**Note:** Pixel color depth does not affect power consumption or DCLK frequency.

**Figure 6-3. Absolute Max I/O Current De-rating Curve (All Speeds and Core Voltages)**

# Electrical Specifications (Continued)

## 6.7 AC CHARACTERISTICS

The following tables list the AC characteristics including output delays, input setup requirements, input hold requirements, and output float delays. The rising-clock-edge reference level $V_{REF}$, and other reference levels are shown in Table 6-13. Input or output signals must cross these levels during testing.

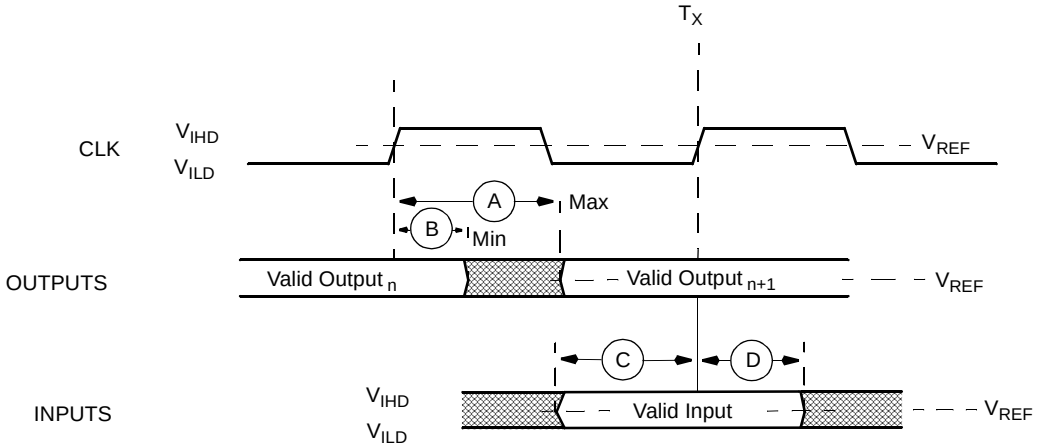
Input setup and hold times are specified minimums that define the smallest acceptable sampling window for which a synchronous input signal must be stable for correct operation.

All AC tests are performed at $V_{CC2}$ = 2.1V to 2.31V (2.2V Nominal), $V_{CC2}$ = 2.37V to 2.63V (2.5V Nominal), $V_{CC2}$ = 2.76V to 3.05V (2.9V Nominal), $V_{CC3}$ = 3.0V to 3.6V (3.3V Nominal), $T_C$ = 0°C to 85°C, $R_L$ = 50 ohms, and $C_L$ = 50 pF unless otherwise specified

While most minimum, maximum, and typical AC characteristics are only shown as a single value, they are tested and guaranteed across the entire processor core voltage range of 2.2V to 2.9V (nominal). AC characteristics that are affected significantly by the core voltage or speed grade are documented accordingly.

**Table 6-13. Drive Level and Measurement Points for Switching Characteristics**

| Symbol | Voltage (V) |
|---|---|
| $V_{REF}$ | 1.5 |
| $V_{IHD}$ | 2.4 |
| $V_{ILD}$ | 0.4 |




**Legend:** A = Maximum Output Delay Specification
B = Minimum Output Delay Specification
C = Minimum Input Setup Specification
D = Minimum Input Hold Specification

**Figure 6-4. Drive Level and Measurement Points for Switching Characteristics**

## Electrical Specifications (Continued)

**Table 6-14. Clock Signals (Refer to Figures 6-5 and 6-6)**

| Symbol | Parameter | SYSCLK = 33 MHz | | | SYSCLK = 30 MHz | | | Units | Comments |
|---|---|---|---|---|---|---|---|---|---|
| | | Min | Typ | Max | Min | Typ | Max | | |
| t1 | SYSCLK Period | 29.75 | 30.0 | 30.25 | 33.05 | 33.3 | 33.55 | ns | Note 1 |
| t2 | SYSCLK Period Stability | | | ±250 | | | ±250 | ps | |
| t3 | SYSCLK High Time | 10.5 | | | 11.66 | | | ns | |
| t4 | SYSCLK Low Time | 10.5 | | | 11.66 | | | ns | |
| t5 | SYSCLK Fall Time | 0.5 | | 1.5 | 0.5 | | 1.5 | ns | Note 3 |
| t6 | SYSCLK Rise Time | 0.5 | | 1.5 | 0.5 | | 1.5 | ns | Note 3 |
| t9 | SDCLK_OUT, SDCLK[3:0] Period | | | | | | | | |
| | 166 MHz / 2.5 | 13 | 15.0 | 17 | | | | ns | Note 2 |
| | 180 MHz / 2.5 | | | | 11.9 | 13.9 | 16.9 | | |
| | 180 MHz / 3 | | | | 14.7 | 16.7 | 18.7 | | |
| | 200 MHz / 3 | 13 | 15.0 | 17 | | | | | |
| | 233 MHz / 3 | 10.9 | 12.9 | 15.9 | | | | | |
| | 233 MHz / 3.5 | 13 | 15.0 | 17 | | | | | |
| | 266 MHz / 3.5 | 11.1 | 13.1 | 16.1 | | | | | |
| | 266 MHz / 4 | 13 | 15.0 | 17 | | | | | |
| t10 | SDCLK_OUT, SDCLK[3:0] High Time | | | | | | | | |
| | 166 MHz / 2.5 | 6.5 | | | | | | ns | Note 2 |
| | 180 MHz / 2.5 | | | | 5.95 | | | | |
| | 180 MHz / 3 | | | | 7.35 | | | | |
| | 200 MHz / 3 | 6.5 | | | | | | | |
| | 233 MHz / 3 | 5.45 | | | | | | | |
| | 233 MHz / 3.5 | 6.5 | | | | | | | |
| | 266 MHz / 3.5 | 5.55 | | | | | | | |
| | 266 MHz / 4 | 6.5 | | | | | | | |
| t11 | SDCLK_OUT, SDCLK[3:0] Low Time | | | | | | | | |
| | 166 MHz / 2.5 | 6.5 | | | | | | ns | Note 2 |
| | 180 MHz / 2.5 | | | | 5.95 | | | | |
| | 180 MHz / 3 | | | | 7.35 | | | | |
| | 200 MHz / 3 | 6.5 | | | | | | | |
| | 233 MHz / 3 | 5.45 | | | | | | | |
| | 233 MHz / 3.5 | 6.5 | | | | | | | |
| | 266 MHz / 3.5 | 5.55 | | | | | | | |
| | 266 MHz / 4 | 6.5 | | | | | | | |

## Electrical Specifications (Continued)

**Table 6-14. Clock Signals (Refer to Figures 6-5 and 6-6) (Continued)**

| Symbol | Parameter | SYSCLK = 33 MHz | | | SYSCLK = 30 MHz | | | Units | Comments |
|---|---|---|---|---|---|---|---|---|---|
| | | Min | Typ | Max | Min | Typ | Max | | |
| t12 | SDCLK_OUT, SDCLK[3:0] Fall Time | | | | | | | | |
| | 166 MHz / 2.5 | 0.5 | | | | | | ns | Note 3 |
| | 180 MHz / 2.5 | | | | 0.5 | | | | |
| | 180 MHz / 3 | 0.5 | | | | | | | |
| | 200 MHz / 3 | 0.5 | | | | | | | |
| | 233 MHz / 3 | 0.5 | | | | | | | |
| | 233 MHz / 3.5 | 0.5 | | | | | | | |
| | 266 MHz / 3.5 | 0.5 | | | | | | | |
| | 266 MHz / 4 | 0.5 | | | | | | | |
| t13 | SDCLK_OUT, SDCLK[3:0] Rise Time | | | | | | | | |
| | 166 MHz | 0.45 | | | | | | ns | Note 3 |
| | 180 MHz | | | | 0.45 | | | | |
| | 180 MHz | | | | | | | | |
| | 200 MHz | 0.45 | | | | | | | |
| | 233 MHz | 0.45 | | | | | | | |
| | 233 MHz | | | | | | | | |
| | 266 MHz | 0.45 | | | | | | | |
| | 266 MHz | | | | | | | | |

**Notes:** 1. A SYSCLK of 30 MHz corresponds to a core frequency of 180 MHz. A SYSCLK of 33 MHz corresponds to core frequencies of 166, 200, 233, and 266 MHz.

2. SDCLK calculations are based on the following officially supported configurations:

166 MHz (5x) / 2.5 = 66.4 MHz SDCLK_OUT
180 MHz (5x) / 2.5 = 72 MHz SDCLK_OUT
180 MHz (6x) / 3 = 60 MHz SDCLK_OUT
200 MHz (6x) / 3 = 66.7 MHz SDCLK_OUT
233 MHz (7x) / 3 = 77.7 MHz SDCLK_OUT
233 MHz (7x) / 3.5 = 66.6 MHz SDCLK_OUT
266 MHz (8x) / 3.5 = 76 MHz SDCLK_OUT
266 MHz (8x) / 4 = 66.5 MHz SDCLK_OUT

3. SDCLK_OUT and SYSCLK rise and fall times are measured between $V_{IH}$ min and $V_{IL}$ max with a 50 pF load.

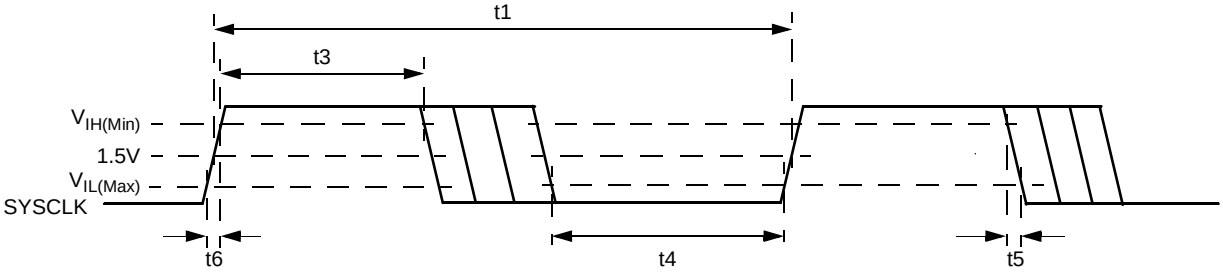



**Figure 6-5. SYSCLK Timing and Measurement Points**

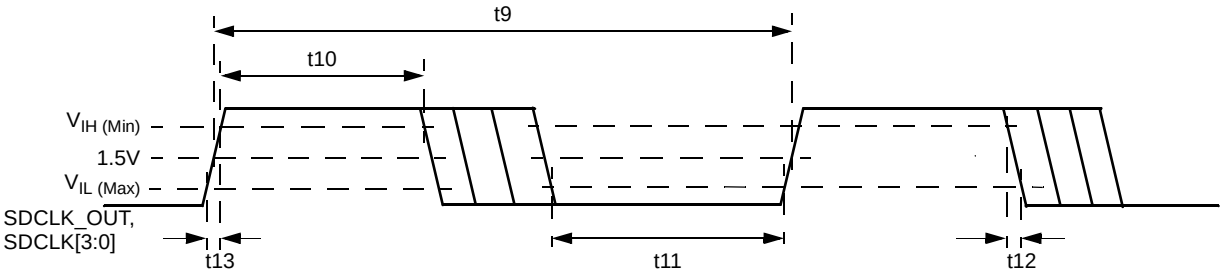



**Figure 6-6. SDCLK[3:0] Timing and Measurement Points**

## Electrical Specifications (Continued)

**Table 6-15. System Signals**

| Parameter | Min | Max | Unit | Comments |
|---|---|---|---|---|
| Setup Time for RESET, INTR | 5 | | ns | Note |
| Hold Time for RESET, INTR | 2 | | ns | Note |
| Setup Time for SMI#, SUSP#, FLT# | 5 | | ns | |
| Hold Time for SMI#, SUSP#, FLT# | 2 | | ns | |
| Valid Delay for IRQ13, SUSPA# | 2 | 15 | ns | |
| Valid Delay for SERIALP | 2 | 15 | ns | |

**Note:** The system signals may be asynchronous. The setup/hold times are required for determining static behavior.

## Electrical Specifications (Continued)

**Table 6-16. PCI Interface Signals (Refer to Figures 6-7 and 6-8)**

| Symbol | Parameter | Min | Max | Unit | Comments |
|---|---|---|---|---|---|
| $t_{VAL1}$ | Delay Time, SYSCLK to Signal Valid for Bused Signals | 2 | 11 | ns | |
| $t_{VAL2}$ | Delay Time, SYSCLK to Signal Valid for GNT# | 2 | 9 | ns | Notes 1, 2 |
| $t_{ON}$ | Delay Time, Float to Active | 2 | | ns | |
| $t_{OFF}$ | Delay Time, Active to Float | | 28 | ns | |
| $t_{SU1}$ | Input Setup Time for Bused Signals | 7 | | ns | |
| $t_{SU2}$ | Input Setup Time for REQ# | 6 | | ns | Notes 1, 2 |
| $t_H$ | Input Hold Time to SYSCLK | 0 | | ns | |

**Notes:** 1. GNT# and REQ# are point-to-point signals. All other PCI interface signals are bused. Refer to Chapter 4 of PCI Local Bus Specification, Revision 2.1, for more detailed information.

2. Maximum timings are improved over the PCI Local Bus Specification, Revision 2.1. This allows a PAL or some other circuit to use a REQ/GNT pair to expand the number of REQ/GNT pairs available to the system (See application note, “GXLV Processor Series: Request/Grant Pair Expansion”.

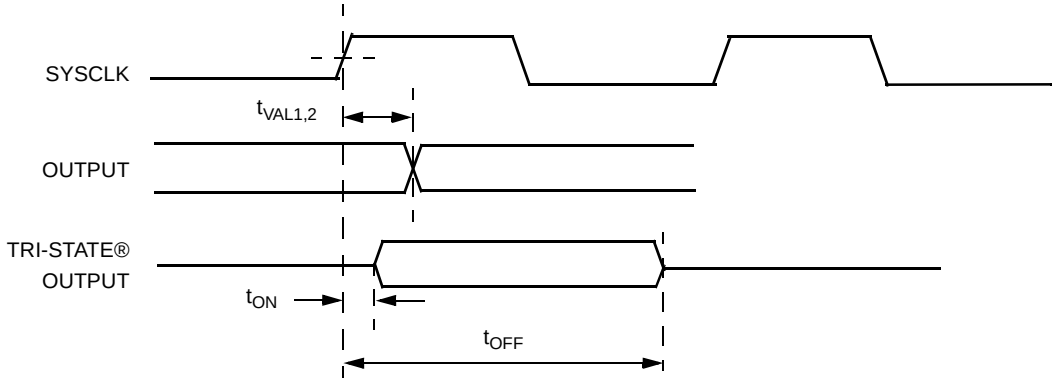



**Figure 6-7. Output Timing**

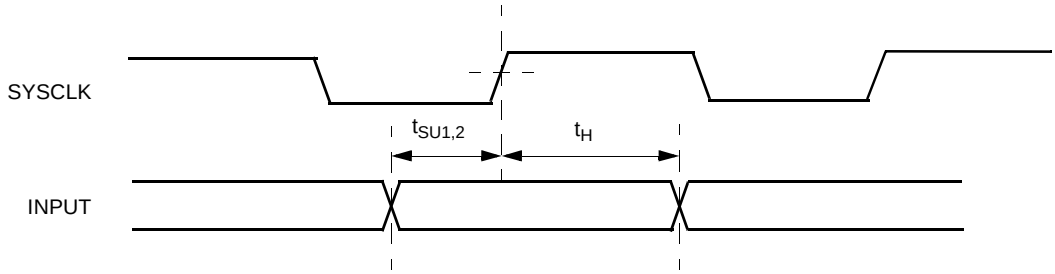



**Figure 6-8. Input Timing**

## Electrical Specifications (Continued)

**Table 6-17. SDRAM Interface Signals (Refer to Figures 6-9 and 6-10)**

| Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t1 | RASA#, RASB#, CASA#, CASB#, WEA#, WEB#, CKEA, CKEB, DQM[7:0], CS[3:0]# Ouput Valid from SDCLK[3:0] | t1 Min = z – 1.5 | t1 Max = z – 1.0 | ns |
| t2 | MA[12:0], BA[1:0] Output Valid from SDCLK[3:0] | t2 Min = z – 1.7 | t2 Max = z – 1.2 | ns |
| t3 | MD[63:0] Output Valid from SDCLK[3:0] | t2 Min = z – 1.6 | t3 Max = z – 0.3 | ns |
| t4 | MD[63:0] Read Data in Setup to SDCLK_IN | 0 | | ns |
| t5 | MD[63:0] Read Data Hold to SDCLK_IN | 2.0 | | ns |

**Calculation of minimum and maximum values of t1, t2, and t3**: (see Figure 4-10 on page 124)

x =shift value applied to SHFTSDCLK field where SHFTSDCLK field = GX_BASE+8404h[5:3].
y = core clock period ÷ 2
z = (x * y)

**Equation Example:**

A 200 MHz GXLV processor interfacing with a 66 MHz SDRAM bus, having a shift value of 2:
x = 2
core clock period = 1/(200 MHz) = 5 ns
y = 5 ÷ 2
t1 Min = (2 * (5 ÷ 2)) – 1.5 = 3.5 ns
t1 Max = (2 * (5 ÷ 2)) – 1.0 = 4.0 ns

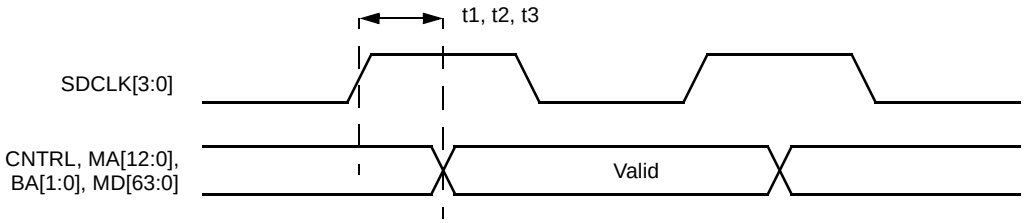



**Figure 6-9. Output Valid Timing**

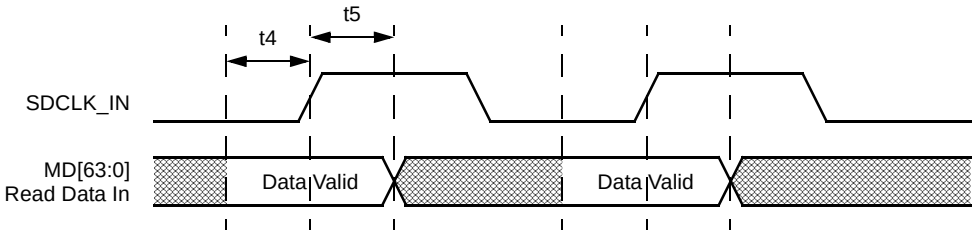



**Figure 6-10. Setup and Hold Timings - Read Data In**

## Electrical Specifications (Continued)

**Table 6-18. Video Interface Signals (Refer to Figures 6-11 through 6-13)**

| Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t1 | PCLK Period | 6.5 | 40 | ns |
| t2 | PCLK High Time | 3 | | ns |
| t3 | PCLK Low Time | 3 | | ns |
| t4 | PIXEL[17:0], CRT_HSYNC, CRT_VSYNC, FP_HSYNC, FP_VSYNC, ENA_DISP Valid Delay from PCLK Rising Edge | 2 | 5 | ns |
| t5 | VID_CLK Period | 8.5 | | ns |
| t6 | VID_RDY Setup to VID_CLK Rising Edge | 5 | | ns |
| t7 | VID_RDY Hold to VID_CLK Rising Edge | 2 | | ns |
| t8 | VID_VAL, VID_DATA[7:0] Valid Delay from VID_CLK Rising Edge | 2 | 5 | ns |
| t9 | DCLK Period | 6.5 | | ns |
| t10 | DCLK Rise/Fall Time | | 2 | ns |
| tcyc | DCLK Duty Cycle | 40 | 60 | % |

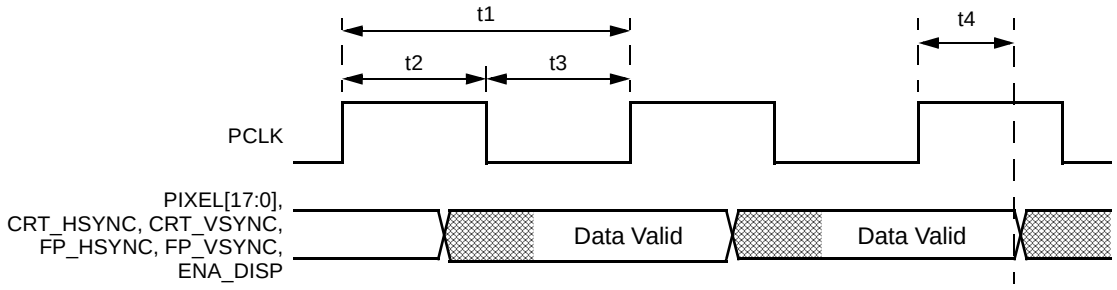



**Figure 6-11. Graphics Port Timing**

## Electrical Specifications (Continued)

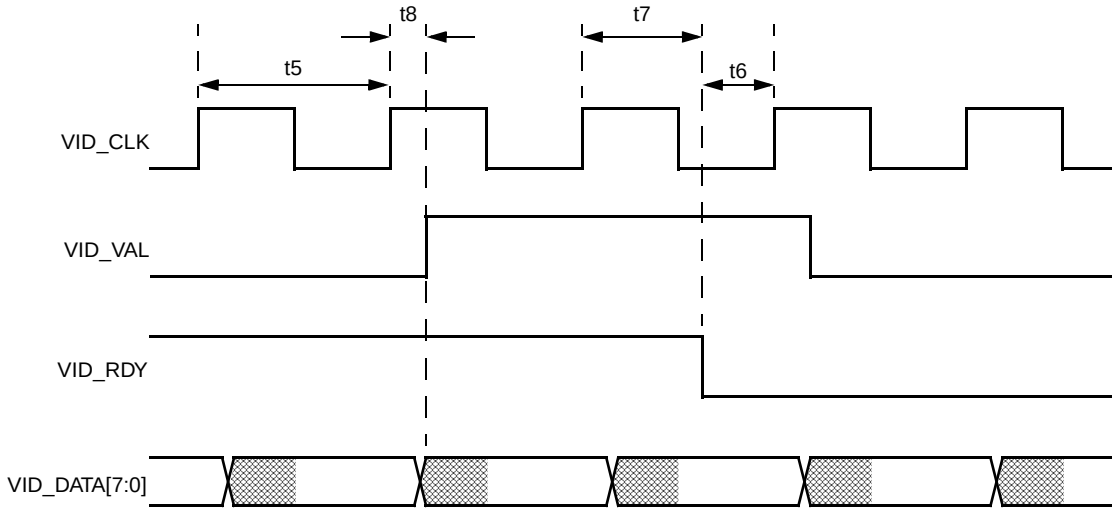



**Figure 6-12. Video Port Timing**

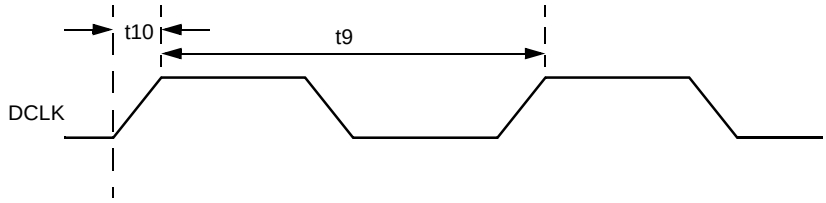



**Figure 6-13. DCLK Timing**

## Electrical Specifications (Continued)

**Table 6-19. JTAG AC Specification (Refer to Figures 6-14 and 6-15)**

| Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| | TCK Frequency (MHz) | | 25 | MHz |
| t1 | TCK Period | 40 | | ns |
| t2 | TCK High Time | 10 | | ns |
| t3 | TCK Low Time | 10 | | ns |
| t4 | TCK Rise Time | | 4 | ns |
| t5 | TCK Fall Time | | 4 | ns |
| t6 | TDO Valid Delay | 3 | 25 | ns |
| t7 | Non-test Outputs Valid Delay | 3 | 25 | ns |
| t8 | TDO Float Delay | | 30 | ns |
| t9 | Non-test Outputs Float Delay | | 36 | ns |
| t10 | TDI, TMS Setup Time | 8 | | ns |
| t11 | Non-test Inputs Setup Time | 8 | | ns |
| t12 | TDI, TMS Hold Time | 7 | | ns |
| t13 | Non-test Inputs Hold Time | 7 | | ns |

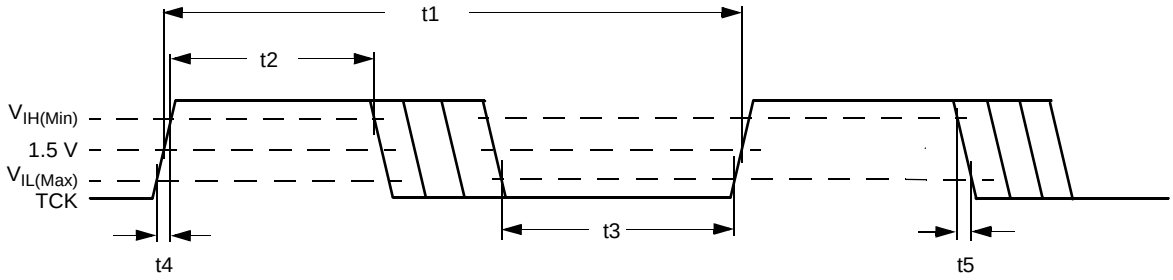



**Figure 6-14. TCK Timing and Measurement Points**

## Electrical Specifications (Continued)

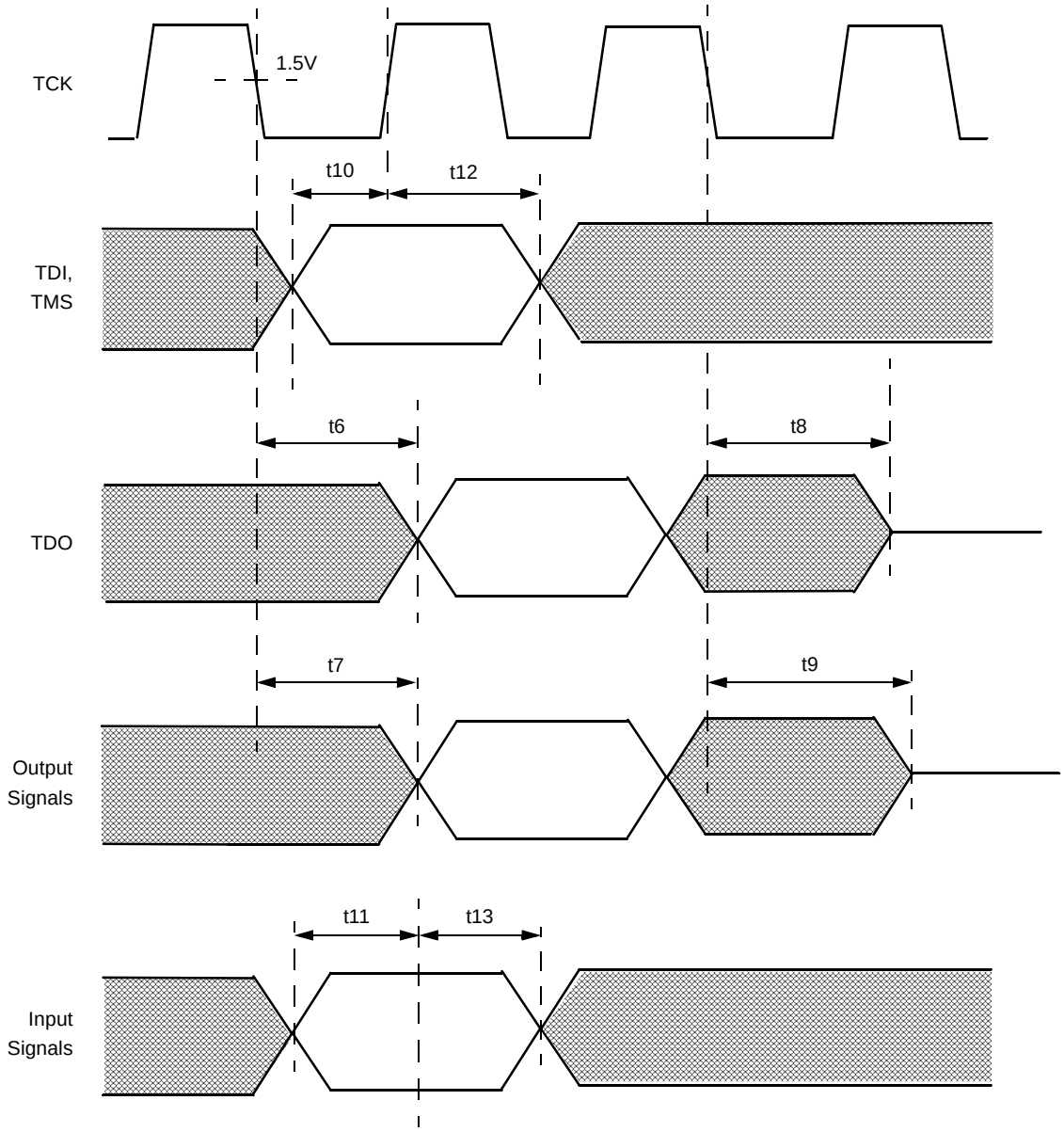



**Figure 6-15. JTAG Test Timings**

# 7.0 Package Specifications

The thermal characteristics and mechanical dimensions for the Geode GXLV processor are provided on the following pages.

## 7.1 THERMAL CHARACTERISTICS

Table 7-1 shows the junction-to-case thermal resistance of the SPGA and BGA package and can be used to calculate the junction (die) temperature under any given circumstance.

**Table 7-1. Junction-to-Case Thermal Resistance for SPGA and BGA Packages**

| Package | $\theta_{JC}$ |
|---|---|
| SPGA | 1.7 °C/W |
| BGA | 1.1 °C/W |

Note that there is no specification for maximum junction temperature given since the operation of both SPGA and BGA devices are guaranteed to a case temperature range of 0°C to 85°C (see $T_C$ in Table 6-4 on page 189). As long as the case temperature of the device is maintained within this range, the junction temperature of the die will also be maintained within its allowable operating range. However, the die (junction) temperature under a given operating condition can be calculated by using the following equation:

$$T_J = T_C + (P * \theta_{JC})$$

where:

$T_J$ = Junction temperature (°C)

$T_C$ = Case temperature at top center of package (°C)

P = Maximum power dissipation (W)

$\theta_{JC}$ = Junction-to-case thermal resistance (°C/W)

These examples are given for reference only. The actual value used for maximum power (P) and ambient temperature ($T_A$) is determined by the system designer based on system configuration, extremes of the operating environment, and whether active thermal management (via Suspend Modulation) of the processor is employed.

A maximum junction temperature is not specified since a maximum case temperature is. Therefore, the following equation can be used to calculate the maximum thermal resistance required of the thermal solution for a given maximum ambient temperature:

$$\theta_{CS} + \theta_{SA} = \frac{T_C - T_A}{P}$$

where:

$\theta_{CS}$ = Max case-to-heatsink thermal resistance (°C/W) allowed for thermal solution

$\theta_{SA}$ = Max heatsink-to-ambient thermal resistance (°C/W) allowed for thermal solution

$T_A$ = Max ambient temperature (°C)

$T_C$ = Max case temperature at top center of package (°C)

P = Max power dissipation (W)

If thermal grease is used between the case and heatsink, $\theta_{CS}$ will reduce to about 0.01 °C/W. Therefore, the above equation can be simplified to:

$$\theta_{CA} = \frac{T_C - T_A}{P}$$

where:

$\theta_{CA} = \theta_{CS}$ = Max case-to-ambient thermal resistance (°C/W) allowed for thermal solution.

The calculated $\theta_{CA}$ value (examples shown in Table 7-2) represents the maximum allowed thermal resistance of the selected cooling solution which is required to maintain the maximum $T_C$ (shown in Table 6-4 on page 189) for the application in which the device is used.

## Package Specifications (Continued)

**Table 7-2. Case-to-Ambient Thermal Resistance Examples @ 85°C**

| Core Voltage ($V_{CC2}$) | Core Frequency | Maximum Power | $\theta_{CA}$ for Different Ambient Temperatures (°C/W) | | | | |
|---|---|---|---|---|---|---|---|
| | | | 20°C | 25°C | 30°C | 35°C | 40°C |
| 2.9V (Nominal) | 266 MHz | 7.7W | 8.44 | 7.79 | 7.14 | 6.49 | 5.84 |
| 2.5V (Nominal) | 233 MHz | 5.4W | 12.04 | 11.11 | 10.19 | 9.26 | 8.33 |
| 2.2V (Nominal) | 200 MHz | 3.8W | 17.11 | 15.08 | 14.47 | 13.18 | 11.84 |
| | 180 MHz | 3.6W | 18.06 | 16.67 | 15.28 | 13.89 | 12.50 |
| | 166 MHz | 3.4W | 19.12 | 17.65 | 16.18 | 14.71 | 13.24 |

### 7.1.1 Heatsink Considerations

Table 7-2 shows the maximum allowed thermal resistance of a heatsink for particular operating environments. The calculated values, defined as $\theta_{CA}$, represent the required ability of a particular heatsink to transfer heat generated by the processor from its case into the air, thereby maintaining the case temperature at or below 85°C. Because $\theta_{CA}$ is a measure of thermal *resistivity*, it is inversely proportional to the heatsink's ability to dissipate heat or it's thermal *conductivity*.

**Note:** A "perfect" heatsink would be able to maintain a case temperature equal to that of the ambient air inside the system chassis.

Looking at Table 7-2, it can be seen that as ambient temperature ($T_A$) increases, $\theta_{CA}$ decreases, and that as power consumption of the processor (P) increases, $\theta_{CA}$ decreases. Thus, the ability of the heatsink to dissipate thermal energy must increase as the processor power increases and as the temperature inside the enclosure increases.

While $\theta_{CA}$ is a useful parameter to calculate, heatsinks are not typically specified in terms of a single $\theta_{CA}$. This is because the thermal resistivity of a heatsink is not constant across power or temperature. In fact, heatsinks become slightly less efficient as the amount of heat they are trying to dissipate increases. For this reason, heatsinks are typically specified by graphs that plot heat dissipation (in watts) vs. mounting surface (case) temperature rise above ambient (in °C). This method is necessary because ambient and case temperatures fluctuate constantly during normal operation of the system. The system designer must be careful to choose the proper heatsink by matching the required $\theta_{CA}$ with the thermal dissipation curve of the device under the entire range of operating conditions in order to make sure that the maximum case temperature from Table 6-4 on page 189 is never exceeded. To choose the proper heatsink, the system designer must make sure that the calculated $\theta_{CA}$ falls above the curve (shaded area). The curve itself defines the minimum temperature rise above ambient that the heatsink can maintain.

See Figure 7-1 as an example of a particular heatsink under consideration.

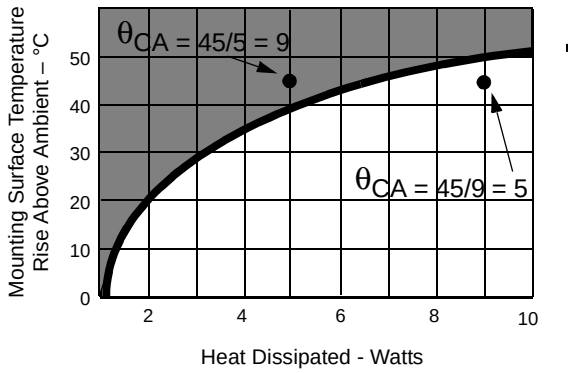



**Figure 7-1. Heatsink Example**

## Package Specifications (Continued)

### Example 1

Assume P (max) = 5W and $T_A$ (max) = 40°C.

Therefore:

$$\theta_{CA} = \frac{T_C - T_A}{P}$$

$$\theta_{CA} = \frac{(85 - 40)}{5}$$

$$\theta_{CA} = 9$$

In this case, the heatsink under consideration is more than adequate since at 5W worst case, it can maintain a 40°C case temperature rise above ambient ($\theta_{CA}$ = 9) when a maximum of 45°C ($\theta_{CA}$ = 8) is required.

### Example 2

Assume P (max) = 10W and $T_A$ (max) = 40°C.

Therefore:

$$\theta_{CA} = \frac{T_C - T_A}{P}$$

$$\theta_{CA} = \frac{(85 - 40)}{9}$$

$$\theta_{CA} = 5$$

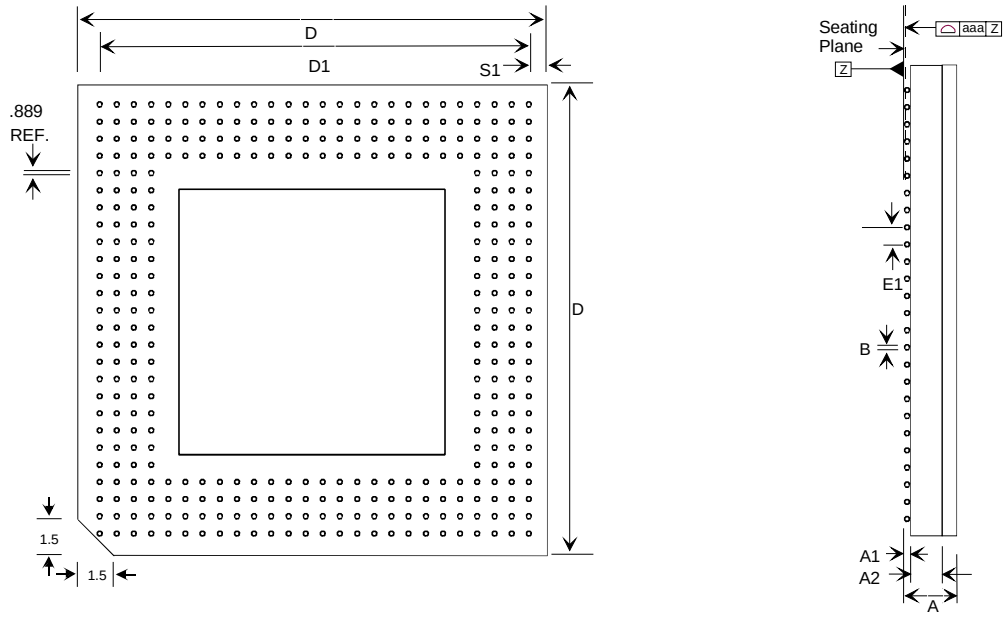
In this case, the heatsink under consideration is NOT adequate to maintain the 45°C case temperature rise above ambient for a 9W processor.

For more information on thermal design considerations or heatsink properties, refer to the Product Selection Guide of any leading vendor of thermal engineering solutions.

## Package Specifications (Continued)

### 7.2 MECHANICAL PACKAGE OUTLINES

Dimensions for the BGA package are shown in Figure 7-2. Figure 7-3 shows the SPGA dimensions. Table 7-3 gives the legend for the symbols used in both package outlines.




| Sym | Millimeters | | Inches | |
|---|---|---|---|---|
| | Min | Max | Min | Max |
| A | 1.45 | 2.23 | 0.057 | 0.088 |
| A1 | 0.50 | 0.70 | 0.020 | 0.028 |
| A2 | 0.43 | 0.83 | 0.017 | 0.033 |
| aaa | | 0.20 | | 0.008 |
| B | 0.60 | 0.90 | 0.024 | 0.035 |
| D | 34.80 | 35.20 | 1.370 | 1.386 |
| D1 | 31.55 | 31.95 | 1.242 | 1.258 |
| D2 | 32.80 | 35.20 | 1.291 | 1.386 |
| E1 | 1.12 | 1.42 | 0.044 | 0.056 |
| F | | 0.35 | | 0.014 |
| S1 | 1.42 | 1.82 | 0.056 | 0.072 |

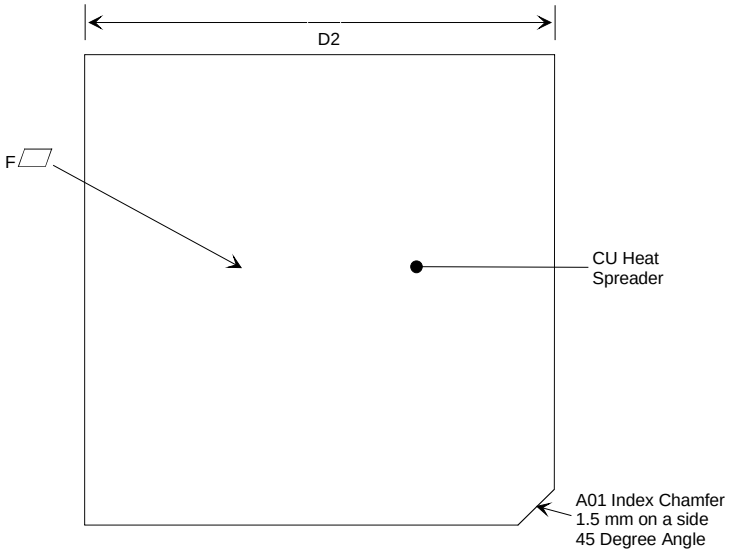
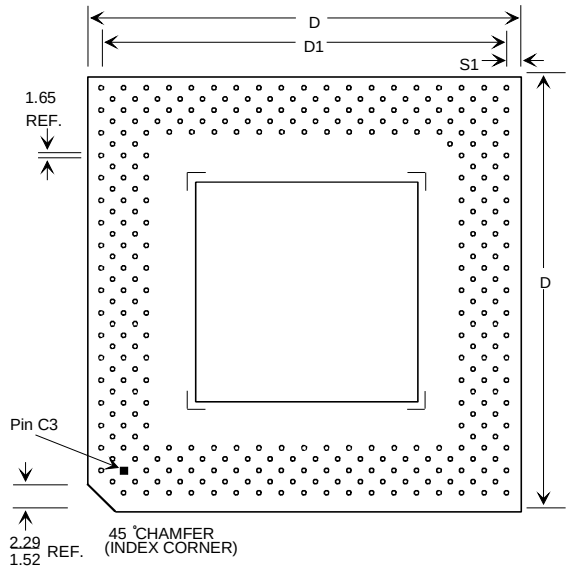
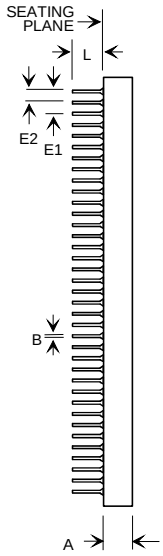



**Figure 7-2. 352-Terminal BGA Mechanical Package Outline**

## Package Specifications (Continued)




| Sym | Millimeters | | Inches | |
|---|---|---|---|---|
| | Min | Max | Min | Max |
| A | 2.51 | 3.07 | 0.099 | 0.121 |
| B | 0.43 | 0.51 | 0.017 | 0.020 |
| D | 49.28 | 49.91 | 1.940 | 1.965 |
| D1 | 45.47 | 45.97 | 1.790 | 1.810 |
| E1 | 2.41 | 2.67 | 0.095 | 0.105 |
| E2 | 1.14 | 1.40 | 0.045 | 0.055 |
| F | -- | 0.127 Diag | -- | 0.005 Diag |
| L | 2.97 | 3.38 | 0.117 | 0.133 |
| S1 | 1.65 | 2.16 | 0.065 | 0.085 |

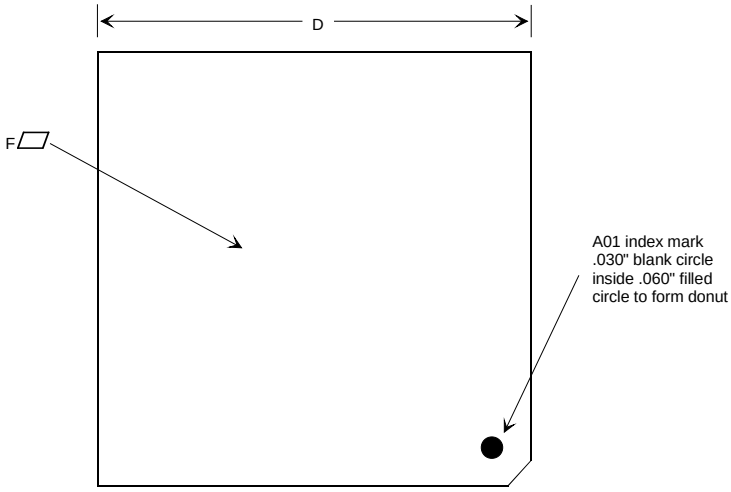



**Figure 7-3. 320-Pin SPGA Mechanical Package Outline**

## Package Specifications (Continued)

**Table 7-3. Mechanical Package Outline Legend**

| Symbol | Meaning |
|---|---|
| A | Distance from seating plane datum to highest point of body |
| A1 | Solder ball height |
| A2 | Laminate thickness (excluding heat spreader) |
| aaa | Coplanarity |
| B | Pin or solder ball diameter |
| D | Largest overall package outline dimension |
| D1 | Length from outer pin center to outer pin center |
| D2 | Heat spreader outline dimension |
| E1 | BGA: Solder ball pitch<br>SPGA: Linear spacing between true pin position centerlines |
| E2 | Diagonal spacing between true pin position centerlines |
| F | Flatness |
| L | Distance from seating plane to tip of pin |
| S1 | Length from outer pin/ball center to edge of laminate |

# 8.0 Instruction Set

This section summarizes the Geode GXLV processor instruction set and provides detailed information on the instruction encodings. The instruction set is divided into four categories:

- Processor Core Instruction Set - listed in Table 8-27 on page 223.
- FPU Instruction Set - listed in Table 8-29 on page 235.
- MMX Instruction Set - listed in Table 8-31 on page 240.
- Extended MMX Instruction Set - listed in Table 8-33 on page 245.

These tables provide information on the instruction encoding, and the instruction clock counts for each instruction. The clock count values for these tables are based on the following assumptions

1. All clock counts refer to the internal processor core clock frequency. For example, clock doubled GXLV processor cores will reference a clock frequency that is twice the bus frequency.
2. The instruction has been prefetched, decoded and is ready for execution.
3. Bus cycles do not require wait states.
4. There are no local bus HOLD requests delaying processor access to the bus.
5. No exceptions are detected during instruction execution.
6. If an effective address is calculated, it does not use two general register components. One register, scaling and displacement can be used within the clock count shown. However, if the effective address calculation uses two general register components, add one clock to the clock count shown.
7. All clock counts assume aligned 32-bit memory/IO operands.
8. If instructions access a 32-bit operand on odd addresses, add one clock for read or write and add two clocks for read and write.
9. For non-cached memory accesses, add two clocks (clock doubled GXLV processor cores) or four clocks (clock tripled GXLV processor cores), assuming zero wait state memory accesses.
10. Locked cycles are not cacheable. Therefore, using the LOCK prefix with an instruction adds additional clocks as specified in item 9 above.

## 8.1 GENERAL INSTRUCTION SET FORMAT

Depending on the instruction, the GXLV processor core instructions follow the general instruction format shown in Table 8-1.

These instructions vary in length and can start at any byte address. An instruction consists of one or more bytes that can include prefix bytes, at least one opcode byte, a mod r/m byte, an s-i-b byte, address displacement, and immediate data. An instruction can be as short as one byte and as long as 15 bytes. If there are more than 15 bytes in the instruction, a general protection fault (error code 0) is generated.

The fields in the general instruction format at the byte level are summarized in Table 8-2 and detailed in the following subsections.

**Table 8-1. General Instruction Set Format**

| | | Register and Address Mode Specifier | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | mod r/m Byte | | | s-i-b Byte | | | | |
| Prefix (optional) | Opcode | mod | reg | r/m | ss | index | base | Address Displacement | Immediate Data |
| 0 or More Bytes | 1 or 2 Bytes | 7:6 | 5:3 | 2:0 | 7:6 | 5:3 | 2:0 | 0, 8, 16, or 32 Bits | 0, 8, 16, or 32 Bits |

**Table 8-2. Instruction Fields**

| Field Name | Description |
|---|---|
| Prefix (optional) | Prefix Field(s): One or more optional fields that are used to specify segment register override, address and operand size, repeat elements in string instruction, LOCK# assertion. |
| Opcode | Opcode Field: Identifies instruction operation. |
| mod | Address Mode Specifier: Used with r/m field to select addressing mode. |
| reg | General Register Specifier: Uses reg, sreg3 or sreg2 encoding depending on opcode field. |
| r/m | Address Mode Specifier: Used with mod field to select addressing mode. |
| ss | Scale factor: Determines scaled-index address mode. |
| index | Index: Determines general register to be used as index register. |
| base | Base: Determines general register to be used as base register. |
| Address Displacement | Displacement: Determines address displacement. |
| Immediate Data | Immediate Data: Immediate data operand used by instruction. |

## Instruction Set (Continued)

### 8.1.1 Prefix (Optional)

Prefix bytes can be placed in front of any instruction to modify the operation of that instruction. When more than one prefix is used, the order is not important. There are five types of prefixes that can be used:

1. Segment Override explicitly specifies which segment register the instruction will use for effective address calculation.
2. Address Size switches between 16-bit and 32-bit addressing by selecting the non-default address size.
3. Operand Size switches between 16-bit and 32-bit operand size by selecting the non-default operand size.
4. Repeat is used with a string instruction to cause the instruction to be repeated for each element of the string.
5. Lock is used to assert the hardware LOCK# signal during execution of the instruction.

Table 8-3 lists the encoding for different types of prefix bytes.

**Table 8-3. Instruction Prefix Summary**

| Prefix | Encoding | Description |
|---|---|---|
| ES: | 26h | Override segment default, use ES for memory operand. |
| CS: | 2Eh | Override segment default, use CS for memory operand. |
| SS: | 36h | Override segment default, use SS for memory operand. |
| DS: | 3Eh | Override segment default, use DS for memory operand. |
| FS: | 64h | Override segment default, use FS for memory operand. |
| GS: | 65h | Override segment default, use GS for memory operand. |
| Operand Size | 66h | Make operand size attribute the inverse of the default. |
| Address Size | 67h | Make address size attribute the inverse of the default. |
| LOCK | F0h | Assert LOCK# hardware signal. |
| REPNE | F2h | Repeat the following string instruction. |
| REP/REPE | F3h | Repeat the following string instruction. |

### 8.1.2 Opcode

The opcode field specifies the operation to be performed by the instruction. The opcode field is either one or two bytes in length and may be further defined by additional bits in the mod r/m byte. Some operations have more than one opcode, each specifying a different form of the operation. Certain opcodes name instruction groups. For example, opcode 80h names a group of operations that have an immediate operand and a register or memory operand. The reg field may appear in the second opcode byte or in the mod r/m byte.

The opcode may contain w, d, s and eee opcode fields, for example, as shown in Table 8-27 on page 223.

#### 8.1.2.1 w Field (Operand Size)

When used, the 1-bit w field selects the operand size during 16-bit and 32-bit data operations. See Table 8-4.

**Table 8-4. w Field Encoding**

| w Field | Operand Size | |
|---|---|---|
| | 16-Bit Data Operations | 32-Bit Data Operations |
| 0 | 8 bits | 8 bits |
| 1 | 16 bits | 32 bits |

#### 8.1.2.2 d Field (Operand Direction)

When used, the 1-bit d field determines which operand is taken as the source operand and which operand is taken as the destination. See Table 8-5.

**Table 8-5. d Field Encoding**

| d Field | Direction of Operation | Source Operand | Destination Operand |
|---|---|---|---|
| 0 | Register-to-Register or Register-to-Memory | reg | mod r/m or mod ss-index-base |
| 1 | Register-to-Register or Memory-to-Register | mod r/m or mod ss-index-base | reg |

## Instruction Set (Continued)

#### 8.1.2.3 s Field (Immediate Data Field Size)

When used, the 1-bit s field determines the size of the immediate data field. If the s bit is set, the immediate field of the opcode is 8 bits wide and is sign-extended to match the operand size of the opcode. See Table 8-6.

**Table 8-6. s Field Encoding**

| s Field | Immediate Field Size | | |
|---|---|---|---|
| | 8-Bit Operand Size | 16-Bit Operand Size | 32-Bit Operand Size |
| 0 (or not present) | 8 bits | 16 bits | 32 bits |
| 1 | 8 bits | 8 bits (sign-extended) | 8 bits (sign-extended) |

#### 8.1.2.4 eee Field (MOV-Instruction Register Selection)

The eee field (bits [5:3]) is used to select the control, debug and test registers in the MOV instructions. The type of register and base registers selected by the eee field are listed in Table 8-7. The values shown in Table 8-7 are the only valid encodings for the eee bits.

**Table 8-7. eee Field Encoding**

| eee Field | Register Type | Base Register |
|---|---|---|
| 000 | Control Register | CR0 |
| 010 | Control Register | CR2 |
| 011 | Control Register | CR3 |
| 100 | Control Register | CR4 |
| 000 | Debug Register | DR0 |
| 001 | Debug Register | DR1 |
| 010 | Debug Register | DR2 |
| 011 | Debug Register | DR3 |
| 110 | Debug Register | DR6 |
| 111 | Debug Register | DR7 |
| 011 | Test Register | TR3 |
| 100 | Test Register | TR4 |
| 101 | Test Register | TR5 |
| 110 | Test Register | TR6 |
| 111 | Test Register | TR7 |

### 8.1.3 mod and r/m Byte (Memory Addressing)

The mod and r/m fields within the mod r/m byte, select the type of memory addressing to be used. Some instructions use a fixed addressing mode (e.g., PUSH or POP) and therefore, these fields are not present. Table 8-8 lists the addressing method when 16-bit addressing is used and a mod r/m byte is present. Some mod r/m field encodings are dependent on the w field and are shown in Table 8-9.

**Table 8-8. mod r/m Field Encoding**

| mod Field | r/m Field | 16-Bit Address Mode with mod r/m Byte | 32-Bit Address Mode with mod r/m Byte and No s-i-b Byte Present |
|---|---|---|---|
| 00 | 000 | DS:[BX+SI] | DS:[EAX] |
| 00 | 001 | DS:[BX+DI] | DS:[ECX] |
| 00 | 010 | SS:[BP+SI] | DS:[EDX] |
| 00 | 011 | SS:[BP+DI] | DS:[EBX] |
| 00 | 100 | DS:[SI] | s-i-b is present (See Table 8-15) |
| 00 | 101 | DS:[DI] | DS:[d32] |
| 00 | 110 | DS:[d16] | DS:[ESI] |
| 00 | 111 | DS:[BX] | DS:[EDI] |
| | | | |
| 01 | 000 | DS:[BX+SI+d8] | DS:[EAX+d8] |
| 01 | 001 | DS:[BX+DI+d8] | DS:[ECX+d8] |
| 01 | 010 | SS:[BP+SI+d8] | DS:[EDX+d8] |
| 01 | 011 | SS:[BP+DI+d8] | DS:[EBX+d8] |
| 01 | 100 | DS:[SI+d8] | s-i-b is present (See Table 8-15) |
| 01 | 101 | DS:[DI+d8] | SS:[EBP+d8] |
| 01 | 110 | SS:[BP+d8] | DS:[ESI+d8] |
| 01 | 111 | DS:[BX+d8] | DS:[EDI+d8] |
| | | | |
| 10 | 000 | DS:[BX+SI+d16] | DS:[EAX+d32] |
| 10 | 001 | DS:[BX+DI+d16] | DS:[ECX+d32] |
| 10 | 010 | SS:[BP+SI+d16] | DS:[EDX+d32] |
| 10 | 011 | SS:[BP+DI+d16] | DS:[EBX+d32] |
| 10 | 100 | DS:[SI+d16] | s-i-b is present (See Table 8-15) |
| 10 | 101 | DS:[DI+d16] | SS:[EBP+d32] |
| 10 | 110 | SS:[BP+d16] | DS:[ESI+d32] |
| 10 | 111 | DS:[BX+d16] | DS:[EDI+d32] |
| | | | |
| 11 | xxx | See Table 8-9. | See Table 8-9 |

**Note:** d8 refers to 8-bit displacement, d16 refers to 16-bit displacement., and d32 refers to a 32-bit displacement.

**Table 8-9. General Registers Selected by mod r/m Fields and w Field**

| mod | r/m | 16-Bit Operation | | 32-Bit Operation | |
|---|---|---|---|---|---|
| | | w = 0 | w = 1 | w = 0 | w = 1 |
| 11 | 000 | AL | AX | AL | EAX |
| 11 | 001 | CL | CX | CL | ECX |
| 11 | 010 | DL | DX | DL | EDX |
| 11 | 011 | BL | BX | BL | EBX |
| 11 | 100 | AH | SP | AH | ESP |
| 11 | 101 | CH | BP | CH | EBP |
| 11 | 110 | DH | SI | DH | ESI |
| 11 | 111 | BH | DI | BH | EDI |

### 8.1.4 reg Field

The reg field (Table 8-10) determines which general registers are to be used. The selected register is dependent on whether a 16- or 32-bit operation is current and on the status of the w bit.

**Table 8-10. General Registers Selected by reg Field**

| reg | 16-Bit Operation | | 32-Bit Operation | |
|---|---|---|---|---|
| | w = 0 | w = 1 | w = 0 | w = 1 |
| 000 | AL | AX | AL | EAX |
| 001 | CL | CX | CL | ECX |
| 010 | DL | DX | DL | EDX |
| 011 | BL | BX | BL | EBX |
| 100 | AH | SP | AH | ESP |
| 101 | CH | BP | CH | EBP |
| 110 | DH | SI | DH | ESI |
| 111 | BH | DI | BH | EDI |

#### 8.1.4.1 sreg2 Field (ES, CS, SS, DS Register Selection)

The sreg2 field (Table 8-11) is a 2-bit field that allows one of the four 286-type segment registers to be specified.

**Table 8-11. sreg2 Field Encoding**

| sreg2 Field | Segment Register Selected |
|---|---|
| 00 | ES |
| 01 | CS |
| 10 | SS |
| 11 | DS |

#### 8.1.4.2 sreg3 Field (FS and GS Segment Register Selection)

The sreg3 field (Table 8-12) is 3-bit field that is similar to the sreg2 field, but allows use of the FS and GS segment registers.

**Table 8-12. sreg3 Field Encoding**

| sreg3 Field | Segment Register Selected |
|---|---|
| 000 | ES |
| 001 | CS |
| 010 | SS |
| 011 | DS |
| 100 | FS |
| 101 | GS |
| 110 | Undefined |
| 111 | Undefined |

### 8.1.5 s-i-b Byte (Scale, Indexing, Base)

The s-i-b fields provide scale factor, indexing and a base field for address selection. The ss, index and base fields are described next.

#### 8.1.5.1 ss Field (Scale Selection)

The ss field (Table 8-13) specifies the scale factor used in the offset mechanism for address calculation. The scale factor multiplies the index value to provide one of the components used to calculate the offset address.

**Table 8-13. ss Field Encoding**

| ss Field | Scale Factor |
|---|---|
| 00 | x1 |
| 01 | x2 |
| 01 | x4 |
| 11 | x8 |

#### 8.1.5.2 index Field (Index Selection)

The index field (Table 8-14) specifies the index register used by the offset mechanism for offset address calculation. When no index register is used (index field = 100), the ss value must be 00 or the effective address is undefined.

**Table 8-14. index Field Encoding**

| Index Field | Index Register |
|---|---|
| 000 | EAX |
| 001 | ECX |
| 010 | EDX |
| 011 | EBX |
| 100 | none |
| 101 | EBP |
| 110 | ESI |
| 111 | EDI |

#### 8.1.5.3 Base Field (s-i-b Present)

In Table 8-8, the note “s-i-b is present” for certain entries forces the use of the mod and base field as listed in Table 8-15. The first two digits in the first column of Table 8-15 identifies the mod bits in the mod r/m byte. The last three digits in the first column of this table identify the base fields in the s-i-b byte.

**Table 8-15. mod base Field Encoding**

| mod Field within mode/rm Byte (bits 7:6) | base Field within s-i-b Byte (bits 2:0) | 32-Bit Address Mode with mod r/m and s-i-b Bytes Present |
|---|---|---|
| 00 | 000 | DS:[EAX+(scaled index)] |
| 00 | 001 | DS:[ECX+(scaled index)] |
| 00 | 010 | DS:[EDX+(scaled index)] |
| 00 | 011 | DS:[EBX+(scaled index)] |
| 00 | 100 | SS:[ESP+(scaled index)] |
| 00 | 101 | DS:[d32+(scaled index)] |
| 00 | 110 | DS:[ESI+(scaled index)] |
| 00 | 111 | DS:[EDI+(scaled index)] |
| | | |
| 01 | 000 | DS:[EAX+(scaled index)+d8] |
| 01 | 001 | DS:[ECX+(scaled index)+d8] |
| 01 | 010 | DS:[EDX+(scaled index)+d8] |
| 01 | 011 | DS:[EBX+(scaled index)+d8] |
| 01 | 100 | SS:[ESP+(scaled index)+d8] |
| 01 | 101 | SS:[EBP+(scaled index)+d8] |
| 01 | 110 | DS:[ESI+(scaled index)+d8] |
| 01 | 111 | DS:[EDI+(scaled index)+d8] |
| | | |
| 10 | 000 | DS:[EAX+(scaled index)+d32] |
| 10 | 001 | DS:[ECX+(scaled index)+d32] |
| 10 | 010 | DS:[EDX+(scaled index)+d32] |
| 10 | 011 | DS:[EBX+(scaled index)+d32] |
| 10 | 100 | SS:[ESP+(scaled index)+d32] |
| 10 | 101 | SS:[EBP+(scaled index)+d32] |
| 10 | 110 | DS:[ESI+(scaled index)+d32] |
| 10 | 111 | DS:[EDI+(scaled index)+d32] |

## 8.2 CPUID INSTRUCTION

The CPUID instruction (opcode 0FA2) allows the software to make processor inquiries as to the vendor, family, model, stepping, features and also provides cache information. The GXLV supports both the standard and National Semiconductor extended CPUID levels.

The presence of the CPUID instruction is indicated by the ability to change the value of the ID Flag, bit 21 in the EFLAGS register.

The CPUID level allows the CPUID instruction to return different information in the EAX, EBX, ECX, and EDX registers. The level is determined by the initialized value of the EAX register before the instruction is executed. A summary of the CPUID levels is shown in Table 8-16.

**Table 8-16. CPUID Levels Summary**

| CPUID Type | Initialized EAX Register | Returned Data in EAX, EBX, ECX, EDX Registers |
|---|---|---|
| Standard | 0000 0000h | Maximum standard levels, CPU vendor string |
| Standard | 0000 0001h | Model, family, type and features |
| Standard | 0000 0002h | TLB and cache information |
| Extended | 8000 0000h | Maximum extended levels |
| Extended | 8000 0001h | Extended model, family, type and features |
| Extended | 8000 0002h | CPU marketing name string |
| Extended | 8000 0003h | |
| Extended | 8000 0004h | |
| Extended | 8000 0005h | TLB and L1 cache description |

### 8.2.1 Standard CPUID Levels

The standard CPUID levels are part of the standard x86 instruction set.

#### 8.2.1.1 CPUID Instruction with EAX = 0000 0000h

Standard function 0h (EAX = 0) of the CPUID instruction returns the maximum standard CPUID levels as well as the processor vendor string.

After the instruction is executed, the EAX register contains the maximum standard CPUID levels supported. The maximum standard CPUID level is the highest acceptable value for the EAX register input. This does not include the extended CPUID levels.

The EBX through EDX registers contain the vendor string of the processor as shown in Table 8-17.

**Table 8-17. CPUID Data Returned when EAX = 0**

| Register (Note) | Returned Contents | | | Description |
|---|---|---|---|---|
| EAX | 2 | | | Maximum Standard Level |
| EBX | 69 (iryC) | 72 | 7943 | Vendor ID String 1 |
| EDX | 73 (snlx) | 6E | 4978 | Vendor ID String 2 |
| ECX | 64 (daet) | 61 | 6574 | Vendor ID String 3 |

**Note:** The register column is intentionally out of order.

## Instruction Set (Continued)

#### 8.2.1.2 CPUID Instruction with EAX = 00000001h

Standard function 01h (EAX = 1) of the CPUID instruction returns the processor type, family, model, and stepping information of the current processor in the EAX register (see Table 8-18). The EBX and ECX registers are reserved.

**Table 8-18. EAX, EBX, ECX CPUID Data Returned when EAX = 1**

| Register | Returned Contents | Description |
|---|---|---|
| EAX[3:0] | xx | Stepping ID |
| EAX[7:4] | 4 | Model |
| EAX[11:8] | 5 | Family |
| EAX[15:12] | 0 | Type |
| EAX[31:16] | - | Reserved |
| EBX | - | Reserved |
| ECX | - | Reserved |

The standard feature flags supported are returned in the EDX register as shown in Table 8-19. Each flag refers to a specific feature and indicates if that feature is present on the processor. Some of these features have protection control in CR4. Before using any of these features on the processor, the software should check the corresponding feature flag. Attempting to execute an unavailable feature can cause exceptions and unexpected behavior. For example, software must check EDX bit 4 before attempting to use the Time Stamp Counter instruction.

**Table 8-19. EDX CPUID Data Returned when EAX = 1**

| EDX | Returned Contents* | Feature Flag | CR4 Bit |
|---|---|---|---|
| EDX[0] | 1 | FPU On-Chip | - |
| EDX[1] | 0 | Virtual Mode Extension | - |
| EDX[2] | 0 | Debug Extensions | - |
| EDX[3] | 0 | Page Size Extensions | - |
| EDX[4] | 1 | Time Stamp Counter | 2 |
| EDX[5] | 1 | RDMSR / WRMSR Instructions | - |
| EDX[6] | 0 | Physical Address Extensions | - |
| EDX[7] | 0 | Machine Check Exception | - |
| EDX[8] | 1 | CMPXCHG8B Instruction | - |
| EDX[9] | 0 | On-Chip APIC Hardware | - |
| EDX[10] | 0 | Reserved | - |
| EDX[11] | 0 | SYSENTER / SYSEXIT Instructions | - |
| EDX[12] | 0 | Memory Type Range Registers | - |
| EDX[13] | 0 | Page Global Enable | - |

**Table 8-19. EDX CPUID Data Returned when EAX = 1 (Continued)**

| EDX | Returned Contents* | Feature Flag | CR4 Bit |
|---|---|---|---|
| EDX[14] | 0 | Machine Check Architecture | - |
| EDX[15] | 1 | Conditional Move Instructions | - |
| EDX[16] | 0 | Page Attribute Table | - |
| EDX[22:17] | 0 | Reserved | - |
| EDX[23] | 1 | MMX Instructions | - |
| EDX[24] | 0 | Fast FPU Save and Restore | - |
| EDX[31:25] | 0 | Reserved | - |
| **Note:** *0 = Not Supported | | | |

#### 8.2.1.3 CPUID Instruction with EAX = 00000002h

Standard function 02h (EAX = 02h) of the CPUID instruction returns information that is specific to the National Semiconductor family of processors. Information about the TLB is returned in EAX as shown in Table 8-20. Information about the L1 cache is returned in EDX.

**Table 8-20. Standard CPUID with EAX = 00000002h**

| Register | Returned Contents | Description |
|---|---|---|
| EAX | xx xx 70 xxh | TLB is 32 entry, 4-way set associative, and has 4 KB pages. |
| EAX | xx xx xx 01h | The CPUID instruction needs to be executed only once with an input value of 02h to retrieve complete information about the cache and TLB. |
| EBX | | Reserved |
| ECX | | Reserved |
| EDX | xx xx xx 80h | L1 cache is 16 KB, 4-way set associated, and has 16 bytes per line. |

## Instruction Set (Continued)

### 8.2.2 Extended CPUID Levels

Testing for extended CPUID instruction support can be accomplished by executing a CPUID instruction with the EAX register initialized to 80000000h. If a value greater than or equal to 8000 0000h is returned to the EAX register by the CPUID instruction, the processor supports extended CPUID levels.

#### 8.2.2.1 CPUID Instruction with EAX = 80000000h

Extended function 8000 0000h (EAX = 80000000h) of the CPUID instruction returns the maximum extended CPUID levels supported by the current processor in EAX (Table 8-21). The EBX, ECX, and EDX registers are currently reserved.

**Table 8-21. Maximum Extended CPUID Level**

| Register | Returned Contents | Description |
|---|---|---|
| EAX | 80000005h | Maximum Extended CPUID Level (six levels) |
| EBX | - | Reserved |
| ECX | - | Reserved |
| EDX | - | Reserved |

#### 8.2.2.2 CPUID Instruction with EAX = 80000001h

Extended function 80000001h (EAX = 80000001h) of the CPUID instruction returns the processor type, family, model, and stepping information of the current processor in EAX. The EBX and ECX registers are reserved.

The extended feature flags supported are returned in the EDX register as shown in Table 8-23. Each flag refers to a specific feature and indicates if that feature is present on the processor. Some of these features have protection control in CR4. Before using any of these features on the processor, the software should check the corresponding feature flag.

**Table 8-22. EAX, EBX, ECX CPUID Data Returned when EAX = 80000001h**

| Register | Returned Contents | Description |
|---|---|---|
| EAX[3:0] | xx | Stepping ID |
| EAX[7:4] | 4 | Model |
| EAX[11:8] | 5 | Family |
| EAX[15:12] | 0 | Processor Type |
| EAX[31:16] | - | Reserved |
| EBX | - | Reserved |
| ECX | - | Reserved |

**Table 8-23. EDX CPUID Data Returned when EAX = 80000001h**

| EDX | Returned Contents* | Feature Flag | CR4 Bit |
|---|---|---|---|
| EDX[0] | 1 | FPU On-Chip | - |
| EDX[1] | 0 | Virtual Mode Extension | - |
| EDX[2] | 0 | Debugging Extension | - |
| EDX[3] | 0 | Page Size Extension (4 MB) | - |
| EDX[4] | 1 | Time Stamp Counter | 2 |
| EDX[5] | 1 | Model-Specific Registers (via RDMSR / WRMSR Instructions) | - |
| EDX[6] | 0 | Reserved | - |
| EDX[7] | 0 | Machine Check Exception | - |
| EDX[8] | 1 | CMPXCHG8B Instruction | - |
| EDX[9] | 0 | Reserved | - |
| EDX[10] | 0 | Reserved | - |
| EDX[11] | 0 | SYSCALL / SYSRET Instruction | - |
| EDX[12] | 0 | Reserved | - |
| EDX[13] | 0 | Page Global Enable | - |
| EDX[14] | 0 | Reserved | - |
| EDX[15] | 1 | Integer Conditional Move Instruction | - |
| EDX[16] | 0 | FPU Conditional Move Instruction | - |
| EDX[22:17] | 0 | Reserved | - |
| EDX[23] | 1 | MMX | - |
| EDX[24] | 1 | 6x86MX Multimedia Extensions | - |
| **Note:** 0 = Not supported | | | |

#### 8.2.2.3 CPUID Instruction with EAX = 80000002h, 80000003h, 80000004h

Extended functions 80000002h through 80000004h (EAX = 80000002h, EAX = 80000003h, EAX = 80000004h) of the CPUID instruction returns an ASCII string containing the name of the current processor. These functions eliminate the need to look up the processor name in a lookup table. Software can simply call these functions to obtain the name of the processor. The string may be 48 ASCII characters long, and is returned in little endian format. If the name is shorter than 48 characters long, the remaining bytes will be filled with ASCII NUL characters (00h).

**Table 8-24. Official CPU Name**

| 8000 0002h | | 8000 0003h | | 8000 0004h | |
|---|---|---|---|---|---|
| EAX | CPU Name 1 | EAX | CPU Name 5 | EAX | CPU Name 9 |
| EBX | CPU Name 2 | EBX | CPU Name 6 | EBX | CPU Name 10 |
| ECX | CPU Name 3 | ECX | CPU Name 7 | ECX | CPU Name 11 |
| EDX | CPU Name 4 | EDX | CPU Name 8 | EDX | CPU Name 12 |

#### 8.2.2.4 CPUID Instruction with EAX = 80000005h

Extended function 80000005h (EAX = 80000005h) of the CPUID instruction returns information about the TLB and L1 cache to be looked up in a lookup table. Refer to Table 8-25.

**Table 8-25. Standard CPUID with EAX = 80000005h**

| Register | Returned Contents | Description |
|---|---|---|
| EAX | -- | Reserved |
| EBX | xx xx 70 xxh | TLB is 32 entry, 4-way set associative, and has 4 KB Pages. |
| EBX | xx xx xx 01h | The CPUID instruction needs to be executed only once with an input value of 02h to retrieve complete information about the cache and TLB. |
| ECX | xx xx xx 80h | L1 cache is 16 KB, 4-way set associated, and has 16 bytes per line. |
| EDX | -- | Reserved |

# Instruction Set (Continued)

## 8.3 PROCESSOR CORE INSTRUCTION SET

The instruction set for the GXLV processor core is summarized in Table 8-27. The table uses several symbols and abbreviations that are described next and listed in Table 8-26.

### 8.3.1 Opcodes

Opcodes are given as hex values except when they appear within brackets as binary values.

### 8.3.2 Clock Counts

The clock counts listed in the instruction set summary table are grouped by operating mode (real and protected) and whether there is a register/cache hit or a cache miss. In some cases, more than one clock count is shown in a column for a given instruction, or a variable is used in the clock count.

### 8.3.3 Flags

There are nine flags that are affected by the execution of instructions. The flag names have been abbreviated and various conventions used to indicate what effect the instruction has on the particular flag.

**Table 8-26. Processor Core Instruction Set Table Legend**

| Symbol or Abbreviation | Description |
|---|---|
| **Opcode** | |
| # | Immediate 8-bit data. |
| ## | Immediate 16-bit data. |
| ### | Full immediate 32-bit data (8, 16, 32 bits). |
| + | 8-bit signed displacement. |
| +++ | Full signed displacement (16, 32 bits). |
| **Clock Count** | |
| / | Register operand/memory operand. |
| n | Number of times operation is repeated. |
| L | Level of the stack frame. |
| \| | Conditional jump taken \| Conditional jump not taken. (e.g. “4\|1” = 4 clocks if jump taken, 1 clock if jump not taken). |
| \ | CPL ≤ IOPL \ CPL > IOPL (where CPL = Current Privilege Level, IOPL = I/O Privilege Level). |
| **Flags** | |
| OF | Overflow Flag. |
| DF | Direction Flag. |
| IF | Interrupt Enable Flag. |
| TF | Trap Flag. |
| SF | Sign Flag. |
| ZF | Zero Flag. |
| AF | Auxiliary Flag. |
| PF | Parity Flag. |
| CF | Carry Flag. |
| x | Flag is modified by the instruction. |
| - | Flag is not changed by the instruction. |
| 0 | Flag is reset to “0”. |
| 1 | Flag is set to “1”. |
| u | Flag is undefined following execution the instruction. |

## Instruction Set (Continued)

### Table 8-27. Processor Core Instruction Set Summary

| Instruction | Opcode | Flags | | | | | | | | | Real Mode | Prot'd Mode | Real Mode | Prot'd Mode |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OF | DF | IF | TF | SF | ZF | AF | PF | CF | Clock Count (Reg/Cache Hit) | | Issues | |
| **AAA** *ASCII Adjust AL after Add* | 37 | u | - | - | - | u | u | x | u | x | 3 | 3 | | |
| **AAD** *ASCII Adjust AX before Divide* | D5 0A | u | - | - | - | x | x | u | x | u | 7 | 7 | | |
| **AAM** *ASCII Adjust AX after Multiply* | D4 0A | u | - | - | - | x | x | u | x | u | 19 | 19 | | |
| **AAS** *ASCII Adjust AL after Subtract* | 3F | u | - | - | - | u | u | x | u | x | 3 | 3 | | |
| **ADC** *Add with Carry* | | | | | | | | | | | | | | |
| Register to Register | 1 [00dw] [11 reg r/m] | x | - | - | - | x | x | x | x | x | 1 | 1 | b | h |
| Register to Memory | 1 [000w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Memory to Register | 1 [001w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Immediate to Register/Memory | 8 [00sw] [mod 010 r/m]### | | | | | | | | | | 1 | 1 | | |
| Immediate to Accumulator | 1 [010w] ### | | | | | | | | | | 1 | 1 | | |
| **ADD** *Integer Add* | | | | | | | | | | | | | | |
| Register to Register | 0 [00dw] [11 reg r/m] | x | - | - | - | x | x | x | x | x | 1 | 1 | b | h |
| Register to Memory | 0 [000w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Memory to Register | 0 [001w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Immediate to Register/Memory | 8 [00sw] [mod 000 r/m]### | | | | | | | | | | 1 | 1 | | |
| Immediate to Accumulator | 0 [010w] ### | | | | | | | | | | 1 | 1 | | |
| **AND** *Boolean AND* | | | | | | | | | | | | | | |
| Register to Register | 2 [00dw] [11 reg r/m] | 0 | - | - | - | x | x | u | x | 0 | 1 | 1 | b | h |
| Register to Memory | 2 [000w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Memory to Register | 2 [001w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Immediate to Register/Memory | 8 [00sw] [mod 100 r/m]### | | | | | | | | | | 1 | 1 | | |
| Immediate to Accumulator | 2 [010w] ### | | | | | | | | | | 1 | 1 | | |
| **ARPL** *Adjust Requested Privilege Level* | | | | | | | | | | | | | | |
| From Register/Memory | 63 [mod reg r/m] | - | - | - | - | - | x | - | - | - | | 9 | a | h |
| **BB0_Reset** *Set BLT Buffer 0 Pointer to the Base* | 0F 3A | | | | | | | | | | 2 | 2 | | |
| **BB1_Reset** *Set BLT Buffer 1 Pointer to the Base* | 0F 3B | | | | | | | | | | 2 | 2 | | |
| **BOUND** *Check Array Boundaries* | | | | | | | | | | | | | | |
| If Out of Range (Int 5) | 62 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 8+INT | 8+INT | b, e | g,h,j,k,r |
| If In Range | | | | | | | | | | | 7 | 7 | | |
| **BSF** *Scan Bit Forward* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F BC [mod reg r/m] | - | - | - | - | - | x | - | - | - | 4/9+n | 4/9+n | b | h |
| **BSR** *Scan Bit Reverse* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F BD [mod reg r/m] | - | - | - | - | - | x | - | - | - | 4/11+n | 4/11+n | b | h |
| **BSWAP** *Byte Swap* | 0F C[1 reg] | - | - | - | - | - | - | - | - | - | 6 | 6 | | |
| **BT** *Test Bit* | | | | | | | | | | | | | | |
| Register/Memory, Immediate | 0F BA [mod 100 r/m]# | - | - | - | - | - | - | - | - | x | 1 | 1 | b | h |
| Register/Memory, Register | 0F A3 [mod reg r/m] | | | | | | | | | | 1/7 | 1/7 | | |
| **BTC** *Test Bit and Complement* | | | | | | | | | | | | | | |
| Register/Memory, Immediate | 0F BA [mod 111 r/m]# | - | - | - | - | - | - | - | - | x | 2 | 2 | b | h |
| Register/Memory, Register | 0F BB [mod reg r/m] | | | | | | | | | | 2/8 | 2/8 | | |
| **BTR** *Test Bit and Reset* | | | | | | | | | | | | | | |
| Register/Memory, Immediate | 0F BA [mod 110 r/m]# | - | - | - | - | - | - | - | - | x | 2 | 2 | b | h |
| Register/Memory, Register | 0F B3 [mod reg r/m | | | | | | | | | | 2/8 | 2/8 | | |
| **BTS** *Test Bit and Set* | | | | | | | | | | | | | | |
| Register/Memory | 0F BA [mod 101 r/m] | - | - | - | - | - | - | - | - | x | 2 | 2 | b | h |
| Register (short form) | 0F AB [mod reg r/m] | | | | | | | | | | 2/8 | 2/8 | | |

## Instruction Set (Continued)

### Table 8-27. Processor Core Instruction Set Summary (Continued)

| Instruction | Opcode | Flags | | | | | | | | | Real Mode | Prot'd Mode | Real Mode | Prot'd Mode |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OF | DF | IF | TF | SF | ZF | AF | PF | CF | Clock Count (Reg/Cache Hit) | | Issues | |
| **CALL** *Subroutine Call* | | | | | | | | | | | | | | |
| Direct Within Segment | E8 +++ | - | - | - | - | - | - | - | - | - | 3 | 3 | b | h,j,k,r |
| Register/Memory Indirect Within Segment | FF [mod 010 r/m] | | | | | | | | | | 3/4 | 3/4 | | |
| Direct Intersegment<br>-Call Gate to Same Privilege<br>-Call Gate to Different Privilege No Par's<br>-Call Gate to Different Privilege m Par's<br>-16-bit Task to 16-bit TSS<br>-16-bit Task to 32-bit TSS<br>-16-bit Task to V86 Task<br>-32-bit Task to 16-bit TSS<br>-32-bit Task to 32-bit TSS<br>-32-bit Task to V86 Task | 9A [unsigned full offset, selector] | | | | | | | | | | 9 | 14<br>24<br>45<br>51+2m<br>183<br>189<br>123<br>186<br>192<br>126 | | |
| Indirect Intersegment<br>-Call Gate to Same Privilege<br>-Call Gate to Different Privilege No Par's<br>-Call Gate to Different Privilege m Par's<br>-16-bit Task to 16-bit TSS<br>-16-bit Task to 32-bit TSS<br>-16-bit Task to V86 Task<br>-32-bit Task to 16-bit TSS<br>-32-bit Task to 32-bit TSS<br>-32-bit Task to V86 Task | FF [mod 011 r/m] | | | | | | | | | | 11 | 15<br>25<br>46<br>52+2m<br>184<br>190<br>124<br>187<br>193<br>127 | | |
| **CBW** *Convert Byte to Word* | 98 | - | - | - | - | - | - | - | - | - | 3 | 3 | | |
| **CDQ** *Convert Doubleword to Quadword* | 99 | - | - | - | - | - | - | - | - | - | 2 | 2 | | |
| **CLC** *Clear Carry Flag* | F8 | - | - | - | - | - | - | - | - | 0 | 1 | 1 | | |
| **CLD** *Clear Direction Flag* | FC | - | 0 | - | - | - | - | - | - | - | 4 | 4 | | |
| **CLI** *Clear Interrupt Flag* | FA | - | - | 0 | - | - | - | - | - | - | 6 | 6 | | m |
| **CLTS** *Clear Task Switched Flag* | 0F 06 | - | - | - | - | - | - | - | - | - | 7 | 7 | c | l |
| **CMC** *Complement the Carry Flag* | F5 | - | - | - | - | - | - | - | - | x | 3 | 3 | | |
| **CMOVA/CMOVNBE** *Move if Above/Not Below or Equal* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 47 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMOVBE/CMOVNA** *Move if Below or Equal/Not Above* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 46 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMOVAE/CMOVNB/CMOVNC** *Move if Above or Equal/Not Below/Not Carry* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 43 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMOVB/CMOVC/CMOVNAE** *Move if Below/Carry/Not Above or Equal* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 42 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMOVE/CMOVZ** *Move if Equal/Zero* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 44 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMOVNE/CMOVNZ** *Move if Not Equal/Not Zero* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 45 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMOVG/CMOVNLE** *Move if Greater/Not Less or Equal* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 4F [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMOVLE/CMOVNG** *Move if Less or Equal/Not Greater* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 4E [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMOVL/CMOVNGE** *Move if Less/Not Greater or Equal* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 4C [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMOVGE/CMOVNL** *Move if Greater or Equal/Not Less* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 4D [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMOVO** *Move if Overflow* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 40 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMOVNO** *Move if No Overflow* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 41 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMOVP/CMOVPE** *Move if Parity/Parity Even* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 4A [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMOVNP/CMOVPO** *Move if Not Parity/Parity Odd* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 4B [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |

## Instruction Set (Continued)

### Table 8-27. Processor Core Instruction Set Summary (Continued)

| Instruction | Opcode | Flags | | | | | | | | | Real Mode | Prot'd Mode | Real Mode | Prot'd Mode |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OF | DF | IF | TF | SF | ZF | AF | PF | CF | Clock Count (Reg/Cache Hit) | | Issues | |
| **CMOVS** *Move if Sign* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 48 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMOVNS** *Move if Not Sign* | | | | | | | | | | | | | | |
| Register, Register/Memory | 0F 49 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| **CMP** *Compare Integers* | | | | | | | | | | | | | | |
| Register to Register | 3 [10dw] [11 reg r/m] | x | - | - | - | x | x | x | x | x | 1 | 1 | b | h |
| Register to Memory | 3 [101w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Memory to Register | 3 [100w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Immediate to Register/Memory | 8 [00sw] [mod 111 r/m] ### | | | | | | | | | | 1 | 1 | | |
| Immediate to Accumulator | 3 [110w] ### | | | | | | | | | | 1 | 1 | | |
| **CMPS** *Compare String* | A [011w] | x | - | - | - | x | x | x | x | x | 6 | 6 | b | h |
| **CMPXCHG** *Compare and Exchange* | | | | | | | | | | | | | | |
| Register1, Register2 | 0F B [000w] [11 reg2 reg1] | x | - | - | - | x | x | x | x | x | 6 | 6 | | |
| Memory, Register | 0F B [000w] [mod reg r/m] | | | | | | | | | | 6 | 6 | | |
| **CMPXCHG8B** *Compare and Exchange 8 Bytes* | 0F C7 [mod 001 r/m] | - | - | - | - | - | - | - | - | - | | | | |
| **CPUID** *CPU Identification* | 0F A2 | - | - | - | - | - | - | - | - | - | 12 | 12 | | |
| **CPU_READ** *Read Special CPU Register* | 0F 3C | | | | | | | | | | 1 | 1 | | |
| **CPU_WRITE** *Write Special CPU Register* | 0F 3D | | | | | | | | | | 1 | 1 | | |
| **CWD** *Convert Word to Doubleword* | 99 | - | - | - | - | - | - | - | - | - | 2 | 2 | | |
| **CWDE** *Convert Word to Doubleword Extended* | 98 | - | - | - | - | - | - | - | - | - | 3 | 3 | | |
| **DAA** *Decimal Adjust AL after Add* | 27 | - | - | - | - | x | x | x | x | x | 2 | 2 | | |
| **DAS** *Decimal Adjust AL after Subtract* | 2F | - | - | - | - | x | x | x | x | x | 2 | 2 | | |
| **DEC** *Decrement by 1* | | | | | | | | | | | | | | |
| Register/Memory | F [111w] [mod 001 r/m] | x | - | - | - | x | x | x | x | - | 1 | 1 | b | h |
| Register (short form) | 4 [1 reg] | | | | | | | | | | 1 | 1 | | |
| **DIV** *Unsigned Divide* | | | | | | | | | | | | | | |
| Accumulator by Register/Memory<br>Divisor: Byte<br>Word<br>Doubleword | F [011w] [mod 110 r/m] | - | - | - | - | x | x | u | u | - | <br>20<br>29<br>45 | <br>20<br>29<br>45 | b,e | e,h |
| **ENTER** *Enter New Stack Frame* | | | | | | | | | | | | | | |
| Level = 0 | C8 ##,# | - | - | - | - | - | - | - | - | - | 13 | 13 | b | h |
| Level = 1 | | | | | | | | | | | 17 | 17 | | |
| Level (L) > 1 | | | | | | | | | | | 17+2*L | 17+2*L | | |
| **HLT** *Halt* | F4 | - | - | - | - | - | - | - | - | - | 10 | 10 | | l |
| **IDIV** *Integer (Signed) Divide* | | | | | | | | | | | | | | |
| Accumulator by Register/Memory<br>Divisor: Byte<br>Word<br>Doubleword | F [011w] [mod 111 r/m] | - | - | - | - | x | x | u | u | - | <br>20<br>29<br>45 | <br>20<br>29<br>45 | b,e | e,h |
| **IMUL** *Integer (Signed) Multiply* | | | | | | | | | | | | | | |
| Accumulator by Register/Memory<br>Multiplier: Byte<br>Word<br>Doubleword | F [011w] [mod 101 r/m] | x | - | - | - | x | x | u | u | x | <br>4<br>5<br>15 | <br>4<br>5<br>15 | b | h |
| Register with Register/Memory<br>Multiplier: Word<br>Doubleword | 0F AF [mod reg r/m] | | | | | | | | | | <br>5<br>15 | <br>5<br>15 | | |
| Register/Memory with Immediate to Register2<br>Multiplier: Word<br>Doubleword | 6 [10s1] [mod reg r/m] ### | | | | | | | | | | <br>6<br>16 | <br>6<br>16 | | |
| **IN** *Input from I/O Port* | | | | | | | | | | | | | | |
| Fixed Port | E [010w] # | - | - | - | - | - | - | - | - | - | 8 | 8/22 | | m |
| Variable Port | E [110w] | | | | | | | | | | 8 | 8/22 | | |
| **INS** *Input String from I/O Port* | 6 [110w] | - | - | - | - | - | - | - | - | - | 11 | 11/25 | b | h,m |

## Instruction Set (Continued)

### Table 8-27. Processor Core Instruction Set Summary (Continued)

| Instruction | Opcode | Flags OF | DF | IF | TF | SF | ZF | AF | PF | CF | Real Mode Clock Count (Reg/Cache Hit) | Prot'd Mode Clock Count (Reg/Cache Hit) | Real Mode Issues | Prot'd Mode Issues |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **INC** *Increment by 1* | | | | | | | | | | | | | | |
| Register/Memory | F [111w] [mod 000 r/m] | x | - | - | - | x | x | x | x | - | 1 | 1 | b | h |
| Register (short form) | 4 [0 reg] | | | | | | | | | | 1 | 1 | | |
| **INT** *Software Interrupt* | | | | | | | | | | | | | | |
| INT i | CD # | - | - | x | 0 | - | - | - | - | - | 19 | | b,e | g,j,k,r |
| Protected Mode: | | | | | | | | | | | | | | |
| -Interrupt or Trap to Same Privilege | | | | | | | | | | | | 33 | | |
| -Interrupt or Trap to Different Privilege | | | | | | | | | | | | 55 | | |
| -16-bit Task to 16-bit TSS by Task Gate | | | | | | | | | | | | 184 | | |
| -16-bit Task to 32-bit TSS by Task Gate | | | | | | | | | | | | 190 | | |
| -16-bit Task to V86 by Task Gate | | | | | | | | | | | | 124 | | |
| -16-bit Task to 16-bit TSS by Task Gate | | | | | | | | | | | | 187 | | |
| -32-bit Task to 32-bit TSS by Task Gate | | | | | | | | | | | | 193 | | |
| -32-bit Task to V86 by Task Gate | | | | | | | | | | | | 127 | | |
| -V86 to 16-bit TSS by Task Gate | | | | | | | | | | | | 187 | | |
| -V86 to 32-bit TSS by Task Gate | | | | | | | | | | | | 193 | | |
| -V86 to Privilege 0 by Trap Gate/Int Gate | | | | | | | | | | | | 64 | | |
| INT 3 | CC | | | | | | | | | | INT | INT | | |
| INTO | CE | | | | | | | | | | | | | |
| If OF==0 | | | | | | | | | | | 4 | 4 | | |
| If OF==1 (INT 4) | | | | | | | | | | | INT | INT | | |
| **INVD** *Invalidate Cache* | 0F 08 | - | - | - | - | - | - | - | - | - | 20 | 20 | t | t |
| **INVLPG** *Invalidate TLB Entry* | 0F 01 [mod 111 r/m] | - | - | - | - | - | - | - | - | - | 15 | 15 | | |
| **IRET** *Interrupt Return* | | | | | | | | | | | | | | |
| Real Mode | CF | x | x | x | x | x | x | x | x | x | 13 | | | g,h,j,k,r |
| Protected Mode: | | | | | | | | | | | | | | |
| -Within Task to Same Privilege | | | | | | | | | | | | 20 | | |
| -Within Task to Different Privilege | | | | | | | | | | | | 39 | | |
| -16-bit Task to 16-bit Task | | | | | | | | | | | | 169 | | |
| -16-bit Task to 32-bit TSS | | | | | | | | | | | | 175 | | |
| -16-bit Task to V86 Task | | | | | | | | | | | | 109 | | |
| -32-bit Task to 16-bit TSS | | | | | | | | | | | | 172 | | |
| -32-bit Task to 32-bit TSS | | | | | | | | | | | | 178 | | |
| -32-bit Task to V86 Task | | | | | | | | | | | | 112 | | |
| **JB/JNAE/JC** *Jump on Below/Not Above or Equal/Carry* | | | | | | | | | | | | | | |
| 8-bit Displacement | 72 + | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| Full Displacement | 0F 82 +++ | | | | | | | | | | 1 | 1 | | |
| **JBE/JNA** *Jump on Below or Equal/Not Above* | | | | | | | | | | | | | | |
| 8-bit Displacement | 76 + | - | - | - | - | - | - | - | - | | 1 | 1 | | r |
| Full Displacement | 0F 86 +++ | | | | | | | | | | 1 | 1 | | |
| **JCXZ/JECXZ** *Jump on CX/ECX Zero* | E3 + | - | - | - | - | - | - | - | - | - | 2 | 2 | | r |
| **JE/JZ** *Jump on Equal/Zero* | | | | | | | | | | | | | | |
| 8-bit Displacement | 74 + | - | - | - | - | - | - | - | - | | 1 | 1 | | r |
| Full Displacement | 0F 84 +++ | | | | | | | | | | 1 | 1 | | |
| **JL/JNGE** *Jump on Less/Not Greater or Equal* | | | | | | | | | | | | | | |
| 8-bit Displacement | 7C + | - | - | - | - | - | - | - | - | | 1 | 1 | | r |
| Full Displacement | 0F 8C +++ | | | | | | | | | | 1 | 1 | | |
| **JLE/JNG** *Jump on Less or Equal/Not Greater* | | | | | | | | | | | | | | |
| 8-bit Displacement | 7E + | - | - | - | - | - | - | - | - | | 1 | 1 | | r |
| Full Displacement | 0F 8E +++ | | | | | | | | | | 1 | 1 | | |

## Instruction Set (Continued)

### Table 8-27. Processor Core Instruction Set Summary (Continued)

| Instruction | Opcode | Flags | | | | | | | | | Real Mode | Prot'd Mode | Real Mode | Prot'd Mode |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OF | DF | IF | TF | SF | ZF | AF | PF | CF | Clock Count (Reg/Cache Hit) | | Issues | |
| **JMP** *Unconditional Jump* | | | | | | | | | | | | | | |
| 8-bit Displacement | EB + | - | - | - | - | - | - | - | - | - | 1 | 1 | b | h,j,k,r |
| Full Displacement | E9 +++ | | | | | | | | | | 1 | 1 | | |
| Register/Memory Indirect Within Segment | FF [mod 100 r/m] | | | | | | | | | | 1/3 | 1/3 | | |
| Direct Intersegment<br>-Call Gate Same Privilege Level<br>-16-bit Task to 16-bit TSS<br>-16-bit Task to 32-bit TSS<br>-16-bit Task to V86 Task<br>-32-bit Task to 16-bit TSS<br>-32-bit Task to 32-bit TSS<br>-32-bit Task to V86 Task | EA [unsigned full offset, selector] | | | | | | | | | | 8 | 12<br>22<br>186<br>192<br>126<br>189<br>195<br>129 | | |
| Indirect Intersegment<br>-Call Gate Same Privilege Level<br>-16-bit Task to 16-bit TSS<br>-16-bit Task to 32-bit TSS<br>-16-bit Task to V86 Task<br>-32-bit Task to 16-bit TSS<br>-32-bit Task to 32-bit TSS<br>-32-bit Task to V86 Task | FF [mod 101 r/m] | | | | | | | | | | 10 | 13<br>23<br>187<br>193<br>127<br>190<br>196<br>130 | | |
| **JNB/JAE/JNC** *Jump on Not Below/Above or Equal/Not Carry* | | | | | | | | | | | | | | |
| 8-bit Displacement | 73 + | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| Full Displacement | 0F 83 +++ | | | | | | | | | | 1 | 1 | | |
| **JNBE/JA** *Jump on Not Below or Equal/Above* | | | | | | | | | | | | | | |
| 8-bit Displacement | 77 + | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| Full Displacement | 0F 87 +++ | | | | | | | | | | 1 | 1 | | |
| **JNE/JNZ** *Jump on Not Equal/Not Zero* | | | | | | | | | | | | | | |
| 8-bit Displacement | 75 + | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| Full Displacement | 0F 85 +++ | | | | | | | | | | 1 | 1 | | |
| **JNL/JGE** *Jump on Not Less/Greater or Equal* | | | | | | | | | | | | | | |
| 8-bit Displacement | 7D + | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| Full Displacement | 0F 8D +++ | | | | | | | | | | 1 | 1 | | |
| **JNLE/JG** *Jump on Not Less or Equal/Greater* | | | | | | | | | | | | | | |
| 8-bit Displacement | 7F + | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| Full Displacement | 0F 8F +++ | | | | | | | | | | 1 | 1 | | |
| **JNO** *Jump on Not Overflow* | | | | | | | | | | | | | | |
| 8-bit Displacement | 71 + | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| Full Displacement | 0F 81 +++ | | | | | | | | | | 1 | 1 | | |
| **JNP/JPO** *Jump on Not Parity/Parity Odd* | | | | | | | | | | | | | | |
| 8-bit Displacement | 7B + | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| Full Displacement | 0F 8B +++ | | | | | | | | | | 1 | 1 | | |
| **JNS** *Jump on Not Sign* | | | | | | | | | | | | | | |
| 8-bit Displacement | 79 + | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| Full Displacement | 0F 89 +++ | | | | | | | | | | 1 | 1 | | |
| **JO** *Jump on Overflow* | | | | | | | | | | | | | | |
| 8-bit Displacement | 70 + | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| Full Displacement | 0F 80 +++ | | | | | | | | | | 1 | 1 | | |
| **JP/JPE** *Jump on Parity/Parity Even* | | | | | | | | | | | | | | |
| 8-bit Displacement | 7A + | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| Full Displacement | 0F 8A +++ | | | | | | | | | | 1 | 1 | | |
| **JS** *Jump on Sign* | | | | | | | | | | | | | | |
| 8-bit Displacement | 78 + | - | - | - | - | - | - | - | - | - | 1 | 1 | | r |
| Full Displacement | 0F 88 +++ | | | | | | | | | | 1 | 1 | | |
| **LAHF** *Load AH with Flags* | 9F | - | - | - | - | - | - | - | - | - | 2 | 2 | | |
| **LAR** *Load Access Rights* | | | | | | | | | | | | | | |
| From Register/Memory | 0F 02 [mod reg r/m] | - | - | - | - | - | x | - | - | - | | 9 | a | g,h,j,p |
| **LDS** *Load Pointer to DS* | C5 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 4 | 9 | b | h,i,j |

## Instruction Set (Continued)

### Table 8-27. Processor Core Instruction Set Summary (Continued)

| Instruction | Opcode | Flags | | | | | | | | | Real Mode | Prot'd Mode | Real Mode | Prot'd Mode |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OF | DF | IF | TF | SF | ZF | AF | PF | CF | Clock Count (Reg/Cache Hit) | | Issues | |
| **LEA** *Load Effective Address* | | | | | | | | | | | | | | |
| No Index Register | 8D [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | |
| With Index Register | | | | | | | | | | | 1 | 1 | | |
| **LES** *Load Pointer to ES* | C4 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 4 | 9 | b | h,i,j |
| **LFS** *Load Pointer to FS* | 0F B4 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 4 | 9 | b | h,i,j |
| **LGDT** *Load GDT Register* | 0F 01 [mod 010 r/m] | - | - | - | - | - | - | - | - | - | 10 | 10 | b,c | h,l |
| **LGS** *Load Pointer to GS* | 0F B5 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 4 | 9 | b | h,i,j |
| **LIDT** *Load IDT Register* | 0F 01 [mod 011 r/m] | - | - | - | - | - | - | - | - | - | 10 | 10 | b,c | h,l |
| **LLDT** *Load LDT Register* | | | | | | | | | | | | | | |
| From Register/Memory | 0F 00 [mod 010 r/m] | - | - | - | - | - | - | - | - | - | | 8 | a | g,h,j,l |
| **LMSW** *Load Machine Status Word* | | | | | | | | | | | | | | |
| From Register/Memory | 0F 01 [mod 110 r/m] | - | - | - | - | - | - | - | - | - | 11 | 11 | b,c | h,l |
| **LODS** *Load String* | A [110 w] | - | - | - | - | - | - | - | - | - | 3 | 3 | b | h |
| **LSL** *Load Segment Limit* | | | | | | | | | | | | | | |
| From Register/Memory | 0F 03 [mod reg r/m] | - | - | - | - | - | x | - | - | - | | 9 | a | g,h,j,p |
| **LSS** *Load Pointer to SS* | 0F B2 [mod reg r/m] | - | - | - | - | - | - | - | - | - | 4 | 10 | a | h,i,j |
| **LTR** *Load Task Register* | | | | | | | | | | | | | | |
| From Register/Memory | 0F 00 [mod 011 r/m] | - | - | - | - | - | - | - | - | - | | 9 | a | g,h,j,l |
| **LEAVE** *Leave Current Stack Frame* | C9 | - | - | - | - | - | - | - | - | - | 4 | 4 | b | h |
| **LOOP** *Offset Loop/No Loop* | E2 + | - | - | - | - | - | - | - | - | - | 2 | 2 | | r |
| **LOOPNZ/LOOPNE** *Offset* | E0 + | - | - | - | - | - | - | - | - | - | 2 | 2 | | r |
| **LOOPZ/LOOPE** *Offset* | E1 + | - | - | - | - | - | - | - | - | - | 2 | 2 | | r |
| **MOV** *Move Data* | | | | | | | | | | | | | | |
| Register to Register | 8 [10dw] [11 reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | b | h,i,j |
| Register to Memory | 8 [100w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Register/Memory to Register | 8 [101w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Immediate to Register/Memory | C [011w] [mod 000 r/m] ### | | | | | | | | | | 1 | 1 | | |
| Immediate to Register (short form) | B [w reg] ### | | | | | | | | | | 1 | 1 | | |
| Memory to Accumulator (short form) | A [000w] +++ | | | | | | | | | | 1 | 1 | | |
| Accumulator to Memory (short form) | A [001w] +++ | | | | | | | | | | 1 | 1 | | |
| Register/Memory to Segment Register | 8E [mod sreg3 r/m] | | | | | | | | | | 1 | 6 | | |
| Segment Register to Register/Memory | 8C [mod sreg3 r/m] | | | | | | | | | | 1 | 1 | | |
| **MOV** *Move to/from Control/Debug/Test Regs* | | | | | | | | | | | | | | |
| Register to CR0/CR2/CR3/CR4 | 0F 22 [11 eee reg] | - | - | - | - | - | - | - | - | - | 20/5/5 | 18/5/6 | | l |
| CR0/CR2/CR3/CR4 to Register | 0F 20 [11 eee reg] | | | | | | | | | | 6 | 6 | | |
| Register to DR0-DR3 | 0F 23 [11 eee reg] | | | | | | | | | | 10 | 10 | | |
| DR0-DR3 to Register | 0F 21 [11 eee reg] | | | | | | | | | | 9 | 9 | | |
| Register to DR6-DR7 | 0F 23 [11 eee reg] | | | | | | | | | | 10 | 10 | | |
| DR6-DR7 to Register | 0F 21 [11 eee reg] | | | | | | | | | | 9 | 9 | | |
| Register to TR3-5 | 0F 26 [11 eee reg] | | | | | | | | | | 16 | 16 | | |
| TR3-5 to Register | 0F 24 [11 eee reg] | | | | | | | | | | 8 | 8 | | |
| Register to TR6-TR7 | 0F 26 [11 eee reg] | | | | | | | | | | 11 | 11 | | |
| TR6-TR7 to Register | 0F 24 [11 eee reg] | | | | | | | | | | 3 | 3 | | |
| **MOVS** *Move String* | A [010w] | - | - | - | - | - | - | - | - | - | 6 | 6 | b | h |
| **MOVSX** *Move with Sign Extension* | | | | | | | | | | | | | | |
| Register from Register/Memory | 0F B[111w] [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | b | h |
| **MOVZX** *Move with Zero Extension* | | | | | | | | | | | | | | |
| Register from Register/Memory | 0F B[011w] [mod reg r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | b | h |
| **MUL** *Unsigned Multiply* | | | | | | | | | | | | | | |
| Accumulator with Register/Memory | F [011w] [mod 100 r/m] | x | - | - | - | x | x | u | u | x | | | b | h |
| Multiplier: Byte | | | | | | | | | | | 4 | 4 | | |
| Word | | | | | | | | | | | 5 | 5 | | |
| Doubleword | | | | | | | | | | | 15 | 15 | | |

## Instruction Set (Continued)

### Table 8-27. Processor Core Instruction Set Summary (Continued)

| Instruction | Opcode | Flags | | | | | | | | | Real Mode | Prot'd Mode | Real Mode | Prot'd Mode |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OF | DF | IF | TF | SF | ZF | AF | PF | CF | Clock Count (Reg/Cache Hit) | | Issues | |
| **NEG** *Negate Integer* | F [011w] [mod 011 r/m] | x | - | - | - | x | x | x | x | x | 1 | 1 | b | h |
| **NOP** *No Operation* | 90 | - | - | - | - | - | - | - | - | - | 1 | 1 | | |
| **NOT** *Boolean Complement* | F [011w] [mod 010 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | b | h |
| **OIO** *Official Invalid Opcode* | 0F FF | - | - | x | 0 | - | - | - | - | - | 1 | 8-125 | | |
| **OR** *Boolean OR* | | | | | | | | | | | | | | |
| Register to Register | 0 [10dw] [11 reg r/m] | 0 | - | - | - | x | x | u | x | 0 | 1 | 1 | b | h |
| Register to Memory | 0 [100w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Memory to Register | 0 [101w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Immediate to Register/Memory | 8 [00sw] [mod 001 r/m] ### | | | | | | | | | | 1 | 1 | | |
| Immediate to Accumulator | 0 [110w] ### | | | | | | | | | | 1 | 1 | | |
| **OUT** *Output to Port* | | | | | | | | | | | | | | |
| Fixed Port | E [011w] # | - | - | - | - | - | - | - | - | - | 14 | 14/28 | | m |
| Variable Port | E [111w] | | | | | | | | | | 14 | 14/28 | | |
| **OUTS** *Output String* | 6 [111w] | - | - | - | - | - | - | - | - | - | 15 | 15/29 | b | h,m |
| **POP** *Pop Value off Stack* | | | | | | | | | | | | | | |
| Register/Memory | 8F [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1/4 | 1/4 | b | h,i,j |
| Register (short form) | 5 [1 reg] | | | | | | | | | | 1 | 1 | | |
| Segment Register (ES, SS, DS) | [000 sreg2 111] | | | | | | | | | | 1 | 6 | | |
| Segment Register (FS, GS) | 0F [10 sreg3 001] | | | | | | | | | | 1 | 6 | | |
| **POPA** *Pop All General Registers* | 61 | - | - | - | - | - | - | - | - | - | 9 | 9 | b | h |
| **POPF** *Pop Stack into FLAGS* | 9D | x | x | x | x | x | x | x | x | x | 8 | 8 | b | h,n |
| **PREFIX BYTES** | | | | | | | | | | | | | | |
| Assert Hardware LOCK Prefix | F0 | - | - | - | - | - | - | - | - | - | | | | m |
| Address Size Prefix | 67 | | | | | | | | | | | | | |
| Operand Size Prefix | 66 | | | | | | | | | | | | | |
| Segment Override Prefix<br>-CS<br>-DS<br>-ES<br>-FS<br>-GS<br>-SS | <br>2E<br>3E<br>26<br>64<br>65<br>36 | | | | | | | | | | | | | |
| **PUSH** *Push Value onto Stack* | | | | | | | | | | | | | | |
| Register/Memory | FF [mod 110 r/m] | - | - | - | - | - | - | - | - | - | 1/3 | 1/3 | b | h |
| Register (short form) | 5 [0 reg] | | | | | | | | | | 1 | 1 | | |
| Segment Register (ES, CS, SS, DS) | [000 sreg2 110] | | | | | | | | | | 1 | 1 | | |
| Segment Register (FS, GS) | 0F [10 sreg3 000] | | | | | | | | | | 1 | 1 | | |
| Immediate | 6 [10s0] ### | | | | | | | | | | 1 | 1 | | |
| **PUSHA** *Push All General Registers* | 60 | - | - | - | - | - | - | - | - | - | 11 | 11 | b | h |
| **PUSHF** *Push FLAGS Register* | 9C | - | - | - | - | - | - | - | - | - | 2 | 2 | b | h |
| **RCL** *Rotate Through Carry Left* | | | | | | | | | | | | | | |
| Register/Memory by 1 | D [000w] [mod 010 r/m] | x | - | - | - | - | - | - | - | x | 3 | 3 | b | h |
| Register/Memory by CL | D [001w] [mod 010 r/m] | u | - | - | - | - | - | - | - | x | 8 | 8 | | |
| Register/Memory by Immediate | C [000w] [mod 010 r/m] # | u | - | - | - | - | - | - | - | x | 8 | 8 | | |
| **RCR** *Rotate Through Carry Right* | | | | | | | | | | | | | | |
| Register/Memory by 1 | D [000w] [mod 011 r/m] | x | - | - | - | - | - | - | - | x | 4 | 4 | b | h |
| Register/Memory by CL | D [001w] [mod 011 r/m] | u | - | - | - | - | - | - | - | x | 8 | 8 | | |
| Register/Memory by Immediate | C [000w] [mod 011 r/m] # | u | - | - | - | - | - | - | - | x | 8 | 8 | | |
| **RDMSR** *Read Tmodel Specific Register* | 0F 32 | - | - | - | - | - | - | - | - | - | | | | |
| **RDTSC** *Read Time Stamp Counter* | 0F 31 | - | - | - | - | - | - | - | - | - | | | | |
| **REP INS** *Input String* | F3 6[110w] | - | - | - | - | - | - | - | - | - | 17+4n | 17+4n\<br>32+4n | b | h,m |
| **REP LODS** *Load String* | F3 A[110w] | - | - | - | - | - | - | - | - | - | 9+2n | 9+2n | b | h |
| **REP MOVS** *Move String* | F3 A[010w] | - | - | - | - | - | - | - | - | - | 12+2n | 12+2n | b | h |

## Instruction Set (Continued)

### Table 8-27. Processor Core Instruction Set Summary (Continued)

| Instruction | Opcode | Flags | | | | | | | | | Real Mode | Prot'd Mode | Real Mode | Prot'd Mode |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OF | DF | IF | TF | SF | ZF | AF | PF | CF | Clock Count (Reg/Cache Hit) | | Issues | |
| **REP OUTS** *Output String* | F3 6[111w] | - | - | - | - | - | - | - | - | - | 24+4n | 24+4n\ 39+4n | b | h,m |
| **REP STOS** *Store String* | F3 A[101w] | - | - | - | - | - | - | - | - | - | 9+2n | 9+2n | b | h |
| **REPE CMPS** *Compare String* | | | | | | | | | | | | | | |
| Find non-match | F3 A[011w] | x | - | - | - | x | x | x | x | x | 11+4n | 11+4n | b | h |
| **REPE SCAS** *Scan String* | | | | | | | | | | | | | | |
| Find non-AL/AX/EAX | F3 A[111w] | x | - | - | - | x | x | x | x | x | 9+3n | 9+3n | b | h |
| **REPNE CMPS** *Compare String* | | | | | | | | | | | | | | |
| Find match | F2 A[011w] | x | - | - | - | x | x | x | x | x | 11+4n | 11+4n | b | h |
| **REPNE SCAS** *Scan String* | | | | | | | | | | | | | | |
| Find AL/AX/EAX | F2 A[111w] | x | - | - | - | x | x | x | x | x | 9+3n | 9+3n | b | h |
| **RET** *Return from Subroutine* | | | | | | | | | | | | | | |
| Within Segment | C3 | - | - | - | - | - | - | - | - | - | 3 | 3 | b | g,h,j,k,r |
| Within Segment Adding Immediate to SP | C2 ## | | | | | | | | | | 3 | 3 | | |
| Intersegment | CB | | | | | | | | | | 10 | 13 | | |
| Intersegment Adding Immediate to SP | CA ## | | | | | | | | | | 10 | 13 | | |
| Protected Mode: Different Privilege Level<br>-Intersegment<br>-Intersegment Adding Immediate to SP | | | | | | | | | | | | <br>35<br>35 | | |
| **ROL** *Rotate Left* | | | | | | | | | | | | | | |
| Register/Memory by 1 | D[000w] [mod 000 r/m] | x | - | - | - | - | - | - | - | x | 2 | 2 | b | h |
| Register/Memory by CL | D[001w] [mod 000 r/m] | u | - | - | - | - | - | - | - | x | 2 | 2 | | |
| Register/Memory by Immediate | C[000w] [mod 000 r/m] # | u | - | - | - | - | - | - | - | x | 2 | 2 | | |
| **ROR** *Rotate Right* | | | | | | | | | | | | | | |
| Register/Memory by 1 | D[000w] [mod 001 r/m] | x | - | - | - | - | - | - | - | x | 2 | 2 | b | h |
| Register/Memory by CL | D[001w] [mod 001 r/m] | u | - | - | - | - | - | - | - | x | 2 | 2 | | |
| Register/Memory by Immediate | C[000w] [mod 001 r/m] # | u | - | - | - | - | - | - | - | x | 2 | 2 | | |
| **RSDC** *Restore Segment Register and Descriptor* | 0F 79 [mod sreg3 r/m] | - | - | - | - | - | - | - | - | - | 11 | 11 | s | s |
| **RSLDT** *Restore LDTR and Descriptor* | 0F 7B [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 11 | 11 | s | s |
| **RSTS** *Restore TSR and Descriptor* | 0F 7D [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 11 | 11 | s | s |
| **RSM** *Resume from SMM Mode* | 0F AA | x | x | x | x | x | x | x | x | x | 57 | 57 | s | s |
| **SAHF** *Store AH in FLAGS* | 9E | - | - | - | - | x | x | x | x | x | 1 | 1 | | |
| **SAL** *Shift Left Arithmetic* | | | | | | | | | | | | | | |
| Register/Memory by 1 | D[000w] [mod 100 r/m] | x | - | - | - | x | x | u | x | x | 1 | 1 | b | h |
| Register/Memory by CL | D[001w] [mod 100 r/m] | u | - | - | - | x | x | u | x | x | 2 | 2 | | |
| Register/Memory by Immediate | C[000w] [mod 100 r/m] # | u | - | - | - | x | x | u | x | x | 1 | 1 | | |
| **SAR** *Shift Right Arithmetic* | | | | | | | | | | | | | | |
| Register/Memory by 1 | D[000w] [mod 111 r/m] | x | - | - | - | x | x | u | x | x | 2 | 2 | b | h |
| Register/Memory by CL | D[001w] [mod 111 r/m] | u | - | - | - | x | x | u | x | x | 2 | 2 | | |
| Register/Memory by Immediate | C[000w] [mod 111 r/m] # | u | - | - | - | x | x | u | x | x | 2 | 2 | | |
| **SBB** *Integer Subtract with Borrow* | | | | | | | | | | | | | | |
| Register to Register | 1[10dw] [11 reg r/m] | x | - | - | - | x | x | x | x | x | 1 | 1 | b | h |
| Register to Memory | 1[100w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Memory to Register | 1[101w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Immediate to Register/Memory | 8[00sw] [mod 011 r/m] ### | | | | | | | | | | 1 | 1 | | |
| Immediate to Accumulator (short form) | 1[110w] ### | | | | | | | | | | 1 | 1 | | |
| **SCAS** *Scan String* | A [111w] | x | - | - | - | x | x | x | x | x | 2 | 2 | b | h |
| **SETB/SETNAE/SETC** *Set Byte on Below/Not Above or Equal/Carry* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 92 [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SETBE/SETNA** *Set Byte on Below or Equal/Not Above* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 96 [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SETE/SETZ** *Set Byte on Equal/Zero* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 94 [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |

## Instruction Set (Continued)

### Table 8-27. Processor Core Instruction Set Summary (Continued)

| Instruction | Opcode | Flags | | | | | | | | | Real Mode | Prot'd Mode | Real Mode | Prot'd Mode |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OF | DF | IF | TF | SF | ZF | AF | PF | CF | Clock Count (Reg/Cache Hit) | | Issues | |
| **SETL/SETNGE** *Set Byte on Less/Not Greater or Equal* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 9C [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SETLE/SETNG** *Set Byte on Less or Equal/Not Greater* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 9E [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SETNB/SETAE/SETNC** *Set Byte on Not Below/Above or Equal/Not Carry* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 93 [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SETNBE/SETA** *Set Byte on Not Below or Equal/Above* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 97 [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SETNE/SETNZ** *Set Byte on Not Equal/Not Zero* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 95 [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SETNL/SETGE** *Set Byte on Not Less/Greater or Equal* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 9D [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SETNLE/SETG** *Set Byte on Not Less or Equal/Greater* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 9F [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SETNO** *Set Byte on Not Overflow* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 91 [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SETNP/SETPO** *Set Byte on Not Parity/Parity Odd* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 9B [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SETNS** *Set Byte on Not Sign* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 99 [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SETO** *Set Byte on Overflow* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 90 [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SETP/SETPE** *Set Byte on Parity/Parity Even* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 9A [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SETS** *Set Byte on Sign* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 98 [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | | h |
| **SGDT** *Store GDT Register* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 01 [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 6 | 6 | b,c | h |
| **SIDT** *Store IDT Register* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 01 [mod 001 r/m] | - | - | - | - | - | - | - | - | - | 6 | 6 | b,c | h |
| **SLDT** *Store LDT Register* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 00 [mod 000 r/m] | - | - | - | - | - | - | - | - | - | | 1 | a | h |
| **STR** *Store Task Register* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 00 [mod 001 r/m] | - | - | - | - | - | - | - | - | - | | 3 | a | h |
| **SMSW** *Store Machine Status Word* | 0F 01 [mod 100 r/m] | - | - | - | - | - | - | - | - | - | 4 | 4 | b,c | h |
| **STOS** *Store String* | A [101w] | - | - | - | - | - | - | - | - | - | 2 | 2 | b | h |
| **SHL** *Shift Left Logical* | | | | | | | | | | | | | | |
| Register/Memory by 1 | D [000w] [mod 100 r/m] | x | - | - | - | x | x | u | x | x | 1 | 1 | b | h |
| Register/Memory by CL | D [001w] [mod 100 r/m] | u | - | - | - | x | x | u | x | x | 2 | 2 | | |
| Register/Memory by Immediate | C [000w] [mod 100 r/m] # | u | - | - | - | x | x | u | x | x | 1 | 1 | | |
| **SHLD** *Shift Left Double* | | | | | | | | | | | | | | |
| Register/Memory by Immediate | 0F A4 [mod reg r/m] # | u | - | - | - | x | x | u | x | x | 3 | 3 | b | h |
| Register/Memory by CL | 0F A5 [mod reg r/m] | | | | | | | | | | 6 | 6 | | |
| **SHR** *Shift Right Logical* | | | | | | | | | | | | | | |
| Register/Memory by 1 | D [000w] [mod 101 r/m] | x | - | - | - | x | x | u | x | x | 2 | 2 | b | h |
| Register/Memory by CL | D [001w] [mod 101 r/m] | u | - | - | - | x | x | u | x | x | 2 | 2 | | |
| Register/Memory by Immediate | C [000w] [mod 101 r/m] # | u | - | - | - | x | x | u | x | x | 2 | 2 | | |
| **SHRD** *Shift Right Double* | | | | | | | | | | | | | | |
| Register/Memory by Immediate | 0F AC [mod reg r/m] # | u | - | - | - | x | x | u | x | x | 3 | 3 | b | h |
| Register/Memory by CL | 0F AD [mod reg r/m] | | | | | | | | | | 6 | 6 | | |
| **SMINT** *Software SMM Entry* | 0F 38 | - | - | - | - | - | - | - | - | - | 84 | 84 | s | s |
| **STC** *Set Carry Flag* | F9 | - | - | - | - | - | - | - | - | 1 | 1 | 1 | | |

## Instruction Set (Continued)

### Table 8-27. Processor Core Instruction Set Summary (Continued)

| Instruction | Opcode | Flags | | | | | | | | | Real Mode | Prot'd Mode | Real Mode | Prot'd Mode |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OF | DF | IF | TF | SF | ZF | AF | PF | CF | Clock Count (Reg/Cache Hit) | | Issues | |
| **STD** *Set Direction Flag* | FD | - | 1 | - | - | - | - | - | - | - | 4 | 4 | | |
| **STI** *Set Interrupt Flag* | FB | - | - | 1 | - | - | - | - | - | - | 6 | 6 | | m |
| **SUB** *Integer Subtract* | | | | | | | | | | | | | | |
| Register to Register | 2 [10dw] [11 reg r/m] | x | - | - | - | x | x | x | x | x | 1 | 1 | b | h |
| Register to Memory | 2 [100w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Memory to Register | 2 [101w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Immediate to Register/Memory | 8 [00sw] [mod 101 r/m] ### | | | | | | | | | | 1 | 1 | | |
| Immediate to Accumulator (short form) | 2 [110w] ### | | | | | | | | | | 1 | 1 | | |
| **SVDC** *Save Segment Register and Descriptor* | 0F 78 [mod sreg3 r/m] | - | - | - | - | - | - | - | - | - | 20 | 20 | s | s |
| **SVLDT** *Save LDTR and Descriptor* | 0F 7A [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 20 | 20 | s | s |
| **SVTS** *Save TSR and Descriptor* | 0F 7C [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 21 | 21 | s | s |
| **TEST** *Test Bits* | | | | | | | | | | | | | | |
| Register/Memory and Register | 8 [010w] [mod reg r/m] | 0 | - | - | - | x | x | u | x | 0 | 1 | 1 | b | h |
| Immediate Data and Register/Memory | F [011w] [mod 000 r/m] ### | | | | | | | | | | 1 | 1 | | |
| Immediate Data and Accumulator | A [100w] ### | | | | | | | | | | 1 | 1 | | |
| **VERR** *Verify Read Access* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 00 [mod 100 r/m] | - | - | - | - | - | x | - | - | - | | 8 | a | g,h,j,p |
| **VERW** *Verify Write Access* | | | | | | | | | | | | | | |
| To Register/Memory | 0F 00 [mod 101 r/m] | - | - | - | - | - | x | - | - | - | | 8 | a | g,h,j,p |
| **WAIT** *Wait Until FPU Not Busy* | 9B | - | - | - | - | - | - | - | - | - | 1 | 1 | | |
| **WBINVD** *Write-Back and Invalidate Cache* | 0F 09 | - | - | - | - | - | - | - | - | - | 23 | 23 | t | t |
| **WRMSR** *Write to Model Specific Register* | 0F 30 | - | - | - | - | - | - | - | - | - | | | | |
| **XADD** *Exchange and Add* | | | | | | | | | | | | | | |
| Register1, Register2 | 0F C[000w] [11 reg2 reg1] | x | - | - | - | x | x | x | x | x | 2 | 2 | | |
| Memory, Register | 0F C[000w] [mod reg r/m] | | | | | | | | | | 2 | 2 | | |
| **XCHG** *Exchange* | | | | | | | | | | | | | | |
| Register/Memory with Register | 8[011w] [mod reg r/m] | - | - | - | - | - | - | - | - | - | 2 | 2 | b,f | f,h |
| Register with Accumulator | 9[0 reg] | | | | | | | | | | 2 | 2 | | |
| **XLAT** *Translate Byte* | D7 | - | - | - | - | - | - | - | - | - | 5 | 5 | | h |
| **XOR** *Boolean Exclusive OR* | | | | | | | | | | | | | | |
| Register to Register | 3 [00dw] [11 reg r/m] | 0 | - | - | - | x | x | u | x | 0 | 1 | 1 | b | h |
| Register to Memory | 3 [000w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Memory to Register | 3 [001w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Immediate to Register/Memory | 8 [00sw] [mod 110 r/m] ### | | | | | | | | | | 1 | 1 | | |
| Immediate to Accumulator (short form) | 3 [010w] ### | | | | | | | | | | 1 | 1 | | |

## Instruction Set (Continued)

**Instruction Issues for Instruction Set Summary**

**Issues a through c apply to real address mode only:**

a. This is a protected mode instruction. Attempted execution in real mode will result in exception 6 (invalid opcode).

b. Exception 13 fault (general protection) will occur in real mode if an operand reference is made that partially or fully extends beyond the maximum CS, DS, ES, FS, or GS segment limit (FFFFH). Exception 12 fault (stack segment limit violation or not present) will occur in real mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.

c. This instruction may be executed in real mode. In real mode, its purpose is primarily to initialize the CPU for protected mode.

d. -

**Issues e through g apply to real address mode and protected virtual address mode:**

e. An exception may occur, depending on the value of the operand.

f. LOCK# is automatically asserted, regardless of the presence or absence of the LOCK prefix.

g. LOCK# is asserted during descriptor table accesses.

**Issues h through r apply to protected virtual address mode only:**

h. Exception 13 fault will occur if the memory operand in CS, DS, ES, FS, or GS cannot be used due to either a segment limit violation or an access rights violation. If a stack limit is violated, an exception 12 occurs.

i. For segment load operations, the CPL, RPL, and DPL must agree with the privilege rules to avoid an exception 13 fault. The segment's descriptor must indicate "present" or exception 11 (CS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an exception 12 occurs.

j. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert LOCK# to maintain descriptor integrity in multiprocessor systems.

k. JMP, CALL, INT, RET, and IRET instructions referring to another code segment will cause an exception 13, if an applicable privilege rule is violated.

l. An exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).

m. An exception 13 fault occurs if CPL is greater than IOPL.

n. The IF bit of the Flags register is not updated if CPL is greater than IOPL. The IOPL and VM fields of the Flags register are updated only if CPL = 0.

o. The PE bit of the MSW (CR0) cannot be reset by this instruction. Use MOV into CR0 if desiring to reset the PE bit.

p. Any violation of privilege rules as apply to the selector operand does not cause a Protection exception, rather, the zero flag is cleared.

q. If the processor's memory operand violates a segment limit or segment access rights, an exception 13 fault will occur before the ESC instruction is executed. An exception 12 fault will occur if the stack limit is violated by the operand's starting address.

r. The destination of a JMP, CALL, INT, RET, or IRET must be in the defined limit of a code segment or an exception 13 fault will occur.

**Issue s applies to National Semiconductor-specific SMM instructions:**

s. All memory accesses to SMM space are non-cacheable. An invalid opcode exception 6 occurs unless SMI is enabled and SMAR size > 0, and CPL = 0 and [SMAC is set or if in an SMI handler].

**Issue t applies to cache invalidation instruction with the cache operating in write-back mode:**

t. The total clock count is the clock count shown plus the number of clocks required to write all "modified" cache lines to external memory.

## 8.4 FPU INSTRUCTION SET

The processor core is functionally divided into the FPU, and the integer unit. The FPU processes floating point instructions only and does so in parallel with the integer unit.

For example, when the integer unit detects a floating point instruction without memory operands, after two clock cycles the instruction passes to the FPU for execution. The integer unit continues to execute instructions while the FPU executes the floating point instruction. If another FPU instruction is encountered, the second FPU instruction is placed in the FPU queue. Up to four FPU instructions can be queued. In the event of an FPU exception, while other FPU instructions are queued, the state of the CPU is saved to ensure recovery.

The FPU instruction set is summarized in Table 8-29. The table uses abbreviations that are described Table 8-28.

**Table 8-28. FPU Instruction Set Table Legend**

| Abbr. | Description |
|---|---|
| n | Stack register number. |
| TOS | Top of stack register pointed to by SSS in the status register. |
| ST(1) | FPU register next to TOS. |
| ST(n) | A specific FPU register, relative to TOS. |
| M.WI | 16-bit integer operand from memory. |
| M.SI | 32-bit integer operand from memory. |
| M.LI | 64-bit integer operand from memory. |
| M.SR | 32-bit real operand from memory. |
| M.DR | 64-bit real operand from memory. |
| M.XR | 80-bit real operand from memory. |
| M.BCD | 18-digit BCD integer operand from memory. |
| CC | FPU condition code. |
| Env Regs | Status, Mode Control and Tag Registers, Instruction Pointer and Operand Pointer. |

## Instruction Set (Continued)

### Table 8-29. FPU Instruction Set Summary

| FPU Instruction | Opcode | Operation | Clock Count | Issue |
|---|---|---|---|---|
| **F2XM1** *Function Evaluation $2^x$-1* | D9 F0 | TOS <--- $2^{TOS}$-1 | 92 - 108 | 2 |
| **FABS** *Floating Absolute Value* | D9 E1 | TOS <--- \| TOS \| | 2 | 2 |
| **FADD** *Floating Point Add* | | | | |
| Top of Stack | DC [1100 0 n] | ST(n) <--- ST(n) + TOS | 4 - 9 | |
| 80-bit Register | D8 [1100 0 n] | TOS <--- TOS + ST(n) | 4 - 9 | |
| 64-bit Real | DC [mod 000 r/m] | TOS <--- TOS + M.DR | 4 - 9 | |
| 32-bit Real | D8 [mod 000 r/m] | TOS <--- TOS + M.SR | 4 - 9 | |
| **FADDP** *Floating Point Add, Pop* | DE [1100 0 n] | ST(n) <--- ST(n) + TOS; then pop TOS | | |
| **FIADD** *Floating Point Integer Add* | | | | |
| 32-bit integer | DA [mod 000 r/m] | TOS <--- TOS + M.SI | 8 - 14 | |
| 16-bit integer | DE [mod 000 r/m] | TOS <--- TOS + M.WI | 8 - 14 | |
| **FCHS** *Floating Change Sign* | D9 E0 | TOS <--- - TOS | 2 | |
| **FCLEX** *Clear Exceptions* | (9B) DB E2 | Wait then Clear Exceptions | 5 | |
| **FNCLEX** *Clear Exceptions* | DB E2 | Clear Exceptions | 3 | |
| **FCMOVB** *Floating Point Conditional Move if Below* | DA [1100 0 n] | If (CF=1) ST(0) <--- ST(n) | 4 | |
| **FCMOVE** *Floating Point Conditional Move if Equal* | DA [1100 1 n] | If (ZF=1) ST(0) <--- ST(n) | 4 | |
| **FCMOVBE** *Floating Point Conditional Move if Below or Equal* | DA [1101 0 n] | If (CF=1 or ZF=1) ST(0) <--- ST(n) | 4 | |
| **FCMOVU** *Floating Point Conditional Move if Unordered* | DA [1101 1 n] | If (PF=1) ST(0) <--- ST(n) | 4 | |
| **FCMOVNB** *Floating Point Conditional Move if Not Below* | DB [1100 0 n] | If (CF=0) ST(0) <--- ST(n) | 4 | |
| **FCMOVNE** *Floating Point Conditional Move if Not Equal* | DB [1100 1 n] | If (ZF=0) ST(0) <--- ST(n) | 4 | |
| **FCMOVNBE** *Floating Point Conditional Move if Not Below or Equal* | DB [1101 0 n] | If (CF=0 and ZF=0) ST(0) <--- ST(n) | 4 | |
| **FCMOVNU** *Floating Point Conditional Move if Not Unordered* | DB [1101 1 n] | If (DF=0) ST(0) <--- ST(n) | 4 | |
| **FCOM** *Floating Point Compare* | | | | |
| 80-bit Register | D8 [1101 0 n] | CC set by TOS - ST(n) | 4 | |
| 64-bit Real | DC [mod 010 r/m] | CC set by TOS - M.DR | 4 | |
| 32-bit Real | D8 [mod 010 r/m] | CC set by TOS - M.SR | 4 | |
| **FCOMP** *Floating Point Compare, Pop* | | | | |
| 80-bit Register | D8 [1101 1 n] | CC set by TOS - ST(n); then pop TOS | 4 | |
| 64-bit Real | DC [mod 011 r/m] | CC set by TOS - M.DR; then pop TOS | 4 | |
| 32-bit Real | D8 [mod 011 r/m] | CC set by TOS - M.SR; then pop TOS | 4 | |
| **FCOMPP** *Floating Point Compare, Pop Two Stack Elements* | DE D9 | CC set by TOS - ST(1); then pop TOS and ST(1) | 4 | |
| **FCOMI** *Floating Point Compare Real and Set EFLAGS* | | | | |
| 80-bit Register | DB [1111 0 n] | EFLAG set by TOS - ST(n) | 4 | |
| **FCOMIP** *Floating Point Compare Real and Set EFLAGS, Pop* | | | | |
| 80-bit Register | DF [1111 0 n] | EFLAG set by TOS - ST(n); then pop TOS | 4 | |
| **FUCOMI** *Floating Point Unordered Compare Real and Set EFLAGS* | | | | |
| 80-bit Integer | DB [1110 1 n] | EFLAG set by TOS - ST(n) | 9 - 10 | |
| **FUCOMIP** *Floating Point Unordered Compare Real and Set EFLAGS, Pop* | | | | |
| 80-bit Integer | DF [1110 1 n] | EFLAG set by TOS - ST(n); then pop TOS | 9 - 10 | |
| **FICOM** *Floating Point Integer Compare* | | | | |
| 32-bit integer | DA [mod 010 r/m] | CC set by TOS - M.WI | 9 - 10 | |
| 16-bit integer | DE [mod 010 r/m] | CC set by TOS - M.SI | 9 - 10 | |
| **FICOMP** *Floating Point Integer Compare, Pop* | | | | |
| 32-bit integer | DA [mod 011 r/m] | CC set by TOS - M.WI; then pop TOS | 9 - 10 | |
| 16-bit integer | DE [mod 011 r/m | CC set by TOS - M.SI; then pop TOS | 9 - 10 | |

## Instruction Set (Continued)

### Table 8-29. FPU Instruction Set Summary (Continued)

| FPU Instruction | Opcode | Operation | Clock Count | Issue |
|---|---|---|---|---|
| **FCOS** *Function Evaluation: Cos(x)* | D9 FF | TOS <--- COS(TOS) | 92 - 141 | 1 |
| **FDECSTP** *Decrement Stack pointer* | D9 F6 | Decrement top of stack pointer | 4 | |
| **FDIV** *Floating Point Divide* | | | | |
| Top of Stack | DC [1111 1 n] | ST(n) <--- ST(n) / TOS | 24 - 34 | |
| 80-bit Register | D8 [1111 0 n] | TOS <--- TOS / ST(n) | 24 - 34 | |
| 64-bit Real | DC [mod 110 r/m] | TOS <--- TOS / M.DR | 24 - 34 | |
| 32-bit Real | D8 [mod 110 r/m] | TOS <--- TOS / M.SR | 24 - 34 | |
| **FDIVP** *Floating Point Divide, Pop* | DE [1111 1 n] | ST(n) <--- ST(n) / TOS; then pop TOS | 24 - 34 | |
| **FDIVR** *Floating Point Divide Reversed* | | | | |
| Top of Stack | DC [1111 0 n] | TOS <--- ST(n) / TOS | 24 - 34 | |
| 80-bit Register | D8 [1111 1 n] | ST(n) <--- TOS / ST(n) | 24 - 34 | |
| 64-bit Real | DC [mod 111 r/m] | TOS <--- M.DR / TOS | 24 - 34 | |
| 32-bit Real | D8 [mod 111 r/m] | TOS <--- M.SR / TOS | 24 - 34 | |
| **FDIVRP** *Floating Point Divide Reversed, Pop* | DE [1111 0 n] | ST(n) <--- TOS / ST(n); then pop TOS | 24 - 34 | |
| **FIDIV** *Floating Point Integer Divide* | | | | |
| 32-bit Integer | DA [mod 110 r/m] | TOS <--- TOS / M.SI | 34 - 38 | |
| 16-bit Integer | DE [mod 110 r/m] | TOS <--- TOS / M.WI | 34 - 38 | |
| **FIDIVR** *Floating Point Integer Divide Reversed* | | | | |
| 32-bit Integer | DA [mod 111 r/m] | TOS <--- M.SI / TOS | 34 - 38 | |
| 16-bit Integer | DE [mod 111 r/m] | TOS <--- M.WI / TOS | 34 - 38 | |
| **FFREE** *Free Floating Point Register* | DD [1100 0 n] | TAG(n) <--- Empty | 4 | |
| **FINCSTP** *Increment Stack Pointer* | D9 F7 | Increment top-of-stack pointer | 2 | |
| **FINIT** *Initialize FPU* | (9B)DB E3 | Wait, then initialize | 8 | |
| **FNINIT** *Initialize FPU* | DB E3 | Initialize | 6 | |
| **FLD** *Load Data to FPU Register* | | | | |
| Top of Stack | D9 [1100 0 n] | Push ST(n) onto stack | 2 | |
| 80-bit Real | DB [mod 101 /m] | Push M.XR onto stack | 2 | |
| 64-bit Real | DD [mod 000 r/m] | Push M.DR onto stack | 2 | |
| 32-bit Real | D9 [mod 000 r/m] | Push M.SR onto stack | 2 | |
| **FBLD** *Load Packed BCD Data to FPU Register* | DF [mod 100 r/m] | Push M.BCD onto stack | 41 - 45 | |
| **FILD** *Load Integer Data to FPU Register* | | | | |
| 64-bit Integer | DF [mod 101 r/m] | Push M.LI onto stack | 4 - 8 | |
| 32-bit Integer | DB [mod 000 r/m] | Push M.SI onto stack | 4 - 6 | |
| 16-bit Integer | DF [mod 000 r/m] | Push M.WI onto stack | 3 - 6 | |
| **FLD1** *Load Floating Const.= 1.0* | D9 E8 | Push 1.0 onto stack | 4 | |
| **FLDCW** *Load FPU Mode Control Register* | D9 [mod 101 r/m] | Ctl Word <--- Memory | 4 | |
| **FLDENV** *Load FPU Environment* | D9 [mod 100 r/m] | Env Regs <--- Memory | 30 | |
| **FLDL2E** *Load Floating Const.= $Log_2(e)$* | D9 EA | Push $Log_2(e)$ onto stack | 4 | |
| **FLDL2T** *Load Floating Const.= $Log_2(10)$* | D9 E9 | Push $Log_2(10)$ onto stack | 4 | |
| **FLDLG2** *Load Floating Const.= $Log_{10}(2)$* | D9 EC | Push $Log_{10}(2)$ onto stack | 4 | |
| **FLDLN2** *Load Floating Const.= Ln(2)* | D9 ED | Push $Log_e(2)$ onto stack | 4 | |
| **FLDPI** *Load Floating Const.= $\pi$* | D9 EB | Push $\pi$ onto stack | 4 | |
| **FLDZ** *Load Floating Const.= 0.0* | D9 EE | Push 0.0 onto stack | 4 | |
| **FMUL** *Floating Point Multiply* | | | | |
| Top of Stack | DC [1100 1 n] | ST(n) <--- ST(n) $\times$ TOS | 4 - 9 | |
| 80-bit Register | D8 [1100 1 n] | TOS <--- TOS $\times$ ST(n) | 4 - 9 | |
| 64-bit Real | DC [mod 001 r/m] | TOS <--- TOS $\times$ M.DR | 4 - 8 | |
| 32-bit Real | D8 [mod 001 r/m] | TOS <--- TOS $\times$ M.SR | 4 - 6 | |
| **FMULP** *Floating Point Multiply & Pop* | DE [1100 1 n] | ST(n) <--- ST(n) $\times$ TOS; then pop TOS | 4 - 9 | |
| **FIMUL** *Floating Point Integer Multiply* | | | | |
| 32-bit Integer | DA [mod 001 r/m] | TOS <--- TOS $\times$ M.SI | 9 - 11 | |
| 16-bit Integer | DE [mod 001 r/m] | TOS <--- TOS $\times$ M.WI | 8 - 10 | |

## Instruction Set (Continued)

### Table 8-29. FPU Instruction Set Summary (Continued)

| FPU Instruction | Opcode | Operation | Clock Count | Issue |
|---|---|---|---|---|
| **FNOP** *No Operation* | D9 D0 | No Operation | 2 | |
| **FPATAN** *Function Eval: Tan$^{-1}$(y/x)* | D9 F3 | ST(1) <--- ATAN[ST(1) / TOS]; then pop TOS | 97 - 161 | 3 |
| **FPREM** *Floating Point Remainder* | D9 F8 | TOS <--- Rem[TOS / ST(1)] | 82 - 91 | |
| **FPREM1** *Floating Point Remainder IEEE* | D9 F5 | TOS <--- Rem[TOS / ST(1)] | 82 - 91 | |
| **FPTAN** *Function Eval: Tan(x)* | D9 F2 | TOS <--- TAN(TOS); then push 1.0 onto stack | 117 - 129 | 1 |
| **FRNDINT** *Round to Integer* | D9 FC | TOS <--- Round(TOS) | 10 - 20 | |
| **FRSTOR** *Load FPU Environment and Register* | DD [mod 100 r/m] | Restore state | 56 - 72 | |
| **FSAVE** *Save FPU Environment and Register* | (9B)DD [mod 110 r/m] | Wait, then save state | 57 - 67 | |
| **FNSAVE** *Save FPU Environment and Register* | DD [mod 110 r/m] | Save state | 55 - 65 | |
| **FSCALE** *Floating Multiply by 2$^n$* | D9 FD | TOS <--- TOS $\times$ 2$^{(ST(1))}$ | 7 - 14 | |
| **FSIN** *Function Evaluation: Sin(x)* | D9 FE | TOS <--- SIN(TOS) | 76 - 140 | 1 |
| **FSINCOS** *Function Eval.: Sin(x)& Cos(x)* | D9 FB | temp <--- TOS;<br>TOS <--- SIN(temp); then<br>push COS(temp) onto stack | 145 - 161 | 1 |
| **FSQRT** *Floating Point Square Root* | D9 FA | TOS <--- Square Root of TOS | 59 - 60 | |
| **FST** *Store FPU Register* | | | | |
| Top of Stack | DD [1101 0 n] | ST(n) <--- TOS | 2 | |
| 64-bit Real | DD [mod 010 r/m] | M.DR <--- TOS | 2 | |
| 32-bit Real | D9 [mod 010 r/m] | M.SR <--- TOS | 2 | |
| **FSTP** *Store FPU Register, Pop* | | | | |
| Top of Stack | DB [1101 1 n] | ST(n) <--- TOS; then pop TOS | 2 | |
| 80-bit Real | DB [mod 111 r/m] | M.XR <--- TOS; then pop TOS | 2 | |
| 64-bit Real | DD [mod 011 r/m] | M.DR <--- TOS; then pop TOS | 2 | |
| 32-bit Real | D9 [mod 011 r/m] | M.SR <--- TOS; then pop TOS | 2 | |
| **FBSTP** *Store BCD Data, Pop* | DF [mod 110 r/m] | M.BCD <--- TOS; then pop TOS | 57 - 63 | |
| **FIST** *Store Integer FPU Register* | | | | |
| 32-bit Integer | DB [mod 010 r/m] | M.SI <--- TOS | 8 - 13 | |
| 16-bit Integer | DF [mod 010 r/m] | M.WI <--- TOS | 7 - 10 | |
| **FISTP** *Store Integer FPU Register, Pop* | | | | |
| 64-bit Integer | DF [mod 111 r/m] | M.LI <--- TOS; then pop TOS | 10 - 13 | |
| 32-bit Integer | DB [mod 011 r/m] | M.SI <--- TOS; then pop TOS | 8 - 13 | |
| 16-bit Integer | DF [mod 011 r/m] | M.WI <--- TOS; then pop TOS | 7 - 10 | |
| **FSTCW** *Store FPU Mode Control Register* | (9B)D9 [mod 111 r/m] | Wait Memory <--- Control Mode Register | 5 | |
| **FNSTCW** *Store FPU Mode Control Register* | D9 [mod 111 r/m] | Memory <--- Control Mode Register | 3 | |
| **FSTENV** *Store FPU Environment* | (9B)D9 [mod 110 r/m] | Wait Memory <--- Env. Registers | 14 - 24 | |
| **FNSTENV** *Store FPU Environment* | D9 [mod 110 r/m] | Memory <--- Env. Registers | 12 - 22 | |
| **FSTSW** *Store FPU Status Register* | (9B)DD [mod 111 r/m] | Wait Memory <--- Status Register | 6 | |
| **FNSTSW** *Store FPU Status Register* | DD [mod 111 r/m] | Memory <--- Status Register | 4 | |
| **FSTSW AX** *Store FPU Status Register to AX* | (9B)DF E0 | Wait AX <--- Status Register | 4 | |
| **FNSTSW AX** *Store FPU Status Register to AX* | DF E0 | AX <--- Status Register | 2 | |
| **FSUB** *Floating Point Subtract* | | | | |
| Top of Stack | DC [1110 1 n] | ST(n) <--- ST(n) - TOS | 4 - 9 | |
| 80-bit Register | D8 [1110 0 n] | TOS <--- TOS - ST(n | 4 - 9 | |
| 64-bit Real | DC [mod 100 r/m] | TOS <--- TOS - M.DR | 4 - 9 | |
| 32-bit Real | D8 [mod 100 r/m] | TOS <--- TOS - M.SR | 4 - 9 | |
| **FSUBP** *Floating Point Subtract, Pop* | DE [1110 1 n] | ST(n) <--- ST(n) - TOS; then pop TOS | 4 - 9 | |
| **FSUBR** *Floating Point Subtract Reverse* | | | | |
| Top of Stack | DC [1110 0 n] | TOS <--- ST(n) - TOS | 4 - 9 | |
| 80-bit Register | D8 [1110 1 n] | ST(n) <--- TOS - ST(n) | 4 - 9 | |
| 64-bit Real | DC [mod 101 r/m] | TOS <--- M.DR - TOS | 4 - 9 | |
| 32-bit Real | D8 [mod 101 r/m] | TOS <--- M.SR - TOS | 4 - 9 | |

## Instruction Set (Continued)

### Table 8-29. FPU Instruction Set Summary (Continued)

| FPU Instruction | Opcode | Operation | Clock Count | Issue |
|---|---|---|---|---|
| **FSUBRP** *Floating Point Subtract Reverse, Pop* | DE [1110 0 n] | ST(n) <--- TOS - ST(n); then pop TOS | 4 - 9 | |
| **FISUB** *Floating Point Integer Subtract* | | | | |
| 32-bit Integer | DA [mod 100 r/m] | TOS <--- TOS - M.SI | 14 - 29 | |
| 16-bit Integer | DE [mod 100 r/m] | TOS <--- TOS - M.WI | 14 - 27 | |
| **FISUBR** *Floating Point Integer Subtract Reverse* | | | | |
| 32-bit Integer Reversed | DA [mod 101 r/m] | TOS <--- M.SI - TOS | 14 - 29 | |
| 16-bit Integer Reversed | DE [mod 101 r/m] | TOS <--- M.WI - TOS | 14 - 27 | |
| **FTST** *Test Top of Stack* | D9 E4 | CC set by TOS - 0.0 | 4 | |
| **FUCOM** *Unordered Compare* | DD [1110 0 n] | CC set by TOS - ST(n) | 4 | |
| **FUCOMP** *Unordered Compare, Pop* | DD [1110 1 n] | CC set by TOS - ST(n); then pop TOS | 4 | |
| **FUCOMPP** *Unordered Compare, Pop two elements* | DA E9 | CC set by TOS - ST(I); then pop TOS and ST(1) | 4 | |
| **FWAIT** *Wait* | 9B | Wait for FPU not busy | 2 | |
| **FXAM** *Report Class of Operand* | D9 E5 | CC <--- Class of TOS | 4 | |
| **FXCH** *Exchange Register with TOS* | D9 [1100 1 n] | TOS <--> ST(n) Exchange | 3 | |
| **FXTRACT** *Extract Exponent* | D9 F4 | temp <--- TOS;<br>TOS <--- exponent (temp); then<br>push significant (temp) onto stack | 11 - 16 | |
| **FLY2X** *Function Eval. y × Log2(x)* | D9 F1 | ST(1) <--- ST(1) × $Log_2$(TOS); then pop TOS | 145 - 154 | |
| **FLY2XP1** *Function Eval. y × Log2(x+1)* | D9 F9 | ST(1) <--- ST(1) × $Log_2$(1+TOS); then pop TOS | 131 - 133 | 4 |

**FPU Instruction Summary Issues**

All references to TOS and ST(n) refer to stack layout prior to execution.

Values popped off the stack are discarded.

A pop from the stack increments the top of stack pointer.

A push to the stack decrements the top of stack pointer.

**Issues:**

1. For FCOS, FSIN, FSINCOS and FPTAN, time shown is for absolute value of TOS < 3p/4. Add 90 clock counts for argument reduction if outside this range.

   For FCOS, clock count is 141 if TOS < π/4 and clock count is 92 if π/4 < TOS > π/2.

   For FSIN, clock count is 81 to 82 if absolute value of TOS < π/4.

2. For F2XM1, clock count is 92 if absolute value of TOS < 0.5.
3. For FPATAN, clock count is 97 if ST(1)/TOS < π/32.
4. For FYL2XP1, clock count is 170 if TOS is out of range and regular FYL2X is called.
5. The following opcodes are reserved:
   D9D7, D9E2, D9E7, DDFC, DED8, DEDA, DEDC, DEDD, DEDE, DFFC.

   If a reserved opcode is executed, and unpredictable results may occur (exceptions are not generated).

## 8.5 MMX INSTRUCTION SET

The CPU is functionally divided into the FPU unit, and the integer unit. The FPU has been extended to process both MMX instructions and floating point instructions in parallel with the integer unit.

For example, when the integer unit detects an MMX instruction, the instruction passes to the FPU unit for execution. The integer unit continues to execute instructions while the FPU unit executes the MMX instruction. If another MMX instruction is encountered, the second MMX instruction is placed in the MMX queue. Up to four MMX instructions can be queued.

The MMX instruction set is summarized in Table 8-31. The abbreviations used in the table are listed Table 8-30.

**Table 8-30. MMX Instruction Set Table Legend**

| Abbreviation | Description |
|---|---|
| <---- | Result written. |
| [11 mm reg] | Binary or binary groups of digits. |
| mm | One of eight 64-bit MMX registers. |
| reg | A general purpose register. |
| <--sat-- | If required, the resultant data is saturated to remain in the associated data range. |
| <--move-- | Source data is moved to result location. |
| [byte] | Eight 8-bit BYTEs are processed in parallel. |
| [word] | Four 16-bit WORDs are processed in parallel. |
| [dword] | Two 32-bit DWORDs are processed in parallel. |
| [qword] | One 64-bit QWORD is processed. |
| [sign xxx] | The BYTE, WORD, DWORD or QWORD most significant bit is a sign bit. |
| mm1, mm2 | MMX Register 1, MMX Register 2. |
| mod r/m | Mod and r/m byte encoding (Table 8-15 on page 217). |
| pack | Source data is truncated or saturated to next smaller data size, then concatenated. |
| packdw | Pack two DWORDs from source and two DWORDs from destination into four WORDs in destination register. |
| packwb | Pack four WORDs from source and four WORDs from destination into eight BYTEs in destination register. |

## Instruction Set (Continued)

### Table 8-31. MMX Instruction Set Summary

| MMX Instructions | Opcode | Operation and Clock Count (Latency/Throughput) | |
|---|---|---|---|
| **EMMS** *Empty MMX State* | 0F77 | Tag Word <--- FFFFh (empties the floating point tag word) | 1/1 |
| **MOVD** *Move Doubleword* | | | |
| Register to MMX Register | 0F6E [11 mm reg] | MMX reg [qword] <--move, zero extend-- reg [dword] | 1/1 |
| MMX Register to Register | 0F7E [11 mm reg] | reg [qword] <--move-- MMX reg [low dword] | 5/1 |
| Memory to MMX Register | 0F6E [mod mm r/m] | MMX regr[qword] <--move, zero extend-- memory[dword] | 1/1 |
| MMX Register to Memory | 0F7E [mod mm r/m] | Memory [dword] <--move-- MMX reg [low dword] | 1/1 |
| **MOVQ** *Move Quardword* | | | |
| MMX Register 2 to MMX Register 1 | 0F6F [11 mm1 mm2] | MMX reg 1 [qword] <--move-- MMX reg 2 [qword] | 1/1 |
| MMX Register 1 to MMX Register 2 | 0F7F [11 mm1 mm2] | MMX reg 2 [qword] <--move-- MMX reg 1 [qword] | 1/1 |
| Memory to MMX Register | 0F6F [mod mm r/m] | MMX reg [qword] <--move-- memory[qword] | 1/1 |
| MMX Register to Memory | 0F7F [mod mm r/m] | Memory [qword] <--move-- MMX reg [qword] | 1/1 |
| **PACKSSDW** *Pack Dword with Signed Saturation* | | | |
| MMX Register 2 to MMX Register 1 | 0F6B [11 mm1 mm2] | MMX reg 1 [qword] <--packdw, signed sat-- MMX reg 2, MMX reg 1 | 1/1 |
| Memory to MMX Register | 0F6B [mod mm r/m] | MMX reg [qword] <--packdw, signed sat-- memory, MMX reg | 1/1 |
| **PACKSSWB** *Pack Word with Signed Saturation* | | | |
| MMX Register 2 to MMX Register 1 | 0F63 [11 mm1 mm2] | MMX reg 1 [qword] <--packwb, signed sat-- MMX reg 2, MMX reg 1 | 1/1 |
| Memory to MMX Register | 0F63 [mod mm r/m] | MMX reg [qword] <--packwb, signed sat-- memory, MMX reg | 1/1 |
| **PACKUSWB** *Pack Word with Unsigned Saturation* | | | |
| MMX Register 2 to MMX Register 1 | 0F67 [11 mm1 mm2] | MMX reg 1 [qword] <--packwb, unsigned sat-- MMX reg 2, MMX reg 1 | 1/1 |
| Memory to MMX Register | 0F67 [mod mm r/m] | MMX reg [qword] <--packwb, unsigned sat-- memory, MMX reg | 1/1 |
| **PADDB** *Packed Add Byte with Wrap-Around* | | | |
| MMX Register 2 to MMX Register 1 | 0FFC [11 mm1 mm2] | MMX reg 1 [byte] <---- MMX reg 1 [byte] + MMX reg 2 [byte] | 1/1 |
| Memory to MMX Register | 0FFC [mod mm r/m] | MMX reg[byte] <---- memory [byte] + MMX reg [byte] | 1/1 |
| **PADDD** *Packed Add Dword with Wrap-Around* | | | |
| MMX Register 2 to MMX Register 1 | 0FFE [11 mm1 mm2] | MMX reg 1 [sign dword] <---- MMX reg 1 [sign dword] + MMX reg 2 [sign dword] | 1/1 |
| Memory to MMX Register | 0FFE [mod mm r/m] | MMX reg [sign dword] <---- memory [sign dword] + MMX reg [sign dword] | 1/1 |
| **PADDSB** *Packed Add Signed Byte with Saturation* | | | |
| MMX Register 2 to MMX Register 1 | 0FEC [11 mm1 mm2] | MMX reg 1 [sign byte] <--sat-- MMX reg 1 [sign byte] + MMX reg 2 [sign byte] | 1/1 |
| Memory to Register | 0FEC [mod mm r/m] | MMX reg [sign byte] <--sat-- memory [sign byte] + MMX reg [sign byte] | 1/1 |
| **PADDSW** *Packed Add Signed Word with Saturation* | | | |
| MMX Register 2 to MMX Register 1 | 0FED [11 mm1 mm2] | MMX reg 1 [sign word] <--sat-- MMX reg 1 [sign word] + MMX reg 2 [sign word] | 1/1 |
| Memory to Register | 0FED [mod mm r/m] | MMX reg [sign word] <--sat-- memory [sign word] + MMX reg [sign word] | 1/1 |
| **PADDUSB** *Add Unsigned Byte with Saturation* | | | |
| MMX Register 2 to MMX Register 1 | 0FDC [11 mm1 mm2] | MMX reg 1 [byte] <--sat-- MMX reg 1 [byte] + MMX reg 2 [byte] | 1/1 |
| Memory to Register | 0FDC [mod mm r/m] | MMX reg [byte] <--sat-- memory [byte] + MMX reg [byte] | 1/1 |
| **PADDUSW** *Add Unsigned Word with Saturation* | | | |
| MMX Register 2 to MMX Register 1 | 0FDD [11 mm1 mm2] | MMX reg 1 [word] <--sat-- MMX reg 1 [word] + MMX reg 2 [word] | 1/1 |
| Memory to Register | 0FDD [mod mm r/m] | MMX reg [word] <--sat-- memory [word] + MMX reg [word] | 1/1 |
| **PADDW** *Packed Add Word with Wrap-Around* | | | |
| MMX Register 2 to MMX Register 1 | 0FFD [11 mm1 mm2] | MMX reg 1 [word] <---- MMX reg 1 [word] + MMX reg 2 [word] | 1/1 |
| Memory to MMX Register | 0FFD [mod mm r/m] | MMX reg [word] <---- memory [word] + MMX reg [word] | 1/1 |
| **PAND** *Bitwise Logical AND* | | | |
| MMX Register 2 to MMX Register 1 | 0FDB [11 mm1 mm2] | MMX reg 1 [qword] <--logic AND-- MMX reg 1 [qword], MMX reg 2 [qword] | 1/1 |
| Memory to MMX Register | 0FDB [mod mm r/m] | MMX reg [qword] <--logic AND-- memory [qword], MMX reg [qword] | |
| **PANDN** *Bitwise Logical AND NOT* | | | |
| MMX Register 2 to MMX Register 1 | 0FDF [11 mm1 mm2] | MMX reg 1 [qword] <--logic AND -- NOT MMX reg 1 [qword], MMX reg 2 [qword] | 1/1 |
| Memory to MMX Register | 0FDF [mod mm r/m] | MMX reg [qword] <--logic AND-- NOT MMX reg [qword], Memory [qword] | 1/1 |

## Instruction Set (Continued)

### Table 8-31. MMX Instruction Set Summary (Continued)

| MMX Instructions | Opcode | Operation and Clock Count (Latency/Throughput) | |
|---|---|---|---|
| **PCMPEQB** *Packed Byte Compare for Equality* | | | |
| MMX Register 2 with MMX Register 1 | 0F74 [11 mm1 mm2] | MMX reg 1 [byte] <--FFh-- if MMX reg 1 [byte] = MMX reg 2 [byte]<br>MMX reg 1 [byte]<--00h-- if MMX reg 1 [byte] NOT = MMX reg 2 [byte] | 1/1 |
| Memory with MMX Register | 0F74 [mod mm r/m] | MMX reg [byte] <--FFh-- if memory[byte] = MMX reg [byte]<br>MMX reg [byte] <--00h-- if memory[byte] NOT = MMX reg [byte] | 1/1 |
| **PCMPEQD** *Packed Dword Compare for Equality* | | | |
| MMX Register 2 with MMX Register 1 | 0F76 [11 mm1 mm2] | MMX reg 1 [dword] <--FFFF FFFFh-- if MMX reg 1 [dword] = MMX reg 2 [dword]<br>MMX reg 1 [dword]<--0000 0000h--if MMX reg 1[dword] NOT = MMX reg 2 [dword] | 1/1 |
| Memory with MMX Register | 0F76 [mod mm r/m] | MMX reg [dword] <--FFFF FFFFh-- if memory[dword] = MMX reg [dword]<br>MMX reg [dword] <--0000 0000h-- if memory[dword] NOT = MMX reg [dword] | 1/1 |
| **PCMPEQW** *Packed Word Compare for Equality* | | | |
| MMX Register 2 with MMX Register 1 | 0F75 [11 mm1 mm2] | MMX reg 1 [word] <--FFFFh-- if MMX reg 1 [word] = MMX reg 2 [word]<br>MMX reg 1 [word]<--0000h-- if MMX reg 1 [word] NOT = MMX reg 2 [word] | 1/1 |
| Memory with MMX Register | 0F75 [mod mm r/m] | MMX reg [word] <--FFFFh-- if memory[word] = MMX reg [word]<br>MMX reg [word] <--0000h-- if memory[word] NOT = MMX reg [word] | 1/1 |
| **PCMPGTB** *Pack Compare Greater Than Byte* | | | |
| MMX Register 2 to MMX Register 1 | 0F64 [11 mm1 mm2] | MMX reg 1 [byte] <--FFh-- if MMX reg 1 [byte] > MMX reg 2 [byte]<br>MMX reg 1 [byte]<--00h-- if MMX reg 1 [byte] NOT > MMX reg 2 [byte] | 1/1 |
| Memory with MMX Register | 0F64 [mod mm r/m] | MMX reg [byte] <--FFh-- if memory[byte] > MMX reg [byte]<br>MMX reg [byte] <--00h-- if memory[byte] NOT > MMX reg [byte] | 1/1 |
| **PCMPGTD** *Pack Compare Greater Than Dword* | | | |
| MMX Register 2 to MMX Register 1 | 0F66 [11 mm1 mm2] | MMX reg 1 [dword] <--FFFF FFFFh-- if MMX reg 1 [dword] > MMX reg 2 [dword]<br>MMX reg 1 [dword]<--0000 0000h--if MMX reg 1 [dword]NOT > MMX reg 2 [dword] | 1/1 |
| Memory with MMX Register | 0F66 [mod mm r/m] | MMX reg [dword] <--FFFF FFFFh-- if memory[dword] > MMX reg [dword]<br>MMX reg [dword] <--0000 0000h-- if memory[dword] NOT > MMX reg [dword] | 1/1 |
| **PCMPGTW** *Pack Compare Greater Than Word* | | | |
| MMX Register 2 to MMX Register 1 | 0F65 [11 mm1 mm2] | MMX reg 1 [word] <--FFFFh-- if MMX reg 1 [word] > MMX reg 2 [word]<br>MMX reg 1 [word]<--0000h-- if MMX reg 1 [word] NOT > MMX reg 2 [word] | 1/1 |
| Memory with MMX Register | 0F65 [mod mm r/m] | MMX reg [word] <--FFFFh-- if memory[word] > MMX reg [word]<br>MMX reg [word] <--0000h-- if memory[word] NOT > MMX reg [word] | 1/1 |
| **PMADDWD** *Packed Multiply and Add* | | | |
| MMX Register 2 to MMX Register 1 | 0FF5 [11 mm1 mm2] | MMX reg 1 [dword] <--add-- [dword]<---- MMX reg 1 [sign word]*MMX reg 2[sign word] | 2/1 |
| Memory to MMX Register | 0FF5 [mod mm r/m] | MMX reg 1 [dword] <--add-- [dword] <---- memory [sign word] * Memory [sign word] | 2/1 |
| **PMULHW** *Packed Multiply High* | | | |
| MMX Register 2 to MMX Register 1 | 0FE5 [11 mm1 mm2] | MMX reg 1 [word] <--upper bits-- MMX reg 1 [sign word] * MMX reg 2 [sign word] | 2/1 |
| Memory to MMX Register | 0FE5 [mod mm r/m] | MMX reg 1 [word] <--upper bits-- memory [sign word] * Memory [sign word] | 2/1 |
| **PMULLW** *Packed Multiply Low* | | | |
| MMX Register 2 to MMX Register 1 | 0FD5 [11 mm1 mm2] | MMX reg 1 [word] <--lower bits-- MMX reg 1 [sign word] * MMX reg 2 [sign word] | 2/1 |
| Memory to MMX Register | 0FD5 [mod mm r/m] | MMX reg 1 [word] <--lower bits-- memory [sign word] * Memory [sign word] | 2/1 |
| **POR** *Bitwise OR* | | | |
| MMX Register 2 to MMX Register 1 | 0FEB [11 mm1 mm2] | MMX reg 1 [qword] <--logic OR-- MMX reg 1 [qword], MMX reg 2 [qword] | 1/1 |
| Memory to MMX Register | 0FEB [mod mm r/m] | MMX reg [qword] <--logic OR-- MMX reg [qword], memory[qword] | 1/1 |
| **PSLLD** *Packed Shift Left Logical Dword* | | | |
| MMX Register 1 by MMX Register 2 | 0FF2 [11 mm1 mm2] | MMX reg 1 [dword] <--shift left, shifting in zeroes by MMX reg 2 [dword]-- | 1/1 |
| MMX Register by Memory | 0FF2 [mod mm r/m] | MMX reg [dword] <--shift left, shifting in zeroes by memory[dword]-- | 1/1 |
| MMX Register by Immediate | 0F72 [11 110 mm] # | MMX reg [dword] <--shift left, shifting in zeroes by [im byte]-- | 1/1 |
| **PSLLQ** *Packed Shift Left Logical Qword* | | | |
| MMX Register 1 by MMX Register 2 | 0FF3 [11 mm1 mm2] | MMX reg 1 [qword] <--shift left, shifting in zeroes by MMX reg 2 [qword]-- | 1/1 |
| MMX Register by Memory | 0FF3 [mod mm r/m] | MMX reg [qword] <--shift left, shifting in zeroes by [qword]-- | 1/1 |
| MMX Register by Immediate | 0F73 [11 110 mm] # | MMX reg [qword] <--shift left, shifting in zeroes by [im byte]-- | 1/1 |

## Instruction Set (Continued)

### Table 8-31. MMX Instruction Set Summary (Continued)

| MMX Instructions | Opcode | Operation and Clock Count (Latency/Throughput) | |
|---|---|---|---|
| **PSLLW** *Packed Shift Left Logical Word* | | | |
| MMX Register 1 by MMX Register 2 | 0FF1 [11 mm1 mm2] | MMX reg 1 [word] <--shift left, shifting in zeroes by MMX reg 2 [word]-- | 1/1 |
| MMX Register by Memory | 0FF1 [mod mm r/m] | MMX reg [word] <--shift left, shifting in zeroes by memory[word]-- | 1/1 |
| MMX Register by Immediate | 0F71 [11 110mm] # | MMX reg [word] <--shift left, shifting in zeroes by [im byte]-- | 1/1 |
| **PSRAD** *Packed Shift Right Arithmetic Dword* | | | |
| MMX Register 1 by MMX Register 2 | 0FE2 [11 mm1 mm2] | MMX reg 1 [dword] <--arith shift right, shifting in zeroes by MMX reg 2 [dword--] | 1/1 |
| MMX Register by Memory | 0FE2 [mod mm r/m] | MMX reg [dword] <--arith shift right, shifting in zeroes by memory[dword]-- | 1/1 |
| MMX Register by Immediate | 0F72 [11 100 mm] # | MMX reg [dword] <--arith shift right, shifting in zeroes by [im byte]-- | 1/1 |
| **PSRAW** *Packed Shift Right Arithmetic Word* | | | |
| MMX Register 1 by MMX Register 2 | 0FE1 [11 mm1 mm2] | MMX reg 1 [word] <--arith shift right, shifting in zeroes by MMX reg 2 [word]-- | 1/1 |
| MMX Register by Memory | 0FE1 [mod mm r/m] | MMX reg [word] <--arith shift right, shifting in zeroes by memory[word--] | 1/1 |
| MMX Register by Immediate | 0F71 [11 100 mm] # | MMX reg [word] <--arith shift right, shifting in zeroes by [im byte]-- | 1/1 |
| **PSRLD** *Packed Shift Right Logical Dword* | | | |
| MMX Register 1 by MMX Register 2 | 0FD2 [11 mm1 mm2] | MMX reg 1 [dword] <--shift right, shifting in zeroes by MMX reg 2 [dword]-- | 1/1 |
| MMX Register by Memory | 0FD2 [mod mm r/m] | MMX reg [dword] <--shift right, shifting in zeroes by memory[dword]-- | 1/1 |
| MMX Register by Immediate | 0F72 [11 010 mm] # | MMX reg [dword] <--shift right, shifting in zeroes by [im byte]-- | 1/1 |
| **PSRLQ** *Packed Shift Right Logical Qword* | | | |
| MMX Register 1 by MMX Register 2 | 0FD3 [11 mm1 mm2] | MMX reg 1 [qword] <--shift right, shifting in zeroes by MMX reg 2 [qword] | 1/1 |
| MMX Register by Memory | 0FD3 [mod mm r/m] | MMX reg [qword] <--shift right, shifting in zeroes by memory[qword] | 1/1 |
| MMX Register by Immediate | 0F73 [11 010 mm] # | MMX reg [qword] <--shift right, shifting in zeroes by [im byte] | 1/1 |
| **PSRLW** *Packed Shift Right Logical Word* | | | |
| MMX Register 1 by MMX Register 2 | 0FD1 [11 mm1 mm2] | MMX reg 1 [word] <--shift right, shifting in zeroes by MMX reg 2 [word] | 1/1 |
| MMX Register by Memory | 0FD1 [mod mm r/m] | MMX reg [word] <--shift right, shifting in zeroes by memory[word] | 1/1 |
| MMX Register by Immediate | 0F71 [11 010 mm] # | MMX reg [word] <--shift right, shifting in zeroes by imm[word] | 1/1 |
| **PSUBB** *Subtract Byte With Wrap-Around* | | | |
| MMX Register 2 to MMX Register 1 | 0FF8 [11 mm1 mm2] | MMX reg 1 [byte] <---- MMX reg 1 [byte] subtract MMX reg 2 [byte] | 1/1 |
| Memory to MMX Register | 0FF8 [mod mm r/m] | MMX reg [byte] <---- MMX reg [byte] subtract memory [byte] | 1/1 |
| **PSUBD** *Subtract Dword With Wrap-Around* | | | |
| MMX Register 2 to MMX Register 1 | 0FFA [11 mm1 mm2] | MMX reg 1 [dword] <---- MMX reg 1 [dword] subtract MMX reg 2 [dword] | 1/1 |
| Memory to MMX Register | 0FFA [mod mm r/m] | MMX reg [dword] <---- MMX reg [dword] subtract memory [dword] | 1/1 |
| **PSUBSB** *Subtract Byte Signed With Saturation* | | | |
| MMX Register 2 to MMX Register 1 | 0FE8 [11 mm1 mm2] | MMX reg 1 [sign byte] <--sat-- MMX reg 1 [sign byte] subtract MMX reg 2 [sign byte] | 1/1 |
| Memory to MMX Register | 0FE8 [mod mm r/m] | MMX reg [sign byte] <--sat-- MMX reg [sign byte] subtract memory [sign byte] | 1/1 |
| **PSUBSW** *Subtract Word Signed With Saturation* | | | |
| MMX Register 2 to MMX Register 1 | 0FE9 [11 mm1 mm2] | MMX reg 1 [sign word] <--sat-- MMX reg 1 [sign word] subtract MMX reg 2 [sign word] | 1/1 |
| Memory to MMX Register | 0FE9 [mod mm r/m] | MMX reg [sign word] <--sat-- MMX reg [sign word] subtract memory [sign word] | 1/1 |
| **PSUBUSB** *Subtract Byte Unsigned With Saturation* | | | |
| MMX Register 2 to MMX Register 1 | 0FD8 [11 mm1 mm2] | MMX reg 1 [byte] <--sat-- MMX reg 1 [byte] subtract MMX reg 2 [byte] | 1/1 |
| Memory to MMX Register | 0FD8 [11 mm reg] | MMX reg [byte] <--sat-- MMX reg [byte] subtract memory [byte] | 1/1 |
| **PSUBUSW** *Subtract Word Unsigned With Saturation* | | | |
| MMX Register 2 to MMX Register 1 | 0FD9 [11 mm1 mm2] | MMX reg 1 [word] <--sat-- MMX reg 1 [word] subtract MMX reg 2 [word] | 1/1 |
| Memory to MMX Register | 0FD9 [11 mm reg] | MMX reg [word] <--sat-- MMX reg [word] subtract memory [word] | 1/1 |
| **PSUBW** *Subtract Word With Wrap-Around* | | | |
| MMX Register 2 to MMX Register 1 | 0FF9 [11 mm1 mm2] | MMX reg 1 [word] <---- MMX reg 1 [word] subtract MMX reg 2 [word] | 1/1 |
| Memory to MMX Register | 0FF9 [mod mm r/m] | MMX reg [word] <---- MMX reg [word] subtract memory [word] | 1/1 |
| **PUNPCKHBW** *Unpack High Packed Byte, Data to Packed Words* | | | |
| MMX Register 2 to MMX Register 1 | 0F68 [11 mm1 mm2] | MMX reg 1 [byte] <--interleave-- MMX reg 1 [up byte], MMX reg 2 [up byte] | 1/1 |
| Memory to MMX Register | 0F68 [11 mm reg] | MMX reg [byte] <--interleave-- memory [up byte], MMX reg [up byte] | 1/1 |
| **PUNPCKHDQ** *Unpack High Packed Dword, Data to Qword* | | | |
| MMX Register 2 to MMX Register 1 | 0F6A [11 mm1 mm2] | MMX reg 1 [dword] <--interleave-- MMX reg 1 [up dword], MMX reg 2 [up dword] | 1/1 |
| Memory to MMX Register | 0F6A [11 mm reg] | MMX reg [dword] <--interleave-- memory [up dword], MMX reg [up dword] | 1/1 |

## Instruction Set (Continued)

### Table 8-31. MMX Instruction Set Summary (Continued)

| MMX Instructions | Opcode | Operation and Clock Count (Latency/Throughput) | |
|---|---|---|---|
| **PUNPCKHWD** *Unpack High Packed Word, Data to Packed Dwords* | | | |
| MMX Register 2 to MMX Register 1 | 0F69 [11 mm1 mm2] | MMX reg 1 [word] <--interleave-- MMX reg 1 [up word], MMX reg 2 [up word] | 1/1 |
| Memory to MMX Register | 0F69 [11 mm reg] | MMX reg [word] <--interleave-- memory [up word], MMX reg [up word] | 1/1 |
| **PUNPCKLBW** *Unpack Low Packed Byte, Data to Packed Words* | | | |
| MMX Register 2 to MMX Register 1 | 0F60 [11 mm1 mm2] | MMX reg 1 [word] <--interleave-- MMX reg 1 [low byte], MMX reg 2 [low byte] | 1/1 |
| Memory to MMX Register | 0F60 [11 mm reg] | MMX reg [word] <--interleave-- memory [low byte], MMX reg [low byte] | 1/1 |
| **PUNPCKLDQ** *Unpack Low Packed Dword, Data to Qword* | | | |
| MMX Register 2 to MMX Register 1 | 0F62 [11 mm1 mm2] | MMX reg 1 [word] <--interleave-- MMX reg 1 [low dword], MMX reg 2 [low dword] | 1/1 |
| Memory to MMX Register | 0F62 [11 mm reg] | MMX reg [word] <--interleave-- memory [low dword], MMX reg [low dword] | 1/1 |
| **PUNPCKLWD** *Unpack Low Packed Word, Data to Packed Dwords* | | | |
| MMX Register 2 to MMX Register 1 | 0F61 [11 mm1 mm2] | MMX reg 1 [word] <--interleave-- MMX reg 1 [low word], MMX reg 2 [low word] | 1/1 |
| Memory to MMX Register | 0F61 [11 mm reg] | MMX reg [word] <--interleave-- memory [low word], MMX reg [low word] | 1/1 |
| **PXOR** *Bitwise XOR* | | | |
| MMX Register 2 to MMX Register 1 | 0FEF [11 mm1 mm2] | MMX reg 1 [qword] <--logic exclusive OR-- MMX reg 1 [qword], MMX reg 2 [qword] | 1/1 |
| Memory to MMX Register | 0FEF [11 mm reg] | MMX reg [qword] <--logic exclusive OR-- memory[qword], MMX reg [qword] | 1/1 |

## 8.6 EXTENDED MMX INSTRUCTION SET

National Semiconductor has added instructions to its implementation of the Intel MMX architecture in order to facilitate writing of multimedia applications. In general, these instructions allow more efficient implementation of multimedia algorithms, or more precision in computation than can be achieved using the basic set of MMX instructions. All of the added instructions follow the SIMD (single instruction, multiple data) format. Many of the instructions add flexibility to the MMX architecture by allowing both source operands of an instruction to be preserved, while the result goes to a separate register that is derived from the input.

Table 8-33 summarizes the Extended MMX Instructions. The abbreviations used in the table are listed in Table 8-32.

Configuration control register CCR7(0) at Index EBh (see Table 3-11 on page 53) must be set to allow the execution of the Extended MMX instructions.

**Table 8-32. Extend MMX Instruction Set Table Legend**

| Abbreviation | Description |
|---|---|
| <---- | Result written. |
| [11 mm reg] | Binary or binary groups of digits. |
| mm | One of eight 64-bit MMX registers. |
| reg | A general purpose register. |
| <--sat-- | If required, the resultant data is saturated to remain in the associated data range. |
| <--move-- | Source data is moved to result location. |
| [byte] | Eight 8-bit BYTEs are processed in parallel. |
| [word] | Four 16-bit WORDs are processed in parallel. |
| [dword] | Two 32-bit DWORDs are processed in parallel. |
| [qword] | One 64-bit QWORD is processed. |
| [sign xxx] | The BYTE, WORD, DWORD or QWORD most significant bit is a sign bit. |
| mm1, mm2 | MMX Register 1, MMX Register 2. |
| mod r/m | Mod and r/m byte encoding (Table 8-15 on page 217). |
| pack | Source data is truncated or saturated to next smaller data size, then concatenated. |
| packdw | Pack two DWORDs from source and two DWORDs from destination into QWORDs in destination register. |
| packwb | Pack QWORDs from source and QWORDs from destination into eight BYTEs in destination register. |

## Instruction Set (Continued)

### Table 8-33. Extended MMX Instruction Set Summary

| MMX Instructions | Opcode | Operation and Clock Count | |
|---|---|---|---|
| **PADDSIW** *Packed Add Signed Word with Saturation Using Implied Destination* | | | |
| MMX Register plus MMX Register to Implied Register | 0F51 [11 mm1 mm2] | Sum signed packed word from MMX register/memory ---> signed packed word in MMX register, saturate, and write result ---> implied register | 1 |
| Memory plus MMX Register to Implied Register | 0F51 [mod mm r/m] | | 1 |
| **PAVEB** *Packed Average Byte* | | | |
| MMX Register 2 with MMX Register 1 | 0F50 [11 mm1 mm2] | Average packed byte from the MMX register/memory with packed byte in the MMX register. Result is placed in the MMX register. | 1 |
| Memory with MMX Register | 0F50 [mod mm r/m] | | 1 |
| **PDISTIB** *Packed Distance and Accumulate with Implied Register* | | | |
| Memory, MMX Register to Implied Register | 0F54 [mod mm r/m] | Find absolute value of difference between packed byte in memory and packed byte in the MMX register. Using unsigned saturation, accumulate with value in implied destination register. | 2 |
| **PMACHRIW** *Packed Multiply and Accumulate with Rounding* | | | |
| Memory to MMX Register | 0F5E[mod mm r/m] | Multiply the packed word in the MMX register by the packed word in memory. Sum the 32-bit results pairwise. Accumulate the result with the packed signed word in the implied destination register. | 2 |
| **PMAGW** *Packed Magnitude* | | | |
| MMX Register 2 to MMX Register 1 | 0F52 [11 mm1 mm2] | Set the destination equal ---> the packed word with the largest magnitude, between the packed word in the MMX register/memory and the MMX register. | 2 |
| Memory to MMX Register | 0F52 [mod mm r/m] | | 2 |
| **PMULHRIW** *Packed Multiply High with Rounding, Implied Destination* | | | |
| MMX Register 2 to MMX Register1 | 0F5D [11 mm1 mm2] | Packed multiply high with rounding and store bits 30 - 15 in implied register. | 2 |
| Memory to MMX Register | 0F5D [mod mm r/m] | | 2 |
| **PMULHRW** *Packed Multiply High with Rounding* | | | |
| MMX Register 2 to MMX Register 1 | 0F59 [11 mm1 mm2] | Multiply the signed packed word in the MMX register/memory with the signed packed word in the MMX register. Round with 1/2 bit 15, and store bits 30 - 15 of result in the MMX register. | 2 |
| Memory to MMX Register | 0F59 [mod mm r/m] | | 2 |
| **PMVGEZB** *Packed Conditional Move If Greater Than or Equal to Zero* | | | |
| Memory to MMX Register | 0F5C [mod mm r/m] | Conditionally move packed byte from memory ---> packed byte in the MMX register if packed byte in implied MMX register is greater than or equal ---> zero. | 1 |
| **PMVLZB** *Packed Conditional Move If Less Than Zero* | | | |
| Memory to MMX Register | 0F5B [mod mm r/m] | Conditionally move packed byte from memory ---> packed byte in the MMX register if packed byte in implied MMX register is less than zero. | 1 |
| **PMVNZB** *Packed Conditional Move If Not Zero* | | | |
| Memory to MMX Register | 0F5A [mod mm r/m] | Conditionally move packed byte from memory ---> packed byte in the MMX register if packed byte in implied MMX register is not zero. | 1 |
| **PMVZB** *Packed Conditional Move If Zero* | | | |
| Memory to MMX Register | 0F58 [mod mm r/m] | Conditionally move packed byte from memory ---> packed byte in the MMX register if packed byte in implied the MMX register is zero. | 1 |
| **PSUBSIW** *Packed Subtracted with Saturation Using Implied Destination* | | | |
| MMX Register 2 to MMX Register 1 | 0F55 [11 mm1 mm2] | Subtract signed packed word in the MMX register/memory from signed packed word in the MMX register, saturate, and write result ---> implied register. | 1 |
| Memory to MMX Register | 0F55 [mod mm r/m] | | 1 |

# Appendix A Support Documentation

## A.1 ORDER INFORMATION

| Order Number | Part Marking | Core Frequency (MHz) | Core Voltage ($V_{CC2}$) | Temperature (Degree C) | Package |
|---|---|---|---|---|---|
| 30070-53 | GXLV-266P 2.9V 70C | 266 | 2.9V | 70 | SPGA |
| 30071-53 | GXLV-266P 2.9V 85C | 266 | 2.9V | 85 | SPGA |
| 30170-53 | GXLV-266B 2.9V 70C | 266 | 2.9V | 70 | BGA |
| 30171-53 | GXLV-266B 2.9V 85C | 266 | 2.9V | 85 | BGA |
| 30057-33 | GXLV-233P 2.5V 85C | 233 | 2.5V | 85 | SPGA |
| 30157-33 | GXLV-233B 2.5V 85C | 233 | 2.5V | 85 | BGA |
| 30046-23 | GXLV-200P 2.2V 85C | 200 | 2.2V | 85 | SPGA |
| 30144-23 | GXLV-200B 2.2V 85C | 200 | 2.2V | 85 | BGA |
| 30036-23 | GXLV-180P 2.2V 85C | 180 | 2.2V | 85 | SPGA |
| 30134-23 | GXLV-180B 2.2V 85C | 180 | 2.2V | 85 | BGA |
| 30026-13 | GXLV-166P 2.2V 85C | 166 | 2.2V | 85 | SPGA |
| 30129-13 | GXLV-166B 2.2V 85C | 166 | 2.2V | 85 | BGA |

## A.2 DATA BOOK REVISION HISTORY

This document is a report of the revision/creation process of the data book for the GXLV Processor. Any revisions (i.e., additions, deletions, parameter corrections, etc.) are recorded in the tables below.

**Table A-1. Revision History**

| Revision # (PDF Date) | Revisions / Comments |
|---|---|
| 0.0 (2/5/98) | Creation phase |
| 0.1 (7/7/99) | Creation phase continues - added instruction set. |
| 0.2 (9/15/99) | Creation phase continues - added integrated functions. Also edited other sections. |
| 0.3 (10/29/99) | Creation phase continues - major edits to Display Controller and PCI Controller sections. Also edited other sections. |
| 0.4 (11/12/99) | Creation phase continues - edited all sections after formal reviews. |
| 1.0 (12/1/99) | Posted to web site. |
| 1.1 (4/6/00) | Formatting changes and engineering changes. See Table A-2 for details. |

**Table A-2. Edits to Current Revision**

| Section | Revision |
|---|---|
| 3.0 Processor Programming | • Combined bits 1 and 2 of Configuration Control Register 1 in Table 3-11 on page 52. |
| 6.0 Electricals | • All references to Recommended Operating Conditions became Operating Conditions.<br>• Table 6-3 on page 188 - The $V_{CC2}$ maximum voltage for 2.9V changed from 3.6V to 3.2V. |

**LIFE SUPPORT POLICY**

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

**National Semiconductor Corporation**
Americas
Tel: 1-800-272-9959
Fax: 1-800-737-7018
Email: support@nsc.com

**National Semiconductor Europe**
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Francais Tel: +33 (0) 1 41 91 8790

**National Semiconductor Asia Pacific Customer Response Group**
Tel: 65-2544466
Fax: 65-2504466
Email: ap.support@nsc.com

**National Semiconductor Japan Ltd.**
Tel: 81-3-5639-7560
Fax: 81-3-5639-7507

**www.national.com**

National does not assume any responsibility for use of any circuitry described, no circuit patent licenses are implied and National reserves the right at any time without notice to change said circuitry and specifications.

Общество с ограниченной ответственностью «МосЧип» ИНН 7719860671 / КПП 771901001
Адрес: 105318, г.Москва, ул.Щербаковская д.3, офис 1107

**Данный компонент на территории Российской Федерации**

**Вы можете приобрести в компании MosChip.**

Для оперативного оформления запроса Вам необходимо перейти по данной ссылке:

http://moschip.ru/get-element

Вы можете разместить у нас заказ для любого Вашего проекта, будь то серийное производство или разработка единичного прибора.

В нашем ассортименте представлены ведущие мировые производители активных и пассивных электронных компонентов.

Нашей специализацией является поставка электронной компонентной базы двойного назначения, продукции таких производителей как XILINX, Intel (ex.ALTERA), Vicor, Microchip, Texas Instruments, Analog Devices, Mini-Circuits, Amphenol, Glenair.

Сотрудничество с глобальными дистрибьюторами электронных компонентов, предоставляет возможность заказывать и получать с международных складов практически любой перечень компонентов в оптимальные для Вас сроки.

На всех этапах разработки и производства наши партнеры могут получить квалифицированную поддержку опытных инженеров.

Система менеджмента качества компании отвечает требованиям в соответствии с ГОСТ Р ИСО 9001, ГОСТ РВ 0015-002 и ЭС РД 009

**Офис по работе с юридическими лицами:**

105318, г.Москва, ул.Щербаковская д.3, офис 1107, 1118, ДЦ «Щербаковский»

Телефон: +7 495 668-12-70 (многоканальный)

Факс: +7 495 668-12-70 (доб.304)

E-mail: info@moschip.ru

Skype отдела продаж:
moschip.ru
moschip.ru_4
moschip.ru_6
moschip.ru_9