AMD Alchemy™

# AMD Alchemy™ Au1500™ Processor Data Book

*March 2006*

**Publication ID: 30361D**

# Contents

# List of Figures

# List of Tables

# 1 Overview

The AMD Alchemy™ Au1500™ processor is a high-performance, low-power, high integration system-on-a-chip (SOC) designed for use in the Internet edge device market. These devices are customer premise equipment products, including both wireless and wired Internet access devices and portable compute devices, as well as Internet infrastructure products such as routers and line cards. Devices based on the Au1500 are capable of processing high performance digital video and data streams in either battery or line-powered environments. The Au1500 processor contains a PCI controller, providing a flexible interface to external peripherals.

## 1.1 Product Description

The Au1500 processor is a complete SOC based on the MIPS32™ instruction set. Designed for maximum performance at low power, the processor runs up to 500 MHz. Power dissipation is less than 700 mW for the 400 MHz version. Highly integrated with on-chip memory controllers and Internet access peripherals, the Au1500 processor runs a variety of operating systems, including Windows® CE, Linux, and VxWorks. Moreover, the integration of peripherals with the unique, high performance, MIPS-compatible core provides low system cost, small form factor, low system power requirement, simple designs at multiple performance points and thus, short design cycles.

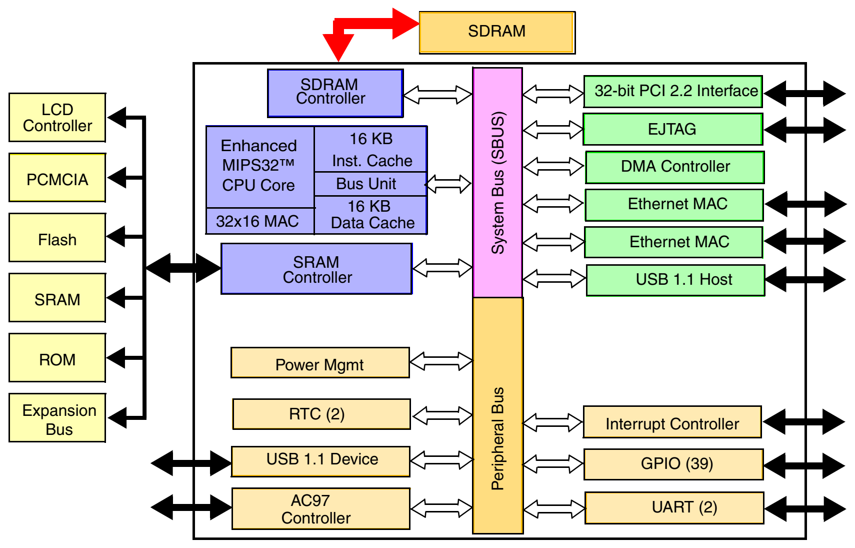


**Figure 1-1. Block Diagram**

## 1.2 Features

**High Speed MIPS CPU Core**

- 333, 400, or 500 MHz
- MIPS32 instruction set 32-bit architecture
- 16 KB instruction and 16 KB data caches
- High speed multiply-accumulate (MAC) and divide unit
- 1.5V core @ 333 MHz and 400 MHz,
  1.8V core @ 500 MHz
- 3.3V I/O
- Pipeline:
  — Scalar 5-stage pipeline
  — Load/store adder in I-stage (instr decode)
  — Scalar branch techniques optimized: Pipelined register file access in fetch stage
  — Zero penalty branch
- Multiply-Accumulate (MAC) and Divide Unit:
  — Max issue rate of one 32x16 MAC per clock
  — Max issue rate of one 32x32 MAC per every other clock
  — Operates in parallel to CPU pipeline
  — Executes all integer multiply and divide instructions
  — 32 x 16-bit MAC hardware
-

**Highly-Integrated System Peripherals**

- GPIOs (39 total, 22 dedicated for system use)
- Two 10/100 Ethernet MAC controllers
- USB 1.1 device and host controllers
- Two UARTs
- AC97 controller
- PCI interface
- PCMCIA interface

**High-Bandwidth Memory Buses**

- 100 MHz SDRAM controller (@400 MHz)
- SRAM/Flash EPROM controller

**Caches**

- 16 KB non-blocking data cache
- 16 KB instruction cache
- Instruction and data caches are 4-way set associative
- Write-back with read-allocate
- Cache Management Features:
  — Programmable allocation policy
  — Line locking
- Prefetch instructions (instruction and data)
- High speed access to on-chip buses

**MMU**

- Instruction and data watch registers for software break-points
- Separate interrupt exception vector
- TLB Features:
  — 32 dual-entry fully-associative
  — Variable page sizes: 4 KB to 16 MB
  — 4-entry ITB

**Low System Power**

- Core / Power:
  — 333 MHz / 400 mW
  — 400 MHz/ 700 mW
  — 500 MHz/ 1.2 W
- Power-Saving Modes:
  — Idle
  — Sleep
- Pseudo-static design to 0 Hz

**Package**

- 424 BGA (Ball Grid Array), 19x19 mm

**Operating System Support**

- Microsoft® Windows® CE
- Linux
- VxWorks

**Development Tool Support**

- Complete MIPS32™ Compatible Tool Set
- Numerous 3rd-Party Compilers, Assemblers and Debuggers

# 2 CPU

The Au1500 CPU core is a unique implementation of the MIPS32 instruction set architecture (ISA) designed for high frequency and low power. This chapter provides information on the implementation details of this MIPS32 compliant core.

The full description of the MIPS32 architecture is provided in the “MIPS32$^{TM}$ Architecture For Programmers” documentation, available from MIPS Technologies, Inc. The information contained in this chapter supplements the MIPS32 architecture documentation.

## 2.1 Core

The Au1500 CPU core (Au1) is a high performance, low power implementation of the MIPS32 architecture. Figure 2-1 shows a block diagram of the core.

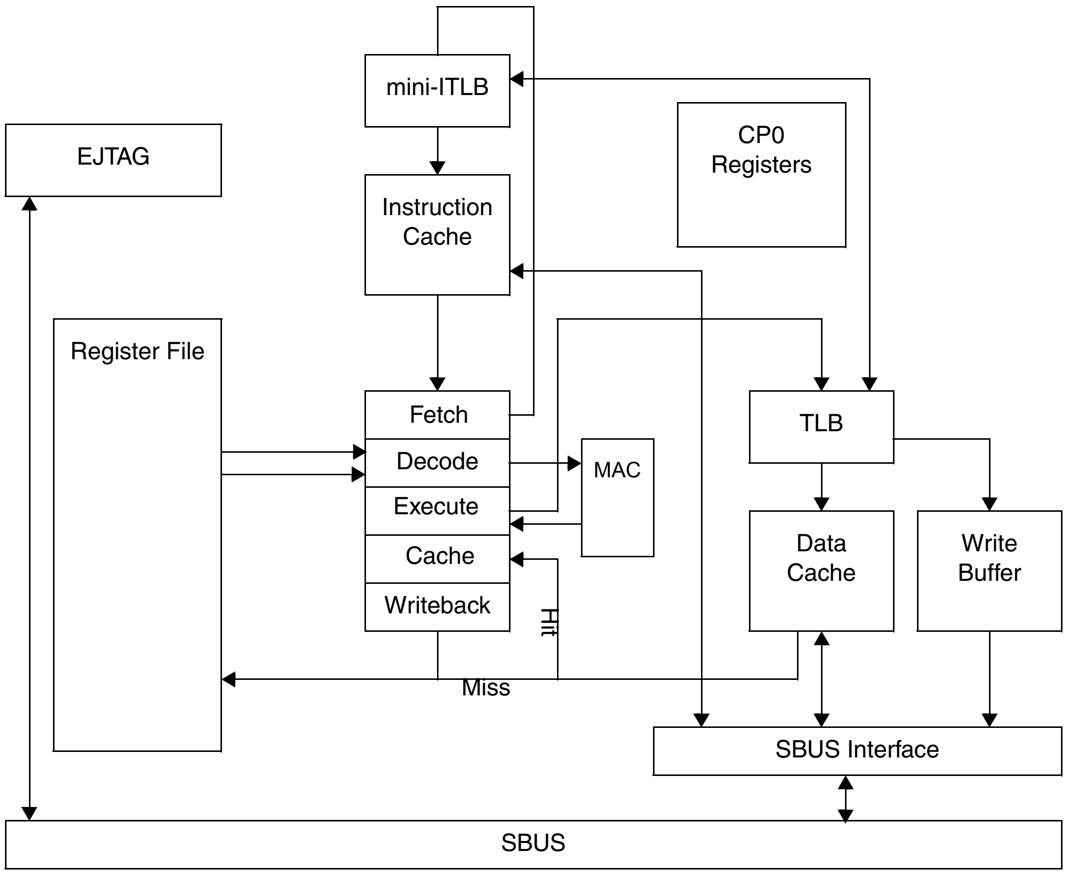


**Figure 2-1. Au1 Core Diagram**

The Au1 core contains a five-stage pipeline. All stages complete in a single cycle when data is present. All pipeline hazards and dependencies are enforced by hardware interlocks so that any sequence of instructions is guaranteed to execute correctly. Therefore, it is not necessary to pad legacy MIPS hazards (such as load delay slots and coprocessor accesses) with NOPs.

The general purpose register file has two read ports and one write port. The write port is shared with data cache loads and the pipeline Writeback stage.

### 2.1.1 Fetch Stage

The Fetch stage retrieves the next instruction from the instruction cache, where it is passed to the Decode stage. If the instruction is not present in the instruction cache, then the fetch address is forwarded to the virtual memory unit in order to fulfill the request. Instruction fetch stalls until the next instruction is available.

### 2.1.2 Decode Stage

The Decode stage prepares the pipeline for executing the instruction. In the Decode stage, the following occur in parallel:

- The instruction is decoded.
- Control for the instruction is generated.
- Register data is read.
- The branch target address is generated.
- The load/store address is generated.

Instructions stall in the Decode stage if dependent data or resources are not yet available. At the end of the Decode stage a new program counter value is sent to the Fetch stage for the next instruction fetch cycle.

### 2.1.3 Execute Stage

In the Execute stage, instructions that do not access memory are processed in hardware (shifters, adders, logical, comparators, etc.). Most instructions complete in a single cycle, but a few require multiple cycles (**CLO**, **CLZ**, **MUL**).

The virtual address calculation begins in the Decode stage so that physical address calculation can complete in the Execute stage, in time to initiate the access to the data cache in the Execute stage. If the physical address misses in the TLB, a TLB exception is posted.

Multiplies and divides are forwarded to the Multiply Accumulate unit. These instructions require multiple cycles to execute and operate mostly independent of the main five-stage pipeline.

All exception conditions (arithmetic, TLB, interrupt, etc.) are posted by the end of the Execute stage so that exceptions can be signalled in the Cache stage.

### 2.1.4 Cache Stage

In the Cache stage, load and store accesses complete.

Loads that hit in the data cache obtain the data in the Cache stage. If a load misses in the data cache, or is to a non-cacheable location, then the request is sent to the System Bus (SBUS) to be fulfilled. Load data is forwarded to dependent instructions in the pipeline.

Stores that hit in the data cache are written into the cache array. If a store misses in the data cache, or is to a non-cacheable location, then the store is sent to the write buffer.

If any exceptions are posted, an exception is signaled and the Au1 core is directed to fetch instructions at the appropriate exception vector address.

### 2.1.5 Writeback Stage

In the Writeback stage, results are posted to the general purpose register file, and forwarded to other stages as needed.

### 2.1.6 Multiply Accumulate Unit

The Multiply Accumulate unit (MAC) executes all multiply and divide instructions. The MAC is composed of a 32x16 bit pipelined array multiplier that supports early out detection, divide block, and the HI and LO registers used in calculations.

The MAC operates in parallel with the main five-stage pipeline. Instructions in the main pipeline that do not have dependencies on the MAC calculations execute simultaneously with instructions in the MAC unit.

A multiply calculation of 16x16 or 32x16 bits can complete in one cycle. The 32x16 bit multiply must have the sign-extended 16-bit value in register operand rt of the instruction.

32x32 bit multiplies may be started every other CPU cycle. The 32x32 multiplies will complete in two cycles if the results are written to the general purpose registers.

If the results are written to the HI/LO registers then three cycles are required for 16x16 and 32x16 bit multiplies. 32x32 bit multiplies that use HI/LO will complete in four cycles.

Divide instructions complete in a maximum of 35 cycles.

## 2.2 Caches

The Au1 core contains independent, on-chip 16KB instruction and data caches. As shown in Figure 2-2, each cache contains 128 sets and is four-way set associative with 32 bytes per way (cache line).

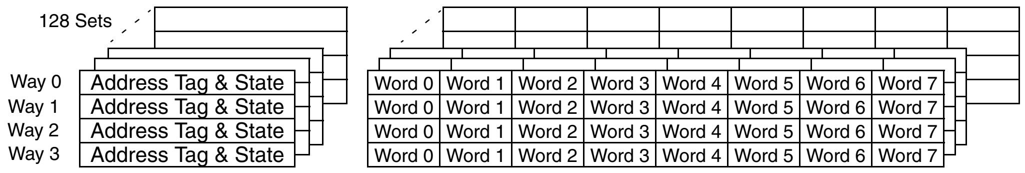


**Figure 2-2. Cache Organization**

A cache line is tagged with a 20-bit physical address, a lock bit, and a valid bit. Data cache lines also include coherency and dirty status bits. The physical address tag contains bits 31:12 of the physical address; as such, physical addresses in which bits 35:32 are non-zero must be mapped non-cacheable.

**Cache Line State**

| Bit | 31:12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Physical Address Tag | | | | | | | | | D | S | L | V |

A cache line address is always 32-byte aligned. The cache is indexed with the lower, untranslated bits (bits 11:5) of the virtual address, allowing the virtual-to-physical address translation and the cache access to occur in parallel.

**Cache Address Decode**

| Bit | 31:12 | 11:5 | 4:0 |
|---|---|---|---|
| | Virtual/Physical Address | Set Select | Byte Select |

### 2.2.1 Cache Line Replacement Policy

In general, the caches implement a least recently used (LRU) replacement policy. Each cache set maintains true LRU status bits (MRU, nMRU and LRU) to determine which cache line is selected for replacement. However, software can influence which cache line is replaced by marking memory pages as *streaming*, or by *locking* lines in the cache.

### 2.2.2 Cache Line Locking Support

The **CACHE** instruction is used to lock individual lines in the cache. A locked line is not subject to replacement. All four lines in a set can not be locked at once; at least one line is always available for replacement. To *unlock* individual cache lines use the **CACHE** instruction with a ‘hit invalidate’ command opcode. See Section 2.2.5 "Cache Management" on page 19 for further discussion of the **CACHE** instruction.

### 2.2.3 Cache Streaming Support

Streaming is typically characterized as the processing of a large amount of transient instructions and/or data. In traditional cache implementations (without explicit support for streaming), transient instructions and/or data quickly displace useful, recently used items in the cache. This yields poor utilization of the cache and results in poor system performance.

The Au1 caches explicitly support streaming by placing instructions and/or data marked as streaming into way 0 of the cache. This method ensures that streaming does not purge the cache(s) of useful, recently used items, while permitting transient instructions and/or data to be cached. The CCA bits in the TLB entry indicate if a page contains streaming instructions and/or data. In addition, the **PREF** instruction is available to software to allow data to be placed in the data cache in advance of its use.

### 2.2.4 Cache Line Allocation Behavior

When an instruction fetch misses in the instruction cache, or a data load misses in the data cache, a burst fill operation is performed to fill the cache line from memory. The cache line is selected by the following algorithm:

```
MRU is most recently used
nMRU is next most recently used
nLRU is next least recently used
LRU is least recently used

Cache Miss:
if (Streaming CCA=6) then Replacement = 0,
else if (LRU is !Valid or !Locked) then Replacement = LRU
else if (nLRU is !Valid or !Locked) then Replacement = nLRU
else if (nMRU is !Valid or !Locked) then Replacement = nMRU
else Replacement = MRU

Cache Hit:
new MRU = Hit Way
```

In short, the LRU selection is true LRU but with the following priorities:

1) Streaming: cache misses are forced to way 0.

2) Locking: cache misses follow policy above and set Lock bit.

3) Normal: true LRU replacement.

Table 2-1 summarizes cache line allocation for misses, as well as cache hit behavior. The table also shows how prefetching and cache locking affect the cache for hits and misses.

**Table 2-1. Cache Line Allocation Behavior**

| Operation | Hit | Miss |
|---|---|---|
| **NORMAL** | | |
| Data load, Instruction fetch | Read data from whichever cache line contains the address. | Allocate and fill cache line; clear Lock bit; return read data. |
| Data store | Write data to whichever cache line contains the address. | Send to the write buffer. |
| **STREAMING (CCA=6)** | | |
| Data load, Instruction fetch | Read data from whichever cache line contains the address. | Allocate and fill cache line in Way 0; maintain Lock bit; return read data. |
| Data store | Write data to whichever cache line contains the address. | Send to the write buffer. |
| PREF (data prefetch instruction with 0x4 hint) | No action taken—data remains in current cache line. | Allocate and fill cache line in Way 0; maintain Lock bit. |
| **LOCKING** | | |
| CACHE 0x1D/0x1C (cache management instruction with Lock opcode) | Set Lock bit in whichever cache line contains the address. | Allocate and fill cache line; set Lock bit. |

### 2.2.5 Cache Management

The caches are managed with the **CACHE** instruction. The effect of the **CACHE** instruction is immediately visible to subsequent data accesses. Table 2-2 shows the cache operations, including the opcode to direct the operation to either the instruction cache or data cache. (An “n/a” indicates that the operation is not applicable.)

**Table 2-2. Cache Operations**

| Operation | CACHE[20..18] Encoding | Opcode for Instruction Cache | Opcode for Data Cache |
|---|---|---|---|
| Index Invalidate | 000 | 0x00 | 0x01 (with writeback) |
| Index Load Tag | 001 | 0x04 | 0x05 |
| Index Store Tag | 010 | 0x08 | 0x09 |
| Hit Invalidate | 100 | 0x10 | 0x11 |
| Fill | 101 | 0x14 | N/A |
| Hit Writeback and Invalidate | 101 | N/A | 0x15 |
| Hit Writeback | 110 | N/A | 0x19 |
| Fetch and Lock | 111 | 0x1C | 0x1D |

These cache operations permit initialization, locking/unlocking and management of the caches.

### 2.2.6 Cache Coherency Attributes (CCA)

The cache coherency attributes (CCA) field in Config0[K0] and in the TLB determine the cache-ability of accesses to memory. Cached accesses (except CCA = 4) are performed critical-word-first to improve performance. The Au1 implements the following:

**Table 2-3. Cache Coherency Attributes (CCA)**

| CCA | CCA (3 Bits) | Description |
|---|---|---|
| 0, 1 | 00x | Reserved (undefined). |
| 2 | 010 | Uncached, non-mergeable, non-gatherable. Required by the MIPS32 architecture. In addition, data is not merged within the write buffer to achieve a truly uncached effect. |
| 3 | 011 | Cached, mergeable, gatherable. |
| 4 | 100 | Cached, mergeable, gatherable (word 0 first). Word 0 is always accessed first; that is, the cache line is accessed in word order (word 0, 1...7). |
| 5 | 101 | Cached, mergeable, gatherable. |
| 6 | 110 | Cached, mergeable, gatherable, streaming. Instructions and/or data are placed into way 0. |
| 7 | 111 | Uncached, mergeable, gatherable. Even though data is not cached, data stores sent to the write buffer are subject to merging and gathering in the write buffer. |

### 2.2.7 Instruction Cache

The instruction cache is a 16 KB, four-way set associative cache. The instruction cache services instruction fetch requests from the Fetch stage of the pipeline.

An instruction cache line state consists of a 20-bit physical address tag, a lock bit (L) and a valid bit (V).

**Instruction Cache line state**

| Bit | 31–12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Physical Address Tag | | | | | | | | | | | L | V |

#### 2.2.7.1 Instruction Cache Initialization and Invalidation

Out of reset, all instruction cache lines are invalidated; thus the instruction cache is ready for use.

To invalidate the instruction cache in software, a loop of index invalidate **CACHE** instructions for each of the lines in the cache invalidates the cache.

```
    li t0,(16*1024) # Cache size
    li t1,32 # Line size
    li t2,0x80000000 # First KSEG0 address
    addu t3,t0,t2 # terminate address of loop
loop:
    cache 0,0(t2) # Icache indexed invalidate tag
    addu t2,t1 # compute next address
    bne t2,t3,loop
    nop
```

#### 2.2.7.2 Instruction Cache Line Fills

If an instruction fetch address hits in the instruction cache, the instruction word is returned to the Fetch stage. If the fetch address misses in the cache, and the address is cacheable, then the instruction cache performs a burst transfer from the memory subsystem to fill a cache line, and returns the instruction word to the Fetch stage.

The instruction cache line is selected by the replacement policy described in Section 2.2.1 "Cache Line Replacement Policy" on page 17.

#### 2.2.7.3 Instruction Cache Coherency

The instruction cache does not maintain coherency with the data cache. Coherency between the instruction cache and the data cache is the responsibility of software. However, the data cache snoops during instruction cache line fills.

Maintaining coherency is important when loading programs into memory, creating exception vector tables, or for self-modifying code. In these circumstances, memory is updated with new instructions using store instructions which places the new instructions in the data cache, but not in the instruction cache (thus the instruction cache may contain old instructions).

To maintain coherency, software must use the **CACHE** instruction to invalidate the modified range of program addresses in the instruction cache. Because the data cache snoops during instruction cache line fills, it is not necessary to writeback the data cache prior to invalidating the instruction cache. An instruction fetch to the newly loaded/modified program correctly fetches the new instructions.

#### 2.2.7.4 Instruction Cache Control

The cache-ability of instructions is controlled by three mechanisms:

- Config0[K0] field
- The CCA bits in the TLB
- The **CACHE** instruction

The Config0[K0] field contains a cache coherency attribute (CCA) setting to control the cache-ability of KSEG0 region. At reset, this field defaults to CCA=3 (cacheable).

The CCA bits in the TLB entry control the cache-ability of the KUSEG, KSEG2, and KSEG3 regions. Each TLB entry specifies a CCA setting for the pages mapped by the TLB.

The **CACHE** instruction manages the caches, including the ability to lock lines in the cache. Valid instruction cache operations are the following:

- Index Invalidate
- Index Load Tag
- Index Store Tag
- Hit Invalidate
- Fill
- Fetch and Lock

The effect of the **CACHE** instruction is visible to subsequent instructions not already in the pipeline. Instructions already in the fetch and decode stages of the pipeline are not affected by a cache operation on the instruction cache.

### 2.2.8 Data Cache

The data cache is a 16 KB four-way set associative write-back cache. Data cache accesses are distributed across the Execute and Cache pipeline stages.

A data cache line state consists the 20-bit physical address tag, a dirty bit (D), a coherency bit (S), a lock bit (L) and a valid bit (V).

**Data cache line state**

| Bit 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Physical Address Tag | | | | | | | | | | | | | | | | | | | | | | | | | | | | D | S | L | V |

The data cache employs a read-allocate policy. Cache lines can be replaced on loads, but not on stores. Stores that miss in the data cache are forwarded to the write buffer.

The data cache supports hit-under-miss for one outstanding miss. If an access misses in the data cache, the data cache services the next access while the memory subsystem provides the data for the missed access. If the next access hits in the data cache, the data is available immediately; otherwise the cache stalls the access until the first access completes.

#### 2.2.8.1 Data Cache Initialization and Invalidation

Out of reset, all data cache lines are invalidated; thus the data cache is ready for use.

To invalidate the data cache in software, a loop of indexed writeback invalidate **CACHE** instructions for each of the lines in the cache invalidates the cache.

```
    li t0,(16*1024) # Cache size
    li t1,32 # Line size
    li t2,0x80000000 # First KSEG0 address
    addu t3,t0,t2 # terminate address of loop
loop:
    cache 1,0(t2) # Dcache indexed invalidate tag
    addu t2,t1 # compute next address
    bne t2,t3,loop
    nop
```

#### 2.2.8.2 Data Cache Line Fills

A data cache access is initiated in the Execute stage which allows a cache hit or miss indication and all exceptions to be signaled early in the Cache stage. If the data address hits in the data cache, the data is available in the Cache stage. If the data address misses in the data cache, and the address is cacheable, the data cache performs a burst fill to a cache line, forwarding the critical word to the Cache stage.

The data cache line is selected by the replacement policy described in Section 2.2.1 "Cache Line Replacement Policy" on page 17. If the line selected contains modified data (cache line is valid and has its dirty bit set by a store hit), then the cache line is moved to a cast-out buffer, the cache line is filled from memory and the load request fulfilled, and then the cast-out buffer is written to memory.

#### 2.2.8.3 Data Cache Coherency

The data cache snoops coherent SBUS transactions to maintain data coherency with other SBUS masters (i.e. DMA). If a coherent read transaction on the SBUS hits in the data cache, the data cache provides the data. If a coherent write transaction on the SBUS hits in the data cache, the data cache updates its internal array with the data. If a coherent transaction (read or write) misses in the data cache, the data cache array is unchanged by the transaction.

Loads and stores which hit in the data cache can bypass previous stores in cacheable regions. The read-allocate data cache policy forwards store-misses to the write buffer. Subsequent loads and stores which hit in the data cache, and to a different cache line address than store-misses, are fulfilled immediately (while store-misses may still be in the write buffer). However, if a load address hits in a cache-line address of an item in the write buffer, the load is stalled until the write buffer commits the corresponding store.

The data cache also maintains coherency with other caching masters. When a load is serviced from another caching master, both caching masters set the shared bit for the affected cache line. Then if a store occurs to a data cache line with the shared bit set, the cache line address is broadcast on the SBUS to invalidate cache lines in other caching masters that contain the same address.

The data cache is single-ported; therefore transactions on the SBUS are prioritized over accesses by the core. However, the data cache design prevents the SBUS from saturating the data cache indefinitely, which ensures that the core can make forward progress.

When changing the CCA encoding in Config0[K0] or the TLB to a different CCA encoding, software must ensure that data integrity is not compromised by first pushing modified (dirty) data to memory within the page. This is especially important when changing from a coherent CCA encoding to a non-coherent CCA encoding.

#### 2.2.8.4 Data Cache Control

The cache-ability of data accesses is controlled by four mechanisms:

- Config0[K0] field
- The CCA bits in the TLB
- The **CACHE** instruction
- The **PREF** instruction

The Config0[K0] field contains a cache coherency attribute (CCA) setting to control the cache-ability of KSEG0 region. At reset, this field defaults to 011b, cacheable.

The CCA bits in the TLB entry control the cache-ability of the KUSEG, KSEG2, and KSEG3 regions. Each TLB entry specifies a CCA setting for the pages mapped by the TLB.

The **CACHE** instruction manages the caches; including the ability to lock lines in the cache. Valid data cache operations are:

- Index Writeback Invalidate
- Index Load Tag
- Index Store Tag
- Hit Invalidate (unlocks)
- Hit Writeback and Invalidate
- Hit Writeback
- Fetch and Lock

The effect of the **CACHE** instruction is immediately visible to subsequent data accesses.

The **PREF** instruction places data into the data cache. The following prefetch hints are implemented:

- 0x00 - Normal load
- 0x04 - Streaming load

The streaming load hint directs the data be placed into way 0 of the data cache (even if the line is locked), thus permitting transient data to be cached and non-transient data to remain in the cache for improved performance. Data cache streaming support combined with the **PREF** instruction enhances multimedia processing.

## 2.3 Write Buffer

The Au1 write buffer is depicted in Figure 2-3. All non-cacheable processor stores and data cache store-misses (the data cache is a read-allocate policy) are routed through the write buffer.

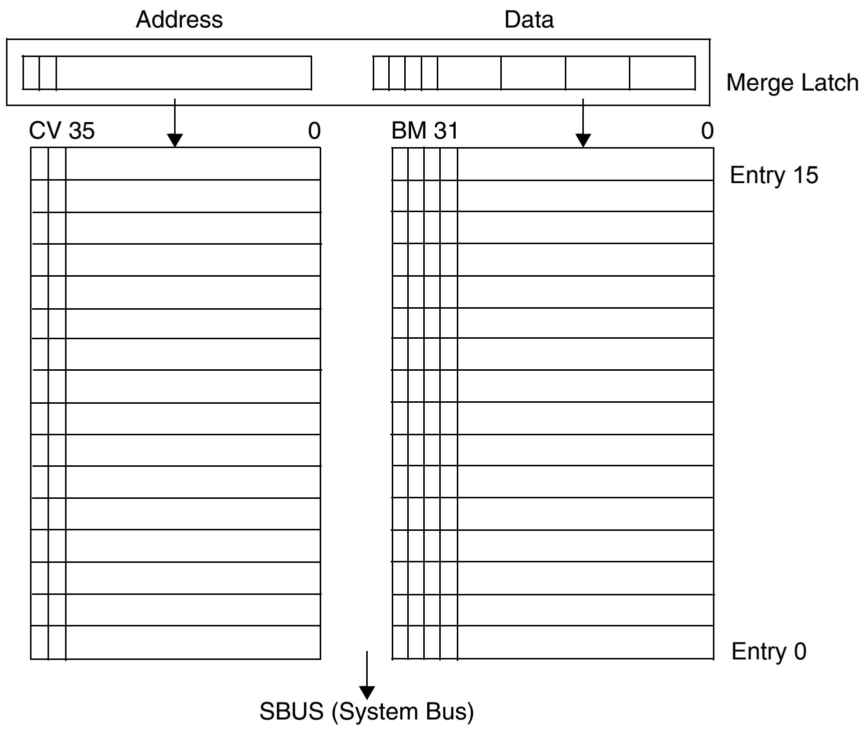


**Figure 2-3. Au1 Write Buffer**

The write buffer is a 16-word deep first-in-first-out (FIFO) queue. All processor stores arrive first at the merge latch, where merging and gathering decisions are performed, and then travel through the queue. The write buffer arbitrates for the SBUS to perform consolidated transfers to the main memory.

A write buffer FIFO entry contains the address (word address), the data and associated byte masks (BM), and two control bits. The four BM bits indicate which bytes within the word contain valid data. The two control bits are the Valid bit which indicates if the entry is valid, and the Closed (C) bit. When a C bit is set, the write buffer initiates a request to the SBUS so that it can transfer data to memory. The circumstances for which the C bit is set are described below.

The write buffer is capable of variable-length burst writes to memory. The length can vary from one word to eight words, and is determined by the C bits in the write buffer. During each beat of the burst, the appropriate bytes to write are selected from the corresponding byte masks. As each entry is written to memory, it is popped from the FIFO, advancing each entry in the FIFO by one. In other words, entry 0 is always presented to the SBUS for writing.

When the write buffer has at least one empty entry, processor stores do not stall, thus improving processor performance.

The write buffer is disabled by setting Config0[WD] to 1. In this instance, all non-cacheable and data cache store-misses stall until the write completes. The remaining description of the write buffer operation assumes Config0[WD] is 0. Out of reset, Config0[WD] is 0.

### 2.3.1 Merge Latch

All processor stores first arrive at the merge latch. Logic within the merge latch decides what action to take with the incoming data.

1) The incoming address is the *same word address* as the merge latch address. This case is for Merging, which occurs within the merge latch itself.

2) The incoming word address is *sequentially adjacent* to the merge latch word address (incoming address is merge latch address + 4). This case is for Gathering. The merge latch contents are propagated to the FIFO with the C bit cleared for this entry.

3) Neither 1 nor 2 is true. The merge latch contents are propagated to the FIFO with the C bit set for this entry.

If the merge latch contents are propagated to the FIFO, the incoming address and data are placed in the merge latch for future comparisons. Furthermore, if the incoming address is the last word address of the maximum burst line size (the least significant 5 bits are 0x1C), then the C bit is set.

### 2.3.2 Write Buffer Merging

Write buffer merging combines stores destined for the *same word address*. Merging places the incoming data into the appropriate data byte(s) within the merge latch.

Write buffer merging is particularly useful for sequential, incremental address write operations, such as string operations. With write buffer merging, the writes are merged into 32-bit writes which reduces the number of accesses to the memory and increases the effective throughput to main memory.

This example demonstrates merging: these five byte writes occur in sequence:

```
0x00001000 = 0xAB
0x00001001 = 0xCD
0x00001002 = 0xDE
0x00001003 = 0xEF
0x00001002 = 0xBE
```

After the first four writes, the data in the merge latch contains 0xABCDDEEF. However, after the fifth write, the merge latch data now contains 0xABCDBEEF.

So long as the incoming word address is the same as the merge latch word address, the data can change without a processor stall or access to memory.

Write buffer merging is controlled by the Config[NM] bit and the TLB[CCA] setting. When Config0[NM] is 1 or TLB[CCA] is 2, the merge latch does not perform merging. Conversely, Config0[NM] is 0 or TLB[CCA] is not 2 enables merging. Out of reset, Config0[NM] is 0.

**Note:** Merging takes place *only* in the merge latch. As such, writes to an address which are in the FIFO (but not in the merge latch) do not merge. In the example below, writes to 0x0001000 and 0x0001002 do not merge because the intervening write to address 0x00001005 is not in the same word address which caused 0x00001000 to leave the merge latch.

```
000 = 0xAB
005 = 0xCD
002 = 0xDE
```

### 2.3.3 Write Buffer Gathering

Write buffer gathering combines *sequentially adjacent* word addresses for burst transfers to the main memory. When a C bit is set, all queue entries from zero (0) up to and including the entry with its C bit set (N) are written to main memory in a single burst.

Write buffer gathering is particularly useful for sequential, incremental address store operations, such as string operations. With write buffer gathering, the stores are combined into bursts up to 32-bytes (eight words) in length which reduces the number of accesses to the memory and increases effective throughput.

Here is an example of an eight-word burst. The burst could result from code which sequentially writes words (optimized memcpy(), for example). These eight word writes occur in sequence:

```
0x00001000
0x00001004
0x00001008
0x0000100C
0x00001010
0x00001014
0x00001018
0x0000101C
```

The entries corresponding to word addresses 0x00001000 through 0x00001018 have C bit set to zero. When address 0x0000101C arrives, its C bit is set. When the write buffer is granted the SBUS, it bursts all eight entries to main memory.

Here is an example of two-word burst. This burst may be typical of application software. These four word writes occur in sequence:

```
0x00001000
0x00001004
0x0000100C
0x00001008
```

The C bit is cleared for the 0x00001000 entry and is set for the 0x00001004 entry. These two words are then burst to main memory. The 0x0000100C entry also has its C bit set, and is written to memory. The 0x00001008 will reside in the merge latch until displaced by a subsequent store.

### 2.3.4 Write Buffer Reads

When a read from memory is initiated, the read cache-line address (A35..A5) is compared against all cache-line addresses in the write buffer. If the read cache-line address matches a write buffer cache-line address, the read is stalled. The write buffer then flushes entries to memory until the read address no longer matches a write buffer cache-line address. The read is then allowed to complete. The write buffer ensures data integrity by not allowing reads to bypass writes.

### 2.3.5 Write Buffer Coherency

Non-cacheable stores and/or data cache store-misses reside in the write buffer, possibly indefinitely. Furthermore, the write buffer does not snoop SBUS transactions (e.g. integrated peripheral DMA engines). To ensure the write buffer contents are committed to memory, a **SYNC** instruction must be issued.

Issuing a **SYNC** instruction prior to enabling each DMA transfer from memory buffers and/or structures is necessary. Without the **SYNC**, the DMA engine may retrieve incomplete buffers and/or structures (the remainder of which may be in the write buffer).

Issuing a **SYNC** instruction after a store to an I/O region where stores have side effects is necessary. Without the **SYNC** instruction, the store may not leave the write buffer to achieve the side effects (e.g. clearing an interrupt acknowledge bit).

Note that a read access does not guarantee a complete write buffer flush since the write buffer flushes as few entries as necessary until the read address no longer matches an address in the write buffer.

## 2.4 Virtual Memory

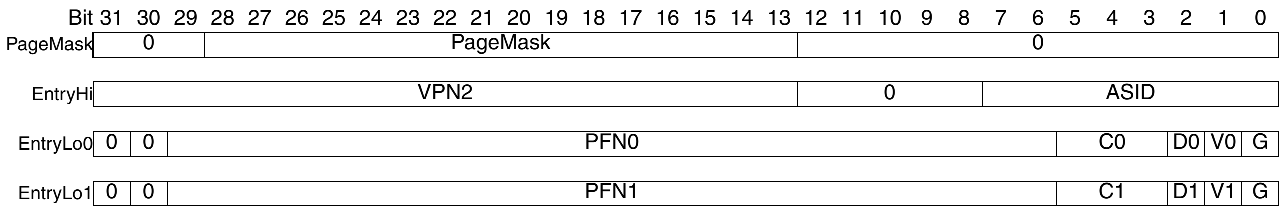
The Au1 implements a TLB-based virtual address translation unit which is compliant with the MIPS32 specification. This scheme is similar to the R4000 TLB and CP0 implementation. The “MIPS32 Architecture For Programmers Volume III” contains all the information relevant to a TLB-based virtual address translation unit.

The virtual address translation architecture is composed of a main 32-entry fully associative TLB array. To improve instruction fetch performance, a 4-entry fully associative instruction TLB is implemented. This miniature instruction TLB is fully coherent with the main TLB array and is completely transparent to software.

Each TLB entry maps a 32-bit virtual address to a pair of 36-bit physical addresses. The page size of a TLB entry is variable under software control, from 4 KB to 16 MB.

A TLB entry is described below.

**TLB Entry**


Bit 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
PageMask
0
PageMask
0
EntryHi
VPN2
0
ASID
EntryLo0
0
0
PFN0
C0
D0
V0
G
EntryLo1
0
0
PFN1
C1
D1
V1
G

The size of the page(s) that the TLB entry translates is determined by PageMask. The valid values for PageMask range from 4 KB to 16 MB, according to Table 2-4.

**Table 2-4. Values for Page Size and PageMask Register**

| Page Size | PageMask Register | Bits [28:13] | PFN Select Bit |
|---|---|---|---|
| 4 KB | 0x00000000 | 0000000000000000 | 12 |
| 16 KB | 0x00006000 | 0000000000000011 | 14 |
| 64 KB | 0x0001E000 | 0000000000001111 | 16 |
| 256 KB | 0x0007E000 | 0000000000111111 | 18 |
| 1 MB | 0x001FE000 | 0000000011111111 | 20 |
| 4 MB | 0x007FE000 | 0000001111111111 | 22 |
| 16 MB | 0x01FFE000 | 0000111111111111 | 24 |

The PageMask determines the number of significant bits in the 32-bit address generated by the program (either as a load/store address or an instruction fetch address). The upper, significant bits of the program address are compared against the upper, significant bits of VPN2. When an address match occurs, the even/odd PFN select bit of the program address selects either PFN0 (even) or PFN1 (odd) as the upper bits of the resulting 36-bit physical address.

The TLB mechanism permits mapping a larger, 36-bit physical address space into the smaller 32-bit program address space. The Au1 implements an internal 36-bit physical address SBUS which is then decoded by integrated peripherals, and by chip-selects for external memories and peripherals.

The cache coherency attributes (CCA) of the physical page are controlled by the TLB entry. The valid values are described in Table 2-3 on page 19. In general, I/O spaces require a non-cacheable setting, whereas memory can utilize a cacheable setting.

**Note:** Physical addresses in which address bits [35:32] are non-zero must be mapped non-cached (CCA = 2).

The TLB array is managed completely by software. Software can implement a TLB replacement algorithm that is either random (via the **TLBWR** instruction) or deterministic (via the **TLBWI** instruction). Hardware is available to segment the TLB via the Wired register so different replacement strategies can be used for different areas of the TLB.

## 2.5 Exceptions

The Au1 core implements a MIPS32 compliant exception scheme. The scheme consists of the exception vector entry points in both KSEG0 and KSEG1, and the exception code (ExcCode) encodings to determine the nature of the exception.

### 2.5.1 Exception Causes

The nature of an exception is reported in the Cause[ExcCode] field. The Au1 core can generate the following exceptions:

**Table 2-5. Cause[ExcCode] Encodings**

| ExcCode | Mnemonic | Description |
|---|---|---|
| 0 | Int | Interrupt |
| 1 | Mod | TLB modification exception |
| 2 | TLBL | TLB exception (load or instruction fetch) |
| 3 | TLBS | TLB exception (store) |
| 4 | AdEL | Address error exception (load or instruction fetch) |
| 5 | AdES | Address error exception (store) |
| 6 | IBE | Bus error exception (instruction fetch) |
| 7 | DBE | Bus error exception (data reference: load or store) |
| 8 | Sys | Syscall exception |
| 9 | Bp | Breakpoint exception |
| 10 | RI | Reserved instruction exception |
| 11 | CpU | Coprocessor Unusable exception |
| 12 | Ov | Arithmetic Overflow exception |
| 13 | Tr | Trap exception |
| 23 | WATCH | Reference to Watchpoint address |
| 24 | MCheck | Machine Check (duplicate TLB entry) |

The Au1 core does not implement hardware floating-point. As a result, all floating-point instructions generate the Reserved Instruction (RI) exception, which permits floating-point operations to be emulated in software.

In addition, the Au1 core does not recognize Soft Reset, Non-Maskable Interrupt (NMI), or Cache Error exception conditions.

### 2.5.2 Interrupt Architecture

The Au1 core implements a MIPS32 compliant interrupt mechanism in which eight interrupt sources are presented to the core. Each interrupt source is individually maskable to either enable or disable the core from detecting the interrupt. Interrupts are generated by software, integrated interrupt controllers, performance counters and timers, as noted in Table 2-6.

**Table 2-6. CPU Interrupt Sources**

| Interrupt Source | CP0 Cause Register Bit | CP0 Status Register Bit |
|---|---|---|
| Software Interrupt 0 | 8 | 8 |
| Software Interrupt 1 | 9 | 9 |
| Interrupt Controller 0:<br>Request 0<br>Request 1 | <br>10<br>11 | <br>10<br>11 |
| Interrupt Controller 1:<br>Request 0<br>Request 1 | <br>12<br>13 | <br>12<br>13 |
| Performance Counters | 14 | 14 |
| Count/Compare | 15 | 15 |

All interrupt sources are equal in priority; that is, the interrupt sources are not prioritized in hardware. As a result, software determines the relative priority of the interrupt sources. When Cause[ExcCode]=0, software must examine the Cause[IP] bits to determine which interrupt source is requesting the interrupt.

For more information on Interrupt Controller 0 and 1 see Section 6.0 "Interrupt Controller" on page 95.

## 2.6 MIPS32™ Instruction Set

The Au1 core implements the instruction set defined in “MIPS32 Architecture For Programmers Volume II: The MIPS32 Instruction Set”. The floating-point instructions are not implemented in the Au1 core, but may be emulated in software.

The MIPS32 ISA is characterized as a combination of the R3000 user level instructions (MIPSII) and the R4000 memory management and kernel mode instructions (32-bit MIPSIII).

### 2.6.1 CACHE Instruction

The **CACHE** instruction permits management of the Au1 instruction and data caches. The valid operations are listed in Table 2-2 "Cache Operations" on page 19.

For *data* cache operations, the effect of the **CACHE** instruction is immediately visible to subsequent data accesses. However, for *instruction* cache operations, the effect of the **CACHE** instruction is not visible to subsequent instructions already in the Au1 core pipeline. Therefore, care should be exercised if modifying instruction cache lines containing the **CACHE** and subsequent instructions.

When issuing the **CACHE** instruction with indexed operations (Index Invalidate, Index Load Tag and Index Store Tag) the format of the effective address is as follows:

**CACHE Index Operation Address Decode**

| Bit 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 |
|---|---|---|---|
| 0x8000 | Way | Set/Index | Byte Select |

The effective address base should be 0x80000000 (KSEG0) to avoid possible TLB exceptions, and place zeros in the remainder of the effective address. The format correlates to a 16KB cache that is 4-way set associative with 128 sets and 32-byte line size.

Software must not use the Index Store Tag **CACHE** operation to change the Dirty, Lock and Shared state bits. To set the Lock bit, software must use the Fetch and Lock **CACHE** operation.

The Index Load Tag and Index Store Tag **CACHE** operations utilize CP0 registers DTag, DData, ITag and IData. The format of data for Index Tag operations is depicted in the description of these registers.

**CACHE** operations that require an effective address (i.e., not the Index operations) do not generate the Address Error Exception or trigger data watchpoint exceptions.

### 2.6.2 PREF Instruction

The **PREF** instruction prefetches data into the data cache. Data is prefetched to improve algorithm performance by placing the data in the cache in advance of its use, thus minimizing stalls due to data cache load misses. See also Section 2.2.8.4 "Data Cache Control" on page 22 for more on how to use **PREF**.

If the effective address computed by the **PREF** instruction does not translate in the TLB (i.e. the address would cause a TLBL exception), no exception is generated and the cache is unchanged.

The Au1 core implements the following **PREF** instruction hints:

- 0x00 - Normal load
- 0x04 - Streaming load

A **PREF** instruction using any other hint value becomes a **NOP** for the Au1 core.

### 2.6.3 WAIT Instruction

The **WAIT** instruction places the Au1 core in one of two low power modes: IDLE0 and IDLE1. The low power mode is encoded in the **WAIT** instruction bits 24:6 (implementation-dependent code). A value of 0 selects IDLE0, and the value 1 selects IDLE1. Other values are not supported and must not be used.

In the IDLE0 low power mode, the Au1 core stops clocks to all possible core units but continues to snoop the SBUS to maintain data coherency.

In the IDLE1 low power mode, the Au1 core stops clocks to all possible core units, including the data cache, so data coherency is no longer maintained.

In either Idle mode, the general purpose registers and the CP0 registers are preserved, so that when Idle mode is exited by an appropriate event, the Au1 core resumes processing instructions in exactly the same context as prior to entering Idle mode.

To enter the low power mode, the **WAIT** instruction must be followed by at least four NOPs, and the entire instruction sequence must be fetched from the instruction cache. More specifically, if the core fetches the **WAIT** and **NOP** instructions from main memory, then the mechanisms for accessing memory will prevent the core from entering low power mode. This is the recommended code sequence:

```
    .global au1_wait
au1_wait:
    la  t0,au1_wait # obtain address of au1_wait
    cache 0x14,0(t0) # fill icache with first 8 insns
    cache 0x14,32(t0) # fill icache with next 8 insns
    sync
    nop
    wait 0
    nop
    nop
    nop
    nop
    j ra
```

When the Au1 core is in Idle mode, the Count register increments at an unpredictable rate; therefore the Count/Compare registers can not be used as the system timer tick when using the **WAIT** instruction to enter an Idle mode.

## 2.7 Coprocessor 0

Coprocessor 0 (CP0) is responsible for virtual memory, cache and system control.

The MIPS32 ISA provides for differentiation of the CP0 implementation. The Au1 core has a unique CP0 that is compliant with MIPS32 specification.

The Au1 CP0 registers are listed in Table 2-7.

**Table 2-7. Coprocessor 0 Register Definitions**

| Register Number | Sel | Register Name | Description | Compliance (Note 1) |
|---|---|---|---|---|
| 0 | 0 | Index | Pointer into TLB array | Required |
| 1 | 0 | Random | Pseudo-random TLB pointer | Required |
| 2 | 0 | EntryLo0 | Low half of TLB entry for even pages | Required |
| 3 | 0 | EntryLo1 | Low half of TLB entry of odd pages | Required |
| 4 | 0 | Context | Pointer to a page table entry | Required |
| 5 | 0 | PageMask | Variable page size select | Required |
| 6 | 0 | Wired | Number of locked TLB entries | Required |
| 7 | 0 |  | Reserved | Reserved |
| 8 | 0 | BadVAddr | Bad virtual address | Required |
| 9 | 0 | Count | CPU cycle count | Required |
| 10 | 0 | EntryHi | High half of TLB entries | Required |
| 11 | 0 | Compare | CPU cycle count interrupt comparator | Required |
| 12 | 0 | Status | Status | Required |
| 13 | 0 | Cause | Reason for last exception | Required |
| 14 | 0 | EPC | Program Counter of last exception | Required |
| 15 | 0 | PRId | Processor ID and Revision | Required |
| 16 | 0 | Config | Configuration Registers (aka Config0) | Required |
| 16 | 1 | Config1 | Configuration Register 1 | Required |
| 17 | 0 | LLAddr | Load Link Address | Optional |
| 18 | 0 | WatchLo | Data memory break point low bits | Optional |
| 18 | 1 | IWatchLo | Instruction fetch breakpoint low bits | Optional |
| 19 | 0 | WatchHi | Data memory break point high bits | Optional |
| 19 | 1 | IWatchHi | Instruction fetch breakpoint high bits | Optional |
| 20 | 0 |  | Reserved | Reserved |
| 21 | 0 |  | Reserved | Reserved |
| 22 | 0 | Scratch | Scratch register | Au1 |
| 23 | 0 | Debug | EJTAG control register | Optional |
| 24 | 0 | DEPC | PC of EJTAG debug exception | Optional |
| 25 | 0 | Reserved | Reserved | Au1 Reserved |
| 25 | 1 | Reserved | Reserved | Au1 Reserved |
| 26 | 0 |  | Reserved | Reserved |
| 27 | 0 |  | Reserved | Reserved |
| 28 | 0 | DTag | Data cache tag value | Au1 |
| 28 | 1 | DData | Data cache data value | Au1 |
| 29 | 0 | ITag | Instruction cache tag value | Au1 |
| 29 | 1 | IData | Instruction cache data value | Au1 |
| 30 | 0 | ErrorEPC | Program counter at last error | Required |
| 31 | 0 | DESave | EJTAG debug exception save register | Optional |

Note 1. A compliance of “Required” denotes a register required by the MIPS32 architecture. “Optional” denotes an optional register in the MIPS32 architecture which is implemented in the Au1 core. “Au1” denotes an optional register unique to the Au1 core. “Reserved” denotes a register that is not implemented.

### 2.7.1 Index Register (CP0 Register 0, Select 0)

The Index register is required for TLB-based virtual address translation units.

**Index** **CP0 Register 0, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | P | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | Index | | | | |
| Def. | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31 | P | Probe Failure. | R | UNPRED |
| 30:5 | Reserved | Must always write zeros, always reads zeros | R | 0 |
| 4:0 | Index | TLB Index | R/W | UNPRED |

### 2.7.2 Random Register (CP0 Register 1, Select 0)

The Random register is required for TLB-based virtual address translation units.

**Random** **CP0 Register 1, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | Random | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:5 | Reserved | Must always write zeros, always reads zeros | R | 0 |
| 4:0 | Random | TLB Random Index | R | 31 |

### 2.7.3 EntryLo0, EntryLo1 Register (CP0 Registers 2 and 3, Select 0)

The EntryLo0 and EntryLo1 registers are required for TLB-based virtual address translation units.

**EntryLo0, EntryLo1** **CP0 Registers 2 and 3, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | | PFN | | | | | | | | | | | | | | | | | | | | | | | | C | | | D | V | G |
| Def. | 0 | 0 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:30 | Reserved | Ignored on writes, returns zero on read | R | 0 |
| 29:6 | PFN | Page Frame Number. Corresponds to physical address bits 35..12. | R/W | UNPRED |
| 5:3 | C | Cache coherency attribute of the page. See Table 2-3 "Cache Coherency Attributes (CCA)" on page 19. | R/W | UNPRED |
| 2 | D | Dirty bit. | R/W | UNPRED |
| 1 | V | Valid bit | R/W | UNPRED |
| 0 | G | Global bit | R/W | UNPRED |

### 2.7.4 Context Register (CP0 Register 4, Select 0)

The Context register is required for TLB-based virtual address translation units.

**Context** **CP0 Register 4, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PTEBase | | | | | | | | | BadVPN2 | | | | | | | | | | | | | | | | | | | 0 | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:23 | PTEBase | Used by the operating system as a pointer into the current PTA array in memory. | R/W | UNPRED |
| 22:4 | BadVPN2 | Contains virtual address bits 31..13 upon a TLB exception. | R | UNPRED |
| 3:0 | Reserved | Reserved | R | 0 |

### 2.7.5 PageMask Register (CP0 Register 5, Select 0)

The PageMask register is required for TLB-based virtual address translation units.

**PageMask** **CP0 Register 5, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | | | Mask | | | | | | | | | | | | | | | | 0 | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:29 | Reserved | Ignored on write, returns zero on read. | R | 0 |
| 28:13 | Mask | The Mask field is a bit mask in which a “1” bit indicates that the corresponding bit of the virtual address should not participate in the TLB match. See Table 2-4 "Values for Page Size and PageMask Register" on page 26. | R/W | UNPRED |
| 12:0 | Reserved | Ignored on write, returns zero on read. | R | 0 |

### 2.7.6 Wired Register (CP0 Register 6, Select 0)

The Wired register is required for TLB-based virtual address translation units.

**Wired** **CP0 Register 6, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | Wired | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:5 | Reserved | Ignored on write, returns zero on read. | R | 0 |
| 4:0 | Wired | TLB wired boundary | R/W | 0 |

### 2.7.7 BadVAddr Register (CP0 Register 8, Select 0)

The BadVAddr register is required for TLB-based virtual address translation units.

**BadVAddr** **CP0 Register 8, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BadVAddr | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | BadVAddr | Bad virtual address | R | UNPRED |

### 2.7.8 Count Register (CP0 Register 9, Select 0)

The Count register is a required register for a constant rate timer. This counter increments 1:1 with the core frequency.

**Count** **CP0 Register 9, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Count | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

During IDLE0 or IDLE1 mode, the Count register increments at an unpredictable rate; therefore the Count/Compare registers can not be used as the system timer tick when using the **WAIT** instruction to enter an Idle mode.

During Sleep mode, this register will not increment.

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | Count | Interval counter | R/W | 0 |

### 2.7.9 EntryHi Register (CP0 Register 10, Select 0)

The Index register is required for TLB-based virtual address translation units.

**EntryHi** **CP0 Register 10, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | VPN2 | | | | | | | | | | | | | | | | | | | 0 | | | | | ASID | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | 0 | 0 | 0 | 0 | 0 | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:13 | VPN2 | Virtual address bits 31..13. | R/W | UNPRED |
| 12:8 | Reserved | Ignored on write, returns zero on read. | R | 0 |
| 7:0 | ASID | Address space identifier | R/W | UNPRED |

### 2.7.10 Compare Register (CP0 Register 11, Select 0)

The Compare register is a required register for generating an interrupt from the constant rate timer.

**Compare** **CP0 Register 11, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Compare | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | Compare | Interval counter compare value | R/W | UNPRED |

### 2.7.11 Status Register (CP0 Register 12, Select 0)

The Status register is a required register for general control of the processor.

**Status** **CP0 Register 12, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | CU0 | RP | 0 | RE | 0 | 0 | BEV | 0 | SR | NMI | 0 | 0 | 0 | IM | | | | | | | | 0 | 0 | 0 | UM | 0 | ERL | EXL | IE |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31 | CU3 | This bit is zero. Coprocessor 3 is not implemented. | R | 0 |
| 30 | CU2 | This bit is zero. Coprocessor 2 is not implemented. | R | 0 |
| 29 | CU1 | This bit is zero. Coprocessor 1 is not implemented. | R | 0 |
| 28 | CU0 | Controls access to coprocessor 0. | R/W | 0 |
| 27 | RP | Reduced power. This bit has no effect. | R/W | 0 |
| 25 | RE | Reverse-endian. | R/W | 0 |
| 22 | BEV | Boot exception vectors. | R/W | 1 |
| 20 | SR | Soft reset. | R/W | 0 |
| 19 | NMI | Non-maskable interrupt | R/W | 0 |
| 15:8 | IM | Interrupt mask | R/W | 0 |
| 4 | UM | User-mode. | R/W | 0 |
| 3 | R0 | This bit is zero; Supervisor-mode not implemented | R | 0 |
| 2 | ERL | Error Level | R/W | 1 |
| 1 | EXL | Exception Level | R/W | 0 |
| 0 | IE | Interrupt Enable | R/W | 0 |

### 2.7.12 Cause Register (CP0 Register 13, Select 0)

The Cause register is a required register for general exception processing.

**Cause** **CP0 Register 13, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BD | 0 | CE | | 0 | | | | IV | WP | 0 | | | | | | IP | | | | | | | | 0 | ExcCode | | | | | 0 | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31 | BD | Exception in branch delay slot | R | 0 |
| 29:28 | CE | Coprocessor error | R | 0 |
| 23 | IV | Interrupt vector | R/W | 0 |
| 22 | WP | Watchpoint exception deferred | R/W | 0 |
| 15:10 | IP[7:2] | Hardware interrupts pending | R | 0x20 |
| 9:8 | IP[1:0] | Software interrupts pending | R/W | 0 |
| 6:2 | ExcCode | Exception Code | R | 0 |

## 2.7.13 Exception Program Counter (CP0 Register 14, Select 0)

The Exception Program Counter (EPC) register is a required register for general exception processing.

**EPC** **CP0 Register 14, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | EPC | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | EPC | Exception Program Counter | R/W | UNPRED |

## 2.7.14 Processor Identification (CP0 Register 15, Select 0)

The PRId register is a required register for processor identification.

**PRId** **CP0 Register 15, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Company Options | | | | | | | | Company ID | | | | | | | | Processor ID | | | | | | | | Revision | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:24 | Company Options | System-on-a-chip (SOC) identification:<br>0 Au1000<br>1 Au1500<br>2 Au1100 | R | 1 |
| 23:16 | Company ID | Company ID assigned by MIPS Technologies. AMD's ID = 3. | R | 3 |
| 15:8 | Processor Core ID | Identifies the core revision:<br>0 Reserved<br>1 Au1 revision 1<br>2 Au1 revision 2 | R | 2 |
| 7:0 | Revision | Contains a manufacturing-specific revision level.<br>0 Silicon stepping AB; silicon revision 1.0<br>1 Silicon stepping AC; silicon revision 1.1<br>2 Silicon stepping AD; silicon revision 1.2 | R | SOC specific |

### 2.7.15 Configuration Register 0 (CP0 Register 16, Select 0)

The Config0 register is a required register for various processor configuration and capability.

**Config0** **CP0 Register 16, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M | CT | | | | | DD | CD | UM | WD | NM | SM | OD | 0 | 0 | TM | BE | AT | | AR | | | MT | | | 0 | | | | K0 | | |
| Def. | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31 | M | Denotes Config1 register available at select 1 | R | 1 |
| 30:26 | CT | Reserved, must write 0 | R/W | 0 |
| 25 | DD | Reserved, must write 0 | R/W | 0 |
| 24 | CD | Reserved, must write 0 | R/W | 0 |
| 23 | UM | Reserved, must write 0 | R/W | 0 |
| 22 | WD | Reserved, must write 0 | R/W | 0 |
| 21 | NM | Reserved, must write 0 | R/W | 0 |
| 20 | SM | Reserved, must write 0 | R/W | 0 |
| 19 | OD | Reserved, must write 0 | R/W | 0 |
| 16 | TM | Reserved, must write 0 | R/W | 0 |
| 15 | BE | Indicates the endian mode. | R | 1 |
| 14:13 | AT | Architecture type is MIPS32. | R | 0 |
| 12:10 | AR | Architecture revision is Revision 1. | R | 0 |
| 9:7 | MT | MMU type is standard TLB. | R | 1 |
| 2:0 | K0 | KSEG0 is cacheable, coherent. | R/W | 3 |

### 2.7.16 Configuration Register 1 (CP0 Register 16, Select 1)

The Config1 register is a required register for various processor configuration and capability.

**Config1** **CP0 Register 16, Select 1**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | MMU Size - 1 | | | | | | IS | | | IL | | | IA | | | DS | | | DL | | | DA | | | C2 | MD | PC | WR | CA | EP | FP |
| Def. | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 30:25 | MMU Size - 1 | Number of entries in the TLB minus one. The TLB has 32 entries. | R | 31 |
| 24:22 | IS | Instruction cache sets per way is 128. | R | 1 |
| 21:19 | IL | Instruction cache line size is 32 bytes. | R | 4 |
| 18:16 | IA | Instruction cache associativity is 4-way. | R | 3 |
| 15:13 | DS | Data cache sets per way is 128. | R | 1 |
| 12:10 | DL | Data cache line size is 32 bytes. | R | 4 |
| 9:7 | DA | Data cache associativity is 4-way. | R | 3 |
| 6 | C2 | Coprocessor 2 is not implemented. | R | 0 |
| 5 | MD | Always returns zero on read. | R | 0 |
| 4 | PC | Performance Counter registers are not implemented. | R | 0 |
| 3 | WR | Watchpoint registers are implemented. | R | 1 |
| 2 | CA | Code compression is not implemented. | R | 0 |
| 1 | EP | EJTAG is implemented. | R | 1 |
| 0 | FP | FPU is not implemented. | R | 0 |

### 2.7.17 Load Linked Address Register (CP0 Register 17, Select 0)

The LLAddr register provides the physical address of the most recent Load Linked instruction.

**LLAddr** **CP0 Register 17, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | LLAddr | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | LLAddr | Load Linked Address | R | UNPRED |

### 2.7.18 Data WatchLo Register (CP0 Register 18, Select 0)

The WatchLo and WatchHi registers are the interface to the data watchpoint facility.

**WatchLo** **CP0 Register 18, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | VAddr | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | R | W |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:3 | VAddr | The virtual address to match | R/W | UNPRED |
| 1 | R | If this bit is a one, then watch exceptions are enabled for loads that match the address. | R/W | 0 |
| 0 | W | If this bit is a one, then watch exceptions are enabled for stores that match the address. | R/W | 0 |

### 2.7.19 Instruction WatchLo Register (CP0 Register 18, Select 1)

The IWatchLo and IWatchHi registers are the interface to the instruction watchpoint facility.

**IWatchLo** **CP0 Register 18, Select 1**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | VAddr | | | | | | | | | | | | | | | | | | | | | | | | | | | | | I | 0 | 0 |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:3 | VAddr | The virtual address to match | R/W | UNPRED |
| 2 | I | If this bit is a one, then watch exceptions are enabled for instruction accesses that match the address. | R/W | 0 |

## 2.7.20 Data WatchHi Register (CP0 Register 19, Select 0)

The WatchLo and WatchHi registerS are the interface to the data watchpoint facility.

**WatchHi** **CP0 Register 19, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | M | G | 0 | | | | | | ASID | | | | | | | | 0 | | | | Mask | | | | | | | | | 0 | | |
| Def. | 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | X | X | X | X | X | 0 | 0 | 0 | 0 | X | X | X | X | X | X | X | X | X | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31 | M | Another pair of Watch registers is implemented at the next Select index. | R | 1 |
| 30 | G | If this bit is one, then the ASID field is ignored and any address that matches causes a watch exception. | R | UNPRED |
| 23:16 | ASID | ASID value which is required to match that in the EntryHi register if the G bit is zero in the WatchHi register. | R/W | UNPRED |
| 11:3 | Mask | Any bit in this field that is a one inhibits the corresponding address bit from participating in the address match. | R/W | UNPRED |

## 2.7.21 Instruction WatchHi Register (CP0 Register 19, Select 1)

The IWatchLo and IWatchHi registers are the interface to the instruction watchpoint facility.

**IWatchHi** **CP0 Register 19, Select 1**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | G | 0 | | | | | | ASID | | | | | | | | 0 | | | | Mask | | | | | | | | | 0 | | |
| Def. | 0 | X | 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | X | X | X | X | X | 0 | 0 | 0 | 0 | X | X | X | X | X | X | X | X | X | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 30 | G | If this bit is one, then the ASID field is ignored and any address that matches causes a watch exception. | R | UNPRED |
| 23:16 | ASID | ASID value which is required to match that in the EntryHi register if the G bit is zero in the WatchHi register. | R/W | UNPRED |
| 11:3 | Mask | Any bit in this field that is a one inhibits the corresponding address bit from participating in the address match. | R/W | UNPRED |

## 2.7.22 Scratch Register (CP0 Register 22, Select 0)

The Scratch register exists for the convenience of software. Upon a read, this register returns the value last written into it.

**Scratch** **CP0 Register 22, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Scratch | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | Scratch | This register is present for the convenience of software. | R/W | UNPRED |

### 2.7.23 Debug Register (CP0 Register 23, Select 0)

The Debug register is part of the interface to the EJTAG facility.

**Debug** **CP0 Register 23, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | DBD | DM | 0 | LSNM | 0 |  |  |  |  |  |  |  |  |  | 001 |  |  | DExcCode |  |  |  |  | 0 | SSt | 0 | 0 | DINT | 0 |  |  | DBp | DSS |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | X | X | X | X | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31 | DBD | Debug exception in branch delay slot. | R | UNPRED |
| 30 | DM | If this bit is a one, then in debug mode. | R | 0 |
| 28 | LSNM | Load/stores are performed in the normal fashion when in debug mode. | R/W | 0 |
| 17:15 | 001 | EJTAG version 2.5 | R | 001 |
| 14:10 | DExcCode | Cause[ExcCode] for normal exceptions in debug mode. | R | UNPRED |
| 8 | SSt | Enable single step mode | R/W | 0 |
| 5 | DINT | Last debug exception was asynchronous debug interrupt | R | 0 |
| 1 | DBp | Last debug exception was an SDBPP instruction | R | 0 |
| 0 | DSS | Last debug exception was a single step | R | 0 |

### 2.7.24 DEPC Register (CP0 Register 24, Select 0)

The DEPC register is part of the interface to the EJTAG facility.

**DEPC** **CP0 Register 24, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | DEPC |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | DEPC | Debug exception program counter. | R/W | UNPRED |

### 2.7.25 Data Cache Tag Register (CP0 Register 28, Select 0)

The DTag and DData registers are the interface to the data cache array. This cache interface is unique to the Au1.

**Note:** This register corresponds to the TagLo register in the MIPS32 ISA specification.

**DTag** **CP0 Register 28, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | TAG |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | MRU |  | NMRU |  | LRU |  | 0 | 0 | D | S | L | V |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | 0 | 0 | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:12 | TAG | TAG represents bits [31:12] of a physical memory address. Bits [35:32] of the physical address are always zero. | R/W | UNPRED |
| 11:10 | MRU | Most recently used way. | R/W | UNPRED |
| 9:8 | NMRU | Next most recently used way. | R/W | UNPRED |
| 7:6 | LRU | Least recently used way. | R/W | UNPRED |
| 3 | D | Cache line is dirty (modified). | R/W | UNPRED |
| 2 | S | Cache line is shared (for data cache snoops). | R/W | UNPRED |
| 1 | L | Locked. This bit is set by the user to prevent overwriting of the cache line. | R/W | UNPRED |
| 0 | V | Cache line valid. | R/W | UNPRED |

### 2.7.26 Data Cache Data Register (CP0 Register 28, Select 1)

The DTag and DData registers are the interface to the data cache array.

**Note:** This register corresponds to the DataLo register in the MIPS32 ISA specification.

**DData** **CP0 Register 28, Select 1**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Data | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | Data | Data from the data cache line. | R | UNPRED |

### 2.7.27 Instruction Cache Tag Register (CP0 Register 29, Select 0)

The ITag and IData registers are the interface to the instruction cache array. This cache interface is unique to the Au1.

**Note:** This register corresponds to the TagHi register in the MIPS32 ISA specification.

**ITag** **CP0 Register 29, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TAG | | | | | | | | | | | | | | | | | | | | MRU | | NMRU | | LRU | | 0 | | | | L | V |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | 0 | 0 | 0 | 0 | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:12 | TAG | TAG represents bits [31:12] of a physical memory address. Bits [35:32] of the physical address are always zero. | R/W | UNPRED |
| 11:10 | MRU | Most recently used way. | R/W | UNPRED |
| 9:8 | NMRU | Next most recently used way. | R/W | UNPRED |
| 7:6 | LRU | Least recently used way. | R/W | UNPRED |
| 1 | L | Locked. This bit is set by the user to prevent overwriting of the cache line. | R/W | UNPRED |
| 0 | V | Cache line valid. | R/W | UNPRED |

### 2.7.28 Instruction Cache Data Register (CP0 Register 29, Select 1)

The ITag and IData registers are the interface to the instruction cache array.

**Note:** This register corresponds to the DataHi register in the MIPS32 ISA specification.

**IData** **CP0 Register 29, Select 1**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Data | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | Data | Data from the instruction cache line. | R | UNPRED |

### 2.7.29 ErrorEPC Register (CP0 Register 30, Select 0)

The ErrorEPC register is a required register for exception processing.

**ErrorEPC** **CP0 Register 30, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ErrorEPC | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | ErrorEPC | Error Exception Program Counter | R/W | UNPRED |

### 2.7.30 DESAVE Register (CP0 Register 31, Select 0)

The DESAVE register is part of the interface to the EJTAG facility.

**DESAVE** **CP0 Register 31, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | DESAVE | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | DESAVE | Debug save scratch register, for debug handlers. | R/W | UNPRED |

## 2.8 System Bus (SBUS)

The Au1 core communicates with memories and peripherals via the System Bus (SBUS). The SBUS is a 36-bit physical address and 32-bit data bus which is internal to the Au1500 processor. The SBUS is the coherency point within the Au1500 processor.

### 2.8.1 SBUS Arbitration

The SBUS supports multiple masters—the Au1 core and peripheral DMA engines. The SBUS is granted to the masters in a least-recently-used/fair scheme. This scheme prevents two or more masters from consuming the entire SBUS bandwidth, while permitting low latency access to the SBUS for masters which request the bus infrequently (such as peripherals).

The SBUS requestors in the Au1500 processor are:

- Au1 core
- Ethernet MAC controller (2)
- USB Host controller
- PCI controller
- DMA controller

The Au1 presents a single request to the SBUS arbiter for the three possible requestors: the data cache, the instruction cache and the write buffer. The data cache has the highest priority and the write buffer the lowest priority among the three requests. However, the write buffer priority becomes the highest when the data cache requests a load to an address in the write buffer to allow the write buffer to empty prior to fulfilling the data cache load.

The SBUS arbiter has four bus arbitration slots for handling the SBUS masters:

- Slot 0: Au1 core (data cache, instruction cache, write buffer)
- Slot 1: Ethernet MAC controllers
- Slot 2: DMA controller
- Slot 3: USB host controller and PCI controller

The arbitration scheme for the SBUS is round-robin; that is, each bus master slot has an equal opportunity to obtain access to the SBUS. For a particular SBUS master X, if no other SBUS masters request the bus, then bus master X immediately wins the SBUS. By contrast, if all other SBUS masters request the bus, then bus master X must wait for three other SBUS master slots to transfer before it wins the SBUS, as shown in Figure 2-4.

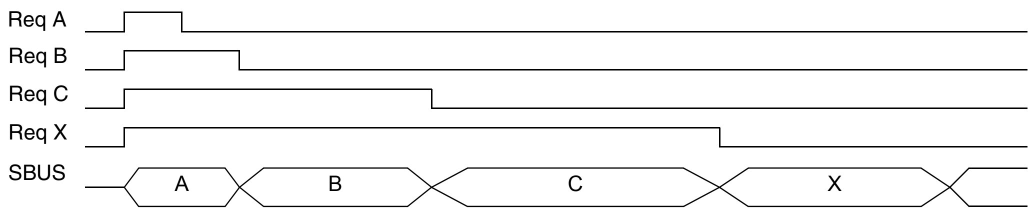


**Figure 2-4. SBUS Arbitration**

When a SBUS master wins arbitration of the SBUS, it performs transfers to/from the integrated peripherals, SDRAM, or the Static bus.

### 2.8.2 SBUS Coherency Model

The SBUS is the coherency point within the Au1500 processor. An SBUS master (i.e. Au1 core or peripheral DMA engine) marks each SBUS transaction as either coherent or non-coherent. SBUS transactions marked as coherent are then snooped by all caching masters (i.e. Au1 data cache). An SBUS transaction that is marked non-coherent is not snooped by caching masters.

The Au1 core is a coherent, caching master. The Au1 data cache snoops SBUS transactions; if a read transaction hits in the data cache then the data cache provides the data, if a write transaction hits in the data cache then the data cache array is updated with the new data.

The integrated peripherals (with DMA engines) can be configured for coherent or non-coherent operation. The ‘C’ bit in the peripheral/module enable register directs whether peripheral SBUS transactions are to be marked coherent or non-coherent. If a peripheral is configured for coherent operation, then it is not necessary to writeback and invalidate Au1 data cache lines which hit in the memory buffers used by DMA engines. If, on the other hand, the peripheral is configured for non-coherent operation, then software must ensure that memory buffers used by the DMA engines are not in the data cache (else the data cache and/or the memory buffer may contain old, stale data).

The decision to use, or not use, coherent SBUS transactions is left to the application. However, peripheral device drivers using coherent SBUS transactions will perform better than drivers not using coherent SBUS transactions since the need to writeback the data cache is eliminated.

## 2.9 EJTAG

EJTAG is supported per the MIPS EJTAG Rev. 2.5 specification. EJTAG provides for CPU and board level bring-up and debug.

# 3 Memory Controllers

The Au1500 processor contains two memory controllers, one for SDRAM and one for static devices.

The SDRAM controller supports SDRAM, SMROM and SyncFlash.

The static device controller supports SRAM, Flash, ROM, page mode ROM, PCMCIA/Compact Flash devices, and an external LCD controller interface.

Both memory controllers support software configurable memory address spaces. This allows designers to keep memory regions contiguous. For example, a system with 4 MB initially installed would locate the memory at physical address 0. Normally, adding 16 MB would create a 12 MB gap in the memory map. With the address configuration options in the Au1500 the 4 MB can be relocated to start at 16 MB, and the new memory can be located at 0 to allow a 20 MB contiguous memory pool.

All registers in the Memory Controller block are located off of the base address shown in Table 3-1.

**Table 3-1. Memory Controller Block Base Address**

| Name | Physical Base Address | KSEG1 Base Address |
|---|---|---|
| mem | 0x0 1400 0000 | 0xB400 0000 |

The system designer has the choice of booting from 32-bit Flash, 16-bit Flash, 32-bit SMROM, and 32-bit SyncFlash. The ROMSEL and ROMSIZE configuration is discussed in more detail in Section 9.3 "Boot" on page 182. Table 9-1 on page 182 shows how the state of ROMSEL and ROMSIZE determines where the processor boots from.

## 3.1 SDRAM Memory Controller

The SDRAM memory controller of the Au1500 processor is designed for glueless interface to one, two, or three ranks of SDRAM or SMROM. SDRAM and SyncFlash are run at 1/2 the internal System Bus (SBUS) speed. The SBUS defaults to 1/2 the processor clock speed so that SDRAM or SyncFlash will run at 99 MHz with a 396 MHz Au1500. SMROM operates at 1/4 the speed of the SBUS. The SBUS divider is programmable, see Section 8.4.4 "Device Power Management - Sleep" on page 174 for more information.

The SDRAM interface supports three chip selects (SDCS[2:0]#), corresponding to three ranks of SDRAM. Each chip select can be configured to support either SDRAM or SMROM. In addition, chip select 0 can be configured for SyncFlash (no other chip selects can be used to support SyncFlash). For chip selects configured as SDRAM or SyncFlash (on chip select 0) the controller keeps one row open for up to four banks per chip select allowing fast accesses and reducing the need to issue precharge cycles.

Note: The SDRAM memory controller supports a maximum of two loads per chip select.

When RESETIN# is negated, code is fetched from SMROM/SyncFlash if SMROM/SyncFlash boot is selected. When using SMROM or SyncFlash for boot, the SMROMCKE signal should be used for the SMROM/SyncFlash CKE. If SMROM or SyncFlash are being used (but not for boot), SDCKE should be used for the clock enable.

After boot internal configuration registers can be written to enable SDRAM chip selects. When a chip select is enabled the SDCKE is driven asserted and clocks are started. Software must wait 10 µs for the SDRAM clock to stabilize before any device specific initialization steps.

All SDRAM/SMROM ranks must be 32 bits wide. Support is included for SDRAM with 2 or 4 banks, 11 to 13 row address bits, and 7 to 11 column address bits. It is also possible to send explicit commands to the SDRAM, under software control, for diagnostic, initialization, or power management purposes.

SDRAM clocks keep running during a runtime reset to allow any transaction in progress to complete. This avoids the possibility of bus contention when the part is brought out of reset.

Note that the SDRAM controller assumes the following external SDRAM configuration:

- Burst Length = 8
- Addressing Mode = Sequential
- Write Mode = Burst Read and Write

### 3.1.1 SDRAM Controller Programming Model

The SDRAM controller contains a number of registers which configure the operation of the interface. All registers in the SDRAM controller block are located off of the base address shown in Table 3-1 "Memory Controller Block Base Address" on page 43. Table 3-2 shows the memory map of the register block.

**Table 3-2. SDRAM Configuration Registers**

| Offset (Note 1) | Register Name | Description |
|---|---|---|
| 0x0000 | mem_sdmode0 | SDRAM chip select n (SDCS*n*#) mode configuration register (timing and functionality) |
| 0x0004 | mem_sdmode1 | |
| 0x0008 | mem_sdmode2 | |
| 0x000C | mem_sdaddr0 | SDCS*n*# address configuration and enable |
| 0x0010 | mem_sdaddr1 | |
| 0x0014 | mem_sdaddr2 | |
| 0x0018 | mem_sdrefcfg | Refresh Configuration and Timing |
| 0x001C | mem_sdprecmd | Issue PRECHARGE to all enabled SDRAM chip selects |
| 0x0020 | mem_sdautoref | Issue AUTO REFRESH to all enabled SDRAM chip selects |
| 0x0024 | mem_sdwrmd0 | Write data to SDCS*n*# SDRAM mode configuration register |
| 0x0028 | mem_sdwrmd1 | |
| 0x002C | mem_sdwrmd2 | |
| 0x0030 | mem_sdsleep | Force SDRAM into self refresh mode |
| 0x0034 | mem_sdsmcke | Toggle SMROMCKE pin |

Note 1. See Table 3-1 on page 43 for base address.

### 3.1.2 SDRAM Registers

Each chip select is configured by two registers, a mode register and an address configuration register.

#### 3.1.2.1 Chip Select Mode Configuration Registers

The format and reset values of the chip select mode configuration registers is shown in the following figure. The timing parameters (Tcl, Tcrd, Trp, Twr, Tmrd, and Tras) correspond directly to times shown in the SDRAM timing diagrams. Times are measured in SDRAM/SMROM clock cycles.

The default values for chip select zero correspond to values for SMROM operation. Chip select 1 and 2 are configured with the slowest timing values at reset.

Reserved fields should be written as zeros and ignored on read to preserve compatibility with future versions of the product.

**mem_sdmode0 - CS0 Mode Configuration** **Offset = 0x0000**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | SF | F | SR | BS | RS | | CS | | | Tras | | | | Tmrd | | Twr | | Trp | | Trcd | | Tcl | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |

**mem_sdmode1 - CS1 Mode Configuration** **Offset = 0x0004**

**mem_sdmode2 - CS2 Mode Configuration** **Offset = 0x0008**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | F | SR | BS | RS | | CS | | | Tras | | | | Tmrd | | Twr | | Trp | | Trcd | | Tcl | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:24 | — | Reserved, should be cleared. | R | 0 |
| 23 | SF | Selects SyncFlash operation. SyncFlash is available only on $\overline{\text{SDCS0\#}}$. For other chip selects, this bit is reserved and should be cleared.<br>0 SyncFlash is not being used.<br>1 SyncFlash is being used. | R/W | 0 |
| 22 | F | Setting the F bit allows the SDRAM controller to assume that no caching master except the core will access this memory space. This allows accesses to begin sooner.<br>Note that the CPU core is the only possible caching master, so it is safe for the system designer to set this bit. | R/W | 0 |
| 21 | SR | Chip select operating mode.<br>0 SDRAM/SyncFlash operation.<br>1 SMROM operation. | R/W | See above |
| 20 | BS | Select Number of Banks.<br>0 Chip select controls 2-bank SDRAM.<br>1 Chip select controls 4-bank SDRAM.<br>Note: This bit must be cleared for SMROM support. | R/W | See above |
| 19:18 | RS | This field sets the number of bits in the row address as shown below:<br>RS Row Address Size<br>00 11<br>01 12<br>10 13<br>11 Reserved | R/W | See above |
| 17:15 | CS | This field sets the number of bits in the column address as shown below:<br>CS Column Size<br>000 7<br>001 8<br>010 9<br>011 10<br>100 11<br>All other values are reserved. | R/W | See above |
| 14:11 | Tras | This field designates the minimum delay from a activate to a precharge command.<br>(Tras + 1) is the actual number of clock cycles. | R/W | 15 |
| 10:9 | Tmrd | This field sets the required delay from an external load of the SDRAM mode register (not the chip select mode register) to an activate command.<br>(Tmrd + 1) is the actual number of clock cycles. | R/W | 3 |
| 8:7 | Twr | The Twr field sets the write recovery time. This is the last data for a write to a precharge. This field is sometimes referred to a Tdpl.<br>(Twr + 1) is the actual number of clock cycles. | R/W | 3 |
| 6:5 | Trp | This field sets the time from precharge to the next activate command.<br>(Trp + 1) is the actual number of clock cycles. | R/W | 3 |
| 4:3 | Trcd | This field sets the RAS to CAS delay.<br>(Trcd + 1) is the actual number of clock cycles. | R/W | See Above |
| 2:0 | Tcl | This field sets the minimum CAS latency timing. This is the time from CAS to DATA on reads.<br>(Tcl + 1) is the actual number of clock cycles. | R/W | See Above |

### 3.1.2.2 SDRAM Chip Select Address Configuration Registers (mem_sdaddr*n*)

The SDRAM chip-select address configuration registers (**mem_sdaddr*n***) assign an address range for each chip select. As shown below, each register contains a base address, an address comparison mask, and an enable bit.

**mem_sdaddr0 - SDCS0# Address Configuration** **Offset = 0x000C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | E | CSBA | | | | | | | | | | CSMASK | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Rs | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**mem_sdaddr1 - SDCS1# Address Configuration** **Offset = 0x0010**

**mem_sdaddr2 - SDCS2# Address Configuration** **Offset = 0x0014**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | E | CSBA | | | | | | | | | | CSMASK | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:21 | — | Reserved, should be cleared. | R | 0 |
| 20 | E | Enable.<br>0 Chip select is disabled.<br>1 Chip select is enabled. | R/W | 0, except for **mem_sdaddr0** (Note 1) |
| 19:10 | CSBA | Chip select base address. Specifies bits 31:22 of the physical base address for this chip select. (The lower bits of the base address are zero.) | R/W | 0x3FF, except for **mem_sdaddr0** where the default value is 0x7F. |
| 9:0 | CSMASK | Chip select address mask. Specifies which bits of CSBA are used to decode this chip select. | R/W | 0x3FF |

Note 1. The E bits for the chip selects SDCS1# and SDCS2# are automatically cleared (disabled) coming out of a runtime or hardware reset. For SDCS0#, however, the reset value of the E bit depends on ROMSEL and ROMSIZE: SDCS0#'s E bit is set when the ROMSEL and ROMSIZE pins indicate that the SMROM/SyncFlash should be used for the boot vector (ROMSEL==1, ROMSIZE==0). See also Section 9.3 "Boot" on page 182.

Once enabled (E bit set), a chip select is asserted when the following condition is met:

```
(phys_addr & addr_mask) == base_addr

where

phys_addr:   32-bit physical address output on the internal SBUS (from the TLB
             for memory-mapped regions) (Bits 35:32 of the physical address are zeros.)

addr_mask:   address comparison mask taken from CSMASK

base_addr:   chip select base address taken from CSBA
```

Note that chip select regions must be programmed so that each chip select occupies a unique area of the physical address space. Programming overlapping chip select regions results in undefined operation.

### 3.1.2.3 Refresh Configuration Register

The refresh configuration register sets the timing of SDRAM refresh for all chip selects. Since the timing for these signals apply to all chip selects, if different types of SDRAM is used the worst case timing must be applied. The format of the refresh configuration register is as follows:

**mem_sdrefcfg - Refresh Configuration** **Offset = 0x0018**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Trc | | | | Trpr | | E | RI | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:28 | Trc | The Trc field specifies the minimum time from the start of an auto refresh cycle to an activate command for all SDRAM chip selects.<br>(Trc + 1) is the actual number of clock cycles. | R/W | 0xf |
| 27:26 | ‘ | This field specifies the minimum time from a precharge to the start of a refresh cycle for all SDRAM chip selects. This is used because a precharge all command is automatically initiated before an auto refresh command. This value should be programmed with the worst case Trp from the **sdr_csmode*n*** registers.<br>(Trpm + 1) is the actual number of clock cycles. | R/W | 3 |
| 25 | E | When this bit is set, refresh is enabled for all chip selects configured as SDRAM. | R/W | 0 |
| 24:0 | RI | Refresh Interval - This field specifies the maximum refresh interval in SBUS clocks for all SDRAM ranks.<br>The refresh interval is for each individual refresh so for a system with a row address size of 12 (4096 rows) and memory with a refresh time of 64 ms (all rows), the individual refresh interval will be 15.7 µs (64 ms/4096). With a SBUS clock of 198 MHz, the RI value should be 0xC24 (15.7 µs / (1/198 MHz). | R/W | 0x1FFFFFF |

### 3.1.2.4 Precharge All Command Register

Writing any value to the **mem_sdprecmd** register issues a precharge all command to all enabled SDRAM chip selects. This can be used for initialization sequences that require certain operations to be performed in a deterministic order.

Reading from the **mem_sdprecmd** register is unpredictable.

**mem_sdprecmd - Precharge All Command Reg** **Offset = 0x001C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | PA | Writing any value to PA will cause a precharge command to be issued to all enabled SDRAM chip selects. | W | UNPRED |

### 3.1.2.5 Auto Refresh Command Register

Writing to the **mem_sdautoref** register performs an auto refresh command on all enabled SDRAM chip selects. This can be used for initialization sequences that require specific operations to be performed in a deterministic order. To insure future compatibility the value written should always be zero.

Reading from the **mem_sdautoref** register will return the current value of the refresh timer.

**mem_sdautoref - Auto Refresh Command** **Offset = 0x0020**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | AR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | AR | Writing a value to AR causes an AUTO REFRESH command to be issued to all enabled SDRAM chip selects. | R/W | UNPRED |

### 3.1.2.6 External SDRAM Mode Register Access

The **mem_sdwrmd0**, **mem_sdwrmd1**, and **mem_sdwrmd2** command registers allow software to directly write to the mode registers in SDRAM connected to each chip select. This can be used in initialization sequences that require certain operations be performed in a deterministic order.

**mem_sdwrmd0 - Write CS0 SDRAM Mode** **Offset = 0x0024**

**mem_sdwrmd1 - Write CS1 SDRAM Mode** **Offset = 0x0028**

**mem_sdwrmd2 - Write CS2 SDRAM Mode** **Offset = 0x002C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BA | | WM | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:30 | BA[1:0] | Bank address. These bits are reflected on the SDBA[1:0] signals. They can be used to write to the *extended* mode register (for synchronous Flash and battery RAM, for example). These bits must be cleared otherwise. | W | UNPRED |
| 29:0 | WM | The value written to this register is written to the external SDRAM mode register for the corresponding chip select. | W | UNPRED |

### 3.1.2.7 SDRAM Sleep/Self Refresh Command Register

Writing any value to this register performs sends a self refresh command on all enabled SDRAM chip selects. This command can be used for the SDRAM power down sequence which requires specific commands to be performed in a deterministic order.

After performing self refresh the SDRAM controller will hold SDCKE low and wait until a Sleep exit sequence or reset is performed. For this reason nothing should access the SDRAM after this command has been issued.

**mem_sdsleep -SDRAM Sleep** **Offset = 0x0030**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SL | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | SL | Writing any value to SL will issue a self refresh command on all enabled chip selects. | W | UNPRED |

#### 3.1.2.8 SMROMCKE Toggle Register

Writing to this register causes the state of the SMROMCKE signal to change. SMROMCKE will default to high when booting from SMROM or Sync Flash. This is used during power-up configuration to change the SMROM burst size from 4 to 8 beats. This command register does not affect the SDRAM SDCKE signal.

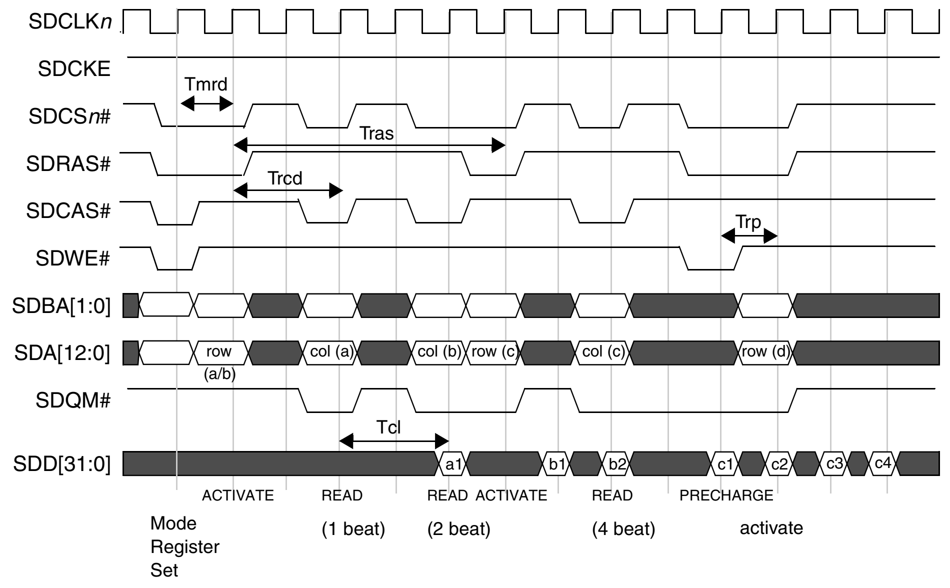
**mem_sdsmcke -SMROMCKE Toggle** **Offset = 0x0034**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | ST |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | ST | Writing to ST (regardless of the value written) inverts the current state of SMROMCKE. | W | UNPRED |

### 3.1.3 SDRAM Timing

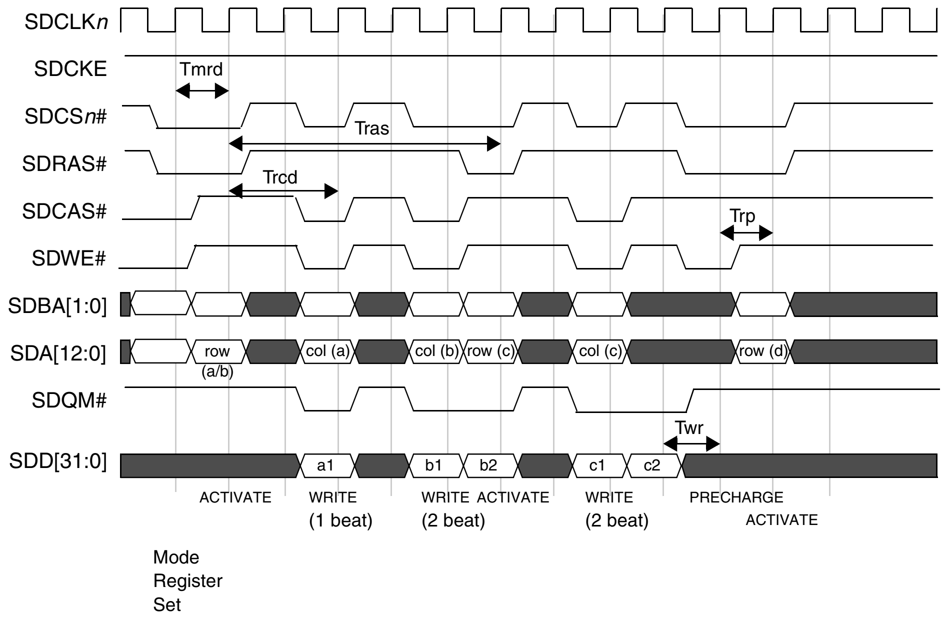
The following figures show examples of typical read, typical write and refresh timing.



The above timing represents the following:
1. Tras = 4 (5 SDRAM clock cycles)
2. Trp = 0 (1 SDRAM clock cycles)
3. Trcd = 1 (2 SDRAM clock cycles)
4. Tcl = 1 (2 SDRAM clock cycles)
5. Tmrd = 0 (2 SDRAM clock cycles)

The above timing is presented to concisely display the different SDRAM timing parameters. The functional bus behavior may differ from that displayed.

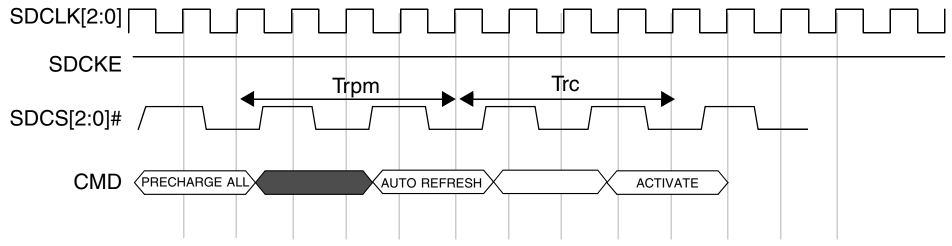
**Figure 3-1. SDRAM Typical Read Timing**



The above timing represents the following:
1. Tras = 4 (5 SDRAM clock cycles)
2. Trp = 0 (1 SDRAM clock cycles)
3. Trcd = 1 (2 SDRAM clock cycles)
4. Tmrd = 0 (1 SDRAM clock cycles)

The above timing is presented to concisely display the different SDRAM timing parameters. The functional bus behavior may differ from that displayed.

**Figure 3-2. SDRAM Typical Write Timing**



This example assumes that all SDCLK ranks ([2:0]) are enabled.

The above timing represents the following:
1. Trpm = 3 (4 SDRAM clock cycles)
2. Trc = 3 (4 SDRAM clock cycles)

**Figure 3-3. SDRAM Refresh Timing**

### 3.1.4 SDRAM Hardware Considerations

Table 3-3 shows the signals associated with the SDRAM interface.

**Table 3-3. SDRAM Signals**

| Signal | Input/ Output | Description |
|---|---|---|
| SDA[12:0] | O | Address Outputs. A0-A12 are driven during the ACTIVE command (row-address A0-A12) and READ/WRITE command to select one location out of the memory array in the respective bank. The address outputs also provide the opcode during a LOAD MODE REGISTER command. |
| SDBA[1:0] | O | Bank Address Outputs. SDBA1 and SDBA0 define to which bank the ACTIVE, READ, WRITE or PRECHARGE command is being applied. The SDBA signal values are programmed in **mem_sdwrmd*n***[BA]. |
| SDD[31:0] | IO | SDRAM data bus. |
| SDQM[3:0]# | O | Input/Output Mask. SDQM# is a mask signal for write accesses and an output enable signal for read accesses. SDQM0# masks SDD[7:0], SDQM1# masks SDD[15:8], SDQM2# masks SDD[23:16], SDQM3# masks SDD[31:24]. |
| SDRAS# | O | Command Outputs. SDRAS#, SDCAS# and SDWE# (along with SDCSn#) define the command being sent to the SDRAM rank. |
| SDCAS# | O | |
| SDWE# | O | |
| SDCLK[2:0] | O | Clock output corresponding to each of the three chip selects. Clock speed is 1/2 SBUS frequency when corresponding SDCSn# is set to SDRAM or SyncFlash, 1/4 SBUS frequency when corresponding SDCSn# is set to SMROM. |
| SDCS[2:0]# | O | Programmable chip selects (3 ranks). |
| SDCKE | O | Clock enable for SDRAM. |
| SMROMCKE | O | Synchronous Mask ROM Clock Enable. This signal must be pulled high if the system is booting from SMROM.<br>Muxed with GPIO[6]. If ROMSEL and ROMSIZE are configured to boot from Synchronous Mask ROM, SMROMCKE will control the pin out of reset, else GPIO[6] will control the pin out of reset. |

## 3.2 Static Bus Controller

The static bus controller provides a general purpose interface to a variety of external peripherals and memory devices. Each of the four static bus chip selects may be programmed to support standard Flash memory, ROM, Page Mode Flash/ ROM, SRAM, I/O peripherals, PCMCIA/Compact Flash devices, or an LCD controller. Because of the similarity of Compact Flash and PCMCIA, references to PCMCIA should be taken as applicable to Compact Flash except where noted.

The Au1500 processor allows control of different device types by reconfiguring what control signals chip select *n* manages based on how the device type field (DTY) is encoded in the **mem_stcfg*n*** register. All device types use the same address and data bus signals, RAD[31:0] and RD[31:0].

Descriptions of all device types are provided in Section 3.2.2 "Static RAM, I/O Device and Flash Device Types" on page 60, Section 3.2.3 "PCMCIA/Compact Flash Device Type" on page 62, and Section 3.2.4 "LCD Controller Device Type" on page 68.

A read to the static bus causes a 32-bit access. This can cause a potential problem with volatile devices because a single 16-bit read results in two 16-bit reads on the external bus.

Chip selects may be programmed for fixed access times or an external wait signal may be used to provide a variable delay per access.

While the static bus controller is a synchronous device internally, no external clock is available to reference the control signals. The internal clock comes from the SBUS clock. Configuring the SBUS clock determines the internal reference for the static controller and associated timings.

### 3.2.1 Static Controller Programming Model

The properties of each static controller chip select are determined by a set of registers. All registers in the Static Controller block are located off of the base address shown in Table 3-1 "Memory Controller Block Base Address" on page 43. Table 3-4 shows the registers and offsets for the static bus controller.

After modifying the configuration of a chip select, software must issue a **SYNC** instruction before write accesses to the chip select are allowed.

**Table 3-4. Static Bus Controller Configuration Registers**

| Offset (Note 1) | Register Name | Description |
|---|---|---|
| 0x1000 | mem_stcfg0 | Configuration for RCS0# |
| 0x1004 | mem_sttime0 | Timing parameters for RCS0# |
| 0x1008 | mem_staddr0 | Address region control for RCS0# |
| 0x1010 | mem_stcfg1 | Configuration for RCS1# |
| 0x1014 | mem_sttime1 | Timing parameters for RCS1# |
| 0x1018 | mem_staddr1 | Address region control for RCS1# |
| 0x1020 | mem_stcfg2 | Configuration for RCS2# |
| 0x1024 | mem_sttime2 | Timing parameters for RCS2# |
| 0x1028 | mem_staddr2 | Address region control for RCS2# |
| 0x1030 | mem_stcfg3 | Configuration for RCS3# |
| 0x1034 | mem_sttime3 | Timing parameters for RCS3# |
| 0x1038 | mem_staddr3 | Address region control for RCS3# |

Note 1. See Table 3-1 on page 43 for base address.

### 3.2.1.1 Static Bus Configuration Registers

The static bus configuration registers (**mem_stcfg*n***) configure the basic properties of each chip select. Support is included for static RAM, Flash, ROM, PCMCIA, LCD, and other types of I/O devices.

When programming a chip select as an I/O, LCD, or PCMCIA device the address comparison mask will expect an address with the upper nibble set as shown in Table 3-5 "Device Type Encoding" on page 55 for the different device types. The TLB must be set up accordingly to map addresses to the memory region captured by the associated chip select.

For example, to configure the TLB for use with an LCD controller, bits 29:26 of CoProcessor register Entry Lo must be 0b1110 (in addition to the other steps necessary to set up the TLB). These bits represent address bits 35:32 of the physical address which must be 0xE in order for the address to match successfully when a chip select is enabled as an LCD device.

Since the RAM and Flash have an upper nibble of zero, it is not necessary to use the TLB to access devices set up with these types.

**mem_stcfg0** **Offset = 0x1000**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | BV | D5 | AV | BE | TS | EW | H | BS | PM | RO | | DTY | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Rs | 0 | 0 | 0 | 0 | 1 | 1 |

**mem_stcfg1** **Offset = 0x1010**
**mem_stcfg2** **Offset = 0x1020**
**mem_stcfg3** **Offset = 0x1030**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | BE | TS | EW | H | BS | PM | RO | | DTY | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:13 | — | Reserved, should be cleared. | R | 0 |
| 12 | BV | Burst size visible. When this bit is set the burst size for static transfers will be output for chip selects not configured as LCD or PCMCIA. The burst size output is one less than the number of 32-bit words to be transferred. For 16-bit chip selects twice as many beats will occur. The mapping of the burst size to pins is shown in Table 3-6 "Burst Size Mapping" on page 55.<br>This bit is a global attribute and is present only in **mem_stcfg0**. | R/W | 0 |
| 11 | D5 | Divide by 5. Setting this bit will divide the SBUS clock by 5 to generate LCLK. When D5 is cleared the SBUS clock is divided by 4 to generate LCLK.<br>This bit is a global attribute and is present only in **mem_stcfg0**. | R/W | 0 |
| 10 | AV | Address visible. Setting this bit will place the address for all internal accesses to the SBUS on the static address bus. This is intended to be used as a debug aid and should not be used during normal operation as it will increase system power usage.<br>This bit is a global attribute and is present only in **mem_stcfg0**. | R/W | 0 |
| 9 | BE | Endianness.<br>0 Little Endian<br>1 Big Endian<br>Program this bit to match the endianness of the processor. This bit should not be set for PCMCIA. | R/W | 0 |
| 8 | TS | Time scale for chip select timing parameters.<br>0 Do not scale the timing parameters.<br>1 Multiply the timing parameters by a factor of four. This option allows for longer access times. | R/W | 0 |
| 7 | EW | When the EW bit is set the EWAIT# input is allowed to stretch the bus access time. The EW bit does not apply to chip selects operating in LCD or PCMCIA mode because they have different wait mechanisms. | R/W | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 6 | H | Half Bus. Selects the data bus width for the chip select.<br>0 32-bit bus.<br>1 16-bit bus using bits 15:0 of the data bus.<br>For PCMCIA device type, clear this bit.<br>For LCD device type, set this bit. | R/W | 0, except for **mem_stcfg0** where the default value is determined by ROMSEL and ROMSIZE out of reset. See Table 9-1 on page 182. |
| 5 | BS | Burst Size for Page Mode Accesses. Selects the burst size for page mode accesses. Valid only in page mode (PM=1).<br>0 4 beats<br>1 8 beats | R/W | 0 |
| 4 | PM | If the PM bit is set the chip select will operate in page mode. This allows quick access to sequential locations in memory. Page mode applies only to reads.<br>See Section 3.2.6.1 "Page Mode Transfers" on page 70. | R/W | 0 |
| 3 | RO | If the RO bit is set the chip select will operate in read only mode. This will inhibit the generation of write cycles to the chip select. Any attempt to write to the address region controlled by a read only chip select will be ignored. | R/W | 0 |
| 2:0 | DTY | Device type. Selects the type of device controlled by the static controller chip select. A list of device types and encodings is shown in Table 3-5 "Device Type Encoding" on page 55.<br>Programming multiple chip selects as LCD or PCMCIA is illegal. Only one of each is supported. | R/W | 0 (SRAM), except for **mem_stcfg0** where the default value is 3 (Flash). |

**Table 3-5. Device Type Encoding**

| DTY | Chip Select Function | PFN[35:32]<br>(upper nibble of physical address) | Reference |
|---|---|---|---|
| 0 | Static RAM | 0x0 | Section 3.2.2 on page 60 |
| 1 | I/O Device | 0xD | Section 3.2.2 on page 60 |
| 2 | PCMCIA Device/Compact Flash | 0xF | Section 3.2.3 on page 62 |
| 3 | Flash Memory | 0x0 | Section 3.2.2 on page 60 |
| 4 | LCD Device (RCS2# only) | 0xE | Section 3.2.4 on page 68 |
| 5–7 | Reserved | | |

**Table 3-6. Burst Size Mapping**

| Signal | Pin |
|---|---|
| burst_size[2] | LWR0# |
| burst_size[1] | LRD1# |
| burst_size[0] | LRD0# |

#### 3.2.1.2 Static Timing Registers

The static timing registers allow software to control the timing of each phase of a static bus access. The names of the timing parameters correspond directly to timing parameters shown on the timing diagrams.

All timing parameters are expressed as a number of clock cycles. The base clock frequency is the SBUS clock.

The actual number of clocks for each timing parameter ($T_{parameter}$) is shown in Table 3-7. Note that the timing behavior for Tcsh is different and is shown in Table 3-8.

**Table 3-7. Actual Number of Clocks for Timing Parameters (Except Tcsh)**

| Device Type | TS = 0 | TS = 1 |
|---|---|---|
| Static RAM, I/O, Flash | $T_{parameter} + 1$ | $(4 * T_{parameter}) + 1$ |
| PCMCIA Device/Compact Flash | $T_{parameter} + 2$ | $(4 * T_{parameter}) + 2$ |

**Table 3-8. Actual Number of Clocks for Tcsh**

| Tcsh Value | TS = 0 | TS = 1 |
|---|---|---|
| 0000 | 3 | 3 |
| 0001 | 3 | 6 |
| 0010 | 6 | 12 |
| 0011 | 6 | 15 |
| 0100 | 6 | 18 |
| 0101 | 9 | 24 |
| 0110 | 9 | 27 |
| 0111 | 9 | 30 |
| 1000 | 12 | 36 |
| 1001 | 12 | 39 |
| 1010 | 12 | 42 |
| 1011 | 15 | 48 |
| 1100 | 15 | 51 |
| 1101 | 15 | 54 |
| 1110 | 18 | 60 |
| 1111 | 18 | 63 |

**mem_sttime0 (I/O, Flash, SRAM config)** **Offset = 0x1004**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | ▨ | Twcs |  |  | Tcsh |  |  |  | ▨ | ▨ | ▨ | ▨ | Twp |  |  |  |  |  | Tcsw |  |  |  | Tpm |  |  |  | Ta |  |  |  |  |  |
| Def. | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |

**mem_sttime1 (I/O, Flash, SRAM config)** **Offset = 0x1014**
**mem_sttime2 (I/O, Flash, SRAM and LCD config)** **Offset = 0x1024**
**mem_sttime3 (I/O, Flash, SRAM config)** **Offset = 0x1034**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | ▨ | Twcs |  |  | Tcsh |  |  |  | ▨ | ▨ | ▨ | ▨ | Twp |  |  |  |  |  | Tcsw |  |  |  | Tpm |  |  |  | Ta |  |  |  |  |  |
| Def. | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31 | — | Reserved, should be cleared. | R | 0 |
| 30:28 | Twcs | This field specifies the required chip select hold time after a write pulse.<br>See Table 3-7 "Actual Number of Clocks for Timing Parameters (Except Tcsh)" on page 56 for the actual number of clock cycles. | R/W | 0x3 |
| 27:24 | Tcsh | Chip select hold-off. Specifies the minimum number of cycles that the chip select must remain inactive between accesses. The next transaction through the static bus controller is held off until the Tcsh parameter is satisfied. If this next access falls within another chip select's memory region, the new set of timing parameters associated with the controlling chip select take effect once the new transaction begins.<br>Note that the SBUS can arbitrarily extend the time between accesses for internal operations. This can add up to about five additional clocks to the programmed time.<br>See Table 3-8 "Actual Number of Clocks for Tcsh" on page 56 for the actual number of clock cycles. | R/W | 0xF |
| 23:20 | — | Reserved, should be cleared. | R | 0 |
| 19:14 | Twp | This field specifies the duration of the write enable.<br>See Table 3-7 "Actual Number of Clocks for Timing Parameters (Except Tcsh)" on page 56 for the actual number of clock cycles. | R/W | 0x3F |
| 13:10 | Tcsw | Chip select to write. Defines the delay from the assertion of chip select until the write strobe and byte enables are asserted.<br>See Table 3-7 "Actual Number of Clocks for Timing Parameters (Except Tcsh)" on page 56 for the actual number of clock cycles. | R/W | 0xF |
| 9:6 | Tpm | This field determines the number of cycles required from a burst address change until read data is valid if the PM bit is set in the **mem_stcfg*n*** register.<br>See Table 3-7 "Actual Number of Clocks for Timing Parameters (Except Tcsh)" on page 56 for the actual number of clock cycles.<br>Ta determines the access time for the *first* beat of each burst. | R/W | 0xF |
| 5:0 | Ta | The Ta parameter determines the number of cycles required for the assertion of the chip select.<br>For page mode accesses Ta determines the access time up to the first beat of each burst, or the first beat after a page mode wrap.<br>See Table 3-7 "Actual Number of Clocks for Timing Parameters (Except Tcsh)" on page 56 for the actual number of clock cycles. | R/W | 0x3F, except for **mem_sttime0** where the default value is 0x1D. |

**mem_sttime0 (PCMCIA config)** **Offset = 0x1004**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Tmst | | | | | | | | Tmsu | | | | | | | Tmih | | | | | | Tist | | | | | | Tisu | | | | |
| Def. | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |

**mem_sttime1 (PCMCIA config)** **Offset = 0x1014**
**mem_sttime2 (PCMCIA config)** **Offset = 0x1024**
**mem_sttime3 (PCMCIA config)** **Offset = 0x1034**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Tmst | | | | | | | | Tmsu | | | | | | | Tmih | | | | | | Tist | | | | | | Tisu | | | | |
| Def. | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:24 | Tmst | This field specifies the strobe width during memory accesses to PCMCIA chip selects.<br>The timing duration depends on the time-scale option **mem_stcfg*n***[TS]:<br>When TS=0, (Tmst + 2) is the number of cycles to the end of the strobe; however, the read occurs at (Tmst + 1).<br>When TS=1, [(4 * Tmst) + 2] is the number of cycles to the end of the strobe; however, the read occurs at [(4 * Tmst) + 1]. | R/W | 0xFF |
| 23:17 | Tmsu | This field specifies the setup time from chip select to strobe during memory accesses to PCMCIA chip selects.<br>See Table 3-7 "Actual Number of Clocks for Timing Parameters (Except Tcsh)" on page 56 for the actual number of clock cycles. | R/W | 0x7F |
| 16:11 | Tmih | This field specifies the hold time for address, data, and chip selects from the end of the strobe for both memory and I/O cycles to PCM-CIA chip selects.<br>See Table 3-7 "Actual Number of Clocks for Timing Parameters (Except Tcsh)" on page 56 for the actual number of clock cycles. | R/W | 0x3F |
| 10:5 | Tist | This field specifies the strobe width for I/O accesses for a chip select configured for PCMCIA.<br>The timing duration depends on the time-scale option **mem_stcfg*n***[TS]:<br>When TS=0, (Tmst + 2) is the number of cycles to the end of the strobe; however, the read occurs at (Tmst + 1).<br>When TS=1, [(4 * Tmst) + 2] is the number of cycles to the end of the strobe; however, the read occurs at [(4 * Tmst) + 1]. | R/W | 0x3F, except for **mem_sttime0** where the default value is 0x3E. |
| 4:0 | Tisu | This field specifies the setup time from chip select to strobe during I/O accesses for PCMCIA.<br>See Table 3-7 "Actual Number of Clocks for Timing Parameters (Except Tcsh)" on page 56 for the actual number of clock cycles. | R/W | 0x1F, except for **mem_sttime0** where the default value is 0x1D. |

### 3.2.1.3 Static Chip Select Address Configuration Registers (mem_staddr*n*)

The static memory chip-select address configuration registers (**mem_staddr*n***) assign an address range for each chip select. As shown below, each register contains a base address, an address comparison mask, and an enable bit.

**mem_staddr0 - RCS0# Address Configuration** **Offset = 0x1008**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | E | CSBA | | | | | | | | | | | | | | CSMASK | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | Rs | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**mem_staddr1 - RCS1# Address Configuration** **Offset = 0x1018**
**mem_staddr2 - RCS2# Address Configuration** **Offset = 0x1028**
**mem_staddr3 - RCS3# Address Configuration** **Offset = 0x1038**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | E | CSBA | | | | | | | | | | | | | | CSMASK | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:29 | — | Reserved, should be cleared. | R | 0 |
| 28 | E | Enable<br>0 Chip select is disabled.<br>1 Chip select is enabled. | R/W | 0, except for **mem_staddr0** (Note 1) |
| 27:14 | CSBA | Chip select base address.<br>Specifies bits 31:18 of the physical base address for this chip select. The upper nibble of the chip select address is determined by the device type selected in **mem_stcfg*n***[DTY]. The lower bits of the base address are zeros. | R/W | 0x3FFF, except for **mem_staddr0** where the default value is 0x7F0. |
| 13:0 | CSMASK | Chip select address mask.<br>Specifies bits 31:18 of the address comparison mask used to decode this chip select. (The upper nibble of the address comparison mask is determined by **mem_stcfg*n***[DTY]. The lower bits of the mask are zeros.) | R/W | 0x3FFF |

Note 1. The enable (E) bits for chip selects RCS1#, RCS2#, and RCS3# are automatically cleared (disabled) coming out of a runtime or hardware reset. For RCS0#, however, the reset value of the E bit depends on ROMSEL: Holding ROMSEL low indicates that ROM should be used for the boot vector (and RCS0#'s E bit is set); otherwise, RCS0# is disabled. See also Section 9.3 "Boot" on page 182.

Once enabled, a chip select is asserted when the following condition is met:

```
(phys_addr & addr_mask) == base_addr

where:

phys_addr:   36-bit physical address output on the internal SBUS (from the TLB
             for memory-mapped regions)

addr_mask:   address comparison mask taken from CSMASK

base_addr:   chip select base address taken from CSBA
```

Note that chip select regions must be programmed so that each chip select occupies a unique area of the physical address space. Programming overlapping chip select regions results in undefined operation.

### 3.2.2 Static RAM, I/O Device and Flash Device Types

This section describes the static RAM interface which is implemented when the device type (**mem_stcfg*n***[DTY]) is programmed to 0, 1 or 3. (See Section 3.2.1.1 "Static Bus Configuration Registers" on page 54.)

The static RAM, I/O device and Flash device types are all similar. The I/O device type is identical to the static RAM type except that it expects the upper nibble of the system address (bits [35:32]) to be 0xD. The only difference between the Flash device type and the static RAM device type is that the Flash timing allows for a chip select hold time after a write pulse using **mem_sttime*n***[Twcs].

Other than these differences, the static RAM, I/O device and Flash device types share the same timing and control signals. The control signals are shown in Table 3-9.

**Table 3-9. Static RAM, I/O Device and Flash Control Signals**

| Pin Name | Input/Output | Description |
|---|---|---|
| RAD[31:0] | O | Address bus |
| RD[31:0] | I/O | Data bus |
| RBE[3:0]# | O | Byte enables:<br>RBE0# corresponds to RD[7:0].<br>RBE1# is for RD[15:8].<br>RBE2# is for RD[23:16].<br>RBE3# is for RD[31:24]. |
| RWE# | O | Write enable. |
| ROE# | O | Output enable. |
| RCS[3:0]# | O | Programmable Chip Selects (4 banks). RCS*n*# is not used when configured as a PCMCIA device. |
| EWAIT# | I | Can be used to stretch the bus access time when enabled through **mem_stcfg*n***[EW]. |

#### 3.2.2.1 Static Memory Timing

The following figures show static memory timing. Figure 3-4 on page 61 illustrates static memory read timing, and Figure 3-6 on page 61 illustrates static memory write timing. The EWAIT# timing diagrams are presented to show how EWAIT# will hold the cycle past Ta for reads and Twp for writes.

Setup, hold, and delay timing specifications (electrical switching characteristics) are presented in Section 12.0 "Electrical and Thermal Specifications". (See Section 12.6.2 "Static Bus Controller Timing" on page 222.)

Timing parameters do not take into account SBUS overhead which may add inter-access delays. These delays are dependent on system design and are affected by the number of bus masters and the ability of other devices to hold the bus.

## Read Timing

Read accesses to the static bus always retrieve 32-bits of data. As such, all four byte enables are asserted during the 32-bit access or the two 16-bit beats. The control signals (RCS*n*#, ROE#, and RBE[1:0]#) span both accesses. The only signal that changes state to indicate the start of the second beat is RADDR[1].

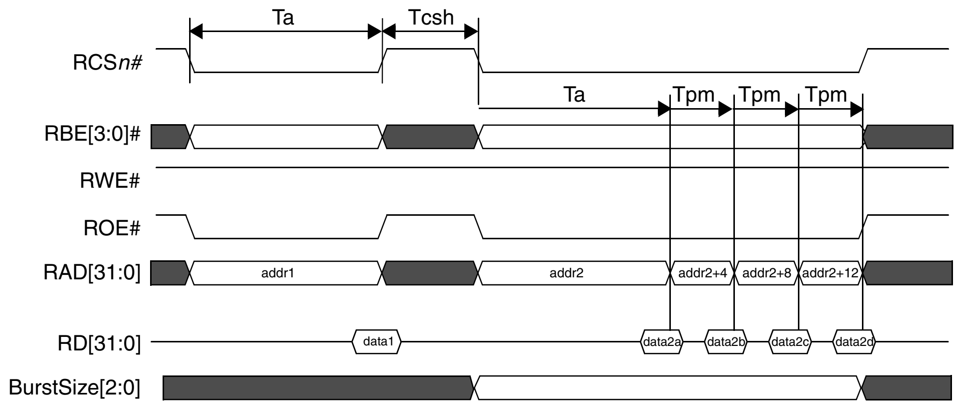


**Figure 3-4. Static Memory Read Timing (Single Read Followed by Burst)**

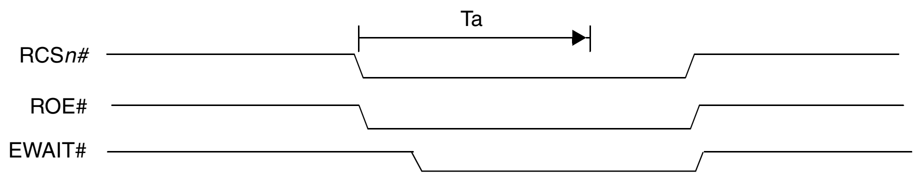


**Figure 3-5. Static Memory Read EWAIT# Timing**

## Write Timing

The timing diagrams below show the static bus write timing for I/O and SRAM device types. Figure 3-6 shows a single 32-bit write on a 32-bit chip select.

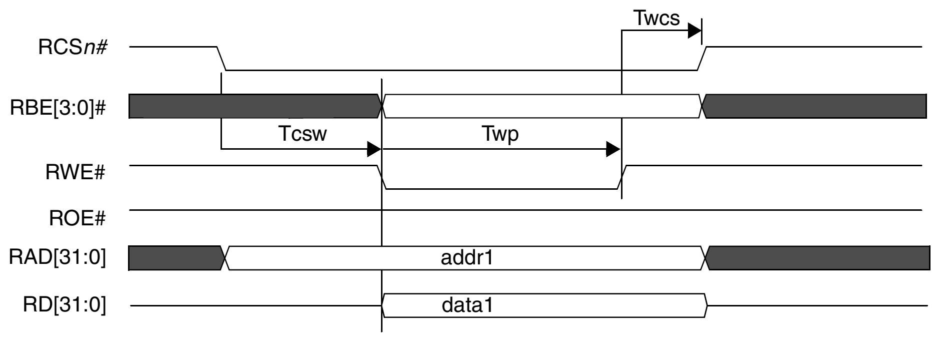


**Figure 3-6. Static Memory Write Timing**

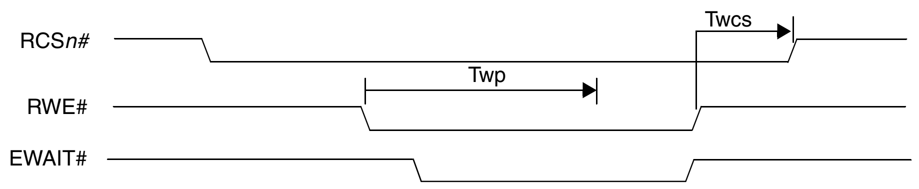


**Figure 3-7. Static Memory Write EWAIT# Timing**

### 3.2.3 PCMCIA/Compact Flash Device Type

Because of the similarity of Compact Flash and PCMCIA, references to PCMCIA should be taken as applicable to Compact Flash except where noted. The PCMCIA peripheral is designed to the PCMCIA2.1 specification—but only for the bus transactions as described in this section.

The Au1500 processor provides a PCMCIA host adapter when the device type is programmed for PCMCIA. The static controller interface provides the required bus signals necessary to control a PCMCIA interface. Auxiliary signals, such as card detect and voltage sense, can be implemented with GPIOs if desired.

The PCMCIA host interface adapter will support memory, attribute and I/O transactions. External logic can be added to support DMA transfers. The Au1500 processor supports only 8- and 16-bit load and store instructions (byte and halfword instructions) to PCMCIA devices. 32-bit accesses are not supported.

The PCMCIA interface provides control signals defined for PCMCIA devices. If two devices are required then external logic must be added to allow for both cards to share the bus. Note that when a chip select is programmed as a PCMCIA device that the associated RCS*n*# is not used.

The PCMCIA interface occupies a 36-bit address space with the upper 4 bits equal to 0xF. The TLB is required to generate addresses that will activate a chip select with a device type of “PCMCIA”.

I/O, Memory and Attribute spaces are differentiated by addr[31:30]. Table 3-10 shows the mapping.

**Table 3-10. PCMCIA Memory Mapping**

| Physical Address | PCMCIA Mapping |
|---|---|
| 0xF 0xxx xxxx | I/O |
| 0xF 4xxx xxxx | Attribute Memory |
| 0xF 8xxx xxxx | Memory |

**Note:** Each of the PCMCIA physical address spaces have a maximum size of 64 MBytes. Any access beyond the 64-MByte space will alias back into the defined region.

Table 3-11 enumerates the signals to support the PCMCIA interface.

**Table 3-11. PCMCIA Interface Signals**

| Pin Name | Input/Output | Description |
|---|---|---|
| RAD[31:0] | O | Address Bus. |
| RD[15:0] | I/O | Data Bus. |
| PREG# | O | When this signal is asserted card access is limited to attribute memory when a memory access occurs and to I/O ports when an I/O access occurs. |
| PCE[2:1]# | O | Card Enables. |
| POE# | O | Memory Output Enable. |
| PWE# | O | Memory Write Enable. |
| PIOR# | O | I/O Read Cycle Indication. |
| PIOW# | O | I/O Write Cycle Indication. |
| PWAIT# | I | This signal is asserted by the card to delay completion of a pending cycle.<br>Note that this signal should be tied high through a resistor when the PCMCIA interface is not used. |
| PIOS16# | I | 16-bit port select. Note that this signal should be tied high through a resistor when the PCMCIA interface is not used. |
| ROE# | O | Output Enable - This output enable is intended to be used as a data transceiver control. During a PCMCIA transaction, ROE# remains asserted (low) as configured in the timing registers (mem_sttime*n*) for reads and negated (high) for writes. |

Figure 3-8 and Figure 3-9 on page 64 show a one and two card PCMCIA implementation. For the two card implementation RAD26 is used as a card select signal. Both figures assume that the PCMCIA card can be hot swapped at any time—note the use of isolation buffers on the shared bus. If the card is fixed in the system much of the interface logic can be removed. A Compact Flash implementation is very similar to the PCMCIA implementation except that the number of address lines used is fewer.

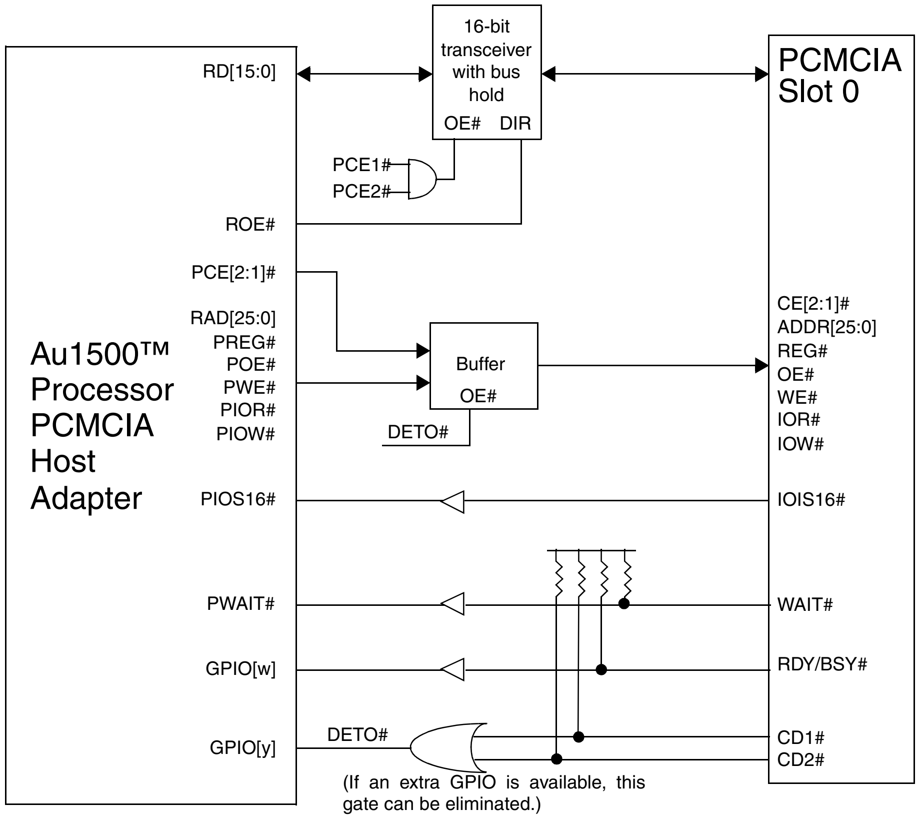


**Figure 3-8. One Card PCMCIA Interface**

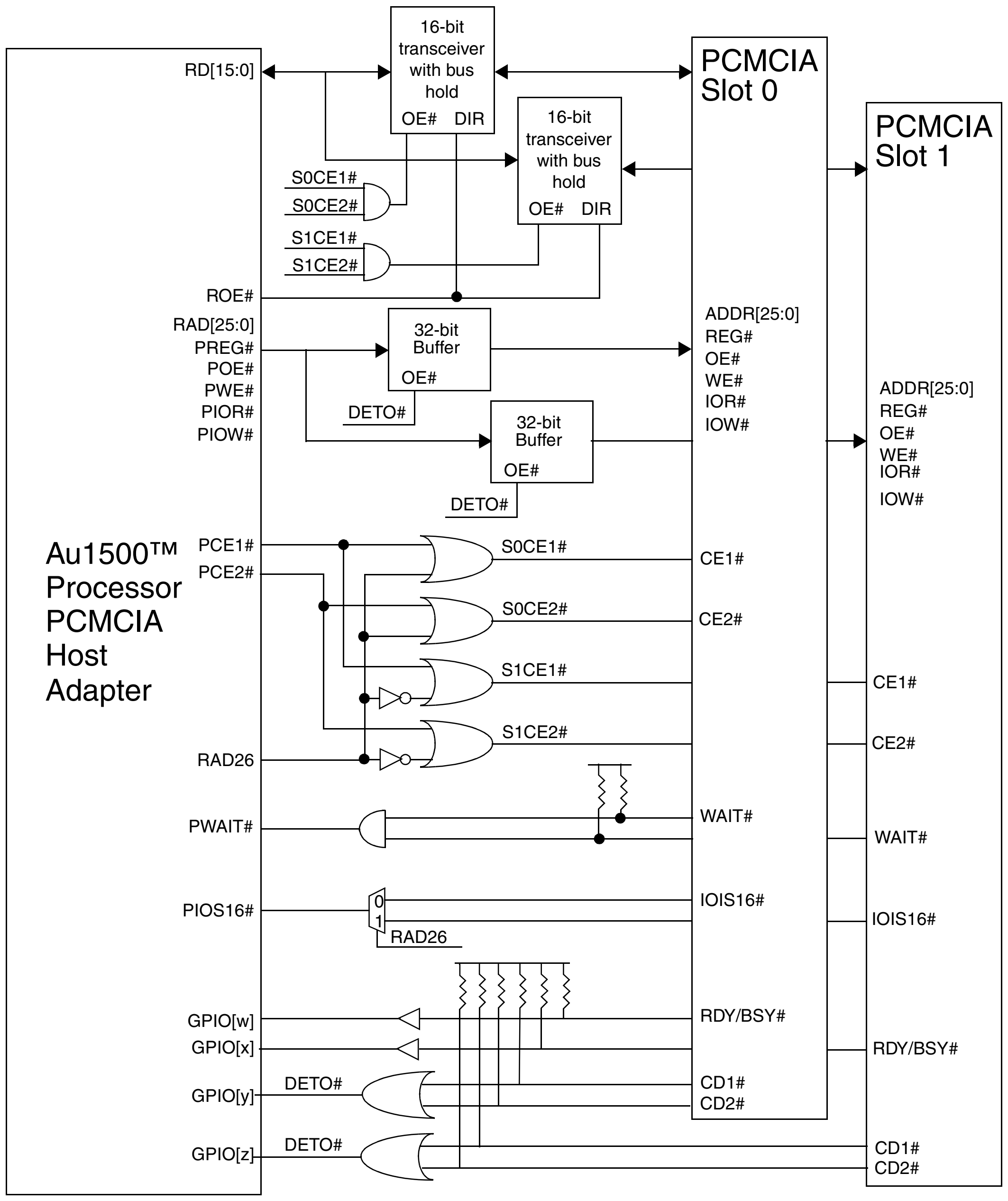


**Figure 3-9. Two Card PCMCIA Interface**

### 3.2.3.1 PCMCIA/CompactFlash Interface

The figures on the following pages illustrate the functional timing of the PCMCIA interface, including memory read timing, memory write timing, I/O read timing, and I/O write timing. The PWAIT# timing diagrams are presented to show how PWAIT# will hold the cycle past Tmst for memory reads and writes and Tist for I/O reads and writes.

Setup and hold time requirements are presented in Section 12.6.2 "Static Bus Controller Timing" on page 222.

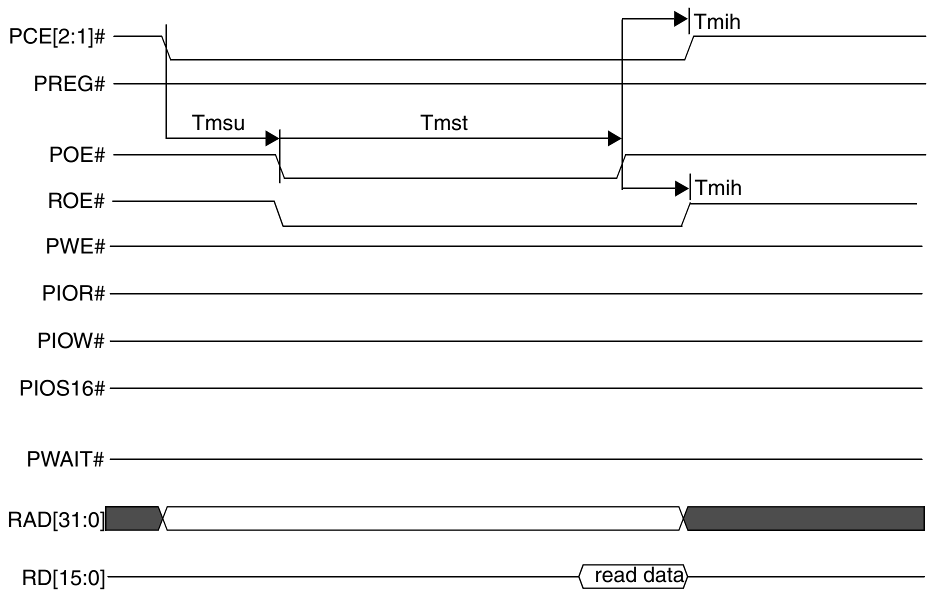


**Figure 3-10. PCMCIA Memory Read Timing**

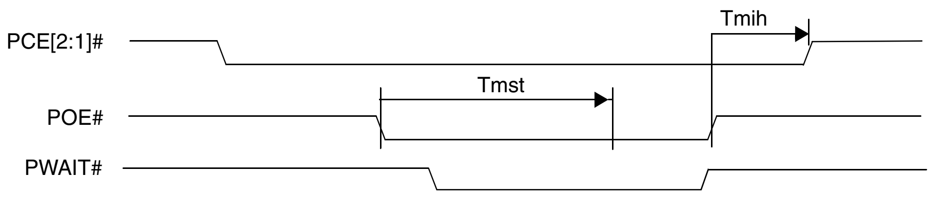


**Figure 3-11. PCMCIA Memory Read PWAIT# Timing**

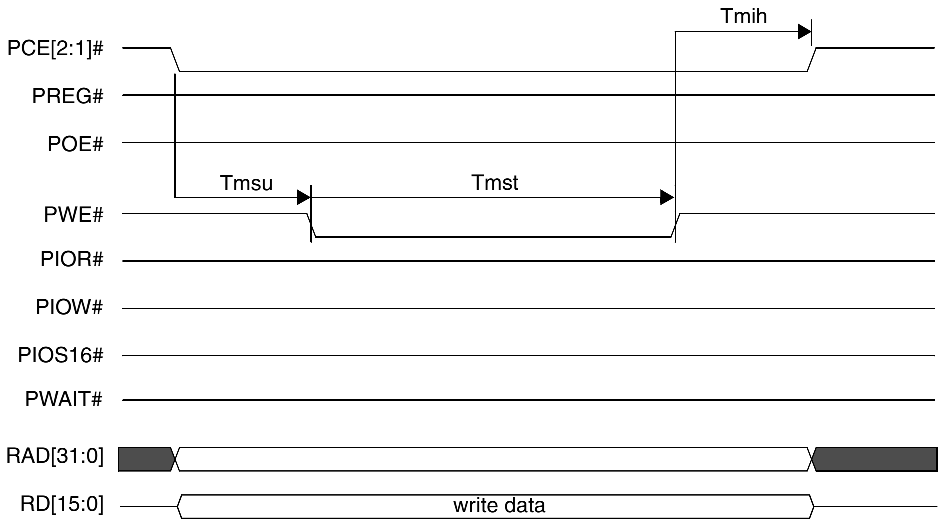


**Figure 3-12. PCMCIA Memory Write Timing**

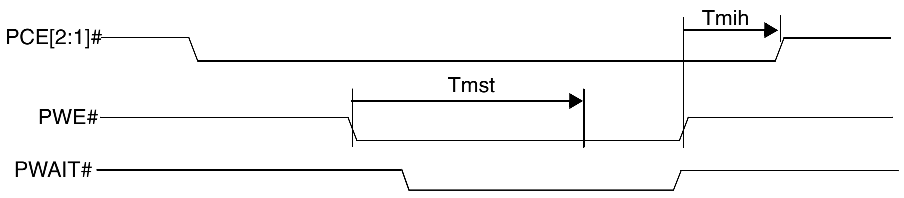


**Figure 3-13. PCMCIA Memory Write PWAIT# Timing**

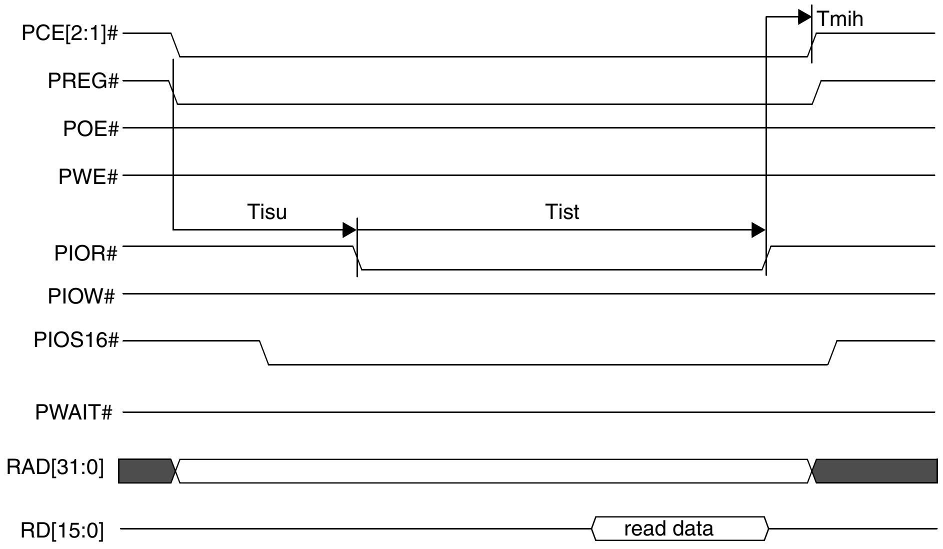


**Figure 3-14. PCMCIA I/O Read Timing**

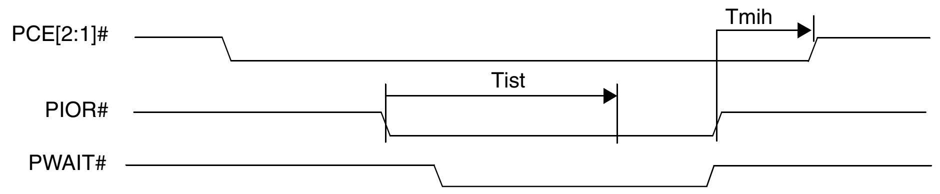


**Figure 3-15. PCMCIA I/O Read PWAIT# Timing**

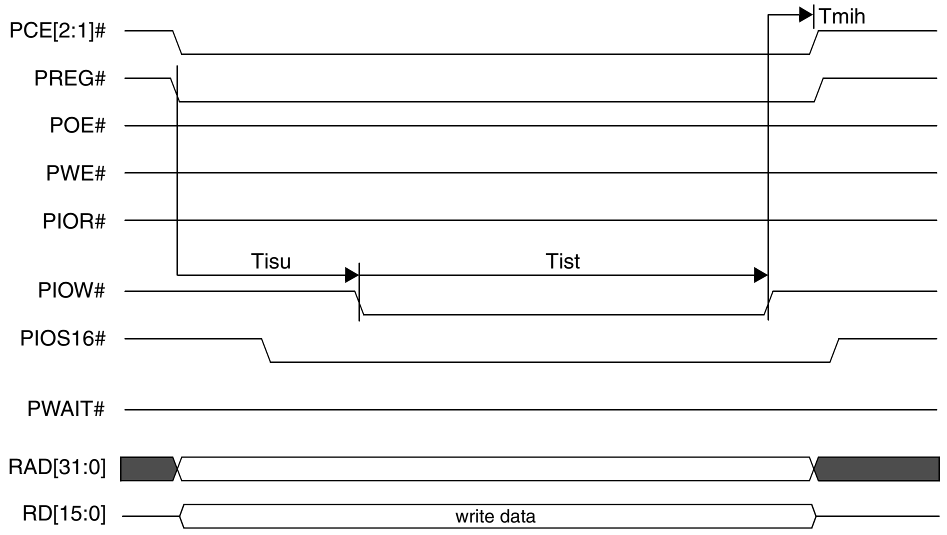


**Figure 3-16. PCMCIA I/O Write Timing**

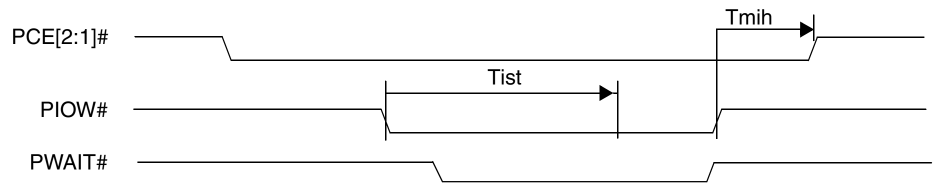


**Figure 3-17. PCMCIA I/O Write PWAIT# Timing**

### 3.2.4 LCD Controller Device Type

The Au1500 processor provides a LCD controller host adapter when the device type is programmed for an LCD. The static controller interface provides the bus signals necessary to interface to most LCD controllers.

A dedicated clock LCLK is provided for the LCD interface. The LCLK rate is the SBUS rate divided by a factor programmed in **mem_stcfg0**[D5]; see Section 3.2 "Static Bus Controller" on page 53.

The Au1500 supports 8-, 16-, and 32-bit load and store instructions (byte, halfword, and word instructions) to the LCD controller interface.

The LCD controller occupies 36-bit address space with the upper 4 bits equal to 0xE. The MMU is required to generate addresses that will generate a chip select with a device type of “LCD”.

Table 3-12 lists the control signals to support the LCD controller.

**Table 3-12. LCD Controller Interface Signals**

| Signal | Input/Output | Function |
|---|---|---|
| RAD[31:0] | O | Address Bus. |
| RD[15:0] | I/O | Data Bus. |
| RCS[3:0]# | O | Chip Selects. |
| LCLK | O | Interface Clock. |
| LWAIT# | I | Extend Cycle. |
| LRD[1:0]# | O | Read Indicators. |
| LWR[1:0]# | O | Write Indicators. |

### 3.2.5 LCD Controller Interface Timing

The following figures shows the LCD timing. The LWAIT# timing diagrams are presented to show how LWAIT# will hold the cycle past Ta for memory reads and Twp for memory writes.

LWAIT# timing requirements as well as setup and hold times are presented in Section 12.6.2 "Static Bus Controller Timing" on page 222.

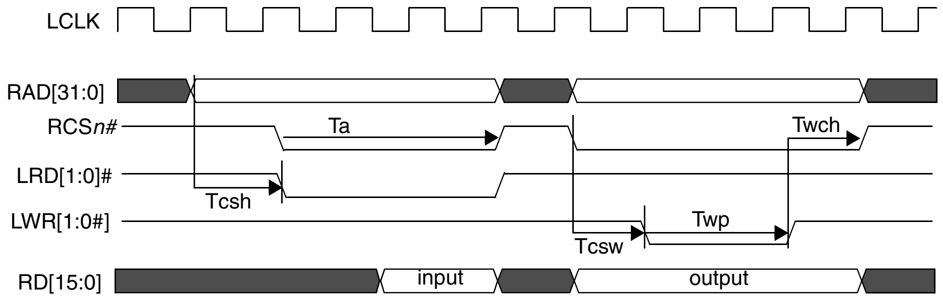


**Figure 3-18. LCD Controller Timing**

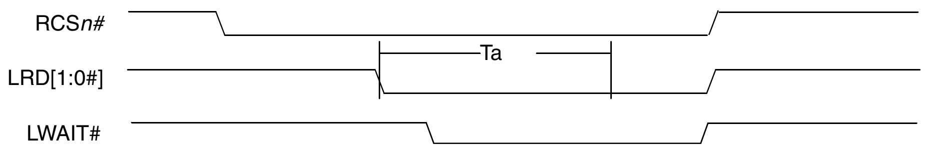


**Figure 3-19. LCD Read LWAIT# Timing**

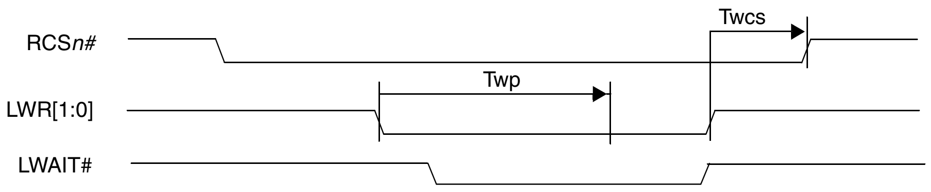


**Figure 3-20. LCD Write LWAIT# Timing**

## 3.2.6 Static Bus Controller Programming Considerations

### 3.2.6.1 Page Mode Transfers

The static bus controller provides a page mode for quick read access to sequential locations in memory. Setting **mem_stcfg*n***[PM] selects page mode operation for the chip select. The burst size (4 or 8 beats) for page mode transfers is programmed in **mem_stcfg*n***[BS].

Depending on the speed of the external memory device, the system designer can adjust two timing parameters in **mem_sttime*n*** for page mode transfers:

- *Ta* is the time from chip select assertion to the first beat of valid data. Ta is the time required for the initial access to a peripheral device. Ta must allow time for the peripheral device to load its read buffer or activate the next page. Note that the page size depends on the peripheral device.
- *Tpm* is the time between beats.

Figure 3-4 "Static Memory Read Timing (Single Read Followed by Burst)" on page 61 shows an example page mode read with the timing parameters Ta and Tpm.

The static bus controller does not check for page boundaries during page mode reads. The addressing is sequential regardless of alignment. An access which crosses a page boundary may return invalid data if Tpm does not allow enough time for the external memory device to update its read buffer or activate the next page. If the system designer cannot ensure adequate address alignment to avoid crossing page boundaries, Tpm must be long enough to accommodate potential page updates.

In general, page-boundary timing issues do not arise for instruction fetches because they are always accessed first-word-first and therefore are properly aligned. Data fetches, however, may have page-boundary timing issues because they are accessed critical-word-first unless CCA=4 (fetch the first-word first).

Note that EWAIT# can delay only the start of the burst (extend the Ta timing). That is, EWAIT# cannot be used to account for varying timing between beats (extend the Tpm timing) that may occur even for transfers within a page.

#### Halfword Ordering and 16-bit Chip Selects

Because the static bus controller is not aware of the endian mode of the Au1 core, potential halfword swapping conflicts can arise. Upon reset, chip selects default to little-endian byte ordering (**mem_stcfg**[BE] = 0). Figure 3-21 on page 70 shows the data formats for the 32-bit SBUS and for a little-endian 16-bit chip select.

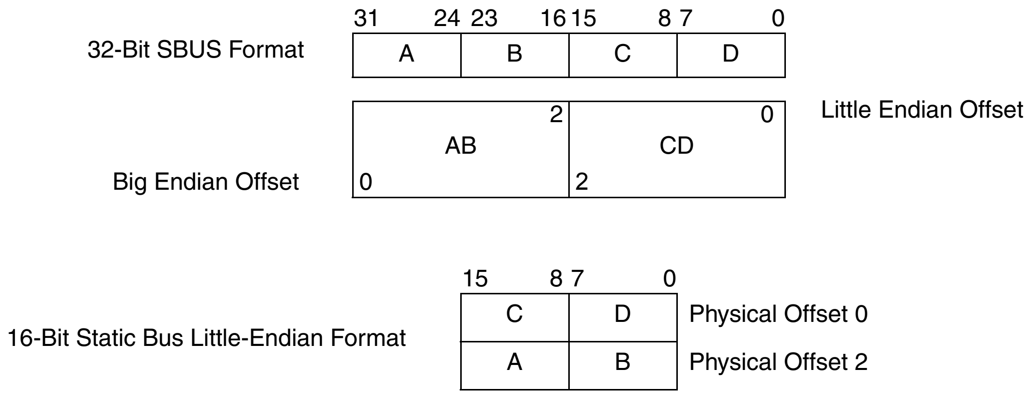


**Figure 3-21. 16-Bit Chip Select Little-Endian Data Format (Default)**

When a 16-bit chip select is in little-endian mode, the static bus controller accesses the least-significant halfword CD at physical offset 0 and accesses the most-significant halfword AB at physical offset 2. When the Au1 core is also in little-endian mode, the requested Au1 core offsets match the physical offsets of the 16-bit device. That is, the static bus controller and the Au1 core have the same view of memory. However, when the processor core is in big-endian mode, the default ordering of the static bus controller effectively reverses the ordering of the halfwords from what the big-endian Au1 core expects, as shown in Figure 3-22.

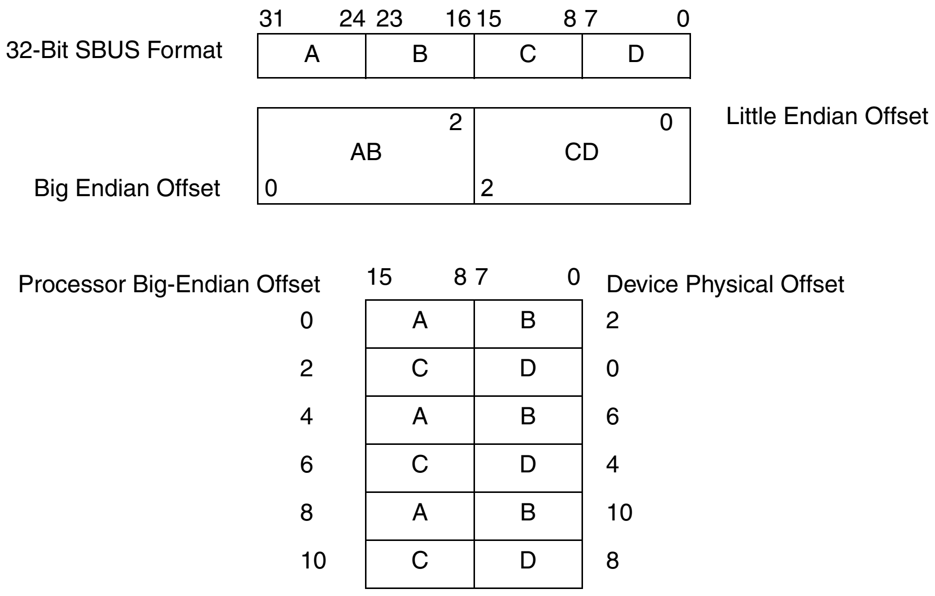


**Figure 3-22. Big-Endian Au1 Core and Little-Endian 16-Bit Chip Select**

For RAM memories, the halfword swapping has no side-effects because reads and writes are consistent. However, for ROM, Flash memories, and peripherals, be aware of the following side effects:

- For ROM and Flash, the memory contents are halfword-swapped throughout the entire 16-bit device memory.
- For Flash and peripherals, the programming register offsets are also halfword-swapped.

To prevent halfword swapping, configure the chip select for big-endian mode (**mem_stcfg**[BE] = 1) before accessing the memory. (If booting from static memory, see Section 9.3.1 "Endianness and 16-Bit Static Bus Boot" on page 182.) The static bus controller inverts RAD1 for transfers on 16-bit chip selects in big-endian mode, as shown in Figure 3-23 on page 72.

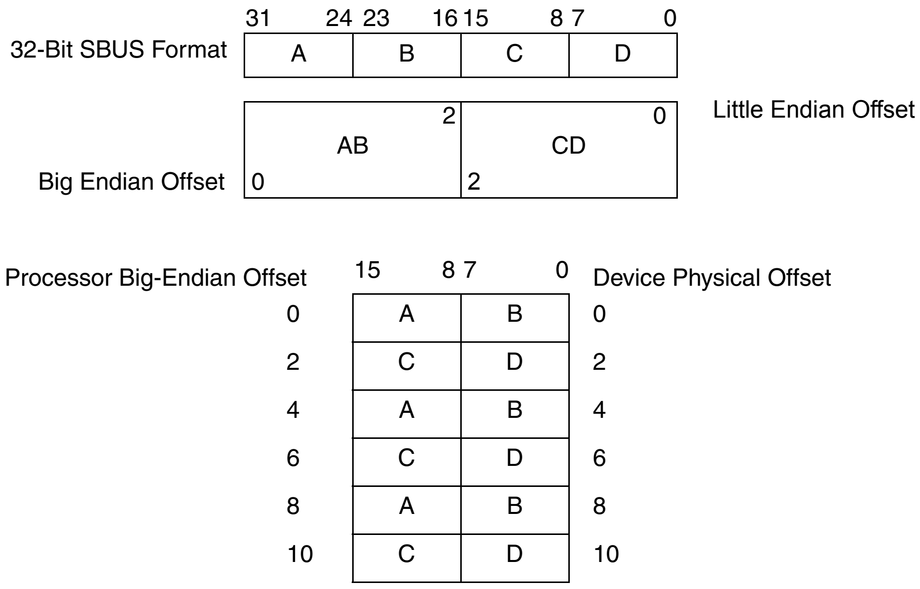


**Figure 3-23. Big-Endian Au1 Core and Big-Endian 16-Bit Chip Select**

# PCI 2.2 Bus Controller 4

The Au1500 processor contains a PCI 2.2 compliant controller to allow connection of external peripherals. The PCI bus may be operated at 33 or 66 MHz. The PCI clock can be driven by the Au1500 processor or by an external source. Only 32-bit wide operation is supported. The controller is capable of both executing processor initiated master transactions to PCI and accepting external PCI target transactions to the Au1500 processor's local memory.

The PCI controller can operate as either a host or satellite subsystem. When operating as a host, all PCI system configuration is done by the Au1500, and the PCI header space is not visible from the PCI bus. When operating as a satellite, PCI header space is only accessible by an external host, not by the Au1500 processor core. The Au1500 configuration registers are only visible from the processor in both host and satellite mode. The pin PCI_CFG determines whether the Au1500 processor operates as a host or a satellite.

The PCI controller contains an internal arbiter for five PCI devices including the Au1500 processor itself. This allows a glueless interface for up to four external PCI devices. The internal arbiter may be disabled if an external arbiter is desired.

The PCI controller contains 4 data FIFOs to enhance performance to and from PCI. These FIFOs are sized to minimize stalls and increase the System Bus (SBUS) efficiency during PCI bus transactions.

## 4.1 PCI Memory Map

Table 4-1 shows the location of PCI interface elements within the Au1500 memory map. Note that the addresses shown are 36-bit physical addresses. It is necessary to use the MMU to generate addresses to access PCI areas with the exception of the cacheable memory window, Au1500 processor configuration registers, and PCI header space.

**Table 4-1. PCI Memory Map**

| Physical Address | Function |
|---|---|
| 0x0 1400 50xx | Au1500 Configuration Registers |
| 0x0 1400 51xx | PCI Header Space |
| 0x4 xxxx xxxx | PCI Non-cacheable Memory Space |
| 0x5 xxxx xxxx | PCI I/O Space |
| 0x6 xxxx xxxx | PCI External Configuration Space |

A window within the first 4 GB can be configured to access cacheable PCI memory space. This window is configured using the **pci_cmem** register. Accesses to the PCI Non-cacheable Memory, I/O, and External Configuration spaces will cause memory, I/O, or configuration cycles to be run on PCI respectively. The lower 32 bits of the 36-bit address are reflected on the PCI bus for these transactions. The Au1500 can generate both Type 0 and Type 1 configuration cycles. Setting physical address bit [31] when accessing PCI external configuration space will generate a Type 1 configuration cycle. Clearing this bit will generate a Type 0 configuration cycle. On the PCI bus, bit [31] will be 0 for both Type 0 and Type 1 configuration cycles.

### 4.1.1 Local PCI Configuration Register Block

Table 4-2 shows the register map for the Au1500 processor local configuration register block. The upper 256 bytes map to the PCI configuration header space defined by the PCI specification. When the PCI controller is configured to operate in satellite mode, the PCI header space is not accessible through the Au1500 PCI configuration space. In satellite mode, the PCI header space is accessible through external PCI configuration cycles. When the PCI controller is configured to operate in host mode, the PCI header space is not accessible via PCI by external devices. All appropriate configuration registers in the lower 256 bytes of the address map, the Au1500 processor configuration registers, must be written to the appropriate values before writing the upper registers or configuring the system in either Host or Satellite mode. In Host mode, all configuration registers must be written before accessing external devices.

**Table 4-2. PCI Bus Controller Configuration Registers**

| Offset from 0x0 1400 5000 (Physical) | Register Name | Description |
|---|---|---|
| 0x000 | pci_cmem | PCI Cacheable Memory Region Register |
| 0x004 | pci_config | Configuration/Error Register |
| 0x008 | pci_b2bmask_cch | Back-to-Back Mask/Class Code High Register |
| 0x00C | pci_b2bbase0_venid | Back-to-Back Base 0/Vendor ID Register |
| 0x010 | pci_b2bbase1_subid | Back-to-Back Base 1/Subsystem ID Register |
| 0x014 | pci_mwmask_dev | MBAR Address Mask/Device ID Register |
| 0x018 | pci_mwbase_rev_ccl | MBAR Window Base/Revision/Class Code Low Register |
| 0x01C | pci_err_addr | PCI Error Address Register |
| 0x020 | pci_spec_intack | PCI Special/Int Ack Cycle Register |
| 0x100 | pci_id | Device and Vendor ID Register |
| 0x104 | pci_statcmd | Status and Command Register |
| 0x108 | pci_classrev | Class and Revision Code |
| 0x10C | pci_param | Parameter Register (BIST, header type, latency timer, cache line size) |
| 0x110 | pci_mbar | Memory Base Address Register (MBAR) |
| 0x140 | pci_timeout | Timeout Register |

### 4.1.1.1 PCI Cacheable Memory Region Register

The **pci_cmem** register configures the location of the cacheable memory window to PCI memory space. This allows processor initiated burst memory reads and writes to the PCI bus. Note that when the cacheable memory window is enabled, it must be mapped to CCA encoding 4. A cacheable memory window size of 256 KB to 2 GB is supported. A PCI memory transaction is executed when the following condition is met:

(physical_addr[31:18] & cm_mask) == cm_base

Bits [13:0] is the cacheable memory mask. Bits [27:14] is the base address of the cacheable memory region. Bit [28] is the cacheable memory enable.

**pci_cmem - PCI cacheable memory region** **Offset = 0x000**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | E | CM_BASE | | | | | | | | | | | | | | CM_MASK | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:29 | — | Reserved, should be cleared. | R | 0 |
| 28 | E | Enable.<br>0 Disable PCI cacheable memory.<br>1 Enable PCI cacheable memory. | R/W | 0 |
| 27:14 | CM_BASE | Specifies bits [31:18] of the starting/base address for the PCI cacheable memory region | R/W | 0x0 |
| 13:0 | CM_MASK | Specifies which bits of physical_addr[31:18] are used to decode the PCI cacheable memory region. | R/W | 0x0 |

### 4.1.1.2 PCI Configuration/Error Register

The **pci_config** register configures the general operation of the PCI interface. It allows configuration of the Au1500 on-chip arbiter, interrupt generation, and improved error reporting. Bits [31:28] are the upper error address of the PCI Error Address register. Bits [27:22] determine what conditions were present during an error; see Section 4.3.6 "PCI Errors" on page 83 for further discussion. These bits can be used in conjunction with the PCI Status register defined in the PCI 2.2 specification.

Bit [19] enables byte masking for byte and halfword reads from memory or the configuration space. Note that if BME (bit 19) is not set, reads less than 32 bits from these areas are not supported.

Bit [16] is the non-cacheable bit. If set, all PCI master initiated accesses to the Au1500 memory space will be run as non-cacheable. The default is for all accesses to be marked cacheable on the Au1500 SBUS. Bits [13:8] are the PCI interrupt mask. If a bit is a zero, the appropriate condition will not assert the PCI interrupt. This mask can be used to assist debug by only interrupting on specific conditions. Bits [7:4] are used to control address/data manipulation for big endian support. Refer to Section 4.3.8 "Endian Support" on page 84 for a detailed functional description of these bits. Bits [3:0] configure and enable the on-chip arbiter.

**pci_config - PCI configuration/error** **Offset = 0x004**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ERR_ADDR | | | | ERD | ET | EF | EP | EM | BM | ▨ | PD | BME | ▨ | ▨ | NC | IA | ▨ | IP | IS | IMM | ITM | ITT | IPB | SIC | | ST | SM | AEN | R2H | R1H | CH |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:28 | EA[35:32] | These bits are the upper 4 bits of the error address register. | R/W | 0 |
| 27 | ERD | PCI error occurred on a read/write#. If an error is detected for a PCI access, ERD reflects the direction of the transaction.<br>0 Error occurred on a write.<br>1 Error occurred on a read. | R/W | 0 |
| 26 | ET | PCI error occurred while the Au1500 was a target. | R/W | 0 |
| 25 | EF | PCI fatal error detected. These errors include Master and Target Aborts as well as certain errors not specified in the PCI Specification, but which may be important to the user. | R/W | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 24 | EP | PCI parity error detected. Parity errors are detected either by the Au1500 bus interface on data it receives, or by the target device on the PCI bus and passed to the Au1500 via PCI_PERR#. | R/W | 0 |
| 23 | EM | Multiple PCI errors detected. If an error was detected while EF or EP is asserted, EM is set. | R/W | 0 |
| 22 | BM | PCI Arbiter detected bad master—does not assert PCI_FRAME# (or negate PCI_REQ*n*#) within 16 clocks of being granted the PCI bus. | R/W | 0 |
| 21 | — | Reserved, should be cleared. | R | 0 |
| 20 | PD | PCI Disable. When this bit is set the PCI controller responds to target accesses with a PCI RETRY. | R/W | 1 |
| 19 | BME | Byte mask enable for reads. This bit applies only to 8- and 16-bit *read* accesses to *memory* and the *configuration space*. (The appropriate byte lane masking is automatically applied for I/O reads less than 32 bits. Also, *writes* of any length and destination always have appropriate byte lane masking.)<br>0 Do not apply byte lane masking during 8-, 16-bit reads. (All byte enables are asserted regardless of transaction size.)<br>1 Mask the appropriate byte lanes during 8-, 16-bit reads. | R/W | 0 |
| 18:17 | — | Reserved, should be cleared. | R | 0 |
| 16 | NC | Target accesses to Au1500 memory are marked as non-coherent if this bit is set. | R/W | 0 |
| 15 | IA | PCI_INTA# Enable. This bit is reflected in bit 8 of offset 0x3C in configuration space. If the Au1500 can generate a PCI interrupt this bit should be set. | R/W | 0 |
| 14 | — | Reserved, should be cleared. | R | 0 |
| 13 | IP | Assert the interrupt signal back to the interrupt controller if PCI_PERR# is detected. This is bit 15 of the status register in PCI space. | R/W | 0 |
| 12 | IS | Assert the interrupt signal back to the interrupt controller if a PCI_SERR# is generated. This is bit 14 of the status register in PCI space. | R/W | 0 |
| 11 | IMM | Assert the interrupt signal back to the interrupt controller if a Master-Abort is asserted and the Au1500 processor is the master. This is bit 13 of the status register in PCI space. | R/W | 0 |
| 10 | ITM | Assert the interrupt signal back to the interrupt controller if a Target-Abort is detected while the Au1500 processor is a Master. This is bit 12 of the status register in PCI space. | R/W | 0 |
| 9 | ITT | Assert the interrupt signal back to the interrupt controller if a Target-Abort is asserted and the Au1500 is the Target. This is bit 11 of the status register in PCI space. | R/W | 0 |
| 8 | IPB | Assert the interrupt signal back to the interrupt controller if a PCI_PERR# is detected, the Au1500 is the bus master and the perr_enable bit is set in PCI configuration space. This is bit 8 of the status register in PCI space. | R/W | 0 |
| 7:6 | SIC | These bits determine the address and data swapping of I/O and Configuration cycles initiated by the Au1500 to/from external devices. Please refer to Section 4.3.8 "Endian Support" on page 84 for a detailed description of these bits. | R/W | 0 |
| 5 | ST | Swap data on PCI target transactions initiated by external PCI devices to/from Au1500 memory. | R/W | 0 |
| 4 | SM | Swap data on Au1500 processor initiated master memory transactions to/from PCI. | R/W | 0 |
| 3 | AEN | When set this bit enables the Au1500 internal arbiter. | R/W | 0 |
| 2 | R2H | When set, PCI_REQ2# arbitrates into the high priority arbiter. | R/W | 0 |
| 1 | R1H | When set, PCI_REQ1# arbitrates into the high priority arbiter. | R/W | 0 |
| 0 | CH | When set, the Au1500 processor arbitrates into the high priority arbiter. | R/W | 0 |

### 4.1.1.3 PCI Back-to-Back Mask/Class Code High, Base0/PCI Subsystem Vendor ID, and Base1/PCI Subsystem ID Registers

Master initiated fast back-to-back transfers require that the subsequent access is to the same target as the previous access. The Au1500 processor allows the programmer to set up two target back-to-back windows. The Mask Register configures the size of the windows, while the base registers set the base address of each target window. A fast back-to-back transaction is attempted when ((*physical_addr & b2b_mask == b2b_base0)* | (*physical_addr & b2b_mask == b2b_base1)*). Fast back-to-back is enabled by setting bit 9 of the **pci_statcmd** register which is at offset 0x104 in the PCI Configuration space.

The lower 16 bits of the Fast Back-to-Back Mask register sets the Class Code High field. This field is the upper 16 bits of the Class Code defined by the PCI 2.2 specification. It is combined with the Class Code Low field to form the full 24-bit Class Code. This field can be written from the processor only and are reflected out to the PCI bus on configuration read cycles to address 0x08 when the Au1500 is configured as a satellite.

The lower 16 bits of the Fast Back-to-Back Base0 register sets the Subsystem Vendor ID, and the lower 16 bits of the Fast Back-to-Back Base1 register is the Subsystem ID. These fields can be written from the processor only, and allow for identification of different systems which contain an Au1500. These fields are reflected out to the PCI bus on configuration read cycles to address 0x2C when the Au1500 is configured as a satellite.

**pci_b2bmask_cch - PCI back-to-back mask/class code high** **Offset = 0x008**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | B2BMASK | | | | | | | | | | | | | | | | CCH | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | B2BMASK | This field is the mask used to qualify fast back-to-back transfers. | R/W | 0 |
| 15:0 | CCH | Class Code High. These bits are reflected as the upper two bytes of the class code field in PCI configuration register space. | R/W | 0 |

**pci_b2bbase0_venid - PCI back-to-back base** **Offset = 0x00C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | B2BBASE0 | | | | | | | | | | | | | | | | SVID | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | B2BBASE0 | This field specifies a base address for fast back-to-back transfers. | R/W | 0 |
| 15:0 | SVID | This field is reflected as the subsystem vendor ID in PCI configuration space. | R/W | 0 |

**pci_b2bbase1_subid - PCI back-to-back base** **Offset = 0x010**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | B2BBASE1 | | | | | | | | | | | | | | | | SUBID | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | B2BBASE1 | This field specifies a base address for fast back-to-back transfers. | R/W | 0 |
| 15:0 | SUBID | This field is reflected as the subsystem ID in PCI configuration space. | R/W | 0 |

#### 4.1.1.4 PCI Memory Window Mask/Device ID and Base/Revision/Class Code Low Registers

The Au1500 processor allows PCI masters access to a window into memory. This window can be sized from 64 KB to 2 GB. The Memory Window Mask register configures the size of the memory window visible from the PCI bus. The Memory Window Base register configures the location of the memory window visible from the PCI bus. This allows a mapping from the PCI address to memory address. If a one to one mapping is required, the Memory Window Base register should be written with the same value as the Memory Base Address register which is located at offset 0x110 of the PCI Configuration space in Host mode. An external PCI access hits in the memory window when (*pci_addr & mwmask == pci_mbar*). The physical address to the Au1500 memory system becomes ((*pci_addr & mwmask#) | mwbase*).

The lower 16 bits of the Memory Window Mask register is the Device ID field as defined by the PCI 2.2 specification. This field can be written from the processor only and is reflected out to the PCI bus on configuration read cycles to address 0x00 when the Au1500 is configured as a satellite.

The lower 8 bits of the Memory Window Base register is the Class Code Low field. This field is the lower 8 bits of the Class Code defined by the PCI 2.2 specification. It is combined with the Class Code High field to form the full 24 bit Class Code. This field can be written from the processor only and are reflected out to the PCI bus on configuration read cycles to address 0x08 when the Au1500 processor is configured as a satellite.

Bits [15:8] of the Memory Window Base register is the Revision ID field as defined by the PCI 2.2 specification. This field can be written from the processor only and is reflected out to the PCI bus on configuration read cycles to address 0x00 when the Au1500 is configured as a satellite.

**pci_mwmask_dev - PCI memory window mask/device ID** **Offset = 0x014**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | MWMASK | | | | | | | | | | | | | | | | DEVID | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | MWMASK | This field sets the mask used to qualify target accesses to the PCI visible memory window. | R/W | 0 |
| 15:0 | DEVID | This field is reflected as the device ID in PCI configuration space. | R/W | 0 |

**pci_mwbase_rev_ccl - PCI memory window base/revision/class code low** **Offset = 0x018**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | MWBASE | | | | | | | | | | | | | | | | REVID | | | | | | | | CCL | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | MWBASE | This field sets the base address used to qualify target accesses to the PCI visible memory window. | R/W | 0 |
| 15:8 | REVID | This field is reflected as the revision ID in PCI configuration space. | R/W | 0 |
| 7:0 | CCL | These bits are reflected as the low byte of the class code field in PCI configuration space. | R/W | 0 |

#### 4.1.1.5 PCI Error Address Register

This register captures error addresses on target writes, master reads and master writes. The upper 4 bits of the address are captured in the PCI Configuration/Error register. This address is intended to be used in conjunction with the PCI header error bits described in the PCI 2.2 specification. The error address captured is valid when the EF or EP bit is set in the Configuration/Error register. The address for target write errors may not be the specific identical error address due to synchronization issues. If multiple errors occur, this register contains the address of the first error.

**pci_err_addr - PCI error address** **Offset = 0x01C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ERRADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

#### 4.1.1.6 PCI Special/Int Ack Register

This register allows the programmer to initiate special and interrupt acknowledge cycles on the PCI bus. A read to this register invokes an interrupt acknowledge cycle, and a write executes a special cycle. The register value read is undefined and the actual value written is ignored.

**pci_specintack** **Offset = 0x0020**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SPECINTACK[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x |

### 4.1.2 PCI Visible Configuration Registers

The following registers are defined in *PCI Local Bus Specification, revision 2.2*. These registers are visible from the PCI bus as configuration space when the Au1500 is operating in satellite mode. When the Au1500 processor is operating in host mode these registers are available in the local configuration register block. For detailed description of these registers and their fields please refer to *PCI Local Bus Specification, revision 2.2*.

The offsets given in the register descriptions below are the offsets within the Au1500 local configuration register block. The offset in the PCI configuration space is given by the PCI specification.

#### 4.1.2.1 PCI Device/Vendor ID Register

The **pci_id** register is defined in the PCI 2.2 specification. This register is visible at offset 0x00 in configuration header space from the PCI bus when the device is configured to operate in satellite mode. In host mode, this register is available at offset 0x100 of the PCI local configuration space.

**pci_id - PCI configuration device/vendor ID** **Offset = 0x0100**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | DID | | | | | | | | | | | | | | | | VID = 0x1755 | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

#### 4.1.2.2 PCI Status and Command Register

The **pci_statcmd** register is defined in the PCI 2.2 specification. This register is visible at offset 0x04 in configuration header space from the PCI bus when the device is configured to operate in satellite mode. In host mode and also again in satellite mode, this register is available at offset 0x104 of the PCI local configuration space.

**pci_statcmd - PCI configuration status/command** **Offset = 0x0104**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | STATUS | | | | | | | | | | | | | | | | CMD | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

#### 4.1.2.3 PCI Class and Revision Register

The **pci_classrev** register is defined in the PCI 2.2 specification. The CLASS field is constructed by concatenating the class code high and class code low fields in the local configuration block. This register is visible at offset 0x08 in configuration header space from the PCI bus when the device is configured to operate in satellite mode. In host mode, this register is available at offset 0x108 of the PCI local configuration space.

**pci_classrev - PCI configuration class code/revision** **Offset = 0x0108**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | CLASS | | | | | | | | | | | | | | | | | | | | | | | | REV | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

#### 4.1.2.4 PCI Parameter Register

The **pci_param** register is defined in the PCI 2.2 specification. This register is visible at offset 0x0C in configuration header space from the PCI bus when the device is configured to operate in satellite mode. In host mode, this register is available at offset 0x10C of the PCI local configuration space.

**pci_param - PCI configuration parameters** **Offset = 0x010C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIST | | | | | | | | HT | | | | | | | | LT | | | | | | | | CLS | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

#### 4.1.2.5 PCI Memory Base Register

The **pci_mbar** register configures the PCI address base of the window into the Au1500 processor memory space. The **pci_mbar** register is defined in the PCI 2.2 specification. This register is visible at offset 0x10 in configuration header space from the PCI bus when the device is configured to operate in satellite mode. In host mode, this register is available at offset 0x110 of the PCI local configuration space.

**pci_mbar - PCI configuration memory base address** **Offset = 0x0110**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | MBA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

#### 4.1.2.6 PCI Timeout Register

The **pci_timeout** register specifies the maximum number of attempted retries per transaction and the maximum time spent waiting for a target to be ready for a transfer.

The **pci_timeout** register is defined in the PCI 2.2 specification. This register is visible at offset 0x40 in configuration header space from the PCI bus when the device is configured to operate in satellite mode. In host mode, this register is available at offset 0x140 of the PCI local configuration space.

**pci_timeout - PCI configuration timeout** **Offset = 0x0140**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | MR | | | | | | | | TO | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | — | Reserved. | R/W | 0 |
| 15:8 | MR | Max retries. Contains the maximum number of retries to be attempted per transaction. Valid values range from 1 to 255. Note that a value of 0 disables this function, which means no limit is placed on the number of retries attempted. | R/W | 0x80 |
| 7:0 | TO | Target-ready (PCI_TRDY#) timeout. Contains the maximum number of PCI clocks to wait for PCI_TRDY# assertion before terminating a transfer. Valid values range from 1 to 255. Note that a value of 0 disables this function, which means no time limit is placed on waiting for PCI_TRDY#. | R/W | 0x80 |

### 4.1.3 Accesses to Processor Memory from PCI

The Au1500 processor supports one memory window visible from the PCI bus. This memory window can be configured to a minimum size of 64 KB and a maximum size of 2 GB. The PCI address base of the window is set in the **pci_mbar** register. The location of the window within the Au1500 address space is configured in the **pci_wbase** register. The size is configured in the **pci_wmask** register.

Only static memory accesses and SDRAM memory accesses are supported from PCI. Accesses to Au1500 processor peripherals and non-prefetchable external devices on the static interface are not supported. Also, all reads from Au1500 memory initiated from PCI will be prefetched.

The Au1500 processor cache will snoop accesses to cacheable memory from PCI if the NC bit is clear in the PCI Configuration/Error register and substitute cache contents when a hit occurs. By locking an area of memory into the cache it is possible to transfer information to and from PCI without causing external memory cycles.

## 4.2 PCI Arbiter

The Au1500 contains an on-chip PCI arbiter. This arbiter supports 4 external devices and can be disabled if the use of an external arbiter is desired. When the on-chip arbiter is enabled, the Au1500 processor has four request inputs and four grant outputs. When an external arbiter is used, the input PCI_REQ0# is re-defined to be PCI_GNT0#, and the output PCI_GNT0# is re-defined to be PCI_REQ0#. The other request/grant signals are unused when an external arbiter is used.

A two-level fair algorithm scheme is implemented, which ensures each master access to the bus independent of other requests. The two levels of arbitration allows bus masters to be assigned high or low priority according to their need to use the bus. Bus masters with a greater need to use the bus should be assigned to the high priority arbiter versus the low priority arbiter. As a group, the low priority requests will arbitrate into the high priority arbiter. In this way, high-priority bus masters cannot dominate the bus to the point of excluding low-priority masters which are continually requesting the bus.

Two of the external master's and the Au1500 are configurable for high/low priority. The other two external masters are tied to either a fixed high or low priority, as shown in Table 4-3. Priorities should be initialized by software before masters begin to request use of the bus.

**Table 4-3. PCI Arbiter Priority Configuration**

| Master | Priority |
|---|---|
| External Master 3 | High Priority |
| External Master 2 | pci_config[2] |
| External Master 1 | pci_config[1] |
| External Master 0 | Low Priority |
| Au1500 | pci_config[0] |

The arbiter implements bus parking so that when no other masters are requesting the use of the bus, the last master to have ownership of the bus retains that ownership until another request is received. Out of system reset, the Au1500 processor is the default owner and assumes responsibility for driving the PCI address bus, C/BE bus and PAR. This ensures that no excess current is drawn due to floating nodes.

If a master receives a grant by requesting the bus and does not initiate a transaction within 16 PCI clocks after receiving the grant, the master is assumed to be broken and an error is reported to the bus controllers' PCI configuration/error register, **pci_config**[22].

## 4.3 Implementation Specifics

The following sections describe Au1500 implementation specific issues.

### 4.3.1 PCI Clock Generation

The Au1500 processor provides the capability to internally generate a clock on the PCI clock output pin (PCI_CLKO). For systems where internal generation of the PCI clock is insufficient, the Au1500 processor also provides the option to use an externally generated PCI clock.

#### 4.3.1.1 Clock Connections

For internal PCI clock generation, the PCI_CLKO pin is used to generate the PCI clock. Note that the generated clock must be fed back into the PCI_CLK pin to drive the PCI interface logic. If an externally generated clock is used, the clock needs to be driven into the PCI_CLK pin, and the PCI_CLKO pin is left unconnected.

#### 4.3.1.2 Limitations on Internal PCI Clock Generation

The internal frequency generation circuitry of the Au1500 processor is flexible, but there are limitations on the frequencies that can be generated. Either the CPU PLL or the AUX PLL may be used as the root source of the PCI clock generator. The AUX PLL has the advantage that it is independent of CPU frequency. However, the system designer must take into account that the AUX PLL is also used to generate the USB clocks, which are required to be 48 MHz.

All of the frequencies in the system are generated from the 12 MHz oscillator. The AUX PLL multiplies this up to some base frequency, from which the target frequency may be divided. The dividers can be programmed to divide only by even values, which places further constraints on the available frequencies.

If the AUX PLL is to be used to generate both the USB clock and the PCI clock, it should be programmed to 384 MHz by programming the **sys_auxpll** register to 32. This allows generation of the 48 MHz USB clock by setting the frequency generator to divide by eight. Setting a frequency generator to divide by six can generate a 64 MHz PCI clock, or dividing by twelve can generate a 32 MHz clock. If the AUX PLL is *not* used for USB clock generation, it can be programmed to 396 MHz and yield 66 MHz and 33 MHz with the same dividers. Alternately, if the CPU is being run at 396 MHz, the CPU PLL can be used as the root clock for a frequency generator with the same effect.

#### 4.3.1.3 Programming Example

The CPU PLL and AUX PLL frequencies are programmed by writing to the **sys_cpupll** and **sys_auxpll** registers. The value written is multiplied by twelve to yield the resulting PLL frequency. The following configuration example uses the AUX PLL and results in a 64 MHz PCI clock on the PCI_CLKO pin:

1) Write 0x20 to **sys_auxpll** for an AUX PLL frequency of 384 MHz.
2) Choose a frequency generator for the PCI clock by programming the **sys_clksrc** register:
   - Write 0b100 to the MPC field. (For this example frequency generator 2 is used.)
   - Clear the CPC bit so that the PCI clock is not further divided.
3) Program the **sys_freqctrl0** register:
   - Program the FRDIV2 field to 0x02 so that the frequency is divided by six.
   - Set the FE2 bit to enable output from frequency generator 2.
   - Set the FS2 bit to select the AUX PLL as the clock source.

For a description of the clock manager registers, see Section 8.1 "Clocks" on page 154.

### 4.3.2 PCI Interrupts

The PCI_INT[A:D]# signals are only inputs to the Au1500 processor. These signals are routed to the interrupt controller so that the Au1500 can be interrupted on the assertion of these signals. If software needs to generate an interrupt to an external device, a GPIO should be used.

### 4.3.3 PCI Reset (PCI_RST# and PCI_RSTO#)

The PCI_RST# pin is input only. For systems which require the PCI bus be held in reset as the Au1500 powers up, GPIO[200] (unlike other GPIOs) is configured to output a 0 on power up. System software can use GPIO[200] as a PCI reset output signal (labeled as PCI_RSTO#) to reset the PCI bus without resetting the processor. When the PCI bus is reset, the Au1500 processor PCI configuration registers are also reset. If the state of PCI_RST# is needed by software, another GPIO signal should be used to monitor the status of this pin.

### 4.3.4 Boot from PCI

The Au1500 has the ability to boot from a memory device on PCI. When the ROMSEL and ROMSIZE pins are both tied high, the Au1500 processor fetches the reset vector from PCI memory space. It is required that the boot memory be located at the Au1500 reset vector physical address because no address translation occurs on master transactions to PCI. The Au1500 processor may not be the clock source or the PCI reset source signal (PCI_RSTO# on GPIO[200]) if booting from PCI, since code is required to configure these functions. Also, the internal arbiter will be used when booting from PCI.

### 4.3.5 Host/Satellite Configuration (PCI_CFG)

The PCI_CFG pin determines whether the PCI interface behaves as a host or satellite. If this pin is grounded at reset the PCI interface will be in satellite mode, with configuration registers visible to the PCI bus. If this pin is pulled high at reset the interface will be set to host mode. In host mode the PCI configuration registers are only visible from the internal SBUS and the Au1500 is responsible for configuration of the bus.

### 4.3.6 PCI Errors

The Au1500 processor error registers (the PCI Configuration/Error register **pci_config** and the PCI Error Address register **pci_err_addr**) are intended to be used to supplement the PCI 2.2 Configuration Status bits.

PCI error conditions are reflected in the state of **pci_config**[27:22]. These bits allow software to determine if one or more errors occurred and the cause of at least the first error. Bit 23 (EM) is active only when a second error of any kind shown occurs—that is, when an error is pending while another error occurs. The EM bit allows software to know that multiple errors have occurred since the last time the error conditions were cleared. Once errors are cleared, EM is also cleared so that a subsequent error will be logged as a 'first error'. Table 4-4 shows the possible error conditions reflected in **pci_config**[27:24].

**Table 4-4. PCI Error Conditions on pci_config[27:24]**

| PCI Mode | PCI Error Condition | Bit 27 ERD | Bit 26 ET | Bit 25 EF | Bit 24 EP |
|---|---|---|---|---|---|
| Target | Parity Error is detected by the Au1500 on a PCI Write to Au1500 memory (and PCI_PERR# is asserted) | 0 | 1 | 0 | 1 |
| Master | PCI Master Abort on Configuration Space Read access, due to no response from target device on PCI_DEVSEL# (no device present on selected PCI_IDSEL) | 1 | 0 | 1 | 0 |
| Master | PCI Master Abort on Configuration Space Write access, due to no response from target device on PCI_DEVSEL# (no device present on selected PCI_IDSEL) | 0 | 0 | 1 | 0 |
| Master | PCI Master Abort on any PCI Read Access, due to transaction not completed in time—address claimed, but PCI_TRDY# or PCI_STOP# not asserted within the number of clocks programmed in **pci_timeout**[TO]. | 1 | 0 | 1 | 0 |
| Master | PCI Master Abort on any PCI Write Access, due to transaction not completed in time—address claimed, but PCI_TRDY# or PCI_STOP# not asserted within the number of clocks programmed in **pci_timeout**[TO]. | 0 | 0 | 1 | 0 |
| Master | PCI Access retried to the same device signaling RETRY more than the number of times programmed in **pci_timeout**[MR]. | 0 | 0 | 1 | 0 |
| Master | PCI Target abort signaled by the addressed Target device | 0 | 0 | 1 | 0 |
| Master | Au1500 detects a PCI parity error condition on data received from a target during a read | 1 | 0 | 0 | 1 |
| Master | PCI Target signals a PCI parity error condition on PCI_PERR#, from data received from the Au1500 on a PCI Write Transaction | 0 | 0 | 0 | 1 |

There may be error situations where **pci_config** and **pci_err_addr** are not updated when an error occurs. On Au1500 initiated accesses that end in an fatal error, up to two transactions after the error could be discarded because of the asynchronous nature between the PCI bus and the SBUS. However, the address of the transaction to which the fatal error occurred will still be captured.

### 4.3.7 Readline/ReadMultiple Support

The Au1500 processor will generate Read Line and Read Multiple commands on the PCI bus for Au1500 initiated PCI accesses. If the burst size of the transaction is equal to the cacheline size programmed into the command/status configuration register, then the transaction will be run as a read line transaction on PCI. If the burst size of the transaction is greater than the non-zero cacheline size and a multiple of the non-zero cacheline size, then a read multiple command will be run.

### 4.3.8 Endian Support

The Au1500 processor PCI block contains support for data swapping and address manipulation to support big endian systems. The Au1 processor core can be set to big endian or little endian. The memory interfaces write according to the byte masks and read the entire 32-bit word. With respect to the Au1500, if the processor is in little endian mode, all accesses are correct since PCI is little endian.

For big endian cores there is a choice to be made. The PCI data bus can be connected the same as for little endian or the bytes can be swapped. Table 4-5 shows how the bytes are connected for swapped configurations:

**Table 4-5. Data Bus Swapping**

| SBUS | PCI | Swapped PCI |
|---|---|---|
| 31:24 | 31:24 | 7:0 |
| 23:16 | 23:16 | 15:8 |
| 15:8 | 15:8 | 23:16 |
| 7:0 | 7:0 | 31:24 |

For memory accesses between the Au1500 processor and external devices, software can choose whether or not to swap the bytes. If one selects to have no swapping, then 32-bit word accesses will read/write values as expected but byte accesses will be big endian from the core and little endian from PCI. For example, if a PCI device has 32-bit control registers and they were accessed with lw/sw from the core, the bit layout would match the devices' specification. But if one of the registers pointed to a buffer in memory that was written/read as a byte stream, the external PCI device would access the data little endian and the processor would access it big endian. Unless the PCI external device has a big endian mode the data would need to be byte swapped by software. The second option would be to swap the data bytes causing the byte accesses to be correct at the cost of word registers needing to be swapped by software. The Au1500 provides the hardware for swapping the data bytes. The endian policy of what to do is left to the system designer based on what accesses are easier to swap, if any.

Four control bits are provided to allow system designers to program when and how swapping is done. The four control bits are outlined in Table 4-6:

**Table 4-6. Swapping Configurations**

| pci_config Bits | Function |
|---|---|
| 4 (SM) | If this bit is set, then for accesses to PCI memory space that are initiated by the Au1 core or DMA devices on the Au1500 SBUS the bytes and byte masks are swapped. If this bit is clear then the bytes are not swapped. |
| 5 (ST) | If this bit is set, then for accesses to memory from an external PCI master the bytes and byte masks are swapped. If this bit is clear then the bytes are not swapped. |
| 7:6 (SIC) | For accesses to PCI I/O space initiated by the Au1 core or DMA devices on the Au1500 processor SBUS, the bytes must be swapped or the address changed in big endian mode so that valid PCI I/O accesses can be generated for byte and halfword accesses. The four encodings provide the following actions for both I/O and configuration accesses:<br>00 The address and byte masks are unchanged.<br>01 On byte accesses, invert the lower 2 address bits. On halfword accesses, invert the lower address bit. This encoding only affects I/O accesses.<br>10 On byte and halfword accesses swap the data bytes and leave the address unchanged.<br>11 Swap the data bytes on all accesses. |

Table 4-7 shows the supported modes for the SM, ST and SIC bits based on the endianness of the processor.

**Table 4-7. Supported Endian/Swapping Modes**

| Mode | Big Endian | Little Endian |
|---|---|---|
| SM | yes | yes |
| ST | yes | yes |
| SIC = 00 | no | yes |
| SIC = 01 | yes (only affects I/O transactions) | no |
| SIC = 10 | yes | no |
| SIC = 11 | yes | no |

### 4.3.9 PCI Signals

The signals supporting the PCI bus are shown in Table 4-8.

**Table 4-8. PCI Bus Support Signals**

| PCI Signal | Input/Output | Definition |
|---|---|---|
| PCI_AD[31:0] | I/O | PCI Address/Data. |
| PCI_CBE[3:0]# | I/O | PCI Bus Command and Byte Enables. |
| PCI_FRAME# | I/O | PCI Cycle Frame. |
| PCI_IRDY# | I/O | PCI Initiator Ready. |
| PCI_TRDY# | I/O | PCI Target Ready. |
| PCI_STOP# | I/O | PCI Target Stop. |
| PCI_PERR# | I/O | PCI Parity Error. |
| PCI_SERR# | O | PCI System Error. |
| PCI_PAR | I/O | PCI Parity. |
| PCI_DEVSEL# | I/O | PCI Device Select. |
| PCI_IDSEL | I | PCI initialization device select input. |
| PCI_LOCK# | I/O | PCI Lock for Atomic Operations. |
| PCI_REQ[3:0]# | I | PCI Arbiter Bus Request Inputs. |
| PCI_GNT[3:0]# | O | PCI Arbiter Bus Grant Outputs. |
| PCI_INT[A:D]# | I/O | PCI interrupts. |
| PCI_CLKO | O | PCI Clock Output.<br>**Termination Note:** Requires an external 22-Ω resistor placed in series within 0.5 inches of the part. |
| PCI_CLK | I | PCI Input Clock.<br>**Termination Note:** Should be tied low through a resistor when the PCI bus is not used. |
| PCI_RST# | I | PCI Reset Input.<br>**Termination Note:** Should be tied low through a resistor when the PCI bus is not used. |
| PCI_RSTO# (GPIO[200]) | O | PCI Reset Output. Unlike other GPIOs, GPIO[200] defaults as an output with a zero voltage level coming out of reset. For this reason, GPIO[200] can be controlled by system software to act as a PCI reset output signal (labeled PCI_RSTO#) if needed. |

**Table 4-8. PCI Bus Support Signals (Continued)**

| PCI Signal | Input/Output | Definition |
|---|---|---|
| PCI_CFG | I | PCI configuration. Determines the mode (satellite or host) for the PCI interface at hardware reset.<br>For satellite mode, tie PCI_CFG low.<br>For host mode, tie PCI_CFG high.<br>Note that PCI_CFG must not change state after system power up. |

### 4.3.10 Other Notes

The Au1500 PCI contoller cannot be used with external PCI-to-PCI bridges that have PCI bus-mastering devices on the secondary bus which target the Au1500 memory.

If the parity enable bit is cleared, the Au1500 will assert PCI_DEVSEL# on target writes in one PCI_CLK. If the parity error enable bit is set, the Au1500 processor will drive PCI_DEVSEL# in two cycles of PCI_CLK on target writes.

The Au1500 will not generate or respond to Dual Address Cycles. External PCI masters should not send Dual Address Cycles to the Au1500 processor.

The Au1500 processor will not respond to I/O Cycles. External PCI masters should not send I/O cycles to the Au1500 processor.

The Au1500 will not respond to Special Cycles. External PCI masters should not send Special cycles to the Au1500 processor.

Only Type 0 Configuration cycles are decoded by the Au1500. The Au1500 can generate both Type 0 and Type 1 configuration cycles.

The PCI_SERR# signal is only an output. The Au1500 processor will attempt to complete transactions which receive a PCI_SERR#.

The Memory Write and Invalidate command is not supported by the Au1500. Setting the Memory Write and Invalidate capable bit in the PCI configuration header has no effect.

The Au1500 processor supports PCI cacheline sizes of eight or four 32-bit words.

# 5 DMA Controller

The Au1500 processor contains an eight-channel DMA controller. Each channel is capable of transferring data between memory and any of the integrated peripherals or between memory and a memory-mapped FIFO through the Static Controller using a GPIO as a request.

Note that memory-to-memory transfers are not supported by the DMA controller. That is, one side of the DMA transfer must have an incrementing address (memory buffer), while the other side must have a *fixed* address (FIFO).

GPIO[4], GPIO[5], GPIO[208] and GPIO[209] can be programmed to act as external DMA request signals. When configured for this special system function, the pins are labeled as follows:

- GPIO[4] becomes DMA_REQ0.
- GPIO[5] becomes DMA_REQ1.
- GPIO[208] becomes DMA_REQ2.
- GPIO[209] becomes DMA_REQ3.

See Section 5.2 "Using GPIO as External DMA Requests (DMA_REQn)" on page 92 to configure these GPIO signals to act as DMA requests.

## 5.1 DMA Configuration Registers

Each channel of the DMA is configured by a register block. A channel register block contains seven registers. The 36-bit physical base address of the register block for each channel is shown in Table 5-1.

**Table 5-1. DMA Channel Base Addresses**

| DMA Channel | Base Address | KSEG1 Base Address | Priority |
|---|---|---|---|
| dma0 | 0x0 1400 2000 | 0xB400 2000 | 0 (highest) |
| dma1 | 0x0 1400 2100 | 0xB400 2100 | 1 |
| dma2 | 0x0 1400 2200 | 0xB400 2200 | 2 |
| dma3 | 0x0 1400 2300 | 0xB400 2300 | 3 |
| dma4 | 0x0 1400 2400 | 0xB400 2400 | 4 |
| dma5 | 0x0 1400 2500 | 0xB400 2500 | 5 |
| dma6 | 0x0 1400 2600 | 0xB400 2600 | 6 |
| dma7 | 0x0 1400 2700 | 0xB400 2700 | 7 (lowest) |

Each register block contains the registers shown in Table 5-2.

**Table 5-2. DMA Channel Configuration Registers**

| Offset | Register Name | Description |
|---|---|---|
| 0x0000 | dma_moderead | Read channel mode register |
| 0x0000 | dma_modeset | Set bits in channel mode register |
| 0x0004 | dma_modeclr | Clear bits in channel mode register |
| 0x0008 | dma_peraddr | Address of peripheral FIFO |
| 0x000C | dma_buf0addr | Starting address of buffer 0 |
| 0x0010 | dma_buf0size | Transfer size and remaining transfer count for buffer 0 |
| 0x0014 | dma_buf1addr | Starting address of buffer 1 |
| 0x0018 | dma_buf1size | Transfer Size and remaining transfer count for buffer 1 |

Table 5-3 shows the different peripherals that are capable of DMA. The device ID, transfer size, and transfer width (device FIFO width) are configurable fields in the **dma_mode** register. The FIFO address is a physical address whose address should be programmed in the **dma_peraddr** register and in the DAH field of the **dma_mode** register.

Enabling multiple DMA channels with the same device ID is undefined.

**Table 5-3. Peripheral Addresses and Selectors**

| Peripheral Device | Device ID | Transfer Size | Device FIFO Width | FIFO Physical Address |
|---|---|---|---|---|
| UART 0 Transmit | 0 | Programmable | 8 | 0x0 1110 0004 |
| UART 0 Receive | 1 | Programmable | 8 | 0x0 1110 0000 |
| DMA_REQ0 (GPIO[4]) | 2 | Programmable | Programmable | Programmable |
| DMA_REQ1 (GPIO[5]) | 3 | Programmable | Programmable | Programmable |
| AC97 Transmit | 4 | 4 | 16 | 0x0 1000 0008 |
| AC97 Receive | 5 | 4 | 16 | 0x0 1000 0008 |
| UART3 Transmit | 6 | Programmable | 8 | 0x0 1140 0004 |
| UART3 Receive | 7 | Programmable | 8 | 0x0 1140 0000 |
| USB Device Endpoint 0 Receive | 8 | 4 | 8 | 0x0 1020 0000 |
| USB Device Endpoint 0 Transmit | 9 | 4 | 8 | 0x0 1020 0004 |
| USB Device Endpoint 1 Transmit | 10 | 4 | 8 | 0x0 1020 0008 |
| USB Device Endpoint 2 Transmit | 11 | 4 | 8 | 0x0 1020 000C |
| USB Device Endpoint 3 Receive | 12 | 4 | 8 | 0x0 1020 0010 |
| USB Device Endpoint 4 Receive | 13 | 4 | 8 | 0x0 1020 0014 |
| DMA_REQ2 (GPIO[208]) | 14 | Programmable | Programmable | Programmable |
| DMA_REQ3 (GPIO[209]) | 15 | Programmable | Programmable | Programmable |

### 5.1.1 DMA Channel Mode Registers

Each DMA channel is controlled by a mode register. The current value of the register can be read from the **dma_moderead** register but can not be set to an arbitrary value in a single operation. Instead, the configuration register is controlled by two registers: **dma_modeset** and **dma_modeclr**.

- The **dma_modeset** register *sets* bits in the channel mode register when the corresponding bit is written as a one. (Bits written as zero do not affect the corresponding mode bit.)
- The **dma_modeclr** register *clears* bits in the channel mode register when the corresponding bit is written as a one. (Bits written as zero do not affect the corresponding mode bit.)

The Au1500 processor has been designed to simplify the DMA control process by removing the need for a semaphore to control access to the registers. This is because there is no need to read, modify, write, as there are separate registers for setting and clearing a bit. In this way a function can freely manipulate the DMA channels associated with that function.

An arbitrary value may be written to a field within the register with the following sequence:

```
dma_modeset = new_value & field_mask;
dma_modeclr = ~new_value & field_mask;
```

The Transfer Size and Device Width fields must be programmed to match the FIFO of the peripheral chosen with the DID field according to Table 5-3.

For the UART FIFOs the transfer size is programmable. It is the programmers responsibility to insure that the Transfer Size matches the trigger depth set in the UART FIFO control register. See Section 7.5 "UART Interfaces" on page 140 for more information.

For external DMA using GPIO signals as requests (DMA_REQ*n*), the system designer must ensure that the Transfer Size and Device Width match the external FIFO and that memory is packed accordingly.

Note that before issuing a DMA request, the receiving or transmitting FIFO must be prepared to complete a full transaction (4 or 8 datums, depending on **dma_mode**[TS]) without risking overflow or underflow.

**dma_moderead - Read DMA Mode Register** **Offset = 0x0000**
**dma_modeset - Set DMA Mode Register** **Offset = 0x0000**
**dma_modeclr - Clear DMA Mode Register** **Offset = 0x0004**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | DAH | | | | DID | | | | ▨ | ▨ | BE | DR | TS | DW | | NC | IE | H | G | AB | D1 | BE1 | D0 | BE0 |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:24 | — | Reserved, should be cleared. | R | 0 |
| 23:20 | DAH | Device Address High. Provides the most significant 4 bits of physical device address. | R/W | 0 |
| 19:16 | DID | Device ID. Identifies the peripheral device to act as source or destination; see Table 5-3 "Peripheral Addresses and Selectors" on page 88. | R/W | 0 |
| 15:14 | — | Reserved, should be cleared. | R | 0 |
| 13 | BE | Big Endian.<br>0 Little Endian byte order<br>1 Big Endian byte order | R/W | 0 |
| 12 | DR | Device Read.<br>0 Data is transferred from memory to device.<br>1 Data is transferred from device to memory. | R/W | 0 |
| 11 | TS | Transfer Size<br>Number of datums transferred per transaction. The device width is programmed in DW.<br>0 4 datums. (Valid for all device widths.)<br>1 8 datums. (Valid for 8-bit and 16-bit device widths only.) | R/W | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 10:9 | DW | Device FIFO Width.<br>00 Transfer width is 8 bits.<br>01 Transfer width is 16 bits.<br>10 Transfer width is 32 bits. (Not valid for TS=1.)<br>11 Reserved | RW | 0 |
| 8 | NC | Not Coherent.<br>0 Memory reads and writes are marked coherent on the SBUS.<br>1 Memory reads and writes are marked non coherent on the SBUS.<br>For more information on coherency see Section 2.8.2 "SBUS Coherency Model" on page 42 for more information on coherency. | R/W | 0 |
| 7 | IE | Interrupt Enable.<br>0 No interrupts will be generated.<br>1 Interrupts are generated when either D1 or D0 is set. | R/W | 0 |
| 6 | H | Channel Halted.<br>0 Channel is active.<br>1 Channel is halted.<br>This bit should be used to determine if the channel has been halted after the G bit has been cleared. | R | 0 |
| 5 | G | Channel Go. Setting the channel go bit enables the channel. When this bit is cleared the DMA controller does not arbitrate for this channel regardless of the state of the buffer enable bits. When the go bit is cleared by software the channel configuration should not be modified until the DMA controller sets the halt bit to indicate that the channel is inactive and therefore safe to be reconfigured. | R/W | 0 |
| 4 | AB | Active Buffer.<br>0 Buffer 0 is currently in use by the DMA.<br>1 Buffer 1 is currently in use by the DMA.<br>This field can be read to determine what buffer the DMA will service next if there is not a DMA transaction in progress. During a DMA transaction this bit will reflect the buffer currently being used.<br>Note that the DMA alternates between the two buffers. In other words, it is not possible to only use one buffer, DMA transactions must be switched between each buffer. | R | 0 |
| 3 | D1 | Done 1. The D1 bit is set by the DMA controller to indicate that a transfer to or from buffer 1 is complete. This bit must be cleared by the processor. | R/W | 0 |
| 2 | BE1 | The BE1 bit enables buffer 1. This bit is set by the processor and cleared by the DMA controller when the buffer has been filled or emptied. This bit may be cleared by the processor only when the H bit is set. | R/W | 0 |
| 1 | D0 | Done 0. The D0 bit is set by the DMA controller to indicate that a transfer to or from buffer 0 is complete. This bit must be cleared by the processor. | R/W | 0 |
| 0 | BE0 | The BE0 bit enables buffer 0. This bit is set by the processor and cleared by the DMA controller when the buffer has been filled or emptied. This bit may be cleared by the processor only when the H bit is set. | R/W | 0 |

### 5.1.2 DMA Peripheral Device Address

The peripheral device address register contains a pointer to the peripheral FIFO to be used as a source or destination. Software is responsible for matching the peripheral address to the correct value of the Device ID (DID) field in the mode register. The correspondence between FIFO address and DID values is shown in Table 5-3 "Peripheral Addresses and Selectors" on page 88. The physical address of the FIFO must be used.

The DAH field from the **dma_mode** register is used as the most significant four bits of the FIFO physical address.

**dma_peraddr - DMA Peripheral Address Register** **Offset = 0x0008**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | ADDR | Peripheral FIFO address | R/W | 0 |

### 5.1.3 DMA Buffer Starting Address Registers

Each DMA channel has two buffers, labeled buffer0 and buffer1. The starting address of each buffer should be written to the **dma_buf0addr** and **dma_buf0addr** registers respectively. The starting address must be cache line (32 bytes) aligned.

The 4 most significant bits of the buffer address are held in the *BAH* field of the **dma_buf0size** and **dma_buf1size** registers.

The starting address must explicitly be written before each DMA transaction, even if the address has not changed from the previous, as **dma_bufnaddr** will change during the DMA transaction.

Note that the DMA alternates between the two buffers. In other words, it is not possible to use only one buffer—DMA transactions must be switched between each buffer. The AB bit in the **dma_mode** register can be used to determine the active buffer.

**dma_buf0addr - Buffer0 Starting Address** **Offset = 0x000C**
**dma_buf1addr - Buffer1 Starting Address** **Offset = 0x0014**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | ADDR | Lower 32 bits of the physical starting address of the DMA memory buffer. | R/W | 0 |

### 5.1.4 DMA Channel Buffer Size Registers

The size of each DMA buffer is given by the **dma_buf0size** and **dma_buf1size** registers. The buffer size registers also contributes the most significant four bits of the buffer physical address.

This register should be programmed with the block size of the buffer in datums. While a DMA transaction is in progress, it indicates the number of datums remaining in the transfer.

Note that the DMA alternates between the two buffers. In other words, it is not possible to only use one buffer, DMA transactions must be switched between each buffer. The active-buffer bit **dma_mode**[AB] can be used to determine which buffer is active at a given time.

**dma_buf0size - Buffer 0 Size** **Offset = 0x0010**
**dma_buf1size - Buffer 1 Size** **Offset = 0x0018**

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 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 |
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | BAH | | | | SIZE | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:20 | — | Reserved, should be cleared. | R/W | 0 |
| 19:16 | BAH | Buffer Address High. Provides the 4 most significant bits of the buffer address. | R/W | 0 |
| 15:0 | SIZE | Buffer Size and Count Remaining. Indicates the number of datums remaining in the current transfer. | R/W | 0 |

## 5.2 Using GPIO as External DMA Requests (DMA_REQ*n*)

To use GPIO[4], GPIO[5], GPIO[208] or GPIO[209] as an external DMA request (DMA_REQ*n*) follow these steps:

1) For GPIO[4] and GPIO[5]: Write the **sys_pininputen** to enable the GPIO to be used as an input. See Section 8.3 "Primary General Purpose I/O and Pin Functionality" on page 168 for more information.

   For GPIO[208] and GPIO[209]: Be sure that the secondary GPIO block is enabled in **gpio2_enable**. Also, set the corresponding interrupt-enable bit in **gpio2_inten**. See Section 7.6 "Secondary General Purpose I/O" on page 149 for more information.

2) For GPIO[4] and GPIO[5]: TRI-STATE the GPIO to make it an input through the **sys_triout** register. See Section 8.3 "Primary General Purpose I/O and Pin Functionality" on page 168 for more information.

   For GPIO[208] and GPIO[209]: Configure the signals as inputs in **gpio2_dir**. See Section 7.6 "Secondary General Purpose I/O" on page 149 for more information.

3) Set the **dma_peraddr** register to point to the external device data port. The Static Bus Controller must be configured correctly to recognize this address.

4) Program the mode register to match the direction of transfer and peripheral attributes.

The DMA_REQ*n* signal must be driven high to request a DMA transfer and must remain high until the DMA transaction is started. Once started, the DMA transaction continues until finished regardless of the DMA request signal state. A DMA transaction refers to a DMA transfer of one transfer size as defined in the DMA mode register (**dma_mode**[TS]).

DMA_REQ*n* should be tied to the external FIFO threshold indicator. In this way the DMA_REQ*n* signal asserts when the FIFO threshold is reached and remains asserted until the FIFO fills or empties past the threshold (after the DMA transaction starts). DMA_REQ*n* should then negate after the FIFO threshold is met from the opposite direction (approaches full for a transmit or approaches empty for a read). The threshold should be designed such that a complete DMA transaction (4 or 8 datums) can occur without risking overflow or underflow.

## 5.3 Programming Considerations

The following pseudo code is for setting up and servicing a DMA channel:

```
SetupDMA() {

Make sure interrupts are enabled globally (CP0 Reg 12, bit 0)

Enable interrupt controller for this DMA channel, high/level

Program the DMA controller {
    dma_modeclr = 0xFFFFFFFF
    dma_buf0size = buffer size (up to 65535)
    dma_buf1size = buffer size (up to 65535)
    dma_peraddr = Address of peripheral FIFO
    dma_buf0addr = physical base address of buffer
    dma_buf1addr = physical base address of next buffer

    Write the dma_modeset register {
        Enable interrupt
        Enable both buffers
        Set endianness
        Set data width, 8-bit, 16-bit, 32-bit
        Set Device ID
        Set transfer size, 4-datum burst, 8-datum burst
        set coherency = 0 (memory is coherent)
        set go = 1
    }
}

InterruptHandler() {
Note: This routine assumes it is called from context save/
    restore routine at 0x80000200.

Check for hardware interrupt from interrupt controller 0,
    request 0: (CP0 Reg 13, bit 10) = 1

Read interrupt controller 0 ic_req0int (interrupt status)
    and check if source is from this DMA channel, bits[13:6]

if it is this DMA channel {

    Check dma_moderead to see which buffer is done: D0 or D1
    if (D0 is set) {
        write dma_modeclr bit D0 = 1 to clear interrupt
        if there is another buffer to send {
            dma_buf0addr = physical base address of buffer
            dma_buf0size = buffer size (up to 65535)
            write dma_modeset bit BE0 = 1 to enable buffer
            }
        }
    }
    if (D1 is set) {
        write dma_modeclr bit D1 = 1 to clear interrupt
        if there is another buffer to send {
            dma_buf1addr = physical base address of buffer
            dma_buf1size = buffer size (up to 65535)
            write dma_modeset bit BE1 = 1 to enable buffer
            }
        }
    }
}
Issue sync
}
```

# 6 Interrupt Controller

There are two interrupt controllers in the Au1500 processor. Each interrupt controller supports 32 interrupt sources. Interrupts can generate a signal to bring the Au1500 processor out of an IDLE0 or IDLE1 state and generate a CPU interrupt.

Each interrupt controller has two outputs referred to as requests 0 and 1. Each of these outputs are connected to the CPU core. See Section 2.5 "Exceptions" on page 27 for a complete Au1500 processor interrupt architecture discussion. Table 6-1 shows the interrupt controller connections to the CPU.

**Table 6-1. Interrupt Controller Connections to the CPU**

| Interrupt Source | CP0 Cause Register Bit |
|---|---|
| Interrupt Controller 0: | |
| Request 0 | 10 |
| Request 1 | 11 |
| Interrupt Controller 1: | |
| Request 0 | 12 |
| Request 1 | 13 |

## 6.1 Interrupt Controller Sources

Table 6-2 shows the mapping of interrupt sources for Interrupt Controller 0 and 1.

Care should be taken to select the correct interrupt type (level or edge triggered) so that an interrupt is not missed. In general, level interrupts are chosen when multiple sources from a single peripheral might cause an interrupt. In this way the programmer will not miss a subsequent interrupt from a particular source while servicing the previous one.

Edge triggered interrupts can be used when there is only a single source for an interrupt. Edge triggered interrupts must be used when an interrupt is caused by an internal event and not tied to a register bit where it is latched and held until cleared by the programmer.

Details about the interrupt sources can be found in the respective peripheral sections.

**Table 6-2. Interrupt Sources**

| Controller | Interrupt Number | Source | Type |
|---|---|---|---|
| 0 | 0 | UART0 | High Level |
| 0 | 1 | PCI_INTA# | Low Level |
| 0 | 2 | PCI_INTB# | Low Level |
| 0 | 3 | UART3 | High Level |
| 0 | 4 | PCI_INTC# | Low Level |
| 0 | 5 | PCI_INTD# | Low Level |
| 0 | 6 | DMA0 | High Level |
| 0 | 7 | DMA1 | High Level |
| 0 | 8 | DMA2 | High Level |
| 0 | 9 | DMA3 | High Level |

Table 6-2. Interrupt Sources (Continued)

| Controller | Interrupt Number | Source | Type |
|---|---|---|---|
| 0 | 10 | DMA4 | High Level |
| 0 | 11 | DMA5 | High Level |
| 0 | 12 | DMA6 | High Level |
| 0 | 13 | DMA7 | High Level |
| 0 | 14 | TOY (tick) | Rising Edge |
| 0 | 15 | TOY Match 0 | Rising Edge |
| 0 | 16 | TOY Match 1 | Rising Edge |
| 0 | 17 | TOY Match 2 | Rising Edge |
| 0 | 18 | RTC (tick) | Rising Edge |
| 0 | 19 | RTC Match 0 | Rising Edge |
| 0 | 20 | RTC Match 1 | Rising Edge |
| 0 | 21 | RTC Match 2 | Rising Edge |
| 0 | 22 | PCI Error Interrupt | Internal |
| 0 | 23 | Reserved | N/A |
| 0 | 24 | USB Device Interrupt Request | High Level |
| 0 | 25 | USB Device Suspend Interrupt | Rising/Falling edge |
| 0 | 26 | USB Host | Low Level |
| 0 | 27 | AC97 ACSYNC | Rising Edge |
| 0 | 28 | MAC 0 DMA Done | High Level |
| 0 | 29 | MAC 1 DMA Done | High Level |
| 0 | 30 | Reserved | N/A |
| 0 | 31 | AC97 Command Done | Rising Edge |
| 1 | n = 0..15 | GPIO[n] | System Dependent |
| 1 | 16 | GPIO[200] | System Dependent |
| 1 | 17 | GPIO[201] | System Dependent |
| 1 | 18 | GPIO[202] | System Dependent |
| 1 | 19 | GPIO[203] | System Dependent |
| 1 | 20 | GPIO[20] | System Dependent |
| 1 | 21 | GPIO[204] | System Dependent |
| 1 | 22 | GPIO[205] | System Dependent |
| 1 | 23 | GPIO[23] | System Dependent |
| 1 | 24 | GPIO[24] | System Dependent |
| 1 | 25 | GPIO[25] | System Dependent |
| 1 | 26 | GPIO[26] | System Dependent |
| 1 | 27 | GPIO[27] | System Dependent |
| 1 | 28 | GPIO[28] | System Dependent |
| 1 | 29 | GPIO[206] | System Dependent |
| 1 | 30 | GPIO[207] | System Dependent |
| 1 | 31 | Logical OR of GPIO[215:208] (See Section 7.6.2.4 "Interrupt Enable Register" on page 151.) | System Dependent |

Figure 6-1 shows the Interrupt Controller logic diagram. Where applicable, the names in the diagram correspond to bit *n* in the relative control register.

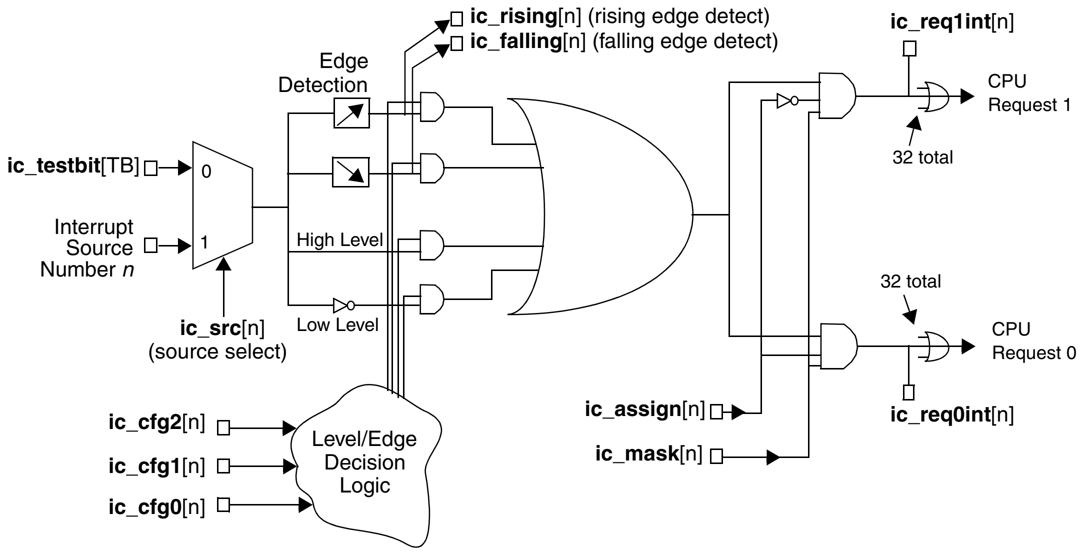


**Figure 6-1. Interrupt Controller Logic**

## 6.2 Register Definitions

The design of the software interface to the interrupt controller is based on the premise that software tasks should be able to access the value and control of an individual port without blocking other tasks from accessing ports of interest to them. This interrupt controller design removes the need to arbitrate via a semaphore access to the interrupt controller registers. The result is faster and simpler interrupt controller accessing.

Table 6-3 shows the base address for each interrupt controller.

**Table 6-3. Interrupt Controller Base Addresses**

| Name | Physical Base Address | KSEG1 Base Address |
|---|---|---|
| ic0_base | 0x0 1040 0000 | 0xB040 0000 |
| ic1_base | 0x0 1180 0000 | 0xB180 0000 |

Each interrupt controller has an identical set of registers that controls its set of 32 interrupts. Table 6-4 shows the interrupt controller registers and their associated offsets. Certain offsets are shared but address different internal registers depending on whether the access is a read or a write. The register description details the functionality of the register. Bit *n* of a particular register is associated with interrupt *n* of the corresponding controller.

**Table 6-4. Interrupt Controller Registers**

| Offset (Note 1) | Register Name | Type | Register Description | Default |
|---|---|---|---|---|
| 0x0040 | ic_cfg0rd | R | Configuration 0 register | UNPRED |
| 0x0040 | ic_cfg0set | W | Configuration 1 register | |
| 0x0044 | ic_cfg0clr | W | Configuration 2 register | |
| 0x0048 | ic_cfg1rd | R | The combined field consisting of **ic_cfg2**[*n*], **ic_cfg1**[*n*], and **ic_cfg0**[*n*] specifies the trigger characteristics for interrupt *n* as shown in Table 6-5 "Interrupt Configuration Register Function" on page 100. | UNPRED |
| 0x0048 | ic_cfg1set | W | | |
| 0x004C | ic_cfg1clr | W | | |
| 0x0050 | ic_cfg2rd | R | | UNPRED |
| 0x0050 | ic_cfg2set | W | | |
| 0x0054 | ic_cfg2clr | W | | |
| 0x0054 | ic_req0int | R | Shows active interrupts on request 0.<br>Used by host software to determine the source of the interrupt. | 0x0000 0000 |
| 0x0058 | ic_srcrd | R | Selects the source of the interrupt between a test bit and the designated source.<br>0 The test bit (**ic_testbit**[TB]) is used as interrupt source.<br>1 Peripheral interrupt (controller 0) or GPIO signal (controller 1) is used for interrupt source. | UNPRED |
| 0x0058 | ic_srcset | W | | |
| 0x005C | ic_srcclr | W | | |
| 0x005C | ic_req1int | R | Shows active interrupts on request 1.<br>Used by host software to determine the source of the interrupt. | 0x0000 0000 |
| 0x0060 | ic_assignrd | R | Assigns the interrupt to one of the CPU requests.<br>0 Assign interrupt to request 1.<br>1 Assign interrupt to request 0. | UNPRED |
| 0x0060 | ic_assignset | W | | |
| 0x0064 | ic_assignclr | W | | |
| 0x0068 | ic_wakerd | R | Controls whether the interrupt can cause a wakeup from IDLE0 or IDLE1.<br>0 No wakeup from Idle<br>1 Interrupt will cause wakeup from Idle.<br>The associated interrupt must still be enabled to wake from Idle. | 0x0000 0000 |
| 0x0068 | ic_wakeset | W | | |
| 0x006C | ic_wakeclr | W | | |
| 0x0070 | ic_maskrd | R | Interrupt enable.<br>0 Disable the interrupt.<br>1 Enable the interrupt. | 0x0000 0000 |
| 0x0070 | ic_maskset | W | | |
| 0x0074 | ic_maskclr | W | | |
| 0x0078 | ic_risingrd | R | Designates active rising edge interrupts. If an interrupt is generated off of a rising edge, the associated rising edge detection bit must be cleared after detection. | UNPRED |
| 0x0078 | ic_risingclr | W | | |
| 0x007C | ic_fallingrd | R | Designates active falling edge interrupts. If an interrupt is generated off of a falling edge, the associated falling edge detection bit must be cleared after detection. | UNPRED |
| 0x007C | ic_fallingclr | W | | |
| 0x0080 | ic_testbit | R/W | This is a single bit register that is mapped to all the source select inputs for testing purposes. | UNPRED |

Note 1. See Table 6-3 on page 97 for base address.

### 6.2.1 Interrupt Controller Registers

Each register (except the test-bit register) is 32 bits wide with bit *n* in each register affecting interrupt *n* in the corresponding controller.

***rd**
***set**
***clr**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | FUNC[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | FUNC[n] | The function of each register is given in Table 6-4 "Interrupt Controller Registers" on page 98. FUNC[n] controls the functionality of interrupt *n* in the corresponding controller. | *rd - read only<br>*set - write only<br>*clr - write only<br>See the following explanation. | See Table 6-4 "Interrupt Controller Registers" on page 98. |

The test-bit register contains the test bit that can be used as a test source for each interrupt. Figure 6-1 "Interrupt Controller Logic" on page 97 shows how the test bit connects to the interrupt source-select logic.

**ic_testbit** **Offset = 0x0080**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | TB |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:1 | — | Reserved, should be cleared. | R/W | UNPRED |
| 0 | TB | Test bit value used as an alternate interrupt source. | R/W | UNPRED |

Certain interrupt controller registers have the same offset but offer different functionality. This is by design. Care should be taken when programming the registers because a read from one location may reference something different from a write to the same location.

Registers ending in *rd, *set and *clr have the following functionality:

- *rd registers are read only registers will read back the current value of the register.
- *set registers are write only registers and will set to 1 all bits that are written 1. Writing a value of 0 will have no impact on the corresponding bit.
- *clr registers are write only registers and will clear to zero all bits that are written 1. Writing a value of 0 will have no impact on the corresponding bit.

The three configuration registers have a special functionality in that the value associated with ic_cfg2[*n*], ic_cfg1[*n*], ic_cfg0[*n*] uniquely control interrupt *n*'s functionality as shown in Table 6-5.

**Table 6-5. Interrupt Configuration Register Function**

| ic_cfg2[n] | ic_cfg1[n] | ic_cfg0[n] | Function |
|---|---|---|---|
| 0 | 0 | 0 | Interrupts Disabled |
| 0 | 0 | 1 | Rising Edge Enabled |
| 0 | 1 | 0 | Falling Edge Enabled |
| 0 | 1 | 1 | Rising and Falling Edge Enabled |
| 1 | 0 | 0 | Interrupts Disabled |
| 1 | 0 | 1 | High Level Enabled |
| 1 | 1 | 0 | Low Level Enabled |
| 1 | 1 | 1 | Both Levels and Both Edges Enabled |

## 6.3 Hardware Considerations

When using a GPIO or peripheral as an interrupt source, it is important that the associated pin functionality has been enabled in the **sys_pinfunc** register. In addition when using a GPIO, the GPIO must first be enabled as an input. See Section 8.3 "Primary General Purpose I/O and Pin Functionality" on page 168 for more information.

## 6.4 Programming Considerations

The Au1500 has been designed to simplify the interrupt control process by removing the need for a semaphore to control access to the registers. This is because there is no need to read, modify, write, as there are separate registers for setting and clearing a bit. In this way a function can freely manipulate the interrupts associated with that function.

If using edge triggered interrupts, it is important to clear the associated edge detection bit or future interrupts will not be seen.

Programming an interrupt controller can be broken into the following steps (the **set**, **clr**, and **rd** portion of the register name has been omitted):

1) Identify the interrupt number, *n,* with the associated peripheral or GPIO.
2) Use **ic_src**[*n*] to assign the interrupt to the associated peripheral/GPIO (or the test bit can be used if testing the interrupt).
3) Set the **ic_cfg2**[*n*], **ic_cfg1**[*n*] and **ic_cfg0**[*n*] bits to the correct configuration for the corresponding interrupt (edge, level, polarity).
4) Assign the interrupt to a CPU request using **ic_assign**[*n*].
5) Use **ic_wake**[*n*] to assign the interrupt to wake the processor from IDLE if necessary or clear this register bit to keep the interrupt from waking the processor from IDLE.
6) If the interrupt is an edge triggered interrupt, clear the edge detect register (**ic_risingclr** or **ic_fallingclr**) before enabling.
7) Finally, enable the interrupt through **ic_mask**[*n*].

When taking an interrupt the following steps should be taken:

1) Read **ic_req0int** and **ic_req1int** to determine the interrupt number *n*.
2) Use **ic_fallingrd** and **ic_risingrd** to determine if the interrupt was edge triggered. If the interrupt is edge triggered, use **ic_fallingclr**[*n*] or **ic_risingclr**[*n*] to clear the edge detection circuitry.
3) If the interrupt is to be disabled write **ic_maskclr**[*n*].
4) Service the interrupt.

# 7 Peripheral Devices

This section provides descriptions of the peripheral devices of the Au1500 processor.  This includes an AC97 controller, USB Host and Device interfaces, two 10/100 Ethernet MACs and two UARTs.

Each peripheral contains an enable register. All other registers within each peripheral's register block should not be accessed until the enable register is written the correct sequence to bring the peripheral out of reset. Accessing the peripheral register block before a peripheral is enabled will result in undefined results.

## 7.1 AC97 Controller

The Au1500 processor contains an AC97 controller which incorporates an AC-link capable of bridging to an AC97 compliant codec.

### 7.1.1 AC97 Registers

The AC97 controller is controlled by a register block whose physical base address is shown in Table 7-1. The register block consists of 5 registers as shown in Table 7-2.

**Table 7-1. AC97 Base Address**

| Name | Physical Base Address | KSEG1 Base Address |
|---|---|---|
| ac97_base | 0x0 1000 0000 | 0xB000 0000 |

**Table 7-2. AC97 Registers**

| Offset (Note 1) | Register Name | Description |
|---|---|---|
| 0x0000 | ac97_config | AC-link Configuration |
| 0x0004 | ac97_status | Controller Status |
| 0x0008 | ac97_data | TX/RX Data |
| 0x000C | ac97_cmmd | Codec Command |
| 0x000C | ac97_cmmdresp | Codec Command Response |
| 0x0010 | ac97_enable | AC97 Block Control |

Note 1. See Table 7-1 for base address.

#### 7.1.1.1 AC-Link Configuration Register

The configuration register contains bits necessary to configure and reset the AC-link and codec.

**ac97_config - AC-link Configuration** **Offset = 0x0000**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | RC | | | | | | | | | | XS | | | | | | | | | | SG | SN | RS |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:23 | — | Reserved, should be cleared. | R | 0 |
| 22:13 | RC | Receive Slots. The bits set in RC will control what data from valid slots are put into the input buffer.<br>The corresponding valid bits in the AC97 tag (slot 0 of SDATA_IN) must be marked valid for the incoming PCM data to be put in the input buffer.<br>Slot 3 is mapped to bit 13, slot 4 to 14 and so on.<br>Note: The programmer must ensure that the codec is configured such that there will be valid data in the slots corresponding to what receive slots are enabled. | R/W | 0 |
| 12:3 | XS | Transmit Slots. The bits making up XMIT_SLOTS map to the valid bits in the AC97 tag (slot 0 on SDATA_OUT) and indicate which outgoing slots have valid PCM data. Bit 3 maps to slot 3, bit 4 to slot 4 and so on. Setting the corresponding bit indicates to the codec that valid data will be in the respective slot. The number of valid bits will designate how many words will be pulled out of the FIFO per audio frame. | R/W | 0 |
| 2 | SG | SYNC Gate. Setting this bit to 1 will gate the clock from being driven on SYNC. This allows the SN bit to control the value on SYNC. In combination with SN, the SG bit can be used to initiate a warm reset. | R/W | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 1 | SN | SYNC Control. This bit controls the value of the SYNC signal when SG (SYNC gate) is set. In combination with SG, the SN bit can be used to initiate a warm reset. | R/W | 0 |
| 0 | RS | AC-link Reset (ACRST#) Control. To initiate a cold AC97 reset, set the RS bit to drive the ACRST# signal low. After satisfying the ACRST# low time for the codec, clear this bit to negate ACRST#. | R/W | 0 |

### 7.1.1.2 AC97 Controller Status

The AC97 Controller Status register contains status bits for the transmit and receive FIFOs, command status and the codec.

**ac97_status - Controller Status** **Offset = 0x0004**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | XU | XO | RU | RO | RD | CP | | TE | TF | | RE | RF |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:12 | — | These bits are reserved. | R | UNPRED |
| 11 | XU | Transmit Underflow. When set, this bit indicates that the transmit FIFO has experienced an underflow. This sticky bit is cleared when written (0 *or* 1). | R | 0 |
| 10 | XO | Transmit Overflow. When set, this bit indicates that the transmit FIFO has experienced an overflow. This sticky bit is cleared when written (0 *or* 1). | R | 0 |
| 9 | RU | Receive Underflow. When set, this bit indicates that the receive FIFO has experienced an underflow. This sticky bit is cleared when written (0 *or* 1). | R | 0 |
| 8 | RO | Receive Overflow. When set, this bit indicates that the receive FIFO has experienced an overflow. This sticky bit is cleared when written (0 *or* 1). | R | 0 |
| 7 | RD | Ready. This bit is mapped from the CODEC_READY bit in the SDATA_IN tag word. It indicates that the codec is properly booted and ready for normal operation. | R | 0 |
| 6 | CP | Command Pending. This bit indicates that there is a command pending on the AC-link. A write to the Codec Command register will cause this bit to be set until the command is completed. The command is completed for a write when the data has been written out on slot 2. The command is completed for a read request when the status data has been read from the corresponding read request. (This means that a read request could be pending for more than 1 cycle depending on the latency of the read.)<br>The command register should not be written until the CP bit is clear.<br>An interrupt can be enabled to indicate when a command is done. The source of this interrupt is an internal pulse so either rising edge or falling edge interrupt should be used for this interrupt. | R | 0 |
| 5 | — | Reserved. | R | UNPRED |
| 4 | TE | Transmit Empty. When set this bit indicates that the transmit FIFO is empty. | R | 0 |
| 3 | TF | Transmit Full. When set this bit indicates the transmit FIFO is full. | R | 0 |
| 2 | — | Reserved. | R | UNPRED |
| 1 | RE | Receive Empty. When set this bit indicates that the receive FIFO is empty. | R | 0 |
| 0 | RF | Receive Full. When set this bit indicates that the receive FIFO is full. | R | 0 |

#### 7.1.1.3 TX/RX Data

The TX/RX Data register is the transmit FIFO’s input to the when written to and the receive FIFO’s output when read from. Each FIFO has twelve 16-bit entries. Care should be taken to monitor the status register to insure that there is room for data in the FIFO for a read or write transaction. This will be taken care of automatically if using DMA (see Section 7.1.3 "Programming Considerations" on page 106).

The number of bits set in XMIT_SLOTS will correspond with how many samples are pulled out of the FIFO and aligned in the respective slots. The number of bits set in RECV_SLOTS will correspond with the number of samples placed in the FIFO from the respective slots in SDATA_IN.

**ac97_data - TX/RX Data** **Offset = 0x0008**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | DATA_WORD[15:0] | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | — | Reserved, should be cleared. | R | 0 |
| 15:0 | DATA_WORD | Data Word. This is where data is written to or read from the FIFO. Each data word is 16 bits. | R/W | 0 |

#### 7.1.1.4 Codec Command

The Codec Command and Command Response registers share the same physical address.

The Codec Command register is used to send read and write commands to the codec. For write commands, the DATA field will be written to the register indicated by the INDEX field. For read commands, the DATA field should be written zero. The value read from the register indicated by INDEX will appear in the Codec Response register when the Command Pending bit in the status register (**ac97_status**[CP]) returns to 0.

The Codec Command register should only be written if **ac97_status**[CP] is 0.

**ac97_cmmd - Codec Command** **Offset = 0x000C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | DATA | | | | | | | | | | | | | | | | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | RW | INDEX | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | DATA | Data. These bits will be the actual 16-bit word written to the register indicated by INDEX if RW is a 0. If RW is set (indicating a read), these bits should be written 0. | W | 0 |
| 15:8 | — | Reserved, should be cleared. | W | 0 |
| 7 | RW | Read/Write# Bit (1=read, 0=write). This bit maps to the Read/Write# bit in the command address and designates whether the current operation will be a read or a write. | W | 0 |
| 6:0 | INDEX | Codec Register Index. These bits will address the specific register to be read or written to inside the codec. | W | 0 |

### 7.1.1.5 Codec Command Response

The Codec Command and Response registers share the same physical address.

After a read command is sent through the Codec Command register, the response can be read from the Codec Response register. The command response becomes valid when the Command Pending bit in the status register (**ac97_status**[CP]) is cleared; however, the response remains valid for only one AC97 frame length in duration (20.8 µs).

**ac97_cmmdresp - Codec Command Response** **Offset = 0x000C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | READ_DATA | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | — | Reserved, should be cleared. | R | 0 |
| 15:0 | READ_DATA | Read Data. These bits will be the response to the last read command sent to the codec. The read data becomes valid after the read command is completed (**ac97_status**[CP] = 0).<br>Note that READ_DATA remains valid for only *one* AC97 frame (20.8 µs) and should therefore be read immediately after **ac97_status**[CP] is cleared. | R | 0 |

### 7.1.1.6 AC97 Enable

The AC97 Enable register is used to enable and reset the entire AC97 Controller block. The routine for bringing the AC97 controller out of reset is as follows:

1) Set the CE bit to enable clocks while leaving the block disabled (D=1).
2) Clear the D bit to enable the peripheral.

**ac97_enable - AC97 Block Control** **Offset = 0x0010**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | D | CE |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:2 | — | Reserved, should be cleared. | W | 0 |
| 1 | D | AC97 Controller Disable. Setting this bit will reset the AC97 block. After enabling the clock with CE, this bit should be cleared for normal operation. | W | 1 |
| 0 | CE | Clock Enable. This bit should be set to enable the clock driving the AC97 Controller. It can be cleared to disable the clock for power considerations. | W | 0 |

### 7.1.2 Hardware Considerations

The AC-link consists of the signals listed in Table 7-3.

**Table 7-3. AC-Link Signals**

| Signal | Input/Output | Definition |
|---|---|---|
| ACSYNC | O | Fixed Rate Sample Sync. |
| ACBCLK | I | Serial Data Clock. |
| ACDO | O | TDM Output Stream. |
| ACDI | I | TDM Input Stream. |
| ACRST# | O | Codec Reset. |

For changing pin functionality please refer to the **sys_pinfunc** register in Section 8.3 "Primary General Purpose I/O and Pin Functionality" on page 168.

### 7.1.3 Programming Considerations

The AC97 block supports DMA transfers and interrupts. The use of the DMA or interrupts is program dependent and is not required to use the AC97 controller.

To use DMA for AC97 memory transfers the transmit and receive functions will each need a dedicated DMA channel. The DMA peripheral address register (**dma_peraddr**) in the DMA configuration registers will be set to point to the AC97 **ac97_data** register. The DMA mode register (**dma_mode**) will need to be set up with the correct Device ID (DID). The Device Read bit (DR) will depend on whether the channel is being used for receive or transmit. Typically the Device Width (DW) should be set to 16 bits and the transfer size bit (TS) should be cleared because the FIFO threshold indicators correspond to four-datum transfers. This assumes that the audio samples are aligned in memory on a 16-bit audio sample boundary. The DMA will automatically monitor the transmit and receive request bits and feed data accordingly.

An interrupt ("AC97 Command Done" in interrupt controller 0) can be enabled to indicate when a command is completed. The source of this interrupt is an internal pulse so either rising edge or falling edge interrupt should be used for this interrupt.

When the AC97 ACSYNC interrupt is enabled in interrupt controller 0, an interrupt will occur corresponding to the rising edge of the ACSYNC signal. Internally a pulse is generated from the rising edge of the ACSYNC signal and fed to the interrupt controller. Regardless of the edge enabled in the interrupt controller the interrupt will come after the rising edge of ACSYNC. Enabling a rising edge interrupt will interrupt the processor closest to the rising edge of ACSYNC.

The output FIFO for the AC-link is shared for all slots so care should be taken that there is a correspondence with the number of valid bits being set and the number of valid samples written to the transmit FIFO or aligned in memory for DMA or erroneous results will occur. It is the programmer's responsibility to ensure that the number of samples written to the FIFO corresponds with the number of valid slots enabled. Data will automatically be pulled out of the FIFO in the order of what slots are enabled. In other words if slots 3, 4, 6 and 9 are enabled, the programmer should write samples corresponding to data for slots 3, 4, 6, and 9, in that order, to the FIFO.

To insure against underflow at least *x* words should be written per audio frame where *x* is the number of slots enabled. This is a mean rate over time and the actual write rate may differ depending on latency requirements, DMA buffer size, and the number of slots enabled.

Care should be taken that there is a correspondence with the number of valid bits that have been set and the number of valid samples read from the receive FIFO or erroneous results will occur.

The input FIFO for the AC-link is shared for all slots so care should be taken that there is a correspondence with the number of valid bits that are set and the number of samples read from the receive FIFO or erroneous results will occur. It is the programmer's responsibility to ensure that the number of samples read from the FIFO corresponds with the number of valid slots enabled. Data will automatically be put in the FIFO in the order of what slots are enabled. In other words if slots 3 and 4, are enabled, the programmer should read samples corresponding to data for slots 3 and 4, in that order, from the FIFO.

To insure against overflow at least *x* words should be read per audio frame where *x* is the number of slots enabled. This is a mean rate over time and the actual read rate may differ depending on latency requirements, DMA buffer size, and the number of slots enabled.

## 7.2 USB Host Controller

The Au1500 processor USB host controller conforms to the Open HCI interface specification, revision 1.0, and is USB 1.1 compliant. Two root hub ports, port 0 and port 1, are provided. The base of the Open HCI register block is shown in Table 7-4.

**Table 7-4. USB Host Base Address**

| Name | Physical Base Address | KSEG1 Base Address |
|---|---|---|
| usbh_base | 0x0 1010 0000 | 0xB010 0000 |

Only 32-bit accesses are allowed to the Open HCI registers.

All interrupts as described in the Open HCI specification are supported. These interrupts are combined when brought to the interrupt controller into one active-low interrupt (negative-edge triggered does not work). The interrupt controller should be programmed to reflect this by setting the USB Host interrupt to low level. See Section 6.0 "Interrupt Controller" on page 95 for details.

### 7.2.1 USB Host Enable Register

This register is not part of the OpenHCI registers; however, it shares the same base address. The **usbh_enable** register controls the reset and clocks to the USB Host controller. When initializing the USB Host controller the programmer should first enable clocks, then enable the module (remove from reset), then wait for the RD bit to be set before performing Open-HCI initialization.

The correct routine for bringing the USB Host Controller out of reset is as follows:

1) Set the CE bit to enable clocks.

2) Set the E bit to enable the peripheral (at this time the C and BE bits should be configured appropriately for the system).

3) Clear the HCFS bit in the HcControl register to reset the OHCI state.

4) Wait for the RD bit to be set before issuing any commands to the OpenHCI controller.

To put the USB Host Controller into reset the following steps should be taken:

1) Set the HCFS bit in the HcControl register.

2) Clear the E and CE bits.

**usbh_enable** **Offset = 0x7FFFC**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | RD | CE | E | C | BE |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:5 | — | Reserved, must be cleared. | R/W | 0 |
| 4 | RD | Reset Done. Wait for this bit to be set before issuing any commands to the OpenHCI controller.<br>Note: When writing to the **usbh_enable** register, this bit position must be 0. | R | 0 |
| 3 | CE | Clock Enable. When this bit is set, clocks are enabled to the USB Host controller. | R/W | 0 |
| 2 | E | Enable. This bit enables the USB Host controller. When this bit is clear the controller is held in reset. | R/W | 0 |
| 1 | C | Coherent. If this bit is set memory accesses by the controller will be marked coherent on the SBUS. When this bit is clear memory accesses by the USB Host controller are non coherent.<br>For more information on coherency see Section 2.8.2 "SBUS Coherency Model" on page 42 for more information on coherency. | R/W | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 0 | BE | Big Endian. When this bit is set the controller interprets data buffers in Big Endian byte order. When this bit is clear the controller interprets data buffers in Little Endian byte order.<br>Setting the BE bit does not swap the control structures defined in the OHCI specification. Endpoint descriptors (section 4.2), transfer descriptors (section 4.3), and the HCCA (host controller communications area, section 4.4) should always be written as words to ensure proper operation. | R/W | 0 |

### 7.2.2 USB Host Signals

Table 7-5 shows the signals associated with the two USB host root hub ports. The USB root hub port pins have USB 1.1 compliant drivers with the addition of the external circuitry noted in the signal description.

**Table 7-5. USB Host Signals**

| Signal | Input/Output | Description |
|---|---|---|
| USBH0P | I/O | Positive signal of differential USB host port 0 driver<br>Requires an external 15 kohm pull-down resistor and ESD protection diode (transient voltage suppressor) to be USB 1.1 compliant.<br>Termination Note: Requires an external 27-Ω resistor placed in series within 0.5 inches of the part.<br>Muxed with USBDP which controls the pin out of reset. |
| USBH0M | I/O | Negative signal of differential USB host port 0 driver<br>Requires an external 15 kohm pull-down resistor and ESD protection diode (transient voltage suppressor) to be USB 1.1 compliant.<br>Termination Note: Requires an external 27 ohm resistor placed in series within 0.5 inches of the part.<br>Muxed with USBDM which controls the pin out of reset. |
| USBH1P | I/O | Positive signal of differential USB host port 1 driver.<br>Requires an external 15 kohm pull-down resistor and ESD protection diode (transient voltage suppressor) to be USB 1.1 compliant.<br>Termination Note: Requires an external 27 ohm resistor placed in series within 0.5 inches of the part. |
| USBH1M | I/O | Negative signal of differential USB host port 1 driver.<br>Requires an external 15 kohm pull-down resistor and ESD protection diode (transient voltage suppressor) to be USB 1.1 compliant.<br>Termination Note: Requires an external 27 ohm resistor placed in series within 0.5 inches of the part. |

For changing pin functionality please refer to the **sys_pinfunc** register in Section 8.3 "Primary General Purpose I/O and Pin Functionality" on page 168.

## 7.3 USB Device Controller

The Au1500 processor USB device controller supports endpoints 0, 1, 2, 3, and 4. Endpoint 0 is always configured as a bidirectional control endpoint. Endpoints 1 and 2 are always IN endpoints and endpoints 3 and 4 are always OUT endpoints.

IN is from device to host. From the device perspective these endpoints are written, so the associated registers are tagged with write or wr.

OUT is from host to device. From the device perspective these endpoints are read, so the associated registers are tagged with read or rd.

### 7.3.1 USB Device Controller Registers

The USB device registers are located off of the base address shown in Table 7-6. Table 7-7 shows the offsets of each register from the register base

**Table 7-6. USB Device Base Address**

| Name | Physical Base Address | KSEG1 Base Address |
|---|---|---|
| usbd_base | 0x0 1020 0000 | 0xB020 0000 |

**Table 7-7. USB Device Register Block**

| Offset (Note 1) | Name | Function |
|---|---|---|
| 0x0000 | usbd_ep0rd | Read from endpoint 0 |
| 0x0004 | usbd_ep0wr | Write to endpoint 0 |
| 0x0008 | usbd_ep1wr | Write to endpoint 1 |
| 0x000c | usbd_ep2wr | Write to endpoint 2 |
| 0x0010 | usbd_ep3rd | Read from endpoint 3 |
| 0x0014 | usbd_ep4rd | Read from endpoint 4 |
| 0x0018 | usbd_inten | Interrupt Enable Register |
| 0x001c | usbd_intstat | Interrupt Status Register |
| 0x0020 | usbd_config | Write Configuration Data |
| 0x0024 | usbd_ep0cs | Endpoint 0 control and status |
| 0x0028 | usbd_ep1cs | Endpoint 1 control and status |
| 0x002c | usbd_ep2cs | Endpoint 2 control and status |
| 0x0030 | usbd_ep3cs | Endpoint 3 control and status |
| 0x0034 | usbd_ep4cs | Endpoint 4 control and status |
| 0x0038 | usbd_framenum | Current frame number |
| 0x0040 | usbd_ep0rdstat | EP0 Read FIFO Status |
| 0x0044 | usbd_ep0wrstat | EP0 Write FIFO Status |
| 0x0048 | usbd_ep1wrstat | EP1 Write FIFO Status |
| 0x004c | usbd_ep2wrstat | EP2 Write FIFO Status |
| 0x0050 | usbd_ep3rdstat | EP3 Read FIFO Status |
| 0x0054 | usbd_ep4rdstat | EP4 Read FIFO Status |
| 0x0058 | usbd_enable | USB Device Controller Enable |

Note 1. See Table 7-6 for base address.

### 7.3.1.1 Endpoint FIFO Read and Write Registers

The endpoint FIFO read and write registers provide access to the endpoint FIFOs. Each endpoint FIFO is unidirectional. FIFO read registers may not be written, and FIFO write registers return unpredictable results if read.

Only the least significant byte of the FIFO registers contain data.

| | |
|---|---|
| **usbd_ep0rd** | **Offset = 0x0000** |
| **usbd_ep0wr** | **Offset = 0x0004** |
| **usbd_ep1wr** | **Offset = 0x0008** |
| **usbd_ep2wr** | **Offset = 0x000C** |
| **usbd_ep3rd** | **Offset = 0x0010** |
| **usbd_ep4rd** | **Offset = 0x0014** |

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | DATA | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:8 | — | Reserved, should be cleared. | R/W | 0 |
| 7:0 | DATA | Data. Byte of data to be written to, or read from the endpoint FIFO. | R/W | 0 |

### 7.3.1.2 Interrupt Registers

Each endpoint has an interrupt enable register and an interrupt status register. The two registers have identical formats. When a condition becomes true the corresponding bit is set in the **usbd_intstat** register. If a bit is set in the interrupt enable register and the corresponding condition becomes true, then an interrupt is issued. The interrupt for the USB device should be programmed to high level.

Interrupts and pending conditions must be cleared by writing a 1 to the corresponding bit in the **usbd_intstat** register.

| | |
|---|---|
| **usbd_inten** | **Offset = 0x0018** |
| **usbd_intstat** | **Offset = 0x001C** |

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | SF | H5 | H4 | H3 | H2 | H1 | H0 | C5 | C4 | C3 | C2 | C1 | C0 |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:13 | — | Reserved, should be cleared. | R/W | 0 |
| 12 | SF | Start of Frame. This interrupt issues when an SOF token is received. | R/W | 0 |
| 11:6 | H5:H0 | FIFO Half Full. These interrupts issue when the corresponding FIFO reaches the half full/half empty mark. The bits correspond as follows:<br>H0 - ep0rd<br>H1 - ep0wr<br>H2 - ep1wr<br>H3 - ep2wr<br>H4 - ep3rd<br>H5 - ep4rd | R/W | 0 |
| 5:0 | C5:C0 | Complete. These interrupts issue when a transmission or reception completes on the corresponding FIFO. For the read FIFOs (ep0rd, ep3rd, and ep4rd) these interrupts indicate the reception of a DATA0 or DATA1 packet, or a SETUP packet (ep0rd FIFO only). For the write FIFOs (ep0wr, ep1wr, and ep2wr) these interrupts indicate the transmission of a DATA0 or DATA1 packet. The bits correspond as follows:<br>C0 - ep0rd<br>C1 - ep0wr<br>C2 - ep1wr<br>C3 - ep2wr<br>C4 - ep3rd<br>C5 - ep4rd | R/W | 0 |

#### 7.3.1.3 Device Configuration Register

The device configuration register allows configuration data to be loaded to the controller after reset.

**usbd_config** **Offset = 0x0020**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | CFGDATA | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:8 | — | Reserved, should be cleared. | R/W | 0 |
| 7:0 | CFGDATA | Configuration data byte. Use this field to write the configuration data block to the controller one byte at a time. | R/W | 0 |

The device configuration data is a 25-byte block which contains the configuration information for the five supported endpoints. Each endpoint requires five configuration bytes in the format shown in Figure 7-1.

| Bit: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | Endpoint number | | | | 0 | 1 | 0 | 0 |
| Byte 1 | 0 | 0 | Type | | Direction | Max packet size[9:7] | | |
| Byte 2 | Max packet size[6:0] | | | | | | | 0 |
| Byte 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Byte 4 | 0 | 0 | 0 | 0 | FIFO number | | | |

**Figure 7-1. Endpoint Configuration Data Structure**

The configuration fields are described in Table 7-8.

**Table 7-8. Endpoint Configuration Field Descriptions**

| Field | Description |
|---|---|
| Endpoint Number | Although the endpoint number ranges from 0 to 15, only endpoints 0, 1, 2, 3, and 4 are supported. It is highly recommended that the example values in Table 7-9 on page 112 be used for this field. |
| Type | Endpoint type.<br>00 Control<br>01 Isochronous<br>10 Bulk<br>11 Interrupt |
| Direction | Endpoint direction. (Does not apply to control endpoints.)<br>0 Out<br>1 In |
| Max Packet Size | Maximum packet size (in bytes). Note that for control, bulk, and interrupt endpoints, the maximum packet size is limited to 64 bytes. Only isochronous endpoints can accept packets up to 1023 bytes.<br>000 0000 000 = 0 bytes<br>000 0000 001 = 1 byte<br>...<br>111 1111 111 = 1023 bytes |
| FIFO Number | This field designates which FIFO the endpoint uses. For endpoint 0 this field is ignored since endpoint 0 always uses FIFOs 0 and 1. It is highly recommended that the example values in Table 7-9 on page 112 be used for this field. |

After the controller is removed from reset, the device configuration data must be written to the **usbd_config** register in order beginning with byte 0. Bytes are written individually using unsigned 32-bit words, as shown the following example code:

```
for (i=0; i<25; i++)
    *usbd_config = (unsigned int) cfg_data_bytes[i];
```

An example configuration data block is shown in Table 7-9.

**Table 7-9. Example Endpoint Configuration Data Block**

| Byte | Value | Description |
|---|---|---|
| 0 | 0000 0100 | Endpoint number = 0 |
| 1 | 0000 0000 | Type = control |
| 2 | 1000 0000 | Direction = bidirectional |
| 3 | 0000 0000 | Max packet size = 64 bytes |
| 4 | 0000 0000 | FIFOs 0 and 1 |
| 5 | 0001 0100 | Endpoint number = 1 |
| 6 | 0011 1000 | Type = interrupt |
| 7 | 1000 0000 | Direction = in |
| 8 | 0000 0000 | Max packet size = 64 bytes |
| 9 | 0000 0010 | FIFO 2 |
| 10 | 0010 0100 | Endpoint number = 2 |
| 11 | 0010 1000 | Type = bulk |
| 12 | 1000 0000 | Direction = in |
| 13 | 0000 0000 | Max packet size = 64 bytes |
| 14 | 0000 0011 | FIFO 3 |
| 15 | 0011 0100 | Endpoint number = 3 |
| 16 | 0010 0000 | Type = bulk |
| 17 | 1000 0000 | Direction = out |
| 18 | 0000 0000 | Max packet size = 64 bytes |
| 19 | 0000 0100 | FIFO 4 |
| 20 | 0100 0100 | Endpoint number = 4 |
| 21 | 0010 0000 | Type = bulk |
| 22 | 1000 0000 | Direction = out |
| 23 | 0000 0000 | Max packet size = 64 bytes |
| 24 | 0000 0101 | FIFO 5 |

### 7.3.1.4 Endpoint Control Registers

The endpoint control registers define parameters and reflect operational conditions for each endpoint.

**usbd_ep0cs** **Offset = 0x0024**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | SU | N | A | B |  |  |  |  |  | SZ |  |  |  |  | FS |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**usbd_ep1cs** **Offset = 0x0028**
**usbd_ep2cs** **Offset = 0x002C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | SU | N | A |  |  |  |  |  |  | SZ |  |  |  |  | FS |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**usbd_ep3cs** **Offset = 0x0030**
**usbd_ep4cs** **Offset = 0x0034**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | SU | N | A |  |  |  |  |  |  |  |  |  |  |  | FS |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:15 | — | Reserved, should be cleared. | R/W | 0 |
| 14 | SU | Setup Received - This bit is set when a SETUP packet is received from the host. It is only valid for EP 0. | R | 0 |
| 13 | N | NAK. This bit is set when an operation does not complete successfully or when data in a receive FIFO should be ignored. For most cases this implies a returned NAK in response to a DATA packet or an incorrect CRC. | R | 0 |
| 12 | A | ACK. This bit is set when an operation completes successfully. Most of the time this means that the Host returned an ACK to a DATA (or SETUP) packet or that a packet was received correctly and an ACK returned to the Host.<br>Isochronous DATA and SETUP packets deviate from this model. For these types of packets the A bit indicates successful transmission or reception but no ACK is returned or expected. | R | 0 |
| 11 | B | Alternate ACK - Set when a DATA frame is correctly received on endpoint 0. (B is not present on other endpoints.) | R | 0 |
| 10:1 | SZ | The SZ field specifies the data size of an IN transfer. The SZ field applies only to endpoints 0, 1, and 2. | R/W | 0 |
| 0 | FS | Force Stall. Setting this bit places the endpoint in a stalled condition. Any transaction directed to the endpoint is answered with a STALL response. STALL is typically used to indicate that the endpoint has halted. Note that a Clear Feature command received via the USB does not clear a stall condition forced by this bit. | R/W | 0 |

### 7.3.1.5 Current Frame Number

This register provides the current frame number from the start of frame packet.

**usbd_framenum - Current Frame Number** **Offset = 0x0038**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | FN |  |  |  |  |  |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:11 | — | Reserved, should be cleared. | R | 0 |
| 10:0 | FN | Frame Number. This field contains the frame number from the start of frame packet. | R | 0 |

### 7.3.1.6 FIFO Status Registers

Each FIFO has a status register that indicates the current state and any error conditions. The USB FIFOs are 1-byte wide and eight bytes deep.

| | |
|---|---|
| **usbd_ep0rdstat** | **Offset = 0x0040** |
| **usbd_ep0wrstat** | **Offset = 0x0044** |
| **usbd_ep1wrstat** | **Offset = 0x0048** |
| **usbd_ep2wrstat** | **Offset = 0x004C** |
| **usbd_ep3rdstat** | **Offset = 0x0050** |
| **usbd_ep4rdstat** | **Offset = 0x0054** |

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | FL | UF | OF | FCNT | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:7 | — | Reserved, should be cleared. | R/W | 0 |
| 6 | FL | Flush FIFO. Setting this bit flushes the corresponding FIFO and discards any data contained in it. | W | 0 |
| 5 | UF | Underflow Flag. Set when attempting a read from an empty FIFO. Clear this flag by writing a 1 to it. | R/W | 0 |
| 4 | OF | Overflow Flag. Set if a byte is written to a full FIFO. Clear this flag by writing a 1 to it. | R/W | 0 |
| 3:0 | FCNT | FIFO Count. Reflects the current number of bytes (0 to 8) in the corresponding FIFO. | R | |

### 7.3.1.7 Device Controller Enable Register

The USB device controller enable register (**usbd_enable**) controls the clocks and reset to the device controller. The programmer should first enable clocks before enabling the device controller. To bring the USB device out of reset, follow these steps:

1) Set the CE bit to enable clocks.
2) Delay for a period greater than or equal to 1 µs.
3) Set the E bit to enable the peripheral.
4) Delay at least 1 µs before programming any registers in the peripheral.

**usbd_enable** **Offset = 0x0058**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | SI | CE | E |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:3 | — | Reserved, should be cleared. | W | 0 |
| 2 | SI | Streaming Isochronous Mode. Clearing this bit allows isochronous endpoints to service IN and OUT transactions when the endpoint interrupt is pending. This mode is enabled by default.<br>Setting this bit (not recommended) forces ISO endpoints to wait for pending interrupts to be cleared before accepting further data. | W | 0 |
| 1 | CE | Clock Enable. Clearing this bit disables all clocks to the USB Device core. Setting this bit allows normal operation. | W | 0 |
| 0 | E | Enable. When this bit is cleared the Device Controller will be held in reset. Setting this bit enables normal operation. | W | 0 |

### 7.3.2 Programming Considerations

#### 7.3.2.1 Removing the Controller from Reset

The following sequence of operations must be applied to remove the controller from reset.

1) Write a 0x0002 to the **usbd_enable** register to enable the clocks.
2) Wait 1 µs.
3) Write a 0x0003 to the **usbd_enable** register to remove the controller from reset.
4) Wait 1µs.
5) Write 25 bytes of configuration data to the **usbd_config** register.

There are no special constraints on *entering* the reset state: One write to the **usbd_enable** register may be used to turn the clocks off and reset the controller.

Note: Accessing the endpoint control registers (**usbd_ep*n*cs)**, frame number register (**usbd_framenum**) or configuration data register (**usbd_config**) while the USB device is in suspend mode will result in a System Bus (SBUS) deadlock. This will inhibit any further operation of the CPU, including EJTAG debugger operation.

#### 7.3.2.2 Latency Requirements

The time from reception of a token such as IN or OUT until the controller must source the corresponding DATA frame is very short. It is not practical to wait for a token before preparing the buffer for the response. Buffers must be posted before the token is received.

The token itself is not passed to the buffer—only DATA and SETUP frames are transferred. When a DATA or SETUP frame is received the difference between an OUT and a SETUP can be determined by examining the SU bit in the **usbd_ep0cs** register. (Only endpoint 0 should receive SETUP packets.)

If an IN endpoint is enabled and no data is available in the FIFO the endpoint will NAK. Underrunning the FIFO during a transfer (after the first byte has been written to the FIFO) will result in a bit stuff error.

#### 7.3.2.3 Using DMA

DMA should be used for all transfers with the exception of the FIFO cleanout described below for OUT transactions.

For IN transactions the size in the **usbd_ep*n*cs** register should be set to MAXPACKET for all but the last buffer and to the actual remaining transfer size for the last packet. The DMA size must be set to match **usbd_ep*n*cs** before the DMA is enabled for proper frame transmission.

If the last buffer of an IN series is a full MAXPACKET in length it may be necessary to set the size in the **usbd_ep*n*cs** register to zero and write a byte to the FIFO to enable the transmission of a zero length DATA frame since this is often the indicator for end-of-transfer. In this case the FIFO must be cleared before the next buffer is set up.

For OUT endpoints the DMA may be programmed to a larger size than a transfer will use. When the endpoint completes the FIFO should be examined to see if there are any remaining bytes available. These bytes must be read from the FIFO under program control since the DMA will not receive a request when less than 4 bytes are in the FIFO.

For endpoint zero it is necessary to keep a DMA buffer enabled at all times since SETUP and OUT transactions can come at any time. The user should implement a circular buffer and extract transactions from this buffer in software, rather than trying to have the DMA place transactions into separate buffers.

#### 7.3.2.4 Servicing Interrupts

When an interrupt is received the **usbd_intstat** register should be read to determine the cause of the interrupt. Once the interrupt has been serviced the **usbd_intstat** register should be written with the same value to clear the interrupt.

When an IN or OUT transaction is completed the device controller will NAK all further IN/OUT tokens until the interrupt is cleared by writing the **usbd_intstat** register. This allows the interrupt service routine time to drain the FIFO and set up for the next transaction rather than concatenating data from separate transactions. This automatic wait can be disabled for isochronous endpoints by clearing **usbd_enable**[SI]. SETUP packets can never be delayed with NAK.

For bulk, isochronous, and interrupt endpoints the A and N flags are somewhat redundant. Only one of them should be set for a given transaction.

For the control endpoint the flags are broken out to provide separate feedback for various phases of control transactions. This is necessary since only the IN and OUT phases can be paused with NAKs. SETUP packets must be absorbed gracefully at all times.

The A flag (ACK) is set to indicate successful reception of OUT or SETUP packets. The B flag (alternate ACK) is set to indicate successful reception of OUT data only. (SETUP packets do not affect this flag.) The N flag (NAK) is set to indicate an unsuccessful attempt to send data in response to an IN token.

This combination of flags allows all situations to be decoded. The most complex of these is when a SETUP packet immediately follows an OUT phase used to acknowledge the previous transaction. Without this separation the acknowledgement would be lost.

#### 7.3.2.5 Automatic Execution of Commands

Some standard setup commands directed at endpoint zero are automatically serviced by the USB device hardware. These commands are still passed to the memory buffer. No further action is required to service these commands although they may be used to signal state changes within the software.

The following commands are automatically serviced:

- Set Address
- Set/Clear Feature
- Set/Get Configuration
- Set/Get Interface
- Get Status

#### 7.3.2.6 Detecting USB Reset

The USB device controller does not provide a way to detect reset on the USB. It is recommended that if a device needs to change state on reset it should use the reception of a **Set Address** command to indicate that a reset has occurred.

#### 7.3.2.7 Automatic Suspension

If the USB device is Idle for more than 5 ms, the device controller enters a suspend state. In this state the device controller does not consume data. A rising edge suspend interrupt is provided to inform the CPU when this occurs. The suspend interrupt may also be used to detect the exit from suspend by using the falling edge of the interrupt.

Note: Because the USB device controller will suspend itself if left Idle, the USB device configuration routine (including programming the interrupt controller to recognize request and suspend interrupts from the USB device) must be *fully* completed within 5 ms of bringing the peripheral out of its reset state.

#### 7.3.2.8 Re-establishing a Connection after Reset

During software initialization of the USB device controller, the USBDP and USBDM signals do not automatically enter a disconnect-bus state in which both signals go low for more than 2.5 µs. Instead, after a runtime or hardware reset of the system, the signals stay in a connect-bus state in which USBDP remains high and USBDM remains low. This prevents the USB host from recognizing the need to establish a new bus enumeration, and the logical communication flow remains disrupted.

To re-establish logical communication after reset, system initialization software can control a GPIO signal to temporarily (more than 2.5 µs) disable power to USBDP. It is recommended to use the GPIO to toggle an LDO (low drop-out) voltage regulator placed between the USB power supply (VBUS) and the pull-up resistor attached to USBDP.

### 7.3.3 Programming Examples for USB Device

#### 7.3.3.1 Initialization

1) Configure 4 8MHz USB device clock from AUX PLL.

```
sys_auxpll = 16;    // set the AUX PLL to 192 MHz (12 MHz x 16)
sys_freqctrl0 |= 0x3;  // enable FREQ0 and select
                    AUX PLL as the FREQ0 source
sys_clksrc |= 0xB;  // divide FREQ0 by 4 to obtain
                    48 MHz, and select FREQ0 as the USB clock
```

2) Enable USB Device Controller.

```
usbd_enable = 0x02;    // enable USBD clocks
wait at least 1us;
usbd_enable = 0x03;    // remove reset from USBD Controller
wait at least 1us;
```

3) Write 25-byte configuration data to the Configuration Register.

```
for( i = 0; i < 25; ++i )
{
          usbd_config = (unsigned int) config_data_bytes[i];
}
wait at least 1us;
```

4) Set up Endpoint Control Registers (example).

```
usbd_ep0cs = 64 << 1;  // set endpoint 0 MAXPACKET
usbd_ep1cs = 8 << 1;   // set endpoint 1 MAXPACKET
usbd_ep2cs = 8 << 1;   // set endpoint 2 MAXPACKET
usbd_ep3cs = 8 << 1;   // set endpoint 3 MAXPACKET
usbd_ep4cs = 8 << 1;   // set endpoint 4 MAXPACKET
```

5) Clear FIFO Status Registers.

```
//  clear Overflow Flag, Underflow Flag, Flush FIFO
usbd_ep0rdstat = 0x70;
usbd_ep0wrstat = 0x70;
usbd_ep1wrstat = 0x70;
usbd_ep2wrstat = 0x70;
usbd_ep3rdstat = 0x70;
usbd_ep4rdstat = 0x70;
```

6) Configure DMA channels.

```
// assign a DMA channel for endpoint 0 receive and build
   multiple buffer descriptors
// assign a DMA channel for endpoint 0 transmit and build
   multiple buffer descriptors
// assign a DMA channel for endpoint 1 transmit and build
   multiple buffer descriptors, if necessary
// assign a DMA channel for endpoint 2 transmit and build
   multiple buffer descriptors, if necessary
// assign a DMA channel for endpoint 3 receive and build
   multiple buffer descriptors, if necessary
// assign a DMA channel for endpoint 4 receive and build
   multiple buffer descriptors, if necessary
```

7) Configure the interrupt type for the USB device request (interrupt controller 0, number 24) as high-level.

8) Configure the interrupt type for the USB device suspend (interrupt controller 0, number 25) as rising-edge.

9) Start the Endpoint 0 receive DMA.

10) Enable USB Interrupts.

```
usbd_inten = 0x0000003f;// enable transfer-complete interrupts
```

#### 7.3.3.2 Interrupt Handler

The steps to handle an interrupt are shown below. This example handler is for a general USB application and may not be sufficient for a specific application. The handler must be installed before interrupts are enabled.

1) Obtain the USBD interrupt status.

```
status = usbd_intstat// obtain usbd_intstat
```

2) Execute each interrupt condition.

```
// check if endpoint transfer complete
{
   // if ep0rd completed, execute the process for ep0rd.
   // if ep0wr completed, execute the process for ep0wr.
   // if ep1wr completed, execute the process for ep1wr.
   // if ep2wr completed, execute the process for ep2wr.
   // if ep3rd completed, execute the process for ep3rd.
   // if ep4rd completed, execute the process for ep4rd.
}
```

3) Clear the USBD interrupt status.

```
usbd_intstat = status;// clear interrupts
```

### 7.3.4 Hardware Considerations

Table 7-10 shows the signals associated with the USB device. The USB root hub port pins have USB 1.1 compliant drivers with the addition of the external circuitry noted in the signal description. The USB device implementation is full speed with the required termination noted in Table 7-10. Low speed is not supported.

**Table 7-10. USB Device Signals**

| Signal | Input/Output | Description |
|---|---|---|
| USBDP | I/O | Positive signal of differential USB device driver.<br>Requires a 1.5 kohm pull-up resistor to denote a full speed device. Also requires an external ESD protection diode (transient voltage suppressor) to be USB 1.1 compliant.<br>**Termination Note:** Requires an external 27 ohm resistor placed in series within 0.5 inches of the part.<br>Muxed with USBH0P |
| USBDM | I/O | Negative signal of differential USB device driver.<br>Requires an external ESD protection diode (transient voltage suppressor) to be USB 1.1 compliant.<br>**Termination Note:** Requires an external 27 ohm resistor placed in series within 0.5 inches of the part.<br>Muxed with USBH0M |

For changing pin functionality please refer to the **sys_pinfunc** register in Section 8.3 "Primary General Purpose I/O and Pin Functionality" on page 168.

## 7.4 Ethernet MAC Controller

The Au1500 processor contains two Ethernet MAC devices. The MAC provides the interface between the host application and the PHY layer through the Media Independent Interface (MII). The PHY layer device is external to the processor.

The MAC supports the protocol requirements to meet the Ethernet/IEEE 802.3 specification. The MAC operates in both half and full duplex modes. In half duplex mode the MAC is compliant with section 4 of ISO/IEC 8802-3 (ANSI/IEEE Standard) and ANSI/IEEE 802.3.

The MAC provides programmable enhanced features designed to minimize host supervision, bus utilization and pre/post message processing. These features include ability to disable retries after a collision, dynamic FCS generation on a frame by frame basis, automatic pad field insertion and deletion to enforce minimum frame size attributes, automatic retransmission and detection of collision frames. The MAC can sustain transmission or reception of minimal size back-to-back packets at full line speed with an inter-packet gap of 9.6 µs for 10 Mbps and 0.96 µs for 100 Mbps.

A dedicated DMA engine is implemented to support the MAC so that the general purpose DMA is not required.

The primary attributes of the MAC are:

- Transmit and receive message data encapsulation with framing and error detection.
- Frame boundaries are delimited and frames are synchronized. Error detection is done at the physical medium transmission level.
- Media access management is supported through medium allocation and contention resolution. This is accomplished through collision avoidance and handling. The MAC handles collision per the ISO 8802.3 specification.
- Support for flow control during full duplex mode is accomplished by decoding of control frames and disabling the transmitter in conjunction with generation of control frames.
- The serial control interface supports the MII protocol to interface to an MII based PHY.

The MAC features are:

- IEEE 802.3, 802.3u, 803.3x specification compliance
- 10/100 Mbps data transfer rates
- IEEE 802.3 compliant MII interface to talk to an external PHY
- Full and half duplex
- CSMA/CD in half duplex
- Flow control support for full duplex
- Collision detection and auto retransmit on collisions in half duplex
- Preamble generation and removal
- Automatic 32 bit CRC generation and checking
- Optional automatic Pad stripping on the receive packets.
- Loopback support on the MII
- Filtering modes supported on the Ethernet side:
  — One 48 bit perfect address
  — 64 hash-filtered multicast addresses
  — Pass all multicast addresses
  — Promiscuous Mode
  — Pass all incoming packets with a status report
  — Toss bad packets
- Separate 32-bit status returned for transmit and receive packets
  — Jumbo packet (0x2800 bytes)
  — Big/Little Endian data format support

The following PHY interfaces are supported:

- MII - Ethernet 4-bit parallel PHY per IEEE 802.3u spec
- MII Management - 2-wire bus to control and receive status from PHY
- HPNA 1.0 support across MII

The control registers for the MAC are used for address filtering, packet filter for good and bad frames, 48-bit MAC address with a local station address, a multicast table for filtering multicast frames and more. Each register is 32 bits wide.

### 7.4.1 Ethernet Registers

The two Ethernet MACs contained in the Au1500 processor are located at the base addresses shown in Table 7-11. In addition, the base addresses for the enable registers and the MAC DMA registers are shown.

**Table 7-11. Ethernet Base Addresses**

| Name | Physical Base Address | KSEG1 Base Address |
|---|---|---|
| mac0_base | 0x0 1150 0000 | 0xB150 0000 |
| mac1_base | 0x0 1151 0000 | 0xB151 0000 |
| macen_base | 0x0 1152 0000 | 0xB152 0000 |
| macdma0_base | 0x0 1400 4000 | 0xB400 4000 |
| macdma1_base | 0x0 1400 4200 | 0xB400 4200 |

### 7.4.2 MAC Registers

The Ethernet MAC registers are listed in Table 7-12.

**Table 7-12. MAC Register Descriptions**

| Offset (Note 1) | Register Name | Description |
|---|---|---|
| 0x0000 | mac_control | Operation Mode and Address Filter. |
| 0x0004 | mac_addrhigh | High 16 bits of the MAC Physical Address. |
| 0x0008 | mac_addrlow | Lower 32 bits of the MAC Physical Address. |
| 0x000C | mac_hashhigh | High 32 bits of the Multicast Hash Address. |
| 0x0010 | mac_hashlow | Low 32 bits of the Multicast Hash Address. |
| 0x0014 | mac_miictrl | Control of PHY Management Interface. |
| 0x0018 | mac_miidata | Data to be written or read from PHY over control interface. |
| 0x001C | mac_flowctrl | Control Frame Generation Control. |
| 0x0020 | mac_vlan1 | VLAN1 Tag. |
| 0x0024 | mac_vlan2 | VLAN2 Tag. |

Note 1. Each Ethernet MAC has an identical register set with identical offsets from **mac0_base** and **mac1_base**. See Table 7-11 for base address.

### 7.4.2.1 MAC Control Register

The MAC Control Register establishes the receive and transmit operating modes and controls for address filtering and packet filtering.

Note that the PM, PR, IF, HP and HO bits in the MAC Control register will determine the address filtering mode. The RA, DB, PC and PB bits will determine the packet filter mode. The first bit of the destination address will determine if the address is a physical address (first bit = 0) or a multicast address (first bit = 1). If all bits in the destination address are set to 1 then the address is a broadcast address.

**mac_control** **Offset = 0x0000**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | RA | EM | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | DO | LM | | F | PM | PR | IF | PB | HO | ▨ | HP | LC | DB | DR | ▨ | AP | BL | | DC | ▨ | TE | RE | ▨ | ▨ |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31 | RA | Receive All.<br>0 Normal Operation.<br>1 All incoming packets will be received regardless of the destination address.<br>The address filter status is reported in Receive Status bit Filtering Fail.<br>The Packet Filter bit in the Receive Status is set for all error-free frames regardless of the Destination Address field. | R/W | 0 |
| 30 | EM | Endian Mode for Data Buffers.<br>0 Little endian.<br>1 Big endian. | R/W | 0 |
| 29:24 | — | Reserved, should be cleared. | R | 0 |
| 23 | DO | Disable Receive Own.<br>0 The MAC receives all packets that are given by the PHY.<br>1 The MAC disables reception of frames when the TXEN is asserted. The MAC ignores any loop backed receive packets.<br>This bit should be cleared when the full duplex mode bit is set or the Operating Mode is set to other than Normal Mode. | R/W | 0 |
| 22:21 | LM | Loopback Operating Mode.<br>00 Normal Mode.<br>01 Internal Loopback.<br>10 External Loopback.<br>11 Reserved. | R/W | 00 |
| 20 | F | Full Duplex Mode.<br>0 Half duplex mode.<br>1 Full duplex mode.<br>Note: Be sure to disable both the transmitter and receiver before changing duplex modes. | R/W | 0 |
| 19 | PM | Pass All Multicast.<br>0 Normal.<br>1 All incoming frames with a multicast destination address (first bit in the destination address field is ‘1’) are received and the Filter Fail bit reset.<br>Incoming frames with physical address destinations are filtered according to HP (bit 13) and HO (bit 15). | R/W | 0 |
| 18 | PR | Promiscuous Mode.<br>0 Normal<br>1 Any incoming valid frame is received regardless of its destination address.<br>The Filter Fail bit is always cleared in Promiscuous Mode. | R/W | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 17 | IF | Inverse Filtering.<br>0 Normal<br>1 Physical addresses are checked with inverse filtering. In other words if the address passes a perfect address filter, the frame is not passed; if the address fails a perfect filter, the frame is passed.<br>This is valid only during perfect filtering mode. | R/W | 0 |
| 16 | PB | Pass Bad Frames.<br>0 Normal<br>1 All incoming frames that passed the address filtering are received including runt frames, collided frames, or truncated frames caused by buffer overflow.<br>The Packet Filter bit is set for error frames that pass the Address filtering. If all received bad frames are required, promiscuous mode (bit 18) should be set. | R/W | 0 |
| 15 | HO | Hash Only Filtering Mode.<br>0 Perfect address filtering mode for physical addresses<br>1 Imperfect address filtering mode both for physical and multicast addresses<br>Setting this bit is valid only if HP=1. | R/W | 0 |
| 14 | — | Reserved, should be cleared. | R | 0 |
| 13 | HP | Hash/Perfect Filtering Mode<br>0 Address Check block does a perfect address filter of incoming frames according the address specified in the MAC Address register.<br>1 Address Check block does imperfect address filtering of multicast incoming frames according to the hash table specified in the multicast Hash Table Register. If the Hash Only (HO) bit is set, then physical addresses are imperfectly filtered too. If the Hash Only bit (HO) is reset, then physical addresses are perfect address filtered according to the MAC Address Register. | R/W | 0 |
| 12 | LC | Late Collision Control.<br>0 Abort frame transmission on a late collision.<br>1 Enable the retransmission of the collided frame even after the collision period (late collision).<br>In either case the Late Collision Status is appropriately updated in the Transmit Packet Status.<br>This bit is valid only when operating in half duplex mode. | R/W | 0 |
| 11 | DB | Disable Broadcast Frames.<br>0 Forward all the broadcast frames to the application. (Packet Filter bit is set.)<br>1 Disable the reception of broadcast frames. (Packet Filter bit is cleared.) | R/W | 0 |
| 10 | DR | Disable Retry.<br>0 The MAC will attempt 16 transmissions before signaling a retry error.<br>1 The MAC will attempt transmission of a frame only once. When a collision is seen on the bus, the MAC will ignore the current frame and go to the next frame and a retry error will be reported in the Transmit Status.<br>This bit is valid only when operating in half duplex mode. | R/W | 0 |
| 9 | — | Reserved, should be cleared. | R | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 8 | AP | Automatic Pad Stripping.<br>0 Pass all the incoming frames to the host unmodified.<br>1 Strip the pad field on all the incoming frames if the length field is less than 46 bytes. The FCS field is also stripped, because it is computed at the transmitting station based on the data and pad field characters and will therefore be invalid for a receive frame that has had the pad characters stripped. Receive frames which have a length field of 46 bytes or greater will be passed to the host unmodified (FCS is not stripped).<br>Pad stripping is done only on the IEEE 802.3 formatted frames (frames with Length field). | R/W | 0 |
| 7:6 | BL | Backoff Limit. The Backoff limit determines the integer number of slot times the MAC waits before rescheduling a transmission attempt (during retries after a collision). | R/W | 00 |
| 5 | DC | Deferral Check.<br>0 The deferral check is disabled in the MAC and the MAC defers indefinitely.<br>1 The deferral check is enabled in the MAC. The MAC will abort the transmission attempt if it has deferred for more than 24,288 bit times. Deferring starts when the transmitter is ready to transmit, but is prevented from doing so because CRS is active. Defer time is not cumulative. In other words, if the transmitter defers, then transmits, collides, backs off, and then has to defer again after completion of backoff, the deferral timer resets to 0 and restarts.<br>This bit is valid only when operating in half duplex mode. | R/W | 0 |
| 4 | — | Reserved, should be cleared. | R | 0 |
| 3 | TE | Transmitter Enable.<br>0 The MAC transmitter is disabled and will not transmit any frames on the MII interface.<br>1 The MAC transmitter is enabled and it will transmit frames from the buffer on to the MII interface. | R/W | 0 |
| 2 | RE | Receiver Enable.<br>0 The MAC receiver is disabled and will not receive any frames from the MII interface.<br>1 The MAC receiver is enabled and will receive frames from the MII interface. | R/W | 0 |
| 1:0 | — | Reserved, should be cleared. | R | 0 |

### 7.4.2.2 MAC Address High and Low Registers

The MAC Address High Register contains the upper 16 bits of the physical address of the MAC. The MAC Address Low Register contains the lower 32 bits of the physical address of the MAC.

It is the responsibility of the system designer to provide the MAC address for the system.

The MAC address will be compared with the destination address from the incoming frame with PADR[0] (bit 0 of the Mac Address Low register) being compared with the first bit of the destination address and PADR[47] (bit 15 of the MAC Address High register) compared with the 48th bit of the destination address.

Example: To program the MAC address 00.50.c2.0c.20.10 the MAC address registers should be programmed as follows:

```
mac_addrhigh = 0x00001020
mac_addrlow = 0x0CC25000
```

**mac_addrhigh** **Offset = 0x0004**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | PADR[47:32] | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | — | Reserved, should be cleared. | R | 0 |
| 15:0 | PADR[47:32] | Physical Address [47:32]. Contains the upper 16 bits (47 to 32) of the Physical Address of the MAC. | R/W | 0xFFFF |

**mac_addrlow** **Offset = 0x0008**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PADR[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | PADR[31:0] | Physical Address [31:0]. Contains the lower 32 bits (31 to 0) of the Physical Address of the MAC. | R/W | 0xFFFFFFFF |

### 7.4.2.3 Multicast Address High and Low Hash Table Registers

The 64-bit multicast address hash table is used for group address filtering. For hash filtering, the contents of the destination address in the incoming frame is passed through the CRC logic and the upper 6 bits of the CRC register are used to index the contents of the Hash table. The most significant bit determines the register to be used (1 = Hi, 0 = Low), while the other five bits determine the bit within the register. A value of ‘00000’ selects the bit 0 of the selected register and a value of ‘11111’ selects the bit 31 of the selected register.

If the corresponding bit in the hash table is '1', then the multicast frame is accepted, otherwise it is rejected. If the Pass All Multicast is set, then all multi-cast frames are accepted regardless of the multi-cast hash values. The Multi Cast Hash Table High Register contains the higher 32 bits of the hash table and the Multi Cast Hash Table Low Register contains the lower 32 bits of the hash table.

**mac_hashhigh** **Offset = 0x000C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | MCH[63:32] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | MCH[63:32] | Multicast Address Hash Table High. These bits map to the upper 32 bits of the 64 bit hash table. | R/W | 0x00000000 |

**mac_hashlow** **Offset = 0x0010**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | MCH[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | MCH[31:0] | Multicast Address Hash Table Low. These bits map to the lower 32 bits of the 64 bit hash table. | R/W | 0x00000000 |

### 7.4.2.4 MII Control Register

The MII Address Register is used to control and generate the Management cycles to the External PHY Controller chip. A write to this register will generate a read/write access on the MII Management Interface (MDIO/MDC) bus to an external PHY device.

**mac_miictrl** **Offset = 0x0014**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | PHYADDR[4:0] | | | | | MIIREG[4:0] | | | | | | | | | MW | MB |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | — | Reserved, should be cleared. | R | 0 |
| 15:11 | PHYADDR | PHY Address. These bits tell which of the 32 possible PHY devices are being accessed. | R/W | 00000 |
| 10:6 | MIIREG | MII Register. These bits select the desired MII register in the selected PHY device. | R/W | 00000 |
| 5:2 | — | Reserved, should be cleared. | R | 0 |
| 1 | MW | MII Write.<br>0 Operation will be a read (data read is placed in MII Data Register)<br>1 Operation will be a write (data to be written is taken from MII Data Register) | R/W | 0 |
| 0 | MB | MII Busy. This bit should read a logic 0 before writing to the MII address and MII data registers. This bit should be reset to 0 when writing to the MII address register.<br>This bit will be set by the MAC to signify that a read or write access to the external PHY is in progress. For a write operation the data register should be kept valid until this bit is cleared by the MAC. For a read operation the MII data register is invalid until this bit is cleared by the MAC.<br>The MII address register should not be modified until this bit is cleared. The MAC clears this bit after the PHY access is done. | R/W | 0 |

### 7.4.2.5 MII Data Register

The MII Data Register contains the data to be written to the PHY register specified in the MII address register, or it contains the read data from the PHY register whose address is specified in the MII address register.

**mac_miidata** **Offset = 0x0018**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | MIIDATA[15:0] | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | — | Reserved, should be cleared. | R | 0 |
| 15:0 | MIIDATA | MII Data<br>16-bit value read from the PHY after a MII read operation, or the 16-bit data value to be written to the PHY before a MII write operation. | R/W | 0x0000 |

### 7.4.2.6 Flow Control Register

This register is used to control the generation and reception of the Control (PAUSE Command) frames by the MAC's Flow control block. A write to this register with the busy bit set to '1' triggers the Flow Control block to generate a PAUSE Control frame. The fields of the control frame are selected as specified in the 802.3x specification with the Pause Time field from this register used in the "Pause Time" field of the control frame. The Busy bit will remain set until the control frame is transmitted. The Host has to insure that the Busy bit is cleared before writing to the register. The Pass Control Frames bit indicates to the MAC whether or not to pass the control frame to the Host. The Flow Control Enable bit enables the receive portion of the Flow Control block.

**mac_flowctrl** **Offset = 0x001C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PT[15:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | PC | FE | FB |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | PT | Pause Time. This field will be used in the PAUSE TIME field in the generation of the PAUSE control frame. | R/W | 0x0000 |
| 15:3 | — | Reserved, should be cleared. | R | 0 |
| 2 | PC | Pass Control Frame<br>0 The MAC decodes the control frames but does not pass the frames to the Host. The Control Frame bit in the Receive Status (bit 25) is set and the Transmitter Pause Mode signal gives the current status of the Transmitter, but the PacketFilter bit in the Receive Status is reset to signal the application to flush the frame.<br>1 Control frames are passed to the Host. The MAC decodes the control frame (PAUSE) and disables the transmitter for the specified amount of time. The Control Frame bit in the Receive Status (bit 25) is set, and the Transmitter Pause Mode signal indicates the current state of the MAC Transmitter. | R/W | 0 |
| 1 | FE | Flow Control Enable<br>0 The flow control operation in the MAC is disabled, and the MAC does not decode the frames for control frames.<br>1 The MAC is enabled for flow control operation and it will decode all the incoming frames for control frames. If the MAC receives a valid control frame (PAUSE command), it will disable the transmitter for the specified time.<br>This bit is valid only in full duplex mode. | R/W | 0 |
| 0 | FB | Flow Control Busy Status. This bit should read a logic 0 before writing to the Flow Control register. To initiate a PAUSE control frame the host must set this bit. During a transfer of Control Frame, this bit remains set to signify that a frame transmission is in progress. After the completion of PAUSE control frame transmission, the MAC clears FB. | R/W | 0 |

### 7.4.2.7 VLAN1 Tag Register

**mac_vlan1** **Offset = 0x0020**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | VL1TAG[15:0] | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | — | Reserved, should be cleared. | R | 0 |
| 15:0 | VL1TAG | VLAN 1 Tag Identifier. This field will be compared with the 13th and 14th bytes of the incoming frame. If a nonzero match occurs the VLAN 1 Frame bit will be set in the receiver status packet. In addition, the legal length of a frame is increased from 1518 bytes to 1522 bytes. | R/W | 0xFFFF |

#### 7.4.2.8 VLAN2 Tag Register

**mac_vlan2** **Offset = 0x0024**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | VL2TAG[15:0] | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | — | Reserved, should be cleared. | R | 0 |
| 15:0 | VL2TAG | VLAN 2 Tag Identifier<br>This field will be compared with the 13th and 14th bytes of the incoming frame. If a nonzero match occurs the VLAN 2 Frame bit will be set in the receiver status packet. In addition the legal length of a frame is increased from 1518 bytes to 1538 bytes. | R/W | 0xFFFF |

### 7.4.3 MAC Enable Registers

Each Ethernet MAC has an identical enable register. Both enable registers are located off of the **macen_base** shown in Table 7-11 "Ethernet Base Addresses" on page 120.

#### 7.4.3.1 MAC0 and MAC1 Enable

The enable register for each MAC contains a bit that enables the entire block. The block should be disabled if not in use to minimize power consumption. In addition, each enable register contains a toss bit (TS) which prevents frames that do not pass the address filter from being put into memory.

The process for bringing the MAC out of reset is as follows:

1) Enable clocks (CE=1).

2) Bring E[2:0] high together with the other bits configured as desired (keeping clocks enabled).

**Note:** MAC clocks must be running before the internal MAC registers are accessed.

**macen_mac0** **Offset = 0x0000**
**macen_mac1** **Offset = 0x0004**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | JP | E2 | E1 | C | TS | E0 | CE |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:7 | — | Reserved, should be cleared. | W | 0 |
| 6 | JP | Jumbo Packet Enable.<br>0 Normal (Max packet length = 0x800 bytes)<br>1 Enable Jumbo Packet (Max packet length = 0x2800 bytes) | W | 0 |
| 5:4 | E[2:1] | Enable field bits 2 and 1. Together with E0, this field resets and enables the MAC.<br>000 Reset<br>111 Enable<br>All other combinations are invalid. | W | 00 |
| 3 | C | Coherent.<br>0 Memory accesses are marked coherent on SBUS<br>1 Memory accesses are marked non coherent on SBUS<br>For more information on coherency see Section 2.8.2 "SBUS Coherency Model" on page 42. | W | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 2 | TS | Disable Toss.<br>0 Only frames passing the address filter are passed to memory. Frames which fail length error, CRC error, or other non-address filter failures are still passed to memory.<br>Frames are not passed to memory if the filter fail bit is set, or the frame is a broadcast frame and broadcast frames have been disabled.<br>In promiscuous mode all frames are passed to memory unless the disable broadcast bit is set which prevents broadcast frames from being passed to memory.<br>Frames that are not passed to memory are transparent to software—no status or indication informs software.<br>1 All frames are passed to memory, regardless of address filter result. | W | UNPRED |
| 1 | E[0] | Enable field bit 0. See description for E[2:1]. | W | 0 |
| 0 | CE | Clock Enable.<br>0 Clocks disabled to MAC<br>1 Clocks enabled to MAC | W | 0 |

### 7.4.4 MAC DMA Registers

Each MAC has four DMA buffers for both receive and transmit (four for RX, four for TX). The DMA buffers are serviced in a round-robin fashion. Each MAC has a 32-word FIFO for both transmit and receive. The transfer size for the MAC DMA is eight WORDs. Both the FIFO size and transfer size are taken care of automatically by the MAC DMA and are transparent to the programmer except that all memory buffers must be implemented on a cache line boundary (32 bytes).

The MAC DMA registers can be described as a set of transmit and receive entries off of the MAC DMA base addresses shown in Table 7-11 "Ethernet Base Addresses" on page 120.

Each MAC DMA base address contains eight entries which correspond to four transmit buffer entries and four receive buffer entries as shown in Table 7-13.

**Table 7-13. MAC DMA Entries**

| Offset (Note 1) | Entry Prefix | Entry Name |
|---|---|---|
| 0x000 | tx0 | Transmit Buffer 0 |
| 0x010 | tx1 | Transmit Buffer 1 |
| 0x020 | tx2 | Transmit Buffer 2 |
| 0x030 | tx3 | Transmit Buffer 3 |
| 0x100 | rx0 | Receive Buffer 0 |
| 0x110 | rx1 | Receive Buffer 1 |
| 0x120 | rx2 | Receive Buffer 2 |
| 0x130 | rx3 | Receive Buffer 3 |

Note 1. See Table 7-11 on page 120 for base address.

Within each receive entry there are two registers implemented as shown in Table 7-14. (The third and fourth reserved entries are shown for completeness but are not used.)

**Table 7-14. MAC DMA Receive Entry Registers**

| Offset | Receive Entry Register | Description |
|---|---|---|
| 0x0 | stat | Status register |
| 0x4 | addr | Address/enable register |
| 0x8 | Reserved | Nothing is implemented at this offset. |
| 0xC | Reserved | Nothing is implemented at this offset. |

Within each transmit entry, there are three registers implemented as shown in Table 7-15. (The fourth reserved entry is shown for completeness but is not used.)

**Table 7-15. MAC DMA Transmit Entry Registers**

| Offset | Transmit Entry Register | Description |
|---|---|---|
| 0x0 | stat | Status register |
| 0x4 | addr | Address/enable register |
| 0x8 | len | Length register |
| 0xC | Reserved | Nothing is implemented at this offset. |

To calculate the address of a specific MAC DMA buffer all offsets should be combined. For example the physical address of the MAC1 receive buffer 3 address register is calculated as follows:

```
macdma1_rx3addr = macdma1_base + rx3 + addr
                = 0x0 1400 4200 + 0x130 + 0x4
                = 0x0 1400 4334
```

Another way to look at the DMA register addresses is to view them as built off of the base address using an indexed approach to build the address for each unique register within the block. In other words, each bit (or set of bits) within the address will select a parameter of the DMA Register (TX/RX, Buffer number, Status/Address/Length register) until a unique address is formed selecting a single register.

To build the address for a unique register the bits should be set according to the definitions in Table 7-16.

**Table 7-16. MAC DMA Block Indexed Address Bit Definitions**

| AddrBits | Description |
|---|---|
| 8 | TX/RX.<br>0 Transmit Block<br>1 Receive Block |
| 7:6 | These bits should be cleared. |
| 5:4 | MAC DMA Buffer.<br>00 Buffer 0<br>01 Buffer 1<br>10 Buffer 2<br>11 Buffer 3 |
| 3:2 | Register Select<br>00 Status Register<br>01 Address/Enable Register<br>10 Length Register (valid for transmit only)<br>11 Reserved |
| 1:0 | These bits should be cleared because the registers are aligned on a word boundary. |

The enumerated DMA registers are shown in Section A.1 "Memory Map" on page 255.

### 7.4.4.1 MAC DMA Receive Registers

There are two receive registers for each DMA channel associated with each MAC: the status register and the address/enable register. The length register is not applicable to the receive DMA channel, as the length will be determined by the size of the received packet (typically the size of a frame for a complete, successful reception). The receive memory buffers should be 0x800 bytes when Jumbo Packets are not enabled and to 0x2800 when Jumbo packets are enabled. This will allow for the worst case maximum reception length.

In the naming of the receive registers dummy variables *m* and *n* have been inserted into the name to designate MAC number (*m*) and buffer number (*n*).

#### Receive Status

This register contains the receive packet status bits sent by the MAC after receiving a frame. This register is only valid after a receive transaction has been enabled by the host and the done bit has been set by the MAC in the Address/Enable Register to indicate that the transaction is complete.

The MI bit should be checked by software after receiving a frame to verify that the received frame is valid.

**macdma*m*_rx*n*stat** **Offset = 0x0**

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 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 |
| | MI | PF | FF | BF | MF | UC | CF | LE | V2 | V1 | CR | DB | ME | FT | CS | FL | RF | WT | L[13:0] | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31 | MI | Missed Frame.<br>0 The frame is received normally by the Application without any latency or error violations<br>1 Indicates that a frame was missed due to an internal FIFO overrun. | R | UNPRED |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 30 | PF | Packet Filter.<br>0 Indicates that the current frame failed the packet filter.<br>1 Indicates that the current frame passed the packet filter that is implemented in the MAC.<br>Packet Filter will indicate failed frame when any of the following conditions happens.<br>a. FF = 0 and frame is not a Broadcast or RA is 1<br>b. Frame is Broadcast and DB is 0<br>c. Frame is not Control Frame or PC is 1<br>d. No Error Status or PB Frames is 1<br>e. Unsupported Control Frame is 0<br>The Application can use this bit to decide whether to keep the packet in the memory or flush the packet from the memory.<br>Note that frames with length greater than max Ethernet size (1500 bytes-normal, 1518 bytes - VLAN1, 1538 bytes - VLAN2) will create an error status thus failing the packet filter. The frames may still be valid with failure only due to frame size. | R | UNPRED |
| 29 | FF | Filtering Fail.<br>0 Current frame passed address filtering<br>1 Destination Address field in the current frame failed the Address filtering. | R | UNPRED |
| 28 | BF | Broadcast Frame.<br>0 Destination address is not Broadcast.<br>1 Destination address is all 1’s indicating broadcast address. | R | UNPRED |
| 27 | MF | Multicast Frame.<br>0 Destination address is not multicast.<br>1 Destination address is multicast (the first bit is 1). | R | UNPRED |
| 26 | UC | Unsupported Control Frame.<br>0 If the Control Frame bit is set, this bit indicates a supported control frame has been received (Pause Frame).<br>1 The MAC observed an unsupported Control Frame. This is set when a control frame is received and the opcode field is unsupported, or the length is not equal to minFrameSize (64 bytes). This bit is set only when the MAC is operating in the full-duplex mode. | R | UNPRED |
| 25 | CF | Control Frame.<br>0 Current frame is not a control frame.<br>1 Current frame is a control frame. This bit is only set when operating in Full Duplex mode. | R | UNPRED |
| 24 | LE | Length Error.<br>0 No length error occurred.<br>1 The current frame Length value is inconsistent with the total number of bytes received in the current frame. When the number of bytes received in the data field are more than what indicated in the Length/Type field, the additional bytes are assumed to be PAD bytes and the Length Error bit is not set. When the number of bytes received in the data field is less than what was indicated in the Length/Type field, the Length Error bit is set. This is valid when the Frame Type is set to ’0’ (802.3 Frame).<br>This bit is not applicable for frame lengths greater than max Ethernet size (1500 bytes- normal, 1518 bytes - VLAN1, 1538 bytes - VLAN2). | R | UNPRED |
| 23 | V2 | VLAN2 ID.<br>0 No match with VLAN2 tag<br>1 The current frame is tagged with a VLAN2 ID. The thirteenth and fourteenth bytes of the frame were a nonzero match with the VLAN2 tag register. | R | UNPRED |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 22 | V1 | VLAN1 ID.<br>0 No match with VLAN1 tag<br>1 The current frame is tagged with a VLAN1 ID. The thirteenth and fourteenth bytes of the frame were a nonzero match with the VLAN1 tag register. | R | UNPRED |
| 21 | CR | CRC Error.<br>0 No CRC error in current frame<br>1 CRC error occurred in received frame.<br>This bit is not applicable for frame lengths greater than max Ethernet size (1500 bytes- normal, 1518 bytes - VLAN1, 1538 bytes - VLAN2). If a CRC check is required it must be done in software. | R | UNPRED |
| 20 | DB | Dribbling Bit.<br>0 An integer multiple of eight bits was received.<br>1 A non-integer multiple of eight bits was received. This bit is not valid if either the Collision Seen bit or Runt Frame bit is set. If this bit is set and the CRC Error bit is 0, then the packet is still valid. | R | UNPRED |
| 19 | ME | MII Error.<br>0 No MII error.<br>1 MII error during frame reception. | R | UNPRED |
| 18 | FT | Frame Type.<br>0 IEEE 802.3 Frame.<br>1 Ethernet-type frame (frame length field is greater than max Ethernet size (1500 bytes- normal, 1518 bytes - VLAN1, 1538 bytes - VLAN2).<br>This bit is still applicable when Jumbo packets are enabled.<br>This bit is not valid for runt frames of less than 14 bytes. | R | UNPRED |
| 17 | CS | Collision Seen.<br>0 No collision seen during frame reception.<br>1 The frame was damaged by a collision that occurred after the 64 bytes following the start of frame delimiter (SFD). This is a late collision. | R | UNPRED |
| 16 | FL | Frame Too Long.<br>0 Frame size is less than or equal to max Ethernet frame size (1500 bytes- normal, 1518 bytes - VLAN1, 1538 bytes - VLAN2).<br>1 Frame size is greater than the maximum Ethernet specified size (1500 bytes- normal, 1518 bytes - VLAN1, 1538 bytes - VLAN2). This also applies when Jumbo packets are enabled.<br>Frame too long is only a length indication and does not cause frame truncation. | R | UNPRED |
| 15 | RF | Runt Frame.<br>0 Frame was not damaged in collision window.<br>1 Frame was damaged by a collision or premature termination before the collision window passed. | R | UNPRED |
| 14 | WT | Watchdog Timeout.<br>0 Frame was received before timeout occurred.<br>1 The receive watchdog timer expired while receiving the frame. The watchdog timer inside the MAC is programmed to be twice the MaxFrameLength. When set, the Frame Length field is invalid.<br>Any time the max frame length is exceeded (0x800 bytes for normal mode, 0x2800 with Jumbo packets enabled) the WT bit will be set. | R | UNPRED |
| 13:0 | L[13:0] | Frame Length. Indicates length in bytes of the received frame. The host should take into account how the Automatic Pad Stripping (AP) bit in the corresponding MAC control register is set, as this will affect how the length field and frame contents should be interpreted. | R | UNPRED |

### Receive Buffer Address/Enable Register

This register contains the starting address for the receive buffer. The host should ensure that the memory buffer is set up to accommodate the worst case largest frame size to be able to handle all received packets. At worst case the MAC will receive 0x800 bytes before aborting a receive in normal mode or 0x2800 bytes when Jumbo packets have been enabled in the **macen_mac*n*** register.

After the transaction has been enabled this register should not be written until the DN bit has been set.

The buffer for the DMA must be cache line aligned so the lowest 5 bits are not used as part of the address. These bits have been employed as done and enable bits that are exclusive of the address.

**macdma*m*_rx*n*addr** **Offset = 0x4**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ADDR[31:5] | | | | | | | | | | | | | | | | | | | | | | | | | | | ▨ | CB | | DN | EN |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:5 | ADDR | Buffer Address. Upper 27 bits of the starting physical address for the DMA buffer. This address must be cache line (32 bytes) aligned so only 27 bits are used. This address must be written for each DMA transaction (the address will not remain after the transaction is enabled) | R/W | 0 |
| 4 | — | Reserved, should be cleared. | R | 0 |
| 3:2 | CB | Current Buffer. Current DMA Receive Buffer | R | UNPRED |
| 1 | DN | Transaction Done. This bit will be set by hardware to indicate that the receive transaction has been completed and that the receive packet status is valid.<br>If the respective MAC DMA interrupt is enabled (see Section 6.0 "Interrupt Controller" on page 95), an interrupt will be generated when this bit is set. Done bits for all TX and RX buffers are or'ed together for this interrupt so a high level interrupt should be used.<br>This bit must be cleared explicitly by software after checking for done. This will also clear the interrupt. | R/W | UNPRED |
| 0 | EN | MAC DMA Enable. When set, this bit enables a DMA receive transaction to the memory location designated in ADDR. | R/W | UNPRED |

### 7.4.4.2 MAC DMA Transmit Registers

There are three transmit registers, including the status register, the address/enable register, and the length register. In the naming of the receive registers dummy variables *m* and *n* have been inserted into the name to designate MAC number (*m*) and buffer number (*n*).

### Transmit Packet Status Register

This register contains the transmit packet status bits sent by the MAC after transmitting a frame. This register is valid after a transmit transaction has been enabled by the host and the done bit has been set by the MAC in the Address/Enable Register to signify that the transmit transaction is complete.

If either PR (bit 31) or FA (bit 0) is set then the frame was not sent successfully and the application should resend the frame.

**macdma*m*_tx*n*stat** **Offset = 0x0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PR | ▨ | | | | | | | | | | | | | | | | | CC | | | | LO | DF | UR | EC | LC | ED | LS | NC | JT | FA |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31 | PR | Packet Retry.<br>0 Transmission of current packet is complete.<br>1 The Application has to restart the transmission of the frame (packet) when this bit is set to ‘1’. The successful/unsuccessful completion of the frame’s transmission is indicated by the Frame Aborted bit (bit 0). | R | UNPRED |
| 30:14 | — | These bits are reserved. | R | UNPRED |
| 13:10 | CC | Collision Count.<br>This 4-bit count indicates the number of collisions that occurred before the frame was transmitted. This bit is not valid when the Excessive Collisions bit is set.<br>This bit is valid only when the MAC is operating in half-duplex mode. | R | UNPRED |
| 9 | LO | Late Collision Observed.<br>0 No late collision observed during transmission.<br>1 Indicates that the MAC observed a late collision (collision after 64 bytes into transmission of frame), but retransmitted the frame in the next retransmission attempt. This bit will be set when the Late Collision bit is set.<br>This bit is valid only when the MAC is operating in half-duplex mode. | R | UNPRED |
| 8 | DF | Deferred.<br>0 Transmitter did not defer when transmitting.<br>1 The transmitter had to defer while ready to transmit a frame.<br>This bit is valid only when operating in half-duplex mode. | R | UNPRED |
| 7 | UR | Under Run.<br>0 No data under run.<br>1 The transmitter aborted the message because of data under run during the frame’s transmission. | R | UNPRED |
| 6 | EC | Excessive Collisions.<br>0 Transmission did not abort due to excessive collisions.<br>1 Transmission aborted after 16 successive collisions. If the Disable Retry bit is set, this bit is set after the first collision and the transmission of the frame will be aborted.<br>This bit is valid only when operating in half-duplex mode. | R | UNPRED |
| 5 | LC | Late Collision.<br>0 No late collision.<br>1 Transmission was aborted due to collision occurring after the collision window of 64 bytes. This bit is not valid if under run error is set.<br>This bit is valid only when operating in half-duplex mode. | R | UNPRED |
| 4 | ED | Excessive Deferral.<br>0 No excessive deferral.<br>1 Transmission has ended because of excessive deferral of over 24,288 bit times during the transmission, if the defer bit is set high in the control register.<br>This bit is valid only when operating in half-duplex mode. | R | UNPRED |
| 3 | LS | Loss of Carrier.<br>0 No loss of carrier.<br>1 The loss of carrier occurred during the frame's transmission (i.e., the CRS input was inactive for one or more bit times when the frame is being transmitted).<br>This bit is valid only when operating in half-duplex mode. | R | UNPRED |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 2 | NC | No Carrier.<br>0 Carrier present<br>1 The carrier signal from the transceiver was not present during transmission.<br>This bit is valid only when operating in half-duplex mode. | R | UNPRED |
| 1 | JT | Jabber Timeout.<br>0 No jabber timeout.<br>1 The MAC transmitter has been active for an abnormally long time (twice the Ethernet maxFrameLength size). | R | UNPRED |
| 0 | FA | Frame Aborted.<br>0 Current frame was successfully transmitted.<br>1 The transmission of the current frame has been aborted by the MAC because of one or more of the following conditions:<br>Jabber Timeout (bit 1)<br>No Carrier (bit 2)<br>Loss of Carrier (bit 3)<br>Excessive Deferral (bit 4)<br>Late Collision (bit 5)<br>Retry Count exceeds the attempt limit (bit 6).<br>Data under run (bit 7) | R | UNPRED |

### Transmit Buffer Address/Enable Register

This register contains the starting address for the transmit memory buffer. The MAC DMA transfers the number of bytes designated in the Length register.

The buffer for the DMA must be cache line aligned so the lowest 5 bits are not used as part of the address. These bits have been employed as done and enable bits and are exclusive of the address.

**macdma*m*_tx*n*addr** **Offset = 0x4**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ADDR[31:5] | | | | | | | | | | | | | | | | | | | | | | | | | | | ▨ | CB | | DN | EN |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:5 | ADDR | Buffer Address. Upper 27 bits of the starting physical address for the DMA buffer—but not including the most significant nibble address bits 35:32. This address must be cache line (32 bytes) aligned so only 27 bits are used.<br>Note: This address must be written for each DMA transaction (the address will not remain after the transaction is enabled). | R/W | 0 |
| 4 | — | Reserved, should be cleared. | R | 0 |
| 3:2 | CB | Current Buffer. Current DMA Transmit Buffer | R | UNPRED |
| 1 | DN | Transaction Done. This bit will be set by hardware to indicate that the transmit transaction has been completed and that the transmit packet status is valid.<br>If the respective MAC DMA interrupt is enabled (see Section 6.0 "Interrupt Controller" on page 95), an interrupt will be generated when this bit is set. Done bits for all TX and RX buffers are or'ed together for this interrupt so a high level interrupt should be used.<br>This bit must be cleared explicitly by software after checking for done. This will also clear the interrupt. | R/W | UNPRED |
| 0 | EN | MAC DMA Enable. When set, this bit enables a DMA transmit transaction from the memory location designated in ADDR. | R/W | |

**Transmit Buffer Length Register**

This register contains the length of the memory buffer in bytes to be transmitted.

**macdma*m*_tx*n*len** **Offset = 0x8**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | LEN[13:0] | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:14 | — | Reserved, should be cleared. | R | 0 |
| 13:0 | LEN | Buffer Length. This field sets the length of the memory buffer (in bytes).<br>When the normal bit is set the length can only be up to 0x800 bytes.<br>When the Jumbo packets are enabled in the enable register the length can be set up to 0x2800 bytes. | R/W | 0 |

## 7.4.5 Hardware Connections

Table 7-17 shows the signals associated with the two Ethernet MAC MII interfaces.

**Table 7-17. Ethernet Signals**

| Signal | Input/Output | Description |
|---|---|---|
| **Ethernet Controller 0 (MAC0)** | | |
| N0TXCLK | I | Continuous clock input for synchronization of transmit data. 25 MHz when operating at 100 Mbps and 2.5 MHz when operating at 10 Mbps. |
| N0TXEN | O | Indicates that the data nibble on N0TXD[3:0] is valid. |
| N0TXD[3:0] | O | Nibble wide data bus synchronous to N0TXCLK. For each N0TXCLK period in which N0TXEN is asserted, TXD[3:0] will have the data to be accepted by the PHY. While N0TXEN is de-asserted the data presented on TXD[3:0] should be ignored. |
| N0RXCLK | I | Continuous clock that provides the timing reference for the data transfer from the PHY to the MAC. N0RXCLK is sourced by the PHY. The N0RXCLK shall have a frequency equal to 25% of the data rate of the received signal data stream (typically 25 MHz at 100 Mbs and 2.5 MHz at 10 Mbs). |
| N0RXDV | I | Active high. Indicates that a receive frame is in process and that the data on N0RXD[3:0] is valid. |
| N0RXD[3:0] | I | RXD[3:0] is a nibble wide data bus driven by the PHY to the MAC synchronous with N0RXCLK. For each N0RXCLK period in which N0RXDV is asserted, RXD[3:0] will transfer four bits of recovered data from the PHY to the MAC. While N0RXDV is de-asserted, RXD[3:0] will have no effect on the MAC. |
| N0CRS | I | N0CRS shall be asserted by the PHY when either transmit or receive medium is non Idle. N0CRS shall be deasserted by the PHY when both the transmit and receive medium are Idle. N0CRS is an asynchronous input. |
| N0COL | I | N0COL shall be asserted by the PHY upon detection of a collision on the medium, and shall remain asserted while the collision condition persists. N0COL is an asynchronous input. The N0COL signal is ignored by the MAC when operating in the full duplex mode. |
| N0MDC | O | N0MDC is sourced by the MAC to the PHY as the timing reference for transfer of information on the N0MDIO signal. N0MDC is an aperiodic signal that has no maximum high or low times. The N0MDC frequency is fixed at SBUS clock divided by 160. |
| N0MDIO | I/O | N0MDIO is the bidirectional data signal between the MAC and the PHY that is clocked by N0MDC. |

**Table 7-17. Ethernet Signals (Continued)**

| Signal | Input/Output | Description |
|---|---|---|
| **Ethernet Controller 1 (MAC1)** | | |
| N1TXCLK | I | Continuous clock input for synchronization of transmit data. 25 MHz when operating at 100 Mbps and 2.5 MHz when operating at 10 Mbps. |
| N1TXEN | O | Active high. Indicates that the data nibble on N1TXD[3:0] is valid.<br>Muxed with GPIO[24]. GPIO[24] is the default signal coming out of hardware reset, runtime reset, and Sleep. |
| N1TXD[3:0] | O | Nibble wide data bus synchronous to N1TXCLK. For each N1TXCLK period in which N1TXEN is asserted, TXD[3:0] will have the data to be accepted by the PHY. While N1TXEN is de-asserted the data presented on TXD[3:0] should be ignored.<br>Muxed with GPIO[28:25]. GPIO[28:25] are the default signals coming out of hardware reset, runtime reset, and Sleep. |
| N1RXCLK | I | Continuous clock that provides the timing reference for the data transfer from the PHY to the MAC. N1RXCLK is sourced by the PHY. The N1RXCLK shall have a frequency equal to 25% of the data rate of the received signal data stream (typically 25 MHz at 100 Mbps and 2.5 MHz at 10 Mbps) |
| N1RXDV | I | Active high. Indicates that a receive frame is in process and that the data on N1RXD[3:0] is valid. |
| N1RXD[3:0] | I | RXD[3:0] is a nibble wide data bus driven by the PHY to the MAC synchronous with N1RXCLK. For each N1RXCLK period in which N1RXDV is asserted, RXD[3:0] will transfer four bits of recovered data from the PHY to the MAC. While N1RXDV is de-asserted, RXD[3:0] will have no effect on the MAC. |
| N1CRS | I | N1CRS shall be asserted by the PHY when either transmit or receive medium is non Idle. N1CRS shall be deasserted by the PHY when both the transmit and receive medium are Idle. N1CRS is an isochronous input. |
| N1COL | I | N1COL shall be asserted by the PHY upon detection of a collision on the medium, and shall remain asserted while the collision condition persists. N1COL is an asynchronous input. The N1COL signal is ignored by the MAC when operating in the full duplex mode. |
| N1MDC | O | N1MDC is sourced by the MAC to the PHY as the timing reference for transfer of information on the N1MDIO signal. N1MDC is an aperiodic signal that has no maximum high or low times. The N1MDC frequency is fixed at SBUS clock divided by 160. |
| N1MDIO | I/O | N1MDIO is the bidirectional data signal between the MAC and the PHY that is clocked by N1MDC. |

MAC1 shares its pins with GPIO[28:24]; these pins must be assigned to MAC1 in order to use MAC1. Please see Section 8.3 "Primary General Purpose I/O and Pin Functionality" on page 168 for more information.

### 7.4.6 Programming Considerations

The Ethernet MAC is designed such that the application could use a pool of memory buffers for both the transmit and receive functions.

The lowest level device driver would respond to the MAC DMA interrupt and swap out the filled DMA buffers for those that are empty for the receive case. For the transmit case the driver should provide ready to transmit buffers to the DMA while reclaiming empty buffers. Four transmit and receive DMA buffers are allocated for each MAC to allow for latency to service the lowest level MAC DMA interrupt.

At the next level in software the device driver can parse the valid data out of the frame for receive, or build the frame for transmit. The number of memory buffers needed in the pool will depend on how fast the parsing can occur for worst case receive bursts, and any minimum transmit latency requirements.

From this level the application or protocol stack can take the data and apply it as needed.

### 7.4.7 Initialization

This section demonstrates the functional requirements for getting the MAC running. This assumes that the programmer has already performed the Au1500 bringup.

1) Interrupt Controller - a high level interrupt should be used as the interrupt is triggered with an OR'ing of the DN (Done) bits.
2) DMA Controller Setup
3) MAC Registers - It is the system designer's responsibility to set up addresses.
4) Memory - Depending on how the system is built, there could be a pool of memory buffers which can be used for parsing and building of frames. Individual buffers would be swapped in and out of the 4 active receive and transmit DMA buffers as needed. This strategy would require some sort of minimal memory management within the Ethernet driver to insure chronology of Ethernet frames.

The following is a transmit example in a basic form. Typically this would be split between an interrupt handler and another higher layer.

1) Construct Frame
2) Set length in **macdma*m*_tx*n*len** register
3) Set address of memory buffer and enable transmit. During this time the physical memory buffer and address and length registers should not be disrupted or transmit contents will be undefined.
4) Wait for done. This can be done by waiting for the interrupt handler or polling the done signal in the **macdma*m*_tx*n*addr** register.
5) Read status. Its validity is signaled by the reception of the done signal.

The following is a basic receive example:

1) Enable all receive buffers with four different memory buffer addresses.
2) Wait for interrupt. Conversely the done bit could be polled. During this time the physical memory buffer and address registers should not be disrupted or receive contents will be undefined.
3) Replace all full buffers with empty memory buffers.
4) Read Status for full buffers.
5) Parse frames.

## 7.5 UART Interfaces

The Au1500 contains two UART interfaces. Each UART has the following features:

- 5 - 8 Data Bits
- 1 - 2 Stop Bits
- Even, Odd, Mark, or No Parity
- 16-byte Transmit and Receive FIFOs
- Interrupts for Receive FIFO Full, Half Full, and Not Empty
- Interrupts for Transmit FIFO Empty
- False Start Bit Detection
- Full Modem Control Signals on UART3
- Capable of speeds up to 1.5 Mbps to enable connections with Bluetooth and other peripherals through a UART interface
- Similar to personal computer industry standard 16550 UART

### 7.5.1 Programming Model

Each UART is controlled by a register block. Table 7-18 lists the base address for each UART register block.

**Table 7-18. UART Register Base Addresses**

| Name | Physical Base Address | KSEG1 Base Address |
|---|---|---|
| uart0_base | 0x0 1110 0000 | 0xB110 0000 |
| uart3_base | 0x0 1140 0000 | 0xB140 0000 |

UART0 and UART3 are capable of being used with DMA. See Section 5.0 "DMA Controller" on page 87 for more information.

### 7.5.2 UART Registers

Each register block contains the registers listed in Table 7-19.

**Table 7-19. UART Registers**

| Offset (Note 1) | Register Name | Description |
|---|---|---|
| 0x0000 | uart_rxdata | Received Data FIFO |
| 0x0004 | uart_txdata | Transmit Data FIFO |
| 0x0008 | uart_inten | Interrupt Enable Register |
| 0x000C | uart_intcause | Pending Interrupt Cause Register |
| 0x0010 | uart_fifoctrl | FIFO Control Register |
| 0x0014 | uart_linectrl | Line Control Register |
| 0x0018 | uart_mdmctrl | Modem Line Control Register (UART3 only) |
| 0x001C | uart_linestat | Line Status Register |
| 0x0020 | uart_mdmstat | Modem Line Status Register (UART3 only) |
| 0x0024 | uart_autoflow | Automatic Hardware Flow Control (UART3 only) |
| 0x0028 | uart_clkdiv | Baud Rate Clock Divider |
| 0x0100 | uart_enable | Module Enable Register |

Note 1. See Table 7-18 for base address.

#### 7.5.2.1 Received Data FIFO

The **uart_rxdata** register contains the next entry in the received data FIFO. This register is read only.

**uart_rxdata - Received Data FIFO** **Offset = 0x0000**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | RXDATA | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:8 | — | Reserved, should be cleared. | R | 0 |
| 7:0 | RXDATA | Receive Data | R | 0 |

#### 7.5.2.2 Transmit Data FIFO

The **uart_txdata** register provides access to the transmit data FIFO. This register is write only.

**uart_txdata - Transmit Data FIFO** **Offset = 0x0004**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | TXDATA | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:8 | — | Reserved, should be cleared. | R | 0 |
| 7:0 | TXDATA | Transmit Data | R | 0 |

#### 7.5.2.3 Interrupt Enable Register

The **uart_inten** register contains bits which enable interrupts under certain operational conditions.

**uart_inten - Interrupt Enable Register** **Offset = 0x0008**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | MIE | LIE | TIE | RIE |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:4 | — | Reserved, should be cleared. | R | 0 |
| 3 | MIE | Modem Status Interrupt Enable (UART3 only). When the MIE bit is set an interrupt is generated when changes occur in the state of the optional modem control signals available with UART3.<br><br>**System Note**: For systems that use the UART3 interface but do *not* use the optional modem control signals (**sys_pinfunc**[UR3]=0), the modem status interrupts must be disabled (MIE=0) to avoid false UART3 interrupts when using GPIO[9], GPIO[10], GPIO[11], or GPIO[12] as a general-purpose system input. | R/W | 0 |
| 2 | LIE | Line Status Interrupt Enable. When the LIE bit is set an interrupt is generated when errors (overrun, framing, stop bits) or break conditions occur. | R/W | 0 |
| 1 | TIE | Transmit Interrupt Enable. When the TIE bit is set an interrupt is generated when the transmit FIFO is not full. | R/W | 0 |
| 0 | RIE | Receive Interrupt Enable. When the RIE bit is set the UART will generate an interrupt on received data ready (*DR* bit in the **uart_linestat** register) or a character time out. | R/W | 0 |

### 7.5.2.4 Interrupt Cause Register

The **uart_intcause** register contains information about the cause of the current interrupt.

**uart_intcause - Interrupt Cause Register** **Offset = 0x000c**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | IID | | IP |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:4 | — | Reserved, should be cleared. | R | 0 |
| 3:1 | IID | Interrupt Identifier. The IID field identifies the highest priority current interrupt condition. Table 7-20 "Interrupt Cause Encoding" lists the priorities and encodings of each interrupt condition. | R | 0 |
| 0 | IP | No Interrupt Pending.<br>0 An interrupt is pending.<br>1 No interrupts are pending. | R | 1 |

Table 7-20 contains information about the interrupt cause encoding.

**Table 7-20. Interrupt Cause Encoding**

| IID | Priority | Type | Source |
|---|---|---|---|
| 0 | 5 (lowest) | Modem Status | DD, TRI, DR or DC of **uart_mdmstat** |
| 1 | 4 | Transmit Buffer Available | TT of **uart_linestat** |
| 2 | 3 | Receive Data Available | The receive FIFO having greater than RFT (of **uart_fifoctrl**) bytes in it if FIFOs are enabled.<br>DR of **uart_linestat** if FIFOs are disabled. |
| 3 | 1 (highest) | Receive Line Status | OE, PE, FE, BI in **uart_linestat** register |
| 4 | | Reserved | |
| 5 | | Reserved | |
| 6 | 2 | Character Time Out | Character has been in receive FIFO for 0x300 UART clocks (set by **uart_clkdiv**) |
| 7 | | Reserved | |

### 7.5.2.5 FIFO Control Register

The **uart_fifoctrl** register provides control of character buffering options.

**uart_fifoctrl - FIFO Control Register** **Offset = 0x0010**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | RFT | | TFT | | MS | TR | RR | FE |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:8 | — | Reserved, should be cleared. | R | 0 |
| 7:6 | RFT | Receive FIFO Threshold. A receive threshold interrupt is generated when the number of characters in the receiver FIFO is greater than or equal to the trigger level listed below:<br>00 Trigger depth = 1<br>01 Trigger depth = 4<br>10 Trigger depth = 8<br>11 Trigger depth = 14<br>If using DMA it is important that the receive FIFO threshold and transmit FIFO threshold are the same and programmed consistently with the transfer size for the DMA channel being used. See Section 5.0 "DMA Controller" on page 87 for more information. | R/W | 0 |
| 5:4 | TFT | Transmit FIFO Threshold. A transmit threshold interrupt is generated if the number of valid characters contained in the transmit FIFO is less than or equal to the trigger depth. The encoding of trigger depth for each value of TFT is shown below:<br>00 Trigger depth = 0<br>01 Trigger depth = 4<br>10 Trigger depth = 8<br>11 Trigger depth = 12<br>If using DMA it is important that the receive FIFO threshold and transmit FIFO threshold are the same and programmed consistently with the transfer size for the DMA channel being used. See Section 5.0 "DMA Controller" on page 87 for more information. | R/W | 0 |
| 3 | MS | Mode Select. If the MS bit is clear interrupts are generated by the receiver when any data is available and by the transmitter when there is no data to transmit. Setting the MS bit causes interrupts to be generated based on FIFO threshold levels. | R/W | 0 |
| 2 | TR | Transmitter Reset. Writing a one to the TR bit will clear the transmit FIFO and reset the transmitter. The transmit shift register is not cleared. | R/W | 0 |
| 1 | RR | Receiver Reset. Writing a one to the RR bit will clear the receiver FIFO and reset the receiver. The receiver shift register is not cleared. | R/W | 0 |
| 0 | FE | FIFO Enable. The FE bit enables the 16 byte FIFOs on transmit and receive. When the FE bit is clear both FIFOs will have an effective depth of 1 byte. | R/W | 0 |

### 7.5.2.6 Line Control Register

The **uart_linectrl** register provides control over the data format and parity options.

**uart_linectrl - Line Control Register** **Offset = 0x0014**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | SB | PAR | | PE | ST | WLS | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:7 | — | Reserved, should be cleared. | R | 0 |
| 6 | SB | Send Break. Setting the SB bit will force the transmitter output to zero. | R/W | 0 |
| 5:4 | PAR | Parity Select. Selects the parity encoding for the transmitter and receiver.<br>00 Odd parity<br>01 Even parity<br>10 Mark parity<br>11 Zero parity | R/W | 0 |
| 3 | PE | Parity Enable. If the PE bit is clear parity will not be sent or expected. If the PE bit is set parity is selected according to the PAR field. | R/W | 0 |
| 2 | ST | Stop Bits. If the ST bit is clear one stop bit is sent and expected. Setting the ST bit selects 1.5 stop bits for 5 bit characters and 2 stop bits for all other character lengths. | R/W | 0 |
| 1:0 | WLS | Word Length Select. The WLS field selects the number of data bits in each character. The number of bits is WLS+5. | R/W | 0 |

### 7.5.2.7 Modem Control Register

The **uart_mdmctrl** register allows the state of the output modem control signals to be set. The external modem signals are only available on UART3.

**uart_mdmctrl - Modem Control Register** **Offset = 0x0018**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | LB | I1 | I0 | RT | DT |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:5 | — | Reserved, should be cleared. | R | 0 |
| 4 | LB | Loopback.<br>0 No loopback (normal operation)<br>1 Enable loopback for self-test. Establish the internal connections shown below:<br>Output Signal — Looped Back To<br>TXD — RXD<br>DTR# — DSR#<br>RTS# — CTS#<br>I0# — RI#<br>I1# — DCD# | R/W | 0 |
| 3 | I1 | Internal Line 1 State. When the I1 bit is set the internal I1# line for this port is driven low. This can be used in loopback mode. | R/W | 0 |
| 2 | I0 | Internal Line 0 State. When the I0 bit is set the external I0# line for this port is driven low.This can be used in loopback mode. | R/W | 0 |
| 1 | RT | Request To Send. When the RT bit is set the external RTS# line for this port is driven low.<br>**Note:** This bit has no effect if **uart_autoflow**[AE] is set. | R/W | 0 |
| 0 | DT | Data Terminal Ready. When the DT bit is set the external DTR# line for this port is driven low. | R/W | 0 |

### 7.5.2.8 Line Status Register

The **uart_linestat** register reflects the state of the interface.

Bits in this register are set when the listed condition and cleared when this register is read.

**uart_linestat - Line Status Register** **Offset = 0x001C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | RF | TE | TT | BI | FE | PE | OE | DR |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:8 | — | Reserved, should be cleared. | R | 0 |
| 7 | RF | Receiver FIFO Contains Error. This bit is set when one of the characters in the receive FIFO contains a parity error, framing error, or break indication. | R | 0 |
| 6 | TE | Transmit Shift Register Empty. This bit is set when the transmit shift register is empty and there are no more characters in the FIFO. | R | 0 |
| 5 | TT | Transmit Threshold. This bit is set when the transmitter FIFO depth is less than or equal to the value of the TFT field in the FIFO control register.<br>When FIFOs are not enabled this bit is set when the transmitter data register is empty | R | 0 |
| 4 | BI | Break Indication. This bit is set if a break is received. When a break is detected a single zero character is received. The BI bit is valid when the zero character is at the top of the receive FIFO. This bit must be cleared with a read to **uart_linestat** before more characters are received. | R | 0 |
| 3 | FE | Framing Error. The FE bit is set when a valid stop bit is not detected. This bit reflects the state of the character at the top of the receive FIFO. The FE bit is cleared by a read to **uart_linestat**. | R | 0 |
| 2 | PE | Parity Error. The PE bit is set when the received character at the top of the FIFO contains a parity error. This bit is cleared by reading **uart_linestat**. | R | 0 |
| 1 | OE | Overrun Error. The OE bit is set when a receiver overrun occurs. This bit is cleared when **uart_linestat** is read. | R | 0 |
| 0 | DR | Data Ready. The DR bit is set when the receive FIFO contains valid characters. | R | 0 |

### 7.5.2.9 Modem Status Register

The **uart_mdmstat** register reflects the state of the external modem signals. Reading this register will clear any delta indications and the corresponding interrupt. The external modem signals are optional and are present only on UART3.

**uart3_mdmstat - Modem Status Register** **Offset = 0x0020**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | CD | RI | DS | CT | DD | TRI | DR | DC |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:8 | — | Reserved, should be cleared. | R | 0 |
| 7 | CD | Data Carrier Detect. The CD bit reflects the status of the external DCD# pin. | R | 0 |
| 6 | RI | Ring Indication. The RI bit reflects the status of the external RI# pin. | R | 0 |
| 5 | DS | Data Set Ready. The DS bit reflects the status of the external DSR# pin. | R | 0 |
| 4 | CT | Clear To Send. The CT bit reflects the status of the external CTS# pin. | R | 0 |
| 3 | DD | Delta DCD. The DD bit is set when a change occurs in the state of the external DCD# pin. | R | 0 |
| 2 | TRI | Terminate Ring Indication. The TRI bit is set when a positive edge occurs in the state of the external RI# pin. | R | 0 |
| 1 | DR | Delta DSR. The DR bit is set when a change occurs in the state of the external DSR# pin. | R | 0 |
| 0 | DC | Delta CTS. The DC bit is set when a change occurs in the state of the external CTS# pin. | R | 0 |

### 7.5.2.10 Automatic Hardware Flow Control Register

The **uart_autoflow** register controls automatic hardware flow control using modem control signals CTS# and RTS#. Upon enabling this mode, internal logic controls the output signal RTS# based upon the data register state and threshold levels. The internal logic asserts RTS (low) to request data until the internal receive FIFO reaches its preset threshold. In this mode RTS# cannot be controlled with the **uart_mdmctrl**[RT] bit. The input signal CTS# controls the transmission of data by loading the transmit shift register from the data register only while CTS# is asserted (low). Once the transmit shift register is loaded with data, it sends the entire character regardless of the CTS# signal state.

**uart_autoflow - Automatic Hardware Flow Control Register** **Offset = 0x0024**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | AE |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:1 | — | Reserved, should be cleared. | R | 0 |
| 0 | AE | Autoflow Enable. Setting this bit enables automatic hardware flow control on UART3. Enabling this mode overrides software control of the signals. | R/W | 0 |

### 7.5.2.11 Clock Divider Register

The **uart_clkdiv** register contains the divider used to generate the baud rate clock. The input to the UART clock divider is the internal peripheral bus clock, which is derived from the System Bus (SBUS) clock; see Figure 8-1 "Clocking Topology" on page 154.

**uart_clkdiv - Clock Divider Register** **Offset = 0x0028**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | CLKDIV | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | — | Reserved. | — | — |
| 15:0 | CLKDIV | Clock Divider. The baud rate of the interface is computed as follows:<br>Baud rate = CPU / (SD * 2 * CLKDIV * 16)<br>CPU: CPU clock<br>SD: SBUS divider<br>See Section 8.4 "Power Management" on page 172 for information on changing SD. | R/W | 0x1 |

### 7.5.2.12 UART Enable

The **uart_enable** register controls reset and clock enable to the UART

The correct routine for bringing the USB Device out of reset is as follows:

1) Set the CE bit to enable clocks.
2) Set the E bit to enable the peripheral.

**uart_enable - UART Enable Register** **Offset = 0x0100**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | E | CE |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:2 | — | Reserved, should be cleared. | R | 0 |
| 1 | E | Enable. When the E bit is clear the entire module is held in reset. After enabling clocks, this bit should be set to enable normal operation. | R/W | 0 |
| 0 | CE | Clock Enable. When the CE bit is clear the module clock source is inhibited. This can be used to place the module in a low power Stand-by state. The CE bit should be set before the module is enabled for proper bringup. | R/W | 0 |

### 7.5.3 Hardware Considerations

The UART signals are listed in Table 7-21. For changing pin functionality please refer to the **sys_pinfunc** register in Section 8.3 "Primary General Purpose I/O and Pin Functionality" on page 168.

**Table 7-21. UART Signals**

| Signal | Input/Output | Definition |
|---|---|---|
| **UART0** | | |
| U0TXD | O | UART0 Transmit. Muxed with GPIO[20]. GPIO[20] is the default signal coming out of hardware reset, runtime reset, and Sleep. |
| U0RXD | I | UART0 Receive. |
| **UART3** | | |
| U3TXD | O | UART3 Transmit. Muxed with GPIO[23]. GPIO[23] is the default signal coming out of hardware reset, runtime reset, and Sleep. |
| U3RXD | I | UART3 Receive. |
| U3CTS# | I | Clear to Send (optional). Muxed with GPIO[9]. GPIO[9] is the default signal coming out of hardware reset, runtime reset, and Sleep.<br>**System Note**: For systems that use the UART3 interface without the optional modem control signals (**sys_pinfunc**[UR3]=0), the modem status interrupts must be disabled (**uart3_inten**[MIE]=0) to avoid false UART3 interrupts when using GPIO[9], GPIO[10], GPIO[11], or GPIO[12] as an input. |
| U3DSR# | I | Data Set Ready (optional). Muxed with GPIO[10]. GPIO[10] is the default signal coming out of hardware reset, runtime reset, and Sleep.<br>**System Note**: For systems that use the UART3 interface without the optional modem control signals (**sys_pinfunc**[UR3]=0), the modem status interrupts must be disabled (**uart3_inten**[MIE]=0) to avoid false UART3 interrupts when using GPIO[9], GPIO[10], GPIO[11], or GPIO[12] as an input. |
| U3DCD# | I | Data Carrier Detect (optional). Muxed with GPIO[11]. GPIO[11] is the default signal coming out of hardware reset, runtime reset, and Sleep.<br>**System Note**: For systems that use the UART3 interface without the optional modem control signals (**sys_pinfunc**[UR3]=0), the modem status interrupts must be disabled (**uart3_inten**[MIE]=0) to avoid false UART3 interrupts when using GPIO[9], GPIO[10], GPIO[11], or GPIO[12] as an input. |
| U3RI# | I | Ring Indication (optional). Muxed with GPIO[12]. GPIO[12] is the default signal coming out of hardware reset, runtime reset, and Sleep.<br>**System Note**: For systems that use the UART3 interface without the optional modem control signals (**sys_pinfunc**[UR3]=0), the modem status interrupts must be disabled (**uart3_inten**[MIE]=0) to avoid false UART3 interrupts when using GPIO[9], GPIO[10], GPIO[11], or GPIO[12] as an input. |
| U3RTS# | O | Request to Send (optional). Muxed with GPIO[13]. GPIO[13] is the default signal coming out of hardware reset, runtime reset, and Sleep. |
| U3DTR# | O | Data Terminal Ready (optional). Muxed with GPIO[14]. GPIO[14] is the default signal coming out of hardware reset, runtime reset, and Sleep. |

## 7.6 Secondary General Purpose I/O

The Au1500 processor contains two GPIO blocks (primary and secondary). This section describes the programming model of the *secondary* GPIO block which corresponds to signals labeled GPIO[200] through GPIO[215]. (For a description of the primary GPIO block refer to Section 8.3 "Primary General Purpose I/O and Pin Functionality" on page 168 in the system control block description.)

GPIO[200] is configured at reset to output a low logic level. This pin can be used as a PCI reset output (PCI_RSTO#) if it is required that the processor generate PCI reset. All other secondary GPIOs are configured as inputs on reset.

### 7.6.1 GPIO2 Programming Model

The secondary GPIO (GPIO2) logic block is controlled by a register block referenced from the base address described in Table 7-22.

**Table 7-22. GPIO2 Register Base Addresses**

| Name | Physical Base Address | KSEG1 Base Address |
|---|---|---|
| gpio2_base | 0x0 1170 0000 | 0xB170 0000 |

### 7.6.2 GPIO2 Registers

The secondary GPIO register block is shown in Table 7-23.

**Table 7-23. GPIO2 Registers**

| Offset (Note 1) | Register Name | Description |
|---|---|---|
| 0x0000 | gpio2_dir | GPIO2 Direction |
| 0x0004 | — | Reserved |
| 0x0008 | gpio2_output | GPIO2 Data Output |
| 0x000C | gpio2_pinstate | GPIO2 Pin State |
| 0x0010 | gpio2_inten | GPIO2 Interrupt Enable (for GPIO[215:208]) |
| 0x0014 | gpio2_enable | GPIO2 Enable |

Note 1. See Table 7-22 for base address.

#### 7.6.2.1 Direction Register

The **gpio2_dir** register controls the direction of each GPIO2 signal. Note that this register only controls the output enable for the output buffer. Clearing a bit in this register disables the output for the corresponding pin making it possible to read an externally driven input. Output enable control can also be used to emulate an open drain driver.

**gpio2_dir - Direction Register** **Offset = 0x0000**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | DIR | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | — | Reserved, should be cleared. | R | 0 |
| 15:0 | DIR | Direction Control. Each bit controls the I/O direction of one GPIO signal in the secondary block. Bits 15:0 correspond to GPIO[215:200].<br>0 Pin is an input (output disabled).<br>1 Pin is an output.<br>Note that the GPIO[200] default direction is out because GPIO[200] is designed to act as a PCI reset output signal (PCI_RSTO#) if needed. | R/W | 0x0001 |

#### 7.6.2.2 Data Output Register

The **gpio2_output** register controls the output data for the secondary GPIOs. Data bits 15:0 are output to the corresponding GPIO when the enable bit is set for that bit during a write to this register. For example, to output a '1' on GPIO[200] and a '0' on GPIO[201] without changing the output of any other GPIOs, write the value 0x00030001 to **gpio2_output**.

**gpio2_output - Data Output Register** **Offset = 0x0008**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ENA | | | | | | | | | | | | | | | | DATA | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | ENA[15:0] | Data Output Write Enable. ENA[15:0] corresponds to DATA[15:0]. Note that ENA is write-only and should be ignored on reads.<br>0 Disable modifications to corresponding bit in DATA[15:0].<br>1 Enable modifications to corresponding bit in DATA[15:0]. | W | 0x0001 |
| 15:0 | DATA[15:0] | Output Data. DATA[15:0] corresponds to GPIO[215:200]. The DATA bit values are reflected in the corresponding GPIO output signal level.<br>When modifying a bit in DATA[15:0], the corresponding bit in ENA[15:0] must be set to allow the write. This mechanism allows individual data bits to be modified without affecting DATA[15:0] as a whole. | R/W | 0 |

#### 7.6.2.3 Pin State Register

The **gpio2_pinstate** register reflects the current state of the corresponding secondary GPIO pin.

**gpio2_pinstate - Pin State** **Offset = 0x000C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | | | | | | | | | | | | | | | | DATA | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | — | These bits are reserved and will be read 0. | R | 0 |
| 15:0 | DATA | Current Pin State for GPIO[215:200] | R | 0 |

### 7.6.2.4 Interrupt Enable Register

The **gpio2_inten** register contains bits which enable interrupts under certain operational conditions. Note that **gpio2_inten** applies only to interrupts on GPIO[215:208]. The GPIO[215:208] signals are OR'd together to create *one* interrupt source (source number 31 on interrupt controller 1), as shown in Figure 7-2. (The GPIO[207:200] signals can be used as *independent* interrupt sources.) See Section 6.0 "Interrupt Controller" on page 95 for more information on how to program interrupts.

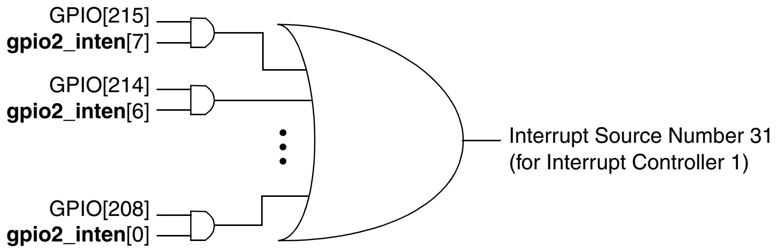


**Figure 7-2. Logic for Interrupt Source Number 31 on Interrupt Controller 1**

**gpio2_inten - Interrupt Enable Register** **Offset = 0x0010**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | EN | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:8 | — | Reserved, should be cleared. | R | 0 |
| 7:0 | EN | Interrupt enable bits [7:0] correspond to GPIO[215:208].<br>Setting a bit enables the signal's OR'd contribution to interrupt source number 31 (on interrupt controller 1). | R/W | 0 |

### 7.6.2.5 Enable Register

The **gpio2_enable** register controls the clocks and reset to the secondary GPIO block.

**gpio2_enable - Enable Register** **Offset = 0x0014**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | MR | CE |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:2 | — | Reserved, should be cleared. | R | 0 |
| 1 | MR | Module Reset. When this bit is set the module is held in reset. | R/W | 1 |
| 0 | CE | Clock Enable. When this bit is clear the module clocks are disabled. | R/W | 0 |

# 8 System Control

The Au1500 processor contains a robust system control strategy that includes the means to control the following:

- Clocking (See Section 8.1 "Clocks" on page 154.)
- Time of Year and Real Time Clock counters (See Section 8.2 "Time of Year Clock and Real Time Clock" on page 163.)
- GPIO control (See Section 8.3 "Primary General Purpose I/O and Pin Functionality" on page 168.)
- Power management (See Section 8.4 "Power Management" on page 172.)

All registers in the system control block are located off of the base address shown in Table 8-1.

**Table 8-1. System Control Block Base Address**

| Name | Physical Base Address | KSEG1 Base Address |
|---|---|---|
| sys_base | 0x0 1190 0000 | 0xB190 0000 |

The registers in the system control block are affected differently by events such as power-on hardware reset, Sleep and runtime reset (see Section 9.0 "Power-up, Reset and Boot" on page 180 for a discussion on the different reset types). Each register is documented with how it will be affected by the different system states. Care should be taken by the system designer to observe what registers will and will not revert to defaults when the different events occur.

## 8.1 Clocks

The Au1500 processor supports two oscillator inputs: 12 MHz and 32.768 KHz. This section documents the clock domains driven directly and indirectly by the 12 MHz input. The 32.768 KHz clock input drives the two programmable counters intended for use as a real time clock (RTC) and time of year clock (TOY). The programmable counters are documented in Section 8.2 "Time of Year Clock and Real Time Clock" on page 163. (See Section 12.10 "Crystal Specifications" on page 234 for the specifications of both crystals.)

The Au1500 processor contains two PLLs driven by the 12 MHz oscillator and a clocking block from which the following are derived:

- CPU Clock
- Core Cycle Counter register clocked by the CPU Clock
- System Bus (SBUS) Clock
- Peripheral Bus Clock
- SDRAM Bus Clock
- Programmable Clocks needed by certain peripherals
- Programmable Clocks EXTCLK[1:0] for external use (provided on pins shared with the GPIO[3:2] signals)

Figure 8-1 shows the basic clocking topology and the relationship between the CPU Clock, the SBUS clock and the Peripheral Clock. As shown, the SBUS frequency is derived by dividing the CPU Clock by the value SD programmed in the **sys_powerctrl** register. (See Section 8.4 "Power Management" on page 172 for the **sys_powerctrl** register definition.) The Peripheral Bus clock and the SDRAM bus are fixed at the SBUS frequency divided by 2. Figure 8-1 also shows the peripheral blocks driven by clock sources derived from the programmable clock generator logic (as described in Section 8.1.2 "Clock Generation" on page 155).

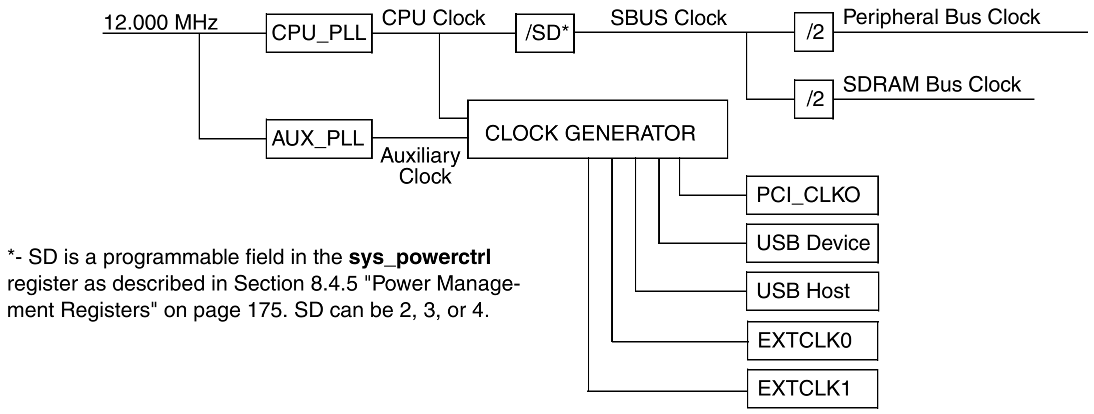

12.000 MHz
CPU_PLL
CPU Clock
/SD*
SBUS Clock
/2
Peripheral Bus Clock
/2
SDRAM Bus Clock
AUX_PLL
Auxiliary Clock
CLOCK GENERATOR
PCI_CLKO
USB Device
USB Host
EXTCLK0
EXTCLK1
*- SD is a programmable field in the **sys_powerctrl** register as described in Section 8.4.5 "Power Management Registers" on page 175. SD can be 2, 3, or 4.

**Figure 8-1. Clocking Topology**

### 8.1.1 Clock Register Descriptions

The clock manager registers and their associated offsets are listed in Table 8-2.

**Table 8-2. Clock Generation Registers**

| Offset (Note 1) | Register Name | Description | Reset Type |
|---|---|---|---|
| 0x0020 | sys_freqctrl0 | Controls (source, enable, and divider) frequency generators 0, 1, and 2 | Hardware |
| 0x0024 | sys_freqctrl1 | Controls (source, enable, and divider) frequency generators 3, 4, and 5 | Hardware |
| 0x0028 | sys_clksrc | Controls (source and divider) the derived clocks | Hardware |
| 0x0060 | sys_cpupll | Changes CPU PLL frequency | Hardware |
| 0x0064 | sys_auxpll | Changes Auxiliary PLL frequency | Hardware & Runtime |

Note 1. See Table 8-1 on page 153 for base address.

### 8.1.2 Clock Generation

This section documents registers for the clock generation block which provides clocks to some peripheral devices and as well as two externally available clocks. The clock generation subsystem is split into two sets of distinct blocks which allows up to six distinct frequencies to drive up to six clock sources. Figure 8-2 on page 156 shows a logical representation of one of the six identical frequency generators and how the six frequency sources are mapped to one of the six identical internal clock sources. The names in the figure correspond to the bit names in the control registers. Figure 8-3 on page 156 shows a pictorial representation of the relationship between the frequency generator blocks to the clock source blocks.

Each peripheral has clock restrictions as follows. If these restrictions are not met, the peripheral will not operate correctly.

The USB Device Clock must be programmed to 48 MHz.

The USB Host Clocks must be programmed to 48 MHz.

The EXTCLK[1:0] clocks can be programmed for system use.

Note that the EXTCLK[1:0] clocks have a maximum frequency rating of ($F_{max}$ / 16), *where* $F_{max}$ is the maximum frequency rating for the part. For example, for a 400 MHz part be sure the EXTCLK[1:0] clocks are programmed to run at no more than 25 MHz. (See also Section 12.9 "External Clock Specifications" on page 233.)

Note also that the EXTCLK[1:0] clocks are multiplexed signals and require programming of the **sys_pinfunc** register (see Section 8.3.1.1 "Pin Function" on page 168) as follows:

- EXTCLK0 shares a pin with GPIO[2]. If EXTCLK0 is to be used, **sys_pinfunc**[EX0] must be set to allow the clock to drive this pin. In addition, **sys_pinfunc**[CS] must be cleared.
- EXTCLK1 shares a pin with GPIO[3]. If EXTCLK1 is to be used, **sys_pinfunc**[EX1] must be set to allow the clock to drive this pin.

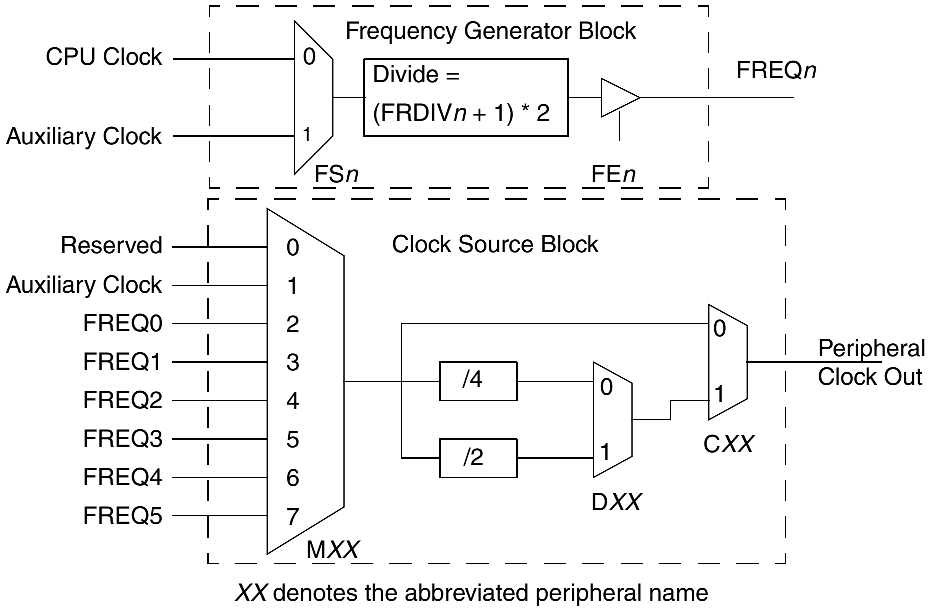


**Figure 8-2. Frequency Generator and Clock Source Block Diagram**

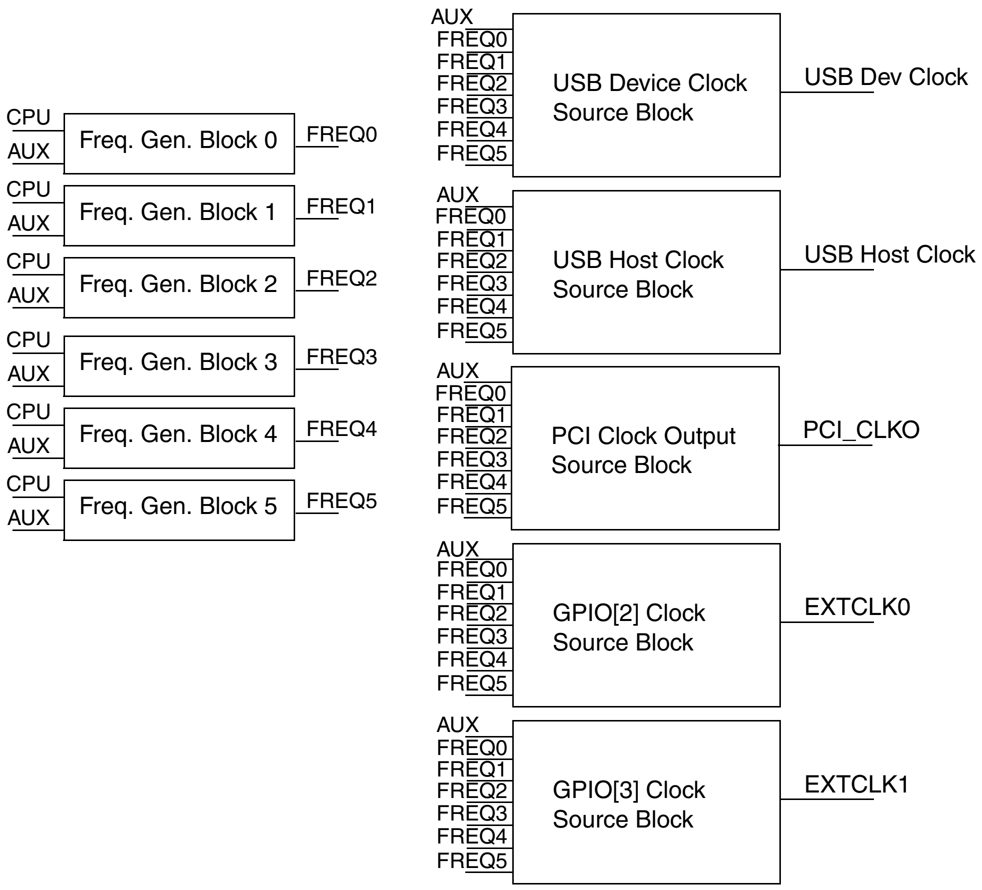


**Figure 8-3. Frequency Generator and Clock Source Mapping**

### 8.1.2.1 Frequency Control 0

This register controls the frequency generator block for output frequencies 0, 1, and 2. This register will reset to defaults only on a hardware reset. During a runtime reset and during Sleep this register will retain its value.

**sys_freqctrl0** **Offset = 0x0020**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | FRDIV2[7:0] | | | | | | | | FE2 | FS2 | FRDIV1[7:0] | | | | | | | | FE1 | FS1 | FRDIV0[7:0] | | | | | | | | FE0 | FS0 |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:30 | — | Reserved, should be cleared. | R | 0 |
| 29:22 | FRDIV2 | Divider for Frequency Generator 2. The frequency divider is (FRDIV + 1) * 2, *where* FRDIV is the value programmed in this field. | R/W | 0 |
| 21 | FE2 | Frequency Generator Output Enable 2.<br>0 Disable output.<br>1 Enable output. | R/W | 0 |
| 20 | FS2 | Frequency Generator 2 Source.<br>0 CPU Core clock.<br>1 Auxiliary clock. | R/W | 0 |
| 19:12 | FRDIV1 | Divider for Frequency Generator 1. The frequency divider is (FRDIV + 1) * 2, *where* FRDIV is the value programmed in this field. | R/W | 0 |
| 11 | FE1 | Frequency Generator 1 Output Enable.<br>0 Disable output.<br>1 Enable output . | R/W | 0 |
| 10 | FS1 | Frequency Generator 1 Source.<br>0 CPU Core clock.<br>1 Auxiliary clock. | R/W | 0 |
| 9:2 | FRDIV0 | Divider for Frequency Generator 0. The frequency divider is (FRDIV + 1) * 2, *where* FRDIV is the value programmed in this field. | R/W | 0 |
| 1 | FE0 | Frequency Generator 0 Output Enable.<br>0 Disable output.<br>1 Enable output . | R/W | 0 |
| 0 | FS0 | Frequency Generator 0 Source.<br>0 CPU Core clock.<br>1 Auxiliary clock. | R/W | 0 |

### 8.1.2.2 Frequency Control 1

This register controls the frequency generator block for output frequencies 3, 4, and 5. This register will reset to defaults only on a hardware reset. During a runtime reset and during Sleep this register will retain its value.

**sys_freqctrl1** **Offset = 0x0024**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | FRDIV5[7:0] | | | | | | | | FE5 | FS5 | FRDIV4[7:0] | | | | | | | | FE4 | FS4 | FRDIV3[7:0] | | | | | | | | FE3 | FS3 |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:30 | — | Reserved, should be cleared. | R | 0 |
| 29:22 | FRDIV5 | Divider for Frequency Generator 5. The frequency divider is (FRDIV + 1) * 2, *where* FRDIV is the value programmed in this field. | R/W | 0 |
| 21 | FE5 | Frequency Generator 5 Output Enable.<br>0 Disable output.<br>1 Enable output . | R/W | 0 |
| 20 | FS5 | Frequency Generator 5 Source.<br>0 CPU Core clock.<br>1 Auxiliary clock. | R/W | 0 |
| 19:12 | FRDIV4 | Divider for Frequency Generator 4. The frequency divider is (FRDIV + 1) * 2, *where* FRDIV is the value programmed in this field. | R/W | 0 |
| 11 | FE4 | Frequency Generator 4 Output Enable.<br>0 Disable output.<br>1 Enable output . | R/W | 0 |
| 10 | FS4 | Frequency Generator 4 Source.<br>0 CPU Core clock.<br>1 Auxiliary clock. | R/W | 0 |
| 9:2 | FRDIV3 | Divider for Frequency Generator 3. The frequency divider is (FRDIV + 1) * 2, *where* FRDIV is the value programmed in this field. | R/W | 0 |
| 1 | FE3 | Frequency Generator 3 Output Enable.<br>0 Disable output.<br>1 Enable output . | R/W | 0 |
| 0 | FS3 | Frequency Generator 3 Source.<br>0 CPU Core clock.<br>1 Auxiliary clock. | R/W | 0 |

### 8.1.2.3 Clock Source Control

This register controls the clock source for all output clocks. This register will reset to defaults only on a hardware reset. During a runtime reset and during Sleep this register will retain its value.

**sys_clksrc** **Offset = 0x0028**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ME1[2:0] | | | DE1 | CG3 | ME0[2:0] | | | DE0 | CE0 | MPC[2:0] | | | DPC | CPC | MUH[2:0] | | | DUH | CUH | MUD[2:0] | | | DUD | CUD | ▨ | ▨ | ▨ | ▨ | ▨ |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | Read/Write | Default |
|---|---|---|---|---|
| 31:30 | — | Reserved, should be cleared. | R | 0 |
| 29:27 | ME1 | EXTCLK1 Clock Mux Input Select. See Table 8-3 on page 160. | R/W | 000 |
| 26 | DE1 | EXTCLK1 Clock Divider Select.<br>0 Divide by 4.<br>1 Divide by 2. | R/W | 0 |
| 25 | CE1 | EXTCLK1 Clock Select.<br>0 Clock is taken directly from mux. (The divider select bit DE1 has no effect.)<br>1 Clock is taken from 2/4 divider. | R/W | 0 |
| 24:22 | ME0 | EXTCLK0 Clock Mux Input Select. See Table 8-3 on page 160. | R/W | 000 |
| 21 | DE0 | EXTCLK0 Clock Divider Select.<br>0 Divide by 4.<br>1 Divide by 2. | R/W | 0 |
| 20 | CE0 | EXTCLK0 Clock Select.<br>0 Clock is taken directly from mux. (The divider select bit DE0 has no effect.)<br>1 Clock is taken from 2/4 divider. | R/W | 0 |
| 19:17 | MPC | PCI Clock Mux Input Select. See Table 8-3 on page 160. | R/W | 000 |
| 16 | DPC | PCI Clock Divider Select,<br>0 Divide by 4.<br>1 Divide by 2. | R/W | 0 |
| 15 | CPC | PCI Clock Select,<br>0 Clock is taken directly from mux. (The divider select bit DPC has no effect.)<br>1 Clock is taken from 2/4 divider. | R/W | 0 |
| 14:12 | MUH | USB Host Clock Mux Input Select. See Table 8-3 on page 160. | R/W | 000 |
| 11 | DUH | USB Host Clock Divider Select.<br>0 Divide by 4.<br>1 Divide by 2. | R/W | 0 |
| 10 | CUH | USB Host Clock Select.<br>0 Clock is taken directly from mux. (The divider select bit DUH has no effect.)<br>1 Clock is taken from 2/4 divider. | R/W | 0 |
| 9:7 | MUD | USB Device Clock Mux Input Select. See Table 8-3 on page 160. | R/W | 000 |
| 6 | DUD | USB Device Clock Divider Select.<br>0 Divide by 4.<br>1 Divide by 2. | R/W | 0 |
| 5 | CUD | USB Device Clock Select.<br>0 Clock is taken directly from mux. (The divider select bit DUD has no effect.)<br>1 Clock is taken from 2/4 divider. | R/W | 0 |
| 4:0 | — | Reserved. | | |

The specific values written to the 3-bit clock-mux-input-select fields are shown in Table 8-3. The FREQ*n* selections come from the output of the corresponding frequency generators, as shown in Figure 8-2 on page 156.

**Table 8-3. Clock Mux Input Select Values**

| Value | Meaning |
|---|---|
| 000 | No clocking |
| 001 | Auxiliary Clock |
| 010 | FREQ0 |
| 011 | FREQ1 |
| 100 | FREQ2 |
| 101 | FREQ3 |
| 110 | FREQ4 |
| 111 | FREQ5 |

### 8.1.3 PLL Control

There are two registers for controlling the two PLLs integrated into the Au1500 processor. Each PLL is independently programmable. Note that when programming the PLL control registers, the system designer must not violate the rated frequency limits of the Au1500 processor. Configuring the PLLs outside this frequency range causes undefined behavior.

For higher frequencies the Au1500 processor core requires a higher core voltage ($V_{DDI}$). Care should be taken that the system is providing the correct voltage for the operating frequency before changing the CPU clock. See Section 12.5 "DC Parameters" on page 218, for full information about the voltage/frequency requirements of the Au1500 processor.

The Core Cycle Counter register located at CP0 register 9 can be used to count core cycles. Please see Section 2.7 "Coprocessor 0" on page 30, for more information.

The two PLLs in the Au1500 processor drive the CPU clock and the auxiliary clock. The default PLL multiplier value is 16 for the CPU clock and 0 for the AUXPLL which has the following implications assuming a 12 MHz crystal on XTI12 and XTO12:

- CPU Clock = 192 MHz
- Auxiliary Clock = Disabled
- SBUS Clock = 96 MHz (SD  - SBUS divider - defaults to 2)
- Peripheral Bus = 48 MHz
- SDRAM Bus = 48 MHz.

When modifying the CPU clock frequency approximately 20 µs elapse while the CPU and bus clocks shut off and the CPU PLL locks to the new frequency. During this period instructions are not executed and interrupts are not serviced. Interrupts are serviced once execution begins again at the new frequency.

### 8.1.3.1 CPU PLL Control

The CPU PLL control register (**sys_cpupll**) resets to its default value only for a hardware reset. That is, after Sleep, and during a runtime reset the CPU PLL retains its frequency.

Note that when programming the CPU PLL control register the system designer must not violate the rated frequency limits of the Au1500 processor. Configuring the PLL outside this frequency range causes undefined behavior.

This register is read/write, but the value read is valid only after initialization. After coming out of reset, hardware reset or Sleep, this register must first be written for the value read back to be valid. For this reason it is suggested that this register be initialized at boot time regardless if the value is changed from default.

After writing to the **sys_cpupll** register, the system automatically halts for 20 µs to allow for the PLL to relock and clocks to become stable.

**sys_cpupll** **Offset = 0x0060**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | PLL[5:0] | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:6 | — | Reserved, should be cleared. | R/W | 0 |
| 5:0 | PLL | CPU PLL Multiplier. Determines the integer multiplier used to multiply the CPU PLL to generate the CPU clock.<br>For example, with the default of 16 and a 12 MHz OSC frequency, the CPU clock frequency is 192 MHz.<br>Note that PLL multiplier values that place the clock frequency outside of rated limits are invalid.<br>0–15: Reserved and undefined.<br>16–(*n*-1): Valid PLL multiplier.<br>*n*–63: Reserved and undefined.<br>Where *n* is the smallest PLL multiplier that would cause the CPU clock frequency to exceed the rated frequency limits of the part. | R/W | 0x10 |

### 8.1.3.2 Auxiliary PLL Control

The auxiliary PLL control register (**sys_auxpll**) resets to its default value on hardware reset, after Sleep, and during a runtime reset. This register is read/write, but the value read is valid only after initialization. For this reason it is recommended that system software initialize this register at hardware reset, runtime reset and Sleep, even if programming its default value.

Note that when programming the auxiliary PLL control register the system designer must not violate the rated frequency limits of the Au1500 processor. Configuring the PLL outside this frequency range causes undefined behavior.

Unlike the **sys_cpupll** register, writing **sys_auxpll** does not cause the system to halt. As a consequence, clocks taken from the AUX PLL may be unstable for up to 20 µs. To ensure stable clocks during AUX PLL lock time, the **sys_cpupll** register can be written with its current value to force the system to halt for 20 µs.

**sys_auxpll** **Offset = 0x0064**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | PLL[5:0] | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:6 | — | Reserved, should be cleared. | R/W | 0 |
| 5:0 | PLL | Auxiliary PLL Multiplier. Determines the integer multiplier used to multiply the auxiliary PLL to generate the auxiliary clock.<br>For example, with a value of 12 and a 12 MHz OSC frequency, the auxiliary clock frequency will be 144 MHz.<br>Note that PLL multiplier values that place the clock frequency outside of rated limits are invalid.<br>0: Disable the auxiliary PLL.<br>1–7: Reserved and undefined.<br>8–(*n*-1): Valid PLL multiplier.<br>*n*–63: Reserved and undefined.<br>Where *n* is the smallest PLL multiplier that would cause the auxiliary clock frequency to exceed the rated frequency limits of the part. | R/W | 0x00 |

## 8.1.4 Hardware Considerations

Note also that the EXTCLK[1:0] clocks are multiplexed signals and require programming of the **sys_pinfunc** register (see Section 8.3.1.1 "Pin Function" on page 168) as follows:

When using the external clocks from the clock generation block, the **sys_pinfunc** register must be programmed such that GPIO[2] and/or GPIO[3] are configured to be driven by EXTCLK0 and/or EXTCLK1.

Section 12.10 "Crystal Specifications" on page 234, define the crystal specifications.

## 8.1.5 Programming Considerations

When changing the CPU PLL value through the **sys_cpupll** register, the system automatically halts for 20 µs to allow clocks to stabilize. During this time no interrupts are serviced, potentially affecting real-time systems. However, modifying the **sys_auxpll** register does *not* cause the system to halt, and therefore clocks taken from the AUX PLL may be unstable for up to 20 µs. To ensure stable clocks while the AUX PLL locks, the **sys_cpupll** register can be written with its current value to force the system to halt for 20 µs.

## 8.2 Time of Year Clock and Real Time Clock

The Au1500 processor contains two programmable counters designed for use as a time of year clock (TOY) and real time clock (RTC). Because the TOY continues counting through Sleep, a TOY counter match can be used as a wake-up source. The RTC, however, will power down in Sleep mode.

Note that both the TOY and RTC counters are driven by the 32.768 KHz clock input. The clock input source can be a crystal or external clock. (See Section 12.10 "Crystal Specifications" on page 234 for crystal details.)

Each programmable counter employs a register to initialize the counter or load a new value, a trim divider to adjust the incoming 32.768 KHz clock, and three match registers which have associated interrupts that trigger on a match. Each counter is also able to generate an interrupt on every tick. All interrupts are maintained through the interrupt controller. Both programmable counters share a status register.

Figure 8-4 shows the functional block diagram of both the TOY and the RTC. The registers used to implement the block, including the counter control register (**sys_cntrctrl**), are described in the following section.

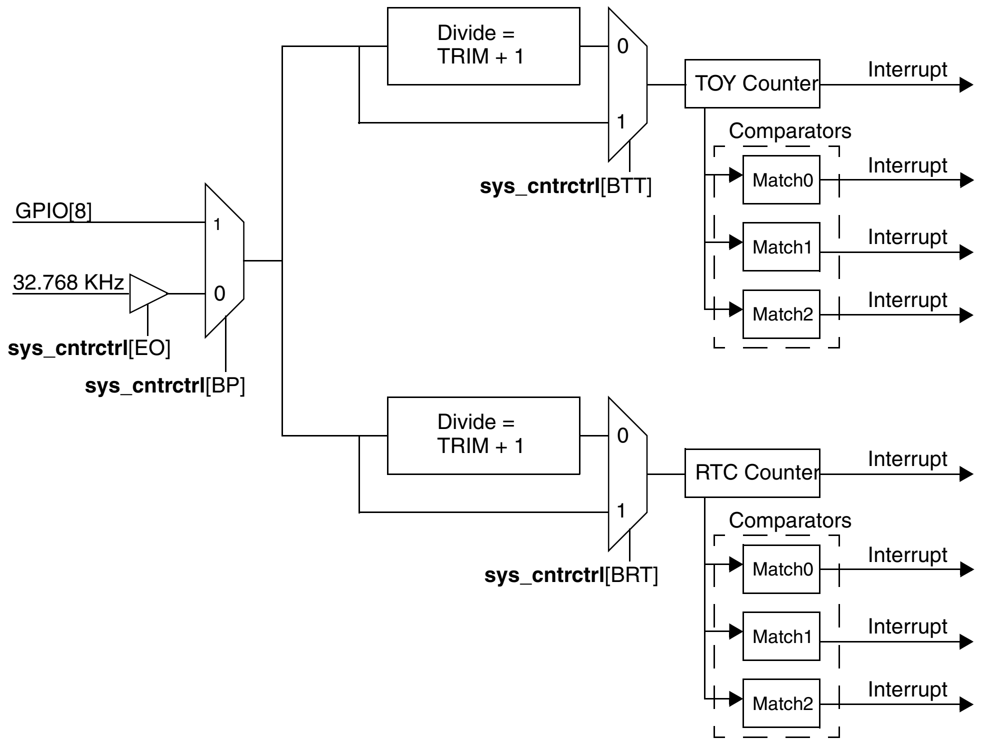


**Figure 8-4. TOY and RTC Block Diagram**

### 8.2.1 Time of Year Clock and Real Time Clock Registers

Each counter operates identically with the only difference being that the TOY continues counting through Sleep and the RTC does not.

The programmable counter control registers and their associated offsets are listed in Table 8-4. When functionality is identical for registers in the different programmable counters, only one general register description is presented with offsets pointing to the specific registers.

**Table 8-4. Programmable Counter Registers**

| Offset (Note 1) | Register Name | Description | Reset Type |
|---|---|---|---|
| 0x0000 | sys_toytrim | Trim value for 32.768 KHz clock source for TOY | Hardware |
| 0x0004 | sys_toywrite | TOY counter value is written through this register. | Hardware |
| 0x0008 | sys_toymatch0 | TOY match 0 value for interrupt generation. | Hardware |
| 0x000C | sys_toymatch1 | TOY match 1 value for interrupt generation. | Hardware |
| 0x0010 | sys_toymatch2 | TOY match 2 value for interrupt generation. | Hardware |
| 0x0014 | sys_cntrctrl | Control register for TOY and RTC. | Hardware |
| 0x0040 | sys_toyread | TOY counter value is read from this register. | Hardware |
| 0x0044 | sys_rtctrim | Trim value for 32.768 KHz clock source for RTC. | Hardware |
| 0x0048 | sys_rtcwrite | RTC counter value is written through this register. | Hardware |
| 0x004C | sys_rtcmatch0 | RTC match 0 value for interrupt generation. | Hardware |
| 0x0050 | sys_rtcmatch1 | RTC match 1 value for interrupt generation. | Hardware |
| 0x0054 | sys_rtcmatch2 | RTC match 2 value for interrupt generation. | Hardware |
| 0x0058 | sys_rtcread | RTC counter value is read from this register. | Hardware |

Note 1. See Table 8-1 on page 153 for base address.

#### 8.2.1.1 Trim Register

The TOY trim write status bit (**sys_cntrctrl**[TTS]) must be clear before writing **sys_toytrim**. It is set upon writing this register and is cleared by hardware when the write takes effect.

The RTC trim write status bit (**sys_cntrctrl**[RTS]) must be clear before writing **sys_rtctrim**. It is set upon writing this register and is cleared by hardware when the write takes effect.

This register is unpredictable at power on. During a runtime reset and during Sleep this register retains its value.

**sys_toytrim - TOY Trim** **Offset = 0x0000**
**sys_rtctrim - RTC Trim** **Offset = 0x0044**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | TRIM[15:0] | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:16 | — | Reserved, should be cleared. | R | 0 |
| 15:0 | TRIM | Divide value for 32.768 KHz input. Divide = TRIM + 1. | R/W | UNPRED |

### 8.2.1.2 Counter Write

The TOY value write status bit (**sys_cntrctrl**[TS]) must be clear before writing **sys_toywrite**. It is set upon writing this register and is cleared by hardware when the write takes effect.

The RTC value write status bit (**sys_cntrctrl**[RS]) must be clear before writing **sys_rtcwrite**. It is set upon writing this register and is cleared by hardware when the write takes effect.

This register is unpredictable at power on. During a runtime reset and during Sleep this register retains its value.

**sys_toywrite - TOY counter value write** **Offset = 0x0004**
**sys_rtcwrite - RTC counter value write** **Offset = 0x0048**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | COUNT[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | COUNT | Counter Write. The respective counter will be updated with the value written to this register at the next trimmed clock. | W | UNPRED |

### 8.2.1.3 Match Registers

The corresponding write status bit (**sys_cntrctrl**[TM*n*] or **sys_cntrctrl**[RM*n*]) must be clear before writing the below registers. It is set upon writing the register and is cleared by hardware when the write takes effect.

Each match register is capable of causing an interrupt as shown in Section 6.0 "Interrupt Controller" on page 95. The **sys_toymatch2** can be used to wake up from Sleep; see Section 8.4.5.2 "Wakeup Source Mask Register" on page 176. See also Section 8.2.2 "Programming Considerations" on page 167.

These registers are unpredictable at power on. During a runtime reset and during Sleep these registers retain their value.

**sys_toymatch0 - TOY Match 0** **Offset = 0x0008**
**sys_toymatch1 - TOY Match 1** **Offset = 0x000C**
**sys_toymatch2 - TOY Match 2** **Offset = 0x0010**
**sys_rtcmatch0 - RTC Match 0** **Offset = 0x004C**
**sys_rtcmatch1 - RTC Match 1** **Offset = 0x0050**
**sys_rtcmatch2 - RTC Match 2** **Offset = 0x0054**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | MATCH[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | MATCH | A match with the counter and the value in this register causes an interrupt. | R/W | UNPRED |

### 8.2.1.4 TOY and RTC Counter Control

The TOY and RTC counter control register (**sys_cntrctrl**) contains control bits and status bits to configure and control both programmable counters.

Write Status Bits: These bits indicate the status of the latest update to the respective register/field. When the corresponding register/field is written, this bit is set indicating that there is a write pending. When this bit is cleared the write has taken place. Software should poll the correct bit and insure that it is 0 before updating the respective register/field.

This register resets to default values only on a hardware reset. During a runtime reset and during Sleep this register retains its value.

**sys_cntrctrl** **Offset = 0x0014**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | RTS | RM2 | RM1 | RM0 | RS | | BP | | BRT | | BTT | | EO | CCS | | 32S | TTS | TM2 | TM1 | TM0 | TS |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:21 | — | Reserved, should be cleared. | R | 0 |
| 20 | RTS | **sys_rtctrim** Write status. | R | 0 |
| 19 | RM2 | **sys_rtcmatch2** write status. | R | 0 |
| 18 | RM1 | **sys_rtcmatch1** write status. | R | 0 |
| 17 | RM0 | **sys_rtcmatch0** write status. | R | 0 |
| 16 | RS | **sys_rtcwrite** write status<br>0 No write is pending. It is safe to write to the register.<br>1 A write is pending. Do not write to the register. | R | R |
| 15 | — | Reserved, should be cleared. | R | 0 |
| 14 | BP | Bypass the 32.768 KHz OSC<br>0 Select Oscillator Input (XTI32, XTO32).<br>1 GPIO[8] drives the counters. This is a test mode where GPIO[8] can drive the counters from an external source or through software using the GPIO controller. | R/W | 0 |
| 13 | — | Reserved, should be cleared. | R/W | 0 |
| 12 | BRT | Bypass RTC Trim.<br>0 Normal operation.<br>1 The RTC is driven directly by the 32.768 KHz clock, bypassing the trim. | R/W | 0 |
| 11 | — | Reserved, should be cleared. | R/W | 0 |
| 10 | BTT | Bypass TOY Trim.<br>0 Normal operation.<br>1 The TOY is driven directly by the 32.768 KHz clock, bypassing the trim. | R/W | 0 |
| 9 | — | Reserved, should be cleared. | R | 0 |
| 8 | EO | Enable 32.768 KHz Oscillator. Enables the clock for the RTC/TOY block.<br>0 Disable the clock.<br>1 Enable the clock.<br>Regardless of the clock source (crystal or overdriven clock through XTI32/ XTO32, or bypass through GPIO[8]), the EO bit must be set to enable the RTC/TOY counters. After enabling the clock by setting EO, poll the oscillator status bit (32S) until it returns a '1'. Once 32S is set, wait an additional one second to allow for frequency stabilization within the block before accessing other RTC/TOY registers (not including **sys_cntrctrl**).<br>**Note:** If the oscillator is being overdriven or bypassed through GPIO[8], be sure to set EO only *after* a stable clock is being driven into the part. | R/W | 0 |
| 7 | CCS | **sys_cntrcntrl** write status | R | 0 |
| 6 | — | Reserved, should be cleared. | R | 0 |
| 5 | 32S | 32.768 KHz Oscillator Status. Detects two consecutive 32 KHz cycles from the clock source for the RTC/TOY block.<br>0 Clock is not running.<br>1 Clock is running.<br>**Note:** Be sure to wait 1 second after 32S is set to allow for frequency stabilization within the block before accessing RTC/TOY registers. | R | UNPRED |
| 4 | TTS | **sys_toytrim** write status. | R | 0 |
| 3 | TM2 | **sys_toymatch2** write status. | R | 0 |
| 2 | TM1 | **sys_toymatch1** write status. | R | 0 |
| 1 | TM0 | **sys_toymatch0** write status. | R | 0 |
| 0 | TS | **sys_toywrite** write status<br>0 No write is pending. It is safe to write to the register.<br>1 A write is pending. Do not write to the register. | R | 0 |

### 8.2.2 Programming Considerations

To change the values of the counter and match registers, software must poll the state of the corresponding status bit in **sys_cntrctrl**. When the corresponding write status bit (**sys_cntrctrl**[TTS,TM*n*,TS] or **sys_cntrctrl**[RTS,RM*n*,RS]) is 0 it is okay to write a new value. Once the new value is written to the register the status bit will change to a 1. When the write status bit is 1 the new value is being updated in supporting hardware. When the write status changes to a 0 then the new value is active in the device.

## 8.3 Primary General Purpose I/O and Pin Functionality

The Au1500 processor contains two separate GPIO blocks (primary and secondary). This section covers the programming model for the primary general purpose I/O (GPIO) signals. The Au1500 processor supports 39 GPIOs, 23 of which are controlled by the primary GPIO block. For a description of the programming model for the secondary GPIO block see Section 7.6 "Secondary General Purpose I/O" on page 149.

This section also documents how to change the functionality of multiplexed pins. These pins can function at the system level as a GPIO signal, or they can be assigned a signal function dedicated to an integrated peripheral device.

Each GPIO can be configured as either an input or an output. The GPIO ports also can be connected to the internal interrupt controllers to generate an interrupt from input signals. See Section 6.0 "Interrupt Controller" on page 95 for information on interrupts.

### 8.3.1 Pin Functionality

To maximize the functionality of the Au1500 processor, many of the pins have multiple uses. Note that if a pin is programmed for a certain use, any other functionality associated with that pin can not be utilized at the same time. In other words, a pin can not be used as a GPIO at the same time it is assigned to a peripheral device.

(For reference,Figure 11-1 on page 201 shows a block diagram of all external signals. Signals that are multiplexed on one pin will show the shared function in parentheses.)

#### 8.3.1.1 Pin Function

This register resets to its default state at hardware reset, runtime reset and Sleep.

**sys_pinfunc** **Offset = 0x002C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | CS | USB | U3 | ▨ | U1 | SRC | EX1 | EX0 | ▨ | UR3 | ▨ | ▨ | NI2 | U0 | ▨ | ▨ | ▨ |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:17 | — | These bits are reserved and should be cleared. | R | all 0s |
| 16 | CS | Clock Select. Applies only when EX0 = 1.<br>0 EXTCLK0 will drive pin.<br>1 32 KHz OSC clock will drive pin. | R/W | 0 |
| 15 | USB | USB Functionality<br>0 USBDP and USBDM will drive pins (pins are connected to USB device module).<br>1 USBH0P and USBH0M will drive pins (pins are connected to USB host port 0). | R/W | 0 |
| 14 | U3 | UART3/GPIO[23].<br>0 U3TXD drives pin.<br>1 Pin is configured for GPIO[23]. | R/W | 1 |
| 13:12 | — | These bits are reserved and should be written as 0b11. | R/W | 11 |
| 11 | SRC | GPIO[6]/SMROMCKE.<br>0 Pin is configured for GPIO[6].<br>1 SMROMCKE drives pin. | R/W | 0 |
| 10 | EX1 | GPIO[3]/EXTCLK1.<br>0 Pin is configured for GPIO[3].<br>1 EXTCLK1 will drive pin. | R/W | 0 |
| 9 | EX0 | GPIO[2] / (EXTCLK0 or 32 KHz OSC).<br>0 Pin is configured for GPIO[2].<br>1 Pin is configured for EXTCLK0 or 32 KHz OSC. CS (bit 16) selects whether EXTCLK0 or the 32 KHz OSC drives the pin. | R/W | 0 |
| 8 | — | This bit is reserved and should be written as 0. | R/W | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 7 | UR3 | GPIO[14:9]/UART3.<br>0 Pins are configured as GPIO[14:9].<br>1 Pins are configured for optional UART3 flow control. U3DTR#, U3RTS#, U3RI#, U3DCD#, U3DSR#, and U3CTS# will drive pins.<br>**System Note**: For systems that use the UART3 interface but do *not* use the optional modem control signals (UR3=0), the modem status interrupts must be disabled (**uart3_inten**[MIE]=0) to avoid false UART3 interrupts when using GPIO[9], GPIO[10], GPIO[11], or GPIO[12] as an input. | R/W | 0 |
| 6:5 | — | These bits are reserved and should be written as 0b01. | R/W | 01 |
| 4 | NI2 | MAC1/GPIO[28:24].<br>0 Pins are configured as Ethernet port 1. N1TXD[3:0], and N1TXEN will drive port.<br>1 Pins are configured as GPIO[28:24]. | R/W | 1 |
| 3 | U0 | UART0/GPIO[20].<br>0 Pin is configured for U0TXD (necessary for UART0 operation).<br>1 Pin is configured as GPIO[20]. | R/W | 1 |
| 2:0 | — | These bits are reserved and should be written as 0b101. | R/W | 101 |

### 8.3.2 Primary GPIO Control Registers

The primary GPIOs on the Au1500 processor have been designed to simplify the GPIO control process by removing the need for a semaphore to control access to the registers. This is because there is no need to read, modify, write, as there are separate registers for setting and clearing a bit. In this way a function can freely manipulate its associated GPIOs without interfering with other functions.

Figure 8-5 shows the logical implementation of each GPIO. The names represent bit *n* of the corresponding register which affect GPIO[n].

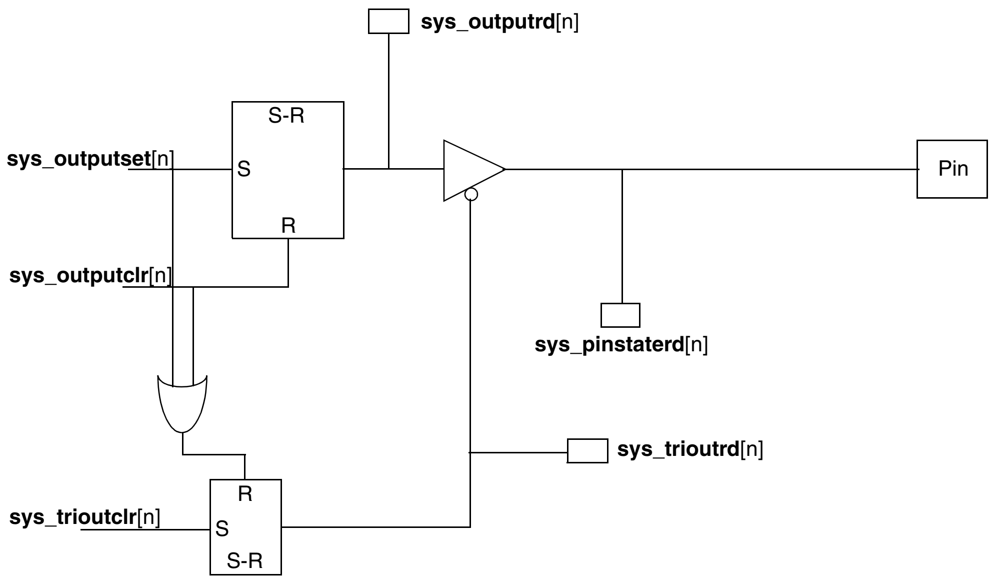


Figure 8-5. GPIO Logic Diagram

The following table shows the GPIO control registers and the associated offsets from **sys_base**. Certain registers share offsets and have different functionality depending on whether the access is a read or a write. The register descriptions detail the functionality of each register. Bit *n* of a particular register should be associated with GPIO[n] for all registers except **sys_pininputen**.

**Table 8-5. GPIO Control Registers**

| Offset (Note 1) | Register Name | Register Description | Default |
|---|---|---|---|
| 0x0100 | sys_trioutrd | The TRI-STATE/Output state register shows the current state of the GPIO.<br>0 GPIO[n] is in TRI-STATE. To TRI-STATE GPIO[n] set the corresponding bit in the **sys_trioutclr** register.<br>1 Output is enabled. To enable GPIO[n] as an output programming GPIO[n] as a 0 or 1 using the **sys_outputclr**[n] or **sys_outputset**[n] registers.<br>If the pin is not an output it should be in TRI-STATE. | 0x00000000<br>(all GPIOs are TRI-STATED) |
| 0x0100 | sys_trioutclr | | |
| 0x0108 | sys_outputrd | Controls the state of the GPIO[n] as an output.<br>0 To output a low level, set **sys_outputclr**[n].<br>1 To output a high level, set **sys_outputset**[n].<br>Programming a bit value in the output register brings the pin out of TRI-STATE mode and enables the output. | UNPRED |
| 0x0108 | sys_outputset | | |
| 0x010C | sys_outputclr | | |
| 0x0110 | sys_pinstaterd | Allows the pin state to be read when an input. This register will also give the output state. | UNPRED |
| 0x0110 | sys_pininputen | Any write to this register allows GPIO[31:0] to be used as inputs. This register must be written before any GPIO can be used as an input, an interrupt source, or for use as a wake up source. | UNPRED |

Note 1. See Table 8-1 on page 153 for base address.

### 8.3.2.1 GPIO Control Registers

Each GPIO control register is 32 bits wide with bit *n* in each register affecting GPIO[n].

These registers will reset to defaults only on a hardware reset. During a runtime reset and during Sleep this register will retain its value.

See Table 8-5 for the default values at hardware reset.

***rd**
***set**
***clr**

```
Bit 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
                                                            FUNC[31:0]
```

| Bits | Name | Description | Read/Write | Default |
|---|---|---|---|---|
| 31:0 | FUNC[n] | The function of each register is given in the previous table. FUNC[n] controls the functionality of GPIO[n]. | *_read - read only<br>*_set - write only<br>*_clear - write only<br>See the following text. | 0 |

Certain registers in the list have the same offset but offer different functionality depending on whether a read or a write is being performed.

Registers ending in *rd, *set and *clr have the following functionality:

- *rd registers are read only registers will read back the current value of the register.
- *set registers are write only registers and will set to 1 all bits that are written 1. Writing a value of 0 will have no impact on the corresponding bit.
- *clr registers are write only registers and will clear to zero all bits that are written 1. Writing a value of 0 will have no impact on the corresponding bit.

#### 8.3.2.2 GPIO Input Enable

The **sys_pininputen** is a 32-bit, write-only register. When this register is written, the input functionality of all GPIOs is enabled. This register enables GPIOs for use as an input but does not explicitly configure all GPIOs as inputs. The value of the GPIO control registers and the pin function register will define the state of each GPIO.

GPIOs cannot be used as inputs until this register is written. This write is required only once per hardware reset (i.e., Sleep and a runtime reset will not require another write to this register).

**sys_pininputen** **Offset = 0x0110**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | EN |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:1 | — | Reserved, should be cleared. | W | N/A |
| 0 | EN | A write to this bit (0 or 1) enables all GPIOs to be used as inputs. | W | N/A |

### 8.3.3 Hardware Considerations

The system pin function register (**sys_pinfunc**) controls the functionality of many GPIO/peripheral pins. If a pin is programmed for a certain functionality, all other functionality associated with that pin is disabled.

For example, if **sys_pinfunc**[U3] is cleared configuring the pin as U3TXD, GPIO[23] can not be used as a GPIO nor can the GPIO be configured as an interrupt. Conversely if **sys_pinfunc**[U3] is set configuring the pin as GPIO[23], U3TXD (and thus the UART3 interface) is not usable. GPIO[23] can be used as a GPIO and to generate interrupts.

### 8.3.4 Using GPIO for External DMA Requests

See Section 5.2 "Using GPIO as External DMA Requests (DMA_REQn)" on page 92 for information.

## 8.4 Power Management

The Au1500 processor contains a robust power management scheme allowing multiple levels of power conservation to enable the system designer options depending on whether power conservation or system responsiveness is more critical.

In the Au1500 processor, power management can be broken into three different areas:

- CPU
- Peripherals
- Device

The lowest power state consists of putting the entire device into a Sleep state. The CPU also supports two Idle states that differ as to whether bus snooping is supported. In addition each peripheral can have its clocks disabled when not in use thus significantly reducing the power draw by those blocks not in use.

The flow chart in Figure 8-6 shows the different stages of power management for the CPU (IDLE0,1) and the device (SLEEP) and how each state is entered and left. Note that any interrupt can be used to bring the CPU out of either Idle state while only a GPIO[7:0] or **sys_toymatch2** interrupt can be enabled (in **sys_wakemsk**) to bring the device out of Sleep.

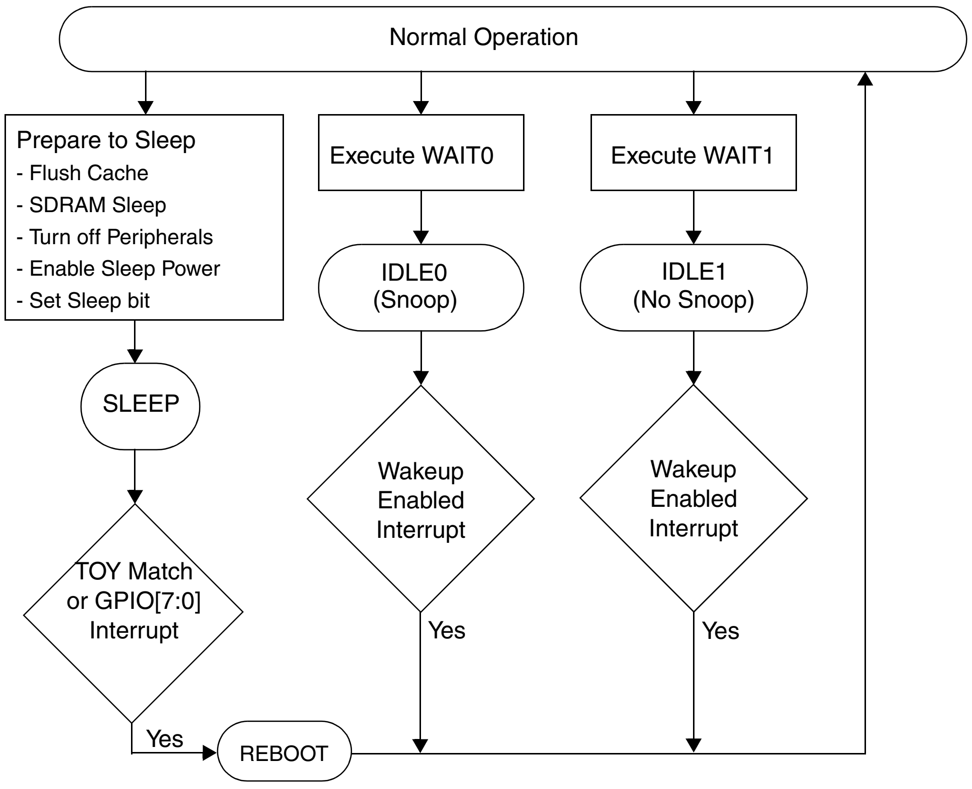


**Figure 8-6. Sleep and Idle Flow Diagram**

### 8.4.1 CPU Power Management - Idle

The CPU can be put into two different low-power Idle modes (IDLE0 and IDLE1) by using the wait instruction:

- In the IDLE0 state the CPU snoops the bus and cache coherency is maintained.
- In the IDLE1 state the CPU does not snoop the bus and cache coherency is lost.

The wait instruction and at least four instructions following it must be in the cache for the wait to occur. See Section 2.6.3 "WAIT Instruction" on page 29 for more information.

At all times the MMU, data cache, execution and multiply-and-accumulate blocks are placed in a low power state if they are not being used.

### 8.4.2 Returning from Idle

The processor wakes from the Idle state (IDLE0 or IDLE1) upon receiving an interrupt. The time required for the processor core to return to normal execution is as follows:

- Five to ten CPU clocks are needed to restart clocks to the CPU.
- It takes an additional ten CPU clocks for the core to recognize the interrupt and begin fetching the interrupt service routine.

Therefore, a maximum of 20 CPU clocks are required to resume normal instruction pipeline execution. If the interrupt service routine is in the instruction cache, the instruction returns immediately; otherwise, there is an additional delay while fetching the instruction from memory.

### 8.4.3 Peripheral Power Management

Peripheral power management is handled through clock management and disabling of unused peripherals. Table 8-6 lists the peripherals and their related power management registers. The actual register descriptions should be referred to for programming details.

Note that when separate reset/peripheral enable and clock-enable bits are provided, the reset must be applied first, and then the clocks should be disabled. This will simplify programming, as the suggested bring up sequence is typically to first enable clocks and then subsequently to bring the peripheral out of reset.

**Table 8-6. Peripheral Power Management**

| Peripheral | Power Management Register | Power Management Strategy |
|---|---|---|
| USB Host | usbh_enable | When the USB host is not in use the E bit can be cleared to disable the host. The CE bit should also be cleared to disable clocks to the block. |
| USB Device | usbd_enable | When the USB device is not in use the E bit can be cleared to disable the host. The CE bit should also be cleared to disable clocks to the block. |
| Ethernet MAC*n* | macen_mac*n* | When either block is not being used, the respective E[2:0] bits should be cleared to disable the MAC, and the CE bit should be cleared to gate clocks to the MAC. |
| UART*n* | uart*n*_enable | When a UART is not being used, the E bit should be cleared to hold the part in reset and the CE bit should be cleared to disable clocks to the block. |
| Primary General Purpose I/O (GPIO) Controller | sys_trioutclr | Although there is not a specific low-power configuration for the primary GPIOs, tristating the unused GPIOs minimizes their power usage. |
| Secondary General Purpose I/O (GPIO2) Controller | gpio2_enable | If no GPIO2 signals are being used, the GPIO2 module reset (MR) bit should be set to place the module in reset. Also, clear the CE bit to disable clocks to the block. (By default, the GPIO2 module is disabled coming out of reset.) |
| Programmable Counters (TOY and RTC) | sys_cntrctrl | If both the TOY and RTC are not being used, then disable the oscillator. |
| AC97 Controller | ac97_enable | If the AC97 block is not in use, the D bit should be used to disable the module and the CE bit should be disabled to gate clocks from the block. |

### 8.4.4 Device Power Management - Sleep

The Sleep state of the Au1500 processor puts the entire device into a low-power state. Sleep is the lowest power state of the part and requires a complete system initialization on wakeup. There are multiple steps to take when going into Sleep and waking up to insure data integrity. During this state all registers values outside the system control block are lost and cache coherency is not maintained.

The programmable counter 0 (intended for TOY) continues clocking and remains functional during Sleep. However, the programmable counter 1, as well as other clocks throughout the Au1500, are disabled during Sleep.

When coming out of Sleep there is a programmable delay defined by **sys_powerctrl**[VPUT]. This is the time that the system designer has to ensure $V_{DDI}$ is stable from the rising edge of PWR_EN.

To enter Sleep the following steps should be taken. This code should be run from Flash, or conversely the system programmer should guarantee that this code will run from cache because after SDRAM is put into auto-refresh mode, memory accesses will no longer work.

1) Enable Sleep Power by writing to the **sys_slppwr** register.
2) Turn off all peripherals. (Explicitly turning off all peripherals in use ensures a graceful transition to Sleep mode.)
3) Push dirty data out of the cache. (During Sleep cached data is lost.)
4) If SDRAM contents are to be kept through Sleep, SDRAM should be put into auto-refresh mode. See Section 3.1 "SDRAM Memory Controller" on page 44 for more information.

   If SDRAM is not needed to be maintained through Sleep, disable the SDRAM.
5) If using one of GPIO[7:0] as a wakeup source, **sys_pininputen** must be written to enable the GPIO as an input if this has not already been done at system startup.
6) The **sys_wakemsk** register should be set with the appropriate value according to what signal(s) should wake the processor.
7) The **sys_wakesrc** register should be written to explicitly clear any pending wake interrupts.
8) Enable Sleep by writing to the **sys_sleep** register. This step puts the system to Sleep.
9) As the system enters Sleep mode, the PWR_EN signal is negated. This can be used to disable $V_{DDI}$ if needed.

When the processor takes a Sleep interrupt to wake up, the following steps should be taken:

1) After the Sleep interrupt is taken, the PWR_EN signal is asserted by hardware. Within the time indicated by **sys_powerctrl**[VPUT], the system must ensure that $V_{DDI}$ is stable.
2) The processor will then boot from physical address 0x0 1FC0 0000 as normal.
3) If Sleep is to be used by the system and a different flow should be followed when coming out of Sleep the **sys_wakesrc** should be read to determine if the processor is coming out of Sleep and what caused the wakeup. The system should then write the **sys_wakesrc** register to clear this information.
4) The processor will need to perform complete system initialization. All registers except those described as otherwise in the System Control Block will be at their default values.

#### 8.4.4.1 Sleep Sequence and Timing

As the processor enters Sleep mode, the system designer has the option of disabling $V_{DDI}$ to conserve power. The PWR_EN signal defines the Sleep window. Figure 8-7 shows the Sleep sequence.

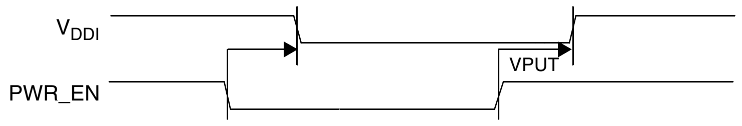


**Figure 8-7. Sleep Sequence**

The system designer must ensure $V_{DDI}$ is stable from the rising edge of PWR_EN within the time period as programmed in **sys_powerctrl**[VPUT]. Note that VDDXOK (not shown) remains asserted during the Sleep sequence.

## 8.4.5 Power Management Registers

The power management registers and their associated offsets are listed in Table 8-7. These registers are located off of the base shown in Table 8-1 "System Control Block Base Address" on page 153.

**Table 8-7. Power Management Registers**

| Offset (Note 1) | Register Name | Description | Reset Type |
|---|---|---|---|
| 0x0018 | sys_scratch0 | User-defined register that retains its value through Sleep. | Hardware |
| 0x001C | sys_scratch1 | User-defined register that retains its value through Sleep. | Hardware |
| 0x0034 | sys_wakemsk | Sets which GPIO or whether TOY match can cause Sleep wakeup. | Hardware |
| 0x0038 | sys_endian | Sets Big or Little Endian. | Hardware & Runtime |
| 0x003C | sys_powerctrl | Sets System Bus (SBUS) divider and power-up time. | Mixed - See Register Description |
| 0x005C | sys_wakesrc | Gives source of Sleep wakeup. | Hardware |
| 0x0078 | sys_slppwr | Initiates power state for Sleep mode. | Hardware |
| 0x007C | sys_sleep | Initiates Sleep mode. | Hardware |

Note 1. See Table 8-1 on page 153 for base address.

### 8.4.5.1 Scratch Registers

The scratch registers keep their values through Sleep and runtime resets. These registers allow the system programmer to save user-defined state information or a pointer to a context so that the previous context can be restored when coming out of Sleep, if needed. Note that the scratch registers have unpredictable default values after a hardware reset.

**sys_scratch0** **Offset = 0x0018**
**sys_scratch1** **Offset = 0x001C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | SCRATCH[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | SCRATCH | User-defined information. | R/W | UNPRED |

#### 8.4.5.2 Wakeup Source Mask Register

For each individual bit that is set, the corresponding signal or event (for the case of the TOY match) can be used to cause a Sleep wakeup.

A high level on the enabled GPIO will cause the interrupt to trigger.

This register will reset to defaults only on a hardware reset. During a runtime reset and during Sleep this register retains its value.

**sys_wakemsk** **Offset = 0x0034**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | M2 | GPIO[7:0] | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:9 | — | Reserved, should be cleared. | R | 0 |
| 8 | M2 | Setting this bit enables the programmable TOY Counter Match Register 2 (**sys_toymatch2**) to cause a wakeup interrupt. See Section 8.2.1.3 "Match Registers" on page 165. | R/W | 0 |
| 7:0 | GPIO[7:0] | Setting bit *n* causes GPIO[*n*] to cause a Sleep wakeup. | R/W | 0 |

#### 8.4.5.3 Endianness Register

To change the endianness of the Au1500 processor is a three step process as follows:

1) Program the endianness bit in the system endianness register (**sys_endian**[EN]).
2) Read the **sys_endian** register. (This is required to ensure the final write to the CP0 register will update the endian value.)
3) Read the CP0 register **Config0**. (See Section 2.7.15 "Configuration Register 0 (CP0 Register 16, Select 0)" on page 36.)
4) Write the value read back into the CP0 **Config0** register. The act of writing the CP0 register will put the processor into the endian state as programmed in **sys_endian**[EN].

This register as well as the processor endianness will reset to big endian after a hardware reset, runtime reset and after Sleep.

**sys_endian** **Offset = 0x0038**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | EN |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:1 | — | Reserved, should be cleared. | R | UNPRED |
| 0 | EN | Endianness.<br>0 Big Endian.<br>1 Little Endian. | R/W | 0 |

### 8.4.5.4 Power Control Register

Bits[6:5] of this register are reset to default values for a hardware reset, runtime reset and after Sleep.

Bits[4:0] of this register reset to default values only on a hardware reset. During a runtime reset and during Sleep these bits retain their values.

**sys_powerctrl** **Offset = 0x003C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | SI | SB | | VPUT | | SD | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:7 | — | Reserved, should be cleared. | R | 0 |
| 6 | SI | Idle State SBUS Clock Divider Enable<br>0 The Idle state SBUS clock divider is disabled.<br>1 Enable the SBUS clock to be divided by an additional factor of 2 when the processor is in an IDLE state (taken through the WAIT instruction). All peripheral bus clocks (such as the SDRAM and UART controllers) will be internally compensated with no programmer intervention required.<br>**Note:** SD must be programmed to 00 (divide by two) when SI is set. | R/W | 0 |
| 5 | SB | SBUS Clock Divider Enable<br>0 The SBUS clock divider is disabled.<br>1 Enable the SBUS clock to be divided by an additional factor of 2 when there is no bus activity. All clocks derived from the peripheral bus clock (such as the SDRAM and UART controllers) will be internally compensated with no programmer intervention required.<br>**Note:** SD must be programmed to 00 (divide by two) when SB is set. | R/W | 0 |
| 4 | — | Reserved, should be cleared. | R | 0 |
| 3:2 | VPUT | $V_{DDI}$ Power-up Time.<br>00 20 ms<br>01 5 ms<br>10 100 ms<br>11 2 µs | R/W | Hardware Reset<br>00 |
| 1:0 | SD | SBUS Clock Divider.<br>00 2<br>01 3<br>10 4<br>11 Reserved | R/W | Hardware Reset<br>00 |

### 8.4.5.5 Wakeup Cause Register

Before setting the Sleep bit this register should be cleared. This register will retain pending interrupts according to the setting in the **sys_wakemsk** register even if those events did not occur during Sleep. In other words if a GPIO's functionality is multiplexed between multiple functions, a high level could cause the associated **sys_wakesrc** bit to be set even if the action did not occur during Sleep.

The bits in this register must be explicitly cleared as they will hold their values through Sleep and a runtime reset.

All bits in this register are set by hardware and cleared by any write to this register.

**sys_wakesrc** **Offset = 0x005C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | M2 | GP7 | GP6 | GP5 | GP4 | GP3 | GP2 | GP1 | GP0 | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | SW | IP |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | Rs | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Rs |  |  |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:25 | — | Reserved, should be cleared. | R/W | 0 |
| 24 | M2 | Programmable TOY Match 2 caused wakeup from Sleep. Set by hardware on Sleep wakeup due to TOY match.<br>Cleared by hardware on VDDXOK assertion.<br>This bit must be explicitly cleared by software (any write) because it holds its value through Sleep and runtime reset. | R/W | 0 |
| 23 | GP7 | GPIO[*n*] caused wakeup from Sleep. Set by hardware on Sleep wakeup due to GPIO[*n*].<br>This bit must be explicitly cleared by software (any write) because it holds its value through Sleep and runtime reset. | R/W | 0 |
| 22 | GP6 |  | R/W | 0 |
| 21 | GP5 |  | R/W | 0 |
| 20 | GP4 |  | R/W | 0 |
| 19 | GP3 |  | R/W | 0 |
| 18 | GP2 |  | R/W | 0 |
| 17 | GP1 |  | R/W | 0 |
| 16 | GP0 |  | R/W | 0 |
| 15:2 | — | Reserved, should be cleared. | R/W | 0 |
| 1 | SW | Sleep Wakeup. This bit is set by hardware on a Sleep wakeup and cleared by software by a write to this register.<br>A runtime reset can be detected if both SW and IP are 0 at boot.<br>This bit must be explicitly cleared by software (any write) because it holds its value through Sleep and runtime reset. | R/W | 0 |
| 0 | IP | Initial Power-up. This bit is set by hardware on a hardware reset and cleared by software by a write to this register.<br>A runtime reset can be detected if both SW and IP are 0 at boot.<br>This bit must be explicitly cleared by software (any write) because it holds its value through Sleep and runtime reset. | R/W | 1 |

### 8.4.5.6 Sleep Power Register

**sys_slppwr** **Offset = 0x0078**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SP | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | SP | A write to this register prepares the internal power supply for going to Sleep. | W | UNPRED |

### 8.4.5.7 Sleep Register

**sys_sleep** **Offset = 0x007C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SL | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | SL | A write to this register puts system to Sleep. | W | UNPRED |

# 9 Power-up, Reset and Boot

This section presents the power-up, hardware reset and runtime reset sequence for the Au1500 processor. In addition the boot vector is described.

## 9.1 Power-up Sequence

The Au1500 processor power structure is designed such that the external I/O voltage ($V_{DDX}$) is driven separately from the core voltage ($V_{DDI}$). In this way the core voltage can be sourced at lower voltages saving power. In addition the Au1500 processor is designed to allow the system designer to remove the core voltage during Sleep to maximize power efficiency.

Two signals VDDXOK and PWR_EN are used to facilitate this power strategy. VDDXOK is used as a signal to the processor that power on $V_{DDX}$ is stable. Stable is defined as having reached 90% of its nominal value. PWR_EN is an output from the Au1500 that is asserted after VDDXOK is asserted and can be used as an enable to the regulator that is providing the core voltage, $V_{DDI}$.

The following describes the power-up sequence for the Au1500 processor:

1) Apply $V_{DDX}$ (3.3V I/O power).
2) When $V_{DDX}$ has reached 90% of nominal, assert VDDXOK.
3) The Au1500 processor then asserts PWR_EN which can be used to enable the regulator driving $V_{DDI}$ (CPU power).

Figure 9-1 shows the power-up sequence, including arrows representing causal dependencies. For the timing specifications of this sequence, refer to Section 12.7.1 "Power-up Sequence Timing" on page 230.

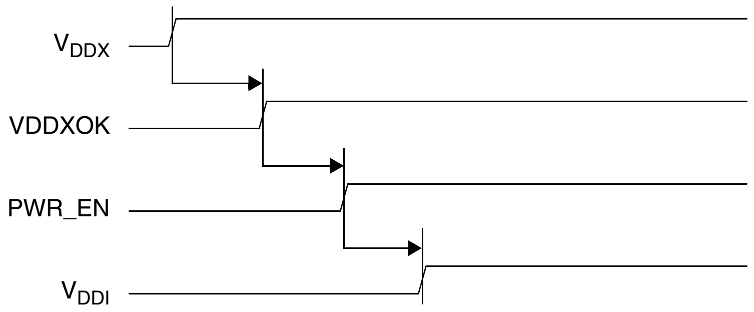


**Figure 9-1. Power-up Sequence**

## 9.2 Reset

A hardware reset is defined as a reset in which both VDDXOK and RESETIN# are toggled. Typically this happens only at power-on, but a system designer can choose to tie VDDXOK and RESETIN# together in which case all resets will be hardware resets.

For a runtime reset, power remains applied and only the RESETIN# signal is toggled. Note that certain registers, specifically some of those in the system control block, are not affected by this type of reset. See the register description for the register in question for more information. If a register is not reset to defaults by both hardware reset and runtime reset, it is noted in the register description.

### 9.2.1 Hardware Reset

For a hardware reset, VDDXOK makes a transition from low to high followed by RESETIN# negating (transitioning from low to high). The following sequence describes a hardware reset:

1) ROMSEL and ROMSIZE should be terminated in the design so the appropriate boot type occurs. These values should not change during runtime.
2) At the same time or after VDDXOK is asserted, RESETIN# can be negated. In other words, RESETIN# can not be negated before VDDXOK is asserted. This allows VDDXOK and RESETIN# to be tied together.
3) RESETOUT# is negated after RESETIN# is negated.

Figure 9-2 shows the hardware reset sequence, including arrows representing causal dependencies. For the timing specifications of this sequence, refer to Section 12.7.2 "Hardware Reset Timing" on page 231.

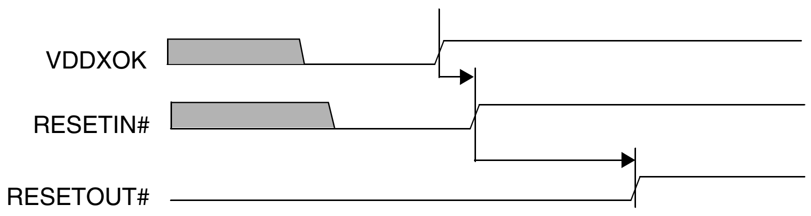


**Figure 9-2. Hardware Reset Sequence**

### 9.2.2 Runtime Reset

During runtime (after power is stable) the reset sequence can be broken down as follows:

1) During a runtime reset it is assumed that $V_{DDX}$ and $V_{DDI}$ remain at their nominal voltage. In addition, VDDXOK must remain asserted; otherwise, a hardware reset will occur. PWR_EN remains asserted by the Au1500 processor.
1) RESETIN# is held asserted long enough to be recognized as a valid reset.
2) The processor acknowledges the reset by asserting RESETOUT#.
3) After RESETIN# is released, the processor signals the end of the reset by negating RESETOUT#.

Note that certain registers (specifically those in the system control block) are not affected by a runtime reset. Note also that ROMSEL and ROMSIZE should already be terminated in the design so the appropriate boot type occurs—these values should not change during runtime.

Figure 9-3 shows the runtime reset sequence, including arrows representing causal dependencies. For the timing specifications of this sequence, refer to Section 12.7.3 "Runtime Reset Timing" on page 232.

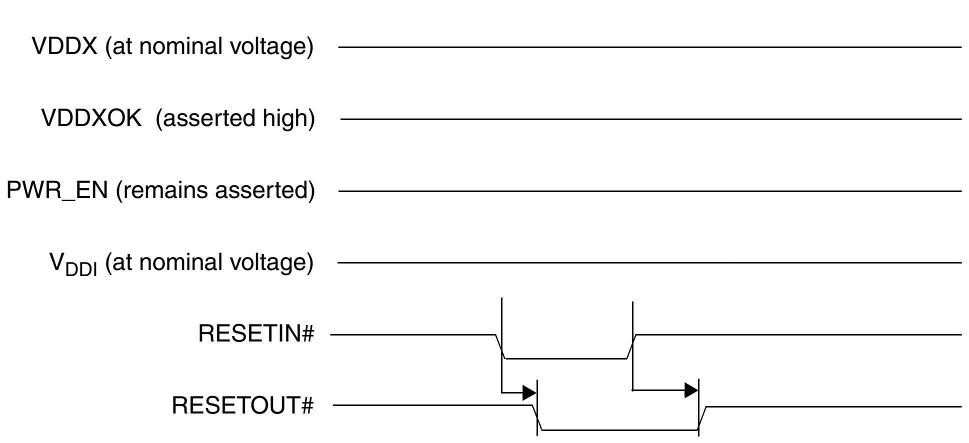

VDDX (at nominal voltage)
VDDXOK (asserted high)
PWR_EN (remains asserted)
$V_{DDI}$ (at nominal voltage)
RESETIN#
RESETOUT#

**Figure 9-3. Runtime Reset Sequence**

## 9.3 Boot

For both hardware and runtime resets, the CPU boots from KSEG1 address 0xBFC0 0000 which is translated to physical address 0x0 1FC0 0000; therefore, the system designer should place the start of the boot code at 0x0 1FC0 0000.

The ROMSEL and ROMSIZE signals determine the boot device type and width according to Table 9-1. The system designer should configure ROMSEL and ROMSIZE appropriately. Note that ROMSEL and ROMSIZE should not change during runtime.

**Table 9-1. ROMSEL and ROMSIZE Boot Device**

| ROMSEL | ROMSIZE | Boot Device Type and Width |
|---|---|---|
| 0 | 0 | Boot from 32-bit ROM interface. |
| 0 | 1 | Boot from 16-bit ROM interface. |
| 1 | 0 | Boot from 32-bit SMROM interface and Sync Flash boot. |
| 1 | 1 | Boot from PCI memory space. |

RCS0# is configured to be enabled for 0x0 1FC0 0000 at default when booting from a ROM device (ROMSEL = 0, ROMSIZE = x). See Section 3.2 "Static Bus Controller" on page 53, for more information about the default timing and size of the address enabled at reset.

SDCS0# is configured to be enabled for 0x0 1FC0 0000 at default when booting from a SMROM device (ROMSEL = 1, ROMSIZE = 0). See Section 3.1 "SDRAM Memory Controller" on page 44, for more information about the default timing and size of the address enabled at reset.

### 9.3.1 Endianness and 16-Bit Static Bus Boot

When booting from a 16-bit chip select on the static bus, the system designer must be sure the data format (endianness) is consistent across the Au1 core, the static bus controller, and the software image itself. This section describes how to make endianness consistent for both little- and big-endian systems.

For more on how the endian mode affects the behavior of 16-bit static bus chip selects, see "Halfword Ordering and 16-bit Chip Selects" on page 70.

**Note:** When programming ROM or Flash devices with a part programmer, take care to ensure that the programmer is not swapping bytes or halfwords erroneously. The configuration of the part programmer is often a source of error when initially bringing-up a new design.

### 9.3.2 16-Bit Boot for Little-Endian System

Booting from 16-bit ROM or Flash in a system that is intended to run the Au1 core in little-endian mode is very straightforward. Generally speaking, the boot code and/or the application is compiled for little-endian. Because the the Au1 core defaults to big-endian mode, the boot code must change the Au1 core endianness to little-endian *before* any data accesses (to the 16-bit chip-select). The resulting boot code and/or application image is placed in the ROM/Flash memory in the little-endian format.

Even though the Au1 core starts in big-endian mode, the static bus controller properly retrieves instructions needed to boot the system since the application image is in little-endian format and the static bus controller defaults to little-endian ordering out of reset.

### 9.3.3 16-Bit Boot for Big-Endian System

Booting from 16-bit ROM or Flash in a system that is intended to run the Au1 core in big-endian mode is very straightforard, but does need one extra, important step.

Generally speaking, the boot code and/or the application is compiled for big-endian. The boot code must set the **mem_stcfg**[BE] bit before it can properly fetch/reference the big-endian image. The resulting boot code and/or application image is placed in the ROM/Flash memory in the big-endian format.

In this situation, there is the dilema that, out of reset, the Static Bus controller defaults to little-endian ordering, but the application image itself is in big-endian format. The solution is to place the following code at the reset exception vector (KSEG1 address 0xBFC 00000, physical adddress 0x0 1FC0 0000):

```
    .long 0xb4003c08 # lui     t0,0xb400
    .long 0x10003508 # ori     t0,t0,0x1000
    .long 0x00008d09 # lw      t1,0(t0)
    .long 0x02003529 # ori     t1,t1,0x200
    .long 0x0000ad09 # sw      t1,0(t0)
    .long 0x00000000 # nop
    .long 0x00000000 # nop
    .long 0x00000000 # nop
    .long 0x00000000 # nop
```

The code does a read-modify-write of register **mem_stcfg0** to set the BE bit. The values in the .long statements above are the halfword-swapped opcodes of the instructions in the comments to the right. With this technique, these first few instructions are actually in the little-endian format to match the static bus controller out of reset, and set **mem_stcfg**[BE] which in turns allows the remainder of the big-endian memory contents to be accessed properly. The NOPs are necessary to ensure that the Au1 core pipeline does not contain incorrectly [halfword swapped] prefetched instructions. Note too that the NOP opcode 0x00000000 is the same instruction regardless of endian ordering.

**Note:** The boot code should set **mem_stcfg0**[BE] as early as possible, preferrably as the first activity. It is especially important to ensure that no cachable accesses take place to the 16-bit device, else the cache will contain the halfword swapped contents of the 16-bit memory.

### 9.3.4 System Boot

For system debug, the processor can be configured to boot from the EJTAG probe through the EJTAG port; see Section 10.0 "EJTAG Implementation" on page 184 for more information.

# 10 EJTAG Implementation

The Au1500 processor implements EJTAG following the MIPS' EJTAG 2.5 Specification. This section presents the EJTAG implementation on the Au1500 processor while concentrating on those features from the EJTAG 2.5 specification which are implementation specific. In addition, those features which have not been implemented or any differences in the Au1500 processor implementation of EJTAG from the rev 2.5 specification are also noted.

It is assumed that the EJTAG 2.5 specification will be referenced for implementation details not covered here. If a particular bit is not implemented it can be assumed that the functionality associated with the bit is not implemented or not applicable unless otherwise noted.

The following features comprise the EJTAG implementation on the Au1500 processor:

- Extended instructions SDBBP and DERET
- Debug exceptions
- Extended CP0 registers DEBUG, DEPC and DESAVE
- EJTAG memory range 0xFF200000-0xFF3FFFFF
- Instruction/data breakpoints through the watch exception (specific to Au1500)
- Processor bus breakpoints (from EJTAG 2.0)
- Memory overlay (from EJTAG 2.0)
- EJTAG tap per IEEE1149.1

Note that the optional data and instruction breakpoint features from the EJTAG 2.5 specification are not implemented.

## 10.1 EJTAG Instructions

Both SDBBP and DERET are supported by the Au1500 processor:

- SDBBP causes a Debug Breakpoint exception.
- DERET is used to return from a Debug Exception.

## 10.2 Debug Exceptions

The following exceptions will cause entry into debug mode.

- DSS - debug single step
- DINT - debug interrupt, processor bus break
- DBp - execution of SDBBP instruction
- DWATCH - debug watch exception. Au1500 processor-specific implementation allowing CPU watch exception to cause debug exception. See description of the Section 10.4.2.6 "EJWatch Register (TAP Instruction EJWATCH)" on page 198 register.

Note that other normal exceptions, when taken in debug mode, will be handled by the debug exception handler.

## 10.3 Coprocessor 0 Registers

The Coprocessor 0 Registers for EJTAG are shown in Table 10-1.

**Table 10-1. Coprocessor 0 Registers for EJTAG**

| Register Number | Select | Name | Description |
|---|---|---|---|
| 23 | 0 | debug | Debug indications and controls for the processor |
| 24 | 0 | depc | Program Counter at last debug exception or exception in debug mode |
| 31 | 0 | desave | Debug exception save register |

### 10.3.1 Debug Register (CP0 Register 23, Select 0)

The Debug register contains the cause of the most recent debug exception and exception in Debug Mode. It also controls single stepping. Only the DM bit and the EJTAGver field are valid when read from the Debug register in Non-Debug Mode; the value of all other bits and fields is UNPREDICTABLE.

The following bits and fields are updated only on debug exceptions and/or exceptions in Debug Mode:

- DSS, DBp, DINT are updated on both debug exceptions and on exceptions in Debug Modes.
- DExcCode is updated on normal exceptions in Debug Mode, and is undefined after a debug exception.
- DBD is updated on both debug and on normal exceptions in Debug Modes.

**debug** **CP0 Register 23, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | DD | DM | ND | LS |  |  | CD |  |  |  |  |  |  |  |  | VER |  |  |  | DEXCOSE |  |  | NS | SS |  |  | DI |  |  |  | DB | DS |
| Def. | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | x | x | x | x | x | 0 | 0 | 0 | 0 | x | 0 | 0 | 0 | x | x |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31 | DD | DBD. Indicates whether the last debug exception or exception in Debug Mode occurred in a branch or jump delay slot.<br>0 Not in delay slot<br>1 In delay slot | R | UNPRED |
| 30 | DM | Indicates that the processor is operating in Debug Mode.<br>0 Processor is operating in Non-Debug Mode<br>1 Processor is operating in Debug Mode | R | 0 |
| 29 | ND | NoDCR.<br>0 DSEG is present. | R | 0 |
| 28 | LS | LSNM. Controls access of loads/stores between dseg and remaining memory when dseg is present and while in debug mode.<br>0 Loads/stores in dseg address range go to dseg<br>1 Loads/stores in dseg address range go to system memory | R/W | 0 |
| 27 | — | Reserved, should be cleared. *This bit is called Doze in the EJTAG 2.5 specification and was not implemented.* | R | 0 |
| 26 | — | Reserved, should be cleared. *This bit is called Halt in the EJTAG 2.5 specification and was not implemented.* | R | 0 |
| 25 | CD | CountDM. This bit is 0, indicating that the counter will be stopped in debug mode. | R | 0 |
| 24 | — | Reserved, should be cleared. *This bit is called IBusEP in the EJTAG 2.5 specification and was not implemented.* | R | 0 |
| 23 | — | Reserved, should be cleared. *This bit is called MCheckP in the EJTAG 2.5 specification and was not implemented.* | R | 0 |
| 22 | — | Reserved, should be cleared. *This bit is called CacheEP in the EJTAG 2.5 specification and was not implemented.* | R | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 21 | — | Reserved, should be cleared. *This bit is called DBusEP in the EJTAG 2.5 specification and was not implemented.* | R | 0 |
| 20 | — | Reserved, should be cleared. *This bit is called IEXI in the EJTAG 2.5 specification and was not implemented.* | R | 0 |
| 19 | — | Reserved, should be cleared. *This bit is called DDBSImpr in the EJTAG 2.5 specification and was not implemented.* | R | 0 |
| 18 | — | Reserved, should be cleared. *This bit is called DDBLImpr in the EJTAG 2.5 specification and was not implemented.* | R | 0 |
| 17:15 | VER | EJTAGver.<br>1 EJTAG Version 2.5 | R | 1 |
| 14:10 | DEXCODE | DExcCode. Indicates the cause of the latest exception in Debug Mode.<br>The field is encoded as the ExcCode field in the Cause register for those exceptions that can occur in Debug Mode (the encoding is shown in the MIPS32 specification), with addition of code 30 with the mnemonic CacheErr for cache errors.<br>This value is undefined after a debug exception. | R | UNPRED |
| 9 | NS | NoSSt.<br>0 Single step is implemented. | R | 0 |
| 8 | SS | SSt. Controls whether single-step feature is enabled:<br>0 No enable of single-step feature<br>1 Single-step feature enabled | R/W | 0 |
| 7:6 | — | Reserved, should be cleared. | R | 0 |
| 5 | DI | DINT. Indicates that a Debug Interrupt exception occurred. This could be either a Processor Bus Break (indicated by BS0 in the Processor Bus Break Status Register) or EJTAG break. The BS0 bit should be checked to see what caused the exception.<br>Cleared on exception in Debug Mode.<br>0 No Debug Interrupt exception<br>1 Debug Interrupt exception | R | UNPRED |
| 4 | — | Reserved, should be cleared. *This bit is called DIB in the EJTAG 2.5 specification and was not implemented.* | R | 0 |
| 3 | — | Reserved, should be cleared. *This bit is called DDBS in the EJTAG 2.5 specification and was not implemented.* | R | 0 |
| 2 | — | Reserved, should be cleared. *This bit is called DDBL in the EJTAG 2.5 specification and was not implemented.* | R | 0 |
| 1 | DB | DBp. Indicates that a Debug Breakpoint exception occurred. Cleared on exception in Debug Mode.<br>0 No Debug Breakpoint exception<br>1 Debug Breakpoint exception | R | UNPRED |
| 0 | DS | DSS. Indicates that a Debug Single Step exception occurred. Cleared on exception in Debug Mode.<br>0 No debug single-step exception<br>1 Debug single-step exception | R | UNPRED |

### 10.3.2 Debug Exception Program Counter Register

The Debug Exception Program Counter (DEPC) register is a read/write register that contains the address at which processing resumes after the exception has been serviced.

Hardware updates this register on debug exceptions and exceptions in Debug Mode.

For precise debug exceptions and precise exceptions in Debug Mode, the DEPC register contains either:

- The virtual address of the instruction that was the direct cause of the exception; or
- the virtual address of the immediately preceding branch or jump instruction, when the exception-causing instruction is in a branch delay slot, and the Debug Branch Delay (BDB) bit in the Debug register is set.

For imprecise debug exceptions and imprecise exceptions in Debug Mode, the DEPC register contains the address at which execution is resumed when returning to Non-Debug Mode.

**depc - Debug Exception Program Counter** **CP0 Register 24, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | DEPC[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | DEPC | Debug Exception Program Counter | R/W | UNPRED |

### 10.3.3 Debug Exception Save Register

The Debug Exception Save (DESAVE) register is a read/write register that functions as a simple scratchpad register.

The debug exception handler uses this to save one of the GPRs, which is then used to save the rest of the context to a pre-determined memory area, for example, in the dmseg. This register allows the safe debugging of exception handlers and other types of code where the existence of a valid stack for context saving cannot be assumed.

**desave - Debug Exception Save Register** **CP0 Register 31, Select 0**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | DESAVE[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | DESAVE | Debug Exception Save contents | R/W | UNPRED |

## 10.4 EJTAG Memory Range

In debug mode accesses to virtual 0xFF200000-0xFF3FFFFF bypass translation.

The debug memory is split into two logical divisions:

- dmseg: 0xFF20 0000-0xFF2F FFFF
- drseg: 0xFF30 0000-0xFF3F FFFF

Note that the physical address addr[35:32] of this range is zero.

dmseg is the memory range that will be serviced by the probe TAP in debug mode for all instruction accesses to this virtual address range and for data accesses if the LSNM in the Debug Register is 0.

drseg is the memory range containing the EJTAG memory mapped registers and is accessible when LSNM in the Debug Register is 0.

### 10.4.1 EJTAG Memory Mapped Registers

Table 10-2 shows the EJTAG memory mapped registers located in drseg.

**Table 10-2. EJTAG Memory Mapped Registers at 0xFF300000**

| Offset | Register | Description |
|---|---|---|
| 0x0000 | dcr | Debug Control Register |
| 0x000C | pbs | Processor Break Status |
| 0x0300 | pab | Processor Address Bus Break |
| 0x0304 | pdb | Processor Data Break |
| 0x0308 | pdm | Processor Data Mask |
| 0x030C | pbcam | Processor Control/Address Mask |
| 0x0310 | phab | Processor High Address Break |
| 0x0314 | pham | Processor High Address Mask |

The EJTAG implementation in the Au1500 processor does not employ data breakpoints and instruction breakpoints as described in the EJTAG 2.5 specification. Instead it offers Processor breakpoints as described in the EJTAG 2.0.0 specification.

The Processor Bus Match registers monitor the bus interface of the MIPS CPU and provide debug exception or trace trigger for a given physical address and data.

In addition, the implementation allows the CPU watchpoints to cause a debug exception. This functionality is enabled through the EJTAG TAP port. Please see Section 10.4.2.6 "EJWatch Register (TAP Instruction EJWATCH)" on page 198 for details.

### 10.4.1.1 Debug Control Register

The Debug Control Register (DCR) controls and provides information about debug issues. The width of the register is 32 bits. The DCR is located in the drseg at offset 0x0000.

**dcr - Debug Control Register** **Offset = 0x0000**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | EN | | | | | | | | | | | | DB | IB | | | | | | | | | | | | IE | NE | NP | SR | PE |
| Def. | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:30 | — | Reserved, should be cleared. | R | 0 |
| 29 | EN | ENM.<br>1 Processor is big Endian in both debug and kernel mode. | R | 1 |
| 28:18 | — | Reserved, should be cleared. | R | 0 |
| 17 | DB | DataBrk.<br>0 No data hardware breakpoints implemented. | R | 0 |
| 16 | IB | InstBrk.<br>0 No instruction hardware breakpoints implemented. | R | 0 |
| 15:5 | — | Reserved, should be cleared. | R | 0 |
| 4 | IE | IntE.<br>1 Interrupt enabled in debug mode depending on other enabling mechanisms. | R | 1 |
| 3 | NE | NMIE.<br>1 Non-Maskable Interrupt is enable for non-debug mode.<br>The NMI is not implemented in the Au1500 so this bit has no applicability. | R | 1 |
| 2 | NP | NMIPend.<br>0 No NMI pending.<br>The NMI is not implemented in the Au1500 so this bit has no applicability. | R | 0 |
| 1 | SR | SRstE.<br>1 Soft reset is fully enabled.<br>Soft Reset is not implemented in the Au1500 so this bit has no applicability. | R | 1 |
| 0 | PE | ProbEn. Indicates value of the ProbEn value in the ECR register.<br>0 No access should occur to dmseg<br>1 Probe services accesses to dmseg | R | Same value as ProbEN in ECR |

### 10.4.1.2 Processor Bus Break Status Register

**pbs - Processor Bus Break Status** **Offset = 0x000C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ▨ | OLP | ▨ | ▨ | BCN | | | | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | BS |
| Def. | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31 | — | Reserved, should be cleared. | R | 0 |
| 30 | OLP | 1 Memory overlay functionality is implemented for processor breaks. | R | 1 |
| 29:28 | — | Reserved, should be cleared. | R | 0 |
| 27:24 | BCN | Number of Processor Breaks.<br>1 One Channel has been implemented for the Processor Bus Break. | R | 1 |
| 23:15 | — | Reserved, should be cleared. | R | 0 |
| 14:1 | — | Reserved, should be cleared. *These bits are the Bsn bits in the EJTAG 2.0.0 specification and are not needed since only one break is implemented.* | R | 0 |
| 0 | BS | Break Status. This bit, when set, indicates that a processor bus break or processor bus trigger has occurred. BS can be cleared by activating PrRst (EJTAG CONTROL Register), hard reset and also by writing a ‘0’ to it.<br>The Debug handler must clear this bit before returning from debug mode. | R/W | 0 |

### 10.4.1.3 Processor Address Bus Break

This register contains the bits of the physical Processor Address Bus Break.

**pab - Processor Address Bus Break** **Offset = 0x0300**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PAB[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | PAB | Processor Address Bus Break 0. This index contains the lower 32 bits of the physical address. In combination with the high order address bits, these bits make up the break address. | R | UNPRED |

### 10.4.1.4 Processor Data Bus Break

This register specifies the data value for the Processor Data Bus match.

**pdb - Processor Data Bus Break** **Offset = 0x0304**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PDB[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | PDB | Processor Data Bus Break 0. This index contains the 32 bits of the data bus match. | R | UNPRED |

### 10.4.1.5 Processor Data Mask/Upper Overlay Address Mask

This register is dual purpose depending on the value of the Overlay Enable bit in the Bus Break Control and Address Mask.

This register specifies the mask value for the Processor Data Mask register. Each bit corresponds to a bit in the data register.

**pdm_uoam - Processor Data Mask** **Offset = 0x0308**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PDM[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def. | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | PDM | Applies only when OE not enabled. Processor Data Mask 0. When OE in the pbcam register is not enabled.<br>0 Data bit is not masked, data bit is compared.<br>1 Data bit is masked, data bit is not compared. | R | UNPRED |
| 31:24 | UOAM | Applies only when OE is enabled. Upper Overlay Address Mask. These bits represent bits 31:24 of the address mask and are combined with the LAM and HAM fields to create a complete 36 bit address mask.<br>0 Address bit is not masked, address bit is compared.<br>1 Address bit is masked, address bit is not compared.<br>Note that bits [23:0] are not used when OE is set and should be written 0. | R/W | UNPRED |

### 10.4.1.6 Processor Bus Break Control and Address Mask

This register selects the Processor Bus match function to enable debug break or trace trigger. It also includes control bits to enable comparison as well as mask bits to exclude address bits from comparison.

Note that all processor break exceptions are imprecise.

**pbcam - Bus Break Control and Address Mask** **Offset = 0x030C**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | LAM[31:8] | | | | | | | | | | | | | | | | | | | | | | | | DC | DU | DIU | | OE | ▨ | ▨ | BE |
| Def. | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:8 | LAM | Address Mask. These bits specify the mask value for the 24 lower bits of the Processor Address register (PBA0[23..0]). Each bit corresponds to the same bit in PBA0.<br>0 Address bit is not masked, address bit is compared.<br>1 Address bit is masked, address bit is not compared. | R/W | UNPRED |
| 7 | DC | Data Store to Cached Area. This bit enables the comparison on Processor Address and Data Bus for Data Store to the Cached area.<br>0 Processor Address and Data is not compared for storing data to the Cached area.<br>1 Processor Address and Data is compared for storing data to the Cached area. | R/W | UNPRED |
| 6 | DU | Data Store To Uncached Area. This bit enables the comparison on Processor Address and Data Bus for Data Store to the uncached area.<br>0 Processor Address and Data is not compared for storing data into the un-cached area.<br>1 Processor Address and Data is compared for storing data into the un-cached area. | R/W | UNPRED |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 5:4 | DIU | Data or Instruction fetch or load from Uncached Area. These bits enable the comparison on Processor Address and Data Bus for Data or Instruction load and fetch from the un-cached area.<br>00 Processor Address and Data is not compared for loading data or fetching instruction from the un-cached area.<br>11 Processor Address and Data is compared for loading data or fetching instruction from the un-cached area.<br>Bits 5 and 4 were named ILUC and DFUC in the EJTAG 2.0.0 specification and were implemented separately for instruction and data fetches. | R | UNPRED |
| 3 | OE | Overlay Enable. When this bit is 1 and the processor physical address, masked by the HAM, UOAM and the LAM fields (all 36 bits of the address mask), matches the PHAB and PAB registers, then the memory request is redirected to the EJTAG Probe.<br>The processor bus break can not be used for normal break, function if the OLE bit is set, so BE must be set to 0. The behavior is otherwise undefined.<br>Overlay is only valid for memory regions. It is not valid for I/O or debug space and the behavior is unpredictable if addresses within this space are used. | R/W | 0 |
| 2 | — | Reserved, should be cleared. *This bit is called TE in the EJTAG 2.0.0 specification and was not implemented.* | R | 0 |
| 1 | — | Reserved, should be cleared. *This bit is called CBE in the EJTAG 2.0.0 specification and was not implemented.* | R | 0 |
| 0 | BE | Break Enable. This bit enables the Processor Bus break function.<br>0 Processor Bus break function is disabled.<br>1 Processor Bus break function is enabled.<br>If Break Enable is set and the processor physical address, masked by the HAM and the LAM fields (UOAM is only for overlay so bits 31:24 are not masked here), matches the PHAB and PAB registers, and the processor data bus matches the PDB register (masked by PDM), then a debug exception to the processor is generated.<br>The BS bit in the Processor Bus Break Status register is set and the DINT bit in the Debug Register is set. If the debug exception handler is already running (DM=‘1’), then the debug exception will not be taken until DM = 0.<br>This functionality is mutually exclusive to OLE so only one of OLE or BE should be set at any time. | R/W | 0 |

### 10.4.1.7 Processor High Address Bus Break

This register specifies the high order address for the processor address bus break.

**pha - Processor High Address Bus Break** **Offset = 0x0310**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | HA[3:0] | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:4 | — | Reserved, should be cleared. | R | 0 |
| 3:0 | HA | These bits map to the high physical address bits 35:31. | R/W | UNPRED |

### 10.4.1.8 Processor High Address Mask

This register specifies the high order address mask for the processor address bus break.

**pham - Processor High Address Mask** **Offset = 0x0314**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | HAM[3:0] | | | |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:4 | — | Reserved, should be cleared. | R | 0 |
| 3:0 | HAM | High Address Mask for address bits [35:31].<br>0 Data bit is not masked, data bit is compared<br>1 Data bit is masked, data bit is not compared | R/W | UNPRED |

### 10.4.2 EJTAG Test Access Port (TAP)

The EJTAG TAP contains the 5 TAP pins and a 16 state controller with a 5 bit instruction register.

Table 10-3 shows the 5-bit instructions supported by the Au1500.

**Table 10-3. EJTAG Instruction Register Values**

| Hex Value | Instruction | Function |
|---|---|---|
| 0x00 | EXTEST | Boundary Scan |
| 0x01 | IDCODE | Selects ID Register |
| 0x02 | SAMPLE | Boundary Scan Sample/Preload (IEEE JTAG Instruction) |
| 0x03 | IMPCODE | Selects Implementation Register |
| 0x04 | — | Reserved |
| 0x05 | — | This reserved register is for test mode HIZ - TRI-STATE all output pins and Select Bypass register. |
| 0x06 | — | This reserved register is for test mode CLAMP - IEEE Clamp pins and select bypass register. |
| 0x07 | — | Reserved |
| 0x08 | ADDRESS | Selects Address Register. |
| 0x09 | DATA | Selects Data Register. |
| 0x0A | CONTROL | Selects EJTAG Control Register. |
| 0x0B | ALL | Selects the Address, Data and EJTAG Control registers. |
| 0x0C | EJTAGBOOT | Makes the processor take a debug exception after reset. |
| 0x0D | NORMALBOOT | Makes the processor execute the reset handler after reset. |
| 0x0E-0x1B | — | Reserved |
| 0x1C | EJWATCH | Selects Watch register |
| 0x1D-0x1E | — | Reserved |
| 0x1F | BYPASS | Bypass mode |

#### 10.4.2.1 Device Identification (ID) Register

The Device ID register is a 32-bit read-only register that identifies the specific device implementing EJTAG.

**IDCODE - Device Identification** **TAP Instruction IDCODE**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | VER | | | | PNUM | | | | | | | | | | | | | | | | MANID | | | | | | | | | | | ▨ |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:28 | VER | Identifies the version of the device. | R | 0 |
| 27:12 | PNUM | Identifies the part number of the device. | R | 0x03E8 |
| 11:1 | MANID | Identifies the manufacturer ID code for the device. MANID[6:0] are derived from the last byte of the JEDEC code with the parity bit discarded. MANID[10:7] provides a binary count of the number of bytes in the JEDEC code that contain the continuation character (0x7F). When the number of continuations characters exceeds 15, these four bits contain the modulo-16 count. | R | 0x147 |
| 0 | — | This bit is reserved and should be written a 1. | R | 1 |

### 10.4.2.2 Implementation Register

The Implementation register is a 32-bit read-only register that identifies features implemented in this EJTAG compliant processor, mainly those accessible from the TAP.

**IMPCODE - Implementation** **TAP Instruction IMPCODE**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | VER |  |  | R3 | ▨ | ▨ | ▨ | DI | ▨ | AS |  | ▨ | ▨ | ▨ | ▨ | M16 | ▨ | ND | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | M32 |
| Def. | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:29 | EJTAGver | 1 EJTAG version 2.5 | R | 1 |
| 28 | R3 | 0 R3k privileged environment | R | 0 |
| 27:25 | — | Reserved, should be cleared. | R | 0 |
| 24 | DI | 0 DINT signal from the probe is not supported. | R | 0 |
| 23 | — | Reserved, should be cleared. | R | 0 |
| 22:21 | AS | 10 8-bit ASID | R | 10 |
| 20:17 | — | Reserved, should be cleared. | R | 0 |
| 16 | M16 | 0 No MIPS16 support | R | 0 |
| 15 | — | Reserved, should be cleared. | R | 0 |
| 14 | ND | 1 No EJTAG DMA support | R | 1 |
| 13:1 | — | Reserved, should be cleared. | R | 0 |
| 0 | MIPS32/64 | 0 32-bit processor | R | 0 |

### 10.4.2.3 Data Register

The read/write Data register is used for opcode and data transfers during processor accesses. The width of the Data register is 32 bits.

The value read in the Data register is valid only if a processor access for a write is pending, in which case the data register holds the store value. The value written to the Data register is only used if a processor access for a pending read is finished afterwards, in which case the data value written is the value for the fetch or load. This behavior implies that the Data register is not a memory location where a previously written value can be read afterwards.

**DATA** **TAP Instruction DATA or ALL**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | DATA[31:0] |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Def. | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31:0 | DATA | Data used by processor access | R/W | UNPRED |

### 10.4.2.4 Address Register

The read-only Address register provides the address for a processor access. The width of the register is 36 bits.

The value read in the register is valid if a processor access is pending, otherwise the value is undefined. The two LSBs of the register are used with the Psz field from the EJTAG Control register to indicate the size and data position of the pending processor access transfer. These bits are not taken directly from the address referenced by the load/store (i.e. these bits are encoded with Psz).

**ADDRESS** **TAP Instruction ADDRESS or ALL**

| Bit | 35 | 34 | 33 | 32 | 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[36:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Def | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 35:0 | Address | Address used by processor access | R | UNPRED |

### 10.4.2.5 EJTAG Control Register (ECR)

The 32-bit EJTAG Control Register (ECR) handles processor reset, Debug Mode indication, access start, finish, and size and read/write indication. The ECR also:

- Controls debug vector location and indication of serviced processor accesses.
- Allows debug interrupt request.
- Indicates processor low-power mode.

The EJTAG Control register is not updated/written in the Update-DR state unless the Reset occurred; that is RO (bit 31) is either already 0 or is written to 0 at the same time. This condition ensures proper handling of processor accesses after a reset.

Bits that are R/W in the register return their written value on a subsequent read, unless other behavior is defined. Internal synchronization hardware thus ensures that a written value is updated for reading immediately afterwards, even when the TAP controller takes the shortest path from the Update-DR to Capture-DR state.

Note: To ensure a write is successful to the PE, PT and EB bits when the processor is undergoing a clock change (for PLL lock/relock), the host must continue writing these bits until the write is verified by reading the change. Failure to do this could result in the write of these bits being lost.

Reset of the processor can be indicated in the TCK domain a number of TCK cycles after it is removed in the processor clock domain in order to allow for proper synchronization between the two clock domains.

**ECR - EJTAG Control Register** **TAP Instruction CONTROL or ALL**

| Bit | 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | RO | PSZ | | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | DZ | ▨ | ▨ | PRW | PA | ▨ | PR | PE | PT | ▨ | EB | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | DM | ▨ | ▨ | ▨ |
| Def. | 1 | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 31 | RO | Indicates if a processor reset has occurred since the bit was cleared:<br>0 No reset occurred<br>1 Reset occurred<br>The RO bit stays set as long as reset is applied.<br>This bit must be cleared to acknowledge that the reset was detected. The EJTAG Control register is not updated in the Update-DR state unless RO is 0 or written to 0 at the same time. This is in order to ensure correct handling of the processor access after reset. | R/W0 | 1 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 30:29 | PSZ | Indicates the size of a pending processor access, in combination with the Address register.<br>00 Byte<br>01 Halfword<br>10 Word<br>11 Triple<br>This field is valid only when a processor access is pending; otherwise, the read value is undefined. | R | UNPRED |
| 28:23 | — | Reserved, should be cleared. | R | 0 |
| 22 | DZ | Doze. Indicates if the processor is in a WAIT state:<br>0 Processor is not in a wait state.<br>1 Processor is in a wait state. | R | 0 |
| 21 | — | Reserved, should be cleared. *This bit is called Halt in the EJTAG 2.0.0 specification and was not implemented*. | R | 0 |
| 20 | — | Reserved, should be cleared. *This bit is called PerRst in the EJTAG 2.0.0 specification and was not implemented*. | R | 0 |
| 19 | PRW | Indicates read or write of a pending processor access.<br>0 Read processor access, for a fetch/load access<br>1 Write processor access, for a store access<br>This value is defined only when a processor access is pending. | R | UNPRED |
| 18 | PA | Indicates a pending processor access and controls finishing of a pending processor access. When read:<br>0 No pending processor access<br>1 Pending processor access<br>A write of 0 finishes a processor access if pending; otherwise operation of the processor is UNDEFINED if the bit is written to 0 when no processor access is pending. A write of 1 is ignored. | R/W0 | 0 |
| 17 | — | Reserved, should be cleared. | R | 0 |
| 16 | PR | Controls the processor reset.<br>0 No processor reset applied<br>1 Processor reset applied<br>Setting this bit to 1 will apply a processor reset. When this bit is read back it will always read a 0. Note that startup latencies should be observed when applying reset. | R/W | 0 |
| 15 | PE | Controls indication to the processor of whether the probe expects to handle accesses to EJTAG memory through servicing of processor accesses.<br>0 Probe does not service processor accesses<br>1 Probe will service processor accesses<br>The ProbEn bit is reflected as a read-only bit in the Debug Control Register (DCR) bit 0.<br>When a read from this bit shows a change, the new value has taken effect in the DCR. This handshake mechanism ensures that the setting from the TCK clock domain takes effect in the processor clock domain.<br>However, a change of the ProbEn prior to setting the EjtagBrk bit is ensured to affect execution of the debug handler due to the debug exception.<br>Not all combinations of ProbEn and ProbTrap are allowed.<br>Please see the previous note about writing this bit (in Section 10.4.2.5 "EJTAG Control Register (ECR)" on page 196). | R/W | Determined by EJTAG-BOOT |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 14 | PT | Controls location of the debug exception vector:<br>0 Normal memory 0xBFC0 0480<br>1 EJTAG memory 0xFF20 0200<br>When a read from this bit shows a change, the new value has taken effect in the DCR. This handshake mechanism ensures that the setting from the TCK clock domain takes effect in the processor clock domain.<br>However, a change of the ProbTrap prior to setting the EjtagBrk bit is ensured to affect execution of the debug handler due to the debug exception.<br>Not all combinations of ProbEn and ProbTrap are allowed.<br>Please see the previous note about writing this bit (in Section 10.4.2.5 "EJTAG Control Register (ECR)" on page 196). | R/W | Determined by EJTAG-BOOT |
| 13 | — | Reserved, should be cleared. | R | 0 |
| 12 | EB | Requests a debug interrupt exception to the processor when this bit is written as 1. This bit is cleared by hardware when the processor enters debug mode. If software then sets EB while the processor is already in debug, the request is not ignored but is delayed. That is, once the processor returns to normal mode, the pending debug exception request immediately sends the processor back into debug.<br>A write of 0 is ignored. The debug request restarts the processor clock if the processor was in a wait mode, which stopped the processor clock. The read value indicates a pending Debug Interrupt exception requested through this bit:<br>0 No pending Debug Interrupt exception requested through this bit<br>1 Pending Debug Interrupt exception<br>The read value can, but is not required to, indicate other pending DINT debug requests (for example, through the DINT signal).<br>Please see the previous note about writing this bit (in Section 10.4.2.5 "EJTAG Control Register (ECR)" on page 196). | R/W1 | Determined by EJTAG-BOOT |
| 11:4 | — | Reserved, should be cleared | R | 0 |
| 3 | DM | Indicates if the processor is in Debug Mode.<br>0 Processor is in Non-Debug Mode<br>1 Processor is in Debug Mode | R | 0 |
| 2:0 | — | Reserved, should be cleared. | R | 0 |

### 10.4.2.6 EJWatch Register (TAP Instruction EJWATCH)

The EJWatch register is used to enable CPU watchpoints to cause a debug exception. This functionality is unique to the Au1500.

**EJWATCH** **TAP Instruction EJWATCH**

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | WATCH |
| Def. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 7:3 | — | Reserved, should be cleared. | R | 0 |
| 2 | — | Reserved, should be cleared. This bit is the Global Scan test bit. | R | 0 |
| 1 | — | Reserved, should be cleared. This bit is a Test Mode bit. | R | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 0 | WATCH | This bit controls the debug functionality of the CPU watch register.<br>0 Normal Watch Exception Mode<br>1 Debug Watch Exception Mode<br>- Blocks writes to Watch register in non-debug mode<br>- Watch Exception will become debug exceptions with DEXCODE=23<br>- The PC will be saved in the DEPC (not in the EPC as with a normal watch exception).<br>Note that the Status, Cause, and EPC will not be affected by a debug watch exception when this bit is enabled. | R/W | 0 |

### 10.4.2.7 Bypass Register (TAP Instruction BYPASS)

The Bypass register is a one-bit read-only register, which provides a minimum shift path through the TAP. This register is also defined in IEEE 1149.1.

**BYPASS** **TAP Instruction BYPASS**

| | | |
|---|---|---|
| Bit | | 0 |
| | | BP |
| Def. | | 0 |

| Bits | Name | Description | R/W | Default |
|---|---|---|---|---|
| 0 | BP | Ignored on writes; returns zeros on reads. | R | 0 |

## 10.4.3 EJTAG TAP Hardware Considerations

The EJTAG interface consists of the signals listed in Table 10-4.

**Table 10-4. EJTAG Signals**

| Signal | Input/Output | Definition |
|---|---|---|
| TRST# | I | Asynchronous TAP reset |
| TDI | I | Test data input to the instruction or selected data registers. This signal will be sampled on the rising edge of TCK |
| TDO | O | Test data output from the instruction or data register. This signal will transition on the falling edge (valid on rising edge) of TCK |
| TMS | I | Control signal for TAP controller. This signal is sampled on the wising edge of TCK. |
| TCK | I | Control clock for updating TAP controller and shifting data through instruction or selected data register. |

Note that the EJTAG TAP signal TCK must always be less than 1/4 the System Bus (SBUS) clock speed for proper operation. In addition, termination as shown in EJTAG 2.5 spec must be followed.

# 11 Signal Descriptions

This section describes the external signals on the Au1500 processor.

In order to maximize the functionality of the Au1500 processor, many of the pins have multiple uses. Note that if a pin is configured for one use, any other functionality associated with that pin can not be utilized at the same time. In other words a pin can not be used as a general-purpose I/O signal at the same time it is assigned to a peripheral device. (See Section 8.3.1 "Pin Functionality" on page 168.)

Figure 11-1 on page 201 shows the external signals of the Au1500 processor. All signals are grouped according to their functional block. Signals that share a pin are listed with the multiplexed signal name in parentheses—the signal name shown in **bold** is the default.

**Note:** A signal with an “#” is *active-low*; that is, the signal is considered asserted (active) when low and negated when high. *Active-high* signals (no #) are considered asserted when high and negated when low.

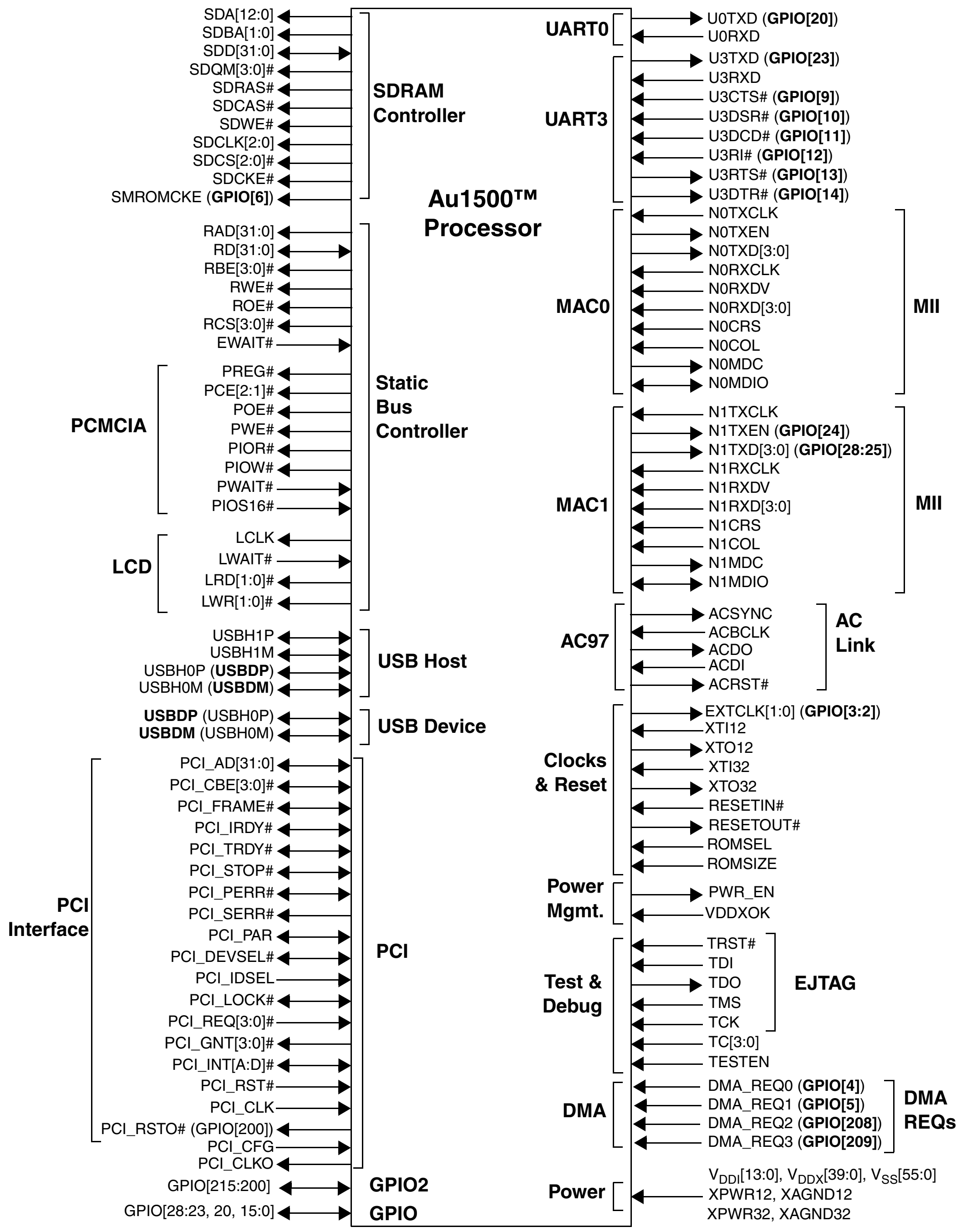


**Figure 11-1. External Signals**

Table 11-3 on page 203 gives a description of all external signals on the Au1500 processor. The signals have been grouped by functional block. Signals that require external termination are noted in the description. Table 11-3 also defines the default state of the signals during a hardware reset, a runtime reset, and Sleep. The abbreviations used for the signal types and the signal states are defined in Table 11-1 and Table 11-2.

**Table 11-1. Signal Type Abbreviations for Table 11-3**

| Signal Type | Definition |
|---|---|
| I | Input. Note that all *unused* input pins should be terminated low or high via direct connection to either ground or power. |
| O | Output |
| I/O | Bidirectional |
| Z | Tristatable |
| P | Power |
| G | Ground |

**Table 11-2. Signal State Abbreviations for Table 11-3**

| Signal State | Definition |
|---|---|
| 0 | Driven low |
| 1 | Driven high |
| IN | Signal is a input. |
| LV | If driven, an output signal continues to be driven at the last value before a reset or entering Sleep. |
| HIZ | TRI-STATE mode |
| ON | Clock remains on *if already enabled*. |
| DEP | Depends (Signal-specific explanations are provided in table footnotes.) |
| UN | Unpredictable |
| NC | Not connected |
| NA | Does not apply because this signal is not the default function coming out of a hardware or runtime reset. |

Note for Table 11-3 that the signal states shown in the far-right column are valid *during* Sleep. When waking from Sleep, the processor performs an internal system reset that produces the same signal behavior as a *runtime* reset with two exceptions:

- SDRAM interface behavior. During and after a runtime reset the SDRAM configuration *mode* registers retain their values to allow a transaction in progress to complete; the remaining SDRAM configuration registers revert to their default values. Waking from Sleep, however, *all* SDRAM configuration registers revert to their default values, and the interface behaves the same as when coming out of a *hardware* reset.
- PWR_EN behavior. During a runtime reset PWR_EN remains asserted. During Sleep, PWR_EN is negated. Waking from Sleep, PWR_EN is asserted according to the timing specified inSection 8.4.4.1 "Sleep Sequence and Timing" on page 174.

.

**Table 11-3. Signal Description**

| Signal | Type | Description | Reset | | During Sleep |
|---|---|---|---|---|---|
| | | | HW | Run Time | |
| **SDRAM Interface** | | | | | |
| SDA[12:0] | O | Address Outputs: A0-A12 are sampled during the ACTIVE command (row-address A0-A12) and READ/ WRITE command to select one location out of the memory array in the respective bank. The address outputs also provide the opcode during a LOAD MODE REGISTER command. | UN | UN | LV |
| SDBA[1:0] | O | Bank Address Outputs: SDBA1 and SDBA0 define to which bank the ACTIVE, READ, WRITE, or PRECHARGE command is being applied. | UN | UN | LV |
| SDD[31:0] | I/O | SDRAM Data Bus. During a hardware reset the SDRAM data bus cycles from low voltage to hi-Z and then low as follows:<br>0 after VDDXOK is asserted.<br>In TRI-STATE when $V_{DDI}$ is on and RESETOUT# is asserted.<br>0 after hardware reset sequence is complete. | (See description at left.) | HIZ | LV |
| SDQM[3:0]# | O | Input/Output Mask: SDQM# is an input mask signal for write accesses and an output enable signal for read accesses.<br>SDQM0# masks SDD[7:0].<br>SDQM1# masks SDD[15:8].<br>SDQM2# masks SDD[23:16].<br>SDQM3# masks SDD[31:24]. | 1 | 1 | LV |
| SDRAS# | O | Command Outputs. SDRAS#, SDCAS#, and SDWE# (along with SDCSn#) define the command being sent to the SDRAM rank. | 1 | 1 | LV |
| SDCAS# | O | | 1 | 1 | LV |
| SDWE# | O | | 1 | 1 | LV |
| SDCLK[2:0] | O | Clock output corresponding to each of the three chip selects. Clock speed is 1/2 System Bus (SBUS) frequency when corresponding SDCS*n*# is set to SDRAM, 1/4 SBUS frequency when corresponding SDCS*n*# is set to SMROM. | 0 | ON | LV<br>LV |
| SDCS[2:0]# | O | Programmable chip selects. | 1 | 1 | LV |
| SDCKE | O | Clock enable for SDRAM. | 0 | 1 | LV |
| SMROMCKE | O | Synchronous Mask ROM Clock Enable. Valid only when ROMSEL=1 and ROMSIZE=0. Must be pulled high if the system is booting from SMROM.<br>Muxed with GPIO[6]. If ROMSEL and ROMSIZE are configured to boot from synchronous mask ROM, the SMROMCKE signal is selected for the pin coming out of reset; otherwise, GPIO[6] is selected. | 1 | 1 | LV |

**Table 11-3. Signal Description (Continued)**

| Signal | Type | Description | Reset | | During Sleep |
|---|---|---|---|---|---|
| | | | HW | Run Time | |
| **Static Bus (SRAM/IO/PCMCIA/Flash/ROM/LCD) Interface - Common Signals** | | | | | |
| RAD[31:0] | O | Address Bus. | UN | UN | LV |
| RD[31:0] | I/O | Data Bus. | 0 | UN | LV |
| RBE[3:0]# | O | Byte Enable. RBE0# corresponds to RD[7:0], RBE1# corresponds to RD[15:8], RBE2# corresponds to RD[23:16], RBE3# corresponds to RD[31:24]. | 1 | 1 | LV |
| RWE# | O | Write Enable. | 1 | 1 | LV |
| ROE# | O | Output Enable. | 1 | 1 | LV |
| RCS[3:0]# | O | Programmable Chip Selects. RCS*n*# is not used when configured as a PCMCIA device. | 1 | 1 | LV |
| EWAIT# | I | Can be used to stretch the bus access time when enabled. This input is not recognized for chip selects configured as LCD or PCMCIA because these buses have their own wait mechanisms. | IN | IN | LV |
| **PCMCIA** | | | | | |
| PREG# | O | Register-Only Access Signal. | 1 | 1 | LV |
| PCE[2:1]# | O | Card Enables. | 1 | 1 | LV |
| POE# | O | Output Enable. | 1 | 1 | LV |
| PWE# | O | Write Enable. | 1 | 1 | LV |
| PIOR# | O | Read Cycle Indication. | 1 | 1 | LV |
| PIOW# | O | Write Cycle Indication. | 1 | 1 | LV |
| PWAIT# | I | Extend Cycle. Note that this signal should be tied high through a resistor when the PCMCIA interface is not used. | IN | IN | LV |
| PIOS16# | I | 16-bit Port Select. Note that this signal should be tied high through a resistor when the PCMCIA interface is not used. | IN | IN | LV |
| **LCD Controller Chip Interface** | | | | | |
| LCLK | O | Interface Clock. | 0 | 0 | LV |
| LWAIT# | I | Extend Cycle. Note that this signal should be tied high through a resistor when not used. | IN | IN | LV |
| LRD[1:0]# | O | Read Indicators. | 1 | 1 | LV |
| LWR[1:0]# | O | Write Indicators. | 1 | 1 | LV |
| **PCI Bus and Configuration** | | | | | |
| PCI_AD[31:0] | I/O | PCI Address/Data. | Z | Z | LV |
| PCI_CBE[3:0]# | I/O | PCI Bus Command and Byte Enables. | Z | Z | LV |
| PCI_FRAME# | I/O | PCI Cycle Frame. | Z | Z | LV |
| PCI_IRDY# | I/O | PCI Initiator Ready. | Z | Z | LV |
| PCI_TRDY# | I/O | PCI Target Ready. | Z | Z | LV |
| PCI_STOP# | I/O | PCI Target Stop. | Z | Z | LV |
| PCI_PERR# | I/O | PCI Parity Error. | Z | Z | LV |

**Table 11-3. Signal Description (Continued)**

| Signal | Type | Description | Reset HW | Reset Run Time | During Sleep |
|---|---|---|---|---|---|
| PCI_SERR# | O | PCI System Error. | Z | Z | LV |
| PCI_PAR | I/O | PCI Parity. | Z | Z | LV |
| PCI_DEVSEL# | I/O | PCI Device Select. | Z | Z | LV |
| PCI_IDSEL | I | PCI Initialization Device Select Input. | IN | IN | LV |
| PCI_LOCK# | I/O | PCI Lock for Atomic Operations. | Z | Z | LV |
| PCI_REQ[3:0]# | I | PCI Arbiter Bus Request Inputs. | IN | IN | LV |
| PCI_GNT[3:0]# | O | PCI Arbiter Bus Grant Outputs. | HIZ | 1 | LV |
| PCI_INT[A:D]# | I/O | PCI Interrupts. | IN | IN | LV |
| PCI_CLKO | O | PCI Clock Output.<br>**Note:** Requires an external 22-Ω series resistor placed within 0.5 inches of the part. | 0 | ON | LV |
| PCI_CLK | I | PCI Input Clock.<br>**Note:** Should be tied low through a resistor when the PCI bus is not used. | IN | IN | LV |
| PCI_RST# | I | PCI reset input<br>**Note:** Should be tied low through a resistor when the PCI bus is not used. | IN | IN | LV |
| PCI_RSTO# (GPIO[200]) | O | PCI Reset Output. Unlike other GPIOs, GPIO[200] defaults as an output with a zero voltage level coming out of reset. For this reason, GPIO[200] can be controlled by system software to act as a PCI reset output signal (labeled PCI_RSTO#) if needed.<br>**Note:** Should be tied low through a pull-down resistor. | HIZ | 0 | LV |
| PCI_CFG | I | PCI configuration. Determines the mode (satellite or host) for the PCI interface at hardware reset.<br>For satellite mode, tie PCI_CFG low.<br>For host mode, tie PCI_CFG high.<br>Note that PCI_CFG must not change state after system power up. | IN | IN | LV |
| **USB Host** | | | | | |
| USBH1P | I/O | Positive signal of differential USB host port 1 driver.<br>Requires an external 15 kohm pull-down resistor and ESD protection diode (transient voltage suppressor) to be USB 1.1 compliant.<br>**Note:** Requires an external 27 ohm series resistor placed within 0.5 inches of the part. | IN | IN | LV |
| USBH1M | I/O | Negative Signal of Differential USB Host Port 1 Driver. Requires an external 15 kohm pull-down resistor and ESD protection diode (transient voltage suppressor) to be USB 1.1 compliant.<br>**Note:** Requires an external 27 ohm series resistor placed within 0.5 inches of the part. | IN | IN | LV |

**Table 11-3. Signal Description (Continued)**

| Signal | Type | Description | Reset |  | During Sleep |
|---|---|---|---|---|---|
|  |  |  | HW | Run Time |  |
| USBH0P | I/O | Positive Signal of Differential USB Host Port 0 Driver. Requires an external 15 kohm pull-down resistor and ESD protection diode (transient voltage suppressor) to be USB 1.1 compliant.<br>**Note:** Requires an external 27ohm series resistor placed within 0.5 inches of the part.<br>Muxed with USBDP. USBDP is the default signal coming out of hardware reset, runtime reset, and Sleep. | NA | NA | LV |
| USBH0M | I/O | Negative Signal of Differential USB Host Port 0 Driver. Requires an external 15 kohm pull-down resistor and ESD protection diode (transient voltage suppressor) to be USB 1.1 compliant.<br>**Note:** Requires an external 27 ohm series resistor placed within 0.5 inches of the part.<br>Muxed with USBDM. USBDM is the default signal coming out of hardware reset, runtime reset, and Sleep. | NA | NA | LV |
| **USB Device** |  |  |  |  |  |
| USBDP | I/O | Positive Signal of Differential USB Device Driver. Requires a 1.5 kohm pull-up resistor to denote a full speed device. Also requires an external ESD protection diode (transient voltage suppressor) to be USB 1.1 compliant.<br>**Note:** Requires an external 27 ohm series resistor placed within 0.5 inches of the part.<br>Muxed with USBH0P. USBDP is the default signal coming out of hardware reset, runtime reset, and Sleep. | IN | IN | LV |
| USBDM | I/O | Negative Signal of Differential USB Device Driver. Requires an external ESD protection diode (transient voltage suppressor) to be USB 1.1 compliant.<br>**Note:** Requires an external 27 ohm series resistor placed within 0.5 inches of the part.<br>Muxed with USBH0M. USBDM is the default signal coming out of hardware reset, runtime reset, and Sleep. | IN | IN | LV |
| **UART0** |  |  |  |  |  |
| U0TXD | O | UART0 Transmit. Muxed with GPIO[20]. GPIO[20] is the default signal coming out of hardware reset, runtime reset, and Sleep. | NA | NA | LV |
| U0RXD | I | UART0 Receive. | IN | IN | IN |

Table 11-3. Signal Description (Continued)

| Signal | Type | Description | Reset | | During Sleep |
|---|---|---|---|---|---|
| | | | HW | Run Time | |
| **UART3** | | | | | |
| U3TXD | O | UART3 Transmit. Muxed with GPIO[23]. GPIO[23] is the default signal coming out of hardware reset, runtime reset, and Sleep. | NA | NA | LV |
| U3RXD | I | UART3 Receive. | IN | IN | LV |
| U3CTS# | I | Clear to Send. Muxed with GPIO[9]. GPIO[9] is the default signal coming out of hardware reset, runtime reset, and Sleep.<br>**Note:** For systems that use the UART3 interface without the optional modem control signals (**sys_pinfunc**[UR3]=0), the modem status interrupts must be disabled (**uart3_inten**[MIE]=0) to avoid false UART3 interrupts when using GPIO[9], GPIO[10], GPIO[11], or GPIO[12] as an input. | NA | NA | LV |
| U3DSR# | I | Data Set Ready. Muxed with GPIO[10]. GPIO[10] is the default signal coming out of hardware reset, runtime reset, and Sleep.<br>**Note:** For systems that use the UART3 interface without the optional modem control signals (**sys_pinfunc**[UR3]=0), the modem status interrupts must be disabled (**uart3_inten**[MIE]=0) to avoid false UART3 interrupts when using GPIO[9], GPIO[10], GPIO[11], or GPIO[12] as an input. | NA | NA | LV |
| U3DCD# | I | Data Carrier Detect. Muxed with GPIO[11]. GPIO[11] is the default signal coming out of hardware reset, runtime reset, and Sleep.<br>**Note:** For systems that use the UART3 interface without the optional modem control signals (**sys_pinfunc**[UR3]=0), the modem status interrupts must be disabled (**uart3_inten**[MIE]=0) to avoid false UART3 interrupts when using GPIO[9], GPIO[10], GPIO[11], or GPIO[12] as an input. | NA | NA | LV |
| U3RI# | I | Ring Indication. Muxed with GPIO[12]. GPIO[12] is the default signal coming out of hardware reset, runtime reset, and Sleep.<br>**Note:** For systems that use the UART3 interface without the optional modem control signals (**sys_pinfunc**[UR3]=0), the modem status interrupts must be disabled (**uart3_inten**[MIE]=0) to avoid false UART3 interrupts when using GPIO[9], GPIO[10], GPIO[11], or GPIO[12] as an input. | NA | NA | LV |
| U3RTS# | O | Request to Send. Muxed with GPIO[13]. GPIO[13] is the default signal coming out of hardware reset, runtime reset, and Sleep. | NA | NA | LV |

**Table 11-3. Signal Description (Continued)**

| Signal | Type | Description | Reset HW | Reset Run Time | During Sleep |
|---|---|---|---|---|---|
| U3DTR# | O | Data Terminal Ready. Muxed with GPIO[14]. GPIO[14] is the default signal coming out of hardware reset, runtime reset, and Sleep. | NA | NA | LV |
| **Ethernet Controller 0** | | | | | |
| N0TXCLK | I | Continuous clock input for synchronization of transmit data. 25 MHz when operating at 100 Mbps and 2.5 MHz when operating at 10 Mbps. | IN | IN | LV |
| N0TXEN | O | Indicates that the data nibble on N0TXD[3:0] is valid. | 0 | UN | LV |
| N0TXD[3:0] | O | Nibble wide data bus synchronous to N0TXCLK. For each N0TXCLK period in which N0TXEN is asserted, TXD[3:0] will have the data to be accepted by the PHY. While N0TXEN is de-asserted the data presented on TXD[3:0] should be ignored. | 0 | UN | LV |
| N0RXCLK | I | Continuous clock that provides the timing reference for the data transfer from the PHY to the MAC. N0RXCLK is sourced by the PHY. The N0RXCLK shall have a frequency equal to 25% of the data rate of the received signal data stream (typically 25 MHz at 100 Mbps and 2.5 MHz at 10 Mbps). | IN | IN | LV |
| N0RXDV | I | Indicates that a receive frame is in process and that the data on N0RXD[3:0] is valid. | IN | IN | LV |
| N0RXD[3:0] | I | RXD[3:0] is a nibble wide data bus driven by the PHY to the MAC synchronous with N0RXCLK. For each N0RXCLK period in which N0RXDV is asserted, RXD[3:0] will transfer four bits of recovered data from the PHY to the MAC. While N0RXDV is de-asserted, RXD[3:0] will have no effect on the MAC. | IN | IN | LV |
| N0CRS | I | N0CRS shall be asserted by the PHY when either transmit or receive medium is non Idle. N0CRS shall be deasserted by the PHY when both the transmit and receive medium are Idle. N0CRS is an asynchronous input. | IN | IN | LV |
| N0COL | I | N0COL shall be asserted by the PHY upon detection of a collision on the medium, and shall remain asserted while the collision condition persists. N0COL is an asynchronous input. The N0COL signal is ignored by the MAC when operating in the full duplex mode. | IN | IN | LV |
| N0MDC | O | N0MDC is sourced by the MAC to the PHY as the timing reference for transfer of information on the N0MDIO signal. N0MDC is an aperiodic signal that has no maximum high or low times. The minimum high and low times for N0MDC will be 160 ns each, and the minimum period for N0MDC will be 400 ns. | 0 | UN | LV |
| N0MDIO | I/O | N0MDIO is the bidirectional data signal between the MAC and the PHY that is clocked by N0MDC. | HIZ | UN | LV |

**Table 11-3. Signal Description (Continued)**

| Signal | Type | Description | Reset | | During Sleep |
|---|---|---|---|---|---|
| | | | HW | Run Time | |
| **Ethernet Controller 1** | | | | | |
| N1TXCLK | I | Continuous clock input for synchronization of transmit data. 25 MHz when operating at 100 Mbps and 2.5 MHz when operating at 10 Mbps. | IN | IN | LV |
| N1TXEN | O | Indicates that the data nibble on N1TXD[3:0] is valid. Muxed with GPIO[24]. GPIO[24] is the default signal coming out of hardware reset, runtime reset, and Sleep. | NA | NA | LV |
| N1TXD[3:0] | O | Nibble wide data bus synchronous to N1TXCLK. For each N1TXCLK period in which N1TXEN is asserted, TXD[3:0] will have the data to be accepted by the PHY. While N1TXEN is de-asserted the data presented on TXD[3:0] should be ignored.<br>Muxed with GPIO[28:25]. GPIO[28:25] are the default signals coming out of hardware reset, runtime reset, and Sleep. | NA | NA | LV |
| N1RXCLK | I | Continuous clock that provides the timing reference for the data transfer from the PHY to the MAC. N1RXCLK is sourced by the PHY. The N1RXCLK shall have a frequency equal to 25% of the data rate of the received signal data stream (typically 25 MHz at 100 Mbps and 2.5 MHz at 10 Mbps). | IN | IN | LV |
| N1RXDV | I | Indicates that a receive frame is in process and that the data on N1RXD[3:0] is valid. | IN | IN | LV |
| N1RXD[3:0] | I | RXD[3:0] is a nibble wide data bus driven by the PHY to the MAC synchronous with N1RXCLK. For each N1RXCLK period in which N1RXDV is asserted, RXD[3:0] will transfer four bits of recovered data from the PHY to the MAC. While N1RXDV is de-asserted, RXD[3:0] will have no effect on the MAC. | IN | IN | LV |
| N1CRS | I | N1CRS shall be asserted by the PHY when either transmit or receive medium is non Idle. N1CRS shall be deasserted by the PHY when both the transmit and receive medium are Idle. N1CRS is an asynchronous input. | IN | IN | LV |
| N1COL | I | N1COL shall be asserted by the PHY upon detection of a collision on the medium, and shall remain asserted while the collision condition persists. N1COL is an asynchronous input. The N1COL signal is ignored by the MAC when operating in the full duplex mode. | IN | IN | LV |
| N1MDC | O | N1MDC is sourced by the MAC to the PHY as the timing reference for transfer of information on the N1MDIO signal. N1MDC is an aperiodic signal that has no maximum high or low times. The minimum high and low times for N1MDC will be 160 ns each, and the minimum period for N1MDC will be 400 ns. | 0 | UN | LV |
| N1MDIO | I/O | N1MDIO is the bidirectional data signal between the MAC and the PHY that is clocked by N1MDC. | 0 | UN | LV |

**Table 11-3. Signal Description (Continued)**

| Signal | Type | Description | Reset | | During Sleep |
|---|---|---|---|---|---|
| | | | HW | Run Time | |
| **AC-Link** | | | | | |
| ACSYNC | O | Fixed Rate Sample Sync. | 0 | 0 | LV |
| ACBCLK | I | Serial Data Clock. | IN | IN | LV |
| ACDO | O | TDM Output Stream. | 0 | 0 | LV |
| ACDI | I | TDM Input Stream. | IN | IN | LV |
| ACRST# | O | Codec Reset. | 1 | 0 | LV |
| **EJTAG** | | | | | |
| TRST# | I | Asynchronous TAP Reset. | IN | IN | LV |
| TDI | I | Test Data Input to the instruction or selected data registers. Sampled on the rising edge of TCK. | IN | IN | LV |
| TDO | O | Test Data Output from the instruction or data register. Transitions occur on the falling edge (valid on rising edge) of TCK. | HIZ | UN | LV |
| TMS | I | Control signal for TAP controller. Sampled on the rising edge of TCK. | IN | IN | LV |
| TCK | I | Control clock for updating TAP controller and shifting data through instruction or selected data register. | IN | IN | LV |
| **Test** | | | | | |
| TC[3:0] | I | Test Clock Inputs (not used in typical application). Should be pulled low for normal operation. | IN | IN | LV |
| TESTEN | I | Test Enable (not used in typical applications). Should be pulled low for normal operation. | IN | IN | LV |
| **GPIO** | | | | | |
| GPIO[1:0] | IOZ | General Purpose I/O. | HIZ | DEP (Note 1) | LV |
| GPIO[3:2] | IOZ | General Purpose I/O. Muxed with EXTCLK[1:0]. GPIO[3:2] are the default signals coming out of hardware reset, runtime reset, and Sleep. | HIZ | DEP (Note 1) | LV |
| GPIO[4] | IOZ | General Purpose I/O. Can be configured as DMA_REQ0. | HIZ | DEP (Note 1) | LV |
| GPIO[5] | IOZ | General Purpose I/O. Can be configured as DMA_REQ1. | HIZ | DEP (Note 1) | LV |
| GPIO[6] | IOZ | General Purpose I/O. Muxed with SMROMCKE. If ROMSEL and ROMSIZE are configured to boot from synchronous mask ROM, the SMROMCKE signal is selected for the pin coming out of reset; otherwise, GPIO[6] is selected. | HIZ | DEP (Note 1) | LV |
| GPIO[7] | IOZ | General Purpose I/O. | HIZ | DEP (Note 1) | LV |
| GPIO[8] | IOZ | General Purpose I/O. | HIZ | DEP (Note 1) | LV |

**Table 11-3. Signal Description (Continued)**

| Signal | Type | Description | Reset HW | Reset Run Time | During Sleep |
|---|---|---|---|---|---|
| GPIO[9] | IOZ | General Purpose I/O. Muxed with U3CTS#. GPIO[9] is the default signal coming out of hardware reset, runtime reset, and Sleep.<br>**Note:** For systems that use the UART3 interface without the optional modem control signals (**sys_pinfunc**[UR3]=0), the modem status interrupts must be disabled (**uart3_inten**[MIE]=0) to avoid false UART3 interrupts when using GPIO[9], GPIO[10], GPIO[11], or GPIO[12] as an input. | HIZ | DEP (Note 1) | LV |
| GPIO[10] | IOZ | General Purpose I/O. Muxed with U3DSR#. GPIO[10] is the default signal coming out of hardware reset, runtime reset, and Sleep.<br>**Note:** For systems that use the UART3 interface without the optional modem control signals (**sys_pinfunc**[UR3]=0), the modem status interrupts must be disabled (**uart3_inten**[MIE]=0) to avoid false UART3 interrupts when using GPIO[9], GPIO[10], GPIO[11], or GPIO[12] as an input. | HIZ | DEP (Note 1) | LV |
| GPIO[11] | IOZ | General Purpose I/O. Muxed with U3DCD#. GPIO[11] is the default signal coming out of hardware reset, runtime reset, and Sleep.<br>**Note:** For systems that use the UART3 interface without the optional modem control signals (**sys_pinfunc**[UR3]=0), the modem status interrupts must be disabled (**uart3_inten**[MIE]=0) to avoid false UART3 interrupts when using GPIO[9], GPIO[10], GPIO[11], or GPIO[12] as an input. | HIZ | DEP (Note 1) | LV |
| GPIO[12] | IOZ | General Purpose I/O. Muxed with U3RI#. GPIO[12] is the default signal coming out of hardware reset, runtime reset, and Sleep.<br>**Note:** For systems that use the UART3 interface without the optional modem control signals (**sys_pinfunc**[UR3]=0), the modem status interrupts must be disabled (**uart3_inten**[MIE]=0) to avoid false UART3 interrupts when using GPIO[9], GPIO[10], GPIO[11], or GPIO[12] as an input. | HIZ | DEP (Note 1) | LV |
| GPIO[13] | IOZ | General Purpose I/O. Muxed with U3RTS#. GPIO[13] is the default signal coming out of hardware reset, runtime reset, and Sleep. | HIZ | DEP (Note 1) | LV |
| GPIO[14] | IOZ | General Purpose I/O. Muxed with U3DTR#. GPIO[14] is the default signal coming out of hardware reset, runtime reset, and Sleep. | HIZ | DEP (Note 1) | LV |
| GPIO[15] | IOZ | General Purpose I/O. | HIZ | DEP (Note 1) | LV |

Table 11-3. Signal Description (Continued)

| Signal | Type | Description | Reset | | During Sleep |
|---|---|---|---|---|---|
| | | | HW | Run Time | |
| GPIO[20] | IOZ | General Purpose I/O. Muxed with U0TXD. GPIO[20] is the default signal coming out of hardware reset, runtime reset, and Sleep. | HIZ | DEP (Note 1) | LV |
| GPIO[23] | IOZ | General Purpose I/O. Muxed with U3TXD. GPIO[23] is the default signal coming out of hardware reset, runtime reset, and Sleep. | HIZ | DEP (Note 1) | LV |
| GPIO[24] | IOZ | General Purpose I/O. Muxed with N1TXEN. GPIO[24] is the default signal coming out of hardware reset, runtime reset, and Sleep. | HIZ | DEP (Note 1) | LV |
| GPIO[28:25] | IOZ | General Purpose I/O. Muxed with N1TXD[3:0]. GPIO[28:25] are the default signals coming out of hardware reset, runtime reset, and Sleep. | HIZ | DEP (Note 1) | LV |
| GPIO[200] | IOZ | General Purpose I/O (secondary GPIO block). Unlike other GPIOs, GPIO[200] defaults as an output with a zero voltage level coming out of reset. For this reason, GPIO[200] can be controlled by system software to act as a PCI reset output signal (labeled PCI_RSTO#) if needed. **Note:** Should be tied low through a resistor. | HIZ | 0 | LV |
| GPIO[207:201] | IOZ | General Purpose I/O (secondary GPIO block). | HIZ | HIZ | LV |
| GPIO[208] | IOZ | General Purpose I/O (secondary GPIO block). Can be configured as DMA_REQ2. | HIZ | HIZ | LV |
| GPIO[209] | IOZ | General Purpose I/O (secondary GPIO block). Can be configured as DMA_REQ3. | HIZ | HIZ | LV |
| GPIO[215:210] | IOZ | General Purpose I/O (secondary GPIO block). | HIZ | HIZ | LV |
| **External Clocks** | | | | | |
| EXTCLK[1:0] | O | General Purpose External Clocks. Muxed with GPIO[3:2]. GPIO[3:2] are the default signals coming out of hardware reset, runtime reset, and Sleep. | NA | NA | LV |
| **System DMA Requests** | | | | | |
| DMA_REQ0 (GPIO[4]) | I | GPIO[4] can be configured as an external, system DMA request input. | HIZ | HIZ | LV |
| DMA_REQ1 (GPIO[5]) | I | GPIO[5] can be configured as an external, system DMA request input. | HIZ | HIZ | LV |
| DMA_REQ2 (GPIO[208]) | I | GPIO[208] can be configured as an external, system DMA request input. | HIZ | HIZ | LV |
| DMA_REQ3 (GPIO[209]) | I | GPIO[209] can be configured as an external, system DMA request input. | HIZ | HIZ | LV |

Table 11-3. Signal Description (Continued)

| Signal | Type | Description | Reset HW | Reset Run Time | During Sleep |
|---|---|---|---|---|---|
| **System Clocks and Reset** | | | | | |
| XTI12 | I | Internally Compensated 12 MHz (typical) Crystal Input.<br>**Note:** The termination depends on the application as follows:<br>Crystal—Connect crystal between XTI12 and XTO12.<br>Overdriven—Connect to external 12 MHz clock source and drive complementary to XTO12. | | | |
| XTO12 | O | Internally compensated 12 MHz (typical) Crystal Output.<br>**Note:** The termination depends on the application as follows:<br>Crystal—Connect crystal between XTI12 and XTO12.<br>Overdriven—Connect to external 12 MHz clock source and drive complementary to XTI12. | | | |
| XTI32 | I | Internally Compensated 32.768 KHz (typical) Crystal Input<br>**Note:** The termination depends on the application as follows:<br>Crystal—Connect crystal between XTI32 and XTO32.<br>Overdriven—Connect to external 32.768 KHz clock source through a series 10 KOHM resistor and drive complementary to XTO32.<br>Not used—Connect to $V_{DDX}$. | | | |
| XTO32 | O | Internally compensated 32.768 KHz (typical) crystal output<br>**Note:** The termination depends on the application as follows:<br>Crystal—Connect crystal between XTI32 and XTO32.<br>Overdriven—Connect to external 32.768 KHz clock source through a series 10 Kohm resistor and drive complementary to XTI32.<br>Not used—Connect to $V_{DDX}$. | | | |
| RESETIN# | I | CPU Reset Input. | IN | IN | LV |
| RESETOUT# | O | Buffered output of CPU reset input (RESETIN#). | 0 | 0 | 0 |
| ROMSEL | I | Determines if boot is from ROM or SMROM.<br>ROMSEL should be terminated appropriately as these signals should not change during runtime. | IN | IN | LV |

**Table 11-3. Signal Description (Continued)**

| Signal | Type | Description | Reset HW | Reset Run Time | During Sleep |
|---|---|---|---|---|---|
| ROMSIZE | I | Latched at the rising edge of reset to determine if ROM width is 16 or 32 bits. ROMSIZE should be terminated appropriately as these signals should not change during runtime. | IN | IN | LV |
| **Power Management** | | | | | |
| PWR_EN | O | Power Enable Output. This signal is intended to be used as the regulator enable for $V_{DDI}$ (core power). | 1 | 1 | 0 |
| VDDXOK | I | Input to signal that $V_{DDX}$ is stable. | IN | IN | LV |
| **Power/Ground** | | | | | |
| $V_{DDI}$ | P | Internal Core Voltage. **Note:** Follow the power supply layout guidelines in Section 12.11.2 "Decoupling Recommendations" on page 235. | | | |
| $V_{DDX}$ | P | External I/O voltage. **Note:** Follow the power supply layout guidelines in Section 12.11.2 "Decoupling Recommendations" on page 235. | | | |
| $V_{SS}$ | G | Ground | | | |
| XPWR12 | P | 12 MHz (typical) oscillator and PLL power. **Note:** Connect to $V_{DDX}$ through a 10 ohm resistor. In addition a 22 µF capacitor in parallel with a 0.01 µF capacitor should be placed from this pin to XAGND12. | | | |
| XAGND12 | G | 12 MHz (typical) oscillator and PLL ground. | | | |
| XPWR32 | P | 32.768 KHz (typical) oscillator and PLL power. Because XPWR32 powers other circuitry also, it should be connected even if the oscillator is not used. **Note:** Connect to $V_{DDX}$ through a 10 ohm resistor. In addition a 22 µF capacitor in parallel with a 0.01 µF capacitor should be placed from this pin to XAGND32. | | | |
| XAGND32 | G | 32.768 KHz (typical) oscillator and PLL ground | | | |

Note 1. Depends on **sys_trioutrd** and **sys_outputset**. During a runtime reset, **sys_pinfunc** returns to its default value, but the GPIO control registers **sys_trioutrd** and **sys_outputset** remain unchanged.

# 12 Electrical and Thermal Specifications

This chapter provides *preliminary* electrical specifications for the Au1500 processor, including the following:

- Absolute Maximum Ratings
- Thermal Characteristics
- DC Parameters
- AC Parameters
- Power-up, Reset, Sleep, and Idle Timing
- External Clock Specifications
- Crystal Specifications
- System Design Considerations

## 12.1 Absolute Maximum Ratings

Table 12-1 shows the absolute maximum ratings for the Au1500 processor. These ratings are stress ratings, operating at or beyond these ratings for extended periods of time may result in damage to the Au1500 processor.

Unless otherwise designated all voltages are relative to $V_{SS}$.

**Table 12-1. Absolute Maximum Ratings**

| Parameter | Description | Min | Max | Unit |
|---|---|---|---|---|
| $V_{DDI}$ | Core Voltage | $V_{SS}$ - 0.5 | 2 | V |
| $V_{DDX}$ | I/O Voltage | $V_{SS}$ - 0.5 | 3.6 | V |
| XPWR12, XPWR32 | Oscillator Voltage | $V_{SS}$ - 0.5 | 3.6 | V |
| $V_{IN}$ | Voltage Applied to Any Pin | $V_{SS}$ - 0.5 | $V_{DDX}$ + 0.5 | V |
| $T_{CASE}$ Commerical | Package Operating Temperature | 0 | 85 | °C |
| $T_{CASE}$ Industrial (333 MHz part only) | Package Operating Temperature | -40 | 100 | °C |
| $T_S$ | Storage Temperature | -40 | 125 | °C |

## 12.2 Undershoot

The minimum DC voltage on input or I/O pins is -0.5V. However, during voltage transitions, the device can tolerate undershoot to -2.0V for up to 20 ns, as shown in Figure 12-1.

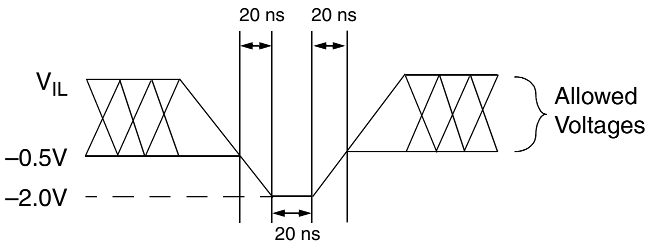


**Figure 12-1. Voltage Undershoot Tolerances for Input and I/O Pins**

## 12.3 Overshoot

The maximum DC voltage on input or I/O pins is ($V_{DDX}$ + 0.5) V. However, during voltage transitions, the device can tolerate overshooting $V_{DDX}$ to ($V_{DDX}$ + 2.0) V for up to 20 ns, as shown in Figure 12-2.

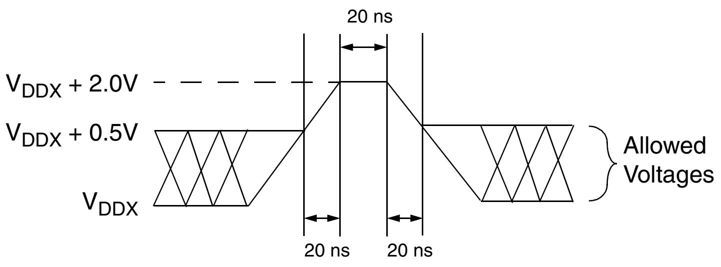


**Figure 12-2. Voltage Overshoot Tolerances for Input and I/O Pins**

## 12.4 Thermal Characteristics

Table 12-2 shows the thermal characteristics for the Au1500 processor.

**Table 12-2. Thermal Characteristics with Changing Air Flow Conditions**

| Parameter | Still Air (Note 1) | 50 lfpm (Note 2) (0.25 m/s) | 100 lfpm (0.5 m/s) | 150 lfpm (0.75 m/s) | 200 lfpm (1.0 m/s) | Unit |
|---|---|---|---|---|---|---|
| $\Theta_{JA}$ | 33.1 | 31.8 | 30.7 | 30.0 | 28.6 | °C/W |
| $\Psi_{JT}$ | 5.0 | | | | | °C/W |

Note 1. Measured without forced air—natural convection only.

Note 2. Linear feet per minute.

## 12.5 DC Parameters

Table 12-3 shows the DC parameters for the Au1500 processor. Unless otherwise designated all voltages are relative to $V_{SS}$.

The operating requirements for the power supply voltages ($V_{DDX}$ and $V_{DDI}$) are given in the sections describing the DC characteristics for the different operating frequencies, beginning with Table 12-4.

**Table 12-3. DC Parameters**

| Parameter | Description | Min | Nominal | Max | Unit |
|---|---|---|---|---|---|
| $V_{IHX}$ | Input High Voltage | 2.4 | | | V |
| $V_{ILX}$ | Input Low Voltage | | | 0.2 * $V_{DDX}$ | V |
| $V_{OHX}$ @ 2 mA | Output High Voltage | 0.8 * $V_{DDX}$ | | | V |
| $V_{OLX}$ @ 2 mA | Output Low Voltage | | | 0.2 * $V_{DDX}$ | V |
| $I_I$ | Input Leakage Current | | | 5 | µA |
| $C_{IN}$ (Note 1) | Input Capacitance | | 5 | | pF |
| IXPWR12 (Note 2) | XPWR12 Current | | 1 | 3 | mA |
| IXPWR32 (Note 2) | XPWR32 Current | | 1 | 3 | mA |

Note 1. This parameter is by design and not tested.

Note 2. Does not apply during Sleep.

### 12.5.1 Power and Voltage for 333, 400, and 500 MHz Rated Parts

The tables that follow give the voltage and power parameters for the individual MHz rated parts.

**Table 12-4. Voltage and Power Parameters for 333 MHz Part**

| Parameter | Min | Typ | Max | Unit |
|---|---|---|---|---|
| $V_{DDI}$ | 1.4 | 1.5 | 1.6 | V |
| $V_{DDX}$, XPWR12, XPWR32 (Note 1) | 3.0 | 3.3 | 3.6 | V |
| Power: $_{VDDI}$ | | 340 | 755 (Note 2) | mW |
| Power: $V_{DDX}$ | | 60 | 290 (Note 2) | mW |
| IDLE Power (Note 3) | | 176 | | mW |
| Sleep Current ($V_{DDI}$ = $V_{SS}$) | | | 50 | µA |

Note 1. XPWR12 and XPWR32 should be connected to $V_{DDX}$. For a description of this circuit connection, see the entries for XPWR12 and XPWR32 in Table 11-3 "Signal Description" on page 203.

Note 2. While the maximum power numbers should be used when specifying a regulator for a system, the numbers are well above the typical power consumption because none of the power-saving design features (such as IDLE, or the automatic SBUS divider) are enabled. Note that because the particular application software and external loading affect the power consumption on a given system design, certain conditions may exist which could cause the maximum power consumption to be different than shown.

Note 3. *IDLE power* is the power measured when the processor core is in the IDLE0 state. (IDLE0 maintains cache coherency by snooping the SBUS; IDLE1 does not snoop the bus. Because caches are turned off during the IDLE1 state, IDLE1 consumes less power than IDLE0.) Typically the IDLE state is entered during an operating system's wait loop in which the core has no processes to run. While the processor core is in IDLE, clocks to the core are gated off; however, all registers retain their values, and the peripherals, DMA engine, and the interrupts remain active so that the system is still functional.

**Table 12-5. Voltage and Power Parameters for 400 MHz Part**

| Parameter | Min | Typ | Max | Unit |
|---|---|---|---|---|
| $V_{DDI}$ | 1.4 | 1.5 | 1.6 | V |
| $V_{DDX}$, XPWR12, XPWR32 (Note 1) | 3.0 | 3.3 | 3.6 | V |
| Power: $V_{DDI}$ | | 595 | 870 (Note 2) | mW |
| Power: $V_{DDX}$ | | 105 | 340 (Note 2) | mW |
| IDLE Power (Note 3) | | 205 | | mW |
| Sleep Current ($V_{DDI} = V_{SS}$) | | | 50 | µA |

Note 1. XPWR12 and XPWR32 should be connected to $V_{DDX}$. For a description of this circuit connection, see the entries for XPWR12 and XPWR32 in Table 11-3 "Signal Description" on page 203.

Note 2. While the maximum power numbers should be used when specifying a regulator for a system, the numbers are well above the typical power consumption because none of the power-saving design features (such as IDLE, or the automatic SBUS divider) are enabled. Note that because the particular application software and external loading affect the power consumption on a given system design, certain conditions may exist which could cause the maximum power consumption to be different than shown.

Note 3. *IDLE power* is the power measured when the processor core is in the IDLE0 state. (IDLE0 maintains cache coherency by snooping the SBUS; IDLE1 does not snoop the bus. Because caches are turned off during the IDLE1 state, IDLE1 consumes less power than IDLE0.) Typically the IDLE state is entered during an operating system's wait loop in which the core has no processes to run. While the processor core is in IDLE, clocks to the core are gated off; however, all registers retain their values, and the peripherals, DMA engine, and the interrupts remain active so that the system is still functional.

**Table 12-6. Voltage and Power Parameters for 500 MHz Part**

| Parameter | Min | Typ | Max | Unit |
|---|---|---|---|---|
| $V_{DDI}$ | 1.71 | 1.8 | 1.89 | V |
| $V_{DDX}$, XPWR12, XPWR32 (Note 1) | 3.0 | 3.3 | 3.6 | V |
| Power: $V_{DDI}$ | | 1080 | 1625 (Note 2) | mW |
| Power: $V_{DDX}$ | | 120 | 450 (Note 2) | mW |
| IDLE Power (Note 3) | | 293 | | mW |
| Sleep Current ($V_{DDI} = V_{SS}$) | | | 50 | µA |

Note 1. XPWR12 and XPWR32 should be connected to $V_{DDX}$. For a description of this circuit connection, see the entries for XPWR12 and XPWR32 in Table 11-3 "Signal Description" on page 203.

Note 2. While the maximum power numbers should be used when specifying a regulator for a system, the numbers are well above the typical power consumption because none of the power-saving design features (such as IDLE, or the automatic SBUS divider) are enabled. Note that because the particular application software and external loading affect the power consumption on a given system design, certain conditions may exist which could cause the maximum power consumption to be different than shown.

Note 3. *IDLE power* is the power measured when the processor core is in the IDLE0 state. (IDLE0 maintains cache coherency by snooping the SBUS; IDLE1 does not snoop the bus. Because caches are turned off during the IDLE1 state, IDLE1 consumes less power than IDLE0.) Typically the IDLE state is entered during an operating system's wait loop in which the core has no processes to run. While the processor core is in IDLE, clocks to the core are gated off; however, all registers retain their values, and the peripherals, DMA engine, and the interrupts remain active so that the system is still functional.

## 12.6 AC Parameters

This section describes the AC parameters for I/O devices in the Au1500 processor. Each class of output signal has different capacitive loads. As the capacitance on the load increases the propagation delay will increase. These specifications assume the maximum capacitive load to be 50 pF for all I/O signals other than the SDRAM interface.

The timing of those signals which have synchronous relationships or have a defined requirement are given. The timing diagrams are shown to illustrate the timing only and should not necessarily be interpreted as the functional timing of the port.

It is assumed that the timing and/or functionality of the protocol related to the port is adhered to by the external system. The protocol timing is not necessarily presented here and the appropriate section or specification should be referenced for complete functional timing parameters.

Timing measurements are made from 50% threshold to 50% threshold.

Certain timing parameters are based off of the internal System Bus (SBUS) clock. When this is the case the symbol $T_{sys}$ is used. $T_{sys}$ is defined in nanoseconds as:

$T_{sys}$ = SD/CPU

The symbol CPU should be interpreted as the CPU clock speed in MHz as set by the CPU PLL. See Section 8.1 "Clocks" on page 154 for details. The symbol SD is the SBUS divider. See Section 8.4 "Power Management" on page 172 for details.

## 12.6.1 SDRAM Timing and Loading

The SDRAM controller loading limits are as follows:

- SDRAM outputs *excluding the clocks and chip-selects* can support a maximum capacitive load of 35 pF (six 5 pF gate loads and 5 pF representing the trace).
- Each *clock* and each *chip-select* supports a maximum capacitive load of 15 pF (two 5 pF gate loads and 5 pF representing the trace).

The SDRAM is a high speed interface. Reflection and propagation delays should be accounted for in the system design. As a general rule of thumb, unterminated etches should be kept to 6 inches or less.

**Table 12-7. SDRAM Controller Interface**

| Signal | Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|---|
| SDCLK*n* | Tsdclk | SDCLK*n* Clock Cycle | 2 * Tsys | | ns |
| SDCS*n*#, SDRAS#, SDCAS#, SDWE#, SDBA[1:0], SDA[12:0], SDQM[3:0]#, SDD[31:0] (output) | Tsdd | Delay from SDCLK*n* | Tsdclk/4 – 1.5 | Tsdclk/4 + 2 | ns |
| SDD[31:0] (input) | Tsdsu | Data setup to SDCLK*n* | 1.8 | | ns |
| SDD[31:0] (input) | Tsdh | Data hold from SDCLK*n* | 1.7 | | ns |

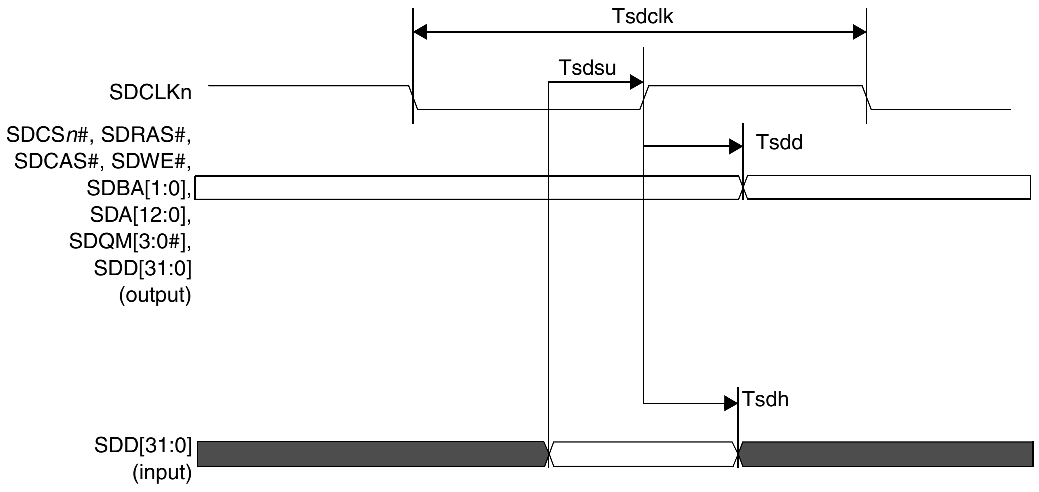


**Figure 12-3. SDRAM Timing**

### 12.6.2 Static Bus Controller Timing

The timing presented in registers **mem_sttime*n*** are not presented here. The parameters in these registers are presented in a certain number of clock cycles and are accurate to within ±2 ns.

**Table 12-8. Static RAM, I/O Device and Flash Timing**

| Signal | Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|---|
| RBE[3:0]#, ROE#, RAD[31:0], burst-size[2:0] | Trcd | Delay from RCSn#. | -2 | +2 | ns |
| RD[31:0] (read) | Trsu | Data setup to RCS*n*#.<br>Note that Trsu does not apply when EWAIT# is used to extend the cycle. | 15 | | ns |
| RD[31:0] (read) | Trsue | Data setup to EWAIT#. Note that Trsue applies only when EWAIT# is used to extend the cycle. | 0 | | ns |
| RD[31:0] (read) | Trh | Data hold from RCSn#. | 0 | | ns |
| RD[31:0] (write) | Trod | Delay from RWE# to data out | -2 | 2 | ns |
| EWAIT# | Trwsu | EWAIT# setup to RCS*n*# for reads, or RWE# for writes.<br>If EWAIT# does not meet this setup time the cycle will not be held. | 3 * Tsys + 15 | | ns |
| RCS# (reads), RWE# (writes) | Trwd | Delay from EWAIT#. | 2 * Tsys | 3 * Tsys + 15 | |
| burstsize[2:0] | Trbd | Delay from RCS*n*#. | | Tsys + 2 | |

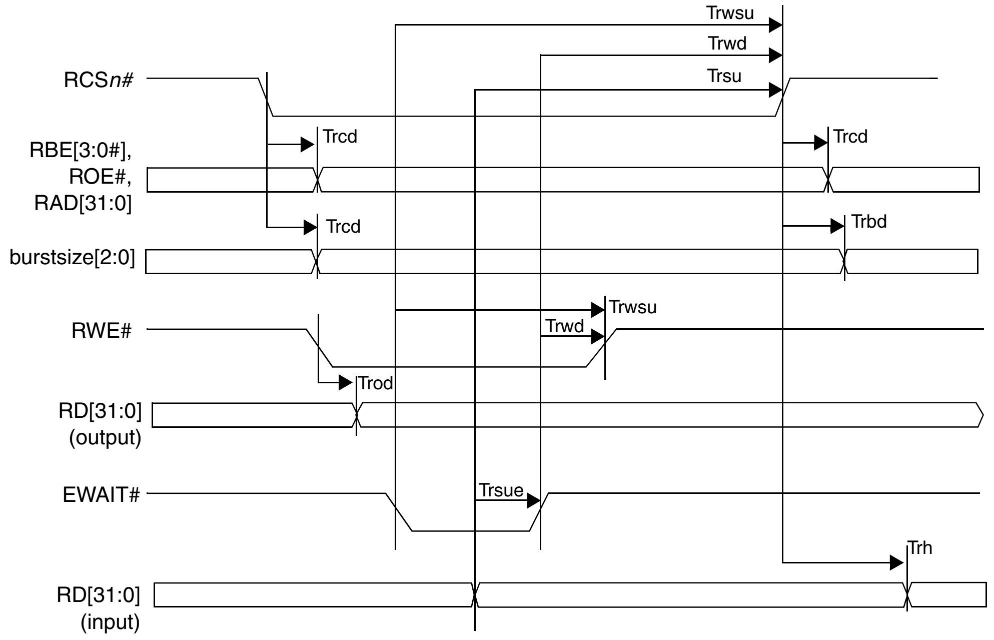


**Figure 12-4. Static RAM, I/O Device and Flash Timing**

**Table 12-9. PCMCIA Timing**

| Signal | Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|---|
| PREG#, RAD[31:0], RD[31:0] (output) | Tpcd | Delay from PCE*n*# | -2 | +2 | ns |
| PIOS16# | Tpios | PIOS16# setup to PIOR#, PIOW# | 4 * Tsys + 15 | | ns |
| PIOS16# | Tpioh | PIOS16# hold from PIOR#, PIOW# | 0 | | ns |
| ROE# | Tpoed | ROE# delay from POE#, PIOR# | -2 | +2 | ns |
| RD[15:0] (input) | Tpsu | Data setup to POE#, PIOR#. Note that Tpsu does not apply when PWAIT# is used to extend the cycle. | Tsys + 15 | | ns |
| RD[15:0] (input) | Tpsup | Data setup to PWAIT#. Note that Tpsup applies only when PWAIT# is used to extend the cycle. | 0 | | ns |
| RD[31:0] | Tph | Data hold from POE#, PIOR# | 0 | | ns |
| PWAIT# | Tpwsu | PWAIT# setup to POE#, PWE#, PIOR#, PIOW#<br>If PWAIT# does not meet this setup time the cycle will not be held | 4 * Tsys + 15 | | ns |
| POE#, PWE#, PIOR#, PIOW# | Tpwd | POE#, PWE#, PIOR#, PIOW# delay from PWAIT#. | 3 * Tsys | 4 * Tsys + 15 | ns |

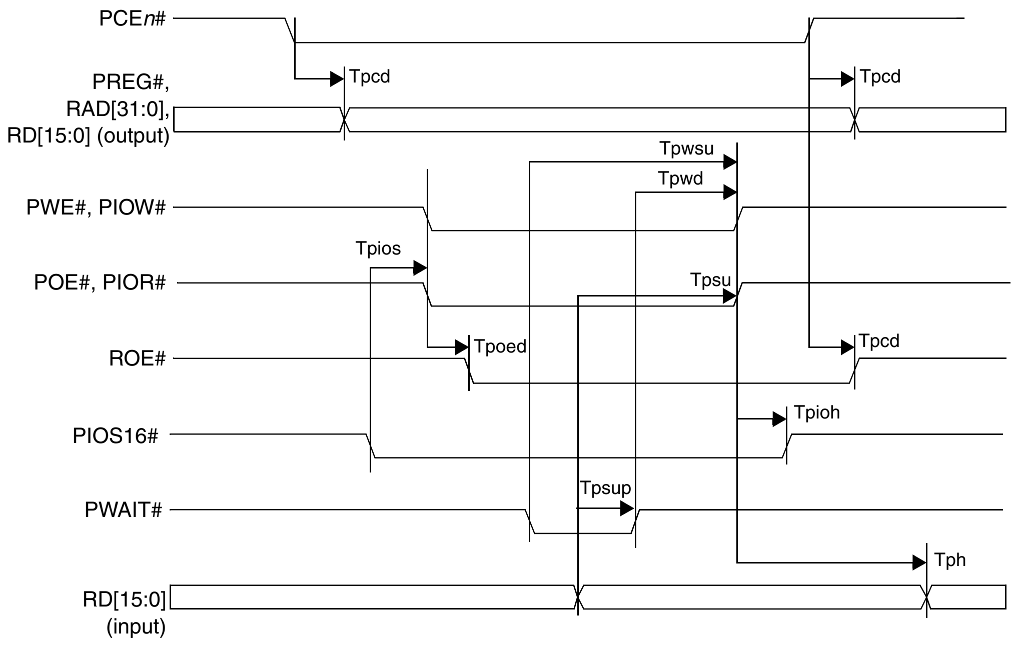


**Figure 12-5. PCMCIA Host Adapter Timing**

Table 12-10. LCD Timing

| Signal | Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|---|
| LCLK | Tlclk | LCLK Period. This parameter is programmed in **mem_stcfg0**[D5]. | Tsys * 4 | Tsys * 5 | ns |
| RCS*n*#, RAD[31:0], RD[15:0] (output) | Tlcd | Delay from LCLK | -2 | 2 | ns |
| RD[15:0] (input) | Tlsu | Data setup to LRD*n*# | 15 | | ns |
| RD[15:0] (input) | Tlh | Data hold from LRD*n*# | 0 | | ns |
| LWAIT# | Tlwsu | LWAIT# setup to LRD*n*# for reads, or LWR*n*# for writes. If LWAIT# does not meet this setup time the cycle will not be held. | 3 * Tsys + Tlclk + 15 | | ns |
| LRD*n*# (reads), LWR*n*# (writes) | Tlwd | Delay from LWAIT# | 2 * Tsys | 3 * Tsys + Tlclk + 15 | ns |

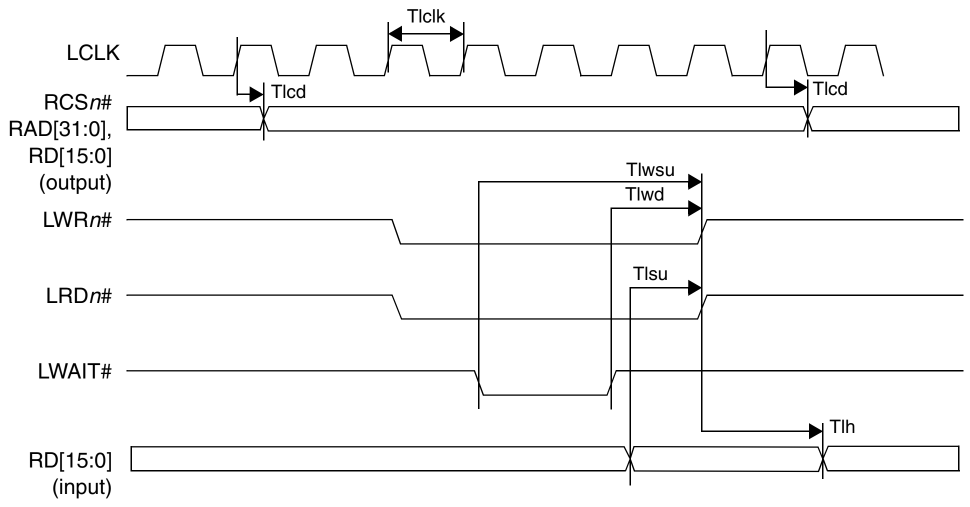


Figure 12-6. LCD Interface Timing

## 12.6.3 PCI Timing and Loading

The PCI controller conforms to the PCI 2.2 Local Bus Specification for both the 33 MHz and 66 MHz options. The timing shown in Table 12-11 assumes a maximum of four PCI loads with each slot counting as one load. Trace lengths of PCI signals should be kept to a maximum of 9 inches.

**Table 12-11. PCI Controller Interface**

| | | | 66 MHz | | 33 MHz | | |
|---|---|---|---|---|---|---|---|
| **Signal** | **Symbol** | **Parameter** | **Min** | **Max** | **Min** | **Max** | **Unit** |
| PCI_CLK | Tpclk | PCI_CLK clock cycle | 15 | 30 | 30 | • | ns |
| All outputs | Tpcd | Delay from PCI_CLK | 2 | 6 | 2 | 11 | ns |
| All inputs | Tpsu | Data setup to PCI_CLK | 3 | | 7 | | ns |
| | Tpdh | Data hold from PCI_CLK | 0 | | 0 | | ns |

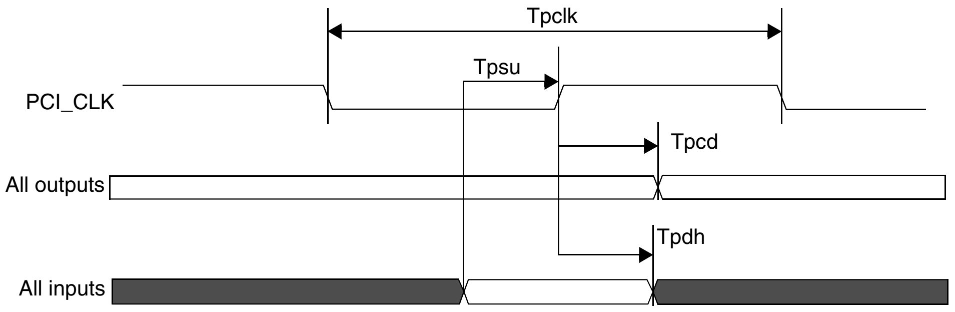


**Figure 12-7. PCI Timing**

### 12.6.4 GPIO Input Timing Requirements

### 12.6.5 GPIO Input Edge Rate

For level-sensitive GPIO inputs, edge rates as slow as 5 ms can be used. Note that no hysteresis is used on the inputs so for edge-sensitive inputs (such as clocks and edge-triggered interrupts) use a 20-ns (or faster) edge rate to ensure that noise does not cause false edges as the signal transitions through the threshold region.

### 12.6.6 GPIO Interrupt Timing

For system designs using GPIO signals as level-triggered interrupts, the signal level must be stable for at least 10 ns in order for a signal state change to be detected. See Table 12-12 and Figure 12-8.

**Table 12-12. GPIO Timing for Interrupts**

| Signal | Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|---|
| GPIO[n] | Tmin | Minimum high or low time for interrupt.<br>The level is programmable. This timing reflects the minimum active period for the level programmed. | 10 | | ns |

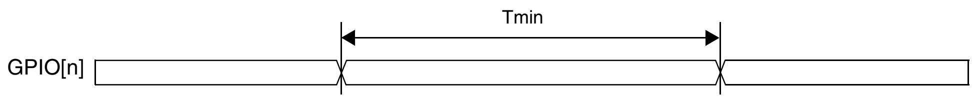


**Figure 12-8. GPIO Interrupt Timing**

## 12.6.7 Peripheral Timing

This section contains the electrical timing specifications for the integrated peripherals.

### 12.6.7.1 Ethernet MII Timing

**Table 12-13. Ethernet MII Timing**

| Signal | Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|---|
| N0TXCLK, N0RXCLK, N1TXCLK, N1RXCLK | Teth | Ethernet transmit/receive clock cycle time (25% of data rate) | 40 ± 100ppm (100 Mbps) 400 ± 100ppm (10 Mbps) | | ns |
| | | Ethernet transmit/ receive clock duty cycle | 35 | 65 | % |
| N0TXEN, N0TXD[3:0], N1TXEN, N1TXD[3:0] | Ted | Delay from TXCLK to TXEN, TXD[3:0] | 0 | 25 | ns |
| N0RXD[3:0], N0RXDV N1RXD[3:0], N1RXDV | Tesu | Setup time before RXCCLK for RXD, and RXDV | 10 | | ns |
| | Teh | Hold time from RXCLK for RXD, and RXDV | 10 | | ns |
| N0MDC, N1MDC | Tmdc | MDC cycle time | SBUS clock / 160 | | |
| | | MDC duty cycle | 40 | 60 | % |
| N0MDIO, N1MDIO | Tmdd | Delay from MDC to MDIO | 0 | 300 | ns |
| | Tmsu | Setup time before MDC for MDIO | 10 | | ns |
| | Tmh | Hold time from MDC for MDIO | 10 | | ns |
| | Tmz | Delay from MDC to MDIO TRI-STATE | 0 | 300 | ns |
| N0CRS, N0COL, N1CRS, N1COL | Ta | Minimum active time | | | |

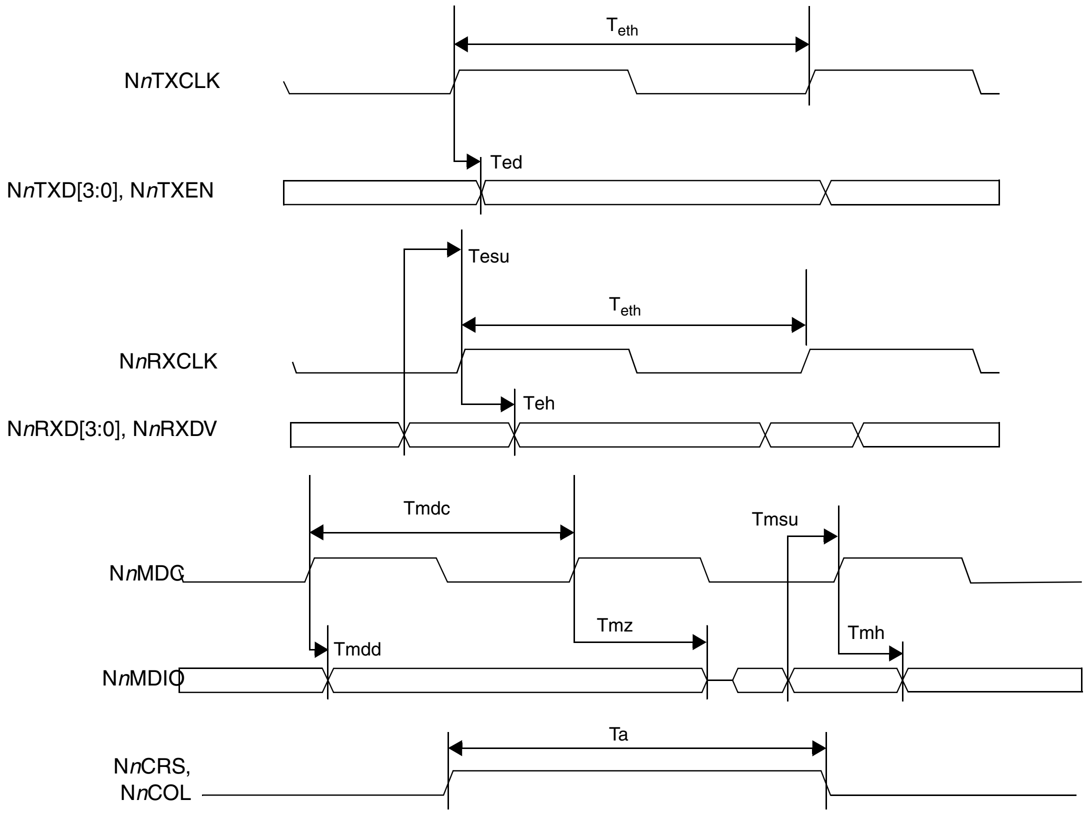


**Figure 12-9. Ethernet MII Timing Diagram**

### 12.6.7.2 AC97 Timing

**Table 12-14. AC-Link Interface Timing**

| Signal | Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|---|
| ACBCLK | Tabc | AC97 bit clock cycle time | 12.288 (typical) | | MHz |
| | Tabh | AC97 bit clock high time | 36 | 45 | ns |
| | Tabl | AC97 bit clock low time | 36 | 45 | ns |
| ACSYNC | Tacs | AC97 sync cycle | 48 (typical) | | KHz |
| | Tacsh | AC97 sync high time | 1.3 (typical) | | µs |
| | Tacsl | AC97 sync low time | 19.5 (typical) | | µs |
| ACSYNC ACDO ACDI | Tad | Delay from ACBCLK to ACSYNC and ACDO on output | | 15 | ns |
| | Tasu | Setup before ACBCLK for ACDI | 10 | | ns |
| | Tah | Hold after ACBCLK for ACDI | 10 | | ns |

**Note:** ACRST# is an asynchronous signal controlled by software through the register **ac97_config**. It has no relationship to the other AC97 signals.

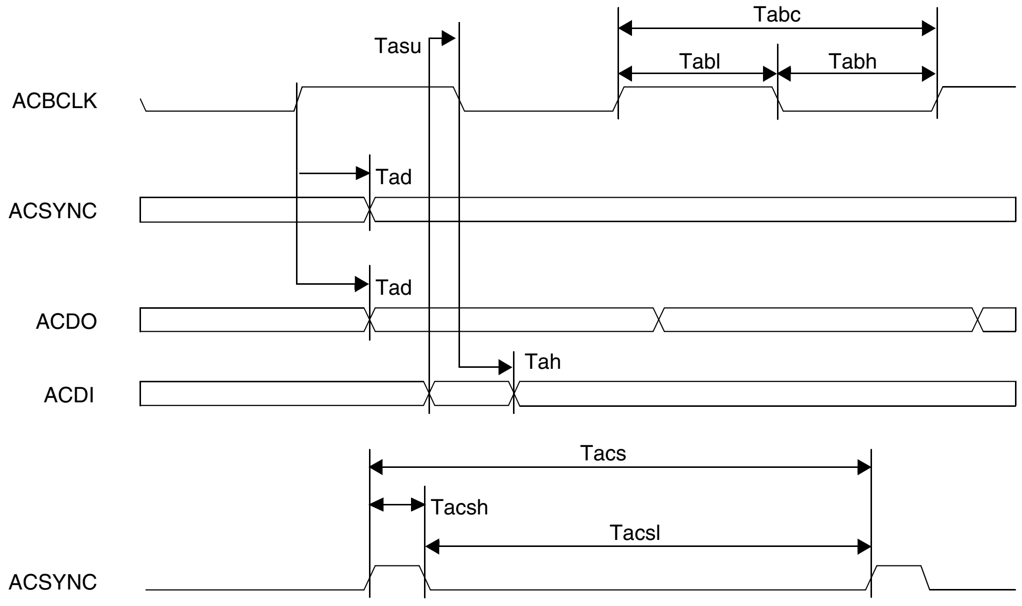


**Figure 12-10. AC-Link Timing Diagram**

### 12.6.7.3 EJTAG Interface Timing

**Table 12-15. EJTAG Interface Timing**

| Signal | Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|---|
| TCK | Tec | EJTAG TCK cycle time | 40 | | ns |
| | Tech | TCK high time | 10 | | ns |
| | Tecl | TCK low time | 10 | | ns |
| TMS, TDI | Tesu | Setup before TCK for TMS and TDI | 5 | | ns |
| | Teh | Hold after TCK for TMS and TDI | 3 | | ns |
| TDO | Teco | Delay from TCK to TDO on output | | 15 | ns |
| | Tecz | Delay from TCK to TDO TRI-STATE | | 15 | ns |
| TRST# | Trstl | TRST# low time | 25 | | ns |

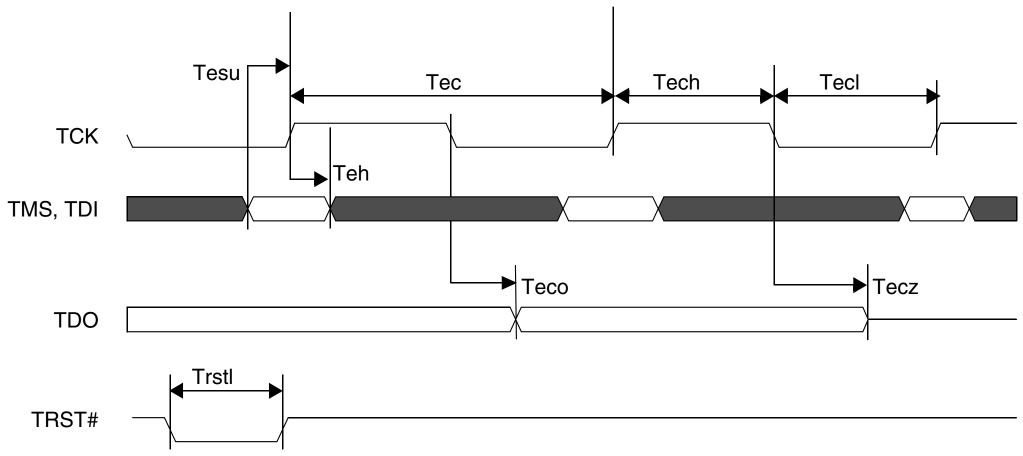


**Figure 12-11. EJTAG Timing Diagram**

## 12.7 Power-up and Reset Timing

This section provides the timing specifications for the power-up sequence, and the hardware and runtime reset sequences. (See Section 9.0 "Power-up, Reset and Boot" on page 180 for functional descriptions of the sequences.)

### 12.7.1 Power-up Sequence Timing

**Table 12-16. Power-up Timing Parameters**

| Parameter | Description | Min | Max | Unit |
|---|---|---|---|---|
| Tvo | $V_{DDX}$ at 90% of nominal to VDDXOK asserted | 0 | | ns |
| Tpen | VDDXOK asserted to PWR_EN driven high | | 30 | ns |
| Tvi | PWR_EN to $V_{DDI}$ stable | | 20 | ms |

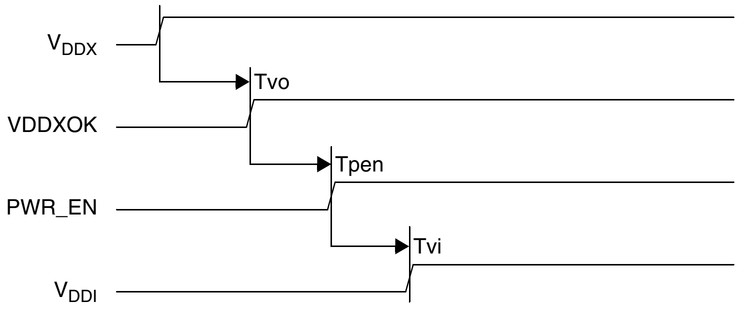


**Figure 12-12. Power-up Sequence**

### 12.7.2 Hardware Reset Timing

Table 12-17. Hardware Reset Timing Parameters

| Parameter | Description | Min | Typ | Max |
|---|---|---|---|---|
| Tvxr | VDDXOK asserted to RESETIN# deasserted | 0 ns | | System Dependent |
| Tvl | VDDXOK low time | 1 µs | | |
| Trstl | RESETIN# low time | 1 µs | | |
| Tvro | RESETIN# to RESETOUT# delay<br>MAX = max[750 ns, 170 ms - Tvxr] | 600 ns | | See Description |
| Trocs | RESETOUT# to RCS0#/SDCS0# asserted. | | 135 ns | 1 µs |

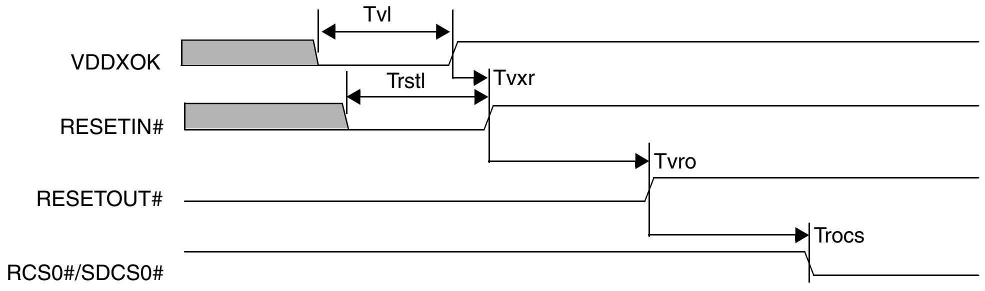
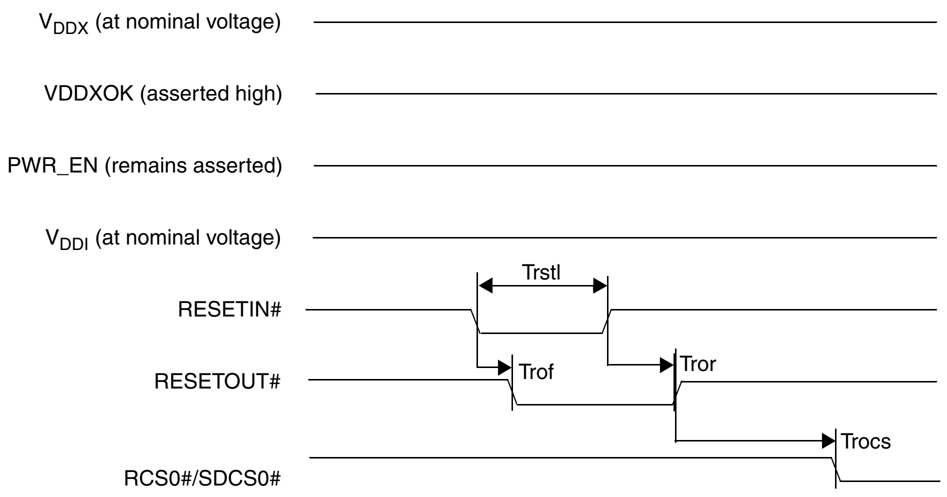


Figure 12-13. Hardware Reset Sequence

### 12.7.3 Runtime Reset Timing

**Table 12-18. Runtime Reset Timing Parameters**

| Parameter | Description | Min | Typ | Max |
|---|---|---|---|---|
| Trstl | RESETIN# low time | 1 µs | | |
| Trof | RESETIN# falling to RESETOUT# falling<br>MAX: 25 ns + (0.5 * (CPU Clock/2)) | | | See Description |
| Tror | RESETIN# rising to RESETOUT# rising<br>MAX: 25 ns + (0.5 * (CPU Clock/2)) + (120 * CPU Clock) | 120 CPU clocks | | See Description |
| Trocs | RESETOUT# to RCS0#/SDCS0# asserted. Note that the timing values shown assume a 400 MHz CPU clock. | | 65 ns | 500 ns |



$V_{DDX}$ (at nominal voltage)
VDDXOK (asserted high)
PWR_EN (remains asserted)
$V_{DDI}$ (at nominal voltage)
Trstl
RESETIN#
Trof
Tror
RESETOUT#
Trocs
RCS0#/SDCS0#

**Figure 12-14. Runtime Reset Sequence**

## 12.8 Asynchronous Signals

**GPIO**

The GPIO signals are driven by software. Note, however, when GPIO signals are used as inputs, there are timing requirements to ensure signal state changes are recognized cleanly; see Section 12.6.4 "GPIO Input Timing Requirements" on page 226.

**UART**

All UART signals are asynchronous to other external signals.

**USB**

All USB signals are asynchronous to other external signals. The USB protocol should be followed for appropriate operation.

## 12.9 External Clock Specifications

The EXTCLK[1:0] external clocks have a maximum frequency rating of ($F_{max}$ / 16), *where* $F_{max}$ is the maximum frequency rating for the part. Table 12-19 provides the EXTCLK[1:0] specifications.

**Table 12-19. External Clock EXTCLK[1:0] Specifications**

| | 333 MHz | | 400 MHz | | 500 MHz | | |
|---|---|---|---|---|---|---|---|
| **Specification** | **Min** | **Max** | **Min** | **Max** | **Min** | **Max** | **Unit** |
| Frequency | | 20.81 | | 25 | | 31.25 | MHz |
| Frequency jitter | | 4 | | 4 | | 4 | % |
| Duty cycle | 40 | 60 | 40 | 60 | 40 | 60 | % |

## 12.10 Crystal Specifications

Note that load capacitors for the external oscillators are integrated into the Au1500 processor so no external circuitry is required when using the specified crystal. For design layout considerations concerning the crystals, see Section 12.11.1 "Crystal Layout" on page 235.

Table 12-20 provides the specification for the parallel resonant 12 MHz crystal to be placed between XTI12 and XTO12. Table 12-21 provides the specification for the parallel resonant 32 KHz crystal to be placed between XTI32 and XTO32.

**Table 12-20. 12 MHz Crystal Specification**

| Specification | Min | Typ | Max | Unit |
|---|---|---|---|---|
| Resonant Frequency | 11 | 12 | 15 | MHz |
| Frequency Stability | | | ±100 | ppm |
| Motional Resistance | | | 60 | Ohms |
| Shunt Capacitance | | <5 | 7 | pF |
| Load Capacitance (Note 1) | 8 | 12 | 20 | pF |
| Drive Level | | | 100 | µW |
| Crystal Type | AT Cut | | | |

Note 1. This capacitance is integrated on the Au1500.

**Table 12-21. 32.768 KHz Crystal Specification**

| Specification | Min | Typ | Max | Unit |
|---|---|---|---|---|
| Resonant Frequency | | 32.768 | | KHz |
| Equivalent Series Resistance | | | 50k | Ohms |
| Shunt Capacitance | | 1.5 | 2.0 | pF |
| Load Capacitance (Note 1) | 6 | | 12 | pF |
| Motional Capacitance | | 3 | 4 | fF |
| Drive Level | | | 1 | µW |
| Quality Factor | 40k | | | |
| Crystal Type | Tuning Fork | | | |

Note 1. This capacitance is integrated on the Au1500.

## 12.11 System Design Considerations

This section provides information for system-level design issues.

### 12.11.1 Crystal Layout

The crystal layouts are critical. Without using vias, place traces directly over a ground plane on the top layer with keep-outs on all surrounding sides. Trace lengths should be less than 0.5 inches, and trace widths should be set to the minimum signal trace width for the design. Be sure not to allow other signals to come within 0.025 inches of these sensitive analog signals.

### 12.11.2 Decoupling Recommendations

This section provides recommendations for minimizing noise in a system. Note that specific decoupling requirements are system dependent.

To filter noise on the power supplies, $V_{DDX}$ and $V_{DDI}$, as well as XPWR12 and XPWR32, should be bypassed to ground using 10 µF capacitors: For each of the four sides of the package, place a capacitor within 0.5 inches.

To filter high-frequency noise, capacitors in the 10 nF range should be placed under the package:

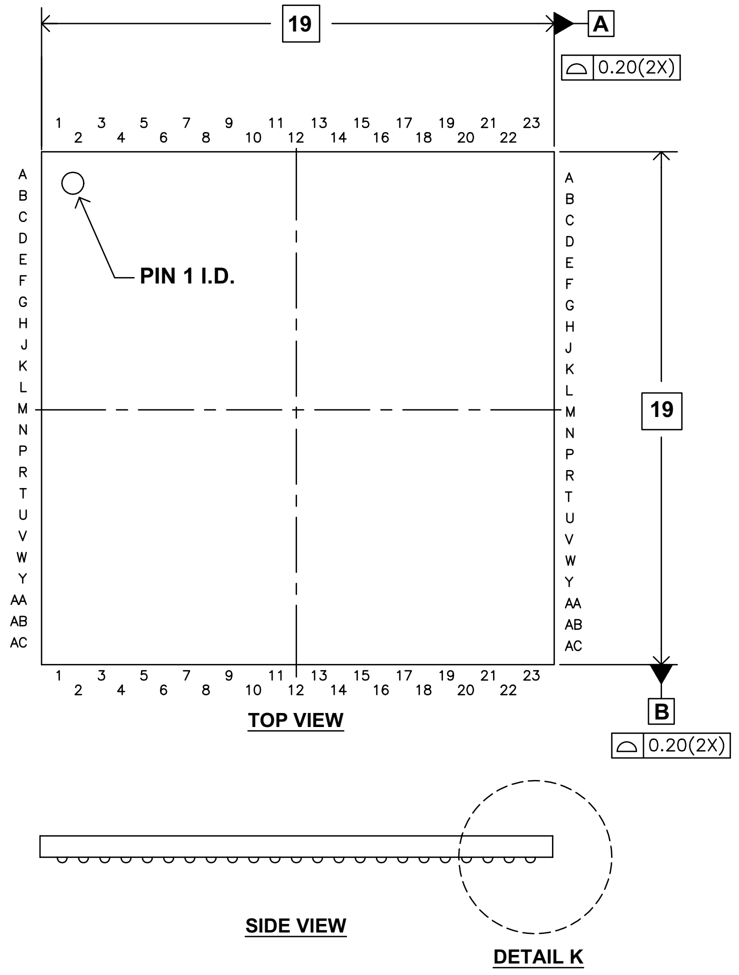
- For minimal high-frequency decoupling, use six to eight 10 nF capacitors.
- For systems requiring a broader spectrum of high-frequency noise be filtered, use four 15 nF and four 6.8 nF capacitors.

# 13 Packaging, Pin Assignments, and Ordering Information

This chapter provides information about the Au1500 processor package and pin assignment, as well as providing ordering information. The contents of the chapter are organized as follows:

- The package dimensions are shown in Figure 13-1 starting on page 238. The Au1500 is packaged in a 424-pin LF-PBGA device.
- Figure 13-2 (starting on page 240) is the connection diagram showing the pin and signal placement on the package. For pins that provide multiple signal functions, the default signal is shown first followed by the alternate signal in parentheses. Note that the black square in the upper-left hand corner indicates where the device is keyed.
- The pin assignment listing ordered by pin number starts on page 242.
- The pin assignment listing sorted by default signal starts on page 248.
- The pin assignment listing sorted by alternate signal starts on page 253.
- Ordering information is supplied on page 254.

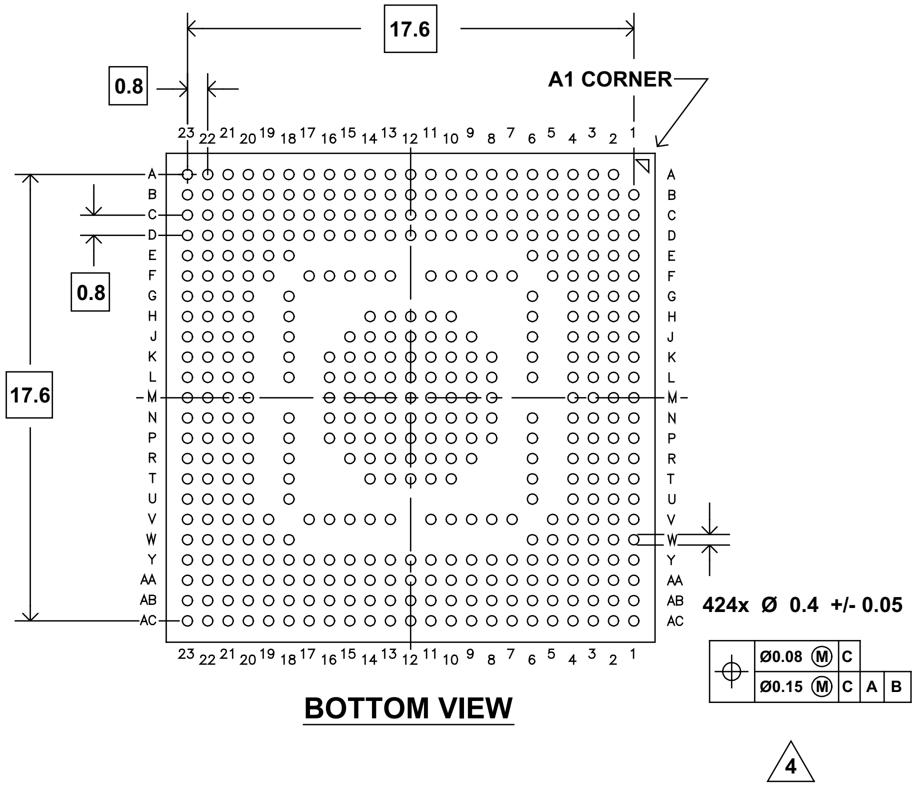
## 13.1 Packaging



**NOTES**

1. DIMENSIONING AND TOLERANCING PER ASME Y14.5M-1994 .
2. ALL DIMENSIONS ARE IN MILLIMETERS .
3. BALL POSITION DESIGNATION PER JESD 95-1, SPP-010.
4. BALL DIAMETER IS MEASURED AT ITS MAXIMUM DIMENSION IN A PLANE PARALLEL TO DATUM C.
5. THIS PACKAGE IS DIMENSIONED IN THE MANNER OF JEDEC OUTLINE MO-205 REV E, VARIATION Ax.

**Figure 13-1. Package Dimensions**



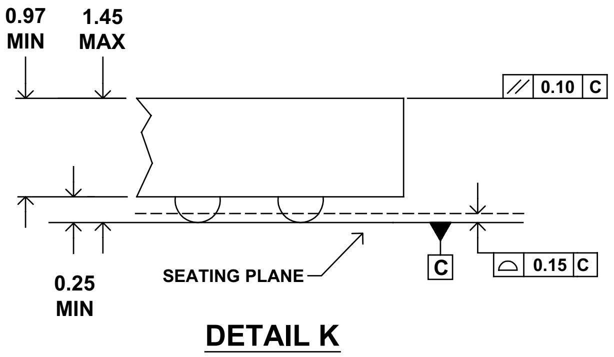


**Figure 13-1. Package Dimensions (Continued)**

## 13.2 Pin Assignment

**Figure 13-2. Connection Diagram—Top View**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **A** | | XTO32 | XTI32 | XTO12 | XTI12 | RESETIN# | GPIO[2] (EXTCLK0) | GPIO[4] (DMA_REQ0) | PIOR# | PREG# | PCE2# | EWAIT# |
| **B** | TESTEN | XPWR32 | XAGND32 | XPWR12 | XAGND12 | $V_{DDX}$ | $V_{SS}$ | GPIO[5] (DMA_REQ1) | POE# | PCE1# | LWAIT# | RD1 |
| **C** | SDA8 | $V_{SS}$ | TMS | VDDXOK | ROMSEL | TC0 | TC1 | TC2 | PIOW# | PWE# | LRD0# | LWR1# |
| **D** | SDA7 | $V_{DDX}$ | RESETOUT# | TRST# | ROMSIZE | GPIO[1] | GPIO[3] (EXTCLK1) | TC3 | GPIO[6] (SMROMCKE) | GPIO[7] | LRD1# | LCLK |
| **E** | SDA3 | SDA4 | SDA11 | SDA12 | PWR_EN | GPIO[0] | | | | | | |
| **F** | SDA2 | $V_{SS}$ | SDA9 | SDA10 | SDA6 | | $V_{DDX}$ | $V_{DDX}$ | PIOS16# | PWAIT# | LWR0# | |
| **G** | SDBA1 | $V_{DDX}$ | SDA5 | SDA0 | | $V_{DDX}$ | | | | | | |
| **H** | SDQM3# | SDBA0 | SDA1 | SDQM2# | | $V_{DDX}$ | | | | $V_{DDX}$ | $V_{DDI}$ | $V_{DDI}$ |
| **J** | SDCKE | SDCAS# | SDQM0# | SDQM1# | | SDCS2# | | | $V_{DDX}$ | $V_{SS}$ | $V_{SS}$ | $V_{SS}$ |
| **K** | SDWE# | SDRAS# | SDCLK2 | SDCLK1 | | SDD31 | | $V_{DDX}$ | $V_{SS}$ | $V_{SS}$ | $V_{SS}$ | $V_{SS}$ |
| **L** | SDCS0# | SDCS1# | SDD30 | SDD27 | | SDD25 | | $V_{DDI}$ | $V_{SS}$ | $V_{SS}$ | $V_{SS}$ | $V_{SS}$ |
| **M** | SDD28 | SDD29 | SDD26 | SDCLK0 | | | | $V_{DDI}$ | $V_{SS}$ | $V_{SS}$ | $V_{SS}$ | $V_{SS}$ |
| **N** | SDD24 | SDD23 | SDD22 | SDD21 | | SDD17 | | $V_{DDI}$ | $V_{SS}$ | $V_{SS}$ | $V_{SS}$ | $V_{SS}$ |
| **P** | SDD20 | SDD19 | SDD16 | SDD15 | | SDD12 | | $V_{DDX}$ | $V_{SS}$ | $V_{SS}$ | $V_{SS}$ | $V_{SS}$ |
| **R** | SDD18 | SDD14 | SDD11 | SDD8 | | $V_{DDX}$ | | | $V_{DDX}$ | $V_{SS}$ | $V_{SS}$ | $V_{SS}$ |
| **T** | SDD13 | $V_{SS}$ | SDD9 | SDD7 | | $V_{DDX}$ | | | | $V_{DDX}$ | $V_{DDI}$ | $V_{DDI}$ |
| **U** | SDD10 | $V_{DDX}$ | SDD5 | GPIO[15] | | $V_{DDX}$ | | | | | | |
| **V** | SDD6 | SDD4 | GPIO[14] (U3DTR#) | SDD0 | SDD1 | | $V_{DDX}$ | $V_{DDX}$ | GPIO[27] (N1TXD2) | N1CRS | N0COL | |
| **W** | SDD3 | SDD2 | ACRST# | GPIO[214] | GPIO[211] | GPIO[208] (DMA_REQ2) | | | | | | |
| **Y** | GPIO[13] (U3RTS#) | ACDO | GPIO[11] (U3DCD#) | GPIO[215] | GPIO[12] (U3RI#) | GPIO[212] | GPIO[210] | GPIO[28] (N1TXD3) | GPIO[25] (N1TXD0) | N0MDIO | N0TXD3 | N0RXDV |
| **AA** | ACSYNC | ACBCLK | GPIO[8] | GPIO[209] (DMA_REQ3) | TDI | GPIO[207] | GPIO[206] | TCK | GPIO[26] (N1TXD1) | N1RXD1 | N0TXD1 | N0TXEN |
| **AB** | ACDI | $V_{SS}$ | $V_{DDX}$ | TDO | N1COL | $V_{SS}$ | $V_{DDX}$ | N1RXD3 | GPIO[24] (N1TXEN) | N1RXD0 | N1TXCLK | GPIO[205] |
| **AC** | GPIO[10] (U3DSR#) | GPIO[9] (U3CTS#) | GPIO[213] | N1MDIO | N1MDC | N1RXDV | N1RXCLK | N1RXD2 | N0MDC | N0TXD2 | N0TXD0 | GPIO[204] |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

## Figure 13-2. Connection Diagram—Top View (Continued)

| 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RD2 | RD6 | RD8 | RD17 | RD18 | RD19 | RD23 | RD24 | RAD1 | RAD5 | RAD8 | **A** |
| RD3 | RD7 | RD12 | RD14 | $V_{DDX}$ | $V_{SS}$ | RD25 | RD30 | $V_{DDX}$ | $V_{SS}$ | RAD4 | **B** |
| RD4 | RD11 | RD15 | RD21 | RD26 | RD27 | RD28 | RAD0 | RAD2 | RAD3 | RAD13 | **C** |
| RD5 | RD10 | RD16 | RD20 | RD22 | RD31 | RAD7 | RAD6 | RAD15 | RAD17 | RAD18 | **D** |
| | | | | | RD29 | RAD10 | RAD9 | RAD11 | RAD21 | RAD22 | **E** |
| RD0 | RD9 | RD13 | $V_{DDX}$ | $V_{DDX}$ | | RAD12 | RAD14 | RAD16 | $V_{DDX}$ | RAD23 | **F** |
| | | | | | $V_{DDX}$ | | RAD19 | RAD20 | $V_{SS}$ | RAD26 | **G** |
| $V_{DDI}$ | $V_{DDX}$ | | | | $V_{DDX}$ | | RAD25 | RAD24 | RAD28 | RAD27 | **H** |
| $V_{SS}$ | $V_{SS}$ | $V_{DDX}$ | | | RBE2# | | RAD30 | RAD29 | RBE0# | RAD31 | **J** |
| $V_{SS}$ | $V_{SS}$ | $V_{SS}$ | $V_{DDI}$ | | RCS2# | | RWE# | RBE3# | RBE1# | ROE# | **K** |
| $V_{SS}$ | $V_{SS}$ | $V_{SS}$ | $V_{DDI}$ | | PCI_STOP# | | U0RXD | RCS1# | RCS3# | RCS0# | **L** |
| $V_{SS}$ | $V_{SS}$ | $V_{SS}$ | $V_{DDI}$ | | | | PCI_FRAME# | PCI_IRDY# | GPIO[23] (U3TXD) | GPIO[20] (U0TXD) | **M** |
| $V_{SS}$ | $V_{SS}$ | $V_{SS}$ | $V_{DDI}$ | | PCI_REQ0# | | PCI_CLKO | PCI_CLK | PCI_DEVSEL# | U3RXD | **N** |
| $V_{SS}$ | $V_{SS}$ | $V_{DDX}$ | $V_{DDX}$ | | PCI_PAR | | PCI_GNT1# | PCI_GNT0# | PCI_LOCK# | PCI_TRDY# | **P** |
| $V_{SS}$ | $V_{SS}$ | $V_{DDX}$ | | | PCI_REQ3# | | PCI_SERR# | PCI_CFG | PCI_REQ2# | PCI_REQ1# | **R** |
| $V_{DDI}$ | $V_{DDI}$ | | | | $V_{DDX}$ | | PCI_AD4 | PCI_AD3 | PCI_GNT3# | PCI_GNT2# | **T** |
| | | | | | $V_{DDX}$ | | PCI_AD10 | PCI_AD8 | $V_{DDX}$ | PCI_PERR# | **U** |
| N0RXD3 | N0RXD2 | PCI_INTB# | $V_{DDX}$ | $V_{DDX}$ | | PCI_AD11 | PCI_AD15 | PCI_AD6 | $V_{SS}$ | PCI_AD0 | **V** |
| | | | | | PCI_AD16 | PCI_AD17 | USBH1M | PCI_AD5 | PCI_AD2 | PCI_AD1 | **W** |
| N0RXCLK | GPIO[202] | PCI_INTC# | PCI_INTA# | PCI_AD25 | PCI_AD21 | USBDP (USBH0P) | USBDM (USBH0M) | PCI_AD13 | PCI_AD9 | PCI_AD7 | **Y** |
| GPIO[203] | GPIO[201] | PCI_CBE3# | PCI_CBE1# | PCI_AD31 | PCI_AD26 | PCI_AD22 | PCI_AD20 | USBH1P | $V_{DDX}$ | PCI_AD12 | **AA** |
| N0RXD1 | N0CRS | PCI_RST# | PCI_INTD# | $V_{SS}$ | $V_{DDX}$ | PCI_AD29 | PCI_AD27 | PCI_AD23 | $V_{SS}$ | PCI_AD14 | **AB** |
| N0RXD0 | N0TXCLK | GPIO[200] (PCI_RSTO#) | PCI_CBE2# | PCI_CBE0# | PCI_IDSEL | PCI_AD30 | PCI_AD28 | PCI_AD24 | PCI_AD19 | PCI_AD18 | **AC** |
| **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | |

**Table 13-1. Pin Assignment — Sorted by Pin Number**

| Pin No. | Default Signal | Alternate Signal |
|---|---|---|
| A2 | XTO32 | |
| A3 | XTI32 | |
| A4 | XTO12 | |
| A5 | XTI12 | |
| A6 | RESETIN# | |
| A7 | GPIO[2] | EXTCLK0 |
| A8 | GPIO[4] | DMA_REQ0 |
| A9 | PIOR# | |
| A10 | PREG# | |
| A11 | PCE2# | |
| A12 | EWAIT# | |
| A13 | RD2 | |
| A14 | RD6 | |
| A15 | RD8 | |
| A16 | RD17 | |
| A17 | RD18 | |
| A18 | RD19 | |
| A19 | RD23 | |
| A20 | RD24 | |
| A21 | RAD1 | |
| A22 | RAD5 | |
| A23 | RAD8 | |
| B1 | TESTEN | |
| B2 | XPWR32 | |
| B3 | XAGND32 | |
| B4 | XPWR12 | |
| B5 | XAGND12 | |
| B6 | $V_{DDX}$ | |
| B7 | $V_{SS}$ | |
| B8 | GPIO[5] | DMA_REQ1 |
| B9 | POE# | |
| B10 | PCE1# | |
| B11 | LWAIT# | |
| B12 | RD1 | |
| B13 | RD3 | |
| B14 | RD7 | |
| B15 | RD12 | |
| B16 | RD14 | |
| B17 | $V_{DDX}$ | |
| B18 | $V_{SS}$ | |
| B19 | RD25 | |
| B20 | RD30 | |
| B21 | $V_{DDX}$ | |
| B22 | $V_{SS}$ | |
| B23 | RAD4 | |
| C1 | SDA8 | |
| C2 | $V_{SS}$ | |
| C3 | TMS | |
| C4 | VDDXOK | |
| C5 | ROMSEL | |
| C6 | TC0 | |
| C7 | TC1 | |
| C8 | TC2 | |
| C9 | PIOW# | |
| C10 | PWE# | |
| C11 | LRD0# | |
| C12 | LWR1# | |
| C13 | RD4 | |
| C14 | RD11 | |
| C15 | RD15 | |
| C16 | RD21 | |
| C17 | RD26 | |
| C18 | RD27 | |
| C19 | RD28 | |
| C20 | RAD0 | |
| C21 | RAD2 | |
| C22 | RAD3 | |
| C23 | RAD13 | |
| D1 | SDA7 | |
| D2 | $V_{DDX}$ | |
| D3 | RESETOUT# | |
| D4 | TRST# | |
| D5 | ROMSIZE | |
| D6 | GPIO[1] | |
| D7 | GPIO[3] | EXTCLK1 |
| D8 | TC3 | |
| D9 | GPIO[6] | SMROMCKE |
| D10 | GPIO[7] | |

**Table 13-1. Pin Assignment — Sorted by Pin Number (Continued)**

| Pin No. | Default Signal | Alternate Signal |
|---|---|---|
| D11 | LRD1# | |
| D12 | LCLK | |
| D13 | RD5 | |
| D14 | RD10 | |
| D15 | RD16 | |
| D16 | RD20 | |
| D17 | RD22 | |
| D18 | RD31 | |
| D19 | RAD7 | |
| D20 | RAD6 | |
| D21 | RAD15 | |
| D22 | RAD17 | |
| D23 | RAD18 | |
| E1 | SDA3 | |
| E2 | SDA4 | |
| E3 | SDA11 | |
| E4 | SDA12 | |
| E5 | PWR_EN | |
| E6 | GPIO[0] | |
| E18 | RD29 | |
| E19 | RAD10 | |
| E20 | RAD9 | |
| E21 | RAD11 | |
| E22 | RAD21 | |
| E23 | RAD22 | |
| F1 | SDA2 | |
| F2 | $V_{SS}$ | |
| F3 | SDA9 | |
| F4 | SDA10 | |
| F5 | SDA6 | |
| F7 | $V_{DDX}$ | |
| F8 | $V_{DDX}$ | |
| F9 | PIOS16# | |
| F10 | PWAIT# | |
| F11 | LWR0# | |
| F13 | RD0 | |
| F14 | RD9 | |
| F15 | RD13 | |
| F16 | $V_{DDX}$ | |
| F17 | $V_{DDX}$ | |
| F19 | RAD12 | |
| F20 | RAD14 | |
| F21 | RAD16 | |
| F22 | $V_{DDX}$ | |
| F23 | RAD23 | |
| G1 | SDBA1 | |
| G2 | $V_{DDX}$ | |
| G3 | SDA5 | |
| G4 | SDA0 | |
| G6 | $V_{DDX}$ | |
| G18 | $V_{DDX}$ | |
| G20 | RAD19 | |
| G21 | RAD20 | |
| G22 | $V_{SS}$ | |
| G23 | RAD26 | |
| H1 | SDQM3# | |
| H2 | SDBA0 | |
| H3 | SDA1 | |
| H4 | SDQM2# | |
| H6 | $V_{DDX}$ | |
| H10 | $V_{DDX}$ | |
| H11 | $V_{DDI}$ | |
| H12 | $V_{DDI}$ | |
| H13 | $V_{DDI}$ | |
| H14 | $V_{DDX}$ | |
| H18 | $V_{DDX}$ | |
| H20 | RAD25 | |
| H21 | RAD24 | |
| H22 | RAD28 | |
| H23 | RAD27 | |
| J1 | SDCKE | |
| J2 | SDCAS# | |
| J3 | SDQM0# | |
| J4 | SDQM1# | |
| J6 | SDCS2# | |
| J9 | $V_{DDX}$ | |

**Table 13-1. Pin Assignment — Sorted by Pin Number (Continued)**

| Pin No. | Default Signal | Alternate Signal |
|---|---|---|
| J10 | $V_{SS}$ | |
| J11 | $V_{SS}$ | |
| J12 | $V_{SS}$ | |
| J13 | $V_{SS}$ | |
| J14 | $V_{SS}$ | |
| J15 | $V_{DDX}$ | |
| J18 | RBE2# | |
| J20 | RAD30 | |
| J21 | RAD29 | |
| J22 | RBE0# | |
| J23 | RAD31 | |
| K1 | SDWE# | |
| K2 | SDRAS# | |
| K3 | SDCLK2 | |
| K4 | SDCLK1 | |
| K6 | SDD31 | |
| K8 | $V_{DDX}$ | |
| K9 | $V_{SS}$ | |
| K10 | $V_{SS}$ | |
| K11 | $V_{SS}$ | |
| K12 | $V_{SS}$ | |
| K13 | $V_{SS}$ | |
| K14 | $V_{SS}$ | |
| K15 | $V_{SS}$ | |
| K16 | $V_{DDI}$ | |
| K18 | RCS2# | |
| K20 | RWE# | |
| K21 | RBE3# | |
| K22 | RBE1# | |
| K23 | ROE# | |
| L1 | SDCS0# | |
| L2 | SDCS1# | |
| L3 | SDD30 | |
| L4 | SDD27 | |
| L6 | SDD25 | |
| L8 | $V_{DDI}$ | |
| L9 | $V_{SS}$ | |
| L10 | $V_{SS}$ | |
| L11 | $V_{SS}$ | |
| L12 | $V_{SS}$ | |
| L13 | $V_{SS}$ | |
| L14 | $V_{SS}$ | |
| L15 | $V_{SS}$ | |
| L16 | $V_{DDI}$ | |
| L18 | PCI_STOP# | |
| L20 | U0RXD | |
| L21 | RCS1# | |
| L22 | RCS3# | |
| L23 | RCS0# | |
| M1 | SDD28 | |
| M2 | SDD29 | |
| M3 | SDD26 | |
| M4 | SDCLK0 | |
| M8 | $V_{DDI}$ | |
| M9 | $V_{SS}$ | |
| M10 | $V_{SS}$ | |
| M11 | $V_{SS}$ | |
| M12 | $V_{SS}$ | |
| M13 | $V_{SS}$ | |
| M14 | $V_{SS}$ | |
| M15 | $V_{SS}$ | |
| M16 | $V_{DDI}$ | |
| M20 | PCI_FRAME# | |
| M21 | PCI_IRDY# | |
| M22 | GPIO[23] | U3TXD |
| M23 | GPIO[20] | U0TXD |
| N1 | SDD24 | |
| N2 | SDD23 | |
| N3 | SDD22 | |
| N4 | SDD21 | |
| N6 | SDD17 | |
| N8 | $V_{DDI}$ | |
| N9 | $V_{SS}$ | |
| N10 | $V_{SS}$ | |

**Table 13-1. Pin Assignment — Sorted by Pin Number (Continued)**

| Pin No. | Default Signal | Alternate Signal |
|---|---|---|
| N11 | $V_{SS}$ | |
| N12 | $V_{SS}$ | |
| N13 | $V_{SS}$ | |
| N14 | $V_{SS}$ | |
| N15 | $V_{SS}$ | |
| N16 | $V_{DDI}$ | |
| N18 | PCI_REQ0# | |
| N20 | PCI_CLKO | |
| N21 | PCI_CLK | |
| N22 | PCI_DEVSEL# | |
| N23 | U3RXD | |
| P1 | SDD20 | |
| P2 | SDD19 | |
| P3 | SDD16 | |
| P4 | SDD15 | |
| P6 | SDD12 | |
| P8 | $V_{DDX}$ | |
| P9 | $V_{SS}$ | |
| P10 | $V_{SS}$ | |
| P11 | $V_{SS}$ | |
| P12 | $V_{SS}$ | |
| P13 | $V_{SS}$ | |
| P14 | $V_{SS}$ | |
| P15 | $V_{DDX}$ | |
| P16 | $V_{DDX}$ | |
| P18 | PCI_PAR | |
| P20 | PCI_GNT1# | |
| P21 | PCI_GNT0# | |
| P22 | PCI_LOCK# | |
| P23 | PCI_TRDY# | |
| R1 | SDD18 | |
| R2 | SDD14 | |
| R3 | SDD11 | |
| R4 | SDD8 | |
| R6 | $V_{DDX}$ | |
| R9 | $V_{DDX}$ | |
| R10 | $V_{SS}$ | |
| R11 | $V_{SS}$ | |
| R12 | $V_{SS}$ | |
| R13 | $V_{SS}$ | |
| R14 | $V_{SS}$ | |
| R15 | $V_{DDX}$ | |
| R18 | PCI_REQ3# | |
| R20 | PCI_SERR# | |
| R21 | PCI_CFG | |
| R22 | PCI_REQ2# | |
| R23 | PCI_REQ1# | |
| T1 | SDD13 | |
| T2 | $V_{SS}$ | |
| T3 | SDD9 | |
| T4 | SDD7 | |
| T6 | $V_{DDX}$ | |
| T10 | $V_{DDX}$ | |
| T11 | $V_{DDI}$ | |
| T12 | $V_{DDI}$ | |
| T13 | $V_{DDI}$ | |
| T14 | $V_{DDI}$ | |
| T18 | $V_{DDX}$ | |
| T20 | PCI_AD4 | |
| T21 | PCI_AD3 | |
| T22 | PCI_GNT3# | |
| T23 | PCI_GNT2# | |
| U1 | SDD10 | |
| U2 | $V_{DDX}$ | |
| U3 | SDD5 | |
| U4 | GPIO[15] | |
| U6 | $V_{DDX}$ | |
| U18 | $V_{DDX}$ | |
| U20 | PCI_AD10 | |
| U21 | PCI_AD8 | |
| U22 | $V_{DDX}$ | |
| U23 | PCI_PERR# | |
| V1 | SDD6 | |
| V2 | SDD4 | |

**Table 13-1. Pin Assignment — Sorted by Pin Number (Continued)**

| Pin No. | Default Signal | Alternate Signal |
|---|---|---|
| V3 | GPIO[14] | U3DTR# |
| V4 | SDD0 | |
| V5 | SDD1 | |
| V7 | $V_{DDX}$ | |
| V8 | $V_{DDX}$ | |
| V9 | GPIO[27] | N1TXD2 |
| V10 | N1CRS | |
| V11 | N0COL | |
| V13 | N0RXD3 | |
| V14 | N0RXD2 | |
| V15 | PCI_INTB# | |
| V16 | $V_{DDX}$ | |
| V17 | $V_{DDX}$ | |
| V19 | PCI_AD11 | |
| V20 | PCI_AD15 | |
| V21 | PCI_AD6 | |
| V22 | $V_{SS}$ | |
| V23 | PCI_AD0 | |
| W1 | SDD3 | |
| W2 | SDD2 | |
| W3 | ACRST# | |
| W4 | GPIO[214] | |
| W5 | GPIO[211] | |
| W6 | GPIO[208] | DMA_REQ2 |
| W18 | PCI_AD16 | |
| W19 | PCI_AD17 | |
| W20 | USBH1M | |
| W21 | PCI_AD5 | |
| W22 | PCI_AD2 | |
| W23 | PCI_AD1 | |
| Y1 | GPIO[13] | U3RTS# |
| Y2 | ACDO | |
| Y3 | GPIO[11] | U3DCD# |
| Y4 | GPIO[215] | |
| Y5 | GPIO[12] | U3RI# |
| Y6 | GPIO[212] | |
| Y7 | GPIO[210] | |
| Y8 | GPIO[28] | N1TXD3 |
| Y9 | GPIO[25] | N1TXD0 |
| Y10 | N0MDIO | |
| Y11 | N0TXD3 | |
| Y12 | N0RXDV | |
| Y13 | N0RXCLK | |
| Y14 | GPIO[202] | |
| Y15 | PCI_INTC# | |
| Y16 | PCI_INTA# | |
| Y17 | PCI_AD25 | |
| Y18 | PCI_AD21 | |
| Y19 | USBDP | USBH0P |
| Y20 | USBDM | USBH0M |
| Y21 | PCI_AD13 | |
| Y22 | PCI_AD9 | |
| Y23 | PCI_AD7 | |
| AA1 | ACSYNC | |
| AA2 | ACBCLK | |
| AA3 | GPIO[8] | |
| AA4 | GPIO[209] | DMA_REQ3 |
| AA5 | TDI | |
| AA6 | GPIO[207] | |
| AA7 | GPIO[206] | |
| AA8 | TCK | |
| AA9 | GPIO[26] | N1TXD1 |
| AA10 | N1RXD1 | |
| AA11 | N0TXD1 | |
| AA12 | N0TXEN | |
| AA13 | GPIO[203] | |
| AA14 | GPIO[201] | |
| AA15 | PCI_CBE3# | |
| AA16 | PCI_CBE1# | |
| AA17 | PCI_AD31 | |
| AA18 | PCI_AD26 | |
| AA19 | PCI_AD22 | |
| AA20 | PCI_AD20 | |
| AA21 | USBH1P | |
| AA22 | $V_{DDX}$ | |
| AA23 | PCI_AD12 | |
| AB1 | ACDI | |
| AB2 | $V_{SS}$ | |

**Table 13-1. Pin Assignment — Sorted by Pin Number (Continued)**

| Pin No. | Default Signal | Alternate Signal |
|---|---|---|
| AB3 | $V_{DDX}$ | |
| AB4 | TDO | |
| AB5 | N1COL | |
| AB6 | $V_{SS}$ | |
| AB7 | $V_{DDX}$ | |
| AB8 | N1RXD3 | |
| AB9 | GPIO[24] | N1TXEN |
| AB10 | N1RXD0 | |
| AB11 | N1TXCLK | |
| AB12 | GPIO[205] | |
| AB13 | N0RXD1 | |
| AB14 | N0CRS | |
| AB15 | PCI_RST# | |
| AB16 | PCI_INTD# | |
| AB17 | $V_{SS}$ | |
| AB18 | $V_{DDX}$ | |
| AB19 | PCI_AD29 | |
| AB20 | PCI_AD27 | |
| AB21 | PCI_AD23 | |
| AB22 | $V_{SS}$ | |
| AB23 | PCI_AD14 | |
| AC1 | GPIO[10] | U3DSR# |
| AC2 | GPIO[9] | U3CTS# |
| AC3 | GPIO[213] | |
| AC4 | N1MDIO | |
| AC5 | N1MDC | |
| AC6 | N1RXDV | |
| AC7 | N1RXCLK | |
| AC8 | N1RXD2 | |
| AC9 | N0MDC | |
| AC10 | N0TXD2 | |
| AC11 | N0TXD0 | |
| AC12 | GPIO[204] | |
| AC13 | N0RXD0 | |
| AC14 | N0TXCLK | |
| AC15 | GPIO[200] | PCI_RSTO# |
| AC16 | PCI_CBE2# | |
| AC17 | PCI_CBE0# | |
| AC18 | PCI_IDSEL | |
| AC19 | PCI_AD30 | |
| AC20 | PCI_AD28 | |
| AC21 | PCI_AD24 | |
| AC22 | PCI_AD19 | |
| AC23 | PCI_AD18 | |

**Table 13-2. Pin Assignment — Sorted Alphabetically by Default Signal**

| Default Signal | Alternate Signal | Pin No. |
|---|---|---|
| ACBCLK | | AA2 |
| ACDI | | AB1 |
| ACDO | | Y2 |
| ACRST# | | W3 |
| ACSYNC | | AA1 |
| EWAIT# | | A12 |
| GPIO[0] | | E6 |
| GPIO[1] | | D6 |
| GPIO[2] | EXTCLK0 | A7 |
| GPIO[3] | EXTCLK1 | D7 |
| GPIO[4] | DMA_REQ0 | A8 |
| GPIO[5] | DMA_REQ1 | B8 |
| GPIO[6] | SMROMCKE | D9 |
| GPIO[7] | | D10 |
| GPIO[8] | | AA3 |
| GPIO[9] | U3CTS# | AC2 |
| GPIO[10] | U3DSR# | AC1 |
| GPIO[11] | U3DCD# | Y3 |
| GPIO[12] | U3RI# | Y5 |
| GPIO[13] | U3RTS# | Y1 |
| GPIO[14] | U3DTR# | V3 |
| GPIO[15] | | U4 |
| GPIO[20] | U0TXD | M23 |
| GPIO[23] | U3TXD | M22 |
| GPIO[24] | N1TXEN | AB9 |
| GPIO[25] | N1TXD0 | Y9 |
| GPIO[26] | N1TXD1 | AA9 |
| GPIO[27] | N1TXD2 | V9 |
| GPIO[28] | N1TXD3 | Y8 |
| GPIO[200] | PCI_RSTO# | AC15 |
| GPIO[201] | | AA14 |
| GPIO[202] | | Y14 |
| GPIO[203] | | AA13 |
| GPIO[204] | | AC12 |
| GPIO[205] | | AB12 |
| GPIO[206] | | AA7 |
| GPIO[207] | | AA6 |
| GPIO[208] | DMA_REQ2 | W6 |
| GPIO[209] | DMA_REQ3 | AA4 |
| GPIO[210] | | Y7 |
| GPIO[211] | | W5 |
| GPIO[212] | | Y6 |
| GPIO[213] | | AC3 |
| GPIO[214] | | W4 |
| GPIO[215] | | Y4 |
| LCLK | | D12 |
| LRD0# | | C11 |
| LRD1# | | D11 |
| LWAIT# | | B11 |
| LWR0# | | F11 |
| LWR1# | | C12 |
| N0COL | | V11 |
| N0CRS | | AB14 |
| N0MDC | | AC9 |
| N0MDIO | | Y10 |
| N0RXCLK | | Y13 |
| N0RXD0 | | AC13 |
| N0RXD1 | | AB13 |
| N0RXD2 | | V14 |
| N0RXD3 | | V13 |
| N0RXDV | | Y12 |
| N0TXCLK | | AC14 |
| N0TXD0 | | AC11 |
| N0TXD1 | | AA11 |
| N0TXD2 | | AC10 |
| N0TXD3 | | Y11 |
| N0TXEN | | AA12 |
| N1COL | | AB5 |
| N1CRS | | V10 |
| N1MDC | | AC5 |
| N1MDIO | | AC4 |
| N1RXCLK | | AC7 |
| N1RXD0 | | AB10 |
| N1RXD1 | | AA10 |
| N1RXD2 | | AC8 |
| N1RXD3 | | AB8 |
| N1RXDV | | AC6 |
| N1TXCLK | | AB11 |

**Table 13-2. Pin Assignment — Sorted Alphabetically by Default Signal (Continued)**

| Default Signal | Alternate Signal | Pin No. |
|---|---|---|
| PCE1# | | B10 |
| PCE2# | | A11 |
| PCI_AD0 | | V23 |
| PCI_AD1 | | W23 |
| PCI_AD2 | | W22 |
| PCI_AD3 | | T21 |
| PCI_AD4 | | T20 |
| PCI_AD5 | | W21 |
| PCI_AD6 | | V21 |
| PCI_AD7 | | Y23 |
| PCI_AD8 | | U21 |
| PCI_AD9 | | Y22 |
| PCI_AD10 | | U20 |
| PCI_AD11 | | V19 |
| PCI_AD12 | | AA23 |
| PCI_AD13 | | Y21 |
| PCI_AD14 | | AB23 |
| PCI_AD15 | | V20 |
| PCI_AD16 | | W18 |
| PCI_AD17 | | W19 |
| PCI_AD18 | | AC23 |
| PCI_AD19 | | AC22 |
| PCI_AD20 | | AA20 |
| PCI_AD21 | | Y18 |
| PCI_AD22 | | AA19 |
| PCI_AD23 | | AB21 |
| PCI_AD24 | | AC21 |
| PCI_AD25 | | Y17 |
| PCI_AD26 | | AA18 |
| PCI_AD27 | | AB20 |
| PCI_AD28 | | AC20 |
| PCI_AD29 | | AB19 |
| PCI_AD30 | | AC19 |
| PCI_AD31 | | AA17 |
| PCI_CBE0# | | AC17 |
| PCI_CBE1# | | AA16 |
| PCI_CBE2# | | AC16 |
| PCI_CBE3# | | AA15 |
| PCI_CFG | | R21 |
| PCI_CLK | | N21 |
| PCI_CLKO | | N20 |
| PCI_DEVSEL# | | N22 |
| PCI_FRAME# | | M20 |
| PCI_GNT0# | | P21 |
| PCI_GNT1# | | P20 |
| PCI_GNT2# | | T23 |
| PCI_GNT3# | | T22 |
| PCI_IDSEL | | AC18 |
| PCI_INTA# | | Y16 |
| PCI_INTB# | | V15 |
| PCI_INTC# | | Y15 |
| PCI_INTD# | | AB16 |
| PCI_IRDY# | | M21 |
| PCI_LOCK# | | P22 |
| PCI_PAR | | P18 |
| PCI_PERR# | | U23 |
| PCI_REQ0# | | N18 |
| PCI_REQ1# | | R23 |
| PCI_REQ2# | | R22 |
| PCI_REQ3# | | R18 |
| PCI_RST# | | AB15 |
| PCI_SERR# | | R20 |
| PCI_STOP# | | L18 |
| PCI_TRDY# | | P23 |
| PIOR# | | A9 |
| PIOS16# | | F9 |
| PIOW# | | C9 |
| POE# | | B9 |
| PREG# | | A10 |
| PWAIT# | | F10 |
| PWE# | | C10 |
| PWR_EN | | E5 |
| RAD0 | | C20 |
| RAD1 | | A21 |
| RAD2 | | C21 |
| RAD3 | | C22 |
| RAD4 | | B23 |
| RAD5 | | A22 |

**Table 13-2. Pin Assignment — Sorted Alphabetically by Default Signal (Continued)**

| Default Signal | Alternate Signal | Pin No. |
|---|---|---|
| RAD6 | | D20 |
| RAD7 | | D19 |
| RAD8 | | A23 |
| RAD9 | | E20 |
| RAD10 | | E19 |
| RAD11 | | E21 |
| RAD12 | | F19 |
| RAD13 | | C23 |
| RAD14 | | F20 |
| RAD15 | | D21 |
| RAD16 | | F21 |
| RAD17 | | D22 |
| RAD18 | | D23 |
| RAD19 | | G20 |
| RAD20 | | G21 |
| RAD21 | | E22 |
| RAD22 | | E23 |
| RAD23 | | F23 |
| RAD24 | | H21 |
| RAD25 | | H20 |
| RAD26 | | G23 |
| RAD27 | | H23 |
| RAD28 | | H22 |
| RAD29 | | J21 |
| RAD30 | | J20 |
| RAD31 | | J23 |
| RBE0# | | J22 |
| RBE1# | | K22 |
| RBE2# | | J18 |
| RBE3# | | K21 |
| RCS0# | | L23 |
| RCS1# | | L21 |
| RCS2# | | K18 |
| RCS3# | | L22 |
| RD0 | | F13 |
| RD1 | | B12 |
| RD2 | | A13 |
| RD3 | | B13 |
| RD4 | | C13 |
| RD5 | | D13 |
| RD6 | | A14 |
| RD7 | | B14 |
| RD8 | | A15 |
| RD9 | | F14 |
| RD10 | | D14 |
| RD11 | | C14 |
| RD12 | | B15 |
| RD13 | | F15 |
| RD14 | | B16 |
| RD15 | | C15 |
| RD16 | | D15 |
| RD17 | | A16 |
| RD18 | | A17 |
| RD19 | | A18 |
| RD20 | | D16 |
| RD21 | | C16 |
| RD22 | | D17 |
| RD23 | | A19 |
| RD24 | | A20 |
| RD25 | | B19 |
| RD26 | | C17 |
| RD27 | | C18 |
| RD28 | | C19 |
| RD29 | | E18 |
| RD30 | | B20 |
| RD31 | | D18 |
| RESETIN# | | A6 |
| RESETOUT# | | D3 |
| ROE# | | K23 |
| ROMSEL | | C5 |
| ROMSIZE | | D5 |
| RWE# | | K20 |
| SDA0 | | G4 |
| SDA1 | | H3 |
| SDA2 | | F1 |
| SDA3 | | E1 |
| SDA4 | | E2 |
| SDA5 | | G3 |

Table 13-2. Pin Assignment — Sorted Alphabetically by Default Signal (Continued)

| Default Signal | Alternate Signal | Pin No. |
|---|---|---|
| SDA6 | | F5 |
| SDA7 | | D1 |
| SDA8 | | C1 |
| SDA9 | | F3 |
| SDA10 | | F4 |
| SDA11 | | E3 |
| SDA12 | | E4 |
| SDBA0 | | H2 |
| SDBA1 | | G1 |
| SDCAS# | | J2 |
| SDCKE | | J1 |
| SDCLK0 | | M4 |
| SDCLK1 | | K4 |
| SDCLK2 | | K3 |
| SDCS0# | | L1 |
| SDCS1# | | L2 |
| SDCS2# | | J6 |
| SDD0 | | V4 |
| SDD1 | | V5 |
| SDD2 | | W2 |
| SDD3 | | W1 |
| SDD4 | | V2 |
| SDD5 | | U3 |
| SDD6 | | V1 |
| SDD7 | | T4 |
| SDD8 | | R4 |
| SDD9 | | T3 |
| SDD10 | | U1 |
| SDD11 | | R3 |
| SDD12 | | P6 |
| SDD13 | | T1 |
| SDD14 | | R2 |
| SDD15 | | P4 |
| SDD16 | | P3 |
| SDD17 | | N6 |
| SDD18 | | R1 |
| SDD19 | | P2 |
| SDD20 | | P1 |
| SDD21 | | N4 |
| SDD22 | | N3 |
| SDD23 | | N2 |
| SDD24 | | N1 |
| SDD25 | | L6 |
| SDD26 | | M3 |
| SDD27 | | L4 |
| SDD28 | | M1 |
| SDD29 | | M2 |
| SDD30 | | L3 |
| SDD31 | | K6 |
| SDQM0# | | J3 |
| SDQM1# | | J4 |
| SDQM2# | | H4 |
| SDQM3# | | H1 |
| SDRAS# | | K2 |
| SDWE# | | K1 |
| TC0 | | C6 |
| TC1 | | C7 |
| TC2 | | C8 |
| TC3 | | D8 |
| TCK | | AA8 |
| TDI | | AA5 |
| TDO | | AB4 |
| TESTEN | | B1 |
| TMS | | C3 |
| TRST# | | D4 |
| U0RXD | | L20 |
| U3RXD | | N23 |
| USBDM | USBH0M | Y20 |
| USBDP | USBH0P | Y19 |
| USBH1M | | W20 |
| USBH1P | | AA21 |
| $V_{DDI}$ | | H11, H12, H13, K16, L8, L16, M8, M16, N8, N16, T11, T12, T13, T14 |

**Table 13-2. Pin Assignment — Sorted Alphabetically by Default Signal (Continued)**

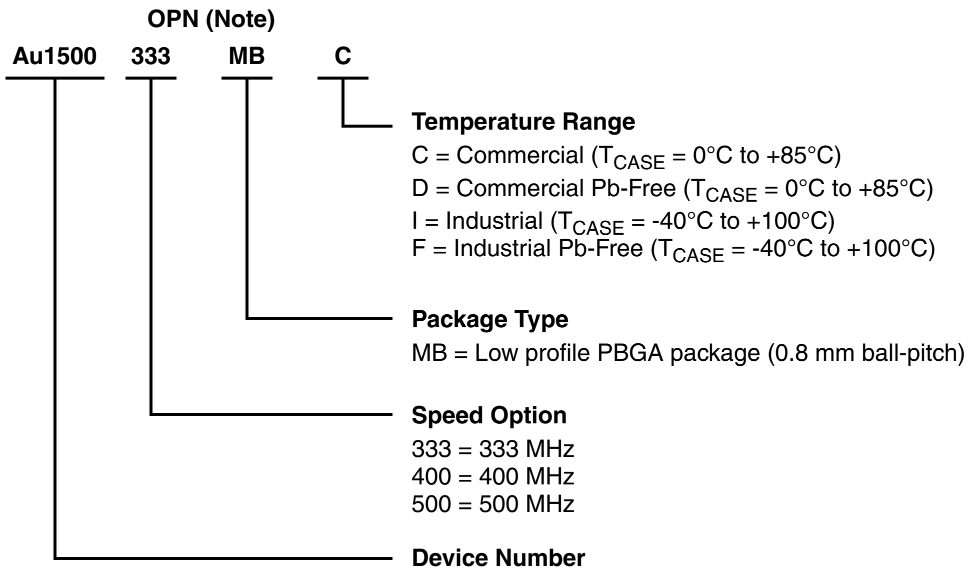
| Default Signal | Alternate Signal | Pin No. |
|---|---|---|
| $V_{DDX}$ | | B6, B17, B21, D2, F7, F8, F16, F17, F22, G2, G6, G18, H6, H10, H14, H18, J9, J15, K8, P8, P15, P16, R6, R9, R15, T6, T10, T18, U2, U6, U18, U22, V7, V8, V16, V17, AA22, AB3, AB7, AB18 |
| VDDXOK | | C4 |
| $V_{SS}$ | | B7, B18, B22, C2, F2, G22, J10, J11, J12, J13, J14, K9, K10, K11, K12, K13, K14, K15, L9, L10, L11, L12, L13, L14, L15, M9, M10, M11, M12, M13, M14, M15, N9, N10, N11, N12, N13, N14, N15, P9, P10, P11, P12, P13, P14, R10, R11, R12, R13, R14, T2, V22, AB2, AB6, AB17, AB22 |
| XAGND12 | | B5 |
| XAGND32 | | B3 |
| XPWR12 | | B4 |
| XPWR32 | | B2 |
| XTI12 | | A5 |
| XTI32 | | A3 |
| XTO12 | | A4 |
| XTO32 | | A2 |

**Table 13-3. Pin Assignment — Alternate Signals Sorted Alphabetically**

| Alternate Signal | Default Signal | Pin No. |
|---|---|---|
| EXTCLK0 | GPIO[2] | A7 |
| EXTCLK1 | GPIO[3] | D7 |
| DMA_REQ0 | GPIO[4] | A8 |
| DMA_REQ1 | GPIO[5] | B8 |
| DMA_REQ2 | GPIO[208] | AA3 |
| DMA_REQ3 | GPIO[209] | AA4 |
| N1TXD0 | GPIO[25] | Y9 |
| N1TXD1 | GPIO[26] | AA9 |
| N1TXD2 | GPIO[27] | V9 |
| N1TXD3 | GPIO[28] | Y8 |
| N1TXEN | GPIO[24] | AB9 |
| PCI_RSTO# | GPIO[200] | AC15 |
| SMROMCKE | GPIO[6] | D9 |
| U0TXD | GPIO[20] | M23 |
| U3CTS# | GPIO[9] | AC2 |
| U3DCD# | GPIO[11] | Y3 |
| U3DSR# | GPIO[10] | AC1 |
| U3DTR# | GPIO[14] | V3 |
| U3RI# | GPIO[12] | Y5 |
| U3RTS# | GPIO[13] | Y1 |
| U3TXD | GPIO[23] | M22 |
| USBH0M | USBDM | Y20 |
| USBH0P | USBDP | Y19 |

## 13.3 Ordering Information

Ordering information for the AMD Alchemy™ Au1500™ processor is contained in this section. The ordering part number (OPN) is formed by a combination of elements. An example of the OPN is shown in Figure 13-3. Valid OPN combinations are provided in Table 13-4.



**OPN (Note)**

**Au1500 333 MB C**

**Temperature Range**
C = Commercial ($T_{CASE}$ = 0°C to +85°C)
D = Commercial Pb-Free ($T_{CASE}$ = 0°C to +85°C)
I = Industrial ($T_{CASE}$ = -40°C to +100°C)
F = Industrial Pb-Free ($T_{CASE}$ = -40°C to +100°C)

**Package Type**
MB = Low profile PBGA package (0.8 mm ball-pitch)

**Speed Option**
333 = 333 MHz
400 = 400 MHz
500 = 500 MHz

**Device Number**

**Note:** Spaces are added to the ordering number shown above for viewing clarity only.

**Figure 13-3. OPN Example**

**Table 13-4. Valid OPN Combinations**

| Device Number | Speed Option | Package Type | Temperature Range |
|---|---|---|---|
| Au1500 | 333 | MB | C |
| | | | D |
| | | | I |
| | | | F |
| Au1500 | 400 | MB | C |
| | | | D |
| Au1500 | 500 | MB | C |
| | | | D |
| **Note:** Consult your local AMD sales office to confirm availability of specific valid combinations and to check on newly released combinations possibly not listed. | | | |

# A Support Documentation

## A.1 Memory Map

The peripheral devices on the Au1500 processor contain memory-mapped registers visible to software. Table A-1 contains the memory map for the peripheral devices and physical memory. The addresses are 36 bits wide.

**Table A-1. Basic Au1500™ Processor Physical Address Memory Map**

| Start Address | End Address | Size (MB) | Function |
|---|---|---|---|
| 0x0 0000 0000 | 0x0 0FFF FFFF | 256 | Memory KSEG 0/1 |
| 0x0 1000 0000 | 0x0 11FF FFFF | 32 | I/O Devices on Peripheral Bus |
| 0x0 1200 0000 | 0x0 13FF FFFF | 32 | Reserved |
| 0x0 1400 0000 | 0x0 17FF FFFF | 64 | I/O Devices on System Bus |
| 0x0 1800 0000 | 0x0 1FFF FFFF | 128 | Memory Mapped. 0x0 1FC0 0000 must contain the boot vector so this is typically where Flash or ROM is located. |
| 0x0 2000 0000 | 0x0 7FFF FFFF | 1536 | Memory Mapped |
| 0x0 8000 0000 | 0x0 EFFF FFFF | 1792 | Memory Mapped. Currently this space is memory mapped, but it should be considered reserved for future use. |
| 0x0 F000 0000 | 0x0 FFFF FFFF | 256 | Debug Probe |
| 0x1 0000 0000 | 0x3 FFFF FFFF | 4096 * 3 | Reserved |
| 0x4 0000 0000 | 0x4 FFFF FFFF | 4096 | PCI Uncached Memory Space |
| 0x5 0000 0000 | 0x5 FFFF FFFF | 4096 | PCI I/O Space |
| 0x6 0000 0000 | 0x6 FFFF FFFF | 4096 | PCI Configuration Space |
| 0x7 0000 0000 | 0xC FFFF FFFF | 4096 * 7 | Reserved |
| 0xD 0000 0000 | 0xD FFFF FFFF | 4096 | I/O Device |
| 0xE 0000 0000 | 0xE FFFF FFFF | 4096 | External LCD Controller Interface |
| 0xF 0000 0000 | 0xF FFFF FFFF | 4096 | PCMCIA Interface |

The Au1500 processor system bus devices are mapped at the addresses based at 0x0 1400 0000. See Table A-2 for complete addresses.

**Table A-2. System Bus Devices Physical Address Memory Map**

| Start Address | End Address | Size | Function |
|---|---|---|---|
| 0x0 1400 0000 | 0x0 1400 0FFF | 4 KB | SDRAM Memory Controller |
| 0x0 1400 1000 | 0x0 1400 1FFF | 4 KB | SRAM/Flash Memory Controller |
| 0x0 1400 2000 | 0x0 1400 2FFF | 4 KB | DMA |
| 0x0 1400 4000 | 0x0 1400 4FFF | 4 KB | Ethernet DMA |

The Au1500 processor peripheral bus devices are based at 0x0 1100 0000. The individual memory spaces of the devices are defined in Table A-3.

**Table A-3. Peripheral Bus Devices Physical Address Memory Map**

| Start Address | End Address | Size | Function |
|---|---|---|---|
| 0x0 1000 0000 | 0x0 100F FFFF | 1 MB | AC97 Controller |
| 0x0 1010 0000 | 0x0 101F FFFF | 1 MB | USB Host |
| 0x0 1020 0000 | 0x0 102F FFFF | 1 MB | USB Device |
| 0x0 1030 0000 | 0x0 103F FFFF | 1 MB | |
| 0x0 1040 0000 | 0x0 104F FFFF | 1 MB | Interrupt Controller 0 |
| 0x0 1050 0000 | 0x0 110F FFFF | 12 MB | |
| 0x0 1110 0000 | 0x0 111F FFFF | 1 MB | UART0 |
| 0x0 1120 0000 | 0x0 113F FFFF | 2 MB | |
| 0x0 1140 0000 | 0x0 114F FFFF | 1 MB | UART3 |
| 0x0 1150 0000 | 0x0 115F FFFF | 1 MB | Ethernet MAC |
| 0x0 1160 0000 | 0x0 116F FFFF | 1 MB | |
| 0x0 1170 0000 | 0x0 117F FFFF | 1 MB | Secondary GPIO |
| 0x0 1180 0000 | 0x0 118F FFFF | 1 MB | Interrupt Controller 1 |
| 0x0 1190 0000 | 0x0 119F FFFF | 1 MB | System Control: RTC, TOY, Timers, Primary GPIO, Power Management |

### A.1.1 Device Memory Map

Table A-4 lists all of the devices which are memory mapped to the Au1500 processor core. These devices are all mapped within KSEG1 (non-cached, non-TLB). All 32-bit addresses are translated into 36-bit addresses by changing bits [31:29] to zero and adding bits [35:32] which are set to zero.

**Table A-4. Device Memory Map**

| Register | KSEG1 Address | Physical Address |
|---|---|---|
| **AC97 Controller- Section 7.1 on page 102** | | |
| ac97_config | 0xB000 0000 | 0x0 1000 0000 |
| ac97_status | 0xB000 0004 | 0x0 1000 0004 |
| ac97_data | 0xB000 0008 | 0x0 1000 0008 |
| ac97_cmmd | 0xB000 000C | 0x0 100 0000C |
| ac97_cmmdresp | 0xB000 000C | 0x0 1000 000C |
| ac97_control | 0xB000 0010 | 0x0 1000 0010 |
| **USB Host Controller - Section 7.2 on page 107** | | |
| Open HCI Register Set Base | 0xB010 0000 | 0x0 1010 0000 |
| usbh_enable | 0xB017 FFFC | 0x0 1017 FFFC |
| **USB Device Controller - Section 7.3 on page 109** | | |
| usbd_ep0rd | 0xB020 0000 | 0x0 1020 0000 |
| usbd_ep0wr | 0xB020 0004 | 0x0 1020 0004 |
| usbd_ep1wr | 0xB020 0008 | 0x0 1020 0008 |
| usbd_ep2wr | 0xB020 000C | 0x0 1020 000C |
| usbd_ep3rd | 0xB020 0010 | 0x0 1020 0010 |
| usbd_ep4rd | 0xB020 0014 | 0x0 1020 0014 |
| usbd_inten | 0xB020 0018 | 0x0 1020 0018 |
| usbd_intstat | 0xB020 001C | 0x0 1020 001c |
| usbd_config | 0xB020 0020 | 0x0 1020 0020 |
| usbd_ep0cs | 0xB020 0024 | 0x0 1020 0024 |
| usbd_ep1cs | 0xB020 0028 | 0x0 1020 0028 |
| usbd_ep2cs | 0xB020 002C | 0x0 1020 002C |
| usbd_ep3cs | 0xB020 0030 | 0x0 1020 0030 |
| usbd_ep4cs | 0xB020 0034 | 0x0 1020 0034 |
| usbd_ep0rdstat | 0xB020 0040 | 0x0 1020 0040 |
| usbd_ep0wrstat | 0xB020 0044 | 0x0 1020 0044 |
| usbd_ep1wrstat | 0xB020 0048 | 0x0 1020 0048 |
| usbd_ep2wrstat | 0xB020 004c | 0x0 1020 004c |
| usbd_ep3rdstat | 0xB020 0050 | 0x0 1020 0050 |
| usbd_ep4rdstat | 0xB020 0054 | 0x0 1020 0054 |
| usbd_enable | 0xB020 0058 | 0x0 1020 0058 |
| **Interrupt Controller 0 - Section 6.2 on page 97** | | |
| ic0_cfg0rd | 0xB040 0040 | 0x0 1040 0040 |
| ic0_cfg0set | 0xB040 0040 | 0x0 1040 0040 |
| ic0_cfg0clr | 0xB040 0044 | 0x0 1040 0044 |
| ic0_cfg1rd | 0xB040 0048 | 0x0 1040 0048 |
| ic0_cfg1set | 0xB040 0048 | 0x0 1040 0048 |
| ic0_cfg1clr | 0xB040 004C | 0x0 1040 004C |
| ic0_cfg2rd | 0xB040 0050 | 0x0 1040 0050 |
| ic0_cfg2set | 0xB040 0050 | 0x0 1040 0050 |
| ic0_cfg2clr | 0xB040 0054 | 0x0 1040 0054 |
| ic0_req0int | 0xB040 0054 | 0x0 1040 0054 |
| ic0_srcrd | 0xB040 0058 | 0x0 1040 0058 |
| ic0_srcset | 0xB040 0058 | 0x0 1040 0058 |
| ic0_srcclr | 0xB040 005C | 0x0 1040 005C |
| ic0_req1int | 0xB040 005C | 0x0 1040 005C |
| ic0_assignrd | 0xB040 0060 | 0x0 1040 0060 |
| ic0_assignset | 0xB040 0060 | 0x0 1040 0060 |
| ic0_assignclr | 0xB040 0064 | 0x0 1040 0064 |
| ic0_wakerd | 0xB040 0068 | 0x0 1040 0068 |
| ic0_wakeset | 0xB040 006C | 0x0 1040 006C |
| ic0_wakeclr | 0xB040 0070 | 0x0 1040 0070 |
| ic0_maskrd | 0xB040 0070 | 0x0 1040 0070 |
| ic0_maskset | 0xB040 0074 | 0x0 1040 0074 |
| ic0_maskclr | 0xB040 0078 | 0x0 1040 0078 |
| ic0_risingrd | 0xB040 0078 | 0x0 1040 0078 |
| ic0_risingclr | 0xB040 007C | 0x0 1040 007C |
| ic0_fallingrd | 0xB040 007C | 0x0 1040 007C |
| ic0_fallingclr | 0xB040 0080 | 0x0 1040 0080 |
| **UART0 - Section 7.5.2 on page 140** | | |
| uart0_rxdata | 0xB110 0000 | 0x0 1110 0000 |
| uart0_txdata | 0xB110 0004 | 0x0 1110 0004 |
| uart0_inten | 0xB110 0008 | 0x0 1110 0008 |
| uart0_intcause | 0xB110 000C | 0x0 1110 000C |
| uart0_fifoctrl | 0xB110 0010 | 0x0 1110 0010 |
| uart0_linectrl | 0xB110 0014 | 0x0 1110 0014 |
| — | 0xB110 0018 | 0x0 1110 0018 |
| uart0_linestat | 0xB110 001C | 0x0 1110 001C |
| — | 0xB110 0020 | 0x0 1110 0020 |
| uart0_clkdiv | 0xB110 0028 | 0x0 1110 0028 |
| uart0_enable | 0xB110 0100 | 0x0 1110 0100 |
| **UART3 - Section 7.5.2 on page 140** | | |
| uart3_rxdata | 0xB140 0000 | 0x0 1140 0000 |
| uart3_txdata | 0xB140 0004 | 0x0 1140 0004 |

Table A-4. Device Memory Map (Continued)

| Register | KSEG1 Address | Physical Address |
|---|---|---|
| uart3_inten | 0xB140 0008 | 0x0 1140 0008 |
| uart3_intcause | 0xB140 000C | 0x0 1140 000C |
| uart3_fifoctrl | 0xB140 0010 | 0x0 1140 0010 |
| uart3_linectrl | 0xB140 0014 | 0x0 1140 0014 |
| uart3_mdmctrl | 0xB140 0018 | 0x0 1140 0018 |
| uart3_linestat | 0xB140 001C | 0x0 1140 001C |
| uart3_mdmstat | 0xB140 0020 | 0x0 1140 0020 |
| uart3_autoflow | 0xB140 0024 | 0x0 1140 0024 |
| uart3_clkdiv | 0xB140 0028 | 0x0 1140 0028 |
| uart3_enable | 0xB140 0100 | 0x0 1140 0100 |
| **Ethernet Controller MAC0 - Section 7.4.2 on page 120** | | |
| mac0_control | 0xB150 0000 | 0x0 1150 0000 |
| mac0_addrhigh | 0xB150 0004 | 0x0 1150 0004 |
| mac0_addrlow | 0xB150 0008 | 0x0 1150 0008 |
| mac0_hashhigh | 0xB150 000C | 0x0 1150 000C |
| mac0_hashlow | 0xB150 0010 | 0x0 1150 0010 |
| mac0_miictrl | 0xB150 0014 | 0x0 1150 0014 |
| mac0_miidata | 0xB150 0018 | 0x0 1150 0018 |
| mac0_flowctrl | 0xB150 001C | 0x0 1150 001C |
| mac0_vlan1 | 0xB150 0020 | 0x0 1150 0020 |
| mac0_vlan2 | 0xB150 0024 | 0x0 1150 0024 |
| **Ethernet Controller MAC1 - Section 7.4.2 on page 120** | | |
| mac1_control | 0xB151 0000 | 0x0 1151 0000 |
| mac1_addrhigh | 0xB151 0004 | 0x0 1151 0004 |
| mac1_addrlow | 0xB151 0008 | 0x0 1151 0008 |
| mac1_hashhigh | 0xB151 000C | 0x0 1151 000C |
| mac1_hashlow | 0xB151 0010 | 0x0 1151 0010 |
| mac1_miictrl | 0xB151 0014 | 0x0 1151 0014 |
| mac1_miidata | 0xB151 0018 | 0x0 1151 0018 |
| mac1_flowctrl | 0xB151 001C | 0x0 1151 001C |
| mac1_vlan1 | 0xB151 0020 | 0x0 1151 0020 |
| mac1_vlan2 | 0xB151 0024 | 0x0 1151 0024 |
| **Ethernet Controller Enable - Section 7.4.3 on page 128** | | |
| macen_mac0 | 0xB152 0000 | 0x0 1152 0000 |
| macen_mac1 | 0xB152 0004 | 0x0 1152 0004 |
| **Secondary GPIO - Section 7.6.2 on page 149** | | |
| gpio2_dir | 0xB170 0000 | 0x0 1170 0000 |
| reserved | 0xB170 0004 | 0x0 1170 0004 |
| gpio2_output | 0xB170 0008 | 0x0 1170 0008 |
| gpio2_pinstate | 0xB170 000C | 0x0 1170 000C |
| gpio2_inten | 0xB170 0010 | 0x0 1170 0010 |
| gpio2_enable | 0xB170 0014 | 0x0 1170 0014 |
| **Interrupt Controller 1 - Section 6.2 on page 97** | | |
| ic1_cfg0rd | 0xB180 0040 | 0x0 1180 0040 |
| ic1_cfg0set | 0xB180 0040 | 0x0 1180 0040 |
| ic1_cfg0clr | 0xB180 0044 | 0x0 1180 0044 |
| ic1_cfg1rd | 0xB180 0048 | 0x0 1180 0048 |
| ic1_cfg1set | 0xB180 0048 | 0x0 1180 0048 |
| ic1_cfg1clr | 0xB180 004C | 0x0 1180 004C |
| ic1_cfg2rd | 0xB180 0050 | 0x0 1180 0050 |
| ic1_cfg2set | 0xB180 0050 | 0x0 1180 0050 |
| ic1_cfg2clr | 0xB180 0054 | 0x0 1180 0054 |
| ic1_req0int | 0xB180 0054 | 0x0 1180 0054 |
| ic1_srcrd | 0xB180 0058 | 0x0 1180 0058 |
| ic1_srcset | 0xB180 0058 | 0x0 1180 0058 |
| ic1_srcclr | 0xB180 005C | 0x0 1180 005C |
| ic1_req1int | 0xB180 005C | 0x0 1180 005C |
| ic1_assignrd | 0xB180 0060 | 0x0 1180 0060 |
| ic1_assignset | 0xB180 0060 | 0x0 1180 0060 |
| ic1_assignclr | 0xB180 0064 | 0x0 1180 0064 |
| ic1_wakerd | 0xB180 0068 | 0x0 1180 0068 |
| ic1_wakeset | 0xB180 006C | 0x0 1180 006C |
| ic1_wakeclr | 0xB180 0070 | 0x0 1180 0070 |
| ic1_maskrd | 0xB180 0070 | 0x0 1180 0070 |
| ic1_maskset | 0xB180 0074 | 0x0 1180 0074 |
| ic1_maskclr | 0xB180 0078 | 0x0 1180 0078 |
| ic1_risingrd | 0xB180 0078 | 0x0 1180 0078 |
| ic1_risingclr | 0xB180 007C | 0x0 1180 007C |
| ic1_fallingrd | 0xB180 007C | 0x0 1180 007C |
| ic1_fallingclr | 0xB180 0080 | 0x0 1180 0080 |
| **Clock Controller - Section 8.1.1 on page 155** | | |
| sys_freqctrl0 | 0xB190 0020 | 0x0 1190 0020 |
| sys_freqctrl1 | 0xB190 0024 | 0x0 1190 0024 |
| sys_clksrc | 0xB190 0028 | 0x0 1190 0028 |
| sys_cpupll | 0xB190 0060 | 0x0 1190 0060 |
| sys_auxpll | 0xB190 0064 | 0x0 1190 0064 |
| **TOY & RTC - Section 8.2.1 on page 164** | | |
| sys_toytrim | 0xB190 0000 | 0x0 1190 0000 |
| sys_toywrite | 0xB190 0004 | 0x0 1190 0004 |
| sys_matchtoy0 | 0xB190 0008 | 0x0 1190 0008 |
| sys_matchtoy1 | 0xB190 000C | 0x0 1190 000C |
| sys_matchtoy2 | 0xB190 0010 | 0x0 1190 0010 |

Table A-4. Device Memory Map (Continued)

| Register | KSEG1 Address | Physical Address |
|---|---|---|
| sys_cntrctrl | 0xB190 0014 | 0x0 1190 0014 |
| sys_toyread | 0xB190 0040 | 0x0 1190 0040 |
| sys_rtctrim | 0xB190 0044 | 0x0 1190 0044 |
| sys_rtcwrite | 0xB190 0048 | 0x0 1190 0048 |
| sys_rtcmatch0 | 0xB190 004C | 0x0 1190 004C |
| sys_rtcmatch1 | 0xB190 0050 | 0x0 1190 0050 |
| sys_rtcmatch2 | 0xB190 0054 | 0x0 1190 0054 |
| sys_rtcread | 0xB190 0058 | 0x0 1190 0058 |
| **Primary GPIO - Section 8.3.2 on page 169** | | |
| sys_pinfunc | 0xB190 002C | 0x0 1190 002C |
| sys_trioutrd | 0xB190 0100 | 0x0 1190 0100 |
| sys_trioutclr | 0xB190 0100 | 0x0 1190 0100 |
| sys_outputrd | 0xB190 0108 | 0x0 1190 0108 |
| sys_outputset | 0xB190 0108 | 0x0 1190 0108 |
| sys_outputclr | 0xB190 010C | 0x0 1190 010C |
| sys_pinstaterd | 0xB190 0110 | 0x0 1190 0110 |
| sys_pininputen | 0xB190 0110 | 0x0 1190 0110 |
| **Power Management - Section 8.4.5 on page 175** | | |
| sys_scratch0 | 0xB190 0018 | 0x0 1190 0018 |
| sys_scratch1 | 0xB190 001C | 0x0 1190 001C |
| sys_wakemsk | 0xB190 0034 | 0x0 1190 0034 |
| sys_endian | 0xB190 0038 | 0x0 1190 0038 |
| sys_powerctrl | 0xB190 003C | 0x0 1190 003C |
| sys_wakesrc | 0xB190 005C | 0x0 1190 005C |
| sys_slppwr | 0xB190 0078 | 0x0 1190 0078 |
| sys_sleep | 0xB190 007C | 0x0 1190 007C |
| **SDRAM Controller - Section 3.1.2 on page 45** | | |
| mem_sdmode0 | 0xB190 0000 | 0x0 1400 0000 |
| mem_sdmode1 | 0xB190 0004 | 0x0 1400 0004 |
| mem_sdmode2 | 0xB190 0008 | 0x0 1400 0008 |
| mem_sdaddr0 | 0xB190 000C | 0x0 1400 000C |
| mem_sdaddr1 | 0xB190 0010 | 0x0 1400 0010 |
| mem_sdaddr2 | 0xB190 0014 | 0x0 1400 0014 |
| mem_sdrefcfg | 0xB190 0018 | 0x0 1400 0018 |
| mem_sdprecmd | 0xB190 001C | 0x0 1400 001C |
| mem_sdautoref | 0xB190 0020 | 0x0 1400 0020 |
| mem_sdwrmd0 | 0xB190 0024 | 0x0 1400 0024 |
| mem_sdwrmd1 | 0xB190 0028 | 0x0 1400 0028 |
| mem_sdwrmd2 | 0xB190 002C | 0x0 1400 002C |
| mem_sdsleep | 0xB190 0030 | 0x0 1400 0030 |
| mem_sdsmcke | 0xB190 0034 | 0x0 1400 0034 |

| Register | KSEG1 Address | Physical Address |
|---|---|---|
| **Static Bus Controller - Section 3.2.1 on page 53** | | |
| mem_stcfg0 | 0xB190 1000 | 0x0 1400 1000 |
| mem_sttime0 | 0xB190 1004 | 0x0 1400 1004 |
| mem_staddr0 | 0xB190 1008 | 0x0 1400 1008 |
| mem_stcfg1 | 0xB190 1010 | 0x0 1400 1010 |
| mem_sttime1 | 0xB190 1014 | 0x0 1400 1014 |
| mem_staddr1 | 0xB190 1018 | 0x0 1400 1018 |
| mem_stcfg2 | 0xB190 1020 | 0x0 1400 1020 |
| mem_sttime2 | 0xB190 1024 | 0x0 1400 1024 |
| mem_staddr2 | 0xB190 1028 | 0x0 1400 1028 |
| mem_stcfg3 | 0xB190 1030 | 0x0 1400 1030 |
| mem_sttime3 | 0xB190 1034 | 0x0 1400 1034 |
| mem_staddr3 | 0xB190 1038 | 0x0 1400 1038 |
| **DMA Controller 0 - Section 5.1 on page 87** | | |
| dma0_moderead | 0xB190 2000 | 0x0 1400 2000 |
| dma0_modeset | 0xB190 2000 | 0x0 1400 2000 |
| dma0_modeclr | 0xB190 2004 | 0x0 1400 2004 |
| dma0_peraddr | 0xB190 2008 | 0x0 1400 2008 |
| dma0_buf0addr | 0xB190 200C | 0x0 1400 200C |
| dma0_buf0size | 0xB190 2010 | 0x0 1400 2010 |
| dma0_buf1addr | 0xB190 2014 | 0x0 1400 2014 |
| dma0_buf1size | 0xB190 2018 | 0x0 1400 2018 |
| **DMA Controller 1 - Section 5.1 on page 87** | | |
| dma1_moderead | 0xB190 2100 | 0x0 1400 2100 |
| dma1_modeset | 0xB190 2100 | 0x0 1400 2100 |
| dma1_modeclr | 0xB190 2104 | 0x0 1400 2104 |
| dma1_peraddr | 0xB190 2108 | 0x0 1400 2108 |
| dma1_buf0addr | 0xB190 210C | 0x0 1400 210C |
| dma1_buf0size | 0xB190 2110 | 0x0 1400 2110 |
| dma1_buf1addr | 0xB190 2114 | 0x0 1400 2114 |
| dma1_buf1size | 0xB190 2118 | 0x0 1400 2118 |
| **DMA Controller 2 - Section 5.1 on page 87** | | |
| dma2_moderead | 0xB190 2200 | 0x0 1400 2200 |
| dma2_modeset | 0xB190 2200 | 0x0 1400 2200 |
| dma2_modeclr | 0xB190 2204 | 0x0 1400 2204 |
| dma2_peraddr | 0xB190 2208 | 0x0 1400 2208 |
| dma2_buf0addr | 0xB190 220C | 0x0 1400 220C |
| dma2_buf0size | 0xB190 2210 | 0x0 1400 2210 |
| dma2_buf1addr | 0xB190 2214 | 0x0 1400 2214 |
| dma2_buf1size | 0xB190 2218 | 0x0 1400 2218 |

Table A-4. Device Memory Map (Continued)

| Register | KSEG1 Address | Physical Address |
|---|---|---|
| **DMA Controller 3 - Section 5.1 on page 87** | | |
| dma3_moderead | 0xB190 2300 | 0x0 1400 2300 |
| dma3_modeset | 0xB190 2300 | 0x0 1400 2300 |
| dma3_modeclr | 0xB190 2304 | 0x0 1400 2304 |
| dma3_peraddr | 0xB190 2308 | 0x0 1400 2308 |
| dma3_buf0addr | 0xB190 230C | 0x0 1400 230C |
| dma3_buf0size | 0xB190 2310 | 0x0 1400 2310 |
| dma3_buf1addr | 0xB190 2314 | 0x0 1400 2314 |
| dma3_buf1size | 0xB190 2318 | 0x0 1400 2318 |
| **DMA Controller 4- Section 5.1 on page 87** | | |
| dma4_moderead | 0xB190 2400 | 0x0 1400 2400 |
| dma4_modeset | 0xB190 2400 | 0x0 1400 2400 |
| dma4_modeclr | 0xB190 2404 | 0x0 1400 2404 |
| dma4_peraddr | 0xB190 2408 | 0x0 1400 2408 |
| dma4_buf0addr | 0xB190 240C | 0x0 1400 240C |
| dma4_buf0size | 0xB190 2410 | 0x0 1400 2410 |
| dma4_buf1addr | 0xB190 2414 | 0x0 1400 2414 |
| dma4_buf1size | 0xB190 2418 | 0x0 1400 2418 |
| **DMA Controller 5 - Section 5.1 on page 87** | | |
| dma5_moderead | 0xB190 2500 | 0x0 1400 2500 |
| dma5_modeset | 0xB190 2500 | 0x0 1400 2500 |
| dma5_modeclr | 0xB190 2504 | 0x0 1400 2504 |
| dma5_peraddr | 0xB190 2508 | 0x0 1400 2508 |
| dma5_buf0addr | 0xB190 250C | 0x0 1400 250C |
| dma5_buf0size | 0xB190 2510 | 0x0 1400 2510 |
| dma5_buf1addr | 0xB190 2514 | 0x0 1400 2514 |
| dma5_buf1size | 0xB190 2518 | 0x0 1400 2518 |
| **DMA Controller 6 - Section 5.1 on page 87** | | |
| dma6_moderead | 0xB190 2600 | 0x0 1400 2600 |
| dma6_modeset | 0xB190 2600 | 0x0 1400 2600 |
| dma6_modeclr | 0xB190 2604 | 0x0 1400 2604 |
| dma6_peraddr | 0xB190 2608 | 0x0 1400 2608 |
| dma6_buf0addr | 0xB190 260C | 0x0 1400 260C |
| dma6_buf0size | 0xB190 2610 | 0x0 1400 2610 |
| dma6_buf1addr | 0xB190 2614 | 0x0 1400 2614 |
| dma6_buf1size | 0xB190 2618 | 0x0 1400 2618 |
| **DMA Controller 7 - Section 5.1 on page 87** | | |
| dma7_moderead | 0xB190 2700 | 0x0 1400 2700 |
| dma7_modeset | 0xB190 2700 | 0x0 1400 2700 |
| dma7_modeclr | 0xB190 2704 | 0x0 1400 2704 |
| dma7_peraddr | 0xB190 2708 | 0x0 1400 2708 |
| dma7_buf0addr | 0xB190 270C | 0x0 1400 270C |
| dma7_buf0size | 0xB190 2710 | 0x0 1400 2710 |
| dma7_buf1addr | 0xB190 2714 | 0x0 1400 2714 |
| dma7_buf1size | 0xB190 2718 | 0x0 1400 2718 |
| **Ethernet Controller DMA Channels -Section 7.4.4 on page 129** | | |
| macdma0_tx0stat | 0xB190 4000 | 0x0 1400 4000 |
| macdma0_tx0addr | 0xB190 4004 | 0x0 1400 4004 |
| macdma0_tx0len | 0xB190 4008 | 0x0 1400 4008 |
| macdma0_tx1stat | 0xB190 4010 | 0x0 1400 4010 |
| macdma0_tx1addr | 0xB190 4014 | 0x0 1400 4014 |
| macdma0_tx1len | 0xB190 4018 | 0x0 1400 4018 |
| macdma0_tx2stat | 0xB190 4020 | 0x0 1400 4020 |
| macdma0_tx2addr | 0xB190 4024 | 0x0 1400 4024 |
| macdma0_tx2len | 0xB190 4028 | 0x0 1400 4028 |
| macdma0_tx3stat | 0xB190 4030 | 0x0 1400 4030 |
| macdma0_tx3addr | 0xB190 4034 | 0x0 1400 4034 |
| macdma0_tx3len | 0xB190 4038 | 0x0 1400 4038 |
| macdma0_rx0stat | 0xB190 4100 | 0x0 1400 4100 |
| macdma0_rx0addr | 0xB190 4104 | 0x0 1400 4104 |
| macdma0_rx1stat | 0xB190 4110 | 0x0 1400 4110 |
| macdma0_rx1addr | 0xB190 4114 | 0x0 1400 4114 |
| macdma0_rx2stat | 0xB190 4120 | 0x0 1400 4120 |
| macdma0_rx2addr | 0xB190 4124 | 0x0 1400 4124 |
| macdma0_rx3stat | 0xB190 4130 | 0x0 1400 4130 |
| macdma0_rx3addr | 0xB190 4134 | 0x0 1400 4134 |
| macdma1_tx0stat | 0xB190 4200 | 0x0 1400 4200 |
| macdma1_tx0addr | 0xB190 4204 | 0x0 1400 4204 |
| macdma1_tx0len | 0xB190 4208 | 0x0 1400 4208 |
| macdma1_tx1stat | 0xB190 4210 | 0x0 1400 4210 |
| macdma1_tx1addr | 0xB190 4214 | 0x0 1400 4214 |
| macdma1_tx1len | 0xB190 4218 | 0x0 1400 4218 |
| macdma1_tx2stat | 0xB190 4220 | 0x0 1400 4220 |
| macdma1_tx2addr | 0xB190 4224 | 0x0 1400 4224 |
| macdma1_tx2len | 0xB190 4228 | 0x0 1400 4228 |
| macdma1_tx3stat | 0xB190 4230 | 0x0 1400 4230 |
| macdma1_tx3addr | 0xB190 4234 | 0x0 1400 4234 |
| macdma1_tx3len | 0xB190 4238 | 0x0 1400 4238 |
| macdma1_rx0stat | 0xB190 4300 | 0x0 1400 4300 |
| macdma1_rx0addr | 0xB190 4304 | 0x0 1400 4304 |
| macdma1_rx1stat | 0xB190 4310 | 0x0 1400 4310 |

Table A-4. Device Memory Map (Continued)

| Register | KSEG1 Address | Physical Address |
|---|---|---|
| macdma1_rx1addr | 0xB190 4314 | 0x0 1400 4314 |
| macdma1_rx2stat | 0xB190 4320 | 0x0 1400 4320 |
| macdma1_rx2addr | 0xB190 4324 | 0x0 1400 4324 |
| macdma1_rx3stat | 0xB190 4330 | 0x0 1400 4330 |
| macdma1_rx3addr | 0xB190 4334 | 0x0 1400 4334 |

## A.1.2 Programming Tips

### A.1.2.1 Memory Mapped Registers

Peripheral, or system device registers should all be marked with the CCA bits to non-cacheable. Access must be on 32-bit boundaries, one 32-bit value at a time. See Section 2.2 "Caches" on page 17 for more information.

## A.2 Data Book Notations

This section addresses some of the terminology used in this book.

### A.2.1 Unpredictable and Undefined

The terms UNPREDICTABLE and UNDEFINED are used throughout this book to describe the behavior of the processor in certain cases. UNDEFINED behavior or operations can occur only as the result of executing instructions in a privileged mode (i.e., in Kernel Mode or Debug Mode, or with the CP0 usable bit set in the Status register). Unprivileged software can never cause UNDEFINED behavior or operations. Conversely, both privileged and unprivileged software can cause UNPREDICTABLE results or operations.

#### A.2.1.1 Unpredictable

UNPREDICTABLE results may vary from processor implementation to implementation, instruction to instruction, or as a function of time on the same implementation or instruction. Software can never depend on results that are UNPREDICTABLE. UNPREDICTABLE operations may cause a result to be generated or not. If a result is generated, it is UNPREDICTABLE. UNPREDICTABLE operations may cause arbitrary exceptions.

UNPREDICTABLE results or operations have several implementation restrictions:

- Implementations of operations generating UNPREDICTABLE results must not depend on any data source (memory or internal state) which is inaccessible in the current processor mode
- UNPREDICTABLE operations must not read, write, or modify the contents of memory or internal state which is inaccessible in the current processor mode. For example, UNPREDICTABLE operations executed in user mode must not access memory or internal state that is only accessible in Kernel Mode or Debug Mode or in another process
- UNPREDICTABLE operations must not halt or hang the processor

UNPRED, used to describe the default state of registers, should be taken as meaning UNPREDICATABLE.

#### A.2.1.2 Undefined

UNDEFINED operations or behavior may vary from processor implementation to implementation, instruction to instruction, or as a function of time on the same implementation or instruction. UNDEFINED operations or behavior may vary from nothing to creating an environment in which execution can no longer continue. UNDEFINED operations or behavior may cause data loss.

UNDEFINED operations or behavior has one implementation restriction:

- UNDEFINED operations or behavior must not cause the processor to hang (that is, enter a state from which there is no exit other than powering down the processor). The assertion of any of the reset signals must restore the processor to an operational state.

### A.2.2 Register Fields

In general, fields marked as reserved should be considered UNPREDICTABLE. In other words, these fields should be written zeros and ignored on a read to preserve future compatibility.

## A.3 Differences between Au1500™ and Au1000™ Processors

### A.3.1 Peripherals

The Au1500 processor does not have the following peripherals that are present on the Au1000 processor:

- IrDA
- UART1
- UART2
- SSI
- $I^2S$

The Au1500 processor has added these functions not present on the Au1000 processor:

- PCI 2.2 compliant interface
- General Purpose I/O (GPIO): 39 total, 22 dedicated. (Au1000 has 32 total, 5 dedicated.)

### A.3.2 Miscellaneous

The NIC base address has changed from 0x0 1050 0000 to 0x0 1150 0000.

Some inputs to the interrupt controller have changed due to the addition/removal of blocks. Refer to the interrupt controller section for the Au1500 processor interrupt map.

A new CCA encoding has been added to the Au1500 processor. If CCA == 4, all system bus accesses will be cacheline aligned (i.e., no cacheline wrapping is supported). When the PCI Cacheable Memory space is used, it must be mapped to CCA == 4.

JTAG memory overlay of the PCI boot memory is not supported.

## A.4 Data Book Revision History

This document is a report of the revision/creation process of the data book for the Au1500 processor. Any revisions (i.e., additions, deletions, parameter corrections, etc.) are recorded in the table(s) below.

**Table A-5. Revision History**

| Revision (Date) | Description |
|---|---|
| A | See the *AMD Alchemy™ Au1500™ Processor Specification Update* (publication ID 27362). |
| B | |
| C<br>(June 2005) | |
| D<br>(March 2006) | Reformatted to bring page count down and correct some minor errors. See Table A-6 for details. |

**Table A-6. Edits to Current Revision**

| Section | Revisions / Comments |
|---|---|
| **All Sections / General** | • Reformatted document for page, figure, table and section titles.<br>— All registers now have either a Heading 3 or 4 associated with it so the electronic PDF will be more useful.<br>• Changed active low signals to use “#” instead of an overbar (e.g., $\overline{\text{ACRST}}$ changed to ACRST#).<br>• Omitted Index and added back cover page.<br>• Removed “Preliminary” mark. |
| **Section 1.0 "Overview"** | • Figure 1-1 "Block Diagram" on page 13:<br>— Changed pages (moved forward to first page of Section 1.0).<br>• Section 1.2 "Features" on page 14:<br>— - Modified second bullet under HIgh-Bandwidth Memory Buses (removed “with NAND/NOR Flash support”).<br>• Moved what was Section 1.3 “Data Book Notations” and Section 1.4 “Differences between Au1500™ and Au1000™ Processors” in rev c to the Appendix, Sections A.2 and A.3, respectively. |
| **Section 5.0 "DMA Controller"** | • Table 5-1 "DMA Channel Base Addresses" on page 87:<br>— Corrected cell heading from “KSEG0” to “KSEG1”.<br>• Section 5.1.4 "DMA Channel Buffer Size Registers" on page 92:<br>— PDF of rev C was messed up and could not be read. Fixed. |
| **Section 8.0 "System Control"** | • Section 8.4.4 "Device Power Management - Sleep" on page 174:<br>— In the second list of steps, corrected physical address in step 2 (i.e., changed from “0x1FC0_0000 to “0x0 1FC0 0000”.<br>• Section 8.3.1.1 "Pin Function" on page 168:<br>— Added active low designator to UART3 signals U3CTS#, U3DSR#, U3DCD#, U3RI#, U3RTS#, U3DTR# in bit 7. |
| **Section 9.0 "Power-up, Reset and Boot"** | • Section 9.3 "Boot" on page 182:<br>— Corrected the way KSEG1 and Physical addresses were called out (i.e., changed “0xBFC0_0000” to “0xBFC0 0000” and “0x1FC0_0000” to “0x0 1FC0 0000”. |

Table A-6. Edits to Current Revision

| Section | Revisions / Comments |
|---|---|
| **Section 11.0 "Signal Descriptions"** | • Section 11.0 "Signal Descriptions" on page 200:<br>— Modified "Note" to reflect use of "#" instead of "overbar" to indicate active low signal.<br>• Figure 11-1 "External Signals" on page 201:<br>— Added active low designator to UART3 signals U3CTS#, U3DSR#, U3DCD#, U3RI#, U3RTS#, U3DTR#.<br>• Table 11-3 "Signal Description", "UART3" on page 207:<br>— Added active low designator to UART3 signals U3CTS#, U3DSR#, U3DCD#, U3RI#, U3RTS#, U3DTR#.<br>• Table 11-3 "Signal Description", "GPIO" on page 210:<br>— Added active low designator to UART3 signals mentioned in GPIO9 (U3CTS#), GPIO10 (U3DSR#), GPIO11 (U3DCD#), GPIO12 (U3RI#), GPIO13 (U3RTS#), and GPIO14 (U3DTR#). |
| **Section 12.0 "Electrical and Thermal Specifications"** | • Table 12-1 "Absolute Maximum Ratings" on page 216:<br>— Updated table to reflect $T_{CASE}$ Commercial and $T_{CASE}$ Industrial.<br>• Section 12.5.1 "Power and Voltage for 333, 400, and 500 MHz Rated Parts" on page 218:<br>— This section was three sections (one section for each part), combined into one section. |
| **Section 13.0 "Packaging, Pin Assignments, and Ordering Information"** | • Figure 13-2 "Connection Diagram—Top View" on page 240:<br>— Added active low designator to UART3 signals U3CTS#, U3DSR#, U3DCD#, U3RI#, U3RTS#, U3DTR#.<br>• Table 13-1 "Pin Assignment — Sorted by Pin Number" on page 242:<br>— Added active low designator to UART3 signals U3CTS#, U3DSR#, U3DCD#, U3RI#, U3RTS#, U3DTR#.<br>• Table 13-2 "Pin Assignment — Sorted Alphabetically by Default Signal" on page 248:<br>— Added active low designator to UART3 signals U3CTS#, U3DSR#, U3DCD#, U3RI#, U3RTS#, U3DTR#.<br>• Table 13-3 "Pin Assignment — Alternate Signals Sorted Alphabetically" on page 253:<br>— Added active low designator to UART3 signals U3CTS#, U3DSR#, U3DCD#, U3RI#, U3RTS#, U3DTR#.<br>• Figure 13-3 "OPN Example" and Table 13-4 "Valid OPN Combinations" on page 254:<br>— Updated to reflect $T_{CASE}$ instead of $T_A$ in the Temperature Range description.<br>— Added Commerical and Industrial Pb-Free data. |

www.amd.com