UNIVERSITY OF CALIFORNIA AT BERKELEY
COLLEGE OF ENGINEERING
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Checkpoint 3
SDRAM

1.0 Motivation

This checkpoint serves two purposes:

1. Familiarize you with SDRAM
2. Provides an easy way to buffer outgoing video

It is important to be familiar with SDRAM as it is an increasingly common component in many designs. In addition to providing large, cheap storage for projects like those in EECS150, SDRAM forms the heart of every computer.

Most digital designs are only as useful as the amount of storage they contain. Even streaming Digital Signal Processing (DSP) applications, the canonical example of a stateless or memory-less design often require significant amounts of RAM for computations like matrix transposes, FFT and delays.

Because it is so expensive to build large or fast SRAM, DRAM has become nearly ubiquitous. The largest hurdle to using DRAM is that it must be "refreshed" periodically in order to maintain its contents. The second largest problem with any kind of RAM is often that it is asynchronous, making it difficult to interface with. Both of these are somewhat mitigated by using Synchronous DRAM or SDRAM, which provides a synchronous interface with guaranteed timing, including a 3 cycle command sequence which will automatically perform a refresh operation.

Because of its relative ease of use, and nearly universal inclusion in large digital designs, SDRAM is likely to remain a permanent part of digital design for many years to come. Therefore this checkpoint is meant not only to provide you an easy way to buffer video data, which will simplify your project, but will give you significant experience in working with SDRAM.

At this point in class, you are expected to be able to design and implement a large majority of the circuits on your own. This documentation is meant only as a loose guide, and you should feel free to alter any specifications you feel necessary. Figures and numbers presented here should only be used conceptually. Actual timing charts provided by Xilinx should be used when designing your circuit.

"BECAUSE YOU WILL BE KEEPING AND RELYING ON THIS CODE FOR MONTHS, IT WILL ACTUALLY SAVE YOU MANY STRESSFUL HOURS TO ENSURE IT WORKS WELL NOW, RATHER THAN WHEN YOU ARE ABOUT TO FINISH THE PROJECT"

2.0 Introduction

In figure 1 below you can see the organization of all of the modules you will be building for checkpoint #3, as well as those which you must reuse from checkpoint #2 and #1.

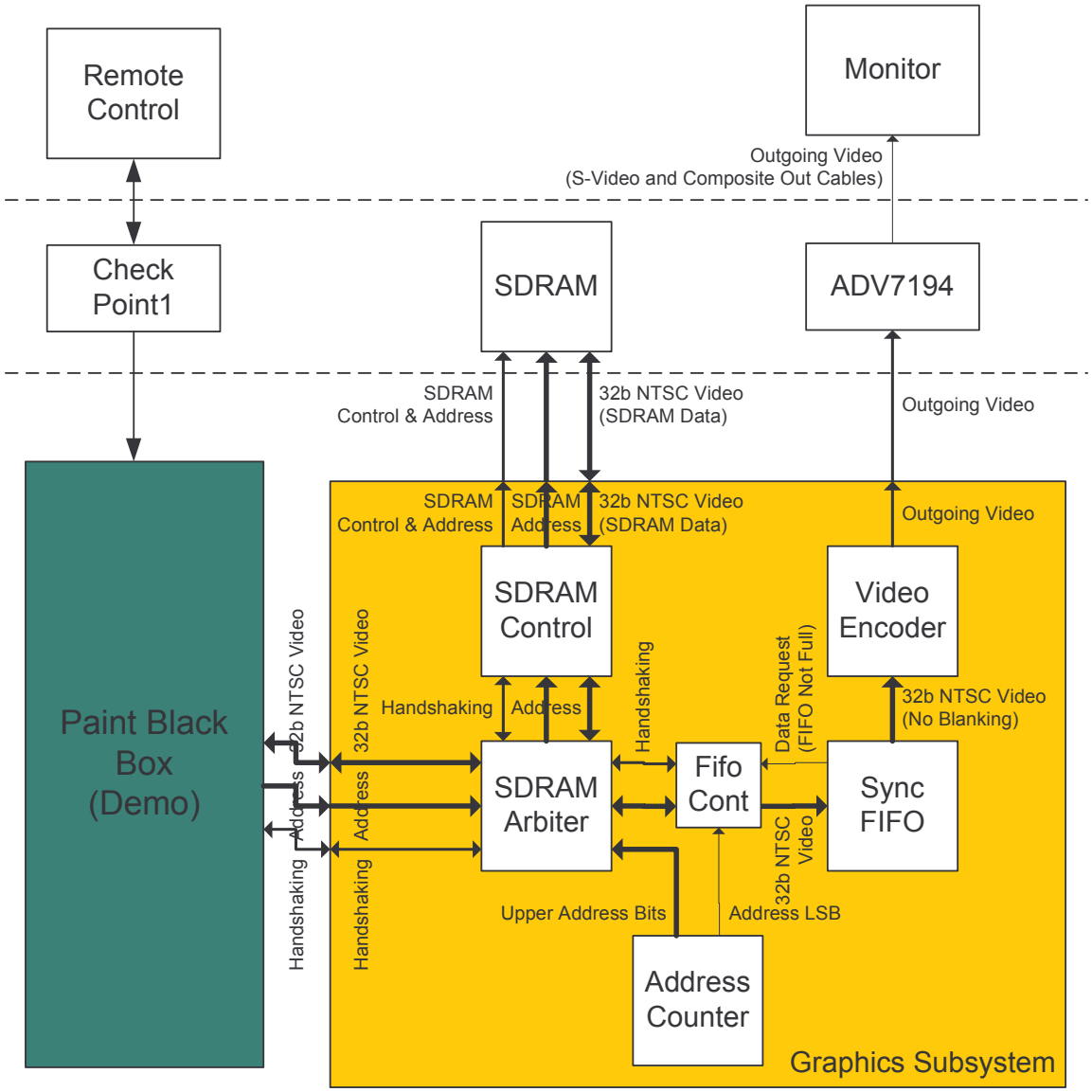

Figure 1: Checkpoint 2 & 3 Organization

2.1 SDRAM Subsystem

The modules you will building for this checkpoint is a combination of a protocol bridge and a command FSM, which will take care of issuing and timing SDRAM commands leaving your other modules free to ignore the details of working with SDRAM. You will be building two main modules to access the SDRAM: SDRAM Control, and the SDRAM Arbiter.

The primary responsibilities of your SDRAM Control module are:

1. Initialize SDRAM
    a. Wait 100us after reset

   b. Issue an initialization sequence to properly set-up the SDRAM chips
2. Issue SDRAM commands
   a. When a read or a write is ready, it should be sent to the SDRAM chips
   b. Your module must take care to ensure proper burst timing
   c. If the system is idle, an “autorefresh” command must be issued to guard against data loss.

Essentially your SDRAM Control module will abstract away the fact that you are working with SDRAM, both by taking care of the detailed and specific timing requirements, and by automatically issuing “autorefresh” commands to guard against data loss in the SDRAM.

The primary responsibilities of your SDRAM Arbiter module are:
1. Provide multiple access ports to the SDRAM Controller
   a. The Paint engine, which is provided as a black box at this stage, and is a write only port
   b. The Video Fifo, which reads data from the SDRAM and feeds it into the Video Encoder.
2. Deals with hand shaking between the individual ports and the SDRAM Control.

Your SDRAM Arbiter is like a funnel which controls the interface between your SDRAM controller and the two ports: the paint engine and video fifo. Your arbiter will have to allow both our game engine and your fifo to access SDRAM, and insure that they do not conflict.

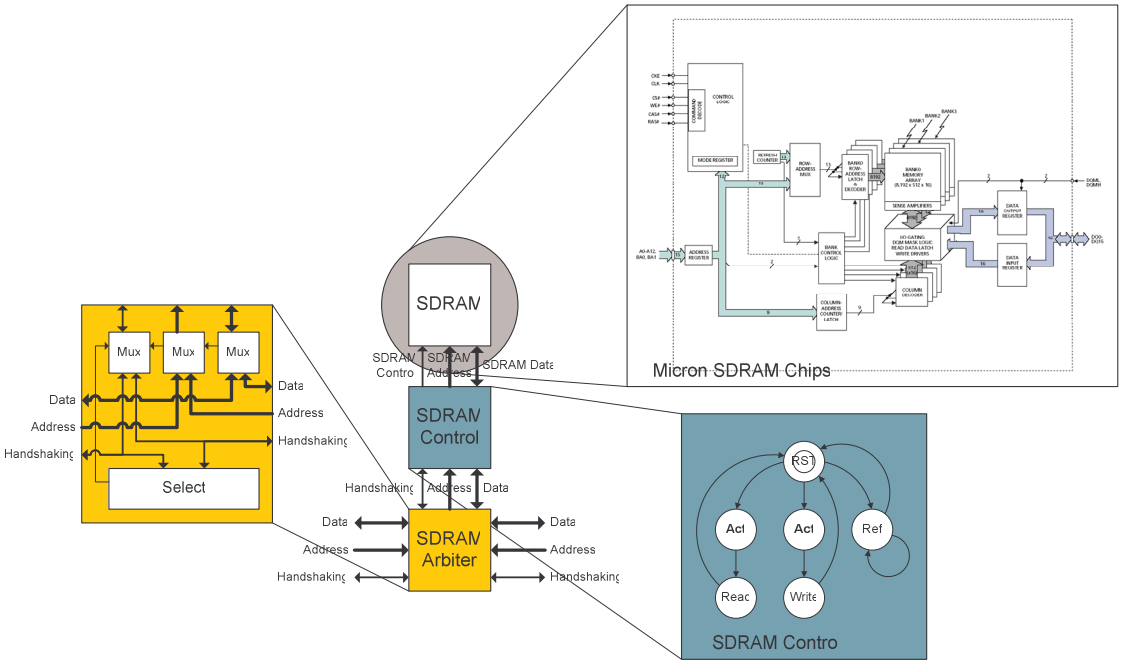

Figure 2: Memory Subsystem Organization

2.2 Video in SDRAM

Of course the point of this checkpoint is to store video in SDRAM. This section details the format of the video data in memory.

Given that the video data is already organized into 32bit words, and that there are two 16bit wide SDRAM chips for a total of a 32bit wide memory. In order to keep things simple, you will be sending the same control signals to both SDRAM chips.

The organization of pixels into words and words into bursts is shown in figure 3 below.

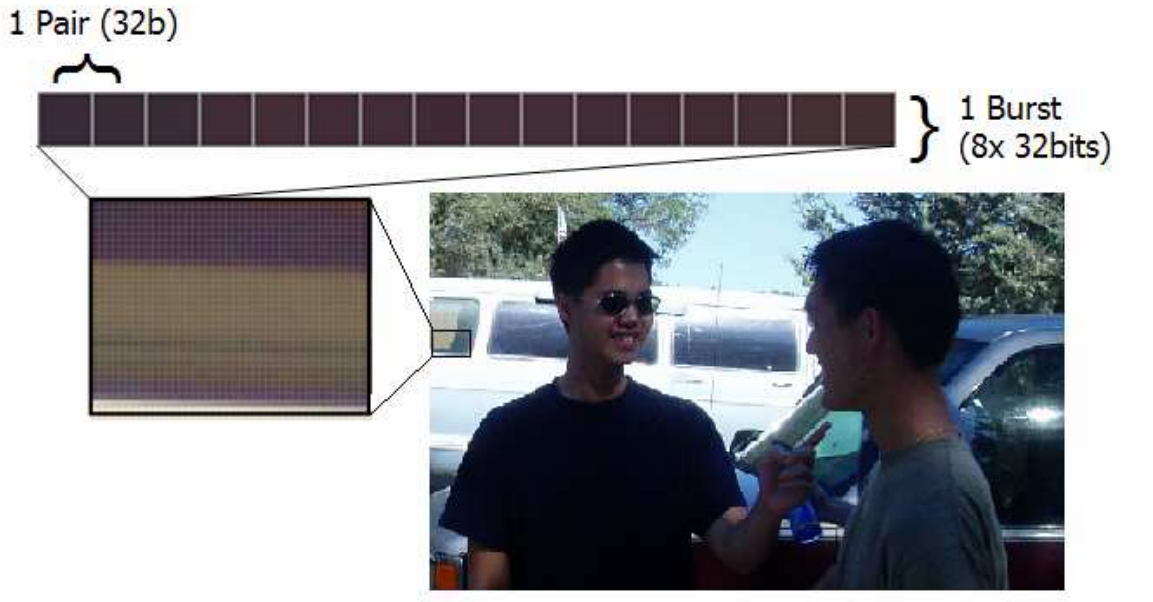

Figure 3: Video Memory Organization

This simply shows the organization of the pixels within a burst. In addition, we must decide on a format for how to organize the bursts. In order to interoperate with the game demo, you will need to conform to a specific address format:

```
RowAddress = {{pad{1'b0}}, PixelRow, Field}
BankAddress = 2'h0
ColumnAddress = {BurstColumn, 3'h0}
```

Notice that this addressing scheme has a few subtleties. First of all, you must **pad the top of the row address with 4*1'b0s**, to make it total a 13bit for address. The `PixelRow` is a number from 0 to 252 or 253, indicating the active video line and `Field` is the 1bit field indicator using the **same row addressing scheme as in checkpoint #2**.

The **bank address is fixed at 2'h0**, because we need only a fraction of the total memory available.

The **column address is based on the video pixel column**. Notice that in this case the `BurstColumn` indicates which BURST you wish to access. The `3'h0` on the right ensures that you will always **start at the beginning of a burst**. For more information about word ordering within bursts, see the SDRAM Datasheet.

By **dividing the SDRAM address into row and column along the same boundaries as the video row and column**, we have **simplified debugging**, so that you can easily see which pixels are being read and written at any time.

3.0 Prelab

Please make sure to complete the prelab before you attend your lab section. **You will not be able to finish this checkpoint in 3hrs! Labs are getting progressively longer.**

1. **Read this handout thoroughly**.
    a. Pay particular attention to section **4.0 Lab Procedure** as it describes what you will be doing in detail.
2. Examine the documents page of the website
    a. **You will need to get used to reading datasheets, like these. They form the core of information available to hardware designers.**
    b. http://www-inst.eecs.berkeley.edu/~cs150/fa04/Documents.htm#Datasheets
    c. **Read the MT48LC16M16 Datasheet**
    d. **It is expected that you use the data sheets as your primary source of information.**
3. **Examine the Verilog** provided for this checkpoint
    a. There isn't much, so it should be pretty clear
    b. Checkpoint4.v is simply a black-box for the demo paint engine
4. **Start your design ahead of time**.
    a. Begin with **schematics** and **bubble-and-arc** diagrams
        i. Make sure to design your SDRAM Controller FSM
        ii. Make sure to design your SDRAM Arbiter FSM
        iii. Design the handshaking protocols between the two and stick with them.
    b. Come prepared for your design review
        i. You will need to provide bubble and arc diagrams as well as high level design schematics.
    c. Start building your testbenches early
        i. Perhaps have one person design a module and the other design a testbench, and then switch.
        ii. You will need separate testbenches for your SDRAM Controller and SDRAM Arbiter
        iii. You cannot pass this checkpoint without good testbenches
5. **This will be the longest checkpoint, plan accordingly**.
    a. You will need to test and debug your verilog thoroughly.
    b. **You must build a reliable interface with a real hardware component!**

4.0 Lab Procedure

Remember to **manage your Verilog, projects and folders well**. Doing a poor job of managing your files can cost you **hours of rewriting code**, if you accidentally delete your files.

4.1 SDRAMControl.v

This is the **main module you will need to build** for this checkpoint. We are giving you complete design freedom in how to build this module. The only requirements are that it works well with your SDRAM arbiter and that it connect to the SDRAM chips on the CaLinx2 board.

The essential functions of you module are listed in section 2.0 Introduction, above. Your module must:

1. Initialize SDRAM
    a. Wait 100us after reset
    b. Issue an initialization sequence to properly set-up the SDRAM chips (See pages 12-14 and 40 of the SDRAM Datasheet)
2. Issue SDRAM commands
    a. When a read or a write is ready, it should be sent to the SDRAM chips (See pages 46, 51, 53, 58 of the SDRAM Datasheet)
    b. Your module must take care to ensure proper burst timing
    c. If idle for too long, an "autorefresh" command must be issued to guard against data loss (See page 43 of the SDRAM Datasheet)

You will find that aside from sequencing the commands from page 15 of the SDRAM Datasheet, you will need to ensure that they are timed correctly. At the bottom of every timing diagram, there is a table of timing information, which you will need to use to guarantee that the commands happen not only in the right order but at the right times. For this **you will need to know that the memory chips are the -7E models, and the clock is running at 27MHz.** Remember that clock signals provided on the data sheet may or may not be running at 27MHz.

In order to test the functionality of your SDRAM Control, Micron Technologies Publishes a simulation module "mt48lc16m16a2.v". We have included it in the zip file. This simulates the wait cycles from the SDRAM and has a few warning messages. Keep in mind a few things when using this in your test bench.

1. The module is more rigorous than we are, so some of the warning messages may not apply.
2. The data line is 16 bits, so you will need 2 instantiations in your test bench
3. Testing is not a substitute for getting it on the board, it must work on the chip, not just in simulation.
4. Read the documentation on this module before using it/asking questions. It is available on the website under documentation.

4.2 SDRAMArbiter.v

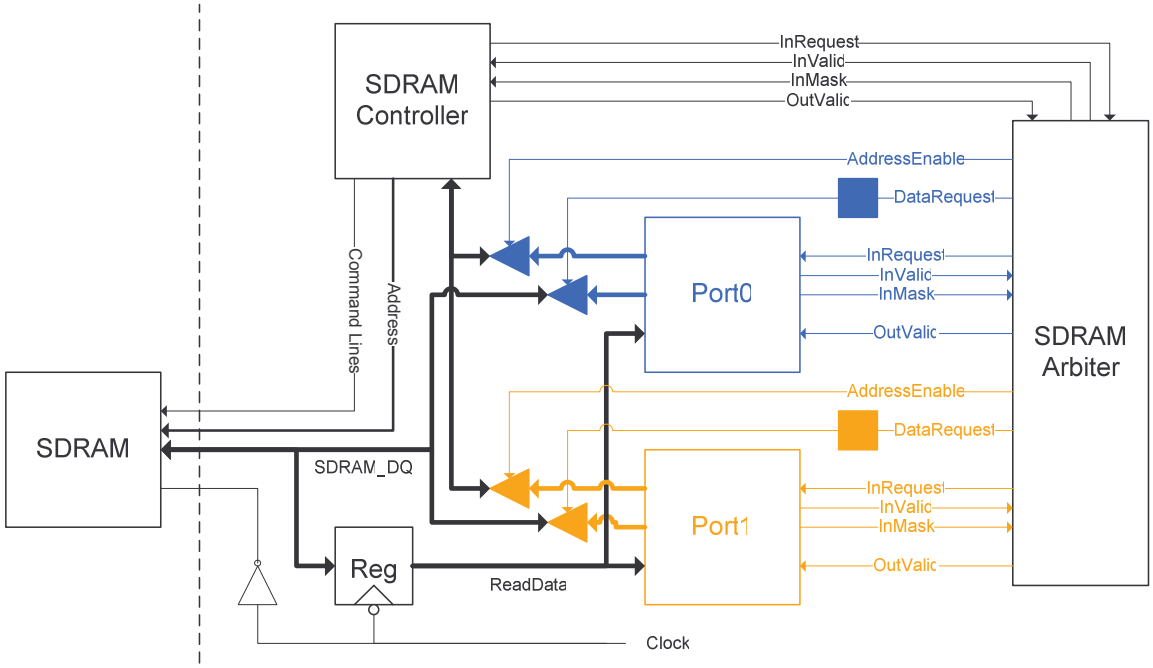

Figure 4: SDRAM Controller and Arbiter Interaction

Your SDRAM Arbiter will be the glue that fits this checkpoint together. It will interact with both the Checkpoint4.v file, containing the paint engine and the FIFO system which you will design according to section 4.3 Video below.

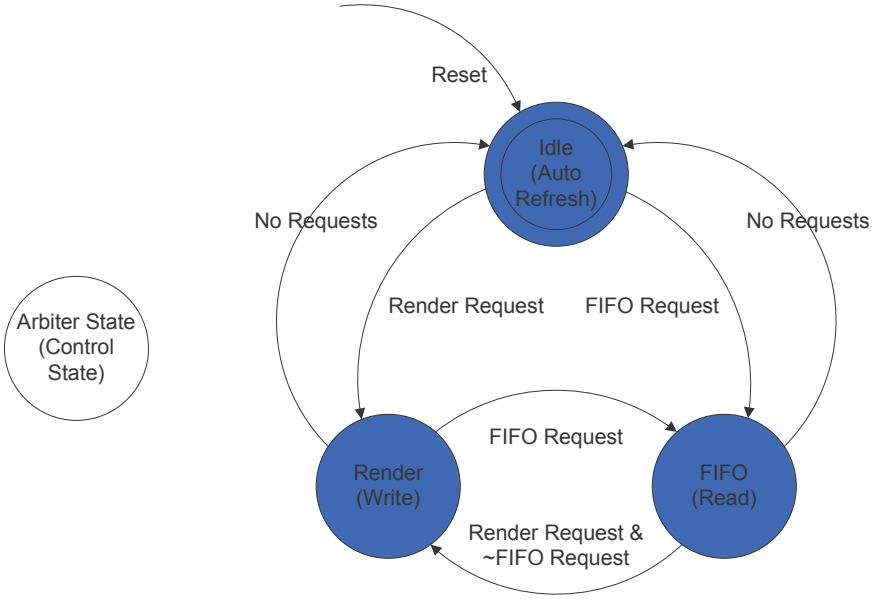

Figure 5: SDRAM Arbiter State Diagram

In addition to figure 4 and 5 above which lay out a rough sketch of the state machine you will need and the connections you will need, you have the following table of

input and output specifications to work from. **Be sure to take note of the timing details included in the table. You should draw some example timing diagrams.**

Signal	Width	Dir	Description
DIn	32	I	Data to be written to the SDRAM
AIn	24	I	SDRAM Address ({Row, Bank, Column})
DOut	32	O	DOut from SDRAM. Remember to include the negedge register as shown in figure 4.
InRequest	1/port	O	This signal indicates that the arbiter is requesting a new operation from the specified port. Notice that when the arbiter is idle, it will request new operations from all ports once every other cycle.
InDataRequest	1/port	O	This signal is used by the arbiter to request new data to be written to the SDRAM. While the arbiter is servicing a port, InDataRequest should go high when the SDRAM is ready to receive data. The port should provide the data on DIn the cycle AFTER this signal is asserted.
InAddressEnable	1/port	O	When this signal is on, the specified port should drive the address for its current request onto the AIn bus. Essentially this is the control for the tristates on the AIn bus (see figure 4)
InValid	1/port	I	Indicates that the port has a valid operation it wishes the SDRAM to perform. The arbiter sets InRequest high, the next cycle, the port should return an InValid if it wishes to perform a read/write. Invalid should remain high until the next InRequest. If this is 1'b0, the port has no operation and should remain low until the next InRequest.
InWriteEnable	1/port	I	Indicates that the current operation for this port is a write. This signal, like InValid may only change the cycle after InRequest is 1'b1. For now, the Paint engine is a write only port, while the Video Fifo is a read only port.
InMask	1/port	I	This signal will change with Din, on every cycle after InDataRequest is 1'b1. It should be routed to the DQMH and DQML lines to control whether data is written to the SDRAM on individual words of the burst.

`OutValid`	1/port	O	Output from the arbiter indicating that a read on this port is currently in progress and that a word of data for that operation is available on `DOut` on THIS cycle.

Table 1: Port Specification for SDRAMArbiter.v

4.3 Video Counters

In order to get data from the SDRAM to the Video Encoder, we will use a FIFO as a buffer. FIFOs are good for two things: data rate matching and buffering. We will need both of these features.

The data rate from the SDRAM is 32bits/cycle during a burst, and 0bits/cycle otherwise, in comparison to the data rate required by the video encoder: 32bits/4cycles we obviously have a mismatch. The trick is that the Video Encoder asks for data in a very specific order. Thus the FIFO will allow you to read data from SDRAM ahead of what the video encoder needs and buffer it until the video encoder is ready for it.

Shown in figure 6 below is the rather simple circuit you will need to create to ensure that the FIFO does not become empty.

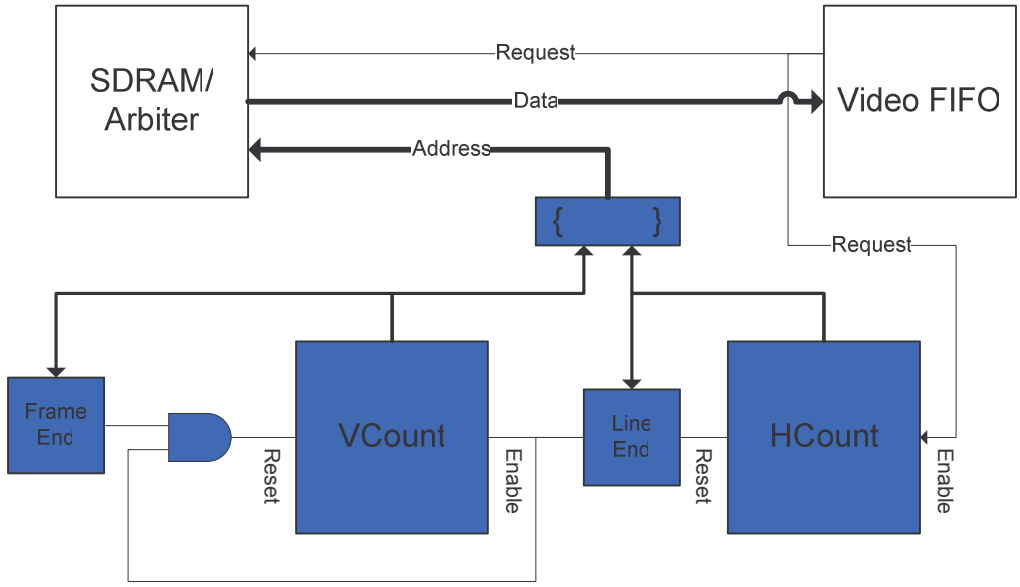

Figure 6: FIFO Fill Circuit

One of the key design points of this circuit is deciding when to read from the SDRAM. In order to keep the FIFO as full as possible without losing data, you should read from SDRAM whenever there is space for another burst in the FIFO.

You must also pay close attention to the addressing scheme laid out in section 2.2 Video in SDRAM.

4.4 fifo_sync32d.v

The sync fifo is a simple, two interface module, you write into one and read from the other. The way this module assists is to buffer your data and allow the Video Encoder and the SDRAM to run at different rates.

Signal	Width	Dir	Description
clk	1	I	The write clock signal (EthernetClock)
sinit	1	I	Reset, will empty the FIFO
din	32	I	Data input bus
Wr_en	1	I	Write the value on din into the FIFO on the clock
Rd_en	1	I	Read enable, dout will be valid after the clock
dout	32	O	Data output bus, valid next cycle after rd_en
full	1	O	Indicates that the FIFO is currently full
empty	1	O	Indicates that the FIFO is currently empty
data_count	2	O	Indicates how many words are in the FIFO 2'b00 means 0-¼ full 2'b01 means ¼ - ½ full 2'b10 means ½- ¾ full 2'b11 means ¾ -1 full

Table 2: Port Specification for fifo_sync32.v

Your fifo should never go empty, thus if it is less than ¾ full, you should immediately request data from the SDRAM.

4.5 Cursor

The paint engine draws the correct colors into SDRAM when told to do so by the user. However, the paint engine does not draw the Cursor. Instead, the paint engine outputs the Cursor X and Y coordinates, the width and Height of the Cursor, and the Color being used. For this check point, you are responsible for making the Cursor appear on the screen and for the center of the Cursor to change color (see demo bit file).

a. On reset, as well as when an Erase all is selected, the Cursor X and Y coordinates are {0,0} and the Width and Height coordinates are {~1023,~1023}.

b. The X-Y coordinate is always the upper left corner of the square

c. X,Y, Width, Height are each 10 bits wide, Color is 32 bits wide.

d. All values change at the same time, whenever the user indicates.

e. X starts from the far left(0) and goes to the far right(~725), Y goes from the top (0) and goes to the bottom(~512).

4.6 Black Boxes

For this checkpoint, you are given a few Black Boxes. The Paint Engine, with a wrapper file Checkpoint4.v, as well as fifo_sync32.v. In order to add these black boxes to your project, you must instantiate it somewhere in your project the way you would with any other module. Then copy/paste the .edn or .edf to your project directory. The paint engine follows all the port specifications laid out here, and you must interface with it correctly in order to get the desired operation.

4.7 Tips, Notes, and Caution

Before attempting to dive into this check point keep in mind a few things as you design your module

1. The Video fifo relies on the fact that it can predict what the Video encoder wants before the encoder requests the data. If your counts are off, or if your fifo goes empty for even one cycle, your fifo and your encoder will no longer match. Thus you will be feeding incorrect data to the encoder and you will see horizontal and vertical scrolling on the SVideo Screen.

2. To ensure that the video fifo never goes empty, you should ALWAYS give preference to the Video fifo port and ignore the game engine until you are ready to deal with it.

3. At it's peak, the Video Encoder will request data at a rate of 32bits/4cycles. The SDram feeds you 32bits/cycle for 8 cycles followed by a wait period until the next burst of data is available. Thus to use 8 pieces of data, the Video Encoder will take 32 cycles. This means that in the worse case scenario (a write followed by a read), the SDRAM Controller/Arbiter must be able to go through the entire process of requesting data in less than 32 cycles.

4. Remember SDRAM is actual hardware. All timing issues must be dealt with carefully, make sure your signals are active during the correct number of cycles. If your InMask, InDataRequest, or OutValid Signals are incorrect, you will notice weird unexplainable errors.

5. Make sure both you and your partner understand how all modules work and communicate with one another. Determine a handshaking protocol between each module before you start, and stick to it.

6. To synthesize black boxes, make sure the .edn file is copy pasted into your project directory, and that the .v wrapper is included in your project.

7. The black boxes do not contain any tristates. They simply output the data onto the line. If you want to tristate anything, you must create one on your own.

8. In Figure 4 there is a NEGATIVE edge triggered Flip flop in between the SDRAM and Port1/Port0. Make sure that this register is declared as an IO register as in Checkpoint 2 and that a ".Clock(~Clock)" is sent in as the parameter.

9. The game demo assumes that you have a correct Checkpoint 1 working with stabilized outputs from the controls. You will send in a 30 bit wire (N64Status in FGPATop2+.v) representing the button signals as laid out in Checkpoint 1.

10. Start early, this checkpoint is meant to be hard and time consuming. Good luck, and remember that there are always more important things than CS150.

5.0 Checkpoint3 Checkoff

Name: _____ Name: _____
Section: _____

I Simulation _____ (40%)
- a. SDRAM Controller
- b. SDRAM Arbiter & FIFO
- c. SDRAM & Video Encoder

II Cursor Display _____ (20%)
Display Demo Game on Board _____ (40%)

III Hours Spent: _____

IV Total: _____

V TA: _____

RevB – 9/22/2004	Jack Tzeng	Updated to be used in Fall2005
RevA – 10/28/2004	Greg Gibeling	Built from Checkpoint1 from Spring2004