BeMicro SDK Lab Instructions

ARROW ELECTRONICS, INC.

November 2010, Version 10.0 | 11/12/2010 | Tutorial

Table of Contents

This page intentionally left blank

Overview

Learning to Use the Nios II Embedded Development Suite

This lab teaches you how to use the Nios II Embedded Development Suite (EDS) to develop code for embedded solutions running on the Nios II processor in Altera FPGA's. You will learn how to navigate the tool and to Run and Debug code on an Embedded Target board, the BeMicroSDK.

In addition you will learn how Hardware accelerators can be used to vastly enhance the computational throughput of the embedded target. You will also learn about Micrium's uC/Probe debug tool that allows real time instrumentation of your embedded design.

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

MODULE 1: Getting Started

Module Objective

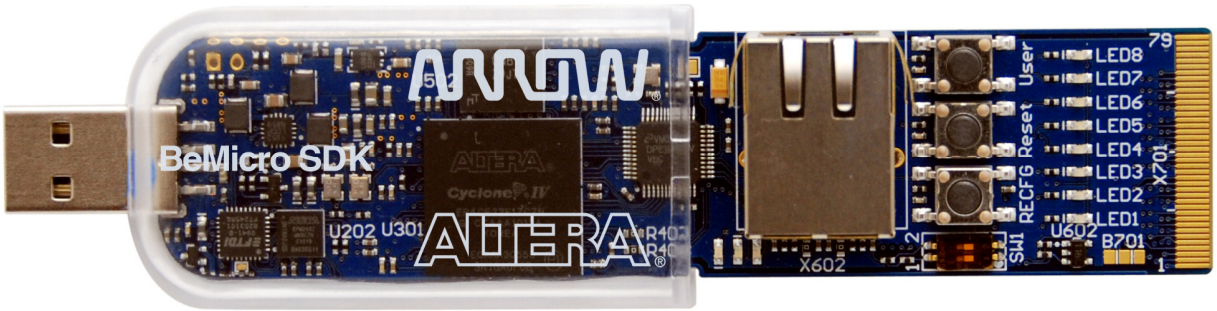
Your first objective is to ensure that you have all of the items needed and to install the tools so that you are ready to create and run your design.

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 is available via the Altera Complete Design Suite DVD or by downloading from the web.

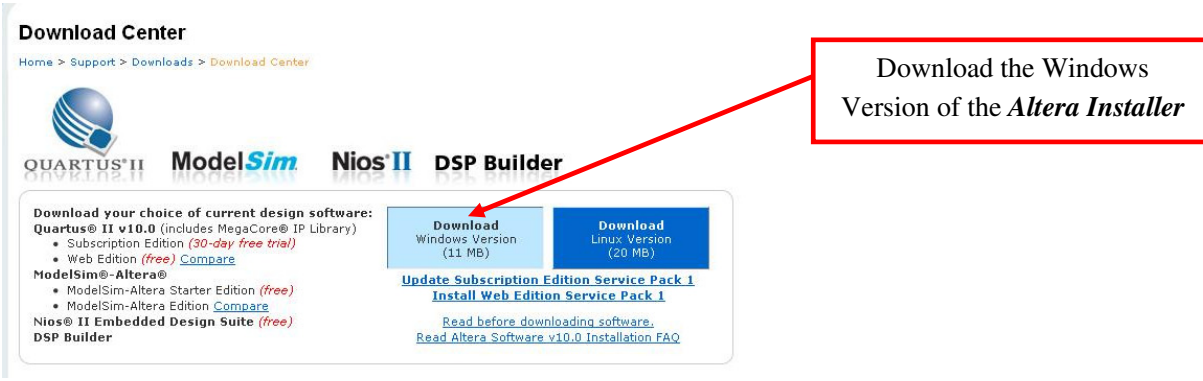
If you already have both Quartus II and the Nios II EDS installed on your machine, you may skip ahead to Section 1.3 to install Micrium uC/Probe Software.

INSTALLING FROM THE DVD-ROM: Select **Install Free Package** and Press **Next** until you reach the page titled **End User License Agreement**. Click on the **checkbox** at the **bottom** of the page. Press **Next** until you reach the page titled **Select Components**. Please skip ahead to step 4 of the installation instructions.

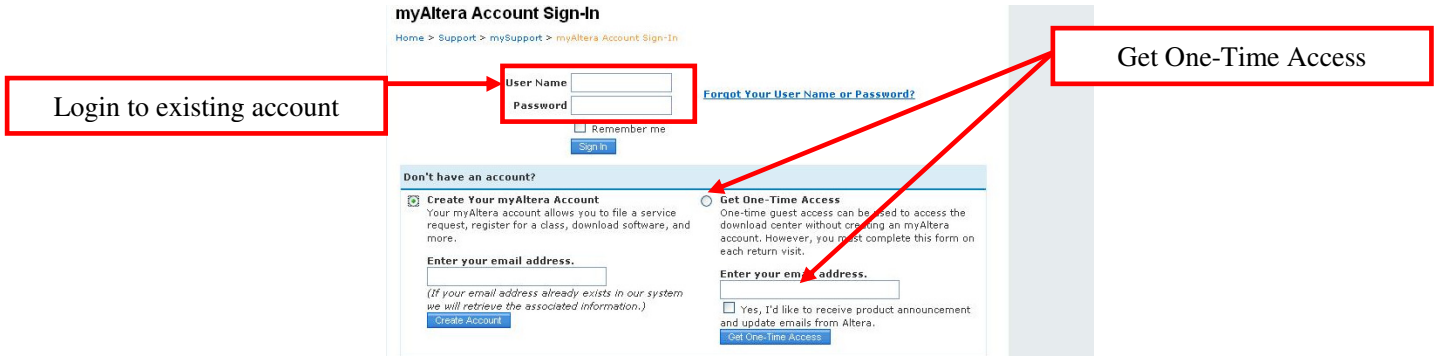
INSTALLING FROM THE WEB: Please follow steps 1 through 4 of the installation instructions.

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

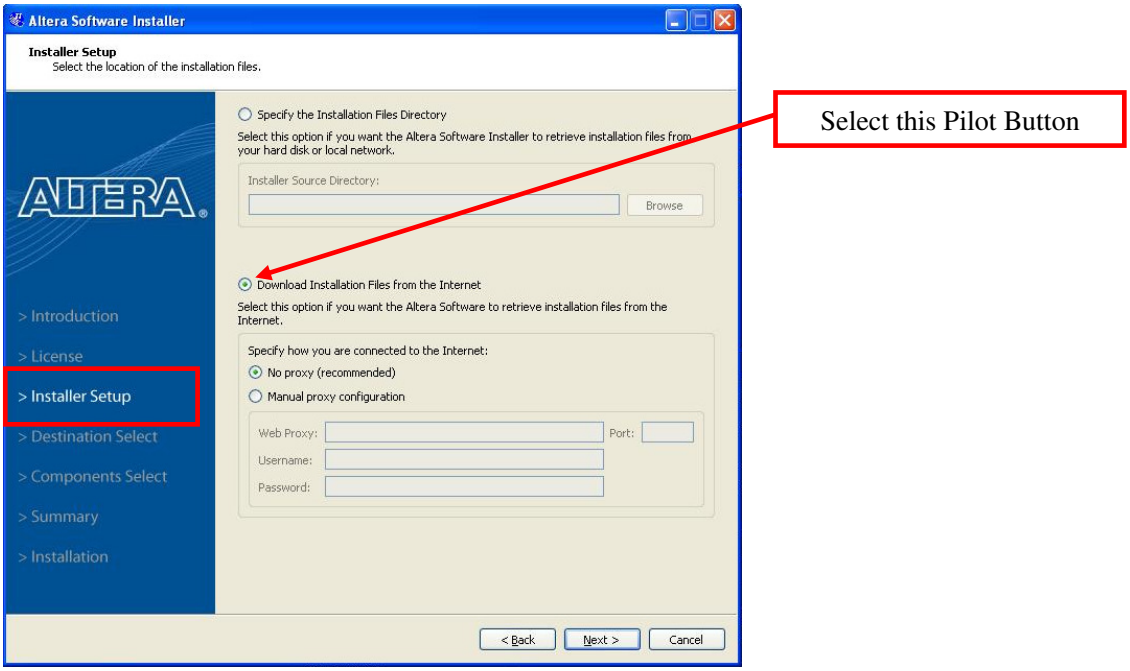
1. Go to the Altera Download web page at https://www.altera.com/download/dnl-index.jsp



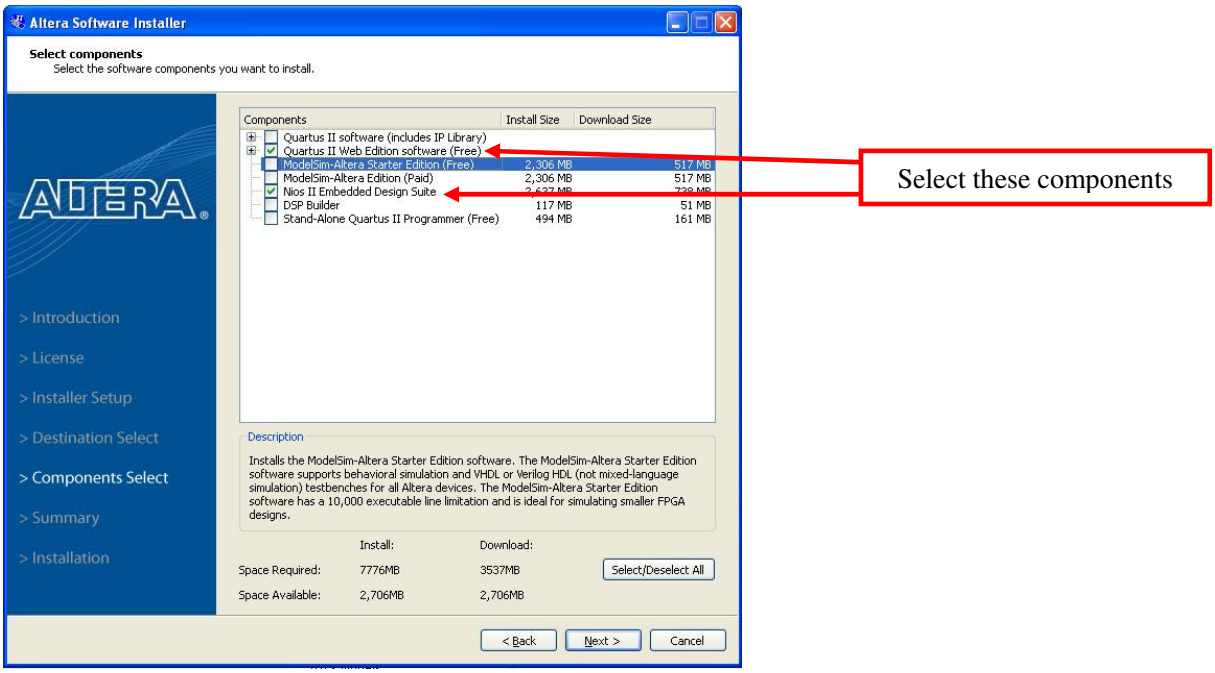
2. Login to myAltera account. Use your existing login, **or** get ***One-Time*** Access



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

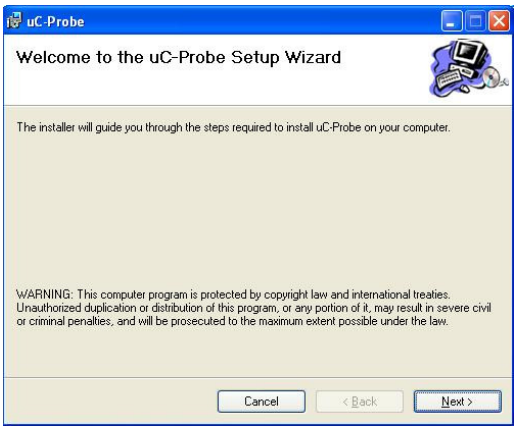


4. Select Components. Select ***Quartus II Software Web Edition*** and ***Nios II Embedded Design Suite components*** for download.



1.3 Install the Micrium uC/Probe Software

The Micrium uC/Probe Trial version is available via download from the web at http://micrium.com/download/Micrium-uC-Probe-Setup-Trial.exe. Download, save and run the setup file and then follow the instruction guidelines.



This Trial version does not expire and allows up to a maximum of eight application symbols.

1.4 Extract the BeMicro SDK Installation and Lab Files

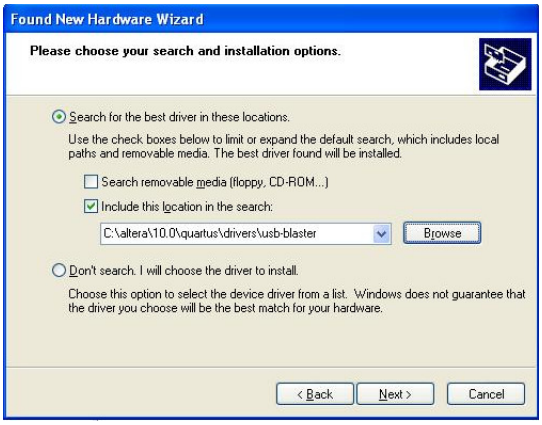
Create a folder called **altera_trn** on your PC (eg. C:\altera_trn). Download the BeMicroSDK.zip ZIP archive from the http://www.arrow.com/bemicrosdk web page and then extract to the **altera_trn** folder on your PC. Make sure that there are ***NO SPACES*** in the directory path.

1.5 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 dialog 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!!

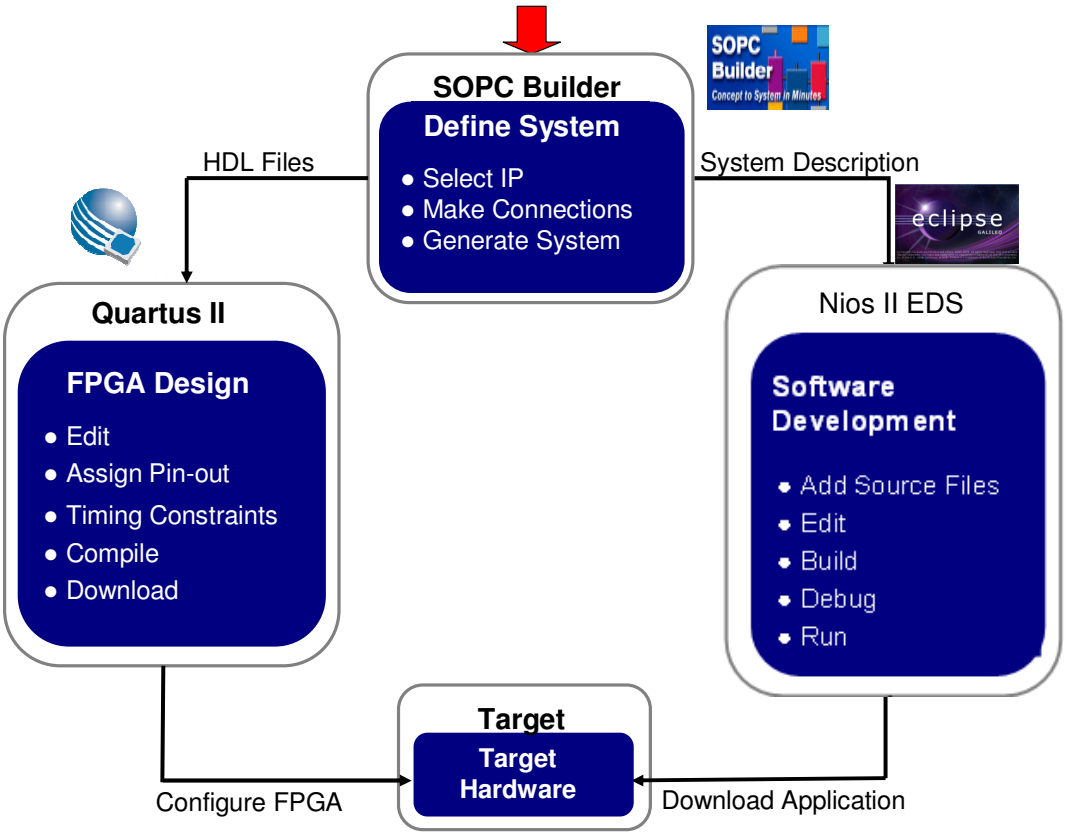
You have just completed all the setup and installation requirements and are now ready to examine the system-level design.

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



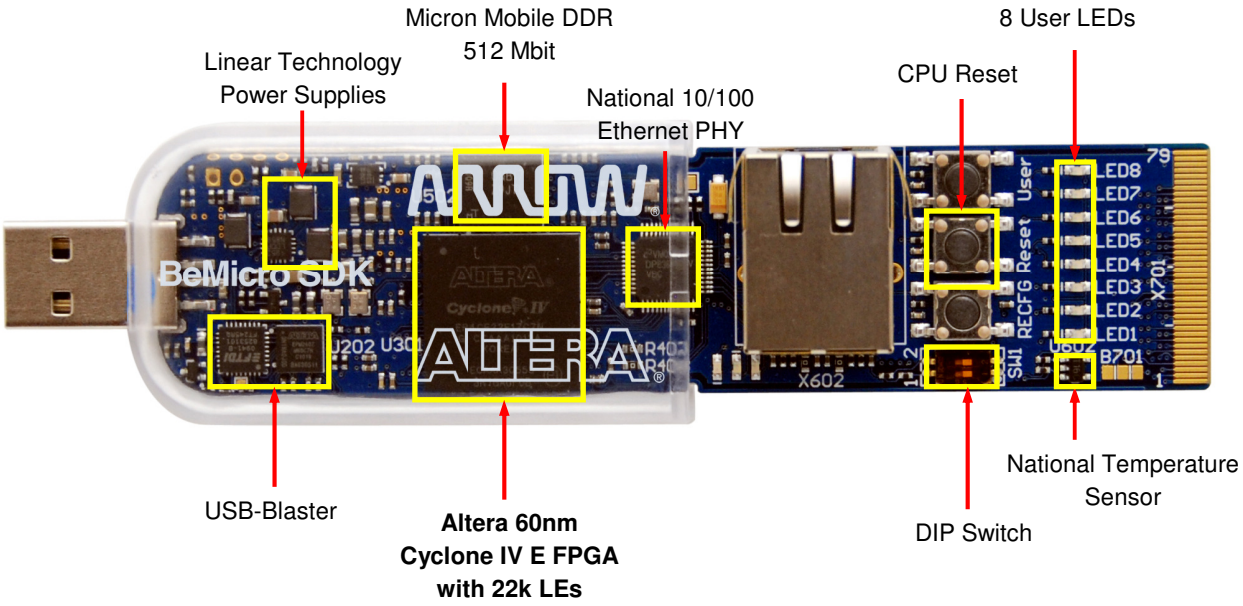
The above diagram depicts the typical flow for system design. System definition is performed using SOPC Builder. The results are two-fold:

- System description that the Nios II Embedded Development Suite, 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

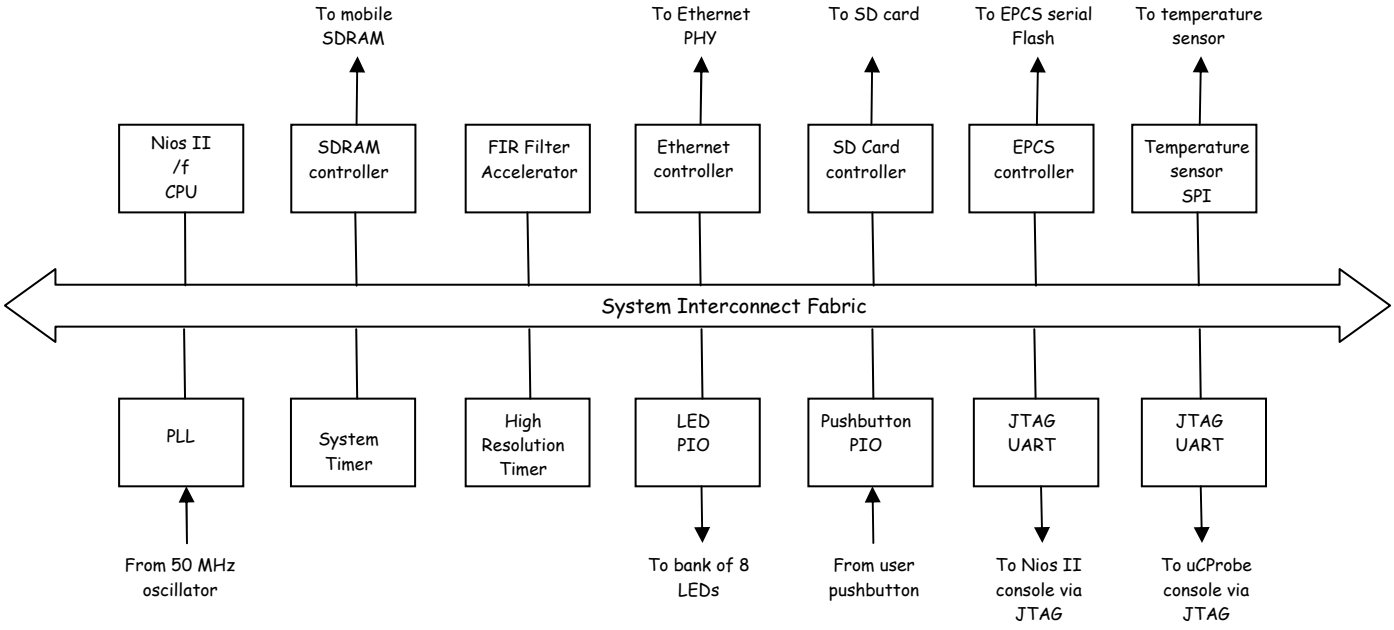
Examine the components on the BeMicro SDK board hardware:



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:



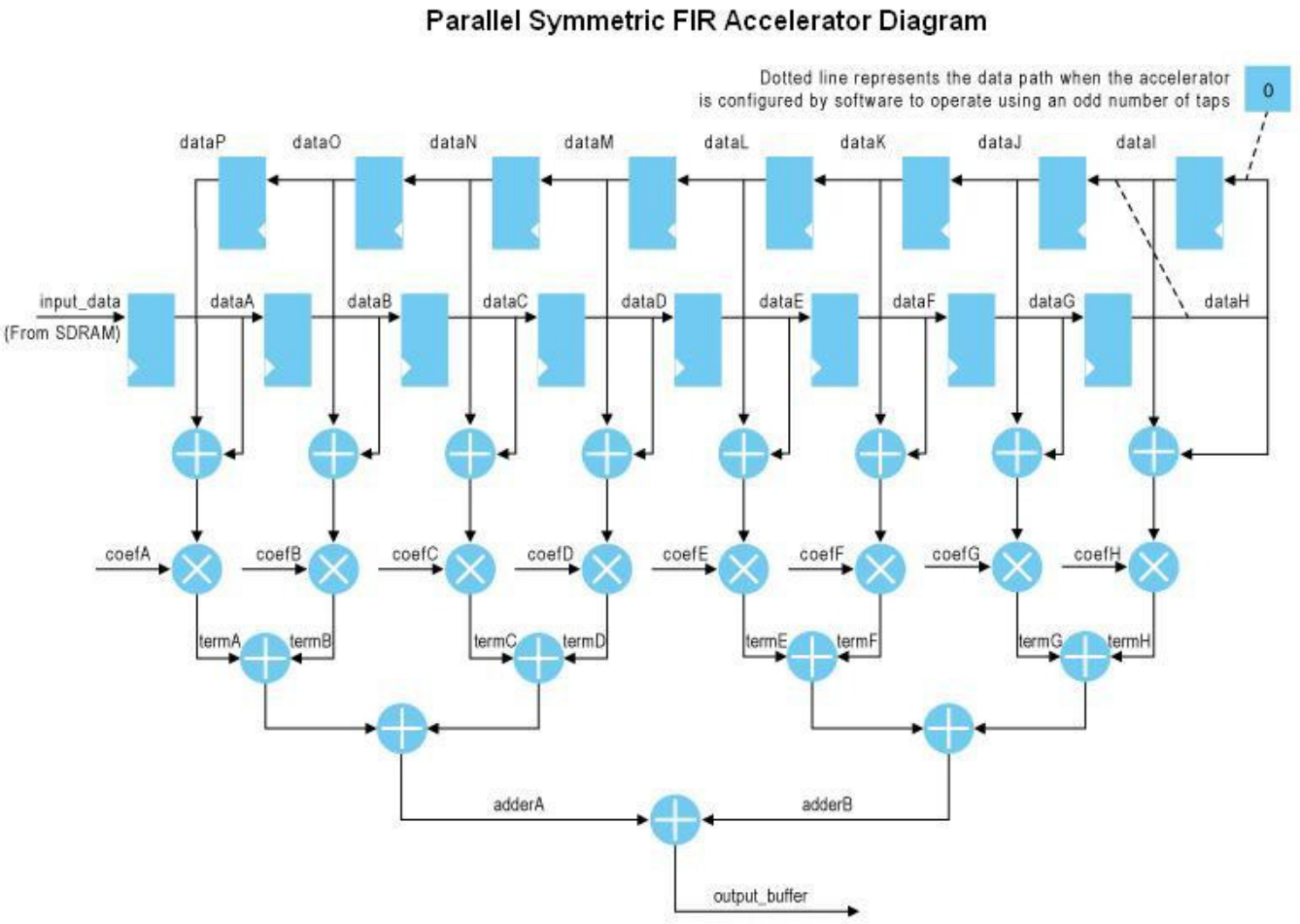
The system was entered in SOPC Builder using a standard library of re-useable IP blocks. The System Interconnect Fabric is automatically generated by SOPC Builder and binds the blocks together. The system interconnect manages dynamic bus-width matching, interrupt priorities, arbitration and address mapping.

This system is capable of running operating systems such as uC-OSII or Linux. Today you will be using the system description file provided by SOPC Builder to create a software application for the architecture shown in the sketch. You will also become familiar with the development and debug features available in the Nios II EDS.

2.4 Software Lab

The software lab implements a finite impulse response (FIR) filter with both a software function and its hardware accelerator equivalent. The flexibility of the FPGA makes it possible to add custom accelerators to any embedded design to improve computational throughput.

A FIR filter can be constructed using coefficient constants, adders , multipliers and registers for storage. The FIR filter structure is shown below.



This FIR filter has the following characteristics

- Low pass with a cutoff frequency of 1.25MHz
- 15 Taps
- Hanning Window with a sampling rate of 10^7
- Symmetrical

The input data is comprised of two sine waves at low and high frequencies

- First Sine: Amplitude = 100, Frequency = 2.5MHz
- Second Sine: Amplitude = 1000, Frequency = 156.25KHz

The input data is fed through the filter architecture and the results are moved to a destination data-buffer. The input data is separately filtered by both the software and hardware filters. The results from each are written to data-buffers in external memory and the results are compared for exactness.

Finally the time taken to compute the filter results is measured for both the software and hardware filters. The time for each is compared and an acceleration factor is computed.

There are two main files that we will use in the lab. lab1.c implements the software FIR filter while lab2.c implements both the software and hardware FIR filters. The filter functions are located in the src subdirectory. The uC-CPU, uC-LIB and uC-Probe folders contain all of the embedded code needed to support the uC/Probe on the embedded target.

CONGRATULATIONS!!

You now understand the architecture of the BeMicro SDK and the tool flow used to create it.

MODULE 3: Create a New Project

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.

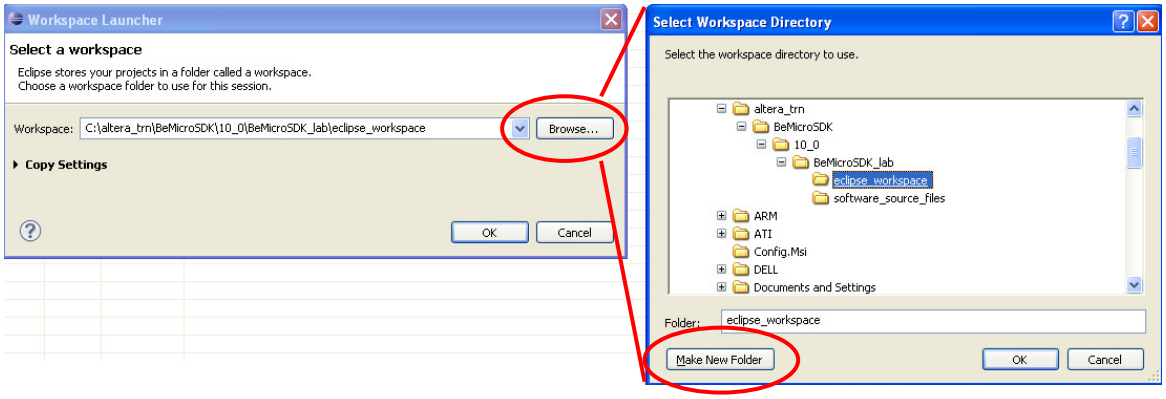
3.1 Launch the Nios II Software Build Tools for Eclipse

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**
NOTE: Windows 7 users will need to right-click and select "Run as Administrator". Another method is to right-click and select Properties and click on the Compatibility tab and select the Run This Program As An Administrator check box, which will make this a permanent change.

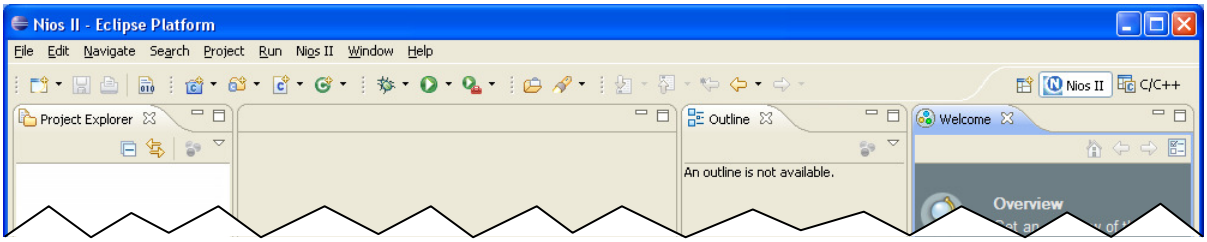
1. Initialize Eclipse workspace

When Eclipse first launches, a dialog 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 BeMicroSDK_lab sub-directory and click **Make New Folder** to create a folder for your software project. Name the folder "eclipse_workspace".

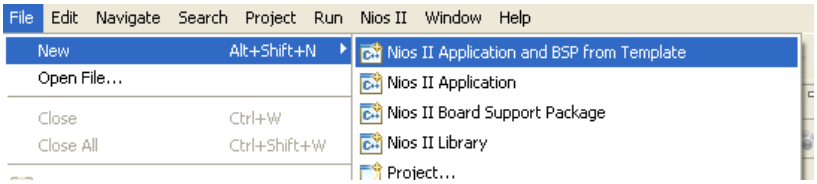


After selecting the workspace directory, click "**OK**" and Eclipse will launch and the workbench will appear in the Nios II perspective.



3.2 Create a new software project in the SBT

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



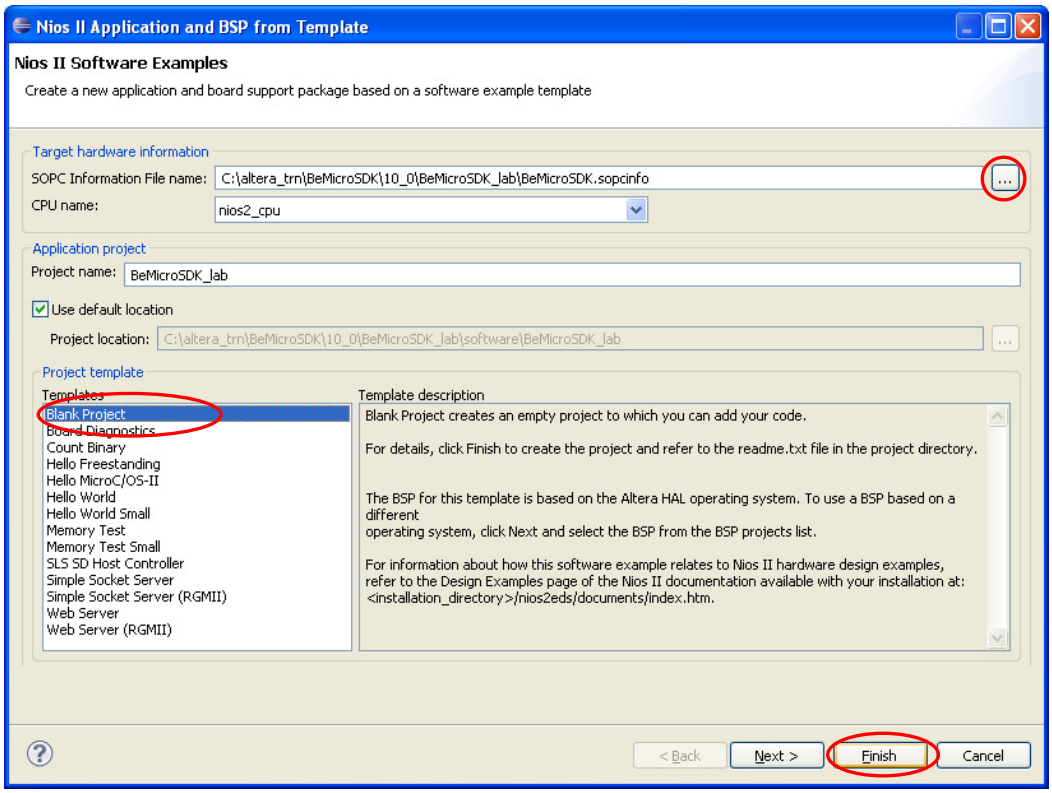
This lab is based on a hardware design that was created using Altera's SOPC Builder tool. SOPC Builder creates a hardware description file called a .sopcinfo file. This file describes all the hardware components within the design as well as the memory map and interrupt structure. The Nios II SBT tools use the .sopcinfo file to build the appropriate drivers for the Board Support Package (BSP) project

Define the Target Hardware Information

- Click the **Browse** button in the **SOPC Information File Name** dialog box.
- Select the **BeMicroSDK.sopcinfo** file located in **BeMicroSDK**_lab directory.

Choose a Project Template

- Set the name of the **Application** project to "**BeMicroSDK_lab**".
- Select the **Blank Project** template under **Project Template**.
- Click the **Finish** button.

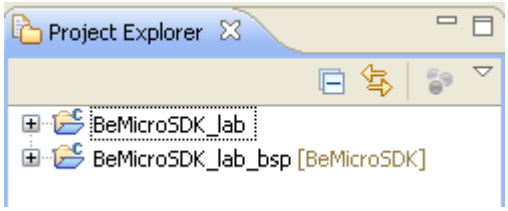


The tool will create two new software project directories

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.



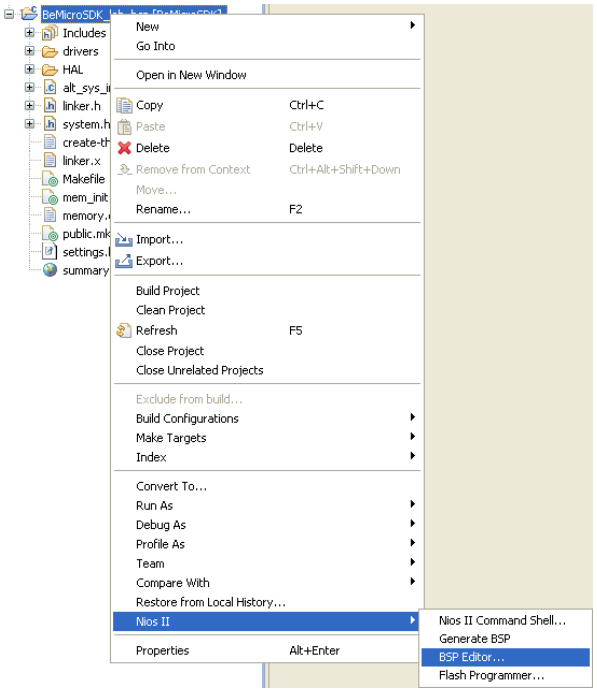
Initial content of the project

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.

3.3 Configure the 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, the memory in which stack and heap should be allocated and whether an operating system or network stack should be included with this BSP.

- Right click on the **BeMicroSDK_lab_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**.

Select the "Common" settings view. In the "Common" settings view, **change** the following settings:

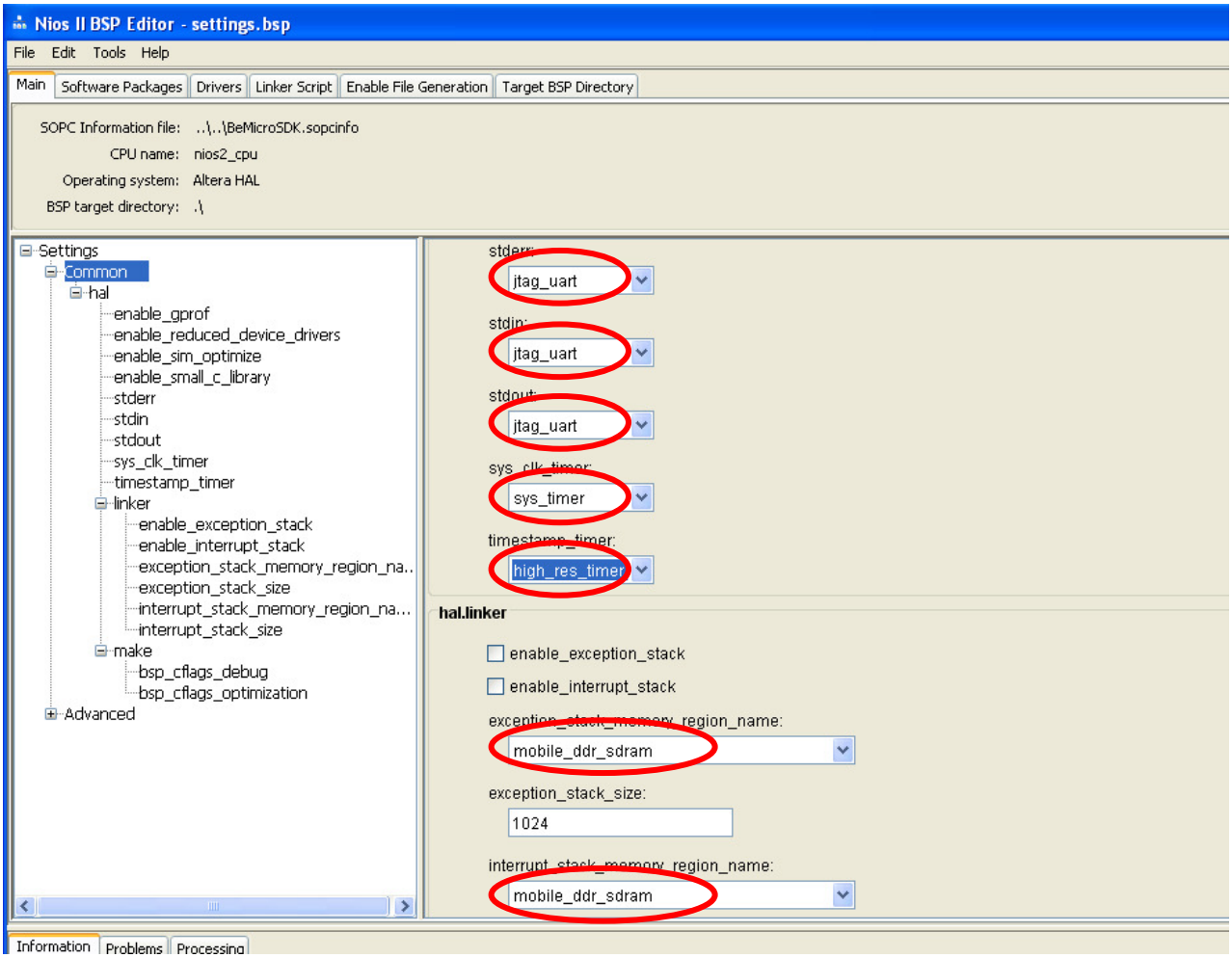
1. Select the **jtag_uart** for **stdin**, **stdout** and **stderr** messages. Note that you have more than one choice.

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

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

4. Select **mobile_ddr_sdram** as the linker target for **exception_stack_memory_region_name**.

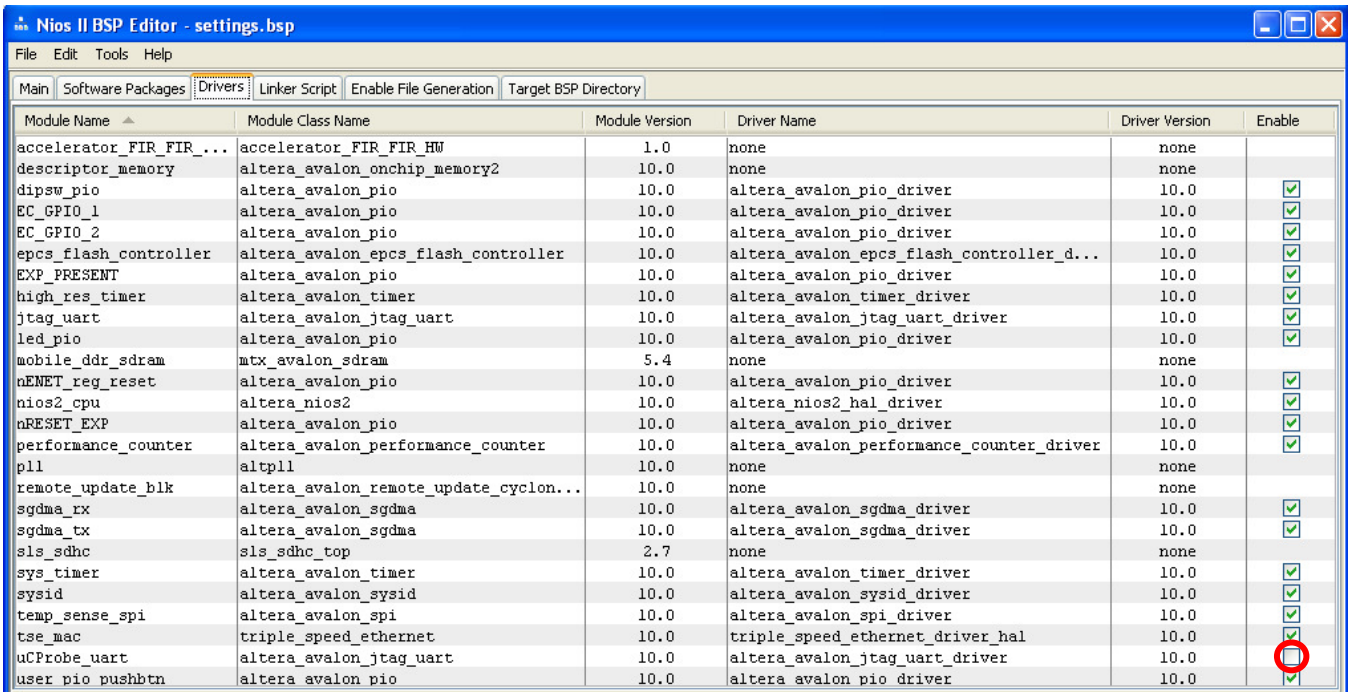
5. Select **mobile_ddr_sdram** as the linker target for **interrupt_stack_memory_region_name**.


Nios II BSP Editor - settings.bsp
File Edit Tools Help
Main Software Packages Drivers Linker Script Enable File Generation Target BSP Directory
SOPC Information file: ..\..\BeMicroSDK.sopcinfo
CPU name: nios2_cpu
Operating system: Altera HAL
BSP target directory: .\
Settings
Common
hal
enable_gprof
enable_reduced_device_drivers
enable_sim_optimize
enable_small_c_library
stderr
stdin
stdout
sys_clk_timer
timestamp_timer
linker
enable_exception_stack
enable_interrupt_stack
exception_stack_memory_region_na..
exception_stack_size
interrupt_stack_memory_region_na...
interrupt_stack_size
make
bsp_cflags_debug
bsp_cflags_optimization
Advanced
stderr:
jtag_uart
stdin:
jtag_uart
stdout:
jtag_uart
sys_clk_timer:
sys_timer
timestamp_timer:
high_res_timer
hal.linker
enable_exception_stack
enable_interrupt_stack
exception_stack_memory_region_name:
mobile_ddr_sdram
exception_stack_size:
1024
interrupt_stack_memory_region_name:
mobile_ddr_sdram
Information Problems Processing

Click on the **Drivers** tab

All drivers are selected by **default**. The application may not require that the **BSP** support all of these drivers.

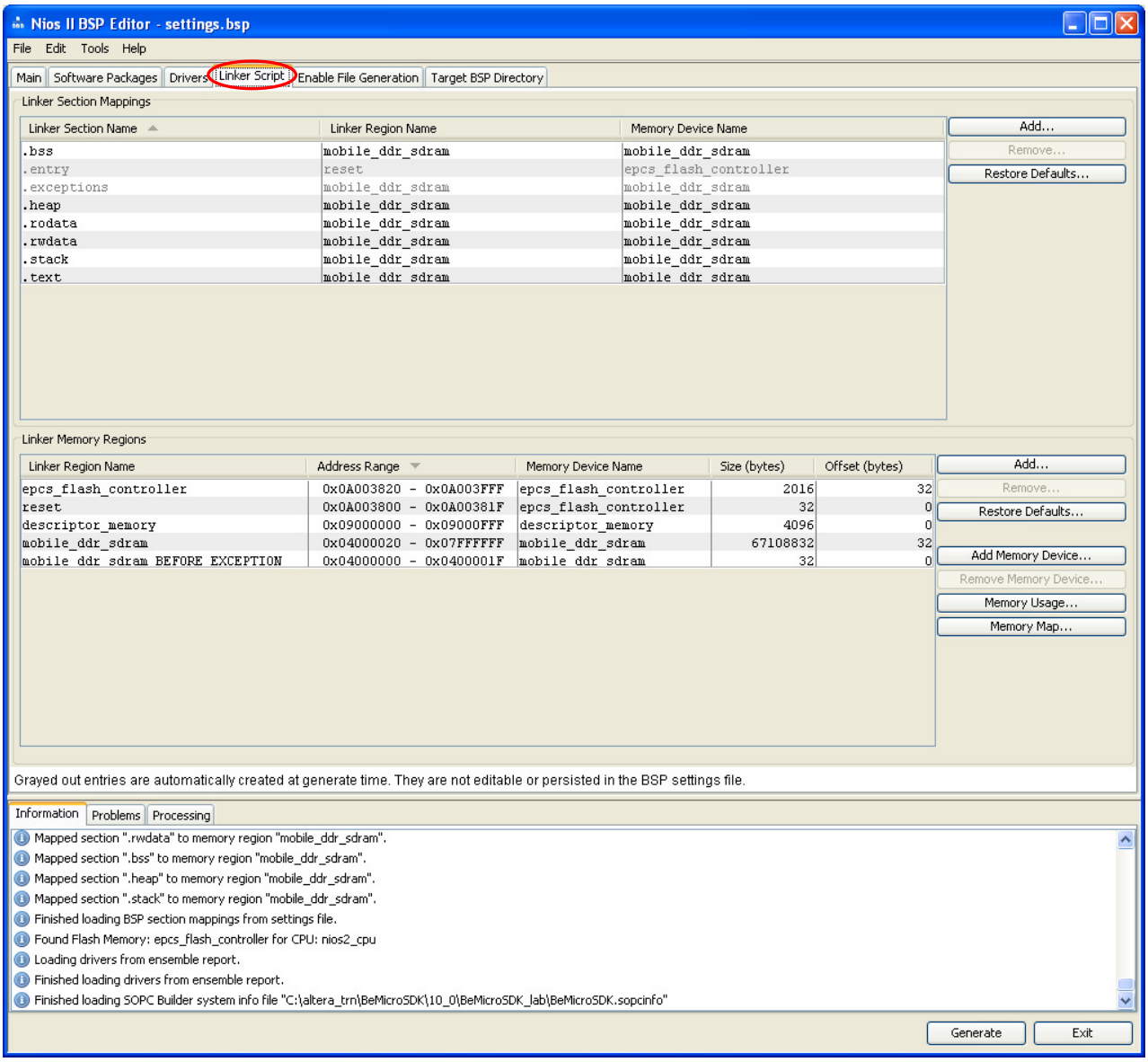
- Disable the uCProbe_uart driver by un-checking the enable check box.



Module Name	Module Class Name	Module Version	Driver Name	Driver Version	Enable
accelerator_FIR_FIR_...	accelerator_FIR_FIR_HW	1.0	none	none	
descriptor_memory	altera_avalon_onchip_memory2	10.0	none	none	
dipsw_pio	altera_avalon_pio	10.0	altera_avalon_pio_driver	10.0	☑
EC_GPIO_1	altera_avalon_pio	10.0	altera_avalon_pio_driver	10.0	☑
EC_GPIO_2	altera_avalon_pio	10.0	altera_avalon_pio_driver	10.0	☑
epcs_flash_controller	altera_avalon_epcs_flash_controller	10.0	altera_avalon_epcs_flash_controller_d...	10.0	☑
EXP_PRESENT	altera_avalon_pio	10.0	altera_avalon_pio_driver	10.0	☑
high_res_timer	altera_avalon_timer	10.0	altera_avalon_timer_driver	10.0	☑
jtag_uart	altera_avalon_jtag_uart	10.0	altera_avalon_jtag_uart_driver	10.0	☑
led_pio	altera_avalon_pio	10.0	altera_avalon_pio_driver	10.0	☑
mobile_ddr_sdram	mtx_avalon_sdram	5.4	none	none	
nENET_reg_reset	altera_avalon_pio	10.0	altera_avalon_pio_driver	10.0	☑
nios2_cpu	altera_nios2	10.0	altera_nios2_hal_driver	10.0	☑
nRESET_EXP	altera_avalon_pio	10.0	altera_avalon_pio_driver	10.0	☑
performance_counter	altera_avalon_performance_counter	10.0	altera_avalon_performance_counter_driver	10.0	☑
pll	altpll	10.0	none	none	
remote_update_blk	altera_avalon_remote_update_cyclon...	10.0	none	none	
sgdma_rx	altera_avalon_sgdma	10.0	altera_avalon_sgdma_driver	10.0	☑
sgdma_tx	altera_avalon_sgdma	10.0	altera_avalon_sgdma_driver	10.0	☑
sls_sdhc	sls_sdhc_top	2.7	none	none	
sys_timer	altera_avalon_timer	10.0	altera_avalon_timer_driver	10.0	☑
sysid	altera_avalon_sysid	10.0	altera_avalon_sysid_driver	10.0	☑
temp_sense_spi	altera_avalon_spi	10.0	altera_avalon_spi_driver	10.0	☑
tse_mac	triple_speed_ethernet	10.0	triple_speed_ethernet_driver_hal	10.0	☑
uCProbe_uart	altera_avalon_jtag_uart	10.0	altera_avalon_jtag_uart_driver	10.0	☐
user_pio_pushbtn	altera_avalon_pio	10.0	altera_avalon_pio_driver	10.0	☑

The BSP code size could be further reduced by disabling device drivers that will not be used.

Click on the **Linker Script** tab and change the following linker regions:



- Point the **.heap** linker section to the **mobile_ddr_sdram**.

- Point the **.stack** linker section to the **mobile_ddr_sdram**.

Ensure that the other sections are all pointing to **mobile_ddr_sdram**.

- Select **File** -> **Save** to save the board support package configuration to the **settings.bsp** file.

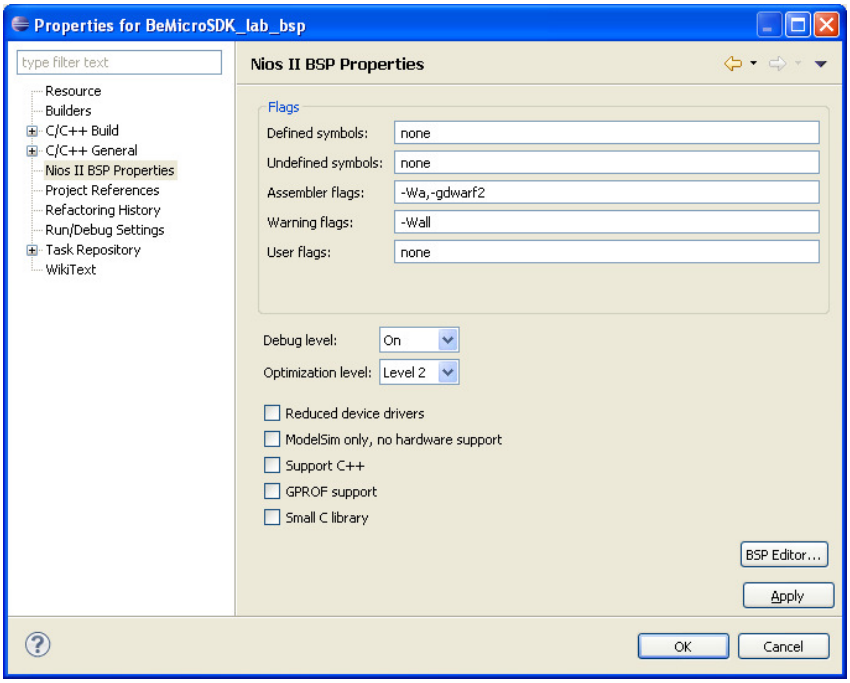
- Click the **Generate** button to update the BSP.

- When the generate has completed, select **File** -> **Exit** to close the BSP Editor.

3.4 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 **BeMicroSDK_lab_bsp** software project and select **Properties** from the right-click menu.
- On the left-hand menu, select **Nios II BSP Properties**.
- 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**.
- Since our software does not make use of C++, **uncheck** "**Support C++**".

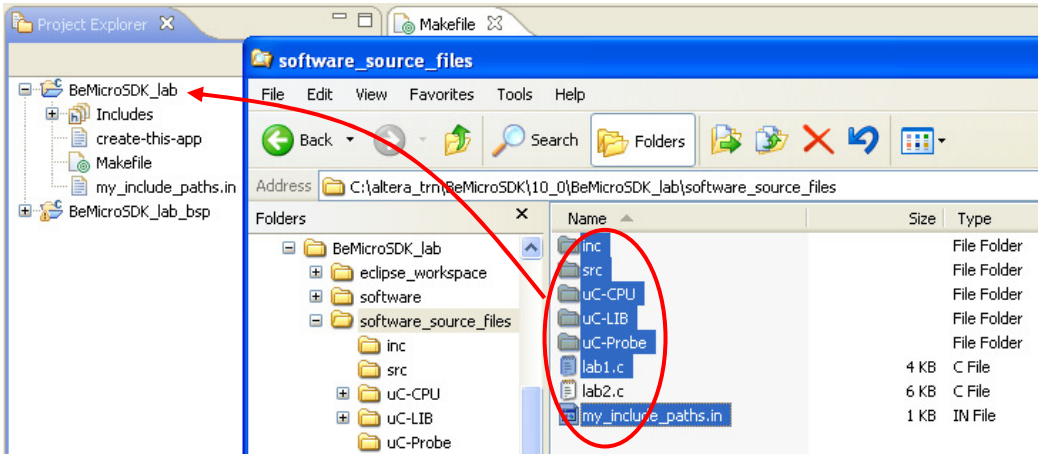


- Click **Apply**. Click **OK**.

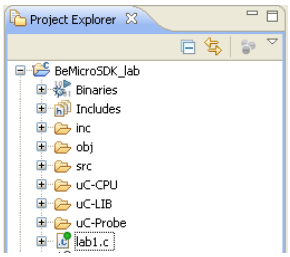
3.5 Add source code to the project

In Windows Explorer locate the project directory which contains a directory called "**software_source_files**" This directory contains an "**inc**" directory, "**src**" directory and "**lab1.c**" and "**lab2.c**" files. It also contains three other sub directories that are used to support the uCProbe tool on the embedded target. You will copy these files and directories from Windows Explorer and **drag and drop them** into the Eclipse software project directory, "BeMicroSDK_lab".

- **Select** all the files **EXCEPT lab2.c** and **drag** them over the "BeMicroSDK_lab" directory in the SBT window and **drop** the files onto the project folder.



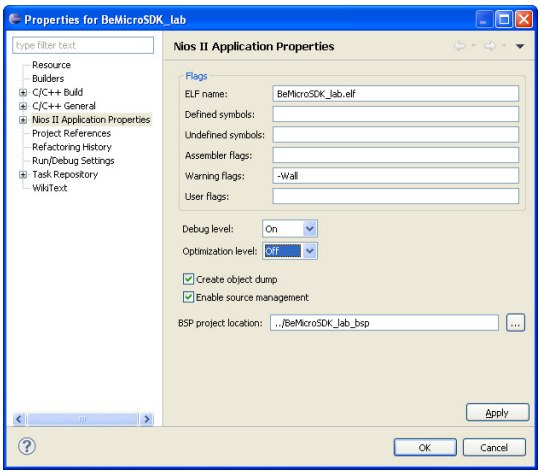
- This should cause the source files to be physically copied into the file system location of the software project directory and register these source files within the Eclipse workspace so that they appear in the Project Explorer file listing.



3.6 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 "BeMicroSDK_lab" as well.

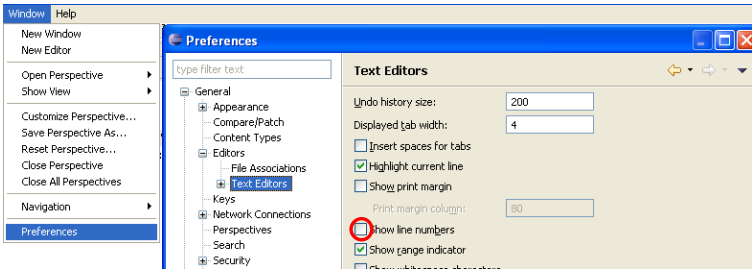
- **Right click** on the **BeMicroSDK_lab** software project and select **Properties** from the right-click menu.
- On the left-hand menu, select the **Nios II Application Properties** tab
- Change the **Optimization level** setting to **Off**.



3.7 Define Application Include Directories

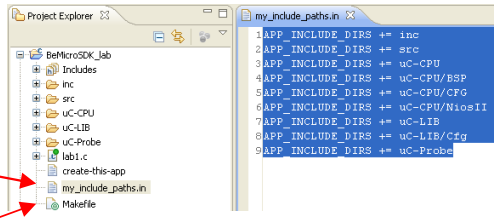
It is useful to be able to see code line numbers. Go ahead and enable line numbers in the source code

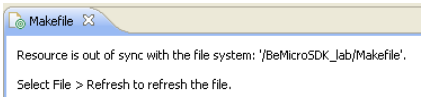
- Select **Window -> Preferences**
- In the **Preferences** window expand **General** then expand **Editors** and then select **Text Editors**
- Click on **Show Line Numbers**
- **Click OK**

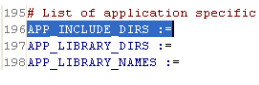


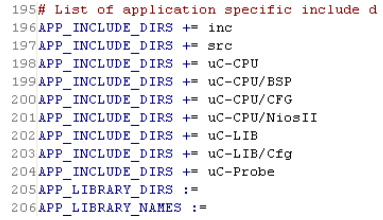
Application code can be conveniently organized in a directory structure. This section shows how to define these paths in the makefile.

- The paths have been provided for you. **Double click** on "**my_include_paths.in**" to open the file.
- Click the **Ctrl** and **A** keys to **select** all the text. Click the **Ctrl** and **C** keys to **copy** all the text.
- **Double click** on "**Makefile**" to open the file.
- If you see the message shown here about resources being out of sync, **right click** on the **Makefile** and select **Refresh**
- Select line **196 " APP_INCLUDE_DIRS :="**
- Click the **Ctrl** and V keys to **replace line 196** with the **include paths**.
- Click the **Ctrl** and **S** keys to **save** the **Makefile**.









CONGRATULATIONS!!

You now understand how to create a new BeMicroSDK Application and BSP project and define the properties of that project.

MODULE 4: Compile, Download and Run the Software project

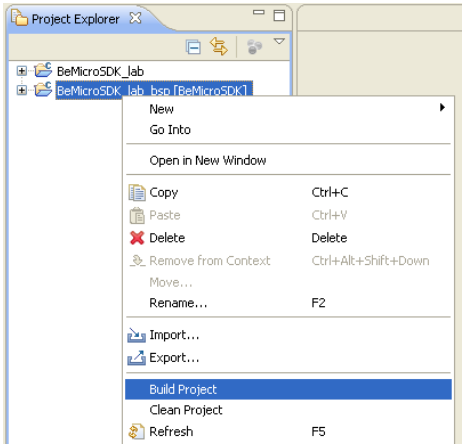
Module Objective

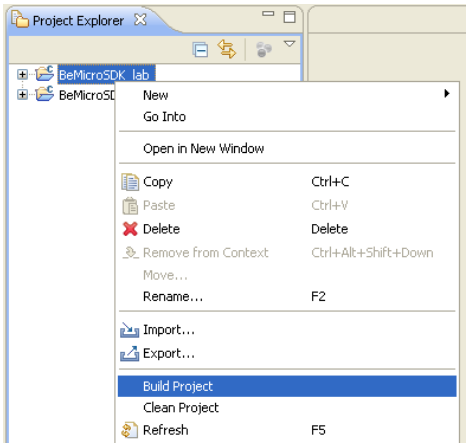
In this module you use the Nios II Software Build Tools (SBT) for Eclipse to build the Application and BSP projects and that will run on your system. You will verify that the System ID value of the .sopcinfo hardware description file matches the System ID peripheral in the BeMicroSDK FPGA hardware image. This is a critical step since a hardware engineer can make modifications to the hardware image that could result in memory map or other system modifications. The Application will be downloaded to the BeMicroSDK board memory from where it will be executed.

4.1 Build the Application and BSP Projects

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

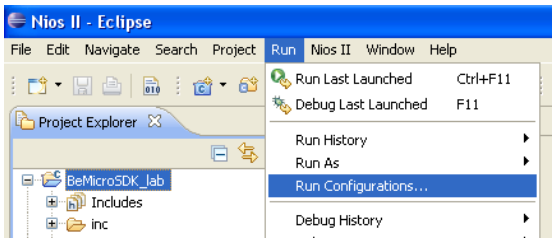
These 2 steps will compile and build the associated board support package, then the actual application software project itself. The result of the compilation process will be an Executable and Linked Format file for the application, the (*.elf) file.



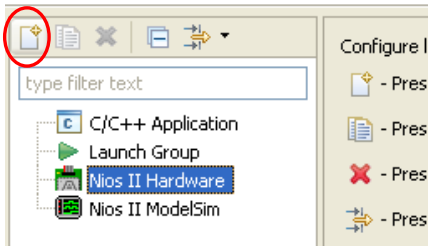


4.2 Verify the Board Connection

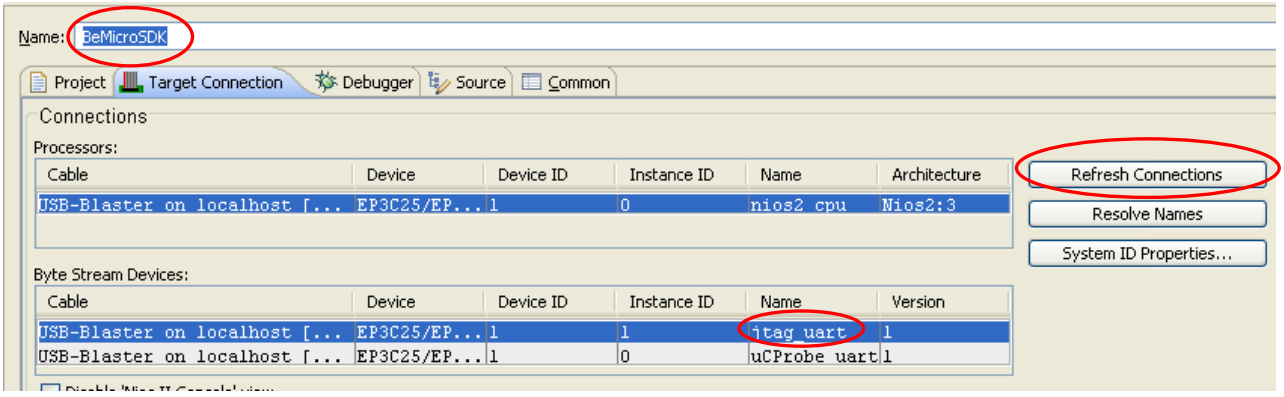
The BeMicroSDK hardware is designed with a System ID peripheral. This Peripheral is assigned a unique value based on when the hardware design was last modified in the SOPC Builder tool. SOPC Builder also places this information in the .sopcinfo hardware description file. The BSP is built based on the information in the .sopcinfo file. It is important to compare the contents of the System ID peripheral in the FPGA with that in the .sopcinfo file and verify that they match before running an application.



- Select the BeMicroSDK_lab **Application** project.
- Select **Run -> Run Configurations**…
- Select the **Nios II Hardware** hardware configuration type.
- Press the **New Button** to create a new **configuration**



- Change the **configuration** name to **BeMicroSDK** and click **Apply**.
- On the **Target Connection tab**, press the **Refresh Connections** button. You may need to expand the window or scroll to the right to see this button.
- Select the **jtag_uart** as the **Byte Stream Device** for stdio.



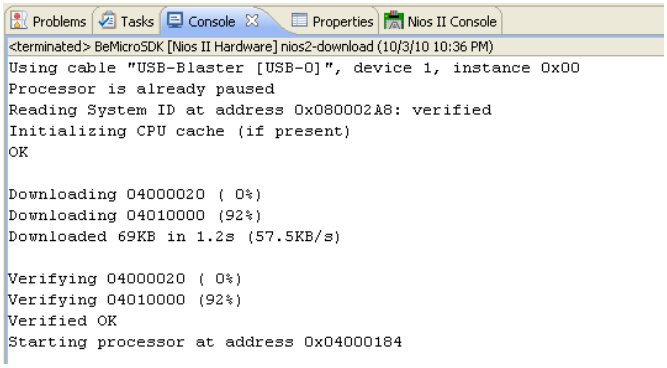
If the System ID mismatch error is reported, press the "Refresh Connections" button to update the System ID properties.

4.3 Run the software application on the target

To run the software project on the Nios II processor:

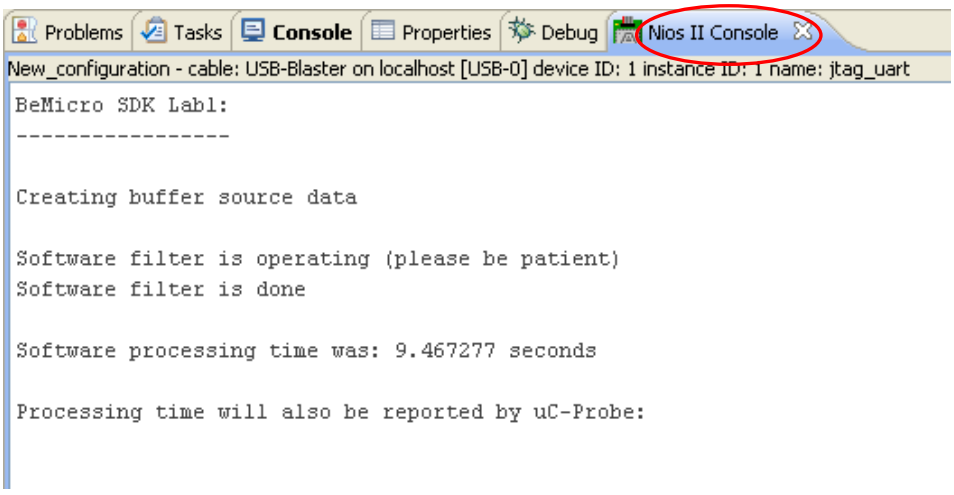
- Press the **Run** button in the Run Configurations window.

This will re-build the software project to create an up–to-date executable and then download the code into memory on our BeMicroSDK hardware. The debugger resets the Nios II processor, and it executes the downloaded code. Note that the code is verified in memory before it is executed.



4.4 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. **Results** will also be **reported** to the user via the **uC/Probe** tool in **Section 6**.



CONGRATULATIONS!! **You now understand how to build, download & run a new BeMicroSDK Application to the BeMicroSDK target board..**

MODULE 5: Examine the Nios II SBT Debug Perspective

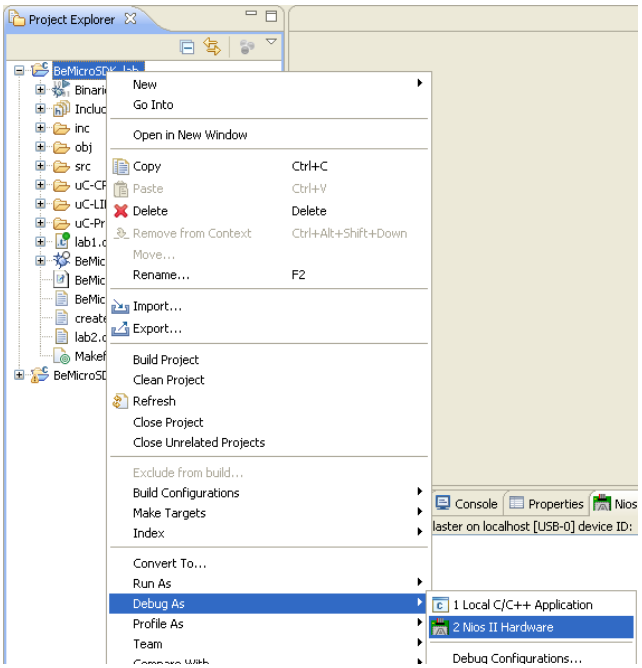
Module Objective

In this module you will examine the debug capabilities of the Nios II Software Build Tools (SBT) for Eclipse. You will learn how to switch to the debug perspective and set breakpoints in your code. You will also learn how to single step through code, step into functions and inspect registers, variables and memory locations.

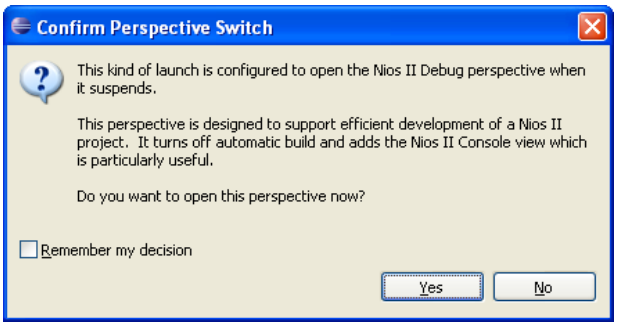
5.1 Start a Debug Session

The debug session launches a different perspective to the run session. This perspective or view allows the user to see a number of different debug windows that will aid in the navigation and understanding of the code being implemented.

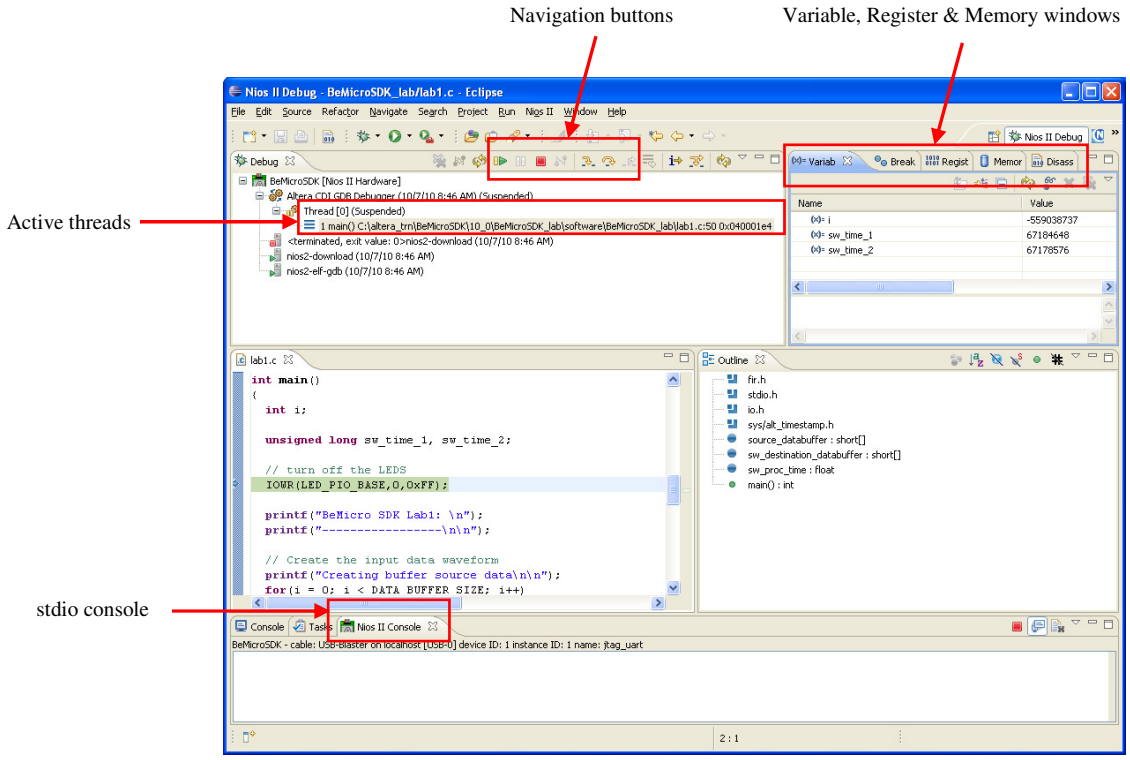
- **Right click** on the **BeMicroSDK_lab** application.
- Select **Debug As** and then select **Nios II Hardware**



- Confirm **Perspective Switch**. Select **Yes**



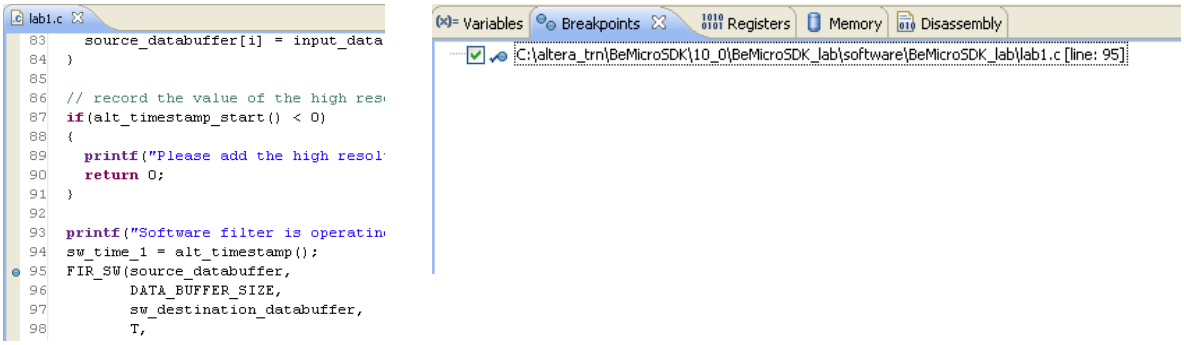
5.2 Examine the Nios II Debug Perspective



5.3 Setting Breakpoints

A breakpoint is traditionally defined as a place in your program where you want execution to stop so that you can examine program variables and data structures.

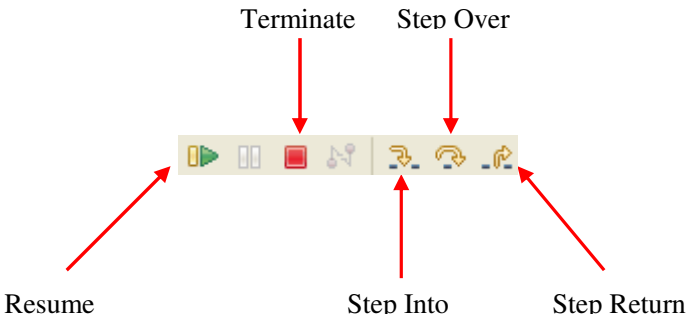
- **Double Click** in the margin to left of line **95** to insert a **breakpoint** at **FIR_SW**
- **Click** on the **Breakpoints Tab**. Notice that the **breakpoint** is now **listed**.



5.4 Run , Stop & Step

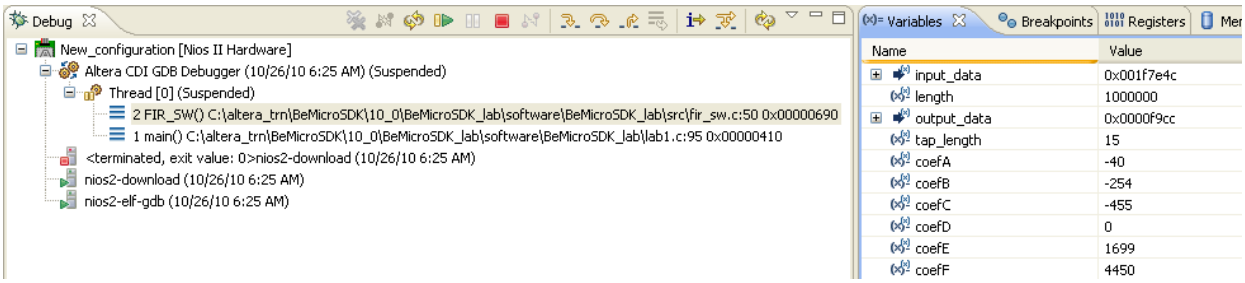
The **Nios II SBT** allow the following code **navigation options**. Code can be **run continuously or run till it hits a breakpoint**. Code can also be **stepped line** by **line**. When stepping two options are available. The first is to **step into** a **software** function. A **step return** can then be used to **go back** to the line after the function was called. The second option is to **step over** any software functions.

The **following controls** are available in the Debug Perspective



Run to the breakpoint then step into the FIR_SW function.

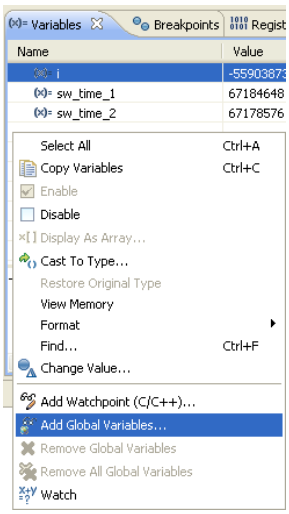
- **Click** on the **Resume** button**.** The code **halts** at the breakpoint on line **95** of **lab1.c**.
- **Click** on the **Step Into** button. The code **halts** at line **50** of **fir_sw.c**. This can be seen in the Debug Tab where are active threads are listed.
- Note that all **variables local** to a **thread** are shown by **default**. Click on the **Variables** Tab.

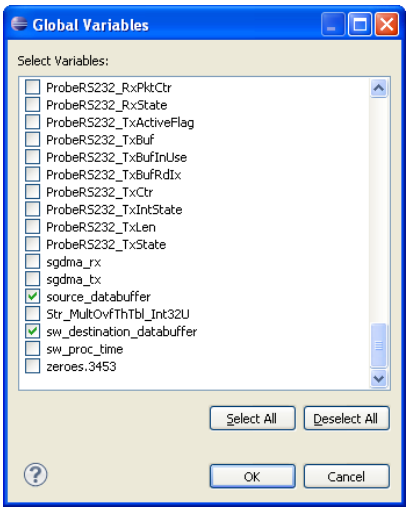


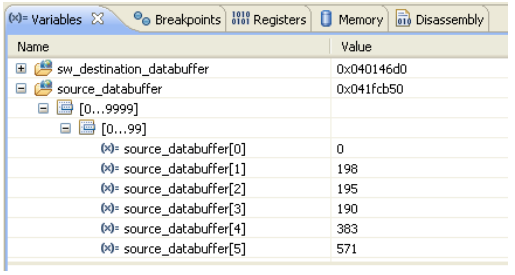
Global variables are not automatically visible in the variable Tab, but they can be manually added.

- Click on the **Variables** Tab and right-click in the background, then select **Add Global Variables.**
- Click on the **source_databuffer** and **sw_destination_databuffer** global variables.

- Expand the **source_databuffer** variable as shown below. It is possible to see the **sine wave values** in the **array**.

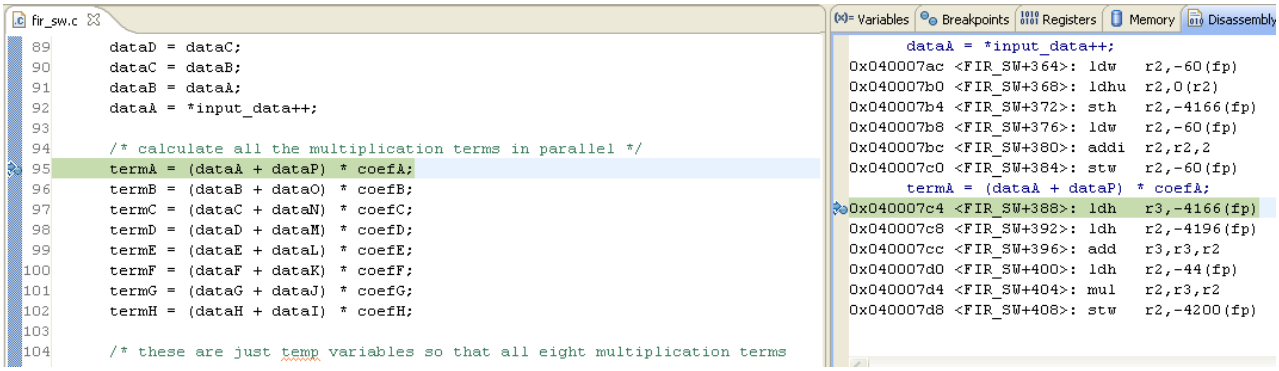






Step through the C source code and view the equivalent disassembled code.

- Click on the **Disassembly** Tab and press the Step Over button a few times.
- Notice that the **active line** of **code** in both the C and Disassembly windows are **highlighted** in **green**
- Also notice the close **correlation** between the **C** and **assembly** code. This is because we have the **code optimization** for the Application is currently **off**.



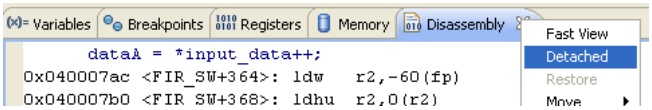
A fun exercise that you can do later would be to change the compiler optimization to Level 2 and then see the correlation between the C and assembly code .

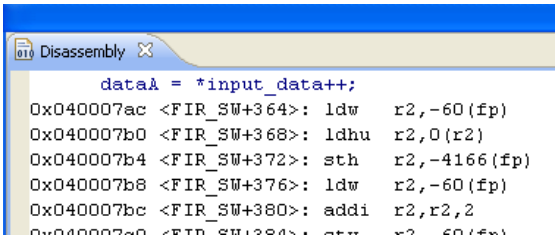
5.5 Memory & Register examination

The Nios II SBT tools provide access to the processor registers and any location within the systems memory map

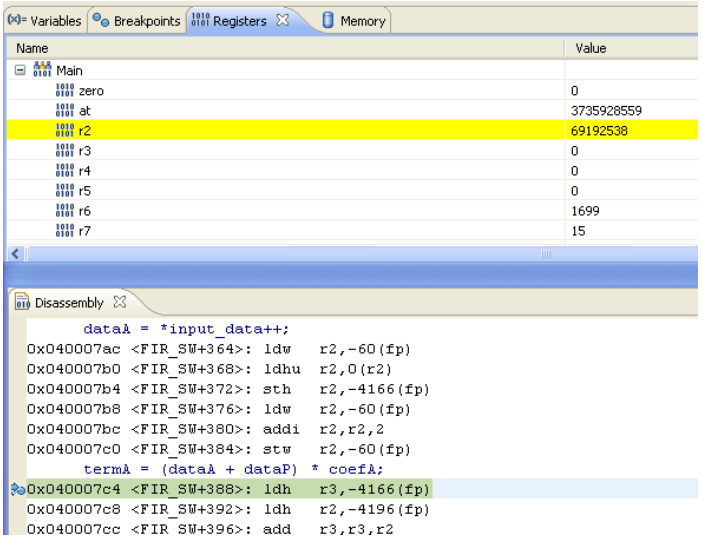
Register Examination. Detach the **Disassembly** window and move lower so both the Disassembly and **Register** windows are **visible**.

- Right click on the Disassembly Tab and select Detached. Drag the disassembly window lower



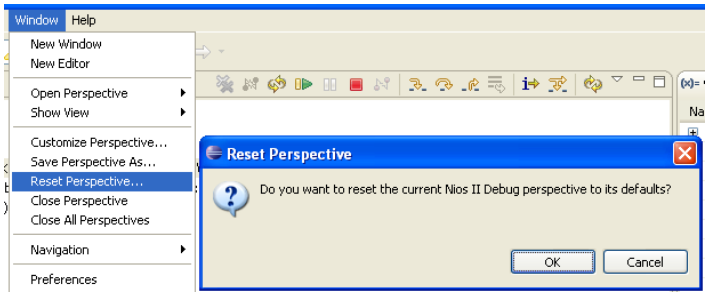


- **Click** on the **Registers** Tab. **Expand** Main.
- **Click** on **Step Over** a few times.
- **Note** the active **assembly** line code change
- **Note** that **register** changes are **highlighted** in **yellow**



Reset the Debug Perspective

- Click on the **Window** Tab and select **Reset Perspective**
- Select **OK** in the Reset Perspective **Dialog** box



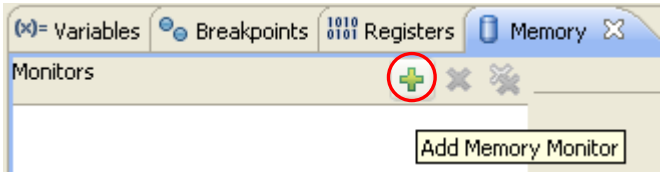
Return from the FIR_SW function to main.

- **Click** on the **Step Return** button

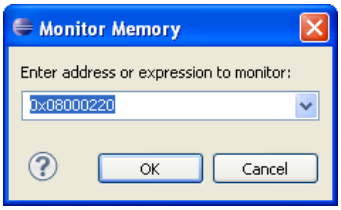
Memory Examination

The Nios II SBT tools allows the user to view or alter any location in the Embedded Systems memory map. This is done with memory monitor windows. Multiple memory monitors may be opened.

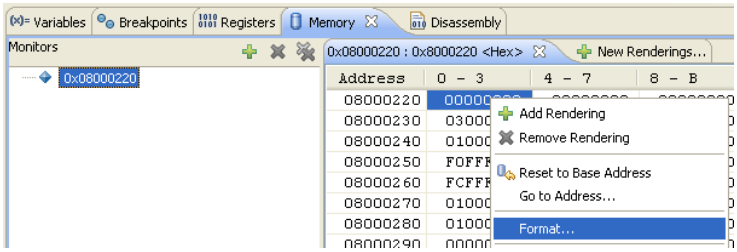
- **Click** on the **Memory Tab**
- **Click** on the **Add Memory Monitor** button

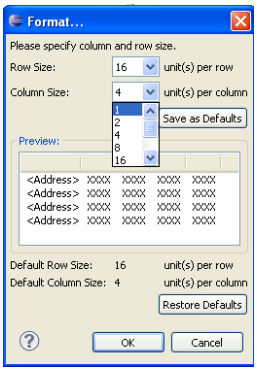


- Enter **0x08000220**. This is the **address** of the LED PIO port

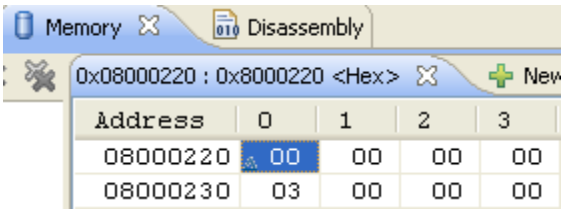


- **Right click** within the **address** map and Select **Format**.
- In the **Format** window Click on the **Column size pull-down** menu and select **1**.

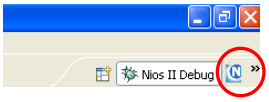




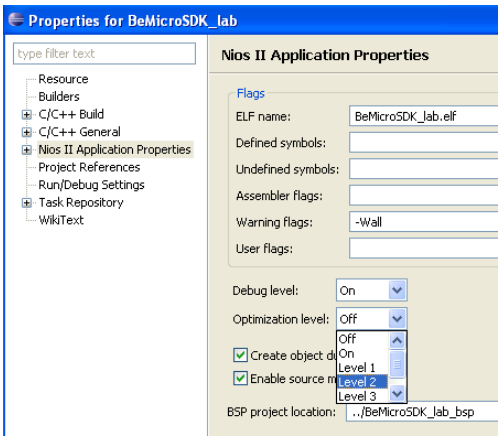
- **Observe** the **LEDs** on the **BeMicroSDK**.
- **Click** on address **0x08000220**.
- **Change** the **value** to **0x00**.
- **Observe** the **LEDs** again
- **Change** the **value** back to **0xff**



Revert to the Nios II Perspective

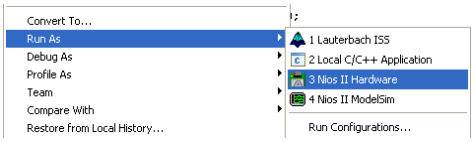


Set the Compiler Optimization for the BeMicroSDK_lab Application to Level 2



- **Right click** on the **BeMicroSDK_lab** Application 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 **OK**

Run the BeMicroSDK_lab Application



- **Right click** on the **BeMicroSDK_lab** Application project and select **Run As -> Nios II Hardware** from the right-click menu.

5.6 Profiling

The Nios II SBT tools provide a code profiler. The profiler provides an estimation, not an exact representation, of the processor time spent in different functions. To use the GNU profiler successfully with your custom hardware design, you must ensure that your design includes a system clock timer.

When the Application is run as hardware a profiling file gmon.out will be written into your software project directory. To save time this has been pre built for you and the results are displayed below. Please do **NOT** attempt this now. **See appendix B to implement this AFTER you have completed MODULE 7**.

lab1.c | gmon.out

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 86.19      2.31     2.31        1     2.31     2.31  FIR_SW
  6.34      2.48     0.17        1     0.17     2.48  main
  3.94      2.59     0.11        4     0.03     0.03  altera_avalon_jtag_uart_close
  2.24      2.65     0.06        8     0.01     0.01  temperature_alarm_callback
  1.12      2.68     0.03        1     0.03     0.03  _exit
  0.16      2.68     0.00        2     0.00     0.00  altera_avalon_jtag_uart_timeout
```

- Notice that 86.19 % of the total processing time (2.31s) was spent in the software FIR filter.
- 6.34% (0.17s) was spent in main initializing the arrays
- 3.94% was spent managing printf statements (altera_avalon_jtag_uart)
- 2.24% was spent servicing the temperature callback function

CONGRATULATIONS!!

You have successfully examined a number of debug features provided by the Nios II Software Build Tools.

MODULE 6: Instrument the Design using uC/Probe

Module Objective

A notable challenge in embedded systems development is to overcome the lack of feedback that such systems typically provide. Many developers resort to blinking LEDs or instrumenting their code with printf() in order to determine whether or not their systems are running as expected. Performance and memory footprint may be negatively impacted (Heisenberg effect). Micrium provides a unique tool named µC/Probe to assist these developers. With this tool, developers can effortlessly read and write the variables on a running embedded system.

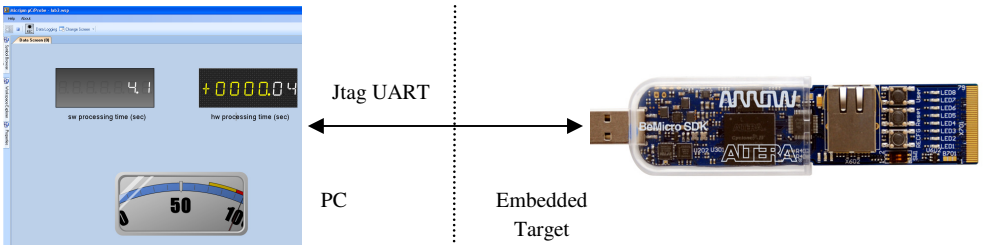
µC/Probe is a visualization tool, meaning that it allows developers to see inside embedded systems. In this module you will learn how to visually instrument the lab design with uC/Probe. You will also see how µC/Probe can gather data from running systems without negatively impacting application performance.

6.1 How uC/Probe functions

µC/Probe is made up of two components

- A Windows program that serves as a customizable interface to running embedded systems
- Embedded code that responds to requests from the Windows program

µC/Probe can be used to create a custom graphical interface for an embedded system. Developers simply drag and drop components onto data screens. µC/Probe accepts an executable file as input. Any compiler that produces executable files of the ELF format can be used with µC/Probe

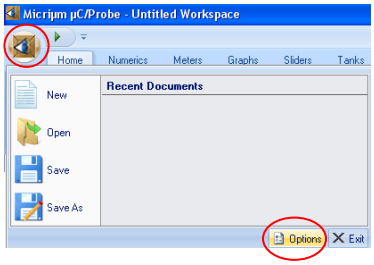


6.2 Setting up uC/Probe

Launch uC/Probe

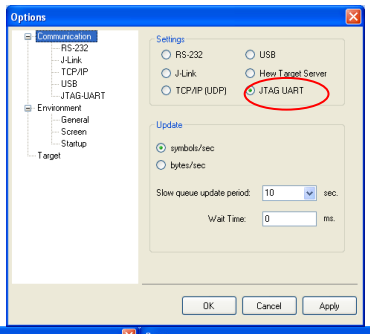
Launch uC/Probe from the **Start -> All Programs -> Micrium -> uC-Probe.** A small dialog box is launched. **Press** the **Evaluate** button.

Select uC/Probe **options.**



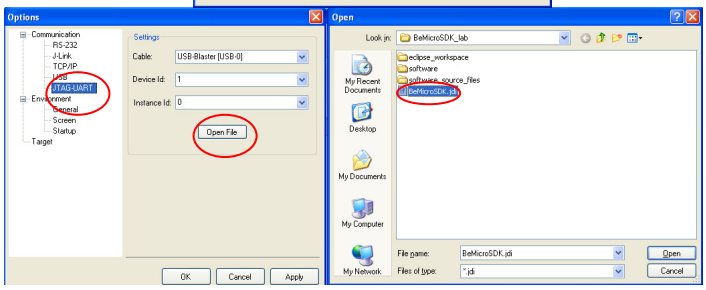
- **Click** on the **uC/Probe** icon on the **top left** portion of the screen**.**
- **Click** on the options button to open the dialog box

Set target board **communication** protocol as JTAG UART

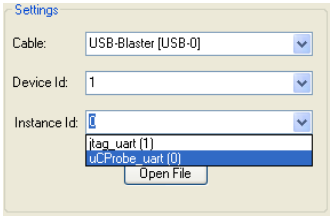


- **Click** on the **Communication Tab** icon on the **top left** portion of the dialog box
- **Select the JTAG UART** pilot button.

Setup JTAG UART **communication** settings



- Select the JTAG-UART option.
- Press the Open File button to select the JTAG Debug Information file (.jdi)
- Navigate to the <install directory>BeMicroSDK_lab directory folder and select the BeMicroSDK.jdi file. Press Open.
- Type the value **1** in the the **Device Id** window.



- Select uCProbe_uart(0) from the **Instance Id** pulldown menu. **Press Apply and OK** to exit the options menu. The embedded target has two UARTs. uCProbe will be communicating with the uCProbe_uart.

Populate the Symbol Browser

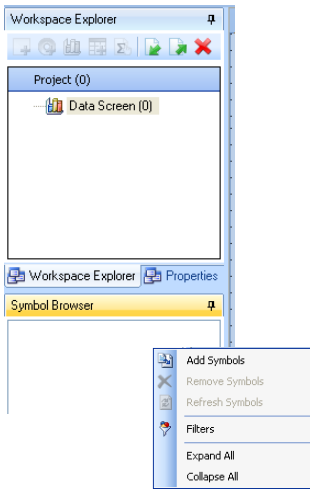
uC/Probe is able to extract variable information from an embedded target by using that targets ELF file. The ELF file provides symbol listings of all global variables and their location within the targets memory space. The ELF file is generated by the Nios II SBT tools during the Application build process.

```
04014594 g     O .bss    00000004 sw_proc_time
```

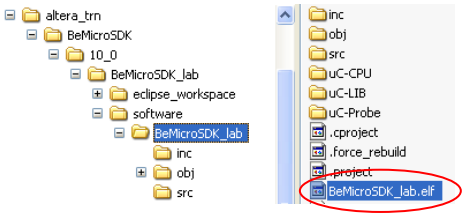
Add symbols.

- **Locate** the **Symbol Browser** within the Workspace Explorer

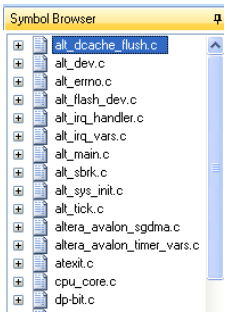
- **Right click** within the Symbol Browser. Select **Add Symbols** from the menu



- **Navigate** to the **BeMicroSDK_lab** software application folder and **select** the **BeMicroSDK_lab.elf**. Select **Open**.



- **Note** that the **Symbol** Browser has been **populated** with **all global variables** within the embedded **target** project.

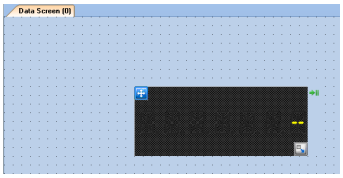


6.3 Create an instrument Data Screen in uC/Probe

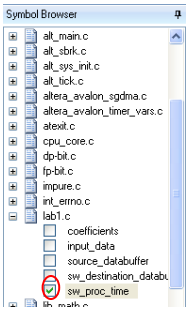
Add a Numeric Instrument Component **to the Data Screen**

- **Select the Numerics Tab**

- **Click** on the selected **numeric indicator** and **drag** it into the **Data Screen**



- **Expand** the **symbols** within **lab1.c** and **select the sw_proc_time check box**



6.4 Run & Display Results

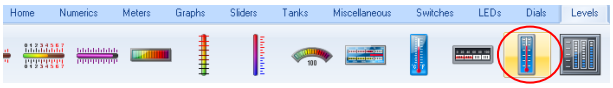
- **Press** the **play** button to **begin** the **capture** session. The result is displayed in **seconds.**

The software processing time is only calculated **once** when the target code is run. You will now **instrument** a **variable** that is **constantly updating.** If you inspect **lab1.c** you will see the following **lines** of **code.** This initializes the temperature sensor on the BeMicroSDK board by setting up an alarm_callback that is updated every 300mS.

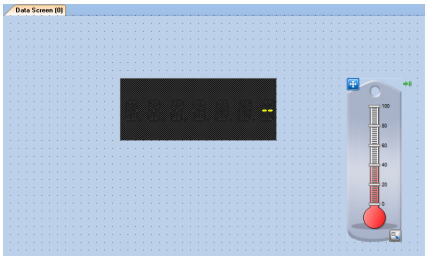
```
65  //enable the temperature sensor
66  temp_sensor_init();
```

- Press the **stop** button

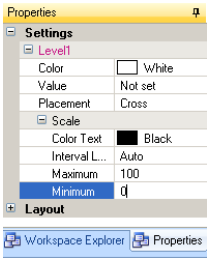
- **Select** the **Levels** Tab



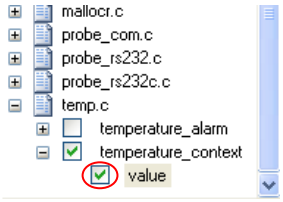
- **Click** on the selected **levels indicator** and **drag** it into the **Data Screen**



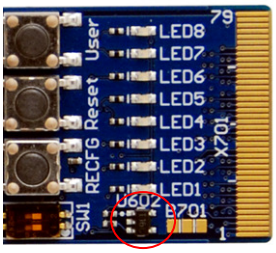
- **Click** on the properties tab set the following values**. Minimum = 65, Maximum = 90.** You can **alter** these values once you get a **baseline** of the **temperature** in the room.



- **Expand** the **symbols** within **temp.c** and then **expand the temperature_context symbol** and **select** the **value check box**

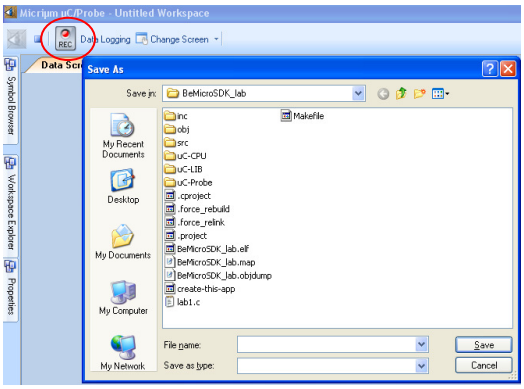


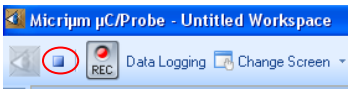
- **Press** the **play** button to **begin** the **capture** session.
- **Place your finger on the temperature sensor. Observe the temperature gauge on the Data screen**

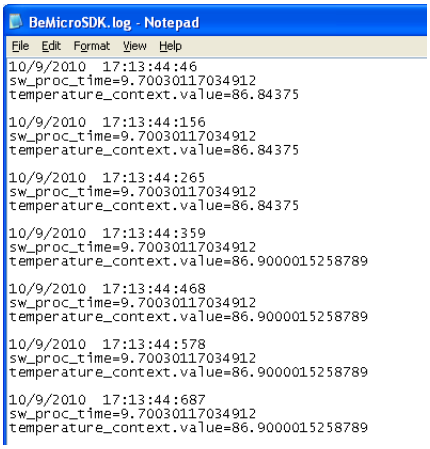


Log the Captured **Data**

- **Press** the **REC** button to start recording
- Enter **BeMicroSDK.log** for the filename
- Place your finger on the temperature sensor.
- Press the **stop** button to end the capture session
- Open the log file to observe the recorded data.







CONGRATULATIONS!!

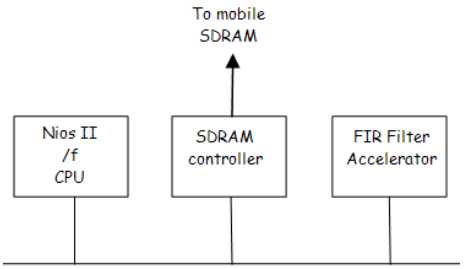
You have successfully used uC/Probe to instrument the embedded target design.

MODULE 7: Use Hardware Accelerators to Improve Performance

Module Objective

In this module you will see how **carefully selected** application **functions**, that consume large amounts of processor bandwidth, can be **replaced** with **hardware accelerators**. These accelerators are highly **optimized** for that specific **function**.

In this lab we have **added** a custom **FIR_HW accelerator** to the **BeMicroSDK** embedded design. It is wired via the System Interconnect Interface to the Nios II CPU.



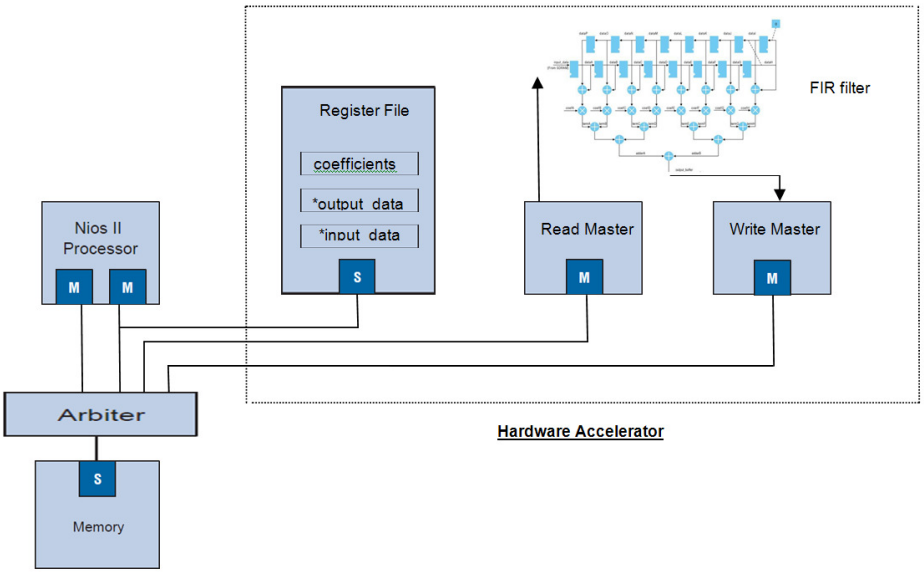
A quick **examination** of the **software** function **versus** its equivalent **hardware** accelerator is **illuminating**. You can **examine** these by opening the **files** in the **src** folder. The appropriate **information** is passed to the FIR_SW function. **Pointers** to the **input_data** and **output_data buffers** are provided as are all the filter **coefficient** values. The FIR_SW function **operates** in a **do while** loop until "**length**" number of **samples** have been **processed**. **Every** step is **performed** by the **Nios** II CPU.

```
void FIR_SW(short * input_data,  /* 15
            long length,
            short * output_data,
            unsigned char tap_length,
            short coefA,
            short coefB,
            short coefC,
            short coefD,
            short coefE,
            short coefF,
            short coefG,
            short coefH,
            unsigned char divide_order)
{

  long adderA, adderB;
  short dataA, dataB, dataC, dataD, dat
  short dataI, dataJ, dataK, dataL, dat
  long termA, termB, termC, termD, term
  long output_buffer[OUTPUT_BUFFER_LENG
  unsigned short indexA, indexB;
```

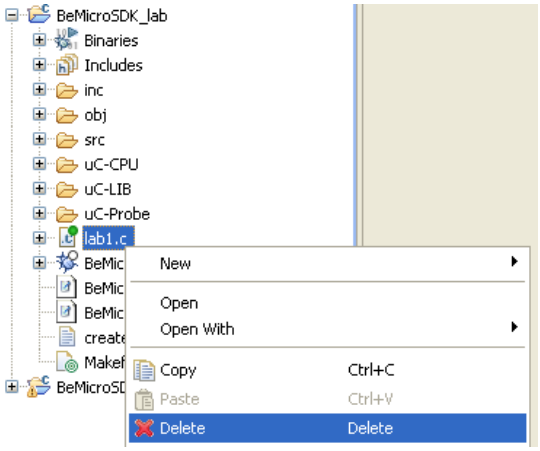
```
__inline__ void FIR_HW ( short *   input_data,
                long length,
                short *   output_data,
               unsigned char tap_length,
                short coefA,
                short coefB,
                short coefC,
                short coefD,
                short coefE,
                short coefF,
                short coefG,
                short coefH,
               unsigned char divide_order
             )
{
  IOWR_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE, (4), (int) (input_data));
  IOWR_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE, (8), (int) (length));
  IOWR_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE, (12), (int) (output_data));
  IOWR_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE, (16), (int) (tap_length));
  IOWR_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE, (20), (int) (coefA));
  IOWR_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE, (24), (int) (coefB));
  IOWR_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE, (28), (int) (coefC));
  IOWR_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE, (32), (int) (coefD));
  IOWR_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE, (36), (int) (coefE));
  IOWR_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE, (40), (int) (coefF));
  IOWR_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE, (44), (int) (coefG));
  IOWR_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE, (48), (int) (coefH));
  IOWR_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE, (52), (int) (divide_order));
  alt_dcache_flush_all();
  /* Write 1 to address 0 starts the accelerator */
  IOWR_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE,(0 * sizeof(int)),1);
  /* Poll. When read from address 0 returns 1, the accelerator is done */
  while(IORD_32DIRECT(ACCELERATOR_FIR_FIR_HW_MANAGED_INSTANCE_CPU_INTERFACE0_BASE,(0 * sizeof(int))) == 0)
  {}
```

In the **FIR_HW** function we see the **same parameter** information being **passed** from **main**. That is where the **similarities end**. The **accelerator** is comprised of a **large register file** which is **addressable** by the **Nios** II CPU and the function **parameters** are written (using the IOWR macro) to the register file. The accelerator is **started** by writing the value one to offset address 0. The **register** file **contains** the **input** and **output** data **pointers**. The read and write **masters independently** move **data** in and out of **external memory** while **funneling** it through the **hardware FIR** filter. It is an **independent co-processor**.

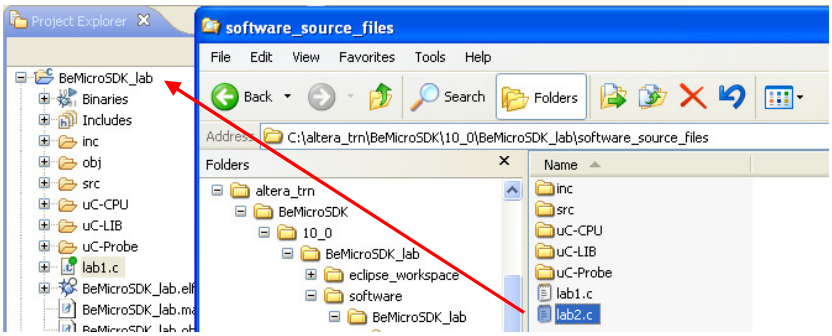


7.1 Build the Application and BSP Projects with Accelerator

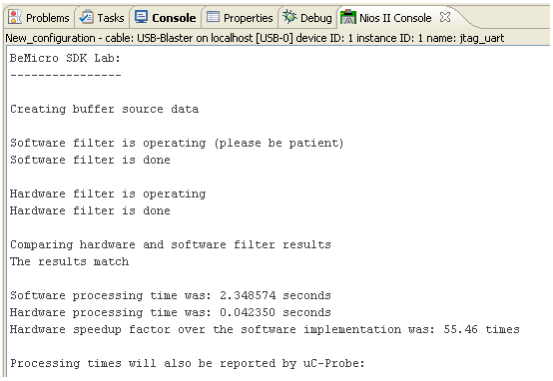
Build the **App** with the **accelerator**



- Switch back to the **Nios II SBT** tools
- **Right click** on **lab1.c**. Select **Delete**
- **Click** on **lab2.c**. **Drag** it into the BeMicroSDK_lab **App** folder



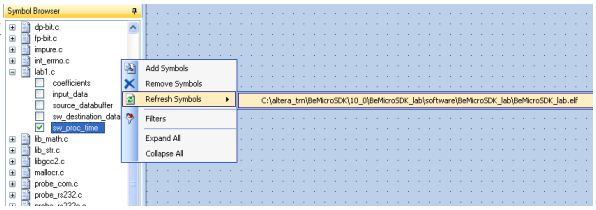
- **Right click** on BeMicroSDK_lab **App**. Select Build Project.
- **Right click** on BeMicroSDK_lab **App**. Select Run As -> Nios II Hardware.



7.2 Instrument uC/Probe

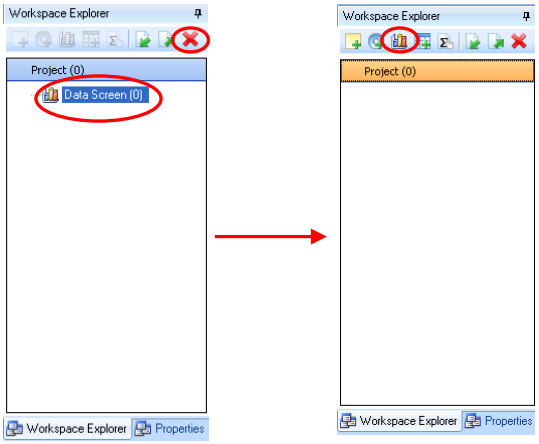
Update uC/Probe with the **new Symbol** Table

- Switch back to the **uC/Probe** tool
- **Right click** on the **Symbol Browser and** select **refresh** symbols. The **location** and availability of **symbols** will vary after a design is **recompiled**.



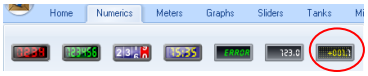
Delete the existing **Data Screen** and **create** a **new** one

- Click on the **Workspace Explorer Tab and** select Data Screen(0). **Press the delete button to remove it.**
- **Select** the **Add Data Screen** button to **add** a new **canvas**.

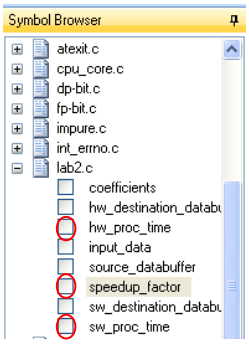


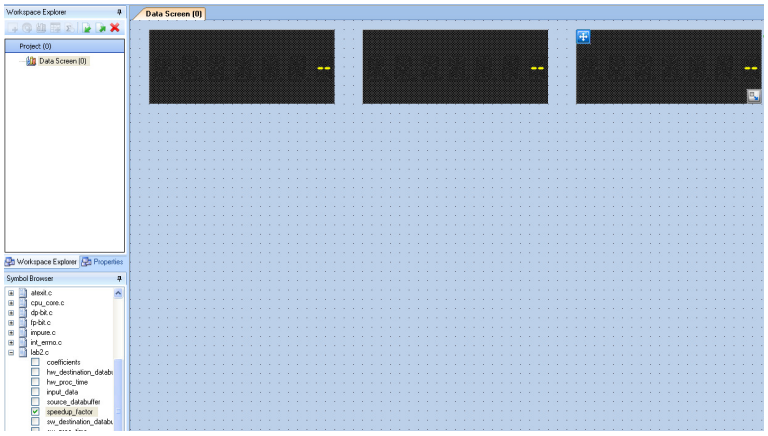
Instrument the **accelerated** design

- **Select** and drag the **numeric indicator** three times onto the **Data Screen canvas**.



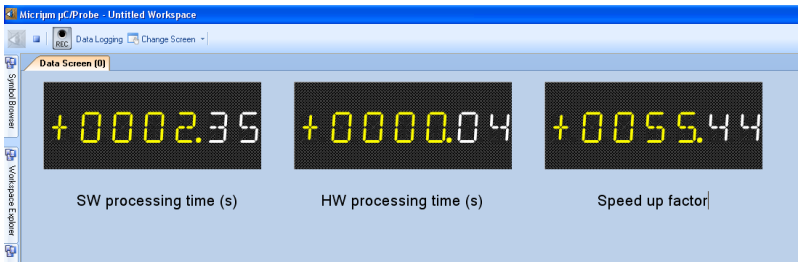
- **Expand** the **symbols** within **lab2.c.** Select the **first numeric indicator** and the **sw_proc_time** check box**.**
- Select the **second numeric indicator** and the **hw_proc_time** check box**.**
- Select the **third numeric indicator** and the **speedup_factor** check box**.**





7.3 Results of Acceleration

- **Press** the **play** button to **begin** the **capture** session. The results for the first two numeric indicators are displayed in **seconds.** The last is a **ratio** and shows the **speed up factor** of the **hardware** FIR function **over** the **software** FIR function



It is of interest to analyse the profile gmon.out file for lab2. The results below show a close resemblance to the uC/Probe findings

gmon.out

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 80.03      2.34     2.34        1     2.34     2.34  FIR_SW
 11.26      2.67     0.33        1     0.33     2.71  main
  5.52      2.84     0.16        4     0.04     0.04  altera_avalon_jtag_uart_close
  1.37      2.88     0.04        1     0.04     0.04  FIR_HW
  1.09      2.91     0.03        1     0.03     0.03  exit
```

Appropriately selected **Hardware accelerators** (or co processors) use **moderate** amounts of **resources** in the **FPGA** and yield **disproportionate increases** of computational **throughput**. In this **example** we would have to run the processor **55 times** faster to get the **same throughput** that the **hardware accelerator yields**. Hardware accelerators **allow** the **embedded design** to **achieve** the required **system performance** while **keeping clocking** rates and **power** utilization to a **minimum**.

CONGRATULATIONS!!

You have successfully used hardware accelerators to improve system performance..

MODULE 8: Taking the Next Step

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

Some of the resource available are:

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/n2sw_nii5v2.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/hb/qts/qts_qii5v4.pdf

Embedded Peripherals Handbook
http://www.altera.com/literature/hb/nios2/n2cpu_nii5v3.pdf

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

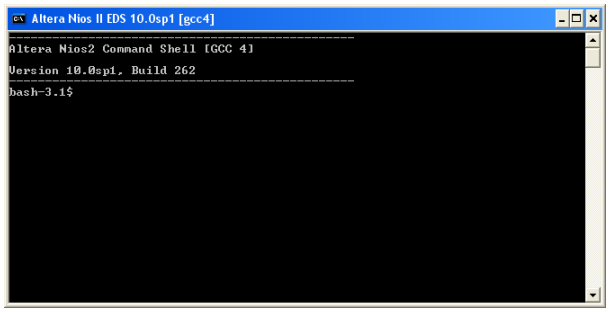
Appendix A: Restoring the Factory Hardware Image

The **BeMicroSDK** board has been **flashed** with an FPGA **hardware** image and a **Software image**.

During development **new images** can be **flashed** to the board. To **restore** the **factory image** follow these **steps**.

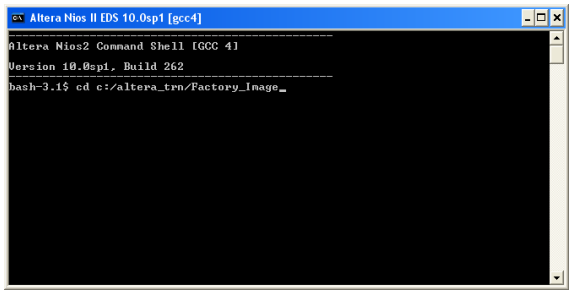
Open a Nios II Command Shell

- Start->Programs->Altera->Nios II EDS 10.0 -> Nios II 10.0 Command Shell



CD to the Factory_Image Directory

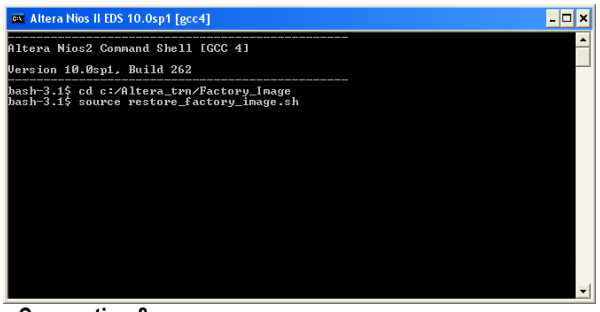
- **cd c:/altera_trn/Factory_Image**

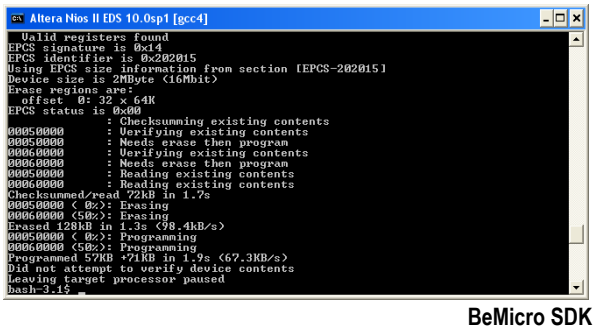


Connect the BeMicroSDK board to a USB port on the PC.

Run the Script and wait for it to complete.

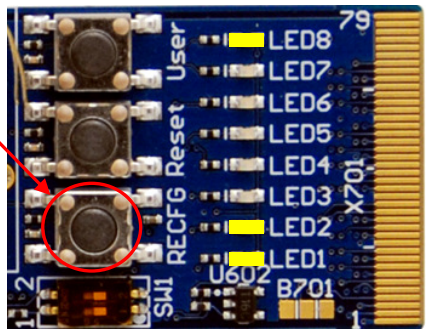
- At the **prompt** type "**source restore_factory_image.sh**"





Force the hardware to reconfigure with the new image.

- **Press the Reconfig Button.**



A temperature sensor software application has also been flashed to the board. It reads the temperature sensor and obtains the temperature in degrees F. These are displayed on the eight LEDs in a Binary Coded Decimal (BCD) format. For example the figure above is displaying a 83 deg. F

Appendix B: Using the Profiler

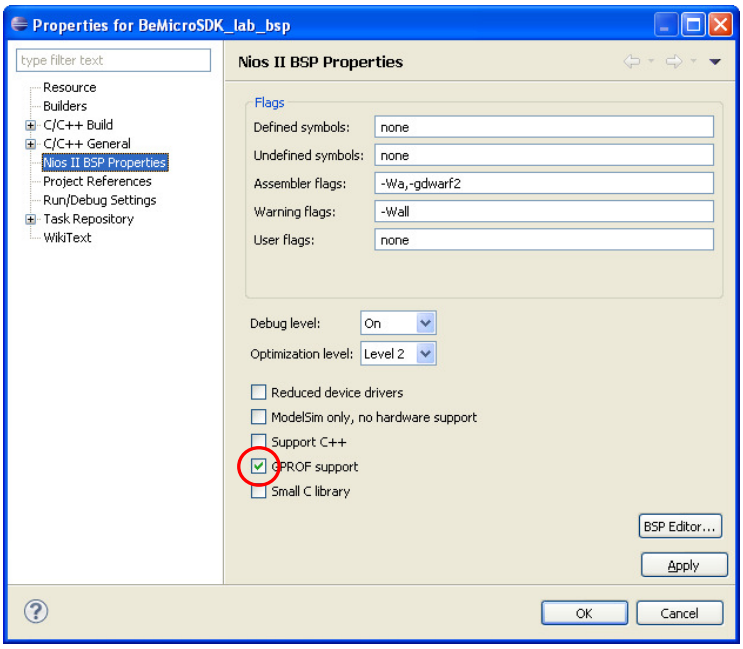
Profiling

The Nios II SBT tools provide a code profiler. The profiler determines the percentage of time spent in each function by interpolation, based on periodic samplings of the program counter. The periodic samples are tied to the system clock's timer tick. The profiler can only take samples when interrupts are enabled, and therefore cannot record the processor cycles spent in interrupt routines. The GNU profiler cannot profile individual functions. You can use the profiler to profile the entire system, or not at all.

The profiling data is a sampling of the program counter taken at the resolution of the system timer tick. Therefore, it provides an estimation, not an exact representation, of the processor time spent in different functions. To use the GNU profiler successfully with your custom hardware design, you must ensure that your design includes a system clock timer.

Enable Profiling in the BeMicroSDK_lab_bsp project

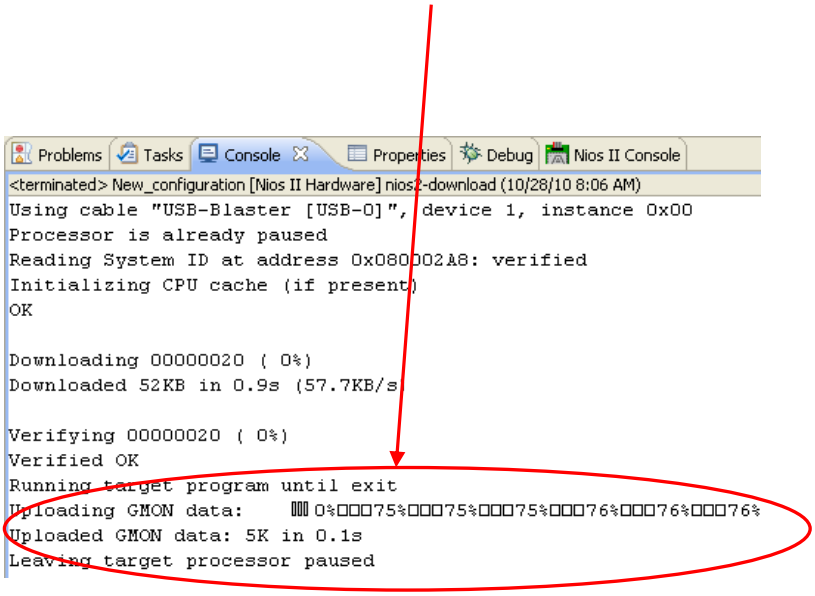
- **Right click** on the **BeMicroSDK_lab_bsp** software project and select **Properties** from the right-click menu.
- On the left-hand menu, select the **Nios II BSP Properties** tab
- **Click** on the **GPROF** support **check** box
- Select **OK**



Rebuild the BeMicroSDK_lab Application project and execute it.

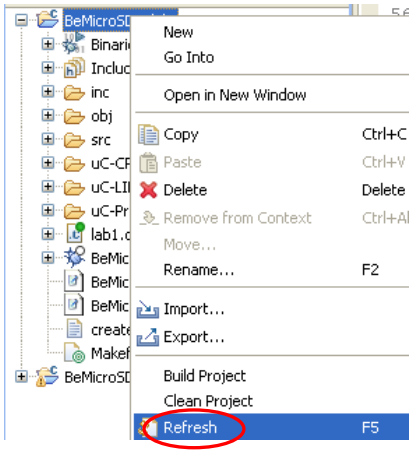
- **Right click** on BeMicroSDK_lab **App**. Select Run As -> Nios II Hardware.

Pay attention to the console output tab. Notice that after the application has completed and exited that the gmon data is uploaded from the target to the SBT tools.

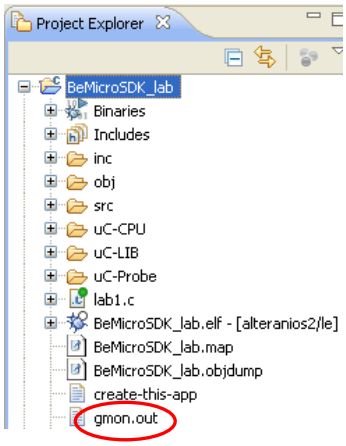


Refresh the BeMicroSDK_lab Application project and examine the profiler output.

- **Right click** on BeMicroSDK_lab **App** project. Select Refresh.



- Notice that a new file gmon.out has been added to the BeMicroSDK_lab **App** project .



- **Double click** on **gmon.out** to open it in a **profiling** window.

The results of acceleration become quite apparent when you study the profiler output. The SW_FIR function takes 2.32s while the HW_FIR function only takes 0.04s.

Please refer to Altera Application Note 391 : Profiling Nios II Systems for more detailed information on how to use the profiler.

gmon.out

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 84.76      2.32     2.32        1     2.32     2.32  FIR_SW
 12.04      2.65     0.33        1     0.33     2.69  main
  1.46      2.69     0.04        1     0.04     0.04  FIR_HW
  1.16      2.72     0.03        1     0.03     0.03  _exit
  0.64      2.74     0.02        9     0.00     0.00  temperature_alarm_callback
  0.00      2.74     0.00      284     0.00     0.00  alt_irq_handler
  0.00      2.74     0.00      275     0.00     0.00  alt_avalon_timer_sc_irq
  0.00      2.74     0.00      275     0.00     0.00  alt_tick
```