**California State University, Fresno   
Lyles College of Engineering   
Electrical and Computer Engineering Department**   
**TECHNICAL REPORT**   
   
Assignment: Number 5  
Experiment Title: HW/SW Co-Design of an Embedded System on FPGA  
Course Title: ECE 178 (Embedded Systems)   
Instructor: Dr. Reza Raeisi   
   
 Prepared by: Anthony Herrick   
 Student ID #300144976  
Date Submitted: 10/09/2022

**INSTRUCTOR SECTION**   
   
Comments: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
   
Final Grade: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

TABLE OF CONTENTS

[Objective 4](#_Toc116242898)

[Hardware Requirements 4](#_Toc116242899)

[Software Requirements 4](#_Toc116242900)

[Background 4](#_Toc116242901)

[Project Overview 7](#_Toc116242902)

[Project Procedure 8](#_Toc116242903)

[Conclusion 30](#_Toc116242904)

[Appendix A (Part 1 Code) 31](#_Toc116242905)

[Appendix B (Part 2 Code) 33](#_Toc116242906)

[Appendix C (Part 3 Code) 36](#_Toc116242907)

[Appendix D (Part 4 Code) 37](#_Toc116242908)

[Appendix E (Part 5 Code) 40](#_Toc116242909)

LIST OF FIGURES

[1. Soft Core Embedded System Design 5](#_Toc116242871)

[2. 7-Segment Display Address Spaces 6](#_Toc116242872)

[3. 7-Segment Display Breakdown 7](#_Toc116242873)

[4. Register Values Before Call 8](#_Toc116242874)

[5. After Entering the Divide Subroutine 9](#_Toc116242875)

[6. Values After the First CONT loop 9](#_Toc116242876)

[7. Part 1 Final Values 10](#_Toc116242877)

[8. Value of Switches being loaded into r4 11](#_Toc116242878)

[9. Loading the Byte Sized Value for the 7 Segment 11](#_Toc116242879)

[10. Board After Value is Loaded into 7-Segment 12](#_Toc116242880)

[11. Squared Value on the LEDs 13](#_Toc116242881)

[12. After SRLI 14](#_Toc116242882)

[13. After And 15](#_Toc116242883)

[14. After r10 is iterated 16](#_Toc116242884)

[15. Starting Values Entering the Fourth Loop 17](#_Toc116242885)

[16. Fourth Loop After srli and and 18](#_Toc116242886)

[17. End of Part Three Program Registers 19](#_Toc116242887)

[18. Part 3 Test Case Results 20](#_Toc116242888)

[19. Registers before entering Ones Loop 21](#_Toc116242889)

[20. Registers after exiting Ones Subroutine 22](#_Toc116242890)

[21. Before Entering Ones subroutine Second Time 23](#_Toc116242891)

[22. After Exiting Ones Subroutine 24](#_Toc116242892)

[23. Final Values Of Program 25](#_Toc116242893)

[24. 7 Segment Part 4 26](#_Toc116242894)

[25. Starting values after Initialization 27](#_Toc116242895)

[26. Registers after Returning from Count 28](#_Toc116242896)

[27. Part 5 LEDs 29](#_Toc116242897)

# Objective

Upon the completion of this lab, one will have co-designed an embedded soft-core system and application development in an FPGA design environment. We will be using Intel’s EDA design tool Qsys, in conjunction with the Quartus software, and Intel FPGA monitor.

# Hardware Requirements

* Computer with Intel FPGA Monitor program 16.1
* Computer with Quartus Prime 16.1.
* DE2-115 Board
* A-B USB Cable

# Software Requirements

* Intel FPGA monitor program 16.1 or greater
* Quartus Prime with Qsys, version 16.1.

# Background

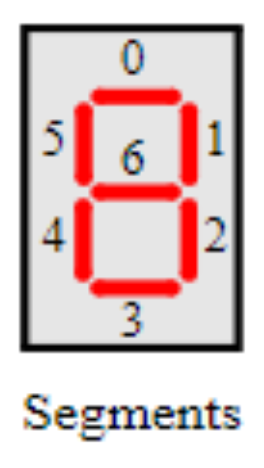
In order to write to a 7-segment display, we have two address spaces, one is for the four most significant bits and the second for the four least significant bits. This is because the DE2-115 board has eight 7-Segment displays. From the design in the last lab, the address spaces for these were, 0x2020 and 0x2030. The typical address spaces for the DE2-115 board can be seen below.

Diagram

Description automatically generated

## 7-Segment Display Address Spaces

Now, from this, we also need to know how to manipulate the 7-Segments in the way that we want. So, being that the display is an active low device, in order to light up a certain display we must pass a 1 into the bit that we want to light up. Now, the segments are controlled individually, so, passing a 7 bit value of 0b1111000 will turn on the bits corresponding to bit zero, one, and two. How these bits correspond to the actual 7 segment display can be seen below.



## 7-Segment Display Breakdown

With this knowledge we can begin this project.

# Project Overview

For this lab, we will be using the soft-core embedded system that was developed in a previous project in order to perform a variety of tasks on the DE2-115 board using a C/C++ runtime enviroment. So, to begin, we must first download our system onto the board, and set up a NIOS II Eclipse enviroment. Next, we create a program to request an option one through 6. Options one through four turn on that respective green LED, option 5 will scroll left and 6 scroll right. Lastly, we will create a program to represent the maximum of an 8 bit unsigned and 8 bit signed number as well as the minimum of a 8 bit signed number.

.

# Project Procedure

To begin this lab, we first open the Quartus Prime Programmer. The programmer can be seen in the figure below.

Graphical user interface, text

Description automatically generated

## Quartus Prime Programmer

Next, we enter the hardware setup. Here we select the USB-Blaster and click add hardware. This will add the USB-Blaster to our hardware list and begin to allow us to download the system onto the board. This window can be seen below.

Graphical user interface, text, application, email

Description automatically generated

## Hardware Setup

Next after clicking auto detect, we are able to select our board, and in our case the device is the EP4CE115 for the DE2-115 board. So, seen below, we select the device and click okay.

Graphical user interface, text, application

Description automatically generated

## Selecting Device

Next, we must download the sof file onto the board. So, we double click in the file section and add the sof file from a previous lab, Lab 3.

Graphical user interface, text, email

Description automatically generated

## Adding the File to be Downloaded onto the Board

After adding the sof file, it is fairly simple, we must next check the program configuration box, and click start. This will download the system onto the DE2-115 board. A successful download can be seen in the figure below, marked by the progress bar in the top right being at 100%, and the successes in the console.

Graphical user interface, text, application

Description automatically generated

## Completed Download onto the Board

After downloading the system on the board, we then launch NIOS II Software Build tools for Eclipse. This program can be found in the tools drop down menu in Quartus Prime. From here, we create a NIOS II Application and BSP from template, and the pop-out window can be seen below.

Graphical user interface, application

Description automatically generated

## NIOS II Application and BSP from Template

After setting the SOPC Info file to the correct one corresponding to our board, setting the CPU Name, as well as the project name, and location, we select the Hello World from the project templates. We can see this in the figure below.

Graphical user interface, text, application

Description automatically generated

## NIOS II Application and BSP From Template Settings

This will then create a simple hello world program as well as create the necessary files and folders for the project. The Program can be seen in the figure below.

Graphical user interface, application, Word

Description automatically generated

## Hello\_world.c Program

This is a very simple, program which just prints “Hello from Nios II” in the console. We can see the project breakdown in the project explorer below.

Graphical user interface, application

Description automatically generated

## Project Files

From this, we can see the important files that we will be using in this program. System.h shows all of the values of our PIO devices, a well as other useful information about the system. However, in order to run the program a couple steps must be completed, seen in the figure below

Graphical user interface, application

Description automatically generated

## Building the project

To run this project, we first right click the project and click build project. Then, we go into the BSP editor, by right clicking the BSP project, and clicking properties. This will then open the window seen below.

Graphical user interface, text, application, email

Description automatically generated

## BSP File Properties

After opening this window, we select NIOS II BSP Properties from the table on the left. The following figure with the NIOS II BSP Properties then appears.

Graphical user interface, text, application, email

Description automatically generated

## NIOS II BSP Properties

Here we can see the BSP properties for the project, by going into the BSP Editor we can see the settings more in depth.

Graphical user interface, text, application, email

Description automatically generated

## BSP Editor

In the BSP Editor main, we can see the HAL settings along with other properties, we can also go into the different tabs in the editor in order to see things like the Software Packages seen in the figure below, drivers immediately following that, and lastly the Linker Script following that.

Graphical user interface, text, application

Description automatically generated

## Software Packages Settings

Graphical user interface, text, application, email

Description automatically generated

## Drivers Settings

A screenshot of a computer

Description automatically generated with medium confidence

## Linker Script Settings

Here we can see all the different settings we have in the BSP. However, for the purpose of this project, we will be leaving all of these settings default. Next, to run our program, we right click the project, Run as, then NIOS II Hardware. Seen in the figure below.

Graphical user interface, application

Description automatically generated

## Running As Nios II Hardware

After selecting Run as Nios II Hardware, a window will appear to set up the connection of the USB blaster. Here we refresh the connections and select the USB-Blaster. This is shown in the figure below.

Graphical user interface, text, application, email

Description automatically generated

## Setting the Target Connection

After the Target connection is set, run as Nios II Hardware again, and the program will run, when the program runs the following is displayed in the console.

Graphical user interface, application

Description automatically generated

## Output from Hello\_World

So, now that this is done, we create a program for part 1 which is to request a user input of 1 through 6. An input of one through four will turn on that respective LED, where an input of 5 to 6 will scroll left and scroll right respectively. The code for this part can be seen in appendix A.

The approach that was used for Part 1, was to first print the options that the user had to the console. They will select an option, options one through four will simply output 1, 2, 4, 8 to the Green LEDs PIO base address respectively. Option 5 will set the value to out – (out/2). This is so that, if you are at 8, the 4th LED is on, then scroll right will set your new value to output to 4, which is the third bit essentially creating a walking ones.

Part 2 was to display the min and max of 8 bit unsigned values as well as the max of an 8 bit unsigned value to the LEDs, as well as its decimal equivalent to the seven segment displays. The code for this part can be seen in Appendix B.

# Conclusion

This lab accomplished its goal of using the softcore embedded system formed in the last exercise, and Intel’s FPGA design enviroment in order to complete certain tasks. We performed binary to decimal conversions, displayed an input value from the switches on the 7-Segment Display, and the square of that value on the LEDs, counted the longest string of ones from a list, and created a counter in certain memory bits. This further bolstered the knowledge of PIO devices and how they work in a NIOS II coding enviroment.

# Appendix A (Part 1 Code)

**#include** <stdio.h>

**#include** <system.h>

**#include** <alt\_types.h>

**#include** "altera\_avalon\_pio\_regs.h"

**int** **main**()

{

**printf**("Options:\n1. Turn on the first green LED\n2. Turn on the second green LED\n");

**printf**("3. Turn on the third green LED\n3\4. Turn on the fourth green LED\n");

**printf**("5. Scroll Left\n6. Scroll Right\n");

**int**\* userin = 0;

**int** option = 0;

alt\_u8 out = 00000000;

**while**(1){

**printf**("\nSelect an Option:");

**scanf**("%d", userin);

option = \*userin;

**switch**(option){

**case** 1:

out = 1;

IOWR(GREEN\_LEDS\_BASE, 0, out);

**break**;

**case** 2:

out = 2;

IOWR(GREEN\_LEDS\_BASE, 0, out);

**break**;

**case** 3:

out = 4;

IOWR(GREEN\_LEDS\_BASE, 0, out);

**break**;

**case** 4:

out = 8;

IOWR(GREEN\_LEDS\_BASE, 0, out);

**break**;

**case** 5:

**if** (out == 8)

out = 1;

**else**

out = out + out;

IOWR(GREEN\_LEDS\_BASE, 0, out);

**break**;

**case** 6:

**if** (out == 1)

out = 8;

**else**

out = out - (out/2);

IOWR(GREEN\_LEDS\_BASE, 0, out);

**break**;

**default**:

**return** 0;

}

}

**return** 0;

}

# Appendix B (Part 2 Code)

**#include** <stdio.h>

**#include** <system.h>

**#include** <alt\_types.h>

**#include** "altera\_avalon\_pio\_regs.h"

**int** **main**(){

alt\_u32 outSevenSeg =0;

**int** Max8Bitu = 255;

**int** Max8Bits = 127;

**int** Min8Bits = -127;

outSevenSeg = 4294967295;

IOWR(SEVSEG4MSB\_BASE, 0, outSevenSeg);

**while**(1){

Display(Max8Bitu);

IOWR(RED\_LEDS\_BASE, 0, Max8Bitu);

delay();

Display(Max8Bits);

IOWR(RED\_LEDS\_BASE, 0, Max8Bits);

delay();

Display(Min8Bits);

IOWR(RED\_LEDS\_BASE, 0, (0b000000000011111111 & Min8Bits));

delay();

}

**printf**("complete");

**return** 0;

}

**void** **Display**(**int** Value){

**int** FourthSeg;

**if** (Value < 0){

Value = abs(Value);

FourthSeg = 0b0111111;

}

**else**

FourthSeg = 0b1111111;

**int** SEVENSEGLUT[9] = {0b01000000, 0b01111001, 0b00100100, 0b00110000, 0b00011001, 0b00010010, 0b00000010, 0b01111000, 0b00000000, 0b00011000};

**int** R;

**int** Count = 0;

**int** toDisplay = 0;

**int** shiftVal = 0;

**while** (Value != 0){

R = Value % 10;

toDisplay = toDisplay + (SEVENSEGLUT[R] << shiftVal);

Value = (Value - R) / 10;

shiftVal = shiftVal + 8;

}

toDisplay = toDisplay + (FourthSeg << shiftVal);

IOWR(SEVSEGMENT\_4LSB\_BASE, 0, toDisplay);

**return**;

}

**void** **delay**(){

**int** delay = 0;

**while**(delay < 2000000)

{

delay++;

}

**return**;

}