SoC Hardware Lab Instructions

Version 13.0 **5/20/2013** **Tutorial**

Table of Contents

OVERVIEW

The **Altera SoC** combines a **Hard Processing System** (HPS) and an **FPGA** on a **single device**. The HPS has dual core **ARM Cortex-A9 MPUs** and a host of peripherals such as DDR3 controllers, Ethernet MACs, SPI controllers and many more. The FPGA portion of the device is tightly coupled through **high performance bridges** to the HPS. The designer can add peripherals they create or from third party IP to the FPGA and map it into the HPS. **Thus you have a flexible and very powerful solution**.

This hardware lab provides an answer to following questions that a hardware developer might have:

How do I build a complete customized HPS SoC system?

How do I create a HPS in Qsys to realize a custom ARM SoC system with bridges to the FPGA?

How do I configure the HPS in Qsys to realize a unique set of HPS peripherals for a custom SoC system?

How do I use standard Qsys components to create a customized set of FPGA peripherals for my SoC system?

How do I use System Console to verify the peripherals in my SoC system are working properly?

How do I use ModelSIM to simulate and therefore validate the peripherals in my SoC system?

The HPS is **configured** using **Qsys**, Altera's SoC/FPGA IP integration tool. Configuration includes **selecting DDR memory**, determining **clock frequencies** and selecting which **HPS peripherals** your design will use. As such Qsys inherently has most of the information to satisfy the questions asked above. Qsys is also used to define the **HPS peripheral pin outs and** Quartus is used to define the **FPGA peripheral pin outs**.

These two Altera FPGA development tools will generate the **files** needed for the **transfer of design information** from the **hardware** to the **software** domain. A portion of the hardware modules will create a set of handoff files that are required to build a preloader, a system register set (including FPGA registers) and the files required to create a **Device Tree** that will support any operating system. For instance, the .socpinfo file will be used by the software designer to create a Device Tree which will in turn be utilized to provide an interface for the **Device Drivers in Linux or other operating system**.

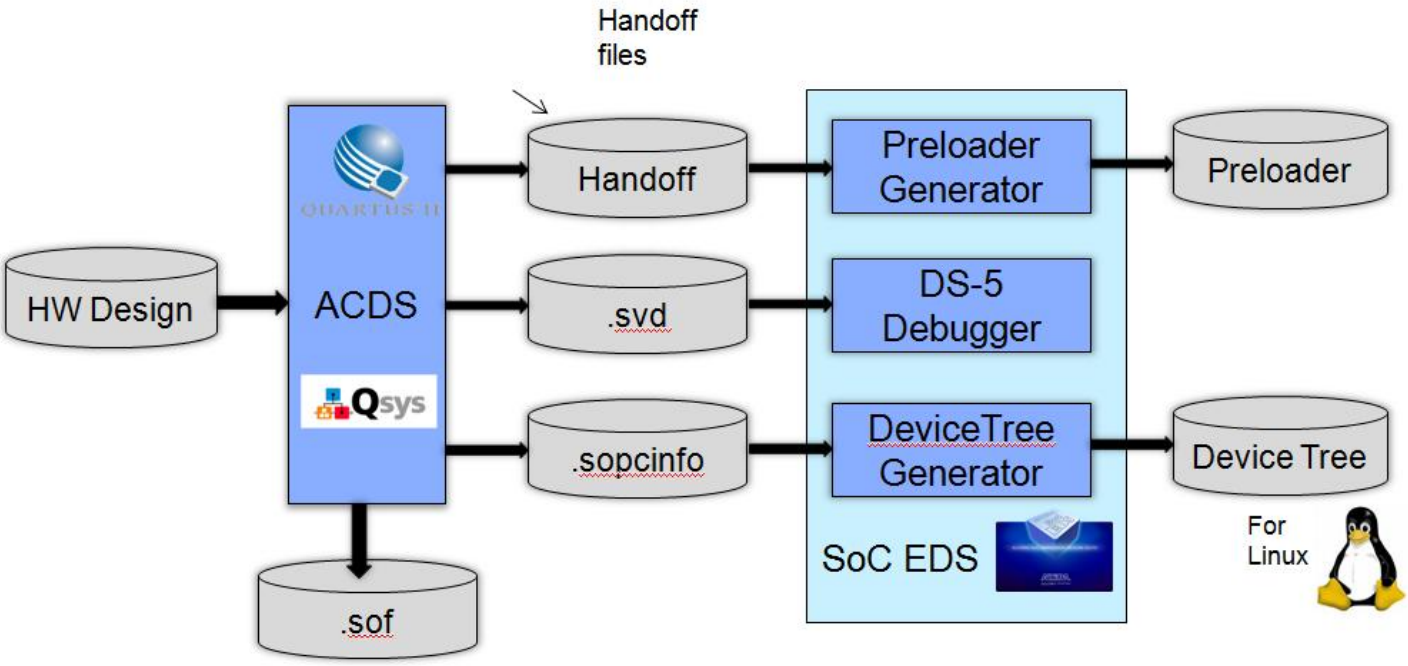
Qsys creates the files required for Hardware to Software domain transfer:

The **diagram** below shows three main areas of **transfer** from the **hardware** to **software** domains.

1. The **files** necessary to create a custom **preloader**

2. The .**svd** file that describes the FPGA **peripherals** and is used by the DS-5 **register** function

3. The **sopcinfo** file that describes all of the HPS **devices** selected in Qsys and also those custom peripherals added in the FPGA. These are used to build a **device tree**. The device tree is used by the Linux kernel to determine which device drivers to load at boot time.



Hardware Module Summary:

The Hardware labs are based on completing the **Golden Hardware Reference Design** (GHRD) that is provided with the **SoCKit**. You will examine the **architecture** of the GHRD in **Module 2**.

In **Module 3** you will learn how use Quartus II to create a Quartus II project. In **Module 4** you will utilize Qsys to build your HPS based SoC system complete with a set of HPS peripherals to interface to the peripherals on the SoCKit. Next, you will create a custom set of FPGA peripherals to interface to the HPS that also interface to peripherals on the SoCKit.

In **Module 5** you will see how to complete the Quartus II project to include pin assignments and timing constraints for the HPS and FPGA peripherals that were instantiated in Qsys.

In **Module 6** you will see how to **validate** the **peripherals** created in the FPGA using system console.

Module 7 is a bonus lab that shows how to **utilize ModelSIM for Hardware validation of the FPGA peripherals that were created in Qsys.**

Goal of this Hardware Lab:
The goal of this hardware lab is to customize the SoC hard processor system (HPS) and build custom peripherals that will be integrated with the HPS. Altera's SoC has the flexibility and customizability of adding additional peripherals to the FPGA by using Qsys.

The SoC and the custom peripherals will in turn be used by the software lab where the peripherals will be accessible by the Linux system.

This lab teaches you how to customize the HPS and peripherals. As the lab progresses, you will see how quick and easy it is to build entire systems using Altera's system integration tool, Qsys, to configure and integrate pre-verified IP blocks.

Caution:

Do not continue until you have read the following:

The names that the lab document directs you to choose for files, components, and other objects in this exercise **must be spelled *exactly* as directed.**

MODULE 1. Getting Started

Your first objective is to ensure that you have all of the items needed and to install the tools so that you are ready to create and run your design.

List of Required Items:

- Arrow Electronics **SoCKit** development board
- **Quartus II** v13.0 Web Edition
- Computer with Windows 7, 4 GB RAM, minimum of I3 core and over 10 GB free hard disk space for the Quartus II install
- **Lab Design Files**

1.1 Acquiring Cyclone V SoCKit

To order a SoCKit please click on the link below

Order an SoCKit from Arrow Electronics

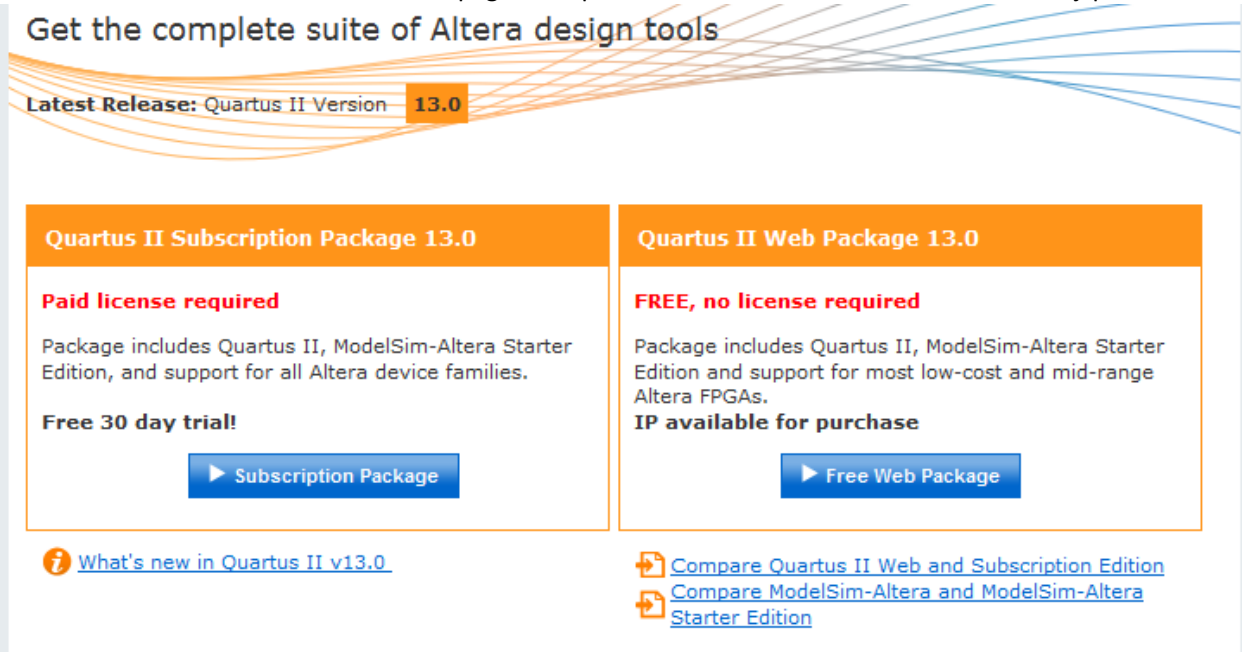
1.2 Install the Altera Design Software

You will need to install the following design software package:

- **Quartus II Web Edition design software v13.0.** – FPGA synthesis and compilation tool that contains QSys and the MegaCore IP library with the SoC processor

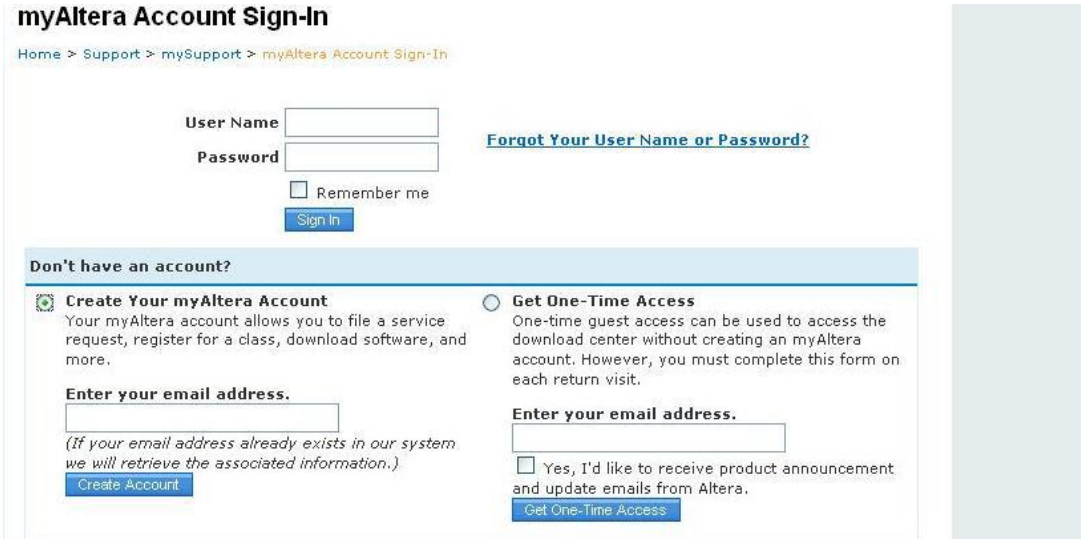
The following steps will guide you through the installation instructions. Quartus II Web Edition can be downloaded from the Altera web site. ***Please carefully follow the steps shown below***.

- Go to the Altera Download web page at https://www.altera.com/download/dnl-index.jsp

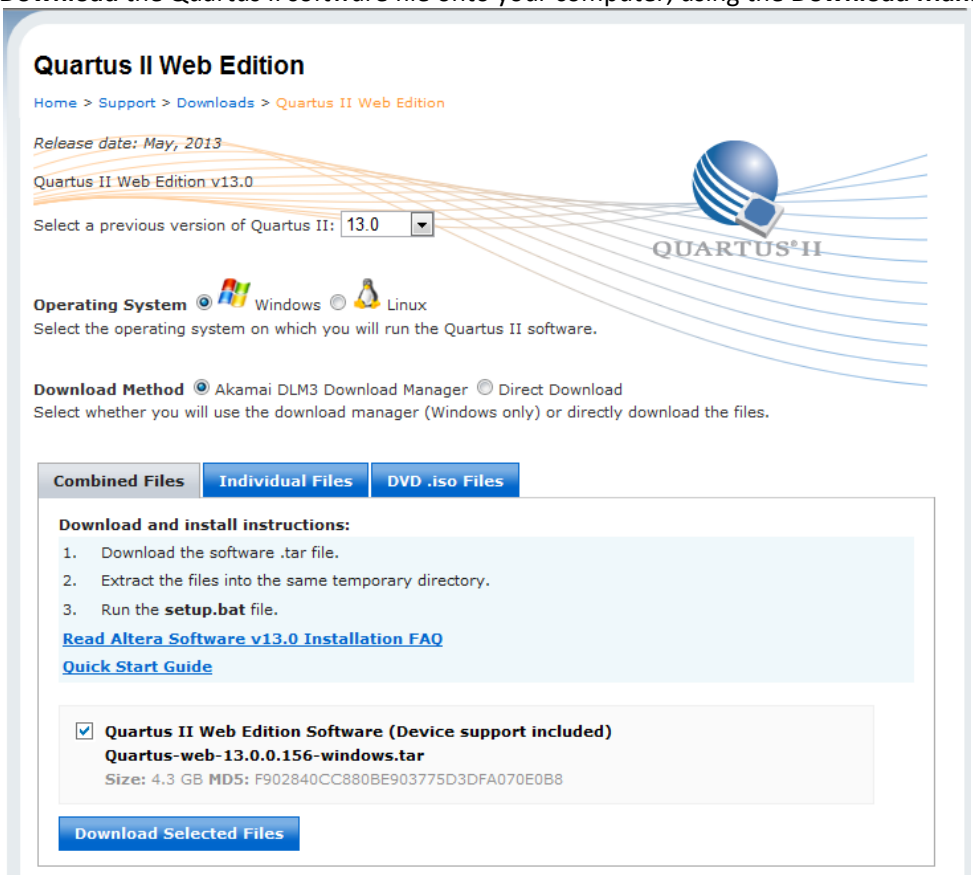


- Select "**Quartus II Web Package 13.0.** Press the **Free Web Package** button.

- Login to **myAltera** account. Use your **existing login**, or **Get One-Time Access**.



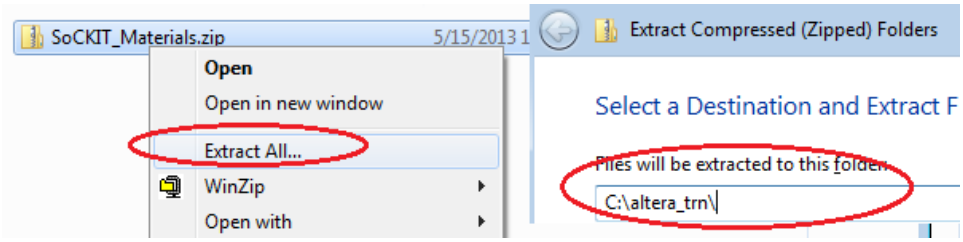
- Select **Quartus II Web Edition, Windows**:
- **Download** the Quartus II software file onto your computer, using the **Download Manager**.



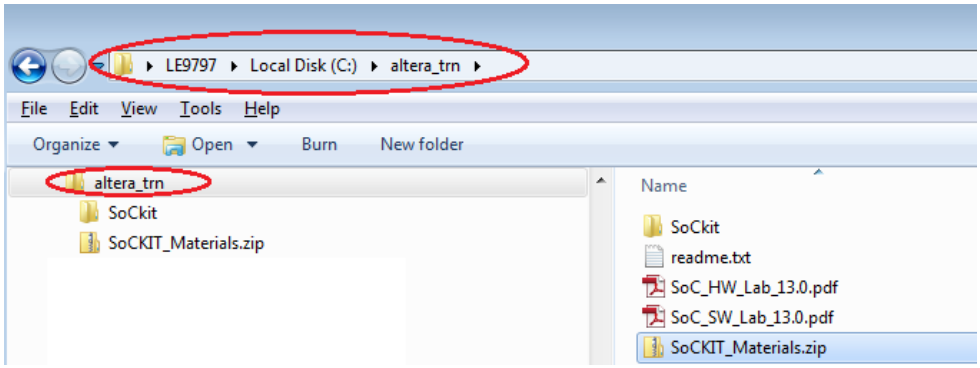
- If **Internet Explorer** is used, there could be a window that states "A website wants to opens web content using this program on your computer". **Select Allow**.
- After the **file** is downloaded on the computer, select the *.exe file, and **install the software**.
- **Select** all of the **defaults** for the installation.

1.3 Extract the SoCKit Lab Files.

- Create a folder **c:\altera_trn** on your PC.
- Browse to Arrow SoCKit webpage at **http://www.arrownac.com/solutions/sockit/**
- Click on the **Download the SoCKIT lab materials link**
- Download **SoCKIT_Materials.zip** file and save it to **c:\altera_trn** on your PC
- Extract the **SoCKIT_Materials.zip** file to this folder



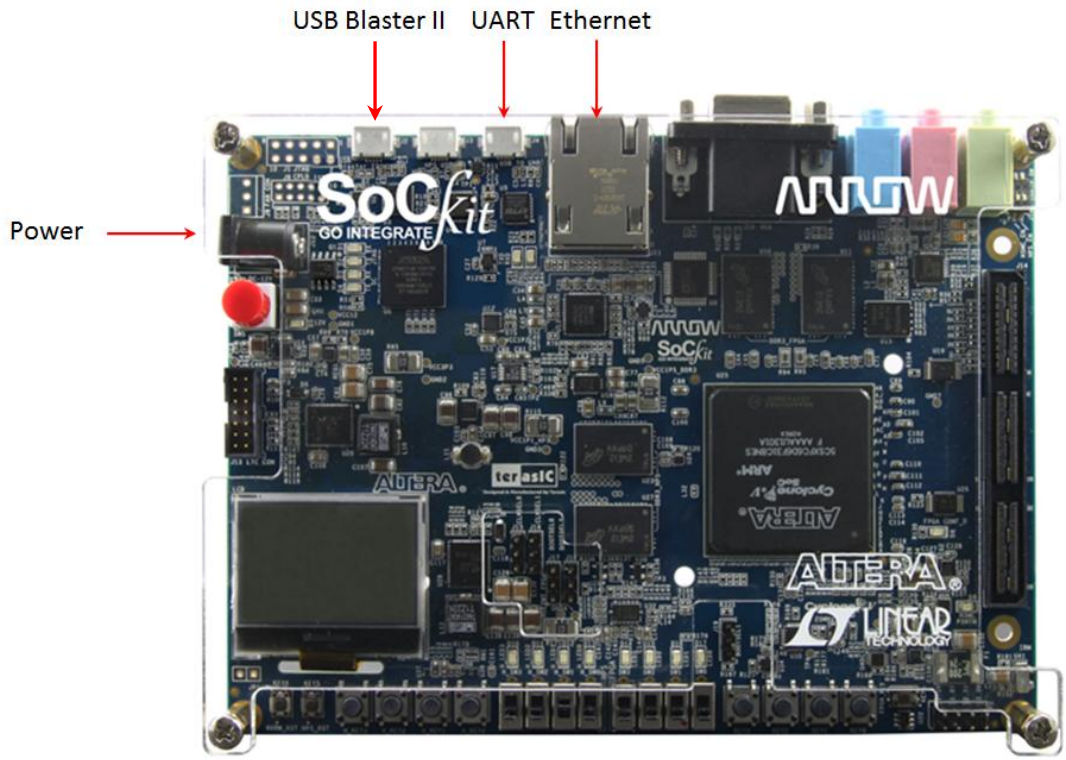
- The **c:\altera_trn** directory should look like this



1.4 Get the Cyclone V SoCKit ready for the Labs (Complete this at the Workshop)

Please connect cables to the connectors shown in the diagram below. All cables are provided in your SoCKit.

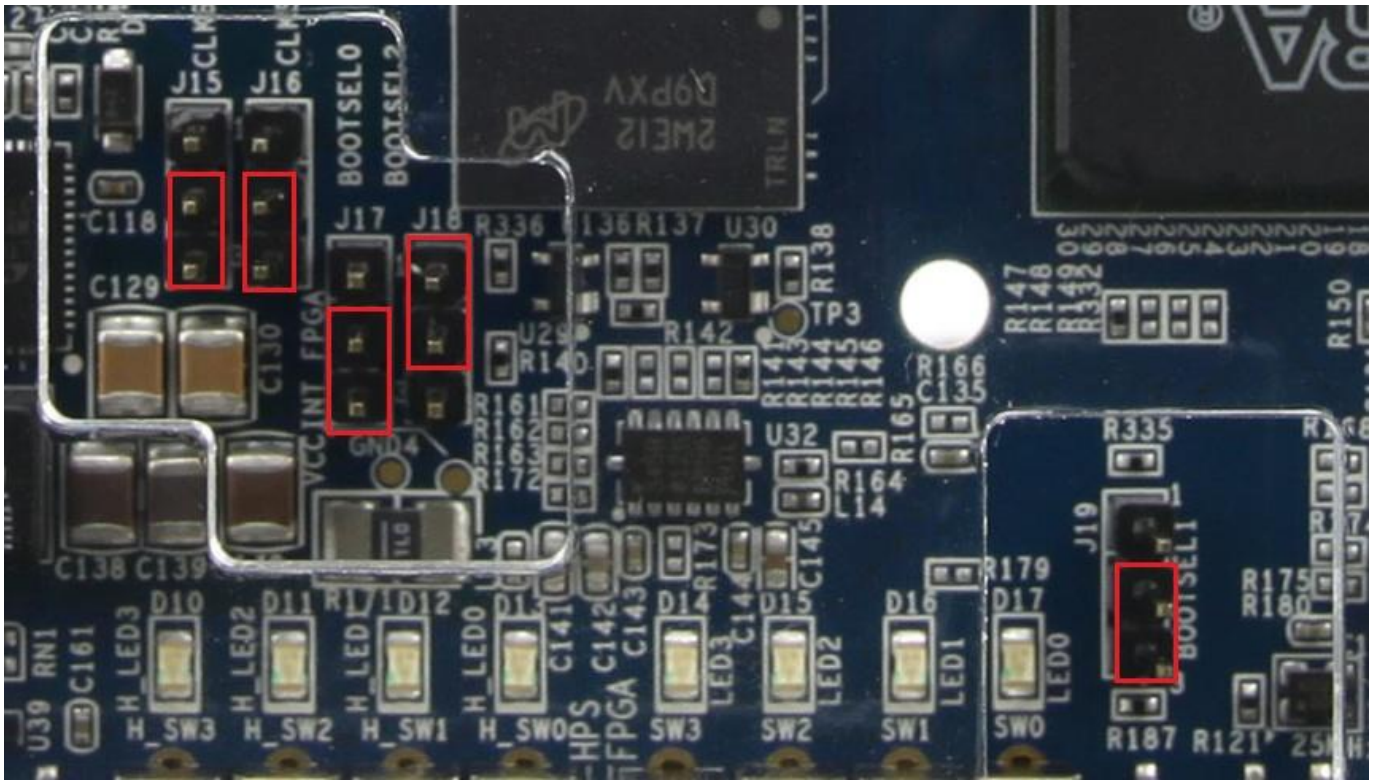
- Connect the micro USB cable to the USB host connector on your laptop and to the USB Blaster II connector on the SoCKit.
- Connect the Power Supply to the Power connector on the SoCKit.



There are a few jumpers that require configuring before proceeding with the labs.

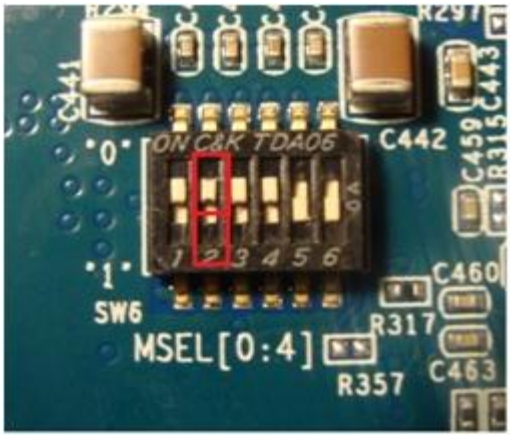
- **BOOTSEL[2..0]** jumpers. These should be configured as "100" to select boot from SD card 3.3V
- **CLKSEL[1..0]** jumpers. These should be configured as "00" for the slowest HPS peripheral clock speed option.

Please ensure that the jumpers are **configured** as **indicated** below.



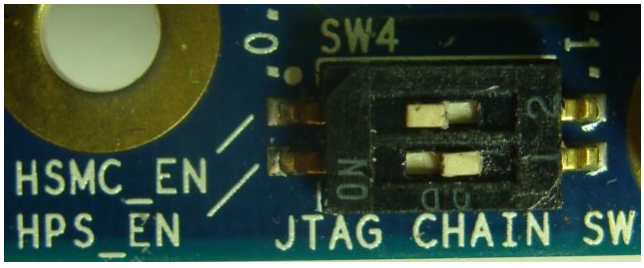
Modify the default **MSEL** bit settings.

- **SW6** is located on the **bottom** side of the SoCKit.
- Please change MSEL[0:4] to 00001.
- To do so move MSEL[1] to the '0' position.



Verify that the **JTAG chain is correctly configured**. The **JTAG chain switch** is located in to the right of the **green audio** connector.

- **HSMC_EN** should be **disabled** (left position) and the **HPS_EN** should be **enabled** (right position).

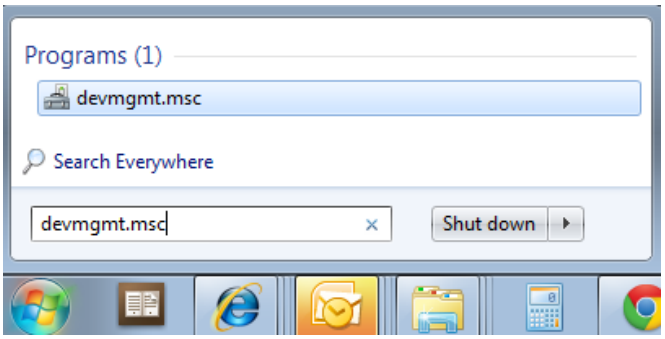


1.5 Install the USB Blaster II Device Driver (Complete this at the Workshop)

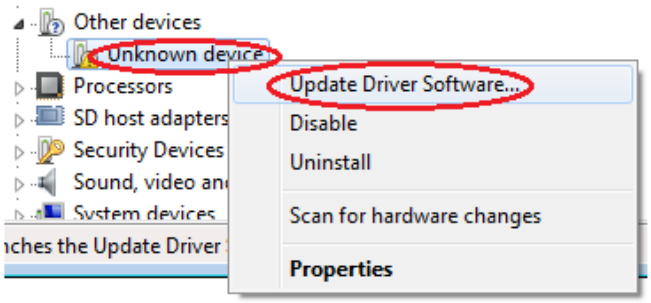
- **Turn your SoCKit on.**

Your Windows PC will try to install the driver but will not succeed.

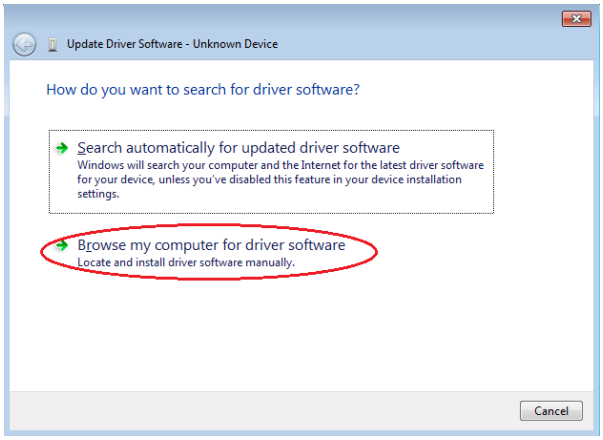
- Press the Windows **Start** button. Enter **devmgmt.msc**. Press enter to open the Device Manager.



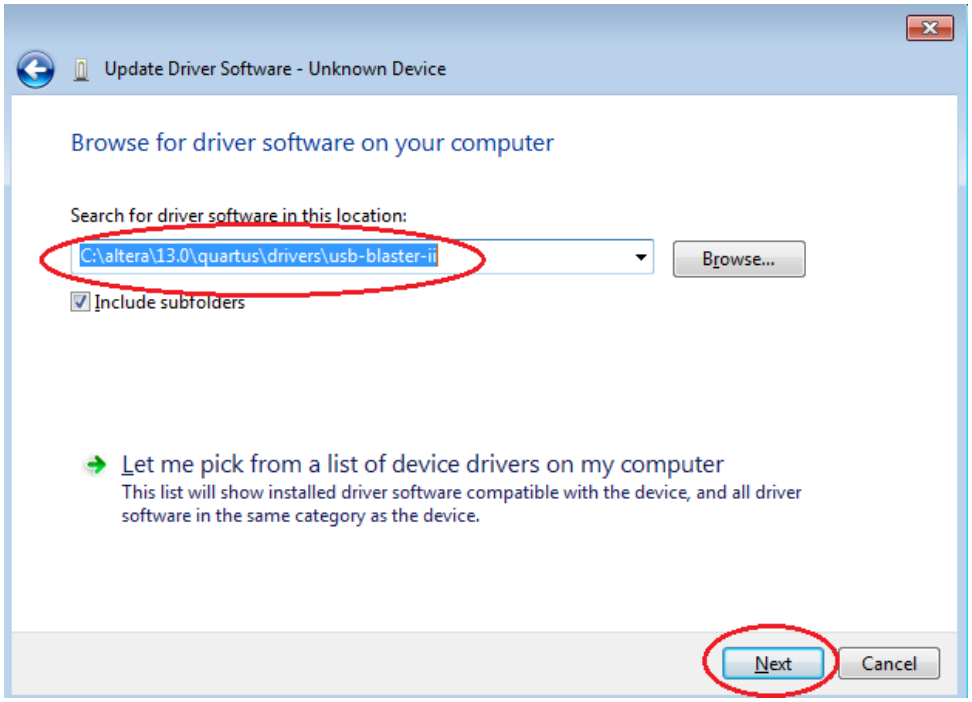
- Navigate to **Other Devices** in the Device Manager. Expand it to see **Unknown Device**
- Right click on **Unknown Device**. Select **Update Driver Software**.



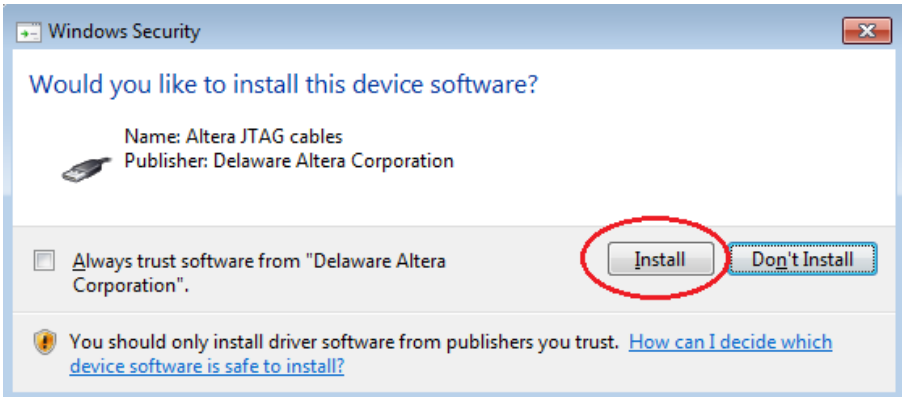
- Select **Browse my computer for driver software**.



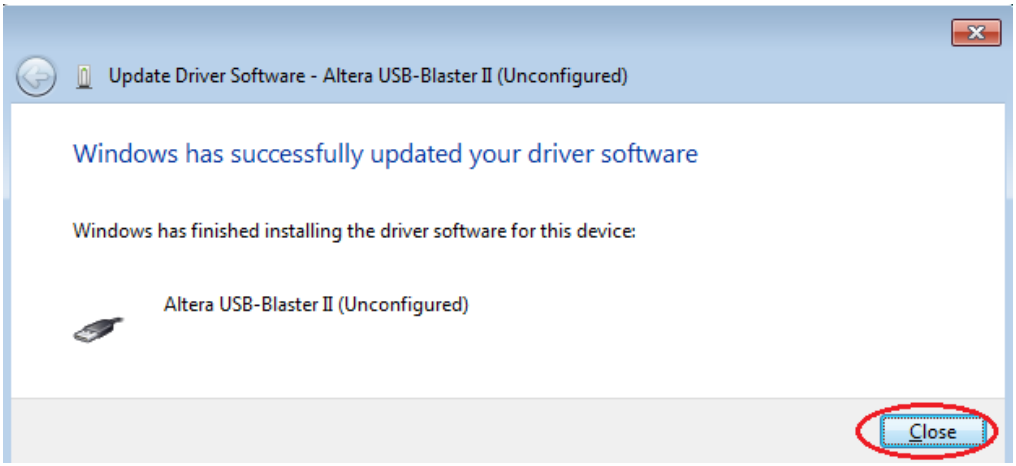
- Specify the driver software **location**. Press **Next**.



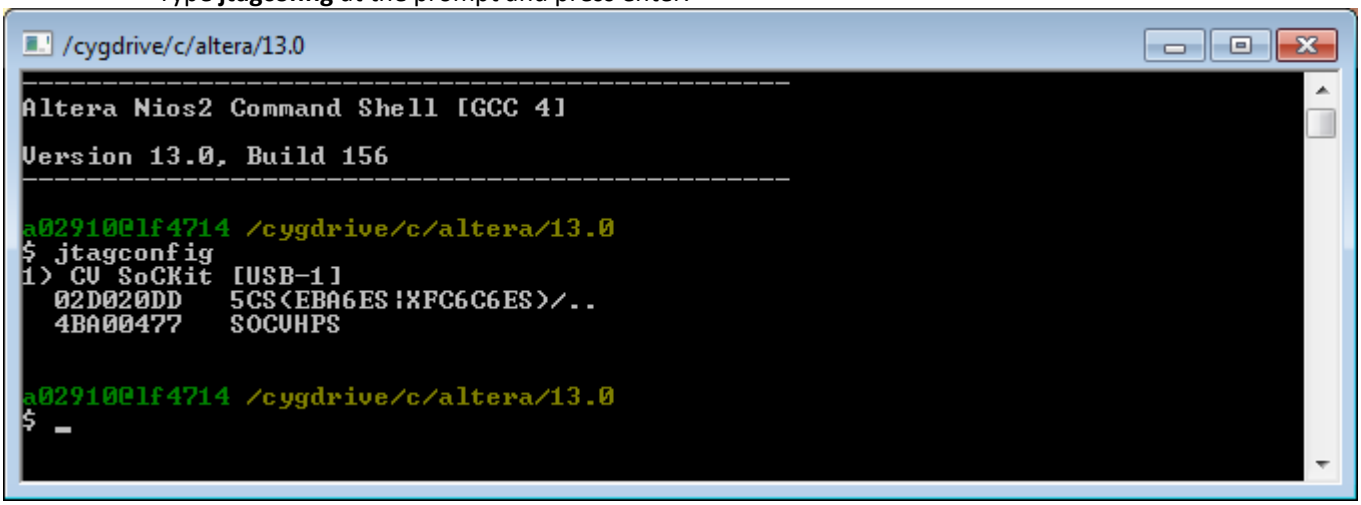
- Click Install on the next Screen



- **Wait** for the driver to **complete** its **installation**. Press Close. **Notice** that **device** is considered to be **Unconfigured**.



- Open a **NIOS II 13.0 Command** Shell, select: **Start -> Altera13.0.0.156 Web Edition** -> **NIOS II EDS 13.0.0.156** -> **NIOS II 13.0 Command Shell**
- Type **jtagconfig** at the prompt and press enter.



CONGRATULATIONS!!

You have just completed all the setup and installation requirements and are now ready to examine the system-level design.

MODULE 2. Examine the System Design

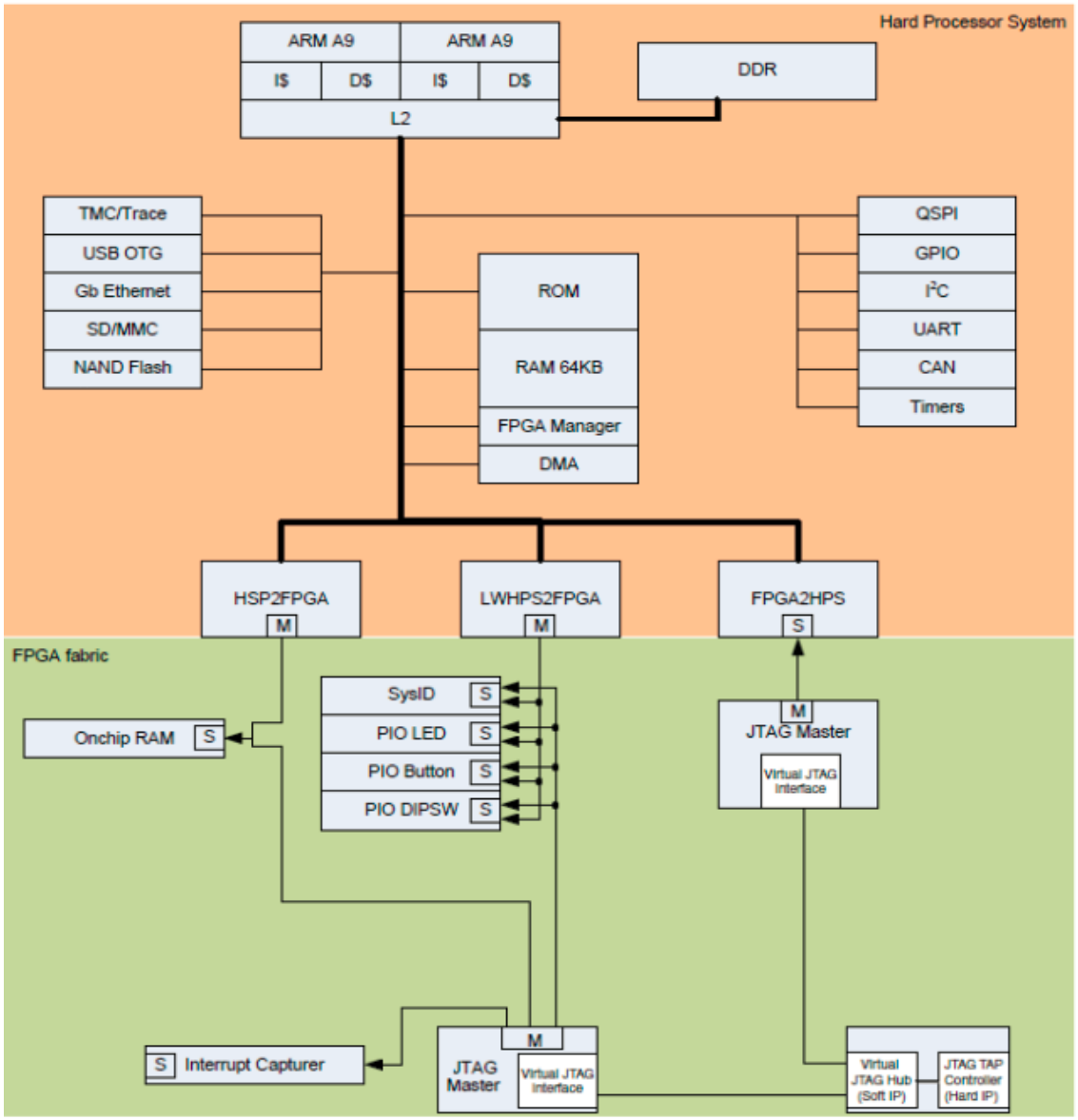
Module Objective

In this module you will review the architecture of the design that will be created in Qsys. You will also examine the layout of the SoCKit. Developing software for an Altera SoC requires an understanding of the design flow of the Qsys system integration tool. Typically, a design starts with system requirements. These system requirements become inputs to the system definition. System definition is then first step for implementation in the design flow process.

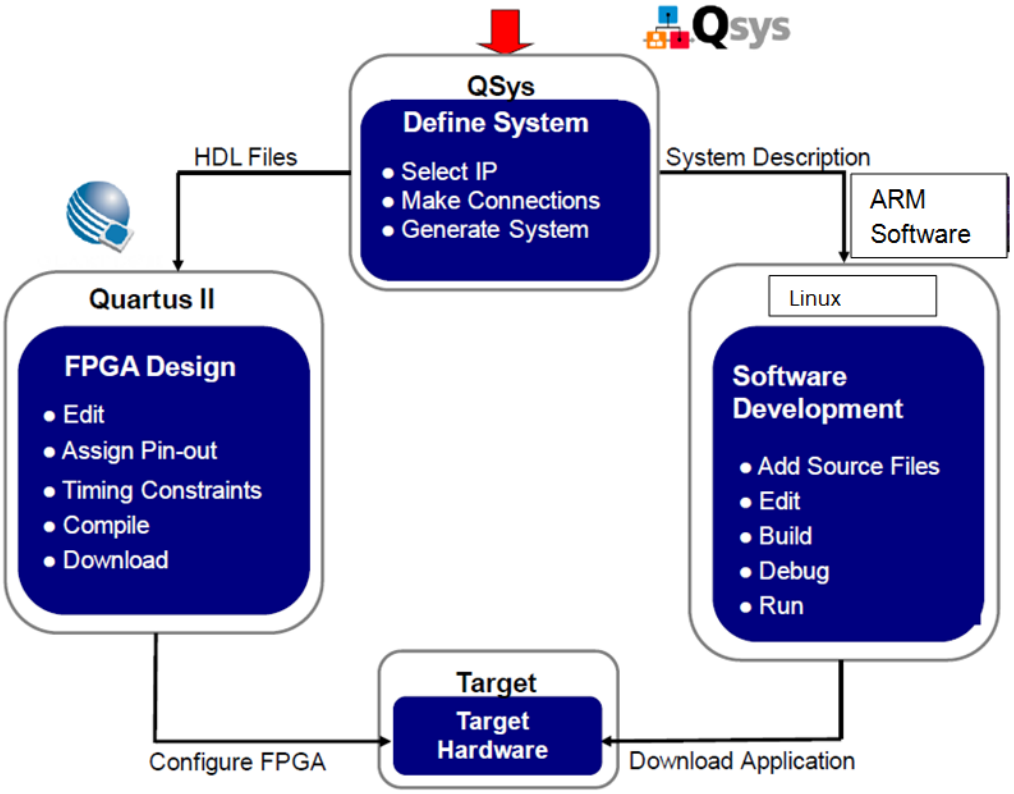
2.1 System Architecture

There are many components on the SoCKit that can be used, including the LCD, flash, Audio DACs, and IR.

The system that we will finish creating in Qsys is built by using a standard library of re-useable IP blocks. The orange section of this diagram is the HPS section, while the green section is the FPGA section. The HPS section was configured in the HPS component in Qsys. There are three bridges between the HPS and FPGA sections. You will focus on the LWHPS2FPGA bridge connected peripherals for this lab, specifically the LED PIO. The LED PIO is mapped through the bridge into the HPS addressable map.

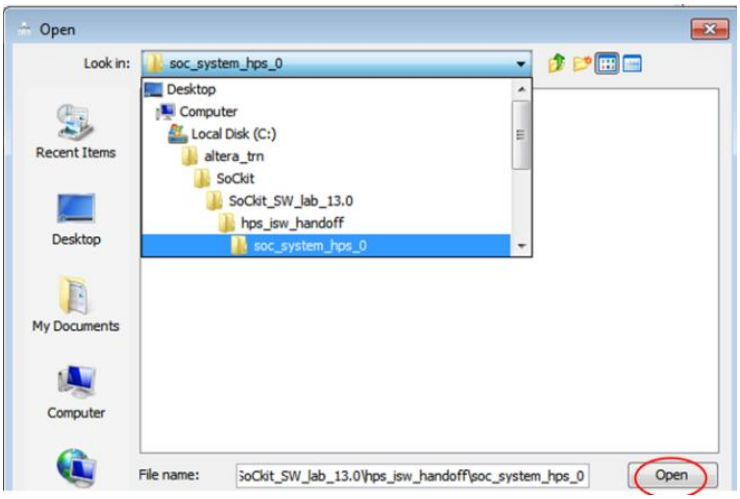


2.2 Examine the System Tool Flow



The **diagram** above depicts the **typical** flow for an SoC design. Qsys and Quartus II **generate** the following **sets** of **files**:

- A set of **XML files** are created that define the system description. These XML files are **utilized by the ARM DS-5** software tool to create a project for the **software application**. You can find the files here:

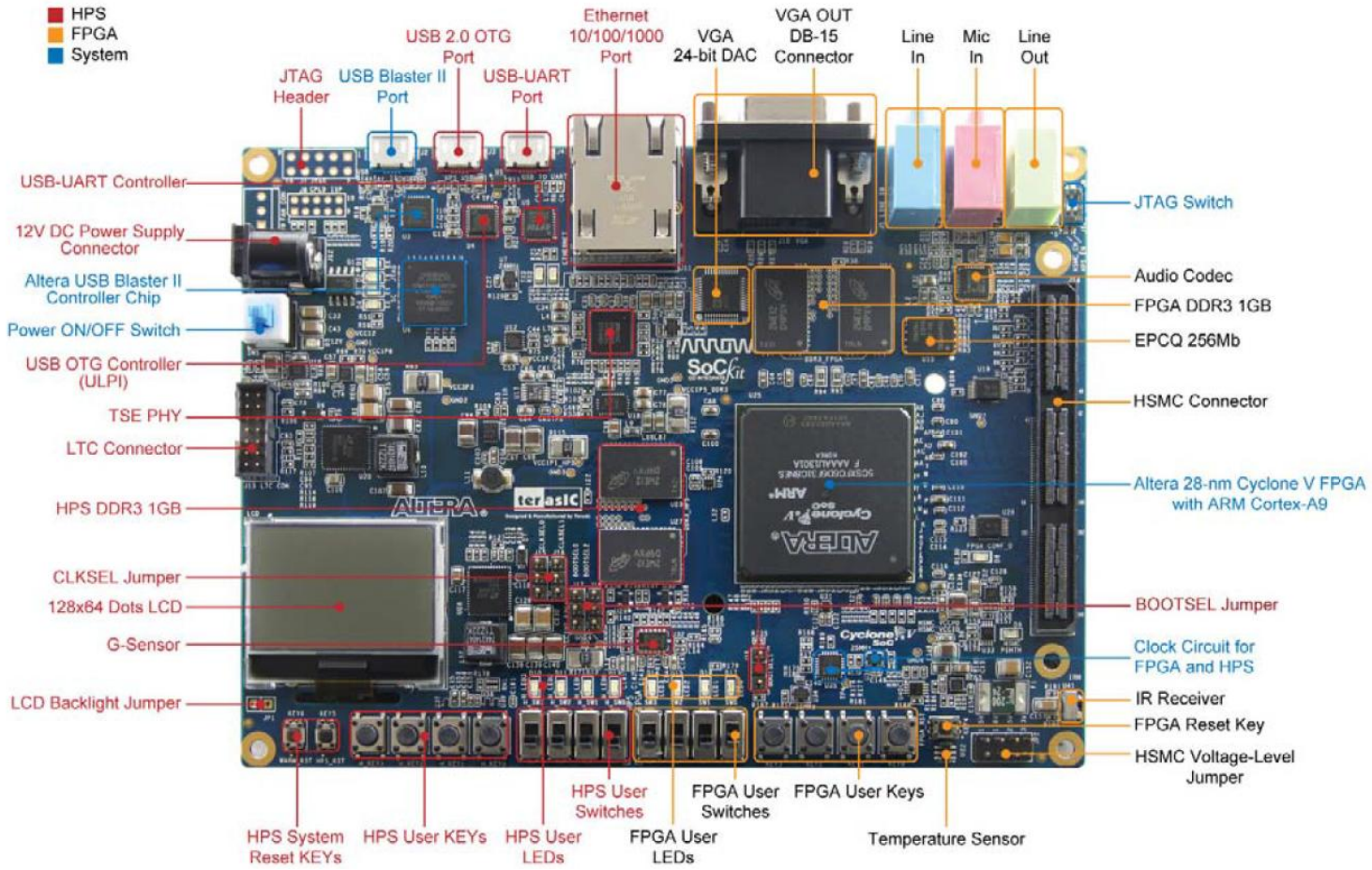


- Qsys also generates the HDL files (Verilog or VHDL) for the defined system. These HDL files are then used by Quartus II to compile and generate a set of files that defines the hardware system. This set of files includes the HDL files, Tcl (Tool Command Language) files that define dedicated pin locations for selected HPS peripherals, Tcl files that define the Multiport Memory Controller in the HPS & FPGA, QIP files that include: selected IP and SDC (Synopsis Design Constraint files) utilized by TimeQuest to constrain the complete system design.

- Quartus II will then generate an simple SOF (SRAM Object File) image that is used to configure the FPGA.

2.3 Examine Arrow's Cyclone V SoCKit

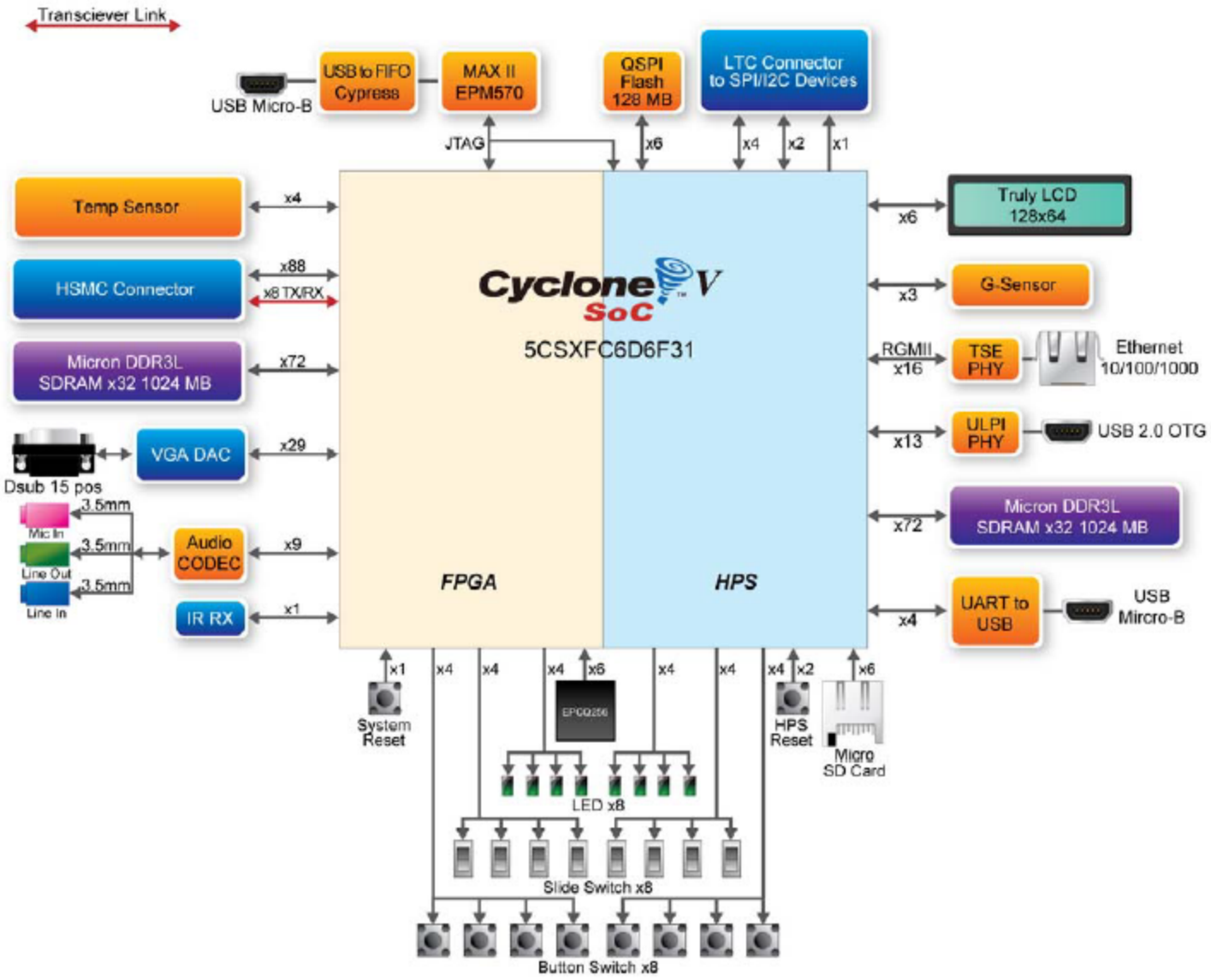
Examine the components on the Cyclone V SoC board hardware:



2.4 Block Diagram of the SoCKit

There are many components included on the SoCKit, including the LCD, flash, Audio DACs, and IR.

An block diagram of the board:



CONGRATULATIONS!!

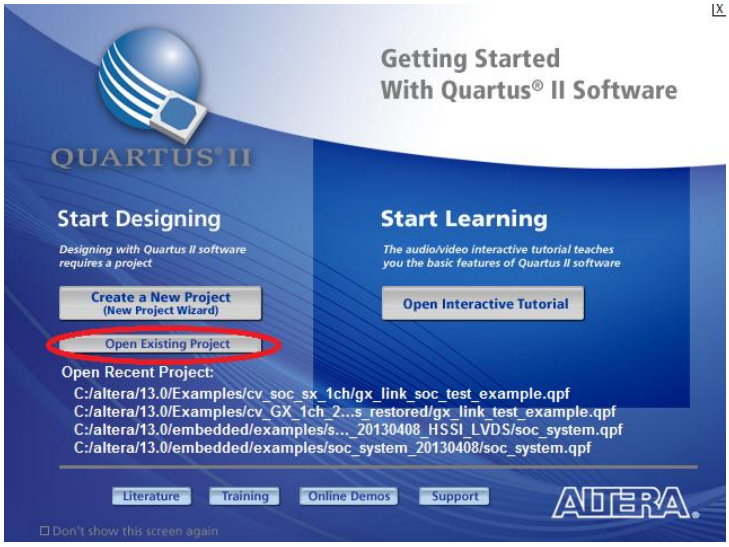
You have just completed the examination of the system-level design

MODULE 3. Set up the Quartus II project

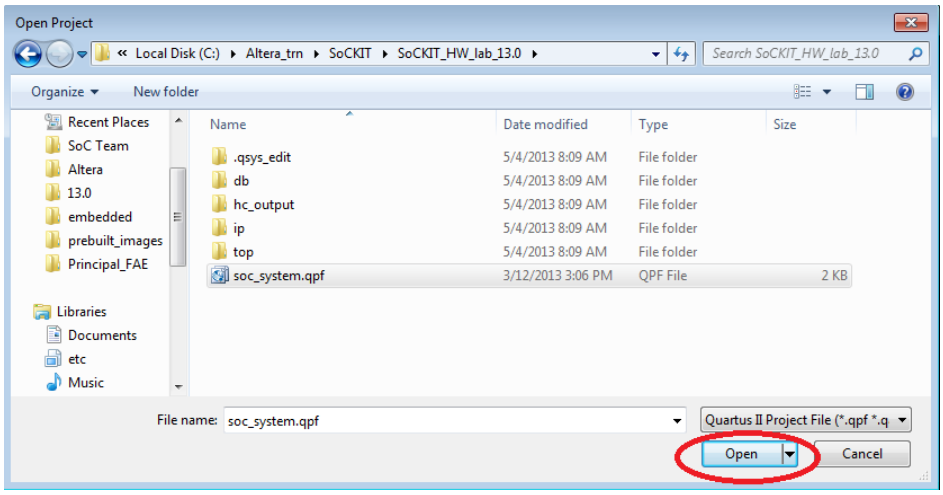
In this section, you will open a Quartus II project that contains the Qsys system. In addition, you will specify I/O constraints and settings for this design by executing a Tcl script.

3.1 Launch Quartus II Project

- Launch the **Quartus II v13.0** software from **Start** -> **All Programs** -> **Altera**
- A splash screen will appear, select **Open Existing Project**:



- Now browse to the directory: **C:\Altera_trn\SoCKit\SoCKit_HW_lab_13.0** and select **soc_system.qpf** and then select **Open.**

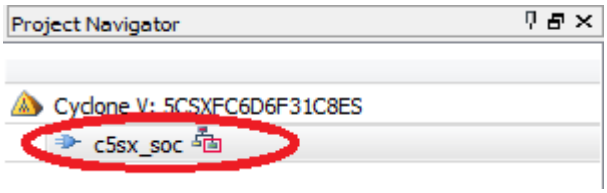


If you close the splash screen without opening the project:

- Select **File** -> **Open Project** and browse to the directory **c:\altera_trn\SoCKit\SoCKit_HW_lab_13.0**.
- Select **soc_system.qpf**.

- The Quartus II project will open. The project already contains a top level Verilog file (..**\top\c5sx_soc.v**) and a Qsys project (**soc_system.qsys**) that will be modified in the following modules.

- Please take a look at the top level file. To do this, **double click** on the **c5sx_soc** icon in the Project navigator Window or (select: **File** -> **Open** and browse to the **..\top** directory and open **c5sx_soc.v**)



- The **c5sx_soc.v** contains all of the I/O for the HPS instance as well as all the FPGA I/O. **In addition**, you will find the **instance** for the **Qsys component**, **soc_system** at the end of the file.

CONGRATULATIONS!!

Your Quartus II project is set up. You are ready to start building your Qsys system.

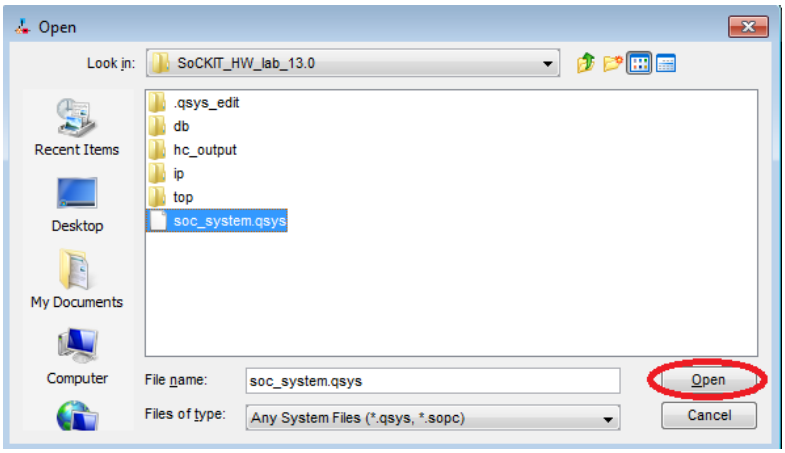
MODULE 4. Build the Qsys System

Module Objective

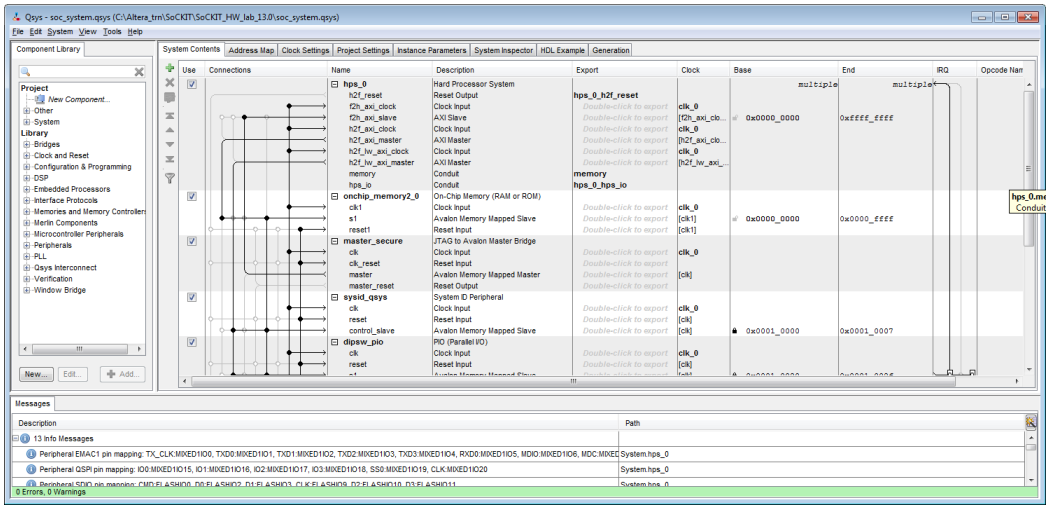
In this module you add the standard and custom components to the system, make connections where required, assign the clocks, set arbitration priorities and generate the system.

4.1 Launch Qsys

- From the **Tools** menu, select "**Qsys**". There may be a slight delay while the Qsys application launches.
- Open the File named **soc_system.qsys**



There will be various components that are already included in the Qsys system, while others will need to be built.

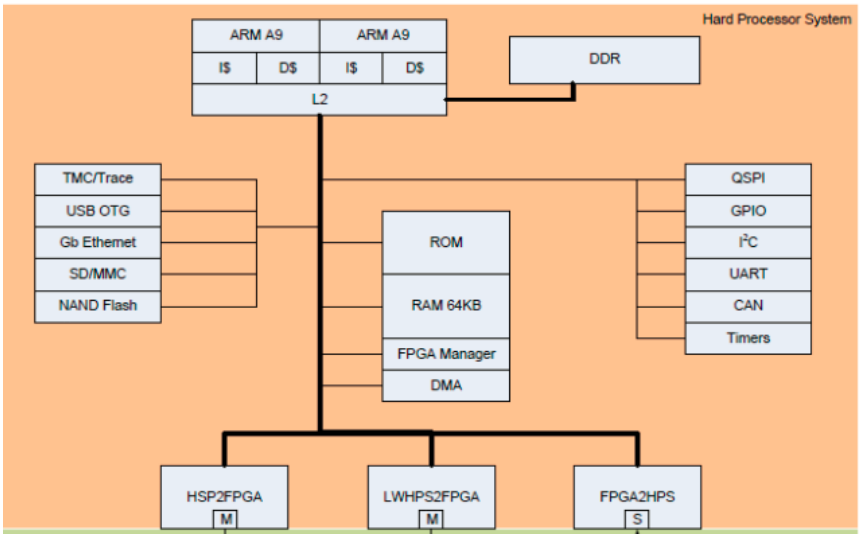


4.2 Build the Qsys System

The first component that you will **verify and change** is the HPS (**Hard Processor System**).

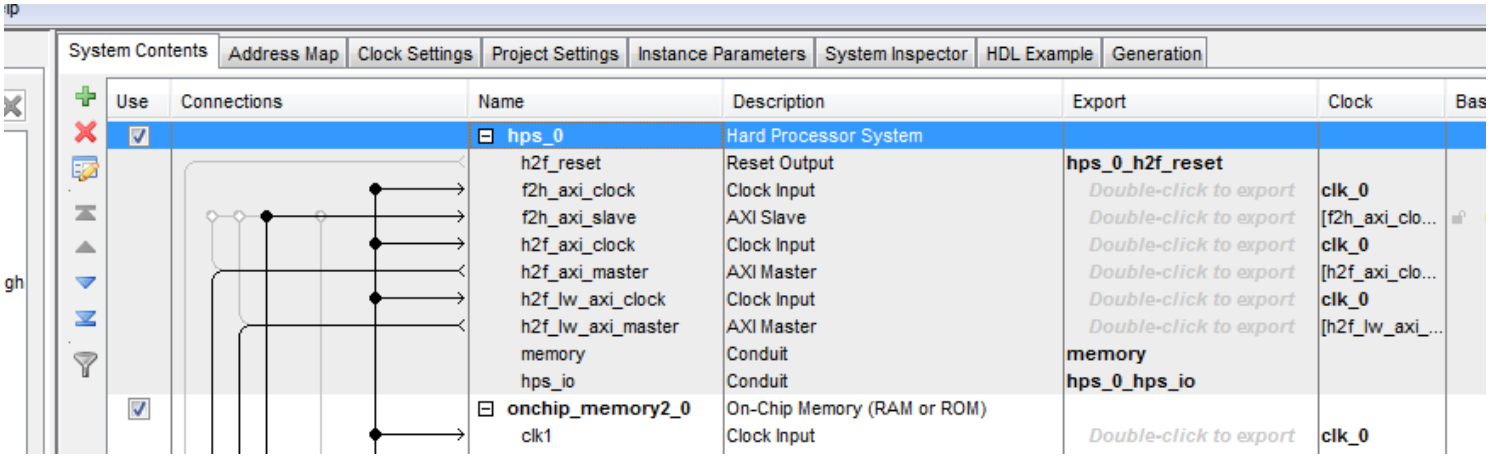
Verify the Hard Processor System

The Hard Processor System (HPS) consists of the **dual ARM Cortex A9** with various **peripherals enabled**. The following is a **block diagram** of some of the entities available in the HPS.



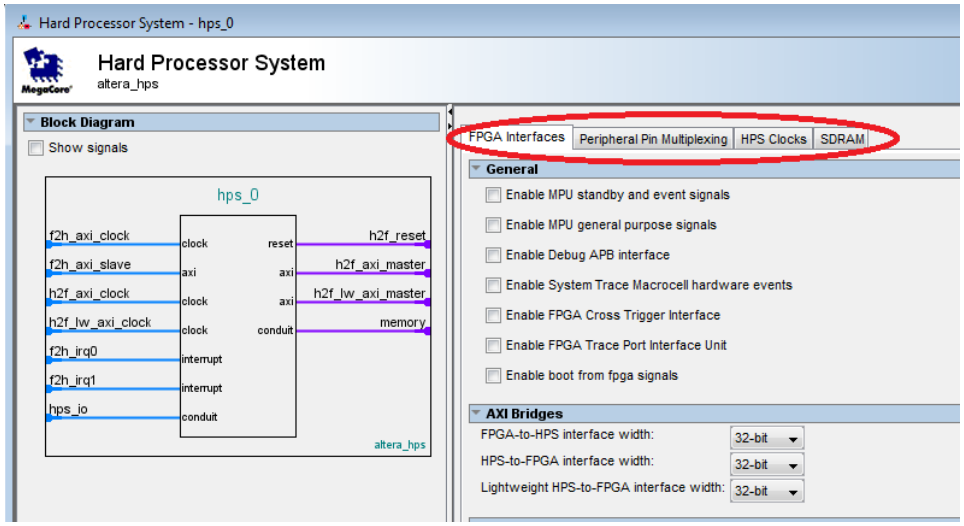
Configure the HPS

- In the Qsys window, **double-click** on the hps_0 component



- We will now configure the HPS with the correct pin multiplexing for the peripherals to accommodate the interfaces on the SoCKit. This process will also include configuring the clocks, the Multiport Memory Controller, the three high speed ports: HPS to the FPGA, FPGA to HPS and FPGA to Multiport Memory Controller and various other settings. For complete details please refer to the Cyclone V Device Handbook, Volume 3: Hard Processor System Technical Reference Manual

Please note that there are multiple tabs for: **FPGA Interfaces**, **Peripheral Pin Multiplexing**, **HPS Clocks** and under the **SDRAM** tab there are sub-tabs that are used to configure the HPS.



4.2.1 Configure the FPGA Interfaces for the HPS

Under the **FPGA Interfaces** tab, there are various options in the **General**, **AXI Bridges**, **FPGA to HPS SDRAM**, **Resets**, **DMA Peripheral Request**, and **Interrupts** sections.

Verify that the **MPU standby and event signals** are **disabled**. These signals are utilized to indicate if the microprocessor is in standby mode to the FPGA and can wake up an MPCore processor from a wait for event (WFE) state.

Verify that **Enable MPU general purpose signals** are **disabled.** This is used to enable a pair of 32-bit unidirectional general purpose interfaces between the FPGA and the **FPGA Manager** in the HPS.

- The FPGA Manager offers the following: "Used to configure the FPGA, Mimics passive parallel 32-bit configuration, Partial Reconfiguration, Compressed FPGA configuration images, AES encrypted configuration images, Monitors the configuration-related signals, provides 32 general purpose inputs and 32 general-purpose outputs to the FPGA".

Ensure that the **"Enable Debug APB interface" , "Enable System Trace Macrocell Hardware events", "Enable FPGA Cross Trigger Interface",** and **"Enable FPGA Trace Port Interface Unit"** are **disabled**.

Explanation of terms:

- **Debug APB interface** - Enables debug interface to the FPGA, allowing access to debug components in the HPS.
- **System Trace Macrocell hardware events** - Enables System Trace Macrocell (STM) hardware events, allowing logic inside the FPGA to insert messages into the trace stream.
- **Enable FPGA Cross Trigger interface** - Enables the cross trigger interface (CTI), which allows trigger sources and sinks to interface with the embedded cross trigger (ECT).
- **Enable FPGA Trace Port Interface Unit** - Enables an interface between the trace port interface unit (TPIU) and logic in the FPGA. The TPIU is a bridge between on-chip trace sources and a trace port.

Ensure that the **"Enable boot from fpga signals"** is **disabled**

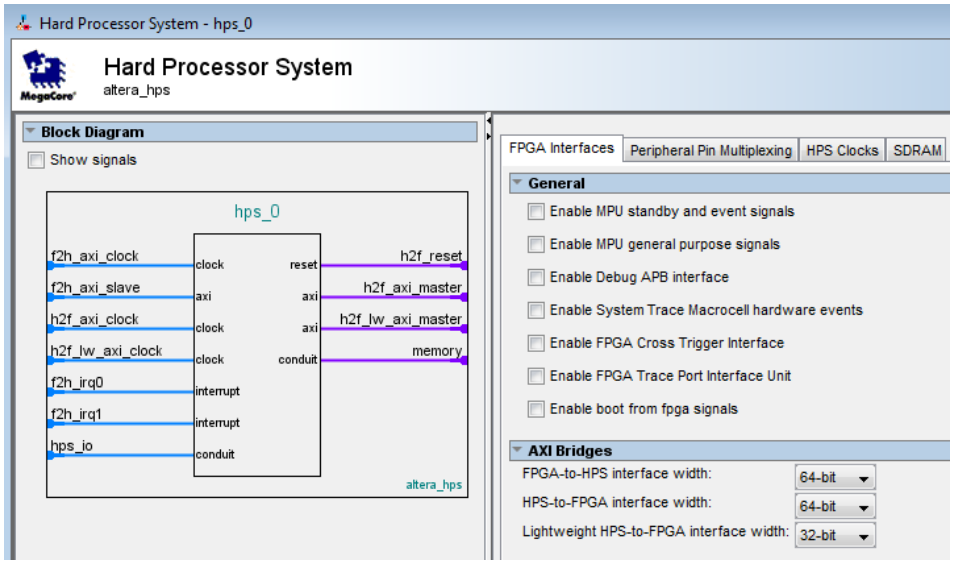
- **Enable boot from FPGA ready** - Enables an input to the HPS indicating whether a preloader is available in on-chip RAM. If the input is asserted, a preloader image is ready at memory location 0.
- **Enable boot from FPGA on failure** - Enables an input to the HPS indicating whether a fallback preloader is available in on-chip RAM. If the input is asserted, a fallback preloader image is ready at memory location 0. The fallback preloader is to be used only if the HPS boot ROM does not find a valid preloader image in the selected flash memory device.

In the **AXI Bridge** section, ensure both the **"FPGA-to-HPS interface width"** is set to **64-bit and "HPS-to-FPGA interface width"** is set to **64-bit.** Both of these interfaces can be set to 32, 64, 128-bits or Unused.

- Enabling the FPGA to HPS interface allows masters within the FPGA to access to HPS peripherals.
- Enabling the HPS to FPGA interfaces allows HPS masters to access the FPGA peripherals.

Verify that the "enable the lightweight HPS-to -FPGA interface width" is to be set to 32 bit. A 32 bit AXI interface optimized for low latency is thus enabled.

After making these changes, the HPS should now look like this:



Scrolling down the FPGA interface tab, there are more options. There are sections for the **FPGA-to-HPS SDRAM Interface**, **Resets** and **DMA Peripheral Request**.

Verify the **FPGA to HPS SDRAM Interface** section **is left blank.**

- This enables the FPGA to gain access to the HPS SDRAM subsystem from the FPGA fabric by providing up to six ports to the SDRAM. The bus interface provided to the FPGA can be AXI-3 or Avalon Memory Mapped.

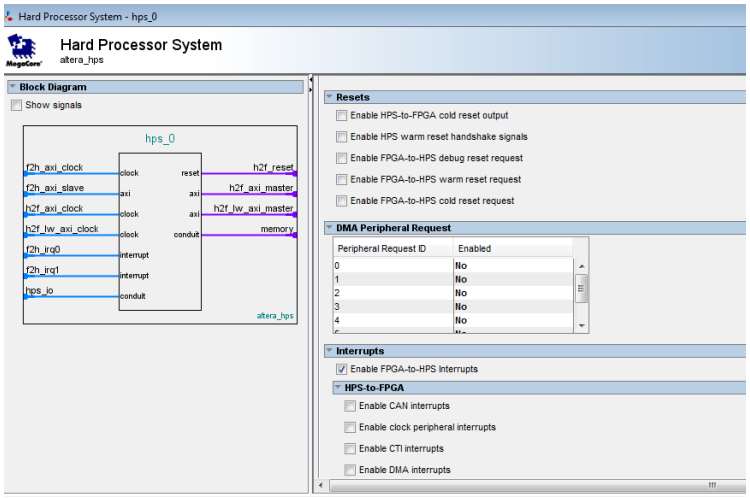
Verify the **Resets** are set to **disabled** In the **Reset section**. There are several options and the Reset Manager is very sophisticated. Therefore, please refer to the Cyclone V Device Handbook Volume 3: Hard Processor System Reference Manual: Section I, Chapter 3. for details on the Reset Manager. The bullets below are descriptive. Do NOT implement these changes.

- **Enable HPS-to-FPGA cold reset output** - Enable interface for HPS-to-FPGA cold reset output
- **Enable HPS warm reset handshake signals** - Enable an additional pair of reset handshake signals allowing soft logic to notify the HPS when it is safe to initiate a warm reset in the FPGA fabric.
- **Enable FPGA-to-HPS debug reset request** - Enable interface for FPGA-to-HPS debug reset request
- **Enable FPGA-to-HPS warm reset request** - Enable interface for FPGA-to-HPS warm reset request
- **Enable FPGA-to-HPS cold reset request** - Enable interface for FPGA-to-HPS cold reset request

Verify the **DMA peripheral request** is set to the **default** of **No** for each of the eight channels. The designer can enable each direct memory access (DMA) controller peripheral request ID individually. Each request ID enables an interface for FPGA soft logic to request one of eight logical DMA channels to the FPGA.

Ensure that the "Enable FPGA-to-HPS interrupts" is **enabled.** This enables interrupt signals from the FPGA to the MPU in the HPS .
All of the "HPS to FPGA" interrupts should all be disabled. An interrupt signal can be provided to the FPGA for each one of the peripherals that are included in this section.

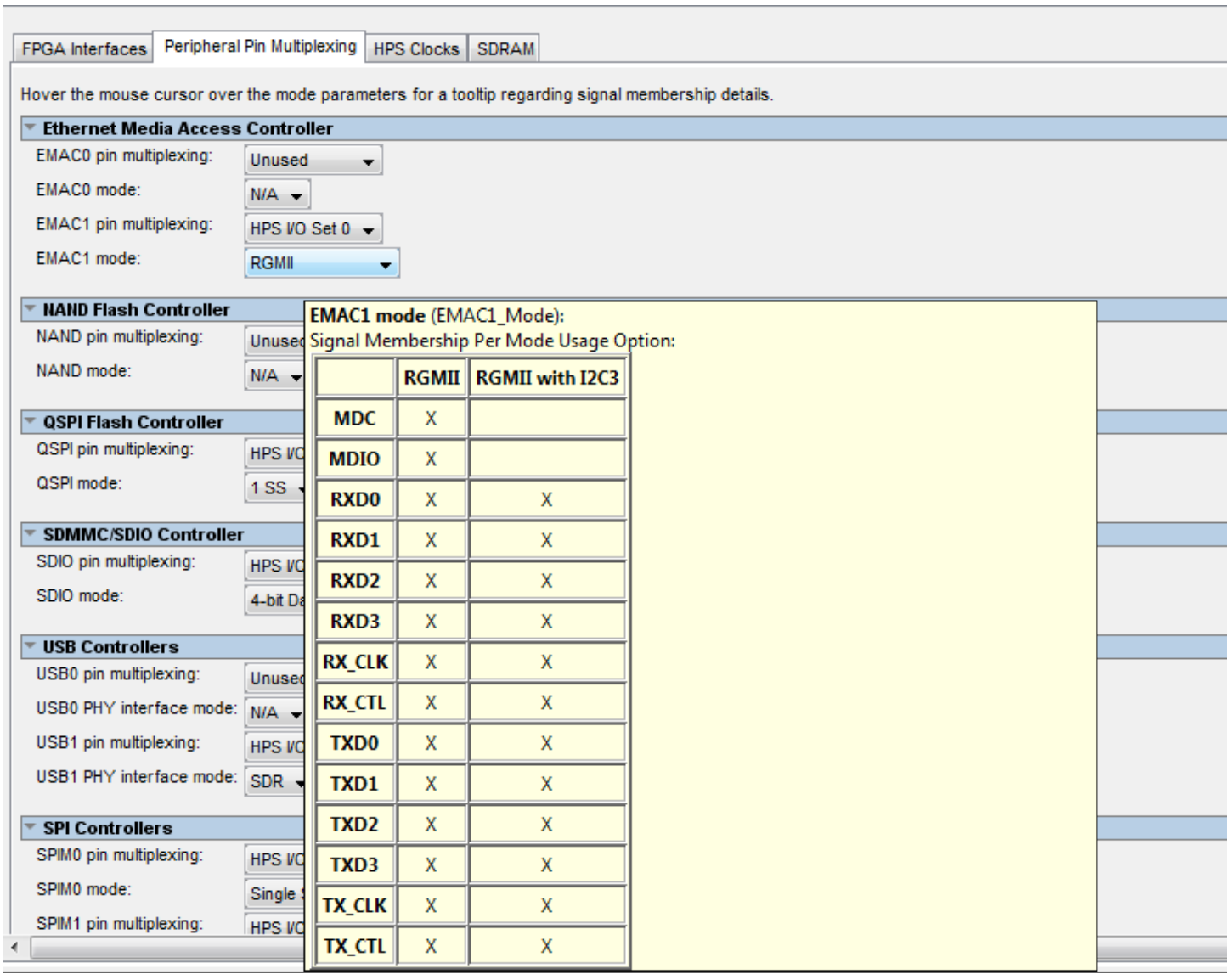
After making these changes, the HPS should now look like this:



4.2.2 Configure HPS' Peripheral Pin Multiplexing (MAC, NAND, QSPI, SDIO, USB)

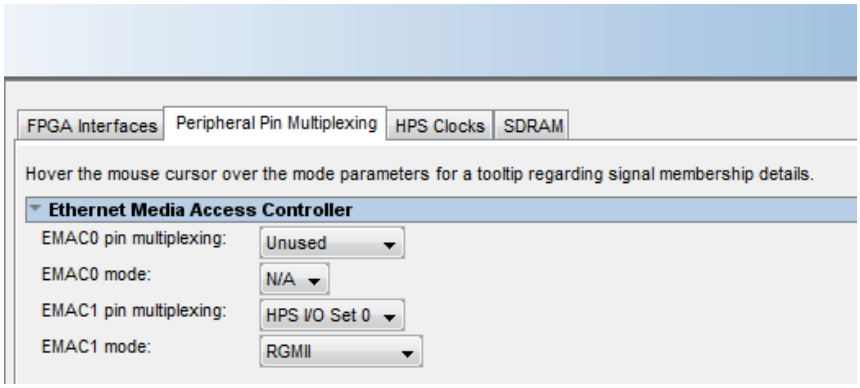
Under the **Peripheral Pin Multiplexing** tab, there are options to enable the HPS peripherals. The peripherals in the HPS are available to as many as three sets of HPS I/O pins. An **HPS pin multiplexing spreadsheet** is available to make this a simpler and more intuitive task for the designer. (*Please note that all HPS peripherals are also now available for pin out via the FPGAs pins.)*

By hovering over a particular peripherals **mode:,** a list of the signal to pin for a particular I/O set per pin is displayed in a pop-up box.

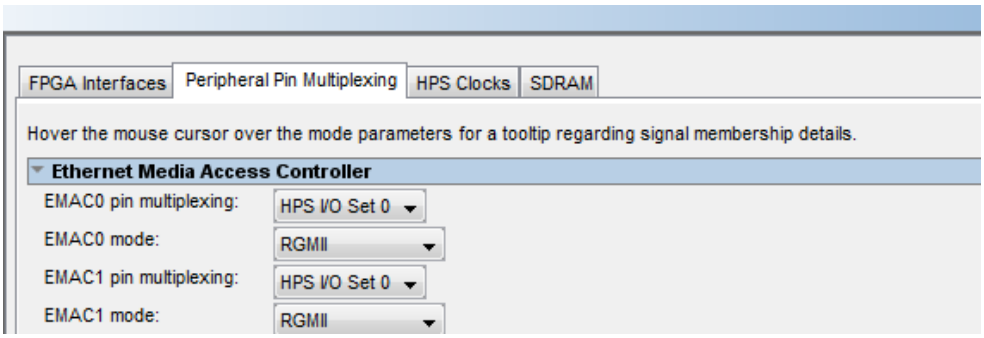


To see a pin multiplexing error, change the EMAC0 pin multiplexing which is default set to Unused to be HPS I/O Set 0.

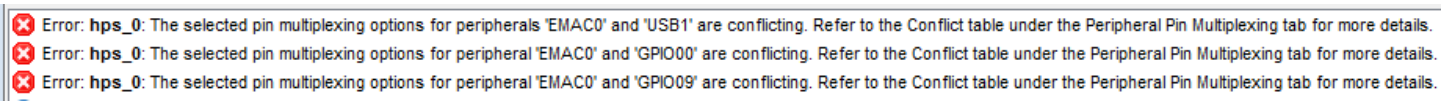
Change the default setting in this archive from



to be

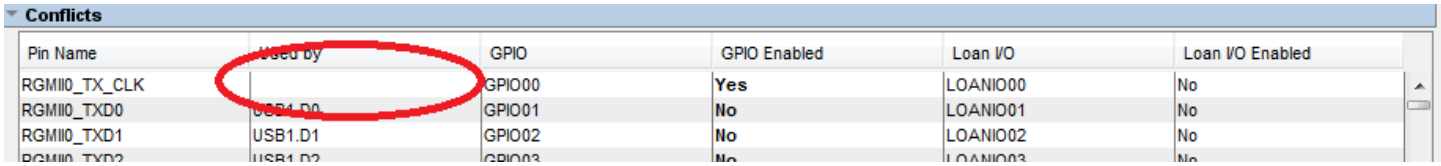


There will be errors at the bottom of the screen:

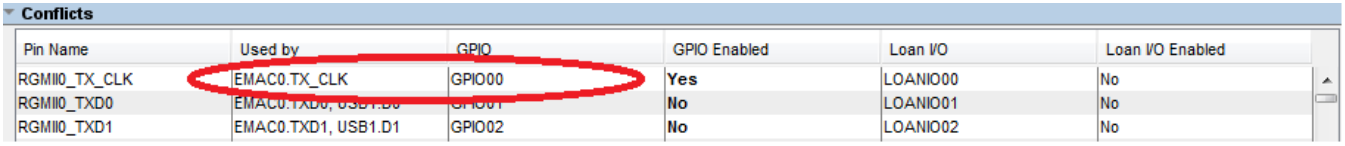


These errors occur since each peripheral available on the HPS go to at least one set of HPS I/O. The multiplexing for each I/O Set is controlled by selecting the specific HPS peripheral's **I/O Set**. When the EMAC pin multiplexing was selected, the pins for several of the other peripherals had already used these I/O pins, and therefore a conflict. (*Again, an **HPS pin multiplexing spreadsheet** is available to make this a simpler and more intuitive task for the designer.) Please note that all pins that aren't **utilized** by an HPS peripheral can by **Enabled as a GPIO pin**.*

No Conflicts:

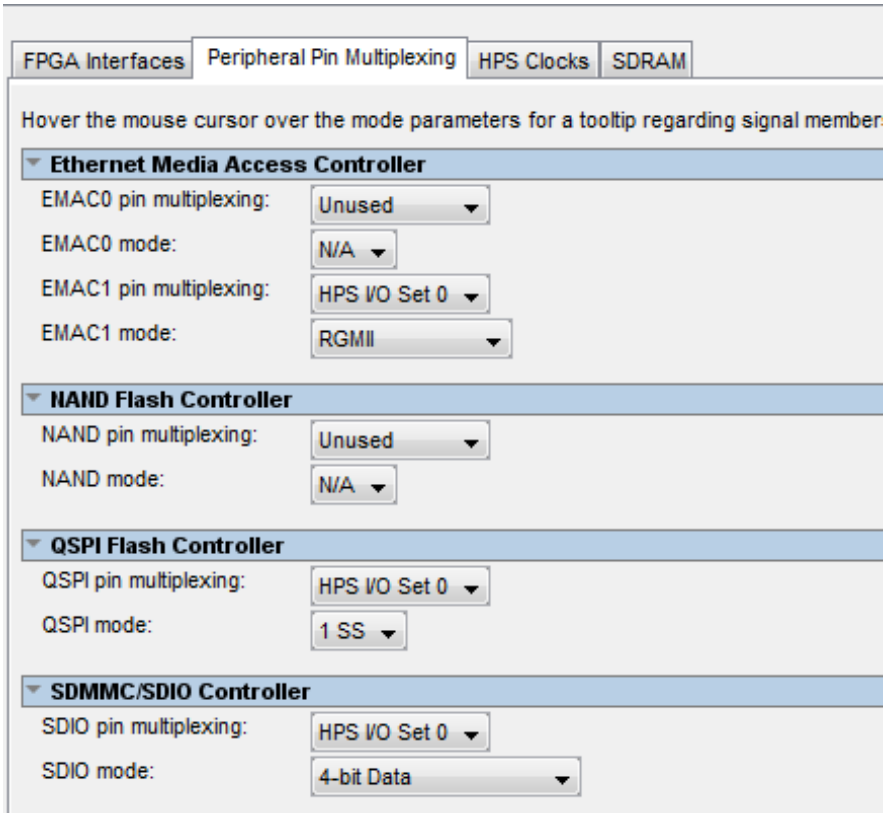


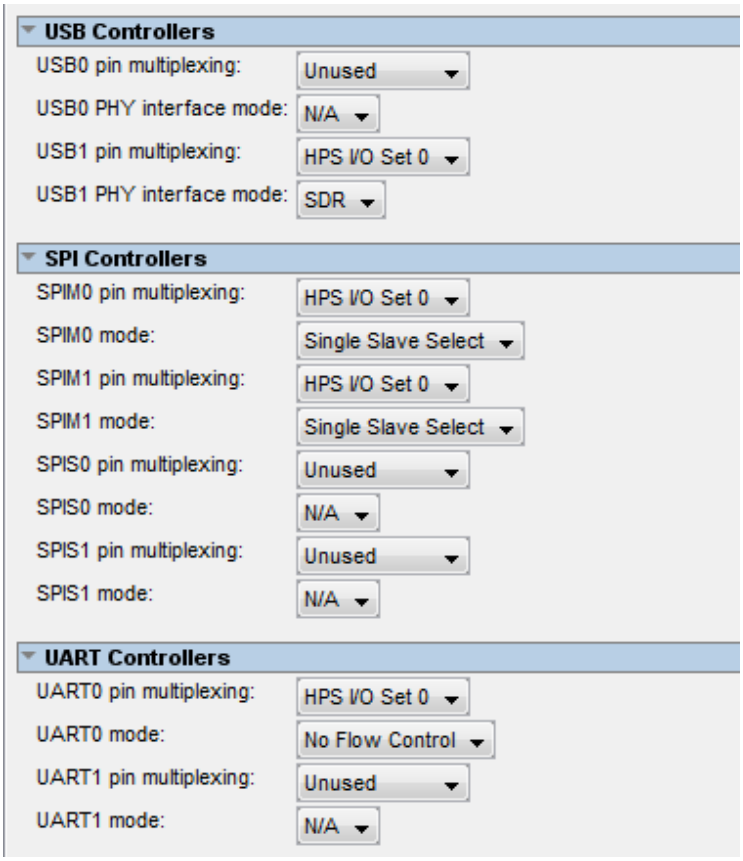
Conflicts:

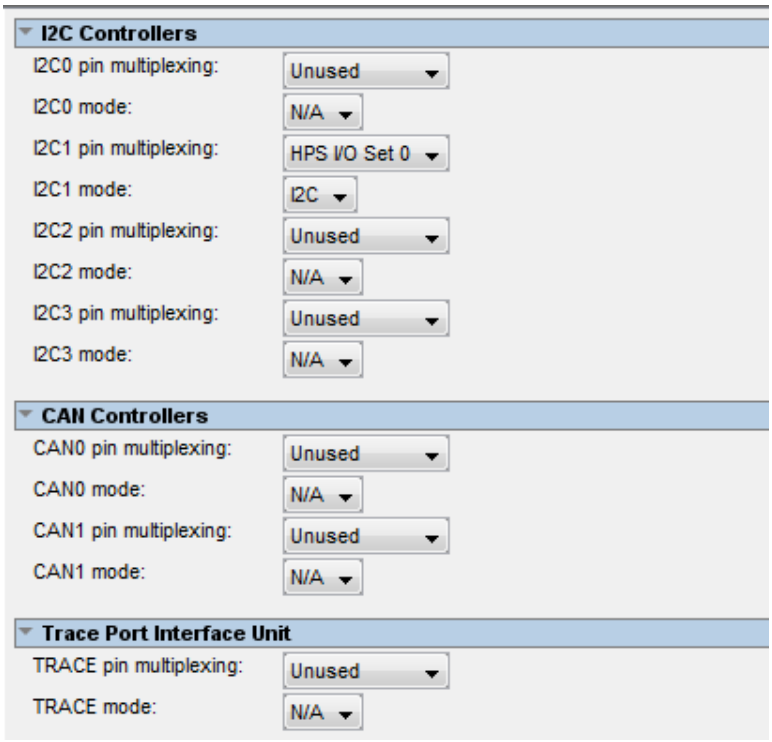


Change the EMAC0 pin multiplexing back to **Unused** to remove the errors**.**

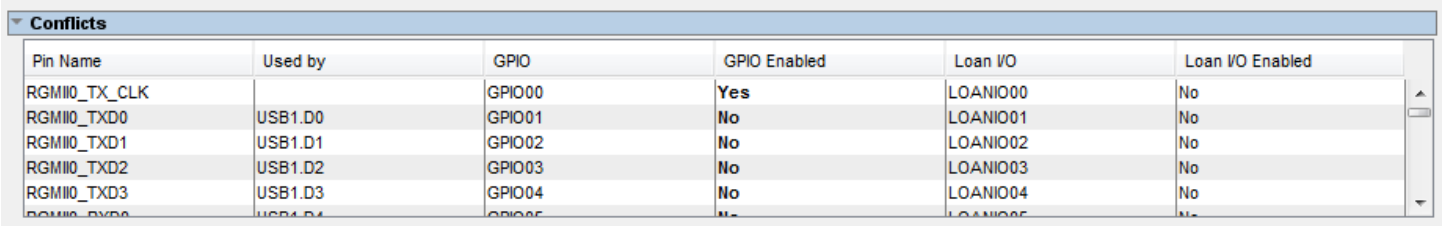
As the selected pins need to match the Cyclone V SoC board layout and they need to avoid errors, **verify that the Qsys settings match the following screenshots** to enable the Ethernet MAC, QSPI Flash Controller, SDMMC, USB Controller, SPI Controllers, Uart Controllers, and I2C Controllers.




USB Controllers
USB0 pin multiplexing:
Unused
USB0 PHY interface mode:
N/A
USB1 pin multiplexing:
HPS I/O Set 0
USB1 PHY interface mode:
SDR
SPI Controllers
SPIM0 pin multiplexing:
HPS I/O Set 0
SPIM0 mode:
Single Slave Select
SPIM1 pin multiplexing:
HPS I/O Set 0
SPIM1 mode:
Single Slave Select
SPIS0 pin multiplexing:
Unused
SPIS0 mode:
N/A
SPIS1 pin multiplexing:
Unused
SPIS1 mode:
N/A
UART Controllers
UART0 pin multiplexing:
HPS I/O Set 0
UART0 mode:
No Flow Control
UART1 pin multiplexing:
Unused
UART1 mode:
N/A


I2C Controllers
I2C0 pin multiplexing:
Unused
I2C0 mode:
N/A
I2C1 pin multiplexing:
HPS I/O Set 0
I2C1 mode:
I2C
I2C2 pin multiplexing:
Unused
I2C2 mode:
N/A
I2C3 pin multiplexing:
Unused
I2C3 mode:
N/A
CAN Controllers
CAN0 pin multiplexing:
Unused
CAN0 mode:
N/A
CAN1 pin multiplexing:
Unused
CAN1 mode:
N/A
Trace Port Interface Unit
TRACE pin multiplexing:
Unused
TRACE mode:
N/A

There should be **no errors** and the conflict setting should look as follows:

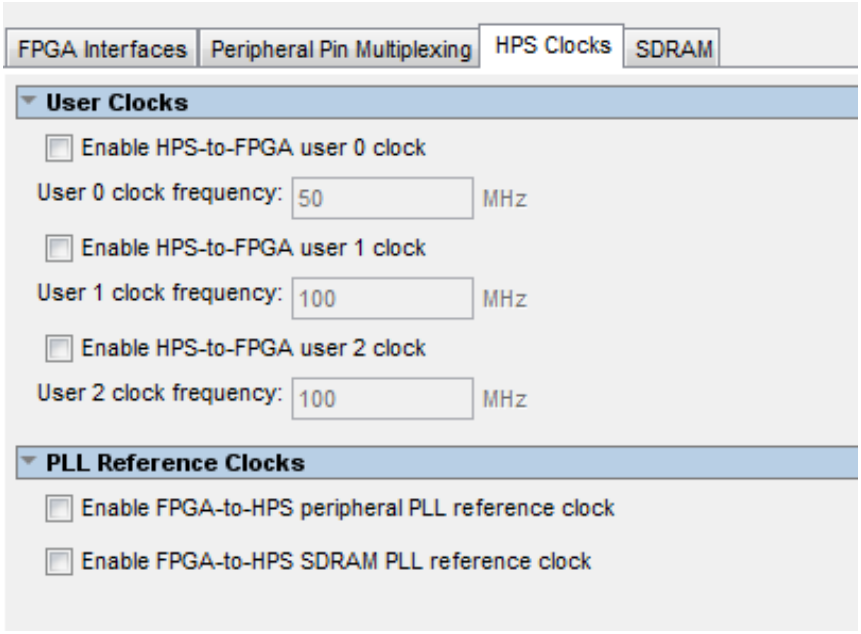


4.2.3 Configure HPS Clocks

Select the **HPS Clock**s tab, there are settings to enable clocks between the HPS to FPGA and FPGA to HPS.

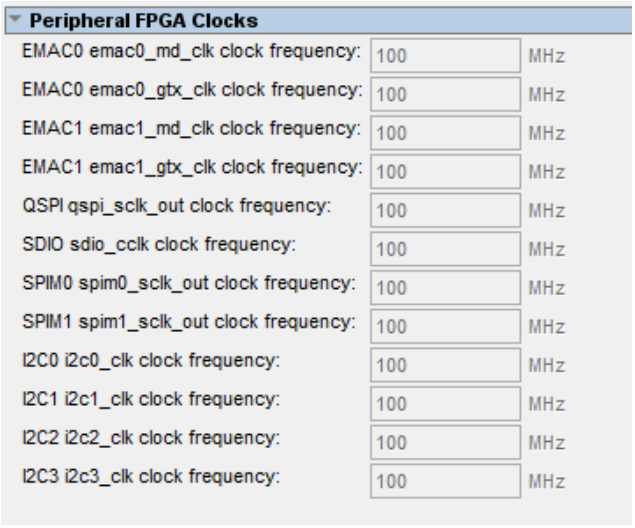
- Enable **HPS-to-FPGA user** 0, 1, 2 **clock** - Enable main PLL from HPS to FPGA
  User 0, 1, 2 **clock frequency** - Specify the maximum expected frequency for the main PLL

The PLL reference clocks between the HPs and FPGA are NOT enabled for this lab. Therefore, your screenshot should be as shown below:

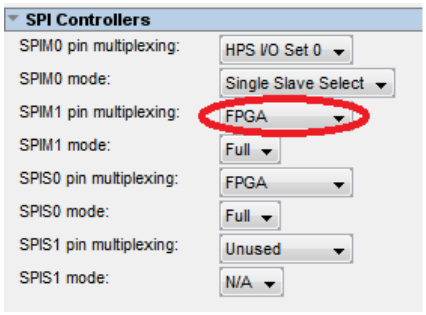


- **Enable FPGA-to-HPS peripheral PLL reference clock** - Enables the interface for FPGA fabric to supply reference clock to HPS peripheral PLL
- **Enable FPGA-to-HPS SDRAM PLL reference** clock - Enables the interface for FPGA fabric to supply reference clock to HPS SDRAM PLL

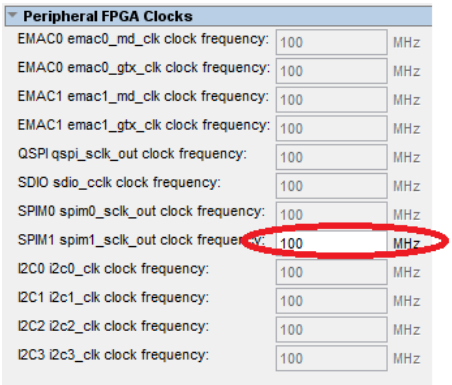
None of the HPS Peripherals were selected to be available in the FPGA; therefore, none of these clocks are **Peripheral FPGA Clocks** :



For example, if SPM1 were selected to be available in the FPGA,



Then, the result in the **HPS Clocks** tab would be:



4.2.4 Configure SDRAM (The HPS External Memory Interface)

Please note that Altera has an 8 Hour class available for Implementing, Simulating, and Debugging External Memory Interfaces and this resource should be utilized for an in depth understanding of EMIFs.

Under the **SDRAM** tab, there are options to set the SDRAM parameters for the HPS External Memory Interface. The SoCKit has two Micron 1.35V DDR3L SDRAM devices connected to the HPS (256Mb x 16 x 2 = 1GB at 1.5V vs. 1.35v).

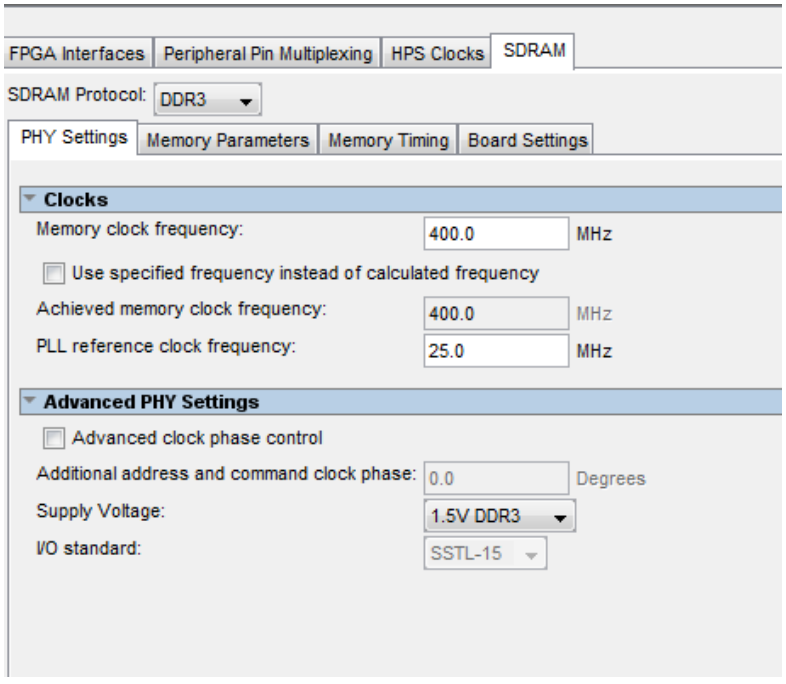
There are four tabs for the SDRAM configuration: **PHY Settings**, **Memory Parameters**, **Memory Timing**, and **Board Settings**.

Select the PHY Settings, the clock and Advanced PHY Settings are required.

Change the memory clock frequency from 300 MHz to 400 MHz. This is the rate for the Micron memory devices.

Verify that the supply voltage is set to be **1.5V V DDR3.** The 1.35 V variation of this Micron device is used; but the Vdd/Vddq are connected to 1.5v in order to reduce the system power supply complexity for SoCKit.

The settings should now look like:



Select the **Memory Parameters** tab. The settings are needed to match the DDR3 device. A table from the Micron datasheet shows the row address, bank address and column address. The Micron memory used on this board has the parameters in the last column (256 Meg x 16).

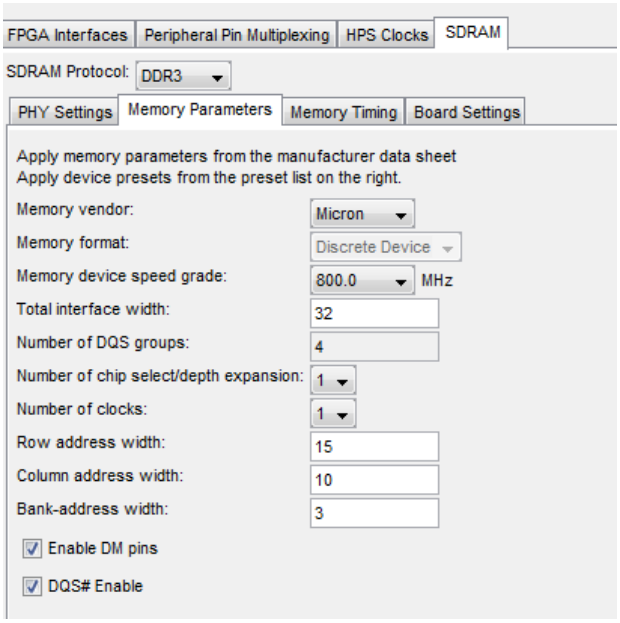
4Gb: x4, x8, x16 DDR3L SDRAM
Description

Table 2: Addressing

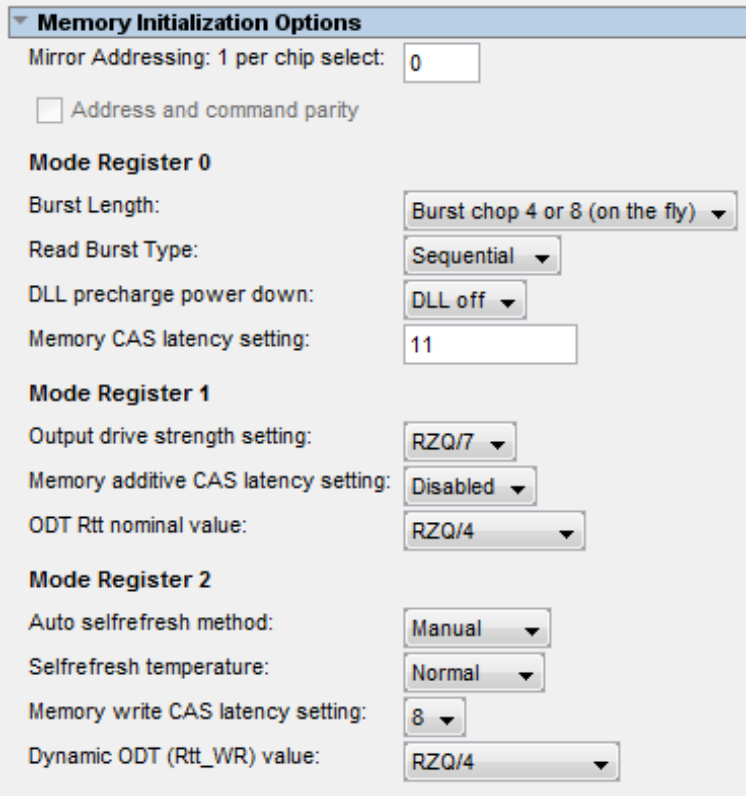
Parameter	1 Gig x 4	512 Meg x 8	256 Meg x 16
Configuration	128 Meg x 4 x 8 banks	64 Meg x 8 x 8 banks	32 Meg x 16 x 8 banks
Refresh count	8K	8K	8K
Row address	64K (A[15:0])	64K (A[15:0])	32K (A[14:0])
Bank address	8 (BA[2:0])	8 (BA[2:0])	8 (BA[2:0])
Column address	2K (A[11, 9:0])	1K (A[9:0])	1K (A[9:0])
Page size	1KB	1KB	2KB

The datasheet also shows that the DM and DQS# pins are enabled.

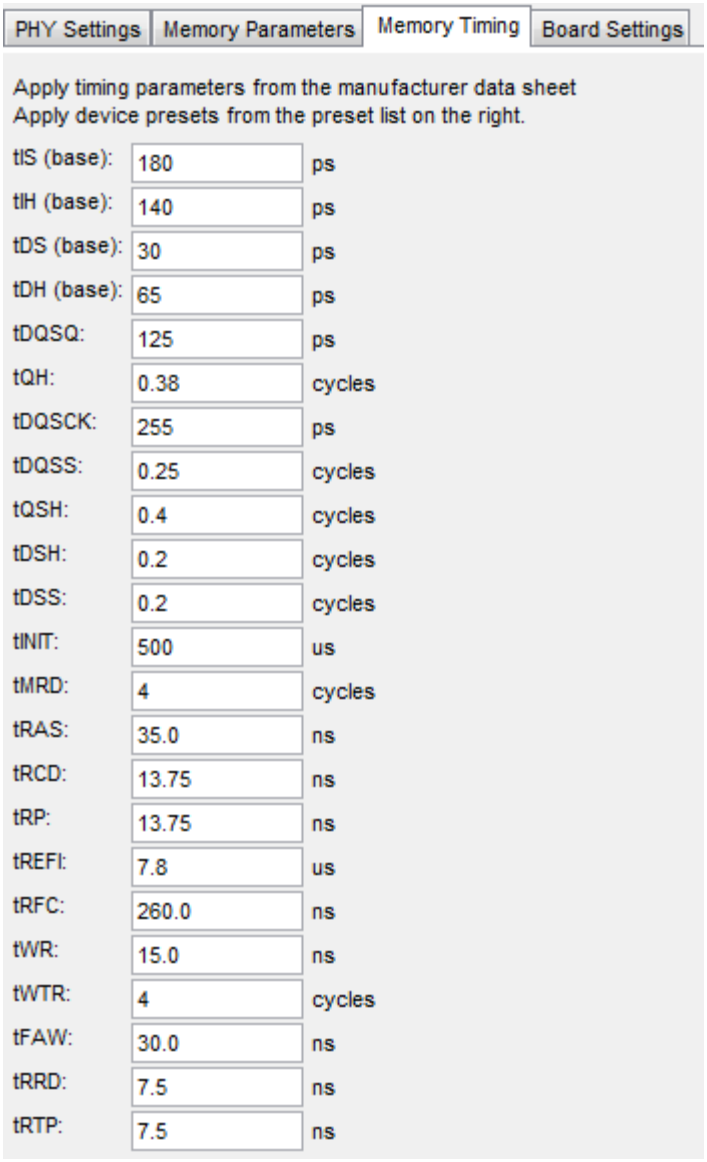
Verify the parameters that are selected as shown below:



The **Memory Initialization Options** are at the bottom of this page, where the values are again taken from the Micron datasheet.



Under the **Memory Timing** tab**,** timing parameters need to be verified:

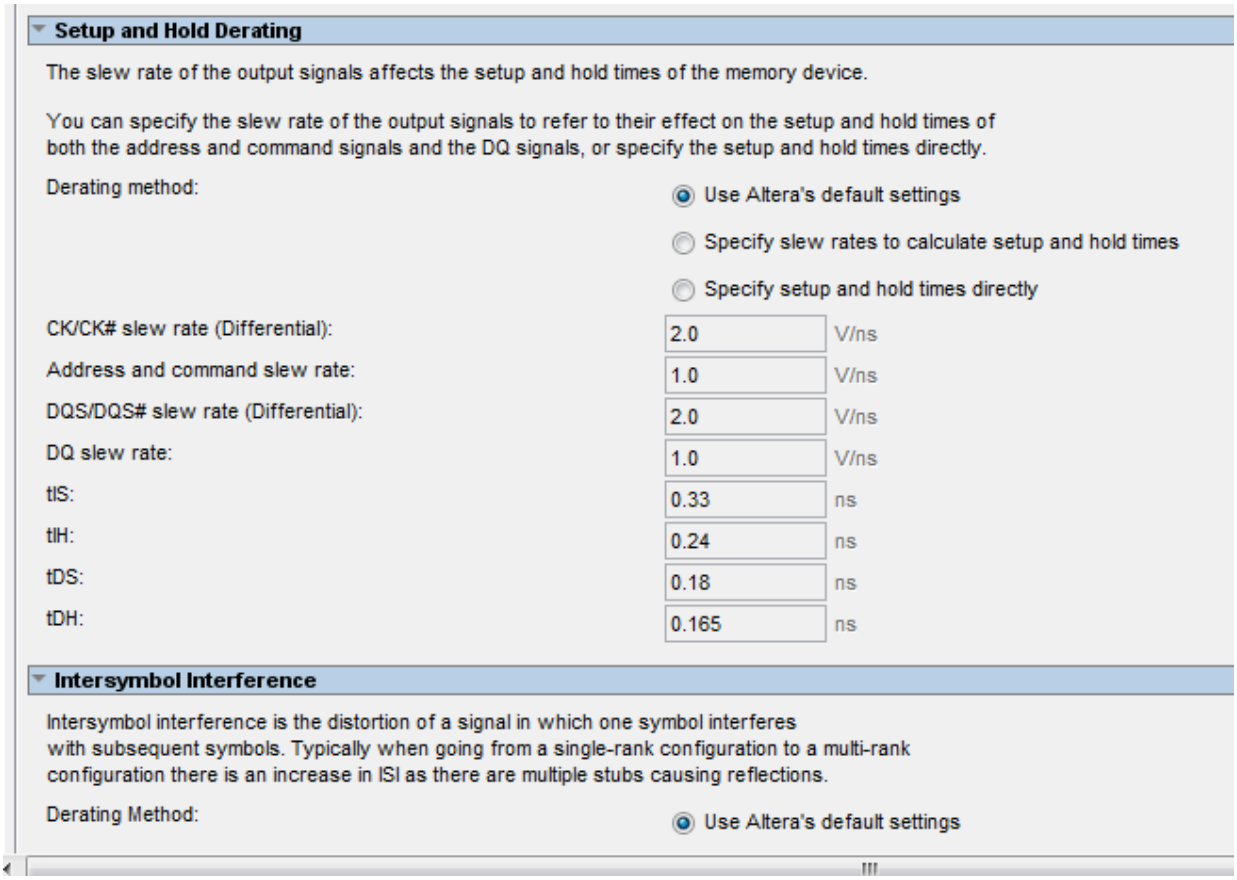


The memory timings listed above match those in the datasheet.

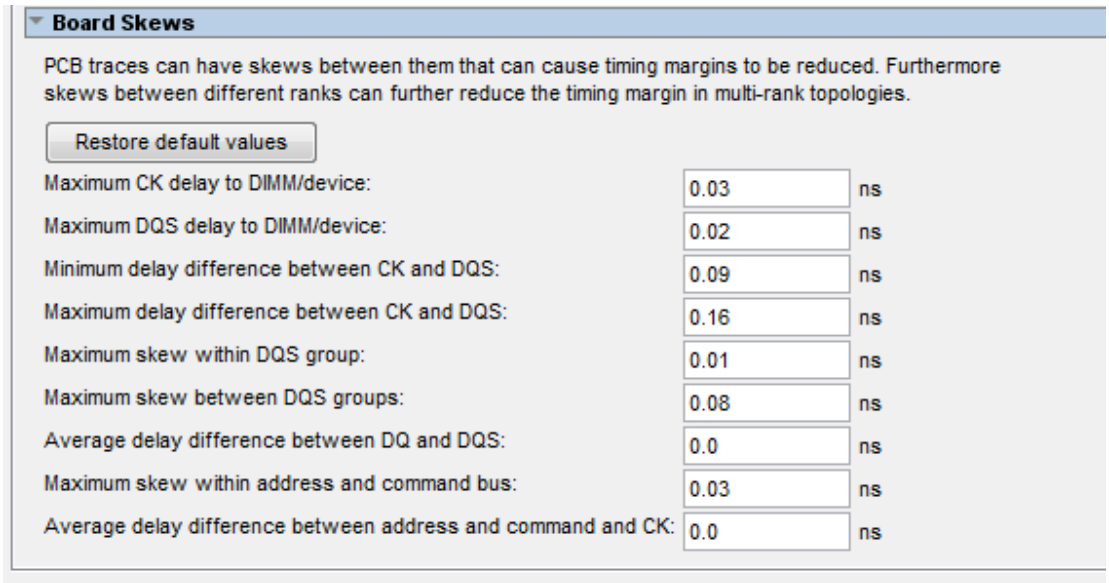
Under the **Board Settings** tab:

Verify that the "Setup and Hold Derating" is set to Use Altera's default settings.

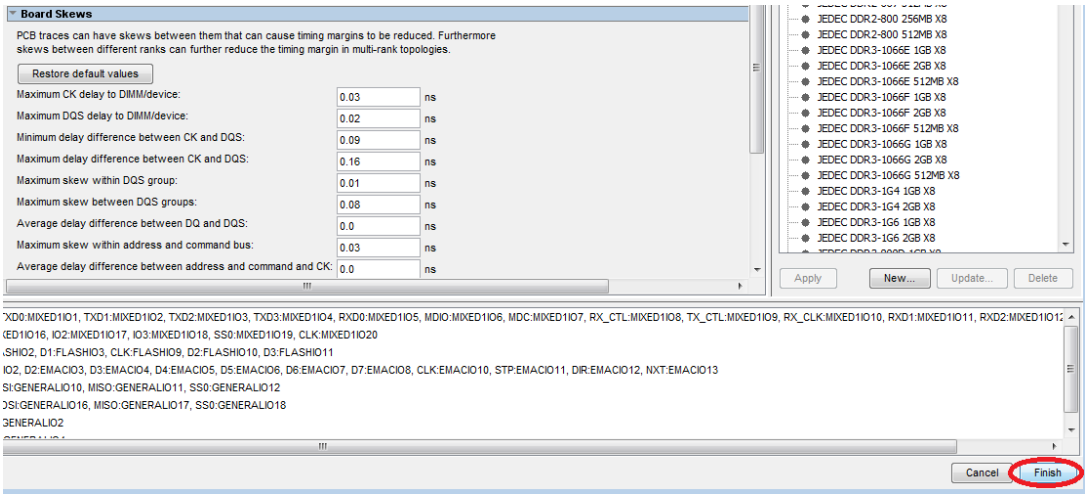
Verify that the Intersymbol interference should be left as default to Use Altera's default settings.



Since the board design is complete, the board skews, which are linked to timing differences between traces are known. These settings should be set to:



At the bottom of the **Hard Processor System** component, select **Finish**, as the settings for the Hard Processor System are now configured.



In the export column associated with the HPS_0, there are two signals that need to be exported to the top level of the project. This is the reason why there are two signals already associated as seen below.

hps_0	Hard Processor System		
h2f_reset	Reset Output	hps_0_h2f_reset	
f2h_axi_clock	Clock Input	Double-click to export	clk_0
f2h_axi_slave	AXI Slave	Double-click to export	[f2h_axi_clo...
h2f_axi_clock	Clock Input	Double-click to export	clk_0
h2f_axi_master	AXI Master	Double-click to export	[h2f_axi_clo...
h2f_lw_axi_clock	Clock Input	Double-click to export	clk_0
h2f_lw_axi_master	AXI Master	Double-click to export	[h2f_lw_axi_...
memory	Conduit	memory	
hps_io	Conduit	hps_0_hps_io	

Add and configure FPGA Peripherals

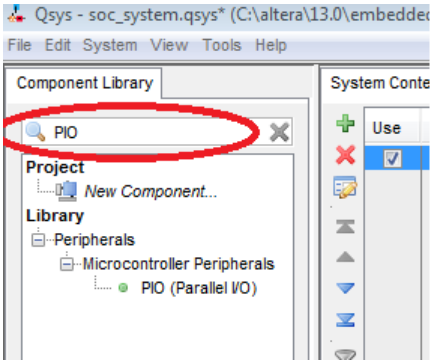
The next step is to add and configure the FPGA peripherals in Qsys.

4.2.5 Configure LED PIO

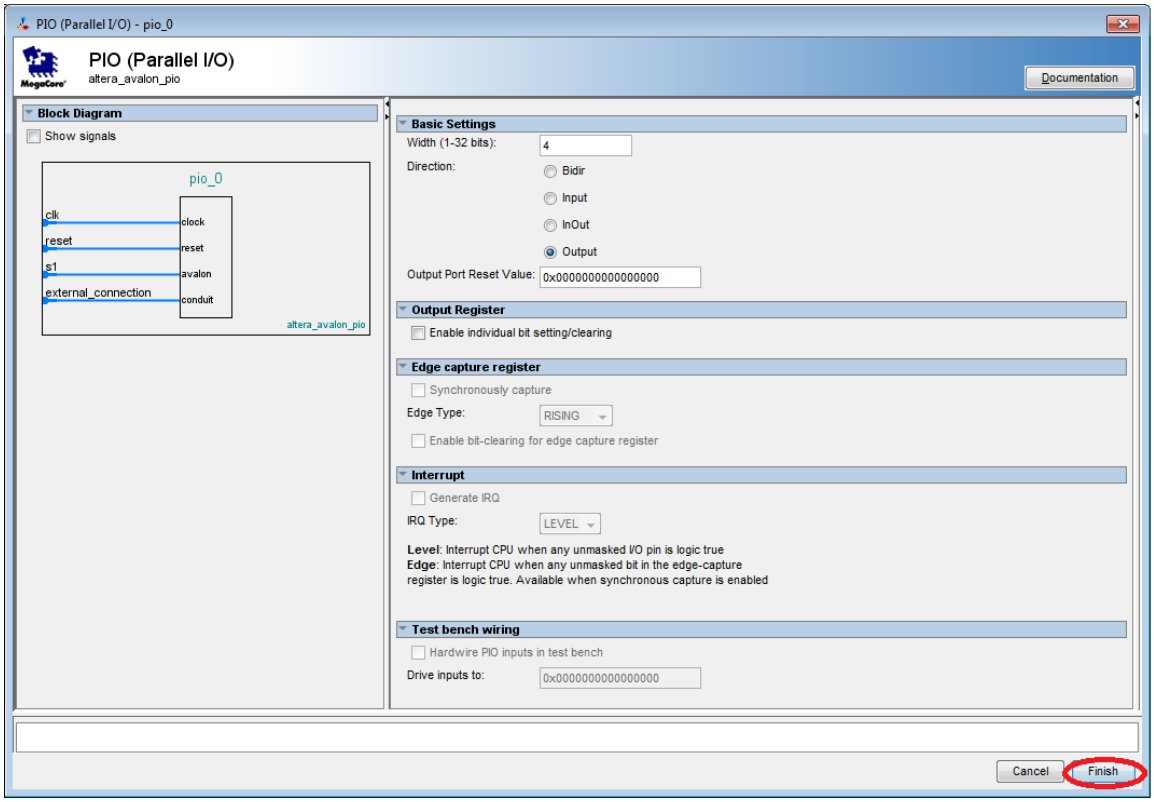
The SoCKit board has four LEDs connected to the FPGAs I/O pins. These LEDs are driven with an output PIO component.

To add this component

- Type PIO in the Search Window
- Double click on the PIO (Parallel I/O) to add to your system:



- Set the **Width** to be **4**, which is the number of LEDS on the board connected to the FPGA I/O pins.
- Insure that the direction is set to **Output**.



Select **Finish**.

Change the default name of the PIO component to be **led_pio**.

- To change the name, **select the component** (High Light), right click and select **Rename**.

Since the LEDs connections will be driven by the FPGAs I/O, the LED signals will need to be **exported** to the top level of the project. To do so:

- double click in the export column associated with the external connection of the **led_pio** and type **fpga_led_pio.**

The component should now look like:

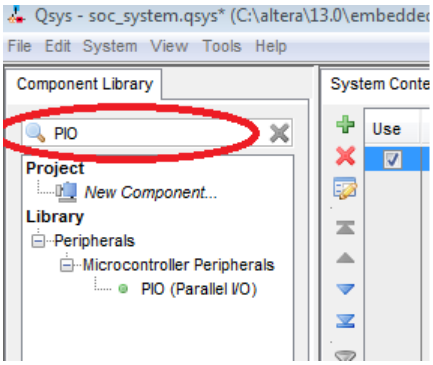
⊟ led_pio	PIO (Parallel I/O)	
clk	Clock Input	Double-click to export
reset	Reset Input	Double-click to export
s1	Avalon Memory Mapped Slave	Double-click to export
external_connection	Conduit Endpoint	fpga_led_pio

4.2.6 Configure Button PIO

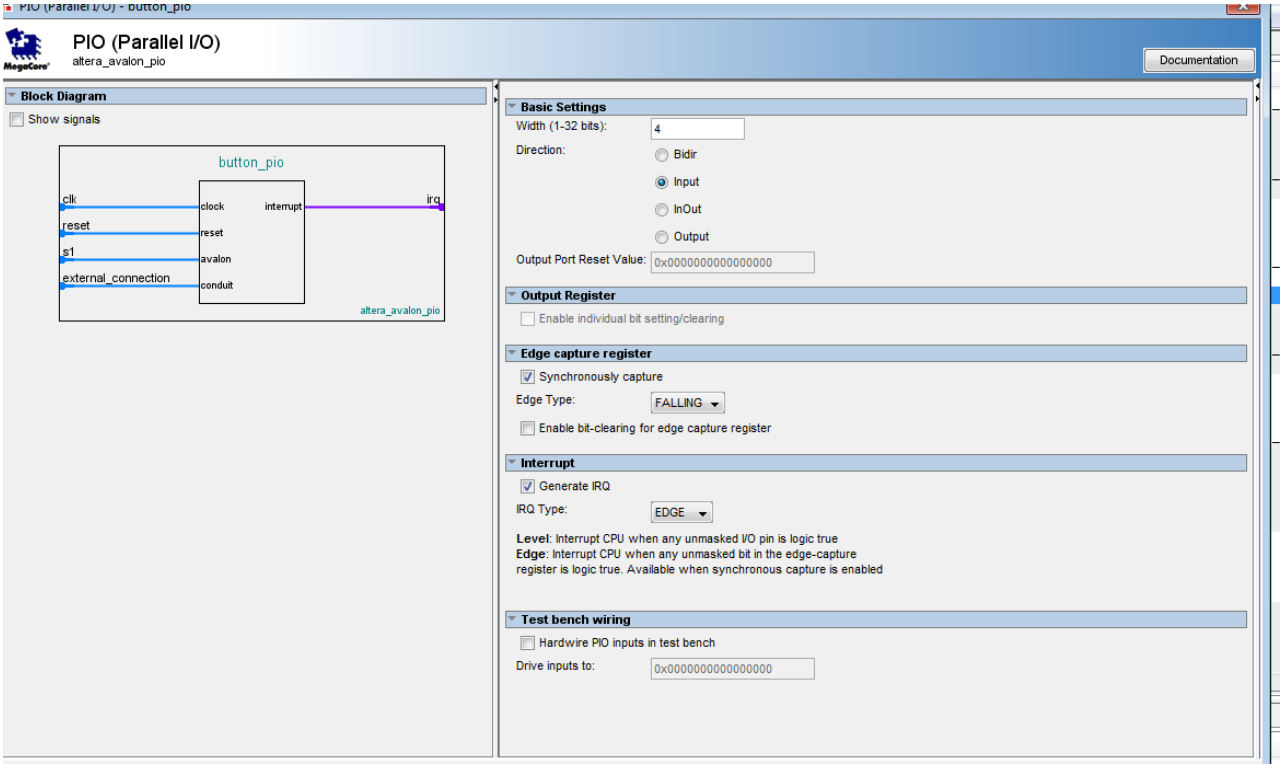
The SoCKit has four **push buttons** that are **connected** to the **FPGA**. The PIO peripheral will be configured as an Input and will be used to read in the push buttons connected to the FPGAs I/O.

To add the 4 input component:

- type PIO in the Search Window
- double click on the PIO (Parallel I/O) to add to your system:



- Set the **width** to be **4. E**nsure that the direction is set to **Input**.
- Set the **Edge capture register** to **Synchronously capture** on the **FALLING** edge.
- Enable **Generate IRQ.** Set the **IRQ Type** to **Edge.**



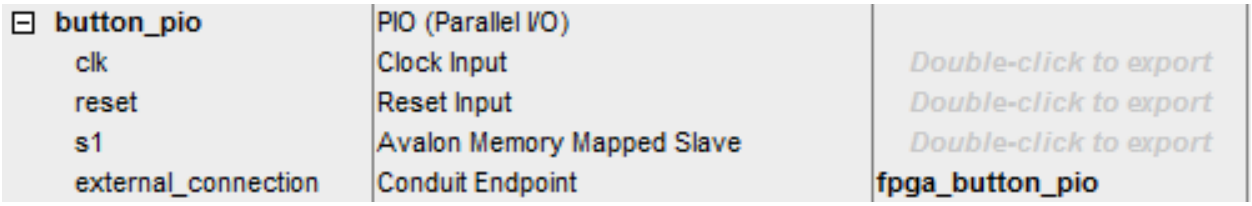
Select **Finish**.

Change the default name of the PIO component to be **button_pio**. To change the name, **select the component** (high light), **right click** and select **Rename**.

Since the Button PIO connections will be inputs to the FPGAs I/O, the Button signals will need to be **exported** to the top level of the project.

- To do so, **double click** within the export column associated with the external connection of this component.
- Type in **fpga_button_pio**.

The settings for the component should now look like:



Save the Qsys system, select: **File** -> **Save**.

Review other Qsys components

There are many other components in the Qsys system that have already been configured. These components have already been configured for the SoCKit embedded system. Therefore, these components do not need to be configured.

A summary of these components:

The FPGAs array provides **on chip memory** blocks that can be used to build up internal RAM (or ROM) blocks of memory that is available for any Master in the Qsys system. This provides the HPS Cortex-A9 MPU access to very low-latency, high speed memory for code or variable storage. This is the **onchip_memory2_0** component.

The **JTAG to Avalon Master** accepts encoded streams of bytes of transaction data on the JTAG interface and initiates Avalon-MM (multi-master) transactions on the Avalon-MM interface. The JTAG to Avalon Master is also used for debugging, with tools such as System Console and SignalTap. Both System Console and SignalTap will be used later in this workshop.

The **System ID peripheral** is a very important peripheral to include in your system. It allows the software development tools to validate that the software application is being built for the correct hardware system. Basically, it will not allow software to be executed on an incompatible hardware configuration.

The SoCKit has four **DIP switches** on it that are connected to the FPGAs I/O pins. The **dipsw_pio** is an input PIO peripheral that is used to read in the DIP Switch settings in a fashion similar to the **button_pio** peripheral.

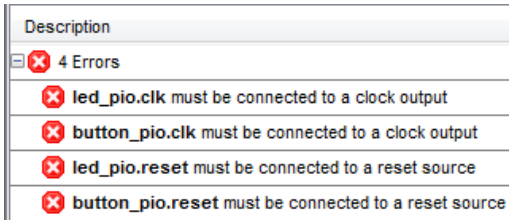
Software developers need to have access to a debug serial port from the target to leverage printf debugging, input control commands, log status information, etc. The **jtag_uart** peripheral connects to the debugger console and provides an interface to the developers console for that and other purposes.

Interrupts are signals that need immediate attention. Interrupts have higher priority than other processes. The **interrupt_capturer** component is an Avalon Memory Mapped module (written in Verilog) to capture system interrupts and pass them on to the HPS Cortex-A9 MPU.

4.3 System Configuration

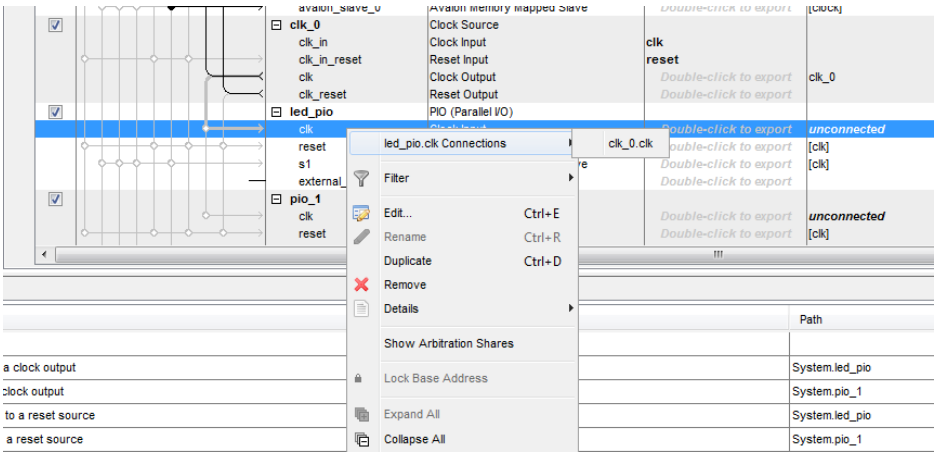
4.3.1 Connect HPS interfaces to FPGA Peripherals

The Qsys components that were just created (led_pio and button_pio) have not been connected; therefore, there are errors. These errors will be removed in the next steps.



The following steps will connect the components to the system; these include the Avalon Memory Mapped signals as well as the clock and reset signals. There are two methods that can be utilized to connect your new system components. Visually by using the patch panel to connect the nodes or busses (dots) or by right clicking on the menu (as described below).

To connect the led_pio to the system via the menu method: **right click** on the **clk** signal of the **led_pio**. The available connections are connected as shown:



- Select **Clk_0.clk**

The clock of the PIO is now connected to the 100 MHz clock from the FPGAs dedicated clock input pin AF14.

The following connections will be made with the **same process of right clicking on the signal** and then selecting the signal to be connected. **The following table describes what signals are to be connected together**

Name of Component	Name of Signal	What component that the signal is to be connected to	What signal of the component that is to be connected to
led_pio	Clk	Clk_0	clk
led_pio	reset	Clk_0	clk_reset
led_pio	s1	master_non_sec	master
led_pio	s1	hps_0	h2f_lw_axi_master
button_pio	clk	clk_0	clk
button_pio	reset	clk_0	clk_reset
button_pio	s1	master_non_sec	master
button_pio	s1	hps_0	h2f_lw_axi_master

As the connections are made, the errors at the bottom of the Qsys window will be removed. Since the IRQs have yet to be set there will still be errors and they will be removed in the next section.

4.3.2 Set IRQs

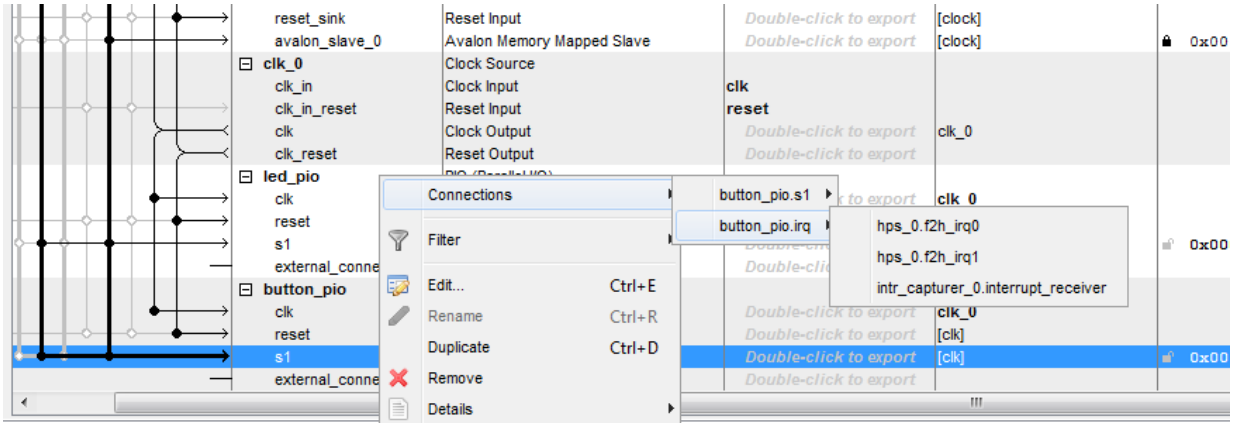
Components with interrupts can be set to have higher priority than other system components; therefore, the components with interrupts our Qsys system need to be assigned.

The DIP switch, button and JTAG all have interrupts that will be captured by the interrupt capture module. These interrupts will be connected to the HPS component. The **dipsw_pio** and **jtag_uart** components already have the interrupts assigned to them.

The **pio_button** component also needs to have an **IRQ assigned to it.**

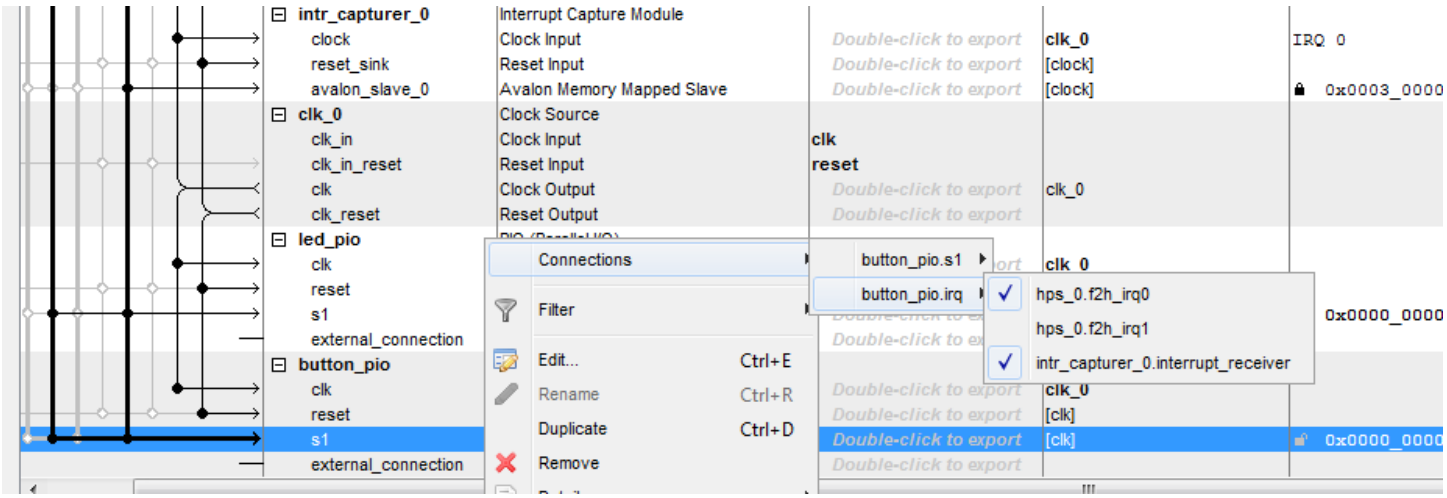
To assign IRQ, first the push button needs to connected to the IRQs and then the interrupts level needs to be assigned.

- Right click on the s1 of the **button_pio** so that the **button_pio irq** can be selected, as seen in the following screenshot.



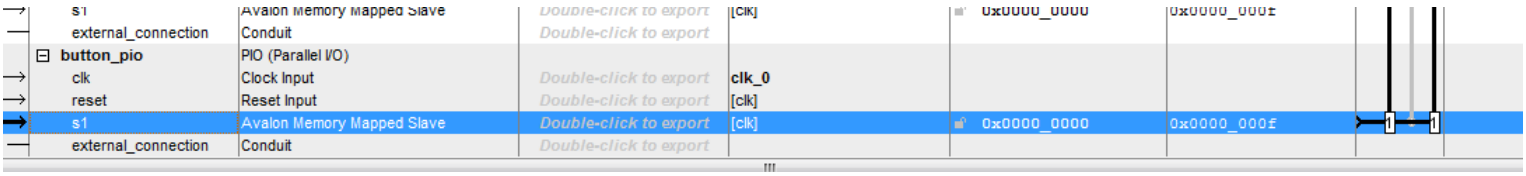
- Select the **hps0.f2h_irq0** so that this interrupt is selected.
- Repeat this step again, but this time select the **intr_capturer_0.interrupt_receiver**.

The interrupts should now look like:



Next, verify the **interrupt level** for the push button.

- **Select the IRQ column** (second to last column in the Qsys), where the level of the IRQ that will be assigned.
- Click in the box and **type in the number 1**.



Errors still remain at the bottom of the Qsys screen and these will be removed in the following steps.

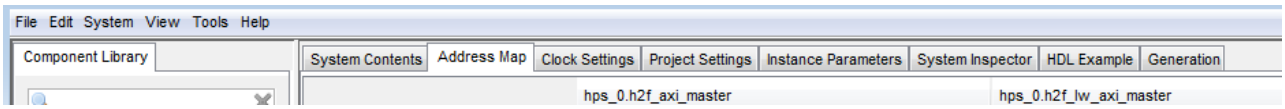
4.3.3 Set Base Addresses

The system has a memory map. A systems memory map consists of addresses that are assigned to a component and these address ranges cannot overlap. The addresses can be assigned automatically or manually. For this workshop, the addresses are assigned manually since the software portion of this workshop will use these addresses.

To assign base addresses:

- Select the Address Map tab in Qsys. This is a table that includes all of the memory -mapped slaves in the design and the address range that each connected memory-mapped master uses to address that slave. The blank cells, which are by default, implies that there is no connection between that master and slave.

The "Address Map" tab is seen as:



The only address values that need to be changed are the **led_pio** and **button_pio** .

Manually, change the values to:

	Hps_0.h2f_axi_master	Hps_0.h2f_lw_axi_master	Master _secure. master	Master_non_sec .master
led_pio.s1		**0x0001_0040** - 0x0001_004f		**0x0001_0040** - 0x0001_004f
button_pio.s1		**0x0001_00c0**- 0x0001_00cf		**0x0001_00c0**-0x0001_00cf

Now all of the errors in your Qsys system should be eliminated!

For reference only, the address map should now be:

	Hps_0.h2f_axi_master	Hps_0.h2f_lw_axi_master	Master _secure. master	Master_non_sec .master
Hps_0_f2h_axi_slave			0x0000_0000- 0xffff_ffff	
Onchip_memory2_0.s1	0x000_000- 0x000_ffff			0x000_000- 0x000_ffff
Sysid_qsys.control_slave		0x0001_0000- 0x0001_0007		0x0001_000- 0x0001_0007
Dipsw_pio.s1		0x0001_0080- 0x0001_008f		0x0001_0080- 0x0001_008f

	Hps_0.h2f_axi_master	Hps_0.h2f_lw_axi_master	Master _secure.master	Master_non_sec.master
Jtag_uart.avalon_jtag_slave		0x0002_0000-0x0002_0007		0x0002_0000-0x0002_0007
Intr_capturer_0.avalon_slave_0				0x0003_0000-0x0003_0007
Led_pio.s1		0x0001_0040 - 0x0001_004f		0x0001_0040 - 0x0001_004f
Button_pio.s1		0x0001_00c0-0x0001_00cf		0x0001_00c0-0x0001_00cf

Once these values are entered, go back to the **System Contents** tab and go the **led_pio** and select the row associated with the base addresses, as shown below.

In the Base address column, **select the lock icon** by the Base Address, so that the Base Address is locked, as seen below. Locking an address prevents a base address from being changed.

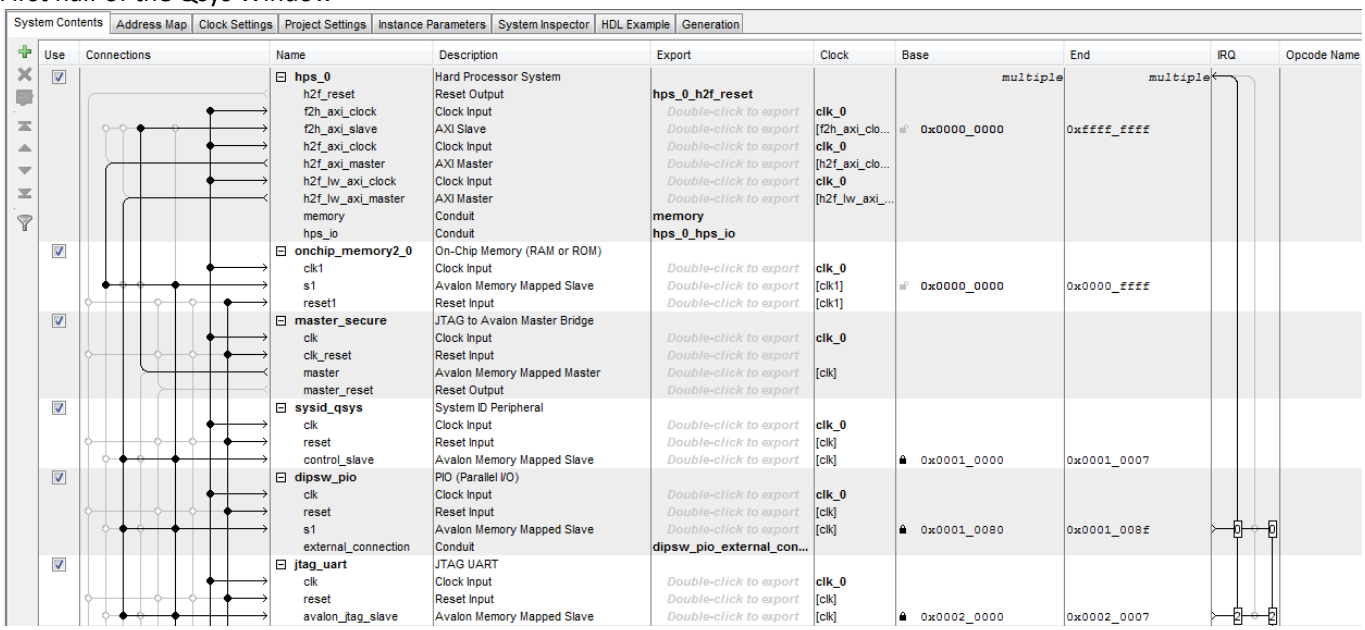
Repeat the same step for **button_pio** so that its address range is also locked.

- Save the Qsys system, with **File** -> **Save**.

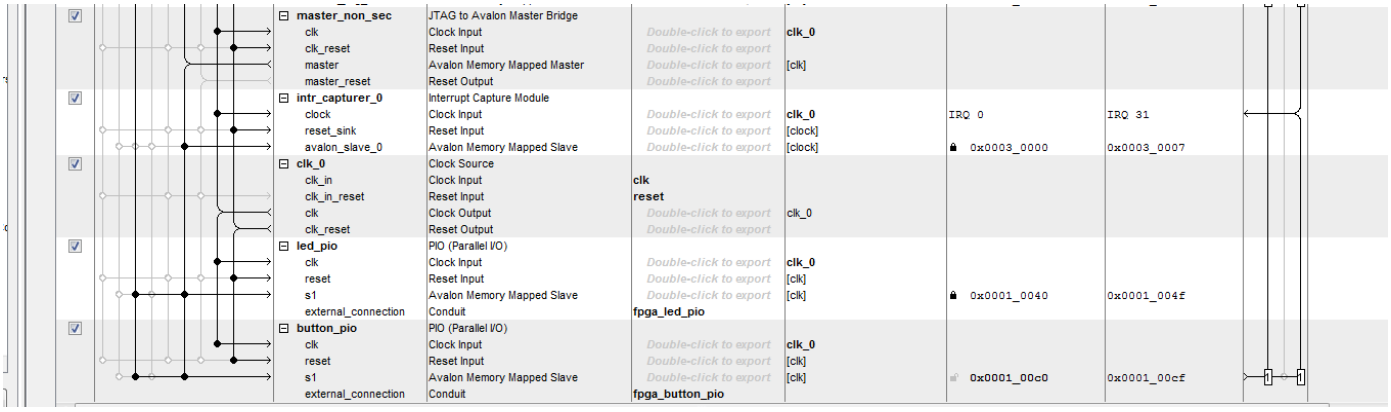
4.4 Generate the System

Please Double-check to make sure that all the *component names, clocks and base addresses* in your Qsys system match the names below.

First half of the Qsys Window



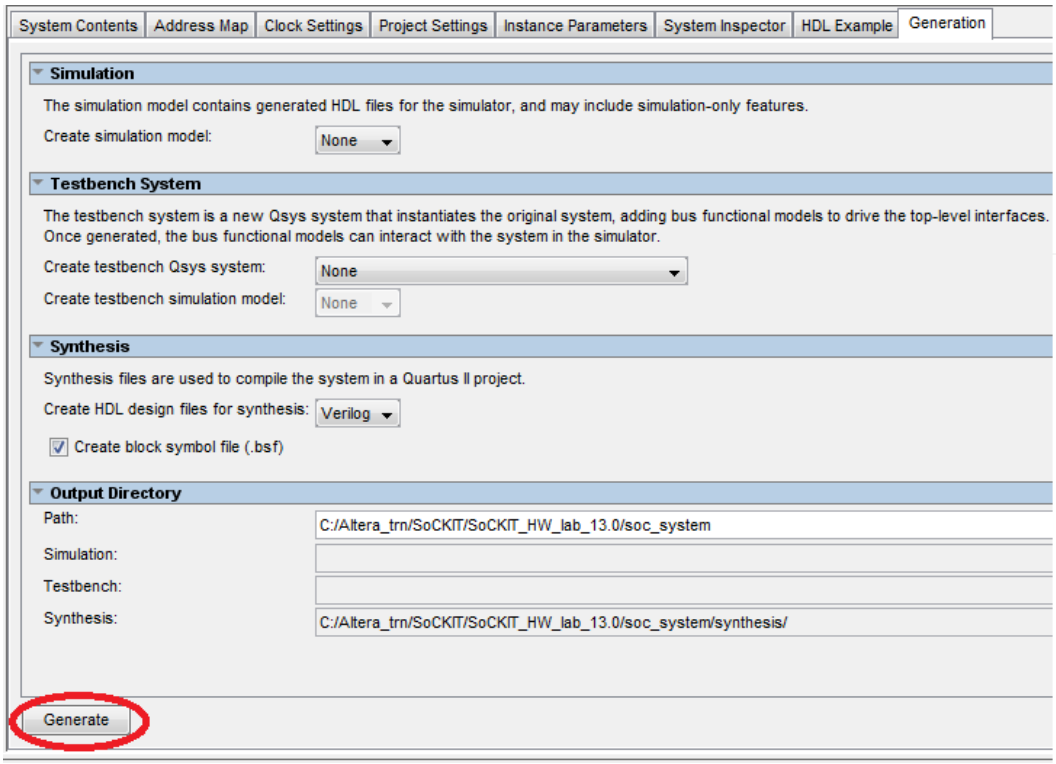
Second half of the Qsys window



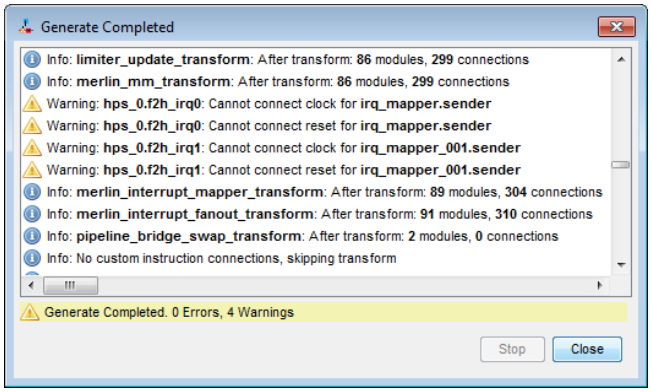
Select the "**Generation**" Tab.

.

Select **Generate** button. If it asks you to save, select Yes.



You will receive the following warnings, but they can be disregarded:

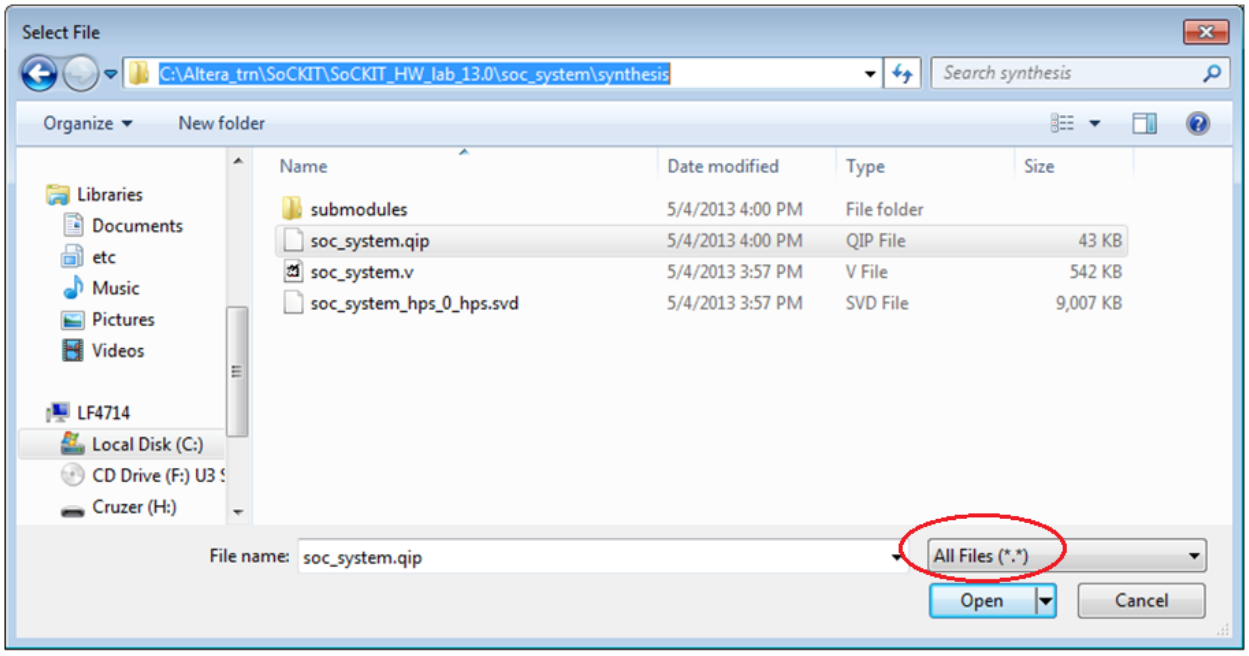


Exit Qsys by clicking the **Exit** button and click **Save** when it asks if you would like to save the system.

Qsys will now create:

Qsys will **generates** the **HDL** files (Verilog or VHDL) for the defined system. These HDL files are then used by Quartus II to compile and generate a set of files that **defines the hardware system**. This set of files includes the HDL files, Tcl (Tool Command Language) files that define dedicated pin locations for selected HPS peripherals, Tcl files that define the Multiport Memory Controller in the HPS & FPGA, QIP files that include: selected IP and SDC (Synopsis Design Constraint files) utilized by TimeQuest to constrain the complete system design. You can find these files here:

- Set the **filter** below to be **All Files**



CONGRATULATIONS!!

You have just built your first Qsys system!

MODULE 5. Complete the Quartus II Project

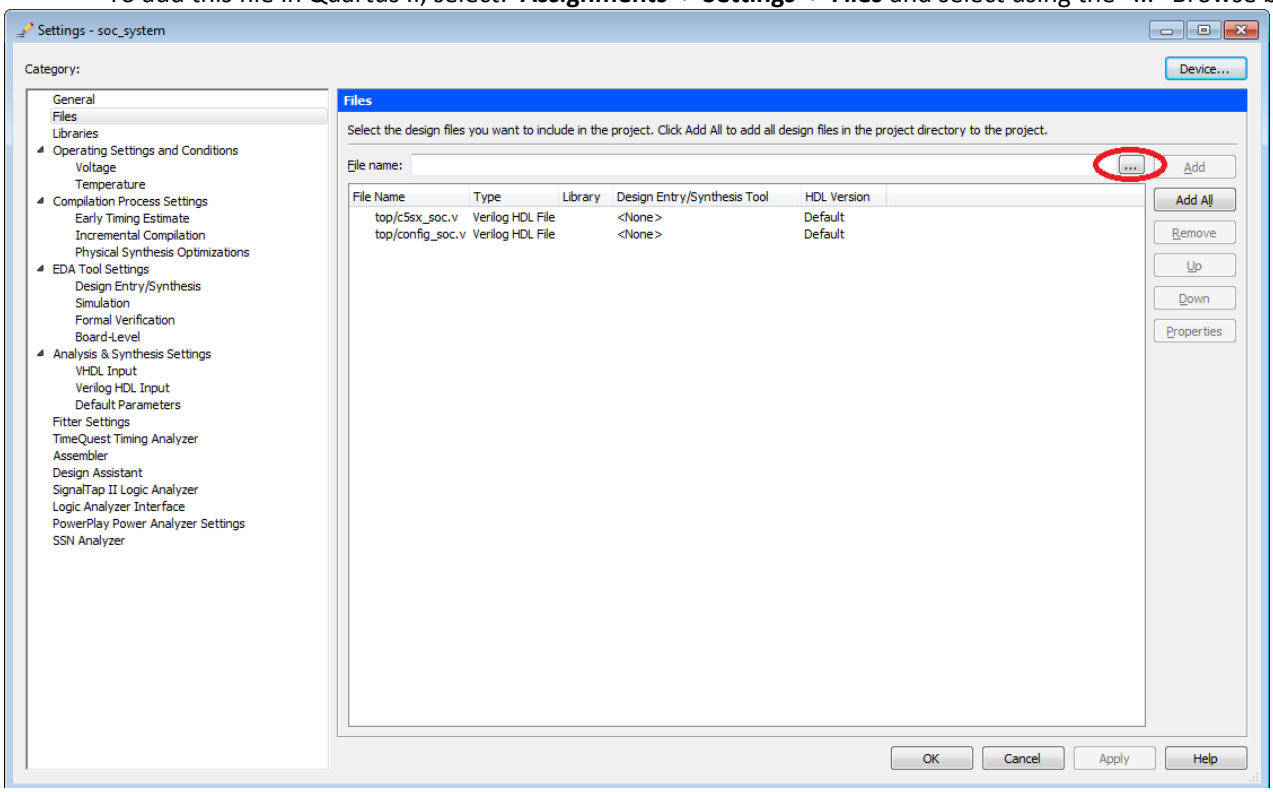
Module Objective:

In this module you complete the Quartus II project by adding the generated Qsys system to the top level entity. Use Quartus II tool to perform analysis, synthesis, fitting, place and route as well as the static timing analysis. At the end of the compilation, an SRAM object file (*.SOF) will be generated for the FPGA. This SOF will then be downloaded to the Cyclone V SoC device via the USB Blaster and the Quartus II programmer.

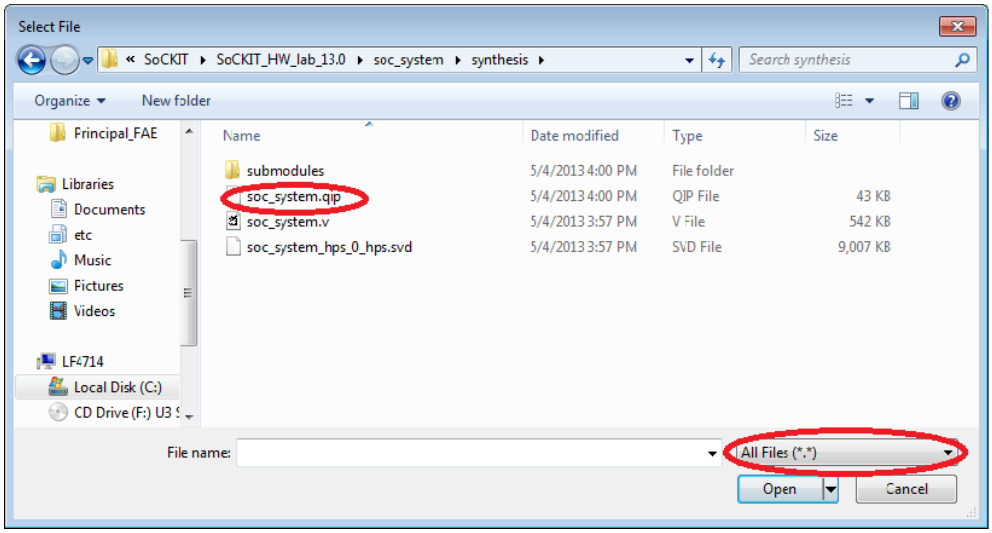
5.1 Set up the Quartus II project to point to the correct files

When the generate button in Qsys in the previous step was selected, Qsys generated numerous HDL (Verilog) files that will be utilized by the synthesis tool in Quartus II (QIS). These files need to be added to the Quartus II project so that they can be compiled in the next step. **However**, rather than adding all of the files separately, **there is a single file**, **soc_system.qip**, that will contains the **paths for all of the IP cores**.

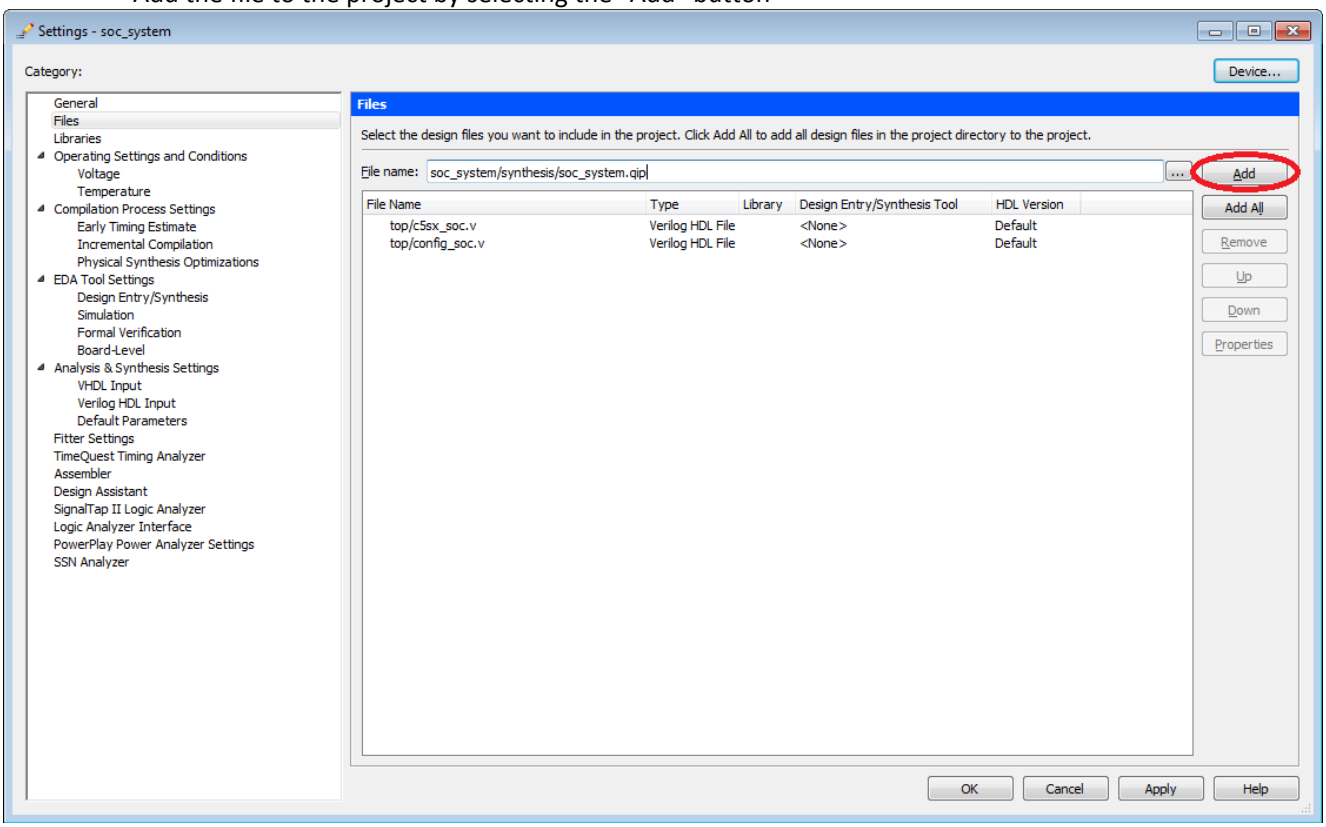
- To add this file in Quartus II, select: **Assignments** -> **Settings** -> **Files** and select using the "..." Browse button



- Next, select the **system_soc.qip** file (select All Files (*.*) drop down dialog to see the file)

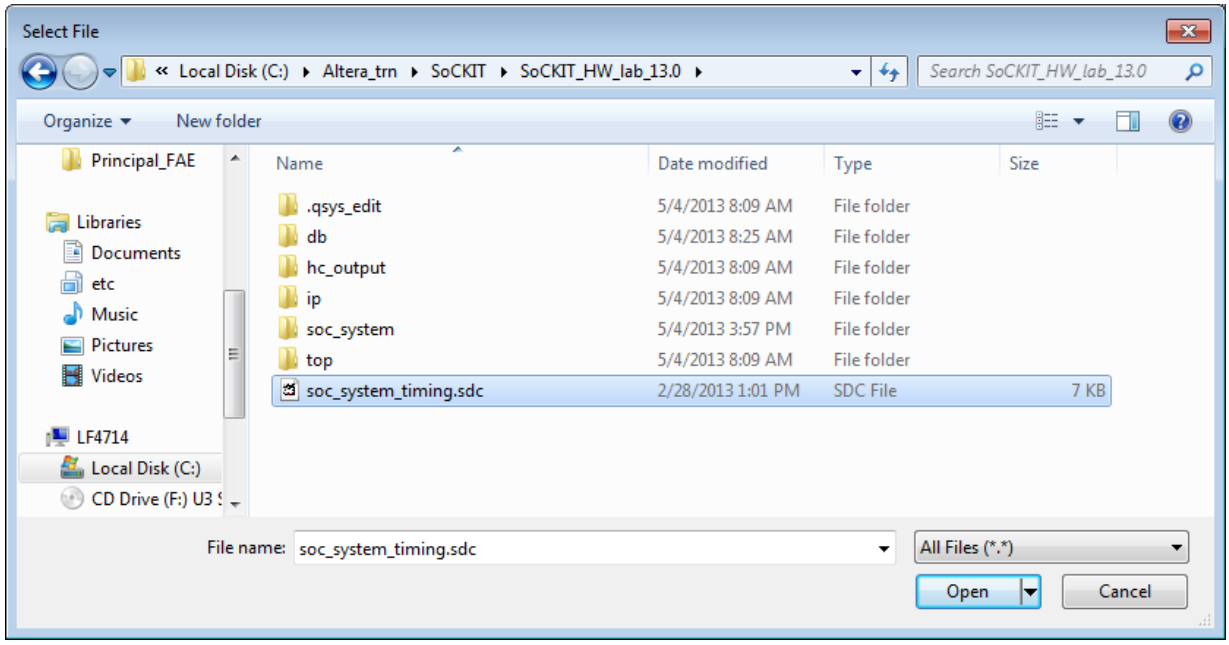


- Then, select "Open"
- Add the file to the project by selecting the "Add" button

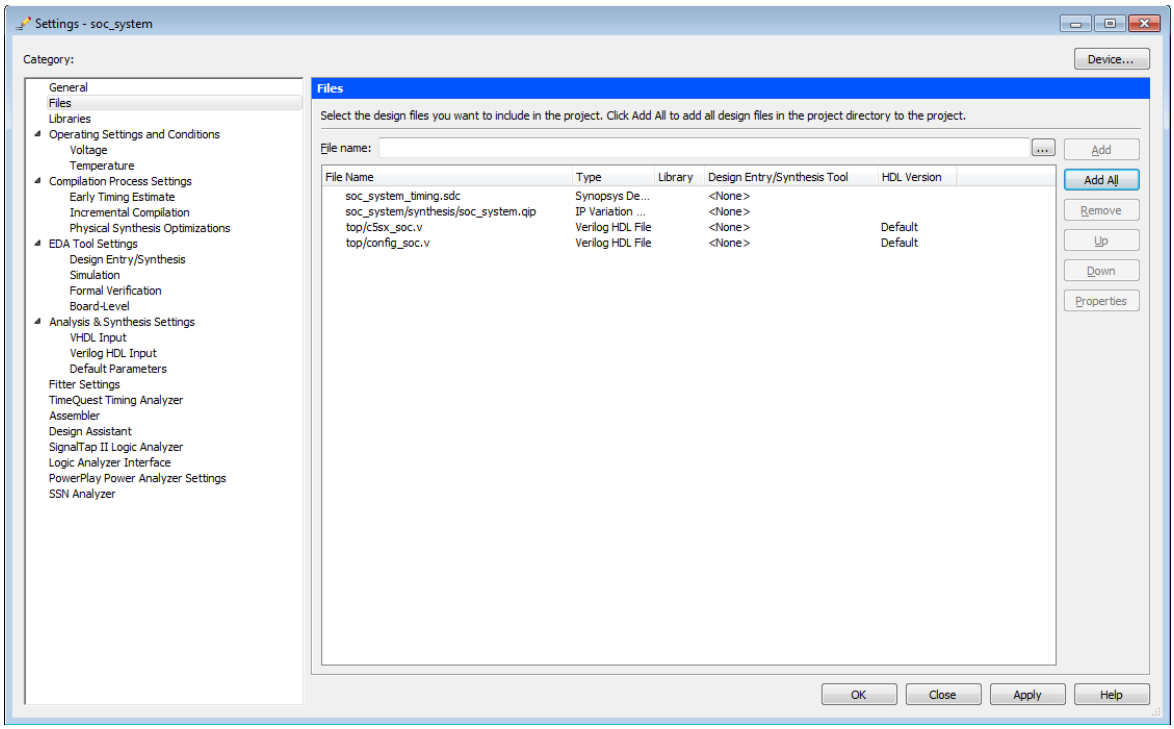


- Select **Apply**.

- Using the same window and use the same process in the two previous steps to add the **soc_system_timing.sdc**



- The window should look like:



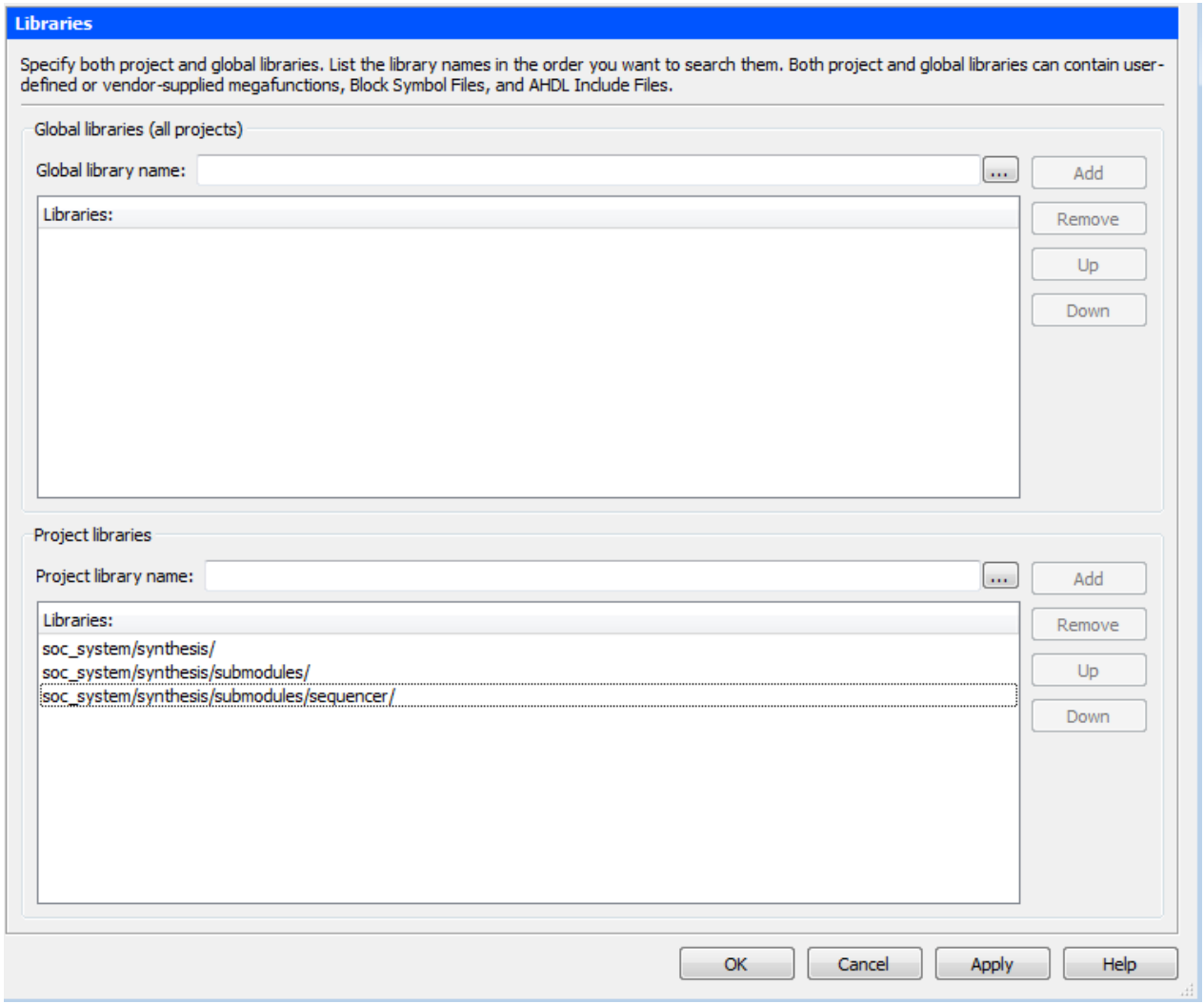
- Select **Apply**, Select **OK**

- The next step is to add the synthesis directories to the project. Quartus II uses these files in these directories to compile the design.
- To do this, select **Assignments** -> **Settings** -> **Libraries** and select in the Project Library section, with the ... (Browse) button, the three HPS libraries.

  soc_system/synthesis/
  soc_system/synthesis/submodules/
  soc_system/synthesis/submodules/sequencer/

- Select Add after adding each library.

  The end result should look like:

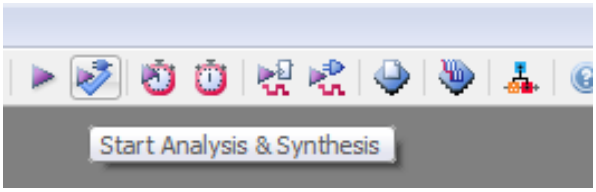


- Select Apply
- Select OK to finish.

5.2 Analysis and Synthesis

Before the pin-outs can be added to Quartus II and Analysis and Synthesis needs to be completed. Analysis and Synthesis is the stage that analyzes and synthesizes design files and creates a net-list within a project database. These nets can then be assigned to actual device pins.

- Select the icon with the checkmark (purple arrow with blue checkmark) at the top of the Quartus II window, as seen below.

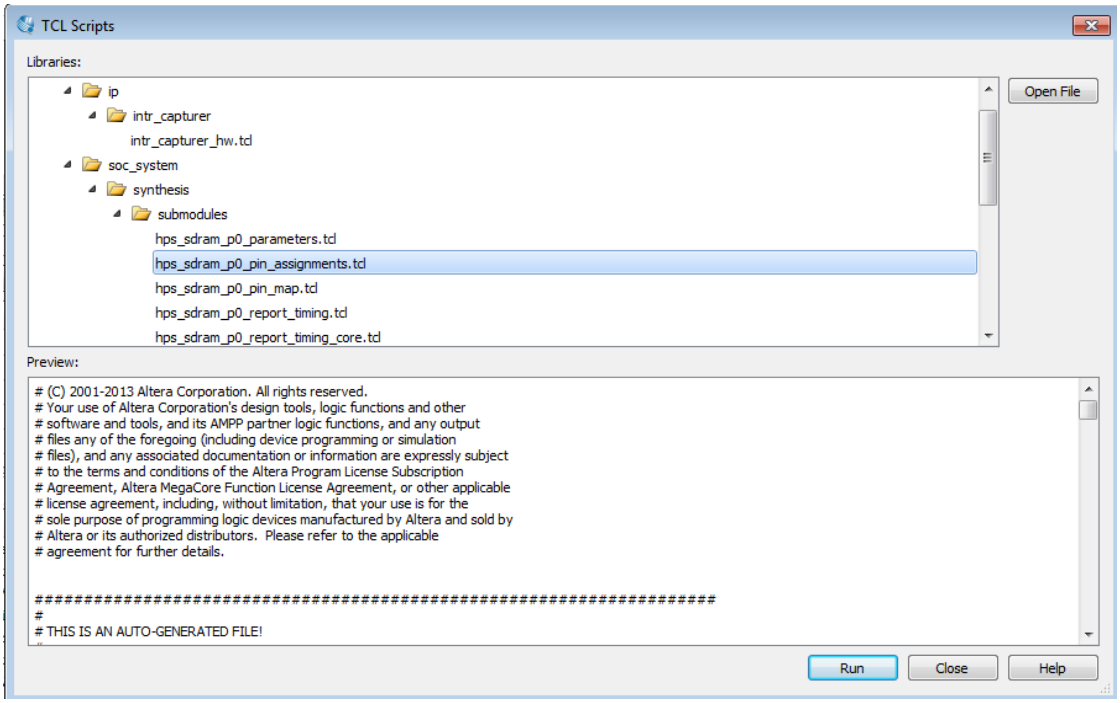


- Analysis and Synthesis will now run. Once it is complete, select OK.
- There should be no errors.
- If there are errors, there will be error messages at the bottom of the page that will describe the errors. These errors will need to be resolved before continuing.

5.3 Adding Pin assignments

Since an HPS was instantiated in the Qsys system, pin assignments other than memory pins, do not need to be specified in Quartus II. The HPS pin assignments are automatically assigned when the HPS was instantiated and this information is contained in the XML files, which the software development tools will utilize. However, the HPS memory pins will need to be assigned, since there are External Memory Interface variations that can occur. This task is completed by running a Tcl script that was created by Qsys for this purpose.

- To run the TCL script, select **Tools** -> **TCL** script and select the **hps_sdram_p0_pin_assignments.tcl** as seen below.



- Select **Run**.
- Select **OK** after the TCL script states that it has correctly added pins.

5.4 Compile (Optional step for this lab)

At this point the design is ready for compilation.

Since a full compilation can take a while, depending on the computer being used, there are precompiled files available which will be used in **Module 6**.

The generated file is: soc_system.sof

To complete a full compilation, select: Processing -> Start Compilation. There should be no errors in the compile, and you should see the successful completion dialog when it is finished. You will see some warnings related to the files from the automatically generated system, missing assignments/features and incomplete pin assignments but these will not affect the functionality of the system.

The output of the compilation is a SOF file entitled "**soc_system.sof**"

MODULE 6. Hardware Debug Flow (System Console)

System Console is used for low-level system debug over JTAG on any Qsys based system

- Tcl-based

  Familiar development tool language

- Interactive

  Opens as a separate window

  Opens in Qsys or in the Nios® II Command Shell

- Scriptable

  Tcl files can be “sourced”

  Supports command line arguments

  Supports standard input/output

Examples of Use

- Low-level Debug

  Board bring-up and interface testing

  System clock, reset, and JTAG chain validity testing

  SOPC Builder component functionality testing

- System-level Debug

  Provide test vectors, return response

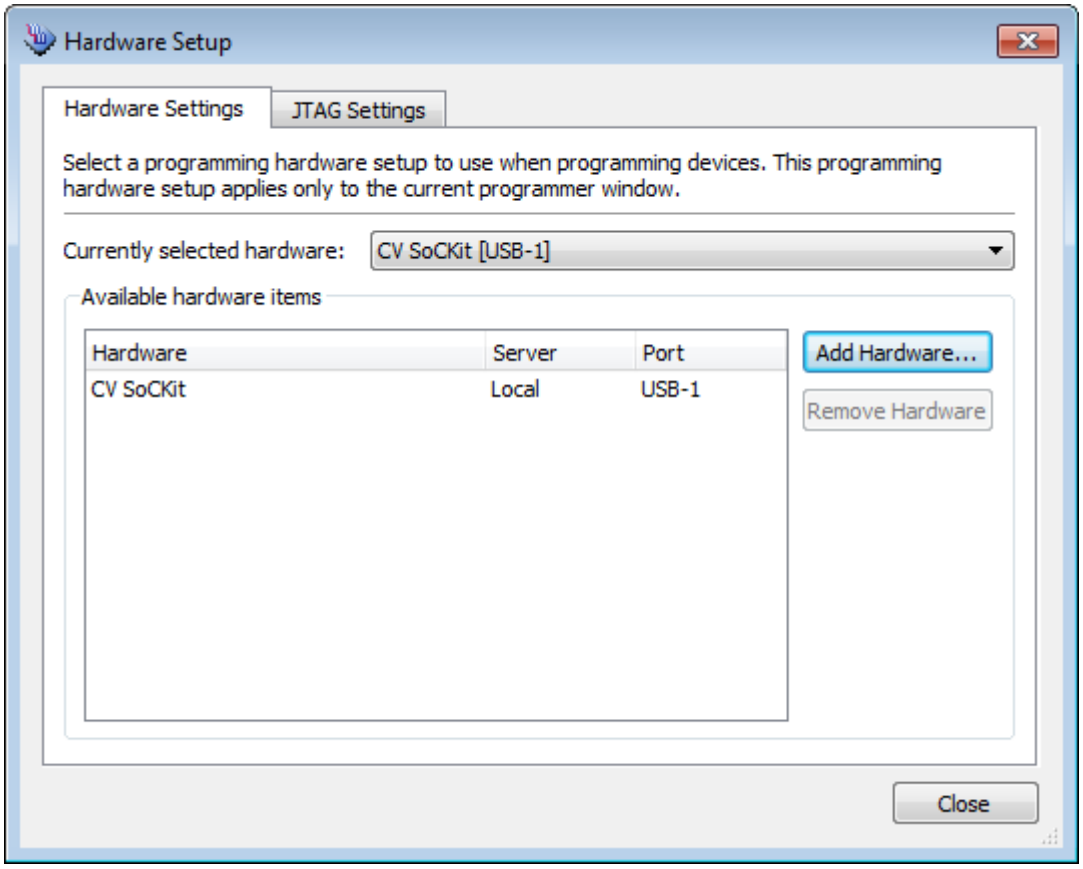
  No processor required

For more detailed information, please download and read the System Console User Guide

6.1 Downloading and Programming FPGA

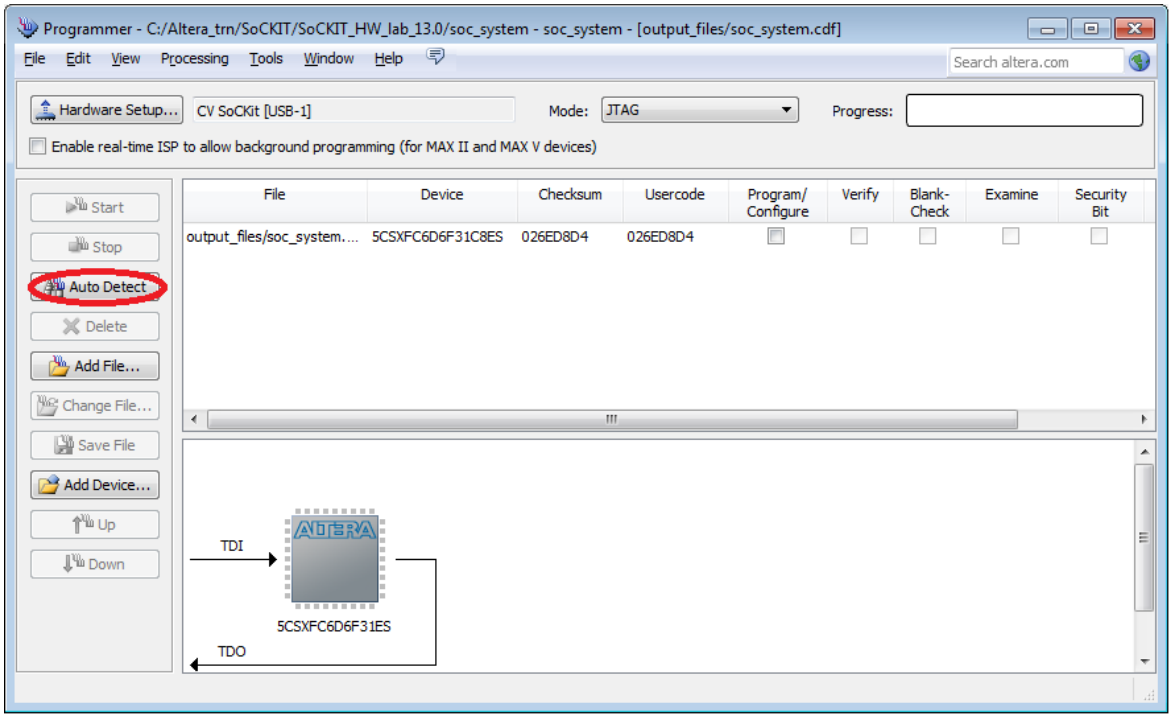
Previously, when the USB Blaster driver was enabled, the SoCKit was plugged in, the programming dip switch was enabled and the cables were connected, ensure that the SoCKit is still powered on, and cables are still connected.

- Within Quartus II, select **Tools** -> **Programmer**.
- Select Hardware Setup (top left hand side of the Programming Window) and ensure that the currently selected hardware is CV SoCKit [USB-1]. It should be on by default.

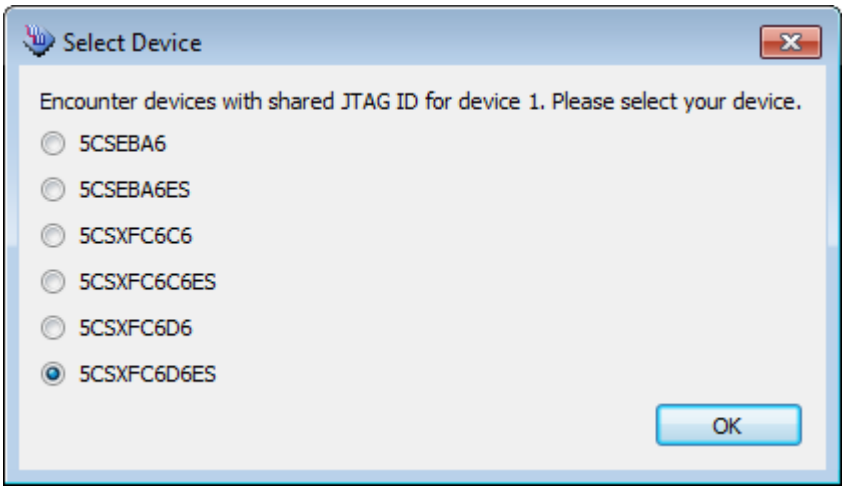


- Select **Close**.

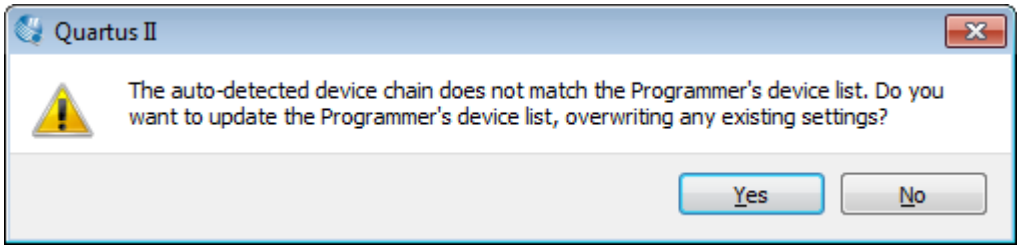
- Select "Auto Detect"



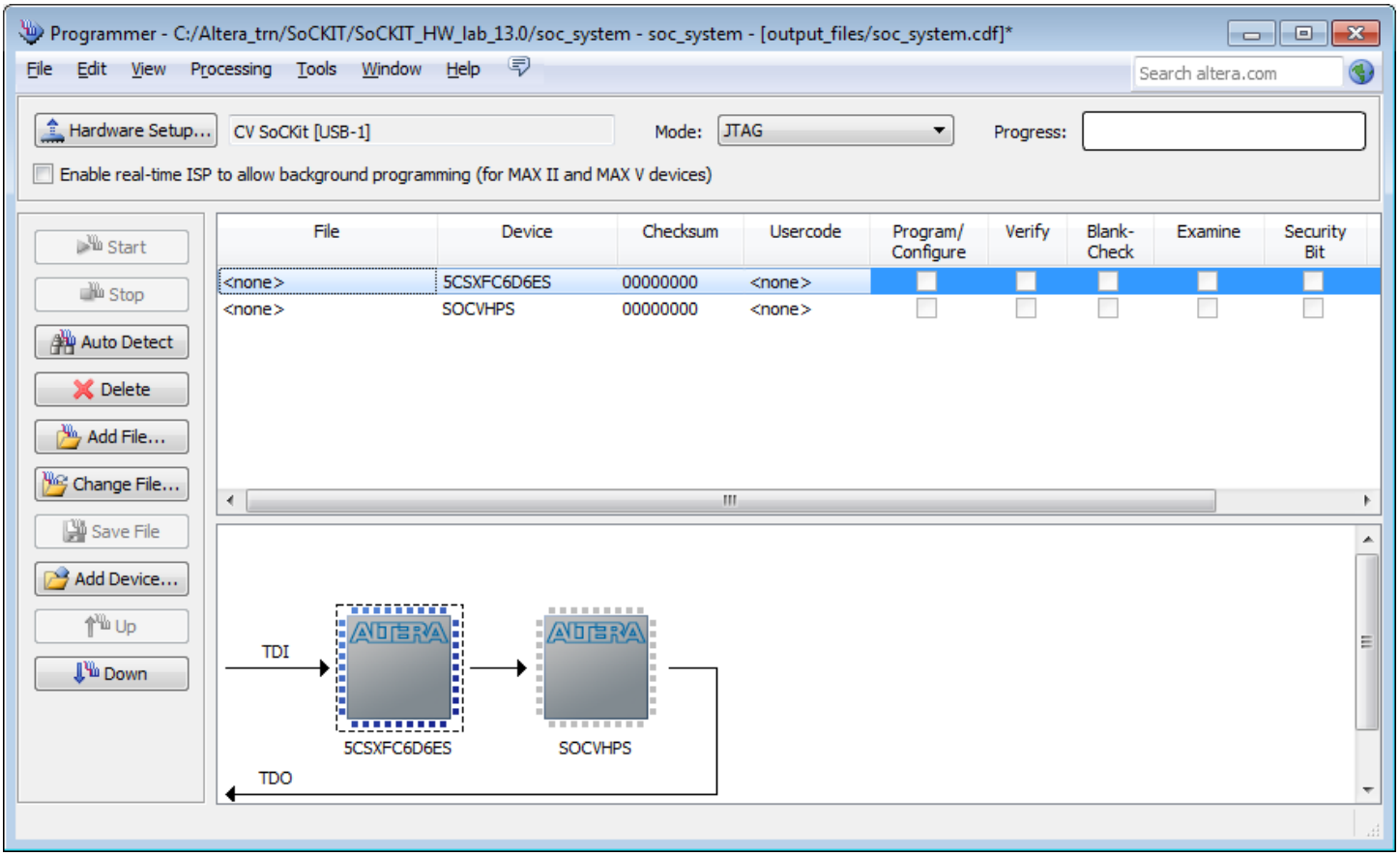
- Select the correct device: "**5CSXFC6D6ES**" and then **OK**



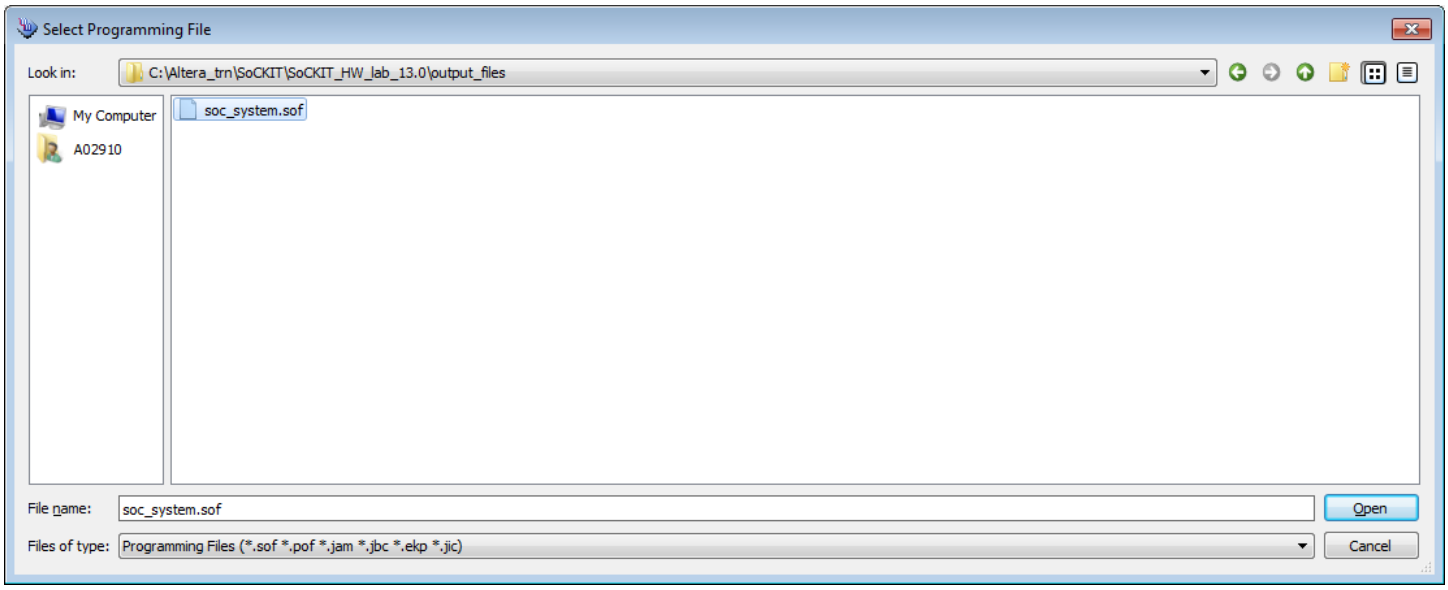
- Select Yes:



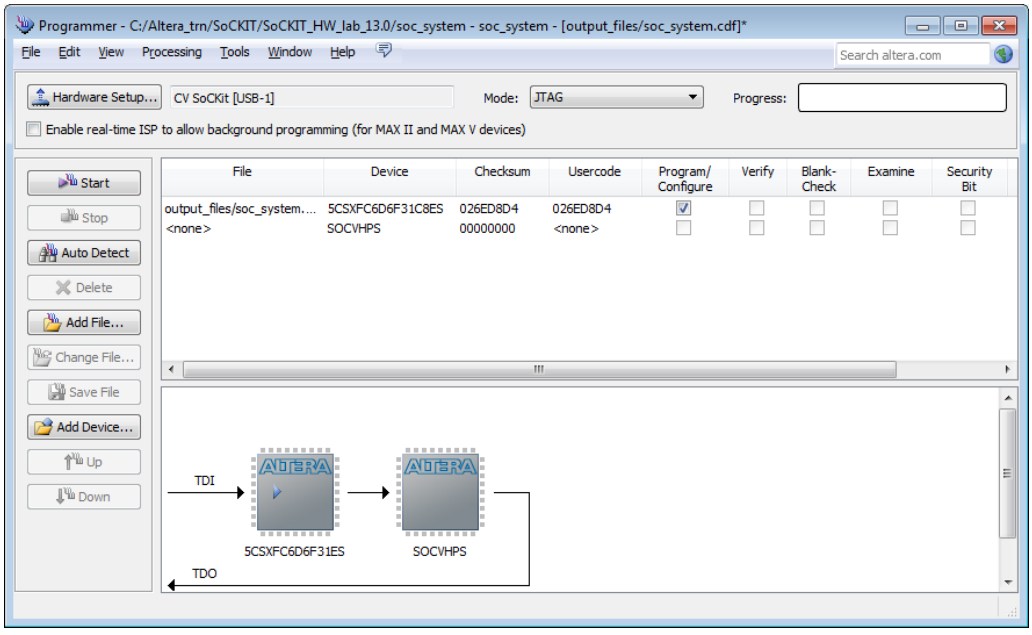
- The Programming window should now appear as shown:



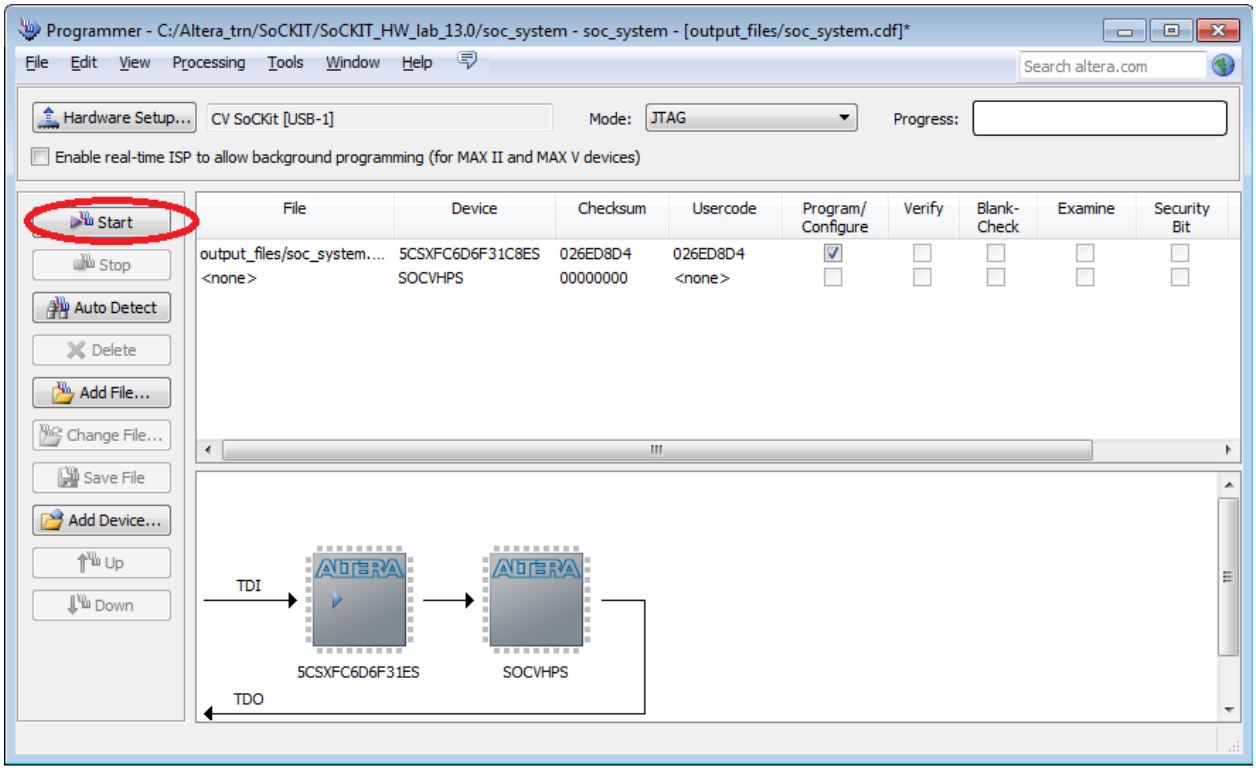
- Select the file row for the 5CSXFC6D6ES
- Select, **Add File**
- Select the soc_system.sof , as shown in the .\output_files directory.



- Select **Open**
- Select the checkbox in the program/configure column and a check will appear.
- The window should then now look as shown **(If not, then delete the extra 5CSXFCD6ES device):**



- Press the Start Button as seen below to program the FPGA.



After programming the FPGA the progress indicator should indicate 100% complete (as seen above) There should be no error messages.

6.2 Executing System Console Scripts

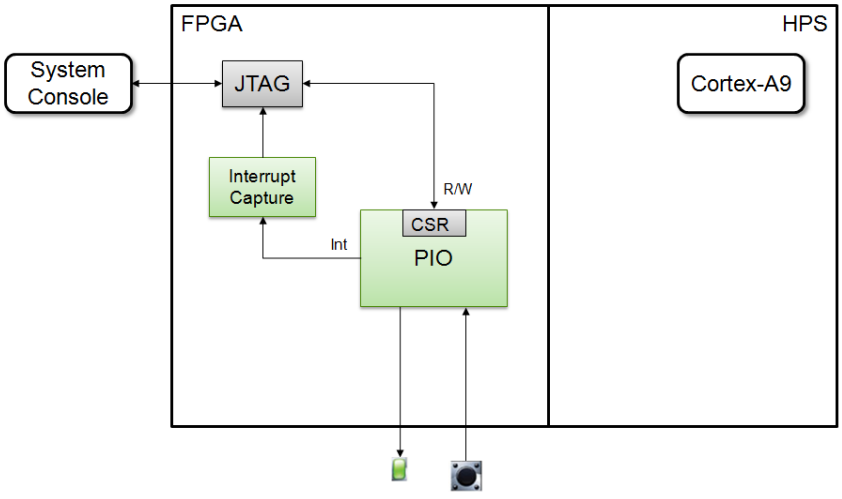
There are different ways to debug a design. Since Qsys was utilized to build the system this can be completed by using System Console. System Console is a low level hardware debug tool that is built with Tcl and runs Tcl scripts and commands.

A functional test of the components instantiated in the FPGA will be realized with System Console. The software lab will also utilize system console and the ARM DS-5 tool to cross-trigger from the DS-5 to the FPGA and FPGA to DS-5.

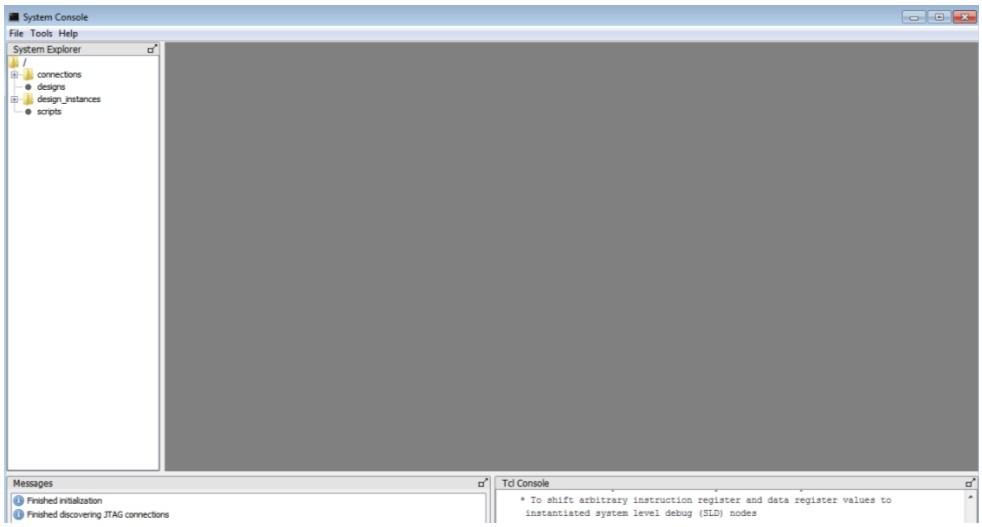
When creating and debugging SoC based systems, the hardware Engineer will want to validate the peripherals and any Qsys IP that has been created on the FPGA side. This will be shown in the following steps using System Console.

Basic System Console Test

This diagram shows an overview of the interface from System Console to the Qsys design that includes the PIO registers (button_pio & led_pio). A set of Tcl commands will be executed from the System Console to read from the system registers associated with the address of the push buttons (KEY3-KEY0) and then write to the system register associated the LEDs. Depending upon the KEYs read, this write will then illuminate the associated LED by bit position.

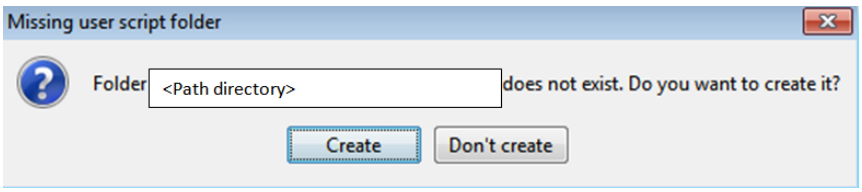


- To run System Console in Quartus II, select: **Tools** -> **Qsys**. This will open the Qsys window.
- Select the **soc_system. qsys** file so that your Qsys design will open.
- In Qsys, select: **Tools** -> **System Console**.
- A new window will now appear as shown:

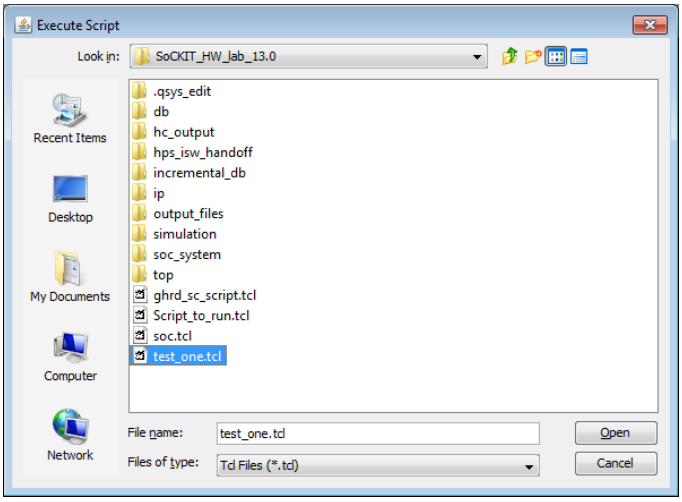


Although the Tcl commands can be executed from the Tcl Console command line, a script was created to automate the process.

- To run the script, select: **File** -> **Execute Script**.
- There will be a pop up window that appears that looks as follows:
- Select: **Don't Create**.

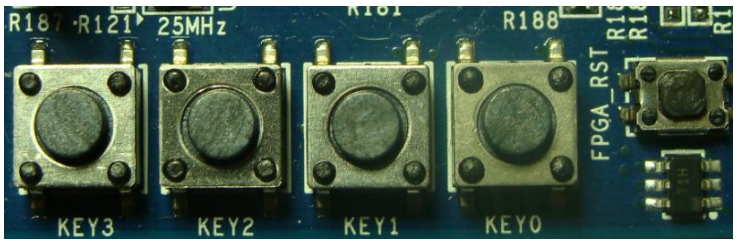


- Scroll to the correct path where you un-archived the lab files, and select **test_one.tcl**.



- **Do not** select Open yet.

- Before selecting Open, there are various push button combinations to try. The FPGA push buttons are the buttons on the bottom right hand side of the board.



- **Hold down push button KEY0 and then select Open**.
- You should also see the following hex numbers: **0x0e 0x00**.
- In the messages window, **System Console test done** will appear.
- **Release** the **KEY0** button.
- **LED0** will not illuminate

- Other combinations are possible when executing the **test_one.tcl** script:

Press KEY(s) then run **test_one.tcl**	Result for LEDs and System Console
Hold down KEY1 and run the test_one.tcl script. Release the buttons when the words System Console test done in the Window show up	1101 = 0x0d (LED1 will not illuminate)
Hold down KEY1 and **KEY0** at the same time and then run test_one.tcl..	1100 = 0x0c (LED0 & LED1 will not illuminate)
Do not select any of the **KEYs** and then run test_one.tcl.	1111 = 0x0f (All LEDs are illuminated)

- Close the System Console window, select: **File** -> **Exit**

If the design was compiled in Quartus and the System Console scripts were not run successfully, precompiled files can be extracted from the "precompiled_files_for_system_console_module" directory. Copy the .jdi and .sof files to the output_directory. Copy the .sopcinfo file to the "SoCkit_HW_lab_13.0" directory. Redo sections 6.1 and 6.2

Congratulations!

You have just run the System Console debugging window.

6.3 Experiments with the System Console Window (Optional)

- **Open** the **test_one.tcl** file with an **editor** and take a look at the code.
- To perform any commands in the Tcl Console you will have to type in the following commands:

  % **set AvailableServices [get_service_types]**
  % **set jtag_master [lindex [get_service_paths master] 0]**
  % **open_service master $jtag_master**

- master_write_8 $jtag_master 0x1_0040 $CurSwitch

  You can change the value of $CurSwitch to be a value of 4, for example so that it now looks for
  % **master_write_8 $jtag_master 0x1_0040 4**

- This command writes a 4 (0100) to the address of the FPGAs LEDs. Therefore, only LED2 will be illuminated. (The KEY(s) are active low, therefore; the reason for the difference when reading from the KEY(s) address and writing to the LED(s) address.
- When you have finished close the service with the following command:
  % **close_service master $jtag_master**

System Console is an extremely valuable troubleshooting/debug tool that allows you to create graphical user interfaces called dashboards. These dashboards can interact with the Qsys IP on the device. Examples of dashboards that have been created for debugging include the Transceiver Tool Kit and the External Memory Interface Toolkit.

Congratulations!

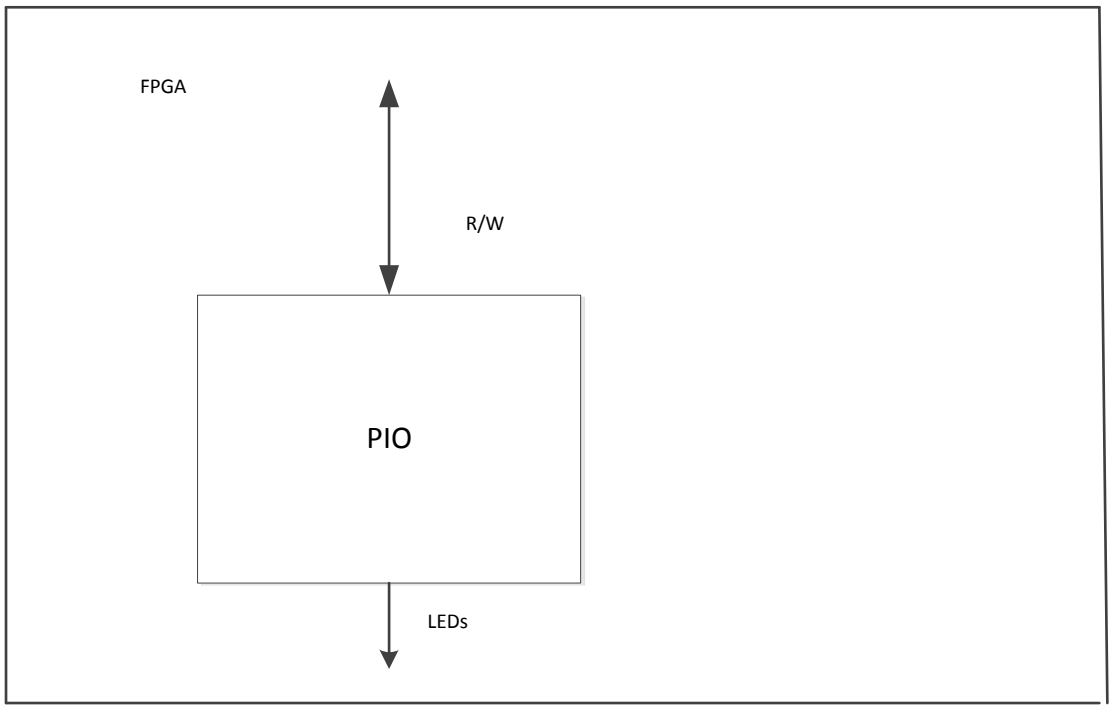
You have just completed using System Console.

MODULE 7. Hardware Validation with Simulation (Do at home Exercise)

This module describes how to simulate the FPGA hardware you created in Qsys.

Simulation allows the design to be verified before it is programmed into the device. Quartus II allows both RTL and gate level simulation. RTL simulation is a cycle-accurate simulation and will be covered in the Module.

The RTL simulation can consist of the entire design or sub-components of the design. In this module the RTL simulation will only include the PIO component in the system.



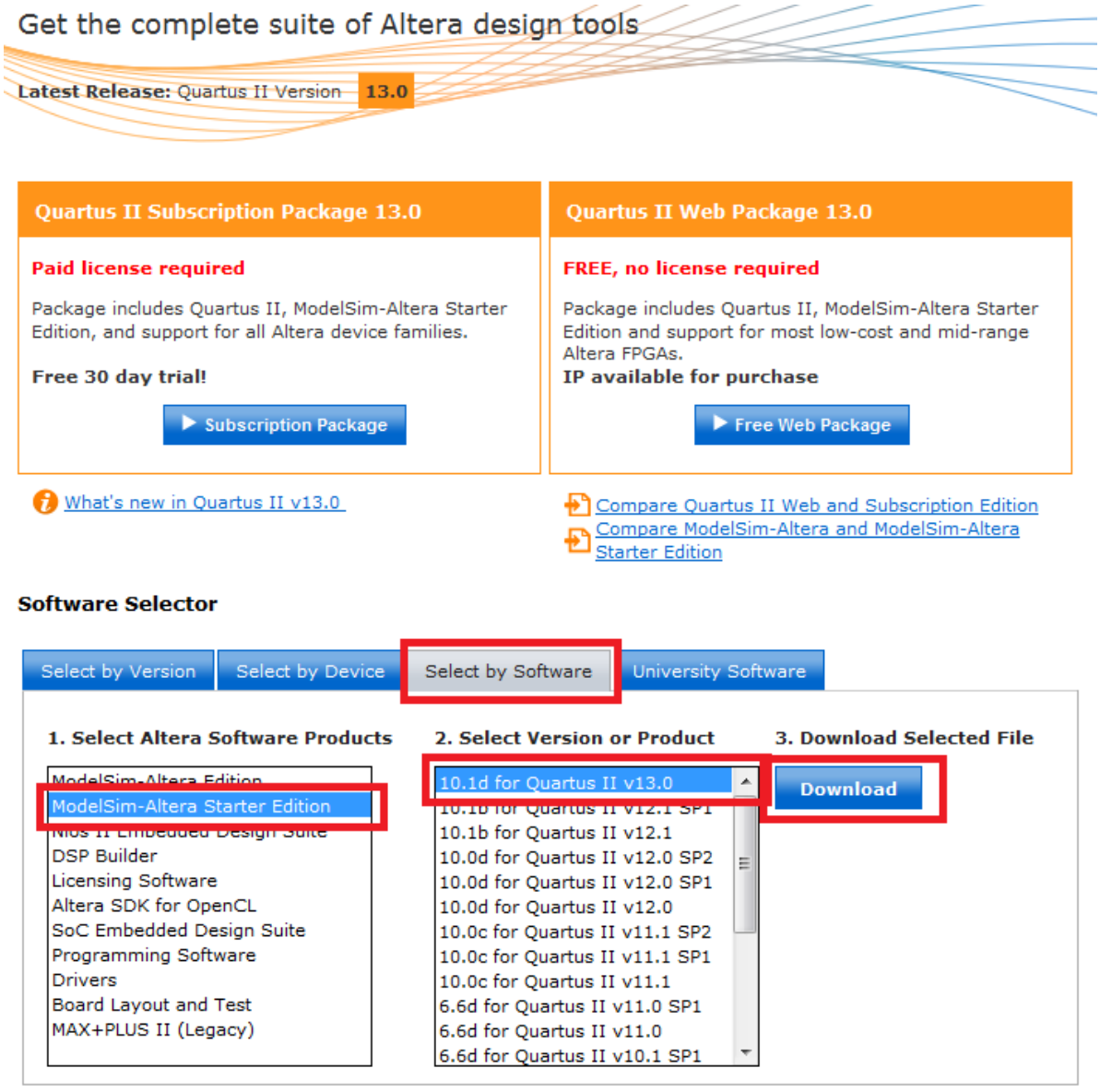
The PIO uses Avalon signals to toggle on/off the LEDs.

7.1 Installing ModelSim-Altera

1. ModelSim-Altera (simulator software) is needed to be installed before this Module can be completed. ModelSim-Altera can be downloaded from the following URL:

   https://www.altera.com/download/sw/dnl-sw-index.jsp

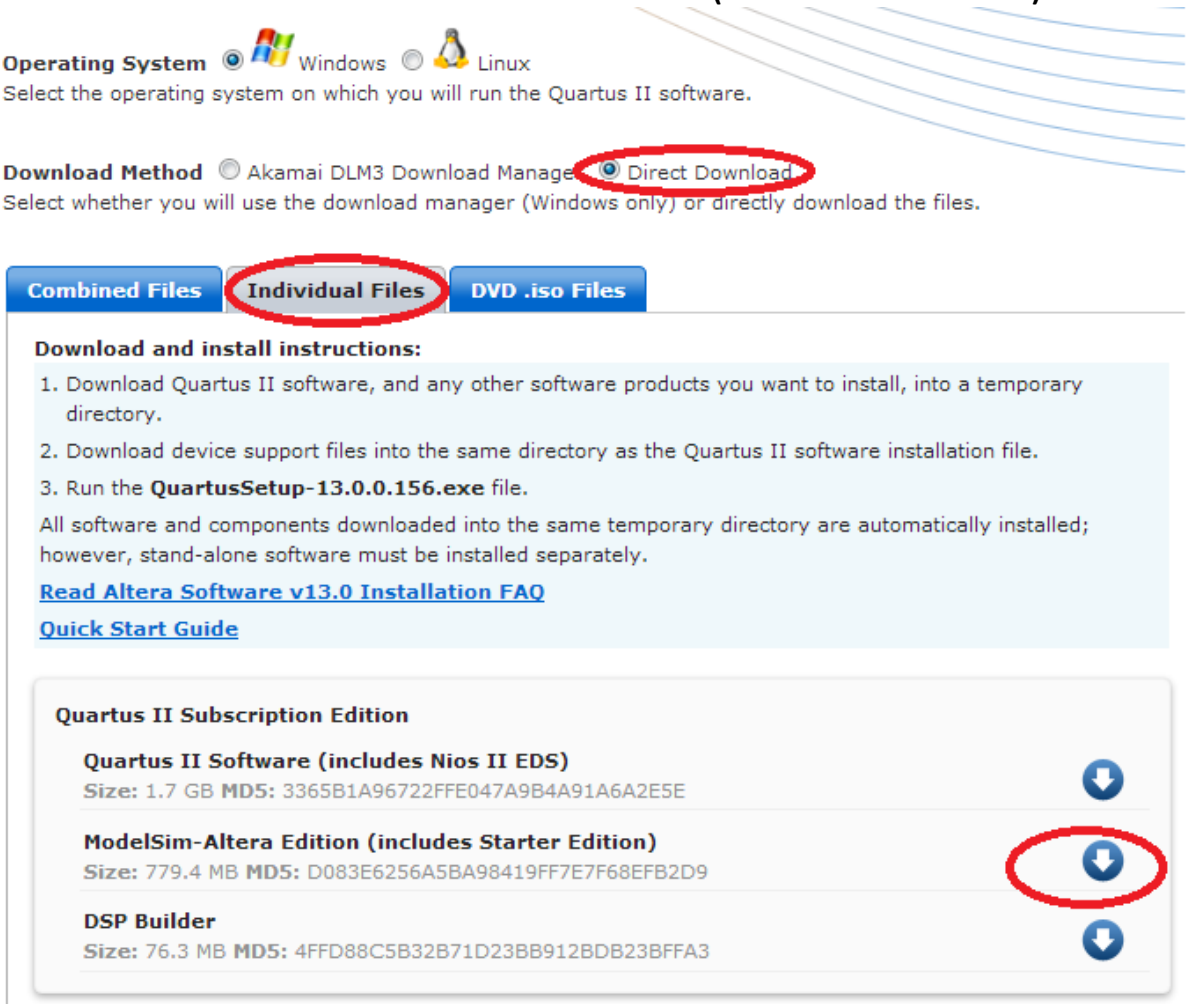
2. Select: **Select by Software** under Software Selector
3. Select: **ModelSIM-Altera Starter Edition**
4. Select: **Select Version or Product**: 10.1d for Quartus II v13.0
5. Select: **Download**



6. Select **Direct Download**

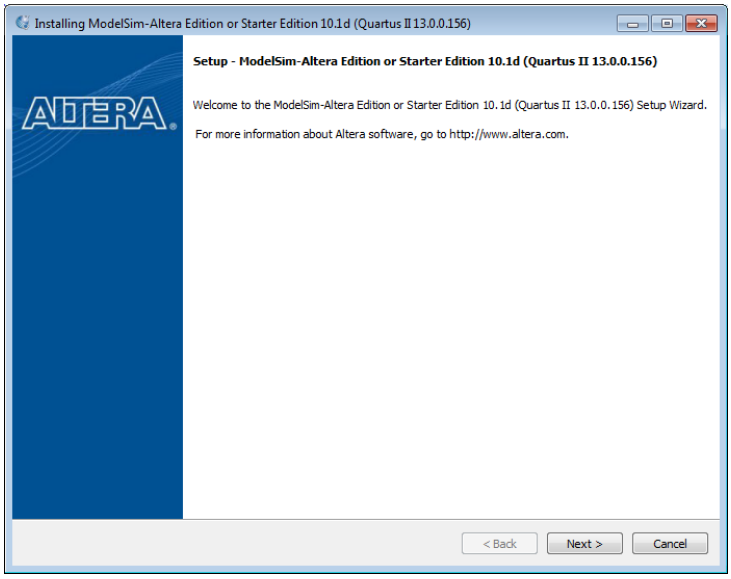
7. Select: **Individual Files**

8. Select: **Down Arrow** to download **ModelSIM-Altera Edition (includes Starter Edition)**

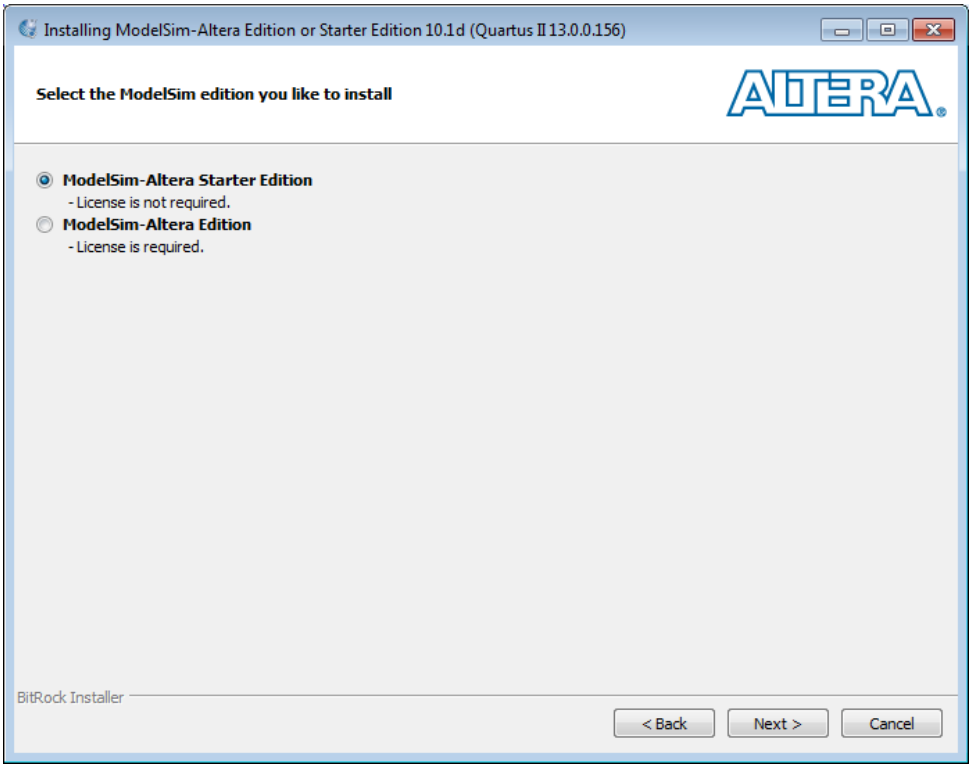


9. Go to the download directory and run the executable: **ModelSIMSetup.exe**

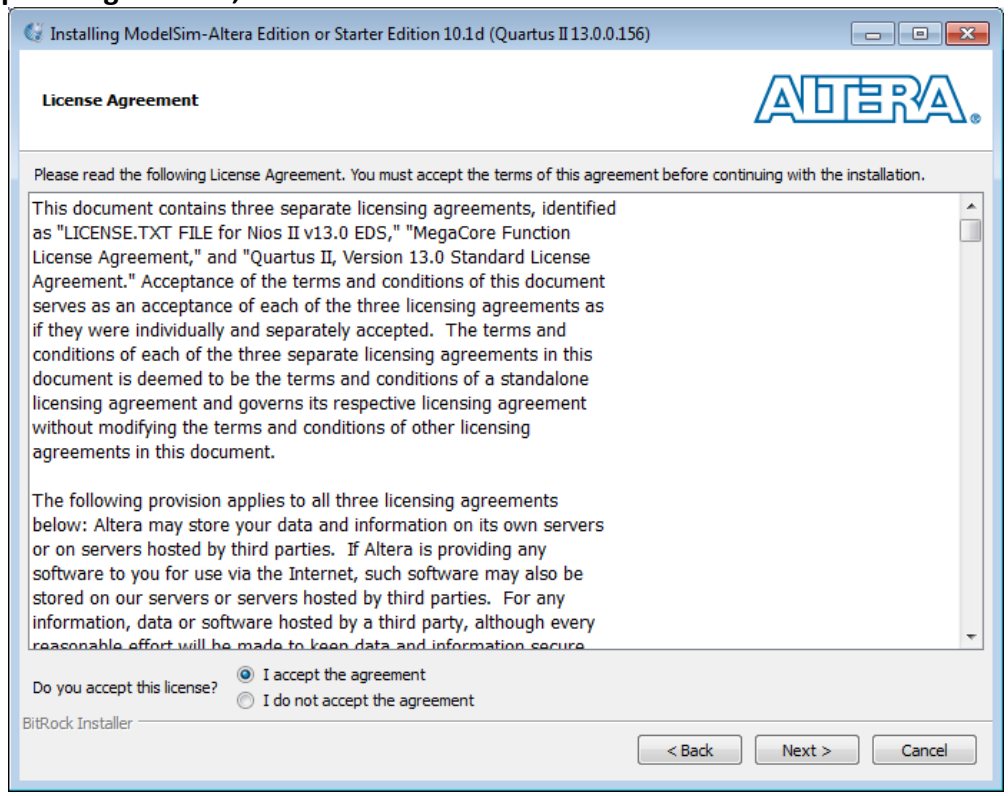
10. Select **NEXT** as shown in the following screenshot



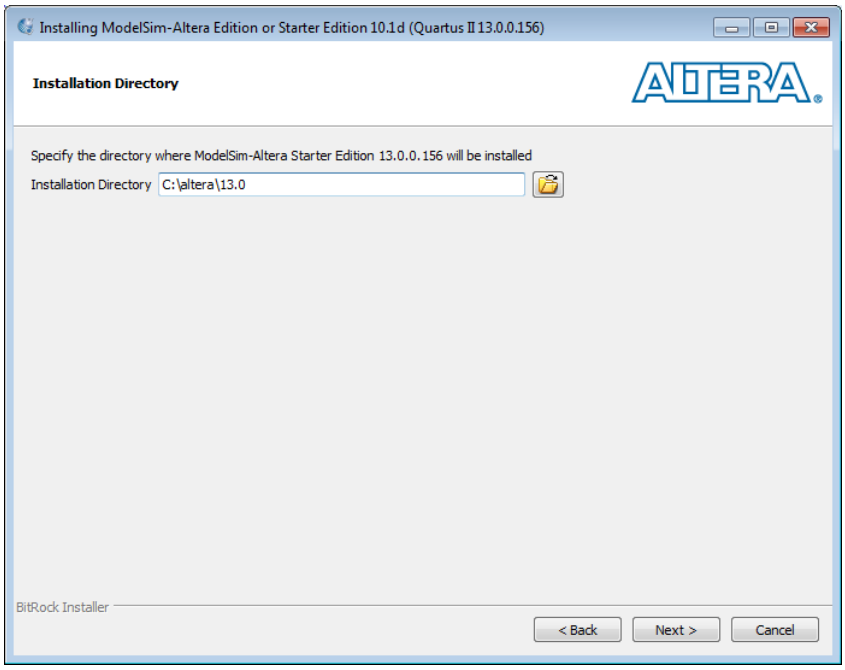
11. Select **ModelSim-Altera Starter Edition, then Next**



12. ***Select*: I accept the agreement,** then **Next**



13. Navigate to the Select Products page and select ModelSim-Altera Starter Edition (Free)



You will only need to install ModelSim-Altera Starter, since Quartus II was installed in a previous step.

7.2 Set EDA Tool Settings in Quartus II

1. **Open Quartus II**, select: **Start** -> **All Programs** -> **Altera 13.0** -> **Quartus II 13.0** -> **Quartus II 13.0**

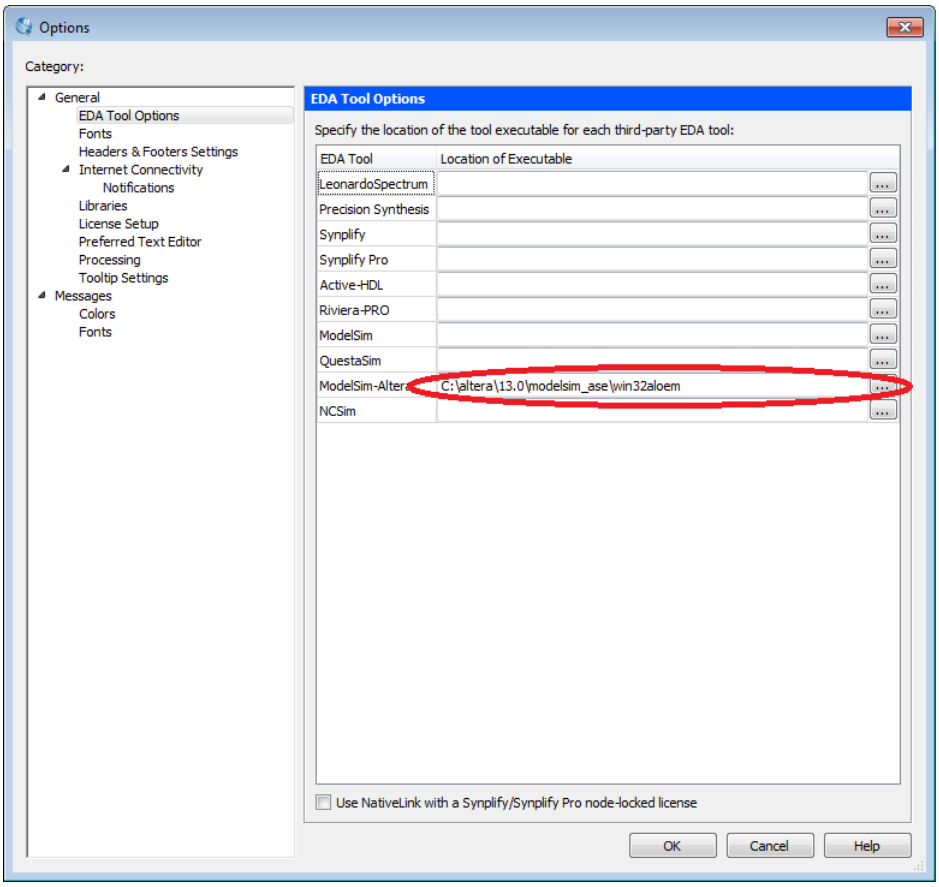
2. Open your design, select: **File** -> **Recent Projects** and select:
   C:\Altera_trn\SoCKit\SoCKit_HW_lab_13.0\soc_system.qpf

3. Setup Nativelink for ModelSim-Altera in Quartus II. Select: **Tools** -> **Options**

4. Under the **Category**: **General**: select: **EDA Tool Options**

5. Next, select the browse button  and browse to your ModelSIM Installation Directory and select the path to win32aloem

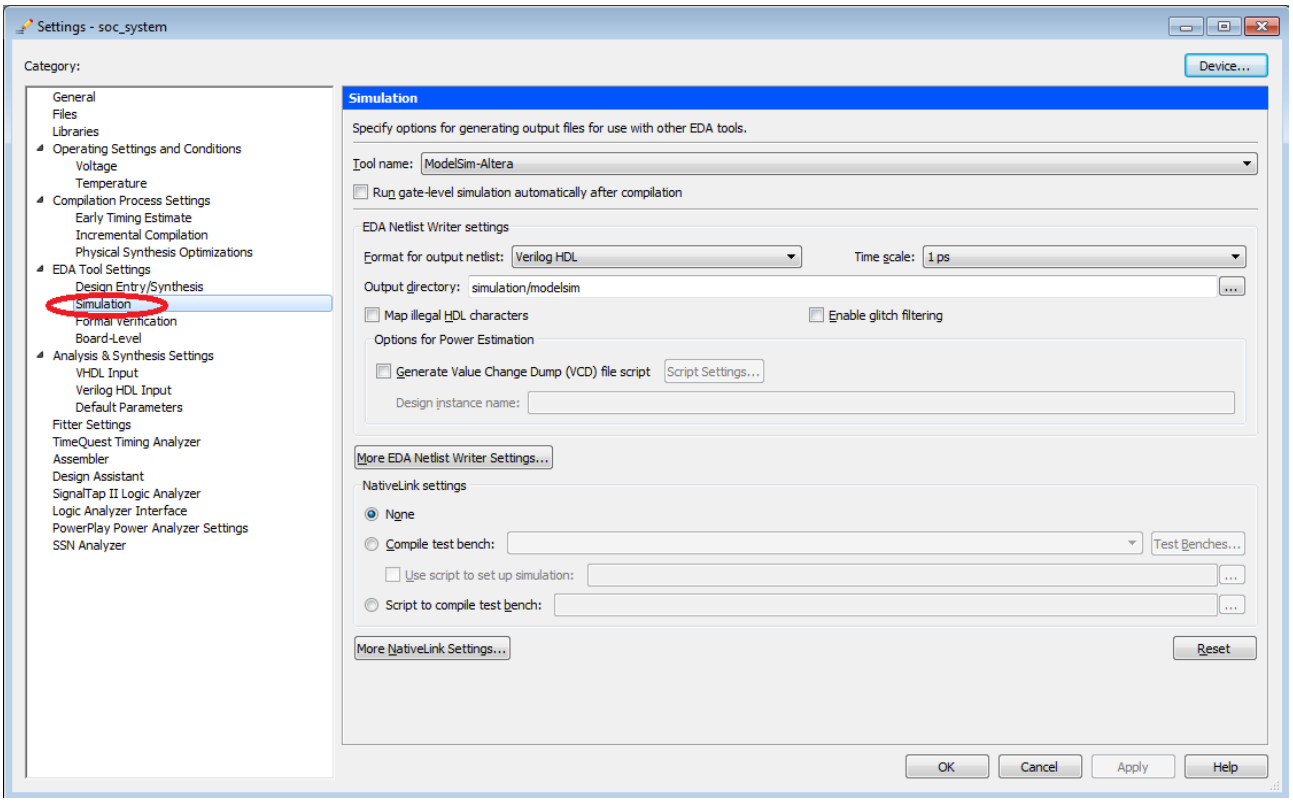
   C:\altera\13.0\modelsim_ase\win32aloem



6. Select **OK**.

7. In Quartus II, select: **Assignments** -> **Settings**

8. On the left hand side of the window, select the EDA Tool Settings -> Simulation to the following:



9. Select **OK**.

7.3 Run RTL Simulation

1. In Quartus II, select: **Processing** -> **Start** -> **Analysis and Synthesis**.

2. To run the RTL simulation, in Quartus II, select: **Tools** -> **Run Simulation Tool** -> **RTL Simulation**.
   This step will automate the set up of ModelSim-Altera and compiling code within ModelSim-Altera.

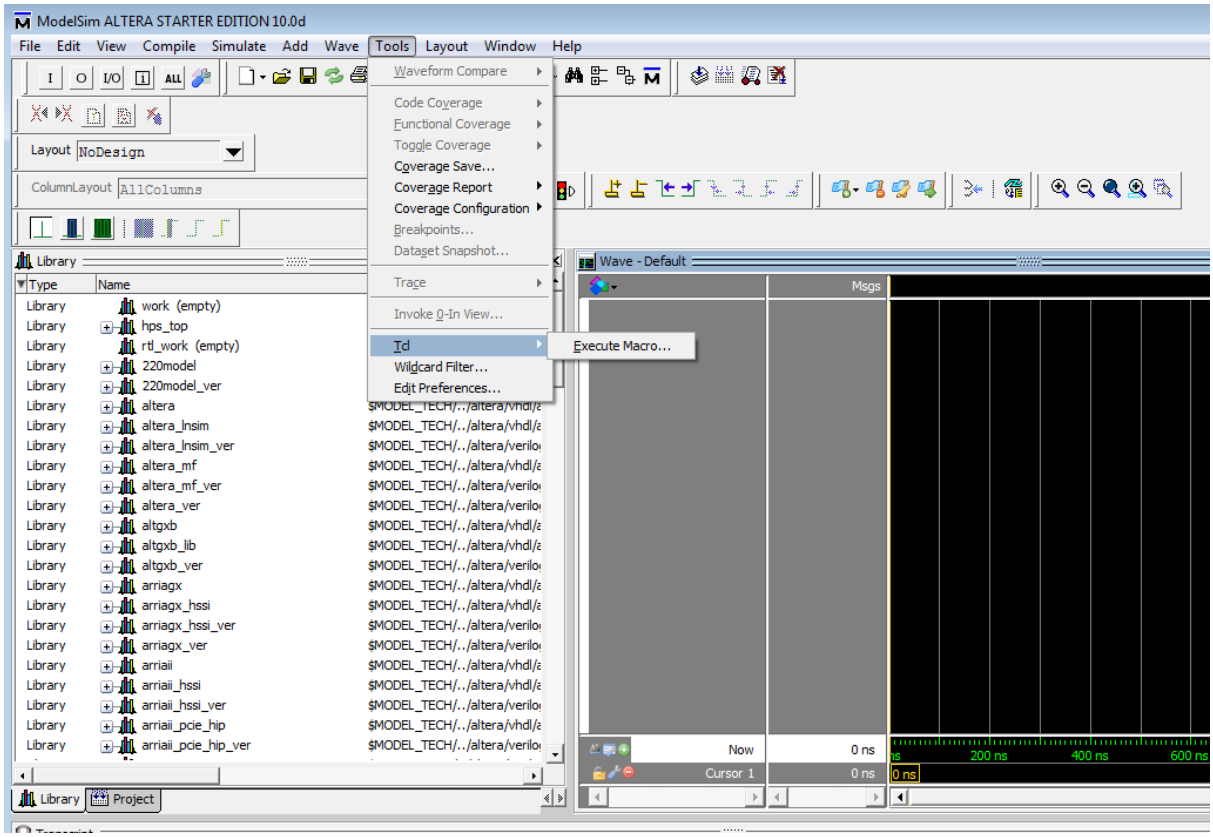
   Please disregard the errors in Modelsim

3. Once the compilation in the Transcript Window is finished, the next step is to run the simulation.

Simulate code

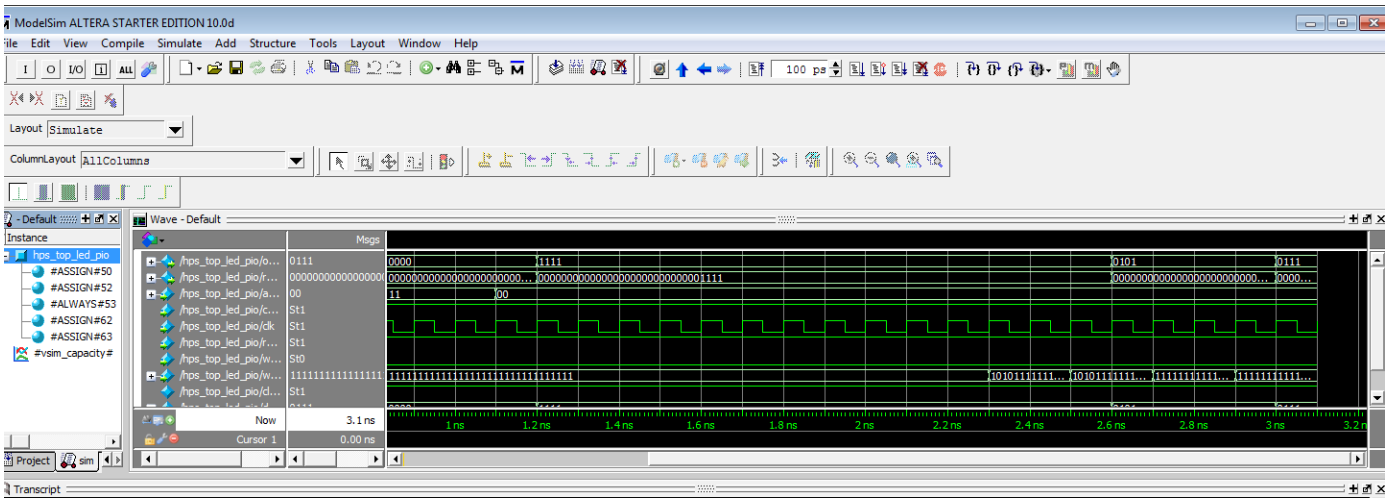
The signals and waveforms can be added manually or via script. A script was created, and it can be executed to automate the simulation.

1. To run the script in ModelSIM select: Tools -> Tcl -> Execute Macro

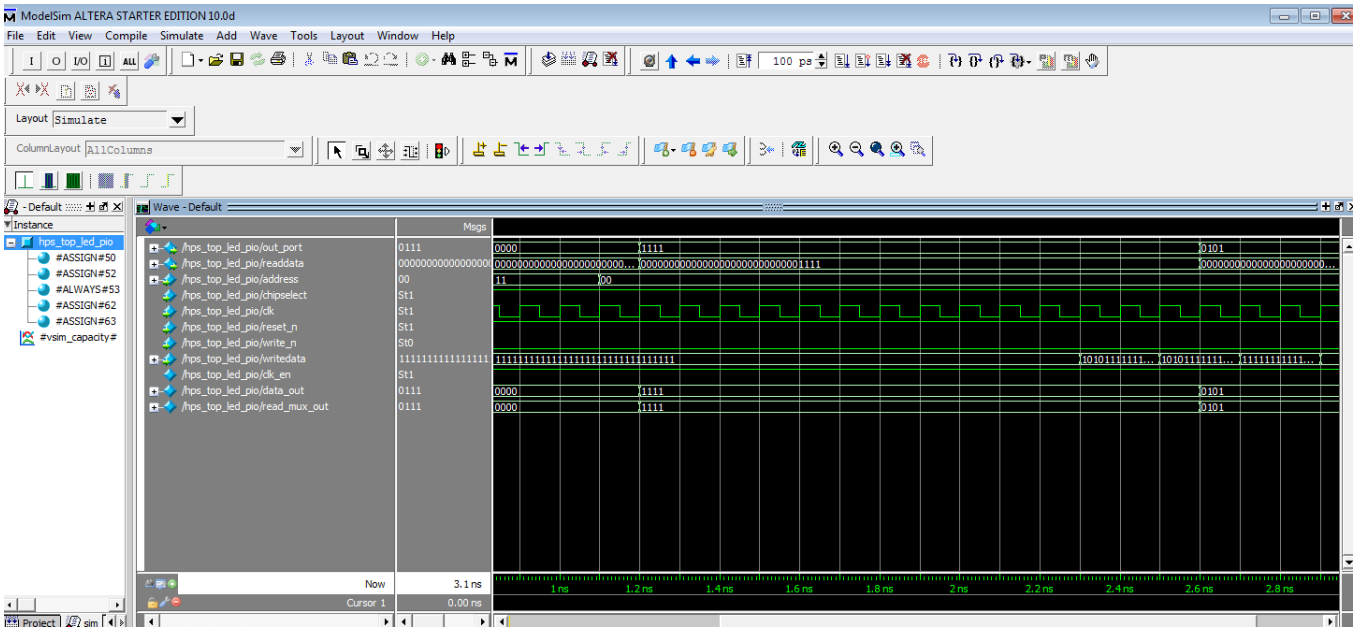


2. Select: C:\Altera_trn\SoCKit\SoCKit_HW_lab_13.0**Script_to_run.tcl.**

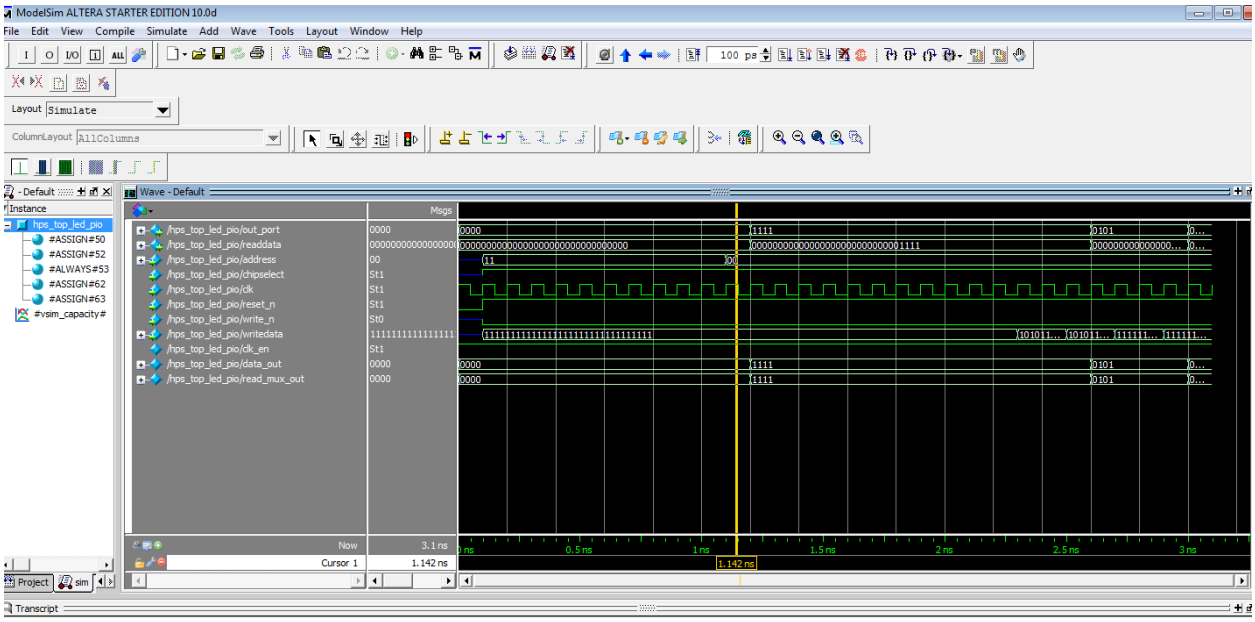
3. The simulation results should now be displayed as:



4. Drag cursor1 in the Wave window so that the signal names can be shown completely:



5. Select: **View** -> **Zoom** -> **Zoom Full** to see the entire simulation:



7.4 Validate simulation

Validation of simulation is important to ensure that the results in the designs are correct. For the PIO component, there are input and output signals. The input signals are chipselect, clk, reset_n, write_n, address[1..0], and writedata[31..0]. The output signals are out_port[3..0] and readdata[31..0] as seen below.

Compare the simulation waveform results to the following comments

Inputs:

1. Chipselect (soc_system_led_pio/chipselect) is set to high for the entire simulation; therefore, the **led_pio** component is enabled.
2. Clock (soc_system_led_pio/clk) is set to toggle, since is a clock signal.
3. Reset_n (soc_system_led_pio/reset_n) is first initialized as 0 and then high (1) to ensure the reset is correct.
4. Write_n (soc_system_led_pio/write_n) is set to 0 since it is an inverted write signal.
5. Address (**soc_system**_led_pio/address_n) is set first to 11 (to show that there is no output when the address is 11), and then 00 (to show there is output with this address)
6. Writedata (soc_system_led_pio/writedata) is a 32 bit Avalon number that is written to the PIO.

Outputs:

1. out_port (soc_system _led_pio/out_port) is the 4 bits to the LED. The 1111 signal, which comes for the writedata, occurs one clock cycle after the address is set to 00. The 1111 signal then changes to 0101, after one clock cycle of the writedata. This shows that LED does toggle as expected.
2. Readdata (soc_system _led_pio/readdata) is the read data. This PIO does not use this signal, because the PIO is set to be an output.

Congratulations!

You have just run a simulation.

MODULE 8. Taking the next Step

Altera has a number of resource available to assist you in further product development at www.altera.com/embedded

Some of the resources available are:

Get more information about Qsys:

System Design with Qsys Reference Manual
http://www.altera.com/literature/hb/qts/qsys_intro.pdf

Creating custom Qsys Components
http://www.altera.com/literature/hb/qts/qsys_components.pdf

Visit the rocketboards.org community web site

http://www.rocketboards.org/

Arrow SoCKit Evaluation Board support site
http://www.rocketboards.org/foswiki/Documentation/ArrowSoCKitEvaluationBoard

Altera SoC Development Board support site
http://www.rocketboards.org/foswiki/Documentation/AlteraSoCDevelopmentBoard

Get more information about the SoC HPS

Hard Processor System Technical Reference Manual
http://www.altera.com/literature/hb/cyclone-v/cv_5v4.pdf

Get more information about the SoC Embedded Design Tools

Embedded Software for the Cortex-A9 MPCore Processor
http://www.altera.com/devices/processor/arm/cortex-a9/software/proc-a9-embedded-software.html

Get additional SoC training (discounted from $695 per course to $99 for workshop attendees)

Designing with an ARM based SoC
http://www.altera.com/education/training/courses/ISOC101

Developing Software for an ARM based SoC
http://www.altera.com/education/training/courses/ISOC102

For all resources visit www.altera.com/embedded