Table of Contents

MODULE 1: Getting Started ........................................................................................................3
  1.1 Acquire the BeMicro SDK Development Board .........................................................3
  1.2 Install the Altera Design Software ..........................................................................3
  1.3 Extract the BeMicro SDK Installation and Lab Files ..............................................6
  1.4 Install the USB-Blaster Device Driver .................................................................6

MODULE 2: Examine the System Design ...............................................................................9
  2.1 Examine the System Tool Flow ........................................................................9
  2.2 Examine the BeMicro SDK Kit .............................................................................10
  2.3 System Architecture ............................................................................................11

MODULE 3: Set Up the Quartus II Project ...........................................................................12
  3.1 Create New Quartus II Project ..............................................................................12
  3.2 Add Files to the Project ........................................................................................13
  3.3 Specify Family and Device Settings ......................................................................14
  3.4 Select EDA Tool Settings ....................................................................................15
  3.5 Execute Setup Script ............................................................................................15

MODULE 4: Build the QSys System .....................................................................................17
  4.1 Launch QSys Builder ............................................................................................17
  4.2 Manage Clocks ......................................................................................................17
  4.3 Build the QSys System ........................................................................................17
  4.4 System Configuration ...........................................................................................45
  4.5 Generate the System ............................................................................................48

MODULE 5: Complete the Quartus II Project ......................................................................50
  5.1 Set up the Quartus II Project to point to the proper timing constraint files .........50
  5.2 Download the FPGA configuration .......................................................................54

MODULE 6: Build the Software Application .........................................................................57
  6.1 Launch the Nios II Software Build Tools for Eclipse ............................................57
  6.2 Create a new software project in the SBT ............................................................58
  6.3 Add source code to the project .............................................................................59
  6.4 Configure Board Support Package .......................................................................60
  6.5 Configure BSP Project Build Properties ...............................................................61
  6.6 Configure Application Project Build Properties ..................................................62
  6.7 Build the software project .....................................................................................63
  6.8 Run the software application on the target ............................................................64
  6.9 Software Application Output ................................................................................65
  6.8 Interact with the Software Application ..............................................................68
  6.9 Edit the Application .............................................................................................68

Taking the Next Step .............................................................................................................69
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 QSys 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 v12.1, Nios® II EDS 12.1, 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
- 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 v12.1** – FPGA synthesis and compilation tool that contains QSys and the MegaCore IP library with the Nios II processor IP core

2) **Nios II EDS v12.1** – 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.

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 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>\12.1\quartus\drivers\usb-blaster”. If Windows presents you with a message that the drivers have not passed Windows Logo testing, please click “Continue Anyway”.

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

1.5 Install the Microtronix Mobile DDR SDRAM IP core

Visit the following URL and request an evaluation copy of the Mobile DDR SDRAM IP core:


Select the "Arrow BeMicro SDK" in the product drop down dialog box. After filling out the form and submitting you will receive an e-mail with a link for an opencore plus license. After submitting a link to download the Arrow BeMicro MDDR Core will be displayed:

Please download the zip file, which will include the IP core and License File. The IP core should be unzipped to the IP directory in the project directory. Next, the license file needs to be unzipped and appended to your current license file (if you have a license file) or just simply point to the unzipped file license file included in the download. In QII, select: Tools -> License Setup ...

Next point to the 4240_BeMicro_any_ON_lic.dat file or to your <mylicense>.dat file with the contents of 4240_BeMicro_any_ON_lic.dat appended to your file.
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 requires an understanding of the design flow between the QSys system integration 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 QSys. 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.

![Examine the System Design](Image)
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 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 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 QSys 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 QSys 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 12.1 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_qsys_12_1’ 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 then you may need to change directories to end up in the ./embedded_hw_lab/ directory. Select “BeMicroII_ver” as the top-level Verilog file.
Set Up the Quartus II Project

Add Files to the Project

- Click the **Browse** button and navigate to the project directory and **open** the folder entitled “bemicrosdk_embedded_hw_qsys_12_1”.

---

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_qsys_12_1”.

---

Click Next.
Set Up the Quartus II Project

- **Select** and click **Open** to add the appropriate top-level file that you chose in the previous step: BeMicroII_ver.v as well as the two reset related files, and the the_M_DDR_PLL.v file

  - BeMicroII_ver.v: This is the top-level Verilog entity for the Quartus II Project.
  - reset_counter.vhd: This is a wizard generated counter used by the reset logic.
  - reset_logic.v: This is some Verilog code to release reset after FPGA power up.

- Click **Add** to add the files listing chart.
- 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. (Make sure to click "Run" prior to clicking "OK".)
A new Quartus II TCL Console pane will appear. Ensure that there are no errors and then you can close the pane:

CONGRATULATIONS!!

Your Quartus II project is set up. You are ready to start building your QSys system.
MODULE 4: Build the QSys System

Module Objective

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

4.1 Launch QSys

- From the Tools menu, select “QSys”. There may be a slight delay while the QSys application launches.

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 QSys 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:

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

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 QSys System

1. 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.
On “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.

```
[Diagram showing ALTPLL configuration for c0]
```

“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.

```
[Diagram showing ALTPLL configuration for c1]
```
“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 Avalon 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.)
- 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.

2. **Name the Clocks**

**Reason:** We will be using the using the various clock generated by the PLL to clock the components in our system. Naming them will allow us to better manage and keep track of our clocking architecture.

- Click on the "Clock Settings" tab.

- Double 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. This just happens to match the frequency of the input oscillator on the BeMicroSDK. Leave the frequency set at 50.0.

- Similarly, double click into each of the pll_c0, pll_c1 and pll_c2 fields and rename them **pll_c0_100**, **pll_c1_60** and **pll_c2_40** respectively.

- Your Clock Settings should now appear as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext_clk_50</td>
<td>External</td>
<td>50.0</td>
</tr>
<tr>
<td>pll_c0_100</td>
<td>pll_c0</td>
<td>100.0</td>
</tr>
<tr>
<td>pll_c1_60</td>
<td>pll_c1</td>
<td>60.0</td>
</tr>
<tr>
<td>pll_c2_40</td>
<td>pll_c2</td>
<td>40.0</td>
</tr>
</tbody>
</table>

3. Connect the incoming clock and reset to the PLL

**Reason:** 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 return to the view of the components in our system. At this point, there are two components, a "Clock Source" component that was in the system by default when QSys first launched and the "Avalon ALTPLL" component that we added in the first step. 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 component.

- 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** clock output port of the Clock Source to the **inclk_interface** of the Avalon ALTPLL component. Similarly connect the **clk_reset** reset output port of the Clock Source to the **inclk_interface_reset** of the Avalon ALTPLL component. Your resulting connections should look as follows:

4. 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 **nios2_bemicro_system**.

As your work through the design, occasionally save your work using **Save** on the **File** menu.

5. **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.

6. Configure clock source for the Nios II processor

- Rename the Nios II component from "nios2_qsys_0" to "nios2_cpu".

At this point there are 3 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. Note that we made this connection with the connection dots in an earlier step. Then change the setting for the nios2_cpu to be driven by the pll_c0_100 source. Notice that the connection in the “Connections” column is automatically made for you. Either method can be used to make clock connections in QSys. Your system should now look as follows:
- Connect the pll_slave to the **data_master** of the nios2_cpu.
  Your system should now look as follows:

- Connect the nios2_cpu/reset_n to both the ext_clk_50_clk_reset and nios2_cpu/jtag_debug_module_reset
  Your system should now look as follows:
7. 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 **32768 bytes** to create a 32 KB RAM. Click **Finish.**
- Right click on the Name field and choose **Rename** from the pop up menu. Name this RAM component “onchip_ram”.

- Using the Clock column, connect the Clock Input of the onchip_ram to the pll_c0_100 clock source:

  ![Clock Connections](image)

  - Using the Connections column, connect the s1 Avalon Memory Mapped Slave interface of the onchip_ram to the nios2_cpu/instruction_master and nios2_cpu/data_master.
  - Next, connect the reset1 Reset Input to the ext_clk_50_clk_reset.
  - Then, connect the reset1 Reset Input to the nios2_cpu/jtag_debug_module_reset

8. **Add an Microtronix Multi-port Mobile DDR Controller**

   Expand **Microtronix** and double click on **On-Multi-port Mobile DDR Controller**.
Set Port A of Port Type Burst, Buffer Size 16 Words and Max Burst Length 128 Words
Set Port B of Port Type Burst, Buffer Size 16 Words and Max Burst Length 128 Words
Set Memory to the following: **Total Data Width**: 16 bits, **Bank Address Bits**: 2, **Row Address Bits**: 13, **Column Address bits**: 10. **Prefix**: `RAM_`, **Total number of discrete memory devices**: 1, **Total Number of Clock Pairs**: 1, **DQS Pins per Discrete Memory Device**: 2, **Custom Scheduler Parameter**: 0.
Set Timing to the following: **SDRAM Freq**: 133, **Memory Init Time**: 200, **Precharge Command Period**: 15, **Load Mode Register Command Period**: 15, **Auto Refresh Command Period**: 72, **Average Periodic Refresh Interval**: 7800, **Active Write or Read Delay**: 15, **Active to Precharge Command**: 45, **Active to Active Command Period**: 55 ns.
Select Finish.

Rename mtx_avalon_sdram to mobile_ddr_sdram

Connect as shown below: port_a: data_master of nios2_cpu, port_b: instruction_master of nios2_cpu.

Connect the Clock Input of both clock_port_a and clock_port_b to pll_c0_100

Then, connect both reset_porta and reset_portb to: ext_clk_50_clk_reset

Next, connect both reset_porta and reset_portb Reset Inputs to the nios2_cpu/jtag_debug_module_reset

Finally, click in the "click to export" field next to the sdram_interface conduit and name: mobile_ddr_sdram_external

9. 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 **Memory Interfaces**. 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**.

- In the clock column, select **pll_c0_100** as the clock for the `epcs_flash_controller`.

- Connect **reset** Reset Input: `ext_clk_50/clk_reset`

- Connect **reset** Reset Input to the `nios2_cpu/jtag_debug_module_reset`

- Leave the Conduit Endpoint blank for the **epcs_flash_controller**.
- Connect the `epcs_control_port` to both `nios2_cpu/data_master` and `nios2_cpu/instruction_master`.

10. 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 **Address Units** to WORDS. Change the **Command FIFO depth** to 8 and the **Response FIFO depth** to 32 and both the synchronizer depths to 3.

- Right click on the Name field and choose **Rename** from the pop up menu. Name this bridge "slow_periph_bridge".
  
  Connect the nios2_cpu’s instruction_master and data_master ports to the s0 Avalon Memory Mapped Slave of this bridge. The m0 master port will be connected in the upcoming steps.

Connect s0_reset and m0_reset **Reset Input**: ext_clk_50/clk_reset

Connect s0_reset and m0_reset **Reset Input** to the nios2_cpu/jtag_debug_module_reset

In the clock column, select **pll_c0_100** as the clock for the s0_clk
In the clock column, select **pll_c2_40** as the clock for the m0_clk
11. 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**.

![Interval Timer Configuration](image)

- **Rename** the component “**sys_timer**”.
- Change the connection on the s1 slave port of the peripheral to be connected to the m0 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 m0 port.

- Connect **reset** Reset Input: **ext_clk_50/clk_reset**
- Connect **reset** Reset Input to the nios2_cpu/jtag_debug_module_reset
**12. 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 m0 master port of the slow_periph_bridge.
- Connect reset Reset Input: ext_clk_50/clk_reset
- Connect reset Reset Input to the nios2_cpu/jtag_debug_module_reset
- In the clock column, select **pll_c2_40** as the clock for the clk Clock Input

**13. 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.
1. Rename the peripheral “led_pio”.

2. Change the connection on the s1 slave port of the peripheral to be connected to the m0 master port of the slow_periph_bridge.

3. Connect reset Reset Input: ext_clk_50/clk_reset

4. Connect reset Reset Input to the nios2_cpu/jtag_debug_module_reset

5. In the clock column, select pll_c2_40 as the clock for the clk Clock Input

6. Finally, click in the "click to export" field next to the external_connection Conduit Endpoint and name: led_pio_external

14. 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”.
- Change the connection on the s1 slave port of the peripheral to be connected to the m0 master port of the slow_periph_bridge.
- Connect `clock_reset_reset` Reset Input: ext_clk_50/clk_reset
- Connect `clock_reset_reset` Reset Input to the nios2_cpu/jtag_debug_module_reset
- In the clock column, select `pll_c2_40` as the clock for the `clock_reset` Clock Input
- Finally, click in the "click to export" field next to the external_connection Conduit Endpoint and name: `led_pwm_external`

**15. 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”.
- Check the **Synchronously capture** and **Any edge** options.
- On the Simulation tab, check the **Hardwire PIO inputs in the test bench** option and drive the inputs to **0x3**.
Build the QSys System

- Click Finish
- Rename the peripheral “dipsw_pio”.
- Change the connection on the s1 slave port of the peripheral to be connected to the m0 master port of the slow_periph_bridge.
- Connect reset Reset Input: ext_clk_50/clk_reset
- Connect reset Reset Input to the nios2_cpu/jtag_debug_module_reset
- In the clock column, select pll_c2_40 as the clock for the clk Clock Input
- Finally, click in the "click to export" field next to the external_connection Conduit Endpoint and name: dipsw_pio_external

16. 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”.
- Check the Synchronously capture and Falling edge options 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 0x1
- Click **Finish**

- **Rename** the peripheral “user_pio_pushbtn”.
- Change the connection on the s1 slave port of the peripheral to be connected to the m0 master port of the slow_periph_bridge.
- Connect **reset** Reset Input: ext_clk_50/clk_reset
- Connect **reset** Reset Input to the nios2_cpu/jtag_debug_module_reset
- In the clock column, select **pll_c2_40** as the clock for the **clk** Clock Input
- Finally, click in the "click to export" field next to the external_connection Conduit Endpoint and name: **user_pio_pushbtn_external**
17. 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”.
- Change the connection on the s1 slave port of the peripheral to be connected to the m0 master port of the slow_periph_bridge.
- Connect **reset** Reset Input: ext_clk_50/clk_reset
- Connect **reset** Reset Input to the nios2_cpu/jtag_debug_module_reset
- In the clock column, select **pll_c2_40** as the clock for the clk Clock Input
- Finally, click in the “click to export” field next to the external_connection Conduit Endpoint and name: **temp_sense_spi_external**
18. 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”.
- Change the connection on the s1 slave port of the peripheral to be connected to the m0 master port of the slow_periph_bridge.
- Connect reset Reset Input: ext_clk_50/clk_reset
- Connect reset Reset Input to the nios2_cpu/jtag_debug_module_reset
- In the clock column, select pll_c2_40 as the clock for the clk Clock Input
19. 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.
- Change the connection on the s1 slave port of the peripheral to be connected to the m0 master port of the slow_periph_bridge.
- Connect **reset** Reset Input: ext_clk_50/clk_reset
- Connect **reset** Reset Input to the nios2_cpu/jtag_debug_module_reset
- In the clock column, select **pll_c2_40** as the clock for the **clk** Clock Input

![System ID Peripheral Diagram](image)

20. 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 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 m0 master port of the slow_periph_bridge.
• Connect global_signals_clock_reset Reset Input: ext_clk_50/clk_reset
• Connect global_signals_clock_reset Reset Input to the nios2_cpu/jtag_debug_module_reset
• In the clock column, select pll_c2_40 as the clock for the clk Clock Input

At this point all the components to the QSys 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_c6 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.

**Make sure you connect the IRQs 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**

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

- First, select the epcs_flash_controller address and set to 0x0, then lock it by clicking on the lock icon.

- From the System menu, choose Assign Base Addresses. The tool will assign appropriate base addresses for the components by taking their widths into consideration.
From the System menu, choose Assign Interrupt Numbers. 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 nios2_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 mobile_ddr_sdram_port_b 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 QSys System.

THE RESULTING QSYS SYSTEM
4.5 Generate the System

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

Select the "Generation Tab" and then click the Generate button. If it asks you to save select Yes.

QSys 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 QSys 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 QSys system!
MODULE 5: Complete the Quartus II Project

Module Objective

In this module you complete the Quartus II project by adding the generated QSys system to the top-level entity. 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 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).

QSys 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.
- Select TimeQuest Timing Analyzer under category
- 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.
- In the “SDC files to include in the project” click on the “…” and select the mobile_ddr_sdram.sdc file found in the project directory. Click on the “Add” button to add it to the list of SDC files.
- In the “SDC files to include in the project” click on the “…” and select the nios2_bemicro_system.qip file found in the nios2_bemicro_system/synthesis directory. Click on the “Add” button to add it to the list of files.
Select "Apply" and then "OK"
Next add the nios2_bemicro_system synthesis directories to the project.

Select, Project -> Add Remove Files in Project... then select the Libraries Category, next select the "..." button and add the library ..\nios2_bemicro_system\synthesis next add the library ..\nios2_bemicro_system\synthesis\submodules

Next add the nios2_bemicro_system.qip to the project.

Select the Files Category, next select the "..." and select the nios2_bemicro_system.qip file found in the nios2_bemicro_system/synthesis directory. Click on the “Add” button to add it to the list of files.
Select "Apply" and "OK"

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 assignments/features and incomplete pin assignments but these will not affect the functionality of the system.

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

**WARNING YOU WILL RECEIVE AN ERROR** in the report if you don’t have a permanent license for the Microtronix IP core. This since a time limited file is created vs. a permanent SRAM Object file (*.sof). **BUT THIS IS OK since the BeMicroII_time_limited.sof is created!**

### 5.2 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 **USB-Blaster**. Click **Close**.

After clicking the **Auto Detect** button you should find that the programmer detects the **EP4CE22** device on your BeMicro SDK and the BeMicroII_ver_time_limited.sof file should be in the File field.

If the `bemicro_lab_***.sof` is not in the file field, then Double click on the `<none>` in the File field, or select `<none>` and click on the **Change File** button. Finally, 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 12.1 -> Nios II 12.1 Software Build Tools for Eclipse** or alternatively it can be launched from the **QSys -> 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 QSys.

- **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”. Click “OK”.

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 QSys Information File, click the Browse button to locate the \textit{nios2\_bemicro\_system\_sopcinfo} file located in the Quartus II project directory.
- Set the name of the Application project to \textit{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 QSys system. This project inherits the name from the main software project and appends "_bsp" to that.

![Project Explorer](image)

**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 in the explorer window, right mouse click and then select **copy**.
- **Then select** the “**bemicro_led_control**” directory in the SBT window and **paste** the files onto the project folder.

![Paste Files](image)

Select the directory: bemicro_led_control_src, then select **OK**.
6.4 Configure Board Support Package

- Configure the board support package 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 subsections (Program memory, Read-only data memory, Read/write data memory) will be stored into mobile_ddr_sdram. We will point the linker to place the heap and stacks in mobile_ddr_sdram 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 mobile_ddr_sdram 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 `bemicroLedControl_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++".

---

![Image of Nios II BSP Properties](image.png)

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, select the Nios II Application Properties tab.

Change the Optimization level setting to Level 2.

Click Apply. Click OK.

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.
Build the Software Application
If the application is executing successfully, the console output should appear as shown below:

```console
bemicro_ied_control NiosII Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtag_uart

bemicro_ied_control program starting...

CONGRATULATIONS! You have successfully compiled a Nios II project!

Press 'u' to count up          Press 'i' to set PWM led intensity
Press 'd' to count down       Press '3' to count by threes
```

Downloaded from Arrow.com.
6.8  **Interact with the Software Application**

Once you have the bemicroLedControl 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:
  ```c
  update_ledg()
  ```
- In the beginning of the subroutine add the following line:
  ```c
  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 QSys 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 QSys system to your top level as a stub or even add your design to the QSys 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

Get more information about the Nios II Processor

The Nios II Processor Reference Handbook

Get more information about the Nios II Software Development Tools

The Nios II Software Developers Handbook

Get more information about Embedded System Design

Embedded Design Guide

Get more information about QSys and Embedded IP Peripherals

QSys (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

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