uPD30401 VR4000SC 64-Bit Microprocessor Preliminary Hardware User's Manual

Publication ID: **IEU-1331**
Publication Date: **December 1, 1992**

Company: **N E C ELECTRONICS INC**

A

B

CONTENTS

CONTENTS OF FIGURES

Fig. Number	Title, Page	

CONTENTS OF TABLES

CHAPTER 1 GENERAL

1.1 Introduction

The VR4000SC processor supports interfaces to secondary cache, system interface, and boot time mode control. This document describes the connectivity and operation of each of these interfaces.

1.2 Operation Fundamentals

A **word** is the basic data element of the VR4000SC processor. A **word** is a thirty-two bit data element. A sixty-four bit data element is referred to as a **double word**, a sixteen bit data element is referred to as a **half word** and an eight bit data element is referred to as a **byte**.

1.3 Clocking Fundamentals

The VR4000SC processor bases all clocking methodology on the single clock input **MasterClock** at the desired operational frequency for the processor. MasterClock is multiplied by two internally, using phase locked loop techniques, to generate the processor internal clock, **PClock**. PClock is used by the processor's execution units, and to sequence the secondary cache interface. All secondary cache interface transaction protocol and parameters are specified in terms of PCycles, where a PCycle is the period of PClock or half the period of MasterClock.

PClock is divided by a programmable divisor to generate the processor internal clock, **SClock**, and the system interface clocks, **TClock** and **RClock**. SClock is used by the processor to clock all internal registers that sample system interface inputs and drive system interface outputs. TClock and RClock are driven off the processor for use by an external agent. The PClock to SClock divisor is programmed via the boot time mode control interface as described in the Boot Time Mode Control Interface section. All system interface transaction protocol and parameters are specified in terms of SCycles, where a SCycle is the period of SClock, unless otherwise specified.

See **CHAPTER 10 CLOCKING** for further details on the clocking behavior of the VR4000SC processor.

1.4 Ordering Information

Part Number	Package	Quality Grade
μPD30401RJ-50	447-pin ceramic PGA (without heatsink)	Standard
μPD30401RM-50	447-pin ceramic PGA (with heatsink)	Standard
μPD30401RS-50	447-pin ceramic LGA (without heatsink)	Standard
μPD30401RT-50	447-pin ceramic LGA (with heatsink)	Standard

Remark LGA: Land Grid Array

Please refer to "Quality grade on NEC Semiconductor Devices" (Document number IEI-1209) published by NEC Corporation to know the specification of quality grade on the devices and its recommended applications.

1.5 Pin Configuration

Fig. 1-1 Pin Configuration (Bottom View)

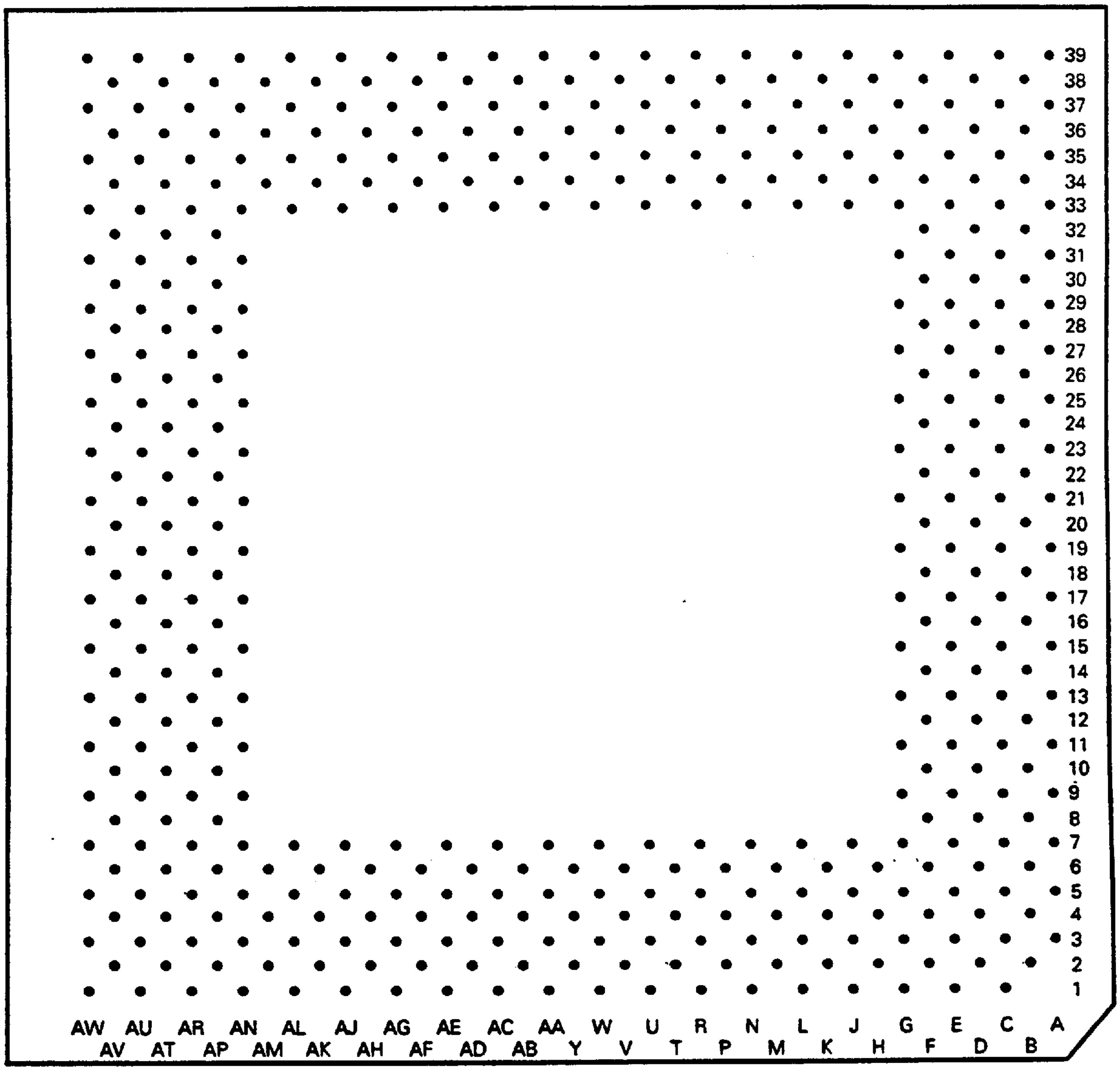

39
38
37
36
35
34
33
32
31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
AW AU AR AN AL AJ AG AE AC AA W U R N L J G E C A
AV AT AP AM AK AH AF AD AB Y V T P M K H F D B

Table 1-1 Pin Configuration (1/3)

No.	Name	No.	Name	No.	Name	No.	Name
AW37	$\overline{\text{ColdReset}}$	AA3	SCAddr11	G25	SCData10	AF4	SCData48
AV2	$\overline{\text{ExtRqst}}$	W3	SCAddr12	E29	SCData11	AJ3	SCData49
C39	$\overline{\text{Fault}}$	Y6	SCaddr13	G31	SCData12	AJ7	SCData50
AV24	$\overline{\text{GrpRun}}$	W5	SCAddr14	C35	SCData13	AP8	SCData51
AV20	$\overline{\text{GrpStall}}$	W7	SCAddr15	K36	SCData14	AT10	SCData52
AV32	IOIn	W1	SCAddr16	N35	SCData15	AR13	SCData53
AV28	IOOut	U3	SCAddr17	AE3	SCData16	AR15	SCData54
AL1	$\overline{\text{Int0}}$	AN7	SCAddr0W	AG5	SCData17	AT18	SCData55
AA35	IC(H) **Note 1**	AN5	SCAddr0X	AK4	SCData18	AU23	SCData56
AA39	IC(H) **Note 1**	AM6	SCAddr0Y	AN9	SCData19	AT26	SCData57
U39	JTCK	AL7	SCAddr0Z	AU9	SCData20	AR27	SCData58
N39	JTDI	M6	$\overline{\text{SCDCS}}$	AN13	SCData21	AN29	SCData59
J39	JTDO	G19	SCDChk0	AT14	SCData22	AP32	SCData60
G37	JTMS	T34	SCDChk1	AR17	SCData23	AN35	SCData61
AA37	MasterClock	AP20	SCDChk2	AT22	SCData24	AJ35	SCData62
AJ39	MasterOut	AD34	SCDChk3	AU25	SCData25	AE33	SCData63
B8	ModeClock	C19	SCDChk4	AN27	SCData26	V4	SCData64
AV8	ModeIn	R37	SCDChk5	AR29	SCData27	R5	SCData65
AV16	$\overline{\text{NMI}}$	AU19	SCDChk6	AN31	SCData28	N5	SCData66
AM34	RClock0	AE37	SCDChk7	AR35	SCData29	E5	SCData67
AL33	RClock1	C17	SCDChk8	AK36	SCData30	G9	SCData68
AW7	$\overline{\text{RdRdy}}$	N37	SCDChk9	AG35	SCData31	E11	SCData69
AV12	$\overline{\text{Release}}$	AU17	SCDChk10	T6	SCData32	G13	SCData70
AU39	$\overline{\text{Reset}}$	AG37	SCDChk11	L3	SCData33	D14	SCData71
Y2	Open **Note 2**	E19	SCDChk12	L7	SCData34	C21	SCData72
U5	SCAPar0	R35	SCDChk13	E7	SCData35	D22	SCData73
U1	SCAPar1	AR19	SCDChk14	G11	SCData36	E25	SCData74
P4	SCAPar2	AE35	SCDChk15	E13	SCData37	G27	SCData75
AL5	SCAddr1	R3	SCData0	E15	SCData38	C31	SCData76
AG1	SCAddr2	R7	SCData1	G17	SCData39	F32	SCData77
AE7	SCAddr3	L5	SCData2	C23	SCData40	J35	SCData78
AC1	SCAddr4	F8	SCData3	F24	SCData41	M34	SCData79
AC5	SCAddr5	C9	SCData4	E27	SCData42	AC7	SCData80
AC3	SCAddr6	F12	SCData5	D30	SCData43	AE5	SCData81
AA1	SCAddr7	G15	SCData6	C33	SCData44	AG7	SCData82
AB4	SCAddr8	E17	SCData7	E35	SCData45	AR5	SCData83
AA5	SCAddr9	G21	SCData8	L35	SCData46	AR9	SCData84
AA7	SCAddr10	C25	SCData9	R33	SCData47	AR11	SCData85

Note 1. Be sure to connect this pin to V_{DD}.

2. Leave unconnected.

Remark See **CHAPTER 14 PIN SUMMARY** for pin functions.

Table 1-1 Pin Configuration (2/3)

No.	Name	No.	Name	No.	Name	No.	Name
AN15	SCData86	AL37	SCData126	U33	Status0	AM38	SysAD30
AP16	SCData87	AG33	SCData127	U35	Status1	AH38	SysAD31
AU21	SCData88	N1	$\overline{\mathrm{SCOE}}$	V36	Status2	R1	SysAD32
AN23	SCData89	J1	$\overline{\mathrm{SCTCS}}$	W35	Status3	L1	SysAD33
AR25	SCData90	AN21	SCTChk0	W37	Status4	H2	SysAD34
AP28	SCData91	AN19	SCTChk1	AC37	Status5	E1	SysAD35
AU31	SCData92	AU15	SCTChk2	AC35	Status6	C3	SysAD36
AR33	SCData93	AP12	SCTChk3	AC33	Status7	A5	SysAD37
AL35	SCData94	AU7	SCTChk4	W39	SyncIn	A11	SysAD38
AH34	SCData95	AR7	SCTChk5	AN39	SyncOut	A15	SysAD39
U7	SCData96	AH6	SCTChk6	T2	SysAD0	A23	SysAD40
N3	SCData97	K4	SCTag0	M2	SysAD1	A27	SysAD41
N7	SCData98	G7	SCTag1	J3	SysAD2	A31	SysAD42
C5	SCData99	C7	SCTag2	G3	SysAD3	A35	SysAD43
E9	SCData100	D10	SCTag3	C1	SysAD4	C37	SysAD44
C11	SCData101	C15	SCTag4	A3	SysAD5	E39	SysAD45
C13	SCData102	D18	SCTag5	A9	SysAD6	H38	SysAD46
F16	SCData103	F20	SCTag6	A13	SysAD7	M38	SysAD47
E21	SCData104	E23	SCTag7	A21	SysAD8	AE1	SysAD48
G23	SCData105	D26	SCTag8	A25	SysAD9	AJ1	SysAD49
C27	SCData106	C29	SCTag9	A29	SysAD10	AM2	SysAD50
F28	SCData107	G29	SCTag10	A33	SysAD11	AR1	SysAD51
E31	SCData108	E33	SCTag11	B38	SysAD12	AU3	SysAD52
G33	SCData109	G35	SCTag12	E37	SysAD13	AW5	SysAD53
J37	SCData110	L33	SCTag13	G39	SysAD14	AW11	SysAD54
N33	SCData111	L37	SCTag14	L39	SysAD15	AW15	SysAD55
AD6	SCData112	P36	SCTag15	AD2	SysAD16	AW23	SysAD56
AG3	SCData113	AF36	SCTag16	AH2	SysAD17	AW27	SysAD57
AJ5	SCData114	AJ37	SCTag17	AL3	SysAD18	AW31	SysAD58
AU5	SCData115	AJ33	SCTag18	AN3	SysAD19	AW35	SysAD59
AN11	SCData116	AN37	SCTag19	AU1	SysAD20	AU37	SysAD60
AU11	SCData117	AU35	SCTag20	AW3	SysAD21	AR39	SysAD61
AU13	SCData118	AR31	SCTag21	AW9	SysAD22	AL39	SysAD62
AN17	SCData119	AU29	SCTag22	AW13	SysAD23	AG39	SysAD63
AR21	SCData120	AN25	SCTag23	AW21	SysAD24	A17	SysADC0
AP24	SCData121	AR23	SCTag24	AW25	SysAD25	R39	SysADC1
AU27	SCData122	J5	$\overline{\mathrm{SCWrW}}$	AW29	SysAD26	AW17	SysADC2
AT30	SCData123	J7	$\overline{\mathrm{SCWrX}}$	AW33	SysAD27	AD38	SysADC3
AU33	SCData124	H6	$\overline{\mathrm{SCWrY}}$	AV38	SysAD28	A19	SysADC4
AN33	SCData125	G5	$\overline{\mathrm{SCWrZ}}$	AR37	SysAD29	T38	SysADC5

Remark See **CHAPTER 14 PIN SUMMARY** for pin functions.

Table 1-1 Pin Configuration (3/3)

No.	Name	No.	Name	No.	Name	No.	Name
AW19	SysADC6	H36	VDD	B22	Gnd	AP30	Gnd
AC39	SysADC7	K6	VDD	B30	Gnd	AP34	Gnd
G1	SysCmd0	K38	VDD	B36	Gnd	AP36	Gnd
E3	SysCmd1	P2	VDD	D2	Gnd	AT2	Gnd
B2	SysCmd2	P34	VDD	D6	Gnd	AT6	Gnd
B12	SysCmd3	T4	VDD	D12	Gnd	AT12	Gnd
B16	SysCmd4	T36	VDD	D20	Gnd	AT20	Gnd
B20	SysCmd5	V6	VDD	D28	Gnd	AT28	Gnd
B24	SysCmd6	V38	VDD	D34	Gnd	AT34	Gnd
B28	SysCmd7	Y38	VDD	D38	Gnd	AT38	Gnd
B32	SysCmd8	AB2	VDD	F4	Gnd	AV4	Gnd
A37	SysCmdP	AB34	VDD	F6	Gnd	AV10	Gnd
H34	TClock0	AD4	VDD	F10	Gnd	AV18	Gnd
J33	TClock1	AD36	VDD	F18	Gnd	AV26	Gnd
AE39	VDDOk	AF6	VDD	F26	Gnd	AV36	Gnd
AN1	$\overline{\text{ValidIn}}$	AF38	VDD	F34	Gnd		
AR3	$\overline{\text{ValidOut}}$	AK2	VDD	F36	Gnd		
A7	$\overline{\text{WrRdy}}$	AK34	VDD	K2	Gnd		
W33	VDDSense	AM4	VDD	K34	Gnd		
U37	GndSense	AM36	VDD	M4	Gnd		
AA33	VDDP	AP2	VDD	M36	Gnd		
Y34	GndP	AP10	VDD	P6	Gnd		
A39	VDD	AP18	VDD	P38	Gnd		
B6	VDD	AP26	VDD	V2	Gnd		
B10	VDD	AP38	VDD	V34	Gnd		
B18	VDD	AT4	VDD	Y4	Gnd		
B26	VDD	AT8	VDD	Y36	Gnd		
B34	VDD	AT16	VDD	AB6	Gnd		
D4	VDD	AT24	VDD	AB36	Gnd		
D8	VDD	AT32	VDD	AB38	Gnd		
D16	VDD	AT36	VDD	AF2	Gnd		
D24	VDD	AV6	VDD	AF34	Gnd		
D32	VDD	AV14	VDD	AH4	Gnd		
D36	VDD	AV22	VDD	AH36	Gnd		
F2	VDD	AV30	VDD	AK6	Gnd		
F14	VDD	AV34	VDD	AK38	Gnd		
F22	VDD	AW1	VDD	AP4	Gnd		
F30	VDD	AW39	VDD	AP6	Gnd		
F38	VDD	B4	Gnd	AP14	Gnd		
H4	VDD	B14	Gnd	AP22	Gnd		

Remark See **CHAPTER 14 PIN SUMMARY** for pin functions.

Fig. 1-2 Pad Configuration (Top View)

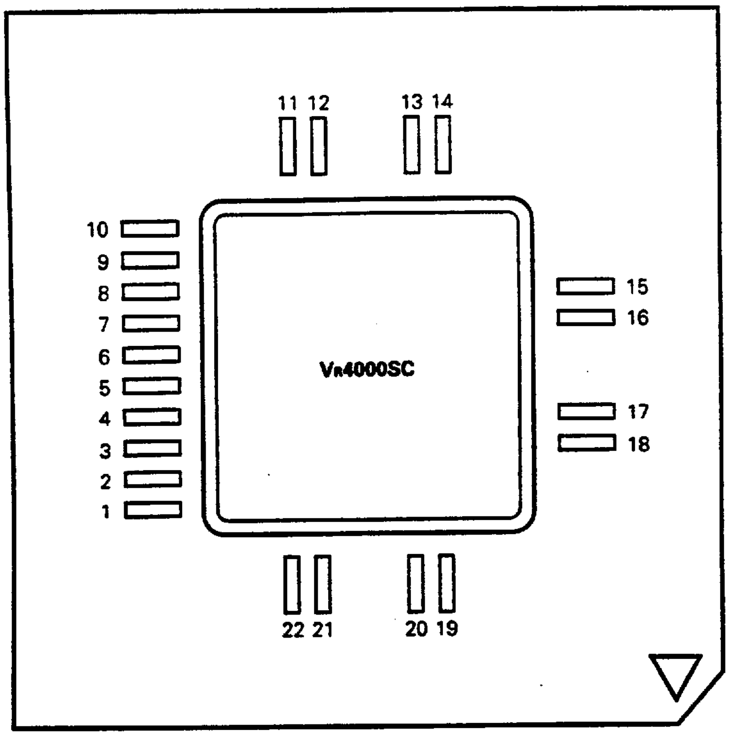


Table 1-2 Pad Configuration

No.	Name	No.	Name	No.	Name	No.	Name
1	V_{DD}	7	V_{DD}P	13	V_{DD}	19	V_{DD}
2	Gnd	8	PLLCap1	14	Gnd	20	Gnd
3	PLLCap0	9	V_{DD}	15	V_{DD}	21	V_{DD}
4	GndP	10	Gnd	16	Gnd	22	Gnd
5	GndP	11	V_{DD}	17	V_{DD}		
6	V_{DD}P	12	Gnd	18	Gnd		

CHAPTER 2 CACHE COHERENCY

The VR4000SC processor manages its primary and secondary caches using a write back methodology, that is, stores write data into the caches, but a modified cache line is not written back to memory until the cache line is replaced, or until the cache line is exported or flushed from the secondary cache. When the contents of a cache line is not consistent with memory, it is said to be **dirty**. Many systems, in particular multiprocessor systems, or systems that employ input/output (IO) devices that are capable of direct memory access (DMA), may require the system to behave as if the caches are always consistent with memory and each other. Schemes for maintaining consistency between multiple write back caches or between write back caches and memory are referred to as **cache coherency protocols**.

The processor, in its secondary cache mode provides a set of cache states and mechanisms for manipulating the contents and state of the cache that are sufficient to implement a variety of cache coherency protocols both snoopy and directory based. In particular, the processor supports both the **write invalidate** and **write update** protocols simultaneously.

The coherency protocol for lines in the cache is controllable via bits in the translation look-aside buffer (TLB) on a per TLB page basis. Specifically, the TLB contains three bits per entry that control the coherency attributes of a page. The three bits are encoded to provide five possible coherency attributes per page, **uncached, sharable, update, exclusive**, and **noncoherent**. A processor in the no-secondary cache mode supports only the uncached and noncoherent coherency attributes.

If a page has the coherency attribute uncached, the processor will issue a word or partial word read or write directly to main memory for any load or store to a location within that page. Lines within an uncached page are assumed never to be cache resident.

If the coherency attribute is sharable, the processor will issue a coherent block read for a load miss to a location within the page, and a coherent block read that requests exclusivity for a store miss to a location with the page. In most systems, coherent reads require snoops or directory checks to occur while noncoherent reads do not. A coherent read that requests exclusivity implies that the processor will function most efficiently if the requested cache line is returned to it in an exclusive state.

If the coherency attribute is exclusive, the processor will issue a coherent block read the requests exclusivity for a load or store miss to a location within the page.. Cache lines within the page will be managed with a write invalidate protocol. Load linked store conditional instruction sequences must insure that the link location is not in a page managed with the exclusive coherency attribute.

If the coherency attribute is noncoherent, the processor will issue a noncoherent block read for a load or store miss to a location within the page.

The encoding of the coherency attributes in the TLB is specified in **VR4000 USER'S MANUAL (PRELIMINARY) ARCHITECTURE**. The behavior of the processor on load misses, store misses, and store hits to shared cache lines for each of the coherency attributes is summarized in **Table 2-1 Coherency Attributes and Processor Behavior**.

Table 2-1 Coherency Attributes and Processor Behavior

Attribute	Load Miss	Store Miss	Store Hit Shared
uncached	Main memory read	Main memory write	NA
noncoherent	Noncoherent read	Noncoherent write	Invalidate **Note**
exclusive	Coherent read exclusive	Coherent read exclusive	Invalidate **Note**
sharable	Coherent read	Coherent read exclusive	Invalidate
update	Coherent read	Coherent read	Update

Note This should not occur under normal circumstances.

The following sections describe the primary and secondary cache states provided by the processor, the cache state transitions performed by the processor during execution, and the mechanisms provided for an external agent to manipulate the state and contents of the primary and secondary cache.

2.1 Cache States

The VR4000SC maintains four primary cache states and five secondary cache states. The five secondary cache states are:

- Invalid
- Shared
- Dirty Shared
- Clean Exclusive
- Dirty Exclusive

The four primary cache states are:

- Invalid
- Shared
- Clean Exclusive
- Dirty Exclusive

The primary cache state shared corresponds to the secondary cache states shared and dirty shared.

The cache state of a line in the processor's primary or secondary cache indicates the validity, shared, dirty and ownership attributes of the cache line. A cache line that does not contain valid information must be marked **invalid**, a cache line in any state other than invalid contains valid information. A cache line that is present in more than one cache in the system is said to be **shared** and must be in one of the shared states. A cache line that is present in exactly one cache in the system is said to be **exclusive** and may be in one of the exclusive states. A cache line that contains data that is consistent with memory is said to be **clean** and may be in one of the clean states. A cache line that contains data that is not consistent with memory is said to be **dirty** and must be in one of the dirty states, or in the shared state. The processor has a concept of ownership for cache lines. When the processor is the **owner** of a particular cache line it is responsible for writing the cache line back to memory when it is replaced in the course of satisfying a cache miss, or during the execution of a cache instruction. A cache line is owned by the processor if its secondary cache state is dirty exclusive or dirty shared.

The primary and secondary cache states have been chosen to maintain all of the state information that the processor may need during execution in the primary cache, while maintaining all of the state information that an external agent may need to manage a cache coherency protocol in the secondary cache.

The primary cache states indicate the following cache line attributes:

Invalid	The cache line does not contain valid information.
Shared	The cache line contains valid information and may be present in another processor's cache. The cache line may or may not be consistent with memory, and may or may not be owned.
Clean Exclusive	The cache line contains valid information and is not present in any other processor's cache. The cache line is consistent with memory and is not owned.
Dirty Exclusive	The cache line contains valid information and is not present in any other processor's cache. The cache line is inconsistent with memory and owned.

The secondary cache states indicate the following cache line attributes:

Invalid	The cache line does not contain valid information.
Shared	The cache line contains valid information and may be present in another processor's cache. The cache line may or may not be consistent with memory, but is not owned.
Dirty Shared	The cache line contains valid information and may be present in another processor's cache. The cache line is inconsistent with memory and owned.
Clean Exclusive	The cache line contains valid information and is not present in any other processor's cache. The cache line is consistent with memory and is not owned.
Dirty Exclusive	The cache line contains valid information and is not present in any other processor's cache. The cache line is inconsistent with memory and owned.

2.2 Cache State Changes During Processor Execution

The initial state of a cache line is specified by an external agent when it supplies the cache line. During the course of processor execution, the processor may change the state of a cache line. The following events will cause changes to the state of the cache:

A store to a clean exclusive cache line will cause the state to be changed to dirty exclusive in both the primary and secondary caches.

2.3 Cache Line Write Back

The processor will write a cache line back to memory when it is replaced, or written back to memory as the result of executing a cache instruction, if the cache line is in the state dirty exclusive. When the processor writes a cache line back to memory, it does not ordinarily retain a copy of the cache line, and the state of the cache line is changed to invalid. However, if a cache line is written back to memory using the hit writeback cache instruction, the processor will retain a copy of the cache line. If the cache line is retained, the processor will change its state to clean exclusive if the secondary cache state was dirty exclusive before the write.

Whether or not the processor is retaining the line is signaled by the processor during a write.

2.4 Manipulation of the Caches by an External Agent

The VR4000SC provides mechanisms for an external agent to examine and manipulate the state and contents of the primary and secondary caches:

An external agent must specify the state in which data, supplied in response to a processor read request, is to be loaded into the processor's caches. Data may be loaded in any of the four valid secondary cache states. Data returned by the external agent must not be marked as invalid. The secondary cache state will be mapped to a primary cache state as described previously.

An external agent may issue a snoop request to the processor which will cause the processor to return the secondary cache state of the specified cache line. At the same time it will change the state of the specified cache line in both the primary and secondary caches, according to a state change function specified by the external agent, atomically with respect to the response to the snoop request.

An external agent may issue an invalidate request or an update request to the processor. An invalidate request will cause the processor to change the state of the specified cache line to invalid in both the primary and secondary caches. An update request will cause the processor to write the specified data element into the specified cache line, and either change the state of the cache line to shared in both the primary and secondary caches, or leave the state of the cache line unchanged, depending on the nature of the update request. An external agent may issue updates, without changing the state of the cache line, to cache lines that are either in exclusive or shared states.

An external agent may issue an intervention request which will cause the processor to return the secondary cache state of the specified cache line, and the contents of the specified secondary cache line under certain conditions related to the state of the cache line and the nature of the intervention request. At the same time the processor will change the state of the specified cache line in both the primary and secondary caches, according to a state change function specified by the external agent, atomically with respect to the response to the intervention request.

2.5 Cache Line Ownership

The VR4000SC has a concept of ownership for cache lines. The ownership of a cache line is maintained as follows:

A processor assumes ownership of a cache line when the state of the cache line transitions to dirty shared or dirty exclusive. For responses to processor coherent read requests in which the data is returned with an indication that it must be loaded in the dirty shared or dirty exclusive state, the cache state is set at the completion of the read response when the last word of read response data is returned. Therefore, the processor will assume ownership of the cache line at the completion of the read response when the last word of read response data is returned.

The processor gives up ownership of a cache line when the state of the cache line transitions to invalid, shared, or clean exclusive. For processor write requests the state of the cache line will transition to invalid if the cache line is replaced, or clean exclusive or shared if the cache line is retained. In either case, the cache state transition will occur at the completion of the write request when the last word of write data is transmitted to the external agent. Therefore, the processor will give up ownership of the cache line at the completion of the write request when the last word of write data is transmitted to the external agent.

For external requests, other than read responses, any cache state change associated with the external request will occur at the completion of the external request and therefore any change of ownership resulting from the cache state change will occur at the completion of the external request.

CHAPTER 3 SECONDRY CACHE INTERFACE

The VR4000SC is designed to operate with an external secondary cache. The secondary cache is accessible to the processor and to the system interface. The cache contains data, cache tags and cache line state bits.

3.1 Secondary Cache Overview

The VR4000SC secondary cache is assumed to consist of one bank of industry standard static RAMs with output enables. The VR4000SC secondary cache consists of quad-word (128 bit) wide data array and a 25-bit wide tag array. Check fields are added to both the data and tag arrays to improve data integrity. The secondary cache may be configured as joint or split instruction/data. The maximum secondary cache size is 4 MBytes and the minimum secondary cache size is 128 kBytes for joint and 256 kBytes for split instruction/data. The secondary cache is direct-mapped, and is addressed with the lower part of the physical address.

3.2 Secondary Cache Interface Signal Description

The signals that connect the VR4000SC processor to its secondary cache are described in this section.

3.2.1 Secondary Cache Interface Signal Summary

SCData(127:0):	(i/o)	A 128-bit bus used to read or write cache data from/to the secondary cache.
SCDChk(15:0):	(i/o)	A 16-bit bus which conveys two ECC fields that cover the upper or lower 64 bits of the SCData from/to the secndary cache.
SCTag(24:0):	(i/o)	A 25-bit bus used to read or write cache tags from/to the secondary cache.
SCTChk(6:0):	(i/o)	A 7-bit bus which conveys an ECC field that covers the **SCTag** from/to the secondary cache.
SCAddr(17:1)	(o)	A 17-bit bus which addresses the secondary cache.
SCAddr0Z:	(o)	Bit 0 of the secondary cache address.
SCAddr0Y:	(o)	Bit 0 of the secondary cache address.
SCAddr0X:	(o)	Bit 0 of the secondary cache address.
SCAddr0W:	(o)	Bit 0 of the secondary cache address.
SCAPar(2:0):	(o)	A 3-bit bus which conveys the parity of the **SCAddr** bus and the cache control lines $\overline{\textbf{SCOE}}$, $\overline{\textbf{SCWr}}$, $\overline{\textbf{SCDCS}}$ and $\overline{\textbf{SCTCS}}$.
$\overline{\textbf{SCOE}}$**:**	(o)	A signal which enables the outputs of the secondry cache RAMs.
$\overline{\textbf{SCWrZ}}$**:**	(o)	Secondary cache write enable.
$\overline{\textbf{SCWrY}}$**:**	(o)	Secondary cache write enable.
$\overline{\textbf{SCWrX}}$**:**	(o)	Secondary cache write enable.
$\overline{\textbf{SCWrW}}$**:**	(o)	Secondary cache write enable.
$\overline{\textbf{SCDCS}}$**:**	(o)	A signal which enables the chip select pins of the secondary cache RAMs associated with **SCData** and **SCDChk**.
$\overline{\textbf{SCTCS}}$**:**	(o)	A signal which enables the chip select pins of the secondary cache RAMs associated with **SCTag** and **SCTChk**.

3.2.2 Details of Secondary Cache Interface Signals

The interface to the VR4000SC secondary cache is designed to maximize the efficiency of servicing primary cache misses. The width of the data portion of secondary cache interface is chosen to be 128 bits to support a data rate into the primary cache that is near the processor to primary cache bandwidth during normal operation. To assure that this bandwidth is maintained, each data, tag and check pin must be connected to only one static RAM device. The **SCAddr** bus, the **$\overline{\text{SCOE}}$** signal, the **$\overline{\text{SCDCS}}$** signal and the **$\overline{\text{SCTCS}}$** drive a large number of static RAM devices, so one level of external buffering between the VR4000SC and the cache array is necessary.

The speed of the secondary cache interface is limited by buffered control signals. Critical control signals are duplicated to minimize this effect. The **$\overline{\text{SCWr}}$** signal and **SCAddr(0)** are duplicated four times so that external buffering will not be required. When an 8-word (256-bit) primary cache line is used, these signals can be controlled significantly faster to reduce the time of the two back-to-back transfers. These duplicated control signals are specified to drive 11 parts each, so that a total of 44 RAM packages can be used in the cache array. This permits a cache design using 64 kByte by 4 bit or 256 kByte by 4 bit standard static RAMs. Other cache designs are also acceptable, for example a smaller cache design using 228 kByte by 8 bit static RAMs as it would present less load on the address pins and control signals. Note that duplicated signals like **$\overline{\text{SCWrW}}$**, **$\overline{\text{SCWrX}}$**, **$\overline{\text{SCWrY}}$** and **$\overline{\text{SCWrZ}}$** will be described in this document as though they were a single signal, which in this case is called **$\overline{\text{SCWr}}$**.

The benefit of duplicating **SCAddr(0)** will be greater if fast sequential static cache RAMs become available. If **SCAddr(0)** is attached to a static RAM address bit that effects column decode only, the read cycle time with respect to that pin should approximate the output enable time of the RAM and for fast static RAMs should be half that of the nominal read cycle time.

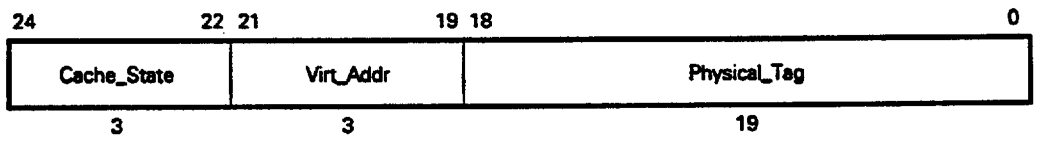
When the split instruction/data cache mode is enabled, assertion of the top **SCAddr** bit, **SCAddr(17)** will enable the instruction half of the cache instead of the data half.

It is possible to design a cache that supports both joint and split instruction/data configurations with less than the maximum cache size. SCAddr(12:0) must be used to address the cache in all configurations. SCAddr(17) must be used to support the split instruction/data configuration. Any of SCAddr(16:13) may be omitted because of the fixed width of the physical tag array.

The **SCDChk** bus is divided into two fields to cover the upper and lower 64 bits of **SCData**. This form is required to keep the width of internal data paths to 64 bits.

The **SCTag** bus is divided into three fields, as shown in **Fig. 3-1 SCTag Fields**. The lower 19 bits consist of the upper physical address bits. The upper three bits consist of cache state, which can be one of Invalid (I), Clean Exclusive (CE), Dirty Exclusive (DE), Shared (S) and Dirty Shared (DS). The middle three bits are used to maintain information about the virtual address used for caching parts of a secondary cache line in the primary cache. The middle three-bit field holds bits 14 through 12 of the virtual address.

Fig. 3-1 SCTag Fields



The middle field of **SCTag** is needed to locate entries in the primary caches. The VR4000SC has two primary caches, one for instruction and one for data, which are direct-mapped and are indexed using a subset of the lower 15 bits of the virtual address (implementation dependent, based on primary cache size). If a cache coherency request is processed that requires a cache state change or invalidation, the middle three bits of the **SCTag** portion of the secondary cache allow primary cache lines affected by that cache coherency request to be found. The three bits of information stored are the three lowest virtual address bits above the page offset. This information is loaded during secondary cache misses. On each secondary cache access the virtual address bits are compared with the values found in the secondary cache tag. If a mismatch occurs, a trap is taken and the trap handler can modify the bits in the secondary cache tag to hold the new values, and the old values are used by the trap handler to purge primary cache locations, so that all primary cache lines holding valid data have indexes known to the secondary cache. This mechanism also helps preserve the integrity of cached accesses to a physical address using differing virtual addresses known as virtual synonyms.

The $\overline{\textbf{SCDCS}}$ and $\overline{\textbf{SCTCS}}$ are needed to disable reads or writes of the data array or tag array when the other array is being accessed. These signals are useful for saving power on snoop and invalidate requests, as accesses to the data array are not necessary. These signals are also useful for writing data from the data primary cache to the secondary cache, as the secondary cache state cannot always be determined from the primary cache state.

3.3 Control of Secondary Cache Interface

The control of the secondary cache is configurable for various clock rates and static RAM speeds. All configurable parameters are specified in multiples of PClock, which runs at twice the frequency of the external system clock, MasterClock. Boot time mode control registers will hold the various configuration parameters, so that they can be specified when initializing the VR4000SC.

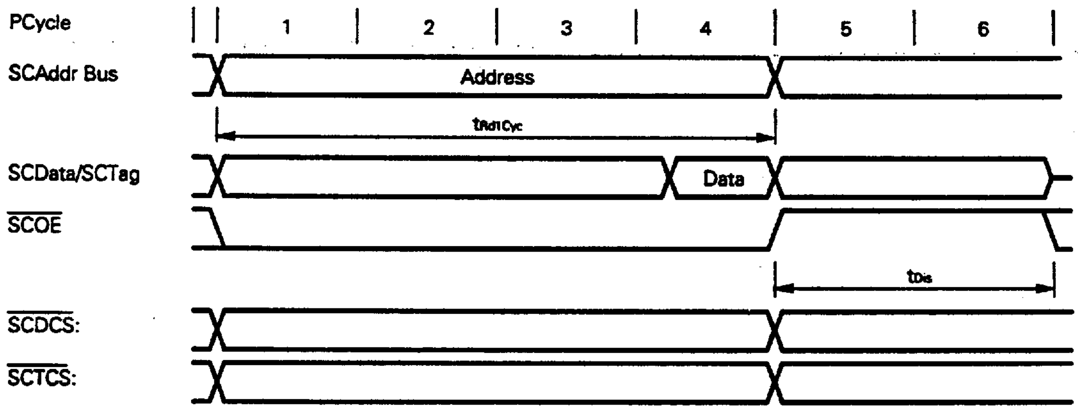
3.3.1 Read Cycles

Each secondary cache read sequence begins with the driving of the address pins. The output enable signal $\overline{\textbf{SCOE}}$ is asserted at the same time.

There are two basic read cycles: a four-word read, and an eight-word read.

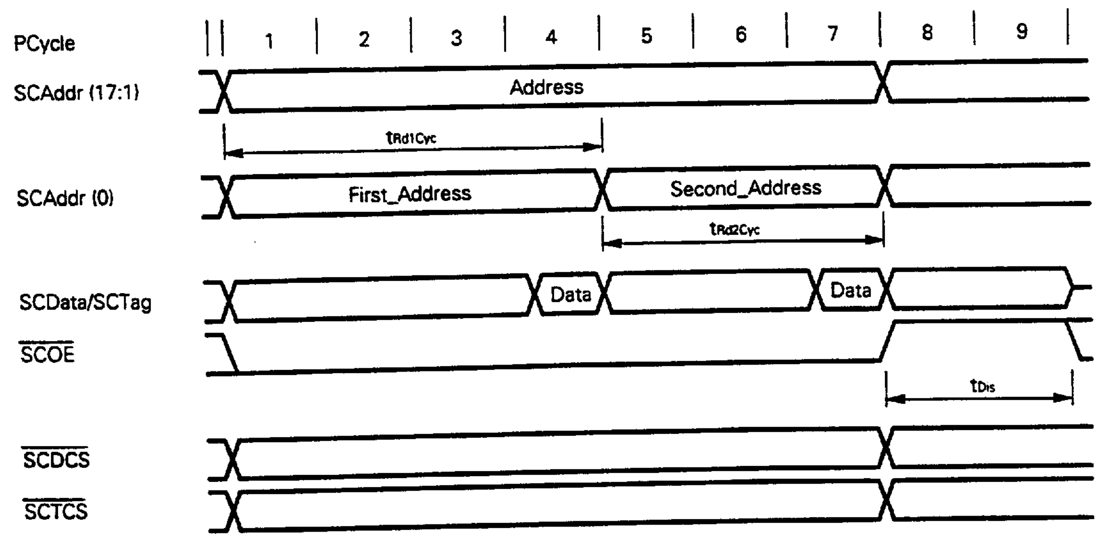
For the four-word read, there are two parameters of interest. The first parameter is read sequence cycle time, TRd1Cyc, which specifies the time from the driving of the **SCAddr** bus to the sampling of the **SCData** bus. The second parameter is the cache output disable time TDis, which specifies the time from the end of a read cycle to the start of the next write cycle. **Fig. 3-2 Four-Word Read Cycle** illustrates the four word read sequence.

Fig. 3-2 Four-Word Read Cycle



For the eight-word read, there is one additional parameter of interest: the time from the first sample point to the second sample point, TRd2Cyc. The lower order address bit, **SCAddr(0)** is changed at the same time as the first read sample point. **Fig. 3-3 Eight-Word Read Cycle** illustrates the eight word read sequence.

Fig. 3-3 Eight-Word Read Cycle



All read cycles can be aborted by changing the address. A new cycle starts beginning with the edge on which the address is changed. Additionally, the period TDis after a read cycle can be interrupted any time by the start of a new read cycle. If a read cycle is aborted by a writ cycle, **SCOE** must be deasserted for the TDis period, before the write cycle can commence. Read cycles can also be extended indefinitely. There is no requirement to change the address at the end of a read cycle.

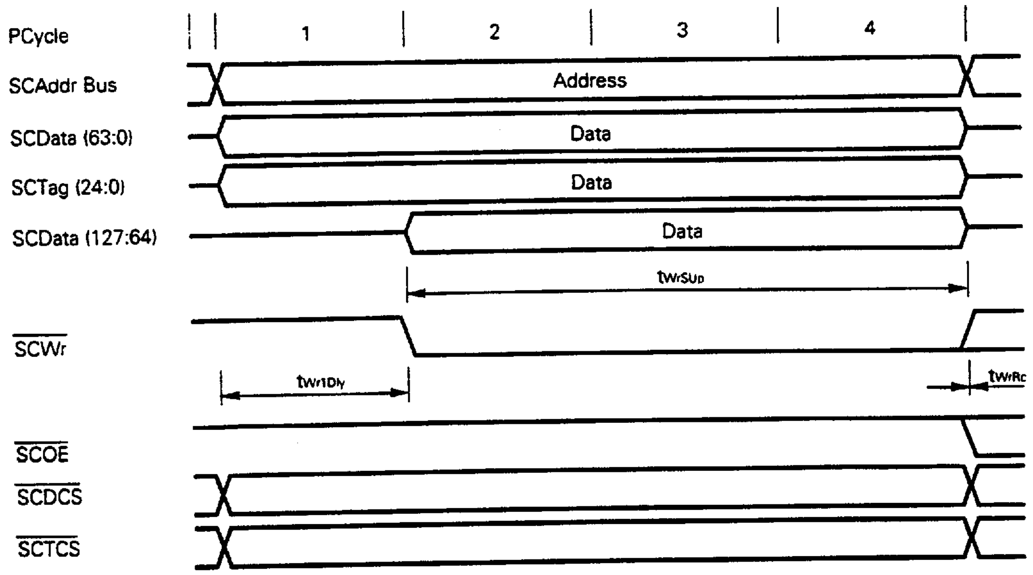
3.3.2 Write Cycles

Like the read sequence, the secondary cache write sequence begins with the driving of the address pins.

There are two basic write cycles: a four-word write, and an eight word write.

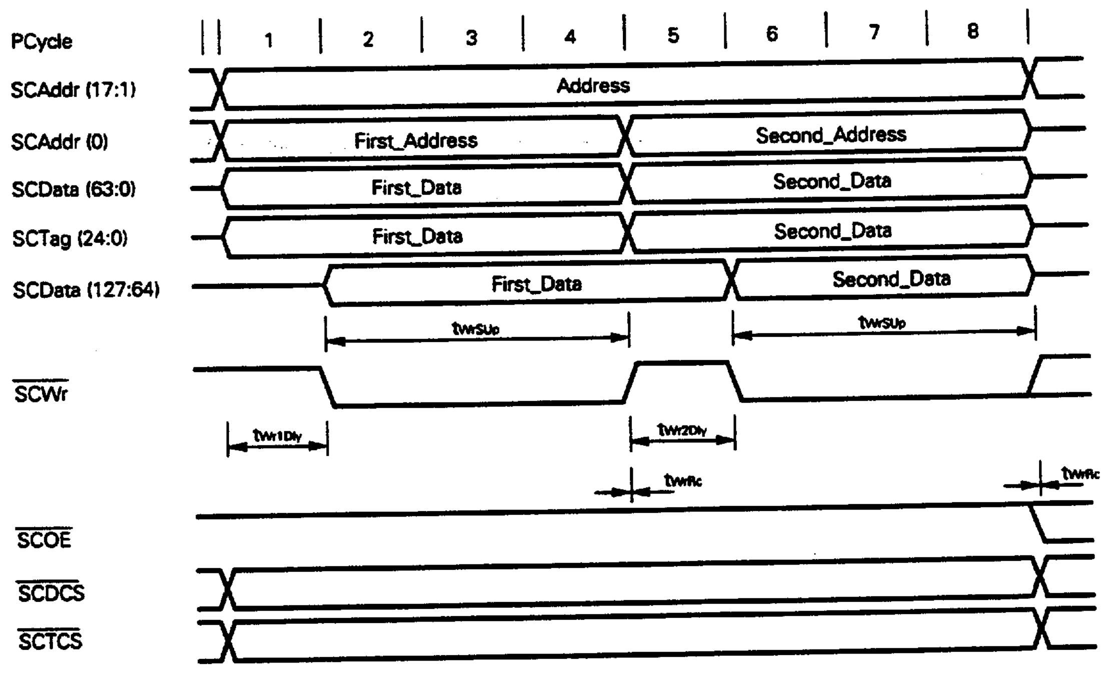
For the four-word write, there are several parameters of interest. The first parameter, TWr1Dly is the time from driving address to the assertion of **SCWr**. The second parameter, TWrSUp is the time from driving the second data double-word to the deassertion of **SCWr**. The final parameter, TWrRc, is the time from the deassertion of **SCWr** to the beginning of the next cycle. TWrRc will be zero for most cache designs. Note that the upper data double word and the lower data double word will normally be driven one cycle apart. This reduces the peak current consumption in the output drivers. **Fig. 3-4 Four-Word Write Cycle** illustrates the four word write sequence. The order of driving the upper versus the lower halves of SCData are not fixed, either the upper or the lower halves might be driven first.

Fig. 3-4 Four-Word Write Cycle



The eight word write has one new parameter. The parameter, TWr2Dly, is the time from changing the low-order address bit **SCAddr(0)** to the assertion of **$\overline{\text{SCWr}}$** the second time. The lower half of **SCData** will be driven out on the same edge as the change in **SCAddr(0)**. **Fig. 3-5 Eight-Word Write Cycle** illustrates the eight word write sequence.

Fig. 3-5 Eight-Word Write Cycle



When receiving data from the system interface, it is possible that the first data double word will arrive several cycles before the second. In this case, the cache state machine will simply wait until that data is available before asserting **$\overline{\text{SCWr}}$** and will extend the **$\overline{\text{SCWr}}$** until TWrSUp after the driving of the second data item.

CHAPTER 4 SYSTEM INTERFACE

The system interface allows the processor access to external resources required to satisfy cache misses while also allowing an external agent access to certain processor internal resources. In the large package configuration, the system interface also provides the processor mechanisms with which to maintain the cache coherency of shared data, while providing an external agent mechanisms with which to maintain system wide multiprocessor cache coherency.

4.1 System Interface Overview

An event that occurs within the processor that requires access to external system resources will be referred to as a **system event**. System events include, a load that misses in both the primary and secondary caches, a store that misses in both the primary and secondary caches, a store that hits in either the primary or secondary data cache on a shared line, and an uncached load or store. A miss in both caches will require the write back to memory of the cache line that is being replaced if it is in one of the dirty cache states. Cache instructions will also cause system events under certain circumstances. For more details on VR4000SC Cache instructions see **VR4000 USER'S MANUAL (PRELIMINARY) ARCHITECTURE.**

When a system event occurs, the processor will issue a request or a series of requests called **processor requests** through the system interface to access some external resource and service the event. The processor's system interface must be connected to some external agent that understands the system interface protocol and can coordinate the access to system resources.

Processor requests include read, write, null write requests. Reads are requests for a block, double word, word or partial word of data from main memory or another system resource. Writes provide a block, double word, word or partial word of data to be written to main memory or another system resource. Null writes indicate that an expected write has been obviated as a result of some external request.

An external agent may require access to the processor's caches or to some processor internal resource. In this event the external agent will issue a request to the processor through the system interface called an **external request** to provide the access.

External requests include read, write, invalidate, update, snoop, intervention, and null requests. Reads are requests for a word of data from some processor internal resource. Writes provide a word of data to be written to some processor internal resource. Invalidates specify a cache line that must be marked invalid in the processor's primary and secondary caches. Updates provide a double word, word or partial word of data to be written to the processor's primary and secondary caches. Snoop requests are used to interrogate the processor's secondary cache to discover if the processor has a valid copy of a particular cache line and if so what cache state the line is in. Snoop requests require the processor to return an indication of the state of the cache line at the specified physical address in the secondary cache if it is present. Intervention requests require the processor to return an indication of the state of the cache line at the specified physical address in the secondary cache and the contents of the cache line from the primary and secondary caches under certain conditions related to the state of the cache line and the nature of the intervention request. Null requests require no action by the processor, rather they simply provide a mechanism for an external agent to either return control of the secondary cache to the processor, or to return control of the system interface to the processor.

When the processor or an external agent receives a read request, it must access the specified resource and return the requested data. For external read requests, the data will be returned directly in response to the read request. For processor read requests the read request and the return of data by an external agent in response to the read request are disconnected or split. The response data may be returned at any time after the read request, and the system interface is not in use by the read during the time between the read request and the return of response data. An external agent may initiate an unrelated external request before it returns the response data for a processor read.

The return of data in response to a processor read request will be accomplished via a read response. While a read response is technically also an external request, read responses have different characteristics than all other external requests in that arbitration for the system interface must not be performed for read responses. For this reason, read responses will be treated separately from all other external requests, and will be called simply read responses.

Processor read requests that have been issued but for which data has not yet been returned are said to be pending. A read is pending until the read data has been returned. Note that the data identifier associated with the response data may signal that the returned data is erroneous, causing the processor to take a bus error.

External read requests are not split. The system interface is in use between the read request and the return of data by the processor.

A processor read request is complete after the last word of response data has been received from the external agent. A processor write request is complete after the last word of data has been transmitted.

An external read request is complete after the processor returns the requested word of data. An external write request is complete after the word of data has been transmitted. An external invalidate or update request is complete after the request has been transmitted. An external snoop request is complete after the processor returns the state of the specified cache line. An external intervention request is complete after the processor returns the state of the specified cache line, if the processor does not return the contents of the cache line, or after the processor returns the last word of data for the specified cache line.

The processor must manage the flow of processor requests and external requests. The flow of external requests is controlled by the processor via the external request arbitration signals $\overline{\textbf{ExtRqst}}$, and $\overline{\textbf{Release}}$. An external agent must acquire mastership of the system interface before it is allowed to issue an external request by asserting $\overline{\text{ExtRqst}}$ and waiting for the processor to assert $\overline{\text{Release}}$ for one cycle. The processor will not assert $\overline{\text{Release}}$ until it is ready to accept an external request. Mastership of the system interface is always returned to the processor after an external request has been issued, and the processor will not accept a subsequent external request until it has finished the current one.

Processor requests are managed by the processor in two distinct modes, **secondary cache mode** and **no-secondary cache mode**, which reflect the presence or absence of a secondary cache, programmable via the boot time mode control interface. The allowed modes of operation are dependent on the package configuration for the processor. A processor in the large configuration package may be programmed to run in secondary cache mode or no-secondary cache mode.

In no-secondary cache mode, the processor will issue requests in a strict sequential fashion; that is, the processor is only allowed to have one request pending at any time. The processor will issue a read request and wait for the read response before issuing any subsequent requests. The processor will issue a write request only if there are no reads pending.

The processor provides the signals $\overline{\textbf{RdRdy}}$ and $\overline{\textbf{WrRdy}}$ to allow an external agent to manage the flow of processor requests. $\overline{\text{RdRdy}}$ controls the flow of processor read, invalidate, and update requests while $\overline{\text{WrRdy}}$ controls the flow of processor write requests. Processor null write requests must always be accepted, they cannot be delayed by either $\overline{\text{RdRdy}}$ of $\overline{\text{WrRdy}}$. The processor samples the signal $\overline{\text{RdRdy}}$ to determine the issue cycle for a processor read, invalidate, or update request and the processor samples the signal $\overline{\text{WrRdy}}$ to determine the issue cycle of a processor write request. The issue cycle for a processor read request is defined to be the first address cycle for the request for which the signal $\overline{\text{RdRdy}}$ was asserted two cycles previously. The issue cycle for a processor write request is defined to be the first address cycle for the write request for which the signal $\overline{\text{WrRdy}}$ was asserted two cycles previously. If the processor wishes to issue a request but is unable to because one of the signals $\overline{\text{RdRdy}}$ or $\overline{\text{WrRdy}}$ is de-asserted, the processor will repeat the address cycle for the request until the issue cycle is accomplished. Once the issue cycle is accomplished, data transmission will begin for a request that includes data. There will always be one and only one issue cycle for any processor request.

The processor will accept external requests while attempting to issue a processor request by releasing the system interface to slave state in response to an assertion of $\overline{\text{ExtRqst}}$. Note that the rules governing the issue cycle of a processor request are strictly applied to determine the action the processor is taking. The processor will either accomplish the issue of the processor request, in which case the processor request will be completed in its entirety before an external request will be accepted, or the processor will release the system interface to slave state without accomplishing the issue of the processor request. In the latter case, the processor will attempt to issue the processor request again after the external request is completed, and the rules governing issue cycle will again apply.

In no-secondary cache mode an external agent must be capable of accepting a processor read request at any time there are no processor read requests pending and the signal $\overline{\text{RdRdy}}$ has been asserted for two or more cycles. An external agent must be capable of accepting a processor write request at any time there are no processor read requests pending and the signal $\overline{\text{WrRdy}}$ has been asserted for two or more cycles.

In secondary cache mode, the processor will issue requests both individually as in no-secondary cache mode and in groups that begin with a processor read request called **clusters**. Specifically, the processor will issue individual read requests and write requests and the processor will issue clusters. A cluster consists of a processor read request followed by one or two additional processor requests issued while the read request is pending. All of the requests that are part of a cluster must be accepted before the response to the read request that begins the cluster may be returned to the processor. A cluster will consist of a processor read request followed by a write request.

A write request that is part of a cluster does obey the $\overline{\text{WrRdy}}$ rules for issue, and the processor will accept external requests between the issue of a processor read request and the issue of a processor write request within a cluster. The processor signals that it is issuing a cluster that contains a processor write request by issuing a read with write forthcoming request instead of an ordinary read request to start the cluster. The read with write forthcoming request is identified by a bit in the command for processor read requests. The external agent must accept all of the requests that form a cluster before it may return a response to the read request that began the cluster. The behavior of the processor is undefined if the external agent returns a response to a processor read request that begins a cluster before accepting all of the requests that form the cluster.

Since the processor will accept external requests between the issue of a read with write forthcoming request that begins a cluster and the issue of the write request that completes the cluster, it is possible for an external request to obviate the need for the write request within the cluster. For instance, if the external agent were to issue an external invalidate request that targeted the cache line the processor was attempting to write back, the state of the cache line would be changed to invalid, and the write back for the cache line would no longer be needed. In this event, the processor will issue a processor null write request after completing the external request to complete the cluster. Processor null write requests do not obey the $\overline{\text{WrRdy}}$ flow control rules for issue, but rather issue with a single address cycle regardless of the state of $\overline{\text{WrRdy}}$. Any external request that changes the state of a cache line from dirty exclusive or dirty shared to clean exclusive, shared, or invalid will obviate the need for a write back of that cache line.

In secondary cache mode, an external agent must be capable of accepting a processor write request at any time there are no processor read requests pending, or there is a processor read with write forthcoming request pending, no unacknowledged processor invalidate or update requests that are compulsory, and the signal $\overline{\text{WrRdy}}$ has been asserted for two or more cycles.

After issuing a processor read request, the processor will not attempt to issue a subsequent read request until it has received a read response for the read request, whether it began a cluster or not. The processor will not attempt to issue a subsequent request until at least four cycles after the issue cycle of the write request.

The following sections detail the sequence, protocol and syntax of processor and external requests. Sequence refers to the precise series of requests that a processor generates to service a system event. Protocol refers to the cycle by cycle signal transitions that occur on the processor's system interface pins to realize a processor or external request. Syntax refers to the precise definition of bit patterns on encoded buses such as the command bus.

4.2 Processor Request Sequencing

The processor will generate a request or a series of requests through the system interface to satisfy system events. Processor requests are managed in two distinct modes, secondary cache mode and no-secondary cache mode. The following sections detail the sequence of requests generated by the processor for each system event in secondary cache and no-secondary cache mode.

4.2.1 Primary and Secondary Cache Miss on a Load

When the processor misses in both the primary and secondary caches on a load, it must obtain the cache line that contains the data element to be loaded from an external agent before it can proceed. If the new cache line will replace a current cache line that is in the state dirty exclusive or dirty shared, the current cache line must be written back before the new line can be loaded in the primary and secondary caches.

The processor will examine the coherency attribute in the TLB entry for the page that contains the requested cache line and if the coherency attribute is exclusive it will issue a coherent read request that also requests exclusivity. If the coherency attribute is sharable or update the processor will issue a coherent read request and if the coherency attribute is noncoherent the processor will issue a noncoherent read request.

In no-secondary cache mode, the processor will issue a read request for the cache line that contains the data element to be loaded, wait for an external agent to provide the read data in response to the read request, and then, if the current cache line must be written back, the processor will issue a write request for the current cache line.

In secondary cache mode, the processor will issue a read request for the cache line that contains the data element to be loaded if the current cache line does not need to be written back and the coherency attribute for the page that contains the requested cache line is anything other than exclusive. If the current cache line needs to be written back and the coherency attribute for the requested cache line is not exclusive, the processor will issue a cluster consisting of a read with write forthcoming request for the cache line that contains the data element to be loaded followed by a write request for the current cache line. If the current cache line needs to be written back, the coherency attribute for the page that contains the requested cache line is exclusive, the processor will issue a cluster consisting of an exclusive read with write forthcoming request followed by a write request for the current cache line.

4.2.2 Primary and Secondary Cache Miss on a Store

When the processor misses in both the primary and secondary caches on a store, it must obtain the cache line that contains the target location of the store from an external agent before it can proceed. In secondary cache mode, if the new cache line will replace a current cache line that is in the state dirty exclusive or dirty shared, the current cache line must be written back before the new line can be loaded in the primary and secondary caches. In no-secondary cache mode, if the new cache line will replace a current cache line that is in the state dirty exclusive, the current cache will be moved to an internal write buffer before the new cache line is loaded in the primary cache.

The processor will examine the coherency attribute in the TLB entry for the page that contains the requested cache line to see if this cache line is being maintained with a write invalidate or a write update cache coherency protocol. If the coherency attribute is sharable or exclusive a write invalidate protocol is in effect, and a coherent read that also requests exclusivity will be issued. If the coherency attribute is update a write update protocol is in effect and a coherent read request will be issued. If the coherency attribute is noncoherent a noncoherent read request will be issued.

In no-secondary cache mode, the processor will issue a read request for the cache line that contains the data element to be loaded, wait for an external agent to provide the read data in response to the read request, and then, if the current cache line must be written back, the processor will issue a write request for the current cache line.

In secondary cache mode, the processor will issue a read request for the cache line that contains the target location of the store if the current cache line does not need to be written back and the coherency attribute for the page that contains the requested cache line is noncoherent. If the current cache line does not need to be written back, the coherency attribute for the page that contains the requested cache line is sharable or exclusive, the processor will issue a cluster consisting of a read request. If the current cache line needs to be written back and the coherency attribute for the requested cache line is noncoherent, the processor will issue a cluster consisting of a read with write forthcoming request for the cache line that contains the target location of the store followed by a write request for the current cache line. If the current cache line needs to be written back, the coherency attribute for the page that contains the requested cache line is sharable or exclusive, the processor will issue a cluster consisting of a read with write forthcoming request, followed by a write request for the current cache line.

4.2.3 Secondary Cache Hit on a Store to a Shared Line

When the processor hits in the secondary cache on a line that is marked shared or dirty shared, an invalidate or update request must be issued and receive an acknowledge before the store can be completed. The processor will check the coherency attribute in the TLB for the page that contains the cache line that is the target of the store to determine if the cache line is being managed using a write invalidate or write update cache coherency protocol. If the coherency attribute is sharable or exclusive a write invalidate protocol is in effect, and the processor will issue an invalidate request. If the coherency attribute is update a write update protocol is in effect, and the processor will issue an update request. The processor will not complete the store until an external agent signals an acknowledge for the invalidate or update request.

4.2.4 Uncached Load or Store

When the processor performs an uncached load, it will issue a noncoherent read request. When the processor performs an uncached store, it will issue a write request.

4.2.5 Cache Instructions

The VR4000SC processor provides a variety of cache instructions for use in maintaining the state and contents of the primary and secondary caches. During the execution of cache instructions the processor may issue write requests, or invalidate requests. For further details on cache instructions see **VR4000 USER'S MANUAL (PRELIMINARY) ARCHITECTURE.**

4.3 External Request Handling

An external agent must arbitrate with the processor for access to the system interface before it can issue an external request. The external agent will signal that it wishes to begin an external request and wait for the processor to signal that it is ready to accept the request before issuing any new external request. The processor will decide based on its internal state and the current state of the system interface when to accept a new external request. The processor will signal that it is ready to accept an external request based on the following criteria.

(1) If there are no processor requests pending, the processor will decide based on its internal state whether to accept the external request, or rather to issue a new processor request. The processor may issue a new processor request while the external agent is requesting access to the system interface to issue an external request.

(2) The processor will accept an external request after completing a processor request or a processor request cluster that is in progress.

(3) While waiting for the assertion of $\overline{\text{RdRdy}}$ to issue a processor read request the processor will accept an external request provided that the request is delivered to the processor one or more cycles before $\overline{\text{RdRdy}}$ is asserted.

(4) While waiting for the assertion of $\overline{\text{WrRdy}}$ to issue a processor write request the processor will accept an external request provided that the request is delivered to the processor one or more cycles before $\overline{\text{WrRdy}}$ is asserted.

(5) While waiting for the response to a read request and after the VR4000SC has made an uncompelled change to slave state, an external agent may submit an external request before providing the read response data.

4.4 Load Linked Store Conditional Considerations

Generally the execution of a load linked store conditional instruction sequence is not visible at the system interface, that is no special requests are generated due to the execution of this instruction sequence. However, there is one situation for which the execution of a load linked store conditional instruction sequence will be visible as a change in the nature of a processor read request.

Specifically, if the data location targeted by a load linked store conditional instruction sequence maps to the same cache line that the instruction area containing the load linked store conditional code sequence is mapped to, then immediately after executing the load linked instruction the cache line that contains the link location will be replaced by the instruction line containing the code. The link address is kept in a register separate from the cache and remains active as long as the link bit remains set. The link bit is set by the load linked instruction, and is cleared by any change of cache state for the cache line containing the link address, or a return from exception.

In order for the load linked store conditional instruction sequence to work correctly all coherency traffic targeting the link address must be visible to the processor, and the cache line containing the link location must remain in a shared state in every cache in the system. This guarantees that a store conditional executed by some other processor is visible to the processor as a coherence request which changes the state of the cache line that contains the link location. To accomplish this, a read request issued by the processor which causes the replacement of a cache line that contains the link location while the link bit is set will indicate that the link address is being retained. The link address retained bit in the command for the read request will be asserted to provide this indication. This informs the external agent that even though the processor has replaced this cache line and no longer has it present in its cache, it still must see any coherence traffic that targets this cache line.

In addition, any snoop or intervention request that targets a cache line which is not present in the cache, but for which the snoop or intervention address matches the current link address while the link bit is set, will return an indication that the cache line is present in the cache in a shared state. A shared indication is returned even though the processor does not actually have the data content of the cache line. This is consistent since the processor never returns data in response to an intervention request for a cache line that is in the shared state. The shared response guarantees that the cache line that contains the link location will remain in a shared state in all other processor's caches, and therefore that any other processor that attempts a store conditional to this link location must issue a coherence request in order to complete the store conditional.

4.5 System Interface Endianess

The endianess of the system interface is programmed at boot time via the boot time mode control interface and is fixed until the next time the processor mode bits are read. The endianess of the system interface and the external system cannot be changed by software. The reverse endian bit can be set by software to reverse the interpretation of endianess inside the processor, but the endianess of the system interface remains unchanged. For further details on the reverse endian bit see **VR4000 USER'S MANUAL (PRELIMINARY) ARCHITECTURE**.

4.6 System Interface Protocol

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.

4.6.1 Introduction

The system interface is register to register. That is, processor outputs come directly from output registers and begin to change with the rising edge of SClock and processor inputs are fed directly to input registers that latch the inputs with the rising edge of SClock. Therefore, if an input to the processor is changed during a particular cycle such that the new value is sampled at the end of the cycle, the earliest the processor can change one of its outputs in response to the input change is two cycles later. This methodology was chosen to allow the system interface to run at the highest possible clock frequency.

The primary communication paths for the system interface are a sixty-four bit address and data bus, **SysAD(63:0)** and a nine bit command bus, **SysCmd(8:0)**. The SysAD bus and the SysCmd bus are bidirectional, that is they are driven by the processor to issue a processor request, and by an external agent to issue an external request. When the processor is driving the SysAD bus and the SysCmd bus the system interface is in **master state**, when an external agent is driving the SysAD bus and the SysCmd bus the system interface is in **slave state**.

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

Cycles in which the SysAD bus contains a valid address are called address cycles and cycles in which the SysAD bus contains a valid data element are called data cycles. In master state the processor will assert the signal $\overline{\textbf{ValidOut}}$ whenever the SysAD bus and the SysCmd bus are valid. In slave state an external agent will assert the signal $\overline{\textbf{ValidIn}}$ whenever the SysAD bus and the SysCmd bus are valid.

The SysCmd bus is used to identify the contents of the SysAD bus during any cycle in which it is valid. The most significant bit of the SysCmd bus is always used to indicate whether the current cycle is an address cycle or a data cycle. During address cycles, the remainder of the SysCmd bus will contain a system interface command. The encoding of system interface commands is detailed in the section on system interface syntax. During data cycles, the remainder of the SysCmd bus will contain an indication of whether this is the last data element to be transmitted and other information about the data element. The contents of the SysCmd bus during data cycles is called a **data identifier**. The encoding of data identifiers is detailed in **4.8 System Interface Syntax**.

A request through the system interface consists of one or more identical address cycles, followed by a series of data cycles for requests that include data. The most efficient request through the system interface will consist of a single address cycle followed by a single data cycle or a number of data cycles sufficient to transmit a block of data.

4.6.2 System Interface Arbitration

When an external agent needs to issue an external request through the system interface, it must first get the system interface into slave state. The transition from master state to slave state is arbitrated by the processor using the system interface handshake signals $\overline{\text{ExtRqst}}$ and $\overline{\text{Release}}$. An external agent will signal that it wishes to issue an external request by asserting $\overline{\text{ExtRqst}}$, and the processor will release the system interface from master state to slave state by asserting $\overline{\text{Release}}$ for one cycle when it is ready to accept an external request. The system interface will return to master state as soon as the issue of the external request is complete. Having asserted $\overline{\text{ExtRqst}}$, an external agent must not de-assert $\overline{\text{ExtRqst}}$ until the processor asserts $\overline{\text{Release}}$. After the processor asserts $\overline{\text{Release}}$, the external agent should de-assert $\overline{\text{ExtRqst}}$ no more than two cycles after the assertion of $\overline{\text{Release}}$. An external agent may continue to assert $\overline{\text{ExtRqst}}$ if another external request follows the current request. After the first external request completes, the processor must assert $\overline{\text{Release}}$ again before the second external request is issued to the processor.

The system interface will remain in master state until the external agent requests and is granted the system interface or until the processor issues a read request, or completes the issue of a cluster. Whenever a processor read request is pending, after the issue of a read request or after the issue of all of the requests in a cluster, the processor will switch the system interface to slave state even though the external agent is not arbitrating to issue an external request. This transition to slave state is specifically to allow the external agent to return read response data. The external agent must not assert the signal $\overline{\text{ExtRqst}}$ for the purposes of transitioning the system interface to slave state to return read response data. $\overline{\text{ExtRqst}}$ will only be asserted when the external agent needs to get the system interface into slave state so that it can issue an external request

The signal $\overline{\text{ExtRqst}}$ is strictly used to arbitrate for the system interface that is to request the transition of the system interface from master state to slave state. $\overline{\text{ExtRqst}}$ must always de-assert two cycles after a cycle in which $\overline{\text{Release}}$ is asserted unless the external agent wishes to perform a subsequent external request. $\overline{\text{ExtRqst}}$ must not be asserted while the system interface is in slave state unless the external agent wishes to perform a subsequent external request.

The transition of the system interface from master state to slave state initiated by the processor when a processor read request is pending will be referred to as an uncompelled change to slave state. An uncompelled change to slave state will occur during or some number of cycles after the issue cycle of a read request or the last cycle of the last request in a cluster. The number of cycles depends on the state of the cache, the presence of a secondary cache and the secondary cache parameters. After an uncompelled change to slave state the system interface will remain in slave state until the external agent issues an external request, after which the system interface will return to master state. An external agent must note that the processor has performed an uncompelled change to slave state and begin driving the address and data bus and the command bus As long as the system interface is in slave state, the external agent will begin an external request without arbitrating for the system interface, that is without asserting $\overline{\text{ExtRqst}}$.

4.6.3 System Interface Signal Descriptions

The system interface address and data bus is the SysAD bus. The system interface command bus is the SysCmd bus.

The SysAD bus and SysCmd bus valid signal that is asserted by the processor in master state is $\overline{\text{ValidOut}}$. The SysAD bus and SysCmd bus valid signal that is asserted by an external agent in slave state is $\overline{\text{ValidIn}}$.

The **SysADC** bus provides eight check bits for the SysAD bus. The nature of the check bits is programmable via the boot time mode control interface. The check bits may represent even byte parity for the contents of the SysAD bus, or they may be interpreted according to the code described in **CHAPTER 5 ERROR CHECKING AND CORRECTING (ECC)** to detect and correct single bit errors and detect double bit errors or three or four bit errors within a nibble on the SysAD bus. For a description of even parity, see the appendix on even parity. For further details on the ECC characteristics of the VR4000SC, see **CHAPTER 5 ERROR CHECKING AND CORRECTING (ECC)**.

The signal **SysCmdP** is an even parity bit over the nine bits of the SysCmd bus generated by the processor in master state. SysCmdP is not checked by the processor in slave state. For a description of even parity, see **APPENDIX B EVEN PARITY.**

System interface arbitration is implemented using the signals $\overline{\text{ExtRqst}}$ and $\overline{\text{Release}}$.

Processor request flow control is implemented using the signals $\overline{\text{RdRdy}}$ and $\overline{\text{WrRdy}}$.

4.6.4 System Interface Maintenance Signals

In addition to the signals used to implement the system interface request protocol, the system interface includes maintenance signals necessary for the operation of the processor. These include a master clock input for the processor, **MasterClock**, which must be connected to a continuous clock signal at the desired operation frequency of the processor; a processor synchronization clock output, **SyncOut** and a processor synchronization clock input, **SyncIn** that must be connected together to allow the processor internal clock synchronization logic to compensate for pad driver and receiver delays; a master clock output, **MasterOut**, aligned with MasterClock for use in clocking external logic that must run at MasterClock frequency; three reset related inputs, **VDDOk**, **ColdReset**, and **Reset**, an eight bit status bus, **Status(7:0)** that is encoded to indicate the current operation status of the processor; a system fault processor output, **Fault** that is the mismatch indication of the boundary comparators when the processor is running in master checker mode; two transmit clock outputs, **TClock(1:0)** and two receive clock outputs, **RClock(1:0)** at the programmed operation frequency of the system interface.

4.6.5 System Interface Signal Summary

SysAD(63:0):	(i/o)	A 64-bit bus used for address and data transmission between the processor and an external agent.
SysADC(7:0):	(i/o)	An 8-bit bus containing check bits for the SysAD bus.
SysCmd(8:0):	(i/o)	A 9-bit bus used for command and data identifier transmission between the processor and an external agent.
SysCmdP:	(i/o)	A single even parity bit over the SysCmd bus.
ValidIn:	(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.
ValidOut:	(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.
ExtRqst:	(i)	Signals that an external agent wishes to issue an external request.
Release:	(o)	Signals that the processor is releasing the system interface to slave state.
RdRdy:	(i)	Signals that an external agent is capable of accepting a processor read, invalidate, or update request in both no-secondary cache and secondary cache mode or a read followed by a potential update request in secondary cache mode.
WrRdy:	(i)	Signals that an external agent is capable of accepting a processor write request in both no-secondary cache and secondary cache mode.
TClock(1:0):	(o)	Two identical transmit clocks at the operation frequency of the system interface.
RClock(1:0):	(o)	Two identical receive clocks at the operation frequency of the system interface.
MasterClock:	(i)	Master clock input at the operation frequency of the processor.
MasterOut:	(i)	Master clock output aligned with MasterClock.
SyncOut:	(o)	Synchronization clock output.
SyncIn:	(i)	Synchronization clock input.
Status(7:0):	(o)	An 8-bit bus that indicates the current operation status of the processor.
VDDOk:	(i)	When asserted, this signal indicates to the VR4000SC that the +5 volt power supply has been above 4.75 volts for more than 100 milliseconds and will remain stable. The assertion of VDDOk will initiate the reading of the boot time mode control serial stream.
ColdReset:	(i)	This signal must be asserted for a power on reset or a cold reset. The clocks SClock, TClock, and RClock begin to cycle and are synchronized with the deassertion edge of ColdReset.
Reset:	(i)	This signal must be asserted for any reset sequence. It may be asserted synchronously or asynchronously for a cold reset, or synchronously to initiate a warm reset.
Fault:	(o)	Mismatch output of boundary comparators.

4.6.6 System Interface Request Descriptions

The following sections will illustrate the protocol of each processor and external request through text and detailed timing diagrams. The timing diagrams use abbreviations to show the contents of encoded busses during cycles in which they are defined. Following is a list of abbreviations used for each bus and their definitions.

Global:		
	Unsd –	Unused.
SysAD bus:		
	Addr –	Physical address.
	Data<n>–	Data element number n of a block of data.
SysCmd bus:		
	Cmd –	An unspecified system interface command.
	Read –	A read request command.
	RwWF –	A read with write forthcoming request command.
	Write –	A write request command.
	Null –	A null request command.
	SINull –	A system interface release null request command.
	SCNull –	A secondary cache release null request command.
	Ivd –	An invalidate request command.
	Upd –	An update request command.
	Ivtn –	An intervention request command.
	Snoop –	A snoop request command.
	NData –	A noncoherent data identifier for a data element other than the last data element.
	NEOD –	A noncoherent data identifier for the last data element.
	CData –	A coherent data identifier for a data element other than the last data element.
	CEOD –	A coherent data identifier for the last data element.

Two closely spaced wavy vertical lines in a timing diagram indicate a repetition of the current cycle. That is the cycle broken by the wavy lines may represent one or more identical cycles. This is referred to as a break in the timing diagram and is used to keep the timing diagrams concise and readable.

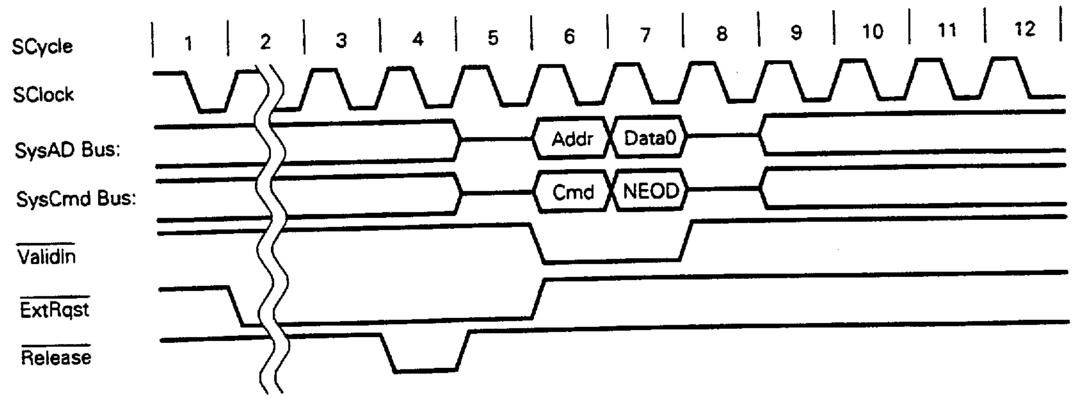
4.6.7 Arbitration Protocol

System interface arbitration is implemented using the signals $\overline{\text{ExRqst}}$ and $\overline{\text{Release}}$. When an external agent wishes to issue an external request, it will assert $\overline{\text{ExtRqst}}$. The processor will wait until it is ready to handle an external request and assert $\overline{\text{Release}}$ for one cycle before it tri-states the SysAD bus and SysCmd bus. The external agent will begin driving the SysAD bus and the SysCmd bus two cycles after a cycle in which $\overline{\text{Release}}$ is asserted. The external agent will always deassert $\overline{\text{ExtRqst}}$ two cycles after a cycle in which $\overline{\text{Release}}$ is asserted unless the external agent wishes to perform a subsequent external request. The external agent will always release the SysAD bus and the SysCmd bus at the completion of an external request.

The processor will assert $\overline{\text{Release}}$ for one cycle as a processor read request is issued or sometimes after a processor read request is issued to perform an uncompelled change to slave state. An external agent must begin driving the SysAD bus and the SysCmd bus two cycles after the cycle in which $\overline{\text{Release}}$ is asserted. After an uncompelled change to slave state, the processor will return to master state at the end of the next external request, which may be the read response, or may be some other external request.

The processor to system handshake for external requests is illustrated in **Fig. 4-1 Arbitration Protocol for External Requests.**

Fig. 4-1 Arbitration Protocol for External Requests


SCycle
1
2
3
4
5
6
7
8
9
10
11
12
SClock
SysAD Bus:
Addr
Data0
SysCmd Bus:
Cmd
NEOD
$\overline{\text{ValidIn}}$
$\overline{\text{ExtRqst}}$
$\overline{\text{Release}}$

4.6.8 Processor Read Request Protocol

A processor read request is issued with the system interface in master state by driving a read command on the SysCmd bus and a read address on the SysAD bus and asserting $\overline{\text{ValidOut}}$ for one cycle. Only one processor read request may be pending at a time. The processor must wait for and retire an external read response before initiating a subsequent read.

The processor will make an uncompelled change to slave state either at the issue cycle of the read request or sometime after the issue cycle of the read request by asserting the $\overline{\text{Release}}$ signal for one cycle. Once in slave state, an external agent may return the requested data via a read response. An external agent must not assert the signal $\overline{\text{ExtRqst}}$ for the purposes of returning a read response, but rather must wait for the uncompelled change to slave state. The signal $\overline{\text{ExtRqst}}$ may be asserted before or during a read response for the purposes of performing an external request other than a read response.

When a read is pending, $\overline{\text{ExtRqst}}$ is asserted, and $\overline{\text{Release}}$ is asserted for one cycle it may be unclear if this assertion of $\overline{\text{Release}}$ is in response to $\overline{\text{ExtRqst}}$, or represents an uncompelled change to slave state. The only situation in which this assertion of $\overline{\text{Release}}$ may not be considered an uncompelled change to slave state is if the system interface is operating in secondary cache mode, the read request was a read with write forthcoming read request, and the expected write request has not yet been issued by the processor. In this case, the write request must be accepted by the external agent before the read response can be issued. In all other cases, the assertion of $\overline{\text{Release}}$ may be considered to be an uncompelled change to slave state or to be in response to the assertion of $\overline{\text{ExtRqst}}$. In this situation, the processor will accept either a read response, or any other external request. If an external request other than a read response is issued, the processor will perform another uncompelled change to slave state after processing of the external request is complete.

The read response may either return the requested data, or an indication that the returned data is erroneous, if the requested data could not be successfully retrieved, which will cause the processor to take a bus error.

A processor read request and an uncompelled change to slave state occurring as the read request is issued is illustrated in **Fig. 4-2 Processor Read Request Protocol**. A processor read request and the subsequent uncompelled change to slave state occurring sometime after the read request is issued is illustrated in **Fig. 4-3 Processor Read Request Protocol, Change to Slave State Delayed**.

Fig. 4-2 Processor Read Request Protocol

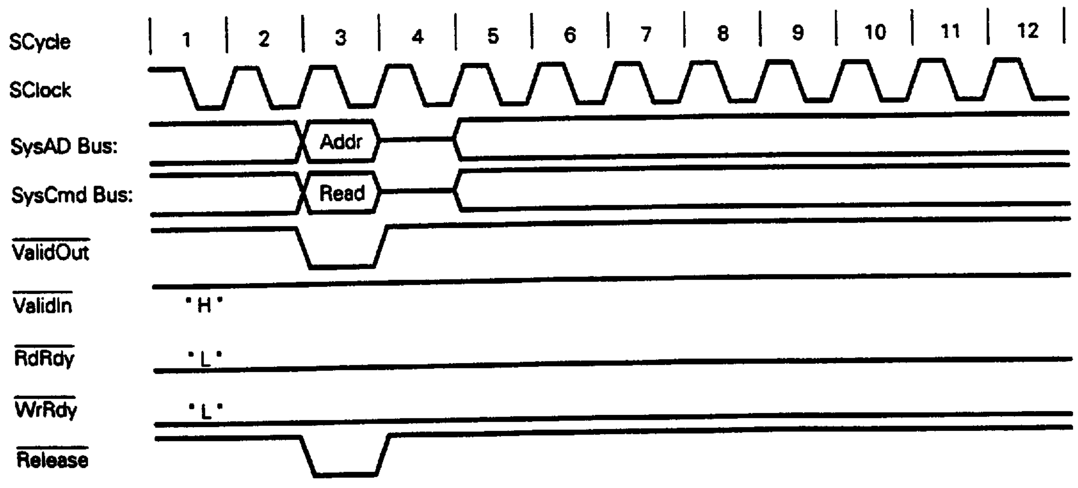
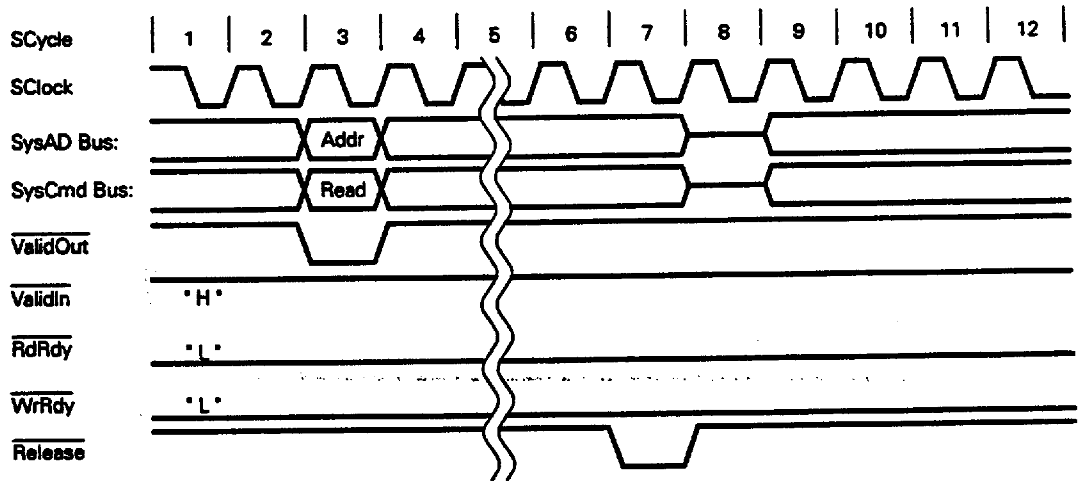


Fig. 4-3 Processor Read Request Protocol, Change to Slave State Delayed



4.6.9 Processor Write Request Protocol

Processor write requests are issued with one of two protocols. Double word, word, and partial word writes use a single word write request protocol. Write requests for a block of data use a block write request protocol. Processor write requests are issued with the system interface in master state.

A processor single word write request is issued by driving a write command on the SysCmd bus and a write address on the SysAD bus and asserting $\overline{\text{ValidOut}}$ for one cycle, followed by driving a data identifier on the SysCmd bus and data on the SysAD bus and asserting $\overline{\text{ValidOut}}$ for one cycle. The data identifier associated with the data cycle must contain a last data cycle indication.

A processor block write request is issued by driving a write command on the SysCmd bus and a write address on the SysAD bus and asserting $\overline{\text{ValidOut}}$ for one cycle, followed by driving a data identifier on the SysCmd bus and data on the SysAD bus and asserting $\overline{\text{ValidOut}}$ for a number of cycles sufficient to transmit the block of data. The data identifier associated with the last data cycle must contain a last data cycle indication. The first data cycle may not immediately follow the address cycle. A processor single word write request is illustrated in **Fig. 4-4 Processor**

Single Word Write Request Protocol. A processor block write request for eight words of data is illustrated in **Fig. 4-5 Processor Block Write Request Protocol (a)** and **Fig. 4-6 Processor Block Write Request Protocol (b)**.

Fig. 4-4 Processor Single Word Write Request Protocol

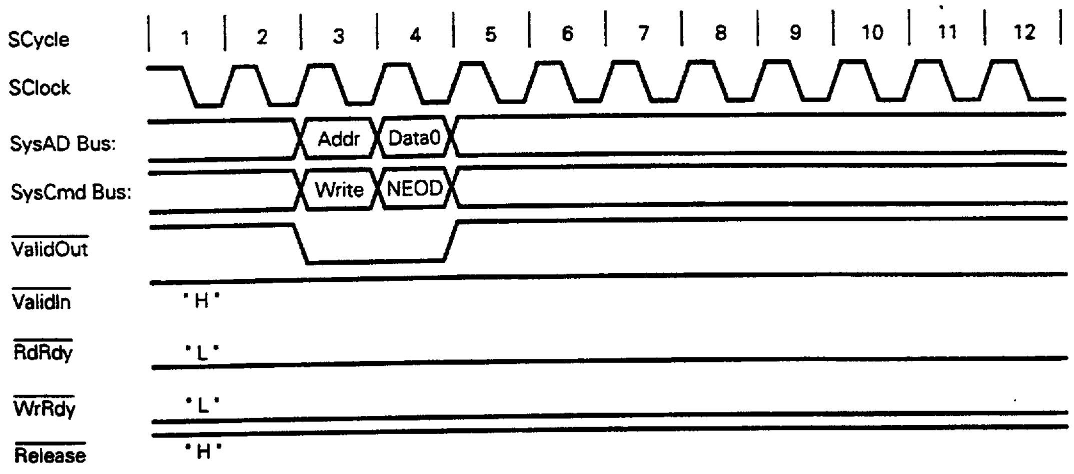


Fig. 4-5 Processor Block Write Request Protocol (a)

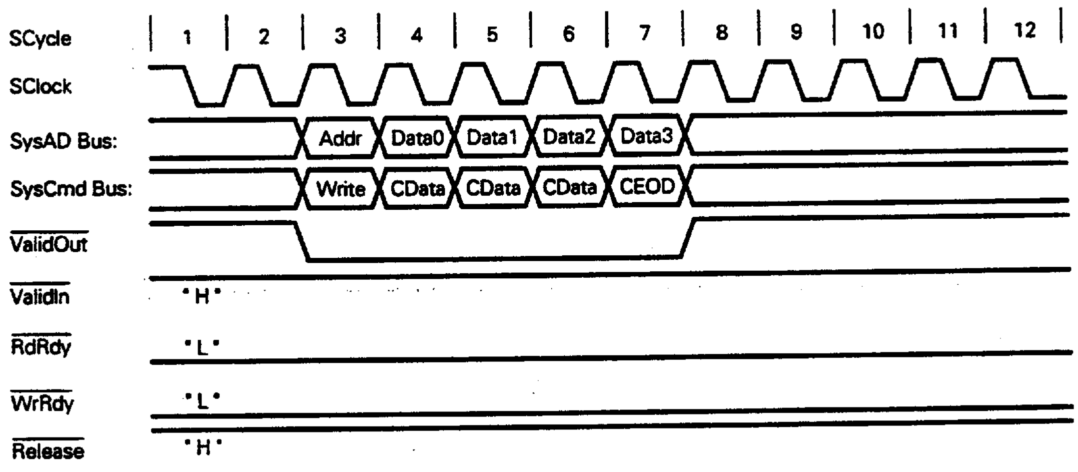
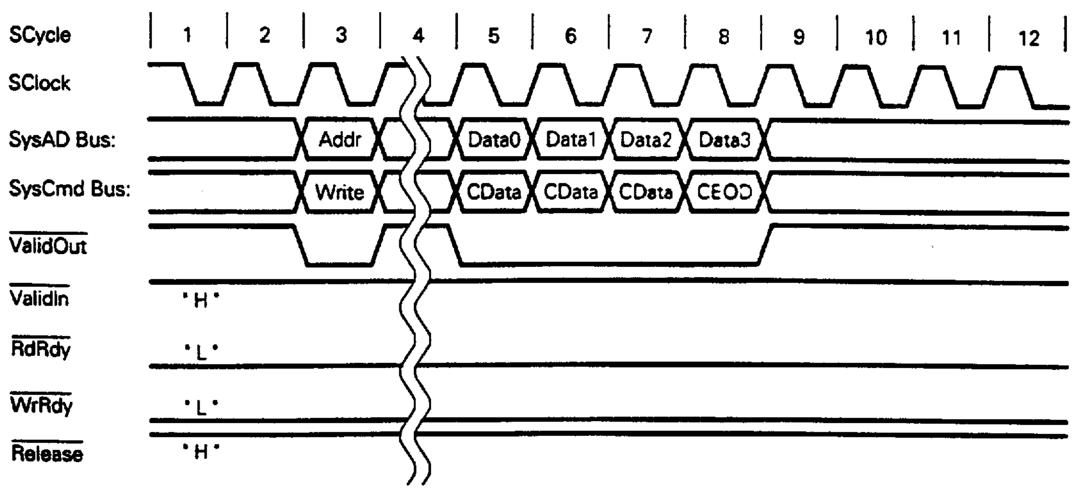


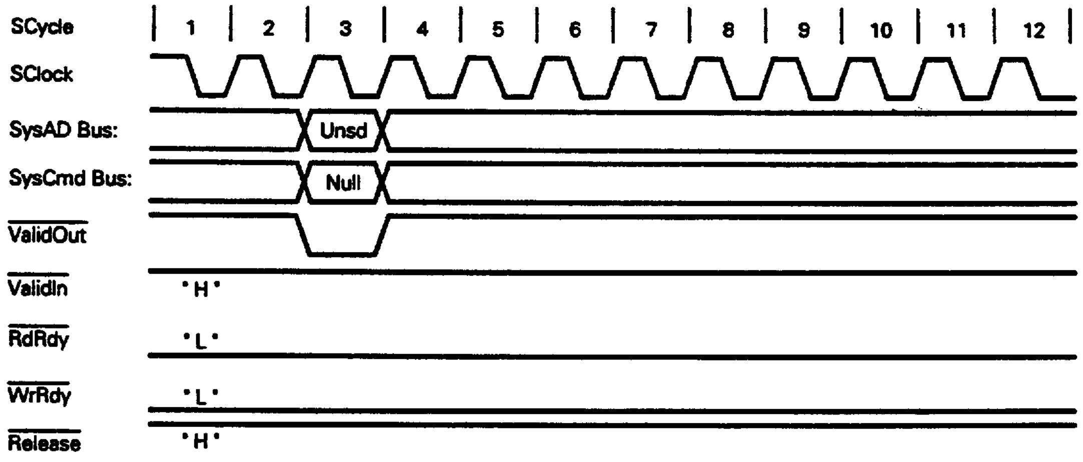
Fig. 4-6 Processor Block Write Request Protocol (b)



4.6.10 Processor Null Write Request Protocol

A processor null write request is issued with the system interface in master state by driving a null command on the SysCmd bus and asserting $\overline{\text{ValidOut}}$ for one cycle. The SysAD bus is unused during the address cycle associated with a null write request. Processor null write requests cannot be flow controlled with either $\overline{\text{RdRdy}}$ or $\overline{\text{WrRdy}}$, but rather always issue with a single address cycle. A processor null write request is illustrated in **Fig. 4-7 Processor Null Write Request Protocol**.

Fig. 4-7 Processor Null Write Request Protocol

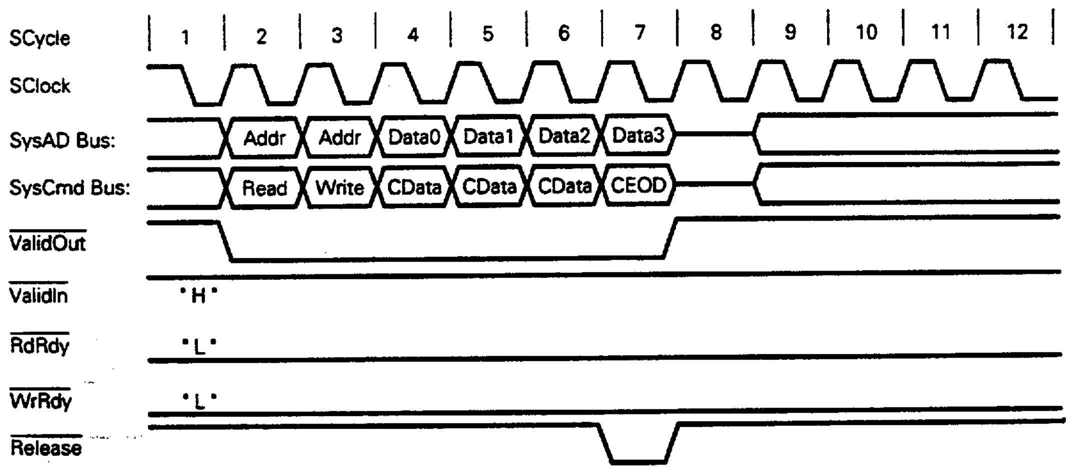


4.6.11 Processor Cluster Protocol

In secondary cache mode, the processor will issue requests both individually as in no-secondary cache mode and in groups that begin with a processor read request called **clusters**. A cluster consists of a processor read request followed by one or two additional processor requests issued while the read request is pending. All of the requests that are part of a cluster must be accepted before the response to the read request that begins the cluster may be returned to the processor. A cluster will include a processor read request followed by a write request.

The protocol of each of the requests that form a cluster is as described above. The number of unused cycles between the requests that form a cluster may be zero or greater. The processor will make an uncompelled change to slave state either during or following the last cycle of the last request in the cluster. A cluster consisting of a read request followed by a potential update request followed by a block write request for eight words of data with minimum spacing between the requests that form the cluster and an uncompelled change to slave state at the earliest opportunity is illustrated in **Fig. 4-8 Processor Cluster Protocol.**

Fig. 4-8 Processor Cluster Protocol



4.6.12 External Request Protocol

External requests may only be issued with the system interface in slave state. An external agent must assert ExtRqst to arbitrate for the system interface, and wait for the processor to release the system interface to slave state before issuing an external request. If the system interface is already in slave state, i.e. the processor has previously performed an uncompelled change to slave state, an external agent may begin an external request immediately.

After issuing an external request an external agent must return the system interface to master state. If the external agent does not have any additional external requests to perform, ExtRqst must be de-asserted two cycles after the cycle in which Release is asserted. An external agent may hold ExtRqst asserted if it needs to issue a string of external requests, but it must wait for the processor to assert Release and return the system interface to slave state before it may proceed with the next external request. For a string of external requests, the external agent must de-assert ExtRqst two cycles after the cycle in which Release is asserted for the last external request in the string. The processor will continue to handle external requests as long as ExtRqst is held asserted, however, the processor will not release the system interface to slave state for a subsequent external request until it has completed the current request. A string of external requests will not be interrupted by a processor request as long as ExtRqst is held asserted throughout the issue of the string of external requests.

(1) External Read Request Protocol

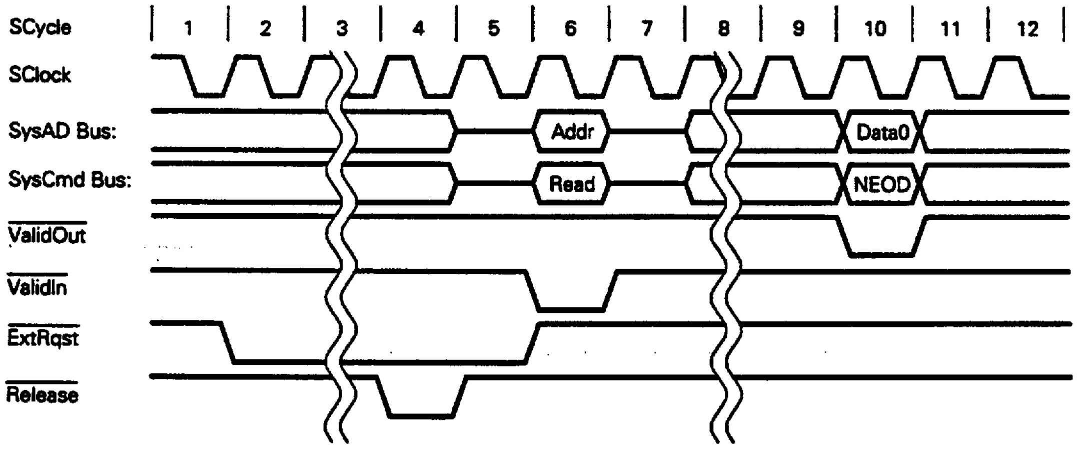
External read requests use a non-split protocol that does not allow any other request to occur at the system interface between the external read request and the read response. The protocol of an external read request encompasses the request from an external agent and the response from the processor.

An external read request consists of driving a read request command on the SysCmd bus and a read request address on the SysAD bus and asserting $\overline{\text{ValidIn}}$ for one cycle. After the address and command are sent, the external agent will release the SysCmd and SysAD busses and allow the processor to begin driving them. The processor will access the data that is the target of the read and return the data to the external agent. The processor accomplishes this by driving a data identifier on the SysCmd bus, the response data on the SysAD bus, and asserting $\overline{\text{ValidOut}}$ for one cycle. The data identifier will indicate that this is response data and contain a last data cycle indication. The processor will continue driving the SysCmd and SysAD busses after the read response is returned to transition the system interface back to master state.

External read requests are only allowed to read a word of data from the processor. The processor response to external read requests for any data element other than a word is undefined.

An external read request with the system interface initially in master state is illustrated in **Fig. 4-9 External Read Request, System Interface in Master State**.

Fig. 4-9 External Read Request, System Interface in Master State



Remark Version 1.2 of the VR4000SC does not contain any resources that are readable with an external read request. Version 1.2 of the VR4000SC will return a bus error response to any external read request.

(2) External Null Request Protocol

The VR4000SC processor supports two kinds of external null requests. A system interface release external null request is used to return the system interface to master state after it has been released to slave state without affecting the processor. A scache release external null request is used to return ownership of the secondary cache to the processor while the system interface remains in slave state for some period of time. This is important since any time the processor releases the system interface to slave state to accept an external request, it also acquires ownership of the secondary cache for use by the external request in anticipation of handling a coherence request. When an external agent requests ownership of the system interface for the purposes of using the SysAD bus for a transfer unrelated to the processor this ownership of the secondary cache will prevent the processor from satisfying subsequent primary cache misses. The scache release external request can be issued by the external agent to return ownership of the secondary cache to the processor. External null requests require no action from the processor other than to return the system interface to master state or to regain ownership of the secondary cache.

An external null request consists of driving a null request command on the SysCmd bus and asserting $\overline{\text{ValidIn}}$ for one cycle. The SysAD bus is unused during the address cycle associated with an external null request. After the address cycle is issued the null request is complete. For a system interface release external null request the external agent will release the SysCmd and SysAD busses and allow the system interface to return to master state. For a scache release external null request the system interface will remain in slave state. A scache release external null request with the system interface initially in master state is illustrated in **Fig. 4-10 Secondary Cache Release External Null Request**. A system interface release external null request with the system interface initially in slave state is illustrated in **Fig. 4-11 System Interface Release External Null Request**.

Fig. 4-10 Secondary Cache Release External Null Request

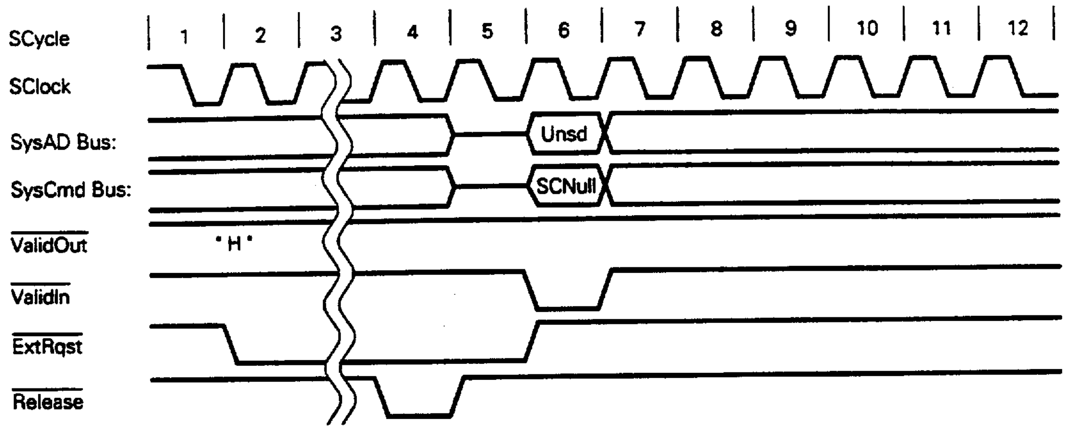
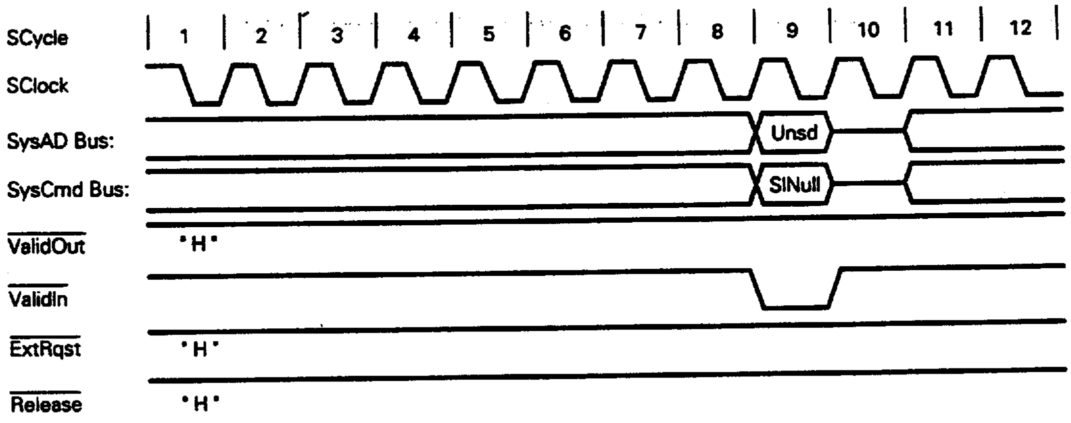


Fig. 4-11 System Interface Release External Null Request



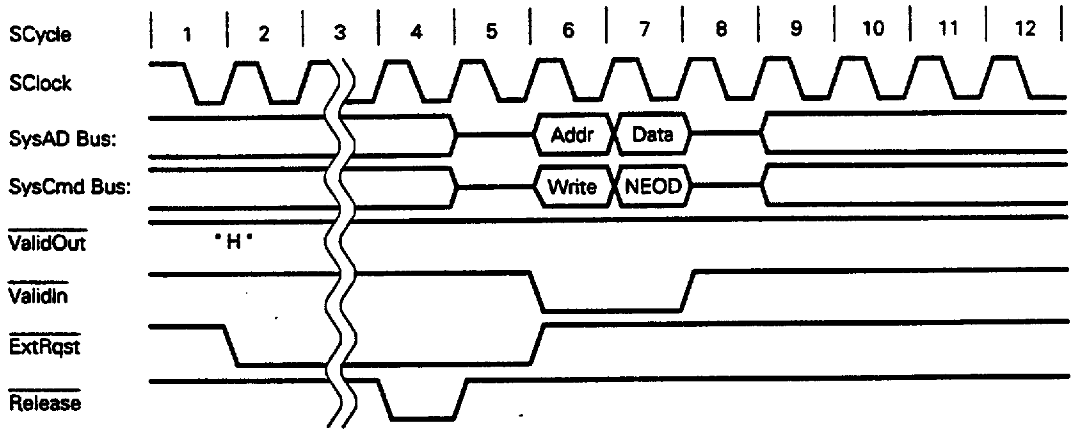
(3) External Write Request Protocol

External write requests use a protocol identical to the processor single word write protocol except that the signal $\overline{\text{ValidIn}}$ is asserted instead of the signal $\overline{\text{ValidOut}}$. An external write request consists of driving a write command on the SysCmd bus and a write address on the SysAD bus and asserting $\overline{\text{ValidIn}}$ for one cycle, followed by driving a data identifier on the SysCmd bus and data on the SysAD bus and asserting $\overline{\text{ValidIn}}$ for one cycle. The data identifier associated with the data cycle must contain a last data cycle indication. After the data cycle is issued the write request is complete and the external agent will release the SysCmd and SysAD busses and allow the system interface to return to master state.

External write requests are only allowed to write a word of data to the processor. The behavior of the processor in response to an external write request for any data element other than a word is undefined.

An external write request with the system interface initially in master state is illustrated in **Fig. 4-12 External Write Request**.

Fig. 4-12 External Write Request

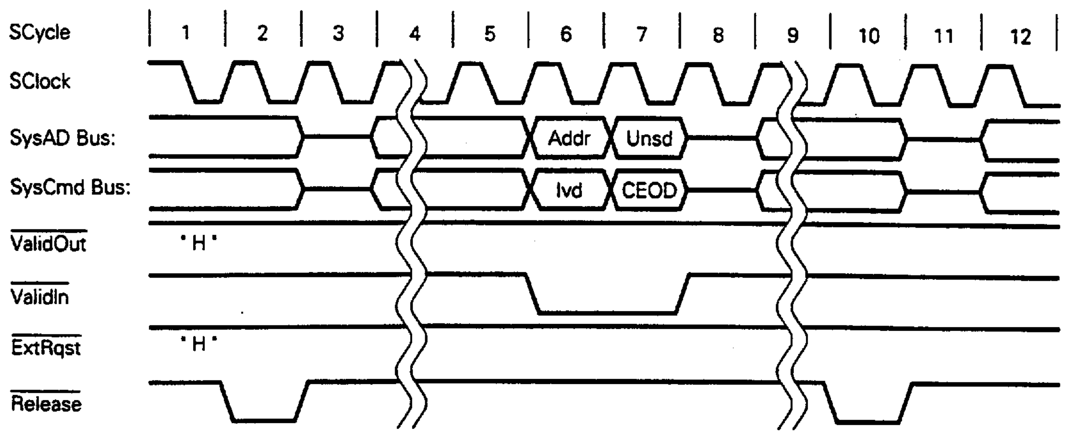


Remark The only writable resource in the version 1.2 of the VR4000SC is the processor interrupts.

(4) External Invalidate and Update Request Protocol

External invalidate and update requests use a protocol identical to that for external write requests. The data element provided with an update request may be a double word, word, or partial word. The single data cycle will be unused for an invalidate request. An external invalidate request following an uncompelled change to slave state is illustrated in **Fig. 4-13 External Invalidate Request Following an Uncompelled Change to Slave State.**

Fig. 4-13 External Invalidate Request Following an Uncompelled Change to Slave State



(5) Read Response Protocol

An external agent must return data to the processor in response to a processor read request by first waiting for the processor to perform an uncompelled change to slave state, 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 external agent will release the SysCmd and SysAD busses and allow the system interface to return to master state. Note that the processor will always perform an uncompelled change to slave state at some time after issuing a read request.

The data identifier for the data cycles must indicate that this is response data, and the data identifier associated with the last data cycle must contain a last data cycle indication. For read responses to coherent block read requests, each data identifier must include an indication of the cache state in which to load the response data. The cache state provided with each data identifier must be the same and must be either clean exclusive, dirty exclusive, shared, or dirty shared. The behavior of the processor if the cache state provided with the data identifiers is changed during the transfer of the block of data or if the cache state provided is invalid is undefined. 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; that is in response to a processor read request. The behavior of the processor if a read response is presented to it when there is no processor read pending is undefined. Further, if the processor issues a read with write forthcoming request, a processor write request or a processor null write request must be accepted before the read response may be returned. The behavior of the processor is undefined if the read response is returned before a processor write request is accepted.

A processor word read request followed by a word read response is illustrated in **Fig. 4-14 Process Word Read Request Followed by a Word Read Response**. A read response for a processor block read with the system interface already in slave state is illustrated in **Fig. 4-15 Block Read Response, System Interface Already in Slave State**.

Fig. 4-14 Processor Word Read Request Followed by a Word Read Response

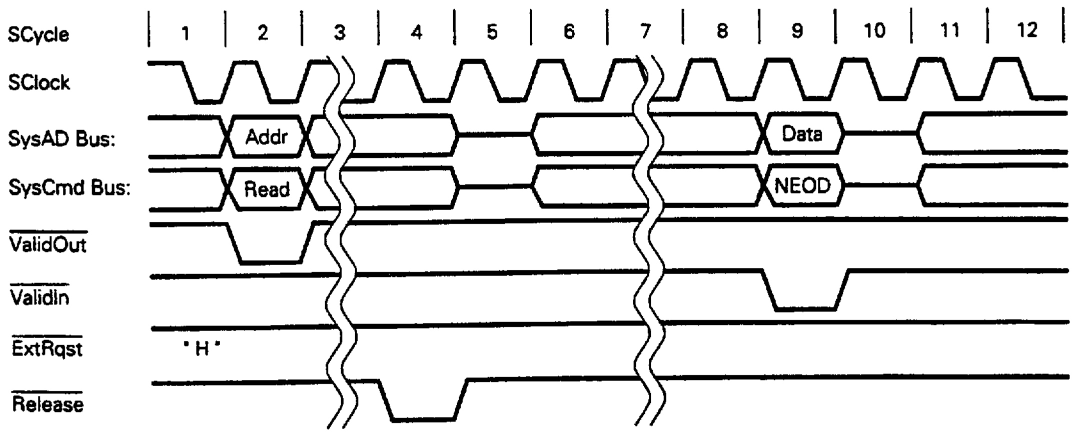
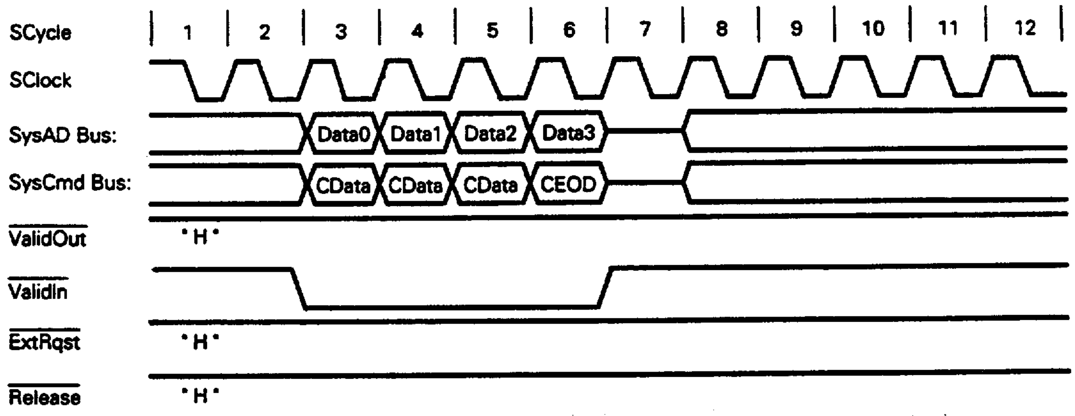


Fig. 4-15 Block Read Response, System Interface Already in Slave State



(6) External Intervention Request Protocol

External intervention requests use a protocol similar to that for external read requests except that a cache line size block of data may be returned along with an indication of the cache state for the cache line, depending on the state of the cache line and the value of the **data return** bit in the intervention request command.

The data return bit in the intervention request command may indicate return on dirty or return on exclusive. If the data return bit indicates return on dirty and the cache line that is the target of the intervention request is in the state dirty exclusive or dirty shared, the contents of the cache line will be returned in response to the intervention request. If the data return bit indicates return on exclusive and the cache line that is the target of the intervention request is in the state clean exclusive or dirty exclusive, the contents of the cache line will be returned in response to the intervention request. Otherwise, the response to the intervention request will not include the contents of the cache line but rather will simply indicate the state of the cache line that is the target of the intervention request. Note that if the cache line that is the target of the intervention request is not present in the cache at all, i.e. a tag comparison for the cache line at the target cache address fails, the cache line that is the target of the intervention request will be considered to be in the invalid state.

The processor will return an indication of the cache state in which a cache line was found but not its contents by driving a coherent data identifier that indicates the state of the cache line on the SysCmd bus, and asserting $\overline{\text{ValidOut}}$ for one cycle. The SysAD bus is unused during this data cycle. The data identifier will indicate that this is a response data cycle and will contain a last data cycle indication.

The processor will return the contents of a cache line along with an indication of the cache state in which it was found by issuing a sequence of data cycles sufficient to transmit the contents of the cache line. The data identifier transmitted with each data cycle will indicate the cache state in which the cache line was found and that this is response data. The data identifier associated with the last data cycle will contain a last data cycle indication.

If the contents of a cache line is returned in response to an intervention request, it will be returned in sub-block order starting with the double word at the address supplied with the intervention request. For further details on sub-block ordering see **APPENDIX A SUB-BLOCK ORDERING**. Note, however, that if the intervention address targets the double word at the beginning of the block sub-block ordering is equivalent to sequential ordering.

An external intervention request to a cache line found in the shared state with the system interface initially in master state is illustrated in **Fig. 4-16 External Intervention Request, Shared Line, System Interface in Master State**. An external intervention request to a cache line found in the dirty exclusive state with the system interface initially in slave state is illustrated in **Fig. 4-17 External Intervention Request, Dirty Exclusive Line, System Interface in Slave State**.

Fig. 4-16 External Intervention Request, Shared Line, System Interface in Master State

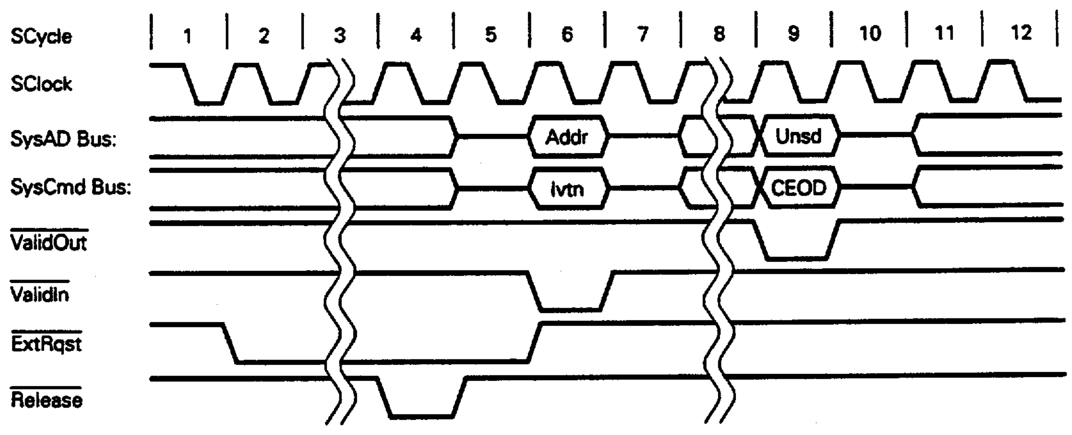
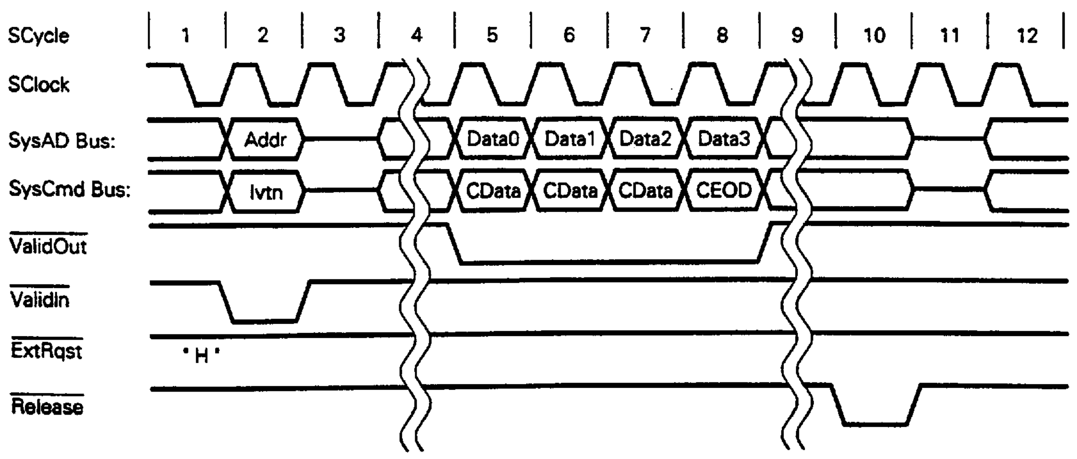


Fig. 4-17 External Intervention Request, Dirty Exclusive Line, System Interface in Slave State



4.6.13 External Snoop Request Protocol

External snoop requests use a protocol identical to that for external read requests, except that the processor will respond to a snoop request with an indication of the current cache state for the cache line that is the target of the snoop request instead of data. The processor accomplishes this by driving a coherent data identifier on the SysCmd bus, and asserting ValidOut for one cycle. The SysAD bus is unused during the snoop response. The processor will continue driving the SysCmd and SysAD busses after the snoop response is returned to transition the system interface back to master state.

Note that if the cache line that is the target of the snoop request is not present in the cache at all, i.e. a tag comparison for the cache line at the target cache address fails, the cache line that is the target of the snoop request will be considered to be in the invalid state.

An external snoop request issued with the system interface in master state is illustrated in **Fig. 4-18 External Snoop Request, System Interface in Master State.** An external snoop request issued with the system interface in slave state is illustrated in **Fig. 4-19 External Snoop Request, System Interface in Slave State.**

Fig. 4-18 External Snoop Request, System Interface in Master State

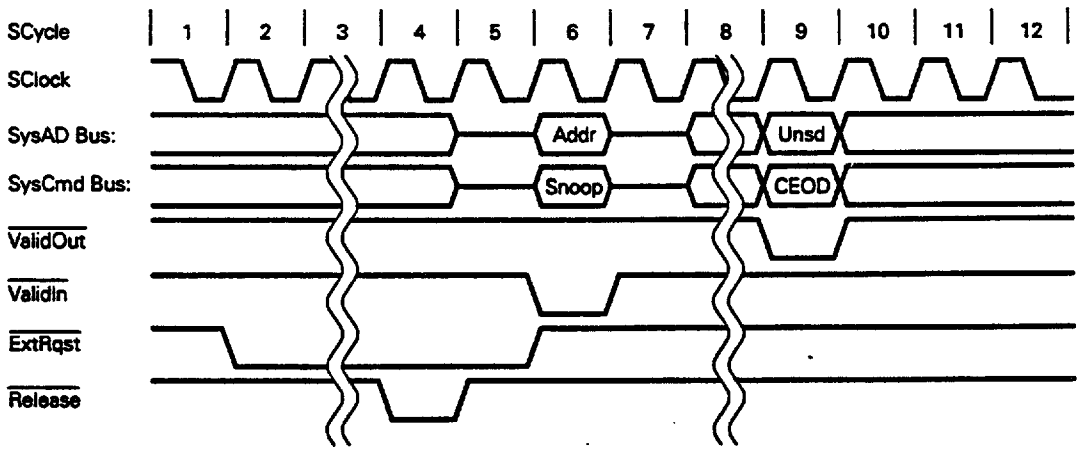
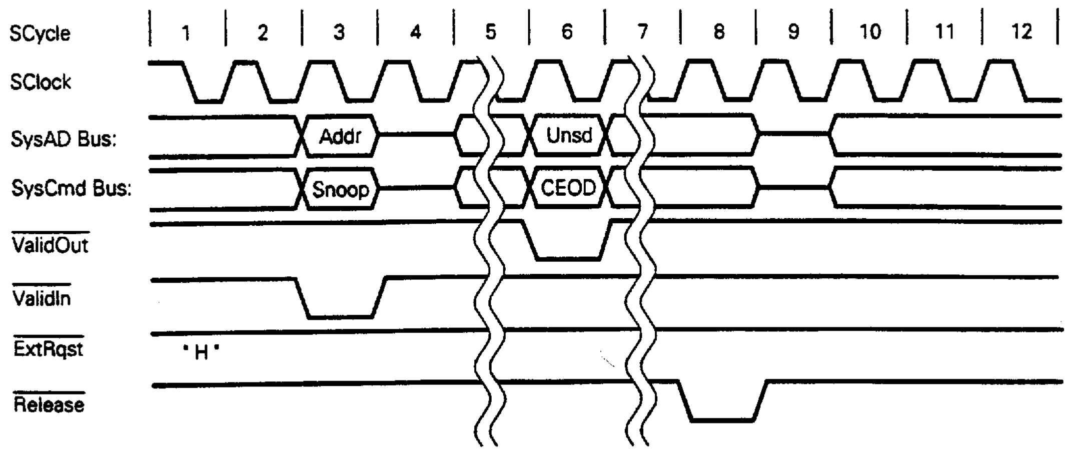


Fig. 4-19 External Snoop Request, System Interface in Slave State



4.6.14 Processor Request and Cluster Flow Control

The signal RdRdy may be used by an external agent to control the flow of a processor read request. The processor samples the signal RdRdy to determine if the external agent is currently capable of accepting a read request. The signal WrRdy controls the flow of a processor write request. The processor will not complete the issue of a read request until it issues an address cycle for the request such that the signal RdRdy was asserted two cycles previously. The processor will not complete the issue of a write request until it issues an address cycle for the write request such that the signal WrRdy was asserted two cycles previously.

Two processor write requests in which the issue of the second is delayed for the assertion of WrRdy are illustrated in **Fig. 4-20 Two Processor Write Requests, Second Write Delayed for the Assertion of WrRdy**. A processor cluster in which the issue of the read write request is delayed for the assertion of RdRdy is illustrated in **Fig. 4-21 Processor Read Request within a Cluster Delayed for the Assertion of RdRdy**. A processor cluster in which the issue of the write request is delayed for the assertion of WrRdy is illustrated in **Fig. 4-22 Processor Write Request within a Cluster Delayed for the Assertion of WrRdy**. The issue of a processor write request delayed for the assertion of WrRdy and the completion of an external invalidate request is illustrated in **Fig. 4-23 Processor Write Request Delayed for the Assertion of WrRdy and the Completion of an External Invalidate Request**.

Fig. 4-20 Two Processor Write Requests, Second Write Delayed for the Assertion of $\overline{\text{WrRdy}}$

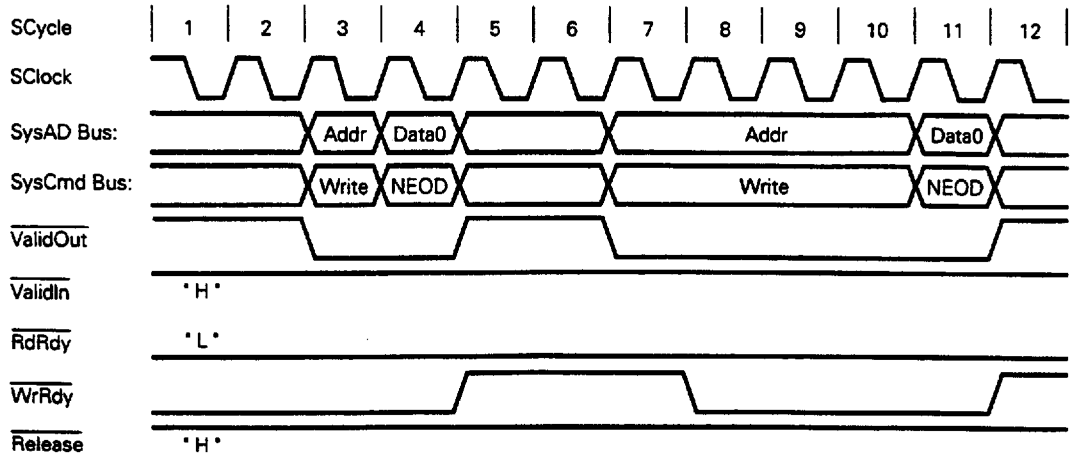


Fig. 4-21 Processor Read Request Within a Cluster Delayed for the Assertion of $\overline{\text{RdRdy}}$

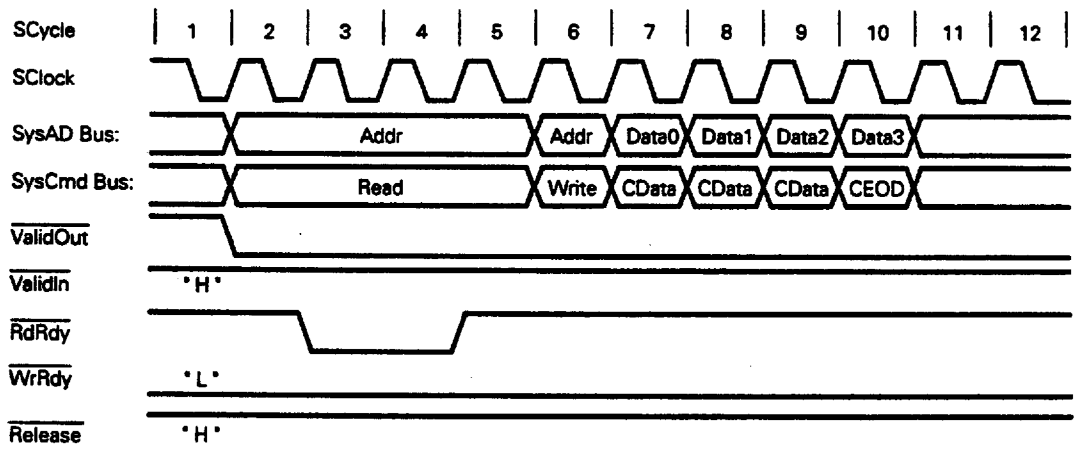


Fig. 4-22 Write Request Within a Cluster Delayed for the Assertion of WrRdy

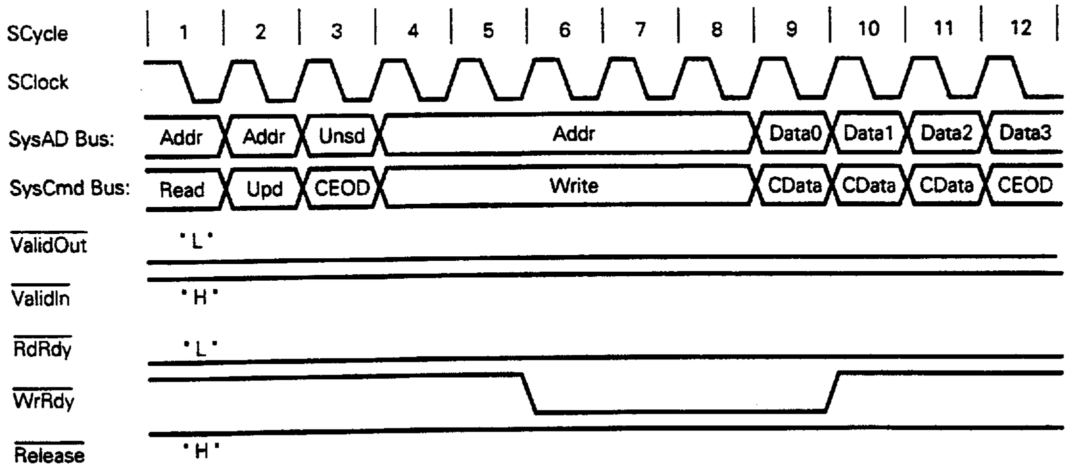
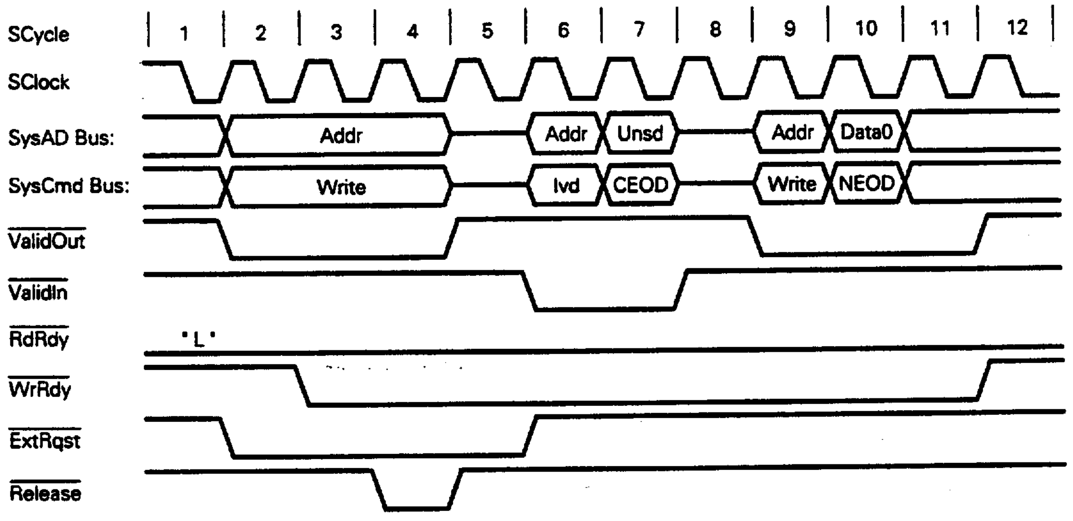


Fig. 4-23 Processor Write Request Delayed for the Assertion of WrRdy and the Completion of an External Invalidate Request



4.6.15 Data Rate Control

The system interface supports a maximum data rate of one double word per cycle. The maximum data rate the processor can support is directly related to the secondary cache access time, if the access time is too long, the processor will not be able to transmit and accept data at the maximum rate.

The rate at which data is delivered to the processor may be chosen by an external agent by driving data and asserting $\overline{\text{ValidIn}}$ every n cycles instead of every cycle. The processor will only interpret cycles during which $\overline{\text{ValidIn}}$ 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 at any rate it chooses but must not deliver data to the processor faster than it is capable of accepting it.

Because the secondary cache is organized as a 128 bit RAM array, the processor will operate most efficiently if data is delivered to it in pairs of double words. It is most efficient to reduce the data rate by delivering a pair of double words to the processor, followed by some number of unused cycles, followed by another pair of double words. The pattern should be chosen to repeat at a rate determined by the secondary cache write cycle time. However, the processor will accept data in any pattern as long as the time between the transfer of any pair of odd numbered double words is greater than or equal to the write cycle time of the secondary cache. Double words in the transfer pattern are numbered beginning at zero such that the odd numbered words are the second, fourth, sixth, and so on words transferred.

The maximum processor data rate for each of the possible secondary cache write cycle times and the most efficient data pattern for each data rate is illustrated in **Table 4-1 Maximum Processor Data Rates**. In this and subsequent tables data patterns are specified using the letters "D" and "x". "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 data rate. For example, a data pattern specified by the sequence of letters "DDxx", to achieve a data rate of two words every four cycles, is a data pattern in which two data cycles are followed by two unused cycles followed by two data cycles and two unused cycles, and so on. A read response in which data is provided to the processor at a rate of two words every three cycles using the data pattern "DDx" is shown in **Fig. 4-24 Read Response, Reduced Data Rate, System Interface in Slave State.**

If data is delivered to the processor at a rate that exceeds the maximum the processor can support, based on the secondary cache write cycle time, the behavior of the processor is undefined The secondary cache write cycle time is the sum of the parameters TWr1Dly, TWrSUp, and TWrRc described in the section on secondary cache write cycles. The rate at which the processor transmits data is programmable at boot time via the boot time mode control interface. The transmit data rate may be programmed to any of the data rates and data patterns listed in **Table 4-2 Transmit Data Rates**, as long as the programmed data rate does not exceed the maximum the processor can support, based on the secondary cache access time. If a transmit data rate is programmed that exceeds the maximum the processor can support, the behavior of the processor is undefined. A processor write request for which the processor transmit data rate has been programmed to 1 double word every two cycles using the data pattern "DDxx" is shown in **Fig. 4-25 Processor Write Request, Transmit Data Rate Reduced.**

Table 4-1 Maximum Processor Data Rates

SCache Write Cycle Time	Max Data Rate	Best Data Pattern
<= 4 PCycles	1 Double/1 Cycle	D
5-6 PCycles	2 Doubles/3 Cycles	DDx
7-8 PCycles	1 Double/2 Cycles	DDxx
9-10 PCycles	2 Doubles/5 Cycles	DDxxx
11-12 PCycles	1 Double/3 Cycles	DDxxxx

Fig. 4-24 Read Response, Reduced Data Rate, System Interface in Slave State

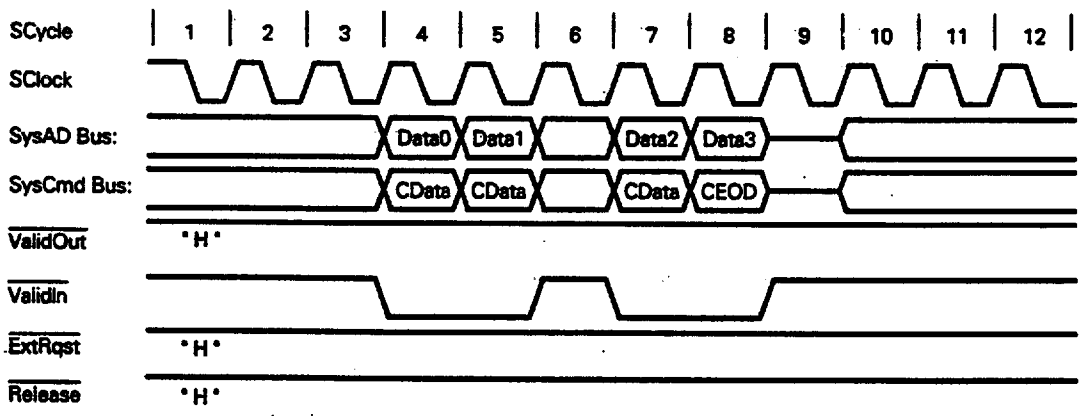
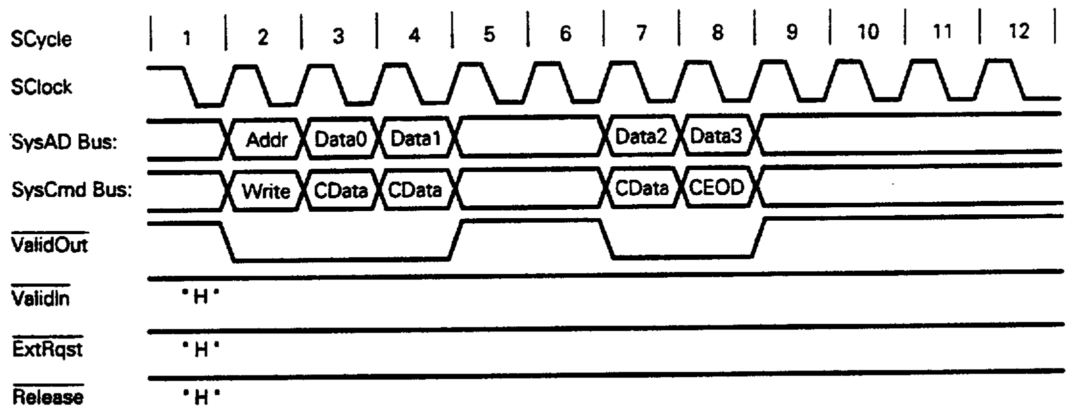


Table 4-2 Transmit Data Rates

Data Rate	Data Pattern	Max SCache Access
1 Double/1 Cycle	D	4 PCycles
2 Doubles/3 Cycles	DDx	6 PCycles
1 Double/2 Cycles	DDxx	8 PCycles
1 Double/2 Cycles	DxDx	8 PCycles
2 Doubles/5 Cycles	DDxxx	10 PCycles
1 Double/3 Cycles	DDxxxx	12 PCycles
1 Double/3 Cycles	DxxDxx	12 PCycles
1 Double/4 Cycles	DDxxxxxx	16 PCycles
1 Double/4 Cycles	DxxxDxxx	16 PCycles

Fig. 4-25 Processor Write Request, Transmit Data Rate Reduced



4.6.16 Multiple Drivers on the SysAD Bus

In most VR4000SC 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 must coordinate the usage of the SysAD bus using the arbitration handshake signals and the external null requests.

To implement an independent transmission on the SysAD bus that does not involve the processor, the external agent will request the SysAD bus to issue an external request. After the processor releases the system interface to slave state, the external agent may issue a scache release external null request to return ownership of the secondary cache to the processor, if the processor is being used with a secondary cache. The external agent may then allow the independent transmission to take place on the SysAD bus making sure that $\overline{\text{ValidIn}}$ is not asserted while the transmission is occurring. When the transmission is complete, the external agent will issue a system interface release external null request to return the system interface to master state.

4.7 Cycle Counts for System Interface Interactions

The VR4000SC processor specifies minimum and maximum cycle counts for various processor transactions and for the processor's response time to external requests to facilitate system design with the VR4000SC. Processor requests themselves are constrained by the system interface request protocol and the cycle counts for such requests can be determined by examining the protocol. The spacing between requests within a cluster, the waiting period for the processor to release the system interface to slave state in response to an external request, and the response time for an external request that requires a response is variable and subject to minimum and maximum cycle counts. The remainder of this section will describe and tabulate the minimum and maximum cycle counts for these system interface interactions.

The minimum and maximum number of unused cycles between the requests within a cluster is a function of processor internal activity. The minimum number of unused cycles separating requests within a cluster is zero, the requests may be adjacent. The maximum number of unused cycles separating requests within a cluster varies depending on the requests that form the cluster. The minimum and maximum number of unused cycles separating requests within a cluster is summarized in **Table 4-3 Unused Cycles Separating Requests within a Cluster**.

Table 4-3 Unused Cycles Separating Requests within a Cluster

From Processor Request	To Processor Request	Minimum Unused Cycles	Maximum Unused Cycles
Read	Invalidate or Update	0	2
Read	Write	0	2

The number of cycles the processor may wait to release the system interface to slave state for an external request will be referred to as the **release latency**. The release latency is a function of processor internal activity and processor request activity. The processor will release the system interface to accept an external request under the conditions described above. When no processor requests are in progress internal activity, such as refilling the primary cache from the secondary cache, may cause the processor to wait some number of cycles before releasing the system interface. Release latency will be considered in three categories:

(1) release latency when the external request signal is asserted during the cycle two cycles before the last cycle of a processor request or two cycles before the last cycle of the last request in a cluster.

(2) release latency when the external request signal is not asserted during a processor request or cluster, or asserts during the last cycle of a processor request or cluster.

(3) release latency when the processor does an uncompelled change to slave state.

The minimum and maximum release latency for requests that fall into categories (1), (2) and (3) above is summarized in **Table 4-4 Release Latency for Category (1), (2) and (3) External Requests**.

Table 4-4 Release Latency for Category (1), (2) and (3) External Requests

Category	Minimum Note	Maximum Note
(1)	4	6
(2)	4	24
(3)	0	TBD

Note These cycle counts are preliminary and subject to change.

The number of cycles the processor may take to respond to an external request that requires a response, that is, an external intervention request, read request, or snoop request, will be referred to as the **intervention response latency, external read response latency,** or **snoop response latency** respectively. The number of cycles of latency is the number of unused cycles between the address cycle of the request and the first data cycle of the response. Intervention response latency and snoop response latency is a function of processor internal activity and secondary cache access time. The minimum and maximum intervention response latency and snoop response latency as a function of secondary cache access time is summarized in **Table 4-5 Intervention Response Latency and Snoop Response Latency**. External read response latency is purely a function of processor internal activity. The minimum and maximum external read response latency is summarized in **Table 4-6 External Read Response Latency**.

Table 4-5 Intervention Response Latency and Snoop Response Latency

Max SCache Access	Intervention response latency Note		Snoop response latency Note	
	Min	Max	Min	Max
<= 4 PCycles	6	26	6	26
5-6 PCycles	8	28	8	28
7-8 PCycles	10	30	10	30
9-10 PCycles	12	32	12	32
11-12 PCycles	14	34	14	34

Note These cycle counts are preliminary and subject to change.

Table 4-6 External Read Response Latency

	Min Note	Max Note
External Read Response Latency	4	4

Note These cycle counts are preliminary and subject to change.

4.8 System Interface Syntax

System interface commands specify the precise nature and attributes of any system interface request during the address cycle for the request. System interface data identifiers specify the attributes of a data element transmitted during a system interface data cycle. The following sections describe the syntax, that is the bitwise encoding, of system interface commands and data identifiers.

4.8.1 System Interface Command and Data Identifier Syntax

System interface commands and data identifiers are encoded in nine 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. Bit eight of the SysCmd bus determines whether the current contents of the SysCmd bus is a command or a data identifier and therefore whether the current cycle is an address cycle or a data cycle. For system interface commands SysCmd(8) must be asserted (0). For system interface data identifiers SysCmd(8) must be de-asserted (1).

For system interface commands and data identifiers associated with external requests, reserved bits and reserved fields in the command or data identifier should be de-asserted, that is set to one (1) or all ones respectively. For system interface commands and data identifiers associated with processor requests, reserved bits and reserved fields in the command or data identifier are undefined.

(1) System Interface Command Syntax

This section will define the encoding of the SysCmd bus for system interface commands. A common encoding is used for all system interface commands. SysCmd(8) must be asserted (0) for all system interface commands. For all system interface commands SysCmd(7:5) specify the system interface request type which may be read, write, null, invalidate, update, intervention, or snoop. The encoding of SysCmd(7:5) for system interface commands is illustrated in **Table 4-7 Encoding of SysCmd(7:5) for System Interface Commands** below.

Table 4-7 Encoding of SysCmd(7:5) for System Interface Commands

SysCmd(7:5)	Command
0	Read Request.
1	Read Request, Write Request forthcoming.
2	Write Request.
3	Null Request.
4	Invalidate Request.
5	Update Request.
6	Intervention Request.
7	Snoop Request.

For read requests, the remainder of the SysCmd bus specifies the attributes of the read. SysCmd(4:3) encode block, coherency, and exclusivity attributes for the read. A read request with a write request forthcoming cannot be a double word, word, or partial word read. For both coherent and noncoherent block reads SysCmd(2) specifies whether the address of the cache line being replaced by this read request is being retained in the link address register and SysCmd(1:0) encode the block size for the read. For double word, word, or partial word reads SysCmd(2:0) encode the size of the read data in bytes. The encoding of SysCmd(4:3) for read commands is shown in **Table 4-8 Encoding of SysCmd(4:3) for Read Requests** below. The encoding of SysCmd(2:0) for block reads, or double word, word, or partial word reads is shown in **Table 4-9 Encoding of SysCmd(2:0) for Block Read Requests**, and **Table 4-10 Encoding of SysCmd(2:0) for Double Word, Word, or Partial Word Read Requests** respectively.

Table 4-8 Encoding of SysCmd(4:3) for Read Requests

SysCmd(4:3)	Read attributes.
0	Coherent block read.
1	Coherent block read, exclusivity requested.
2	Noncoherent block read.
3	Double word, single word, or partial word read.

Table 4-9 Encoding of SysCmd(2:0) for Block Read Requests

SysCmd(2)	Link address retained indication.
0	Address not retained.
1	Link address retained.
SysCmd(1:0)	**Read block size.**
0	Four words.
1	Eight words.
2	Sixteen words.
3	Thirty-two words.

Table 4-10 Encoding of SysCmd(2:0) for Double Word, Word, or Partial Word Read Requests

SysCmd(2:0)	Read data size.
0	One byte valid. (Byte).
1	Two bytes valid. (Half Word).
2	Three bytes valid (Tri-Byte).
3	Four bytes valid. (Word).
4	Five bytes valid. (Quinti-Byte).
5	Six bytes valid. (Sexti-Byte).
6	Seven bytes valid. (Septi-Byte).
7	Eight bytes valid. (Double Word).

For write requests, the remainder of the SysCmd bus specifies the attributes of the write. SysCmd(4:3) encode block attributes for the write. For block writes SysCmd(2) specifies whether the cache line associated with the write request will be replaced or retained after the write is completed and SysCmd(1:0) encode the block size for the write. For double word, word, or partial word writes SysCmd(2:0) encode the size of the write data in bytes. The encoding of SysCmd(4:3) for write commands is shown in **Table 4-11 Encoding of SysCmd(4:3) for Write Requests** below. The encoding of SysCmd(2:0) for block writes or double word, word, or partial word writes is shown in tables **Table 4-12 Encoding of SysCmd(2:0) for Block Write Requests** and **Table 4-13 Encoding of SysCmd(2:0) for Double Word, Word, or Partial Word Write Requests** respectively.

Table 4-11 Encoding of SysCmd(4:3) for Write Requests

SysCmd(4:3)	Write attributes.
0	Reserved.
1	Reserved.
2	Block write.
3	Double word, single word, or partial word write.

Table 4-12 Encoding of SysCmd(2:0) for Block Write Requests

SysCmd(2)	Cache line replacement attributes.
0	Cache line replaced.
1	Cache line retained.
SysCmd(1:0)	**Write block size.**
0	Four words.
1	Eight words.
2	Sixteen words.
3	Thirty-two words.

Table 4-13 Encoding of SysCmd(2:0) for Double Word, Word, or Partial Word Write Requests

SysCmd(2:0)	Write data size.
0	One byte valid. (Byte).
1	Two bytes valid. (Half Word).
2	Three bytes valid. (Tri-Byte).
3	Four bytes valid. (Word).
4	Five bytes valid. (Quinti-Byte).
5	Six bytes valid. (Sexti-Byte).
6	Seven bytes valid. (Septi-Byte).
7	Eight bytes valid. (Double Word).

Processor null write requests, system interface release external null requests, and scache release external null requests all use the null request command. For processor null requests, SysCmd(4:3) specifies that this is a null write request. For external null requests, SysCmd(4:3) specifies whether this is a system interface release null request or a scache release null request. The encoding of SysCmd(4:3) for processor null requests is shown in **Table 4-14 Encoding of SysCmd(4:3) for Processor Null Requests**. The encoding of SysCmd(4:3) for external null requests is shown in **Table 4-15 Encoding of SysCmd(4:3) for External Null Requests**.

Table 4-14 Encoding of SysCmd(4:3) for Processor Null Requests

SysCmd(4:3)	Null attributes.
0	Null write.
1	Reserved.
2	Reserved.
3	Reserved.

Table 4-15 Encoding of SysCmd(4:3) for External Null Requests

SysCmd(4:3)	Null attributes.
0	System interface release.
1	Scache release.
2	Reserved.
3	Reserved.

The encoding of SysCmd(4:0) for external invalidate and update requests is shown in **Table 4-16 Encoding of SysCmd(4:0) for External Invalidate or Update Requests**.

Table 4-16 Encoding of SysCmd(4:0) for External Invalidate or Update Requests

SysCmd(4)	**Reserved.**
SysCmd(3)	**Update cache state change attributes.**
0	Cache state changed to Shared.
1	No change to cache state.
SysCmd(2:0)	**Update data size.**
0	One byte valid. (Byte).
1	Two bytes valid. (Half Word).
2	Three bytes valid. (Tri-Byte).
3	Four bytes valid. (Word).
4	Five bytes valid. (Quinti-Byte).
5	Six bytes valid. (Sexti-Byte)
6	Seven bytes valid. (Septi-Byte).
7	Eight bytes valid. (Double Word).

SysCmd(3) is the data response on dirty bit for intervention requests or reserved for snoop requests. If the data response on dirty bit is asserted the processor will return the contents of the cache line in response to an intervention request if the line is found in state dirty exclusive or dirty shared. If the data response on dirty bit is de-asserted the processor will return the contents of the cache line in response to an intervention request if the line is found in state clean exclusive or dirty exclusive. For both snoop and intervention requests, SysCmd(2:0) specify a cache state change function to be applied to the cache line atomically with respect to the intervention or snoop response.
The encoding of SysCmd(4:0) for intervention requests is shown in **Table 4-17 Encoding of SysCmd(4:0) for Intervention Requests**. The encoding of SysCmd(4:0) for snoop requests is shown in **Table 4-18 Encoding of SysCmd(4:0) for Snoop Requests**.

Table 4-17 Encoding of SysCmd(4:0) for Intervention Requests

SysCmd(4)	**Reserved.**
SysCmd(3)	**Data response on dirty bit.**
0	Return cache line data if in state dirty exclusive or dirty shared.
1	Return cache line data if in state clean exclusive or dirty exclusive.
SysCmd(2:0)	**Cache state change function.**
0	No change to cache state.
1	If cache state clean exclusive, change to shared, otherwise no change to cache state.
2	If cache state clean exclusive or shared, change to invalid, otherwise no change to cache state.
3	If cache state clean exclusive, change to shared or if cache state dirty exclusive, change to dirty shared, otherwise no change to cache state.
4	If cache state clean exclusive, dirty exclusive, or dirty shared, change to shared, otherwise no change to cache state.
5	Change to invalid regardless of current cache state.
6	Reserved.
7	Reserved.

Table 4-18 Encoding of SysCmd(4:0) for Snoop Requests

SysCmd(4)	**Reserved.**
SysCmd(3)	**Reserved.**
SysCmd(2:0)	**Cache state change function.**
0	No change to cache state.
1	If cache state clean exclusive, change to shared, otherwise no change to cache state.
2	If cache state clean exclusive or shared, change to invalid, otherwise no change to cache state.
3	If cache state clean exclusive, change to shared or if cache state dirty exclusive, change to dirty shared, otherwise no change to cache state.
4	If cache state clean exclusive, dirty exclusive, or dirty shared, change to shared, otherwise no change to cache state.
5	Change to invalid regardless of current cache state.
6	Reserved.
7	Reserved.

(2) System Interface Data Identifier Syntax

This section will define the encoding of the SysCmd bus for system interface data identifiers. A common encoding is used for all system interface data identifiers. SysCmd(8) must be de-asserted (1) for all system interface data identifiers. System interface data identifiers have two formats, one for coherent data and a second for noncoherent data. Data associated with processor block write requests and processor double word, word, or partial word write requests is noncoherent. Data associated with processor update requests is noncoherent. Data returned in response to a processor coherent block read request is coherent while data returned in response to a processor noncoherent block read request or a processor double word, word, or partial word read request is noncoherent. Data associated with external update requests is noncoherent. Data associated with external write requests is noncoherent. Data returned in response to an external read request is noncoherent. Data returned in response to an external intervention request is coherent.

For both coherent and noncoherent data identifiers, both processor and external, SysCmd(7) marks the data element as the last data element, and SysCmd(6) indicates whether the data is response data or not. Response data is data returned in response to a read request or an intervention request. SysCmd(5) is the good data bit and indicates whether the data element is error free or not. Erroneous data contains an uncorrectable error. Erroneous data returned to the processor will cause a processor bus error. The processor will deliver data with the good data bit de-asserted when a primary parity error is detected for a transmitted data item. A secondary cache data ECC error can be detected by comparing the value transmitted on SysADC and SysADC. For external data identifiers, both coherent and noncoherent, SysCmd(4) indicates to the processor whether to check the data and check bits for this data element and SysCmd(3) is reserved. For processor data identifiers, both coherent and noncoherent, SysCmd(4:3) are reserved.

For coherent data identifiers SysCmd(2:0) indicate a cache state for the data. The cache state will provide the cache state with which to load the cache line for responses to processor coherent read requests. The cache state will indicate the cache state in which the line was found for data associated with the response to an external intervention request or for the data cycle issued in response to an external snoop request. For noncoherent data identifiers SysCmd(2:0) is reserved.

The encoding of SysCmd(7:3) for processor data identifiers is illustrated in **Table 4-19 Encoding of SysCmd(7:3) for Processor Data Identifiers**. The encoding of SysCmd(7:3) for external data identifiers is illustrated in **Table 4-20 Encoding of SysCmd(7:3) for External Data Identifiers**. The encoding of SysCmd(2:0) for coherent data identifiers is illustrated in **Table 4-21 Encoding of SysCmd(2:0) for Coherent Data Identifiers.**

Table 4-19 Encoding of SysCmd(7:3) for Processor Data Identifiers

SysCmd(7)	**Last data element indication.**
0	Last data element.
1	Not the last data element.
SysCmd(6)	**Response data indication.**
0	Data is response data.
1	Data is not response data.
SysCmd(5)	**Good data indication.**
0	Data is error free.
1	Data is erroneous.
SysCmd(4:3)	**Reserved.**

Table 4-20 Encoding of SysCmd(7:3) for External Data Identifiers

SysCmd(7)	**Last data element indication.**
0	Last data element.
1	Not the last data element.
SysCmd(6)	**Response data indication.**
0	Data is response data.
1	Data is not response data.
SysCmd(5)	**Good data indication.**
0	Data is error free.
1	Data is erroneous.
SysCmd(4)	**Data checking enable.**
0	Check the data and check bits.
1	Don't check the data and check bits
SysCmd(3)	**Reserved.**

Table 4-21 Encoding of SysCmd(2:0) for Coherent Data Identifiers

SysCmd(2:0)	Cache state.
0	Invalid.
1	Reserved.
2	Reserved.
3	Reserved.
4	Clean Exclusive.
5	Dirty Exclusive.
6	Shared.
7	Dirty Shared.

4.9 System Interface Addresses

System interface addresses are full 36 bit physical addresses presented on the least significant 36 bits (bits 35 through 0) of the SysAD bus during address cycles. The remaining bits of the SysAD bus are unused during address cycles. Addresses associated with double word, word, or partial word transactions, i.e. double word, word, or partial word read and write requests and update requests, are aligned for the size of the data element. Specifically, for double word requests, the low order three bits of the address will be zero, for 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, tri-byte, quinti-byte, sexti-byte and septi-byte requests the address provided will be a byte address. For further details on addresses supplied by the VR4000SC processor on double word, word, and partial word accesses see **VR4000 USER'S MANUAL (PRELIMINARY) ARCHITECTURE**.

Addresses associated with block requests are aligned to double word boundaries; that is the low order three bits of the address will be zero. The order in which data is returned in response to a processor block read request can be programmed via the boot time mode control interface to sequential ordering or sub-block ordering. If sequential ordering is enabled the processor will always deliver the address of the double word at the beginning of the block on a block read request. An external agent must return the block of data sequentially starting at the beginning of the block. If sub-block ordering is enabled the processor will deliver the address of the double word within the block that it wants returned first. An external agent must return the block of data using sub-block ordering starting with the addressed double word. For further details on sub-block ordering see **APPENDIX A SUB-BLOCK ORDERING**. Only a VR4000SC in the large package configuration with a secondary cache may be programmed to use sequential ordering.

For block write requests, the VR4000SC processor will always deliver the double word address of the double word at the beginning of the block and deliver data beginning with the double word at the beginning of the block and progressing sequentially through the double words that form the block.

4.10 Processor Internal Address Map

External reads and writes to the VR4000SC processor are provided to access processor internal resources that may be of interest to an external agent. However, version 1.2 of the VR4000SC does not contain any resources that are readable with an external read request. Version 1.2 of the VR4000SC will return a bus error response to any external read request. The only writable resource in version 1.2 of the VR4000SC is the processor interrupts.

The processor decodes bits 6:4 of the address associated with an external read or write request to determine which processor internal resource is the target of the request For version 1.2 of the VR4000SC, the only processor internal resource available for access by an external request is the interrupt resource, and it is only accessible via an external write request. The interrupt resource is accessed via an external write request with an address of 000 on bits 6:4 of the SysAD bus. See **CHAPTER 8 PROCESSOR INTERRUPTS** for further details on external writes to the interrupt resource.

4.11 Coherence Conflicts

The VR4000SC processor in the large package configuration will both issue processor coherence requests and accept external coherence requests. Processor coherence requests are processor coherent read requests. External coherence requests are external invalidate, update, snoop and intervention requests. Because of the overlapped nature of the system interface it is possible for processor coherence requests and external coherence requests to **conflict**. That is, it is possible for an external coherence request to reference an address that targets the same cache line as a pending processor read request. The processor does not contain comparators to detect such conflicts. The processor uses the secondary cache as the single point of reference to determine the coherency actions it will take and only checks the state of the secondary cache at specific times.

For pending processor coherent read requests conflicting external requests cannot effect the behavior of the processor. The processor will only issue a read request for a particular cache line if it does not have a copy of that cache line. Therefore, any external coherence request that targets a cache line that is also the target of a pending processor coherent read request will not find the line present in the cache. External coherence requests do not change the state of the cache unless the cache line they target is present. Since no change can be made to the state of the cache for the line that is the target of the pending processor read request, no external coherence requests can effect the read request. Therefore, external coherence requests that conflict with a pending processor coherent read request may be issued to the processor and will effectively be discarded by the processor.

The interactions between processor coherence requests and conflicting external coherence requests, tabulated by processor state, is summarized in **Table 4-22 Coherence Conflicts Summary (a)** and **Table 4-23 Coherence Conflicts Summary (b)**. The processor can be in one of the following states:

(1) Idle, no processor transactions currently pending;

(2) Read Pending, a processor coherent read request has been issued but the read response has not yet been received;

Table 4-22 Coherence Conflicts Summary (a)

Processor State	Conflicting External Coherence Request			
	Invalidate	Invalidate w/Cancel	Update	Update w/Cancel
Idle	NA	Undefined	NA	Undefined
Read Pending	OK	Undefined	OK	Undefined

Table 4-23 Coherence Conflicts Summary (b)

Processor State	Conflicting External Coherence Request			
	Intervention	Intervention w/Cancel	Snoop	Snoop w/Cancel
Idle	NA	Undefined	NA	Undefined
Read Pending	OK	Undefined	OK	Undefined

4.12 System Implications of Coherence Conflicts

The constraints that the VR4000SC processor places on the handling of conflicting coherency transactions has certain implications for the design of a system employing the VR4000SC. This section will consider, as an example, a particular snoopy, split-read, bus based system and the requirements for that system to correctly handle coherence conflicts.

4.12.1 System Model

The system model consists of the following components:

(1) Four processor subsystems, each consisting of a VR4000SC processor, a secondary cache, and an external agent. The agent communicates with the VR4000SC, accepting processor requests and issuing external requests, and with the system bus likewise issuing and receiving bus requests.

(2) A memory subsystem that communicates with main memory and the system bus.

(3) A system bus with the following characteristics:

It is a multiple master, request based, arbitrated bus in which an agent that wishes to perform a transaction on the bus must request the bus and wait for global arbitration logic to supply a grant signal before assuming mastership of the bus. Once mastership has been granted, the agent may begin a transaction.

It supports a read transaction, read exclusive transaction, write transaction, and invalidate transaction.

It is a split-read bus in that independent transactions may occur on the bus between a read request from a particular agent and the return of data by the target of the read request. The return of data by the target of the read request will be referred to as the read response.

It is a snoopy bus in that all agents connected to the bus must monitor all of the traffic on the bus to correctly maintain cache coherency.

I/O is not considered in this system model.

CHAPTER 5 ERROR CHECKING AND CORRECTING (ECC)

The VR4000SC processor provides sixteen check bits for the secondary cache data bus, seven check bits for the secondary cache tag bus and eight check bits for the system interface address and data bus. The sixteen check bits for the secondary cache data bus are organized as eight check bits for the upper sixty-four bits of the data bus and eight check bits for the lower sixty-four bits of the data bus. In addition, a single check bit is provided for the system interface command bus.

The eight check bits for the system interface address and data bus provide either even byte parity or are generated in accordance with a single error correcting double error detecting (SECDED) code that also detects any three of four bit error in a nibble. The eight check bits for each half of the secondary cache data bus are always generated in accordance with the SECDED code.

The processor checks data using parity or the SECDED code as it passes from the system interface to the secondary cache and as it is moved from the secondary cache to the primary cache or to the system interface. The processor passes the check bits for data accessed from the secondary cache directly to the system interface without change as it checks it. The processor does not check data received from the system interface for external updates and external writes. It is possible to force the processor to not check data from the system interface for read responses using a bit in the data identifier. The processor does generate correct check bits for double word, word, or partial word data transmitted to the system interface. The processor does not check addresses received from the system interface, but does generate correct check bits for addresses transmitted to the system interface. The processor does not contain a data corrector, rather, the processor will trap when an error is detected based on the data check bits. Software, in conjunction with an off processor data corrector, is responsible for correcting the data when the SECDED code is employed.

The seven check bits for the secondary cache tag bus are generated in accordance with a single error correcting double error detecting (SECDED) code that also detects any three or four bit error in a nibble. The processor generates check bits for the tag when it is written into the secondary cache and checks the tag whenever the secondary cache is accessed. The processor contains a corrector for the secondary cache tag. The tag corrector is not in-line for processor accesses due to primary cache misses. When a tag error is detected on a processor access due to a primary cache miss the processor will trap. Software, using the VR4000SC cache management primitives, will cause the tag to be corrected. When executing the cache management primitives, the processor uses the corrected tag to generate write back addresses and cache state. For external accesses the tag corrector is in-line; that is the response to external accesses will be based on the corrected tag. The processor will still trap on tag errors detected during external accesses to allow software to repair the contents of the cache if possible.

The check bit for the system interface command bus provides even parity over the nine bits of the system interface command bus. This parity bit is generated correctly when the system interface is in master state, but is not checked when the system interface is in slave state.

The buses that are covered by check bits and their contents and whether they are checked or not for various processor internal and external transactions is summarized in **Table 5-1 Error Checking and Correcting Summary for Internal Transactions** and **Table 5-2 Error Checking and Correcting Summary for External Transactions.**

Table 5-1 Error Checking and Correcting Summary for Internal Transactions (1/2)

Bus	Secondary Cache to Primary Cache	Primary Cache to Secondary Cache	Uncached Load	Uncached Store
Processor or Secondary Cache Data	Checked, Trap on Error	Primary cache parity	From System Interface	Not Checked
Secondary Cache Data check bits	Checked, Trap on Error	Generated	NA	NA
Secondary Cache Tag & check bits	Checked, not corrected Trap on error	Generated	NA	NA
System Int Addr/ Cmd & check bits transmit	NA	NA	Generated	Generated
System Int Addr/ Cmd & check bits receive	NA	NA	Not Checked	NA
System Int Data	NA	NA	Not Checked	From Processor

Table 5-1 Error Checking and Correcting Summary for Internal Transactions (2/2)

Bus	Store to Shared Cache Line	Cache Instruction	Secondary Cache Load from System Int	Secondary Cache Write to System Int
Processor or Secondary Cache Data	Checked on read part of RMW, Trap on Error	Not Checked	From System Int unchanged	Checked, Trap on Error
Secondary Cache Data check bits	Checked on read part of RMW, Trap on	Not Checked	From System Int unchanged	Checked, Trap or Error
Secondary Cache Tag & check bits	Checked on read part of RMW, Trap on Error	Checked and corrected	Generated	Checked, not corrected, Trap on
System Int Addr/ Cmd & check bits transmit	Generated	Generated	Generated	Generated
System Int Addr/ Cmd & check bits receive	NA	NA	Not Checked	NA
System Int Data	From processor	From Secondary	Checked, Trap on	From Secondary

Table 5-2 Error Checking and Correcting Summary for External Transactions (1/2)

Bus	Read Request	Write Request	Invalidate or Update Request
Processor or Secondary Cache Data	NA	NA	Checked on read part of RMW, Trap on Error
Secondary Cache Data check bits	NA	NA	Checked on read part of RMW, Trap on Error
Secondary Cache Tag & check bits	NA	NA	Checked on read part of RMW, Trap on Error
System Int Addr/ Cmd & check bits transmit	Generated	NA	NA
System Int Addr/ Cmd & check bits receive	Not Checked	Not Checked	Not Checked
System Int Data	From processor	Not Checked	Not Checked

Table 5-2 Error Checking and Correcting Summary for External Transactions (2/2)

Bus	Intervention Request Data Returned	Intervention Request State Returned	Snoop Request
Processor or Secondary Cache Data	Checked, Trap on Error	Checked, Trap on Error	Checked, Trap on Error
Secondary Cache Data check bits	Checked, Trap on Error	Checked, Trap on Error	Checked, Trap on Error
Secondary Cache Tag & check bits	Checked and corrected on read part of RMW, Trap on Error, Generation on write part of RMW if written.	Checked and corrected on read part of RMW, Trap on Error, Generation on write part of RMW if written.	Checked and corrected on read part of RMW, Trap on Error, Generation on write part of RMW if written.
System Int Addr/ Cmd & check bits transmit	Generated	Generated	Generated
System Int Addr/ Cmd & check bits receive	Not Checked	Not Checked	Not Checked

5.1 Single Error Correcting Double Error Detecting Codes

The ECC codes chosen for the processor's secondary cache data and secondary cache tag are single error correcting double error detecting codes that also detect three or four bit errors within a nibble. These codes were developed from codes proposed by M.Y. Hsiao in his paper, "A Class of Optimal Minimum Odd-weight-column SEC-DED Codes". The 64 bit data code is a modification of one of the 64 bit codes proposed by Hsiao to include the ability to detect three and four bit errors within a nibble. The 25 bit tag code was created using the patterns observed in the 64 bit data code.

The data code has the following properties:

(1) It is a single error correcting, double error and three or four bit error within a nibble detecting code.
(2) It provides 64 data bits protected by 8 check bits yielding 8 bit syndromes.
(3) It is minimal in that each parity tree used to generate the syndrome has only 27 inputs, the minimum possible number.
(4) It provides byte XOrs of the data bits as part of the XOr trees used to build the parity generators. This allows picking byte parity out of the XOr trees that generate or check the code.
(5) Single bit errors are indicated by syndromes that contain exactly 3 ones or by syndromes that contain exactly 5 ones in which bits 0-3 or bits 4-7 of the syndrome are all one This makes it possible to decode the syndrome to find which data bit is in error with 4 input NAND gates, provided a pre-decode AND of bits 0-3 and bits 4-7 of the syndrome is available. For the check bits a full 8 bit decode of the syndrome is required.
(6) Double bit errors are indicated by syndromes that contain an even number of ones.
(7) Three bit errors within a nibble are indicates by syndromes that contain 5 ones in which bits 0-3 of the syndrome and bits 4-7 of the syndrome are not all one.
(8) Four bit errors within a nibble are indicated by syndromes that contain 4 ones. Because this is an even number of ones, four bit errors within a nibble look like double bit errors.

The tag code has the following properties:

(1) It is a single error correcting, double error and three or four bit error within a nibble detecting code.
(2) It provides 25 data bits protected by 7 check bits yielding 7 bit syndromes.
(3) It provides byte XOrs of the data bits as part of the XOr trees used to build the parity generators. This allows picking byte parity out of the XOr trees that generate or check the code.
(4) Single bit errors are indicated by syndromes that contain exactly 3 ones. This makes it possible to decode the syndrome to find which data bit is in error with 3 input NAND gates. For the check bits a full 7 bit decode of the syndrome is required.
(5) Double bit errors are indicated by syndromes that contain an even number of ones.
(6) Three bit errors within a nibble are indicated by syndromes that contain 5 ones or 7 ones.
(7) Four bit errors within a nibble are indicated by syndromes that contain 4 ones or 6 ones. Because these are even numbers of ones, four bit errors within a nibble look like double bit errors.

The parity check matrices for the data ECC code and the tag ECC code specifying the distribution of data and check bits across nibbles are shown in **Fig. 5-1 Parity Check Matrix for the Data ECC Code** and **Fig. 5-2 Parity Check Matrix for the Tag ECC Code.**

Fig. 5-1 Parity Check Matrix for the Data ECC Code

Check Bit			43		52											70		61	
Data Bit		6666 3210	55 98	5555 7654	55 32	5544 1098	4444 7654	4444 3210	3333 9876	3333 5432	3322 1098	2222 7654	2222 3210	1111 9876	1111 5432	11 10	9876	54	3210
Number of ones per row	27	1111	11..	11..	1...	1...	....	1111	1111	....	1...	1...	1...	1...	....	1.1.	.1..	1...	1...
	27	1111	1...	1...	1...	.1..	....	....	....	1111	.1..	.1..	.1..	.1..	1111	11..	11..	1.1.	.1..
	27	....	1...	11..	1.1.	..1.	1111	1111	....	....	..1.	..1.	..1.	..1.	1111	1...	1...	11..	..1.
	27	....	1.1.	.1..	11..	...1	1111	....	1111	1111	...1	...1	...1	...1	....	1...	11..	1...	...1
	27	1...	.1.1	..11	.1..	....	1...	1...	1...	1...	1111	1111	....	1111	1...	11..	...1	.1..	....
	27	.1..	11..	..1.	.1.1	1111	.1..	.1..	.1..	.1..	....	....	1111	1111	.1..	.1..	..11	.1..	....
	27	..1.	.1..	..11	11..	1111	..1.	..1.	..1.	..1.	1111	....	....	....	..1.	.1..	..1.	.1.1	1111
	27	...1	.1..	...1	.1..	....	...1	...1	...1	...1	....	1111	1111	....	...1	.1.1	..11	11..	1111
Number of ones per column		3333	5511	3333	5511	3333	3333	3333	3333	3333	3333	3333	3333	3333	3333	5511	3333	5511	3333

5

6427525 0083632 200

Fig. 5-2 Parity Check Matrix for the Tag ECC Code

Check Bit		0	12	34	56				
Data Bit		222 432	22 10	11 98	11 76	1111 5432	11 1098	7654	3210
Number of ones per row	11	. 1. .	1. . .	1. . .	. . . 1	1111	1. . .	1. . .	1. . .
	13	1. . .	. 1. .	. 1. .	. . 1.	1111	1111	. . . .	. 1. .
	10	. . 1.	1. . .	. . . 1	1. . .	. . . .	1111	. 1. .	. . 1.
	10	. 1. .	. 1. .	. . 1.	. 1. .	1. . .	. 1. .	1111	. . . .
	13	1. . .	. . . 1	1. . .	1. . .	. 1. .	. . . .	1111	1111
	11	. . 1.	. . 1.	. 1. .	. 1. .	. . 1.	. . 1.	. . 1.	1111
	14	1111	11. .	11. .	11. .	. . . 1	. . . 1	. . . 1	. . . 1
Number of ones per column		3331	3311	3311	3311	3333	3333	3333	3333

CHAPTER 6 BOOT TIME MODE CONTROL INTERFACE

Fundamental operational modes for the processor are initialized via the boot time mode control interface. The boot time mode control interface is a serial interface operating at a very low frequency to allow the initialization information to be kept in a low cost EPROM.

Immediately after the VDDOk signal is asserted, the processor will read a serial bit stream of 256 bits to initialize all fundamental operational modes. After initialization is complete, the processor will continue to drive the serial clock output but no further initialization bits will be read.

6.1 Boot Time Mode Control Interface Signal Summary

ModeIn:	(i)	Serial boot mode data in.
ModeClock:	(o)	Serial boot mode data clock out at the MasterClock frequency divided by 256.

6.2 Boot Time Mode Control Interface Operation

While the VDDOk signal is de-asserted, the **ModeClock** output will be held asserted. After the VDDOk signal is asserted, the first bit in the initialization bit stream must be present at the **ModeIn** input. The processor will synchronize the ModeClock output at the time VDDOk is asserted, and the first rising edge of the ModeClock will occur 256 MasterClock clock cycles after VDDOk is asserted. After each rising edge of the ModeClock, the next bit of the initialization bit stream must be presented at the ModeIn input. The processor will sample exactly 256 initialization bits from the ModeIn input on the rising edge of the ModeClock.

6.3 Boot Time Modes

The correspondence between bits of the initialization bit stream and processor mode settings is illustrated in **Table 6-1 Boot Time Modes**. Bit 0 of the bit stream is the bit presented to the processor when VDDOk is de-asserted. For mode settings defined by multiple bit fields, the lowest numbered serial bit in the field is the most significant bit.

Table 6-1 Boot Time Modes (1/4)

Serial Bit	Value	Mode Setting
0		**BlkOrder:** Secondary Cache Mode block read response ordering.
	0	Sequential ordering.
	1	Sub-block ordering.
1		**EIBParMode:** Specifies nature of system interface check bus.
	0	SECDED error checking and correcting mode.
	1	Byte parity.
2		**EndBit:** Specifies byte ordering.
	0	Little Endian ordering.
	1	Big Endian ordering.
3		**Reserved.**
4		**NoSCMode:** Specifies presence of secondary cache.
	0	Secondary cache present.
	1	No secondary cache present.
5:6		**SysPort:** System Interface port width, bit 6 most significant.
	0	64 bits.
	1-3	Reserved.
7		**SC64BitMd:** Secondary cache interface port width.
	0	128 bits.
	1	Reserved.
8		**EISpltMd:** Specifies secondary cache organization
	0	Secondary cache unified.
	1	Reserved.
9:10		**SCBlkSz:** Secondary cache line size, bit 10 most significant.
	0	4 words.
	1	8 words.
	2	16 words.
	3	32 words.
11:14		**XmitDatPat:** System interface data rate, bit 14 most significant.
	0	D
	1	DDx
	2	DDxx
	3	DxDx
	4	DDxxx
	5	DDxxxx
	6	DxxDxx
	7	DDxxxxxx
	8	DxxxDxxx
	9-15	Reserved.

Table 6-1 Boot Time Modes (2/4)

Serial Bit	Value	Mode Setting
15:17		**SysCkRatio:** PClock to SClock divisor, frequency relationship between SClock, RClock, and TClock and PClock, bit 17 most significant.
	0	Divide by 2.
	1	Divide by 3.
	2	Divide by 4.
18	0	Reserved.
19		**TimIntDis:** Timer Interrupt disables timer interrupts, otherwise the interrupt used by the timer becomes a general-purpose interrupt.
	0	Timer Interrupt enabled.
	1	Timer Interrupt disabled.
20		**PotUpdDis:** Potential update enable allows potential updates to be issued. Otherwise only compulsory updates are issued.
	0	Potential updates enabled.
	1	Potential updates disabled.
21:24		**TWrSup:** Secondary cache write deassertion delay, TWrSup in PCycles, bit 24 most significant.
25:26		**TWr2Dly:** Secondary cache write assertion delay 2, TWr2Dly in PCycles, bit 26 most significant.
27:28		**TWr1Dly:** Secondary cache write assertion delay 1, TWr1Dly in PCycles, bit 28 most significant.
29		**TWrRck:** Secondary cache write recovery time, TWrRc in PCycles.
	0	0 cycles.
	1	1 cycle.
30:32		**TDis:** Secondary cache disable time, TDis in PCycles, bit 32 most significant.
33:36		**TRd2Cyc:** Secondary cache read cycle time 2, TRdCyc2 in PCycles, bit 36 most significant.
37:40		**TRd1Cyc:** Secondary cache read cycle time 1, TRdCyc1 in PCycles, bit 40 most significant.
41	0	Reserved.
42	0	Reserved.
43	0	Reserved.
44	0	Reserved.
45	0	Reserved.
46		**Pkg179:** VR4000SC Package type.
	0	Large (447 pin).
	1 **Note**	Small (179).

Note The behavior of the VR4000SC is undefined when it is selected.

Table 6-1 Boot Time Modes (3/4)

Serial Bit	Value	Mode Setting
47:49		**CycDivisor:** This mode determines the clock divisor for the reduced-power mode. When the RP bit in the Status register is set to one, the pipeline clock is divided by one of the following values.
	0	Divide by 2
	1	Divide by 4
	2	Divide by 8
	3	Divide by 16
	4-7	Reserved.
50		**Drv0_50:** Drive the outputs out in 0.50 × MasterClock period.
	0	Drive at some other fraction of a MasterClock period.
	1	Drive at 0.50 × MasterClock period.
51		**Drv0_75:** Drive the outputs out in 0.75 × MasterClock period.
	0	Drive at some other fraction of a MasterClock period.
	1	Drive at 0.75 × MasterClock period.
52		**Drv1_00:** Drive the outputs out in 1.00 × MasterClock period.
	0	Drive at some other fraction of a MasterClock period.
	1	Drive at 1.00 × MasterClock period.
53:56		**InitP:** Initial values for the state bits that determine the pull-down di/dt and switching speed of the output buffers. Bit 53 is the most significant.
	0	Fastest pull-down rate.
	1-14	Intermediate pull-down rates.
	15	Slowest pull-down rate.
57:60		**InitN:** Initial values for the state bits that determine the pull-up di/dt and switching speed of the output buffers. Bit 57 is the most significant.
	0	Slowest pull-up rate.
	1-14	Intermediate pull-up rates.
	15	Fastest pull-up rate.
61		**EnblDPLLR:** Enables the negative feedback loop that determines the di/dt and switching speed of the output buffers only during ColdReset.
	0	Disable di/dt control mechanism.
	1	Enable di/dt control mechanism.
62		**EnblDPLL:** Enables the negative feedback loop that determines the di/dt and switching speed of the output buffers during ColdReset and during normal operation.
	0	Disable di/dt control mechanism.
	1	Enable di/dt control mechanism.

Table 6-1 Boot Time Modes (4/4)

Serial Bit	Value	Mode Setting
63		**DsblPLL:** Enables PLLs that match MasterIn and produce RClock, TClock, SClock and the internal clocks.
	0	Enable PLLs.
	1	Disable PLLs.
64		**SRTristate:** Controls when output-only pins are tristated
	0	Only when $\overline{\text{ColdReset}}$ is asserted.
	1	When $\overline{\text{Reset}}$ or $\overline{\text{ColdReset}}$ are asserted.

CHAPTER 7 RESET SEQUENCE FOR THE VR4000SC PROCESSOR

The VR4000SC processor requires a multi-level reset sequence using the V_{DD}Ok, $\overline{\text{ColdReset}}$, and $\overline{\text{Reset}}$ inputs. For a power-on reset or a cold reset the following sequence must be applied to the VR4000SC:

(1) Stable V_{DD} of at least 4.75 volts from the +5 volt power supply must be applied to the VR4000SC continuously for at least 100 milliseconds along with a stable continuous system clock at the desired operational frequency of the VR4000SC.

(2) After at least 100 milliseconds of stable V_{DD} and MasterClock the V_{DD}Ok input to the VR4000SC may be asserted. While V_{DD}Ok is de-asserted, all pins will be three-stated with the exception of ModeClock. The assertion of V_{DD}Ok will cause the VR4000SC to begin driving a clock on the ModeClock output and to read the boot time mode control serial data stream. For further details on mode initialization see **CHAPTER 6 BOOT TIME MODE CONTROL INTERFACE**. After the mode bits have been read in, the VR4000SC will allow its internal phase locked loops to lock, stabilizing the processor internal clock, PClock, the SyncOut-SyncIn clock path and the master clock output, MasterOut. V_{DD}Ok must become asserted at least 100 milliseconds before the de-assertion of $\overline{\text{ColdReset}}$.

(3) Once the boot time mode control serial data stream has been read by the VR4000SC, the $\overline{\text{ColdReset}}$ input may be de-asserted. $\overline{\text{ColdReset}}$ must remain asserted for at least 100 milliseconds after the assertion of V_{DD}Ok. $\overline{\text{ColdReset}}$ must be de-asserted synchronously with MasterClock. The processor internal clock, SClock, and the system interface clock, TClock, and RClock will begin to cycle with the de-assertion of $\overline{\text{ColdReset}}$. The state of SClock, TClock, and RClock is undefined until $\overline{\text{ColdReset}}$ is de-asserted. The de-asserted edge of $\overline{\text{ColdReset}}$ is used to synchronize the edges of SClock, TClock, and RClock, potentially across multiple processors in a multi-processor system.

(4) After $\overline{\text{ColdReset}}$ is de-asserted and SClock, TClock and RClock have stabilized, $\overline{\text{Reset}}$ may be de-asserted to allow the VR4000SC to begin to run. $\overline{\text{Reset}}$ must be held asserted for at least 64 MasterClock cycles after the de-assertion of $\overline{\text{ColdReset}}$. $\overline{\text{Reset}}$ must be de-asserted synchronously with MasterClock.

A cold reset requires the same sequence as described above except that power is presumed to have been stable for a long time before the assertion of the reset inputs and the de-assertion of V_{DD}Ok. V_{DD}Ok must be de-asserted for a minimum of 64 MasterClock cycles before re-asserting to begin the reset sequence.

To effect a warm reset, the $\overline{\text{Reset}}$ input may be asserted synchronously with MasterClock and held asserted for at least 64 MasterClock cycles before being de-asserted synchronously with MasterClock. The processor internal clocks, PClock and SClock, and the system interface clocks, TClock and RClock will not be affected by a warm reset, and the boot time mode control serial data stream will not be read by the VR4000SC on a warm reset.

The master clock output, MasterOut, is provided for use in generating the reset related signals for the VR4000SC that must be synchronous with MasterClock.

After a power on reset, cold reset, or warm reset, all processor internal state machines will be reset, and the processor will begin execution at the reset vector. All processor internal state is preserved during a warm reset, although the precise state of the caches will depend on whether a cache miss sequence has been interrupted by resetting the processor state machines.

$\overline{\text{ColdReset}}$ must be asserted when V_{DD}Ok asserts. The behavior of the VR4000SC is undefined if V_{DD}Ok asserts while $\overline{\text{ColdReset}}$ is de-asserted.

CHAPTER 8 PROCESSOR INTERRUPTS

The VR4000SC processor supports six hardware interrupts, two software interrupts, and a non-maskable interrupt as described in **VR4000 USER'S MANUAL (PRELIMINARY) ARCHITECTURE**. The six hardware interrupts on the VR4000SC are accessible via external write requests in the large package configuration. The large package configuration has one pin, Int0, dedicated as the hardware interrupt. The non-maskable interrupt is accessible via external write requests and a dedicated pin.

External writes to the processor are directed based on a processor internal address map to various processor internal resources. An external write to any address with SysAD[6..4] = 0 will be a write to the **interrupt register**. During the data cycle, SysAD[22..16] are the write enables for the 7 individual interrupt register bits (one non-maskable interrupt + 6 regular Interrupts), and SysAD[6..0] are the values to be written into these bits. This allows us to set and clear any subset of the interrupt register with a single write request. Bits 5:0 of the interrupt register are directly readable as bits 15:10 of the Cause register. The bit 0 of the interrupt register is bitwise ORed with the current value of the interrupt pin **$\overline{\text{Int(0)}}$** and the result is directly readable as bit 10 of the Cause register. Bit 6 of the interrupt register is ORed with the current value of the non-maskable interrupt pin **$\overline{\text{NMI}}$** to form the non-maskable interrupt input to the VR4000SC.

CHAPTER 9 PROCESSOR STATUS OUTPUTS

The VR4000SC processor provides eight status outputs, Status(7:0), that change with each rising edge of MasterClock to indicate the processor's internal state during each of the two most recent PCycles. Status(7:0) is treated as two fields, Status(3:0) indicates the processor's internal state during the most recent PCycle and Status(7:4) indicates the processor's internal state during the PCycle preceding the most recent PCycle. The encoding of processor internal state for Status(7:4) or Status(3:0) is shown in **Table 9-1 Encoding of Processor Internal State for Status(7:4) or Status(3:0)**. The four bit decode describes the instruction occupying the WB stage during a given PCycle.

Table 9-1 Encoding of Processor Internal State for Status(7:4) or Status(3:0)

Status(7:4) or Status(3:0)	Processor internal state	
0	Run cycle:	Other integer instruction
1	Run cycle:	Integer Load
2	Run cycle:	Integer Untaken Branch
3	Run cycle:	Integer Taken Branch
4	Run cycle:	Integer Store
5	Reserved	
6	Reserved	
7	Run cycle:	Killed by integer slip
8	Stall cycle:	Other stall type
9	Stall cycle:	Primary Instruction Cache
a	Stall cycle:	Primary Data Cache
b	Stall cycle:	Secondary Cache
c	Run cycle:	Floating Point
d	Run cycle:	Killed by branch
e	Run cycle:	Killed by exception
f	Run cycle:	Killed by floating point slip

CHAPTER 10 CLOCKING

The VR4000SC processor bases all internal and external clocking on the single clock input MasterClock. The processor generates the clock output SyncOut at the same frequency as MasterClock and aligns SyncIn with MasterClock. SyncOut must be connected to the clock input SyncIn so that the processor can compensate for output driver delays and input buffer delays in aligning SyncIn with MasterClock. The processor generates the clock output MasterOut at the same frequency as MasterClock and aligns MasterOut with SyncOut. MasterOut is provided for use in clocking external logic that must cycle at MasterClock frequency, such as the reset logic.

The processor generates the internal clock PClock for all internal latches and registers at twice the frequency of MasterClock and precisely aligns every other rising edge of PClock with the rising edge of MasterClock.

The processor divides PClock by a programmable divisor, programmed via the boot time mode control interface, to generate the internal clock SClock. SClock is used by the processor to sample data at the system interface and to clock data into the processor's system interface output registers. The rising edges of SClock are aligned with rising edges of PClock.

Data provided to the processor must be setup a minimum of t_{DS} nano-seconds (ns) before the rising edge of SClock and held valid for a minimum of t_{DH} ns after the rising edge of SClock. This setup and hold time is required for data to propagate through the processor's input buffers and meet the setup and hold times for the processor's input latches.

Data provided by the processor will become stable a minimum of t_{DM} ns after the rising edge of SClock and a maximum of t_{DO} ns after the rising edge of SClock. This drive off time is the sum of the maximum delay through the processor's output drivers and the maximum clock to Q delay of the processor's output registers.

Certain processor inputs, specifically VDDOk, $\overline{\text{ColdReset}}$, and $\overline{\text{Reset}}$ are sampled based on MasterClock while certain processor outputs, specifically Status(7:0) are driven out based on MasterClock. The same setup, hold, and drive off parameters, t_{DS}, t_{DH}, t_{DM}, and t_{DO}, will apply to these inputs and outputs but with respect to MasterClock instead of SClock.

The values of t_{DS}, t_{DH}, t_{DM}, and t_{DO} for various speed ratings of VR4000SC processor are tabulated in the data sheet (to be issued).

The processor generates two output clocks, RClock and TClock, at exactly the same frequency as SClock to be used by an external agent to sample and drive data. RClock is a receive clock that can be used by an external agent to clock its input registers. TClock is a transmit clock that can be used by an external agent to clock its output registers, and as the global system clock for the logic that makes up the external agent.

TClock is identical to SClock and the edges of TClock are precisely aligned with the edges of SClock. RClock is skewed with respect to TClock and SClock so that it leads TClock by 25% of the SClock, TClock, and RClock cycle time. TClock and RClock are not used by the processor.

The alignment of SyncOut, PClock, SClock, TClock, and RClock is accomplished by the processor with internal Phase Locked Loop (PLL) circuits that generate aligned clocks. PLL circuits by their nature are only capable of generating aligned clocks for MasterClock frequencies in a limited range. Minimum and maximum frequencies for MasterClock for various speed ratings of the VR4000SC processor are tabulated in the data sheet (to be issued). Clocks generated using PLL circuits contain some inherent inaccuracy in their alignment with the MasterClock called jitter. That is, a clock aligned with MasterClock by the processor's PLL circuits may lead or trail MasterClock by some maximum amount referred to as the maximum jitter. Maximum jitter for the clocks generated by various speed ratings of the VR4000SC processor is tabulated in the data sheet (to be issued).

The relationship of MasterClock, SyncOut, PClock, SClock, TClock, and RClock, and the characteristics of data on the SysAD bus when data is driven by the processor and received by the processor is illustrated in **Fig. 10-1 Processor Clocks, PClock to SClock Divisor of 2** and frequency of PClock divided by two while in **Fig. 10-2 Processor Clocks, PClock to SClock Divisor of 4**, SClock, TClock, and RClock are programmed to the frequency of PClock divided by four.

Fig. 10-1 Processor Clocks, PClock to SClock Divisor of 2

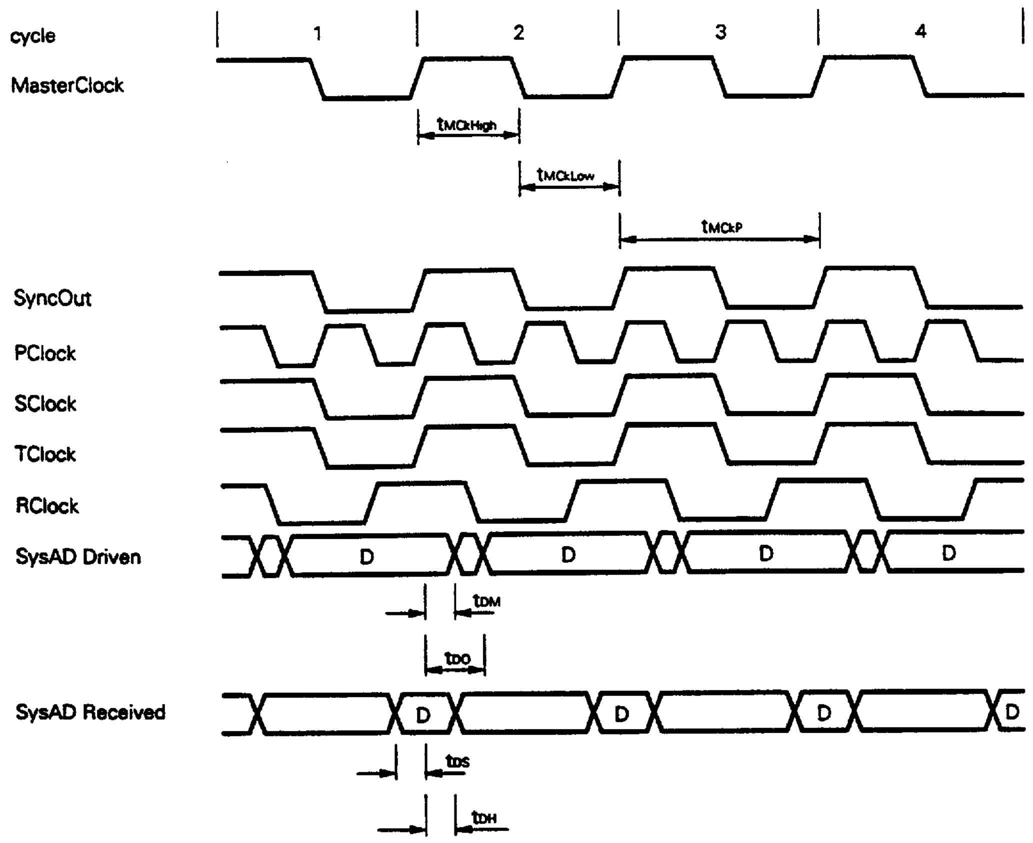
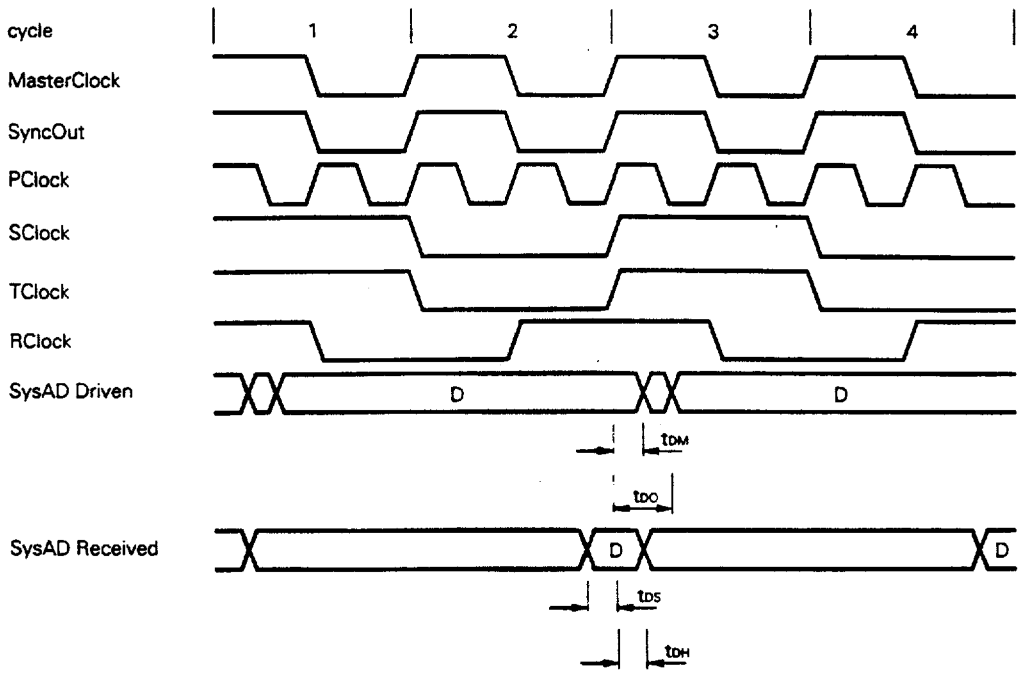


Fig. 10-2 Processor Clocks, PClock to SClock Divisor of 4



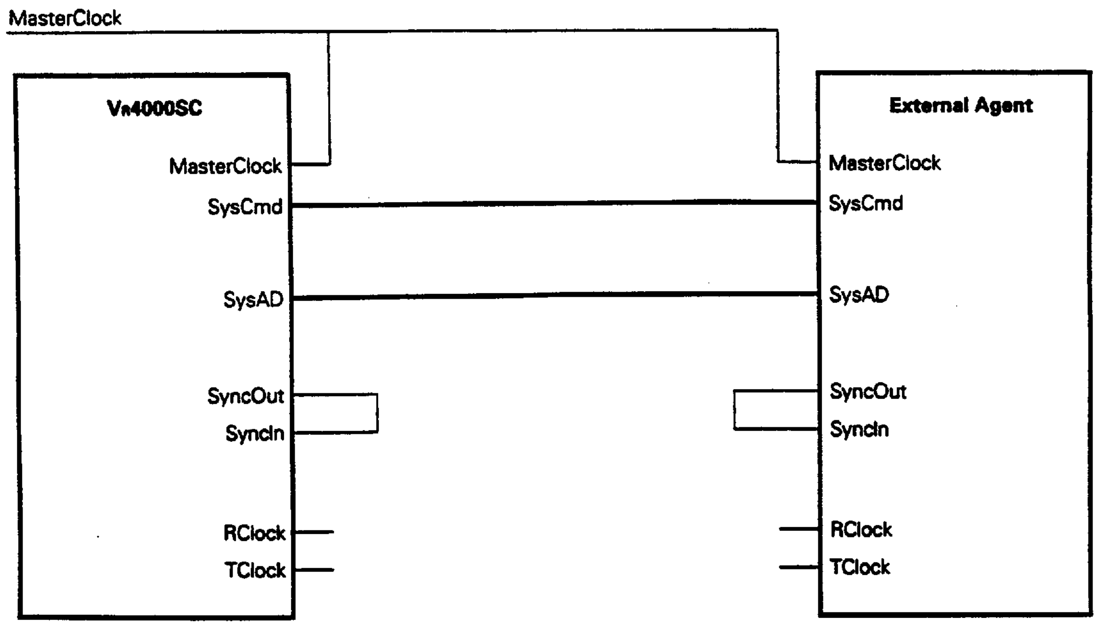
10.1 Clock Interfacing to a Phase Locked System

When the VR4000SC processor is employed in a phase locked system all of the components of the system must phase lock their operation to a common MasterClock. In such a system the delivery of data and sampling of data will have common characteristics for all components with perhaps different delay values for the components. The **transmission time**, the amount of time a signal has to propagate along the trace from one component to another, between any two components A and B of a phase locked system can be calculated from the following equation:

Transmission Time = (SClock period) – (t_{DO} for A) – (t_{DS} for B)
– (Clock Jitter for A Max) – (Clock Jitter for B Max)

A block level diagram of a phase locked system employing the VR4000SC processor is shown in **Fig. 10-3 Phase Locked System Employing the VR4000SC Processor**.

Fig. 10-3 Phase Locked System Employing the VR4000SC Processor



10.2 Clock Interfacing to a System Without Phase Lock

When the VR4000SC processor is employed in a system in which the other components are not capable of phase lock to a common MasterClock the output clocks RClock and TClock may be used to clock the remainder of the system. Two clocking methodologies are possible using RClock and TClock: the first better suited to communication with an external agent built from gate arrays and the second better suited to communication with an external agent built from discrete CMOS logic devices.

In the first clocking methodology, tailored for communication with an external agent built from gate arrays both RClock and TClock are used for clocking within the gate arrays. RClock is provided specifically so that a gate array may buffer it internally and use the buffered version to clock registers that sample VR4000SC outputs. These sample registers should be immediately followed by staging registers that are clocked by an internally buffered version of TClock. The buffered version of TClock should be used as the global system clock for the logic inside the gate array and as the clock for all registers that drive VR4000SC inputs.

Requiring staging registers following the registers that sample VR4000SC outputs places a constraint on the sum of the clock to Q delay of the sample registers and the setup time of the synchronizing registers inside the gate arrays. The sum must be less than 25% of the RClock and TClock period minus the maximum clock jitter of both RClock and TClock minus the maximum delay mismatch for the internal clock buffers on RClock and TClock.

The transmission time for a signal from the VR4000SC to an external agent composed of gate arrays in a system without phase lock can be calculated from the following equation:

Transmission Time = (75% of TClock period) – (t_{DO} for VR4000SC)

+ (External Clock Buffer Delay Min)

– (External Sample Register Setup Time)

– (Clock Jitter for VR4000SC Internal Clocks Max)

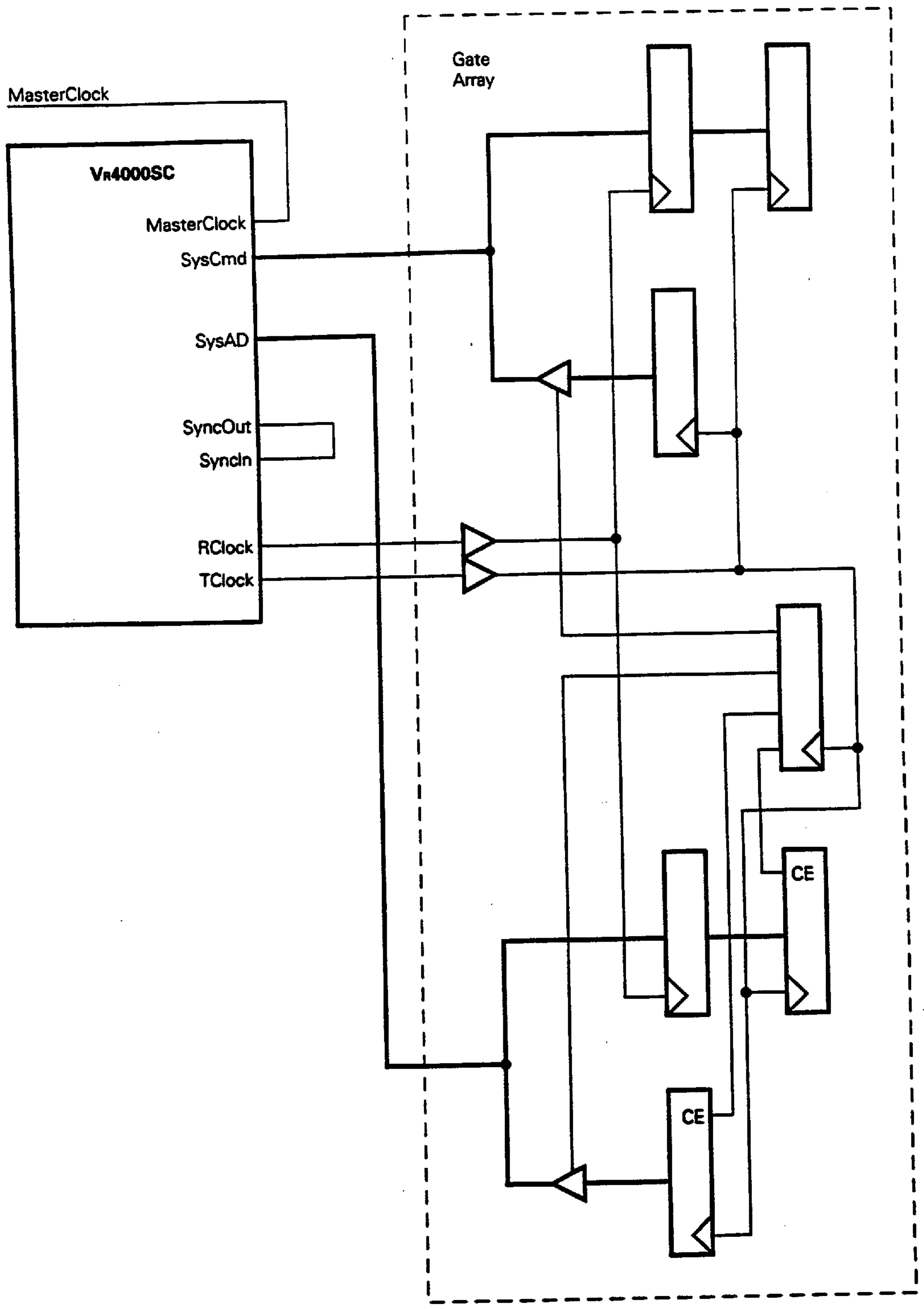
– (Clock Jitter for RClock Max)

The transmission time for a signal from an external agent composed of gate arrays to the VR4000SC in a system without phase lock can be calculated from the following equation:

Transmission Time = (TClock period) – (tDS for VR4000SC)
– (External Clock Buffer Delay Max)
– (External Output Register Clock to Q Delay Max)
– (Clock Jitter for TClock Max)
– (Clock Jitter for VR4000SC Internal Clocks Max)

A block level diagram of a system without phase lock employing the VR4000SC processor and an external agent composed of a gate array is shown in **Fig. 10-4 System Without Phase Lock Employing the VR4000SC Processor (a)**.

Fig. 10-4 System Without Phase Lock Employing the VR4000SC Processor (a)



In the second clocking methodology tailored for communication with an external agent built from discrete CMOS logic devices, matched delay clock buffers are used to allow the VR4000SC to generate aligned clocks for the external logic. One of the matched delay clock buffers is inserted in the processor's SyncOut SyncIn clock alignment path. This has the effect of skewing SyncOut, MasterOut, SClock, RClock, and TClock to lead MasterClock by the delay of the matched delay clock buffer while leaving PClock aligned with MasterClock. The remaining matched delay clock buffers can be used to generate a buffered version of TClock that will be aligned with MasterClock. The alignment error of the buffered version of TClock will be the sum of the maximum delay mismatch of the matched delay clock buffers and the maximum clock jitter of TClock. The buffered version of TClock will be used to clock registers that sample VR4000SC outputs, as the global system clock for the discrete logic that forms the external agent, and to clock registers that drive VR4000SC inputs.

The transmission time for a signal from the VR4000SC to an external agent composed of discrete CMOS logic devices can be calculated from the following equation:

Transmission Time = (TClock period) – (t_{DO} for VR4000SC)
– (External Sample Register Setup Time)
– (External Clock Buffer Delay Mismatch Max)
– (Clock Jitter for VR4000SC Internal Clocks Max)
– (Clock Jitter for TClock Max)

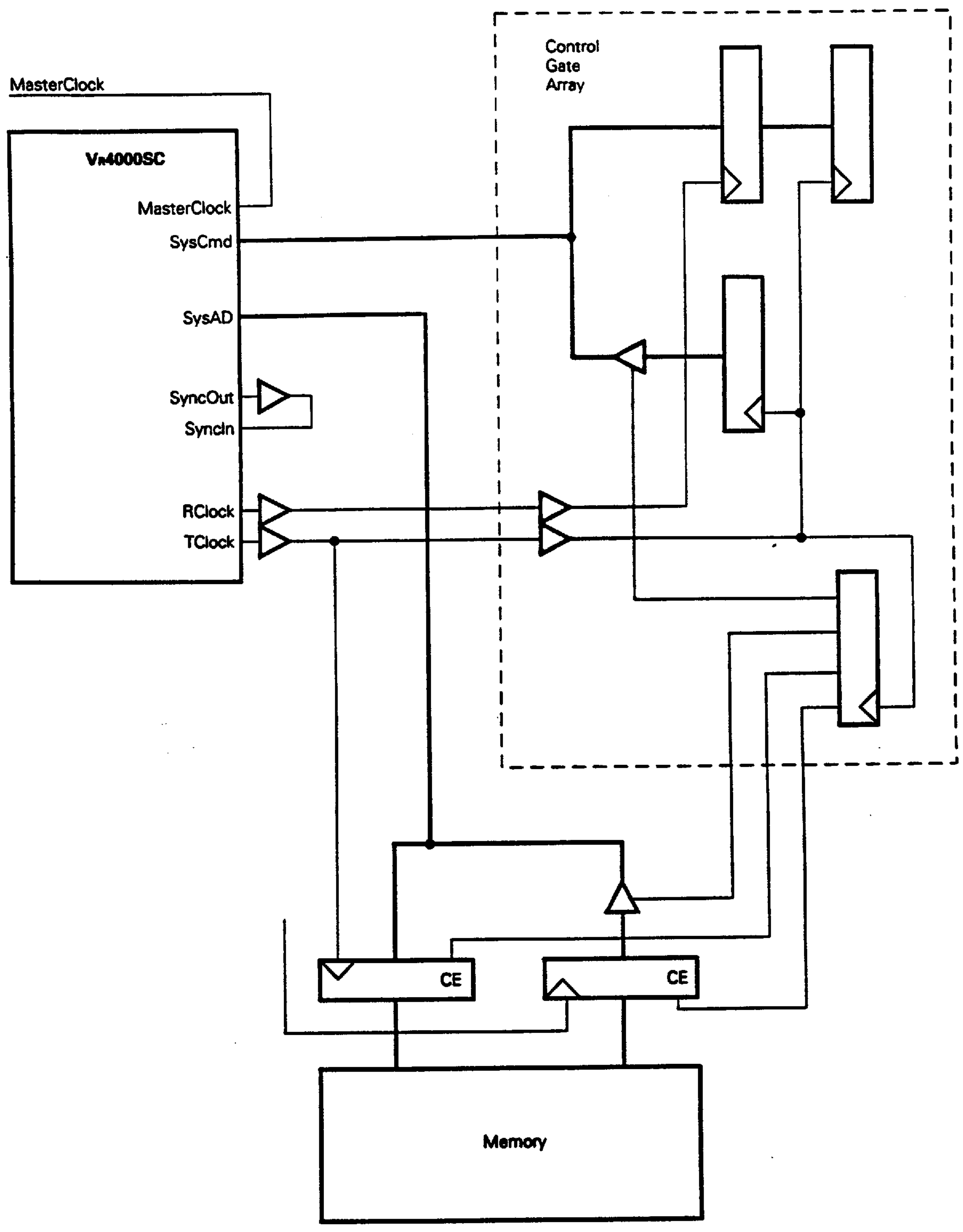
The transmission time for a signal from an external agent composed of discrete CMOS logic devices can be calculated from the following equation:

Transmission Time = (TClock period) – (t_{DS} for VR4000SC)
– (External Output Register Clock to Q Delay Max)
– (External Clock Buffer Delay Mismatch Max)
– (Clock Jitter for VR4000SC Internal Clocks Max)
– (Clock Jitter for TClock Max)

Note that using this clocking methodology the hold time of data driven from the VR4000SC to an external sampling register is a critical parameter. In order to guarantee hold time, the minimum output delay of the VR4000SC, t_{DM}, must be greater than the sum of the minimum hold time for the external sampling register, the maximum clock jitter for VR4000SC internal clocks, the maximum clock jitter for TClock, and the maximum delay mismatch of the external clock buffers.

A block level diagram of a system without phase lock employing the VR4000SC processor and an external agent composed of both a gate array and discrete CMOS logic devices is shown in **Fig. 10-5 System Without Phase Lock Employing the VR4000SC Processor (b)**.

Fig. 10-5 System Without Phase Lock Employing the VR4000SC Processor (b)



CHAPTER 11 OUTPUT BUFFER di/dt CONTROL MECHANISM

The speed of the VR4000SC output drivers is controlled by a negative feedback loop that insures that the drive off times are only as fast as necessary to meet the system requirement of single cycle transfers. This guarantees the minimum ground bounce due to the L*di/dt of the switching buffers consistent with the system timing requirements. Four bits are used to control each of the pull-up and pull-down delays. They are initially set to the values in the mode bits InitN<3..0> for pull-up and InitP<3..0> for pull-down.

Under normal conditions, the di/dt control mechanism is expected to be constantly enabled so that it can compensate the output buffer delay for any changes in the temperature or power supply voltage. The EnblDIDT mode bit should be set for this mode of operation.

For situations where the jitter associated with the operation of the di/dt control mechanism cannot be tolerated and where the variation in temperature and supply voltage after ColdReset is expected to be small, the di/dt control mechanism can be instructed to lock only during ColdReset and thereafter retain its control values. The EnblDIDTR mode bit should be set and the EnblDIDT mode bit should be cleared for this mode of operation.

In addition, if both the EnblDIDT and EnblDIDTR mode bits are cleared, the speed of the output buffers can be set with the InitP<3..0> and InitN<3..0> mode bits

The drive off delays can be set through the mode bits. Currently, delay of 0.5T, 0.75T, and T are supported corresponding to the Drv0.50, Drv0.75, and Drv1.00 mode bits where T is the MasterClock period. For example, in the Drv0.75 mode, the entire signal transmission path including the clock-to-Q, output buffer drive time, board flight time, input buffer delay, and setup time will be traversed in 0.75 * the MasterClock period plus or minus the jitter due to the di/dt control mechanism.

All output drivers on the VR4000SC, with the exception of the clock drivers, are controlled by the di/dt control mechanism. The delay due to the output buffer drive time component of the SCAddr<17..0>, SC64Addr, SCOEB, SCWRB, SCDCSB, and SCTCSB pins is approximately 66% of the delay of drivers of the other pins.

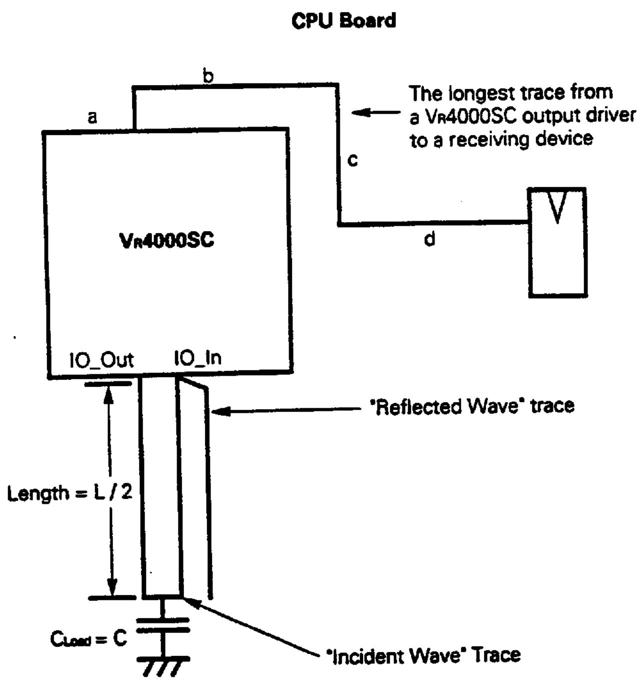
The VR4000SC determines the worst case propagation delay from an VR4000SC output driver to a receiving device by measuring the transmission line delay of the trace that connects the VR4000SC IO_Out and IO_In pins. This representative trace must have one and a half times the length and approximately the same capacitive loading as the worst case trace on any VR4000SC output.

The designer determines the trace characteristics by:

- measuring the longest path from an VR4000SC output driver to a receiving device: L
- calculating the maximum capacitive loading on any signal pin: C
- connecting an "incident wave" trace of length L with a capacitive loading of C between the IO_In and IO_Out pins of the VR4000SC
- and connecting a "reflected wave" trace of length L/2 to the IO_In pin of the VR4000SC.

A VR4000SC with appropriate traces connected to the IO_In and IO_Out pins is illustrated in **Fig. 11-1 IO_In/ IO_Out Board Trace.**

Fig. 11-1 IO_In/IO_Out Board Trace



L = a+b+c+d
C = Total Capacitance Loading of the worst case trace

CHAPTER 12 PLL PASSIVE COMPONENTS

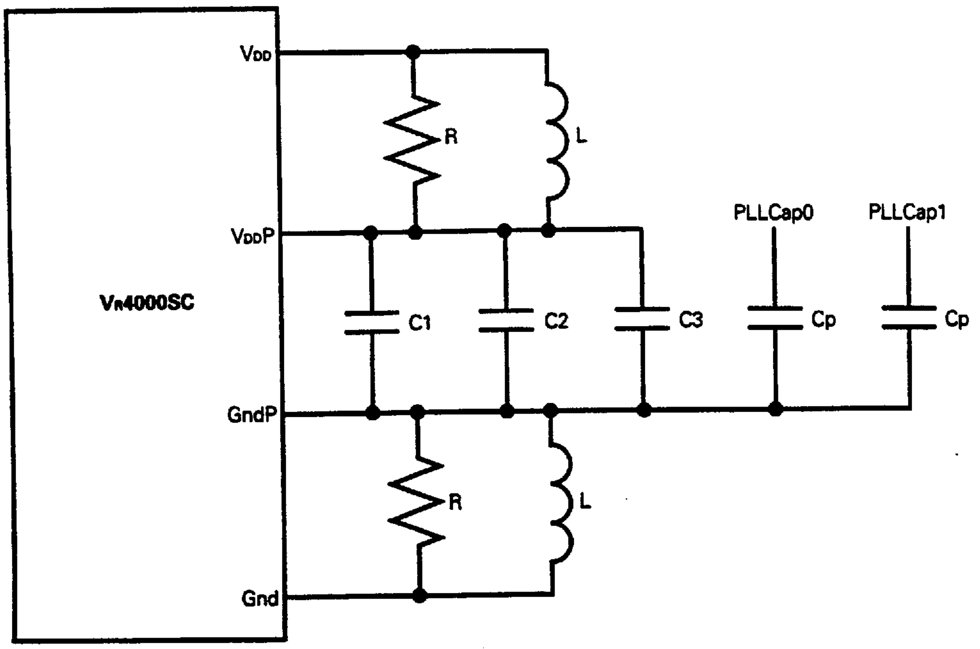
The Phase Locked Loops require several passive components for their operation. These passive components are attached to the PLLCap0, PLLCap1, $V_{DD}P$, and GndP pins and are illustrated below. The capacitors for the PLLCap0 and PLLCap1 pins are connected in the figure to GndP but they could also be connected to $V_{DD}P$. Note that the capacitors connecting the PLLCap0 and PLLCap1 pins to either $V_{DD}P$ or GndP are to be provided externally.

A schematic diagram showing the passive components and their interconnect to the VR4000SC is illustrated in **Fig. 12-1 PLL Passive Components.**

It is essential to isolate the analog power and ground ($V_{DD}P$/GndP) from the other power and ground pins. Initial evaluations with R=5 Ω, C1=1 nF, C2=0.1 μF, C3=10 μF, and Cp=470 pF have yielded good results on test boards. Since the optimum values for the filter components depend on the application and the system noise environment, these values should be considered as starting points for further experimentation within the application specific context. In addition, the chocks (inductors) can be considered as an alternative to the resistors for power supply filtering.

Remark The test refers to the pins connecting PLLCap0 and PLLCap1 to pins. These pins are not externally available through pins on the package. The connections to the bond pads PLLCap0 and PLLCap1 are incorporated in the package.

Fig. 12-1 PLL Passive Components



Remark 1. The inductors may be used to replace the resistors in the filter circuit. They may be also excluded, used in parallel with the resistors or used in place of the resistors. Since noise filtering is application specific, inductors may improve noise reduction in some applications.

2. Since noise filtering is, in part, application specific, the values given for the filter elements are only suggested values that may be changed depending on the application. The suggested starting value for the inductor is 1000 μH.

CHAPTER 13 JTAG INTERFACE

The VR4000SC processor provides a boundary scan interface using the industry standard JTAG protocol.

13.1 JTAG Interface Signal Summary

JTDI:	(i)	JTAG serial data in.
JTDO:	(o)	JTAG serial data out.
JTMS:	(i)	JTAG command signal.
JTCK:	(i)	JTAG serial clock input.

13.2 JTAG Functionality

The JTAG boundary scan mechanism is intended to provide a capability for testing the interconnect between the VR4000SC processor, the printed circuit board to which it is attached, and the other components on the board. In addition the JTAG boundary scan mechanism is intended to provide a rudimentary capability for low speed logical testing of the secondary cache RAMs. The JTAG boundary scan mechanism is not intended to provide any capability for testing the VR4000SC processor itself.

In accordance with the JTAG specification the VR4000SC processor contains a TAP controller, JTAG instruction register, JTAG boundary scan register, and JTAG bypass register. However, the VR4000SC JTAG implementation provides only the **external test** functionality of the boundary scan register.

13.2.1 JTAG Test Access Port (TAP)

The JTAG Test Access Port consists of the 4 pins described above. Data is serially scanned into one of the three registers (Instruction register, Bypass register, Boundary Scan register) from the JTDI pin, and is scanned out from the selected one of these registers onto the JTDO pin. The JTDI input feeds the LSB of the selected register, and the MSB of the selected register appears on the JTDO output. The JTMS input controls the state transitions of the main TAP controller state machine.

Data on the JTDI and JTMS pins is sampled on the rising edge of the JTCK input clock signal. Data on the JTDO pin changes on the falling edge of the JTCK clock signal.

13.2.2 JTAG TAP Controller

The VR4000SC implements the 16-state JTAG TAP controller as defined in the IEEE JTAG specification.

The TAP controller state machine can be put in its Reset state in one of two ways. Deassertion of the VDDOk input will reset the TAP controller. Keeping the JTMS input signal asserted through five consecutive rising edges of the JTCK clock input will also send the TAP controller state machine into its Reset state. In either case, keeping JTMS asserted will maintain the Reset state.

13.2.3 Instruction Register

The VR4000SC's JTAG instruction register is three bits wide and is encoded as follows.

MSB...LSB	Selected Data Register
0 0 0	Boundary Scan register (External Test only)
x x 1	Bypass Register
x 1 x	Bypass Register
1 x x	Bypass Register

The Instruction register is composed of two stages – the shift register stage and the parallel output latch. When the TAP controller is in the Reset state, the value 7 (111) is loaded into the parallel output latch, thus selecting the Bypass register as the default. When the TAP controller is in the Capture-IR state, the value 4 (100) is loaded into the shift register stage. When the TAP controller is in the Shift-IR state, data is serially shifted into the shift register stage of the Instruction register from the JTDI input pin, and the MSB of the Instruction register's shift register stage is shifted out onto the JTDO pin. When the TAP controller is in the Update-IR state, the current data in the shift register stage is loaded into the parallel output latch.

13.2.4 Bypass Register

The Bypass Register is one bit wide. When the TAP controller is in the Shift-DR (Bypass) state, the data on the JTDI pin is shifted into the bypass register, and the bypass register's output is shifted out onto the JTDO output pin.

13.2.5 Boundary Scan Register

The Boundary Scan register is 319 bits wide. The three most significant bits control the output enables on the various bidirectional buses. The most significant bit is the JTAG output enable bit for the SysAD, SysADC, SysCmd and SysCmdP buses. The next most significant bit is the JTAG output enable for the SCData and SCDChk buses. The third most significant bit is the JTAG output enable for the SCTag and SCTChk buses. The remaining 316 bits correspond to 316 signal pads of the VR4000SC. The scan order of these bits is listed in **APPENDIX C JTAG ORDERING** at the end of this document.

When the TAP controller is in the Reset state, the three most significant bits of the Boundary Scan register are set to "0" (the default JTAG output enable control on all the bidirectional pins is to disable the outputs). When the TAP controller is in the Capture-DR (Boundary Scan) state, the data currently present on all the VR4000SC's input and I/O pins are latched into the Boundary Scan register. The Boundary Scan register bits corresponding to output pins are arbitrary in this state and must not be checked during the scan out process. When the TAP controller is in the Shift-DR (Boundary Scan) state, data is serially shifted into the Boundary Scan register from the JTDI pin, and the contents of the Boundary Scan register are shifted out onto the JTDO pin. When the TAP controller is in the Update-DR (Boundary Scan) state, the current data in the Boundary Scan register is latched into its parallel output latch, and the bits corresponding to output pins and those IO pins whose outputs are enabled (by the three MSBs of the Boundary Scan register) are enabled onto the VR4000SC's pins.

13.3 Implementation Specific Details

- The MasterClock, MasterOut, SyncIn and SyncOut pads do not have JTAG.
- Some pairs of output pads share a single JTAG bit. These are: SCAddr0W and SCAddr0X, SCAddr0Y and SCAddr0Z, SCWrBW and SCWrBX, SCWrBY and SCWrBZ, TClock[0] and TClock[1], RClock[0] and RClock[1].
- All input pads data are first latched into a Processor Clock based register in the pad cell before they are captured into the Boundary Scan register in the Capture-DR (Boundary Scan) state. When the Phase locked loop is disabled, the processor clock is half the frequency of MasterClock. Therefore the data setup required at the input pads is greater than two MasterClock periods before the rising edge of JTCK when the TAP controller is in the Capture-DR (Boundary Scan) state.
- The output enable controls generated from the three most significant bits of the Boundary Scan register are latched into a Processor Clock based register before they actually enable the data onto the pads. Therefore the delay from the rising edge of JTCK in the Update-DR (Boundary Scan) state to data valid at the output pins of the chip is greater than two MasterClock periods.

Secondary cache interface pins available only on the large package configuration:

SCData(127:0):	(i/o)	A 128-bit bus used to read or write cache data from/to the secondary cache.
SCDChk(15:0):	(i/o)	A 16-bit bus which conveys two ECC fields that cover the upper or lower 64 bits of the SCData from/to the secondary cache.
SCTag(24:0):	(i/o)	A 25-bit bus used to read or write cache tags from/to the secondary cache.
SCTChk(6:0):	(i/o)	A 7-bit bus which conveys an ECC field that covers the **SCTag** from/to the secondary cache.
SCAddr(17:1):	(o)	A 17-bit bus which addresses the secondary cache.
SCAddr0Z:	(o)	Bit 0 of the secondary cache address.
SCAddr0Y:	(o)	Bit 0 of the secondary cache address.
SCAddr0X:	(o)	Bit 0 of the secondary cache address.
SCAddr0W:	(o)	Bit 0 of the secondary cache address.
SCAPar(2:0):	(o)	A 3-bit bus which conveys the parity of the **SCAddr** bus and the cache control lines $\overline{\text{SCOE}}$, $\overline{\text{SCWR}}$, $\overline{\text{SCDCS}}$ and $\overline{\text{SCTCS}}$.
$\overline{\text{SCOE}}$:	(o)	A signal which enables the outputs of the secondary cache RAMs.
$\overline{\text{SCWrZ}}$:	(o)	Secondary cache write enable.
$\overline{\text{SCWrY}}$:	(o)	Secondary cache write enable.
$\overline{\text{SCWrX}}$:	(o)	Secondary cache write enable.
$\overline{\text{SCWrW}}$:	(o)	Secondary cache write enable.
$\overline{\text{SCDCS}}$:	(o)	A signal which enables the chip select pins of the secondary cache RAMs associated with **SCData** and **SCDChk**.
$\overline{\text{SCTCS}}$:	(o)	A signal which enables the chip select pins of the secondary cache RAMs associated with **SCTag** and **SCTChk**.

System interface pins available on both package configurations:

SysAD(63:0):	(i/o)	A 64-bit bus used for address and data transmission between the processor and an external agent.
SysADC(7:0):	(i/o)	An 8-bit bus containing check bits for the SysAD bus.
SysCmd(8:0):	(i/o)	A 9-bit bus used for command and data identifier transmission between the processor and an external agent.
SysCmdP:	(i/o)	A single even parity bit over the SysCmd bus.
$\overline{\text{ValidIn}}$:	(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.
$\overline{\text{ValidOut}}$:	(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.
$\overline{\text{ExtRqst}}$:	(i)	Signals that the system interface needs to submit an external request.
$\overline{\text{Release}}$:	(o)	Signals that the processor is releasing the system interface to slave state.

$\overline{\text{RdRdy}}$:	(i)	Signals that an external agent is capable of accepting a processor read, invalidate, or update request in both non-overlap and overlap mode or a read followed by a potential invalidate or update request in overlap mode.
$\overline{\text{WrRdy}}$:	(i)	Signals that an external agent is capable of accepting a processor write request in both non-overlap and overlap mode.

Interrupt pin available on both package configurations:

$\overline{\text{Int(0)}}$:	(i)	One of six general processor interrupts, bit-wise ORed with bit 0 of the interrupt register.

Non-maskable interrupt pin available on both package configurations:.

$\overline{\text{NMI}}$:	(i)	Non-maskable interrupt, ORed with bit 6 of the interrupt register.

Boot time mode control interface pins available on both package configurations:

ModeIn:	(i)	Serial boot mode data in.
ModeClock:	(o)	Serial boot mode data clock out at the system clock frequency divided by 256.

JTAG interface pins available on both package configurations:

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.

Maintenance pins available on both package configurations:

TClock(1:0):	(o)	Two identical transmit clocks at the operation frequency of the system interface.
RClock(1:0):	(o)	Two identical receive clocks at the operation frequency of the system interface.
MasterClock:	(i)	Master clock input at the operation frequency of the processor.
MasterOut:	(o)	Master clock output aligned with MasterClock.
SyncOut:	(o)	Synchronization clock output.
SyncIn:	(i)	Synchronization clock input.
IOOut:	(o)	Output slew rate control feedback loop output. Must be connected to IOIn through a delay loop that models the IO path from the VR4000SC to an external agent.
IOIn:	(i)	Output slew rate control feedback loop input.
VDDOk:	(i)	When asserted, this signal indicates to the VR4000SC that the +5 volt power supply has been above 4.75 volts for more than 100 milliseconds and will remain stable. The assertion of VDDOk will initiate the reading of the boot time mode control serial stream.
$\overline{\text{ColdReset}}$:	(i)	This signal must be asserted for a power on reset or a cold reset. The clocks SClock, TClock, and RClock begin to cycle and are synchronized with the deassertion edge of $\overline{\text{ColdReset}}$. $\overline{\text{ColdReset}}$ must be de-asserted synchronously with MasterClock.
$\overline{\text{Reset}}$:	(i)	This signal must be asserted for any reset sequence. It may be asserted synchronously or asynchronously for a cold reset, or synchronously to initiate a warm reset. $\overline{\text{Reset}}$ must be de-asserted synchronously with MasterClock.
$\overline{\text{GrpRun}}$:	(o)	Run pulse generated after a group of instructions completes.
$\overline{\text{GrpStall}}$:	(i)	Stall signal which will stall the processor after the current group of instructions completes.
$\overline{\text{Fault}}$:	(o)	Mismatch output of boundary comparators.

VDDP:	(i)	Quiet VDD for the internal phase locked loop.
GndP:	(i)	Quiet Gnd for the internal phase locked loop.

Maintenance pins available only on the large package configuration:

Status(7:0):	(o)	An 8-bit bus that indicates the current operation status of the processor.
VDDSense:	(i/o)	This is a special pin used only in component testing and characterization. It provides a separate, direct connection from the on-chip VDD node to a package pin without attaching to the in-package power planes. Test fixtures treat VDDSense as an analog output pin; the voltage at this pin directly shows the behavior of the on-chip VDD. Thus, characterization engineers can easily observe the effects of di/dt noise, transmission line reflections, etc. VDDSense should be connected to VDD in functional system designs.
GndSense:	(i/o)	GndSense provides a separate, direct connection from the on-chip Gnd node to a package pin without attaching to the in-package ground planes. GndSense should be connected to Gnd in functional system designs.

APPENDIX A SUB-BLOCK ORDERING

Sub-block ordering is an order for transmitting the data elements that form a block of data when the data element transmitted first is not the data element at the beginning of the block. Sub-block ordering causes the data elements of the block to be transmitted in an order that fills out sub-blocks of increasing size. For the VR4000SC, the smallest data element of a block transfer is a double word, therefore, the double word at the target address is transferred first, followed by the double word that fills out the quad word that contains the starting double word. Next the quad word that fills out an octal word containing the starting quad word is transferred in the same order as the first quad word, followed by the octal word that fills out a hex word containing the starting octal word in the same order as the first octal word, and so on through sub-blocks of increasing size until the entire block has been transferred.

Perhaps an easier way to consider sub-block ordering is to look at a method for generating the addresses, within the block, of the double words to be transferred for sub-block ordering. A simple method for generating such addresses is to bit-wise XOr the starting double word address with the output of a binary counter that is counting the double words in the block starting at double word zero.

The following tables illustrate the sequence of double words transferred using sub-block ordering for a thirty-two word block based on three different starting block addresses. For these illustrations the double words in the block will be identified by their block addresses. The block address for each double word in a block is derived by numbering the double words in the block sequentially starting with zero.

The tables also include a binary count of the double words in the block to illustrate the XOr relationship between this count, the starting address and the block addresses of the double words transferred.

Table A-1 Sequence of Double Words Transferred Using Sub-block Ordering (a)

Cycle	Starting block address	Binary count	Double word transferred
1	0010	0000	0010
2	0010	0001	0011
3	0010	0010	0000
4	0010	0011	0001
5	0010	0100	0110
6	0010	0101	0111
7	0010	0110	0100
8	0010	0111	0101
9	0010	1000	1010
10	0010	1001	1011
11	0010	1010	1000
12	0010	1011	1001
13	0010	1100	1110
14	0010	1101	1111
15	0010	1110	1100
16	0010	1111	1101

A

Table A-2 Sequence of Double Words Transferred Using Sub-block Ordering (b)

Cycle	Starting block address	Binary count	Double word transferred
1	1011	0000	1011
2	1011	0001	1010
3	1011	0010	1001
4	1011	0011	1000
5	1011	0100	1111
6	1011	0101	1110
7	1011	0110	1101
8	1011	0111	1100
9	1011	1000	0011
10	1011	1001	0010
11	1011	1010	0001
12	1011	1011	0000
13	1011	1100	0111
14	1011	1101	0110
15	1011	1110	0101
16	1011	1111	0100

Table A-3 Sequence of Double Words Transferred Using Sub-block Ordering (c)

Cycle	Starting block address	Binary count	Double word transferred
1	0101	0000	0101
2	0101	0001	0100
3	0101	0010	0111
4	0101	0011	0110
5	0101	0100	0001
6	0101	0101	0000
7	0101	0110	0011
8	0101	0111	0010
9	0101	1000	1101
10	0101	1001	1100
11	0101	1010	1111
12	0101	1011	1110
13	0101	1100	1001
14	0101	1101	1000
15	0101	1110	1011
16	0101	1111	1010

APPENDIX B EVEN PARITY

The descriptions of parity as even or odd often have different meanings to different people. Even parity is defined for the VR4000SC as follows:

For a field of n bits protected by a single parity bit, if the number of ones among the n data bits is an even number, then the even parity bit will be a 0. If the number of ones among the n data bits is an odd number, then the even parity bit will be a 1. For example, if all n of the data bits are 0, the parity bit will also be a 0 if there are no data bits in error. If all n of the data bits are 1, and the number of data bits n is an odd number, then the parity bit will be a 1 if there are no data bits in error.

APPENDIX C JTAG ORDERING

The following list is the order of the pins associated with the JTAG Boundary Scan Register starting from JTDI and ending at JTDO.

SCDChk[13]
SysADC[1]
SCDChk[1]
SysADC[5]
SCDChk[5]
Status[0]
Status[1]
Status[2]
Status[3]
$\overline{\text{IvdErr}}$
Status[4]
$\overline{\text{IvdAck}}$
Status[5]
Status[6]
Status[7]
SCDChk[7]
SysADC[7]
SCDChk[3]
SysADC[3]
SCDChk[15]
V_{DD}Ok
SCTag[16]
SCDChk[11]
SCData[63]
SysAD[63]
SCData[31]
SysAD[31]
SCData[127]
SCTag[17]
SCData[95]
SCData[62]
SysAD[62]
SCData[30]
SysAD[30]
SCData[126]
SCTag[18]
SCData[94]
RClock[1..0] (share the same JTAG bit)
SCTag[19]
SCData[61]
SysAD[61]
SCData[29]
SysAD[29]
SCData[125]
$\overline{\text{Reset}}$
SCTag[20]
SCData[93]
SCData[60]
SysAD[60]
SCData[28]
SysAD[28]
SCData[124]
$\overline{\text{ColdReset}}$
SCTag[21]
SCData[92]
SCData[59]
SysAD[59]
SCData[27]
SysAD[27]
SCData[123]
IOIn
SCTag[22]
SCData[91]
SCData[58]
SysAD[58]
SCData[26]
SysAD[26]
SCData[122]
IOOut
SCTag[23]
SCData[90]
SCData[57]
SysAD[57]
SCData[25]
SysAD[25]
SCData[121]
$\overline{\text{GrpRun}}$
SCTag[24]
SCData[89]
SCData[56]
SysAD[56]
SCData[24]
SysAD[24]
SCData[120]

$\overline{\text{GrpStall}}$
SCTChk[0]
SCData[88]
SCDChk[6]
SysADC[6]
SCDChk[2]
SysADC[2]
SCDChk[14]
$\overline{\text{NMI}}$
SCTChk[1]
SCDChk[10]
SCData[55]
SysAD[55]
SCData[23]
SysAD[23]
SCData[119]
$\overline{\text{Release}}$
SCTChk[2]
SCData[87]
SCData[54]
SysAD[54]
SysAD[22]
ModeIn
SCData[22]
$\overline{\text{RdRdy}}$
SCData[118]
SCData[86]
SCData[53]
SysAD[53]
SCData[21]
SysAD[21]
SCData[117]
$\overline{\text{ExtRqst}}$
SCTChk[3]
SCData[85]
SCData[52]
SysAD[52]
SCData[20]
SysAD[20]
SCData[116]
$\overline{\text{ValidOut}}$
SCTChk[4]
SCData[84]
SCData[51]
SysAD[51]
SCData[19]
SysAD[19]
SCData[115]
$\overline{\text{ValidIn}}$
SCTChk[5]
SCData[83]
SCAddr0W,X (share the same JTAG bit)
SCAddr0Y,Z (share the same JTAG bit)
SCAddr[[1]
SCData[50]
SysAD[50]
SCData[18]
SysAD[18]
SCData[114]
$\overline{\text{Int[0]}}$
SCTCHk[6]
SCData[82]
SCData[49]
SysAD[49]
SCData[17]
SysAD[17]
SCData[113]
SCAddr[2]/$\overline{\text{Int[1]}}$
SCAddr[3]
SCData[81]
SCData[48]
SysAD[48]
SCData[16]
SysAD[16]
SCData[112]
SCAddr[4]/$\overline{\text{Int[2]}}$
SCAddr[5]
SCData[80]
SCAddr[6]
SCAddr[7]
SCAddr[8]
SCAddr[9]
SCAddr[10]
SCAddr[11]
SC64Addr
SCAddr[12]
SCAddr[13]
SCAddr[14]
SCAddr[15]
SCAddr[16]
SCAddr[17]
SCData[64]
SCAPar[0]
SCAPar[1]/$\overline{\text{Int[3]}}$
SCData[96]
SysAD[0]

SCData[0]
SysAD[32]
SCData[32]
SCData[65]
SCAPar[2]
$\overline{\text{SCOE}}$/$\overline{\text{Int}}$[4]
SCData[97]
SysAD[1]
SCData[1]
SysAD[33]
SCData[33]
SCData[66]
$\overline{\text{SCDCS}}$
$\overline{\text{SCTCS}}$/$\overline{\text{Int}}$[5]
SCData[98]
SysAD[2]
SCData[2]
SysAD[34]
SCData[34]
SCTag[0]
$\overline{\text{SCWrW,X}}$ (share the same JTAG bit)
$\overline{\text{SCWrY,Z}}$ (share the same JTAG bit)
SCData[67]
SCTag[1]
SysCmd[0]
SCData[99]
SysAD[3]
SCData[3]
SysAD[35]
SCData[35]
SCData[68]
SCTag[2]
SysCmd[1]
SCData[100]
SysAD[4]
SCData[4]
SysAD[36]
SCData[36]
SCData[69]
SCTag[3]
SysCmd[2]
SCData[101]
SysAD[5]
SCData[5]
SysAD[37]
SCData[37]
SCData[70]
$\overline{\text{WrRdy}}$

ModeClk
SCData[102]
SysAD[6]
SCData[6]
SysAD[38]
SCData[38]
SCData[71]
SCTag[4]
SysCmd[3]
SCData[103]
SysAD[7]
SCData[7]
SysAD[39]
SCData[39]
SCDChk[8]
SCTag[5]
SysCmd[4]
SCDChk[12]
SysADC[0]
SCDChk[0]
SysADC[4]
SCDChk[4]
SCData[72]
SCTag[6]
SysCmd[5]
SCData[104]
SysAD[8]
SCData[8]
SysAD[40]
SCData[40]
SCData[73]
SCTag[7]
SysCmd[6]
SCData[105]
SysAD[9]
SCData[9]
SysAD[41]
SCData[41]
SCData[74]
SCTag[8]
SysCmd[7]
SCData[106]
SysAD[10]
SCData[10]
SysAD[42]
SCData[42]
SCData[75]
SCTag[9]

SysCmd[8]
SCData[107]
SysAD[11]
SCData[11]
SysAD[43]
SCData[43]
SCData[76]
SCTag[10]
SysCmdP
SCData[108]
SysAD[12]
SCData[12]
SysAD[44]
SCData[44]
SCData[77]
SCTag[11]
$\overline{\text{Fault}}$
SCData[109]
SysAD[13]
SCData[13]
SysAD[45]
SCData[45]
SCTag[12]
TClock[1..0] (share the same JTAG bit)
SCData[78]
SCTag[13]
SCData[110]
SysAD[14]
SCData[14]
SysAD[46]
SCData[46]
SCData[79]
SCTag[14]
SCData[111]
SysAD[15]
SCData[15]
SysAD[47]
SCData[47]
SCDChk[9]
SCTag[15]
SCTag_OE (JTAG output enable control for SCTag and SCTChk buses)
SCData_OE (JTAG output enable control for SCData and SCDChk buses)
SysAD_OE (JTAG output enable control for SysAD, SysADC, SysCmd and SysCmdP buses)