R4300   RISC Processor Specification

Revision 2.2

July, 1995

MIPS Technologies Inc.
2011 N. Shoreline Blvd.
Mountain View, Ca. 94039-7311
http://www.mips.com

MTI CONFIDENTIAL

This document contains information that is proprietary to MIPS Technologies Inc. and is authorized only to employees of MIPS Technologies Inc. and to those persons specifically designated by MIPS Technologies Inc.

The information in this document is preliminary and is subject to change without notice.

MIPS Technologies Inc. reserves the right to change any products described herein to improve function or design. MIPS does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under patent rights nor imply the rights of others.

mips
MIPS Technologies Inc.
A subsidiary of Silicon Graphics Inc.

Table Of Contents

List of Tables

List of Figures

1.0 Introduction

This document contains the specification of the R4300 processor chip. The description of the internals of R4300 is not intended to reflect the details of R4300 implementation, but rather to give a general idea about the main features of the chip microarchitecture. The description of the R4300 external system interface is detailed enough to enable system designers to design systems based on the R4300 processor.

R4300 is a low cost, low power processor that is compatible with the MIPS-I, MIPS-II and MIPS-III Instruction Set Architecture (ISA) as defined in the “MIPS R-Series Architecture” specification. The terms R4300 and “the processor” are used interchangeably throughout this document.

The chip does not support external secondary caches, nor multiprocessing. The floating point operations are fully supported by an on-chip FPU implementation.

1.1 Reference Documents

MIPS R-Series Architecture Specification.
IEEE-754/1985 IEEE Standard for Binary Floating-Point Arithmetic
IEEE Std.1149.1/D6; IEEE JTAG Document
IEEE Std.8 sec.2 Interface Standard for Low Voltage CMOS (LVCMOS)
MIPS R4000 Microprocessor User’s Manual (MIPS part number M8-00040)
(Appendix A notes some of the differences between R4300 and the R4000)

1.2 Data Formats and Addressing

R4300 supports all of the data formats defined by the architecture specification (byte, halfword, word, and double word). Unaligned accesses are supported through the explicit instructions, such as LWR, LWL, LDR, LDL, and etc. The byte ordering is configurable into either big-endian or little-endian alignment via the Configuration register (BE) bit. Additionally, reverse endian (RE) is supported for user code. The machine uses byte addressing with alignment constraints for halfword, word, and doubleword accesses.

1.3 Registers

There are 32 general registers. Each register is a doubleword (64 bits). These registers are essentially equivalent except for r0, which is hardwired to a zero value, and r31, which is used as an implicit link register for jump and link instructions.

The R4300 processor supports the special registers Program Counter (PC), Multiply/Divide higher word register (HI), and Multiply/Divide lower word register (LO).

The floating point coprocessor contains 32 floating-point registers which are 64 bits wide. These registers support the single and double precision operand format. The floating point registers may be configured for MIPS II or MIPS III compatibility.

The on-chip system control coprocessor (CP0) uses 25 registers. These registers are 32 bits wide except for EntryHi, XContext, EPC, and ErrorPC, which are 64 bits wide.

1.4 Spec Objectives

Table 1: Summary of Spec Objectives

ISA compatibility	MIPS-I, MIPS-II and MIPS-III
Clock frequency	10Mhz min / 62.5MHz max
Pipeline clock	10Mhz min / 100MHz max
System interface clock	62.5MHz
Supply voltage (internal and I/O)	min 3.0V typ 3.3V max 3.6V (for 100MHz operation)
Power dissipation	typ 1.8W for 100MHz @3.3V, max operating freq
Die area	43.0 mm^2 using 0.35um CMOS technology with 3 metal and 2 poly layers.
Package	120 pin p-QFP
Junction temperature	min 0 degC max 125 degC
Caches	On-chip, direct mapped, 16kBi, 8kBd

TLB	32 double entries
	page size: 4kB to 16MB, in 4x increment
	VSIZE=40
	PSIZE=32

System interface	32-bit Address/Data bus
Performance (est.)	60 SPECint 45 SPECfp (assumes a system with memory latency of 175ns and D fill pattern.)

2.0 Overview

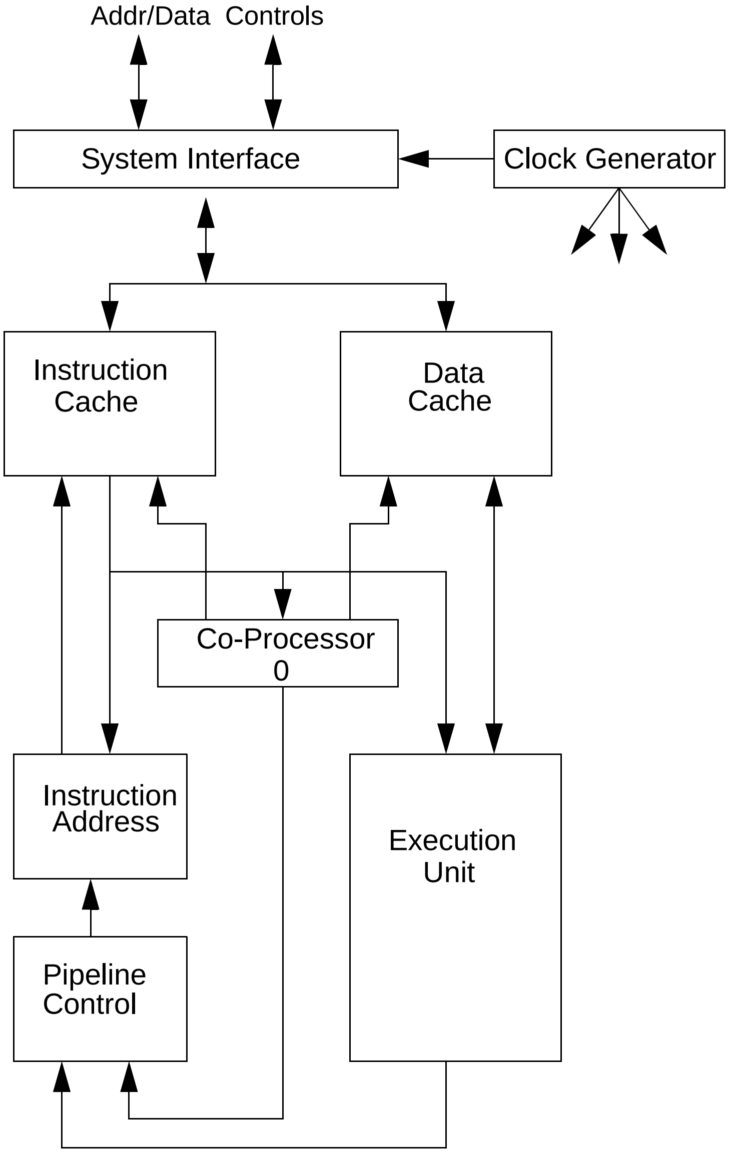
The R4300 processor has a five-stage execution pipeline. Each pipeline stage takes one pcycle (pclock runs at a multiple frequency of MasterClock set by the DivMode(1:0) pins). The execution of each instruction thus has a latency of at least five pcycles. An instruction might take longer; for example, the pipeline must stall when the required data is not in the cache and must be retrieved from main memory. 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 cycle.

The five stages of the R4300 pipeline are:

1. Instruction Fetch **IC**
2. Decode, file read, branch/jump **RF**
3. Execution **EX**
4. Data cache read **DC**
5. File or data cache write **WB**

The pipeline includes one pclock delay slot for branch type instructions. There is a hardware interlock supported for load type instructions.

Figure 1: Processor block diagram



The Processor block diagram shows the main components of the R4300 chip. They include:

1. Execution unit
2. Coprocessor 0
3. Pipeline control
4. Instruction address
5. Instruction cache
6. Data cache
7. System interface
8. Clock generation

The **execution unit** contains the hardware resources necessary to execute all of the MIPS integer and floating point instructions. It contains a 64-bit wide register file, a 64-bit wide integer/Mantissa data path, and a 12-bit wide exponent data path.

Coprocessor 0 contains the memory management unit and the exception processing unit. The memory management unit is responsible for effective address translation from virtual to physical, and for performing memory access checks between different memory segments (kernel, supervisor, user).

The translation lookaside buffer (TLB) technique is used to provide translation of virtual address to physical. R4300 chip supports VSIZE=40, and PSIZE=32. R4300 (like R4000 / R4400) supports seven different page sizes of 4 KBytes, 16 KBytes, 64 KBytes, 256 KBytes, 1 MBytes, 4 MBytes & 16 MBytes. The TLB contains 32 entries, each entry mapping to an odd/even pair of page frame numbers.

The exception processing unit contains all of the system control coprocessor registers. The format of data in these registers is described in detail in section 6 of this spec.

Pipeline control assures that the R4300 instruction pipeline operates properly when conditions such as: cache misses, flush buffer full, multicycle instructions, or system exceptions occur.

Instruction address calculates the effective address of the next instruction to be fetched. It contains the PC incrementer, branch address adder, and the conditional branch address selector.

Instruction cache is a directly mapped, virtually indexed, physically tagged cache. Each cache line includes 8 instructions, 21 tag bits, and 1 line valid bit. The cache data interface is 64 bits wide. Cache parity is not supported.

Data cache is a directly mapped, virtually indexed, physically tagged, write back cache. Each cache line includes 4 words of data, 21 tag bits, 1 line dirty bit, and 1 line valid bit. The cache read operation takes 1 cycle, but a store operation keeps the D-cache busy for 2 cycles. Cache parity is not supported.

System interface allows the processor access to external resources required to satisfy internal requirements. It contains a 32-bit wide, multiplexed address and data bus, clock signals, interrupts, and a number of control signals.

Clock generator multiplies the input clock frequency (MasterClock) to produce the pipeline clock. The ratio of PClock (pipeline clock) to MasterClock is set by DivMode(1:0) pins of the chip. The DivMode values of 0, 1, 2 and 3 define PClock to MasterClock ratios of 1:1, 1.5:1, 2:1, and 3:1 respectively. The system interface clock runs at the same frequency as the MasterClock. Using the Reduced Power (RP) bit of the Status Register, both pipeline and interface clocks can be switched to run at quarter speed. To minimize the skew between the input clock and the internal clocks, the chip uses phase lock loop (PLL) technology.

3.0 Operation Fundamentals

3.1 Power management

One of the objectives of the design of the R4300 chip is to minimize the power dissipation in order to make this chip suitable for use in battery operated systems, as well as in an environment where low power consumption/heat dissipation is desirable. Several architectural choices were made to achieve this objective. They include:

- Write back cache which reduces the system bus store traffic.
- Segmented direct mapped I-cache which reduces power consumption in the I-cache by enabling only the segment which contains the requested address.
- Doubleword read from the I-cache into a buffer. During typical sequential cache accesses the next instruction can be found in this buffer reducing the frequency of I-cache access.
- Integrating the fpu and integer units into a single execution unit with shared resources reduces the size of the chip, thus reducing overall interconnect capacitance.
- Using a 5 stage pipeline results in reduced logic, interconnect and overall smaller device sizes.

Also particular attention was paid to circuit and logic design to lower the power consumption of the chip.

The chip supports both normal and reduced power operation modes. Transitions from one of these modes to the other is caused by the software setting or resetting the RP bit in the CP0 Status Register. Setting the reduced power mode (RP=1) results in the chip reducing the pipeline clock frequency and the system interface clock frequency by a factor of 4.

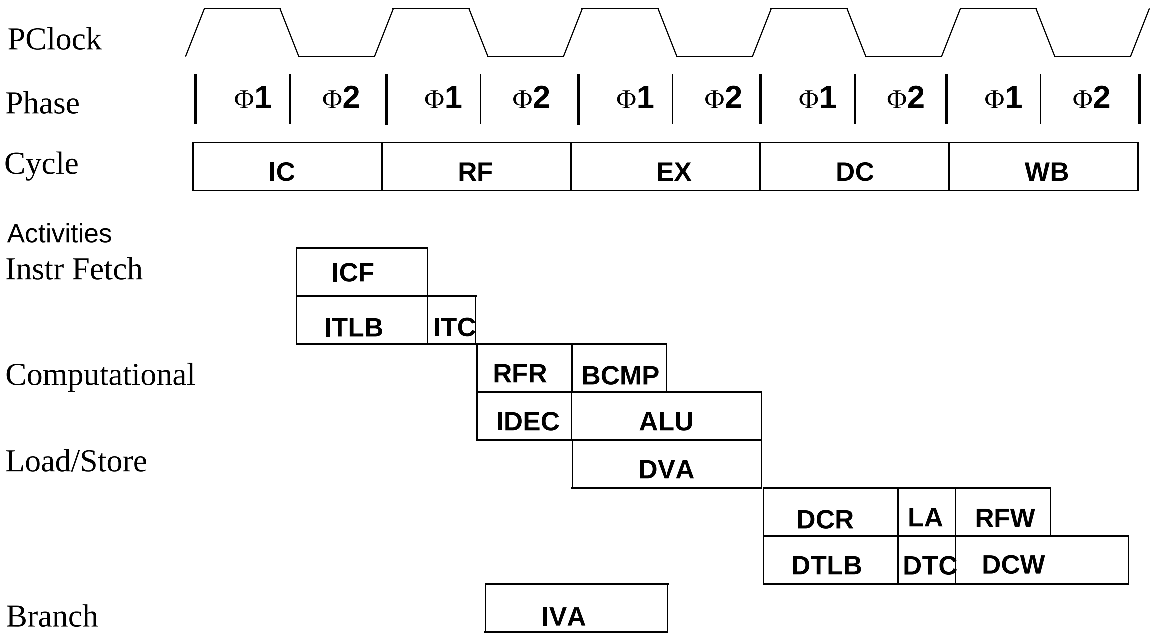
To further reduce the power consumption, the system may want to turn the power off for some of the system components. The R4300 chip supports this requirement by allowing the software easy access to all of the registers including GPR, FPR, and all CP0 registers. Thus, the state can be restored via the software reloading all of these registers after the power is restored. The cache content will not be saved, and therefore the cache should be invalidated during the power up routine and flushed during the power down routine. The R4300 chip supports all of the relevant cache and TLB operation instructions which will allow the invalidating all of cache and TLB locations. At power up/reset, when the Reset signal at the pin is deasserted, execution begins at the reset vector (0xbfc00000) in BigEndian mode. This vector is located within the unmapped and uncached address space so that the cache and TLB need not be initialized to handle this exception. As in the R4000, the contents of all registers are undefined when this exception occurs, except for the Random register which is initialized to TLBENTRIES-1, the Wired register which is initialized to 0, and the Status register which has SR and TS bits equal to 0 and ERL and BEV which are equal to 1. The other bits of the Status register are undefined. Upon ColdReset Config register bits BE is initialized to 1 for BigEndian mode and the EP field is initialized to 0 for "D" data rate. These bits can be written by software and are unaffected by Soft Reset or NMI.

3.2 Processor Pipeline

3.2.1 Pipeline Overview

The R4300 processor has a five-stage execution pipeline. That is, each instruction takes five clocks to complete and a new instruction can start on each clock. The pipeline clock, PClock, is a multiple of the frequency of the external system clock. The following figure illustrates the activities happening within each pipe stage as the function of each instruction type.

Figure 2: Pipeline Activities



Cycle	Phase	Mnemonic	Descriptions
IC	Φ1		
	Φ2	**ICF**	Instruction Cache Fetch
		ITLB	Instruction micro-TLB read
RF	Φ1	**ITC**	Instruction cache Tag Check
	Φ2	**RFR**	Register File Read
		IDEC	Instruction DECode
		IVA	Instruction Virtual Address calculation
EX	Φ1	**BCMP**	Branch Compare
	Φ1,Φ2	**ALU**	Arithmetic Logic operation
		DVA	Data Virtual Address calculation
DC	Φ1	**DCR**	Data Cache Read
		DTLB	Data joint-TLB read
	Φ2	**LA**	Load data Alignment
		DTC	Data cache Tag Check
WB	Φ1	**DCW**	Data Cache Write
		RFW	Register File Write

3.2.2 Pipeline Interlocks and Exceptions

The processor has a simple pipe which allows the overlap of instruction execution across the five pipe stages. The pipeline does in-order issue, in-order execution, and in-order completion, the same order as in the instruction stream.

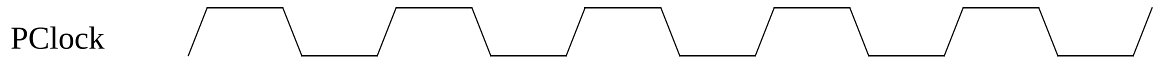
Pipeline flow is interrupted when an interlock condition is detected or when an exception occurs. The Interlock condition is resolved by stalling the whole pipe, while an exception aborts the relevant instruction as well as all those that follow and calls an exception handler from a predefined address.

The following figure illustrates the various interlock conditions and the different types of exceptions, at their predefined pipe stages.

For cases of simultaneous interlocks and exceptions from different pipeline stages, the interlocks and exceptions from the later pipeline stages will be processed before those from earlier pipeline stages i.e. exception from DC stage takes precedence to a coincident exception from the RF stage.

For simultaneous interlocks and exceptions from the same pipeline stage, the following figure lists different interlocks and exceptions in decreasing order of priority with the highest priority listed first.

Figure 3: Pipeline Interlocks and Exceptions



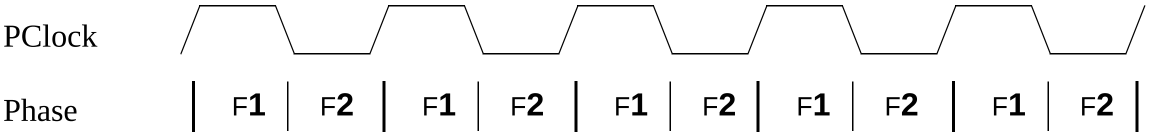
Stalls

IC	RF	EX	DC	WB
	ITM	LDI	DCM	CP0I
	ICB	MCI	DCB	
		CPI	COp	

Exceptions

IC	RF	EX	DC	WB
	IADE	SYSC	RST	
	ITLB	BRPT	NMI	
	IBE	CPU	OVFL	
		RSVD	TRAP	
			FPE	
			CPE	
			DADE	
			DTLB	
			WAT	
			DBE	
			INTR	

	Interlocks		Exceptions
ITM	Instruction TLB Miss	IADE	Instruction Address Error
ICB	Instruction Cache Busy	ITLB	Instruction TLB exception
LDI	Load Data Interlock	IBE	Instruction Bus Error
MCI	Multi-cycle Interlock	SYSC	SYSCALL instruction
CPI	Coprocessor 2 Interlock	BRPT	Breakpoint Instruction
DCM	Data Cache Miss	CPU	Coprocessor Unusable
DCB	Data Cache Busy	RSVD	Reserved Instruction
COp	Cache Op	RST	External reset exception
CP0I	CP0 Bypass Interlock	NMI	External NMI exception
		OVFL	integer overflow
		TRAP	TRAP instruction exception
		FPE	Floating Point exceptions
		CPE	Coprocessor 2 Exception
		DADE	Data Address Error
		DTLB	Data TLB exception
		WAT	Reference to Watch Address
		DBE	Data Bus Error
		INTR	External Interrupt signals



Stalls

IC	RF	EX	DC	WB
	ITM ICM	MCI LDI CPI	DCM DCB COp	

Exceptions

IC	RF	EX	DC	WB
	ITLB IBE	CPE Trap INTR	DTLB DBE WAT INTR	

	Interlocks		Exceptions
ITM	Instruction TLB Miss	ITLB	Instruction TLB invalid/refill
ICM	Instruction Cache Miss	IBE	Instruction Bus Error
MCI	Multi-cycle Interlock	CPE	Coprocessor Exception1
LDI	Load Interlock	Trap	Instruction Exceptions: syscall, breakpoint, trap, reserved instruction, coprocessor unusable, overflow, FP exceptions.
CPI	Coprocessor Interlock1		
DCM	Data Cache Miss		
DCB	Data Cache Busy		
COp	Cache Op		
		INTR	Interrupt, NMI, Reset
		DTLB	Data TLB invalid/modified/refill
		WAT	Reference to Watch Address

Notes

1. Coprocessor Interlock stall CPI and Coprocessor Exception CPE are defined to

3.2.3 Pipeline Operation

The pipeline operation can best be illustrated by a few examples that show how certain typical instructions are executed. The instructions used in these examples are: ADD, JALR, BEQ, TLT, LW, and SW. Note that the floating point instructions are executed in the pipeline just like multicycle integer instructions.

3.2.3.1 Add ADD rd,rs,rt

IC stage. In phase 2, the fourteen least significant bits (LSBs) of the virtual address are used to address the I-cache. The two most significant bits of these are used to select one of four I-cache banks. The remaining LSBs are used to address the selected bank and the cache line physical page frame number (PPFN) tag. The micro-TLB translates the virtual page frame number (VPFN) to the PPFN. Late in phase 1 of the next pipe stage RF, the PPFN is compared with the PPFN tag from the cache and the cache hit/miss signal is produced. The virtual PC is incremented by 4 so that in the following cycle the next sequential instruction can be fetched.

RF stage. During phase 2, the 2-port register file is addressed with the rs and rt fields and the register's data becomes valid at the register file's output. Meanwhile, the bypass muxes select inputs from the EX or DC stage output depending on the need for an operand bypass.

EX stage. The ALU controls are set to do an A+B operation, the operands flow into the ALU inputs, and the ALU operation is started. The results of the ALU is latched into the ALU output latch during phase 2.

DC stage. A no_op stage for this instruction. The data from the output of the EX (ALU) is moved into the output latch of the DC.

WB stage. During phase 1, the WB latch feeds the data to the input of the register file. The file is addressed with the rd field and the file write strobe is enabled. By the end of phase 1, the data is written into the register file.

3.2.3.2 Jump and Link Register JALR rd,rs

IC stage. Please refer to the ADD instruction.

RF stage. During phase 2, the register addressed by the rs field is read out of the register file.

EX stage. During phase 1, the value of register rs is clocked into the virtual PC latch, which will then be used in phase 2 to fetch the next instruction.

The value of the virtual PC incremented during the RF stage is incremented again to produce the link address PC+8 where PC is the address of the JALR instruction. The resulting value is the PC to which the program will eventually return. This value is placed in the Link output latch of the Instruction Address unit.

DC stage. The PC+8 value is moved from the Link output latch to the output latch of the DC pipeline stage.

WB stage. Refer to the ADD instruction. Note that if no value is explicitly provided for rd then register 31 is used as the default. If rd is explicitly specified, it cannot be the same register addressed by rs; otherwise, the result of executing such an instruction is undefined, as stated in the MIPS R-Series Architecture manual.

3.2.3.3 Branch on Equal BEQ rs,rt,offset

IC stage. Refer to the ADD instruction.

RF stage. During phase 2, the register file is addressed with the rs and rt fields and the contents of these registers are placed in the register file output latch.

EX stage. During phase 1, a check is performed to determine if each corresponding bit position of these two operands has equal values. If they are equal, the next PC will be set to PC+target, where target is the sign extended offset field. If they are not equal, the next PC will be set to PC+4. The next PC resulted from the branch comparison is valid at the beginning of phase 2 for instruction fetch.

DC stage. no_op

WB stage. no_op

3.2.3.4 Trap if Less Than TLT rs,rt

IC stage. Refer to the ADD instruction.

RF stage. Refer to the ADD instruction.

EX stage. The ALU controls are set to do an A-B operation, the operands flow into the ALU inputs, and the ALU operation is started. The results of the ALU is latched into the ALU output latch during phase 2.

DC stage. The sign bits of operands and of the ALU output latch are checked to determine whether the “Less_Than” condition is met. If it is met, the next PC is loaded with the value of the exception vector. The instructions in the previous pipeline stages are killed. The EXL bit of the CP0 Status Register is checked. If the "Less_Than" condition is not met then the PC continues incremented and sequential instruction flow is executed.

WB stage. The CP0 EPC Register is loaded with the value of the PC for the trap instruction if the “Less_Than” condition was met and the EXL bit was not set in the DC stage.

3.2.3.5 Load Word LW rt,offset(base)

IC stage. Refer to the ADD instruction.

RF stage. Refer to the ADD instruction. Note that the base field is in the same place as the rs field.

EX stage. Refer to the ADD instruction. For this instruction, the inputs to the ALU come from the GPR[base] and from the sign extended offset field. The result of the ALU operation latched into the ALU output latch in phase 2 represents the effective virtual address of the operand.

DC stage. The D-cache is accessed in parallel with the TLB. The cache index field is compared with the Physical Frame Number (PFN) field of the TLB entry. After passing through the load aligner, the aligned data is placed in the DC output latch during phase 2.

WB stage. During phase 1, the cache read data is written into the file addressed by the rt field.

3.2.3.6 Store Word SW rt,offset(base)

IC stage. Refer to the ADD instruction.

RF stage. Refer to the LW instruction.

EX stage. Refer to the LW instruction for the effective address calculation. From the RF output latch the GPR[rt] is sent through the bypass mux and into the main shifter. The shifter performs the byte align operation for the operand. The results of the ALU and the shift operations are latched in the output latches during phase 2.

DC stage. Refer to the LW instruction for the cache access. Additionally, the merged data from the load aligner is moved into the store data output latch during phase 2.

WB stage. If there was a cache hit, the content of the store data output latch is written into the D-cache at the appropriate word location.

Note that all store class of instructions use the Data Cache for two consecutive pipeline clock cycles. If the following instruction requires to use the Data Cache, the pipeline will be stalled by one pcycle, to complete the writing of an aligned store data.

4.0 Execution Unit

4.1 Goals

The execution unit is designed to reduce power consumption and simplify the hardware requirements while providing an adequate level of performance by maximizing usage of each functional element. For R4300's target applications, floating point performance is less critical than integer performance.

4.2 Overview

The instruction execution unit is tightly coupled to the on-chip cache memory system, I/DCache, and the on-chip memory management unit, CP0. The 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 co-processor CP1, are processed by the same hardware as for the integer instructions. That is to say, there is no separate floating point coprocessor. However, the execution of floating-point instructions can still be disabled via the coprocessor usability **CU** bit defined in the CP0 Status Register.

The pipelined execution unit performs different functions at different times, depending on the instruction stream. Multi-cycle operations such as floating-point addition require a more elaborate control sequence than single-cycle operations such as integer addition. Since simplicity is the key to achieving low-cost and low-power in this design, multi-cycle instructions are not allowed to overlap with the execution of any other instructions. That is, the pipeline will stall until the current instruction in the EX stage completes its multi-cycle execution sequence. Because of this, the pipeline control is simple, the precise exception is easily maintained and the requirement for a dual write-port register file is eliminated.

In order to support system power down mode, all internal state information vital to restart the processor from the point of power cut off is read and write accessible. Prior to power off, all this information must have been read and saved off chip into the non-volatile memory. Also, it is the system's responsibility to power off the chip when the system is in idle state. Note that the Load Link **LLbit** bit is not required to be saved since it would be cleared by the cache invalidation during the power up routine.

Pipeline control within the execution unit is specified with a simple coprocessor interface protocol. This makes it easy to support future enhancements such as a graphic accelerator or other special purpose coprocessor.

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 are inactive. Also, when functional elements in the data path are idle, they operate on a constant selected to minimize power dissipation, instead of on data from the bus.

4.3 Functional Description

The instruction execution unit is designed to process the MIPS-III instruction set efficiently and cost effectively, while still maintaining downward compatibility with the MIPS-I and MIPS-II ISA.

Even though this processor does not support a multiprocessor operating environment, the synchronization support instructions as defined in MIPS-II/-III ISA -- load linked and store conditional, are still processed correctly, in order to be compatible with R4000. The load link (LLbit) is emulated -- set by the LL instruction, cleared by an ERET or cache miss, and tested by the SC instruction. The only operation to the LLbit that is excluded is reset due to a cache invalidation by an external agent.

Note that the SYNC instruction is executed as a NOP instruction since all load/store instructions in this processor are executed in program order.

4.3.1 Instruction latencies

Here is the latency for integer instructions, measured in processor pipeline clock cycles:

Table 2: Integer instruction latencies

Instruction Types	Pipeline Clock Cycles
Mult	5
MultU	5
DMult	8
DMultU	8
Div, DivU	37
DDiv, DDivU	69
Branch instr[1]	1
Jump instr[1]	[1]
Load instr[2]	1
Store instr	1

Notes:

1. The taken branch instruction is fetched in the EX stage of the branch instruction. The branch comparison and the target address calculation are done in phase 1 of the EX stage. The MIPS architecturally defined branch delay slot of one cycle is still required. Similarly, the jump instruction also requires one delay slot.
2. To be compatible with MIPS-II instruction set, hardware will interlock if the result of a load is to be used by the immediately following instruction.

All data movement between the floating-point and memory is accomplished by coprocessor load and store operations. Data may be directly moved between the floating-point coprocessor and the integer processor by move to and move from coprocessor instructions:

Table 3: Instruction Latencies/Execution Rate on Floating-Point Data Movement

Instruction	Pipeline clock cycles
LWC1[1]	2/1
SWC1	1/1
LDC1[1]	2/1
SDC1	1/1
MTC1	1/1
MFC1	1/1
DMTC1	1/1
DMFC1	1/1
CTC1	1/1

Notes:

1. The hardware will interlock for one cycle if the load result is used by the instruction in the load delay slot.

To obtain the optimum performance at the given power and cost goals, the R4300 pipeline does not perform an EX to RF bypass for the floating-point result of a convert, computational, LWC1 or LDC1 instruction. If the subsequent FP instruction in RF stage depends on the result of the current EX-stage FP instruction, the current EX-stage FP instruction will complete. Its EX-stage result will be registered into the DC stage. Meanwhile, the RF-stage FP instruction will advance into the EX-stage. It is then stalled for one pipeline clock to wait for the result to be bypassed from DC-to-EX, before it begins the execution.

Note that this EX-to-RF bypass limitation does not apply to integer operations nor to floating-point data movement instructions (except LWC1 and LDC1).

Note the extra clock latency for the LWC1 and LDC1 instructions is due to the one clock Load Data Interlock stall for bypassing the data from the DC stage into the EX stage.

The R4300 execution unit incorporates variable latency for most floating-point operations, to provide some (modest) boost in performance, and in some cases to reduce the complexity of the design. The variable latency design is as simple as possible. Thus if during a multi-cycle an exception can be detected by early examining the source operands (i.e. a source exception), the instruction takes only two cycles in EX. Likewise, if the non-exceptional result of zero or infinity can be determined by just examining the operands, then the result is also be returned in two cycles (e.g. infinity times a non-zero number). FP exceptions, other than source exceptions do not terminate an instruction until it has completed as if there were no exception. In other words, result exceptions are not reported as they are found, instead they are reported at the end of normal execution.

Floating point add and subtract terminates on the second cycle if a source exception occurs or if at least one operand is zero or infinity. The instruction completes on the third cycle in all other cases.

Floating point multiply finishes in two cycles if a source exception is detected, or if, during the first cycle, the result can be determined to be zero or infinity. It finishes in the second cycle if at least one of the operands is a power of 2. In all other cases it takes the full number (i.e. the maximum specified for the format) of cycles to complete. Thus, multiply does not finish as soon as the remaining bits are zero. Also, since multiply uses the whole EX stage of the datapath on every iteration, there can be no overlap between multiply and add.

Floating Point Divide and Square Root finish in the second cycle on either a source exception or if the result can be determined to be zero or infinity. Otherwise they continue to take the maximum amount of cycles. Integer divide's variable latencies are like floating point divide's except that integer divide never returns an answer on the first cycle and it can't cause any exceptions. Thus, integer divide always takes the maximum number of cycles.

Floating-point convert instructions also complete in the second cycle for various trivial cases.

Execution latencies of a floating-point instruction without data dependency are shown in the following table:

Table 4: Floating-point Instruction Latencies[1]

	Pipeline clock cycles[3]			
Instruction.Format	S	D	W	L
Add.fmt	3	3	-	-
Sub.fmt	3	3	-	-
Mul.fmt	5	8	-	-
Div.fmt	29	58	-	-
Sqrt.fmt	29	58	-	-
Abs.fmt	1	1	-	-
Mov.fmt	1	1	-	-
Neg.fmt	1	1	-	-
Round.W.fmt	5	5	-	-
Trunc.W.fmt	5	5	-	-
Ceil.W.fmt	5	5	-	-
Floor.W.fmt	5	5	-	-
Round.L.fmt	5	5	-	-
Trunc.L.fmt	5	5	-	-
Ceil.L.fmt	5	5	-	-
Floor.L.fmt	5	5	-	-
Cvt.s.fmt	-	2	5	5
Cvt.d.fmt	1	-	5	5
Cvt.W.fmt	5	5	-	-
Cvt.L.fmt	5	5	-	-
C.cond.fmt	1	1	-	-
BC1T[2]	1			

Notes:

1. If the FP register result of a FP instruction (except Mov.fmt) is needed by the subsequent instruction, an additional one pipeline clock is required for the result to be bypassed from DC to EX stage.
2. Architecturally defined branch delay slot of one also applies to all branch instructions on floating point coprocessor condition.
3. The trivial cases for the multicycle FP instructions take two pipeline clocks to complete.

All CPU/FPU instructions that are not mentioned in the above tables have a latency of one pipeline clock cycle.

4.3.2 Unit Organization

The execution unit's data path consists of:

- A 64-bit Integer/Mantissa Data Path
- An Operand Bypass Network
- 32 64-bit Integer Registers
- 32 64-bit Floating-point Registers
- A 12-bit Exponent Data Path
- A 64-bit Instruction Virtual Address Generator

The execution control logic comprises of:

- An Instruction Decoder
- A Pipeline Run/Stall state machine
- An Integer multiply/divide sequencer
- A Floating-point instruction sequencer
- An Operand Bypass control
- An Exception detector/generator

4.3.2.1 Integer/Mantissa Data Path

The integer/mantissa data path is 64 bits wide and is compatible with both 32-bit and 64-bit operands for integer and floating-point numbers. Proper sign-extension or zero-fill is performed on the operands prior to any computations. Correspondingly, sign-extension or zero-fill is performed on intermediate results when required. Exception testing such as integer overflow is done with proper bit extraction.

The data path handles partial-word read or write operations for load and store instructions in both big- or little-endian mode. This mode is specified by the combination of the big-endian memory system mode bit as set by the BigEndian BE bit in CP0 Configuration Register and ReverseEndian RE mode bit in CP0 Status Register.

For the store class of instructions, the main bi-directional shifter performs an alignment shift on the register read data. No concatenation of register read data with original memory data is necessary since the Data Cache has byte-wide write enable controls.

For the load class of instructions, it is necessary from the performance point of view to maintain a load delay of one clock cycle. Due to the timing requirements imposed by this load delay goal, 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 integer/mantissa data path has only one carry-propagate adder, one multiplier, and one bi-directional shifter for all integer and floating-point computation instructions. The adder is also used to compute data virtual address for load and store, and to compare two operands in trap instructions. The multiplier is used for both integer and floating-point multiplication, in single or double precision. The shifter has built-in guard/round/sticky collection logic for an FP pre-alignment shift. The shifter is also responsible for FP post-normalization, integer variable shift, and store alignment shift.

The data path has one Leading Zero Counter for floating-point normalization shift calculation, one Boolean Logic functional unit for integer logic operations, and a floating-point unpacker and a repacker. The unpacker breaks down single- and double-precision formatted operands into sign, exponent and mantissa fields, while the Repacker does the reverse process.

4.3.2.2 Operand Bypass Network

For performance reasons, it is necessary for the results in the instruction execution EX stage and the data cache access DC stage to be available to subsequent instructions waiting for it in the following EX stage as the source operands rs and/or rt. Thus, the Operand Bypass Network is built into the data path to allow feedback of results registered from the EX and DC stages to the input of EX stage.

Similarly, to maintain the minimum branch delay slot of one pipeline clock cycle for all branch instructions on the floating-point co-processor condition, the result from the preceding floating point compare instruction in the EX, DC, or WB stage will be fed back for the branch condition testing in the RF stage.

In cases where the result of a particular instruction (Load) is not available until the DC stage but is required by the next instruction in its EX stage then the hardware will cause Load Data Interlock stall. The Load Data Interlock stalls the pipeline for one clock while the data from the DC stage is bypassed into the EX input registers for use during the next clock cycle.

4.3.2.3 Register File

The register file has total of sixty-four 64-bit wide registers to accommodate the MIPS-III ISA requirement of 32 general-purpose GPR registers and 32 floating-point FGR registers. The register file is a two read-port one write-port design.

Of the many user selectable operating modes, two modes affect the accessibility of this register file. The UX mode bit in the CP0 Status Register selects between the 32- or 64-bit user addressing and operating modes. When operating in the 32-bit user mode, the most-significant 32 bits of the GPR registers contains the sign-extended value of the register bit 31. The FR mode bit selects between 16 or 32 floating-point registers. When operating with 16 floating-point registers, all odd physically addressed registers are inaccessible.

In addition to the integer and floating-point general-purpose registers, GPR and FGR, six special registers are included in the MIPS-III ISA:

- A 64-bit Program Counter **PC**
- A 64-bit Integer Multiply/Divide Register higher result **Hi**
- A 64-bit Integer Multiply/Divide Register lower result **Lo**
- A 1-bit Load/Link Register **LLbit**
- A 32-bit Floating-point Implementation/Revision Register **FCR0**
- A 32-bit Floating-point Control/Status Register **FCR31**

Among all these special registers, registers Hi, Lo and FCR31 are read and write accessible to system software to allow retrieval and restoration of the processor states during the system power up and down routines.

4.3.2.4 Floating-Point Instruction Execution

In conjunction with system software, execution of single and double-precision floating-point instructions conform to the ANSI IEEE-754/1985 standard for binary floating-point arithmetic. The MIPS-II/III ISA specifies that the extended and quad data formats are not implemented in hardware, thus, the unimplemented operation exception is initiated for these formats.

Floating-point exceptions are logically precise. These include five IEEE-specified exceptions: underflow **U**, overflow **O**, divide-by-zero **Z**, invalid **V** and inexact **I**, plus the MIPS-defined unimplemented operation **E**. This last exception is not maskable in the R4300 implementation.

The hardware implementation cost is significant for processing denormalized **DEN** input operands, except for compare instructions. Thus, the unimplemented operation exception is initiated, and software emulation takes over when a denorm is encountered. For un-normalized results a properly signed zero or minimum norm is returned if the Flush-to-Zero mode **FS** bit is set in the Floating-point Status **FCR31** register. If the FS bit is not set, an unimplemented operation exception is taken.

For IEEE-specified invalid operations, with the trap disabled, the default quiet Not-a-Number QNaN is generated.

The exponent data path is a 12-bit dual carry-select adder, which is used for exponent subtraction, pre-alignment shift calculation, and exponent addition for post-normalization final exponent update.

4.3.2.5 Instruction Address Unit

The Instruction Address unit is responsible for the generation of 64-bit instruction virtual addresses to be used by TLB, ICache and CP0. The unit has its own incrementor to calculate the next sequential PC address. To maintain a branch latency of 1 pipeline clock cycle, it has its own equality comparator and a separate ripple-carry adder to generate the branch target address.

In addition, the Address unit has Exception Vector Generation logic to decode the type of exception and then present the appropriate vector as the next PC address. It also has the Exception PC **EPC** register pipe chain to maintain a history of PC addresses for each pipe stage so that the PC address associated with the exception causing instruction can be loaded into the EPC or ErrorEPC register. Note that Exception Handling is done in the CP0 unit.

Sources for instruction execution exceptions include:

- Integer Overflows
- Traps
- System Calls
- Breakpoints
- Reserved Instructions
- Coprocessor Unusable Exceptions
- Floating-point Exceptions

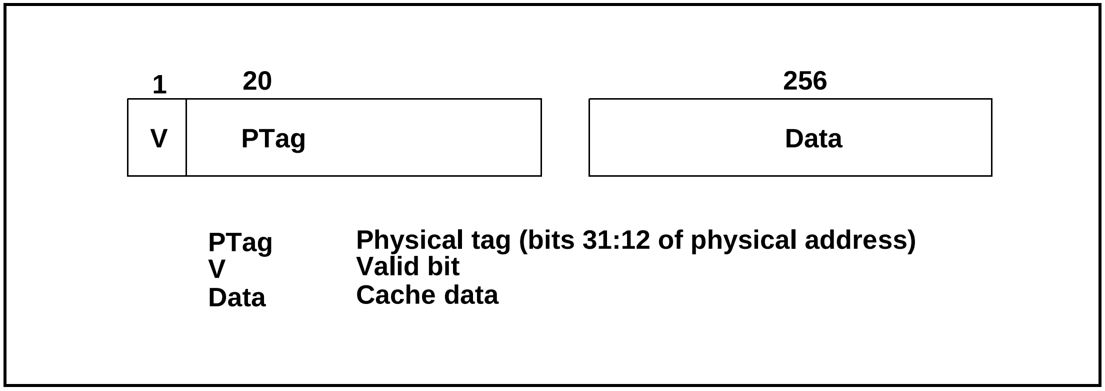
5.0 Data and Instruction Caches

R4300 contains on-chip instruction and data caches. Both the data cache and the instruction cache are direct mapped. The data cache implements a Write-Back write policy. R4300 has an instruction cache size of 16 kilobytes and a data cache size of 8 kilobytes. Since both instruction and data caches are larger than TLB page size (4 KBytes), software should exercise caution to avoid the “aliasing problem”. This “aliasing problem” arises when two different virtual addresses map to the same physical page.

5.1 Cache Organization

The format of an instruction cache line is shown below. Cache parity is not supported on R4300.

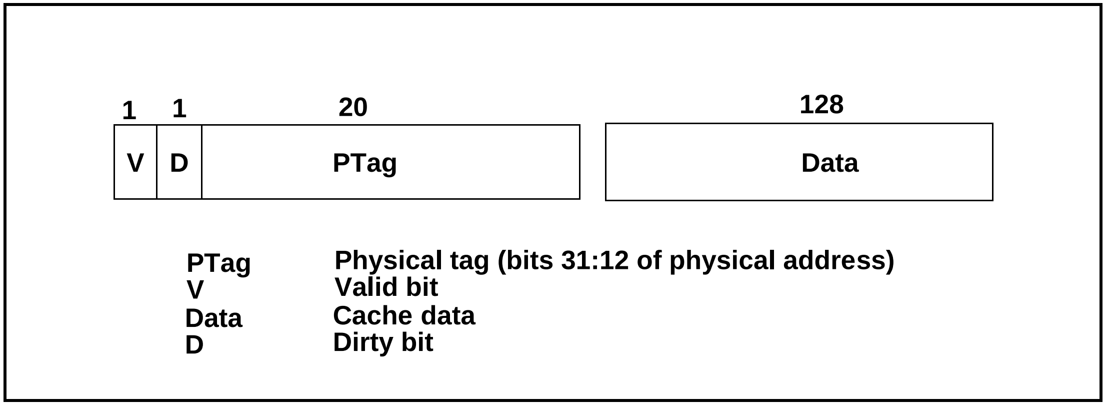
Figure 4: Format of Instruction Cache line



PTag Physical tag (bits 31:12 of physical address)
V Valid bit
Data Cache data

The format of a data cache line is shown below.

Figure 5: Format of Data Cache line



PTag Physical tag (bits 31:12 of physical address)
V Valid bit
Data Cache data
D Dirty bit

5.2 Cache States

5.2.1 Instruction Cache

The Instruction cache maintains two cache states:

- Invalid
- Valid

A cache line in an *Invalid* state does not contain valid information. A cache line in a *Valid* state contains valid information.

5.2.2 Data Cache

The Data cache maintains three cache states:

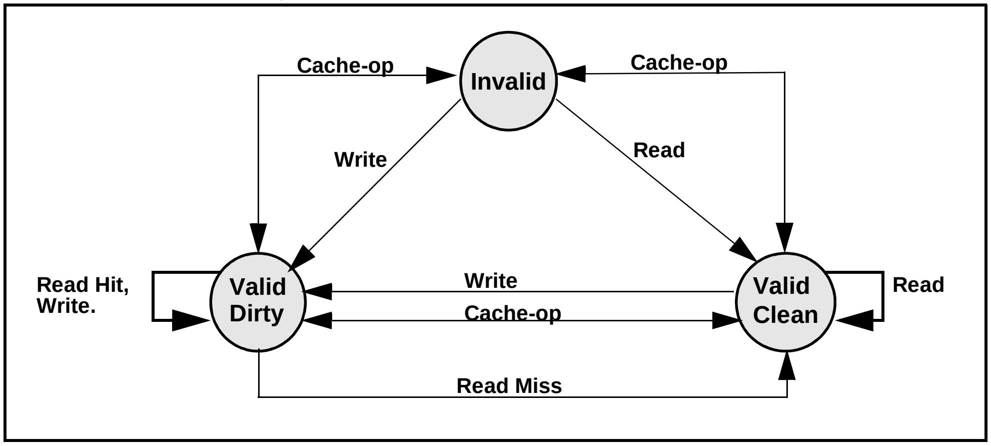
- Invalid
- Valid Clean
- Valid Dirty

A cache line in an *Invalid* state does not contain valid information. A cache line in a *Valid Clean* state contains valid information and consistent with main memory. A cache line in *Valid Dirty* state, has valid data but it is not consistent with main memory.

5.2.2.1 Data Cache State transition

The following diagram illustrates the data cache state transition sequence.

Figure 6: Data Cache State Transition



5.2.3 Cache state change during processor execution

The state of a *valid* cache line may be modified due to the processor executing a cache operation. These operations are discussed later in this document.

5.2.4 Manipulation of the Caches by an External Agent

R4300 does not provide any mechanisms for an external agent to examine and manipulate the state and contents of the caches.

5.2.5 Cache Line

The line size for the instruction cache is 8 Words (32 Bytes). The line size for the data cache is 4 Words (16 Bytes).

5.2.6 Instruction Cache line replacement

During an instruction cache miss, a memory read is issued. After the requested line has returned from memory, it will be written to the instruction cache. At that time the pipeline will resume execution, and the instruction cache will be re-accessed.

5.2.7 Data Cache line replacement

Since data cache is Write-Back, a cache line load will be issued to main memory on a load or store miss. Details are described below.

Data load miss:

- If the cache block is not dirty, it will be replaced with the requested line.
- If the cache block is dirty, its contents will be moved to the flush buffer, the requested line will be loaded into the cache block, and data in the flush buffer will be written to memory.

Data Store miss:

- If the cache block is not dirty, the requested line will be merged with the store data and written to cache.
- If the cache block is dirty, its contents will be moved to the flush buffer, the requested line will be merged with the store data and written to cache, and data in the flush buffer will be written to memory.

After the data from memory is written to the data cache, the pipe will resume execution.

5.3 Cache Access Time

The instruction and data caches can be accessed for reads in one processor cycle.

Data writes are pipelined and effectively terminate in one processor cycle. In the first cycle, the tag is checked and the second cycle the data is written into the data RAM.

5.4 Cache Miss Handling

For an instruction cache miss, the sequence is:

(1) Move the instruction physical address to the pads.
(2) Wait for a pipeline clock *PClock,* aligned with the system clock *SClock* boundary.
(3) Read memory.
(4) Move memory data to the instruction cache array.
(5) Write memory data into the instruction cache array.
(6) Restart the processor pipe.

The instruction cache miss penalty in number of *PClocks* is: 4 + (Clock Alignment) + (Memory Access & Transfer Time for Entire Cache Line) + 1.

For a data cache miss, the sequence is based on a “Critical Data Word First” scheme. The processor will restart its pipe as soon as the memory supplies the desired word in the first doubleword of a block transfer. The sequence is summarized as following:

(1) Move the data physical address to pads. At the same time, move the dirty victim cache line to the flush buffer if needed.
(2) Wait for a pipeline clock *PClock*, aligned with system clock *SClock* boundary.
(3) Read memory.
(4) Receive the desired doubleword.
(5) Receive other doubleword. Meanwhile, move the desired data to the processor pipe. Interlock the data cache from being accessed by a subsequent instruction.
(6) Move memory data to the data cache array. Continue to interlock the data cache.
(7) Write memory data into the data cache array. Continue to interlock the data cache.

The data cache miss penalty in number of *PClocks* is: 3 + (Clock ALignment) + (Memory Access & Transfer Time for Critical Doubleword) + 1.

5.5 Cache Operations

R4300 supports all Mips R4000PC / R4200 / R4400PC processor cache operations. Cache operation instructions are part of MIPS-III ISA and are described in greater detail in the Mips R-Series Architecture manual. A list of cache operations are provided in the following table.

Table 5: Cache Operations

Name	Caches	Operation
Index Invalidate	Instruction	Sets the cache state of cache block to invalid.
Index WriteBack Invalidate	Data	Examines cache state, if Valid Dirty, then that block is written back to main memory. Then the cache block is set to invalid.
Index Load Tag	Instruction & Data	Read the tag for the cache block at the specified index and place it into TagLo.
Index Store Tag	Instruction & Data	Write the tag for the cache block at the specified index from the TagLo register.
Create Dirty Exclusive	Data	If the cache does not contain the specified address, and the block is Valid Dirty the block will be written back to main memory. Then the tag will be set to the specified physical address and will be marked valid.
Hit Invalidate	Instruction & Data	If the cache block contains the specified address, cache block will be marked invalid.
Hit WriteBack Invalidate	Data	If the cache block contains the specified address, and it is Valid Dirty, the data will be written back to main memory. Then, the cache block is marked invalid.
Fill	Instruction	Fill the Instruction cache block from main memory.
Hit WriteBack	Data	If the cache block contains the specified address, and it is marked Valid Dirty, the block will be written back to main memory, and marked Valid Clean.
Hit WriteBack	Instruction	If the cache block contains the specified address, the block will be written back to main memory unconditionally.

5.6 Reset Effects

Cold or Warm reset will not set the cache states to invalid. The invalidation of caches is left to software.

5.7 Flush Buffer

R4300 contains an on-chip flush buffer. This 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 with 4 double word data. For uncached write operations, flush buffer can accept any combination of single or doubleword 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 reference 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. The format of the flush buffer is shown in the figure below.

Figure 7: Flush Buffer format

1	4	32	64
R/W	Size	Address	Data
R/W	Size	Address	Data
R/W	Size	Address	Data
R/W	Size	Address	Data

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 this buffer until the requested data is returned from external interface. At this time the processor pipeline will continue to run while the flush buffer writes the dirty data to the external interface.

If this buffer is full and the processor attempts a load or a store which requires external resources, the processor pipeline will stall until this buffer is no longer full.

6.0 Cache Test Mode.

This mode allows an I.C. tester or external test circuit to read and write the internal instruction and data cache memories directly via the use of a few pins.

6.1 Cache Memory Description

The R4300 caches have four functional parts: Data Cache Data (DCData), Data Cache Tag (DCTag), Instruction Cache Data (ICData) and Instruction Cache Tag (ICTag).

DCData is composed of two banks of static RAM. Each bank is 256 rows deep. Each of these rows contains two 64-bit words. This gives a total of 128 columns per row. A 2:1 multiplexor is used to connect the double-bit columns to a 64 bit bus.

DCTag is composed of a 44 bit wide by 256 bit deep static RAM. Each 44-bit row consists of two 22 bit tags. Each tag contains a 20 bit physical page address, a valid bit, and a dirty bit.

ICData is composed of four banks of static RAM. Each bank is 256 rows deep. Each of these rows contains two 64-bit words. This gives a total of 128 columns per row. A 2:1 multiplexor is used to connect the double-bit columns to a 64-bit staging register, which in turn is multiplexed onto a 32-bit instruction bus.

ICTag is composed of a 42 bit wide by 256 bit deep static RAM. A 2:1 multiplexor connects the double-bit columns to a 21 bit tag bus. The 21 bits include 20 bits of physical page address, plus a valid bit.

6.2 Test Mode Description

Cache Test Mode uses the existing data path to move data between the caches and the I/O pins. Access of the instruction and data caches can be pipelined. Since the internal datapath of the processor are 64 bits and the system interface bus is 32 bits, data is transferred on the bus in two consecutive system clock cycles, but is written to and read from the cache internally in one processor clock cycle. Cache Test Mode requires the external clock to processor clock ratio to be 2:1 (DivMode(1:0) pins should have the value “10”). Since the caches are written with 64-bit data, the index must be aligned to double word addresses (i.e. SysAD<2..0> = 000), and tag indexes must be aligned to block addresses, which is 16 bytes for data cache and 32 bytes for instruction cache (i.e. SysAD<3..0> = 0000 for dcache, SysAD<4..0> = 00000 for instruction cache).

For both reads and writes, the addressed double word data will be presented on the bus in little endian order, the least significant bits <31..0> will be Data0 and the most significant bits <63..32> will be Data1.

For reads, address and command are clocked in, and three cycles later data is clocked out of the processor for both data and tag portions of instruction and data caches. For writes, DCData, DCTag portions and ICData portion behave similarly, the index is followed by the write data and writes can be issued back-to-back; however ICTag writes require one dead cycle between successive writes.

A timing diagram is given in the following sections illustrating read and write transactions.

Note that before test mode can function, the internal and external clock must be stable. For more information see R4300 processor external specification on power up procedure.

To enable the Cache Test Mode on R4300, JTAG commands are utilized to set the internal Cache Test Mode Sticky bit. Upon power up the following sequence of events enable Cache Test Mode:

DivMode(1..0) set to “10” (divide by two mode)

ColdResetB asserted (low)

ResetB asserted (low)

ColdResetB deasserted (high) after the clocks are stable

Shift in “110” into JTAGIR register (this sets the Cache Test Mode Sticky register)

To enable the cache test mode on the PGA packaged R4300 which has an external TestModeB pin, the following signals must be asserted. This is the normal R4300 power up sequence, but with the TestModeB pin asserted.

TestModeB asserted (low) while ColdResetB is asserted (low)

DivMode(1..0) set to “10” (divide by two mode)

ResetB asserted (low)

ColdResetB deasserted (high) after the clocks are stable

In this state most internal control logic is reset, and the processor pipeline is stalled. During cache test, all unused input only pins must be driven and held to a stable logic level. While in cache test mode the processor will only drive the SysAD bus in the fourth (and fifth for Data1) cycle of a read command. At all other times these inputs must be driven by the Tester to a valid logic level to prevent damage to the input buffers of the processor.

6.3 Test Mode Commands

The IntB pins are used as a command bus for cache test mode. The following is a list of available commands in cache test mode.

- IntB[2..0] = 000 DCData array Read.
- IntB[2..0] = 001 DCTag array Read.
- IntB[2..0] = 010 DCData array Write.
- IntB[2..0] = 011 DCTag array Write.
- IntB[2..0] = 100 ICData array Read.
- IntB[2..0] = 101 ICTag array Read.
- IntB[2..0] = 110 ICData array Write.
- IntB[2..0] = 111 ICTag array Write.
- IntB[4] = 1 NOP.

IntB[4] should be used as command strobe, where a read or write command will only take place if this pin is low. This allows idle cycles / NOP cycles in test mode.

It is necessary to issue at least 6 NOP commands to initialize the internal state of the processor once the test mode is activated.

6.4 Cache Memory Address

The cache index is clocked in from SysAD bus and held in a cache Index register. IntB[3] is used as an enable for this register. When this input is low a new index is read from the SysAD bus. The bits of the SysAD bus that are used for this are indicated below.

- For ICData portion SysAD[13..3] is used as index to cache.
- For ICTag portion SysAD[13..5] is used as index to cache.
- For DCData portion SysAD[12..3] is used as index to cache.
- For DCTag portion SysAD[12..4] is used as index to cache.

6.5 Cache Read

After a cache read command is clocked in, the processor will read the selected cache and then output the data onto SysAD bus. The Tester must stop driving the SysAD bus one cycle after it has issued the read command. After four cycles the processor will drive the SysAD bus and provide the data requested. It will drive the bus for one system cycle.

If the target is DCData or ICData, a 64 bit doubleword, will be read.Thus all bits of the SysAD bus will be driven. If the target is either DCTag or ICTag, only a subset of the SysAD bus will be driven.

When reading the DCTag the SysAD bus will contain the following values:

- SysAD[31..12] 20 Tag Bits
- SysAD[11] Valid Bit
- SysAD[10] Dirty Bit
- SysAD[9..0] Index

When reading the ICTag the SysAD bus will contain the following values:

- SysAD[31..12] 20 Tag Bits
- SysAD[11] Valid Bit
- SysAD[10..0] Index

The following diagram illustrates cache read timing. Note that the behavior is common to all portions of cache.

Figure 8: DCData and ICData Read Timing

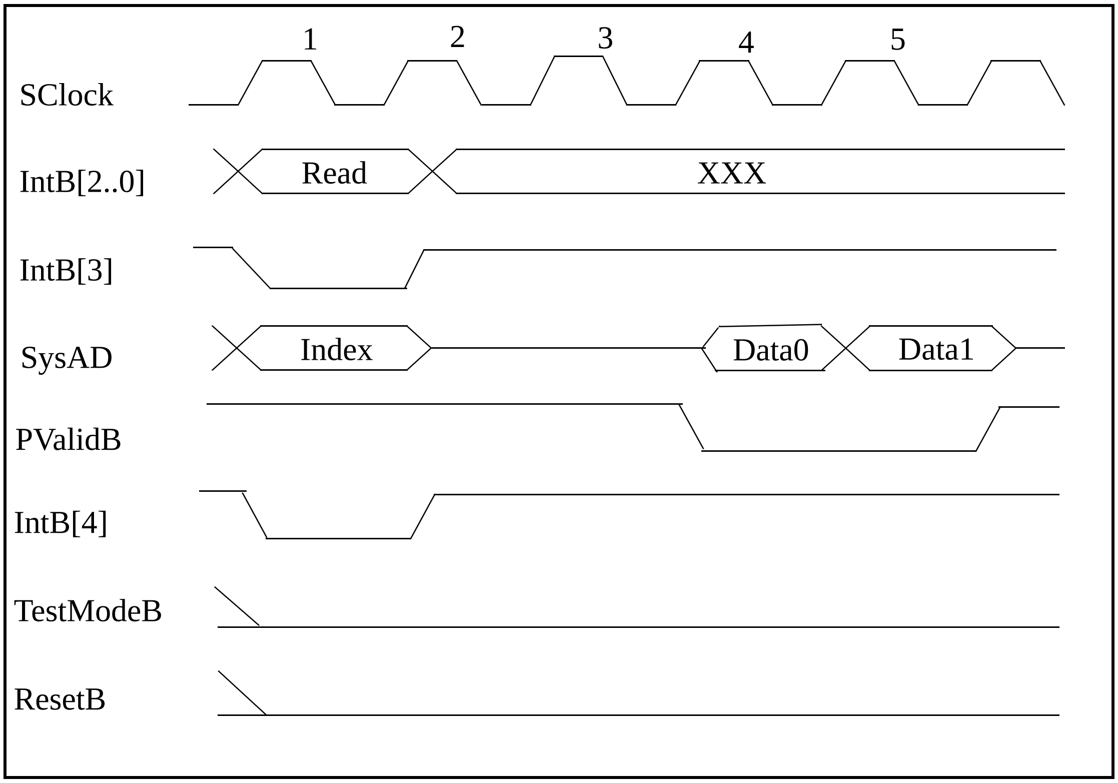
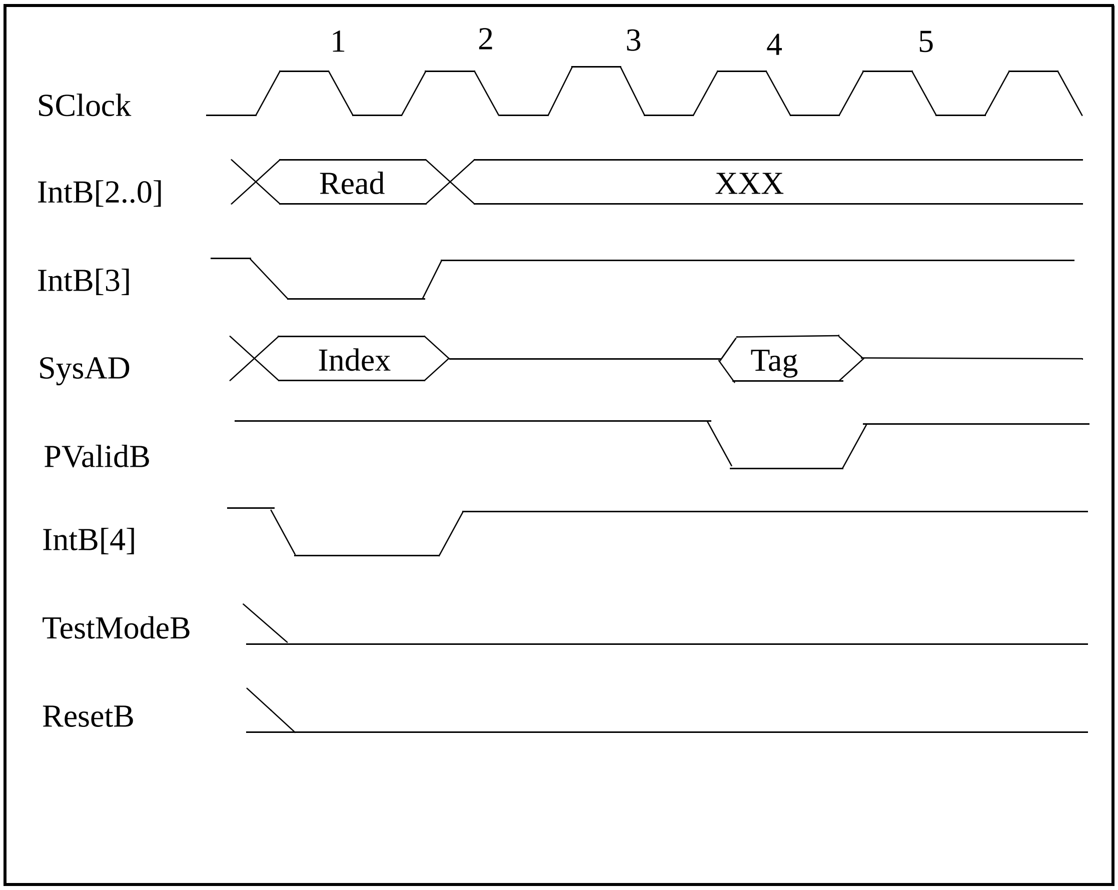


Figure 9: DCTag and ICTag Read Timing



6.6 Cache Write

A cache write consists of a write command on the IntB pins followed by write data on the next cycle. Pipelined writes are possible by latching an index once and then writing data to different cache elements using the same address.

When writing DCTag portion the SysAD bus should contain the following values:

- SysAD[31..12] 20 Tag Bits
- SysAD[11] Valid Bit
- SysAD[10] Dirty Bit
- SysAD[9..0] Unspecified

When writing ICTag portion the SysAD bus should contain the following values:

- SysAD[31..12] 20 Tag Bits
- SysAD[11] Valid Bit
- SysAD[10..0] Unspecified

The following diagrams illustrate cache write timing.

Figure 10: DCData and ICData Write Timing

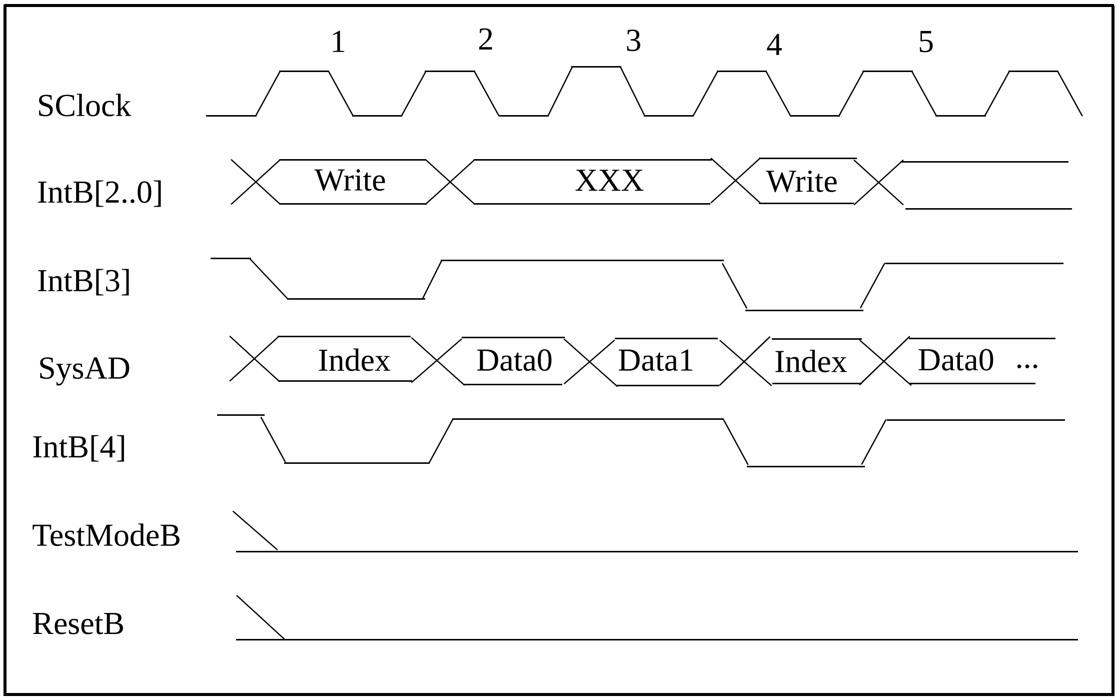


Figure 11: DCTag Write Timing

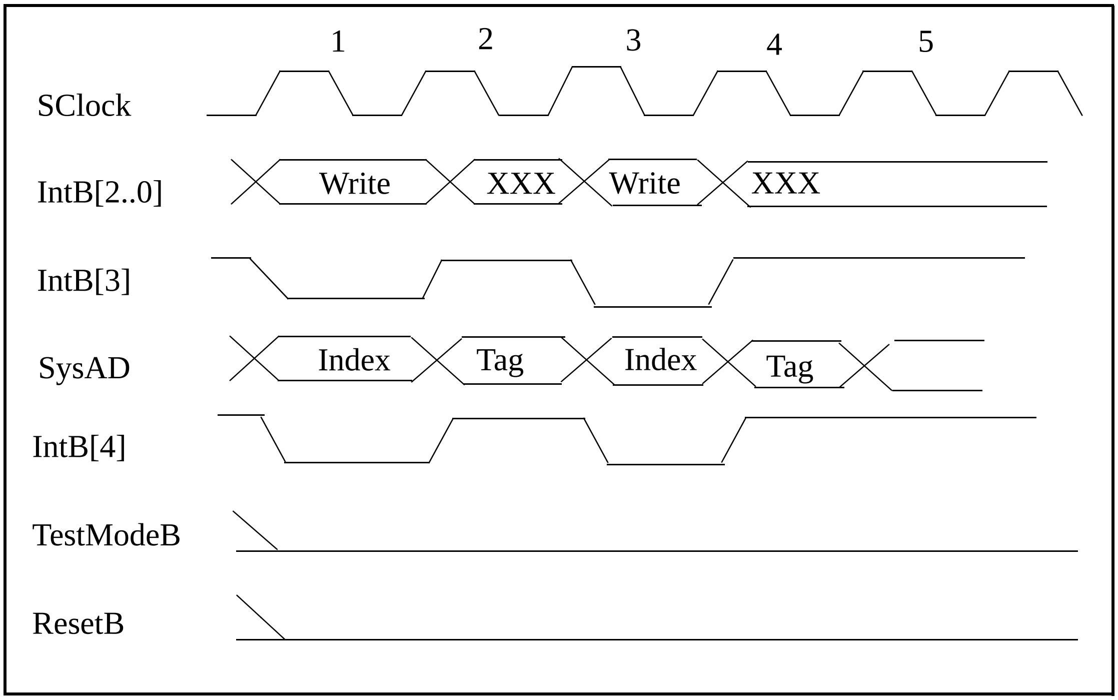
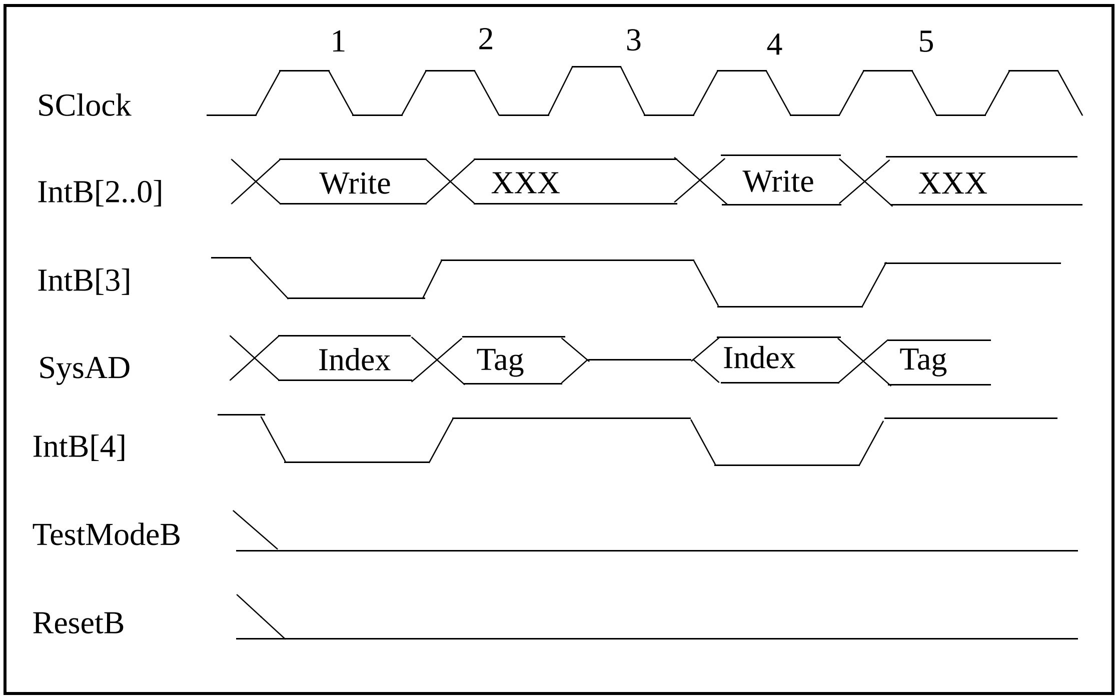
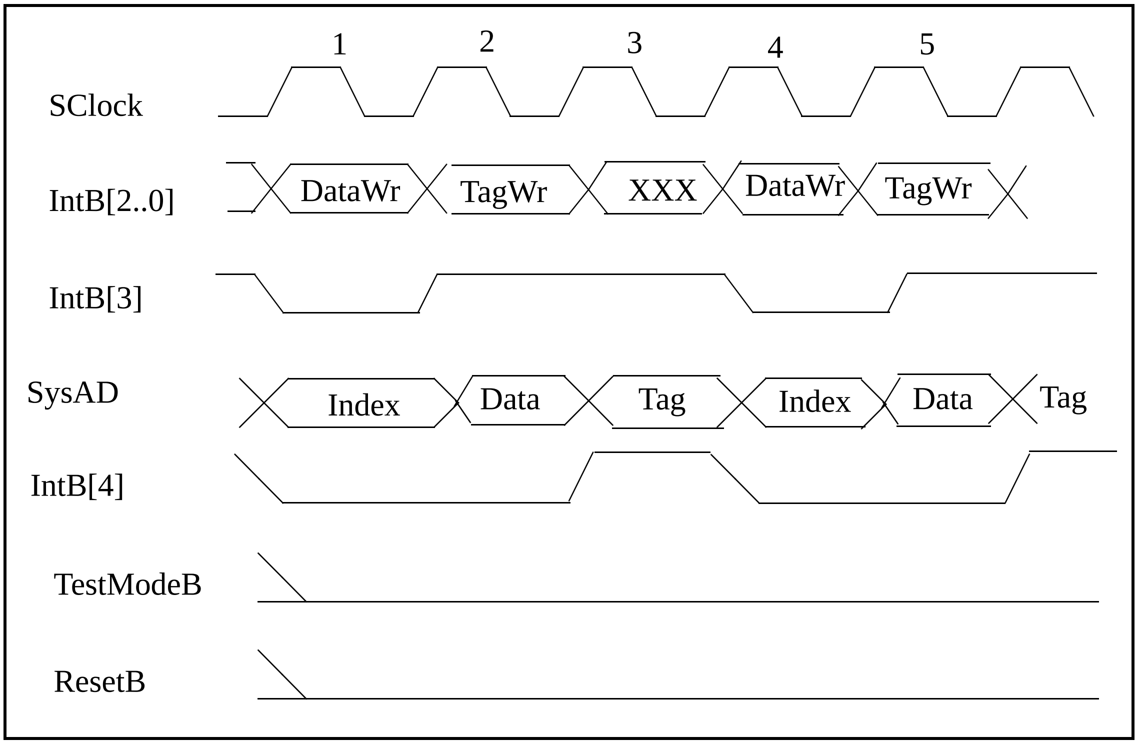


Figure 12: ICTag Write



The following figure illustrates a back to back write cycle. In this case the cache address is latched and held into the index register. The first write is to a cache data bank; the second write is to the cache tag bank. Note that the cache address in common for both writes.

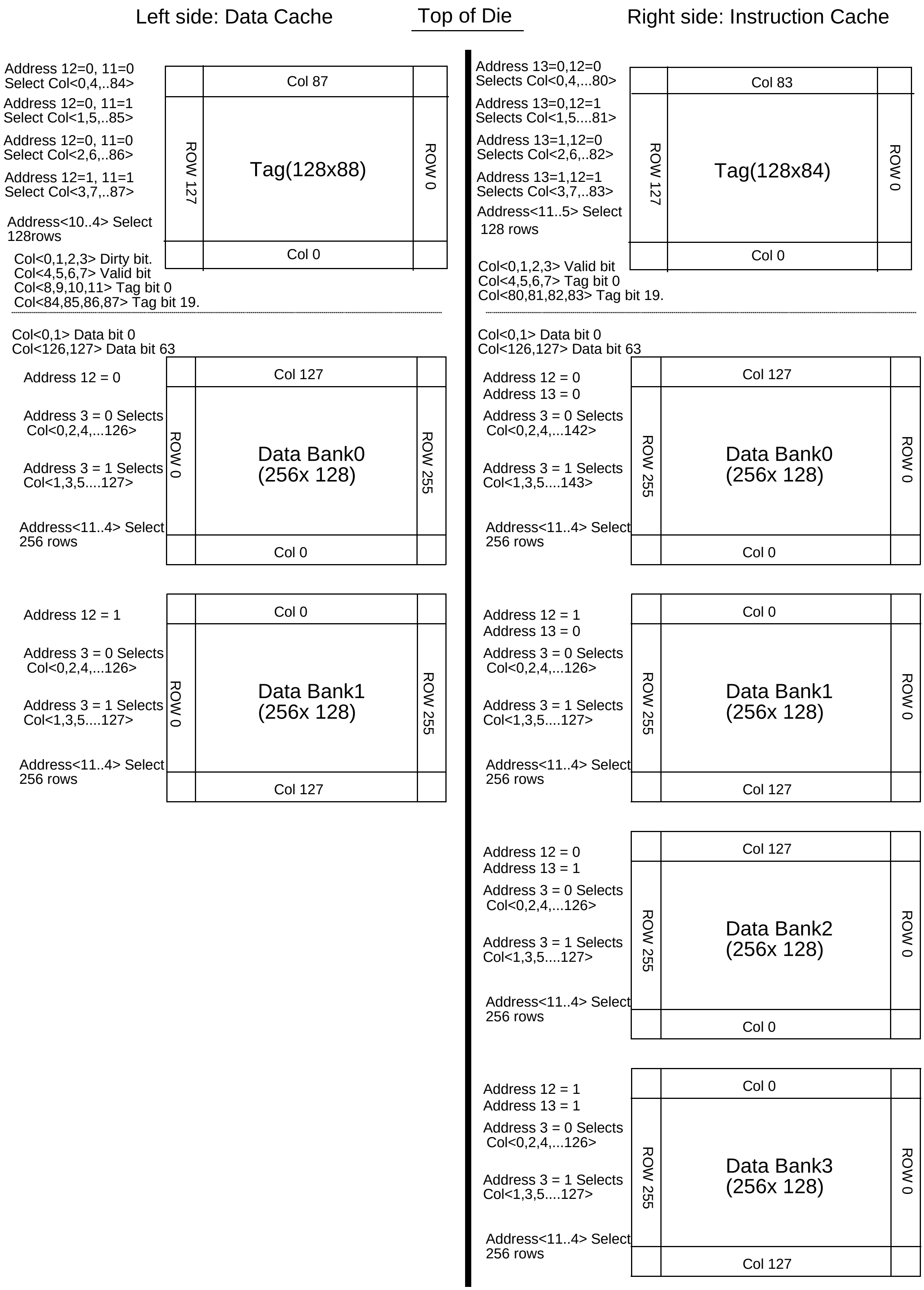
Figure 13: Instruction and Data Cache Back-to-Back Data/Tag Write Timing



6.7 Cache Organization

Two figures are provided below which illustrate a topological view and the formats of the instruction and data caches. The topological view also contains address decoding scheme for each memory portion.

Figure 14: Cache RAM Topological View



7.0 System Control Coprocessor (CP0)

CP0 registers and instructions provide access to the TLB and the caches. They also provide an ability to manipulate the modes in which the processor can operate. The facilities for handling interrupts and other exceptions are also controlled via CP0. Finally, any implementation dependent test or debug features are contained in CP0.

The CP0 registers on the R4300 are bit for bit compatible with the R4000, and can be written and read the same way. The effect of writing these registers differ slightly from the R4000, as noted in Appendix A. Differences from the R4000.

7.1 R4300 Control Coprocessor Registers

The following table lists all CP0 registers used on the R4300. Attempting to write any unused register is undefined and may have an unpredictable effect on the R4300 processor. Attempting to read any unused register is undefined and may result in unpredictable data.

Table 6: System Control Coprocessor CP0 register list

Number	Mnemonic	Description
0	Index	Programmable Pointer into TLB array
1	Random	Random Pointer into TLB array
2	EntryLo0	Low half of TLB entry for even VPN
3	EntryLo1	Low half of TLB entry for odd VPN
4	Context	Pointer to kernel PTE table
5	PageMask	TLB Page Mask
6	Wired	Number of wired TLB entries
7	-----	Unused
8	BadVAddr	Bad Virtual Address
9	Count	Timer Count
10	EntryHi	High half of TLB entry
11	Compare	Timer Compare
12	SR	Status Register
13	Cause	Cause of last exception
14	EPC	Exception Program Counter
15	PRId	Processor Revision Identifier
16	Config	Configuration Register
17	LLAddr	Load Linked Address
18	WatchLo	Memory Reference Trap address lower bits
19	WatchHi	Memory Reference Trap address upper bits
20	XContext	Context Register for MIPS III addressing
21-25	----	Unused
26	PErr	Parity error in cache
27	-----	Unused
28	TagLo	Cache Tag Register
29	TagHi	Cache Tag Register (Reserved)
30	ErrorEPC	Error Exception Program Counter
31	-----	Unused

7.1.1 Index Register (0)

The Index register is a 6 bit read/write register which specifies an entry into the on-chip TLB. The high order bit indicates the success or failure of a TLBP operation. The Index register is used to specify an entry in the TLB to be used by the TLBR and TLBWI instructions. Note the Index field contains six bits, but the R4300 only implements 32 entries. Thus, software should not load the Index register with a value greater than 31 and attempt a TLBR or TLBWR operation.

31	30 6	5 0
P	0	Index

where:

P Result of last Probe operation. Set to 1 if last TLB Probe instruction was unsuccessful.
index Index to entry in TLB.
0 Must be all zeroes on reads and writes.

7.1.2 Random Register (1)

The Random register is a read only register of 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 31. The TLB Random register is used to specify the entry in the TLB affected by the TLBWR instruction. Upon Cold reset, or when the Wired register is written, this register will be set to the upper limit.

31 6	5 0
0	Random

where:

Random Index to entry in TLB.
0 Must be zero on all reads and writes.

7.1.3 EntryLo0 Register (2)

The EntryLo0 Register is a read/write register that is used to access on-chip TLB. EntryLo0 is for even virtual pages. It is used by the TLBR, TLBWI, and TLBWR instructions. EntryLo0 contains the Page Frame Number, along with several configuration bits for the TLB entry.

32 bit mode:

31 30	29 6	5 4 3	2	1	0
0	PFN	C	D	V	G

64 bit mode:

63 30	29 6	5 4 3	2	1	0
0	PFN	C	D	V	G

where:

PFN	Page Frame Number	
C	Cache Algorithm	If C = 0 1 0, then the page is uncached. If C = 0 1 1, then the page is cached. Any other value (although undefined) defaults to the page being cached.
V	Page Valid if 1, Invalid if 0.	
G	If set in both Lo0 and Lo1, then ignore ASID.	
0	Must be zero on all reads and writes.	
D	Page Dirty if 1, Clean if 0.	

7.1.4 EntryLo1 Register (3)

The EntryLo1 Register is a read/write register that is used to access on-chip TLB. EntryLo1 is for odd virtual pages. It is used by the TLBR, TLBWI, and TLBWR instructions. EntryLo1 contains the Page Frame Number, along with several configuration bits for the TLB entry.

32 bit mode:

31 30	29 6	5 4 3	2	1	0
0	PFN	C	D	V	G

64 bit mode:

63 30	29 6	5 4 3	2	1	0
0	PFN	C	D	V	G

where:

PFN	Page Frame Number.	
C	Cache Algorithm	If C = 0 1 0, then the page is uncached. If C = 0 1 1, then the page is cached. Any other value (although undefined) defaults to the page being cached.
V	Page Valid if 1, Invalid if 0.	
G	If set in both Lo0 and Lo1, then ignore ASID.	
D	Page Dirty if 1, Clean if 0.	
0	Must be zero on all reads and writes.	

7.1.5 Context Register (4)

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 (bits 31..13) of the most recently translated 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 pages of 16M bytes, this value must be shifted and masked.

The PTEBase is a read / write field, and contains the base address of the PTE table of the current user address space.

32 bit mode:

31 23	22 4	3 0
PTEBase	BadVPN2	0

64 bit mode:

63 23	22 4	3 0
PTEBase	BadVPN2	0

where:

PTEBase Base address of the Page Entry Table.
BadVPN2 Virtual Page Number of the failed virtual address divided by two.
0 Must be zero on reads and write

7.1.6 PageMask Register (5)

The PageMask register is a read/write register that is used when reading or writing an on-chip TLB. The TLBR, TLBWI, 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 whether virtual address bits 24..13 participate in the comparison. This implements a variable page size on a per entry basis. R4300 implements 4K, 16K, 64K, 256K, 1M, 4M and 16M pages.

31 25	24 13	12 0
0	MASK	0

where:

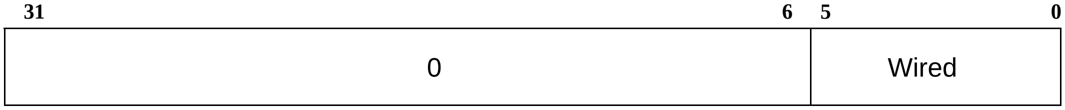
MASK Mask for Virtual Page Size. The following table gives MASK values for all seven page sizes. Any other value is undefined for R4300.
0 Must be zeroes on both read and write.

Table 1:

Page Size	MASK field bits											
	24	23	22	21	20	19	18	17	16	15	14	13
4K	0	0	0	0	0	0	0	0	0	0	0	0
16K	0	0	0	0	0	0	0	0	0	0	1	1
64K	0	0	0	0	0	0	0	0	1	1	1	1
256K	0	0	0	0	0	0	1	1	1	1	1	1
1M	0	0	0	0	1	1	1	1	1	1	1	1
4M	0	0	1	1	1	1	1	1	1	1	1	1
16M	1	1	1	1	1	1	1	1	1	1	1	1

7.1.7 Wired Register (6)

The TLB Wired register is a read/write register that specifies the boundary between the wired and random regions in the TLB. This register is set to 0 upon Cold Reset. Writing to this register also sets the Random register to 31. Writing a value greater than 31 to this register will result in undefined behavior. The TLB entry specified by the Wired field is not considered to be in the wired portion of the TLB.



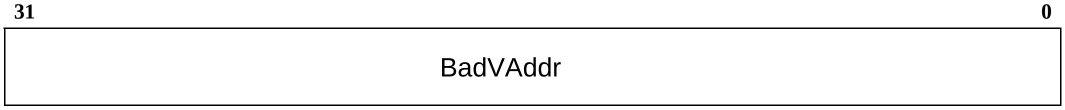
where:

Wired	TLB wired boundary.
0	Must be all zeroes on read and write.

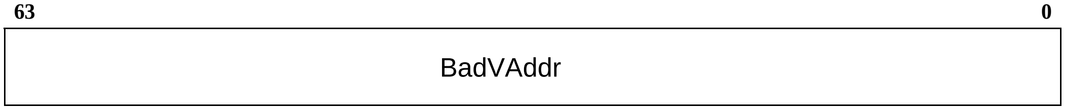
7.1.8 BadVAddr Register (8)

The Bad Virtual Address register is a read-only register that displays the most recently translated virtual address that failed to have a valid translation or that had an addressing error.

32 bit mode:



64 bit mode:

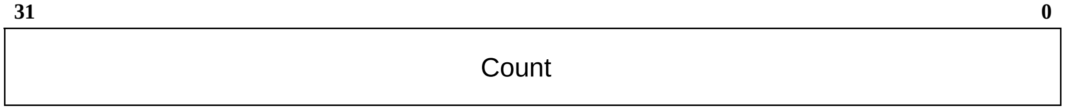


where:

BadVAddr	Most recently translated virtual address that failed to have a valid translation or that had an addressing error.

7.1.9 Count Register (9)

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 R4300, 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.



where:

Count	Current Count Value; updated at one-half PClock frequency.

7.1.10 EntryHi Register (10)

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.

32 bit mode:

31 13	12 8	7 0
VPN2	0	ASID

64 bit mode:

63 62	61 40	39 13	12 8	7 0
R	FILL	VPN2	0	ASID

where:

R	Region (00=user, 01=supervisor, 11=kernel) used to match $\text{VAddr}_{63..62}$
FILL	Reserved; undefined on read, should be 0 or -1 on write (should sign extend the virtual page number).
VPN2	Virtual Page Number divided by 2.
0	Must be zero on reads and writes.
ASID	Address Space Identifier.

7.1.11 Compare Register (11)

The Compare register is a read/write register. When the value of the Count Register equals the value of the Compare register, IP_7 of the Cause register is set. This causes an interrupt on the next execution cycle in which that interrupt is enabled. Writing to the Compare register will clear the timer interrupt.

31 0
Compare

where:

Compare	Value to be compared to Count register. An interrupt will be signaled when they are equal.

7.1.12 Status Register (12)

The status register is a read/write register that contains the various mode, enables, and status bits used on R4300. The contents of this register are undefined after a reset, except for TS, which is zero; ERL and BEV, which are one. The SR bit is 0 after a Cold Reset, and 1 after NMI or (Soft) Reset. Also the RP bit is set to 0 after a Cold Reset.

31 28	27	26	25	24	23	22	21	20	19	18	17	16
CU	RP	FR	RE	ITS	rsvd	BEV	TS	SR	0	CH	CE	DE

15 8	7	6	5	4 3	2	1	0
IM	KX	SX	UX	KSU	ERL	EXL	IE

where:

CU Coprocessor Unit usable if 1. CP0 is always usable by the kernel.

RP Reduced Power. Changes clock to quarter speed.

FR If set, enables MIPS III additional floating point registers.

RE Reverse Endian. Changes endianess in user mode.

ITS Instruction Trace Support. Enables trace support.

rsvd Reserved for future use. Read and write as zero.

BEV Controls location of TLB refill and general exception vectors. (0=normal, 1= bootstrap).

TS A TLB Shutdown condition has occurred. This bit is set whenever a multiple TLB hit has occurred (i.e. more then one entry in the TLB matches during a TLB access).

SR Soft Reset or NMI has occurred.

0 Must be zeroes on read and write.

CH This is the CP0 Condition bit. It is readable and writable by software only. It is not set or cleared by hardware.

CE, DE These fields are used only for compatibility reasons with the R4200 and are not used by the R4300 hardware.

IM Interrupt Mask. Enables and disables interrupts.

KX Kernel extended addressing enabled.

SX Supervisor extended addressing enabled.

UX User extended addressing enabled.

KSU Mode (10=user, 01=supervisor, 00=kernel).

ERL Error Level. Normal when zero, error if one.

EXL Exception Level. Normal when zero, exception if one.

IE Interrupt Enable.

7.1.13 Cause Register (13)

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 instruction that took an exception.

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 Count register equals the Compare 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.

31	30	29 28	27 16	15 8	7	6 2	1 0
BD	0	CE	0	IP	0	ExcCode	0

where:

BD	Branch Delay
CE	Coprocessor Error
IP	Interrupt pending
ExcCode	Exception Code
0	Must be zeroes on read and write

Exception Codes are:

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
14	---	Reserved
15	FPE	Floating Point Exception
16-22		Reserved
23	Watch	Reference to WatchHi/WatchLo address
24-31		Reserved

7.1.14 EPC (14)

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.

32 bit mode:

31	0
EPC	

64 bit mode:

63	0
EPC	

where:

EPC	Exception Program Counter.

7.1.15 Processor Revision Identifier (15)

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, MTI 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.

The Implementation number for R4300 is 0x0B.

31 16	15 8	7 0
0	Imp	Rev

where:

Imp	Implementation identifier.
Rev	Revision Number.
0	Returns zeroes on reads.

7.1.16 Configuration Register (16)

The Config register specifies various configuration options that are available for R4300. It is compatible with the R4000 Config register, but only a subset of the options available on the R4000 are possible on R4300. For that reason, there are many fields which are set to constant values.

The EP and BE fields are written to their default values by hardware during Cold Reset. These fields are also readable and writable by software. The default values upon Cold Reset are as follows: EP=0000, BE=1. 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 (Cold or Warm).

The EP and BE are expected to be changed only during processor initialization (done in uncached space before any stores). If there are any changes to these values during normal operation, correct behavior can not be guaranteed.

31	30 28	27 24	23 20	19 18	17	16	15	14	13	12	11 9	8 6	5	4	3	2 0
0	EC	EP	000	0 1	1	0	BE	1	1	0	0 1 0	0 0 1	1	0	CU	K0

where:

EC System clock ratio, read only
110 -> 1 : 1
111 -> 1.5 : 1
000 -> 2 : 1
001 -> 3 : 1
All other values of EC are undefined.
(Note: The system clock ratio encoding for the DivMode pins are not the same as those shown above. For DivMode pin encoding, see “8.3 Signal Descriptions“)

EP Pattern for writeback data on SYSAD port
0000 -> D
0110 -> DxxDxx
All other values of EP are undefined.

BE BigEndian.
0 -> Memory and kernel are Little Endian.
1 -> Memory and kernel are Big Endian.

CU Reserved (Readable and Writable by software).

K0 Kseg0 coherency algorithm.
This has the same format as the C field in EntryLo0 and EntryLo1.
If K0 = 0 1 0, then the region is uncached.
If K0 = 0 1 1, then the region is cached.
Any other value (although undefined) defaults to the region being cached.

0 Returns 0 on read.

1 Returns 1 on read.

7.1.17 Load Linked Address (LLAddr) Register (17)

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.

31 0
PAddr

where:

Paddr Bits 35..4 of the physical address. Note that for R4300, the MSB of the physical address is only 31. A Load Linked instruction will write 0 into bits [31..28] of LLAddr, but a software write can set all 32 bits to any value. This maintains software compatibility with R4000.

7.1.18 WatchLo (18)

R4300 processor provide 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.

31 3	2	1	0
PAddr	0	R	W

where:

PAddr Physical address bits 31..3.
R Trap on Read Access if 1.
W Trap on Write Access if 1.
0 Must be zeroes on reads and writes.

7.1.19 WatchHi (19)

R4300 processors provide 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.

31	3 0
0	PAddr

where:

PAddr Physical address bits [35..32]. Note that for R4300, the MSB of the physical address is only 31. So all of the bits of PAddr are ignored. However all 4 bits of PAddr are software writable. This maintains software compatibility with R4000.
0 Must be zeroes on reads and writes.

7.1.20 XContext Register (20)

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 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 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 page sizes other than 4K bytes, 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.

63 33	32 31	30 4	3 0
PTEBase	R	BadVPN2	0

where:

PTEBase Base address of the Page Entry Table.
BadVPN2 Virtual Page Number of the failed virtual address divided by two.
R Region (00=user, 01=supervisor, 11=kernel).
0 Must be zero on reads and writes.

7.1.21 PErr Register (26)

The PErr register exists only for compatibility reasons with R4000 / R4200. Since R4300 does not implement parity on caches, this register is not used by hardware. It is software readable and writable.

31 8	7 0
0	DIAGNOSTIC

where:

Diagnostic	Eight bit diagnostic field
0	Must be zeroes on all reads and writes

7.1.22 CacheErr Register (27)

The CacheErr register exists only for compatibility reasons with R4000 / R4200. Since R4300 does not implement cache errors, this register is not used by hardware. It is a read only register that returns 0 when read.

31 0
0

7.1.23 TagLo (28) and TagHi (29)

The TagLo register is a 32-bit read/write register used to hold the cache tag information for cache instructions. Cache Store Tag instructions write the data from TagLo into cache and Cache Load Tag instructions read the data from cache into TagLo register.

The TagHi register is reserved for future use.

The Tag registers are written by the CACHE and MTC0 instructions.

TagLo

31 28	27 8	7 6	5 0
0	PTagLo	Pstate	0

TagHi

31 0
0

where:

PTagLo	Physical address bits 31..12
Pstate	Primary Cache State (0=invalid, 3=Dirty)
0	Must be zeroes on reads and writes

7.1.24 ErrorEPC (30)

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

32 bit mode:

<table>
<tr><th align="left">31</th><th align="right">0</th></tr>
<tr><td colspan="2" align="center">ErrorEPC</td></tr>
</table>

64 bit mode:

<table>
<tr><th align="left">63</th><th align="right">0</th></tr>
<tr><td colspan="2" align="center">ErrorEPC</td></tr>
</table>

where:

ErrorEPC	Exception Program Counter for Reset and NMI.

7.2 CP0 Instructions

Coprocessor zero instructions on R4300 will perform the same basic functions as they did on the R4000. The encoding will be identical for R4300.The only difference is in the implementation of the Cache Ops. Please refer to the R-Series Architecture manual for details on the encoding and function of the operations. Note that on R4300 like the R4000, the ERET instruction is supported rather than the RFE instruction.

Table 7: CP0 Instructions

MTC0	Move To Coprocessor 0 MTC0 moves 32 bit sign extended contents of a general register to a Coprocessor 0 register.
MFC0	Move From Coprocessor 0 MFC0 moves 32 bit sign extended contents of a Coprocessor 0 register to a general register.
DMTC0	Double Move To Coprocessor0 DMTC0 moves 64 bit contents of a general register to a 64 bit Coprocessor 0 register. This operation is undefined for 32 bit Coprocessor 0 registers.
DMFC0	Double Move From Coprocessor 0 DMF0 moves contents of a 64 bit Coprocessor register to a general register. This operation is undefined for 32 bit Coprocessor registers.
ERET	Exception Return ERET returns from an exception. Unlike a branch or jump, ERET does not execute the next instruction. The PC will be loaded from the ErrorEPC register if the processor is servicing an error trap (ERL is set), otherwise the PC is loaded from the EPC register.
TLBR	TLB Read TLBR reads a TLB entry indexed by the Index CP0 register. PageMask, EntryHi, EntryLo0 and EntryLo1 registers are loaded with the contents of the TLB.
TLBWI	TLB Write Indexed TLBWI writes a TLB entry indexed by the Index CP0 register. TLB is loaded with the contents of the PageMask, EntryHi, EntryLo0 and EntryLo1 registers.
TLBWR	TLB Write Random TLBWR writes a TLB entry indexed by the Random CP0 register. TLB is loaded with the contents of the PageMask, EntryHi, EntryLo0 and EntryLo1 registers.
TLBP	TLB Probe TLBP probes the TPB for an entry that matches the virtual address specified in the EntryHi register. The Index register is loaded with the address of the TLB entry. If no TLB entry matches, the high-order bit of the Index register is set.

7.2.1 CACHE - Cache Operations

These operations are used to access and manipulate the caches. They are documented in “Table 5: Cache Operations” in the in chapter “5.0 Data and Instruction Caches” of this document, as well as in the R-Series Architecture document.

7.3 R4300 32 bit Virtual Address Space

The following table describes the virtual address space when R4300 operates in 32-bit (MIPS II) mode. Each of the three privilege modes, i.e. Kernel, Supervisor and User, has different access rights to the various segments listed below.

Figure 15: R4300 32-bit Address Space

0xffffffff 0xe0000000	Kernel virtual address space (kseg3) Mapped, 0.5 GB
0xdfffffff 0xc0000000	Supervisor virtual address space (ksseg) Mapped, 0.5 GB
0xbfffffff 0xa0000000	Uncached kernel physical address space (kseg1) Unmapped, 0.5 GB
0x9fffffff 0x80000000	Cached kernel physical address space (kseg0) Unmapped, 0.5 GB
0x7fffffff 0x00000000	User virtual address space (kuseg) Mapped, 2 GB

7.4 R4300 64 bit Virtual Address Space

The following table describes the virtual address space when R4300 operates in 64-bit (MIPS III) mode. Each of the three privilege modes, i.e. Kernel, Supervisor and User, has different access rights to the various segments listed below

Figure 16: R4300 64-bit Address Space

0xffffffff ffffffff 0xffffffff e0000000	Kernel virtual address space (kseg3) Mapped, 0.5 GB
0xffffffff dfffffff 0xffffffff c0000000	Supervisor MIPS II virtual address space (ksseg) Mapped, 0.5 GB
0xffffffff bfffffff 0xffffffff a0000000	Uncached kernel physical address space (kseg1) Unmapped, 0.5 GB
0xffffffff 9fffffff 0xffffffff 80000000	Cached kernel physical address space (kseg0) Unmapped, 0.5 GB
0xffffffff 7fffffff 0xc00000ff 80000000	Address error
0xc00000ff 7fffffff 0xc0000000 00000000	Kernel virtual address space (xkseg) Mapped
0xbfffffff ffffffff 0x80000000 00000000	Kernel physical xkphys space (see below) Unmapped
0x7fffffff ffffffff 0x40000100 00000000	Address error
0x400000ff ffffffff 0x40000000 00000000	Supervisor MIPS III virtual address space (xksseg) Mapped
0x3fffffff ffffffff 0x00000100 00000000	Address error
0x000000ff ffffffff 0x00000000 00000000	User virtual address space (xkuseg) Mapped

Figure 15: R4300 xkphys region detail

0xbfffffff ffffffff 0xb8000001 00000000	Address error
0xb8000000 ffffffff 0xb8000000 00000000	Cache/Coherency Algorithm 7 (Cached) Unmapped, 4 GB
0xb7ffffff ffffffff 0xb0000001 00000000	Address error
0xb0000000 ffffffff 0xb0000000 00000000	Cache/Coherency Algorithm 6 (Cached) Unmapped, 4 GB
0xafffffff ffffffff 0xa8000001 00000000	Address error
0xa8000000 ffffffff 0xa8000000 00000000	Cache/Coherency Algorithm 5 (Cached) Unmapped, 4 GB
0xa7ffffff ffffffff 0xa0000001 00000000	Address error
0xa0000000 ffffffff 0xa0000000 00000000	Cache/Coherency Algorithm 4 (Cached) Unmapped, 4 GB
0x9fffffff ffffffff 0x98000001 00000000	Address error
0x98000000 ffffffff 0x98000000 00000000	Cache/Coherency Algorithm 3 (Cached) Unmapped, 4 GB
0x97ffffff ffffffff 0x90000001 00000000	Address error
0x90000000 ffffffff 0x90000000 00000000	Cache/Coherency Algorithm 2 (Uncached) Unmapped, 4 GB
0x8fffffff ffffffff 0x88000001 00000000	Address error
0x88000000 ffffffff 0x88000000 00000000	Cache/Coherency Algorithm 1 (Cached) Unmapped, 4 GB
0x87ffffff ffffffff 0x80000001 00000000	Address error
0x80000000 ffffffff 0x80000000 00000000	Cache/Coherency Algorithm 0 (Cached) Unmapped, 4 GB

7.5 Translation Lookaside Buffer

Mapped virtual addresses are translated into physical addresses using a translation lookaside buffer (TLB). R4300 implements a fully associative on-chip TLB. This TLB holds both instruction and data pages, and is thus also referred to as the Joint TLB (JTLB). The 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 on R4300 can be 4K, 16K, 64K, 256K, 1M, 4M, or 16M bytes. The page size 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 are 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 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 EntryHi register) matches the ASID field of the TLB entry. While 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 (C, 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 each TLB entry in 64 bit addressing mode is as follows:

255 217	216 205	204 192
-	MASK	-

191 190	189 168	167 141	140	139 136	135 128
R	-	VPN2	G	-	ASID

127 90	89 70	69 68 67	66	65	64
-	PFN	C	D	V	-

63 26	25 6	5 4 3	2	1	0
-	PFN	C	D	V	-

where:

MASK	Comparison Mask (Determines Page Size).
R	Region used to match $VAddr_{63..62}$
VPN2	Virtual Page Number / 2
ASID	Address Space Identifier.
PFN	Page Frame Number (upper 20 bits of physical address).
C	Cache Algorithm If C = 0 1 0, then the page is uncached. If C = 0 1 1, then the page is cached. Any other value (although undefined) defaults to the page being cached.
D	if set, page is dirty (writable).
V	if set, entry is valid.

G if set, page is global when G bits in EntryLo1 & EntryLo0 are set.
- these bits are not stored in the TLB.

The format of each TLB entry in 32 bit addressing mode is as follows:

127 121	120 109	108 96
-	MASK	-

95 77	76	75 72	71 64
VPN2	G	-	ASID

63 58	57 38	37 36 35	34	33	32
-	PFN	C	D	V	-

31 26	25 6	5 4 3	2	1	0
-	PFN	C	D	V	-

where:

MASK Comparison Mask (Determines Page Size).
VPN2 Virtual Page Number / 2
ASID Address Space Identifier
PFN Page Frame Number (upper 20 bits of physical address).
C Cache Algorithm If C = 0 1 0, then the page is uncached.
If C = 0 1 1, then the page is cached.
Any other value (although undefined) defaults to the page being cached.
D if set, page is dirty (writable).
V if set, entry is valid.
G if set, page is global when G bits in EntryLo1 & EntryLo0 are set.
- these bits are not stored in the TLB.

7.5.1 Instruction Micro TLB

The thirty-two double entry TLB described above is a combined TLB that is visible to software. R4300 will also implement a two entry “micro” TLB that is dedicated to instructions. This can be thought of as the primary TLB for instructions. Instructions can access this TLB simultaneously with data accessing the large combined TLB. If there is a miss in the microTLB, there will be a pipeline stall while the new TLB entry is transferred from the combined TLB to the micro TLB. The two entry microTLB is fully associative with a Least Recently Used replacement algorithm. Each micro-TLB entry maps 4KB only. On the R4300, it is guaranteed that the microTLB is always a subset of the software visible TLB.

7.6 R4300 Processor Modes

The R4300 processor will support several user selectable modes. All except ByPass PLL and MasterClock to PClock ratios (DivMode) are set and reset by writing to the Processor Status register (CP0 reg12) and the Configuration Register (CP0 reg16). ByPass PLL and DivMode modes are documented in other sections.

7.6.1 Reduced Power Mode RP (bit 27 in Status Register)

The R4300 processor normally operates in a mode where the processor clock (PClock) operates at a multiple of the Master Clock speed (3, 2, 1.5 or 1 selected by DivMode pins). The System Interface Clock (SClock) operates at the same frequency as Master Clock.

The user may set the Reduced Power mode (RP) via a move to the Status register that sets the RP bit. Upon setting this mode, the processor modifies its clocking scheme to slow the PClock to one fourth of its normal frequency (i.e. 3/4, 2/4 1.5/4 or 1/4 of the Master Clock speed). SClock and TClock will also be derived as one fourth of their normal frequency, (i.e. 1/4 of the Master Clock speed). The clocks will switch frequency within 1 to 16 Master Clock cycles after the move to 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 cold reset.

Software must be careful to execute a code sequence that will guarantee the proper operation of the system upon setting or clearing the *RP* bit. Software must first write any registers on the external agents that must be changed to accommodate the change in frequency (e.g. DRAM refresh counters). Next software must guarantee that the system interface is in an inactive state. This can be accomplished by an uncached read, which will guarantee the flush buffer is empty upon completion of the read. Only then can software attempt to execute the instruction to set or clear the *RP* bit. Finally, software should guarantee that at least the eight instructions immediately preceding and following the Move to Coprocessor register will not cause a cache miss, TLB miss, or exception of any kind.

7.6.2 Floating-Point Registers FR (bit 26 in Status Register)

This 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.

7.6.3 Data Rate EP (bits 27..24 in Configuration Register)

R4300 write data rate can be changed between “D” and “Dxx” modes under software control. It is recommended to change it right after power up during processor initialization before any write operations. This mode is set to “D” on Cold Reset.

7.6.4 System Endianness BE (bit 15 in Configuration Register)

The system endianness information is provided to R4300 through the software writable BE bit. It is recommended to change it right after power up during processor initialization before any non-word memory access operations. This mode is set to BigEndian on Cold Reset.

7.6.5 Reverse Endianness RE (bit 25 in Status Register)

When set, reverses the endianness for user software. This functionality is the same as the R4000.

7.6.6 Instruction Trace Support Mode ITS (bit 24 in Status Register)

R4300 will support a mode to allow the user to track branches or jumps. This mode is set by setting the ITS bit in the Status register. It can be disabled by resetting the ITS bit. This is new functionality not implemented in the R4000.

When the ITS bit is set, the CPU reports all change of flow conditions on the SysAD bus by forcing an instruction cache miss whenever a branch, jump or exception is taken.

7.6.7 Bootstrap Exception Vector BEV (bit 22 in Status Register)

The Bootstrap Exception Vectors (BEV) bit in the Status Register, when set, causes the base exception vector to be located at virtual address of 0xbfc00000. When cleared, this base exception vector is located at 0x80000000. This bit is used when diagnostic tests cause exceptions to occur prior to verifying proper operation of the cache and main memory system.

7.6.8 Kernel eXtended addressing KX (bit 7 in Status Register)

If the KX bit is set, the processor will use the extended addressing TLB refill exception vector for TLB misses on kernel addresses.

7.6.9 Supervisor eXtended addressing SX (bit 6 in 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.

7.6.10 User eXtended addressing UX (bit 5 in 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.

7.6.11 Interrupt Enable IE (bit 0 in Status Register)

When clear, will not allow interrupts with the exception of reset and non-maskable interrupt.

7.7 Processor Interrupts

There are four variations of interrupt available on R4300. These are the non-maskable interrupt, NMI; the external interrupts; software interrupts; and the timer interrupt.

The non-maskable interrupt is signaled by asserting the NMI* pin. It also may be set by an External Write via the SysAD bus. On the data cycle, SysAD[22] is the write enable for SysAD[6], which is the value to be written as the interrupt. As the name implies, this interrupt cannot be masked. An NMI will force a jump to the Reset exception vector.

External interrupts are set by asserting the external interrupt pins Int[4..0]*. These pins will set bits [14..10] of the Cause register (IP). They may be set by an External Write via the SysAD bus. On the data cycle, SysAD[20..16] are the write enables for bits SysAD[4..0], which are the values to be written as interrupts. These interrupts may be masked with the IM field of the Status register.

Software interrupts use bits 9 and 8 of the Cause register, which are bits 1 and 0 of the Interrupt Pending field within the register. These may be written by software, but there is no hardware mechanism to set or clear these bits. These interrupts are maskable.

The timer interrupt is bit 15 of the Cause register, which is bit 7 of the Interrupt Pending field. It will be set whenever the value of the Count register equals the value of the Compare register. This interrupt is maskable via the Interrupt Mask field of the Status register.

7.8 Coprocessor 0 Hazards

For compatibility reasons, the R4300 will maintain the same hazards as the R4000. The following description is for the R4000. Though the pipe stages are different, the hazards will remain the same. Any violation of these hazards will have unpredictable results.

7.8.1 R4000 Hazards

The contents of the System Coprocessor registers and the TLB affect the operation of the processor in many ways. For instance, an instruction that changes CP0 data also affects subsequent instructions that use the data.

In the CPU, general registers are interlocked and the result of an instruction can generally be used by the next instruction; if the result is not available right away, the processor stalls until it is available. CP0 registers and the TLB are not interlocked, however; there may be some delay before a value written by one instruction is available to following instructions.

There is a *required-data dependence* between an instruction that changes a register or TLB entry (a *writer*) and the next instruction that uses it (a *user*). (A writer can write multiple data items, forming multiple writer/user pairs.) The writer/user instruction pair places a *hazard* on the data if there must be a delay between the time the writer instruction writes the data, and the user instruction can use the data.

In addition to instructions, events can be writers and users of CP0 information. For instance, an exception writes information to CP0 registers and events that occur for every instruction, like an instruction fetch, use CP0 information. Therefore, when manipulating CP0 contents, the systems programmer must identify hazards and write code that avoids these hazards.

Table 0-1 describes how to identify and avoid hazards, listing instructions and events that use CP0 registers and the TLB. This table also tells when written information is available (column 3) and when this latest information can actually be used (column 2). *Exception event writer timing* refers to the instruction identified with the exception; *user event timing* information is the pipestage of each instruction during which the user event uses the data. In the case of a hazard, the number of instructions required between a writer and user is:

available_stage - (use_stage + 1)

To identify a hazard, look for an instruction/event writer/user pair that has a required-data dependence and use the timing information in the table to calculate the delay required between the writer and user. If no delay is required, there is no hazard. If there is a hazard, place enough instructions between the writer and user so that the written information is available or effective when the user needs it.

NOTE: Any instructions inserted between a writer / reader pair with a hazard must not depend on or modify the data creating the hazard (for example NOP instructions may be used).

The following steps are used to determine a hazard delay:

1. Find the pipeline stage of the *writer* instruction in which the result is available. For example, the MTC0 instruction writes a CP0 general register, and the new value is available at stage 7.
2. Find the pipeline stage in which the *user* instruction reads or uses the data item that the writer changes. The TLBWR instruction, for example, uses different registers through different stages; all source register values must be stable by stage 5 and remain unchanged through stage 8.

3. Calculate the number of instructions that must be inserted between the hazardous pair, by using this formula: *available_stage - (use_stage + 1)*. For example, with an MTC0/TLBWR pair, MTC0 data is available at stage 7, and TLBWR data must be stable by stage 5 so the computation is: 7 - (5 + 1) = 1. This means 1 instruction must be inserted between the MTC0 and TLBWR. If the result of the computation is less than or equal to zero, there is no hazard and no instructions are required between the pair.

Table 0-1 R4000 Coprocessor 0 Data Writer and User Timing

Instruction or Event	**CP0 Data Used, Stage Used**		**CP0 Data Written, Stage Available**	
MTC0 / DMTC0			CPR[0,rd]	7γδ
MFC0 / DMFC0	CPR[0,rd]	4βγ		
TLBR	Index, TLB	5-7	PageMask, EntryHi, EntryLo0, EntryLo1	8
TLBWI TLBWR	Index or Random, PageMask, EntryHi, EntryLo0, EntryLo1	5-8	TLB	8
TLBP	PageMask, EntryHi	3-6	Index	7
ERET	EPC or ErrorEPC, TLB	4	Status[EXL, ERL]	4-8α
	Status	3	LLbit	7
Index Load Tag			TagLo, TagHi, ECC	8βε
Index Store Tag	TagLo, TagHi, ECC	8ε		
CACHE Hit ops			Status[CH]	8ε
CACHE ops	cache line (see note)	ε	cache line (see note)	ε
Load/Store	EntryHi.ASID Status[KSU, EXL, ERL, RE], Config[K0, DB], TLB	4		
	Config[SB]	7		
	WatchHi, WatchLo	4-5		
Load/Store exception			EPC, Status, Cause, BadVaddr, Context, XContext	8
Instruction fetch exception			EPC, Status	8
			Cause, BadVAddr, Context, XContext	4
Instruction fetch	EntryHi[ASID], Status[KSU, EXL, ERL, RE], Config[K0, IB]	0α		
	Config.SB	3		
	TLB (mapped addresses)	2		
Coproc. usable test	Status[CU, KSU, EXL, ERL]	2		
Interrupt signals sampled	Cause[IP], Status[IM, IE, EXL, ERL]	3		
TLB shutdown			Status.TS	7

EntryHi.ASID refers to the *ASID* field of the *EntryHi* register.
Config[K0, DB] refers to the *K0* and *DB* fields of the *Config* register.

α The *EXL* and *ERL* bits in the *Status* register are permanently cleared in stage 8, if no exceptions abort the ERET. However the effect of clearing them is visible to an instruction fetch starting in stage 4, so the “returned to” instructions use the modified values in the *Status* register.

β Only one instruction is needed to separate Index Load Tag and MFC0 Tag, even though table timing indicates otherwise.

γ An MTC0 of a CPR must not be immediately followed by MFC0 of the same CPR.

δ With an MTC0 to *Status* that modifies *KSU* and sets *EXL* or *ERL*, it is possible for the five instructions following the MTC0 to be executed incorrectly in the new mode, and not correctly in the kernel mode. This can be avoided by setting *EXL* first, and only later changing the value of *KSU*.

ε There must be two non-load, non-CACHE instructions between a store and a CACHE instruction directed to the same primary cache line as the store.

Table 0-2 lists some hazard conditions, and the number of instructions that must come between the writer and the user. The table shows the data item that creates the hazard, and the calculation for the required number of intervening instructions.

Table 0-2 CP0 Hazards and Calculated Delay Times.

Writer	→	User	Hazard On	Instructions Between	Calculation
TLBWR/ TLBWI	→	TLBP	TLB entry	3	8-(4+1)
TLBWR/ TLBWI	→	load/store using new TLB entry	TLB entry	3	8-(4+1)
TLBWR/ TLBWI	→	I-fetch using new TLB entry	TLB entry	5	8-(2+1)
MTCO Status[CU]	→	Coprocessor instruction needs CU set	Status[CU]	4	7-(2+1)
TLBR	→	MFC0 EntryHi	EntryHi	3	8-(4+1)
MTC0 EntryLo0	→	TLBWR/TLBWI	EntryLo0	1	7-(5+1)
TLBP	→	MFC0 Index	Index	2	7-(4+1)
MTC0 EntryHi	→	TLBP	EntryHi	1	7-(5+1)
MTC0 EPC	→	ERET	EPC	2	7-(4+1)
MTC0 Status	→	ERET	Status	3	7-(3+1)
MTC0 Status[IE]	→	instruction interrupted[a]	Status[IE]	3	7-(3+1)

[a] You cannot depend on a delay in effect if the instruction execution order is changed by exceptions. In this case, for example, the minimum delay for IE to be effective is the *maximum* delay before a pending, enabled interrupt can occur.

7.8.2 R4300 Specific Hazards

The R4300 Status register includes the ITS bit, which is not included in the R4000. However, the hazard table still covers this situation. From the table, the Status register would be written in stage 7. The ITS bit can affect the instruction fetch of any taken branch. Instruction fetches are considered to be used at stage 0. Thus there is a hazard of 6 instructions from writing the Status register to any taken branch or jump..

8.0 System Interface

An event that occurs within the processor that requires access to external system resources is referred to as a *system event*. System events include: a fetch that misses in the instruction cache, a load that misses in the data cache, a store that misses in the data cache, an uncached load or store, and actions resulting from the execution of cache instructions.

When a system event occurs the processor issues a request or a series of requests through the system interface to access some external resource to service the event. The system interface must be connected to an external agent that coordinates access to system resources.

Processor requests include read requests, which provide an address to an external agent, and write requests, which provide an address and a word or block of data to be written to an external agent. External Requests include read responses, which provide a block or single transfer of data from an external agent in response to read requests, and write requests, which provide an address and a word of data to be written to a processor resource.

When an external agent receives a read request, it accesses the specified resource and returns the requested data via a read response, which may be returned any time after the read request and at any rate. Processor read requests that have been issued but for which data has not yet been returned are said to be pending. The processor will not issue another request while the read is pending. A processor read request is complete after the last transfer of response data has been received from an external agent. A processor write request is complete after the last word of data has been transmitted.

The Processor is the default master of the system interface. An external agent becomes master of the system interface through arbitration or by default after a processor read request, and returns mastership to the processor after an external request completes and/or after the processor read request has been serviced.

The following sections detail the sequence and timing of processor and external requests. Sequence refers to a series of requests that a processor generates to service a system event. Timing refers to the cycle by cycle signal transitions that occur on the processor's system interface pins to realize a processor or external request.

Note that the following describes the SysAD bus *protocol*. The R4300 processor will always meet the conditions of this protocol. R4300 will be capable of receiving sequences of transactions on the bus at full protocol speed and receiving data on every cycle. The design of external agents must at least meet the requirements of the protocol, and ideally will take advantage of the maximum speed of R4300.

8.1 Sequences

The following sections detail a sequence generated by the processor for each system event.

8.1.1 Fetch miss

When the processor misses in the instruction cache on a fetch it obtains a cache line of instructions from an external agent. The processor issues a read request for the cache line and waits for an external agent to provide the data in response to the read request.

8.1.2 Load Miss

When the processor misses in the data cache on a load, it obtains a cache line of data from an external agent. The processor issues a read request for the cache line and waits for an external agent to provide the data in response to the read request. If the data at the cache location which the incoming line will replace contains valid dirty data, the data will be written to memory. The read will complete before the write of the dirty cast-out data.

8.1.3 Store Miss

When the processor misses in the data cache on a store, it issues a read request to bring a cache line of data into the cache, where it is then updated with the store data. If the data at the cache location which the incoming line will replace contains valid dirty data, the data is written to memory. The read will complete before the write of the dirty cast-out data.

To guarantee that cached data written by a store is consistent with main memory, the corresponding cache line must be explicitly flushed from the cache using a cache operation. The cache operations are detailed in section “5.5 Cache Operations”.

8.1.4 Uncached Load or Store

When the processor performs an uncached load, it issues a read request, and waits for a single transfer of read response data from an external agent. When the processor performs an uncached store, it issues a write request and provides a single transfer of data to the external agent. Note that the processor will not consolidate data on uncached writes. For example, writes of two contiguous halfwords will cause two writes, and will never be grouped into a single word write.

8.1.5 Cache Instructions

The R4300 processor provides a number of cache instructions for use in maintaining the state and contents of the caches. For further details on cache instructions see section “5.5 Cache Operations”.

8.2 Byte Order

The system interface byte order is set by the BigEndian bit in the CP0 Config register. The byte order is big endian when high, and little endian when low. The RE (reverse endian) bit in the CP0 status register can be set by software to reverse the byte order in user mode.

8.3 Signal Descriptions

SysAD(31:0): (i/o) Multiplexed address and data transfer bus between the processor and an external agent.

SysCmd(4:0): (i/o) Used for command and data identifier transmission between the processor and an external agent.

EValid*: (i) Signals that an external agent is driving a valid address or valid data on the SysAD bus and a valid command or data identifier on the SysCmd bus during this cycle.

PValid*: (o) Signals that the processor is driving a valid address or valid data on the SysAD bus and a valid command or data identifier on the SysCmd bus during this cycle.

EReq*: (i) Signals that an external agent requests system interface bus ownership.

PReq*: (o) Signals that the processor requests system interface bus ownership. Also, when the processor experiences a protocol error (i.e. the processor detects that an external agent has preformed an action in violation of the SysAD protocol), the processor will continuously toggle PReq*.

PMaster*: (o) Signals that the processor is the master of the system interface bus.

EOK*: (i) Signals that an external agent is capable of accepting a processor request,

Int(4:0)*: (i) General processor interrupt. These are visible as bits 14 to 10 of the Cause register.

NMI*: (i) Non-maskable interrupt.

Reset*: (i) When asserted, initiates an maintains a warm reset in the processor.

TClock: (o) Transmit clock at the operation frequency of the system interface. Equal in frequency and phase to MasterClock.

MasterClock: (i) Master clock input at the operation frequency of the system interface.

SyncOut: (o) Synchronization clock output.

SyncIn: (i) Synchronization clock input.

ColdReset*: (i) When asserted, this signal indicates to the R4300 processor that the +3.3 volt power supply is stable and the R4300 chip should initiate a cold reset sequence. The assertion of ColdReset* will reset the PLL. Asynchronous.

JTDI: (i) JTAG serial data in.

JTDO: (o) JTAG serial data out.

JTMS: (i) JTAG command signal, signals that the serial data in is command data.

JTCK: (i) JTAG serial clock input.

BypassPLL*: (i) This signal forces MasterClock to bypass the PLL and to feed directly to the clock buffers. This should be used for test only. This signal will be implemented as a non-bonding pad (default deasserted) and may not be a pin on the production package. However this pin will be part of the JTag scan chain.

TestMode* (i) This is used for testing cache directly. This must be deasserted (connected to VCC) for normal operation. This signal will be implemented as a non-bonding pad (default deasserted) and may not be a pin on the production package. However this pin will be part of the JTag scan chain.

DivMode(1:0) (i) These signals are an encoding of the PClock to MasterClock ratios. TClock (system interface clock) will be the same frequency as MasterClock. The encoding of DivMode for an example of 40MHz MasterClock is shown below:

DiveMode(1:0)	MasterClock	TClock	PClock	Ratio
00	40MHz	40MHz	40MHz	1:1
01	40MHz	40MHz	60MHz	1.5:1
10	40MHz	40MHz	80MHz	2:1
11	40MHz	40MHz	120MHz	3:1

The primary communication paths for the system interface are a thirty-two bit address and data bus, SysAD(31:0), and a five bit command bus, SysCmd(4:0). The SysAD bus and the SysCmd bus are driven by the processor to issue a processor request when it is master, indicated by the assertion of PMaster*, and driven by an external agent to issue an external response when the processor is a slave, indicated by deassertion of PMaster*.

A request through the system interface consists of an address, a system interface command that specifies the nature of the request, and a series of data elements if the request is for a write, or a read response for a read. Addresses and data are transmitted on the SysAD bus. System interface commands are transmitted on the SysCmd bus.

When the processor is master it will assert the PValid* signal when the SysAD bus and the SysCmd bus are valid. When the processor is slave, an external agent will assert the EValid* signal when the SysAD bus and the SysCmd bus are valid.

The SysCmd bus is used to identify the contents of the SysAD bus during valid 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 the SysCmd bus contains a system interface command. During data cycles, the remainder of the SysCmd bus contains a data identifier that indicates if the current data cycle is the last, along with other information about the data cycle. The encoding of system commands and data identifiers is detailed in section 8.6 Signal codes.

8.4 Signal timing

The system interface protocol describes the cycle by cycle signal transitions that occur on the pins of the system interface to realize requests between the processor and an external agent.

The summary sub section (8.4.1 Timing Summary) describe the min and max timing of each signal., it is intended to fully specify the signal cycle timing for the system interface protocols. All the other following sub sections in this section illustrate how they are used and provide some bus timing examples.

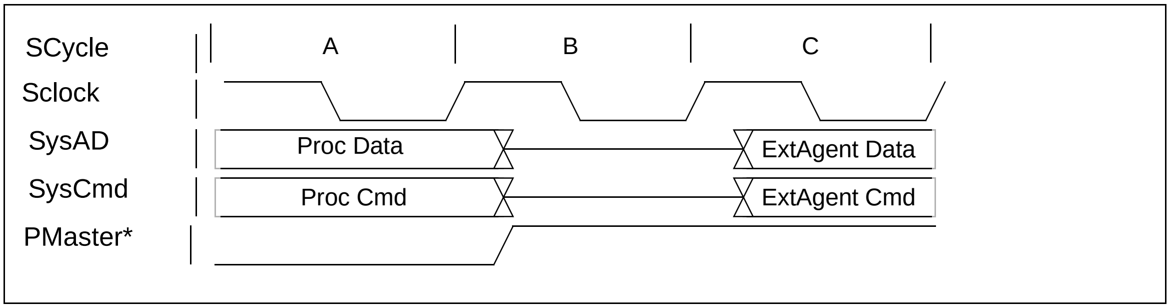
8.4.1 Timing Summary

This timing summary section fully specifies the cycle timing for the system interface.

For each of the timing diagrams in this section, there are many signals that are either not shown or have “grayed out” signal values. Within the context shown a signal with a “grayed” value has no specifically required value, and so it can be any value as long as that value does not violate any other bus value or timing specification. In other words, grayed out signal values denote unknown or don’t care values, within the limits of the spec.

PMaster*: (o) Signals that the processor is the master of the system interface bus.

Figure 17: PMaster* Timing: Processor to ExtAgent

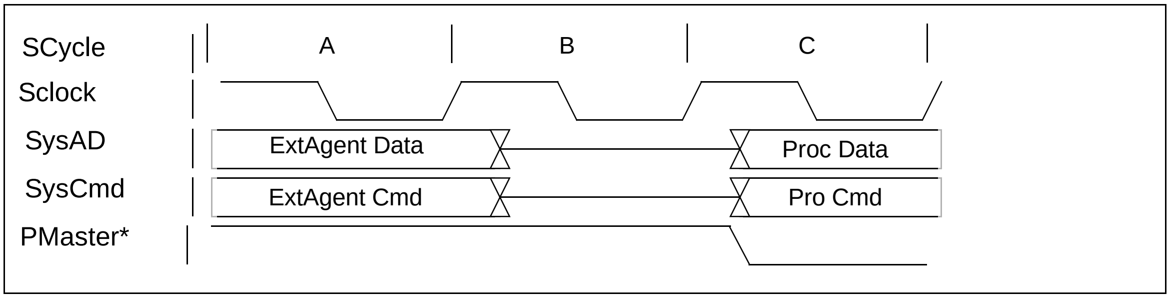


A Processor drives SysAD and SysCmd (processor is master).

B PMaster is deasserted. SysAD and SysCmd is tri-stated (there is no bus master).

C External agent drives SysAD and SysCmd (external agent is master).

Figure 18: PMaster* Timing: ExtAgent to Processor

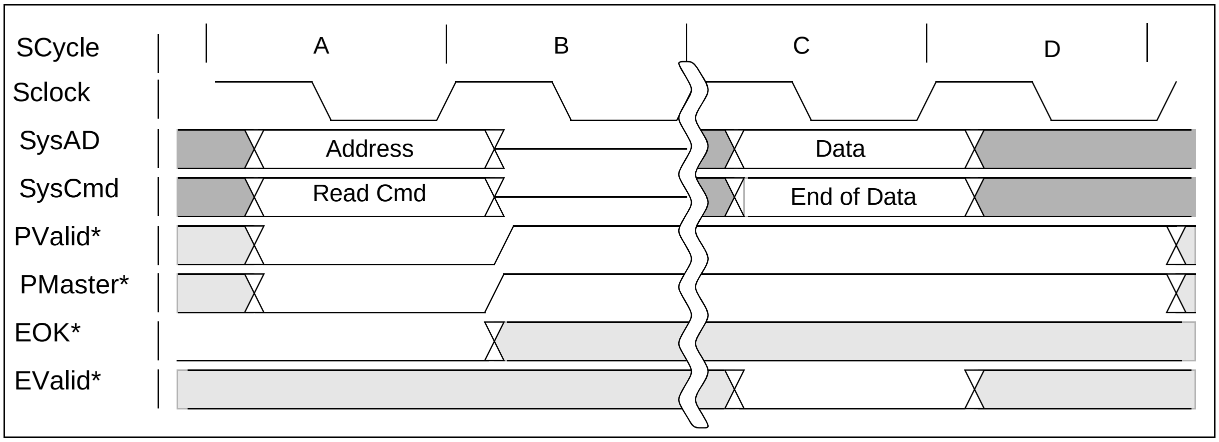


A External agent drives SysAD and SysCmd (external agent is master).

B SysAD and SysCmd is tri-stated (there is no bus master).

C PMaster is asserted. Processor drives SysAD and SysCmd (processor is master).

Figure 19: PMaster* Timing: Processor Read Request



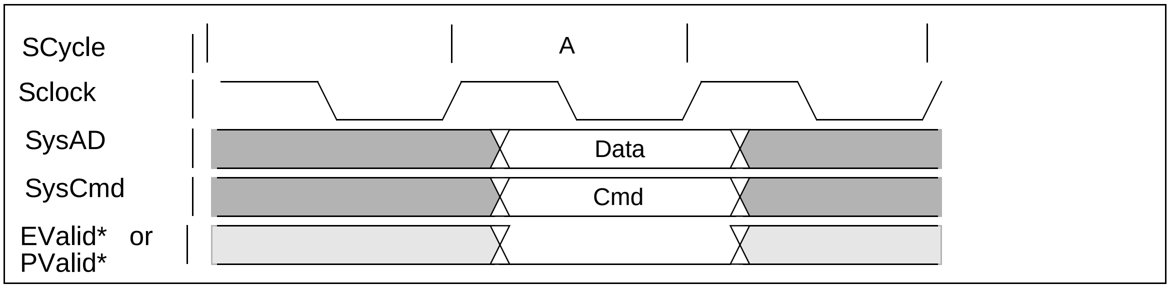
A Processor drives a valid read command and an external agent accepts it.

B PMaster is deasserted. Bus is tri-stated.

C External agent drives last of requested data. For all cycles between B and C the external agent is guaranteed mastership of the bus.

EValid* (i), PValid* (o) Signals a new valid address or valid data on the SysAD bus and a new valid command or data identifier on the SysCmd bus during this cycle. EValid* indicates if an external agent is driving new SysAD and SysCmd values. PValid* indicates if the processor is driving new SysAD and SysCmd values.

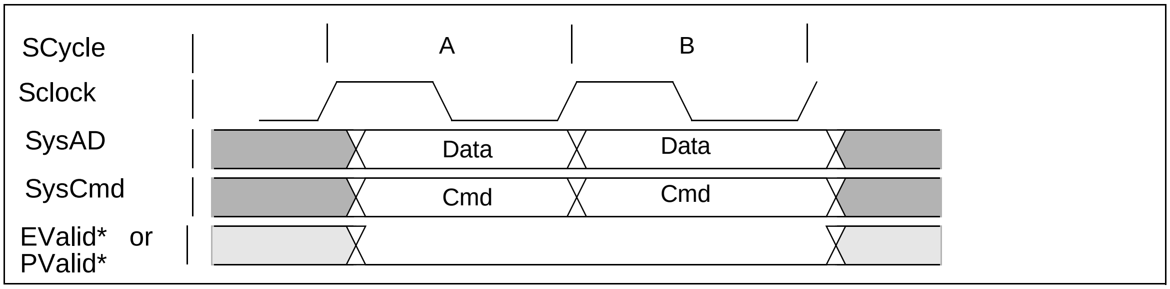
Figure 20: EValid*, PValid* Timing



A: New SysAD and SysCmd values.

Every cycle one of these signals remains asserted indicates that there is a new SysAD and SysCmd value. The <u>only</u> exception to this rule is for processor read & write request commands. If a processor read/write request command is not accepted by the external agent, the processor will repeat the command.

Figure 21: Multi-cycle EValid*, PValid* Timing



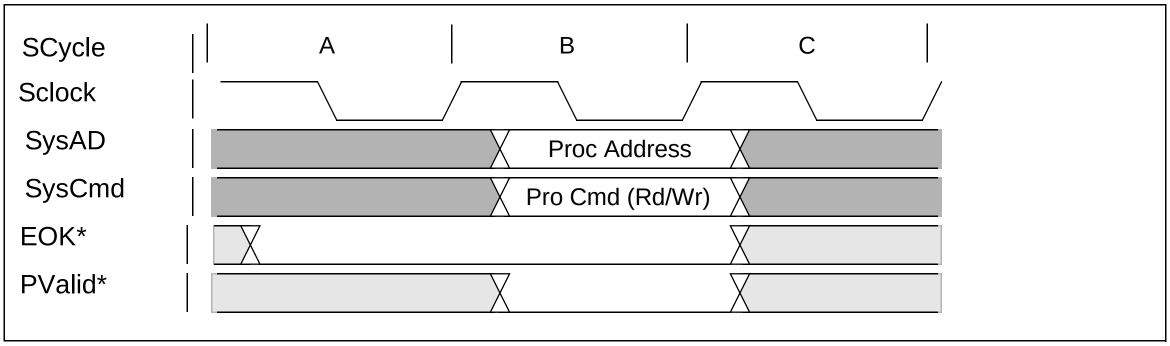
A: New SysAD and SysCmd value.

B: Another new SysAd and SysCmd value.

The only exception is: if A is a processor read/write request command that is not accepted by the external agent, then B will be a repeat of the command and data in A.

EOK*: (i) Signals that an external agent accepts a processor request. An external agent has excepted the processor Rd/Wr command iff (if and only if) the following has occurred:

Figure 22: EOK* Timing



A EOK is active.

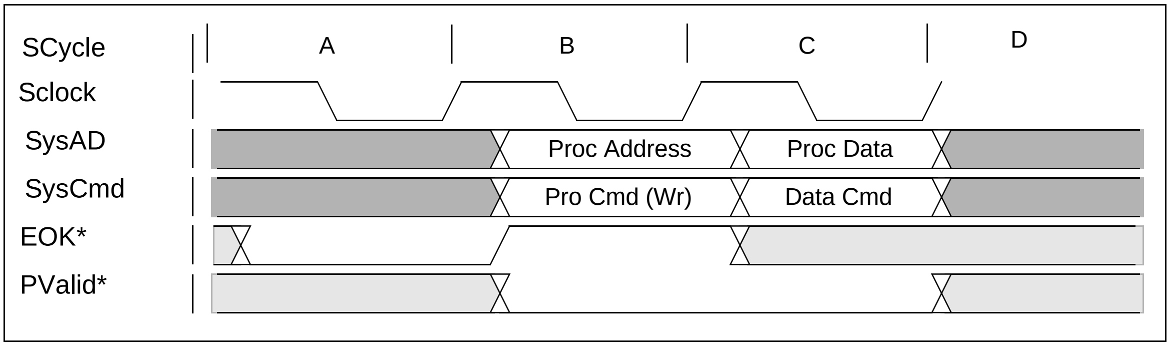
B Processor asserts PValid and drives a Read or Write command. EOK is asserted. External agent excepts the Processor command.

Once the external agent has accepted a processor write command, the agent must be able to accept the entire data size at the programmed data rate immediately following the command.

The external agent may provide read response data to the processor at any rate.

Deasserting EOK may kill a processor read/write request in progress. If this occurs, the external agent must ignore command and data from the processor in the following cycle.

Figure 23: EOK* Timing: Killed Processor Write



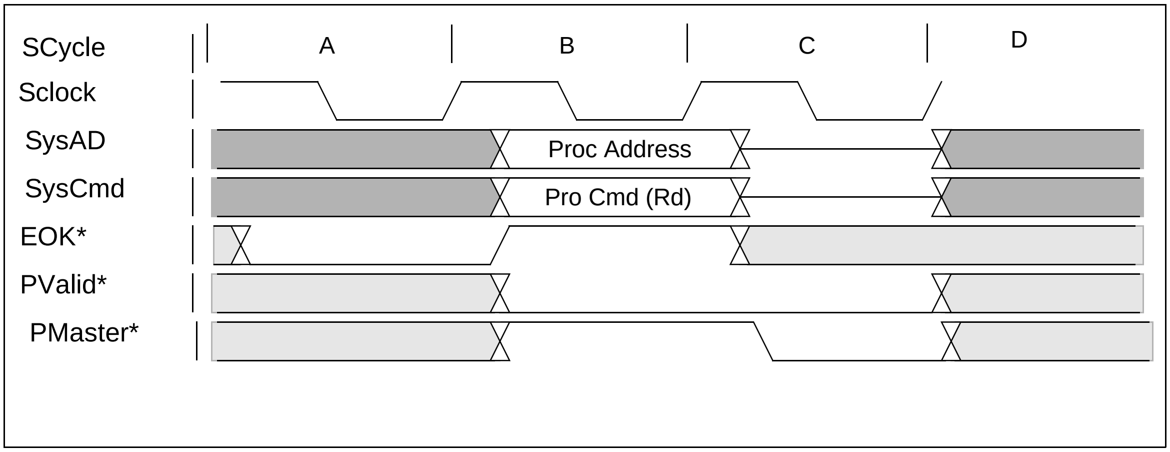
A EOK is active.

B Processor asserts PValid and drives a Read or Write command. EOK is deasserted (external agent has killed the processor's command)

C The external agent must ignore any SysAD and SysCmd data from the processor.

D The external agent <u>does not</u> ignore any SysAD and SysCmd data from the processor.

Figure 24: EOK* Timing: Killed Processor Read



A EOK is active.

B Processor asserts PValid and drives a Read or Write command. EOK is deasserted (external agent has killed the processor's command)

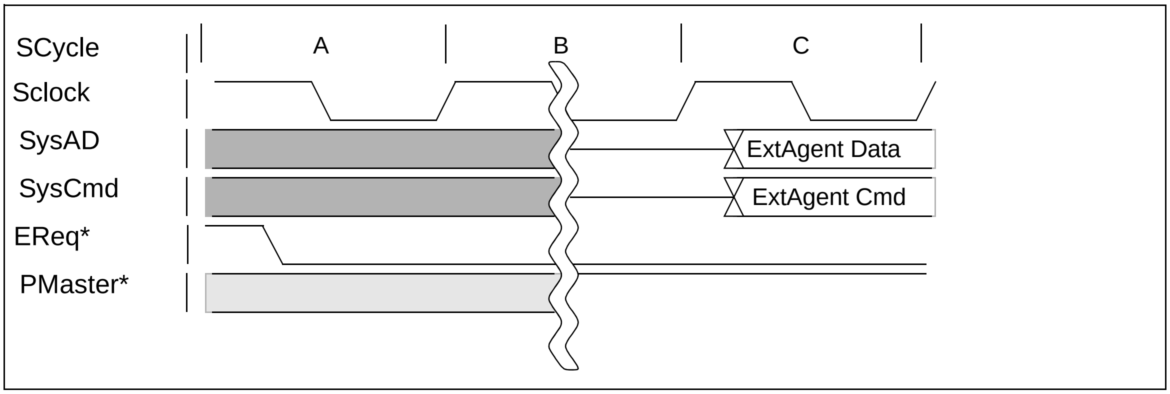
C The external agent must ignore any SysAD and SysCmd data from the processor.

D The external agent does not ignore any SysAD and SysCmd data from the processor.

When a Processor request has been killed, the processor will always retry the same request before giving a new Read/Write request.

EReq*: (i) Signals that an external agent requests system interface bus ownership. To gain mastership of the bus, an external agent must arbitrate with the processor as follows:

Figure 25: EReq* Timing: Bus Request



A External agent asserts EReq.

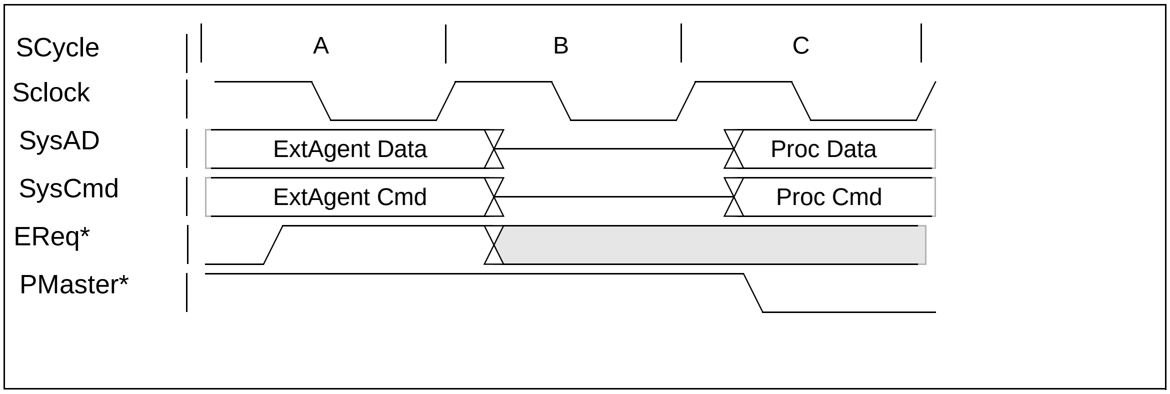
B Wait for PMaster to be deasserted (1 to N cycles).

C External agent drives SysAD and SysCmd. The external agent is guaranteed to maintain mastership of the bus as long as EReq is asserted. If at any time EReq is deasserted, the external agent must go back to step A and re-arbitrate for the bus.

From the time that EReq* is asserted, the external agent is guaranteed to gain mastership of the bus after at most one processor request. However, if EOK* is also being deasserted, the external agent will gain mastership of the bus without having to accept any processor requests.

The external agent gives up mastership of the bus by deasserting EReq:

Figure 26: EReq* Timing: Bus Release



A External agent de-asserts EReq. External agent is driving bus.

B Bus is tristated.

C Processor regains mastership of bus.

Except for a processor read request (see “Figure 19: PMaster* Timing: Processor Read Request” above), using EReq is the only way the external agent gets and maintains bus mastership.

PReq* (o) Signals that the processor is requesting the bus. When the processor is in slave state and has a read/write request to issue, it will assert Preq*.

PReq* is also used to indicate when the R4300 has detected a protocol error. When a protocol error is detected, the processor’s system interface will hang and the PReq* signal will oscillate.

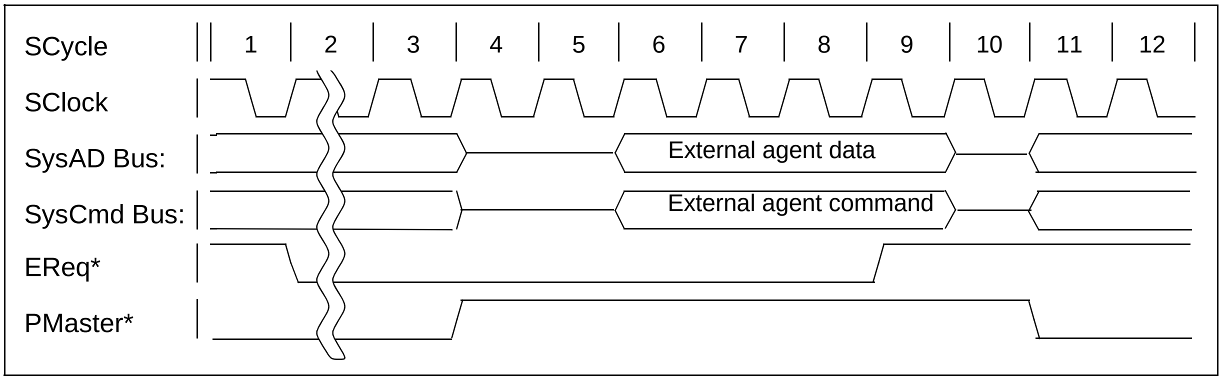
8.4.2 Arbitration

The processor is the default master of the bus and relinquishes ownership of the bus either when an external agent requests and is granted the system interface or until the processor issues a read request. The transition from processor master to slave is arbitrated by the processor using the system interface handshake signals EReq* and PMaster*.

When a processor read request is pending, the processor transitions to slave by de-asserting PMaster*, to allow an external agent to return read response data. The processor remains slave until an external agent issues an end of data read response, when it then transitions to master with the assertion of PMaster*. Note that an external agent is able to maintain mastership of the bus after an end of data read response if the external agent arbitrates for mastership using EReq*.

When the processor is master, an external agent acquires control of the system interface by asserting EReq*, and waiting for the processor to de-assert PMaster*. When the processor is ready to go slave state, it will de-assert PMaster*. The external agent must go through the 3 step arbitration process (see EReq* in “8.4.1 Timing Summary”) before driving the bus. Once the external agent has become master with an EReq*, it may continue to assert EReq* until it is ready to relinquish the bus. That is, once the external agent has become bus master, it can remain master as long as it wants by simply continuing to assert EReq*. The system interface will return to master state (with the processor driving the bus) two cycles after EReq* is de-asserted. An arbitration for external requests example is illustrated in “Figure 27: External Request Arbitration”.

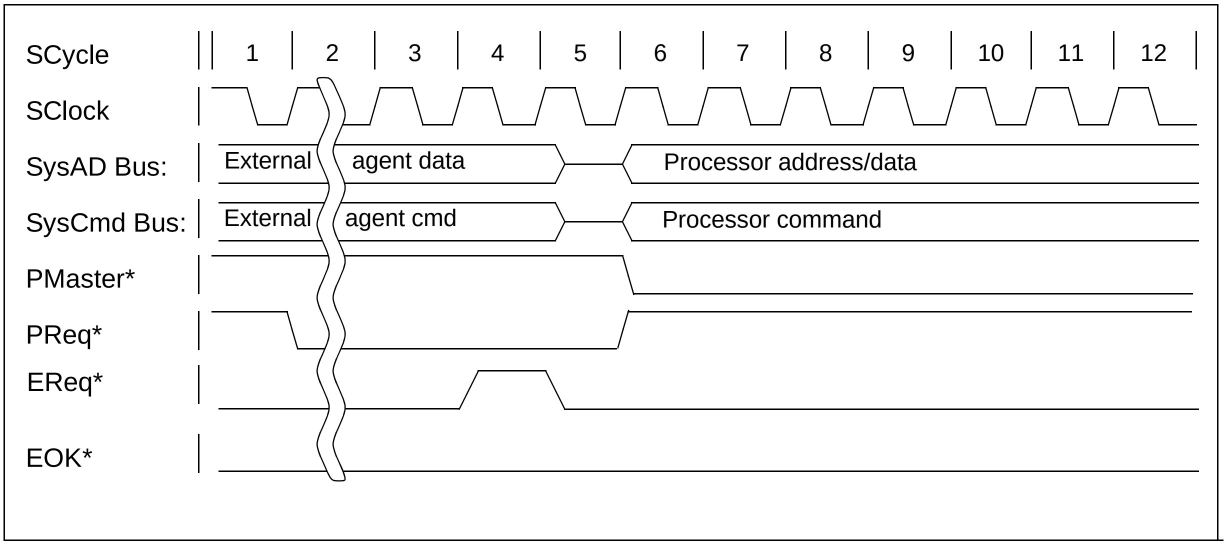
Figure 27: External Request Arbitration



When an external agent is master, it may always respond with data to a read. If the external agent has become master with EReq*, it may issue transactions at will. That is, the processor must always accept any command or data on the bus at any time. There is no means for the processor to hold off the external agent once the external agent is master. However the processor can request the bus by asserting PReq*, and the external agent may or may not honor the request depending on the system priorities.

If the processor is in slave state and needs the bus, it may assert the PReq* line to let the external agent know that it wants the bus. When the processor sees EReq* de-asserted, it resumes ownership, asserts the PMaster* line, and can issue its command. The processor will become master and drive the bus two cycles after EReq* is deasserted. An example of processor request for mastership of the bus and the release of the bus by the external agent is illustrated in “Figure 28: Processor request for bus arbitration and external agent release”.

Figure 28: Processor request for bus arbitration and external agent release



Upon assertion of Reset* or ColdReset*, the processor becomes bus master and the external agent must become slave.

This protocol guarantees that either the processor or an external agent is always bus master. The master should never tristate the bus, except when giving up ownership of the bus under the rules of the protocol.

8.4.3 Issuing Commands

When the processor is mastering the bus and wishes to issue a command, it cannot successfully issue the command until the external agent signals that it is ready to accept one. This is indicated by the EOK* line. Since it is master, the processor may place the command on the bus and continually reissue it while waiting for EOK* to be asserted; however, the command is not considered issued until EOK* has been asserted for two consecutive cycles (see EOK* in "8.4.1 Timing Summary").

If the EOK* signal is asserted in one cycle then deasserted in the next cycle while at the same cycle time a command is issued, that command is considered killed and must be retried. When a command is killed this way, the processor will begin to execute the read/write command. This action must be ignored by the external agent. If a write command is killed, the data cycle following this killed transaction must be ignored. If a read is killed, the processor will release the bus one cycle after and (assuming no EReq*) will regain mastership two cycles after. These actions allow the processor to retry the transaction.

8.4.4 Processor Write Request

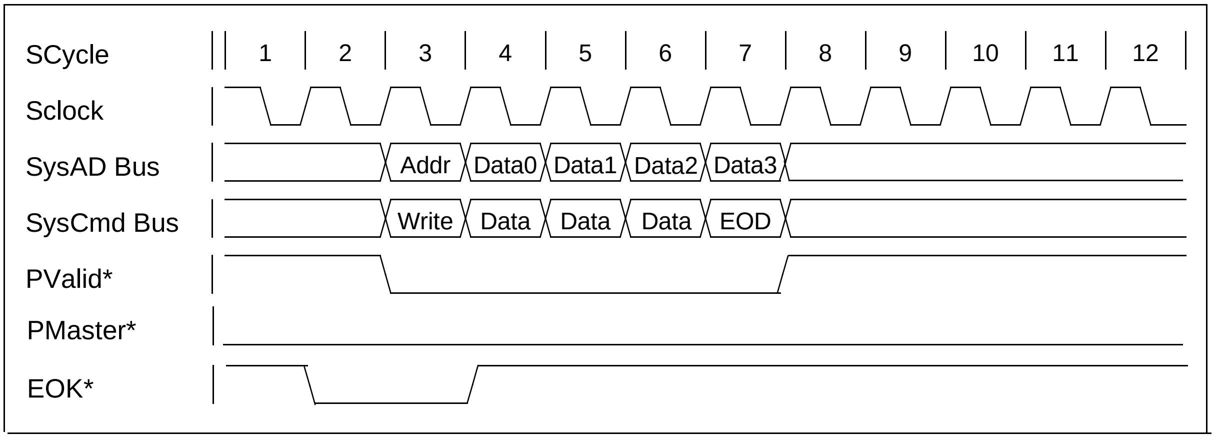
A processor write request is issued by driving a write command on the SysCmd bus, driving a write address on the SysAD bus, and asserting PValid* for one cycle, followed by driving the appropriate number of data identifiers on the SysCmd bus, driving data on the SysAD bus, and asserting PValid*. For 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. R4300) the data will begin on the cycle immediately following the write issue cycle, and will come at a programed cycle data rate thereafter. The processor will drive data at the rate specified by the data rate configuration signals, see "8.4.9 Data Rate Control".

Writes may be cancelled and retried with the EOK signal. See above in "8.4.3 Issuing Commands".

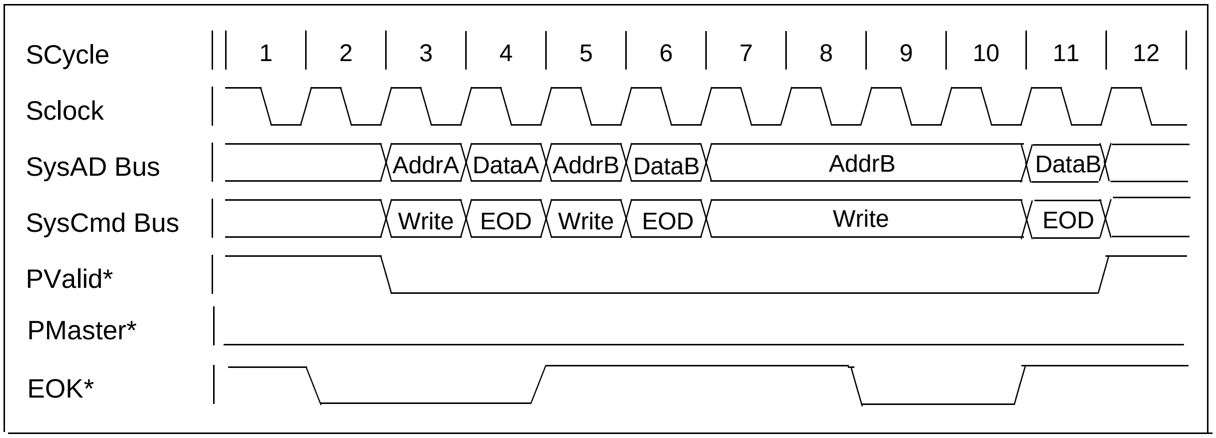
The example in "Figure 29: Processor block write request with D data rate" illustrates the bus transactions for four word data cache block store.

Figure 29: Processor block write request with D data rate



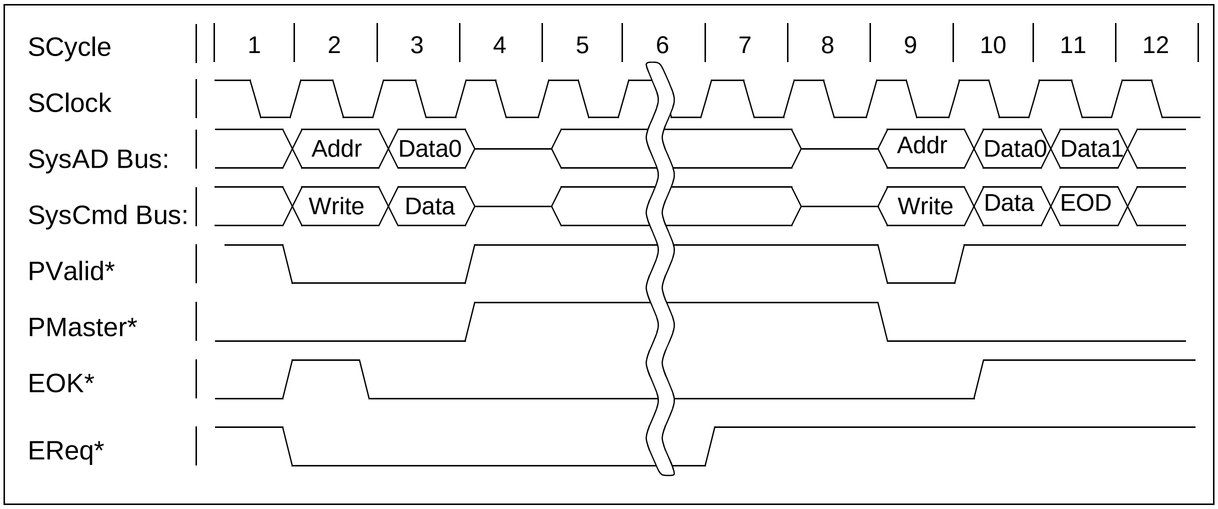
The example in "Figure 30: Processor single write request followed by a killed and retried write request" illustrates a write request which is cancelled by the de-assertion of EOK* during the address cycle of the second write and which is retried when EOK* is asserted again.

Figure 30: Processor single write request followed by a killed and retried write request



“Figure 32: Processor read request” has an example of a killed double word write request where the external agent request and gets the bus. After the external agent releases the bus, the killed write request is retried.

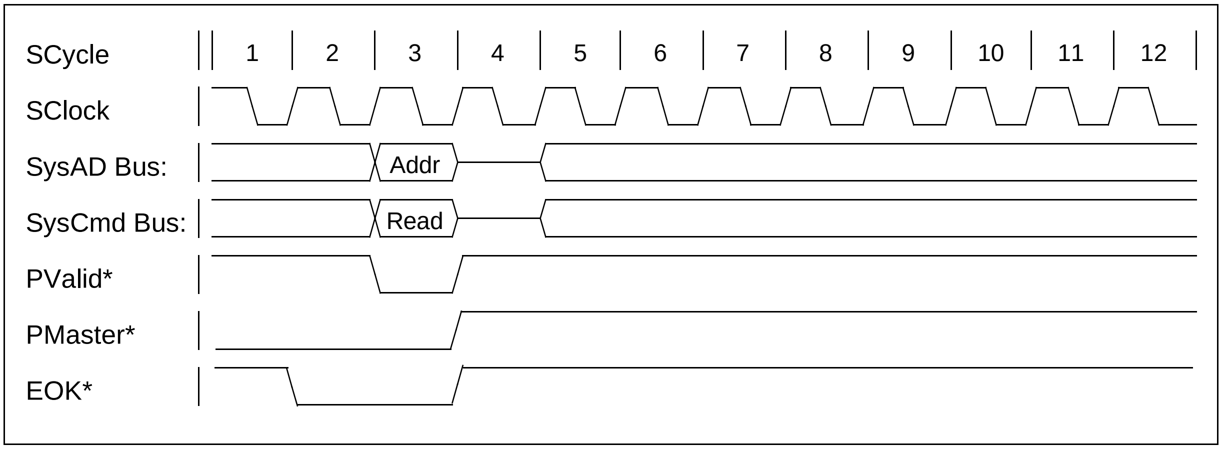
Figure 31: Killed and retried write request with intervening external request



8.4.5 Processor Read Request

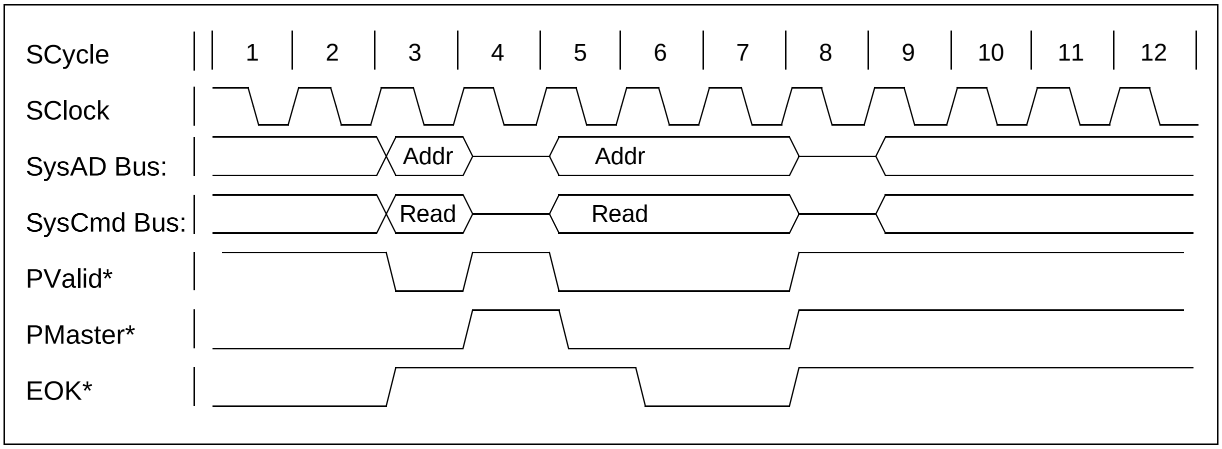
A processor read request is issued by driving a read command on the SysCmd bus, driving a read address on the SysAD bus, and asserting PValid*. Only one processor read request may be pending at a time. The processor must wait for an external read response before starting a subsequent read. The processor 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 32: Processor read request”.

Figure 32: Processor read request



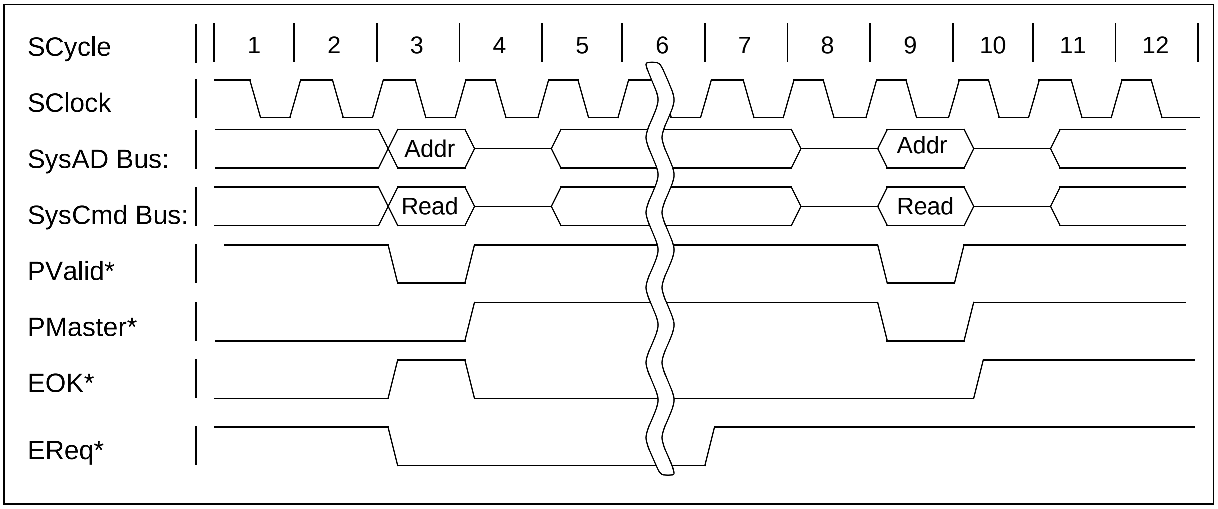
The example in “Figure 32: Processor read request” illustrates a read request that is cancelled by the de-assertion of EOK* during the address cycle. The read is retried when EOK* is asserted again.

Figure 33: Killed and retried processor read request



“Figure 32: Processor read request” has an example of a killed read request where the external agent request and gets the bus. After the external agent releases the bus, the killed read request is retried.

Figure 34: Killed and retried read request with intervening external request

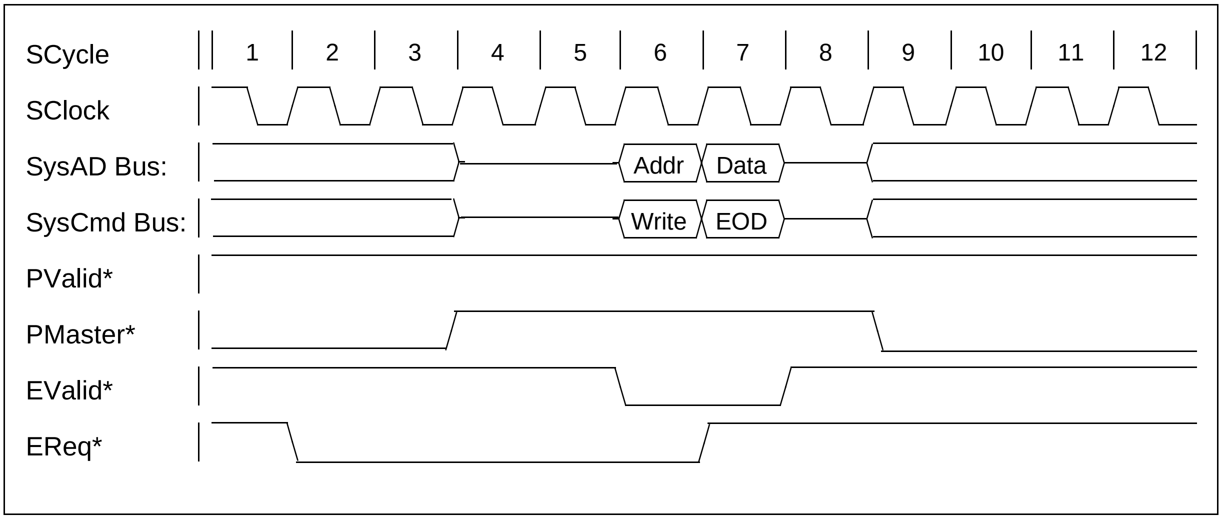


8.4.6 External Write Request

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 (see EReq* in “8.4.1 Timing Summary”).

An external write request example with the processor initially master is illustrated in “Figure 35: External write request”.

Figure 35: External write request



An example of a read response for a processor single word read request that is interrupted by an external agent write request is illustrated in “Figure 39: External write followed by external read response, system interface in slave state”. External writes can not occur in the middle of a data response block. External writes can occur before the first data response of the data block or after the last “EOD” response, but it can not occur between them.

Note: The only writable resources are processor interrupts. An external write to any address is treated as a write to the processor interrupts.

8.4.7 External Read Response

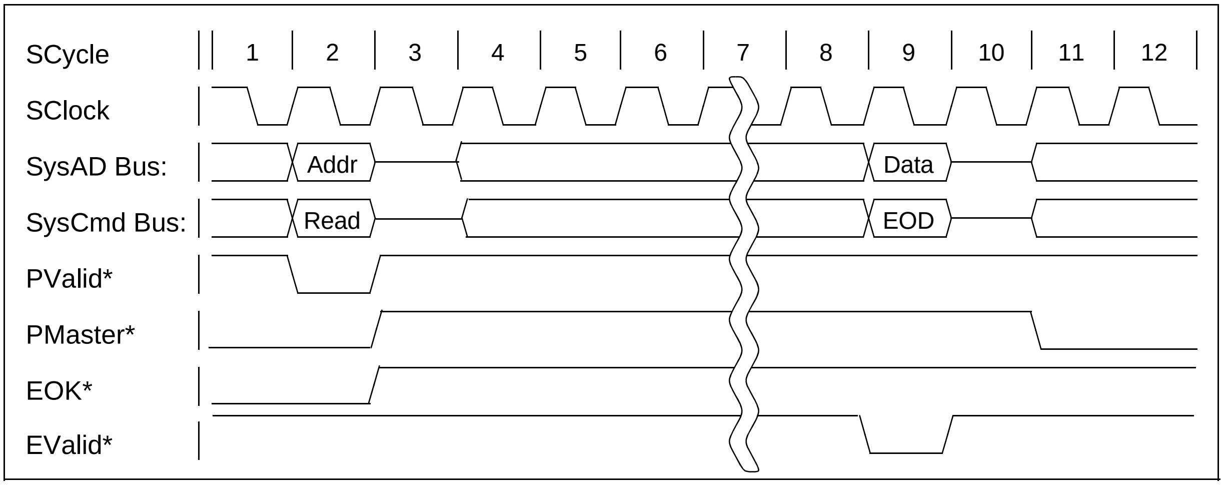
An external agent returns data to the processor in response to a processor read request by waiting for the processor to transition to slave, and then returning the data via a single data cycle or a series of data cycles sufficient to transmit the requested data. After the last data cycle is issued the read response is complete and the processor will become 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.

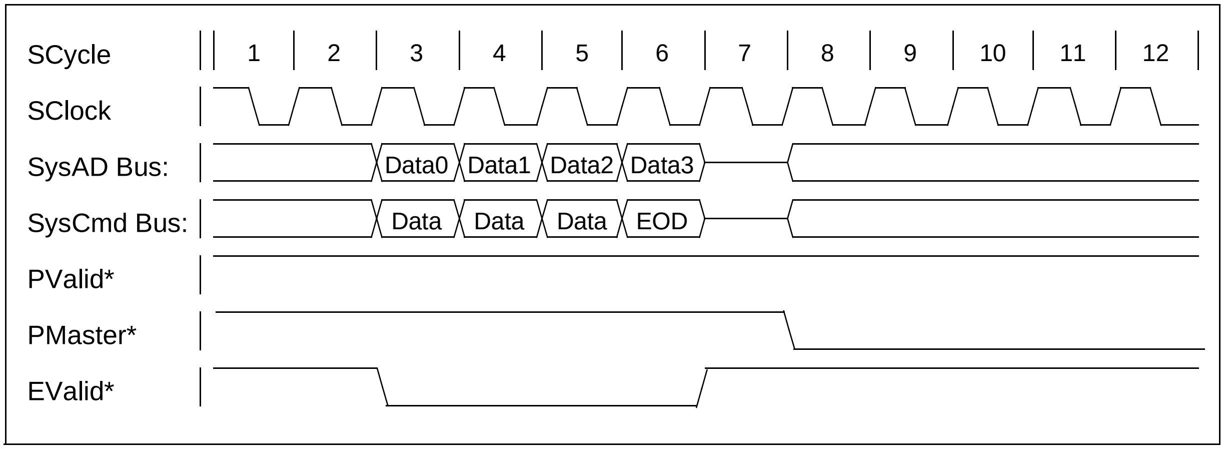
An example of a processor single read request followed by a read response is illustrated in “Figure 36: Single read request followed by read response”.

Figure 36: Single read request followed by read response



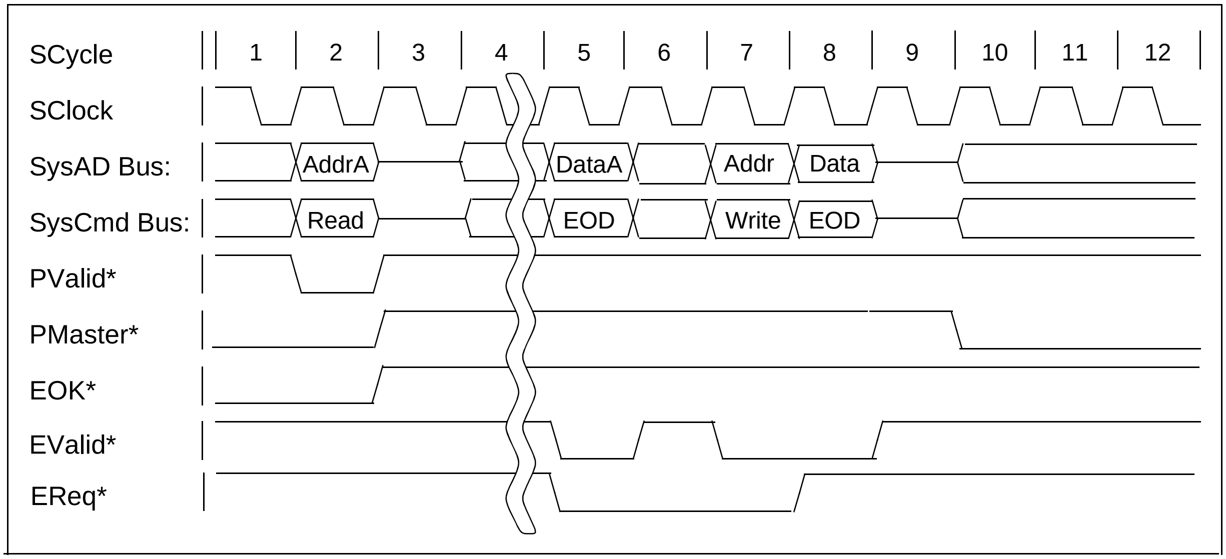
A read response example for a processor block read with the system interface already in slave state is illustrated in “Figure 37: Block read response, system interface already in slave state”.

Figure 37: Block read response, system interface already in slave state



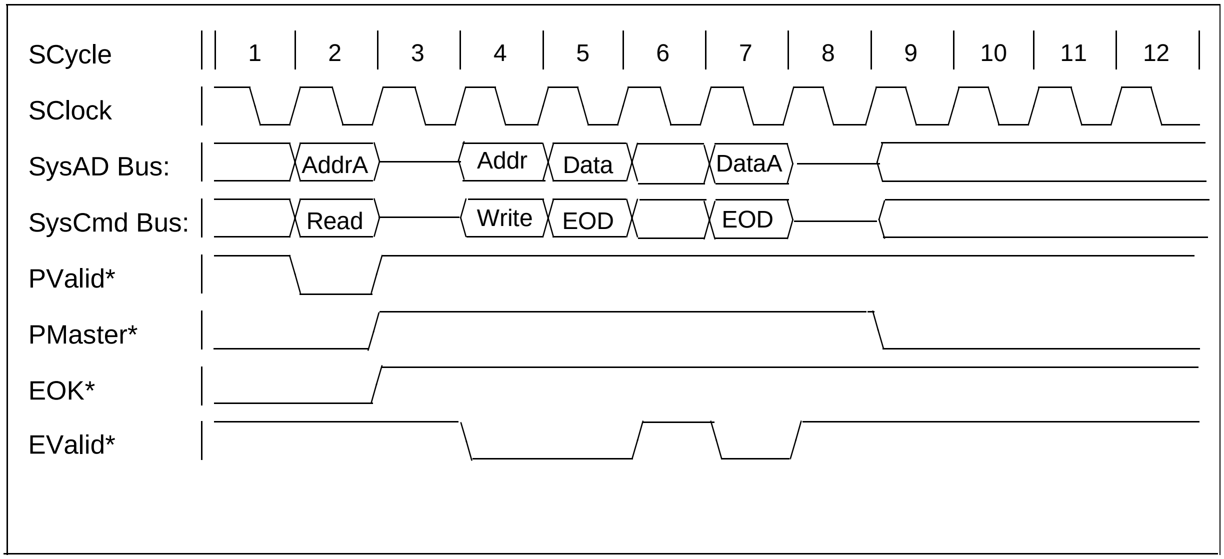
A read response example for a processor single read request followed by an external agent write request is illustrated in "Figure 38: Single read request followed by external write request (external agent keeps bus)".

Figure 38: Single read request followed by external write request (external agent keeps bus)



An example of a read response for a processor single word read request that is interrupted by an external agent write request is illustrated in "Figure 39: External write followed by external read response, system interface in slave state". Cycle 5 is the data for the external write request in cycle 4. Cycle 7 is the read response data.

Figure 39: External write followed by external read response, system interface in slave state



8.4.8 Flow Control

The signal EOK* may be used by an external agent to control the flow of processor read and write requests. While EOK* is de-asserted the processor will repeat the current address cycle until an external agent signals it is ready by asserting EOK*. There is a one cycle delay from the assertion of EOK* to the state in which the Read/Write command becomes valid. EOK* must be asserted for two consecutive cycles for the command issue completion. For more details on the usage of EOK*, see above in “8.4.3 Issuing Commands”. Examples of EOK* use is illustrated in “Figure 40: Delayed processor read request” and in “Figure 41: Two processor write requests, second write delayed”.

Figure 40: Delayed processor read request

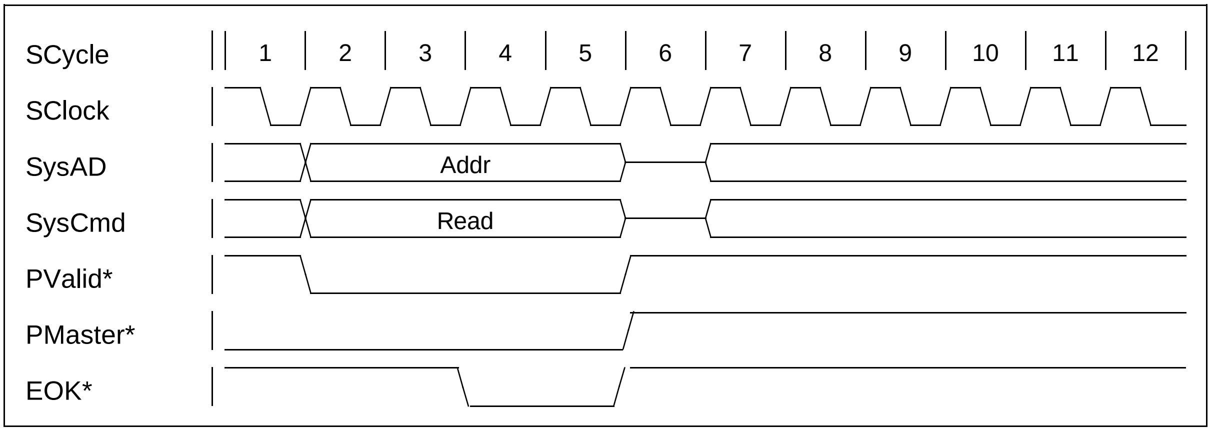
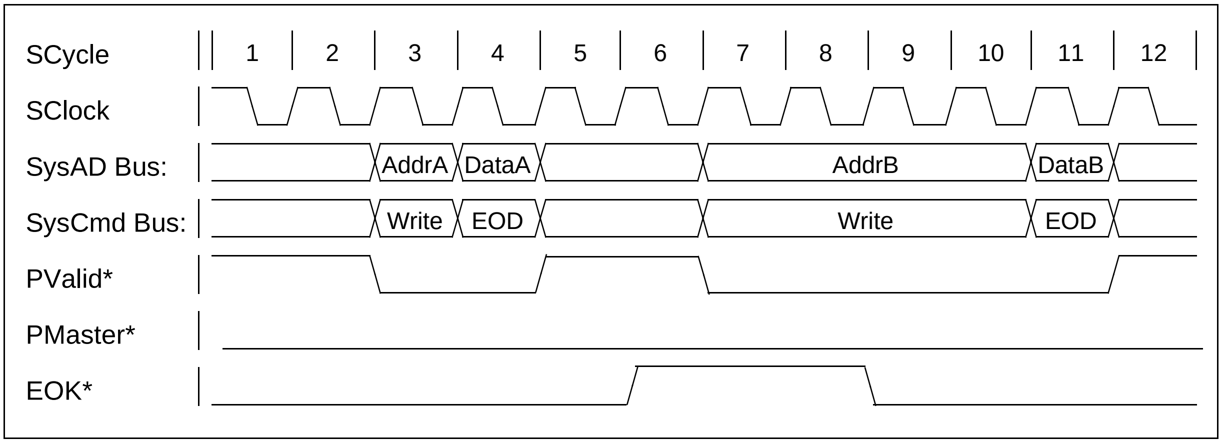


Figure 41: Two processor write requests, second write delayed



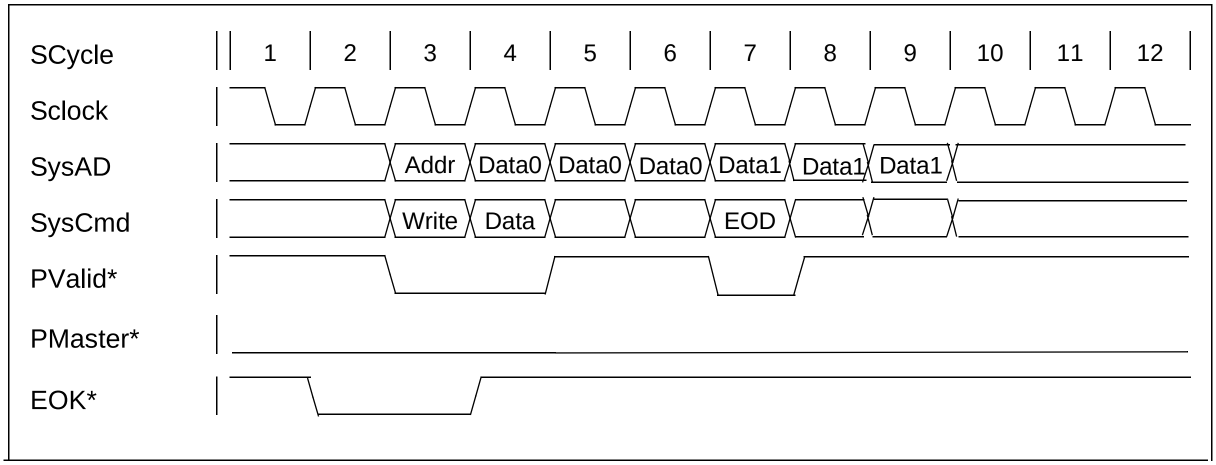
8.4.9 Data Rate Control

The system interface supports a maximum data rate of one word per cycle. The rate at which data is delivered to the processor may be controlled by an external agent by driving data and asserting EValid* only when data is available. The processor will interpret cycles during which EValid* is asserted and the SysCmd bus contains a data identifier as valid data cycles. The processor will continue to accept data until the data word tagged as the last data word is received. An external agent may deliver data to the processor at the system interface maximum data rate.

The rate at which the processor transmits data to an external agent is programmable via the EP field in Coprocessor0 Configuration Register. Data patterns are specified using the letters “D” and “x”, where “D” indicates a data cycle and “x” indicates an unused cycle. A data pattern is specified as a sequence of letters, indicating a sequence of data and unused cycles that will be repeated to provide the appropriate number of data cycles for a given transfer. For example, a data pattern specified by the sequence of letters “DDxx” achieves a data rate of two words every four cycles. A processor block write request example for two words with Dxx pattern is illustrated in “Figure 42: Processor block write request with Dxx data rate”; this transaction results from a store doubleword instruction.

R4300 supports only two modes, D or Dxx. Note that during the cycles indicated by an x in the pattern, the processor will continue to hold the same data as the previous cycle.

Figure 42: Processor block write request with Dxx data rate



8.4.10 Consecutive SysAD Bus Transactions

The following figures (Figure 43 to Figure 46) are miscellaneous examples that illustrate the minimum cycles required between consecutive bus transactions.

Figure 43: Processor single word read followed by block write request

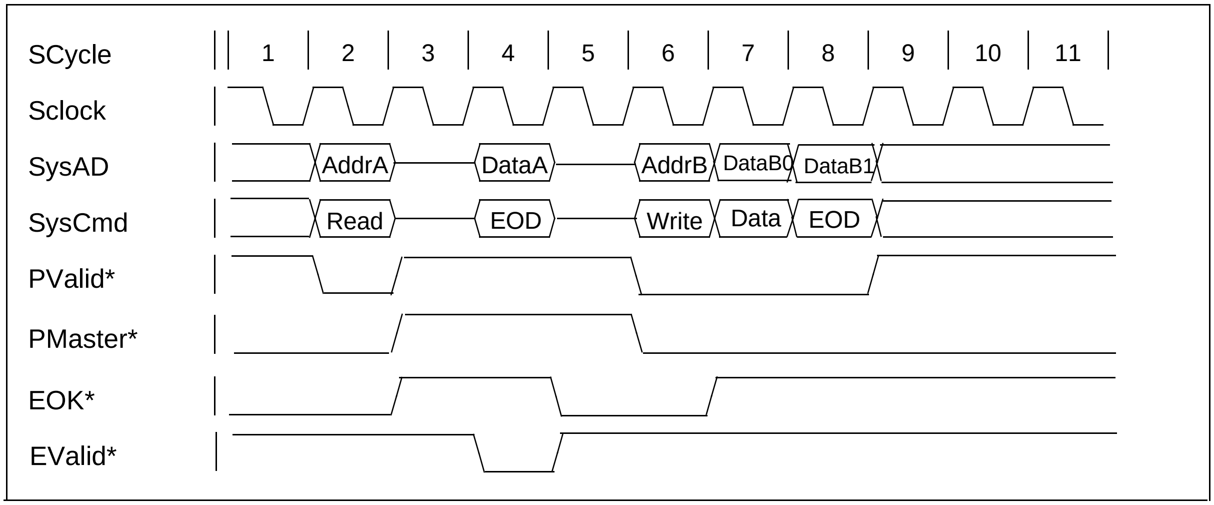


Figure 44: Consecutive processor single word write requests with D data rate

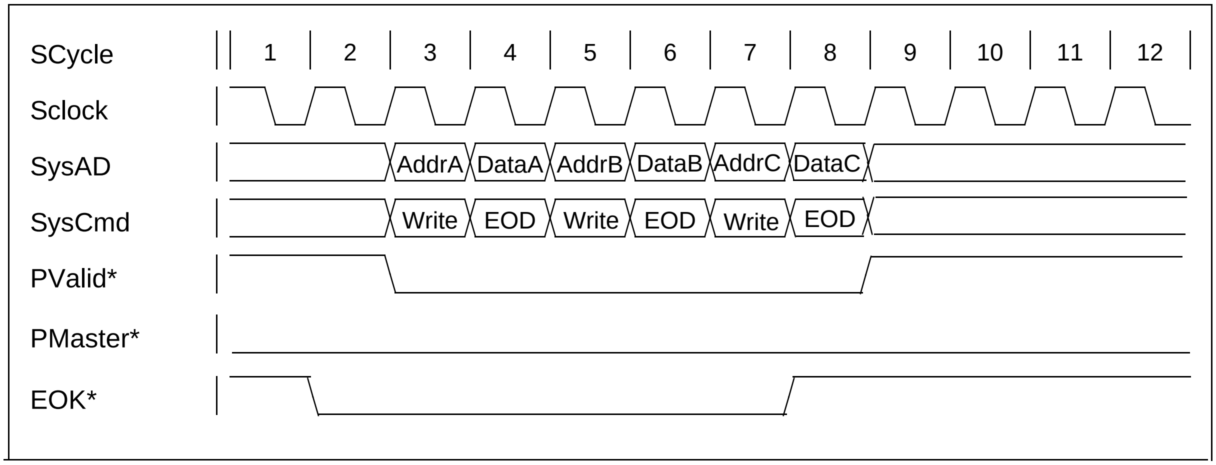


Figure 45: Consecutive processor single word write requests with Dxx data rate

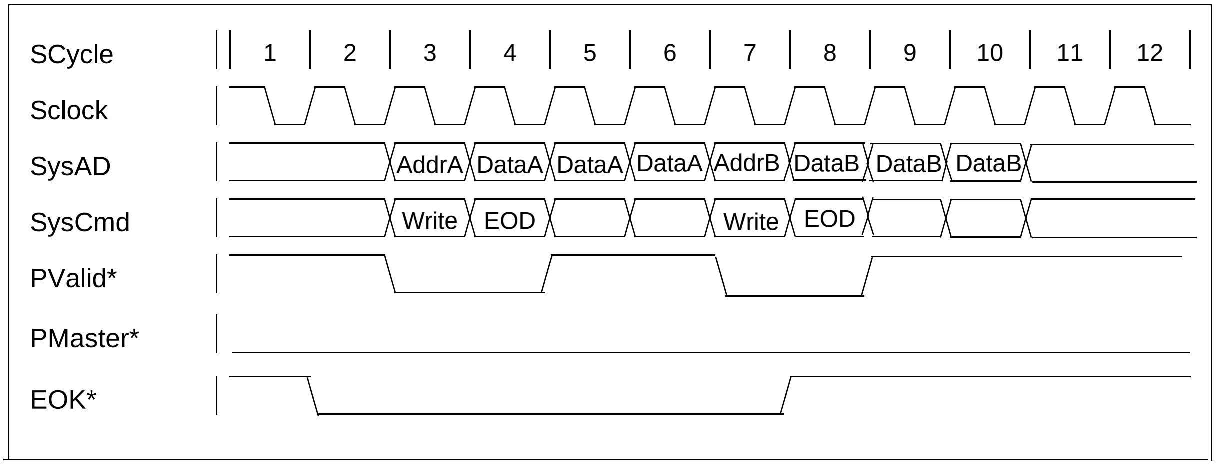
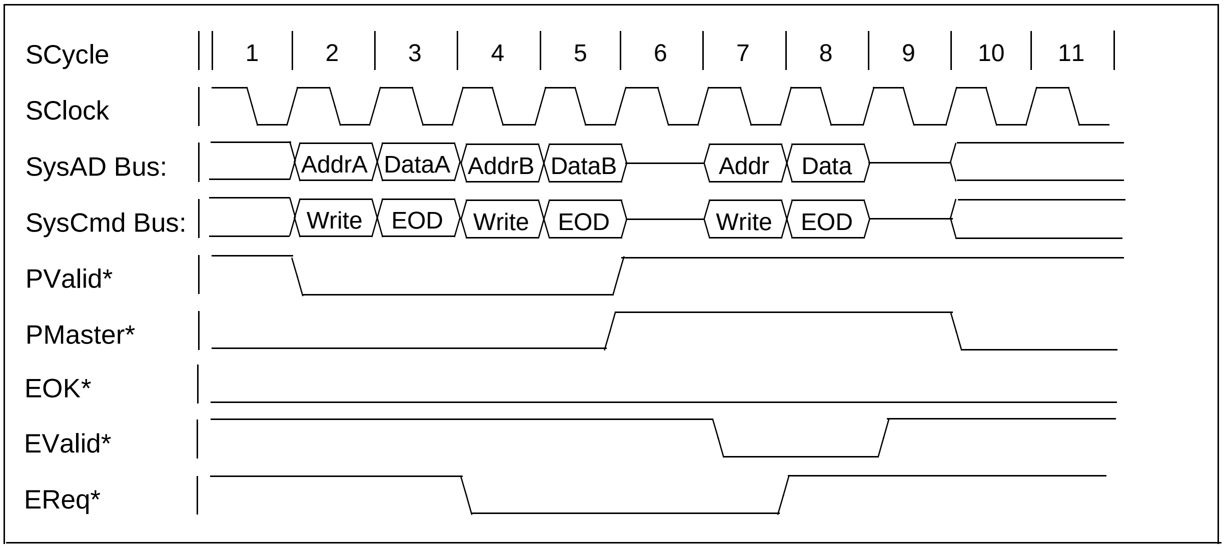


Figure 46: Consecutive processor write requests followed by external write request



8.4.11 Starvation and Deadlock Avoidance.

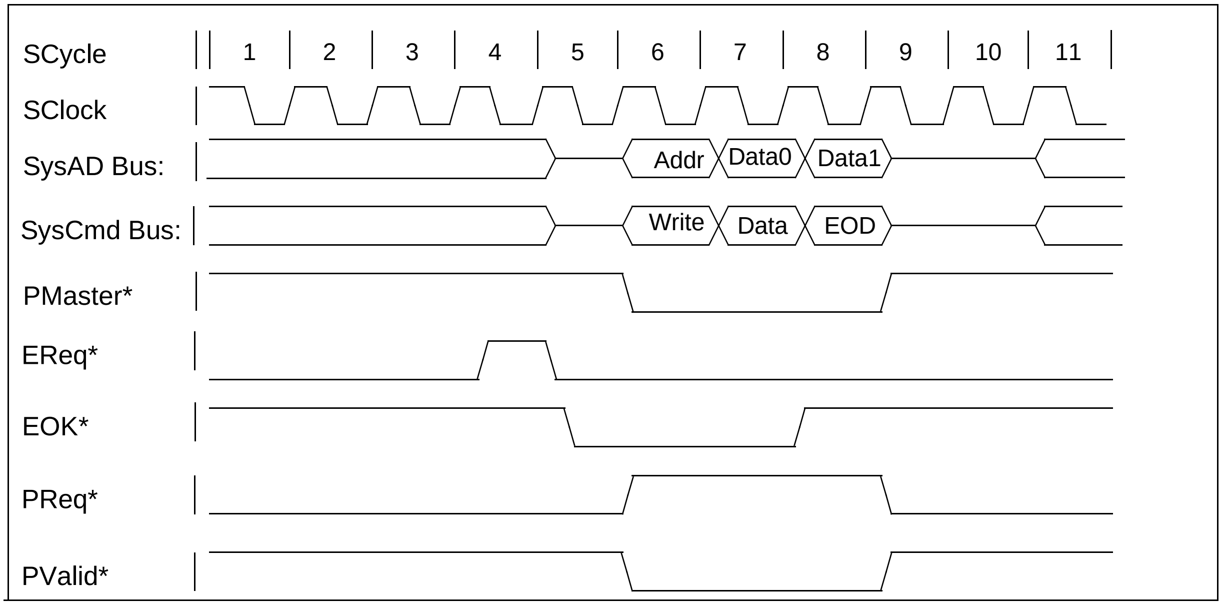
Careful use of the EReq* and PReq* signals will allow a system to avoid starvation and deadlock situations.

Whenever an external agent needs the bus, it can request the bus by asserting EReq*. The external agent is guaranteed to gain mastership of the bus after accepting at most one read/write request from the processor. If the external agent also deasserts EOK*, it is guaranteed to gain mastership of the bus without accepting any read/write request from the processor.

The processor will assert PReq* when it wants to gain ownership of the bus. The external agent can allow the processor to gain bus mastership, perform one read/write request and then give up mastership by: deasserting EReq*, then asserting EReq* and re-arbitrating for the bus while maintaining EOK* asserted. This EReq* deassertion pulse can be a minimum of one cycle in length.

See “Figure 46: Consecutive processor write requests followed by external write request” for an example of an external agent giving up the bus to allow only one read/write request from the processor. Note that the external agent must be ready to accept this request by maintaining EOK* asserted. Otherwise the read/write request will be held off or killed and the processor will give up bus mastership without performing any request. This could lead to starvation of the processor.

Figure 47: External Agent Gives Up Bus for One Processor Request



8.5 Multiple Drivers on the SysAD Bus

In most applications the SysAD bus will be a point to point connection from the processor to a bidirectional registered transceiver in an external agent. For those applications, the SysAD bus has only two possible drivers, the processor and the external agent. However, certain applications may wish to add additional drivers and receivers to the SysAD bus, and allow transmissions to take place over the SysAD bus that the processor is not involved in. To accomplish this the external agent(s) must coordinate the usage of the SysAD bus using the arbitration handshake signals like EReq*, PMaster* and PReq*.

To implement an independent transmission on the SysAD bus that does not involve the processor, the external agent(s) will request the SysAD bus by asserting EReq*. After the processor releases the system interface to slave state, the external agent(s) may allow independent transmission to take place on the SysAD bus, making sure that EValid* input to the processor is not asserted while the transmission is occurring. When the transmission is complete, the external agent(s) de-asserts EReq* to return the system interface to master state. To implement multiple drivers separate Valid lines are required for non-processor chips to communicate.

8.6 Signal codes

System interface commands and data identifiers are encoded in five bits and transmitted from the processor to an external agent or from an external agent to the processor on the SysCmd bus during address and data cycles. When SysCmd(4) is de-asserted, the current cycle is an address cycle and SysCmd(3:0) is a command. When SysCmd(4) is asserted, the current cycle is a data cycle and SysCmd(3:0) is a data identifier.

For commands and data identifiers associated with external requests, all bits and fields have a value or suggested value. For system interface commands and data identifiers associated with processor requests, reserved bits and reserved fields in the command or data identifier are undefined, except where noted.

For all system interface commands SysCmd specifies the system interface request type. The encoding of SysCmd(4) for system interface commands is illustrated in “Table 8: Encoding of system interface commands SysCmd(4)”.

Table 8: Encoding of system interface commands SysCmd(4)

SysCmd(4)	Command
0	Address Cycle.
1	Data Cycle.

For address requests, the remainder of the SysCmd bus specifies the attributes of the address request. SysCmd(3) encodes the address request type. SysCmd(2:0) indicates the size of the address requests. The encoding of SysCmd(3:2) for address requests is shown in “Table 8: Encoding of system interface commands SysCmd(4)”. The encoding of SysCmd(1:0) for block or single address requests is shown in “Table 10: Encoding of SysCmd(1:0) for block address requests” and “Table 11: Encoding of SysCmd(1:0) for single address requests”, respectively.

Table 9: Encoding of SysCmd(3) & SysCmd(2) for Address Cycle

SysCmd(3)	Command
0	Read Request.
1	Write Request.
SysCmd(2)	**Request Size.**
0	Single data.
1	Block data.

Table 10: Encoding of SysCmd(1:0) for block address requests

SysCmd(1:0)	Block size.
0	Two words.
1	Four words.
2	Eight words.
3	Reserved.

Table 11: Encoding of SysCmd(1:0) for single address requests

SysCmd(1:0)	Data size.
0	One byte valid. (Byte).
1	Two bytes valid. (Half Word).
2	Three bytes valid. (Tri-Byte).
3	Four bytes valid. (Single Word).

The encoding of SysCmd(3:0) for processor data identifiers is illustrated in “Table 12: Encoding of SysCmd(3:0) for processor data identifiers”. The encoding of SysCmd(3:0) for external data identifiers is illustrated in “Table 13: Encoding of SysCmd(3:0) for external data identifiers”.

Table 12: Encoding of SysCmd(3:0) for processor data identifiers

SysCmd(3)	Last data element indication.
0	Last data element.
1	Not the last data element.
SysCmd(2)	**Reserved.**
SysCmd(1)	**Reserved for: Good data indication.**
	Processor drives 0 (Data is error free).
SysCmd(0)	**Reserved for: Data checking enable.**
	Processor drives 1 (Disable data checking).

Table 13: Encoding of SysCmd(3:0) for external data identifiers

SysCmd(3)	**Last data element indication.**
0	Last data element.
1	Not the last data element.
SysCmd(2)	**Response data indication.**
0	Data is response data.
1	Data is not response data.
SysCmd(1)	**Reserved for: Good data indication.**
0	Data is error free.
1	Data is erroneous.
SysCmd(0)	**Reserved for: Data checking enable.**
	Processor ignores this field (Suggested drive of 1, disable data checking)

Note: External read requests for processor resources is not supported in R4300.

8.7 Physical Addresses

Physical addresses are driven on all 32 bits (bits 31 through 0) of the SysAD bus during address cycles. Addresses associated with single read and write requests are aligned for the size of the data element. Specifically, for single word requests, the low order two bits of the address will be zero, and for half-word requests, the low order bit of the address will be zero. For byte and tri-byte requests the address provided will be a byte address.

External agents returning read response data must support sub-block ordering. Addresses associated with block read requests are aligned to the word of the desired data. The order in which data is returned in response to a processor block read request is the word containing the addressed data word first, followed by the remaining word(s) in the block.

Block writes are always block aligned.

8.8 Processor Reset and Initialization

The R4300 processor has two reset signals, ColdReset* and Reset*. Unlike the R4000, there is no mode interface; all needed modes are directly controlled by pins on the package (i.e. DivMode for clocks) or by software through the coprocessor 0 Configuration Register (i.e. Endianness, Data Rate).

8.8.1 Cold Reset

Cold Reset is used to completely reset the processor, including clocks. Because of this there is no guarantee of any chip state, except for some bits in the status register and configuration register. Those bits are TS, SR, RP, DC and EP(3:0), which are zero, and ERL, BEV, and BE, which are one.

Once power to the processor is established, ColdReset* must be asserted for a minimum of 64,000 Masterclock cycles to insure time for the on processor clocks to lock to MasterClock. (This is 1.6ms at 40MHz). ColdReset* can be asserted & deasserted asynchronously with the rising edge of MasterClock.

After ColdReset* is deasserted (and Reset* is not being asserted), the processor will branch to the Reset Exception Vector and begin the Cold Reset exception. Note that upon resetting the processor, it becomes the bus master and will drive the SysAD bus.

8.8.2 Warm Reset (also known as Soft Reset)

Warm Reset will reset the processor without losing clocks. This is a purely logical reset. The processor will retain as much of its state as possible. Because Warm Reset takes effect immediately upon assertion of the Reset* signal, multicycle operations such as cache misses or floating point may be abandoned and some loss of data may result.

Warm Reset is started by assertion of the Reset* pin. It must be asserted for a minimum of 16 cycles, and must be asserted & deasserted synchronously with MasterClock, meeting setup and hold times. In general, data in the processor will be preserved for debugging purposes. The TS and RP bits will be set to zero, and SR, ERL, and BEV will be set to one.

Immediately upon being Reset, the processor will branch to the Reset Exception Vector, resume ownership of the SysAD bus and begin driving. If Reset* is asserted in the middle of a SysAD transaction, for example to recover a hung bus, care must be taken to also reset external agents to avoid drive fights on the SysAD bus.

8.8.3 Non Maskable Interrupt (NMI)

The processor can also be forced to branch to the Reset Exception vector by an NMI. To software, this condition is indistinguishable from Soft Reset. However, NMI will only take effect when the processor pipeline is running. Thus NMI can be used to recover the processor from a software hang (e.g., infinite loop) but cannot be used to recover the processor from a hardware hang (e.g. no read response from an external agent). NMI cannot cause drive fights on the SysAD bus and no reset of external agents should be required.

8.8.4 General Reset Information

Note that after resetting the processor, it will be the bus master and drive the SysAD bus. Care must be taken to coordinate reset with other system elements. In general, bus errors immediately before, during, or after Reset may result in unexpected behavior. Also, a very small amount of processor state is guaranteed after a reset of the R4300 processor, meaning extreme care must be taken to correctly initialize the processor via software. The R4300 processor also differs from the R4000 in that only sixteen cycles of Reset* assertion are required.

R4300 differs also from the R4200 processor in setting up the configuration register mode bits. What used to be external pins on R4200 are now software controlled Config register bits which are set to a default value with the de-assertion of Cold Reset and are unaffected by Warm Reset or NMI. Upon Cold Reset, the default modes of the R4300 are BigEndian and Data Rate = D. Since instruction fetches are single word wide, the processor can fetch instructions even from a Little Endian system memory and the first instructions could reset the Config (BE) bit to the system endianness mode. The Config (EP) field, which specifies the data rate should also be set for those systems that only support the slow mode, before any write instruction is executed.

9.0 Exception Handling

The exception handling system is responsible for efficiently handling 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.

9.1 Exception operation

To handle an exception, the processor forces execution of a handler at a fixed address in kernel mode with interrupts disabled. To resume, the PC, operating mode, and interrupt enable must be restored, and thus it is this context that must be saved when an exception is taken.

When an exception occurs, the EPC is loaded with an appropriate restart location at which execution may resume after servicing the exception. The EPC also can be thought of as containing the address of the instruction that caused the exception, or if the instruction was executing in a branch delay slot, the address of the immediate predecessor of the instruction.

The R4300 processor supports a supervisor mode and fast TLB refill for all address spaces. R4300 provides a single interrupt enable (*IE*), a base operating mode (user, supervisor, kernel), an exception level (normal, exception), and an error level (normal, error). Interrupts are enabled with *IE* = 1 and both levels are normal. The operating mode is specified by the base mode when the exception level is normal, and is kernel when exception level is set. Returning from an exception consists of resetting the exception level to normal (see ERET instruction).

9.2 Precision of Exceptions

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 the following instructions are killed, exceptions associated with those instructions are also killed, so that exceptions are not taken in the order detected, but in instruction fetch order.

9.3 Exception Types

The table below lists each of the exception types which are handled by the processor, giving an interpretation of the meaning of each exception.

Table 14: Exception Types

Types	Cause bit	Description
Reset	-	A reset exception, requested by external logic, aborts the current execution stream and starts executing at a unique reset vector provided for this exception.
NMI	-	This is a non-maskable interrupt requested by external logic. The Reset vector is used for this interrupt.
TLB refill	TLBL/TLBS	The referenced address does not match any TLB entry. A separate vector is provided for this exception. This vector is used for all virtual address spaces when the Status register EXL bit is 0.
Extended addressing TLB refill	TLBL/TLBS	The referenced address did not match any TLB entry and the referenced address space is using extended addressing (MIPS III). A separate vector is provided for this exception when the SR exception level (EXL) is 0.
TLB invalid	TLBL TLBS	Virtual-address reference that matches an invalid TLB entry.
TLB modified	Mod	An attempt to write to a virtual address that did not have D bit in the corresponding TLB entry set.
Bus error	IBE/DBE	An external interrupt signaled by bus interface circuitry. A bus error is signaled for events such as bus time-out, and invalid memory addresses or access types.
Address error	AdEL/AdES	An attempt is made to load, fetch, or store a word not aligned on a word boundary or load or store a halfword not aligned on a halfword boundary, or load or store a doubleword not aligned on a doubleword boundary, or to reference a privileged virtual address.
Integer overflow	Ov	An add or subtract operation causes two's complement overflow.
Trap	Tr	A trap operation was executed with a true condition.
System call	Sys	Execution of a SYSCALL instruction.
Breakpoint	Bp	Execution of a BREAK instruction.
Reserved Instruction	RI	Execution of an instruction with a reserved major operation code (bits 31..26), or a SPECIAL instruction with a reserved minor operation code (bits 5..0).
Coprocessor Unusable	CpU	Execution of a coprocessor instruction for which the corresponding coprocessor-usable bit was not set.
Floating Point	FPE	One of several floating-point exceptions. See chapter 3.
Interrupt	Int	One of several interrupt conditions. See the Cause register.
Watch	WATCH	Reference to WatchHi/WatchLo address.

9.4 Exception vectors

The Reset, Soft Reset, and NMI exceptions are always vectored to 0xffff ffff bfc0 0000. The address for other exceptions is a combination of a vector offset and a base address determined by the *BEV* bit of the Status Register.

Table 15: Exception Vectors

Exception	Vector Base	Vector Offset
Reset, Soft Reset, NMI	0xffff_ffff_bfc0_0000	0
TLB Refill, EXL=0	0xffff_ffff_8000_0000(BEV=0) 0xffff_ffff_bfc0_0200 (BEV=1)	000
XTLB Refill, EXL=0	0xffff_ffff_8000_0000(BEV=0) 0xffff_ffff_bfc0_0200 (BEV=1)	080
Other	0xffff_ffff_8000_0000(BEV=0) 0xffff_ffff_bfc0_0200 (BEV=1)	180

9.5 Priority of Exceptions

When multiple exceptions can occur for a single instruction, only one exception is reported, with priority given in the following order:

Reset
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
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

9.5.1 Reset

Cause:

A Reset exception, also known as Cold Reset exception, occurs when the ColdReset* pin transitions from assertion to deassertion. Cold Reset clears all state machines, and leaves the SR bit of the Status Register cleared. CP0 register bits TS, SR, RP and EP(3:0) are initialized to zero, and bits ERL, BEV, and BE are initialized to one. There is no guarantee of state for the rest of the chip. Clocks cannot be counted on during a cold reset. This exception is not maskable.

Handling:

A special exception vector (0xffff ffff bfc0 0000) is provided for this exception. This vector is located within the unmapped and uncached address space so that the cache and TLB need not be initialized to handle this.

Servicing:

The Cold Reset exception is serviced by initializing all processor registers, coprocessor registers, the caches and the memory system, performing diagnostic tests, and bootstrapping the operating system. The reset exception vector is selected to appear within the uncached, unmapped memory space of the machine so that instructions may be fetched and executed while the cache and virtual memory system are still in an undefined state.

9.5.2 Soft Reset

Cause:

A soft reset, sometimes called warm reset, occurs in response to the Reset* pin on the chip transitioning from assertion to deassertion WITHOUT ColdReset* pin being asserted immediately before. So for a soft reset, there must be at least one cycle where neither Reset* nor ColdReset* were asserted. A soft reset immediately resets all state machines, and sets the SR bit of the Status Register. Currently executing operations may be abandoned but, in general, data in the processor will be preserved for debugging purposes. CP0 register bits TS and RP bits will be set to zero, and bits ERL and BEV will be set to one.

Execution begins at the reset vector in response to the Reset* pin being deasserted. This exception is not maskable.

Handling:

A special exception vector (0xffff ffff bfc0 0000) is provided for this exception, same as the Cold Reset exception. This vector is located within the unmapped and uncached address space so that the cache and TLB need not be initialized to handle this.

To differentiate Soft Reset from Cold Reset exception, the SR bit of the Status Register is set.

Servicing:

The Soft Reset exception is serviced by saving as much of the current processor state for diagnostic purposes, and then reinitializing as if for the Cold Reset Exception.

9.5.3 Non-maskable Interrupt

Cause:

The Non-Maskable interrupt *NMI* exception occurs in response to the falling edge of the Non-maskable Interrupt pin, NMI*. An NMI is treated as an interrupt. Upon getting an NMI, R4300 jumps to the reset exception vector and sets the SR bit in the status register. No state machines or other bits in the chip are affected.

This exception is not maskable; it occurs regardless of the settings of the *EXL*, *ERL*, and *IE* Status register bits.

Handling:

The Reset exception vector (0xffff ffff bfc0 0000) is used for this exception. This vector is located within the unmapped and uncached address space so that the cache and TLB need not be initialized to handle this exception. The SR bit of the Status register is set to differentiate this exception from Cold Reset.

Unlike Reset, but like other exceptions, NMI is taken only at instruction boundaries; thus the state of the caches and memory system are preserved by this exception. The caches, TLB, and normal exception vectors need not be properly initialized. The contents of all registers are preserved when this exception occurs, except for the ErrorEPC register, which contains the restart PC, and the *ERL* bit of the Status register, which is set to one.

Servicing:

The NMI exception is serviced by saving the current processor state for diagnostic purposes, and reinitializing as for the Reset exception.

9.5.4 TLB Refill and Extended addressing TLB Refill

Cause:

The TLB refill exception occurs when no TLB entry matches a reference to a mapped address space. This exception is not maskable.

Handling:

Two special vectors are provided for this exception; one for references to 32-bit address spaces, and one for references to 64-bit address spaces. The UX, SX, and KX bits of the Status register determine whether the user, supervisor, or kernel address spaces are 32-bit or 64-bit spaces. All references use this vector when *EXL* is equal to zero in the Status register.

The *TLBL* or *TLBS* code in the Cause register is set, indicating whether the instruction, indicated by the EPC register and *BD* bit in the Cause register, caused the miss via an instruction reference or load or alternatively, via a store.

When this exception occurs, the BadVAddr, Context, XContext, and EntryHi registers contain the virtual address that failed address translation. The EntryHi register also contains the Address Space Identifier from which the translation fault occurred. The Random register normally contains a valid location in which to put a replacement TLB entry. The contents of the EntryLo register is undefined.

The EPC points at the instruction which caused the exception, unless it is in a branch delay slot. If the instruction is in a branch delay slot, the EPC points at the branch instruction which precedes it, and the *BD* bit of the Cause register is set.

Servicing:

To service this exception, the content of the Context or XContext register is used as a virtual address to fetch a memory word containing the physical page frame and access control bits. The memory word is placed into the EntryLo registers, and the EntryHi and EntryLo registers are written into the TLB.

It is possible that the virtual address used to obtain the physical address and access control information is on a page that is also not resident in the TLB. This is efficiently handled by allowing a TLB refill exception in the TLB refill handler. This second exception goes instead to the common exception vector because the *EXL* bit of the Status register is set.

9.5.5 TLB Invalid

Cause:

The TLB Invalid exception occurs when a virtual address reference matches a TLB entry that is marked invalid. This exception is not maskable.

Handling:

The common exception vector is used for this exception. The *TLBL* or *TLBS* code in the Cause register is set, indicating whether the instruction, indicated by the EPC register and *BD* bit in the Cause register, caused the miss via an instruction reference or load or alternatively, via a store.

When this exception occurs, the BadVAddr, Context, XContext, and EntryHi registers contain the virtual address that failed address translation. The EntryHi register also contains the Address Space Identifier from which the translation fault occurred. The Random register normally contains a valid location in which to put a replacement TLB entry. The contents of the EntryLo register is undefined.

The EPC points at the instruction which caused the exception, unless it is in a branch delay slot. If the instruction is in a branch delay slot, the EPC points at the branch instruction which precedes it, and the *BD* bit of the Cause register is set.

Servicing:

The valid bit of a TLB entry is typically cleared when a virtual address does not exist, or when it exists, but is not in main memory (a page fault), or when a trap is desired on any reference to the page (for example to maintain a reference bit). After servicing the particular cause of this exception, the TLB entry is located with TLBP (TLB Probe), and replaced with an entry with the valid bit set.

9.5.6 TLB Modified

Cause:

The TLB modified exception occurs when a store operation’s virtual address reference to memory matches a TLB entry which is marked valid but not dirty/writable. This exception is not maskable.

Handling:

The common exception vector is used for this exception. The *Mod* code in the Cause register is set.

When this exception occurs, the BadVAddr, Context, and EntryHi registers contain the virtual address that failed address translation. The EntryHi register also contains the Address Space Identifier from which the translation fault occurred. The contents of the EntryLo register is undefined.

The EPC points at the instruction which caused the exception, unless it is in a branch delay slot. If the instruction is in a branch delay slot, the EPC points at the branch instruction which precedes it, and the *BD* bit of the Cause register is set.

Servicing:

The kernel uses the failing virtual address or virtual page number to identify the corresponding access control information. The page identified may or may not permit write accesses, and if not permitted, a *Write Protection Violation* has occurred.

Otherwise, if write accesses are permitted, the page frame is marked as dirty/writable by the kernel in its own data structures. The TLBP instruction is used to place the index of the TLB entry which must be altered into the Index register. The EntryLo register is loaded with a word containing the physical page frame and access control bits (with the D bit set), and the EntryHi and EntryLo registers are written into the TLB.

9.5.7 Bus Error

Cause:

The Bus Error exception occurs when signaled by board-level circuitry. Bus error is signaled for events such as bus time-out, backplane bus parity errors, and invalid physical memory addresses or access types. This exception is not maskable.

This error occurs only for these events when they occur synchronously (cache miss refills, uncached references, and unbuffered writes); a bus error resulting from a buffered write transaction must instead be reported using the general interrupt mechanism.

Handling:

The common interrupt vector is used for this exception. The *IBE* or *DBE* code in the Cause register is set, signifying whether the instruction, indicated by the EPC register and *BD* bit in the Cause register, caused the exception via an instruction reference or alternatively, via a load or store.

The EPC points at the instruction which caused the exception, unless it is in a branch delay slot. If the instruction is in a branch delay slot, the EPC points at the branch instruction which precedes it, and the *BD* bit of the Cause register is set.

Servicing:

The physical address at which the fault occurred may be computed from information available in the system control coprocessor registers. If the *IBE* code in the Cause register is set (instruction fetch reference), the virtual address is contained in the EPC register. If the *DBE* code set (load or store reference), the instruction which caused the exception is located at the virtual address contained in the EPC register. If the BD bit of the Cause register is set along with either the IBE or DBE code then the virtual address of the instruction which caused the exception is four plus the contents of the EPC register. The virtual address of the load or store reference can then be obtained by interpreting the instruction. The physical address can be obtained by using the TLBP instruction and reading the EntryLo register to compute the physical page number.

The process executing at the time is handed a bus error signal. This error is usually fatal.

9.5.8 Address Error

Cause:

The Address Error exception occurs when an attempt is made to load, fetch or store a word which is not aligned on a word boundary, or to load or store a halfword which is not aligned on a halfword boundary, or to load or store a doubleword which is not aligned on a doubleword boundary, or to reference a kernel address space from user or supervisor mode, or a supervisor address space from user mode. This exception is not maskable.

Handling:

The common exception vector is used for this exception. The *AdEL* or *AdES* code in the Cause register is set, indicating whether the instruction, indicated by the EPC register and *BD* bit in the Cause register, caused the exception via an instruction reference or load or alternatively, via a store.

When this exception occurs, the BadVAddr register contains the virtual address that was not properly aligned or which referenced a protected address space. The contents of the VPN field of the Context and EntryHi registers is undefined. The contents of the EntryLo register is undefined.

The EPC points at the instruction which caused the exception, unless it is in a branch delay slot. If the instruction is in a branch delay slot, the EPC points at the branch instruction which precedes it, and the *BD* bit of the Cause register is set.

Servicing:

The process executing at the time is handed a segmentation violation signal. This error is usually fatal.

9.5.9 Integer overflow

Cause:

The Integer overflow exception occurs when an ADD, ADDI, SUB, DADD, DADDI, or DSUB instruction results in two’s complement overflow. This exception is not maskable.

Handling:

The common exception vector is used for this exception. The *OV* code in the Cause register are set.

The EPC points at the instruction which caused the exception, unless it is in a branch delay slot. If the instruction is in a branch delay slot, the EPC points at the branch instruction which precedes it, and the BD bit of the Cause register is set.

Servicing:

The process executing at the time is handed an integer overflow signal. This exception may be fatal.

9.5.10 Trap

Cause:

The Trap exception occurs when a TGE, TGEU, TLT, TLTU, TEQ, TNE, TGEI, TGEUI, TLTI, TLTUI, TEQI, or TNEI instruction results in a true condition. This exception is not maskable.

Handling:

The common exception vector is used for this exception. The *Tr* code in the Cause register are set.

The EPC points at the instruction which caused the exception, unless it is in a branch delay slot. If the instruction is in a branch delay slot, the EPC points at the branch instruction which precedes it, and the *BD* bit of the Cause register is set.

Servicing:

The process executing at the time is handed an integer overflow signal. This exception may not be fatal.

9.5.11 System Call

Cause:

The system call exception occurs when an attempt is made to execute the SYSCALL instruction. This exception is not maskable.

Handling:

The common exception vector is used for this exception. The *Sys* code in the Cause register is set.

The EPC points at the SYSCALL instruction, unless it is in a branch delay slot. If the SYSCALL instruction is in a branch delay slot, the EPC points at the branch instruction which precedes it.

If the SYSCALL instruction is in a branch delay slot, the *BD* bit of the Status register (SR) is set, otherwise it is cleared.

Servicing:

Control is transferred to the applicable system routine. To resume execution, the EPC must be altered so that the SYSCALL instruction is not re-executed; this is accomplished by adding 4 to the EPC register before returning. Note that if a SYSCALL instruction is in a branch delay slot, a more complicated algorithm would be required.

9.5.12 Breakpoint

Cause:

The Breakpoint exception occurs when an attempt is made to execute the BREAK instruction. This exception is not maskable.

Handling:

The common exception vector is used for this exception. The *BP* code in the Cause register is set.

The EPC points at the BREAK instruction, unless it is in a branch delay slot. If the BREAK instruction is in a branch delay slot, the EPC points at the branch instruction which precedes it.

If the BREAK instruction is in a branch delay slot, the *BD* bit of the Status register (SR) is set, otherwise it is cleared.

Servicing:

Control is transferred to the applicable system routine. Additional distinctions may be made on the basis of the otherwise unused bits of the BREAK instruction (bits 25..6), by loading the contents of the instruction pointed at by the EPC register. (A value of 4 must be added to the contents of the EPC register to locate the instruction if it resides in a branch delay slot.)

To resume execution, the EPC must be altered so that the BREAK instruction is not re-executed; this is accomplished by adding 4 to the EPC register before returning. Note that if a BREAK instruction is in a branch delay slot, interpretation of the branch instruction would be required in order to resume execution.

9.5.13 Reserved Instruction

Cause:

The Reserved Instruction exception occurs when an attempt is made to execute an instruction whose major opcode (bits 31..26) is undefined or a SPECIAL instruction whose minor opcode (bits 5..0) is undefined. On the R4300 processor, this exception also occurs on REGIMM instruction whose minor opcode (bits 20..16) is undefined. This exception also occurs on MIPS III opcodes when the processor is in user mode and UX=0 in the Status register and when the processor is in supervisor mode and SX=0 in the Status register. This exception is not maskable.

This exception provides a mechanism to interpret instructions which are added to or removed from the MIPS processor architecture at a later time.

Handling:

The common exception vector is used for this exception. The *RI* code in the Cause register is set.

The EPC points at the reserved instruction, unless it is in a branch delay slot. If the reserved instruction is in a branch delay slot, the EPC points at the branch instruction which precedes it.

Servicing:

In current systems, no defined instructions in the architecture are interpreted. The process executing at the time is handed an illegal instruction signal. This error is usually fatal.

9.5.14 Coprocessor Unusable

Cause:

The Coprocessor Unusable exception occurs when an attempt is made to execute a coprocessor instruction for which the corresponding coprocessor unit has not been marked usable, or for coprocessor zero instructions, when the unit has not been marked usable and the process is executing in user mode. This exception is not maskable.

Handling:

The common exception vector is used for this exception. The *CpU* code in the Cause register is set.

The contents of the Coprocessor Usage Error field of the Coprocessor Control register indicates which of the four coprocessors was referenced.

The EPC points at the unusable coprocessor instruction, unless it is in a branch delay slot. If the unusable coprocessor instruction is in a branch delay slot, the EPC points at the branch instruction which precedes it.

Servicing:

The coprocessor unit to which an attempt was made to reference is identified from the Coprocessor Usage Error field. If the process is entitled to access, the coprocessor is marked usable and the corresponding user state is restored into the coprocessor.

If the process is entitled to access to the coprocessor, but it is known not to exist or to have failed, interpretation of the coprocessor instruction is possible. If the *BD* bit is set in the Cause register, the branch instruction must be interpreted; then the coprocessor instruction may be emulated and execution resumed with the EPC advanced past the coprocessor instruction.

If the process is not entitled to access to the coprocessor, the process executing at the time is handed a privileged fault signal. This error is usually fatal.

9.5.15 Interrupt

Cause:

The Interrupt exception occurs when one of the eight interrupt conditions are asserted. The significance of these interrupts is implementation-dependent.

Each of the eight interrupts may be masked by clearing the corresponding bit in the IntMask field of the Status register. All of the eight interrupts may be masked at once by clearing the *IE* bit of the Status register.

Handling:

The common exception vector is used for this exception. The *Int* code in the Cause register is set.

The *IP* field of the Cause register indicates the current interrupt requests. It is possible that more than one of the bits will be set at once, or even that no bits are set (if an interrupt is asserted and then deasserted before this register is read).

Servicing:

If the interrupt is caused by one of the two software-generated exceptions, the interrupt condition is cleared by setting the corresponding Cause register bit to zero.

If the interrupt is hardware-generated, the interrupt condition is cleared by alleviating the condition which is causing the corresponding interrupt pin to be asserted. The manner in which this is accomplished is implementation-dependent.

9.5.16 Watch

Cause:

The Watch exception occurs when a load or store instruction references the physical address specified in the WatchLo and WatchHi system control coprocessor registers. The WatchLo register also specifies whether loads, stores, both, or neither initiate this exception. The CACHE instruction never causes a WATCH exception. The exception is postponed while the *EXL* bit is set in the Status register. This exception is maskable only by setting *EXL* in the Status register.

Handling:

The common exception vector is used for this exception. The *WATCH* code in the Cause register is set.

Servicing:

This exception is intended as a debugging aid. Typically the exception handler will transfer control to a debugger, allowing the user to examine the situation. To continue, the Watch must be disabled for the execution of the faulting instruction and then re-enabled. Execution of the faulting instruction may be accomplished by interpretation, or by setting breakpoints.

9.5.17 Floating Point

Cause:

The Floating Point Exception is used by the floating point coprocessor. Other implementations use one of the hardware interrupts for this exception. This exception is not maskable.

Handling:

The common exception vector is used for this exception. The *FPE* code in the Cause register is set.

The contents of the Floating Point Control Status register indicates the cause of this exception.

Servicing:

This exception is cleared by clearing the appropriate bit in the Floating Point Control Status register. For an unimplemented exception, the kernel should emulate the instruction. For other exceptions, the kernel should pass the exception to the user.

10.0 Clocks

The clocks on the R4300 chip are controlled via an on-chip Phase Locked Loop circuit. This circuit will keep the R4300 chip's internal clock edges aligned with the clock edges of the *MasterClock* signal, which acts as the system master clock.

Inside the R4300 chip, the *MasterClock* signal will be multiplied by a factor determined by DivMode(1:0) inputs to the processor, and then all internal clocks will be derived by dividing that signal down. The R4300 chip has two primary internal clocks, the pipeline clock *PClock*, and the system interface clock *SClock*. While other internal edges may be generated, these will be transparent to the user. *SClock* will be the same frequency and phase as *MasterClock*.

R4300 will provide an external clock available to the system designers and users of the R4300 chip. This clock is the transmit clock *TClock*. *TClock* will be aligned with R4300's *SClock*, and have the same frequency.

10.1 PClock

The pipeline clock *PClock* can be equal to 1x, 1.5x, 2x or 3x the *MasterClock* frequency. This multiplication factor is determined by DivMode(1:0) pins, which are static signal inputs to R4300.

10.2 SClock

The system interface clock, *SClock*, is the same as *MasterClock* frequency. *SClock* is always derived from *PClock*.

10.3 TClock

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.

10.4 Phase Locked Loop

The R4300 clocks are controlled by a phase locked loop circuit. The phase locked loop circuitry can be disabled by a *BypassPLL* pin. In this mode, the internal PClock and SClock will be derived by dividing the clock driven at the *MasterClock* pin of the R4300 processor. This mode can be used for test purposes, as well as in systems whose MasterClock is running too slowly for the phase locked loop circuit to lock to.

10.5 SyncIn/SyncOut

The processor generates *SyncOut* at the same frequency as *MasterClock* and aligns the output of the *SyncIn* input buffer with *MasterClock*. *TClock* is generated at the same frequency as *MasterClock* and aligned with SyncOut.

SyncOut must be connected to *SyncIn* either directly or through an external buffer. The processor can compensate for both output driver and input buffer delays (and, when necessary, delay caused by an external buffer) when aligning *SyncIn* with *MasterClock*.

10.6 Reduced Power Mode

See section "7.6.1 *Reduced Power Mode RP (bit 27 in Status Register)*" for details on this mode.

11.0 JTAG Interface

The R4300 Processor provides a test interface using the JTAG (IEEE Std.1149.1/D6) protocol. R4300 JTAG is fully IEEE compliant.

11.1 JTAG Signals

R4300 implements the standard Test Access Port (TAP). This includes the following four signals:

JTDI: (i)JTAG serial Data In

JTDO:(o)JTAG serial Data Out

JTMS:(i)JTAG Mode Select

JTCK:(i)JTAG Clock

11.2 JTAG Functionality

The minimal JTAG functionality includes the TAP controller, JTAG instruction register, JTAG boundary scan, and a JTAG bypass register.

The TAP controller state machine monitors the JTMS signal and determines the functionality to be implemented. This includes either loading the JTAG instruction register (IR), or beginning a serial data scan through a data register (DR). As the data is scanned in, the state of the JTMS pin will signal new data word, as well as the end of the data stream. The data register to be selected is determined by the contents of the instruction register.

11.2.1 TAP Controller

The Tap Controller implements the JTAG protocol as described in the IEEE standard.

11.2.2 Instruction Register

The instruction register enables the user to select the boundary scan register or the bypass register and any R4300 specific test features that are implemented. The IR register encodings are:

IR	Selection
000	Boundary Scan Register
1XX	Bypass Register
X1X	Bypass Register
XX1	Bypass Register
011	Set Cache_Test_Mode Sticky bit

11.2.3 Bypass Register

In accordance with the IEEE standard, a bypass register is implemented. This one-bit register latches the input from JTDI, and drives its output to JTDO.

11.2.4 Boundary Scan Register

The boundary scan register includes most of the inputs and outputs of the R4300 processor (except some timing critical signals like clocks and pll signals). The pins of the R4300 chip may be configured to drive any arbitrary pattern by scanning into the boundary scan register from the Shift-DR state. Incoming data to the processor may be examined by loading the boundary scan register while in the Capture-DR state and then scanning it out. following is the order of signals in the boundary scan register:

[JTDI] SysAD<4> SysAD<3> SysAD<2> SysAD<1> SysAD<0> PReqB SysAD<31> PValidB SysAD<30> EOKB SysAD<29> SysAD<28> SysAD<27>IntB<2> NMIB SysAD<26> PMasterB SysAD<25> EReqB SysCmd<0> SysCmd<2> ResetB EValidB SysCmd<2> SysCmd<3> ColdResetB SysCmd<4> DivMode<1> SysAD<24> DivMode<0> SysAD<23> IntB<3> SysAD<22> SysAD<21> SysAD<20> TestModeB BypassPLLB TClock SyncOut SysAD<19> SysAD<18> SysAD<17> IntB<4> SysAD<16> SysAD<15> SysAD<14> SysAD<13> SysAD<12> SysAD<11> SysAD<10> IntB<0> SysAD<9> SysAD<8> SysAD<7> SysAD<6> SysAD<5> IntB<1> jSysADEn [JTDO]

12.0 Specifications

12.1 Electrical Characteristics

12.1.1 LVCMOS

R4300 will meet and exceed the JEDEC standard for low-voltage CMOS-compatible VLSI digital circuits (LVCMOS).

12.1.2 DC Characteristics

12.1.2.1 Maximum Ratings

(Operation beyond the limits set forth in this table may impair the useful life of the device.)

Table 16: Maximum Ratings

Parameter	Symbol	Test Conditions	Minimum	Maximum	Units
Supply Voltage	VCC		3.0	3.6	V
Input Voltage	VIN		-.5[1]	VCC + 0.5	V
Storage Temperature	TST		-65	+150	C
Operating Temperature	TC	Case temperature	0	+85	C

Note:

(1) VIN Min. = -3.0V for pulse width less than 15ns.

(2) Not more than one output should be shorted at a time. Duration of the short should not exceed 30 seconds.

12.1.2.2 Operating Parameters

Table 17: Operating Parameters

Parameter	Symbol	Conditions	62.5 MHz		Units
			Minimum	Maximum	
Output HIGH Voltage	VOH	VCC = Min. IOH=-4ma	2.4		V
Clock Output HIGH Voltage[3]	VOHC	VCC = Min. IOH=-4ma	2.7		V
Output LOW Voltage	VOL	VCC = Min. IOL=4ma		.4	V
Input HIGH Voltage[2]	VIH		2	VCC+.5	V
Input LOW Voltage[1,2]	VIL		-.5(1)	.8	V
MasterClock Input HIGH Voltage	VIHC		0.8 VCC	VCC+.5	V
MasterClock Input LOW Voltage	VILC		-.5(1)	0.2 VCC	V
Input Capacitance	CIn			10	pF
Output Capacitance	COut			10	pF
Operating Current	ICC	VCC = 3.0V, TC=0C		0.67	A
Input Leakage	ILeak			10	µA
Input/Output Leakage	IOLeak			20	µA

Note:

(1) VIL Min. = -3.0V for pulse width less than 15ns.

(2) Except for MasterClock input

(3) Applies to TClock output

12.1.3 AC Characteristics

Notes:

All output timings are given assuming 50pf of capacitive load. Output timings should be derated where appropriate as per the table below.

12.1.3.1 MasterClock and Clock Parameters

Table 18: MasterClock and Clock Parameters

Parameter	Symbol	Test Conditions	62.5 MHz		Units
			Minimum	Maximum	
MasterClock High	t_{MCHigh}	Transition ≤ 5ns	4		ns
MasterClock Low	t_{MCLow}	Transition ≤ 5ns	4		ns
MasterClock Freq[1,2]			20	62.5	MHz
MasterClock Period	t_{MCP}		16	50	ns
Clock Jitter	$t_{MCJitter}$			+/-500	ps
MasterClock Rise Time	t_{MCRise}			4	ns
MasterClock Fall Time	t_{MCFall}			4	ns
JTAG Clock Period	$t_{JTAGCKP}$		$4*t_{MCP}$		ns

Note:

(1) Operation of R4300 is only guaranteed with the Phase Lock Loop enabled.

12.1.3.2 System Interface Parameters

Table 19: System Interface Parameters

Parameter	Symbol	Test Conditions	62.5 MHz		Units
			Minimum	Maximum	
Data Output [1,2,3]	t_{DO}		2	8	ns
Data Setup [3]	tDS		3.5		ns
Data Hold [3]	tDH		1.5		ns
Clock Rise Time[4]	tCORise			4	ns
Clock Fall Time[4]	tCOFall			4	ns
Clock High Time[4]	tCOHigh		4		ns
Clock Low Time[4]	tCOLow		4		ns

Note:

(1) Timings are measured from 1.5V of the SClock to 1.5V of signal.

(2) Capacitive load for all output timings besides Status is 50pf.

(3) Data Output, Data Setup and Data Hold apply to all logic signals driven out of or driven into the R4300 on the system interface. Clocks are specified separately.

(4) TCLock.

12.1.3.3 Capacitive Load Deration

Table 20: Capacitive Load Deration

Parameter	Symbol	62.5 MHz		Units
		Min.	Max.	
Load Derate	CLD		2	ns/25pF

12.1.4 Timing Diagrams

Figure 48: MasterClock

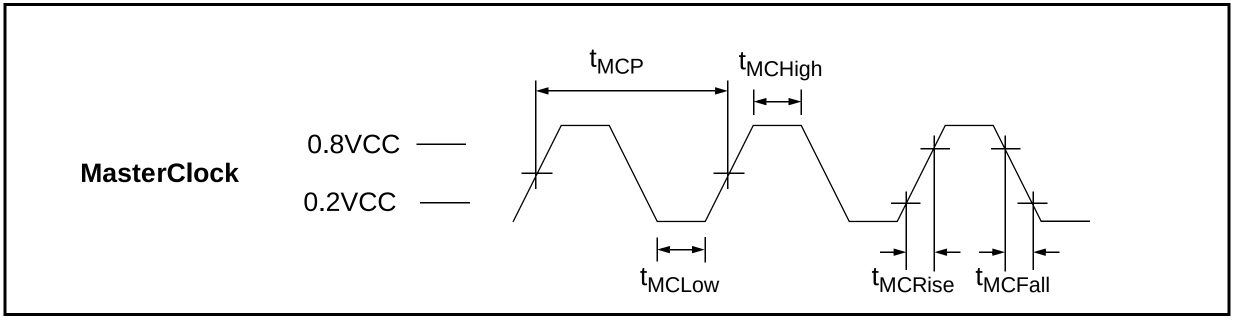


Figure 49: TClock

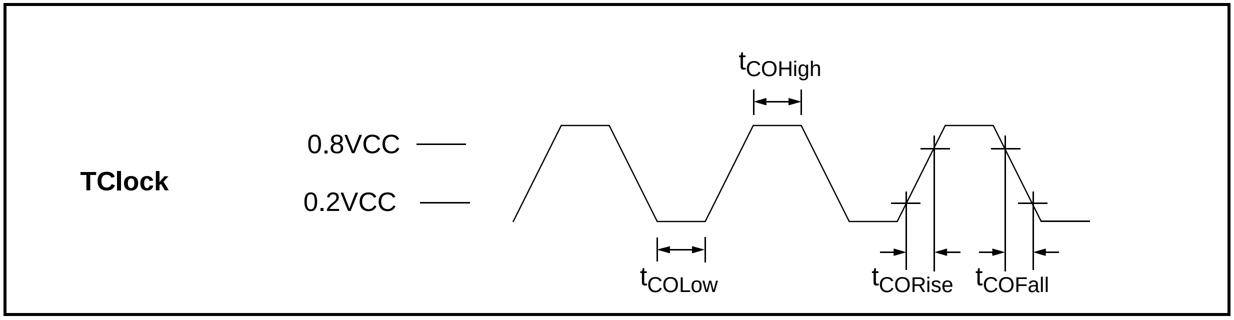
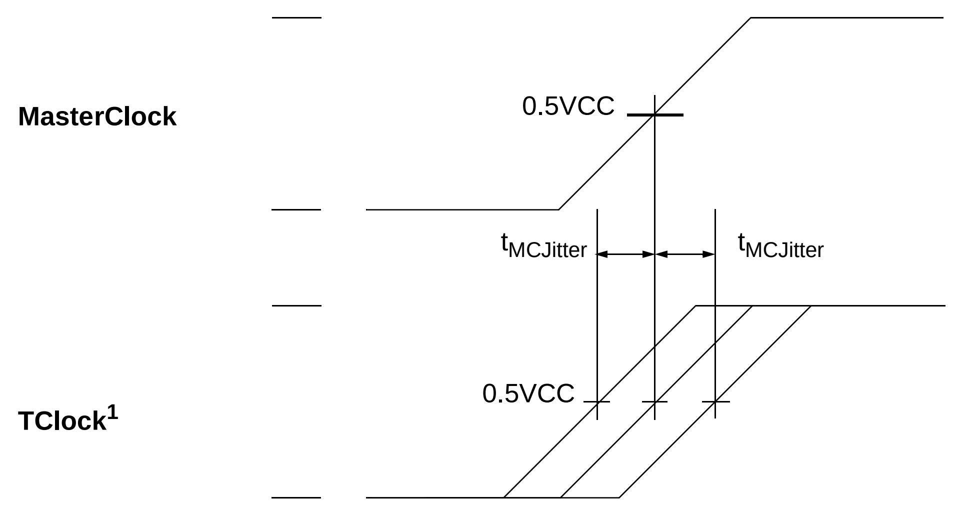


Figure 50: Clock Jitter



(1) With SyncOut shorted to SyncIn by the shortest path, the 50% point of TClock lines up with the 50% point of MasterClock

Note: The SyncIn/SyncOut path and TClock must have the same capacitive loading to match the use of TClock with the MasterClock edges.

Figure 51: Processor clock, PClock to SClock divisor of 2

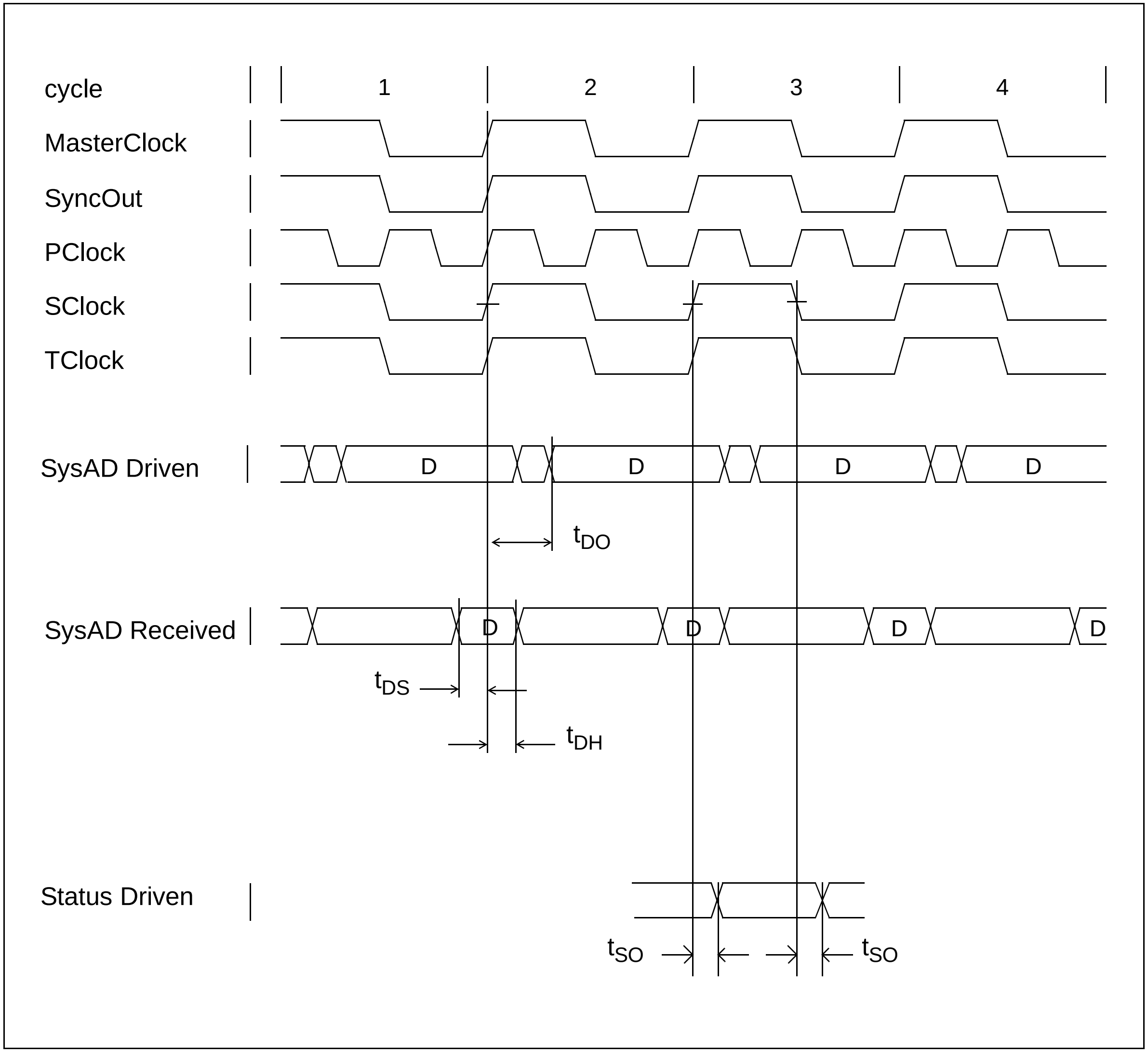
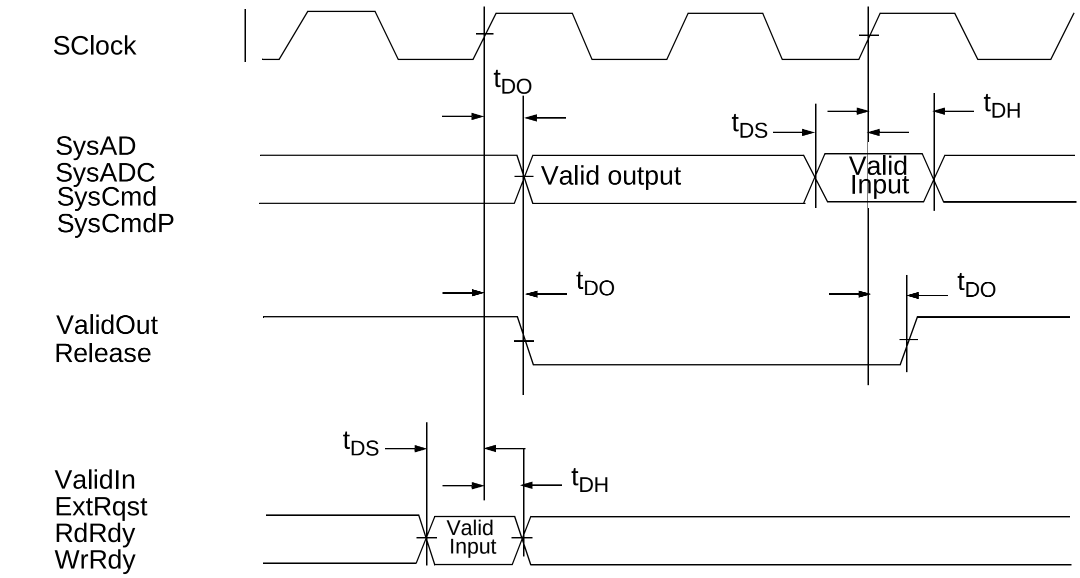


Figure 52: System Interface Edge Timing Relationships



Note: These waveforms only describe edge to edge timing relationships. Functional descriptions are contained in previous chapters.

12.2 PLL Passive Components.

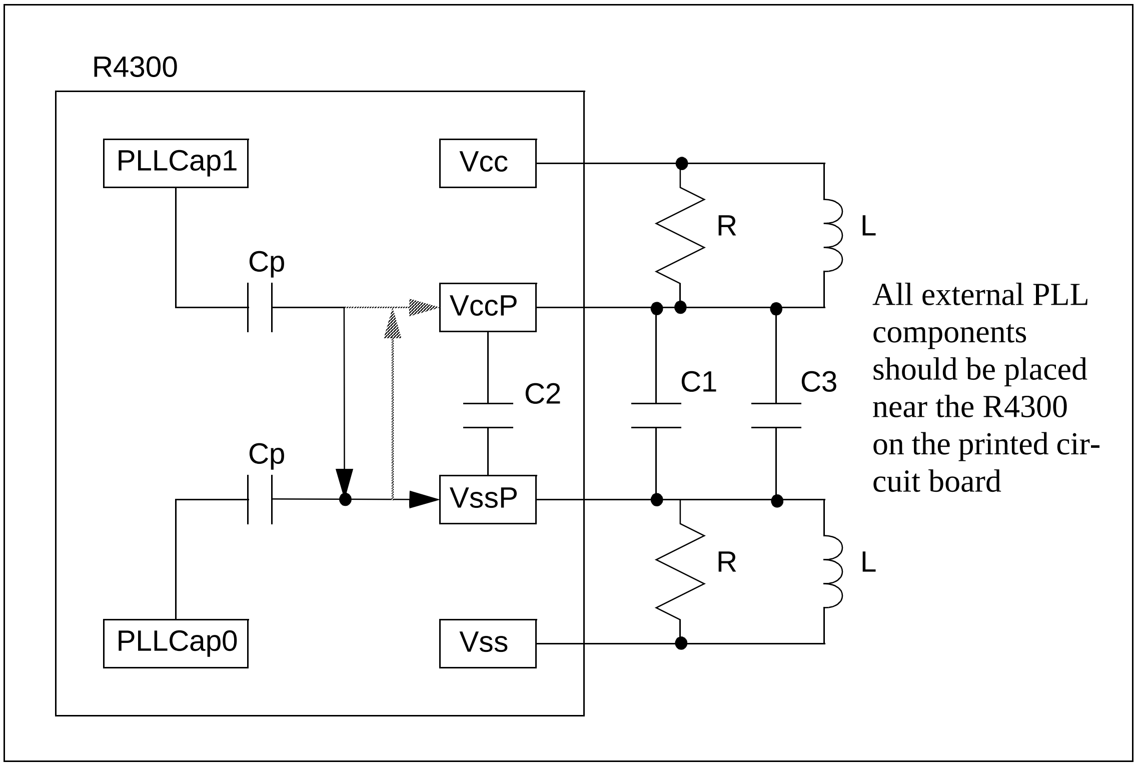
The Phase Locked Loop circuit requires several passive components for proper operation, which are connected to PLLCap0, PLLCap1, VccP, and VssP, as illustrated in “Figure 53: External PLL Passive Components.” The capacitors for PLLCap0 (Cp) and PLLCap1(Cp) can be connected to either VssP (as shown), VccP, or on to VssP and one to VccP.

It is essential to isolate the analog power and ground of the PLL circuit (VccP/VssP) from the regular power and ground (Vcc/Vss). Initial evaluations have yielded good results with the following values:

R	=	5 ohms
C1	=	1 nF
C2	=	82 nF
C3	=	10 uF
Cp	=	470pF

Since the optimum values for the filter components depend upon the application and the system noise environment, these values should be considered as starting points for further experimentation within your specific application. In addition, the chokes (inductors: L) can be considered for use as an alternative to the resisters (R) for use in filtering the power supply.

Figure 53: External PLL Passive Components.



All external PLL components should be placed near the R4300 on the printed circuit board

12.3 Pin Descriptions

SysAD(31:0): (i/o) Multiplexed address and data transfer bus between the processor and an external agent.

SysCmd(4:0): (i/o) Used for command and data identifier transmission between the processor and an external agent.

EValid*: (i) Signals that an external agent is driving a valid address or valid data on the SysAD bus and a valid command or data identifier on the SysCmd bus during this cycle.

PValid*: (o) Signals that the processor is driving a valid address or valid data on the SysAD bus and a valid command or data identifier on the SysCmd bus during this cycle.

EReq*: (i) Signals that an external agent requests system interface bus ownership.

PReq*: (o) Signals that the processor requests system interface bus ownership. Also, when the processor experiences a protocol error (i.e. the processor detects that an external agent has preformed an action in violation of the SysAD protocol), the processor will continuously toggle PReq*.

PMaster*: (o) Signals that the processor is the master of the system interface bus.

EOK*: (i) Signals that an external agent is capable of accepting a processor request,

Int(4:0)*: (i) General processor interrupt. These are visible as bits 14 to 10 of the Cause register.

NMI*: (i) Non-maskable interrupt.

Reset*: (i) When asserted, initiates an maintains a warm reset in the processor.

TClock: (o) Transmit clock at the operation frequency of the system interface. Equal in frequency and phase to MasterClock.

MasterClock: (i) Master clock input at the operation frequency of the system interface.

SyncOut**:** (o) Synchronization clock output.

SyncIn: (i) Synchronization clock input.

ColdReset***:** (i) When asserted, this signal indicates to the R4300 processor that the +3.3 volt power supply is stable and the R4300 chip should initiate a cold reset sequence. The assertion of ColdReset* will reset the PLL. Asynchronous.

JTDI: (i) JTAG serial data in.

JTDO: (o) JTAG serial data out.

JTMS: (i) JTAG command signal, signals that the serial data in is command data.

JTCK: (i) JTAG serial clock input.

BypassPLL*: (i) This signal forces MasterClock to bypass the PLL and to feed directly to the clock buffers. This should be used for test only. This signal will be implemented as a non-bonding pad (default deasserted) and may not be a pin on the production package. However this pin will be part of the JTag scan chain.

TestMode* (i) This is used for testing cache directly. This must be deasserted (connected to VCC) for normal operation. This signal will be implemented as a non-bonding pad (default deasserted) and may not be a pin on the production package. However this pin will be part of the JTag scan chain.

DivMode(1:0) (i) These signals are an encoding of the PClock to MasterClock ratios. TClock (system interface clock) will be the same frequency as MasterClock. The encoding of DivMode for an example of 40MHz MasterClock is shown below:

DiveMode(1:0)	MasterClock	TClock	PClock	Ratio
00	40MHz	40MHz	40MHz	1:1
01	40MHz	40MHz	60MHz	1.5:1
10	40MHz	40MHz	80MHz	2:1
11	40MHz	40MHz	120MHz	3:1

Vss Power for chip.

Vcc Ground for chip.

VssP Quite Vss for PLL.

VccP Quite Vcc for PLL.

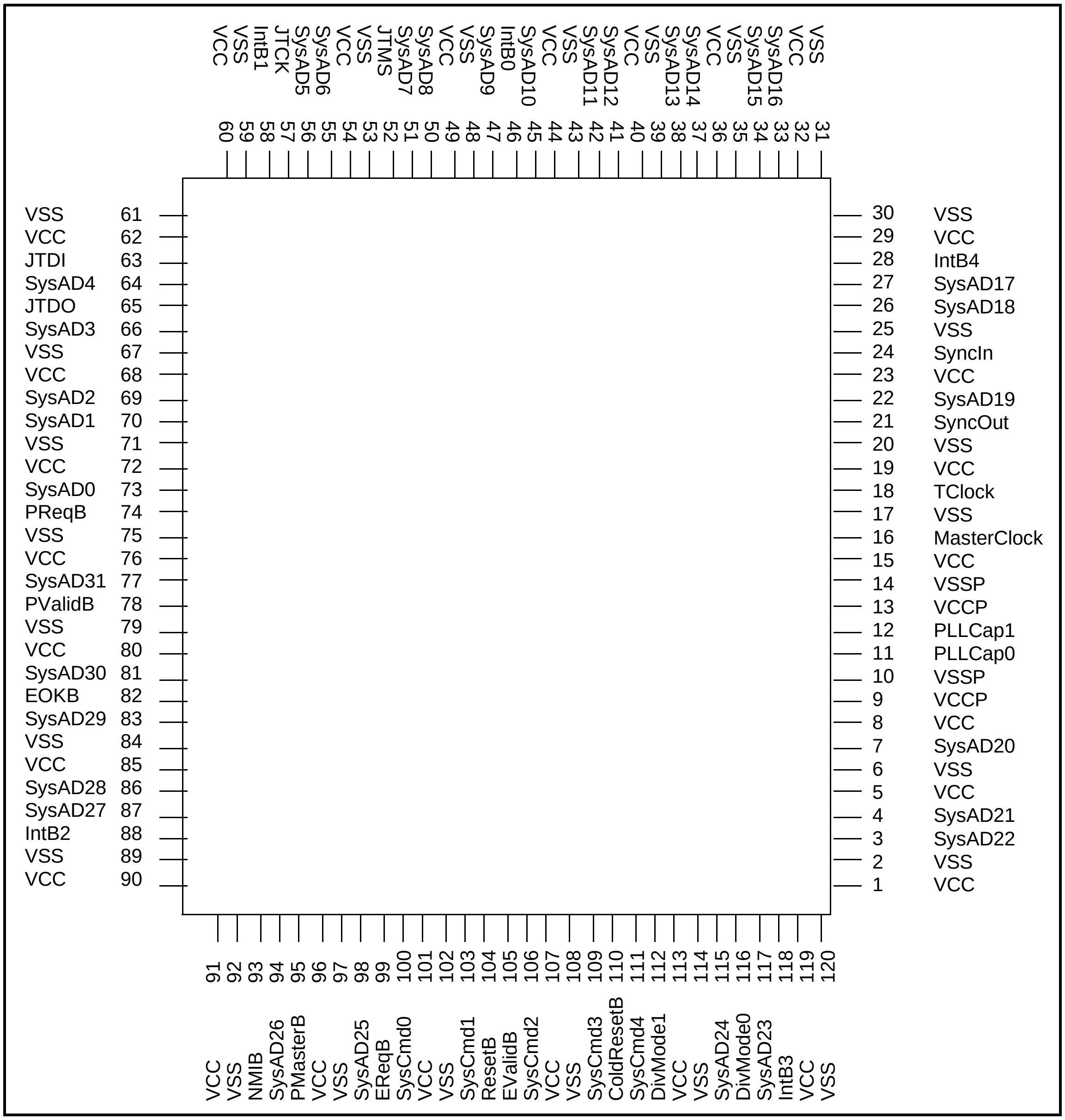
PLLCap0, PLLCap1 Connection for external capacitance component used for PLL low pass filter.

12.4 Pin Specifications.

12.4.1 120-pin PQFP Pin-out

1	VCC	31	VSS	61	VSS	91	VCC
2	VSS	32	VCC	62	VCC	92	VSS
3	SysAD22	33	SysAD16	63	JTDI	93	NMIB
4	SysAD21	34	SysAD15	64	SysAD4	94	SysAD26
5	VCC	35	VSS	65	JTDO	95	PMasterB
6	VSS	36	VCC	66	SysAD3	96	VCC
7	SysAD20	37	SysAD14	67	VSS	97	VSS
8	VCC	38	SysAD13	68	VCC	98	SysAD25
9	VCCP	39	VSS	69	SysAD2	99	EReqB
10	VSSP	40	VCC	70	SysAD1	100	SysCmd0
11	PLLCAP0	41	SysAD12	71	VSS	101	VCC
12	PLLCAP1	42	SysAD11	72	VCC	102	VSS
13	VCCP	43	VSS	73	SysAD0	103	SysCmd1
14	VSSP	44	VCC	74	PReqB	104	ResetB
15	VCC	45	SysAD10	75	VSS	105	EValidB
16	MasterClock	46	IntB0	76	VCC	106	SysCmd2
17	VSS	47	SysAD9	77	SysAD31	107	VCC
18	TClock	48	VSS	78	PValidB	108	VSS
19	VCC	49	VCC	79	VSS	109	SysCmd3
20	VSS	50	SysAD8	80	VCC	110	ColdResetB
21	SyncOut	51	SysAD7	81	SysAD30	111	SysCmd4
22	SysAD19	52	JTMS	82	EOKB	112	DivMode1
23	VCC	53	VSS	83	SysAD29	113	VCC
24	SyncIn	54	VCC	84	VSS	114	VSS
25	VSS	55	SysAD6	85	VCC	115	SysAD24
26	SysAD18	56	SysAD5	86	SysAD28	116	DivMode0
27	SysAD17	57	JTCK	87	SysAD27	117	SysAD23
28	IntB4	58	IntB1	88	IntB2	118	IntB3
29	VCC	59	VSS	89	VSS	119	VCC
30	VSS	60	VCC	90	VCC	120	VSS

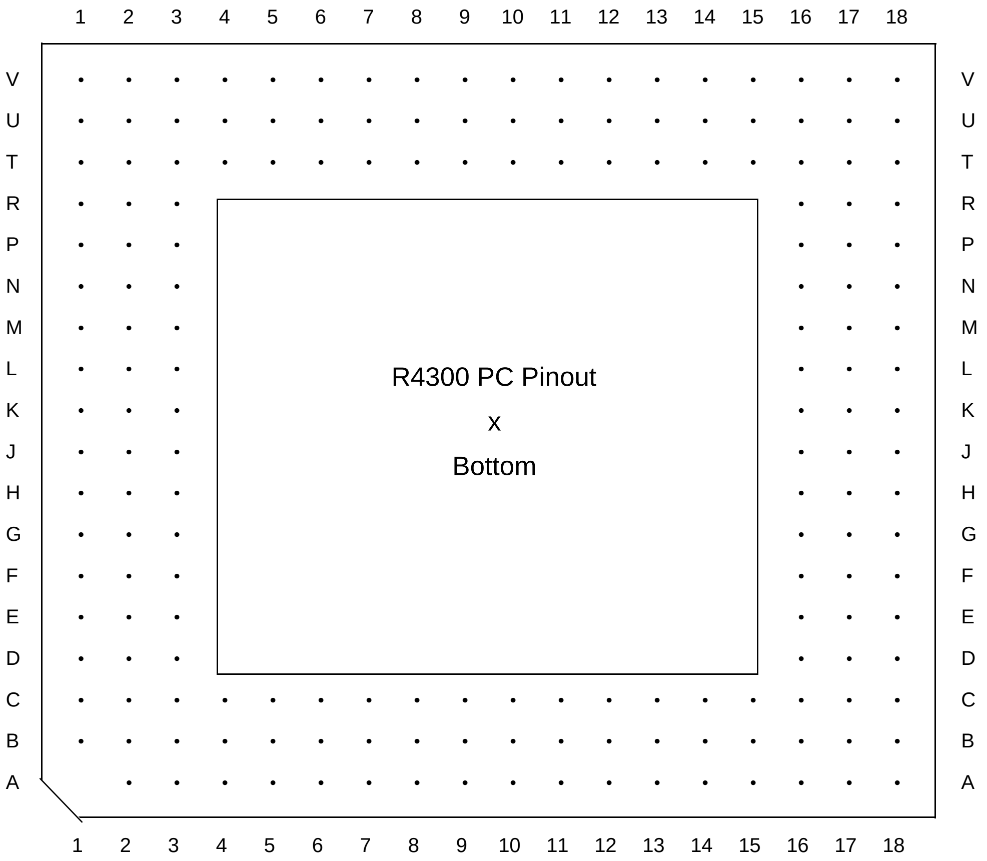
12.4.2 120 Pin PQFP Physical Pin Location



12.4.3 179-pin PGA Pin-out

A1	No pin	C10	SysAD9	K1	VCC	T10	SysCmd2
A2	VCC	C11	NC	K2	SysAD31	T11	ColdResetB
A3	VSS	C12	SysAD11	K3	NC	T12	DivMode1
A4	VCC	C13	SysAD13	K16	VSSP	T13	NC
A5	JTCK	C14	SysAD16	K17	VCCP	T14	DivMode0
A6	VSS	C15	NC	K18	VSS	T15	IntB3
A7	VCC	C16	NC	L1	VSS	T16	NC
A8	VSS	C17	NC	L2	NC	T17	NC
A9	VCC	C18	VSS	L3	PValidB	T18	VCC
A10	VSS	D1	VSS	L16	SysAD20	U1	VCC
A11	VCC	D2	JTDI	L17	TestModeB	U2	NC
A12	VSS	D3	NC	L18	VCC	U3	NC
A13	VCC	D16	NC	M1	VCC	U4	NC
A14	VSS	D17	NC	M2	SysAD30	U5	PMasterB
A15	NC	D18	VCC	M3	EOKB	U6	NC
A16	VCC	E1	NC	M16	SysAD21	U7	EReqB
A17	VSS	E2	JTDO	M17	ByPassPLLB	U8	NC
A18	VSS	E3	SysAD4	M18	VSS	U9	EValidB
B1	VSS	E16	NC	N1	VSS	U10	NC
B2	NC	E17	NC	N2	NC	U11	SysCmd3
B3	NC	E18	SysAD17	N3	NC	U12	NC
B4	NC	F1	VCC	N16	SysAD22	U13	SysCmd4
B5	NC	F2	NC	N17	NC	U14	SysAD24
B6	SysAD6	F3	SysAD3	N18	VCC	U15	
B7	NC	F16	IntB4	P1	SysAD29	U16	NC
B8	JTMS	F17	SysAD18	P2	SysAD28	U17	NC
B9	SysAD8	F18	VSS	P3	SysAD27	U18	VSS
B10	IntB0	G1	VSS	P16	NC	V1	VSS
B11	SysAD10	G2	SysAD1	P17	NC	V2	VSS
B12	SysAD12	G3	SysAD2	P18	VSS	V3	VCC
B13	SysAD14	G16	SyncIn	R1	VCC	V4	VSS
B14	SysAD15	G17	NC	R2	IntB2	V5	SysAD25
B15	NC	G18	NC	R3	NC	V6	VCC
B16	ResetB	H1	VCC	R16	NC	V7	VSS
B17	NC	H2	SysAD0	R17	NC	V8	VCC
B18	VCC	H3	NC	R18	VSS	V9	VSS
C1	VCC	H16	SysAD19	T1	VSS	V10	VCC
C2	NC	H17	SyncOut	T2	NC	V11	VSS
C3	NC	H18	VSS	T3	NC	V12	VCC
C4	NC	J1	VSS	T4	NC	V13	VSS
C5	NC	J2	NC	T5	NMIB	V14	NC
C6	IntB1	J3	PReqB	T6	SysAD26	V15	SysAD23
C7	SysAD5	J16	TClock	T7	SysCmd0	V16	VSS
C8	NC	J17	MasterClock	T8	SysCmd1	V17	VCC
C9	SysAD7	J18	VCC	T9	NC	V18	VSS

12.4.4 179-Pin PGA Physical Pin Location



Appendix A. Differences from the R4000

The R4300 processor implements the MIPS III instruction set, as does the R4000. As with any two hardware implementations of the same architecture, there are some differences between R4300 and the R4000. While R4300 has tried to stay as close to the R4000 implementation as possible both from a system design and software viewpoint, there are some visible differences. This section is an attempt to highlight the differences between R4300 and the R4000. This is not intended to be a complete list and cannot replace a thorough reading of the document.

The primary differences between R4300 and the R4000 are in the system interface bus definition and cache handling. This is primarily because the R4300 processor provides no support for second level caches and has the requirement of fitting into 120 pin PQFP package. R4300 also provides no support for multiprocessing, which has an impact on system design.

A.1 Software visible differences

The functional differences between the R4000 and the R4300 processor that are visible to software are contained in the Coprocessor 0 implementation.

A.1.1 Cache Ops

Because the R4300 processor does not support secondary cache, all of the references to secondary cache in the R4000 implementation do not apply to R4300. Designating either SD or SI (secondary data or instruction cache) for any cache op on R4300 is undefined. All writeback operations will send data from primary cache to main memory. Cache op number seven (7) is undefined for R4300.

The data cache Dirty bit, W bit in R4000, will be cleared for cache op Hit Write Back Data.

R4300 implements one software visible cache line state bit, where R4000, due to multiprocessing, implements two state bits. R4300 writes TagLo[7] when executing Index_Store_Tag_D instruction into cache line state bit, while R4000 writes both TagLo[7] and TagLo[6]. See section A.3.1 for more detail.

A.1.2 Cache Parity

R4300 does not provide parity protection for the caches.

Coprocessor0 CacheErr register (register 27) is not implemented, any accesses to this register are undefined.

Coprocessor0 PErr register (register 26) is used for diagnostic purposes only.

A.1.3 Status Register

R4300 has a slightly different status register from the R4000 The bits that perform identical functionalities are left in the same location on R4300 as they are in the R4000.

One bit of the Status Register that was previously always zero (i.e. unused) is now used. This is the Instruction Trace Support ITS bit.

Status register’s CH bit, the Cp0 Condition bit used for the Branch on Co-Processor 0 BC0T/BC0F instructions, is writable only by software in R4300. But, in R4000, this bit can be set or cleared by the hardware dependent on secondary cache operations, which R4300 does not support.

Status register’s CE and DE bits, which are associated with parity handling, are unused and hardwired to 0.

A.1.4 Configuration Register

The R4300 Configuration register implements a subset of the fields on the R4000, since R4300 does not have as many options. The exact meaning of bits within the configuration register may be different.

A.1.5 Unimplemented Operation Exception and other cause bits

R4000 defines the Unimplemented Operation Exception (E) in FSR to supersede all other cause bits. The Unimplemented Exception handler is expected to ignore and clear all cause bits. Thus, during E exception, all other cause bits are undefined or don't care.

R4300 has a stricter definition than the R4000. It sets no other FP exception cause bits when there is an Unimplemented Operation Exception.

A.1.6 Integer Divide-by-Zero

When an integer division by zero occurs, MIPS ISA specifies the result to be undefined. Under this illegal operation, R4300 returns different value in Register LO than R4000:

Here is what R4000 returns in Hi and Lo for all cases of dividend:

Dividend	Lo	Hi
>= 0	0xFFFFFFFF	dividend
< 0	0x00000001	dividend

and R4300 returns:

Dividend	Lo	Hi
>= 0	0x7FFFFFFF	dividend
< 0	0x80000001	dividend

A.1.7 Cache Parity Error exception

R4300 does not support parity and will never take parity error exception.

A.2 System Design differences

A.2.1 Processor Initialization

The R4000 had a fairly complicated scan based processor initialization scheme. R4300 has far fewer modes; those that are not available via software are hardwired as pins on the package. R4300 also requires only 16 cycles of Reset* assertion during the reset sequence.

A.2.2 System Interface

R4300 uses a similar protocol to the SysAD bus protocol, but not the same. R4300 system interface bus is 32-bits and does not support parity. For full details of the protocol, refer to “System Interface” chapter.

R4300 has fixed sizes for instruction cache lines (8 words) and data cache lines (4 words). Instruction blocks will be written to the memory system as a block of eight words. (The writing of instruction cache lines to memory can be done via the Hit Write Back Cache Op.)

R4300 will support two data rates for write transactions: D and Dxx. The rate to be used by R4300 is programmed via the EP field in Configuration register.

On the R4000, all address cycles are guaranteed to be at least three cycles apart. In fast mode on R4300 (DataRate=D) addresses for back to back writes or reads may only be one cycle apart (ADAD...).

R4300 guarantees that it can handle data coming onto the processor as fast as it can be delivered by the SysAD bus. There will never be a stream of data presented to the R4300 processor exceeding eight words, and R4300 can handle a data pattern of DDDDDDDD.

In order to conserve power, the R4300 processor provides only a single TClock.

A.2.3 RP Bit Effect on System Interface

On the R4000, *SClock* and *TClock* were unaffected by the RP bit. On R4300, setting the RP bit will reduce the clock frequency of *SClock* and *TClock* by a factor of four. If there are any timing dependent features in an external agent that would be affected by this (e.g., DRAM refresh counters), the external agent must provide software a mechanism to accommodate the frequency change.

A.3 Other Differences

A.3.1 I and D Cache

R4300 implements a 16k, direct mapped, virtually indexed instruction cache with a cache line of eight words (thirty-two bytes). The data cache is 8k, direct mapped, virtually indexed with a cache line of four words (sixteen bytes). This implies that any software routines for initializing, invalidating, or flushing cache must take this difference into consideration.

R4300 implements one software visible cache line state bit. Due to multiprocessing support, R4000 implements two cache line state bits. For compatibility, if a cache line state is checked via Index_load_Tag_D cache-op, R4300 will repeat the single state bit twice and write it to TagLo register PState field. The following PStates values are possible for R4300, 00, 11. For R4000, 00,01,10,11 are possible. The R4000PC, which does not support multiprocessing, for an invalid line the state is 00, and for a line which was filled via a cache miss is set to 11, same as R4300. But if the line state is modified via Index_Store_Tag_D cache-op then the only supported values for R4300 are 00, 11. R4300 write TagLo[7] when executing Index_Store_Tag_D instruction into line state bit, R4000 writes both TagLo[7] and TagLo[6].

A.3.2 TLB

A.3.2.1 TLB entries

R4300 implements a thirty-two way, fully associative TLB in which each entry maps two Page Frame Numbers, one even and one odd. While the structure of this is identical to the R4000, the number of entries is reduced from forty-eight entries on the R4000.

A.3.2.2 Interactions between ITM & TLB Ops

When a software TLB instruction is accessing the JTLB during the same cycle as an Instruction TLB Miss (ITM) stall, the R4000 behaves in an undefined manner, and can actually generate a bogus TLB Invalid Exception, especially when the TLB software read/write operation (tlbwi, tlbwr, tlbr) is accessing a completely different entry than that accessed on behalf of the Instruction TLB miss. R4300 has corrected this bogus TLB behavior.

A.3.3 Floating Point Coprocessor

A.3.3.1 Floating-point Datapath

R4300 integrates the operation of Coprocessor 1, the floating point coprocessor, into the main pipeline and datapath of the integer processor. This means that the integer pipe will stall for any multicycle floating point instructions. While this is invisible to software in terms of functionality, code that may have been optimized for the R4000 will likely see no improvement on R4300.

A.3.3.2 Variable latencies

Any multicycle instruction which experiences a source exception (i.e. an exception with its source operands) will not cause a stall. Instead, it will issue a default result on that cycle or report a trapped exception on the next cycle, depending on the trap enable flags. In addition, certain trivial calculations (such as 0 * 0) will be performed with less latency than non-trivial cases of the same operation. The R4000, on the other hand, always has the same latency for each particular instruction whether an exception occurs or not.

A.3.3.3 Cvt.[s,d].l instruction

To convert a 64-bit integer to a single- or double-precision floating point number, R4000 initiates a FP Unimplemented Operation exception when bits 63...52 of an integer are not all ones or all zeros.

Since it has a unified 64-bit datapath for both integer and floating-point operations, R4300 can process more bits out of a 64-bit integer in the long integer to floating-point convert instruction, without added cost to the design. R4300 raises a FP unimplemented operate exception only when bits 63...55 of a 64-bit integer are not all ones or all zeros.

A.3.4 RP Bit Effect on *PClock*

On the R4000, setting the RP bit will reduce the PClock frequency by a factor of two through sixteen. On R4300, this bit will reduce PClock frequency by a factor of four.

A.3.5 Pipeline

R4300 uses a simpler pipeline than the R4000. This pipe is a five stage pipe that strongly resembles the pipeline used on the R3000. R4300 is not superpipelined. This is not visible in any functional way, but may impact how different code sequences may be optimized. This also means that there will be fewer stalls due to the basic pipeline. This increases the CPI component due to cache misses, meaning that the system design of the memory system is even more crucial to the performance of the R4300 processor.

A.3.6 Interrupts

R4300 dedicates bit 15 of the Cause Register, which is bit 7 of the Interrupt Pending field, to the timer interrupt caused when the Compare and Count registers match. Writing to Interrupt register bit 5 via the system interface will have no effect. The R4000 allowed the user to select whether bit 15 was dedicated to the timer interrupt or to Interrupt register bit 5. Interrupt register bit 5 has been eliminated on R4300.

A.3.7 Kernel Physical Address Segment Organization

R4300 implements only two cache coherency algorithms, uncached and cacheable non-coherent. This, and the fact that R4300 implements only 32 physical address bits as compared to the R4000's 36-bit physical addresses, is reflected in the virtual address map in the organization of the Kernel Physical Address Space segment. R4300 breaks this segment further into two valid address spaces, one per cache coherency algorithm, while the rest of the address space is unavailable. The R4000 processor, in contrast, has more valid addresses in this segment. Refer to “Figure 16: R4300 64-bit Address Space” for the organization of this virtual address segment for the R4300 processor.

A.3.8 JTAG

When in Shift-IR and Shift-DR modes, IEEE Std1149.1-1990 on page 5-11 in Table 5-2 states that JTDO should be active. R4300 implements the JTAG controller per IEEE Std1149.1-1990. But, R4000 implements an older version of the standard and does not drive JTDO under these two modes.

Appendix B. Differences from the R4200

The primary differences between R4300 and the R4200 are in the system interface bus architecture and in the absence of parity. The requirement for R4300 to fit into a small PQFP package necessitated a new bus definition, which is very similar to but not the same as the R4200 system interface bus.

Table 1: Summary of differences between R4200 and R4300

Feature	R4200	R4300
System Bus	64 bits	32 bits
Multiplier	No	Yes
Pipeline Frequency	80Mhz	100Mzh
System Frequency	40Mhz	62.5Mhz
Flush Buffers	1 address	4 addresses
Cache Parity	Yes	No
System Bus Parity	Yes	No
Mode Pins	Yes	via Cp0 Config Reg
1:1 PClock / SClock Ratio	No	Yes
1.5:1 PClock / SClock Ratio	No	Yes
2:1 PClock / SClock Ratio	Yes	Yes
3:1 PClock / SClock Ratio	No	Yes
4:1 PClock / SClock Ratio	Yes	No
PQFP Package	208 pin	120 pin
PGA Package	179 pin	179 pin (Debug Only)
Physical Address	33 bits	32 bits
RClock & MasterOut	Yes	No
Fast DataRate	DDx	D
CacheTestMode via JTAG	No	Yes

B.1 Software visible differences

B.1 .1 Cache Parity

R4300 does not provide parity protection for the caches.

Coprocessor0 CacheErr register (register 27) is not implemented, any accesses to this register are undefined.

Coprocessor0 PErr register (register 26) is used for diagnostic purposes only.

B.1 .2 Status Register

Status register’s CE and DE bits, which are associated with parity handling, are unused and hardwired to 0 and 1 respectively.

B.1 .3 Configuration Register

R4200 configuration register fields BE and EP are set by hardware to the values specified by BigEndian and DataRate pins during reset and are read only by software; R4300 sets these fields to default values during ColdReset only and allows software to modify them. Bits[19..18] changed from 00 in R4200 to 01.

B.1 .4 Cache Parity Error Exception

R4300 does not support parity and will never take parity error exception.

B.2 System Interface

R4300 uses a similar protocol to the SysAD bus protocol, but not the same. R4300 system interface bus is 32-bits and does not support parity. For full details of the protocol, refer to chapter “8.0 System Interface”.

Instruction blocks will be written to the memory system as a block of eight words transaction instead of the sequence of four doublewords separated by one dead cycle.

R4300 fast data rate is D instead of DDx. The data rate is software programmable via the Configuration register, whereas it is a pin on R4200.

B.2 .1 Clocks

R4300 does not output MasterOut and RClock.

The clock derivation scheme is different in R4300. Instead of always multiplying MasterClock by 2 to generate PClock, the multiplication factor is now obtained from DivMode(1:0) pins of the chip. This factor can be 1x, 2x, 3x or 1.5x to give the ratios of 1:1, 2:1, 3:1 and 3:2 between PClock and MasterClock respectively. SClock and TClock are the same frequency as MasterClock, instead of being derived from PClock. As with the R4200, RP mode on the R4300 divides PClock, SClock, and TClock by four of their normal frequency.

B.2 .2 Power/Gnd pins

There will be two sets of Vcc/Vss on R4300. One for I/O and core, the other for PLL. R4200 has three sets, one for I/O, one for core and one for PLL.

B.2 .3 Packaging

R4300 package is 120 pin PQFP; while R4200 uses 208 pin PQFP.

B.3 Other Differences

B.3 .1 Physical Address

R4300 physical address and physical address space is 32 bits, while R4200 physical address and space is 33 bits. This results in 20 bits in tag portion of the caches and page frame number fields of TLB low and hi entries.

B.3 .2 Flush Buffer

R4300 has a four double word (64 bits) deep flush buffer to improve performance of back to back uncached write operations.

B.3 .3 Resets

Reset does not need to be asserted during or after assertion of ColdReset. ColdReset does not need to be asserted/deasserted synchronously with MasterClock.

B.3 .4 TLB Shutdown

When multiple entries in the TLB match during a TLB access, the TLB will no longer shutdown and the processor will continue operation. The TS bit in the Status register will still be set.

Appendix C. Glossary

BigEndian
Data organization such that the most significant byte of a word (the leftmost byte) is labeled byte 0.

CP0
Coprocessor Zero. This unit contains its own register set and instructions. It is used for memory management and exception handing.

CP1
Coprocessor One. This is the floating point coprocessor.

Doubleword
A sixty-four bit quantity.

Exception
Any condition that causes the normal flow of instructions to be interrupted and special software assistance to be invoked. Examples of exceptions are resets, external interrupts, TLB misses, error conditions, and arithmetic exceptions such as underflow and overflow.

Flush Buffer
This is a buffer that contains data that is to be written to an external agent. It is filled by either a store to an uncached location, or by a dirty cache line that must be written to memory because a cache miss requires a new line to occupy that entry in cache. The pipeline may continue while the flush buffer is full, but if any more data needs to be written to an external agent, there is an interlock and the pipeline must stall until the flush buffer empties.

Instruction TLB
In general, this is synonymous with microTLB.

Interlock
An interlock is a dependency of one instruction on a processor resource that is not ready due its use by another instruction. Interlocks are resolved on R4300 by stalling the whole pipeline. Examples of interlocks are integer multiplies, which use the ALU for many cycles, cache misses, and load-use interlocks, where an instruction wishes to use a register for which a load has not completed.

JTLB
Joint TLB. This is a 32 double entry (even and odd entries) TLB that can hold both data and instructions. The JTLB is the only TLB used for data, and is a secondary TLB used for instructions, which have their own primary TLB.

LittleEndian
Data organization such that the least significant byte of a word (the rightmost byte) is labeled byte 0.

LLBit
Load Linked bit. This bit is set any time the processor executes a Load Linked instruction. It will be checked when a Store Conditional is attempted, and the store will not occur if this bit is clear. All bits of the source register for the store conditional will be cleared except the LSB, which will have the LLBit written into it. The LLBit will be cleared upon any cache miss or upon execution of ERET. Though this mechanism is intended for multiprocessor use, R4300 will implement it anyway.

MicroTLB
The MicroTLB is a two entry, fully associative, instruction only TLB. If an instruction access should miss this TLB, the R4300 processor will check the JTLB for the translation.

NMI
Non-Maskable Interrupt. This interrupt cannot be disabled by software.

Orion
A mythological figure from ancient history.

Page Size
Memory is divided into blocks known as pages. Virtual addressing is accomplished by using a virtual page address to point to a block of real (physical) memory. The page size is the amount of memory referenced by a virtual (or real) page.

PSIZE
Physical address size in number of bits. R4300 supports 32 bits of physical address.

Run
The normal state of the pipeline, in which each pipestage completes its work and passes the instruction on to the next stage. While the pipeline is running, a new instruction will be fetched and an old instruction will be retired on every clock.

SPECint
The geometric mean of the integer benchmarks of the SPEC 89 benchmark suite.

SPECfp The geometric mean of the floating point benchmarks in the SPEC 89 benchmark suite.

Stall A condition in which the pipeline has stopped moving, waiting for a resource conflict to be resolved.

TLB Translation Lookaside Buffer. This contains entries of virtual addresses and their corresponding physical addresses. The logical TLB is comprised on R4300 of a joint TLB (JTLB), which hold both data and instructions, and a micro TLB (uTLB) which holds only instructions. In general, when the term TLB is used in this document, it is referring to the logical TLB.

uTLB Micro Instruction TLB.

VSIZE Virtual address size in number of bits. R4300 supports 40 bits of virtual address.

wombat A non-arboreal cousin of the koala.

Word A thirty-two bit quantity.

Write-back The cache write algorithm implemented on R4300. Data stored to a cached memory address will be written back into the cache, but not directly out to memory. This is as opposed to a write through algorithm.