ECEN3763 – SDRAM Usage Lab

Week 8

Due Friday, March 11th, 11:59pm – 35 points

Summary:

In order to create a full screen resolution frame buffer, the SDRAM memory on the DE10-Standard board will be used. Before you begin work on the frame buffer system, this lab will allow you to verify correct operation of the SDRAM memory

In Canvas, under Files > Reference Material > System Console, the ug_system_console.pdf file contains a lot of useful information about System Console, especially regarding the Tcl commands you will need to test the SDRAM.

Submission and Grading:

Create a Quartus archive file (.qar) that contains your entire design, including Tcl script. When you run Project > Archive project in Quartus, the Archive Project window opens. If you press the Advanced button, you can see the complete list of files included in the archive. If you don't see your Tcl script in the file list, add it to the archive. Please do me a favor – add the .cdf file created by the Quartus programmer. This will save me a lot of mouse clicks as I grade this lab.

You may choose to include a brief text file describing issues with your final design (if any), or anything else you want me to know about your submission.

This lab will be graded as follows:

Quartus project compiles and works correctly	20 points
Tcl script demonstrating correct SDRAM operation	15 points

Lab Instructions:

Part 1: Create Hardware Design

a. Start by creating a new Quartus project using Terasic System Builder. Include CLOCK, LED, Button, and SDRAM components.

b. Open the new project in Quartus. Open Platform Designer.

c. Use Platform Designer and add the IP to your design. The IP configuration details are given in Part 2 below.

Part 2: Create Design Component in Platform Designer

a. Add one PLL
    a. 2 output clocks, both 143 Mhz, make one clock have a -3 ns phase shift. The phase shifted clock will be exported from Platform Designer and connected to the DRAM_CLK port of your design.

b. Add the SDRAM Controller Intel FPGA IP
    a. Data width: 16 bits
    b. Chip select: 1
    c. Banks: 4
    d. Row address width: 13
    e. Column address width: 10

    Timing tab
    f. CAS latency cycles: 2
    g. Initialization refresh cycles: 8
    h. Issue one refresh….: 7.8 us
    i. Delay after powerup: 100 us
    j. Duration of refresh command: 55 ns
    k. Duration of precharge command: 15 ns
    l. ACTIVE to READ…: 15 ns
    m. Access time: 5.5 ns
    n. Write recovery time: 14 ns

c. Add one PIO
    a. Width 10 bits, output only

d. Add JTAG to Avalon Master Bridge

e. Make connections using the patch panel. The errors shown at the bottom of the screen will help with this. Personally I find the patch panel confusing, it has taken awhile to get comfortable with it.

    a. Clock block (at top of diagram)
        i. clk_in, no connections
        ii. clk_in_reset, no connections

iii. clk, connect to PLL refclk
iv. clk_reset, connect to PLL reset, SDRAM controller reset, PIO reset, and JTAG master clk_reset

b. PLL
   i. Connect one of the outclks with zero phase shift to the SDRAM controller, PIO clk, and JTAG clk
   ii. Export the other outclk and locked signals. You may want to rename the outclk that has the phase shift to make things less confusing.

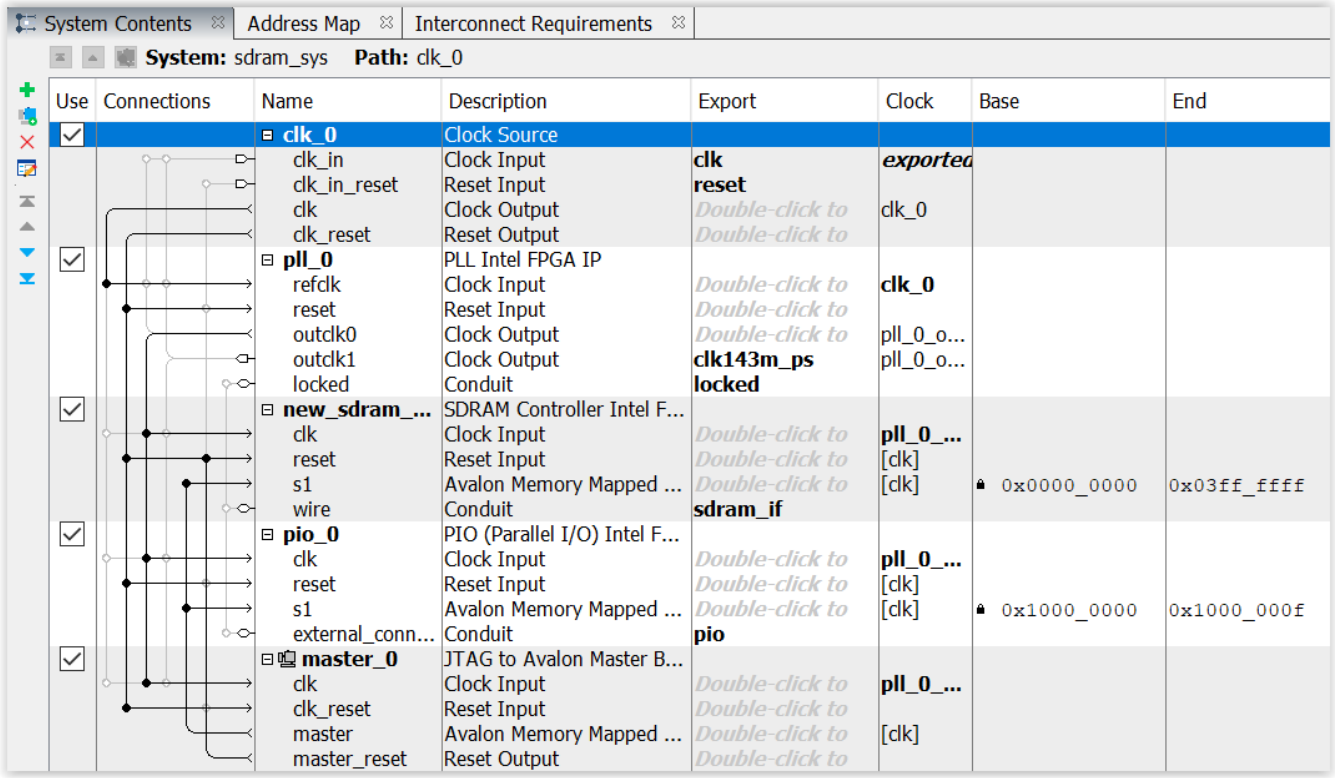
c. JTAG Master Bridge
   i. Connect master output to PIO and SDRAM s1
   ii. Connect master_reset to SDRAM reset

d. SDRAM Controller
   i. Export wire

e. PIO
   i. Export the external_connection

f. Once the JTAG Avalon Master Bridge master connection was made, an error was shown related to the memory map. The error indicates that both the SDRAM controller, PIO block, and PLL Reconfig have the same base address.
   a. Click on the PIO Base address, and change it to 0x1000_0000.
   b. Clock on the tiny lock icons (just to the left of the base addresses) to lock the addressing.
   c. Note the upper address of the SDRAM (0x03ff_ffff).

At this point the only warning shown in the Messages tab is related to the PLL (and is not a problem), and 2 Info Messages.



Part 3: Generate Platform Designer block and add to project

a. Click the Generate HDL button at the bottom right. In the dialog box that opens, make sure that Verilog is selected. You will not use a block symbol file, so can uncheck the box (or leave it, doesn't matter). We will not be simulating so no simulation model is required. And, leave the Output Directory alone, is should point into your Quartus project.

b. Generate the design. When prompted by the Save window, rename the component to something meaningful. When completed, exit Platform Designer.

c. In Quartus, add the Platform Designer source (<name>.qip). The exact location of this file depends on a few things, but likely it is in your <main project directory>/<ip name>/ synthesis directory.

d. With the .qip file added to your project (in the <component_name>\synthesis directory), go to Files view, expand the IP, and look at the .v file directly under the .qip file. Without modifying the file, copy the module instantiation (first 20 or 25 lines) and use this to help you instantiate the Platform Designer IP block into your design.

e. Connect the DRAM signals. The top level port names match up with the PD generated names, except for the dqm (component) and DRAM_LDRM and DRAM_UDQM (top level ports) signals. Concatenate the upper and lower data mask signals together as shown below. Note that your component name and port names will be different based on your PD naming used.

```
        .new_sdram_controller_0_wire_dqm ({DRAM_UDQM, DRAM_LDQM}),
```

f. The PIO connection hooks to the LEDRs.
g. The PLL outclk signal that includes the phase shift connects to the DRAM_CLK port.
h. Connect the reset_reset_n to KEY[0].
i. The locked output signal can be left unconnected.
j. Compile your design. It is interesting to note how many warnings are generated in Quartus for Intel IP. I like to look at this so I don't feel too badly about my designs.
k. Before testing your design, examine the worst case setup and hold slack for this design. In the Table of Contents pane in Quartus, expand Timing Analyzer and click on Multicorner Timing Analysis Summary. Note the worst case setup and hold slack values. Next, expand the Slow 1100mv 85C Model heading, and look at the Fmax summary. The reported clock frequency shows how much frequency margin the design has at the slowest process corner.

Part 4: Does your design work????

a. Program your board. After programming, close the Quartus Programmer. The programmer and System Console both try to capture the USB cable, and will conflict.
b. Open System Console (Tools > System Debugging Tools > System Console).
c. Refer to Appendix 1, and interact with your design using the Tcl console found in System Console. Confirm that the SDRAM on your DE10-Standard board can be written and read reliably.

Part 5: Create Tcl script

Create a Tcl script that writes one value to the first 1000 memory locations. When writing is complete, read back the first 1000 memory locations and confirm the data is correct. If no errors occur, turn on LEDR[9]. If errors are detected, turn on LEDR[0] (and fix your design so that it works).

It is OK to write the same value to all locations, this will simplify the readback verification.

Appendix 1: Tcl scripting in System Console

The code below demonstrates how to get started with System Console. You can find details on all System Console Tcl commands in the the document ug_system_console.pdf. On page 27 are the Avalon-MM commands, which are used when reading and writing from the SDRAM.

Writing a script for System Console can be confusing, as it uses a less than intuitive approach to communicating with your system. You can use the code below (be careful about cut and paste from a PDF file). You will need to write your own code for the SDRAM testing.

If you need some Tcl documentation that explains Tcl if/else and looping syntax, look at https://tcl.tk/man/tcl8.5/tutorial/Tcl7.html and https://tcl.tk/man/tcl8.5/tutorial/Tcl10.html .

System Console Tcl code

You identify the master device using the get_service_paths command, and assign the value to master_path. Next, you claim the path using the claim_service command in line 2. Line 3 performs a similar action except for the JTAG debug commands

The code next issues a reset from the JTAG Master Bridge, then verifies the system has a functional clock, and tests the reset state.

a. Initialization and test code

```
set master_path [lindex [get_service_paths master] 0]
set mpath [claim_service master $master_path ""]
# Locate JTAG path
set jpath [lindex [get_service_paths jtag_debug] 0]

# Reset system
jtag_debug_reset_system $jpath
puts "System reset......"

# Test clock is running and reset is high
set clk [jtag_debug_sense_clock $jpath]
if {$clk == 1} {
    puts "Clocking running properly."
} {else
    puts "Clocking appears to not be running."
}
set rst [jtag_debug_sample_reset $jpath]
puts "Reset level is $rst (should be 1)."
```

b. To write to the PIO block

```
master_write_16 $mpath 0x10000000 0x5555
```

will write the data value to the LEDs on the board.

c. Testing SDRAM

To test the SDRAM, you can use either master_write_memory $mpath <addr> <data> to do 8 bit writes, or you can use master_write_8, master_write_16, or master_write_32 commands to do 8, 16, or 32 bit writes. You can the same commands for reading, just replace write with read in the command names.

```
master_write_16 $mpath 0x0 0x5555 0x1234 0x9864
```

will write three 16 bit words to memory starting at address 0.

```
master_read_16 $mpath 0x0 10
```

will read back 10 16 bits words starting at address 0.

d. Close script

End with

```
close_service master $mpath
```