

**California State University, Fresno**  
**Lyles College of Engineering**  
**Electrical and Computer Engineering Department**

**TECHNICAL REPORT**

Assignment: Number 6  
Experiment Title: EDS-SBT For Eclipse I/o Communication and Interrupt Processing  
Course Title: ECE 178 (Embedded Systems)  
Instructor: Dr. Reza Raeisi

Prepared by: Anthony Herrick  
Student ID #300144976  
Date Submitted: 10/30/2022

**INSTRUCTOR SECTION**

Comments: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Final Grade: \_\_\_\_\_

## TABLE OF CONTENTS

Objective .....	4
Hardware Requirements.....	4
Software Requirements .....	4
Background .....	4
Project Overview .....	6
Project Procedure .....	8
Data Analysis .....	18
Conclusion .....	27
Appendix A (Top Level Instantiation).....	28
Appendix B (Part 1 Code).....	30
Appendix C (Part 2 Code).....	33

## LIST OF FIGURES

1.	7-Segment Display Address Spaces .....	5
2.	7-Segment Display Breakdown .....	6
3.	QSys Keys Properties .....	8
4.	QSys KEYS PIO Connections.....	9
5.	Pin Planner for KEYS.....	10
6.	Adding the File to be Downloaded onto the Board .....	11
7.	Completed Download onto the Board .....	12
8.	Nios II Application and BSP from Template.....	13
9.	Nios II Application and BSP From Template Settings.....	14
10.	Incr and Decr functions.....	15
11.	Delay Function.....	15
12.	Key2 and Key3 ISR Functions .....	16
13.	Handle_key_interrupts Function.....	17
14.	Pio_init and Main Functions .....	17
15.	NiosII Console for Part 1 .....	18
16.	Initial Green LED value .....	19
17.	Set Value to 1 Using Key2.....	20
18.	Count up using Key3 Sw0 High (Mid Count) .....	21
19.	Finished Count up using Key3 Sw0 High.....	22
20.	Finished Counting Down to 1 .....	23
21.	Nios II Output for Part 2 .....	23
22.	Set the value on the Seven Segment to 0 .....	24
23.	Counting up.....	25
24.	Switch from Incrementing to Decrementing.....	26

## Objective

Upon the completion of this lab, one will understand how to develop the DE2-115 board in QSys and Quartus Prime in order to handle interrupts. Along with this, we will have developed the comprehension of the Interrupts in the C/C++ environment Eclipse Software Build tools.

## 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

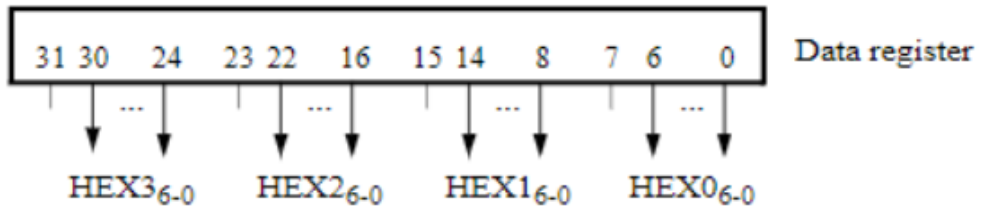
- Quartus Prime with Qsys, version 16.1.

## Background

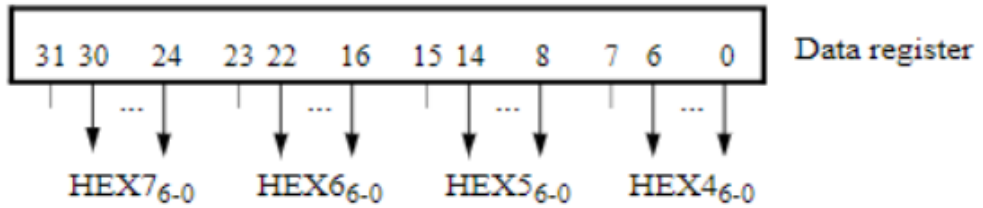
In order to write to a 7-segment display, we have two address spaces, one is for the four most significant bits and the second for the four least significant bits. This is because the DE2-115 board has eight 7-Segment displays. From the design in the last lab, the address spaces for these were, 0x2020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

Address

0x10000020

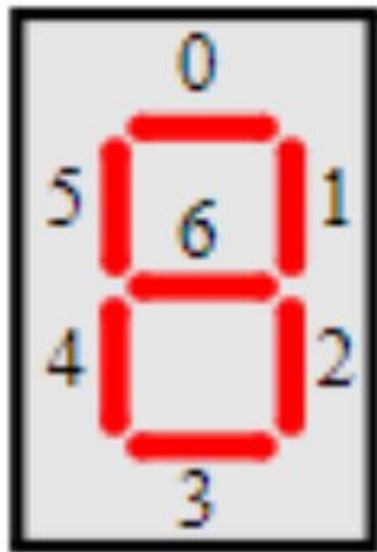


0x10000030



## 1. 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.



Segments

## 2. 7-Segment Display Breakdown

In order to generate an interrupt, there is three main steps that must be accomplished. First is the initialization. Here we set the edge capture pointer, the IRQ Mask, reset the edge capture register, and register the ISR. Setting the IRQ mask is done with the `IOWR_ALTERA_AVALON_PIO_IRQ_MASK` function. This function takes two arguments, the base address and the mask. So for the keys, using the base address and `0b1100` will enable key 3 and 2. Registering the IRQ is also important which is set with `alt_irq_register` function which takes the IRQ, edge capture pointer, and the function name that we are trying to call with the interrupt. We then have the interrupt handler, and the function for the ISR program.

## Project Overview

For this lab, we will be using the soft-core embedded system that was developed in a previous project with a couple modifications to perform a variety of tasks on the DE2-115 board using a C/C++ runtime enviroment. First, alter the system using Quartus Prime In order to incorporate interrupts. Next, there will be two main code sections. The first section of code will use key 2 and key 3 interrupts. Key 2 will set the value on first 4 green LED bits to 1. Key 3 will count up if pressed and the first switch is high, and down if the first switch is low. The second

program will display a two digit number on the first two HEX displays. Key 1 will set the number to 0. Key 2 will increment the number, and Key 3 will decrement the number.

## Project Procedure

To begin this lab, we first open the QSys, and edit a few things from the softcore embedded system design from Lab 3. First, the KEYS are set to a 4 bit width enabling all four keys, as well as the edge capture register turned on to the rising edge of the clock, and the IRQ generation on with the “Edge” type.

The screenshot shows the 'Parameters' window for the 'KEYS' component in a QSys project. The window is titled 'Parameters' with a close button. Below the title bar, it shows 'System: QsysSystem' and 'Path: KEYS'. The component is identified as 'PIO (Parallel I/O)' with the instance name 'altera\_avalon\_pio'. The window is divided into several sections: 'Basic Settings', 'Output Register', 'Edge capture register', 'Interrupt', and 'Test bench wiring'. In 'Basic Settings', the 'Width (1-32 bits)' is set to 4, 'Direction' is set to 'Input' (selected with a radio button), and 'Output Port Reset Value' is 0x0000000000000000. In 'Output Register', 'Enable individual bit setting/clearing' is unchecked. In 'Edge capture register', 'Synchronously capture' is checked, 'Edge Type' is set to 'RISING', and 'Enable bit-clearing for edge capture register' is unchecked. In 'Interrupt', 'Generate IRQ' is checked, and 'IRQ Type' is set to 'EDGE'. Below this, there are two lines of text: 'Level: Interrupt CPU when any unmasked I/O pin is logic true' and 'Edge: Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled'. In 'Test bench wiring', 'Hardwire PIO inputs in test bench' is unchecked, and 'Drive inputs to field.' is 0x0000000000000000.

Parameters

System: QsysSystem Path: KEYS

PIO (Parallel I/O)  
altera\_avalon\_pio

**Basic Settings**

Width (1-32 bits): 4

Direction:

- ☐ Bidir
- ☒ Input
- ☐ InOut
- ☐ Output

Output Port Reset Value: 0x0000000000000000

**Output Register**

☐ Enable individual bit setting/clearing

**Edge capture register**

☒ Synchronously capture

Edge Type: RISING

☐ Enable bit-clearing for edge capture register

**Interrupt**

☒ Generate IRQ

IRQ Type: EDGE

**Level:** Interrupt CPU when any unmasked I/O pin is logic true  
**Edge:** Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled

**Test bench wiring**

☐ Hardwire PIO inputs in test bench

Drive inputs to field.: 0x0000000000000000

### 3. QSys Keys Properties



Along with the KEY properties we must ensure that the KEYS are set and have an Interrupt ID, which is shown to the right of the PIO device. We can see in the figure below that for the KEYS PIO, the Interrupt is labeled number 1.

System: QsysSystem Path: KEYS								
Use	Connections	Name	Description	Export	Clock	Base	End	I...
<input checked="" type="checkbox"/>		<b>Processor</b>	Nios II (Classic) Processor					
		clk	Clock Input	Double-click to	CLK_S...			
		reset_n	Reset Input	Double-click to	[clk]			
		data_master	Avalon Memory Mapped ...	Double-click to	[clk]			
		instruction_m...	Avalon Memory Mapped ...	Double-click to	[clk]			
		d_irq	Interrupt Receiver	Double-click to	[clk]			
		jtag_debug_m...	Reset Output	Double-click to	[clk]			
		jtag_debug_m...	Avalon Memory Mapped ...	Double-click to	[clk]			
		custom_instru...	Custom Instruction Master	Double-click to				
<input checked="" type="checkbox"/>		<b>JTAG_UART</b>	JTAG UART					
		clk	Clock Input	Double-click to	CLK_S...			
		reset	Reset Input	Double-click to	[clk]			
		avalon_jtag_sl...	Avalon Memory Mapped ...	Double-click to	[clk]			
		irq	Interrupt Sender	Double-click to				
<input checked="" type="checkbox"/>		<b>SYSID</b>	System ID Peripheral					
		clk	Clock Input	Double-click to	CLK_S...			
		reset	Reset Input	Double-click to	[clk]			
		control_slave	Avalon Memory Mapped ...	Double-click to	[clk]			
<input checked="" type="checkbox"/>		<b>On_Chip_Mem</b>	On-Chip Memory (RAM o...					
		clk1	Clock Input	Double-click to	CLK_S...			
		s1	Avalon Memory Mapped ...	Double-click to	[clk1]			
		reset1	Reset Input	Double-click to	[clk1]			
<input checked="" type="checkbox"/>		<b>KEYS</b>	PIO (Parallel I/O)					
		clk	Clock Input	Double-click to	CLK_S...			
		reset	Reset Input	Double-click to	[clk]			
		s1	Avalon Memory Mapped ...	Double-click to	[clk]			
		external_conn...	Conduit	Double-click to	keys_external...			
		irq	Interrupt Sender	Double-click to	[clk]			

#### 4. QSys KEYS PIO Connections

Next, after this is set, we generate the Verilog system file from this. We must also alter the top-level design in order to instantiate the module. This Top-level instantiation can be seen in appendix A. After compiling this, we must then set the pins for the KEYS. In order to do this, we go into the pin planner in Quartus Prime, and set the pins to the addresses seen in the figure below.

Groups
Report

Tasks

- ✓ Early Pin Planning
  - Early Pin Planning
  - Run I/O Assignment
  - Export Pin Assignment
  - Pin Finder...
- ✓ Highlight Pins
  - I/O Banks

Named: \*

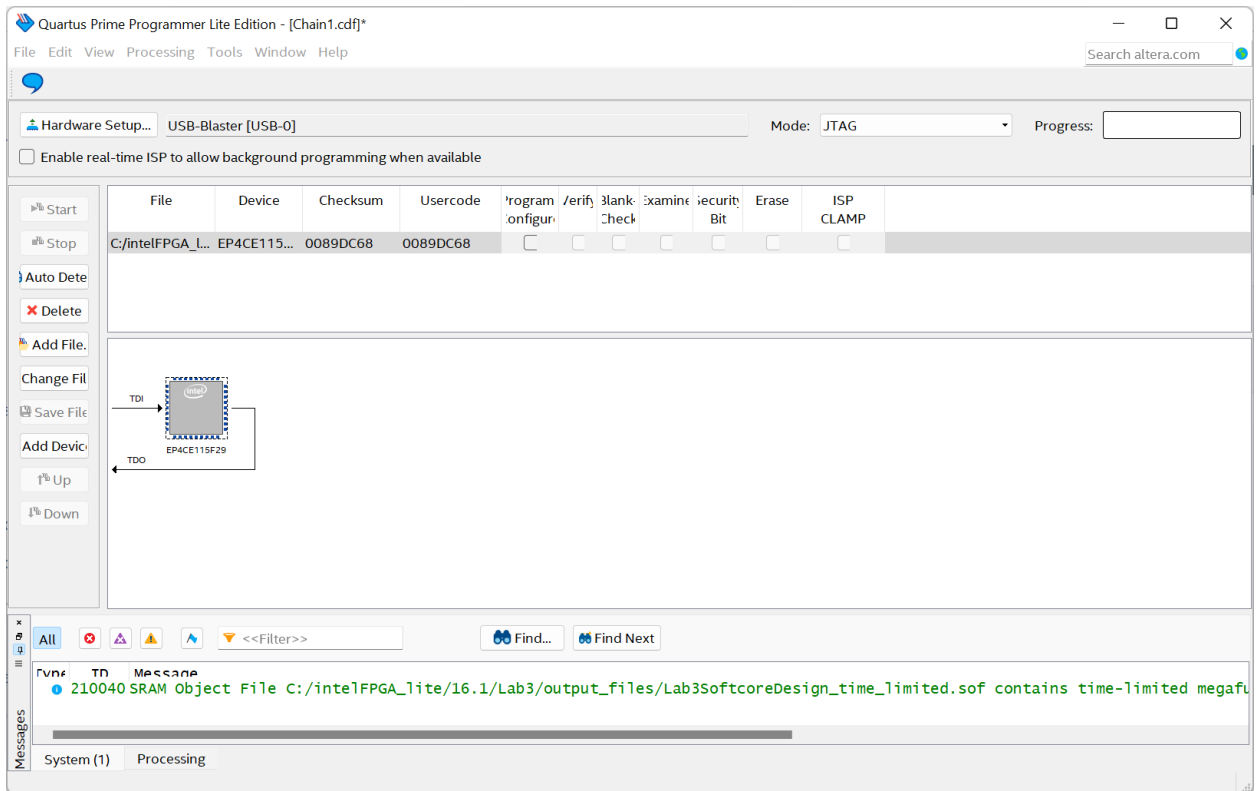
Edit: ✕

✓

Node Name	Direction	Location	I/O Bank	/REF Group	Port Location	I/O Standard
KEY[3]	Input	PIN_R24	5	B5_N0	PIN_R24	2.5 V
KEY[2]	Input	PIN_N21	6	B6_N2	PIN_N21	2.5 V
KEY[1]	Input	PIN_M21	6	B6_N1	PIN_M21	2.5 V
KEY[0]	Input	PIN_M23	6	B6_N2	PIN_M23	2.5 V

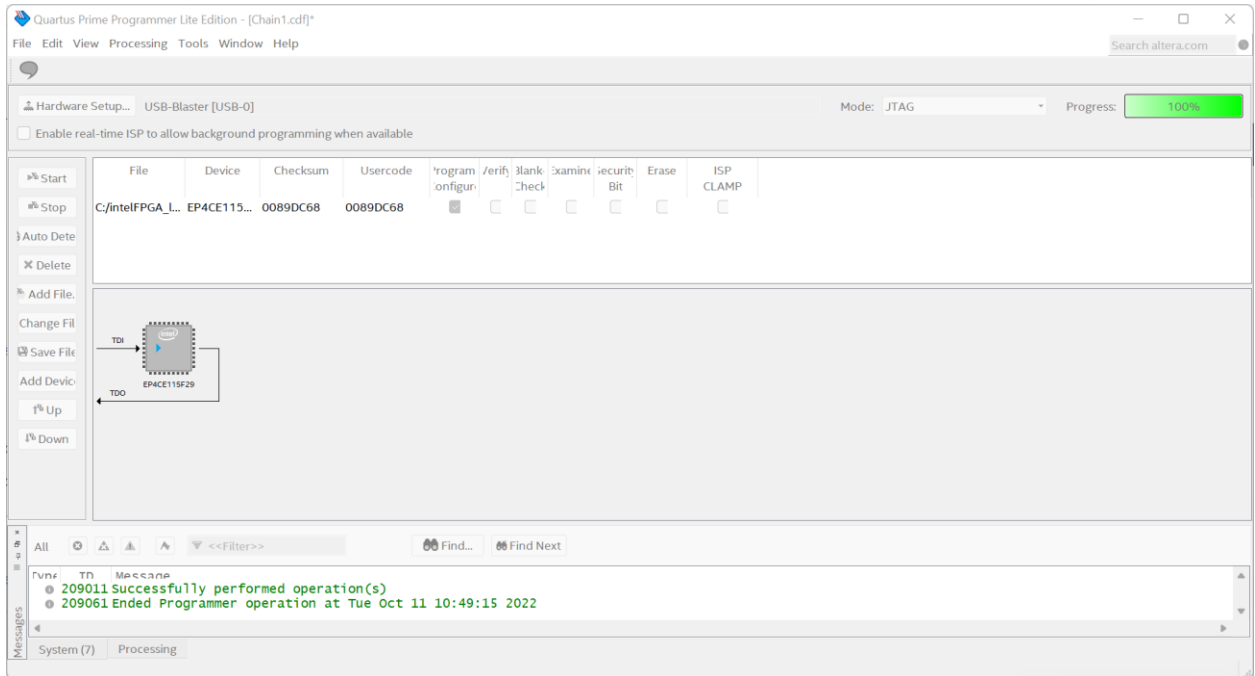
## 5. Pin Planner for KEYS

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. Next we download the system onto the board, the process can be shown in the following figures, starting at adding the file to the Quartus Prime Programmer.



## 6. Adding the File to be Downloaded onto the Board

After adding the sof file, it is 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.



## 7. 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.

Nios II Application and BSP from Template

### Nios II Software Examples

Please specify a .sopcinfo file

**Target hardware information**

SOPC Information File name:  ...

CPU name:

**Application project**

Project name:

☒ Use default location

Project location:  ...

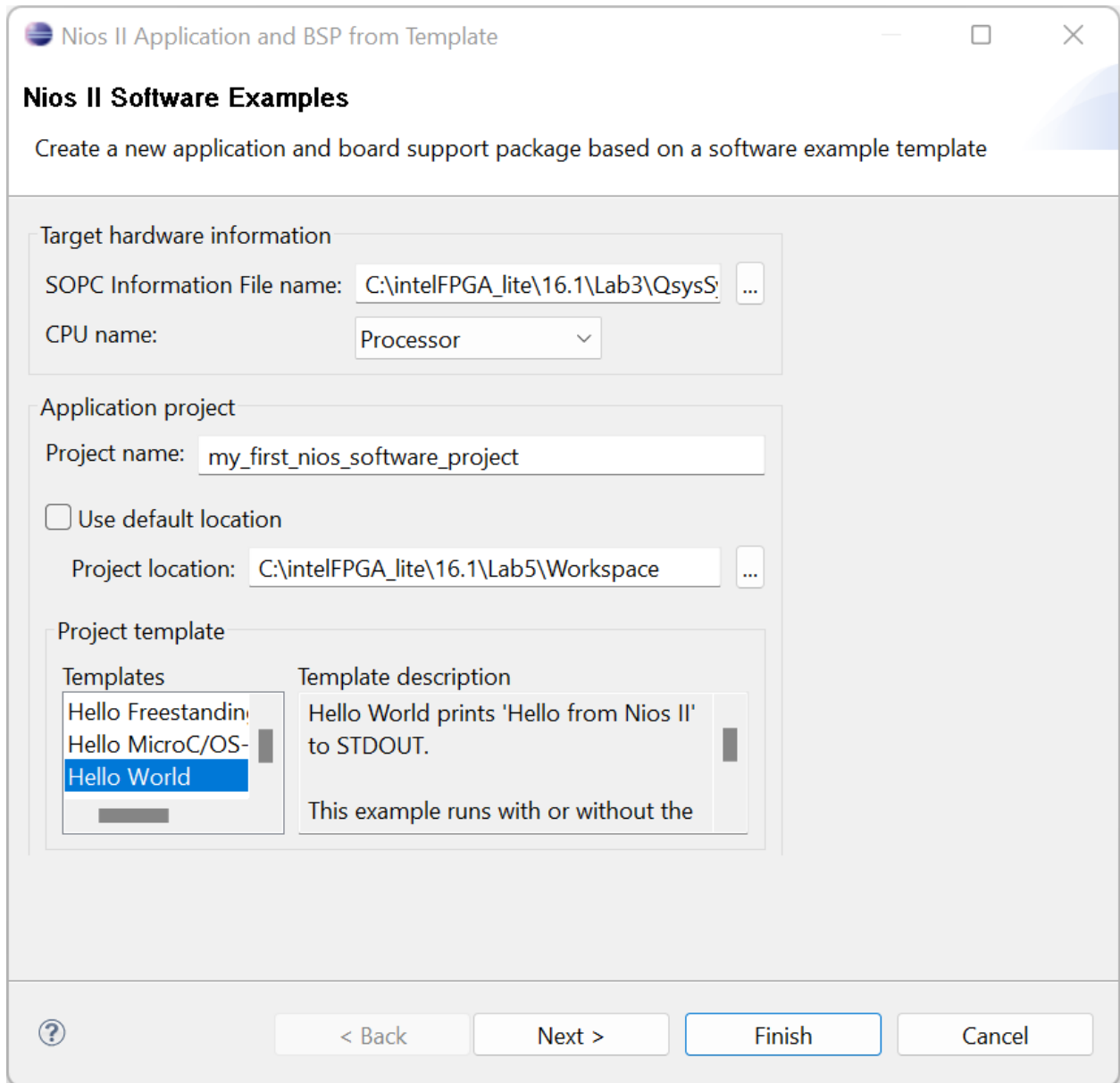
**Project template**

Templates	Template description
Hello Freestanding	
Hello MicroC/OS-	
<b>Hello World</b>	Hello World prints 'Hello from Nios II' to STDOUT.  This example runs with or without the

? < Back Next > Finish Cancel

## 8. 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. This will be our base Project template that we will edit to accomplish our design goals. We can see this in the figure below.



## 9. NIOS II Application and BSP From Template Settings

After creating the NIOS II application and BSP, we are then able to implement our project design specifications. The first part of this project's code can be seen in Appendix B. For this design we have a few functions. The first two of these is the incr and decr function. Seen in the figure below. These functions read the LED value and will either increment or decrement until the counter reaches 9 or 1 and write the output to the Green LEDs.

```

void incr() {
    int LEDVal = 0;
    LEDVal = IORD(GREEN_LEDS_BASE, 0);

    for (int i = LEDVal; i <= 9; i++) {
        IOWR(GREEN_LEDS_BASE, 0, i);
        delayfn();
    }
    return;
}

void decr() {
    int LEDVal = 0;
    LEDVal = IORD(GREEN_LEDS_BASE, 0);

    for (int i = LEDVal; i >= 1; i--) {
        IOWR(GREEN_LEDS_BASE, 0, i);
        delayfn();
    }
    return;
}

```

#### 10. Incr and Decr functions

These functions also call the delayfn function in order to leave some time between each count. The delay function is simply a while loop. This can be seen in the figure below.

```

void delayfn() {
    int delay = 0;
    while(delay < 2000000)
    {
        delay++;
    }
}

```

#### 11. Delay Function

The next functions are the Key3 and Key2 ISRs. The Key3 function loads the value of Sw0 into Sw0 integer value. This is then checked and if it is 1 it will call the incr function and if it is 0 it will call the decr function. Key 2 simply writes 1 to the LEDs and writes to the edge capture register to reset it, as well as reads the PIO to delay the ISR exit. These two functions can be seen below.

```
void key3_isr() {
    // Program
    int Sw0 = 0;
    Sw0 = IORD(SWITCHES_BASE, 0);

    if (Sw0 == 0) {
        decr();
    }
    else if (Sw0 == 1) {
        incr();
    }

    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
    IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    return;
}

void key2_isr() {
    // Program
    IOWR(GREEN_LEDS_BASE, 0, 0x1);

    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
    IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    return;
}
```

## 12. Key2 and Key3 ISR Functions

The next function in our program is handle\_key\_interrupts. This function is passed with context. Which is used in the function that is called when the interrupt is triggered. So, this function's main purpose is to check the edge capture ptr, comparing this with the value of 8 first, which will call Key3\_isr if true, and Key2\_isr if true when compared with 4. This function can be seen below.



```

void handle_key_interrupts(void* context){
    volatile int *edge_capture_ptr = (volatile int*) context;
    *edge_capture_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    if (*edge_capture_ptr & 0x8){
        key3_isr();
    }
    else if (*edge_capture_ptr & 0x4){
        key2_isr();
    }
    return;
}

```

### 13. Handle\_key\_interrupts Function

The last two functions are the pio\_init and the main function. the pio\_init function resets the edge capture to match the altera irq register, enables the keys as the interrupts, resets the edge capture register, and registers the ISR. Lastly, the main function calls the initialization, prints the instructions for the user, writes a random number to the LEDs, and sticks in an infinite loop.

```

void pio_init(){
    void* edge_capture_ptr = KEYS_EDGE_TYPE;
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(KEYS_BASE, 0xC);
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0x0);
    alt_irq_register(KEYS_IRQ, edge_capture_ptr, handle_key_interrupts);

    return;
}

int main(){
    pio_init();
    printf("Key3: \n      Sw0 High: Increment to 9\n      Sw0 Low: Decrement to 1\nKey2: Se
    IOWR(GREEN_LEDS_BASE, 0, 0b00001010);
    while(1){

    }
    return 0;
}

```

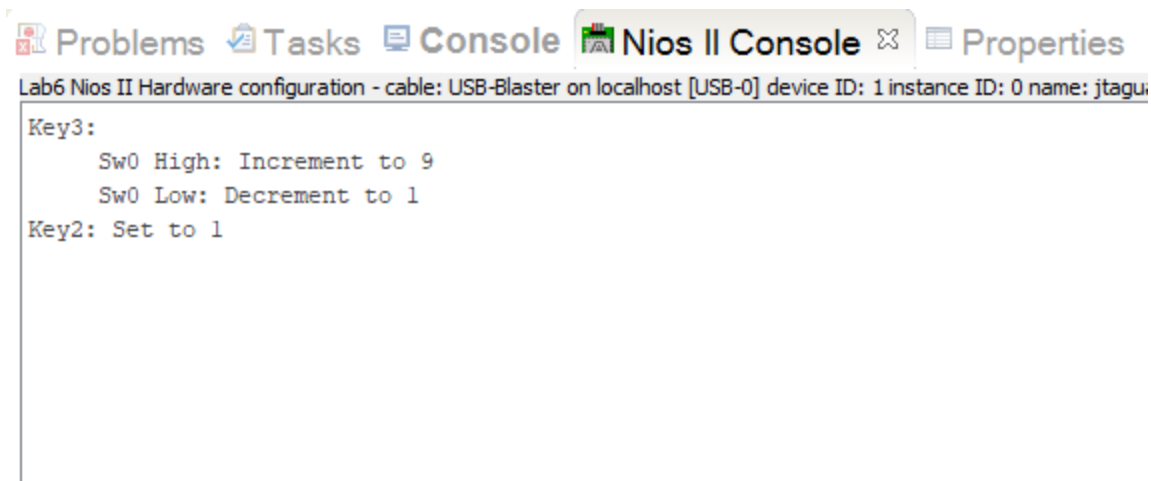
### 14. Pio\_init and Main Functions

With this part completed, we can move into Part 2. The code for this part can be seen in Appendix C. Here, we add a global variable IncDec. This sets allows for us to increment and decrement in the main loop and not within the ISRs. So, the ISRs simply set the IncDec Value and resets the edge capture. The handle key interrupts function does the same thing as in the previous part except now with key 1 being a part as well. The pio\_init also does the same as in the previous part. We have a few new functions display, which displays the value passed into in the HexDisplays as well as masking the other bits so that the seven segment displays are off. As well as the DecodeHex, which passes a value which will first be in the form of the what is the 7-

Segment Display form of the value. It then finds the integer value of those numbers. This is used in order to keep the value currently displayed when switching from incrementing to decrementing. Lastly, the main value will do the same as the last, except in the while loop, checks IncDec integer and will increment or decrement accordingly.

## Data Analysis

Upon beginning the first part of the code, the following is displayed in the NIOSII Console.

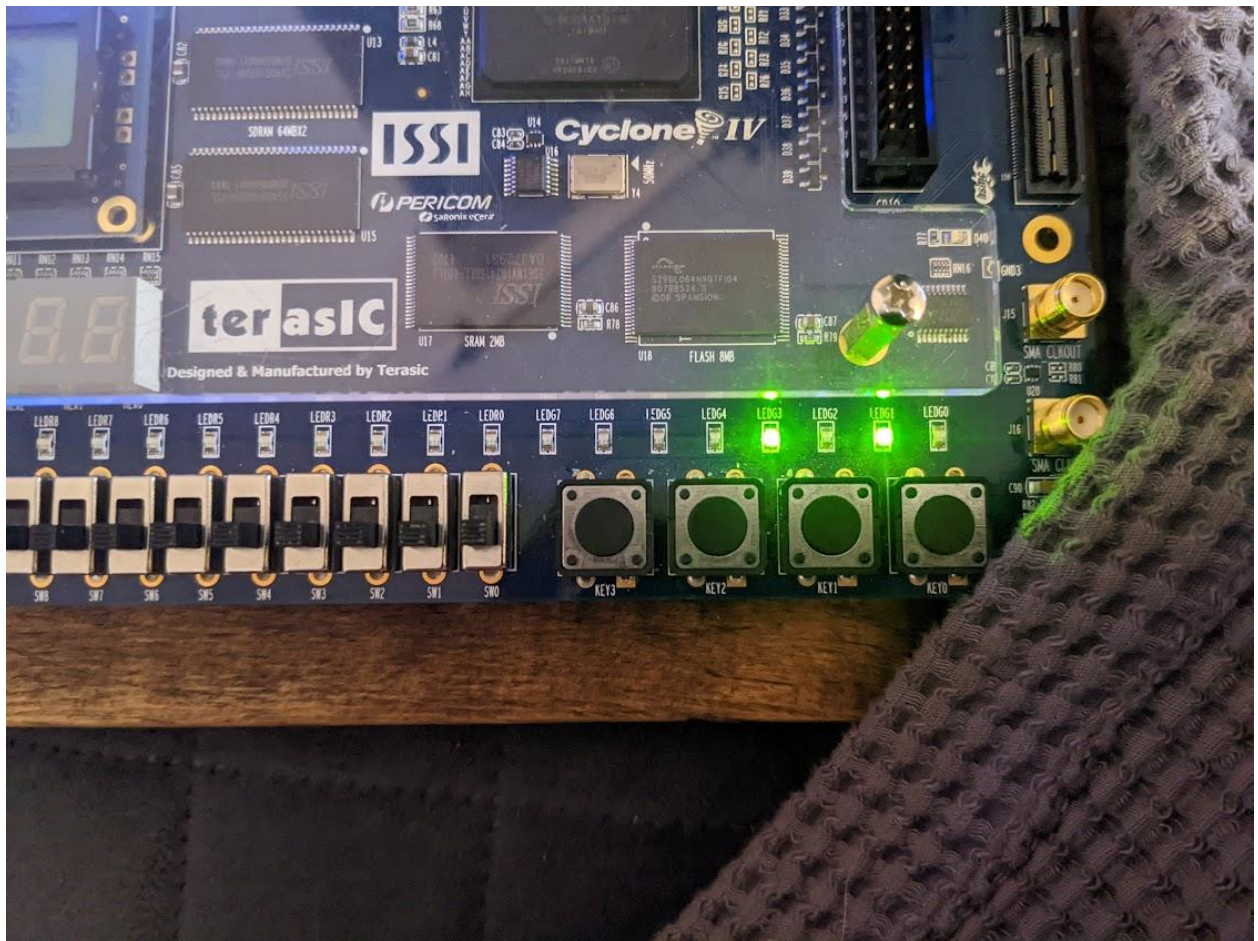


The screenshot shows the Nios II Console window with the following text:

```
Lab6 Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtagui
Key3:
  Sw0 High: Increment to 9
  Sw0 Low: Decrement to 1
Key2: Set to 1
```

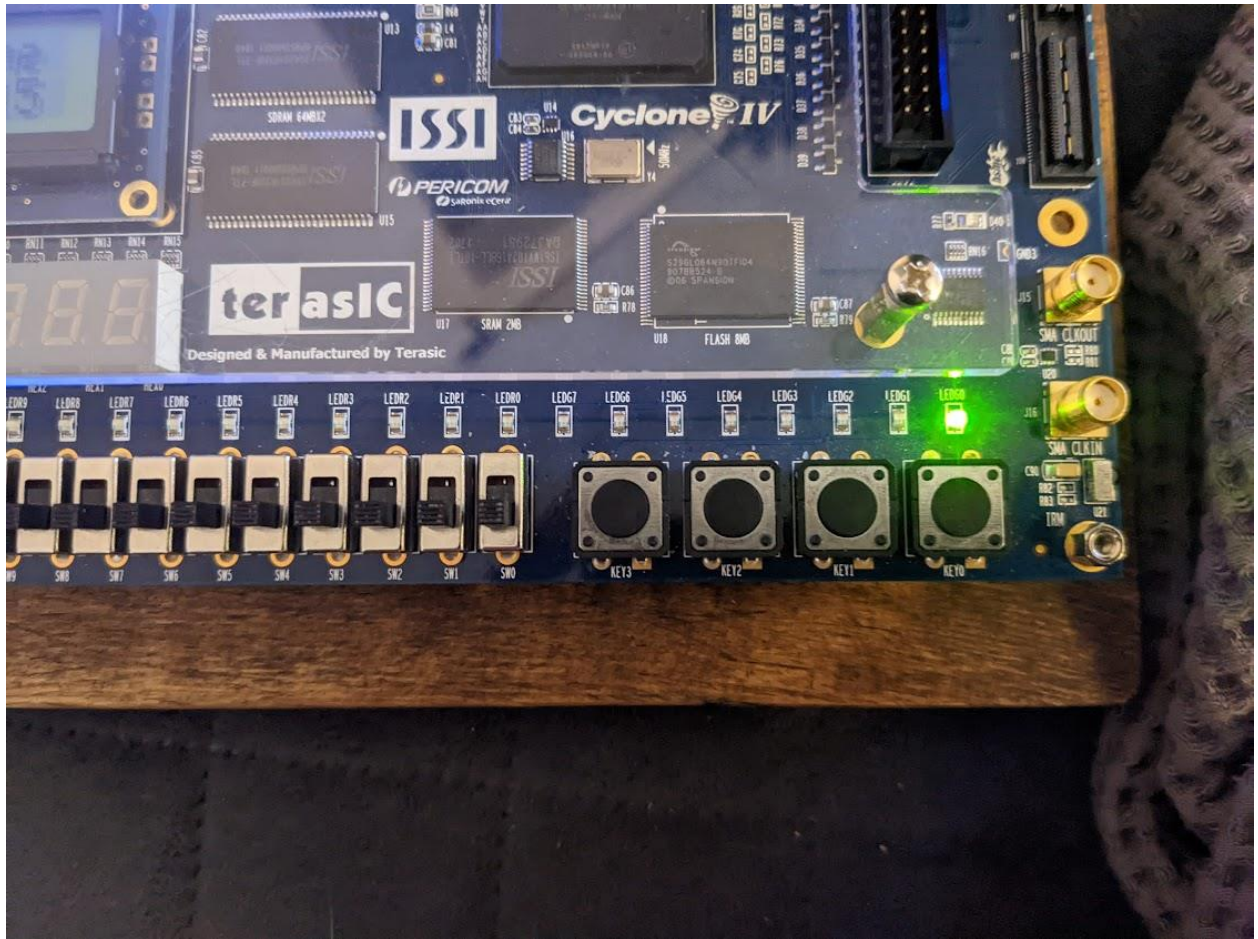
### 15. NiosII Console for Part 1

After starting the following pattern is shown on the LEDs this is hard coded in the main program.



16. Initial Green LED value

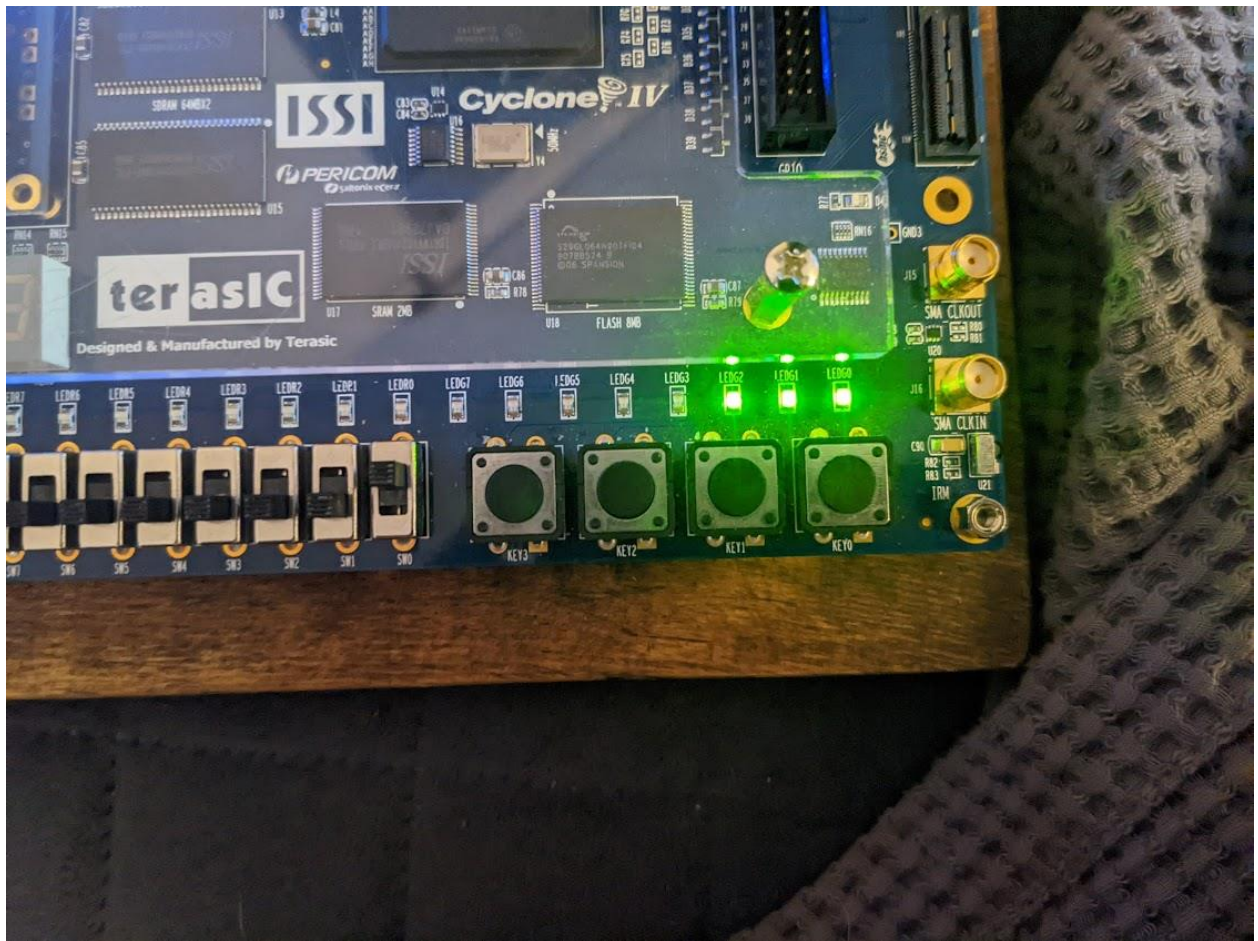
Next, after hitting Key2, it sets the LED to the value of 1 immediately. This can be seen in the figure below.



### 17. Set Value to 1 Using Key2

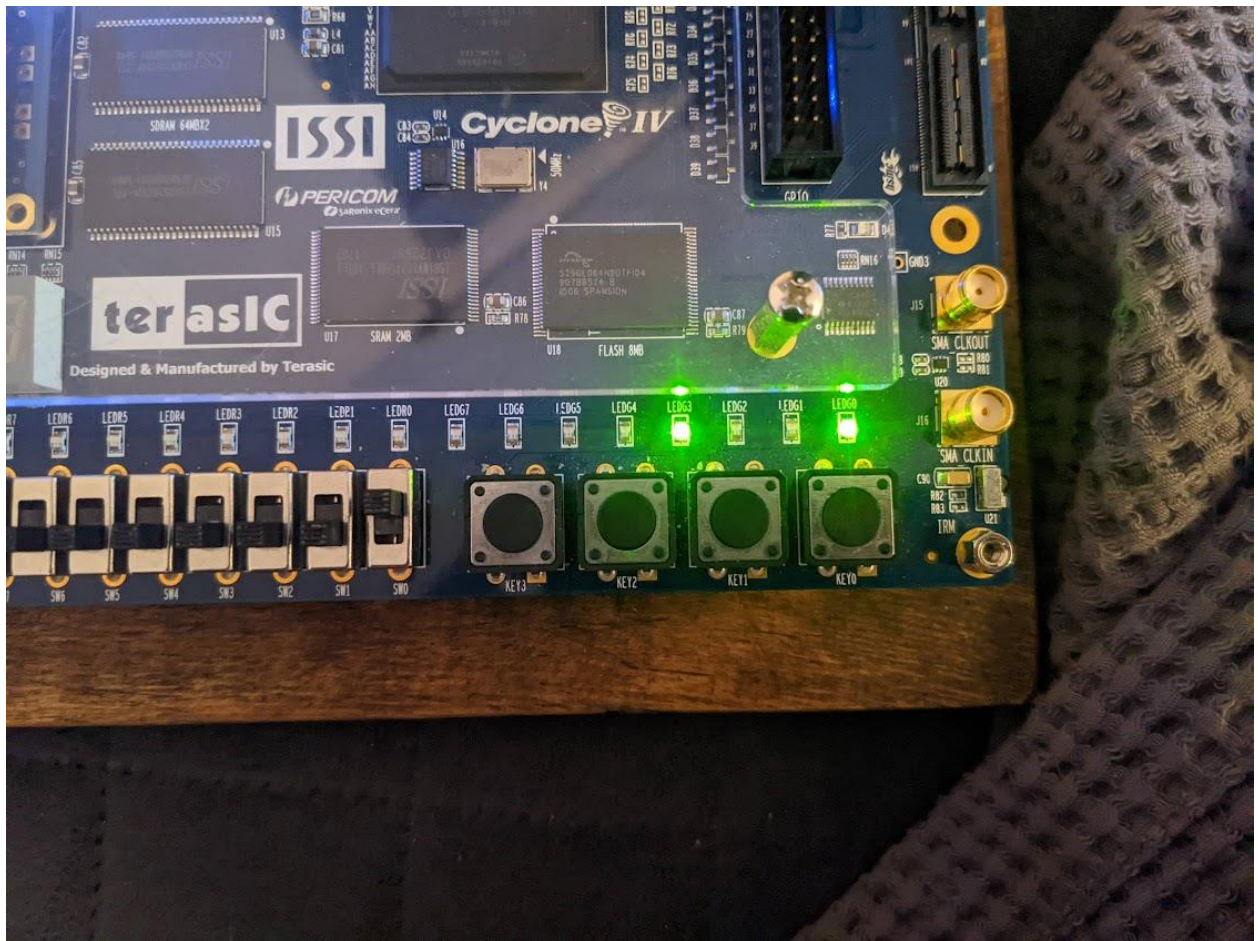
Following this, we can hit Key3 with Sw0 high, and see that it starts to count, In the following figure it is mid count.





18. Count up using Key3 Sw0 High (Mid Count)

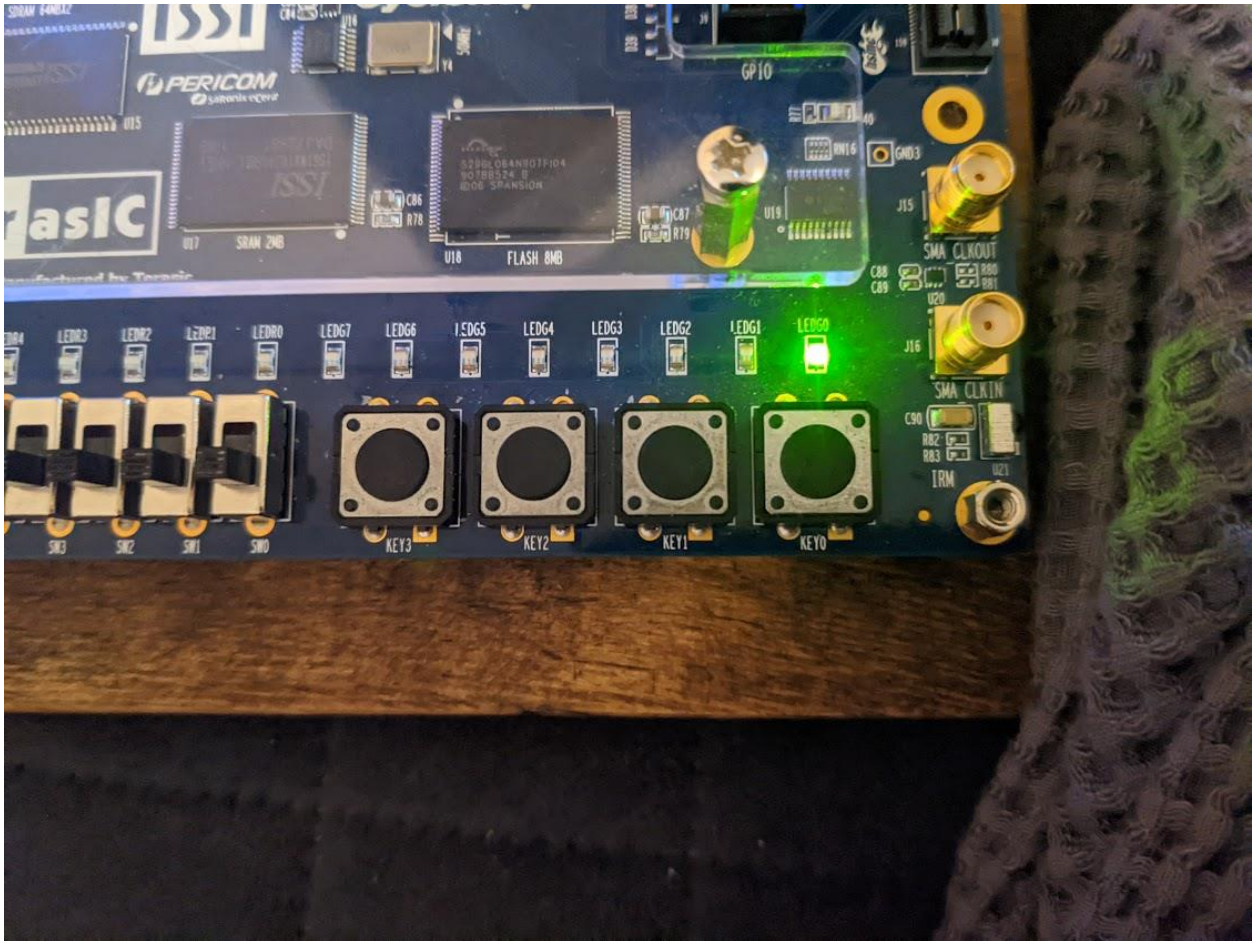
It finishes its count at the value of 9 which is shown in the figure below.



19. Finished Count up using Key3 Sw0 High

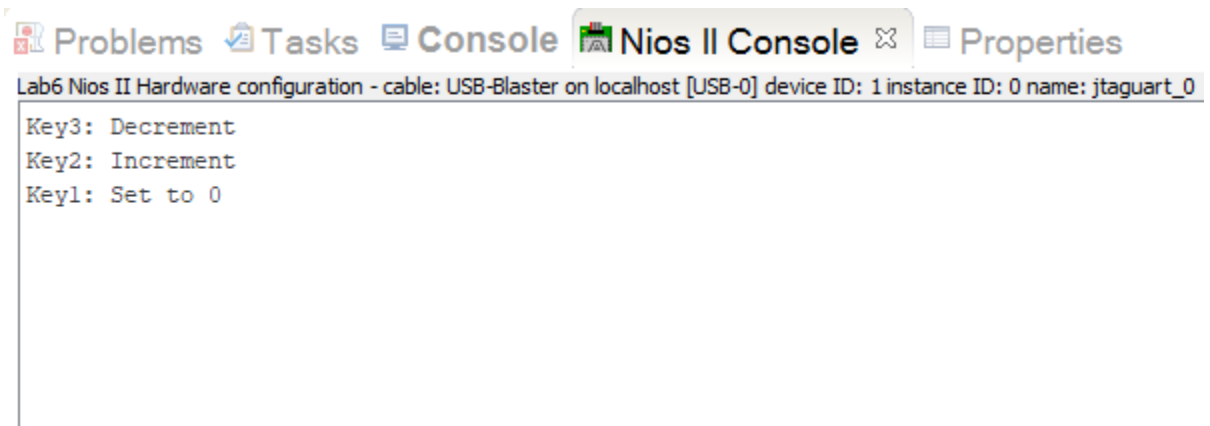
Then we can press Key3 with Sw0 low and it will count down to 1 again seen in the figure below.





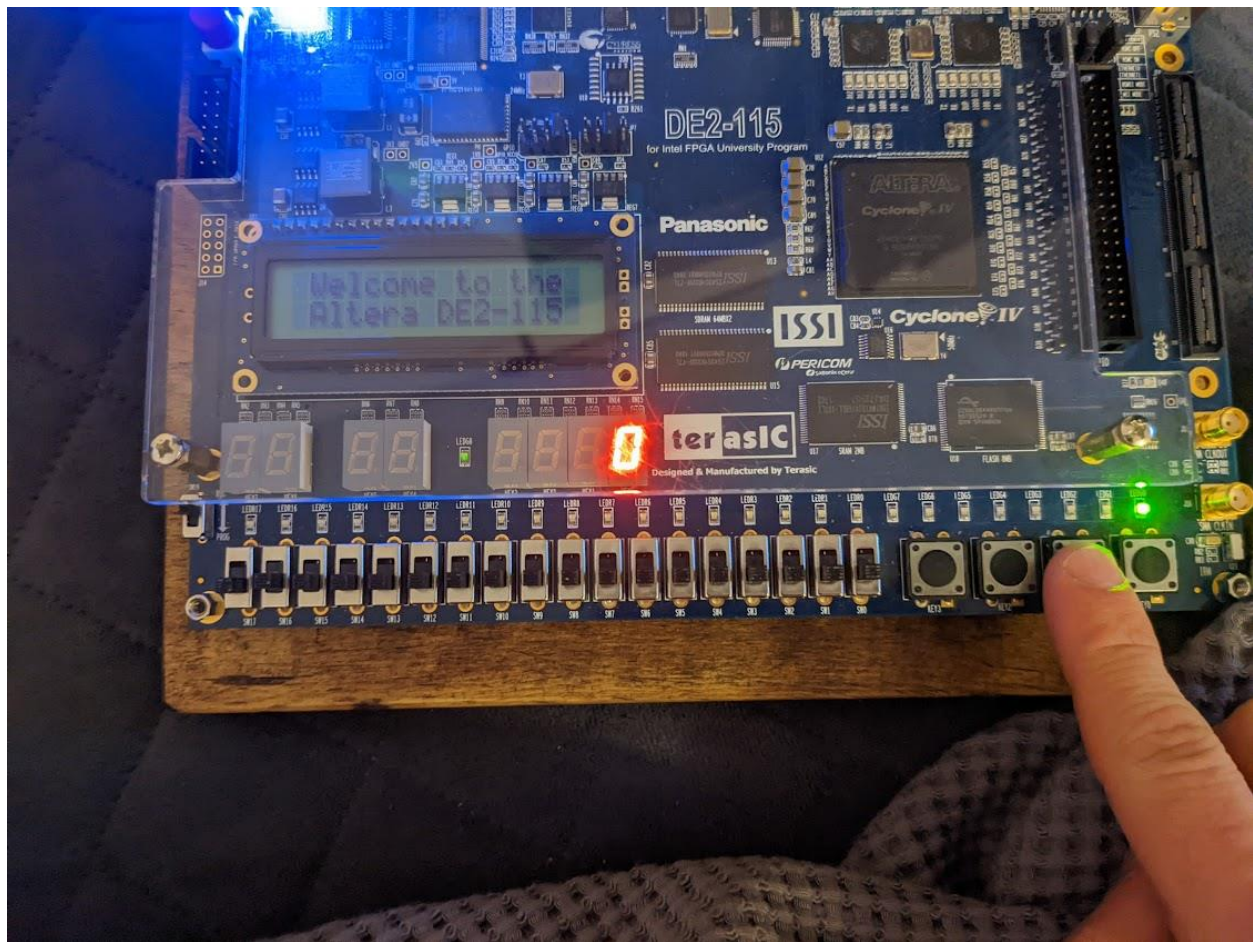
20. Finished Counting Down to 1

Next for Part 2, the following dialog is first displayed in the Nios II Console.



21. Nios II Output for Part 2

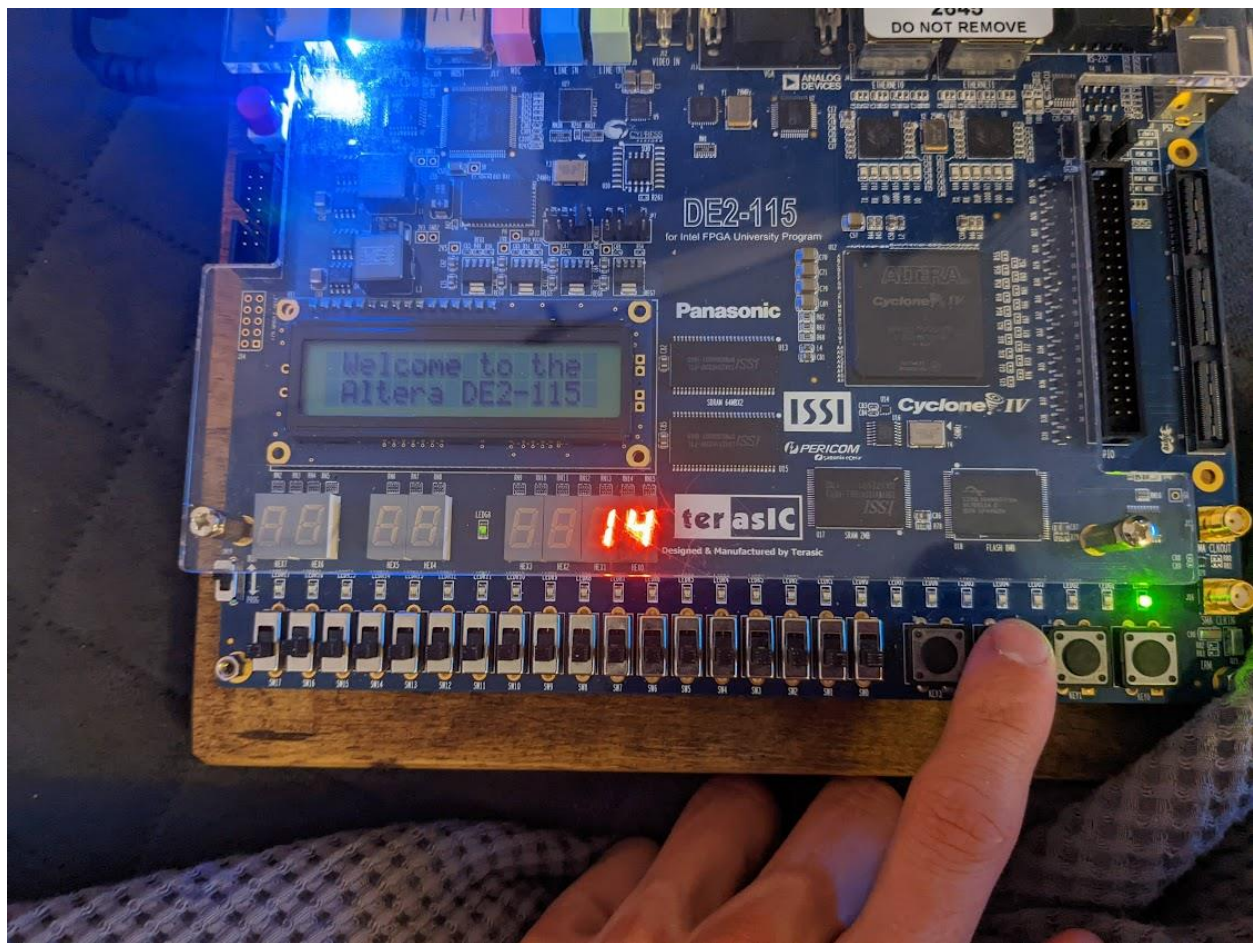
After this, we set the value to zero using the Key 1. This can be seen in the figure below.



22. Set the value on the Seven Segment to 0

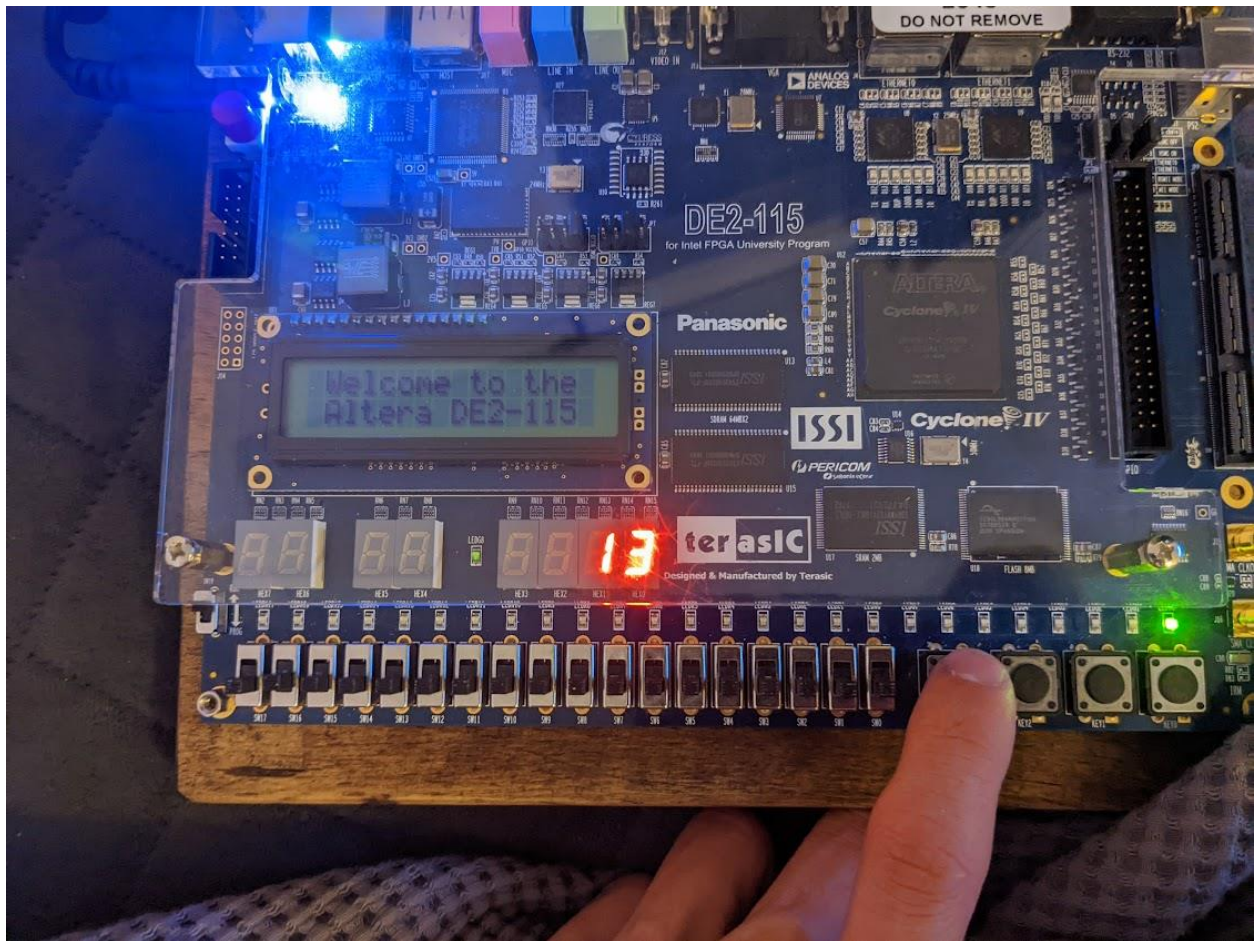
Next, we can use key 2 to begin incrementing. This will start counting until reaching 99 which is the max value when just using the two displays.





### 23. Counting up

At any point, we can set the value back to zero, which will stop incrementing or change to decrementing. The switch to decrementing can be seen below.



#### 24. Switch from Incrementing to Decrementing

With all of this, we can see that the code accomplished its goals and works as intended. The project was a success.

## **Conclusion**

This lab accomplished its goals to further the understanding of implementing Interrupts in a C++ runtime environment. This Lab was very beneficial for going forward, as interrupts are a vital part of embedded systems engineering, and understanding their full potential is coming to fruition. This lab was essential in accomplishing this goal and did so in an interactive and intuitive way.

## Appendix A (Top Level Instantiation)

```
module Lab3SoftcoreDesign (CLOCK_50, SW, KEY, LEDR, LEDG,  
    DRAM_CLK, SevMSB, SevLSB, sdram_wire_addr, sdram_wire_ba,  
    sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n,  
    sdram_wire_dq, sdram_wire_dqm, sdram_wire_ras_n,  
    sdram_wire_we_n);  
    input CLOCK_50;  
    input [17:0] SW;  
    input [3:0] KEY;  
    output [7:0] LEDG;  
    output [17:0] LEDR;  
    output [31:0] SevMSB;  
    output [31:0] SevLSB;  
    output [12:0] sdram_wire_addr;  
    output [1:0] sdram_wire_ba;  
    output sdram_wire_cas_n;  
    output sdram_wire_cke;  
    output sdram_wire_cs_n;  
    inout [31:0] sdram_wire_dq;  
    output DRAM_CLK;  
    output [3:0] sdram_wire_dqm;  
    output sdram_wire_ras_n;  
    output sdram_wire_we_n;
```

```
QsysSystem Softcore (  
    .clk_clk(CLOCK_50),  
    .reset_reset(KEY),  
    .green_leds_external_connection_export(LEDG),  
    .red_leds_external_connection_export(LEDR),  
    .switches_external_connection_export(SW),  
    .sevseg4msb_external_connection_export(SevMSB),  
    .sevsegment_4lsb_external_connection_export(SevLSB),  
    .keys_external_connection_export(KEY),  
    .sdram_wire_addr(sdram_wire_addr),  
    .sdram_wire_ba(sdram_wire_ba),  
    .sdram_wire_cas_n(sdram_wire_cas_n),  
    .sdram_wire_cke(sdram_wire_cke),  
    .sdram_wire_cs_n(sdram_wire_cs_n),  
    .sdram_wire_dq(sdram_wire_dq),  
    .sdram_wire_dqm(sdram_wire_dqm),  
    .sdram_wire_ras_n(sdram_wire_ras_n),  
    .sdram_wire_we_n(sdram_wire_we_n),
```

```
.sdram_clk_clk(DRAM_CLK));  
endmodule
```

## Appendix B (Part 1 Code)

```
// Program for Generating an Interrupt

#include <stdio.h>
#include <system.h>
#include <alt_types.h>
#include "altera_avalon_pio_regs.h"
#include <stdio.h>

void incr(){
    int LEDVal = 0;
    LEDVal = IORD(GREEN_LEDS_BASE, 0);

    for (int i = LEDVal; i <= 9; i++){
        IOWR(GREEN_LEDS_BASE, 0, i);
        delayfn();
    }
    return;
}

void decr(){
    int LEDVal = 0;
    LEDVal = IORD(GREEN_LEDS_BASE, 0);

    for (int i = LEDVal; i >= 1; i--){
        IOWR(GREEN_LEDS_BASE, 0, i);
        delayfn();
    }
    return;
}

void delayfn(){
    int delay = 0;
    while(delay < 2000000)
    {
        delay++;
    }
}

void key3_isr(){
    // Program
    int Sw0 = 0;
```

```

        Sw0 = IORD(SWITCHES_BASE, 0);

        if (Sw0 == 0){
            decr();
        }
        else if (Sw0 == 1){
            incr();
        }

        IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
        IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
        return;
    }

    void key2_isr() {
        // Program
        IOWR(GREEN_LEDS_BASE, 0, 0x1);

        IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
        IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
        return;
    }

    void handle_key_interrupts(void* context){
        volatile int *edge_capture_ptr = (volatile int*) context;
        *edge_capture_ptr =
        IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
        if (*edge_capture_ptr & 0x8){
            key3_isr();
        }
        else if (*edge_capture_ptr & 0x4){
            key2_isr();
        }
        return;
    }

    void pio_init() {
        void* edge_capture_ptr = KEYS_EDGE_TYPE;
        IOWR_ALTERA_AVALON_PIO_IRQ_MASK(KEYS_BASE, 0xC);
        IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0x0);
        alt_irq_register(KEYS_IRQ, edge_capture_ptr,
        handle_key_interrupts);

        return;
    }

    int main() {

```

```
pio_init();
printf("Key3: \n      Sw0 High: Increment to 9\n      Sw0
Low: Decrement to 1\nKey2: Set to 1\n");
IOWR(GREEN_LEDS_BASE, 0, 0b00001010);
while(1){

}
return 0;
}
```



## Appendix C (Part 2 Code)

```
#include <stdio.h>
#include <system.h>
#include <alt_types.h>
#include "altera_avalon_pio_regs.h"
#include <stdio.h>

int IncDec = 0;

void delayfn(){
    int delay = 0;
    while(delay < 2000000)
    {
        delay++;
    }
}

void key3_isr(){
    // Decrement the Num to 0
    IncDec = 2; //Turns IncDec to 2 in order to run Decrement
in main

    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
    IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    return;
}

void key2_isr(){
    // Increment to 99
    IncDec = 1; //Turns IncDec to 1 in order to run in main

    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
    IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    return;
}

void key1_isr(){
    // Set to 0
    IOWR(SEVSEGMENT_4LSB_BASE, 0, 0xFFFFFFFF40); //Sets to 0
    IncDec = 0; //Stops inc or dec
```

```

    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
    IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    return;
}

void handle_key_interrupts(void* context){
    volatile int *edge_capture_ptr = (volatile int*) context;
    *edge_capture_ptr =
    IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    if (*edge_capture_ptr & 0x8){
        key3_isr();
    }
    else if (*edge_capture_ptr & 0x4){
        key2_isr();
    }
    else if (*edge_capture_ptr & 0x2){
        key1_isr();
    }
    return;
}

void pio_init(){
    void* edge_capture_ptr = KEYS_EDGE_TYPE;
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(KEYS_BASE, 0xE);
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0x0);
    alt_irq_register(KEYS_IRQ, edge_capture_ptr,
    handle_key_interrupts);

    return;
}

void display(int Value){
    int SEVENSEGLUT[10] = {0b01000000, 0b01111001, 0b00100100,
    0b00110000, 0b00011001, 0b00010010, 0b00000010, 0b01111000,
    0b10000000, 0b00010000};
    int R;
    int toDisplay = 0;
    int shiftVal = 0;
    int FourthSeg = 0b11111111;
    int toDisplayMask = 0;

    while (Value != 0){ //Displays the value on the Hex Masks
are for 0 values to not be displayed unless the LSB
        R = Value % 10;
        toDisplay = toDisplay + (SEVENSEGLUT[R] << shiftVal);
        Value = (Value - R) / 10;
        shiftVal = shiftVal + 8;
    }
}

```

```

    }
    toDisplay = toDisplay | 0xFFFF0000;
    toDisplayMask = toDisplay & 0x0000FF00;

    if (toDisplayMask == 0){
        toDisplay = toDisplay | 0xFFFFFFFF00;
    }
    toDisplayMask = toDisplay & 0x000000FF;
    if (toDisplayMask == 0){
        toDisplay = 0xFFFFFFFF40;
    }

    IOWR(SEVSEGMENT_4LSB_BASE, 0, toDisplay);
    return;
}

int DecodeHex(int Val){ //Decodes the HEX from the Seven Seg
into an Integer, Used as Starting Point for Inc and Dec
    int SEVENSEGLUT[10] = {0b01000000, 0b01111001, 0b00100100,
0b00110000, 0b00011001, 0b00010010, 0b00000010, 0b01111000,
0b00000000, 0b00010000};
    int i;
    int MASK = 0b01111111;
    int Output = 0;
    int Lookup = Val & MASK;
    int ShiftAmt = 0;
    int Count = 1;

    while (Lookup != 0b1111111){
        for (i = 0; Lookup != SEVENSEGLUT[i]; i++){
            Output = Output + (i*Count);
            Count = Count * 10;
            ShiftAmt = ShiftAmt + 8;
            Lookup = Val & (MASK << ShiftAmt);
            Lookup = Lookup >> ShiftAmt;
        }
        return Output;
    }

}

void incr(){ //Increments from where the Hex Display is to 99
    int HexVal = 0;
    HexVal = IORD(SEVSEGMENT_4LSB_BASE, 0);
    HexVal = DecodeHex(HexVal);
    for (int i = HexVal; i <= 99 && IncDec == 1; i++){
        display(i);
        delayfn();
    }
}

```

```

        return;
    }

    void decr(){ //Decrements from Current val of hex display to 0
        int HexVal = 0;
        HexVal = IORD(SEVSEGMENT_4LSB_BASE, 0);
        HexVal = DecodeHex(HexVal);
        for (int i = HexVal; i >= 0 && IncDec == 2; i--){
            display(i);
            delayfn();
        }
        return;
    }

    int main(){ //Initializes the 7 Segment
        pio_init();
        int outSevenSeg = 0xFFFFFFFF;
        IOWR(SEVSEG4MSB_BASE, 0, outSevenSeg);
        IOWR(SEVSEGMENT_4LSB_BASE, 0, outSevenSeg);

        printf("Key3: Decrement\nKey2: Increment\nKey1: Set to 0");
        while(1){ //Loops through Calling Incr function if IncDec
changes to 1 and Decr if to 0. Doing this here allows for these
fns to be interrupted.
            if (IncDec == 1) incr();
            else if (IncDec == 2) decr();
        }

        return 0;
    }

```