UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering

EEC181 DIGITAL SYSTEMS DESIGN PROJECT WINTER 2016

LAB 1: DE1-SOC System Development Tutorial and Exercises

This tutorial will give you a review of using Verilog HDL in Altera Quartus II to develop embedded systems implemented on FPGAs. You will be introduced to the DE1-SOC (System on Chip) Altera Development Board with both the Cyclone V FPGA system and ARM Cortex-A9 Hard Processor System (HPS). This board allows the use of both FPGA and full processor to function in conjunction, to provide extended capabilities in both the FPGA and Processor domains combined sharing resources. This tutorial presents an introduction to Altera's Qsys system integration tool, which is used to design digital hardware systems that contain components such as processors, memories, I/O interfaces, timers, and the like. We will also presents an introduction to the Altera Monitor Program that can be used to compile, assemble, download and debug programs for ARM Cortex-A9 processor.

It is recommended that you read and work through the Altera references below to obtain additional practice on the tool chain. This lab assumed you have knowledge of Verilog HDL, FPGA programming, and C source code experience.

Before You Begin

Only the First Time

Open a terminal and type this command to setup the environmental variables. (Notice that you have to do this only once. This sets the environmental variables permanently).

```
setup quartus
```

In this course we use Quartus II 64-bit for designing systems, synthesizing and compiling our projects. On Linux machines you can start the program as follows.

```
quartus --64bit
```

- Notice that there is a space between quartus and –64bit.

After you start quartus, it opens a window similar to what is shown in Figure 1 and asks for License information. In this step please make sure you choose the **last** option and then click on OK.

Then you will see the option window which is shown in Figure 2. In front of the license file, please write the following address. Before clicking on OK, please certify that you can see list of supported products under "**Licensed AMPP/MegaCore functions"** as it is illustrated in Figure 2.
Click on OK. Quartus is ready to use.

```
28333@license.engr.ucdavis.edu
```

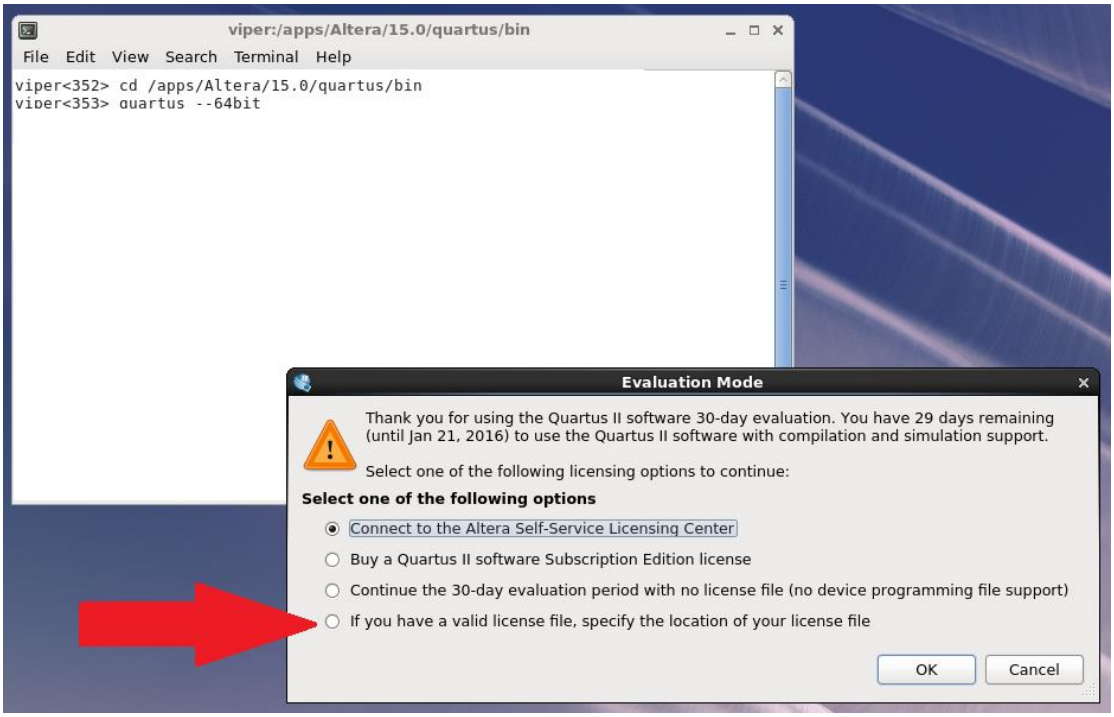


Figure 1. Setting the license server, step 1

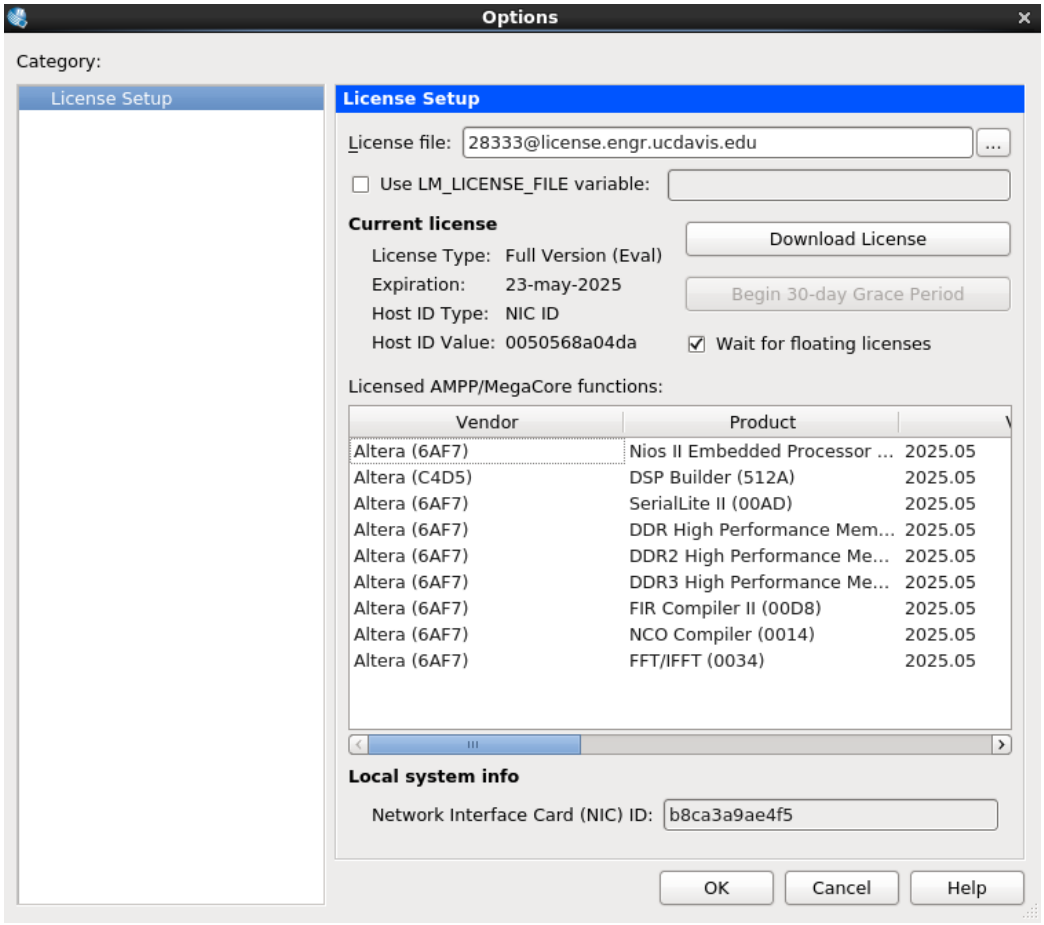


Figure 2. Setting up the license server, step 2

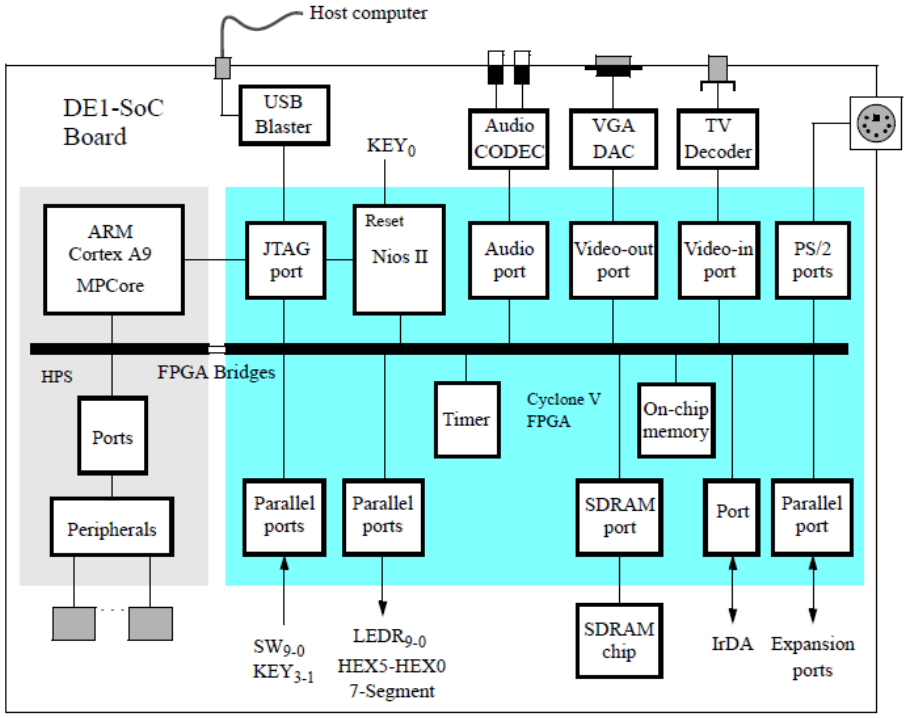


Figure 3. Block diagram of DE1-SOC design. Left is HPS features, Right is FPGA features

I. INTRODUCTION TO QSYS WITH BASIC HPS SYSTEM DESIGN

1. Use New Project Wizard to create a new project:
   Directory: /home/[UserName]/Desktop/lab1
   Project name: lab1
   Top-level entity name: lab1
   Project Type: Empty Project
   Add Files: None. Just click on Next
   Device family: Cyclone V
   Device Name: 5CSEMA5F31C6
   Simulation: Verilog HDL

2. Run Qsys by Selecting Tools -> Qsys

   The Qsys tool is used in conjunction with Quartus II. It allows you to easily create a system by simply selecting the desired functional units and specifying their parameters. You will now design a mini-SOC computer by implementing part of the system in Figure 3, we have to instantiate the following functional units. In the rest of this document, this process is explained.

   ARM Cortex A9 HPS
   JTAG UART
   On-Chip Memory
   PIO (Parallel I/O) for LED's

PIO (Parallel I/O) for HEX3_HEX0
PIO (Parallel I/O) for HEX5_HEX4
PIO (Parallel I/O) for Slider Switches
PIO (Parallel I/O) for Pushbuttons

3. On the opening screen of Qsys, a clock is automatically added for you. This is the system clock that will be used throughout your system. There are many variety of clocks we can use, we will use the default. Change the name of Clk_0 to **System_Clk** by Right Click on the device name and select Rename. Verify that the Clock Settings are configured for an External Clock Source with frequency of 50MHz by Right Click and select Edit. Select Finish when completed to close the window.

4. In the Library menu located by default on the left side, select Processors and Peripherals > Hard Processor System and select Add. A customization window will pop up for you to configure the processor. Change the other settings to the following:

Enable MPU standby and event signals:	Check Box Unchecked
HPS-to-FPGA interface width:	128-bit
FPGA-to-HPS SDRAM Interface:	Select and Remove All FPGA-to-HPS SDRAM ports
Enable FPGA-to-HPS Interrupts:	Check Box Checked

- Go to the "SDRAM" tab and click on the "Memory Parameters" sub-tab. In the "Memory Initialization Options" section, change "ODT Rtt nominal value" under "Mode Register 1" to "RZQ/6".

5. Select Finish to close the HPS window.

6. Rename the HPS component module name to **Arm_A9_HPS**.

7. Add On-Chip Memory or RAM Memory:
   Basic Functions > On Chip Memory > On-Chip Memory (RAM or ROM)

Block type:	RAM (Writable)
Data width:	32
Total memory size:	16384 bytes
Remaining options:	default

   Note: These are the default settings, however, if you require more memory you can add more. The DE1-SOC has a total of 16KB of On-Chip memory, if even more memory is needed, you will need to use the 64MB SDRAM and/or 1GB DDR3 SDRAM Memory on the HPS.

8. Rename the On-Chip Memory component module name to **ram**.

9. Add JTAG Communication between the PC and the System so that you can see the address map of the system:

   Interface Protocols > Serial > JTAG UART

Options:	default

10. Rename the JTAG UART module name to **system_console**.

11. Add PIO port for driving LEDs:

    Processors and Peripherals > Peripherals > PIO (Parallel I/O)

Width (1-32 bits):	10
Direction:	Output

12. Rename the PIO component module name to **led**.

13. Add PIO port for switch inputs:

    Processors and Peripherals > Peripherals > PIO (Parallel I/O)

Width:	10
Direction:	Input
Hardwire PIO Inputs in test bench	Check Box Checked

14. Rename the PIO component module to **switches**.

15. Add PIO port for HEX3_HEX0:

    Processors and Peripherals > Peripherals > PIO (Parallel I/O)

Width:	32
Direction:	Output

16. Rename the PIO component module to **HEX3_HEX0.**

17. Add PIO port for HEX5_HEX4:

    Processors and Peripherals > Peripherals > PIO (Parallel I/O)

Width:	16
Direction:	Output

18. Rename the PIO component module to **HEX5_HEX4.**

19. Add PIO port for Pushbuttons:

    Peripherals > Microcontroller Peripherals > PIO (Parallel I/O)

Width:	4
Direction:	Input
Synchronously Capture:	Check Box Checked
Edge Type:	FALLING
Enable bit-cleaning for edge capture register	Check Box Checked
Generate IRQ	Check Box Checked
IRQ Type:	EDGE
Hardwire PIO Inputs in test bench	Check Box Checked

20. Rename the PIO component module to **pushbuttons.**

21. The possible system connections are displayed in the Connections column in the System Contents tab. Connect the system to the clocks, reset, and data lines. Clicking on the bubbles will connect each component to each other. A bold solid line indicate a successful connection. As the connections are made you see the Errors in the Description Tab will decrease. Your design should look like Figure 4.

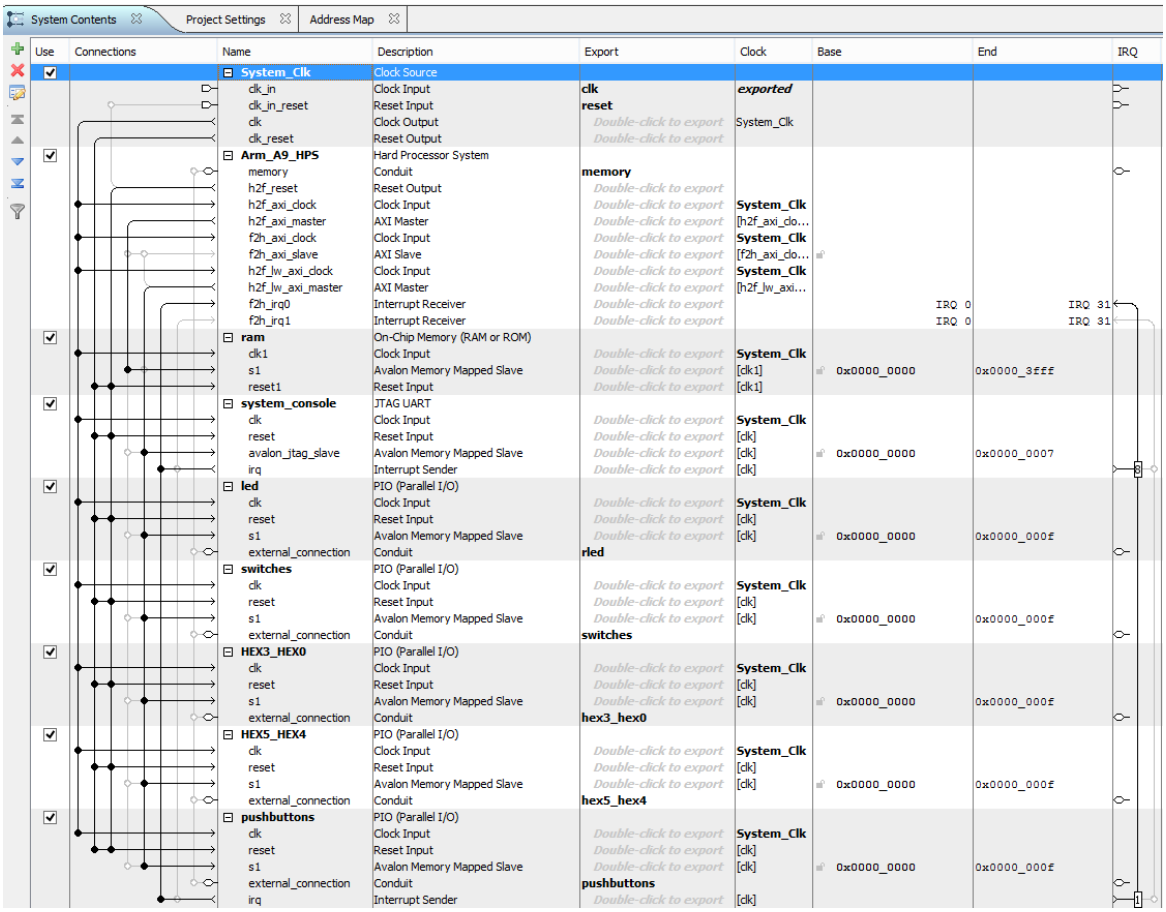


Figure 4. Connection Layout

22. The Export column is used to export pins to the Top-Level Design of the system. Double Click on "Double-Click to export" labels to add an export name to each component that needs external system access. Name the following external connections to its respective names:

    led -> external_connection -> rled
    HEX3_HEX0 -> external_connection -> hex3_hex0
    HEX5_HEX4 -> external_connection -> hex5_hex4
    switches -> external_connection -> switches
    Pushbuttons -> external_connection -> pushbuttons

23. In the IRQ column, change the Push Button IRQ to 1 and System_Console IRQ to 8

24. Your final system design should look like the Figure 5.

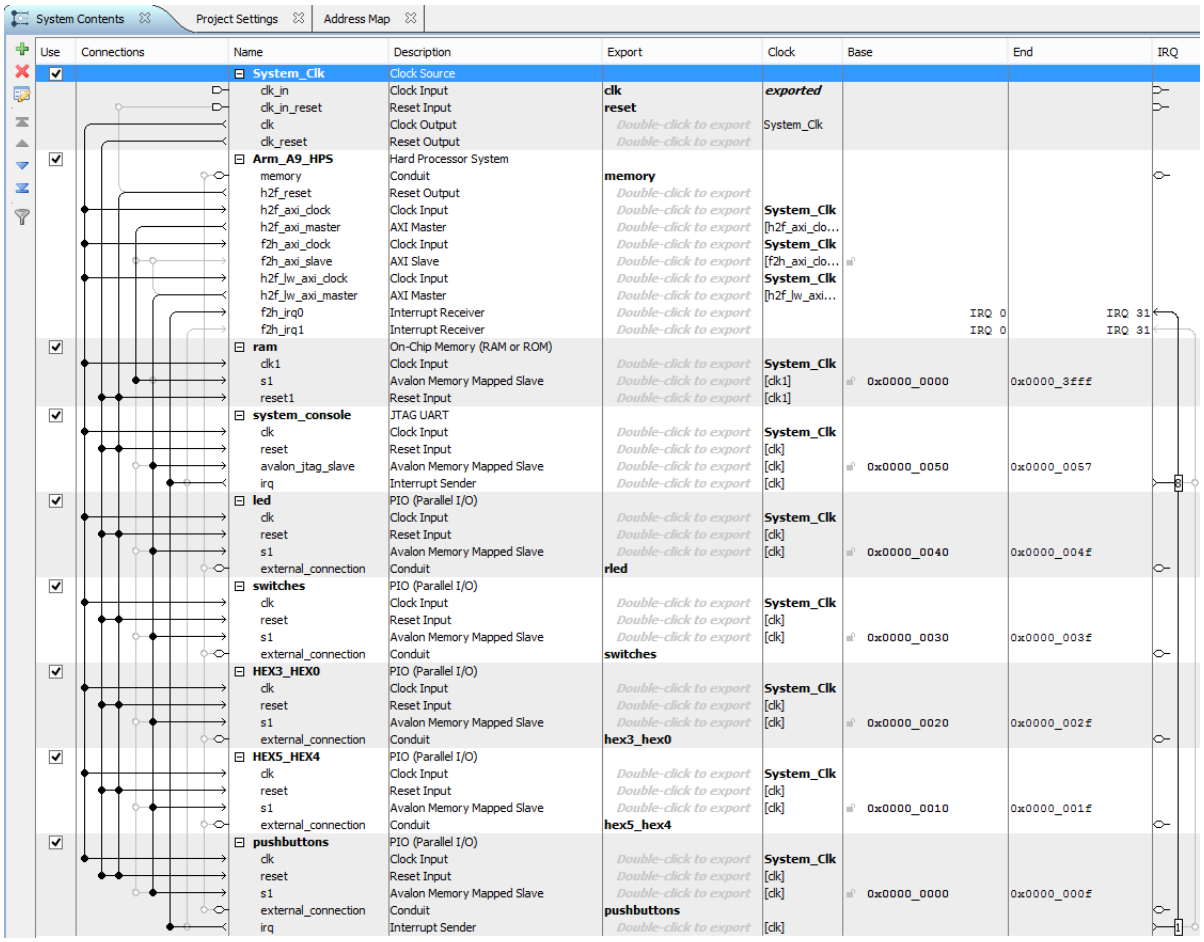


Figure 5. Overall System Setting

25. Notice all the base address are the same. We need to assign addresses to each peripheral so that we can access them in a C source program. Auto-Assign Base Addresses to the Memory Mapped IO:

    Select System > Assign Base Addresses

    Record the base address assignments for the I/O ports LEDs and Switches for use later in your software. The remainder of the errors should be resolved, leaving 1 warning.

    LED Base Address: _____
    Switch Base Address: _____
    HEX3_HEX0 Base Address: _____
    HEX5_HEX4 Base Address: _____
    Push Button Base Address: _____

    NOTE:

    All our LED's, switches, and push buttons are peripheral that are connected to the lightweight HPS-to-FPGA bridge on the HPS processor. The lightweight bridge's region of memory begins at address 0xFF200000, so to find the address of an FPGA peripheral, simply add the peripheral's base address written above to that address. In our case, the

LED peripheral was assigned a base address 0x00000040, therefore, the full address is simply 0xFF200040. Your design could be different.

26. Save your system in your Quartus II project folder, name it mysystem.qsys

27. Click the Generate > Generate HDL ... to generate the Verilog code and files that describe your system in the block symbol file. Click Generate. The system should generate without errors and some warnings. By default the generated files are located in the Quartus II project directory. If you do have errors, those errors would be disclosed to you, as usual, Altera errors are not very descriptive but often those errors are connection errors. The generated files include an .sdc file containing clock timing constraints, and a tcl script file. If simulation is enabled, simulation files are also generated.

    NOTE:

    Any changes to your system in the Verilog code, you will need to regenerate the system in Qsys, otherwise your changes will not be reflected. However, you do have the option to change the generated Verilog code outside of Qsys, the drawback is you will not be able to use Qsys anymore for your design.

28. Click Generate > HDL Example... This will show you what the module instantiation would look like, you can copy and paste this directly into your Top Level design to save time.

29. Exit out of Qsys.

30. Add the mysystem.qip file (IP Variation file) in the directory ../mysystem/synthesis to your Quartus II project. So that your designed system is included in your Quartus II project.

31. Now that you have designed a system, you need to instantiate it in Verilog and connect the PIN assignments from the DE1-SOC to your design. In the current Quartus II project, create a new Verilog HDL file. From Qsys, copy and paste the module instantiation.

    Save you file to lab1.v

    NOTE:

    The I/O signal names must match the names given in the DE1-SOC file in order to import pin number assignments. You can find the pin names of the DE1-SOC in the DE1-SoC user manual uploaded to the class website.

    Your lab1.v should look something similar to Figure 4.

```
module lab1(
   ////////////////////////////////////
   // FPGA Pins
   ////////////////////////////////////

   // Clock pins
   CLOCK_50,

   // Seven Segment Displays
   HEX0,
   HEX1,
   HEX2,
   HEX3,
   HEX4,
   HEX5,

   // Pushbuttons
   KEY,

   // LEDs
   LEDR,

   // Slider Switches
   SW,

   ////////////////////////////////////
   // HPS Pins
   ////////////////////////////////////

   // DDR3 SDRAM
   HPS_DDR3_ADDR,
   HPS_DDR3_BA,
   HPS_DDR3_CAS_N,
   HPS_DDR3_CKE,
   HPS_DDR3_CK_N,
   HPS_DDR3_CK_P,
   HPS_DDR3_CS_N,
   HPS_DDR3_DM,
   HPS_DDR3_DQ,
   HPS_DDR3_DQS_N,
   HPS_DDR3_DQS_P,
   HPS_DDR3_ODT,
   HPS_DDR3_RAS_N,
   HPS_DDR3_RESET_N,
   HPS_DDR3_RZQ,
   HPS_DDR3_WE_N,
);

//=======================================================
//  PORT declarations
//=======================================================

////////////////////////////////////
// FPGA Pins
////////////////////////////////////

input CLOCK_50;
output [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
input [3:0] KEY;
output [9:0] LEDR;
input [9:0] SW;


////////////////////////////////////
// HPS Pins
////////////////////////////////////

// DDR3 SDRAM
output [14:0] HPS_DDR3_ADDR;
output [2:0] HPS_DDR3_BA;
output HPS_DDR3_CAS_N;
output HPS_DDR3_CKE;
output HPS_DDR3_CK_N;
output HPS_DDR3_CK_P;
output HPS_DDR3_CS_N;
output [3:0] HPS_DDR3_DM;
inout [31:0] HPS_DDR3_DQ;
inout [3:0] HPS_DDR3_DQS_N;
inout [3:0] HPS_DDR3_DQS_P;
output HPS_DDR3_ODT;
output HPS_DDR3_RAS_N;
output HPS_DDR3_RESET_N;
input HPS_DDR3_RZQ;
output HPS_DDR3_WE_N;

//=======================================================
//  REG/WIRE declarations
//=======================================================

wire  [31:0] hex3_hex0;
wire  [15:0] hex5_hex4;

assign HEX0 = ~hex3_hex0[6:0];
assign HEX1 = ~hex3_hex0[14:8];
assign HEX2 = ~hex3_hex0[22:16];
assign HEX3 = ~hex3_hex0[30:24];
assign HEX4 = ~hex5_hex4[6:0];
assign HEX5 = ~hex5_hex4[14:8];

//=======================================================
//  Structural coding
//=======================================================

   mysystem u0 (
     // FPGA Side

     // Global signals
     .clk_clk            (CLOCK_50),            //         clk.clk
     .reset_reset_n      (1'b1),                //       reset.reset_n

     .memory_mem_a       (HPS_DDR3_ADDR),       //      memory.mem_a
     .memory_mem_ba      (HPS_DDR3_BA),         //            .mem_ba
     .memory_mem_ck      (HPS_DDR3_CK_P),       //            .mem_ck
     .memory_mem_ck_n    (HPS_DDR3_CK_N),       //            .mem_ck_n
     .memory_mem_cke     (HPS_DDR3_CKE),        //            .mem_cke
     .memory_mem_cs_n    (HPS_DDR3_CS_N),       //            .mem_cs_n
     .memory_mem_ras_n   (HPS_DDR3_RAS_N),      //            .mem_ras_n
     .memory_mem_cas_n   (HPS_DDR3_CAS_N),      //            .mem_cas_n
     .memory_mem_we_n    (HPS_DDR3_WE_N),       //            .mem_we_n
     .memory_mem_reset_n (HPS_DDR3_RESET_N),    //            .mem_reset_n
     .memory_mem_dq      (HPS_DDR3_DQ),         //            .mem_dq
     .memory_mem_dqs     (HPS_DDR3_DQS_P),      //            .mem_dqs
     .memory_mem_dqs_n   (HPS_DDR3_DQS_N),      //            .mem_dqs_n
     .memory_mem_odt     (HPS_DDR3_ODT),        //            .mem_odt
     .memory_mem_dm      (HPS_DDR3_DM),         //            .mem_dm
     .memory_oct_rzqin   (HPS_DDR3_RZQ),        //            .oct_rzqin
     .rled_export        (LEDR),                //        leds.export
     .switches_export    (SW),                  //    switches.export
     .hex3_hex0_export   (hex3_hex0),           //    hex3_hex0.export
     .hex5_hex4_export   (hex5_hex4),           //    hex5_hex4.export
     .pushbuttons_export (~KEY[3:0])  // pushbuttons.export
   );

endmodule
```

Figure 4. Verilog Code to instantiate HPS embedded system

32. After instantiating your embedded system design into your project, you have to make the necessary pin assignments for your embedded system AND add pin assignments specific to the Altera DE1-SOC board. Select Assignments > Pin Planner to see the current pin assignments. It should look like Figure 5, where the pins do not have any assignments.

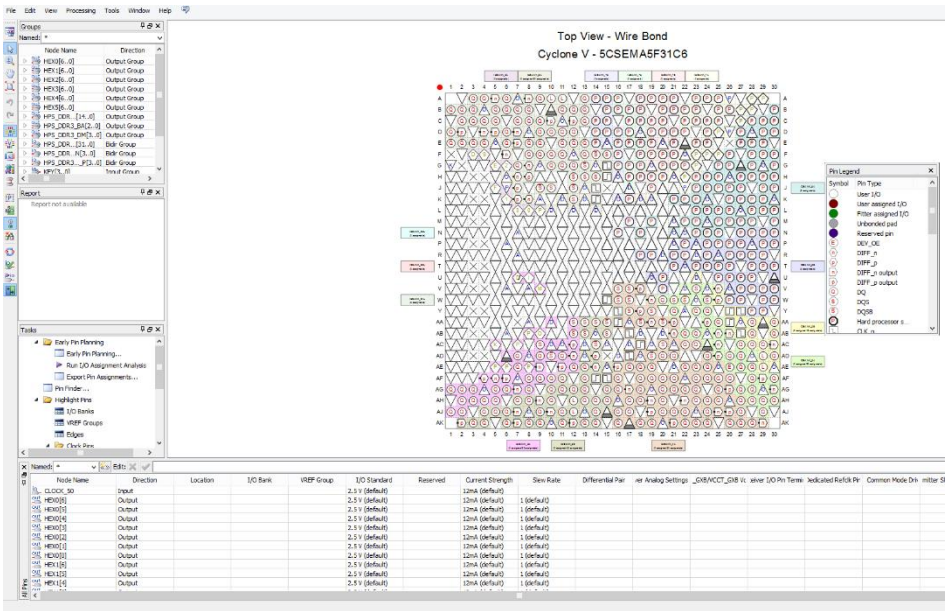


Figure 5. Empty pin asignment layout

33. Close the Pin Planner. Run the Analysis and Synthesis Tool so that Quartus II will determine which new pins need assigning, Processing > Start > Start Analysis & Synthesis.

34. When Qsys generates any HPS component, Qsys also generates the pin assignment TCL Script File (.tcl) to perform pin assignments. The script file name and location is at:

    ../mysystem/sythesis/submodules/hps_sdram_p0_pin_assignments.tcl

    Run this script to assign constraints to the SDRAM component. Select Tools > Tcl Scripts...

    Once the script has run, we can assign the rest of the standard DE1-SOC pins.

35. Perform the other standard DE1-SOC pin assignments, Assignments > Import Assignments…

    Browse to the directory containing the file DE1_SoC.qsf and select this file. This file will be made available to you on the class website. Click Advanced... and Deselect "Import assignments overwrite any conflicting assignments," Figure 6. Click OK twice to complete the Import Assignment.

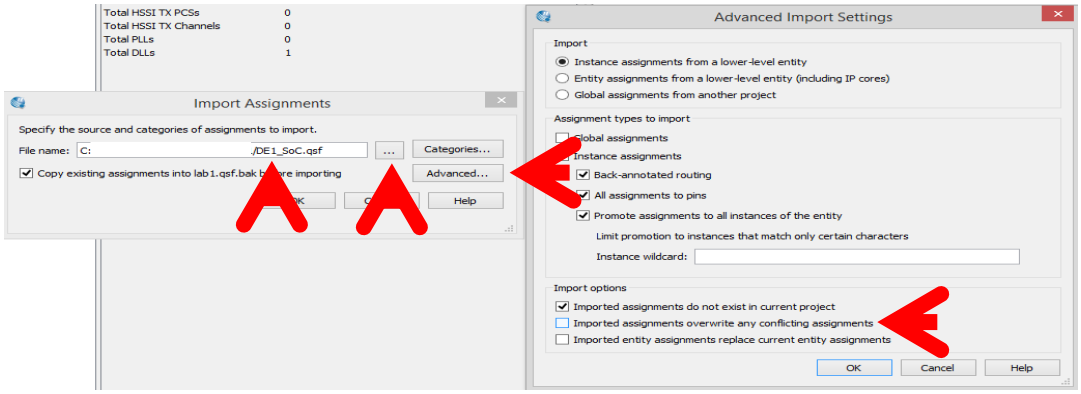


Figure 6. Standard Pin Assignments for DE1-SOC

NOTE:

You will need to reassign pin assignments every time you change the verilog code. Fortunately, Qsys generates a Tcl script which can add these assignments automatically.

36. Open Pin Planner again and this time you will see the pin assignments for each pin, Figure 7.

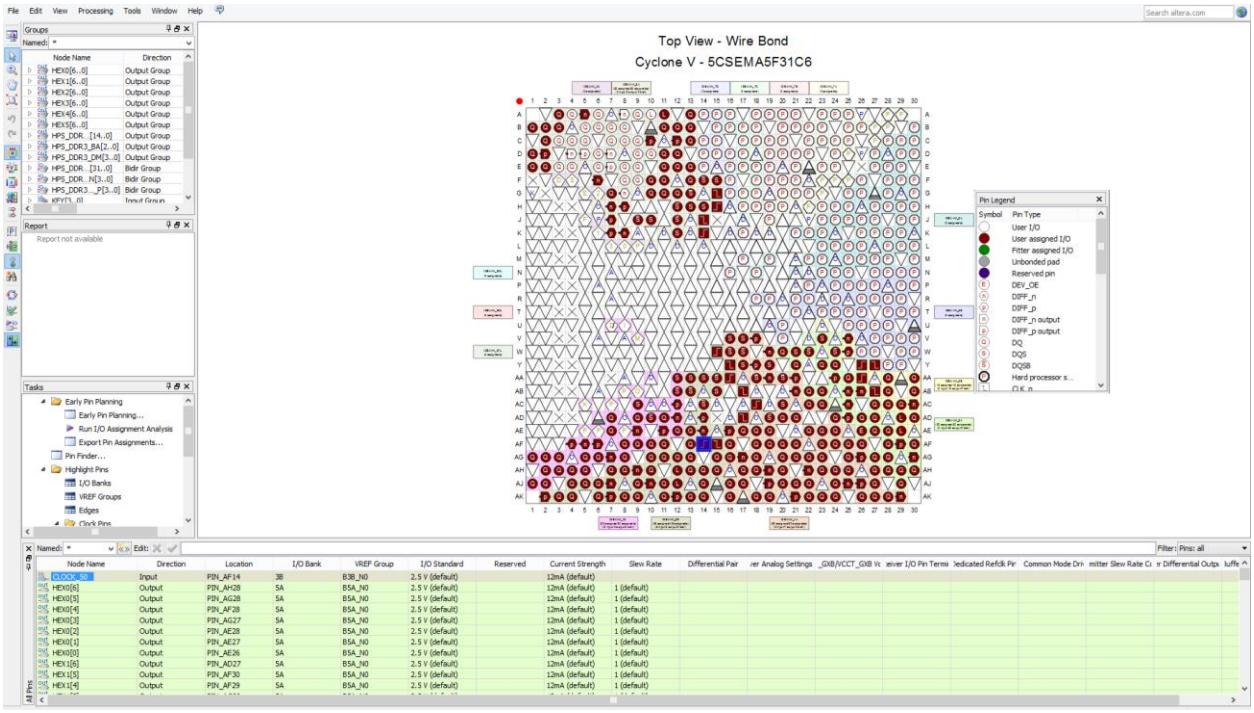


Figure 7. Entire design pin assigments for the DE1-SOC

37. Compile your design in Quartus II. It should compile without any errors. You may see some warning messages such as some signals being unused or having wrong bit-lengths of vectors; these warnings can be ignored.

38. Connect the USB cable from the DE1-SOC to your PC. Open the Quartus II Programmer (Tools > Programmer). Click the Hardware Setup Button and Verify the selected hardware is DE-SOC [USB-1] is selected. Click Auto Detect and select **5CSEMA5** device. If asked to overwrite changes, select Yes. You will see both the HPS and FPGA system. Your Verilog code is for the FPGA system so select the **5CSEMA5** and Download lab1.sof to the DE1-SOC board. You may need to select the Program/Configure checkbox. When this completes, you can close the Programmer window.

39. You have now successfully designed a basic DE1-SOC computer and have downloaded it onto the FPGA. The rest of this lab is to introduce how you can compile, download, and troubleshoot C source code on your mini-computer.

II. INTRODUCTION TO ALTERA PROGRAM MONITOR and DEBUG TIPS:

The Altera program monitor is a specialized tool that allows you to download code to the DE1-SOC board, to compile and run C code, to debug C code by setting breakpoints, line stepping, and memory edit, and memory inspections.

1. Open the Altera Monitor Program. (Open a terminal and type amp).

```
amp
```

2. Create a new project. File > New Project

3. Browse to your Lab1 Folder and enter "lab1" for the project name. Remember no spaces are allowed in Altera tools so any names must be continuous. Select ARM Cortex-A9 for the architecture selection. Click Next.

4. Select <Custom System> in the select a system box. In the system description file selection box, select mysystem.sopcinfo from your lab1 project folder. In the Quartus II programming (SOF) file selection box, select lab1.sof from your lab1 project folder. Click Next.

5. Select C program for Program Type. Click Next.

6. At this point, you will need to add a C source file to be compiled and downloaded to the DE1-SOC computer. Open a simple text editor or any source code editor you prefer. We will use Notepad for this example. Open Notepad and copy the C code in a text file named, led.c, and place the file into your lab1 project folder.

```
// Simple test program
int main(void)
{
        volatile int * led              = (int *) 0xFF200040;    // red LED address
        volatile int * switchptr        = (int *) 0xFF200030;    // SW slider switch address

        int switch_value;

        while (1)
        {
                switch_value = *(switchptr);
                *(led) = switch_value;
        }
}
```

7. Go back to the Altera Program Monitor Program and add the led.c source file into your project. Click Next.

8. Review the system parameters to make sure the following are selected, Click Next to continue:

Host Connection:	DE-SOC [USB-1]
Processor:	Arm_A9_HPS_arm_a9_0
Terminal Device:	system_console

NOTE:

Your DE1-SOC board must be connected and powered on for the Host Connection option to be available for selection.

9. Click Finish to use default memory address range settings.

10. When asked to download the system onto the board, you can click Yes to re-download the Quartus II system design you did above or click No and skip past this step. The tutorial will skip

past this step and click No since we already programmed the DE1-SOC with the Quartus II programmer tool previously.

11. At this point, your DE1-SOC only has the hardware Verilog code programmed and the HPS is ready for C source code. Compile your C source code by selecting Actions > Compile. Your code should compile without errors. You can view the errors in the lower right "Info & Error" box. If your code compiles successfully, a SREC file is generated.

12. Next load the compiled program onto the DE1-SOC, Actions > Load. Sometimes an error will pop-up during the loading process. Looking at the Error box, we see the processor was not able to stop and accept programs, don't worry, just try it again. It may take several tries but usually the second try will work. If you still cannot download to the DE1-SOC board after several tries, it means there is an error in the Verilog portion of your design. Unfortunately, these types of errors could be logic errors and therefore are only apparent during implementation. If successful you will see in the Disassembly window the assembly version of your code. There is a pre-loader followed by your source code, Figure 8.

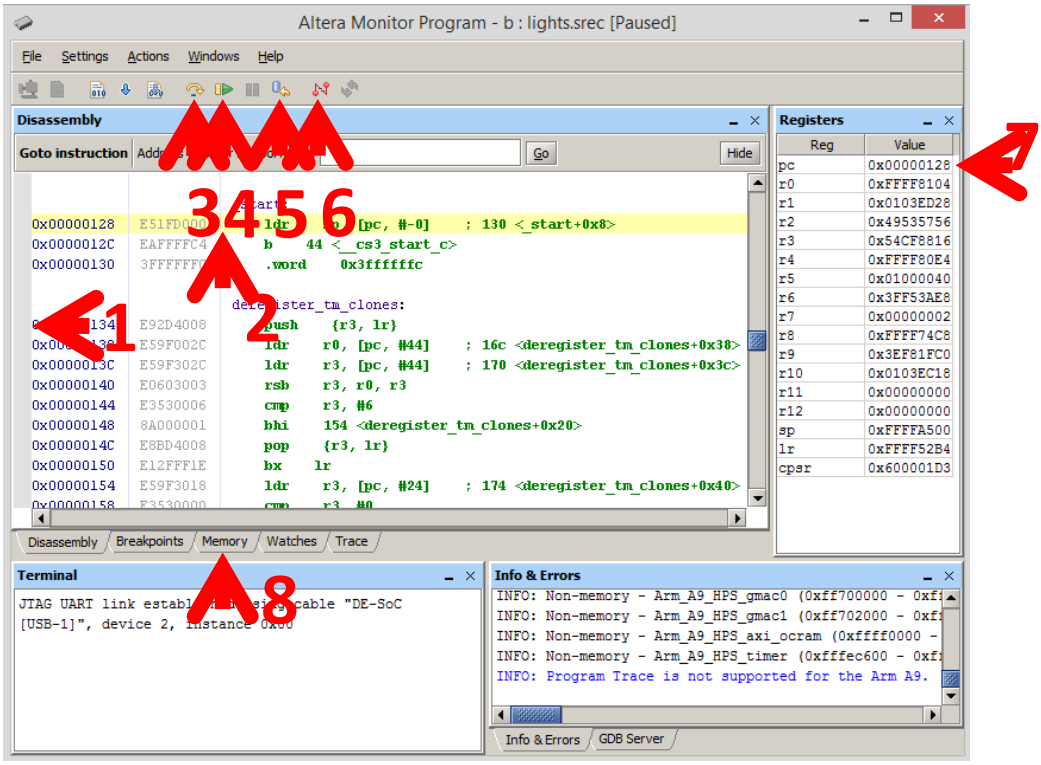


Figure 8. Dissassembly of the C source code after successful download to DE1-SOC

13. Figure 8, is your main debugging window. At this point your C program is downloaded but not running. Following the numbered arrows:

    1. Click into the grey column to set break points throughout the code. The program or PC (Program Counter) register will increment and run the program until a break point is hit, for user control before continuing.
    2. Yellow line indicated the current position of the PC
    3. Single-Step through one instruction at a time

4. Continue to run the program or play the program, the shaded pause button on the right will stop the program at its current instruction.
5. Restart program
6. Disconnect GDB session and from the DE1-SOC board.
7. Current register value. Red values indicate a recently changed value.
8. Memory map of the DE1-SOC, you can see what is stored in each memory address.

There are more advance features to the Altera Monitor Program. It will be helpful for your to read through the documentation on-line.

14. Select Play or Action > Continue, and toggle the switches on the DE1-SOC board. You should see that whatever switch is toggled, its corresponding LED will also turn on.

15. Congratulations you have successfully, designed a mini-computer using both HPS and FPGA resources, introduced to the memory mapped address space, programmed and uploaded a C source code to the HPS, and verified its functionality on the DE1-SOC board. The tutorial has introduced you to the entire tool chain and now it your turn to practice.

16. **Download your system to the DE1-SOC board and demonstrate it to the TA.**

III. PROGRAMMING ASSIGNMENT

The goal of this assignment is to give you practice using the PIO peripherals on the DE1-SOC. We will use the Seven Segment Displays to display two custom light patterns, one at a time dependent on user input from the four (4) pushbuttons. For the switch input, once the user has decided which LED pattern to display on the seven segment display, the user will press a push button to load the pattern to the seven segment display. The seven segment display pattern should not update while the switches are toggling, but the user will be able to preview the switch pattern on the red LED's. You will also implement a horizontal text scrolling feature that we often see on LED banners and a LED rolling pattern. You can also choose the direction of the horizontal scrolling, Left -> Right or Right -> Left. For ambitious students, you can also change the scrolling direction from horizontal to vertical. There are many methods to do this and you are free to proceed with your preference. However, we recommend using the mini-computer designed above for the hardware setup and C source code for a software implementation. At the end of the assignment, you will have experienced how hardware can control the flow of the program.

Modify the design as follows:

1. Display Patterns:
   I. The seven segment display should display: HELLO UUORLD
   II. Switch LED pattern on the seven segment display, i.e.

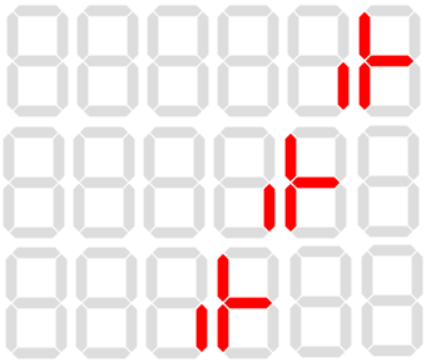
RED LED:	ON	OFF	OFF	ON	ON	ON	OFF	OFF	OFF	OFF
SWITCH POSITION:	UP	DOWN	DOWN	UP	UP	UP	DOWN	DOWN	DOWN	DOWN
SWITCHES:	SW9	SW8	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0

The seven segment display should display the RED LED pattern, i.e.

2. Scrolling and Rolling Movement:
    I. There are more letters than there are seven segment displays on the DE1-SOC, therefore, to be able to see the entire message. You will need to implement a horizontal scroll movement and loop around until a push button is pressed by the user, i.e.

etc...

    II. A rolling movement is implement for the red LED's and loop around until a push button is pressed by the user, i.e.



etc...

3. Push button user input and display control on the Seven Segment Display:
    I. There are 4 push buttons on the DE1-SOC. One push button will toggle between the patterns, either HELLO UUORLD or LED custom pattern displayed on the seven segment display.
    II. One push button will speed up the horizontal scrolling movement displayed on the seven segment display.
    III. One push button will slow down the horizontal scrolling movement displayed on the seven segment display.
    IV. One push button will pause the horizontal scrolling movement displayed on the seven segment display. Pushing the pause again will continue the scrolling movement. While at the pause state, other push buttons will have no effect on

the seven segment display behavior. However, the user can still use the switched to change the red LED pattern at any time.

Download your system to the DE1-SOC board and demonstrate it to the TA.

IV. References

Introduction to the Altera Qsys System Integration Tool:
ftp://ftp.altera.com/up/pub/Altera_Material/13.1/Tutorials/Introduction_to_the_Altera_Qsys_Tool.pdf
Making Qsys Components
ftp://ftp.altera.com/up/pub/Altera_Material/13.1/Tutorials/making_qsys_components.pdf
Altera Monitor Program Tutorial for ARM
ftp://ftp.altera.com/up/pub/Altera_Material/13.1/Tutorials/Altera_Monitor_Program_ARM.pdf