# Lab #3
# Adding SDRAM to your System

## EELE 475
## HARDWARE AND SOFTWARE ENGINEERING
## FOR EMBEDDED SYSTEMS

### Assignment Date: 9/9/14

### Verification Sheet Due Date: 9/16/14

**Lab Description**

The DE2 board has 8MB of SDRAM on it and today's lab is to create a Nios system that will use it. Like last week, you will need to use the following Altera tools:

- **Quartus II** (v13.0) - To compile the VHDL that creates your processor hardware.
- **Programmer** - To download your bit stream to the DE2 board. (QuartusII -> Tools -> Programmer)
- **Qsys** - To create your processor system. (QuartusII -> Tools -> Qsys)
- **NIOS II IDE** - To write, compile, and download your C program. (Qsys -> Tools -> Nios II IDE)

Your starting point will be the system you created last time. Create a new lab3 directory and copy all your files from lab2 into this directory. We will remove the on-chip memory from last lab and replace this with a SDRAM controller to access the 8 MB of DRAM on the DE2 board. We will also create new clock signals that are needed for the SDRAM chip and peripherals.

**Part 1 – Qsys Components.**

Modify your Qsys system by performing the following operations:

1. Remove the on-chip memory.

2. Add a **clock** (under the Clock Settings tab) if one doesn't exist, but there should be one already there. Name this clock **clk** and set it to be 50 MHz. Export the clk_in signal and name it clk. Export the clk_in_reset signal and name it reset.

3. The Nios II processor is the Nios II/s (middle range processor) and should be named **cpu_nios**.

4. Add the SDRAM controller and name it **sdram_controller**. It can be found at: Memories and Memory Controllers -> External Memory Interfaces -> SDRAM Interfaces -> SDRAM Controller. The parameters for the SDRAM controller are found below (we will be creating a custom SDRAM controller). Export the wire signal and name it sdram.

   **SDRAM Controller** (for targeting the onboard DE2 8-Mbyte SDRAM chip)

   **Memory Profile:**

   | | | | | |
   |---|---|---|---|---|
   | **Data Width** | Bits: | 16 | | |
   | **Architecture:** | Chip select: | 1 | Banks: | 4 |
   | **Address Widths:** | Row | 12 | Column | 8 |
   | **Generic Memory model…** | don't select | | | |

   **Timing:**

   | | | |
   |---|---|---|
   | **CAS latency cycles:** | 3 | |
   | **Initialization refresh cycles:** | 2 | |
   | **Issue one refresh command every:** | 15.625 | us |
   | **Delay after powerup, before initialization:** | 100 | us |
   | **Duration of refresh command (t_rfc):** | 70 | ns |
   | **Duration of precharge command (t_rp):** | 20 | ns |
   | **ACTIVE to READ or WRITE delay (t_rcd):** | 20 | ns |
   | **Access time (t_ac):** | 5.5 | ns |
   | **Write recovery time (t_wr, no auto precharge):** | 14 | ns |

5. The LEDs and Switches PIO. The PIO components can found at: Peripherals->Microcontroller Peripherals->PIO (Parallel I/O). Make the PIOs 18 bits instead of 8 bits (so that all the red LEDs and switches can be exercised). Rename the input PIO to ***Switches*** and the output PIO to ***LEDs*** (by right clicking on the name and selecting rename). Export the external_connection signals and name them **switches** and **leds** respectively.

6. Add the JTAG UART, which can be found at: Interface Protocols->Serial->JTAG UART. Set the interrupt level to be 16 (mid-level priority)

**Qsys Connections**

7. In the **clk** component,

    a. click on the **clk** signal to make it bold and connect it to the following:

        i. component: cpu_nios signal:**clk**
        ii. component: sdram_controller signal:**clk**
        iii. component: Switches signal:**clk**
        iv. component: LEDs signal:**clk**
        v. component: jtag_uart signal:**clk**

    b. click on the **clk_reset** signal to make it bold and connect it to the following:

        i. component: cpu_nios signal:**reset_n**
        ii. component: sdram_controller signal:**reset**
        iii. component: Switches signal:**reset**
        iv. component: LEDs signal:**reset**
        v. component: jtag_uart signal:**reset**

8. In the **cpu_nios** component,

    a. click on the **data_master** signal to make it bold and connect it to the following:

        i. component: cpu_nios signal:**jtag_debug_module**
        ii. component:sdram_controller signal:**s1**
        iii. component: Switches signal:**s1**
        iv. component: LEDs signal:**s1**
        v. component: jtag_uart signal:**Avalon_jtag_slave**

    b. click on the **instruction_master** signal to make it bold and connect it to the following:

        i. component: cpu_nios signal:**jtag_debug_module**
        ii. component:sdram_controller signal:**s1**

9. Change the Nios2 **Reset Vector** and **Exception Vector** to point to the SDRAM controller by double clicking the cpu_nios component and changing the corresponding sections.

10. Reassign the base addresses (System->Assign base addresses) and make sure that there are no error messages.

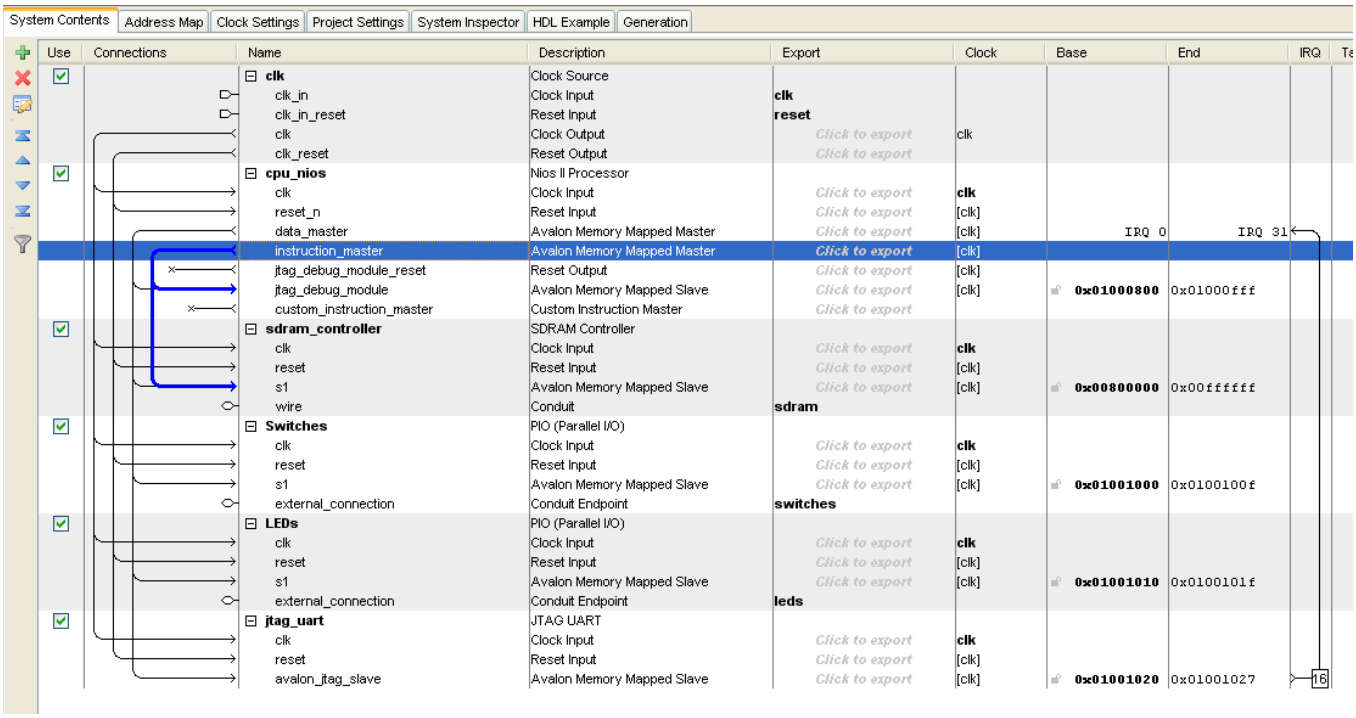
**11.** Under the Project Settings Tab, the parameters should be the following

a. Device Family: Cyclone II
b. EP2C35F672C6
c. Clock crossing adapter type: Handshake
d. Limit interconnect pipeline stages to: 0
e. Generation ID: 0

**12.** Regenerate your system.

**13.** Errors: If the BSP generation (in Eclipse) fails after you modified your system, it is likely that the system has been messed up. You may have to rebuild your Qsys system from the beginning. If it is complaining about the .rodata etc., remove the memory and add it back in.

Here's what your system should look like:



**Part 2 – VHDL –Quartus II top level modification**

***14.*** Create the new clocks:

***a.*** In Quartus, open the MegaWizard: Tools -> MegaWizard Plug-in Manager

***b.*** Create a new custom megafunction variation

***c.*** Name the new block *clockPLL*, and have it placed in your DE2_system directory.

***d.*** Select the megafunction I/O->ALTPLL

***e.*** Select the frequency of the inclk0 to be 50 MHz. This will be the 50 MHz DE2 board clock.

***f.*** Click next and deselect the asynchronous reset input **'areset'** and the **'lock'** output signal 'locked'.

***g.*** Click next until you get to the C0 output screen.

***h.*** Set the output clock frequency of C0 to be 50 MHz.

***i.*** Keep the Clock phase shift at 0 ns. This will be the **Nios clock** (the clock that goes to the Nios CPU component). We need this clock to be at a known phase shift relative to the c1 clock that we will be creating next.

***j.*** Click Next

***k.*** Click 'use this clock' and set the output clock frequency of C1 to be 50 MHz.

***l.*** Set the Clock phase shift to be -3.0 ns. This will be the **DRAM clock** (the clock that will go out of the FPGA to the physical SDRAM chip). We need to adjust the phase due to the propagation delay to the SDRAM chip. We need the clock signal to arrive at the same time the data from the Nios CPU arrives.

***m.*** Click next until you are at the last page (summary). Make sure both the **component declaration file** and the **instantiation template file** are checked. This will make it much easier to add the component and instantiate it in your VHDL code.

***n.*** Click Finish.

***o.*** **Note:** The 50 MHz clock (with phase shift zero) goes to the NIOS processor. The 50 MHz clock with the -3 ns phase shift goes out of the FPGA to the SDRAM chip that is on the DE2 board.

15. **Declare** clockPLL as a component in your design (place it before the *begin* statement in your VHDL code). You can add the component declaration by Edit->Insert Template-> MegaFunctions->VHDL Components->clockPLL.cmp and inserting into your top level VHDL file. (You can delete the Altera copyright comments).

16. **Instantiate** the clockPLL component (after the *begin* statement). You can do this by Edit->Insert Template->MegaFunctions->Instances->clockPLL_inst.vhd and inserting into your top level VHDL file.

17. In the port map, connect up the following signals:

```
clockPLL_inst : clockPLL PORT MAP (
        inclk0   => CLOCK_50,   -- 50 MHz board clock
        c0       => clk_nios,   -- 50 MHz no phase shift (relative to c1)
        c1       => DRAM_CLK    -- 50 MHz -3.0ns phase shift );
```

18. You will need to **declare** the following signals (before the begin statement):

```
signal dram_ba   : std_logic_vector(1 downto 0);
signal dram_dqm  : std_logic_vector(1 downto 0);
signal clk_nios : std_logic;
```

19. You will need to import the nios_Qsys component again like you did in the previous lab and both declare and instantiate it. Notice that it has nine additional signals for the sdram controller. You can cut and paste the code from the HDL Example tab in Qsys (select VHDL).

20. Your instantiation (and a couple signal hookups) should look something like the following (depending on how you named your export signals):

```
u0 : component Nios_Qsys
    port map (
      clk_clk                 => clk_nios,
      reset_reset_n           => KEY(0),
      switches_export         => SW,
      leds_export             => LEDR,
      sdram_dqm               => dram_dqm,
      sdram_we_n              => DRAM_WE_N,
      sdram_cke               => DRAM_CKE,
      sdram_cas_n             => DRAM_CAS_N,
      sdram_ras_n             => DRAM_RAS_N,
      sdram_dq                => DRAM_DQ,
      sdram_cs_n              => DRAM_CS_N,
      sdram_ba                => dram_ba,
      sdram_addr              => DRAM_ADDR   );
```

**21.** Make the following signal connections in your design:

```
DRAM_BA_0  <= dram_ba(0);
DRAM_BA_1  <= dram_ba(1);
DRAM_LDQM <= dram_dqm(0);
DRAM_UDQM <= dram_dqm(1);
```

**22.** Delete or comment out the signal assignments that are driving the DRAM control signals to zero.

**23.** Open up your TimeQuest timing .sdc timing constraint file and add the following statement to constrain the PLL generated clock:

a. derive_pll_clocks

**Note:** a copy of a working .sdc file can be found on the D2L site that you can download and import into your design (which is a different .sdc file from lab2).

**24.** Compile the design in Quartus and download the bit stream to the DE2 board.

**Part 3 – Software**

**25.** As before, Create a directory called *Eclipse_Workspace* and a directory called *Software* in your \DE2_System directory.

**26.** Open up the NOIS II IDE (you can do this from within the Qsys -> Tools -> *Nios II Software Build Tools for Eclipse*)

**27.** Set the workspace to the directory Eclipse_Workspace that you created.

**28.** Select File->New->*Nios II Application and BSP from Template*.

**a.** For the SOPC Information File name, browse to the file Nios_Qsys.sopcinfo found in the DE2_System directory (or where you created your project).

**b.** Name your Project name “hello_world”.

**c.** Select the *“Hello World”* Template. click next.

**d.** Click finish

e. Open the .bsp file with the BSP editor (NiosII->BSP Editor). This time we don’t care about a minimal software footprint, so we won’t select enable_reduced_device_drivers select enable_small_c_library as we did last time. We will just use the default settings. Click generate and then exit.

**29.** Copy the contents of lights.c file from last time and add it to the hello_world.c source file.

30. Check the SOPC Builder and to see what the BASE ADDRESSES are for the switches and LEDs. Update your code with these new value since they are most likely different.

31. Add several messages using the printf() function that will print out before the while loop is entered.

**Part 4 – Run the Software**

32. Compile and run your program like you did in the previous lab. You should see your printf statements show up in the console window.
33. Note: Always save before compiling, otherwise Eclipse will compile your file in its unsaved state (Eclipse hasn't figured this out yet….)
34. If Run doesn't work, you will have to configure it in Run->Configurations.

**Have the instructor verification sheet (given below) signed off when you get the printf() function printing statements to the console window. The LEDs and switches should still work as in lab 2.**

**Instructor Verification Sheet**

# Lab #3
# Adding SDRAM to your SOPC

**EELE 475**
**HARDWARE AND SOFTWARE ENGINEERING**
**FOR EMBEDDED SYSTEMS**

Name : _____

**Demo #1 : Show that printf() statements in your code show up in the console window.**

Verified: _____ Date:_____

**Demo #2 : The LEDs and Switches still work as in lab #2**

Verified: _____ Date:_____