XAPP266 (1.0) February 27, 2002

Synthesizable FCRAM Controller

Author: Curtis Fischaber

Summary

This application note describes how the Virtex™-II architecture can be leveraged to implement a Double Data Rate (DDR) Fast Cycle RAM (FCRAM) controller.

Introduction

Typical DRAM memories are based on a common memory core and cell array. Starting with this same core technology, slight changes in the peripheral logic circuitry have allowed a wide range of higher performance memories to be created, such as EDO, SDRAM, DDR SDRAM, and Direct Rambus DRAM (RDRAM).

However, in continuing to use this same core technology, memories have also inherited the same limitations that exist in the core architecture. So rather than try to continue to target this interface, FCRAM boosts internal performance by redesigning the internal DRAM core. This redesigned core decreases memory latency as well as power consumption. FCRAM therefore offers a great replacement for traditional memory technologies any time large memory densities, high effective bandwidth, or low power consumption are required. Some typical uses of FCRAM range from servers and hardware accelerators to networking devices. FCRAM™ is a trademark of Fujitsu Ltd., Japan.

This application note describes a FCRAM controller design implemented in Virtex-II devices. A brief overview of FCRAM basics are presented, followed by a detailed description of the implemented controller.

DDR FCRAM Review

Basics

This section is a general overview for those unfamiliar with the FCRAM interface and operation. Those already familiar with this memory can go directly to the **FCRAM Controller Design** section.

FCRAM devices operate at a core voltage of 2.5V with SSTL-II I/O. This application note targets the first revision (indicated by FCRAM speed grades -22/-24/-30), which have a maximum clock frequency of 154 MHz. These devices are offered by Fujitsu and Toshiba in 256 Mb densities with a x8 or a x16 configuration (these refer to the number of data (DQ) pins per device).

FCRAM uses a DDR interface that transfers data on both the rising and falling edge of the clock (CLK). Because this effectively doubles the data throughput of the device while maintaining the same clock frequency, this technique has become quite popular in many DRAMs. The rising (positive) clock edge is defined at the point in which `CLK` transitions High and $\overline{\texttt{CLK}}$ transitions Low.

FCRAMs are addressed by row (upper address), column (lower address), and bank (typical FCRAMs have four banks). A memory access (a read or a write operation) is burst oriented, meaning that a memory access starts at a selected bank and address and continues for a set number in a programmed sequence.

The FCRAM control logic consists of two signals, $\overline{\text{CS}}$ and FN. Each FCRAM operation is determined by two consecutive command inputs. The first command determines the read

(RDA) or write (WRA) branch of the controller state machine. Following an RDA command, either a read command (LAL) or a mode register set (MRS) command can be executed. Following a WRA command, either a write command (LAL) or a memory refresh (REF) command can be executed.

An overview of the FCRAM state machine is shown in Figure 1. This diagram has been simplified for single bank operation. The dashed lines indicate an automatic sequence.

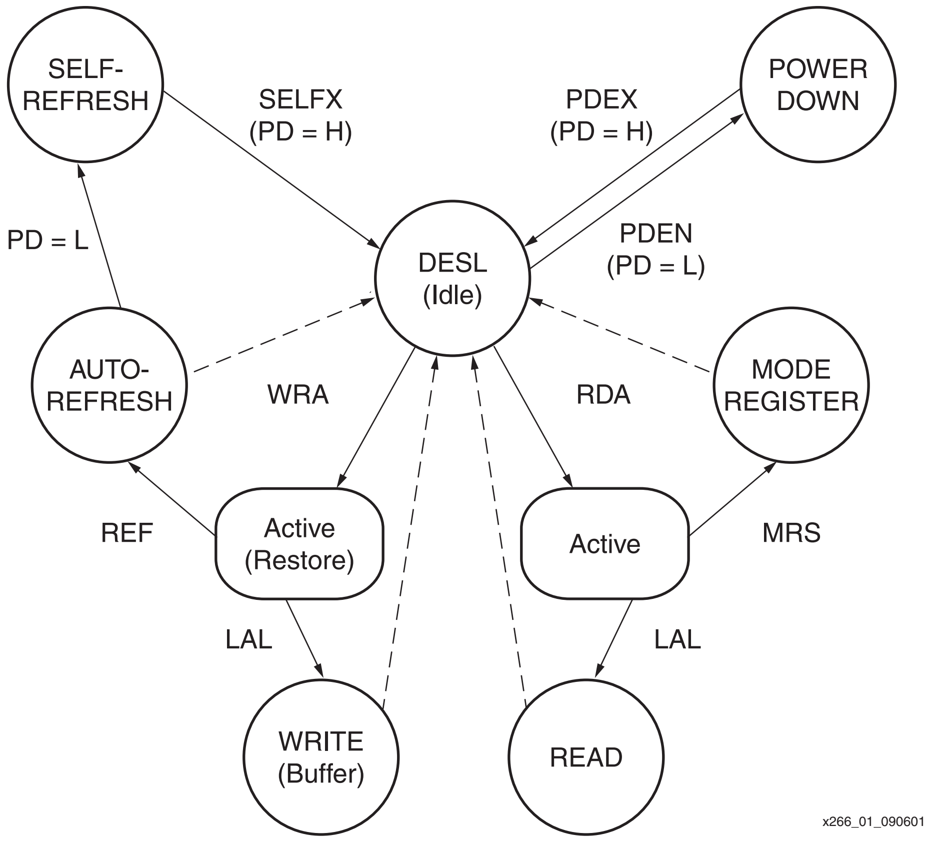

Figure 1: **FCRAM State Machine Diagram**

All FCRAM address and command signals are latched in by the FCRAM on the positive edge of clock. Similar to conventional DDR DRAMs, FCRAM uses a bidirectional data strobe signal (DQS). This strobe is typically used as the clock to capture the data during both reads and writes. During a memory read, the strobe is sent edge aligned with the data from the FCRAM. Therefore, it is the responsibility of the controller to delay the strobe in order to capture the data. During a memory write, the controller must deliver the strobe center-aligned with the data at the FCRAM pins. The FCRAM device then internally matches the delays between the DQ and DQS to capture the data. The FCRAM specification dictates that for every byte of data (eight DQ lines) there is a DQS.

Read Operation

The FCRAM read command (Figure 2) is initiated by the RDA command. This command is issued by asserting $\overline{CS}$ Low and asserting FN High. The target bank and upper address are activated during the RDA command. On the following clock cycle, the LAL command is given. This command is issued by deasserting $\overline{CS}$ High. The lower address is activated during the LAL command.

Data is available from the controller CAS latency (CL) cycles after the read command is issued. The rising and falling edges of DQS indicate valid data on the DQ bus. DQS continues to toggle until the burst length is complete.

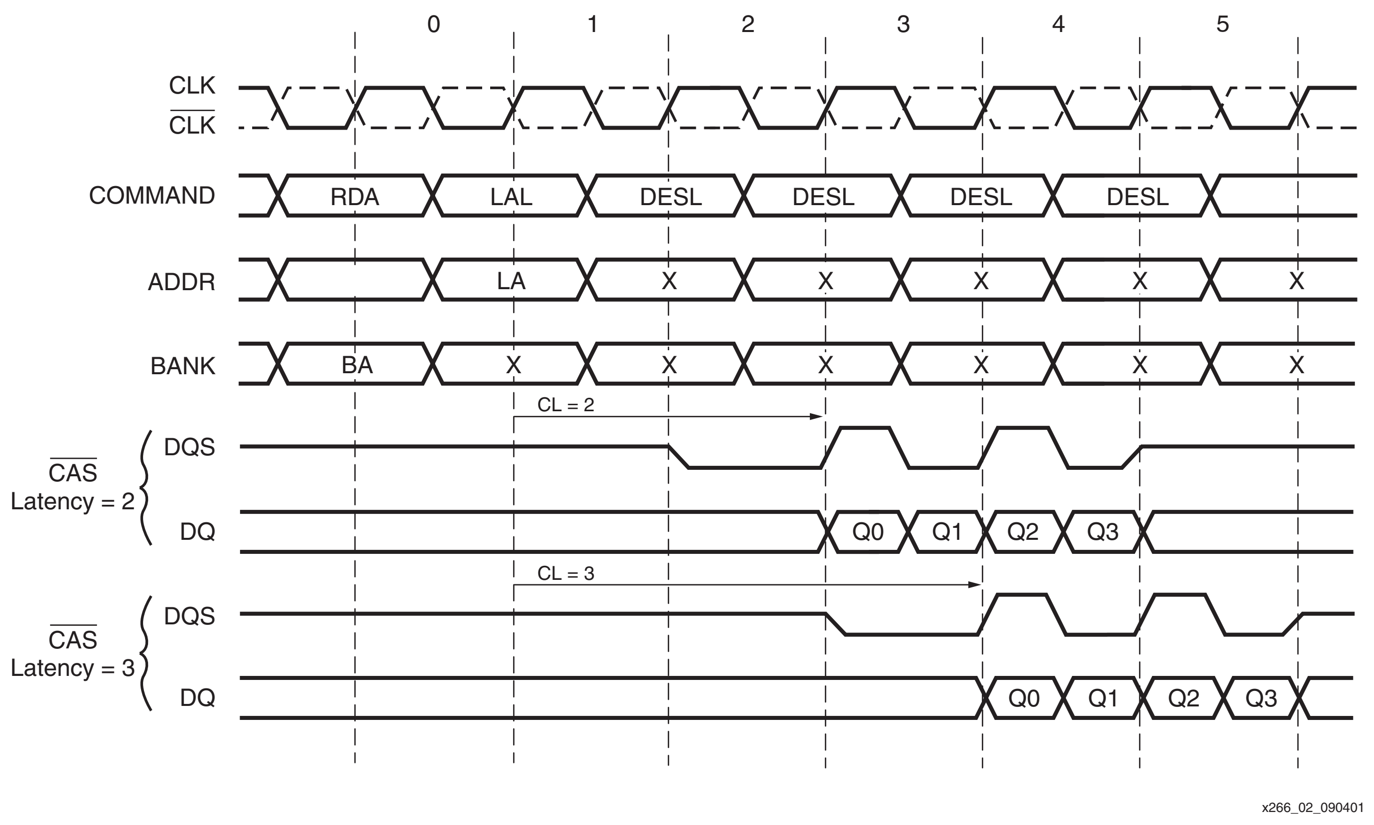

Figure 2: **Read Operation Timing**

Write Operation

The FCRAM write command (Figure 3) is initiated by the WRA command. This command is issued by asserting $\overline{\text{CS}}$ Low and deasserting FN Low. The target bank and upper address are activated during the WRA command. On the following clock cycle, the LAL command is given. This command is issued by deasserting $\overline{\text{CS}}$ High. The lower address is activated during the LAL command.

The data must be presented T_{DS} (data-in setup time) prior to the data strobe (DQS) edge. The first rising edge of DQS typically occurs write data latency (WL) cycles after the LAL command has been issued. The remaining data inputs must be supplied on the subsequent falling and rising edge of DQS until the burst length is complete.

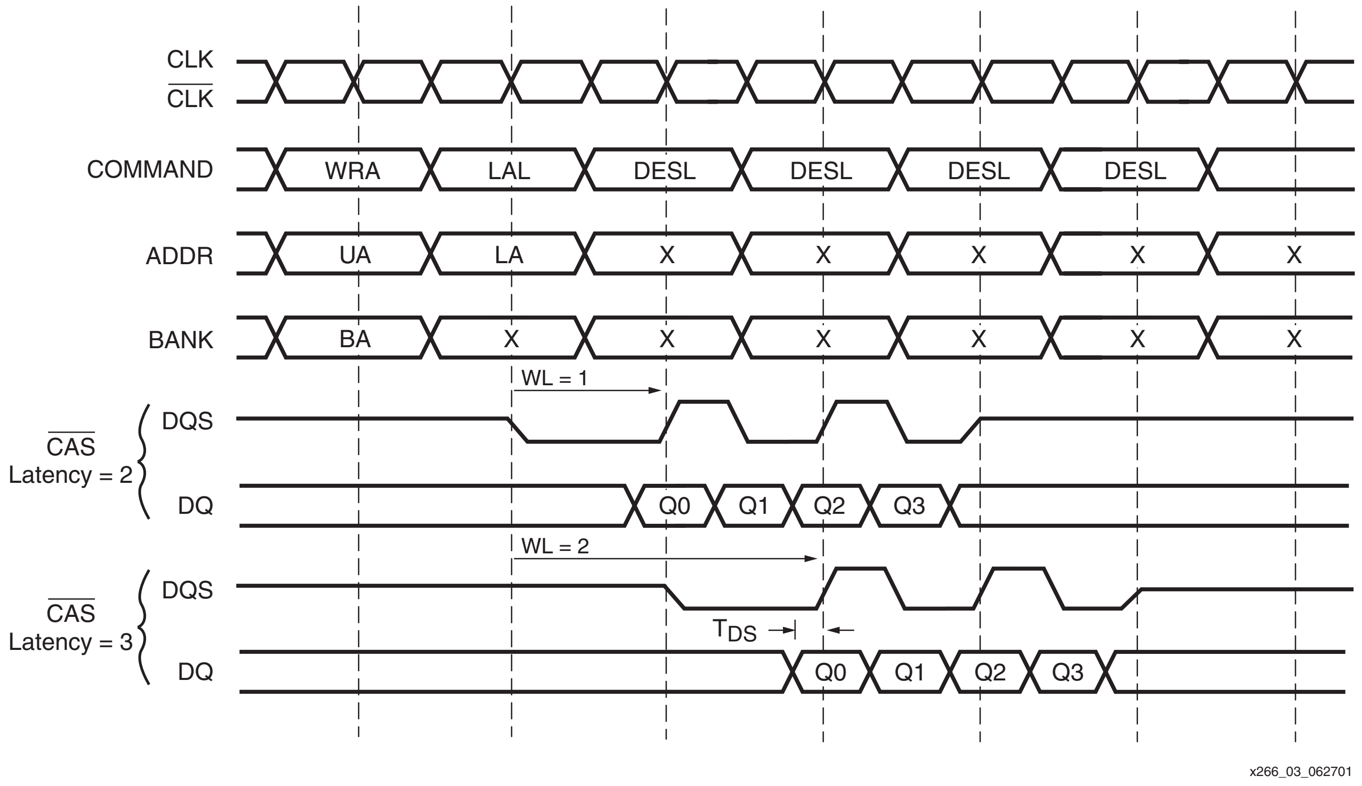

Figure 3: **Write Operation Timing**

Mode Register Set (MRS)

Mode registers define the specific FCRAM mode of operation. On power up, the mode registers are undefined and must be programmed. Once programmed, the register contents are maintained until the power is lost, or until another MRS command is issued and its contents updated.

The FCRAM MRS mode is initiated by the RDA command. This command is issued by asserting $\overline{\text{CS}}$ Low and asserting FN High. In setting the mode registers, the bank and address inputs are ignored during the RDA command. On the following clock cycle, the MRS command is given. This command is issued by asserting $\overline{\text{CS}}$ Low. The values for configuring the FCRAM are issued on the bank and address pins during the MRS command.

Typical FCRAMs have two mode registers, standard, and extended mode registers. Each of these mode registers must be separately configured and are selected based upon the bank input during the MRS command.

The address pins during the MRS command contain the desired FCRAM configuration information. The standard mode register configuration programs the Burst Length (A[2:0]), the Burst Type (A3), the CAS Latency (A[6:4]), and the Test Mode (A7).

The extended mode register configuration programs the DLL Enable (A0) and Output Driver Impedance Control (A1).

Burst Length (BL)

Read and write accesses to the FCRAM are burst oriented. This means that once a row and column are selected, a read or write command will progress across "burst length" number of columns. The burst length setting is programmable, and FCRAM memories support bursts of two and four locations.

Burst Type (BT)

Accesses within a burst can be programmed to be sequential or interleaved.

CAS Latency (CL)

During a read operation, CL is the delay in clock cycles between the registration of a read command (LAL) and the point at which data is valid. During a write operation, CL is the delay in clock cycles between the registration of the write branch selection (WRA) and the point at which data must be supplied to the FCRAM.

Test Mode

Test mode operation is reserved for supplier use. During normal operating mode, this bit should be set to zero.

DLL Enable

By setting this bit to zero, the DLL is enabled. Not all suppliers support disabling the DLL.

Output Driver Impedance Control

At the time of this application note, this field is currently not supported by FCRAM manufacturers. Therefore, this field should be set to zero.

Refresh

FCRAM is similar to other DRAMs in that the conventional capacitor cell is used, which requires refresh operations to be periodically performed in order to maintain the data written into the cell. FCRAM supports auto-refresh and self-refresh.

Auto-refresh is initiated with the WRA command (asserting $\overline{\text{CS}}$ Low and deasserting FN Low), followed by the REF command (asserting $\overline{\text{CS}}$ Low). If the $\overline{\text{PD}}$ pin is asserted Low within two clock cycles of the REF command, the FCRAM will enter the self-refresh state and remain there until $\overline{\text{PD}}$ is released.

FCRAM Controller Design

This section describes the design of a Virtex-II FCRAM controller. The controller has a user interface and an FCRAM interface. The design is written in Verilog and can be modified easily to fit different memory configurations.

The controller design has the following features:

- Programmable burst lengths of two and four
- Programmable CAS latency of two and three
- User initiated and controller initiated refresh
- Initialization sequence
- "Hidden" implementation of lower-level FCRAM functions
- Uses DQS to capture data during an FCRAM read
- Interfaces with DDR FCRAM up to 154 MHz in a Virtex-II -5 device.

Unlike traditional SDRAM, FCRAM does not provide the option to keep the bank/row open after a transaction. Instead, it automatically closes the row and precharges the bank after each access. Therefore, the user must issue a new read or write for each "burst-length" sized access.

Since the FCRAM can only operate with burst lengths of two or four, utilizing the maximum throughput of the FCRAM device could potentially be a rather large overhead for the user. Because these burst lengths are each completed with two clock cycles, the user would have to continually issue a new memory access command every other clock cycle. Before these commands could be issued, the user should check for possible FCRAM violations, such as bank collisions, read-write turnaround times and an expired refresh counter. Further information about these violations can be found in the **FCRAM Controller Operation** section.

Rather than require the user interface to be aware of these issues and take the appropriate action, the FCRAM controller simplifies the user interface by monitoring for these types of violations. The user then can simply issue a memory access command, a starting bank and address location, the number of transfers to be completed, and the FCRAM controller automatically handles all the details of the implementation.

Figure 4 is the top-level block diagram of the FCRAM controller. The module *fcram_cntrl* is the top level FCRAM controller block (Figure 5). It contains sub-modules such as the clock generation circuitry, the controller state machine, the refresh counter, the address counter, and the data path to the FCRAM. All signal references and descriptions are with respect to this module. The module *user_int* is a placeholder for the user interface. In this example, it passes on (either directly or through a pipeline stage) the system signals to the FCRAM controller.

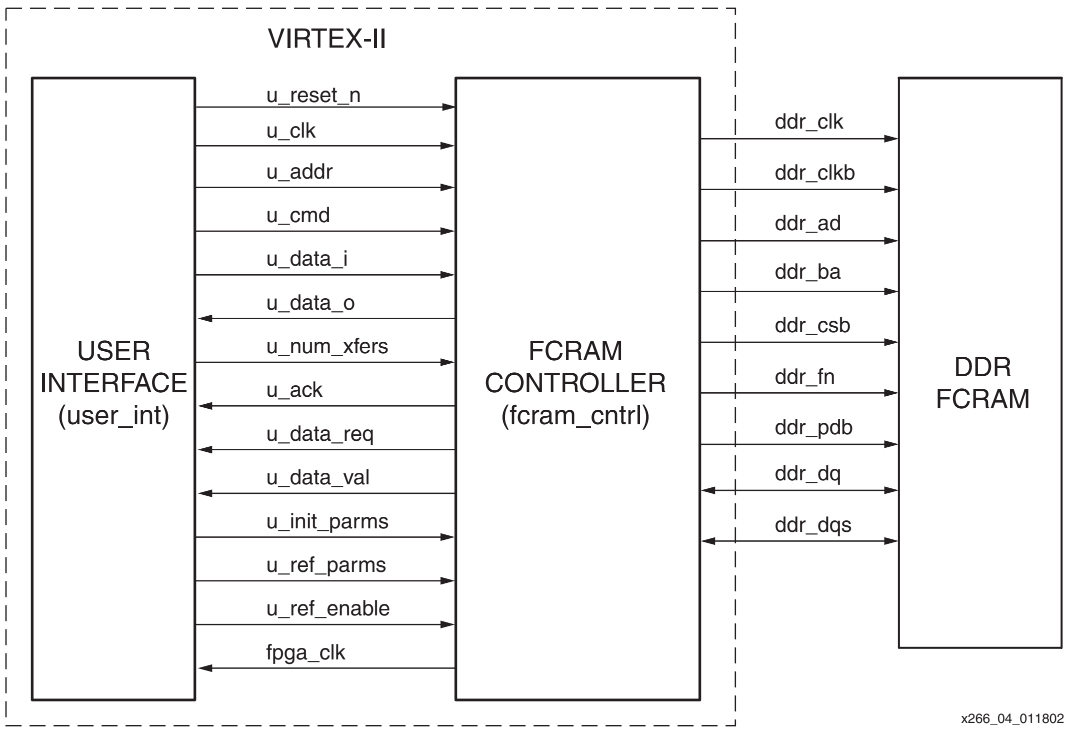

Figure 4: **Top Level Block Diagram**

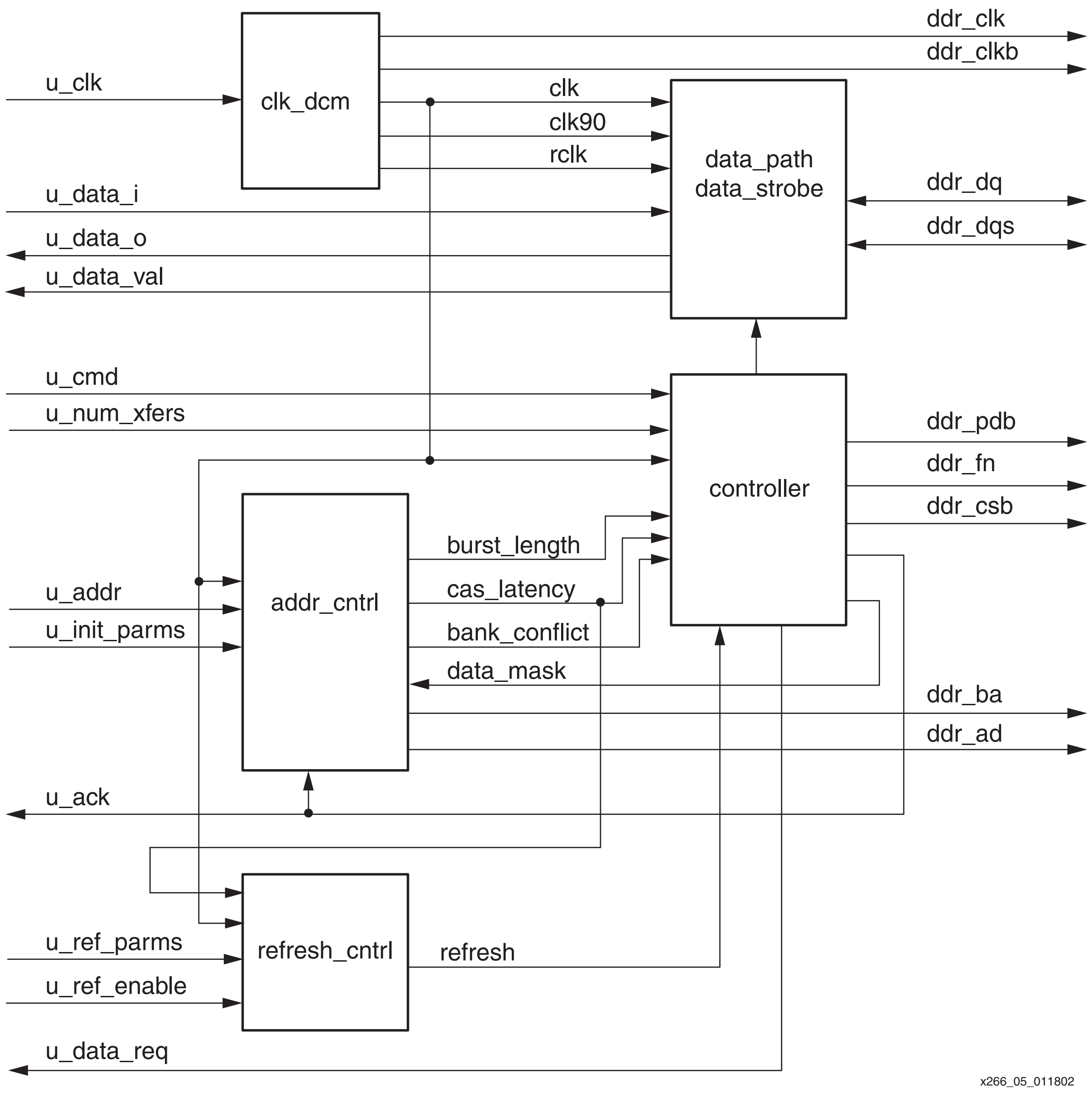

Figure 5: **fcram_cntrl Block Diagram**

FCRAM Controller Operation

Table 1 lists the user interface signals to the FCRAM controller. Table 2 lists the interface signals to the FCRAM devices.

Table 1: **User Interface to FCRAM Controller**

Pin Name	Direction	Width	Description
u_reset_n	In	1	Reset, active Low
u_clk	In	1	Input clock
u_addr	In	27	Address: u_addr = {bank(2), row(15), col(10)}
u_cmd	In	3	Command to be executed by controller [0 x x] NOP [1 0 0] Write Request [1 1 0] Read Request [1 0 1] Self Refresh Request [1 1 1] Auto Refresh Request
u_data_i	In	32	Write Data
u_data_o	Out	32	Read Data
u_num_xfers	In	4	Number of 32-bit data values to transfer
u_ack	Out	1	The controller has acknowledged a command issued by the user interface (guarantees execution)
u_data_req	Out	1	Write data value (`u_data_i`) is supplied by user
u_data_val	Out	1	Read data value (`u_data_o`) is valid
u_init_parms	In	10	Initialization parameters: u_init_parms = {CL(3),BL(3),TE,BT,DE,DIC}
u_ref_parms	In	20	Refresh interval parameters: u_ref_parms = {ref_burst_cnt[3:0], ref_interval_cnt[15:0]}
u_ref_enable	In	1	Enable automatic controller refresh
fpga_clk	Out	1	FCRAM Controller internal clock

Notes:
1. **MSB:** In this design, the higher order bits are the MSB. For example, `u_cmd[2:0] = 100` is a write request

Table 2: **Controller Interface to the FCRAM Devices**

Pin Name	Direction	Width	Description
ddr_clk	Out	1	Clock
ddr_clkb	Out	1	Inverted clock
ddr_ad	Out	15	Address
ddr_ba	Out	2	Bank address
ddr_csb	Out	1	Command
ddr_fn	Out	1	Command
ddr_pdb	Out	1	Command
ddr_dq	In/Out	16	Data
ddr_dqs	In/Out	2	Data Strobe

Data Bus Widths

This application note targets a x16 FCRAM device. However, the data widths are parametizable, and may be easily modified in the HDL code to support various memory configurations, including multiple FCRAM devices. Specific details about modifying memory configurations can be found in **Appendix A**. For illustrative purposes, this application note refers to a data transfer on the user interface as a 32-bit transfer. If the data width is changed, replace the 32-bit references found throughout this application note with the modified value.

No Operation (IDLE/DESL)

```
Set u_cmd = 0xx
```

This command keeps the controller in an IDLE state.

Initialization

The initialization sequence allows the user to set the mode registers of the FCRAM (Table 3). This initialization stage occurs automatically at power on, as well as every time the controller is reset. Therefore, the user is not required to issue commands such as Mode Register Set (MRS) and Extended Mode Register Set (EMRS).

During this sequence, the user interface supplies the initialization parameters to the FCRAM controller. Initialization parameters are passed from the user interface via `u_init_parms`, and are described as:

```
u_init_parms[9:0] = {CL(3),BL(3),TE,BT,DE,DIC}
```

Table 3: **Initialization Parameter Description**

Parameter Name	Width	Description
CL	3	CAS Latency [0 0 x] RESERVED [0 1 0] 2 [0 1 1] 4 [1 x x] RESERVED
BL	3	Burst Length [0 0 0] RESERVED [0 0 1] 2 [0 1 0] 4 [0 1 1] RESERVED [1 x x] RESERVED
TE	1	Test Mode [0] REGULAR MODE (default) [1] TEST MODE
BT	1	Burst Type [0] SEQUENTIAL [1] INTERLEAVE
DE	1	DLL Enable [0] DLL ENABLE (default) [1] DLL DISABLE
DIC[1]	1	Output Drive Impedance Control [0] STANDARD [1] RESERVED

Notes:

1. The DIC option is not currently supported by FCRAM manufacturers, but has been included for future compatibility. Therefore, this bit must be tied Low. `[u_init_parms(0)=0]`

When the reset signal is released, the system first waits for the DCM to lock. Once this occurs, the controller latches in the `u_init_parms` vector and begins the reset/initialization process. Table 4 shows the FCRAM process specifications.

Table 4: **Powerup Initialization and Reset Conditions**

Command	Comments
DESL	12 or more cycles
MRS	MRS command with reset address
DESL	Maintain same address for four or more cycles
DESL	Change address
DESL	Maintain previous address for four or more cycles - End of RESET condition
EMRS	Set extended mode register
MRS	Set mode register
REF	Issue two or more auto-refresh commands
*llock	Wait for llock clock cycles after EMRS
*WRITE	Issue a write command to all four banks

The EMRS, MRS, and REF commands (after the reset condition completes) can occur in any order. Because llock is with respect to the EMRS command, this reference design issues the commands in the order listed to minimize the required initialization time.

The reference design issues all commands except those indicated with an asterisk. The llock clock cycles must occur before issuing the final four Write commands (one to each bank). Some possible ways to complete the startup sequence include issuing the required commands from the user interface at start-up, or modifying the HDL code. Once the commands are issued, the initialization sequence is complete and the FCRAM device is ready for normal operation. Any commands issued while the controller is in an initialization process will violate the FCRAM specification. Further details are provided in the **Initialization Sequence**.

Refresh

There are two ways that refreshes can be performed.

User Initiated Refresh

The user interface indicates that this mode is to be used by setting `u_ref_enable = 0`. In this mode, the user is required to issue the desired refresh command to the FCRAM controller. This is done by setting `u_cmd = 101` (for Self-Refresh) or `u_cmd = 111` (for Auto-Refresh). Once the controller has acknowledged this command (by asserting `u_ack`), the controller then handles the refresh by issuing the required commands to the FCRAM. In the case of a Self-Refresh request, the controller remains in the refresh state as long as the Self-Refresh command is given.

It is the responsibility of the user interface in this mode to ensure refresh commands occur often enough to meet the FCRAM specification.

Controller Initiated Refresh

The user interface indicates that this mode is to be used by setting `u_ref_enable = 1`. In this mode, the controller automatically issues an auto-refresh command to the FCRAM when the refresh interval timer expires.

These refresh commands are only acknowledged during incoming request boundaries. That is, a refresh command will not interrupt a command currently in progress or be inserted in the middle of a multiple burst access. Once the refresh interval timer expires and the current operation completes, the refresh will have highest priority.

The user passes parameters to the controller as follows:

```
u_ref_parms =  {ref_burst_cnt(4), ref_interval_cnt(16)}
```

where,

`ref_burst_cnt` specifies how many refreshes should occur in a row (burst refresh), and `ref_interval_cnt` specifies how often (in clock cycles) a refresh should occur.

Burst Refresh

According to the FCRAM specification, one must wait at least $T_{REFI\text{-}MIN}$ (auto-refresh interval) before another auto-refresh command is issued. It also states that the maximum time between auto-refreshes is $T_{REFI\text{-}MAX}$. But these specifications can be distributed by issuing multiple refreshes (up to eight) in a row. This is the concept of a burst refresh.

For example, if at time zero a single auto-refresh command is issued, the next auto-refresh cannot happen until $T_{REFI\text{-}MIN}$ cycles and must happen before $T_{REFI\text{-}MAX}$ cycles. If instead a burst of *n* auto-refreshes occur, these can be done immediately in a row (do not have to wait $T_{REFI\text{-}MIN}$ cycles), but one must now wait *n* x ($T_{REFI\text{-}MIN}$) cycles before the next auto-refresh command, and no more than *n* x ($T_{REFI\text{-}MAX}$).

Therefore, the more refreshes done in a "burst", the longer one can wait before another auto-refresh must be issued. However, this also will tie up the memory for a longer time while the auto-refreshes are being performed.

Calculating Refresh Interval

If one is not careful in choosing this value, it is possible to violate the FCRAM specification. One should first choose the number of refreshes to be performed in a row (`ref_burst_cnt`). Given the T_{REFI} minimum and maximum values from an FCRAM data sheet, as well as the clock frequency (T_{CK}), one can calculate the values as follows:

$$\texttt{ref\_interval\_cnt}_{MIN} = (t_{REFI\text{-}MIN}) \times (\texttt{ref\_burst\_cnt}) / T_{CK}$$
$$\texttt{ref\_interval\_cnt}_{MAX} = (t_{REFI\text{-}MAX}) \times (\texttt{ref\_burst\_cnt}) / T_{CK} - (\texttt{u\_num\_xfers} + I_{RC})$$

It could be possible that a read or a write transaction is in progress when the auto-refresh counter expires. Therefore, one should include the maximum number of transfers possible, as well as the IDLE time one must wait after a memory access before one can perform a refresh. These values are included in the $\texttt{ref\_interval\_cnt}_{MAX}$ calculation above.

Note that both the `ref_interval_cnt` and the `ref_burst_cnt` include an extra bit for future growth.

Memory Accesses

This section outlines the commands and signals required in order to successfully perform a Read or a Write request to the FCRAM controller.

In general, a memory access works as follows:

- User supplies the desired memory and bank location of the memory access
- User supplies the number of transfers for the memory access
- User issues the read or write command to the controller
- The FCRAM controller acknowledges the command (`u_ack = 1`). Once this acknowledgement occurs, the user may release the memory address, bank location, number of transfers, and the memory access command. At this point, the user may issue the next command to ensure the controller pipeline remains full.
- The user should supply data (during a write) or receive data (during a read).

Burst Transfers

This section explains the controller implementation of a burst memory access.

A single FCRAM memory access is limited to burst length (BL) number of data values, where a data value equals the width of the FCRAM data bus. If the user requires multiple memory accesses to consecutive memory locations, the FCRAM controller can automatically string these memory accesses together to form a burst memory access.

This is accomplished from the user interface through `u_num_xfers`. The value is the number of clock cycles that data will be transferred to or from the controller via the data buses `u_data_i` or `u_data_o`.

For example, setting `u_num_xfers = 1` for a write operation requests data on `u_data_i` for one clock cycle, or stated another way `u_data_req` is High for one clock cycle. Similarly, setting `u_num_xfers = 1` for a read provides data on `u_data_o` for one clock cycle, or stated another way `u_data_val` is High for one clock cycle. Note also that a transfer on the system bus represents two transfers on the FCRAM (DDR) bus. Therefore, for a full burst when `BL = 2`, set `u_num_xfers = 1`. Likewise, for a full burst when `BL = 4`, set `u_num_xfers = 2`.

Similarly, issuing 16 consecutive data transfers on the FCRAM bus can be implemented by a single command from the user interface by setting `u_num_xfers = 8`. Because `u_num_xfers` is a 4-bit number, the user interface has the option of performing up to 16 consecutive memory accesses, or 32 data transfers on the FCRAM bus.

Address Translation

The starting point for the memory access is supplied via `u_addr`. This bus maps to the bank, row, and column address on the FCRAM interface as follows:

```
u_addr[26:0]   = {ba, row, col}

u_addr[26:25]  = bank[1:0]
u_addr[24:10]  = row[14:0]
u_addr[9:0]    = col[9:0]
```

Notes:

1. A 10-bit column value allows the FCRAM controller to be expandable for future FCRAMs. However, one should also ensure that the column addresses beyond what the chosen FCRAM devices support are not accessed. For example, a x16 device uses seven column address bits, therefore `u_addr[9:7]` should be set to zero. Consult your FCRAM data sheet for other memory configurations.

Once the command has been accepted (`u_ack = 1`), the controller latches in the values supplied on `u_addr`. These values are decoded; and during the first command (WRA/RDA), the controller outputs to the FCRAM the upper (row) address and the bank address. During the second command, the controller outputs to the FCRAM the lower address (column).

During each successive read or write operation (combination of WRA/RDA and LAL) within a given request (i.e., `u_num_xfers` number of transfers has not yet completed), the controller automatically increments the bank address by one. Recall (from the Burst Length within the **Mode Register Set (MRS)** section) that once a row and column are selected, a read or write command will progress across "burst length" number of columns. Therefore, when the bank address overflows (i.e., transitions from three to zero), the current address (addr[24:0]) is incremented by the programmed burst length (BL). During a multiple burst access, this accesses the memory through the banks, across the columns, and finally down the rows.

Access Rules

According to the FCRAM specification, once a bank access occurs one must wait I_{RC} cycles (read/write cycle time) before accessing the same bank again. Therefore if a user attempted to issue multiple reads, writes, or a combination of read and write to the same bank within IRC cycles, a bank conflict violation occurs.

Additionally, when issuing a read command followed by a write command to different banks, one must wait I_{RWD} clock cycles (read-write turnaround time) before that command can be executed. Ignoring this specification causes a read-write turnaround violation.

Because the write-read turnaround (I_{WRD}) time is one clock cycle, write-read violations should not occur.

Rather than force the user interface to be aware of these potential problems, the FCRAM controller monitors for bank collisions and read-write turnaround violations. If a requested command would violate the FCRAM specification, the FCRAM controller will handle these conflicts (such as insert IDLE states until the parameter is met).

Read Request

In order to perform a read request, the user interface should set the following:

```
u_addr[26:0] = {ba, row, col}
u_num_xfers  = Number of 32 bit data values to be transferred
u_cmd        = 110
```

These values should be maintained until the command is acknowledged by the FCRAM controller (`u_ack = 1`). Once this acknowledgment occurs, the user interface may release these values and issue the next command. `u_data_val` will go High indicating that `u_data_o` contains valid read data.

Write Request

In order to perform a write request, the user interface should set the following:

```
u_addr[26:0] = {ba, row, col}
u_num_xfers  = Number of 32 bit data values to be transferred
u_data_i     = First 32-bit data value
u_cmd        = 100
```

These values should be maintained until the command is acknowledged by the FCRAM controller (`u_ack = 1`). Once this acknowledgement occurs, the user interface may release `u_addr`, `u_num_xfers`, and `u_cmd`. The first data piece on `u_data_i` should be maintained until the controller requests the data, which is done through `u_data_req`. The first rising clock edge after `u_data_req` is asserted High indicates acceptance of the current 32-bit data value on `u_data_i`, and the next 32-bit data value should be made available on the next clock.

Data Mask

A data mask (DM) allows the user to "mask off" pieces of data during a write command. There are two mechanisms for specifying the data mask depending on the part used (bond out option):

1. Via traditional separate external DM pins.
2. Via encoded mask passed through the address pins (specifically during the LAL command, on pins A14-A11.

The encoded mask method was supplied because it scales better with frequency. This implementation of the FCRAM controller is based off the second implementation – the embedded data mask.

The data mask function implemented in this controller is only applicable for BL = 4 and for an odd number of transactions. This works as follows:

The user interface specifies to the controller how many 32-bit data transfers are to be done. If the user specifies an odd number of 32-bit data transfers (e.g., `u_num_xfers = 3`), this corresponds to one and a half full burst transactions. Because of this half transaction, the FCRAM controller must mask out the last clock cycle of the write command.

This is done through the data mask feature. The controller automatically derives the appropriate data mask value from the `u_num_xfers` and passes this value to the FCRAM via the address pins during the LAL cycle. In this design, there is no way to manually specify a data mask through the user interface.

The data mask is provided for all memory writes during the lower address access. This means that all even transfers and all odd transfers until the last memory transfer will have the mask value set for "write all words." The final transfer in an odd memory transfer with BL = 4 will have the mask value set for "write first two words."

FCRAM Controller Details

Digital Clock Manager (DCM) Implementation

This section describes the *clk_dcm* block. The reference design clocking scheme uses the Virtex-II DCMs, global clock networks, and IOB DDR registers. Figure 6 shows the clocking structure.The first DCM, `DCM_CLK`, generates two clock outputs. One clock output (`clk`) will directly follow the users input clock (`u_clk`). The second clock output (`clk90`) will be a 90° phase shifted version of `u_clk`. The `clk` output also drives the IOB DDR flip-flops used to generate the FCRAM clock (`ddr_clk` and `ddr_clkb`).

The second DCM, `DCM_RCLK`, generates one clock output. This clock (`rclk`) is a phase shifted version of the users input clock (`u_clk`). It is used to recapture data during a memory read from the DQS clock domain. Once captured on the `rclk` clock domain, the reference design transfers the read data to the main system clock domain (`clk`). The phase shift value will be specific to each system, and therefore must be programmed accordingly. Further details on this clocking scheme is found in the **Read Data Path** and in the **Read Recapture Timing Analysis**.

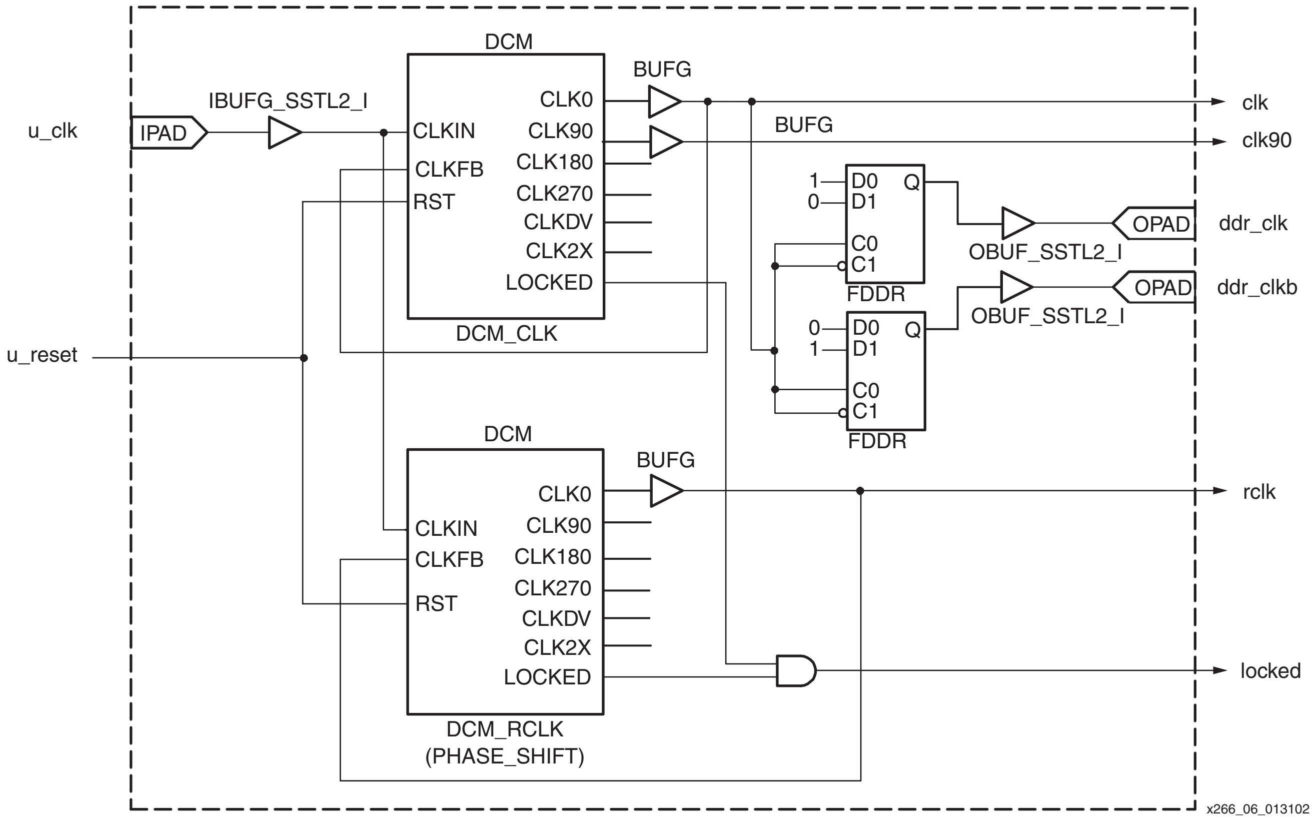

Figure 6: **DCM Implementation in the clk_dcm Block**

Data Path

The Virtex-II devices have enhanced IOBs for direct implementation of DDR functions. This application note leverages this enhanced technology, allowing for full DDR support to be completely contained within the IOBs. Additionally, it allows for all inputs and outputs to the DDR FCRAM interface to be registered within the IOB to minimum clock-to-out delays. Figure 7 shows a standard DDR implementation for a single IOB in the Virtex-II device.

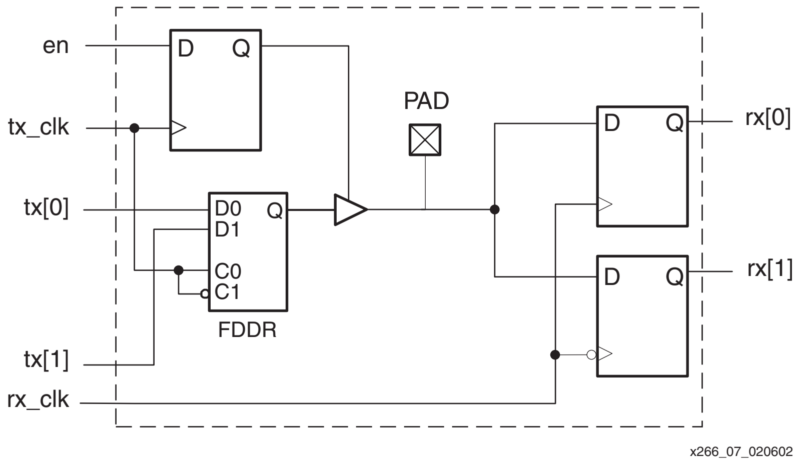

Figure 7: **DDR IOB Example Implementation**

Figure 8 shows a simplified schematic view of the data path and the data strobe generation logic. To present a general view of the data path, this figure has removed the HDL hierarchical boundaries. Additional details are available in the *data_path* and the *data_strobe* HDL files.

For all input signals, an SRL label is used to indicate multiple stage pipeline delays. These delays allow the data (`ddr_dq`) and data strobe (`ddr_dqs`) signals to align with the FCRAM control signals. Since the user data buses (`u_data_i` and `u_data_o`) are SDR and the FCRAM data bus is DDR, the user data buses are twice as wide as the FCRAM data bus. Also note that even though not indicated by Figure 8, the `ddr_dqs` 3-state and output flip-flops, and the `ddr_dq` 3-state, output and input flip-flop are implemented in the Virtex-II IOBs.

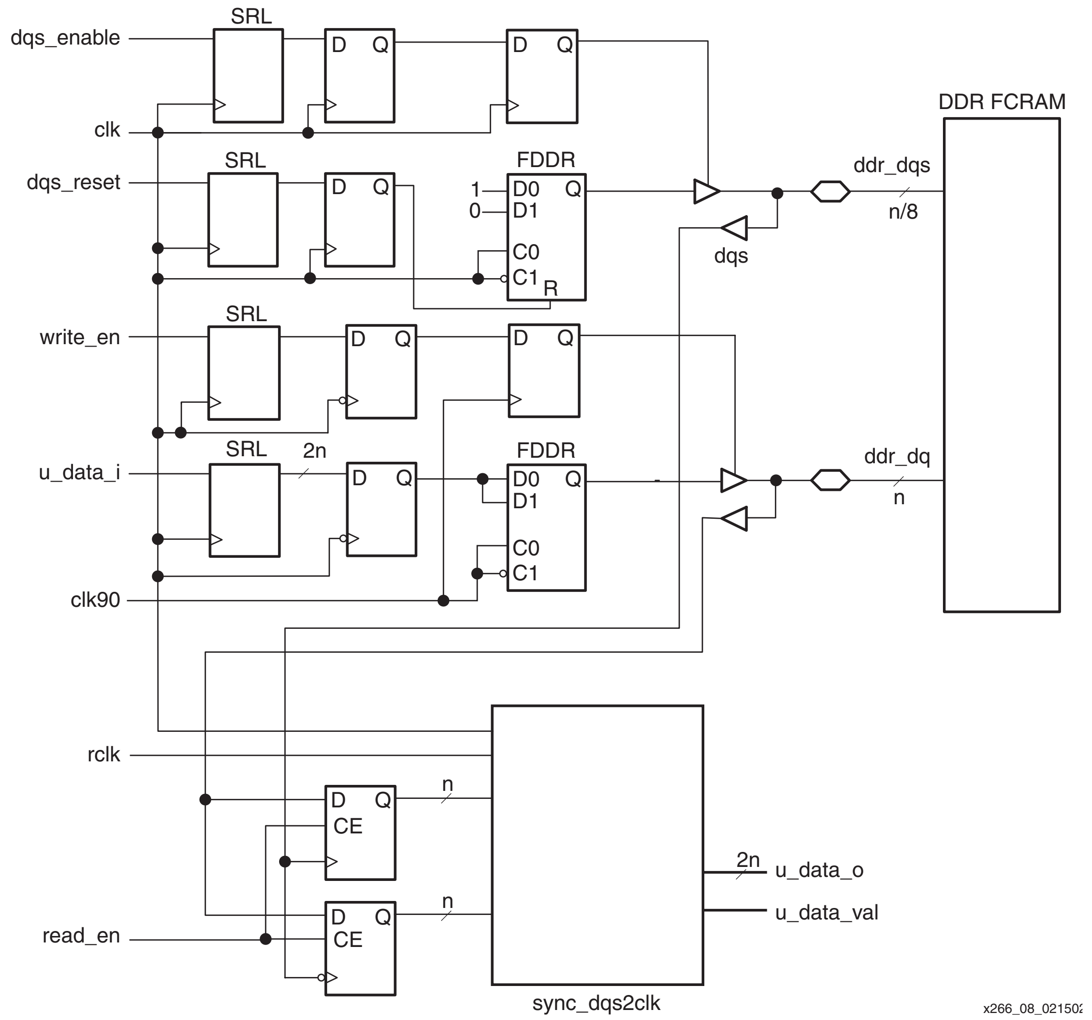

Figure 8: **Data Path**

Write Data Path

During memory writes, the controller must provide the strobe center aligned with the data at the pins of the FCRAM. Additionally, the FCRAM specification gives a relationship between CLK and DQS at the pins of the FCRAM. Generally, the DQS and CLK signals should be approximately phase aligned, although the specification does allow for some skew. In order to minimize this variance, both CLK and DQS are forwarded through DDR flip-flops clocked off `clk` and $\overline{\texttt{clk}}$.

Generated by the *controller* block, the `dqs_enable` signal controls the 3-state output while the `dqs_reset` signal holds the DQS flip-flop in reset. These signals allow the DQS timing parameters (such as the DQS preamble setup time) to be met. Once the `dqs_reset` signal is released, the DDR flip-flop inputs tied to a static one or zero generate the toggling nature of DQS.

Because DQS is generated from `clk`, the DQ signals are forwarded through DDR flip-flops clocked off of `clk90`. This naturally center aligns the data strobe with the data. The `write_en` signal is generated by the *controller* block and controls the 3-state output of the data path. The `u_data_i` is the user data input. Because both of these signals are synchronous to the `clk` domain, they are first transferred to $\overline{\texttt{clk}}$, and then to the `clk90` domain. This eases the timing requirements of the clock domain transfer.

Read Data Path

During memory reads, the FCRAM device provides the DQ and DQS signals to the FPGA. This reference design uses DQS as the clock to capture the read data DQ. DQS is distributed on dedicated local clocking resources, as described in **Pinout Constraints for Local Clock Distribution**. Because DQS is strobing in nature, data captured on the DQS domain must be immediately recaptured.

In order to recapture the data, a relationship between the DQS domain and the system clock domain must be found. The arrival of the data during a memory read will depend on system dependent factors such as board layout. Because of these variables, this reference design uses a DCM to generate a phase shifted version of the system clock (`rclk`). This allows a designer to align the recapture clock with the DQS clock domain, as outlined in **Read Recapture Timing Analysis**.

Data in the DQS domain is written by the `rclk` directly into a dual-port LUT RAM. The system clock reads the data out of the dual-port LUT RAM. Because the recapture clock is asynchronous to the internal system clock, all transfers between clock domains are double-registered to remove any setup, hold, or metastability issues. This recapture and synchronization logic is handled by the *sync_dqs2clk* module. As shown in Figure 8, this module receives the read data, the recaptures the clock, the system clock, and the enable signals (not shown). It generates the `u_data_val` and `u_data_o` signals for the user interface synchronous to the system clock domain.

Controller State Machine

A simplified view of the main controller state machine is shown in Figure 9. This state machine is coded as a one-hot state machine and contains replicated states to reduce the required decoding at each level. Because Figure 9 presents a general overview of the state machine, most duplicate states are omitted. Further information is found in the state machine portion of the *controller* HDL file.

Upon powerup, the controller is in an IDLE state. When reset is released and the DCM locks, the controller automatically begins the initialization process. Once this sequence is completed, the controller moves into the main IDLE state where it is able to accept Read, Write, and Refresh commands.

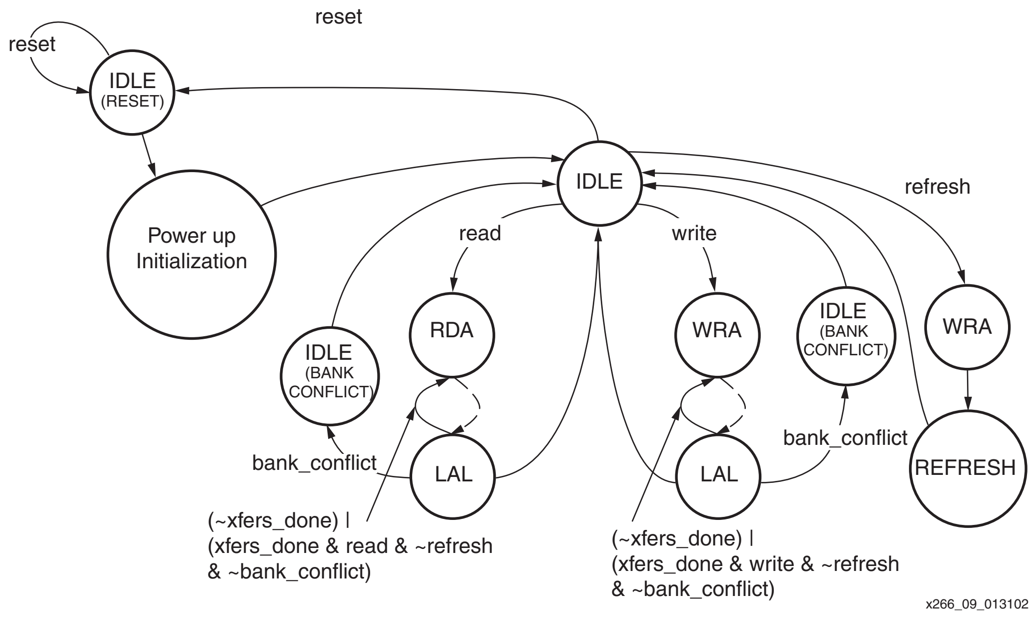

Figure 9: **State Machine Diagram**

Read Command

If u_cmd is set to a read command, the controller enters the RDA state followed by the LAL state. The controller continues to loop through these states until the number of transfers specified has completed. This condition occurs when u_num_xfers completes, which asserts xfers_done = 1.

Once xfers_done is asserted, it is possible for the controller to accept another command. If the user has issued another read command, the refresh counter has not expired (indicated by refresh = 0), and the specified address does not cause a bank conflict (indicated by bank_conflict = 1), then it is possible for the read command to be immediately executed. This is seen in the state machine by the controller moving back into the RDA state, which begins another read command.

If the issued read command causes a bank conflict, then the controller goes to the IDLE (BANK_CONFLICT) state. Likewise, if the issued command is a write command, then the controller must ensure that the read-write turnaround time of the FCRAM is not violated and therefore the controller moves to the IDLE (BANK_CONFLICT) state. This allows the controller to insert IDLE states until the requested bank can be accessed again, as required by the FCRAM specification. This ensures that no access violations can occur.

Finally, if xfers_done is asserted and the refresh counter has expired (indicated by refresh = 1), or if the issued command is not a read or a write command, then the controller will go to the IDLE state. If refresh is asserted, then the controller will automatically go to the WRA state and then into the REFRESH state, where an auto-refresh will be performed. Otherwise the controller will remain in the IDLE state until the next valid command is issued.

Write Command

If u_cmd is set to a write command, the controller enters the WRA state followed by the LAL state. The controller continues to loop through these states until u_num_xfers completes, which asserts xfers_done = 1.

Once xfers_done is asserted, it is possible for the controller to accept another command. If the user has issued another write command, the refresh counter has not expired (refresh = 0), and the specified address does not cause a bank conflict, then it is possible for the write command to be immediately executed. This is seen in the state machine by the controller moving into back into the WRA state, which begins another write command.

If the issued write command causes a bank conflict, then the controller goes to the IDLE (BANK_CONFLICT) state. This allows the controller to insert IDLE states until the requested bank can be accessed again, as required by the FCRAM specification. This ensures that no access violations can occur.

Finally, if xfers_done has been asserted, but the issued command is not a write command or if the refresh counter has expired (refresh = 1), the controller goes to the IDLE state. If refresh has been asserted, then the controller automatically goes to the WRA state and then into the REFRESH state, where an auto-refresh is performed. Otherwise, the controller remains in the IDLE state until the next valid command is issued.

Timing Diagrams

Initialization Sequence

Figure 10 shows the initialization sequence. Initially, the system should be held in reset (u_reset_n = 0) and the initialization data (u_init_parms and u_ref_parms) should be provided to the user interface. In the reference design the system reset is a combination of the user reset and the DCM locked signals. Therefore, when reset is released (u_reset_n = 1), the system waits for the DCM to lock (LOCK_DLL). Once the DCM locks, the controller state machine is released from reset and automatically begins the **Powerup Initialization and Reset Conditions**.

According to the FCRAM specification, the FCRAM DLL is enabled during the EMRS command. Therefore, the user must also wait for the FCRAM DLL to lock (which occurs ILOCK cycles after the EMRS command has been issued) before issuing any commands. Once the

FCRAM DLL locks, the user must issue the four write commands (one to each bank). As indicated in Figure 10, it takes "INIT TIME" clock cycles to issue the power-up initialization commands up to the EMRS command. In this reference design, INIT TIME depends on the programmed CL; for CL = 2, INIT TIME = 29 clock cycles, for CL = 3, INIT TIME = 32 clock cycles. Therefore, once the DCM is locked, the user interface can not issue the four Write commands until INIT TIME + ILOCK clock cycles. Once these commands are issued, the initialization sequence is complete and the system is ready for normal operation.

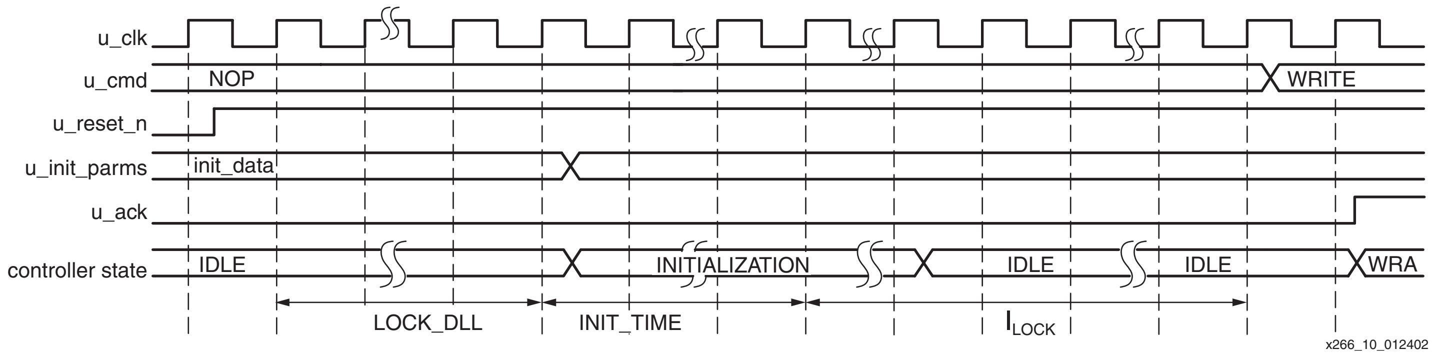

Figure 10: **Initialization Timing Diagram**

Write Cycle

Figure 11 is a timing diagram for consecutive write commands with BL = 4 and CL = 2. In this example, `u_num_xfers` is set to two for both memory transfers. This requires the user interface to supply data via `u_data_i` for four clock cycles, as indicated by `u_data_req`.

Following cycle T_2, a WRITE command is issued on `u_cmd`. Because the controller is in an IDLE state, it is able to immediately accept the command, and at cycle T_3 it moves into the WRA state and asserts the `u_ack` signal, indicating that the request was accepted.

Because the controller expects that the data required to satisfy the write request is available when the write request was made, `u_data_i` contains the first two data pieces for the write. At T_4 `u_data_req` is driven High by the controller. At the next rising clock edge (T_5) the controller accepts these two data pieces, and therefore at the following clock cycle the next data pieces are supplied. Because `u_num_xfers` is set to two, two 32-bit values are presented by the user interface to satisfy the transaction. Notice that this satisfies a complete burst for the FCRAM when the burst length has been programmed to four.

The second memory operation is issued as soon as the first command is acknowledged. Therefore, once `u_ack` is asserted at T4, the user interface issues the second write command as well as the desired address, bank, and number of transfers. Because `u_num_xfers` for the first write request was for two, the earliest this second command can be acknowledged is two clock cycles later. This occurs at T_5.

Since these are consecutive writes and no bank conflicts occurred, the bandwidth for the FCRAM is fully utilized.

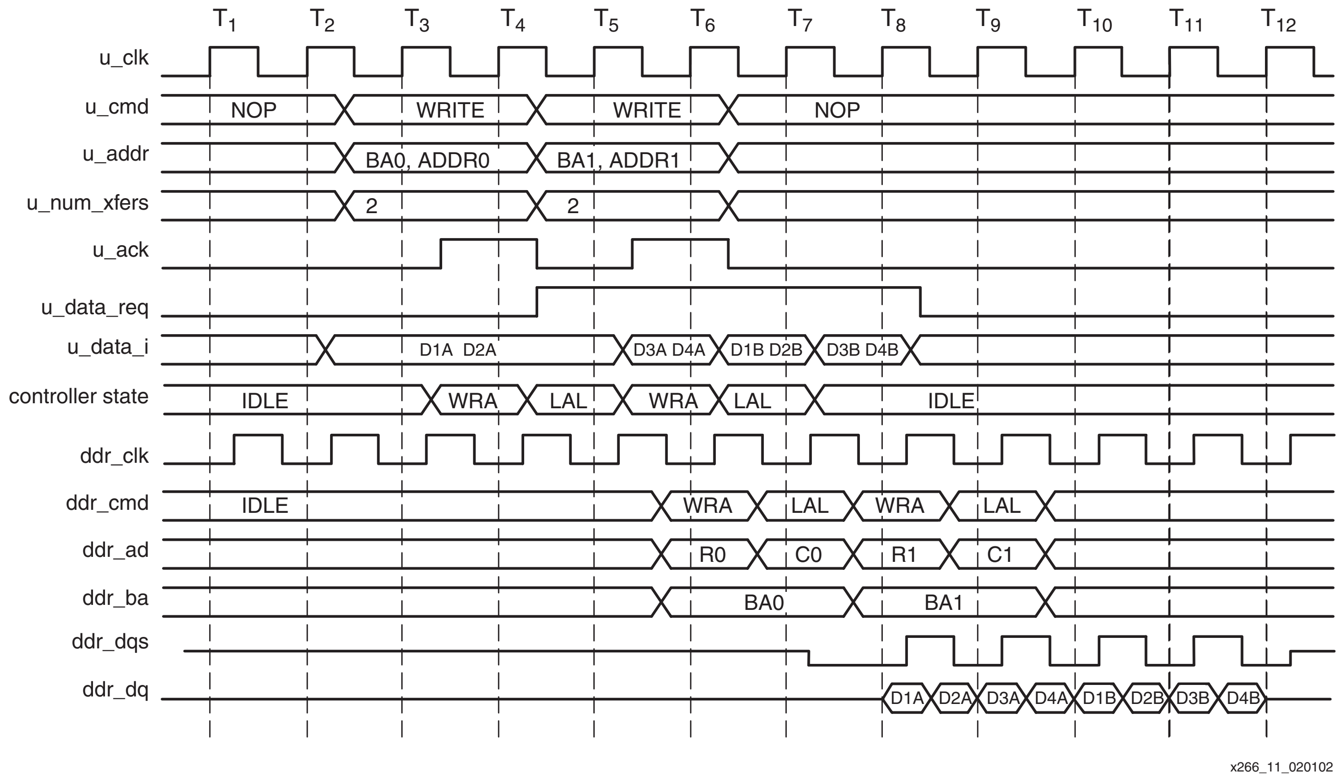

Figure 11: **Write Timing Diagram**

Figure 12 gives another write example, again with BL = 4 and CL = 2. In this example, `u_num_xfers` is set to five. This requires the user interface to supply data via `u_data_i` for five clock cycles, as indicated by `u_data_req`.

The starting address is listed as R0, C0, starting at bank two (BA2). Notice that the bank address is automatically incremented as the consecutive write commands are issued to the FCRAM. Also note that `u_num_xfers` set to five with a burst length of four corresponds to two full-burst writes and half of the third-burst write. Therefore, for the first two write commands the data mask during the WRA command will be set to "write all words." The FCRAM controller has the responsibility of recognizing the final "odd" transfer and setting the data mask to "write first two words" during the appropriate LAL command.This occurs at cycle T_{11}.

When the bank value overflows at T_{10}, the bank address wraps around, and the column address is automatically incremented by burst length. Because BL = 4 and the starting column address is C0, the first command writes across columns C0, C1, C2, and C3. Therefore, when the bank address overflows at T_{10}, the target address is automatically incremented to C4. This occurs at T_{11}.

Notice that for `u_num_xfers = 5`, the required data transfer completes at cycle T_{12}. However, according to the FCRAM specification, the DQS input must continue through the end of burst length, even if the data mask command has been issued. Therefore, DQS continues through cycle T_{13} as required.

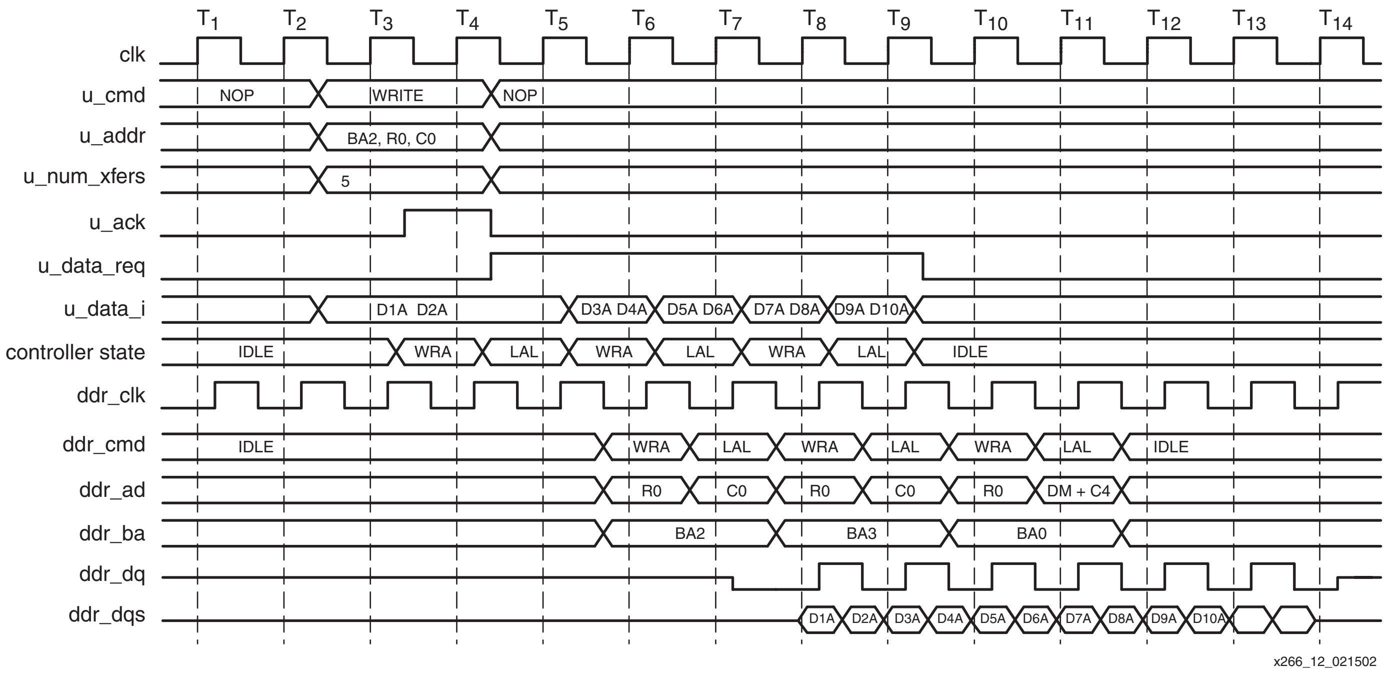

Figure 12: **Write Timing Diagram (2)**

Read Cycle

Figure 13 shows consecutive read commands with BL = 4 and CL = 2.

Following cycle T_2, a READ command is issued on `u_cmd`. Because the controller is in an IDLE state, it is able to immediately accept the command. At cycle T_3 the controller moves into the RDA state and asserts the `u_ack` signal, indicating that the request was accepted.

Since `u_num_xfers = 2`, the controller will return two 32-bit data values on `u_data_o`. The signal `u_data_val` indicates that current 32-bit value on `u_data_o` is valid data from a read request.

The second read request is issued as soon as `u_ack` is seen from the first read request. Because `u_num_xfers` is set to two for both read requests, the controller will provide data to the user interface for four clock cycles, as indicated by `u_data_val`.

Since these are consecutive reads and no bank conflicts occurred, the bandwidth for the FCRAM is fully utilized.

Figure 14 gives another read example, again with BL = 4 and CL = 2.

In this example, `u_num_xfers` is configured for three data transfers. Since the burst length is configured as four, this satisfies one full memory read and half of the second memory read. The FCRAM controller automatically issues these successive read commands and increments the bank address for the second command at T_8. According to the FCRAM specification, read commands do not use the data masks. Therefore, the read command returns data for the two full reads, and it will be up to the FCRAM controller to "mask" the final odd transfer. This is done through the use of `u_data_val`, which transitions Low at T_{16} to indicate the three `u_num_xfers` have completed.

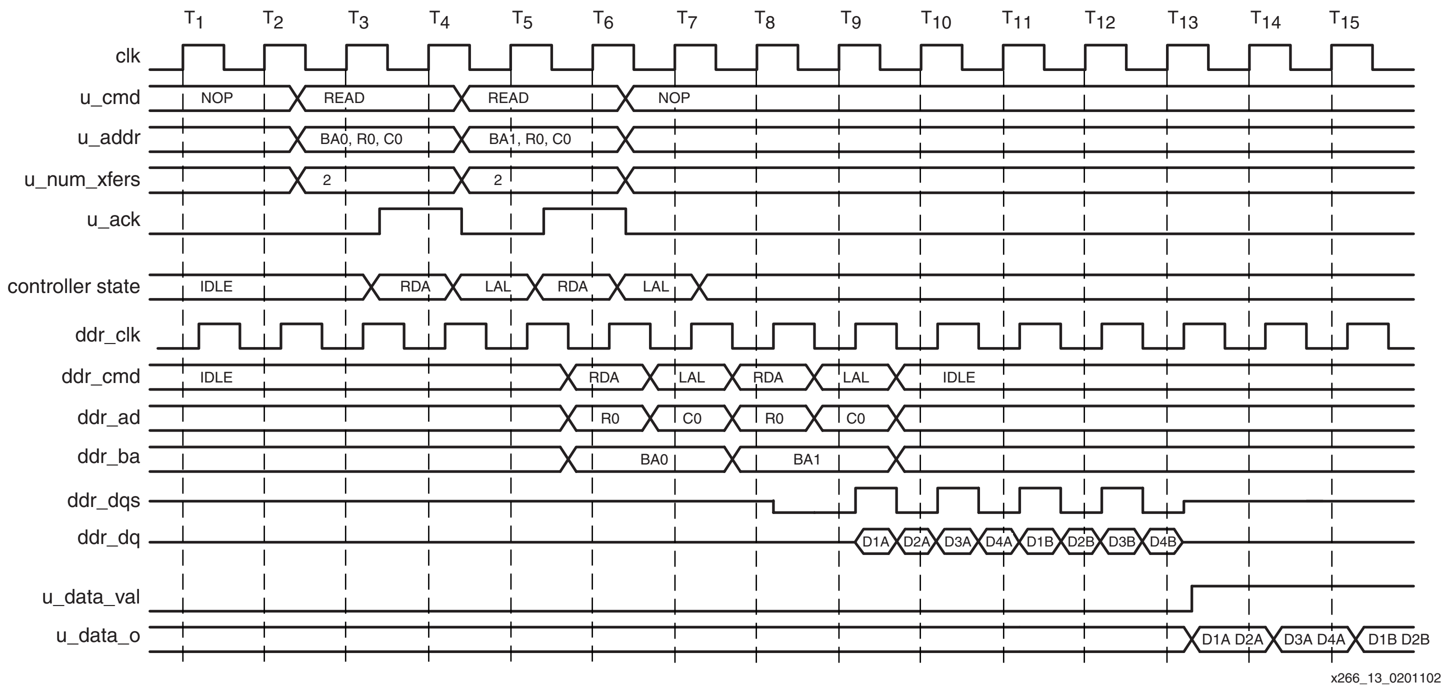

Figure 13: **Read Timing Diagram**

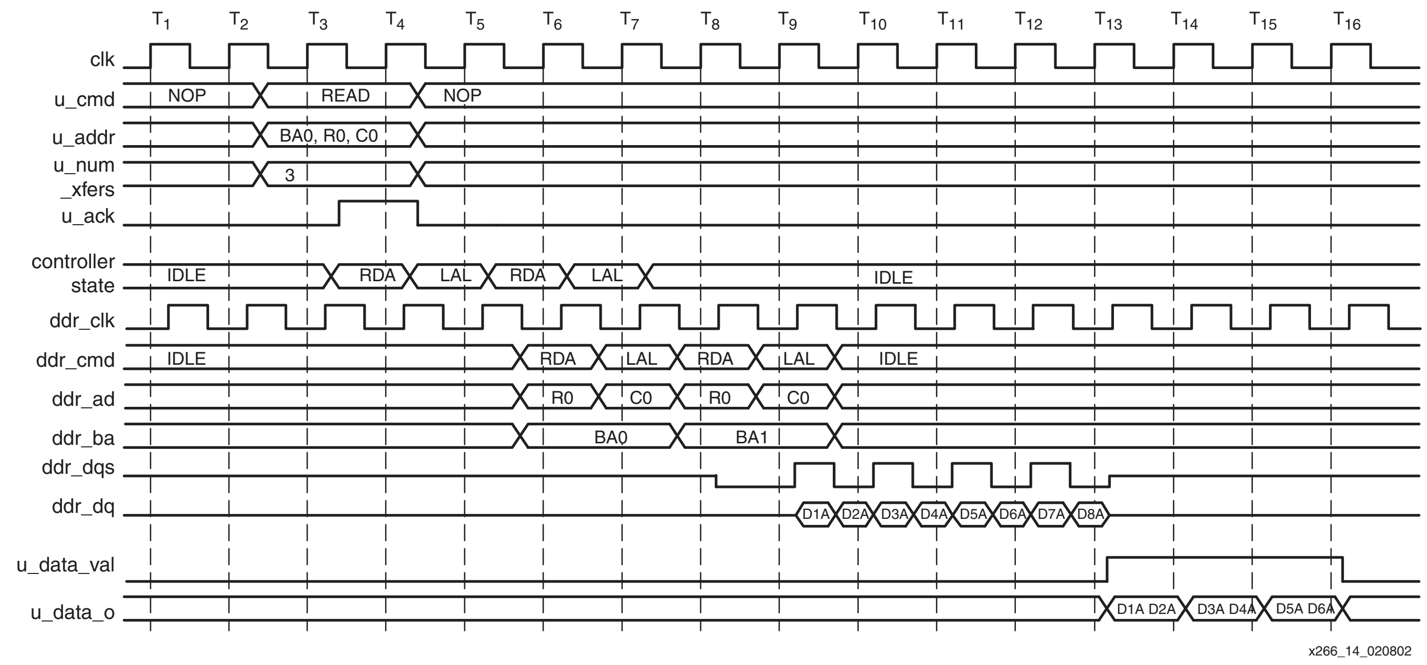

Figure 14: **Read Timing Diagram (2)**

I/O Timing Analysis

The maximum data rate of a fully synchronous system is limited as the clock-to-out of the transmitting device, the flight time of the signal, and the setup time of the receiving device approaches the bit rate time. In an SDR system, the bit rate is simply the reciprocal of the clock frequency (100 MHz SDR = 100 Mb/s = 10 ns bit rate). By using DDR, the bit rate decreases accordingly (100 MHz DDR = 200 Mb/s = 5 ns bit rate).

As clock frequencies continue to increase, this concept begins to limit system performance. The solution implemented by DRAM vendors to boost performance past these limitations is a source-synchronous clocking scheme using a bidirectional data strobe (DQS).

This section includes a sample timing analysis of the reference design. The analysis uses a -5 speed grade Virtex-II device and a -22 speed grade FCRAM device. The parameters used for this analysis are listed in Table 5 and Table 6. Values for these parameters should be taken from the most recent datasheets. For this sample analysis, Virtex-II values are taken from the Xilinx Virtex-II datasheet v1.6(1).

Table 5: **Parameters for a -22 Speed Grade FCRAM**

Parameter	Description	Min	Max	Units
t_{CK}	Clock cycle time	6.5	10	ns
t_{QSQV}	Data output valid time from DQS	0.4 x t_{CK} – 0.4	-	ns
t_{QSQ}	Data output skew from DQS	–0.52	0.52	ns
t_{DS}	Data input setup time from DQS	0.6	-	ns
t_{DH}	Data input hold time from DQS	0.6	-	ns
t_{DSPREH}	DQS input preamble hold time	0.25 x t_{CK}	-	ns
t_{CKQS}	DQS access time from clock	–0.85	0.85	ns
t_{DQSS}	DQS Low to High setup time	0.75 x t_{CK}	1.25 x t_{CK}	ns
t_{IS}	Input setup time (except for DQS and data)	1.0	-	ns
t_{IH}	Input hold time (except for DQS and data)	1.0	-	ns

Table 6: **Parameters for a Virtex-II Device**

Parameter	Description
T_{IOPI}	Input pad delay (SSTL2)
T_{IOPICK}	Input setup time, no delay (SSTL2)
T_{IOICKP}	Input hold time, no delay (SSTL2)
$T_{ICKOFDCM}$	Global clock and off with DCM
T_{OSSTL2_I}	Output switching adjustment (SSTL2-I)
T_{OSSTL2_II}	Output switching adjustment (SSTL2-II)

Read Timing Analysis

During a memory read, the FCRAM device will generate the DQ and DQS signals to be received by the FPGA. Figure 15 shows the timing relationship of these signals taken from the FCRAM specification.

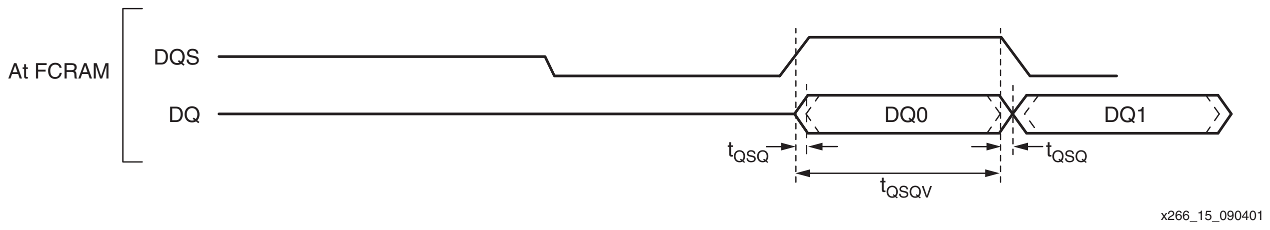

Figure 15: **AC Timing of Read Mode for DQS and DQ**

The FCRAM specification (Figure 15) guarantees "t_{QSQV} = Data Output Valid Time from DQS". The worst-case data output window from the FCRAM is found when "t_{QSQ} = Data Output Skew from DQS" is removed from t_{QSQV}. One must verify that from this window the setup and hold timing for a read cycle can be met at the Virtex-II IOBs.

Figure 15 shows that DQS and DQ will be approximately edge-aligned during a memory read. Therefore, it is the responsibility of the controller to delay the DQS so that the setup and hold time for the Virtex-II IOBs are met across the eight DQ inputs.

Because of the fixed resources in an FPGA, it is difficult to find a route to delay DQS to meet this delay requirement. However, it is possible to determine the route delay for the DQS line inside the FPGA and add additional delay on the board to position the DQS within the DQ data valid window.

Figure 16 shows the timing relationships of DQ to DQS at the pins of the FCRAM, at the pins of the FPGA, and at the IOB flip-flops. In Figure 16, the trace delay value for the DQ lines is referenced as t_{DQ}, and the trace delay value for the DQS line is t_{DQS}.

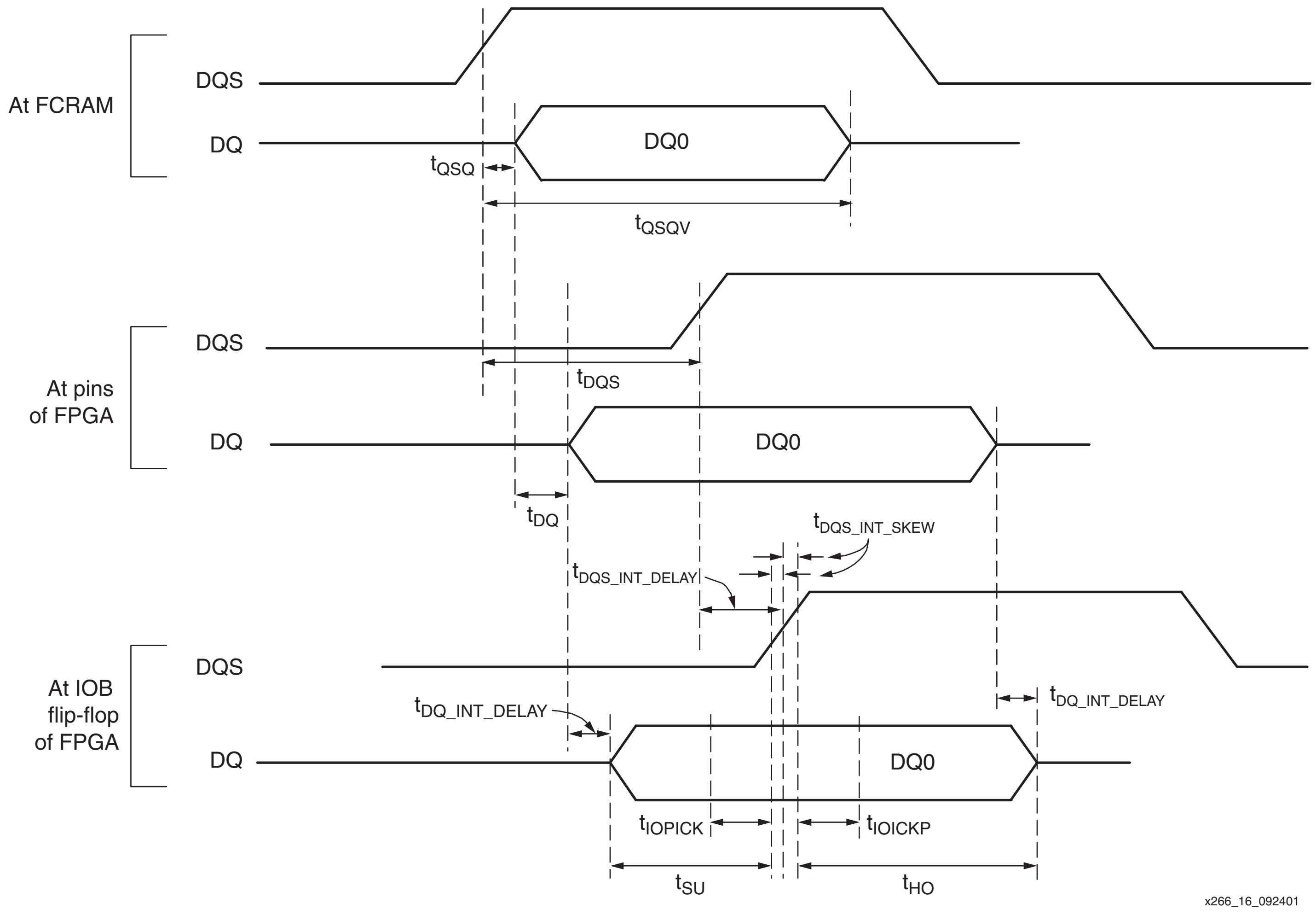

Figure 16: **Set Up and Hold Timing Diagram for Read Cycle**

Once DQS arrives at the pins of the FPGA, it enters an IOB and is routed to the eight data (DQ) IOBs. Therefore, the DQS internal FPGA delay is comprised of the delay through an SSTL2 pad plus the routing delay to the DQ loads. This is represented in Figure 16 as $t_{DQS\_INT\_DELAY}$, with the skew of the DQS signal across the eight DQ loads represented as $t_{DQS\_INT\_SKEW}$. Because the data (DQ) signals are latched in the IOB, there is no route delay.

Because this design uses a local clocking scheme to distribute DQS, certain pinout restrictions must be followed. These are listed in **Pinout Constraints for Local Clock Distribution**. If these criteria are met, the designer should see route delays similar to the sample delays listed in Table 7.

Table 7: **Sample Internal Data Path (DQ) and Data Strobe (DQS) Routing Parameters**

Parameter	Description	Min	Max	Units
$t_{DQ\_INT\_DELAY}$	Input package delay	0.000	0.000	ns
$t_{DQS\_ROUTE\_DELAY}$ (Note 1)	Routing delay from pad input to DQ loads		0.320	ns
$t_{DQS\_INT\_DELAY}$	Input clock delay (t_{IOPI_SSTL2}) + $t_{DQS\_ROUTE\_DELAY}$		1.50	ns
$t_{DQS\_INT\_SKEW}$ (Note 1)	Routing variation	-0.050	0.050	ns

Notes:

1. These values are based on implementation results and, therefore, actual trace numbers should be substituted. Manually lock IOBs to ensure proper placement.

Worst-case setup occurs with the maximum data (DQ) delay and minimum clock (DQS) delay. Likewise, worst-case hold occurs with the maximum clock (DQS) delay and minimum data (DQ) delay. As seen in Figure 16, the difference between t_{SU} (setup) and t_{IOPICK} is the slack on setup. Likewise, the difference between t_{HO} (hold) and t_{IOICKP} is the slack on hold.

$$t_{SU} = (t_{DQS} + t_{DQSINTDELAY} + t_{DQSINTSKEWMIN}) - (t_{QSQ} + t_{DQ} + t_{DQINTDELAY}) > t_{IOPICK}$$

$$t_{HO} = (t_{QSQV} - t_{QSQ}) - (t_{DQS} - t_{QSQ} - t_{DQ}) + t_{DQINTDELAY} - (t_{DQSINTDELAY} + t_{DQSINTSKEWMAX}) > t_{IOICKP}$$

In these equations, the desired result is the trace differential between DQS and DQ ($t_{DQS} - t_{DQ}$). This will give the amount of delay required to center DQS within the data valid window of DQ. Therefore, solving these inequalities for ($t_{DQS} - t_{DQ}$) gives:

$$\begin{aligned} t_{IOPICK} - (t_{DQSINTDELAY} + t_{DQSINTSKEWMIN}) + (t_{QSQ} + t_{DQINTDELAY}) &< t_{DQS} - t_{DQ} \\ \ldots &< -t_{IOICKP} + t_{QSQV} + t_{DQINTDELAY} - (t_{DQSINTDELAY} + t_{DQSINTSKEWMAX}) \end{aligned} \quad \text{(EQ 1)}$$

Using sample values from the Virtex-II data sheet, version 1.6 (Reference 1) gives the resultant:

$$1.38 - (1.50 - 0.05) + (0.52 + 0.0) < t_{DQS} - t_{DQ} < -(-0.81) + ((0.4 \times 6.5) - 0.4) + 0.0 - (1.50 + 0.050)$$

$$0.450\text{ns} < t_{DQS} - t_{DQ} < 1.460\text{ns}$$

This result shows that there is a little more than 1 ns of margin. This analysis is done assuming maximum timing for the FPGA. Due to the internal route delay of DQS, minimum timing analysis should also be done to ensure proper operation at best and worst-cases.

Because minimum timing numbers are not yet available, this analysis assumes a prorating factor (PF) as shown in Table 8.

Table 8: **Virtex-II Minimum Timing Prorating**

Parameter	Description	Value
PF	Prorating factor (percentage of maximum value)	0.25

Minimum timing occurs at the best process corner, operating at the highest voltage and lowest temperature. Because these effects are across the entire die (FPGA), a worst-case timing in one IOB would never exist with a best-case timing in an adjacent IOB. Therefore, this analysis prorates all Virtex-II parameters equally.

From EQ 1 the following minimum timing equation is determined:

$$(t_{IOPICK} \times PF) - ((t_{DQSINTDELAY} + t_{DQSINTSKEWMIN}) \times PF) + (t_{QSQ} + (t_{DQINTDELAY} \times PF)) < t_{DQS} - t_{DQ}$$
$$\ldots < (-t_{IOICKP} \times PF) + t_{QSQV} + (t_{DQINTDELAY} \times PF) - PF(t_{DQSINTDELAY} + t_{DQSINTSKEWMAX}) \qquad \text{(EQ 2)}$$

$$(1.38 \times 0.25) - (1.50 \dot{-} 0.05)0.25 + (0.52 + (0.00 \times 0.25)) < t_{DQS} - t_{DQ}$$
$$\ldots < -((-0.81) \times 0.25) + ((0.4 \times 6.5) - 0.4) + (0.00 \times 0.25) - 0.25(1.50 + 0.050)$$
$$0.503\text{ns} < t_{DQS} - t_{DQ} < 2.015\text{ns}$$

Combining these gives the following range for ($t_{DQS} - t_{DQ}$): $0.503\text{ns} < t_{DQS} - t_{DQ} < 1.460\text{ns}$

Write Timing Analysis

The critical timing for the write cycle is the setup and hold of DQ around DQS. During a memory write, the FCRAM controller generates DQS center aligned with DQ. This creates a one-quarter cycle setup and hold time on the output of the FPGA. However, as discussed in **Read Timing Analysis**, there is a requirement to offset DQS from DQ through additional trace delay. Figure 17 gives a sample timing diagram for a write command with CL = 2. As shown, these trace delays generate additional setup time and subtract from the hold time.

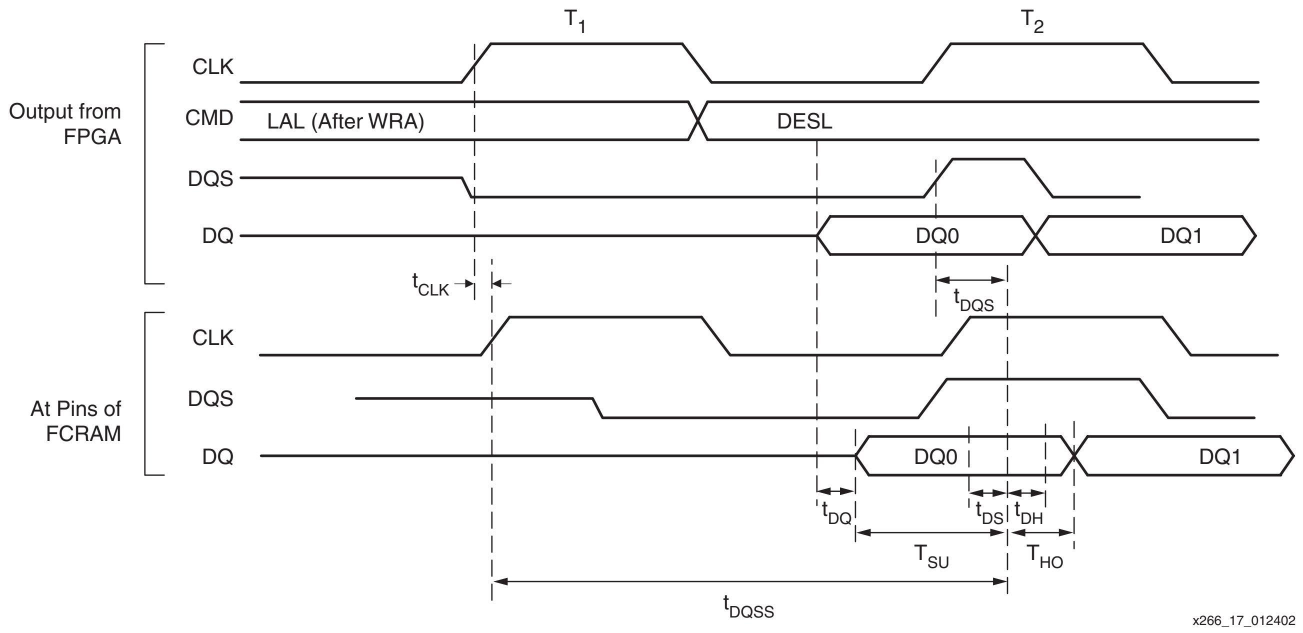

Figure 17: **Write Timing Diagram**

The trace delay of DQ is listed as t_{DQ}, the trace delay of DQS is listed as t_{DQS}, and the trace delay of the CLK is listed as t_{CLK}. The equations for these relationships are shown below, with the FCRAM setup (t_{DS}) and hold (t_{DH}) values listed in Table 3. The difference between T_{SU} (setup) and T_{DS} is the slack on setup. Similarly, the difference between T_{HO} (hold) and T_{DH} is the slack on hold.

$$t_{SU} = \frac{t_{CK}}{4} + t_{DQS} - t_{DQ} > t_{DS}$$

$$t_{HO} = \frac{t_{CK}}{4} + t_{DQ} - t_{DQS} > t_{DH}$$

Solving these equations for ($t_{DQS} - t_{DQ}$) gives:

$$t_{DS} - \frac{t_{CK}}{4} < t_{DQS} - t_{DQ} < \frac{t_{CK}}{4} - t_{DH} \quad \text{(EQ 3)}$$

$$-1.025\text{ns} < t_{DQS} - t_{DQ} < 1.025\text{ns}$$

Another timing requirement that must be met during the Write cycle is the relationship between DQS and the CLK. This requirement is specified as the DQS Low to High setup time (t_{DQSS}), and is the time from the rising edge of the CLK to the rising edge of DQS, as indicated in Figure 17. The FCRAM specification gives both a minimum and maximum value for this parameter (Table 4). Because DQS and CLK are both generated through DDR flip-flops clocked off the same clock, there is originally one clock cycle (t_{CK}) worth of setup time. However, the trace lengths of both CLK and DQS will adjust this value, as well as the output standard adjustment (not shown in Figure 17, but CLK is SSTL2_I, DQS is SSTL2_II per the FCRAM recommendation). Equation 4 highlights this relationship:

$$t_{DQSS(MIN)} < t_{CK} + t_{DQS} + T_{OSSTL2(II)} - t_{CLK} - T_{OSSTL2(I)} < t_{DQSS(MAX)} \quad \text{(EQ 4)}$$

Solving using the values from Table 4:

$$-1.015\text{ns} < t_{DQS} - t_{CLK} < 2.235\text{ns}$$

Read Recapture Timing Analysis

During a memory Read, data is captured by the IOB flip-flops with the DQS signal. Because DQS is a strobing signal, there is no guarantee of a successive clock edge moving the data from the IOB into the second stage of the data path. Therefore, the data must be recaptured from DQS to another clock domain.

This reference design uses a phase-shifted version of the user clock to do this data recapture. In order to use this method, the designer must calculate the required phase shift. This is outlined in Figure 18. The FCRAM clock (`ddr_clk`) is generated by forwarding the internal user clock through the IOB DDR flip-flops. This clock will travel from the FPGA to the FCRAM (t_{CLK}).

As stated in the FCRAM specification, upon receiving the clock, the memory will output the DQS signal within $\pm t_{CKQS}$. The DQS signal travels from the FCRAM to the FPGA (t_{DQS}), where it is routed to the IOB flip-flops ($t_{DQS\_INT\_DELAY}$).

The phase shift feature of the DCM is used to align the recapture clock with the DQS signal internal to the FPGA. Because data is transferred from the DQS domain to the recapture clock domain, the recapture clock should be positioned at the earliest possible arrival of DQS. This ensures the greatest time for the clock domain transfer. Equation 5 is for the recapture clock phase shift:

$$\text{Target Phase Shift} = (T_{ICKOFDCM} \times PF) + t_{CLK(MIN)} + t_{DQS(MIN)} + t_{DQSINTDELAY(MIN)} \quad \text{(EQ 5)}$$

The timing relationship between the DQS and the recapture clock must be constrained. Under best case conditions, there will be one clock period to transfer from the DQS to the recapture clock domain. Under worst case conditions, the transfer from the DQS to the recapture clock domain will have one period minus the difference between the maximum and minimum path timing. Using the prorating value given in Table 8 gives the following equation:

$$\text{DQS to rclk} = t_{CK} - \text{Phase Shift (Max)} - \text{Phase Shift (Min)} \quad \text{(EQ 6)}$$

Where:

$$\text{Phase Shift (Max)} = T_{ICKOFDCM} + t_{CLK(MAX)} + t_{CKQS(MAX)} + t_{DQS(MAX)} + t_{DQSINTDELAY(MAX)}$$

$$\text{Phase Shift (Min)} = \text{Target Phase Shift}$$

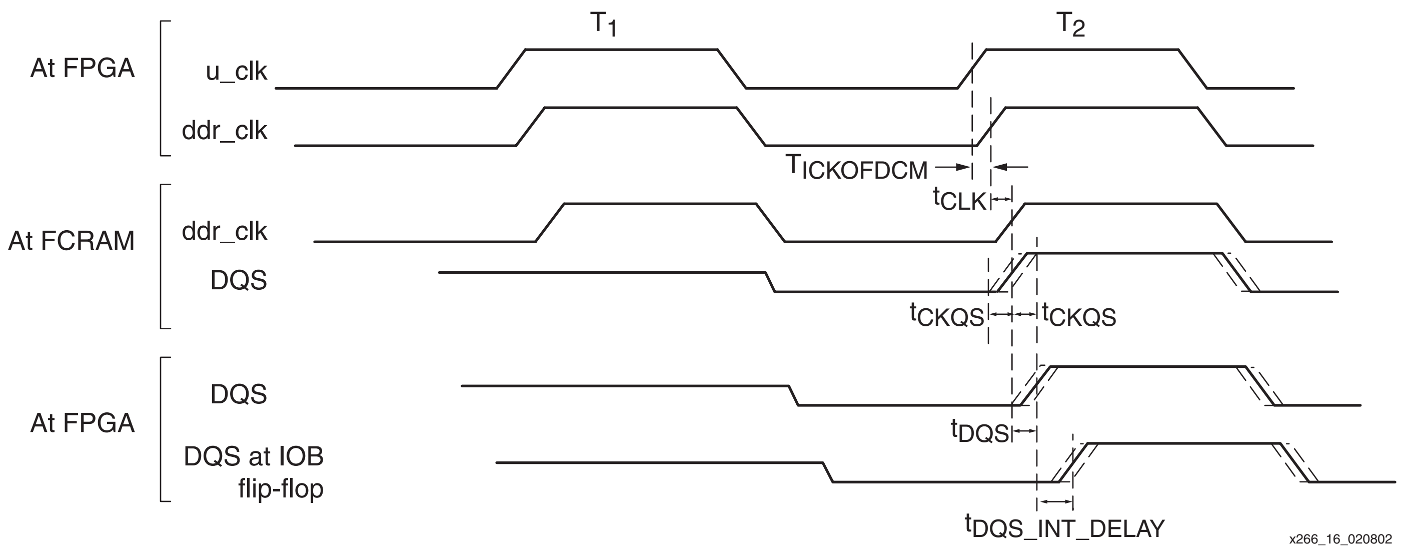

Figure 18: **Clock to DQS Output Delay Time**

Control Timing Analysis

All address and control signals are generated by the controller on the falling edge of `clk`. This naturally creates 1/2 cycle setup and hold timing. These values will be offset by the trace delay of the clock (t_{CLK}) and by the trace delays of the command / address signals (t_{CMD}). The command and address setup (t_{IS}) and hold (t_{IH}) are listed in Table 3.

$$T_{SU} = \frac{t_{CK}}{2} + t_{CLK} - t_{CMD} > t_{IS}$$

$$T_{HO} = \frac{t_{CK}}{2} + t_{CMD} - t_{CLK} > t_{IH}$$

Solving these equations for ($t_{CLK} - t_{CMD}$) gives equation 7.

$$t_{IS} - \frac{t_{CK}}{2} < t_{CLK} - t_{CMD} < \frac{t_{CK}}{2} - t_{IH} \quad \text{(EQ 7)}$$

$$-2.25\ \text{ns} < t_{CLK} - t_{CMD} < 2.25\ \text{ns}$$

Timing Analysis Summary

This section provided a sample analysis of the critical I/O timing in the reference design. The equations are a starting point to form relationships between the clock, data, data strobe, and address/control trace lines between the FPGA and the FCRAM. These relationships guarantee the required I/O timing for both the FPGA and the FCRAM. This analysis should be customized as required to fit a user specific design.

Equations 1 through 3 constrain the relationship between the data and the data strobe traces.

$$\begin{aligned} t_{IOPICK} - (t_{DQSINTDELAY} + t_{DQSINTSKEWMIN}) + (t_{QSQ} + t_{DQINTDELAY}) < t_{DQS} - t_{DQ} \\ \ldots < -t_{IOICKP} + t_{QSQV} + t_{DQINTDELAY} - (t_{DQSINTDELAY} + t_{DQSINTSKEWMAX}) \end{aligned} \quad \text{(EQ 1)}$$

$$\begin{aligned} (t_{IOPICK} \times PF) - ((t_{DQSINTDELAY} + t_{DQSINTSKEWMIN}) \times PF) + (t_{QSQ} + (t_{DQINTDELAY} \times PF)) < t_{DQS} - t_{DQ} \\ \ldots < (-t_{IOICKP} \times PF) + t_{QSQV} + (t_{DQINTDELAY} \times PF) - PF(t_{DQSINTDELAY} + t_{DQSINTSKEWMAX}) \end{aligned} \quad \text{(EQ 2)}$$

$$t_{DS} - \frac{t_{CK}}{4} < t_{DQS} - t_{DQ} < \frac{t_{CK}}{4} - t_{DH} \quad \text{(EQ 3)}$$

Equation 4 specifies the preamble timing for DQS. This generates a relationship between DQS and the clock traces.

$$t_{DQSS(MIN)} < t_{CK} + t_{DQS} + T_{OSSTL2(II)} - t_{CLK} - T_{OSSTL2(I)} < t_{DQSS(MAX)} \quad \text{(EQ 4)}$$

Equation 7 constrains the relationship between the address / control signals and the clock traces between the FPGA and the FCRAM.

$$t_{IS} - \frac{t_{CK}}{2} < t_{CLK} - t_{CMD} < \frac{t_{CK}}{2} - t_{IH} \quad \text{(EQ 7)}$$

The values used in the sample timing analysis give the following relationships:

$$0.503 \text{ ns} < t_{DQS} - t_{DQ} < 1.025 \text{ ns}$$

$$-1.015 \text{ ns} < t_{DQS} - t_{CLK} < 2.235 \text{ ns}$$

$$-2.25 \text{ ns} < t_{CLK} - t_{CMD} < 2.25 \text{ ns}$$

Equation 5 shows a sample calculation to tune the positioning of the recapture clock to a specific system. Equation 6 allows a designer to ensure the path from the data strobe to the recapture clock is properly constrained.

$$\text{Target Phase Shift} = (T_{ICKOFDCM} \times PF) + t_{CLK(MIN)} + t_{DQS(MIN)} + t_{DQSINTDELAY(MIN)} \quad \text{(EQ 5)}$$

$$\text{DQS to rclk} = t_{CK} - \text{Phase Shift (Max)} - \text{Phase Shift (Min)} \quad \text{(EQ 6)}$$

Pinout Constraints for Local Clock Distribution

Before choosing a pinout, the proper resources need to be available for routing the DQS lines from the pad input to the clock pin of the data loads. One way to ensure this is to use the global clock trees to distribute the DQS lines. However, since each DQS line only drives eight data loads, this is a fairly inefficient use of valuable clocking resources.

Virtex-II devices contain local clock distribution networks along the left and right edges of the device. These networks allow for a signal to enter an IOB and connect to a high-speed, low-skew, local routing resource that connects directly from an IOB to a fixed number IOB clock pins. This section gives an overview of these resources and describes how to successfully use them to distribute the DQS clock lines.

As described in the Virtex-II data sheet (Reference 1), each Input/Output Tile contains four IOBs that share a switch-matrix. IOB PAD4 is the top IOB, IOB PAD1 is the bottom IOB. In order for the DQS signal to access a local clock line, the DQS pad must be placed in IOB PAD4 (the top IOB). If IOB PAD4 is not available in the target package for a given tile, then the Input/Output Tile may not be used to for the DQS signal.

Placing the DQS pad in IOB PAD4 gives direct access to a local clock line. This local clock line is a HEX line spanning five rows above the chosen DQS Input/Output Tile (and may also drive back into the chosen DQS Input/Output Tile), and spanning six rows below the chosen DQS Input/Output Tile. The data (DQ) pads must be placed within these 12 rows.

A sample image from FPGA editor (Figure 19) shows a DQS pad (sixth tile or row down from the top) driving the DQ pads located five rows above, as well as six rows below, the DQS pad location. A sample pinout is included with the design files.

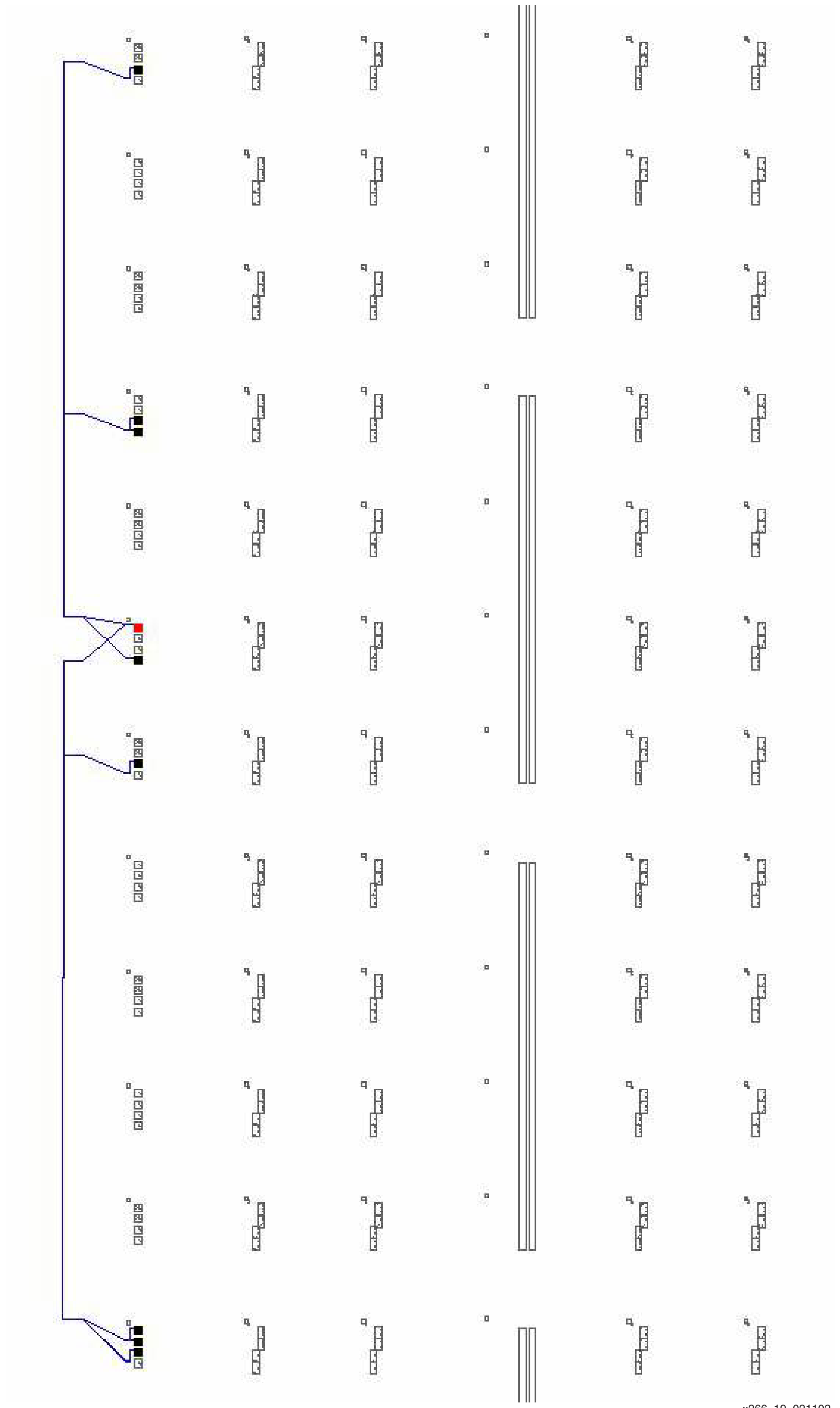

Figure 19: **Sample DQS Routing**

Design Implementation

The reference design includes Verilog source code, constraints file, and a sample implementation script. The design uses three global clock buffers and two DCMs. For a 16-bit data bus, the design requires approximately 600 slices. The reference design is available on the Xilinx FTP site: **ftp://ftp.xilinx.com/pub/applications/xapp/xapp266.zip**

References

Refer to the following documents for additional details.

1. Xilinx Inc., Virtex-II 1.5V Field Programmable Gate Arrays, Data sheet, 2002 **www.xilinx.com**.
2. Xilinx Inc., **XAPP200**, Synthesizable 1.6 GBytes/s DDR SDRAM Controller, Application Note, 2000
3. **Toshiba Inc.**, DDR FCRAM, Data sheet, 2001
4. **Fujitsu Inc.**, DDR FCRAM, Data sheet, 2001

Conclusion

FCRAM is a high performance, low power memory well suited for applications that require large memory densities and high effective bandwidths. This application note presents a general overview of this memory technology, and gives an example of how the Xilinx Virtex-II family FPGA architecture may be leveraged to implement a FCRAM controller.

FCRAM devices use a source-synchronous interface where a bi-directional data strobe is forwarded along with the data and used as the clock to capture data. Timing analysis in this type of system introduces challenges above a traditional fully synchronous system. Therefore, this application note provides a sample timing analysis, including timing diagrams and equations, to help designers verify timing budgets.

Before completing timing closure, board-level timing must be analyzed. Xilinx strongly recommends the use of a board-level design tool for this analysis, including IBIS simulations to ensure proper signal integrity. This analysis should include simulations of trace stackup, trace lengths, pin capacitance due to FCRAM device loading, and verification of proper termination on all signals. Additionally one must ensure adherence to the Simultaneously Switching Outputs (SSO) guidelines as listed in the *Virtex-II User Guide*.

Although the reference design targets a single x16 FCRAM device, the Verilog code may be easily modified to target different memory configurations, as described in **Appendix A**.

Appendix A

The following changes need to be made to the Verilog HDL source code to change the DDR FCRAM memory bus widths.

- `define.v`

Change the memory bus width to the desired value:

```
`define DDR_DATA_WIDTH <desired width>
```

Some components are instantiated in the HDL code, and therefore will need to be modified by the user to support various memory bus widths.

- `data_path.v`

The *data_path* module instantiates the DDR input and output flip-flops for the DQ bus. Instantiations for eight DQ bits (one byte) are contained in the *v2_ddr_iob* module, which can be found in the `data_path.v` HDL file. The number of instantiations must be changed to match the required external memory bus width. For example, if the interface is a single x16 memory, there would be two instantiations of the *v2_ddr_iob* module. If the interface is multiple FCRAM devices to form a x72 bit bus, this would require nine instantiations of this module. Within the HDL file, search for the "DDR IOB Instantiations" section, and modify the number of instantiations to match the target bus width.

- `data_strobe.v`

The data_strobe module instantiates the DDR output flip-flop for the DQS signal. Instantiations for a single DQS bit is contained in the *v2_dqs_iob* module, which can be found in the `data_strobe.v` HDL file. The number of instantiations must be changed to match the required external memory configuration. For example, if the interface is a single x16 memory that contains a data strobe per byte, this would require two instantiations of the *v2_dqs_iob* module. If the interface is multiple FCRAM devices to form a x72 bit bus, this would require nine instantiations of this module. Within the HDL file, search for the "DQS I/O Block Instantiations" section, and modify the number of instantiations to match the target memory configuration.

If the reference design is configured to control multiple FCRAM devices, then the designer must look at signal loading. The clock, address, and control signals are shared across all memory devices. As devices are added, the performance of these signals will decrease. Therefore, IBIS and other board level simulations should be performed to determine the optimal loading and placement of these signals. Typically, the memory vendor supplies additional information on suggested loading. If duplicate drivers are required, then the HDL code should be modified to support these changes.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
02/27/02	1.0	Initial Xilinx release.