

SOC1 Board Howto

Feb 8, 2022

This document was written to go from knowing nothing about VHDL or anything to do with the SOC1 board that's located in ENG412 and programming it. This board is split into two sections:

1. ARM processor
2. Altera FPGA

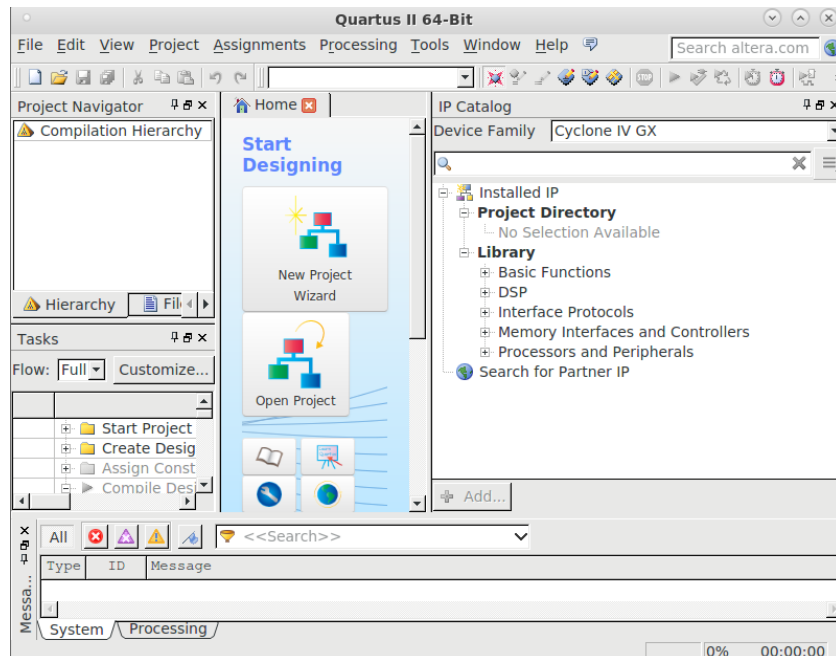
The COE838 course uses this board as an example of a system on chip installation. The idea is that you would connect various hardware components through the FPGA so that you could then program a C program that was compiled for the ARM processor and run within a Linux environment to work with the exposed FPGA hardware components.

Software used:

- Quartus 14.0 for programming the FPGA board
- DS 5 ARM eclipse suite to compile the C code to execute within linux for the ARM processor

To give credit where it's due, the original documentation that I followed was obtained from the COE838 course, specifically lab3. I'm following the lab3.pdf file that is installed in the same google drive folder that this document resides in.

The lab first starts with you firing up Quartus II 14.0 but I'm pretty sure newer versions should also work. We'll test with 14.0. Start Quartus from the Applications menu -> Engineering -> Quartus 14.0. The software is quite fat and may take about 30 seconds simply to start. You will see the following:



Now that you've started quartus.

Next on the home screen, select "New Project Wizard". This will bring up an introduction screen which you can simply click next. You will now need to provide a working directory to store your Quartus design. We're doing this for the COE838, lab3 so I put this in:

/home/faculty/jnaughto/coe838/lab3

These folders didn't exist so I had to create them of course. I also named the project that we were working on as per the lab manual "LED_HEX_FPGA". So the next screen should look like this:

Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

/home/faculty/jnaughto/coe838/lab3

What is the name of this project?

LED_HEX_FPGA

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

LED_HEX_FPGA

[Use Existing Project Settings...](#)

Click next at the bottom of the window. The next window allows you to add files. We'll do that once the project has been created so simply click next once more. Now you will be presented with the Family & Device page which you will need to find the specific Altera FPGA that we're working on. Based on the guide we're to select **5CSEMA5F31C6**. Narrow down the search by setting the Family to **Cyclone V** and the package to **FBGA**. Now in the name filter field you can type 5CSEMA5F31C6 and you'll only see one available device which we are using.

New Project Wizard

Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.
You can install additional device support with the Install Devices command on the Tools menu.

To determine the version of the Quartus II software in which your target device is supported, refer to the [Device Support List](#) webpage.

Device family

Family: Cyclone V (E/GX/GT/SX/SE/ST)

Devices: All

Target device

☐ Auto device selected by the Fitter

☒ Specific device selected in 'Available devices' list

☐ Other: n/a

Show in 'Available devices' list

Package: FBGA

Pin count: Any

Core Speed grade: Any

Name filter: 5CSEMA5F31C6

☒ Show advanced devices

Available devices:

Name	Core Voltage	ALMs	User I/Os	GXB Channel PMA	GXB Channel PCS	PCIe (PIPE) Har
5CSEMA5F31C6	1.1V	32070	457	0	0	0

Now that we've selected our device click next. The next screen I didn't have to touch anything. Essentially we need to make sure that the Tool Name is **ModelSim-Altera** in the Simulation row, and **VHDL** in the format column also on the simulation row. It should look like this:

EDA Tool Settings [page 4 of 5]

Specify the other EDA tools used with the Quartus II software to develop your project.

EDA tools:

Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/...	<None>	<None>	<input type="checkbox"/> Run this tool automatically to synthesize the current design
Simulation	ModelSim-Altera	VHDL	<input type="checkbox"/> Run gate-level simulation automatically after compilation
Formal Verific...	<None>		
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

< Back Next > Finish Cancel Help

You can click next and finally you will be presented with a Summary page:

Summary [page 5 of 5]

When you click Finish, the project will be created with the following settings:

Project directory: /home/faculty/jnaughto/coe838/lab3

Project name: LED_HEX_FPGA

Top-level design entity: LED_HEX_FPGA

Number of files added: 0

Number of user libraries added: 0

Device assignments:

Family name: Cyclone V (E/GX/GT/SX/SE/ST)

Device: 5CSEMA5F31C6

EDA tools:

Design entry/synthesis: <None> (<None>)

Simulation: ModelSim-Altera (VHDL)

Timing analysis: ()

Operating conditions:

Core voltage: 1.1V

Junction temperature range: 0-85 °C

< Back Next > Finish Cancel Help

Click on the Finish button and your quartus workspace has been created and should be ready to use. Next we need to add a few files to our project folder that reside in the /home/courses/coe838/labs/lab3 folder. I've copied these files also into the google drive folder that this document resides in.

My workspace folder for this lab is presently:

`${HOME}/coe838/lab3`

I need to copy all the files in the course directory `/home/courses/coe838/labs/lab3/rtl` into the project folder so I issued:

```
cp /home/courses/coe838/labs/lab3/rtl/* ${HOME}/coe838/lab3
```

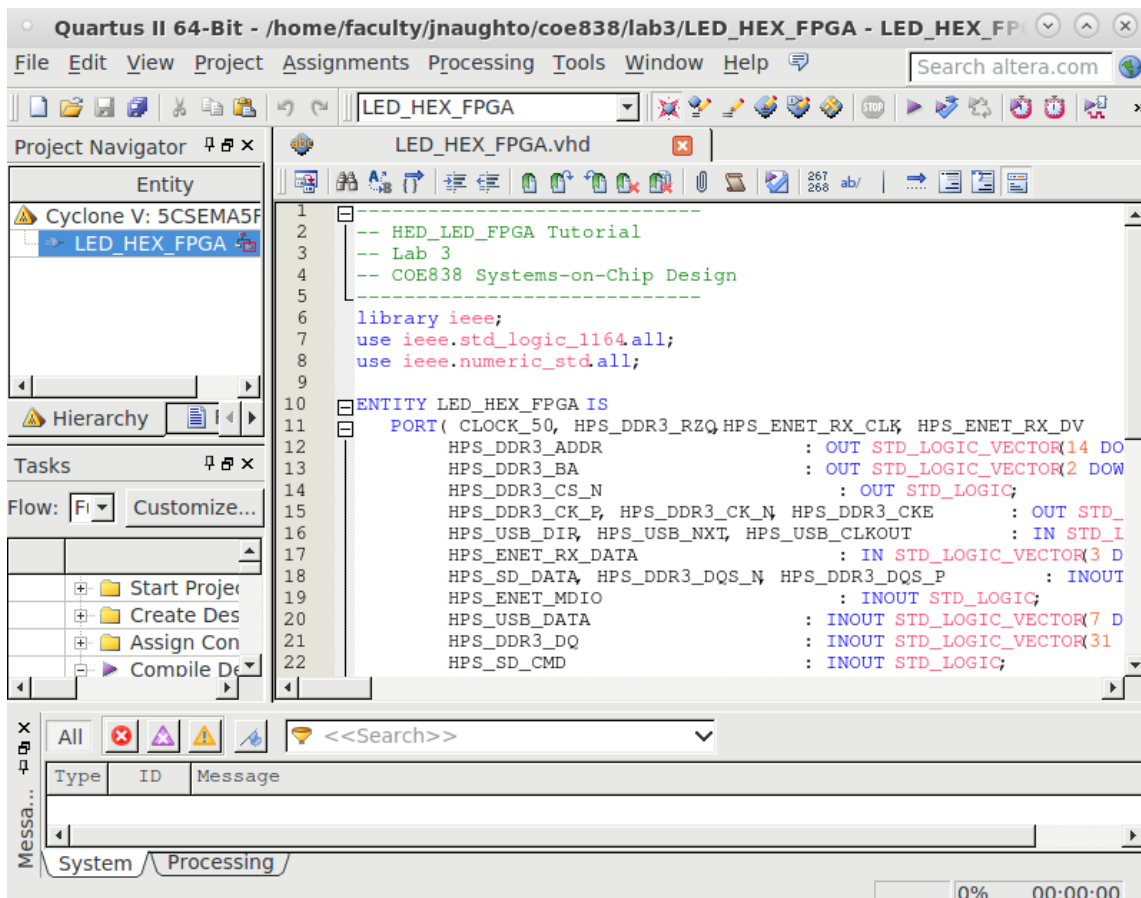
Next I also need to add the entire folder that resides in the course directory to my project folder so I issued:

```
cp -r /home/courses/coe838/labs/lab3/ip ${HOME}/coe838/lab3
```

So my project folder looks like this:

```
jnaughto@clarkson: ~/coe838/lab3$ ls
db/  LED_HEX_FPGA.qpf  LED_HEX_FPGA.vhd*      soc_system.qsys*
ip/  LED_HEX_FPGA.qsf   pin_assignment_DE1-SoC.tcl*
```

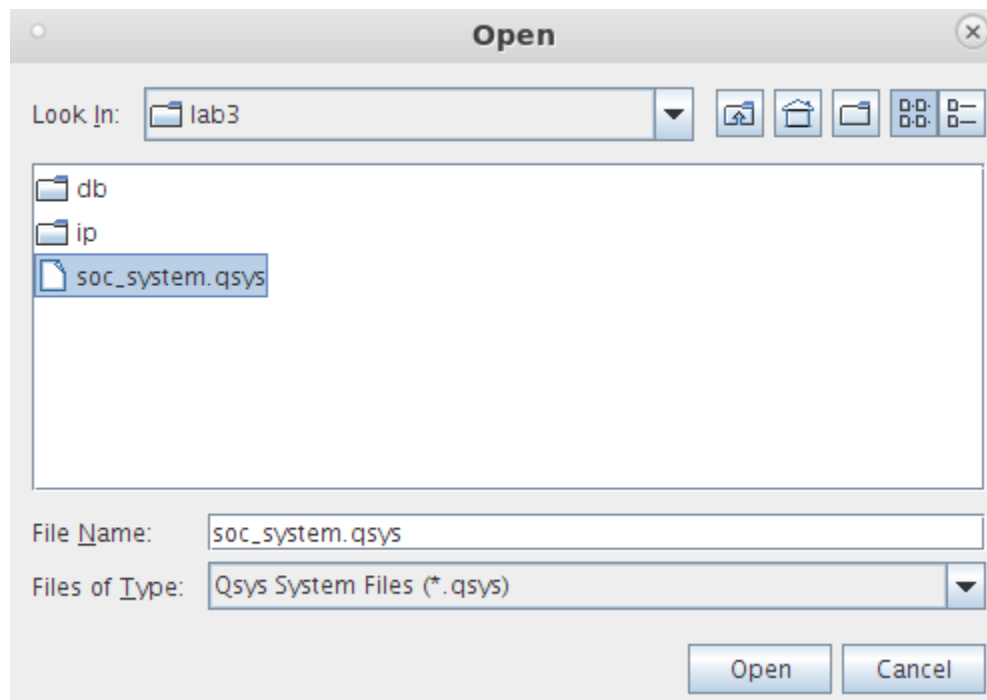
Go back to Quartus II. Select "Project" - "Add/Remove files in project". Select the "..." button and select the LED_HEX_FPGA.vhd file, Press OPEN, click ADD, click Apply and then click OK. Now if you double click on the LED_HEX_FPGA.vhd file in the Entity Column of Quartus you will see the VHDL source template that was given to you:



NOTE: If the source VHDL file does not appear, select "File" - "Open" - LED_HEX_FPGA.vhd. Select "Project" - "Set as Top-Level Entity". Now you should be able to double click on the source VHDL file.

Observe the VHDL code provided to you. The entity contains keywords used by the Cyclone V for its HPS/FPGA pinouts. These names can be found in the pin_assignment_DE1-SoC.tcl file provided to you. Notice that there are comments noting where you should place the SoC component and it's port map given by QSys.

Now we need to start using QSYS it will provide the connections that we need to make between the hardware components, the FPGA and of course the ARM processor. In Quartus, select "Tools" - "QSys". Wait for QSys to launch. A window will start up asking you to select a qsys file. Select the **soc_system.qsys** file that we copied into our work folder:



A QSys system will open containing a clock source and HPS IP block. Make sure to use this .qsys template for all your labs/projects as it contains HPS parameter mappings specifically for the DE1-SoC. We will now add the custom IP cores needed for our SoC and make the proper connections to the HPS/FPGA bridges.

In the QSys "IP Catalog" left window pane, expand "Project" - "Terasic Technologies Inc". Highlight "SEG7_IF" and select "+ Add..". A SEG7_IF parameters window will open. Ensure the following values are specified:

- SEG7_NUM: 6
- ADDR_WIDTH: 3
- DEFAULT_ACTIVE: 1
- LOW_ACTIVE: 1

Click finished in the SEG7_IF window once complete. There will be errors in the messages menu but don't worry these will be corrected after we add all our hardware that we wish to connect up to the FPGA.

Next, we will add a Parallel I/O (PIO) IP block to the system. We will need this to access the LEDs on the FPGA. Select "Processors and Peripherals" - "Peripherals" - "PIO (Parallel I/O)". Press "+Add...". A Parallel I/O parameters window will open. Ensure the following values are specified:

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

Note: the same PIO IP can be selected for accessing the Switches on the DE1-SoC. Click Finish.

In the QSys "System Contents" window, follow the "Name" column, and right click "pio_0". Select rename and change the default name to "led_pio". We can leave SEG7_IF_0 since this name bears meaning. If you chose not to name the IP blocks with the names specified, be conscious of this fact when designing and coding your VHDL in Section 3.3.

Use	Connections	Name	Description	Export
<input checked="" type="checkbox"/>		clk_0	Clock Source	
		clk_in	Clock Input	clk
		clk_in_reset	Reset Input	reset
		clk	Clock Output	Double-click
		clk_reset	Reset Output	Double-click
<input checked="" type="checkbox"/>		hps_0	Arria V/Cyclone V Hard Process...	
		memory	Conduit	memory
		hps_io	Conduit	hps_io
		h2f_reset	Reset Output	hps_0_h2f_re
		h2f_axi_clock	Clock Input	Double-click
		h2f_axi_master	AXI Master	Double-click
		f2h_axi_clock	Clock Input	Double-click
		f2h_axi_slave	AXI Slave	Double-click
		h2f_lw_axi_clock	Clock Input	Double-click
		h2f_lw_axi_master	AXI Master	Double-click
<input checked="" type="checkbox"/>		SEG7_IF_0	SEG7_IF	
		avalon_slave	Avalon Memory Mapped Slave	Double-click
		conduit_end	Conduit	Double-click
		clock_sink	Clock Input	Double-click
		clock_sink_reset	Reset Input	Double-click
<input checked="" type="checkbox"/>		led_pio	PIO (Parallel I/O)	
		clk	Clock Input	Double-click
		reset	Reset Input	Double-click
		s1	Avalon Memory Mapped Slave	Double-click
		external_connection	Conduit	Double-click

Next we need to make our component connections. Next we will use the "Connections" column in QSys to map the IP blocks to the HPS. Each empty circle signifies a connection that may be made between two or more components (ports) in your system. A black/filled circle signifies that a connection/signal has been established between the two components.

Follow clk_0's clk (clk_0.clk) connection down to the SEG7_IF_0 and led_pio components. Establish a connection (i.e. press the empty circle to color black) between clk_0.clk and:

- SEG7_IF_0's clock_sink port
- led_pio's clk port

Use	Connections	Name	Description	Export	Clock
<input checked="" type="checkbox"/>		<div>clk_0</div> <div>clk_in</div> <div>clk_in_reset</div> <div>clk</div> <div>clk_reset</div>	<div>Clock Source</div> <div>Clock Input</div> <div>Reset Input</div> <div>Clock Output</div> <div>Reset Output</div>	<div>clk</div> <div>reset</div> <div>Double-click to export</div> <div>Double-click to export</div>	<div>exported</div> <div>clk_0</div>
<input checked="" type="checkbox"/>		<div>hps_0</div> <div>memory</div> <div>hps_io</div> <div>h2f_reset</div> <div>h2f_axi_clock</div> <div>h2f_axi_master</div> <div>f2h_axi_clock</div> <div>f2h_axi_slave</div> <div>h2f_lw_axi_clock</div> <div>h2f_lw_axi_master</div>	<div>Arria V/Cyclone V Hard Process...</div> <div>Conduit</div> <div>Conduit</div> <div>Reset Output</div> <div>Clock Input</div> <div>AXI Master</div> <div>Clock Input</div> <div>AXI Slave</div> <div>Clock Input</div> <div>AXI Master</div>	<div>memory</div> <div>hps_io</div> <div>hps_0_h2f_reset</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div>	<div>clk_0</div> <div>[h2f_axi_c...</div> <div>clk_0</div> <div>[f2h_axi_c...</div> <div>clk_0</div> <div>[h2f_lw_a...</div>
<input checked="" type="checkbox"/>		<div>SEG7_IF_0</div> <div>avalon_slave</div> <div>conduit_end</div> <div>clock_sink</div> <div>clock_sink_reset</div>	<div>SEG7_IF</div> <div>Avalon Memory Mapped Slave</div> <div>Conduit</div> <div>Clock Input</div> <div>Reset Input</div>	<div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div>	<div>[clock_sink]</div> <div>clk_0</div> <div>[clock_sink]</div>
<input checked="" type="checkbox"/>		<div>led_pio</div> <div>clk</div> <div>reset</div> <div>s1</div> <div>external_connection</div>	<div>PIO (Parallel I/O)</div> <div>Clock Input</div> <div>Reset Input</div> <div>Avalon Memory Mapped Slave</div> <div>Conduit</div>	<div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div>	<div>clk_0</div> <div>[clk]</div> <div>[clk]</div>

Next Make a connection between clk_0.clk_reset and:

- SEG7_IF_0's clock_sink_reset port
- led_pio's reset port

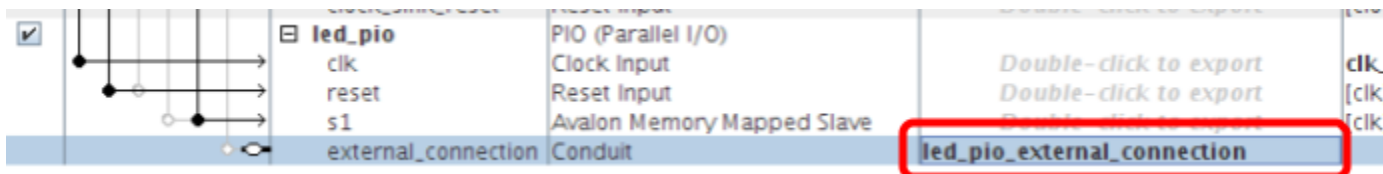
Use	Connections	Name	Description	Export	Clock	
<input checked="" type="checkbox"/>		<div>clk_0</div> <div>clk_in</div> <div>clk_in_reset</div> <div>clk</div> <div>clk_reset</div>	<div>Clock Source</div> <div>Clock Input</div> <div>Reset Input</div> <div>Clock Output</div> <div>Reset Output</div>	<div>clk</div> <div>reset</div> <div>Double-click to export</div> <div>Double-click to export</div>	<div>exported</div> <div>clk_0</div>	
<input checked="" type="checkbox"/>		<div>hps_0</div> <div>memory</div> <div>hps_io</div> <div>h2f_reset</div> <div>h2f_axi_clock</div> <div>h2f_axi_master</div> <div>f2h_axi_clock</div> <div>f2h_axi_slave</div> <div>h2f_lw_axi_clock</div> <div>h2f_lw_axi_master</div>	<div>Arria V/Cyclone V Hard Process...</div> <div>Conduit</div> <div>Conduit</div> <div>Reset Output</div> <div>Clock Input</div> <div>AXI Master</div> <div>Clock Input</div> <div>AXI Slave</div> <div>Clock Input</div> <div>AXI Master</div>	<div>memory</div> <div>hps_io</div> <div>hps_0_h2f_reset</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div>	<div>clk_0</div> <div>[h2f_axi_c...</div> <div>clk_0</div> <div>[f2h_axi_c...</div> <div>clk_0</div> <div>[h2f_lw_a...</div>	
<input checked="" type="checkbox"/>		<div>SEG7_IF_0</div> <div>avalon_slave</div> <div>conduit_end</div> <div>clock_sink</div> <div>clock_sink_reset</div>	<div>SEG7_IF</div> <div>Avalon Memory Mapped Slave</div> <div>Conduit</div> <div>Clock Input</div> <div>Reset Input</div>	<div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div>	<div>[clock_sink]</div> <div>clk_0</div> <div>[clock_sink]</div>	
<input checked="" type="checkbox"/>		<div>led_pio</div> <div>clk</div> <div>reset</div> <div>s1</div> <div>external_connection</div>	<div>PIO (Parallel I/O)</div> <div>Clock Input</div> <div>Reset Input</div> <div>Avalon Memory Mapped Slave</div> <div>Conduit</div>	<div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div>	<div>clk_0</div> <div>[clk]</div> <div>[clk]</div>	

Next Follow hps_0.h2f_lw_axi_master and make a (slave) connection between:

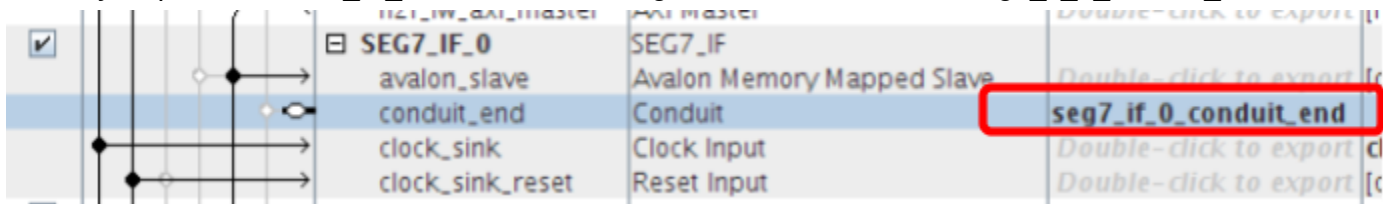
- SEG7_IF_0's avalon_slave port
- led_pio's s1 port

System Contents		Address Map		Interconnect Requirements		Device Family			
Use	Connections	Name	Description	Export	Clock	Bar			
<input checked="" type="checkbox"/>		<div>clk_0</div> <div>clk_in</div> <div>clk_in_reset</div> <div>clk</div> <div>clk_reset</div>	<div>Clock Source</div> <div>Clock Input</div> <div>Reset Input</div> <div>Clock Output</div> <div>Reset Output</div>	<div>clk</div> <div>reset</div> <div>Double-click to export</div> <div>Double-click to export</div>	<div>exported</div> <div>clk_0</div>				
<input checked="" type="checkbox"/>		<div>hps_0</div> <div>memory</div> <div>hps_io</div> <div>h2f_reset</div> <div>h2f_axi_clock</div> <div>h2f_axi_master</div> <div>f2h_axi_clock</div> <div>f2h_axi_slave</div> <div>h2f_lw_axi_clock</div> <div>h2f_lw_axi_master</div>	<div>Arria V/Cyclone V Hard Process...</div> <div>Conduit</div> <div>Conduit</div> <div>Reset Output</div> <div>Clock Input</div> <div>AXI Master</div> <div>Clock Input</div> <div>AXI Slave</div> <div>Clock Input</div> <div>AXI Master</div>	<div>memory</div> <div>hps_io</div> <div>hps_0_h2f_reset</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div>	<div>clk_0</div> <div>[h2f_axi_c...</div> <div>clk_0</div> <div>[f2h_axi_c...</div> <div>clk_0</div> <div>[h2f_lw_a...</div>				
<input checked="" type="checkbox"/>		<div>SEG7_IF_0</div> <div>avalon_slave</div> <div>conduit_end</div> <div>clock_sink</div> <div>clock_sink_reset</div>	<div>SEG7_IF</div> <div>Avalon Memory Mapped Slave</div> <div>Conduit</div> <div>Clock Input</div> <div>Reset Input</div>	<div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div>	<div>[clock_sink]</div> <div>clk_0</div> <div>[clock_sink]</div>	<div>0x0000_00</div>			
<input checked="" type="checkbox"/>		<div>led_pio</div> <div>clk</div> <div>reset</div> <div>s1</div> <div>external_connection</div>	<div>PIO (Parallel I/O)</div> <div>Clock Input</div> <div>Reset Input</div> <div>Avalon Memory Mapped Slave</div> <div>Conduit</div>	<div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div> <div>Double-click to export</div>	<div>clk_0</div> <div>[clk]</div> <div>[clk]</div>	<div>0x0000_00</div>			

Follow led_pio's external connection port horizontally to its "export" field. There is a faded label which says "Double-click to export". Double click the field. A default name will appear called "led_pio_external_connection". This process is referred to as exporting a conduit.



Similarly, export the SEG7_IF_0 conduit. It will be given the default name "seg7_if_0_conduit_end".



The base addresses present on the component conduits may overlap and generate an error in the Messages window. In QSys, select "System" - "Assign Base Address". There should also be no warnings or errors in the Messages window.

Use	Connections	Name	Description	Export	Clock
<input checked="" type="checkbox"/>		clk_0 clk_in clk_in_reset clk clk_reset	Clock Source Clock Input Reset Input Clock Output Reset Output	clk reset Double-click to export Double-click to export	exported clk_0
<input checked="" type="checkbox"/>		hps_0 memory hps_io h2f_reset h2f_axi_clock h2f_axi_master f2h_axi_clock f2h_axi_slave h2f_lw_axi_clock h2f_lw_axi_master	Arria V/Cyclone V Hard Process... Conduit Conduit Reset Output Clock Input AXI Master Clock Input AXI Slave Clock Input AXI Master	memory hps_io hps_0_h2f_reset Double-click to export Double-click to export Double-click to export Double-click to export Double-click to export	clk_0 [h2f_axi_c... clk_0 [f2h_axi_c... clk_0 [h2f_lw_a...
<input checked="" type="checkbox"/>		SEG7_IF_0 avalon_slave conduit_end clock_sink clock_sink_reset	SEG7_IF Avalon Memory Mapped Slave Conduit Clock Input Reset Input	Double-click to export seg7_if_0_conduit_end Double-click to export Double-click to export	[clock_sink] clk_0 [clock_sink]
<input checked="" type="checkbox"/>		led_pio clk reset s1 external_connection	PIO (Parallel I/O) Clock Input Reset Input Avalon Memory Mapped Slave Conduit	Double-click to export Double-click to export Double-click to export led_pio_external_connection	clk_0 [clk] [clk]

Exporting Conduits are used to create external connections so that other components may access and communicate with your IP. In this case, we export led_pio and SEG7 so that the HPS system may communicate with these components. Similarly, the slave connections you make in QSys allow the IPs to communicate with the FPGA fabric and HPS (using the lwh2f bridge/AXI bus and L3 interconnect). The slaves also possess base addresses that you may include in your memory mapped I/O software to control and communicated data to the IPs.

Generating the QSys System

In QSys, go to "Generate" - "HDL Example". Under "HDL Language" select VHDL. Press 'Copy' to copy the template and paste the VHDL code to your top-level entity (in the ARCHITECTURE section). We will use this code as a basis for instantiating and port mapping the HPS component into your SoC design. Press 'Close' when you finish copying the code.

Now there are two parts of VHDL code that will be generated. The part for the ARCHITECTURE section and then of course the section for the physical port mapping. The ARCHITECTURE section will look like this:

```
component soc_system is
  port (
    clk_clk                : in    std_logic                := 'X';           -- clk
    hps_0_h2f_reset_reset_n : out   std_logic;              -- reset_n
    hps_io_hps_io_emac1_inst_TX_CLK : out std_logic;        -- hps_io_emac1_inst_TX_CLK
    hps_io_hps_io_emac1_inst_TXD0 : out std_logic;          -- hps_io_emac1_inst_TXD0
    hps_io_hps_io_emac1_inst_TXD1 : out std_logic;          -- hps_io_emac1_inst_TXD1
    hps_io_hps_io_emac1_inst_TXD2 : out std_logic;          -- hps_io_emac1_inst_TXD2
    hps_io_hps_io_emac1_inst_TXD3 : out std_logic;          -- hps_io_emac1_inst_TXD3
    hps_io_hps_io_emac1_inst_RXD0 : in    std_logic          := 'X';           -- hps_io_emac1_inst_RXD0
    hps_io_hps_io_emac1_inst_RXD1 : inout std_logic          := 'X';           -- hps_io_emac1_inst_RXD1
    hps_io_hps_io_emac1_inst_MDC : out   std_logic;          -- hps_io_emac1_inst_MDC
    hps_io_hps_io_emac1_inst_RX_CTL : in    std_logic          := 'X';           -- hps_io_emac1_inst_RX_CTL
    hps_io_hps_io_emac1_inst_TX_CTL : out   std_logic;        -- hps_io_emac1_inst_TX_CTL
    hps_io_hps_io_emac1_inst_RX_CLK : in    std_logic          := 'X';           -- hps_io_emac1_inst_RX_CLK
    hps_io_hps_io_emac1_inst_RXD1 : in    std_logic          := 'X';           -- hps_io_emac1_inst_RXD1
    hps_io_hps_io_emac1_inst_RXD2 : in    std_logic          := 'X';           -- hps_io_emac1_inst_RXD2
    hps_io_hps_io_emac1_inst_RXD3 : in    std_logic          := 'X';           -- hps_io_emac1_inst_RXD3
    hps_io_hps_io_emac1_inst_CMD : inout std_logic          := 'X';           -- hps_io_emac1_inst_CMD
    hps_io_hps_io_sdio_inst_D0 : inout std_logic             := 'X';           -- hps_io_sdio_inst_D0
    hps_io_hps_io_sdio_inst_D1 : inout std_logic             := 'X';           -- hps_io_sdio_inst_D1
    hps_io_hps_io_sdio_inst_CLK : out   std_logic;           -- hps_io_sdio_inst_CLK
    hps_io_hps_io_sdio_inst_D2 : inout std_logic             := 'X';           -- hps_io_sdio_inst_D2
    hps_io_hps_io_sdio_inst_D3 : inout std_logic             := 'X';           -- hps_io_sdio_inst_D3
    hps_io_hps_io_usb1_inst_D0 : inout std_logic             := 'X';           -- hps_io_usb1_inst_D0
    hps_io_hps_io_usb1_inst_D1 : inout std_logic             := 'X';           -- hps_io_usb1_inst_D1
    hps_io_hps_io_usb1_inst_D2 : inout std_logic             := 'X';           -- hps_io_usb1_inst_D2
    hps_io_hps_io_usb1_inst_D3 : inout std_logic             := 'X';           -- hps_io_usb1_inst_D3
    hps_io_hps_io_usb1_inst_D4 : inout std_logic             := 'X';           -- hps_io_usb1_inst_D4
    hps_io_hps_io_usb1_inst_D5 : inout std_logic             := 'X';           -- hps_io_usb1_inst_D5
    hps_io_hps_io_usb1_inst_D6 : inout std_logic             := 'X';           -- hps_io_usb1_inst_D6
    hps_io_hps_io_usb1_inst_D7 : inout std_logic             := 'X';           -- hps_io_usb1_inst_D7
    hps_io_hps_io_usb1_inst_CLK : in    std_logic             := 'X';           -- hps_io_usb1_inst_CLK
    hps_io_hps_io_usb1_inst_STP : out   std_logic;           -- hps_io_usb1_inst_STP
    hps_io_hps_io_usb1_inst_DIR : in    std_logic             := 'X';           -- hps_io_usb1_inst_DIR
    hps_io_hps_io_usb1_inst_NXT : in    std_logic             := 'X';           -- hps_io_usb1_inst_NXT
    memory_mem_a            : out   std_logic_vector(14 downto 0); -- mem_a
    memory_mem_ba           : out   std_logic_vector(2 downto 0); -- mem_ba
    memory_mem_ck           : out   std_logic;                  -- mem_ck
    memory_mem_ck_n         : out   std_logic;                  -- mem_ck_n
    memory_mem_cke          : out   std_logic;                  -- mem_cke
    memory_mem_cs_n         : out   std_logic;                  -- mem_cs_n
    memory_mem_ras_n        : out   std_logic;                  -- mem_ras_n
    memory_mem_cas_n        : out   std_logic;                  -- mem_cas_n
    memory_mem_we_n         : out   std_logic;                  -- mem_we_n
    memory_mem_reset_n      : out   std_logic;                  -- mem_reset_n
    memory_mem_dq           : inout  std_logic_vector(31 downto 0) := (others => 'X'); -- mem_dq
    memory_mem_dqs          : inout  std_logic_vector(3 downto 0) := (others => 'X'); -- mem_dqs
    memory_mem_dqs_n        : inout  std_logic_vector(3 downto 0) := (others => 'X'); -- mem_dqs_n
    memory_mem_odt          : out   std_logic;                  -- mem_odt
    memory_mem_dm           : out   std_logic_vector(3 downto 0); -- mem_dm
    memory_oct_rzqin        : in    std_logic                   := 'X';           -- oct_rzqin
    reset_reset_n          : in    std_logic                   := 'X';           -- reset_n
    led_pio_external_connection_export : out std_logic_vector(9 downto 0); -- export
    seg7_if_0_conduit_end_export : out std_logic_vector(47 downto 0); -- export
  );
```

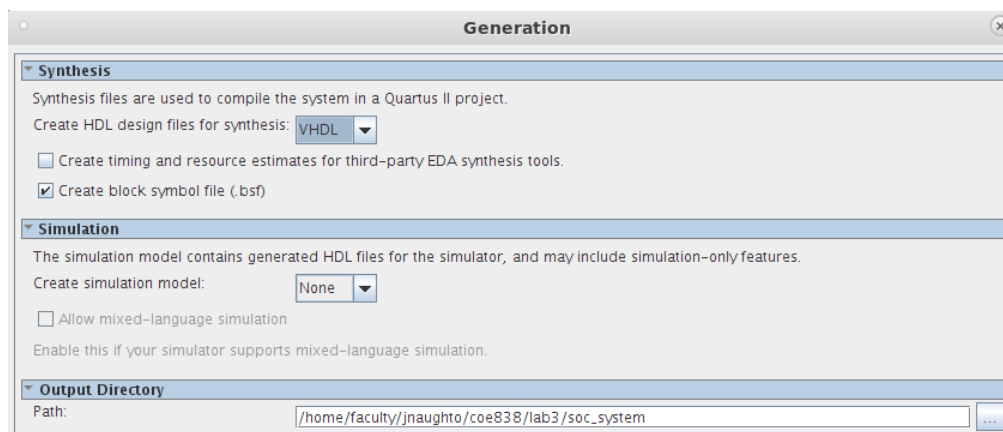
This needs to be transferred to the architecture section of your LED_HEX_FPGA section.

Then the port mapping section will appear as follows:

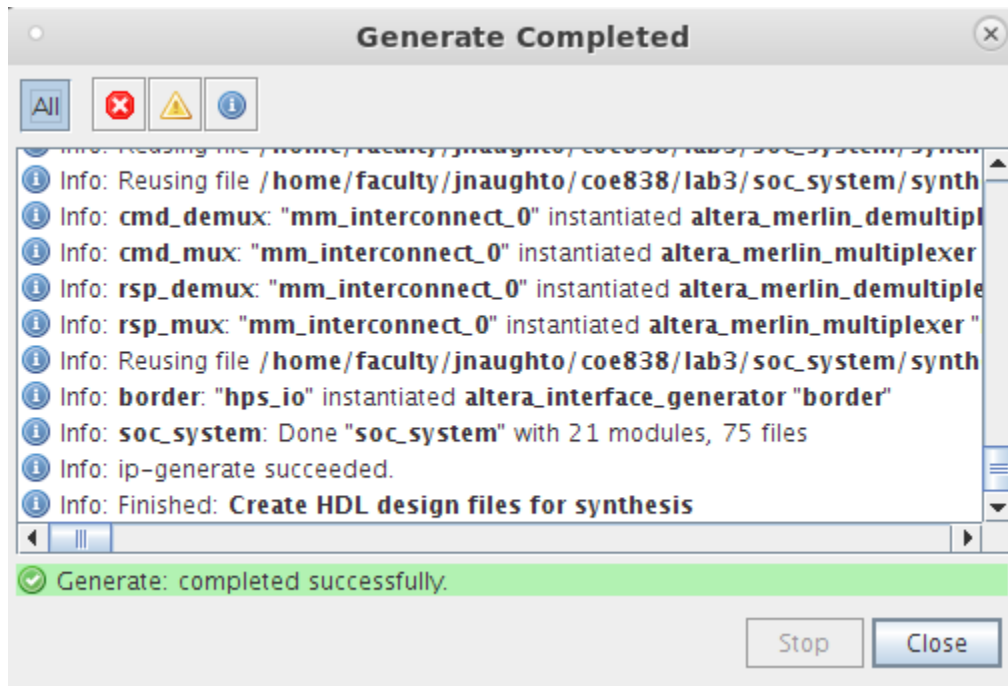
```
u0 : component soc_system
port map (
    clk_clk => CONNECTED_T0_clk_clk, -- clk.clk
    hps_0_h2f_reset_reset_n => CONNECTED_T0_hps_0_h2f_reset_reset_n, -- hps_0_h2f_reset.reset_n
    hps_io_hps_io_emaci_inst_TX_CLK => CONNECTED_T0_hps_io_hps_io_emaci_inst_TX_CLK, -- hps_io.hps_io_emaci_inst_TX_CLK
    hps_io_hps_io_emaci_inst_TXD0 => CONNECTED_T0_hps_io_hps_io_emaci_inst_TXD0, -- .hps_io_emaci_inst_TXD0
    hps_io_hps_io_emaci_inst_TXD1 => CONNECTED_T0_hps_io_hps_io_emaci_inst_TXD1, -- .hps_io_emaci_inst_TXD1
    hps_io_hps_io_emaci_inst_TXD2 => CONNECTED_T0_hps_io_hps_io_emaci_inst_TXD2, -- .hps_io_emaci_inst_TXD2
    hps_io_hps_io_emaci_inst_TXD3 => CONNECTED_T0_hps_io_hps_io_emaci_inst_TXD3, -- .hps_io_emaci_inst_TXD3
    hps_io_hps_io_emaci_inst_RXD0 => CONNECTED_T0_hps_io_hps_io_emaci_inst_RXD0, -- .hps_io_emaci_inst_RXD0
    hps_io_hps_io_emaci_inst_MDI0 => CONNECTED_T0_hps_io_hps_io_emaci_inst_MDI0, -- .hps_io_emaci_inst_MDI0
    hps_io_hps_io_emaci_inst_MDC => CONNECTED_T0_hps_io_hps_io_emaci_inst_MDC, -- .hps_io_emaci_inst_MDC
    hps_io_hps_io_emaci_inst_RX_CTL => CONNECTED_T0_hps_io_hps_io_emaci_inst_RX_CTL, -- .hps_io_emaci_inst_RX_CTL
    hps_io_hps_io_emaci_inst_TX_CTL => CONNECTED_T0_hps_io_hps_io_emaci_inst_TX_CTL, -- .hps_io_emaci_inst_TX_CTL
    hps_io_hps_io_emaci_inst_RX_CLK => CONNECTED_T0_hps_io_hps_io_emaci_inst_RX_CLK, -- .hps_io_emaci_inst_RX_CLK
    hps_io_hps_io_emaci_inst_RXD1 => CONNECTED_T0_hps_io_hps_io_emaci_inst_RXD1, -- .hps_io_emaci_inst_RXD1
    hps_io_hps_io_emaci_inst_RXD2 => CONNECTED_T0_hps_io_hps_io_emaci_inst_RXD2, -- .hps_io_emaci_inst_RXD2
    hps_io_hps_io_emaci_inst_RXD3 => CONNECTED_T0_hps_io_hps_io_emaci_inst_RXD3, -- .hps_io_emaci_inst_RXD3
    hps_io_hps_io_sdio_inst_CMD => CONNECTED_T0_hps_io_hps_io_sdio_inst_CMD, -- .hps_io_sdio_inst_CMD
    hps_io_hps_io_sdio_inst_D0 => CONNECTED_T0_hps_io_hps_io_sdio_inst_D0, -- .hps_io_sdio_inst_D0
    hps_io_hps_io_sdio_inst_D1 => CONNECTED_T0_hps_io_hps_io_sdio_inst_D1, -- .hps_io_sdio_inst_D1
    hps_io_hps_io_sdio_inst_CLK => CONNECTED_T0_hps_io_hps_io_sdio_inst_CLK, -- .hps_io_sdio_inst_CLK
    hps_io_hps_io_sdio_inst_D2 => CONNECTED_T0_hps_io_hps_io_sdio_inst_D2, -- .hps_io_sdio_inst_D2
    hps_io_hps_io_sdio_inst_D3 => CONNECTED_T0_hps_io_hps_io_sdio_inst_D3, -- .hps_io_sdio_inst_D3
    hps_io_hps_io_usb1_inst_D0 => CONNECTED_T0_hps_io_hps_io_usb1_inst_D0, -- .hps_io_usb1_inst_D0
    hps_io_hps_io_usb1_inst_D1 => CONNECTED_T0_hps_io_hps_io_usb1_inst_D1, -- .hps_io_usb1_inst_D1
    hps_io_hps_io_usb1_inst_D2 => CONNECTED_T0_hps_io_hps_io_usb1_inst_D2, -- .hps_io_usb1_inst_D2
    hps_io_hps_io_usb1_inst_D3 => CONNECTED_T0_hps_io_hps_io_usb1_inst_D3, -- .hps_io_usb1_inst_D3
    hps_io_hps_io_usb1_inst_D4 => CONNECTED_T0_hps_io_hps_io_usb1_inst_D4, -- .hps_io_usb1_inst_D4
    hps_io_hps_io_usb1_inst_D5 => CONNECTED_T0_hps_io_hps_io_usb1_inst_D5, -- .hps_io_usb1_inst_D5
    hps_io_hps_io_usb1_inst_D6 => CONNECTED_T0_hps_io_hps_io_usb1_inst_D6, -- .hps_io_usb1_inst_D6
    hps_io_hps_io_usb1_inst_D7 => CONNECTED_T0_hps_io_hps_io_usb1_inst_D7, -- .hps_io_usb1_inst_D7
    hps_io_hps_io_usb1_inst_CLK => CONNECTED_T0_hps_io_hps_io_usb1_inst_CLK, -- .hps_io_usb1_inst_CLK
    hps_io_hps_io_usb1_inst_STP => CONNECTED_T0_hps_io_hps_io_usb1_inst_STP, -- .hps_io_usb1_inst_STP
    hps_io_hps_io_usb1_inst_DIR => CONNECTED_T0_hps_io_hps_io_usb1_inst_DIR, -- .hps_io_usb1_inst_DIR
    hps_io_hps_io_usb1_inst_NXT => CONNECTED_T0_hps_io_hps_io_usb1_inst_NXT, -- .hps_io_usb1_inst_NXT
    memory_mem_a => CONNECTED_T0_memory_mem_a, -- memory.mem_a
    memory_mem_ba => CONNECTED_T0_memory_mem_ba, -- .mem_ba
    memory_mem_ck => CONNECTED_T0_memory_mem_ck, -- .mem_ck
    memory_mem_ck_n => CONNECTED_T0_memory_mem_ck_n, -- .mem_ck_n
    memory_mem_cke => CONNECTED_T0_memory_mem_cke, -- .mem_cke
    memory_mem_cs_n => CONNECTED_T0_memory_mem_cs_n, -- .mem_cs_n
    memory_mem_ras_n => CONNECTED_T0_memory_mem_ras_n, -- .mem_ras_n
    memory_mem_cas_n => CONNECTED_T0_memory_mem_cas_n, -- .mem_cas_n
    memory_mem_we_n => CONNECTED_T0_memory_mem_we_n, -- .mem_we_n
    memory_mem_reset_n => CONNECTED_T0_memory_mem_reset_n, -- .mem_reset_n
    memory_mem_dq => CONNECTED_T0_memory_mem_dq, -- .mem_dq
    memory_mem_dqs => CONNECTED_T0_memory_mem_dqs, -- .mem_dqs
    memory_mem_dqs_n => CONNECTED_T0_memory_mem_dqs_n, -- .mem_dqs_n
    memory_mem_odt => CONNECTED_T0_memory_mem_odt, -- .mem_odt
    memory_mem_dm => CONNECTED_T0_memory_mem_dm, -- .mem_dm
    memory_oct_rzqin => CONNECTED_T0_memory_oct_rzqin, -- .oct_rzqin
    reset_reset_n => CONNECTED_T0_reset_reset_n, -- reset.reset_n
    led_pio_external_connection_export => CONNECTED_T0_led_pio_external_connection_export, -- led_pio_external_connection.export
    seg7_if_0_conduit_end_export => CONNECTED_T0_seg7_if_0_conduit_end_export, -- seg7_if_0_conduit_end.export
);
```

You do not need to add this to the port map block.

Next in QSys, select "Generate" - "Generate HDL...". The Generation window will open. Under Synthesis "Create HDL design files for synthesis:" select VHDL. Leave Create block symbol file (.bsf) enabled. The directory in the field "Output Directory" should match your project directory, with an appended folder name "soc_system" which will be created.



Click "Generate" at the bottom. This will save the QSys system and generate your custom SoC VHDL block called "soc_system". Wait for the system to generate and select "Close" to close the generation window once complete. This step will take some time to complete [30 -> 60 seconds]



At this point, QSys has generated: 1) the soc_system VHDL files needed for your SoC design, 2) a .tcl script for HPS pin assignments and 3) a .qpf file that needs to be included in the Quartus project for successfully synthesizing this HPS/FPGA system.

Next, we will generate the header files required for our HPS C software design. These .h files provide the names and base addresses of our QSys components. Therefore using these .h files we can simply #include the hps_0.h file in our code and virtually access these addresses through memory mapping.

In QSys, select Tools - NIOS II Command Shell [gcc4]. This will open a terminal. cd to your project directory. Once in your project directory, type **sopc-create-header-files**. A message will appear in the terminal stating that the header files have been generated.

```
-----
Altera Nios2 Command Shell [GCC 4]

Version 14.0, Build 209
-----

bash-4.2$ pwd
/home/faculty/jnaughto/coe838/lab3
bash-4.2$ socp-create-header-files
socp-create-header-files: Using SOPC design file ./soc_system.sopcinfo found in .
swinfo2header: Creating macro file 'soc_system.h' for SOPC Builder system 'soc_system'
swinfo2header: Creating macro file 'hps_0.h' for module 'hps_0'
swinfo2header: Creating macro file 'hps_0_bridges.h' for module 'hps_0_bridges'
swinfo2header: Creating macro file 'hps_0_arm_a9_0.h' for module 'hps_0_arm_a9_0'
swinfo2header: Creating macro file 'hps_0_arm_a9_1.h' for module 'hps_0_arm_a9_1'
```

In the NIOS terminal, create a folder called "software" with the command `mkdir software`. Move the generated header files to the software folder using the command `mv *.h software`

```
bash-4.2$ ls
LED_HEX_FPGA.qpf  hps_0_arm_a9_0.h      soc_system
LED_HEX_FPGA.qsf  hps_0_arm_a9_1.h      soc_system.h
LED_HEX_FPGA.vhd  hps_0_bridges.h       soc_system.qsys
db                ip                    soc_system.sopcinfo
hps_0.h           pin_assignment_DE1-SoC.tcl
bash-4.2$ mkdir software
bash-4.2$ mv *.h software
```

Go back to the NIOS terminal and close it. In QSys select "File" - "Save". Once saved, select "Finish" at the bottom to return back to Quartus II.

This point was the point I always had issues with regarding the Lab3 documentation. It appears that the statements left me pondering

1. Using the HDL code copied from QSys, instantiate the `soc_system` component in the ARCHITECTURE section.
2. Use the `u0: soc_system` template obtained from QSys to port map the `soc_system` ENTITY to your overall design. An example of the final VHDL can be found in the Appendix of this lab.

These two items are addressed when we manually did all the mapping prior. You will notice that the original lab refers the student to the appendix for the port mapping and signal lines. Now I reached out to Anita Tino who wrote Lab 3 and it appears that the above block needs to be manually mapped:

Jason Question: The problem is that when we run the Qsys HDL example it gives “`CONNECTED_TO_clk_clk`” instead of “`CLOCK_50`”. Is there a process in quartus that performs the swap and adds the signal line?

Anita Answer: That's correct. You have to manually map the FPGA pins to the ports, hence the Appendix for the students to guide them (basically a solution). Qsys and Quartus will not do this for you magically since 1) there's many different pin mappings the designer could apply to their hardware, and 2) AI hasn't reached FPGA synthesis yet (actually Intel started this initiative late 2021).

Anitia had already done the manual mapping and it's referenced in the appendix. Looking at the appendix there are two areas that need to be copied over SIGNAL and the Port Mapping section. The signal section is simply:

```
SIGNAL hex5_tmp, hex4_tmp, hex3_tmp, hex2_tmp, hex1_tmp, hex0_tmp, hps_0_h2f_reset_reset_n :
STD_LOGIC;
```

The port mapping section that needs to be copied is:

```
u0 : component soc_system
port map (
    clk_clk => CLOCK_50,
    reset_reset_n => '1',
```

```
memory_mem_a => HPS_DDR3_ADDR,
memory_mem_ba => HPS_DDR3_BA,
memory_mem_ck => HPS_DDR3_CK_P,
memory_mem_ck_n => HPS_DDR3_CK_N,
memory_mem_cke => HPS_DDR3_CKE,
memory_mem_cs_n => HPS_DDR3_CS_N,
memory_mem_ras_n => HPS_DDR3_RAS_N,
memory_mem_cas_n => HPS_DDR3_CAS_N,
memory_mem_we_n => HPS_DDR3_WE_N,
memory_mem_reset_n => HPS_DDR3_RESET_N,
memory_mem_dq => HPS_DDR3_DQ,
memory_mem_dqs => HPS_DDR3_DQS_P,
memory_mem_dqs_n => HPS_DDR3_DQS_N,
memory_mem_odt => HPS_DDR3_ODT,
memory_mem_dm => HPS_DDR3_DM,
memory_oct_rzqin => HPS_DDR3_RZQ,
hps_io_hps_io_emac1_inst_TX_CLK => HPS_ENET_GTX_CLK,
hps_io_hps_io_emac1_inst_TXD0 => HPS_ENET_TX_DATA(0),
hps_io_hps_io_emac1_inst_TXD1 => HPS_ENET_TX_DATA(1),
hps_io_hps_io_emac1_inst_TXD2 => HPS_ENET_TX_DATA(2),
hps_io_hps_io_emac1_inst_TXD3 => HPS_ENET_TX_DATA(3),
hps_io_hps_io_emac1_inst_RXD0 => HPS_ENET_RX_DATA(0),
hps_io_hps_io_emac1_inst_MDIO => HPS_ENET_MDIO,
hps_io_hps_io_emac1_inst_MDC => HPS_ENET_MDC,
hps_io_hps_io_emac1_inst_RX_CTL => HPS_ENET_RX_DV,
hps_io_hps_io_emac1_inst_TX_CTL => HPS_ENET_TX_EN,
hps_io_hps_io_emac1_inst_RX_CLK => HPS_ENET_RX_CLK,
hps_io_hps_io_emac1_inst_RXD1 => HPS_ENET_RX_DATA(1),
hps_io_hps_io_emac1_inst_RXD2 => HPS_ENET_RX_DATA(2),
hps_io_hps_io_emac1_inst_RXD3 => HPS_ENET_RX_DATA(3),
hps_io_hps_io_sdio_inst_CMD => HPS_SD_CMD,
hps_io_hps_io_sdio_inst_D0 => HPS_SD_DATA(0),
hps_io_hps_io_sdio_inst_D1 => HPS_SD_DATA(1),
hps_io_hps_io_sdio_inst_CLK => HPS_SD_CLK,
hps_io_hps_io_sdio_inst_D2 => HPS_SD_DATA(2),
hps_io_hps_io_sdio_inst_D3 => HPS_SD_DATA(3),
hps_io_hps_io_usb1_inst_D0 => HPS_USB_DATA(0),
hps_io_hps_io_usb1_inst_D1 => HPS_USB_DATA(1),
hps_io_hps_io_usb1_inst_D2 => HPS_USB_DATA(2),
hps_io_hps_io_usb1_inst_D3 => HPS_USB_DATA(3),
hps_io_hps_io_usb1_inst_D4 => HPS_USB_DATA(4),
hps_io_hps_io_usb1_inst_D5 => HPS_USB_DATA(5),
hps_io_hps_io_usb1_inst_D6 => HPS_USB_DATA(6),
hps_io_hps_io_usb1_inst_D7 => HPS_USB_DATA(7),
hps_io_hps_io_usb1_inst_CLK => HPS_USB_CLKOUT,
hps_io_hps_io_usb1_inst_STP => HPS_USB_STP,
hps_io_hps_io_usb1_inst_DIR => HPS_USB_DIR,
```



```

hps_io_hps_io_usb1_inst_NXT => HPS_USB_NXT,
hps_0_h2f_reset_reset_n => hps_0_h2f_reset_reset_n,
led_pio_external1_connection_export => LEDR,
seg7_if_0_conduit_end_export(47) => hex5_tmp,
seg7_if_0_conduit_end_export(46 DOWNT0 40)=>HEX5,
seg7_if_0_conduit_end_export(39) =>hex4_tmp,
seg7_if_0_conduit_end_export(38 DOWNT0 32)=>HEX4,
seg7_if_0_conduit_end_export(31) =>hex3_tmp,
seg7_if_0_conduit_end_export(30 DOWNT0 24)=>HEX3,
seg7_if_0_conduit_end_export(23) =>hex2_tmp,
seg7_if_0_conduit_end_export(22 DOWNT0 16)=> HEX2,
seg7_if_0_conduit_end_export(15) => hex1_tmp,
seg7_if_0_conduit_end_export(14 DOWNT0 8) => HEX1,
seg7_if_0_conduit_end_export(7) => hex0_tmp,
seg7_if_0_conduit_end_export(6 DOWNT0 0) =>HEX0
);

```

Copy both of these into the LED_HEX_FPGA VHDL code that is in quartus now. Specifically placing the port mapping section below the BEGIN LINE and of course the signal line before the BEGIN Line.

Next, we need to include the SoC system generated by QSys in our project: In Quartus select "Project" - "Add/Remove Files in Project". Navigate to the folder **soc_system/synthesis/** and select **soc_system.qip**. Press Open. Select "Add" - "Apply" - "OK".

Pin Assignment

In Quartus II, select "Tools" - "Tcl Scripts...". There is a note in the lab manual that states:

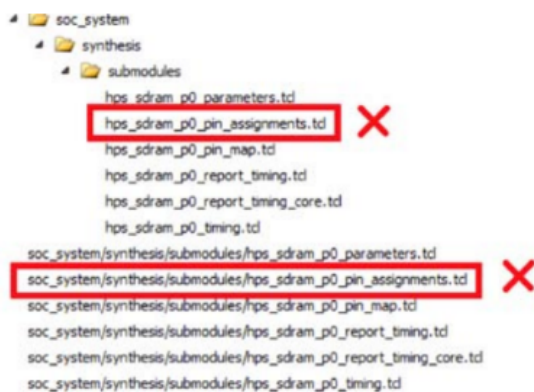


Fig. 7(a): Incorrect HPS Tcl Script Selection

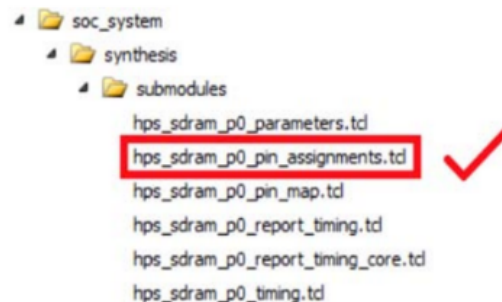
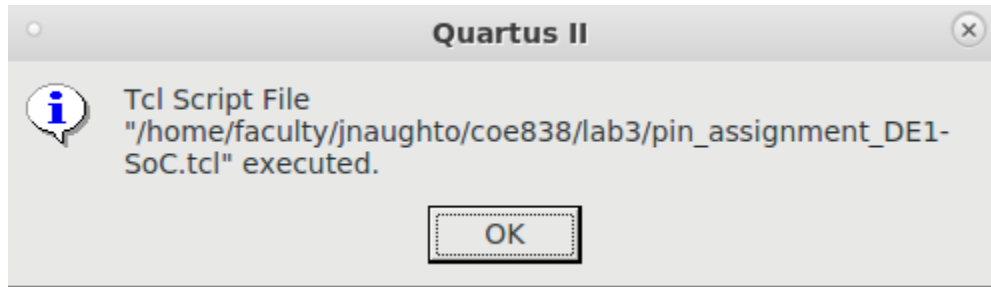


Fig. 7(b): Correct HPS Tcl Script Selection¹

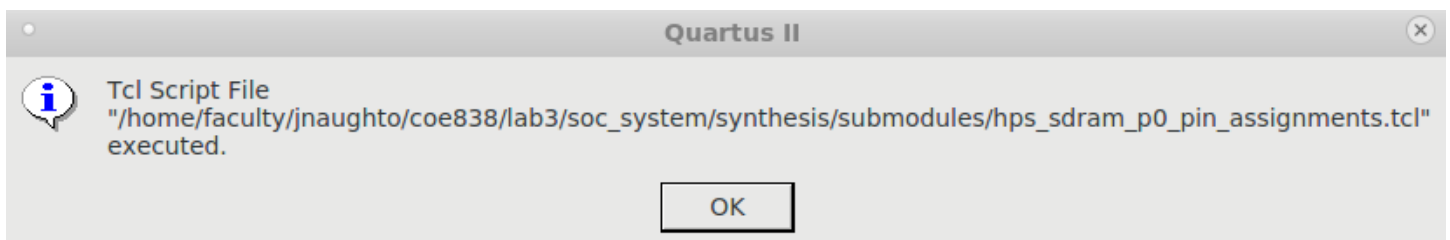
NOTE: A **bug** exists in Quartus 14.0. If you see *hps_sdram_p0_pin_assignments.tcl* listed more than once as shown in Fig. 7(a), exit Quartus and restart. Repeat steps 1 and 2. Before going to the next step, ensure that the window lists the .tcl script once as in Fig. 7(b).

There is a bug with Quartus that will generate a list of scripts as above on the left. If you have this then you need to save your project. Close the project and then reopen the project. You should now only see the scripts listed on the right in the above graphic.

If the script window is correct, Select **pin_assignment_DE1-SoC.tcl** and click Run. A popup window will show up when successful. Click OK.



Next, select the **hps_sdram_p0_pin_assignments.tcl** and click Run. A popup window will show up when successful. Click OK. This will take some time to complete [~ 20 seconds] but you should see:



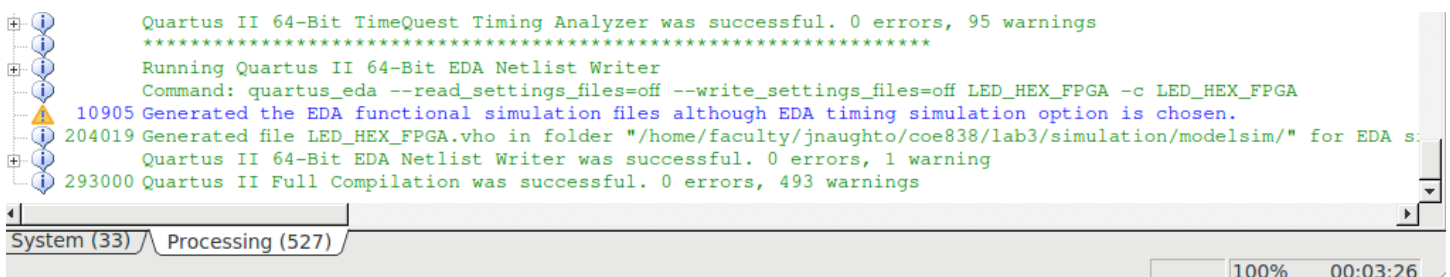
Once complete, click Close on the .tcl script window.

Compile Design

The final step is now to compile the design. Select "Processing" - "Start Compilation". Assuming that the RTL is coded correctly, no errors should exist. However there will likely be multiple warnings. You may ignore these. You may experience the compile fail and produce:

```
11802 Can't fit design in device
Quartus II 64-Bit Fitter was unsuccessful. 1449 errors, 3 warnings
Peak virtual memory: 1057 megabytes
Processing ended: Tue Feb 8 17:28:24 2022
Elapsed time: 00:00:08
Total CPU time (on all processors): 00:00:07
293001 Quartus II Full Compilation was unsuccessful. 1451 errors, 88 warnings
```

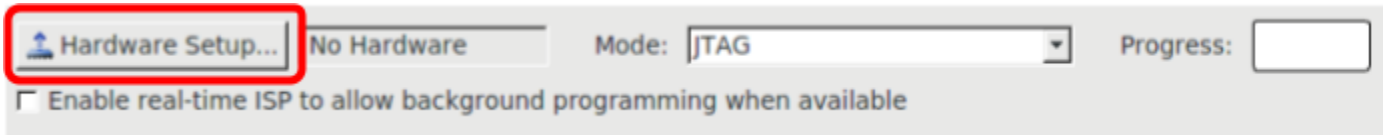
If this happens, select "Tools" - "Tcl Scripts...". re-run the TCL script **hps_sdram_p0_pin_assignments**. Now go back to the compile Select "Processing" - "Start Compilation". You will now notice that the compile completes. The compile process will take about 2-3 minutes to complete as it's quite a process. The finished process you should see at the bottom of the Message window:



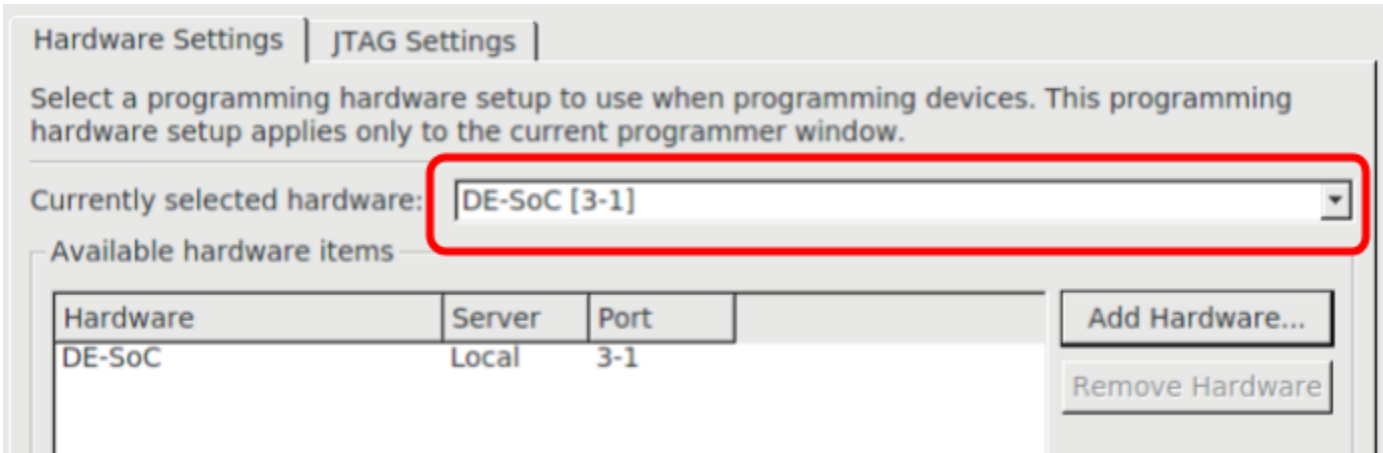
As you can see now the compile is successfully completed.

Programming the FPGA

Now we have the altera image to flash to the FPGA now need to launch the programmer by going to "Tools" – "Programmer", or click the icon. The Quartus Programmer window will open. Press the Hardware Setup button at the top left of the window.



Drop down the Currently selected hardware from being presently "No Hardware" and have it set to DE1-SoC



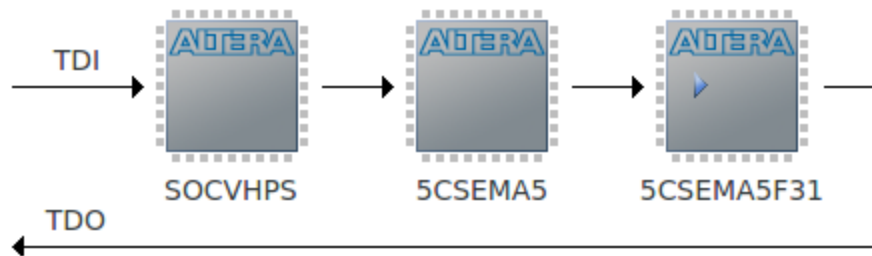
After it's selected, select close.

Note: The programmer must have the USB cabling attached to the desktop computer for Quartus to see the board.

In the Programmer window's left panel, select Auto Detect. This will automatically detect the device connected to your host computer via JTAG. A popup window will appear, select the second option which reads "5CSEMA5" and press OK.



Next, we need to upload the .sof file to the FPGA. In the left pane, select "Add File...". Navigate to your project folder's **output_files** folder and select the **HEX_LED_FPGA.sof** file. Once selected press OK.



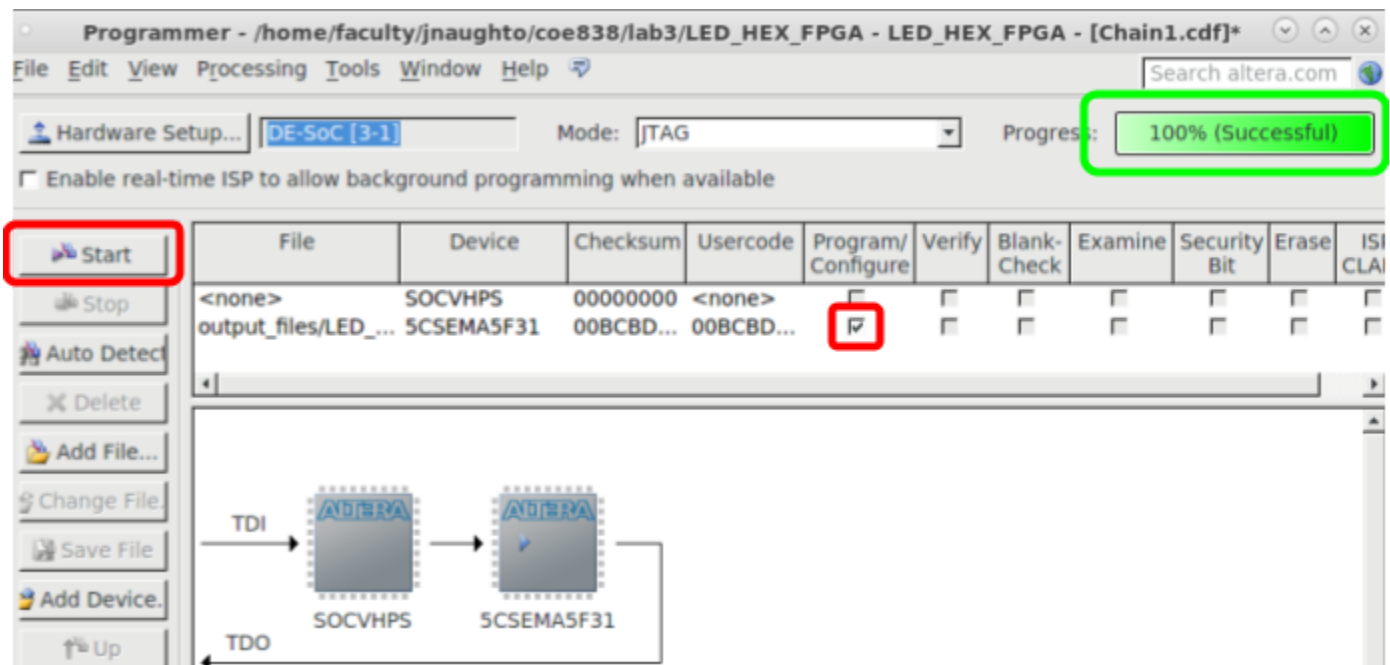
Now you will have 3 chips/devices present in the programmer. Click on the 5CSEMA5 device containing no .sof file (in the top window). Then from the left hand panel click Delete.

File	Device	Checksum	Usercode	Program/Configure	Verify	Blank-Check	Examine
<none>	SOCVHPS	00000000	<none>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<none>	5CSEMA5	00000000	<none>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
output_files/LED_...	5CSEMA5F31	00BCBD...	00BCBD...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

You should now have the SOCVHPS and .sof device present in the programmer space:

File	Device	Checksum	Usercode	Program/Configure	Verify	Blank-Check	Examine
<none>	SOCVHPS	00000000	<none>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
output_files/LED_...	5CSEMA5F31	00BCBD...	00BCBD...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

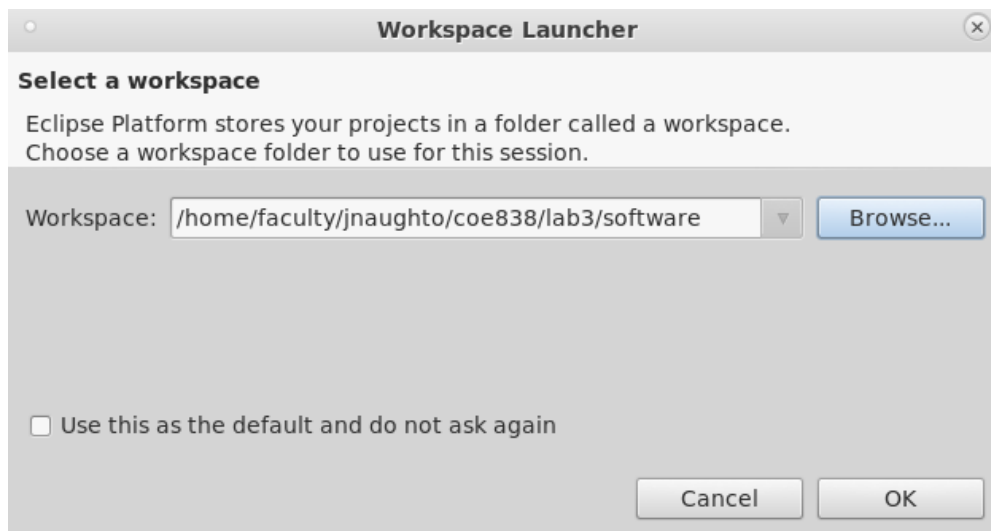
In the Programmer's top window, make sure that the .sof device has a checkmark in the "Program/Configure" checkbox. In the left pane, select start. A status bar in the upper right corner will show the progress of your .sof bitstream upload to the DE1-SoC. You should also see a green light on the board. Wait until it is 100% successful. Your FPGA (hardware prototype) has now been successfully configured as an HPS/FPGA system.



Now you should notice that all the 7 segment displays are full on and there are no changes visually on the SOC1 Board. The FPGA is programmed now we need to compile some C code to interact with the ARM which in turn will access the hardware components on the board attached to the FPGA.

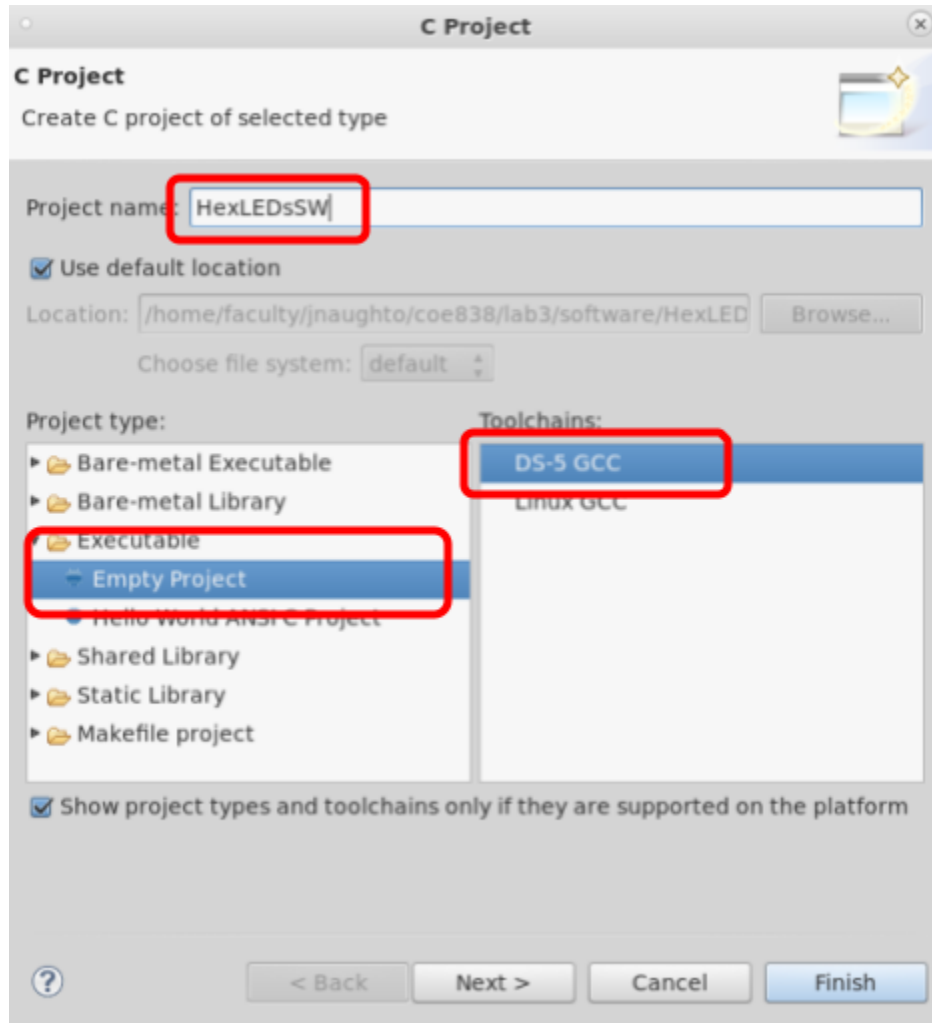
Software Design

On the EE network, navigate to Applications → Engineering → ARM Altera Eclipse for DS-5. Launch the application. The first time you run the ARM DS-5 IDE environment it will take about 45 seconds for the environment files to be setup in your home directory. You will be asked to provide a DS5 working directory. Select your project workspace as the /software folder location you created earlier on. Press OK.

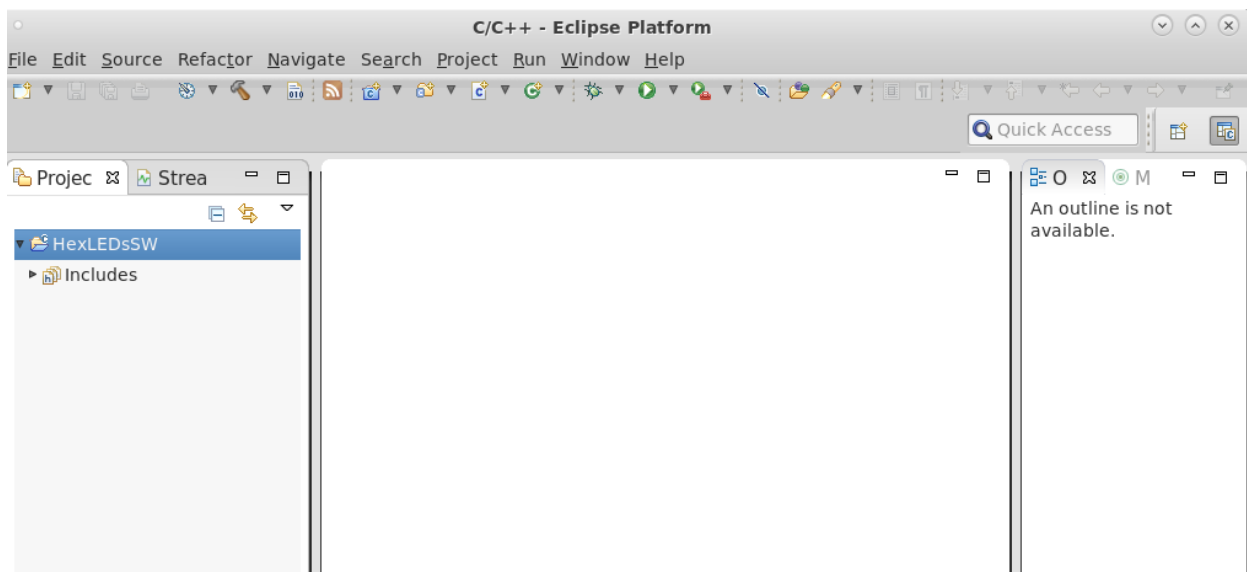


Next create a new project by selecting "File" → "New" → "C-Project". A window will show up. In the Project type window select "Executable" → "Empty Project". Under Toolchains ensure the "DS-5 GCC" compiler is

chosen. Name the project HexLEDsSW, ensure that your default project location is correct, and that your project specifications the graphic below.



Press Finish. Next click on the close tab for the Welcome screen and you will see your project:



The lab source code has already been supplied to you. Simply copy the source code from the course directory into the software folder. For example:

```
cp /home/courses/coe838/labs/lab3/software/*.h ${HOME}/coe838/lab3/software
cp /home/courses/coe838/labs/lab3/software/*.c ${HOME}/coe838/lab3/software
```

Issue these commands obviously on the terminal window to copy the files from the lab course folder into your working folder. Next, create a zip file of all the contents in your software folder (including the .h file generated by NIOS II shell) and call it software.zip. For example:

```
jnaughto@clarkson: ~/coe838/lab3/software$ zip software.zip *.c *.h
adding: led.c (deflated 62%)
adding: main.c (deflated 56%)
adding: seg7.c (deflated 66%)
adding: hps_0_arm_a9_0.h (deflated 84%)
adding: hps_0_arm_a9_1.h (deflated 84%)
adding: hps_0_bridges.h (deflated 84%)
adding: hps_0.h (deflated 60%)
adding: led.h (deflated 40%)
adding: seg7.h (deflated 47%)
adding: soc_system.h (deflated 93%)
```

Now there's a file called **software.zip** in the software folder. We're going to use this to import into the Eclipse editor. Go back to DS-5 and select "File" - "Import" - "General" - "Archive File". Click Next. Under "From archive file" select the browse button, navigate to your software folder and select the software.zip file. A list of files will show. Select "Finish". Your workspace will now have all the files included in your .zip file.

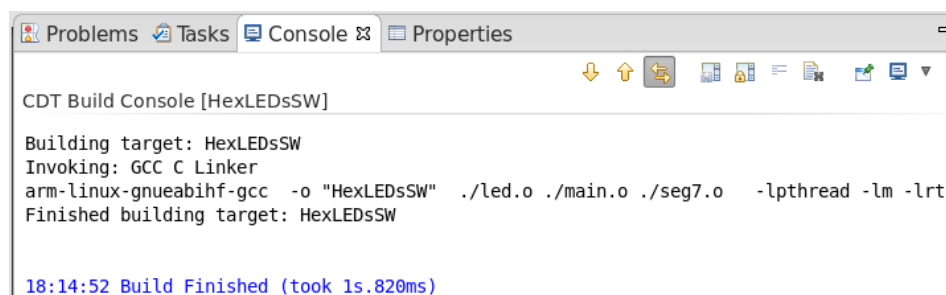
Libraries: Next, we will need to include the necessary libraries for compiling this project. Select "Project" - "Properties" - "C/C++ Build" - "Settings". Under the "Tool Settings" tab go to "GCC C Linker" - "Libraries". Select the icon under the Libraries (-l) option. A window will popup. Type in pthread and press OK. Do this again to add the m and rt libraries.

Includes: Next, we must include a path specific to the Altera SoC hardware API files. In the same Project Properties window, navigate to "C/C++ General" - "Path and Symbols" - "Includes" tab, highlight GNU C and click "Add...". In the directory field type:

/usr/local/Quartus-EDS-14.0/embedded/ip/altera/hps/altera_hwlib/include

and press OK. Select OK at the bottom of the Project Properties window.

Build your project using the shortcut CTRL-B or pressing the icon . A successful build message should appear in the console window.



```
Problems Tasks Console Properties
CDT Build Console [HexLEDsSW]

Building target: HexLEDsSW
Invoking: GCC C Linker
arm-linux-gnueabi-gcc -o "HexLEDsSW" ./led.o ./main.o ./seg7.o -lpthread -lm -lrt
Finished building target: HexLEDsSW

18:14:52 Build Finished (took 1s.820ms)
```

Next, plug in your USB to the host computer. Navigate to your /software folder and into its /Debug folder. Assuming a successful build in Step 5, an executable file named HexLEDsSW (or the equivalent name which you gave your project) will exist. Copy this onto your USB stick. Eject the USB from the host computer.

The SOC1 board is already powered and has the SD card that has a pre-installed linux installation on it. You can access it by running the minicom command at a terminal as:

minicom SOC-USB0

Minicom has already been configured to connect via the serial port to the SOC-DE1 board. You will now see a window pop up. Hit enter and you will be presented with a login prompt:

```
Welcome to minicom 2.6.2

OPTIONS: I18n
Compiled on Jun 10 2014, 03:20:53.
Port /dev/ttyUSB0, 14:46:16

Press CTRL-A Z for help on special keys

Poky 8.0 (Yocto Project 1.3 Reference Distro) 1.3
ttyS0

socfpga login: █
```

If no text is displayed in the terminal, press enter. A prompt from the OS will show up asking for login credentials. The first prompt will show:

- socfpga login: **bob**
- This signifies that your username is **bob**. Next a password prompt will show.
- Password: **bob**
- Thus your username and password is **bob**

Now take the USB key that you've put the HexLEDsSW ARM binary on, and plug in your USB drive into one of the USB ports at the back of the DE1-SoC board. A message will appear listing that you have added a device to the system located at /dev/sda1. You now need to mount your USB to Linux so that you can access the files on your drive. To do this, in the serial terminal type **mount /mnt**

Since you do not have any executable privileges on your mounted device, you must copy your executable to your working directory. Do this by typing the command: **cp /mnt/HexLEDsSW .** (include the . when issuing the command to signify that you wish to copy the executable in the 1st argument to this directory).

```
bob@socfpga:~$ mount /mnt
bob@socfpga:~$ cp /mnt/HexLEDsSW .
bob@socfpga:~$ █
```

In the terminal, type in:

```
launcher HexLEDsSW
```

This will execute the binary (we cross-compiled in DS-5) as an application on your HPS/FPGA system. You should see the LEDs rotate and blink, and the HEX switch numbers and display a message at the end. To stop execution, simply press CTRL-C. The status of the SoC will also be displayed in the serial terminal.

Analyze the main.c file and its support code files. Read the comments and understand the logistics of creating a software application for virtual memory mapping a HPS/FPGA system in Linux.