CAPSTONE PROJECT GUIDE

MODULE III

FPGA CAPSTONE PROJECT MODULE III: ALTERA MAX10 NIOS II HARDWARE

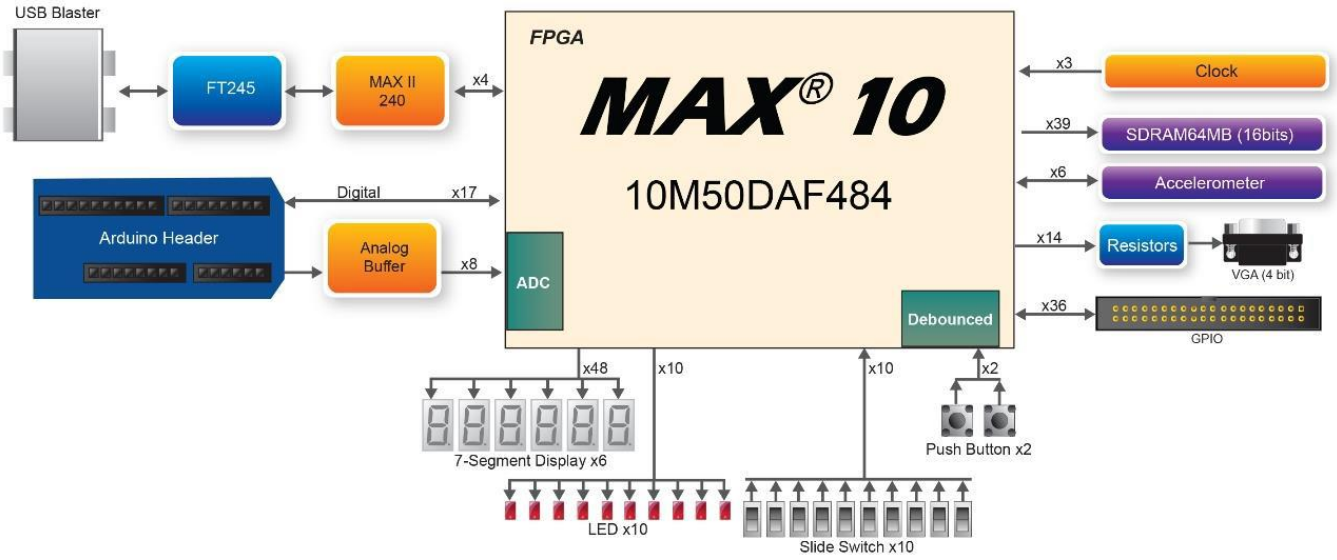
TABLE OF CONTENTS

INTRODUCTION

The DE10-Lite is a FPGA evaluation kit that is designed to get you started with using an FPGA. The DE10-Lite adopts Altera's non-volatile MAX® 10 FPGA built on 55-nm flash process. MAX 10 FPGAs

enhance non-volatile integration by delivering advanced processing capabilities in a low-cost, instant-on, small form factor programmable logic device. The devices also include full-featured FPGA capabilities such as digital signal processing, analog functionality, Nios II embedded processor support, and memory controllers.

The DE10-Lite includes a variety of peripherals connected to the FPGA device, such as 8MB SDRAM, accelerometer, digital-to-analog converter (DAC), temperature sensor, thermal resistor, photo resistor, LEDs, pushbuttons and several different options for expansion connectivity.



PROJECT LEARNING OBJECTIVES

For this project, the objective is for students to:

- Become familiar with the FGPA development flow, particularly in the case of a SoC with software development flow included.
- Appreciate the capability of the MAX10 to create whole systems on a chip.
- Learn how to build systems using the Qsys (Platform Designer) system design tool.
- Learn to create hardware using schematic capture input.
- Learn how to integrate software with hardware in the same device.
- Understand the rationale for each phase of the hardware development flow, including timing constraints, simulation, and programming.
- Design and build a several hardware examples using the MAX10.
- Consider hardware and software tradeoff possibilities available in the SoC architecture.

Do not be afraid to ask questions in the discussion forums as you work on this project. It involves many processes and actions that may seem complicated and confusing at first, but will become clearer as you work through the modules.

The modules will get progressively more difficult and take more time. This is Module 3.

MODULE GOAL

The goal of this module is to develop the hardware for a System on a Chip (SoC). You will construct hardware that creates a NIOS II soft processor along with several interfaces to devices on the DE10-Lite development kit. In this module you will

- Create a working design, using most aspects of the Quartus Prime Design Flow.
- Create hardware for the NIOS II soft processor, including many interfaces, using Qsys (Platform Builder). Instantiate this design into a top-level DE10-Lite HDL file.
- Compile your completed hardware using Quartus Prime.
- Record all your observations in a lab notebook.
- Submit your project files and lab notebook for grading

Completing this module will help prepare you for the work to be done in the module that follows.

I. GENERAL PROCEDURE

Caution:

Do not continue until you have read the following:

The names that this document directs you to choose for files, components, and other objects in this exercise **must be spelled *exactly* as directed.** This nomenclature is necessary because the pre-written software application includes variables that use the names of the hardware peripherals. Naming the components differently can cause the software application to fail. There are also other similar dependencies within the project that require you to enter the correct names.

CREATING A SYSTEM ON CHIP WITH THE NIOS II SOFT PROCESSOR

1. If you have not already done this, download and install Quartus Prime FPGA development software from Intel Altera. Follow the directions in the installation video in Course 1 of the specialization if you need help. Install version 16.1.
2. Follow the instructions in the detailed design section that follows for the NIOS II embedded system. Be sure to record your observations as you go.
    a. If your results look different than the project guide, do not be alarmed, as there may be some differences between the tools used in the guide and your particular installation of the tools.
    b. Do not be surprised when the initial and even subsequent compiles of the FPGA have errors. This will point you to the additional work you need to do.
3. This section is based on completing the output portion of the design of a simple voltmeter.
    a. In Section 1 you will prepare for the project acquiring files and other resources.
    b. In Section 2 you will examine the system design.
    c. In Section 3 you will learn how to use Create a NIOS II system design using Quartus Prime. The amount of work may appear to be daunting, but once you start into you should find that it proceeds fairly quickly.
    d. In Section 4 you will instantiate the Qsys design into the top-level Verilog file.
    e. In Section 5 you will place and route the design.
    f. In Section 6 you will create a programming file that could be used to configure the FPGA fabric.
4. Take your DE10-lite evaluation board and plug the USB cable into a computer. Program the FPGA with your NIOS II embedded system circuit design. Record your observations in your lab notebook - Describe how the device behaves?
5. Record the Fmax for this lab in your lab notebook. If an Fmax is not listed, use the inverted highest clock frequency for this number.
6. Estimate the % utilization of the FPGA logic for this lab at completion.
   Submit a zipped up project file and lab notebook for grading.

II. DETAILED DESIGN

CREATING A NIOS II SOFT PROCESSOR

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:

- Altera Quartus Prime Version 16.1 FPGA Development Software Tool, including the NIOS II Embedded Design Suite (EDS)
- **Module Design Files: Embed.zip**
- Terasic DE10-Lite Development Kit

Before continuing with this Module, ensure that the Altera tools and drivers have been installed.

EXTRACT THE LAB FILES.

- Create a folder **C:\AlteraPrj\DE10liteEmbed** on your PC.
- Copy Embed.zip to this directory
- Extract here to the above directory

Good Work!!

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

2. SYSTEM DESIGN

Section Objective

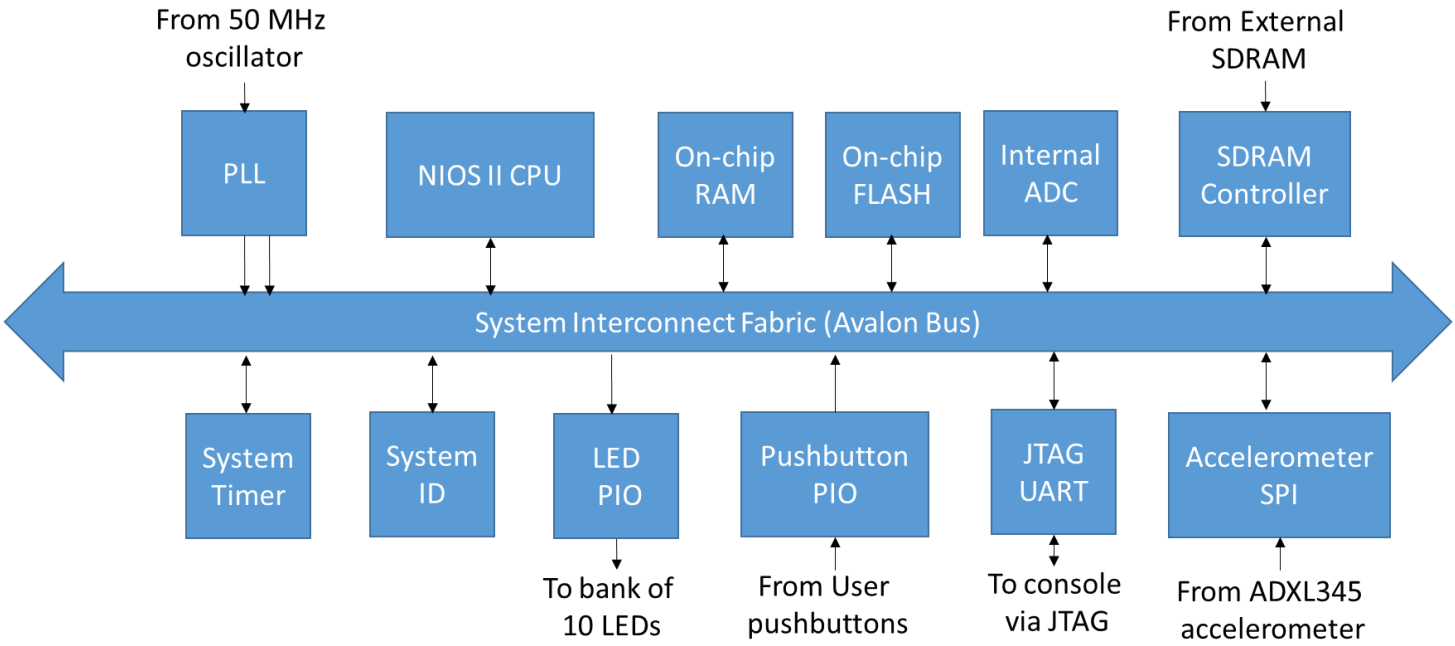
In this Section you will review the architecture of the design that will be created in Quartus Prime with the DE10-Lite Kit as the target. Typically, a design starts with system requirements. These system requirements become inputs to the system definition. System definition is then the first step for implementation in the design flow process.

SYSTEM REQUIREMENTS

During this exercise, you will follow a step-by-step guide to create a more complex design. To do this you will create a design in Qsys, the Altera System Design Tool, which includes a NIOS II softcore CPU,

On-chip RAM and FLASH memory for program and data storage, a 1 ms System Timer, System ID block, MAX10 ADC module, LED and slide switch interfaces, SDRAM controller, SPI bus interface to an accelerometer, and JTAG connections to the software development and FPGA configuration tools, all connected together by a Avalon Bus or Bus Bridge. You will then create a block symbol of this Qsys system, and create software to run on it. The inputs are a System Clock at 50 MHz, a 10 MHz ADC clock, 10 slide switch inputs, SPI data from the accelerometer, and the ADC input from a pin on the Arduino Analog Connector. The outputs are 10 LEDs, and the JTAG UART Debug Console. Here is a block diagram of the system you will construct:

2.1 SYSTEM BLOCK DIAGRAM



The system above can be created in Qsys using a standard library of re-useable IP blocks. The System Interconnect Fabric is automatically generated by Qsys and binds the blocks together. The system interconnect manages dynamic bus-width matching, interrupt priorities, arbitration and address mapping. This system is a full-featured processor system capable of running operating systems such as uC-OSII or Linux.

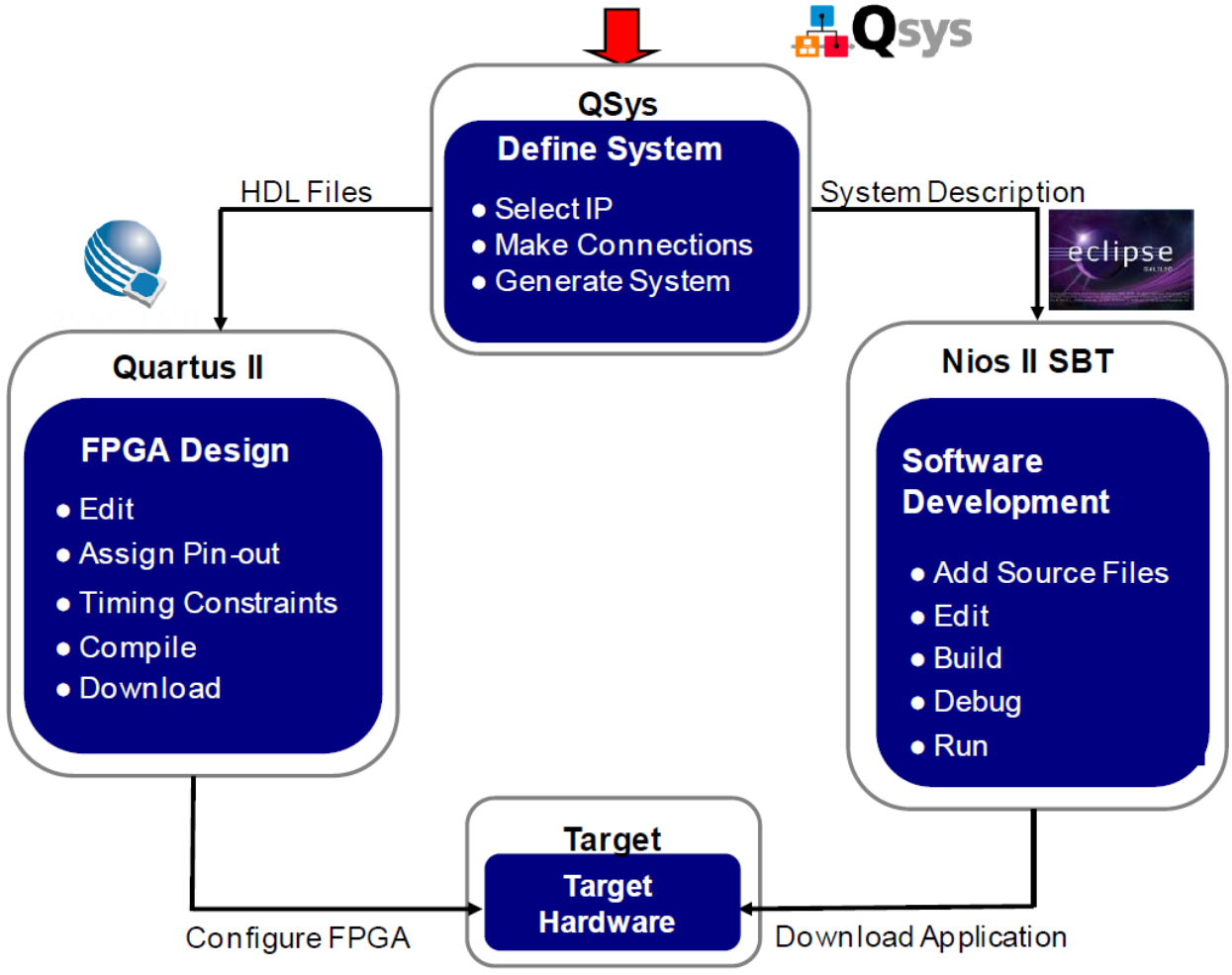
The following pages will guide you through the process of building this basic embedded system. You will build up a subset of the system shown above.

Components of MAX10 Device Used

This project uses the MAX10 PLL block, ADC module, On-chip RAM, On-chip FLASH, JTAG interface and logic fabric.

SYSTEM TOOL FLOW

Here is a diagram of the new system tool flow, which will include both hardware and software development because of the inclusion of the processor:



The above diagram depicts the typical flow for system design. System definition is performed using Qsys. The results are two-fold:

- A system description that the Nios II Software Build Tools, the software development tool, uses to create a new project for the software application.
- HDL files for the system that are used by the Quartus Prime FPGA design software to compile and generate the hardware system.

CONGRATULATIONS!! **You have just completed the review of the system-level design**

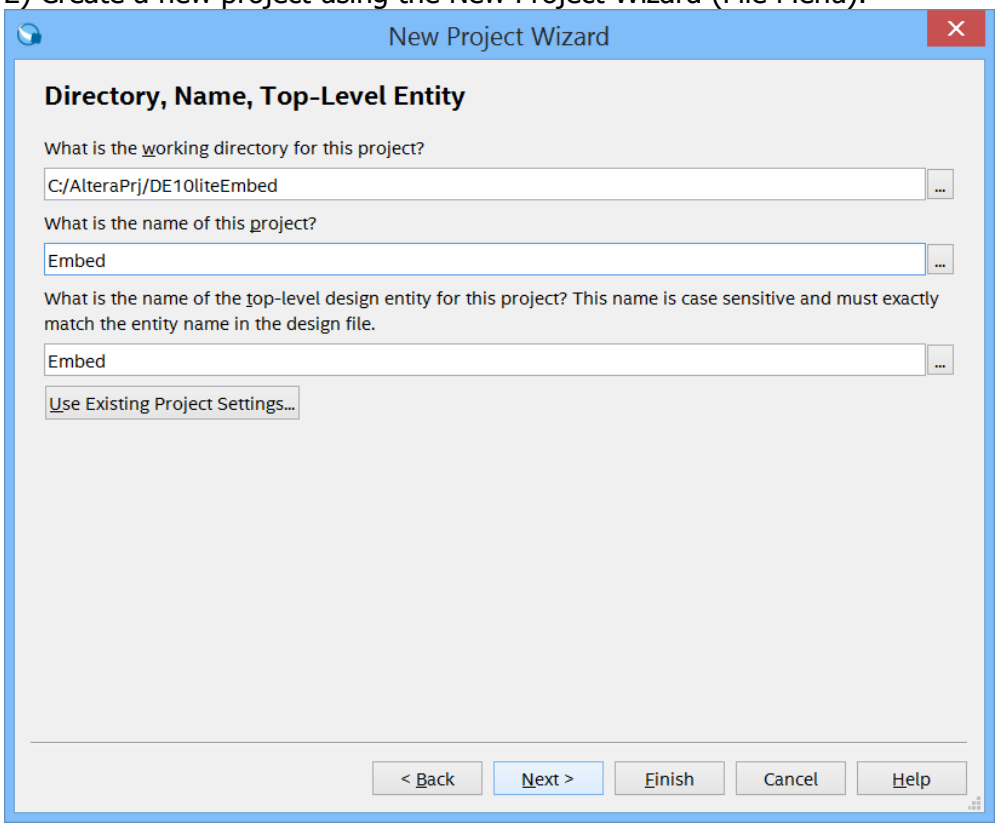
3. CREATING THE NIOS II PROCESSOR HARDWARE DESIGN

In this section will create the Embedded System design using Quartus Prime. Some source files have been provided in the project zip file, as the design is a combination of HDL files, IP cores, and Macro Blocks.

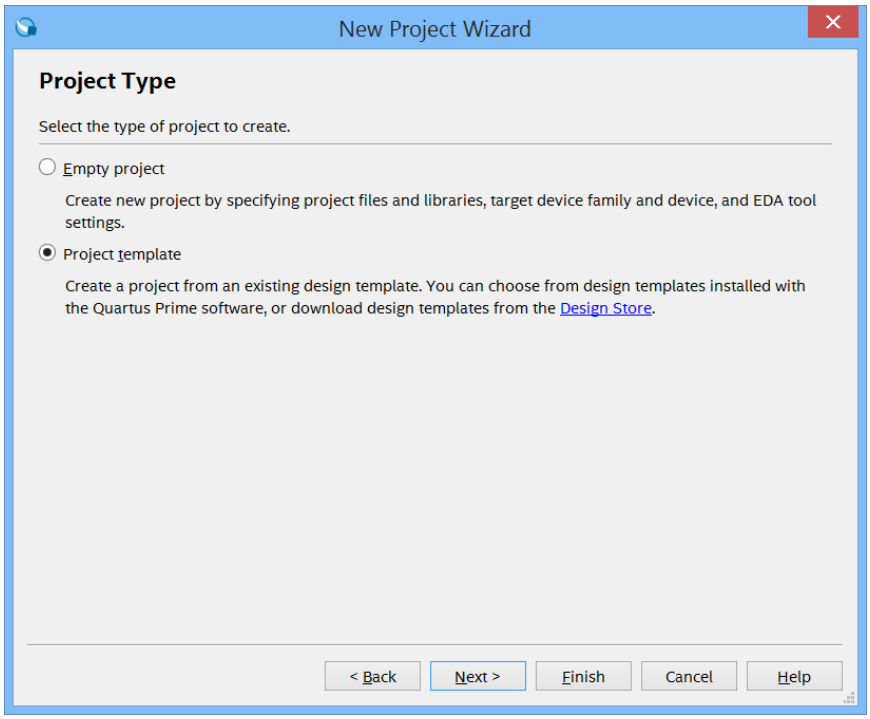
CREATE THE QUARTUS PROJECT

1) Launch Quartus Prime 16.1 (64-bit)

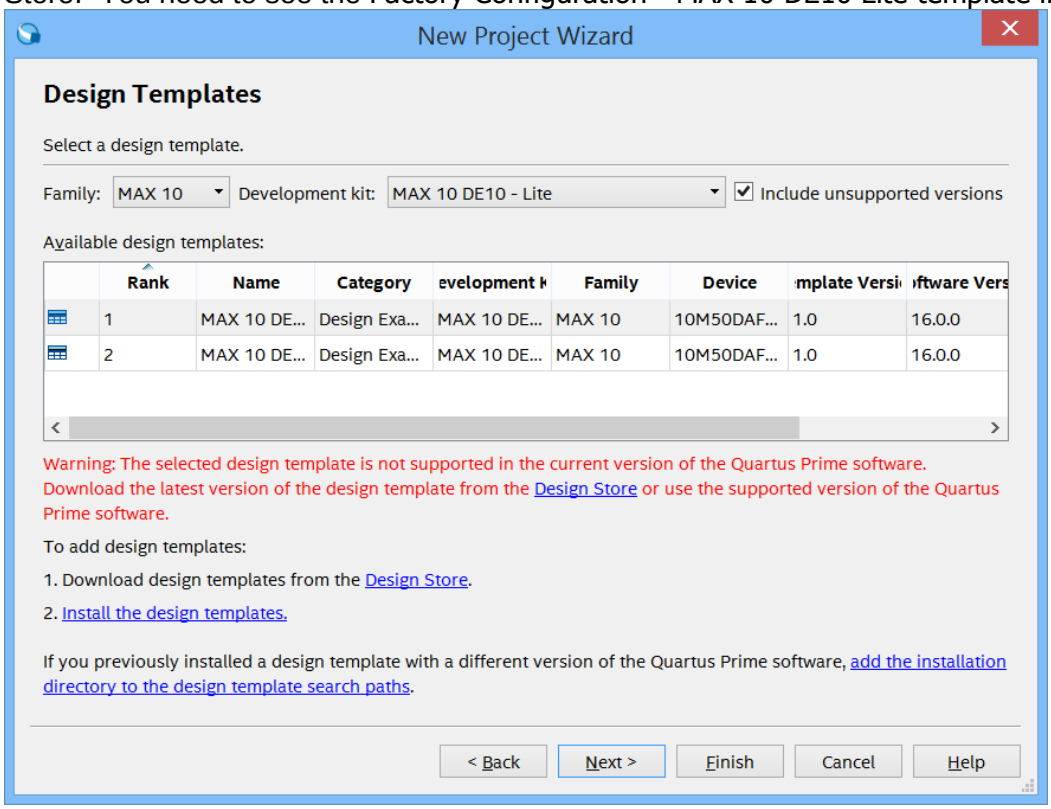
2) Create a new project using the New Project Wizard (File Menu).



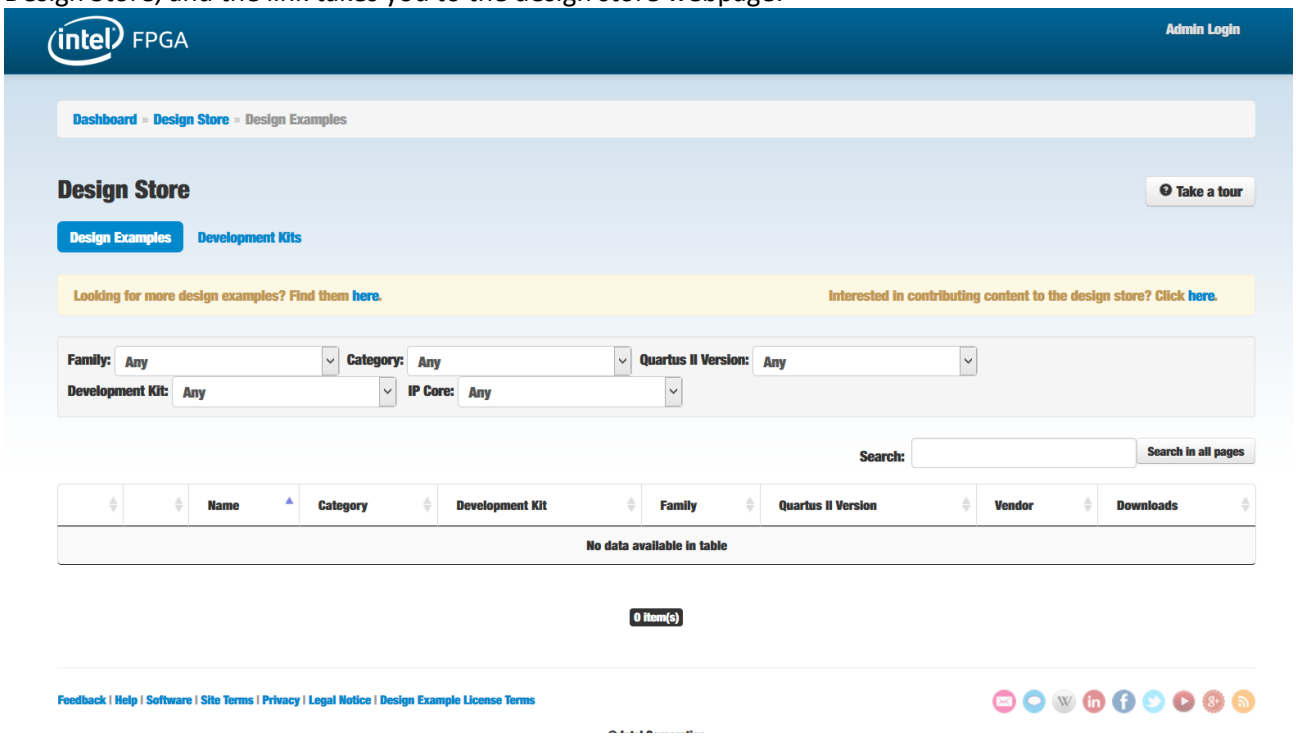
3) In the Project Type dialog box, choose the project template. Hit Next.



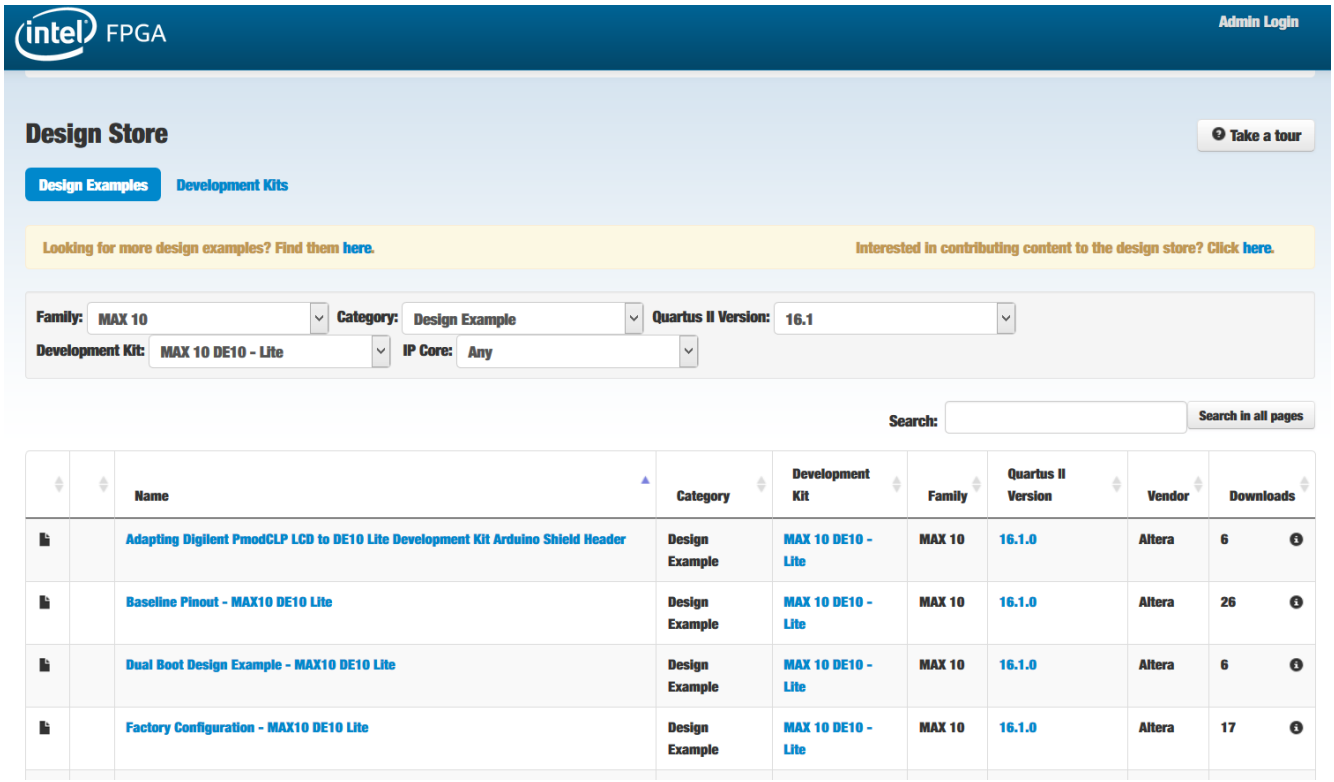
4) On the next box, select MAX10 for the family, and DE10-lite Evaluation Kit for the kit. If the DE-10 Lite kit is not listed in the choices, you may have to download the design templates from the Design Store. You need to see the Factory Configuration - MAX 10 DE10 Lite template in your list.



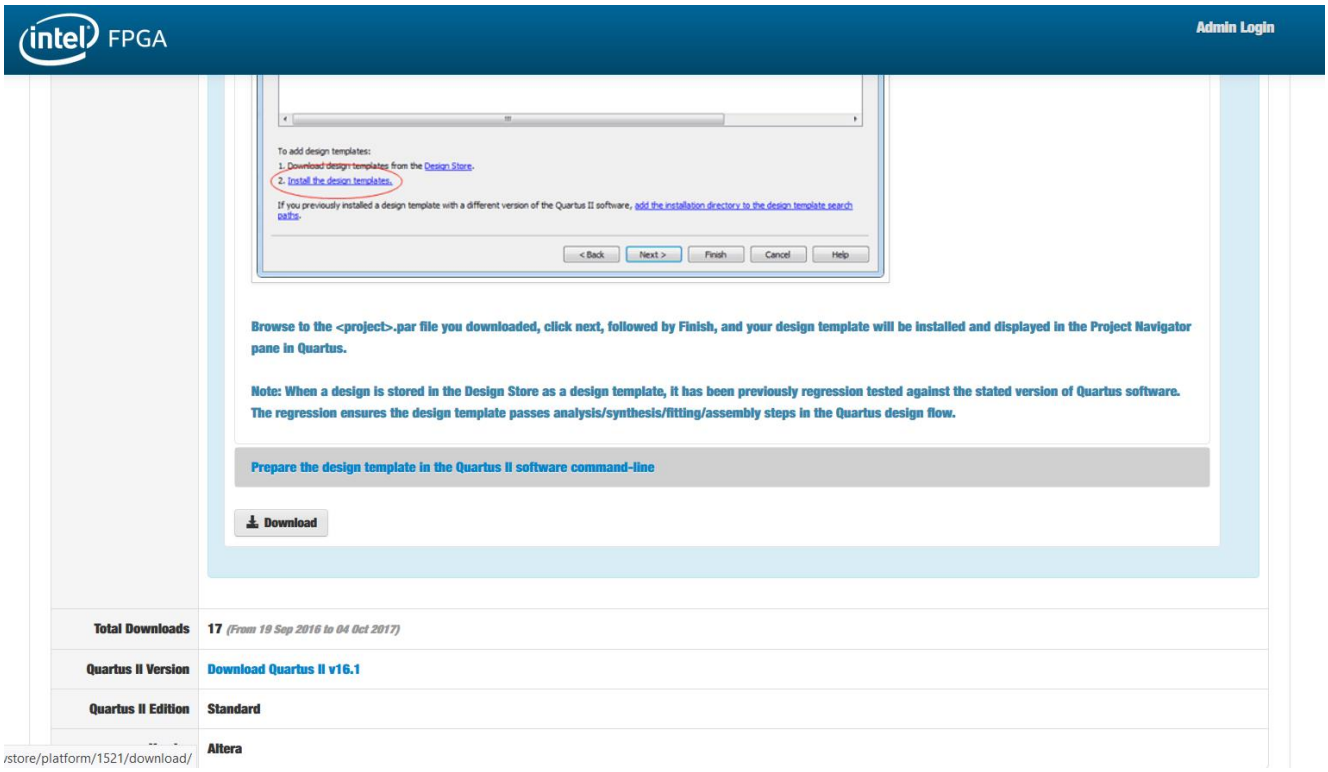
5) If you need to do this, a link is given in instruction #1 of this dialog box that allows you do this. Click on Design Store, and the link takes you to the design store webpage:



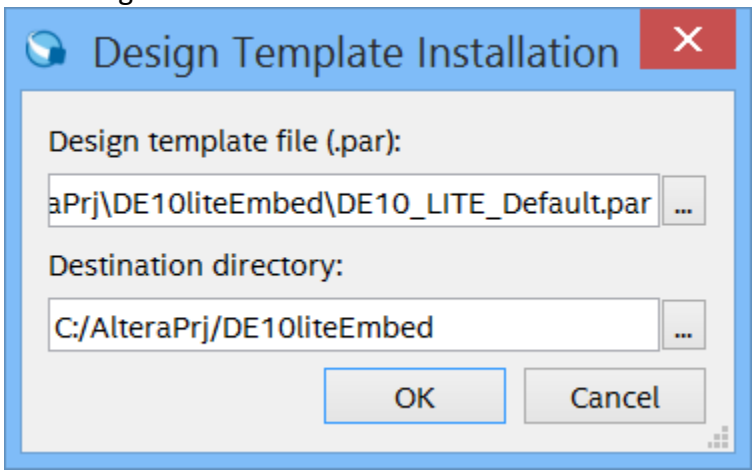
6) Be sure to select Design Examples. Select MAX10 for the family, Design Example for the Category and then the DE10-Lite Evaluation Kit for the kit:



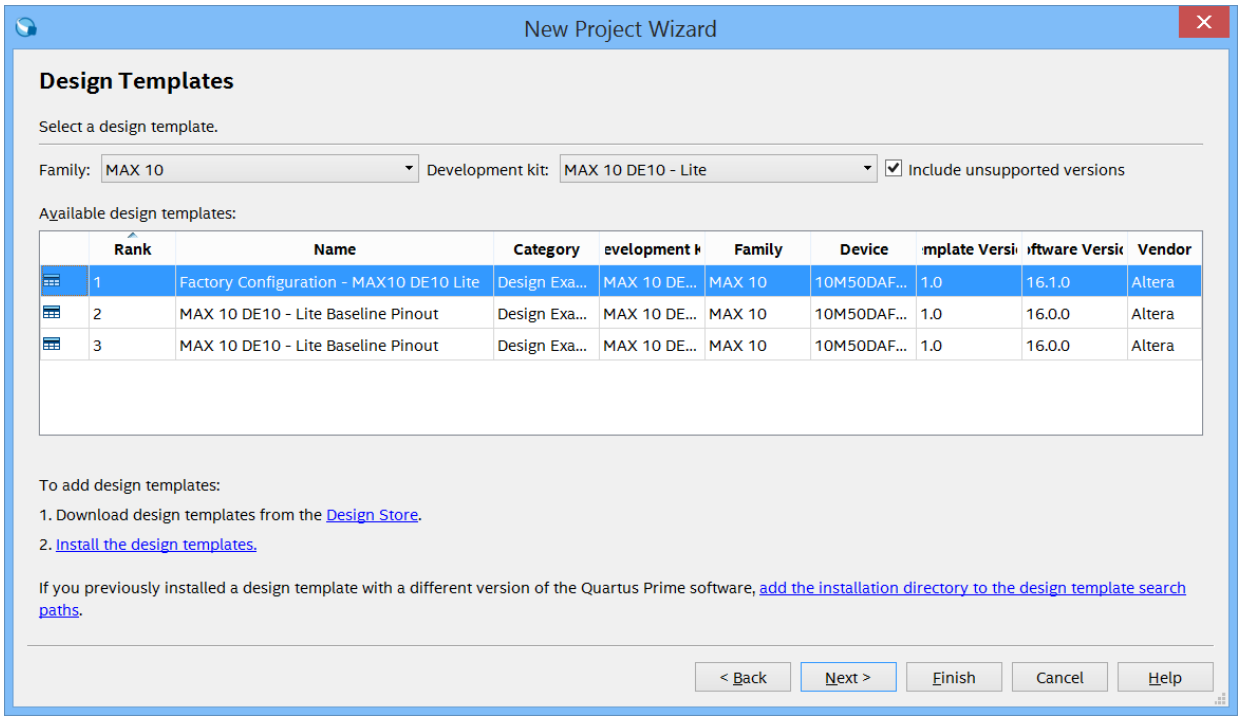
7) Select the Factory Configuration - MAX 10 DE10 Lite Quartus version 16.1. Download the installation package. Save the .par file in the project directory.



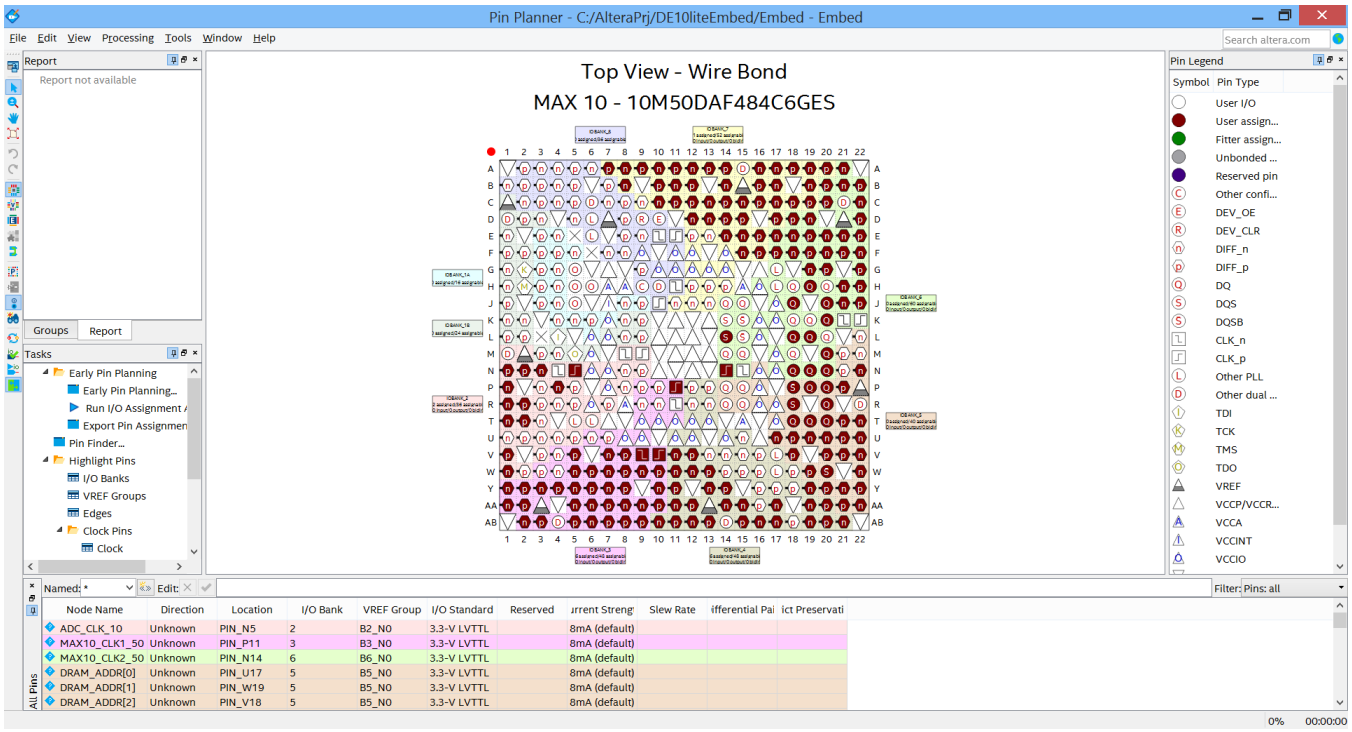
8) After downloading, return to Quartus where you will have to install the design template in the 2nd step as indicated. Click on instruction #2 install the design template, give the location of the .par file in the dialog box.



Click OK. Select the Factory Configuration MAX 10 DE10 - Lite Kit from the Design Template list and hit Next. Hit Finish in the Summary box.



9) When your project opens, Note that the top level is called DE10_Lite_Default. Under files in the project navigator you will see a top level Verilog file and several other Verilog files and .qip files. This is the basis for your design, it has all the external signals defined and pins assigned. If you open the Pin planner, you will see these signals present, and you can verify the pin assignments. These are defined in the .qdf file that comes with the template. The template provides you with a large amount of previously done work, allowing you to get a head start on your design.

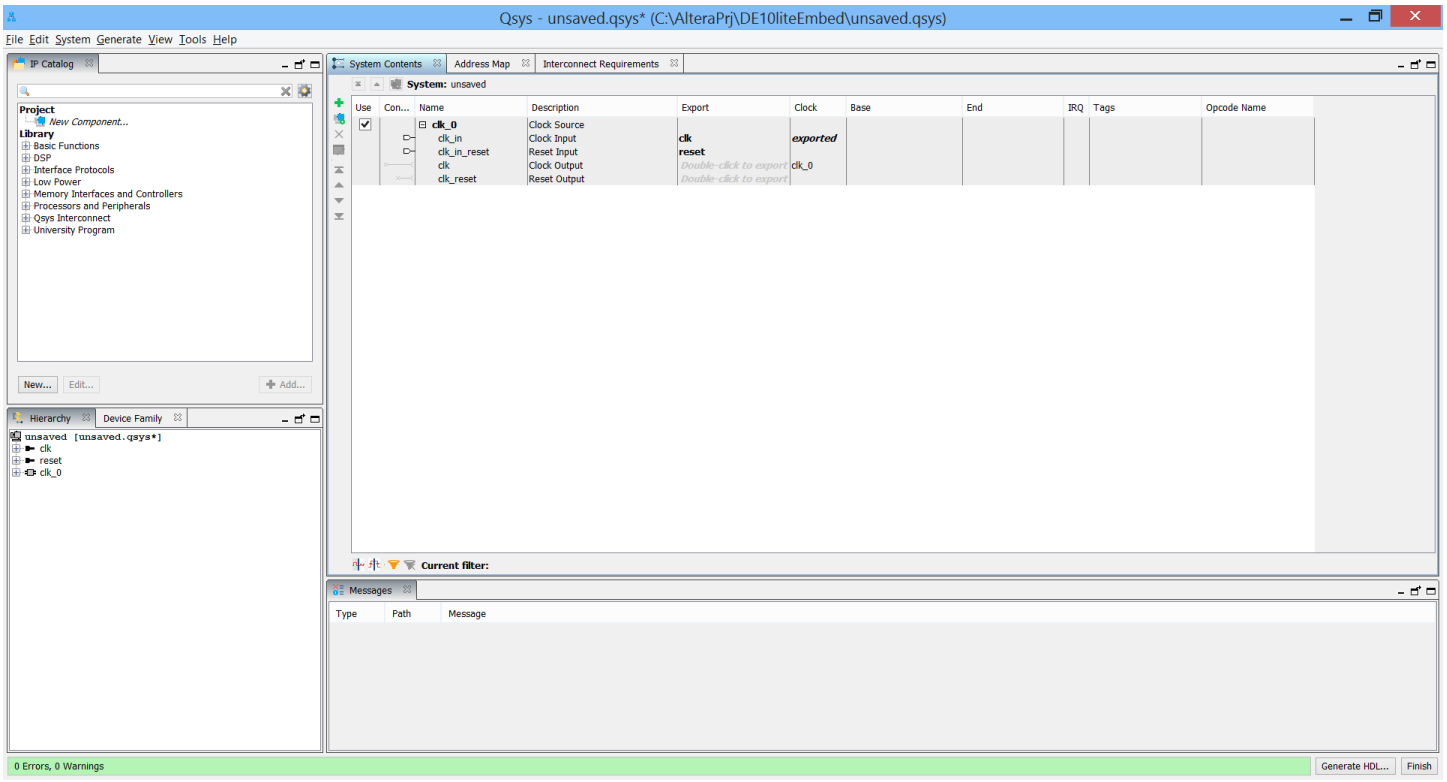


Close the Pin Planner.

We are now ready to create the Qsys system.

CREATE THE QSYS DESIGN

1) **From the Tools menu, select Qsys.** After a moment or two, the Qsys design tool will open. It begins assuming that you need a clock in the design.



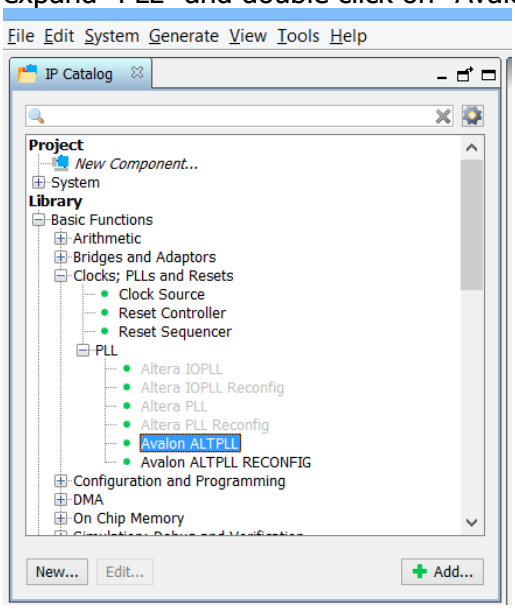
3.1 SYSTEM CLOCK SCHEME

There is a clock generator on the DE10-lite circuit board that generates two 50 MHz clocks and a 10 MHz clock, which serve as clock sources for the FPGA. Other clocks are also required for the Qsys system components as well as for external components such as the SDRAM and the internal ADC. PLLs will be used to provide these clocks. The following table reviews the clocking scheme:

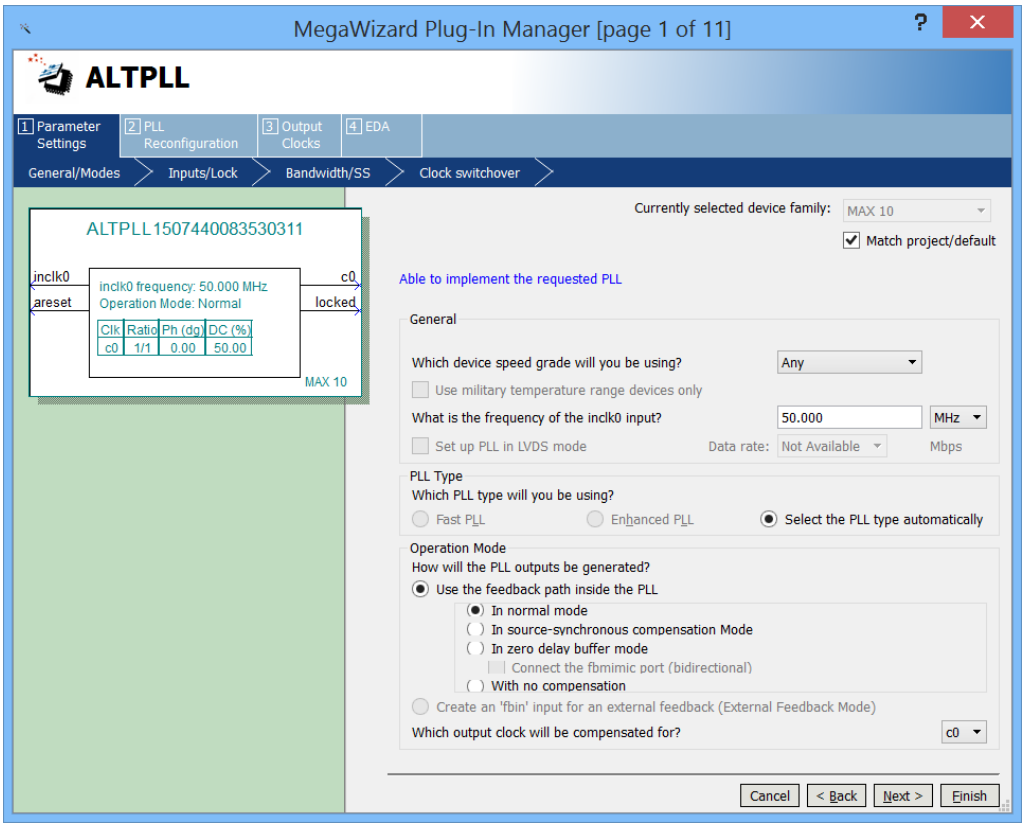
Component	Input Clock Frequency	Source	Designation
1. SDRAM PLL	50 MHz	Oscillator on DE10-Lite	MAX10_CLK1_50
2. Nios II processor and peripherals	80 MHz	Output 'c0' of PLL	sdram_pll_c0
3. SDRAM phase shifted clock	80 MHz (-90° phase)	Output 'c1' of PLL	sdram_pll_c1
4. Slow Peripherals	40 MHz	Output 'c2' of PLL	pll_c2
5. ADC PLL	10 MHz	Oscillator on DE10-Lite	ADC_CLK_10
6. internal ADC	10 MHz	Output 'c0' of ADC PLL	adc_pll_c0

2) **Add the system Avalon ALTPLL** as the clock source for Processor, Peripherals and Memory. This peripheral instantiates a PLL which will generate the clocks for the system.

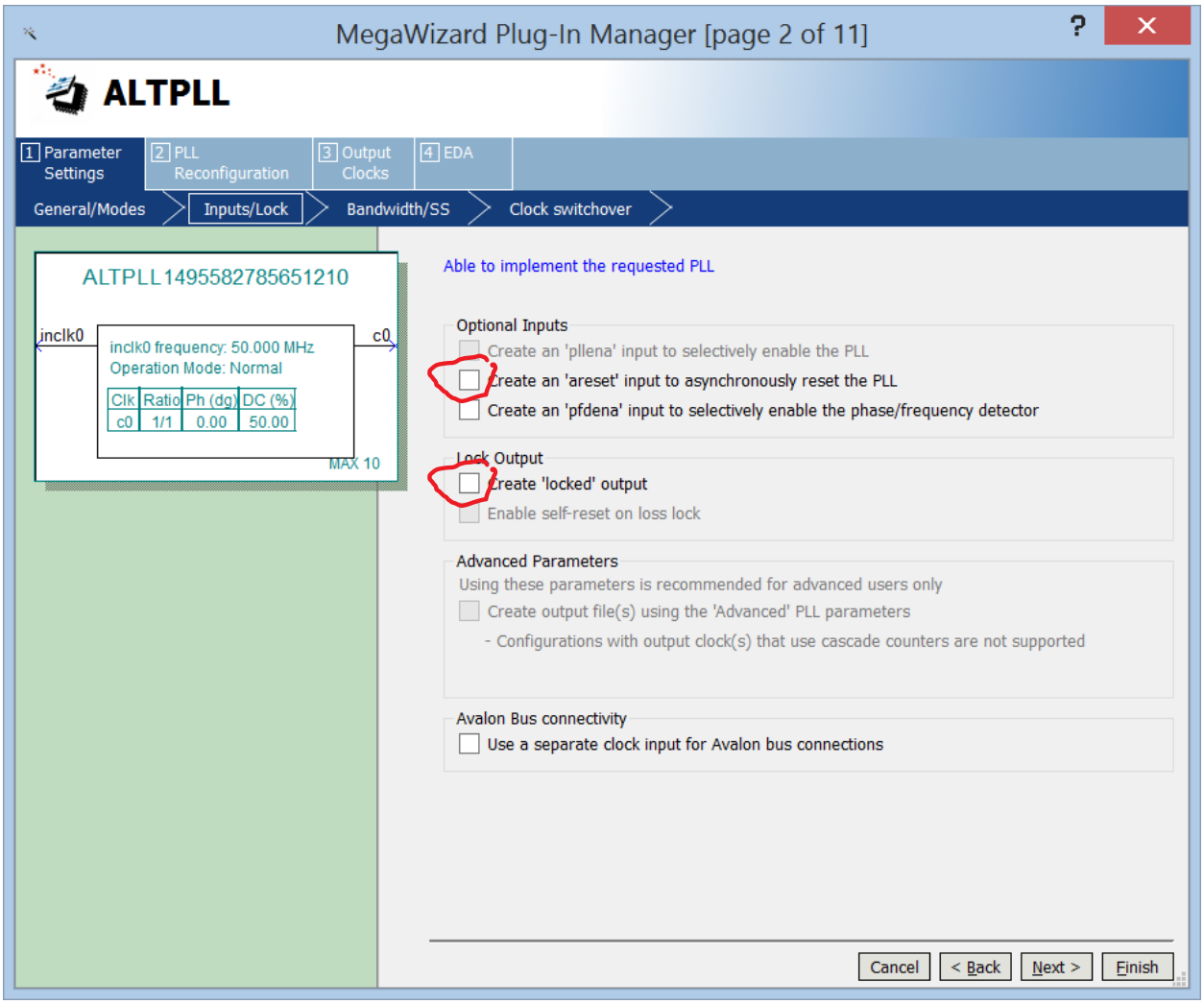
From the IP Catalog pane, expand "Basic Functions," then expand "Clocks; PLLs and Resets," then expand "PLL" and double click on "Avalon ALTPLL." This will add the Avalon ALTPLL module.



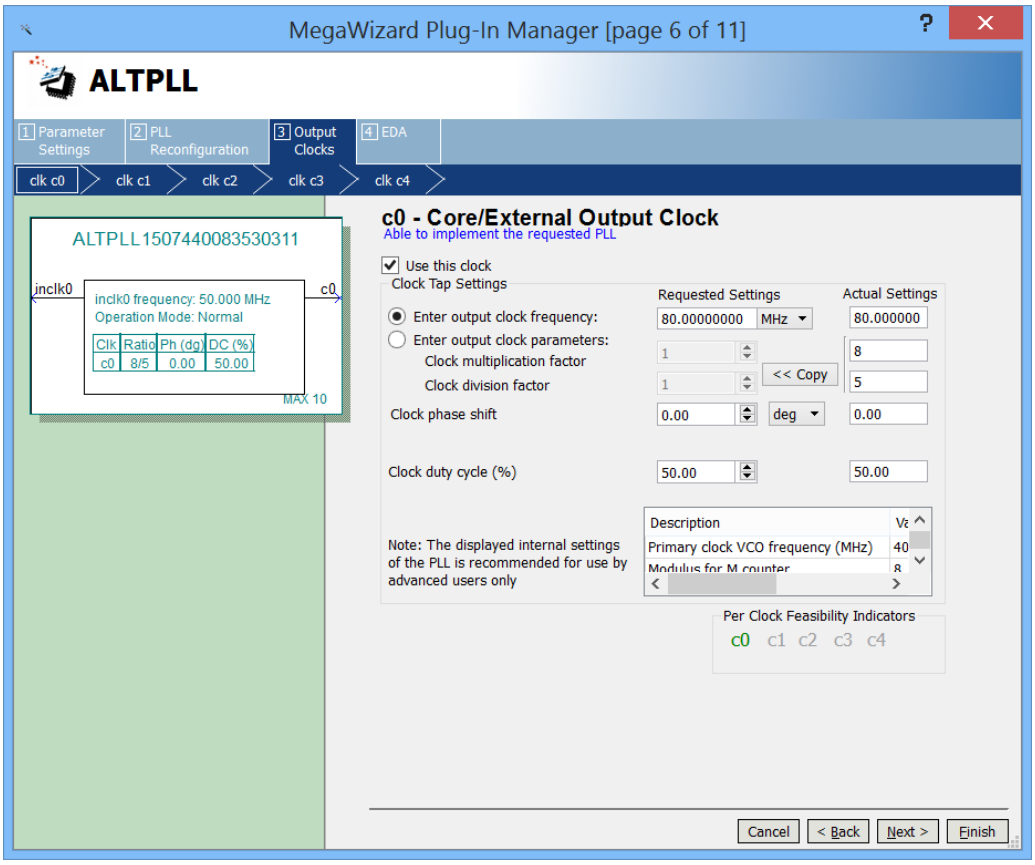
The ATLPLL Megawizard opens. Change the input clock frequency to 50 MHz, Click Next to move to the next tab of the wizard.



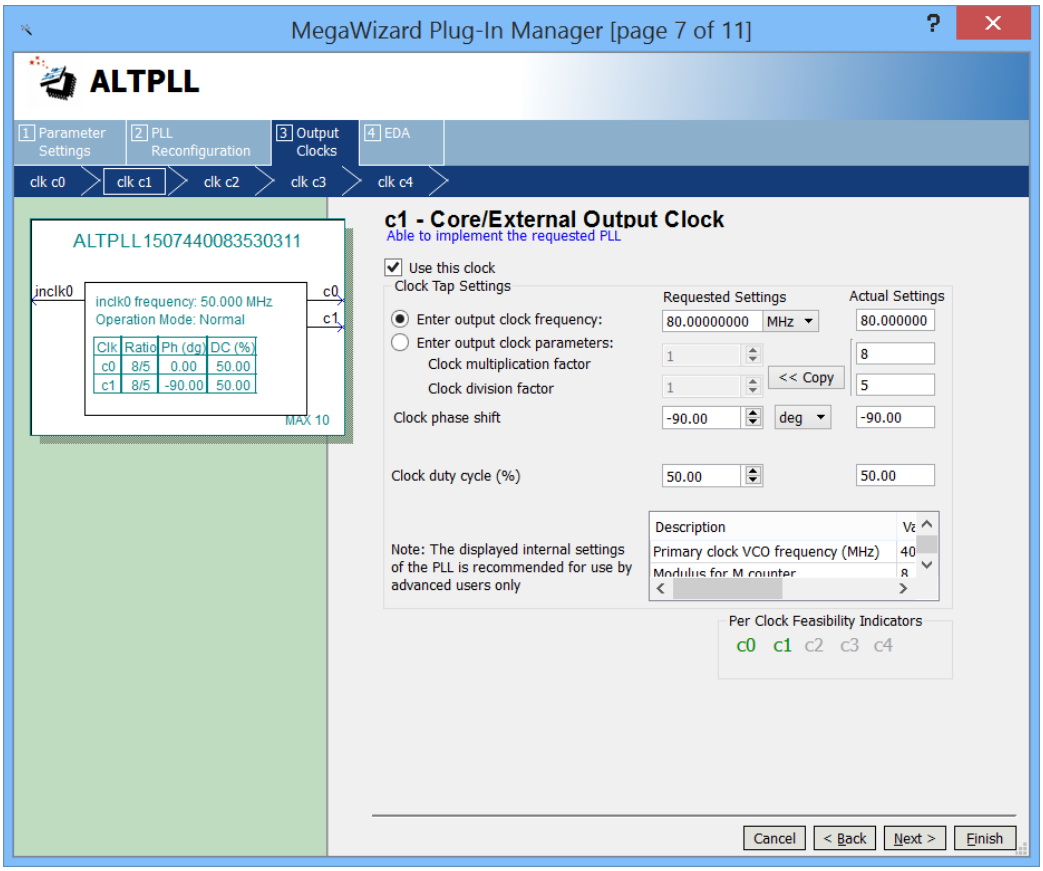
On the next page, Uncheck both "**Create an 'areset' input to asynchronously reset the PLL**" and "**Create 'locked' output**" options.



Click next, and also click next on the bandwidth page, the switchover page, and configuration page. You should now see the c0 Core external output clock. On **"c0 Core/External Output"**: Click on the "Enter output clock frequency" button and enter **80 MHz**. This clock will be used as the processor system clock, clocking the Nios II processor. Hit next.

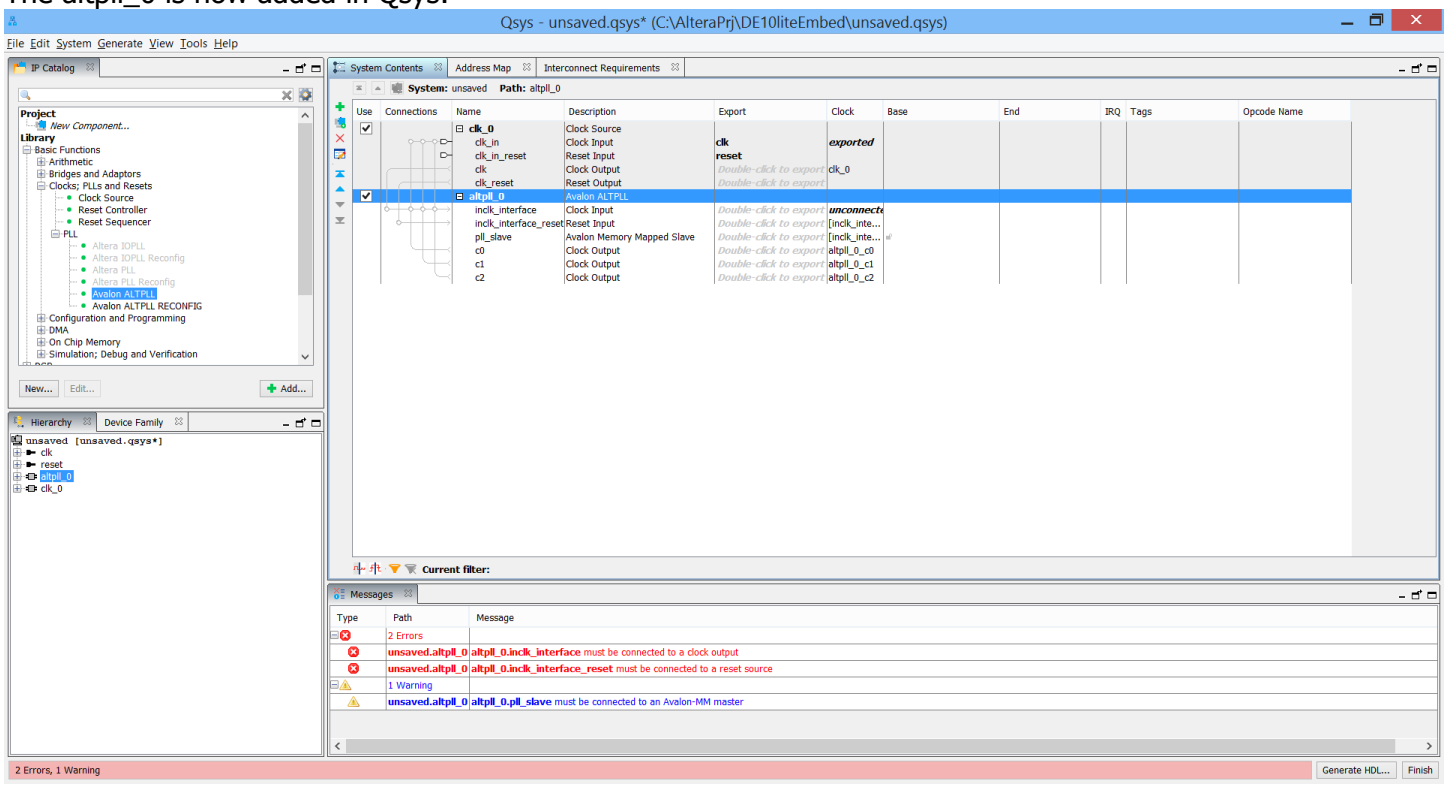


Check the "Use this clock" button for c1. Click on the "Enter output clock frequency" button and enter **80 MHz**. This clock will be used to clock the external SDRAM. In the **Clock phase shift** enter **-90** and leave the units set to **deg**. Click Next.



Check the "Use this clock" button for c2. Click on the "Enter output clock frequency" button and enter **40 MHz**.
This clock will be used to clock various peripherals in the system. Click next for c3 and c4 and then click Finish.

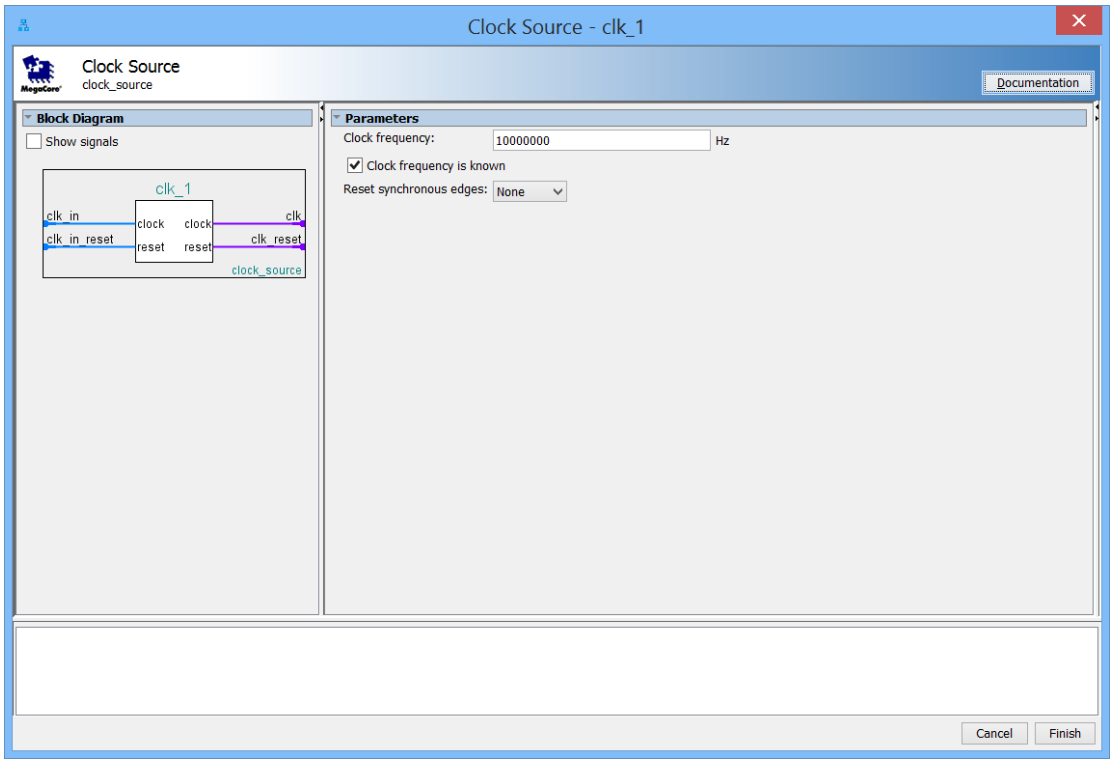
The altpll_0 is now added in Qsys:



Some errors and warnings will appear in the bottom console indicating that various ports are not connected. Ignore these for now. We will address these connections in the upcoming steps. You will continue to see these messages after every component is added, but not to worry as they are good reminders of connections that are yet to be made.

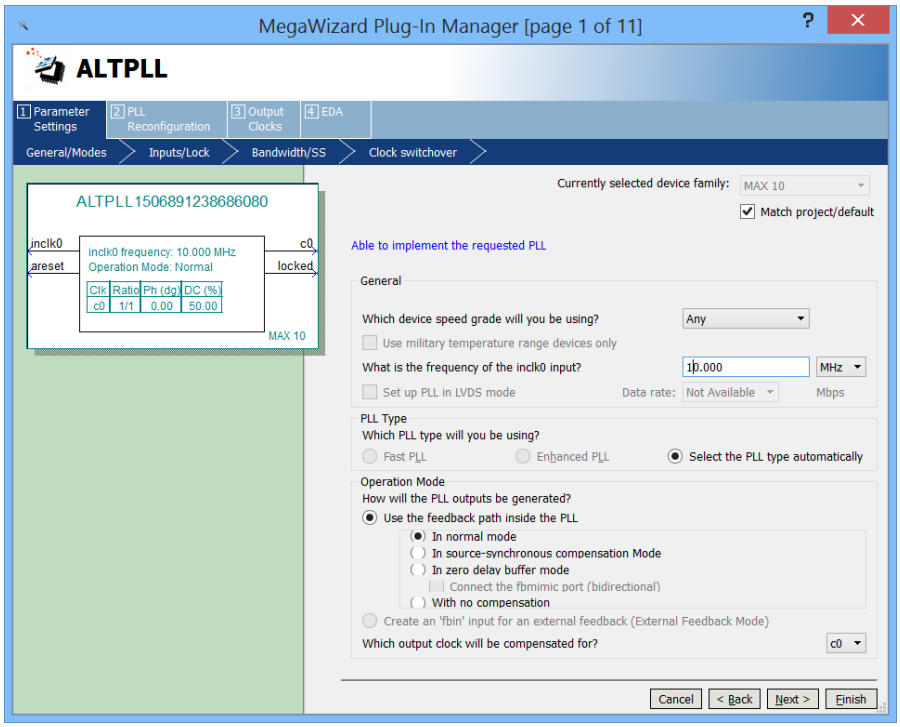
3) Before we add another pll for the ADC clock, we need to **add the external ADC clock input**.

In the IP catalog, under basic functions select Clocks and Clock Source. The Clock Source Dialog should appear:

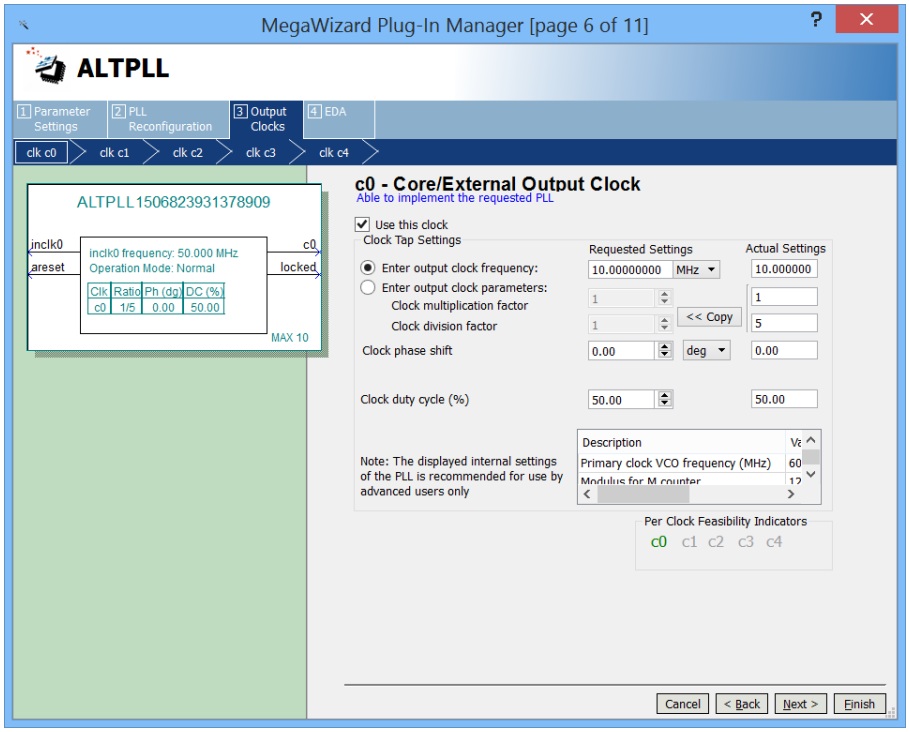


Change the clock Frequency to 10 MHz. Click Finish.

4) Next you need to **add another alt_pll to provide a clock for the ADC module**. In the IP catalog, under Basic Functions, then Clocks, then PLL select Avalon ALTPLL. Double click on the ALTPLL, the ALTPLL MegaWizard Plug-In Manager should appear. Enter 10.000 MHz for the inclk0 input frequency and accept all other defaults:



5) Next click on the Output Clocks tab and **set the output clock frequency to 10 MHz**. This will be used to drive the ADC module.



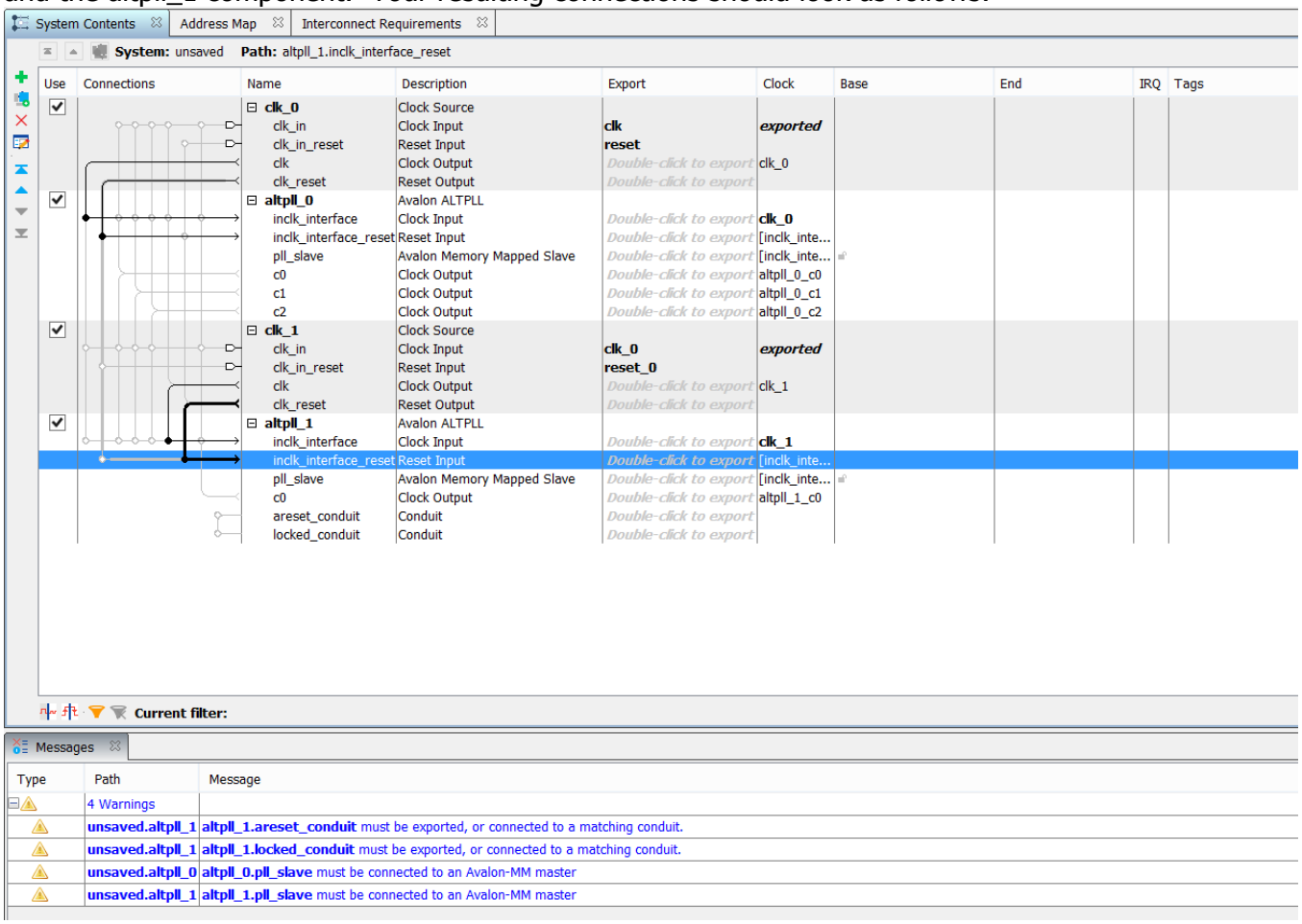
Click **Finish.**

6) **Connect the incoming clock and reset to the PLLs**

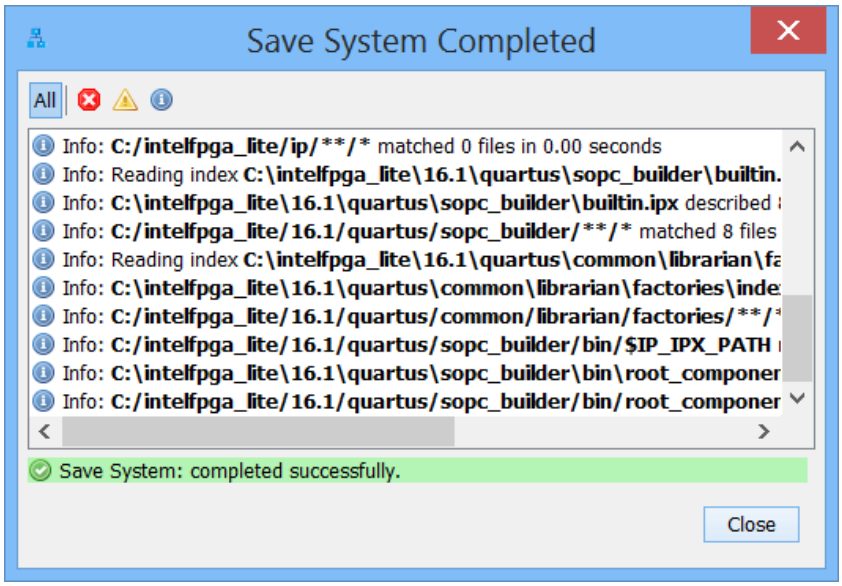
Qsys needs to know what clock and reset sources to use as the input to the PLL component. The clock and reset sources can come from an external source or from another component within the Qsys system. In our case, we will be connecting them to an external clock and reset.

- Click on the "System Contents" tab to the view of the components in our system. At this point, there are four components, a "Clock Source" component that was in the system by default when Qsys first launched, the second clock source you added, the 2 "Avalon ALTPLL" components. The Clock Source component is a Qsys component which brings in a clock and reset source from outside of the Qsys system. We will connect its nodes to the corresponding nodes on the Avalon PLL components.
- In the "Connections" column, hover over the connections and you will then be able to fill in connection dots to make connections.

Connect the clk_0 clock output port of the Clock Source to the inclk_interface of the altpll_0 component. Similarly connect the clk_reset reset output port of the Clock Source to the inclk_interface_reset of the altpll_0 component. Make the same connections between the clk_1 source and the altpll_1 component. Your resulting connections should look as follows:



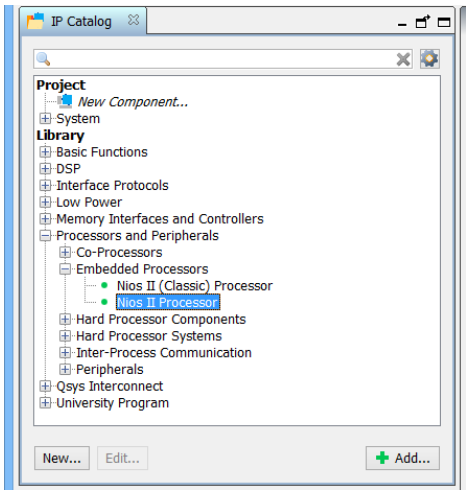
7) We have not yet saved our system. Let us use this opportunity to **save our work**. Click on **Save As** from the **File** menu and save the system as **Embed.qsys**. Close the window after the save completes.



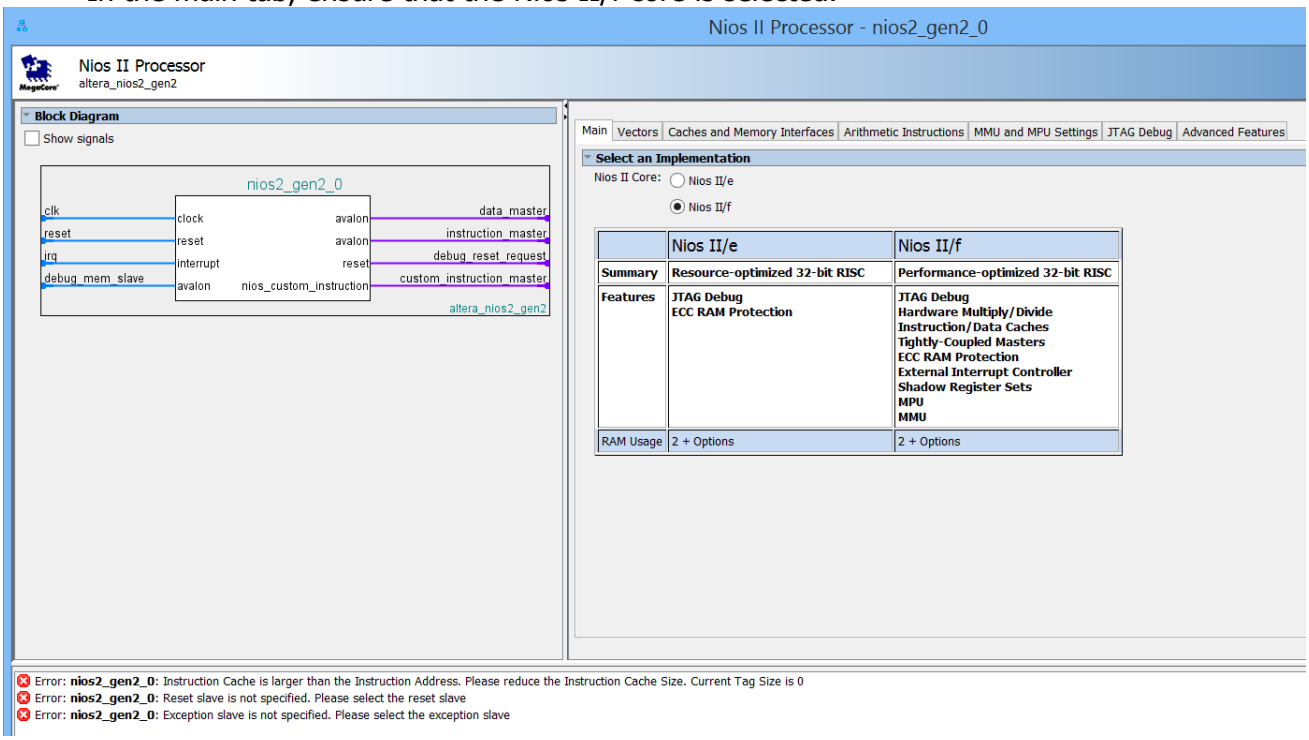
8) **Add a Nios II Processor**.

A CPU is needed to run the control algorithms. The NIOS II is a 32-bit CPU built in FPGA programmable logic, with a number of distinct benefits. Obviously, it can easily be configured and is quite configurable. Many functions of the core are optional, as we shall see. Additionally, combined with available bus interconnections in Qsys, it is very easy to extend the processor system to include a wide range memory and peripherals. You can include only what you need, so the resource usage can be very efficient. The performance is optimized, so speed is reasonable although not what you would expect from a hard-core processor. However, unlike hard-core processors obsolescence is not problem – once designed, the softcore system can be applied to FPGAs now and into the future.

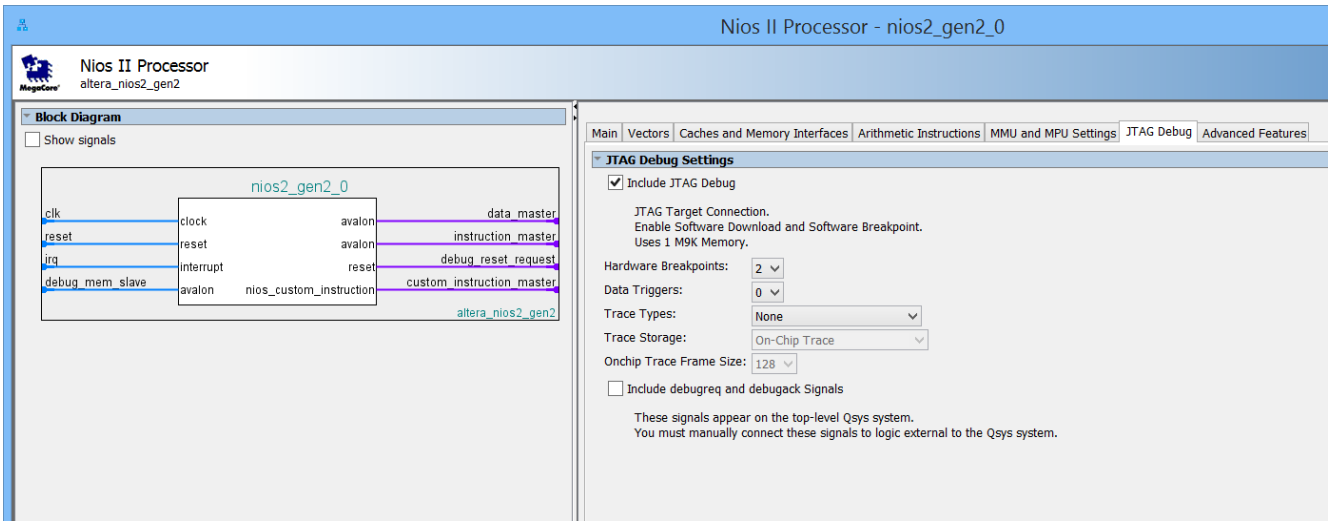
- From the **IP Catalog** pane, under Library, Expand Processors and Peripherals and then **Embedded Processors** and double click on **Nios II Processor)**. The Nios II processor configuration window opens.



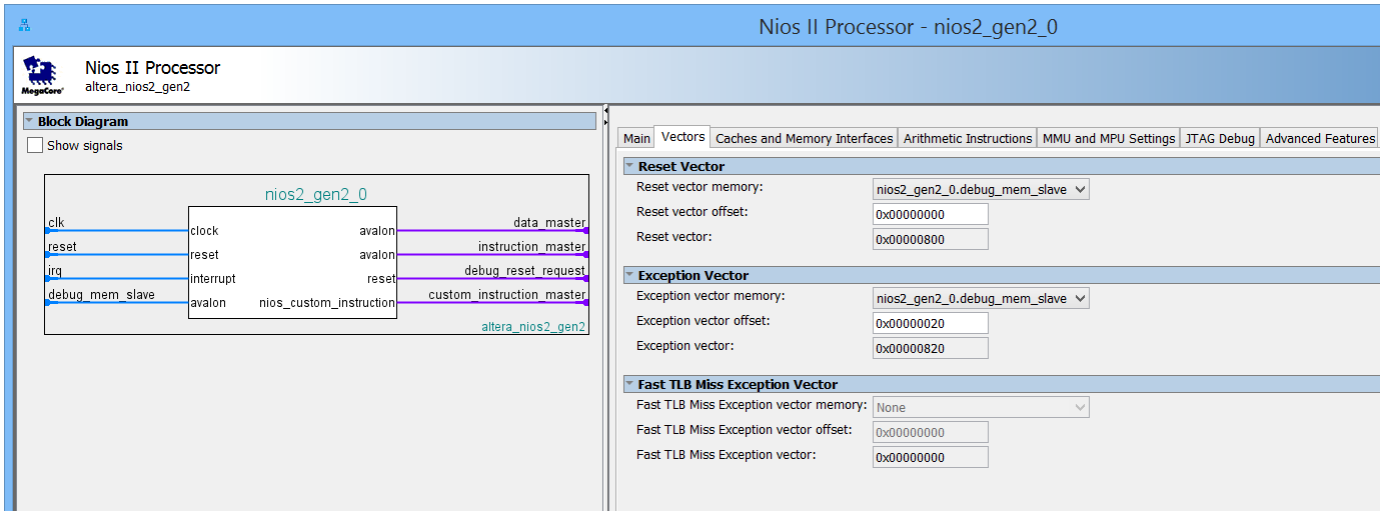
- In the main tab, ensure that the Nios II/f core is selected.



Click on the **JTAG Debug** tab and change the number of **Hardware Breakpoints** from 0 to **2**.



- In the Caches and Memory Interfaces tab, reduce size of the Instruction Cache to 2 Kbytes.
- In the Vectors tab, set the Reset vector Memory to nios2_gen2_0.debug_mem_slave, and do the same with the Exception Vector memory.

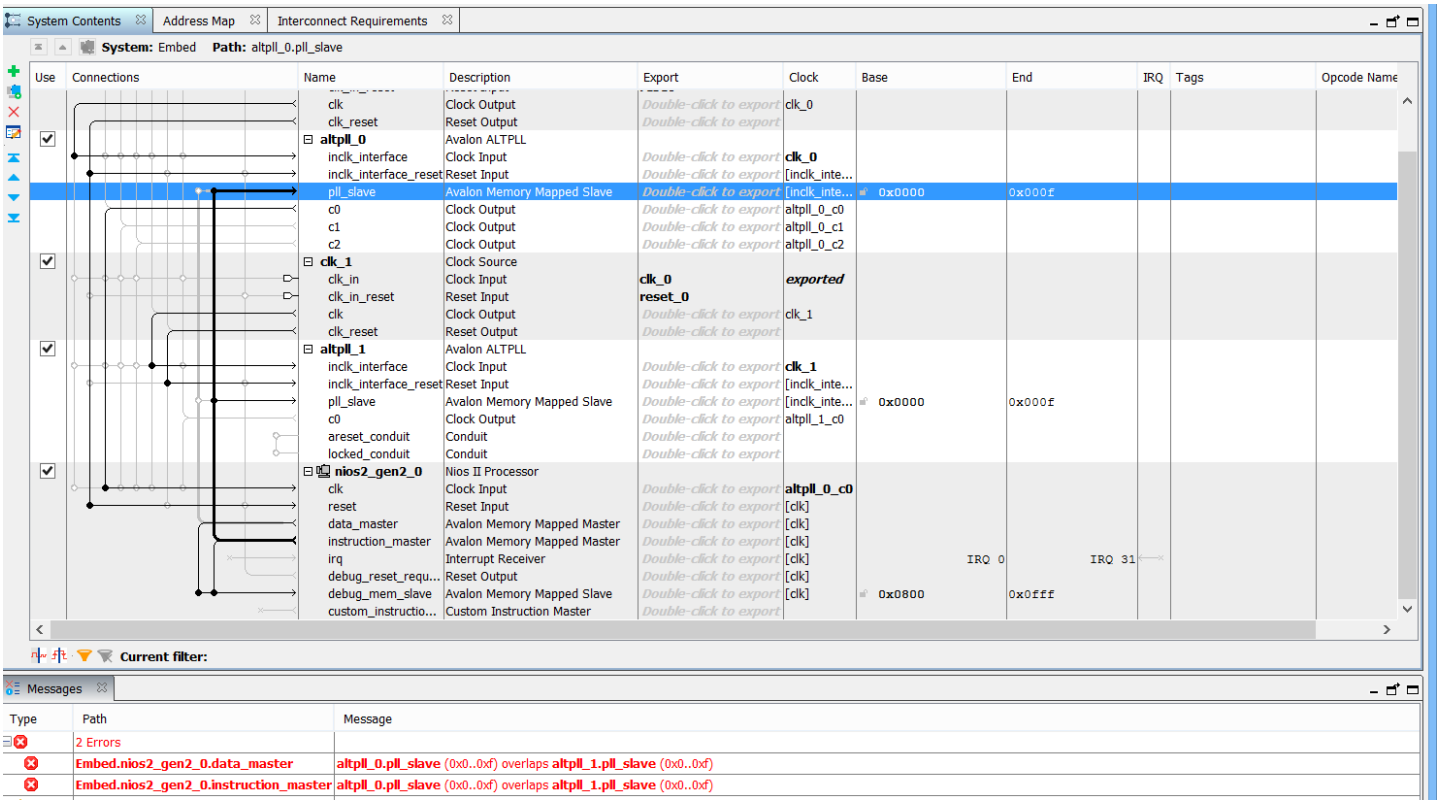


There are numerous other options on the various pages of the dialog box, including the ability to add an MMU and hardware divider, etc. We will keep things simple for now. All the remaining defaults should be accepted. Click **Finish**.

You will likely see 2 error messages asking for the NIOS2 to be connected to a clock and reset. Duh! Connect the NIOS clock input to the alt pll c0 output. Connect the NIOS2 reset input to the clk_0 reset output. The error messages should now go away. Yeah! See how easy this is?

The NIOS2 can reconfigure the PLLs on the fly through the Avalon Memory mapped bus interface. Connect the nios2 data master to the atlpll pll slaves. You may see errors due to overlapping pll base addresses, but we will deal with this a little later.

The system should now look like this:

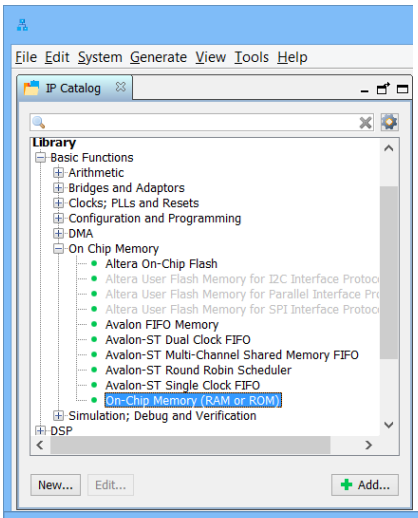


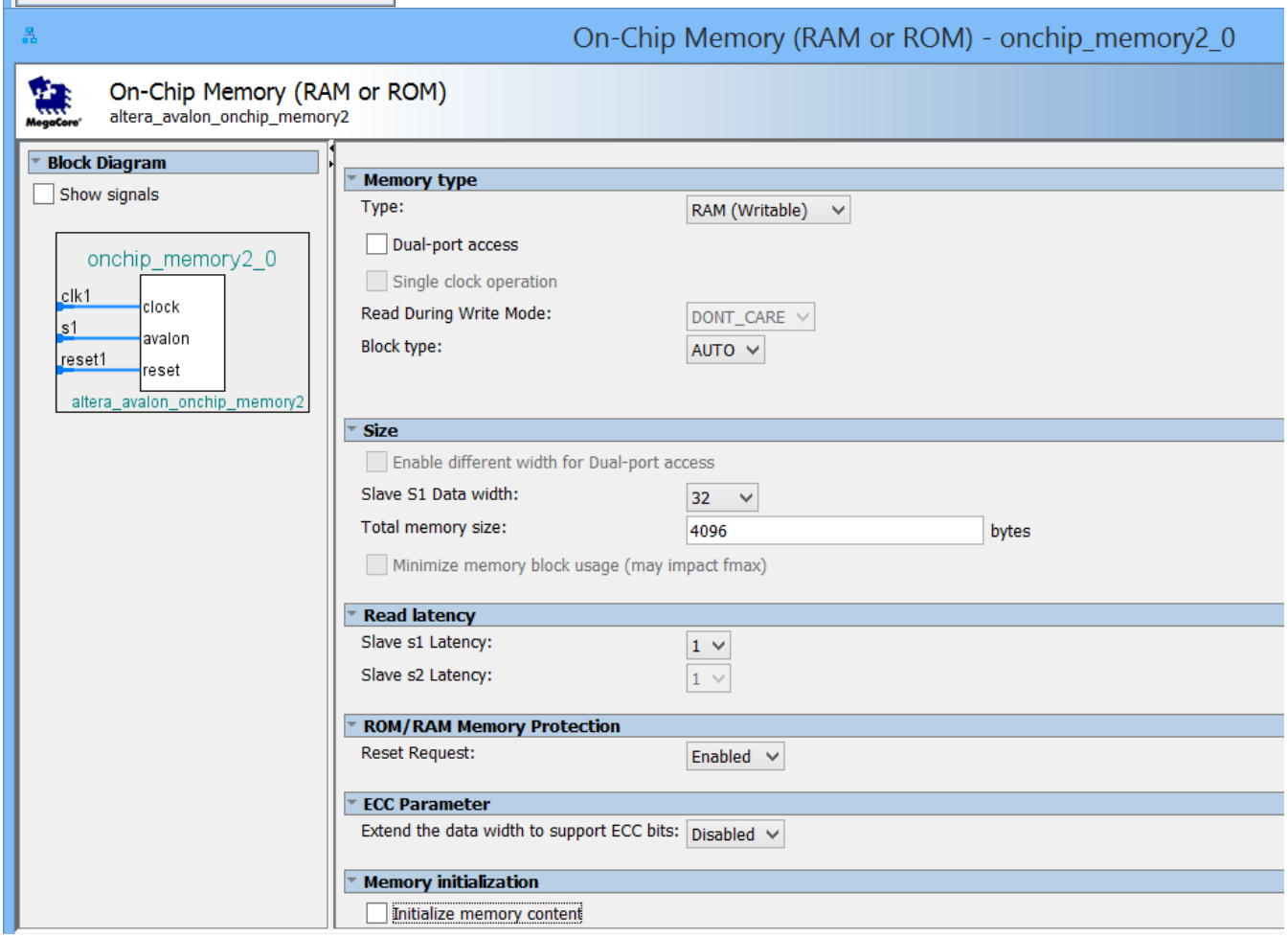
Hit File and then save, and close the save box.

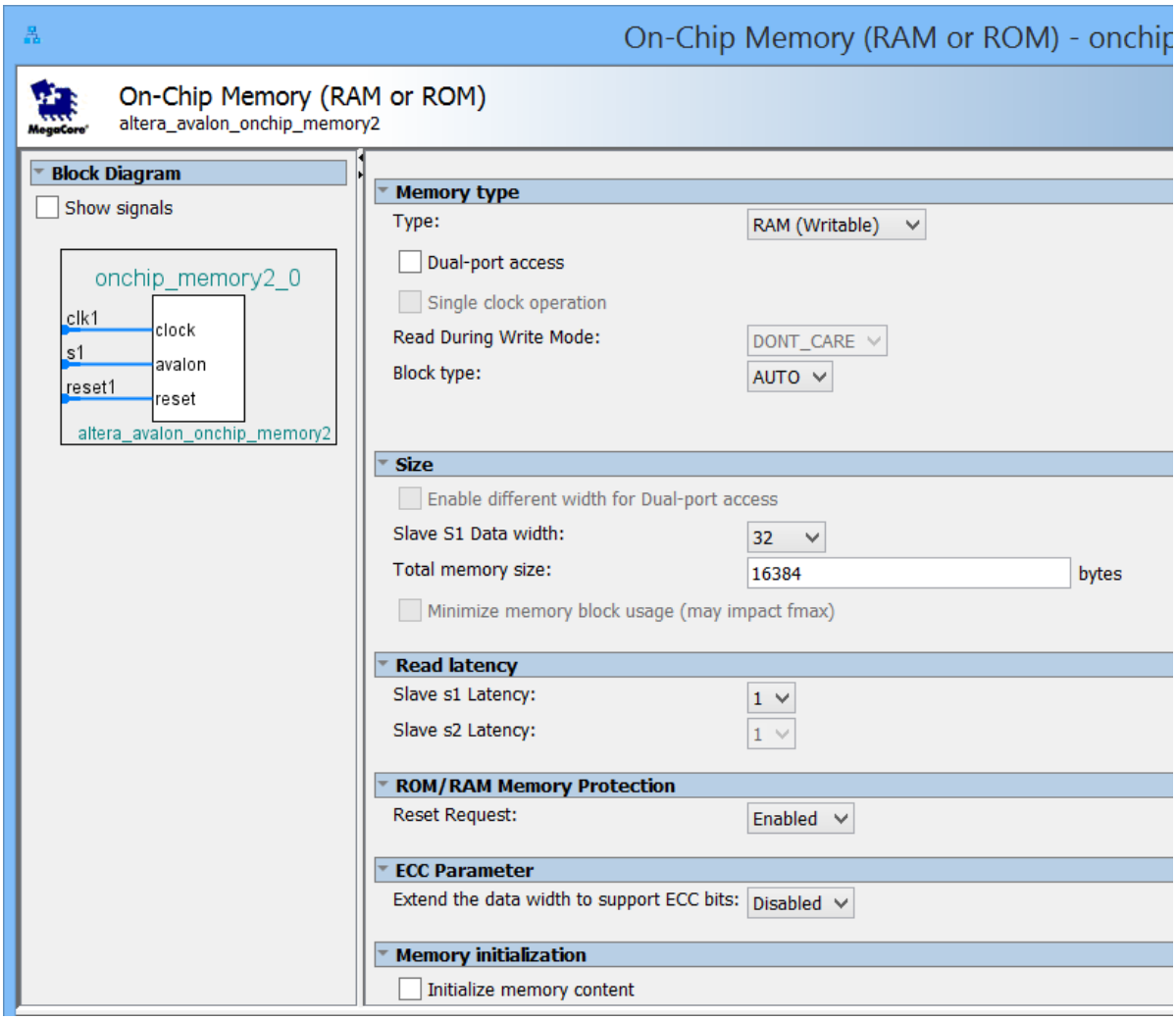
9) Add an on-chip RAM

Altera FPGAs provide internal on-chip memory blocks that can be used to build up an internal RAM (or ROM) block of memory. This provides the processor with access to very low-latency, high-speed memory for code or variable storage.

- In the IP Library on the left, Expand **Basic Functions**, then expand **On Chip Memory** and double click on **On-Chip Memory (RAM or ROM)**.


File Edit System Generate View Tools Help
IP Catalog
Library
Basic Functions
Arithmetic
Bridges and Adaptors
Clocks; PLLs and Resets
Configuration and Programming
DMA
On Chip Memory
Altera On-Chip Flash
Altera User Flash Memory for I2C Interface Protoc
Altera User Flash Memory for Parallel Interface Pr
Altera User Flash Memory for SPI Interface Protoc
Avalon FIFO Memory
Avalon-ST Dual Clock FIFO
Avalon-ST Multi-Channel Shared Memory FIFO
Avalon-ST Round Robin Scheduler
Avalon-ST Single Clock FIFO
On-Chip Memory (RAM or ROM)
Simulation; Debug and Verification
DSP
New...
Edit...
Add...


On-Chip Memory (RAM or ROM) - onchip_memory2_0
On-Chip Memory (RAM or ROM)
altera_avalon_onchip_memory2
Block Diagram
Show signals
onchip_memory2_0
clk1
s1
reset1
clock
avalon
reset
altera_avalon_onchip_memory2
Memory type
Type:
RAM (Writable)
Dual-port access
Single clock operation
Read During Write Mode:
DONT_CARE
Block type:
AUTO
Size
Enable different width for Dual-port access
Slave S1 Data width:
32
Total memory size:
4096
bytes
Minimize memory block usage (may impact fmax)
Read latency
Slave s1 Latency:
1
Slave s2 Latency:
1
ROM/RAM Memory Protection
Reset Request:
Enabled
ECC Parameter
Extend the data width to support ECC bits:
Disabled
Memory initialization
Initialize memory content



In the On-Chip Memory dialog that appears, uncheck the Initialize memory content box, set the "Total memory size" to 16k bytes to create a 16 KB RAM, and then accept all the other defaults and click Finish.

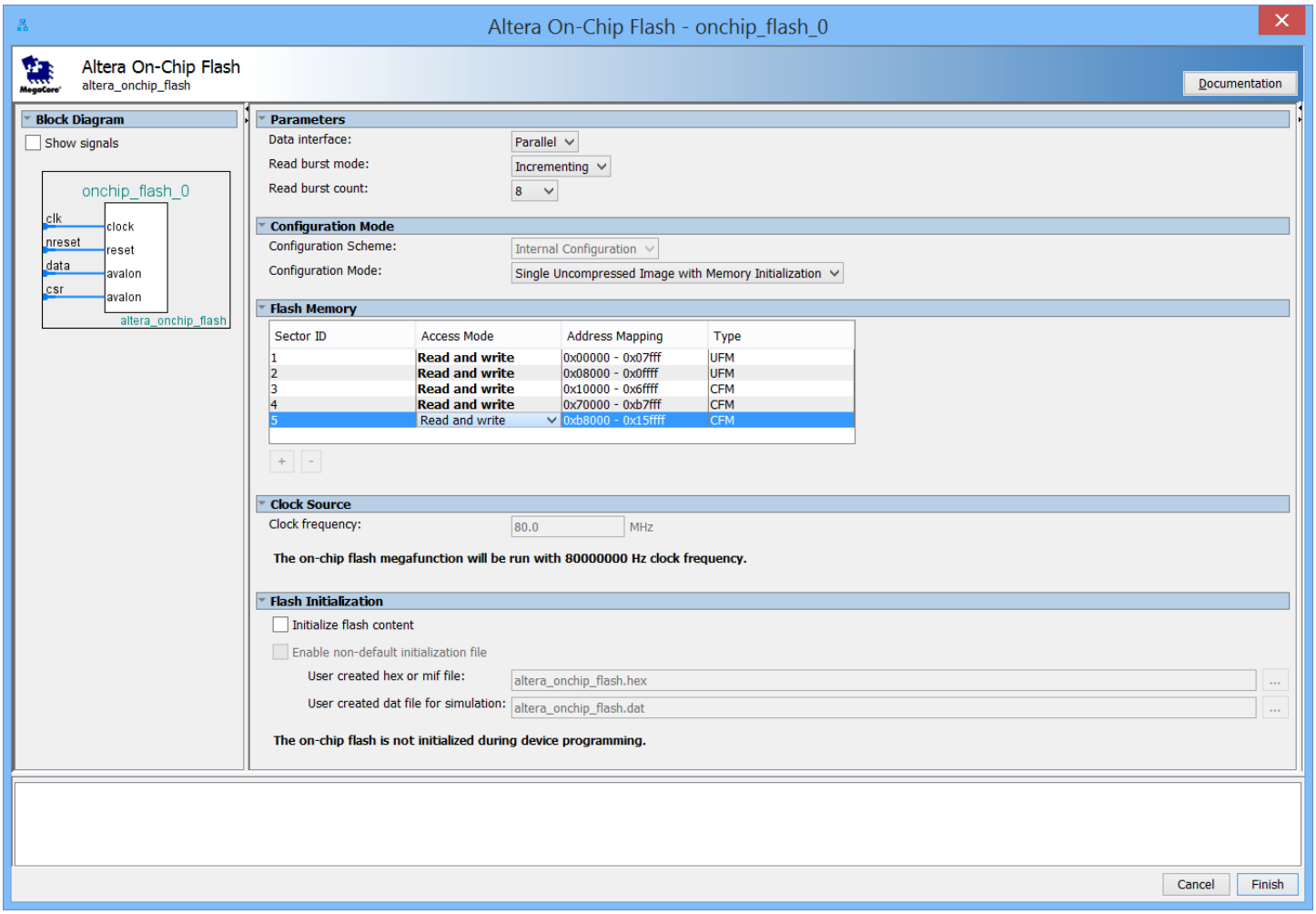
Connect the memory clock input to the alt pll c0 clock. Also connect the memory reset1 signal to the clk_0_reset output. Using the Connections column, connect the s1 Avalon Memory Mapped Slave interface of the onchip RAM to the nios2 instruction_master and nios2 data_master. Set the base address of the onchip RAM s1 to 0x4000.

10) Add an on-chip FLASH.

The MAX 10 FPGA contains on-chip flash which is used to store the FPGA configuration and can also be used to store Nios II code or other non-volatile data.

- In the Library, Expand Basic Functions. Expand On Chip Memory and double click on Altera On-Chip Flash.

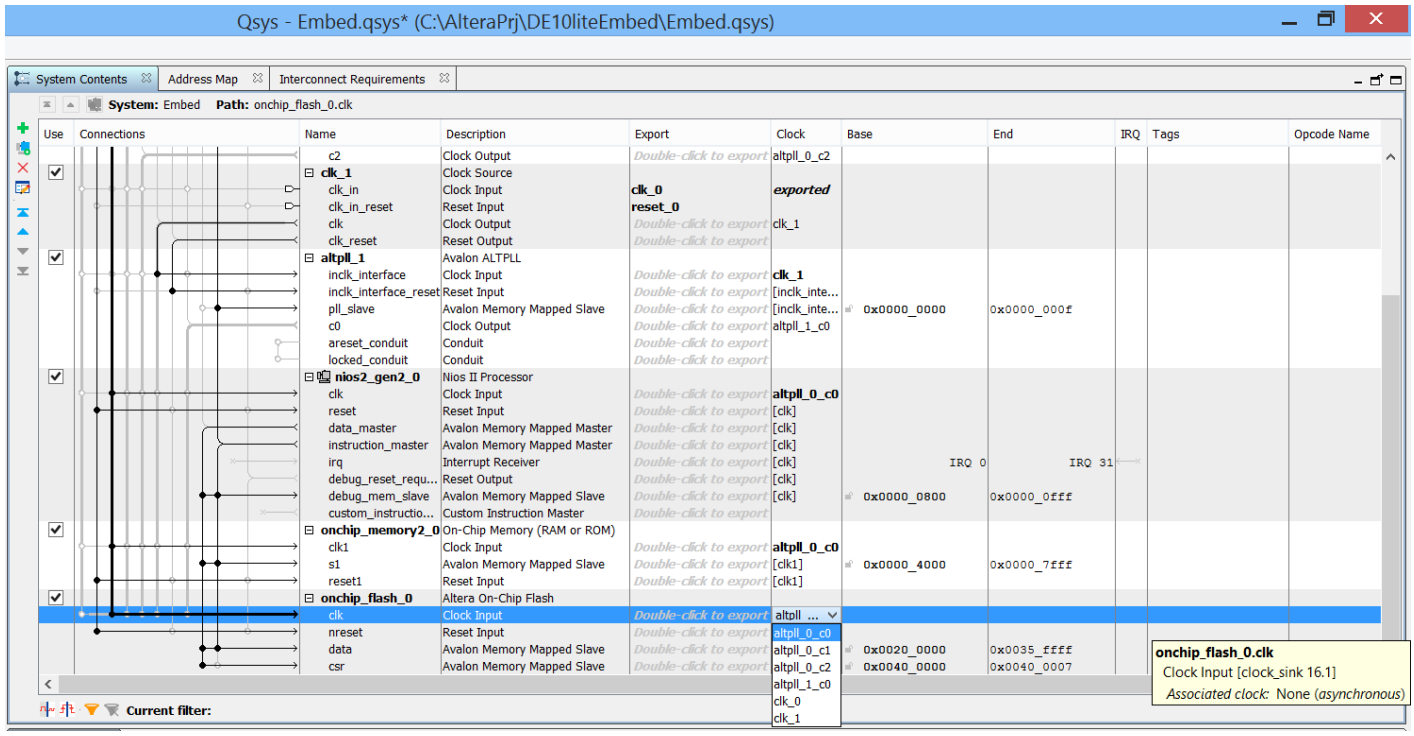
Change the configuration mode to "Single Uncompressed Image with Memory Initialization". Change the clock frequency to 80 MHz. Change the CFM Access mode to Read and Write. Hit Finish.



In the clock column, select altpll_0_c0 as the clock for the onchip flash from the pull-down menu.
Connect the onchip flash reset to the clk_0 reset signal.

- Connect the data to both nios2 data_master and nios2 instruction_master
- Connect the csr (control and status registers) to the nios2 data_master only.

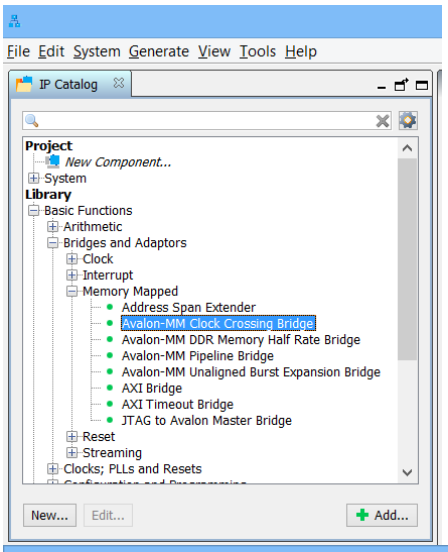
Change the base address of onchip flash data to 0x0020 0000
Change the base address of onchip flash csr to 0x0040 0000.

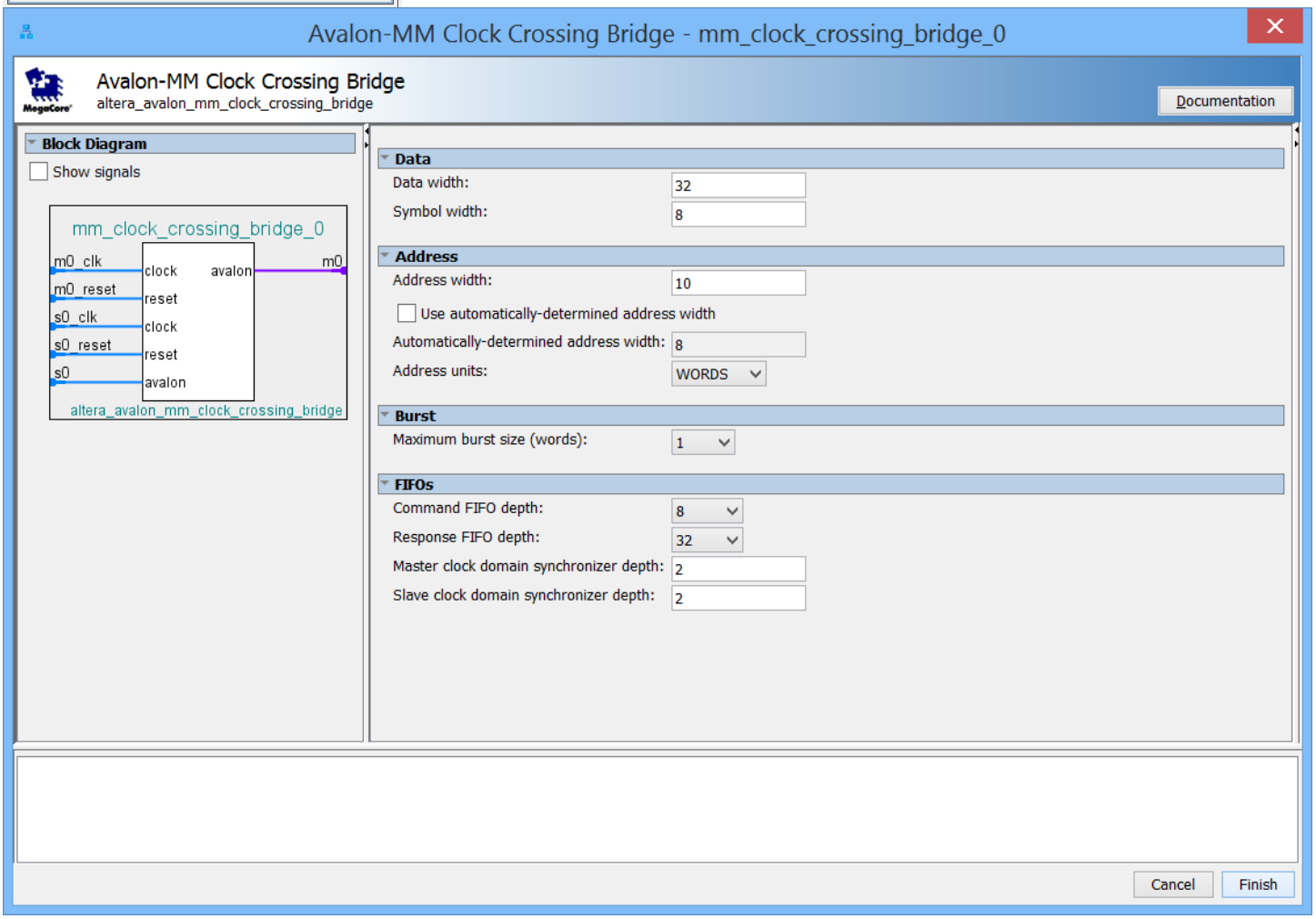


11) Add Avalon-Memory Mapped Clock Crossing Bridge Peripheral for the "slow" peripherals.

We will place several "slow" peripherals in a separate clock domain from the Nios II processor. With the bridge, a single clock crossing bridge is built into the system for all of the slow peripherals. Peripherals often have requirements to work with a different clock, creating a bridge to them is a general technique to handle the different clock domains. A bridge takes data, addressing and control signals on the Avalon bus, and translates them to signals needed by the peripheral so that data can be exchanged between the devices.

From the IP Catalog menu, expand Basic Functions. Expand Bridges and Adapters. Expand Memory Mapped and double click on Avalon-MM Clock Crossing Bridge.
Change the Address Units to WORDS. Change the Command FIFO depth to 8 and the Response FIFO depth to 32.





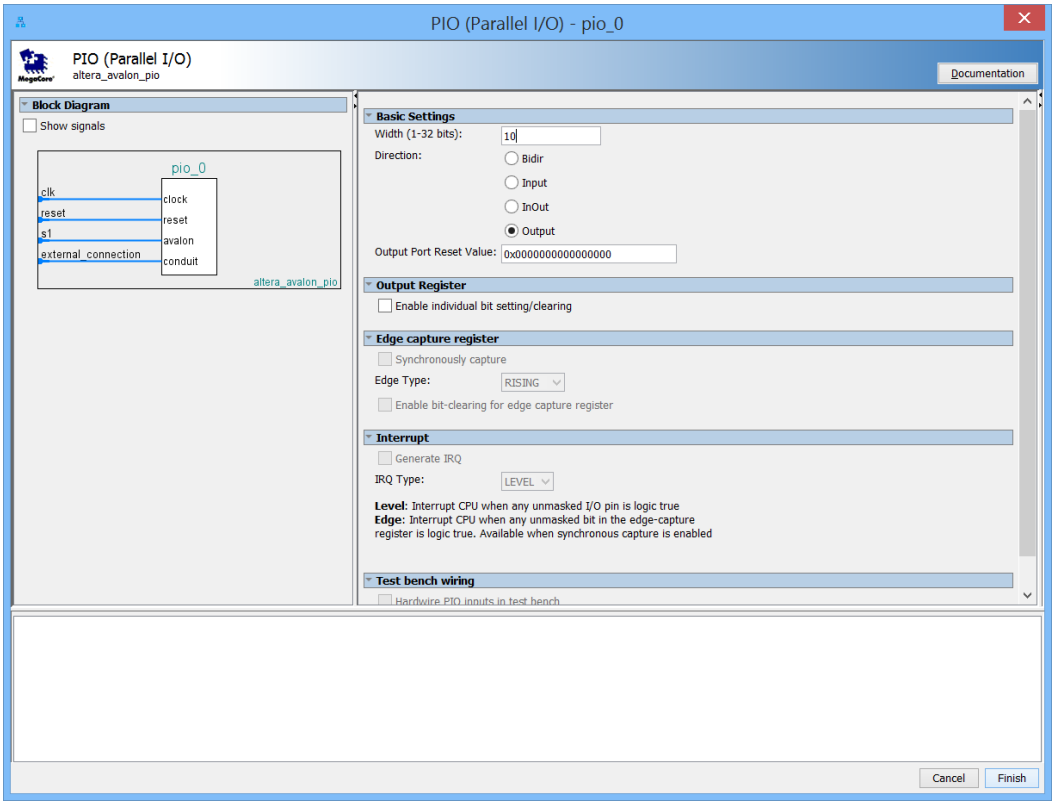
Click Finish.

Connect the memory mapped bridge m0 clock to the pll_0 c2 clock (40 MHz), and the s0 clk to the pll_0 c0 clock. Connect both resets to clk_0 reset as usual.
Connect the nios2 data_master port to the s0 Avalon Memory Mapped Slave of this bridge. The m0 master port will be connected in the upcoming steps.
Change the base address of the Avalon memory mapped slave to 0x0000 2000.

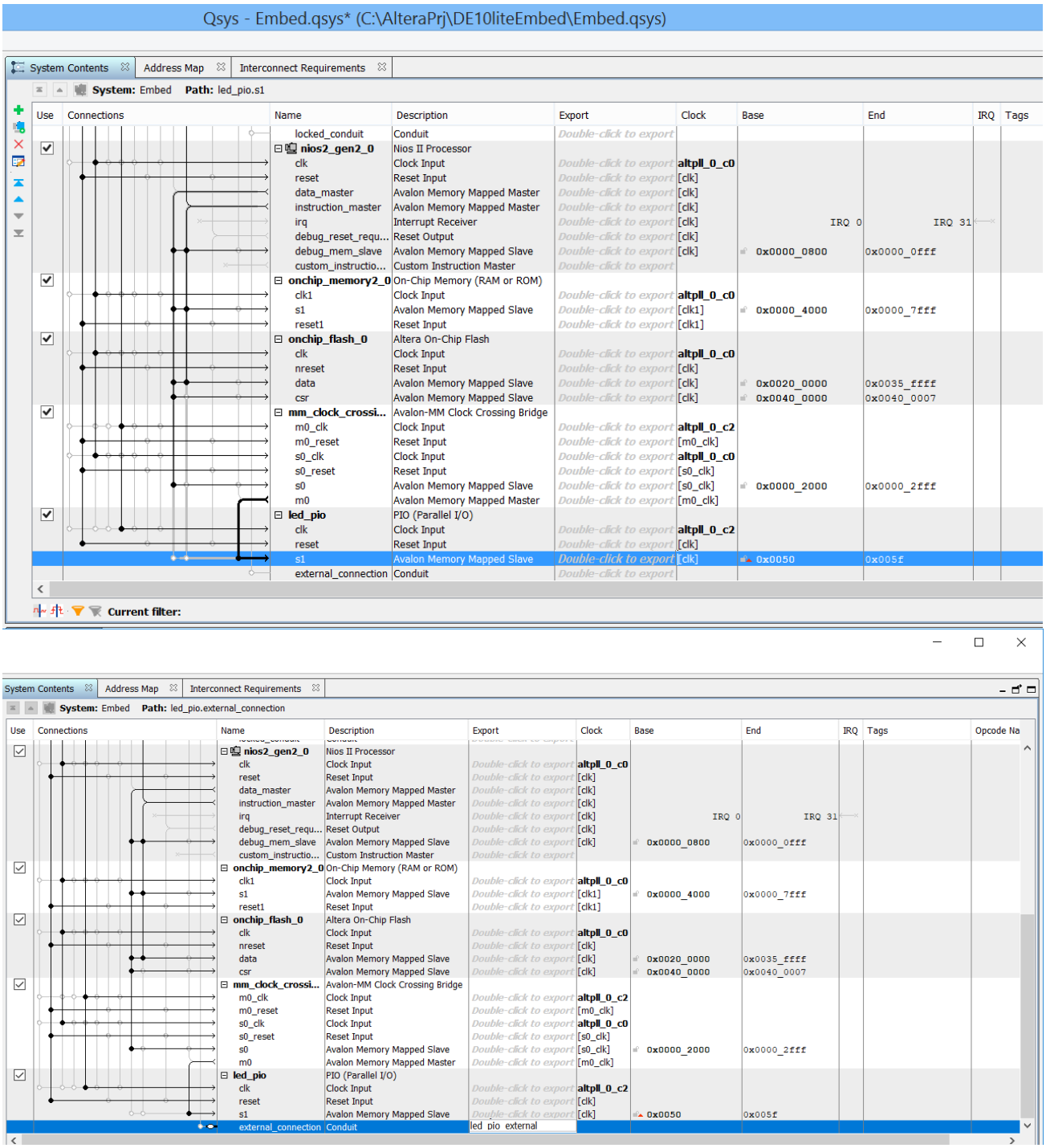
12) Add GPIO Peripheral for LEDs

The DE10-Lite has 10 LEDs. You can drive these LEDs with an output PIO peripheral. We will drive 10 of the LEDs with a PIO peripheral.

- From the IP Catalog menu, expand Processors and Peripherals, expand Peripherals, and double click on PIO (Parallel I/O).
- Set the "Width" to 10 bits. Ensure that the "Direction" is set to "Output" only. Click Finish.



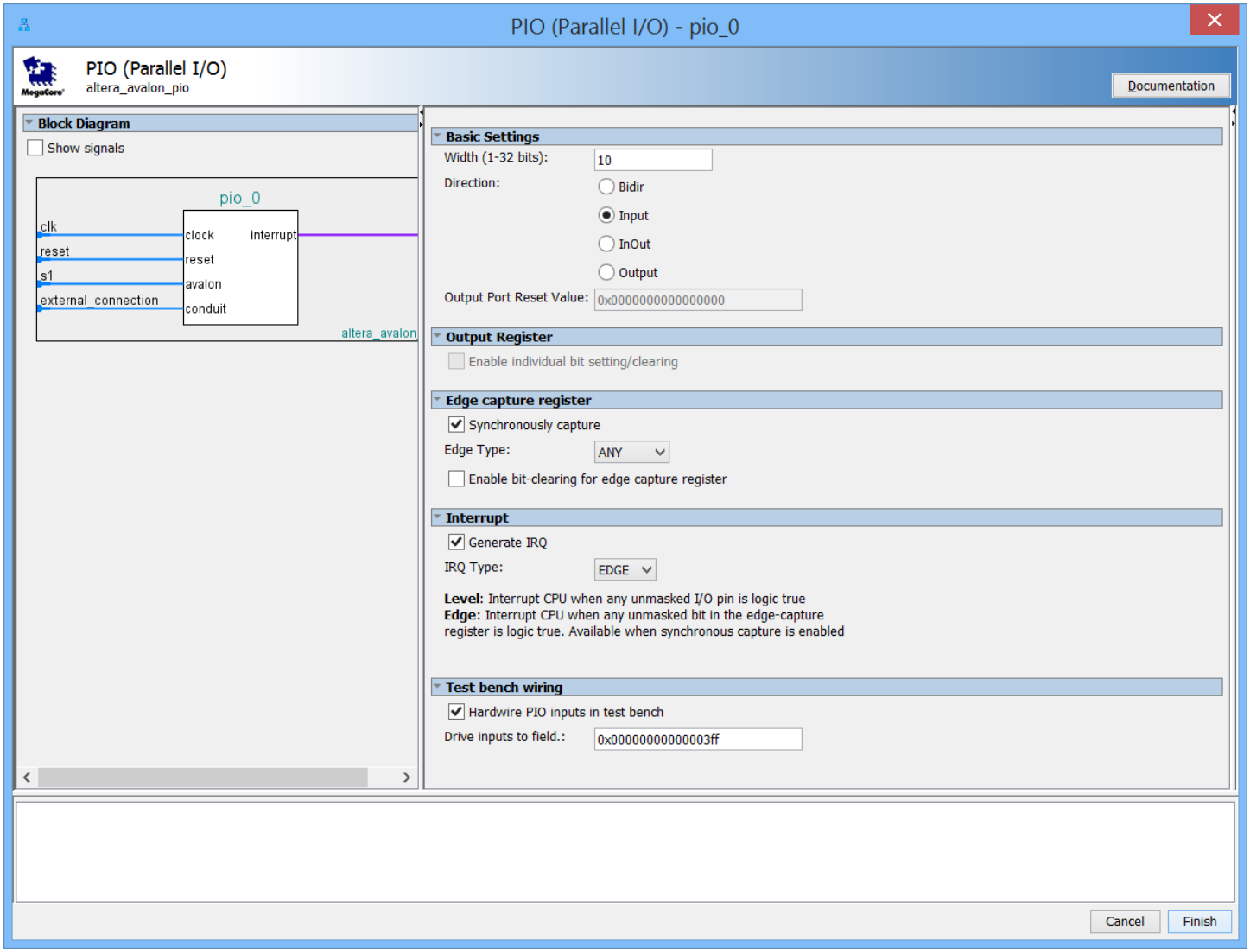
Right-click and rename the peripheral "led_pio".
Change the connection on the s1 slave port of the peripheral to be connected to the m0 master port of the clock crossing bridge.
In the clock column, select altpll_0_c2 as the clock for the clk Clock Input. Use clk_0 reset for the reset input.
Change the base address to 0x0050
Finally, double click in the "click to export" field next to the external_connection Conduit Endpoint and change the name to: led_pio_external



13) Add PIO Peripheral for Slide Switches

The DE10-Lite has 10 slide switches labeled "SW0", "SW1", "SW2" to "SW9" connected to 10 of the FPGA I/O pins. You can use an input PIO peripheral to detect when any of these switches have been toggled and signal an interrupt to the processor. These signals are assumed to be active low. From

the IP Catalog menu, expand Processors and Peripherals, expand Peripherals, and double click on PIO (Parallel I/O).
Set the "Width" to 10 bits. Set "Direction" to "Input."
Check the Synchronously capture and Any edge option in the Edge capture register section.
Check the Generate IRQ and Edge options in the Interrupt section.
Check the Hardwire PIO inputs in the test bench option and drive the inputs to 0x3FF.
Click Finish.

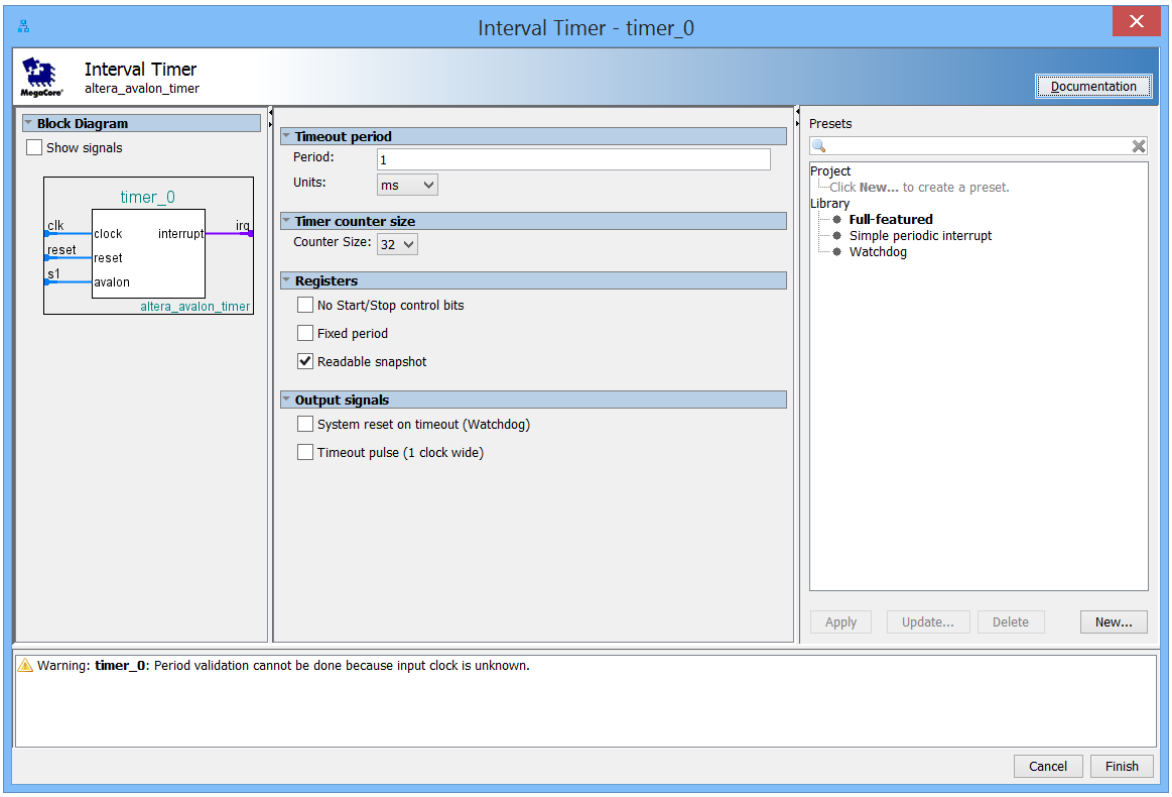


Right-click and rename the peripheral "slide_pio".
Change the connection on the s1 slave port of the peripheral to be connected to the m0 master port of the clock crossing bridge.
In the clock column, select alt pll_0 c2 as the clock for the clk Clock Input, and clk_0 reset as reset as before. Change the base address to 0x0040.
Finally, double click in the "click to export" field next to the external_connection Conduit Endpoint and name: slide_pio _external.

14) Add a 1 ms Interval Timer Peripheral

Many software applications require periodic interrupts to maintain various time bases and timing requirements within the application. A timer is a common and essential peripheral in most processor system.

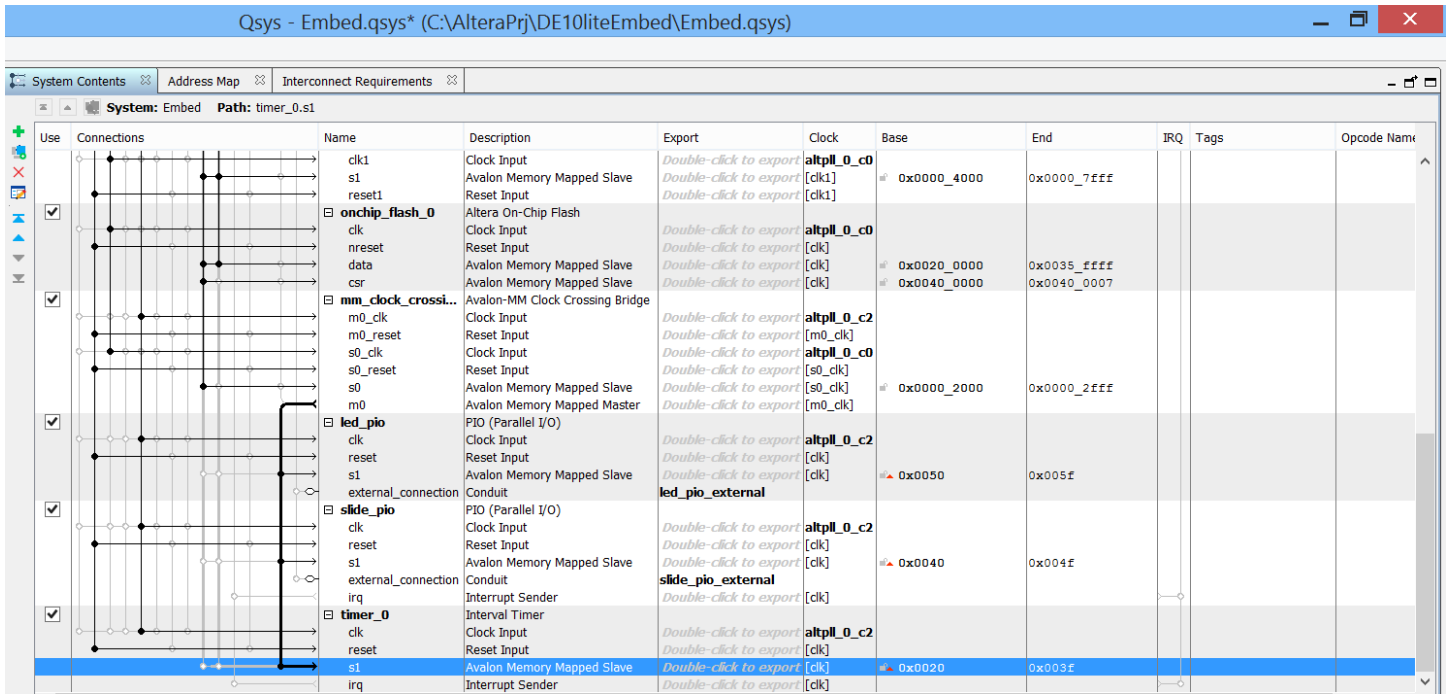
- From the IP Catalog menu, expand Processors and Peripherals, expand Peripherals, and double click on Interval Timer.
- Confirm the timer interval is 1 ms. Click Finish.



Connect the s1 slave port of the peripheral to be connected to the m0 master port of the clock crossing bridge.

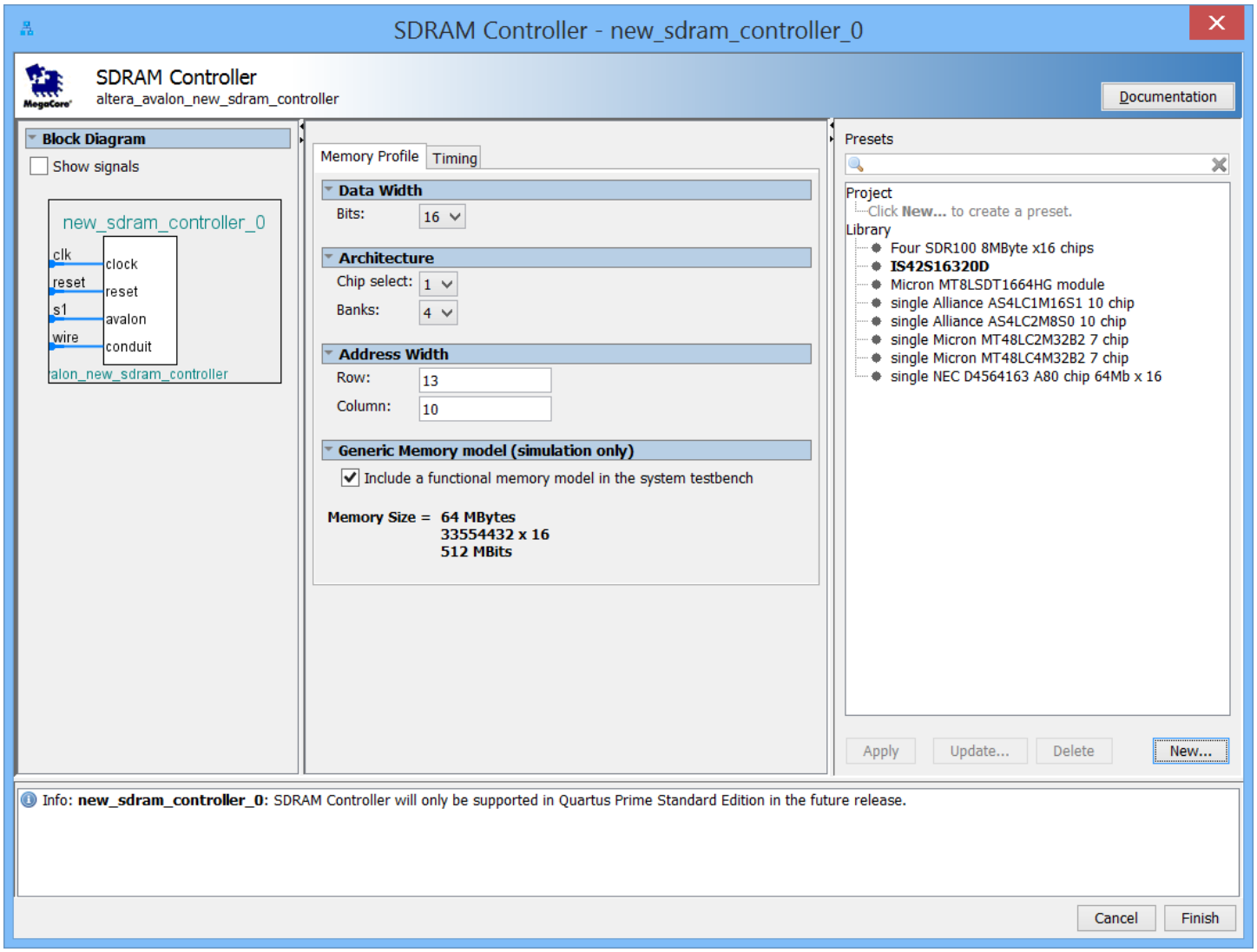
In the clock column, select alt pll c2 as the clock for the clk Clock Input. Attach the reset to clk_reset. Change the base address to 0x0020.

At this point the recent connections should look like this:

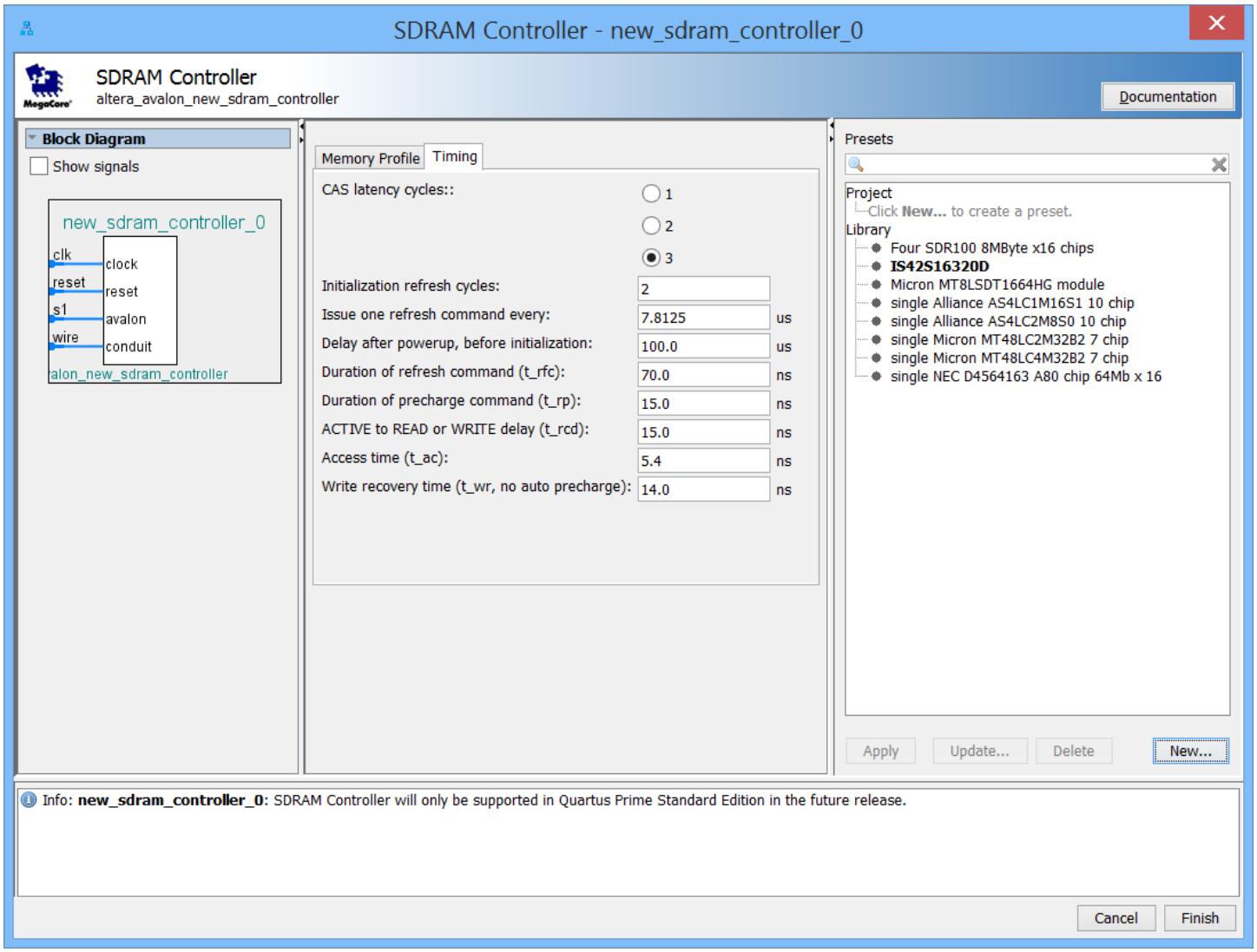


15) Add the SDRAM Controller.

In the IP Catalog, expand Memories Interfaces and Controllers. Expand SDRAM and double click on SDRAM controller. In the dialog box that appears, set the data width to 16, chip select to 1, banks to 4, Row to 13, Column to 10 and include the system testbench model:



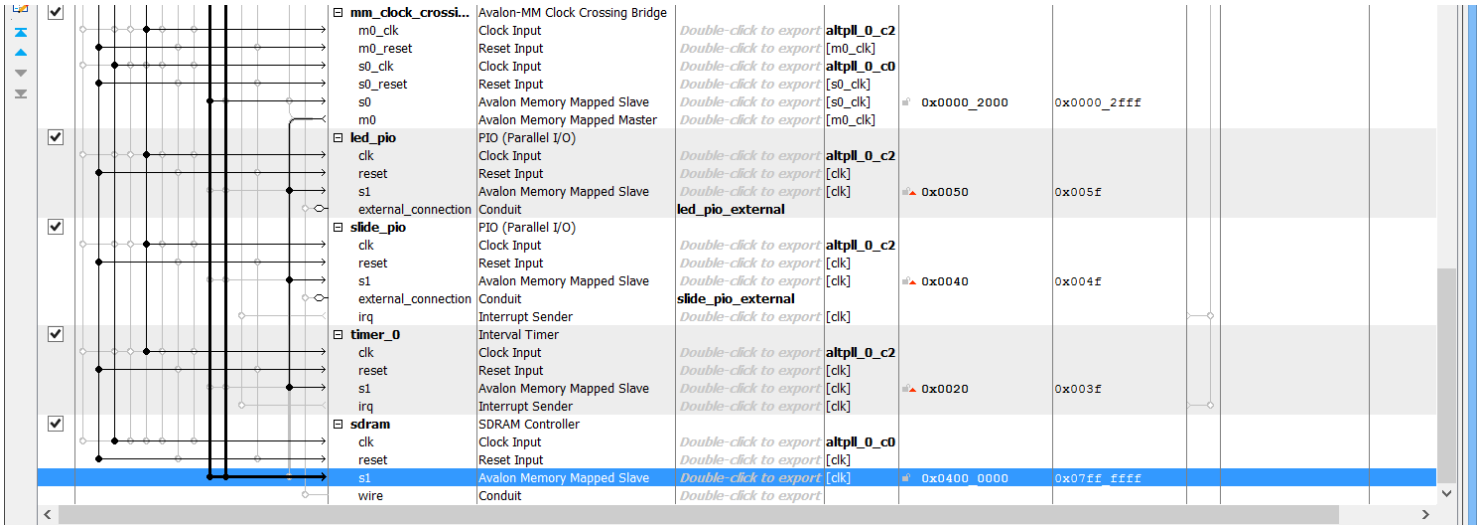
Set the Timing tab to the values below:



Click Finish.

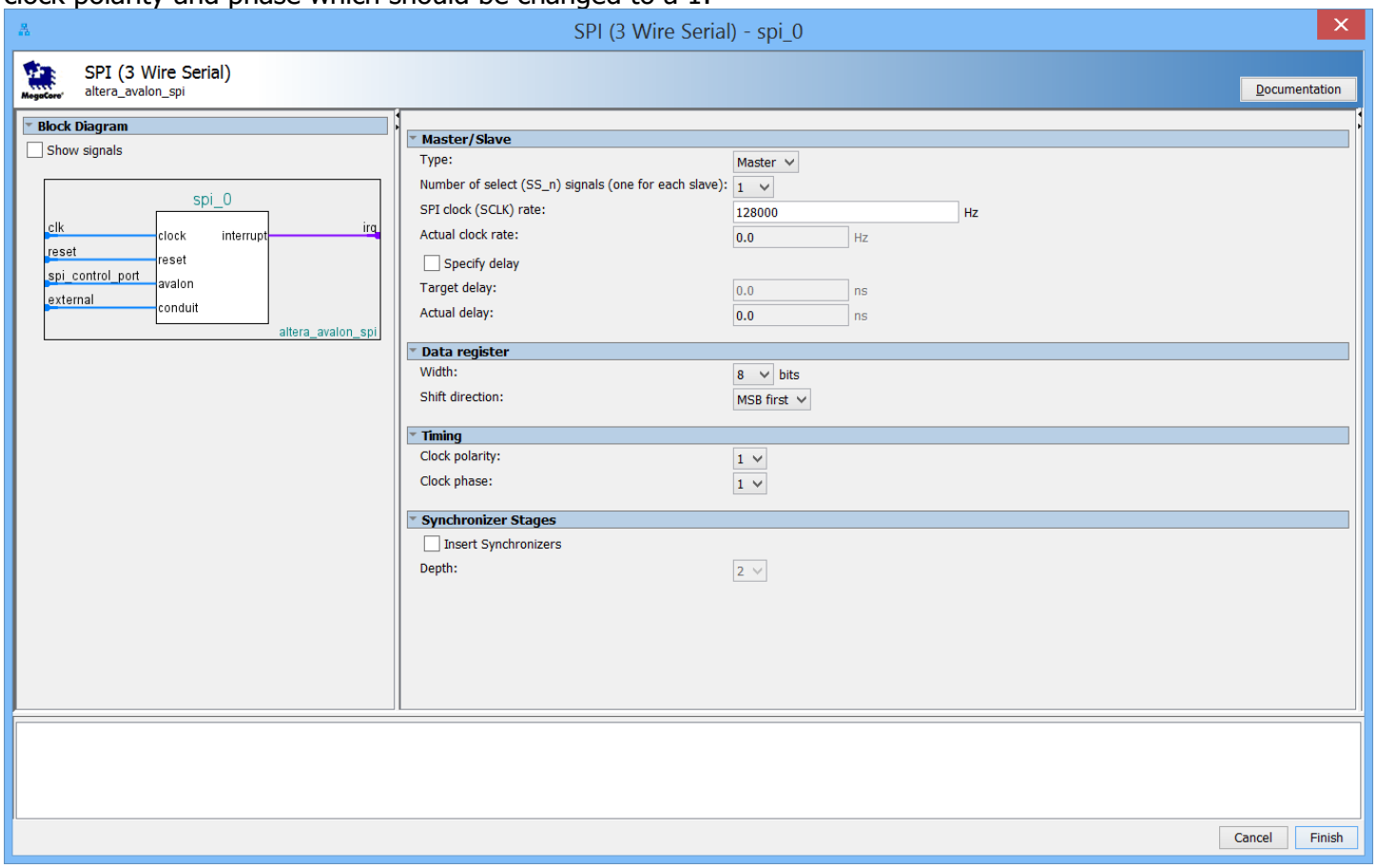
Right click on the Name field and choose Rename from the pop up menu. Name this RAM component "sdram".

- Using the Clock column, change the clock Input of the sdram to the alt_pll_0_c0 clock source.
- Using the Connections column, connect the s1 Avalon Memory Mapped Slave interface of the sdram to the nios2 instruction_master and nios2 data_master.
- Connect the reset to the clk_0 reset output.
- Set the base address to 0x0400 0000.



16) Add a SPI port for the Accelerometer.

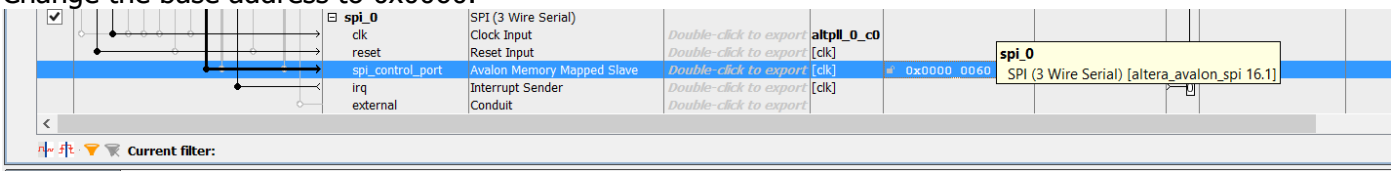
This will connect the ADXL345 accelerometer to the Nios II. In the IP catalog, expand Interface Protocols, expand Serial and then double click on SPI (3-wire serial). Accept all defaults, except for clock polarity and phase which should be changed to a 1:



Click Finish.

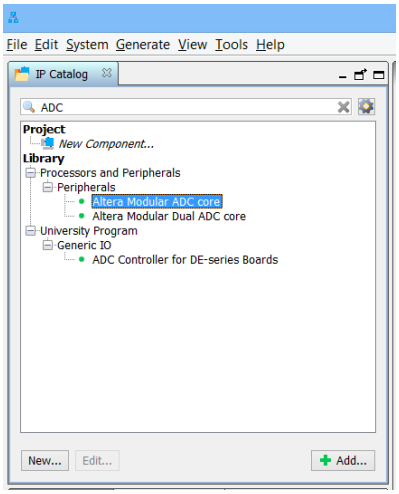
In the clock column, select alt_pll_0_c0 as the clock for the spi_accelerometer.

- Connect the spi_control_port to only the nios2 data_master
- Connect the irq to the nios2 irq
- Connect the reset to clk_0 reset output
- Change the base address to 0x0060.

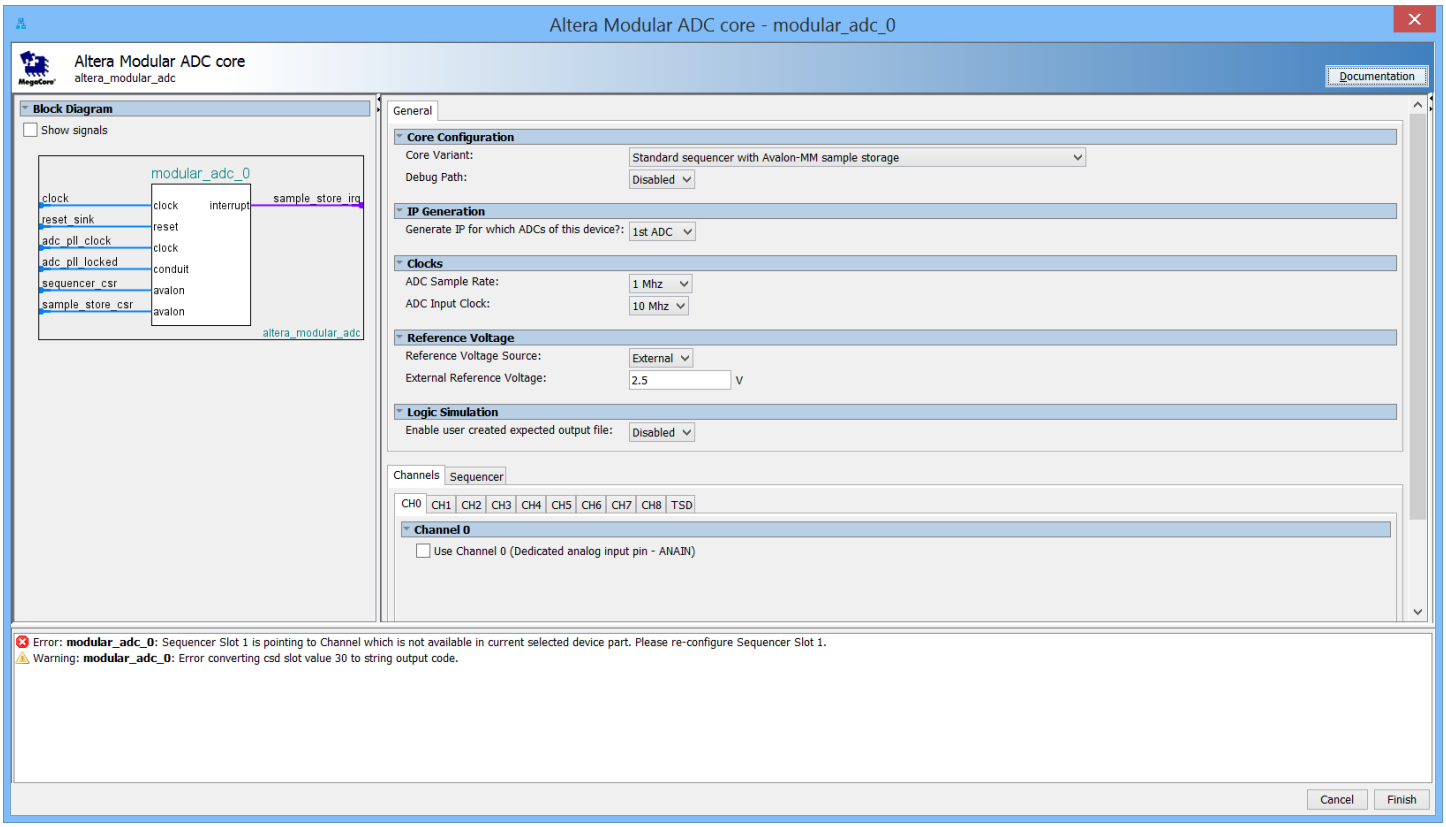


17) Add the ADC module.

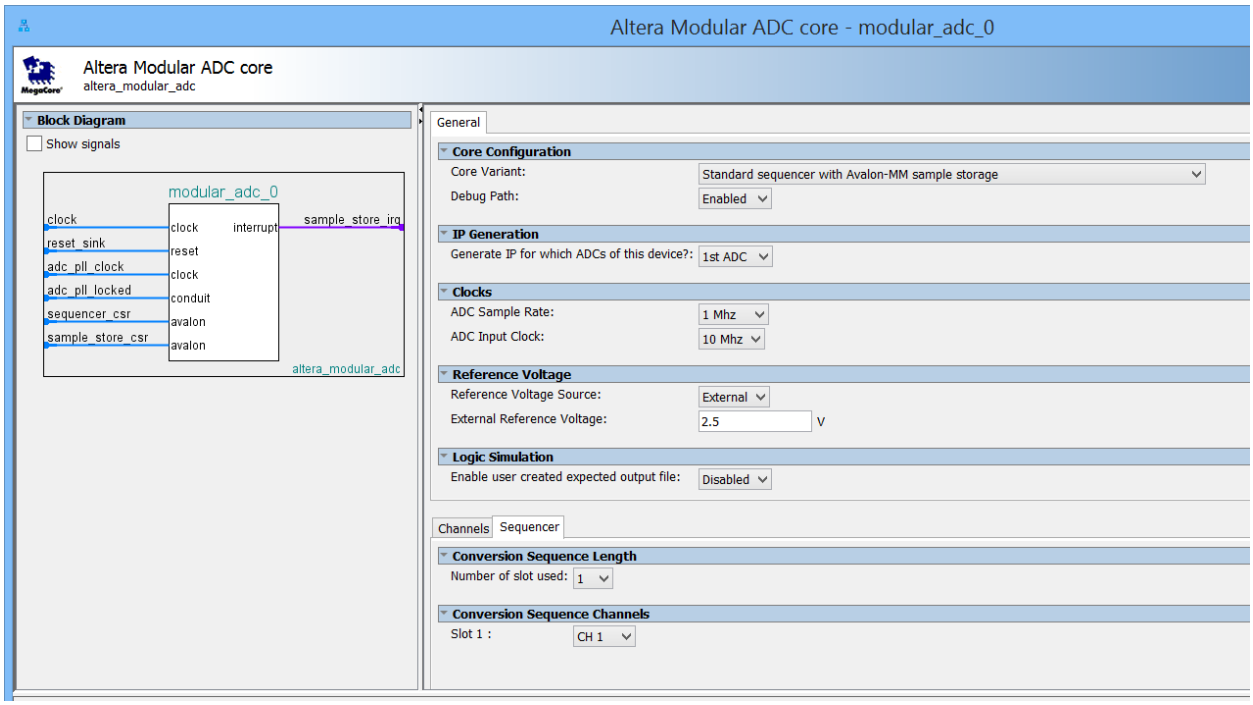
In the IP catalog in the upper left hand corner, type ADC in the search box. Select the Altera Modular ADC core and then click add in the lower right corner of this window to add the ADC module.



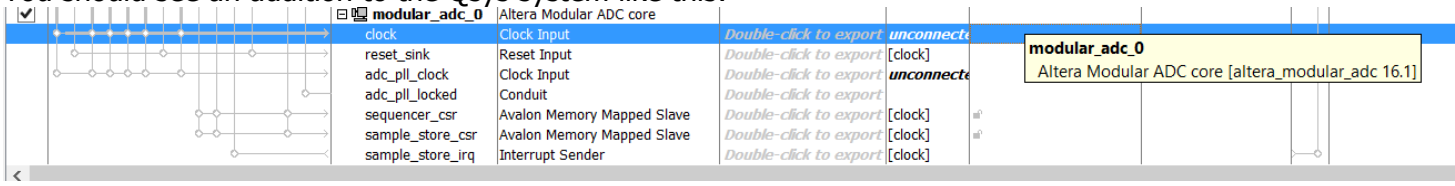
The ADC core configuration window should appear:



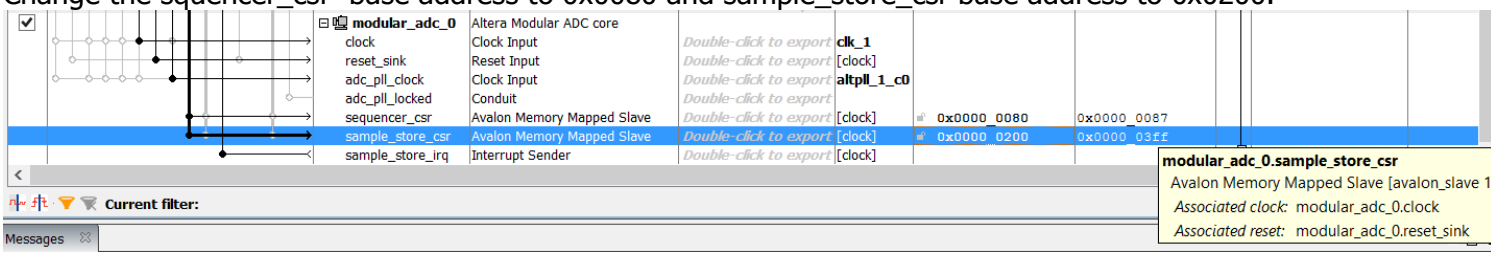
Accept all the settings in the General tab except Enable the Debug Path. In Channels, select CH1 and check the box to use Channel 1 in the Sequencer, set the number of slots used to 1, and set Slot 1 to CH 1. Click Finish.



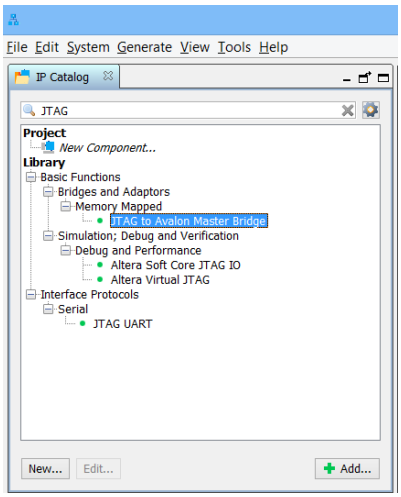
You should see an addition to the Qsys system like this:



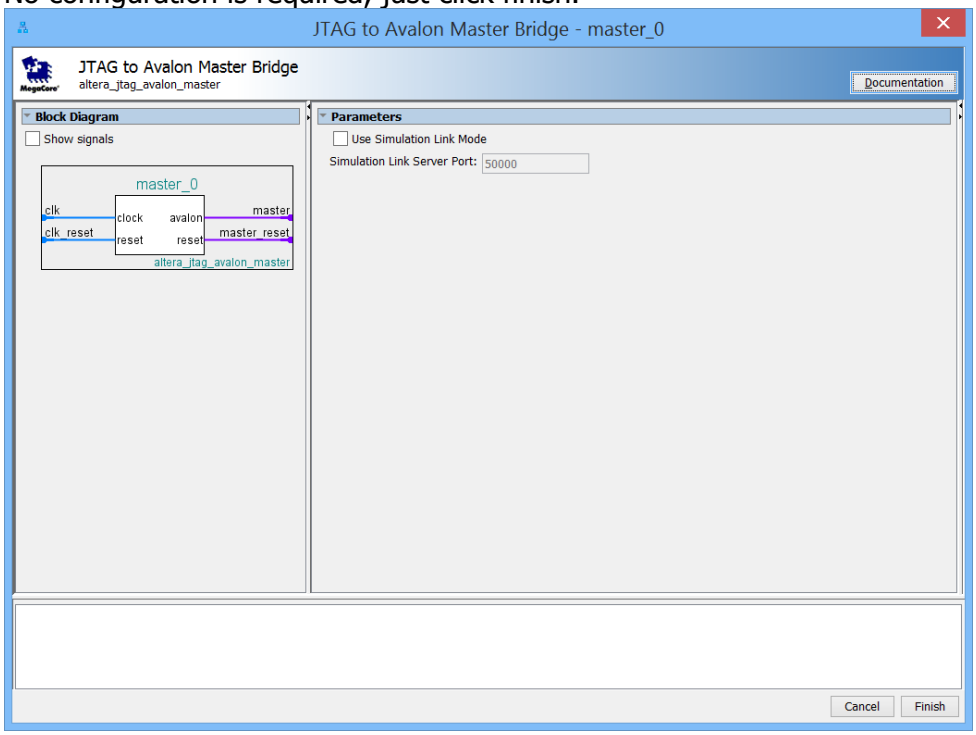
Connect the clock clk_1, and the reset to clk_1 reset output.
Connect the alt_pll_1 c0 output to the adc_pll_clock input, and the altpll_1 locked conduit to the adc_pll_locked Conduit.
Connect the squencer_csr and sample_store_csr to the nios2 data master only.
Connect the sample_store_irq to the nios2 irq.
Change the squencer_csr base address to 0x0080 and sample_store_csr base address to 0x0200.



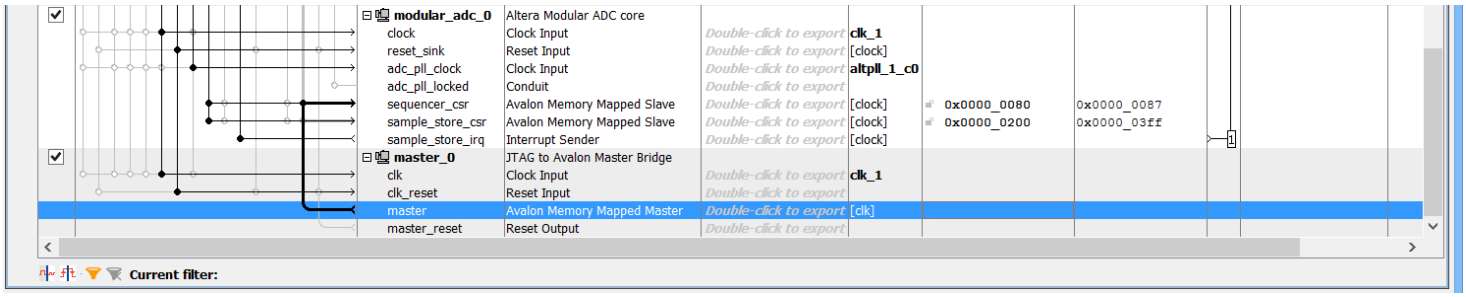
To use the built-in debugging features, you need to add a JTAG to Avalon Master Bridge. This gives you a master interface to talk through with the "System Console" tool which is a part of the Quartus installation (Tools menu). Type JTAG in the IP Catalog search box. Select and add the JTAG to Avalon Master Bridge:



No configuration is required, just click finish.



Connect the JTAG bridge clock to clk_1, and the reset to clk_1 rest. Connect the ADC module sequencer_csr slave to the master_0 master.

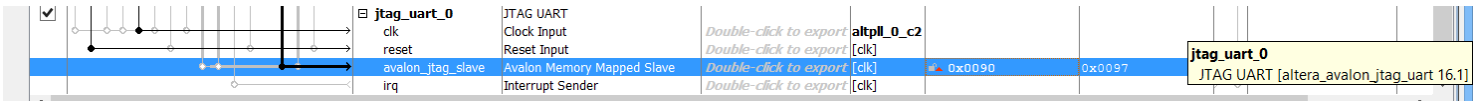


18) Add the JTAG UART Peripheral

Many software developers like 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 is useful for these purposes.

- From the IP Catalog menu, expand Interface Protocols, expand Serial and double click on JTAG UART.
- The default settings are acceptable. Click Finish.
- Change the connection on the Avalon_jtag_slave port of the peripheral to be connected to the m0 master port of the mm_clock_crossing_bridge.
- In the clock column, select alt pll c2 as the clock for the clk Clock Input. Connect the reset to clk_0 reset.

Change the base address to 0x0090.



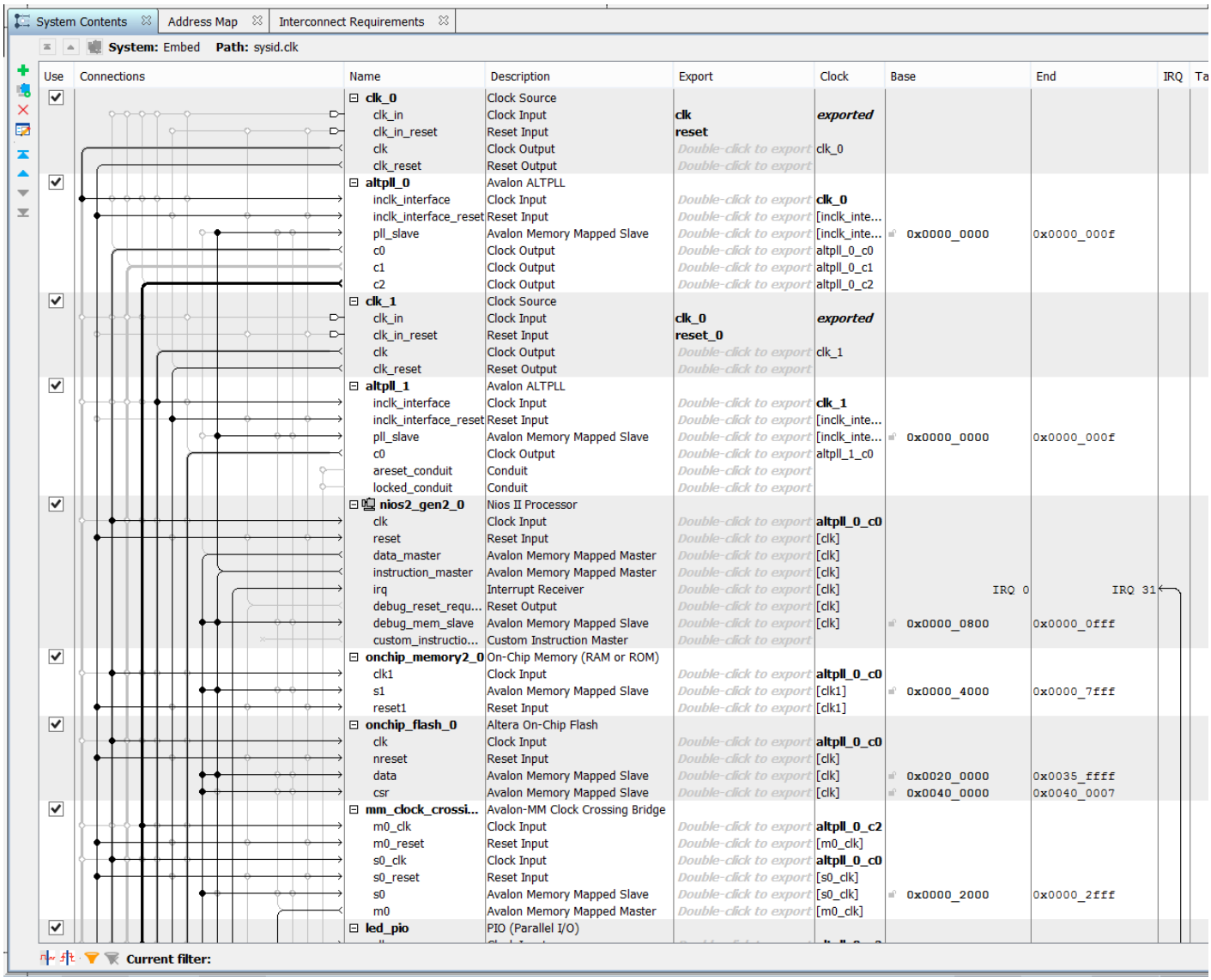
19) **Add a System ID**.

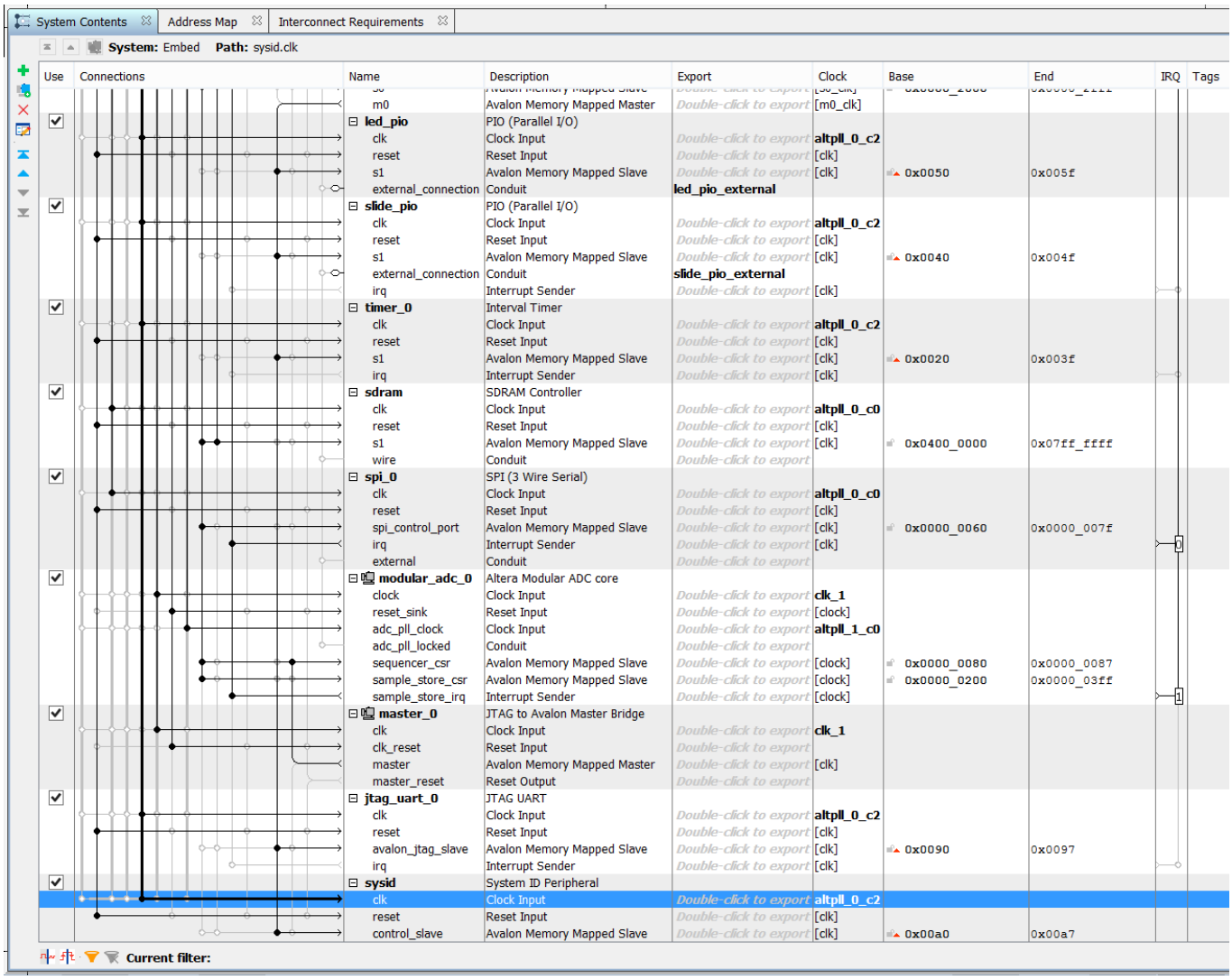
This is a VERY IMPORTANT peripheral to have in your system. It allows the Nios II development tools to validate that the software application is being built for the correct hardware system.

- From the IP Catalog menu, select Basic Functions -> Simulation; Debug and Verification -> Debug and Performance -> System ID Peripheral. Double click to add the component to the system.
- The sysid dialog box appears. Click Finish.
- Rename as "sysid". The component must be named "sysid" to be compatible with Nios II software drivers and build tools.
- Change the connection on the control_slave port of the peripheral to be connected to the m0 master port of the mm_clock_crossing_bridge.
- In the clock column, select altpll_0_c2 as the clock for the clk Input. Connect the reset input to clk_0 reset.

Change the base address to 0x00A0.

Congratulations, you have completed the Qsys design entry! Your result should look something like:


System Contents
Address Map
Interconnect Requirements
System: Embed Path: sysid.clk
Use
Connections
Name
Description
Export
Clock
Base
End
IRQ
clk_0
Clock Source
clk_in
Clock Input
clk
exported
clk_in_reset
Reset Input
reset
clk
Clock Output
clk_0
clk_reset
Reset Output
altpll_0
Avalon ALTPLL
inclk_interface
Clock Input
clk_0
inclk_interface_reset
Reset Input
pll_slave
Avalon Memory Mapped Slave
0x0000_0000
0x0000_000f
c0
Clock Output
altpll_0_c0
c1
Clock Output
altpll_0_c1
c2
Clock Output
altpll_0_c2
clk_1
Clock Source
clk_in
Clock Input
clk_0
exported
clk_in_reset
Reset Input
reset_0
clk
Clock Output
clk_1
clk_reset
Reset Output
altpll_1
Avalon ALTPLL
inclk_interface
Clock Input
clk_1
inclk_interface_reset
Reset Input
pll_slave
Avalon Memory Mapped Slave
0x0000_0000
0x0000_000f
c0
Clock Output
altpll_1_c0
areset_conduit
Conduit
locked_conduit
Conduit
nios2_gen2_0
Nios II Processor
clk
Clock Input
altpll_0_c0
reset
Reset Input
[clk]
data_master
Avalon Memory Mapped Master
instruction_master
Avalon Memory Mapped Master
irq
Interrupt Receiver
IRQ 0
IRQ 31
debug_reset_requ...
Reset Output
debug_mem_slave
Avalon Memory Mapped Slave
0x0000_0800
0x0000_0fff
custom_instructio...
Custom Instruction Master
onchip_memory2_0
On-Chip Memory (RAM or ROM)
clk1
Clock Input
altpll_0_c0
s1
Avalon Memory Mapped Slave
[clk1]
0x0000_4000
0x0000_7fff
reset1
Reset Input
onchip_flash_0
Altera On-Chip Flash
nreset
Reset Input
data
Avalon Memory Mapped Slave
0x0020_0000
0x0035_ffff
csr
Avalon Memory Mapped Slave
0x0040_0000
0x0040_0007
mm_clock_crossi...
Avalon-MM Clock Crossing Bridge
m0_clk
Clock Input
altpll_0_c2
m0_reset
Reset Input
[m0_clk]
s0_clk
Clock Input
altpll_0_c0
s0_reset
Reset Input
[s0_clk]
s0
Avalon Memory Mapped Slave
0x0000_2000
0x0000_2fff
m0
Avalon Memory Mapped Master
led_pio
PIO (Parallel I/O)
Current filter:



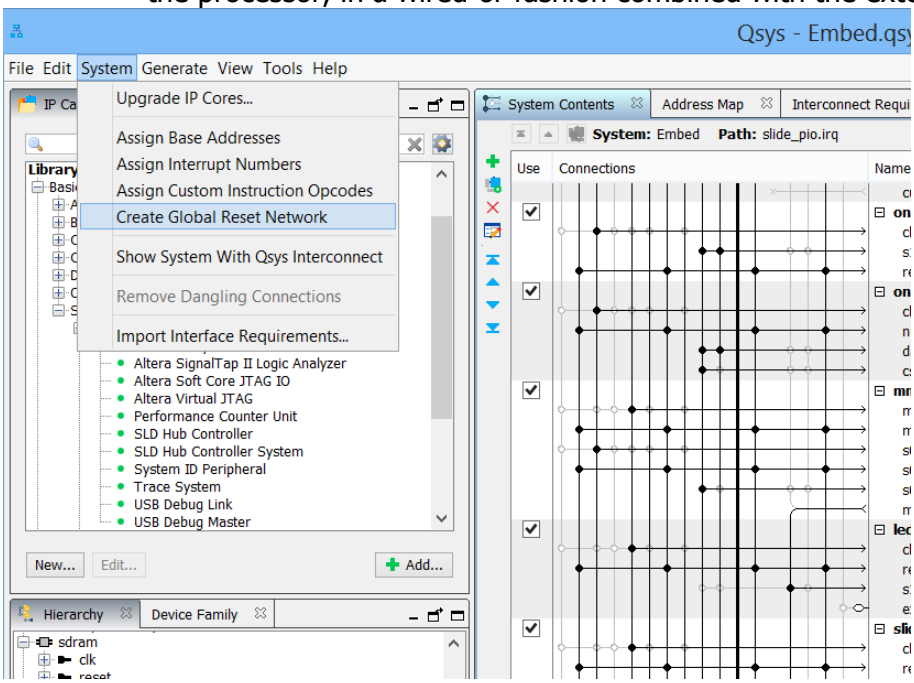
20) Confirm System Configuration.

1. **Clocks**: Only the alt_pll_0 should have clk_0, the external clock as an input in the clock column. Only the alt_pll_1, the ADC module, and JTAG Bridge should have clk_1, the external clock as an input in the clock column. Only the nios2 cpu, the SDRAM Controller, the SPI port, mm clock crossing bridge and onchip memories should use Alt PLL c0. All other components should use Alt PLL c2.
2. **Interrupt Request Lines**, or IRQs: In the IRQ column on the right side, click on the circle by the timer IRQ to connect it to the processor, and label it 0. Then connect the JTAG UART IRQ with label 1, the adc IRQ with label 2, the SPI IRQ with label 3 and finally the slide PIO IRQ with label 4 for the priority.
3. **Create a Global Reset Network.**

   Qsys provides the flexibility to connect individual resets to each of the components in the system. For our system, we simply want all components tied together. Qsys provides an easy

menu item to do exactly this and will save us from having to manually connect the reset inputs to each component.

- From the **System** menu, choose **Create Global Reset Network**. The tool will automatically connect all the resets in the system together. This reset is generated by the processor, in a wired-or fashion combined with the external reset.



4. **Set Base Addresses and Interrupt Priorities**.

   Qsys provides two easy menu items that help clean up address map issues and interrupt priority issues. Although we can enter them manually, we could have skipped this step and let Qsys automatically assign them. Either will work.

   - From the **System** menu, choose **Assign Base Addresses**. The tool will assign appropriate base addresses for the components by taking their widths into consideration.

Component	Manual Base address	Automatic base address
Alt PLL 0	0x0000	0x0940_9220
Alt PLL 1	0x0000	0x0940_9230
NIOS2	0x0000_0800	0x0940_8800
Onchip RAM	0x0000_4000	0x0940_4000
Onchip FLASH	0x0020_0000	0x0920_0000 0x0940_9248
MM Bridge	0x0000_2000	0x0800_0000

LED PIO	0x0050	0x0020
Slide PIO	0x0040	0x0030
Timer	0x0020	0x0000
SDRAM Controller	0x0400_0000	0x0400_0000
SPI Port	0x0060	0x0940_9200
ADC	0x0080 and 0x0200	0x0940_9240 and 0x0940_9000
JTAG UART	0x0090	0x0048
SysID	0x00A0	0x0040

You could use undo (Control Z) to revert to the manual assignments, however I chose to keep the auto-generated ones. This eliminates the last error messages.

- From the **System** menu, choose **Assign Interrupt Numbers**. The tool will update the IRQ mapping accordingly.

5. **Confirm Nios II Boot Configuration.**

In the event of a reset, the software must begin executing from a predefined memory location. This is set by setting the reset vector. Similarly when a software exception event occurs the software must jump to a pre-defined location where the exception handling software resides. This location is set by setting the exception vector.

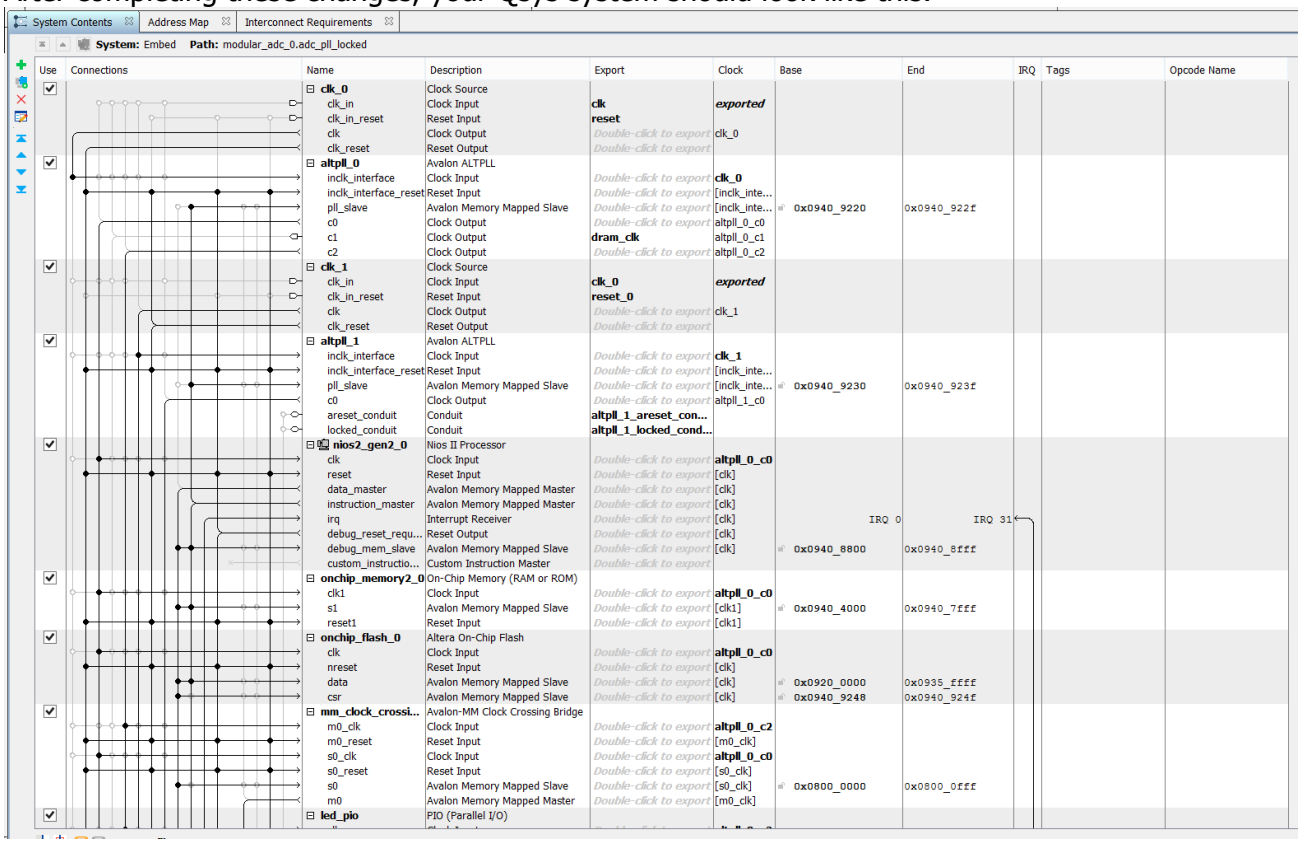
- Double click on the nios2 component to launch the "Nios II Gen2 Processor (Preview)" Parameter Settings GUI.
- Click on the Vectors tab.
- Set the Reset Vector to point to the onchip ram with an **offset** of 0x0. When the Nios II processor comes out of reset, it will begin executing software at this memory location.
- Set the Exception Vector to point to the onchip ram memory with an **offset** of 0x20. When the Nios II processor experiences software exceptions or interrupts, it will jump to this location in memory.

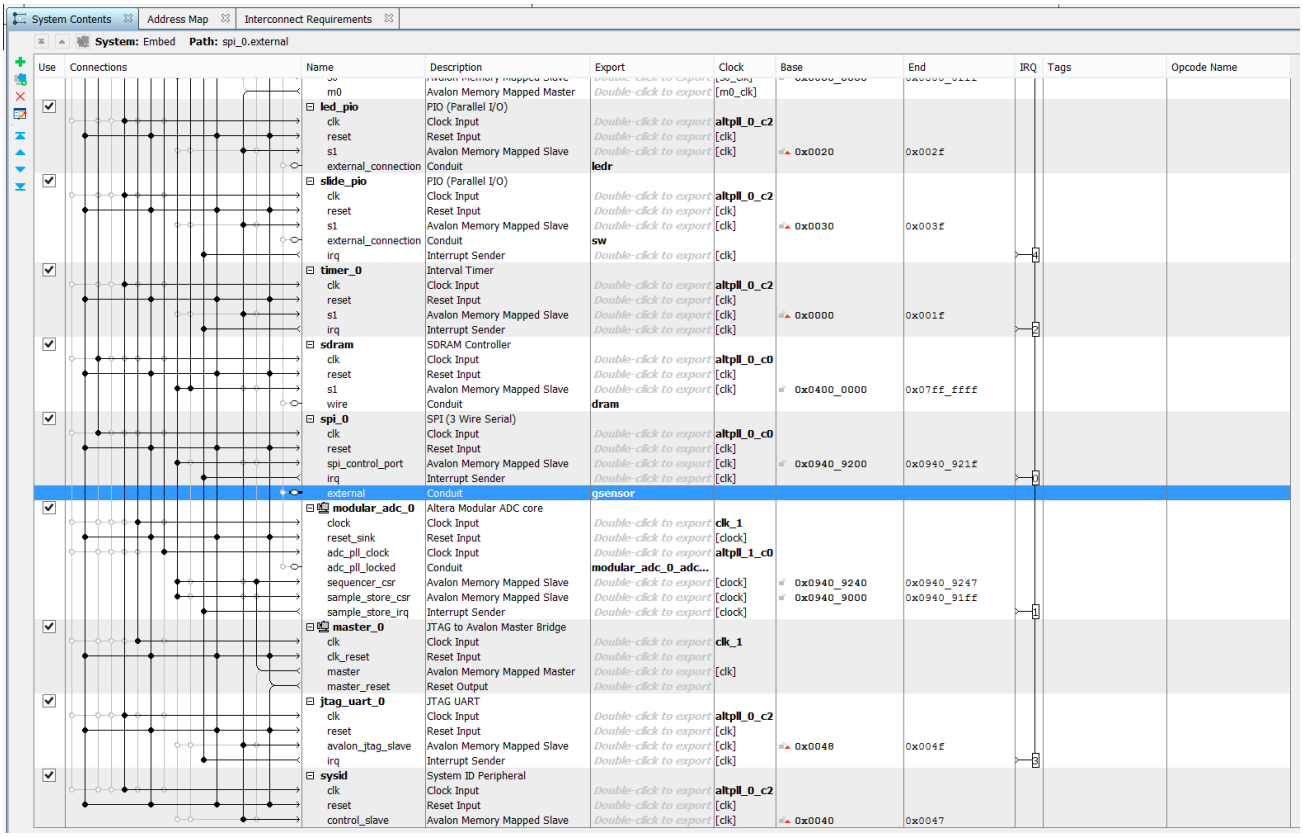
6. **Export Signals to the Top Level.**

To expose signals to the top level, they need to be exported. Several peripherals contain connections to pins on our device, which means that we need to "export" those connections out of Qsys. Qsys provides an Export column in which we can select which interfaces will be available for connections outside of this Qsys block.
We need to export the following interfaces by double-clicking in the Export column and then renaming the export name to match exactly so that the names of those signals match the top-level Verilog or VHDL file provided with this project.

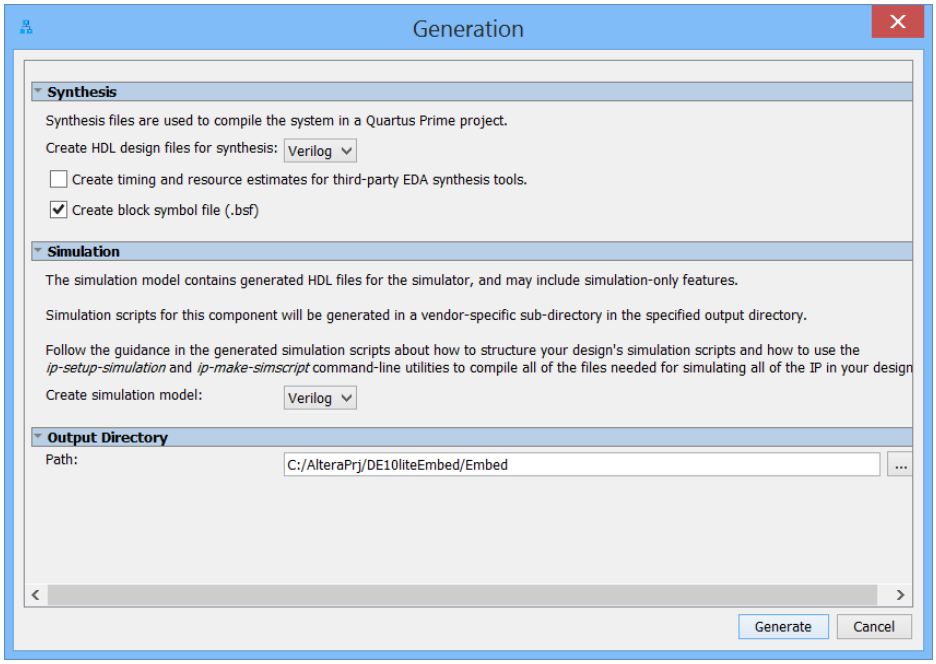
Component	Port to be exported	Export Name
alt_pll_0	c1	DRAM_CLK
alt_pll_1	areset_conduit locked_conduit	altpll_1_areset_conduit altpll_1_locked_conduit
sdram	Wire	DRAM
spi_0	External	GSENSOR
Led_pio	External_connection	LEDR
Slide_pio	External_connection	SW
modular_adc_0	Conduit	modular_adc_0_adc_pll_locked

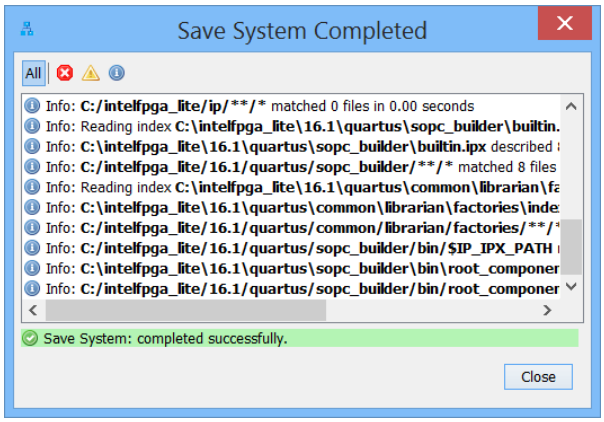
After completing these changes, your Qsys system should look like this:

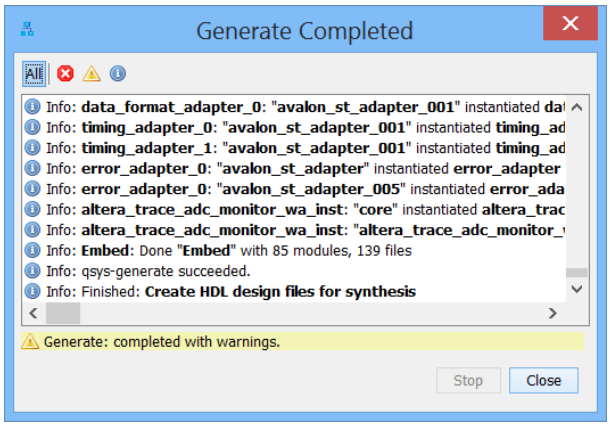




21) Click Generate HDL. Be sure the Create block symbol file box is checked.







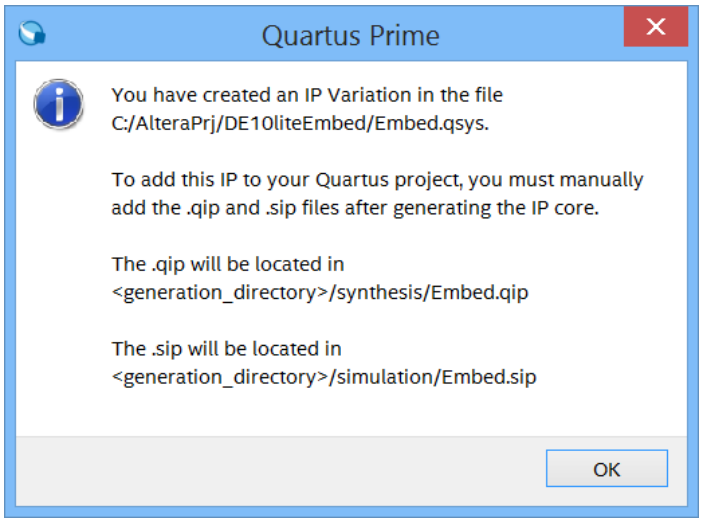
CONGRATULATIONS!! You have just built your custom processor Qsys system!

Now we can complete the Quartus project by adding the generated Qsys system to the top-level entity. We then Compile in the Quartus software to perform analysis, synthesis, fitting, place and route as well as timing analysis. At the end of the compilation, an FPGA image or SRAM object file (*.SOF) will be generated. The FPGA image can be downloaded to the DE10-Lite, at which point the on-board FPGA will function as a processor custom-made for your application.

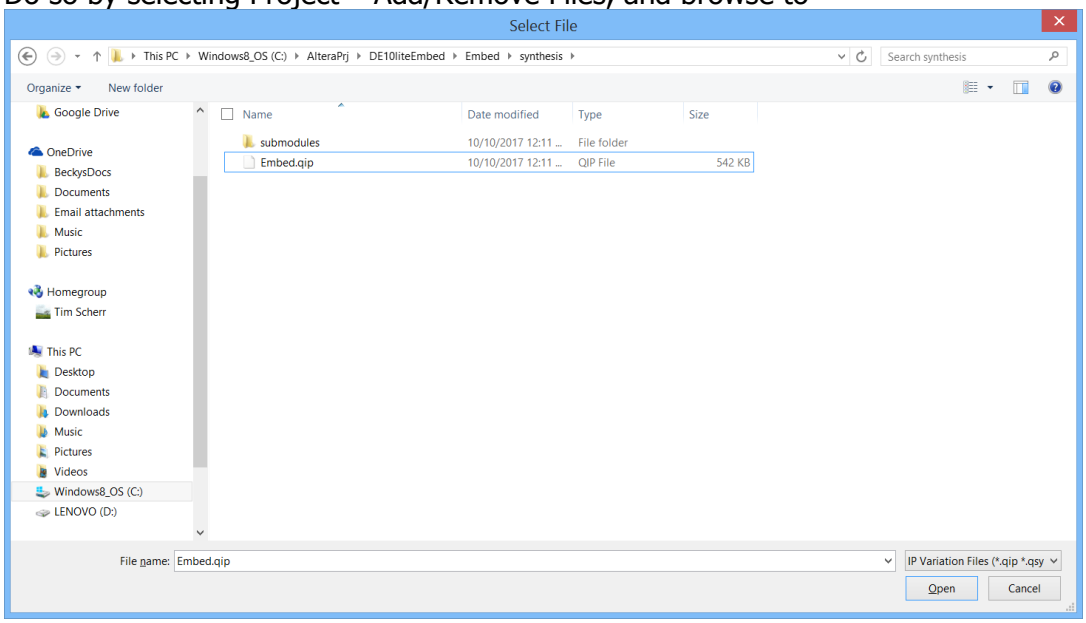
An IP variation file is a file with extension of *.qip that is generated by Qsys systems (or also by standalone IP Catalog block). The *.qip file keeps track of the generated files so that your Quartus project can know what files are needed for FPGA compilation.

22) Save the Qsys system, and exit Qsys.

23) Back in Quartus, you may see a message to add the Embed.qip and Embed.sip files:



Do so by selecting Project – Add/Remove Files, and browse to

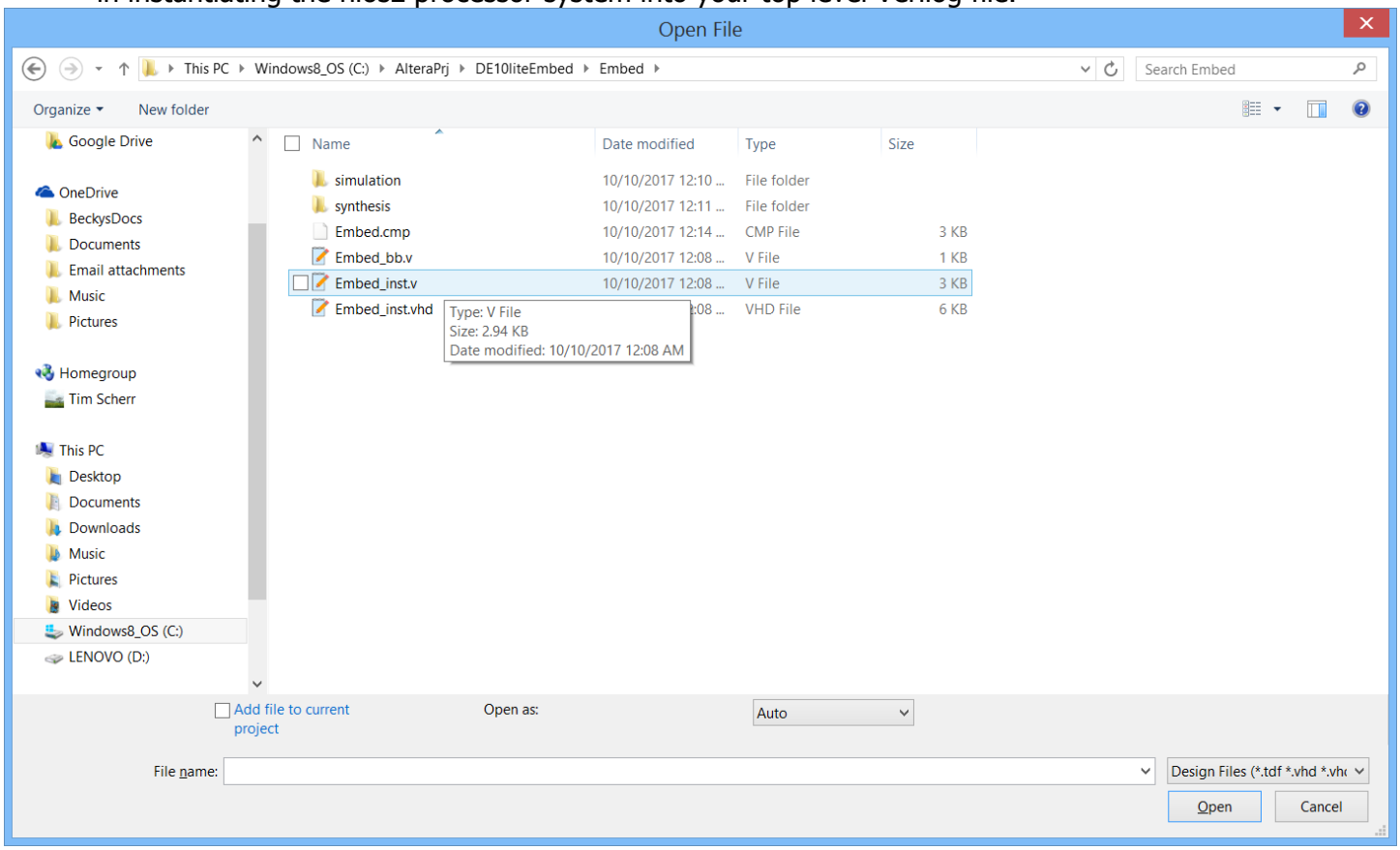


Click open, then apply and OK to add the .qip file.

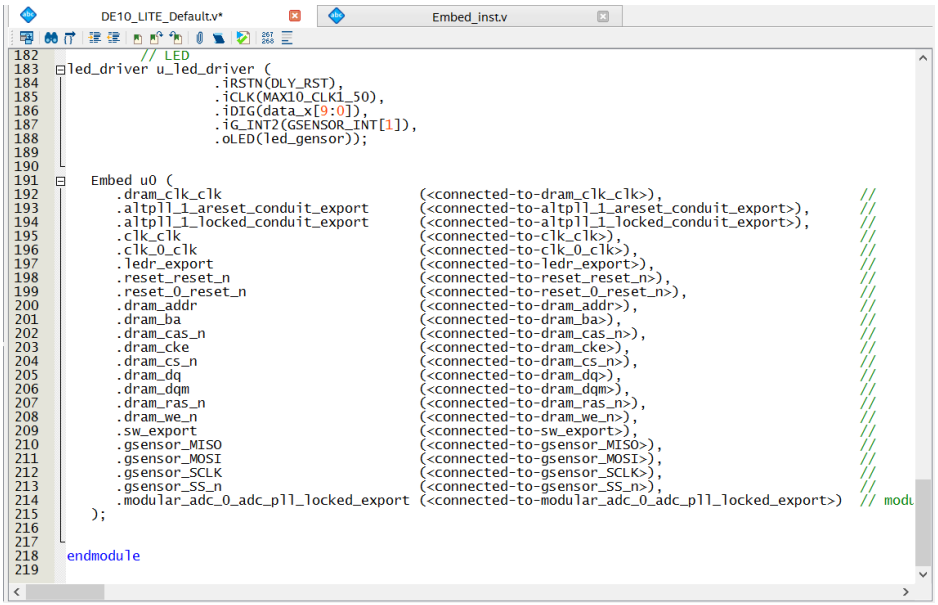
4. INSTANTIATE THE QSYS DESIGN INTO THE TOP-LEVEL VERILOG FILE

The design uses several Verilog files which each define a design entity. The different design entities will need to be connected together to create our FPGA design. While the Verilog design entities can be connected together with Verilog code, another option exists in the Quartus software. Although is possible to create symbols for Verilog files and then be connected together using the Quartus schematic editor, this time we will use instantiation of the Qsys system into the top level Verilog file.

1) Now from the File Menu select Open, double click on Embed, and then select Embed_inst.v. Click on Open. This file should now be open in the main window; it was created by Qsys to aid in instantiating the nios2 processor system into your top level Verilog file.



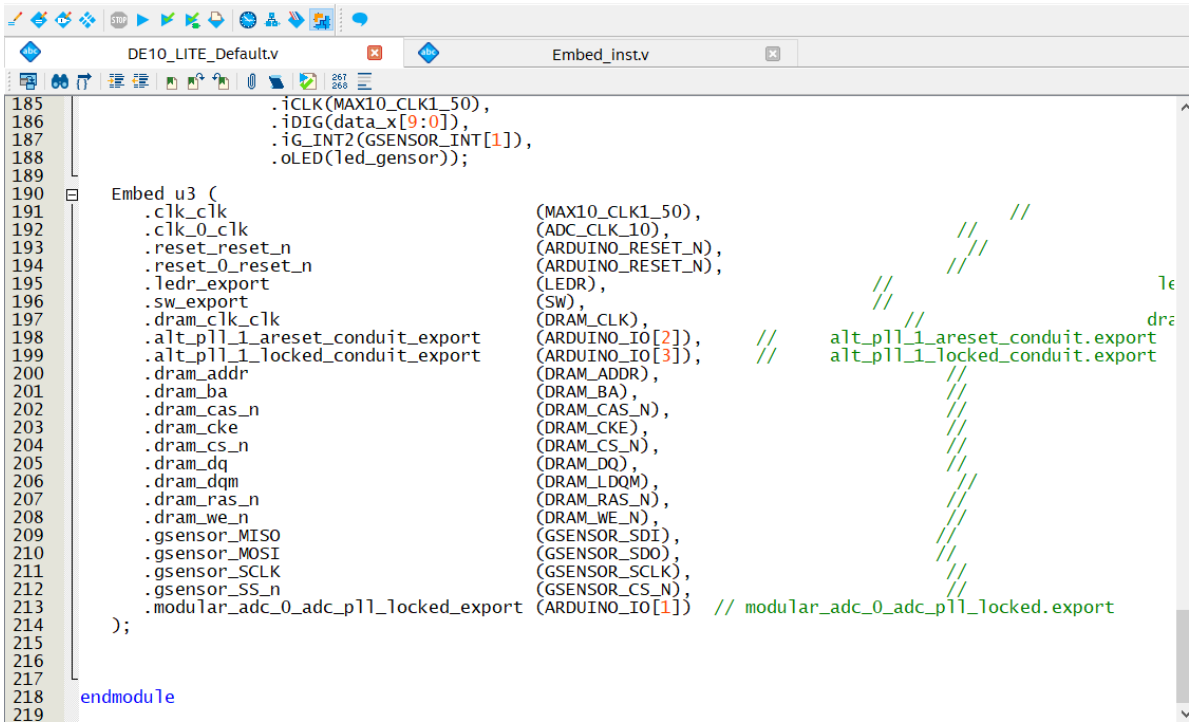
2) Click in this file, and hit control A and control C. Go back to DE10_LITE_Default.v, and paste this in before the endmodule. Now you will need to complete the instantiation by specifying the port signal connections. You will need to change Embed_u0 to u3 or another number to avoid conflicts with other instantiated units.



3) Replace all the connected-to parenthetic expressions as follows:

From	To
<connected-to-dram_clk_clk>	DRAM_CLK
<connected-to-clk_clk>	MAX10_CLK1_50
<connected-to-clk_0_clk>	ADC_CLK_10
<connected-to-ledr_export>	LEDR
<connected-to-reset_reset_n>	ARDUINO_RESET_N
<connected-to-reset_0_reset_n>	ARDUINO_RESET_N
<connected-to-dram_addr>	DRAM_ADDR
<connected-to-dram_ba>	DRAM_BA
<connected-to-dram_cas_n>	DRAM_CAS_N
<connected-to-cke>	DRAM_CKE
<connected-to-cs_n>	DRAM_CS_N
<connected-to-dq>	DRAM_DQ
<connected-to-dqm>	DRAM_LDQM
<connected-to-ras_n>	DRAM_RAS_N
<connected-to-we_n>	DRAM_WE_N
connected-to-sw_export	SW
connected-to-gsensor_MISO	GSENSOR_SDI
connected-to-gsensor_MOSI	GSENSOR_SDO
connected-to-gsensor_SCLK	GSENSOR_SCLK
connected-to-gsensor_SS_n	GSENSOR_CS_N
connected-to-modular_adc_0_adc_pll_locked_export	ARDUINO_IO[1]
connected-to-altpll_1_areset_conduit_export	ARDUINO_IO[2]
connected-to-altpll_1_locked_conduit_export	ARDUINO_IO[3]

When you are done you should have something like:



4) You will also need to comment out the LEDR assignment statement and spi_ee_config block. Add an assign DRAM_UDQM = DRAM_LDQM;

5) Go to the **File** menu and select **Save** to save the changes you have made to the top-level Verilog file. Run an Analysis and Elaboration (or Analysis & Synthesis).

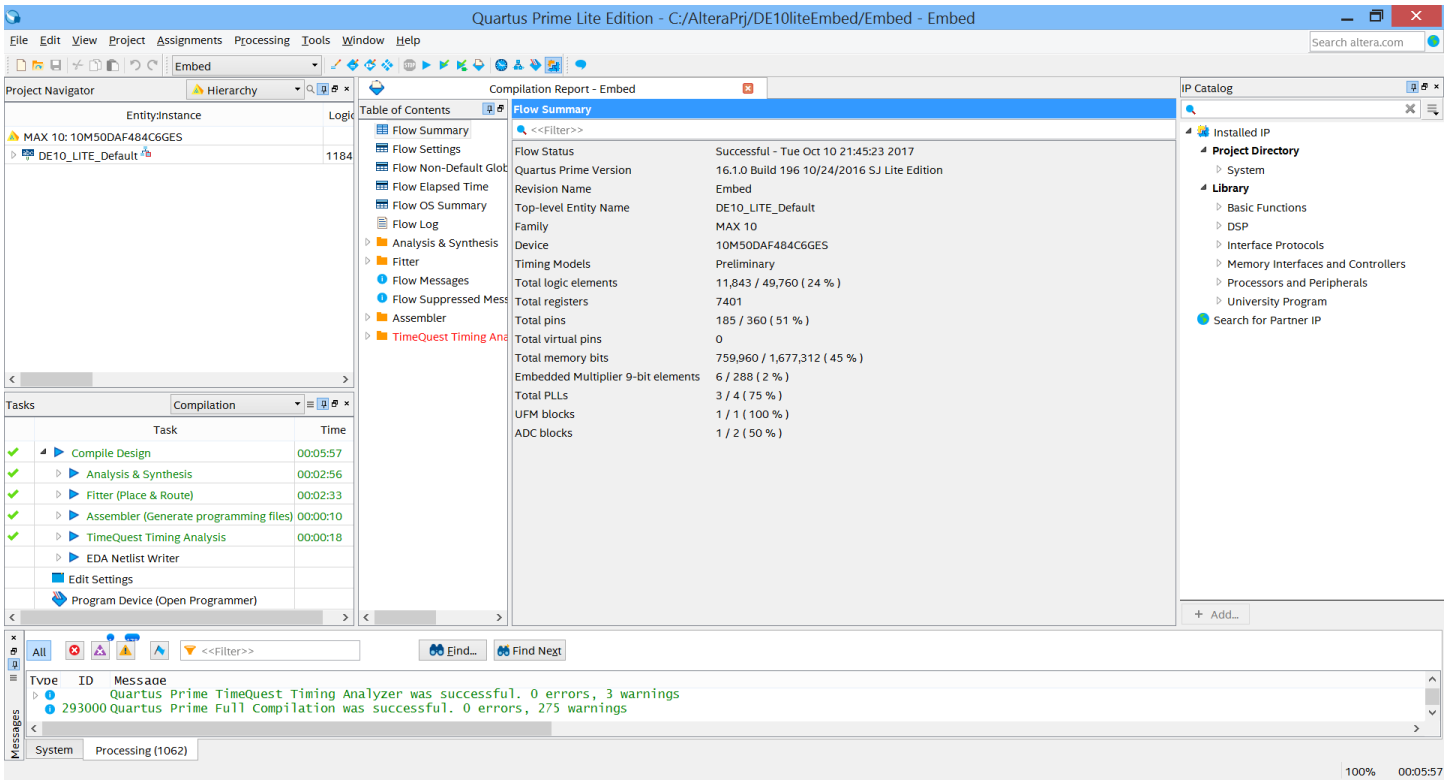
CONGRATULATIONS!!

You've completed the design entry of your custom processor system!

5. PLACING AND ROUTING THE DESIGN

Section Objective: In this section you will do a full compilation on your Embedded System design. An sdc file was provided and included in the project, and Qsys also generates constraints, so we have not entered timing constraints as one normally would.

1. Double click on Compile Design in the Task window. You should see a result something like this, with all the green checkmarks in the Task window:



2. When compilation complete, look at the Flow Messages. Note that there are tabs at the top of the messages window that allow you to filter by message type.
3. Look at the TimeQuest Timing Analyzer, Slow 1200mV 85C Model, Fmax Summary to determine the Fmax. Not that the TimeQuest result is in red because we have not yet constrained all the ports. This is not a concern for now.
4. Look at the Flow summary to determine the total registers (Flip-Flops) and % utilization of logic elements.

Nice Going!!

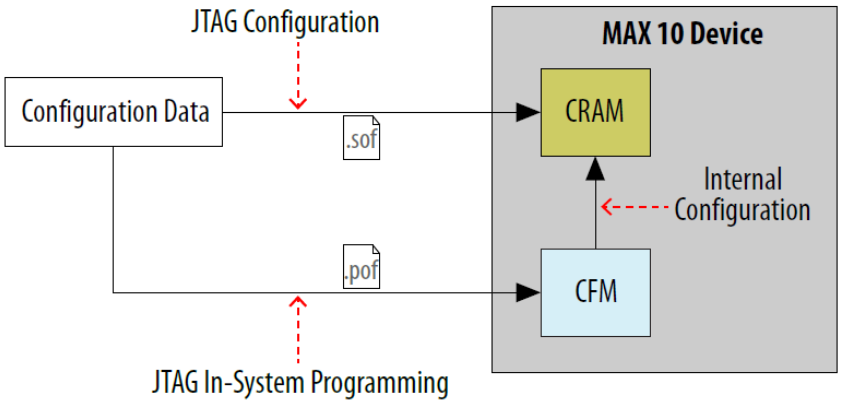
You have just placed and routed your FPGA design.

6. PROGRAMMING THE HARDWARE AND TESTING THE DESIGN

Section Objective: In this section you will learn how to generate a programming file that can be handed off to production.

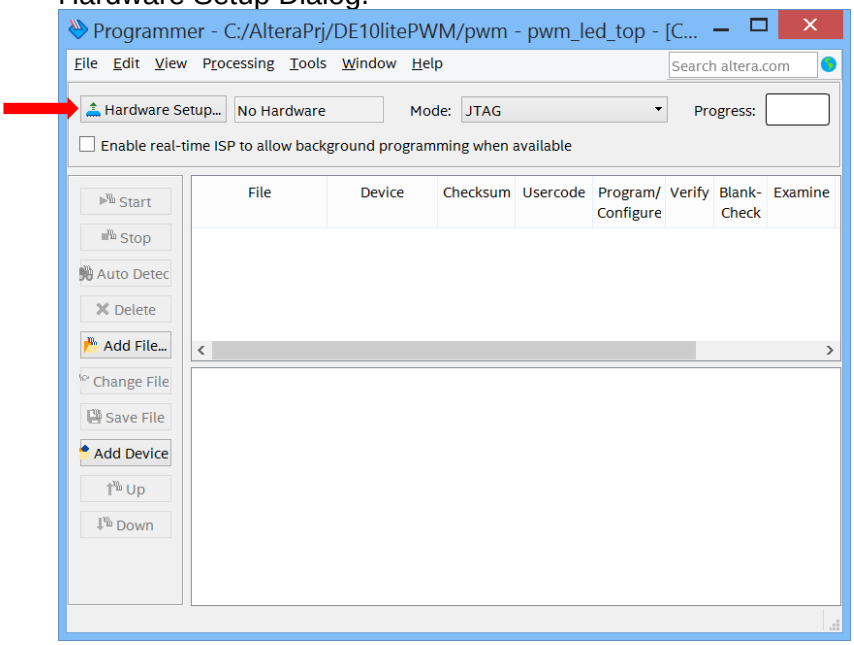
The MAX10 is unique to Altera in that it has internal FLASH memory for configuration, and so there are 2 ways to program it. One is with JTAG as with other FPGAs using a .sof file directly to the SRAM configuration cells, and the other also uses JTAG but programs the configuration flash memory, which is transferred to the SRAM configuration cells on power up. JTAG programming requires a programming cable, like a USB Blaster II or Ethernet Blaster II.

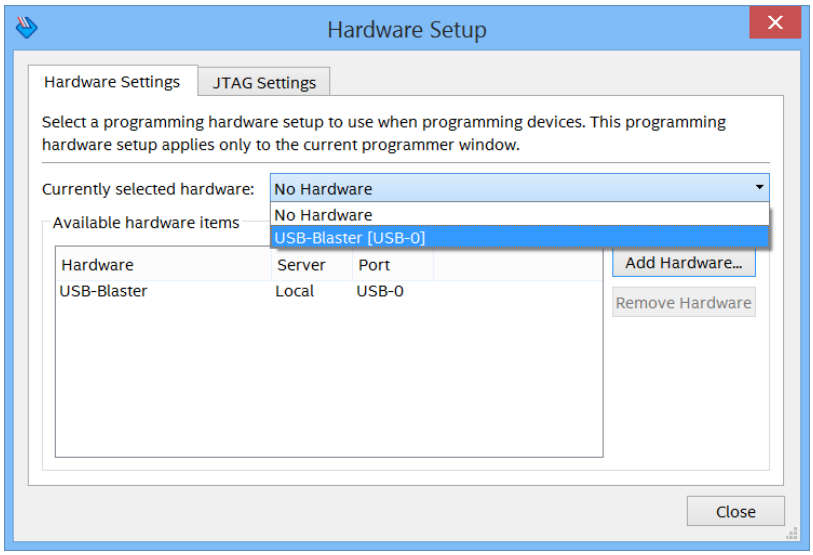
Figure 29: Overview of JTAG Configuration and Internal Configuration for MAX 10 Devices



PROGRAMMING THE DE10-LITE

After Compilation, do the following to program the board:
1) Connect the USB cable to your DE10-Lite kit.
2) Plug the other end of the USB cable to a USB port of your computer.
3) Launch the Quartus Programmer, via the icon or through the Tools menu (**Tools -> Programmer**)
4) Setup the programming hardware. To do this, click the hardware setup button in the upper left corner of the programmer, and select the hardware you want to use. Choose USB Blaster in the Hardware Setup Dialog.

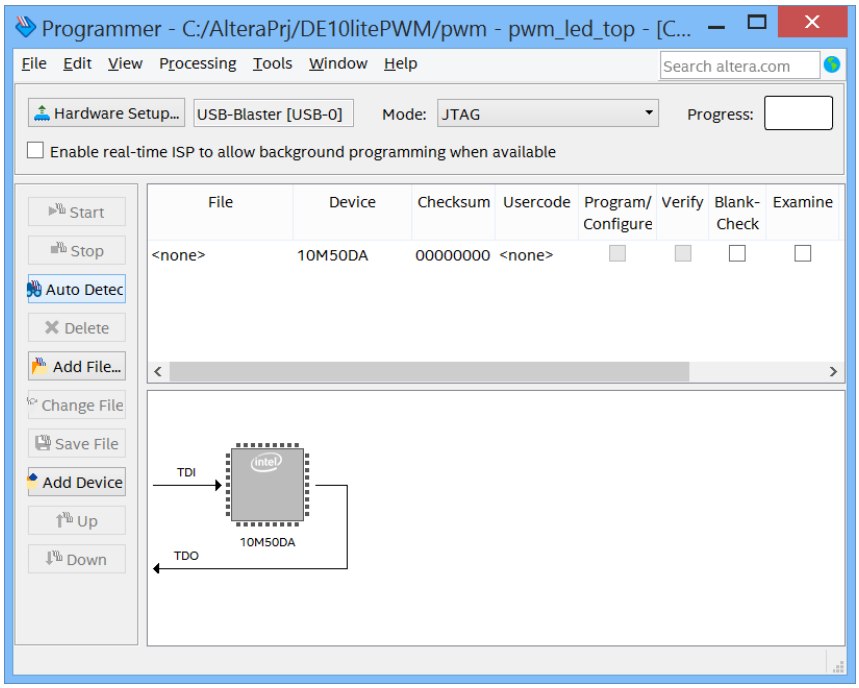




Close the Dialog Box and your setup should appear as this:

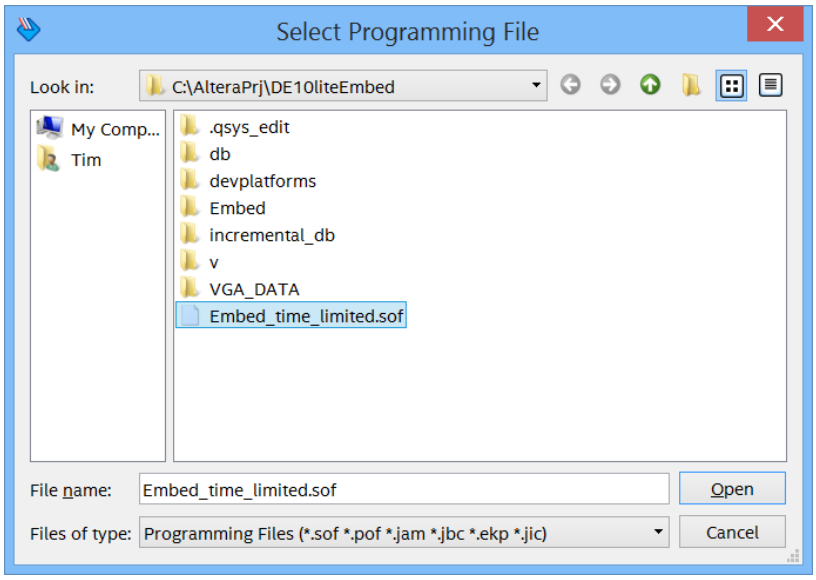
A programming device must have drivers installed and be detected correctly to be listed. Next select the programming mode – the most common is JTAG. The JTAG chain can consist of both non-Altera and Altera devices.

Once the hardware is setup, a toolbar in the programmer provides all the commands needed to control the programming of devices. For example, the order of programming devices on the chain can be arranged. Other common operations include Auto detect, in which the chain is scanned and devices found is reported, and change file, which selects a new file to program into the selected target device. Clicking on Auto Detect makes the programmer show the device chain:



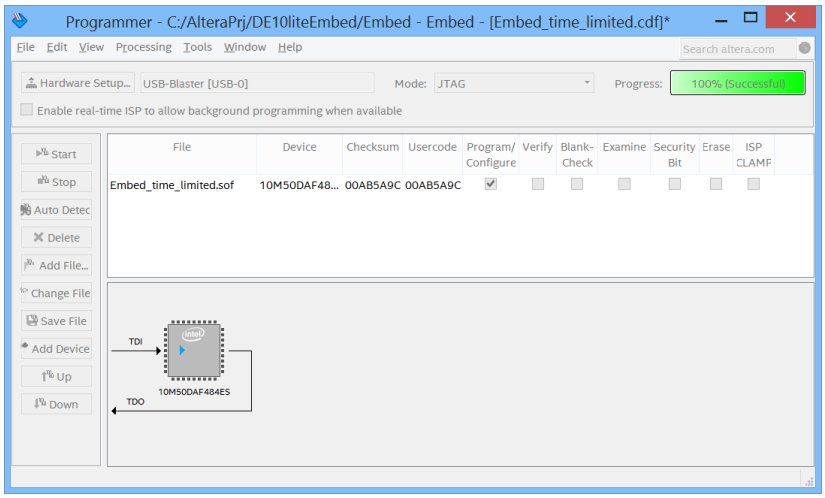
You can also verify a device after it has been programmed, or blank check a device, erase a device, or set a security bit if available.

5) To select the programming file, click Add File, in this case Embed.sof.



Click Open. Back in the programming window, you may have to delete any other entries that are listed.

6) When all the devices are defined and options set correctly, click on Start in the upper left. The progress bar shows the status of the programming and the messages windows provides detailed status, and information about any errors that may occur. When the progress bar reaches 100%, the programming is complete.



Unless you have a license for the Nios II processor, you will obtain a message about a time-limited megafunction. This message indicates that since you do not have a license for the Nios II processor, operation will occur in "OpenCore Plus" evaluation mode. OpenCore Plus allows for tethered or time-limited evaluation prior to purchasing a license.

- ***PLEASE NOTE:*** *OpenCore Plus evaluation mode prevents us from programming the flash of the MAX 10 device and evaluation must occur by programming the FPGA SRAM from the Quartus Programmer each time the kit is powered up using the .sof file.*

If you are running with a time-limited SOF file, then a window pops up on the Quartus Programmer. Just leave this up and ***do not press "Cancel"*** until you are finished using the hardware design that you just downloaded. Closing this dialog will halt the Nios II CPU inside the FPGA.

Preparation for Software Build

The Nios II software build tool requires certain component names to be exact, different than what has been created thus far. To be compliant with the EDS, do the following:

1) In Qsys, rename onchip_memory2_0 to onchip_ram.
2) Also rename jtag_uart_0 to jtag_uart.
3) Double-click on the nios2 component, and on the Vectors tab, change the Reset vector memory to onchip_ram.s1, and the Exception Vector memory to onchip_ram.s1.
4) Save the Qsys system.
5) Select Generate HDL, and then click Generate as before.

6) In Quartus, Compile the design as in section 4. Check to be certain you do not have hold or setup violations. If this is the case, you may have to adjust compiler settings. You should not have to run Timequest.
7) Open the programmer, and program the FPGA with the Embed_time_limited.sof file.

CONGRATULATIONS!!

You have completed Module 3!

III. DELIVERABLES

Deliverables include:

1. Recorded Observations, Test Data, and Images

Include observations recorded in a lab notebook, test data taken, screenshots and any digital pictures of the proceedings. You will need these in order to answer the quiz questions. Be sure to include your lab notebook in your submission.

2. FPGA Project directory zipped for each Part of the Module

Submit the **DE10liteEmbed** project you created. For Module 3, starting with the top level of each part, zip up the entire directory including subfolders, and submit the zip files. This will allow us to replicate your work and help provide feedback on any errors you encounter. With each submission include a ReadMe.txt if appropriate to describe and list the locations of individual file deliverables.

IV. EVALUATION

Any grade awarded pursuant to this project will be based upon deliverables. The following elements will be the primary considerations in evaluating all submitted projects:

1. Reasonable logic utilization and Fmax results within expected bounds.
2. Compilation of hardware designs with no errors.
3. Completion of the project through generation of programming files.
4. Thorough recording of your observations in a lab notebook.

REFERENCES

[1] Intel Altera. (2016). *Max 10 Device Handbook*. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/products/fpga/max-series/max-10/support.html

[2] Intel Altera. (2016). *Qsys System Design Tutorial.* [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/tt/tt_qsys_intro.pdf