CAPSTONE PROJECT GUIDE

MODULE IV

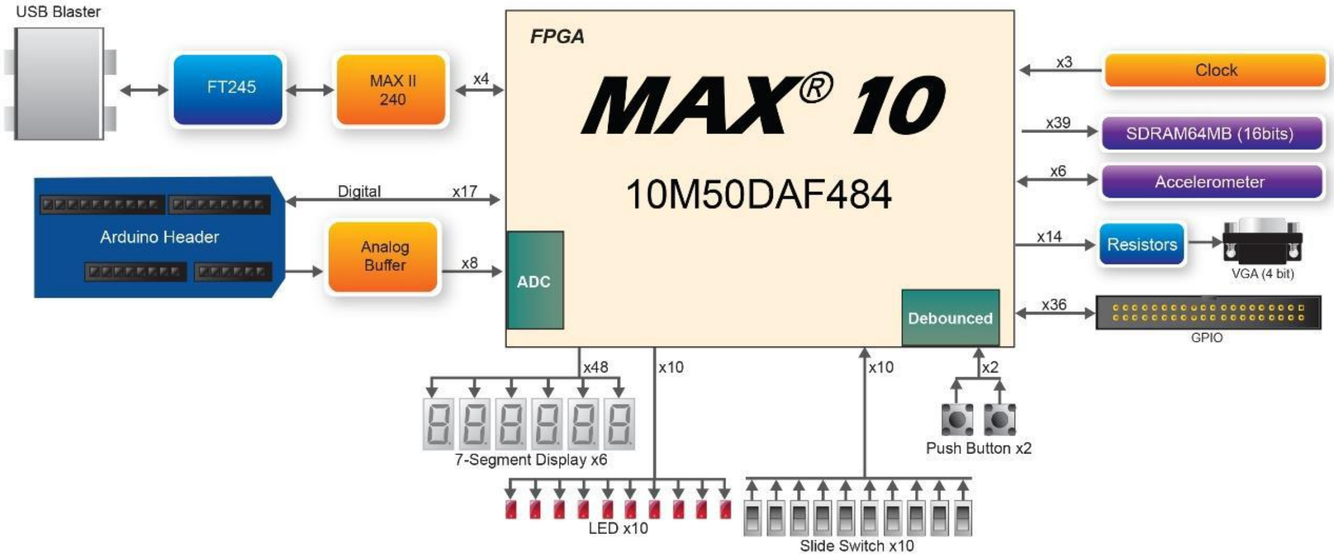
FPGA CAPSTONE PROJECT MODULE IV: SOFTWARE FOR A SYSTEM ON A CHIP

TABLE OF CONTENTS

INTRODUCTION

The DE10-Lite is a FPGA evaluation kit that is designed to get you started with using an FPGA. The DE10-Lite adopts Altera's non-volatile MAX® 10 FPGA built on 55-nm flash process. MAX 10 FPGAs enhance non-volatile integration by delivering advanced processing capabilities in a low-cost, instanton, small form factor programmable logic device. The devices also include full-featured FPGA capabilities such as digital signal processing, analog functionality, Nios II embedded processor support, and memory controllers.

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



PROJECT LEARNING OBJECTIVES

For this project, the objective is for students to:

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

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

The modules will get progressively more difficult and take more time. This is Module 4.

MODULE GOAL

The goal of this module is to develop the software for a System on a Chip (SoC). You will build software for a NIOS II soft processor you built in Module 3, using several interfaces to devices on the DE10-Lite development kit as well. In this module you will

- Enhance and test a working design, using most aspects of the Quartus Prime Design Flow and the NIOS II Software Build Tools (SBT) for Eclipse.
- Create software for the NIOS II soft processor, including many interfaces, using Qsys (Platform Builder) and the SBT.
- Compile your completed software using the SBT.
- Use Quartus Prime to program both the FPGA hardware configuration and software code in you DE10-Lite development kit. Test your new embedded system.
- Record all your observations in a lab notebook pdf
- Submit your project files and lab notebook for grading

Completing this module will finish your work for this course.

I. GENERAL PROCEDURE

Caution:

Do not continue until you have read the following:

The names that this document directs you to choose for files, components, and other objects in this exercise **must be spelled *exactly* as directed.**

CREATING SOFTWARE FOR THE NIOS II SOFT PROCESSOR

1. If you have not already done this, download and install Quartus Prime FPGA development software from Intel Altera. Follow the directions in the installation video in Course 1 of the specialization if you need help. Install version 16.1.
2. Follow the instructions in the detailed design section that follows for the NIOS II embedded system. Be sure to record your observations as you go.
   a. If your results look different than the project guide, do not be alarmed, as there may be some differences between the tools used in the guide and your particular installation of the tools.
   b. Do not be surprised when the initial and even subsequent compiles of the FPGA have errors. This will point you to the additional work you need to do.
3. This section is based on completing the output portion of the design of a simple voltmeter.
   a. In Section 1 you will prepare for the project by acquiring files and other resources.
   b. In Section 2 you will initialize the Eclipse software development environment.
   c. In Section 3 you will create a software project by adding source code.
   d. In Section 4 you will configure the board support package (BSP).
   e. In Section 5 you will build the software project.
   f. In Section 6 you will create a programming file that could be used to configure the FPGA fabric and also load the software application, which will then run on the target board.
   g. In Section 7 you will interact with the software application, edit it and make changes and observe the results in the target.
4. Take your DE10-lite evaluation board and plug the USB cable into a computer. Program the FPGA with your NIOS II based embedded system design including software. Record your observations in your lab notebook - Describe how the device behaves.
5. Record the Fmax for this lab in your lab notebook. If an Fmax is not listed, use the inverted highest clock frequency for this number.
6. Estimate the % utilization of the FPGA logic for this lab at completion.
7. Submit a zipped up project file and lab notebook for grading.

II. DETAILED DESIGN

CREATING SOFTWARE FOR THE NIOS II SOFT PROCESSOR

1. GETTING STARTED

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

List of Required Items:

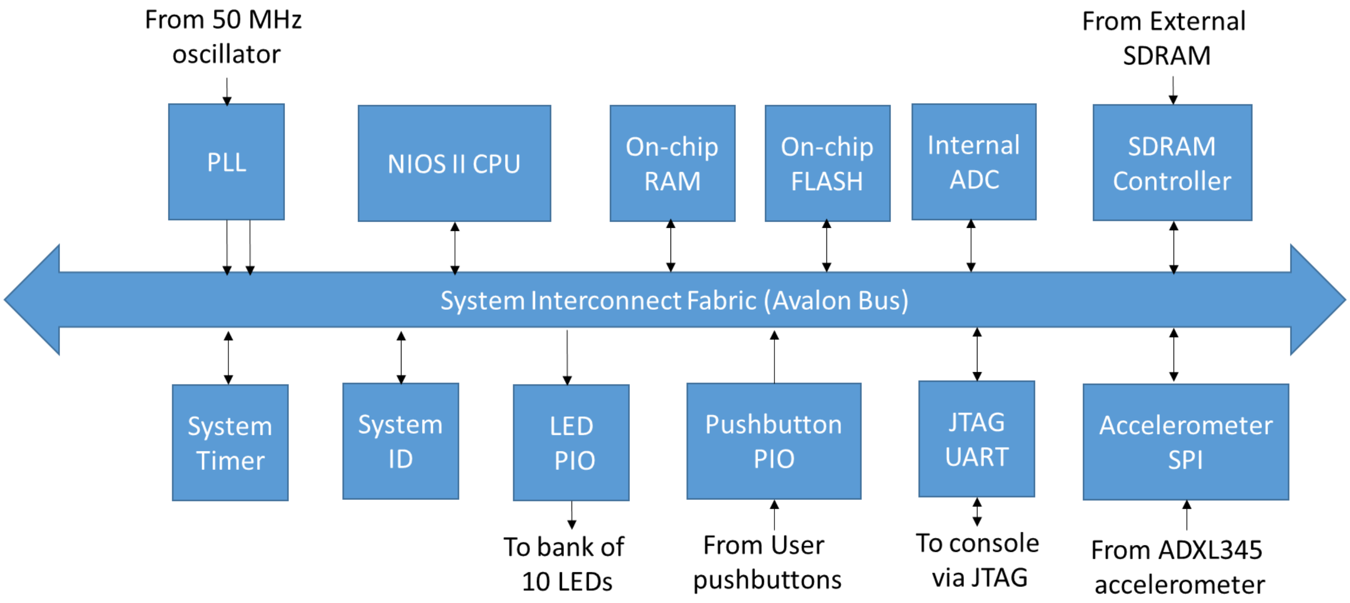
- Altera Quartus Prime Version 16.1 FPGA Development Software Tool
- Altera NIOS II SBT for Eclipse
- **Module Design Files: EmbedSoft.zip**
- Terasic DE10-Lite Development Kit

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

EXTRACT THE LAB FILES.

- Open the folder **C:\AlteraPrj\DE10liteEmbed** on your PC. This should contain the hardware design project that you completed in Module 3.
- Copy EmbedSoft.zip to this directory
- Extract here to the above directory

You will be building software for the NIOS II embedded system you constructed in Module 3. As a reminder, here is the block diagram for that system:



Good Work!!

You have just completed all the setup and installation requirements and are now ready to initialize the Eclipse workspace.

2. INITIALIZE THE ECLIPSE WORKSPACE

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.

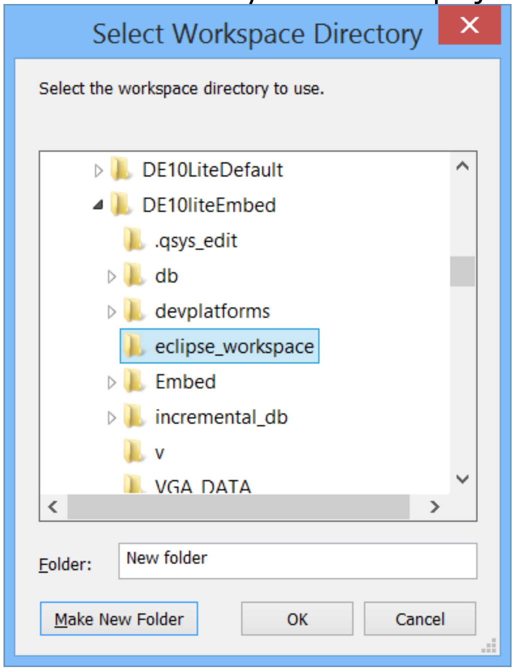
Before you begin with the tools, unzip the EmbedSoft.zip file into the project directory. This should create a directory called Source.

LAUNCH THE NIOS II SOFTWARE BUILD TOOLS FOR ECLIPSE

Launch the Nios II SBT from the **Start -> All Programs -> Intel FPGA 16.1.0.196 Lite Edition -> (64-bit) -> Nios II 16.1 Software Build Tools for Eclipse** or alternatively it can be launched from the **Tools -> Nios II Software Build Tools for Eclipse** menu.

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**".

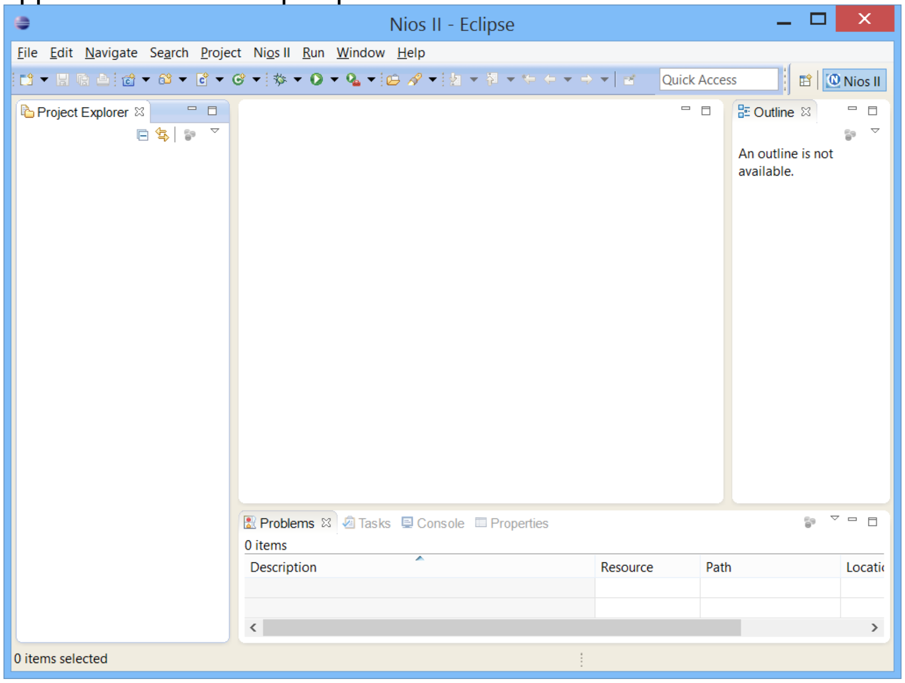


Select Workspace Directory

Select the workspace directory to use.

Folder: New folder

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



Nios II - Eclipse

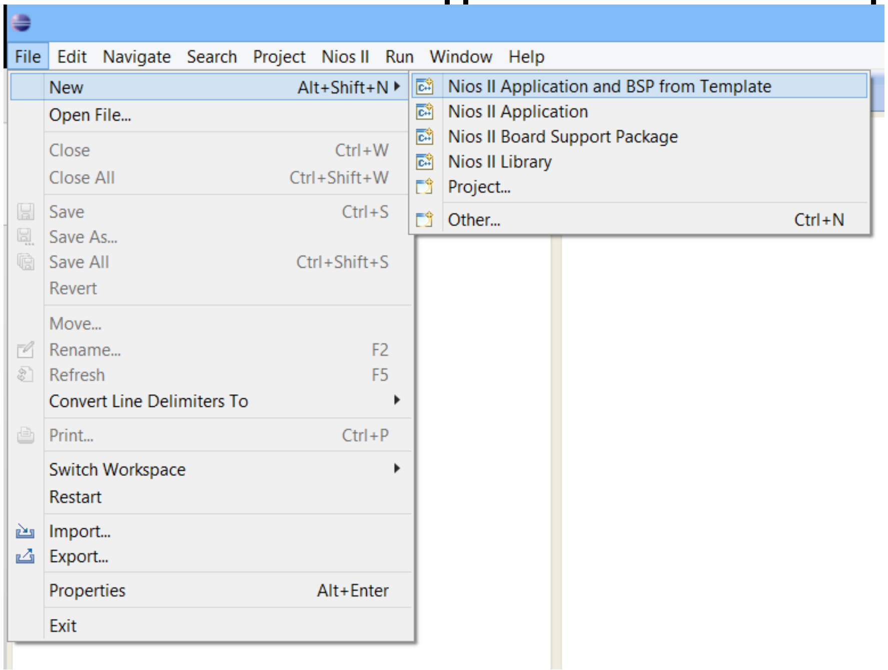
An outline is not available.

0 items

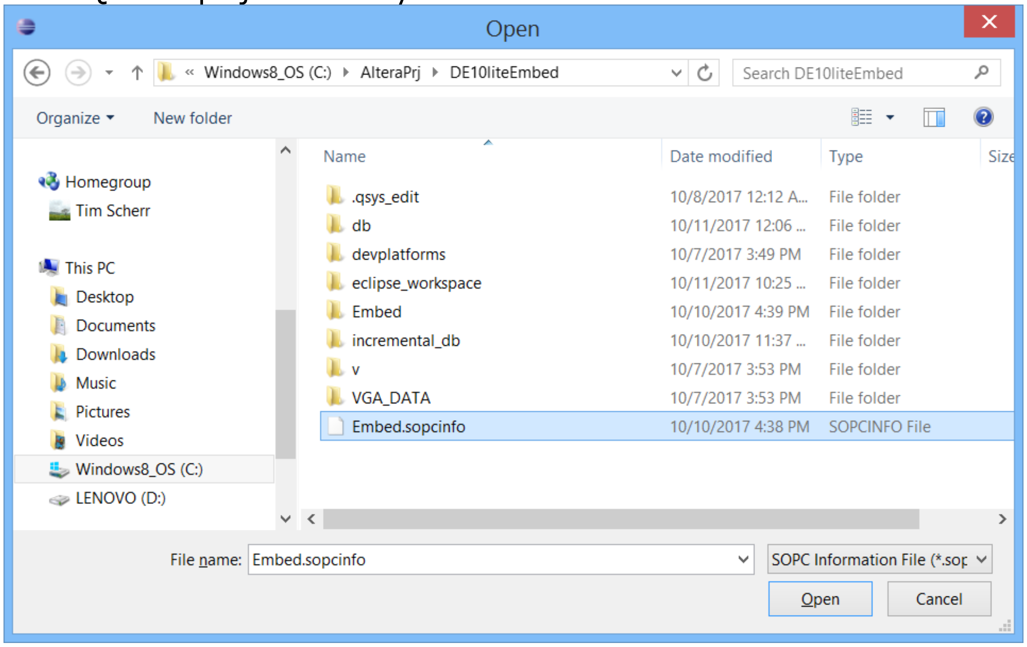
0 items selected

3. 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 **Embed.sopcinfo** file located in the Quartus project directory.

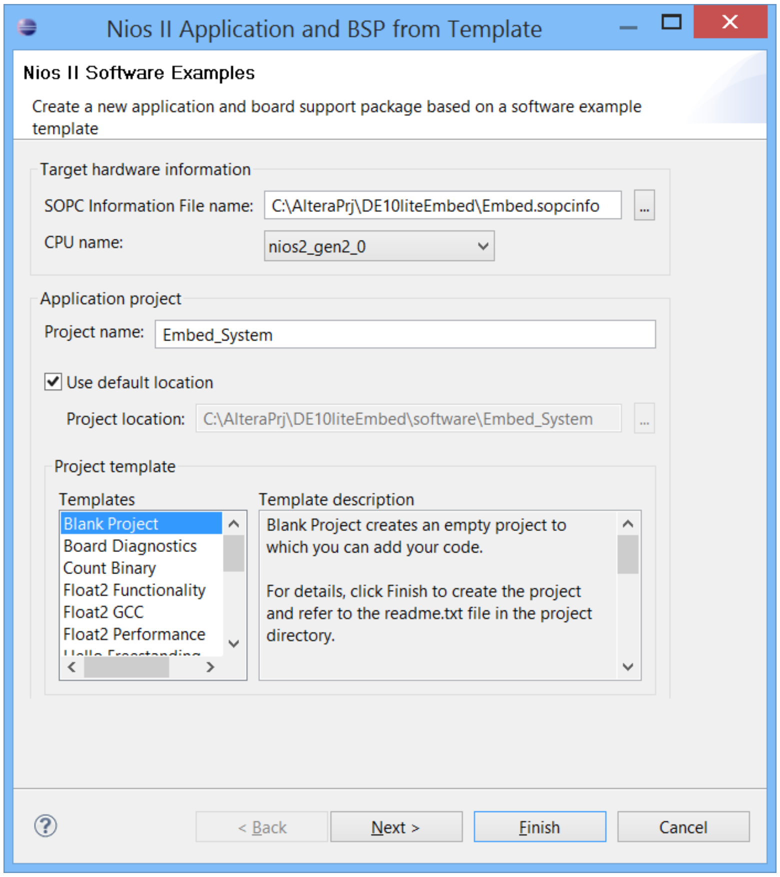


Name	Date modified	Type	Size
.qsys_edit	10/8/2017 12:12 A...	File folder	
db	10/11/2017 12:06 ...	File folder	
devplatforms	10/7/2017 3:49 PM	File folder	
eclipse_workspace	10/11/2017 10:25 ...	File folder	
Embed	10/10/2017 4:39 PM	File folder	
incremental_db	10/10/2017 11:37 ...	File folder	
v	10/7/2017 3:53 PM	File folder	
VGA_DATA	10/7/2017 3:53 PM	File folder	
Embed.sopcinfo	10/10/2017 4:38 PM	SOPCINFO File	

File name: Embed.sopcinfo

SOPC Information File (*.sop

Set the name of the Application project to "**Embed_System**".
Select the **Blank Project** template under **Project Template**.



Nios II Application and BSP from Template

Nios II Software Examples

Create a new application and board support package based on a software example template

Target hardware information

SOPC Information File name: C:\AlteraPrj\DE10liteEmbed\Embed.sopcinfo

CPU name: nios2_gen2_0

Application project

Project name: Embed_System

☑ Use default location

Project location: C:\AlteraPrj\DE10liteEmbed\software\Embed_System

Project template

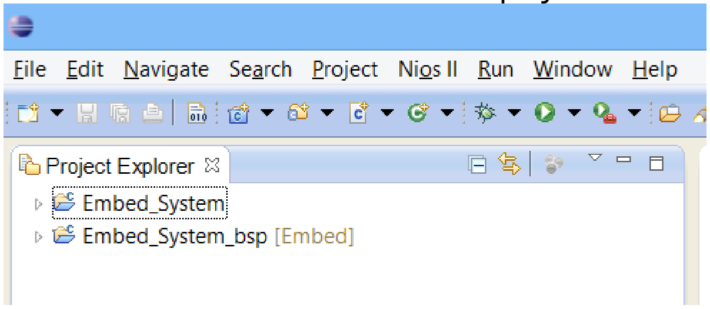
Templates: Blank Project, Board Diagnostics, Count Binary, Float2 Functionality, Float2 GCC, Float2 Performance, Hello Freestanding

Template description: Blank Project creates an empty project to which you can add your code.

For details, click Finish to create the project and refer to the readme.txt file in the project directory.

Click the **Finish** button.

The tool will create two new software project directories



Embed_System

Embed_System_bsp [Embed]

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

a. The application software project itself - this is 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.

Initial content of the project

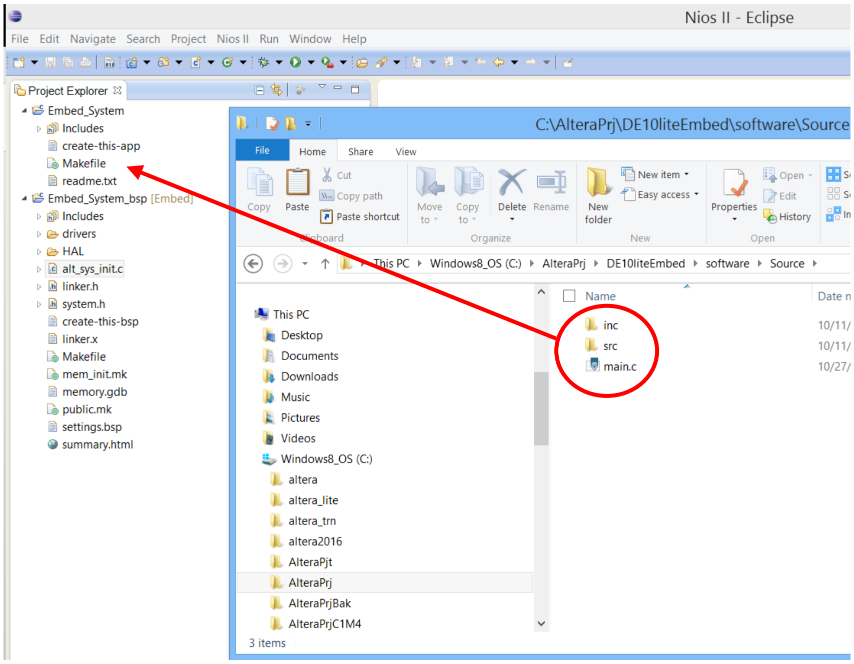
Since you chose the "blank" project template, there are no source files in the application project directory at this time. The BSP contains a directory of software drivers as well as a system.h header file, system initialization source code and substantial other software infrastructure.

CONGRATULATIONS!!

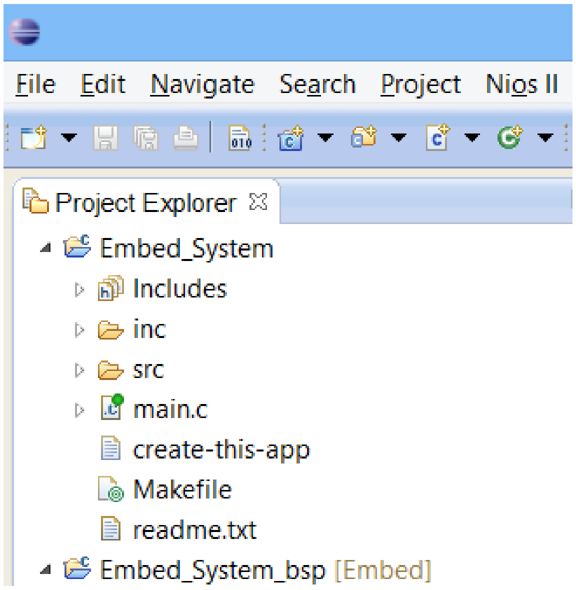
You’ve created a new software project.

ADD SOURCE CODE TO THE PROJECT

In Windows Explorer locate the project directory which contains a directory called “**source**”. 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, “Embed_System”.



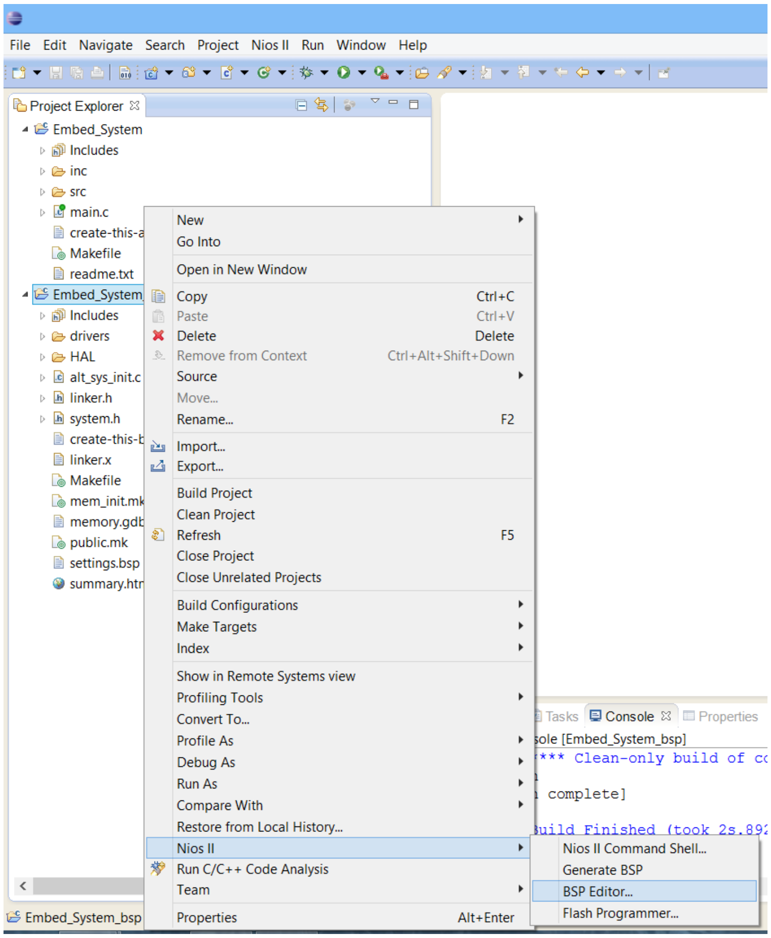
- **Select** the 3 files and **drag** them over the “Embed_System” directory in the SBT window and **drop** the files onto the project folder.
- This should cause the source files to be physically copied into the file system location of the software project directory and register these source files within the Eclipse workspace so that they appear in the Project Explorer file listing.
- Select: OK if it asks you to Copy files and folders.



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

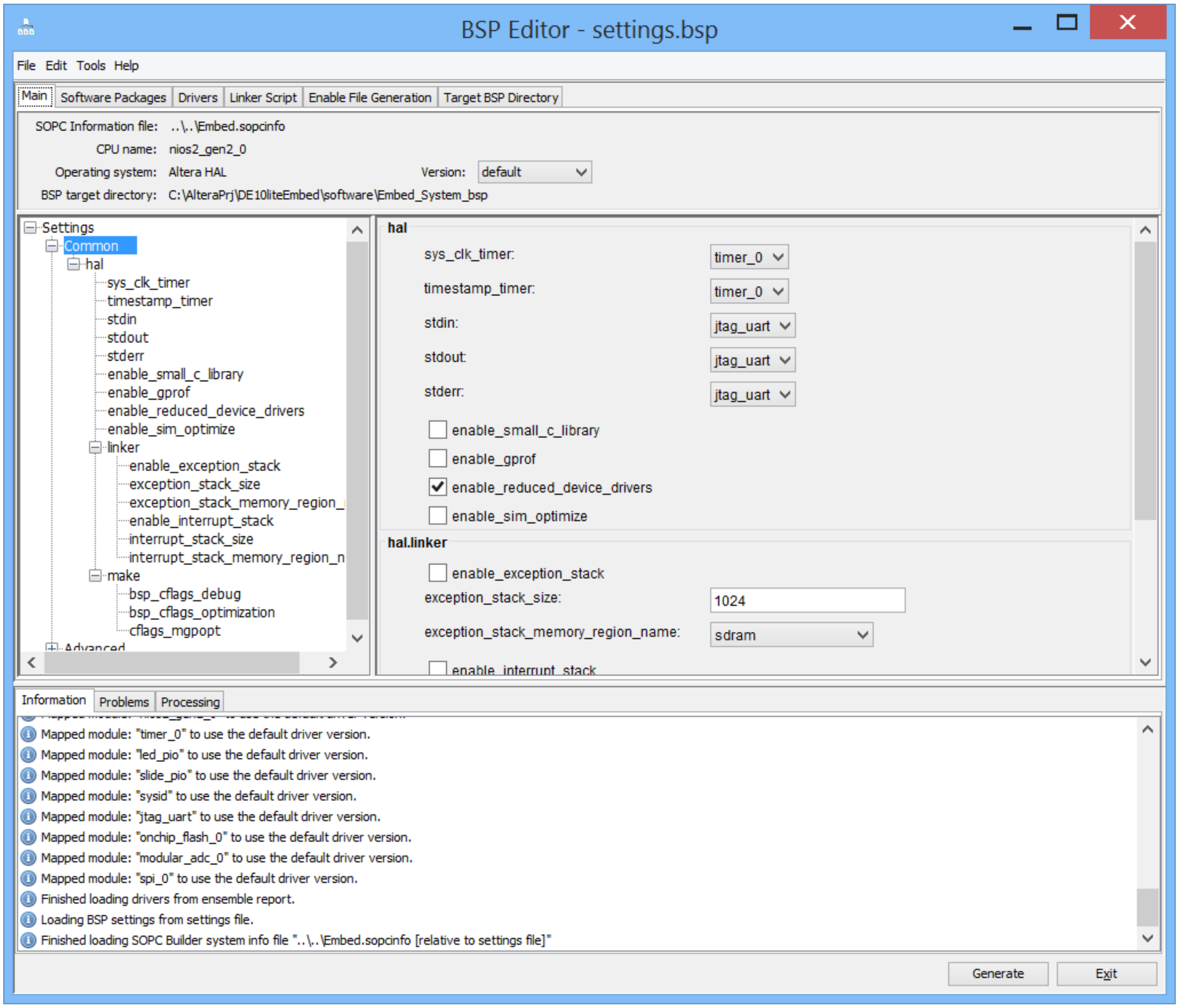


```
sole [Embed_System_bsp]
*** Clean-only build of co
complete]

uild Finished (took 2s.892
```

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

In the “Common” settings view, change the following settings:

- Confirm that the **timer_0** peripheral is selected as the hardware for the **sys_clk_timer**.
- Select the **timer_0** peripheral as the hardware for the **timestamp_timer**.
- Check the box next to **enable_reduced_device_drivers**.
- Select the **onchip_ram** is selected as the linker target for both the **exception_stack_memory_region_name** and the **interrupt_stack_memory_region_name**.



SOPC Information file: ..\..\Embed.sopcinfo

CPU name: nios2_gen2_0

Operating system: Altera HAL Version: default

BSP target directory: C:\AlteraPrj\DE10liteEmbed\software\Embed_System_bsp

hal

sys_clk_timer: timer_0

timestamp_timer: timer_0

stdin: jtag_uart

stdout: jtag_uart

stderr: jtag_uart

☐ enable_small_c_library

☐ enable_gprof

☑ enable_reduced_device_drivers

☐ enable_sim_optimize

hal.linker

☐ enable_exception_stack

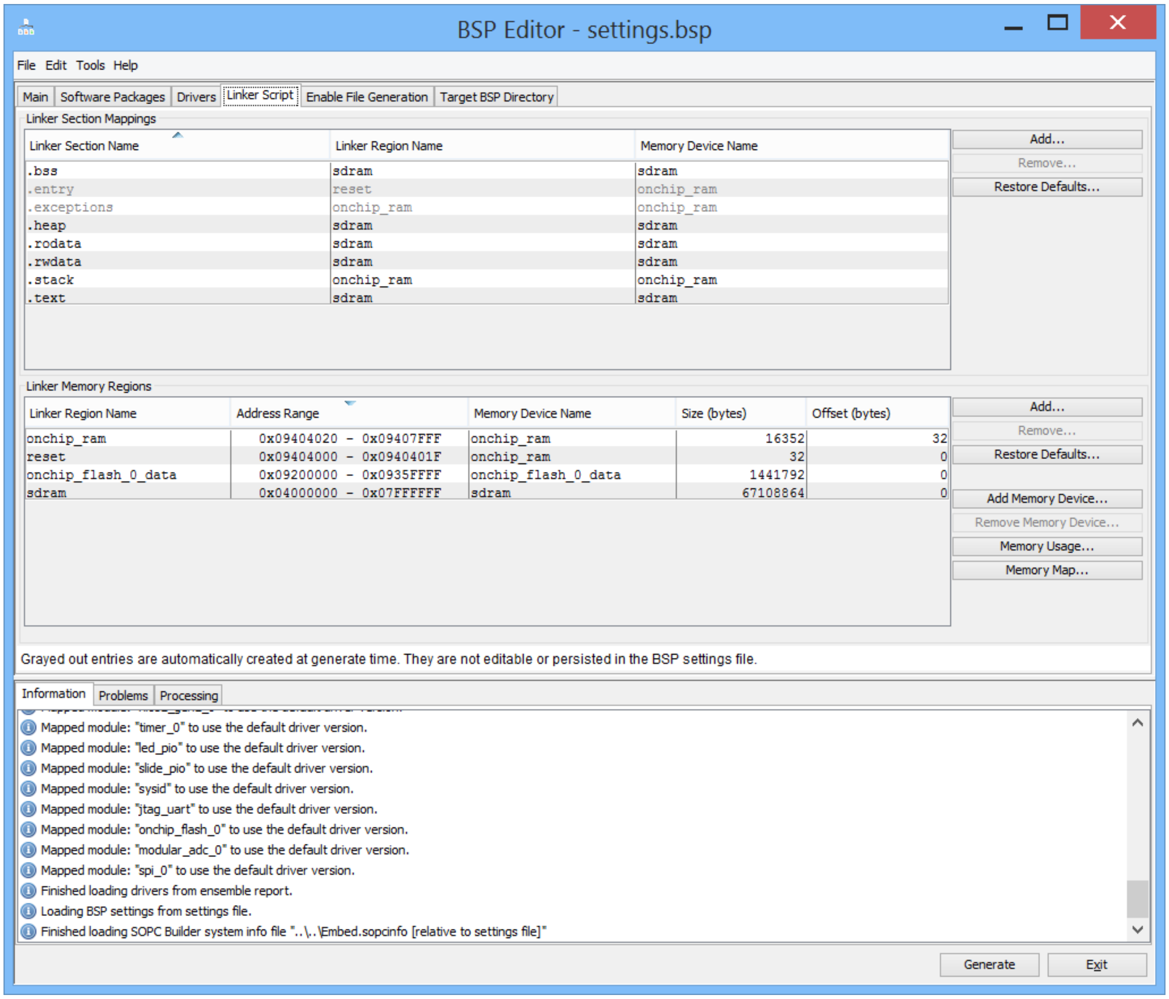
exception_stack_size: 1024

exception_stack_memory_region_name: sdram

☐ enable_interrupt_stack

```
Mapped module: "timer_0" to use the default driver version.
Mapped module: "led_pio" to use the default driver version.
Mapped module: "slide_pio" to use the default driver version.
Mapped module: "sysid" to use the default driver version.
Mapped module: "jtag_uart" to use the default driver version.
Mapped module: "onchip_flash_0" to use the default driver version.
Mapped module: "modular_adc_0" to use the default driver version.
Mapped module: "spi_0" to use the default driver version.
Finished loading drivers from ensemble report.
Loading BSP settings from settings file.
Finished loading SOPC Builder system info file "..\..\Embed.sopcinfo [relative to settings file]"
```

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 project, we have two volatile memories in the system, so we will use this handy tool to adjust all the stack memory region to on chip ram instead of the sdram. It would be possible to assign all the memory regions to onchip ram, but to do so we would have to make the onchip ram larger than 16k.



Linker Section Mappings

Linker Section Name	Linker Region Name	Memory Device Name
.bss	sdram	sdram
.entry	reset	onchip_ram
.exceptions	onchip_ram	onchip_ram
.heap	sdram	sdram
.rodata	sdram	sdram
.rwdata	sdram	sdram
.stack	onchip_ram	onchip_ram
.text	sdram	sdram

Linker Memory Regions

Linker Region Name	Address Range	Memory Device Name	Size (bytes)	Offset (bytes)
onchip_ram	0x09404020 - 0x09407FFF	onchip_ram	16352	32
reset	0x09404000 - 0x0940401F	onchip_ram	32	0
onchip_flash_0_data	0x09200000 - 0x0935FFFF	onchip_flash_0_data	1441792	0
sdram	0x04000000 - 0x07FFFFFF	sdram	67108864	0

Grayed out entries are automatically created at generate time. They are not editable or persisted in the BSP settings file.

```
Mapped module: "timer_0" to use the default driver version.
Mapped module: "led_pio" to use the default driver version.
Mapped module: "slide_pio" to use the default driver version.
Mapped module: "sysid" to use the default driver version.
Mapped module: "jtag_uart" to use the default driver version.
Mapped module: "onchip_flash_0" to use the default driver version.
Mapped module: "modular_adc_0" to use the default driver version.
Mapped module: "spi_0" to use the default driver version.
Finished loading drivers from ensemble report.
Loading BSP settings from settings file.
Finished loading SOPC Builder system info file "..\..\Embed.sopcinfo [relative to settings file]"
```

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

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.

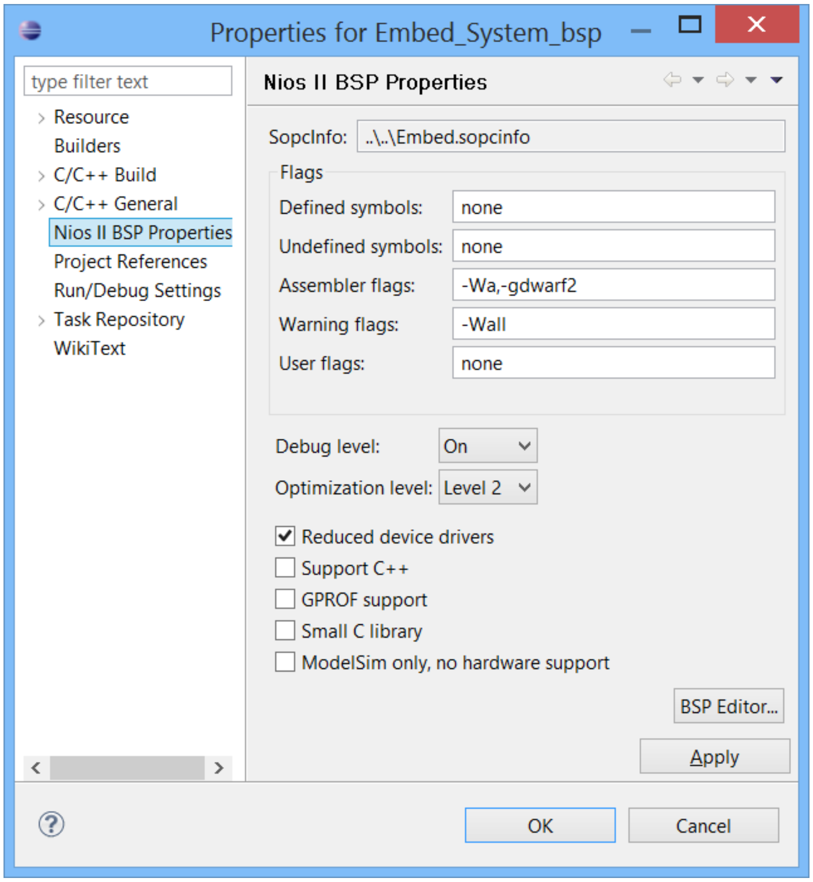
- **Right click** on the **Embed_System_bsp** software project and select **Properties** from the right-click menu.

- On the left-hand menu, select the **Nios II BSP Properties** tab

- During compilation, the code may have various levels of optimization which is a tradeoff between code size and performance. Change the **Optimization level** setting to **Level 2**.
- To keep the software footprint compact, choose **Reduced device drivers**.

- Since our software does not make use of C++, **uncheck** "**Support C++**".

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



Nios II BSP Properties

SopcInfo: ..\..\Embed.sopcinfo

Flags

Defined symbols: none

Undefined symbols: none

Assembler flags: -Wa,-gdwarf2

Warning flags: -Wall

User flags: none

Debug level: On

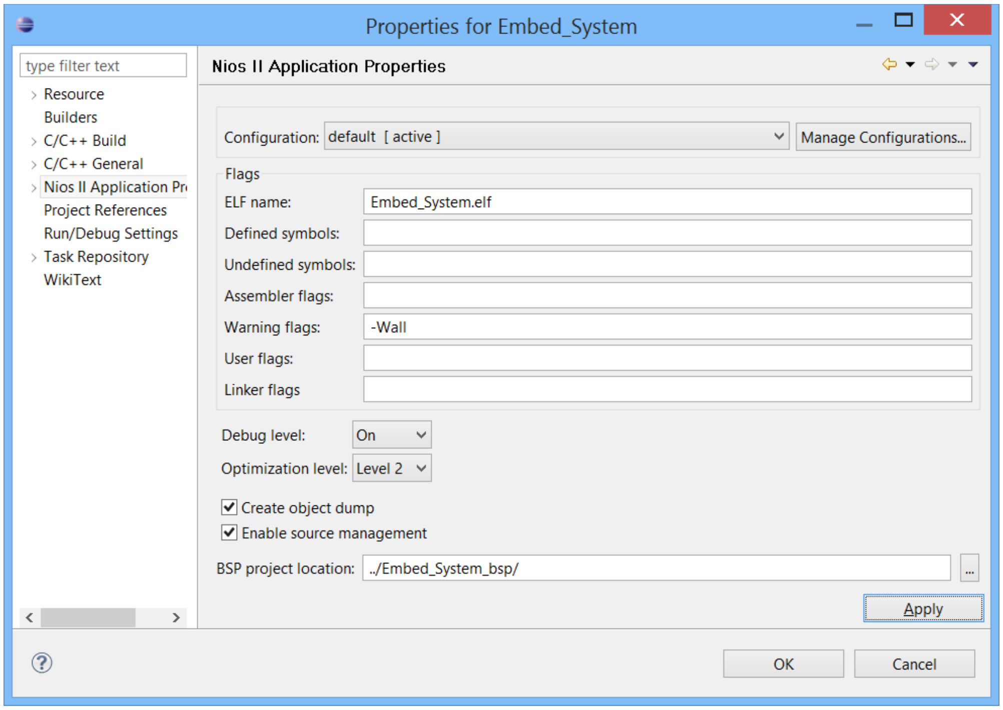
Optimization level: Level 2

☑ Reduced device drivers
☐ Support C++
☐ GPROF support
☐ Small C library
☐ ModelSim only, no hardware support

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 “Embed_System” as well.

- **Right click** on the **Embed_System** 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**.



Nios II Application Properties

Configuration: default [active]

Flags

ELF name: Embed_System.elf
Defined symbols:
Undefined symbols:
Assembler flags:
Warning flags: -Wall
User flags:
Linker flags

Debug level: On
Optimization level: Level 2

☑ Create object dump
☑ Enable source management

BSP project location: ../Embed_System_bsp/

5. BUILD THE SOFTWARE PROJECT

- **Right click** the **Embed_System_bsp** software project and choose **Build Project** to build the board support package.
- When that build completes, **right click** the **Embed_System** 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.

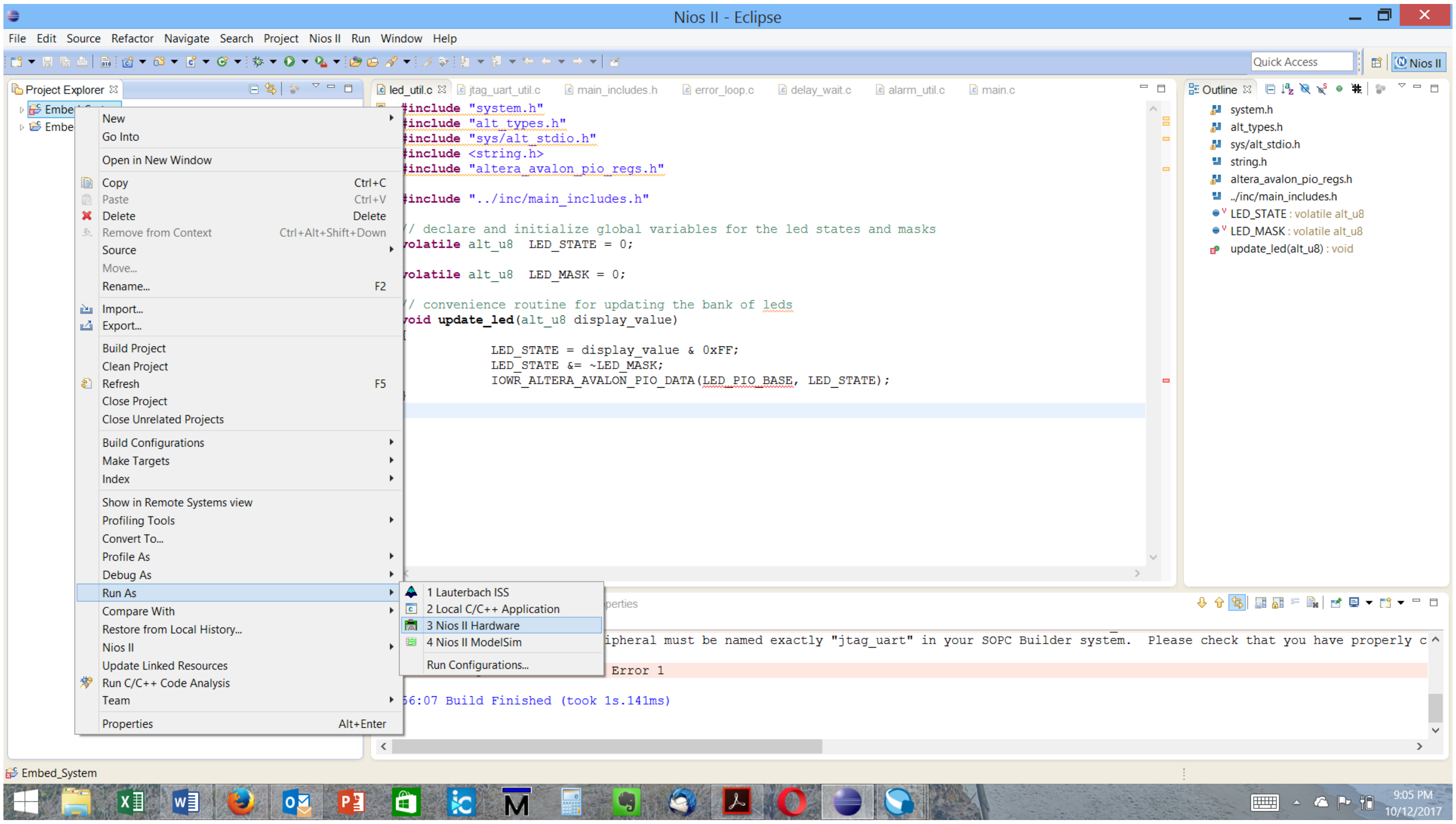
6. RUN THE SOFTWARE APPLICATION ON THE TARGET

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. If you have since turned it off, you will need to repeat the programming FPGA section in Module 3. After downloading the .SOF, keeping the DE10-Lite 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**.

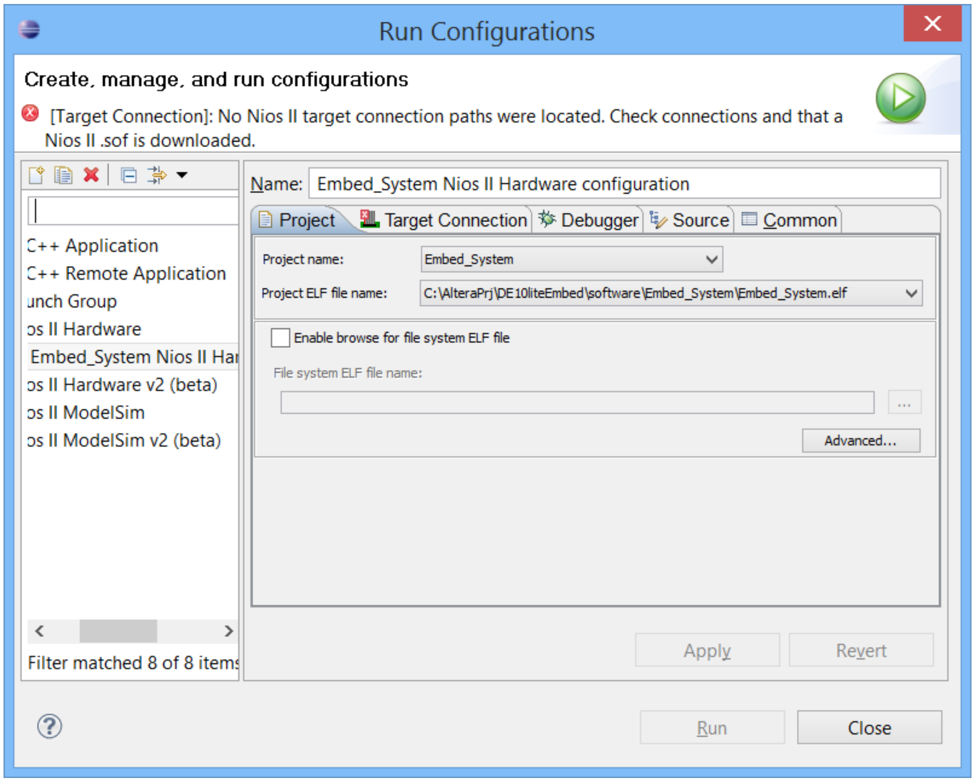


```
#include "system.h"
#include "alt_types.h"
#include "sys/alt_stdio.h"
#include <string.h>
#include "altera_avalon_pio_regs.h"

#include "../inc/main_includes.h"

// declare and initialize global variables for the led states and masks
volatile alt_u8  LED_STATE = 0;

volatile alt_u8  LED_MASK = 0;

// convenience routine for updating the bank of leds
void update_led(alt_u8 display_value)
{
            LED_STATE = display_value & 0xFF;
            LED_STATE &= ~LED_MASK;
            IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, LED_STATE);
}
```

```
ipheral must be named exactly "jtag_uart" in your SOPC Builder system.  Please check that you have properly c
 Error 1

56:07 Build Finished (took 1s.141ms)
```

This will re-build the software project to create an up–to-date executable and then download the code into memory on our DE10-Lite hardware. The debugger resets the Nios II processor, and it executes the downloaded code.

You may see a window indicating no connection was found:



Create, manage, and run configurations

[Target Connection]: No Nios II target connection paths were located. Check connections and that a Nios II .sof is downloaded.

Name: Embed_System Nios II Hardware configuration

Project name: Embed_System

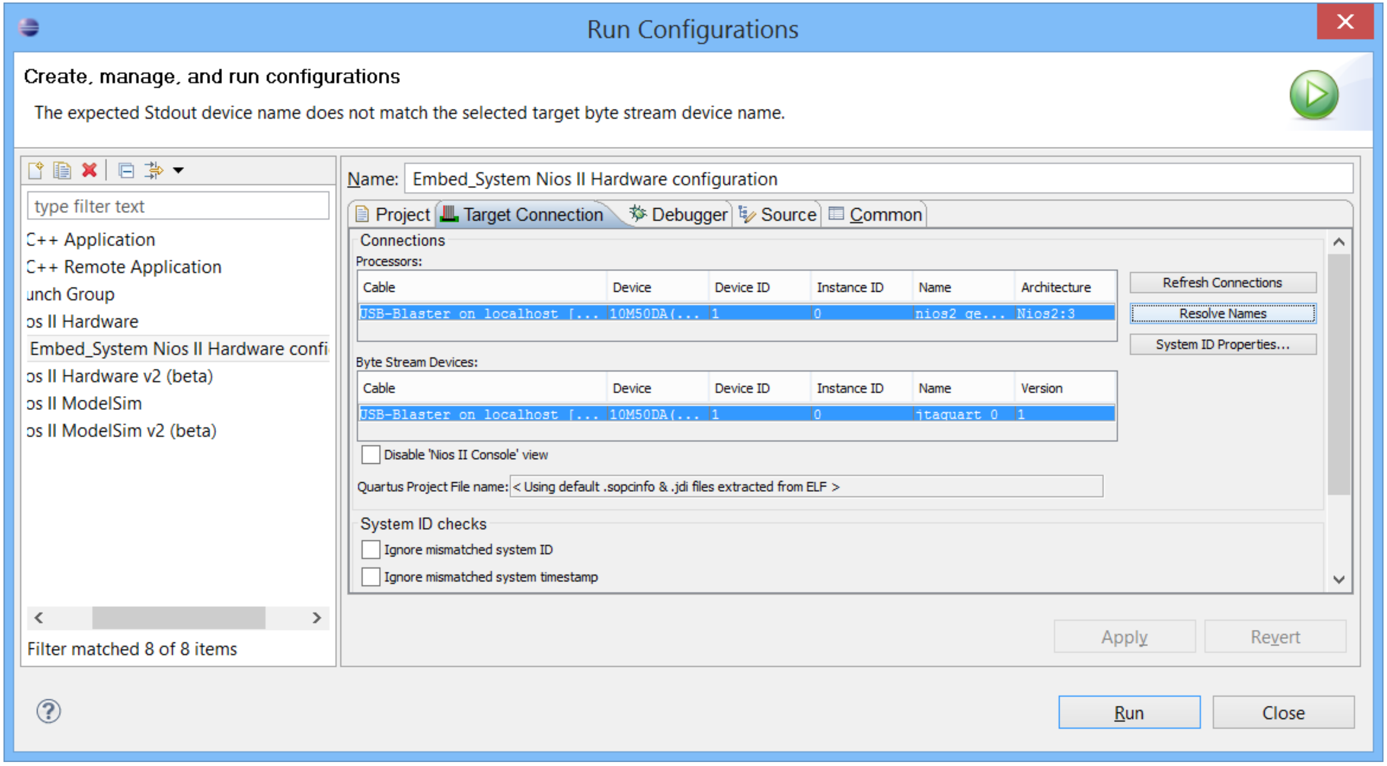
Project ELF file name: C:\AlteraPrj\DE10liteEmbed\software\Embed_System\Embed_System.elf

☐ Enable browse for file system ELF file

File system ELF file name:

Filter matched 8 of 8 items

Note: Do not select Nios II Hardware v2 (Beta). Click on the target connections tab, Click on Refresh Connections and then hit Run,



Create, manage, and run configurations

The expected Stdout device name does not match the selected target byte stream device name.

Name: Embed_System Nios II Hardware configuration

Connections

Processors:

Cable	Device	Device ID	Instance ID	Name	Architecture
USB-Blaster on localhost [...	10M50DA(...	1	0	nios2 ge...	Nios2:3

Byte Stream Devices:

Cable	Device	Device ID	Instance ID	Name	Version
USB-Blaster on localhost [...	10M50DA(...	1	0	jtaguart 0	1

☐ Disable 'Nios II Console' view

Quartus Project File name: < Using default .sopcinfo & .jdi files extracted from ELF >

System ID checks

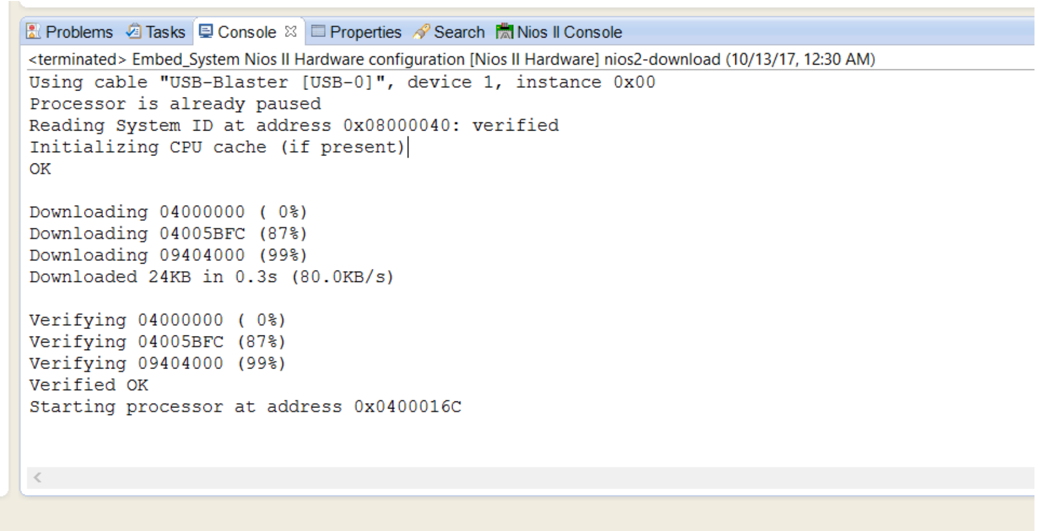
☐ Ignore mismatched system ID

☐ Ignore mismatched system timestamp

Filter matched 8 of 8 items

EXAMINE SOFTWARE APPLICATION OUTPUT

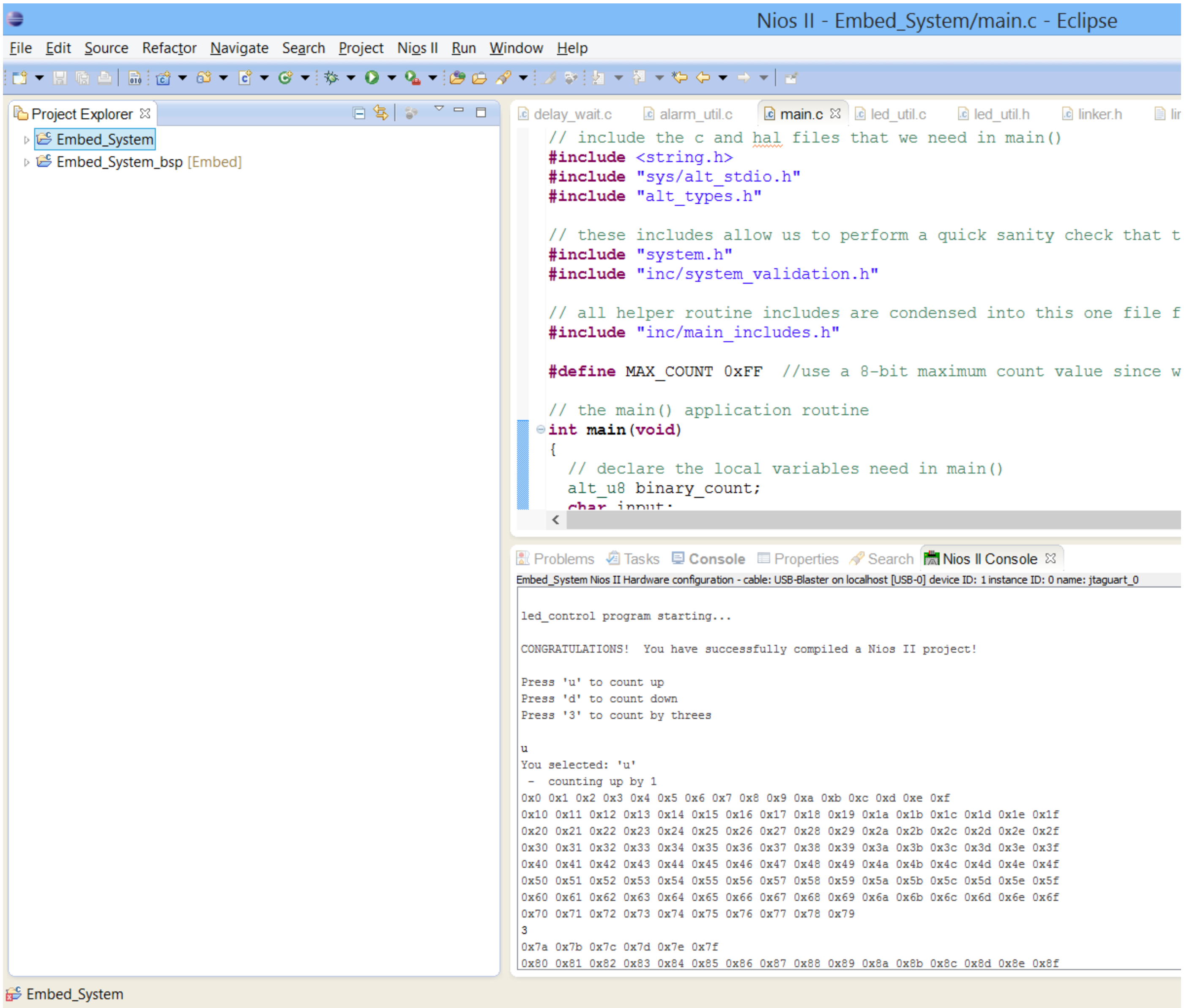
Once the application starts executing, it will relay the messages back to the Nios SBT via the JTAG UART interface and the messages from this interface will be placed into a new “**Nios II Console**” pane within the Eclipse GUI.



<terminated> Embed_System Nios II Hardware configuration [Nios II Hardware] nios2-download (10/13/17, 12:30 AM)

```
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Processor is already paused
Reading System ID at address 0x08000040: verified
Initializing CPU cache (if present)
OK

Downloading 04000000 ( 0%)
Downloading 04005BFC (87%)
Downloading 09404000 (99%)
Downloaded 24KB in 0.3s (80.0KB/s)

Verifying 04000000 ( 0%)
Verifying 04005BFC (87%)
Verifying 09404000 (99%)
Verified OK
Starting processor at address 0x0400016C
```

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



```
// include the c and hal files that we need in main()
#include <string.h>
#include "sys/alt_stdio.h"
#include "alt_types.h"

// these includes allow us to perform a quick sanity check that t
#include "system.h"
#include "inc/system_validation.h"

// all helper routine includes are condensed into this one file f
#include "inc/main_includes.h"

#define MAX_COUNT 0xFF  //use a 8-bit maximum count value since w

// the main() application routine
int main(void)
{
  // declare the local variables need in main()
  alt_u8 binary_count;
  char input;
```

Embed_System Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtaguart_0

```

led_control program starting...

CONGRATULATIONS!  You have successfully compiled a Nios II project!

Press 'u' to count up
Press 'd' to count down
Press '3' to count by threes

u
You selected: 'u'
 -  counting up by 1
0x0 0x1 0x2 0x3 0x4 0x5 0x6 0x7 0x8 0x9 0xa 0xb 0xc 0xd 0xe 0xf
0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f
0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x2a 0x2b 0x2c 0x2d 0x2e 0x2f
0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f
0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f
0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57 0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f
0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x6a 0x6b 0x6c 0x6d 0x6e 0x6f
0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77 0x78 0x79
3
0x7a 0x7b 0x7c 0x7d 0x7e 0x7f
0x80 0x81 0x82 0x83 0x84 0x85 0x86 0x87 0x88 0x89 0x8a 0x8b 0x8c 0x8d 0x8e 0x8f
```

7. INTERACT WITH THE SOFTWARE APPLICATION

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

EDIT THE APPLICATION

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

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

update_led( )

- In the beginning of the subroutine add the following line:

display_value = ~ display_value;

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

CONGRATULATIONS!!

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

To explore further: Write an application for the NIOS that reads the accelerometer and drives a value proportional to its output on the LEDs.

CONGRATULATIONS!!

You have completed Module 4!

III. DELIVERABLES

Deliverables include:

1. Recorded Observations, Test Data, and Images

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

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

Submit the **DE10liteEmbed** project you created. For Module 4, starting with the top level of each part, zip up the entire directory including subfolders, and submit the zip files. This will allow us to replicate your work and help provide feedback on any errors you encounter. With each submission include a ReadMe.txt if appropriate to describe and list the locations of individual file deliverables.

IV. EVALUATION

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

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

REFERENCES

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

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