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

**TECHNICAL REPORT**

Assignment: Number 7
Experiment Title:
Course Title: ECE 178 (Embedded Systems)
Instructor: Dr. Reza Raeisi

Prepared by: Anthony Herrick
Student ID #300144976
Date Submitted: 11/06/2022

**INSTRUCTOR SECTION**

Comments: ______________________________________________

______________________________________________

______________________________________________

______________________________________________

______________________________________________

______________________________________________

______________________________________________

Final Grade: ____________________

TABLE OF CONTENTS

LIST OF FIGURES

## 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++ enviroment Eclipse Software Build tools.

## Hardware Requirements

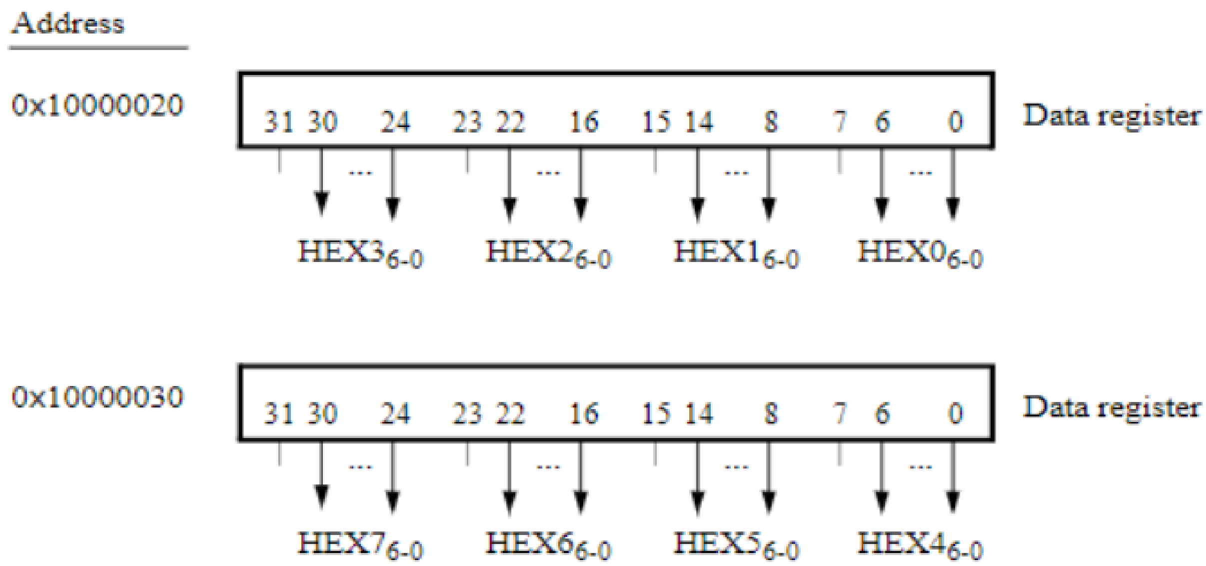
- Computer with Intel FPGA Monitor program 16.1
- Computer with Quartus Prime 16.1.
- DE2-115 Board
- A-B USB Cable

## Software Requirements

- Quartus Prime with Qsys, version 16.1.

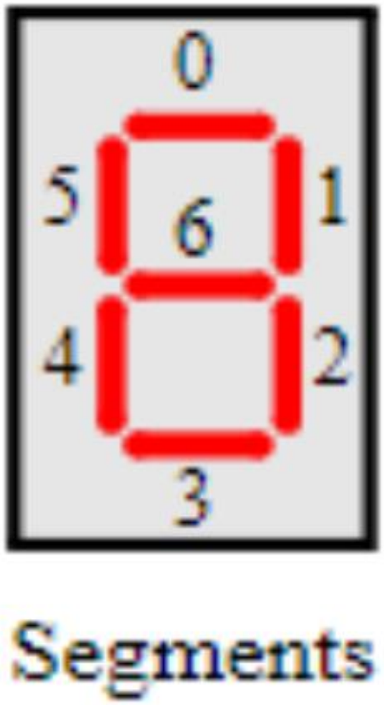
## Background

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.



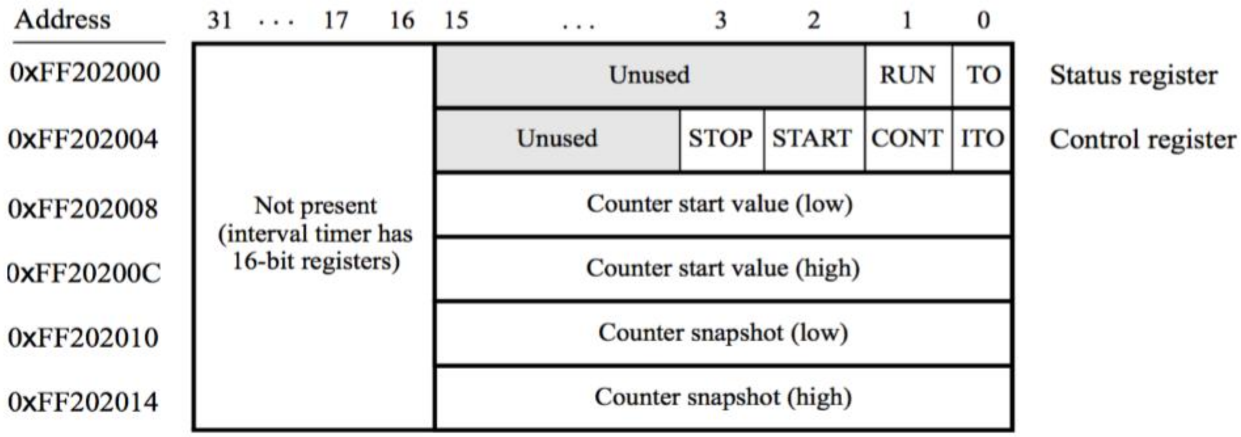

1. 7-Segment Display Address Spaces

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.




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.




3. Interval Timer Registers

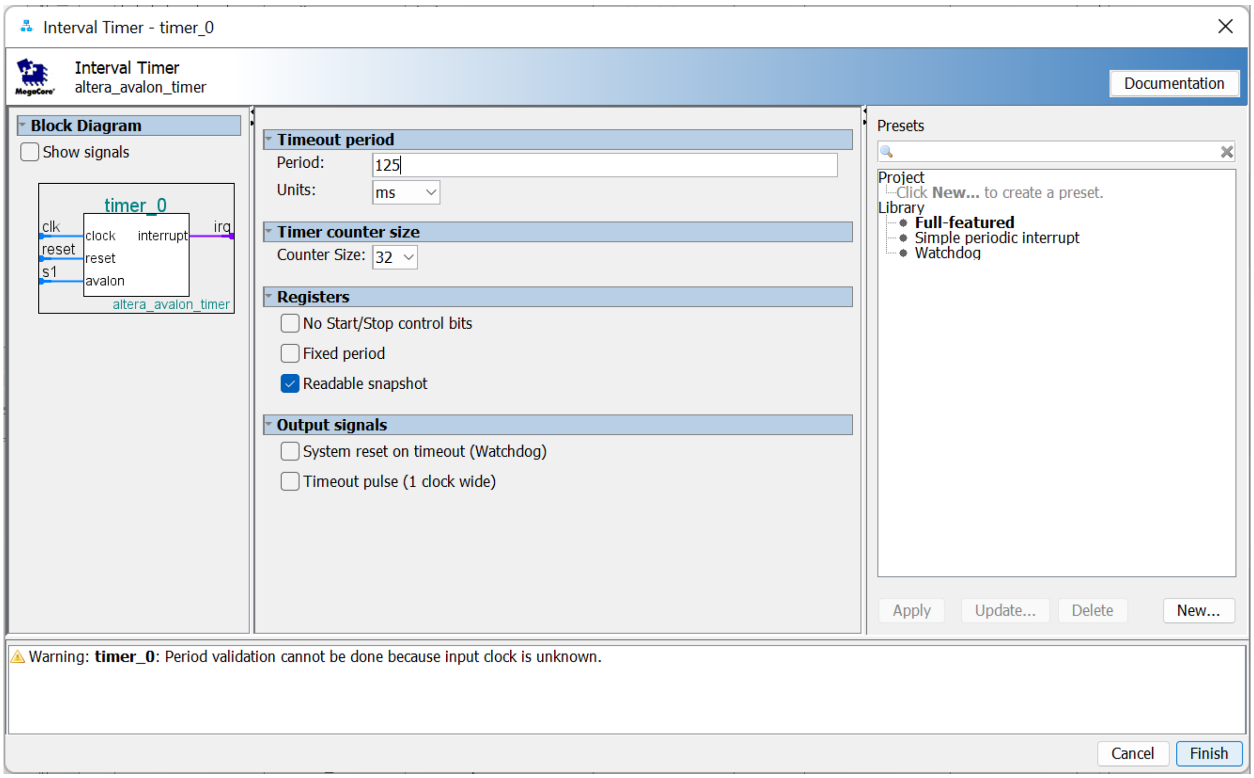
For an interval timer, the registers can be seen above. These have 16-bit register. The counter start value low and high can be altered to change the duration of the timer this is periodlo and periodhi. The control register is then used to begin the timer. This timer can be used to accurately measure time on an FPGA board.

## 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 an interval timer. There is then three parts to this lab, one which counts in odd numbers one to fifteen and then decrements. The next part will create a walking ones which speed can be altered using the key interrupts. The last part will implement a tracker for wether or not the LED was stopped on an active switch. If it was, it will increment the players score by one.

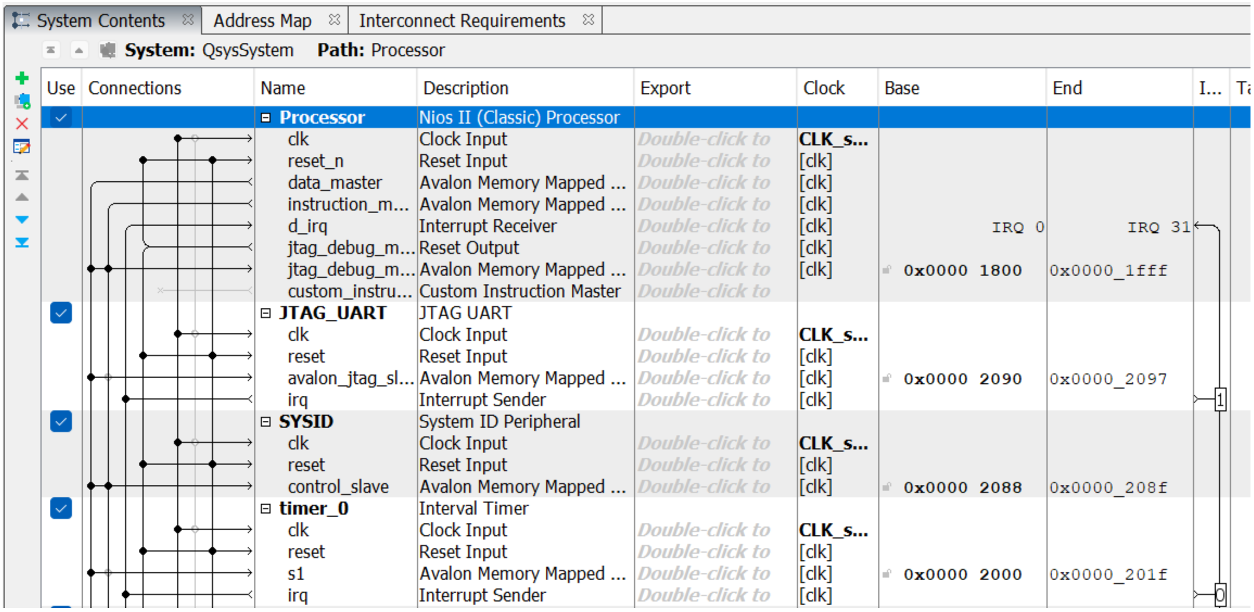
## Project Procedure

To begin this lab, we first open the QSys, and edit a few things from the softcore embedded system design from Lab 6. We add the Interval timer. For our purposes, our smallest time in between the timer interrupt is 125ms so we set the timeout period to 125ms. This will be the resolution of our timer. The settings can be seen below.



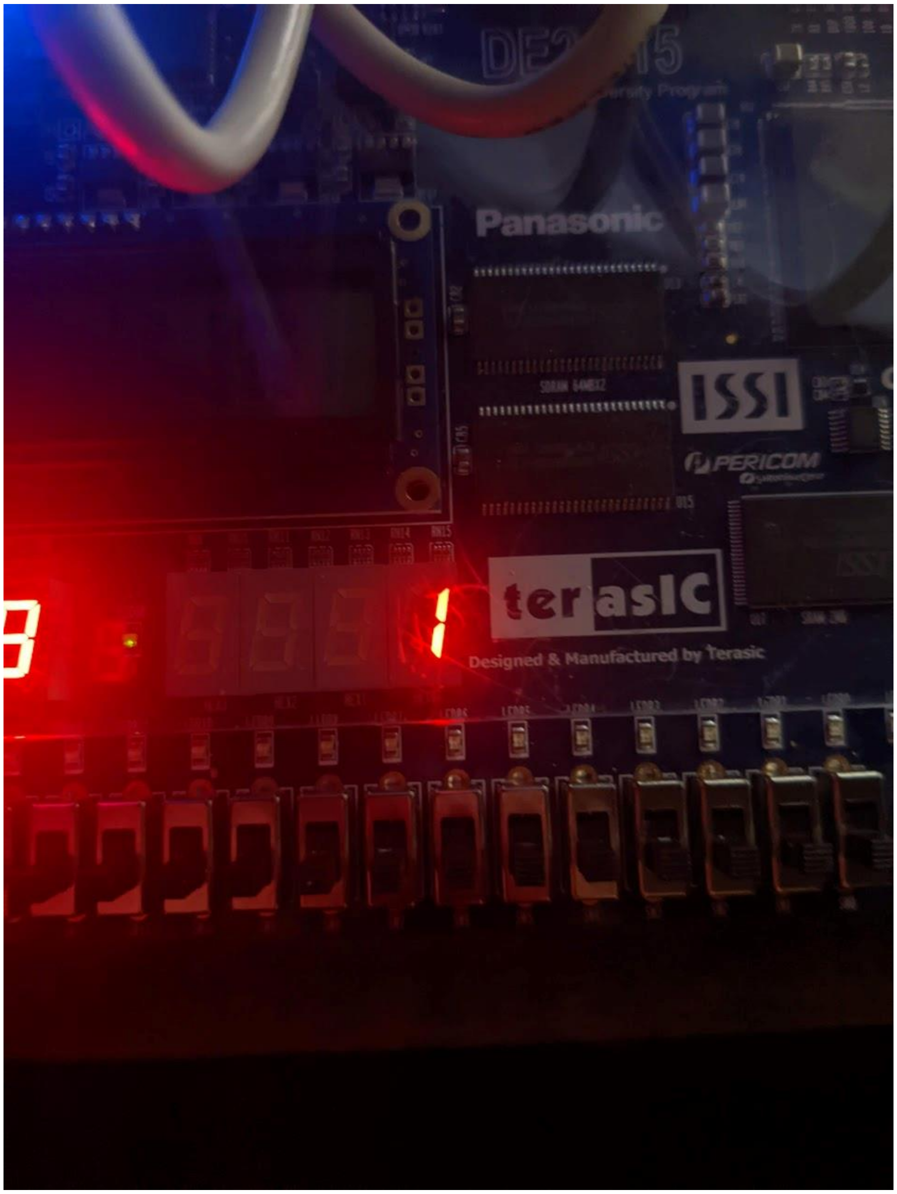

4. Interval Timer Properties

After adding the interval timer to the system, we must connect it to the rest of the system. This can be seen in the figure below.



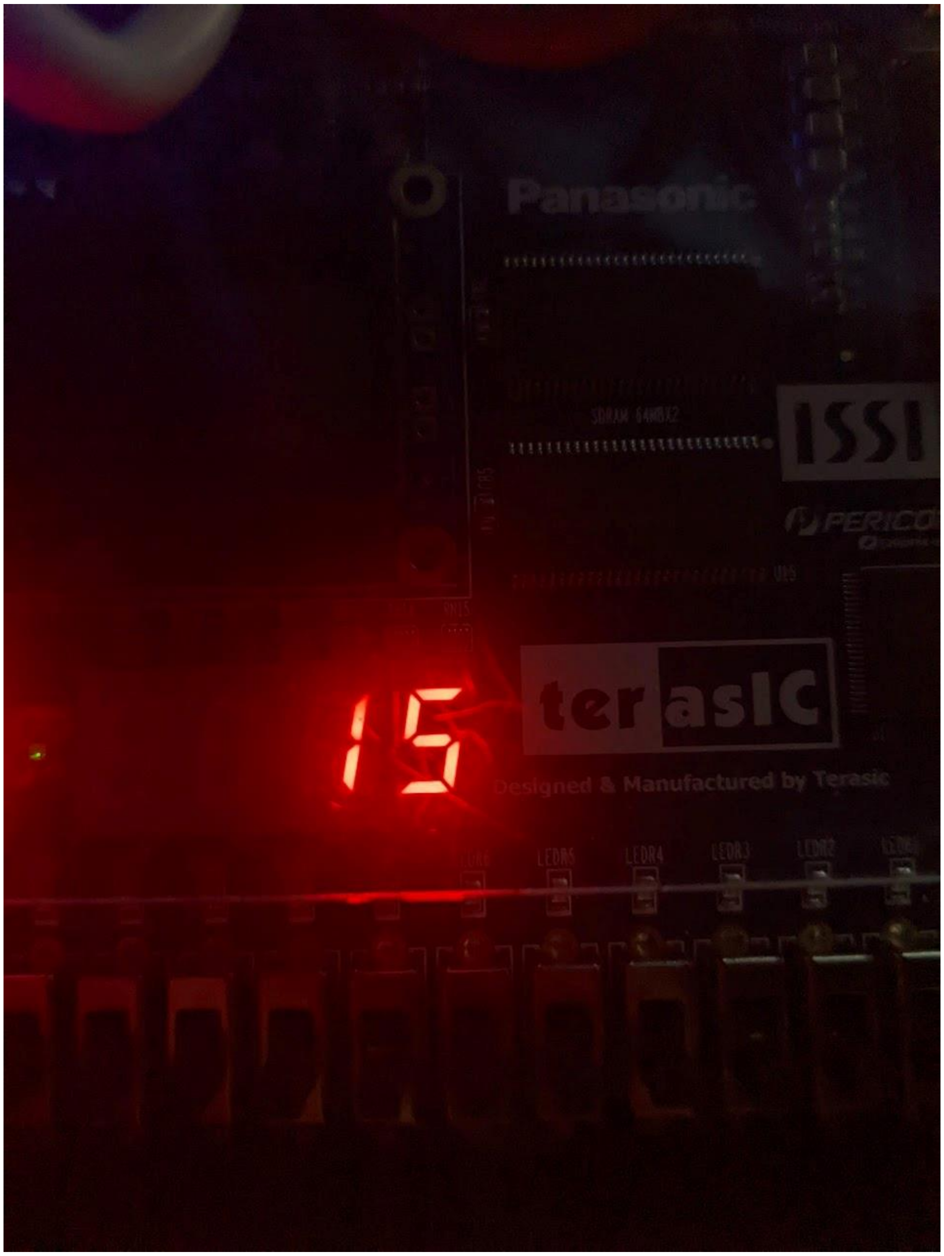

5. QSys Timer Connections

Now the timer is connected in Qsys, the system is then compiled using the code in Appendix A, top level instantiation. Once this is complete, we launch Nios II Software build tools for eclipse, and create the NIOS II project with the BSP. Upon doing this, we can then begin Part one. An odd number counter that goes from one to fifteen and increments every second. This counter can be started and stopped when any key is pressed. The code for this can be seen in Appendix B Part 1 Code. This counter starts at 1. Seen below.



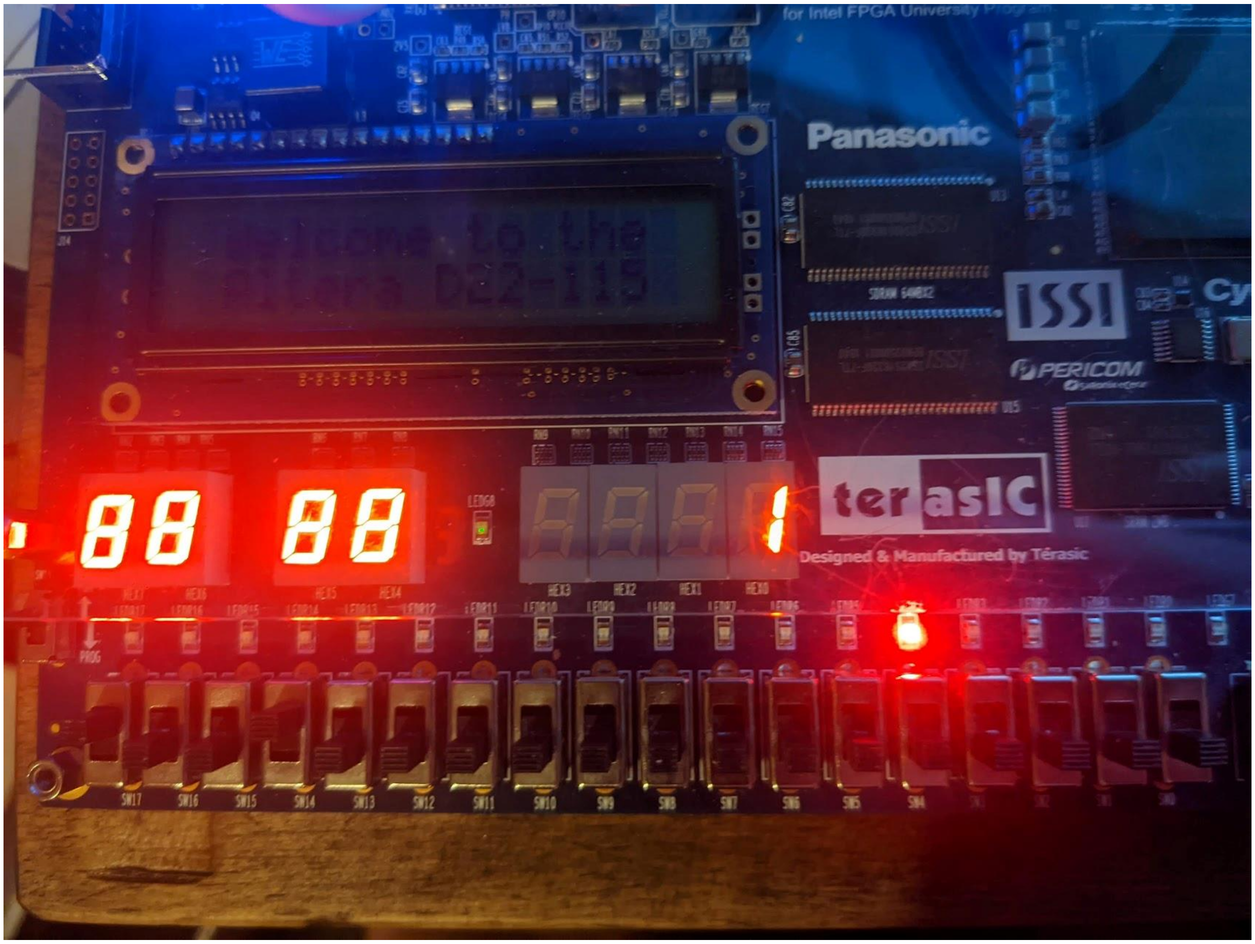
6. Counter starting point

The counter will then count by odd numbers until it reaches 15. Here, we opted to display it as the value 15. The max counter value can be seen in the figure below.



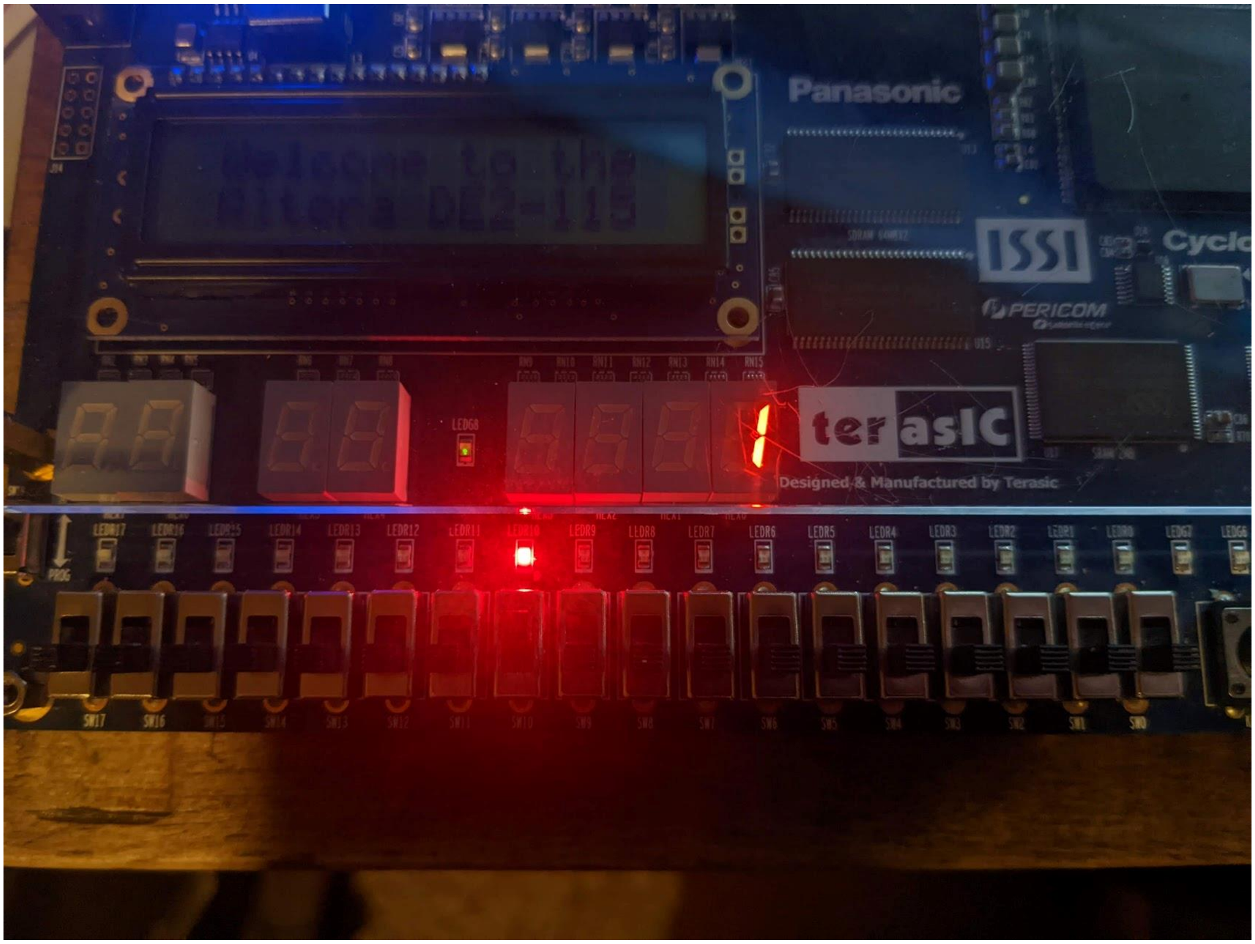

7. Counter Max Value

Upon reaching this value, the program will then count down. 15 to 13 and so on until reaching one and beginning counting up again. Now, with this program complete, we move to part two, which implements a walking ones on the LEDs and changes the speed depending on the Key Pushbutton that is pressed. The code for Part2 can be seen in Appendix C Part 2 code. With this code, the single lit LED will bounce from LED0 to LED17 unless Key0 is pressed which is the Start/Stop key. This LED on can be seen below.




8. Completed Download onto the Board

Walking ones LED on. Now, with this, the Keys change the speed in which the LED moves. Key 1 will make it move every 125ms key 2 every 500ms and key 3 every 1 second. This plays a bigger role in Part 3. Here we turn this into a sort of game. The code for this part can be seen in Appendix D Part 3 Code. Here, we start with all the LEDs off, and the Led in a walking ones that iterates every second. With one switch on, pressing the start/stop key when the LED is over the on switch adds one to the score, this can be seen below.




9. Score 1 Part 3

Here, we can see that the stop button, key 1 was pressed as the LED was over Switch 10, which is also the only key in the on position. Since this happened, the Score shown on the Hex display iterated to 1. This will go until the max score of F (15) is reached. Next, we will be analyzing the code.

## Data Analysis

For every part of this lab, we have used a timer, the code for initializing this timer can be seen below.

```
void init_timer_interrupt (){
    //Register ISR
    alt_ic_isr_register(TIMER_0_IRQ_INTERRUPT_CONTROLLER_ID, TIMER_0_IRQ, (void *)timer_isr, NULL, 0x0);
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE, ALTERA_AVALON_TIMER_CONTROL_CONT_MSK
                                                | ALTERA_AVALON_TIMER_CONTROL_START_MSK
                                                | ALTERA_AVALON_TIMER_CONTROL_ITO_MSK);
}
```

10. Init_timer_interrupt function

This initialization sets the ISR register for the timer so that the timer interrupts the program and when it does interrupt, it calls the function timer_isr. The timer is also initialized using the IOWR_ALTERA_AVALON_TIMER_CONTOL function. This function is writing to the timer base, and writes ones to the Control, Start, and ITO using their masks. Next, we can see the interrupt handler, seen below.

```c
static void timer_isr(void* context, alt_u32 id){
    static int count = 0;
    static int LR = 0;

    //clear interrupt
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE, 0);
    //Do something
    int LED = IORD(RED_LEDS_BASE, 0);
    if (LED == 1)LR = 0;
    if (LED == 131072) LR =1;
    switch(Time){
        case 0:
            if (count % 1000 == 0){
                if (LR == 0)LED = LED * 2;
                else LED = LED / 2;
            }
            break;
        case 1:
            if (count % 500 == 0){
                if (LR == 0)LED = LED * 2;
                else LED = LED / 2;
            }

            break;
        case 2:
            if (count % 125 == 0){
                    if (LR == 0)LED = LED * 2;
                    else LED = LED / 2;
                }

            break;
        default:

            break;
    }
    IOWR(RED_LEDS_BASE, 0, LED);
    //Timer Expires
    count ++;
    return;
```

11. Timer_ISR Function

For this timer, we are triggering the ISR every 1ms. So using modulo we see if it is at 125 ms 500ms or 1 second. However, we could improve the processing power required by this by changing the period hi and period lo values of the timer. Using the LR variable allows us to know wether or not we should be walking left or right, multiplying or dividing by two. These are the necessary functions for implementing a timer.

## Conclusion

This lab accomplished its goals to further the understanding of implementing Timers in a C++ runtime enviroment. Timers are an essential function of FPGA boards and embedded system design. This lab was very important moving forward and being able to be a competent embedded system designer.

## 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(),
```

```
.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)

```
#include <stdio.h>
#include <system.h>
#include <alt_types.h>
#include "altera_avalon_pio_regs.h"
#include <stdio.h>
#include <sys/alt_irq.h>
#include <altera_avalon_timer_regs.h>

void init_timer_interrupt ();
static void timer_isr(void* context, alt_u32 id);
void display(int Value);

int main()
{
	init_timer_interrupt();
	while(1){

	}
  return 0;
}

void init_timer_interrupt (){
	//Register ISR
	alt_ic_isr_register(TIMER_0_IRQ_INTERRUPT_CONTROLLER_ID,
TIMER_0_IRQ, (void *)timer_isr, NULL, 0x0);
	IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE,
ALTERA_AVALON_TIMER_CONTROL_CONT_MSK

	| ALTERA_AVALON_TIMER_CONTROL_START_MSK

	| ALTERA_AVALON_TIMER_CONTROL_ITO_MSK);
}

static void timer_isr(void* context, alt_u32 id){
	static int count = 0;
	static int Num = 1;
	static int IncDec = 0;

	//clear interrupt
	IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE, 0);
	//Do something
	if (count % 8 == 0){
		display(Num); //Display Num
```

```
		if (IncDec == 0){ //Incrementing Check if equal max 
value begin decrement if so
			Num = Num + 2;
			if (Num == 15) IncDec = 1;
		}
		else if(IncDec == 1){ //Incrementing Check if equal 
max value begin decrement if so
			Num = Num - 2;
			if (Num == 1) IncDec = 0;
		}

	}

	//Timer Expires
	count ++;
	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 | 0xFFFFFF00;
	}
	toDisplayMask = toDisplay & 0x000000FF;
	if (toDisplayMask == 0){
		toDisplay = 0xFFFFFF40;
	}

	IOWR(SEVSEGMENT_4LSB_BASE, 0, toDisplay);
```

```
        return;
}
```

## Appendix C (Part 2 Code)

```c
#include <stdio.h>
#include <system.h>
#include <alt_types.h>
#include "altera_avalon_pio_regs.h"
#include <stdio.h>
#include <sys/alt_irq.h>
#include <altera_avalon_timer_regs.h>

void init_timer_interrupt ();
static void timer_isr(void* context, alt_u32 id);
int Time = 0;

void key3_isr(){
	// 1 second Timer
	Time = 0;
	printf("Key 3 Int");
	IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
	IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
	return;
}

void key2_isr(){
	// .5 Second Timer
	Time = 1;

	printf("Key 2 Int");
	IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
	IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
	return;
}

void key1_isr(){
	//.125 s Timer
	Time = 2;
	printf("Key 1 Int");
	IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
	IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
	return;
}

void key0_isr(){
	//Start/Stop
```

```
	printf("Key 0 Int");
	int Status = IORD_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE);
	if (Status != 0){
		IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE,
0b1011);
	}
	else if (Status == 0){
		IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE,
0b0111);
	}

	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();
	}
	else if (*edge_capture_ptr & 0x1){
		key0_isr();
	}
	return;
}

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

	return;
	}


void init_timer_interrupt (){
```

```
	//Register ISR
	alt_ic_isr_register(TIMER_0_IRQ_INTERRUPT_CONTROLLER_ID,
TIMER_0_IRQ, (void *)timer_isr, NULL, 0x0);
	IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE,
ALTERA_AVALON_TIMER_CONTROL_CONT_MSK

	| ALTERA_AVALON_TIMER_CONTROL_START_MSK

	| ALTERA_AVALON_TIMER_CONTROL_ITO_MSK);
}

static void timer_isr(void* context, alt_u32 id){
	static int count = 0;
	static int LR = 0;

	//clear interrupt
	IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE, 0);
	//Do something
	int LED = IORD(RED_LEDS_BASE, 0);;
	if (LED == 1)LR = 0;
	if (LED == 131072) LR =1;
	switch(Time){
		case 0:
			if (count % 1000 == 0){
				if (LR == 0)LED = LED * 2;
				else LED = LED / 2;
			}
			break;
		case 1:
			if (count % 500 == 0){
				if (LR == 0)LED = LED * 2;
				else LED = LED / 2;
			}

			break;
		case 2:
			if (count % 125 == 0){
				if (LR == 0)LED = LED * 2;
				else LED = LED / 2;
			}
			break;
		default:

			break;
	}
	IOWR(RED_LEDS_BASE, 0, LED);
	//Timer Expires
```

```c
    count ++;
    return;
}

int main(){
    init_timer_interrupt();
    pio_init();
    IOWR(RED_LEDS_BASE, 0, 1);
    while(1){

    }
  return 0;
```

## Appendix D (Part 3 Code)

```c
#include <stdio.h>
#include <system.h>
#include <alt_types.h>
#include "altera_avalon_pio_regs.h"
#include <stdio.h>
#include <sys/alt_irq.h>
#include <altera_avalon_timer_regs.h>

void init_timer_interrupt ();
static void timer_isr(void* context, alt_u32 id);
void display(int Value);
int Time = 0;
int Score = 0;


void key3_isr(){
	// 1 second Timer
	Time = 0;
	IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
	IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
	return;
}

void key2_isr(){
	// .5 Second Timer
	Time = 1;

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

void key1_isr(){
	//.125 s Timer
	Time = 2;
	IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
	IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
	return;
}

void key0_isr(){
	//Start/Stop
	int SwVal = IORD(SWITCHES_BASE, 0);
	int Status = IORD_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE);
```

```c
	if (Status != 0){
		IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE,
0b1011);
		if (SwVal == IORD(RED_LEDS_BASE, 0)) {
			Score++;
			if (Score > 16){
				Score = 0;
			}
		}
		else Score = 0;
		display(Score);
	}
	else if (Status == 0){
		IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE,
0b0111);
	}

	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();
	}
	else if (*edge_capture_ptr & 0x1){
		key0_isr();
	}
	return;
}

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

```
	return;
	}


void init_timer_interrupt (){
	//Register ISR
	alt_ic_isr_register(TIMER_0_IRQ_INTERRUPT_CONTROLLER_ID,
TIMER_0_IRQ, (void *)timer_isr, NULL, 0x0);
	IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE,
ALTERA_AVALON_TIMER_CONTROL_CONT_MSK

	| ALTERA_AVALON_TIMER_CONTROL_START_MSK

	| ALTERA_AVALON_TIMER_CONTROL_ITO_MSK);
}

static void timer_isr(void* context, alt_u32 id){
	static int count = 0;
	static int LR = 0;

	//clear interrupt
	IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE, 0);
	//Do something
	int LED = IORD(RED_LEDS_BASE, 0);
	if (LED == 1)LR = 0;
	if (LED == 131072) LR =1;
	switch(Time){
		case 0:
			if (count % 1000 == 0){
				if (LR == 0)LED = LED * 2;
				else LED = LED / 2;
			}
			break;
		case 1:
			if (count % 500 == 0){
				if (LR == 0)LED = LED * 2;
				else LED = LED / 2;
			}

			break;
		case 2:
			if (count % 125 == 0){
					if (LR == 0)LED = LED * 2;
					else LED = LED / 2;
				}
```

```
                break;
            default:

                break;
        }
        IOWR(RED_LEDS_BASE, 0, LED);
        //Timer Expires
        count ++;
        return;
}

int main(){
        init_timer_interrupt();
        pio_init();
        IOWR(RED_LEDS_BASE, 0, 1);
        IOWR(SEVSEGMENT_4LSB_BASE, 0, 0xFFFFFFFF);
        IOWR(SEVSEG4MSB_BASE, 0, 0xFFFFFFFF);
        while(1){

        }
    return 0;
}

void display(int Value){
        int SEVENSEGLUT[16] = {0b01000000, 0b01111001, 0b00100100,
0b00110000, 0b00011001, 0b00010010, 0b00000010, 0b01111000,
0b10000000, 0b00010000, 0b00001000, 0b00000011, 0b01000110
,0b00100001, 0b00000110, 0b00001110};
        int toDisplay = 0;
        int shiftVal = 0;
        if (Value == 0){
                toDisplay = SEVENSEGLUT[Value];
        }

        while (Value != 0){ //Displays the value on the Hex Masks
are for 0 values to not be displayed unless the LSB
        toDisplay = toDisplay + (SEVENSEGLUT[Value] << shiftVal);
        Value = 0;
        shiftVal = shiftVal + 8;
        }
        toDisplay = toDisplay | 0xFFFFFF00;

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