

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

TECHNICAL REPORT

Assignment: Number 8
Experiment Title: Eye Tracking/Hand Synchronization
Course Title: ECE 178 (Embedded Systems)
Instructor: Dr. Reza Raeisi

Prepared by: Anthony Herrick
Student ID #300144976
Date Submitted: 12/08/2022

INSTRUCTOR SECTION

Comments: _____

Final Grade: _____

TABLE OF CONTENTS

Objective	4
Hardware Requirements.....	4
Software Requirements	4
Background	4
Project Overview	8
Project Procedure	9
Data Analysis	Error! Bookmark not defined.
Conclusion	16
Appendix A (Top Level Instantiation).....	17
Appendix B (Part 1 Code).....	19
Appendix C (Part 2 Code).....	24
Appendix D (Part 3 Code)	Error! Bookmark not defined.

LIST OF FIGURES

1.	7-Segment Display Address Spaces	5
2.	7-Segment Display Breakdown	6
3.	Interval Timer Registers	7
4.	Interval Timer Properties	9
5.	QSys Timer Connections	10
6.	Counter starting point	Error! Bookmark not defined.
7.	Counter Max Value	12
8.	Completed Download onto the Board	Error! Bookmark not defined.
9.	Score 1 Part 3	Error! Bookmark not defined.
10.	Init_timer_interrupt function	Error! Bookmark not defined.
11.	Timer_ISR Function	Error! Bookmark not defined.

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

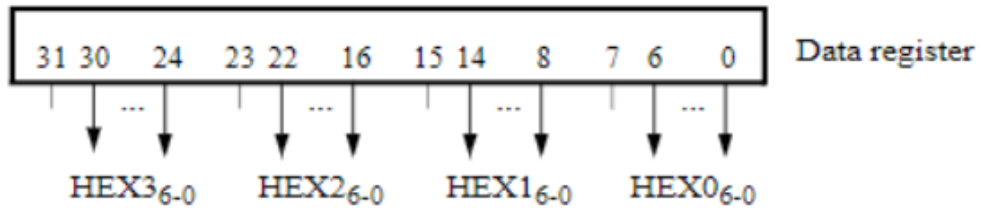
- Quartus Prime with Qsys, version 16.1.

Background

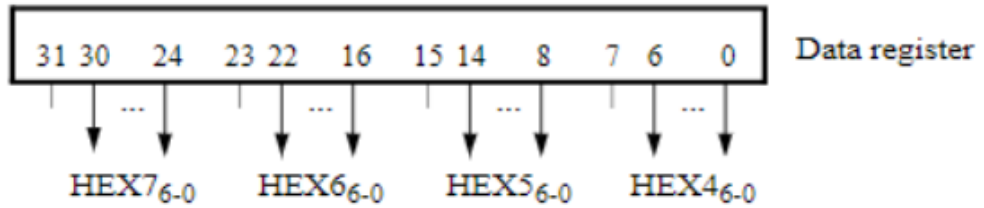
In order to write to a 7-segment display, we have two address spaces, one is for the four most significant bits and the second for the four least significant bits. This is because the DE2-115 board has eight 7-Segment displays. From the design in the last lab, the address spaces for these were, 0x2020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

Address

0x10000020

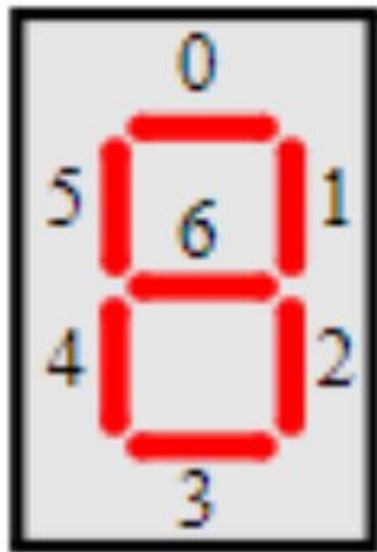


0x10000030



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 are 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.

Address	31	...	17	16	15	...	3	2	1	0			
0xFF202000	Not present (interval timer has 16-bit registers)					Unused				RUN	TO	Status register	
0xFF202004						Unused		STOP	START	CONT	ITO	Control register	
0xFF202008						Counter start value (low)							
0xFF20200C						Counter start value (high)							
0xFF202010						Counter snapshot (low)							
0xFF202014						Counter snapshot (high)							

3. Interval Timer Registers

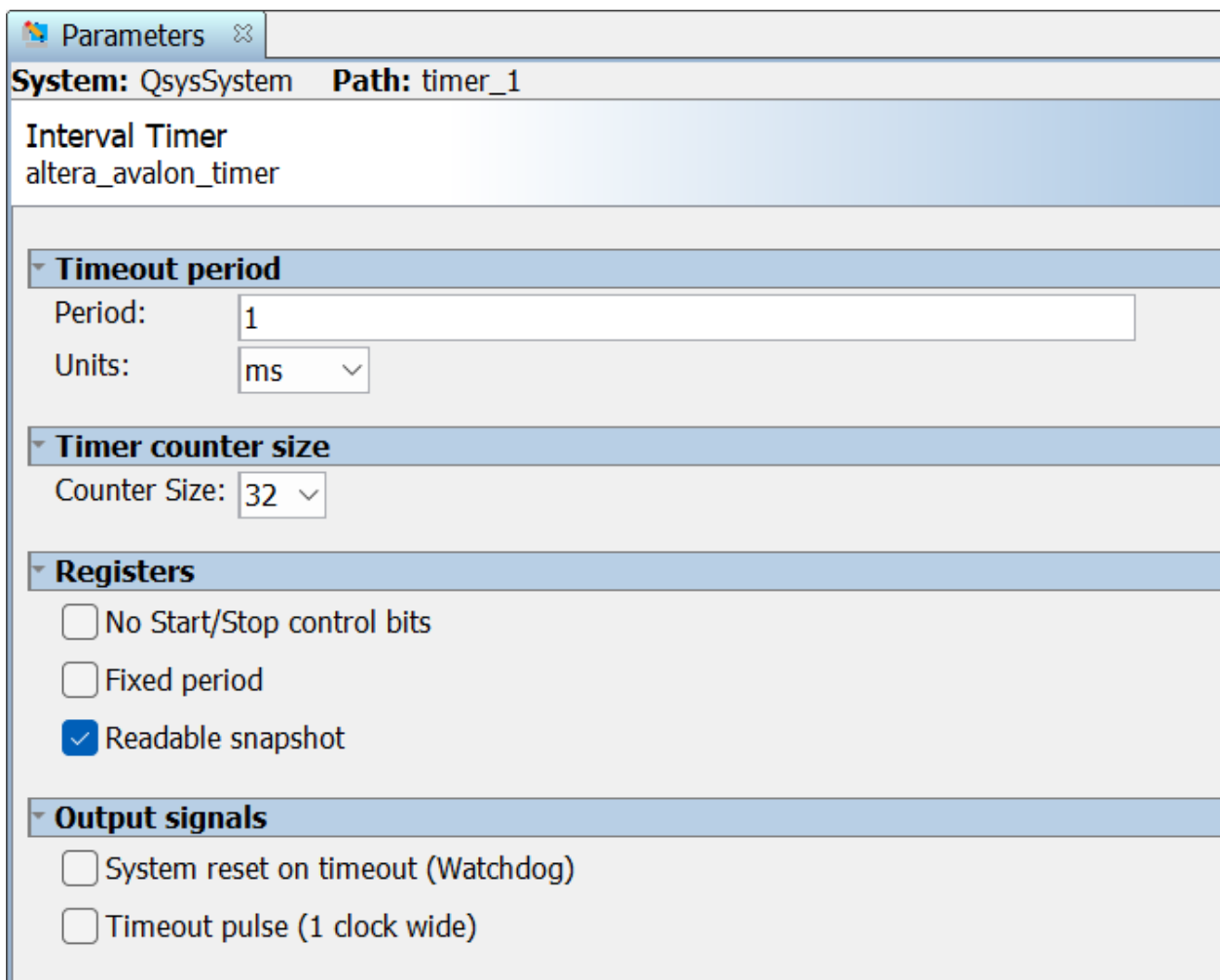
For an interval timer, the registers can be seen above. These have 16-bit register. The counter start value low and high can be altered to change the duration of the timer this is periodlo and periodhi. The control register is then used to begin the timer. This timer can be used to accurately measure 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 environment. We will be creating an application that implements a reaction time test. There will be three pushbuttons, these pushbuttons correspond to clear, start, and stop operations. The clear key will reset the system, setting the 7 Segment to display “Start” and the LEDs to be off. Next, the user will push the start button, here a random delay between 2 and 15 seconds will begin. Upon completion of this delay, the red LEDs will begin to flash on and off and the timer will count. This timer will track how long in milliseconds it takes for the user to press the stop push button. If the user fails to press the button within one second the seven segment will display “FAIL”, if the user presses the key before the LEDs begin flashing they will be met with a “OOPS” on the seven segment, and if they press it in the correct window it will display their time in milliseconds on the seven segment. The second part of this lab was to use JTAG UART IP core in order to use the keyboard keys to complete this process.

Project Procedure

For this lab, we begin by adding the interval timer to the QSys design of this system, Here, we have a timer with a 1ms period. This single timer will be used for all the operations present in the software side of the project.

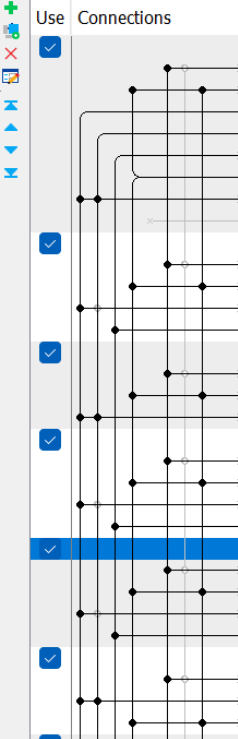


The screenshot shows the 'Parameters' window for the 'Interval Timer' component. The window has a title bar with a 'Parameters' tab and a close button. Below the title bar, the 'System' is set to 'QsysSystem' and the 'Path' is 'timer_1'. The component name 'Interval Timer' and its library path 'altera_avalon_timer' are displayed. The parameters are organized into expandable sections:

- Timeout period**
 - Period: 1
 - Units: ms
- Timer counter size**
 - Counter Size: 32
- Registers**
 - ☐ No Start/Stop control bits
 - ☐ Fixed period
 - ☒ Readable snapshot
- Output signals**
 - ☐ System reset on timeout (Watchdog)
 - ☐ Timeout pulse (1 clock wide)

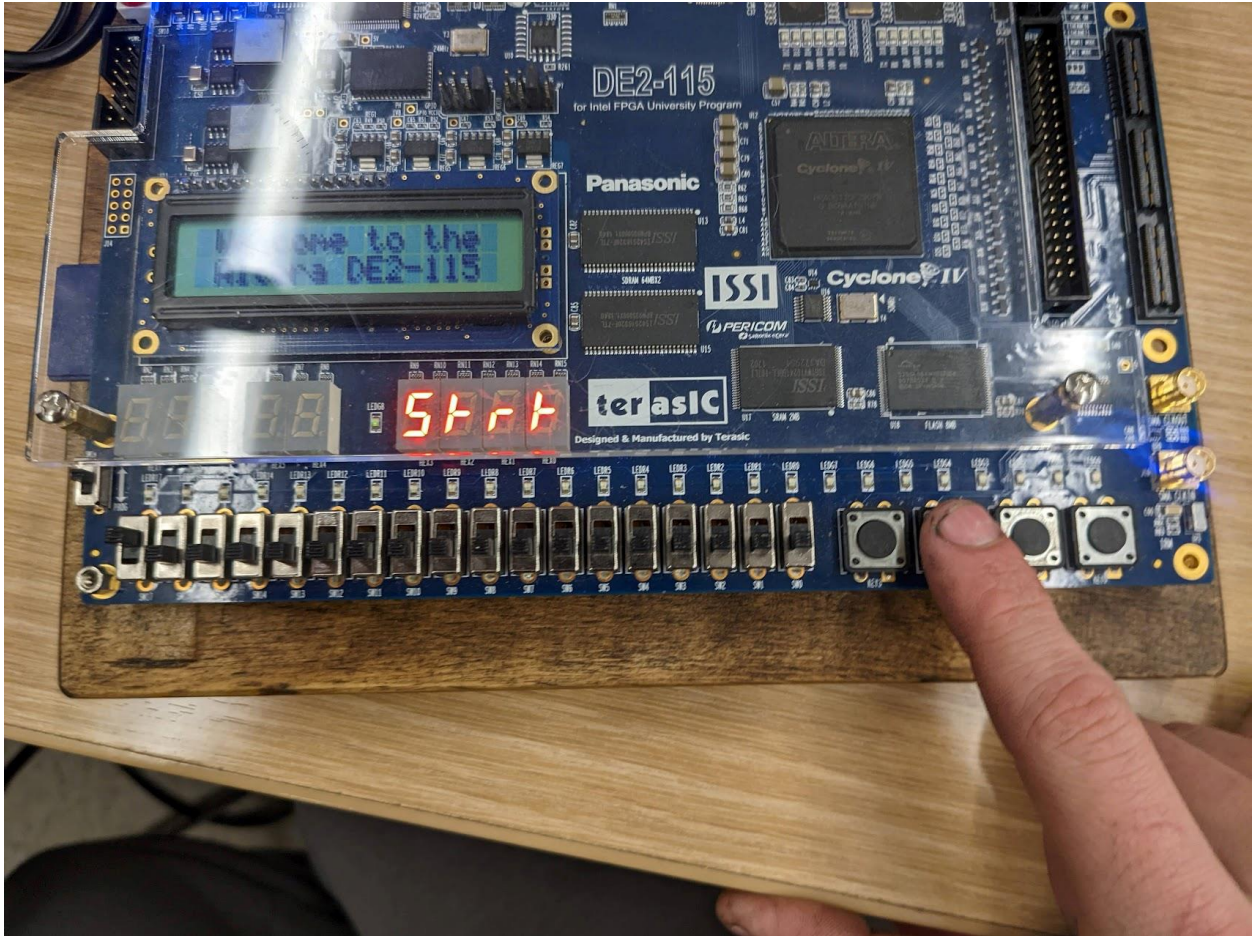
4. Interval Timer Properties

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.

System Contents Address Map Interconnect Requirements								
System: QsysSystem Path: timer_1								
Use	Connections	Name	Description	Export	Clock	Base	End	I...
		Processor	Nios II (Classic) Processor	Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to	CLK_s...	IRQ 0	IRQ 31	
		clk	Clock Input					
		reset_n	Reset Input					
		data_master	Avalon Memory Mapped ...					
		instruction_m...	Avalon Memory Mapped ...					
		d_irq	Interrupt Receiver					
		jtag_debug_m...	Reset Output					
		jtag_debug_m...	Avalon Memory Mapped ...					
		custom_instru...	Custom Instruction Master					
		JTAG_UART	JTAG UART					
		clk	Clock Input	Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to	CLK_s...	# 0x0000 20b0	0x0000_20b7	
		reset	Reset Input					
		avalon_jtag_sl...	Avalon Memory Mapped ...					
		irq	Interrupt Sender					
		SYSID	System ID Peripheral					
		clk	Clock Input					
		reset	Reset Input					
		control_slave	Avalon Memory Mapped ...					
		timer_0	Interval Timer					
		clk	Clock Input					
		reset	Reset Input	Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to	CLK_s...	# 0x0000 2020	0x0000_203f	
		s1	Avalon Memory Mapped ...					
		irq	Interrupt Sender					
		timer_1	Interval Timer					
		clk	Clock Input					
		reset	Reset Input					
		s1	Avalon Memory Mapped ...					
		irq	Interrupt Sender					
		On_Chip_Mem	On-Chip Memory (RAM o...					
		clk1	Clock Input					
		s1	Avalon Memory Mapped ...	Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to	CLK_s...	# 0x0000 2000	0x0000_201f	
		reset1	Reset Input					
		clk1	Clock Input					
		s1	Avalon Memory Mapped ...					
		reset1	Reset Input					
		clk1	Clock Input					
		s1	Avalon Memory Mapped ...					
		reset1	Reset Input					
		clk1	Clock Input					
		s1	Avalon Memory Mapped ...					

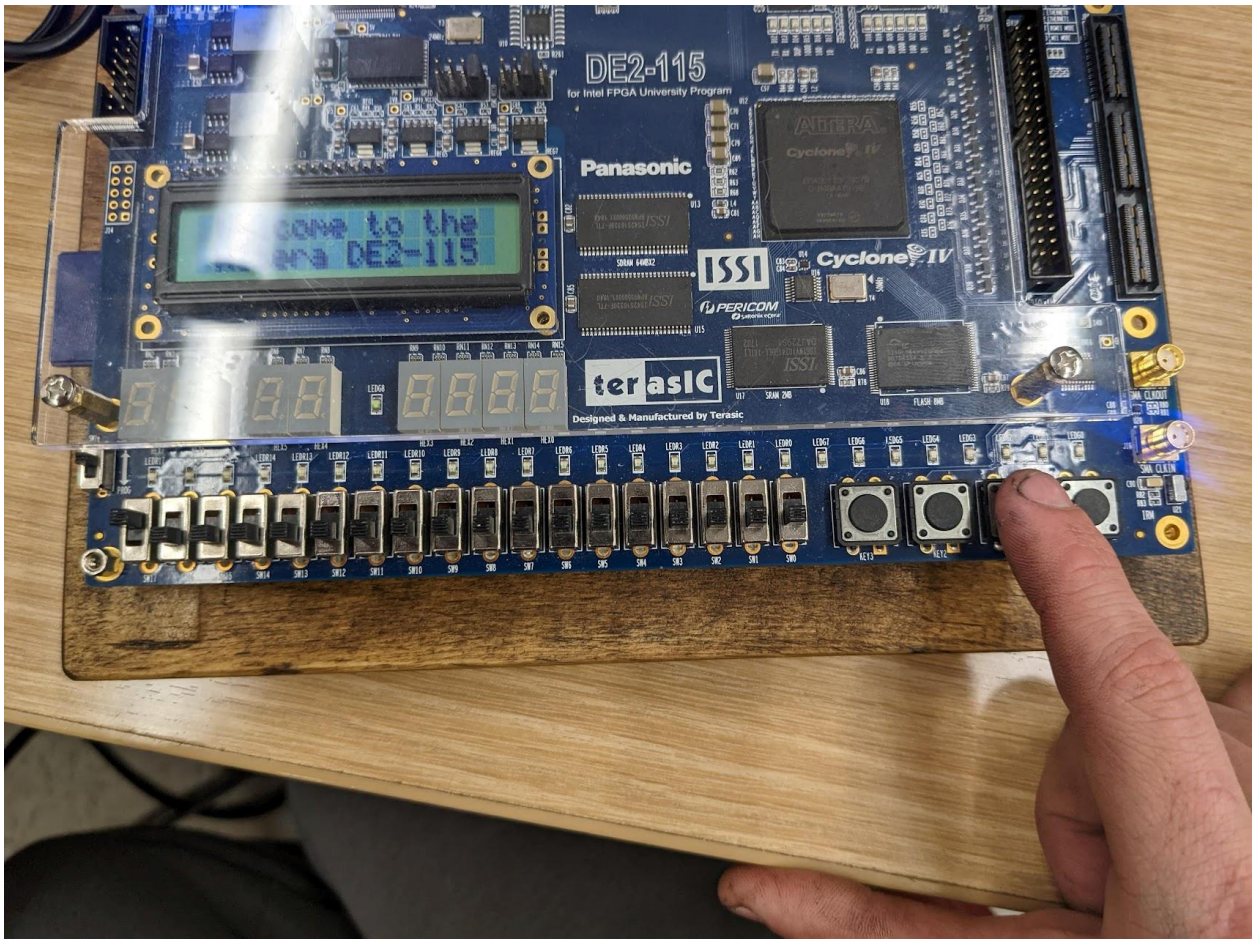
5. QSys Timer Connections

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. The code for this part can be seen in Appendix B part one code. It is shown below the DE2 board after clear is pressed.



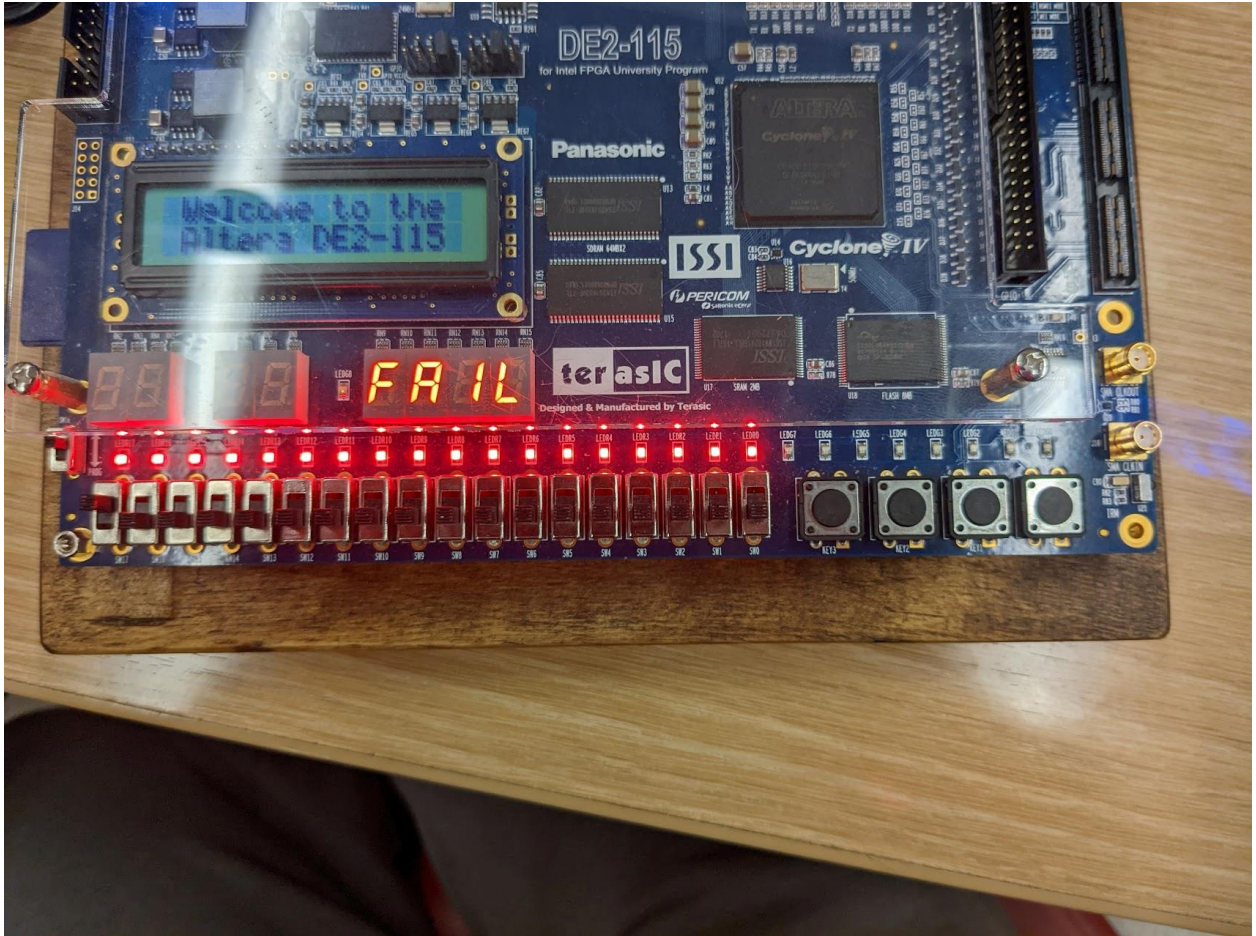
6. Clear Pressed

After Key2 is pressed, the interrupt handler for key2 triggers. For this, our global variable for state is set to one, and the 7 Segment display shows Strt, “Start” as seen above. In this state the timer is turned off, and the interrupt is reset. The program loops and does nothing until the start key is pressed. After Key1 is pressed, the start key, the board then looks as it does below.



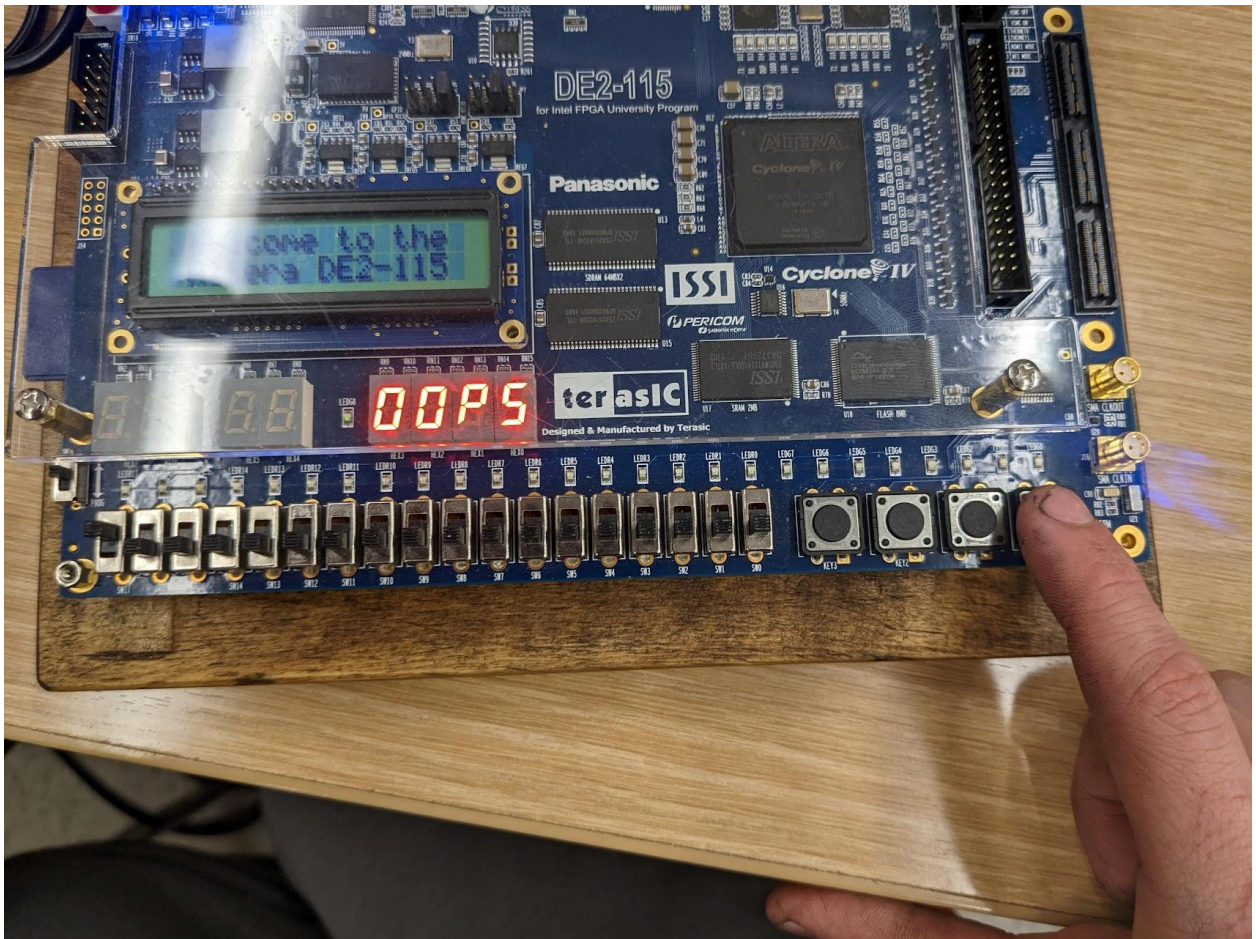
7. Counter Max Value

In this state, the seven segment display is turned off, and the global variable “MSTODELAY” becomes a random value from 2000 to 15000, or 2 to 15 seconds. The timer then begins and decrements the MSTODELAY every millisecond. When the variable becomes 0, the state becomes 3. In this state, every 25ms the LEDs change, and our TIME variable increments every 1ms. This tracks how long it takes for the user to press the stop key. If the time variable goes over 1000, the state becomes 4 which is the fail state. If this occurs, the DE2 looks as it does below.



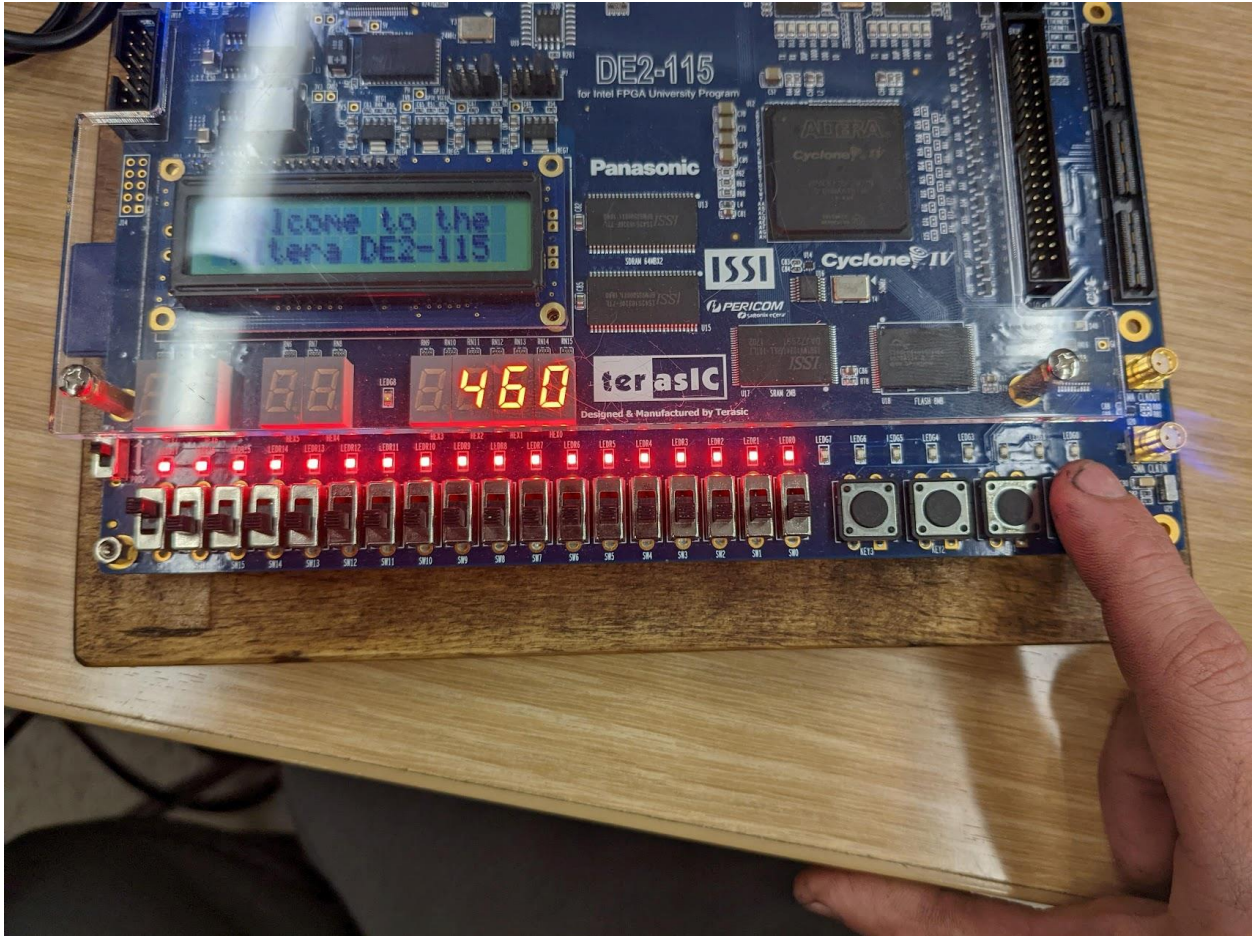
8. Fail state

In this fail state, the program goes back to infinite looping until the clear is pressed, and the “FAIL” is displayed on the seven segments. If key0, the stop key is pressed before state 3 is entered by the program, we enter the second fail state which is oops, seen below.



9. Oops state

Above, we can see the oops state, as stated before, this occurs when key0 is pressed and the program is still in state two, where the MSTODELAY has not yet reached 0. After oops is displayed, the program will then again go back into the infinite loop until the clear is pressed. The last state is the win state, the stop button is pressed after the LEDs begin blinking and before one second. This outcome can be seen below.



10. Win State

Here, for simplicity sake, we are calling this the win state, here the TIME variable is displayed on the seven segment display using our Display function, the red LEDs then stay on until the clear is pressed and we can begin again. In the case above the stop button was pressed 460ms after the lights began blinking. This completes part 1. Part 2 simply used JTAG UART to accomplish the same goals. This code can be seen in Appendix C part2 code. We used the NIOS2 terminal, connecting to JTAG UART, and using the code from Appendix C.

Conclusion

This lab accomplished its goals to further the understanding of implementing Timers in a C++ runtime environment. Timers are an essential function of FPGA boards and embedded system design. We used timer in conjunction with interrupts in order to accomplish our goals in this lab. Being able to use these different systems in our program allowed us to further grasp the knowledge of embedded systems design and firmware as a whole.

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(LED_R),
.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 timer0_isr(void* context, alt_u32 id);
static void timer1_isr(void* context, alt_u32 id);
void display(int Value);

/*Three Push Button Interrupts Clear, Start, Stop.
   Red LEDS flash on and off //Every count ++ %10 = .01 s
   Timer Create Delays as well as Keep track of time //Begin
timer after Start
   Clear Initially display START with no LEDS // No LEDS
   Start is pushed, Turns off 7-Segment and LEDR // 7 Seg
Turned off
   Rand Interval 2-15 seconds flashes on and off //.01
second flash
   Timer Counts
   Stop button pressed, 0000 Milliseconds Displayed on 7
   If Stop pressed after 1 second, Fail, if Pressed before
OOPS

   State 0: Not Started
   State 1: Clear Pressed
   State 2: Start is pressed Before Flashing
   State 3: After Flashing
   State 4: time Expired
   State 5: Stop is Pressed In time
*/
int MSTODELAY = 0;
int TIME = 0;
int state = 0;

void key3_isr() {
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
    IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    return;
}
```

```

void key2_isr() {
    //Clear
    TIME = 0;
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE, 0b1011);
    //Display "START"
    IOWR(SEVSEGMENT_4LSB_BASE, 0,
0b00010010000011110010111100001111);
    IOWR(RED_LEDS_BASE, 0, 0b00000000000000000000);
    state = 1;
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
    IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    return;
}

void key1_isr() {
    //Start
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_1_BASE, 0b0111);
    MSTODELAY = ((rand() % 14) + 2) * 1000;
    //COUNT TILL MSTODELAY
    //When reach MSTODELAY being LED Flashing, and Reset Count
    //Count 1 every ms and Display that value to Display when
STOP is pressed
    if (state == 1) {
        state = 2;
        IOWR(SEVSEGMENT_4LSB_BASE, 0, 0xFFFFFFFF);
    }
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
    IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    return;
}

void key0_isr() {
    //Stop
    int OOPS = 0b01000000010000000000110000010010;
    int FAIL = 0b00001110000010000111100101000111;
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_1_BASE, 0b1011);
    if (state == 2) {
        IOWR(SEVSEGMENT_4LSB_BASE, 0, OOPS);
    }
    else {
        display(TIME);
        state = 5;
    }
    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 *)timer0_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);
    alt_ic_isr_register(TIMER_1_IRQ_INTERRUPT_CONTROLLER_ID,
TIMER_1_IRQ, (void *)timer1_isr, NULL, 0x0);
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_1_BASE,
ALTERA_AVALON_TIMER_CONTROL_CONT_MSK

    | ALTERA_AVALON_TIMER_CONTROL_START_MSK

```

```

        | ALTERA_AVALON_TIMER_CONTROL_ITO_MSK);
    }

static void timer1_isr(void* context, alt_u32 id){
    static int count = 0;
    static int LR = 0;

    //clear interrupt
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_1_BASE, 0);
    //Do something
    if (state == 2){
        if (MSTODELAY == 0){
            state = 3;
        }
        MSTODELAY--;
    }
    if (state == 3){
        TIME++;
        if (TIME % 25 == 0) IOWR(RED_LEDS_BASE, 0,
0b11111111111111111111);
        if (count % 50 == 0) IOWR(RED_LEDS_BASE, 0,
0b00000000000000000000);
    }
    if (TIME == 1000){
        state = 4;
        IOWR(SEVSEGMENT_4LSB_BASE, 0,
0b000011110000010000111100101000111);
    }
    //Timer Expires
    count ++;
    return;
}

static void timer0_isr(void* context, alt_u32 id){
    static int count = 0;

    //clear interrupt
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE, 0);
    //Do something

    //Timer Expires
    count ++;
    return;
}

int main() {

```

```

    init_timer_interrupt();
    pio_init();
    IOWR(SEVSEGMENT_4LSB_BASE, 0, 0xFFFFFFFF);
    IOWR(SEVSEG4MSB_BASE, 0, 0xFFFFFFFF);
    while(1){
        if (MSTODELAY == 0){
as
        }
    }
    return 0;
}

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;
    IOWR(SEVSEGMENT_4LSB_BASE, 0, 0xFFFFFFFF);

    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;
    }
    IOWR(SEVSEGMENT_4LSB_BASE, 0, toDisplay | 0xFF000000);
    return;
}

```

Appendix C (Part 2 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>
#include <altera_avalon_jtag_uart_regs.h>
#include <altera_avalon_jtag_uart.h>

void init_timer_interrupt ();
static void timer0_isr(void* context, alt_u32 id);
static void timer1_isr(void* context, alt_u32 id);
void display(int Value);

/*Three Push Button Interrupts Clear, Start, Stop.
   Red LEDS flash on and off //Every count ++ %10 = .01 s
   Timer Create Delays as well as Keep track of time //Begin
timer after Start
   Clear Initially display START with no LEDs // No LEDs
   Start is pushed, Turns off 7-Segment and LEDR // 7 Seg
Turned off
   Rand Interval 2-15 seconds flashes on and off //.01
second flash
   Timer Counts
   Stop button pressed, 0000 Milliseconds Displayed on 7
   If Stop pressed after 1 second, Fail, if Pressed before
OOPS

   State 0: Not Started
   State 1: Clear Pressed
   State 2: Start is pressed Before Flashing
   State 3: After Flashing
   State 4: time Expired
   State 5: Stop is Pressed In time
*/
int MSTODELAY = 0;
int TIME = 0;
int state = 0;

void clear() {
```



```

    //Clear
    TIME = 0;
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_0_BASE, 0b1011);
    //Display "START"
    IOWR(SEVSEGMENT_4LSB_BASE, 0,
0b00010010000011110010111100001111);
    IOWR(RED_LEDS_BASE, 0, 0b00000000000000000000);
    state = 1;
    return;
}

void start() {
    //Start
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_1_BASE, 0b0111);
    MSTODELAY = ((rand() % 14) + 2) * 1000;
    //COUNT TILL MSTODELAY
    //When reach MSTODELAY being LED Flashing, and Reset Count
    //Count 1 every ms and Display that value to Display when
STOP is pressed
    if (state == 1) {
        state = 2;
        IOWR(SEVSEGMENT_4LSB_BASE, 0, 0xFFFFFFFF);
    }
    return;
}

void stop() {
    //Stop
    //int OOPS = 0b01000000010000000000110000010010;
    //int FAIL = 0b00001110000010000111100101000111;
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_1_BASE, 0b1011);
    if (state == 2) {
        IOWR(SEVSEGMENT_4LSB_BASE, 0,
0b0100000010000000000110000010010);
    }
    else {
        display(TIME);
        state = 5;
    }
    return;
}

void init_timer_interrupt () {
    //Register ISR
    alt_ic_isr_register(TIMER_0_IRQ_INTERRUPT_CONTROLLER_ID,
TIMER_0_IRQ, (void *)timer0_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);
    alt_ic_isr_register(TIMER_1_IRQ_INTERRUPT_CONTROLLER_ID,
    TIMER_1_IRQ, (void *)timer1_isr, NULL, 0x0);
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_1_BASE,
    ALTERA_AVALON_TIMER_CONTROL_CONT_MSK

    | ALTERA_AVALON_TIMER_CONTROL_START_MSK

    | ALTERA_AVALON_TIMER_CONTROL_ITO_MSK);
}

static void timer1_isr(void* context, alt_u32 id){
    static int count = 0;
    static int LR = 0;

    //clear interrupt
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_1_BASE, 0);
    //Do something
    if (state == 2){
        if (MSTODELAY == 0){
            state = 3;
        }
        MSTODELAY--;
    }
    if (state == 3){
        TIME++;
        if (TIME % 25 == 0) IOWR(RED_LEDS_BASE, 0,
0b111111111111111111);
        if (count % 50 == 0) IOWR(RED_LEDS_BASE, 0,
0b000000000000000000);
    }
    if (TIME == 1000){
        state = 4;
        IOWR(SEVSEGMENT_4LSB_BASE, 0,
0b00001110000010000111100101000111);
    }
    //Timer Expires
    count ++;
    return;
}

static void timer0_isr(void* context, alt_u32 id){

```

```

    static int count = 0;

    //clear interrupt
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_0_BASE, 0);
    //Do something

    //Timer Expires
    count ++;
    return;
}

int main(){
    init_timer_interrupt();
    volatile int *JTAG_UART_ptr = (int *) 0x20b0;
    alt_u32 data = *JTAG_UART_ptr;
    IOWR(SEVSEGMENT_4LSB_BASE, 0, 0xFFFFFFFF);
    IOWR(SEVSEG4MSB_BASE, 0, 0xFFFFFFFF);
    printf("starting");
    while(1){
        printf("A");
        //control = *(JTAG_UART_ptr + 1);
        //if (control & 0xFFFF0000){ // Whitespace

        //}
        data = *(JTAG_UART_ptr);
        data = data & 0x000000FF;
        //data =
        IORD_ALTERA_AVALON_JTAG_UART_DATA(JTAG_UART_BASE);
        switch (data){
            case (0x63):
                printf("clear\n");
                *(JTAG_UART_ptr) = 0;
                clear();
                break;

            case (0x73):
                printf("start\n");
                *(JTAG_UART_ptr) = 0;
                start();
                break;

            case (0x70):
                printf("stop\n");
                *(JTAG_UART_ptr) = 0;
                stop();
                break;
            default:

```

```

        break;
    }
    *(JTAG_UART_ptr) = 0;
}
return 0;
}

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;
    IOWR(SEVSEGMENT_4LSB_BASE, 0, 0xFFFFFFFF);

    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;
    }
    IOWR(SEVSEGMENT_4LSB_BASE, 0, toDisplay | 0xFF000000);
    return;
};

```