Digital Systems

Prof. James Clark

ECSE 325
Lab #5: System-on-Chip: HPS-FPGA Bridging

Winter 2018

Introduction

In this lab, you will learn how to use the FPGA hardware with the processor, both provided within the DE1-SoC board. You will be implementing bridging between the HPS and the FPGA to implement functions via switches and LEDs. This lab will take three weeks.

Creating a New Project

In order to have your design working, you should be implementing the following steps carefully.

1. You are provided with a preset file in "myCourses → Labs". Create a new folder for your project, and copy the file 'Altera_Cyclone_V_SOC_Development_Kit_HPS_Presets.qprs' to your project directory.
2. Start Quartus. Using the "New Project Wizard", name your project name as "qsys_lab". Do not add any files to your project. In the "Family, Device & Board Settings" window, click on the Board tab. Choose the DE1-SoC Board. Uncheck "Create top-level design file" (this will be defined later). Click "Finish".

Instantiating the Processor

3. Start Qsys from Quartus Tools menu. Under Qsys Tools/Options add the project directory to the IP Search Path. Click Finish. You shouldnt get any warnings and errors at this point.
4. From the Qsys IP Catalog Library, in the Library section, select the Processors and Peripherals/Hard Processor System/Arria V/Cyclone V Hard Processor System and click "Add".
5. In the window that pops up, select the Preset Altera_Cyclone_V_SOC_Development_Kit_HPS_Presets". This will load in various settings for the hard processor device. Click FINISH.

Instantiating the Memory

6. From the Qsys IP Catalog Library, select Basic Functions/On Chip Memory/On-Chip Memory (RAM or ROM) and click Add.
7. Set the Total Memory Size to 16384, keeping the Data Width at 32 and the Memory Type as RAM (Writable).
8. Uncheck Initialize memory content and click "Finish" to close the window. In Qsys rename the component to "SRAM" (by right-clicking on the name and selecting rename).

Instantiating I/Os

9. From the Qsys IP Catalog Library, select Processors and Peripherals/Peripherals/PIO (Parallel I/O) and click "Add". Set the bit-width to 10. Set direction to OUTPUT. This will be used to turn on/off the single element LEDs on the board. Rename the component to "LEDS".
10. Add another PIO component, but this time with direction set to INPUT. Set the bit width to 10. Rename to "SWITCHES". This will be used to read the state of the slide switches on the board.
11. Add another PIO component, with bit-width 32 and direction OUTPUT. Rename to "HEX3-HEX0". This will be used to control the four rightmost of the six 7-segment LEDs on the board.

Instantiating I/Os

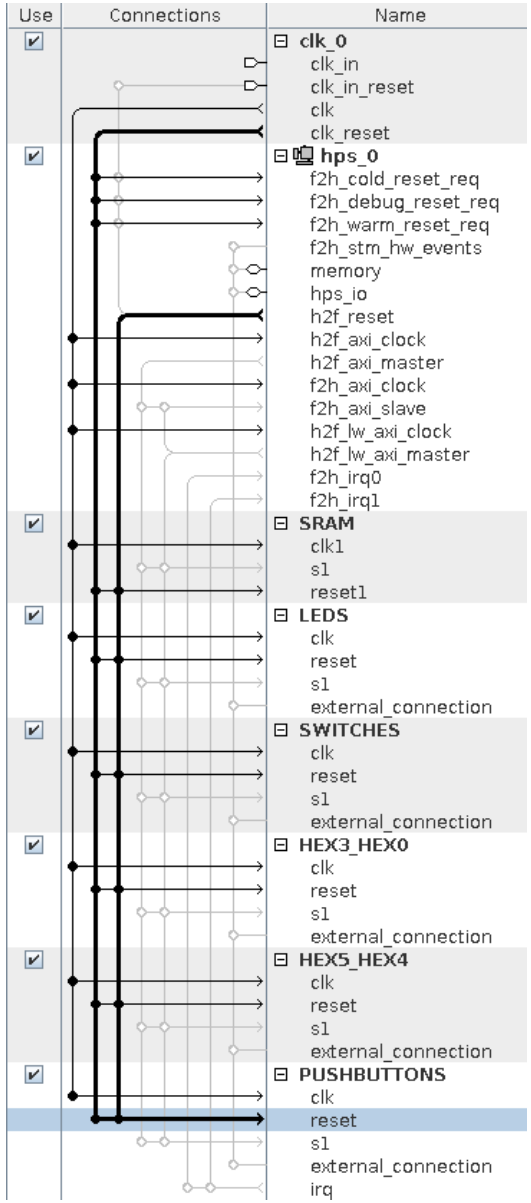
13. Add another PIO component, with bit-width 16 and direction OUTPUT. Rename to "HEX5-HEX4". This will be used to control the two leftmost of the six 7-segment LEDs on the board.
14. Add another PIO component, but this time with direction set to INPUT. Set the bit width to 4. Check the Synchronously Capture box, and set Edge Type to FALLING. Check the Generate IRQ box and set IRQ Type to EDGE. In Qsys rename to "PUSHBUTTONS". This will be used to read the state of the pushbuttons on the board.

Making the Connections - CLK

15. In the Systems Contents pane of the Qsys window, make the connections between the components. Connect all of the clk inputs of the added components to the clk clock output signal of the clk_0 component by clicking on the appropriate bubbles in the Connections column. Also connect this clk to the h2f_axi_clock, f2h_axi_clock and h2f_lw_axi_clock inputs.
16. Connect the clk_reset signal of the clk_0 component to the f2h_cold_reset_req, f2h_debug_reset_req, and f2h_warm_reset_req of the hps_0 component and to the rest inputs of the other added components. Do the same with the h2f_reset output of the hps_0 component (just to the other component reset inputs). This will OR the two reset signals together.

Making the Connections - CLK

The Qsys System Contents window should look like the following at this point:

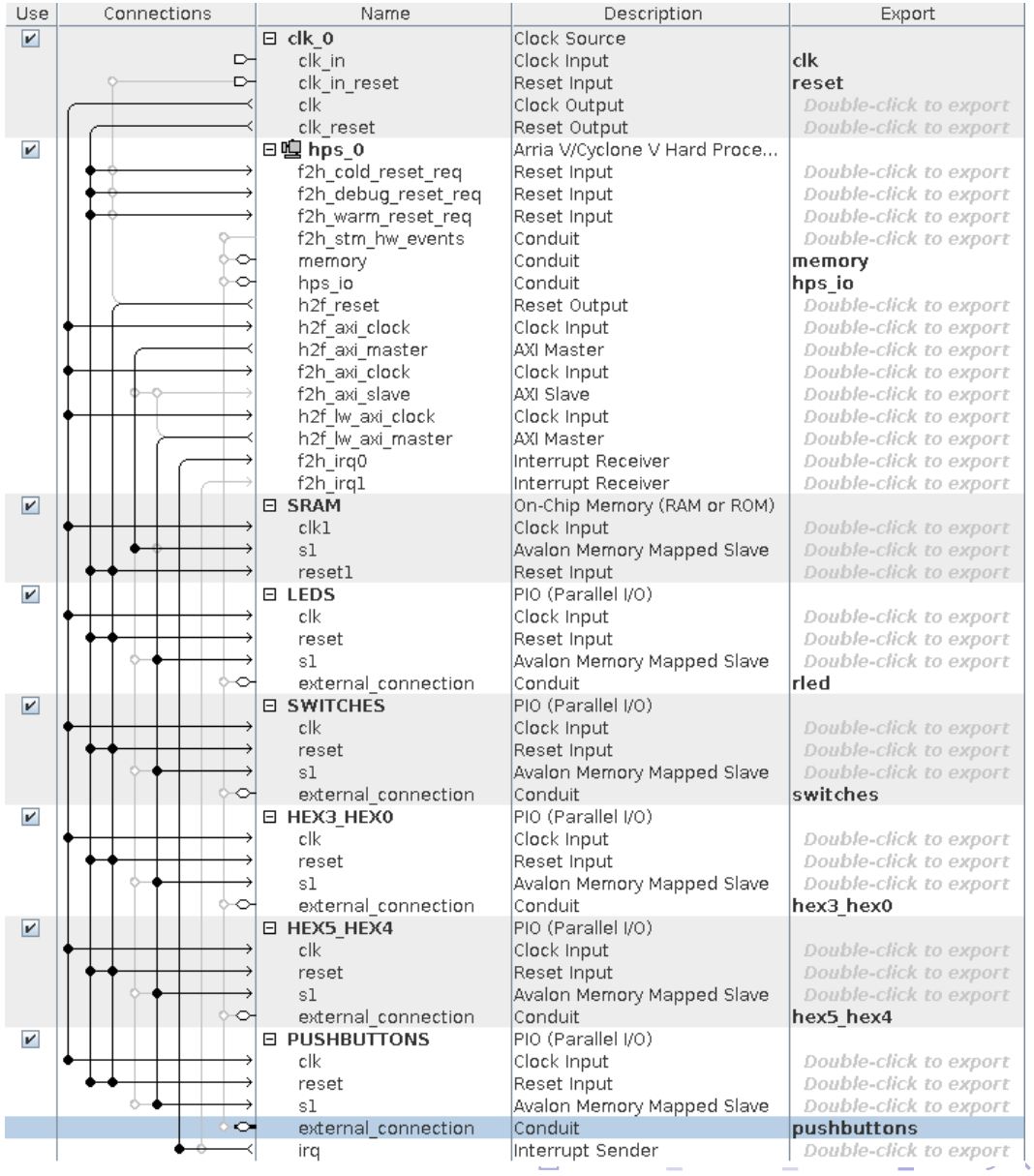


Making the Connections

17. Connect the irq signal of the PUSHBUTTONS component and the SYSTEM_CONSOLE component to the f2h_irq0 of the hps_0 component.
18. Connect the Avalon Memory Mapped Slave (bus) signals (these are labeled s1) of all components except the SRAM component to the h2f_lw_axi_master on the hps_0 component. Connect the s1 of the SRAM component to the h2f_axi_master of the hps_0 component.
19. Double click on the "Double-click-to-export" text in the "Export" column for the port named external_connection for the LEDS component. Type in "rled". This tells Qsys that this component will have some signals that will be connected externally (in this case to the red LEDs on the board). Similarly, export an external connection for the SWITCHES, HEX3_HEX0, HEX5_HEX4, and PUSHBUTTONS components (name these switches, hex3_hex0, hex5_hex4, and pushbuttons, respectively).

Making the Connections

The Qsys System Contents window should look like the following at this point:

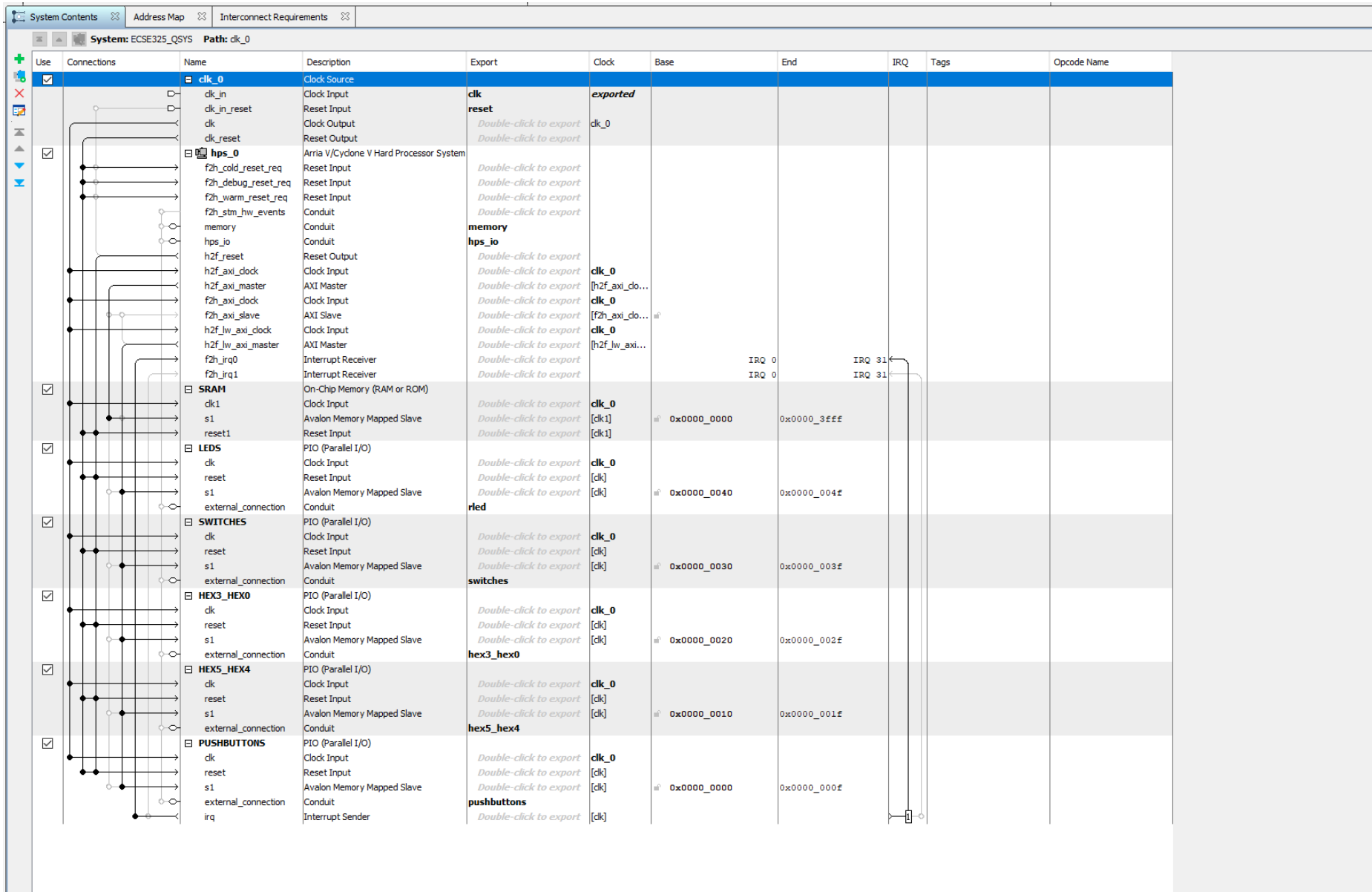


Making the Connections

20. In the IRQ column, change the PUSHBTTONS IRQ to 1.
21. Notice the Base column, these entries show the base addresses of the Avalon interface for each component. Currently they are all set to the same value, 0x0000_0000. We need to make them different. To do this, select System/Assign Base Addresses from the Qsys menu bar.
22. There should be no more error messages in the Qsys Message pane. There may be some warnings.
23. Save the Qsys settings file as "qsys_lab.qsys". Close the save system window when it finishes saving.

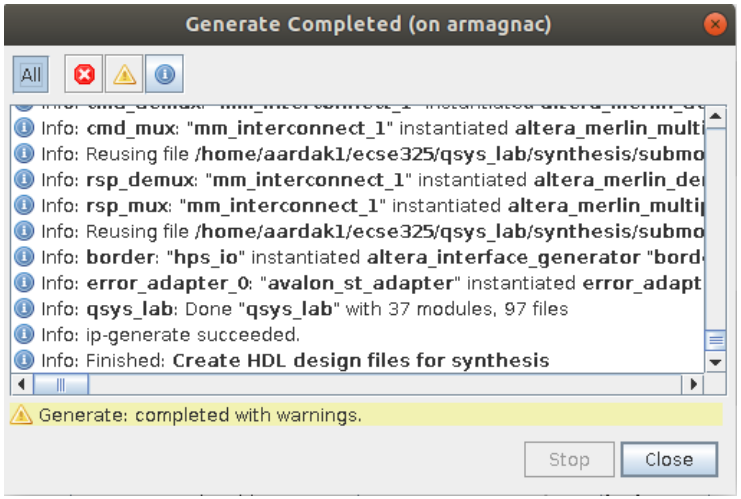
Making the Connections

The Qsys System Contents window should look like the following:



Generating VHDL Files

25. Click on “Generate HDL” in the “Generate” tab. In the “Create HDL design files for synthesis” box select “VHDL”. Uncheck the “Create timing...” box and select “None” for “Create simulation model”, since we will not be doing any simulation. The Output Directory Path should be your project directory. Click on “Generate”. This will produce the VHDL file which basically instantiates the components specified in your Qsys system. Exit Qsys when finished.



Project Setup

26. In Quartus, select Project/Add/Remove Files in Project. Then, add the qsys_lab.qip file in the directory qsys_lab/synthesis (which was created when you ran Generate in Qsys).
27. In Quartus, open the file /synthesis/qsys_lab.vhd (which should have been generated by Qsys). In the /Project menu tab, first select "Add Current File to Project", then select "Set as Top-Level Entity".
28. We will need to map the FPGA pins to the DE1-SoC board connections. Before doing this you should run the Analysis and Synthesis (under Processing/Start/Start Analysis and Synthesis/). This will determine what pins are unassigned and need to be mapped.

Pin Assignments

29. When Qsys generates any HPS component, Qsys also generates the pin assignment TCL Script File (.tcl) to perform pin assignments for the memory connected to the HPS. The script file name and location is at:
../qsys_lab/synthesis/submodules/hps_sdram_p0_pin_assignments.tcl
30. Run this script to assign constraints to the SDRAM component. To do so select Tools/Tcl Scripts... Once the script has run, we can assign the rest of the standard DE1SOC pins.
31. To assign the rest of the pins, select Assignments/Import Assignments and choose the file qsys_tutorial.qsf to load in some pre-defined assignments (otherwise it would quite tedious to enter them all yourself using the Pin Planner). Note that in this assignment, KEY(0) (the rightmost pushbutton) is mapped to a reset signal, and thus will not be accessible by the HPS via the Avalon bus (i.e. the signal PUSHBUTTONS[0] is not connected to anything).
32. Re-compile the design. There should be no errors.

You should be finished up to here by the first session of the lab.

Intel FPGA Monitor Program

33. Start up the Intel FPGA Monitor Program. This allows programming of the HPS (ARM CPU) on the Cyclone V device.
34. Select File/New Project, to startup the new project wizard. Enter the path to the Project Directory, and set the name as qsys_lab. Under the "Select a processor architecture" select "ARM Cortex-A9". Under "Specify a system" select the system "select <Custom System>". For the System description file choose "qsys_lab.sopcinfo" (which should have been generated for you by qsys), and for the FPGA programming file choose "output_files/qsys_lab.sof". In the "Specify a program type" page, choose "C Program". In the "Specify program details" page select the file "test_leds.c". This is a demo program to get you started. Click on "Finish". Do not download the system file yet, unless you have already connected the DE1-SoC board.

Testing the LEDs

35. Open the test_leds.c program file and take a look at it. It consists of some timer code and some code to read the DE1-SoC board switches and write to the DE1-SoC board LEDs (using the HPS-FPGA bridges/Avalon busses that were set up with Qsys).
36. Connect the DE1-SoC board to the desktop computer via a USB cable (use the USB connector on the left side of the board, next to the red power button). Turn on the board power.
37. Download the System settings to the board using /Actions/-Download System. This will download the .sof file generated when you compiled the qsys_lab.vhd top-level file. You might have problems here if the USB cable is not plugged in, or the board power is not on. You should otherwise get a "Download System Success" message window popup.

Testing the LEDs

38. Compile the program with /Actions/Compile. If there are no errors an ".sre" binary file will be generated that can be downloaded to the HPS over the USB cable. To do this, run /Actions/Load. You will know that this works when the debugger icons (just under the File/Edit/Actions bar) become active (not grayed out). Click on the "Continue" icon (or select /Actions/Continue). This will kick the HPS into action. You should now see the time clock counting up, and the red LEDS should now reflect the state of the slide switches on the board.

Observing the C Code for LEDs

39. Play around with the c-code and make some changes. For example, you could change the counting speed or have it count down. Or use the switches to initialize the count to some value. Try reading and writing to the SRAM memory component (e.g. generate random numbers to fill the memory slots and read them back to verify correct operation of the memory).
40. Next, you will use Qsys to add a new component of your own design. This will also be connected to the Avalon bus and communicate with the HPS via the light-weight bridge (h2f_lw_axi).

Creating the VHDL Components

40. First, make two new VHDL components. Call one of these "qsys_lab_custom_component.vhd" and the other "qsys_lab_function.vhd". The first file will just contain a component instantiation for the second design entity. We do it this way so that any changes will just be done to the second file. Since the first file will not be changing we will not require any re-generations with Qsys (after the initial one to tell it about the new component). For now, just make the architecture body of the second file be empty (just begin / end) and the first file just an instantiation of the second design entity as a component. The custom_component design entity should provide the necessary signals for connecting as a slave to an Avalon memory-mapped master. As such, it needs the port signals described in the following VHDL code:
    - clock (clock signal)
    - resetn (active low reset)
    - address (8 bit address to be used internally by the component to direct data)
    - readdata (32 bit data read from the component)
    - writedata (32 bit data to be sent to the component)
    - read (active when a read transaction is to be performed)
    - write (active when a write transaction is to be performed)
    - chipselect (active when a transaction is being performed)

Writing A New VHDL Function

Before you continue, you will have to implement a new VHDL function within "qsys_lab_function.vhd". Your function should read two separate data using the same Avalon bus, but from different addresses. You may use the least significant bit of the address input in your conditional statements to store different inputs to different instantiated signals in your function. You are asked to multiply these two numbers (using the least significant 16-bits of each) and send it to the interface to obtain the result at the display panel.

Connecting the Custom Hardware in Qsys

41. Start Qsys again, and load the qsys_lab.qsys file.
42. In the IP Catalog select "New Component" and click on "Add". An information window will appear, in which you will provide details about the new component. Give it the name "qsys_lab_custom_component" with Display name "custom_component". Leave the Group blank. For the description enter "qsys lab custom component". Enter your group student names in the Create by tab.
43. Select the "Files" tab in the Component Editor window. In the "Synthesis Files" section click on "Add File". Select the top-level VHDL file you just created (qsys_lab_custom_component.vhd). Then click on "Analyze Synthesis Files". This will look at the design entity descriptions to see what the input and output ports are and match them to Avalon interface signals. (if you get a "Info: Error: No modules found when analyzing null." message then you probably have a typo or syntax error in your VHDL file).

Connecting the Custom Hardware in Qsys

44. There will be some error messages in the Messages pane. This is because we still have some work to do!
45. Click on the "Signals & Interfaces" tab at the top of the Component Editor window. This shows the and interfaces in your top level component, and what Qsys thinks they are for relative to the Avalon bus interface. Most of these guesses are wrong, hence the errors in the Message window. Under the clock[1] signal, click on "add interface". Select "new Clock Input". The entry will change to "clock_sink". Drag the clock[1] signal to move it under the clock_sink interface. In the right hand part of the window, change Signal Type by selecting "clk". Drag the resetn signal from the Avalon_slave_0 interface to be under the "clock_reset" interface. Change the Signal Type to "reset_n". The other signal settings will probably be correct and dont need to be changed.

Connecting the Custom Hardware in Qsys

46. There will still be a few remaining error messages. On the left, click on "avalon_slave_0", and under "Associated Clock" select "clock_sink" and under "Associated Reset" select clock_reset. Finally, click on the "clock_reset" interface. Then set its Associated Clock to "clock_sink". All the error messages should now be gone!
47. Click Finish and choose "Yes, Save" to save the qsys related component description files. The new component will now show up in the IP Catalog, in the Project group.

Creating the VHDL Components

48. We can now add the new component to our Qsys system, just like we did with the other components earlier. Connects its ports just as you did for the PUSHBUTTONS component.
49. The address map will now overlap the other component addresses, so we have to redo the /System/Assign Base Addresses command. The addresses may have changed for the components, so you should make sure to check what the new addresses are so that you can change your c-code if needed. Note the memory address range for the new component. It actually has a 10-bit address range (0000-03FF) as the address indicates 8-bit words, whereas the data width is 32-bits.

Generating the new VHDL

50. Re-run the Generate HDL in Qsys. Then you can exit Qsys.
51. Open the Intel FPGA Monitor Program. Modify the test file (save it under a new name) to test the functionality of your new component. Keep in mind that the base addresses will (probably) need to be changed. Also remember that the addresses in the c-program must be in multiples of 4. The address passed to your component will be (c-code address base address)/4. If the address is not a multiple of 4 the HPS will likely hang.
52. Open the Intel FPGA Monitor Program. Modify the test file (save it under a new name) and test the functionality of your new component.

Writeup the Lab Report

You are required to submit a written report and your code to myCourses on the last day of classes (Monday) April 16.

- The report in the electronic file should be in PDF format.
- The report should be written in the standard technical report format.
- Document every design choice clearly.
- Everything should be organized for the grader to easily reproduce your results by running your code through the tools.
- The code should be well-documented and easy to read.
- The grader should not have to struggle to understand your design.