DESIGN AND IMPLEMENTATION OF FIRST IN FIRST OUT MEMORY

A Thesis
submitted towards the partial fulfillment of requirements
for the award of the degree of

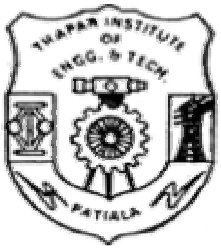
Master of Technology
in
(VLSI Design & CAD)

Submitted by

DHIRAJ PUNJ
Registration No - 6040406

Under the Guidance of

Mr. Sanjay Sharma
Assistant Professor



Department of Electronics and Communication Engineering
THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY,
(Deemed University), PATIALA – 147004, INDIA

June, 2006

DECLARATION

This is to certify that the thesis titled, "Design and Implementation of First in First Out Memory": is an authentic record of my own work carried out as requirements for the award of degree of Master of Technology in VLSI Design & CAD at Thapar Institute of Engineering & Technology (Deemed University), Patiala (Punjab), under the guidance of **Mr. Sanjay Sharma, Assistant Professor (TIET, Patiala)**, during July, 2005 to June, 2006.

Dhiraj Punj

Date: _____

Roll No. - 6040406

Sanjay Sharma
Assistant Professor,
ECE Department,
Thapar Institute of Engg. & Tech.
Patiala (Punjab), India.

Countersigned By:-

Head, ECE Department,
Thapar Institute of Engg. & Tech.
Patiala (Punjab), India.

Dean of Academic Affairs,
Thapar Institute of Engg. & Tech.
Patiala (Punjab), India.

Acknowledgments

I would like to take this opportunity to thank everyone who contributed to the successful completion of this thesis. First of all, I would like to thank Mr Sanjay Sharma, my guide, for providing extensive support and encouragement throughout this work.

I would like to thank my parents and to all my family members and friends for their constant prayers and moral support throughout my life. I deeply thank my mother for her efforts to make me successful.

I also thank my friends Harcharan Singh and Tejinder Singh for their help in my work and all my other friends, for their help and encouragement at times I needed them. I also thank all my friends at Thapar Institute of Engineering and Technology who helped me indirectly in my work.

Last of all I thank GOD for keeping me in good health and spirits throughout my stay at Thapar Institute of Engineering and Technology, Patiala.

ABSTRACT

A digital signal processor includes a computation block with an arithmetic logic unit, a multiplier, a shifter and a register file. The computation block includes a plurality of registers for storing instructions and operands in a bit format as a continuous bit stream, and utilizes a bit/byte transfer mechanism for transferring in a single cycle a bit field of an arbitrary bit length between the plurality of registers and the shifter. The plurality of registers may be general purpose registers located in the register file or a FIFO.

FIFO is also extensively used in ***iterative computations in digital signal processing*** *like the FFT, on an input data record. A FIFO is also used for storing* ***resampled digital samples*** *for readout at a clock rate which follows the time-averaged horizontal frequency of the received digitized television signal. A color television signal,* ***digitized by a sampling clock asynchronous*** *with respect to the television signal's horizontal synchronizing pulses, is applied to a dynamically programmable digital filter which upsamples or downsamples the received digitized television signal over short time periods in response to time-base disturbances in the signal. The user may also select a higher or lower long-term clock frequency.*

FIFOs also find use in packet switching techniques. Because of the unscheduled nature of arrivals to a packet switch, two or more packets may arrive on different inputs destined for the same output. The switch architecture may allow one of these packets to pass through to the output, but the others must be queued for later transmission. the most common use of FIFOs is in the interfacing of two clock domains operating at different frequencies.

Table of Contents

CHAPTER 4

DESIGN OF SELF ADDRESSING FIFO

CHAPTER 5

A FASTER FIFO USING LFSR

List of figures

List of tables

CHAPTER 1
INTRODUCTION

1.1 Introduction

The need for large data transfer from/to mass storage in multiprocessing and data communication fields has become more critical in recent years. Data transfer speed is becoming a bottleneck in these systems and the necessity for large data buffers is widely recognized. Mainframe systems have started to implement expanded storage or semiconductor disks within the system to improve its speed performance. Since the system simplicity is an essential factor in the expanded storage, FIFO memory is believed to be one of the best configurations. Simple controllability also makes FIFO memory most suitable for consumer products. The FIFO concept, for example, can be well adapted to the field/frame memory in TV/VCR or other video display systems. With the progress of digital video processing technology, these memories are now widely in demand. **TV/VCRs** with high picture quality or other features have been produced by employing FIFO technologies.

FIFOs also find good use in packet switching techniques. In packet FIFO, messages arrive asynchronously from a number of sources. Each message is divided, as it comes in, into packets of some maximum length. The packets are collected in a single FIFO queue from which they are transmitted over the trunk. Because of the unscheduled nature of arrivals to a packet switch, two or more packets may arrive on different inputs destined for the same output. The switch architecture may allow one of these packets to pass through to the output, but the others must be queued for later transmission. We study the performance of four different approaches

1.2 Motivation

Many networking/computing applications require high speed switching for multicast traffic at the switch/router level to save network bandwidth. However, existing queuing-based packet switches and scheduling algorithms cannot perform well under multicast traffic. While the speedup requirement makes the output queued switch difficult to scale, the single input queued switch suffers from head of line (HOL) blocking, which severely limits the network throughput. The various multicast scheduling algorithms use FIFO buffers.

1.3 Organization of Thesis

This thesis is organized in the following manner.

Chapter 2 includes an introduction to FIFOs and discussion on applications like multiple clock domain interface.

Chapter 3 discusses asynchronous and synchronous FIFO styles. This chapter also discusses the read and write counters used.

Chapter 4 explains the Design of Self Addressing FIFO.

Chapter 5 deals with First in First out using LFSR.

CHAPTER 2
INTRODUCTION TO FIRST IN FIRST OUT MEMORY

2.1 FIRST IN FIRST OUT Memory

A First-In, First-Out (FIFO) memory is a read/write device that automatically keeps track of the order in which data is entered into the memory and reads the data out in the same order. The memory functions like a parallel-in parallel-out register whose length is always exactly equal to the number of words stored.

The most common application of a FIFO is as a buffer memory between two digital devices operating at different speeds. Even when two devices operate at the same data rate, it is not always possible for both to be operated synchronously. The FIFO provides the necessary data buffering to achieve synchronization, which is a requirement for many signal processing systems. It has been shown that a FIFO memory can be used for data shuffling during the computation of Fast Fourier Transforms (FFT) and a FIFO can be utilized in array processor structures.

A problem associated with the use of FIFO memory is data latency, that is, the time for the data to ripple through the FIFO's stages. If the FIFO is interfaced with a processor, the processor's throughput can be lowered if the FIFO's data latency is higher than the data processing time. There are two cases where this situation can develop. The first is when the processor performs an operation and requests new input data faster than the time it takes for the data to ripple through one FIFO stage. The second and more common event, is when the processor performs iterative computations (i.e., FFT) on an input data record of N samples provided by a FIFO with R stages, where R > N.

The first case can only be accommodated by designing the FIFO with a data ripple rate faster than the computational rate of the processor. A solution to the problem imposed by the second case is offered here by designing a FIFO with programmable length. There are two implications due to this capability. First, the same memory can be used for signal processing applications with different input requirements without experiencing data latency; and second, if only part of the memory length is utilized, the power consumption is lower because the unused portion is not clocked and, therefore, does not dissipate dynamic power.

A very important aspect of the FIFO architecture is the design for testability scheme used to ease the functional testing burden at minimal cost, and at the same time used to give some self-testing capability to the FIFO. The signals generated during self-testing are available output signals and they can be used to achieve fault tolerance at the system level.

2.2 The FIFO Design

The logic design of the FIFO is shown in Figure. 2.1. All the signal terms used in the discussion concerning the FIFO design are illustrated in this figure. The C, (i = 1; . . ,128) signals are provided by the decoder which is driven by the address specifying the desirable FIFO length for a given application. The decoder also drives the “bit-set” circuit that provides the S, signals which only enable the operation of the FIFO control within the selected FIFO length[1]. Figure.2.2 illustrates the generation of these signals.

The length of the FIFO is selected before data is stored in the FIFO. Data words are stored in 128 eight-bit registers connected so the output of one feeds the input of the next. The operation of the FIFO is performed by clocking each register independently so

that data can be selectively shifted through the registers. Each register shifts independently based on the output of the cross- coupled NOR gates associated with each register which determine whether or not that register contains valid data (Figure. 2.1). Initially the FIFO is reset and there is no data stored in it. The FULL(j) (j = l,..., 128) bits are all reset to “0”. When the LOADEN signal becomes “l”, an S-bit data word can be entered into the first register and the FULL-O bit is set to “l”, indicating that valid data is present in the first register. The FULL-l bit of the second register is “0” and this causes it to continually monitor the FULL-O bit of the first register looking for a “1”. When the data

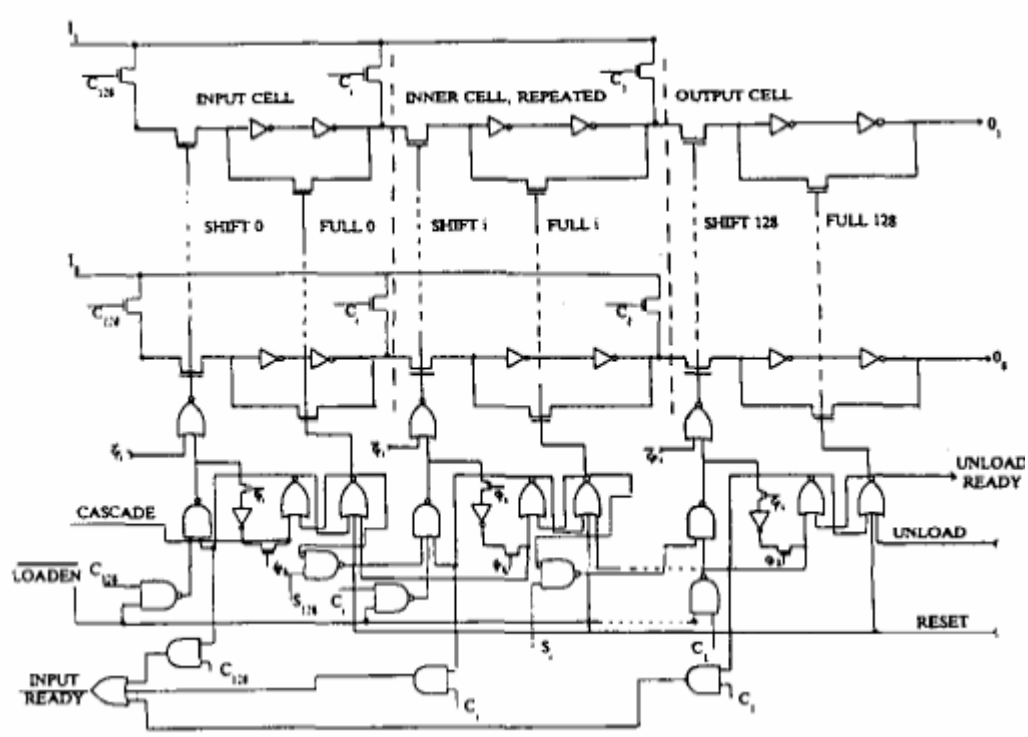


Fig 2.1 Logic design of FIFO storage and Control Section

is stored in the first register the control sees the “1” and generates a SHIFT-l pulse which shifts the data from the first register into the second, sets the FULL-l bit to “1” and resets the FULL-O bit. The same process is repeated until the data arrives at the 128th register.

At this point only FULL-128 (OUTPUT READY) is set to “l", the others having all been reset as the data was shifted into the next register.As soon as the data moves from the first register to the second, the FULL-O bit is reset ‘to “0”. A new data word can now be shifted into the first register. The new data shifts through the registers as long as their FULL bits are “0”. Eventually the data reaches the

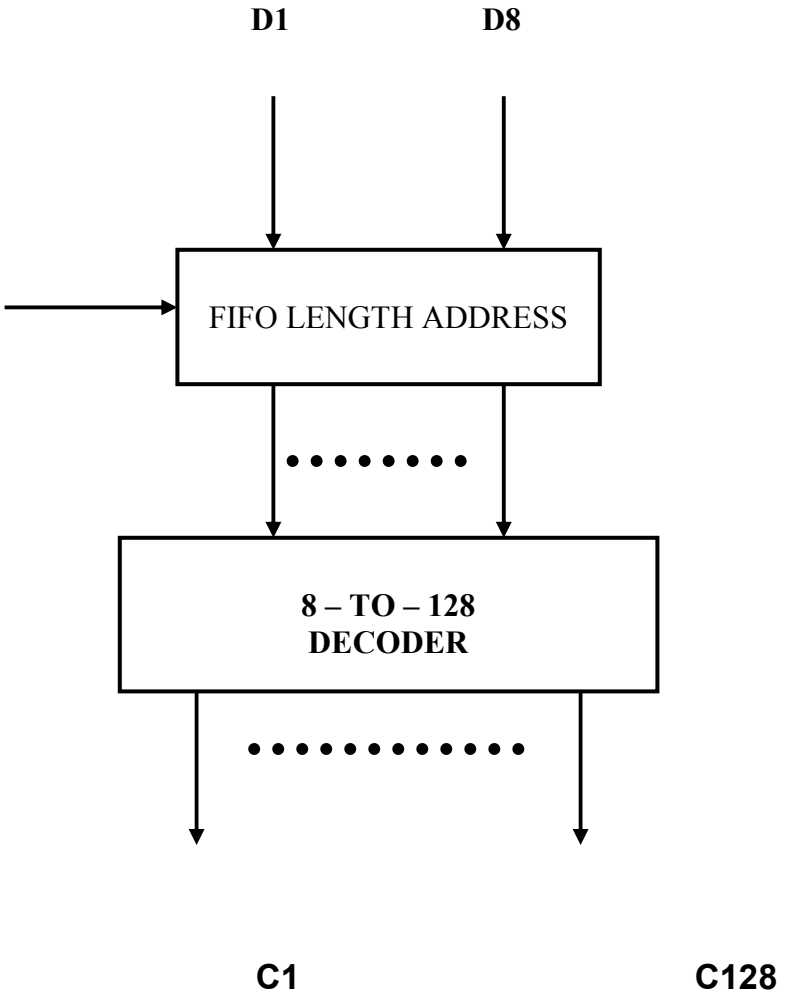


Fig 2.2 The generation of control signals

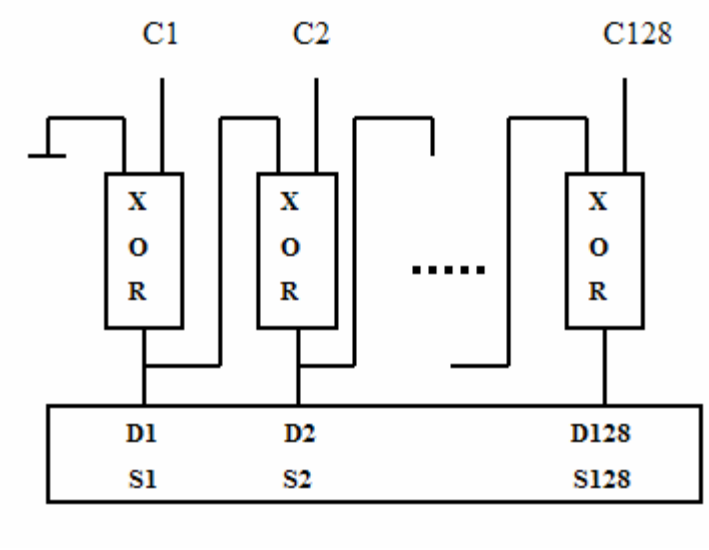


Fig 2.2 The generation of control signals

register immediately preceding the one containing the data, stores itself in that register since no further shifting is possible , and the process is repeated until all data words are entered. When the unload line on the output goes "HIGH" it cause full 128 bit to reset indicating that the 28th register is empty. The next to the last register is shifted into the last register and the "0" on the FULL-128 line (OUTPUT READY) moves back toward the FULL-O as the data words move down one register. This process can continue until all data has been shifted out of the FIFO. When the last word has been read, the FULL-128 (OUTPUT READY) bit remains "0" indicating that there is no data available at the output.

This scheme allows the reading and writing of data to occur completely independently. Data can be written into the FIFO as rapidly as one write per two cycles, after the LOADEN line goes "high". The amount of time required for the first data word to ripple through the registers has been defined as data latency. The data latency can be computed by multiplying the clock period by the shift register stages. Since the length of the FIFO is programmable, the data latency is minimized for applications needing less than 128 input data samples. In addition to the control signals required for the FIFO's operation, the SHIFT-O (INPUT READY), FULL-128 (OUTUNLOAD PUT READY) and CASCADE signals are also provided as outputs along with the +2 phase of the clock, thus making the synchronization and interfacing of multiple FIFO chips simple, therefore allowing the configuration of FIFO memories of any size. FIFO chips can be cascaded to obtain memories longer than 128 stages. In the case of two chips, FIFO-l can be programmed to any length, while FIFO-2 is programmed to its maximum length. The CASCADE output of FIFO-l and the INPUT READY output of FIFO-2 are not used.

Multiple clock domain interfacing

without alteration of the local clock.

2.3.1 Introduction

Many of the methods used to interface clock domains within an integrated circuit use an asynchronous wrapper around synchronous blocks which alters the local clock of the synchronous logic[2]. The purposes of this local clock control are to avoid metastability at the asynchronous-synchronous interface and to provide flow control. A potential problem with this approach is that altering the local clock may cause problems with synchronous logic blocks that employ dynamic logic. It is increasingly desirable to reuse intellectual property (IP) without modification in the design of large integrated circuits in order to reduce time-to-market and reduce design costs. Altering the local clock is at odds with this goal.

The probability of synchronization failure could be made negligibly small using a string of flip-flops tied to the receiving synchronous block's local clock, but at the cost of high latency[3]. Synchronizing flip-flops will be avoided in the design presented here by altering the speed of data flow through the asynchronous interface instead. It is assumed here that flow control is accomplished already by the synchronous logic block, as would be the case if the clock domain boundary was at a multi-master bus interface. To achieve low latency, a modified asynchronous first-in first-out (FIFO) circuit of can be used [4]. One variety of this FIFO can be stopped and restarted without introducing metastability.

2.3.2 Interface design

In Figure. 2.3 a synchronous block of logic in one clock domain (*sclk*) on the left is sending data to a synchronous block of logic in another clock domain (*rclk*) on the right. Both synchronous logic blocks are assumed to capture data on the rising edge of the local clock, but with the addition of an inverter either block could be negative-edge triggered.

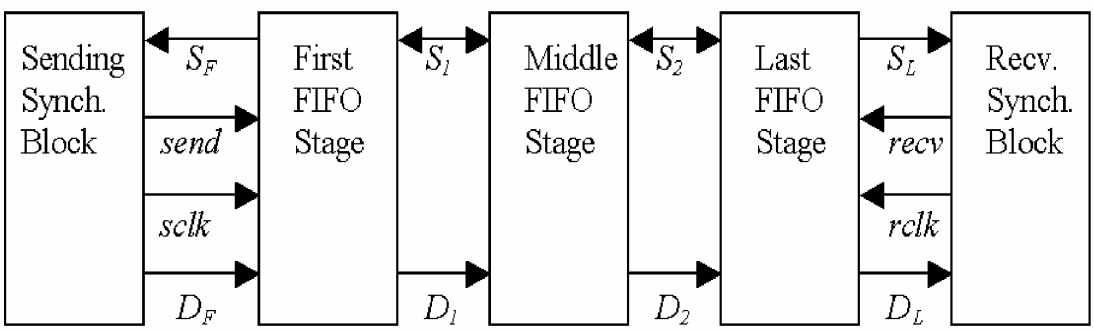


Fig 2.3 Asynchronous FIFO passing Data between Two Synchronous Clock Domains

The data path (*D*) is of arbitrary width in this bundled data system. For wide data paths, care needs to be taken that all of the data bits can forward through the last FIFO stage in the amount of time available during the low portion of the receiving local clock. The interface is shown with three stages of asynchronous FIFO, but the middle stage can be replicated as many times as necessary to lengthen this FIFO.

The local clocks are fed to the first and last FIFO stages in order to avoid metastability. The empty/full FIFO states at the FIFO ends (S_F and S_L) are only allowed to change during the time that the attached clock is high. The setup time for the synchronous storage elements with inputs S_F and S_L comes just before the rising edge of their respective clocks, so if they are not allowed to change when the local clock is low, metastability can not occur. This shifts the burden of synchronization onto the first and last FIFO stages. Also, by not allowing the last FIFO stage to accept new data when it's local clock is low, enough time is allowed for data for forward through the last FIFO stage and the bundled data constraint is maintained (if the data path is not too wide).

The empty/full state in a FIFO (*S*) is a bi-directional signal which serves as both the request and acknowledge. In the variety of used here, a high voltage represents full and a low voltage empty. If a stage is empty (right state conductor is low) and the preceding stage is full (left state conductor is high), then the stage momentarily shorts the left state conductor to Vss and the right state conductor to Vdd. At the same time a short duration positive pulse is sent to the data path portion of the FIFO stage to forward the data from the preceding *D* to the following *D*. In order to place a new data item in the first FIFO stage, the sending synchronous block should test that S_F is low and both set *send* high and place data on D_F if S_F is low. Figure. 2.4 shows how the first FIFO stage sets the predecessor state conductor to full on the rising edge of *sclk*. The read pulse generator in the last FIFO stage is very similar, with the PMOS pull-up transistor replaced by an NMOS pull-down, the NAND replaced with an AND, and *rclk/recv* replacing *sclk/send*.

Since the empty/full state conductor is bi-directional, it is held with a keeper circuit. The pull-up or pull-down transistors in Figure. 2.4 must be able to overpower the weaker

feedback inverter from this keeper. The datapath (one copy of Figure. 2.5 per bit of the datapath) is unidirectional as enforced by the output inverter. This inverter must be able to overpower the weaker feedback inverter of the latch in the following stage. Data forwarding is controlled by a positive pulse on *P* which momentarily allows data to pass.

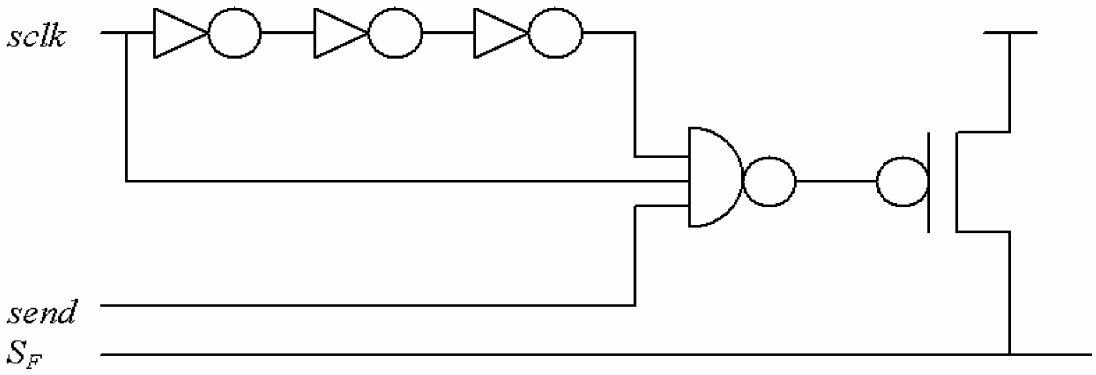


Fig 2.4 Write Pulse Generator in the First FIFO Stage

There are two versions of the FIFO presented by Sutherland and Fairbanks [3]. They are both very fast, but the version not used here is slightly faster than the one chosen for this application. The slower version was chosen since it allowed a FIFO stage to be halted without introducing metastability. The data control pulse (*P*) is also slightly longer in duration for this version, so there is less likelihood that the control pulse will get smeared out as it progresses down the transmission line formed by the data path

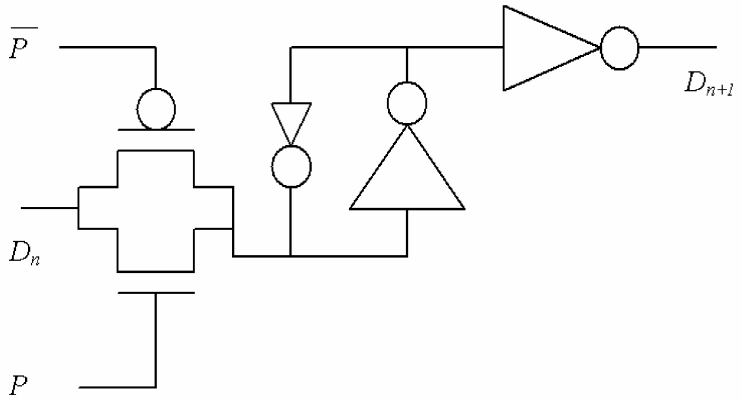


Fig 2.5 Fifo Datapath Element

control pass gates. The original GasP design used only NMOS pass transistors, but full pass gates are required for 1.8 V operation. The portion of the FIFO stage which produces the data control pulse (*P*), resets the predecessor state (S_n) to empty, and sets the successor state (S_{n+1}) to full is shown in Figure. 2.6. The middle stages of the FIFO are similar, except that there is no clock input, the inverter following the clock input is removed, and the cross-coupled NAND gates are replaced with a single inverter. It is not

possible to use the faster style for the middle stages since the faster circuits use opposite state encoding (full low and empty high) and the timing mismatch may not result in correct FIFO operation.

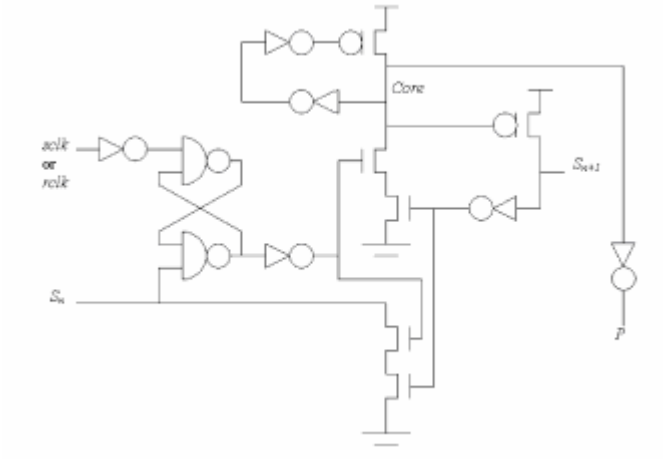


Fig 2.6 Control for First and Last FIFO Stages

The design used a self reset circuit for the data control pulse (*P*) similar to that used for the *Core* node in Figure. 2.6. This allowed *P* to be stretched to compensate for the data path load. For very wide datapaths, a string of inverters is required to obtain the necessary current amplification to drive all of the pass gate inputs. Widening of pulse *P* can be accomplished more easily by using asymmetric inverters in this case. Since this interface is envisioned to pass a system bus across clock domain boundaries in a shared-memory single-chip multiprocessor, the datapath is expected to be wide.

CHAPTER 3
SYNCHRONOUS AND ASYNCHRONOUS FIFO DESIGNS

This chapter describes RAM-based FIFO designs using the dual-port RAM. Synchronous designs with a common read/write clock are described, as well as asynchronous designs with independent read and write clocks will be discussed. Emphasis is on the fast, efficient and reliable generation of the handshake signals FULL and EMPTY, which determine design performance.

3.1 Introduction

Many FIFO designs use the distributed RAM feature to implement First-In-First-Out (FIFO) elastic buffers to form a bridge between subsystems with different clock rates and access requirements. The non-synchronous nature of the single-port RAM confronts the designer with several challenges. Addresses must be multiplexed, independent read and write clocks must be synchronized, and access requests must be arbitrated. The improved synchronous dual-port RAM, solves most of these problems. Since the basic RAM in each CLB has independent write and read addresses, there is no need to multiplex addresses and arbitrate their selection. The synchronous write mechanism simplifies write timing and contributes to much faster operation. The FIFO design effort can now be concentrated on achieving high throughput and low cost, and on solving the fundamental timing problems created by asynchronous read and write clocks. This chapter describes several design examples:

Three different FIFO depths, each with either a common clock (synchronous operation) or with two unsynchronized clocks (asynchronous operation). The first design is a 16 x 16 FIFO where the depth of the basic CLB-RAM is sufficient. This leads to a very fast

and efficient implementation that can run at, or close to, the maximum write speed, even for simultaneous read and write operations.

The second design example is a 32 x 8 FIFO (32 deep, 8 bits wide) that requires input and output data multiplexing between two RAM banks. The address counters are longer than for the 16-deep FIFO, and the control logic generating FULL and EMPTY is more complex, with one additional layer of logic. This slows down operation, and 40 MHz simultaneous asynchronous read and write may be the maximum performance in an XC4000E-3 device. The third design example is a 64 x 8 FIFO (64 deep, 8 bits wide) that requires input and output data multiplexing between four RAM banks.

3.2 Free-Running Read and Write Clocks

These designs assume free-running clocks, activated by their respective enable signals. Without a free-running Read Clock, the asynchronous designs would lock up with an active EMPTY(STRETCHED) output, which can only be terminated by a High level on Read Clock. If this clock is not free-running, the EMPTY(STRETCHED) output stops the external decision-making logic from making Read Clock go High [7].

EMPTY(STRETCHED), therefore, stays active, even after data has been written into the FIFO. FULL(STRETCHED) would behave similarly without a free running Write Clock. Free-running clocks, activated by their respective enable signals, avoid these problems.

3.3 16 x 16 FIFO with Common Clock

This example implements a 16 x 16 FIFO with a common read/write clock and individual read and write clock enables. This is the simplest and fastest design, since it avoids the more challenging issues of asynchronous clocking. Figure 3.1, taken from the XC4000 Series data sheet, shows the basic dual-port 16 x 1 RAM that can be implemented in

*each CLB. Writing is synchronous. Write Enable, Write Address, and Write Data must meet the documented set-up time with respect to the Write Clock. Reading is asynchronous, controlled only by the Read Address. The two unused CLB flip-*flops have uncommitted data inputs and Q outputs, but share the clock signal (although not the clock polarity) with the write port [8].

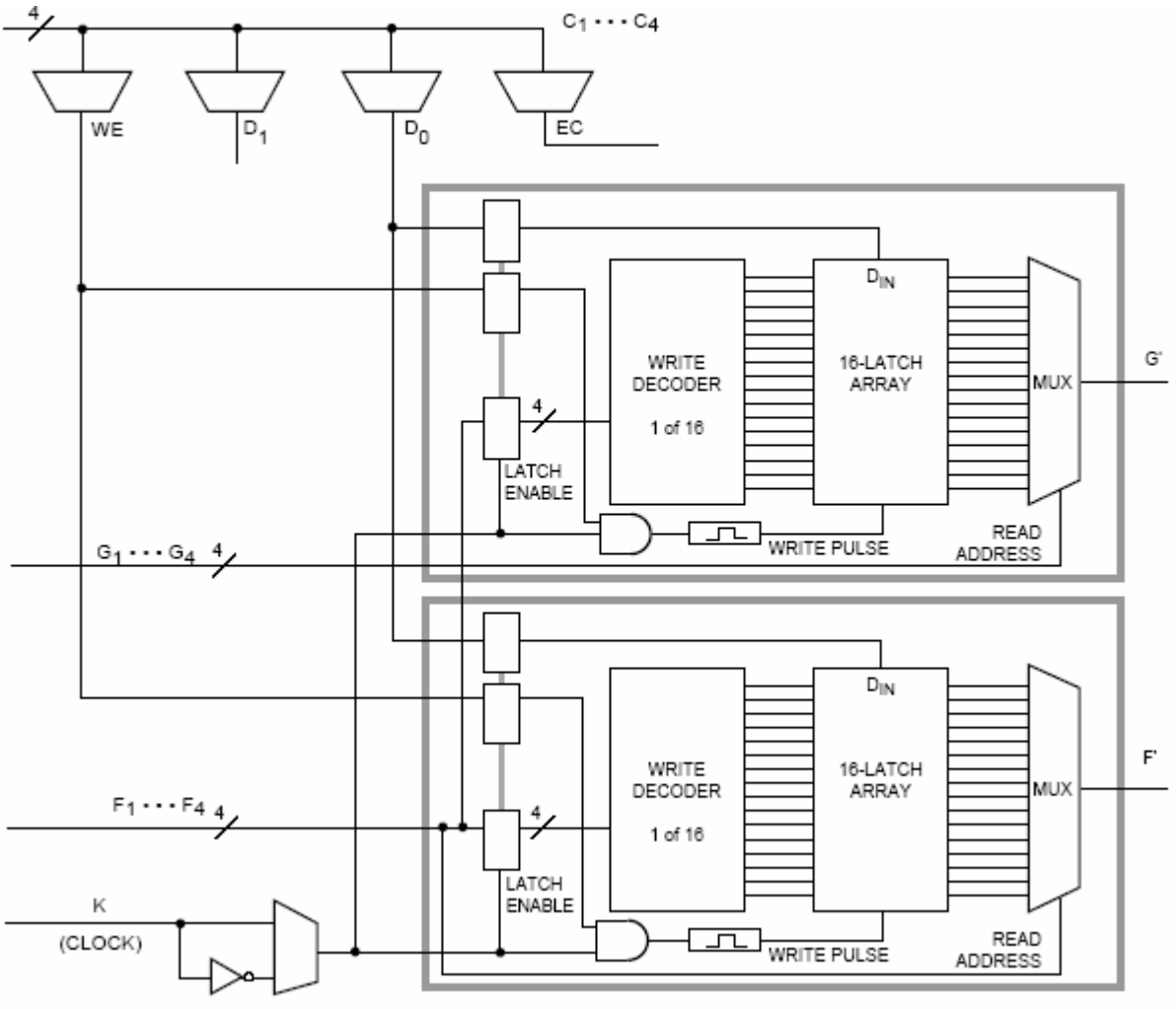


Fig 3.1 : Edge Triggered RAM

Figure 3.2 shows the basic block diagram of the 16 x 16 FIFO. In the synchronous version of this design, read and write clock are identical. The 4-bit read counter and the 4-bit write counter, shown in Figure 3.3, are each implemented as two cascaded 2-bit Gray or Johnson counters. In a fully synchronous design, this choice is not mandatory, but it has advantages in the non-synchronous implementation. It is, however, mandatory that the upper two bits always stay constant for four consecutive counts. Four-bit (LFSR) counters can, therefore, not be used, as will soon become apparent.

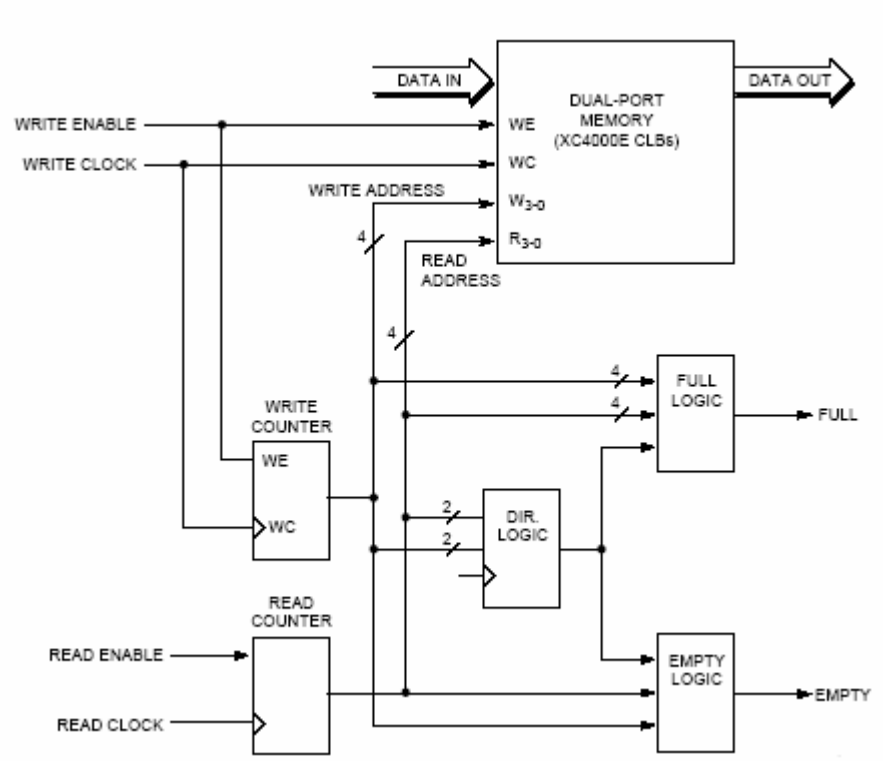


Fig 3.2 : 16 X 16 FIFO Block Diagram

The two 4-bit counters address the RAM in the conventional way. Seen as a “black box”, the FIFO behaves like an elastic shift register: Input Data is accepted by the Write Clock when Write Enable was High during the set-up time before the active Write Clock edge. Output Data is always available at the output port, and is substituted by the next Data, after the Read Clock was enabled by a Read Enable High during the Read Clock set-up time.FULL and EMPTY outputs must be interpreted by external logic to prevent a Write

operation during FULL, or a Read operation during EMPTY. Most of the design effort is spent on the control logic, shown in Figure 3.4, that detects the two abnormal conditions, FULL and EMPTY.

When the FIFO contains 16 words that have not yet been read, the FULL flag must be activated, and further write operations must be avoided. When all words written into the FIFO have been read, the EMPTY flag must be activated, and further read operations must be avoided. When all words written into the FIFO have been read, the EMPTY flag

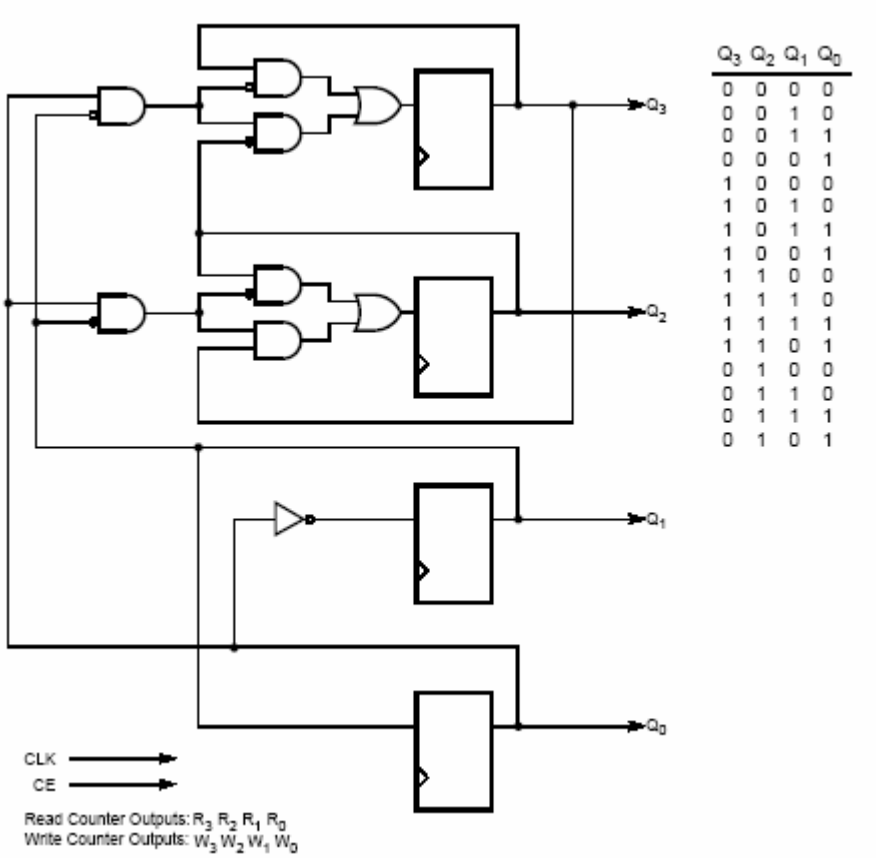


Fig 3.3 : 16-deep FIFO, Read Counter or Write Counter

must be activated, and further read operations must be avoided. Unfortunately, the easily decoded signal for these two abnormal conditions is the same: read address is identical with write address. An additional signal must be created that distinguishes between the

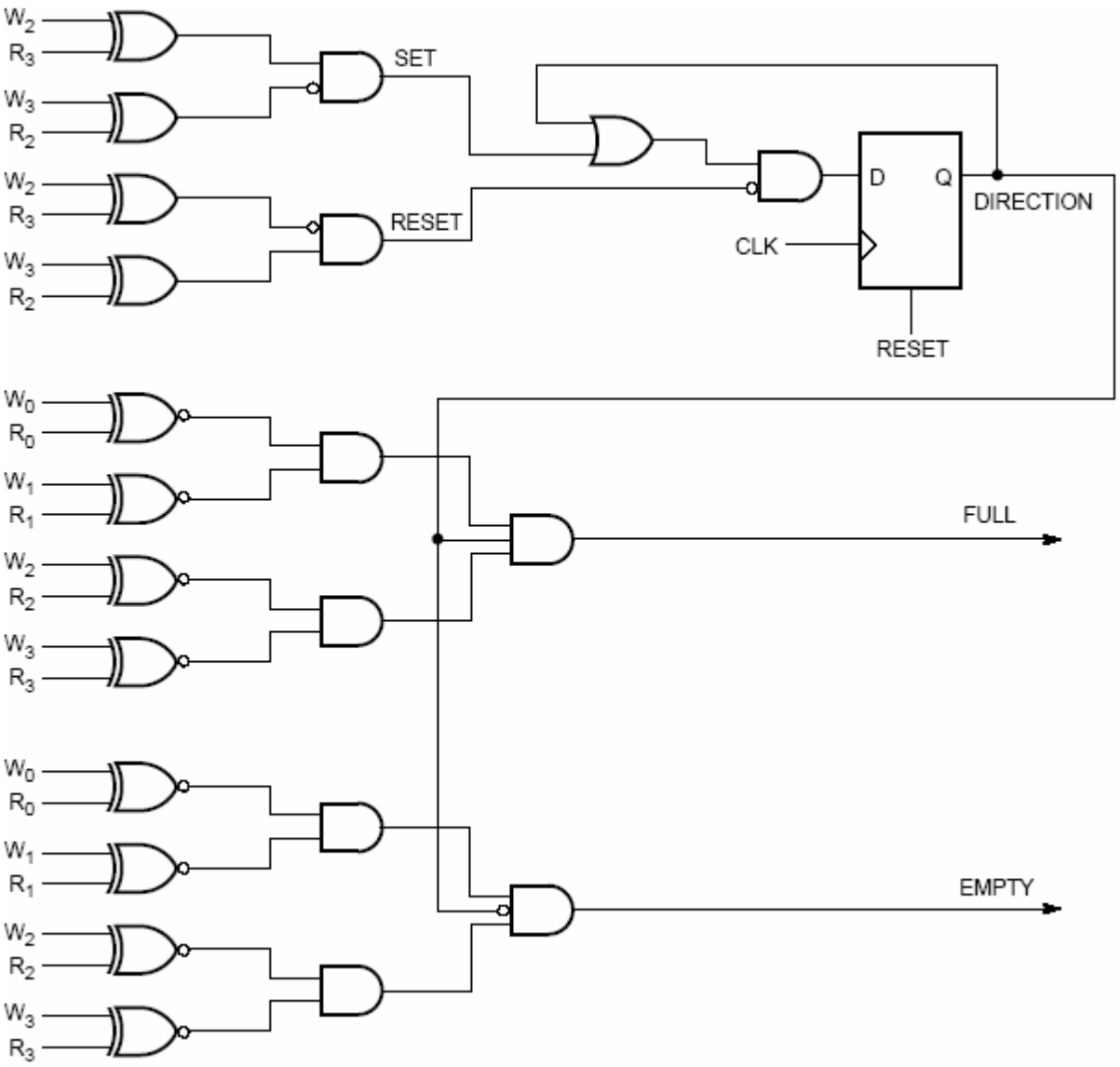


Fig 3.4 : 16-deep FIFO Synchronous Control

two very different conditions of FULL and EMPTY. For this purpose, an auxiliary signal called DIRECTION is created to indicate whether the Write counter is about to catch up with the Read counter, or whether the Read counter is about to catch up with the Write counter. The two most significant bits of both counters are compared, since they indicate in which quadrant of the 16-position circular address space the present address resides. These two most significant bits of both address counters together are used to address two

4-input look-up tables in parallel. The look-up tables (LUTs) [9][10]decode the relative quadrant position of the two counters. The 4-bit LUT address describes one of 16 possible conditions:

• Four addresses describe the situation where the write counter is in the quadrant immediately behind the read counter. This is decoded as a “possibly going full” condition, and sets the DIRECTION latch or flip-flop.

• Another four addresses describe the situation where the write counter is in the quadrant immediately ahead of the read counter. This is decoded as a “possibly going empty” condition, and it resets the DIRECTION latch or flip-flop.

• Four other addresses indicate that the two counters are in the same quadrant, and another four addresses indicate that the two counters are in opposite quadrants. These eight addresses provide no useful information about the relative address position, and thus do not affect DIRECTION.

Note that DIRECTION must start in the reset state when the FIFO is initiated with both counters at zero. DIRECTION is thus established well before the actual FULL or EMPTY condition can occur. There will be at least four, and usually many more, consecutive set or reset inputs to the DIRECTION latch or flip-flop before it is being used to discriminate between FULL or EMPTY.

FULL goes active as a result of the write clock edge that writes data into the last available location. FULL goes inactive as a result of the first read clock that reads one word out of the previously full FIFO. EMPTY goes active as a result of the read clock edge that reads the last available data from the FIFO. EMPTY goes inactive as a result of the first write clock that writes one word into the previously empty FIFO. In a synchronous design, FULL and EMPTY are synchronous control signals, to be used appropriately by the logic external to the FIFO.

3.4 16 x 16 FIFO with Independent Clocks

This example adapts the 16 x 16 FIFO for use with independent read and write clocks. An asynchronous design with separate and unrelated read and write clocks poses additional problems. The RAM array, the counters, and part of the control logic are unaffected, but the DIRECTION, FULL and EMPTY signals require additional attention, as shown in Figure 3.5. There is no common clock; DIRECTION therefore uses a latch, implemented as a combinatorial circuit in a separate CLB. Since the counter bits that determine DIRECTION are Gray-coded, with only one bit changing per transition, the decoding is guaranteed to be glitch-free. Initialization reset is provided by a flip-flop that is set by the first write operation. FULL goes active as a synchronous response to the write clock, but FULL goes inactive as a result of the read clock, asynchronous to the write clock. Since FULL is used only by the write control logic, there is no synchronization problem when FULL goes active, but there is when FULL goes inactive. The easiest solution is to stretch the FULL signal such that it cannot go inactive during the half-period of the write clock immediately preceding the active clock edge.

Figure 3.5 shows this circuit, assuming a rising clock edge. The FULL output can go inactive only while the write clock is High, which gives the write logic enough set-up time to the following rising clock edge. There is still the possibility of metastable confusion if the full condition goes inactive in a very narrow timing window when the latch is about to latch up, i.e.-right after the falling edge of Write Clock. In most cases, this metastable disturbance will have settled well before the next rising clock edge, but if the user is concerned about this low-probability risk, FULL can be stretched by a complete write clock period, which will reduce the likelihood of metastable failure to an insignificant level. Figure 3.5 also shows this alternate design, using a flip-flop clocked by the Write Clock, generating a synchronous falling edge of the FULL signal.

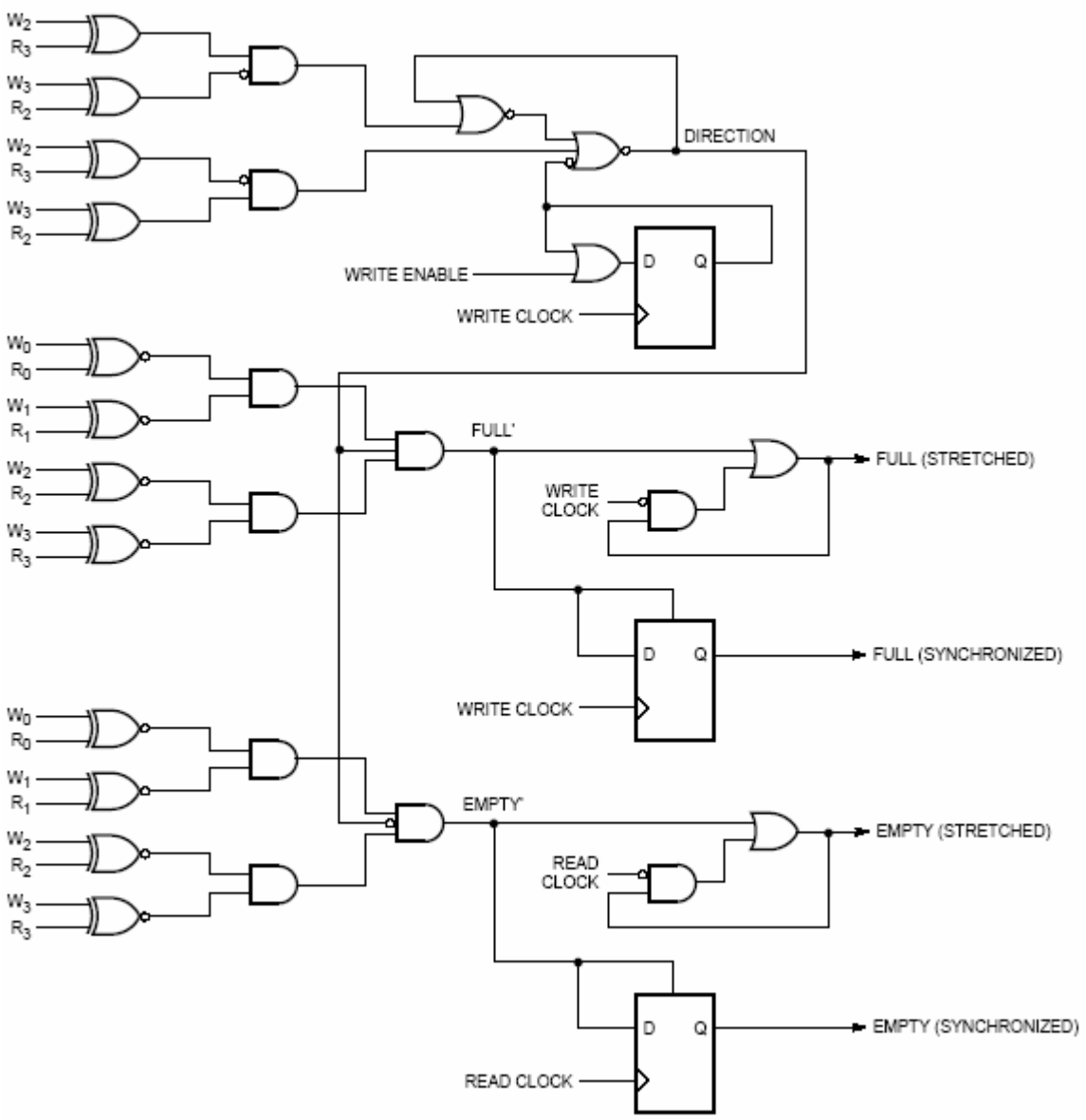


Fig 3.5 16 bit deep FIFO Asynchronous Control

EMPTY is generated by a similar circuit that prevents EMPTY from going inactive while the read clock is Low, or synchronizes it with the other edge of the READ clock. FULL and EMPTY flags are delayed by an extra level of logic, compared to the fully synchronous design. The stretcher circuit adds 2 ns, the alternate flip-flop adds 4 ns, thus reducing peak performance to 58 or 50 MHz, respectively.[9]

3.5 32 x 8 FIFO

Here we implement a 32 x 8 FIFO with independent read and write clocks. Since each CLB can only implement a 16 x 1 dual-port RAM, the 32-deep FIFO uses two memory banks. Longlines distribute data to and from the RAMs. For the write port, WE gates a 1-of-2 decoder. For the read port, TBUFs are used to multiplex the data onto Longlines, as shown in Figure 3.6. The 5-bit read and write address counters each consist of a 3-bit (8 Linear-Feedback-Shift-Register (LFSR) counter followed by a 2-bit Grey or Johnson counter. (See Figure 3.7.) The latter drives the 1-of-2 decoders selecting between memory banks. The DIRECTION detector decodes the quadrant information and generates set and

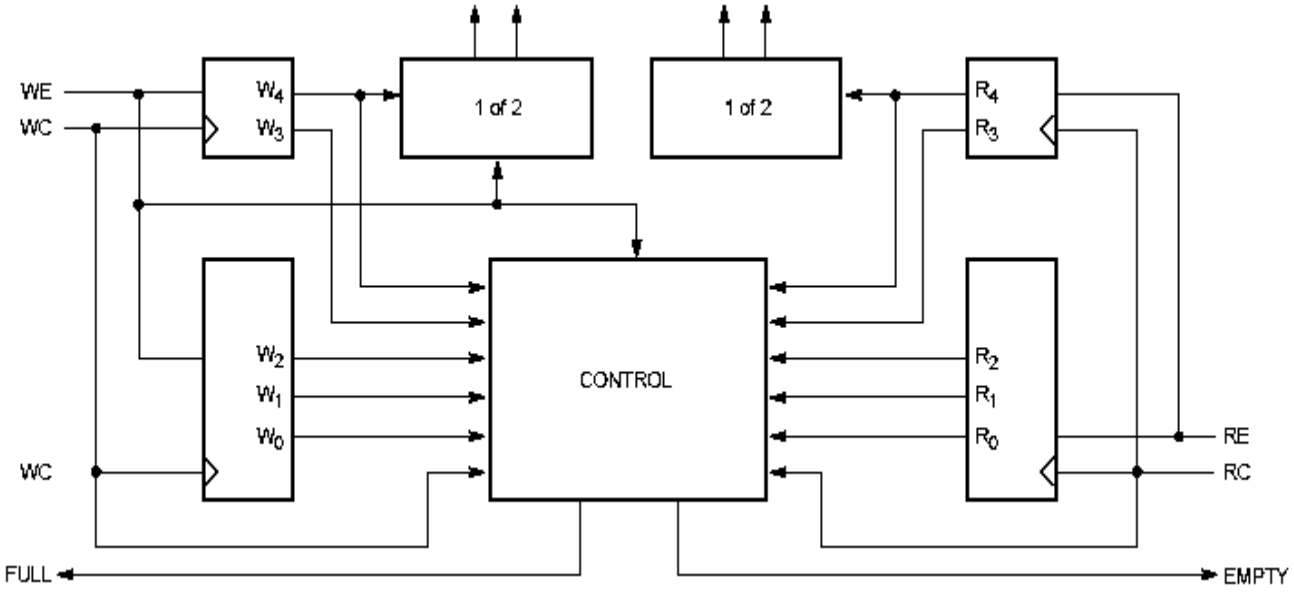


Fig 3.6 : 32 X 8 FIFO Block Diagram

reset signals for the DIRECTION flip-flop or latch. When both counters are identical, either a FULL or EMPTY output flag is generated, depending on the state of DIRECTION. Asynchronous control logic for a 32-deep FIFO is shown in Figure 3.8. For a more detailed description, see the earlier 16 x 16 FIFO section.

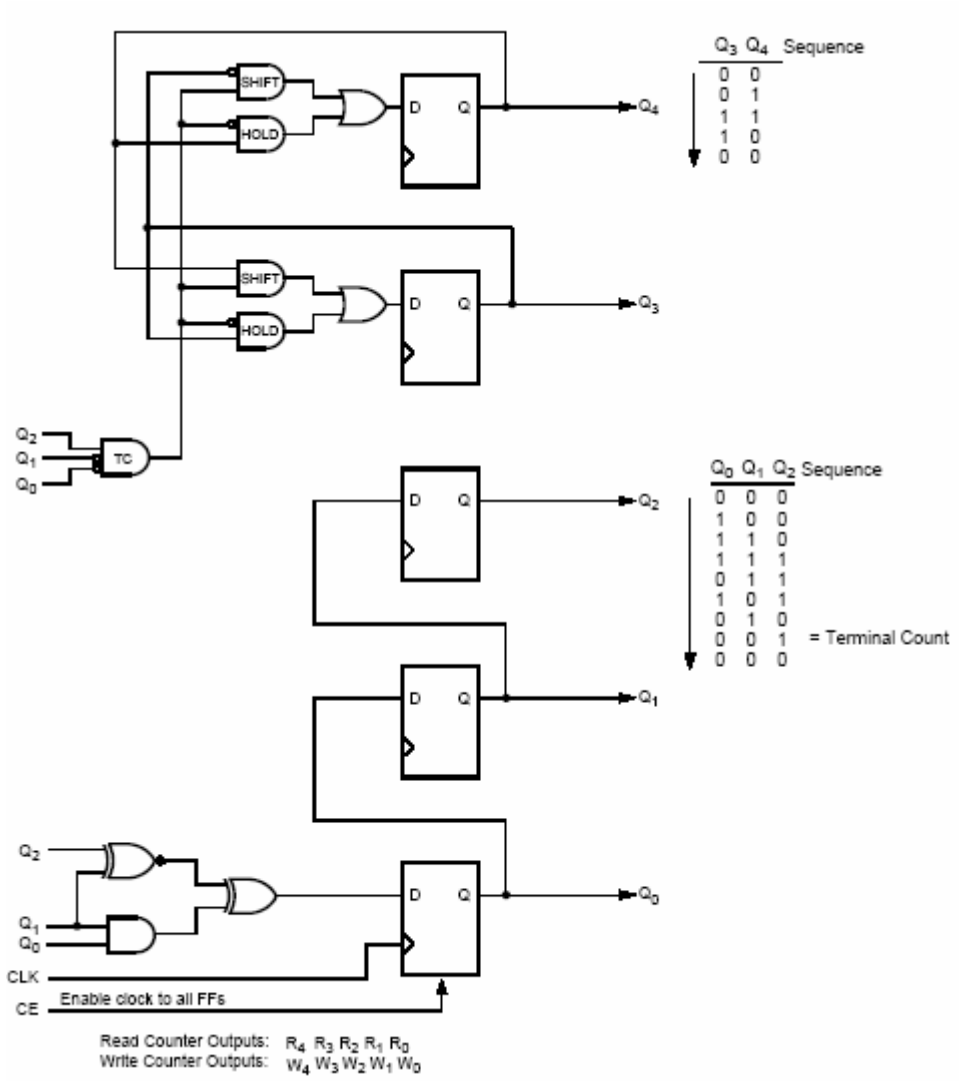


Fig 3.7 : 32-deep FIFO Read Counter or Write Counter

3.6 64 x 8 FIFO

This example implements a 64 x 8 FIFO with independent read and write clocks. Since each CLB can only implement a 16 x 1 dual-port RAM, the 64-deep FIFO uses four

memory banks. Longlines distribute data to and from the RAMs. For the write port, WE gates a 1-of-4 decoder. For the read port, TBUFs are used to multiplex the data onto Longlines, as shown in Figure 3.9.

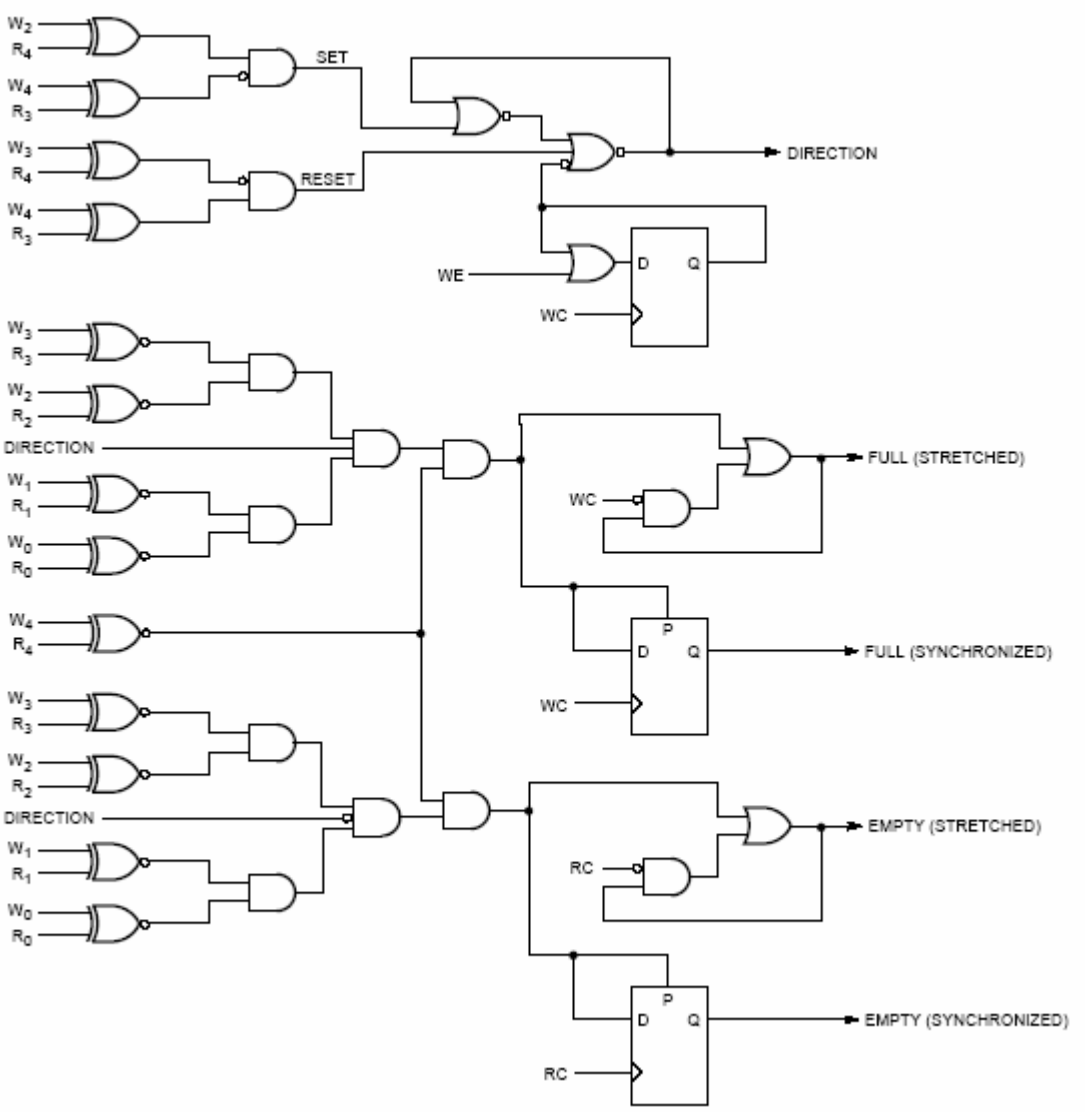


Fig 3.8 : 32-Deep FIFO Asynchronous Control

The 6-bit read and write address counters each consist of a 4-bit ¸16 Linear-Feedback-Shift-Register (LFSR) counter followed by a 2-bit Grey or Johnson counter. (See Figure 2.10.) The latter drives the 1-of-4 decoders selecting between memory banks. The

DIRECTION detector decodes the quadrant information and generates set and reset signals for the DIRECTION flip-flop or latch. When both counters are identical, either a FULL or EMPTY output flag is generated, depending on the state of DIRECTION. Asynchronous control logic for a 64-deep FIFO is shown in Figure 3.11.

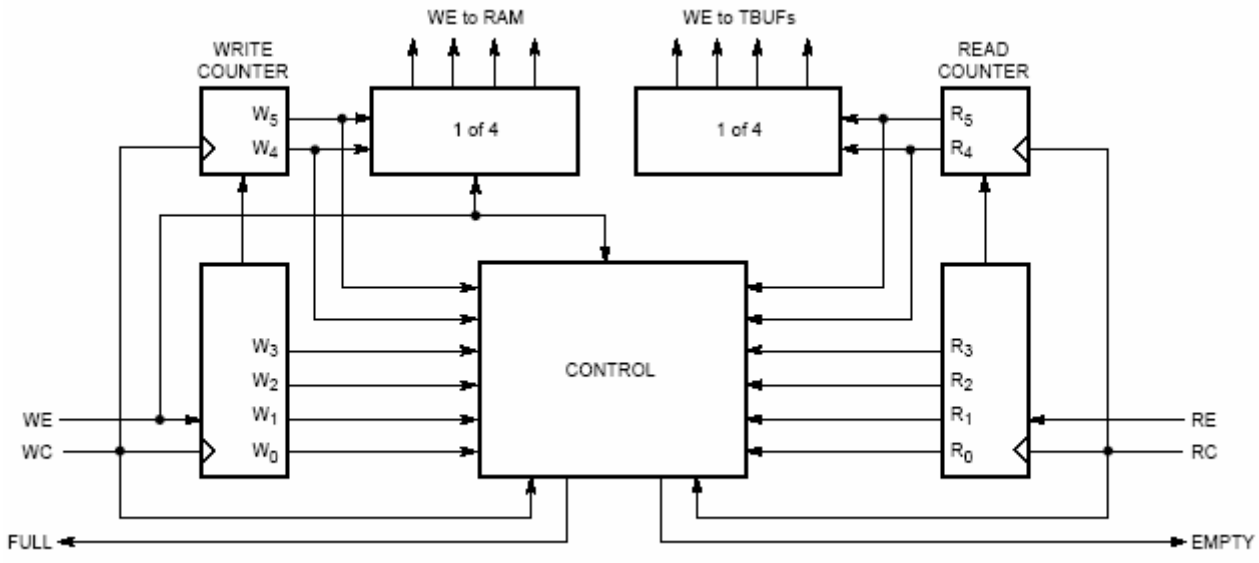


Fig 3.9 : 64 x 8 FIFO Block Diagram

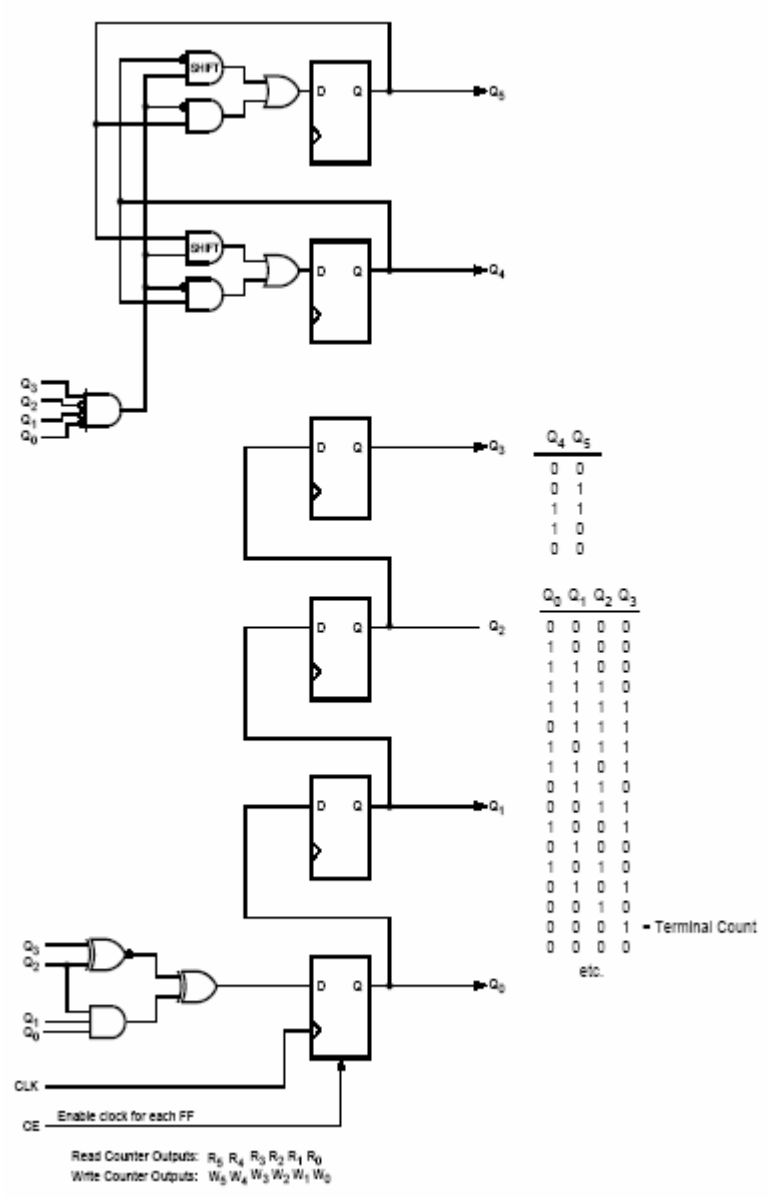


Fig 3.10 : 64-Deep FIFO, Read or Write Counter

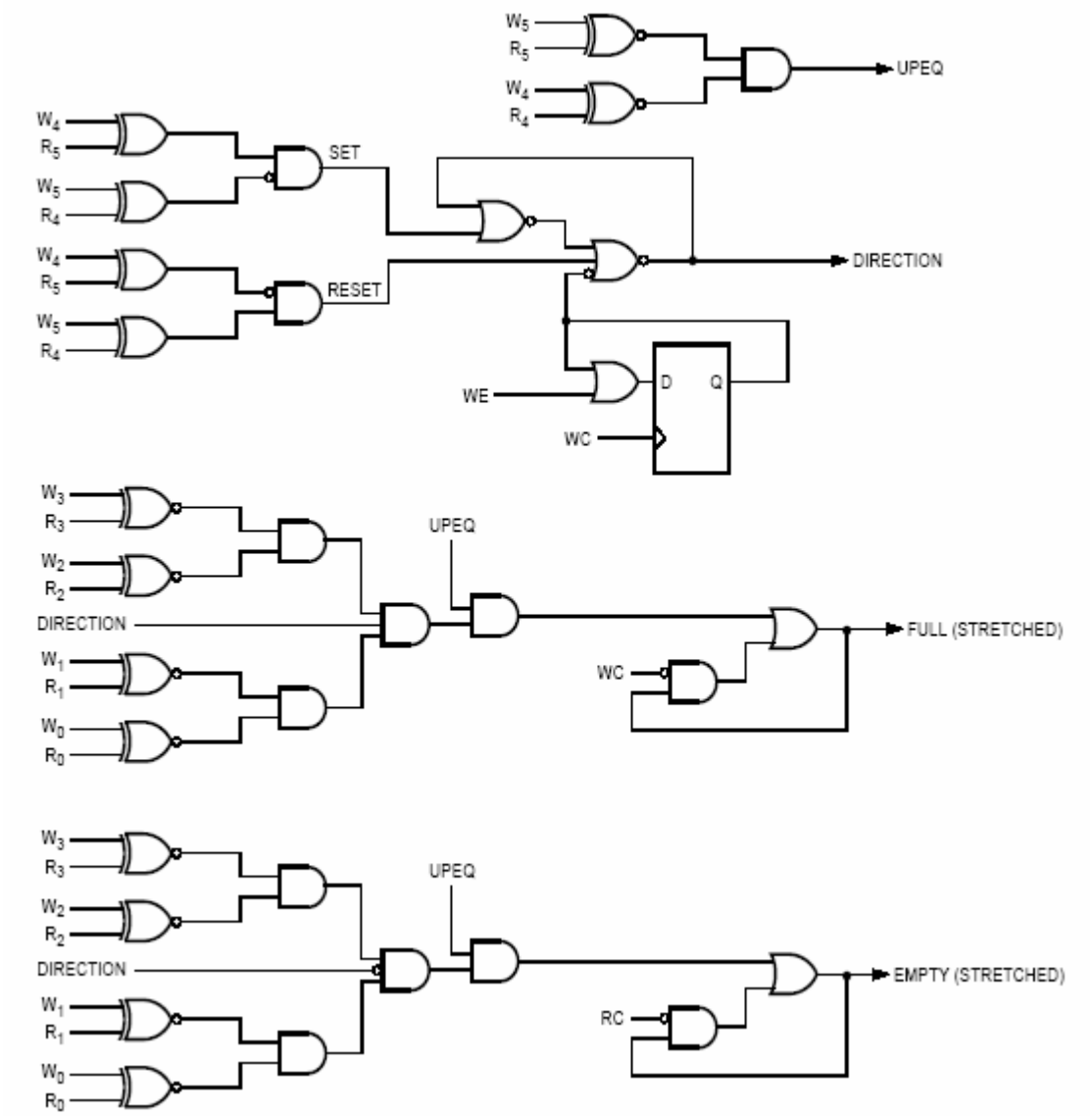


Fig 3.11 : 64-Deep FIFO Asynchronous Control

3.7 An Another Approach

Another approach that can be used is to use dual port memory and the generation of both read and writes pointers as the addresses for each port. Depending on this approach the system can be divided into four functional blocks.

3.7.1 Controller block

The controller block determines and generates the valid read and write signals depending on both the flags and the system read and write requests.

Table 3.1 Read Operation Truth Table

RE	EMPTY	RE_MEM
0	0	0
0	1	0
1	0	1
1	1	0

Table 3.2 Write Operation Truth Table

WE	FULL	WE_MEM
0	0	0
0	1	0
1	0	1
1	1	0

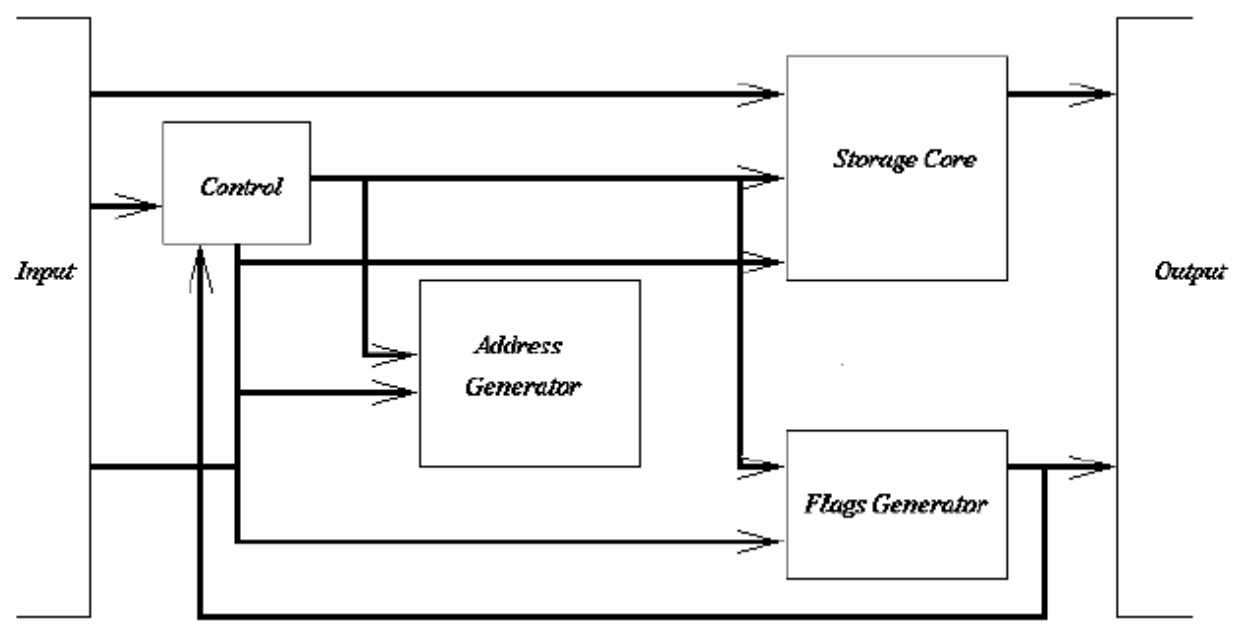


Fig 3.12 System block diagram

Address generation block

This block generates pointers to the next address to be read and to be written

3.7.1.1 Implementation

The block is simply implemented of two 8-bit synchronous counters.

The output of theses two counters represent the two pointers.

The CE chip enable pins are connected to both the read and write enable that are synchronized with the system read and write requests. These lines should be enabled only on valid read or valid write actions. So external logic should control these lines.

Result the clock goes infinitely to the counters but the count up is performed only upon read and write requests

Clear pins are connected to the global reset signal.

3.7.2 Dual port memory

This is the storage core of the FIFO. It consists of 4 signals and one read and one write bus. The four signals include:

Clk : which is the system global clock

Reset is the input signal that is ACTIVE LOW

RE_MEM is the read enable signal and is Active High.

WE_MEM is the write enable signal and it too is Active High.

WA [7:0] is the write address bus

RA [7:0] is the read address input bus

DIN [7:0] is the input data bus

DOUT [7:0] is the output data bus.

3.7.2.1 Implementation

- The write operation is done for each rising edge of the write clock at the address specified for [7:0] when the WE_MEM is active.

- Read operation is not synchronized "asynchronous read" with the clock and it produces output data for each input address RA [7:0]

- A 3-state buffer is located at the output of the dual port in order to enable and disable the output of the FIFO.

- A flip flop with its input connected to the inverted RE_MEM input is to synchronize the read operation with the block clock and to make it active high signal the same as the write operation since the control of the tri state is active low

- A synchronous register is added to the input to synchronize the asynchronous input data to be written.

- A delay flip flop is added between the external WE_MEM input and the WR_EN pin is to synchronize the write command with the input data from the register.

3.7.3 The flags block

This block is one of the most problematic blocks. This block generates the required flags that control and validate the read and write signals. It generates three signals, the FULL signal, which indicates that the FIFO is full and so no further write operation should be attempted. The half-full signal which is just an indication that half the FIFO is full or empty. Finally the empty signal that indicates that the FIFO is empty and no further read operation should be attempted unless a single byte is written in to the FIFO.

Table 3.3 Flag Interface Description

Signal name	Signal Type	Description	Notes
Clk	Input	system global clock	
Reset	Input	system global reset	Active low
RE	Input	Read Enable	Active High
WE	Input	Write Enable	Active High
FULL	Output	Full Flag	Active High
Half_Full	Output	Half Full Flag	Active High
Empty	Output	Empty Flag	Active High

3.7.3.1 Implementation

The main goal of this block is to check both read and write pointers, if they are close together and by how much. If the difference is zero then the FIFO is either full or empty. To determine is it full or empty the previous state of the pointers must be checked.

Instead of implementing this ideas as is , we can use another pointer or indicator that is incremented for each write operation and is decremented for each read operation. when this pointer is zero, the FIFO is empty. When the pointer points to the highest address in the memory it means that the output is full.

Regarding this approach we can use an 8-bit up/down counter to represent the pointer

The counter counts up for valid write operation and counts down for valid read operation.

When there is valid read and write signals a the same time, neither count up nor count down should be done. The same as if there are no valid signals at the input.

This simple arbitration is controlled using CE and count up pins. The truth table of these functions are described in tables 3.4 and 3.5.

Table 3.4 Arbiter Truth Table

RE	WE	CE
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

Table 3.5 Count Direction Truth Table

RE	WE	UP
0	**0**	**X**
0	**1**	**1**
1	**0**	**0**
1	**1**	**X**

So UP = WE

Signal Generation

FULL signal is generated when all the lines of the counter are 1's.

HALF-FULL signal is generated when the MSB bit of the counter is '1'.

EMPTY signal is generated when all lines of the counter are 0's.

CHAPTER 4
DESIGN OF SELF ADDRESSING FIFO

4.1 Introduction

Usually, distributed RAM feature is used to implement First-In-First-Out (FIFO) elastic buffers to form a bridge between subsystems with different clock rates and access requirements.The non-synchronous nature of the single-port RAM confronts the designer with several challenges. Like :

- Addresses must be multiplexed,
- independent read and write clocks must be synchronized
- access requests must be arbitrated.

The improved dual-port RAM solves most of these problems[12]. Since the basic RAM in each CLB has independent write and read addresses, there is no need to multiplex addresses and arbitrate their selection. The synchronous write mechanism simplifies write timing and contributes to much faster operation.

The FIFO design effort can now be concentrated on achieving high throughput and low cost, and on solving the fundamental timing problems created by asynchronous read and write clocks. The self addressing FIFO design employs no external counters. Only flag and status information logic is used. A self-addressing FIFO (first in first out) reference design uses block memories to store both data and address information in a single memory location. These FIFOs are more suitable for data throttling in continuous data systems rather than the full or empty detection required in frame based data systems Their advantage is in using only one clock load. In addition, the status mechanism is very simple.. As there are no counters, the clock load for the incoming and outgoing data is exactly one. Thus, clock skew is no longer an issue

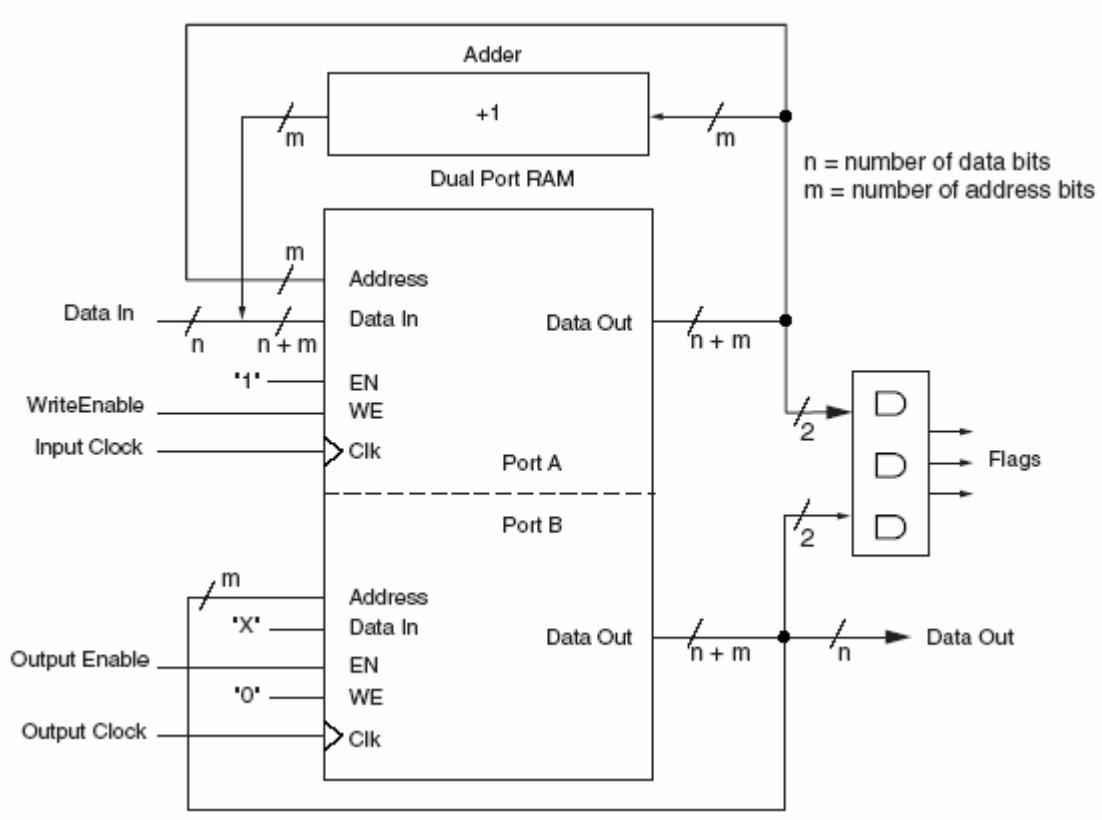


Fig 4.1 Self Addressing FIFO

In Figure 4.1, the self-addressing FIFO counters are implemented as a ROM based sequencer inside the same memory being used for data storage. As there are no counters, the clock load for the incoming and outgoing data is exactly one[13]. Thus, clock skew is no longer an issue, and none of the available 16 global clock trees need to be used. The true dual- port structure of the block RAMs includes input and output data busses plus address and control signals for each port. Data can be independently read from and written to each port. Both read and write accesses are synchronized to the appropriate port clock, with a

positive or negative selectable edge. The only operational limitation is not to write data to a given address from one port while it is being read from the other port. To allow for parity, the basic array size of the block memory is 18 Kbits. Each port may be independently configured for a specific data bus width. For example, Port A can be configured as a 512 x 36 memory, and port B configured differently as a 2048 x 9 memory. Using these features, clock domains are safely crossed at the same time as the data width is converted.

As an example, the output buffer on a communication system could use 36-bit wide data written at 50 MHz, and read as 9-bit wide data at 155 MHz. All the various combinations of memory structure are shown in Table 4.1.

Table 4.1

Possible Dual Port Ram Configurations

PORT A	**PORT B**
512 X 36	512 x 36, 1024 x 18, 2048 x 9, 4096 x 4, 8192 x 2, 16384 x 1
1024 x 18	1024 x 18, 2048 x 9, 4096 x 4, 8192 x 2, 16384 x 1
2048 x 9	2048 x 9, 4096 x 4, 8192 x 2, 16384 x 1
4096 x 4	4096 x 4, 8192 x 2, 16384 x 1
8192 x 2	8192 x 2, 16384 x 1
16384 x 1	16384 x 1

4.2 Implementing a Counter In a Memory

There are two possible methods for implementing a counter inside a memory. Method one is presented for interest while method two forms the basis of the reference design. By deasserting the write enable pin, a synchronous RAM becomes a synchronous

ROM, or in other words, a sequencer. The contents of each of the ROMs can be initialized via the bit stream at device power-up. Essentially any sequence and therefore any state machine (within input and output pin limits) can be implemented in the ROM. The n-bit counter is a trivial case, it is a state machine with “n” inputs and “n” outputs. The count order is determined by the contents of the memory. The count speed is governed by the output enable control, effectively making it a clock enable. With counters implemented in one block memory, and data storage in another a FIFO can be built. However, this method gives a fanout of two on the read and write clocks. Although better than the classic FIFO design, there is still a possibility of clock skew.

The second method uses a feedback mechanism. M address bits are fed back from the RAM output data port to its address port. The same m bits are fed to a simple incrementer circuit. The output of the incrementer circuit is combined with the incoming data word. This next address plus data word is written synchronously into the next memory location. The only initialization needed is presetting the RAM outputs to x00 at reset. The function is available by using the block memory SSRA pin.

We can now imagine a scenario where some of each word in the memory is used as the counter storage, and the rest is used for data storage. This is where the wide data busses comes in handy. For example, a 36-bit word can contain 32 bits of data and four bits of address, to give a 16-deep by 32-wide FIFO. Or the 36-bit word could have nine bits of address and 27 bits of data to give a 512-deep by 27-wide FIFO. The choice is up to the designer. The top two bits of the incrementer circuit are Gray encoded. Only one of these two bits can ever change at any one clock transition. This minimizes any possibility of error when reading the flags, but it makes the incrementer slightly more complicated, and therefore slower. To maximize the speed of the design, a pure binary incrementer could be used, but the flags will need to be checked twice to ensure a valid flag value.

In Figure 4.1, an example data transfer is started by writing data (23) to location x00. After the clock edge, the data output of port A contains x0123. The lower data byte (23) is ignored, the upper byte (x01) is fed back as the address for the next write. A one is added to x01 for merging with the next incoming data. For example, if the next incoming data is 45, then x0245 is written into location x01. This value then appears at the output of port A. The next cycle will write to location x03 and so forth.

Reading data is less complicated. An output clock and optional output enable signal are applied to Port B. Data is read synchronous to the clock. The lower byte is the valid data that was written into Port A of the FIFO. The upper byte contains the address for the next read. The upper byte is therefore fed back to the address inputs of Port B.

Having covered the basic FIFO operation, one very important feature remains. Indicating whether a FIFO is full, empty, or somewhere in between is usually referred to as a flag. Since the top two bits of the counters are Gray coded, only one of the two bits changes at a time. It is safe to compare the top two bits of the write counter with the top

two bits of the read counter, even though they are in two separate time domains. This comparison gives three flag outputs.

Flag0: The FIFO being between empty and one-half full

Flag1: The FIFO being between one- quarter to three-quarters full

Flag2: The FIFO being between one-half to full

This method gives a very simple yet elegant mechanism for handling FIFO requests. To take advantage of this method use flag1 (retimed to the receiver clock domain) as the FIFO read enable. In this way the FIFO is never near overflow, or emptied, and all asynchronous conditions are avoided. A standard system is also assumed to have

continuous input data. A pictorial view of the counter and flag operation is given in Figure 4.2.

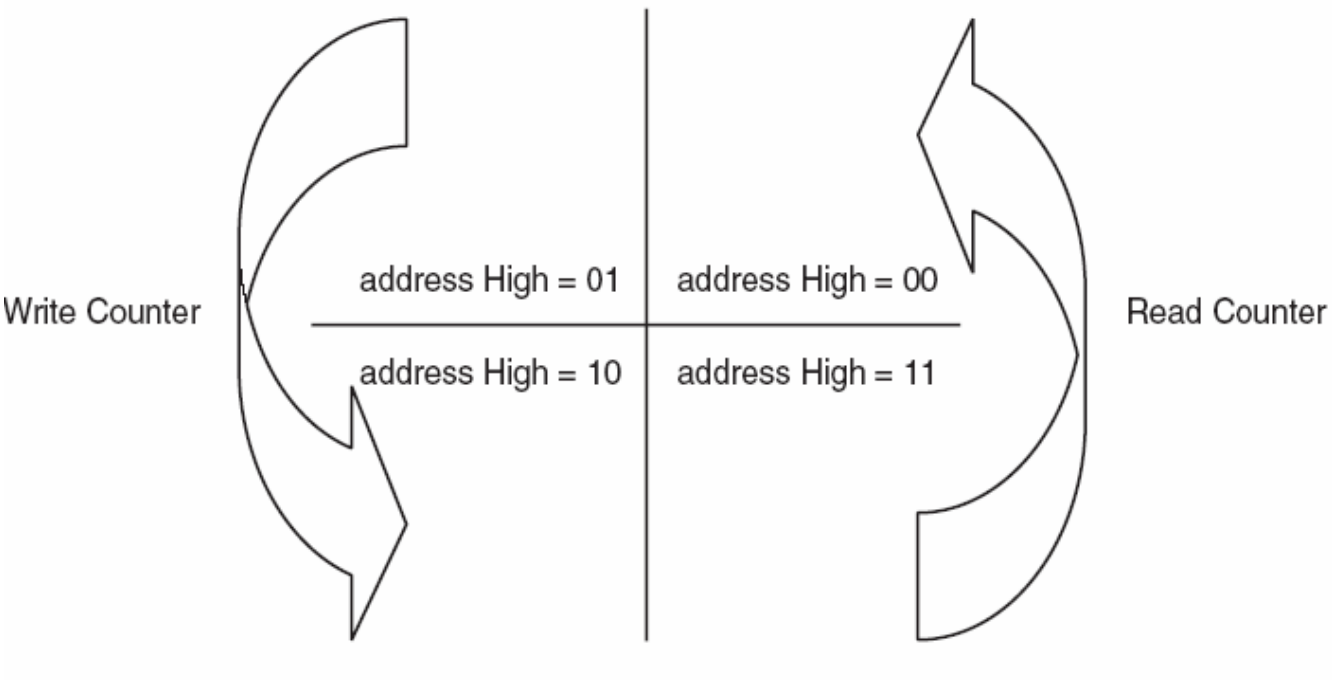


Fig 4.2 Flag Operation

4.3 Conclusion

The self-addressing FIFO is a small and novel mechanism for transferring data between clock domains while avoiding the necessity of using a clock tree. The only constraints are the obvious ones. Data must be valid at the clock edges and the clock period still needs to be controlled. Clock skew is not an issue and therefore general

purpose routing may be used for the input or output clock depending on the system architecture. This shows the ability to build multiple self-addressing FIFOs, allowing designers to input or output data to external devices without ever using the on-board global clock resources. This is extremely useful in the design of many sorts of systems, for example where interfaces to multiple external devices are required.

CHAPTER 5

A FASTER FIRST IN FIRST OUT MEMORY USING LFSR.

5.1 Introduction

Many designs require First-In-First-Out (FIFO) elastic buffers to form a bridge between
subsystems with different clock rates and access requirements. The Spartan-II FPGAs provide dedicated on-chip blocks of 4096 bit dual-port synchronous RAM, which are ideal for use in FIFO applications. As a result, these low-cost FPGAs can significantly reduce system cost by integrating discrete FIFOs along with other complex logic while providing the speed and I/O levels necessary for interface to larger external memories. The Block RAM memory extends the capability of the distributed RAM memory in CLBs, which requires input and output data multiplexing for depths greater than 16 words. Each Block RAM port can be configured independently as 4Kx1, 2Kx2, 1Kx4, 512x8 or 256x16. The Block RAM is fully synchronous for both writing and reading.

This chapter describes a 512x8 FIFO, with the depth and width being adjustable within the HDL code. First the design for a FIFO with common read and write clocks (synchronous) is described. Then the design changes required for the more difficult case of independent read and write clocks are presented. This is referred to as "asynchronous" or "unsynchronized" in reference to the two clocks, although the FIFO logic itself is always synchronous.

5.2 Synchronous Design (Using Common Clocks)

The first design is synchronous, which means it uses common clocks for Read and Write. When both the Read and Write clocks originate from the same source, the FIFO operation and arbitration are simplified, and the Empty and Full flags are easily

generated. See Table 5.1 for the port definitions for this design.

Table 5.1: Port Definitions, Common-Clock Design

Signal name	**Port direction**	**Port name**
Clock_in	input	1
fifo_gsr_in	input	1
write_enable_in	input	1
write_data_in	input	8
read_enable_in	input	1
read_data_out	output	8
full_out	output	1
empty_out	output	1
fifocount_out	output	4

Linear Feedback Shift Registers (LFSRs) are used for both the read (read_addr) and write (write_addr) address counters. Because they are addressing a RAM, a binary counting sequence is not necessary, and the pseudo-random sequence of the LFSRs is acceptable. They use very little logic, and are therefore much faster than a standard binary implementation. The only drawback is that the FIFO size is reduced by one, to 511x8. The fifo_gsr signal resets all counters. The synchronous nature of the Block RAM memory simplifies the timing requirements to meeting setup times. To perform a read, Read Enable (read_enable) is driven High prior to a rising clock edge, and the Read Data (read_data) is presented on the outputs during the next clock cycle (Figure 5.1). To do a Burst Read, simply leave Read Enable High for as many clock cycles as desired. If Empty goes active after reading, then the last word has been read, and the next Read Data would be invalid.

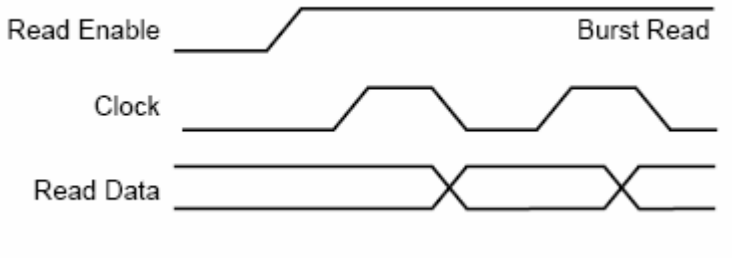


Figure 5.1 Read Cycle

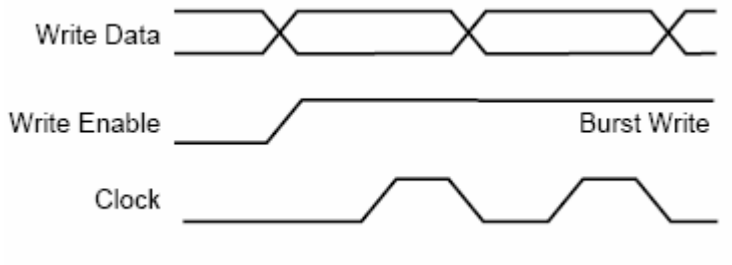


Figure 5.2 Write Cycle

To perform a write, the Write Data (write_data) must be present on the inputs, and Write Enable (write_enable) is driven High prior to a rising clock edge. (See Figure 5.2.) As long as the Full flag is not set, the Write will be executed. To do a Burst Write, the Write Enable is left High, and new Write Data must be available every cycle. The Empty flag is set when the Next Read Address (next_read_addr) is equal to the current Write Address, and only a Read is being performed. This early decoding allows Empty to be set immediately after the last Read. It is cleared after a Write operation (with no simultaneous Read). Similarly, the Full flag is set when the Next Write Address (next_write_addr) is equal to the current Read Address, and only a Write is being performed. It is cleared after a Read operation (with no simultaneous Write). If both a Read and Write are done in the same clock cycle, there is no change to the status flags. During global reset (fifo_gsr), both these signals are driven High, to prevent any external logic from interfacing with the FIFO during this time.

A FIFO count (fifocount) is added for convenience, to determine when the FIFO is 1/2 full, ¾ full, etc. It is a binary count of the number of words currently stored in the FIFO. It is incremented on Writes, decremented on Reads, and stays the same if both operations are performed within the same clock cycle. In this application, only the upper four bits are sent to I/O, but that can easily be modified.

5.3 Changes Required for Independent Clocks

Now we will examine the situation where the read and write clocks are independent or "asynchronous" to each other. The port definitions for this version of the design are shown in Table 5.2.

Table 5.2: Port Definitions, Independent Clock Design

Signal name	Port direction	Port name
write_clock_in	input	1
read_clock_in	input	1
fifo_gsr_in	input	1
write_enable_in	input	1
write_data_in	input	8
read_enable_in	input	1
read_data_out	output	8
full_out	output	1
empty_out	output	1
fifostatus_out	output	5

Keeping in mind that the Block RAM itself is still fully synchronous, and the read and write timing is identical to the common-clock design. In order to operate a FIFO with

independent Read and Write clocks, some asynchronous arbitration logic is needed to determine the status flags. The previous Empty/Full generation logic and associated flip-flops are no longer reliable, because they are now asynchronous with respect to one another, since Empty is clocked by the Read Clock, and Full is clocked by the Write Clock.

To solve this problem, and to maximize the speed of the control logic, additional logic complexity is accepted for increased performance. There are primary 9-bit Read and Write binary address counters, which drive the address inputs to the Block RAM. The binary addresses are converted to Gray-code, and pipelined for a few stages to create several address pointers (read_addrgray, read_nextgray, read_lastgray, write_addrgray, write_nextgray) which are used to generate the Full and Empty flags as quickly as possible.

Gray-code addresses are used so that the registered Full and Empty flags are always clean, and never in an unknown state due to the asynchronous relationship of the Read and Write clocks. In the worst case scenario, Full and Empty would simply stay active one cycle longer, but this would not generate an error. When the Read and Write Gray-code pointers are equal, the FIFO is empty. When the Write Gray-code pointer is equal to the next Read Gray-code pointer, the FIFO is full, having 511 words stored. Additional comparators are used to determine when the FIFO is Almost Empty and Almost Full, so that Empty and Full can be generated on the same clock edge as the last operation.

(Traditional control logic uses an asynchronous signal to set the flags, but this is much slower and limits the overall performance). Unlike the common-clock version, it is not possible to keep a reliable count of the number of words in the FIFO, so a FIFO status output is used instead. It is five bits wide, with the signals representing various ranges of fullness, as seen in Table 5.3.

Table 5.3 : FIFO Status

FIFO Status Bit	Description
Fifostatus[0]	FIFO is between Empty and 1/4 Full
Fifostatus[1]	FIFO is between 1 word and 1/2 Full
Fifostatus[2]	FIFO is between 1/4 Full and 3/4 Full
Fifostatus[3]	FIFO is between 1/2 Full and Full
Fifostatus[4]	FIFO is between 3/4 Full and Full

The FIFO status outputs are mutually exclusive, meaning only one will be High at any one time, but the ranges that they cover overlap. They are based on the Gray-code pointers, and the quadrant deltas that exist between the Read and Write addresses. Because of the nature of Gray-code counting, more precision (such as one based on octants) can be easily added, because the upper bits of a Gray-code address are themselves Gray-coded, so there will not be any incorrect status registered.

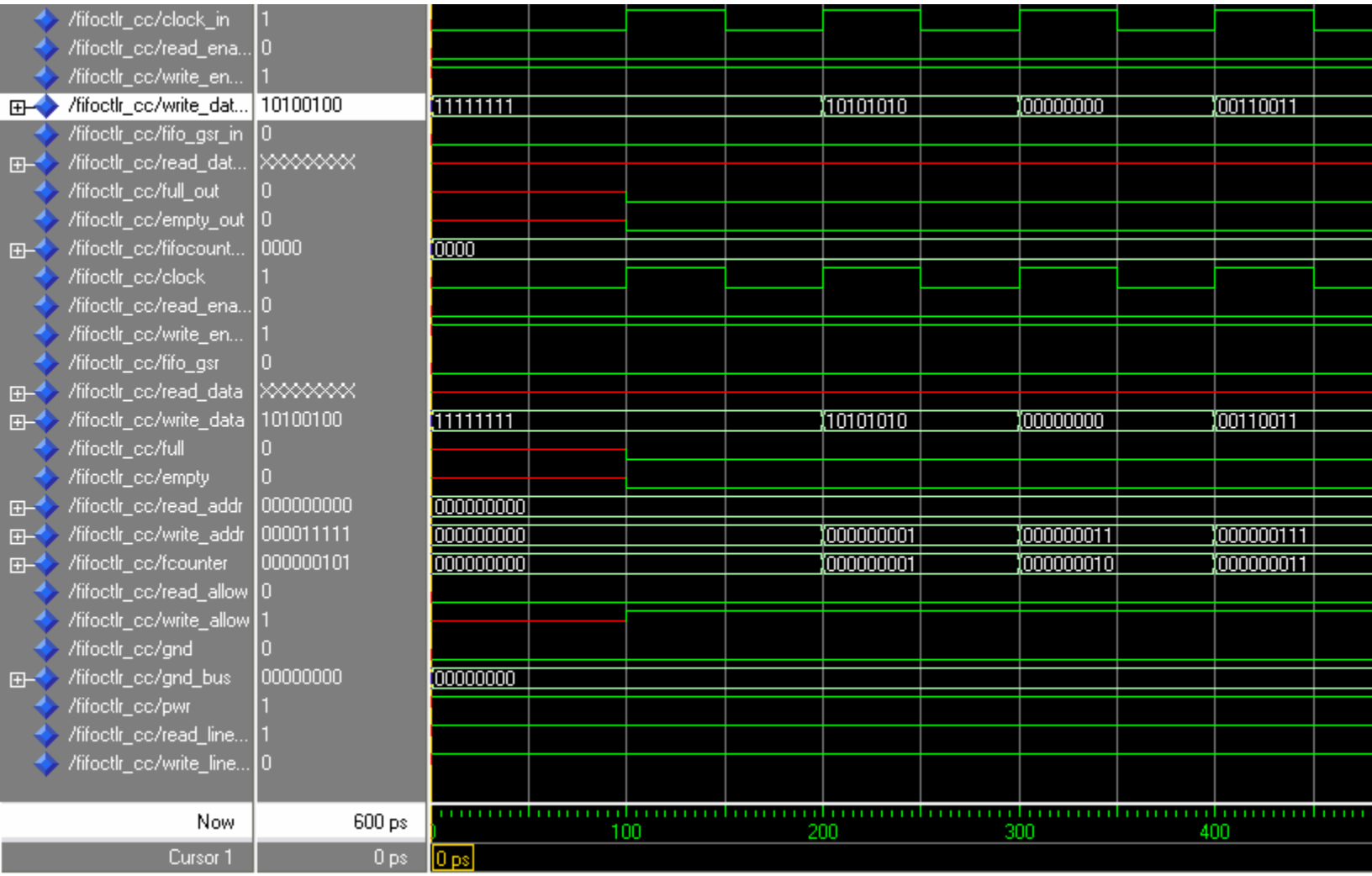


Figure B1 LFSR FIFO in Burst Write Operation with Read Enable Disabled

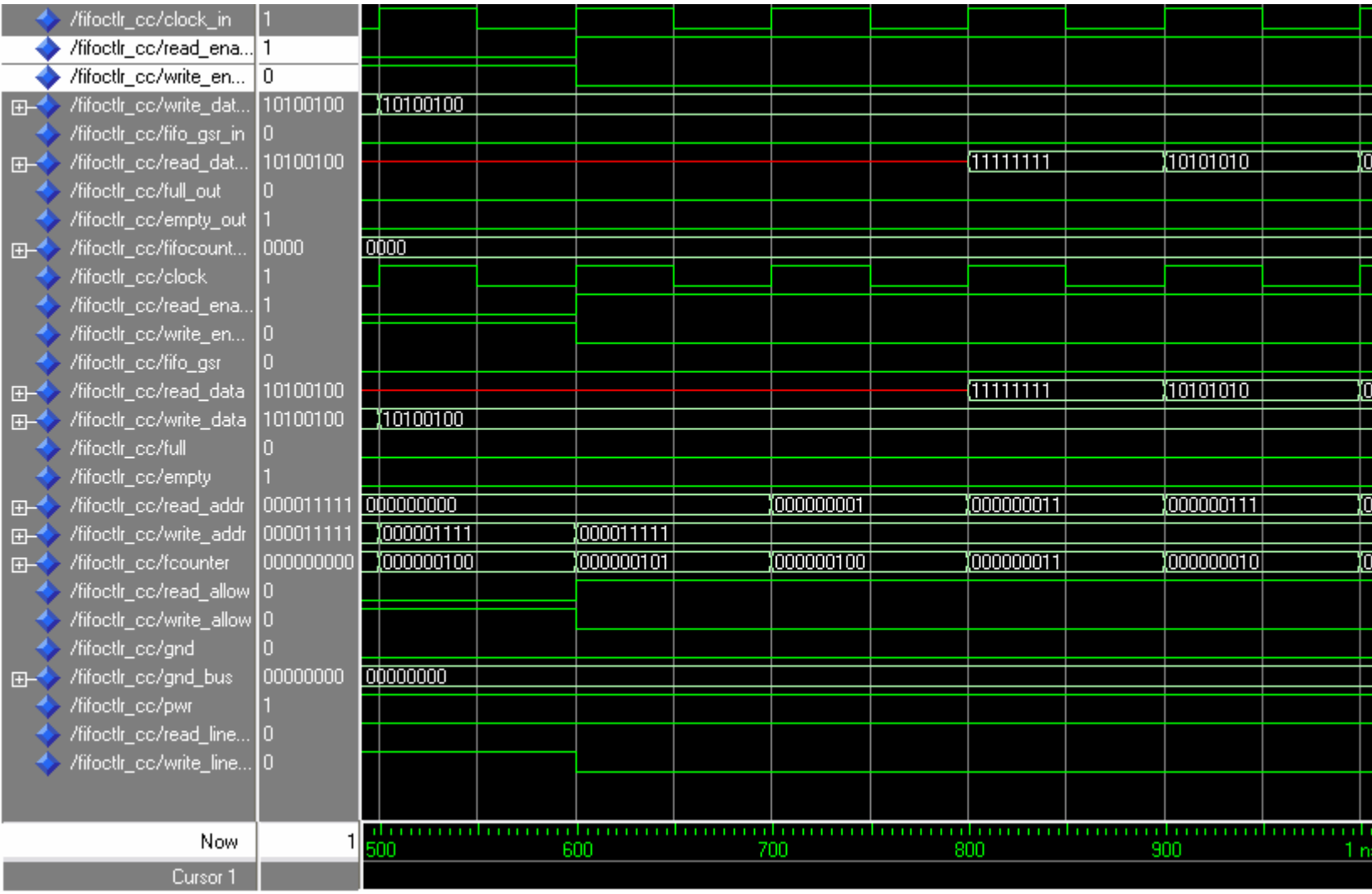


Figure B2 LFSR FIFO at Burst Read

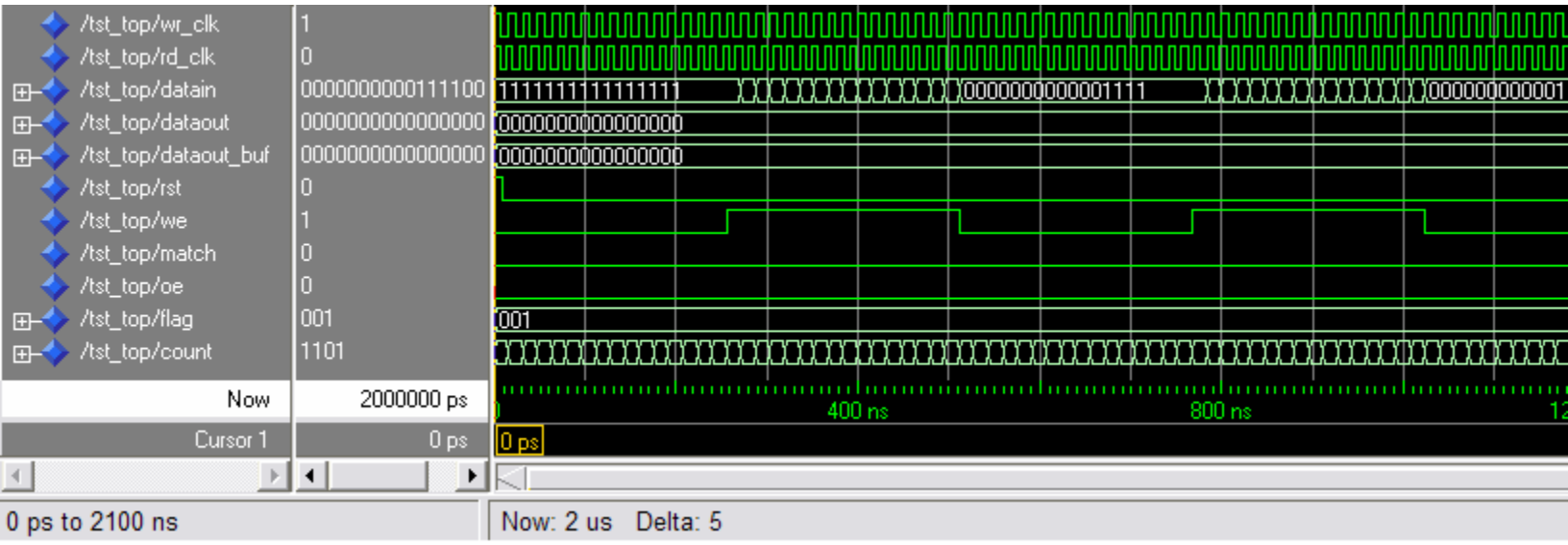


Figure A1 Self Addressing FIFO at Start UP

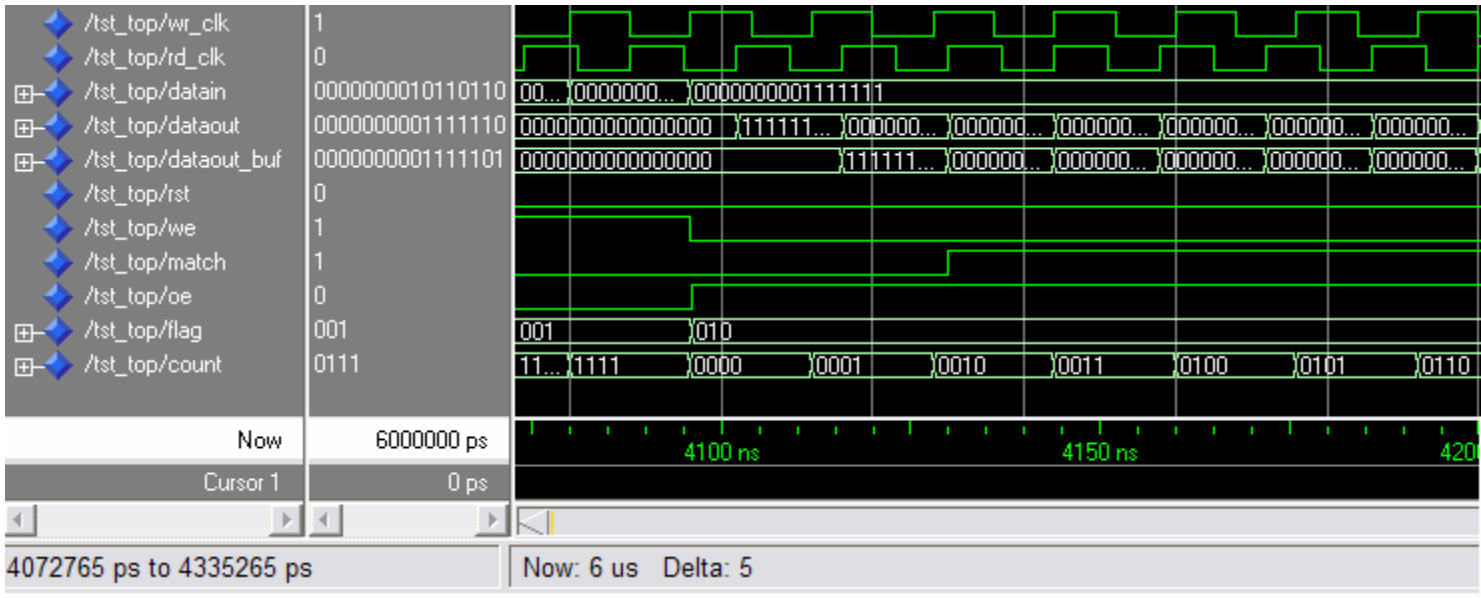
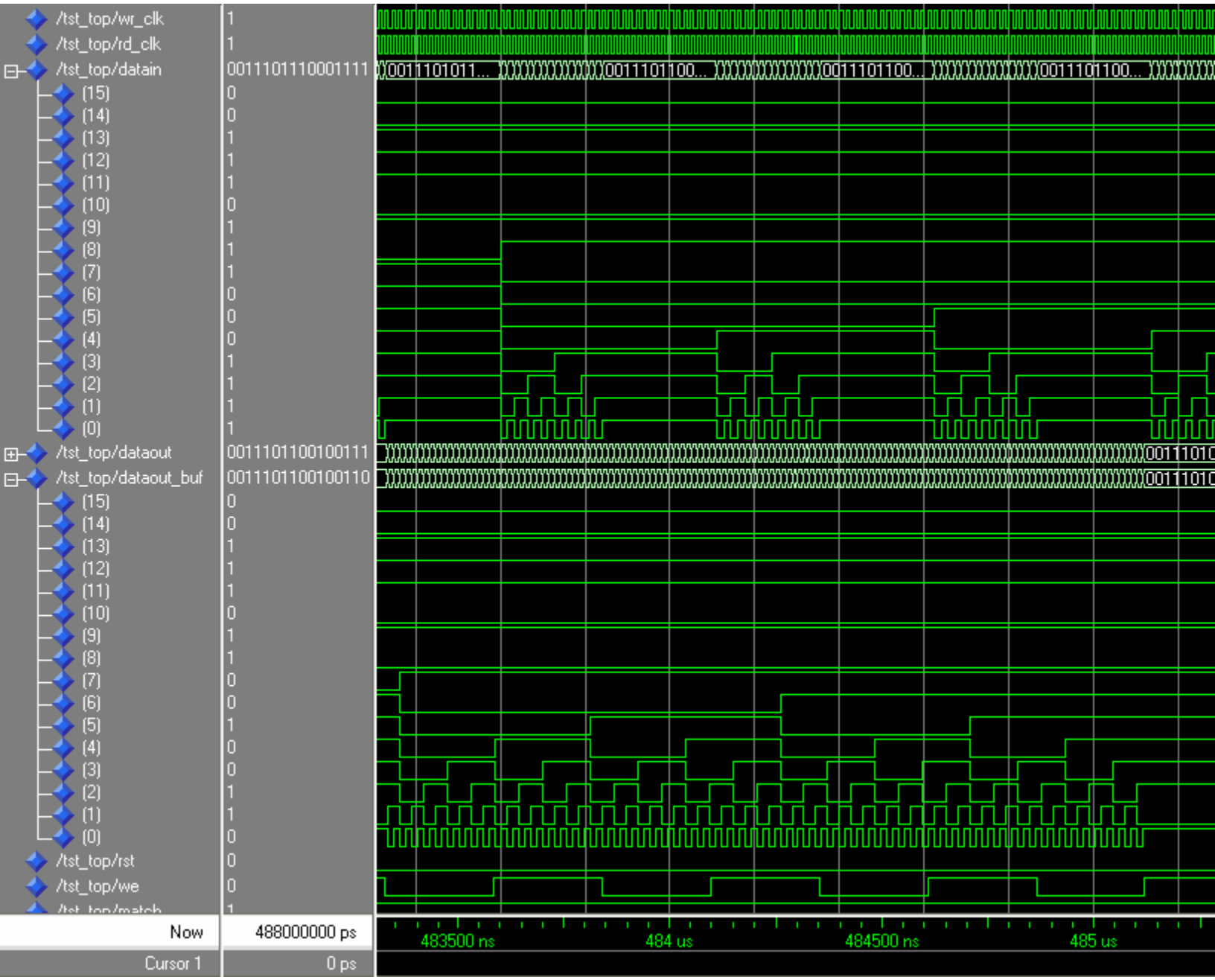


Figure A2 Self Addressing FIFO at signal *oe* High


/tst_top/wr_clk
/tst_top/rd_clk
/tst_top/datain
(15)
(14)
(13)
(12)
(11)
(10)
(9)
(8)
(7)
(6)
(5)
(4)
(3)
(2)
(1)
(0)
/tst_top/dataout
/tst_top/dataout_buf
(15)
(14)
(13)
(12)
(11)
(10)
(9)
(8)
(7)
(6)
(5)
(4)
(3)
(2)
(1)
(0)
/tst_top/rst
/tst_top/we
1
1
0011101110001111
0
0
1
1
1
0
1
1
1
0
0
0
1
1
1
0011101100100111
0011101100100110
0
0
1
1
1
0
1
1
0
0
1
0
0
1
1
0
0
0011101011...
0011101100...
0011101100...
0011101100...
00111010
00111010
Now
488000000 ps
Cursor 1
0 ps
483500 ns
484 us
484500 ns
485 us

REFERENCES

[1] R. Sivaram, C.B. Stunkel, and D.K. Panda, "HIPIQS: A High- Performance Switch Architecture Using Input Queuing," IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 275-289, Mar. 2002.

[2] Z. Zhang and Y. Yang, "Multicast Scheduling in WDM Switching Networks," Proceedings of IEEE International Conference on Communication (ICC '03), pp. 1458-1462, May 2003.

[3] Hashimoto, M., and Nomura, M. "A 20-11s 256Kx4 FIFO memory", IEEE Journal of Solid-State Circuits, vol. 23, no. 2, pp. 490-499, 1998

[4] Nick Kanopoulos and Jill J. Hallenbeck, "A First-In, First-Out Memory for Signal Processing Applications", IEEE Transactions on Circuits And Systems, vol.cas-33, no. 5, pp. 556-558, May 1986

[5] J. Muttersbach, T. Villiger, H. Kaeslin, N. Felber, and W. Fichtner, "Globally-asynchronous locally-synchronous architectures to simplify the design of on-chip systems," Proceedings of the 12th IEEE International ASIC/SOC Conference, pp. 317-321, 1999.

[6] H. Veendrick, "The behavior of flip-flops used as synchronizers and prediction of their failure rate," IEEE Journal of Solid-State Circuits, vol. 15, no. 2, pp. 169-176, 1980.

[7] I. Sutherland and S. Fairbanks, "GasP: a minimal FIFO control," Proceedings of the 7th IEEE International Symposium on Asynchronous Circuits and Systems,

pp. 46-53, 2001.

[8] R.M. Fuhrer, "Sequential Optimization of Asynchronous and Synchronous Finite-State Machines: Algorithms and Tools," PhD Thesis, Columbia University, May 1999

[9] Peter Alfke, " Synchronous and Asynchronous FIFO Designs", Xilinx Application Notes xapp 051, 1996

[10] R.Mossain, L.Wronski, and A. Albicki, "Low power design using double edge triggered flip-flop", IEEE Transactions on VLSI Systems, vol. 2, no.2, pp. 261-265, 1994

[11] H. Wang and P.C. Liu: 'Double-edge-triggered address pointer for low-power high-speed FIFO memories', Electronics Letters, vol. 33, no. 5, 27 feb 1997

[12] Nobutaro Shibata, Mayumi Watanabe, and Yasuyuki Tanabe, "A Current-Sensed High-Speed and Low-Power First-In–First-Out Memory Using a Wordline/Bitline-Swapped Dual-Port SRAM Cell," IEEE Journal of Solid-State Circuits, vol. 37, no. 6, pp. 735-750, June 2002.

[13] P. Giaccone, B. Prabhakar, and D. Shah, "An Efficient Randomized Algorithm for Input-Queued Switch Architecture," Proc. IEEE Hot Interconnects 9, Aug. 2001.

[14] Praveen K. Murthy, Edward A. Lee, "Multidimensional Synchronous Dataflow," IEEE Transactions on Signal Processing, vol. 50, no. 7, pp. 2064-2079 , July 2002

[15] C.-S. Chang, D.-S. Lee, and C.-K. Tu, "Using switched delay lines for exact emulation of FIFO multiplexers with variable length bursts," in Proceedings of IEEE Infocom, vol. 3, San Francisco, CA, pp. 1998–2007, 2003

[16] I. Chlamtac, A. Fumagalli, and C.-J. Suh, “Multibuffer delay line architectures for efficient contention resolution in optical switching nodes,” IEEE Trans. Commun., vol. 48, pp. 2089–2098, Dec. 2000

[17] Bernie New, “Loadable Binary Counters,” Xilinx Application Notes, 1997. XAPP 004. , pp. 73-77.