Description

The R4300i is a low-cost RISC microprocessor optimized for demanding consumer applications. The R4300i provides performance equivalent to a high-end PC at a cost point to enable set-top terminals, games and portable consumer devices. The R4300i is compatible with the MIPS R4000 family of RISC microprocessors and will run all existing MIPS software. Unlike its predecessors designed for use in workstations, the R4300i is expected to lower the cost of systems in which it is used, a requirement for price-sensitive consumer products. The R4300i is also an effective embedded processor, supported by currently available development tools and providing very high performance at a low price-point.

Features

Low Power Dissipation:
- 1.8W (normal)
High Performance at 100MHz:
- 60 SPECint92
- 45 SPECfp92
High bandwidth interface:
- Max throughput 250MB/s
- 32-bit address/data bus
- 4-deep flush buffers
High integration on-chip:
- 16K I-cache, 8K D-cache
- Unified datapath
- 32 double-entry J-TLB
- 2 entry micro I-TLB
Power Management Features:
- Reduced Power Mode
- Instant On/Off
R4000 family compatibility:
- R4000 software compatible
- Runs Windows NT/UNIX

Overview

The success of the MIPS R3000 processor and its derivatives has established the MIPS architecture as an attractive high-performance choice in emerging consumer applications such as interactive TV and games. The R4300i is the 64-bit successor to the R3000 for this class of applications. It is specifically designed to be extremely low-cost from the outset, yet supply the performance necessary for full interactivity.

The R4300i achieves its low-cost goal by holding to a very small die size, simple package and low testing costs. It is further enabled for consumer applications by easily interfacing to other low-cost components. Its low power consumption also reduces system cost by combining high clock speed with limited power drawn, saving on power supply and heat dissipation costs.
Overview (cont.)

The R4300i has a number of design features to reduce power. These include an all-dynamic design, unified integer and floating-point datapath, a reduced power mode of operation, and caches partitioned into separate banks.

Printers, networking devices and other embedded applications will also benefit from the high performance, low cost and high-bandwidth interface of the R4300i. The R4300i is the next standard in low-cost processing in the MIPS Microprocessor family.

R4300i Architecture

Data Format

The R4300i defines a 64-bit double-word, a 32-bit word, a 16-bit halfword and an 8-bit byte. The byte ordering is configurable in either Big-endian or Little-endian format. Figures 1 and 2 show the ordering of bytes for Big-endian and Little-endian conventions.

The width of these registers depends on the mode of operation. In 32-bit mode, they are treated as 32 bits wide. In 64-bit mode, they are treated as 64 bits wide.

The R4300i also contains six special registers: the program counter (PC), multiply/divide result Hi and Lo, Load/Link (LL) bit, and floating point Implementation and Control registers FCR0 and FCR31; shown in Figure 4. The program counter register contains the address of the current instruction. The multiply/divide registers store the result of integer multiply operations and the quotient and remainder of integer divide operations. The load/link bit is dedicated for load-link and store-conditional instructions which can be used to perform SYNC operations. The two floating point control registers, FCR0 and FCR31, provide the implementation/revision information and control/status of the floating point coprocessor (CP1).

In addition, the R4300i operates with up to three tightly coupled coprocessors (designated CP0 through CP2). Coprocessor zero (CP0) is implemented as an integral part of the CPU and supports the virtual memory system together with exception handling. Coprocessor one (CP1) is reserved for the floating point unit, while coprocessor two is reserved for future use.

Coprocessor zero contains registers shown in Figure 5 plus a 32-entry TLB with each entry consisting of an even/odd pair of physical addresses and tag field.

Processor Resources

The R4300i CPU provides sixty-four 64-bit wide registers. Thirty-two of these registers, referred to as General Purpose Register (GPRs), are reserved for integer operations while the other thirty-two register, referred to as Floating Point General Purpose Register (FGRs), are reserved for floating point operations. These two register sets are shown in Figure 3.
Figure 5 Coprocessor-0 Registers

The CP0 registers can be considered as two distinct functional sets. The first set comprises those which support the TLB operations. The second set is formed from those registers that reflect the status of the processor. All of the CP0 registers are accessible by software.

Instruction Set

All R4300i instructions are 32 bits (single word) long aligned on word boundaries, as shown in Figure 6.

This approach simplifies instruction decoding. More complicated operations and addressing modes are synthesized by the compiler. There are three instruction formats: I-type (immediate), J-type (jump), and R-type (register).

Load and Store instructions move data between memory and the general registers. They are all immediate (I-type) instructions. The only addressing mode that load and store instructions directly support is base register plus 16-bit signed extended immediate offset.

Load and store instruction opcodes determine the access type which indicates the size of the data item to be loaded or stored. Regardless of the access type or byte-numbering order (endianness), the address field specifies the lowest byte of the address location being accessed.

Computational instructions perform arithmetic, logical, shift, multiply, and divide operations on values in registers. They occur in both register (R-Type) format, in which both operands are registers, and immediate (I-Type) format, in which one operand is a 16-bit immediate. When operating in 64-bit mode, 32-bit operands must be correctly sign extended.

Jump and branch instructions change the control flow of a program. They occur in both register (R-Type) format, and immediate (I-Type) format. All jump and branch instructions occur with an architecture delay of one instruction; that is, the instruction immediately following the jump or branch is always executed while the target instruction is being fetched.

Special instructions allow the software to initiate traps and are always R-Type.

Exception instructions offer a trapping mechanism to assist in software debug.

Coprocessor instructions perform operations in the respective coprocessors. Coprocessor loads and stores are I-type, and coprocessor computational instructions have coprocessor dependent formats.

Pipeline

The R4300i processor has a five-stage execution pipeline, as shown in Figure 7. Each pipeline stage takes one pclock to execute. The frequency of pclock can be either 1, 1.5, 2, or 3 times the frequency of the MasterClock, depending on the state of DivMode 1:0 signals. The execution of each instruction thus has a latency of at least five pcycles. Once the pipeline has been completely filled, five instructions are always being executed simultaneously. When the pipeline is not stalled, the processor has a throughput of one instruction per pcycle. The pipeline is in-order issue, in-order execution, and in-order completion, the same order as in the instruction stream.
The five stages of the R4300i pipeline are: Instruction Cache (IC), Instruction Decode and Register File Read (RF), Execution (EX), Data Cache Read (DC), and Register File or Data Cache Write Back (WB). Figure 7 shows the activities occurring during each pipeline stage for ALU, load and store, and branch instructions. Below are detail descriptions of the activities during each stage.

**IC:** An instruction address is presented by the address unit and the instruction cache fetch begins. The instruction micro-TLB starts the virtual-to-physical address translation.

**RF:** The instruction becomes available and the instruction decoder decodes the instruction and checks for interlock conditions. The instruction cache tag is checked against the page frame number obtained from the micro-TLB. Any required operands are read from the register file and the result from the EX or DC stages is bypassed to the following EX stage if required. The address unit generates the next instruction address.

**EX:** For ALU class instructions, the ALU performs an arithmetic or logical operation. For load and store class instructions, the ALU generates the data virtual address. For branch instructions, the ALU determines whether the branch condition is true.

**DC:** For load and store instructions, the data cache is accessed and data virtual-to-physical address translation is performed. At the end of the DC stage, the data becomes available and the load-align shifts the data to a word or double-word boundary. In addition, the data cache tag is checked against the page frame number obtained from the joint TLB.

**WB:** For register-to-register or load instructions, the result is written back to the register file. For store instructions, the data cache is updated with the store data.

The R4300i has a branch delay of one cycle and a load delay of one cycle. The branch delay is observed by noting that the branch compare logic operates during the EX pipestage, producing the target address which is available for the IC pipestage of the second subsequent instruction. The first subsequent instruction is the instruction in the delay slot and will be allowed to complete whether the branch is taken or not, as illustrated in Figure 8.

Similarly, the load delay of one is evident when the completion of a load at the end of the DC pipeline stage produces an operand which is available for the IC pipestage of the second subsequent instruction. The load delay is illustrated in Figure 9.
pipeline. On the other hand an exception aborts the relevant instruction and all those that follow.

Figure 10 illustrates the various interlock conditions and the different types of exceptions, at their pre-defined pipeline stages.

The execution unit is tightly coupled to the on-chip cache memory system, instruction and data caches, and the on-chip memory management unit, CP0. This unit has a multifunction pipe and is responsible for the execution of:

- Integer arithmetic and logic instructions
- Floating-point Coprocessor CP1 instructions
- Branch/Jump instructions
- Load/Store instructions
- Exception instructions
- Special instructions

All floating-point instructions, as defined in the MIPS ISA for the floating-point coprocessor CP1, are processed by the same hardware as used for the integer instructions. However, the execution of floating-point instructions can still be disabled via the “Coprocessor Unusable” CU bit defined in the CP0 Status register.

The execution unit uses a modular design approach to further reduce dynamic power consumption. Control logic is partitioned into small independent blocks responsible for a set of instructions. When relevant instructions are not in the instruction stream, the corresponding control blocks become inactive. Also, when functional elements in the datapath are idle, they operate on a constant (0’s or 1’s) selected to minimize power dissipation.

The execution unit’s datapath consists of: a 64-bit integer/mantissa datapath, an operand bypass network, 32 64-bit integer registers, 32 64-bit floating-point registers, a 12-bit exponent datapath, and a 64-bit instruction virtual address generator.

As shown in Figure 11, the integer/mantissa datapath is 64 bits wide and is compatible with both 32-bit and 64-bit operands for integer and floating-point numbers. It has a Boolean Logic functional unit, a Carry-Propagate Adder, a CSA Multiplier, and a bi-directional Shifter.

The Logical Operation unit performs all integer logical operations. The carry-propagated adder is used for all other integer and floating-point computational instructions. The adder is also used to compute data virtual address for load and store instructions, and to compare two operands in trap instructions. The CSA multiplier is used for both integer and floating-point multiplication, in single or double precision. The shifter is responsible for integer variable shifts, store align shifts, and floating-point post-normalization. It also has built-in guard/round/sticky collection logic for floating-point pre-alignment shift. In addition, the datapath has a Leading Zero Counter for floating-point normalization shift calculation, and a floating-point unpacker and repacker.
Figure 11 Integer/Mantissa Datapath

Figure 12 Exponent Datapath
For load and store class instructions, the datapath can handle partial-words in either big- or little-endian mode. For store instructions, the main bi-directional shifter performs an alignment shift on the register read data. No concatenation of register read data with the original memory data is necessary since the data cache has byte write enable controls. For load instructions, it is necessary to maintain a load delay of one pclock cycle. Due to the timing requirements imposed by this load delay, a dedicated byte-wide shifter (Load Aligner) is needed to shift the memory read data in bytes, halfwords, and words in the right or left direction.

The operand bypass network is built into the datapath to allow feedback of results from the EX and DC pipeline stages to the instructions in the following EX pipestage waiting to use the results as source operands rs and/or rt. This allows the following instruction to proceed without having to wait for the results to be written back to the register file. Similarly, to maintain the minimum branch delay slot of one pipeline clock cycle for all branch instructions on the floating-point condition, the results from the preceding floating-point compare instruction in the EX, DC, or WB pipestage will be fed back for branch condition testing in the RF pipestage.

The exponent datapath is 12 bits wide. The twelfth bit (MSB) is used as both sign bit and overflow bit. The exponent datapath consists of a feedback mux and 2 operand muxes to select the inputs from the adder, constant generating logic, a carry select adder, random logic to perform exception detection, and a register to hold the selected result from the adder, as shown in Figure 12.

The inputs to the exponent unit come from the unpack logic, where the exponents are extracted from single- or double-precision floating-point operands. The carry-selected adder performs exponent subtraction, pre-alignment shift calculation, and exponent addition for post-normalization final update. The result is sent to the repack logic to be merged with the mantissa.

The result of the exponent logic is compared with constants or ranges to check for various conditions by the result checker. These conditions include: underflow, overflow in single-precision number, overflow in double-precision number, one, zero, and convert limit check. The checks are performed as soon as data is available from the carry-select adder.

The instruction virtual address unit is responsible for the generation of 64-bit instruction virtual addresses to be used by the micro-TLB, I-Cache and CP0. It has its own incrementor to calculate the next sequential address. It also has an equality comparator and a separate ripple-carry adder to generate the branch target address.

In addition, the address unit has exception vector generator logic to decode the type of exception and then present the appropriate vector as the next PC address. It also has the exception PC register pipe chain to maintain a history of PC addresses for each pipestage so that the PC address associated with the exception causing instruction can be loaded into the Exception Program Counter (EPC) register.

**Cache Organization**

To achieve high performance, increase memory access bandwidth and reduce the latency of load and store instructions, the R4300i processor incorporates on-chip instruction and data caches. Each cache has its own 64-bit datapath and can be accessed in parallel with the other cache. Both the instruction and data caches are direct mapped, virtually indexed, and use physical tags. The R4300i cache organization is shown in Figure 13.

The instruction cache is 16 kilobytes in size. It is organized as eight-word (32-byte) lines with a 21-bit tag entry associated with each line. The tag entry consists of a valid bit (V), and a 20-bit physical tag (bit 31:12 of the physical address). The format of an instruction cache line is shown in Figure 14.
An instruction cache line has two possible cache states: Invalid and Valid. A cache line with invalid cache state does not contain valid information. A cache line in a valid state contains valid information.

The instruction cache is accessible in one p-cycle. Access begins on phase 2 of the IC pipestage and completes at the end of phase 1 of the RF pipestage. Each access will fetch two instructions. Therefore instruction fetching is required only on every other run cycle or when there is a jump/branch instruction in the EX pipestage. When there is a miss detected during an instruction cache access, a memory block read will be initiated from the system interface to replace the current cache line with the desired line.

The data cache is 8 kilobytes in size. It is organized as four-word (16-byte) lines with a 22-bit tag entry associated with each line. The tag entry consists of a valid bit (V), a dirty bit (D), and a 20 bit physical tag (bit 31:12 of the physical address). The format of a data cache line is shown in Figure 15.

A data cache line has three possible cache states: invalid, valid clean, and valid dirty. A data cache line with an invalid state does not contain valid information. A cache line in valid clean state contains valid information and is consistent with main memory. A cache line in valid dirty state contains valid data but is not consistent with main memory. Figure 16 illustrates the data cache state transition sequence.

The data cache is accessible on reads in one p-cycle. The access begins on phase 1 of the DC pipestage and completes at the end of phase 2 of the DC pipestage. Each access will fetch a double word. The data cache writes, however, execute in two p-cycles. A cache read is initiated in the first p-cycle, and a cache write with dirty bit set is initiated in the second p-cycle.

The data cache can be accessed for byte, half-word, three-byte, word, five-byte, six-byte, seven-byte, and double-word. The data size of a partial load is derived from the access type from the integer control unit and the lower three address bits. The data alignment is performed by the datapath load aligner.

To reduce the cache miss penalty, the address of the block read request will point to the location of the desired double-word. Since the data cache has a two double-word line size, the system interface will return the critical double-word first, followed by the remaining double-word. The return data will be written to the cache as it is put on the data bus to be used by the execution unit.

The R4300i processor provides a variety of cache operations for use in maintaining the state and contents of the instruction and data caches. During the execution of the cache operation instructions, the processor may issue processor block read or write request to fill or write-back a cache line.

Flush Buffer

The R4300i Microprocessor contains a 4-entry on-chip flush buffer. The buffer is used as temporary data storage for outgoing data and is organized as a 4 deep fifo; that is it can buffer 4 addresses along with 4 double-words of data. For uncached write operations, the flush buffer can accept
any combination of single or double-word data until it is full, with each write occupying one entry in the buffer. For data cache block write operations, the flush buffer accepts 2 double-words with 1 address, occupying two entries in the buffer. It is able to take two block references at a time. Instruction cache block writes use 4 doublewords with 1 address. Instruction cache block writes occupy the entire flush buffer. The flush buffer is able to take one read memory reference at a time.

Address is a 32-bit physical address, and size indicates the size of data to be transferred out.

During an uncached store, data will be stored in this buffer until it is taken by the external interface. While data awaits in this area, processor pipeline continues to execute.

During a load miss or a store miss to a cache line in the dirty state, a read request for the missing cache line is sent to the external interface. The dirty data is then stored in the flush buffer until the requested data is returned from the external interface. The processor pipeline continues to run while the flush buffer writes the dirty data to the external interface.

If the flush buffer is full and the processor attempts a load or a store which requires external resources, the processor pipeline will stall until the buffer is emptied. Figure 17 shows the layout of the flush buffer.

![Figure 17 Flush Buffer Format](image)

### Coprocessor 0

The R4300i Coprocessor-0 functional unit performs two major functions. The first is memory management for the processor, and the second is exception processing. CP0 is logically divided into four subunits: CP0 Register Files (CP0reg), Translation Look-aside Buffer (TLB), Instruction TLB (ITLB), and CP0 Control Unit (CP0ctl). Figure 18 shows how these subunits interact with each other and other functional units.

The CP0 registers consist of two functionally distinct sets. The first set comprises those that support the TLB operations, and the second set is formed from those registers that reflect the state of the processor. All of the CP0 registers are readable by software. All but the Random, BadVAddr, Prid, and MsKld registers are writable by software. The definitions of the CP0 registers are described in a later section.

The CP0ctl subunit controls data transfer to and from the TLB, ITLB and CP0 registers. It takes the instruction decode signals from the integer unit and interprets them to determine whether a CP0 register is to be read or written. It generates control signals for reading and writing TLB and ITLB entries.

![Figure 18 R4300i Coprocessor 0 Block Diagram](image)

In addition, the CP0ctl handles exception processing. Its function is to communicate with the pipeline control unit for interrupt signalling, and notification of TLB exceptions. It also performs address checking with regard to the type of instruction being executed, and various state bits in the Status register. For example, a User mode process may try to access a kernel data address. In this case, a Data Address Error exception would be detected, since the address Region bits would conflict with the KSU bits in the Status register.

The Translation Look-aside Buffers (micro-TLB and joint TLB) are responsible for the translation of both instruction and data virtual addresses into physical addresses. The physical address is needed by the cache for its tag checks, and by the system interface for off-chip memory access, and various CP0 registers. The jTLB translates the lower 40 bits of the 64-bit virtual address size defined in the MIPS-III architecture, and provides a 32-bit physical address.

The joint TLB contains 32 entries, each of which is simultaneously checked for a match with the extended virtual address. Each TLB entry maps an even-odd pair of pages. The page size is 4K, 16K, 64K, 256K, 1M, 4M or 16M bytes, which is specified on a per-entry basis by the MASK bit-mask field of the entry. The valid values of the MASK field and the effect on the translation is documented in the description of the PageMask register.

A virtual address matches a TLB entry when the virtual page number (VPN) field of the virtual address equals the VPN field of the TLB entry, and either the Global (G) bit of
the TLB entry is set or the address space identifier (ASID) field of the virtual address (as held in the \textit{EntryHi} register) matches the ASID field of the TLB entry. Although the valid (V) bit of the entry must be set for a valid translation to take place, it is not involved in the determination of a matching TLB entry.

The operation of the TLB is not defined if more than one entry in the TLB matches. If one TLB entry matches, the physical address and access control bits (N, D, and V) are retrieved; otherwise a TLB refill exception occurs. If the access control bits (D and V) indicate that the access is not valid, a TLB modification or TLB invalid exception occurs.

The format of a TLB entry in 32-bit addressing mode is shown in Figure 19.

![Figure 19 TLB Entry Format in 32-bit Addressing Mode](image)

The format of a TLB entry in 64-bit mode is shown in Figure 20.

![Figure 20 TLB Entry Format in 64-bit Addressing Mode](image)

The R4300i also implements a two entry micro-TLB that is dedicated for instruction address translation. This allows instruction address translation to be done simultaneously with data address translation. If there is a miss in the micro-TLB, the pipeline will stall while the new TLB entry is transferred from the joint TLB to the micro-TLB. The two entry micro-TLB entry is fully associative with a Least Recently Used replacement algorithm. Each micro-TLB entry maps to a 4K byte page size only.

**CP0 Registers**

Table 1 lists all CP0 registers used in the R4300i. Attempting to write any unused register is undefined and may have an unpredictable effect on the R4300i processor. Attempting to read any unused register is undefined and may result in unpredictable data.

<table>
<thead>
<tr>
<th>Num</th>
<th>Mnemonic</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Index</td>
<td>Programmable pointer into TLB array</td>
</tr>
<tr>
<td>1</td>
<td>Random</td>
<td>Random pointer into TLB array</td>
</tr>
<tr>
<td>2</td>
<td>EntryLo0</td>
<td>Low half of the TLB entry for even VPN</td>
</tr>
<tr>
<td>3</td>
<td>EntryLo1</td>
<td>Low half of the TLB entry for odd VPN</td>
</tr>
<tr>
<td>4</td>
<td>Context</td>
<td>Pointer to kernel PTE table</td>
</tr>
<tr>
<td>5</td>
<td>PageMask</td>
<td>TLB page mask</td>
</tr>
<tr>
<td>6</td>
<td>Wired</td>
<td>Number of wire TLB entries</td>
</tr>
<tr>
<td>7</td>
<td>----</td>
<td>Unused</td>
</tr>
<tr>
<td>8</td>
<td>BadVAddr</td>
<td>Bad virtual address</td>
</tr>
<tr>
<td>9</td>
<td>Count</td>
<td>Timer count</td>
</tr>
<tr>
<td>10</td>
<td>EntryHi</td>
<td>High half of TLB entry</td>
</tr>
<tr>
<td>11</td>
<td>Compare</td>
<td>Timer compare</td>
</tr>
<tr>
<td>12</td>
<td>SR</td>
<td>Status register</td>
</tr>
<tr>
<td>13</td>
<td>Cause</td>
<td>Cause register</td>
</tr>
<tr>
<td>14</td>
<td>EPC</td>
<td>Exception program counter</td>
</tr>
<tr>
<td>15</td>
<td>PRid</td>
<td>Processor revision identifier</td>
</tr>
<tr>
<td>16</td>
<td>Config</td>
<td>Configuration register</td>
</tr>
<tr>
<td>17</td>
<td>LAddr</td>
<td>Load linked address</td>
</tr>
<tr>
<td>18</td>
<td>WatchLo</td>
<td>Memory reference trap address lower bits</td>
</tr>
<tr>
<td>19</td>
<td>WatchHi</td>
<td>Memory reference trap address upper bits</td>
</tr>
<tr>
<td>20</td>
<td>XContext</td>
<td>Context register for MIPS-III addressing</td>
</tr>
<tr>
<td>21-25</td>
<td>----</td>
<td>Unused</td>
</tr>
<tr>
<td>26</td>
<td>PErr</td>
<td>Not Used</td>
</tr>
<tr>
<td>27</td>
<td>CacheErr</td>
<td>Not Used</td>
</tr>
<tr>
<td>28</td>
<td>TagLo</td>
<td>Cache tag register</td>
</tr>
<tr>
<td>29</td>
<td>TagHi</td>
<td>Cache tag register (Reserved)</td>
</tr>
<tr>
<td>30</td>
<td>ErrorEPC</td>
<td>Error exception program counter</td>
</tr>
<tr>
<td>31</td>
<td>----</td>
<td>Unused</td>
</tr>
</tbody>
</table>

The \textit{Index} register is a read/write register in which 6 bits specify an entry into the on-chip TLB. The high order bit indicates the success or failure of a TLBP operation. The TLB \textit{Index} register is used to specify the entry in the TLB affected by the TLBR and TLBWI instructions. Figure 21 shows the \textit{Index} register format.

![Figure 21 Index Register](image)
The **Random** register is a read-only register in which 6 bits specify an entry in the on-chip TLB. This register will decrement on every instruction executed. The values range between a low value determined by the TLB Wired register, and an upper bound of TLBENTRIES-1. The TLB **Random** register is used to specify the entry in the TLB affected by the TLBWR instruction. Upon system reset, or when the Wired register is written, this register will be set to the upper limit. Figure 22 shows the **Random** register format.

![Random Register format](image1)

**Figure 22** CP0 Random Register

The **EntryLo0** and **EntryLo1** registers are a read/write register pair that are used to access on-chip TLB. **EntryLo0** is used for the even virtual pages while **EntryLo1** is used for the odd virtual pages. They contain the Page Frame Number, along with several configuration bits for the TLB entry. They are used by the TLBR, TLBWl, and TLBWR instructions. Figure 23 shows the **EntryLo0** and **EntryLo1** registers format.

![EntryLo0 and EntryLo1 Registers format](image2)

**Figure 23** CP0 EntryLo0 and EntryLo1 Registers

The **Context** register is a read/write register containing a pointer into a kernel Page Table Entry (PTE) array. It is designed for use in the TLB refill handler. The BadVPN2 field is not writable. It contains the VPN of the most recently translated virtual address that did not have a valid translation. It contains bits 31:13 of the virtual address that caused the TLB miss. Bit 12 is excluded because a single TLB entry maps an even-odd page pair. This format can be used directly as an address for pages of size 4K bytes. For all other page sizes the value must be shifted and masked. The PTEBase field is writable as well as readable, and indicates the base address of the PTE table of the current user address space. Figure 24 shows the **Context** register format.

![Context Register format](image3)

**Figure 24** CP0 Context Register

The **PageMask** register is a read/write register that is used when reading or writing an on-chip TLB. The TLBR, TLBWl, and TLBWR instructions use this register as a source or destination. When virtual addresses are presented for translation, the corresponding bits in the TLB specify which of virtual address bits 24.13 participate in the comparison. This implements a variable page size on a per entry basis. R4300i implements 4K, 16K, 64K, 256K, 1M, 4M and 16M pages. Figure 25 shows the **PageMask** register format.

![PageMask Register format](image4)

**Figure 25** CP0 Mask Register

The **Wired** register is a read/write register that specifies the boundary between the wired and random entries of the TLB. This register is set to 0 upon reset. Writing to this register also sets the **Random** register to 31. Writing a value greater than TLBENTRIES-1 to this register will result in undefined behavior. Figure 26 shows the **Wired** register.

![Wired Register format](image5)

**Figure 26** CP0 Wired Register
The Bad Virtual Address register is a read-only register that displays the most recently translated virtual address which failed to have a valid translation or which had an addressing error. Figure 27 shows the BadVAddr register format.

The Count register is a read/write register used to implement timer services. It increments at a constant rate based on the clock cycle. On R4300ii, it will increment at one-half the PClock speed. When the Count register has reached all ones, it will roll over to all zeroes and continue counting. This register is both readable and writable. It is writable for diagnostic purposes. Figure 28 shows the Count register format.

The EntryHi register is a read/write register that is used to access on-chip TLB. It is used by the TLBR, TLBWI, and TLBWR instructions. EntryHi contains the Address Space Identifier (ASID) and the Virtual Page Number. Figure 29 shows the EntryHi register format.

The Compare register is a read/write register. When the value of the Count register equals the value of the Compare register, IP7 of the Cause register is set. This causes an interrupt on the next execution cycle in which interrupt is enabled. Writing to the Compare register will clear the timer interrupt. Figure 30 shows the Compare register format.

The Status register is a read/write register that contains the various mode, enables, and status bits used on R4300i. The contents of this register are undefined after a reset, except for TS, which is zero; ERL and BEV, which are one; and RP, which is zero. The SR bit is 0 after a Cold Reset, and 1 after NMI or (Soft) Reset. Figure 31 shows the Status register format.

**Figure 27** CP0 BadVAddr Register

**Figure 28** CP0 Count Register

**Figure 29** CP0 EntryHi Register

**Figure 30** CP0 Compare Registers

**Figure 31** CP0 Status Register
The *Cause* register is a read/write register that describes the nature of the last exception. A five-bit exception code indicates the cause of the exception and the remaining fields contain detailed information relevant to the handling of certain types of exceptions. The Branch Delay bit indicates whether the EPC has been adjusted to point at the branch instruction which precedes the next restartable instruction. The Coprocessor Error field indicates the unit number referenced by an instruction causing a “Coprocessor Unusable” exception. The Interrupt Pending field indicates which interrupts are pending. This field indicates the current status and changes in response to external signals. IP7 is the timer interrupt bit, set when the *Compare* register equals the *Count* register. IP6:2 are the external interrupts, set when the external interrupts are signalled. An external interrupt is set at one of the external interrupt pins or via a write request on the SysAD bus. IP1:0 are software interrupts, and may be written to set or clear software interrupts. Figure 32 shows the *Cause* register format.

### Figure 32 CP0 Cause Register

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>16</td>
<td>15</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>BD</td>
<td>CE</td>
<td>0</td>
<td>IP</td>
<td>0</td>
<td>ExcCode</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where:
- **BD** Branch Delay
- **CE** Coprocessor Error
- **IP** Interrupt pending

### ExcCode Exception Code:

- 0 Must be zeroes on read and write
- 1 Int Interrupt
- 2 TLB Mod TLB modification exception
- 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 (load or store)
- 7 DBE Bus Error Exception (data reference: load or store)
- 8 Sys SysCall Exception
- 9 Bp Breakpoint Exception
- 10 RI Reserved instruction Exception
- 11 CEPU Coprocessor Unusable Exception
- 12 Ov Arithmetic Overflow Exception
- 13 Tr Trap Exception
- 14 — Reserved
- 15 FPE Floating Point Exception
- 16-22 Reserved
- 23 Watch Reference to WatchHi/WatchLo address
- 24-31 Reserved

The *EPC* register is a read/write register that contains the address at which instruction processing may resume after servicing an exception. For synchronous exceptions, the *EPC* register contains either the virtual address of the instruction which was the direct cause of the exception, or when that instruction is in a branch delay slot, the *EPC* contains the virtual address of the immediately preceding branch or jump instruction and the Branch Delay bit in the *Cause* register is set. If the exception is caused by recoverable, temporary conditions (such as a TLB miss), the *EPC* contains the virtual address of the instruction which caused the exception. Thus, after correcting the conditions, the *EPC* contains a point at which execution can be legitimately resumed. Figure 33 shows the format of *EPC* register.

### Figure 33 CP0 EPC Register

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32 bit Mode:</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>64 bit Mode:</td>
<td>63</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64 bit Mode:</td>
<td>63</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where:
- **EPC** Exception Program Counter

The *PRId* register is a read-only register that contains information that identifies the implementation and revision level of the processor and associated system control coprocessor. The revision number can distinguish some chip revisions. However, MIPS is free to change this register at any time and does not guarantee that changes to its chips will necessarily change the revision number, or that changes to the revision number necessarily reflect real chip changes. For this reason, software should not rely on the revision number to characterize the chip. Figure 34 shows the *Processor Revision Identifier* register format.

### Figure 34 CP0 Revision Identifier Register

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>16</td>
<td>15</td>
<td>8</td>
<td>7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32 bit Mode:</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>64 bit Mode:</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where:
- **Imp** Implementation identifier: 0x0B for R4300i
- **Rev** Revision Number
- 0 Returns zeroes on reads.

The *Config* register specifies various configuration options that are available for R4300i. It is compatible with the R4000 *Config* register, but only a small subset of the options available on the R4000 are possible on R4300i. For that reason, there are many fields which are set to constant values. The EP and BE fields are writable by software only. The CU and K0 fields are readable and writable by software. There is no other mechanism for writing to these fields, and their values are undefined after Reset. Figure 35 shows the *Configuration* register format.
The R4300ii processor provides a debugging feature to detect references to a physical address. Loads or stores to the location specified by the WatchHi/WatchLo register pair cause a Watch trap. Figure 36 shows WatchHi/WatchLo register formats.

The LLAddr register contains the physical address read by the most recent Load Linked instruction. This register exists for diagnostic purposes, and serves no function during normal operation. It is both readable and writable by software. Figure 37 shows the LLAddr register format.

The XContext register is a read/write register containing a pointer into a kernel Page Table Entry (PTE) array. It is designed for use in the XTLB refill handler. The R and BadVPN2 fields are not writable. The register contains the VPN of the most recently translated virtual address that did not have a valid translation. It contains bits 39.13 of the virtual address that caused the TLB miss. Bit 12 is excluded because a single TLB entry maps an even-odd page pair. This format can be used directly as an address for pages of size 4K bytes. For all other page sizes this value must be shifted and masked. The PTEBase field is writable as well as readable, and indicates the base address of the PTE table of the current user address space. Figure 38 shows XContext register format.

The PErr register, shown in Figure 39, is included only to maintain compatibility with the R4200. The R4300i does not implement parity, hence this register is not used by hardware. However, the register is software readable and writable.

---

**Figure 35** CP0 Configuration Register

**Figure 36** CP0 WatchLo and WatchHi Registers

**Figure 37** CP0 LLAddr Register

**Figure 38** CP0 XContext Register

**Figure 39** CP0 PErr Register
The CacheErr register, shown in Figure 40, is included only to maintain compatibility with the R4200. The R4300i does not implement cache errors, hence this register is not used by hardware. It is a read only register which returns a zero value when read.

![Figure 40 CP0 CacheErr Register](image)

The TagLo register is a 32-bit read/write register used to hold the cache tag during cache initialization and diagnostics. The TagHi register is reserved for future use. These two Tag registers are written by the CACHE and MTC0 instructions. Figure 41 shows TagLo and TagHi register formats.

![Figure 41 CP0 TagLo TagHi Registers](image)

The ErrorEPC register is similar to the EPC, but is used to store the PC on Reset and NMI exceptions. It is read/write register and contains the virtual address at which instruction processing may resume after servicing an Error or Reset/NMI exceptions.

The EPC register contains either the virtual address of the instruction which was the direct cause of the exception, or when that instruction is in a branch delay slot, the EPC contains the virtual address of the immediately preceding branch or jump instruction. There is no branch delay slot indication for ErrorEPC. Figure 42 shows ErrorEPC register format.

![Figure 42 CP0 ErrorEPC Register](image)

Memory Management

The R4300i processor provides a full-featured memory management unit (MMU) which uses an on-chip translation look-aside buffer (TLB) to translate virtual address into physical address. The TLB is a fully associative memory that holds 32 entries, which provide mapping to 32 odd/even pairs (64 pages). When address mapping is indicated, each TLB entry is checked simultaneously for a match with the virtual address that is extended with an ASID stored in the EntryHi register. The address is mapped to a page of size of between 4Kbytes and 16Mbytes.

The processor virtual address can be either 32 or 64 bits wide, depending on whether the processor is operating in 32-bit or 64-bit mode.

- In 32-bit mode, addresses are 32 bits wide. The maximum user process size is 2 gigabytes ($2^{31}$).
- In 64-bit mode, addresses are 64 bits wide. The maximum user process size is 1 terabyte ($2^{40}$).

Figure 43 shows the translation of a virtual address into a physical address.

![Figure 43 Virtual-to-Physical Address Translation](image)
Converting a virtual address to a physical address begins by comparing the virtual address from the processor with the virtual address in the TLB; there is a match when the virtual page number (VPN) of the address is the same as the VPN field of the entry, and either:

- the Global (G) bit of the TLB entry is set, or
- the ASID field of the virtual address is the same as the ASID field of the TLB entry.

This match is referred to as a **TLB hit**. If there is no match, a TLB Miss exception is taken by the processor and software is allowed to refill the TLB from a page table of virtual/physical addresses in memory.

If there is a virtual address match in the TLB, the physical address is output from the TLB and concatenated with the Offset, which represents an address within the page frame space.

Figure 44 shows the virtual-to-physical address translation of a 32-bit mode address.

---

**Figure 44** 32-Bit-Mode Virtual Address Translation

- The top portion of Figure 44 shows a virtual address with a 12-bit, or 4-Kbyte, page size, labelled Offset. The remaining 20 bits of the address represent the VPN, and index the 1M-entry page table.
- The bottom portion of Figure 44 shows a virtual address with a 24-bit, or 16-Mbyte, page size, labelled Offset. The remaining 8 bits of the address represent the VPN, and index the 256-entry page table.

---

Figure 45 shows the virtual-to-physical-address translation of a 64-bit mode address. This figure illustrates the two possible page sizes: a 4-Kbyte page (12 bits) and a 16-Mbyte page (24 bits).

- The top portion of Figure 45 shows a virtual address with a 12-bit, or 4-Kbyte, page size, labelled Offset. The remaining 28 bits of the address represent the VPN, and index the 256M-entry page table.
- The bottom portion of Figure 45 shows a virtual address with a 24-bit, or 16-Mbyte, page size, labelled Offset. The remaining 16 bits of the address represent the VPN, and index the 64K-entry page table.

---

**Figure 45** 64-Bit-Mode Virtual Address Translation

The processor has three operating modes that function in both 32- and 64-bit operations: User mode, Supervisor mode, and Kernel mode. The address space for each mode of operation are described in the following sections.

In User mode, a single, uniform virtual address space—labelled User segment—is available; its size is:

- 2 Gbytes (2³¹ bytes) in 32-bit mode (*useg*)
- 1 Tbyte (2⁴⁰ bytes) in 64-bit mode (*xuseg*)

The User segment starts at address 0 and the current active user process resides in either *useg* (in 32-bit mode) or *xuseg* (in 64-bit mode). The TLB identically maps all references to *useg/xuseg* from all modes, and controls cache accessibility*. Figure 46 shows User mode virtual address space.

---

* The cached (C) field in a TLB entry determines whether the reference is cached; see Figure 20.
The processor operates in User mode when the Status register contains the following bit-values:

- KSU bits = 102
- EXL = 0
- ERL = 0

In conjunction with these bits, the UX bit in the Status register selects between 32- or 64-bit User mode addressing as follows:

- when UX = 0, 32-bit useg space is selected
- when UX = 1, 64-bit xuseg space is selected

Supervisor mode is designed for layered operating systems in which a true kernel runs in R4300i Kernel mode, and the rest of the operating system runs in Supervisor mode. Figure 47 shows Supervisor mode address space.

The processor operates in Supervisor mode when the Status register contains the following bit-values:

- KSU = 012
- EXL = 0
- ERL = 0

In conjunction with these bits, the SX bit in the Status register selects between 32- or 64-bit Supervisor mode addressing:

- when SX = 0, 32-bit supervisor space is selected
- when SX = 1, 64-bit supervisor space is selected

The processor operates in Kernel mode when the Status register contains one of the following values:

- KSU = 002
- EXL = 1
- ERL = 1

In conjunction with these bits, the KX bit in the Status register selects between 32- or 64-bit Kernel mode addressing:

- when KX = 0, 32-bit kernel space is selected
- when KX = 1, 64-bit kernel space is selected

The processor enters Kernel mode whenever an exception is detected and it remains in Kernel mode until an Exception Return (ERET) instruction is executed. The ERET instruction restores the processor to the mode existing prior to the exception.

Kernel mode virtual address space is divided into regions differentiated by the high-order bits of the virtual address, as shown in Figure 48.
Exception Management

The R4300i processor receives exceptions from a number of sources, including translation lookaside buffer (TLB) misses, arithmetic overflows, I/O interrupts, and system calls. When the CPU detects one of these exceptions, the normal sequence of instruction execution is suspended and the processor enters Kernel mode.

The processor then disables interrupts and forces execution of a software exception processor (called a handler) located at a fixed address. The handler saves the context of the processor, including the contents of the program counter, the current operating mode (User or Supervisor), and the status of the interrupts (enabled or disabled). This context is saved so it can be restored when the exception has been serviced.

When an exception occurs, the CPU loads the Exception Program Counter (EPC) register with a location where execution can restart after the exception has been serviced. The restart location in the EPC register is the address of the instruction that caused the exception or, if the instruction was executing in a branch delay slot, the address of the branch instruction immediately preceding the delay slot.

The R4300i separates exceptions into four vector spaces: Reset and NMI vector, TLB Refill vector, XTLB Refill vector, and General exception vector. The values of each vector (except Reset and NMI) depend on the Boot Exception Vector (BEV) bit of the Status register, which allows two alternate sets of vectors to be used, with one set pointing to the PROM address space and the other set pointing to cacheable address space.

The Reset and NMI exceptions are always vectored to location 0xBFC0 0000 in 32-bit mode, and location 0xFFFF FFFF BFC0 0000 in 64-bit mode.

The addresses for all other exceptions are a combination of a vector offset and a base address. The base address is determined by the BEV bit of the Status register. Table 2 shows the exception R4300i exception vectors.

### Table 2 Exception Vectors

<table>
<thead>
<tr>
<th>Exception Type</th>
<th>Vector Offset</th>
<th>BEV</th>
<th>R4300i Processor Exception Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>32-bit mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>64-bit mode</td>
</tr>
<tr>
<td>Reset, Soft</td>
<td>0x000</td>
<td>-</td>
<td>0xBFC0 0000</td>
</tr>
<tr>
<td>Reset, NMI</td>
<td></td>
<td></td>
<td>0xFFFF FFFF BFC0 0000</td>
</tr>
<tr>
<td>TLB Refill</td>
<td>0x000</td>
<td>0</td>
<td>0xBFC0 0200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0xFFFF FFFF BFC0 0200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0xBFC0 0080</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0xFFFF FFFF BFC0 0080</td>
</tr>
<tr>
<td>XTLB Refill</td>
<td>0x080</td>
<td>0</td>
<td>0xBFC0 0280</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0xFFFF FFFF BFC0 0280</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0xBFC0 00180</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0xFFFF FFFF BFC0 00180</td>
</tr>
<tr>
<td>All Other</td>
<td>0x180</td>
<td>0</td>
<td>0xBFC0 0380</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0xFFFF FFFF BFC0 0380</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

While more than one exception can occur for a single instruction, only the exception with the highest priority is reported. Table 3 lists all exceptions in the order of their priority.

### Table 3 Exception Priority Order

- Reset (highest priority)
- Soft Reset
- NMI
- Address Error -- Instruction fetch
- TLB Refill -- Instruction fetch
- TLB Invalid -- Instruction fetch
- Bus Error -- Instruction fetch
- System Call
- Breakpoint
- Coprocessor Unusable
- Reserved Instruction
- Trap Instruction
- Integer Overflow
- Floating-point Exception
- Address Error -- Data access
- TLB Refill -- Data access
- TLB Invalid -- Data access
- TLB Modified -- Data write
- Watch
- Bus Error -- Data access
- Interrupt

Exceptions are logically precise. The instruction that causes an exception and all those that follow it are aborted, generally before committing any state, and can be re-executed after servicing the exception. When following instructions are aborted, exceptions associated with those instructions are also aborted. Therefore, exceptions are not taken in the order detected, but rather in instruction fetch order.

The exception handling system is responsible for the efficient handling of relatively infrequent events, such as translation misses, arithmetic overflow, I/O interrupts, and system calls. These events cause the interruption of the normal flow of execution; aborting instructions which cause exceptional conditions and all those which follow and have already begun executing, and a direct jump into a designated handler routine.

The architecture defines a minimal amount of additional state which is saved in coprocessor registers in order to facilitate the analysis of the cause of the exception, the servicing of the event which caused it, and the resumption of the original flow of execution, when applicable.
Interface Signals

The processor interface signals allow the processor to access external resources needed to satisfy cache misses and uncached operations, while permitting an external agent access to some of the processor internal resources. The signals include the System interface, the Clock/Control interface, the Interrupt interface, the Joint Test Action Group (JTAG) interface, and the Initialization interface. Figure 49 illustrates the functional groupings of the processor signals.

The clock/control interface signals make up the interface for clocking and clock synchronization. Table 5 lists the Clock/Control interface signals.

Table 5 Clock/Control Interface Signals

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MasterClock</td>
<td>Input</td>
<td>Master clock input that establishes the processor operating frequency.</td>
</tr>
<tr>
<td>TClock</td>
<td>Output</td>
<td>Transmit clocks that establish the System interface frequency. Tclock is aligned with SyncIn at the MasterClock frequency.</td>
</tr>
<tr>
<td>SyncOut</td>
<td>Output</td>
<td>Synchronization clock output. Must be connected to SyncIn through an inter-connect that models the interconnect between Tclock and the external agent aligned with MasterClock.</td>
</tr>
<tr>
<td>DivMode 1:0</td>
<td>Input</td>
<td>These signals determine the ratio between the MasterClock and the internal processor PClock. DivMode 1:0 are encoded as follows: 00 = 1:1 MasterClock to PClock ratio. 01 = 1.5:1 MasterClock to PClock ratio. 10 = 2:1 MasterClock to PClock ratio. 11 = 3:1 MasterClock to PClock ratio.</td>
</tr>
</tbody>
</table>

The initialization interface signals make up the interface by which an external agent initializes the processor operating parameters. Table 6 lists the initialization interface signals.

Table 6 Initialization Interface Signals

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset*</td>
<td>Input</td>
<td>Used to initiate a soft reset sequence.</td>
</tr>
<tr>
<td>ColdReset*</td>
<td>Input</td>
<td>When asserted, this signal indicates to the R4300i processor that the +3.3 volt power supply is stable and the R4300i chip should initiate a cold reset sequence. The assertion of ColdReset* will reset the PLL.</td>
</tr>
<tr>
<td>PLLCAP0</td>
<td>Input</td>
<td>A capacitor is connected between PLLCAP0 and the clock VssP to insure proper operation of the PLL.</td>
</tr>
<tr>
<td>PLLCAP1</td>
<td>Input</td>
<td>A capacitor is connected between PLLCAP1 and the clock VccP to insure proper operation of the PLL.</td>
</tr>
<tr>
<td>TestMode*</td>
<td>Input</td>
<td>Used for cache testing. This signal must be connected to Vcc during normal operation. This pin will be part of the JTAG scan chain.</td>
</tr>
</tbody>
</table>

The interrupt/status interface signals make up the interface used by the external agent to interrupt the R4300i processor and to monitor instruction execution for the current processor cycle. Table 7 lists the interrupt/status interface signals.

Table 7 Interrupt/Status Interface Signals

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int*(4:0)</td>
<td>Input</td>
<td>Five general processor interrupts. These are visible as bits 14 to 10 of the Cause register.</td>
</tr>
<tr>
<td>NMI*</td>
<td>Input</td>
<td>Non-maskable interrupt.</td>
</tr>
</tbody>
</table>
The JTAG interface signals make up the interface that provides the JTAG boundary scan mechanism. Table 8 lists the JTAG interface signals.

Table 8 JTAG Interface Signals

<table>
<thead>
<tr>
<th>Name</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTDI</td>
<td>Input</td>
<td>Data is serially scanned in through this pin.</td>
</tr>
<tr>
<td>JTCK</td>
<td>Input</td>
<td>The processor receives a serial clock on JTCK. On the rising edge of JTCK, both JTDI and JTMS are sampled.</td>
</tr>
<tr>
<td>JTDI</td>
<td>Output</td>
<td>Data is serially scanned out through this pin.</td>
</tr>
<tr>
<td>JTMS</td>
<td>Input</td>
<td>JTAG command signal, indicating the incoming serial data is command data.</td>
</tr>
</tbody>
</table>

Figure 50 shows the primary communication paths for the System interface: a 32-bit address and data bus, SysAD(31:0), and a 5-bit command bus, SysCmd(4:0). These buses are bidirectional; that is, they are driven by the processor to issue a processor request, and by the external agent to issue an external request.

A request through the System interface consists of: an address, a system interface command that specifies the precise nature of the request, and a series of data elements if the request is for a write, or read response.

Cycles in which the SysAD bus contains a valid address are called address cycles. Cycles in which the SysAD bus contains valid data are called data cycles. The most significant bit of the SysCmd bus is used to indicate whether the current cycle is an address cycle or a data cycle. During address cycles, the remainder of SysCmd contains a system interface command. During data cycles, the remainder of SysCmd contains the data identifier.

The processor will repeat the address cycle until the external agent indicates that it is capable of accepting the request. The last address cycle is called the issue cycle. There are two types of issue cycle: processor read request issue cycles, and processor write request issue cycles.

When the R4300i processor is driving the SysAD and SysCmd buses, the System interface is in master state. When the external agent is driving the SysAD and SysCmd buses, the System interface is in slave state. The processor is the default master of the system interface.

The external agent becomes master of the system interface only through arbitration protocol or uncompelled change to slave state. An uncompelled change to slave state is initiated by the processor. There are two broad categories of requests: processor requests and external requests. Processor requests include: read requests, and write requests. External requests include: read responses and write requests.

A processor request is a request through the System interface, to access some external resource. The following rules apply to processor requests.

- After issuing a processor read request, the processor cannot issue a subsequent read request until it has received a read response.
- It is possible that back-to-back write requests can be issued by the R4300i with no idle cycles on the bus between requests.

A Processor Read Request is issued by driving a read command on the SysCmd bus, driving a read address on the SysAD bus, and asserting PValid*. Only one processor read request may be pending at a time. The processor must wait for an external read response before starting a subsequent read. The processor transitions to slave after the issue cycle of the read request by de-asserting the PMaster* signal. An external agent may then return the requested data via a read response. The external agent, which has become master, may issue any number of writes before sending the read response data. An example of a processor read request and an uncompelled change to slave state occurring as the read request is issued is illustrated in Figure 51.

A Processor Write Request is issued by driving a write command on the SysCmd bus, driving a write address on the SysAD bus, and asserting PValid* for one cycle, followed by driving the appropriate number of data identifiers on the SysCmd bus, driving data on the SysAD bus, and asserting PValid*. For 1- to 4-byte writes, a single data cycle is required. Byte writes of size 5, 6, & 7 are broken up into 2 address/data transactions; one 4 bytes in size, the other 1, 2, or 3 bytes. For all sizes greater than 7 bytes (e.g. 8, 16, 32), 4 bytes will be sent on each data cycle until the appropriate number of bytes have been transferred. When the last piece of data is being transferred, this final data cycle will be tagged as “Last Data” on the command bus.

To be fully compliant with all implementations of this protocol, an external agent should be able to receive write data over any number of cycles with any number of idle...
cycles between any two data cycles. However, for this implementation (i.e. R4300i) the data will begin on the cycle immediately following the write issue cycle, and transfers data at a programmed cycle data rate thereafter. The processor drives data at the rate specified by the data rate configuration signals.

Writes may be cancelled and retried with the EOK signal.

The example in Figure 52 illustrates the bus transactions for a two word data cache block store.

External Write Requests are similar to a processor single write except that the signal EValid* is asserted instead of the signal PValid*. An external write request consists of an external agent driving a write command on the SysCmd bus and a write address on the SysAD bus and asserting EValid* for one cycle, followed by driving a data identifier on the SysCmd bus and data on the SysAD bus and asserting EValid* for one cycle. The data identifier associated with the data cycle must contain a last data cycle indication. Note that the external agent must gain and maintain bus mastership during these transactions.

An external write request example is illustrated in Figure 53.

During An External Read Response the external agent returns data to the processor in response to a processor read request by waiting for the processor to transition to slave, and then returning the data via a single data cycle or a series of data cycles sufficient to transmit the requested data. After the last data cycle is issued the read response is complete and the processor will become the master (assuming EReq* was not asserted). If at the end of the read response cycles, EReq* has been asserted, the processor will remain slave until the external agent relinquished the bus. When the processor is in slave mode and needs access to the SysAD bus, it will assert PReq* and wait until EReq* is de-asserted.

The data identifier associated with a data cycle may indicate that the data transmitted during that cycle is erroneous; however, an external agent must return a block of data of the correct size regardless of erroneous data cycles. If a read response includes one or more erroneous data cycles, the processor will take a bus error.

Read response data must only be delivered to the processor when a processor read request is pending. The behavior of the processor if a read response is presented to it when there is no processor read pending is undefined. A processor single read request followed by a read response is illustrated in Figure 54.

Clocks

The clocks on the R4300i processor are controlled via an on-chip Phase Locked Loop circuit. This circuit keeps the internal clock edges aligned with the clock edges of the MasterClock signal, which acts as the system master clock.

Inside the R4300i processor, the MasterClock signal is multiplied by a factor determined by DivMode 1:0, and then all internal clocks are derived by dividing that signal down. There are two primary internal clocks, the pipeline clock PClock, and the system interface clock SClock. While other internal edges may be generated, these are transparent to the user.

The pipeline clock PClock is a multiple of the MasterClock frequency as determined by DivMode 1:0.

The system interface clock, SClock, is equal to the MasterClock frequency.

TClock is generated by the processor at the same frequency as SClock. It is aligned with SClock. It is used by external agents to drive data, and as the global clock for the external agent. TClock can be thought of as the synchronized external system interface clock.
Mode Selectable Features

The R4300i processor will support several user selectable modes, most of which can be set and reset by writing to the CP0 Status register.

The Reduced Power (RP) mode allows the user to change the processor operating frequency to quarter speed. This is established by setting bit 27 of the Status register.

This feature is included to allow the user to selectively reduce power when the system is not being heavily used. This feature will reduce the power consumed by the processor chip to 25% of its normal value. The default of this mode is normal clocking. The chip will return to this state after any reset.

The Floating-point Register (FR) mode (bit 26 of the Status register) enables the user to access the full set of 32 64-bit floating point registers as defined in MIPS-III. When reset, the processor will access the registers as defined in the MIPS II architecture. This functionality is the same as the R4000.

The Reverse Endianness (RE) mode (bit 25 of the Status register) allows the user to switch byte ordering, between BigEndian and LittleEndian, as seen by user software.

The Instruction Trace Support (ITS) mode allows the user to track branches or jumps. This mode is set by writing bit 24 of the Status register. When the ITS bit is set, the physical address to which the CPU has branched will be reported on the SysAD bus by forcing an instruction cache miss whenever a branch, jump or exception is taken.

The Bootstrap Exception Vectors (BEV), bit 22 in the Status register, when set, causes the TLB refill exception vector to be relocated to a virtual address of 0xbfc00200 and the general exception vector to 0xbfc00380. When cleared, these vectors are normally located at 0x80000000 (TLB refill), 0x00000080 (XTLB refill), and 0x80000180 (general), respectively. This bit is used when diagnostic tests cause exceptions to occur prior to verifying proper operation of the cache and main memory system.

The Kernel Extended Addressing (KX), bit 7 of the Status register, when set will enable the extended addressing TLB refill exception vector to be used for TLB misses on kernel addresses.

The Supervisor Extended Addressing (SX), bit 6 of the Status register, if set, enables MIPS III opcodes in supervisor-mode and causes TLB misses on supervisor addresses to use the Extended TLB refill exception vector.

The User Extended Addressing (UX), bit 5 of the Status register, if set, enables MIPS III opcodes in user-mode and causes TLB misses on user addresses to use the Extended TLB refill exception vector. If clear, implements MIPS II compatibility on virtual address translation.

The Interrupt Enable (IE), bit 0 of the Status register, when clear, will not allow interrupts with the exception of reset and non-maskable interrupt.

LOW POWER DISSIPATION

The R4300i is carefully designed to restrict power dissipation to make the processor suitable for operation in battery-powered devices. Power dissipation in the R4300i is controlled in two ways:

- Power management: The processor may be switched into reduced-power mode or power-down mode by external system logic.
- Reduced Power: A wide variety of logic and circuit design techniques is used to reduce power consumed in normal operating mode.

Power Management

The R4300i may be dynamically switched to one quarter of its normal operating frequency by the setting Reduce Power (RP) mode bit in the Status register. This will slow the pipeline and processor clocks to a frequency equivalent to one-fourth the MasterClock frequency. This event is typically initiated by external logic which sets the RP bit in the Status register via software. On setting this bit, the output of the clock dividers changes to provide the slower clock rate. Processing continues at one quarter the rate as before. There is a corresponding drop in power dissipated to one quarter of that previously.

The Power-down mode is typically initiated by external logic. All dirty cache lines are written back to main memory. All variable registers are readable and writable by software. This allows them to be saved and written to non-volatile memory while the processor is powered down. When power is restored, the processor state may be written back. State information is written using external software routines.

Reduced Power

A variety of circuit and logic design techniques is used to reduce power dissipation in the R4300i. Many of these techniques save only a small amount of power but it is the cumulative effect of these that produces a significant result.

These techniques are:
- 3.3V Power supply
- dynamic (rather than static) logic design
- unified datapath for less capacitance
- turning off unused portions of the chip
- minimized feature set
- 5-stage pipeline for high efficiency
- write-back data cache
- double instruction fetch from I-cache
- use of 2-entry micro I-TLB
- unused logic operates on constants
- gating clocks to pipeline logic to preserve data in a pipeline stall
- Speed de-optimization (no duplicate critical paths)
- Slew rate control and optimal fanout ratios for minimized switching current
- Minimized Vcc to Vss current path
These all combine to produce the low average power dissipation that allows the R4300i to be a practical battery-powered processor.

Differences Between The R4300i And R4200

The primary differences between R4300i and the R4200 are the system interface bus architecture and the absence of parity. The low-cost package requirements of the R4300i necessitated a new bus definition, which is very similar to but not the same as the R4200 system interface bus.

The functional differences between the R4200 and the R4300i processor’s that are visible to software are contained in the Coprocessor 0 implementation.

Table 9 gives a quick reference of the differences between the R4300i and R4200 Microprocessors. A more detailed description of these differences follows.

<table>
<thead>
<tr>
<th>Function</th>
<th>R4200</th>
<th>R4300i</th>
<th>Notes re: R4300i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parity Support</td>
<td>Yes</td>
<td>No</td>
<td>Not Implemented</td>
</tr>
<tr>
<td>CacheErr Register</td>
<td>Yes</td>
<td>No</td>
<td>Not Implemented</td>
</tr>
<tr>
<td>PErr Register</td>
<td>Yes</td>
<td>No</td>
<td>Diagnostic Use Only</td>
</tr>
<tr>
<td>Config Register BE/EP Fields</td>
<td>Hardware Controlled</td>
<td>Software Controlled</td>
<td>Set to default values during ColdReset*</td>
</tr>
<tr>
<td>Config Register bit 12</td>
<td>Reserved</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>Config Register bits [19:18]</td>
<td>00</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>Fast Data Rate</td>
<td>DDx</td>
<td>D</td>
<td>Software programmable</td>
</tr>
<tr>
<td>MasterOut Signal</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>RClock Signal</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Clock Multiplication</td>
<td>Fixed</td>
<td>Programmable</td>
<td>DivMode[1:0]</td>
</tr>
<tr>
<td>Vcc/Vss Grouping</td>
<td>3</td>
<td>2</td>
<td>I/O and Core the same</td>
</tr>
<tr>
<td>Packaging</td>
<td>208 pin PQFP</td>
<td>120 pin PQFP</td>
<td></td>
</tr>
<tr>
<td>Physical Address</td>
<td>33 bits</td>
<td>32 bits</td>
<td></td>
</tr>
<tr>
<td>Flush Buffers</td>
<td>1</td>
<td>4</td>
<td>64-bits each</td>
</tr>
</tbody>
</table>

The following describes Table 9 in more detail.

The R4300i microprocessor does not provide parity protection for the caches. The R4300i does not support parity and will never take parity error exception.

In the R4300i the CacheErr register (register 27) in Coprocessor0 is not implemented, hence any accesses to this register are undefined. In addition, the PErr register (register 26) in Coprocessor0 is used for diagnostic purposes only. The CE and DE bits in the Status register which are associated with parity handling are unused and hardwired to 0 and 1 respectively.
### R4300i Pin-Out

|   |   | Vcc | 31 | Vss | 32 | Vcc | 33 | SysAD16 | 34 | SysAD15 | 35 | Vcc | 36 | Vss | 37 | SysAD14 | 38 | Vcc | 39 | VccP | 40 | Vss | 41 | PLLCAP0 | 42 | PLLCAP1 | 43 | VccP | 44 | VssP | 45 | Vcc | 46 | MasterClock | 47 | Vss | 48 | TClock | 49 | Vcc | 50 | Vss | 51 | SyncOut | 52 | SysAD19 | 53 | Vcc | 54 | SyncIn | 55 | Vss | 56 | SysAD18 | 57 | SysAD17 | 58 | Int4* | 59 | Vcc | 60 | Vss | 61 | Vss | 62 | Vcc | 63 | JTDI | 64 | SysAD4 | 65 | JTDI | 66 | SysAD3 | 67 | Vcc | 68 | SysAD26 | 69 | PMaster* | 70 | SysAD1 | 71 | Vss | 72 | Vcc | 73 | SysAD0 | 74 | PReq* | 75 | SysAD10 | 76 | Vcc | 77 | SysAD9 | 78 | PValid* | 79 | Vss | 80 | SysAD8 | 81 | SysAD31 | 82 | EOK* | 83 | SysAD29 | 84 | Vcc | 85 | SysAD6 | 86 | SysAD28 | 87 | JTCK | 88 | Int2* | 89 | Vss | 90 | Vcc | 91 | Vcc | 92 | Vss | 93 | NMI | 94 | SysAD25 | 95 | EReq* | 96 | Vcc | 97 | Vss | 98 | SysAD25 | 99 | EReq* | 100 | SysCmd0 | 101 | Vcc | 102 | Vss | 103 | SysCmd1 | 104 | Reset* | 105 | SysCmd3 | 106 | SysCmd2 | 107 | Vcc | 108 | Vss | 109 | SysCmd3 | 110 | ColdReset* | 111 | SysCmd4 | 112 | DivMode1 | 113 | Vcc | 114 | Vss | 115 | SysAD24 | 116 | DivMode0 | 117 | SysAD23 | 118 | Int3* | 119 | Vcc | 120 | Vss |