

SoCKit

GO INTEGRATE

SoC Hardware Lab Instructions

Version 14.1

02/03/2015

Tutorial

Table of Contents

OVERVIEW 3

MODULE 1. Getting Started6

1.1	Acquiring Cyclone V SoCKit.....	6
1.2	Install the Altera Design Software.....	7
1.3	Extract the SoCKit Lab Files.	15
1.4	Get the Cyclone V SoCKit ready for the Labs (Complete this at the Workshop)	16
1.5	Install the USB Blaster II Device Driver (Complete this at the Workshop)	19

MODULE 2. Examine the System Design22

2.1	System Architecture	22
2.2	Examine the System Tool Flow.....	23
2.3	Examine Arrow's Cyclone V SoCKit.....	24
2.4	Block Diagram of the SoCKit.....	25

MODULE 3. Set up the Quartus II project26

3.1	Launch Quartus II Project	26
-----	---------------------------------	----

MODULE 4. Build the Qsys System.....28

4.1	Launch Qsys.....	28
4.2	Build the Qsys System	29
4.2.1	Configure the FPGA Interfaces for the HPS	31
4.2.2	Configure HPS' Peripheral Pin Multiplexing (MAC, NAND, QSPI, SDIO, USB).....	35
4.2.3	Configure HPS Clocks.....	40
4.2.4	Configure SDRAM (The HPS External Memory Interface)	49
4.2.5	Configure LED PIO.....	55
4.2.6	Configure Button PIO	57
4.3	System Configuration	59
4.3.1	Connect HPS interfaces to FPGA Peripherals.....	59
4.3.2	Set IRQs	61
4.3.3	Set Base Addresses	63
4.4	Generate the System	69

MODULE 5. Complete the Quartus II Project	73
5.1 Set up the Quartus II project to point to the correct files	73
5.2 Analysis and Synthesis	77
5.3 Adding Pin assignments.....	77
5.4 Compile (Optional step for this lab).....	78
MODULE 6. Hardware Debug Flow (System Console)	85
6.1 Downloading and Programming FPGA.....	80
6.2 Executing System Console Scripts	85
6.3 Experiments with the System Console Window (Optional)	89
MODULE 7. Hardware Validation with Simulation (Do at home Exercise)	90
7.1 Installing ModelSim-Altera	91
7.2 Set EDA Tool Settings in Quartus II	95
7.3 Run RTL Simulation	96
7.4 Validate simulation	99
MODULE 8. Taking the next Step.....	100

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**.

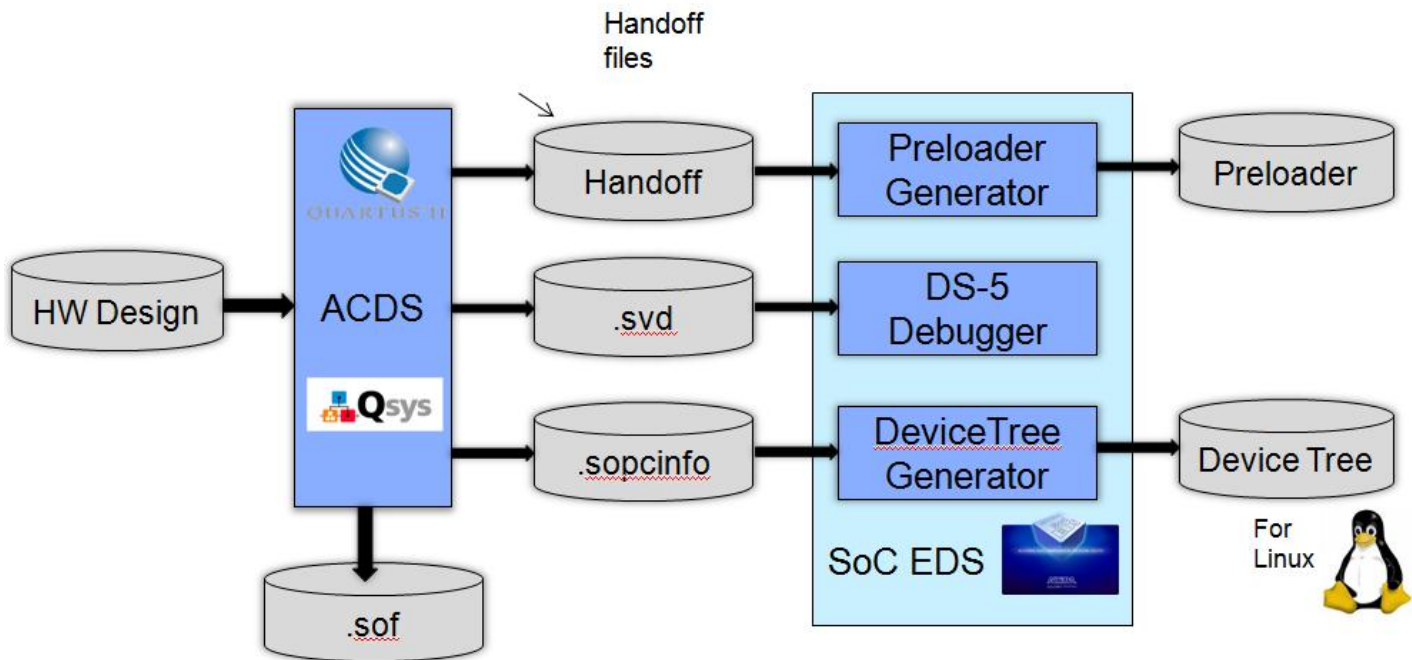
These two Altera FPGA development tools will generate the **files** needed for the **transfer of design information** from the **hardware** to the **software** domain. A portion of the hardware modules will create a set of handoff files that are required to build a preloader, a system register set (including 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** v14.1 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 v14.1.** — 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.*

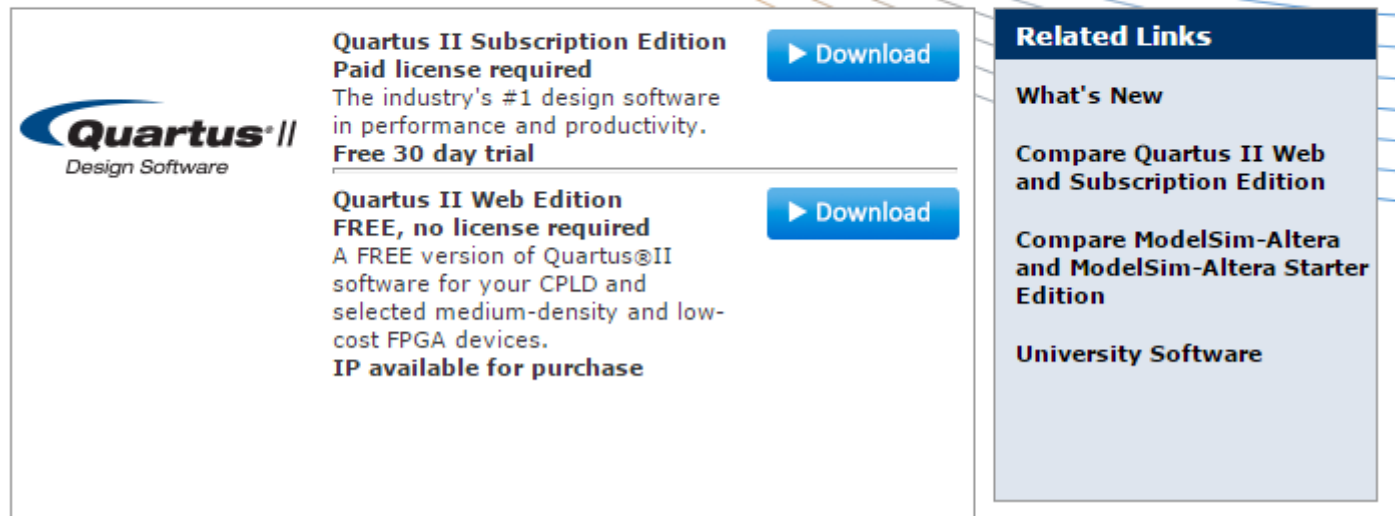
- Go to the Altera Download web page at <https://www.altera.com/download/dnl-index.jsp>


Download Center

[Home](#) > [Support](#) > [Downloads](#) > [Download Center](#)

Latest Release: Quartus II Version **14.1**

[Which version of the Quartus II software supports my device?](#)





Quartus II Subscription Edition
Paid license required
 The industry's #1 design software in performance and productivity.
Free 30 day trial

Quartus II Web Edition
FREE, no license required
 A FREE version of Quartus®II software for your CPLD and selected medium-density and low-cost FPGA devices.
IP available for purchase

▶ Download

▶ Download

Related Links

What's New

Compare Quartus II Web and Subscription Edition

Compare ModelSim-Altera and ModelSim-Altera Starter Edition

University Software

- Select "Quartus II Web Package 14.1. Press the **Free Web Package** button.

- Select **Quartus II Web Edition, Windows:**
- Select the **“Individual Files” Tab**
- Select the **“Download Selected Files” Button**
- **Download** the Quartus II software files onto your computer.

Release date: December, 2014
Latest Release: v14.1
Select release: 14.1

Operating System: Windows Linux

Download Method: Akamai DLM3 Download Manager Direct Download

✓ The Quartus II software version 14.1 supports the following device families: Arria II, Cyclone IV, Cyclone V, MAX II, MAX V, MAX 10 FPGA. [More](#)

Combined Files **Individual Files** **DVD Files** **Additional Software** **Updates**

Download and install instructions: [More](#)
[Read Altera Software v14.1 Installation FAQ](#)
[Quick Start Guide](#)

☒ **Select All**

☒ **Quartus II Web Edition (Free)**
☒ **Quartus II Software (includes Nios II EDS)**
Size: 1.2 GB MD5: FB732633ECB57BE1A73BE45D172918AF UPDATE
☒ **ModelSim-Altera Edition (includes Starter Edition)**
Size: 1.1 GB MD5: C931A4F7F9B4306DD8E8248607993C7C

☒ **Devices**
You must install device support for at least one device family to use the Quartus II software.

☐ **Arria II device support**
Size: 497.7 MB MD5: B329C8FCC2E1315B0E36C11AD41A23F7

☐ **Cyclone IV device support**
Size: 462.7 MB MD5: 599819EBE4DDBFA0B622505B22432E86

☒ **Cyclone V device support**
Size: 1.0 GB MD5: 446D7EE5999226CD3294F890A12C53CC

☐ **MAX II, MAX V device support**
Size: 11.3 MB MD5: C3EDC556AC9770DB2DD63706EECA2654

☐ **MAX 10 FPGA device support**
Size: 289.0 MB MD5: 75F2D4AF1E847FC53AC6B619A35BD2CF

Download Selected Files

- Login to **myAltera** account. Use your **existing login**, or **Create Your myAltera account**.

myAltera Account Sign In
[Home](#) > [myAltera Account Sign In](#)

User Name:

Password: [Forgot Your User Name or Password?](#)

☐ Remember me

Sign In

Don't have an account?

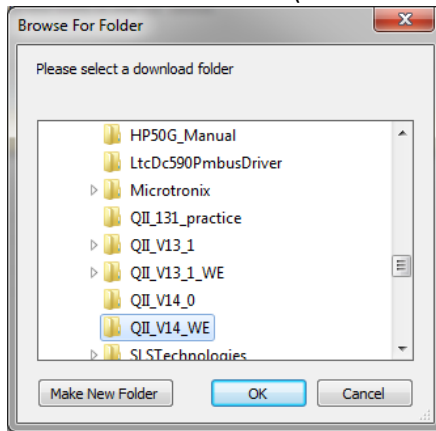
☒ **Create Your myAltera Account**
Your myAltera account allows you to file a service request, register for a class, download software, and more.

Enter your email address:

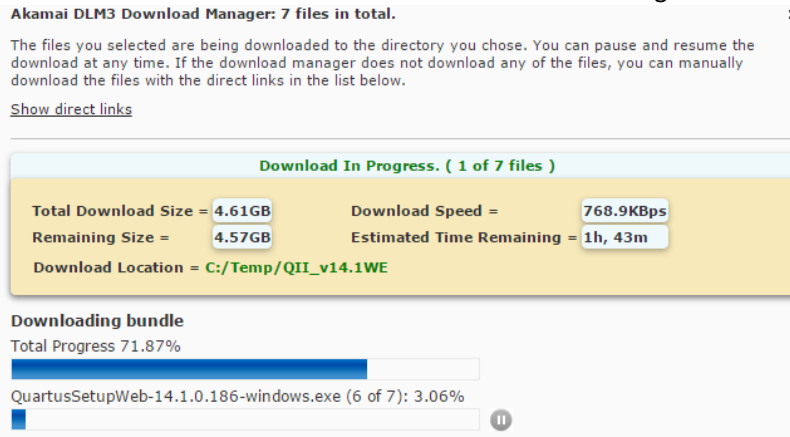
(If your email address already exists in our system we will retrieve the associated information.)

Create Account

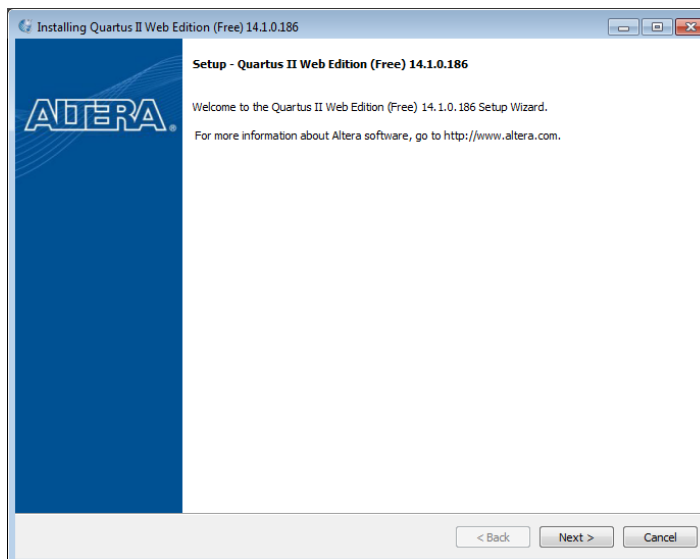
- Select a download folder (Make New Folder for v14.1)



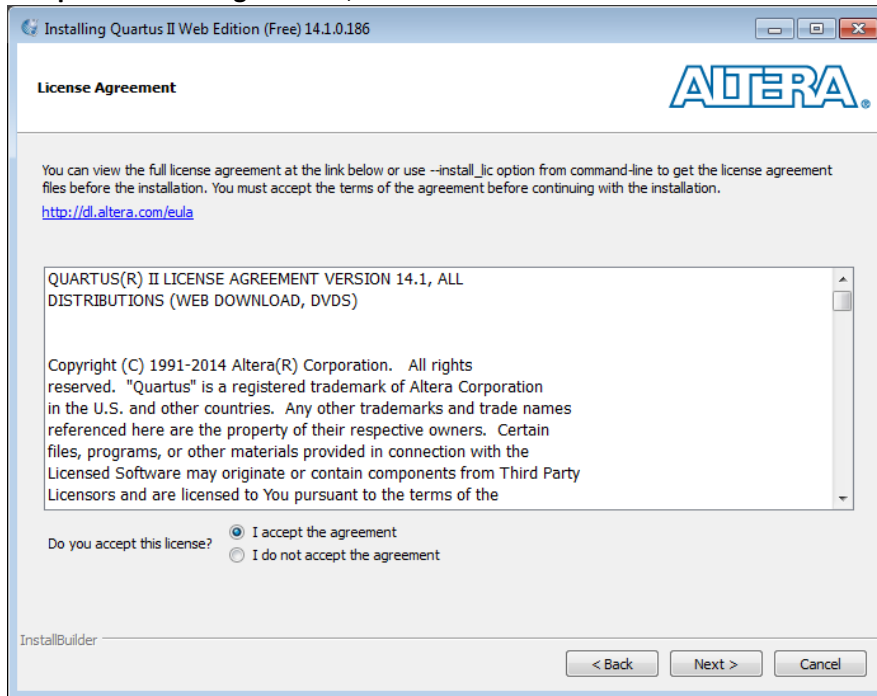
- The files will then be downloaded via the Download Manager



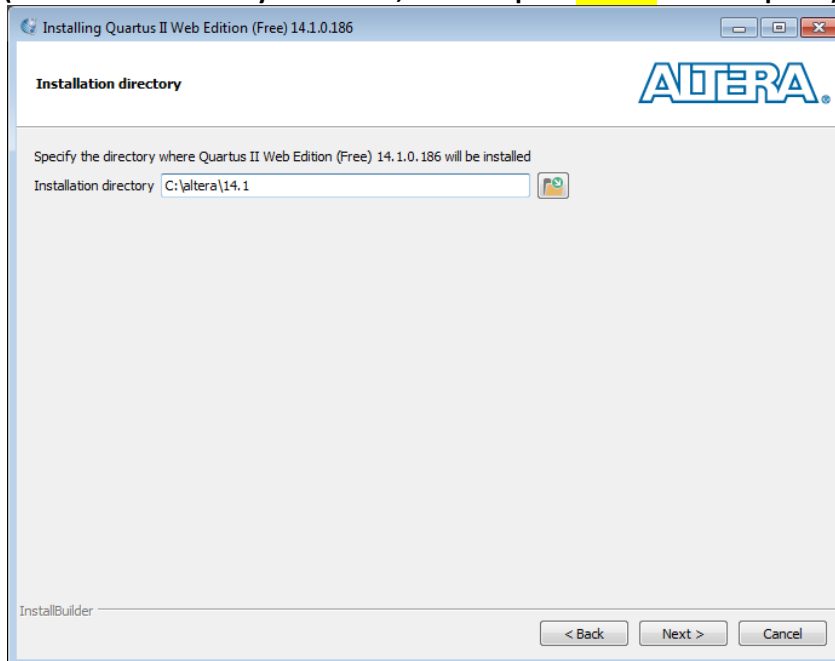
- After the file is downloaded on the computer, select the *.exe file, and install the software.



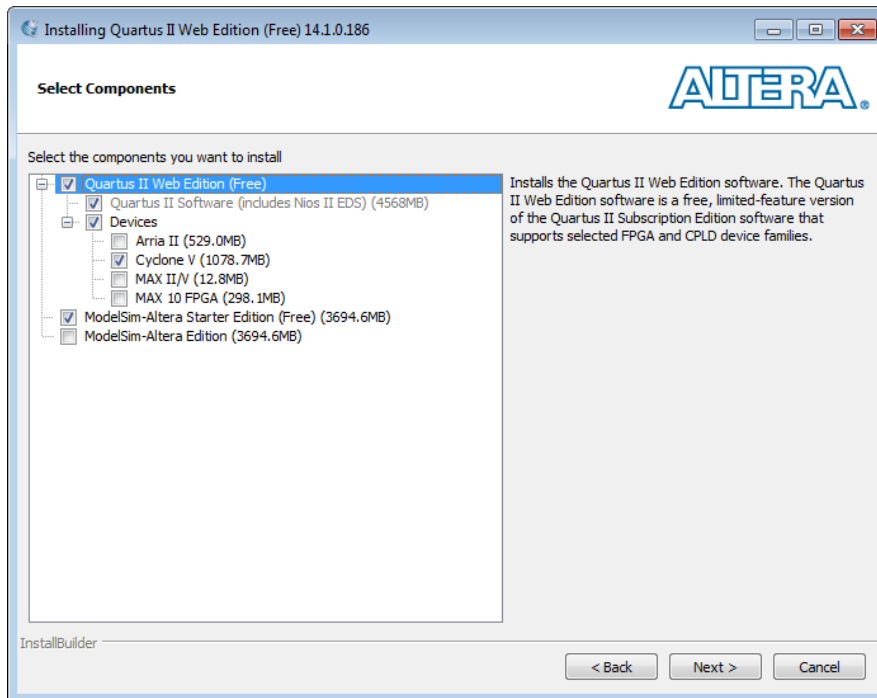
- **Accept the license agreement**, then “Next >”.



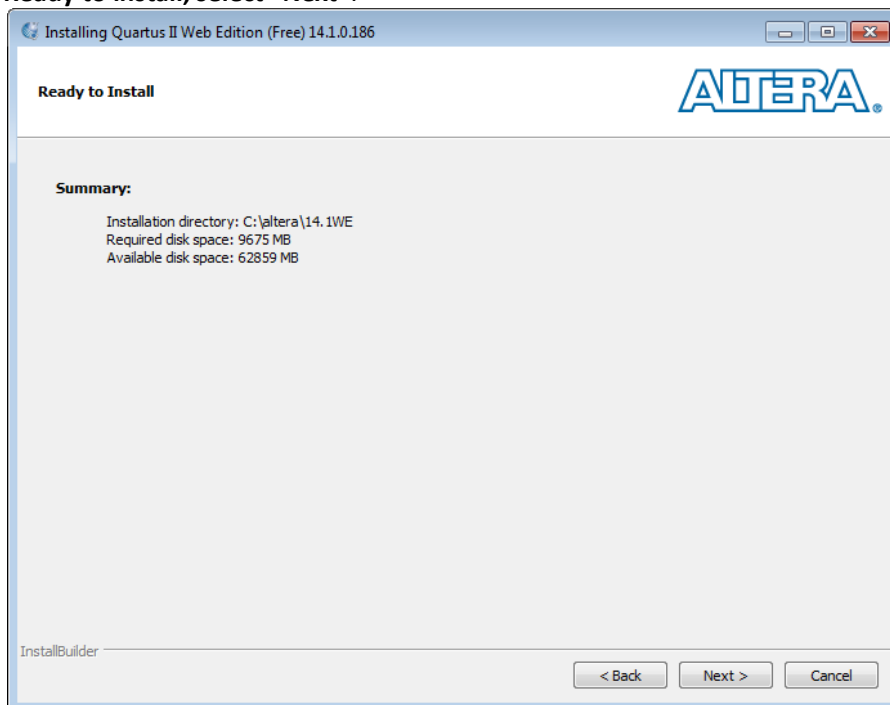
- **Select the default installation directory**, then “Next >”
(If a different directory is selected, then the path **cannot** include spaces).



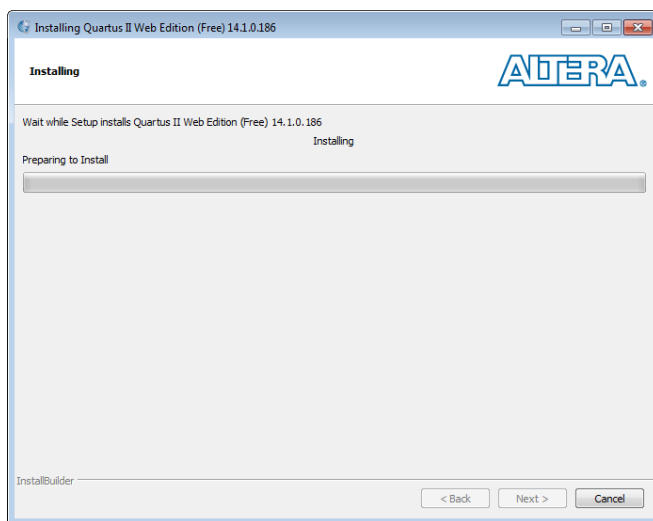
- **Select** Quartus II Web Edition, Cyclone V under Devices and ModelSim-Altera Starter Edition for the installation, then “Next >”



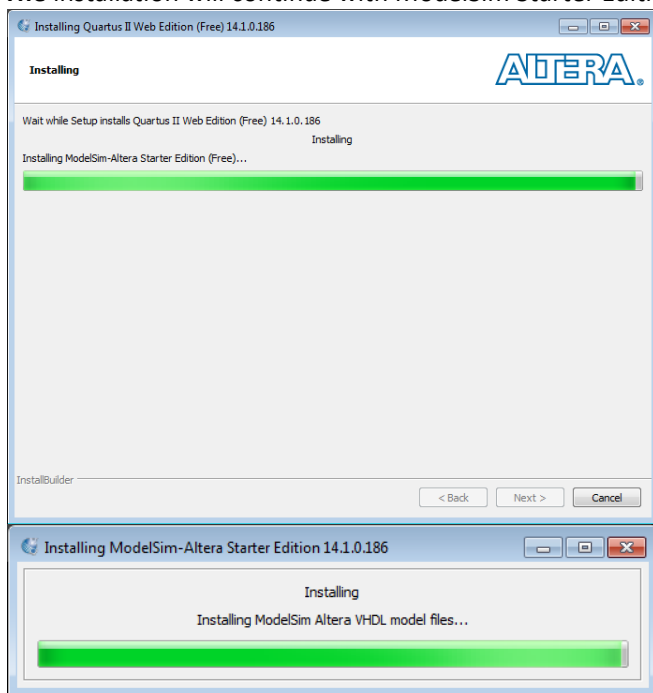
- **Ready to install, select “Next”:**



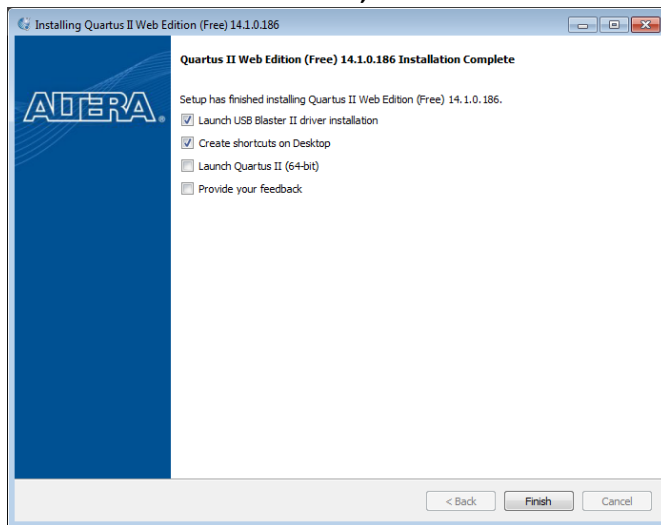
- The installation of QII Web Edition and Cyclone V Device Family will begin:



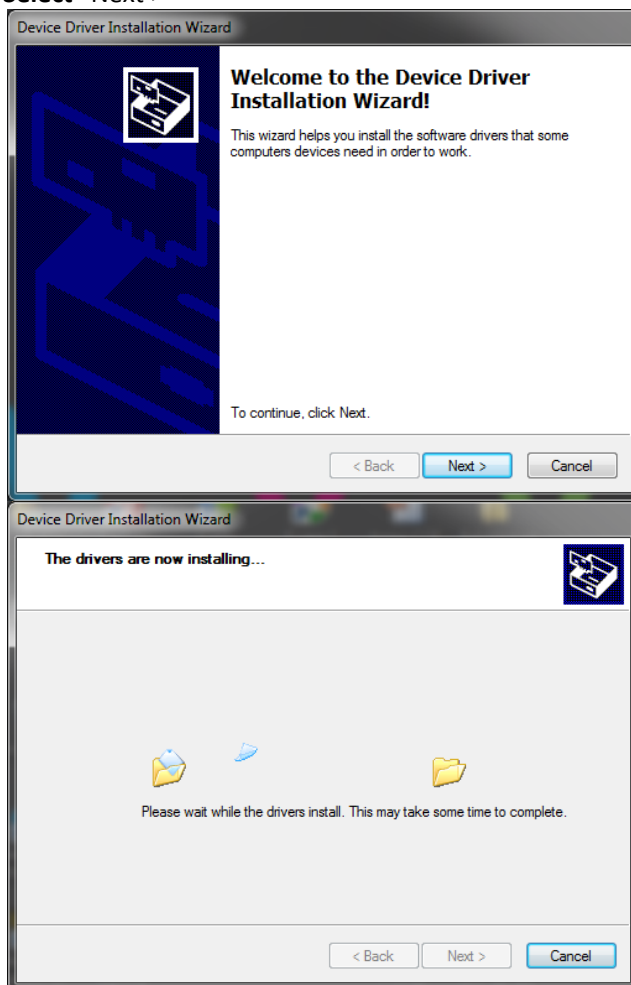
- The installation will continue with ModelSim Starter Edition:



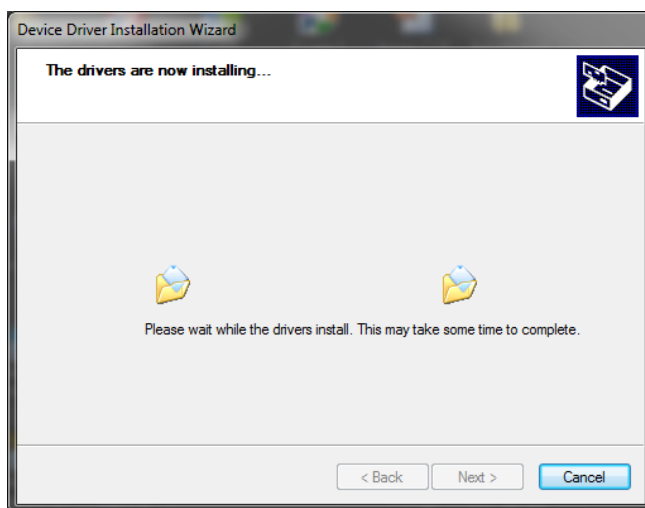
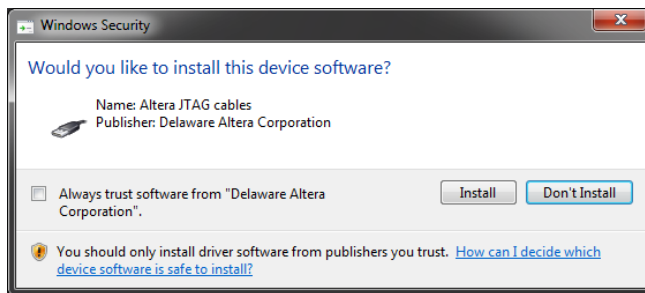
- Install the USB Blaster II Driver, select “Finish”



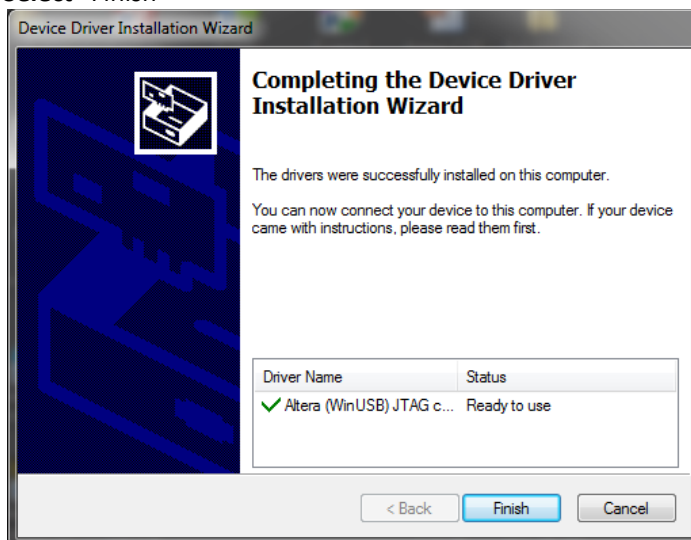
- Select “Next >”




- Select “Install”

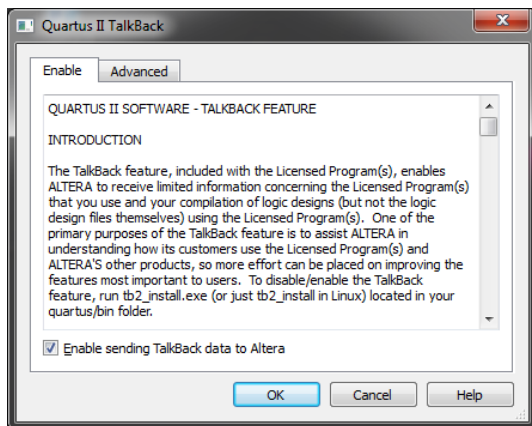


- Select “Finish”



- Select  -> All Programs -> Altera 14.1.0.181 Web Edition -> Quartus II Web Edition 14.1.0.181 -> Quartus II 14.1 (64 bit)

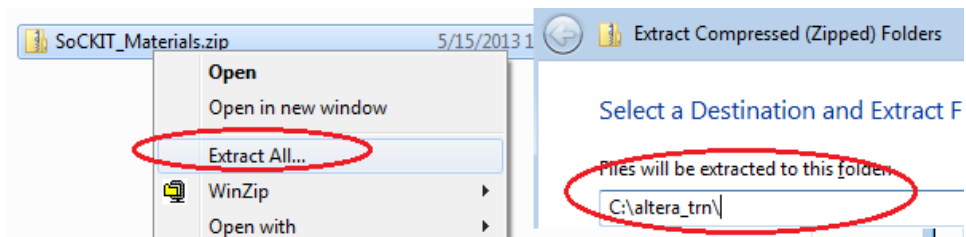
- **Enable TalkBack**



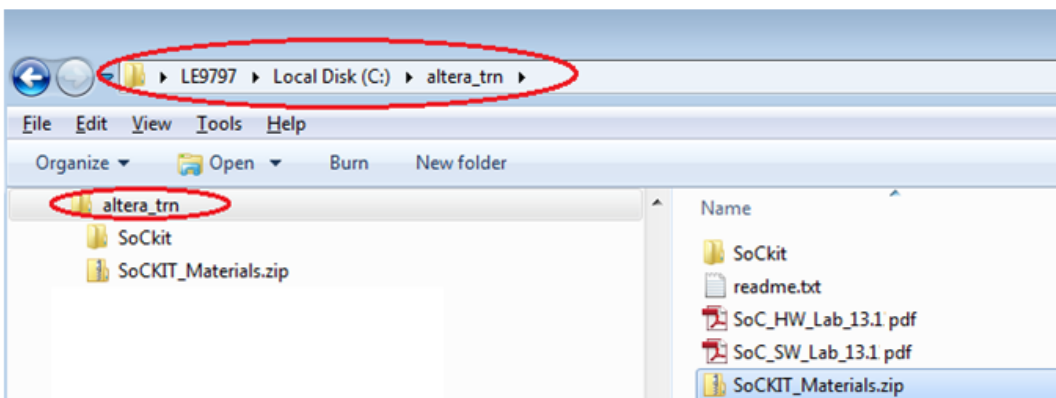
If splash screen above **doesn't** appear, then select: Tools -> Options. Under Category select: "TalkBack Options" Then you can Enable TalkBack

1.3 Extract the SoCKit Lab Files.

- Create a folder **c:\altera_trn** on your PC.
- Click on the following link to download **SoCKit Materials 14.1.zip**
- Save it to **c:\altera_trn** on your PC
- Extract the **SoCKit_Materials_14.1.zip** file to this folder



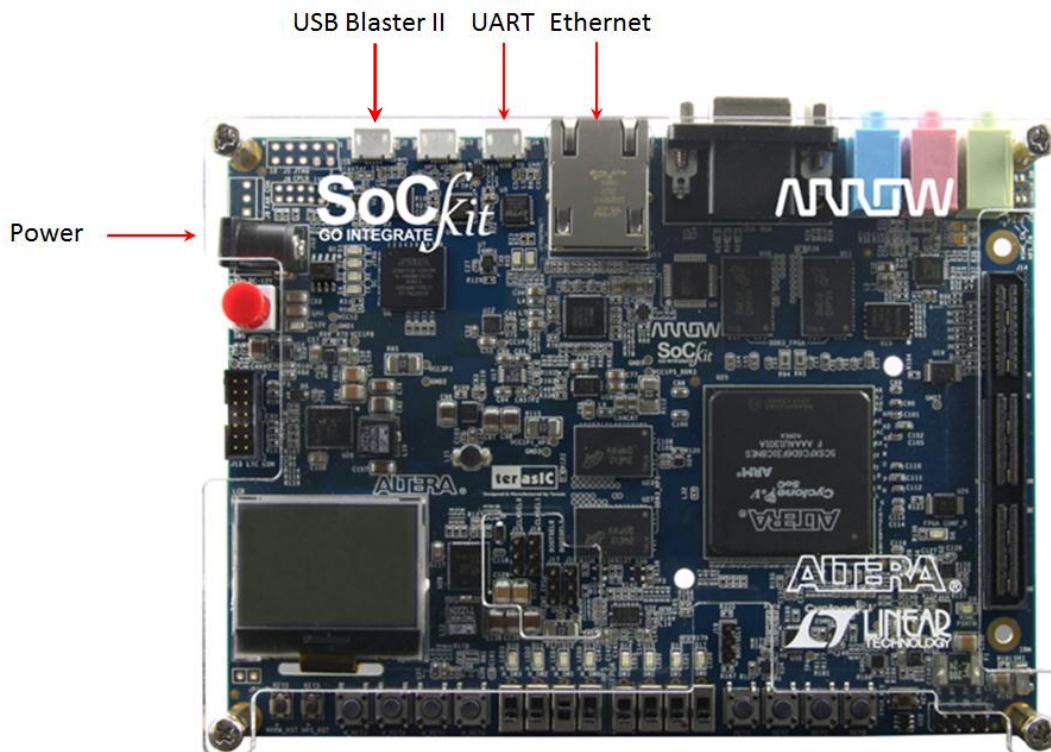
- The **c:\altera_trn** directory should look like this



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.

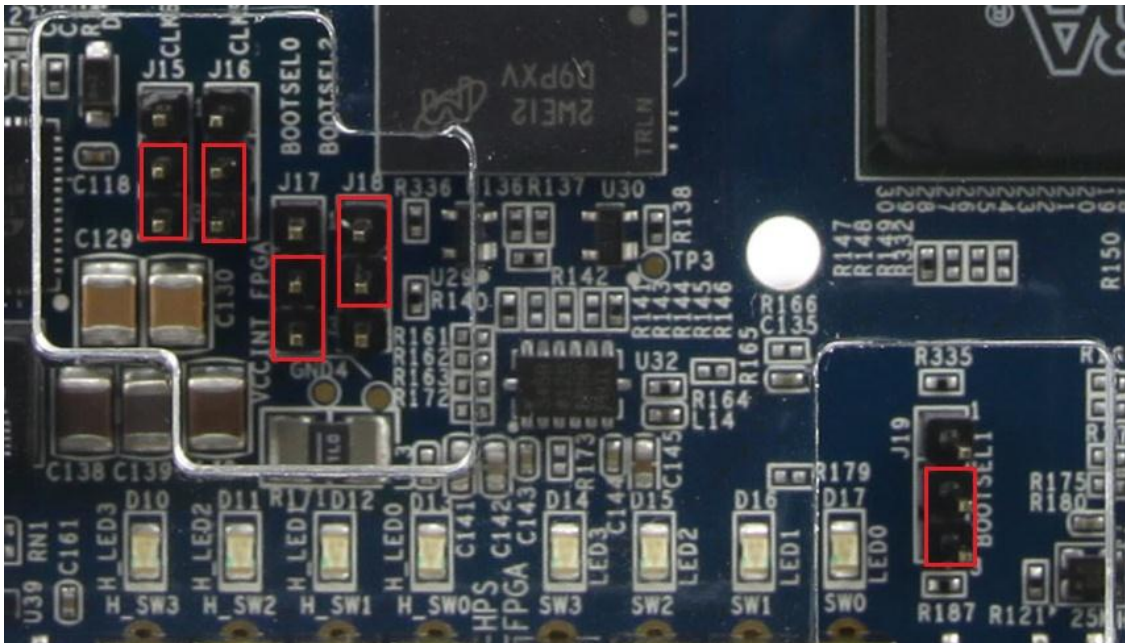
- Connect the micro or mini (Rev E) USB cable to the USB host connector on your laptop and to the USB Blaster II connector on the SoCKit.
- Connect the Power Supply to the Power connector on the SoCKit.



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

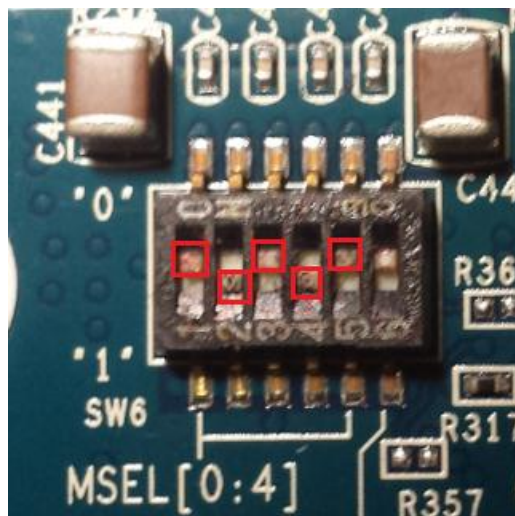
- **BOOTSEL[2..0]** jumpers. These should be configured as "100" to select boot from SD card 3.3V
- **CLKSEL[1..0]** jumpers. These should be configured as "00" for the slowest HPS peripheral clock speed option.

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



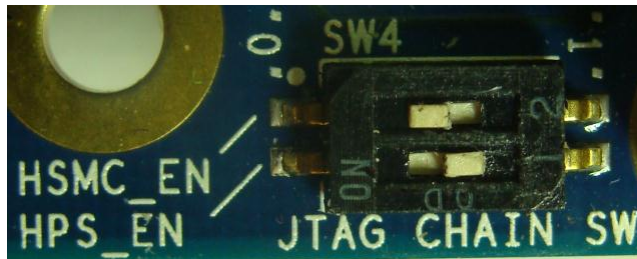
Modify the default **MSEL** bit settings. The board needs to be set to configure in the FPPx32, fast, compressed mode. This will allow u-boot to configure the FPGA.

- **SW6** is located on the **bottom** side of the SoCKit.
- Please change MSEL[0:4] to 01010.




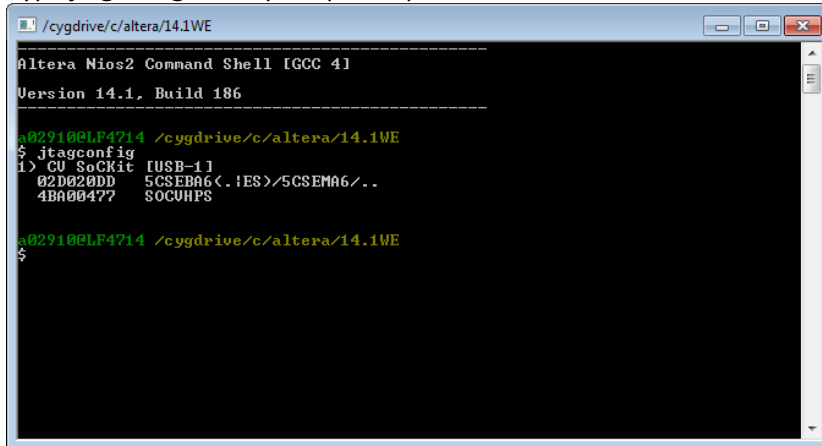
Verify that the **JTAG chain** is **correctly configured**. The **JTAG chain switch** is located in to the right of the **green audio** connector.

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

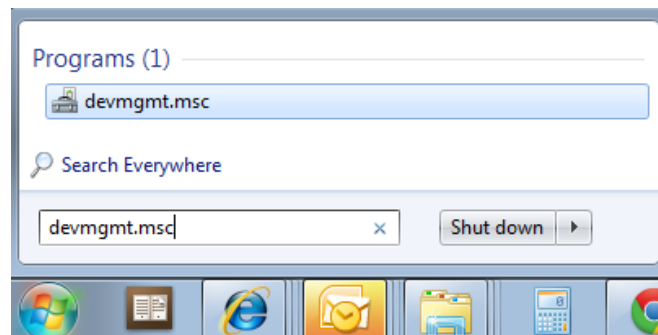


1.5 Install the USB Blaster II Device Driver (Complete if you didn't install with Quartus II in section 1.2)

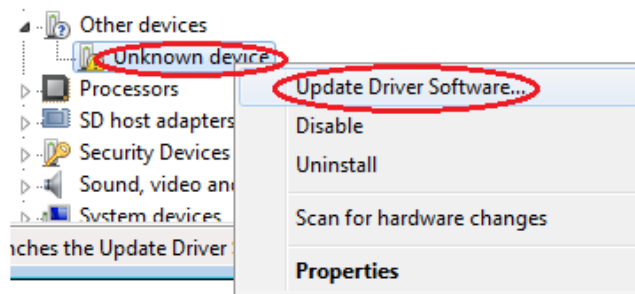
- **Turn your SoCKit on.**
- Open a NIOS II 14.0 Command Shell, select  -> All Programs -> Altera 14.1.0.186 Web Edition -> NIOS II EDS 14.1.0.186 -> NIOS II 14.1 Command Shell
- Type **jtagconfig** at the prompt and press enter.



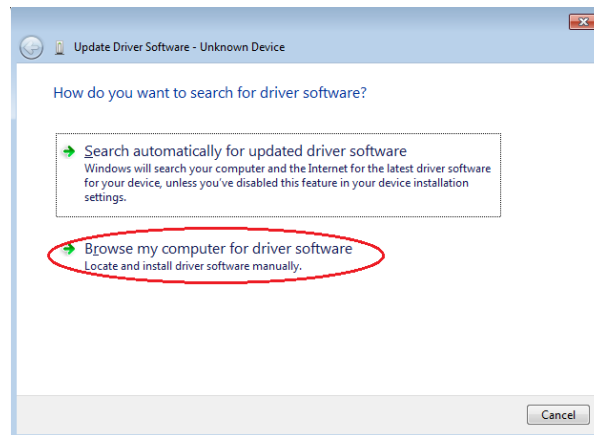
- If the jtagconfig command fails, then complete the following steps
- Press the Windows **Start** button. Enter **devmgmt.msc**. Press enter to open the Device Manager.



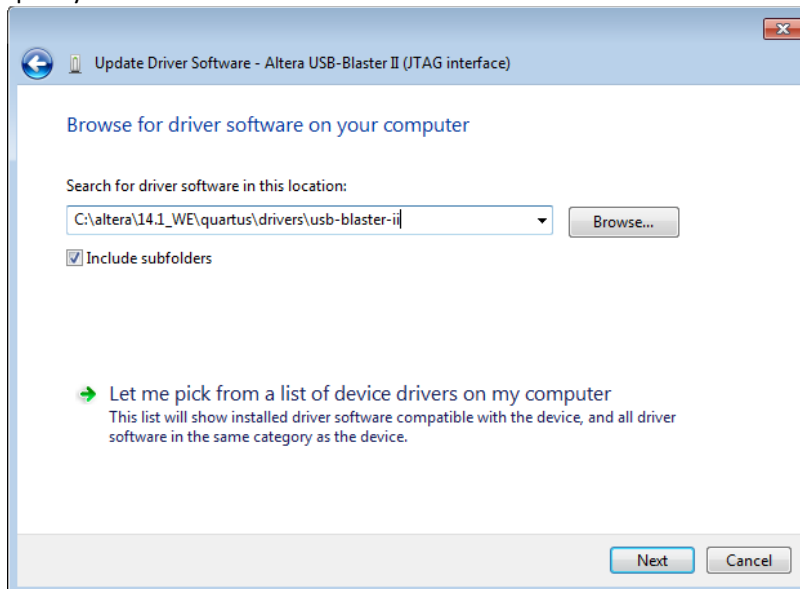
- Navigate to **Other Devices** in the Device Manager. Expand it to see **Unknown Device**
- Right click on **Unknown Device**. Select **Update Driver Software**.



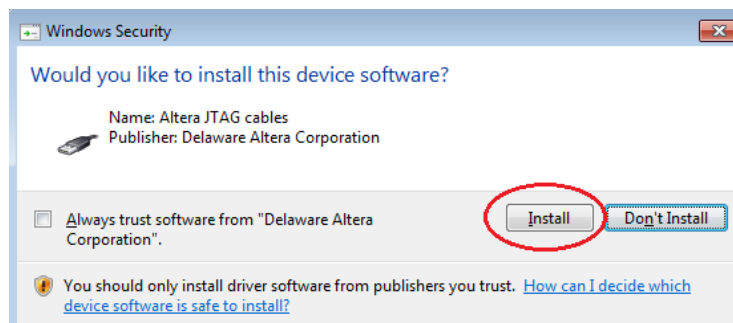
- Select **Browse my computer for driver software**.



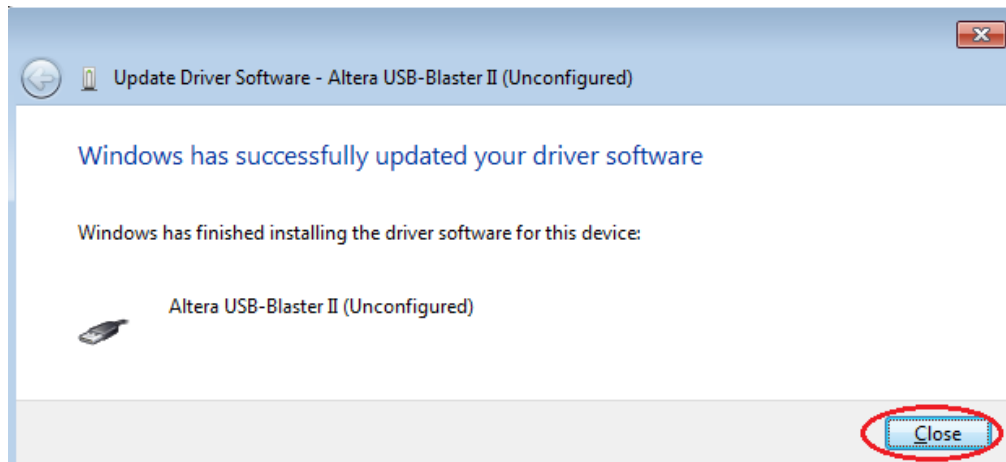
- Specify the driver software **location**. Press **Next**.




- 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 14.1 Command Shell, select  -> All Programs -> Altera 14.1.0.186 Web Edition -> NIOS II EDS 14.1.0.186 -> NIOS II 14.1 Command Shell
- Type **jtagconfig** at the prompt and press enter.

```

/cygdrive/c/altera/14.1WE
-----
Altera Nios2 Command Shell [GCC 4]
Version 14.1, Build 186
-----
a02910ELF4714 /cygdrive/c/altera/14.1WE
$ jtagconfig
1) CU SoCKit [USB-1]
   02D020DD 5CSEBA6<.IES>/5CSEMA6/..
   4BA00477 SOCUPPS
a02910ELF4714 /cygdrive/c/altera/14.1WE
$
    
```

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

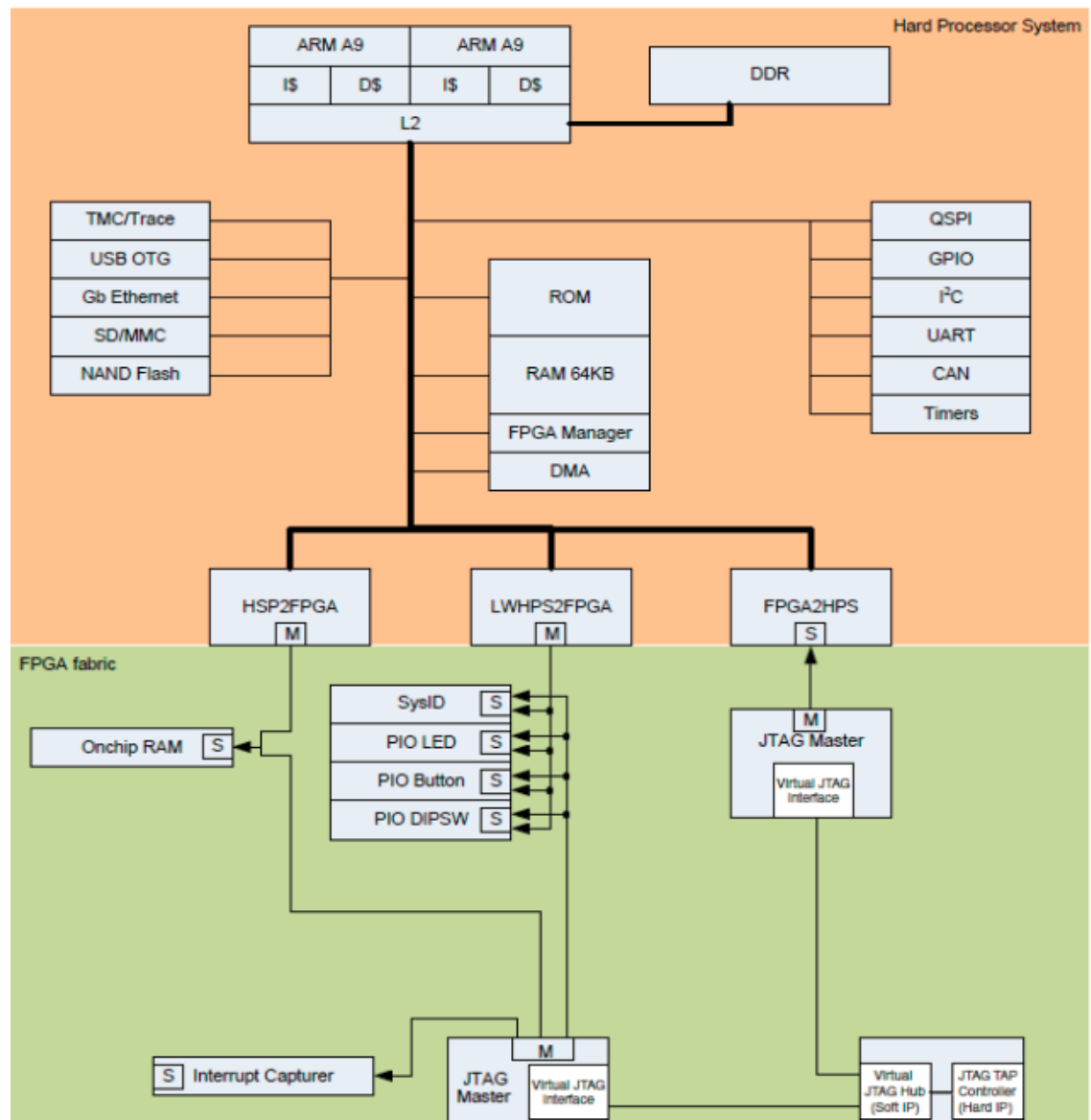
Module Objective

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.

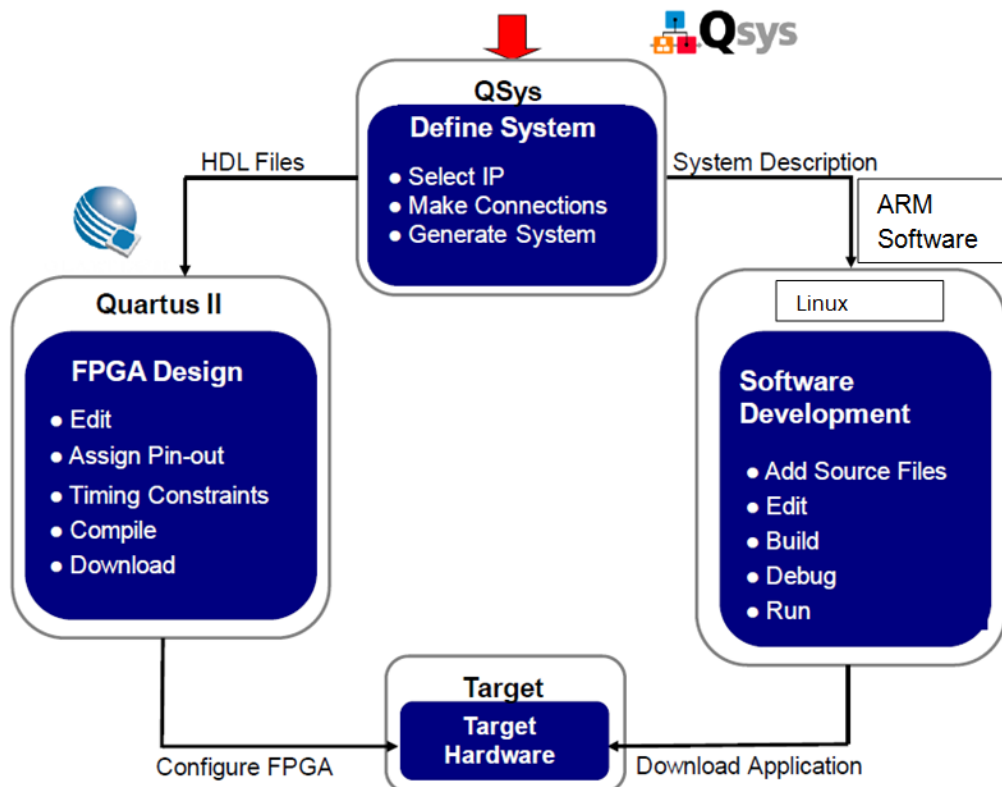
2.1 System Architecture

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 LWHP2FPGA bridge connected peripherals for this lab, specifically the LED PIO. The LED PIO is mapped through the bridge into the HPS addressable map.

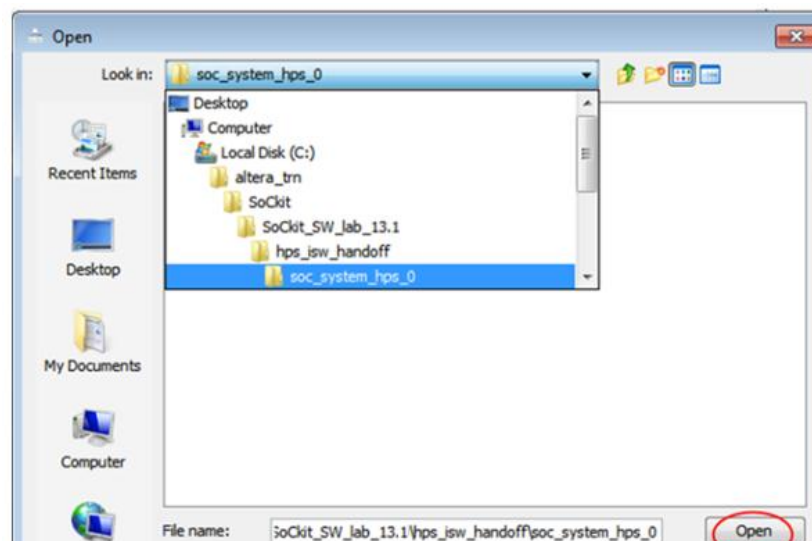


2.2 Examine the System Tool Flow



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

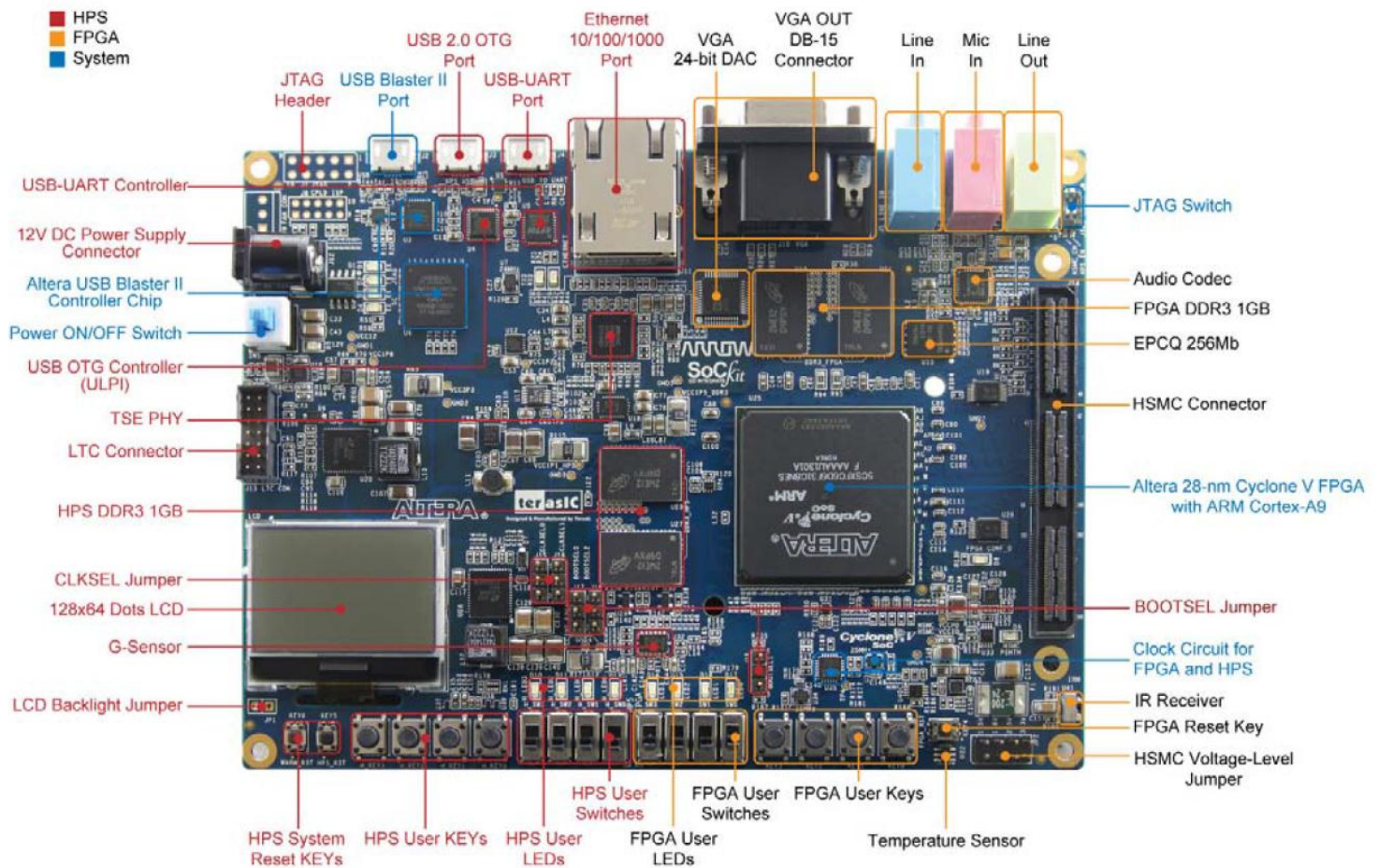
- A set of **XML files** are created that define the system description. These XML files are **utilized by the ARM DS-5** software tool to create a project for the **software application**. You can find the files here:



- 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 (Synopsys Design Constraint files) utilized by [TimeQuest](#) to constrain the complete system design.
- Quartus II will then generate a simple SOF (SRAM Object File) image that is used to configure the FPGA.

2.3 Examine Arrow's Cyclone V SoCKit

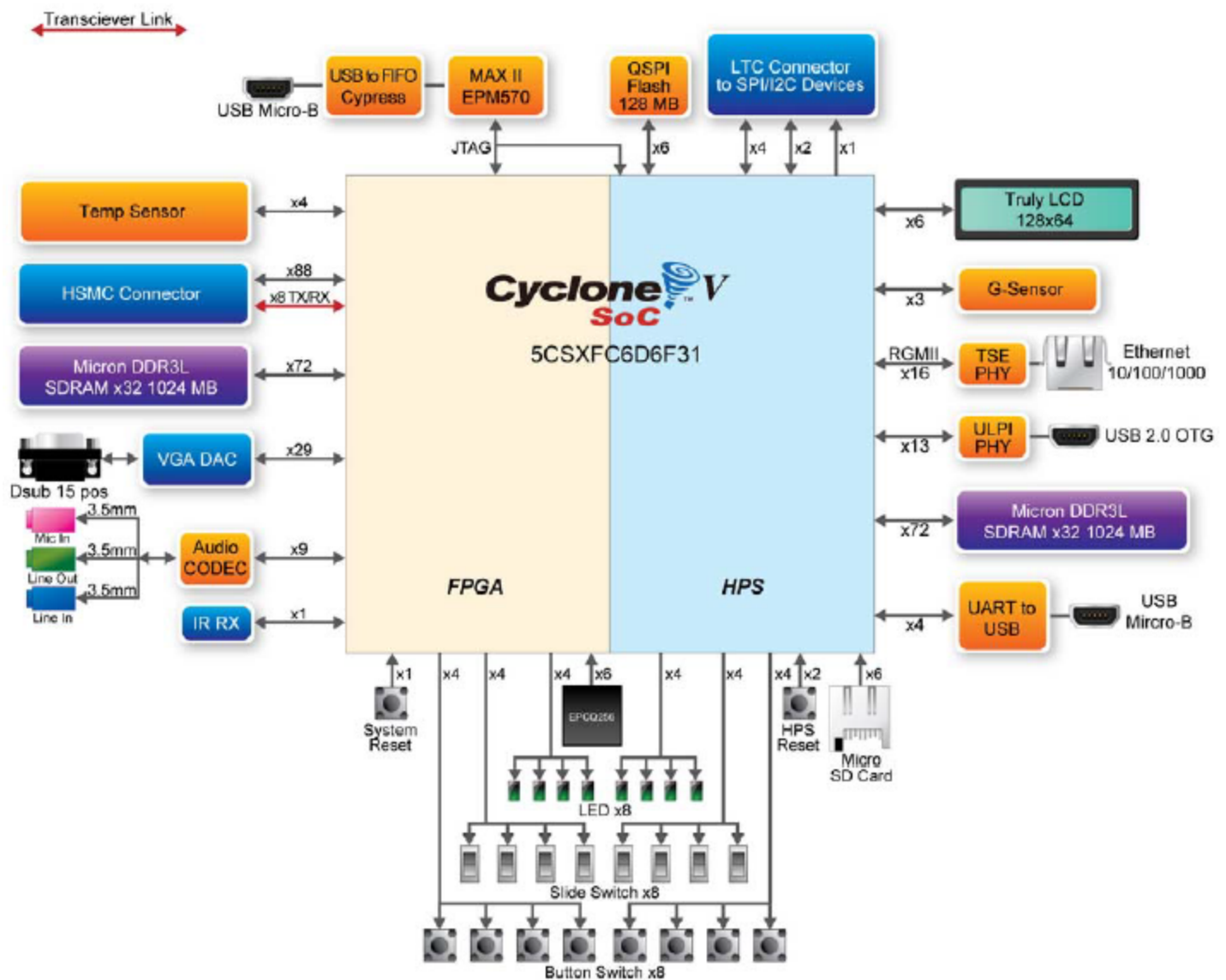
Examine the components on the Cyclone V SoC board hardware:



2.4 Block Diagram of the SoCKit

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

A 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

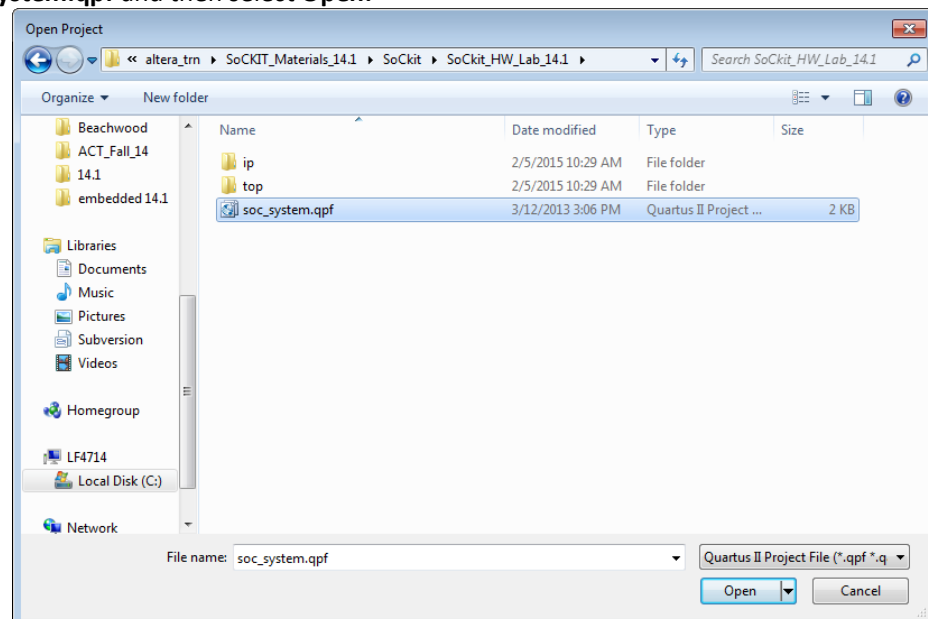
In this section, you will open a Quartus II project that contains the Qsys system. In addition, you will specify I/O constraints and settings for this design by executing a Tcl script.

3.1 Launch Quartus II Project

- Launch the **Quartus II v14.1** software: Select  -> **All Programs -> Altera 14.1.0.186 Web Edition -> Quartus II Web Edition 14.1.0.186 -> Quartus II 14.1 (64 bit)**
- A splash screen will appear, select **Open Existing Project**:

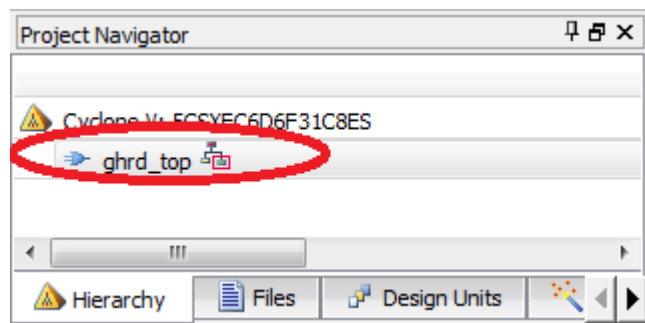


- Now browse to the directory:
C:\altera_trn\SoCKIT_Materials_14.1\SoCKit\SoCKit_HW_lab_14.1 and select **soc_system.qpf** and then select **Open**.



If you close the splash screen without opening the project:

- Select **File -> Open Project** and browse to the directory:
C:\altera_trn\SoCKIT_Materials_14.1\SoCKit\SoCKit_HW_lab_14.1.
- Select **soc_system.qpf.**
- The Quartus II project will open. The project already contains a top level Verilog file (**..\top\ghrd_top.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 **ghrd_top** icon in the Project navigator Window **or** (select: **File -> Open** and browse to the **..\top** directory and open **ghrd_top.v**)



- The **ghrd_top.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.

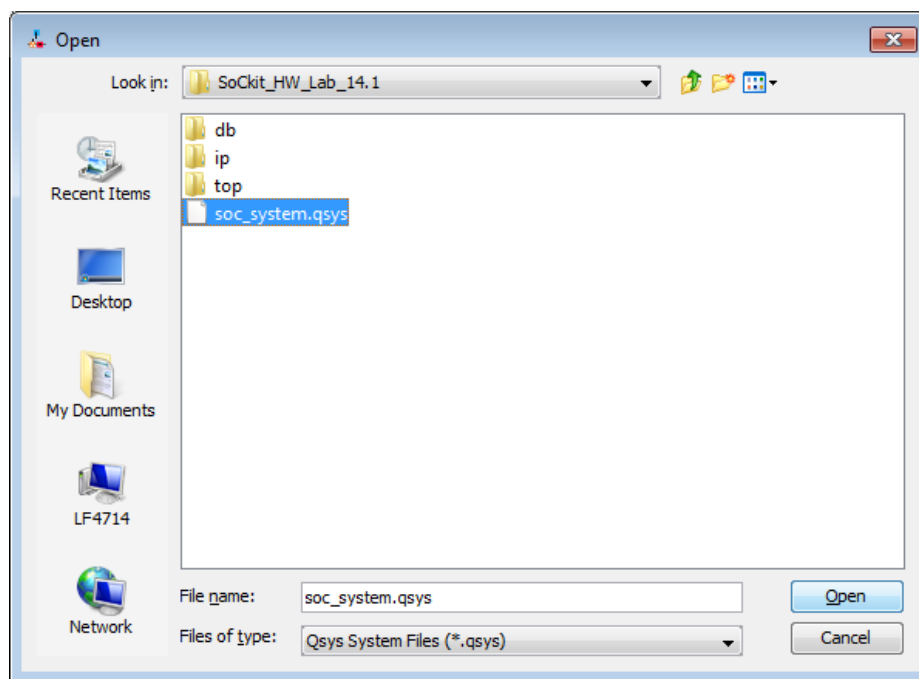
MODULE 4. Build the Qsys System

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**



Configure the HPS

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

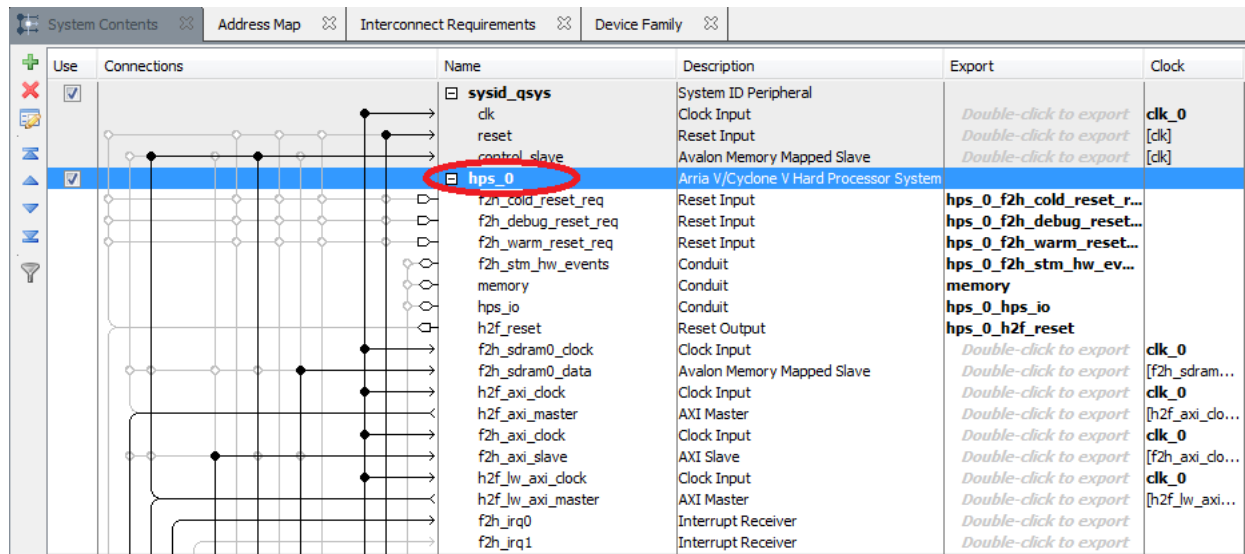


Figure 4.2.1

- 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.

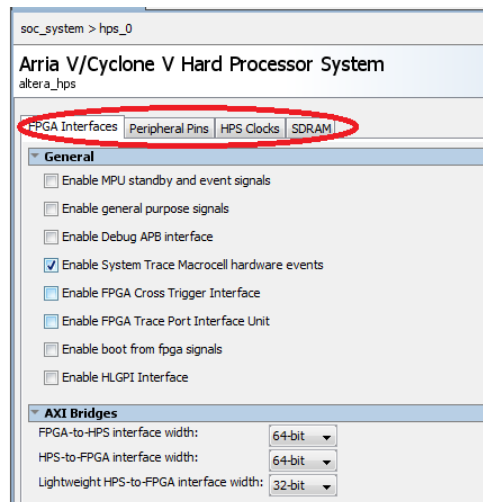


Figure 4.2.2

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** except for the System Trace Macrocell (as shown in Fig. 4.2.2). Refer to the Explanation of options below. 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** (as shown in Fig. 4.2.2). 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 FPGA Cross Trigger Interface"**, and **"Enable FPGA Trace Port Interface Unit"** are **all disabled** (as shown in Fig. 4.2.1.1).

Ensure that the **"Enable System Trace Macrocell Hardware events"** is **enabled** (as shown in Fig. 4.2.1.1).

Explanation of options:

- **Debug APB interface** - Enables debug interface to the FPGA, allowing access to debug components in the HPS.
- **Enable 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** (as shown in Fig. 4.2.1.1).

Explanation of options:

- **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.
- **Enable HGPI Interface** - This will instantiate 14 general purpose DDR inputs. If you check these signals will be available in the HPS IO conduit in Qsys and are at same IO voltage as the DDR interface. Please refer to the [Pin Connection Guidelines](#).

Ensure that the "Enable HGPL Interface" is **disabled** (as shown in Fig. 4.2.1.1).

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** (as shown in Fig. 4.2.1.1). 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 (as shown in Fig. 4.2.1.1).

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

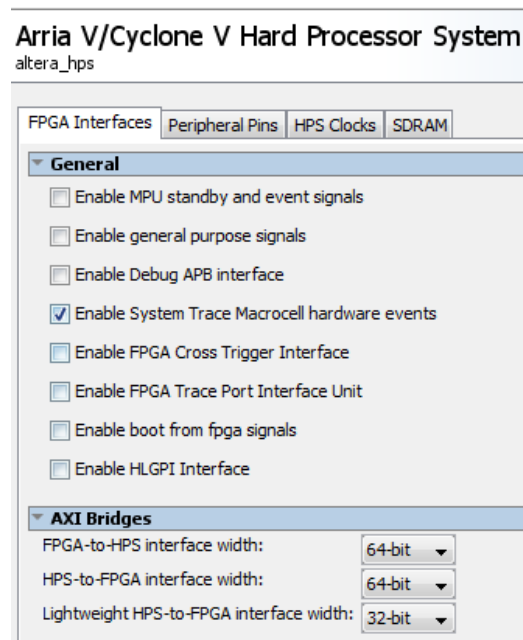


Figure 4.2.1.1

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 has one entry: **f2h_sdram0**.

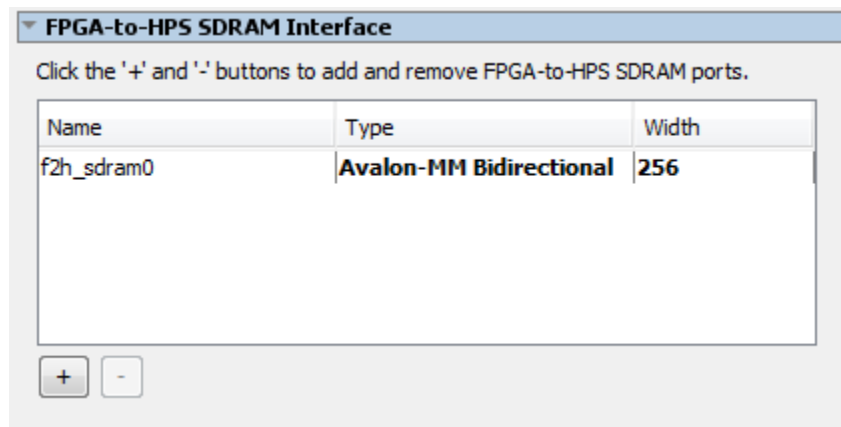


Figure 4.2.1.2

- This enables the FPGA to gain access to the HPS SDRAM subsystem from the FPGA fabric. This particular FPGA to HPS interface provides up to six ports to the HPS's multiport SDRAM controller. The bus interface provided to the FPGA can be AXI-3 or Avalon Memory Mapped.

Ensure that the **"Enable FPGA-to-HPS debug reset request"** and **"Enable FPGA-to-HPS warm reset request"** and **"Enable FPGA-to-HPS cold reset request"** are set to **enabled** (as shown in Fig. 4.2.1.3). 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](#).

Explanation of reset options; **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 (as shown in Fig. 4.2.1.3). 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 (as shown in Fig. 4.2.1.3). This enables interrupt signals from the FPGA to the MPU in the HPS.

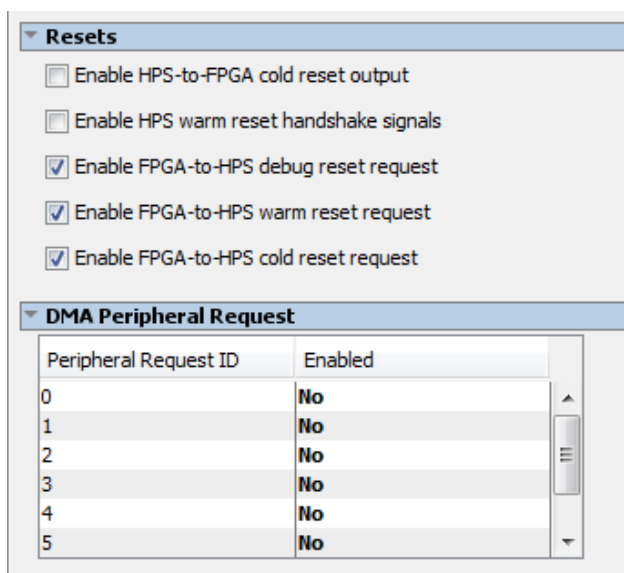


Figure 4.2.1.3

The "HPS to FPGA" interrupts should all be **disabled** (as shown in Figure 4.2.14). 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:

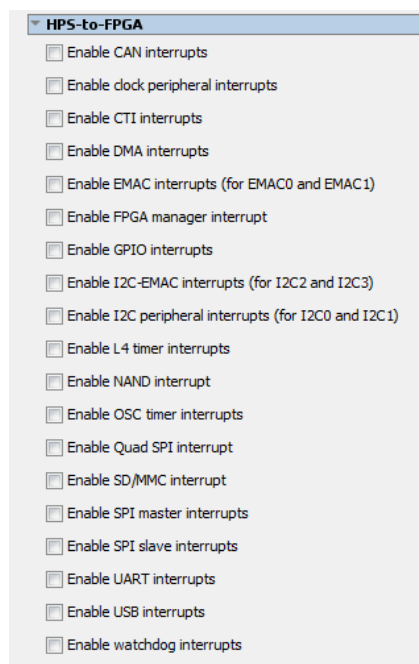


Figure 4.2.1.4

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 table** is available at the bottom of this tab to make it a simpler and more intuitive task for the designer. Also, *please note that all HPS peripherals are available for pin out via the FPGAs pins.*

If you put your cursor over a particular peripherals **mode** icon: a list of the signal to pin for a particular I/O set per pin is displayed in a pop-up box.

Peripheral Pins Configuration:

- Ethernet Media Access Controller**
 - EMAC0 pin: Unused
 - EMAC0 mode: N/A
 - EMAC1 pin: HPS I/O Set 0
 - EMAC1 mode: RGMII
- NAND Flash Controller**
 - NAND pin: Unused
 - NAND mode: N/A
- Quad SPI Flash Controller**
 - QSPI pin: HPS I/O
 - QSPI mode: 1 SS
- SD/MMC Controller**
 - SDIO pin: HPS I/O
 - SDIO mode: 4-bit Data
- USB Controllers**
 - USB0 pin: Unused
 - USB0 PHY interface mode: N/A
 - USB1 pin: HPS I/O
 - USB1 PHY interface mode: SDR with
- SPI Controllers**
 - SPI0 pin: HPS I/O
 - SPI0 mode: Single Slave
 - SPI1 pin: HPS I/O
 - SPI1 mode: Single Slave

EMAC1 mode (EMAC1_Mode): Signal Membership Per Mode Usage Option:

	RGMII	RGMII with I2C3
MDC	X	
MDIO	X	
RXD0	X	X
RXD1	X	X
RXD2	X	X
RXD3	X	X
RX_CLK	X	X
RX_CTL	X	X
TXD0	X	X
TXD1	X	X
TXD2	X	X
TXD3	X	X
TX_CLK	X	X
TX_CTL	X	X

Figure 4.2.2.1

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

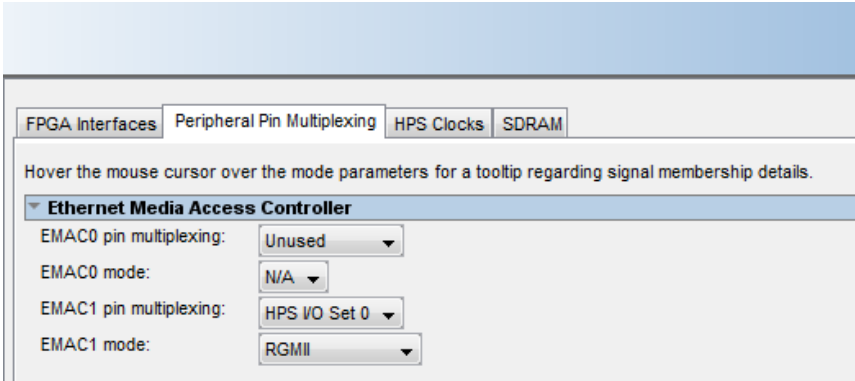


Figure 4.2.2.2

to be

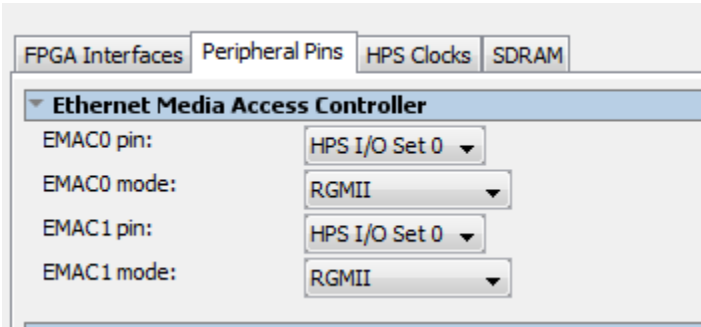


Figure 4.2.2.3

There will be errors in the Messages Window:

Type	Path	Message
3 Errors		
	soc_system.hps_0	Refer to the Peripherals Mux Table for more details. The selected peripherals 'EMAC0' and 'USB1' are conflicting.
	soc_system.hps_0	Refer to the Peripherals Mux Table for more details. The selected peripheral 'EMAC0' and 'GPIO00' are conflicting.
	soc_system.hps_0	Refer to the Peripherals Mux Table for more details. The selected peripheral 'EMAC0' and 'GPIO09' are conflicting.

Figure 4.2.2.4

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. (Starting in version 13.1, an **HPS pin multiplexing table** is available at the bottom of the “Peripheral Pin Multiplexing” tab 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 be **Enabled as a GPIO pin for use as a General Purpose I/O Pin**.

Conflicts (column and row are in Red/Burgundy) :

Peripherals Mux Table			
RGMII0_TX_CLK			EMAC0.TX_CLK (Set0)
RGMII0_TXD0		USB1.D0 (Set0)	EMAC0.TXD0 (Set0)
RGMII0_TXD1		USB1.D1 (Set0)	EMAC0.TXD1 (Set0)
RGMII0_TXD2		USB1.D2 (Set0)	EMAC0.TXD2 (Set0)
RGMII0_TXD3		USB1.D3 (Set0)	EMAC0.TXD3 (Set0)
RGMII0_RXD0		USB1.D4 (Set0)	EMAC0.RXD0 (Set0)
RGMII0_MDIO	I2C2.SDA (Set0)	USB1.D5 (Set0)	EMAC0.MDIO (Set0)
RGMII0_MDC	I2C2.SCL (Set0)	USB1.D6 (Set0)	EMAC0.MDC (Set0)
RGMII0_RX_CTL		USB1.D7 (Set0)	EMAC0.RX_CTL (Set0)
RGMII0_TX_CTL			EMAC0.TX_CTL (Set0)
RGMII0_RX_CLK		USB1.CLK (Set0)	EMAC0.RX_CLK (Set0)
RGMII0_RXD1		USB1.STP (Set0)	EMAC0.RXD1 (Set0)
RGMII0_RXD2		USB1.DIR (Set0)	EMAC0.RXD2 (Set0)
RGMII0_RXD3		USB1.NXT (Set0)	EMAC0.RXD3 (Set0)

Figure 4.2.2.5

No Conflicts: (For the peripheral selected mux_select column is green):

Peripherals Mux Table			
RGMII0_TX_CLK			EMAC0.TX_CLK (Set0)
RGMII0_TXD0		USB1.D0 (Set0)	EMAC0.TXD0 (Set0)
RGMII0_TXD1		USB1.D1 (Set0)	EMAC0.TXD1 (Set0)
RGMII0_TXD2		USB1.D2 (Set0)	EMAC0.TXD2 (Set0)
RGMII0_TXD3		USB1.D3 (Set0)	EMAC0.TXD3 (Set0)
RGMII0_RXD0		USB1.D4 (Set0)	EMAC0.RXD0 (Set0)
RGMII0_MDIO	I2C2.SDA (Set0)	USB1.D5 (Set0)	EMAC0.MDIO (Set0)
RGMII0_MDC	I2C2.SCL (Set0)	USB1.D6 (Set0)	EMAC0.MDC (Set0)
RGMII0_RX_CTL		USB1.D7 (Set0)	EMAC0.RX_CTL (Set0)
RGMII0_TX_CTL			EMAC0.TX_CTL (Set0)
RGMII0_RX_CLK		USB1.CLK (Set0)	EMAC0.RX_CLK (Set0)
RGMII0_RXD1		USB1.STP (Set0)	EMAC0.RXD1 (Set0)
RGMII0_RXD2		USB1.DIR (Set0)	EMAC0.RXD2 (Set0)
RGMII0_RXD3		USB1.NXT (Set0)	EMAC0.RXD3 (Set0)

Figure 4.2.2.5

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.

The screenshot shows the 'Peripheral Pins' tab in the Qsys configuration tool. It displays settings for four controllers:

- Ethernet Media Access Controller:**
 - EMAC0 pin: Unused
 - EMAC0 mode: N/A
 - EMAC1 pin: HPS I/O Set 0
 - EMAC1 mode: RGMII
- NAND Flash Controller:**
 - NAND pin: Unused
 - NAND mode: N/A
- Quad SPI Flash Controller:**
 - QSPI pin: HPS I/O Set 0
 - QSPI mode: 1 SS
- SD/MMC Controller:**
 - SDIO pin: HPS I/O Set 0
 - SDIO mode: 4-bit Data

Figure 4.2.2.6

The screenshot shows the 'Peripheral Pins' tab in the Qsys configuration tool, continuing from the previous one. It displays settings for three more controller groups:

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

Figure 4.2.2.7

I2C Controllers	
I2C0 pin:	Unused
I2C0 mode:	N/A
I2C1 pin:	HPS I/O Set 0
I2C1 mode:	I2C
I2C2 pin:	Unused
I2C2 mode:	N/A
I2C3 pin:	Unused
I2C3 mode:	N/A

CAN Controllers	
CAN0 pin:	Unused
CAN0 mode:	N/A
CAN1 pin:	Unused
CAN1 mode:	N/A

Trace Port Interface Unit	
TRACE pin:	Unused
TRACE mode:	N/A

Figure 4.2.2.8

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

Peripherals Mux Table				
RGMII0_TX_CLK			EMAC0_TX_CLK (Set0)	GPIO00
RGMII0_TXD0		USB1.D0 (Set0)	EMAC0_TXD0 (Set0)	GPIO01
RGMII0_TXD1		USB1.D1 (Set0)	EMAC0_TXD1 (Set0)	GPIO02
RGMII0_TXD2		USB1.D2 (Set0)	EMAC0_TXD2 (Set0)	GPIO03
RGMII0_TXD3		USB1.D3 (Set0)	EMAC0_TXD3 (Set0)	GPIO04
RGMII0_RXD0		USB1.D4 (Set0)	EMAC0_RXD0 (Set0)	GPIO05
RGMII0_MDIO	I2C2.SDA (Set0)	USB1.D5 (Set0)	EMAC0_MDIO (Set0)	GPIO08
RGMII0_MDC	I2C2.SCL (Set0)	USB1.D6 (Set0)	EMAC0_MDC (Set0)	GPIO07
RGMII0_RX_CTL		USB1.D7 (Set0)	EMAC0_RX_CTL (Set0)	GPIO08
RGMII0_TX_CTL			EMAC0_TX_CTL (Set0)	GPIO09
RGMII0_RX_CLK		USB1.CLK (Set0)	EMAC0_RX_CLK (Set0)	GPIO10
RGMII0_RXD1		USB1.STP (Set0)	EMAC0_RXD1 (Set0)	GPIO11
RGMII0_RXD2		USB1.DIR (Set0)	EMAC0_RXD2 (Set0)	GPIO12
RGMII0_RXD3		USB1.NXT (Set0)	EMAC0_RXD3 (Set0)	GPIO13
NAND_ALE	QSPI_SS3 (Set1) (Set0)	EMAC1.TX_CLK (Set0)	NAND.ALE (Set0)	GPIO14
NAND_CE	USB1.D0 (Set1)	EMAC1.TXD0 (Set0)	NAND.CE (Set0)	GPIO15
NAND_CLE	USB1.D1 (Set1)	EMAC1.TXD1 (Set0)	NAND.CLE (Set0)	GPIO16
NAND_RE	USB1.D2 (Set1)	EMAC1.TXD2 (Set0)	NAND.RE (Set0)	GPIO17
NAND_RB	USB1.D3 (Set1)	EMAC1.TXD3 (Set0)	NAND.RB (Set0)	GPIO18
NAND_DQ0		EMAC1.RXD0 (Set0)	NAND.DQ0 (Set0)	GPIO19
NAND_DQ1	I2C3.SDA (Set0)	EMAC1.MDIO (Set0)	NAND.DQ1 (Set0)	GPIO20
NAND_DQ2	I2C3.SCL (Set0)	EMAC1.MDC (Set0)	NAND.DQ2 (Set0)	GPIO21
NAND_DQ3	USB1.D4 (Set1)	EMAC1.RX_CTL (Set0)	NAND.DQ3 (Set0)	GPIO22
NAND_DQ4	USB1.D5 (Set1)	EMAC1.TX_CTL (Set0)	NAND.DQ4 (Set0)	GPIO23
NAND_DQ5	USB1.D6 (Set1)	EMAC1.RX_CLK (Set0)	NAND.DQ5 (Set0)	GPIO24
NAND_DQ6	USB1.D7 (Set1)	EMAC1.RXD1 (Set0)	NAND.DQ6 (Set0)	GPIO25
NAND_DQ7		EMAC1.RXD2 (Set0)	NAND.DQ7 (Set0)	GPIO26
NAND_WP	QSPI_SS2 (Set1) (Set0)	EMAC1.RXD3 (Set0)	NAND.WP (Set0)	GPIO27

NAND_WE		QSPI_SS1 (Set0)	NAND_WE (Set0)	GPIO28
QSPI_IO0	USB1.CLK (Set1)		QSPI_IO0 (Set1) (Set0)	GPIO29
QSPI_IO1	USB1.STP (Set1)		QSPI_IO1 (Set1) (Set0)	GPIO30
QSPI_IO2	USB1.DIR (Set1)		QSPI_IO2 (Set1) (Set0)	GPIO31
QSPI_IO3	USB1.NXT (Set1)		QSPI_IO3 (Set1) (Set0)	GPIO32
QSPI_SS0			QSPI_SS0 (Set1) (Set0)	GPIO33
QSPI_CLK			QSPI.CLK (Set1) (Set0)	GPIO34
QSPI_SS1			QSPI_SS1 (Set1)	GPIO35
SDMMC_CMD		USB0_D0 (Set0)	SDIO_CMD (Set0)	GPIO36
SDMMC_PWREN		USB0_D1 (Set0)	SDIO_PWREN (Set0)	GPIO37
SDMMC_D0		USB0_D2 (Set0)	SDIO_D0 (Set0)	GPIO38
SDMMC_D1		USB0_D3 (Set0)	SDIO_D1 (Set0)	GPIO39
SDMMC_D4		USB0_D4 (Set0)	SDIO_D4 (Set0)	GPIO40
SDMMC_D5		USB0_D5 (Set0)	SDIO_D5 (Set0)	GPIO41
SDMMC_D6		USB0_D6 (Set0)	SDIO_D6 (Set0)	GPIO42
SDMMC_D7		USB0_D7 (Set0)	SDIO_D7 (Set0)	GPIO43
SDMMC_FB_CLK_IN		USB0_CLK (Set0)	SDIO_CLK_IN (Set0)	GPIO44
SDMMC_CCLK_OUT		USB0.STP (Set0)	SDIO.CLK (Set0)	GPIO45
SDMMC_D2		USB0.DIR (Set0)	SDIO.D2 (Set0)	GPIO46
SDMMC_D3		USB0.NXT (Set0)	SDIO.D3 (Set0)	GPIO47
TRACE_CLK			TRACE.CLK (Set0)	GPIO48
TRACE_D0	UART0.RX (Set0)	SPI0.CLK (Set0)	TRACE.D0 (Set0)	GPIO49
TRACE_D1	UART0.TX (Set0)	SPI0.MOSI (Set0)	TRACE.D1 (Set0)	GPIO50
TRACE_D2	I2C1.SDA (Set0)	SPI0.MISO (Set0)	TRACE.D2 (Set0)	GPIO51
TRACE_D3	I2C1.SCL (Set0)	SPI0.SS0 (Set0)	TRACE.D3 (Set0)	GPIO52
TRACE_D4	CAN1.RX (Set0)	SPI1.CLK (Set0)	TRACE.D4 (Set0)	GPIO53
TRACE_D5	CAN1.TX (Set0)	SPI1.MOSI (Set0)	TRACE.D5 (Set0)	GPIO54
TRACE_D6	I2C0.SDA (Set0)	SPI1.SS0 (Set0)	TRACE.D6 (Set0)	GPIO55
TRACE_D7	I2C0.SCL (Set0)	SPI1.MISO (Set0)	TRACE.D7 (Set0)	GPIO56
SPIM0_CLK	UART0.CTS (Set2) (Set1) (Set0)	I2C1.SDA (Set1)	SPIM0.CLK (Set0)	GPIO57
SPIM0_MOSI	UART0.RTS (Set2) (Set1) (Set0)	I2C1.SCL (Set1)	SPIM0.MOSI (Set0)	GPIO58
SPIM0_MISO	UART1.CTS (Set0)	CAN1.RX (Set1)	SPIM0.MISO (Set0)	GPIO59
SPIM0_SS0	UART1.RTS (Set0)	CAN1.TX (Set1)	SPIM0.SS0 (Set0)	GPIO60
UART0_RX	SPIM0.SS1 (Set0)	CAN0.RX (Set0)	UART0.RX (Set1)	GPIO61
UART0_TX	SPIM1.SS1 (Set0)	CAN0.TX (Set0)	UART0.TX (Set1)	GPIO62
I2C0_SDA	SPIM1.CLK (Set0)	UART1.RX (Set0)	I2C0.SDA (Set1)	GPIO63
I2C0_SCL	SPIM1.MOSI (Set0)	UART1.TX (Set0)	I2C0.SCL (Set1)	GPIO64
CAN0_RX	SPIM1.MISO (Set0)	UART0.RX (Set2)	CAN0.RX (Set1)	GPIO65
CAN0_TX	SPIM1.SS0 (Set0)	UART0.TX (Set2)	CAN0.TX (Set1)	GPIO66

Figure 4.2.2.9

4.2.3 Configure HPS Clocks

Select the **HPS Clocks** tab. Under the Input Clocks sub tab; there are selections to set the External Clock pins provided to the HPS: EOSC1 & EOSC2, the ability to enable clocks between the HPS to FPGA and FPGA to HPS. New in version 14.0 is an Output Clock tab.

4.2.3.1 Configuring HPS Input Clocks

The "EOSC1 clock frequency and EOSC2 clock frequency" External Clock Sources should all be set to 25.0 Mhz and 25.0 Mhz (as shown in Figure 4.2.3.1). **Setting EOSC1 & EOSC2 is new in version 14.0.**

The bullets below are descriptive. **Do NOT implement these changes.**

Explanation of user clock options for the Output tab:

- External Clock Sources: EOSC1 and EOSC2 are based upon the frequency at the HPS I/O pins: HPS_CLK1 and HPS_CLK2, D25 and F25 for the 5CSXFC6D6F31C6
- 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
- **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

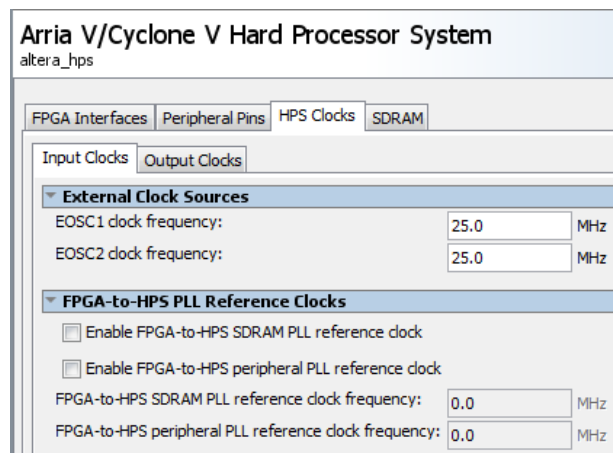


Figure 4.2.3.1

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

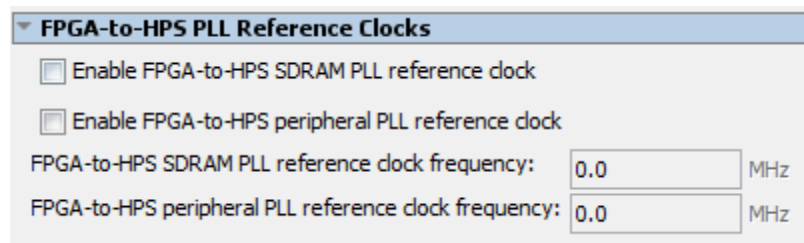


Figure 4.2.3.2

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

Peripheral FPGA Clocks		
EMAC0 emac0_md_clk clock frequency:	100	MHz
EMAC0 emac0_gtx_clk clock frequency:	100	MHz
EMAC1 emac1_md_clk clock frequency:	100	MHz
EMAC1 emac1_gtx_clk clock frequency:	100	MHz
QSPI qspi_sclk_out clock frequency:	100	MHz
SPIM0 spim0_sclk_out clock frequency:	100	MHz
SPIM1 spim1_sclk_out clock frequency:	100	MHz
I2C0 i2c0_clk clock frequency:	100	MHz
I2C1 i2c1_clk clock frequency:	100	MHz
I2C2 i2c2_clk clock frequency:	100	MHz
I2C3 i2c3_clk clock frequency:	100	MHz

Figure 4.2.3.3

For example, if SPM1 were selected to be available in the FPGA under the Peripheral Pins tab,

SPI Controllers	
SPIM0 pin multiplexing:	HPS I/O Set 0
SPIM0 mode:	Single Slave Select
SPIM1 pin multiplexing:	FPGA
SPIM1 mode:	Full
SPIS0 pin multiplexing:	FPGA
SPIS0 mode:	Full
SPIS1 pin multiplexing:	Unused
SPIS1 mode:	N/A

Figure 4.2.3.4

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

Peripheral FPGA Clocks		
EMAC0 emac0_md_clk clock frequency:	100	MHz
EMAC0 emac0_gtx_clk clock frequency:	100	MHz
EMAC1 emac1_md_clk clock frequency:	100	MHz
EMAC1 emac1_gtx_clk clock frequency:	100	MHz
QSPI qspi_sclk_out clock frequency:	100	MHz
SDIO sdio_clk clock frequency:	100	MHz
SPIM0 spim0_sclk_out clock frequency:	100	MHz
SPIM1 spim1_sclk_out clock frequency:	100	MHz
I2C0 i2c0_clk clock frequency:	100	MHz
I2C1 i2c1_clk clock frequency:	100	MHz
I2C2 i2c2_clk clock frequency:	100	MHz
I2C3 i2c3_clk clock frequency:	100	MHz

Figure 4.2.3.5

If you made the change to SPM1 pin multiplexing change it back to UNUSED now!

4.2.3.2 Configuring HPS Output Clocks (Output Clocks is NEW in Version 14.0)

4.2.3.2.1 HPS Output Clocks: Clock Sources

Please leave the settings to the default:

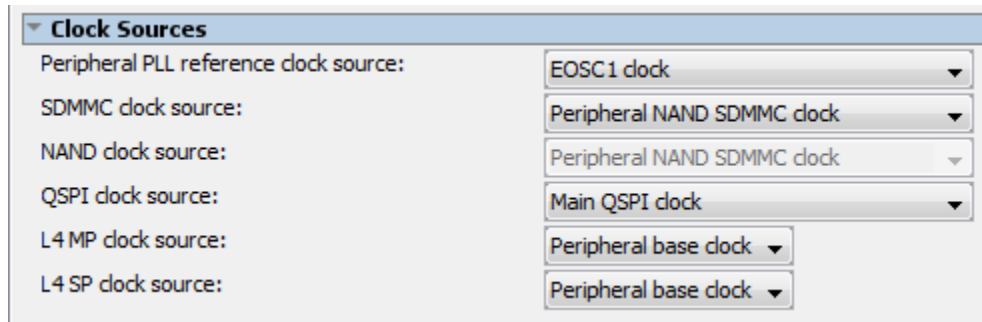


Figure 4.2.3.6

The bullets below are descriptive. Do NOT implement these changes.

Explanation of user clock options for the Output tab:

- **Peripheral PLL reference clock source**: This selects the reference clock for the HPS peripherals and can be set to the: EOSC1 Clock, EOSC2 Clock, FPGA-to-HPS peripheral reference clock. EOSC1 & EOSC2 are based upon the frequency at the HPS I/O pins: HPS_CLK1 and HPS_CLK2, D25 and F25 for the Cyclone V SX 5CSXFC6D6F31C6. The FPGA-to-HPS peripheral reference clock is set under the Input tab. Please refer to Figure 4.2.3.7
- **SDMMC clock source** – Sets the clock source for the Secure Digital/Multimedia Card controller
FPGA-to-HPS peripheral reference clock – f2h_periph_ref_clk as defined in the input Clock tab
Main NAND SDMMC Clock – Selects the PLL output from the “Main Clock Group” MPU, L3 & L4 & Debug PLL C4 from the HPS Clock Manager
Peripheral NAND SDMMC Clock – Selects the PLL output from the “Peripheral Clock Group” Peripheral PLL C3 from the HPS Clock Manager
Please refer to Figure 4.2.3.8
- **NAND clock source** – Sets the clock source for the NAND Flash controller
FPGA-to-HPS peripheral reference clock – f2h_periph_ref_clk as defined in the input Clock tab
Main NAND SDMMC Clock – Selects the PLL output from the “Main Clock Group” MPU, L3 & L4 & Debug PLL C4 from the HPS Clock Manager
Peripheral NAND SDMMC Clock – Selects the PLL output from the “Peripheral Clock Group” Peripheral PLL C3 from the HPS Clock Manager
Please refer to Figure 4.2.3.9

- QSPI clock source** – Sets the clock source for the QSPI controller
 FPGA-to-HPS peripheral reference clock – f2h_periph_ref_clk as defined in the input Clock tab
 Main QSPI Clock – Selects the PLL output from the “Main Clock Group” MPU, L3 & L4 & Debug PLL C3 from the HPS Clock Manager
 Peripheral QSPI Clock – Selects the PLL output from the “Peripheral Clock Group” Peripheral PLL C2 from the HPS Clock Manager
 Please refer to Figure 4.2.3.10
- L4MP clock source** – L4 Master Peripheral Clock: l4_mp_clk
 Main Clock – Selects the PLL output from the “Main Clock Group” main_base_clk or C1 from the Main PLL
 Peripheral base clock – Selects the PLL output from the “Peripheral Clock Group” from the Peripheral PLL C4
- L4SP clock source** – L4 Slave Peripheral Clock: l4_sp_clk
 Main clock - Selects the PLL output from the “Main Clock Group” main_base_clk or C1 from the Main PLL
 Peripheral base clock – Selects the PLL output from the “Peripheral Clock Group” from the Peripheral PLL C4

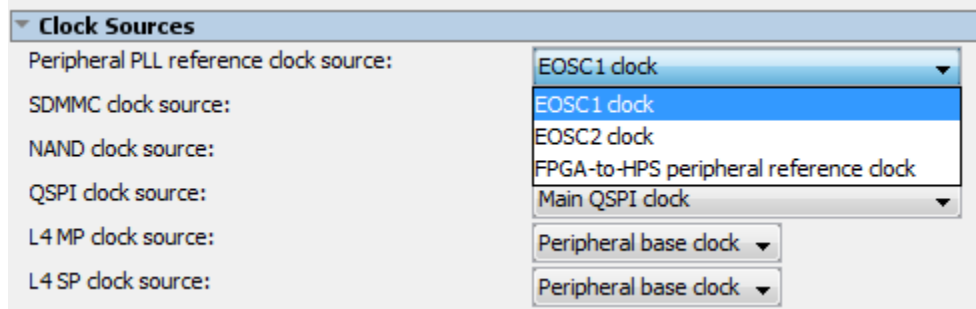


Figure 4.2.3.7

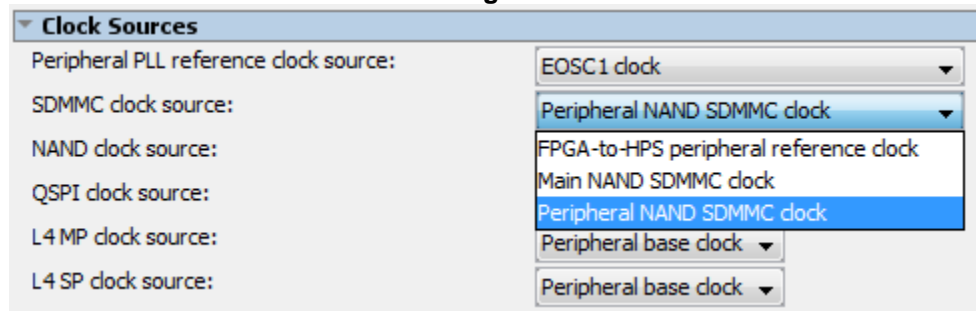


Figure 4.2.3.8

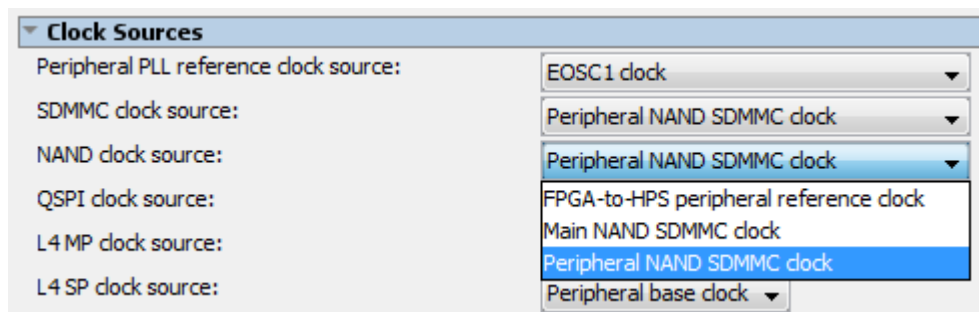


Figure 4.2.3.9

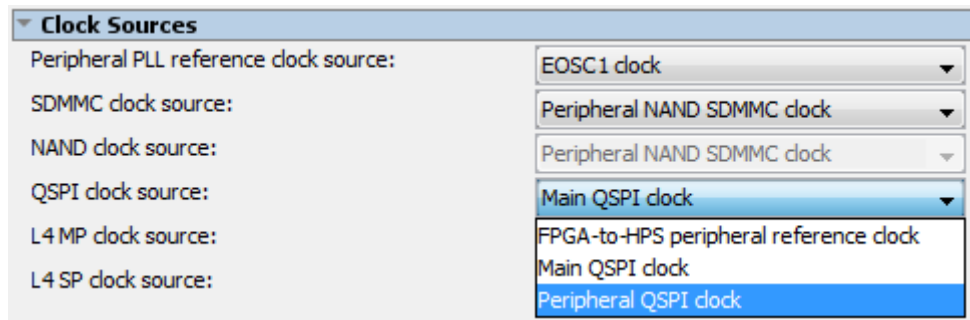


Figure 4.2.3.10

4.2.3.2.2 HPS Output Clocks: Main PLL Output Clocks – Desired Frequencies

The "EOSC1 clock frequency and EOSC2 clock frequency" External Clock Sources drive the HPS Clock manager PLLs (Main Clock Group, Peripheral Clock Group, SDRAM Clock Group & OSC1 Clock Group PLLs).

Please leave the settings (except uncheck Use default MPU clock frequency):

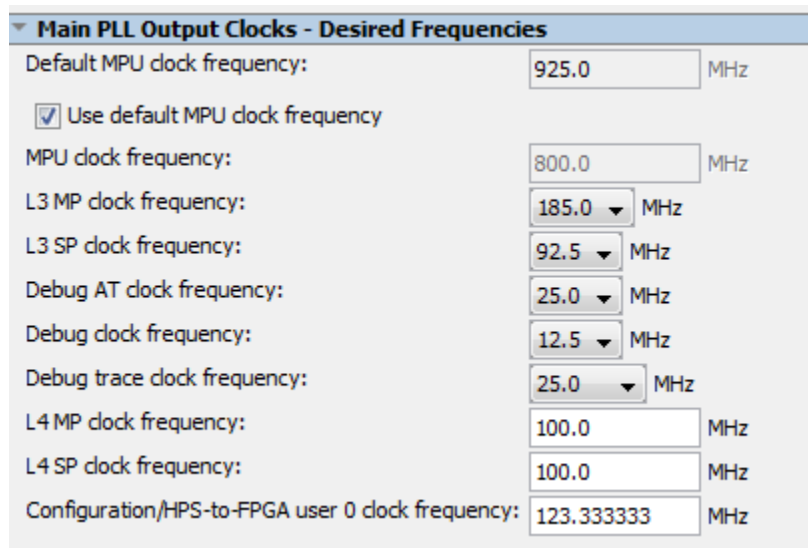


Figure 4.2.3.11

The bullets below are descriptive. Do NOT implement these changes.

- **MPU clock frequency** – Sets the clock frequency for the processor
☐ Use default MPU clock frequency must be unchecked to configure with own frequency
- **L3MP clock frequency** – L3 Master Peripheral clock frequency: l3_mp_clk
 Divides by 1 or 2 the PLL output from the main_base_clk or C1 from the Main PLL
- **L3SP clock source** – L3 Slave Peripheral Clock: l3_sp_clk

- Can be divided by 1 or 2 the L3PM clock (l3_mp_clk)
- **Debug AT clock frequency** – Debug AT Clock: dgb_at_clk
Derived from the Main PLL: C2 output and can be divided by 1 or 2
- **Debug Timer clock frequency** – Debug Timer Clock: dgb_timer_clk
Derived from the Main PLL: C2 output
- **Debug clock frequency** – Debug Clock: dgb_clk
Originates from dbg_at_clk divided by 2 or 4, derived from the Main PLL: C2 output
- **Debug trace clock frequency** – Debug trace clock: dgb_trace_clk
Originates from dbg_base_clk divided by 1, 2, 4, 8 or 16, derived from the Main PLL: C2 output
- **L4 MP clock frequency** – L4 Main Peripheral clock: l4_mp_clk
Originates from dbg_base_clk or periph_base_clk divided by 1, 2, 4, 8 or 16
- **L4 SP clock frequency** – L4 Slave Peripheral clock: l4_sp_clk
Originates from dbg_base_clk or periph_base_clk divided by 1, 2, 4, 8 or 16
- **Configuration HPS-to-FPGA user 0 clock frequency** – h2f_user0_clock
Originates from cfg_h2f_user0_base_clk which is the Main PLL: C5 output

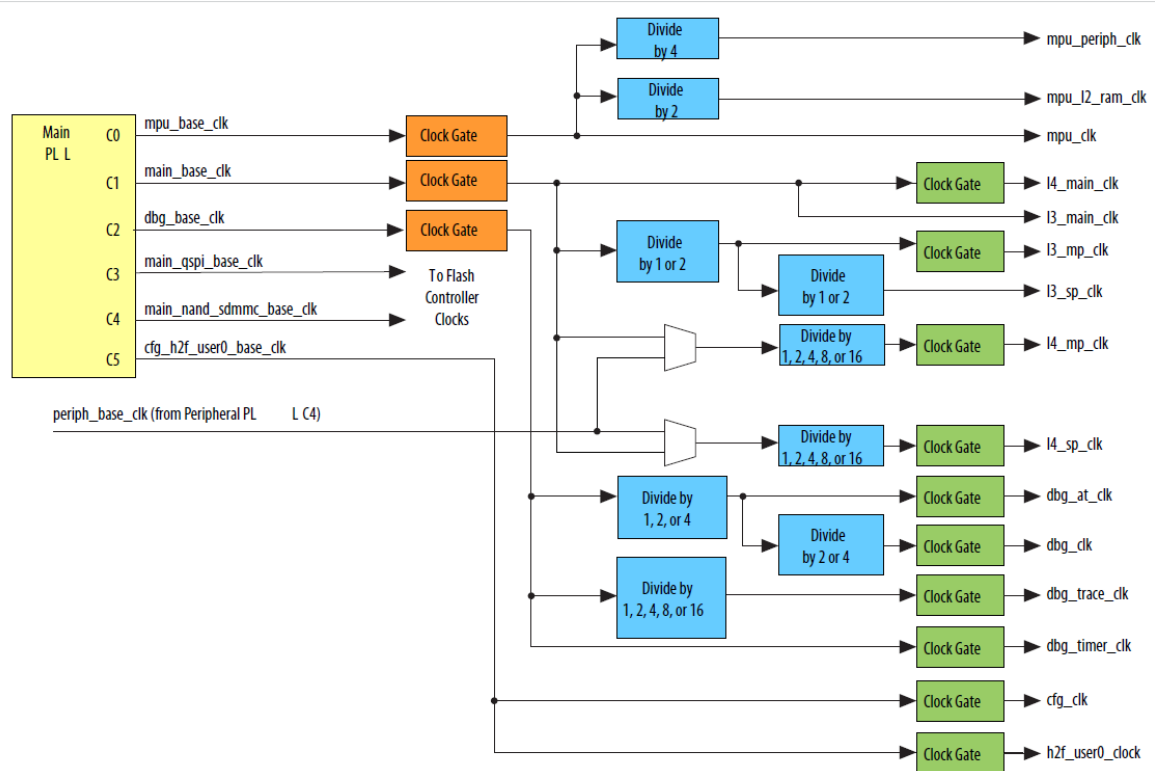


Figure 4.2.3.12

4.2.3.2.3 HPS Output Clocks: Peripheral Output Clocks – Desired Frequencies

Please leave the settings (except uncheck Use default MPU clock frequency):

Peripheral PLL Output Clocks - Desired Frequencies		
SDMMC clock frequency:	200.0	MHz
NAND clock frequency:	12.5	MHz
QSPI clock frequency:	370.0	MHz
EMAC0 clock frequency:	250.0	MHz
EMAC1 clock frequency:	250.0	MHz
USB clock frequency:	200.0	MHz
SPI clock frequency:	200.0	MHz
CAN0 clock frequency:	100.0	MHz
CAN1 clock frequency:	100.0	MHz
GPIO debounce clock frequency:	32000	Hz

Figure 4.2.3.13

- **SDMMC clock frequency** – Originates from the Peripheral PLL: C3 output: periph_nand_sdmmc_base_clk
- **NAND clock frequency** – Originates from the Peripheral PLL: C3 output: periph_nand_sdmmc_base_clk
- **QSPI clock frequency** – Originates from the Peripheral PLL: C3 output: periph_qspi_base_clk
- **EMAC0 clock frequency** – emac0_base_clk
Originates from the Peripheral PLL, C0 output: emac1_base_clk
- **EMAC1 clock frequency** – emac1_base_clk
Originates from the Peripheral PLL, C1 output: emac1_base_clk
- **USB clock frequency** – usb_mp_clk
Originates from the Peripheral PLL, C4 output: periph_base_clk
Can be divided by 1,2,4,6, or 16
- **SPI clock frequency** – spi_m_clk
Originates from the Peripheral PLL, C4 output: periph_base_clk
Can be divided by 1,2,4,6, or 16
- **CAN0 clock frequency** – can0_clk
Originates from the Peripheral PLL, C4 output: periph_base_clk
Can be divided by 1,2,4,6, or 16
- **CAN1 clock frequency** – can1_clk
Originates from the Peripheral PLL, C4 output: periph_base_clk
Can be divided by 1,2,4,6, or 16
- **GPIO clock frequency** – Originates from the Peripheral PLL, C4 output: periph_base_clk
Divided by 24

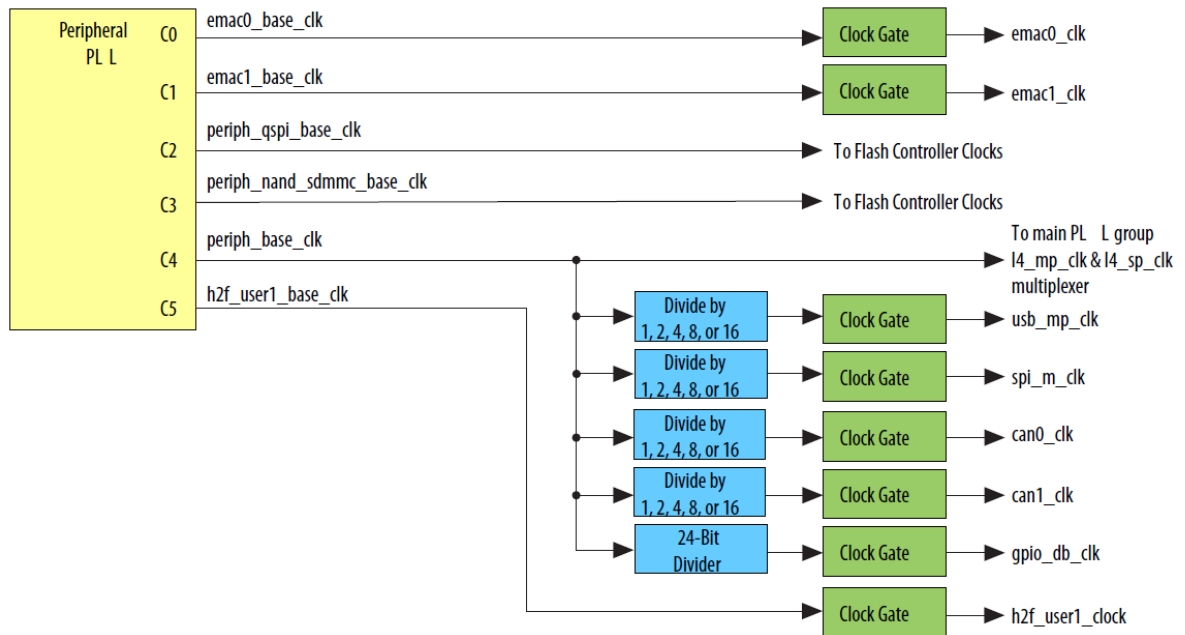


Figure 4.2.3.14

4.2.3.2.4 HPS Output Clocks: HPS-to-FPGA user clocks

Please leave the default settings:

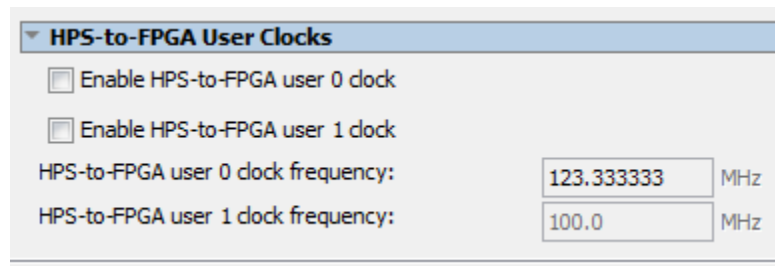


Figure 4.2.3.15

- **Enable HPS-to-FPGA user 0 clock** – h2f_user0_clock
Originates from cfg_h2f_user0_base_clk which is the Main PLL: C5 output
Refer to **Figure 4.2.3.12**
- **Enable HPS-to-FPGA user 1 clock**
Originates from h2f_user1_base_clk which is the Peripheral PLL: C5 output
Refer to **Figure 4.2.3.14**
- **Enable HPS-to-FPGA user 2 clock**
Originates from h2f_user0_base_clk which is the SDRAM PLL: C5 output
Refer to **Figure 4.2.4.2**

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 tab, the clock and Advanced PHY Settings are required.

Change the memory clock frequency from 300 MHz to 400 MHz (as shown in Figure 4.2.4.1). 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:

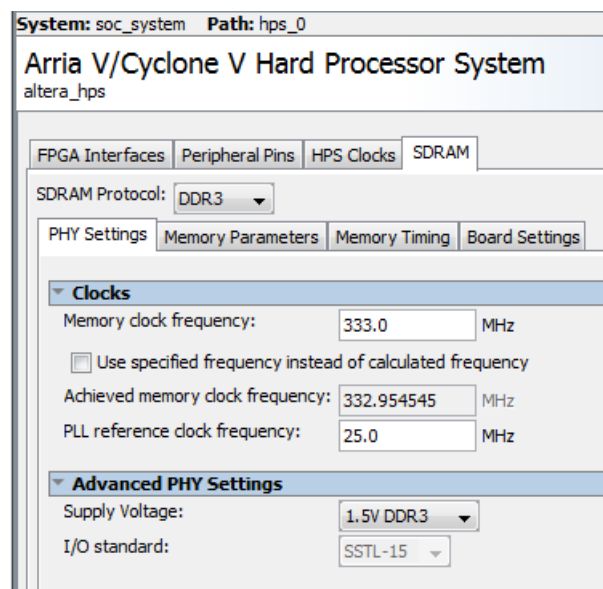


Figure 4.2.4.1

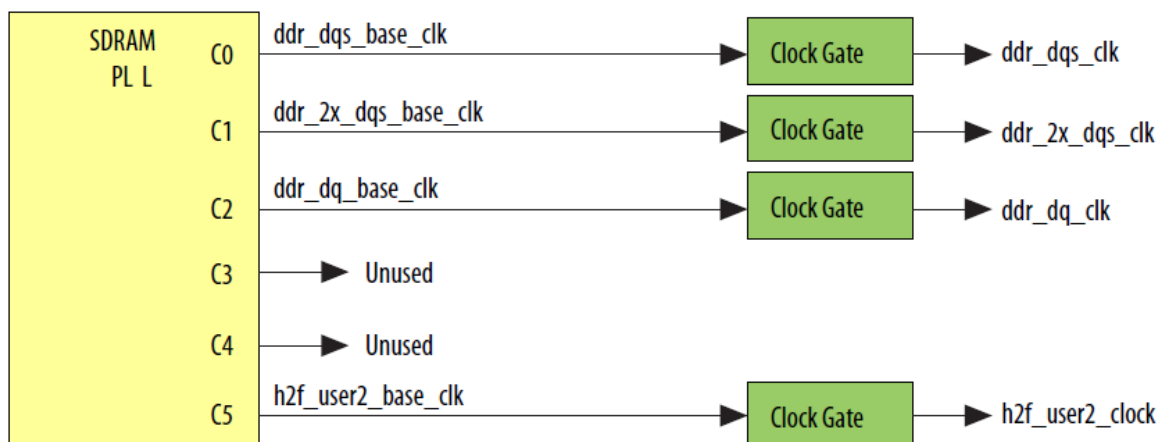


Figure 4.2.4.2

Select the **Memory Parameters tab** (as shown in Figure 4.2.4.2). 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

Figure 4.2.4.2

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

Verify the parameters that are selected as shown below:

Figure 4.2.4.3

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

Figure 4.2.4.4

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

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):	180	ps	
tIH (base):	140	ps	
tDS (base):	30	ps	
tDH (base):	65	ps	
tDQSQ:	125	ps	
tQH:	0.38	cycles	
tDQSCK:	255	ps	
tDQSS:	0.25	cycles	
tQSH:	0.4	cycles	
tDSH:	0.2	cycles	
tDSS:	0.2	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	
tRRD:	7.5	ns	
tRTP:	7.5	ns	

Figure 4.2.4.5

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.

▼ Setup and Hold Derating		
The slew rate of the output signals affects the setup and hold times of the memory device.		
You can specify the slew rate of the output signals to refer to their effect on the setup and hold times of both the address and command signals and the DQ signals, or specify the setup and hold times directly.		
Derating method:	<input checked="" type="radio"/> Use Altera's default settings <input type="radio"/> Specify slew rates to calculate setup and hold times <input type="radio"/> Specify setup and hold times directly	
CK/CK# slew rate (Differential):	2.0	V/ns
Address and command slew rate:	1.0	V/ns
DQS/DQS# slew rate (Differential):	2.0	V/ns
DQ slew rate:	1.0	V/ns
tIS:	0.33	ns
tIH:	0.24	ns
tDS:	0.18	ns
tDH:	0.165	ns
▼ Channel Signal Integrity		
Channel Signal Integrity is a measure of the distortion of the eye due to intersymbol interference or crosstalk or other effects. Typically when going from a single-rank configuration to a multi-rank configuration there is an increase in the channel loss as there are multiple stubs causing reflections. Please perform your channel signal integrity simulations and enter the extra channel uncertainty.		
Derating Method:	<input checked="" type="radio"/> Use Altera's default settings <input type="radio"/> Specify channel uncertainty values	
Address and command eye reduction (setup):	0.0	ns
Address and command eye reduction (hold):	0.0	ns
Write DQ eye reduction:	0.0	ns
Write Delta DQS arrival time:	0.0	ns
Read DQ eye reduction:	0.0	ns
Read Delta DQS arrival time:	0.0	ns

Figure 4.2.4.6

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:

Board Skews		
PCB traces can have skews between them that can cause timing margins to be reduced. Furthermore skews between different ranks can further reduce the timing margin in multi-rank topologies.		
<input type="button" value="Restore default values"/>		
Maximum CK delay to DIMM/device:	0.03	ns
Maximum DQS delay to DIMM/device:	0.02	ns
Minimum delay difference between CK and DQS:	0.09	ns
Maximum delay difference between CK and DQS:	0.16	ns
Maximum skew within DQS group:	0.01	ns
Maximum skew between DQS groups:	0.08	ns
Average delay difference between DQ and DQS:	0.0	ns
Maximum skew within address and command bus:	0.03	ns
Average delay difference between address and command and CK:	0.0	ns

Figure 4.2.4.7

At the top of the **Arria V/Cyclone V Hard Processor System** Parameters Window, close the window by selecting the **X**, as the settings for the Hard Processor System are now configured.

Parameters **X**

soc_system > hps_0

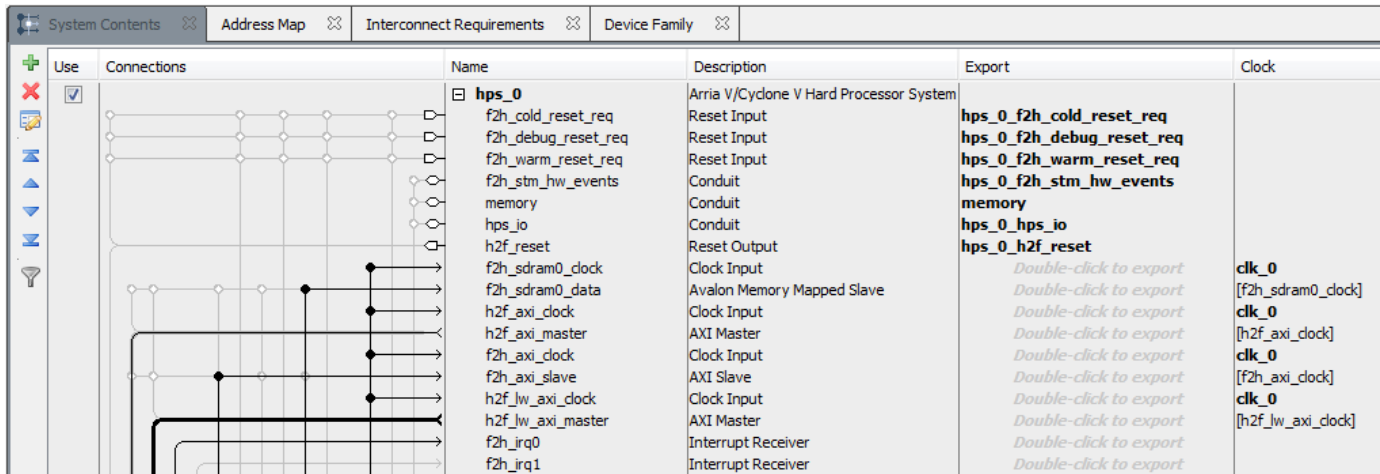
Arria V/Cyclone V Hard Processor System

altera_hps

Board Skews		
PCB traces can have skews between them that can cause timing margins to be reduced. Furthermore skews between different ranks can further reduce the timing margin in multi-rank topologies.		
<input type="button" value="Restore default values"/>		
Maximum CK delay to DIMM/device:	0.03	ns
Maximum DQS delay to DIMM/device:	0.02	ns
Minimum delay difference between CK and DQS:	0.09	ns
Maximum delay difference between CK and DQS:	0.16	ns
Maximum skew within DQS group:	0.01	ns
Maximum skew between DQS groups:	0.08	ns
Average delay difference between DQ and DQS:	0.0	ns
Maximum skew within address and command bus:	0.03	ns
Average delay difference between address and command and CK:	0.0	ns

Figure 4.2.4.8

In the export column associated with the HPS_0, there are five signals that need to be exported to the top level of the project. This is the reason why there are five signals already associated as shown in Figure 4.2.4.9 below.



Use	Connections	Name	Description	Export	Clock
<input checked="" type="checkbox"/>		hps_0	Arria V/Cyclone V Hard Processor System		
		f2h_cold_reset_req	Reset Input	hps_0_f2h_cold_reset_req	
		f2h_debug_reset_req	Reset Input	hps_0_f2h_debug_reset_req	
		f2h_warm_reset_req	Reset Input	hps_0_f2h_warm_reset_req	
		f2h_stm_hw_events	Conduit	hps_0_f2h_stm_hw_events	
		memory	Conduit	memory	
		hps_io	Conduit	hps_0_hps_io	
		h2f_reset	Reset Output	hps_0_h2f_reset	
		f2h_sdram0_clock	Clock Input	Double-click to export	clk_0
		f2h_sdram0_data	Avalon Memory Mapped Slave	Double-click to export	[f2h_sdram0_clock]
		h2f_axi_clock	Clock Input	Double-click to export	clk_0
		h2f_axi_master	AXI Master	Double-click to export	[h2f_axi_clock]
		f2h_axi_clock	Clock Input	Double-click to export	clk_0
		f2h_axi_slave	AXI Slave	Double-click to export	[f2h_axi_clock]
		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_clock]
		f2h_irq0	Interrupt Receiver	Double-click to export	
		f2h_irq1	Interrupt Receiver	Double-click to export	

Figure 4.2.4.9

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:

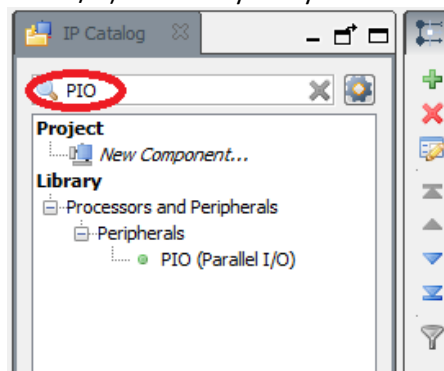


Figure 4.2.5.1

- 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** (as shown in Figure 4.2.5.2)

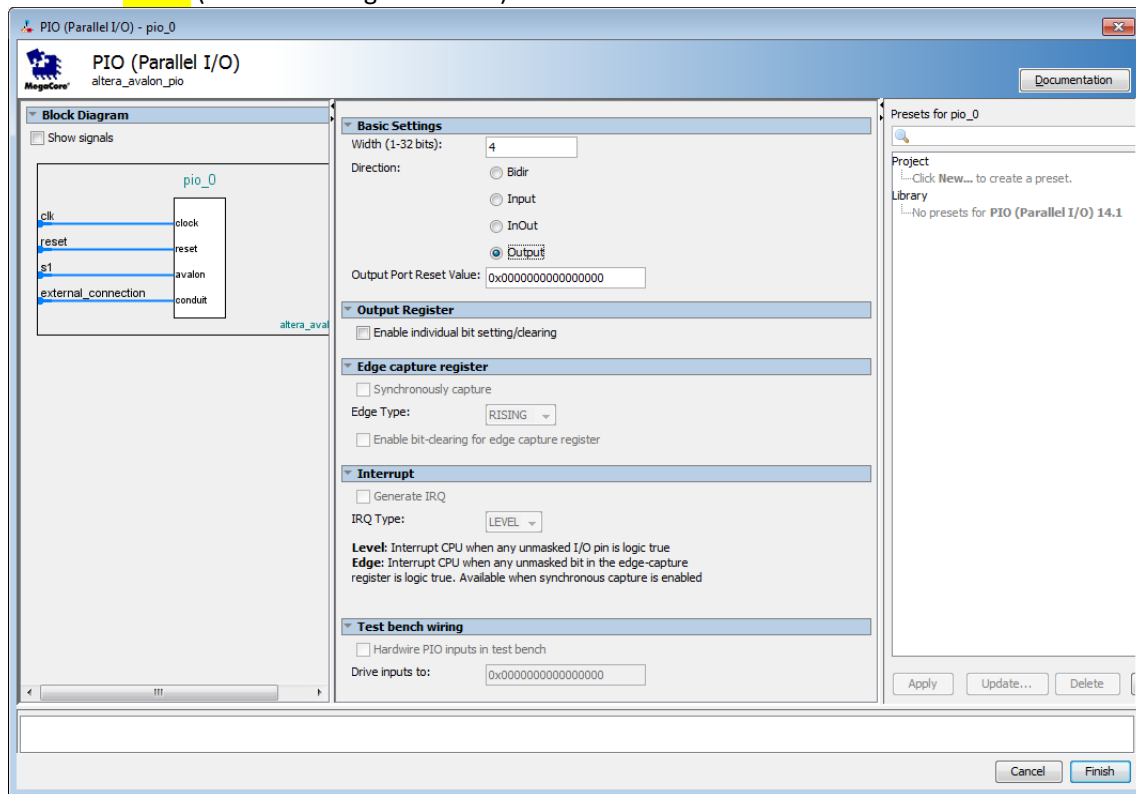


Figure 4.2.5.2

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 the following should automatically show up: **led_pio_external_connection**. If not, then type: **led_pio_external_connection**

The component should now look like:

led_pio	PIO (Parallel I/O)		
clk	Clock Input	<i>Double-click to export</i>	unconnected
reset	Reset Input	<i>Double-click to export</i>	[clk]
s1	Avalon Memory Mapped Slave	<i>Double-click to export</i>	[clk]
external_connection	Conduit	led_pio_external_connection	

Figure 4.2.5.3

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:

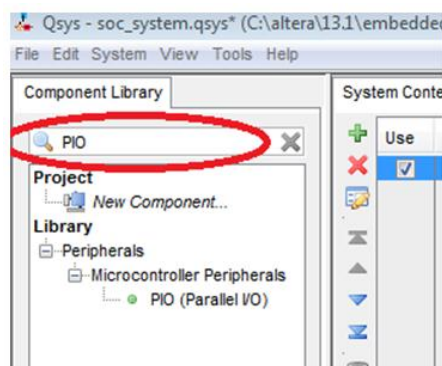


Figure 4.2.6.1

- Set the **width** to be **4**. Ensure 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**.
- Refer to Figure 4.2.6.2 before proceeding to check your settings
- Select **Finish**

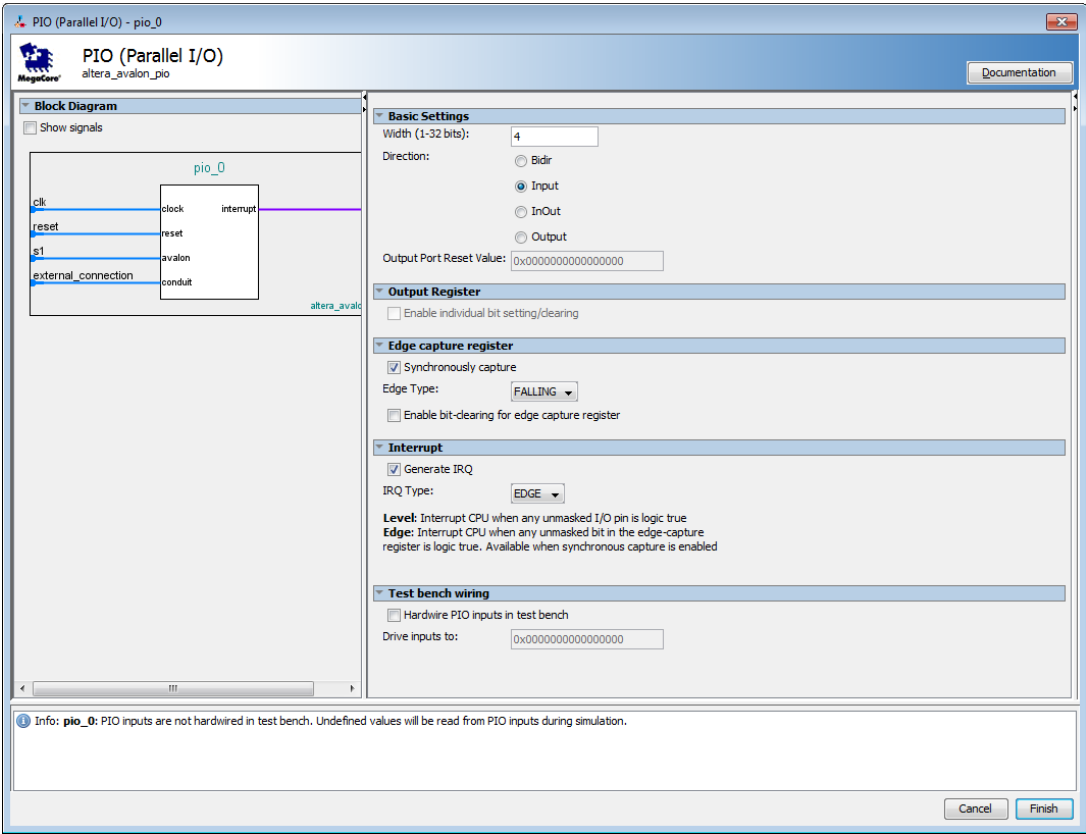


Figure 4.2.6.2

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.
- The following should automatically show up: **button_pio_external_connection**. If not, then type: **button_pio_external_connection**

The settings for the component should now look like:

<input type="checkbox"/> button_pio	PIO (Parallel I/O)		
clk	Clock Input	Double-click to export	unconnected
reset	Reset Input	Double-click to export	[clk]
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]
external_connection	Conduit	button_pio_external_connection	

Figure 4.2.6.2

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.

Type	Path	Message
4 Errors		
	soc_system.led_pio	led_pio.clk must be connected to a clock output
	soc_system.button_pio	button_pio.clk must be connected to a clock output
	soc_system.led_pio	led_pio.reset must be connected to a reset source
	soc_system.button_pio	button_pio.reset must be connected to a reset source

Figure 4.3.1.1

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:

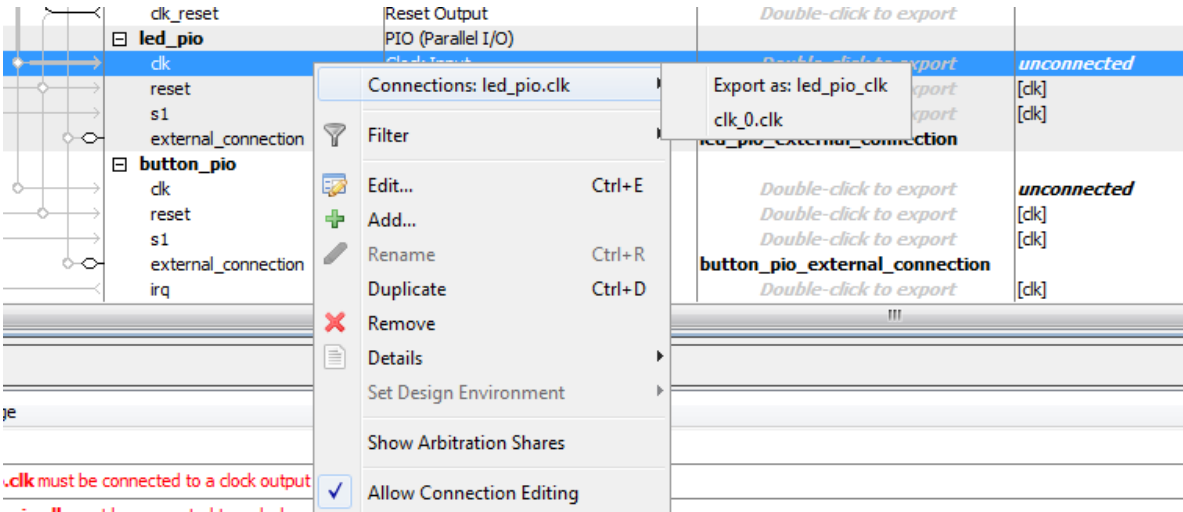


Figure 4.3.1.2

- 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	fpga_only_master	master
led_pio	s1	mm_bridge_0	m0
button_pio	clk	clk_0	clk
button_pio	reset	clk_0	clk_reset
button_pio	s1	fpga_only_master	master
button_pio	s1	mm_bridge_0	m0

Table 4.3.1.1

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 be connected to the IRQs and then the interrupts level needs to be assigned.

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

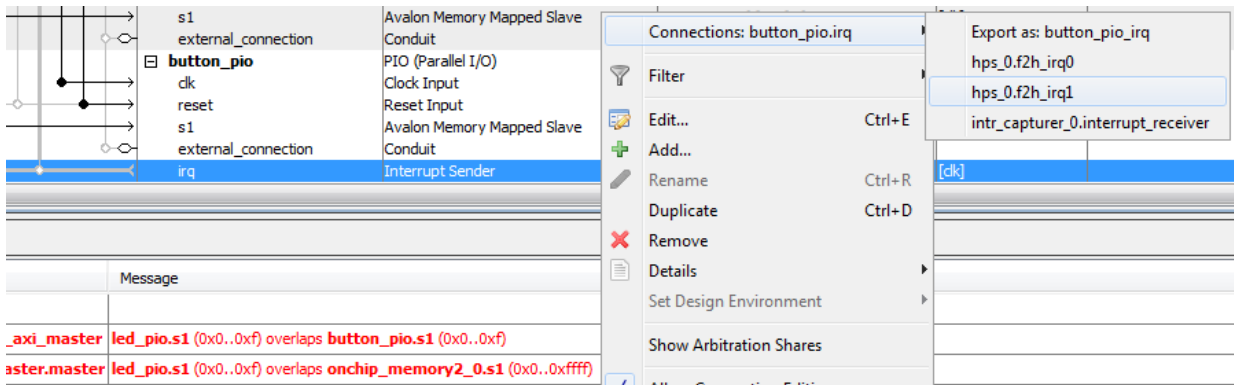


Figure 4.3.2.1

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

The interrupts should now look like:

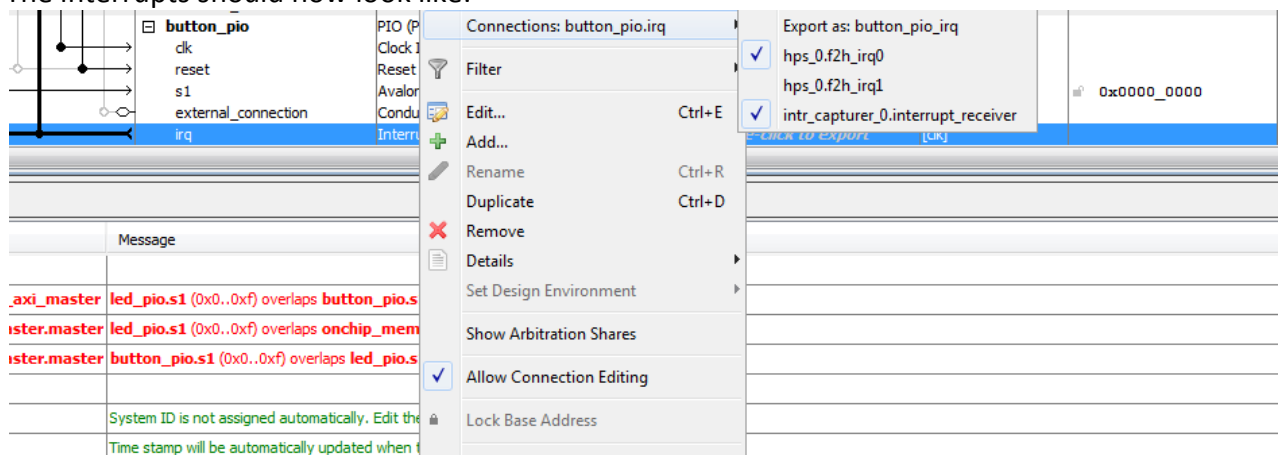


Figure 4.3.2.2

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**.

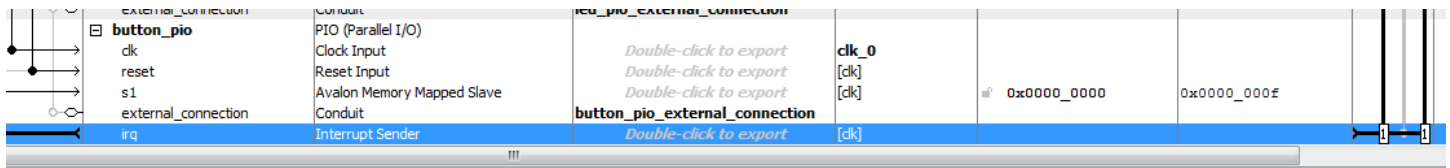


Figure 4.3.2.3

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:

System: soc_system Path: button_pio.irq	f2sdram_only_master.master	fpga_only_master.master	hps_0.h2f_axi_master	hps_0.h2f_lw_axi_master	hps_only_master.master	mm_bridge_0.m0
dpsw_pio.s1		0x0001_0080 - 0x0001_008f				0x0001_0080 - 0x0001_008f
hps_0.f2h_sdram0_data	0x0000_0000 - 0xffff_ffff					
hps_0.f2h_axi_slave					0x0000_0000 - 0xffff_ffff	
intr_capturer_0.avalon_slave_0		0x0003_0000 - 0x0003_0007				
jtag_uart.avalon_jtag_slave		0x0002_0000 - 0x0002_0007				0x0002_0000 - 0x0002_0007
mm_bridge_0.s0				0x0000_0000 - 0x0003_ffff		
onchip_memory2_0.s1		0x0000_0000 - 0x0000_ffff				0x0000_0000 - 0x0000_ffff
sysid_qsys.control_slave		0x0001_0000 - 0x0001_0007				0x0001_0000 - 0x0001_0007
led_pio.s1		0x0001_0040 - 0x0001_004f				0x0001_0040 - 0x0001_004f
button_pio.s1		0x0001_00c0 - 0x0001_00cf				0x0001_00c0 - 0x0001_00cf
onchip_memory2_0.s1 via mm_bridge_0				0x0000_0000 - 0x0000_ffff		
jtag_uart.avalon_jtag_slave via mm_b...				0x0002_0000 - 0x0002_0007		
dpsw_pio.s1 via mm_bridge_0				0x0001_0080 - 0x0001_008f		
button_pio.s1 via mm_bridge_0				0x0001_00c0 - 0x0001_00cf		
led_pio.s1 via mm_bridge_0				0x0001_0040 - 0x0001_004f		
sysid_qsys.control_slave via mm_bridg...				0x0001_0000 - 0x0001_0007		

Figure 4.3.3.1

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

Once these values are entered, go back to the **System Contents** tab and go to 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.

led_pio	PIO (Parallel I/O)				
clk	Clock Input	Double-click to export	clk_0		
reset	Reset Input	Double-click to export	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to export			
external_connection	Conduit	led_pio_external_connection			
button_pio	PIO (Parallel I/O)				
clk	Clock Input	Double-click to export	clk_0		
reset	Reset Input	Double-click to export	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to export			
external_connection	Conduit	button_pio_external_connection			
irq	Interrupt Sender	Double-click to export	[clk]		

Figure 4.3.3.4

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

4.3.4 Set AXI Bridge to Secure

The system has three JTAG to Avalon Master bridges and in Module 5 we want to allow system console the ability to Write to the HPS memory space from the FPGA. Specifically, to the HPS GPIO's (GPIO53, GPIO54, GPIO55 & GPIO56) associated with the LEDs on pins: A24, G21, C24 and E23. In order for the system to allow this capability, the hps_only_master.master port must be set to secure.

In Qsys select and right mouse click on "Connections", a drop down dialog box will appear as shown in Figure 4.3.4.1

Check "Show Security Column".

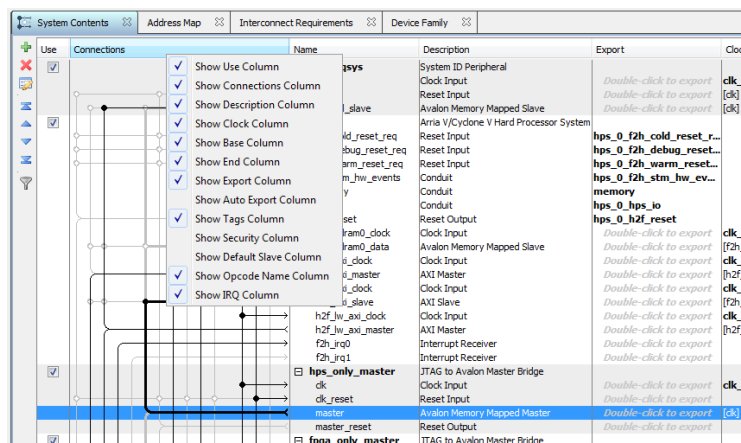


Figure 4.3.4.1

The “Security” Column will now be available in Qsys:

	Name	Description	Export	Clock	Base	End	IRQ	T...	Opcode Nam	Security
	sysid_qsys	System ID Peripheral								
	clk	Clock Input	Double-click to export	clk_0						
	reset	Reset Input	Double-click to export	[clk]						
	control_slave	Avalon Memory Mapped Slave	Double-click to export		0x0001_0000	0x0001_0007				Non-secure
	hps_0	Arria V/Cyclone V Hard Processor System								
	f2h_cold_reset_req	Reset Input	hps_0_f2h_cold_reset_r...							
	f2h_debug_reset_req	Reset Input	hps_0_f2h_debug_reset...							
	f2h_warm_reset_req	Reset Input	hps_0_f2h_warm_reset...							
	f2h_stm_hw_events	Conduit	hps_0_f2h_stm_hw_ev...							
	memory	Conduit	memory							
	hps_io	Conduit	hps_0_hps_io							
	h2f_reset	Reset Output	hps_0_h2f_reset							
	f2h_sdr0_clk	Clock Input	Double-click to export	clk_0						
	f2h_sdr0_data	Avalon Memory Mapped Slave	Double-click to export	[f2h_sdram...	0x0000_0000	0xFFFF_FFFF				Non-secure
	h2f_axi_clk	Clock Input	Double-click to export	clk_0						TrustZone-aware
hps_only_master	h2f_axi_master	AXI Master	Double-click to export	[h2f_axi_do...						
	f2h_axi_clk	Clock Input	Double-click to export	clk_0						
	f2h_axi_slave	AXI Slave	Double-click to export	[f2h_axi_do...	0x0000_0000	0xFFFF_FFFF				Non-secure
	h2f_lw_axi_clk	Clock Input	Double-click to export	clk_0						
	h2f_lw_axi_master	AXI Master	Double-click to export	[h2f_lw_axi...						TrustZone-aware
	f2h_irq0	Interrupt Receiver	Double-click to export				IRQ 0	IRQ 31		
	f2h_irq1	Interrupt Receiver	Double-click to export				IRQ 0	IRQ 31		
	hps_only_master	JTAG to Avalon Master Bridge								
	clk	Clock Input	Double-click to export	clk_0						
	clk_reset	Reset Input	Double-click to export							
	master	Avalon Memory Mapped Master	Double-click to export	[clk]						Non-secure

Figure 4.3.4.2

Select the master under “hps_only_master” and click on “Non-secure” and change to “Secure”:

	hps_only_master	JTAG to Avalon Master Bridge	Double-click to export							
	clk	Clock Input	Double-click to export	clk_0						
	clk_reset	Reset Input	Double-click to export							
	master	Avalon Memory Mapped Master	Double-click to export	[clk]						Non-secure
	master_reset	Reset Output	Double-click to export							
	fpga_only_master	JTAG to Avalon Master Bridge								
	clk	Clock Input	Double-click to export	clk_0						
	clk_reset	Reset Input	Double-click to export							
										Non-secure
										Secure

Figure 4.3.4.3

	hps_only_master	JTAG to Avalon Master Bridge								
	clk	Clock Input	Double-click to export	clk_0						
	clk_reset	Reset Input	Double-click to export							
	master	Avalon Memory Mapped Master	Double-click to export	[clk]						Secure

Figure 4.3.4.4

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

4.3.5 Block Diagram of the Golden Hardware Reference Design

The Golden Hardware Reference Design is an important part of the GSRD and consists of the following components:

- ARM Cortex™-A9 MPCore HPS
- Four user push-button inputs
- Four user DIP switch inputs
- Four user I/O for LED outputs
- 64KB of on-chip memory
- JTAG to Avalon master bridges
- Interrupt capturer for use with System Console
- System ID

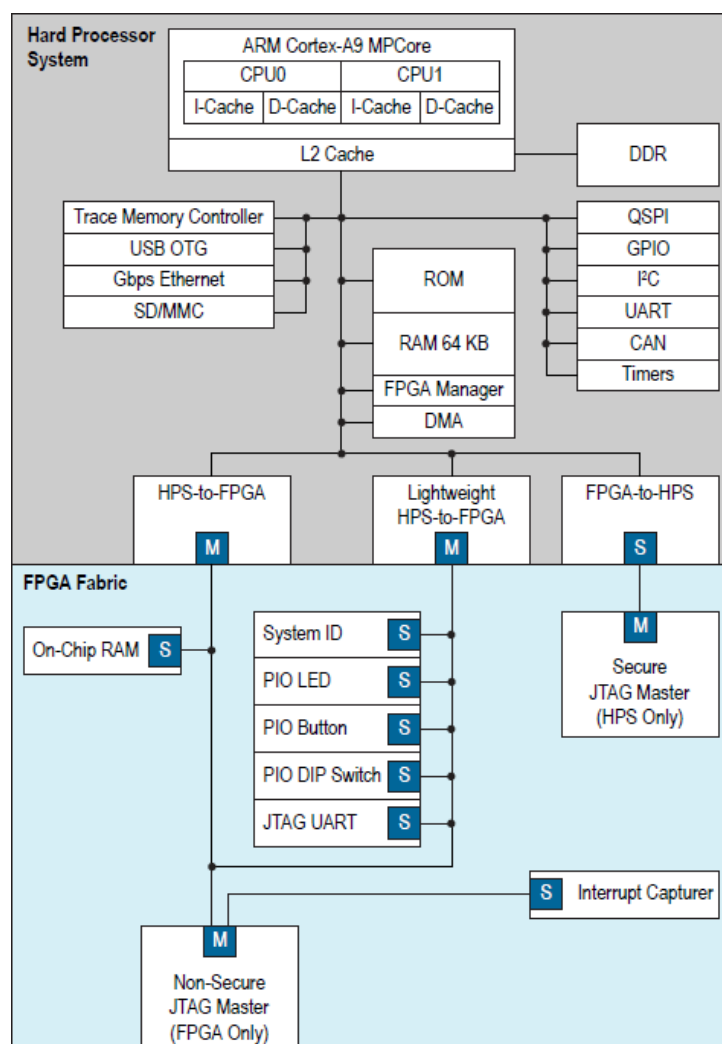


Figure 4.3.5.1

4.3.5.1 MPU Address Map

This section presents the address maps as seen from the MPU (A9) side.

HPS-to-FPGA Address Map

The memory map of soft IP peripherals, as viewed by the microprocessor unit (MPU), starts at HPS-to-FPGA address offset 0xC000_0000. The following table lists the offset of each peripheral in the FPGA portion of the SoC.

Peripheral	Address Offset	Size (bytes)	Attribute
onchip_memory2_0	0x0	64K	On-chip RAM as scratch pad

Table 4.3.5.1

Lightweight HPS-to-FPGA Address Map

The memory map of system peripherals in the FPGA portion of the SoC as viewed by the MPU, which starts at the **lightweight HPS-to-FPGA** base address **0xFF20_0000**, is listed in the following table.

Peripheral	Address Offset	Size (bytes)	Attribute
sysid_qsys	0x1_0000	8	Unique System ID
led_pio	0x1_0040	8	LED output display
dipsw_pio	0x1_0080	8	DIP Switch Input
button_pio	0x1_00c0	8	Push button Input
jtag_uart	0x2_0000	8	JTAG UART console

Table 4.3.5.2

4.3.5.2 JTAG Master Address Map

There are two JTAG master interfaces in the design, one for accessing **non-secure** peripherals in the FPGA fabric, and another for accessing **secure** peripheral in the HPS through the FPGA-to-HPS Interface. The following table lists the address of each peripheral in the FPGA portion of the SoC, as seen through the non-secure JTAG master interface.

Peripheral	Address Offset	Size (bytes)	Attribute
sysid_qsys	0x0001_0000	8	Unique System ID
led_pio	0x0001_0040	8	4 LED outputs
dipsw_pio	0x0001_0080	8	4 DIP Switch inputs
button_pio	0x0001_00c0	8	4 push button inputs
jtag_uart	0x0001_0000	8	JTAG UART console
onchip_memory2_0	0x0000_0000	64K	On-chip RAM

Table 4.3.5.3

4.3.5.3 Interrupt Routing

The HPS exposes 64 interrupt inputs for the FPGA logic. The following table lists the interrupts from soft IP peripherals to the HPS interrupt input interface.

Peripheral	Interrupt Number	Attribute
dipsw_pio	f2h_irq0[0]	4 DIP Switch Inputs
button_pio	f2h_irq0[1]	4 Push Button Inputs
jtag_uart	f2h_irq0[2]	JTAG UART

Table 4.3.5.4

The interrupt sources are also connected to an interrupt capturer module in the system, which enables System Console to be aware of the interrupt status of each peripheral in the FPGA portion of the SoC.

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

System: soc_system Path: onchip_memory2_0.s1									
Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	
<input checked="" type="checkbox"/>		sysid_qsys	System ID Peripheral						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		control_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_0000	0x0001_0007		
<input checked="" type="checkbox"/>		hps_0	Arria V/Cyclone V Hard Processor System						
		f2h_cold_reset_req	Reset Input	hps_0_f2h_cold_reset_req					
		f2h_debug_reset_req	Reset Input	hps_0_f2h_debug_reset_req					
		f2h_warm_reset_req	Reset Input	hps_0_f2h_warm_reset_req					
		f2h_stm_hw_events	Conduit	hps_0_f2h_stm_hw_events					
		memory	Conduit	memory					
		hps_io	Conduit	hps_0_hps_io					
		h2f_reset	Reset Output	hps_0_h2f_reset					
		f2h_sdram0_clock	Clock Input	Double-click to export	clk_0				
		f2h_sdram0_data	Avalon Memory Mapped Slave	Double-click to export	[f2h_sdram...	0x0000_0000	0xffff_ffff		
		h2f_axi_clock	Clock Input	Double-click to export	clk_0				
		h2f_axi_master	AXI Master	Double-click to export	[h2f_axi_do...				
		f2h_axi_clock	Clock Input	Double-click to export	clk_0				
		f2h_axi_slave	AXI Slave	Double-click to export	[f2h_axi_do...	0x0000_0000	0xffff_ffff		
		h2f_jw_axi_clock	Clock Input	Double-click to export	clk_0				
		h2f_jw_axi_master	AXI Master	Double-click to export	[h2f_jw_axi...				
		f2h_irq0	Interrupt Receiver	Double-click to export				IRQ 0	IRQ 31
		f2h_irq1	Interrupt Receiver	Double-click to export				IRQ 0	IRQ 31
<input checked="" type="checkbox"/>		hps_only_master	JTAG to Avalon Master Bridge						
		clk	Clock Input	Double-click to export	clk_0				
		clk_reset	Reset Input	Double-click to export	[clk]				
		master	Avalon Memory Mapped Master	Double-click to export					
		master_reset	Reset Output	Double-click to export					
<input checked="" type="checkbox"/>		fpga_only_master	JTAG to Avalon Master Bridge						
		clk	Clock Input	Double-click to export	clk_0				
		clk_reset	Reset Input	Double-click to export	[clk]				
		master	Avalon Memory Mapped Master	Double-click to export					
		master_reset	Reset Output	Double-click to export					
<input checked="" type="checkbox"/>		f2sdram_only_master	JTAG to Avalon Master Bridge						
		clk	Clock Input	Double-click to export	clk_0				
		clk_reset	Reset Input	Double-click to export	[clk]				
		master	Avalon Memory Mapped Master	Double-click to export					
		master_reset	Reset Output	Double-click to export					
<input checked="" type="checkbox"/>		mm_bridge_0	Avalon-MM Pipeline Bridge						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		s0	Avalon Memory Mapped Slave	Double-click to export		0x0000_0000	0x0003_ffff		
		m0	Avalon Memory Mapped Master	Double-click to export	[clk]				

Figure 4.4.1

Second half of the Qsys window

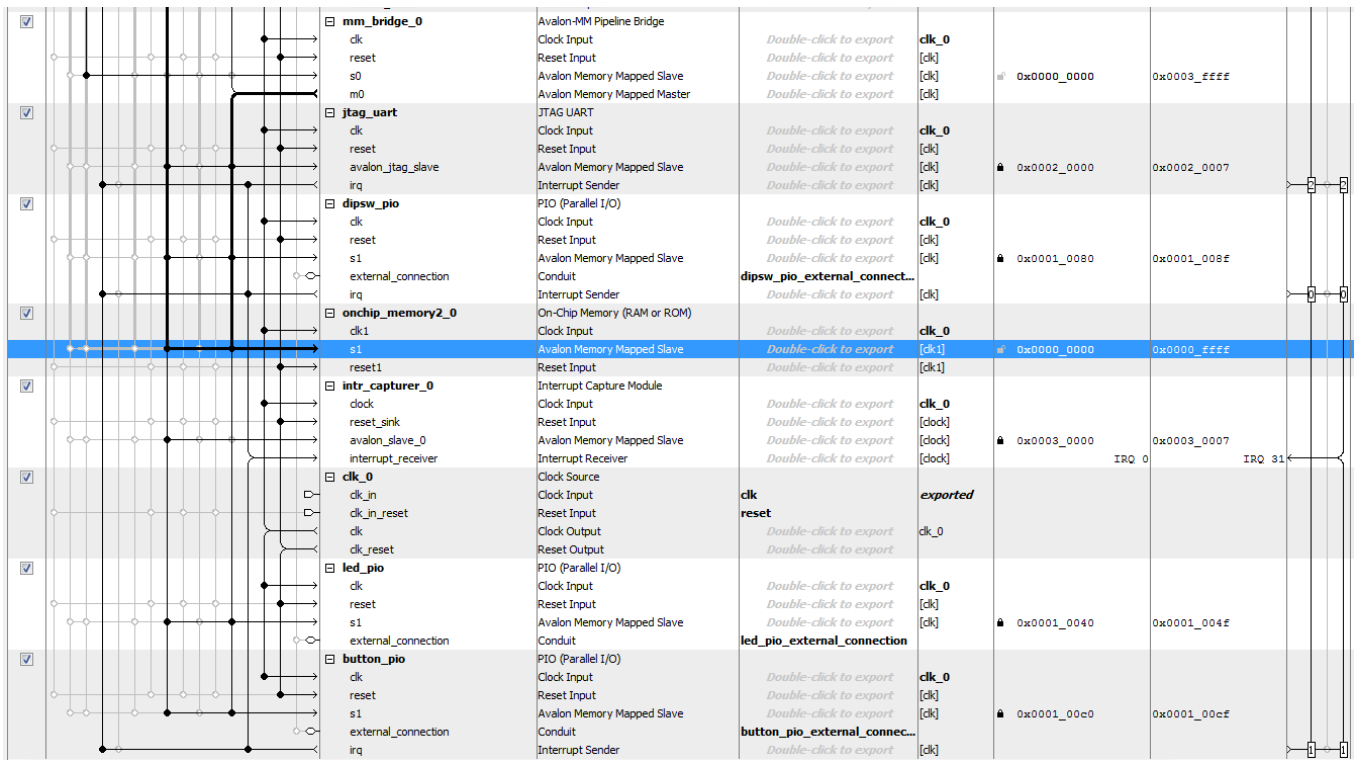


Figure 4.4.2

Select the **"Generate"** from the menu and then select **Generate HDL...** from the drop down menu.
Next select the **Generate** button.
If it asks you to save, select **Yes**.

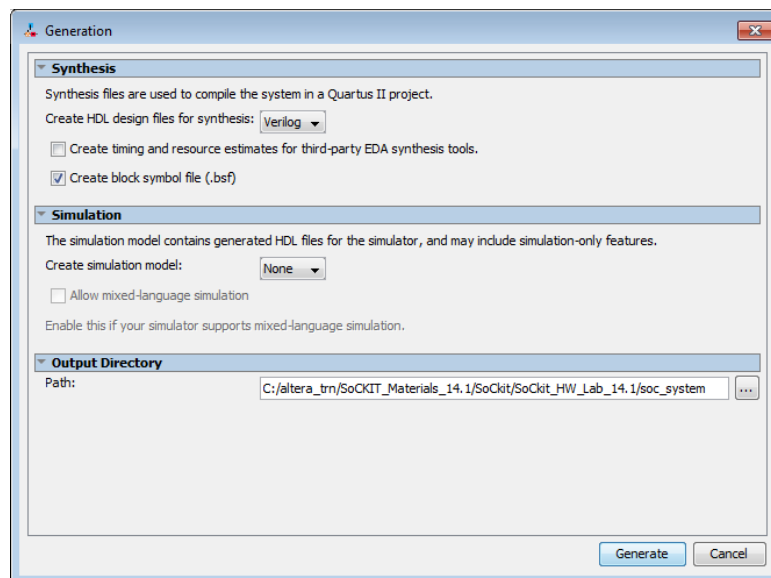


Figure 4.4.3

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

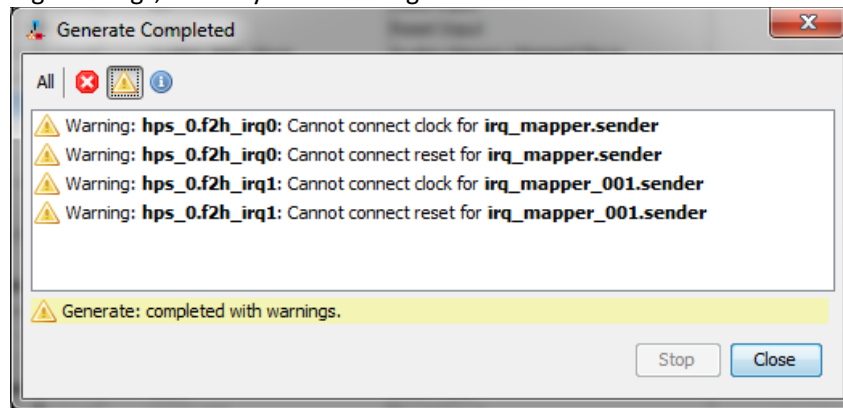


Figure 4.4.4

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

Next, select: **OK** when the message about adding the qip and sip files to the Quartus II project is displayed. This will be taken care of in MODULE 5: Complete the Quartus II Project

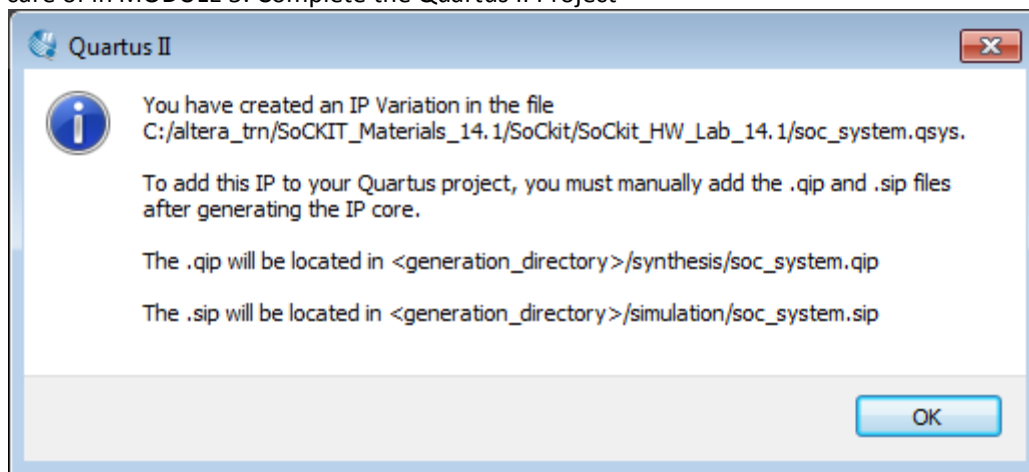


Figure 4.4.5

A simulation model for the System wasn't created; therefore, you will not find the */simulation/soc_system.sip file.

Qsys will **generate** 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, SIP files that include: the Simulation IP files required to complete a simulation. Both the QIP and SIP files use Tcl syntax.

In Quartus II you can find these files here by **selecting File -> Open**, then go to the **soc_system/synthesis** directory (select: All Files(*.*)) for all of the files to be displayed:

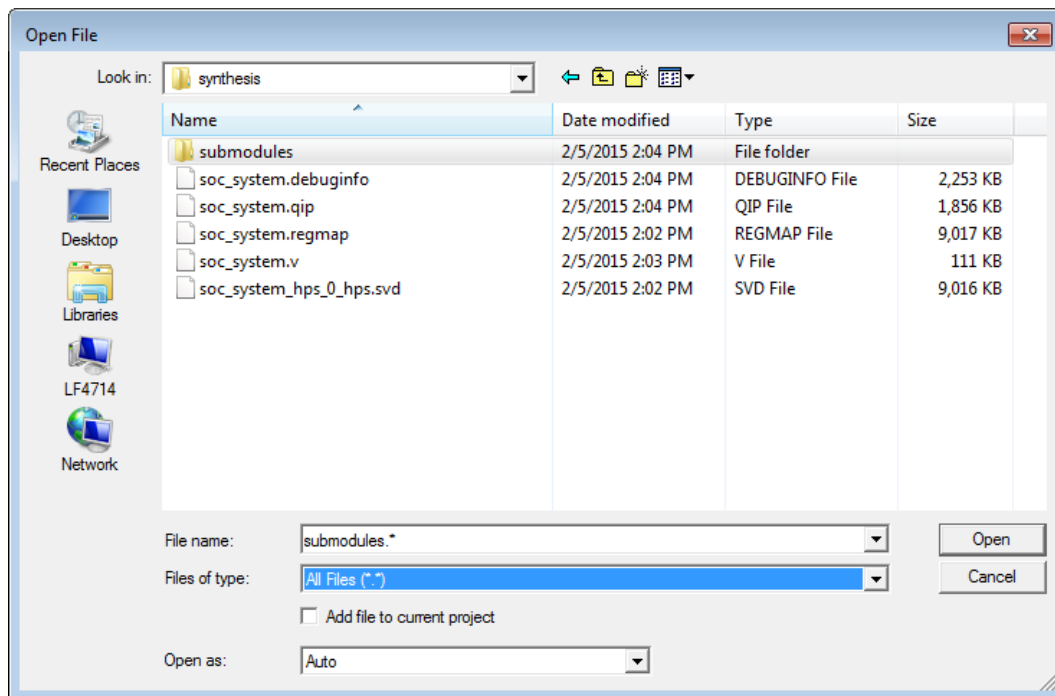


Figure 4.4.5

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

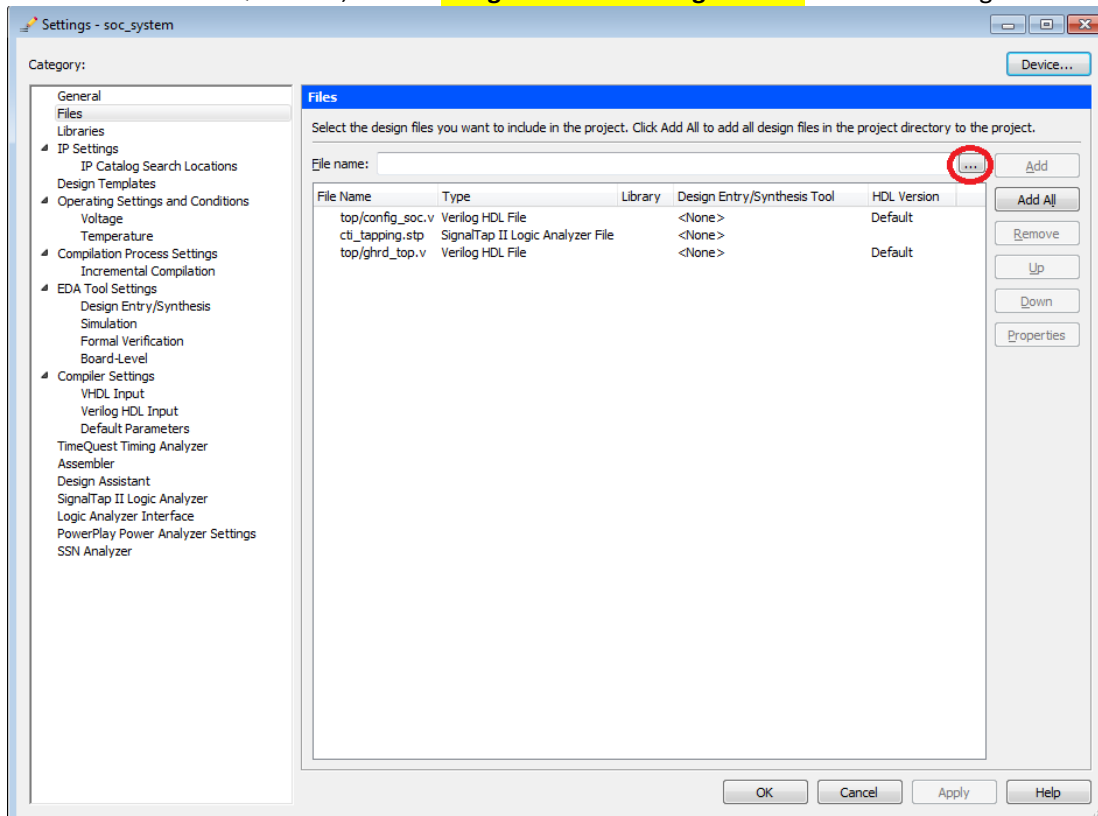


Figure 5.1.1

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

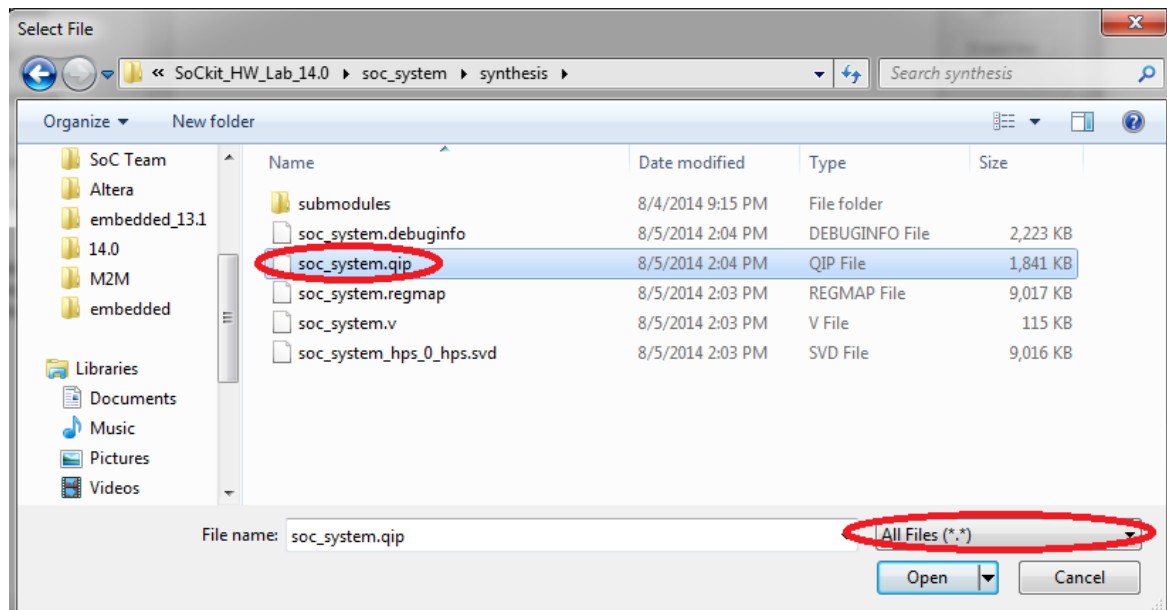


Figure 5.1.2

- Then, select "Open"
- Add the file to the project by selecting the "Add" button
- Select **Apply**.

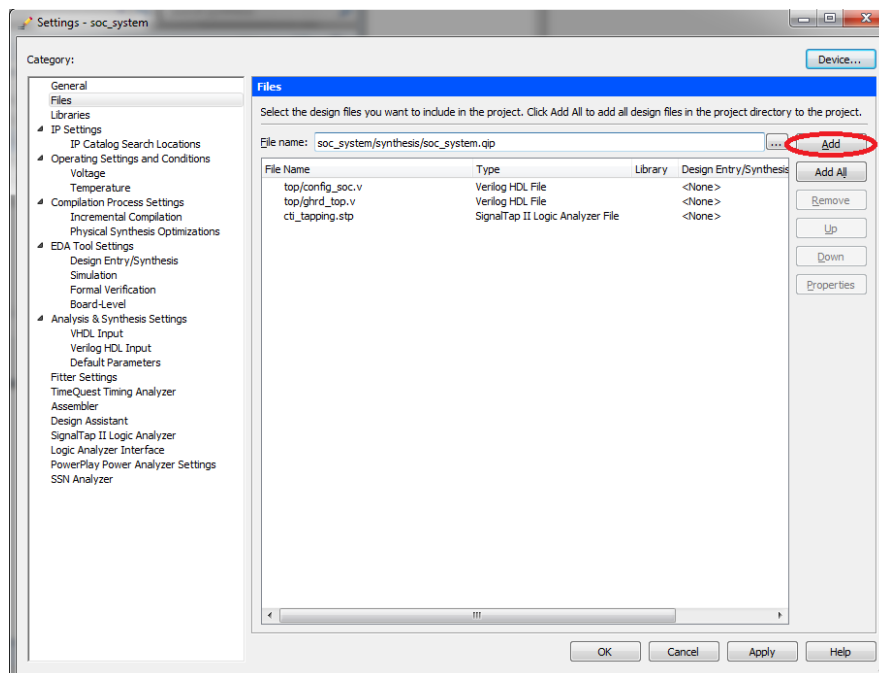


Figure 5.1.3

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

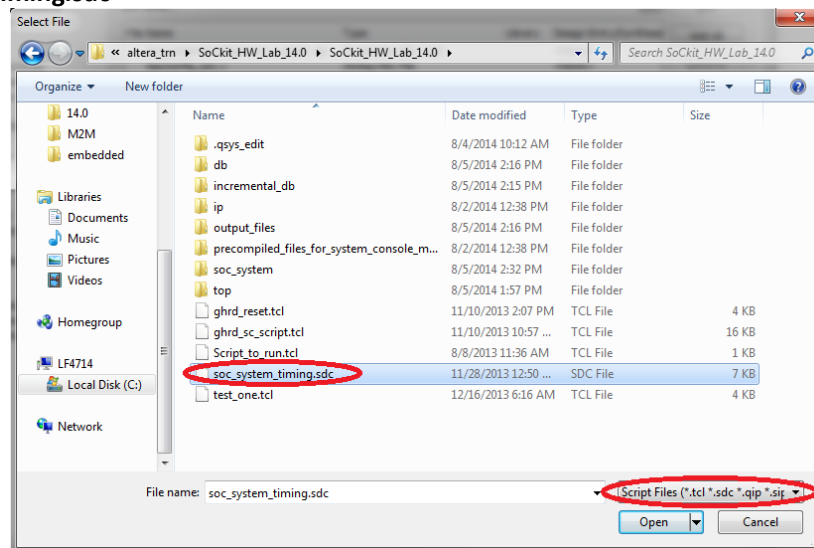


Figure 5.1.4

- Using the same window and use the same process in the two previous steps to add the
/ip/edge_detect/altera_edge_detector.v
/ip/altsource_probe/hps_reset.v
/ip/altsource_probe/hps_reset.qip
/ip/debounce/debounce.v
- The window should look like:

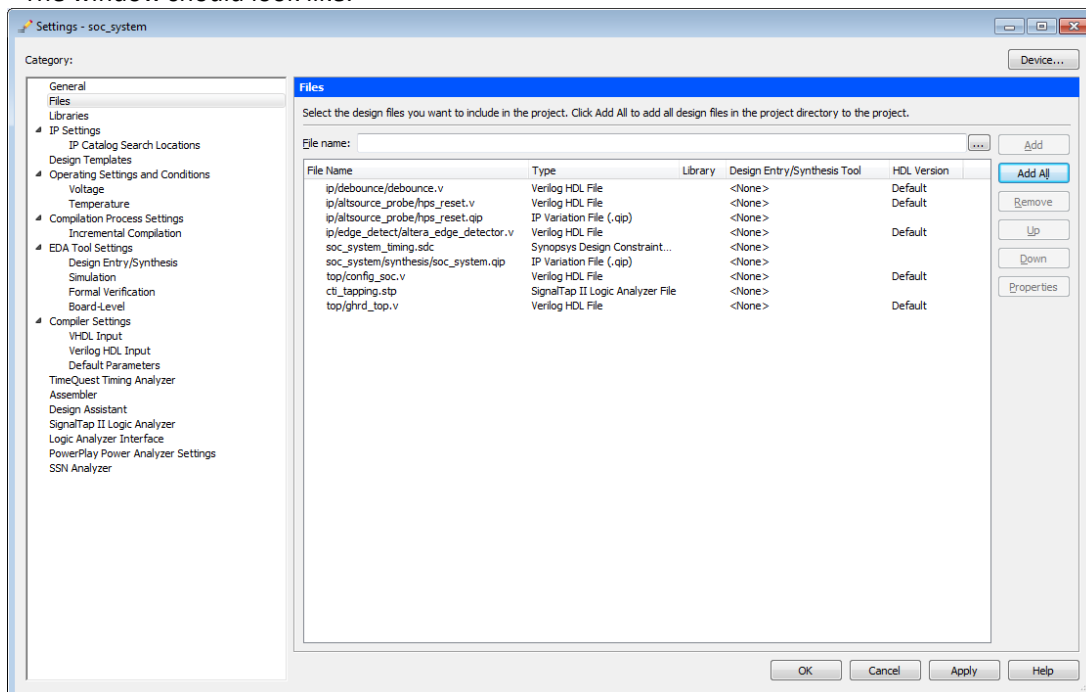


Figure 5.1.5

- 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:

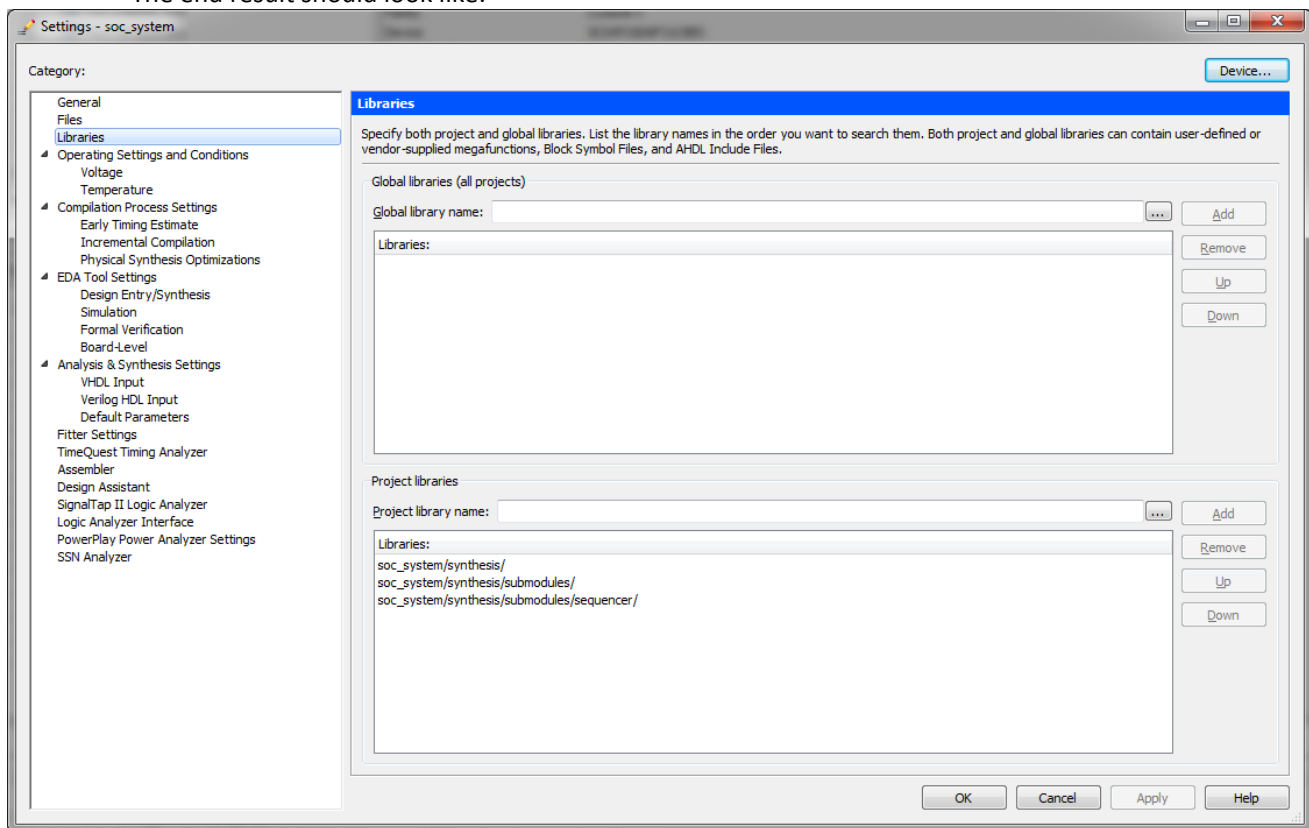


Figure 5.1.6

- Select Apply
- Select OK to finish.

5.2 Analysis and Synthesis

Before the pin-outs can be added to Quartus II an 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.

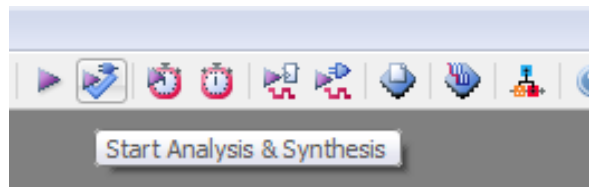


Figure 5.2.1

- 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 scripts...** and select the **hps_sdram_p0_pin_assignments.tcl** as seen below.

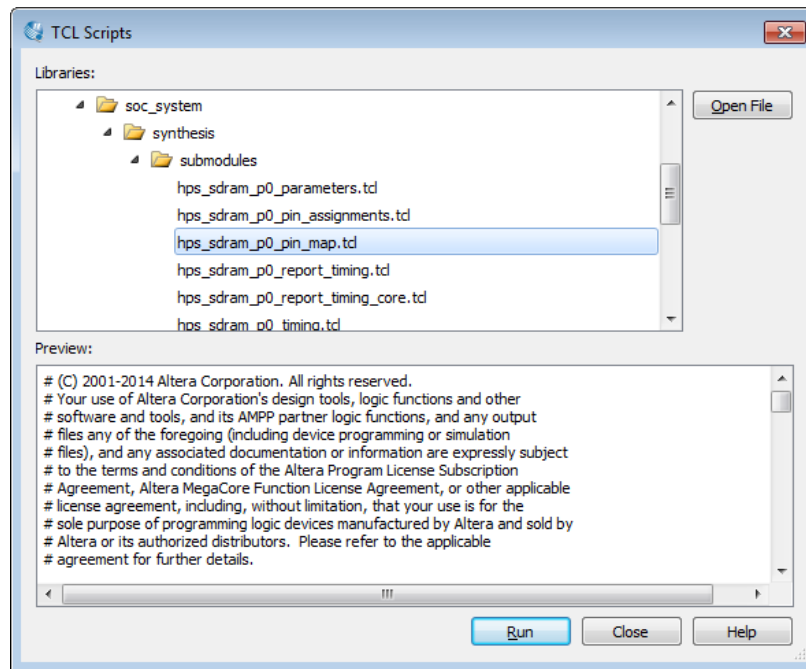


Figure 5.3.1

- Select **Run**.
- Select **Close** 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**" which you can find in the **output_files** directory:

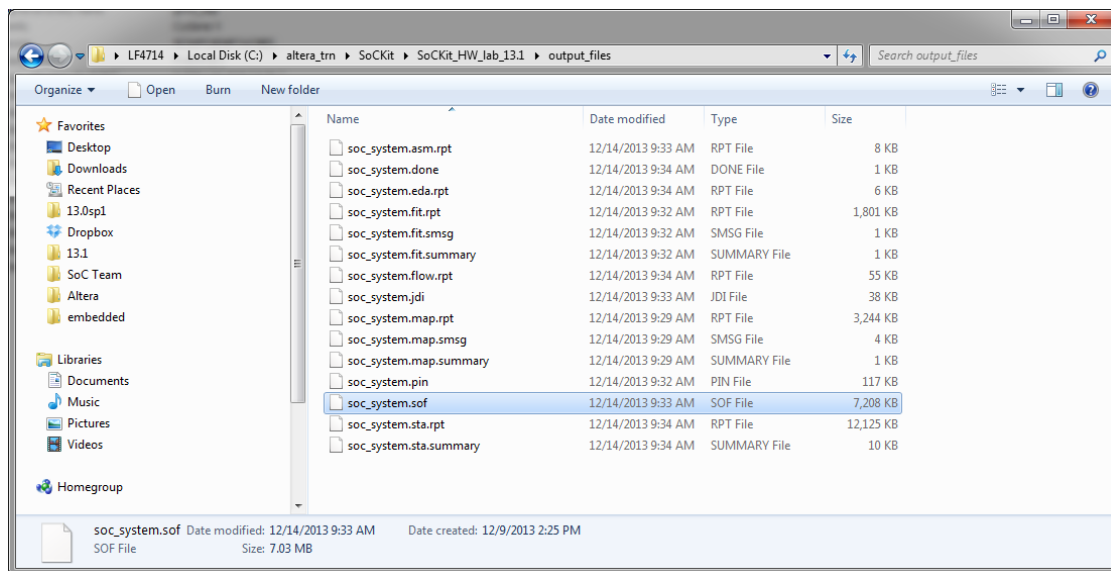


Figure 5.4.1


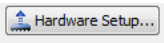
MODULE 6. Hardware Debug Flow (System Console)

Module Objective:

In this module you will debug the Qsys system by utilizing system console. We will execute a Tcl script in the system console tool to write data to the LED_PIO port. The Tcl script sets up the JTAG Master port in the Qsys system and then provides stimulus to write successive bytes of data to the LED_PIO port.

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**. OR Launch the **Quartus II v14.1** software: Select  -> **All Programs -> Altera 14.1.0.186 Web Edition -> Quartus II 14.1 Programmer**
- Select  (top left hand side of the Programming Window) and ensure that the currently selected hardware is CV SoCKit [USB-1]. It should be selected by default. If not currently selected; then **double click** on CV SoCKit.

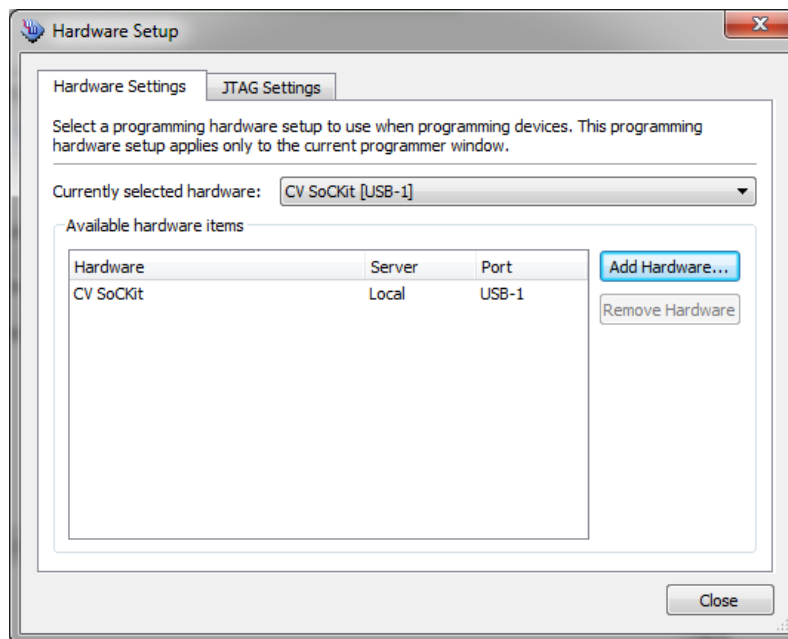


Figure 6.1.1

- Select **Close**.

- Select "Auto Detect"

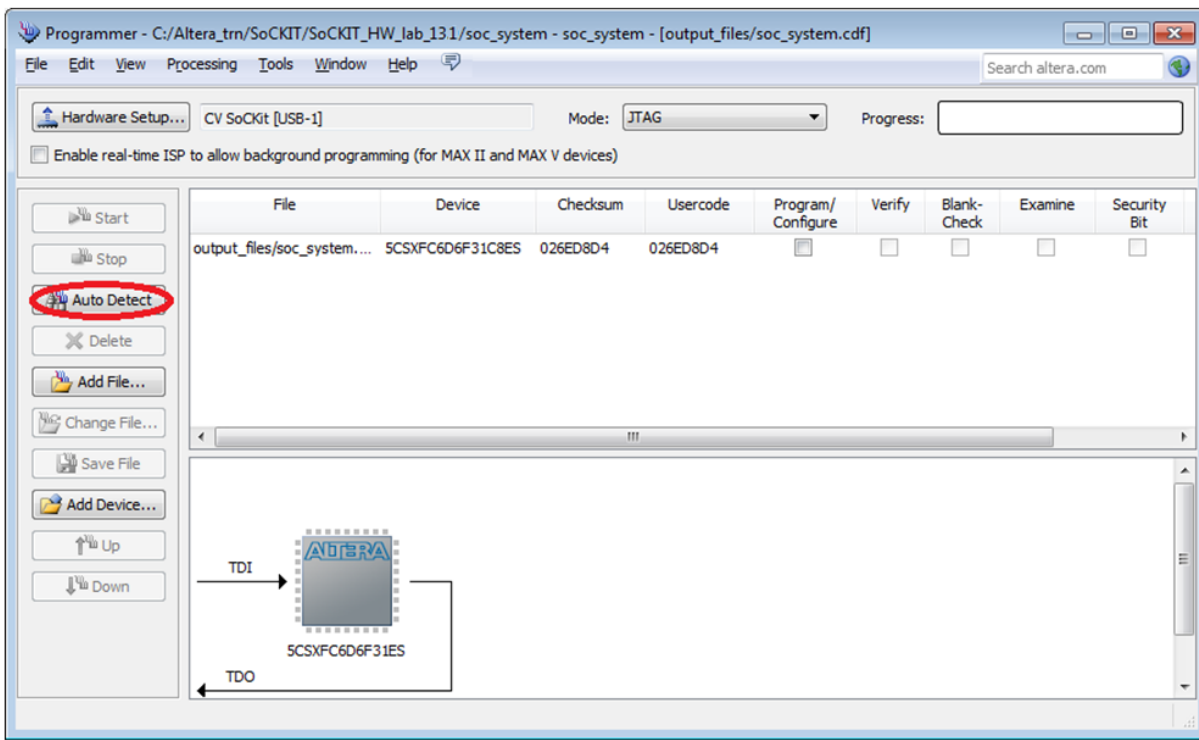


Figure 6.1.2

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

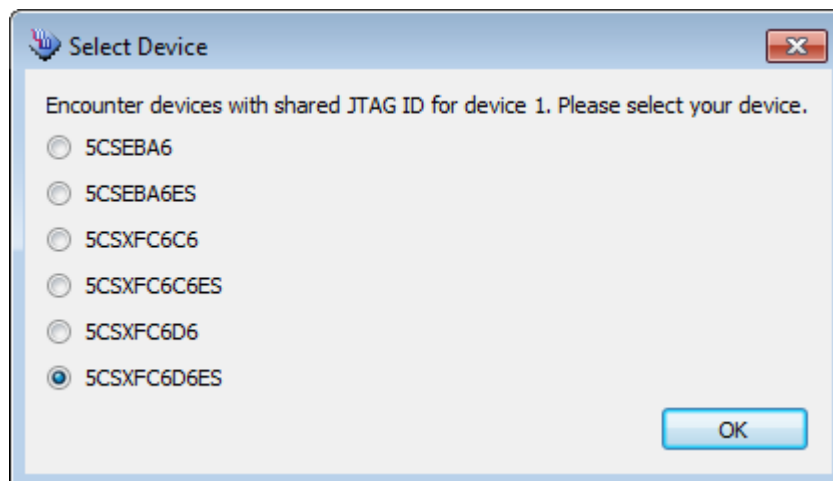


Figure 6.1.3

- If the following appears, Select **Yes**:

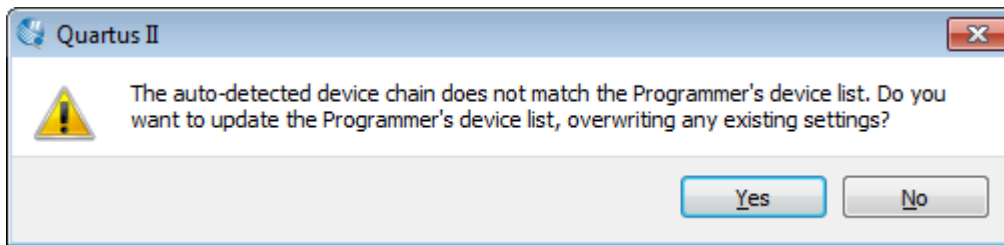


Figure 6.1.4

- The Programming window should now appear as shown in Figure 5.5.5:

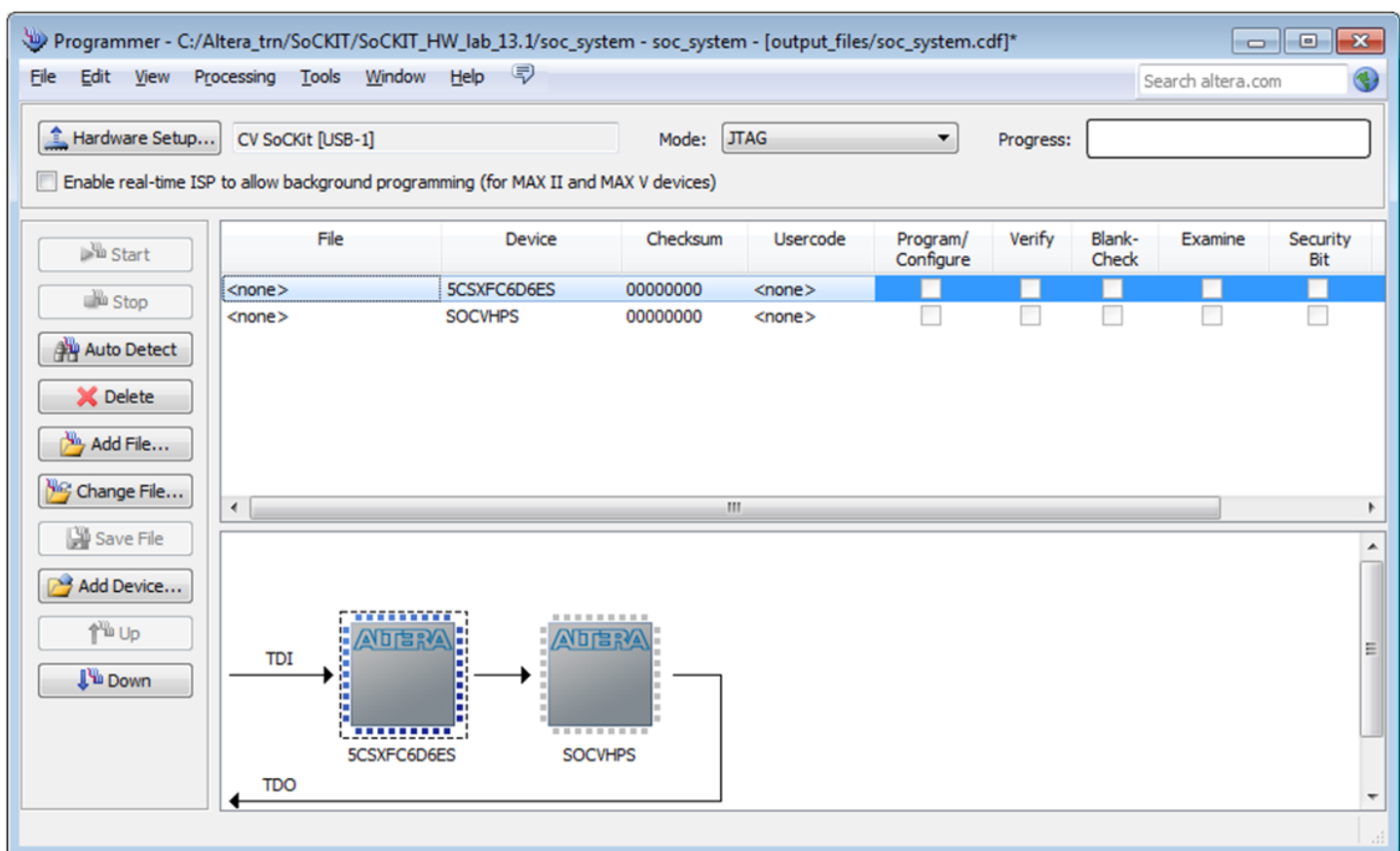
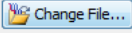


Figure 6.1.5

- Select the file row for the 5CSXFC6D6ES
- Select, **Change File** 
- **If you compiled the design** Select the soc_system.sof, as shown in the .\output_files directory
OR
Select the soc_system.sof, as shown in the .\ precompiled_files_for_system_console_module

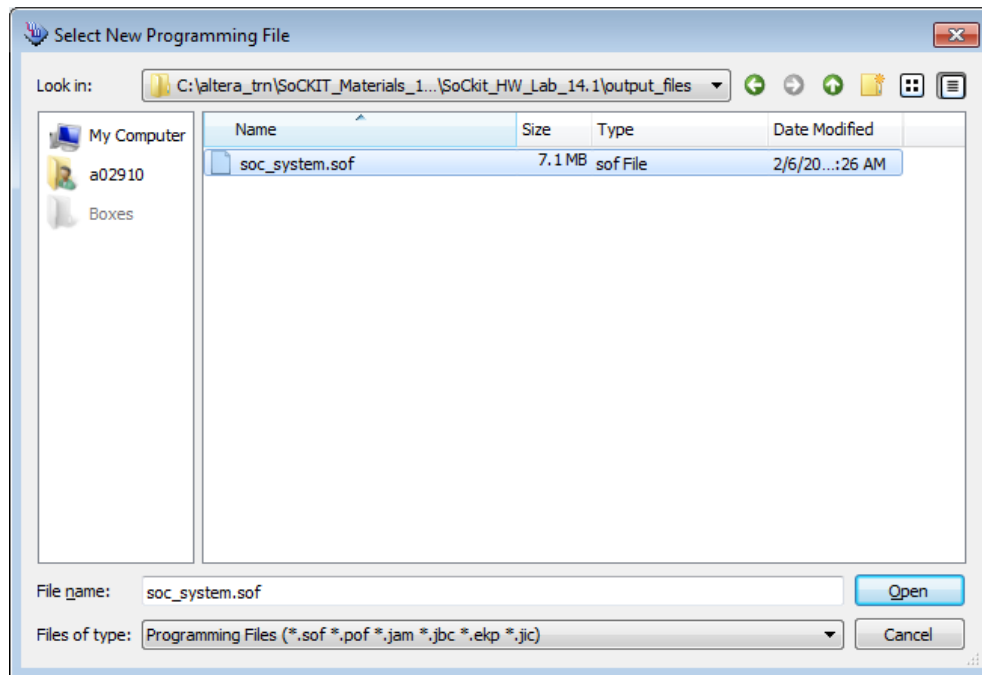


Figure 6.1.5

- 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 5CSXFC6D6ES device**):

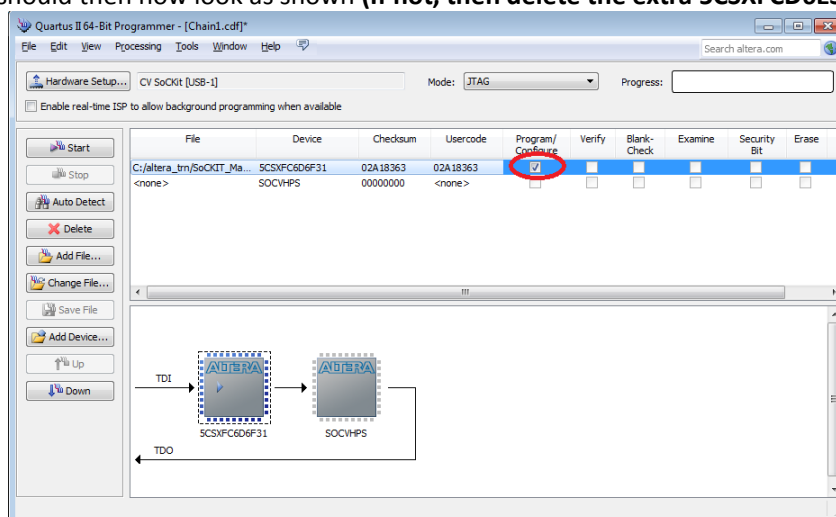


Figure 6.1.6

Press the **Start Button** as shown in Figure 6.1.6 to program the FPGA.

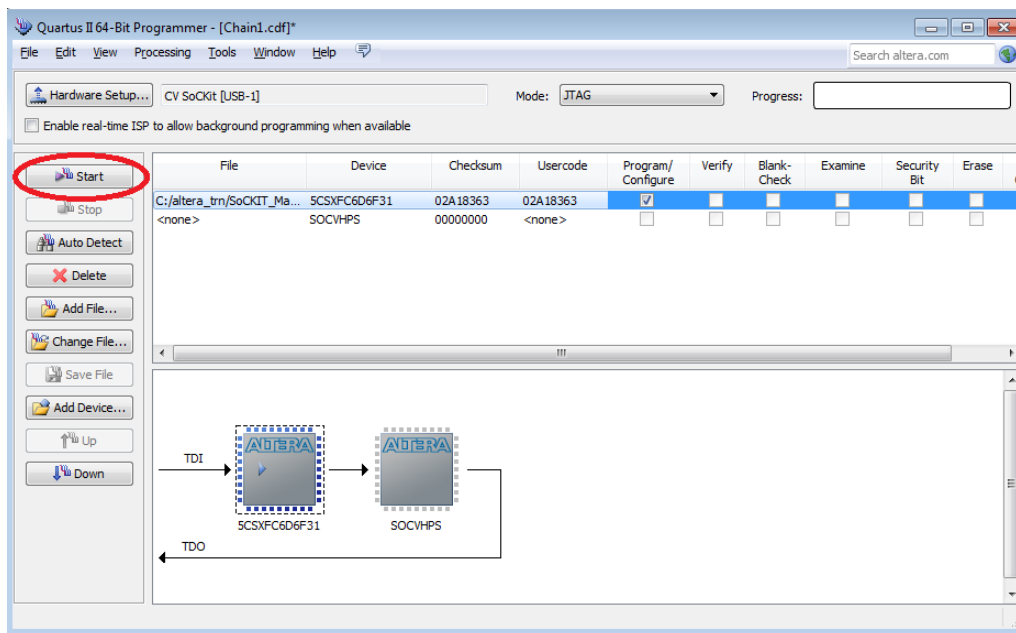


Figure 6.1.6

After programming the FPGA the progress indicator should indicate **100% complete** as shown in Figure 6.1.7. There should be no error messages.

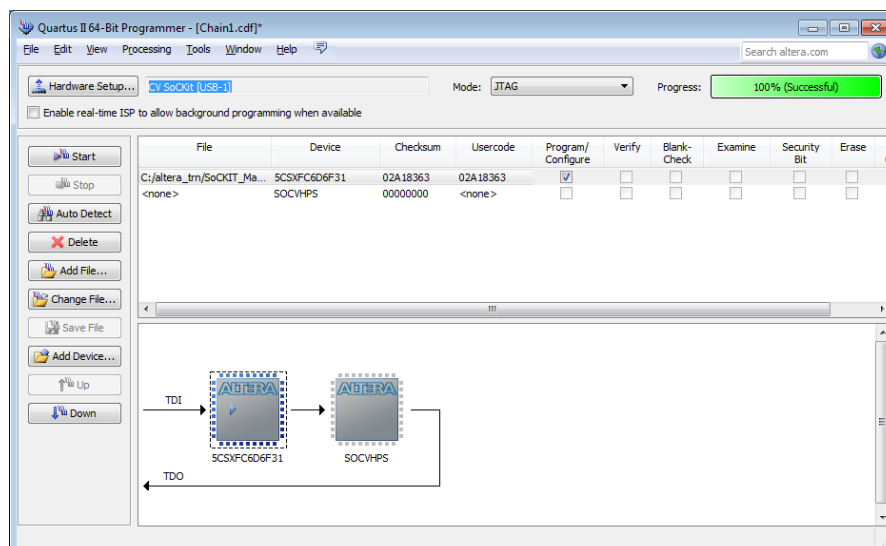


Figure 6.1.7

Select **File -> Exit** to close the Programmer window.

Select the “**No**” button as shown in Figure 6.1.8 to complete closing the Programmer window.

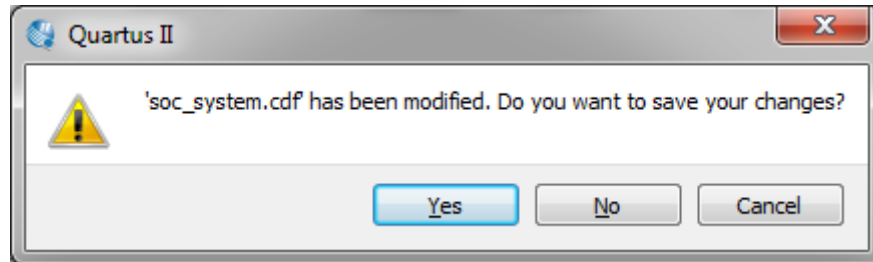


Figure 6.1.8

6.2 Executing System Console Scripts

There are different ways to debug a design. Qsys was utilized to build the system and a JTAG to Avalon Master Bridge was added to the design. As a result, debug 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.

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
 - Qsys 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](#)

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 IP validation will be completed in the following steps utilizing 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.

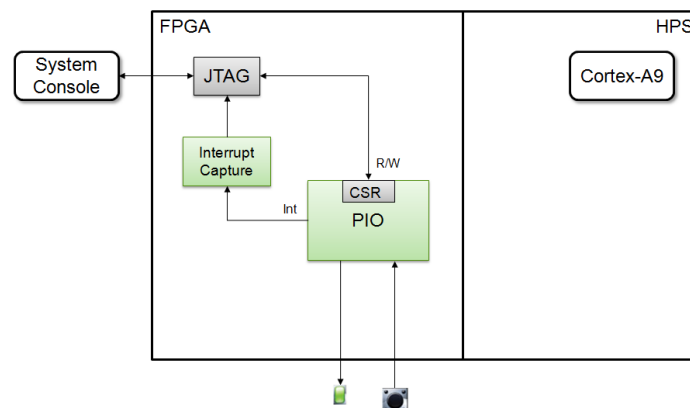


Figure 6.2.1

- 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:

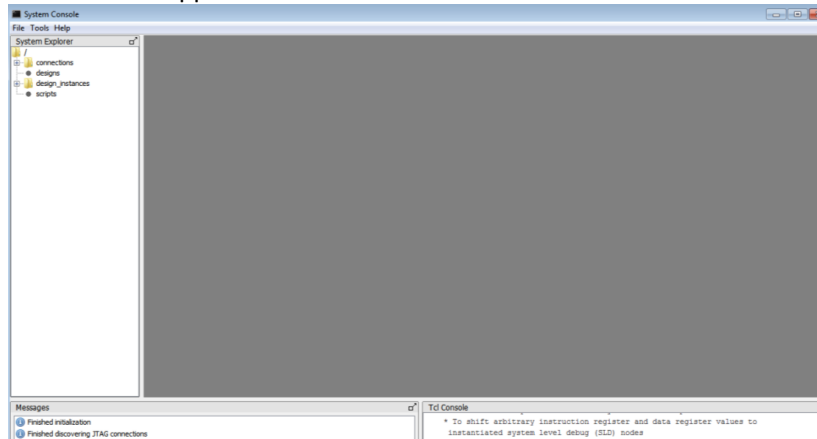


Figure 6.2.2

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:

- To run the script, select: **File -> Execute Script**.
- There will be a pop up window that appears that looks as shown in Figure 6.2.3
- Scroll to the correct path where you un-archived the lab files, and select **test_one.tcl**.
- **Do not** select Open yet.

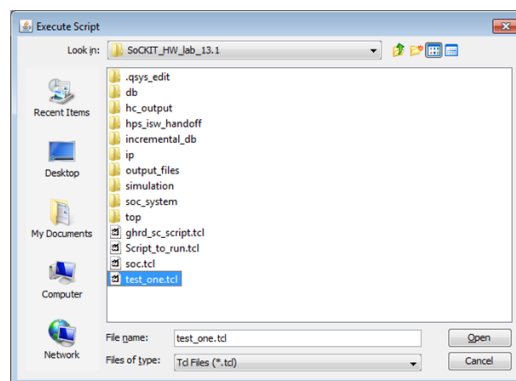


Figure 6.2.3

- 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.



Figure 6.2.4

- 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)

Table 6.2.1

- 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_14.1" 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] 1]
% open_service master $jtag_master
```

- master_write_8 \$jtag_master 0x10040 \$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 0x10040 4
```

- This command writes a 4 (0100) to the address of the FPGAs LEDs (0x0001_0040). 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
- When you have finished close system console:
Select: **File -> Exit**

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](#), the External Memory Interface Toolkit and [the ADC Toolkit for MAX10](#).

Congratulations!

You have just completed using System Console.

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

Module Objective:

In this module you will simulate the LED_PIO hardware in the FPGA 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 LED PIO (Parallel Input Output) component in the system.

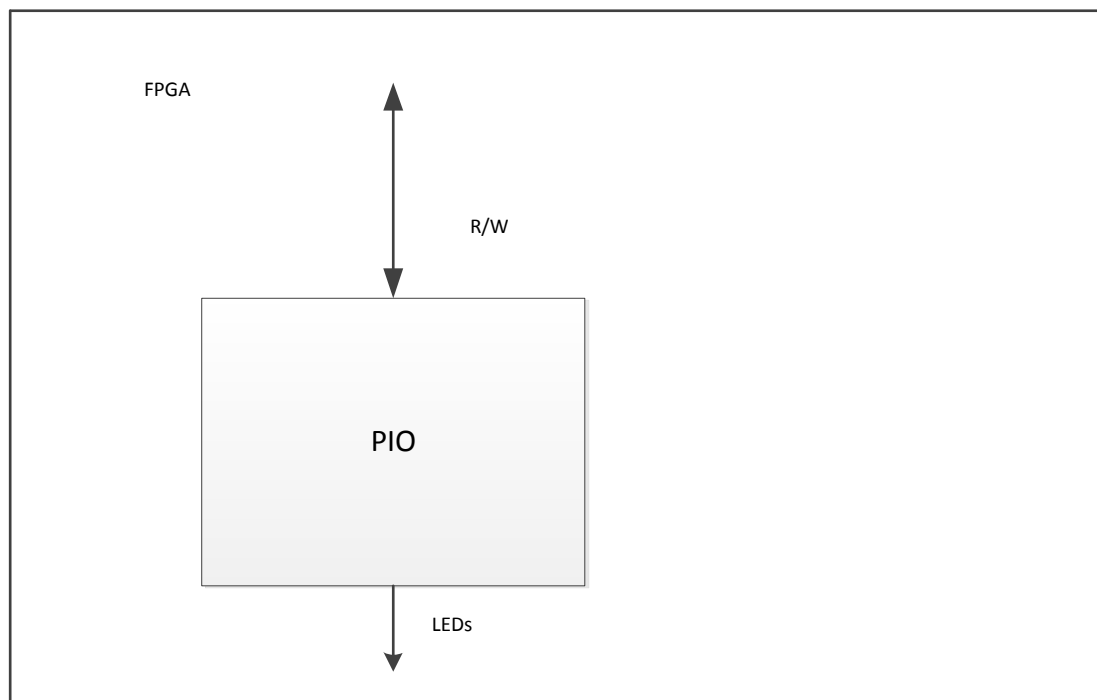


Figure 7.1.1

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

7.1 Installing ModelSim-Altera (Complete 7.1 only if you didn't install ModelSIM in Module 1)

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:** 10.1e for Quartus II v14.1
5. Select: **Download**

Download Center

Home > Support > Downloads > Download Center

Latest Release: Quartus II Version **14.1**

[Which version of the Quartus II software supports my device?](#)



Quartus II Subscription Edition
Paid license required
The industry's #1 design software in performance and productivity.
Free 30 day trial

[Download](#)

Quartus II Web Edition
FREE, no license required
A FREE version of Quartus®II software for your CPLD and selected medium-density and low-cost FPGA devices.
IP available for purchase

[Download](#)

Related Links

What's New

[Compare Quartus II Web and Subscription Edition](#)

[Compare ModelSim-Altera and ModelSim-Altera Starter Edition](#)

[University Software](#)





 <p>ModelSim-Altera software Altera's version of ModelSim software.</p>	 <p>Altera SDK for OpenCL The FPGA industry's first Software Development Kit (SDK) for OpenCL.</p>
 <p>SoC Embedded Design Suite Comprehensive and powerful software development suite for your ARM-based SoC FPGA embedded design.</p>	 <p>DSP Builder Helps to shorten digital signal processing (DSP) design cycles.</p>

Figure 7.1.2

6. Select: **Individual Files**
7. Select: **Download Selected Files** to download **ModelSIM-Altera Edition (includes Starter Edition)**

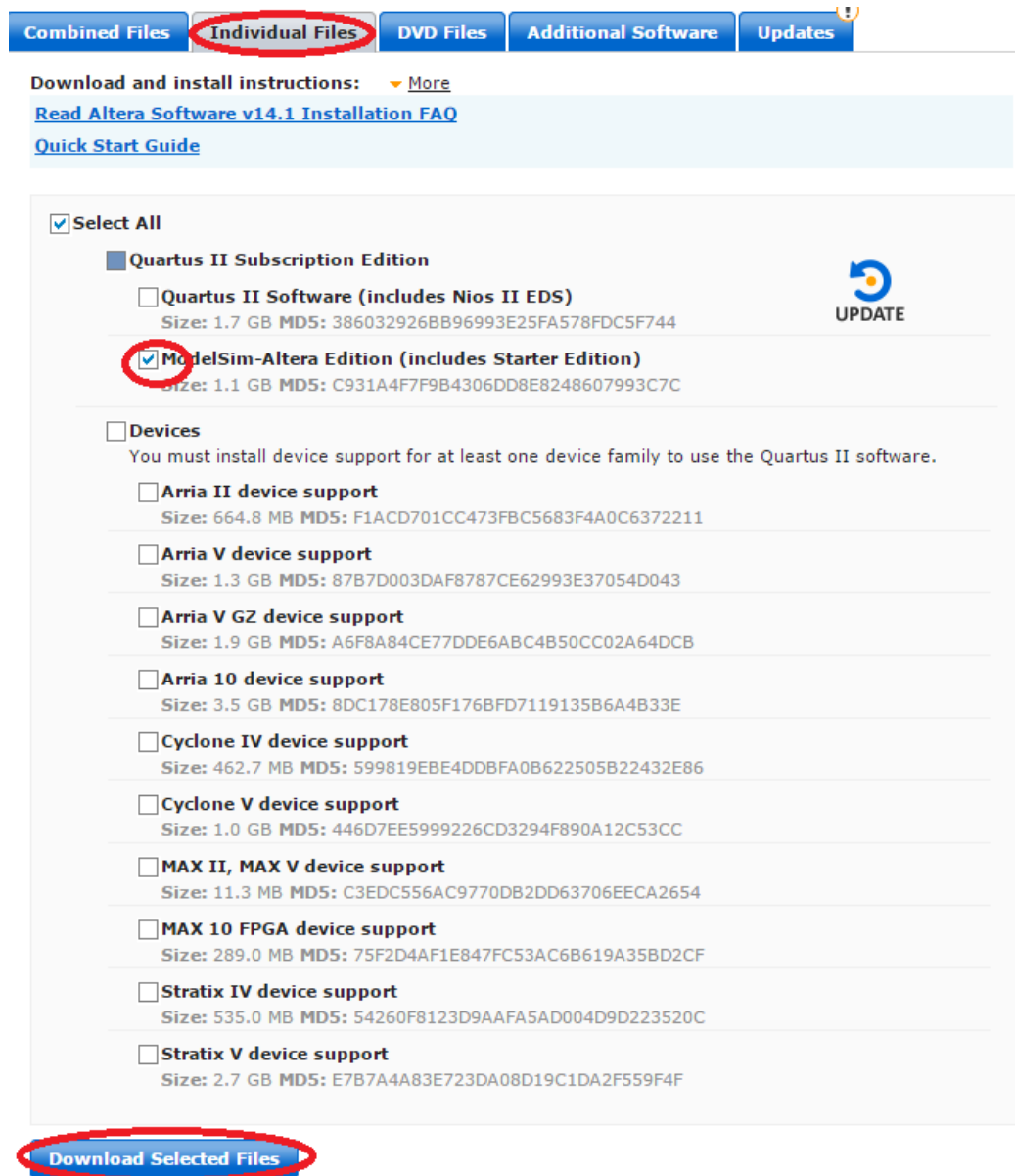


Figure 7.1.3

8. Go to the download directory and run the executable: **ModelSimSetup-14.1.0.186-windows.exe**

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

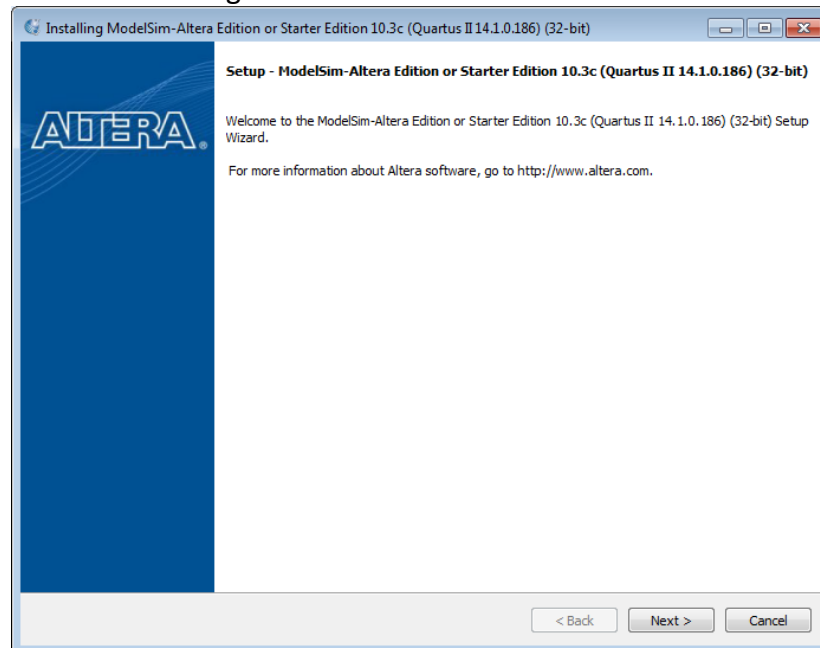


Figure 7.1.4

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

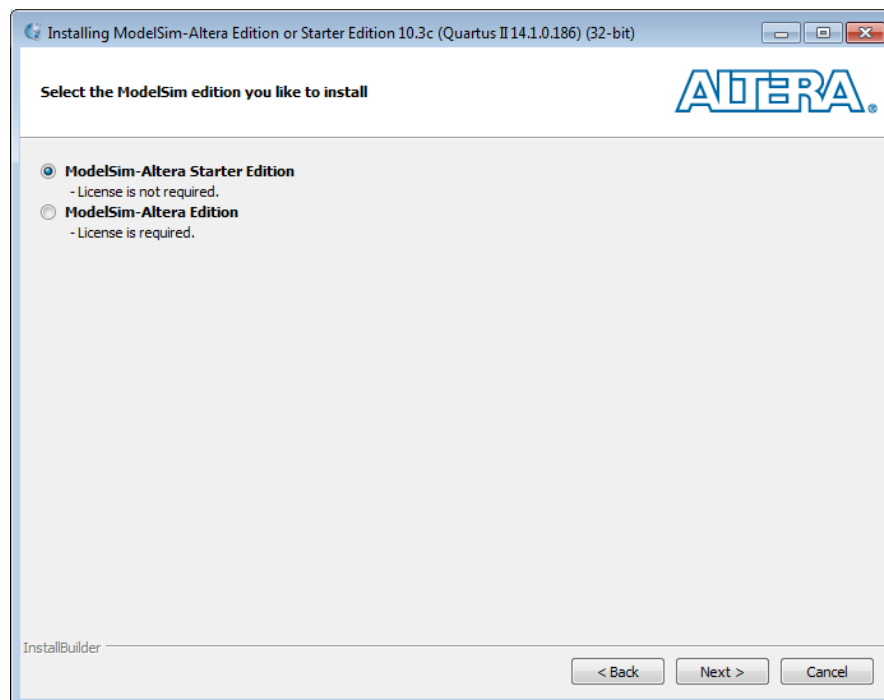


Figure 7.1.5

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

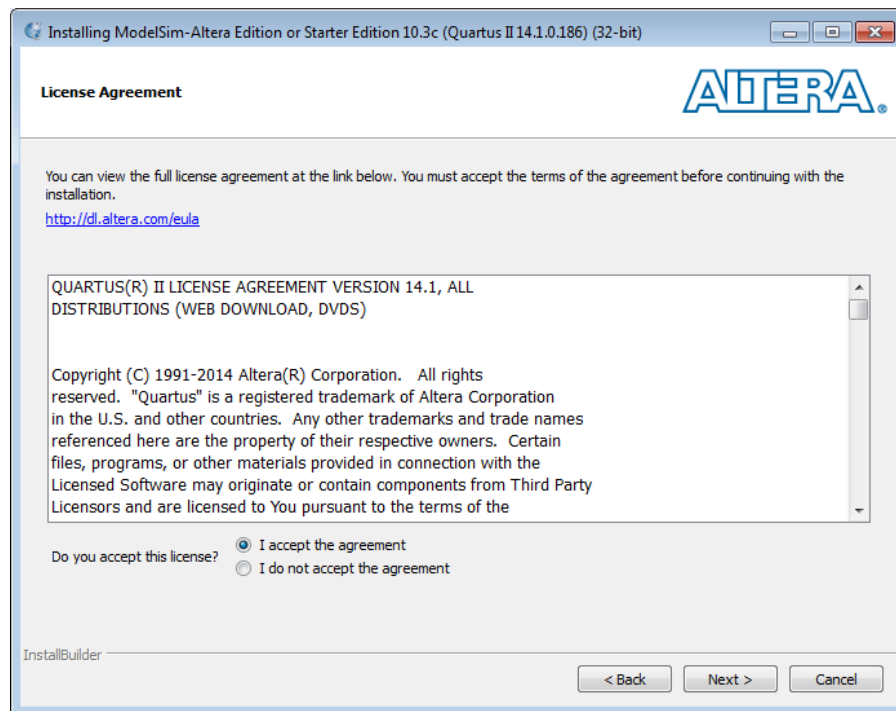


Figure 7.1.6

12. Navigate to the installation directory and select **Next >**

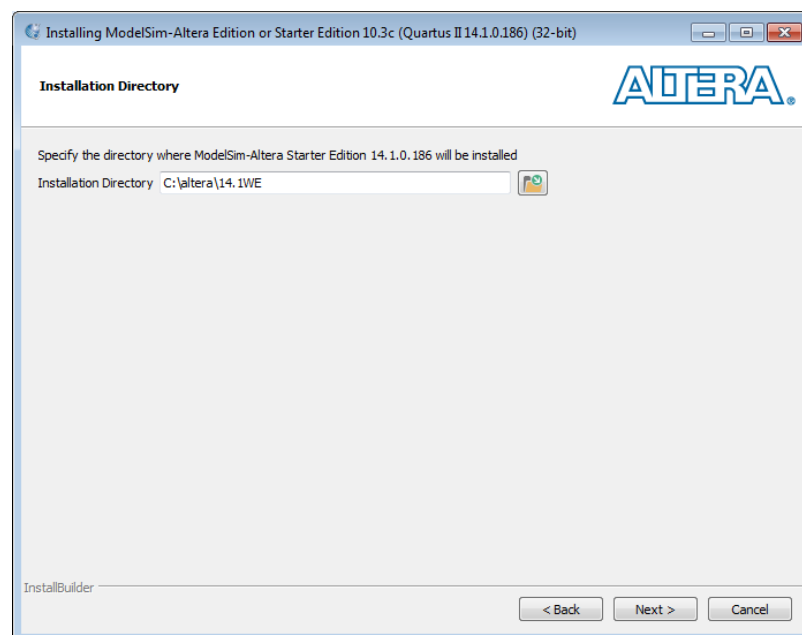




Figure 7.1.7

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

Open Quartus II, Select  -> All Programs -> Altera 14.1.0.186 Web Edition -> Quartus II Web Edition 14.1.0.186 -> Quartus II 14.1 (64)

1. If your design isn't open, then to Open your design, select: **File -> Open Project** and select:
C:\altera_trn\SoCKit\SoCKit_HW_lab_14.1\soc_system.qpf
2. Setup Nativelink for ModelSim-Altera in Quartus II. Select: **Tools -> Options**
3. Under the **Category: General**: select: **EDA Tool Options**
4. Next, select the browse button  and browse to your ModelSIM Installation Directory and select the path to win32aloem

C:\altera\14.1WE\modelsim_ase\win32aloem

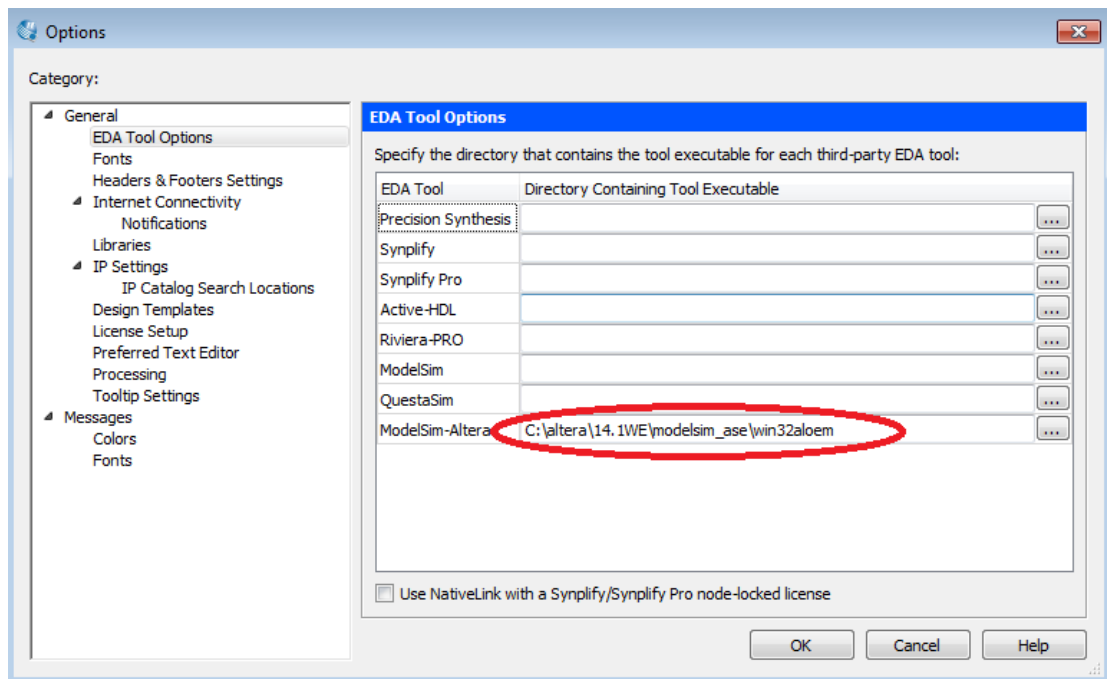
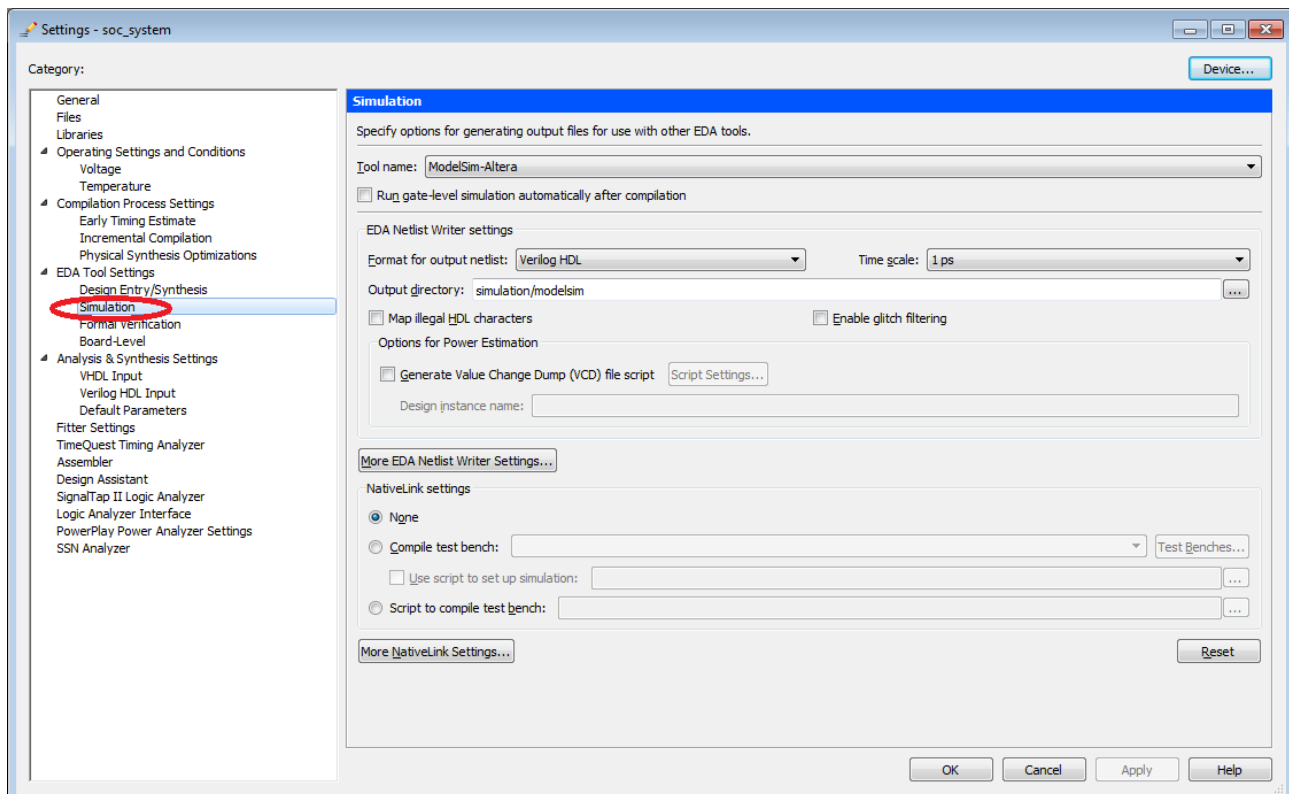


Figure 7.2.1

5. Select **OK**.
6. In Quartus II, select: **Assignments -> Settings**
7. 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.

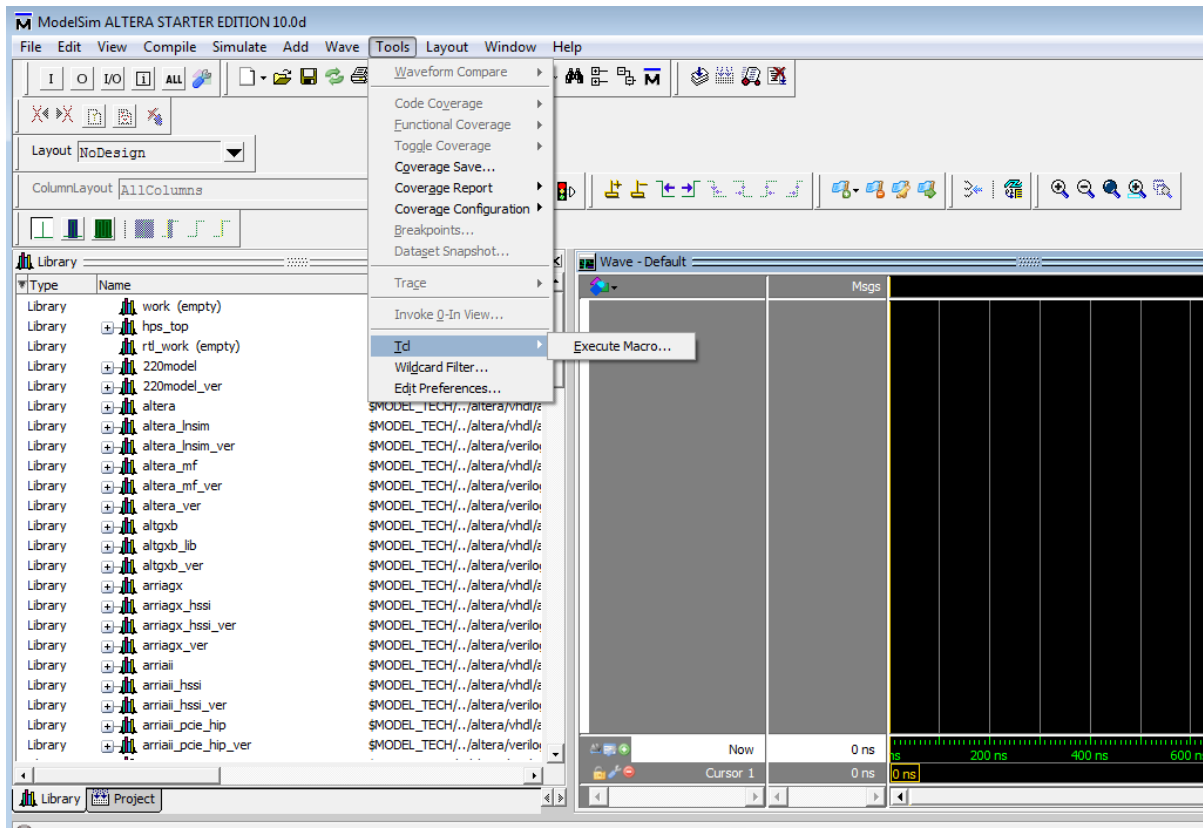
Please disregard the errors in Modelsim

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.

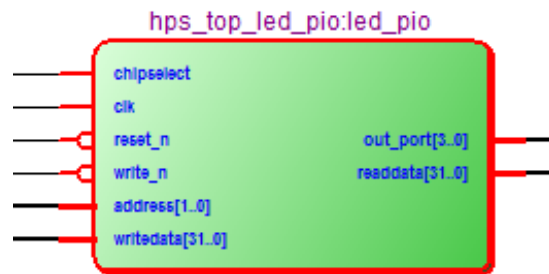
1. To run the script in ModelSIM select: Tools -> Tcl -> Execute Macro



2. Select: C:\altera_trn\SoCKit\SoCKit_HW_lab_14.0\Script_to_run.tcl.

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 resources available to assist you in further product development at www.altera.com/embedded

Some of the resources available are:

Get more information about Qsys and SOC with online training:

http://www.altera.com/education/training/curriculum/embedded_hw/trn-embedded_hw.html

http://www.altera.com/education/training/curriculum/embedded_sw/trn-embedded_sw.html

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

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