



Basic Altera SoC HPS Usage

Version 2.0

March 17, 2014

Corporate HQ & Design Center
380 Stevens Ave. Suite 206
Solana Beach, CA 92075
<http://www.macnica-na.com>

About Macnica Americas

Macnica Americas is a franchised semiconductor distributor for multiple, high-tech suppliers within North America. Our business model emphasizes unsurpassed technical support and knowledge versus other distribution options at no cost premium. Macnica Americas is the North American based division of Macnica Inc., a \$2.4B global leader in semiconductor distribution. We maintain a field support staff as well as centralized design & applications teams.



Optional design services are headquartered in San Diego, CA., USA and offer partial or full turnkey design of FPGAs, power distribution networks, and full PCB design. Our expertise includes all aspects of high speed communications protocols and networking, video broadcast, signal processing, and storage applications. Macnica's specialty is high density, high speed complex FPGA designs utilizing multiple IP cores with fast time to market requirements.

Macnica can help you deliver a winning project with the unique combination of technical support, custom IP, and design services. Setup a meeting today!

<http://www.macnica-na.com/web/americas/home>

License and Terms of Use

This lab with its associated source code and support files, are being provided on an "as-is" basis and as an accommodation. Therefore all warranties, representations or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed.

This source code may only be used in an Altera programmable logic device and may not be distributed without permission from Macnica Americas, Inc. It is provided free of royalties or fees of any kind.

Table of Contents

About Macnica Americas	2
License and Terms of Use	2
1 Lab Overview.....	4
1.1 Introduction and Goals	4
1.2 Hardware and Software Requirements	4
1.3 Assistance.....	4
1.4 Lab Agenda and Milestones	4
2 Lab Instructions.....	6
2.1 Create Project Source	6
2.1.1 Create Quartus Project	6
2.1.2 Create Qsys System – Components and Interconnect.....	8
2.1.3 Create Qsys System – HPS Configuration	10
2.1.4 Create Qsys System –Review, and Generate	13
2.1.5 Instance Qsys System in Quartus	14
2.2 Make Project Assignments.....	15
2.2.1 Source file assignments.....	15
2.2.2 I/O Assignments	15
2.2.3 Timing Assignments	17
2.3 Compile Project and Resolve Errors.....	17
2.3.1 (Purposeful HPS pin out error).....	17
2.3.2 Review Successful Compilation	18
2.4 Interact and Verify	18
2.4.1 System Console – program and link.....	18
2.4.2 System Console – fpga_console.....	20
2.4.3 System Console – hps_console	20
2.4.4 SystemConsole – Custom GUI.....	21
2.5 Review Handoff Files.....	21
3 Notes	22
Document Revision History.....	23

1 Lab Overview

1.1 Introduction and Goals

This lab is designed as a self-paced-learning tool for basic Altera SoC hardware development. In reaching this goal, only a relatively small subset of SoC features, peripherals, and options are explored. It is highly recommended persons attend additional training, such as that offered by Altera directly, for more detailed education on this rather complex flow and device family: [Altera Training](#)

The lab is broken into a series of major sections or milestones representing the common phases of design with the Altera SoC product. Unlike other trainings you may have had, this lab does not explicitly indicate every button to push in explicit order. Instead, your goal is described with the necessary information given with exact steps left to the user. If you are having problems, each section concludes with a series of hints related to the tasks proposed. In the event you are still unable to achieve the desired functionality, or you simply wish to double-check your progress, a completed project has been provided with the lab materials in the `~/SoC_2_lab/solution` folder.

*A minimal working knowledge of Quartus is expected. For someone completely new to the Altera FPGA development tools, it is suggested they complete tutorials integrated within Quartus under the **Help** pull-down menu prior to starting this lab.*

All HDL is coded in Verilog HDL, but the logic is so basic VHDL users should not find this at all a limitation in their experience.

1.2 Hardware and Software Requirements

Review [vWorkshops Getting Started.pdf](#) document for a detailed reference on installing the necessary software and burning the microSD card per following requirement list:

- Macnica Helio SoC Evaluation board w/micro-USB cable
- Quartus 13.1 installed on local machine
- microSD card loaded with Helio image as documented [vWorkshops Getting Started.pdf](#)
- Microsoft Excel 2010 or later (*recommended for optional tools, but not absolute necessary*)

1.3 Assistance

A dedicated e-mail account has been setup to receive support requests for the vWorkshop series. Please identify the course (in this case Basic HPS Usage) in addition to details on the question. workshophelp@macnica.com

1.4 Lab Agenda and Milestones

Create Project Source

You will start from “blank slate” creating a new Quartus project, assembling a relatively simple SoC system using Qsys, and incorporating all project sources. This step includes the rather beefy topic of HPS core instantiation and configuration within the Qsys environment.

Make Project Assignments

All Quartus projects require assignments such as I/O locations, I/O standards, timing analysis, and compilation settings. You will explore ways to view assignments as well as make the appropriate assignments in a design targeting an Altera SoC device. In this case, assignments are selected to be compatible with the Macnica Helio SoC Evaluation platform.

Compile Project and Resolve Errors

Next, you will compile your design observing and resolving common errors in an SoC flow. The lab purposefully sets up a few conditions of error such that you might experience fault messages before and after resolution.

Interact and Verify

Using the live-debug tool System Console, you will load your design into the Macnica Helio SoC Evaluation board and interact with your design in hardware to verify its functionality.

Review Handoff Files

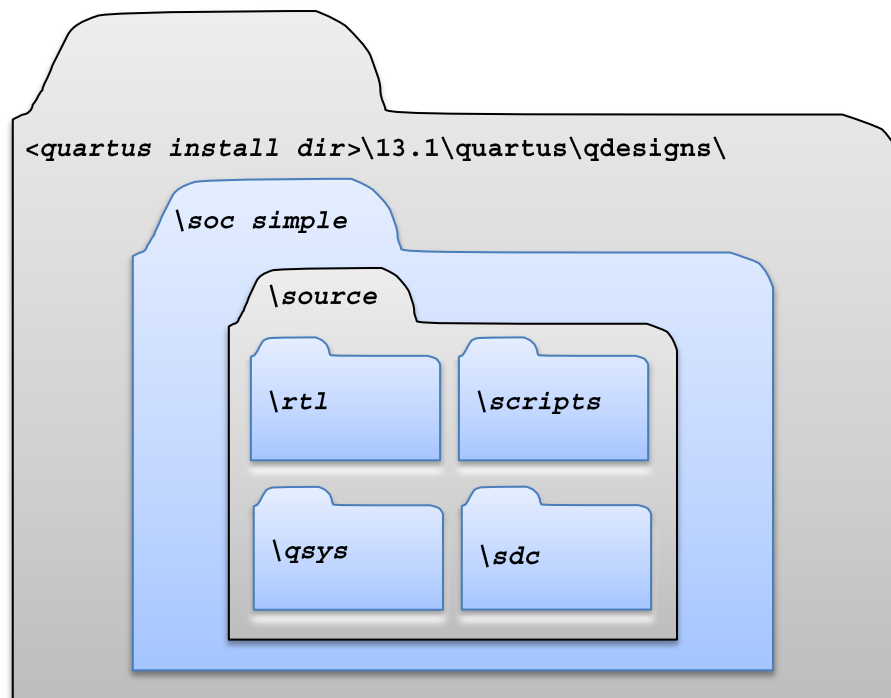
With functionality verified, you will review handoff files of your completed system to the software tools of the Altera SoC development flow.

2 Lab Instructions

2.1 Create Project Source

2.1.1 Create Quartus Project

- In order to match the solution project, create the following directory structure within **<altera_install_dir>/13.1/quartus/qdesigns**. While not required, it is convenient to separate source files in this manner to better identify what is user-created and what is machine generated throughout the lab.

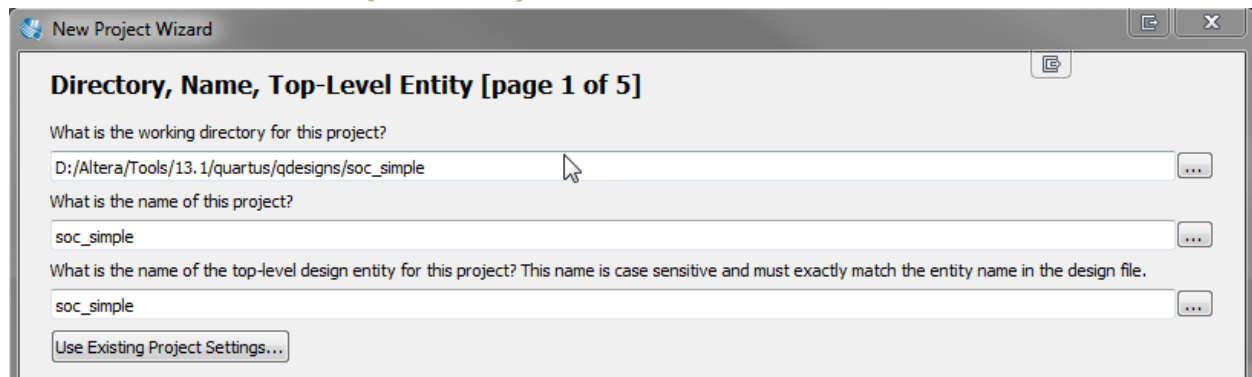


- Launch Quartus and use the **New Project Wizard** to create a new project targeting the Mpression Helio SoC Development kit. In the wizard, use **<altera_install_dir>/13.1/quartus/qdesigns/soc_simple** as the project folder and the name **soc_simple** as the project and top level name. Leave all other settings default except for the target device which must match what is populated on the Mpression Helio SoC Development Board. After completing the wizard, review your directory structure and notice the changes. Review the files with a text editor.

- While not absolutely necessary to complete the lab, it is a good exercise to understand the SoC part of the devkit. Decode the Altera SoC part number using what is provided in */SoC_2_lab/resources* and fill in the table below.

Description:	Value:	Means:
4 Digit Family Variant		
1 Digit Hard IP Code		
1 Digit Product Line Code		
1 Digit Density Code		
1 Digit Transceiver-Count Code		
1 Digit Transceiver-Speed Code		
3 Digit Package Code		
1 Digit Temp Code		
1 Digit Device Speed Code		
Optional Suffix Code		

2.1.1.1 Hints: Create Quartus Project



- Find Quartus on the Start Menu. The 64-bit version is recommended on 64bit Operating Systems due to the increased memory supported.
- **New Project Wizard** is under the Quartus **File** menu
- Locate the correct Altera part number by referring to the physical part silkscreen, schematics, or users-guide provided on <http://www.rocketboards.org> for the latest Mpression Helio documentation

2.1.2 Create Qsys System – Components and Interconnect

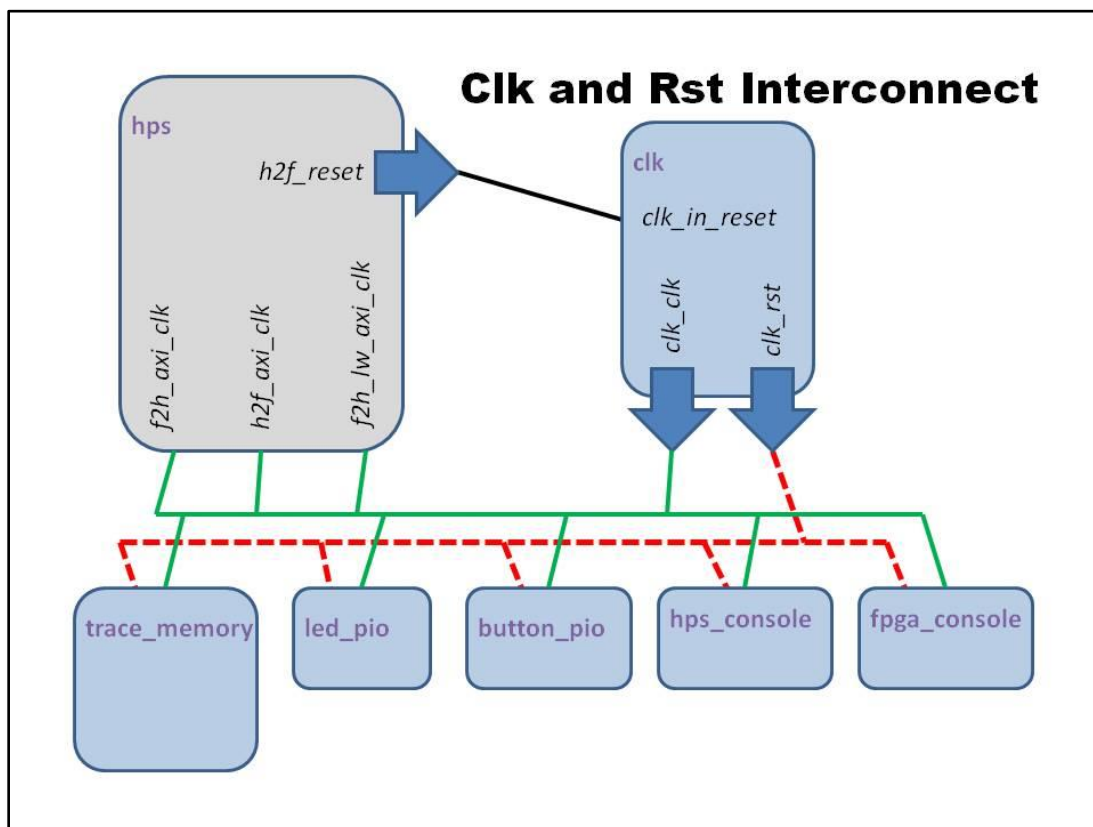
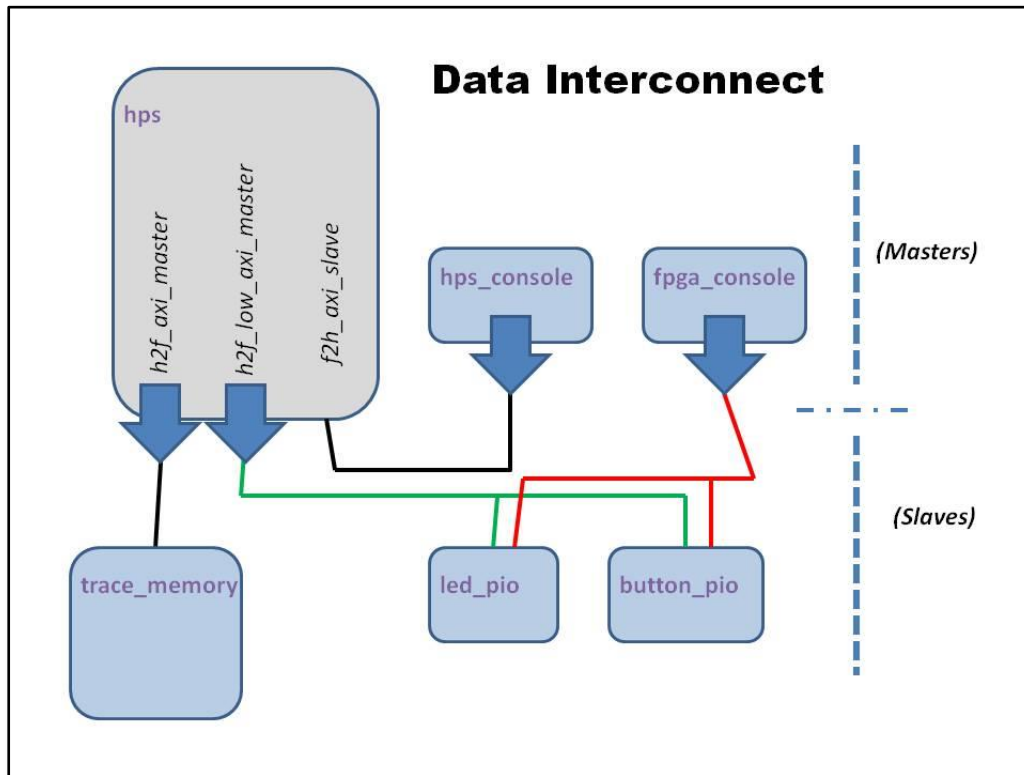
- Invoke Qsys, and use it to add components to the system and rename them. Accept the default configuration for each component added by clicking **Finish** in the lower right -- we modify configurations later. All the information you need in this step is shown in the table below.

Name in Component Library	Name in System
Clock Source (appears by default in new system)	clk
Hard Processor System	hps
On-Chip Memory (RAM)	trace_memory
JTAG to Avalon Master	hps_console
JTAG to Avalon Master	fpga_console
PIO (Parallel I/O)	led_pio
PIO (Parallel I/O)	button_pio

- Double-click the **hps** component to edit its configuration. On the FPGA interfaces panel, uncheck the **MPU Standby and Event** box and then highlight/delete the **FPGA-to-HPS SDRAM** Interface shown beneath the AXI Bridge section. This will remove these connection points in Qsys improving visibility and we will not be using them in our lab.
- Our interconnect options are looking pretty overwhelming. Create two custom filters for viewing the Qsys interconnect per the table below. (Tip: this requires using an icon in the vertical toolbar.

New Filter Name	Match Any	And Match All
Data Interconnect	<blank>	Interface / is not / interrupt Interface / is not / clock Interface / is not / reset Interface / is not / conduit
Clk and Rst Interconnect	Interface / is / clock Interface / is / reset	<blank>
Conduit Interconnect	Interface / is / conduit	<blank>

- Using your filters, replicate the connections of the diagrams below (showing data interconnect) and the diagram below (showing clock and reset interconnect). You can click the dots in the patch-panel or you can right-click and select from a text list – the latter is often easier.



- ❑ Try assigning a custom color to a master bus to further improve visibility; right click when a master bus is highlighted to access this option.
- ❑ Ignore any warning or errors for the time being. Do a **Save-AS** to save your system with name **qsystem** into the **<project>/source/qsys** folder we created earlier. Review that directory for created files. Notice that you could use the contents of the generated subfolder **.qsys_edit** to quickly copy preferences and filters between multiple projects!

2.1.2.1 Hints: Create Qsys System – Components and Interconnect

- *Qsys is launched via the **Tools** pull-down menu or by a shortcut icon on the main toolbar*
- *The provided table shows the component name in the Component Library. Use the search function to quickly locate the components to be added to your system to save manually searching*
- *Renaming is a right-click menu for a selected component. Double-click re-opens an added component.*
- *When adding a component, you are often prompted for configuration options. The **Finish** button in the lower-right will save defaults (or edits) and return to system view.*
- *To create or edit a filter, you must click the icon in the vertical toolbar that looks like a funnel. After the filter is created, you can apply/switch via the right click menu if the filter window is closed.*
- *In Qsys, click the “+” next to a component to minimize it’s interconnect. This can improve visibility greatly, particularly in more complex systems.*

2.1.3 Create Qsys System – HPS Configuration

- ❑ Using the Excel based **HPS_Pin_Muxing_ver0_9_preliminary** planner provided in the lab **/SoC_2_lab/resources** folder, fill out the table below for recommended I/O sets given the provided peripherals we wish to enable for our design.

Desired Peripheral	Desired Options	Suggested Periph #	Suggested Set #
one EMAC (RGMII)	Route to FPGA = No RMII only		
one SD/MMC	Data Width = 4 No PWREN		
one UART	No Flow Control		
one TRACE	Data Width = 8		

- ❑ Double-click/Open the **hps** component to the **Peripheral Pin Multiplexing** tab. Using the pull-downs, activate the <periph># with the pin multiplexing set and in the correct mode from the completed table above. (Note: there is an annoying refresh delay after each change as the tool updates I/O conflict information)
- ❑ We would also like to drive some GPIO from the HPS. Open the Mpression Helio schematics and search for **user_pb_hps** and make note of the HPS GPIO to which these are connected

Signal	HPS_GPIO #
user_pb_hps[3]	
user_pb_hps[2]	
user_pb_hps[1]	
user_pb_hps[0]	

- While looking at the schematics, take note of the HPS_GPIO pins used for the UART interface. Change the HPS I/O set to use these pins.

Signal	HPS_GPIO #
UART_TX	
UART_RX	

- In the bottom section of the hps **Peripheral Pin Muxing** tab, find the four GPIO signals and turn their entry on in the GPIO column by clicking on the appropriate table entry making in **BOLD**. As you do so, notice that there are conflicts with these signals and your selected EMAC. Resolve these errors without changing the GPIO used, the number of EMACs in the system, or applying white-out to your computer monitor. 😊
- We now need to configure the SDRAM. Unfortunately there is no shortcut to entering a fair amount of manual data here. The following graphics show the PHY Settings, Memory Parameters, Memory Timing, and Board Settings for use in our project. The changes from default are circled. These are all values specific to the component (Micron MT41J64M16JT-125) and layout (actual routing lengths) of the **Mpression Helio SoC Development** board. *(Consider yourself lucky, I was going to have you locate the values within the actual Micron datasheet as an exercise....)*

FPGA Interfaces Peripheral Pin Multiplexing HPS Clocks SDRAM

SDRAM Protocol: **DDR3**

PHY Settings Memory Parameters Memory Timing Board Settings

Clocks

Memory clock frequency: **400.0** MHz

☐ Use specified frequency instead of calculated frequency

Achieved memory clock frequency: **400.0** MHz

PLL reference clock frequency: **25.0** MHz

Advanced PHY Settings

Supply Voltage: **1.5V DDR3**

I/O standard: **SSTL-15**

FPGA Interfaces Peripheral Pin Multiplexing HPS Clocks SDRAM

SDRAM Protocol: **DDR3**

PHY Settings Memory Parameters Memory Timing Board Settings

Apply memory parameters from the manufacturer data sheet
Apply device presets from the preset list on the right.

Memory vendor: **Micron**

Memory format: **Discrete Device**

Memory device speed grade: **800.0** MHz

Total interface width: **32**

Number of DQS groups: **4**

Number of chip select/depth expansion: **1**

Number of clocks: **1**

Row address width: **15**

Column address width: **10**

Bank-address width: **3**

☒ Enable DM pins

☒ DQS# Enable

Memory Initialization Options

Mirror Addressing: 1 per chip select: **0**

☐ Address and command parity

Mode Register 0

Burst Length: **Burst chop 4 or 8 (on the fly)**

Read Burst Type: **Sequential**

DLL precharge power down: **DLL off**

Memory CAS latency setting: **7**

Mode Register 1

Output drive strength setting: **RZQ/6**

Memory additive CAS latency setting: **Disabled**

ODT Rtt nominal value: **RZQ/4**

Mode Register 2

Auto selfrefresh method: **Manual**

Selfrefresh temperature: **Normal**

Memory write CAS latency setting: **6**

Dynamic ODT (Rtt_WR) value: **Dynamic ODT off**

FPGA Interfaces Peripheral Pin Multiplexing HPS Clocks SDRAM

SDRAM Protocol: **DDR3**

PHY Settings Memory Parameters Memory Timing Board Settings

Apply timing parameters from the manufacturer data sheet
Apply device presets from the preset list on the right.

tIS (base):	170	ps
tIH (base):	120	ps
tDS (base):	10	ps
tOH (base):	45	ps
tDQSQ:	100	ps
tQH:	0.38	cycles
tDQSQ:	255	ps
tDQSS:	0.27	cycles
tQSH:	0.45	cycles
tDSH:	0.18	cycles
tDSS:	0.18	cycles
tINIT:	500	us
tMRD:	4	cycles
tRAS:	35.0	ns
tRCD:	13.75	ns
tRP:	13.75	ns
tREFI:	7.8	us
tRFC:	260.0	ns
tWR:	15.0	ns
tWTR:	4	cycles
tFAW:	30.0	ns
tRRL:	10.0	ns
tRTP:	10.0	ns

2.1.3.1 Hints:

- If you are not getting **HPS_IO_SET#** options in menu pull-downs of the hps component pin multiplexing tab, you likely have the incorrect part number. See the identified **Device** on the Qsys **Project Settings** tab. You should exit Qsys, correct in Quartus, and re-launch/open your qsystem.qsys file.
- From the schematics, the user_pb_hps[3:0] are on GPIO6, GPIO7, GPIO8, and GPIO9 respectively
- From the schematics, the UART_RX and UART_TX pins are on GPIO65, and GPIO 66 respectively
- To resolve the muxing error, replace EMAC0/Set0 and replace it with EMAC1/Set0

2.1.4 Create Qsys System –Review, and Generate

- We now need to tweak some of our components beyond the hps, largely to match the design of the **Macnica Helio SoC Evaluation Board** we are targeting. Double-click/open each component and enter the configuration changes from the table below.

Component	Config Change From Default
Clk	Clock frequency = 100MHz (field accepts 100m)
trace_memory	Total memory size = 65536 (field accepts 64k)
led_pio	Width = 4. Direction = Output
button_pio	Width = 3. Direction = Input

- Some Qsys components send signaling beyond the Qsys boundary into the rest of the design. These are called exports or conduits. For example, anything we want to bring to pins needs to be exported. Change to our custom filter “Conduit Interconnect” created prior. Make the names in the **Export** column match the table below. (If it exists already, you must double-click to edit the name). This is your HDL representation of the system top-level. Notice how exporting impacts the number of signals and their naming.

Component	Description	Export
hps : memory	conduit	hps_ddr3
hps : hps_io	conduit	hps_io
led_pio : external_connection	conduit	leds
button_pio : external_connection	conduit	buttons

- We have yet to create a reasonable address mapping in our system. Using the data in the following table, enter the values for the base addresses shown and then lock them by clicking the padlock icon. The provided values are somewhat arbitrary; any non-conflicting addressing would work. (You probably want to revert to your “Data Interconnect” view of the system)

Component	Base Address
hps (f2h_axi_slave)	0x0000_0000
trace_memory (s1)	0x0000_0000
led_pio	0x0001_0040
button_pio	0x0001_0080

- Your system should be complete and show **0 Errors, 0 Warnings** in the notification bar along the window bottom. Select **Generate** from the Generate pull-down menu with all options default.

Open your `<project>/source/qsys` directory and familiarize yourself with the files and directories created when the system generates. Notice how naming is derived from the name of the Qsys source file, `qsystem.qsys`.

2.1.4.1 Hints: Create Qsys System – Review and Generate

- Some conduits, like the memory of the **hps** component, are exported by default. Notice our system changes the name from default to modify the prefix used in port naming.
- You must double-click the name (or lack of name) in the **Export** column to create or change the export name
- Your currently applied interconnection filter could be hiding components. The component **clk** will not show when the filter is not showing any interfaces of type **clock**, for example.

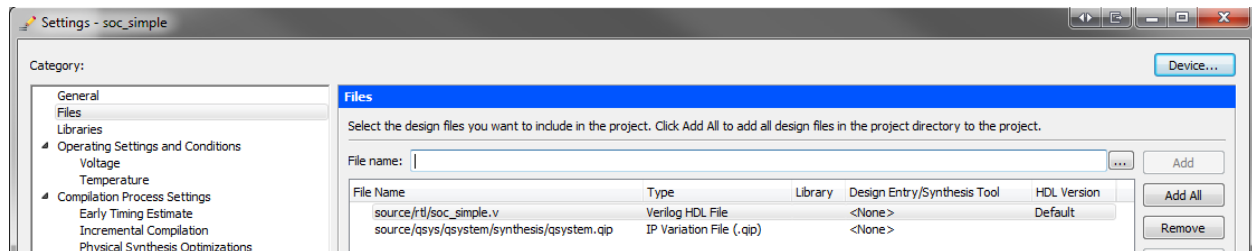
2.1.5 Instance Qsys System in Quartus

- Qsys generates source-code, but you still need to instantiate this into a custom project. But you have caught a break. Some kind soul has already created a top-level module **soc_simple** for this purpose. Copy `soc_simple.v` from the `/SoC_2_lab/resource` folder into your `<project>/source/rtl` directory. Open the file in a text editor and review its simplicity.

2.2 Make Project Assignments

2.2.1 Source file assignments

- Quartus needs to know where to find source files, particularly since we have pushed them down into a subfolder hierarchy. Add the **/source/rtl/soc_simple.v** file and the Qsys output file **/source/qsys/qsystem/synthesis/qqsystem.qip** as source files to the project.

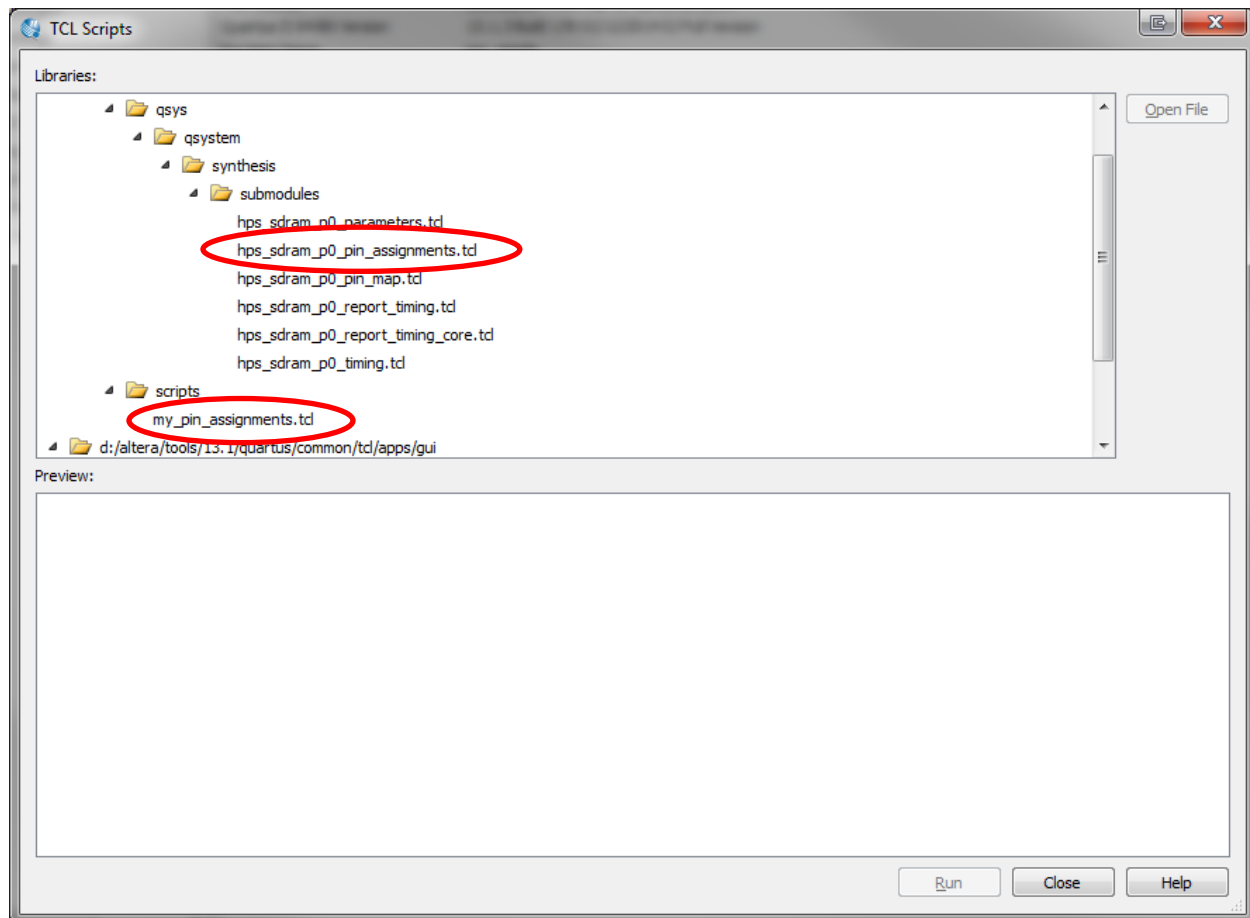


- Perform an Analysis and Synthesis. You will get plenty of warnings but should get no errors. Review the warnings and try to resolve any errors by returning to sections above. If you make any changes to the Qsys system, you will need to regenerate before re-attempting Analysis and Synthesis.

2.2.2 I/O Assignments

- Confirm there are no I/O assignments in your project currently. A successful compile requires location and I/O assignments that match the target hardware. The kind soul has returned and provided these constraints in a .tcl format. Copy the file **my_pin_assignments.tcl** from the lab **/SoC_2_lab/resource** directory to your **<project>/source/scripts** directory. Review the file in a text editor; it is always a good idea to save your own pin assignments in a separate .tcl file like this for easy reapplication in a project. Execute this .tcl file in your project and then re-examine your assignments (see tips if necessary on executing). Be sure to save the project to save the changes to the Quartus settings file.
- Recall from training the SDRAM is a special case and requires its own script be executed. (This fact is reminded to the user in the **my_pin_assignments.tcl** script just executed). Locate and run the script for the SDRAM. Review your project assignments before and after running to observe the results of the script execution.

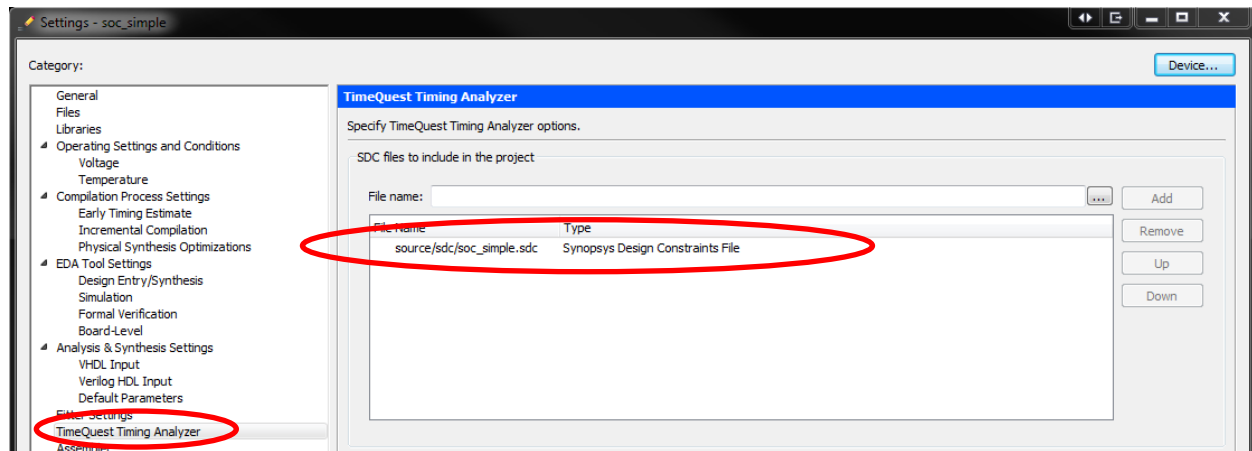
2.2.2.1 Hints: I/O Assignments



- Project source files are modified via the **Assignments: Settings** pull-down menu in Quartus. Be sure you click **Add** after entering a path so it appears in the listing – otherwise the path is not saved
- **Analysis and Synthesis** is initiated under the **Processing** menu pull-down. As another option, you can click on **Analysis and Synthesis** in the **Task** tab of the window usually showing your project hierarchy.
- The **Assignment Editor** and **Pin Planner** are both ways to view your project assignments relating to I/O. You can launch both via the aptly name **Assignments** pull-down menu in Quartus.
- You can browse/run tcl scripts using the GUI shown above. It is the **TCL Scripts** option under the **Tools** pull-down menu in Quartus.
- Alternatively, you can execute scripts by activating the TCL console utility window (under the **View** menu pull-down and using a command like **source**
".:/source/scripts/my_pin_assignments.tcl"
- Only the **hps_sdram_p0_pin_assignments.tcl** file generated by Qsys needs to be run – the others are support scripts for this **hps_sdram_p0_pin_assignments.tcl** script.

2.2.3 Timing Assignments

- Recall from training that peripherals within the HPS can be removed from timing analysis by “cutting” their timing paths. It’s a “kind-soul” trifecta --- copy the file **soc_simple.sdc** from the **/SoC_2_lab/resource** directory to the **<project>/source/sdc** directory. Review the contents of this file.
- You must additionally tell Quartus to use **soc_simple.sdc** during compilation. This is done most easily using the **Timequest** category of **Settings** under the **Assignments** pull-down menu.



2.3 Compile Project and Resolve Errors

- Perform a full compilation of Quartus. This is most often done by pushing the purple play button on the toolbar or selecting **Start Compilation** in the **Processing** pull-down menu. There are two purposeful errors that we will be resolving, so don’t panic when you get a compilation error!

2.3.1 (Purposeful HPS pin out error)

- The supplied **my_pin_assignments.tcl** file includes two, intentional assignment typos to demonstrate the checking Quartus performs on the HPS peripherals. Expand the errors and then compare to the below image of pin-out information Altera provides. Notice that only one of the two problems was caught by Quartus! Can you explain why there is a difference?

Compilation Report - soc_simple									
Assignment Editor*									
<<new>> Filter on node names: *									
		From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
341	✓	in	emac_rxd[1]	Location	PIN_A11	Yes			
342	✓	in	emac_rxd[2]	Location	PIN_C15	Yes			
343	✓	in	emac_rxd[3]	Location	PIN_A9	Yes			
344	✓	out	emac_tx_clk	I/O Standard	3.3-V LVCMOS	Yes	soc_simple		
345	✓	out	emac_tx_clk	Location	PIN_J15	Yes			
346	✓	out	emac_tx_en	I/O Standard	3.3-V LVCMOS	Yes	soc_simple		
347	✓	out	emac_tx_en	Location	PIN_A12	Yes			
348	✓	out	emac_bxd	I/O Standard	3.3-V LVCMOS	Yes	soc_simple		
349	✓	out	emac_bxd[0]	Location	PIN_A16	Yes			
350	✓	out	emac_bxd[1]	Location	PIN_J14	Yes			
351	✓	out	emac_bxd[2]	Location	PIN_A15	Yes			
352	✓	out	emac_bxd[3]	Location	PIN_D17	Yes			
353	✓	in	fpga_button	I/O Standard	2.5 V	Yes	soc_simple		

- Fix the pin assignment errors. Consider making one correction at a time, recompiling your project and understanding what does and does not cause an error.

2.3.1.1 Tips: Purposeful HPS Pin-out Error

- If you are having trouble reading the screen grab, `emac_txd[3]` should be on pin D17 and `emac_rxd[2]` should be on pin C15
- You can just modify the two assignments in error using the **Assignment Editor** of the **Assignments** pull-down menu (save when done). As another option, you can modify the script and re-execute – but you would then have to delete all existing assignments first
- Quartus identifies if a signal of an HPS peripheral is not on a dedicated HPS peripheral I/O at all, but it does not identify if it is on a dedicated peripheral I/O location that contradicts what you configured in the Qsys hps component. Take care in your own custom projects!

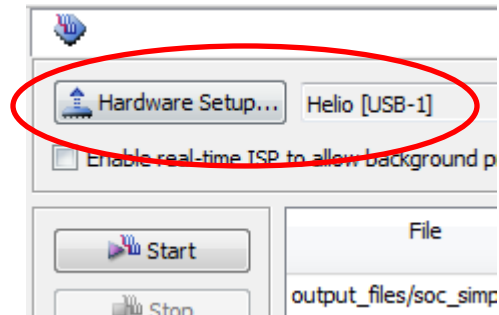
2.3.2 Review Successful Compilation

- The project should compile with no errors. You should be able to flip through the various contents of the Compilation Report. In particular, you should confirm that there is no red in **TimeQuest Timing Analyzer** stage and that the **Assembler** stage has created output files for loading our design into hardware (`<project>/output_files/soc_simple.sof`).

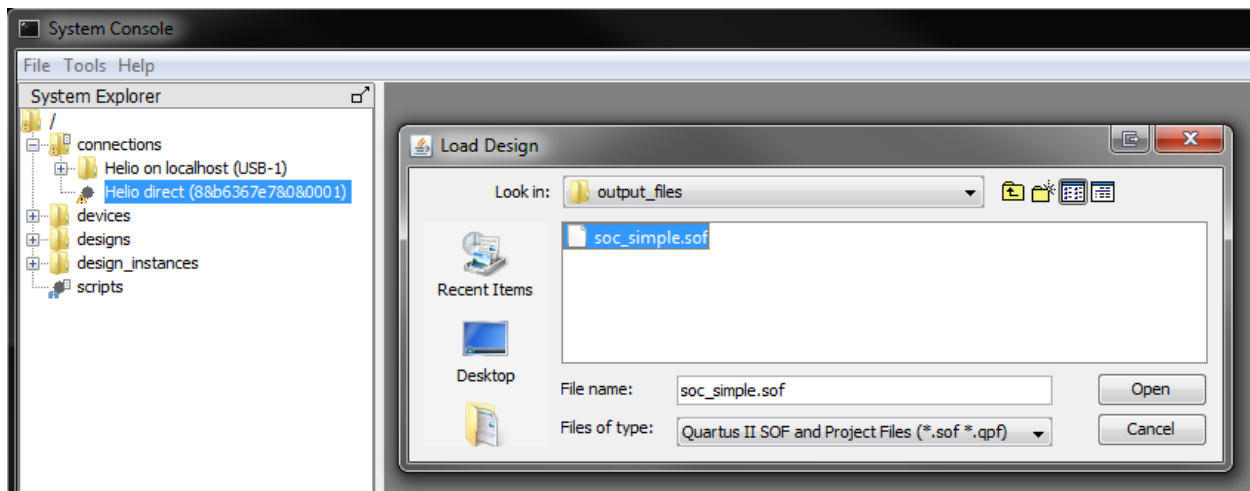
2.4 Interact and Verify

2.4.1 System Console – program and link

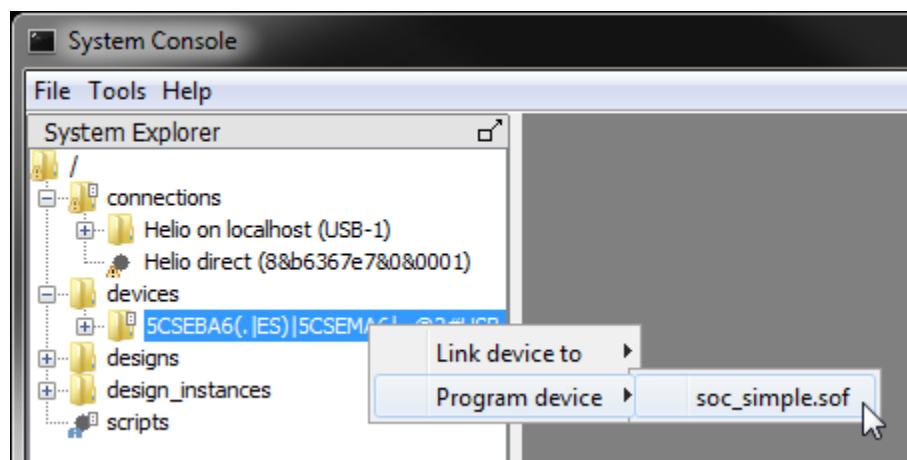
- Connect your machine running Quartus to the Mpression Helio development board port for embedded USB Blaster. Power the board and verify this connection is active (and the driver present) by launching the Quartus Programmer and doing an AutoDetect successfully. You might have to install drivers for the USB Blaster: <http://www.altera.com/download/drivers/dri-index.html>



- While we could program the FPGA with Quartus Programmer, let's do something new. Launch Qsys if it is not already open. Then launch System Console via the **Tools** menu pull-down. Verify the connections list now shows the **Helio** and then Load Design under the file menu and select **<project>/output_files/soc_simple.sof**.

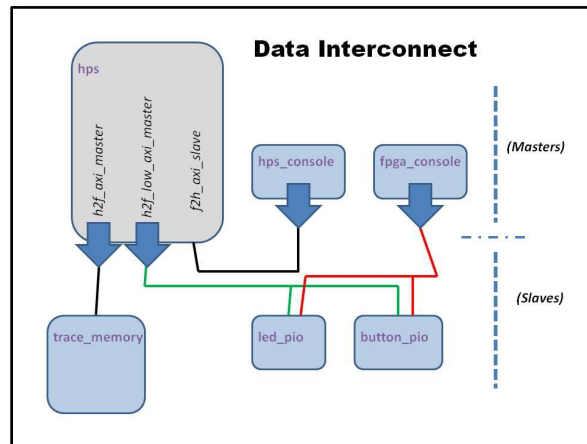


- You can now expand the devices tab and use a right-click menu to program the FPGA with **soc_simple.sof** using SystemConsole. Watch the hardware when you do so and you will see the four user LEDs activate; their default state after configuration in our design.



2.4.2 System Console – fpga_console

- Recall in our system that the fpga_console has direct connection to the led and button peripheral. Use **SystemConsole** to identify the path of the fpga_console and set it to a variable. Then use this variable to open a service path and initiate reads/writes to interact with these peripherals. Refer to training slides, the following screenshots, or hints as necessary.



```
Tcl Console
% set fpga_con [lindex [get_service_paths master] 0]
/devices/5CSEBA6(.|ES)|5CSEMA6|..@2#USB-1#Helio/(link)/JTAG/fpga_console.jtag/phy_1/fpga_console.master
% open_service master $fpga_con

% master_write_memory $fpga_con 0x10040 0x2

% master_read_memory $fpga_con 0x10080 1
0x07
%
```

2.4.2.1 Hints: SystemConsole – fpga_console

- **set <var>** sets a variable in tcl. **\$<var>** uses that value.
- **get_service_paths master** lists the possible master connections available
- you must open a service path before you can use it (e.g. **open_service master <name>**)
- Recall the leds are at address 0x10040 and the buttons at address 0x10080 in our system

2.4.3 System Console – hps_console

- Recall that hps_console does not have direct connection to the leds or button peripheral, but is connected to the fpga-to-hps interface. We can use the hps_console to launch peeks/pokes into the HPS and, if we use the correct address, have these accesses route out the hps-to-fpga interface to the leds and button peripherals! (This works because you should be using an microSD card with an image pre-loaded to configure the HPS bridges)

- Use SystemConsole to identify the path of the hps_console and set it to a variable. Then use this variable to open a service path and initiate reads/writes to interact with these peripherals through the hps. You must add the address offset of the HPS-to-FPGA Lightweight Bridge. Refer to training slides, the following screenshots, or hints as necessary.



```
Tcl Console
-----
% set hps_con [lindex [get_service_paths master] 1]
/devices/5CS(EBA6ES|XPC6C6ES)|..@2#USB-1/(link)/JTAG/hps_console.jtag/phy_0/hps_console
.master
% open_service master $hps_con
% master_write_memory $hps_con 0xff210040 0x7
%
```

2.4.3.1 Hints: SystemConsole – hps_console

- *The hps_console sees the same address space as the HPS. Recall from training the HPS memory map – the HPS-to-FPGA-Lightweight bridge is at address 0xFF200000. We thus OR this offset with the address of our led and button peripherals to peek/poke through the HPS.*

2.4.4 SystemConsole – Custom GUI

- While it is well beyond the scope of this lab, SystemConsole can be used to create custom GUIs using a widget library. As a simple example, review and copy the file **system_console_ledgui.tcl** from **/SoC_2_lab/resource** to your **<project>/source/scripts** directory. Execute this script in SystemConsole via the **File** pull-down menu. You will get an array of button widgets that will toggle the status of the LEDs.

2.5 Review Handoff Files

- Open and review the generated system description file **qsystem.html** within **<project>/source/qsys**. This is an html representation of the same HW/SW handoff data within **qsystem.sopcinfo** and defines the Qsys system containing an HPS instance we have created in this lab.
- Locate the HW/SW handoff folder **<project>/hps_isw_handoff/<system>_hps**. These files are source for SW flow and beyond the scope of this lab other than knowing where to locate them! Sign up for SoC 6 – Altera’s Embedded Development Suite (EDS) ASAP if you need to understand their incorporation into the larger SoC development flow!

3 Notes

Document Revision History

Revision	Date	Comments
0.1	May 8, 2013	Initial Draft
0.2	May 16, 2013	Internal Review
1.0	May 21, 2013	Customer Release
1.1	May 28, 2013	Fixed hps_console graphic to show write lindex value (1 vs 0)
2.0	March 14, 2014	Updated to Q-II 13.1