DDR RAM CONTROLLER FOR THE CYCLONE II FPGA

A Design Project Report

Presented to the Engineering Division of the Graduate School

Of Cornell University

In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering (Electrical)

By

Denis Pyatkov

Project Advisor: Dr. Bruce Land

Degree Date: May 25th

Abstract

Master of Electrical Engineering Program

Cornell University

Design Project Report

Project Title: DDR RAM Controller for the Cyclone II FPGA

Author: Denis Pyatkov

Abstract: This project implements a DDR RAM controller at the RTL level for the Altera Cyclone II FPGA on a DE2 development board. A PCB board was also designed and fabricated to provide an interface between the FPGA and the memory chip. DDR RAM is a widely used memory standard designed to provide storage and retrieval of large amounts of data at very high clock frequencies. The main distinction between this type of memory and other standards is that data is read and written on both rising and falling edges of the input clock. The motivation behind choosing this project was my desire to complement my classroom experience in designing digital systems on the FPGA with a real world application. This standard is used in many different digital systems, including high-performance graphics cards and modern personal computer systems. The challenge in this project is to develop a state machine capable of handling inputs at high clock frequencies while still maintaining an acceptable level of performance. The controller hardware was implemented using Verilog and the PCB board was designed using the ExpressPCB design software.

Simulation and verification was done in a Linux operating system environment. iVerilog was used to synthesize the code and run a simulation, and GTKWave was used to observe the output waveform and perform debugging. Micron provided a Verilog model of the DDR RAM, which was used inside a testbench along with DDR control module. The model provided feedback as to whether the DDR chip was receiving the proper input signals. A design flaw in

the Terasic DE2 board prevented me from implementing my top file on the Cylone II FPGA.

Report Approved by

Project Advisor ____ Date: ___________

Executive Summary

Before work could begin on implementing the design requirements, background knowledge of Double Data Rate (DDR) RAM was established. Research was conducted on JEDEC DDR1 RAM standard as well as the Micron DDR1 memory module specifications. In particular the Micron implementation of the JEDEC standard for a DDR1 memory module was a starting point for the project. From this research, various considerations for the implementation of a DDR RAM controller were observed. Once a suitable foundation about specific requirements was established, work began on designing a hardware model for the Altera CycloneII FPGA.

Before a behavioral Verilog was implemented, much work was put in to designing the necessary components and state machines required to create the memory controller for DDR1 RAM. It was understood that the task of designing this controller would be considerably difficulty than first expected and therefore more thought needed to be put into the design process before an actual model was conceived. A main consideration throughout the initial design process was ease of use, given that the results of this project would potentially be used by students in future years.

Once the logical model was finalized, a behavioral Verilog model of the controller was implemented. Using design tools on a Linux system, including Icarus Verilog and GTKWave, a simulation of the controller was implemented. Micron's DDR RAM Verilog model was used to verify that the controller was

interfacing properly with the RAM in simulations. The Micron DDR model was an invaluable tool in debugging the controller interface.

A testbench for the DDR controller was implemented utilizing the Micron Verilog model. This testbench can be used to test future modifications to the controller if additional features are required by the user. After a simulation model of the controller was fully functional, an attempt was made to modify it to work on an Altera Cyclone II FPGA residing on a Development and Education (DE2) board. Unfortunately due to a design flaw in the Terasic DE2 board the final version of the controller could not be interfaced with the DDR RAM chip. ExpressPCB, a company which specializes in fabricating PCB boards for small design projects, provided software necessary to design the PCB on which a DDR RAM chip was placed.

Table of Contents

Background

Double data rate (DDR) RAM is a widely accepted class of memory storage used in many computers today. It allows for data of sizes up to 64-bit words to be transferred to be read and written from the RAM module on both the positive and negative edges of the clock. While there have been three different iterations of the DDR standard, with the current being DDR3, the first, commonly referred to as DDR1, allows for a maximum input clock frequency of 200 MHz and a maximum storage size for one DDR Ram module of 1 GB. A typical module is shown below (figure 1).

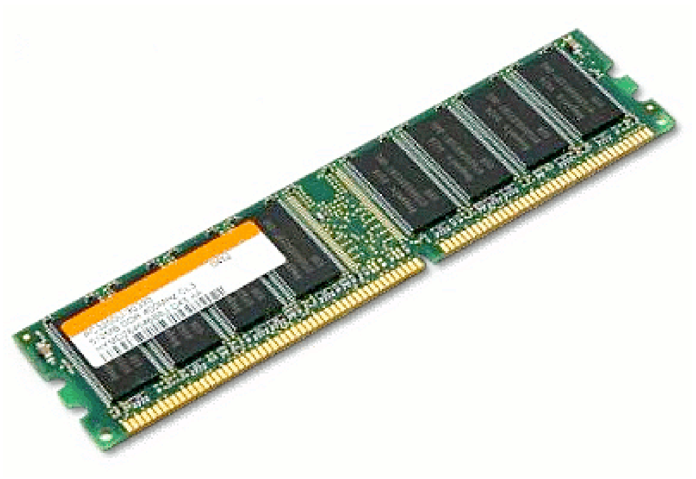


Figure 1. DDR RAM module with a DIMM socket interface

It also interesting to note that the price for brand name 1 GB of DDR RAM rated at a clock frequency of 133 MHz is low as $50. The combination of cheap, but fast memory makes DDR RAM an ideal solution when extra memory capacity is needed for any hardware application. Indeed, the numerous variants of DDR RAM are already used in many devices such as the latest graphics processing cards.

Design Requirements

The purpose of this design project is to develop a fully functional interface between the Altera CycloneII FPGA and a DDR1 RAM module running at an external clock of 133 MHz. The CycloneII FPGA resides on the DE2 board which contains, among other hardware, an expansion slot with 72 free expansion pins available (figure 2). A physical connector between the RAM module needs to be designed, along with the appropriate digital hardware drivers. The hardware will be freely available for any Cornell University students to use for projects involving the DE2 board. It is also hoped that in the process of doing this project, I will learn the tools used in industry for such designs, including the Linux development environment and associated software tools.

While overall the goal will be to successfully interface the RAM with the CycloneII FPGA, ease of use will also be a consideration. A student should be able to integrate this design project into his or her own without much trouble. The following requirements will be addressed by the project:

- Set up a solid developing environment for the development and simulation of the RAM controller
- Develop a digital hardware module written in Verilog to allow the user to easily issue read and write and commands to memory
- Allow several modes of memory access to give the user maximum flexibility in designing applications which use DDR RAM
- Provide physical interface between a single DDR RAM chip and the Altera DE2 board

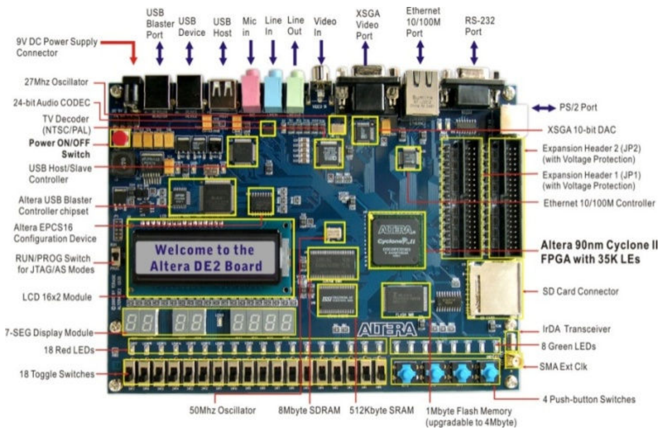

USB Blaster Port
USB Device
USB Host
Mic in
Line In
Line Out
Video In
XSGA Video Port
Ethernet 10/100M Port
RS-232 Port
9V DC Power Supply Connector
27Mhz Oscillator
24-bit Audio CODEC
TV Decoder (NTSC/PAL)
Power ON/OFF Switch
USB Host/Slave Controller
Altera USB Blaster Controller chipset
Altera EPCS16 Configuration Device
RUN/PROG Switch for JTAG/AS Modes
LCD 16x2 Module
7-SEG Display Module
18 Red LEDs
18 Toggle Switches
Welcome to the Altera DE2 Board
PS/2 Port
XSGA 10-bit DAC
Expansion Header 2 (JP2) (with Voltage Protection)
Expansion Header 1 (JP1) (with Voltage Protection)
Ethernet 10/100M Controller
Altera 90nm Cyclone II FPGA with 35K LEs
SD Card Connector
IrDA Transceiver
8 Green LEDs
SMA Ext Clk
4 Push-button Switches
50Mhz Oscillator
8Mbyte SDRAM
512Kbyte SRAM
1Mbyte Flash Memory (upgradable to 4Mbyte)

Figure 2. The DE2 board with a CycloneII FPGA

Range of Solutions

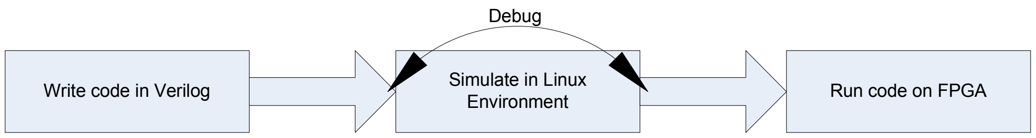
Unlike previous designs implemented in the classroom, DDR RAM is a complex standard with very tight timing requirements. It is almost impossible to debug this memory controller without some kind of simulation and testbench. Simply writing up a behavioral model in Verilog and then downloading the code to the FPGA is not the most efficient way of implementing such a system. In fact, this approach can be quite painful as I learned early on. Even with the right tools and the right mindset, the process takes a long time and requires patience and perseverance.

Digital Hardware Implementation

In industry and in some classes I have taken, much of the hardware development takes place in a Linux environment. This environment provides some very handy tools for a digital hardware designer. Two tools of interest to me were the verilog synthesizer and simulator, iVerilog, and the waveform viewer GTKWave. iVerilog is used to compile the code, run the simulation, and generate a waveform. GTKWave reads the waveform file and allows me to view parts of it for debugging and verification purposes.

Micron, A DDR chip manufacturer, provides a behavioral model of the chip free of charge for download off their website. This allows me to interface my controller with a verilog model of the DDR chip and run simulations to see whether my code works or not. As an added bonus, verifying the controller on a Micron model also ensures that the controller will work with a Micron-fabricated DDR chip. The DDR model also provides feedback and describes the exact error that the controller performed, such as an improperly timed DQS signal.

Combining the Linux environment, my code, and the Micron DDR model, I was able to successfully implement the RTL logic to control a DDR RAM chip. Before any code was ported to the Altera QuartusII software package, I verified my design by simulating it in iVerilog. This was done to avoid having to debug my controller while it was running on the FPGA, which as mentioned earlier is a tedious and intractable task. Below is a logical flow of my design process:



The controller is kept separate from the datapath. It is up to the user to decide how to interface the data IO of the DDR RAM with his or her design. The only goal of the controller is to allow the user initialize the RAM properly with specific operating parameters and send read, right, active, and refresh commands to the DDR RAM. This was done to simplify the state machine of the RAM controller and to allow as much flexibility as possible to the user. The user can operate the RAM with as much complexity or simplicity as the user requires.

Physical Hardware Implementation

DDR RAM modules come in many different configurations. One standard module has a PCB with a DIMM interface and several DDR RAM chips. These are designed to handle words of up to 64 bits to be transferred at every clock edge. Adding in the requirement control signals and clocks, the number of pins required to use this interface exceeded the 72 pins provided at the DE2 expansion headers. Instead, I decided to use one Micron 256 MB DDR chip with a 16-bit data bus, which required 32 input/output (IO) pins. Since no interface between the DE2 board and a DDR RAM chip existed, I had to design one from scratch.

On the advice of my advisor, I chose the ExpressPCB software package to design a schematic and a PCB layout of my board. ExpressPCB has a quick turnaround time. For my board, it took them exactly a week to fabricate the design and ship it. Due to the large number of signal wires that needed to be routed in a small area, I chose to use 4 layers. Two layers were dedicated to power and ground. The top and bottom copper layers were used to route signals from the DDR chip to the 40-pin headers that mated with the DE2 board.

A schematic of the hardware was designed and linked to the PCB layout. This greatly eased the process of routing the traces to their correct pins at the expansion headers. Also, any changes that had to be made to make to accommodate for spacing constraints (such as two traces being too close to each other), could be made in the schematic first and then linked to the layout. Due to the many changes that needed to be to the board it was finally ready for fabrication, this process of modifying the schematic and linking the changes to the layout was invaluable.

DDR RAM Design Parameters

Reading and writing to and from the RAM module is burst oriented. A location for reading or writing is selected via commands sent to the DDR RAM controller on the module. In addition to the location, the length and sequence of accesses is selected. Once the appropriate commands have been received and the appropriate signals asserted, data is transferred via a bi-directional bus. A table in the appendix lists all the pins to control the RAM (figure A1). For proper operation the device must initialized and specific electrical requirements must be met:

- VDD and VDDQ are driven by a single output at 2.5V ± .2 V
- VTT is limited to 1.35 V
- VREF tracks VDDQ/2

Operational Parameters

The mode of operation for the RAM module is selected by setting the appropriate bits in the Mode Register. The Extended Mode Register can also be set. However, as of this time as defined in the standard it does not affect any useful operational parameters. The following parameters can be defined in the Mode Register:

Parameter	**Description**
Burst Length	The number of words read or written to memory in a read or write command. Can be 2, 4 or 8 words.
Burst Type	Determines whether the burst is sequential or interleaved
CAS Latency	The clock cycle latency between the issuance of a read command and when the data is ready on the data bus. Can be 1.5, 2, 2.5, or 3

Operating Mode	Selects normal operating mode or a vendor specific mode used for testing

When a MODE REGISTER SET command is issued to the normal Mode Register, the following bits in the address bus determine its contents:

- A[2:0] – burst length
- A[3] – burst type
- A[6:4] – CAS latency
- A[13:7] – operating mode

An example combination of burst length and type is 8 words and sequential starting with the 0th address. In this case, the words written or read from memory will follow the pattern 0-1-2-3-4-5-6-7. It is crucial the address register is set with the appropriate bits when a MODE REGISTER SET command is issued.

Commands/Instructions

Commands (also referred to as instructions) are issued via the CS, RAS, CAS, WE and address pins. All pins except the address pins are *active high*. There are a total of eight distinct commands the DDR RAM module is designed to accept. The following table describes each command and indicates which bits are required:

Command	**CS**	**RAS**	**CAS**	**WE**	**Addr**	**Description**
NOP	H	X[1]	X	X	X	No operation
ACTIVE	L	L	H	H	Bank/Row	Selects the bank and row of a read or write access
READ	L	H	L	H	Bank/Col	Selects bank and column for a read and starts read

[1] X – Denotes a don't-care input

WRITE	L	H	L	L	Bank/Col	Selects bank and column for a write and starts write
BURST OFF	L	H	H	L	X	Ends a read burst only if autoprecharge is disabled
PRECHARGE	L	L	H	L	Mask	Determines which banks are automatically precharged depending on the mask
AUTO refresh	L	L	L	H	X	Performs auto refresh if CKE is high when command is issued
Mode Register Set	L	L	L	L	Mask	Sets the Mode Register according to the mask

While data is transferred on both the rising and falling edges of clock, commands can only be issued on the rising edge.

Initialization

Following power up of the RAM, the device is initialized in a predefined manner. After the device is given power, there must be a 200 µs wait before commands can be executed. After 200 µs a NOP command is issued and the CKE input is driven high. Next, a PRECHARGE ALL instruction is applied followed by a MODE REGISTER SET command to the extended mode register. Another MODE REGISTER SET command is applied to the normal mode register. These two series of commands specify the operating parameters of the RAM, such as the burst length and sequence.

Following the setting of the mode registers, there must be a wait of 200 cycles before further commands can be applied. After 200 cycles, the device must be put into the idle state by another PRECHARGE ALL command. Once in idle, two AUTO refresh commands are issued followed by a MODE REGISTER SET command with the reset bit low. This will allow the normal operating parameters of the device to be established without resetting. The following summarizes the initialization procedure:

- Wait 200 µs, THEN
- Issue NOP with CKE pin high, THEN
- Issue PRECHARGE ALL instruction, THEN

- Issue MODE REGISTER SET for the extended mode register, THEN
- Issue MODE REGISTER SET for the normal mode register, THEN
- Wait 200 cycles, THEN
- Issue PRECHARGE ALL instruction, THEN
- Issue two AUTO refresh commands, THEN
- Issue MODE REGISTER SET for the normal mode register with reset bit low,
- Device is ready for normal operation

Note that the Mode Register can be reset at any time after initialization. The user can redefine his operation parameters after initialization.

Prepping for a read or write

Before data can be written to memory or read from memory in a specific row inside a bank, that row must be opened. This is done by issuing an ACTIVE instruction along with the bank and row to be opened, specified by bank bits B[1:0] and address bits A[13:0]. If another row from the same bank is to be accessed next, the previous row in that bank must be closed. This is done by issuing a PRECHARGE command to that row. A row in a different bank can be issued an ACTIVE command after the current row in the current bank is activated without a PRECHARGE command. This reduces overhead when accessing data from separate banks because no PRECHARGE command is needed after the initial row is opened.

There is a minimal time interval between when a command can be issued to another bank, called tRRD. There is also a minimal time interval between when a bank is issued an active command and when READ or WRITE can be issued, called tRCD. For a DDR RAM module rated at 133 MHz, tRRD and tRCD is defined as follows.

- tRRD – 2 cycles
- tRCD – 3 cycles

The following timing diagram illustrates issuing ACTIVE commands followed by a memory access to banks x and y (figure 3). Note that tRRD and tRCD are the same length, although that is usually not the case.

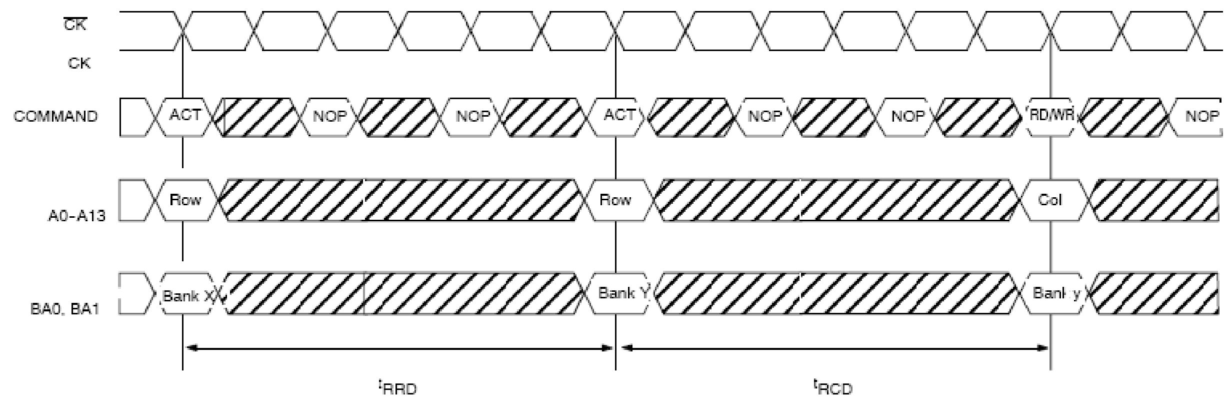


Figure 3. Timing diagrams for read/write prepping

Reading

Once the desired bank and row are activated and a tRCD number of cycles have passed, a READ command can be issued. Along with the read command, the starting column address is asserted as well as the auto precharge bit (A10). If the bit is high, the bank will automatically precharge ("close") itself. This eliminates the need to specifically issue that command after the read is completed. However, if a long stream of columns needs to be read from the same bank and row, it would not make sense to have automatic precharge enabled as this would increase overhead. If the bank is not precharged immediately after the read, a different row on the same bank will not be ready for another READ or WRITE command.

READ commands can be issued sequentially without waiting for data from the first READ command to complete bursting. If two READ commands are issued sequentially, the read data from the first command will be ready for bursting according to the CAS latency settings in the mode register. The following timing diagram illustrates sequential read accesses with a CAS latency of 2 (figure 4). Note that READ commands can be issued in such a way that each new piece of data is from a different column, rather than a sequential one. This provides true random memory access for DDR RAM.

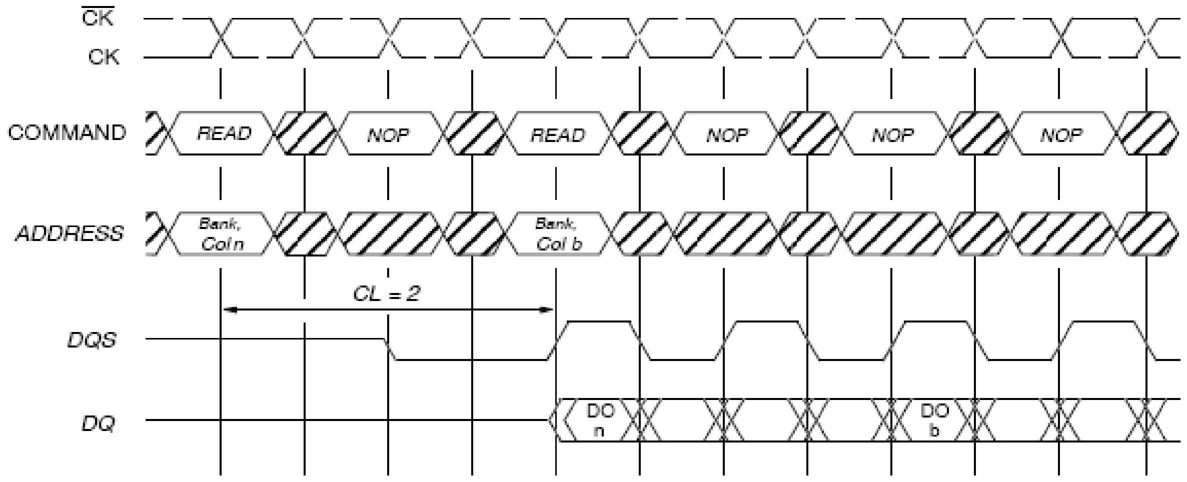
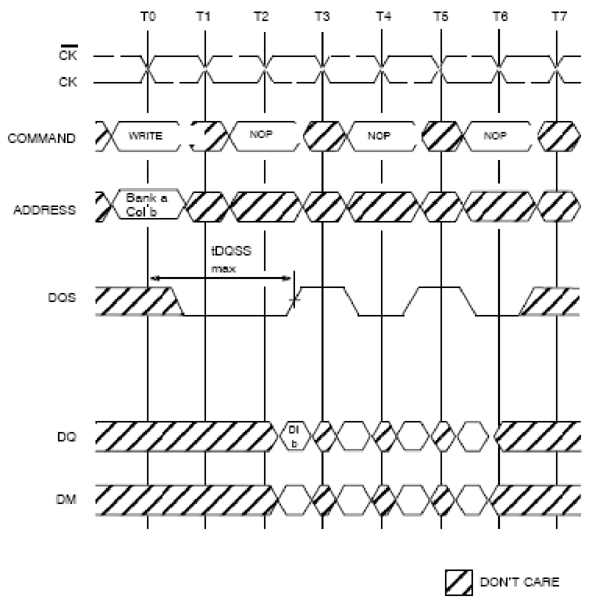


Figure 4. Sequential read commands timing diagram

Note that the DQS pin is an output here. It asserts itself low after the READ command is issued and high coinciding with the first piece of data ready to be read. A PRECHARGE command must be issued 1 cycle before the last burst of read data. This is required to allow an ACTIVE command to be issued to the same bank. The active command to the same bank can only be issued after a tRP cycles have passed after the PRECHARGE command is issued.

Writing

As with the READ command, the WRITE command requires an ACTIVE command for the target bank and row. Similarly, to write to a new row on the same bank, the bank must be precharged. The DQS signal is a don't care input until the WRITE command is issued. On the same clock cycle, DQS must go low. This is called the "write preamble". The first valid data written can be issued after the first negative edge of clock cycle during which the WRITE command was issued. Data is written to RAM on the rising edge and falling edges of the DQS pin. DQS must be properly asserted within .75 to 1.25 clock cycles after the WRITE command is sent. The following timing diagram illustrates the operation of the DQS signal with respect to the timing requirement, called tDQSS (figure 5).



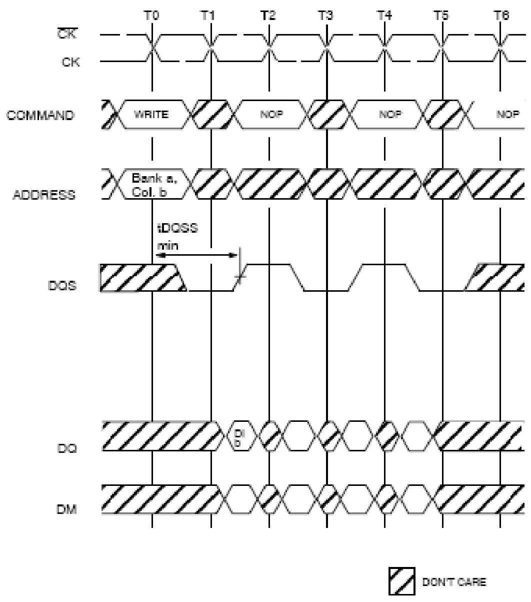


Figure 5. Issuing a write command

READ commands to the same bank and row can be issued at tWTR clock cycles after the last word of data has been written.

WRITE commands can be issued sequentially. To write to sequential columns in the same row, the next WRITE command must be issued when half the data of the previous write has been written. This is illustrated in figure 6.

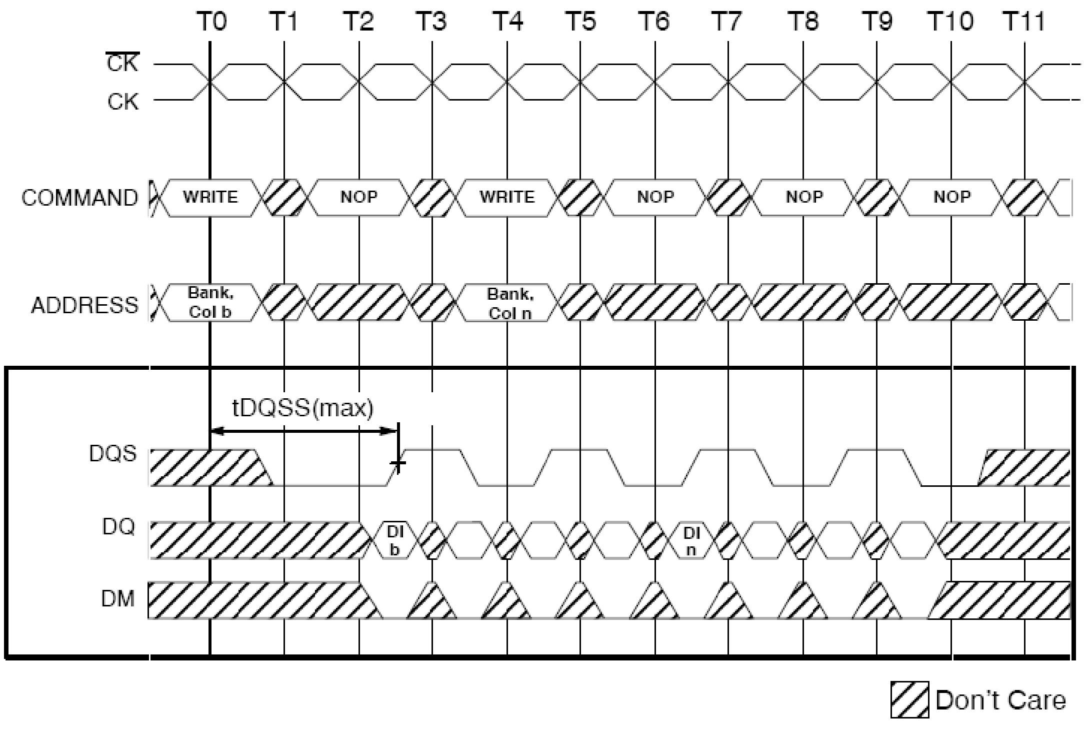


Figure 6. Sequential WRITE commands

Physical Hardware Interface

The physical link between the CyloneII FPGA and the DDR RAM is a major part of this design project. An improperly designed interface will cause the RAM malfunction, possibly destroying the chip itself. The Micron DDR chip used in this project is 256 MB in size. It has a 16-bit wide databus and requires specific reference and Vdd voltages at several pins on the chip. 16 bits is the largest databus possible on current DDR1 chips. In order to attain a higher read and write capacity, DDR RAM module manufacturers put several DDR chips on a PCB and access them in parallel (figure 1).

Due to the high speed interface, the DDR RAM board was designed to mate directly with the DE2 board using two female 40-pin headers. There were no ribbon cables used to connect the board to the expansion headers. This requires precise placement of the header through-holes relative to each other to ensure that the board mates successfully with the DE2 headers. Specifically, the pin 1 of the top header was exactly 20mm away vertically from pin 1 of the bottom the header. As I will explain later, this posed a difficult design challenge when doing the PCB layout.

The Micron DDR chip requires 8 pins to be connected to a Vdd voltage of 2.6V and 1 pin to be connected to a Vref voltage of 1.3V. To provide power I chose to use two Texas Instruments PTH 08080w miniature power modules. Each can be fed an input voltage of 5V and output a separate voltage which can be controlled by a voltage select input on the chip. This is convenient because the expansion headers on the DE2 board have +5V output pins. The following figure is a pinout diagram of the modules:

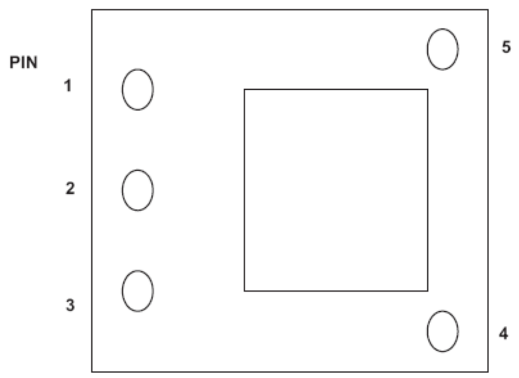


Figure 7. Pin out of the PTH power module

Pins 1 and 2 are input voltage and ground respectively. Pin 3 is the output voltage and pin 4 is the voltage select. A resistor attached to pin 4 determines the output voltage. The resistance required is determined by the following equation:

$$R_{set} = 10\ K\Omega * \frac{.891\,V}{V_{out} - 0.9V} - 1.82K\Omega$$

Therefore in order to get the required output voltages of 2.6V and 1.3V, resistances of 3.40KΩ and 20.5KΩ were used respectively.

Schematic Design

As mentioned earlier, the ExpressPCB design software was used to create a schematic and a PCB layout on the board. The schematic was the first step in the board layout process. The schematic is a circuit-level view of the board. It contains all the capacitors, resistors, chips and headers that require some type of routing in the PCB layout. An example component in the ExpressPCB schematic software can be seen in figure 8.

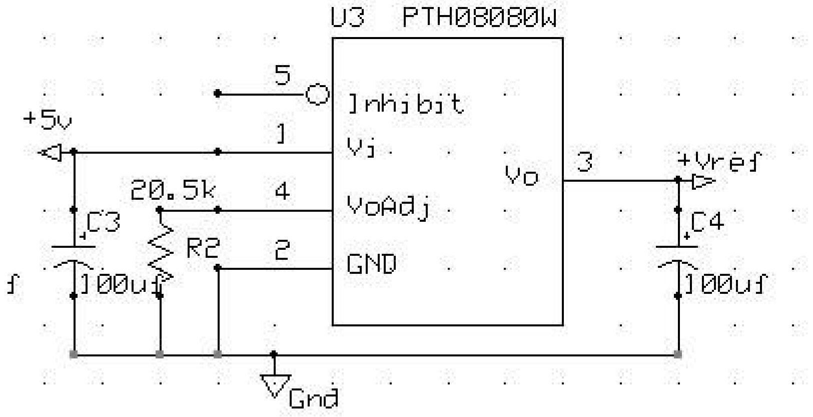


Figure 8. Schematic Entry for Power Module

This component is the TI power module which generates the required 1.3V reference for the DDR chip. The component is not part of the standard library of symbols that come with the software. This component was designed by manually drawing the rectangular shape and pins. It was saved as a custom library component. Note that the output at pin 3 is called +Vref. The corresponding pin that requires a Vref on the DDR Chip symbol is attached to a wire also called +Vref. Also note that the component name U3. This corresponds to a component name in the PCB layout. This allows the schematic to be linked to a PCB layout.

The other major hardware component on the PCB is the DDR RAM chip itself. In addition to the 8 power and reference pins, there are 8 ground pins and 43 IO pins. There are 63 pins total, but some of them are not connected to anything. The IO pins are routed directly to the two 40-pin expansion headers. As with the TI power module, the schematic software did not have a library component for the RAM chip. The 66-pin component was also designed from scratch.

As seen on the schematic in the appendix, two types of capacitors were used to bypass power and ground. .1 µF capacitors were used for the DDR chip and 100 µF electrolytic capacitors were used for the power supply. These capacitors helped reduce voltage fluctuations at the power inputs of the RAM chip.

PCB Layout

The PCB design was one of the more difficult aspects of the project. The major constraint was the number amount of signal wires that needed to be routed in the small amount of space available. The Micron DDR chip comes in 66-Pin TSOP package. It is a surface mount package that has a pitch of .65mm between each pin. 43 pins needed to be routed from the chip to the expansion header through-holes. Because the header through-holes needed to be placed in a specific location relative to the two headers on the DE2, I could not exploit tricks of moving the headers around to make routing easier. For example, I could not place headers on either side of the DDR chip and route signals on the two sides of the chip in opposite directions.

The ExpressPCB software did not have a layout component for the surface mount pads that the DDR chip is soldered on to. The layout out of the pads was customized (figure 9). The Micron DDR RAM datasheet provides the necessary information to design component in the PCB software (figure 10). A custom symbol was also designed the TI power module. This was significantly easier than designing the custom DDR component, given that the tolerances in the TI 5-pin package were more lenient.

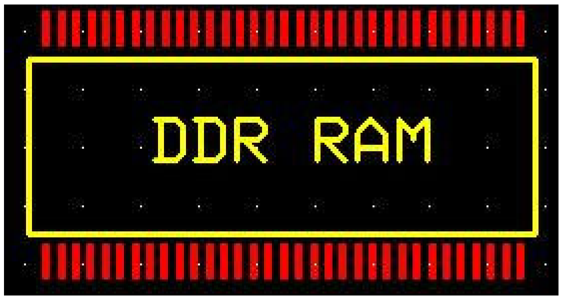


Figure 9. DDR RAM custom symbol

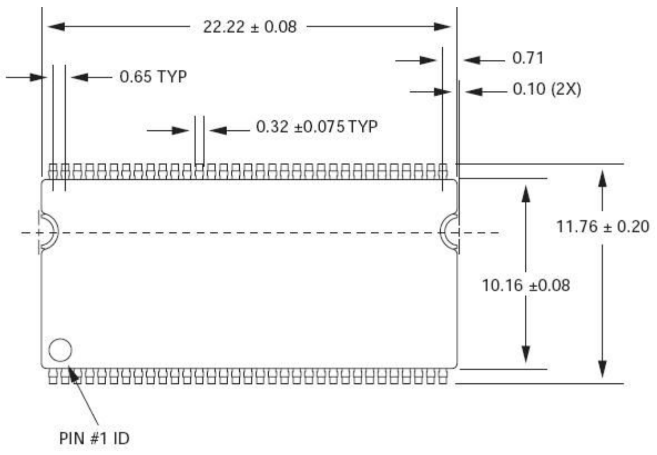


Figure 10. Pin layout

The most difficult part came when traces had to be routed from the DDR solder pads to the expansion header through-holes. The PCB schematic went through several iterations before a final routing was established. One symmetry I could exploit was traces on opposite sides of

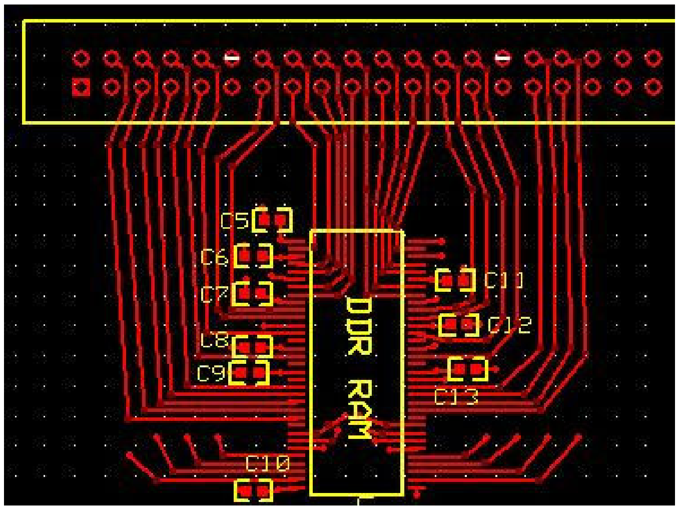


Figure 11. Trace Routing

the chip to opposite sides of the connector (figure 11). As you can see, traces on the left side of the chip are routed to the left side of the connector, and vice versa for the right side. In the end this became the most efficient method of routing. The figure only contains traces on the top copper layer. The traces at the bottom-left, bottom-center, and bottom-right are sent to the bottom copper layer through vias. These are then routed to the top connector, exploiting the symmetry similarly. The final outcome of the design, including pictures, layout and schematic is shown in the appendix.

Digital Hardware Interface

Designing and debugging a high-speed memory controller is inherently difficult without a logical design process. As mentioned earlier, one of the main goals of this project was to set-up a development environment to ease process of debugging my code. One consideration about the hardware that had to be addressed early on was the speed of the state machine which controlled the DDR RAM. The state machine had to cause all of its outputs to change before the next clock tick. While this was not a primary concern in my class projects, the RAM ran much quicker than anything else I had worked with. Much thought was put into how to design the controller as efficiently as possible and implementing it took up a majority of my time early on in the project.

Development Environment

After having worked on a hardware development team as part of an internship, I knew that the preferred development environment for digital hardware engineers is Linux. It provides many convenient tools, including the ability to execute shell scripts to compile a large project quickly. Indeed, some of my classes also used Linux for similar reasons. One of the main advantages is that the operating system, and simulation and waveform software is free of charge and is well documented on-line. If I had any sort of issues I could find a website or bulletin board that could address it. The development environment can be broken down into three components: the virtual machine, iVerilog, and gtkwave.

Virtual Machine

Conveniently, I did not need to have the Linux operating system installed on my machine to enjoy the benefits of Linux. Instead, I installed the VMWare player which could run a Linux virtual machine, that for all intents and purposes had the same functionality as a full Linux install. This cut down on the hassle of having to repartition the hard-drive on my laptop to accommodate for a new OS installation.

The particular virtual Linux operating system I used was called Fedora Core 7. It was one of the standard Linux OS's out there. There already was plenty of support available online and it was compatible with the Eclipse IDE. The Eclipse IDE is a free Integrated Development Environment. While it did not support Verilog/HDL projects initially, a plug-in was downloaded to add this functionality. The code editor in Eclipse can recognize Verilog syntax and was useful for some debugging purposes.

iVerilog

iVerilog is a free synthesis and simulation tool for Linux (or Windows) which supports the IEEE-1364 verilog standard. It can compile source code, run a simulation and generate an output waveform file. The program can be installed in the Linux Virtual Machine so that the iVerilog executables can be run from any directory using the command-line. Verilog code is compiled using the following command:

```
iverilog [file1 file2…] -DSIMULATION -I../ -o tmp.vvp
```

The file tmp.vvp is the synthesized verilog code, including simulation parameters. In order to run a simulation, the following command is issued

```
vvp tmp.vvp
```

This will also generate a waveform output file which can be read by a waveform viewer. Since this project contains many verilog files that have modules used in the top file, it was necessary to write a shell script to compile all of the files for simulation. The script was very simple. It defined *vfiles* variable that contained the location of all files used in the project:

```
vfiles="DDR_TOP_tb.v \
      ddr.v\
      altddio.v\
      ALTERA_DEVICE_FAMILIES.v\
      ../DDR_TOP.v \
      ../ddram_ctrl.v\
      ../ddr_init.v\
      ../ddr_read_write.v
      ../mux_2dir_reg.v\
      ../mux_4dir_reg.v\
      ../reg_1stg.v"
```

Rather than inputting all of the files needed for the simulation manually, the locations were defined in the script. A full version of the final top module simulation script is available in the appendix.

GTKWave

After simulation, some of the debugging was done with GTKWave, a waveform viewer. The software has many convenient features to identify problems in the simulation. It allows the user to select signals from any module involved in the simulation and see their trace. The signal selection can be saved so that each new instance of GTKWave can read a signal file which loads all the signals used previously. Figure 12 contains a sample view of GTKWave.

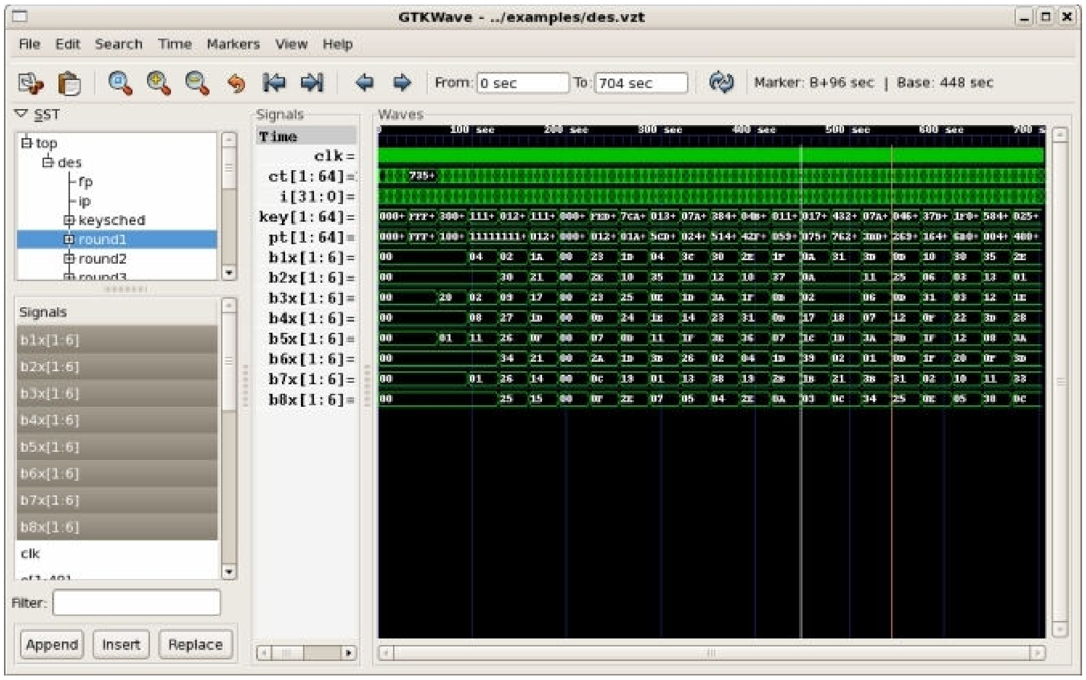

GTKWave - ../examples/des.vzt
File Edit Search Time Markers View Help
From: 0 sec
To: 704 sec
Marker: 8+96 sec | Base: 448 sec
SST
Signals
Waves
Append
Insert
Replace

Figure 10. Sample GTKWave View

The signals can be selected from the left hand pane. The top pane allows the user to change the time increments. Values for specific signals can be searched using the search menu. The current waveform view can also be saved as a postscript (.ps) file which can be converted via a command-line command in Linux, ps2pdf

Digital Hardware Implementation

The most important part of the project was the implementation of the DDR Controller, initialization module and top file. The top file contained the *DDR_TOP.v* contained the interface between the controller, initialization module and the IO interface between the top file and the DDR RAM chip via the general purpose input output (GPIO) headers on the DE2 board. Even though the end-user is responsible for creating an interface between the databus of the DDR chip and his or her design, a sample read/write module was created to show how to do this. The sample read/write logic is placed inside the top file along with the DDR controller and initialization modules.

DDRAM_CTRL.V

The ddram_ctrl.v module controls the DDR RAM chip by setting the ras, cas, and we output bits in the top file as well as the address bits which control which row and column are read or written to. The *rcw_o* output contains the three bits. The control module receives command inputs from the top file and issues commands to the DDR chip via its state machine which determine the modules outputs. In all cases except when an extended command or idle command is issued, *rcw_o* contains values other than *3'b111*. The *addr_o* output is connected to the address output bits in the top file and is used when both an extended command and normal command (i.e. read, write, activate) is being processed. The *done_o* is a single bit flag used by the initialization module. It is activated when the controller finishes processing an extended command.

As was hinted earlier, the state machine which sets the control bits (ras,cas,we) and address bits for the DDR RAM chip was designed with speed in mind. A clever solution was implemented to ensure that the state machine was always in the correct state before the next rising edge of the DDR clock. The outputs of the controller are simply a subset of the bits that encode the state. This means that very little computation is required during each state. The only thing that occurs after a state transition is the calculation of the next state, which is in fact a few simple case statements.

After reset the state machine is the idle state and issues NOPs to the DDR RAM. The next state is selected by user the user issuing a NOP, read, write, refresh, activate or extended command. An extended command is issued by setting *cpu_cmd_i* input to 0. In this case the *arb_cmd_i* input, which determines whether the user wants to issue a NOP, read, write, refresh, activate is ignored. In the extended command state, a mux selects to proper address bits to input into the DDR RAM control so that the user can define operating parameters such as CAS latency. Since the extended command takes 1 cycle to process, the next state after an extended command is issued is *extdn* which sets the *done_o* flag to 1, indicating to the initialization state machine that an extended command has been processed. The following explains what happens when a user issues a non-extended command. Note that in the idle state the controller issues a NOP for every cycle that it stays in the idle state.

- **Idle** (*arb_cmd_i = 00*) – The controller issues a NOP for one cycle and goes back to the idle state
- **Refresh** (*arb_cmd_i = 01*) – The controller issues a refresh command to the DDR RAM, goes into the *waitrfdn* state. Waits 5 cycles and goes to the idle state.
- **Write Activate** (*arb_cmd_i = 10*) – Issues an activate command for a row and bank and issues 2 NOPs before going into the write state. The controller does not allow sequential activate commands to enable writing to multiple banks in one write burst.
- **Read Activate** (*arb_cmd_i = 11*) – Issues an activate command for a row and bank and issues 2 NOPs before going into the read state. The controller does not allow sequential activate commands to enable reading to multiple banks in one write burst.

In both the read and write states, the controller waits until the read or write burst finishes. This is determined by the burst length the user selects when the mode register is set. If the burst length is 4, then the controller will wait 2 cycles (because it takes 2 full cycles to read or write 4 bursts). This is done by initializing a counter *clmdn* which counts down until the burst has finished. During the wait NOPs are issued to the DDR RAM. After the wait, the controller goes into the idle state and waits for its next command. The read and write state also mux in the starting column of the read or write burst via the amxs select bits. The encodings for each state and state transitions can be found in the appendix. Note that the controller does *not* set the bank address bits. Those are set by the user when issuing an activate or extended command.

DDR_INIT.V

An initialization module which links to the controller was implemented as RTL logic. This module implements the initialization procedure outline in the DDR Design Parameters section of this report. Due to several inconsistencies between the JEDEC standard and the Micron datasheet, the initialization module instructs the controller to issue 10 cycles of NOPs after each step during the initialization. This additional step was tested in the verilog model of the DDR RAM and verified to work properly.

The initialization module locks out the read/write state machine from issuing commands via the *init_active* signal. The state machine is turned on when reset is active high. After the reset, the signal goes high until initialization is complete. The state machine issues extended commands to the DDR Controller in the proper sequence. The two important states that determine the operating parameters of the chip are *st_setmr* and *st_setmr*:

- **st_setemr:** sets the extended mode register. Currently there is no support for extended modes of operation in DDR1 standard.
- **st_setmr:** sets the mode register for the following operating parameters
    - Operating mode: Normal
    - CAS Latency: 2
    - Burst Type: 0 – Sequential
    - Burst Length: 2

Note that the CAS latency is set to 2. The controller is *not* designed to handle other CAS latencies. The controller will always insert two NOPs and then start read burst.

DDR_READ_WRITE.V

Although the end user is responsible for providing a datapath based on whatever the user needs the DDR controller for, a sample module is provided to show the user how to properly issue reads or writes. The burst length is set to 2. Therefore for each read or write command two 16-bit words are processed on the IO ports. The read/write state machine is designed in a similar manner as the DDR controller state machine. The encoding for each state is also part of the output of the state machine.

The read/write module issues two writes followed by two reads consecutively. After the reset is command is initiated, the state machine waits for the initialization of the RAM to complete. This is signaled via the *init_active* flag from the initialization module. There are two read states and two write states:

- st_wr1: Writes 0034 starting at bank 1 row 23 column 0
- st_wr2: Writes 0034 starting at bank 0 row 5 column 0
- st_rd1: Reads 2 16-bit words at bank 1 row 23 column 0
- st_rd2: Reads 2 16-bit words at bank 0 row 5 column 0

After the second read, the state machine signals that is done via the *test_done* flag and the simulation ends.

DDR_TOP.V

This file links the initialization, DDR controller, and read/write modules together and provides several necessary signals for the DDR RAM chip as well the three modules. It generates the clocks for the entire system using PLL's. The PLL's smooth the clock so that the rising and falling edges have minimal skew. The output frequency is the same as the input frequency. The user needs to change the PLL's configuration via the Altera PLL Megafunction to allow for a frequency other than 50 MHz to run the clock.

In order to handle reading and writing data on both clock edges the Altera altddio_bidir Megafunction was used (figure 11). 16-bit Write data is inputted to the *datain_h* and *datain_l* ports. At the positive edge, the word *datain_h* is outputted on *padio*, which is connected to the 16-bit wide bidirectional data bus on the DDR chip. At the negative edge *datain_l* is outputted to the *padio* bus. Note that figure 11 is an example from the Altera altddio_bidir user guide. It is

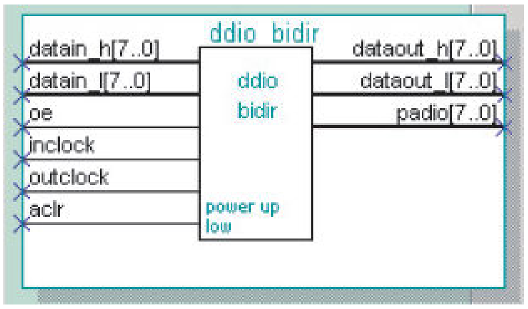


Figure 11. Altera Bidirectional IO block

not representative of the module used in DDR_TOP. The DDR_TOP module has 17-bit wide data busses and no asynchronous clear signal (*aclr*). The 17th bit is the DQS bit which is aligned with the write data. This is a simple workaround to having the DQS proerply aligned with the write data. The input ports are initialized to 0 at power-up. Read data comes into the *padio* bus from the DDR chip at the rising and falling edge of the clock. The output enable port *oe* allows data from the *padio* bus to go to *dataout_h* and *dataout_l* at the positive positive edge. The *oe* port is driven by the read enable signal *ren* coming from the DDR controller. It is interesting to note that in essence, the DDR RAM becomes a single edge triggered RAM with a databus of 32 bits. Figure 12 shows the processes of first receiving read then issuing a write.

In order to simulate the DDR_TOP module, a reset signal is designated as an input. The signal is removed when the DDR_TOP module is running on the FPGA. A button can be used as the reset switch in this case.

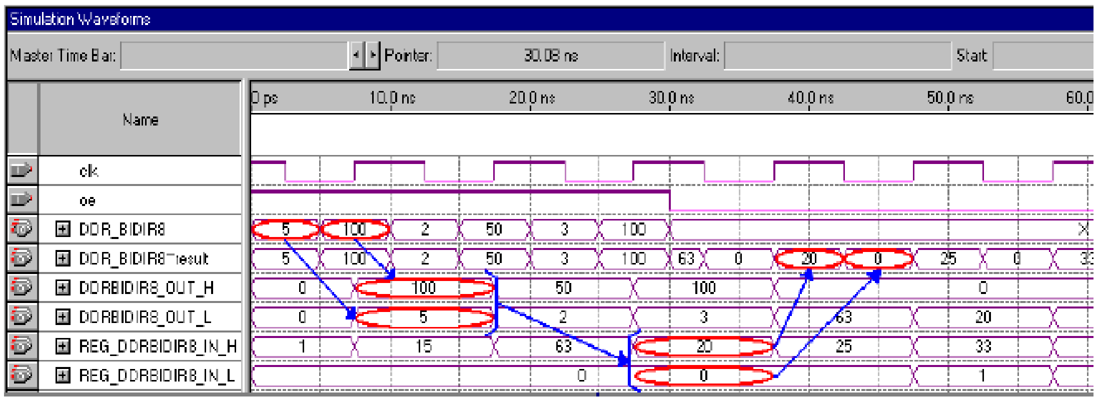


Figure 12. Sample Timing Diagram

Limitations of the DE2 Board

After the simulation of the DDR_TOP module was successful. Work shifted to incorporating the entire code base into the Altera Quartus II software. This software generates the netlist for the CycloneII FPGA and allows the user to interface his or her design with the DE2 board. Components on the DE2 board such as buttons, LEDs, and hex displays can be used. The code was successfully synthesized in QuartusII.

While running my DDR_TOP file it was observed that the output voltage on clock was around 4.2V peak-to-peak (figure 13). Realizing that the pins on the GPIO header were set to output at a too high voltage, I quickly unplugged the DDR daughter board from the DE2 headers. DDR RAM operates at the SSTL-2 IO standard. This means the voltage being inputted into the RAM from the expansion headers (from signals such as RAS,CAS,WE) must not go above the reference voltage of 1.3V with a tolerance of 2%.

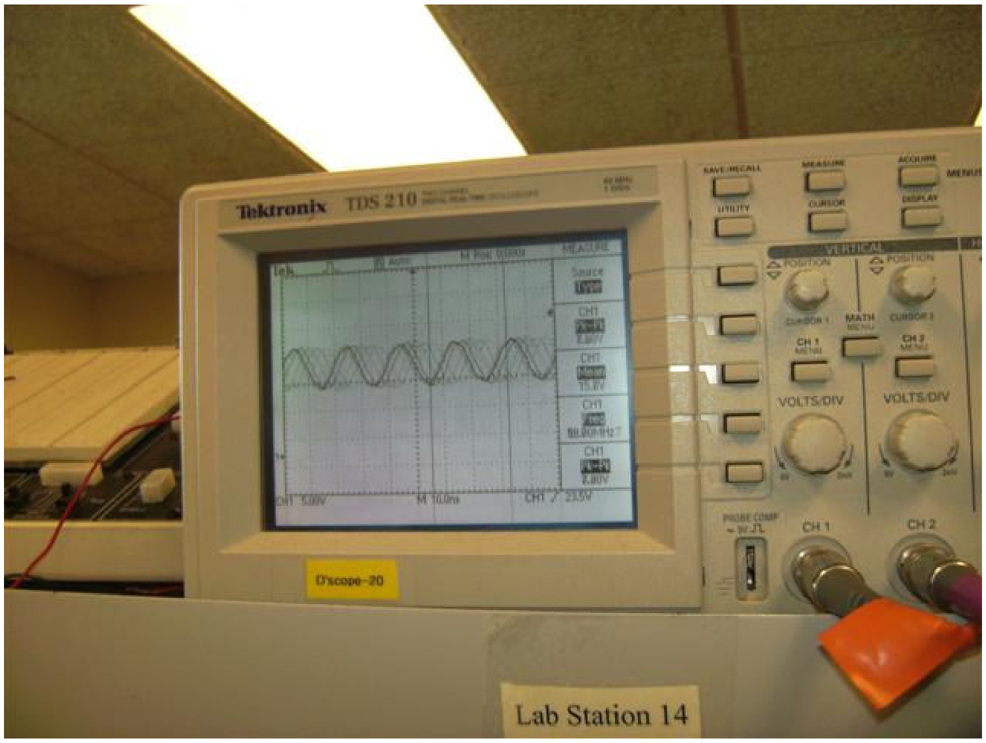
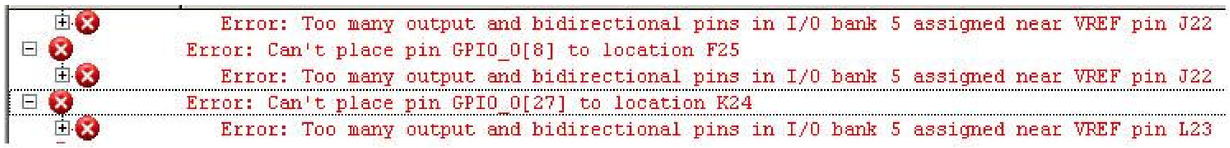


Figure 13. Clock Peak-to-Peak too high

The Quartus software however allows the user to change the IO standard on any of the IO banks (with some limitations) via the Pin Planner wizard. Banks 5 and 6, which the GPIO units were running off, supported SSTL-2. However, a reference voltage was needed on pins 2 and 33 of the GPIO_0. Also, while this is not listed in the documentation in the Cyclone II datasheet, each IO bank can only support 9 output pins being driven when a reference voltage is required, such as in the case when the SSTL-2 standard is needed. This was discovered when the following error messaged popped up during compilation (figure 14).



```
    Error: Too many output and bidirectional pins in I/O bank 5 assigned near VREF pin J22
Error: Can't place pin GPIO_0[8] to location F25
    Error: Too many output and bidirectional pins in I/O bank 5 assigned near VREF pin J22
Error: Can't place pin GPIO_0[27] to location K24
    Error: Too many output and bidirectional pins in I/O bank 5 assigned near VREF pin L23
```

Figure 14. An unfortunate error

This meant that the designers of the DE2 never intended for a DDR RAM interface to work with this board. While the Cyclone II does support the DDR

RAM IO standard, it would require the user to have access to 5 separate IO banks to accommodate for all the outputs, which is not the case with the DE2 board. Further evidence to back up this conclusion can be found in the DE2 schematic. While first GPIO header gives the user access to both voltage references to enable SSTL-2 output, the second GPIO header, which is driven by IO bank 6 only gives the user access to one of those pins. The other reference pin is used to drive a hex display output (figure 15). This is because in the event that a particular IO standard being used for bank 6 does not require a reference voltage, the Vref pin can be used as an output. The DE2 designers decided to take advantage of feature.

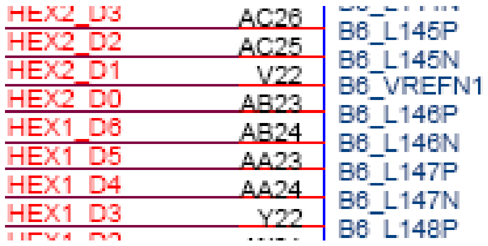


Figure 15. Hex1_D5 connected to second voltage reference pin (V22) of bank 6

Results and Conclusion

Digital Hardware Results

While the final intended goal of running the DDR chip via the Cyclone II FPGA was not realized, the design was proven correct via DDR_TOP module simulation. The *ddr.v* file is provided by Micron for designers to simulate their hardware with a Verilog version of the DDR RAM for debugging via simulation. It allows the user to configure the DDR module exactly as it would appear in hardware. For example, the user can set the DDR verilog model to have a 16-bit wide data bus at size of 256 MB. The module receives all the required input signals and gives feedback as to whether the DDR RAM controller is issuing commands properly. It also lets the user know whether the initialization steps have been properly completed. The feedback generated by the verilog model after running the simulation with the 2 writes and 2 reads described in the *DDR_read_write.v* can be seen in the appendix. Figure 16 shows waveform generated by the simulation.

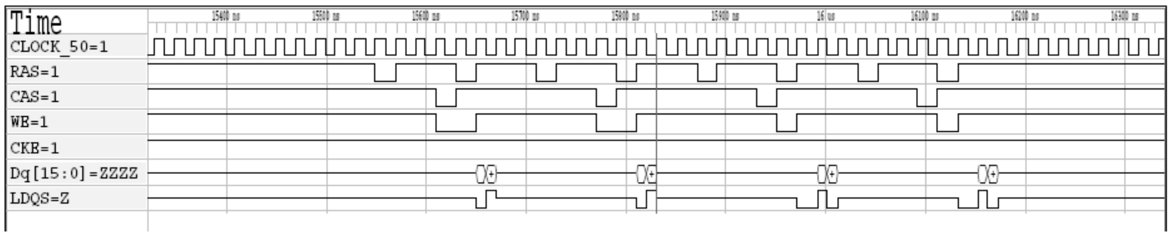


Figure 16. Result of the read/write simulation

First an activate command is sent by the controller to the chip (011), followed by two NOPs, followed by a write command (100), followed by a precharge command (010) and another write command to set the column for the first write. The DQ bus gets two 16-bit words after the precharge and the DQS signal goes low to high. A similar sequence of steps occurs for the second write. Reading requires an activate command, followed two nops, read command, a precharge command, and another read command.

Currently, the DDR controller only supports a CAS latency of 2. This is a minor limitation because a large majority of DDR chips support this CAS latency. Also, the controller does not support multiple active commands before a read or write. The user will not be able to access multiple banks in one read or write burst. This is part of the advanced functionality of the DDR standard and is used rarely.

Two important results were achieved however. I learned a great deal about a common real world standard and I was able to apply skills learned in the classroom and through my own research to successfully design a working DDR RAM controller. I also learned how to use real world tools to develop a system much like an engineer would on the job. These two takeaways will help me greatly as I transition to working as an engineer for a top engineering company.

Physical Hardware Result

The DDR chip board was a successful design. There were only 2 errors that were not caught before the board was ordered. These errors were fixed using a cutting tool to disconnect the erroneous traces. The spacing between surface mount pads was slightly off, but in the end the DDR chip was soldered on to the pads successfully. The board was also a little too big and a part of it had to be drilled off before it could be mated with the DE2 board. A picture of the board is located in the appendix.

Conclusion

This project required me to learn skills on my own and apply them within the span of two semesters. This was an invaluable experience that it taught me how to solve problems on my own and how to overcome difficulties that are at first seemingly insurmountable. While the final intended goal was not achieved, I believe this project is a great success. The controller works perfectly in simulation and only the lack of foresight by the DE2 board designers prevented me from porting to the FPGA.

There is of course room for improvement in my design. It does not support all the possible functionality provided by the DDR standard. But it does good job with the features it does support. My hope is that someone will use my code with the Cyclone II situated on the appropriate hardware to build a truly outstanding project utilizing the high-capacity and high-speed provided by DDR RAM. Overall, this has been a great experience. It will be something that I will draw on in my future work endeavors.

Acknowledgements

I would first like to thank my Dad, Nik, for giving me the inspiration to go ahead with such a difficult undertaking. I would also like to thank Sam Lee for assisting me with soldering the DDR RAM chip to the SMT pads even though they were slightly off (not an easy task). And lastly I would like to thank Bruce Land for taking me on as an advisee and dedicating a lot of his time first semester to teaching 576 and making sure we learned as much as possible about FPGAs in those 14 weeks.

References

1. “256Mb: x4, x8, x16 DDR SDRAM”. http://download.micron.com/pdf/datasheets/dram/ddr/256MBDDRx4x8x16.pdf

2. “Cyclone II Device Handbook”. http://www.altera.com/literature/lit-cyc2.jsp

3. “DE2 Datasheet”. http://users.ece.gatech.edu/~hamblen/DE2/DE2%20Reference%20Manual.pdf

4. “Double Data Rate (DDR) SDRAM Specification” http://download.micron.com/pdf/misc/JEDEC79R2.pdf

5. “Quartus II Development Handbook” http://www.altera.com/literature/lit-qts.jsp

6. “State Machine Coding Styles”. http://www.sunburst-design.com/papers/CummingsSNUG1998SJ_FSM.pdf

Appendix

Table of the controller state encodings

State	rf	spare	ren	wen	dts	amx	done	rcw
st_idle	0	00	0	0	1	00	1	111
st_extcmd	0	00	0	0	1	11	0	111
st_extdn	0	01	0	0	1	00	1	111
s_w_act	0	00	0	0	0	00	0	011
st_w_nop1	0	00	0	0	0	01	0	111
st_w_nop2	0	01	0	0	0	01	0	111
st_write	0	00	1	0	0	01	0	100
st_w_brst	0	00	1	0	0	01	0	111
st_r_act	0	01	0	0	1	00	0	011
st_r_nop1	0	01	0	0	1	01	0	111
st_r_nop2	0	00	0	0	1	01	0	111
st_read	0	01	0	1	1	01	0	101
st_r_brst	0	01	0	1	1	00	0	111
st_refr	1	00	0	0	1	00	0	001
st_refrwtdn	1	00	0	0	1	00	0	111
st_term	0	00	0	0	1	00	0	010
st_nop2	0	11	0	0	1	01	0	111

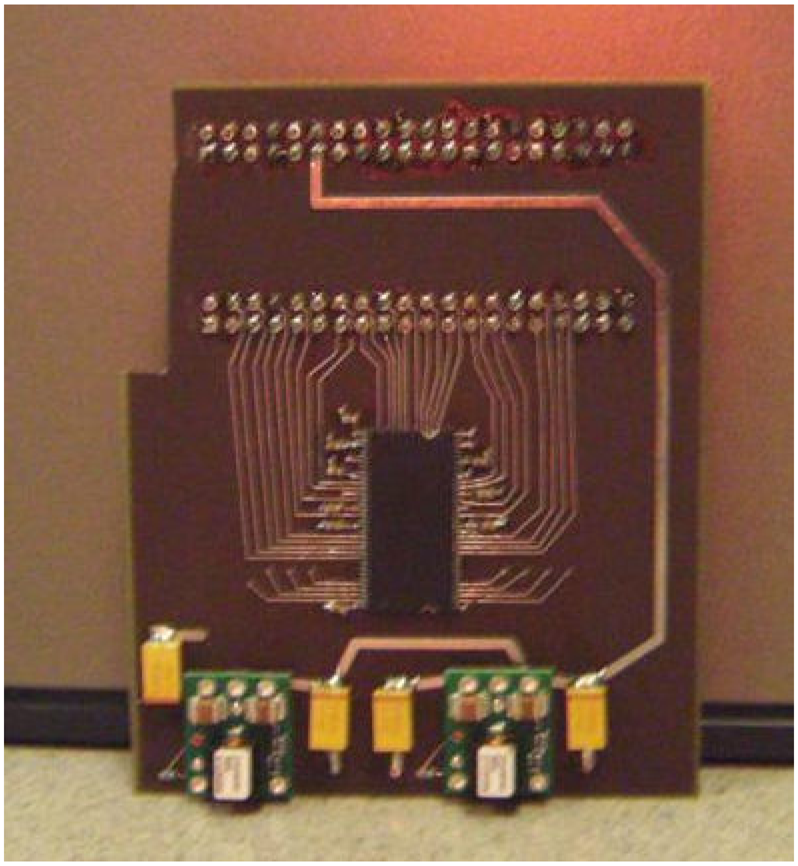
The spare bits are used to differentiate between states which output the same thing out of the DDR controller. For example, both the write activate state (st_w_act) and read activate state (st_r_act) issue the same activate command, but they are distinguished by the spare bits.

DDR Controller State Transition Logic

```
 always @(state or clmdn or cmd[2:0] or rfdn)
      case (state)

        st_idle:
          case (cmd[2:0]) //command to controller, 1xx is always extended
            3'b000:  next = st_idle;  // no cmd
            3'b001:  next = st_refr;  // refresh
            3'b010:  next = st_w_act; // write
            3'b011:  next = st_r_act; // read
            3'b100:  next = st_extcmd;
            3'b101:  next = st_extcmd;
            3'b110:  next = st_extcmd;
            3'b111:  next = st_extcmd;
          endcase // case(cmd[1:0])

        st_refr:     next = st_rfwtdn;
        st_rfwtdn:
          case (rfdn) //rfdn is a delayed signal, takes a few cycles to be asserted
            1'b0:    next = st_rfwtdn;
            1'b1:    next = st_idle;
          endcase // case(rfdn)

        st_extcmd:   next = st_extdn;

        st_extdn:
          case (cmd[2])
            1'b1:    next = st_extdn;
            1'b0:    next = st_idle;
          endcase

        st_w_act:    next = st_w_nop1;
        st_w_nop1:   next = st_w_nop2;
        st_w_nop2:   next = st_write;
        st_write:
          case(clmdn)//counter for write burst, 0 means w_brst is still going
            1'b0:    next = st_w_brst;
            1'b1:    next = st_term;

          endcase // case(clmdn)
        st_w_brst:
          case(clmdn) //counter for write burst, 0 means it’s still going
            1'b0:    next = st_w_brst;
            1'b1:    next = st_term;

          endcase // case(clmdn)

        st_r_act:    next = st_r_nop1;
        st_r_nop1:   next = st_r_nop2;
        st_r_nop2:   next = st_read;
        st_read:
          case(clmdn) //counter for read burst, 0m neas it’s still going
            1'b0:    next = st_r_brst;
            1'b1:    next = st_term;

          endcase // case(clmdn)
        st_r_brst:
          case(clmdn) //counter for read burst, 0m neas it’s still going
            1'b0:    next = st_r_brst;
            1'b1:    next = st_term;

          endcase // case(clmdn)
```

```
  default:     next = st_idle; //default state idle
endcase // case(state)
```

DDR RAM chip board

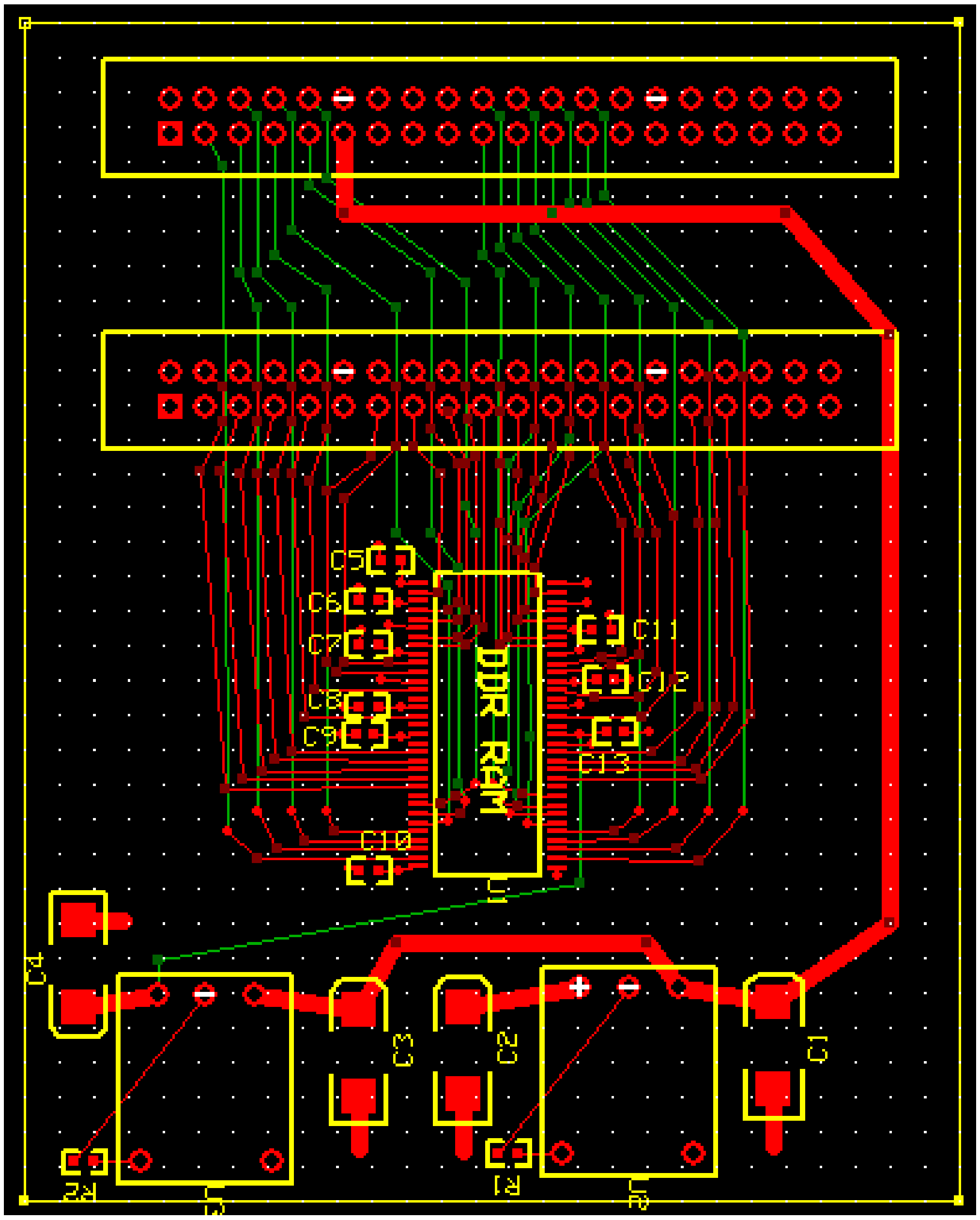


Note the missing rectangular section on the top left-hand side. This was required to make sure the board could fit on to the DE2 board.

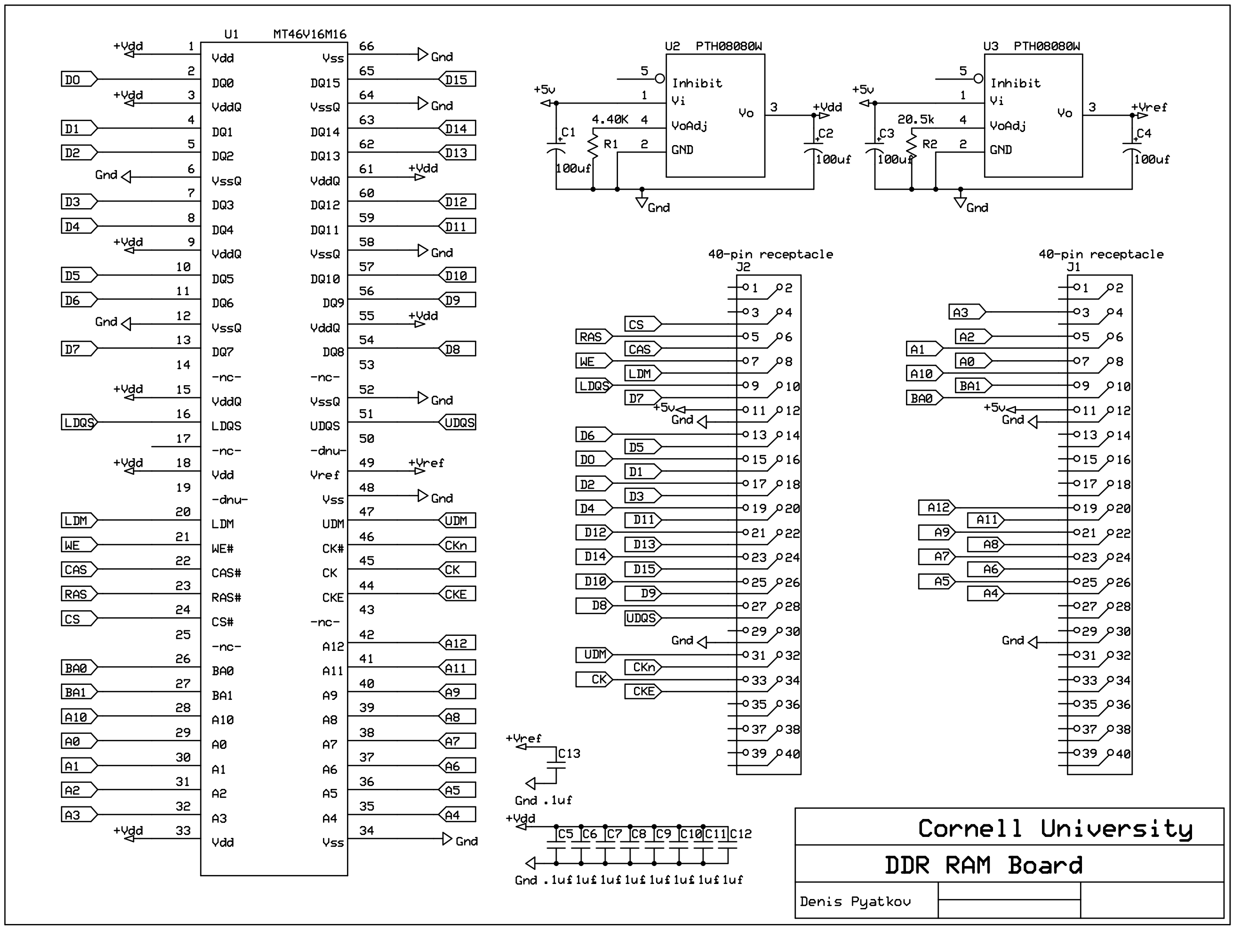
Results from running the simulation of 2 writes followed by reads as well initialization

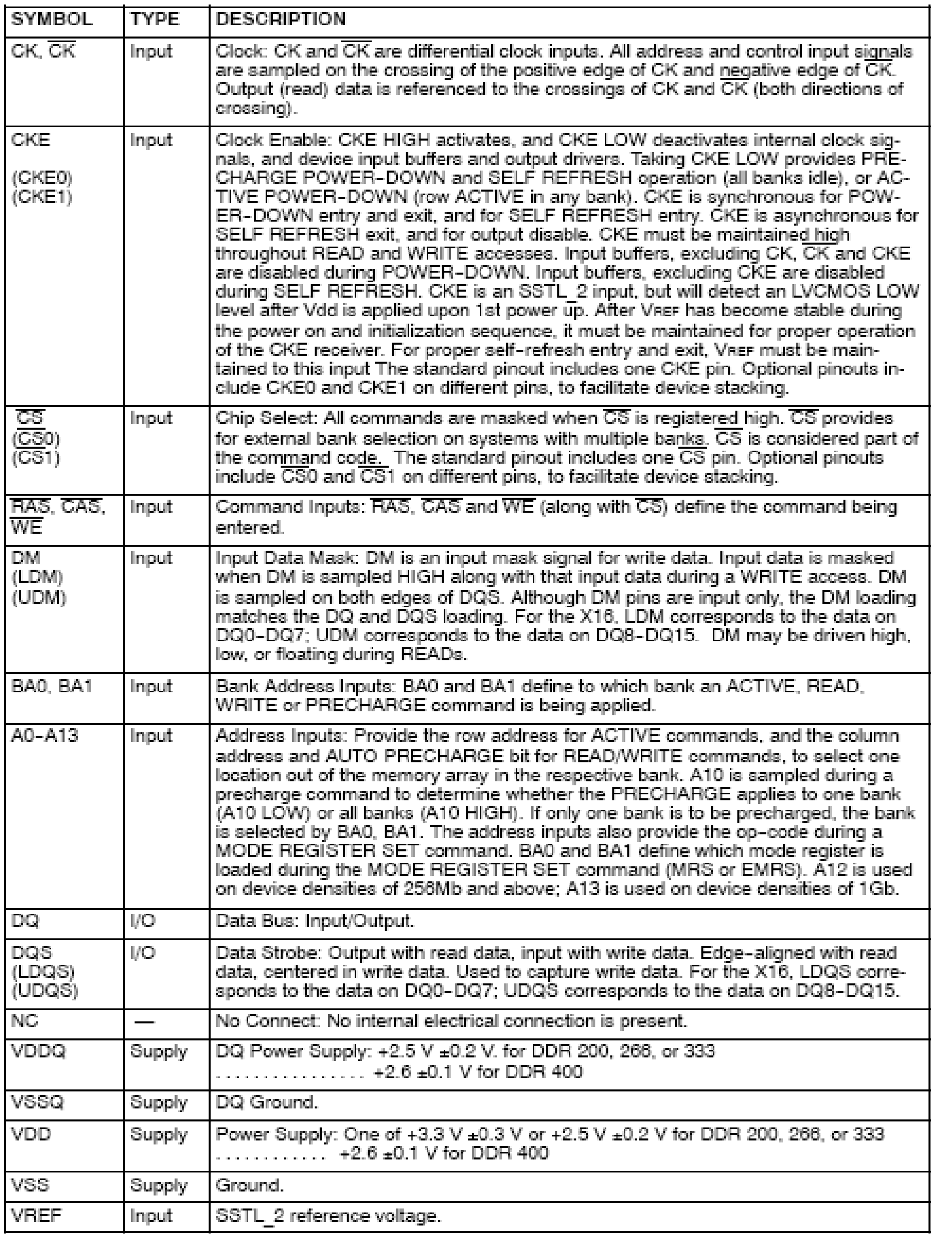
```
At time 10190.000 ns PRE  : Addr[10] = 1, Bank = 00
At time 10410.000 ns EMR  : Extended Mode Register
At time 10410.000 ns EMR  : Enable DLL
At time 10630.000 ns LMR  : Load Mode Register
At time 10630.000 ns LMR  : Burst Length = 2
At time 10630.000 ns LMR  : CAS Latency = 2
At time 10850.000 ns PRE  : Addr[10] = 1, Bank = 00
At time 11070.000 ns AREF : Auto Refresh
At time 11290.000 ns AREF : Auto Refresh
test.ddram: at time 11290.000 ns MEMORY:  Power Up and Initialization
Sequence is complete
At time 11510.000 ns AREF : Auto Refresh
At time 15570.000 ns ACT  : Bank = 1, Row = 0017
At time 15630.000 ns WRITE: Bank = 1, Col = 000
At time 15650.000 ns PRE  : Addr[10] = 0, Bank = 01
At time 15660.000 ns WRITE: Bank = 1, Row = 0017, Col = 000, Data = 34
At time 15670.000 ns WRITE: Bank = 1, Row = 0017, Col = 001, Data = 00
At time 15730.000 ns ACT  : Bank = 0, Row = 0005
At time 15790.000 ns WRITE: Bank = 0, Col = 000
At time 15810.000 ns PRE  : Addr[10] = 0, Bank = 00
At time 15820.000 ns WRITE: Bank = 0, Row = 0005, Col = 000, Data = 34
At time 15830.000 ns WRITE: Bank = 0, Row = 0005, Col = 001, Data = 00
At time 15890.000 ns ACT  : Bank = 1, Row = 0017
At time 15950.000 ns READ : Bank = 1, Col = 000
At time 15970.000 ns PRE  : Addr[10] = 0, Bank = 01
At time 15990.000 ns READ : Bank = 1, Row = 0017, Col = 000, Data = 34
At time 16000.000 ns READ : Bank = 1, Row = 0017, Col = 001, Data = 00
At time 16050.000 ns ACT  : Bank = 0, Row = 0005
At time 16110.000 ns READ : Bank = 0, Col = 000
At time 16130.000 ns PRE  : Addr[10] = 0, Bank = 00
At time 16150.000 ns READ : Bank = 0, Row = 0005, Col = 000, Data = 34
At time 16160.000 ns READ : Bank = 0, Row = 0005, Col = 001, Data = 00
```

Final PCB Layout



Final PCB Schemtic





SYMBOL	TYPE	DESCRIPTION
CK, $\overline{CK}$	Input	Clock: CK and $\overline{CK}$ are differential clock inputs. All address and control input signals are sampled on the crossing of the positive edge of CK and negative edge of $\overline{CK}$. Output (read) data is referenced to the crossings of CK and $\overline{CK}$ (both directions of crossing).
CKE (CKE0) (CKE1)	Input	Clock Enable: CKE HIGH activates, and CKE LOW deactivates internal clock signals, and device input buffers and output drivers. Taking CKE LOW provides PRECHARGE POWER-DOWN and SELF REFRESH operation (all banks idle), or ACTIVE POWER-DOWN (row ACTIVE in any bank). CKE is synchronous for POWER-DOWN entry and exit, and for SELF REFRESH entry. CKE is asynchronous for SELF REFRESH exit, and for output disable. CKE must be maintained high throughout READ and WRITE accesses. Input buffers, excluding CK, $\overline{CK}$ and CKE are disabled during POWER-DOWN. Input buffers, excluding CKE are disabled during SELF REFRESH. CKE is an SSTL_2 input, but will detect an LVCMOS LOW level after Vdd is applied upon 1st power up. After V_{REF} has become stable during the power on and initialization sequence, it must be maintained for proper operation of the CKE receiver. For proper self-refresh entry and exit, V_{REF} must be maintained to this input The standard pinout includes one CKE pin. Optional pinouts include CKE0 and CKE1 on different pins, to facilitate device stacking.
$\overline{CS}$ ($\overline{CS}$0) ($\overline{CS}$1)	Input	Chip Select: All commands are masked when $\overline{CS}$ is registered high. $\overline{CS}$ provides for external bank selection on systems with multiple banks. $\overline{CS}$ is considered part of the command code. The standard pinout includes one $\overline{CS}$ pin. Optional pinouts include $\overline{CS}$0 and $\overline{CS}$1 on different pins, to facilitate device stacking.
$\overline{RAS}$, $\overline{CAS}$, $\overline{WE}$	Input	Command Inputs: $\overline{RAS}$, $\overline{CAS}$ and $\overline{WE}$ (along with $\overline{CS}$) define the command being entered.
DM (LDM) (UDM)	Input	Input Data Mask: DM is an input mask signal for write data. Input data is masked when DM is sampled HIGH along with that input data during a WRITE access. DM is sampled on both edges of DQS. Although DM pins are input only, the DM loading matches the DQ and DQS loading. For the X16, LDM corresponds to the data on DQ0-DQ7; UDM corresponds to the data on DQ8-DQ15. DM may be driven high, low, or floating during READs.
BA0, BA1	Input	Bank Address Inputs: BA0 and BA1 define to which bank an ACTIVE, READ, WRITE or PRECHARGE command is being applied.
A0-A13	Input	Address Inputs: Provide the row address for ACTIVE commands, and the column address and AUTO PRECHARGE bit for READ/WRITE commands, to select one location out of the memory array in the respective bank. A10 is sampled during a precharge command to determine whether the PRECHARGE applies to one bank (A10 LOW) or all banks (A10 HIGH). If only one bank is to be precharged, the bank is selected by BA0, BA1. The address inputs also provide the op-code during a MODE REGISTER SET command. BA0 and BA1 define which mode register is loaded during the MODE REGISTER SET command (MRS or EMRS). A12 is used on device densities of 256Mb and above; A13 is used on device densities of 1Gb.
DQ	I/O	Data Bus: Input/Output.
DQS (LDQS) (UDQS)	I/O	Data Strobe: Output with read data, input with write data. Edge-aligned with read data, centered in write data. Used to capture write data. For the X16, LDQS corresponds to the data on DQ0-DQ7; UDQS corresponds to the data on DQ8-DQ15.
NC	—	No Connect: No internal electrical connection is present.
VDDQ	Supply	DQ Power Supply: +2.5 V ±0.2 V. for DDR 200, 266, or 333 . . . . +2.6 ±0.1 V for DDR 400
VSSQ	Supply	DQ Ground.
VDD	Supply	Power Supply: One of +3.3 V ±0.3 V or +2.5 V ±0.2 V for DDR 200, 266, or 333 . . . . +2.6 ±0.1 V for DDR 400
VSS	Supply	Ground.
VREF	Input	SSTL_2 reference voltage.

Figure A1. Control and Data pins of DDR1 RAM module