DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
UNIVERSITY OF BRITISH COLUMBIA
CPEN 391 – Computer Systems Design Studio
Fall 2015/2016 Term 2

Tutorial 1.6
Adding a Driver for the 16x2 LCD Display to your NIOS System

In this tutorial, we will show how to add a new peripheral to your NIOS system. The peripheral we will add is a driver for the 16x2 LCD Display. In doing so, you will learn how to use Hardware Abstraction Layer (HAL) functions.

The steps we will go through are:

1. Add the peripheral to the QSYS Project and "generate" new hardware
2. Modify the top-level VHDL file to support the new inputs and outputs
3. Compile the hardware using Quartus II and program the DE2 board
4. Write software using HAL to send data to the 16x2 LCD Display

Before starting this tutorial, you must have completed Tutorials 1.2, 1.3, and 1.4. We will start with the project from Tutorial 1.3.

1.0 Adding the Peripheral to the QSYS Project

Starting from Quartus II, open your project from Tutorial 1.2. Enter QSYS and edit the QSYS project from that tutorial. As a reminder, your QSYS project will look something similar to Figure 1.

From the Component Library (on the left), double click on **University Program->Audio & Video->16x2 Character Display**. A form for this peripheral will pop-up. Click on **Finish**, and the peripheral will be added to your QSYS Project, as shown in Figure 2.

To connect the device, do the following:

1. Connect the **clock_reset** pin to the **sys_clock**
2. Connect the **clock_reset_reset** pin to **clk_reset**
3. Connect the **Avalon LCD Slave** pin to the **data_master** pin
4. Choose an address range for this device by clicking on the **Base** column beside **Avalon LCD Slave** pin, and entering **0x00002030**
5. Click on **Click To Export** beside the **external_interface** pin, and enter **lcd_data**

When you finish these connections, your project should look as in Figure 3. Compare carefully to make sure you have not made any mistakes. When you are

satisfied, go to the **Generation** tab, and press **Generate.** The generation should complete with 0 errors and 0 warnings (of not, go back and check your connections).

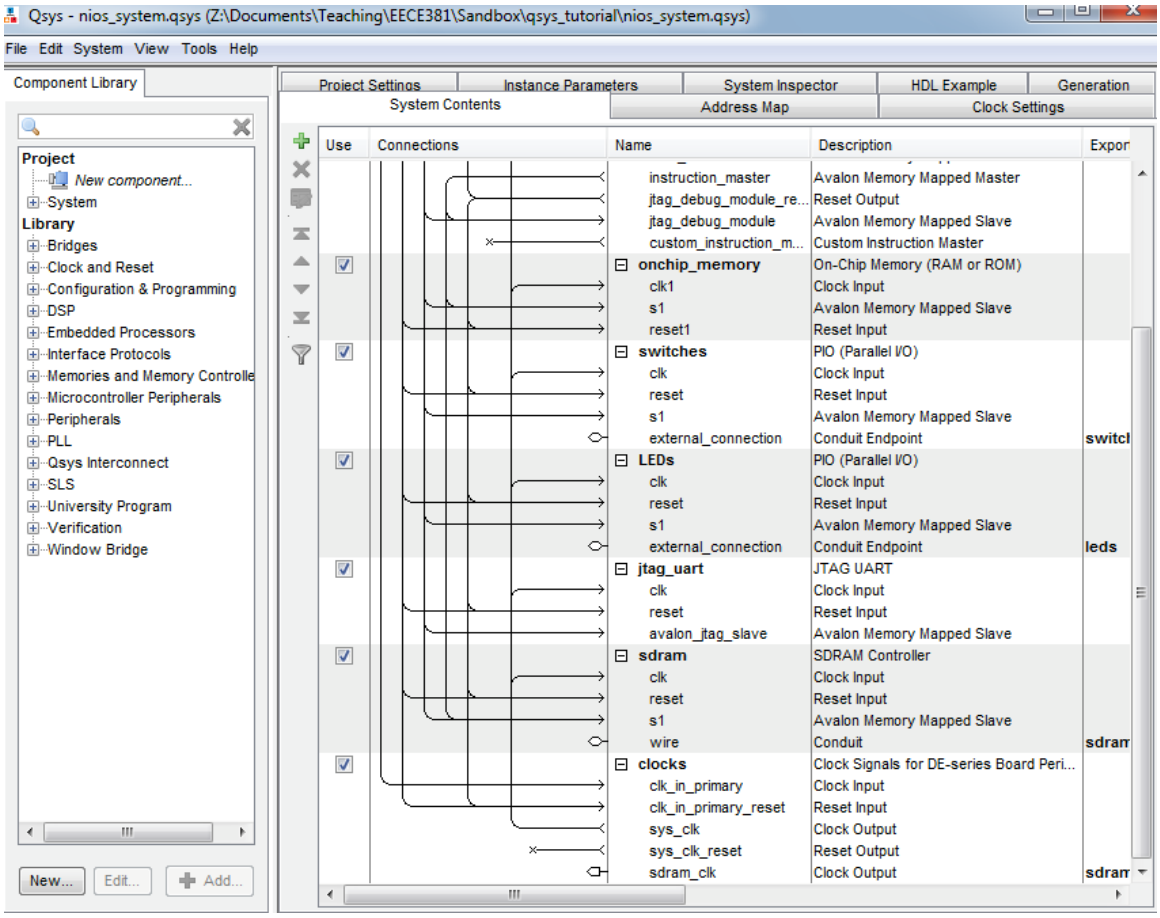

Figure 1: QSYS Project from Tutorial 1.2 (our starting point in this tutorial)

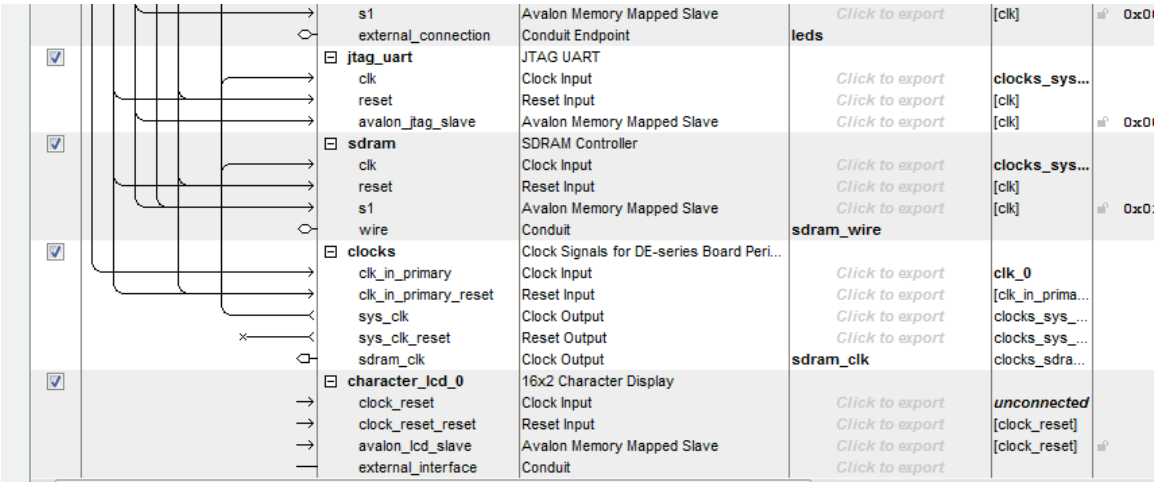

Figure 2: Peripheral Added (but before connections are made)

2.0 Modifying your top-level design and Compiling Your Project

The NIOS core you have just created contains some new output signals, compared to the one you created in previous tutorials. These are the signals that connect the NIOS core to the LCD display, as shown in Figure 3. Depending on when you took EECE 353, you might remember the meaning of these signals from Lab 2 in that course; if not, it doesn't really matter.

To see the names of these new signals, look in your project directory: **qsys_tutorial/nios_system/synthesis** for file **nios_system.v** . Open this file. This is a Verilog file, but even though you don't know Verilog, it should be clear that the top of the file lists the inputs and outputs of the NIOS system (the inner-most box in Figure 4). As you can see, the NIOS system has new outputs named **lcd_data_ON, lcd_data_BLON, lcd_data_EN, lcd_data_RS, lcd_data_RW**, and a bidirectional port named **lcd_data_DATA** (the latter is 8 bit wide). You can also see a list of the new signals by looking at the 'HDL Example' tab of QSYS.

You now need to modify your top-level file (which, if you have been following the tutorials exactly, is still **qsys_tutorial/lights.vhd**) to account for these new signals.

To do this, open the **lights.vhd** file. Scroll down to find the component declaration (starting with "**COMPONENT nios_system PORT**). As you learned in EECE 353, this component declaration must match the i/o specification in your submodule (which in this case is **nios_system.v**) exactly. In our case, we need to add the new signals to this component declaration. When you do this, your component declaration will look something similar to that in Figure 5.

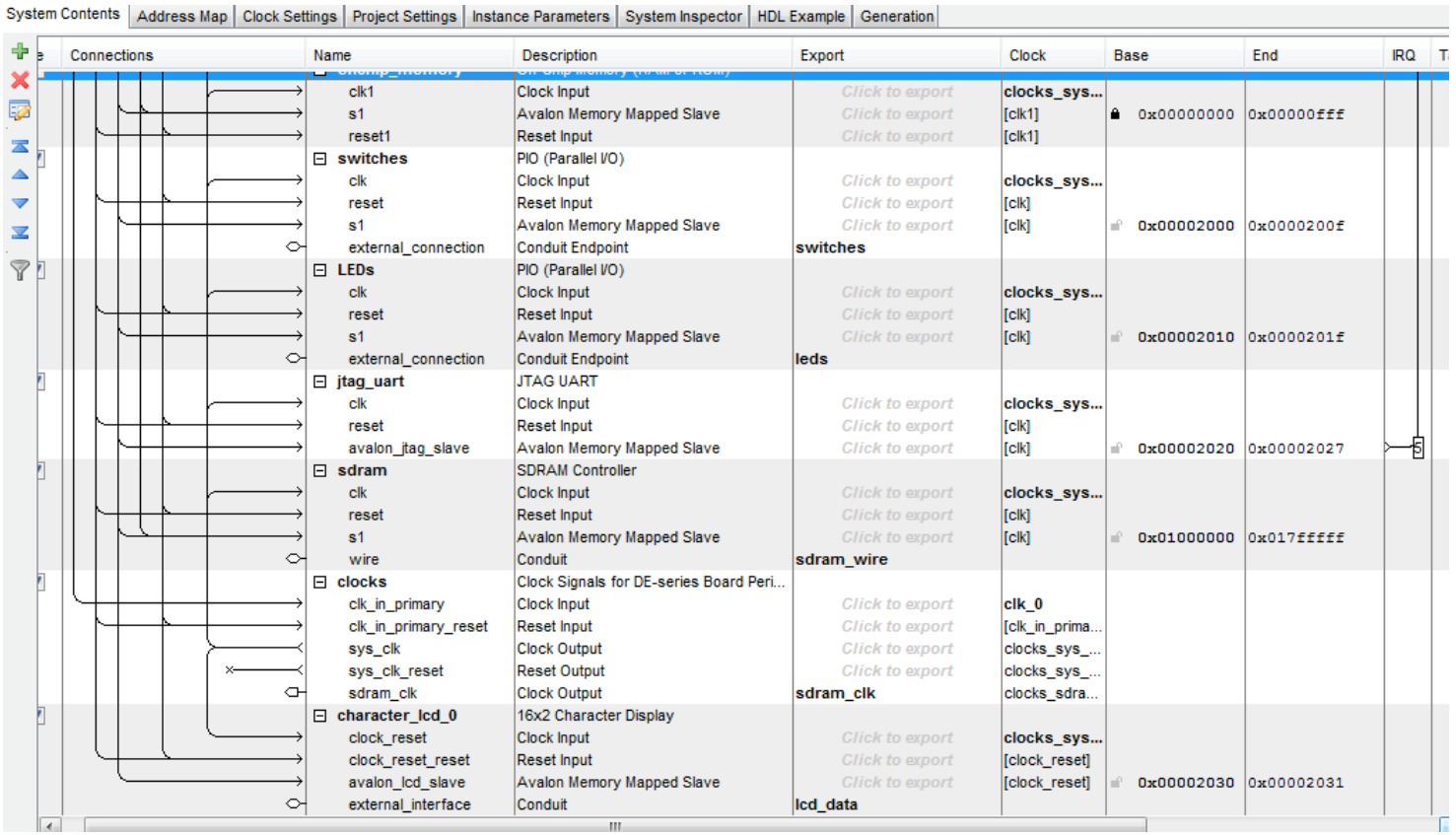

Figure 3: Connections to character_lcd_0

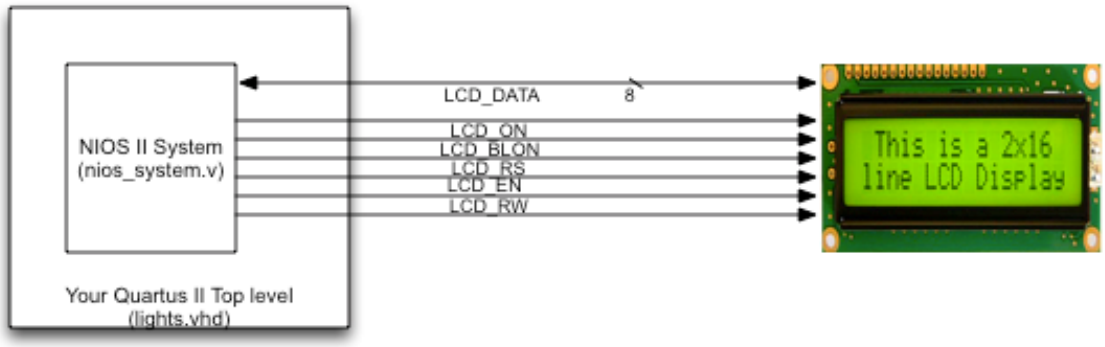

Figure 4: Connections between your NIOS System and the LCD Display

qsys_tutorial3 — vim — 74×30

```
ARCHITECTURE Structure OF lights IS
   COMPONENT nios_system PORT (
      clk_clk : IN STD_LOGIC;
      reset_reset_n : IN STD_LOGIC;
      sdram_clk_clk : OUT STD_LOGIC;
      leds_export : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
      switches_export : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      sdram_wire_addr : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
      sdram_wire_ba : BUFFER STD_LOGIC_VECTOR(1 DOWNTO 0);
      sdram_wire_cas_n : OUT STD_LOGIC;
      sdram_wire_cke : OUT STD_LOGIC;
      sdram_wire_cs_n : OUT STD_LOGIC;
      sdram_wire_dq : INOUT STD_LOGIC_VECTOR(15 DOWNTO 0);
      sdram_wire_dqm : BUFFER STD_LOGIC_VECTOR(1 DOWNTO 0);
      sdram_wire_ras_n : OUT STD_LOGIC;
      sdram_wire_we_n : OUT STD_LOGIC;
      lcd_data_DATA : inout STD_LOGIC_VECTOR(7 downto 0);
      lcd_data_ON : out STD_LOGIC;
      lcd_data_BLON : out STD_LOGIC;
      lcd_data_EN : out STD_LOGIC;
      lcd_data_RS : out STD_LOGIC;
      lcd_data_RW : out STD_LOGIC);
   END COMPONENT;
```

Figure 5: Component declaration for top-level file

Next, you need to add a connection between a signal and the new ports of the submodule. To do this, scroll further down to the component instantiation, and add the corresponding assignments, so that the instantiation looks as shown in Figure 6.

```
NiosII: nios_system PORT MAP (
    clk_clk => CLOCK_50,
    reset_reset_n => KEY(0),
    sdram_clk_clk => DRAM_CLK,
    leds_export => LEDG,
    switches_export => SW,
    sdram_wire_addr => DRAM_ADDR,
    sdram_wire_ba => BA,
    sdram_wire_cas_n => DRAM_CAS_N,
    sdram_wire_cke => DRAM_CKE,
    sdram_wire_cs_n => DRAM_CS_N,
    sdram_wire_dq => DRAM_DQ,
    sdram_wire_dqm => DQM,
    sdram_wire_ras_n => DRAM_RAS_N,
    sdram_wire_we_n => DRAM_WE_N,
    lcd_data_DATA => LCD_DATA,
    lcd_data_ON => LCD_ON,

    lcd_data_EN => LCD_EN,
    lcd_data_RS => LCD_RS,
    lcd_data_RW => LCD_RW,
    lcd_data_BLON => LCD_BLON );
```

Figure 6: Component Instantion in top-level file

Finally, you need to add these signals as outputs of your top level design file. Scroll up to the top of **lights.vhd**, and modify the Entity Part of the description to include the new output signals. The result will appear as in Figure 7. Note that **LCD_DATA** is of mode "inout", and the others are all of mode "out". Save your **lights.vhd** file.

Now, in Quartus II, choose Start Compilation to compile your design with the new NIOS II architecture and your new top-level. It will take some time, you should have no errors (you should expect some warnings as in Tutorial 1.1).

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
ENTITY lights IS
   PORT (
      SW : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      KEY : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
      CLOCK_50 : IN STD_LOGIC;
      LEDG : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
      DRAM_CLK, DRAM_CKE : OUT STD_LOGIC;
      DRAM_ADDR : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
      DRAM_BA_0, DRAM_BA_1 : BUFFER STD_LOGIC;
      DRAM_CS_N, DRAM_CAS_N, DRAM_RAS_N, DRAM_WE_N : OUT STD_LOGIC;
      DRAM_DQ : INOUT STD_LOGIC_VECTOR(15 DOWNTO 0);
      DRAM_UDQM, DRAM_LDQM : BUFFER STD_LOGIC ;
      LCD_DATA : inout STD_LOGIC_VECTOR(7 downto 0);
      LCD_ON, LCD_BLON, LCD_EN, LCD_RS, LCD_RW : out STD_LOGIC);
   END lights;

ARCHITECTURE Structure OF lights IS
```

Figure 7: New Entity Part of lights.vhd

3.0 Using HAL to Create Software that Uses the LCD Display

As in Tutorial 1.3, open Eclipse using **Tools->NIOS II Software Tools for Eclipse** in Quartus II. In Eclipse, chose **File->New->NIOS II Application and BSP from Template**. In the pop-up window, find the appropriate **.sopcinfo** file (from the Quartus II project you have just compiled) and give your project a name (I choose **lcd_demo**). I just used "**hello world**" as my template. As an aside -- note that you don't actually have to create new projects from a template; you can experiment to find other ways to do it; just be sure you end with both an application project and a BSP project.

You will now write software that uses HAL. Appendix A of this document is a datasheet from Altera that describes the 16x2 core in more detail. The first part of the Appendix is useful if you want to engage in "bare metal" programming, as described in Tutorial 1.3. For now, skip ahead to Section 4.2 of the Appendix. This section will show you a list of all the HAL functions related to this core as well as some sample code. Study to sample code to ensure you understand how it is supposed to work.

To demonstrate the functionality, download the NIOS II project (**using NIOS II-> Quartus II Programmer** as before), and modify your **hello_world.c** file by adding HAL calls as in the sample code in Section 4.2 of the appendix.

Play special attention to the line:

```
alt_up_character_lcd_open_dev ("/dev/Char_LCD_16x2”)
```

The argument to this function is the name of the LCD interface core you added in QSYS. If you followed the above instructions carefully, your LCD interface is probably called something like “character_lcd_0”. In that case, you would change the argument to say:

```
alt_up_character_lcd_open_dev ("/dev/character_lcd_0")
```

If you get an error “Could not open the LCD device” when you run this program, check to make sure the name matches your QSYS design as described above.

Run your software by right clicking your project name, and choosing **Run As->NIOS II Hardware**.

You can now experiment with the HAL calls for this device. When you are confident you understand how to write software to control the LCD display, you are done with this tutorial.

The data sheet for the LCD display from Altera is given starting on the next page