BeMicro SDK Embedded System Lab

ARROW ELECTRONICS, INC.

November 2010, Version 10.0 Tutorial

Table of Contents

Overview

What Does It Take To Create Your Own Custom Processor-Based Embedded System?

This lab teaches you how to create a system implemented in programmable logic. You build a processor-based hardware system and run software on it. As the lab progresses, you will see how quick and easy it is to build entire systems using Altera's SOPC Builder to configure and integrate pre-verified IP blocks.

Lab Notes:

Many of the names that the lab asks 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.

Note: This lab guide requires an Arrow Electronics BeMicroSDK FPGA-based MCU Evaluation Board (www.arrow.com/bemicroSDK).

MODULE 1: Getting Started

Module Objective

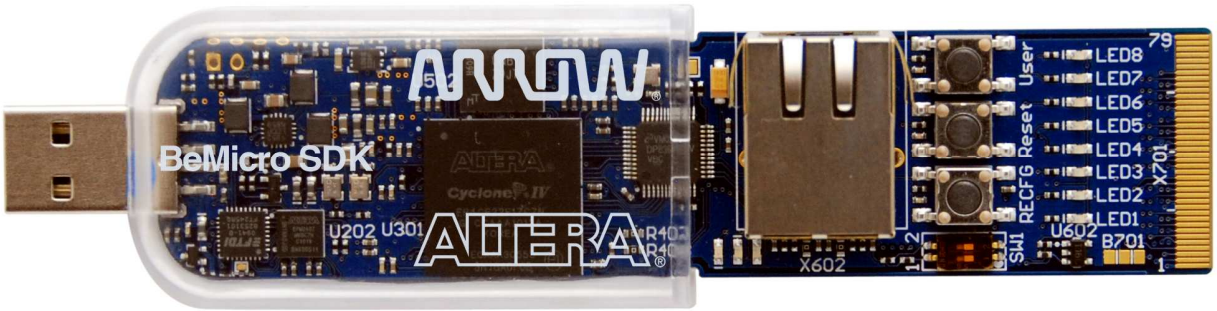
Your first objective is to ensure that you have all of the items needed and to install the tools so that you are ready to create and run your design.

List of required items

- Arrow Electronics BeMicro SDK FPGA-based MCU Evaluation Board
- Design Software (Quartus® II design software v10.0., Nios® II EDS 10.0, Micrium uC/Probe)
- Intel Pentium III or compatible Windows PC, running at 866MHz or faster, with a minimum of 512MB of system memory. **NOTE REGARDING LAB SUPPORTED OPERATING SYSTEMS: 32 bit** versions of **Windows XP**, **Windows Vista** and **Windows 7** are supported by software tools required for this lab. **NO 64 bit** versions are supported
- Lab Design Files

1.1 Acquire the BeMicro SDK Development Board

This development kit can be ordered from http://www.arrow.com/bemicrosdk.



1.2 Install the Altera Design Software

You will need to install **ALL** of the following design software packages:

1) **Quartus II Web Edition design software v10.0** – FPGA synthesis and compilation tool that contains SOPC Builder and the MegaCore IP library with the Nios II processor IP core
2) **Nios II EDS v10.0** – A complete integrated development environment for software development

The Quartus II design software and the Nios II EDS are available via the Altera Complete Design Suite DVD or by downloading from the web.

If you already have both Quartus II and the Nios II EDS installed on your machine, you may skip ahead to Section 1.3 to extract the lab files.

INSTALLING FROM THE DVD-ROM: Please skip ahead to step 4 of the installation instructions.

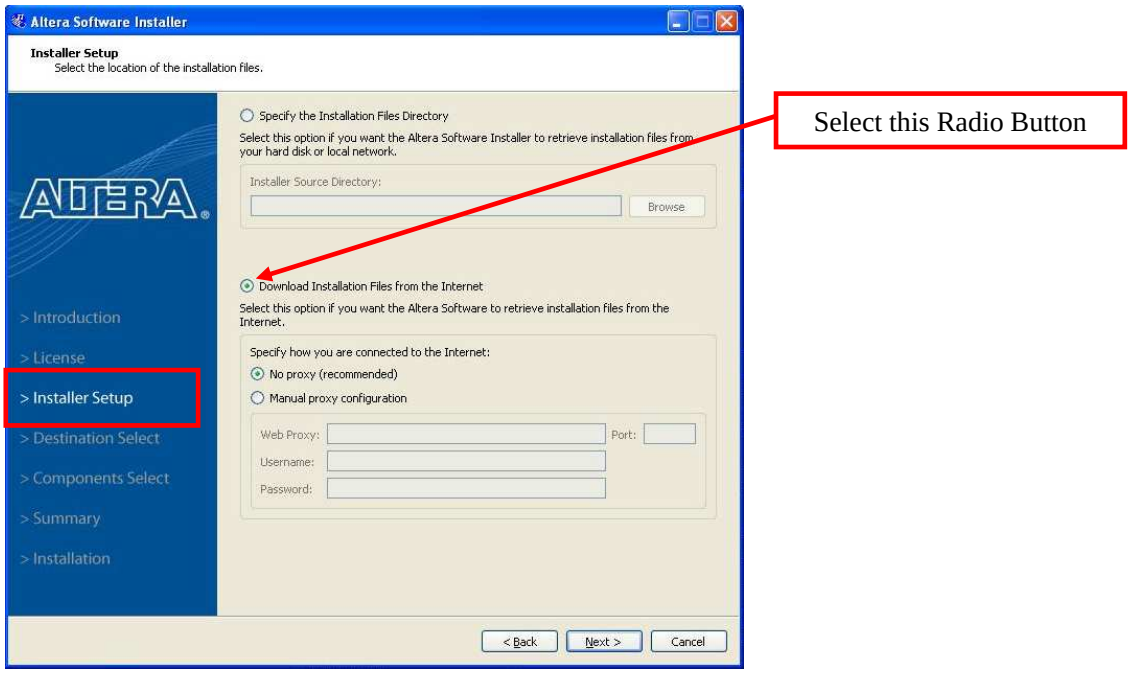
INSTALLING FROM THE WEB: Please follow steps 1 through 4 of the installation instructions.

The Web Edition can be downloaded from the Altera web site. ***Please carefully follow the steps shown below***.

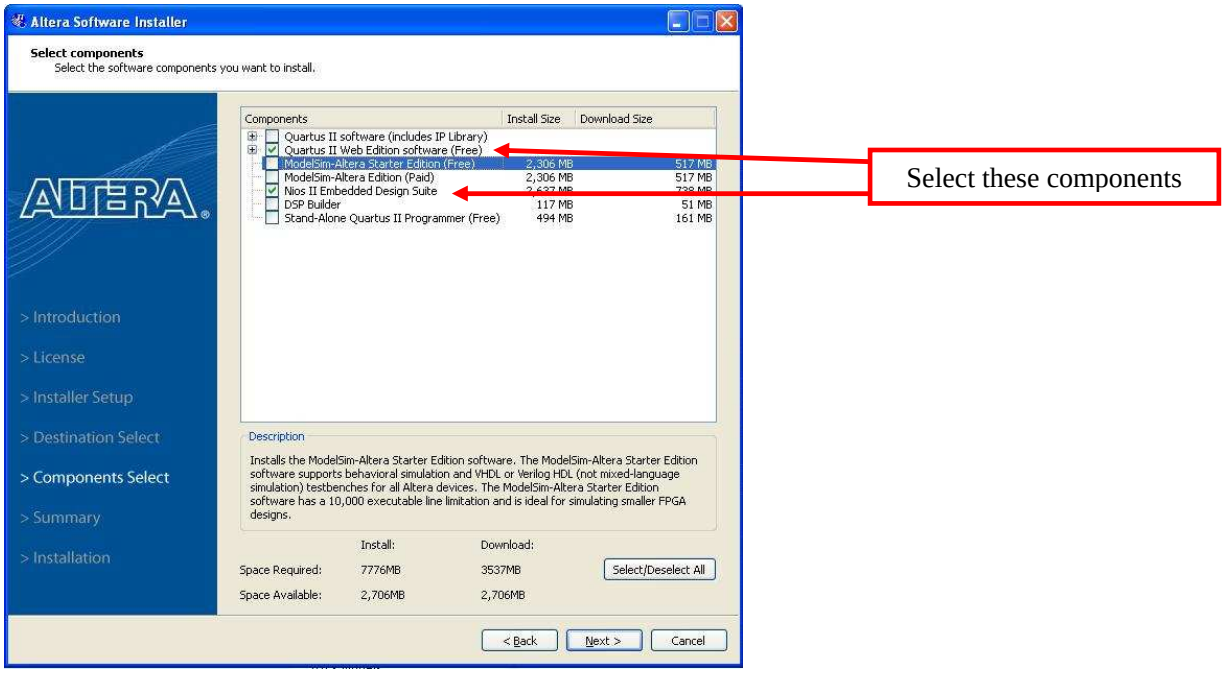
1. Go to the Altera Download web page at https://www.altera.com/download/dnl-index.jsp

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

3. Altera Installer Setup. Run the Altera Installer and Navigate to the ***Installer Setup page***. Select the ***Download Installation Files from the Internet*** radio button



4. Select Components. Select ***Quartus II Software Web Edition*** and ***Nios II Embedded Design Suite components*** for download.



1.3 Extract the BeMicro SDK Installation and Lab Files

Download the BeMicroSDK.zip ZIP archive from the http://www.arrow.com/bemicrosdk web page to a folder on your PC. Make sure that there are ***NO SPACES*** in the directory path.

Note: By installing the software onto your PC you are bound to the license agreement of the software. The complete license agreement can be found in the license_agreement.txt file found in the "<install directory>\driver\" directory. This license agreement, in short, allows you to use the software only in conjunction with Altera FPGA devices purchased from Arrow Electronics or a subsidiary of Arrow.

1.4 Install the USB-Blaster Device Driver

After the Quartus II and Nios II software packages are installed, you can plug the BeMicro SDK board into your USB port. Your Windows PC will find the new hardware and a then the "Found New Hardware Wizard" will come up and request that the driver needs to be installed:

Select "Install from a list or specific location (Advanced)" and continue through the wizard.

In the next dialogue box point the wizard to the drivers which can be found in your Quartus installation directory under "<install directory>\10.0\quartus\drivers\usb-blaster". If Windows presents you with a message that the drivers have not passed Windows Logo testing, please click "Continue Anyway".

If you have trouble with the USB-Blaster installation, please contact your Arrow FAE.

CONGRATULATIONS!!

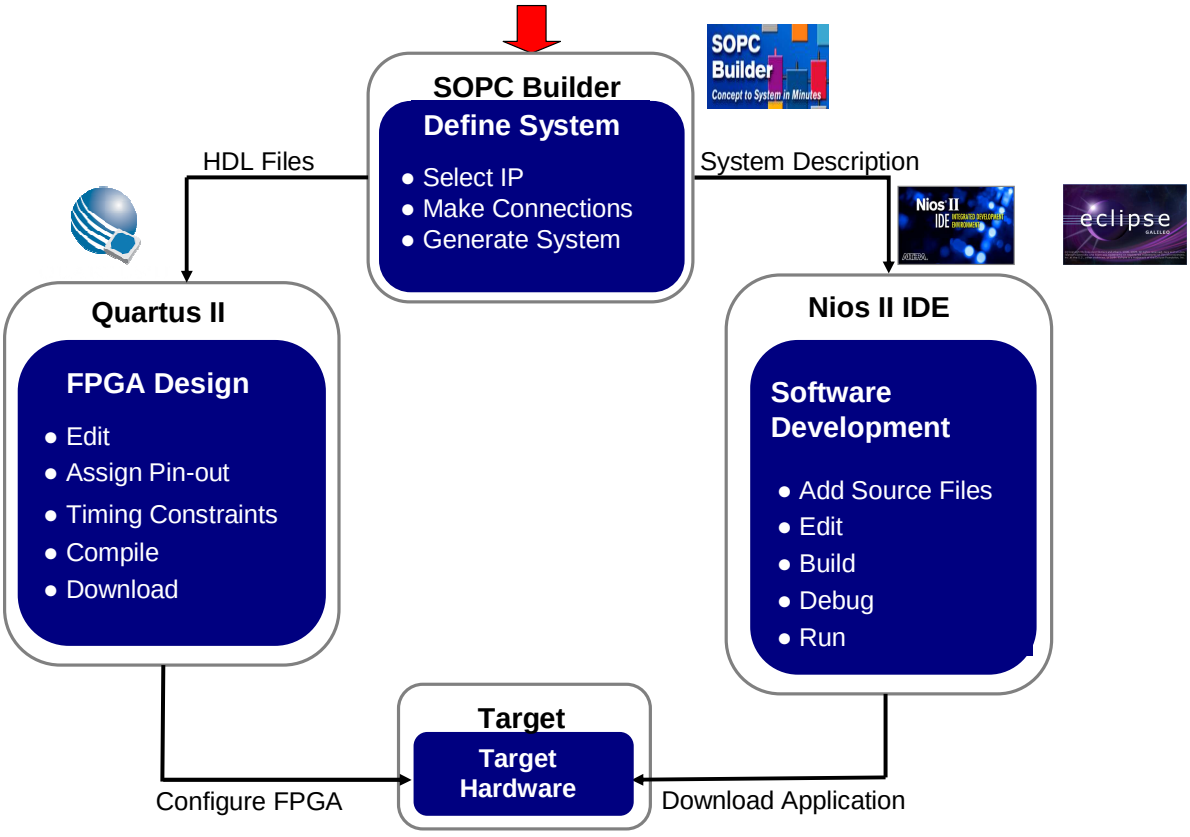
You have just completed all the setup and installation requirements and are now ready to examine the system-level design.

MODULE 2: Examine the System Design

Module Objective

Developing software for an Altera System on a Programmable Chip (SOPC) requires an understanding of the design flow between the SOPC Builder system tool and the Nios II Embedded Development Suite (EDS). Typically, design requirements begin with customer requirements and become inputs to system definition. System definition is hence the first step in the design flow process. For this lab, the system definition and design is complete and an FPGA image derived from that has been flashed into the BeMicro SDK kit. Our objective is to learn how to use the Nios II EDS to build software projects for this system. The objective of this module is to examine the system architecture and development tools that you will be using today.

2.1 Examine the System Tool Flow



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

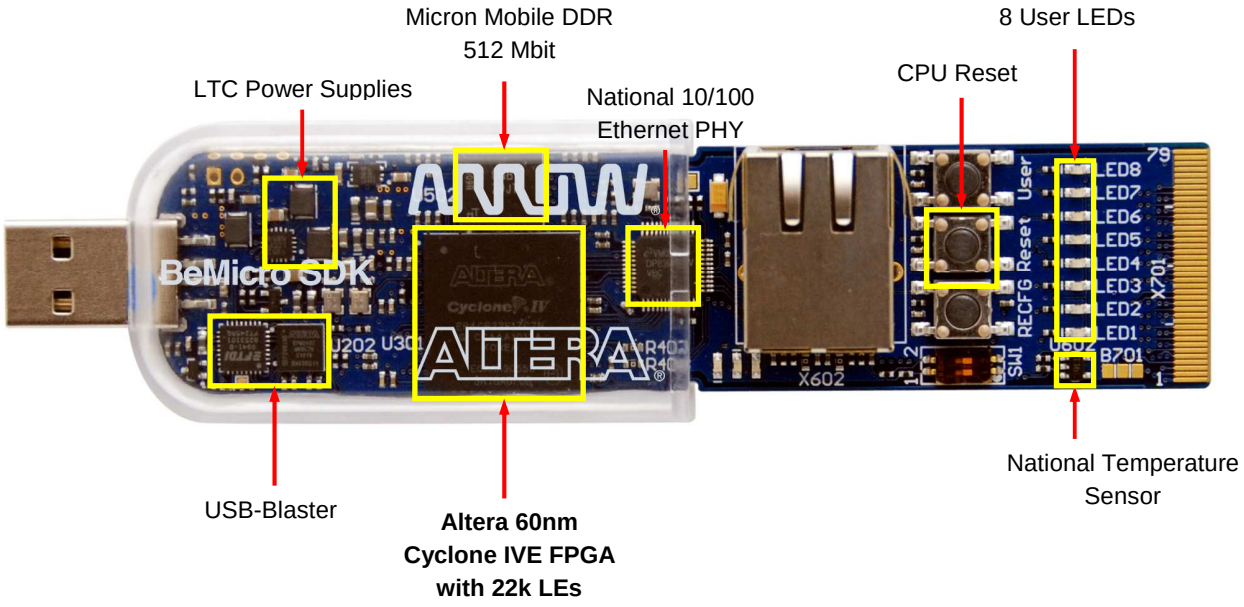
- System description that the Nios II Integrated Development Environment, the software design tool, uses to create a new project for the software application.

- HDL files for the system that are used by the Quartus II FPGA design software to compile and generate the hardware system.

The output of the Hardware Flow is an FPGA image that is used to configure the FPGA. This flow has been completed for you and the FPGA image has been flashed into the BeMicro SDK kit. The output of the Software Flow is an executable from which the Nios II processor executes instructions.

2.2 Examine the BeMicro SDK Kit

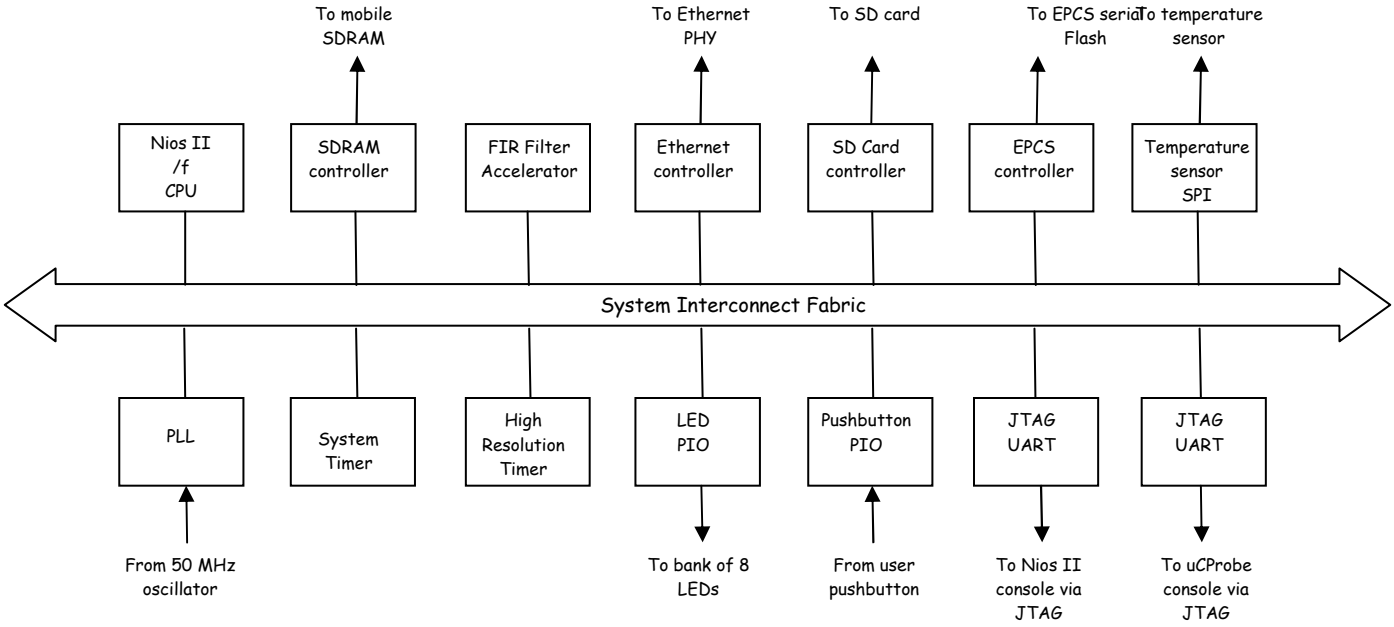
Examine the components on the BeMicro SDK board hardware:



A Micro–SD card connector is located on the reverse side of the board. An Altera serial flash device is also located on the reverse side. This is used to configure the FPGA hardware image and store CPU boot images.

2.3 System Architecture

The BeMicro SDK kit is architected using the components shown in the sketch below:



The system above can be created in SOPC Builder using a standard library of re-useable IP blocks. The System Interconnect Fabric is automatically generated by SOPC Builder 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 a basic embedded system. You will build up a subset of the system shown above.

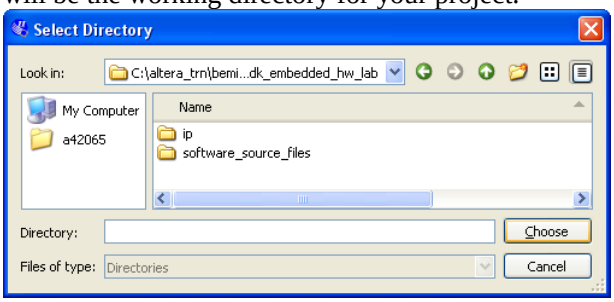
MODULE 3: Set Up the Quartus II Project

In this section, you create a new Quartus II project to contain the SOPC Builder system. The top level is a schematic file, which at this stage is a placeholder containing some minimal reset logic, i.e. a counter that issues a reset to the SOPC system in response to a hard reset.

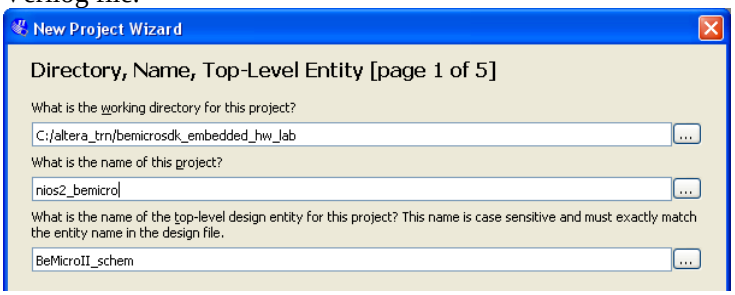
In addition you will specify I/O constraints and settings for this design by executing a Tcl script.

3.1 Create New Quartus II Project

- Launch the **Quartus II 10.0** software from **Start -> All Programs -> Altera**.
- Click on **File -> New Project Wizard.** This will launch the New Project Wizard. An "Introduction" dialogue box may appear. If so, click **Next** to move to the dialogue box for the Name, Directory and Top-Level Entity.
- For the working directory for the project, click the **Browse** button indicated by the "**…**" symbol and navigate to the folder '**bemicrosdk_embedded_hw_lab**' located in the unzipped lab design files. This will be the working directory for your project.



- Name the project "**nios2_bemicro**".
- For the top-level entity click on "**…**" and select "**BeMicroII_schem**" if you prefer to work with a top-level schematic. You may alternatively select "**BeMicroII_ver**" if you prefer to work with a top-level Verilog file.

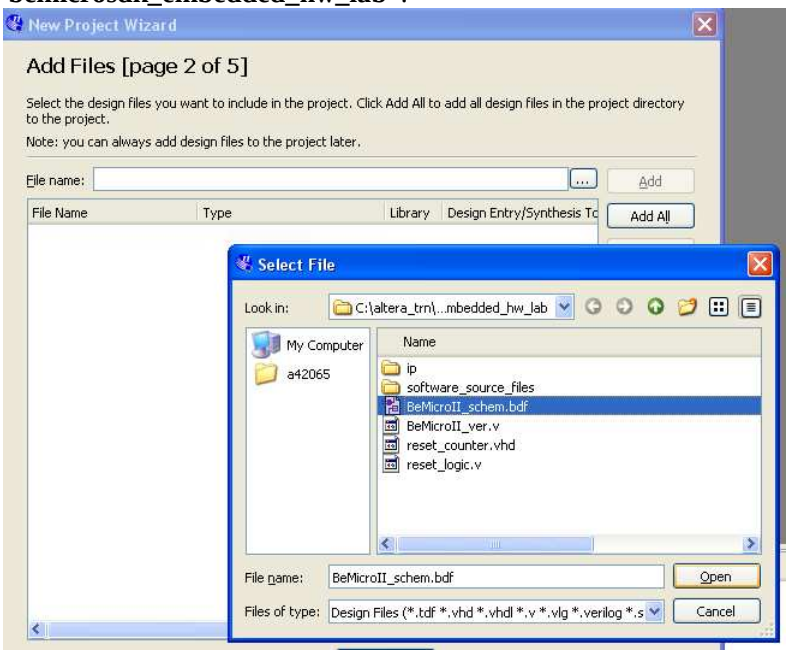


- Click **Next.**

3.2 Add Files to the Project

- In the Wizard window page 2 of 5 you will add files to the new project.

- Click the **Browse** button and navigate to the project directory and **open** the folder entitled "**bemicrosdk_embedded_hw_lab"**.



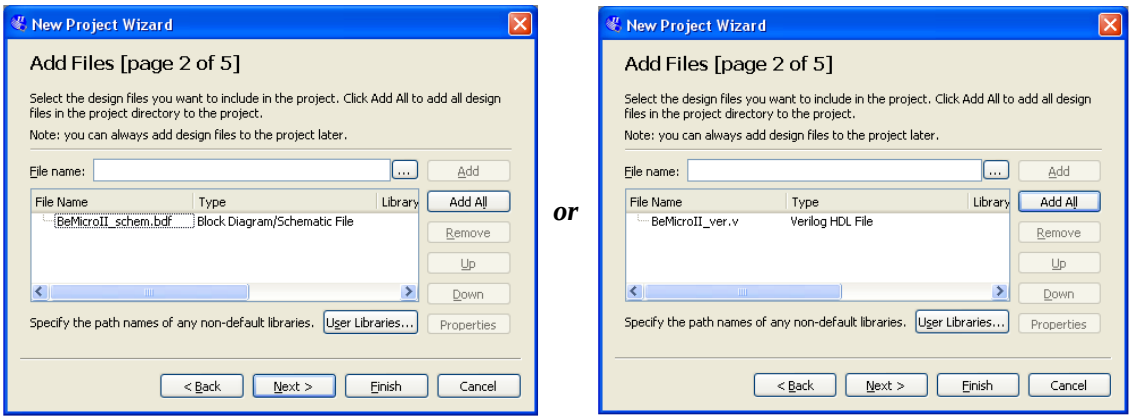
- **Select** and click **Open** to add the appropriate top-level file that you chose in the previous step:
  a. BeMicroII_schem.bdf : This is the top-level schematic entity for the Quartus II Project.

  or

  b. BeMicroII_ver.v : This is an alternate top-level Verilog entity for the Quartus II Project.

- Click **Add** to add the files listing chart.

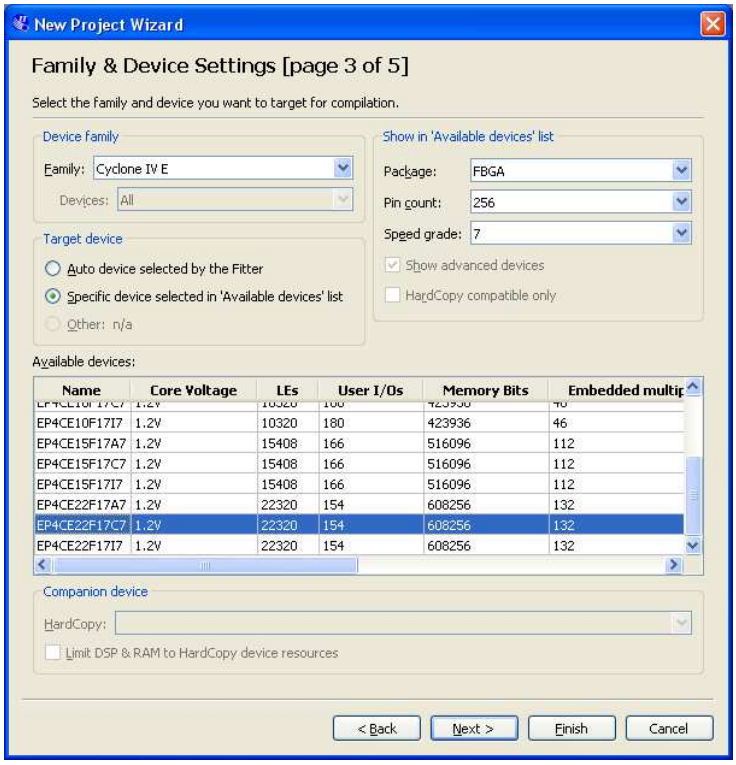
After adding the file, page 2 should show the top-level file that you selected as shown below. Click **Next.**



3.3 Specify Family and Device Settings

In this page you select Cyclone® IV **EP4CE22F17C7** device, which is the device mounted on the BeMicro circuit board.

- First you will need to select **Cyclone IV E** from the Family pulldown.
- You can then use the "Show in 'Available devices' list" option to filter the list of available devices to make selection easier. Select **FBGA** for the package type, **256** for the pin count and **7** for the speed grade. This will give you a shorter list of devices to choose from.
- Select **EP4CE22F17C7** from the "Available devices" list as shown below.



Click **Next**.

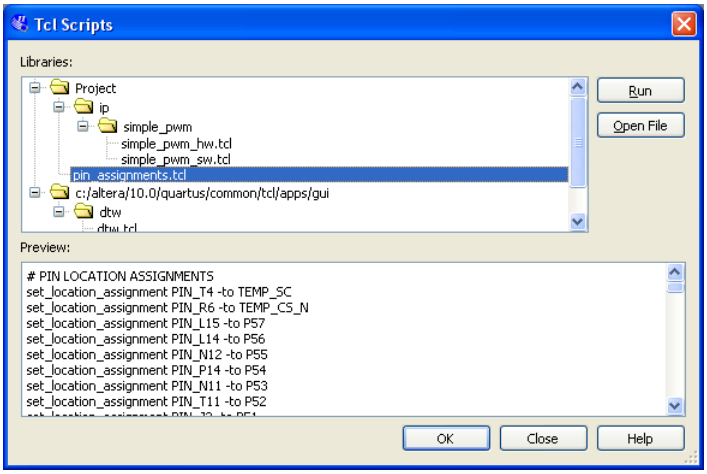
3.4 Select EDA Tool Settings

- Leave <None> selected for all of the options as we will not be using any third party EDA tools. Click **Next**.
- You will see a **Summary** page. Click **Finish**.

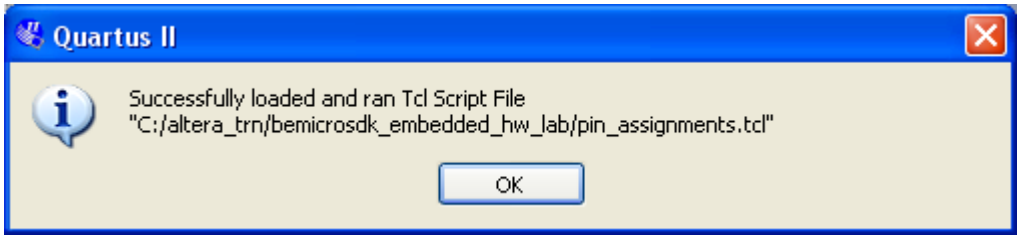
3.5 Execute Setup Script

The I/O pin constraints have been programmed into a Tcl script in order to set up the Quartus II project properly.

- Under the **Tools** menu, select "**Tcl Scripts…**".
- In the **Tcl Scripts** dialog box choose the "**pin_assignments.tcl**" script.
- Click **Run**.



Quartus II will display a message indicating that the Tcl script successfully ran:



Click "**OK**" to dismiss this message.

Click "**Close**" to exit the Tcl Script dialogue box.

CONGRATULATIONS!!

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

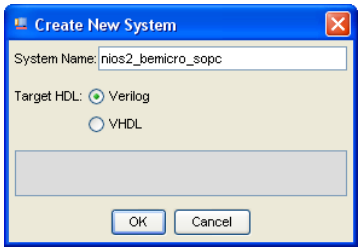
MODULE 4: Build the SOPC System

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 SOPC Builder

- From the **Tools** menu, select "**SOPC Builder**". There may be a slight delay while the SOPC Builder application launches.
- In the "**Create New System**" dialog, select **Verilog** and enter the system name as "**nios2_bemicro_sopc**".
- Click **OK**.



4.2 Manage Clocks

There is a 50 MHz oscillator on the BeMicro SDK, and this will be the clock source input. Other clocks are also required for the SOPC system components as well as for external components such as the SRAM. A PLL will be used to provide these clocks. The following table reviews the clocking scheme:

Clocking Scheme			
Component Name	Input Clock Frequency	Source	Designation
1. PLL	50 MHz	Oscillator on BeMicro SDK	ext_clk_50
2. Nios II processor and Mobile DDR SDRAM Controller	100 MHz	Output 'c0' of PLL	pll_c0
3. Ethernet MAC and SD Card Host Controller	60 MHz	Output 'c1' of PLL	pll_c1
4. Slow Peripherals	40 MHz	Output 'c2' of PLL	pll_c2

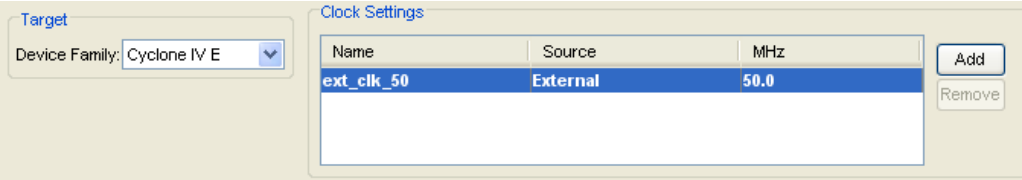
Perform the following instructions to build the system. It is helpful to have the rough sketch of your system handy so you can follow along.

4.3 Build the SOPC System

1. Define the external 50.0 MHz clock source

Reason: The clock coming into the FPGA is sourced by an on-board 50 MHz crystal oscillator. This clock source will feed our SOPC Builder system

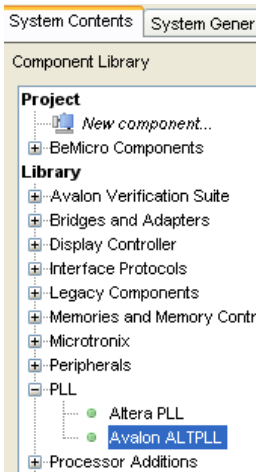
- Click into the clk_0 field and rename it from clk_0 to **ext_clk_50**.
- The default frequency already filled in is 50.0 MHz. Leave the frequency set at 50.0.



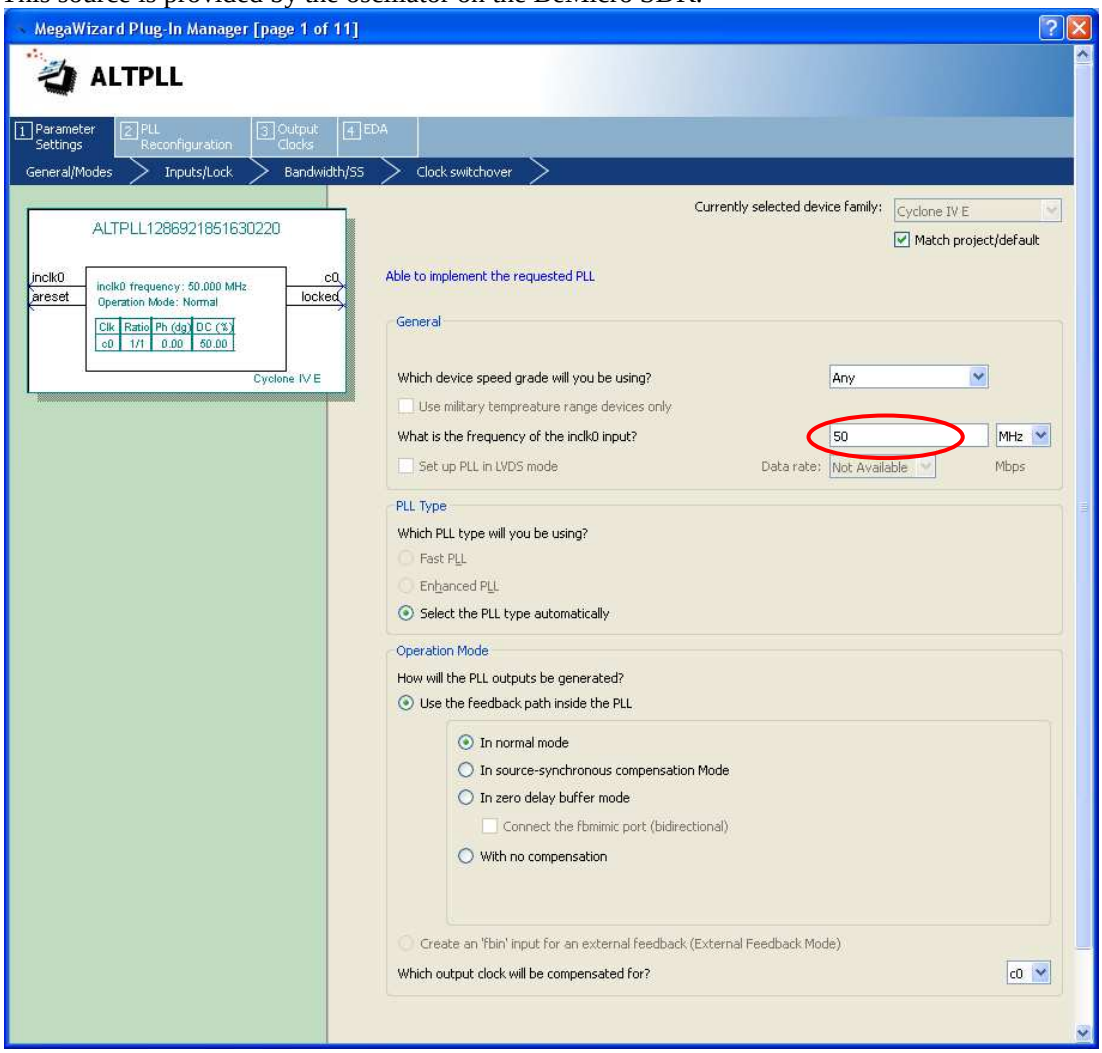
2. Add an Avalon ALTPLL

Reason: This peripheral instantiates a PLL which will generate the clocks for the system.

From the **Component Library** pane, expand **PLL** and double click on **Avalon ALTPLL**.

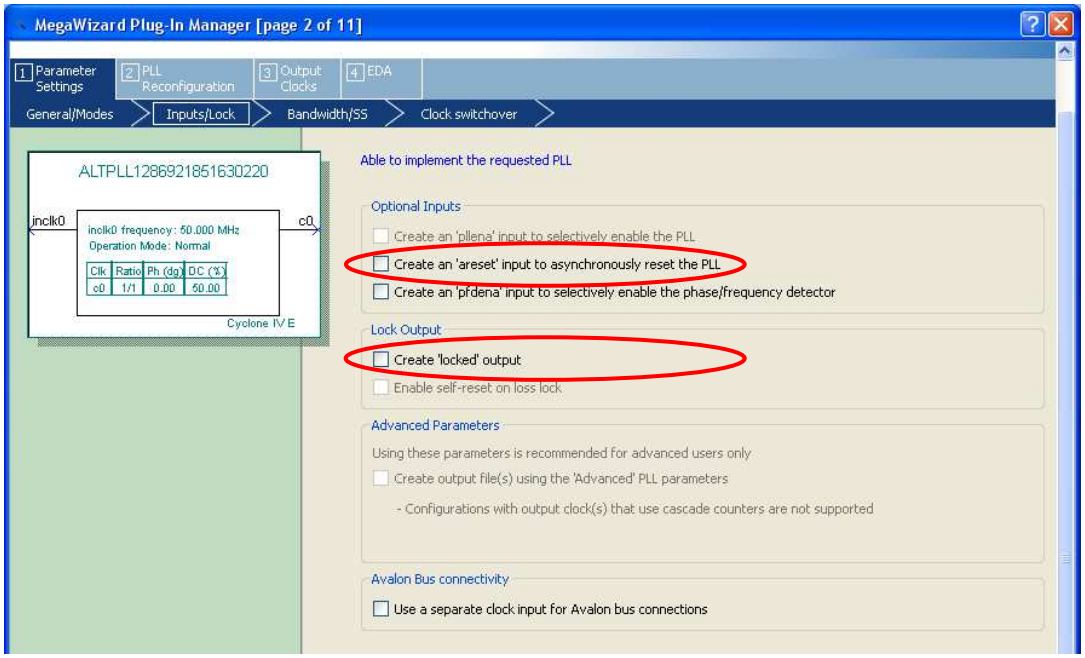


- **"General/Modes" tab (Page 1) of PLL MegaWizard**. Change the frequency of the clock input to **50 MHz**. This source is provided by the oscillator on the BeMicro SDK.



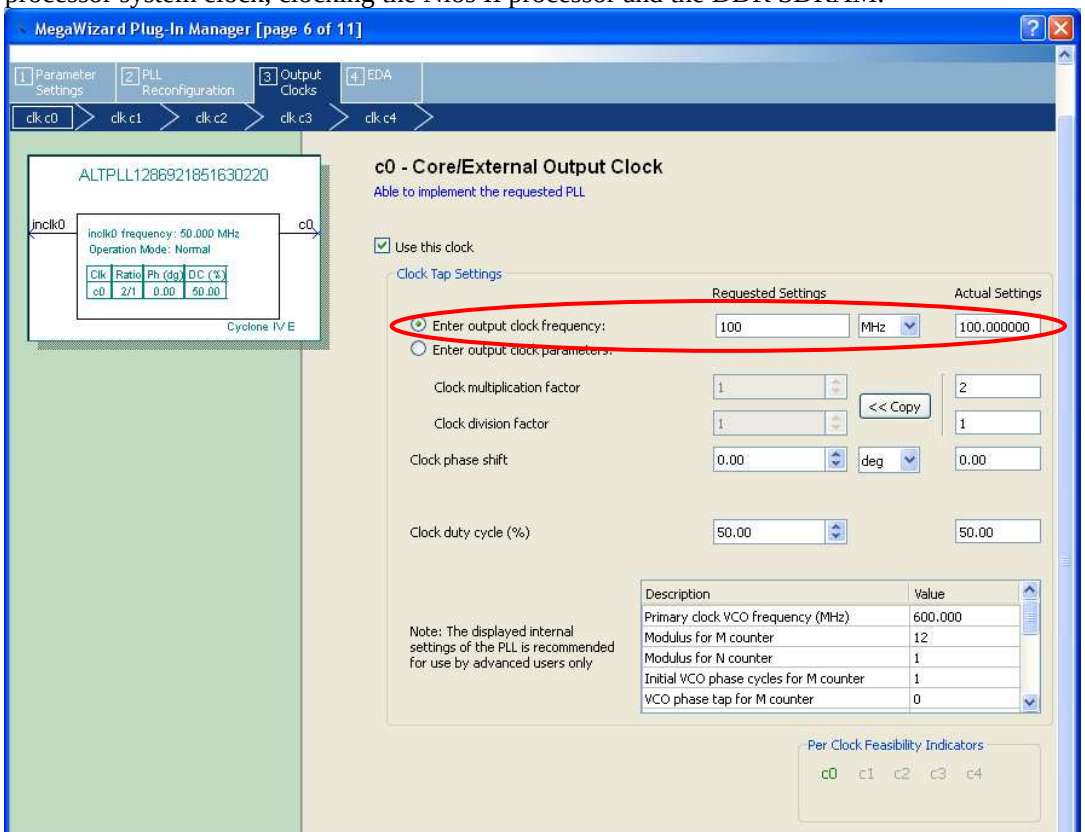
Click **Next** to move to the next tab of the wizard. (You may need to scroll down to see the Next button.)

- **"Inputs/Lock" tab (Page 2)**: Uncheck both "**Create an 'areset' input to asynchronously reset the PLL**" and "**Create 'locked' output**" options.
  Accept all other defaults.

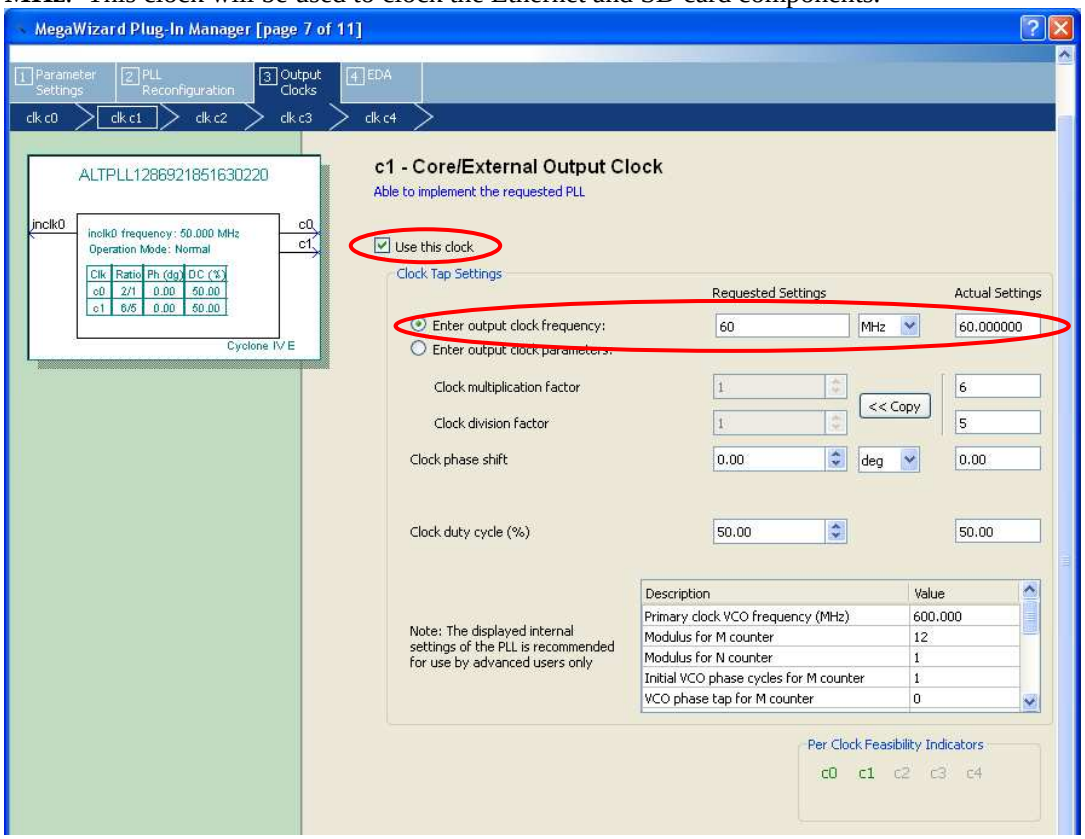


- **Pages 3-5**: Accept all defaults.

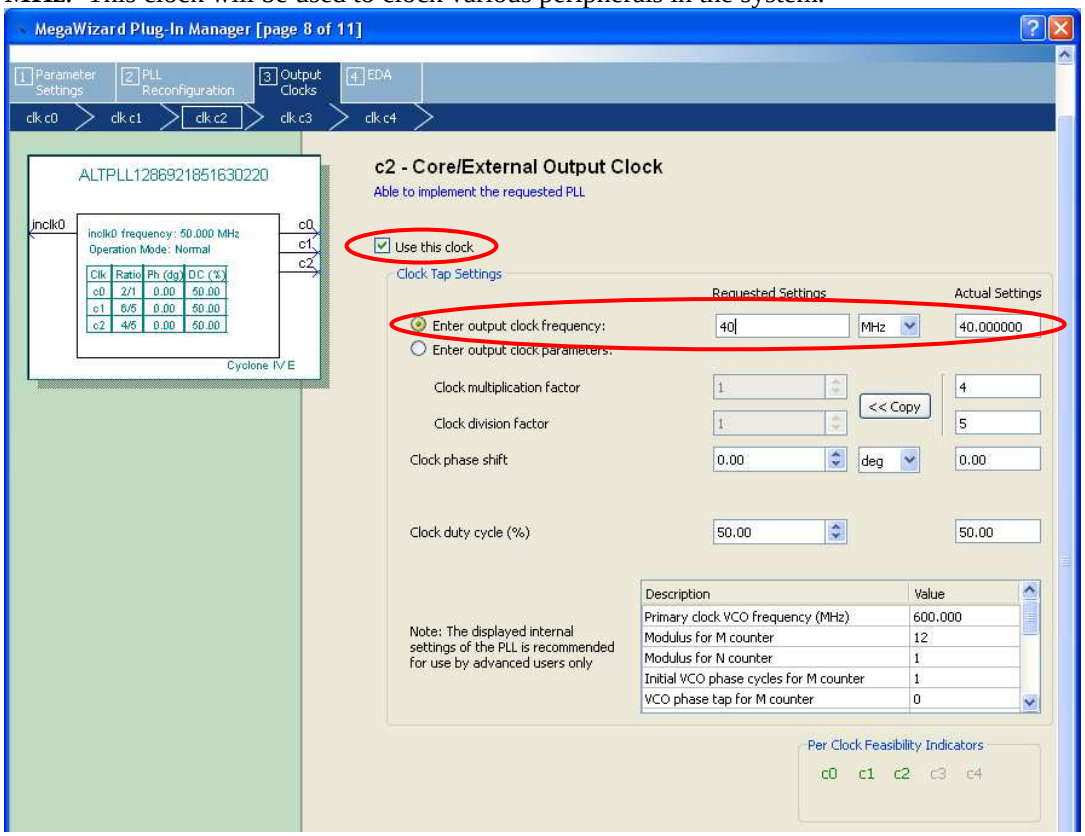
- **"c0 Core/External Output" (Page 6)**: Click "**Enter output clock frequency**". Configure c0 as 100 MHz output. Click on the "Enter output clock frequency" button and enter **100 MHz**. This clock will be used as the processor system clock, clocking the Nios II processor and the DDR SDRAM.



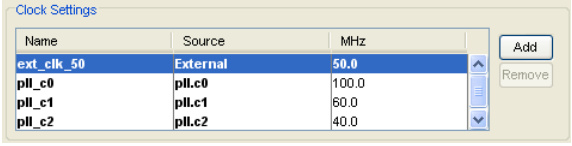
- **"c1 Core/External Output" (Page 7)**: Click "**Enter output clock frequency**". Configure c1 as 60 MHz output. Check the "Use this clock" button. Click on the "Enter output clock frequency" button and enter **60 MHz**. This clock will be used to clock the Ethernet and SD card components.



- **"c2 Core/External Output" (Page 8)**: Click "**Enter output clock frequency**". Configure c2 as 40 MHz output. Check the "Use this clock" button. 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 **Finish.** This will take you to the summary tab.
- Click **Finish** again to close the ALTPLL MegaWizard.
- A component entitled "altpll_0" should appear under Module Name. **Rename** the Avalon ALTPLL component from "altpll_0" to "**pll**". (You can right click to bring up a menu with a rename option.)
- Ensure that the name of the PLL is "pll". In the Clock Settings window (top right), the pll clock and the source clock should appear as shown here.



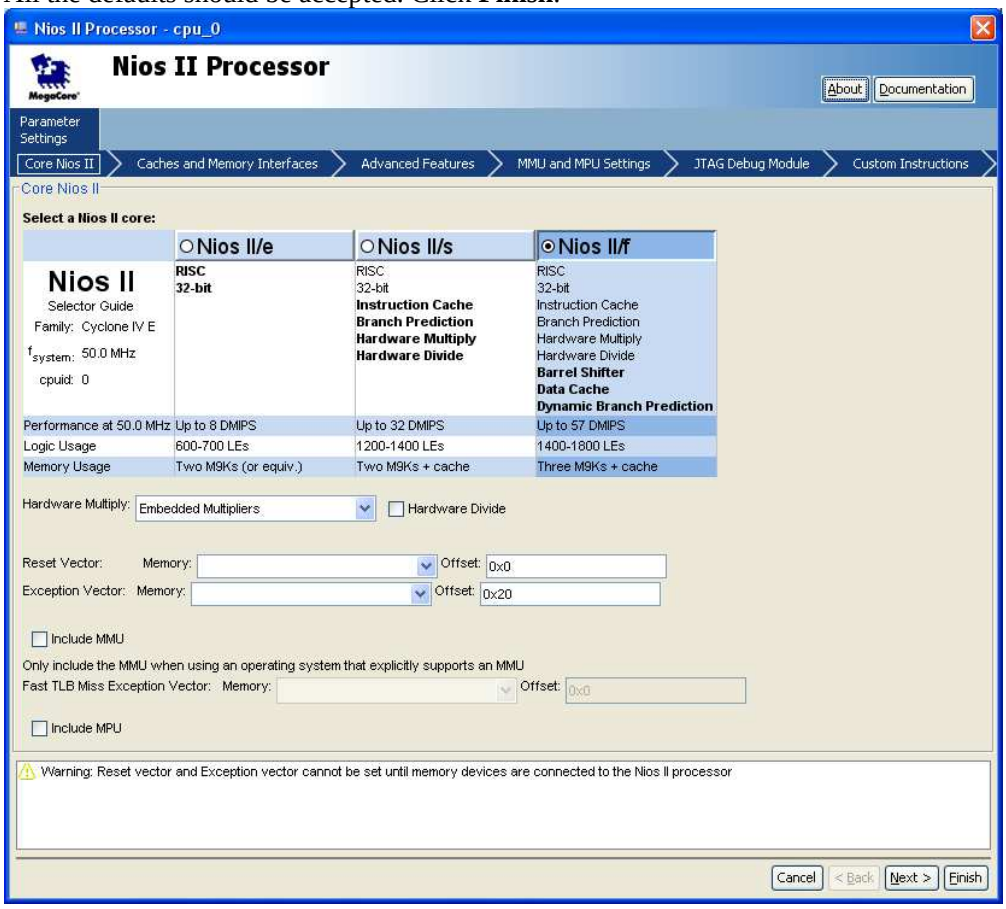
An error will appear in the bottom console indicating that the pll_slave port of the pll peripheral is not connected to a master. Ignore this for now. We will address connections in the upcoming steps.

3. Add a Nios II Processor

Reason: A CPU is needed to run the software applications.

- From the **Component Library** pane, under the Library. Expand **Processors** and double click on **Nios II Processor**.

- Ensure that the Nios II/f core is selected. There are numerous options on the various pages of the MegaWizard. All the defaults should be accepted. Click **Finish**.

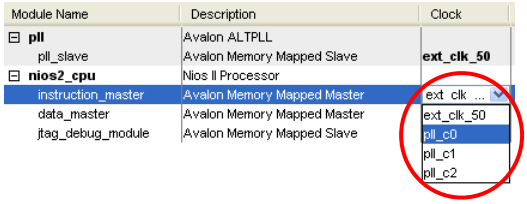


- **Rename** the component "**nios2_cpu**".

Occasionally save your work using **Save** on the **File** menu.

4. Configure clocks for the initial components

- At this point there are 2 components in the system. From the drop-down list in the **Clock** column, ensure that the **PLL** is set up with the **ext_clk_50** source, and then change the setting for the **nios2_cpu** to be driven by the **pll_c0** source.
  Since the **nios2_cpu** is driven by ext_clk_50, click on that field and change the selection to **pll_c0** as shown below:



5. Add an on-chip RAM

Reason: 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.

- Expand **Memories and Memory Controllers**. Expand **On-Chip** and double click on **On-Chip Memory (RAM or ROM)**.
- Set the "Total memory size" to **32 KBytes**. Click **Finish**.
- Right click on the Name field and choose **Rename** from the pop up menu. Name this RAM component "**onchip_ram**".

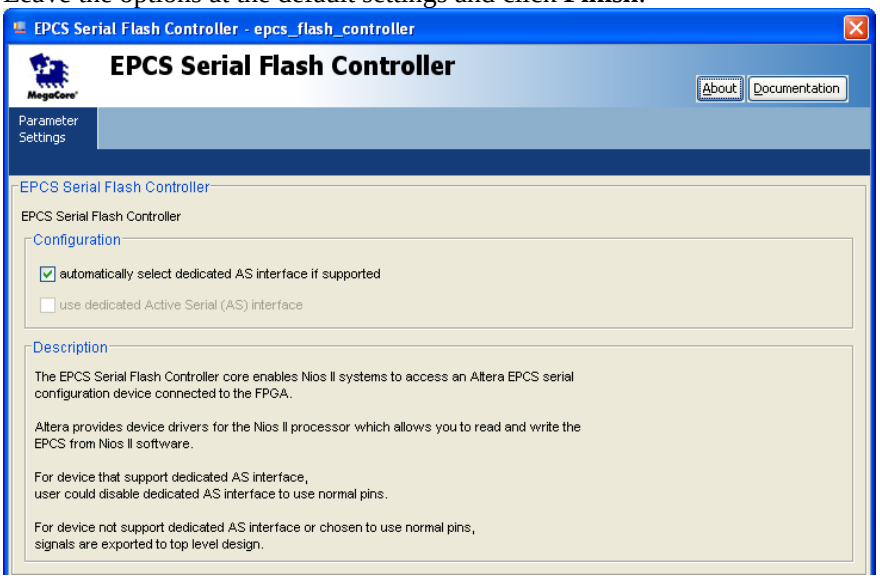
  The cpu/instruction_master and cpu/data_master should already be connected to the onchip_sram/avalon_slave of this interface.

6. Add an EPCS Serial Flash Controller

Reason: Altera FPGAs are volatile and require an external device to provide them with their configuration. There are many different configuration schemes available. On the BeMicro SDK, an Altera 16 Mbit EPCS device (serial config flash) is used. The EP4CE22 device requires 5.8 Mbits of configuration data, which means there will be roughly 10 Mbits of serial flash remaining in the EPCS16 device. The EPCS Controller provide access to the serial

config flash during run-time and the remaining space in the serial config flash can be used as a serial flash for code or data storage.

- Expand **Memories and Memory Controllers**. Expand **Flash** and double click on **EPCS Serial Flash Controller**.
- Leave the options at the default settings and click **Finish**.



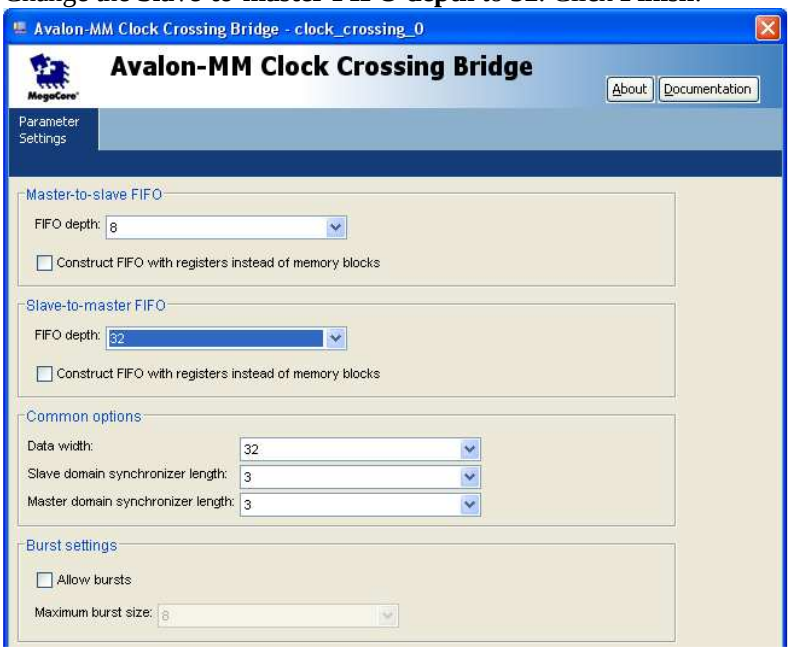
- Rename the component from epcs_flash_controller_0 to **epcs_flash_controller**.

7. **Add Avalon-MM Clock Crossing Bridge Peripheral for the "slow" peripherals.**

Reason: A clock crossing bridge is required because the Nios II processor and the slow peripherals run in different clock domains.

- From the **System Contents** menu, expand **Bridges and Adapters**. Expand **Memory Mapped** and double click on **Avalon-MM Clock Crossing Bridge**.

- Change the **Slave-to-master FIFO depth** to **32**. Click **Finish**.



- Right click on the Name field and choose **Rename** from the pop up menu. Name this bridge "**slow_periph_bridge**".

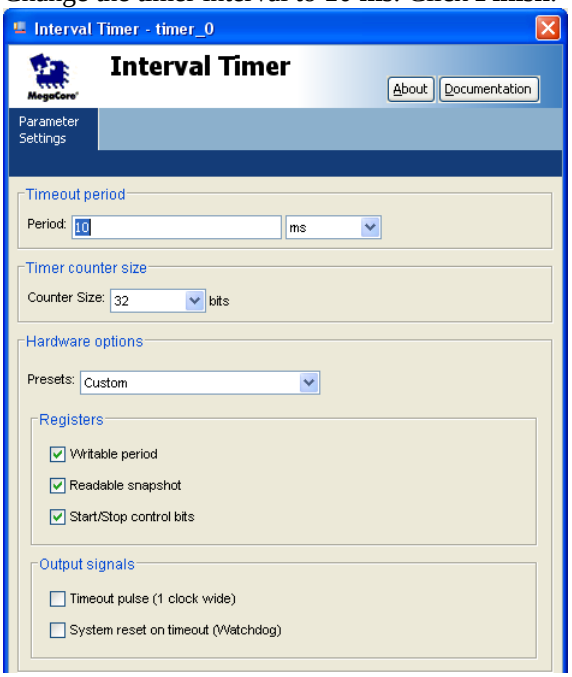
  The nios2_cpu's instruction_master and data_master ports should already be connected to the s1 slave port of this bridge. The m1 master port will be connected in the upcoming steps.

8. Add a 10 ms Interval Timer Peripheral

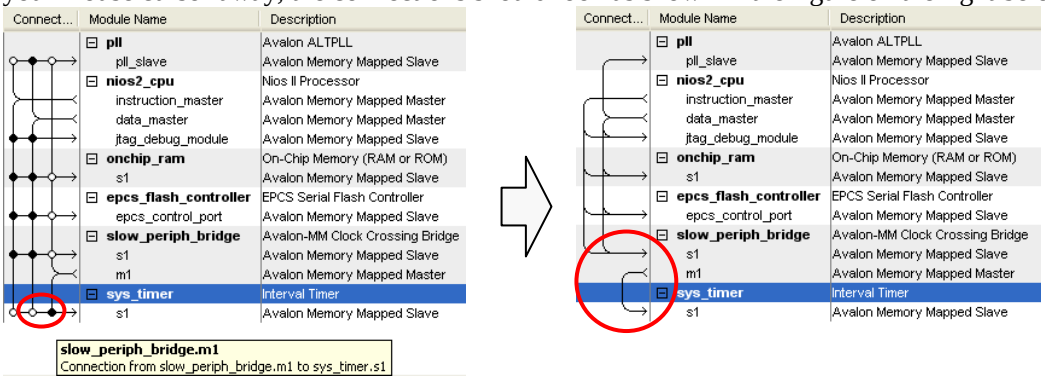
Reason: Many software applications require periodic interrupts to maintain various time bases and timing requirements within the application.

- From the **System Contents** menu, expand **Peripherals**, expand **Microcontroller Peripherals** and double click on **Interval Timer**.

- Change the timer interval to **10 ms**. Click **Finish**.



- **Rename** the component "**sys_timer**".
- Change the connection on the s1 slave port of the peripheral to be connected to the m1 master port of the slow_periph_bridge. To do this you will need to hover your mouse over the connections area so that the connection options appear. Then click on the appropriate circles to disconnect the s1 port from the nios2_cpu data_master port and instead connect it to the slow_periph_bridge m1 port. When you move your mouse cursor away, the connections should look as shown in the figure on the right below:



9. Add a 1 ms Interval Timer Peripheral

Reason: The Nios II HAL library provides a high resolution timer facility that allows software applications to measure time at the system clock rate. A second timer peripheral is useful for this.

- From the **System Contents** menu, expand **Peripherals**, expand **Microcontroller Peripherals** and double click on **Interval Timer**.

- Accept the default settings. Click **Finish**.
- **Rename** the component "**high_res_timer**".
- Similar to what was done with the previous interval timer, change the connection on the s1 slave port of the peripheral to be connected to the m1 master port of the slow_periph_bridge.

10. Add a Performance Counter

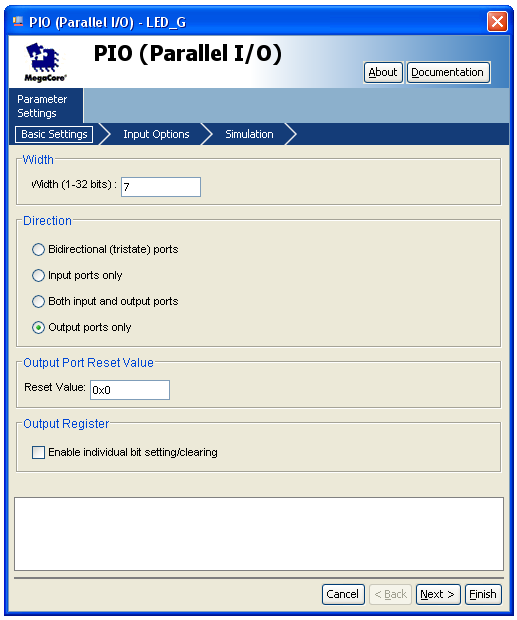
Reason: The performance counter peripheral is a block of counters for timing sections in your software code. With it you can accurately measure execution-time taken by blocks of C-code. Simple, efficient, minimally-intrusive macros allow you to mark the start and end of blocks-of-interest in your program.

- From the **System Contents** menu, expand **Peripherals**, expand **Debug & Performance** and double click on **Performance Counter Unit**.
- Accept the default setting of 3. Click **Finish**.
- **Rename** the component "**performance_counter**".
- Similar to what was done with the previous peripherals, change the connection on the s1 slave port of the peripheral to be connected to the m1 master port of the slow_periph_bridge.

11. Add PIO Peripheral for LEDs

Reason: The BeMicro SDK has 8 LEDs on it. You can drive these LEDs with an output PIO peripheral. We will drive 7 of the LEDs with a PIO peripheral. (The 8th LED will be controlled by a custom PWM peripheral added in the next step.)

- From the **System Contents** menu, expand **Peripherals**, expand **Microcontroller Peripherals** and double click on **PIO (Parallel I/O)**.
- Set the "**Width**" to 7 bits. Ensure that the "**Direction"** is set to "**Output ports only**". Click **Finish**.
- **Rename** the peripheral "**led_pio**".



12. Add PWM Peripheral

Reason: We will use the custom PWM component to control the intensity of the 8th LED.

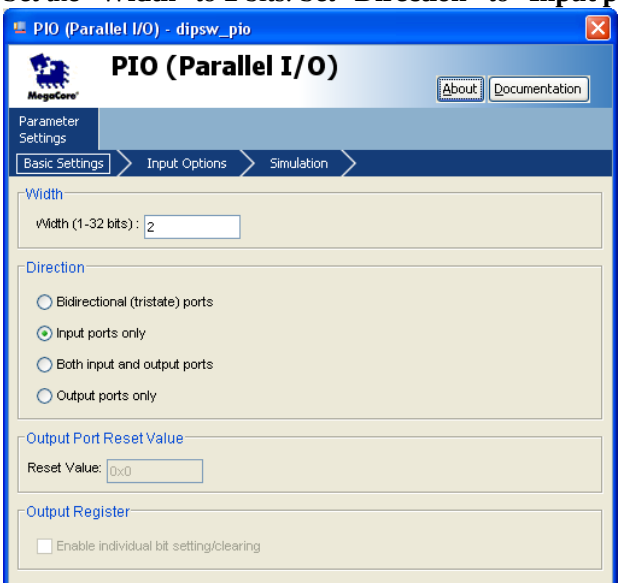
- From the **Project** section of the **Components Library**, expand **BeMicro Components** and double click on **Simple PWM**. *NOTE: This is a very simple custom component developed for educational purposes and the source code for this component is found in the ip/simple_pwm/ subdirectory within the lab project.*
- **Rename** the peripheral "**led_pwm**".

13. Add PIO Peripheral for DIP Switches

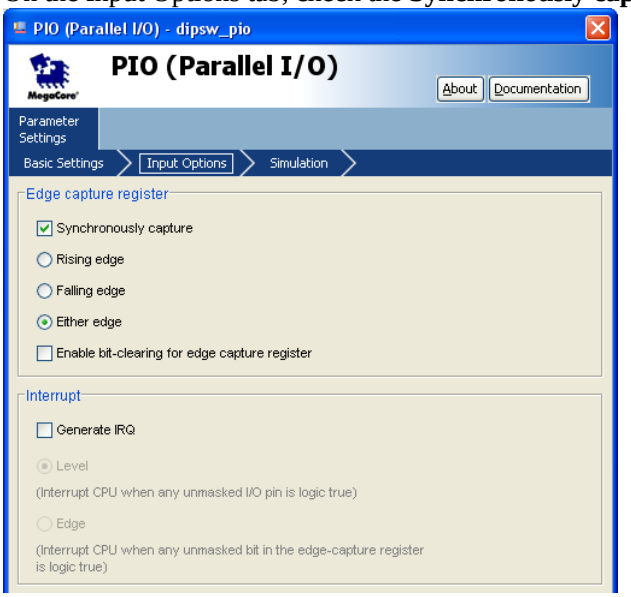
Reason: The BeMicro SDK has 2 DIP switches on it. You can use an input PIO peripheral to read in the DIP switch settings.

- From the **System Contents** menu, expand **Peripherals**, expand **Microcontroller Peripherals** and double click on **PIO (Parallel I/O)**.

- Set the "**Width**" to **2** bits. Set "**Direction"** to "**Input ports only**".

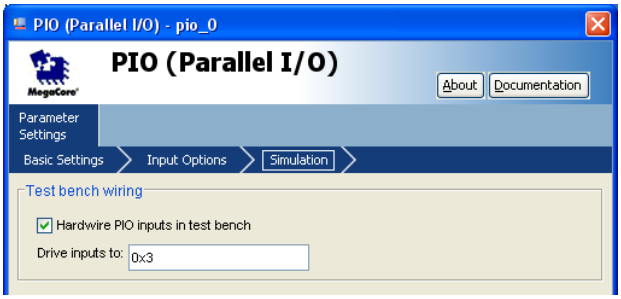


- Click **Next**
- On the Input Options tab, check the **Synchronously capture** and **Either edge** options:



- Click **Next**

- On the Simulation tab, check the **Hardwire PIO inputs in the test bench** option and drive the inputs to **0x3**:



- Click **Finish**
- **Rename** the peripheral "**dipsw_pio**".
- Similar to what was done with the previous peripherals, change the connection on the s1 slave port of the peripheral to be connected to the m1 master port of the slow_periph_bridge.

14. Add PIO Peripheral for Pushbutton Switch

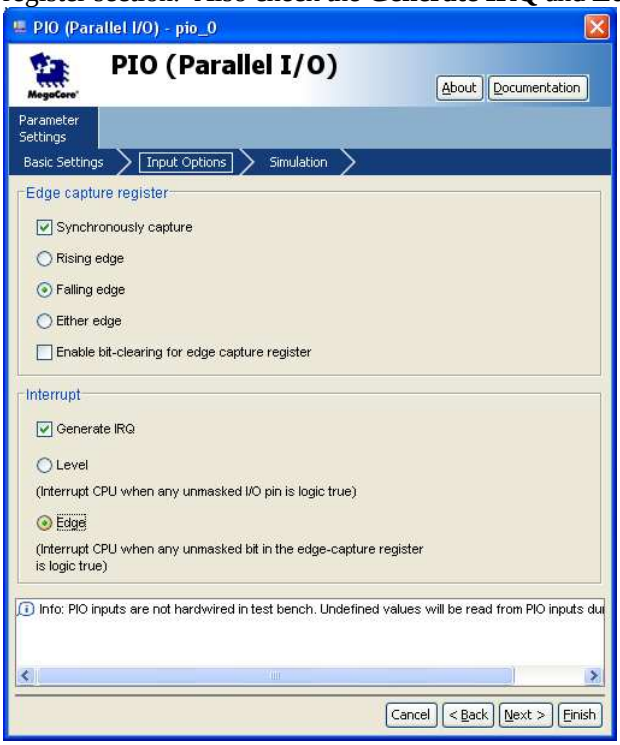
Reason: The BeMicro SDK has a pushbutton switch labeled "User" connected to one of the FPGA I/O pins. You can use an input PIO peripheral to detect when this pushbutton has been pressed and signal an interrupt to the processor.

- From the **System Contents** menu, expand **Peripherals**, expand **Microcontroller Peripherals** and double click on **PIO (Parallel I/O)**.

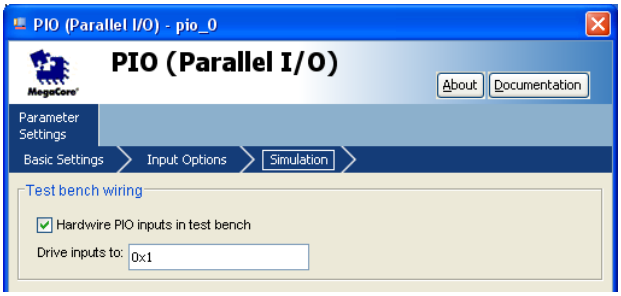
- Set the "**Width**" to **1** bit. Set "**Direction"** to "**Input ports only**".

- Click **Next**

- On the Input Options tab, check the **Synchronously capture** and **Falling edge** options in the Edge capture register section. Also check the **Generate IRQ** and **Edge** options in the Interrupt section:



- Click **Next**
- On the Simulation tab, check the **Hardwire PIO inputs in the test bench** option and drive the inputs to **0x1**:



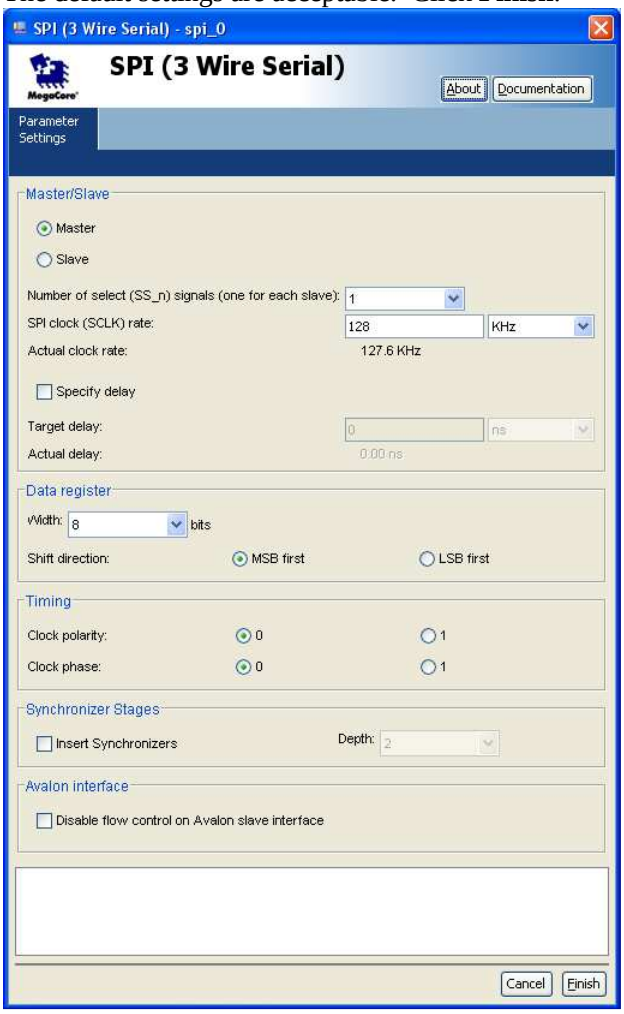
- Click **Finish**
- **Rename** the peripheral "**user_pio_pushbtn**".
- Similar to what was done with the previous peripherals, change the connection on the s1 slave port of the peripheral to be connected to the m1 master port of the slow_periph_bridge.

15. Add SPI Peripheral for accessing the On-board Temperature Sensor

Reason: The BeMicro SDK has a SPI temperature sensor device. You can use the SPI peripheral to access this temperature sensor device.

- From the **System Contents** menu, expand **Interface Protocols**, expand **Serial** and double click on **SPI (3-wire serial)**.

- The default settings are acceptable. Click **Finish**.



- **Rename** as "**temp_sense_spi**".
- Similar to what was done with the previous peripherals, change the connection on the s1 slave port of the peripheral to be connected to the m1 master port of the slow_periph_bridge.

16. Add JTAG UART Peripheral

Reason: 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.

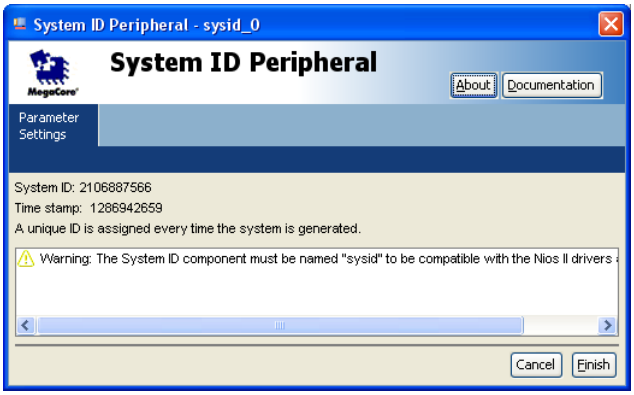
- From the **System Contents** menu, expand **Interface Protocols**, expand **Serial** and double click on **JTAG UART**.
- The default settings are acceptable. Click **Finish**.
- **Rename** as "**jtag_uart**".

- Similar to what was done with the previous peripherals, change the connection on the s1 slave port of the peripheral to be connected to the m1 master port of the slow_periph_bridge.

17. Add a System ID

Reason: 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.

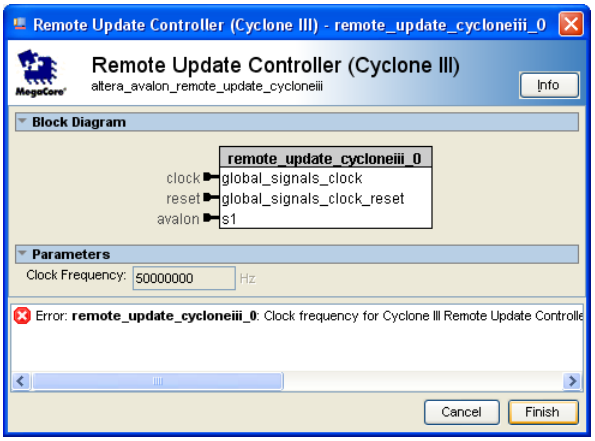
- From the **System Contents** menu, select **Peripherals** -> **Debug and Performance** -> **System ID Peripheral**. Double click to add the component to the system.
- The sysid dialog box appears. Click **Finish**.
- Rename as "**sysid**". The component must be named "sysid" to be compatible with Nios II software drivers and build tools.
- Once again, change the connection on the s1 slave port of the peripheral to be connected to the m1 master port of the slow_periph_bridge.



18. Add a Remote Update Controller

Reason: Cyclone IV FPGAs have a Remote Update feature which allows you to store an updated FPGA image in the config flash and then have the FPGA reconfigure itself with an updated image.

- From the **System Contents** menu, select **Peripherals** -> **FPGA Peripherals** -> **Remote Update Controller**. Double click to add the component to the system.



  You will notice an error indicating the the Remote Update Controller cannot be clocked any higher than 40 MHz. We have not yet configured the clock sources in our system. We will clear up this error a bit later when we set the clock sources for our components.

- Rename the component "**remote_update_blk**".
- Once again, change the connection on the s1 slave port of the peripheral to be connected to the m1 master port of the slow_periph_bridge.

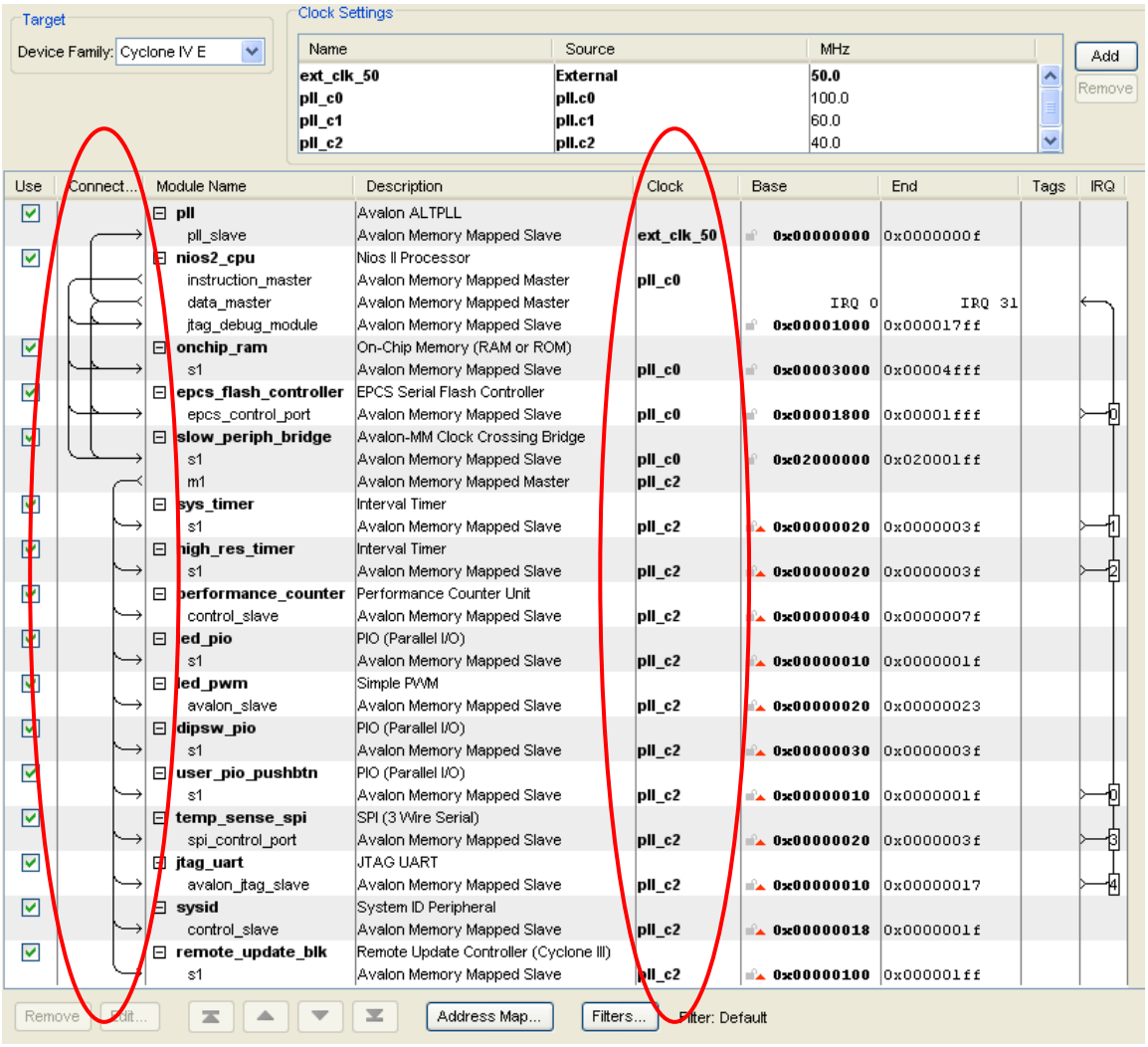
At this point all the components to the SOPC system have been added. Now you need to resolve the lingering system validation errors that have arisen during the design.

4.4 System Configuration

1. Clock source for each component

We need to change the clock source for each of the components such that only the PLL has **ext_clk_50** selected in the Clock" column and that the nios2_cpu, onchip_ram, epcs_flash_controller and the s1 port of the slow_periph_bridge have **pll_c0** selected for their clock sources. The m1 port of the slow_periph_bridge and all remaining components will have **pll_c2** selected for their clock sources.

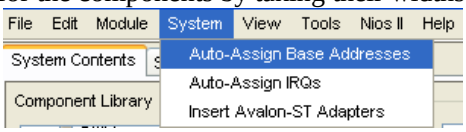
Ensure that your clock sources are configured as shown below.
Also, confirm that your connections are as shown in the connections column. You may have missed the previous steps to modify these connections as you were adding the components.



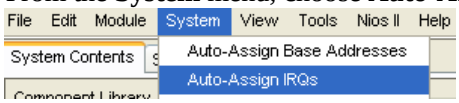
2. Set Base Addresses and Interrupt Priorities

SOPC Builder provides two easy menu items that help clean up address map issues and interrupt priority issues.

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



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

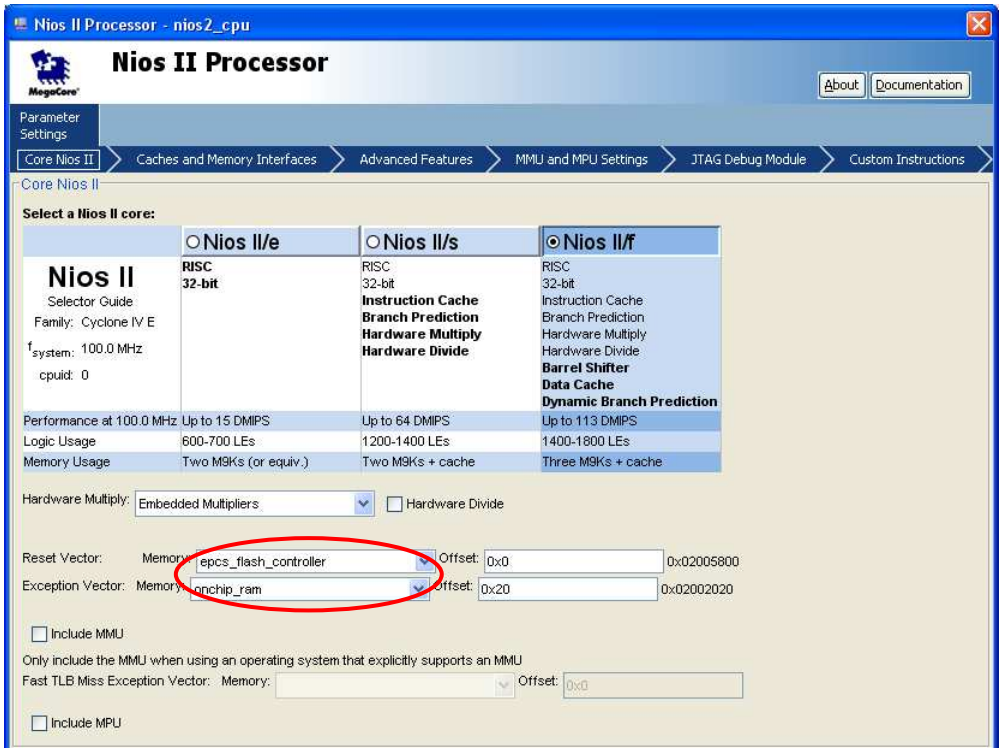


At this point you should only be left with a couple of information messages and reminders that you have yet to specify the CPU reset and exception address configuration.

3. 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.

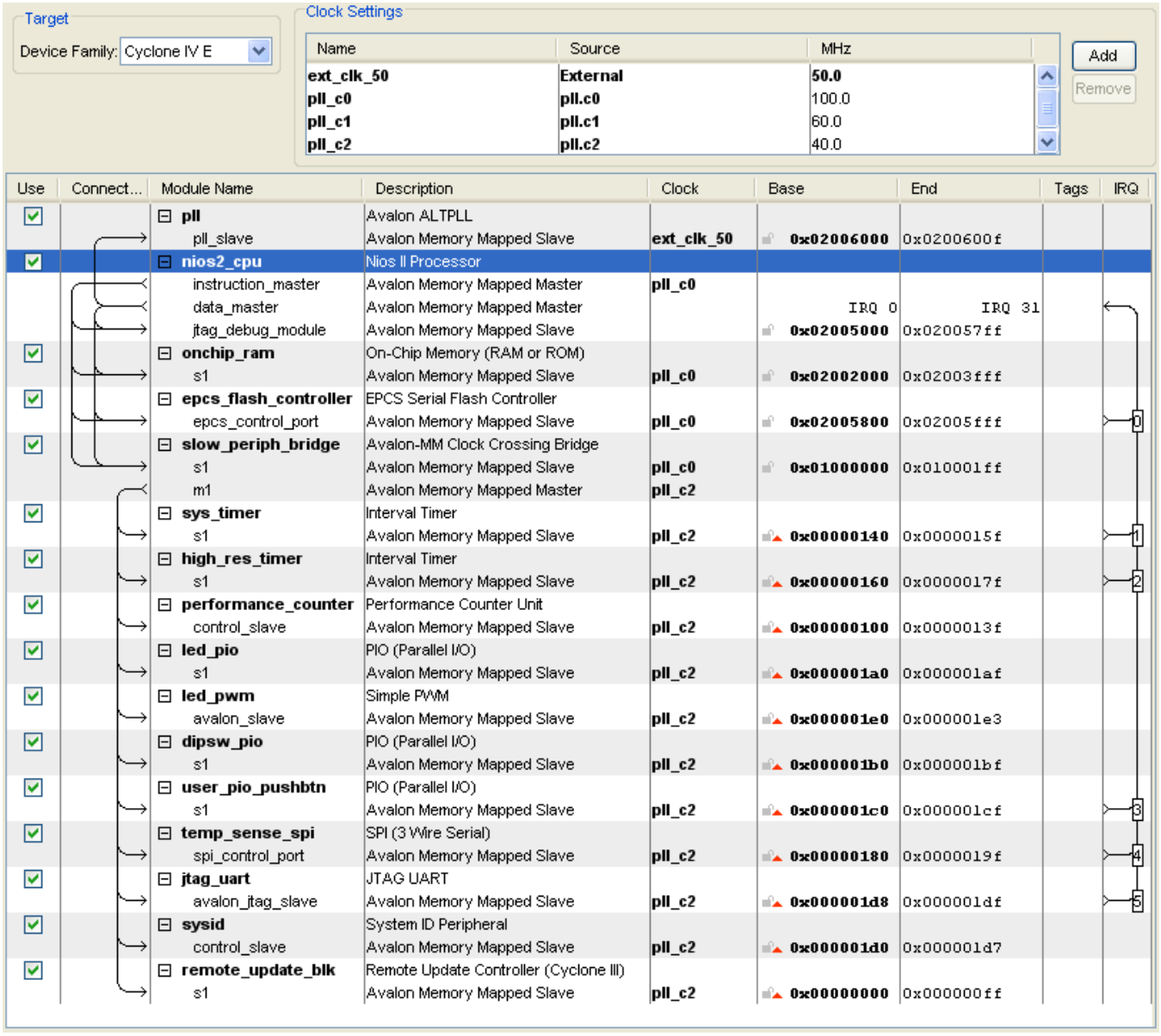
- Double click on the **cpu** peripheral to launch the "Nios II Processor Parameter Settings" GUI.
- Set the **Reset Vector** to point to the **epcs_flash_controller** with an offset of 0x0. When the Nios II processor comes out of reset, it will begin executing software at this memory location.
- Set 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.



Click **Finish**.

This concludes the system configuration. At this point you should have addressed all the system validation issues and can now generate the SOPC Builder System.

THE RESULTING SOPC SYSTEM



4.5 Generate the System

Please Double-check to make sure that all the *component names* in your SOPC system match the component names shown above. Do not worry if the base and end addresses do not match exactly.

Click the **Generate** button. SOPC Builder will now create:

- The HDL for the various components in your system
- System interconnect to connect the components together
- System description file used by the software development tools (the Nios II SBT) to build the software project

Once your system has been successfully generated you will see the info message “Info: System generation was successful”. Exit SOPC Builder by clicking the **Exit** button and click **Save** when it asks if you’d like to save the system.

CONGRATULATIONS!! You have just built your first SOPC system!

MODULE 5: Complete the Quartus II Project

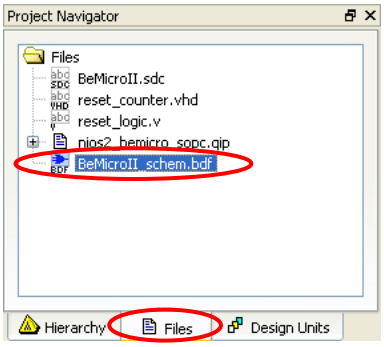
Module Objective

In this module you complete the Quartus II project by adding the generated SOPC system to the top-level entity. Compile in the Quartus II 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 BeMicro SDK, at which point the on-board FPGA will function as a processor custom-made for your application.

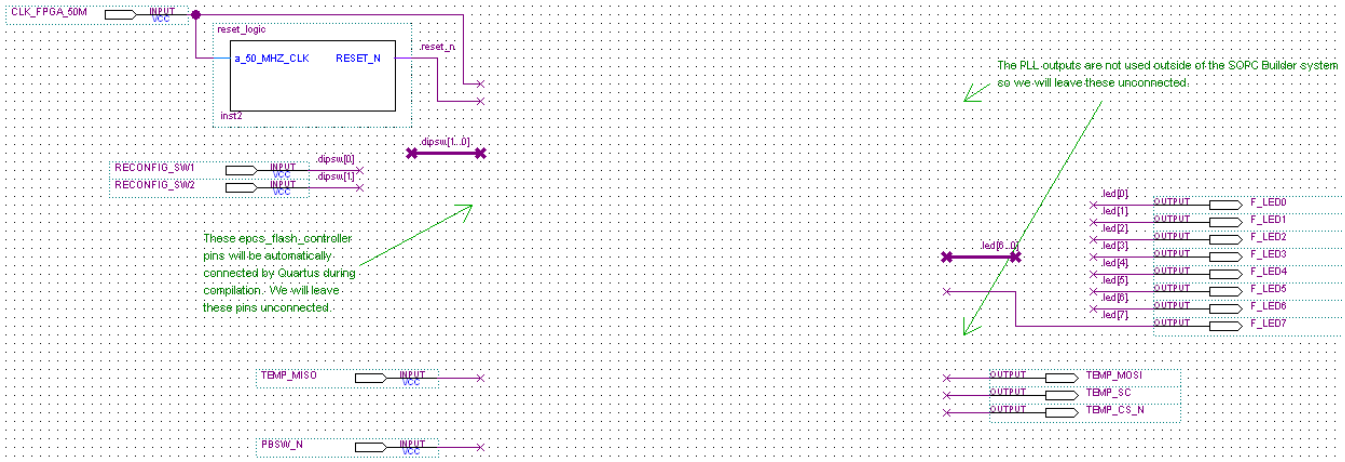
5.1 Complete the Quartus II Project

Note: The following step is only required if you chose the schematic as your top level file at the beginning of the lab.

- In the **Project Navigator** window pane in the Quartus II software, select the **Files** tab and open the **BeMicro _schem.bdf** by double clicking it.

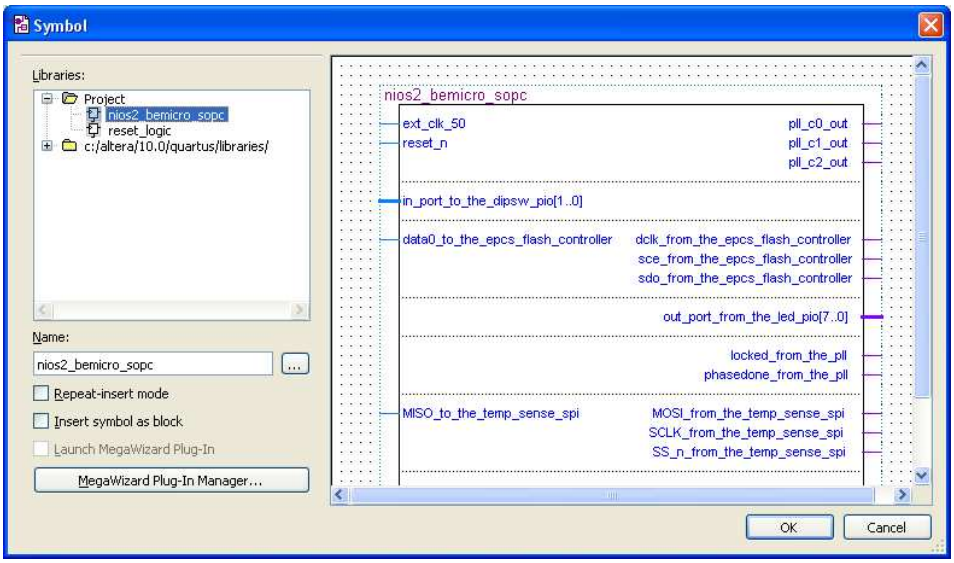


Examine the BeMicroII_schem.bdf shown below:

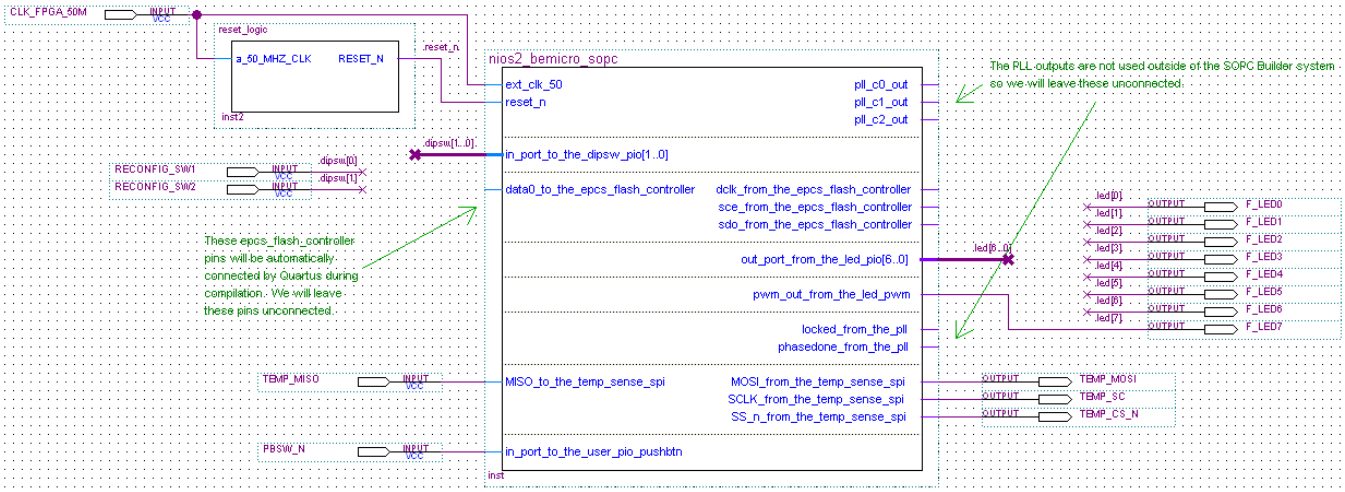


The pins required by the design as well as some reset logic are present, but the main block of logic representing the SOPC Builder system is missing. Add the SOPC Builder system instance to this top level.

- From the **Edit** Menu, click on **Insert Symbol.**
- In the **Libraries** pane expand the **Project** folder and choose the **nios2_bemicro_sopc** symbol. Then click **OK**.



- Align the symbol within the schematic and place the instance into the schematic. You may need to move the connector pins to align them with the pins of the symbol.

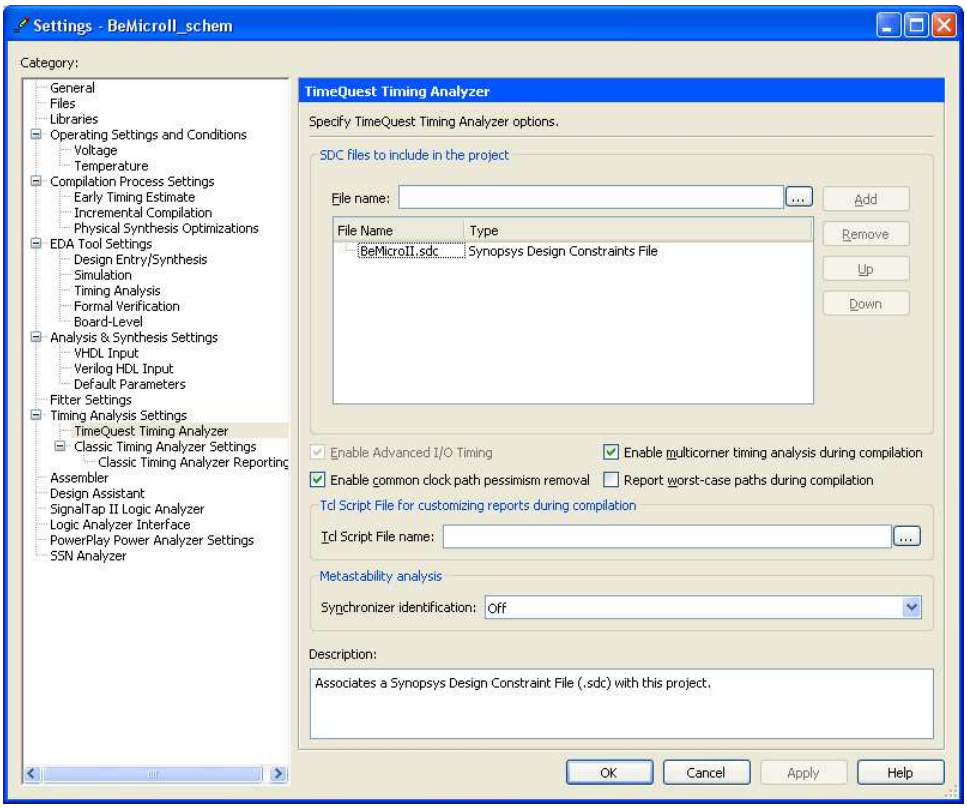


5.2 Set up the Quartus II Project to point to the proper timing constraint files

Reason: Similar to ASIC design, FPGAs require timing analysis since routing within the FPGA device will vary based on where the Quartus II Fitter places the logic. Entering timing constraints will provide the Quartus II Fitter with design goals to make intelligent choices about where to place the logic and other elements in the design and then will provide the Quartus II TimeQuest Timing Analyzer with information so that it can report whether we have met our timing goals. The constraints are coded in an industry standard language called SDC (Synopsis Design Constraints).

SOPC Builder automatically generated SDC files for components which provide timing information. In our system, only the Nios II CPU has generated SDC files. In addition, an SDC file called bemicro_lab.sdc is included with the lab files and instructs TimeQuest to determine our clock rates by analyzing the PLL. We must set up our Quartus project to point to these SDC files.

- From the **Assignments** menu in the Quartus II software, select **Settings**.
- Expand **Timing Analysis Settings** and select **TimeQuest Timing Analyzer**
- In the "SDC files to include in the project" click on the "**…**" and select the **BeMicroII.sdc** file found in the project directory. Click on the "**Add**" button to add it to the list of SDC files.



At this point the design is ready for compilation.

- Click the **Start Compilation** button on the Quartus II tool bar.

The Quartus II software will take a few minutes to compile the design. 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 that relate to the files from the automatically generated system, missing assigments/features and incomplete pin assignments but these will not affect the functionality of the system.

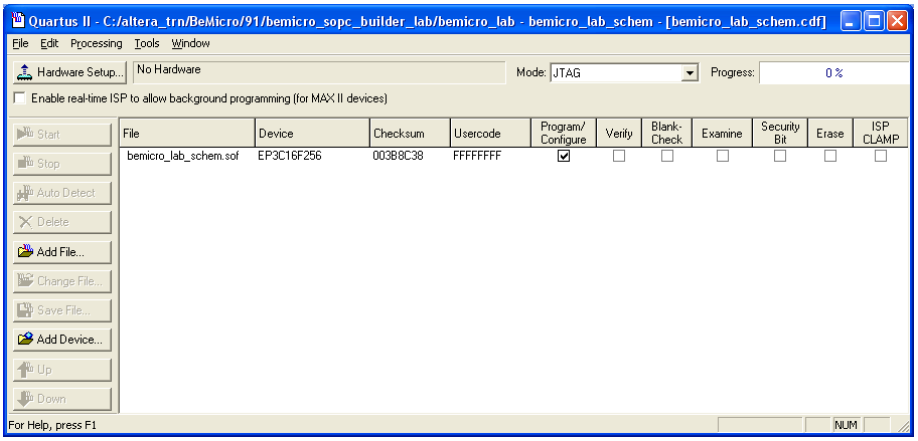
The output of the compilation is a SOF file entitled “BeMicroII.sof” if you have a Nios II license or “BeMicroII_time_limited.sof” if you do not have a license.

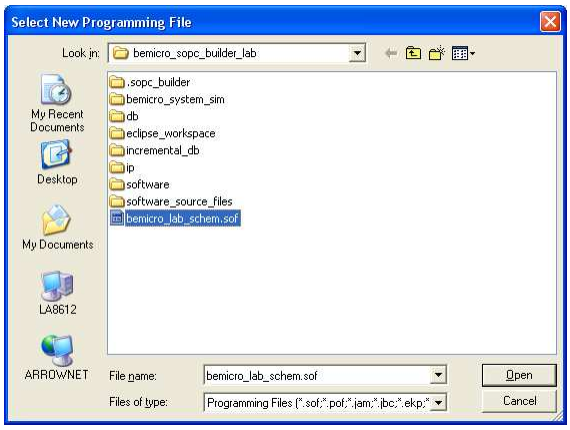
5.3 Download the FPGA configuration

Ensure that your BeMicro kit is plugged into your PC USB port and launch the Quartus II Programmer to configure the FPGA.

- Click on the **Programmer** icon on the Quartus II desktop, or alternatively open the **Programmer** from the **Tools** menu.

- In Quartus II Programmer click **Hardware Setup**. In the **Currently selected hardware** drop box select **Arrow-USB-Blaster**. Click **Close.** Click on **Auto Detect**.
- After clicking the **Auto Detect** button you should find that the programmer detects the **EP4CE22** device on your BeMicro SDK.
- Double click on the **<none>** in the **File** field, or select **<none>** and click on the **Change File** button.
- Select bemicro_lab_***.sof. Click **Open**.

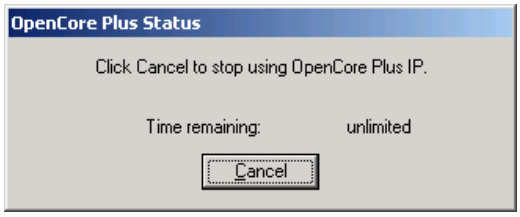




- After selecting your SOF file, click on the **Program/Configure** checkbox.
- Press the **Start** button to program the FPGA.

After programming the FPGA the progress indicator should indicate 100% complete, and there should be no error messages displayed.

NOTE: If you do not have a license for the Nios II processor then your system would have generated the Nios II in OpenCore Plus evaluation mode and your sof programming file will be time-limited. If you are running with a time-limited SOF file, then this window pops up on the Quartus II 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.

CONGRATULATIONS!!

You have just compiled and downloaded the FPGA image onto the target. The processor is ready to run, so all you need to do now is develop the software application and download it to the target.

MODULE 6: Build the Software Application

Module Objective

In this module you use the Nios II Software Build Tools (SBT) for Eclipse to develop the software application that will run on your system. You will create a new software application project, add the software source files to the project, configure the project and build it. The result of the build is an executable (ELF). The application will be downloaded into memory from where it will be executed.

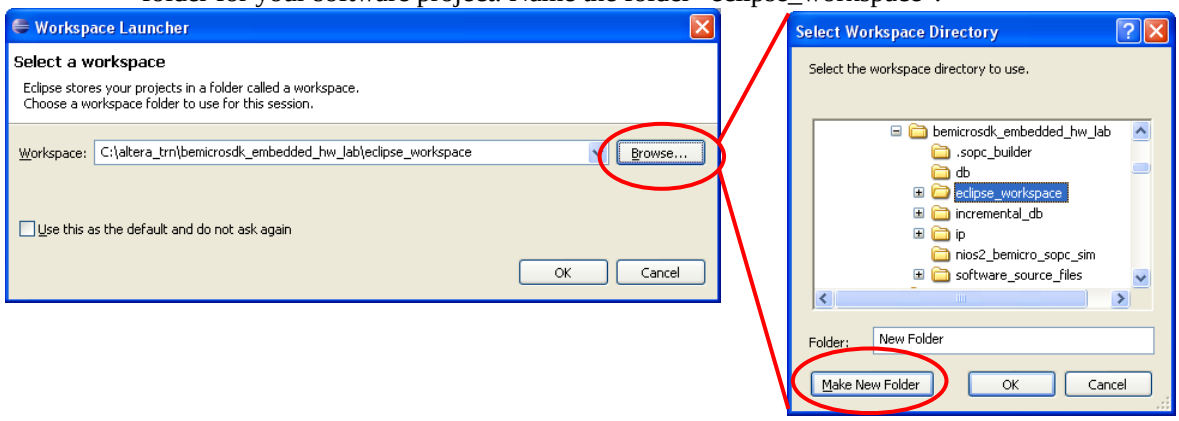
6.1 Launch the Nios II Software Build Tools for Eclipse

Launch the Nios II SBT from the **Start -> All Programs -> Altera -> Nios II EDS 10.0 -> Nios II 10.0 Software Build Tools for Eclipse** or alternatively it can be launched from the **SOPC Builder -> Nios II** menu.

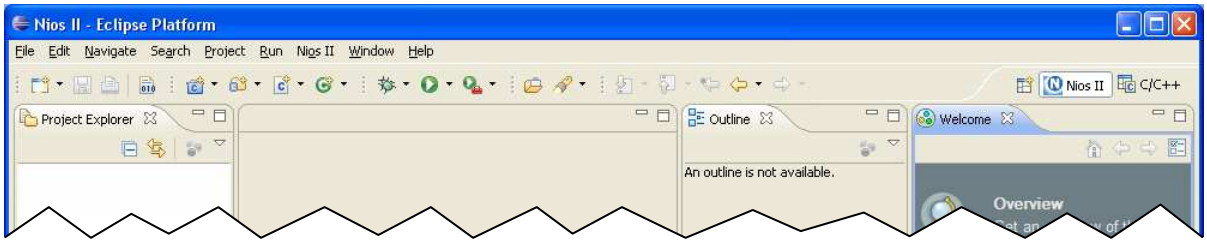
1. Initialize Eclipse workspace

When Eclipse first launches, a dialogue box appears asking what directory it should use for its workspace. It is useful to have a separate Eclipse workspace associated with each hardware project that is created in SOPC Builder.

- **Browse** to the directory that you created the Quartus II project in and click **Make New Folder** to create a folder for your software project. Name the folder "eclipse_workspace".

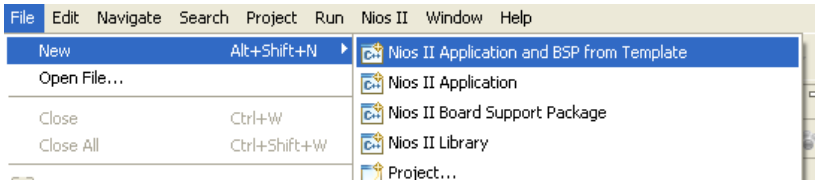


After selecting the workspace directory, click "**OK**" and Eclipse will launch and the workbench will appear in the Nios II perspective.

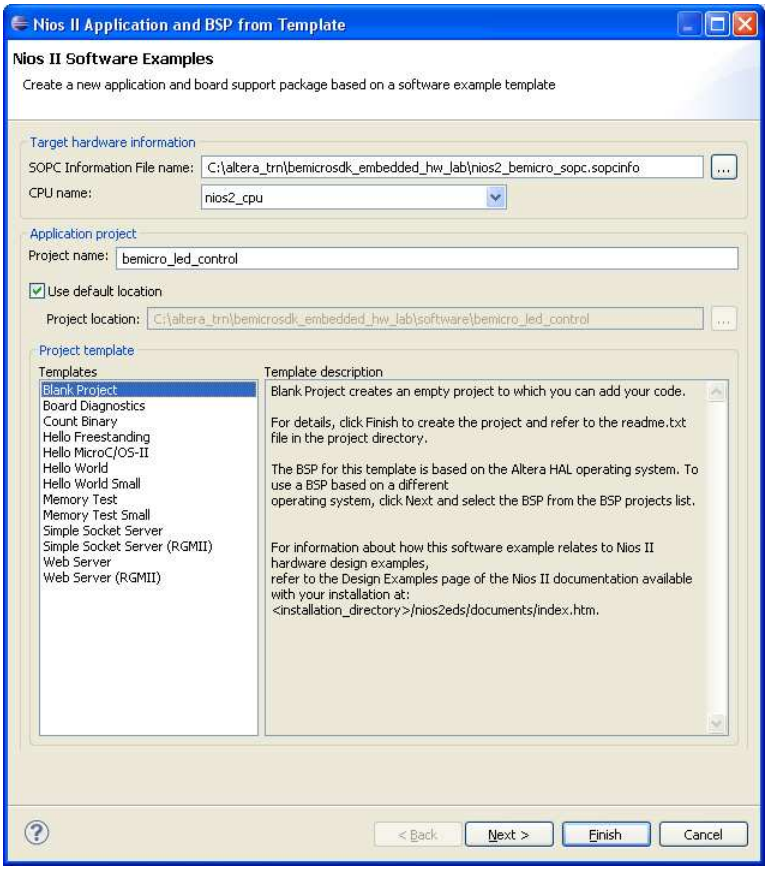


6.2 Create a new software project in the SBT

Select **File -> New -> Nios II Application and BSP from Template**.



- To set the SOPC Information File, click the **Browse** button to locate the **bemicro_system.sopcinfo** file located in the Quartus II project directory.
- Set the name of the Application project to "**bemicro_led_control**".
- Select the **Blank Project** template under **Project Template**.
- Click the **Finish** button.

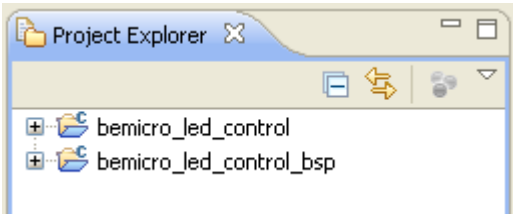


The tool will create two new software project directories

Each Nios II application has 2 project directories in the Eclipse workspace.

a. The application software project itself - this where the application lives.

b. The second is the **Board Support Package (BSP)** project associated with the main application software project. This project will build the system library drivers for the specific SOPC system. This project inherits the name from the main software project and appends "_bsp" to that.



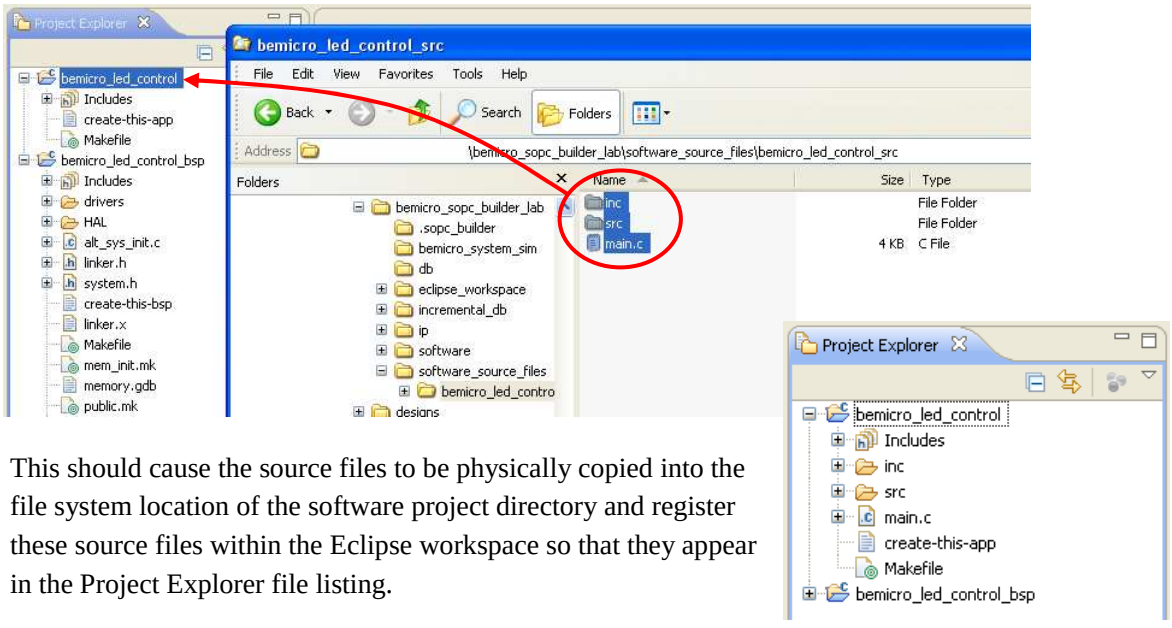
Initial content of the project

Since you chose the "blank" project template, there are no source files in the application project directory at this time. The BSP contains a directory of software drivers as well as a system.h header file, system initialization source code and other software infrastructure.

6.3 Add source code to the project

In Windows Explorer locate the project directory which contains a directory called "**software_source_files**" which contains a directory called "**bemicro_led_control_src**". This directory contains an "**inc**" directory, "**src**" directory and "**main.c**" file. You will copy these files and directories from Windows Explorer into the Eclipse software project directory, "bemicro_led_control".

- **Select** the 3 files and **drag** them over the "bemicro_led_control" directory in the SBT window and **drop** the files onto the project folder.

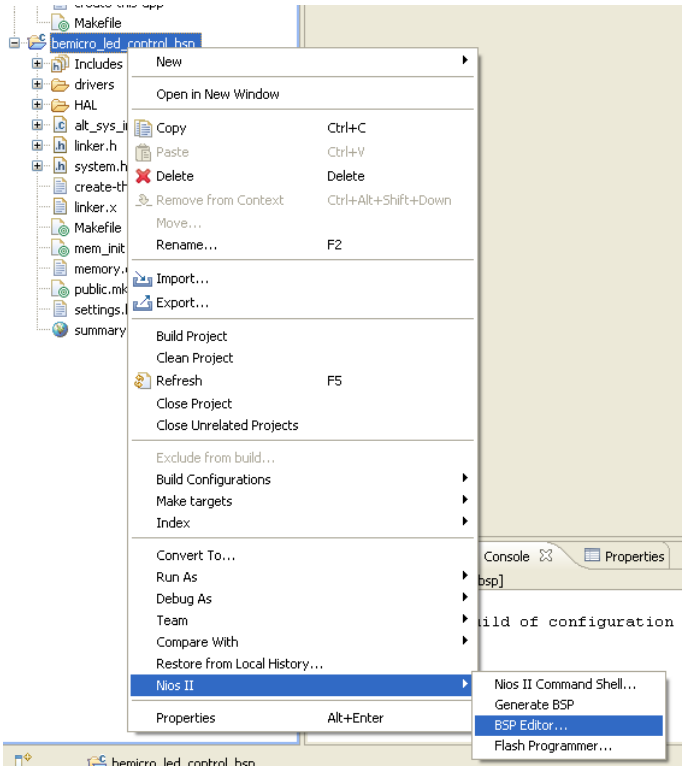


- This should cause the source files to be physically copied into the file system location of the software project directory and register these source files within the Eclipse workspace so that they appear in the Project Explorer file listing.

6.4 Configure Board Support Package

- Configure the board support pakcage to specify the properties of this software system by using the BSP Editor tool. These properties include what interface should be used for stdio and stderr messages, which memory should stack and heap be allocated in and whether an operating system or network stack should be included with this BSP.

- Right click on the **bemicro_led_control_bsp** project and select **Nios II -> BSP Editor…** from the right-click menu.



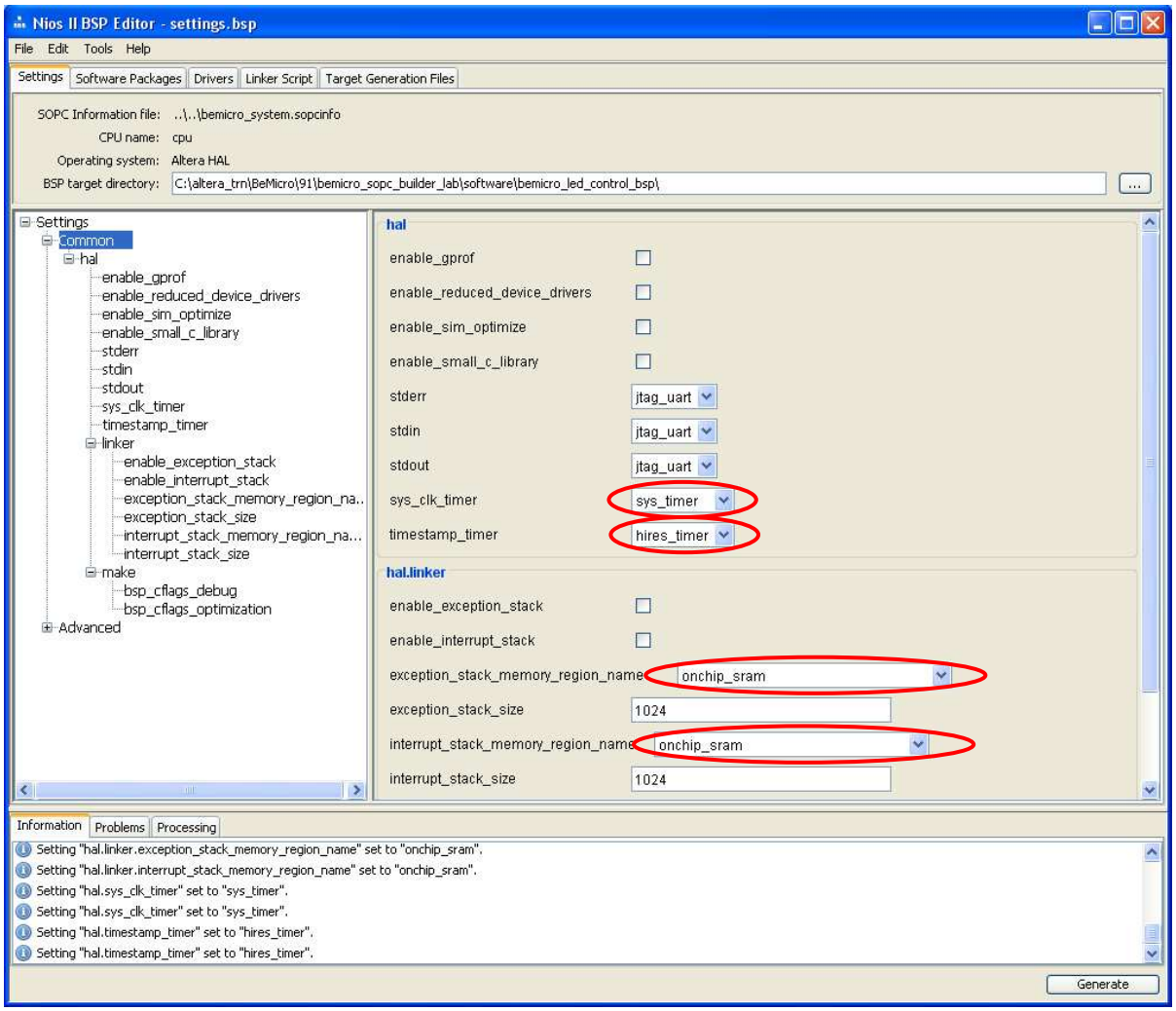
The software project provided in this lab does not make use of an operating system. All stdout, stdin and stderr messages will be directed to the **jtag_uart**. The auto-generated linker script will be used and the various linker sub-sections (Program memory, Read-only data memory, Read/write data memory) will be stored into **onchip_ram**. We will point the linker to place the heap and stacks in **onchip_ram** memory.

In the "Common" settings view, change the following settings:

- Select the **sys_timer** peripheral as the hardware for the **sys_clk_timer**.

- Select the **high_res_timer** peripheral as the hardware for the **timestamp_timer**.

- Ensure that **onchip_ram** is selected as the linker target for both the **exception_stack_memory_region_name** and the **interrupt_stack_memory_region_name**.



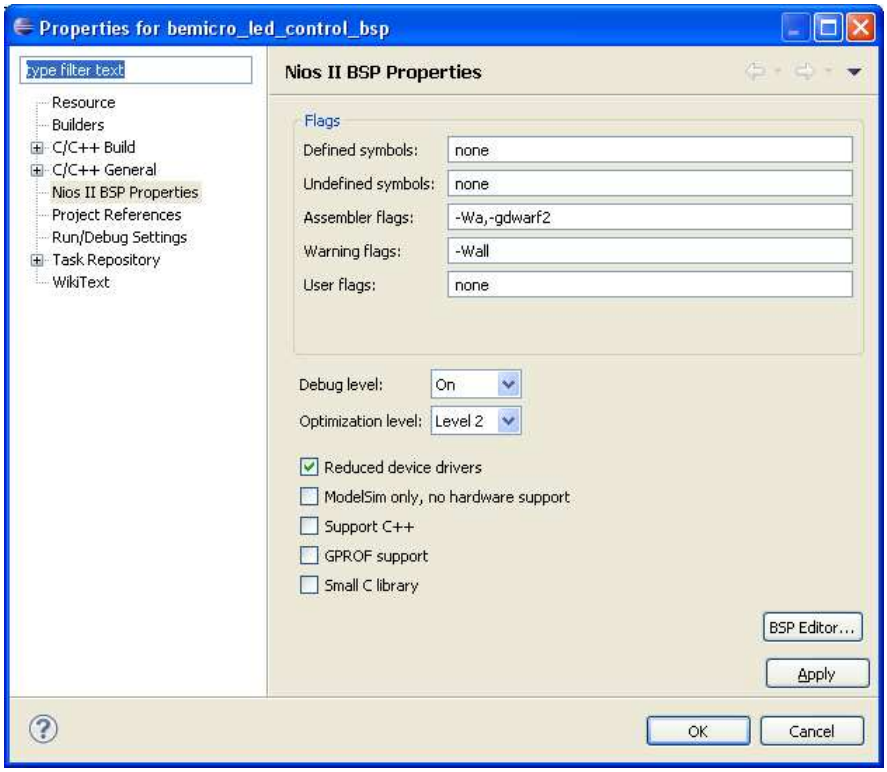
You may wish to click on the **Drivers** tab to observe how the BSP Editor gives you control over what drivers will be built into your Board Support Package. Similarly, you may wish to look over the **Linker Script** tab to observe how the BSP Editor provides you with a mechanism to adjust what memory regions the linker will utilize. In the case of this lab, we have only one volatile memory in the system, but for systems with multiple memories, this is a handy tool.

- Select **File** -> **Save** to save the board support package configuration to the **settings.bsp** file.
- Click the **Generate** button to update the BSP.
- When the generate has completed, select **File** -> **Exit** to close the BSP Editor.

6.5 Configure BSP Project Build Properties

In addition to the board support package settings configured using the BSP Editor, there are other compilation settings managed by the Eclipse environment such as compiler flags and optimization level.

- **Right click** on the **bemicro_led_control_bsp** software project and select **Properties** from the right-click menu.
- On the left-hand menu, select the **Nios II BSP Properties** tab
- During compilation, the code may have various levels of optimization which is a tradeoff between code size and performance. Change the **Optimization level** setting to **Level 2**.
- To keep the software footprint compact, choose **Reduced device drivers**.
- Since our software does not make use of C++, **uncheck** "**Support C++**".



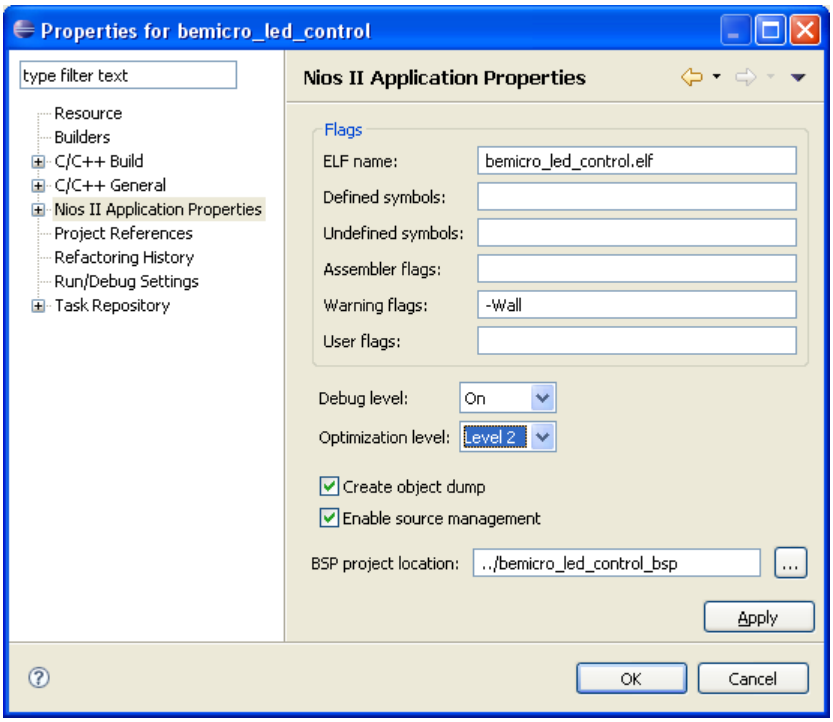
- Click **Apply**. Click **OK**.

6.6 Configure Application Project Build Properties

Just as you configured the optimization level for the BSP project, you should set the optimization level for the application software project "bemicro_led_control" as well.

- **Right click** on the **bemicro_led_control** software project and select **Properties** from the right-click menu.

- On the left-hand menu, the **Nios II Application Properties** tab
- Change the **Optimization** setting to **Level 2**.
- Click **Apply**. Click **OK**.



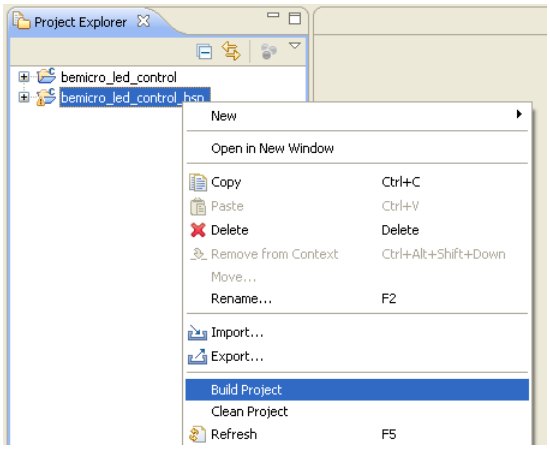
select

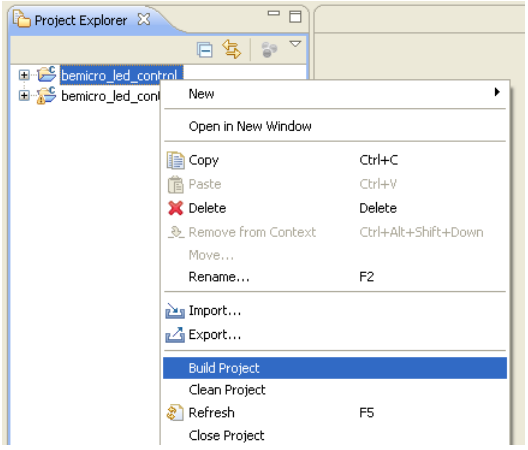
level

6.7 Build the software project

- **Right click** the **bemicro_led_control_bsp** software project and choose **Build Project** to build the board support package.
- When that build completes, **right click** the **bemicro_led_control** application software project and choose **Build Project** to build the Nios II application.

These 2 steps will compile and build the associated board support package, then the actual application software project itself. The result of the compilation process will be an Executable and Linked Format file for the application, the (*.elf) file.





6.8 Run the software application on the target

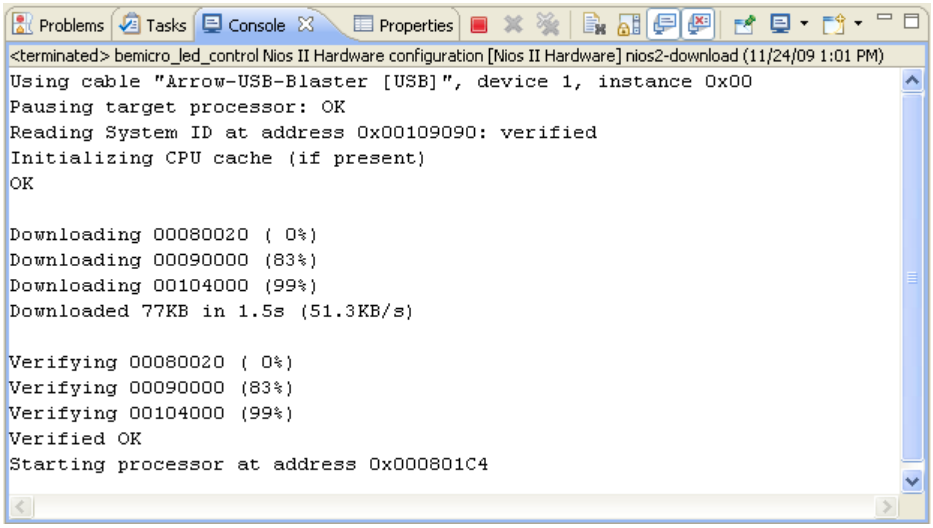
To run any application on the target hardware, two images are needed

- The FPGA hardware image SRAM Object File <.SOF>.
- The software executable the <.ELF>.

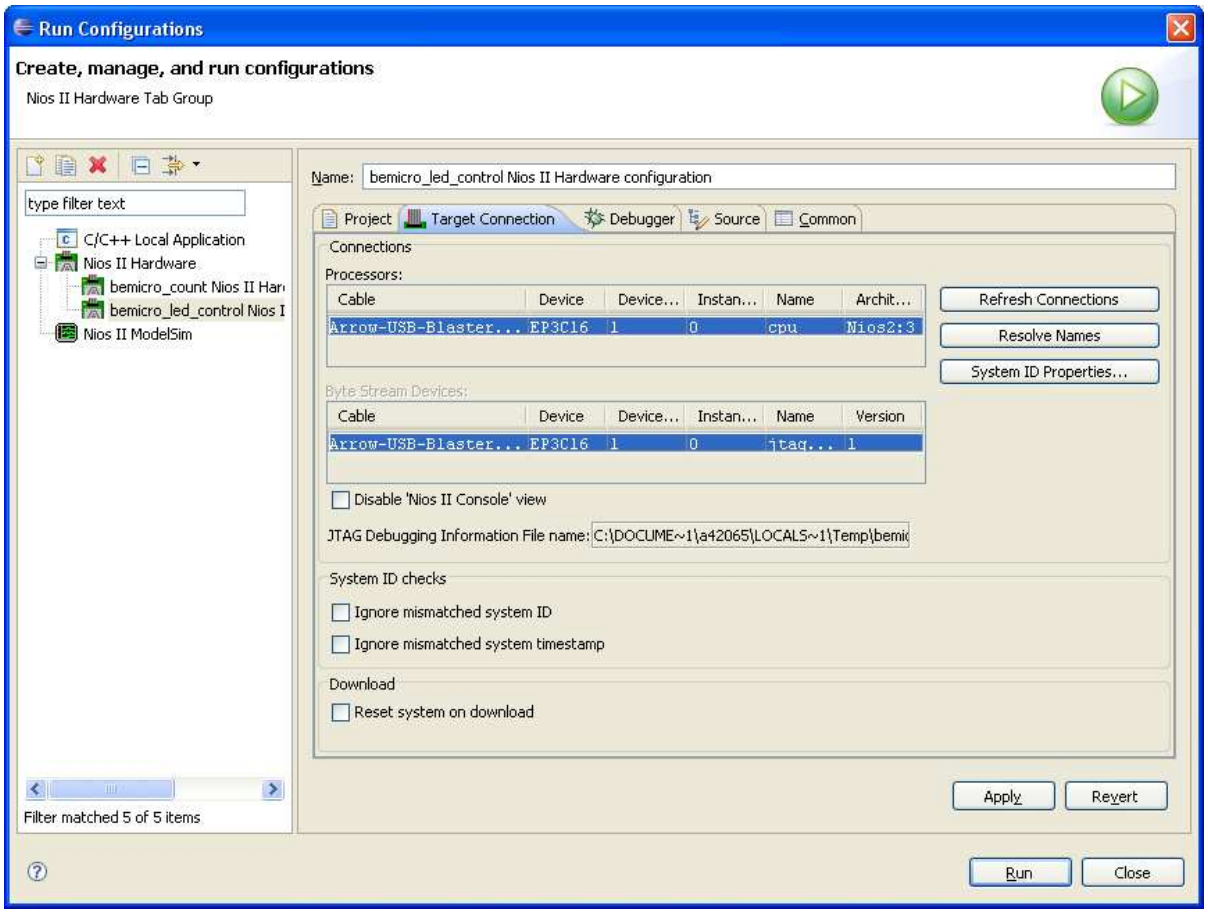
In the previous module you already downloaded the .SOF, so the FPGA is primed and ready to run the software application. Keeping the BeMicro kit still plugged into the USB port, you will download the application via the USB-JTAG link. To run the software project on the Nios II processor:

- **Right click** on the software project directory and choose **Run As** and **Nios II Hardware**.

This will re-build the software project to create an up–to-date executable and then download the code into memory on our BeMicro hardware. The debugger resets the Nios II processor, and it executes the downloaded code.

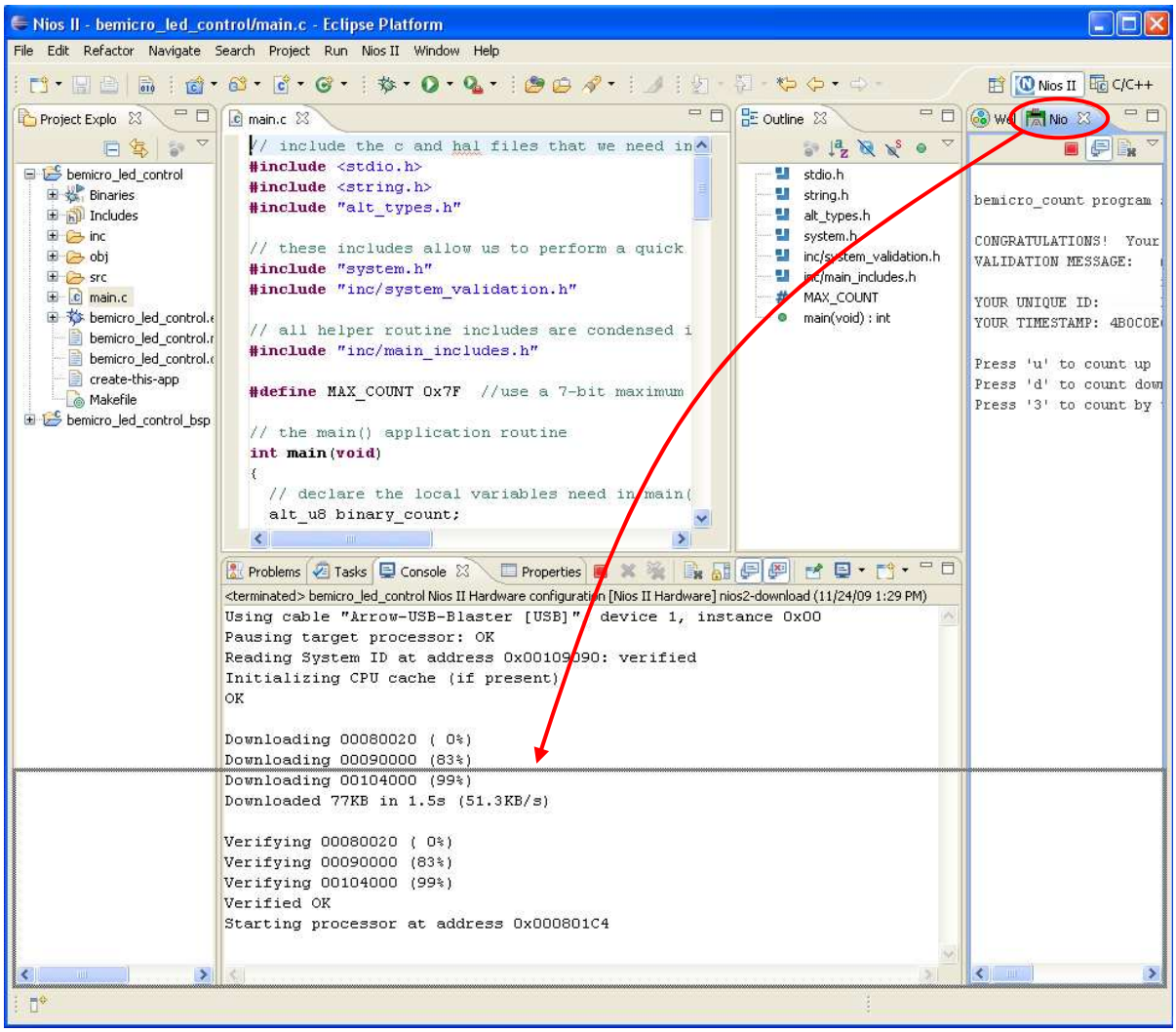


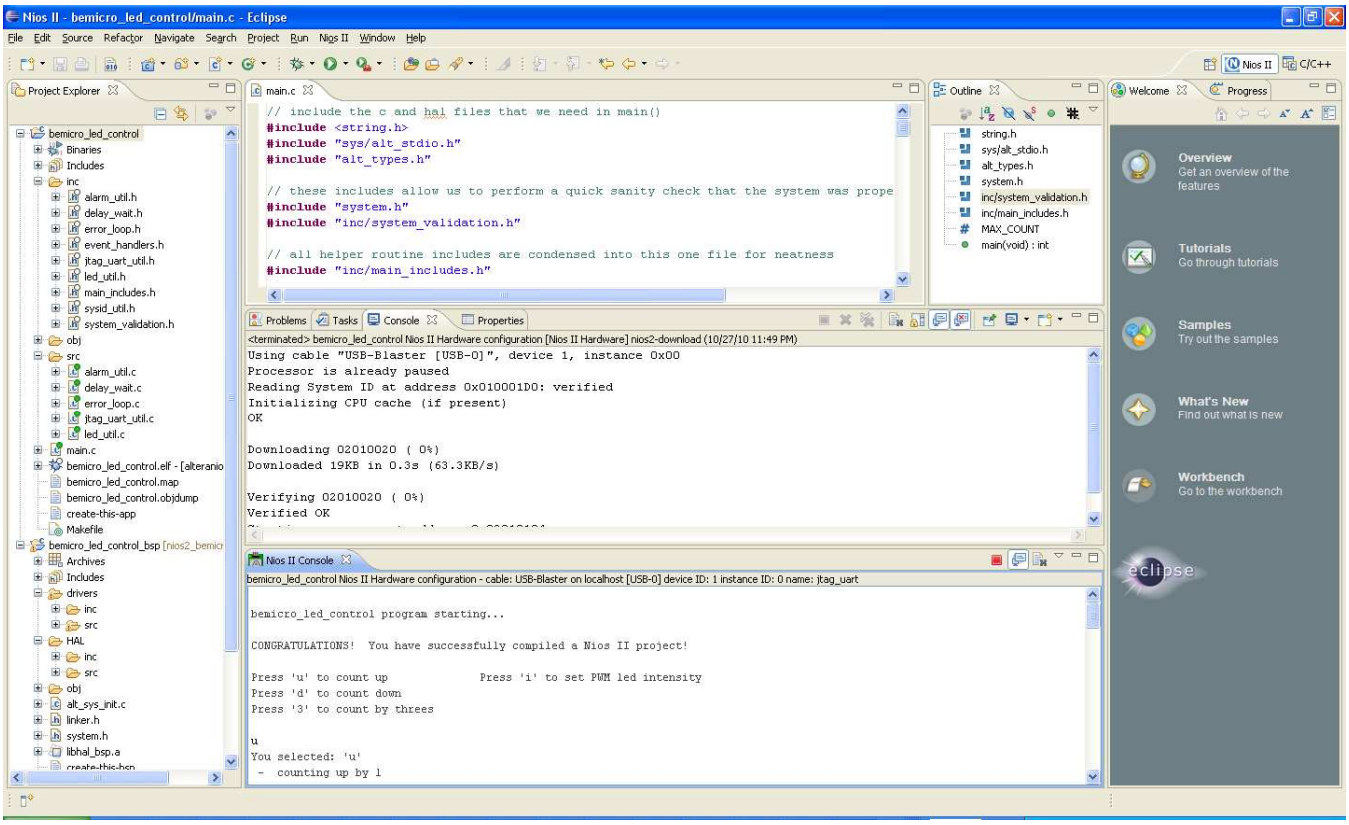
You may receive a message that your target connection could not be determined, and a “Run Configuration” dialogue window will be presented to you. If the “USB-Blaster” does not appear in the Connections lists, then scroll to the right and click on “Refresh Connections” and then select “USB-Blaster”. If more than one JTAG cable is shown in the list then be sure to select the “USB-Blaster” connection. Then click the **Apply** button and then the **Run** button.



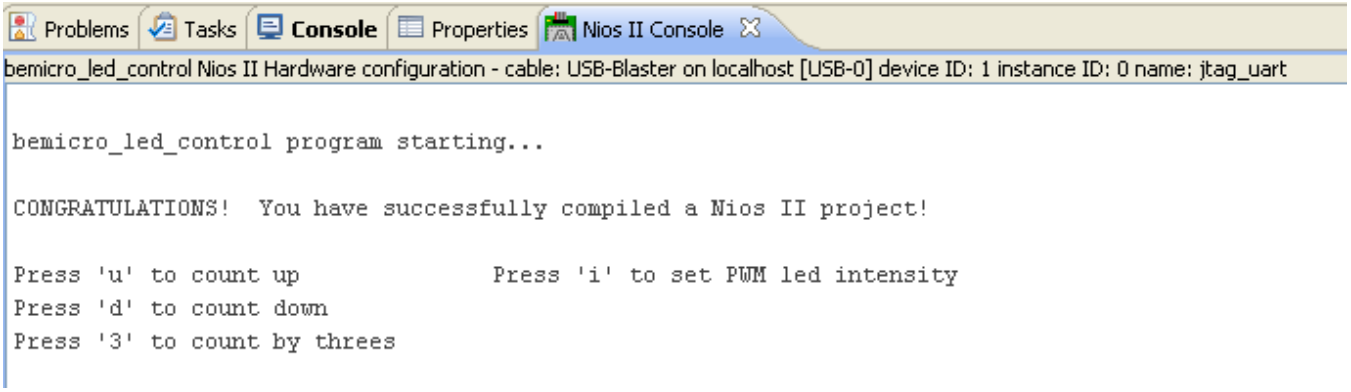
6.9 Software Application Output

Once the application starts executing, it will relay the messages back to the Nios SBT via the JTAG UART interface and the messages from this interface will be placed into a new "**Nios II Console**" pane within the Eclipse GUI. Eclipse places this console in a somewhat inconvenient location on the right of the window which is too narrow for viewing our console output properly. You will need to move this pane to another portion of the Eclipse GUI, resize the pane or maximize the pane in order to see the full console output. An example of how to move the pane is shown below.


Nios II - bemicro_led_control/main.c - Eclipse Platform
File Edit Refactor Navigate Search Project Run Nios II Window Help
Project Explo
bemicro_led_control
Binaries
Includes
inc
obj
src
main.c
create-this-app
Makefile
bemicro_led_control_bsp
main.c
// include the c and hal files that we need in
#include <stdio.h>
#include <string.h>
#include "alt_types.h"
// these includes allow us to perform a quick
#include "system.h"
#include "inc/system_validation.h"
// all helper routine includes are condensed i
#include "inc/main_includes.h"
#define MAX_COUNT 0x7F //use a 7-bit maximum
// the main() application routine
int main(void)
{
// declare the local variables need in main(
alt_u8 binary_count;
Outline
stdio.h
string.h
alt_types.h
system.h
inc/system_validation.h
inc/main_includes.h
MAX_COUNT
main(void) : int
Nios II
C/C++
bemicro_count program
CONGRATULATIONS! Your
VALIDATION MESSAGE:
YOUR UNIQUE ID:
YOUR TIMESTAMP: 4B0C0E
Press 'u' to count up
Press 'd' to count dow
Press '3' to count by
Problems Tasks Console Properties
<terminated> bemicro_led_control Nios II Hardware configuration [Nios II Hardware] nios2-download (11/24/09 1:29 PM)
Using cable "Arrow-USB-Blaster [USB]", device 1, instance 0x00
Pausing target processor: OK
Reading System ID at address 0x00109090: verified
Initializing CPU cache (if present)
OK
Downloading 00080020 (0%)
Downloading 00090000 (83%)
Downloading 00104000 (99%)
Downloaded 77KB in 1.5s (51.3KB/s)
Verifying 00080020 (0%)
Verifying 00090000 (83%)
Verifying 00104000 (99%)
Verified OK
Starting processor at address 0x000801C4



If the application is executing successfully, the console output should appear as shown below:



6.8 Interact with the Software Application

Once you have the bemicro_led_control application running on the Nios II processor, you can interact with the demo by using your keyboard to control the program flow.

6.9 Edit the Application

You can optionally modify the led_util.c source file to change the software such that the counting LEDs are inverted.

- Open the file led_util.c and locate the following subroutine:

  update_ledg( )

- In the beginning of the subroutine add the following line:
  display_value = ~ display_value;

Once this is rebuilt, the application can be rerun to see the change in LEDs.

CONGRATULATIONS!!

You have just built the software application, downloaded it to the target and run the application on the target.

Taking the Next Step

After you have sufficiently familiarized yourself with the embedded system development flow, you may want to add a SOPC Builder system to your design.

If you are starting from scratch, a good idea is to purchase a Nios II development kit, which comes with pre-generated Nios II processor systems to accelerate your development flow as well as the Nios II IP license.

If you already have a working project then you can add the SOPC builder system to your top level as a stub or even add your design to the SOPC Builder system as a custom component.

Either way you will find plenty of resources to get your job done on Altera's embedded resources at www.altera.com/embedded

Purchase an evaluation or development kit

Embedded Development Kit Resources
http://www.altera.com/products/ip/processors/nios2/kits/ni2-dev_kits.html

Get more information about the Nios II Processor

The Nios II Processor Reference Handbook
http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

Get more information about the Nios II Software Development Tools

The Nios II Software Developers Handbook
http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

Get more information about Embedded System Design

Embedded Design Guide
http://www.altera.com/literature/hb/nios2/edh_ed_handbook.pdf

Get more information about SOPC Builder and Embedded IP Peripherals

SOPC Builder (Quartus II) Handbook
http://www.altera.com/literature/lit-sop.jsp

Get Ready made Nios II Processor System Design Examples and Software Applications

Nios II Design Examples page
http://www.altera.com/support/examples/nios2/exm-nios2.html

Get Free Online Tutorials or take an In person training course

Embedded Training Resources
http://www.altera.com/technology/embedded/training/emb-training.html

Get support for your development by joining the Nios II User Community

Nios II User Community
http://www.altera.com/technology/embedded/community/emb-community.html

Get Technical Support from Altera

Embedded Technical Support
http://www.altera.com/technology/embedded/support/emb-support.html

For all resources visit www.altera.com/embedded