APPLICATION OF AN ASYNCHRONOUS FIFO IN A DRAM DATA PATH

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Science

with a

Major in Electrical Engineering

in the

College of Graduate Studies

University of Idaho

by

James B. Johnson

December 2002

Major Professor: R. Jacob Baker, Ph.D.

AUTHORIZATION TO SUBMIT

THESIS

This thesis of James B. Johnson, submitted for the degree of Master of Science with a major in Electrical Engineering and titled "Application of an Asynchronous FIFO in a DRAM Data Path," has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor ______________________________ Date__________
R. Jacob Baker

Committee Members ______________________________ Date__________
Jim Frenzel

______________________________ Date__________
Larry A. Stauffer

Department of Electrical and Computer Engineering Chair ______________________________ Date__________
Joseph J. Freely

College of Engineering Dean ______________________________ Date__________
David E. Thompson

Final Approval and Acceptance by the College of Graduate Studies

______________________________ Date__________
Charles R. Hatch

Abstract

As CMOS dynamic random access memory (DRAM) processes and voltages continue to scale, DRAM array access latency remains relatively constant. Faster logic resulting from process scaling is offset by timing latency associated with constant die size for new generation array densities. The benefits of process scaling are also countered by rising interconnect resistance and voltage scaling. All totaled, the benefits of process scaling are not fully realized and DRAM array access have remained fairly constant for sub-micron processes.

However, system designers are demanding higher sustained bandwidth from DRAM devices. This forces the DRAM circuit designer to find ways to increase data bandwidth from the DRAM outputs while still suffering from limited improvements in array timing. Therefore, next generation devices have adopted data prefetch architectures. By prefetching larger amounts of data on each read access, data throughput at the DRAM output for data serialization can be sustained at a higher rate.

This thesis confronts the problems associated with sustaining data throughput in the DRAM read data path while at the same time attempting to keep data prefetch sizes low. Because initial read data latency is separated from data throughput for consecutive read accesses, a discussion of the relationship between read data latency and array access latency is necessary before introducing a circuit topology for mitigating the throughput problem with minimum impact on read latency. The chosen method is to place an asynchronous FIFO in the data path between the secondary flip-flops of the DRAM array and the synchronous data output serializer. A method for evaluating the performance of the FIFO is established that can be generally applied for other asynchronous FIFO applications.

Acknowledgments

None of the work contained in this thesis would have been possible without the help and support of many people who have contributed to my education and those who have encouraged and supported me throughout my education. I would like to acknowledge the efforts of my thesis committee in their review of this document. I would especially like to acknowledge the hard work of the excellent professors from University of Idaho. Without their hard work, none of my accomplishments, or those of my peers, would be possible.

I would like to single out Dr. Jake Baker for his untiring devotion to providing help and direction to his students. His example and motivation have encouraged me to continue to pursue the career and education directions I have chosen.

Finally, I would like to thank my boss, Brent Keeth. His trust and contributions with many great innovations encouraged me to experiment and take the risks necessary to further our vocation.

Dedication

This thesis is dedicated to my wife, Cassandra. I thank her for tolerating my absence for the long hours required to pursue a degree.

Table of Contents

List of Figures

List of Tables

List of Graphs

1. Introduction

Dynamic Random Access Memory (DRAM) components incorporate several primary circuit principles and topologies that are a standard part of electrical engineering curricula. We find primary use of the principle of charge sharing as well as common circuits such as charge pumps, assorted digital logic and signaling interface circuits all used in the storage and retrieval of digital information [1]. The advantage that DRAM has over other storage medium is that the density and manufacturability of the device is based on the 1T1C storage cell shown in Figure 1.1. The density and cost advantages of the 1T1C cell are offset by the performance impact of refresh and digit line sensing necessary in order to access the stored charge in the capacitor of the 1T1C cell.

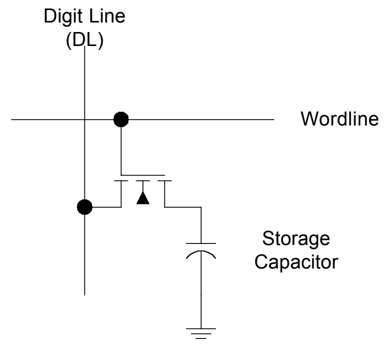


Figure 1.1 DRAM 1T1C Cell

Recent advances in processor performance have lead to higher demands from the memory interface. DRAM circuit designers must consider power, area, process and voltage scaling challenges when designing devices that meet unceasingly increasing bandwidth requirements. Performance limitations of the 1T1C based memory array have lead to specification changes such as increasing data prefetch sizes from the memory array in order to allow increased data bandwidth from the output of the memory device. Changes in device specifications for data prefetch depth are not keeping pace with output bandwidth requirements, thereby forcing DRAM design engineers to improve circuit design methodologies and computer aided design (CAD) tools in order to meet performance challenges.

Combined with these design challenges are consideration of standards setting bodies and legacy operational specifications. Standard setting bodies consider a set of specifications that ensure a commonality between devices designed by different manufacturers. Because the specifications require consensus, compromises that lead to obstacles for improving performance are often a side affect of the standards setting process. Conflicting agendas and variations in manufacturers capabilities in design and manufacturing of DRAM devices further complicate the advancement of the art of DRAM design.

This thesis will focus on one aspect of a problem encountered in furthering the performance capabilities of a DRAM device. The problem relates to accessing stored data in the memory array and the timing of the transfer of that data to the external data bus of the DRAM device. We mention the standards setting process and legacy specifications of DRAM devices because the problem we will consider is related to the maintenance of the DRAM standard in relation to timing of read data from memory to the processor otherwise known as read latency. DRAM operation will not be a focus of this work. We will focus on DRAM operation only as it applies to the problem described in this paper. An overview of DRAM data path operation for accessing read data will be presented in Chapter 2.

The first part of this work will examine the read data timing specification and how increasing clock frequency has lead to the requirement for a clock alignment circuit such as a delay locked loop (DLL) [2,3,4] as part of the common set of circuits used in the design of synchronous DRAM. We will also see that the prefetch architecture required for meeting data throughput requirements at the data outputs has further forced us to consider data pipelining techniques for maintaining data throughput from the array to the data outputs. Timing properties of the DLL circuit and the relationship between internal read latency and column access timing are examined in relation to the application of an asynchronous pipeline used for providing high data throughput to the outputs. A brief treatment of the operation of the DLL circuit is presented in Appendix A of this thesis. If one is unfamiliar or requires a refresher in DLL circuit operation turn to the appendix and review this material before continuing to the first section.

2. Overview of DRAM Read Data Path

There are many references available that discuss the operation of DRAM memory array core read and write operations [1]. Most of these references only delve into the operation of the memory core and supporting circuitry and often overlook the logic interface necessary to control the data input and output operations. This oversight is understandable considering that prior to the introduction of synchronous dynamic memories, the logic operations necessary for controlling the data path between the physical edges of the memory core to the I/O pads were rather trivial. With the introduction of synchronous memory, many of the control and timing responsibilities for transferring data between the memory device and the processor in a computer system have migrated from the memory controller logic to the memory device. Additionally, as memory clock frequencies have increased so too has the complexity of the control and timing problems for controlling the data I/O logic path between the memory core and the data I/O pad. Figure 2.1 provides an illustration of the section of DRAM logic that is the focus of this paper. Figure 2.1 will also provide the necessary background and terminology for a more detailed discussion.

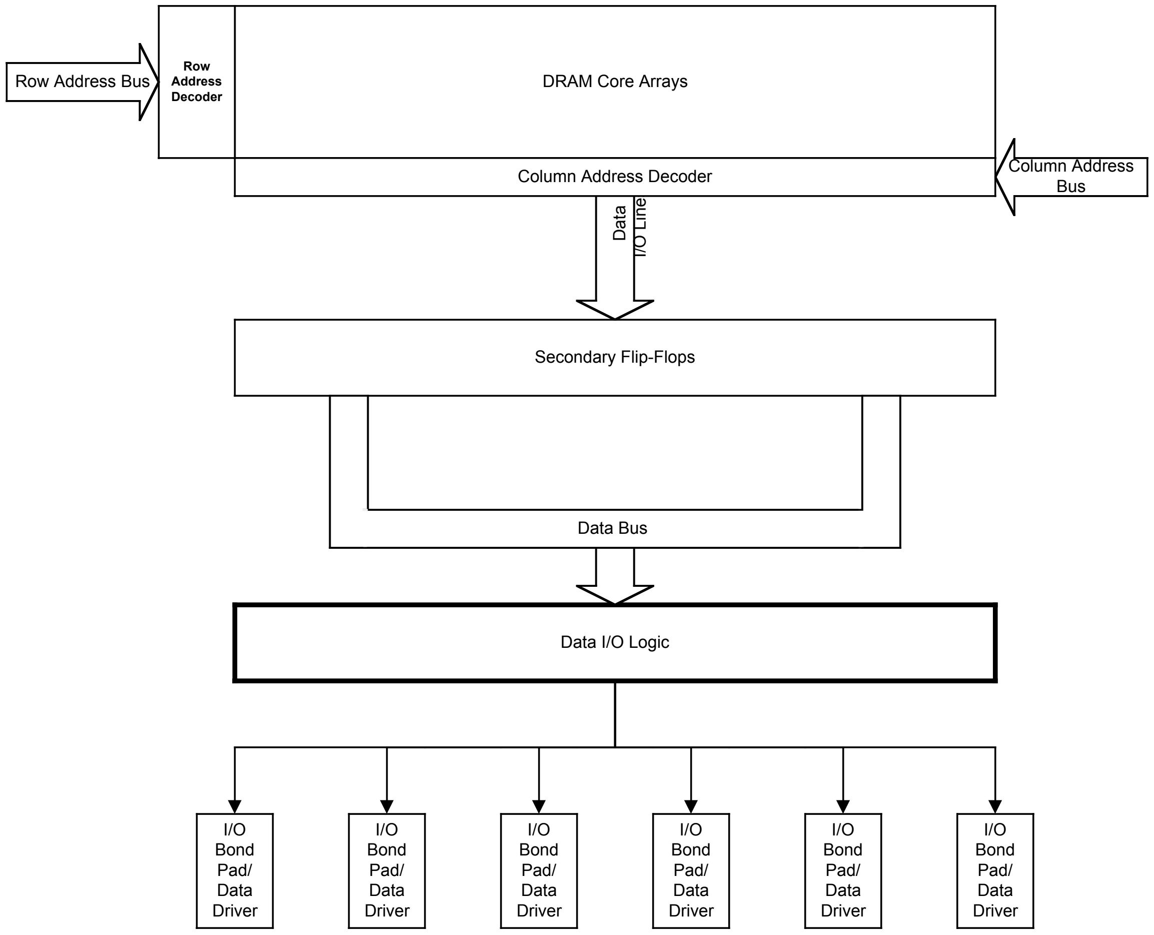

Row Address Bus
Row Address Decoder
DRAM Core Arrays
Column Address Decoder
Column Address Bus
Data I/O Line
Secondary Flip-Flops
Data Bus
Data I/O Logic
I/O Bond Pad/ Data Driver
I/O Bond Pad/ Data Driver
I/O Bond Pad/ Data Driver
I/O Bond Pad/ Data Driver
I/O Bond Pad/ Data Driver
I/O Bond Pad/ Data Driver

Figure 2.1 DRAM I/O Logic Path

In this section, we will examine a potential design for the data path of a dynamic memory device. There are many variations possible for DRAM data path design but in this case we will examine the design in very general terms without being distracted with specific circuit considerations or topologies. This chapter will provide a frame of reference necessary for more detailed discussion about read latency and the role of the command decoder and DLL in maintaining proper read latency. Starting in Chapter 4, we will examine in detail solutions relating to data throughput issues when interfacing a data prefetch architecture from the DRAM array core to a high-bandwidth serial data output structure.

In Figure 2.1, we see the block on the top of the diagram labeled "DRAM Core Arrays." Here we are including the row and column decoders necessary for accessing the 1T1C memory cell as well as the sense-amplifiers used for detecting the logic state indicated by charge in the storage capacitor. Before a specific bit can be transferred out of the arrays, a row must be open by strobing the wordline connected to the gates of the access transistor shown in Figure 1.1. This causes charge sharing between the capacitance of the precharged digit lines (DL) connected to the drain of the access device and the storage capacitor connected to the source of the access device.

When a memory array is first accessed, the wordline or row interconnect line is driven to turn on the access transistor that was shown for the 1T1C cell of Figure 1.1. The charge stored in the capacitive storage device is then transferred to the digit line. At this point, sense amplifiers, that are part of the memory array core, sense the stored charge relative to a reference voltage and drive digit line pairs to CMOS levels. The voltage separation of the digit line pairs is then transferred through I/O transistors to data I/O lines labeled in Figure 2.1. Figure 2.2 illustrates these connections in more detail. The wordline connected to the gate of the access device in Figure 1.1 must be driven to a voltage higher than VDD. This is required because the n-channel device cannot pass a voltage between the drain and source higher than Vgs-Vt before it turns off. The full CMOS level on the DL lines is then transferred to the I/O lines through the I/O devices during a column access. The column (page) access occurs enough time after the wordline (row) access so that full CMOS levels are restored to the digit lines and the memory cell.

Figure 2.2 is a diagram depicting the connecting circuits responsible for detecting the logic state stored in the memory cell. Before a wordline in one of the arrays is fired high, the ISO device for the opposite array must go low in order to isolate the digit lines from the array opposite from the accessed array. The opposite array also has precharge devices (not shown) holding the opposing digitlines at a voltage of VCC/2 on the inside of the ISO devices. The ISO device gates are driven to a level above VCC in order to allow a full VCC level to be transferred through the n-channel device. After the sense amplifiers have been fully engaged, a full CMOS level is driven back into the memory cell. This serves the purpose of refreshing the memory cells until a write might occur. When the memory cell is accessed, the voltage difference between the DL and DL_ lines is sensed with one line connected to its respective data line through the access device enabled by the decoded wordline and the other data line held at the precharge value of VCC/2 since its wordline holds its access device off.

For example, if the wordline for DL is turned on and there is excess charge stored in the storage cell, then the voltage on DL will increase because of the charge sharing between the capacitor and the digit line. Conversely, if there is a lack of charge in the storage cell, then the voltage on DL will decrease as charge is shared between the previously precharged capacitance of the DL and the storage capacitor. The sense amplifier will compare the change in voltage caused by charge sharing between the storage cell and the capacitance of the DL line with the precharge voltage (VCC/2) stored on an opposing DL line from a row that has not been accessed.

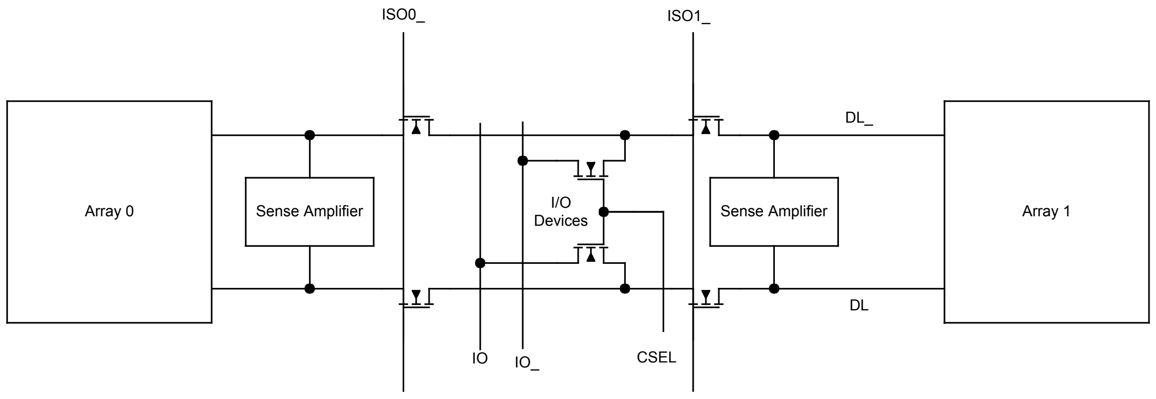


Figure 2.2 I/O Devices Connecting Digit Lines to I/O Lines

There are logic circuits external to the memory arrays that control the timing of data line precharging, row decode/wordline drive and logic for controlling the timing of the sense amplifier enable signals. We should also note that this description might seem like a trivial treatment of the intricacies of accessing and controlling a memory array. For our purposes, we will not delve into the specifics of memory array access. We only want to give perspective as to where the data we will talk about in future sections originates.

Referring to Figure 2.1, we see the signals labeled “Data IO Lines.” This section of the diagram represents interconnect and logic that extends from the edge of the memory array to the secondary flip-flops. The purpose of this logic is to provide an interface to the secondary sense amplifiers that are in turn used for quickly driving data to full CMOS levels in the array peripheral logic. The drive capabilities of the array sense amplifiers shown in Figure 2.2 are necessarily weak because of the large parasitic capacitance associated with array interconnect and the density of the circuits involved. When viewing Figure 2.2, the interconnect labeled IO and IO_ are analogous to the connections labeled “Data IO Lines” in Figure 2.1.

Referring to Figure 2.2, when a column access occurs, the signal CSEL is driven high to turn on the n-channel I/O devices. The CSEL signal is generated by the column address decode logic. Care in sizing the I/O devices relative to the digit line capacitance is important in order to allow sufficient transfer of charge to the I/O lines while still providing a low resistance path from digit lines to I/O lines.

After charge is transferred from the DL to the IO lines, further sensing is necessary. Like the DL signals, the IO signals are precharged prior to sensing. Precharging the IO signals is necessary for fast data detection by the secondary flip-flops. The DLs are driven to full CMOS levels but the I/O devices cannot pass a full VDD level. Also, the parasitic on the IO lines is large which further delays the page access. Therefore, secondary flip-flops are employed at the interface of the array cores to the device peripheral logic. The block labeled “Secondary flip-flops” in Figure 2.1 represents this section of the array logic. The circuits in this block can be direct current sensing amplifiers or a another set of sense amplifiers with more drive capability relative to the array sense amplifiers [1]. The sense amplifier style of secondary flip-flops is often referred to as “helper flip-flops.” Figure 2.3 is an example topology of a helper flip-flop (HFF) circuit.

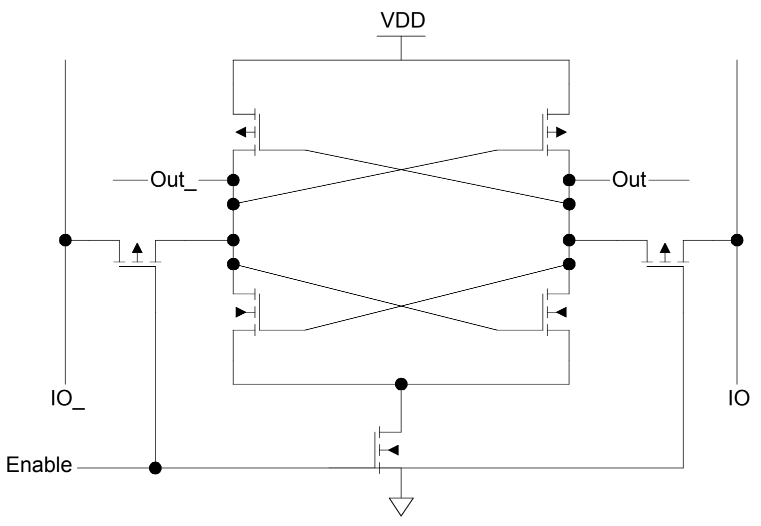


Figure 2.3 Helper Flip-flop Circuit

The HFF circuit is a set of enabled cross-coupled inverters. While the enable signal is low, the latching mechanism is shut off while the IO signals transition after an array access. As the IO signals transition, the enable signal is driven high. When the enable signal is driven to a high voltage, the access devices used to transfer charge from the IO signals to the HFF circuit are shut off and tail current flows to ground through the tail device. Very little IO signal voltage separation is required because the cross-coupled inverters act with positive feedback to latch the correct data. The ability of the HFF circuit to detect a small signal input on the IO lines is one advantage of the HFF circuit. Another advantage of this circuit is the small area of silicon consumed. This small size can allow the HFF circuit to be drawn to pitch with the array IO lines.

2.1 DRAM Interface Signals

At this point, we have examined how data is transferred from the 1T1C memory cell to the edge of the memory array. Data is now physically located at a point in the chip just prior to synchronization for serial output. This will lead us into a discussion of how we can pipeline data from array accesses and maintain the required data throughput to feed the synchronous circuits that generate output data. Notice in the preceding discussion that any mention of a clock signal to synchronously access the memory array is missing. Accesses to the memory array are performed asynchronously while all command and data I/O from the external world are synchronized with a system clock. Figure 2.4 is an external, top-level view of the signals into and out of a DRAM memory device.

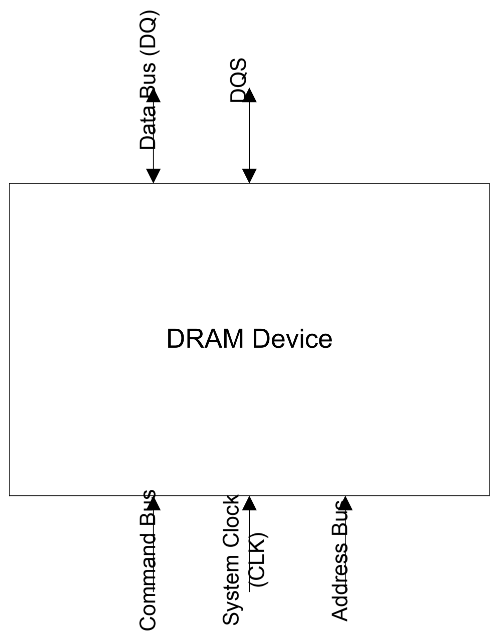


Figure 2.4 Top-level View of DRAM Device

In Figure 2.4, we see that there is a command/address clock, CLK, which is used to time the arrival of commands and addresses to the DRAM device. On the data side, there is a second clock, DQ Strobe (DQS), associated with the timing of data driven to and from the DRAM device. When data is to be written to the device, the DQS signal arrives center aligned to the data signals. By center aligned, we mean to say that the strobe is timed relative to the data to optimize the reliable capture of data into the device. The CLK signal is driven similarly aligned to the command and address signals but is unidirectional from the clock source to the DRAM device. In the case of the DQS signal, it is said to be "bi-directional." This means that the DQS signal is sourced with the data both when read data is driven from the DRAM and when write data is driven to the DRAM. This implies that the DRAM device must be capable of driving the DQS signal timed with read data in a prescribed manner. We will examine the timing and function of the DQS signal as a clocking signal in Chapter 3. Also, we will see that the CLK signal is not totally divorced from data bus timing. The CLK signal is used to provide a continuous timing reference for cycle-based timing of read and write operations.

The command bus consists of several control signals that, when combined and latched with the CLK signal, indicate an operation for the DRAM to perform. For example, most current DRAM command busses consist of a row address strobe (RAS), column address strobe (CAS), write enable (WE), chip select (CS) and clock enable (CKE). For instance, the RAS and CAS signals are used to indicate the application of a valid address for row and column accesses respectively. The WE pin indicates whether a valid CAS signal is for a column read or write access. The CS signal validates a command to a particular device. CKE is used for entering low power operating modes by shutting off clock distribution paths internal to the DRAM. Furthermore, certain combinations of the command signals allow access for programming optional modes of operation for the DRAM. The command bus centralizes all operational communication between the chipset and the DRAM device.

This brief description of an array access gets us to the point where data is entering the area of the DRAM device that is relevant to the focus of this thesis. An important point to note is that going forward we will refer to the portion of the data path extending from the edge of the memory array to the I/O pads as "data path." In the DRAM design world, the term "data path" often refers to the section of the memory array extending from the access device of the 1T1C cell to the data I/O interconnect lines that carry data to the edge of the memory array including the secondary flip-flops. Again, it is important to note that early DRAM designs did not require much logic for transferring data to the data I/O pads. Therefore, the logic from the access device through the secondary flip-flops was essentially the "data path." We will see that increasing complexity and performance requirements in synchronous DRAM (SDRAM) interfaces has lead to more complex circuits for controlling data transfer between the secondary flip-flops and the data I/O pads.

2.2 DRAM Data Bandwidth Requirements

System designers have demanded an increase in data bandwidth from DRAM devices. One alternative involves increasing the physical data bus width while keeping frequency constant. The cost of this improvement is born by the system builders because of the increase in pin count of application specific integrated circuits (ASIC) and the increase in the number of routed signals for inter-chip interconnect. Because of cost issues, system designers tend to avoid this option.

Another method for increasing data bandwidth is to increase DRAM clock frequency. This method allows the physical bus width to remain constant with a linear increase in data bandwidth proportional to the increase in clock frequency. Implementing technology changes for increasing DRAM clock frequency has resulted in increasing research and development costs for DRAM manufacturers. Current system bandwidth improvements have relied more on increasing the DRAM clock frequency and less on increasing physical data bus width. Although in some applications increasing both the bus width and the operating frequency has resulted in drastic performance improvements.

Another recently implemented design change for higher performance memory has been the introduction of double-data rate DRAM (DDR-DRAM). Previous SDRAM designs have driven data on the positive edge of the system memory clock. With DDR-DRAM, the data from the DRAM is driven on both the positive and negative edge of the system clock as shown in Figure 2.5. This method of data delivery results in a doubling of serial data bandwidth relative to standard SDRAM. DDR-DRAM increases data bus performance but still introduces problems for the system designer as it becomes more difficult to close timing budgets as the bit valid times become increasingly narrow.

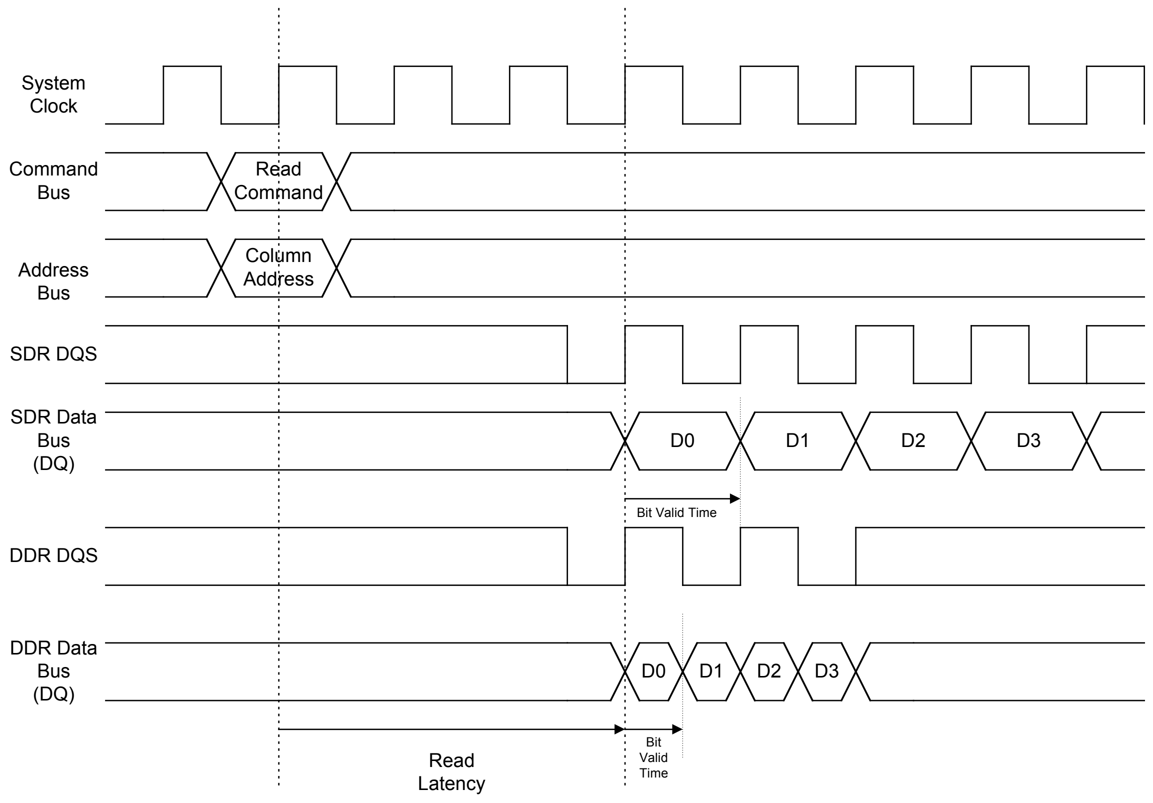


Figure 2.5 SDRAM vs. DDR-DRAM Data Bus Transition Timing

2.3 Data Prefetch Architecture

As the serial data bandwidth requirements increase for DRAM data buses, there is a drastic impact on the data path design and the I/O interface. We must first consider that minimum array access times have remained fairly constant, as process and voltage have scaled [5]. The reason access times have remained constant is that process improvements are partially countered by the accompanying voltage scaling. But also consider that the physical size of memory arrays as well as array density also increase at a similar rate compared to process scaling thus keeping die sizes for new memory generations fairly constant. Also, the parasitic impedances associated with the address and data interconnect in the array are becoming more resistive as process scales which also offsets some of the performance gains of the scaled transistors. When all of these effects are considered, the array access delay has remained relatively constant as process and voltage scales.

This leaves us with the burden of having to increase the serial bandwidth of the DRAM output data bus while the array access delay remains constant. If we were able to decrease the array access delay at the rate which serial data bandwidth was increasing, then no changes would be required in the data path. The obvious solution to this problem has been to increase the data "prefetch" size from the memory array in order to feed data to the high bandwidth data bus. We use the term prefetch to describe the operation of accessing data from the memory array that is not immediately output from the device. When a read command is issued to the DRAM, data is accessed from a column or columns from a previously accessed row. This data must reach the output data path before the expected read latency has expired.

Data prefetch requirements in the DRAM can be traced to microprocessor architecture. Modern processors rely on fast data caches in order to maintain optimal bus utilization rates [6]. Most processors prefetch data from main memory in order to fill cache lines; thereby, taking better advantage of data locality. This has led to the exclusive use of burst mode operation from SDRAM devices.

With the development of SDRAM, burst mode operation of array accesses became standard for DRAM. Burst mode operation is when a single array access command involves multiple columns of array data. For example, in Figure 2.6 we see a timing diagram showing the issuance of a read command and address to the DRAM command/address bus. When the expected read latency has expired, data is driven on the data bus DQ pins. Notice that data changes occur with each clock edge following a single access. In this example, we say the burst length is four since four serial data bits are output from each DQ pin on the data bus. The implication of burst mode operation is that a single column address actually accesses multiple columns in the array.

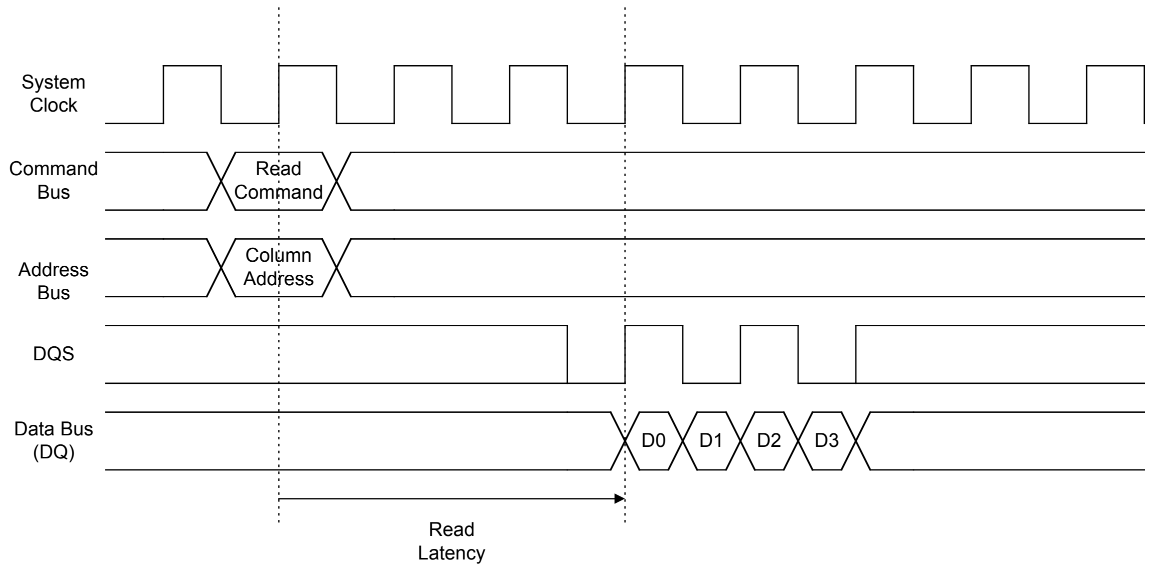


Figure 2.6 Read Command with a Data Burst Length of Four

A numerical example of the impact of a prefetch architecture on the data path is as follows. Say that we have a data bus that is 16 bits wide. Let us also consider that the memory array has 512 columns in each row. If we are to support a burst of 4 bits from each DQ on the data bus, we must prefetch 64 bits of data from the memory arrays. This also means that a single column address will access 4 columns in the array giving us an 8-bit address space to access 128 possible starting addresses. Figure 2.7 is a repeat of Figure 2.1 only the values from our preceding example are assigned to the bussing.

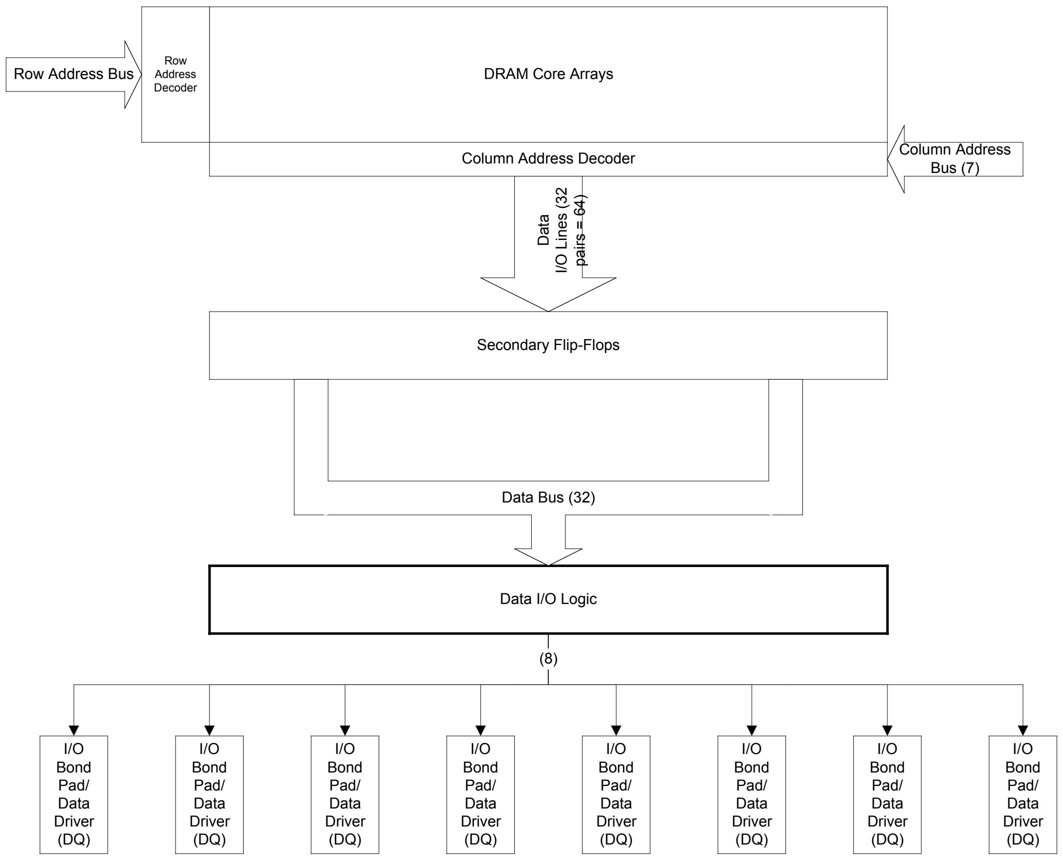


Figure 2.7 Prefetch Bus Sizing for 8-bit DQ Bus

The complications do not end with data bus width requirements. Maximum system clock frequency will also impact the size of the prefetch bus in the data path. Remember that array column access delay has remained relatively constant, as each new array density has been developed. When we increase the serial data frequency we reach two potential limitations. The first limitation arises from read data delay. This is the delay encountered between a read command and when data is driven on the data bus. As the clock period decreases, so does the number of clock cycles that pass between the issuance of a read command and when data is available. We define the read access delay measured in clock cycles as read latency. We will pursue a more rigorous examination of read latency in the next chapter.

The second limitation is the minimum cycle time of the column access. Column cycle time is limited by the time required to cycle precahrge and evaluation of the IO lines. There is also logic overhead related to cycling the column address decoder. Note the distinction between the delay of the first column access (read latency) and the period between consecutive column accesses (column cycle time). Figure 2.8 illustrates two consecutive read commands with a read latency of 3 cycles and a burst length of 4. With a burst length of 4, the minimum column cycle time for continuous data is 2 clock cycles.

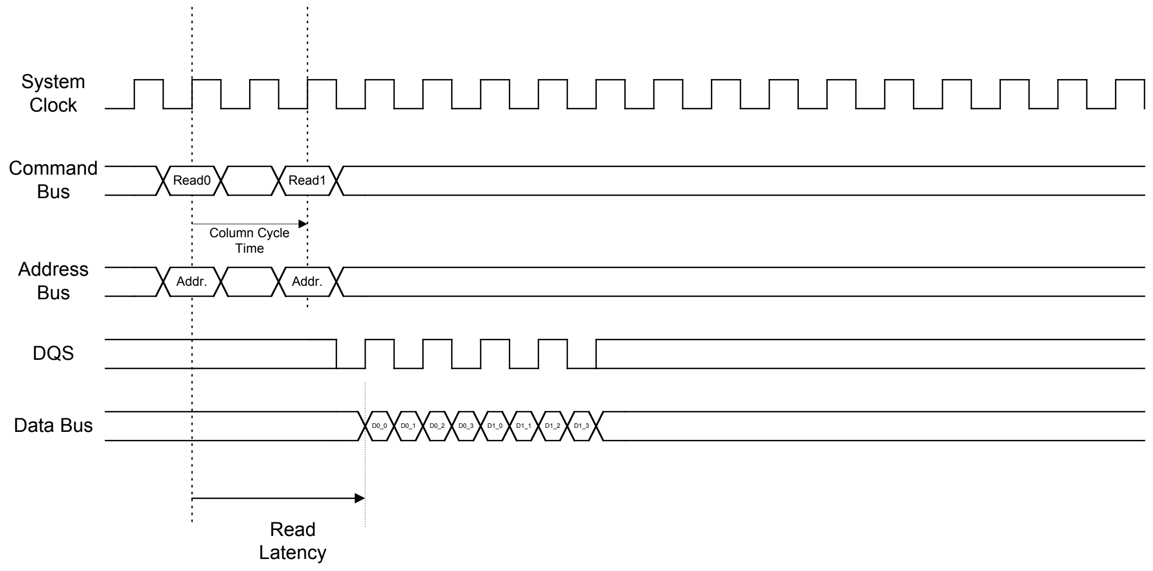


Figure 2.8 Consecutive Read Commands

If we are required to supply a DDR data bus with a minimum burst length, BL, of 4 bits per access, then we know each access would occupy 2 system clock cycles. If we consider a minimum column cycle time, CCT, of 5ns then the maximum clock frequency, f_{cm}, at which the DRAM can operate is determined by:

$$f_{cm} = \frac{BL}{2 \cdot CCT} = \frac{4}{2 \cdot 5ns} = 400MHz \quad \textbf{(2.1)}$$

However, if we increase the minimum burst length to 8 and apply Equation 2.1, the DDR-DRAM can potentially operate at 800 MHz. The downside of this change is that the burst length may not meet processor cache line granularity requirements. Another disadvantage is that larger burst lengths can lead to larger column prefetch sizes. Larger column prefetch sizes can lead to higher power consumption and a significant increase in die area occupied by data path logic.

In this chapter, we have examined the architecture of the DRAM data path. This section has followed data flow from the storage capacitor in the 1T1C memory core through the sense amplifiers, column decoder and helper flip-flops. The goal of this section is to provide us with a concept of where data originates following a read command. In Chapter 3, we will better define read latency and discuss the source synchronous signaling protocol. After we have visited these topics, we will start our discussion of the research performed for this thesis beginning in Chapter 4.

3. Read Latency Timing for High-speed DRAM

In this chapter, we will continue laying the groundwork that will allow us to discuss the solutions for maintaining data throughput in the DRAM data path. The previous sections have described the principles of an array access and burst mode operation of the DRAM data bus. We have also described how continually increasing bandwidth requirements of the data bus combined with relatively constant array access times has led to a prefetch architecture for current SDRAM designs. So far, our discussion has focused on the asynchronous access of the DRAM array. This section will focus on the synchronous portion of a DRAM read access. After we examine the synchronous portion of the read access, we will be fully equipped to study the circuits used to interface the asynchronous access circuitry with the synchronous output circuitry.

3.1 Read Latency

Read latency for a DRAM is defined as the delay between a column read command received at the DRAM command/address pins and the time when data is driven from the DRAM data I/O (DQ) pins. For a synchronous DRAM, this time is specified as the number of clock cycles between the clock edge at which the read command is clocked into the DRAM and the clock edge when data is driven on the DQs. Figure 3.1 is an illustration of how DRAM read latency is specified for a synchronous DRAM.

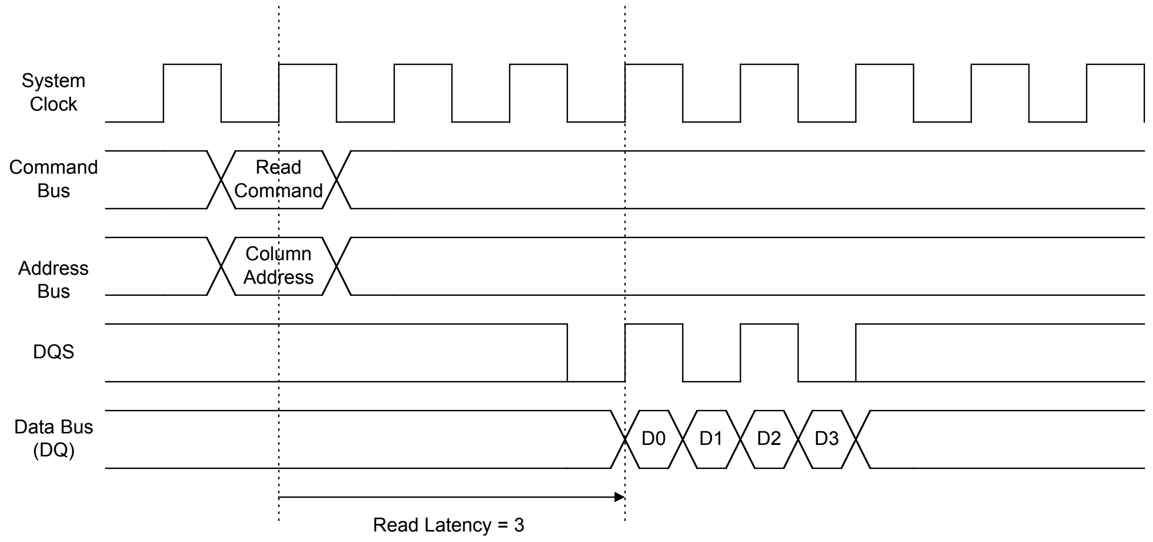


Figure 3.1 Read Latency Definition

Notice in Figure 3.1 how DQ data and the data strobe, DQS, are edge aligned to the DRAM clock as it is driven from the DRAM. The clock edge alignment of the data is accomplished using a clock alignment circuit such as a delay locked loop (DLL). The role of the DLL in maintaining read latency will be discussed later in this section. Aligning the read data to a clock edge at a predetermined number of clock cycles following a read command serves two primary purposes. First, having predictable data latency is useful for testing the DRAM during manufacturing. Predetermined data latency simplifies DRAM testing because it makes it possible to predict when data should be driven by several DRAM tested in parallel; thus, freeing tester resources and resulting in shorter test times. Second, the memory controller can have the option of counting the number of clock cycles following a read command and grossly estimate the arrival time of the data. In actual operation, the memory controller captures DRAM data with each transition of the DQS signal. This signaling standard for data transfer is often referred to as 'source synchronous' timing. Figure 3.2 is an illustration of one possible connection between memory devices and a memory controller chip in a memory sub-system. This illustration will help us conceptualize the discussion to follow.

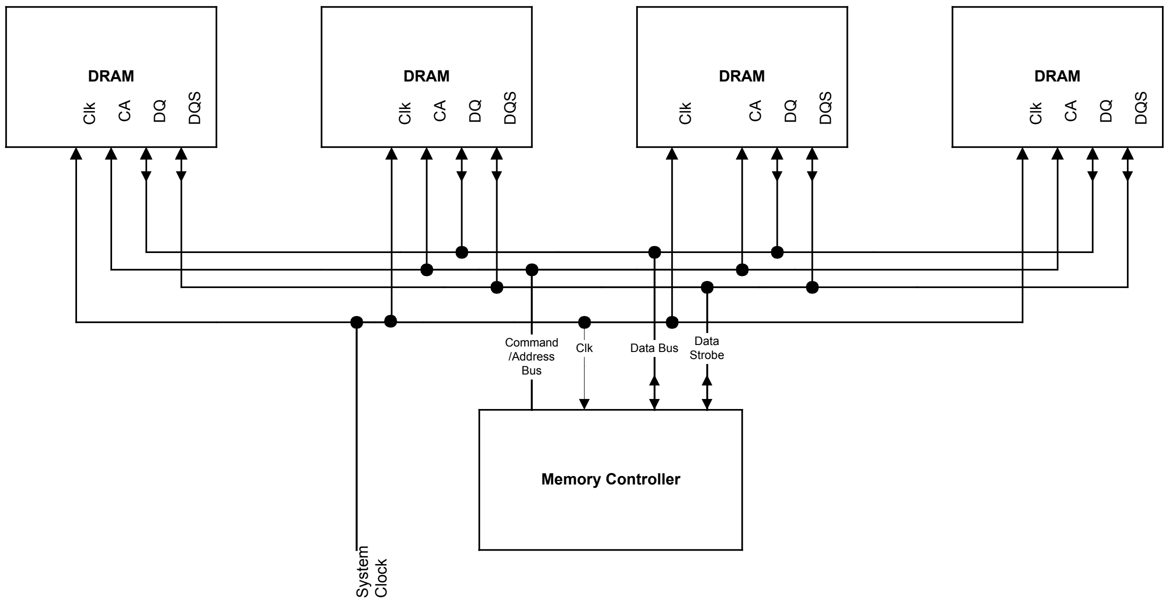


Figure 3.2 Clock, Data and Command/Address Memory Controller Connections to DRAM Devices

3.2 Source-synchronous Interface

In current generation Synchronous DRAM, the DQS signal is timed to transition synchronously with the data driven from the DRAM. This is referred to as a 'source-synchronous' signaling protocol. Figure 3.1 shows how the data strobe, DQS, is aligned to the clock along with the data. When data arrives at the inputs to the memory controller following a read command, there is a period during which the data signals across the data bus share a temporal alignment synchronized to the memory clock. Figure 3.3 demonstrates this idea. We see that the data transitions for each of the bits across the bus are not perfectly aligned. There is a valid bit overlap period across the bus when all of the data bits have a common temporal relationship. We say the bits across the data bus have a common temporal relationship when the data bits in a data word generated at the sending device appear in parallel for a period of time at the receiving device. This temporal alignment is known as a data valid window. In the case of Figure 3.3, the data burst length is 4 and in each bit time of the burst there is a valid window when the members of the data bus share a valid transition period. The critical timing parameters specified for the width of the data valid window are referenced to a transition of the DQS signal. The reason critical timing parameters are referenced to the DQS signal is because DQS is the clock for the source synchronous interface; although, at low clock frequencies, some systems are known to use the main system clock for data capture.

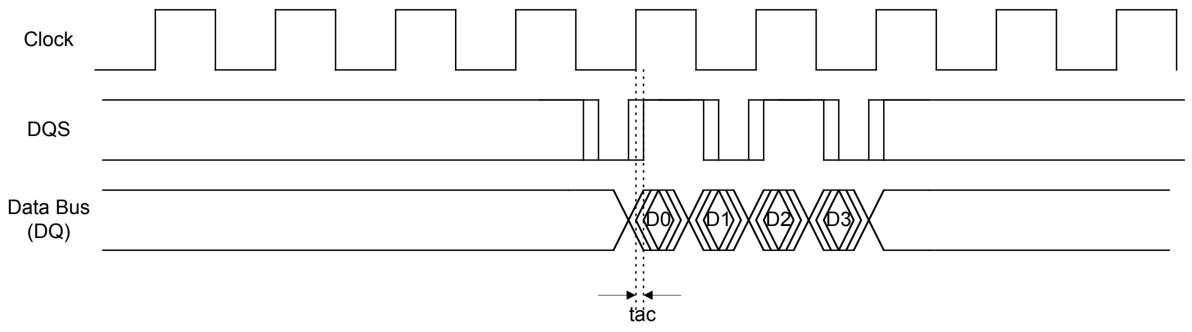


Figure 3.3 Data Bus and DQS Signal Skew Diagram

Current DDR-DRAM standards [7] call for a timing specification, t_{ac}. In Figure 3.3, t_{ac} is labeled as the time between an edge of the system clock and the time data is valid on the data bus. As long as t_{ac} is much less than the clock period, this specification allows us to specify a cycle-based latency for read and write data on the data bus. The t_{ac} timing parameter has historically been a manageable fraction of a clock cycle. As clock frequencies have increased, and t_{ac} has become a larger portion of the clock period, the use of a DLL to align data to the system clock with minimum t_{ac} has become necessary. However, at very high frequencies, some of the weaknesses in DLL designs and clock distribution networks that can cause short-term timing jitter out of the DLL have become significant. Most of the jitter problems can be traced to coarse timing adjustment of the DLL delay line, a lack of supply noise immunity for the delay elements and inadequate power bussing and voltage regulation of DLL circuitry. Design improvements have minimized some of these problems

[8] but the DLL will always contribute some jitter on the data bus. We will more closely examine the role of the DLL in data bus timing later in this chapter.

As frequency continues to increase, process variations, geometric mismatches, data pattern sensitivity due to simultaneous switching operations (SSO) and channel limitations causing inter-symbol interference (ISI) have become significant contributors to data bus skew. Many of these problems continue to be overcome by advances in system signaling technology; thereby, allowing continued increases in clock frequency.

Because the DDR interface, coupled with decreasing clock periods, has significantly reduced the data valid window, a new trend may be developing where the t_{ac} specification is actually allowed to increase as a fraction of the clock period. At the same time the timing specification, t_{dqsq} measuring the variation in timing between the data bus and the DQS signal has been tightened. Figure 3.4 illustrates the relationship between these timing parameters. This trend allows us to continue to increase system clock frequency by taking advantage of the source synchronous interface for data transfer. Also, small swing signaling technologies such as SSTL2 [9] have contributed to increasing the channel bandwidth. These trends still do not relieve the memory controller interface logic from the task of centering the DQS pulse in the middle of the data valid window for data capture.

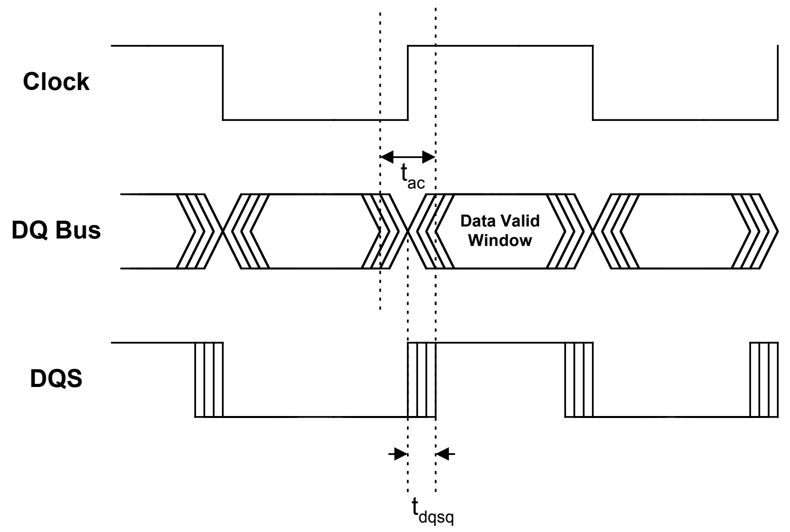


Figure 3.4 t_{ac} and t_{dqsq} Definition

There are various methods that have been proposed for capturing data at very high speeds [10, 11, 12]. In this discussion, we are interested in clock to data timing methodologies and not solving signal integrity problems. Some of these methods involve on-chip timing compensation circuits that serve the purpose of optimizing the alignment of the source synchronous data strobe and the center of the data valid window. This alignment can be derived using multiple delay line taps from a DLL selected by the results of channel initialization using training patterns (patterns generated as cyclic redundancy codes using linear feedback shift registers) [12]. Other proposals have suggested that individual members of the data bus can be independently phase aligned using individual delay lines controlled by the capture of expected data patterns [13].

It is possible to conceive of a situation in which the skew across the data bus is large enough to not allow a temporal word alignment of the data bus signals. In this case, the 'data eye' is closed so that capture with a single DQS signal is not possible. Turning on the persistency of an oscilloscope and measuring the toggling signals across the data bus gives us a picture of the data eye. If there are periods where members across the data bus do not share a temporal commonality, then the eye is considered closed. Figure 3.5 is an example of a closed data eye.

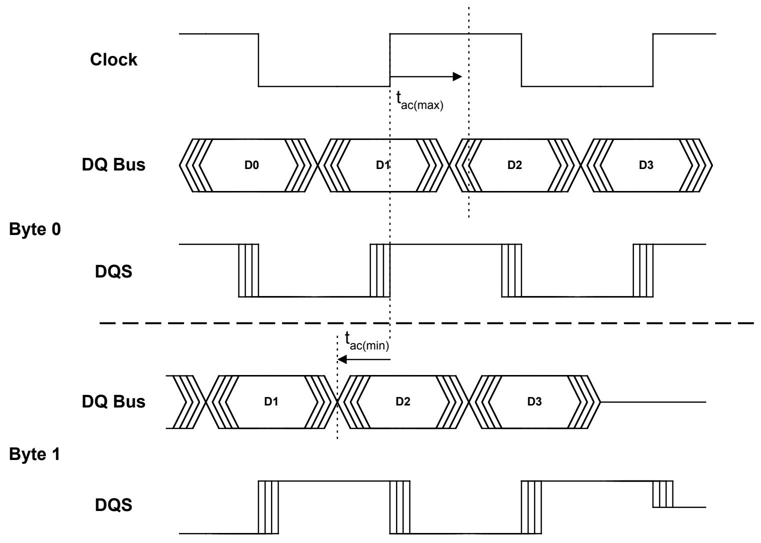


Figure 3.5 Closed Data Eye for Capture Across Two Bytes of the Data Bus

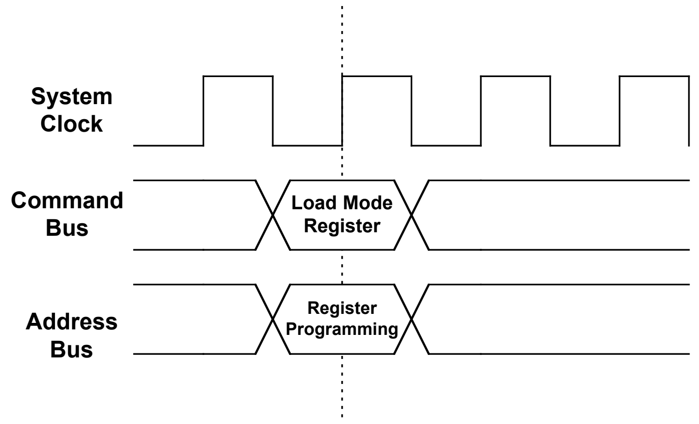
Physically wider data busses have a greater chance for a closed data eye because of data bus skew caused by on-chip clock skew, process gradients across a die, clock jitter, package parasitics and system level interconnect differences over a wide data path. For this reason, separate DQS signals may be assigned to groups of signals that are subsets of the overall system memory bus. These subsets usually represent separate memory chips or even subsets of separate memory chips. This means there is possibly more than one DQS signal routed per memory chip to service the entire width of the DQ bus. For instance, a single DRAM with a 32-bit wide data I/O bus may require 4 DQS signals to service each byte. This is because at high frequencies, the data skew across the 32 I/Os may be large because of differences in process characteristics across the die. Also consider data-dependent switching characteristics affecting power supply rails supplying I/Os at different locations in the DRAM die (SSO is in this category). We also have to be concerned with trying to route a single DQS signal on memory modules and internally to the memory chip. Not only are there on-die characteristics that can affect the width of the data valid window for the data I/Os, but factors external to the DRAM including differences in signal terminations and PCB routing geometries. We also need to consider electrical phenomenon such as signal cross talk, transmission line effects and inductive noise that in turn causes power supply noise. By routing DQS to service a subset of the overall data bus, the DQS signal will have a tighter timing relationship to the data valid window. That is because DQS is driven from the same area of the DRAM die or simply from the same DRAM that is sourcing the data. Also, subdividing the I/O bus for multiple DQS signals simplifies the routing problems at the system level by providing timing margin for a certain level of routing mismatch between subsets of signals across the overall system bus.

After data and DQS are driven from the DRAM I/O, the DQS signal is delayed relative to the data signals in order to center the strobe in the middle of the data eye at the receiving end of the channel [14]. The data strobe alignment shown in Figure 3.1 is accomplished with the use of a controlled delay line on the system printed circuit board (PCB), or by using Phase-locked loops (PLL) or DLL capture clock circuits at the system and/or chip level [14]. There are other methods outside the scope of this work that can be implemented in on-chip logic that can center align a source-synchronous clock with the data valid window [15].

We have mentioned that the DQS signal is bi-directional. This means that the signal is driven from all devices connected to the data bus. In Figure 3.1, the DQS signal is not driven for a period before it begins to transition synchronized with data transitions. This signal state is often referred to as tristate or hi-z state. Depending on the signaling technology used, the DQS signal may begin to transition before the first data valid window. This period of the DQS signal burst cycle is called the preamble of the DQS burst. The preamble is used to 'wake-up' the bi-directional bus and allow the receiving chip to recognize when a transition occurs on DQS.

3.3 Read Latency Programming

There several methods in which Read latency can be established for a DRAM. The most common method requires that the memory controller load a predetermined latency value, expressed in clock cycles, into an operating mode register in the DRAM. Design, testing and characterization of the DRAM during manufacturing establish the range of latency values that can be programmed into the latency register. Referring back to Figure 2.4, the buses labeled command and address are used to write the latency value to the DRAM mode register. This command is referred to as the 'load mode register' (LMR) command. We make an assumption that command and address bus timing relative to clock timing is maintained to data sheet specifications so that commands issued from the memory controller are received by the DRAM. Figure 3.6 is a timing diagram showing bus activity for a LMR command. Table 3.1 is an example of binary values used for read latency programming. These values are assigned by two of the address bits latched from the LMR command. Other bits on the address bus are used to program the device for such things as burst length, output drive impedance, write latency, etc.



Read Latency	B1	B0
4	0	0
3	1	1
2	1	0
1	0	1

Table 3.1 Read Latency Settings

Figure 3.6 Load Mode Register Command

3.4 Read Latency Programming Through Channel Tuning

Another possible method for programming read latency involves sending test patterns from the DRAM after a modified read command is issued from the controller to the DRAM. Performing this operation allows the memory controller to detect read latency while at the same time worst-case noise and ISI conditions are induced on the data bus. As previously mentioned, pseudo-random test patterns are easily generated using linear feedback shift registers (LFSR) [21] so that a cyclical redundant code can be generated and detected using low cost circuits. By seeding the LFSR to start the pattern in a predetermined state, the memory controller can detect the transmission of this state after issuing a modified read command. By counting the number of clock cycles between the read command and the start of the test pattern, the controller can determine the cycle based read latency of a read access. After determining the read latency, the controller can then continue to use same pseudo-random pattern to make fine adjustments to center the capture strobe within the center of the data eye. The memory controller can reverse the channel initialization so that the DRAM can program the correct write latency and optimize the DRAM internal data capture point. All of this assumes that the DRAM command channel can support a command protocol to enter a channel training mode after power is applied. Figure 3.7 is an example of a read training pattern.

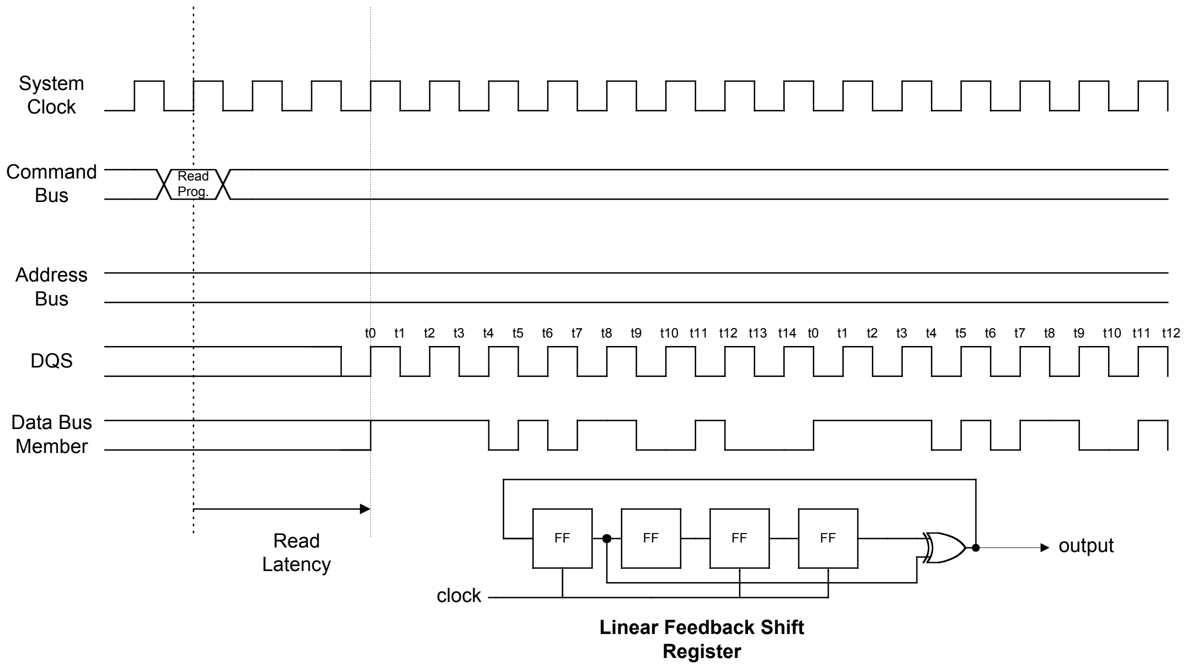

System Clock
Command Bus
Read Prog.
Address Bus
DQS
t0 t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t0 t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12
Data Bus Member
Read Latency
FF
FF
FF
FF
output
clock
Linear Feedback Shift Register

Figure 3.7 Channel Training Read Command and LFSR Circuit

Figure 3.7 illustrates a 15-bit pseudo-random pattern generator and the pattern that is generated from this circuit. LFSR circuits in general generate a maximal length code of (2^n - 1) bits instead of a 2^n bit pattern because an all zeros case in the register will lock the LFSR output low (the same thing happens for the all ones case if we use an xnor gate as the output). Notice some of the properties of the pseudo-random pattern. We see that there is a test of the maximum bandwidth of the data channel during cycles t4-t7. There are also regions in the pattern where a relatively steady state is maintained on the data bus for several bit periods such as t0-t3 and t12-t14. The pseudo-random pattern will maximize ISI [16] on the data channel so that the memory controller can establish optimal capture points for the data. We could also induce SSO and cross talk across the data bus with appropriate shifting of individual members of the data bus while generating the serial test pattern. Minor adjustments can be made to optimize capture points using individual delay lines for each bit on the data bus [11, 15] or by providing an independent DLL delay-line tap selection for each bit across the data bus. If bitwise timing correction is not needed, word-wide adjustments can be made using PLL or DLL circuits [12] so that the capture clock is centered in the data eye at the capture latch in the memory controller.

It is possible to avoid active capture clock adjustment if the source-synchronous interface operates correctly for the signaling technology used. Here, we are concerned with accuracy of interconnect technologies in matching and controlling interconnect delays between the memory controller and DRAM. In this work, we will not go into detail about methods used to capture data. Instead we will be concerned with the DRAM driving read data and using a DLL to temporally align transitions of the DQS signal, data bus and system clock. Read latency and variations in internal timing compensation to maintain correct read latency contributes to variations in the required data throughput from the array to the synchronization circuits. The material presented in this chapter will serve as background information for evaluating the method that is presented in the next chapter for buffering data between the array and the output synchronization circuits.

4. Timing Domains in the Read Data Path

This section introduces the research done for constructing a read data path architecture intended for high-frequency DRAM. We will examine circuit methodologies that will help us maintain data throughput between the array and synchronization circuits despite variations in timing because of programmable read latency, process variations between manufacturing lots and changes in voltage and temperature (PVT). Also of particular interest is pipelining data between the HFF circuits and the synchronous output circuitry. There are three distinct timing domains internal to the DRAM read data path. We will first identify the three timing domains and then begin to examine methods for transferring data between them.

4.1 Array to Synchronous Interface

Moving forward from our discussion of read latency programming, we will now focus on the portion of the data path where the data from the asynchronous array meets the synchronous output circuits. Figure 4.1 shows timing representative of read accesses from the array. Figure 4.2 is a block diagram that illustrates the major circuit blocks in our data path architecture and gives reference for the signals shown in the timing diagram of Figure 4.1. At the top of Figure 4.1 is the command clock domain. The command clock is the external clock delayed by the clock input buffer and any associated buffers and interconnect that route the clock to the command capture latches and the command decoder. Notice in Figure 4.2 that the command clock is also the input clock to the DLL. We try to minimize clock skew between the clocks distributed to the DLL and command decoder.

The command decoder synchronously generates column accesses to the array. All commands entering the command decoder must be decoded immediately following synchronous capture in the command latches. Implementation of the command decoder in this work employed skew tolerant logic clocking methods [17] combined with precharge/evaluate type latches. Using multiphase clocks with phase relationships established for the maximum frequency specification, we are able to avoid any clock period dependency for array accesses while still providing synchronous outputs from the command decoder. We will not discuss specific design issues related to this circuit so as not to be distracted from our focus on the methods developed for maintaining data throughput for fixed read latency at high clock frequencies.

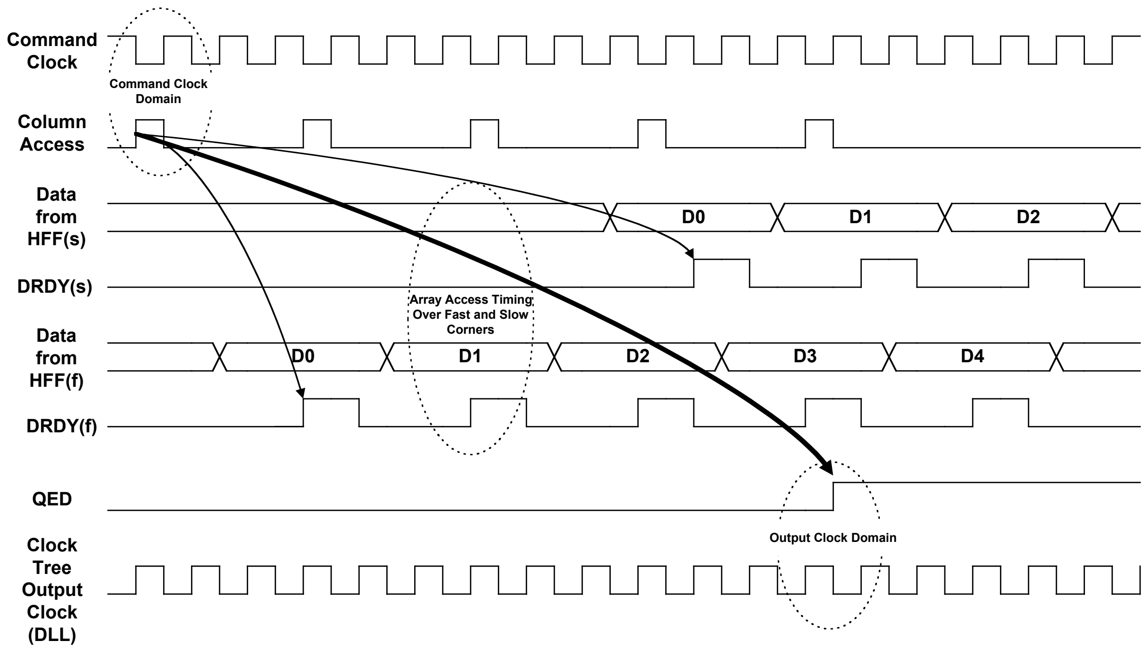


Figure 4.1 DRAM Internal Timing Domains for Read Access

We should clarify what is meant by ‘high frequency’ when we discuss clock frequency for a DRAM. In this work, we define a high frequency clock by comparing the input and output delays internal to the DRAM with the clock period. The input delay is defined as the clock delay through the input clock buffer and interconnect leading to the DLL. The output delay is defined as the data path delay measured from an output clock edge through the data output register and DQ pad driver to the external data pin. The sum of these delays is the same as the IO model delay in the DLL feedback loop (Appendix A). If the total IO delay is equal to the clock period plus synchronization overhead for passing the QED signal between the command clock domain and output clock domain, then we can say we are operating at high frequency and consideration of the methods described in this work become necessary. At very high clock frequencies the IO delay can be multiple clock periods in length making the problem of transferring control and timing information between clock domains even more difficult. In this work we will not examine the output synchronization logic but instead focus on buffering data between the array and synchronization logic.

Referring back to Figure 4.1, the timing of data from the HFF circuits has two important specifications. The first timing specification is the latency of the read access from the array. This is the time it takes for the first set of prefetched data to be passed through the HFF circuits following a read command. The second timing specification is the cycle time of a read access. The required column cycle time is a function of the data prefetch depth, burst length and clock frequency. When designing the data path, we try to make the data bus word width as narrow as possible in order to save area on the die. Therefore, we design the data path to the minimum width based on the minimum burst length and the minimum possible column cycle time. This relationship can be expressed as follows:

$$Col_{\max} = \frac{BL_{\min}}{2 \cdot f_{cm} \cdot CCT_{\min}} \qquad \textbf{(4.1)}$$

$$DW_{\min} = \frac{BL_{\min}}{Col_{\max}} = 2 \cdot f_{cm} \cdot CCT_{\min} \qquad \textbf{(4.2)}$$

where Col_{max} is the maximum number of column cycles per specified burst length, DW_{min} is the minimum data path width required to support the data prefetch depth $BL_{min}$is the minimum burst length, f_{cm} is the maximum clock frequency and CCT_{min} is the minimum possible column cycle time that can be achieved.

Equation 4.1 tells us how many column accesses are possible to service the minimum specified burst length at a minimum clock period. For example, if CCT_{min} = 4ns, f_{cm}= 200MHz and the BL_{min}= 4 then, applying Equation 4.1, Col_{max}= 2.5. This tells us we can complete 2.5 column accesses within the time required for a burst length of 4 at maximum clock frequency. Applying Equation 4.2, we find that Dw_{min}=1.6. We must round this number up to the next highest integer in order find the minimum possible data path width, which is also the data prefetch depth, given the maximum frequency and minimum burst length. If we cannot meet the minimum column cycle time, CCT_{min}, then we must adjust the minimum data path width to accommodate the minimum possible column cycle time. This in turn would increase the data prefetch depth and increase the required area for our data path.

In Figure 4.1, the signal labeled 'Column Access' is used to indicate a column access to the array. If we can cycle the column multiple times within the time required for a minimum burst length at maximum clock frequency, then this signal could potentially cycle more than one time per read command. In our example, the column access signal cycles one time per read command. The command decoder labeled in Figure 4.2 synchronously generates the 'Column Access' signal. This means that each read access prefetches the entire required data depth to accommodate the minimum burst length.

The DRDY signal is used to indicate new data on the data bus. This signal is driven from the HFF circuits and is timed to coincide with data from the HFF circuits. In the architecture of Figure 4.2, the data bus is bi-directional. The DRDY signal must be treated as a data signal since it is driven from more than one location. This means that the DRDY signal is driven from a tri-state driver the same as a data signal. The DRDY signal is 'bundled' with the data on the data bus so that data and DRDY signals have similar transition timing.

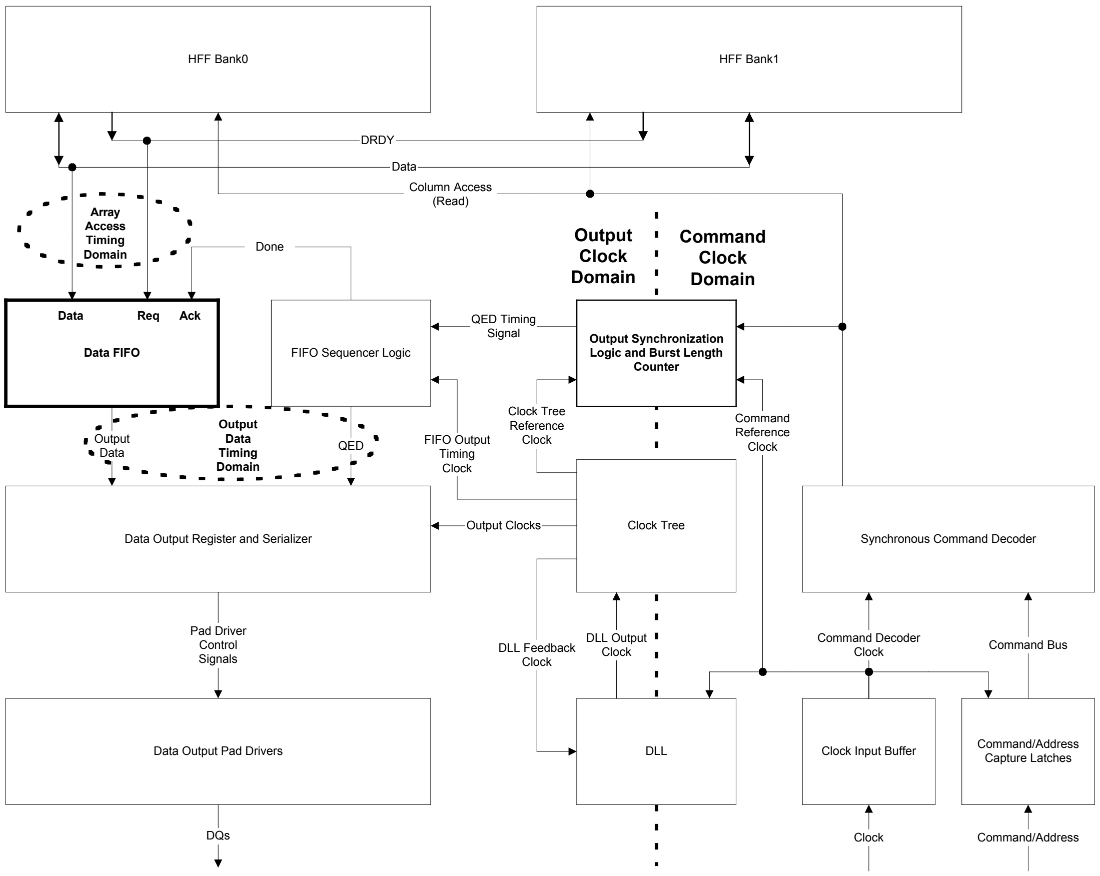


Figure 4.2 Read Data Path Architecture

Data from the HFF circuits must precede the expiration of the internally timed read latency indicated by the QED signal transitioning from low to high. The QED signal is synchronized to the output clock domain and is used to enable the DQ output pad drivers. In Figure 4.1, DRDY(s) shows the timing for data from the array while operating at a low voltage, high temperature corner. DRDY(f) represents timing of data from the array at a high voltage, low temperature corner. At the slow corner, we receive a single access before the QED signal transitions. This is desirable timing at the slow corner because when speed bin testing of memory devices, we want to take advantage of the fastest possible read data latency at a given frequency under worst-case operating conditions. If we were able to cycle the array more than once before synchronizing the output data, then we are not taking full advantage of the array read access time. While operating under fast corner voltage and temperature conditions, more than one array access cycle could occur before synchronizing the output data. In Figure 4.1 there are four accesses before the QED signal transitions as indicated by the pulsing of the DRDY(f) signal.

Process corners can also affect the timing depicted in Figure 4.1. If a memory subsystem has multiple devices from different manufacturing lots operating at different process corners, then the slowest device in the system would determine the overall read latency. We could potentially have some devices completing a single access in the required internal read latency time while other devices complete multiple accesses in the same amount of time similar to the case depicted in Figure 4.1.

Also in Figure 4.1, the QED signal is synchronized to the data output clock generated by the DLL and passed through the clock tree. The timing of the QED signal is dependent on the programmed read latency. Therefore, QED is timed to occur after a fixed number of clock cycles following a read command. Note that the synchronization and sequencing of the QED signal are two independent operations. This is because control for sequencing the QED signal is solely in the output clock domain. However, timing the transition of the QED signal occurs through synchronization between the command clock domain and the output clock domain. The QED signal transferred to the DQ output synchronization circuits serves as a catalyst for synchronizing the data from the array access with the distributed DLL output clock. The block labeled 'Output Synchronization Logic and Burst Length Counter' in Figure 4.2 synchronizes the QED signal between the command clock domain and the output clock domain. In this research, we use timing properties and circuits related to the DLL for properly timing and synchronizing the QED signal to the DQ output register. The variation in timing of the QED signal relative to the DRDY signal has a direct impact on data throughput requirements through the data FIFO labeled in Figure 4.2.

4.2 Data Path FIFO

Returning to the data prefetch architecture, we know that we are required to prefetch either the entire required burst length or multiple fractions of the burst length in order to accommodate the high bandwidth serial data stream at the DRAM DQ outputs. We have also seen that PVT variations can affect the rate at which data is output from the array so that the number of consecutive data accesses that occur before the expiration of the read latency can vary. In order to accommodate the prefetch depth and the varying number of data accesses that occur at various read latencies, we must be able to buffer the data between the array interface and the data output latches. The best way to accomplish this task is by using a first-in-first-out (FIFO) buffer. A common application of a FIFO is to provide constant data throughput between two locations in a data path that might differ in their bandwidth or latency requirements. Referring to Figure 4.2, we see that the data FIFO is used to buffer data between the array timing domain and the output clock timing domain. This timing is separate from the synchronization timing of the QED signal but is not totally unrelated. The timing diagram in Figure 4.1 shows how the timing of the data access from the array is sandwiched between the column access command and the data output timing indicated by the QED signal transitioning synchronously with the data output clock. The timing of the QED signal determines when data is used at the output of the FIFO while the column access timing determines when data is loaded into the inputs of the FIFO.

4.2.1 FIFO Design Choices

FIFOs can be designed in various configurations. One common configuration uses counters that act as pointers to storage locations in the FIFO. A first counter acts as a write pointer and indicates the location for the next data load in the data storage array. A second counter acts as the read pointer and indicates the storage location of the next output data request. Figure 4.3 is a block diagram of such a FIFO. Additional logic must be added to the architecture in Figure 4.3 in order to make certain that the pointer value has settled so as not to write data into storage locations loaded with previously valid write data.

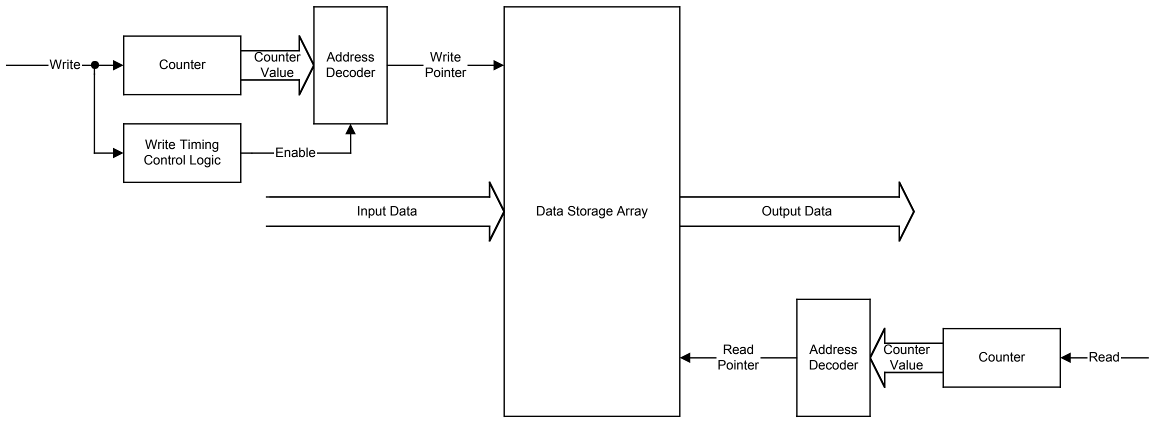


Figure 4.3 Traditional FIFO Architecture

The block labeled ‘Write Timing Control Logic’ in Figure 4.3 is used for timing the valid outputs from the write address decoder. The enable signal routed to the write address decoder is in the decode tree for all of the decoder outputs. This ensures that the address decoder is not transparent as the outputs from the write counter transition. If multiple transitions occur on the counter output, any timing skew between the transitions across the counter value bus will result in transitions through more than one write address in the address decoder. The address decoder enable signal must be timed to allow timing margin to the maximum address decoder delay for the worst-case address transition. Another method to avoid multiple write pointer transitions is to make the counters gray code style counters. In a gray code counter only one signal transitions on each count [18]. Gray code counters require more area than binary counters but offer better write performance since we do not have to consider counter output skew.

Another FIFO architecture [19,20,22] and the one chosen for this work is a self-timed FIFO. This style of FIFO was made popular by the Turing award winning paper *Micropipelines* [19] written by Ivan Sutherland. A block diagram of this type of FIFO is illustrated in Figure 4.4. This FIFO uses speed-independent asynchronous logic for generating control signals. There has been a great deal of research and several papers

[23,24,25] published on methods for improved asynchronous FIFO control logic. Sutherland's work describes a two-phase signaling protocol (also referred to as transition signaling) in which data is transferred between latches using a handshaking communication scheme.

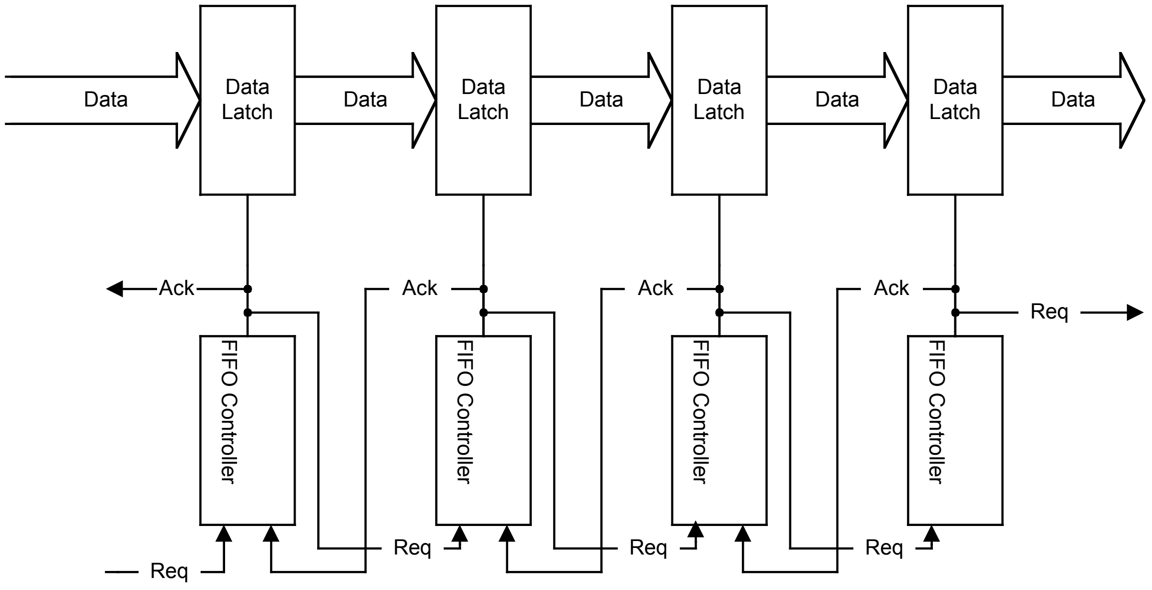


Figure 4.4 Self-timed FIFO

Figure 4.5 is a qualitative look at how the transition signaling protocol is implemented. The transition signaling protocol is not level sensitive but instead communicates with signal transition events. The problems with transition signaling is that the circuits implementing such control tend to consume large area and any logic that is a function of pipeline status requires conversion to level sensitive signaling. In Sutherland's implementation, complex latch circuits are required in order to simplify the latch controllers. If simple transmission gate latches are used, then a conversion from two-phase to four-phase handshaking is required at the latch controller. Figure 4.6 shows Sutherland's implementation of a two-phase latch control circuit that can use simple transmission gate latches. The latch control circuit requires a Muller-C element (to be discussed shortly), an

XOR gate and a toggle circuit [19, 26]. Because the toggle circuit operates on the principle of the transition protocol, this circuit can be prohibitively large for use in deep pipelines.

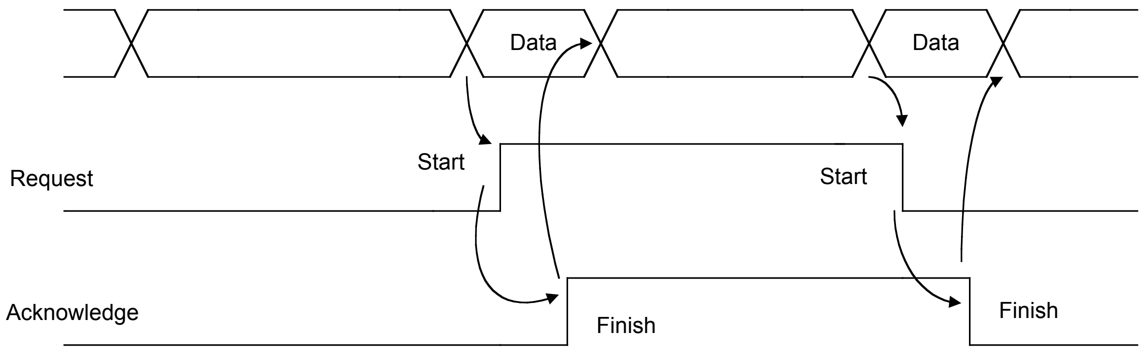


Figure 4.5 Two-phase Handshake Protocol

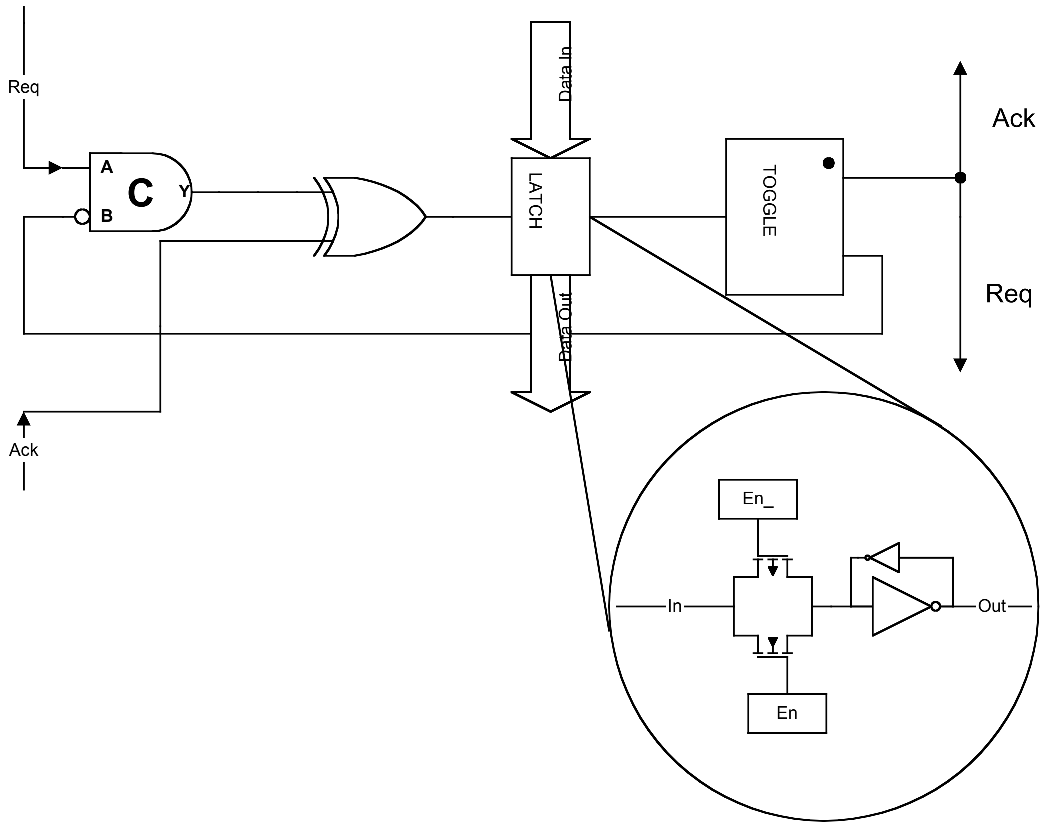


Figure 4.6 Two-phase Latch Controller for Simple Transmission Gate Latch

The four-phase handshaking protocol (also referred to as return-to-zero signaling) is an alternative to the two-phase protocol. Figure 4.7 is an example of the sequence of this protocol. When the request signal on the input of a controller transitions from low to high, data is captured in the latch. The controller then transitions the acknowledge signal back to the previous stage to indicate receipt of the data in the latch. The previous FIFO controller then removes the valid request signal after receiving the acknowledge signal. When the request signal to the controller returns to zero, the controller then resets the acknowledge signal to zero. This describes how a four-phase controller arbitrates a request from the previous controller in a FIFO. The request *input* (Reqin) and acknowledge *output* (Ackin, this name indicates an acknowledgment of input data) at the controller arbitrate data that is input to the controller latches.

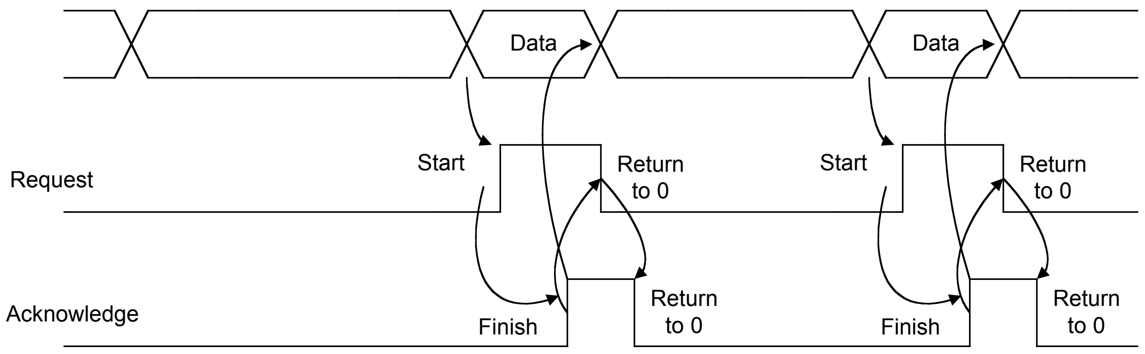


Figure 4.7 Four-phase Signaling Protocol

This same protocol is followed on the output of the controller. After the controller captures data in the latches, the controller then sends a request signal to the next controller in the FIFO. When the next controller captures the data, an acknowledge signal is sent back to the controller indicating the data has been received. After receiving the acknowledgement of the output data, the controller resets the request output and is then prepared for future data transfers from the previous controller in the FIFO. The request *output* (Reqout) and the acknowledge *input* (Ackout, this name indicates an acknowledgment of output data) at the controller arbitrate the data that is output from the controller latches. Figure 4.8 further illustrates the connections necessary for the four-phase signaling protocol.

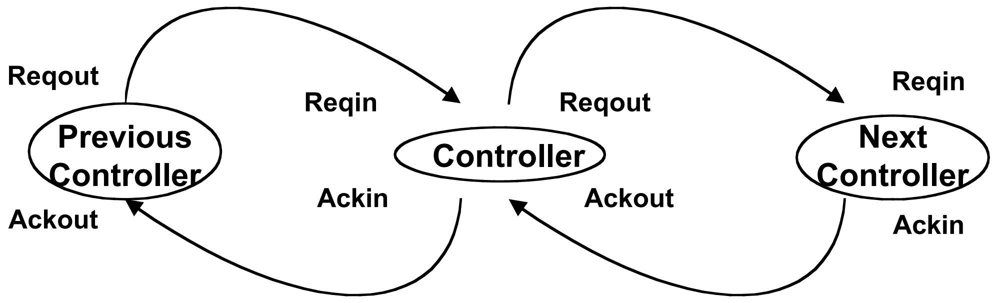


Figure 4.8 Controller Interface for Four-phase Signaling Protocol

There has been work done in the area of four-phase latch controllers that use edge triggered storage registers [26]. Using transparent latches as shown in Figure 4.6 cuts out the master-slave combination and eliminates half the required transistors [27].

4.3 Latch Controller Design

Now that we have established the signaling protocols for self-timed FIFOs, we will now look at controller designs for the four-phase protocol. There are several reasons for deciding to use the four-phase protocol in this application. The first requirement is that our latch controllers have a small footprint. Two-phase latch controllers tend to use more area because they do not have the advantage of using logic levels to indicate status of the controller inputs and outputs. Instead, in the case of transition signaling, the control logic must store previous states in order to ascertain the current state upon the event of a transition [19]. This is a conversion from the two-phase protocol to four-phase protocol. By using return-to-zero signaling, the logic levels automatically indicate the status of the inputs and outputs of the controller.

Another, not so obvious requirement is that the input FIFO controller interfaces to the asynchronous signal DRDY that does not adhere to the request/acknowledge communication loop. Referring back to Figure 4.2, we see that the DRDY signal is generated similarly as a data signal on the read/write data bus from the HFF banks. Because the DRDY signal is generated from multiple locations, there is no simple way for the logic level of the signal to be indicated between all of the signal sources. Therefore, a transition signaling methodology does not fit well with the current architecture. Also, because the HFF banks only store data momentarily between array accesses, the DRDY signal must be bundled with the data and complete a transition cycle within the time of an array access cycle as shown in the timing diagram of Figure 4.1. Therefore, the DRDY signal resembles a Reqin signal to the first controller of the FIFO. The circuits used to time the DRDY signal must be designed to cycle in accordance with the four-phase signaling interface. This means that the width of the DRDY pulse must be long enough so that it is not removed before the acknowledge from the first controller is generated. This causes the open-loop DRDY signal to appear as a closed-loop signal to the first controller in the FIFO. We will revisit the environment of the FIFO after we discuss the design of the latch controllers for our current FIFO design.

4.3.1 Four-phase Latch Controllers

In Figure 4.6, there is a symbol that resembles an and gate with a "C" imprinted on it. This gate is known as a Muller-C gate or a "concurrency element." This gate is very important for modern asynchronous design, particularly where parallel processing is involved [28]; thus, the name concurrency element. The C-element is a two state circuit that can have multiple inputs. In the case shown in Figure 4.6, there are two inputs A and B. Ignoring the inversion bubble on input B, when A and B are in the same state, the output, Y, assumes that state. For example, if A=B=0 then Y=0. Table 4.1 is a truth table showing the operation of the C-element.

A	B	Y
0	0	0
0	1	Yp

1	0	Yp
1	1	1

Table 4.1 C-element Truth Table

We can extend the concept of the C-element to multiple inputs. In that case we would wait until all the inputs to the C-element were at the same value before the output would change to reflect that value. As shown in Figure 4.6, the C-element can have inversions on the inputs so that the output would assume the value of the true inputs. For example, the C-element in Figure 4.6 would assume the value of A whenever $A \neq B$.

There are several options for circuit design of the C-element [19,22]. We can use a dynamic or static design for the C-element circuit. The truth table of Table 4.1 indicates that whatever the circuit style chosen, the circuit must have a way of storing the previous output values. Figure 4.9 shows a dynamic C-element and a static version using a weak feedback inverter to provide current for storing the previous value in the input capacitance of the output inverter. The dynamic style has performance advantages over the static style because in the static version, the input transistors must be sized such that the pull-up or pull-down devices can overcome the current drive from the weak feedback inverter. Therefore, we want to size the weak feedback inverter such that the devices in the feedback inverter provide the minimum current necessary to maintain the value on the storage node to overcome leakage current through the input devices. By keeping the feedback inverter small, we can then make the input devices smaller presenting lower capacitive loading to any circuits driving the devices. Of course, sizing of the input devices must also consider the load of the output inverter and how large the output inverter must be made to drive its output load.

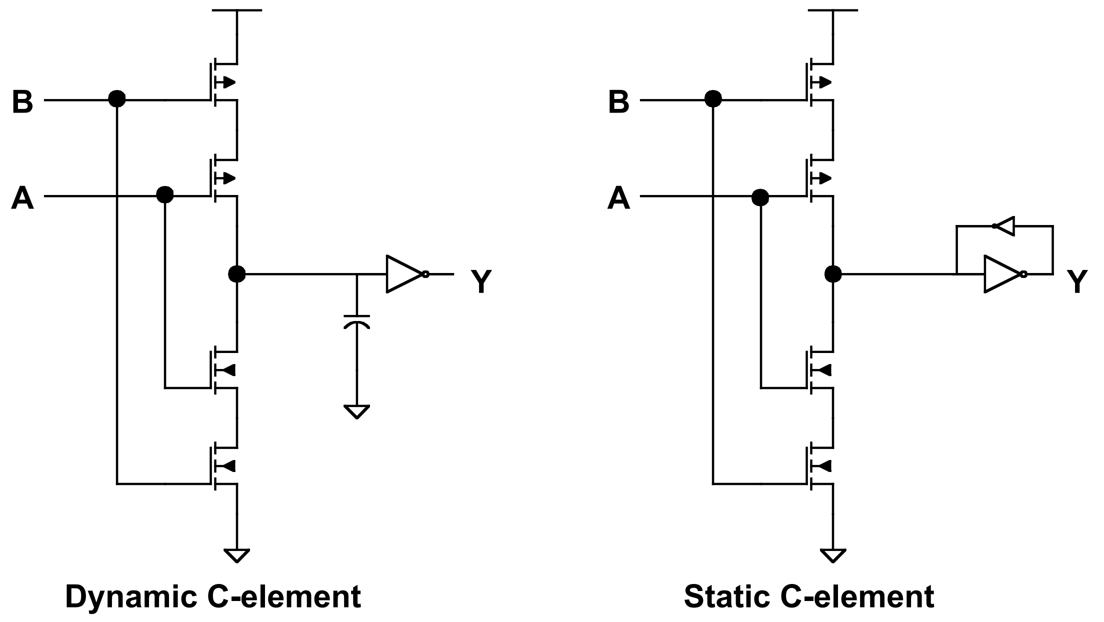


Figure 4.9 C-element Circuit Designs

4.3.2 Simplified Four-phase Latch Controller

Using a C-element, we can form a simple four-phase latch controller [27] as shown in Figure 4.10. The simplicity of this type of controller does provide an area advantage over other implementations. But, as stated in [27], there are disadvantages to using this type of controller. First, there are timing assumptions on how this latch controller must operate. The buffer is necessary because we assume that there are several latches that must be driven by the Lt signal. Because of this, the Ackin signal is naturally delayed back to the previous controller to help ensure that the previous latch controller does not open its latches before the present latch controller latches the new data. Notice, however, that the Reqout signal is driven before the buffer. We can drive the Reqout signal early because the latch does not have to be closed for the next controller to begin sampling the data as long as the data has adequately passed through the present set of latches. Therefore, the C-element delay must not be shorter than the data-in to data-out delay of the latches so that the next controller does not latch invalid data.

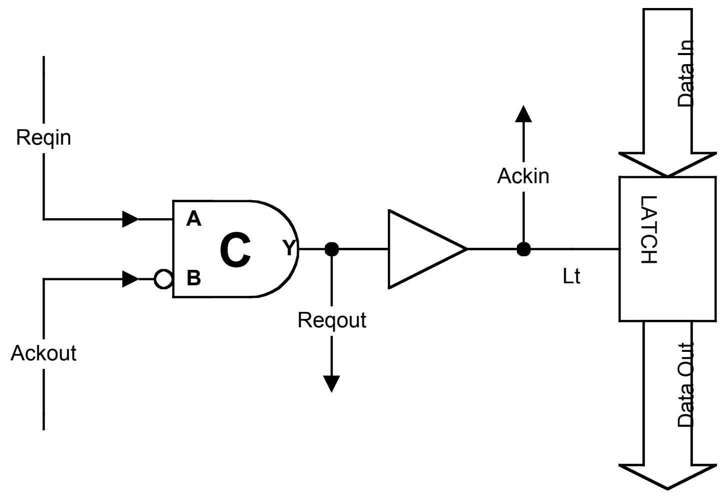


Figure 4.10 Simple Four-phase Latch Control Circuit

Another problem with using the simplified latch controller can be illustrated with a simple Signal Transition Graph (STG) [22,29]. STGs are a subset of Petri Nets [30] and can be used to represent the behavior of an asynchronous control circuit by describing the causality among events. An STG must have particular properties that make all of the possible markings reachable and also prevent the graph from becoming "dead-locked" which means a marking is reached that prevents further sequencing of the graph. We will not delve into the theoretical details behind using STGs [31] for circuit synthesis. Instead we will look at STGs as an alternative to timing diagrams that will illustrate the sequencing of the circuits designed in this work.

Figure 4.11 is a STG describing the operation of the simplified four-phase latch controller of Figure 4.10. The two dots on the STG indicate the *initial marking* of the graph. This is to indicate the starting point of the transition sequencing. In this case, the Reqin+ *place* is fully enabled because the place only has a single input *arc* to which a *token* must be passed or initially marked. When all of the input arcs to a place are marked the transition indicated at the place is said to be enabled and can then *fire*. In our

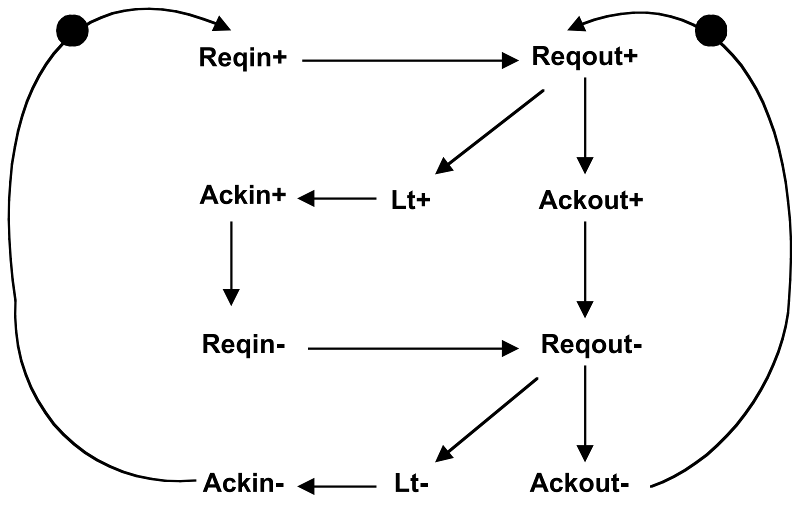


Figure 4.11 STG of Simplified Four-phase Controller

example, the Reqin transition is enabled and transitions high(+). A token is indicated by the dot. Since Reqin+ is enabled, we allow that signal to fire therefore passing tokens to all output arcs of the Reqin+ signal. In this case there is only one output from Reqin+ so a token is passed to the output transition of the Reqin+ signal which is the second input arc required to enable the Reqout+ signal. Because all of the input arcs of the Reqout+ place now have a token, the Reqout+ signal is now fully enabled and can now fire to sequence tokens to all of the its output transitions. After the Reqout+ signal fires, the Lt+ transition and the Ackout+ transition become enabled. After the sequence described above occurs, we will have the marking indicated in Figure 4.12. The Lt+ transition is an output from the circuit description and will fire dependent upon internal circuit delays; however, the Ackout+ transition is a circuit input and will fire dependent upon environmental delays. This is an example of how the STG models concurrency. As long as the required properties of the STG are present for circuit synthesis [31], the STG can be synthesized into a state graph that takes into account all of the possible firing orders of the events [22]. Because even simple STGs can explode into a very large state space, software tools become necessary for synthesizing the logic that implements the state graph description. One such tool called FORCAGE is used in [27] and another popular tool called Petrify [31] is also ideal for STG synthesis applications.

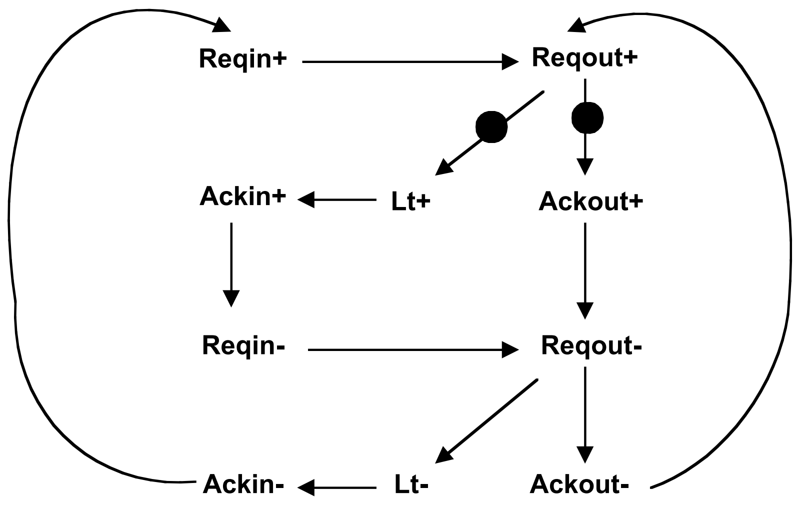


Figure 4.12 STG Marking After Reqin+ and Reqout+ Firings

Looking at the STG describing the simplified four-phase latch controller, we see an undesirable property of this architecture. When we form several such controllers into a FIFO, as illustrated in Figure 4.4, we see that at most only alternating stages can hold new data at any time. This is because Ackout- must transition (and therefore the next latch become empty) before Lt can go high (the present latch becomes full) [27]. This would mean that two adjacent latch controllers are holding the same data. Now the area advantage of the latch controller circuit using the simple TG latch is not realized because the FIFO would have to have a pair of controllers and latches for each storage location to achieve a specific FIFO depth.

4.3.3 Semi-decoupled Four-phase Latch Controller

To solve the problem of not being able to fill all of the stages of the FIFO we need to decouple the input from the output of the latch [27]. We quote copiously from the Furber, et al. paper [27] because Furber's work presents a worthy solution for development of a controller that decouples the input and output of the latch controller. The method used by Furber is to add an internal variable (A) to the simplified four-phase controller STG as shown in Figure 4.13. This variable is used to indicate when the input side of the latch controller is ready to proceed independent of the output side. This allows the latch to close before the next latch is empty because Lt+ becomes concurrent with Ackout-. We can see this concurrency if we consider that data has been transferred to the next set of latches on a previous cycle and Ackout+ remains high because downstream latches are full. Even though the next set of latches is full, the input side of the current latch controller can cycle to latch new data with Lt+ firing as shown with the STG marking sequence in Figure 4.14. Figure 4.14 shows how successive latch controllers can hold unique data in their respective latches. The controller will proceed with transfer of new data after Ackout- occurs.

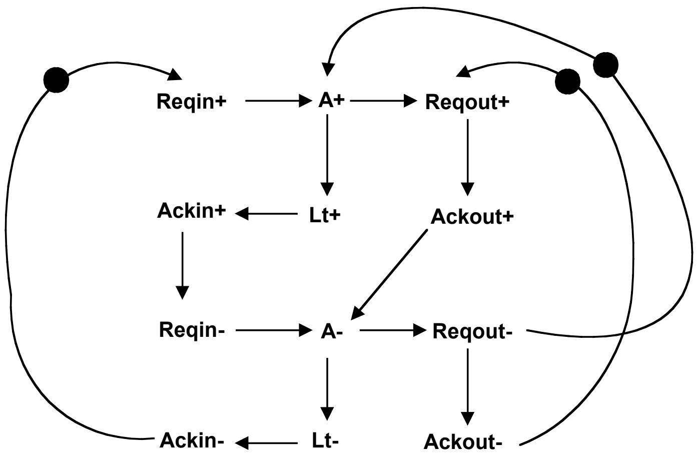


Figure 4.13 Initial Marking of Semi-decoupled Four-phase STG

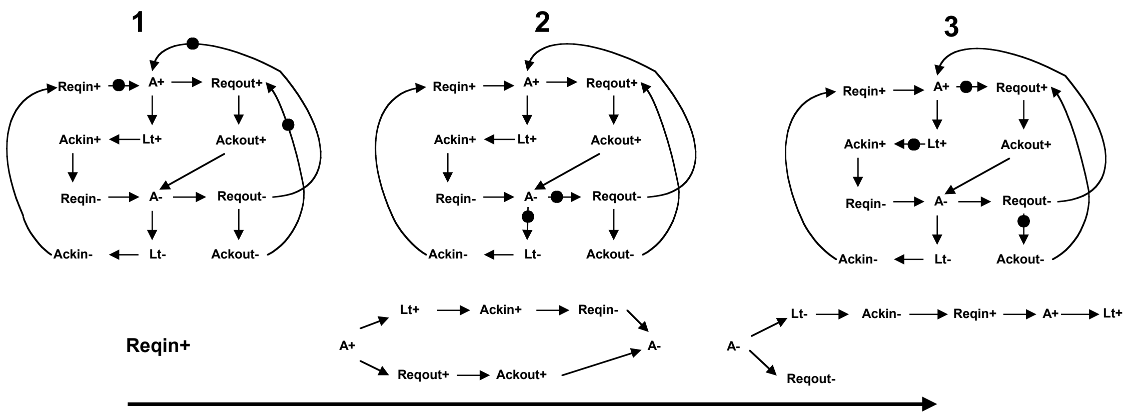

1
2
3
Reqin+
A+
Reqout+
Ackin+
Lt+
Ackout+
Reqin-
A-
Reqout-
Ackin-
Lt-
Ackout-
Reqin+
A+ → Lt+ → Ackin+ → Reqin- → A-
A+ → Reqout+ → Ackout+ → A-
A- → Lt- → Ackin- → Reqin+ → A+ → Lt+
A- → Reqout-

Figure 4.14 STG Marking Sequence Illustrating Lt+ and Ackout- Concurrency

We will not fully pursue the synthesis of the STG in Figure 4.13 for the semi-decoupled latch controller. A state graph is formed from the STG description with the logic to implement the state sequencing derived using binary logic reduction methods [22]. Furber outlines these methods and uses the synthesis tool FORCAGE [27]. The circuit that is derived can be constructed from a modified version of the C-element called the asymmetric Muller C-element.

4.3.4 Asymmetric C-elements

Asymmetric C-elements are C-elements in which some inputs control only one of the output trajectories. An example of an asymmetric C-element is shown in Figure 4.15. The notation used indicates that an input connected to the body of the gate controls both output trajectories, such as input B in Figure 4.15. An input controls only the rising edge of the output when connected to the extension labeled “+,” and the falling edge when connected to the extension labeled “-.” The control circuit derived through syntheses of the STG in Figure 4.13 is constructed of asymmetric C-elements and shown in Figure 4.16 [27].

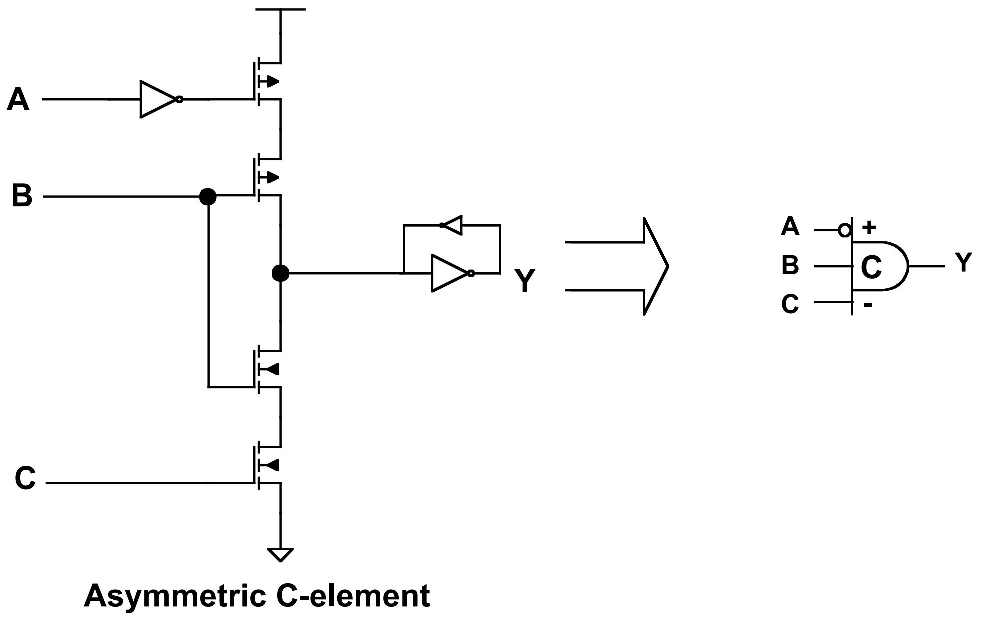


Figure 4.15 Asymmetric C-element Symbolic Notation

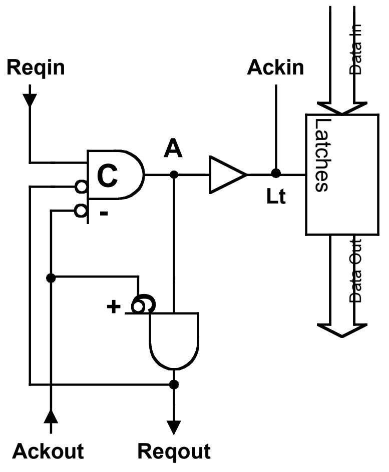


Figure 4.16 Semi-decoupled Latch Control Circuit

4.4 Final Data Path FIFO Design

Now that we have established the design and operation of the semi-decoupled latch controller, we will use this controller design in the design of our data path FIFO. One reason to choose the semi-decoupled controller is that the performance/area ratio is better than the fully decoupled four-phase controller also introduced in Furber's paper [22]. We have already noted the problems with the simplified four-phase latch controller as far as the ability to fill all of the stages of the FIFO. The simplicity of the semi-decoupled controller design combined with the capability to use transmission gate style latches fits well with our goals of a high-performance, small area data FIFO.

In this section, we will revisit the area of the data path design in which the FIFO is used and examine the timing environment where the FIFO buffers data between the array access and the data output register/serializer. Referring to Figure 4.17, we see a recreation of part of the data path block diagram from Figure 4.2. Figure 4.17 illustrates the 8-bit data path slices with 4-bit data prefetch architecture developed for this work. We see that data is supplied from the FIFO output to the inputs of the data output register/serializer. Outputs from the register/serializer are used to control the pull-up and pull-down transistors for the DQ pad driver.

As a review, we are buffering data that is accessed from the array at various PVT dependent latencies. The FIFO is used to bridge the timing differences between the array accesses, driven by the command clock timing domain, and the QED valid time determined by the programmed read latency and sequenced by the data output clock driven from the DLL (refer back to the timing diagram in Figure 4.1).

The data FIFO is designed so that it is deep enough to hold all possible column accesses that can occur within the longest read latency that can be programmed. In other words, a column access causes a FIFO load indicated by the DRDY signal. The completion of the read latency cause the signals Donea_ and Doneb_ to transition indicating that the data

has been consumed at the output serialization circuits. When designing the array data path, we must perform careful characterization of column access
performance using SPICE simulation results. After characterizing the column access timing, we must consider the longest read latency that the device can be programmed according to the DRAM specification. The depth of the FIFO must allow all data accesses to be stored within the column access cycle time from the HFF block. This requires that when the FIFO is full, the Donea_,b_ signals are toggled before any successive DRDY signals strobe.

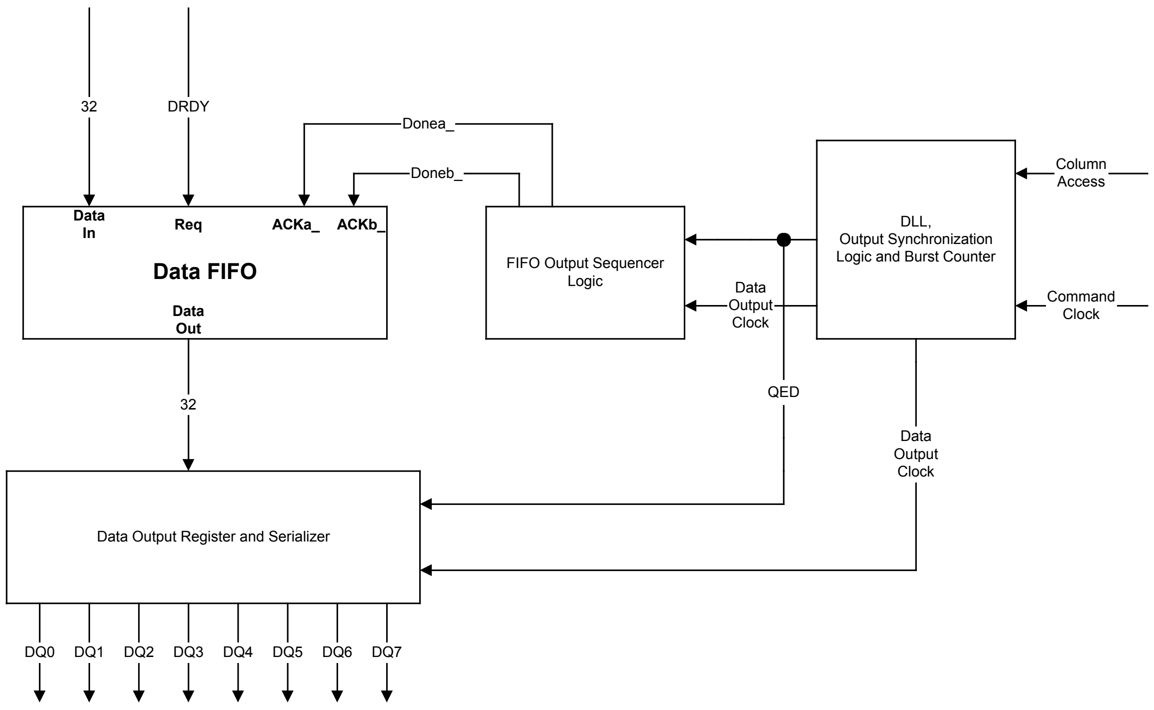


Figure 4.17 Data Path Synchronization Architecture

In the next chapter, we will examine in detail the operation of the self-timed FIFO shown in Figure 4.17. This FIFO is designed to provide correctly timed data at the input of the data output serializer. The data output serializer is the point in the data path that operates at the full data output clock frequency. The serializer performs two primary functions. First, the serializer converts the prefetch data into a single bit stream. Second, the serializer converts the bit states into control signals for separately toggling the pull-up and pull-down structures in the DQ pad driver. We will not go into detail on the operation of the data output register/serializer. Instead, we will focus on how the FIFO architecture is designed to buffer data between the array and the serialization circuit.

5. FIFO Architecture

In this chapter, we will examine the final FIFO design and discuss the requirements that determine the overall FIFO architecture. Continuing our example data path, the FIFO is constructed to supply 8 DQ pad drivers with 4-bit prefetch data. The input of the FIFO must accept 4 bits for each DQ serializer combined with the bundled DRDY signal, while the output of the FIFO must be synchronously timed to supply new data to the data output latch/serializer. Figure 5.1 is a top-level block diagram of the FIFO architecture used to buffer data between 8 DQ pad drivers and a 4-bit prefetch array architecture. Details of the FIFO operation about to be described would change for variations on prefetch depth and serializer function.

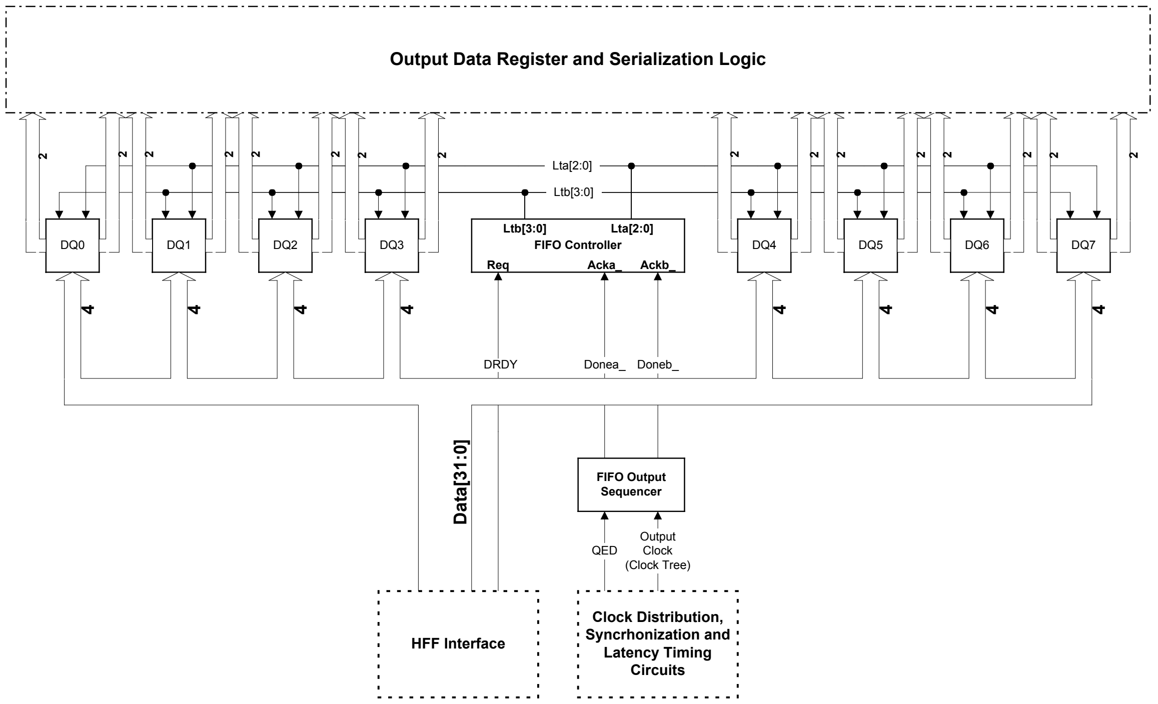


Figure 5.1 Top Level FIFO Architecture

We will first examine the application of the semi-decoupled latch controllers used in the FIFO Controller block. We will then examine the function of the FIFO Output Sequencer circuit and how this circuit sequences the output data from the FIFO to the serialization circuits. Lastly, we will investigate performance metrics for the overall FIFO design.

5.1 FIFO Controller

The FIFO Controller circuit employs the semi-decoupled latch controllers discussed in Chapter 4. One advantage of using these controllers is the area saved by being able to use simple transmission gate latches for the latching mechanism. Another important advantage is that the input req/ack interface is decoupled from the output req/ack interface so that the Ackout- and Lt+ transitions are concurrent. This means that a latch controller can latch new data before the next controller has passed data to a forward set of latches; thus, allowing all latch stages in a FIFO to hold unique data (see Figure 4.21).

The FIFO Controller circuit is constructed as two parallel asynchronous FIFOs. Figure 5.2 is a schematic of the final FIFO design. There are some unusual details to this design that require some explanation. In each latch controller path there appears to be an extra controller. The first controller in each path does not have the Lt signal routed out of the circuit to control a set of latches. The reason for the extra controller is to allow the tri-state data bus to serve as an implied latch. We cannot, however, control the data bus latch latching mechanism because the timing of the data bus is determined by the rate at which read accesses are applied to the device and the timing of the HFF circuits in driving the bus. This timing is entirely open-loop and determined by PVT as was discussed in Chapter 4. The first latch controller in each path represents the status of data in the HFF latches. If the data has not yet been transferred from the HFF circuits to the FIFO, the Reqout signal from the first latch controller will be high.

Another unusual aspect of this design is that both latch controller paths have the same Reqin signal (DRDY) but separate Ackout signals (Donea_, Doneb_). This is because, as will become clearer in the discussion to follow, the timing of data input to each latch controller path is the same while the data output of each latch controller path differs. One problem with using the asynchronous FIFO in this application is that the input and output timing is not tied to the four-phase signaling protocol. We are instead using the four-phase protocol for transferring data between latch stages internal to the FIFO but allowing an open-loop timing relationship at the input and output stages of the FIFO. This will require that we characterize the latch controllers and the memory array accesses to provide the maximum required throughput to fully buffer data from the array (HFF) to the serializers.

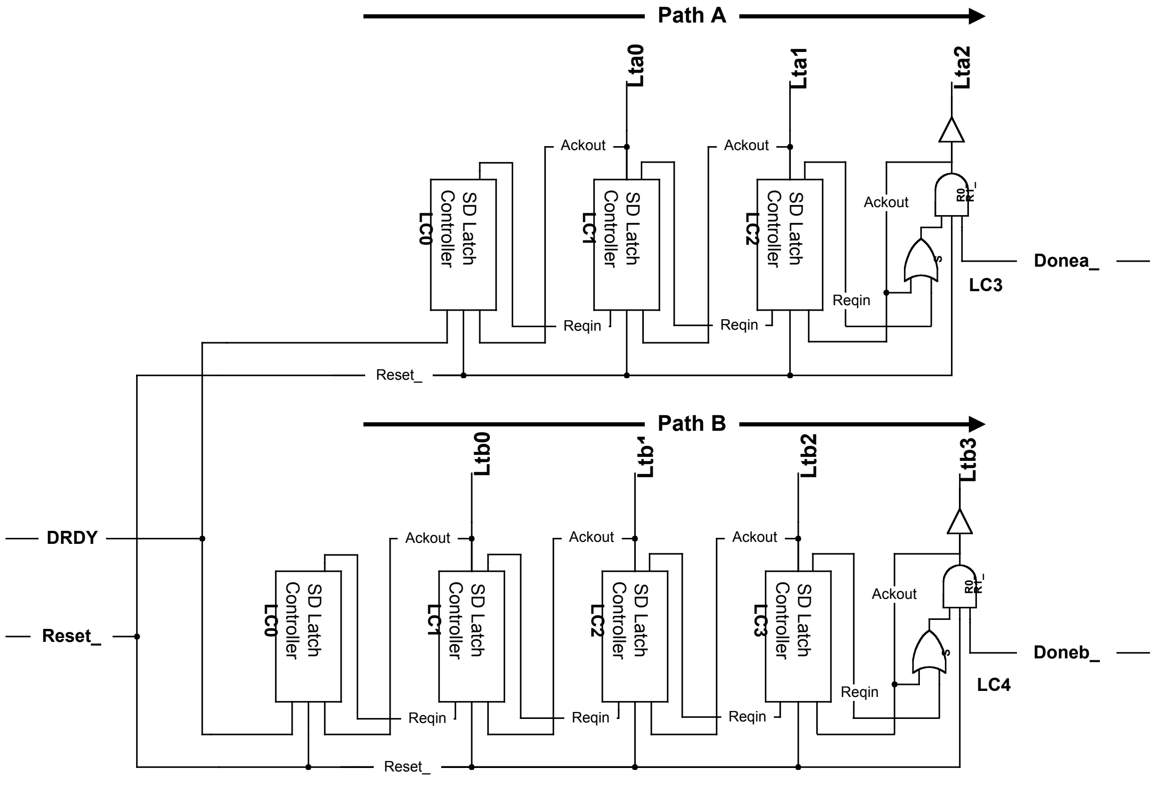


Figure 5.2 FIFO Controller

The last point of interest in Figure 5.2 is the final latch controller in each path. This latch controller is a set/reset latch with the reset signal able to override the set signal. Table 5.1 is a logic table indicating the state transitions of the last latch controller. Because the output stage of each latch controller path does not require a full four-phase interface, a less complex piece of logic can be used to accomplish communication with the last four-phase controller. Examining path A in Figure 5.2, when the full four-phase latch controller, LC2, issues a Reqout+ signal, the output of LC3 goes high closing the latches in the last stage. This signal, Lt2 of LC3, is used as the Ackout input to LC2 and indicates that data is latched in the last stage. The Reqout- signal transitions from LC2 and is applied to input S of LC3 but LC3 maintains a high state on the output, Lt2, in accordance with Table 5.1. Next, when LC3 receives a Donea_ transition on its R1_ input, it toggles Ackout- to LC2 and LC2 is then able to transition Reqout+ in accordance with the STG for the semi-decoupled latch controller shown in Figure 4.21.

R0_	R1_	S	OUT
0	X	X	0
X	0	X	0
1	1	1	1
1	1	0	OUT_{N-1}

Table 5.1 Output Latch Controller Truth Table

The final piece to the FIFO is the set of latches used in each DQ latch circuit. The DQ latch circuit is indicated in Figure 5.1 as a separate set of latches for each DQ data path leading to the serializer circuit. The FIFO controller supplies a set of latch control signals, Lt[x], that close the transmission gate latch when the corresponding signal is '1' and opens the latches when the corresponding Lt signal is '0.' Each set of latches are composed of two serially connected sets of latches representing the control paths A and B of the FIFO Controller. Each serially connected set of latches is further composed of two parallel sets of latches. Taken together, this means that the input stage and output stage of the latch set for each DQ is composed of 4 bits. In Figure 5.1, this configuration is represented as a 4-bit input data path and two 2-bit output data paths. Figure 5.3 illustrates a set of DQ latches.

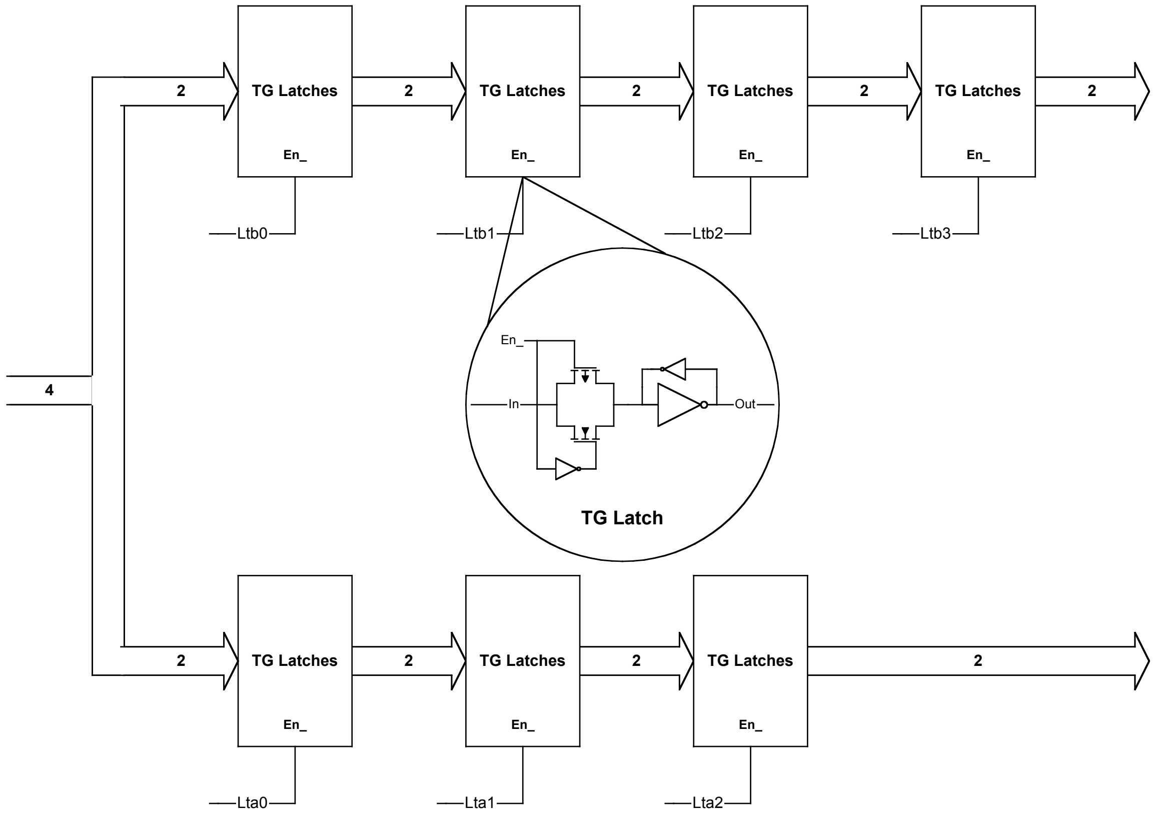


Figure 5.3 DQ Latch Set

5.2 FIFO Output Sequencer

The FIFO Output Sequencer indicated in Figure 5.1 is used to properly time the output data from the FIFO to the serializer circuits. In Figure 5.3, we see that the latch circuits at each DQ are configured to accept 4 bits of data and output data as two sets of 2 bits. Furthermore, the serialization circuit is designed to convert the output of the latch circuit to a single bit of data driven on each edge of the output clock providing a Double-data rate (DDR) output. In order to maintain continuous data for the 4-to-1 serialization, we must be able to change to new data at the input to the serializer without interrupting the data flow out of the serializer. This is accomplished by alternating between the two sets of 2-bit data from the output of the latch circuit. While one set of data is actively converted to a serial data stream, the other set of data is changed following the issuance of a Done(x)_ signal. In a

DDR device, using the configuration outlined above, the data must change on the inputs to the serializer within a single clock cycle. By controlling the Donea_ and Doneb_ signals at the final FIFO Controller stage (Figure 5.2) we alternate between the two output paths so that new data is always available to the serializer. The timing diagram and block diagram of Figure 5.4 illustrates this case.

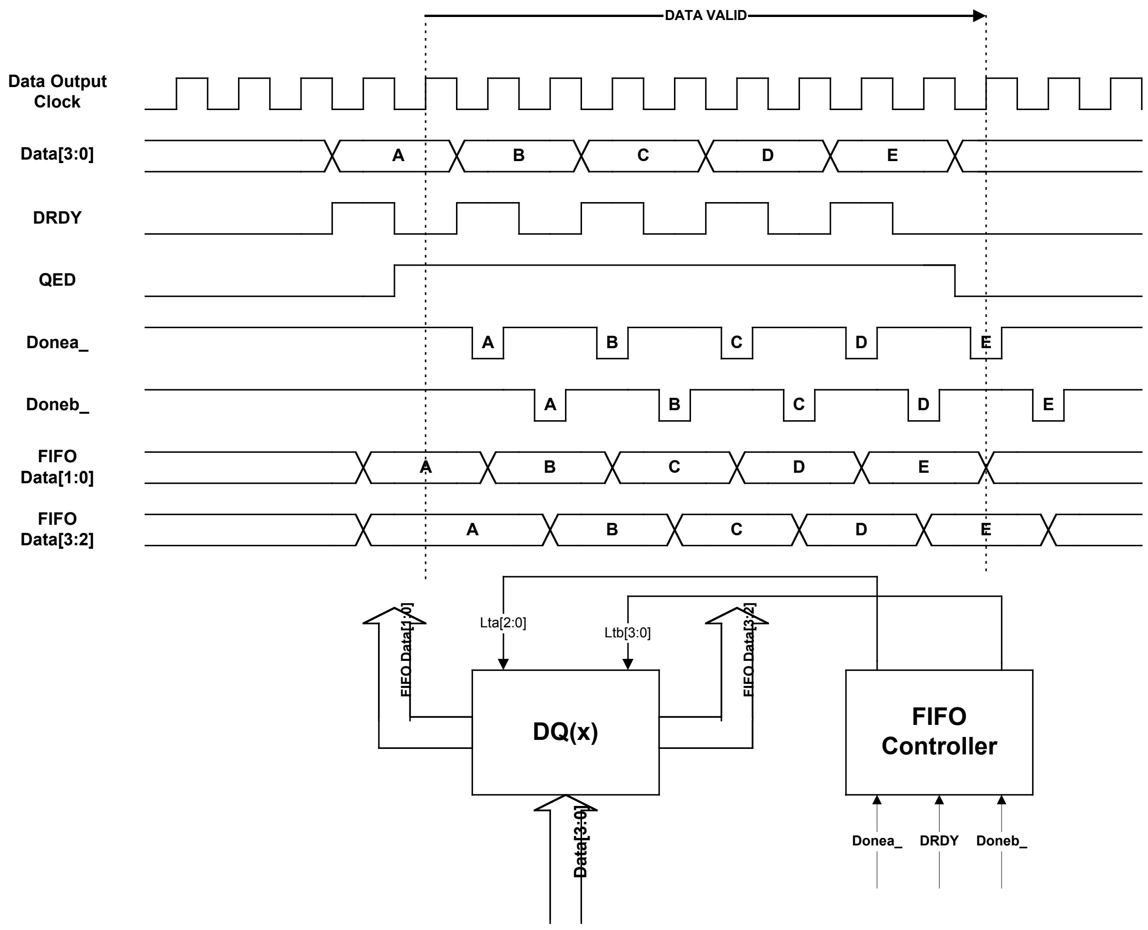


Figure 5.4 FIFO Timing Diagram for a Single DQ

In Figure 5.4, we see that data access A is loaded into the FIFO when DRDY is true. This will load the lower two bits into path A and the upper two bits into path B in Figure 5.3. The FIFO Output Sequencer begins timing the Done(x)_ signals once the QED signal is synchronously captured in the data serializer. Recall, that the QED signal is used to enable the output driver. The QED signal also properly times the read latency out of the device after it is passed through the synchronization circuit indicated in Figure 4.24. As long as the QED signal is valid, the output drivers will be enabled and any data present at the output of the data FIFO will be output from the data serialization circuits. We must synchronously chang the data out of the FIFO with the data serializer if we are to properly time consecutive read data serialization. We see in Figure 5.4 that the QED signal serves the dual purpose of synchronously timing the Done(x)_ signals to the FIFO with the data output serialization and synchronously enabling the DQ pad drivers. The QED signal is the common timing mechanism for timing the serialization of data from the pad driver with the synchronization of data from the output of the FIFO.

There are many options for the design of the FIFO Output Sequencer. Because the FIFO Output Sequencer is a synchronous circuit, the design is greatly simplified compared to the asynchronous design methods introduced in Chapter 4. A written description of the FIFO Output Sequencer is as follows:

The QED signal is synchronously captured on the rising edge of the clock inside the FIFO Output Sequencer. Following the initial capture of QED, the Donea_ signal is pulsed low after the next falling edge of the clock indicating that the first two data bits have been output from the data serializer. The data from the outputs of path A are changed to the data from the next array access (access B in Figure 5.4). One cycle later, on the second falling clock edge relative to the initial capture of QED, Doneb_ is pulsed low to indicate that the last two bits from array access A have been output from the data serializer. The upper two bits of data from array access B are then output from path B in the FIFO providing new data for the output data serializer. This process continues until the QED signal is invalid.

Figure 5.5 is an example of logic that would accomplish the written description of the FIFO Output Sequencer. Working through the logic of Figure 5.5, one can see that the timing of the Done(x)_ signals shown in Figure 5.4 is achieved. This logic was derived heuristically through a thorough understanding of the written description. Notice that the Done(x)_ signals are generated for only a half clock cycle. This is done to allow the fastest possible cycle time through the FIFO. Again referring to Table 5.1, we see that as long as the reset override signal, R1_ is '0,' then Ackout to the previous semi-decoupled latch controller (LC2 in Figure 5.2) is also held low. Referring back to the STG of Figure 4.21, we see that Reqout from LC2 cannot transition to a '0' until the Ackout signal transitions to a '1.' Therefore, the Done(x)_ signals have a minimum requirement that they remain low only long enough to ensure that the Ackout- signal is valid at the preceding latch controller, LC2, to achieve minimum cycle time.

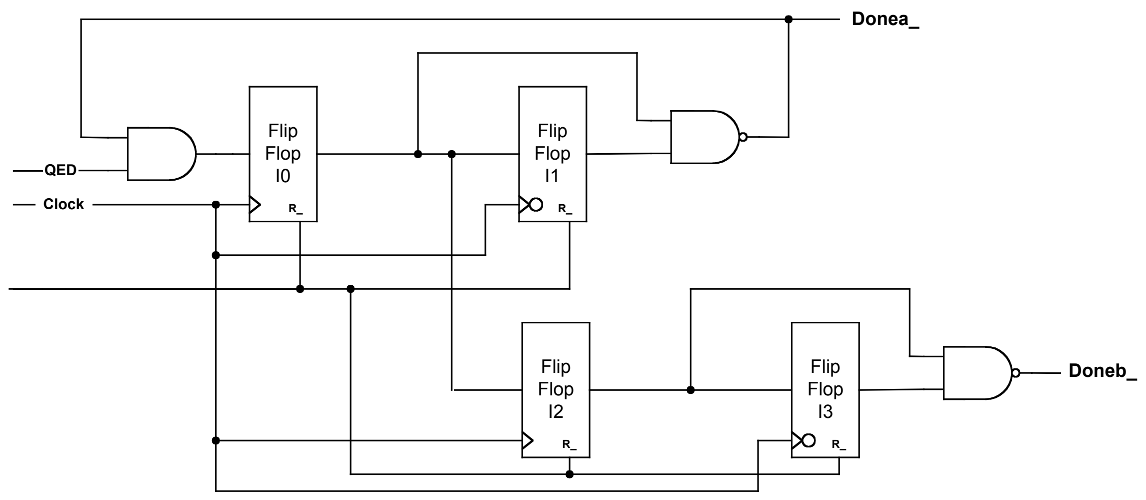


Figure 5.5 FIFO Output Sequencer

Figure 5.5 is only one example of logic that would accomplish the task of sequencing the data from the FIFO to the serialization circuits. The logic style in Figure 5.5 will work as long as the clock period is long enough to allow the transfer of signals between the flip-flops and logic within ½ of a clock cycle. The worst-case delay is between the output of flip-flop I1 through the logic and back to flip-flop I0. Other logic styles can be employed including multi-phase clocked logic [17] combined with precharge-evaluate logic gates.

5.3 FIFO Performance Analysis

In Chapter 4 we outlined the DRAM internal timing paths following a read command at the DRAM inputs. There are essentially two separate timing paths that we are concerned with. The first timing path is the delay in accessing the array column to prefetch data that is eventually loaded into the FIFO; this being the timing path that generates the DRDY signal. The second timing path is the synchronization logic path that generates the QED signal. The QED signal is used to enable the DRAM output drivers and also time the Done(x)_ signals that indicate the consumption of output data from the FIFO. It is important to note that both timing paths, although separate and independent, are generated from the same event; that being the read command issued at the input to the DRAM.

The maximum number of array accesses that can occur at the fastest operating corner and with the DRAM programmed to the maximum read latency specification determines the maximum required FIFO depth. As we saw in Chapter 4 (Figure 4.1), the operating corner of the DRAM determines the number of consecutive accesses that can occur within a given programmed read latency. In other words, the operating corner and programmed read latency value determines how many DRDY transitions can occur before the first Donea_ signal occurs. A first approximation of the number of pipeline stages required for a FIFO is derived as follows:

$$Pipestages \approx \frac{tDone\,\mathrm{max}}{tDRDY\,\mathrm{min}} \qquad \mathbf{(5.1)}$$

where t_{Donemax} is the maximum latency for the first Donea_ signal to occur following a read command while t_{DRDYmin} is the minimum latency for the first DRDY signal transition. We will soon show that t_{Donemax} is a function of Latency and output logic path delay.

In our example, Figure 5.2 shows that there are 4 latch controllers (counting the latch controller utilizing the HFF storage) in path A and 5 latch controllers in path B. We see from the timing diagram that both paths are loaded with the same DRDY signal; although, path B is required to hold data for one extra clock cycle relative to path A. By adding an extra latch controller to path B we ensure that both paths are capable of the same throughput.

5.3.1 FIFO Throughput

The performance of the output data path is directly affected by the throughput of the FIFO. We define throughput of the FIFO as the ability of the asynchronous pipeline to maintain the required data rate at the FIFO output. This means that for every FIFO load, a complementary FIFO extraction must occur within the minimum and maximum cycle times for the pipelined latch controllers, which form the FIFO. The throughput of the asynchronous FIFO is a function of the number of data items present in the pipeline. When the number of data items in the pipeline is small, the throughput is low and the pipeline is said to be "data limited." Likewise, when the pipeline is nearly full, the throughput is limited because empty stages, or "holes," are required to allow data items to flow forward in the pipeline; in this scenario the pipeline is said to be "hole limited" [32]. Often, latency is sacrificed in an asynchronous pipeline to achieve greater throughput. Greater throughput is gained by increasing the number of data items in the pipeline so that a sustainable data rate can be maintained. In our application, throughput is only an issue in the extreme cases. Unlike synchronous pipelines, the latency through an asynchronous pipeline is not limited by clock frequency. Therefore, one great advantage to using an asynchronous pipeline as a FIFO is the relatively constant latency through the pipeline path. This property is very important in our DRAM application since we require fast data array access to meet read latency requirements.

In order to evaluate FIFO performance, we must first take a more detailed look at the circuits used to implement the semi-decoupled latch controller. Referring back to the circuit schematic of Figure 4.23, which is repeated in Figure 5.6, we see that the latch controller is implemented using two asymmetric C-elements. The transistor level implementation of these devices is shown in Figure 5.7. The transistor implementation also shows the Reset_ signal that is used to initialize the latch controller states within the FIFO upon power-up.

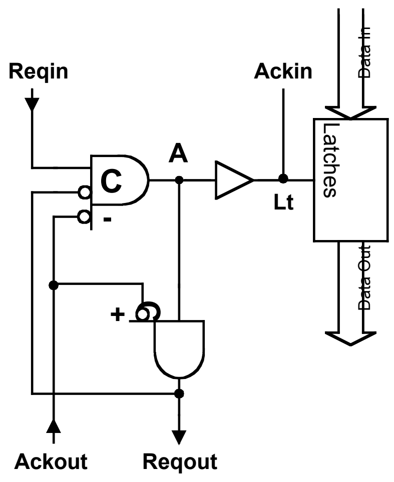


Figure 5.6 Semi-decoupled Latch Controller

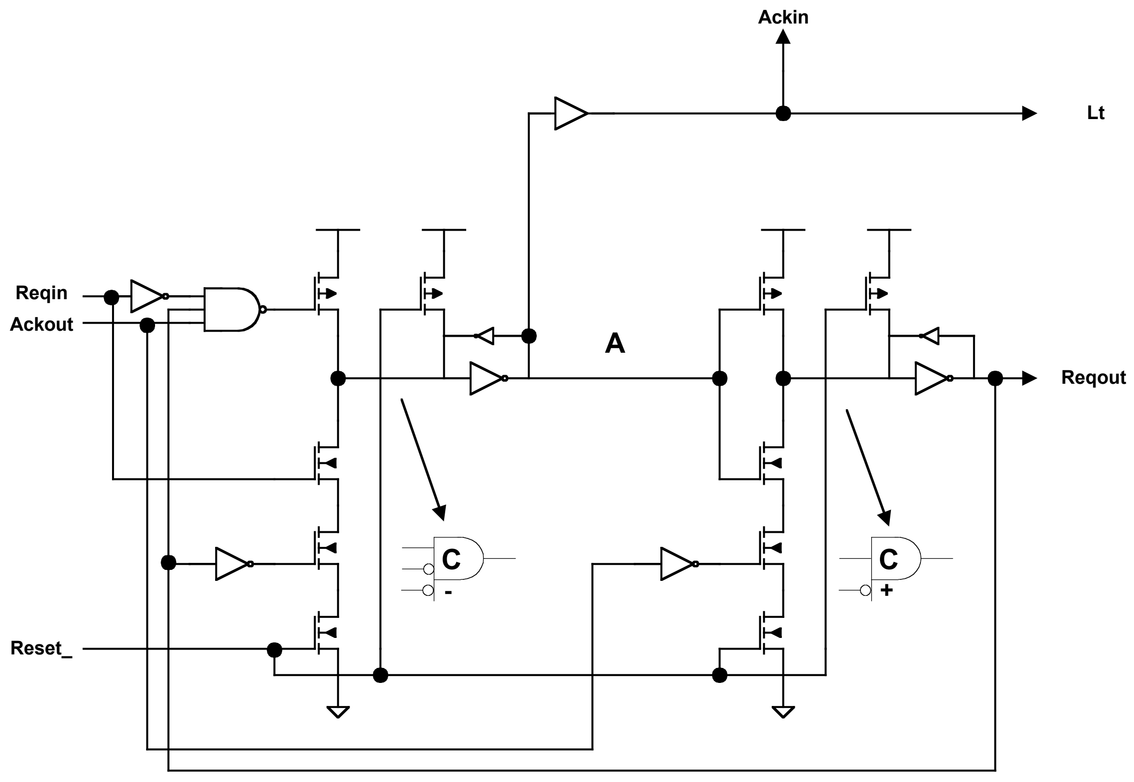


Figure 5.7 Transistor Implementation of Semi-decoupled Latch Controller

5.3.2 Latch Controller Forward Latency

First, we will look at the forward latency through the latch controller circuit. Suppose that the FIFO pipeline is empty. In this case, each FIFO latch controller will be in the state represented by STG marking labeled 1 in Figure 5.8. The forward latency of the data is separate from the forward latency of the control signals. The data will only see the delay through a simple TG latch at each stage of the pipeline. Even though the data will arrive at the output of the FIFO pipeline before the Lt signal of the last stage latches the data, we are concerned with the forward latency of the control signals because the timing of data loaded at the input of the pipeline is independent of the timing of the data extracted from the output of the pipeline. For logically correct operation, we must ensure the control signals at each end of the pipeline are correctly timed. Once we know the forward latency for each stage of the pipeline, then we can calculate the forward latency through the entire pipeline.

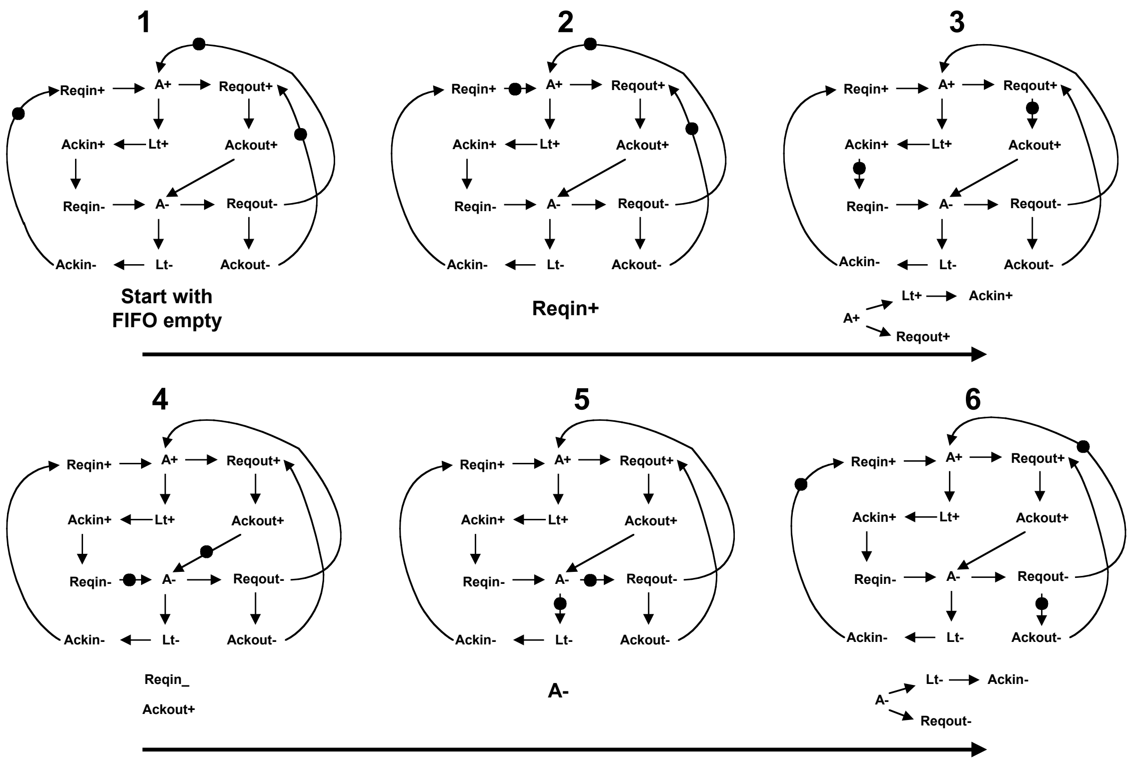


Figure 5.8 STG Markings for Latch Controller Forward Latency

We will follow the STG markings shown in Figure 5.8 and apply these sequences to the circuit topology of Figure 5.7 to determine the latch controller forward latency, t_{FL}. First, the FIFO starts empty until the Reqin+ signal transitions and we move to STG marking 2. From STG marking 2 the circuit transitions to STG marking 3 with only internal circuit delay. The delay component to transition from STG marking 2 to marking 3, is indicated as t_{RAF}, and is derived from the circuit in Figure 5.7 as follows:

$$t_{RAF} = t_{PD} + t_{INV} + t_{BUF} \qquad \textbf{(5.2)}$$

where t_{PD} is the delay through the pull down stack at the input to the inverter driving signal A, t_{INV} is the delay through the inverter and t_{BUF} is the delay through the buffer driving Lt/Ackin. The transition of Reqout+ is concurrent with the transition sequence to Ackin+ as shown below STG marking 3.

Next, the controller is waiting on signals Reqin- and Ackout+ from adjacent controllers to enable transition A- to progress from STG marking 3 to marking 4. These two signals transition concurrently. The Ackin+ signal is slightly slower getting back to the previous controller than the Reqout+ signal is getting to the next controller. Also, the Reqin- signal requires the 3-input NAND gate from the previous controller to transition. Therefore, the transition of A- in STG marking 5 is limited by the transition of Reqin-. We will call the delay from STG marking 3 to marking 4 t_{REQINF} and it is determined from Figure 5.7 as follows:

$$t_{REQINF} = t_{NAND} + 2 \cdot t_{PU} + 2 \cdot t_{INV} + t_{BUF} \qquad \textbf{(5.3)}$$

where t_{NAND} is the delay through the 3-input nand gate and t_{PU} is the delay through the p-channel pullup devices. Of particular interest in this design is that the pulse width of the input signal DRDY determines when Reqin- occurs on the first controller in both pipelines. Also, the complex gate itself only delays the Ackin+ transition from the complex gate used as the last controller in each pipeline of Figure 5.2.

Finally, to get to STG marking 6, requires the A- transition summed with the buffer delay to drive Ackin- back to the previous controller indicating new data can be transferred to the current latch controller. STG marking 6 shows that the controller is prepared to accept new data while the next controller is still holding Lt/Ackout high. This means that the next controller is still holding the previously transferred data and is not ready to accept new data. Again, this is the characteristic of the semi-decoupled latch controller that allows all of the storage locations in the pipeline to hold unique data. The delay to transition from STG marking 4 to marking 6 is indicated as t_{RAR} and can be calculated from our circuit diagram as follows:

$$t_{RAR} = t_{NAND} + t_{PU} + t_{INV} + t_{BUF} \qquad \mathbf{(5.4)}$$

Now we can calculate the total forward latency of the interface between two semi-decoupled latch controllers by summing the component delays derived from the STG markings in Figure 5.8 and the circuit schematic of Figure 5.7:

$$t_{FL} = t_{RAF} + t_{REQINF} + t_{RAR} \qquad \mathbf{(5.5)}$$

5.3.3 Latch Controller Reverse Latency

Another timing parameter of interest is the reverse latency, $t_{RL,}$ through the semi-decoupled latch controller. Suppose that all of the stages in the FIFO contain unique data. In that case, each latch controller can be represented by STG marking 1 in Figure 5.9. We define the reverse latency, $t_{RL,}$ for our semi-decoupled latch controller as the delay from the time the Ackout- signal transitions until the Lt-/Ackin- signal transitions, thereby, allowing new data to be captured in the controller latches.

The reverse latency parameter, t_{RL}, is very important in our application because we need to know how much delay occurs when transferring an empty latch condition, otherwise referred to as a hole, from the last controller stage in the pipeline to the first controller stage in the pipeline when the pipeline is full. A lack of holes in the pipeline is the opposite problem of throughput limitation caused by a lack of data in the pipeline. In order to increase data throughput, in the case of data limited operation, greater latency is required between a data load operation, indicated by DRDY transitioning, and unloading data from the FIFO, indicated by the Done(x)_ signals transitioning. Conversely, hole limited operation results in reduced throughput caused by too much latency between loading and unloading of the FIFO pipeline. Evaluation of hole limited operation helps to determine the cycle time of the FIFO and will allow us to further set limits on the operating frequency of this section of the DRAM data path.

Before looking further at data limited operation and hole limited operation, we will evaluate reverse latency, t_{RL}, employing the same method used to determine the forward latency, t_{FL} in the previous section. First, consider the signal timing for latch controller LC1 in Figure 5.2 when all stages of the FIFO contain unique data. As data is consumed from the FIFO, a hole percolates back toward LC1. After Ackout- to occurs at the input of LC1 we advance to STG marking 2 of Figure 5.9. The state of LC1 continues to STG marking 3 after a delay, t_{AR} derived from Figure 5.7 as follows:

$$t_{AR} = 2 \cdot t_{INV} + t_{PD} \qquad \textbf{(5.6)}$$

where t_{INV} is equal to the delay through an inverter and t_{PD} is the delay through the pull-down n-channel transistor stack at the input to the Reqout inverter driver.

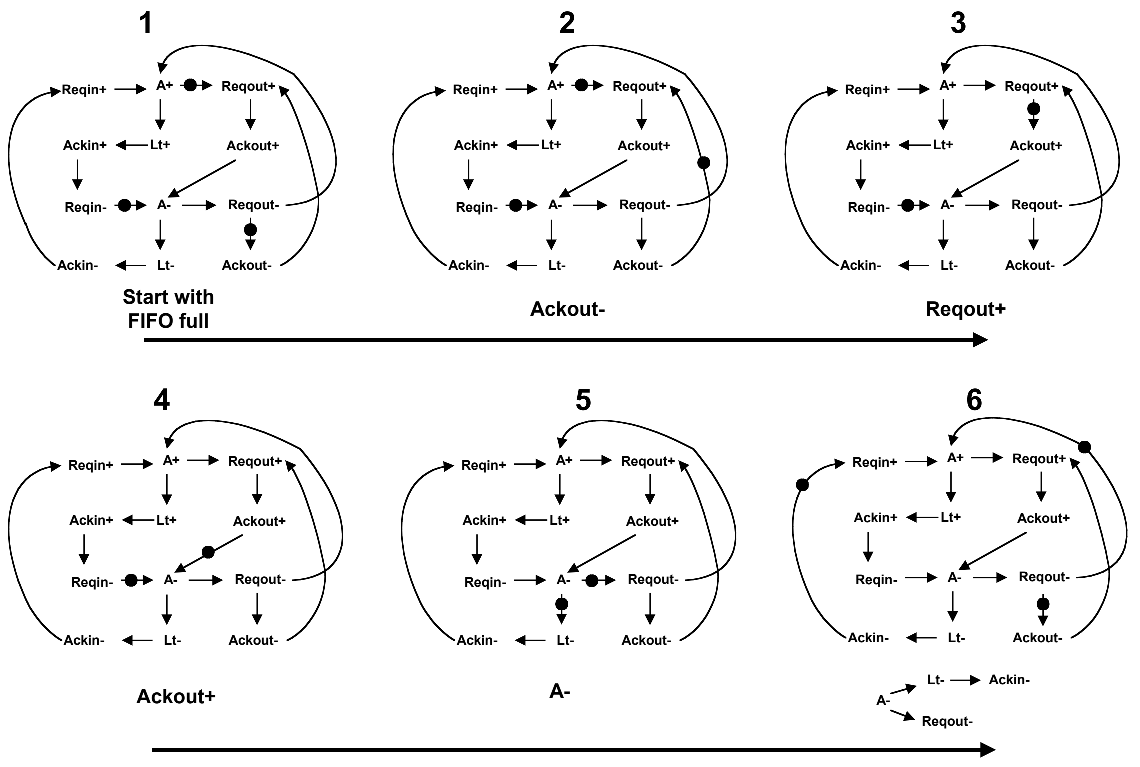


Figure 5.9 STG Marking Sequence for Reverse Latency Timing Analysis

Marking 4 in Figure 5.9 occurs after LC2 acknowledges back to LC1 the Reqout+ transition from LC1 (Reqin+ for LC2). We will call this delay t_{RA} and it can be derived from the circuit in Figure 5.7 by following the Reqin+ to Ackin+ logic path. Remember that Ackin+ for LC2 is Ackout+ for LC1. The marking for LC2 will be identical to STG marking 6 since LC2 has already had a full reverse latency delay as it passes the hole back to LC1. Following the path from Reqin+ to Ackin+ gives us t_{RA} as follows:

$$t_{RA} = t_{PD} + t_{INV} + t_{BUF} \qquad \textbf{(5.7)}$$

where t_{PD} is the delay through the n-channel pull-down stack at the input to the inverter driving signal A, t_{INV} is the inverter delay and t_{BUF} is the delay through the buffer driving Lt and Ackin.

The last component of the reverse latency delay is the circuit delay from Ackout+ to Ackin- shown in STG markings 5-6. STG Marking 6 shows the concurrent modeling capability of the STG. When A- fires, both the Reqout- and Lt-/Ackin- logic paths are enabled. The delay between Ackout+ to Ackin- is labeled t_{AA} and is derived as follows:

$$t_{AA} = t_{NAND} + t_{PU} + t_{INV} + t_{BUF} \qquad \textbf{(5.8)}$$

where t_{NAND} is the delay through the 3 input nand gate, t_{PU} is the delay through the p-channel device at the input to the inverter driving signal A, t_{INV} is the delay through the inverter and t_{BUF} is the delay through the buffer driving Lt and Ackin.

The total reverse latency for a single semi-decoupled latch controller is determined by the sum of the delays outlined above. Think of this delay as the transition of the latch controller from a latched or busy state to an unlatched or available state. The delay parameter, $t_{RL,}$ is calculated as follows:

$$t_{RL} = t_{RA} + t_{AR} + t_{AA} \qquad \textbf{(5.9)}$$

5.3.4 Reverse Latency of the Simplified Latch Controller Interface

The reverse latency calculated above is only true for the interface between two semi-decoupled latch controllers. We must also consider the reverse latency between the latch controllers LC3 and LC2 in pipeline A. Because LC3 is not a semi-decoupled latch controller, we need to consider the effect that timing of the Lt/Ackin signal from LC3 has on LC2.

Again, referring to STG marking 1 in Figure 5.9, we see that until Ackout- occurs, the semi-decoupled latch controller is holding unique data that is ready to be forwarded in the FIFO pipeline. After Ackout- occurs, the latch controller progresses to marking 3 and is requesting service from the next controller. Not until Ackout+ occurs on the latch controller input is the latch controller able to accept new data. This is because the forward latch controller has not acknowledged the current data. The earlier the forward latch controller acknowledges the new data with Ackout+, the sooner the current latch controller can transition Lt-/Ackin- to indicate that new data can be passed from the previous controller.

Consider the interface to the simplified latch controller, LC3. The FIFO Output Sequencer generates the Donea_ signal after the first two bits of the 4-bit burst are consumed. The Donea_ signal is low for ½ of a clock period. Looking at the truth table of Table 5.1, we see that as long as Donea_ (R1_) is low, the output, Lt/Ackin is low. This means that Ackout of LC2 is low for ½ of a clock period. During this time, LC2 signals a request, Reqout+, to LC3 but the request is masked by the Donea_ signal. Not until the Lt/Ackin signal of LC3 transitions high, indicating receipt of new data, can LC2 indicate back to LC1 that new data can be accepted. This is the operation sequence illustrated by STG markings 1-6 in Figure 5.9. Thus, the reverse latency for the LC3 to LC2 interface, t_{RLS}, is as follows:

$$t_{RLS} = \frac{T}{2} \quad \text{iff } \frac{T}{2} \geq t_{AR} \qquad \textbf{(5.10)}$$

where T is the clock period. The reverse latency of this stage is not dependent on t_{AR} from LC2 as long as $\frac{T}{2} \geq t_{AR}$. Equation 5.10 indicates that using the simplified latch controller compromises performance of the FIFO. We trade performance for area when using the simplified controller versus using a semi-decoupled controller in this application.

Now that we have established the logic delays for the forward and reverse latencies through the individual latch controllers, we can proceed to calculation of throughput as a function of clock frequency. As previously mentioned, the throughput of the FIFO is limited both when the FIFO pipeline is data starved or, more formally, data limited; and when the FIFO pipeline is overfed or, more formally, hole limited. In the next section we will examine the FIFO design used in this work and establish a method for calculating throughput as a function of frequency that can be applied in a general case.

5.4 FIFO Throughput as a Function of Clock Frequency

The latch controller latencies calculated in the previous section will now serve as parameters for determining the maximum clock frequency at which the FIFO can operate. As we previously noted, the throughput of the FIFO is a function of the number of data items stored in the pipeline at any given time. We will examine two modes of FIFO operation. The first mode is data limited operation and the second mode is hole limited operation [32]. We will also apply a third criteria for estimating the maximum clock frequency by determining whether the minimum delay between the FIFO load timing and the data extraction timing is greater than the forward data latency of the FIFO. These three criteria will establish a method for deriving general formulas that are necessary for making correct engineering decisions when designing a data path similar to the DRAM data path described in previous chapters.

5.4.1 Array Access versus Read Latency Requirements

First, consider that there is only one data item in the FIFO pipeline at any given time. On each column access cycle, a data item is removed when the Donea_ signal transitions low. In this case, the timing margin between loading data from an array access, indicated by DRDY transitioning high, and the removal of the data, indicated by the Donea_ signal transitioning low, must be at least equal to the time it takes for the data to flow from the input of the FIFO pipeline to the output of the pipeline. This timing constraint says that the last latch controller must transition an Ackin+ signal before Donea_ transitions low in order for the logic to function properly. We need to consider the Reqin+ to Reqout+ delay, t_{RR}, for each intermediate stage of the FIFO pipeline and the Reqin+ to Ackin+, t_{RQA}, delay of the last stage. As long as the last stage transitions its Ackin+ signal before the Donea_ signal transitions low, the logic in the FIFO pipeline will function properly. If this condition does not exist then we would acknowledge data at the output of the pipeline before data arrived meaning the FIFO is data starved and invalid data will be loaded into the output data serializer. We would also risk logic failure in the FIFO pipeline.

Of course, the latency for data to travel through the empty pipeline is assumed to be much shorter than the logic delay associated with the latch control signals. The data must arrive at the data output serializer ½ of a clock cycle, plus register setup time, before Donea_ transitions low. The following analysis assumes that the array access occurs with enough timing margin before the internally timed read latency expires. The required timing stipulation for correct FIFO pipeline operation is expressed below:

$$t_{Done} - t_{DRDY} \geq S_{sd} \cdot t_{RR} + t_{RQA} \qquad \textbf{(5.11)}$$

where t_{DRDY} is the delay from a read command to the DRDY signal transtion; t_{Done} is the delay from a read command to the Donea_ signal transition; and S_{sd} is the number of latch controller stages, excluding the last latch, in the FIFO Controller. The value of t_{RQA} is simply the delay through the complex gate from the Reqout+ transition driven from the previous latch controller to the output Lt/Ackin+ transition of the simplified latch controller. We determine t_{RR} from the circuit diagram in Figure 5.7 by following the logic path from the transition of the Reqin+ signal to the transition of the Reqout+ signal as shown in Equation 5.12:

$$t_{RR} = 2 \cdot t_{PD} + 2 \cdot t_{INV} \qquad \textbf{(5.12)}$$

where t_{PD} is the delay through the pull down stack and t_{INV} is the delay through an inverter.

The parameter, t_{Done} in Equation 5.11 is a function of both the programmed read latency, L, and the delay through the output data path, which is the delay measured through the output data register/serializer and the DQ pad driver. We will call the delay through the output data path t_{output}. The reason we can establish this relationship is because the read command is issued at the DRAM inputs relative to the external clock. Recall that the read latency is also programmed and timed relative to the external clock. Internally, the output of the DLL circuit is used to back time the output clock that drives the data output register/serializer by the delay t_{output}. Therefore, the QED signal, indicated in Figure 5.2, is timed through the synchronization logic to synchronously enable the DQ pad driver through the data output register. This timing is made possible by the back timed alignment of the DLL output clock (Appendix A). We can express the delay between the read command and the synchronous capture of the QED signal at the data output register, t_{QED}, as a function of read latency:

$$t_{QED} = L \cdot t_{CK} - t_{output} \qquad \textbf{(5.13)}$$

where t_{CK} is the clock period. As shown in Figure 5.1, the FIFO Output Sequencer also uses the QED signal to time the assertion of the Done(x)_ signals. The Donea_ signal is asserted on the falling edge of the clock following the positive edge capture of the QED signal (Figure 5.5). Therefore, the delay, t_{Done} can be expressed as a function of read latency and clock period as follows:

$$t_{Done} = t_{QED} + \frac{t_{CK}}{2} = L \cdot t_{CK} - t_{output} + \frac{t_{CK}}{2} = (L + \frac{1}{2}) \cdot t_{CK} - t_{output} \qquad \textbf{(5.14)}$$

Equation 5.14 can be simplified to give us the expression for a clock period boundary as a function of FIFO pipeline forward latency:

$$t_{CK} \geq \frac{t_{DRDY} + t_{output} + (S_{sd} \cdot t_{RR} + t_{RQA})}{L + \frac{1}{2}} . \qquad \textbf{(5.15)}$$

Now we see that by increasing the latency, which in turn increases the timing margin between t_{DRDY} and t_{Done}, we can increase the maximum operating clock frequency based on the latency between the load and unload signals.

One very important aspect to the latency analysis is that the timing parameters are taken at the slowest operating corner (slow process, low voltage, high temperature) and minimum read latency. This is important because it forces the minimum delay between the high transition on DRDY and the low transition on Donea_. Equation 5.15 gives us the minimum clock period required to meet the forward control signal latency through the FIFO pipeline. The boundary condition established by Equation 5.15 must be met for correct FIFO operation.

5.4.2 Data Limited Operation of the FIFO Pipeline

Once we have established a maximum clock frequency from Equation 5.15, we can now analyze the operation of the FIFO for maximum sustainable throughput. Again, suppose the FIFO is empty and one data item is passed through the FIFO at any time. This would require that the data item be extracted before a new item is loaded. The cycle time of the data load signal, DRDY will be a function of the data prefetch depth and clock period. Therefore, in order to maintain sustained data throughput, the delay between DRDY and Donea_ must be at least equal to the time required for the data to flow from the input of the FIFO to the output of the FIFO:

$$\frac{P}{2} \cdot t_{CK} \geq S_{sd} \cdot t_{FL} + t_{SFL} \qquad \textbf{(5.16)}$$

where P is the prefetch depth. We divide P by 2 because the data path is double data rate so that two bits are output per clock cycle.

For the general case, if there are *n* data items in the FIFO pipeline then the forward latency of the pipeline must support *n* data loads. The forward latency condition tests for open locations to load data given the minimum cycle time ensuring that we do not become data limited for sustained throughput:

$$n \cdot \frac{P}{2} \cdot t_{CK} \geq S_{sd} \cdot t_{FL} + t_{SFL} \qquad \textbf{(5.17)}$$

$$t_{CK} \geq \frac{2 \cdot (Ssd \cdot tFL + tSFL)}{n \cdot P} \qquad \textbf{(5.18)}$$

Equation 5.18 is the expression for minimum clock period for data limited operation.

5.4.3 Hole Limited Operation of the FIFO Pipeline

Now let us imagine that the FIFO pipeline is nearly full. As a new data item is loaded into the FIFO input, a data item is simultaneously extracted from the FIFO output. As this occurs, a hole is injected into the output and begins to move toward the input latch controller stage. In our application, the hole must reach the input stage before a request to load more data into the FIFO occurs. This condition generally occurs at the fast operating corner (fast process, high voltage, low temperature) combined with maximum allowable programmed read latency. Under these conditions, faster consecutive array accesses create a high data throughput demand at the input of the FIFO while extraction timing remains relatively fixed by clock frequency, read latency and the output data path delay (t_{Done}).

Analysis of hole limited operation is valid for the short path A of the FIFO Controller (Figure 5.2) because path B is made longer to compensate for the extra clock cycle of latency between Donea_ and Doneb_ transitioning during the initial data extraction. Because both pipelines share the input request signal DRDY, and there is an additional cycle of latency between extraction of data in path B compared to path A, either path can be evaluated independently giving the same estimation of data throughput.
The analysis for hole limited operation is very similar to the analysis of data limited operation examined above. Just as in the data limited analysis, in order to achieve the minimum FIFO throughput for the hole limited condition, the FIFO reverse latency must occur within a column access cycle time:

$$\frac{P}{2} \cdot t_{CK} \geq S_{sd} \cdot t_{RL} + \frac{t_{CK}}{2} \quad \textbf{(5.19)}$$

where P is the data prefetch depth, S_{sd} is the number of semi-decoupled latch stages and t_{RL} is the reverse latency between two semi-decoupled controllers. The addition of ½ t_{CK} represents the pulse width of the Donea_ signal, which is a performance-limiting factor for reverse latency when using the simplified latch controller as the last stage of the FIFO pipeline. Simplifying Equation 5.19 gives us:

$$t_{CK} \geq \frac{2 \cdot S_{sd} \cdot t_{RL}}{P - 1} \qquad \textbf{(5.20)}$$

We can generalize Equation 5.20 for cases where the pipeline is not full. If *n* is the number of unique data items in the FIFO pipeline and S is the total number of latch controller stages in the pipeline, then the pipeline has S-*n* holes available for new data. Each of the available holes can be filled with new data before a hole injected at the output is required to reach the FIFO input stage. Equation 5.20 is generalized as follows:

$$\frac{P}{2} \cdot (S - n) \cdot t_{CK} + \frac{P}{2} \cdot t_{CK} \geq S_{sd} \cdot t_{RL} + \frac{t_{CK}}{2} \qquad \textbf{(5.21)}$$

$$t_{CK} \geq \frac{2 \cdot S_{sd} \cdot t_{RL}}{P \cdot (S - n + 1) - 1} \qquad \textbf{(5.22)}$$

5.4.4 FIFO Performance Boundaries

Equations 15.18 and 15.22 provide upper-bounds on the operating frequency of the FIFO pipeline portion of the DRAM read data path, as a function of the number of data items present in the pipeline. We can express the operating frequency, *F*, as a function of the number of data items in the pipeline as follows:

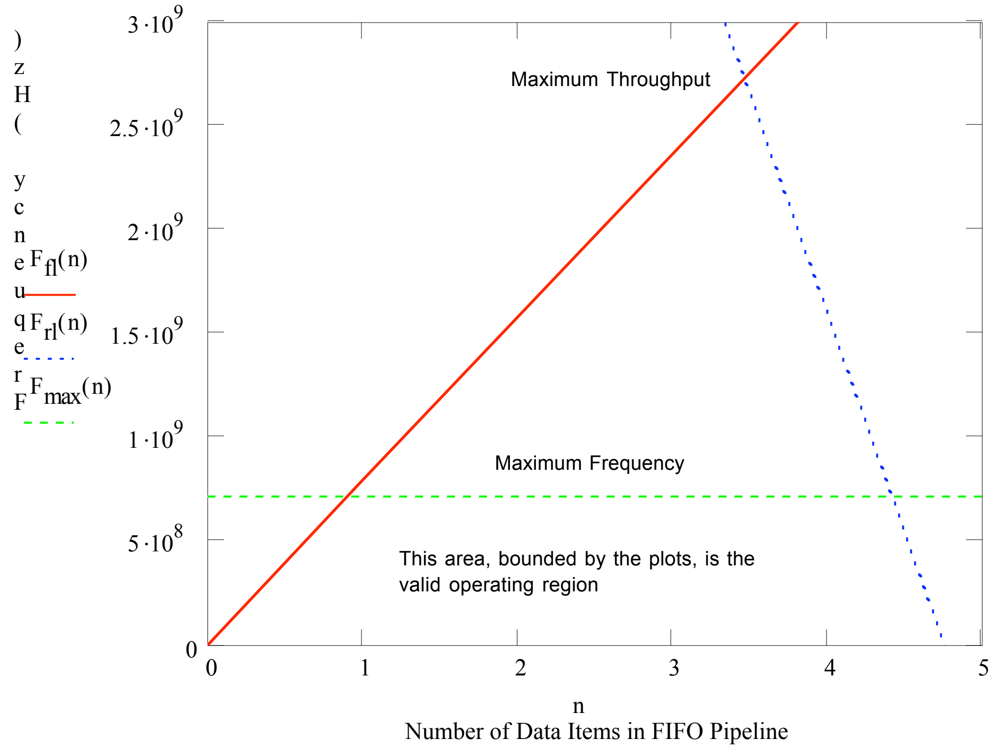
$$F = \frac{1}{t_{CK}} \leq Min\left(\frac{n \cdot P}{2 \cdot (S_{sd} \cdot t_{FL} + t_{SFL})}, \frac{P \cdot (S - n + 1) - 1}{2 \cdot S_{sd} \cdot t_{RL}} \right) \qquad \textbf{(5.23)}$$

Graph 5.1 is a plot of the upper bounds on the FIFO operating frequency. This is a plot of operating frequency versus data items in the pipeline derived from Equation 5.23 and Equation 5.15 [32]. The rising portion of the curve represents data limited operation where data throughput rises linearly with the number of items present in the pipeline. The falling portion of the curve represents data throughput for hole limited operation where data throughput falls with increasing number of data items in the pipeline. The point at which the two curves meet is the maximum throughput capability of the FIFO. The third boundary is the maximum frequency at which the FIFO pipeline can operate based on the minimum difference in delay between the initial DRDY signal high transition and the initial Donea_ signal low transition. This boundary was determined from Equation 5.15 using a read latency value, L, of 8 clock cycles. The numbers used to derive the plot in Graph 5.1 come from actual circuit parameters determined through extensive SPICE simulations of circuits designed in accordance with work done for this thesis. These simulation results are based on a 0.11-micron DRAM process operating between 1.45 volts, 100 degrees Celsius and 2.25 volts, 0 degrees Celsius. The hole-limited operating curve is derived from the fast corner simulations while the data limited curve is derived from slow corner simulations.

Graph 5.1 is used to determine the limitations of operating at a given clock frequency. Let us choose an operating frequency of 700 MHz ($7 \cdot 10^8$ Hz) with a read latency of 8 clock cycles. At this point, we would be just below the calculated maximum frequency value based on Equation 5.15 of 708 MHz. This is a desirable operating point for the slow corner case since we are able to maintain data throughput with only one data item in the pipeline. At this frequency, we also still meet the maximum frequency stipulation based on clock frequency, read latency and output data path delay established in Equation 5.15. Notice also that there is still plenty of data throughput left to allow fast corner operation.

As the operating point of the DRAM transitions from a slow corner to a fast corner, more data items are loaded at the input of the FIFO before the read latency timing expires. Because read latency timing is a function of clock frequency; and variation in the data output delay, which directly affects the timing of the Donea_, is very small, changes in operating corner conditions greatly affect the DRDY signal while having relatively little affect on the timing of Donea_. Therefore, as we move toward fast corner environmental conditions, the operational status of the FIFO moves toward hole limited operation.

Of course, increasing the number of stages in the pipeline does not correct the problem of maximum data throughput. If the operation of the FIFO pipeline became hole limited, additional stages would correct the hole limited problem at the expense of increased forward latency in the pipeline. This would have the affect of lowering the maximum operating frequency, according to Equation 5.15, and would increase the slope of the data limited curve further lowering the maximum operating frequency.



Graph 5.1 Upper bounds on Operating Frequency for FIFO Pipeline

Figures 5.10 and 5.11 are SPICE simulation results included here to illustrate the operation of the FIFO pipeline under various occupancy rates. Figure 5.10 shows operation at the fast corner and Figure 5.11 shows operation of the FIFO pipeline at the slow corner. These simulation results show that the elasticity of the pipeline allows various timing relationships between the DRDY and Done(x)_ signals. In both cases, the operation of the circuit falls under the boundary conditions shown in Graph 5.1.

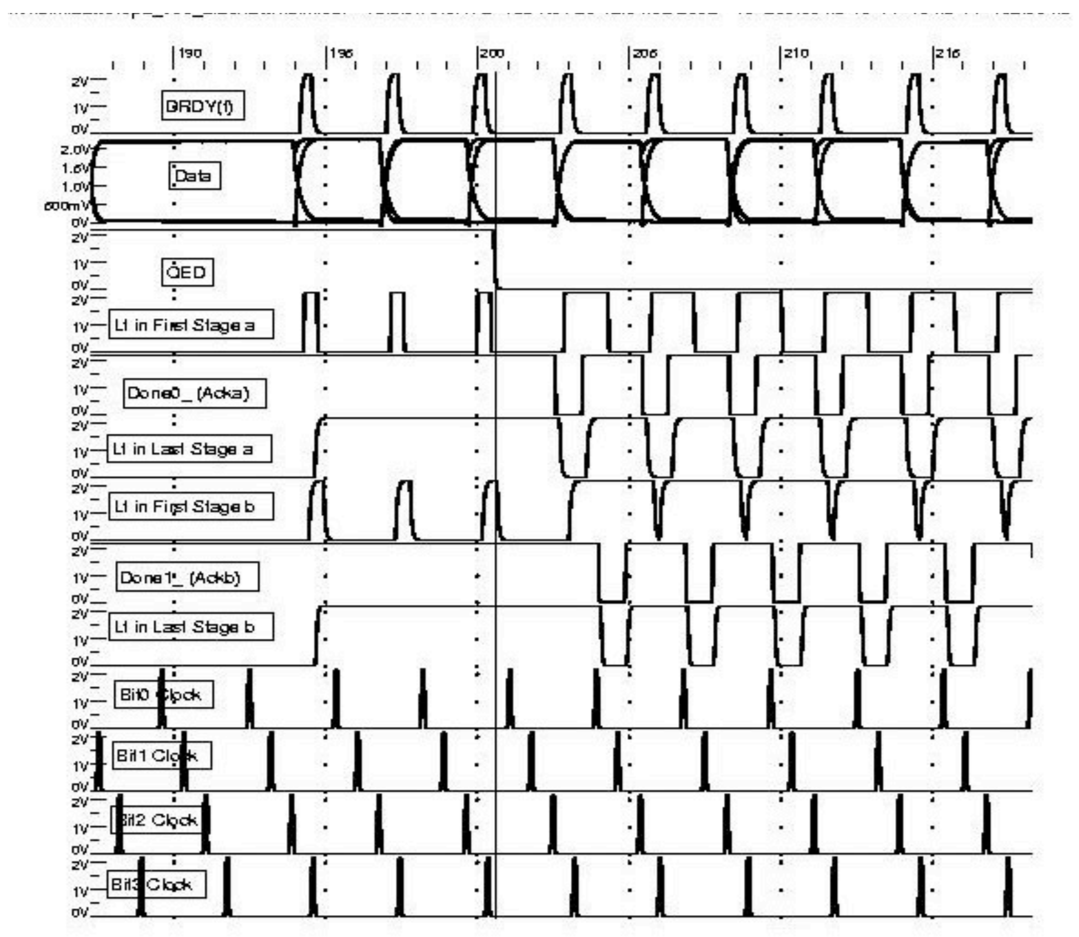


Figure 5.10 SPICE Simulation Results for FIFO Pipeline Operation at Fast Operating Corner

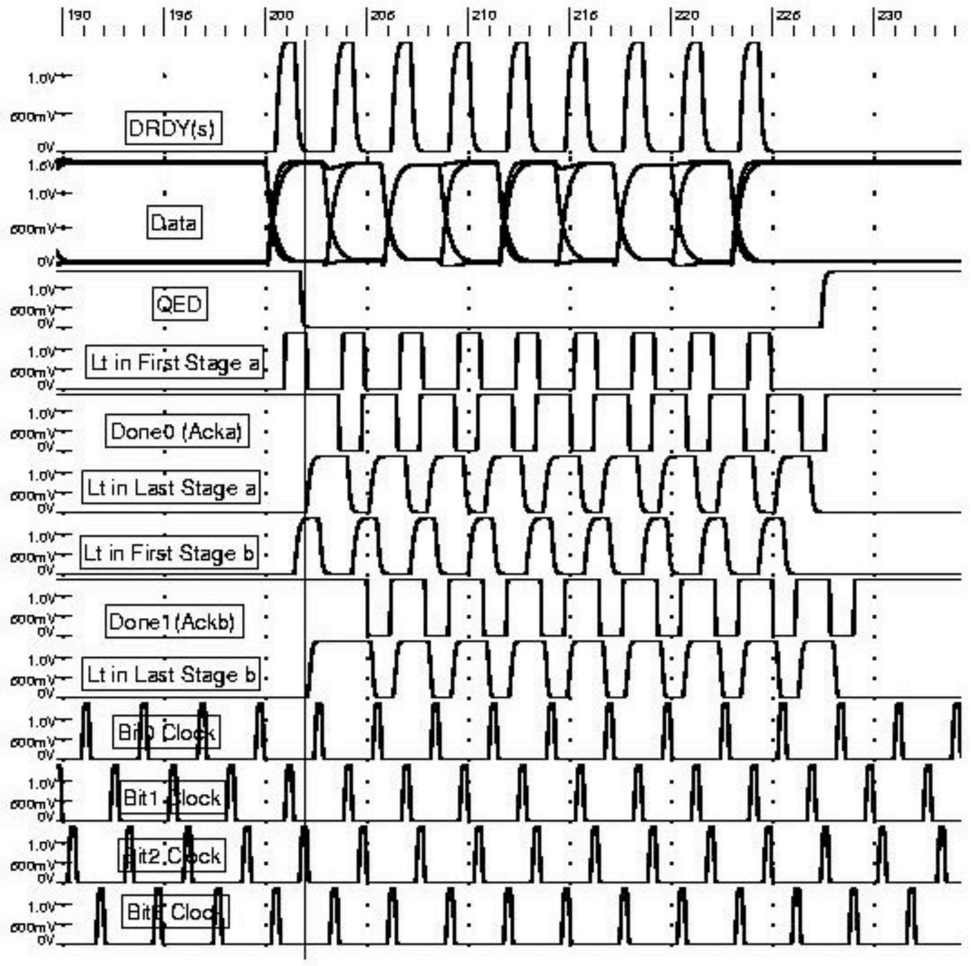


Figure 5.11 SPICE Simulation Results for FIFO Pipeline Operation at Slow Operating Corner

6. Conclusion

In this thesis, we have examined some of the issues surrounding the design of portions of a read data path in a synchronous DRAM. The work done for this thesis was applied to a very high frequency design and, from simulation results, has shown very good performance. We have focused on issues surrounding the data path timing in the area of the HFF and data output serializer interface. There are issues with timing the QED signal from the read command to the correct DLL output clock phase that were mentioned but not comprehensively covered. We were able to show that using a FIFO between the HFF circuits and the data output serializers we are able to provide coherent data according to the output timing established by the QED signal. When a read command is issued internally in the DRAM, two timing paths are established. One path is the array access timing where the DRAM core column access is performed, while the second timing path is the read latency timing where the data from the column access is delivered correctly timed and delivered to the DQ pad driver. It is through the data FIFO that the two timing paths are merged back into a single timing path.

Future work for DRAM data path improvements should involve methods for improved clock synchronization and latency timing techniques. Even though this thesis did not provide detailed analysis of synchronization and clock domain crossing techniques, one should not lose sight of the importance of these circuits for providing timing control of the DRAM data path. Consider that the read command is captured and decoded in the command/capture clock domain while the output synchronization occurs in the DLL output clock domain. The phase relationship between these two clock domains is arbitrary because of the constant phase adjustment of the DLL clock according to variations in the I/O model delay. In the case of this thesis, we have examined the clock domain crossing of data. This is true because input data to the FIFO was generated from the command clock domain controlling the cycle timing of array accesses while the DLL clock domain, coupled with the programmed read latency, determined output timing of the FIFO. Further research should confront the problem of transferring timing information between the command/capture clock domain and the DLL output clock domain, by virtue of the requirement for a DRAM to have a fixed, cycle-based read latency.

The thesis concludes with a comprehensive overview of the engineering behind the design of the asynchronous FIFO pipeline. We were able to establish preferred design architectures and develop performance metrics that can be altered and applied to assorted applications of asynchronous pipelines in a synchronous environment [32]. Future work in the area of asynchronous FIFO pipeline design could include methods of performance improvements through circuit architecture such as GaSP controllers [25] or changes to communication protocol through improvements in sequencing order [24].

References

[1] Keeth, B. and Baker, R.J. *DRAM Circuit Design: A Tutorial* , S.K. Tewksbury and J.E. Brewer, Eds., IEEE Press, Piscataway, NJ, 2001.

[2] Kim, J.J., et al., “A Low Jitter, Mixed-mode DLL for High-speed DRAM Applications,” in *IEEE J. Solid-State Circuits*, vol. 35, Oct. 2000, pp 1430-1436.

[3] Maneatis, J.G., “Low-jitter and Process-independent DLL and PLL Based on Self-biased Techniques,” in *IEEE J. Solid-State Circuits*, vol. 31, Nov. 1998, pp 1728-1732.

[4] Sidiropolous, S. and Horowitz, M., “A Semi-digital Delay Locked Loop with Unlimited Phase Shift Capability and .08 to 400 MHz Operating Range,” in *Dig. Tech. Papers Int. Solid-State Circuits Conference,* Feb. 1997, pp. 332-333.

[5] Dennard, R.H., Gaenssien, F.H., et al., “Design of Ion-implanted MOSFETs with Very Small Physical Dimensions,” in *IEEE J. Solid-State Circuits*, S.C.-9.5, Oct., 1974, pp 256-268.

[6] Patterson, D.A. and Hennessy, J.L., *Computer Organization & Design: The Hardware/Software Interface*, Morgan Kaufmann, San Francisco, CA, 1994 pp. 452-527.

[7] JEDEC standard, “Double Data Rate (DDR) SDRAM Specification, JESD-79, May 2002, Rel. 2.

[8] Lin, F. et al., “Method and System for Delay Control in Synchronization Circuits” Patent Pending.

[9] JEDEC standard, “Stub Series Terminated Logic for 2.5V (SSTL_2),” JESD8-9A , (revision of JESD8-9), December, 2000.

[10] JEDEC standard “Double Data Rate II (DDRII) SDRAM Specification,” Final Specification Pending.

[11] Dally, W.J. and Poulton, J.W. *Digital Systems Engineering*, Cambridge University Press, 1998, pp. 559-567.

[12] SLDRAM Specification, *Micron Technology DRAM Data Book*, 1997.

[13] B. Keeth, et al., “Calibration Technique for Memory Devices,” U.S. Patent No. 6,434,081, 1999.

[14] Collins, H.A. and Nikel, R.E., "DDR-SDRAM, high-speed, source-synchronous interfaces create design challenges," EDN Magazine, September 2, 1999, pp 63-72.

[15] Keeth, B. "Distributed High-speed Data Capture Scheme with Bit-to-bit Timing Correction," U.S. Patent No. 6,430,696, 1999.

[16] Couch, L.W., *Modern Communication Systems*, Prentice-Hall, Upper Saddle River, NJ, 1995, pp. 177-185.

[17] Harris, D., *Skew Tolerant Circuit Design*, Morgan Kaufmann Publishers, San Diego, CA, 2001.

[18] Shaw, A.W., *Logic Circuit Design*, Saunders College Publishing, 1993, page 409.

[19] Sutherland, I.E. "Micropipelines," in *Communications of the ACM*, vol. 32, no. 6, June, 1989, pp. 720-738.

[20] Dally, W.J. and Poulton, J.W. *Digital Systems Engineering*, Cambridge University Press, 1998, pp. 486-502.

[21] Maxfield, C., *Designus Maximus Unleashed!*, Newnes, 1998, pp. 219-232

[22] Myers, C.J., *Asynchronous Circuit Design*, Wiley-Interscience, July, 2001.

[23] Singh, M. and Nowick, S.M., "High Throughput Asynchronous Pipelines for Fine-Grain Dynamic Datapaths," in *Proc. of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, April, 2000, pp. 198-209.

[24] Renaudin, M., Bachar, E.H. and Guyot, A., "A New Asynchronous Pipeline Scheme: Application to the Design of a Self-Timed Ring Divider," in *IEEE J. Solid-State Circuits*, vol. 31, no. 7, July, 1996, pp. 1001-1013

[25] Sutherland, I.E. and Fairbanks, S., "GasP: A Minimal FIFO Control," in *Proc. of the Seventh International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 2001, pp. 46-53.

[26] Yun, K.Y., Beerel, P.A. and Arceo, J., "High-performance Two-Phase Micropipeline Building Blocks: Double Edge-Triggered Latches and Burst-Mode Select and Toggle Circuits," in *IEEE Proc. Circuits, Devices and Syst.*, vol. 143, no. 5, Oct., 1996, pp. 282-288.

[27] Furber, S.B. and Day, P., "Four-Phase Micropipeline Latch Control Circuits," in *IEEE Trans. on Very Large Scale Int. Sys.*, vol. 4, no. 2, Jun., 1996, pp. 247-253.

[28] Unger, S.H., *The Essence of Logic Circuits*, IEEE Press, Piscataway, NJ, 1997, pp. 246-249.

[29] Kovalyov, A. "A Polynomial Algorithm to Compute the Concurrency Relation of a Regular STG," in *Hardware Design and Petri Nets*, Yakolev, A., et al., Eds., Kluwer Academic Publishers, Boston, 2000, pp. 107-126.

[30] Hintz, K. and Tabak, D. *Microcontrollers Architecture, Implementation & Programming*," McGraw-Hill, 1992, pp. 93-98.

[31] Cortadella, J., et al., *Logic Synthesis of Asynchronous Controllers and Interfaces*," Springer, Berlin, 2002.

[32] Singh, M., Tierno, J.A., Rylyakov, A., Rylov, S. and Nowick, S.M., "An Adaptively-Pipelined Mixed Synchronous-Asynchronous Digital FIR Filter Chip Operating at 1.3 GigaHertz," in *Proc. of the Eighth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, IEEE Computer Society, 2002, pp. 84-95.

Appendix A

A.1 Delay Locked Loop Operation in a DRAM Application

The delay locked loop (DLL) circuit is used in new generation DRAM devices to provide accurate alignment of data with the DRAM external clock. In older generation DRAMs, clock frequencies were relatively low and the delay of the input and output circuits internal to the DRAM were a small percentage of the clock period. There is a standard timing parameter that relates the output data from the DRAM to the external data clock called, t_{AC}. This timing relationship is shown below in Figure A.1.

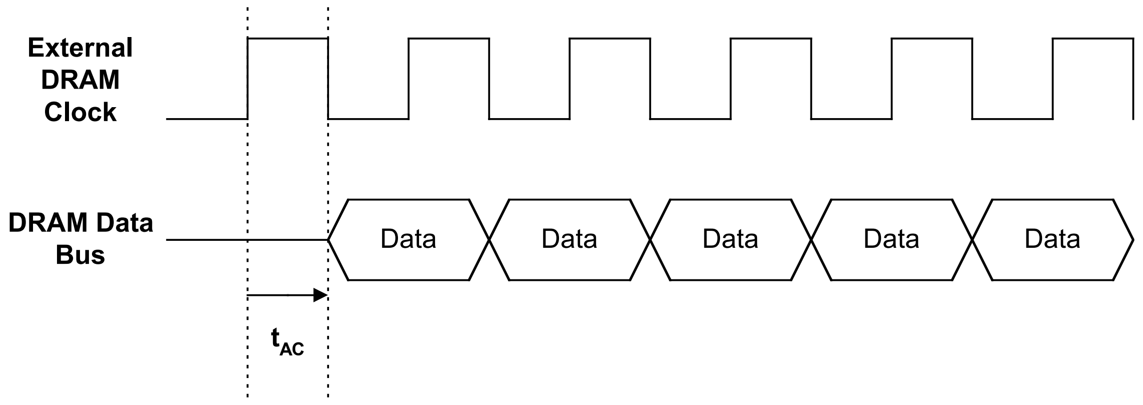


Figure A.1 tAC Timing Parameter Definition

In early synchronous DRAM designs, the external clock was directly routed internally through the DRAM and used to capture command, address and data information. The same internally routed clock was used to drive a synchronous data output latch with the data from the latch passed through the DQ pad driver to environment external to the DRAM. The penalty from this design methodology was that by the time the read data was driven from the DRAM, the clock that drives the output data latch had suffered from internal routing delays.

In addition to the clock routing delays, the data suffered further delay through the data output latch and DQ pad driver circuitry. As long as these delay penalties resulted in data being driven from the DRAM with less than the delay specified by the tAC timing parameter, correct system operation was possible.

As clock frequency has continued to increase, the routing and delay penalties suffered by the data output from the DRAM has become a larger percentage of the clock period. In actuality, the routing and delay penalties have become multiple clock cycles long so that predictable output data timing relative to the external clock is impossible to maintain over changes in process, voltage and temperature. DLL circuits have become a standard circuit used to align the data with the external clock in a predictable manner.

A.2 Basic DLL Operation

Figure A.2 is a top-level block diagram of a simple DLL circuit. We will explain the function of each of the circuit blocks shown and then provide timing relationships that show how the DLL provides synchronous alignment of the DRAM output data with the external clock.

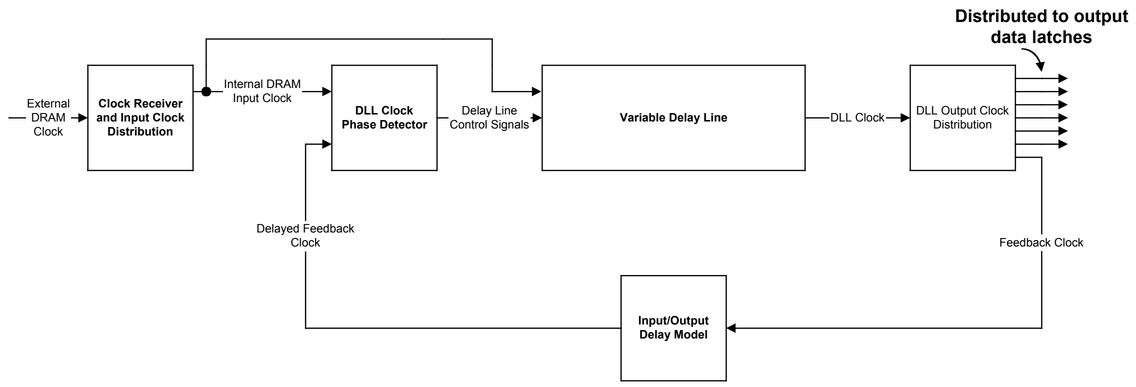

Distributed to output data latches
External DRAM Clock
Clock Receiver and Input Clock Distribution
Internal DRAM Input Clock
DLL Clock Phase Detector
Delay Line Control Signals
Variable Delay Line
DLL Clock
DLL Output Clock Distribution
Delayed Feedback Clock
Feedback Clock
Input/Output Delay Model

Figure A.2 Top Level DLL Block Diagram

Referring to Figure A.2, we start at the left of the diagram with the block labeled "Clock Receiver and Input Clock Distribution." The external clock signal is applied to the input clock pad and is immediately distributed to the clock receiver. The clock receiver detects external clock transitions and, if necessary, converts them to CMOS voltage levels. The clock is then distributed from the output of the clock receiver to the one of the inputs of the block labeled "DLL Clock Phase Detector."

The DLL Clock Phase Detector is a comparator that provides servo control of the entire feedback loop through adjustments of the Variable Delay Line. Initially, the DLL is said to be out of "lock." This means the phase difference between the Internal DRAM Input Clock and the Delayed Feedback Clock at the phase detector is some value other than zero. The phase detector begins to make adjustments to the variable delay line based on the phase alignment of the two input clock signals. As the phase alignment of the input clocks to the phase detector approach 0 degrees, the phase detector stops making adjustments to the variable delay line and the DLL is said to be locked.

The output of the delay line is then fed to a clock distribution network; or, what is commonly referred to as a "clock tree." The outputs of the clock tree are distributed to the output data latches that serve the purpose of synchronizing the data to the clock before the data is driven to the DRAM external environment. In Figure A.2, one of the clocks distributed by the clock tree is routed back through a block labeled "Input/Output Delay Model." The feedback clock is delayed by this logic block and fed back to the second input to the phase detector.

The Input/Output Delay Model (I/O model) is the part of the DLL circuit that helps the data achieve alignment with the external DRAM clock. When the clock that is used to clock the data output latches is fed through this delay block, the phase detector is forced to remove delay from the Variable Delay Line to compensate for the added delay to the feedback clock. The phase detector must remove the same amount of delay from the variable delay line as is added by the Input/Output Delay Model in order to achieve 0 degrees of phase difference at the phase detector inputs.

A.3 Simple DLL Timing Theory

Now we will establish some simple mathematical timing relationships that prove the theory of the DLL operation described above. What we need to prove is that the clock that strobes the data output latches is back-timed relative to the external clock by the sum of the delay through the data output latch and the DQ pad driver.

First, let us follow the clock signal from the input through to the output clock distribution network.

$$t_{CLKOUT} = t_{IN} + t_{DELAYLINE} + t_{TREE} \quad \textbf{(A.1)}$$

where t_{IN} is the input clock delay, $t_{DELAYLINE}$ is the delay through the variable delay line and t_{TREE} is the delay through the clock distribution delay.

Now suppose the clock tree and the I/O model are removed from the feedback path. For the phase detector to achieve phase alignment at its inputs, the delay line would have to be one clock cycle in length. We will define a clock cycle as t_{CK}. Therefore, the variable delay line would have a delay equal to t_{CK}. When we add the clock tree and IO model delay back into the loop, the phase detector will force the delay line to reduce the delay by an amount equal to the delay added to the loop. The expression for the delay through the delay line now becomes:

$$t_{DELAYLINE} = t_{CK} - t_{TREE} - (t_{IN} + t_{DQ}) \quad \textbf{(A.2)}$$

where t_{IN} is the input clock distribution delay from the clock pad to the delay line and t_{DQ} is the output data path delay through the data output latch and the DQ pad driver. Combining Equations A.1 and A.2 gives us the following:

$$t_{CLKOUT} = t_{CK} - t_{DQ} \qquad \textbf{(A.3)}$$

We now see that the output clock is back timed at the data output latch by the amount of delay through the data output latch and DQ pad driver, t_{DQ}. This means that when the clock from the output of the clock tree clocks the data output latch, the data driven from the pad driver is delayed by an even number of clock cycles and is aligned to the external clock.