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

Checkpoint 1
SDRAM

Introduction

Welcome to Checkpoint #1. Read this document thoroughly and get started early. This checkpoint is meant to ease your transition into this semester's project; however, it would be a mistake to take it lightly. **Initial Advice: When we tell you that you need to read the Datasheets – READ THEM!**

1.0 Motivation

This checkpoint serves three purposes:

1. Familiarize you with working with SDRAM
2. Build a simple SDRAM interface for later use in the project
3. Give you practice on reading specification sheets

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 data, which will simplify your project, but will also 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 Overview

Figure 1 below shows the basic datapath you will be implementing for Checkpoint # 1. You will build the SDRAM subsystem that will read and write values we supply into the SDRAM module. The black box provided will calculate whether or not you are storing and retrieving data properly – it will display a count of the times that your data is wrong on the LEDs.

This will be the first piece of the project. **You should read the data sheets and fully understand the operation of the 3rd party devices you will be interfacing with before you come to your lab section.**

In addition, you will be required to present your design at an initial design review with your TA at the start of this week’s lab. **Do not skimp or rush through your design, doing so will cost you many hours of frustration later this week!!**

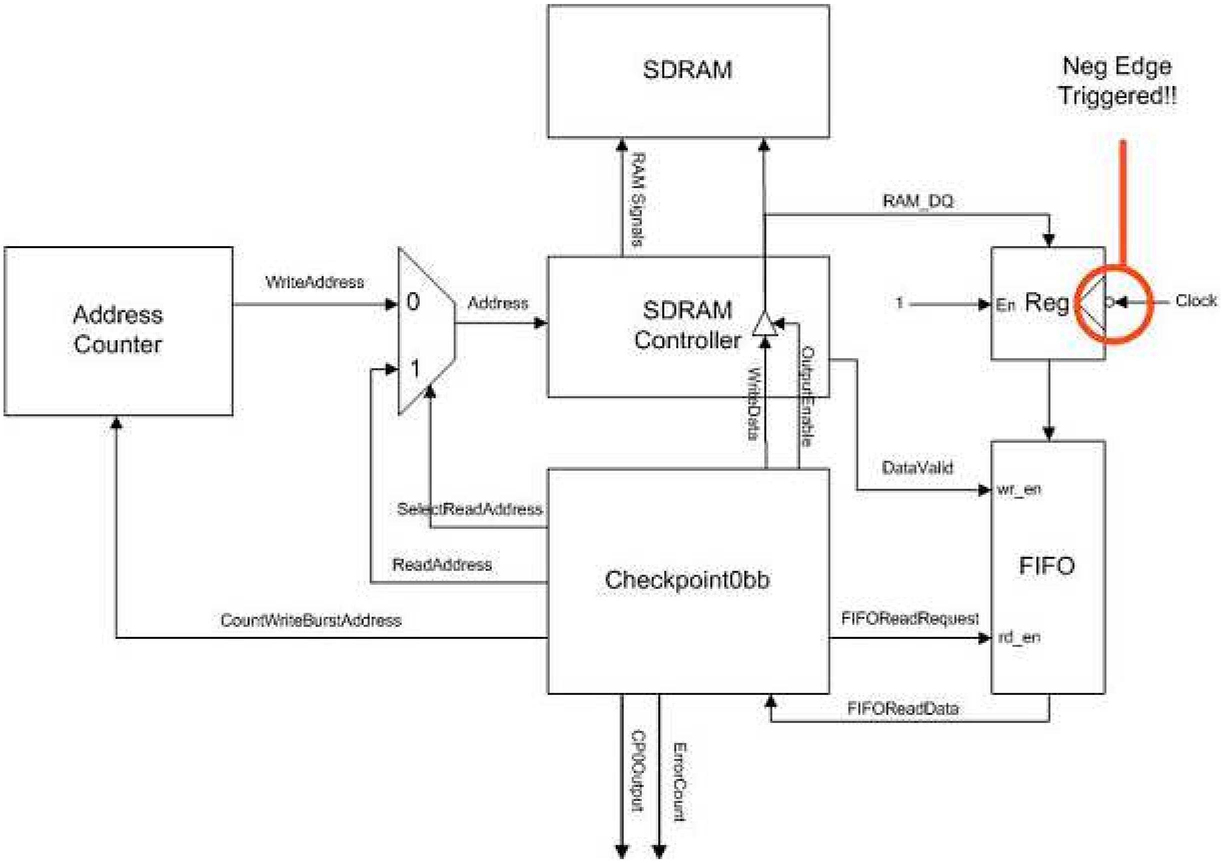

Figure 1: Checkpoint 1 Datapath

2.1 SDRAM Subsystem

The modules you will building for this checkpoint are 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.

The primary responsibilities of your SDRAM Control module are:

1. Initialize SDRAM
    a. Wait at least 100us after reset
    b. Issue an initialization sequence to properly set-up the SDRAM (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
    b. Your module must take care to ensure proper burst timing (See pages 46, 51, 53, 58 of the SDRAM Datasheet)

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.

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. FPGATop will be provided.
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 carefully.
    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 cannot pass this checkpoint without good testbenches
    - d. Make sure you understand tri-state and how to implement this in verilog.
        - i. You should bring as part of your design review the 1 line of code that will implement the necessary tri-state for this checkpoint.
5. **This will be the longest lab that you have ever done, 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. The port specifications are given below. The essential functions of your module are listed in section 2.0 , above. Your module must:

1. Initialize SDRAM
    - a. Wait at least 100us after reset
    - b. Issue an initialization sequence to properly set-up the SDRAM (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 (See pages 46, 51, 53, 58 of the SDRAM Datasheet)
    - b. Your module must take care to ensure proper burst timing

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.

You will be writing and reading 32-bit data to and from the SDRAM. There are two 16-bit SDRAM chips on the boards. The top 16 bits to should be written to one SDRAM chip and the bottom 16 bits to the other.

Note the special Tristate buffer shown in figure 1. This is needed because data written to the SDRAM is fed into the same RAM_DQ bus as data coming out of the SDRAM. The output of your controller on the RAM_DQ line should be high impedance (Z) when OutputEnable is not asserted.

For this project, we will set the burst length to be 8 and the CAS latency to be 2.

Because of the way data will be read in this project, it is not necessary to implement refreshes, though you are free to do so.

This may seem overwhelming at first, but in reality it's just another state machine!

Signal	**Width**	**Dir**	**Description**
`Clock`	1	I	System clock signal
`Reset`	1	I	Reset signal
`Ready`	1	O	Indicates that the SDRAM has finished initialization and is ready to accept read and write requests.
`Done`	1	O	Indicates that the SDRAM has just finished serving a read or write request. This should be asserted on the same cycle that the last piece of data is read or written in the burst.
`DataValid`	1	O	Indicates that the data on the RAM_DQ line is valid for a read.
`ReadRequest`	1	I	Indicates a read request (only high for one cycle).
`WriteRequest`	1	I	Indicates a write request (only high for one cycle).
`OutputEnable`	1	I	Tri-state enable signal to put WriteData on the RAM_DQ line.
`WriteData`	32	I	Data to be written to SDRAM.
`Address`	24	I	{RowAddress, BankAddress, ColumnAddress} Corresponds to the address that will be read from or written to.
`Mask`	1	I	Masking for reads and writes (assign to RAM_DQML and RAM_DQMH when reading/writing).
`RAM_CLK`	1	O	Read the Datasheet. (Assign to ~Clock)
`RAM_CLKE`	1	O	Read the Datasheet.
`RAM_DQMH`	1	O	Read the Datasheet.
`RAM_DQML`	1	O	Read the Datasheet.
`RAM_CS_`	1	O	Read the Datasheet.
`RAM_RAS_`	1	O	Read the Datasheet.
`RAM_CAS_`	1	O	Read the Datasheet.
`RAM_WE_`	1	O	Read the Datasheet.
`RAM_BA`	2	O	Read the Datasheet.
`RAM_A`	13	O	Read the Datasheet.
`RAM_DQ`	32	I/O	Read the Datasheet. (Note: This is declared as inout)

Table 1: Port Specification for SDRAMControl.v

4.2 AddressCounter.v

In this checkpoint, we will be writing and reading from the following section of SDRAM:

Bank 0, Rows 0-199, Columns 0-199

Your counter should start at Row 0 and Column 0 after the reset, and iterate through the appropriate rows and columns sequentially, then wrap back around to Row 0 and Column 0 when you've reached the end. For this checkpoint, your address counter will determine which address to *write to* in SDRAM. Our black box will read this section of memory to verify that the correct data was written to the right place.

You will need to modify and enhance this address counter in later checkpoints to correspond to exactly one "screen" worth of video data, but worry about that later ☺

Signal	**Width**	**Dir**	**Description**
`Clock`	1	I	System clock signal
`Reset`	1	I	Reset signal
`AddressOut`	24	O	{RowAddress, BankAddress, ColumnAddress} - RowAddress is 13-bits wide - BankAddress is 2-bits wide - ColumnAddress is 9-bits wide
`CountBurst`	1	I	Increment the column address by one burst worth of data.
`Bank`	2	I	Indicates which bank of SDRAM to write to (assign to 2'b00 for this checkpoint – You may need to change this for later checkpoints).

Table 2: Port specification for AddressCounter.v

Note that the CountBurst signal requires you to increment the address counter by one burst's worth of data!

4.3 CheckPoint0bb.v

This file will be provided for you as a black box. This module asserts all of the read and write requests to SDRAM that your controller must handle (we guarantee that these requests will never be simultaneously asserted). Also inside of this module is the logic that will generate the data you will write to SDRAM and verify that the data that you wrote was correct. It counts the number of times your data does not match the expected value. The error count should be 0 for a working SDRAM controller.

Signal	**Width**	**Dir**	**Description**
`Clock`	1	I	System clock signal
`Reset`	1	I	Reset signal
`RAMReadRequest`	1	O	Request a read from SDRAM (asserted for one cycle at a time).
`RAMWriteRequest`	1	O	Request a write from SDRAM (asserted for one cycle at a time).
`RAMMask`	1	O	Masking for reads/writes (wire to Mask in SDRAM controller).
`RAMReadAddress`	24	O	Address to read from SDRAM
`RAMDataToWrite`	32	O	Data to write to SDRAM
`RAMReady`	1	I	Ready signal from controller
`RAMDataValid`	1	I	DataValid signal from controller
`RAMDone`	1	I	Done signal from controller
`OutputEnable`	1	O	Tri-state signal to controller
`CountWriteBurstAddress`	1	O	Tell AddressCounter to increment one burst worth.
`SelectReadAddress`	1	O	Mux selector signal to determine which address to use.
`FIFOReadRequest`	1	O	Read enable signal to FIFO
`FIFODataCount`	2	I	DataCount signal from FIFO
`FIFOReadData`	32	I	DOut from FIFO
`Mode`	2	I	Set to 2’b00.
`ErrorCount`	32	O	Indicates number of errors.
`CP0Input`	1	I	Special signal used for check-off.
`CP0Output`	32	O	Special signal used for check-off.

Table 3: Port Specification for Checkpoint0bb.v

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 between two devices of different data rates. You will need to place this fifo in a position to buffer the data coming from the SDRAM.

Signal	**Width**	**Dir**	**Description**
`Clk`	1	I	The write clock signal
`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 4: Port Specification for fifo_sync32.v

4.5 Tips, Notes, and Caution

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

1. Design FIRST! Do not starts writing code until you fully understand the problem and how you plan to implement the solution.

2. After you design your solution, build small modules of code and test them individually. Bottom-up testing is essential at this point.

3. We have provided a file for you called mt48lc16m16a2.v. This is a simulation file for SDRAM. Make sure you test your controller against this file before going to board! It even gives you text feedback in the console! This will save you hours of waiting for Place and Route to finish.

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.

5. Make sure both you and your partner understand how all modules work and communicate with one another.

6. Make use of modules that we have previously given you. You should not be rewriting counters, or registers, or other pieces of code already provided for you from labs 1-5.

7. In Figure 1 there is a NEGATIVE edge-triggered register in between the SDRAM and the FIFO. Make sure that “~Clock” is sent to the Clock input of the register.

8. Remember that RAM_CLK should be assigned to ~Clock in SDRAMControl.v!

9. The important connections that you need to make are shown in figure 1. Study it carefully!

10. The checkpoint0bb writes a bunch of things to SDRAM and reads them all back out again, checking the number of errors in the process. You will get 0 errors from it if checkpoint0bb never gets to write or read anything in the first place. This means that getting 0 errors is not a sure-fire sign that your controller is working.