High Performance Memory Controller for the Intel® 80200 Processor

Application Note

March 2001

Order Number: 273494-001

Contents

Figures

Tables

Revision History

Date	Revision	Description
February 2001	001	New document

1.0 Introduction

This application note discusses the design of a high-performance memory controller for the Intel® 80200 processor, its implementation in an FPGA companion chip, and its application to a high-performance board platform. The Intel 80200 processor is the first microprocessor based on the Intel XScale microarchitecture, which is designed to optimize low power consumption and high performance processing for a wide range of Internet devices from the same core. This document is intended for system architects and hardware design engineers who are developing memory subsystem controllers in FPGA or ASIC silicon for the Intel 80200 processor.

The Intel LRH evaluation platform for the 80200 processor and its FPGA Companion Chip (80200FCC) memory controller are used as design examples throughout this document. The 80200FCC includes a high performance 64-bit 100 MHz SDRAM controller and an 8-bit peripheral bus controller implemented in a Xilinx Spartan-II FPGA.

Considerations for usage and modification of the 80200FCC design and targeting to ASIC implementation are discussed in this document, as are design issues for implementing custom 80200 companion chips. Board-level architectural trade-offs and design issues are also presented.

Complete design collateral for the 80200FCC (including VHDL source code) and the LRH evaluation platform is located at http://developer.intel.com/design/iio/docs/iop310.htm. The LRH evaluation platform can be obtained by contacting the following:

ADI Engineering
Phone: 804-978-2888
Fax: 804-978-1803
Email: info@adiengineering.com
URL: http://www.adiengineering.com/

2.0 Overview

2.1 Intel 80200 Processor

The Intel XScale microarchitecture is a new high-performance, ultra-low power microarchitecture compliant with the ARM v.5TE instruction set. The microarchitecture surrounds the ARM compliant execution core with Instruction and Data Management Units; Instruction, Data and Mini-Data Caches; Write, Fill, Pend and Branch Target Buffers; Power Management, Performance Monitoring, Debug, and JTAG Units; Coprocessor Interface; MAC Coprocessor; and a Core Memory Bus. The Intel XScale microarchitecture allows developers of a wide range of Internet applications to optimize the needs of their applications, from ultra-low power consumption to high performance processing.

The Intel 80200 processor is designed for high performance and low power, leading the industry in MIPs/mW. The Intel 80200 processor integrates an external bus controller and an interrupt controller around the ARM compliant processor core. Intended markets include handheld devices, networking, and remote access servers.

The Intel 80200 processor includes a rich set of architectural features that allow it to achieve high performance. Many of these architectural features are designed to hide memory latency that often is a serious impediment to high performance processors. These features include:

- The ability to continue instruction execution while the data cache is retrieving from external memory
- A write buffer
- Write-back caching
- Various data cache allocation policies which can be configured differently for each application
- Cache locking
- A pipelined external bus

2.1.1 External Bus Interface

The 80200 processor's external bus interface is intended primarily as a memory and I/O bus for the 80200 processor, not as a general purpose multi-master bus, although it is possible to have several masters on it efficiently. It is deeply pipelined to hide external memory latency, and it works well with pipelined memory technologies. The 80200 processor's external bus interface can be configured to operate with either a 32-bit or 64-bit wide data path. The LRH evaluation platform is designed to use 64-bit mode exclusively.

The 80200 processor external bus interface is split into separate request and data buses, both of which are synchronous with respect to the 80200 processor's 100 MHz memory clock (MCLK). The request bus is used to issue read or write requests from the 80200 processor (or other external bus masters) to the memory controller. Each request is presented over two MCLK periods. Upper address and request type information (Write/Read, Lock) are presented during the first clock while lower address and length information are delivered on the second. The 80200 processor can have at most four requests outstanding at a time. A locked request type indicates that the current request is part of an atomic read/write pair which must not be interrupted by an external bus master. Supported request lengths for 32-bit and 64-bit bus configurations can be found in the *Intel® 80200 Processor Developer's Manual* (public order number 273411-001).

Some time after a request is made on the request bus, data must be transferred for that request on the data bus. Each request has a corresponding transaction of one or more data cycles. Data bus transactions must occur in the same order as the requests were made but the request and data buses are otherwise independent. The data bus consists of a 32-bit or 64-bit wide data path and associated data check bits (DCB) for use with ECC memory. Eight byte enable signals (nBE), one per data byte, select which bytes within the 32-bit or 64-bit bus are to be written during a write transaction.

Each data cycle begins with the assertion of the DVALID signal, indicating that a data transfer for the next pending request is to occur two MCLKs later. DVALID is asserted for each data cycle required to complete a request. For burst read accesses, the 80200 processor supports a critical word first protocol which allows the data to be returned starting with the requested DWORD instead of starting at the beginning of the block of data. The CWF signal is used during read bursts to indicate the order in which the data will be returned. This signal is sampled concurrently with the first DVALID of the read data cycle. Similarly, the ABORT signal is used to terminate an invalid request. It is sampled whenever DVALID is asserted.

2.2 Memory Overview

2.2.1 Synchronous DRAM

Synchronous DRAM (SDRAM) is well suited for use with the Intel 80200 processor. It provides high density and high performance at a low cost. Furthermore, the inherent pipelining of SDRAM allows it to operate efficiently with the Intel 80200 processor's pipelined external bus interface.

SDRAM is a burst-oriented memory technology with a fully synchronous command-driven interface. Data is read and written at a rate of one word per clock within a burst transaction. Each internal SDRAM bank has its own row address latch and decoder. With a properly designed controller, this architecture allows one simultaneously open memory row per bank. Read and write accesses falling within any of the open rows can complete, unhindered by the need to activate the row. When row activations or precharges are necessary, they can be hidden by interleaving accesses to the multiple banks.

SDRAM allows command pipelining, with a new command being issued by the controller prior to completion of the current command to keep the data bus fully utilized. After a read command is issued to SDRAM, a time called the "CAS Latency" must pass prior to data being available. SDRAM devices support CAS latencies of either 2 or 3 clocks, with lower latencies providing higher memory performance. Thanks to its pipelined architecture, SDRAM allows the column address to be changed arbitrarily within each open row while still maintaining a transfer rate of one word per clock.

Four 64-Mbit PC100 compliant SDRAM devices were selected for use in the LRH evaluation platform. They are arranged as a single block of 4 M x 64 memory. These devices have a 4 bank internal architecture with 12-bit row and 8-bit column addresses.

2.2.1.1 ECC Memory Overview

Error Correcting Code (ECC) memory is commonly used to protect against memory errors in critical applications such as servers and Internet infrastructure equipment. During writes, the ECC code is computed and written to extra memory bits. During reads, the ECC code is computed from the read data and compared to the ECC code read out of memory. The code is designed such that all single bit errors can be transparently corrected, and all double bit errors and many higher-order bit errors are detectable but not correctable.

The 80200 processor supports ECC for 64-bit wide memory systems. Software running on the 80200 processor may configure memory pages as ECC protected. For such pages, the 80200 processor will automatically check the ECC code associated with the read data, and it will generate an ECC code for write data. The 8-bit ECC codes are transported over the DCB bus, and an additional eight bits of memory storage are required to store the ECC code.

Extra memory for ECC is not provided on the LRH evaluation platform. The 80200FCC memory controller was designed to easily support ECC memory if required, however. An additional 64 Mbit SDRAM device would need to be added (either an 8-bit wide or half of a 16-bit wide device). All SDRAM control signals would be connected in parallel with the other SDRAM, except for the data mask input (DQM) that can be tied low. The DQM signal can simply be tied low since the 80200FCC does not need to use the DQM to assist in terminating a burst read or write cycle.

2.2.2 Flash Memory and Other Peripherals

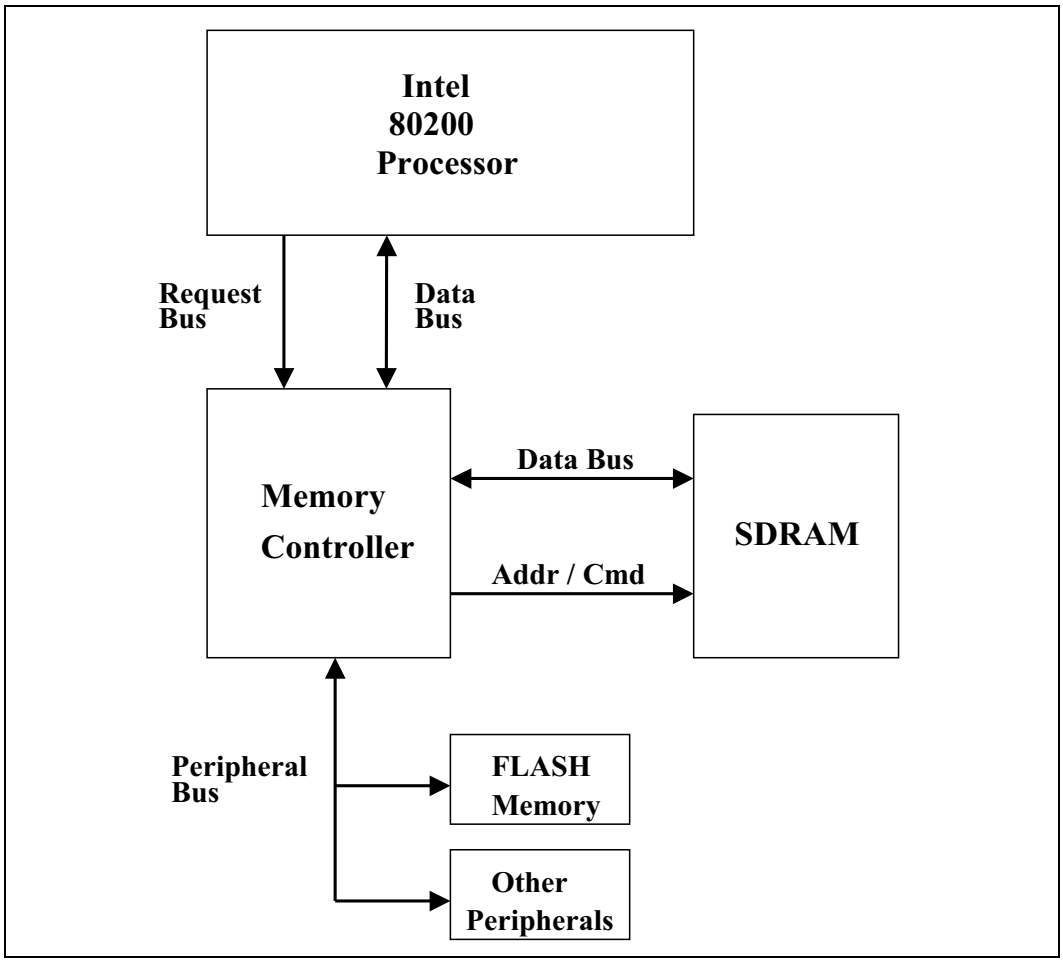
Flash memory is commonly used as non-volatile storage for boot code and as an in-system programmable repository for application code, storing it prior to loading into SDRAM for high-speed execution. Other peripherals such as UARTs and parallel I/O ports typically are added to a system. Because of the low speed of these devices and bus interface issues, they cannot be connected directly to the 80200 processor or SDRAM data bus. A separate peripheral bus interface is required to address the interface issues of these devices.

On the LRH evaluation platform, a lower speed 8-bit peripheral bus is developed by the 80200FCC for connection of less performance critical devices such as Flash memory and I/O.

2.3 Board Architectures

Two basic board architectures are possible with the 80200 processor. In the first approach, shown in Figure 1, the memory controller produces a secondary 64-bit data bus to connect to SDRAM. The 80200 processor, the memory controller, and the SDRAM can all operate synchronously with respect to MCLK. However, additional latency will be inserted for all SDRAM accesses because the memory controller is in the data path.

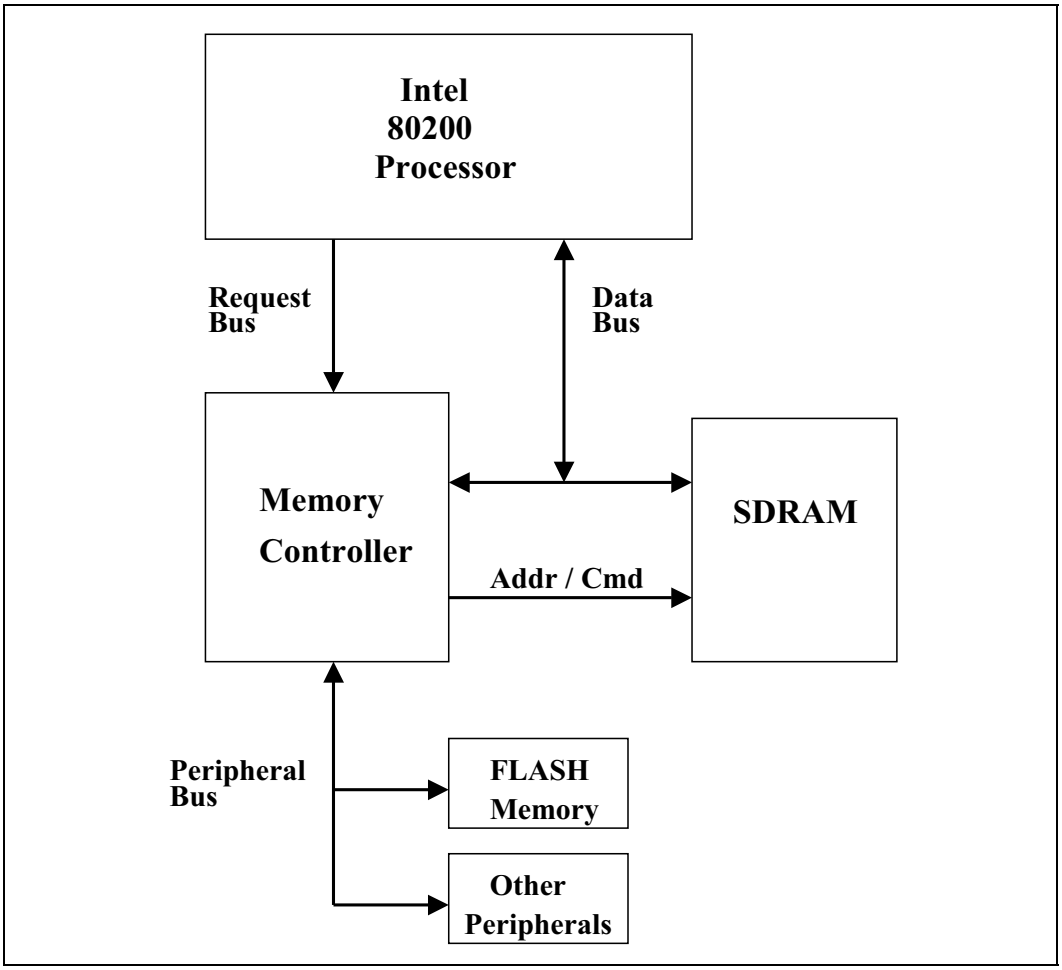
Figure 1. Board Architecture – Separate Data Bus



The main advantage to this approach is that signal integrity and board-level timing issues are reduced relative to the common data bus architecture. While signal integrity is still an issue, signal termination requirements are simplified, and greater flexibility may be allowed in component placement and signal routing. One major disadvantage to the separate data bus approach is the extra implementation costs due to the greater I/O requirements in the memory controller.

In the common data bus architecture, shown in Figure 2, 64-bit SDRAM is connected directly to the 80200 processor data bus. With this approach, a second 64-bit memory data bus is not necessary, providing significant savings in memory controller cost. Because SDRAM data does not pass through the memory controller, access latency is reduced for greater overall system performance.

Figure 2. Board Architecture – Common Data Bus



Signal integrity is a greater concern with the common data bus architecture, though. Since the data bus connects to three drivers/receivers and is not a point-to-point architecture, more careful attention must be paid to simulate board-level signal integrity early on in the design process. Signal integrity simulations must be used to design appropriate signal termination, to select memory controller output driver characteristics, and to guide component placement and signal routing during board layout.

The Intel LRH evaluation board was designed using the common data bus architecture to maximize memory subsystem performance.

2.4 80200FCC Memory Controller Description

The 80200FCC memory controller has the task of accepting requests from the Intel 80200 processor and performing the necessary operations to transfer data with the specified device. The 80200FCC handles access requests to SDRAM and to slower devices residing on the peripheral bus, and it aborts any invalid requests. The memory controller takes advantage of the pipelined 80200 processor bus interface to keep the SDRAM data bus fully utilized, automatically inserting bus turnaround cycles as required. It stores up to four 80200 processor bus requests until the resources are available to complete the request.

High performance SDRAM access is provided by taking advantage of the SDRAM's bank architecture and command pipelining when possible. Memory rows are held open in each of the four SDRAM banks to minimize access latency, especially for performance critical bus requests such as cache line fills which typically access sequential addresses in memory. This is achieved by keeping track of the currently open row in each bank, and closing out rows only when necessary. SDRAM timing parameters, many of which are configurable in the 80200FCC memory controller, are guaranteed through the use of various timers. The 80200FCC supports all 80200 processor bus request types and lengths and the critical word first protocol for SDRAM accesses.

SDRAM refreshes are automatically handled by the 80200FCC's refresh controller. The 80200FCC implements a two-level refresh priority scheme to minimize the impact of SDRAM refreshes on the 80200 processor bus. If the SDRAM is not idle when a refresh cycle is requested by the 80200FCC refresh controller, the request is queued for later execution. Up to eight refresh requests will be queued in this manner. Once eight pending refresh requests accumulate, their priority is increased and they are performed after completion of the current SDRAM cycle. To further reduce refresh overhead, the 80200FCC bursts up to 8 refresh cycles. The 80200FCC also automatically initializes SDRAM on powerup prior to servicing any 80200 processor bus requests.

An 8-bit peripheral bus is provided for connection of less performance critical devices such as Flash memory and I/O ports. The memory controller interfaces the high-speed 64-bit 80200 processor data bus to the slower 8-bit peripheral bus. Chip select generation and configurable wait state control are provided to accommodate a variety of peripheral devices. All 80200 processor peripheral bus read cycle request types and lengths are supported, as is the critical word first protocol. Single byte write requests to the peripheral bus are supported. Peripheral bus write posting reduces the impact of slow peripheral bus devices on SDRAM accesses.

Major features of the 80200FCC memory controller include:

- Automatic SDRAM initialization
- Maintains row state for each SDRAM bank, closing rows only when necessary
- Automatically guarantees SDRAM command sequence timings
- Manages processor data bus by automatically inserting bus turn-around cycles when required
- Queues up to 4 pending processor requests
- Minimizes SDRAM refresh cycle impact (two level refresh request priority, burst refreshes)
- Pipelines SDRAM commands to keep data bus fully utilized
- Provides configurable SDRAM timing parameters
- Supports Critical Word First protocol for SDRAM and other peripherals

- Aborts illegal bus requests
- Provides separate 8-bit peripheral bus, minimizing SDRAM performance impact
- Supports all peripheral read cycle types
- Single stage write posting to peripheral bus
- Sustained SDRAM bandwidth of 800 Mbyte/sec

3.0 Design Considerations

3.1 FPGA Implementation

FPGA implementation of the 80200FCC memory controller was a major consideration that guided and constrained its design. Given the complexity of the SDRAM performance enhancements implemented by the 80200FCC, the system-level operating frequency requirement of 100 MHz is pushing the limits of what today's lower cost FPGAs can achieve. Several architectural trade-offs and implementation techniques were used to reach this goal. If the 80200FCC design were implemented in an ASIC, even higher operating speeds could be achieved, access latencies reduced, or less silicon used by changing the way certain sections of the design are implemented.

Many of today's popular FPGAs use 4-input look up table architectures. Implementing wide logic functions with this type of architecture quickly increases the number of logic levels, limiting the speed at which the design can run. Several techniques were used in the 80200FCC to avoid wide logic functions. One technique was to use pipelining to split wide combinatorial functions across multiple clock cycles. While pipelining increases operating frequency, it also increases latency. One example of this in the 80200FCC design is the logic that compares a request address to the open row addresses in each SDRAM bank. This wide comparison was split across two clock cycles. While achieving the goal of 100 MHz operation, this added an extra clock of latency between the time a request arrives and the first command to the SDRAM can be issued (if no other requests are pending).

Another FPGA-specific implementation strategy was to design finite state machines with shallow state transition logic at the expense of adding more states. One-hot state encoding was used to minimize the number of signals required to represent a particular state, again to minimize state transition logic at the expense of requiring more storage for state information. Another technique was to "pre-calculate" signals in critical paths. For example, many state machine output signals were designed to come directly from flip-flops. Several critical counters were designed with terminal count outputs coming directly from flip-flops instead of from wide combinatorial logic. Information about a bus request, such as SDRAM bank state, is calculated while the request is still in the queue instead of at the output of the queue.

Several of the finite state machines were designed with duplicate state storage. This helped to minimize fanout and thus decrease signal propagation delays. In some cases, the secondary state storage was encoded in a different way to aid in minimizing logic for different sections of the design.

The 80200FCC also took advantage of several FPGA architectural features including Delay-Locked Loops (DLLs), low skew global routing resources, I/O buffer flip-flops, configurable output drivers, internal tri-state buffers, etc. Using these features was crucial in achieving 100 MHz operation.

3.2 Board Level Timing Issues

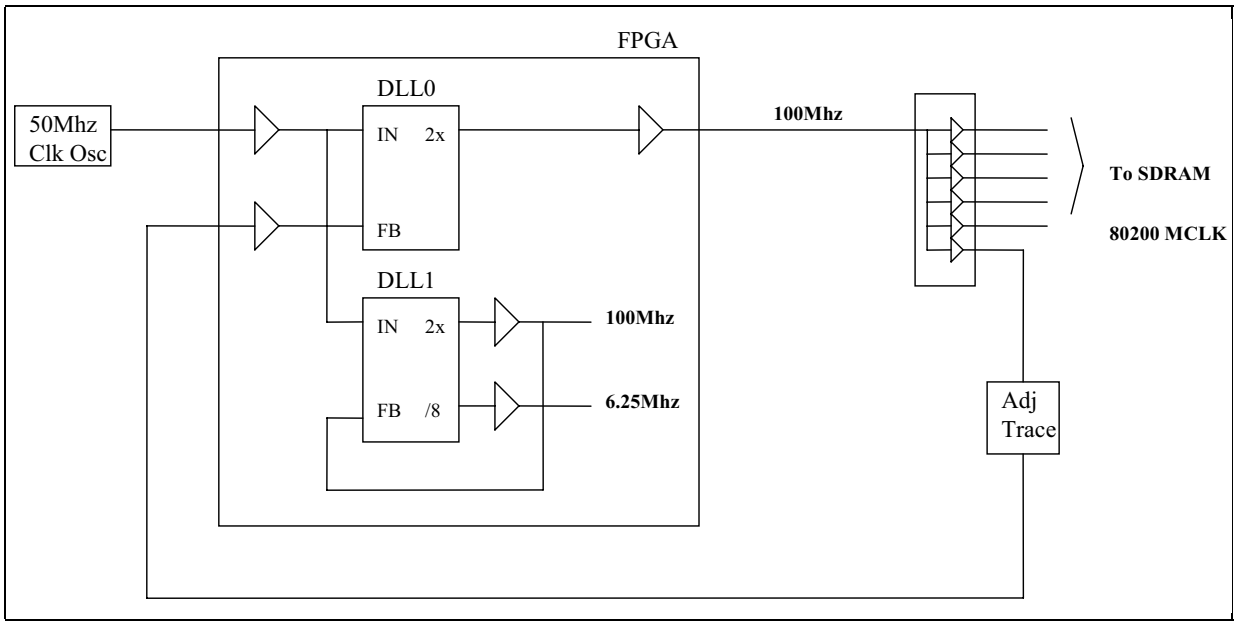
One major source of concern for the design was system-level timing. To minimize board-level clock skew, DLLs inside the FPGA were used to generate the 100 MHz memory bus clock. Using a multiple DLL approach, low skew between the 80200 processor, SDRAM and the memory controller was achieved. A lower speed peripheral bus clock was created for slower peripheral device timing.

Another key approach was to interface all 80200 processor request and data bus signals directly to flip-flops contained inside the FPGA I/O buffers. This greatly minimized setup and hold time requirements for these crucial signals. Signal integrity analysis and simulation were performed during the board design process to make sure that board-level timing was not degraded by signal integrity issues and that appropriate FPGA output buffer drive strengths were selected.

3.2.1 Clock Distribution

The 100 MHz memory bus clock is generated by DLLs inside the FPGA which double a 50 MHz input reference clock. As shown in Figure 3, two DLLs are used to synchronize the 100 MHz internal 80200FCC clock with the 100 MHz clock distributed to the 80200 processor and SDRAM. This dual DLL approach is recommended by Xilinx to minimize skew between the external and internal versions of a clock signal. A trace length adjustable feedback clock was used to allow minor phase adjustments during development if necessary. An external low-skew clock buffer distributes the clock across the board to the four SDRAMs and the 80200 processor.

Figure 3. Clock Distribution



One of the DLLs inside the FPGA was also used to generate a 6.25 MHz clock (50 MHz input clock divided by 8) for use by the slower peripheral bus logic. The peripheral bus clock is synchronous with the 100 MHz memory clock allowing synchronous design techniques to be used between the two clock domains.

A 66 MHz input clock is used as the 80200 processor core logic clock source and for possible future expansion of the 80200FCC to accommodate a 33/66 MHz PCI interface. A DLL divides the 66 MHz reference clock input by 8 to produce a 8.33 MHz clock used by the SDRAM refresh timer and the UART baud clock generator.

3.2.2 Signal Integrity Analysis

The high speed of the 80200 processor external bus requires that board-level signal integrity be thoroughly analyzed and simulated during the design process. Without proper attention, signal integrity issues could result in unreliable operation of the system and could contribute to electromagnetic emissions issues. Signal integrity is of special concern when using the common data bus architecture where the 80200 processor, the companion chip and the SDRAM all connect to a common data bus.

Signal integrity problems with a board design can degrade the timing of high-speed signals enough to violate critical timing requirements. Excessive overshoot and undershoot can exceed absolute maximum device input voltage ratings, overstressing devices on the bus and possibly leading to premature failure. Ringback can lead to multiple transitions on clock signals. Flight time due to propagation and loading effects can degrade timings on critical signals. Crosstalk can also cause serious problems. The impacts of all these effects must be analyzed early-on in the design process to assure reliability of the final design.

In the context of designing an FPGA or ASIC companion chip, several additional signal integrity issues must be addressed. Due to the plethora of I/O cell choices on such devices, it becomes critical to use signal integrity simulations to determine the minimal drive speed and strength I/O cell that can meet all timing requirements under worst-case conditions. The use of excessively fast I/O cell drivers can unnecessarily exacerbate signal integrity and power supply decoupling issues, especially when 64-bit buses are involved.

The LRH board incorporates series termination resistors on all data bus signals, clock signals, and SDRAM control signals. Signal integrity simulations revealed that termination was required to reduce overshoot and undershoot to acceptable limits and series termination was found to be an adequate solution. Series termination also reduced dynamic current requirements of the 80200 processor, the 80200FCC, and SDRAM. This in turn relaxed requirements on power supply decoupling for these devices.

There are many third-party tools available for signal integrity analysis and simulation. Whatever tool is used, the following design approach is recommended:

1. Using IBIS models of the 80200 processor, the companion chip I/O cells, and SDRAM, and estimated pre-layout board-level component placement and board construction details (layer count, board stack-up, dielectric constant), construct a signal integrity simulation environment for representative data, address, and control signals and high-speed clocks.
2. Run signal integrity simulations using estimated pre-layout placement data and examine the results. It is crucial to repeat for all combinations of worst-case and best-case drivers and receivers to make sure that worst-case performance of the system is acceptable.
3. If necessary, add signal terminations, choose different companion chip I/O cells, or modify the signal trace lengths. Iterate steps 1.–3. until a suitable solution is found.
4. Using dynamic current demands predicted by the simulations, design a power supply decoupling network having low enough parasitic inductance and resistance and high enough capacitance.
5. When post-layout physical board parameters, placements, and trace routings are available, repeat steps 1.–5. to validate the final design.

3.3 Flexible SDRAM Selection

Another goal of the 80200FCC design was to make it adaptable to a wide range of SDRAM devices. For this reason, the design targets generic PC100 compliant SDRAM devices. Several useful features available in many SDRAM parts, such as the BURST STOP command and separate burst configuration for reads and writes, are not required by the PC100 guidelines and were therefore not used by the 80200FCC. For this reason, the internal SDRAM burst address counter is not used by the 80200FCC memory controller. Instead, all reads and writes are individual single cycle accesses. Burst cycle addresses are generated by a burst address counter inside the 80200FCC memory controller.

To match the performance of the memory controller to a variety of SDRAM devices, many SDRAM timing parameters are configurable. Table 1 summarizes the SDRAM timing parameters used by the 80200FCC. Configurable timing parameters may be modified by changing a VHDL constant and re-synthesizing the design.

Table 1. SDRAM Timing Parameters

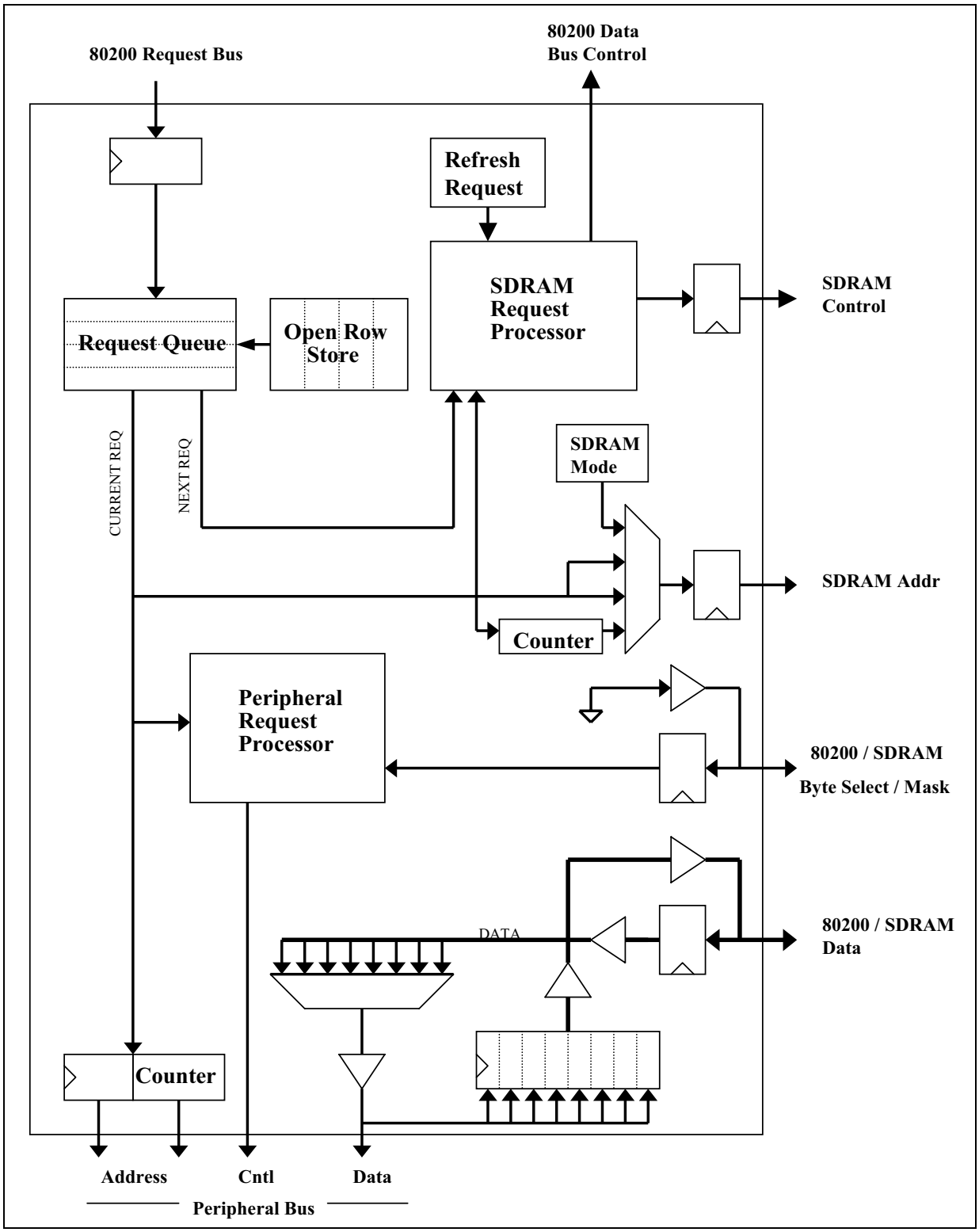
Timing Parameter	Description	Fixed/Configurable	Value (clocks)
Tcl	Cas Latency (READ cmd to data)	Config	2 or 3
Trp	Ras precharge (PRE/PALL to cmd)	Config	2 or 3
Tras	Ras active (ACT to PRE/PALL)	Config	3 or greater
Trcd	Ras to Cas delay (ACT to READ/WRITE)	Config	2 or 3
Trrd	Bank Activate delay (ACT to ACT[diff bank])	Fixed	2
Trc, Trfc	Ras Cycle time (REFRESH/ACT to REFRESH/ACT)	Config	Tras + Trcd
Tmrd	Mode Reg access (MODE to ACT)	Fixed	3
Tdqz	DQM# delay (DQM to data hi-Z)	Fixed	2
Tdpl	Data in to precharge (data in to PRE/PALL)	Fixed	2

4.0 80200FCC Memory Controller Implementation

4.1 Block Diagram

A block diagram of the 80200FCC memory controller is shown below in Figure 4. Each section of the 80200FCC memory controller is described in detail in the remainder of this chapter. It is suggested that the 80200FCC VHDL source code be used as a reference while reading the following sections.

Figure 4. 80200FCC Memory Controller Block Diagram



4.2 Bank State and Open Row Storage

In order to take maximum advantage of the pipelined external bus interface of the 80200 processor and the multi-bank SDRAM architecture, information about the status of each SDRAM bank is maintained in the memory controller. This information allows a row in each SDRAM bank to be held open until access to a different row is requested, avoiding the penalty of having to activate a row each time the SDRAM is accessed.

Associated with each of the four SDRAM banks is a register containing the bank state (IDLE or ACTIVE) along with the currently active row (if any). This information is compared to each 80200 processor bus request and used by the SDRAM Request Processor to determine how best to handle the request. In addition, a timer for each bank is maintained to keep track of how long a row has been open. The SDRAM Request Processor uses this information to guarantee that rows are held open for the minimum time required for proper operation. Whenever an ACTIVATE or PRECHARGE command is issued to the SDRAM, the bank state and open row information is updated accordingly. A PRECHARGE ALL command forces all banks to be marked as idle.

Information stored for each of the four SDRAM banks is as follows:

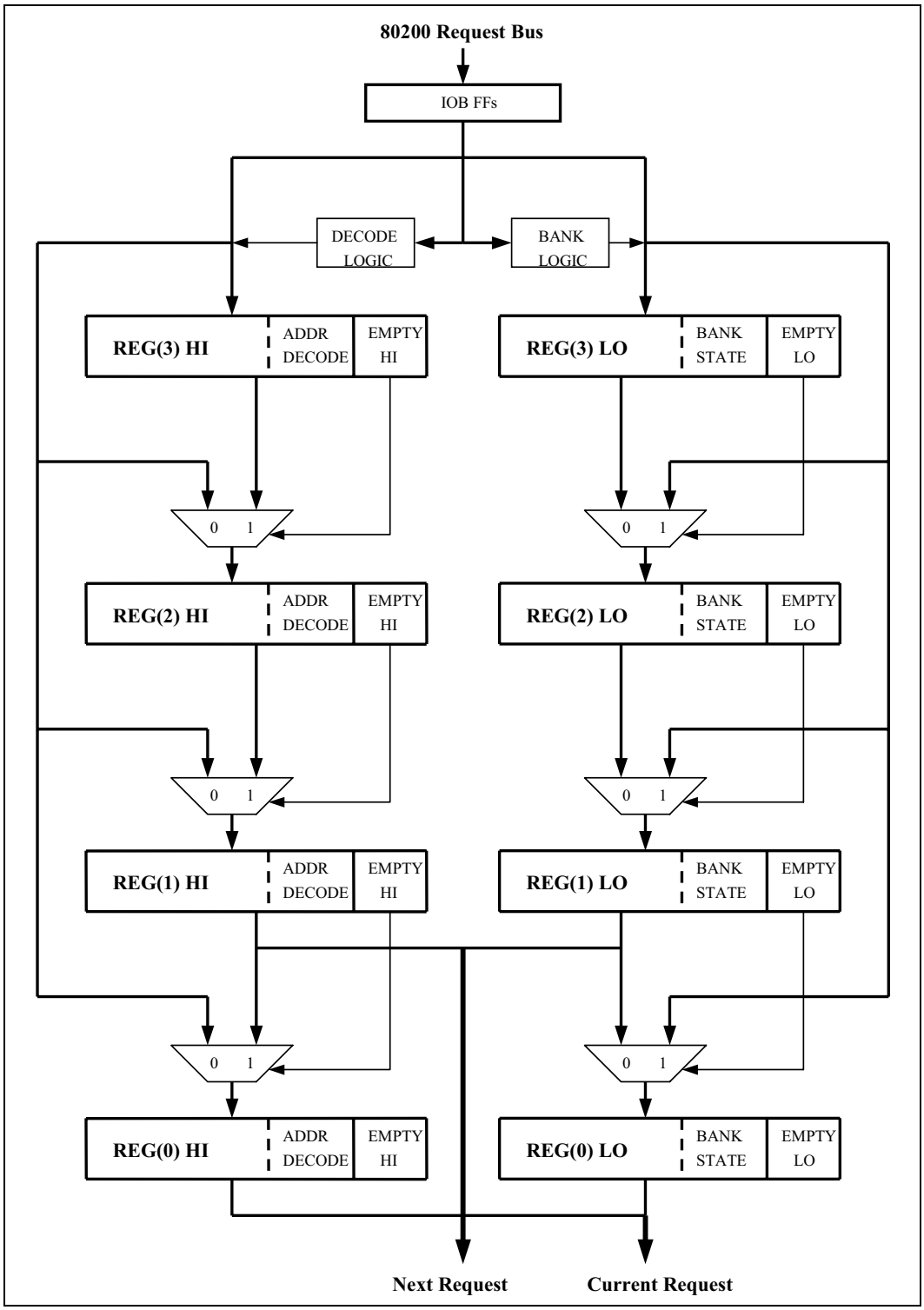
- ACTIVE Indicates status of bank: 1 => Bank currently active. 0 => Bank Idle.
- ROW[11:0] Currently open row (if any)

4.3 Request Queue

As shown in the block diagram in Figure 4, the Request Queue stores incoming 80200 processor bus requests until they can be processed. The Request Queue is essentially a four entry deep FIFO with each entry containing storage for a single bus request. The FIFO was implemented using four registers and a finite state machine to keep track of the state of the FIFO. Other simpler and more space efficient FIFO architectures were considered, but this approach was selected based on several performance considerations. With this approach, it is easy to not only look at the request at the front of the FIFO, but to look at the following request as well. This is necessary in order for the SDRAM Request Processor to feed commands to the SDRAM in a pipelined fashion. This FIFO architecture also allows storage of additional information associated with each request, such as SDRAM bank status, which may need to be updated simultaneously while the request is still in the queue. Also, with this type of FIFO all outputs come directly from flip-flops which aids in meeting system-level timing constraints.

As shown in Figure 5, each of the four FIFO registers has an associated input MUX that selects between the previous register or an incoming the 80200 processor request bus. The MUXs are controlled and registers loaded based on how many entries are currently in the queue, whether there is an entry arriving from the 80200 processor request bus, and whether the SDRAM Request Processor is popping a request from the front of the queue. Incoming 80200 processor requests are immediately stored in the register closest to the front of the FIFO.

Figure 5. Request Queue Architecture



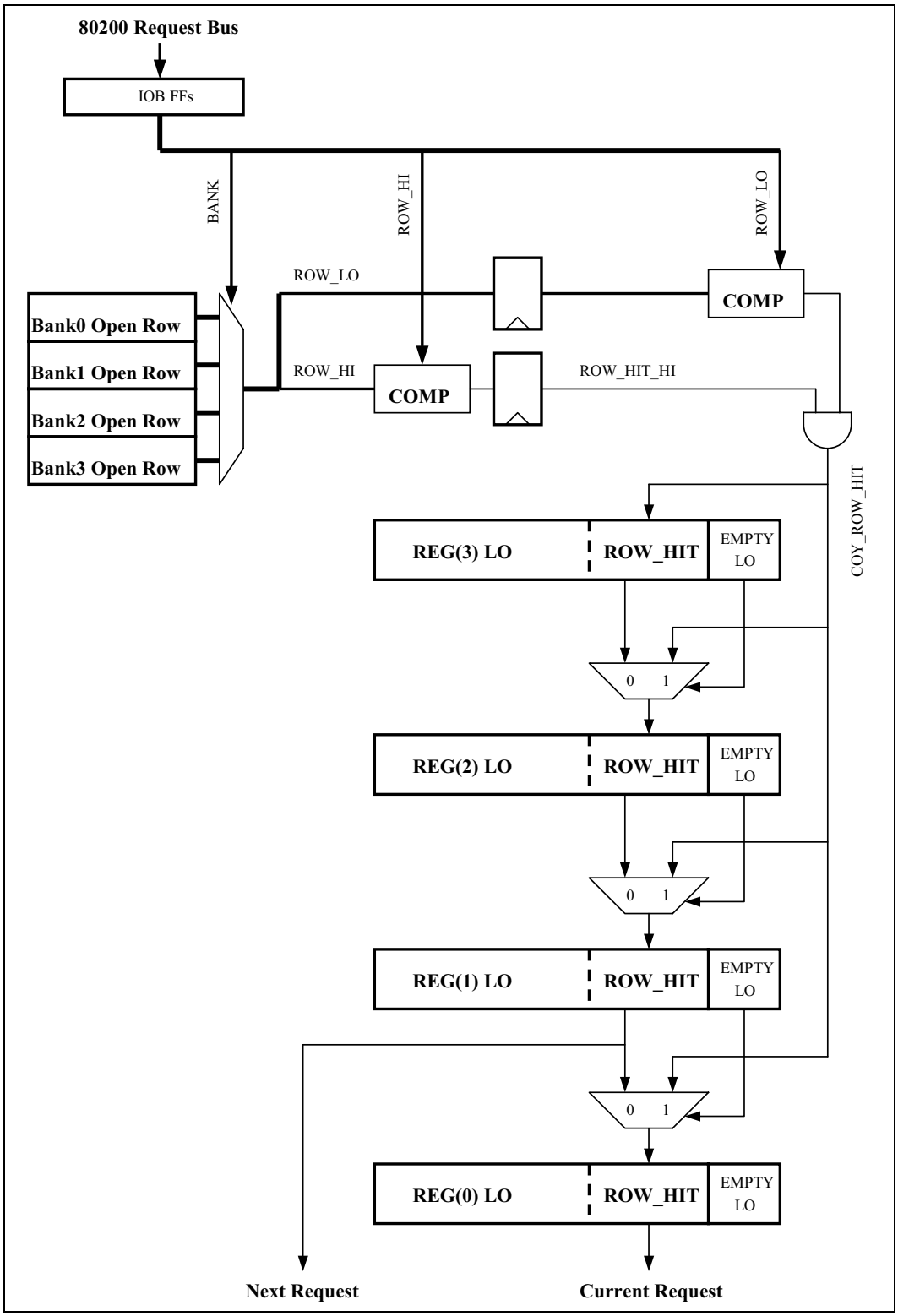
80200 processor bus requests arrive over two clock periods. Upper address and request type information arrive during the first clock, followed by lower address and length information on the second clock. The FIFO registers are split in a similar fashion with information pertaining to the first half of a bus request stored in the high part of the FIFO register and information pertaining to the second half of a request stored in the lower part. Once both halves of a request are loaded into the Request Queue, they are always kept together at the same level in the FIFO. Empty status is kept for each half of each FIFO register. Empty status for the low half is used to indicate that the entire request is present at that level in the FIFO.

Several other features were incorporated into the Request Queue to maximize performance and operating speed. Associated with each entry in the queue is additional information about the 80200 processor bus request including address decode and associated SDRAM bank status. The SDRAM Request Processor looks at this information to quickly determine how to handle the request.

The decode block shown in Figure 5 looks at the upper address of the incoming request to determine if the request is to a valid address and whether it is intended for SDRAM or a peripheral device. This information is stored along with the upper half of the bus request.

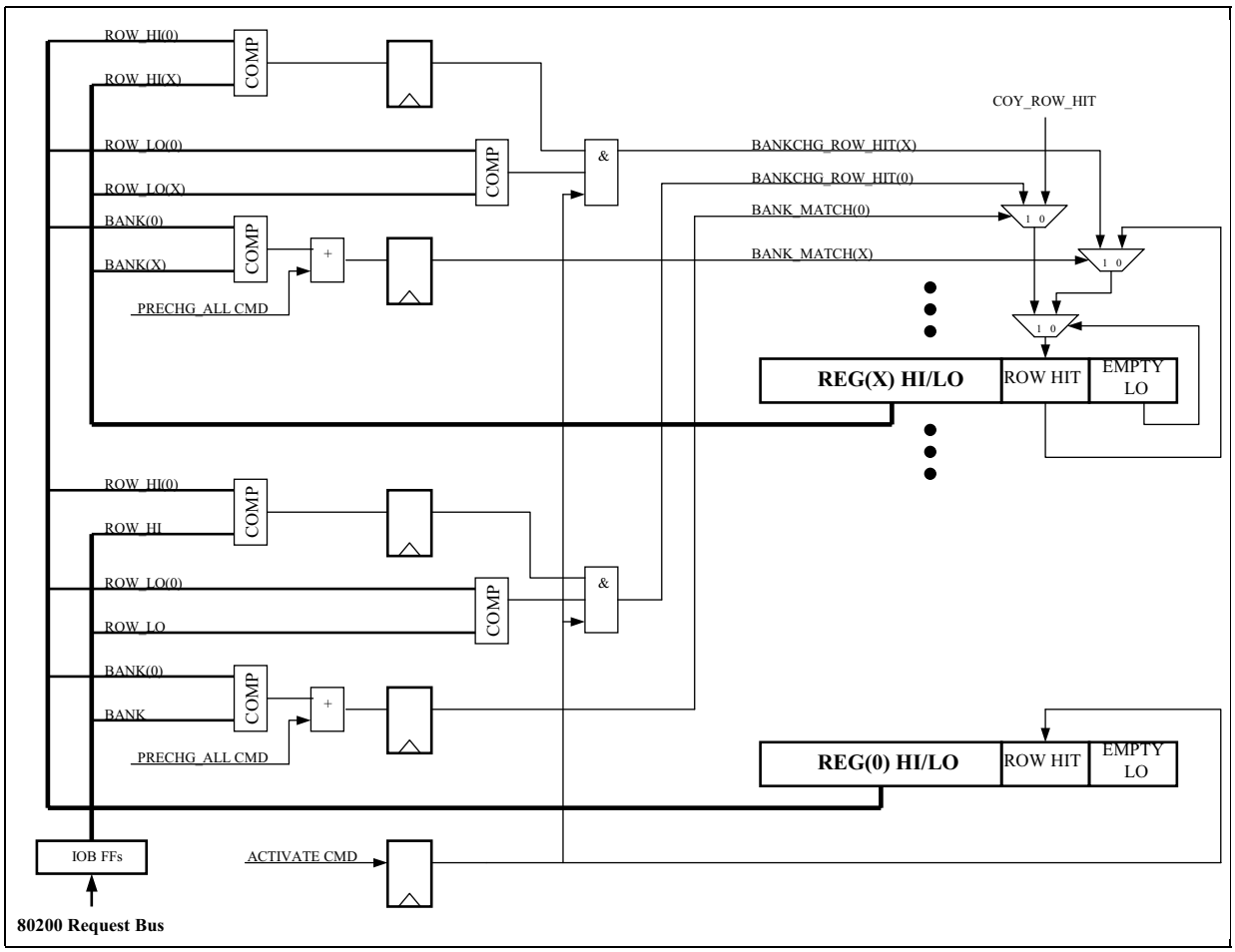
To present the SDRAM Request Processor with associated SDRAM bank status as soon as possible, this information is determined and carried along with each request in the queue. Due to the wide logic comparison required, associated SDRAM bank status is calculated over two clock cycles. This works well with incoming requests since they arrive in two clock periods anyway. When the upper half of a 80200 processor request is captured, the bank and upper row addresses are compared to the current SDRAM bank state/open row information. The lower row addresses are compared while the lower half of the 80200 processor request is captured. The SDRAM Request Processor is presented with two signals which indicate the SDRAM bank information for each bus request; ROW_IDLE (no row active) and ROW_HIT (row already active). The two clock pipeline of the Request Queue ROW_HIT logic is shown in Figure 6.

Figure 6. Request Bank SDRAM Bank State Logic



When the SDRAM bank state changes due to a PRECHARGE or ACTIVATE command, the ROW_HIT and BANK_IDLE information at each level in the FIFO needs to be updated as well if the change affects the SDRAM bank associated with the bus request. This again takes place over two clocks cycles, with the bank and upper row address comparisons occurring during the first clock, and the lower row address compare taking place during the second clock cycle. Figure 7 shows the two stage pipelined ROW_HIT logic associated with each queue entry used during SDRAM bank state changes. Please refer to Figure 6 for the source of the COY_ROW_HIT signal.

Figure 7. SDRAM Bank State Change Logic

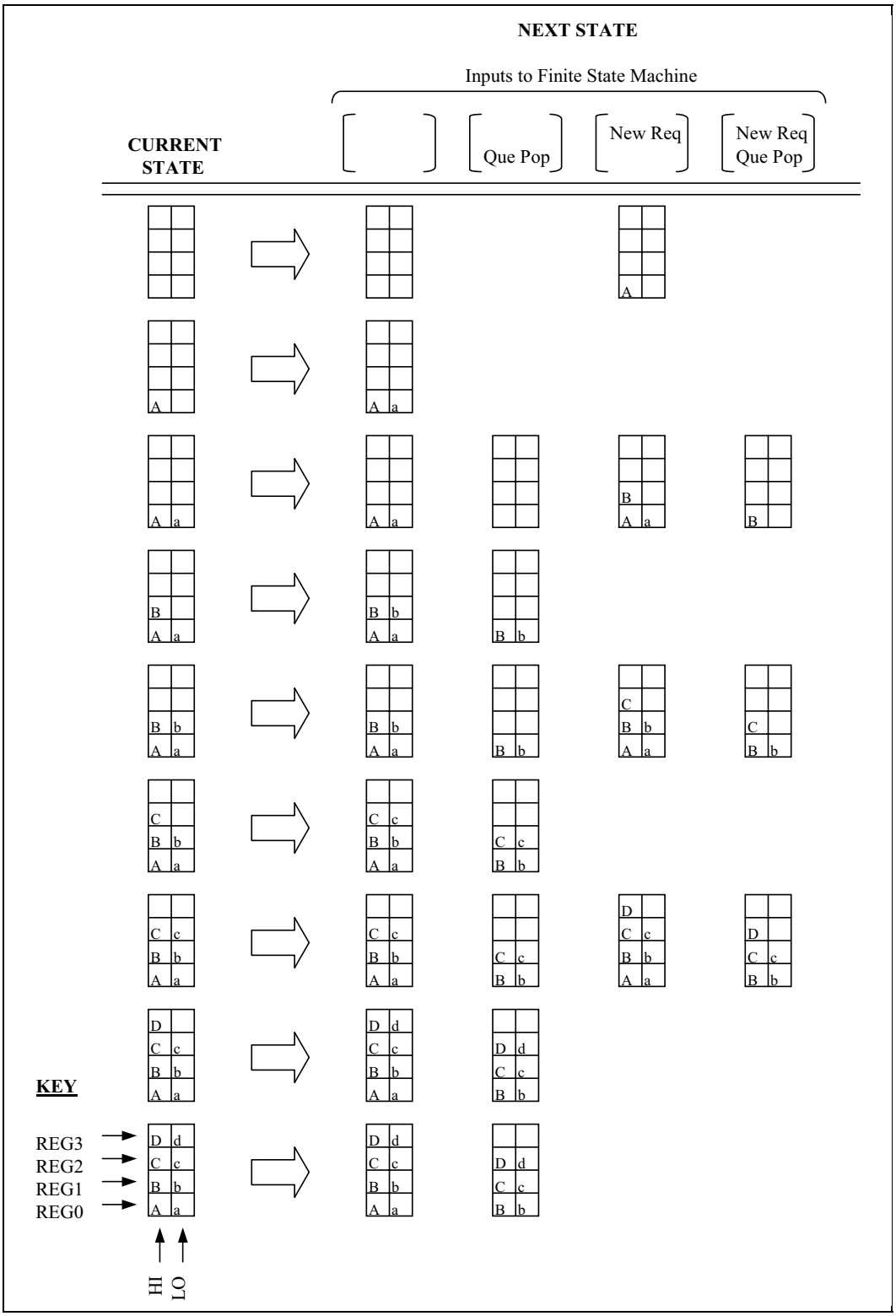


The state of the Request Queue is maintained in two different forms. One form (req_state_typ) is intended for one-hot encoding and is used for next state logic, register load logic, etc. The other form (queue_state_typ) actually maintains one bit for each register half to indicate if there is a valid entry in the that half (hi/lo) of the FIFO register. This information is used to control the MUXs between queue entries and to provide queue entry empty/full status (directly from flip-flops) to the SDRAM Request Processor. This approach also decreases signal loading to aid in achieving 100 MHz operating speed.

Figure 8 shows the Request Queue state logic in table form. State logic depends on two input signals: NEW_REQUEST and QUEUE_POP. The NEW_REQUEST signal is active only during the first half of an incoming 80200 processor bus request. The second half of a bus request follows immediately on the next clock edge. In Figure 8, the first half of an incoming request is indicated by a capital letter and the second half by the lower case letter (i.e. A = upper half of request, a =

lower half). The QUEUE_POP signal is asserted by the SDRAM Request Processor whenever it is done with current request and would like to pop the next request to the front of the queue. Certain inputs are prohibited when the Request Queue is in a particular state. For example, NEW_REQUEST can never be asserted two clocks in a row. These invalid state changes are shown as blank areas in the table.

Figure 8. Request Queue State Logic



4.4 SDRAM Address MUX

The SDRAM address MUX is responsible for selecting the proper information to output to the SDRAM on its multiplexed address bus. It also contains a column address counter for generation of linear addresses during SDRAM burst accesses.

Synchronous DRAM uses a multiplexed address bus and multi-bank memory structure. The LRH board uses four 64Mbit, 16-bit wide SDRAMs which utilize a 12-bit row address, an 8-bit column address and a 2-bit bank address. The bank address is conveyed to the SDRAM via the 2-bit BA[1:0] bus, whereas the row and column address are conveyed by the 12-bit A[11:0] bus in a multiplexed fashion depending on the SDRAM command being issued. Address bit A10 has a special meaning for certain SDRAM commands. During SDRAM READ or WRITE commands, it indicates whether or not to perform an automatic precharge upon command completion. During a PRECHARGE command, it differentiates between precharging the selected bank or precharging all SDRAM banks. The 80200 processor address mapping to SDRAM for the LRH board is shown in Table 2.

Table 2. 80200 Processor to SDRAM Address Mapping

80200 Processor Address [24:0]																							
24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bank Addr[1:0]		Row Addr[11:0]												Column Addr[7:0]								Not Used		

As previously discussed, the 80200FCC memory controller does not use the internal SDRAM burst address counter; therefore it must increment addresses manually for burst accesses. This is accomplished with a two bit counter under control of the SDRAM Request Processor. The first address of a burst is loaded into the counter during the first SDRAM READ or WRITE command. The address counter is linearly incremented each subsequent clock to produce the addresses for the remainder of the burst cycle. This counter is designed to support the Critical Word First protocol of the 80200 processor.

A MUX under the control of the SDRAM Request Processor is used to select the source for the SDRAM address input as shown in Table 3. Note that the PCH entry in the table is the precharge signal, a control signal generated by the SDRAM Request Processor. The SDRAM bank address BA[1:0] bus is non-multiplexed and always receives 80200 processor address [24:23].

Table 3. SDRAM Address Multiplexer

SDRAM Commands	SDRAM A[11:0] Source											
	11	10	9	8	7	6	5	4	3	2	1	0
Set Mode (MODE)	SDRAM Mode[11:0]											
Bank Activate (ACT)	Row Addr[11:0]											
READ or WRITE (burst access)	0	PCH	0	0	Column Address[7:2]						Burst Cntr[1:0]	
All other	0	PCH	0	0	Column Address[7:0]							

4.5 Refresh Controller

To minimize the impact of DRAM refreshes on processor requests, two types of refresh requests are issued to the SDRAM Request Processor: a normal refresh request and a high priority refresh request. A normal refresh request is given lower priority than all other SDRAM access requests and will be postponed until there are no other SDRAM requests pending. A high priority refresh request is generated when a certain number of pending refresh requests are waiting and will be serviced immediately (or as soon as the current SDRAM access is complete).

The Refresh Controller consists of a request timer and a pending request counter. The 8.33 MHz-based timer is configured to roll over when it reaches a count equivalent to the SDRAM refresh period (15.6 usec for the LRH board). The counter keeps track of how many refresh requests are pending. The refresh counter is incremented every time the refresh timer rolls over and decremented when the SDRAM Request Processor has serviced a request as indicated by the SRP2RFSH_RFSH_DONE signal. A low priority refresh request is issued whenever the refresh counter is non-zero. When the refresh counter reaches its maximum count of 8, a high priority refresh is requested. A high priority refresh request has the highest priority of all requests and is guaranteed to be performed prior to the next refresh request being issued. There is therefore no need for protection against the refresh timer rolling over while a high priority refresh request is pending. The refresh request signals both come directly from flip-flops to minimize the delay in reaching the SDRAM Request Processor.

The refresh timer is configured for a longer timebase during powerup in order to assist in keeping track of the SDRAM initialization period. When the refresh counter reaches its maximum count for the first time, the refresh timer timebase changes over to the normal SDRAM refresh period. The auto refresh commands issued during SDRAM initialization are conveniently generated by the eight refresh requests which were accumulated during the initialization period.

4.6 SDRAM Request Processor

The SDRAM Request Processor, as its name implies, handles all SDRAM bus requests from the 80200 processor and Refresh Controller. It is responsible for initializing the SDRAM as well. Using information about the request type (Length, Read/Write, High/Low Priority Refresh, etc.), associated SDRAM bank status (ROW_IDLE, ROW_HIT), and with knowledge of the SDRAM access currently in progress, the SDRAM Request Processor determines the most efficient way to handle the request. It then issues the required commands to the SDRAM in the correct sequence and with proper timing, taking into account fixed and configurable SDRAM timing parameters. It also controls the 80200 processor DVALID signal to coordinate the flow of data between the SDRAM and 80200 processor across the data bus.

The SDRAM Request Processor also gets involved in non-SDRAM requests. When it detects a request for a peripheral bus device, it passes the request on to the Peripheral Request Processor for further processing. It also arranges for the transfer of data between the 80200 processor and the 64-bit wide peripheral bus data I/O flip-flops. Invalid bus requests are handled by the SDRAM Request Processor by asserting the ABORT signal at the appropriate time to terminate the cycle.

Table 4 shows the categorization of all possible 80200 processor requests for a 64-bit wide data bus. Each request is categorized based on the cycle type and length, on address decode, and on associated SDRAM bank state information. Note that only a single bit of length information is examined to determine if a single or multiple cycle access is required to satisfy the request. Read requests can be handled in one data bus cycle (request length of 1, 2, or 4 bytes) or four cycles (request length of 32 bytes). Write requests involve a single data bus cycle (request length of 1, 2, 4, or 8 bytes) or two cycles (request length of 16 bytes). The masking of individual bytes within a 64-bit DWORD is done entirely by the 80200 processor byte enable signals which are driven during the data transfer cycle(s).

Table 4. Bus Request Types from the 80200 Processor

Request Category	Request Description	80200 Processor Info		Addr Decode		Bank State	
		Type	Len (bytes)	Sdram	Valid	Hit	Idle
RD1_IDLE	Read 1 Dword, Idle bank	Rd	1,2,4	1	1	0	1
RD1_HIT	Read 1 Dword, Open row	Rd	1,2,4	1	1	1	-
RD1_MISS	Read 1 Dword, Diff Row	Rd	1,2,4	1	1	0	0
RD4_IDLE	Read 4 Dwords, Idle bank	Rd	32	1	1	0	1
RD4_HIT	Read 4 Dwords, Open row	Rd	32	1	1	1	-
RD4_MISS	Read 4 Dwords, Diff row	Rd	32	1	1	0	0
WR1_IDLE	Write 1 Dword, Idle bank	Wr	1,2,4,8	1	1	0	1
WR1_HIT	Write 1 Dword, Open Row	Wr	1,2,4,8	1	1	1	-
WR1_MISS	Write 1 Dword, Diff row	Wr	1,2,4,8	1	1	0	0
WR2_IDLE	Write 2 Dwords, Idle bank	Wr	16	1	1	0	1
WR2_HIT	Write 2 Dwords, Open row	Wr	16	1	1	1	-
WR2_MISS	Write 2 Dwords, Diff row	Wr	16	1	1	0	0
RD_PBUS	Read Periph Bus	Rd	1,2,4,32	0	1	-	-
WR_PBUS	Write Periph Bus	Wr	1	0	1	-	-
INVALID	Invalid request	-	-	-	0	-	-

4.6.1 SDRAM Initialization

After powerup, the SDRAM must be initialized before data accesses are allowed. This involves a sequence of PRECHARGE and AUTO REFRESH commands followed by a write to the mode register. The SDRAM Request Processor performs these tasks without any assistance from the processor following the release of the board reset. 80200 processor bus requests are held off until SDRAM initialization is complete. The sequence of commands issued to the SDRAM during initialization can be seen in Figure 14.

As discussed in Section 3.3, the SDRAM is configured for a burst length of 1 and the CAS latency is configurable. The values stored in the mode register are as follows:

- Burst Length assigned fixed value of “010” => Burst Length = 1
- Burst Type assigned fixed value of “0” => Sequential
- CAS Latency assigned a configurable value of either “010” => CAS Latency = 2 or “011” => CAS Latency = 3
- All other setting treated as reserved and set to “0”

4.6.2 SDRAM Commands

The SDRAM Request Processor will only issue the commands shown in Table 5 to the SDRAM. Note that several available SDRAM commands are not used. For example, the READ or WRITE WITH AUTOPRECHARGE commands are never issued since rows are always kept open after an access.

Table 5. SDRAM Commands used by 80200FCC

Command	Description	CS#	RAS#	CAS#	WE#	A[10]	A[11:0]	BA[1:0]
DSEL	Device Deselect (NOP)	H	-	-	-	-	-	-
ACT	Bank Activate	L	L	H	H	-	ROW	BANK
PRE	Precharge Selected Bank	L	L	H	L	L	-	BANK
PALL	Precharge All Banks	L	L	H	L	H	-	-
READ	Read Data	L	H	L	H	L	COL*	BANK
WRITE	Write Data	L	H	L	L	L	COL*	BANK
REFRESH	Auto Refresh	L	L	L	H	-	-	-
MODE	Set Mode Register	L	L	L	L	L	MODE	-
* Column address or column address & burst counter. See Table 3.								

4.6.3 Finite State Machine

The SDRAM Request Processor is composed mainly of a large finite state machine consisting of 31 different states. Table 6 shows each state of the finite state machine along with a brief description of what occurs in each state.

Table 6. SDRAM Request Processor State Description

State Name	General State Description	Action
RESET	Held here until board reset released	None
NOP	First command state for SDRAM init	Issue NOP command to SDRAM
INIT_WAIT	Wait for 200 Usec before continuing SDRAM Init	None
PALL_WAIT	Wait for Tras before issuing PALL	None
PALL	Precharge All SDRAM banks	Issue PALL command to SDRAM Start Trp timer
PALL_END	Wait for Trp after PALL	None
CBR	Auto Refresh SDRAM	Issue CBR command to SDRAM Indicate completion of refresh request
CBR_END	Wait for Trfc after CBR	None
MRS	Set SDRAM Mode Register	Issue MRS command to SDRAM Set indication of SDRAM initialized
MRS_END	Wait for Tmrd after MRS	None
IDLE	Waiting for request	None
PRE_WAIT	Wait for Tras before issuing PRE	None
PRE	Precharge SDRAM bank	Issue PRE command to SDRAM Start Trp timer
PRE_END	Wait for Trp after PRE	None
ACT	Activate row in SDRAM bank	Issue ACT command to SDRAM Issue DVALID for write request
ST1_RD	Preparing for READ command	None
ST2_RD	Preparing for READ command	None
RD1	Single cycle read of SDRAM	Issue READ command to SDRAM Issue DVALID Pop Request Queue Start turnaround counter
RD4_1	1st of multi-cycle read of SDRAM	Issue READ command to SDRAM Issue DVALID
RD4_2	2nd of multi-cycle read of SDRAM	Issue READ command to SDRAM Issue DVALID
RD4_3	3rd of multi-cycle read of SDRAM	Issue READ command to SDRAM Issue DVALID
RD4_4	4th of multi-cycle read of SDRAM	Issue READ command to SDRAM Issue DVALID Pop Request Queue Start turnaround counter
RD2WR	delay between read and write	None
ST1_WR	Preparing for WRITE command	None Assert DVALID

Table 6. SDRAM Request Processor State Description (Continued)

State Name	General State Description	Action
ST2_WR	Preparing for WRITE command	None Assert DVALID for multi-cycle write
WR1	Single cycle write to SDRAM	Issue WRITE command to SDRAM Possibly Assert DVALID for next write Pop Request Queue
WR2_1	1st of multi-cycle write to SDRAM	Issue WRITE command to SDRAM Possibly Assert DVALID for next write
WR2_2	2nd of multi-cycle write to SDRAM	Issue WRITE command to SDRAM Possibly Assert DVALID for next write Pop Request Queue
PERIPH	Handle Peripheral Request Processor	Send request to Peripheral Request Proc
PERIPH_WAIT	Wait for Peripheral Request Processor to complete request	Output peripheral bus data to 80200 processor as requested by Periph Request Proc Assert DVALID for data
PERIPH_END	Delay after Periph Request	None

The enforcement of configurable SDRAM timing parameters is handled by a combination of timers, finite state machine logic, and delayed control signals. Timers associated with each bank of the SDRAM keep track of how long a row has been active in order to enforce the Ras Active (Tras) timing constraint. Other timers are maintained to enforce the Ras Precharge (Trp) constraint and to manage bus turn-around timing between read and write cycles. The Ras to Cas delay (Trcd) is handled by adding an extra state between the issuing of the ACTIVATE and READ/WRITE commands. Signals affected by the Cas Latency setting (DVALID, CWF, nBE[7:0]) are output by the finite state machine as if Cas Latency (Tcl) is set for 2 clocks. When Cas Latency is configured for 3 clocks, these signals are delayed one additional clock before being issued.

In general, SDRAM read and write requests are handled by first opening the memory row in the particular SDRAM bank and then issuing one or more READ or WRITE commands. If the row is already open, only the READ or WRITE command(s) is issued. If the bank is idle, the row is opened by issuing an ACTIVATE command to the SDRAM. If a different row is open for the particular SDRAM bank, the bank must first be closed by issuing a PRECHARGE command before the new row can be activated. The state machine guarantees that rows are open a minimum of Tras before precharging by looking at timer outputs from the bank state/open row store logic.

In the case of a SDRAM read request, only single cycle or 4-cycle requests are issued by the 80200 processor. Read data flows from the SDRAM to the 80200 processor either two or three clocks after the SDRAM READ command(s) are issued depending on how SDRAM CAS latency was configured. SDRAM Byte enable signals and the 80200 processor DVALID signal are both asserted 2 clocks prior to the clock in which SDRAM data is to be driven. For read accesses, the SDRAM byte enable signals are driven active by the 80200FCC memory controller so that all SDRAM bytes are driven no matter how many bytes of the 64-bit bus were requested. It is possible for the SDRAM Request Processor to issue an SDRAM READ command every clock if the Request Queue is kept full and SDRAM accesses are all to open rows. A maximum sustained throughput of 800 Mbyte/sec. is therefore obtainable.

In the case of an SDRAM write request, only single cycle or 2-cycle requests are issued by the 80200 processor. Write data flows from the 80200 processor to the SDRAM during the same clock in which the WRITE command is issued to the SDRAM. SDRAM byte mask signals come directly from the 80200 processor (nBE) to select which bytes of the 64-bit bus are written. The 80200

processor DVALID signal is issued two clocks before the data cycle occurs. Because the SDRAM Request Processor looks ahead in the Request Queue, it can maintain an uninterrupted flow of two-cycle SDRAM write requests with a theoretical bandwidth of 800 Mbyte/sec. For single cycle write requests, two SDRAM WRITE commands can be issued every three clocks for a theoretical peak bandwidth of 533 Mbyte/sec. This rate is not sustainable since the 80200 processor takes two clocks to issue each request.

Low priority refresh requests are handled when no other 80200 processor requests are pending. If multiple refresh requests are pending, the SDRAM Request Processor will burst them out as long as no new 80200 processor requests arrive. High priority refresh requests are handled as soon as any in-process SDRAM accesses are complete.

The following figures show state transition logic in waveform format for different categories of requests. Figure 9 is an example of typical 80200 processor external bus activity. It shows an SDRAM write access (to an idle SDRAM bank) followed by a 32-byte SDRAM read. It includes detailed activity on the 80200 processor request and data buses. Figure 10 through Figure 13 show state transitions between different types of 80200 processor requests. Also shown are the SDRAM timing parameters managed by the SDRAM Request Processor. State transitions during SDRAM initialization are presented in Figure 14. The following diagrams are presented with SDRAM timing parameters configured as follows: Tcl = Trp = Trcd = 2 clocks, Tras = 6 clocks. When looking at these figures, keep in mind that the commands issued to the SDRAM are clocked into I/O Buffer Flip Flops and are therefore not seen by the SDRAM until one clock later (this can be clearly seen in Figure 9).

Figure 9. Typical 80200 Processor SDRAM Accesses

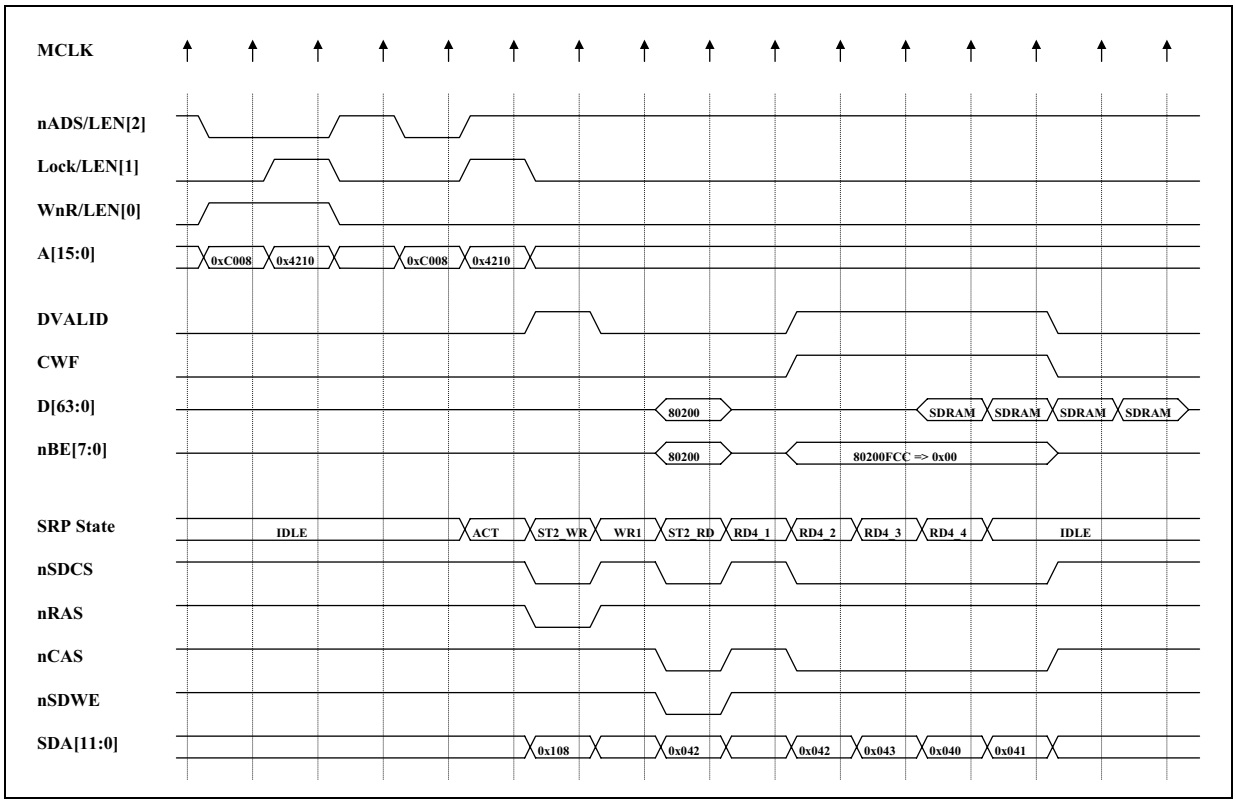
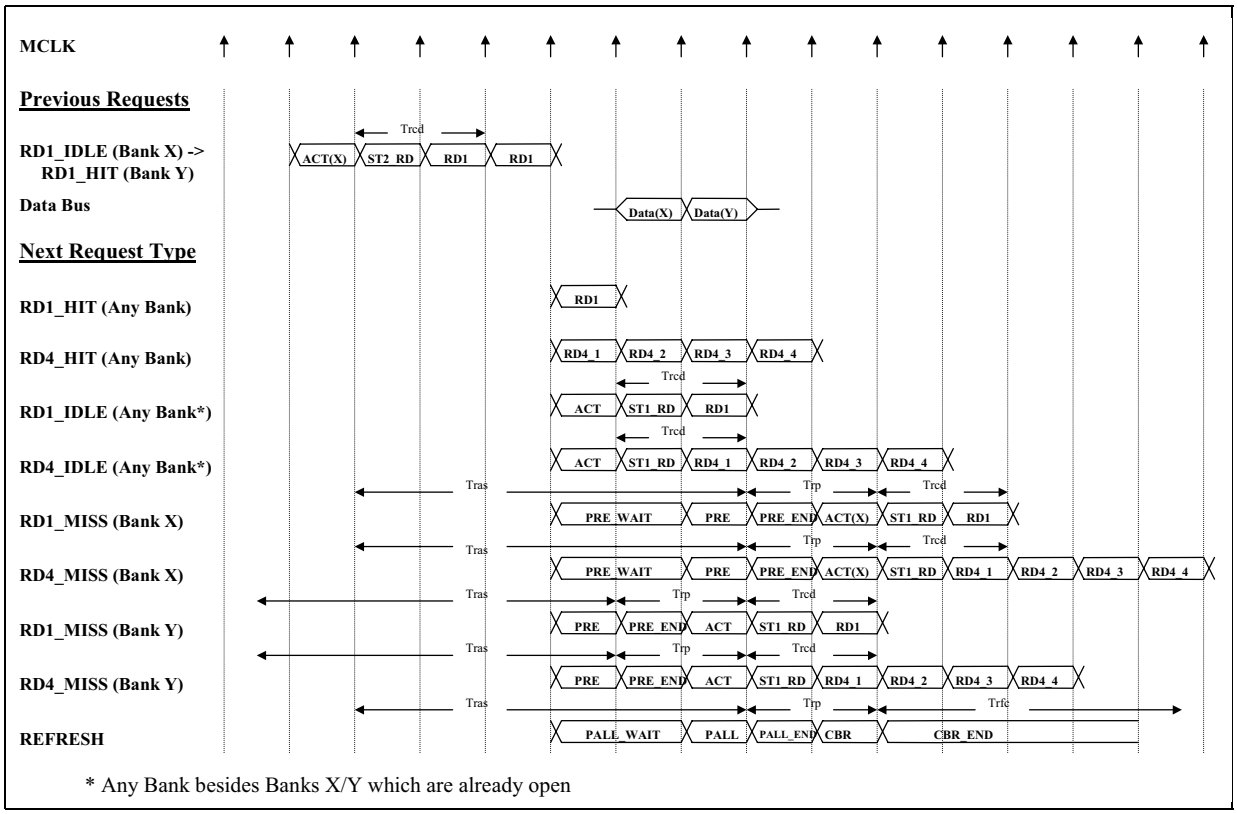
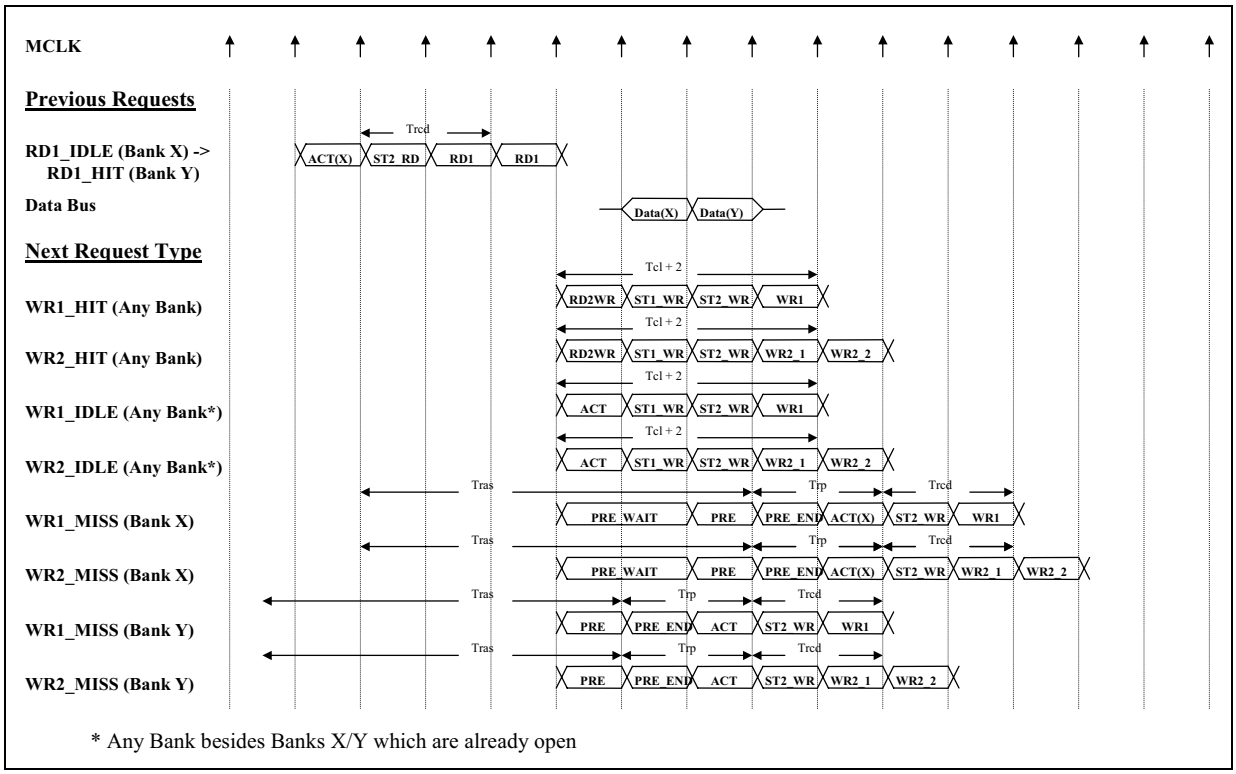


Figure 10. Read Followed by Read Cycle



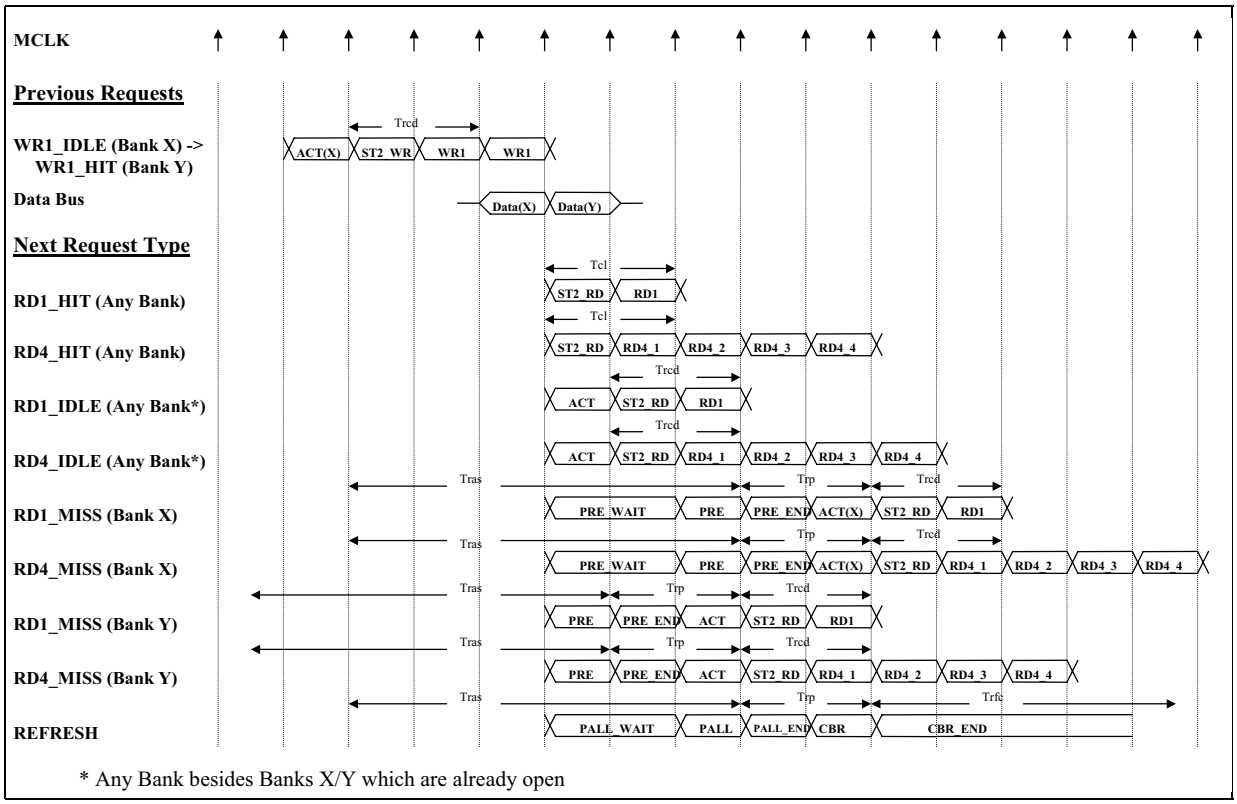
* Any Bank besides Banks X/Y which are already open

Figure 11. Read Followed by Write Cycle



* Any Bank besides Banks X/Y which are already open

Figure 12. Write Followed by Read Cycle



* Any Bank besides Banks X/Y which are already open

Figure 13. Write Followed by Write Cycle

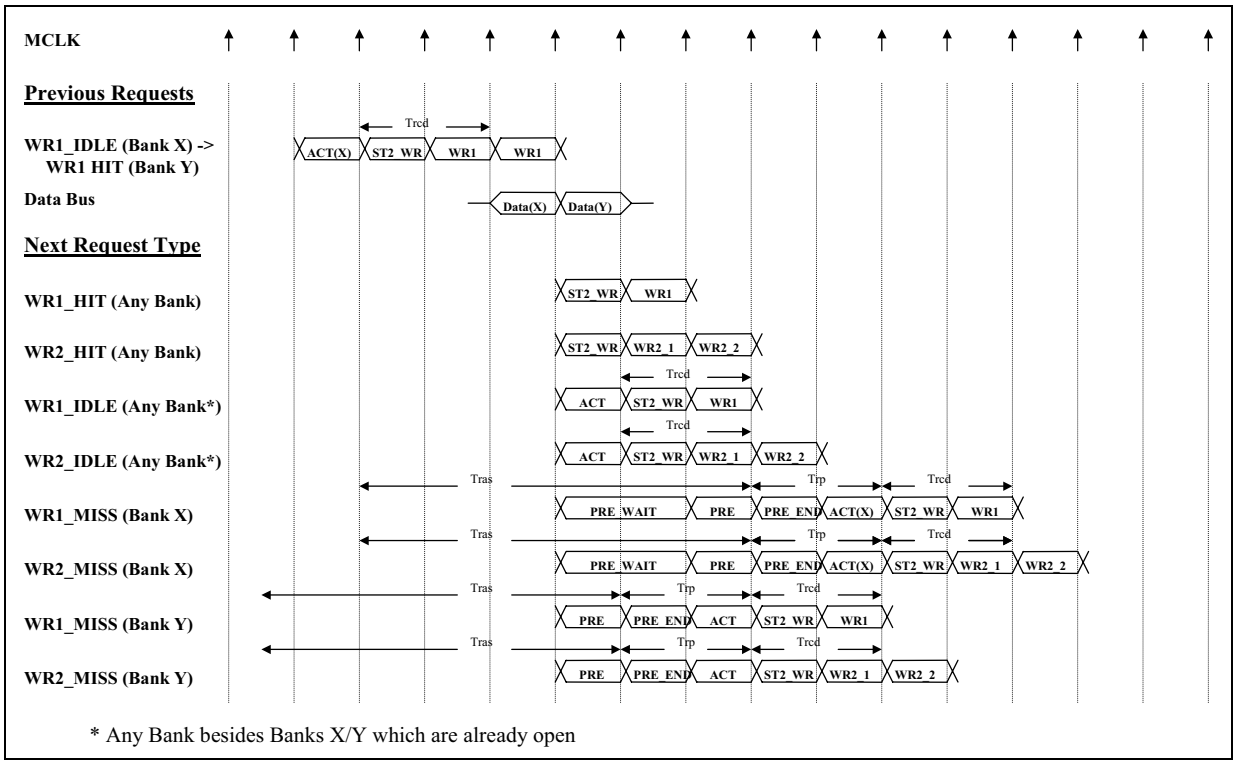
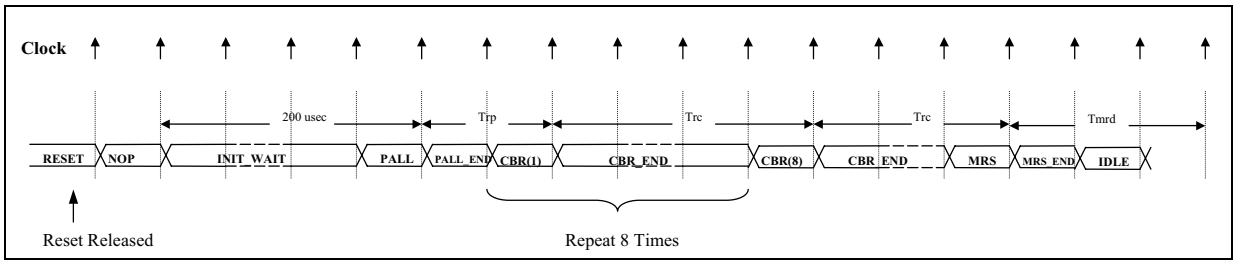


Figure 14. SDRAM Initialization Sequence



4.7 Peripheral Bus Interface

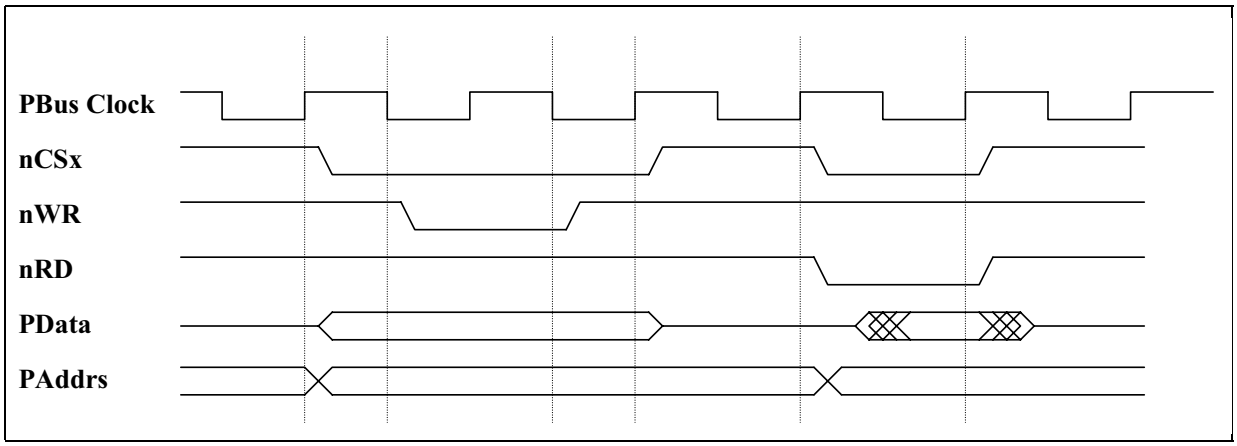
A lower performance bus, separate from the 80200 processor/SDRAM data bus, was added to the 80200FCC memory controller for connection of slower memory and other peripherals. Because of the many different devices that could be connected to this bus, it was designed to be very configurable. On the LRH evaluation board, the peripheral bus consists of an 8-bit data bus, four chip selects, read and write strobes, and 22-bits of address for a maximum of 4 Mbytes of address space per chip select. Associated with each chip select is a fixed delay required for the peripheral to complete an access. Data bus width, the number of chip selects, wait state delays, etc. are all configurable by changing VHDL constants and re-synthesizing the design.

The 80200FCC performs data conversion between the 80200 processor and peripheral devices by steering data between the 8-bit peripheral bus and one of the eight byte lanes of the 80200 processor data bus. As shown in Figure 4, write data is output to the peripheral bus through a byte-

wide, 8-to-1 MUX. Read data from the peripheral bus is latched into one of the 8-bit latches associated with each byte-lane of the 80200 processor data bus. Byte lane selection is determined by the length and address of the peripheral bus request. The 64-bit wide 80200 processor data bus is extended into the 80200FCC for future connection of higher performance interfaces such as PCI. This internal 64-bit bus is shared by many internal devices using 3-state buffers. This is a very efficient way for many devices to share a data bus in an FPGA.

The peripheral bus interface of the 80200FCC is designed to handle any length read request (including 32-byte burst reads) but only single byte write requests. Multi-byte peripheral read requests are performed by executing multiple peripheral accesses. Data is latched in up to eight byte lane latches and returned in a single 80200 processor data bus cycle. For a 32-byte burst cycle, this process is repeated four times returning eight bytes of data on each 80200 processor data bus cycle. For multi-byte reads, the peripheral bus address is incremented by a counter inside the 80200FCC. This counter was designed to support the 80200 processor Critical Word First protocol. Due to the lack of any peripherals on the LRH evaluation board that could take advantage of it, multi-byte peripheral write requests are not supported by the 80200FCC. Figure 15 illustrates a typical single byte peripheral bus write followed by a single byte peripheral bus read.

Figure 15. Typical Peripheral Bus Cycles



To minimize the impact of peripheral bus write cycles on 80200 processor SDRAM requests, the 80200FCC implements write posting to the peripheral bus. When the SDRAM Request Processor detects a peripheral bus write request, it passes the request to the Peripheral Request Processor, schedules the 80200 processor write data cycle, and then moves on to the next request in the Request Queue (assuming the Peripheral Request Processor is not busy with a previous request). No special handling of peripheral bus read requests was implemented in the 80200FCC although there is potential for improvement in this area. The SDRAM Request Processor simply waits for the Peripheral Request Processor to complete the entire peripheral read access before moving on to the next request. While waiting, the SDRAM Request Processor has the responsibility of scheduling the transfer of peripheral bus data back to the 80200 processor across the 80200 processor/SDRAM data bus when requested to do so by the Peripheral Request Processor. One possible performance improvement to the 80200FCC would be to perform any pending SDRAM refresh requests while waiting for a peripheral bus read cycle to complete.

The Peripheral Request Processor consists of a small finite state machine along with additional support logic including cycle counters, additional address decode logic, request length decode logic, etc. The Peripheral Request Processor controls the generation of all the peripheral bus control signals as well as coordinating efforts with the SDRAM Request Processor. All peripheral bus logic is based on an 6.25 MHz peripheral bus clock. This clock is synchronous to the 100 MHz memory bus clock but still requires some special handling of signals that traverse the two time

domains. Signals generated in the 100 MHz time domain must be latched until they are sampled by the logic operating in the peripheral bus clock domain. Conversely, signals generated in the peripheral bus clock domain must be converted to a single 100 MHz wide control signal when destined for the 100 MHz domain.

Coordination between the SDRAM Request Processor and Peripheral Request Processor is done with the assistance of several “handshake” signals. The SRP2PRP_PBUS_REQ signal is asserted by the SDRAM Request Processor to indicate that a peripheral bus request is ready for processing. Upon detecting an assertion of this signal, the Peripheral Request Processor immediately captures the request in its own internal latches and begins processing the request. The PRP2SRP_BUSY signal indicates that the Peripheral Request Processor is busy with a request and is unable to accept and further requests. This is used to insure that only a single peripheral write cycle is posted at a time. While handling read requests, the Peripheral Request Processor accumulates data in the 8-bit wide latches. Once data accumulation is complete, the Peripheral Request Processor latches the data into the 64-bit wide 80200 processor data output Flip Flops. The PRP2SRP_DATA_OUT_REQ or PRP2SRP_DATA_OUT_REQ_DONE signal is then asserted to tell the SDRAM Request Processor to return the data to the 80200 processor. Assertion of the PRP2SRP_DATA_OUT_REQ_DONE signal also indicates that the read cycle is complete.

5.0 Conclusion

In conclusion, this application note describes the design and implementation of a high performance memory controller for the Intel 80200 processor. This design was implemented in an FPGA and can be found on the 80200 processor LRH evaluation board. While certain architectural trade-offs were made to achieve high speed operation in an FPGA implementation, the basic design can be adapted to optimize speed and size for ASIC or full custom implementation as well. Design considerations presented in this document should prove helpful not only for hardware engineers using or modifying the 80200FCC and LRH designs, but for any hardware engineer developing memory controllers and peripheral interfaces for the Intel 80200 processor.