# DESIGNING READ AND WRITE BUFFERS FOR THE R4000™ SYSTEM INTERFACE

APPLICATION NOTE AN-114

By Andrew Ng

## INTRODUCTION

This article describes the basic concepts behind designing with the IDT79R4000 System Interface. The System Interface connects the R4000 CPU to external memory and peripherals. Topics include: (1) what the basic read and write memory transactions look like, (2) the basic architecture for designing buffers and transceivers into the address and data bus paths, and (3) explains the convention of using single level read buffers and multi-level write buffers. The read and write buffers can obviously be implemented with custom FPGAs or ASICs. However, read and write buffers can also be easily implemented using off-the-shelf discrete logic FIFOs and pipelined registers. Thus to more clearly illustrate a read and write buffer implementation, brief discrete logic examples are given using the 18-bit IDT Double-Density FCT16823T register with clock enable, the 16-bit IDT 73200 multi-level pipeline register, and the 8-bit IDT73210 2-level/1-level pipelined registered transceiver.

## THE R4000 MICROPROCESSOR

The IDT79R4000 MIPS CPU brings high performance 64/32-bit computing to a single chip microprocessor and thus extends the family of R3000™ compatible parts from the lower cost 32-bit R3051™ CPU and R3081™ CPU/FPA. Benchmarks for R4000 systems show their performance to be from 35-54VUPS (VAX Units of Performance) and from 44-72 SPECmarks. Initial R4000 parts are being produced to run with an external 50MHz clock frequency and future parts with the same external bus interface are planned with larger primary caches and for frequencies over 75MHz. As shown in the block diagram in Figure 1, the R4000 has high performance in large part because of its superpipelined architecture which allows a 100MHz internal clock speed which is double the external clock speed. The R4000 also has an on-chip floating-point accelerator, on-chip write-back primary instruction and data caches, an optional writeback secondary cache interface, and on-chip memory management. The Reduced

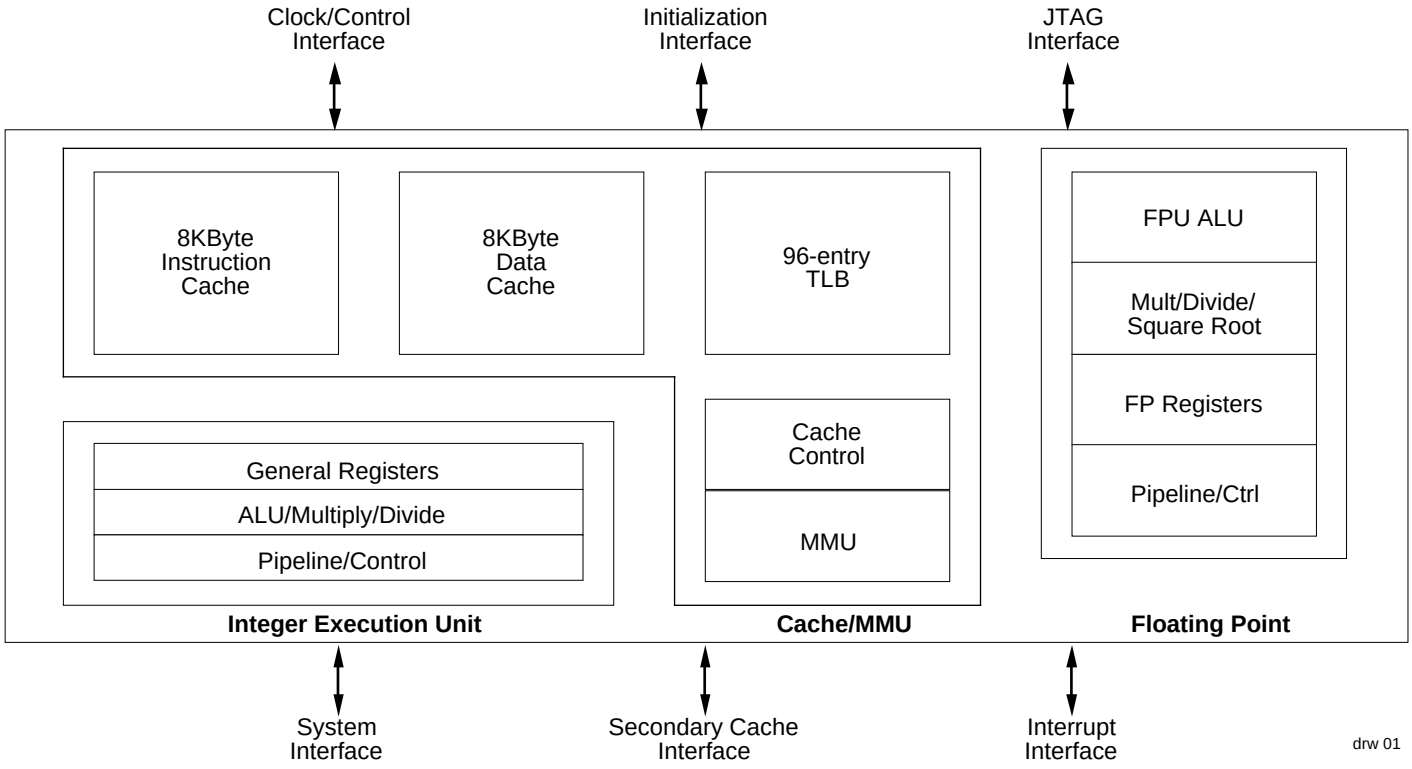


**Figure 1. Block Diagram of the R4000**

Instruction Set Architecture (RISC) and its development environment of optimized operating systems, compilers, and rescheduling assemblers place their emphasis on high performance and speed. The R4000 has 3 variants: (1) the 179-pin R4000PC which comes without a secondary cache interface, (2) the 447-pin R4000SC which comes with a secondary cache interface, and (3) the 447-pin R4000MC which comes with a secondary cache interface and also supports multiprocessing coherency.

### R4000 Clock Interface

One outstanding characteristic of the R4000 bus, in contrast to most microprocessors, is that it uses fully synchronous timing. Thus, every output is generated relative to a clock edge, and has the same propagation delay relative to the clock. Also, every input has the same setup and hold time relative to the clock.

This allows the simplification of worst case timing analysis, so that hardware designers can concentrate on functional issues. In conjunction with the fully synchronous timing, the R4000 has a PLL, which allows it to match the input clock, MasterIn to the master (MasterOut), processor (PClock), system(SClock), and transmit clock (TClock). MasterOut is an output clock which the PLL matches up to MasterIn. PClock is an internal clock which runs at twice the frequency of the MasterIn clock. SClock is also an internal clock which is essentially equivalent to TClock and runs at the same frequency as the MasterIn clock. The PLL also allows the alteration of the slew rate of the outputs relative to the clock and provides an extra receive clock that leads the system clock by 25%, called RClock as can be seen in Figure 2. The SyncIn and SyncOut pins shown in the Clock/Control Interface of Figure 3 automatically compensate the clocks for external buffer delays. Finally, options exist which allow the system, transmit, and receive clocks to be slowed down relative to the processor clock, such that the bus interface can run at 1/2, 1/3, or 1/4 of the normal speed. These options provide flexibility in producing setup, hold, and access times appropriate for various interfaces.

## R4000 SYSTEM INTERFACE

As shown in Figure 3, the R4000 System Interface consists of the signals that connect the CPU to the outside world of peripherals and memory. The System Interface has three major elements:

1. The 64-bit SysAD bus which carries the address and data.
2. The 9-bit SysCmd bus which encodes the type of memory cycle.
3. The control lines to condition the SysCmd bus and control the issue rates of the commands.

This article will discuss each of the System Interface elements in detail.

### R4000 SysAD Bus

The SysAD(63:0) Bus is 64-bits wide and has 8 additional optional ECC/parity bits called SysADC(7:0). The multiplexed SysAD bus is shared between address and data phases. The addresses will be present during the clock cycles where a valid interface command is present on the SysCmd bus. Data will be present for the clock cycles where a valid data identifier is present on the SysCmd bus. During the address phase, only the least significant 36-bits, SysAD(35:0) are used for a 64 GB physical address space. By convention, the upper 28 physical address bits, SysAD(63:36) are driven to 0 with appropriate ECC/parity by the CPU.

### R4000 SysCmd Bus

The SysCmd(8:0) bus is 9-bits wide and has 1 additional optional even parity bit called SysCmdP. The command bus encodes the type of transaction that is present on the system interface. For instance, block reads, block writes, single word reads, single byte writes, etc. are identified by the SysCmd encoding. The MSB (Most Significant Bit), SysCmd(8), indicates whether the cycle is a system interface command or data identifier. Thus SysCmd(8) breaks the encodings into two main cases, as listed in Tables 1, 2, and 3. Only the more common encodings are listed here, although a complete list is available in the User's Manual. Finally, some examples of the more typical 9-bit commands and data identifiers are given in Table 4.

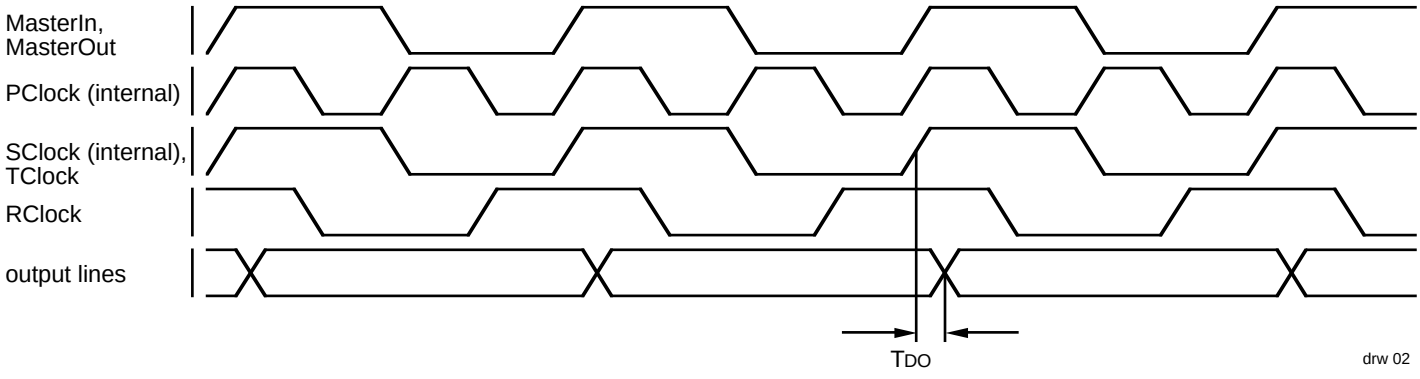


**Figure 2. R4000 Clock Interface Timing (PClock to SClock divisor of 2)**

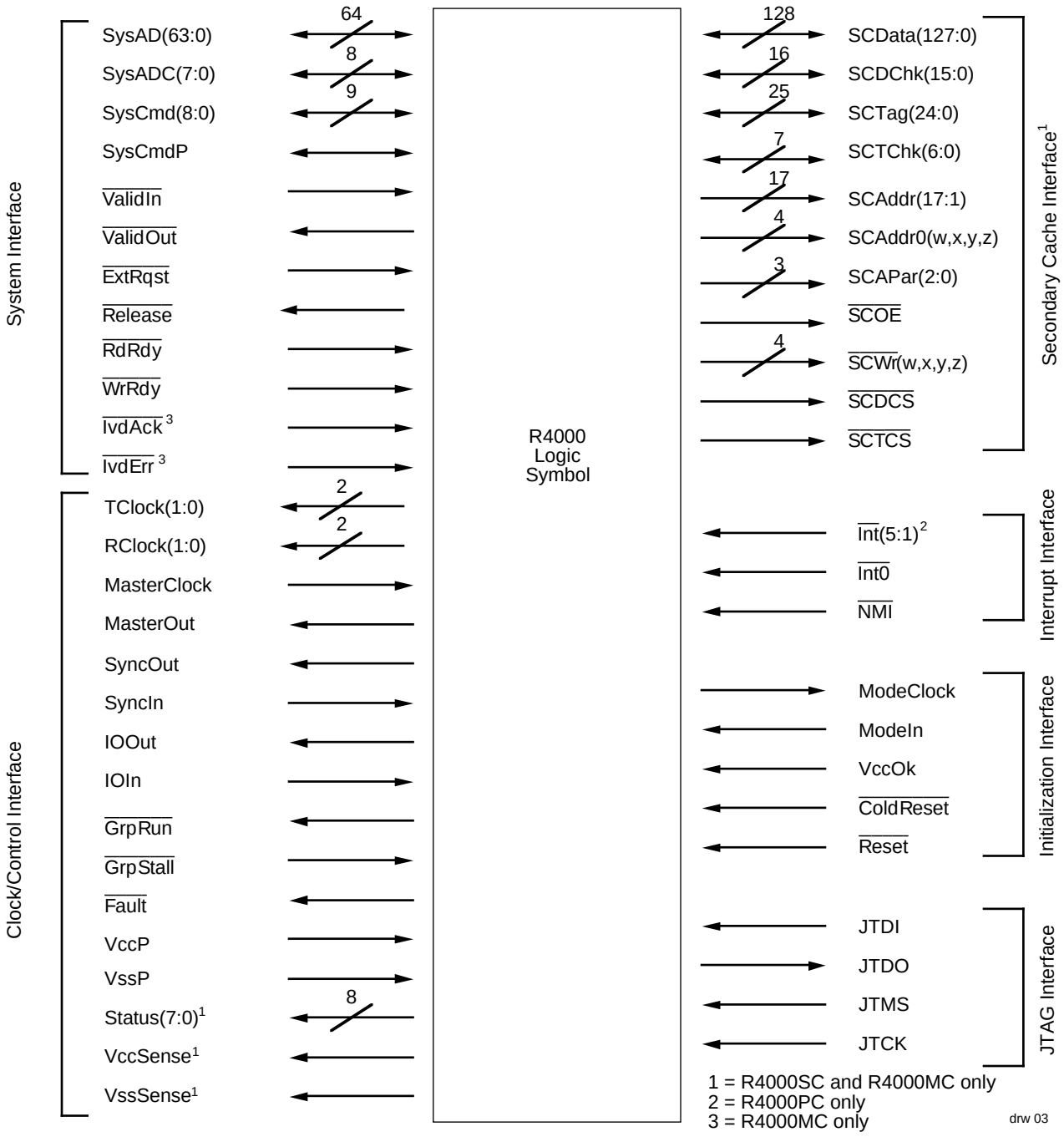


**Figure 3. The R4000 Interfaces**

### R4000 System Interface Control Signals

The System Interface Control Signals communicate when the System Interface busses are valid, and if the external agent, (i.e., the memory), is ready to accept the command. Their descriptions are given in Table 5. Two signals, the output ValidOut and the input ValidIn are used by the CPU and the memory to indicate when they are driving valid signals onto the SysCmd and SysAD busses. For example, when the CPU is driving a valid command/address or write data on the SysCmd bus, it will assert ValidOut, and when the memory system is returning a data identifier on the SysCmd bus and read data on the SysAD bus, it will assert ValidIn. Two input signals, RdRdy and WrRdy, are used by the memory system to communicate whether or not it is ready to handle the next read and write. The output signal Release is used by the CPU or bus master to indicate to the memory system that the master is tri-stating the bus on the next clock. After Release asserts, the memory system can drive the SysAD read data and SysCmd data identifier back to the CPU. The input signal ExtRqst is used by a DMA controller or interrupt controller to

gain control of the bus from the CPU. Finally, the inputs $\overline{\text{InvAck}}$ and $\overline{\text{InvErr}}$ are used only on the R4000MC version to help manage cache coherency.

To illustrate the use of the System Interface, the following sections will give an example for a read memory cycle and a write memory cycle. The sections follow the custom used in R3000/R4000 terminology, to use the term “buffer” in the software sense, meaning, a register location to store data rather than the hardware interpretation of amplifying or isolating a signal without storing it. In the following sections, hardware buffers such as the 8-bit IDT74FCT244T will always be referred to as a "hardware buffer".

| Encoding of SysCmd(8) Command or Data Identifier | |
|---|---|
| 0 | System Interface Command |
| 1 | System Interface Data |

**Table 1. SysCmd Encoding for SysCmd(8)**

| Encoding of SysCmd(7:5) Command | |
|---|---|
| 0 | Read Request |
| 1 | Read Request, Write Request Forthcoming (on the MC/SC only) |
| 2 | Write Request |
| | |
| Encoding of SysCmd(4:3) for Read and Write Requests attributes | |
| 2 | Noncoherent block read or write. |
| 3 | Double word, single word, or partial word read or write. |
| | |
| Encoding of SysCmd(1:0) for Noncoherent Block Read Requests or for Block Write Requests Block size | |
| 0 | Four words. |
| 1 | Eight words. |
| 2 | Sixteen words. |
| 3 | Thirty-two words. |
| | |
| Encoding of SysCmd(2:0) for Double Word, Word, or Partial Word Read Requests or Write Requests data size | |
| 0 | One byte valid (Byte). |
| 1 | Two bytes valid (Halfword). |
| 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). |

**Table 2. SysCmd Encodings for System Commands**

| 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, e.g., read data |
| 1 | Data is not response data, e.g., write data |
| SysCmd(5) Good data indication | |
| 0 | Data is error free. |
| 1 | Data is erroneous, e.g., a bus error |
| SysCmd(4) Data checking enable (on external agent data only) | |
| 0 | Check the data and check bits. |
| 1 | Don’t check the data and check bits. |
| SysCmd(3) Reserved | |
| SysCmd(2:0) Cache state (on R4000MC only). | |
| 0 | Invalid |
| 4 | Clean Exclusive. |
| 5 | Dirty Exclusive. |
| 6 | Shared. |
| 7 | Dirty Shared. |

**Table 3. SysCmd Encodings for Data Identifiers**

| SysCmd(8:0) 876543210 | Description of Command |
|---|---|
| 000010001 | Read request, Noncoherent block, eight words |
| 000011011 | Read request, Double word or smaller, four bytes valid |
| 001010001 | Write request, Block, eight words |
| 001011011 | Write request, Double word or smaller, four bytes valid |
| 110000100 | Read response data not end of block |
| 100000100 | Read response last data |
| 100100100 | Read response last data ignore ECC/parity |
| 111000101 | Write data not end of block |
| 101000101 | Write data, last data |

**Table 4. Examples of Typical SysCmd Commands and Data Identifiers**

## READ INTERFACE TRANSACTIONS

In Figure 3, the read interface state machine looks for ValidOut to assert along with one of the read commands as encoded by SysCmd(8:5). The SysCmd bus in the example is binary 000010001, which is an eight-word block read. Transactions involving a single double-word read are similar. By convention, the block size will either be the primary instruction cache or the primary data cache line size, or if present, the secondary cache line size. The SysAD bus contains the address for the transaction on the same clock as the read request command. The state machine should latch or register the address since the SysAD bus is multiplexed. Thus, each read transaction will only issue one start address even if it is a block read. If the state machine is not ready to handle the command, it should keep RdRdy de-asserted. RdRdy will delay the beginning of the read transaction by keeping the address on the bus. A caveat on RdRdy is that because it is synchronized to a clock edge, the CPU will not respond to it until 2 clock cycles later as shown in the example in Figure 4. The CPU asserts Release to indicate that the CPU is ready to tri-state the SysAD and SysCmd bus on the next clock cycle. The R4000 protocol allows Release to be either a variable number of clocks after ValidOut or possibly concurrent with ValidOut. Thus, the memory system must dedicate an extra state to allow for variable timed Releases. Along with Release, the memory system must also wait for any writes that are in progress to finish, since writes in R4000 systems are FIFOed (use First-In-First-Out buffering). After sampling Release and checking for on-going writes, the memory system can drive the bus and return data. In addition to the data, the memory system must drive a data identifier on the SysCmd bus and drive ValidIn to tell the CPU what it is returning. The memory system has direct control over the data return rate when it issues data identifiers. Some of the memory system return commands include data, end-of-data, and bus error (from Table 3, binary 110000100, 100000100, and 100100100, respectively).

| Pin Name | Type | Description |
|---|---|---|
| ValidOut | Output | Valid Output<br>Signals that the processor is now driving a valid address or data on the SysAD bus and a valid command or data identifier on the SysCmd bus. |
| ValidIn | Input | Valid Input<br>Signals that an external agent is now driving a valid address or data on the SysAD bus and a valid command or data identifier on the SysCmd bus. |
| RdRdy | Input | Read Ready<br>Signals that an external agent can now accept a processor read, invalidate, or update request in both non-overlap (non-secondary cache) and overlap (secondary cache) mode or can accept a read followed by a potential invalidate or update request in MC secondary cache overlap mode. |
| WrRdy | Input | Write Ready<br>Signals that an external agent can now accept a processor write request in both non-overlap (non-secondary cache) and secondary cache overlap mode. |
| Release | Output | Release interface<br>Signals that the processor is releasing the system interface to slave state. |
| ExtRqst | Input | External Request<br>Signals that the system interface needs to submit an external request. |
| IvdAck, IvdErr | Inputs | Invalidate Acknowledge and Invalidate Error<br>Signals on the R4000MC which indicate successful or unsuccessful completion of a processor invalidate or update request for cache coherency. Must be pulled high on other packages (SC). |

**Figure 5. R4000 System Interface Control Lines**

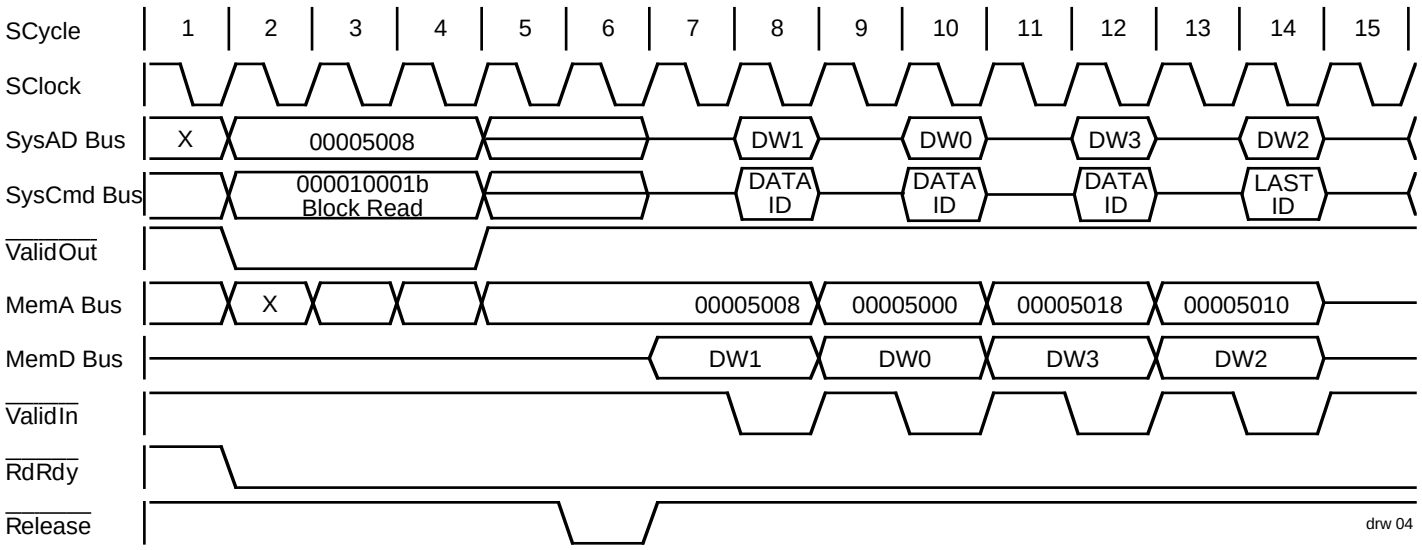


**Figure 4. R4000 Block Read Cycle**

On a block read, the state machine must increment the double word (8 bytes) LSB (Least Significant Bit) address bits of the block and keep returning more double words until the block is finished. These double-word LSB address bits are changed either in a sub-block order [hex (00,08,10,18,...), (08,00,18,10,...), (10,18,00,10,...), or (18,08,10,00,...)] or in a sequential wrap-around order [ hex (00,08,10,18,...), (08, 10, 18, 00), (10, 18, 00, 08), or (18, 00, 08, 18)] depending on the package type and boot-strap configuration. Sub-block ordering requires the original double-word start address to be XOR-ed with the block counter. Sub-block ordering is used to simplify the internal controls, since the word that is needed within a block, (e.g., the instruction), can always be returned in the same place. Sub-block ordering is required on the R4000PC and is optional on the R4000SC/MC.

Note that bus errors on block reads still require the memory system to return an end-of-data command to signal the end of the block, thus allowing the memory system to finish the rest of the block if it desires. Also, uncached memory, and especially I/O interfaces, can ignore ECC/parity generation/checking by using SysCmd(4) to indicate to the CPU that it doesn't want ECC/parity checked.

### The Sieve Search Algorithm

Because the address is generated on the same clock as the command and the $\overline{\text{ValidOut}}$ signal, the address register state machine usually has to implement a "door-to-door search algorithm". In the sieve algorithm, the address registers are enabled and constantly register new addresses on each clock. This means the registers are normally clocking in invalid addresses until the right one comes along. When $\overline{\text{ValidOut}}$ is detected, the address register should stop clocking and will hold the address until the end of the read or write. Thus, the address register is constantly searching for a valid address and incidentally latching in many of incorrect addresses until the correct one comes along.

### R4000 Read Buffer Size

To implement the read buffer, enough buffer locations must be present to store the incoming memory. For the R4000PC, which puts incoming main memory data directly into the primary cache, the maximum incoming memory read rate of 2 words per clock is matched by the CPU's capability to put these words into the primary cache. If a secondary cache is

| Secondary Cache Write Time | Memory Speed | Max. Buffer Levels Needed |
|---|---|---|
| 1–2 SCycles,1–4 PCycles | D | 1 |
| 3 SCycles, 5–6 PCycles | DDx | 1 |
| 4 SCycles, 7–8 PCycles | DDxx | 1 |

**Table 6. Examples of the Maximum Processor Read Data Rates for the MC/SC**

present, then enough time is needed to put the data/instruction into the secondary cache. For the R4000MC/SC, the secondary cache write rates may bandlimit the main memory read buffer if they are slower than main memory. However, this often is not a realistic case, since one of the purposes of the secondary cache is to provide a faster access time than main memory, in addition to isolating microprocessing systems from one another. In Table 6, the number of SCycles (assuming a PClock divide by 2 divisor for SClock) is shown along with the equivalent number of processor PCycles, since the on-chip secondary cache interface uses PCycles to time the secondary cache. The memory speed of the external system is indicated with a D, which means one double word per clock, and possibly followed by one or more x's, which indicate idle clocks. Thus a DDxx pattern indicates 2 double words can be returned every four clocks. A case in which more than one level of read buffering may be desired is shown in the next section.

**Secondary Cache Overlap Mode**

Some complexity is added to the state machine and the interface. The R4000MC/SC (but not the PC) uses a secondary cache overlap mode along with regular reads and writes that can issue a read command, which, in turn, issues a write command between itself and the expected data. For example, when the read command is issued, the write address and the write data are issued, which must be handled or buffered by the memory system. Only then can the memory system return the data for the read. The purpose of the secondary cache overlap mode is to allow the memory interface to better utilize the read access time, if it chooses to do so. Therefore, a DRAM memory system could begin a $\overline{\text{RAS}}$ precharge for the read while buffering the write data, as an example.

Figure 5 displays an example of a secondary cache overlapped read and write. This example uses a 4-word block size. For the secondary cache overlap mode, the state machine should latch/register the read address and then buffer the write. It must also use a signal to indicate that the write has to be delayed until the memory system is done with the read. In these cases, the read buffer needs a set of address registers separate from the write address registers. Note that since secondary cache overlapped writes are caused by writeback misses, the MSBs corresponding to the Secondary Cache address (minus the block size LSB bits) will be the same for a secondary cache overlapped read and write. Even though most address bits must have separate read and write registers, the Secondary Cache block address bits only need one set of registers.

Additional complexity exists for the multiprocessing R4000MC version, such that a potential invalidate or update might come between the read and write portions of the cluster. Therefore, R4000MC interfaces may require an additional address latch/register for the LSB portion of the potential invalidate/update double word address.

In addition, since the $\overline{\text{Release}}$ is definitely delayed by the secondary cache overlapped write data, it is possible for very fast memory systems to want to begin to return data before the CPU can possibly accept it. In these cases, a cost/performance trade-off having more than 1 level of read buffering can be made.

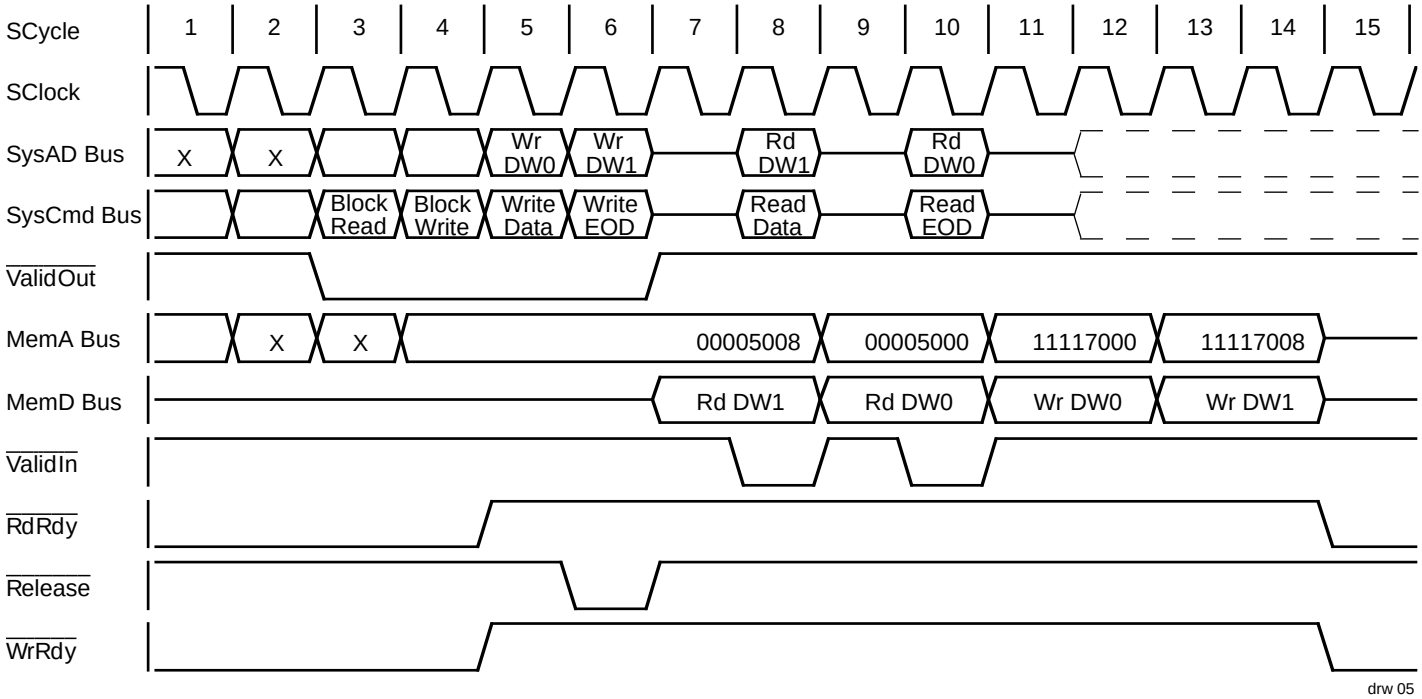


**Figure 5. Secondary Cache Overlap Timing**

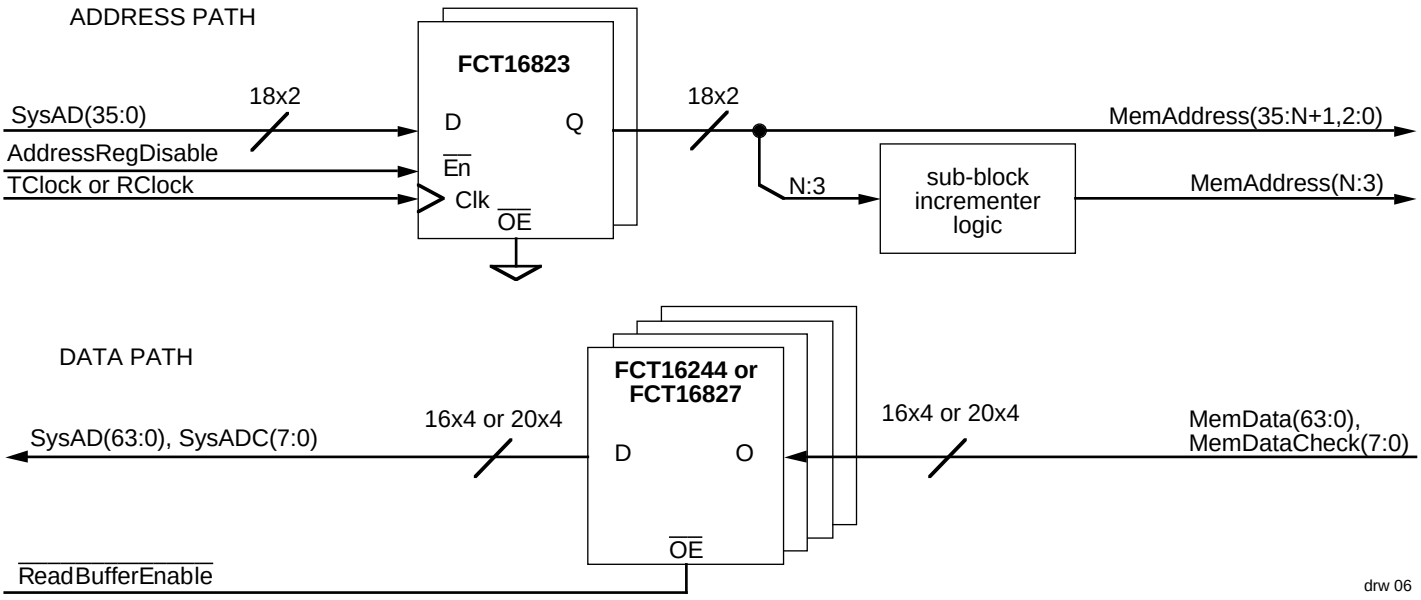


**Figure 6. R4000 Single Level Read Buffer**

## EXAMPLE OF AN R4000 READ BUFFER

The address latch/register for an R4000 memory interface can be built from parts such as the 18-bit FCT16823T register with clock enable. The critical parameter in the latch/register portion of the read interface is the latch's data hold time for the R4000 SysAD bus as shown in Figure 2. This can be solved two ways.

In the first method, the worst case hold time for a typical latch/register such as 16-bit FCT-T logic is 1.5ns which is added to the worst case clock skew from the R4000 is 0.5ns. The 2.0ns total of worst case factors is just met by the 3.5ns minimum data propagation delay ($T_{DO}$) of the R4000. If additional margin is needed — for instance if external clock buffering has additional clock skew — then the following can be done: The characteristic hold time for high-speed CEMOS™ 16-bit FCT-T logic is typically 0ns or less, especially at low temperature. Also, the 3.5ns minimum data propagation timing of the high-speed CEMOS R4000 outputs, which only occurs at low temperature, can be guaranteed to be indirectly delayed upto an additional 2.5ns by changing the slew rate of the outputs. The rise and fall slew rates can be adjusted by programming the serial boot initialization register interface at reset time. By using slower slew rates, which change the rise and fall times and, therefore slightly delay the outputs of the R4000, enough data hold time can be provided to memory interface latches/registers, even when considerable clock skew is taken into consideration.

A second method for providing additional hold time, especially for interfaces made from ASICs and FPGAs, is to use the RClock, as previously shown in Figure 2. The RClock leads the TClock by 25% of the TClock and therefore, at 50MHz provides 5ns of additional hold time. The disadvantage of using the RClock is either the latches/registers must be immediately staged with a set of TClock latches/registers and/or very fast control logic for the clock enable (which typically is TClock based) must be used.

Since the memory system access time is usually equal or greater than the secondary cache access time for the MC/SC systems and the PC systems can handle data as fast as the main memory system can return it, a simple hardware buffer is all that is needed for the data path, such as the 16-bit IDT FCT16244T or 8-bit FCT244T as shown in Figures 6 and 7. Alternatively, a pipeline register with clock enable, such as the 18-bit FCT16823T, could be used for the data path.

In systems with interrupt or external invalidate controllers, if the controller is isolated from the SysAD bus and on the memory side of the system interface, then the address registers may need to be bi-directional. An example of bi-directional registered transceivers with data clock enables is the 16-bit FCT16952T and the 8-bit 74FCT52T.

## R4000 WRITE INTERFACE TRANSACTIONS

A typical write sequence is shown in Figure 8. The write interface state machine looks for $\overline{\text{ValidOut}}$ to assert along with one of the write commands, as encoded by SysCmd(8:5) in Tables 1-3. The SysCmd bus in the example is binary 001010001, which is an eight-word block write. Single double-word transactions are similar. If the state machine is not ready to handle the command, it should keep $\overline{\text{WrRdy}}$ de-asserted. The caveat on using $\overline{\text{WrRdy}}$ is that because it is synchronized to a clock edge, the CPU will not respond to it until 2 clock cycles later. Thus, when $\overline{\text{WrRdy}}$ asserts, the address and $\overline{\text{ValidOut}}$ will remain on the bus for 2 more clocks. The SysAD bus contains the base address for that transaction on the same clock as the write request command. Block writes always increment the address sequentially, i.e., hex (00,08,10,18,...). The state machine should latch or register the address, since the SysAD bus is multiplexed. Each write transaction will only issue one start address, whether it is a single write or a block write, thus, external logic is needed to increment the base address for the memory system.

After the address is generated, $\overline{\text{ValidOut}}$ will be asserted

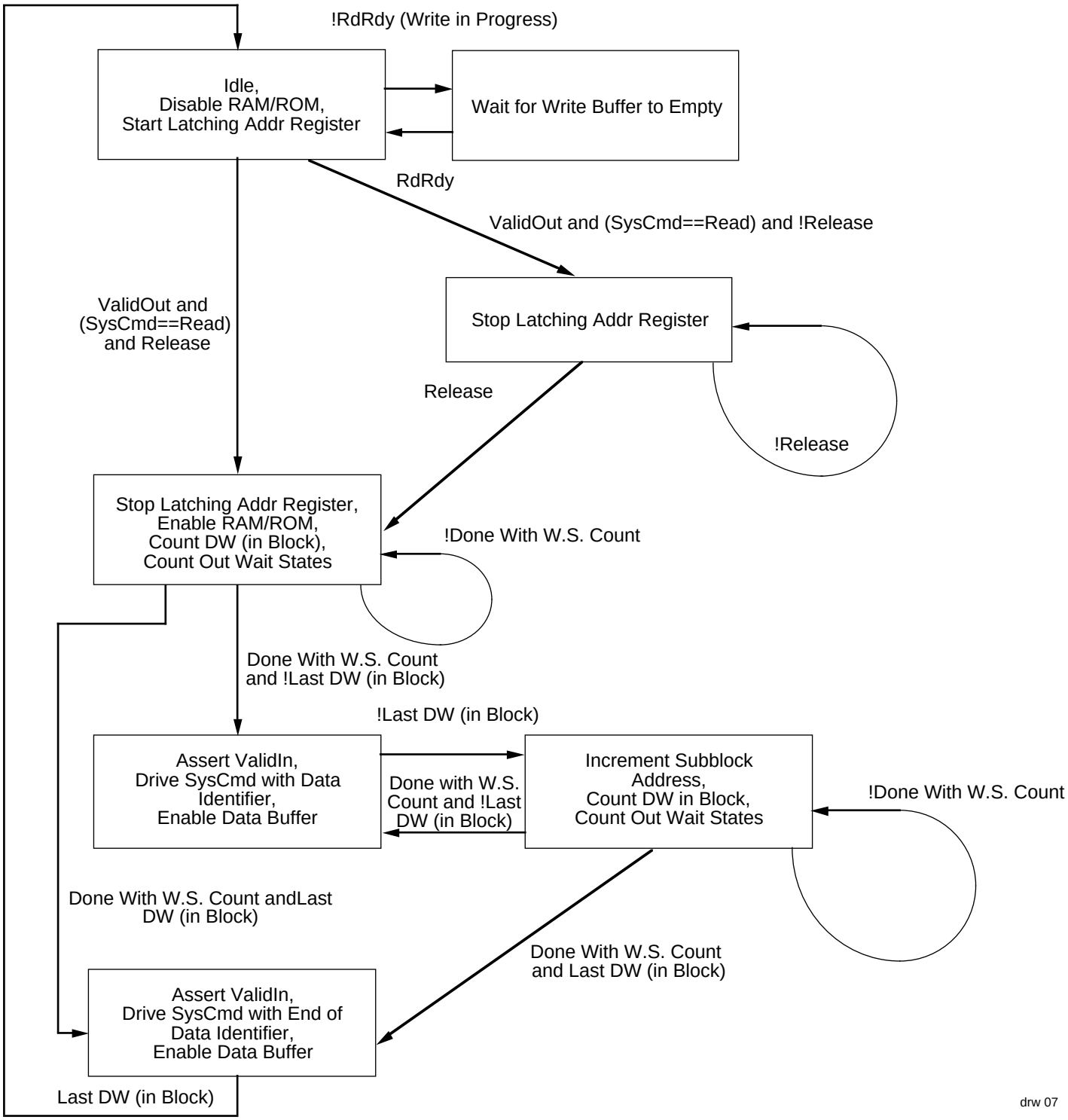


**Figure 7. R4000 Read Interface State Machine**

along with the first data of the write immediately, or a variable number of clocks later. The state machine must add a condition for the variable number of clocks between the address and the first data. If the data is a block write, the remaining data will be generated in a pattern selected by the initialization boot prom as shown in Table 7. In Table 7, Dxx means that a Data clock is followed by two idle clocks between each of the data items. However, no idle cycles are guaranteed after the last Data clock. Because the data rate pattern on writes is preselected at reset time using the serial boot initialization the register interface, the memory system cannot dynamically slow down any further and still control the data rate. Unless data can be written to memory at this preselected rate, the data must be buffered until the memory system can handle it.

Thus, data is written at a rate that further requires external buffering via a FIFO. The major reason for this arrangement is to allow memory writes from the buffer/FIFO to occur at the same time as cached reads. This allows the CPU to execute

cached instructions in parallel with the retiring of write data. In addition, the data caches use a writeback protocol, where data stores are always written to cache, but main memory is only updated when necessary (i.e., when another cache access needs to replace the cache location that is holding the freshly written data). Thus cached load and store fetches can also occur in parallel with the retiring of external system interface writes.

In contrast to reads, writes must indicate bus errors through an interrupt or some other external hardware mechanism. The CPU has an internal write buffer and also expects the memory system to have an additional external write buffer. Therefore, the CPU cannot match a bus error indication to a precise address and data pair, because it is decoupled from when the memory system actually tries the write. The system can choose to save address and data information with external hardware if it needs to match the error to the precise address and data within the write buffer. Uncached writes which are less than a double word wide, (e.g., 1byte), still produce data on the other bytes and the appropriate ECC/parity. However, the data for the unused bytes is pseudo-random, in that the CPU drives out what was last contained in an internal data buffer.

### R4000 Write Buffer Depth

In general, to implement the write buffer, enough buffer locations are needed to store all of the double words in the block write. However, as write data is being written into the buffer at the preselected data pattern rate, it is possible that the first few double words in the block write have been retired to main memory, much like a FIFO. Thus, theoretically, those buffer locations could be reused for the last few double words of the block write, as long as the buffer does not overflow. For memory which has predictable and consistent access time for each word (Static RAM) see Table 8. Not all data rate patterns and buffer sizes are shown, but the other cases can be derived using queuing theory producer/consumer model. Similar to block reads, the maximum block size is the largest primary or secondary cache-line size. For most systems, the control portion of the write buffer is simplified if the number of buffers matches the maximum block size.

DRAM systems complicate the optimal cases due to the first word possibly taking longer than the others because of $\overline{RAS}$ precharge, $\overline{RAS}$ address hold time, or because of the delay from a $\overline{CAS}$-before-$\overline{RAS}$ Refresh. In such cases, de-asserting $\overline{WrRdy}$ until the precharge or refresh is done and then choosing a slow enough data pattern rate to handle burst DRAM column page accesses prevents having to select a very deep buffer.

### Byte Enables

On the memory system logic, the 8-byte enables must be generated from the SysCmd and address for writes that are less than a double-word wide (from 1 to 7 bytes wide). Note that in contrast to most microprocessors, the R4000 will never generate an unaligned write. Thus, the 1 to 8 bytes written will always be contained within a double-word boundary. In addition, if only 1 to 4 bytes are written, they will always be contained within a word boundary. In other words, whenever 5 to 8 bytes are read or written little endian/ big endian, either the LSB/MSB must be at address offset 0 or the MSB/LSB must be at address offset 7, and whenever 1 to 4 bytes are read or written little endian/big endian, either the LSB/MSB must be at address offset 0 or the MSB/LSB must be at address offset 3.

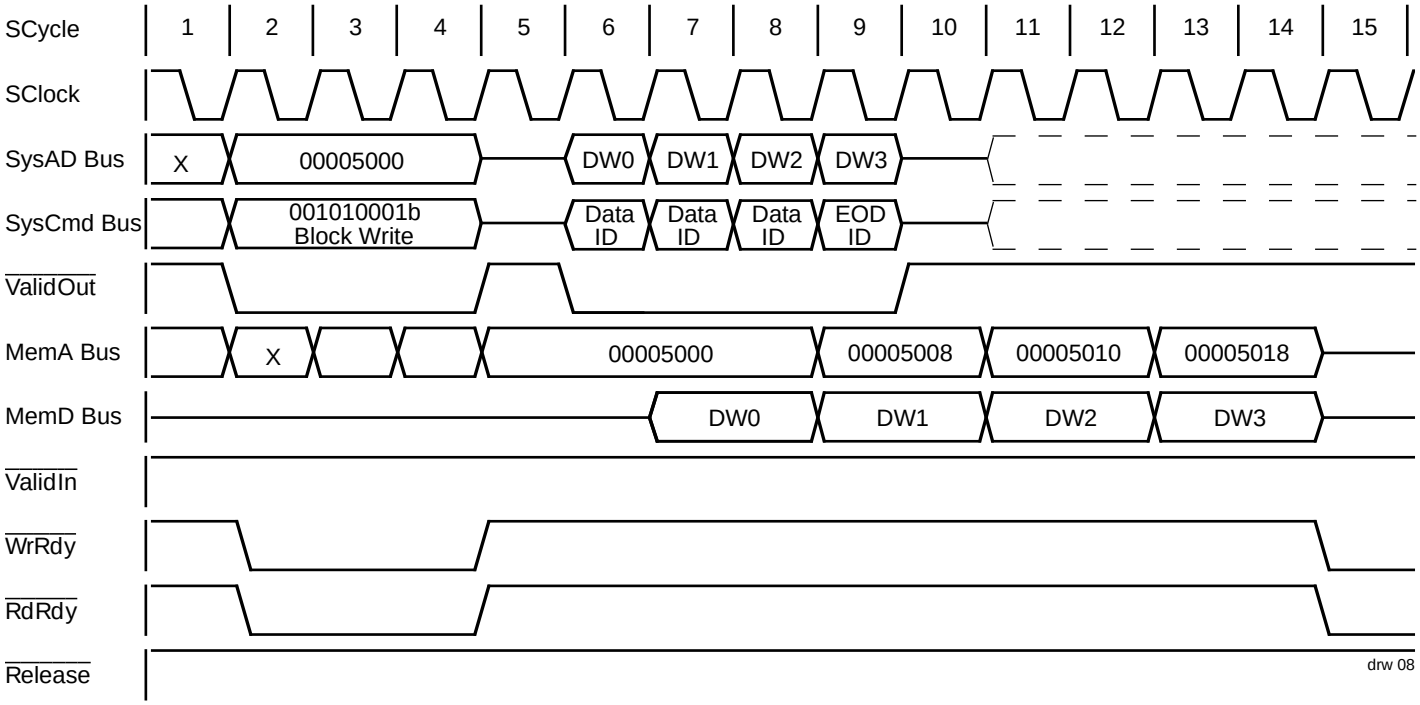


**Figure 8. R4000 Write Block Cycle**

| Serial Init Bits 14:11 | Data Rate Pattern |
|---|---|
| 0 | D |
| 1 | DDx |
| 2 | DDxx |
| 3 | DxDx |
| 4 | DDxxx |
| 5 | DDxxxx |
| 6 | DxxDxx |
| 7 | DDxxxxxx |
| 8 | DxxxDxxx |
| 9–15 | Reserved |

**Table 7. Possible Data Rate Patterns for Block Write**

| Cache Line Size | CPU Rate | Memory Speed | Max. Buffer Levels |
|---|---|---|---|
| 4 | DD | 1 clock | 1 |
| | | ≥ 2 | 2 |
| | DxD | ≤ 2 clocks | 1 |
| | | ≥ 3 | 2 |
| | DxxD | ≤ 3 | 1 |
| | | ≥ 4 | 2 |
| | DxxxD | ≤ 4 | 1 |
| | | ≥ 5 | 2 |
| 8 | DDDD | 1 clock | 1 |
| | | ≥ 3 | 4 |
| | DxDxDxD | ≤ 2 clocks | 1 |
| | | 2 | 3 |
| | | ≥ 3 | 4 |
| | DxxDxxDxxD | ≤ 3 | 1 |
| | | 4–5 | 2 |
| | | 6–11 | 3 |
| | | ≥ 12 | 4 |
| | DxxxDxxxDxxxD | ≤ 4 | 1 |
| | | 5–7 | 2 |
| | | 8–15 | 3 |
| | | ≥ 16 | 4 |
| 16 | DDDDDDDD | Max. Case | 8 |
| 32 | DDD...DDD | Max. Case | 16 |

**Table 8. Maximum Write Buffer Depth Needed For Various Cache Sizes**

For example, for a little endian system, a five-byte write or read, with bytes 0 through 4 enabled, could happen, but a five-byte write or read, with bytes 1 through 5 enabled, could never happen. A non-reduced PLA equation for one of the eight byte enables is shown in Table 8. The other seven byte enables are similar, and the equation can be simplified if the endianess is predetermined, or if it is known that the 64-bit mode won't be used. The re-alignment load/store–left/write instructions lwl, lwr, ldl, ldr, swl, swr, sdr, and sdw are used to develop the byte enable equations.

## EXAMPLE OF AN R4000 WRITE BUFFER

The address buffer for writes is similar to the address buffer for reads and can use the 18-bit FCT16823T. On the R4000PC which does not have secondary cache overlapped commands, the read address buffer can also be used for the write address buffer. The caveat is that $\overline{\text{RdRdy}}$ needs to be asserted during the write so that any potential reads will wait until the write is done with the address buffer before continuing. On the R4000MC/SC, separate registers are needed, as previously discussed, for the read address and the write address so that read, followed by write secondary cache overlap clusters, can be handled. The write address buffer needs to use the same door-to-door search algorithm to hold the address as the read address buffer. The primary difference between the two is that after latching/registering the address, the write buffer needs to increment the addresses for block writes sequentially instead of sub-block ordering. Similar to read, a write address register looks for a write SysCmd along with $\overline{\text{ValidOut}}$ before disabling the clock enable.

The write data buffer could consist of an ASIC or FPGA, however, the write buffer can also be easily implemented using discrete logic FIFOs or pipeline registers. An example is the IDT73200 pipeline register, 16-bits wide and 8 levels deep. It can either load a specific register slot through its instruction pins or automatically ripple data through, similar to a FIFO. Either method is acceptable with the R4000, because the block size is known at the beginning of the transaction. The block size will either be the primary cache line size or, if present, the secondary cache line size. If 16 or 32 locations are needed, then the IDT73200 can be expanded by using two or four in series in the ripple-through mode. Two separate state machines are needed, one for controlling the CPU-to-buffer interface and the other to control the buffer-to-memory interface. On the CPU side-state machine, block writes require the IDT73200 to start latching/registering in new data by incrementing the write pointer so a new register is selected to be written. On the last double-word of a block, the IDT73200 needs to be told when to stop latching new data, since re-pointing to the first location could possibly destroy that data too early. This can either be controlled with a special hold command on its instruction pins, I[3:0] = hex F, or by de-asserting the $\overline{\text{ClkEn}}$ pin after latching the last double word. The memory side needs to implement a state machine which checks to see if a read is in progress from a secondary cache overlapped read. Once ready, the state machine can initiate the write to the memory and select the register to output via the select pins. The logic to select the output register can also be used to generate the sequentially ordered least-significant double-word address bits. $\overline{\text{WrRdy}}$ can be de-asserted during a write to indicate that the buffer is full and to keep any subsequent writes from occurring until the IDT73200 (or a FIFO) can accept more data. A key control issue is to de-assert $\overline{\text{RdRdy}}$ while data is being written to memory, so that subsequent reads will wait for the memory bus to become free. Because $\overline{\text{RdRdy}}$ takes two clocks to react, the de-assertion must take place during the write command.

Other options include using a 4-deep pipelined register such as the 74FCT520, or a 2-deep pipelined register such as the IDT73210. An example using the IDT73210 will be given in the next section.

## EXAMPLE OF AN R4000 INTEGRATED READ AND WRITE BUFFER

Some systems, as shown in Table 7, can retire their writes at a fast enough rate to only require a 2-deep write buffer. These cases are especially prevalent when the cache line size is 4 words. In these cases, the IDT73210 can be used. The part was originally designed for embedded R3000 read and write buffering, and also works well for integrated R4000 read and write buffering. It is an 8-bit transceiver with an extra data input which can generate parity. In one direction it is registered once, while in the other direction, it is registered twice. Thus, by setting it up so that the write buffer uses the 2-deep path, and the read buffer uses the 1-deep path, the part can be used in R4000 systems. The $\overline{BEN}$ and SEL pins can be used to control register ripple-through. The most straightforward way to use the controls requires Y-register loading by the first double-word, followed by ripple-through enabling, so the first double word is put into register Z as the second double word is loaded into register Y. Thus, the second double word must come on the clock cycle immediately following the first double word. This data rate pattern can be achieved by selecting a D or DDx pattern from Table 7 with the serial boot interface reset initialization. Other methods which use features of the IDT73210 not detailed here can be implemented to handle other kinds of data patterns. However, the controls will be more complicated than the above case.

```
!BE_B/ {BYTE ENABLE FOR THE LANE FOR DATABITS 55:48}
 := ((RESET/ AND !VALIDOUT/ AND SYSCMD[8:5]==b'1X1X) AND
    !BIGEND AND (MEMADDR[2:0]==b'110) AND (SYSCMD[2:0]==b'000) OR {LIT BYTE  }
    !BIGEND AND (MEMADDR[2:0]==b'110) AND (SYSCMD[2:0]==b'001) OR {LIT 1/2 WD}
    !BIGEND AND (MEMADDR[2:0]==b'100) AND (SYSCMD[2:0]==b'010) OR {LIT 3BYTE }
    !BIGEND AND (MEMADDR[2:0]==b'101) AND (SYSCMD[2:0]==b'010) OR {LIT 3BYTE }
    !BIGEND AND (MEMADDR[2:0]==b'100) AND (SYSCMD[2:0]==b'011) OR {LIT WORD  }
    !BIGEND AND (MEMADDR[2:0]==b'011) AND (SYSCMD[2:0]==b'100) OR {LIT 5BYTE }
    !BIGEND AND (MEMADDR[2:0]==b'010) AND (SYSCMD[2:0]==b'101) OR {LIT 6BYTE }
    !BIGEND AND (MEMADDR[2:0]==b'000) AND (SYSCMD[2:0]==b'110) OR {LIT 7BYTE }
    !BIGEND AND (MEMADDR[2:0]==b'001) AND (SYSCMD[2:0]==b'110) OR {LIT 7BYTE }

     BIGEND AND (MEMADDR[2:0]==b'001) AND (SYSCMD[2:0]==b'000) OR {BIG BYTE  }
     BIGEND AND (MEMADDR[2:0]==b'000) AND (SYSCMD[2:0]==b'001) OR {BIG 1/2 WD}
     BIGEND AND (MEMADDR[2:0]==b'000) AND (SYSCMD[2:0]==b'010) OR {BIG 3BYTE }
     BIGEND AND (MEMADDR[2:0]==b'001) AND (SYSCMD[2:0]==b'010) OR {BIG 3BYTE }
     BIGEND AND (MEMADDR[2:0]==b'000) AND (SYSCMD[2:0]==b'011) OR {BIG WORD  }
     BIGEND AND (MEMADDR[2:0]==b'000) AND (SYSCMD[2:0]==b'100) OR {BIG 5BYTE }
     BIGEND AND (MEMADDR[2:0]==b'000) AND (SYSCMD[2:0]==b'101) OR {BIG 6BYTE }
     BIGEND AND (MEMADDR[2:0]==b'000) AND (SYSCMD[2:0]==b'110) OR {BIG 7BYTE }
     BIGEND AND (MEMADDR[2:0]==b'001) AND (SYSCMD[2:0]==b'110) OR {BIG 7BYTE }

                (MEMADDR[2:0]==b'000) AND (SYSCMD[2:0]==b'111) OR {DOUBLE WD }
                                          (SYSCMD[4:3]==b'1X)  OR {BLOCK     }
    (!BE_B/ AND !MEM_ACKNOWLEDGE/)
);
```

**Table 9. Byte Enable PLA Equation**

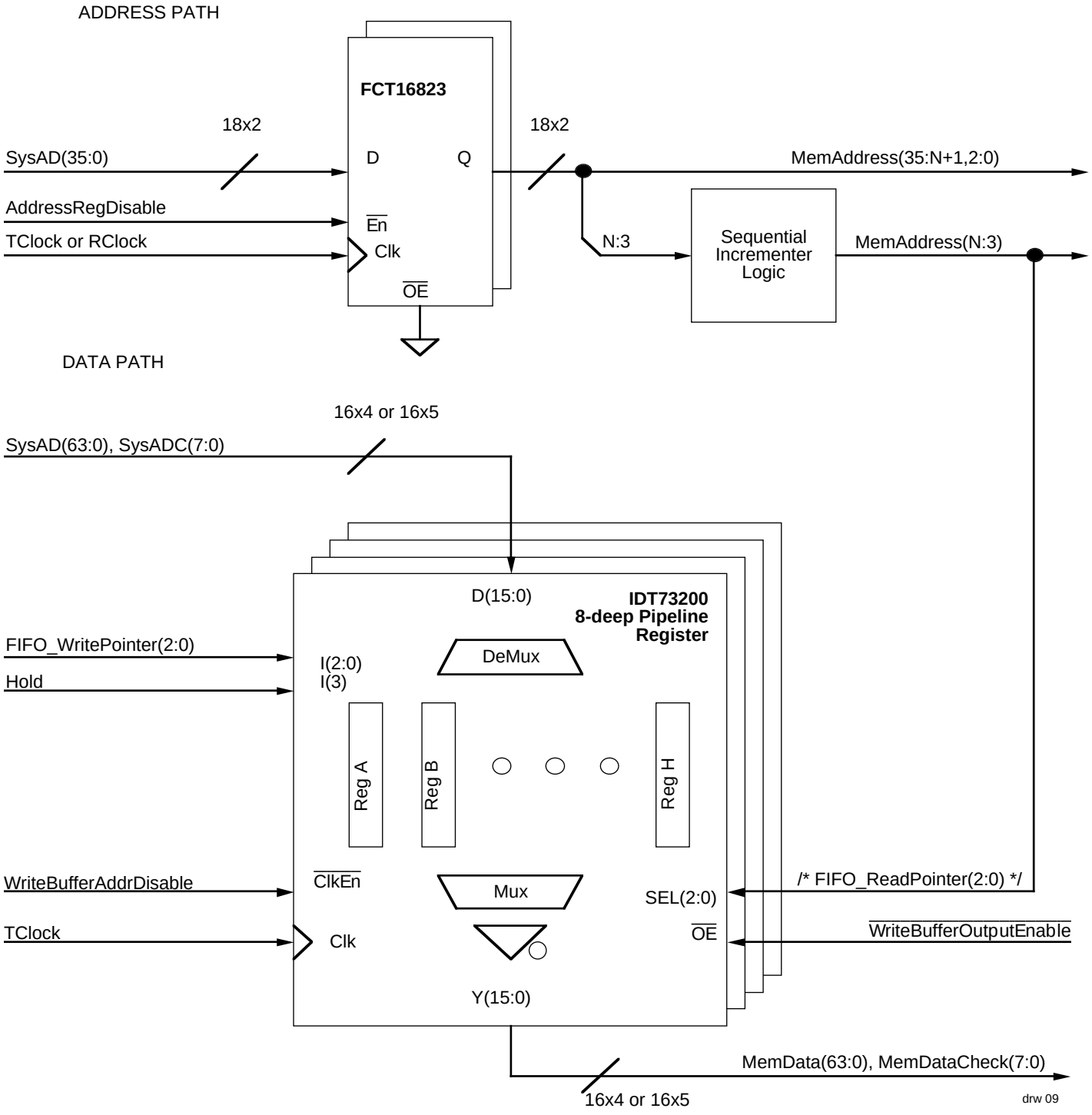


**Figure 9. R4000 8-Level, 64-bit Wide Write Buffer**

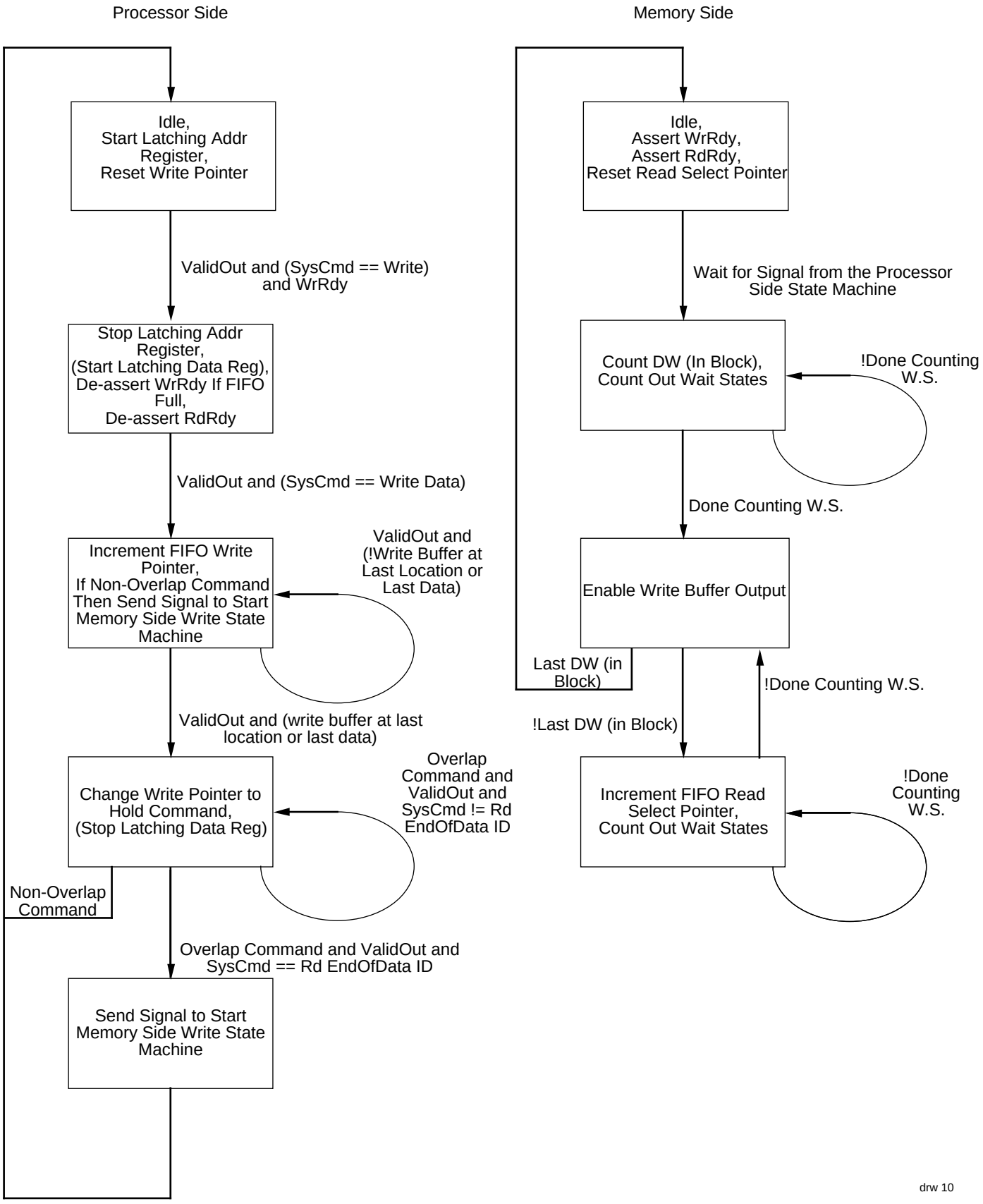


**Figure 10. R4000 Write Buffer State Machine**

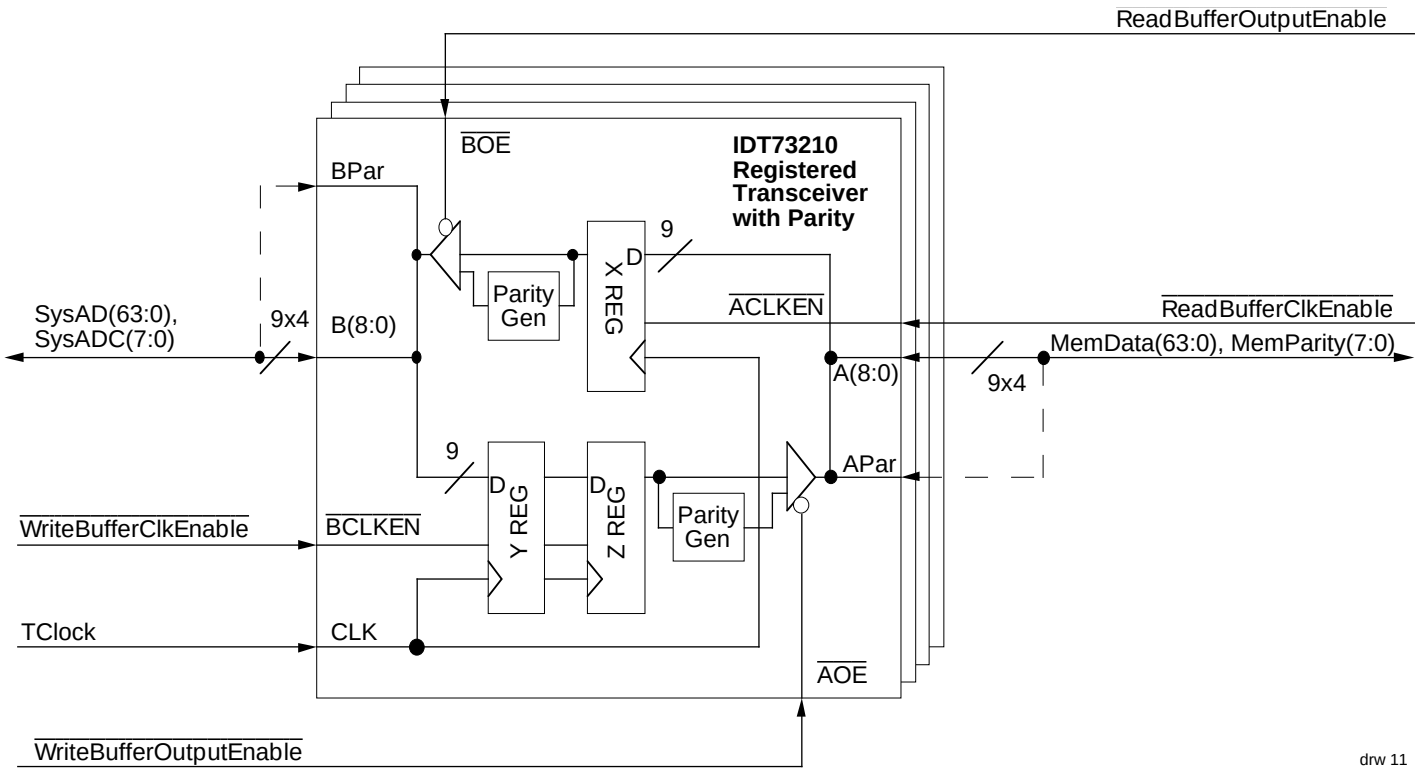


**Figure 11. R4000 Integrated Read and Write Buffer**

## SUMMARY

The R4000 uses three groups of signals for its System Interface between the CPU and main memory, consisting of the SysAD bus, the SysCmd bus, and a small group of control signals. Even though the R4000 uses a high-speed 50MHz bus, worst case timing issues with the R4000 System Interface are greatly simplified because of the completely synchronized bus and control signals. The read and write system interface on the R4000 uses a concept of multi-level buffering/ registering to maintain high throughput, by preventing unnecessary stalls and by allowing operations such as writes to happen in parallel with cached instructions and data. By using multi-level buffering on writes, the CPU can continue to run from cache while the main memory system retires writes at its own speed. Examples using off-the-shelf interface parts such as the FCT16823T 18-bit register with data enable, the 16-bit IDT73200 pipelined register, and the IDT73210 8-bit 2-level/ 1-level registered transceiver show how to easily implement read and write buffers for the R4000.