Final Exam

Zackary McClamma*†

June 7, 2020

1 Introduction

This project consisted of running the μC/OS-II^{TM}operating system with three tasks. The first task blinks a green LED. The second task posts a semaphore that is unlocked by a button ISR, upon unlocking the task then increments a 7 segment display. The third task displays a user menu to a terminal with various options that are discussed in further detail in the proceeding sections of this document. Included with the submission of this document are two videos, one video is of the button, 7 segment display, and the two LEDs, the second video is a capture of the putty terminal when running the EEPROM, DMA, and SDRAM options (I found it difficult to take video with my phone that was watchable while attempting to hold it steady for 3+ minutes for the SDRAM test to complete).

2 System Design

This system uses i2c communications to read and write to an EEPROM device on the board. The options for telling the OS what to do are selected by a user by using the menu displayed in the terminal, the temrinal menu also includes options to run a DMA test and a SDRAM test. The system includes the Nios II processor, an I2C controller, on-chip memory (RAM), a SDRAM controller, a DMA controller, and four peripherals a red LED, a green LED, a seven segment display, and a button. The button is set up to interrupt when pressed and release a Semaphore that allows the seven segment display increment code to execute. The code for turning the green LED on and off is put in the highest priority task and it toggles the LED every 500ms (making it flash every 1 second). The rest of the pieces to this project are accessed through a terminal that runs through a COM port, the terminal will display six options Write EEPROM, Read EEPROM, Turn on Red LED, Turn off Red LED, DMA Test, and SDRAM Test. A block diagram of the system can be seen below in Figure 1

*University of Dayton

†Dept. of Electrical and Computer Engineering, University of Dayton, 300 College Park, Dayton, OH 45469-0226, U.S.A. E-mail: mcclammaz1@udayton.edu

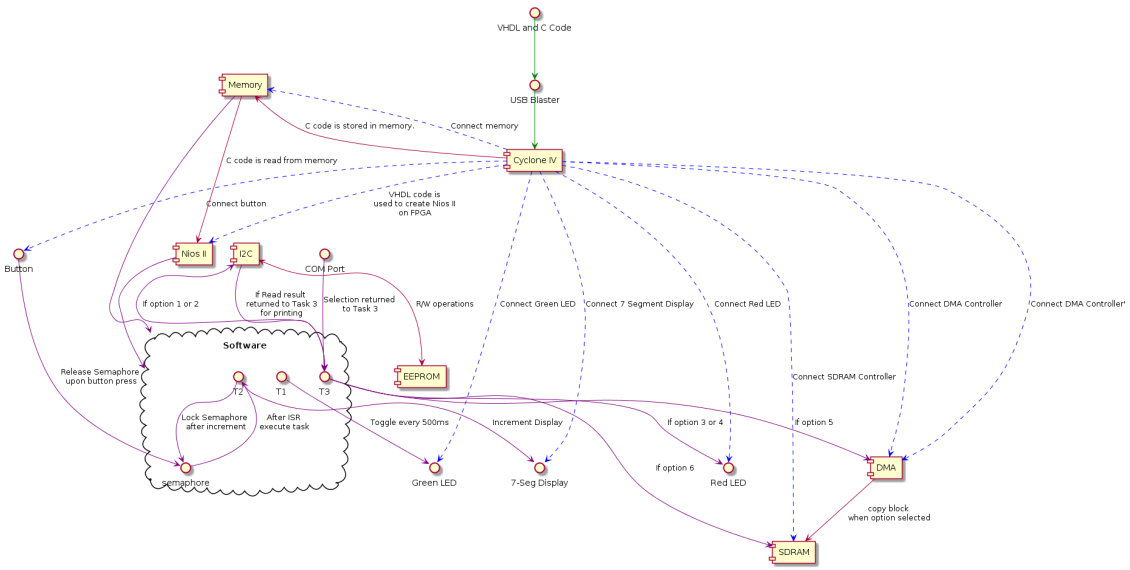

Figure (1) Embedded System Block Diagram

3 Theory of Operation

This system runs the μC/OS-II^{TM}operating system with three tasks. The first task toggles the green LED on and off every 500ms, this task was given the highest priority since it does not require interaction from the user and can run on its own. Making it the highest priority prevents other tasks from blocking it while waiting on input. The second task increments the seven segment display upon a button press. When the program initializes the button registers an ISR with the operating system so that when the button is pressed it triggers the button's ISR. The button ISR releases a Semaphore that is locked in the second task which calls the increment display method. The semaphore once locked waits until it is unlocked to continue executing the code below it, in this case that code is the *increment_display*() method. After the ISR is executed and the display is incremented the code loops back to locking the semaphore until the button ISR is triggered again.

The final task in this system contains all of the menu options that are displayed in the terminal. The first and second options are Write EEPROM and Read EEPROM respectively. Write EEPROM will prompt the user for an address followed by a prompt for data, once the values are given the I2C controller will write the given data to the EEPROM at the address provided. The read EEPROM function will prompt the user for an address and the I2C controller will return the value at that address in the EEPROM. The next two options are fairly straight forward, they are Turn on Red LED and Turn off Red LED, the first will set the value of the value of the LED to 0x01 (on) and the second will set it to 0x00 (off). The final to options in the menu are DMA Test and SDRAM Test. The first will initialize a 1MB block of the SDRAM starting at the base address 0x10000000 to 0x000A0000 (NOTE: this value was arbitrarily chosen to avoid false positives that could result from using all ones or zeros, since they

could just be left over from running a SDRAM test). Once the 1MB block is initialized the DMA Test then copies the 1MB block from the base address to another 1MB section starting at 0x10100000, finally it is run once more copying the block to the next 1MB chunk of memory starting at 0x10200000. The last step of each copy checks the 1MB block to ensure that the values were copied correctly. The final option on the menu is the SDRAM test, which conducts a test by writing all ones to the SDRAM, writing all zeros to the SDRAM, and writing incrementing values to the SDRAM. After each step the SDRAM is checked to ensure that the values were written properly.

4 Results

The EEPROM, Red LED, and the Green LED functioned normally like in the last homework assignment (the code was the base I started this project from) and the results can be seen in Figure 2 below as well as in the video submitted with this document. The new additions to this project were the incrementing of the 7 segment display using a semaphore that is unlocked in the button ISR. I was able to get this to work without much issue, though I did figure out that it needed to be the second highest priority behind the blinking LED because if it was the highest it would stop the LED from blinking if the conditions are right. The SDRAM test is fairly simple write ones, then zeros, then incrementing values each time. When testing the SDRAM as you can see in Figure 3 the process of writing to and reading from the SDRAM takes about 35 seconds each (meaning 35sec to read and 35sec to write), so directly writing to the SDRAM is rather slow. The DMA copy turned out to be much faster, so fast that I couldn't really get a good time estimate because the copy seemed to happen within a single clock tick, I still dont really see how this happened since I am assuming it should take multiple clock ticks to write but I guess it is only 1MB so maybe not, because of this issue where the copy was starting and finishing on the same clock tick I decided to pick a random value to write to the block so that I could ensure that my copy was actually copying something. Turns out that it was and the value was properly written to the block of SDRAM as you can see in the terminal output in Figure 4.

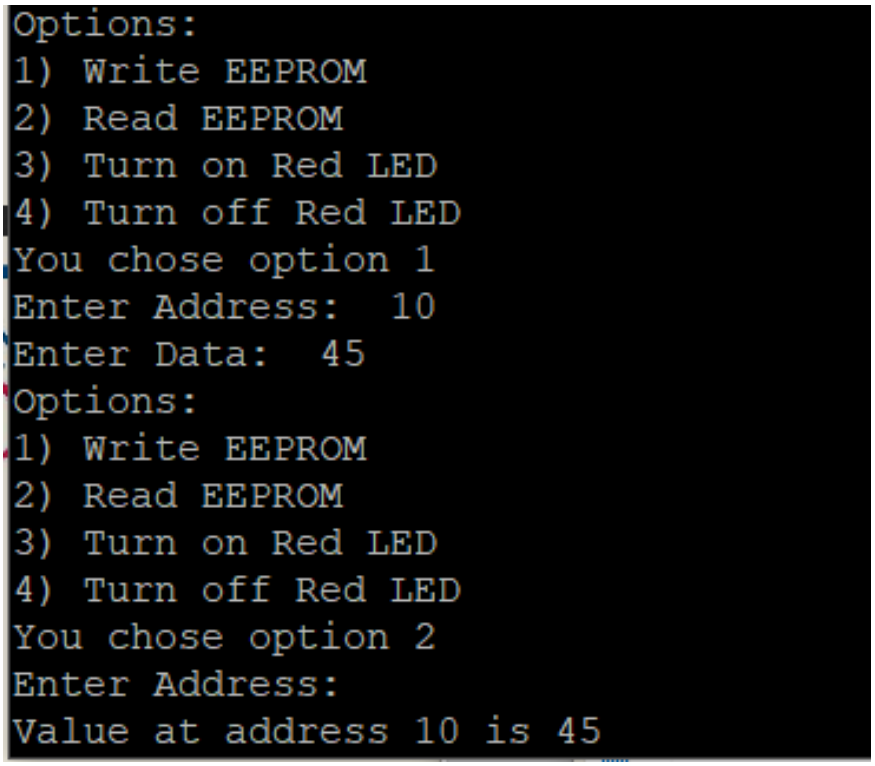

Figure (2) EEPROM Output

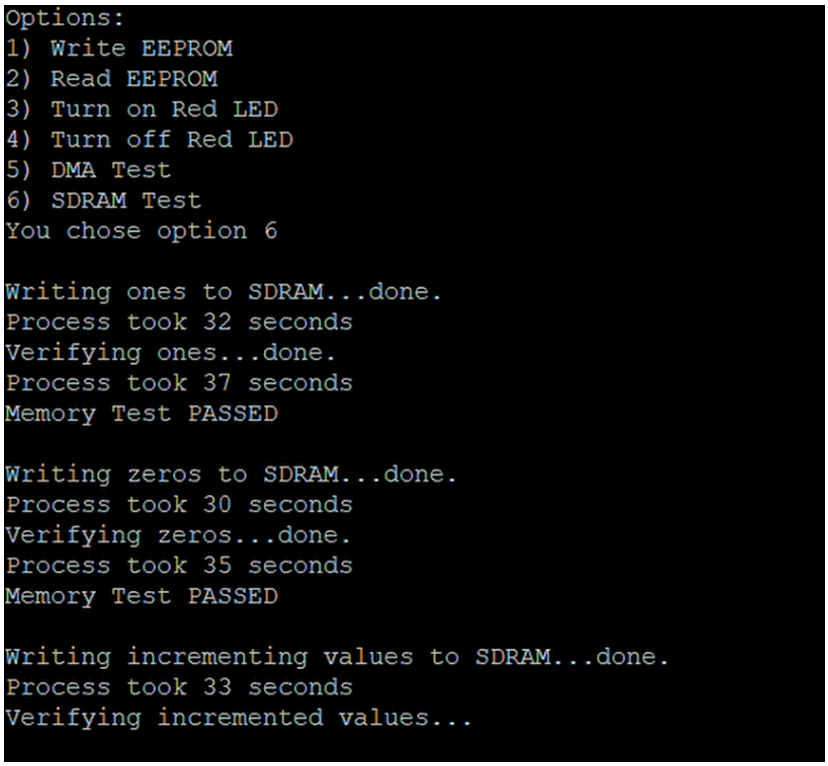

Figure (3) SDRAM Output

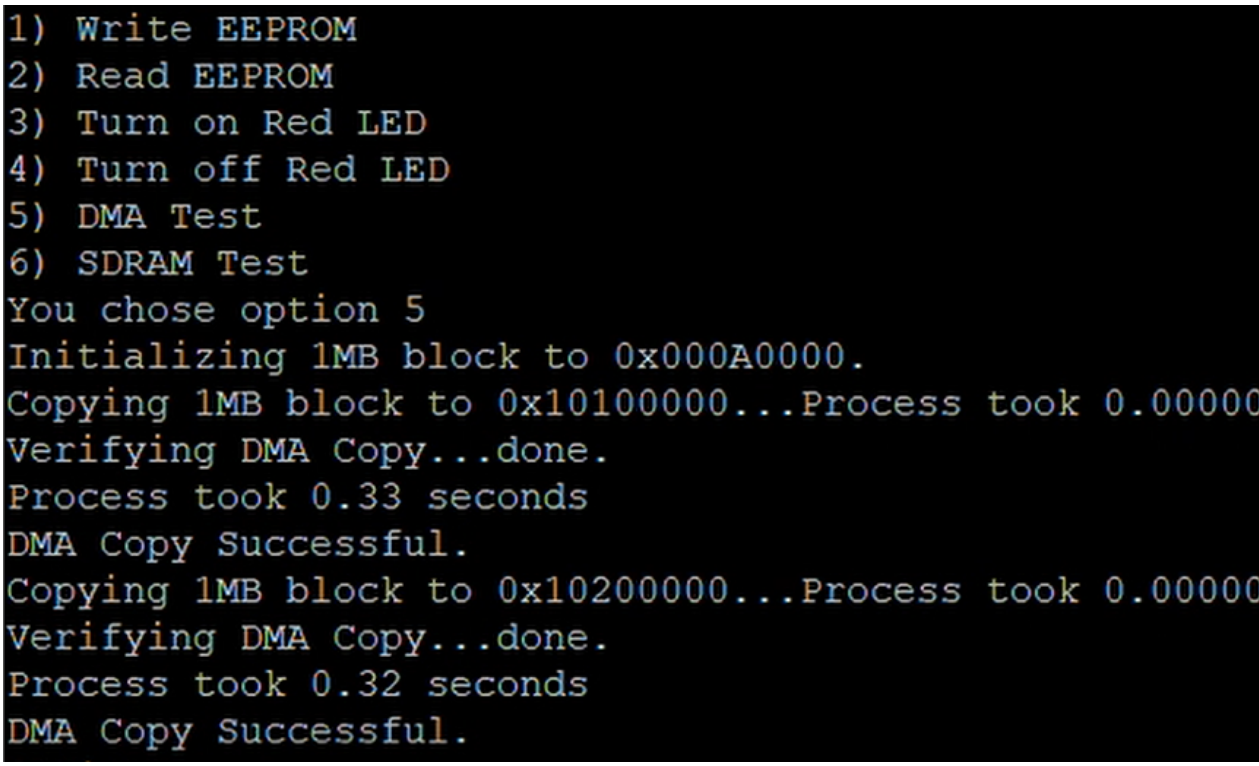

Figure (4) DMA Output

5 Conclusion

I found was able to complete the majority of this project without much trouble. The only portion that really gave me any issues was the DMA copy, I am not sure if it is just so fast that the 100MHz clock isn't fast enough to read it or what, but as I said in the results section I verified the copies each time and the data was there. I attempted to debug the timing quite a few times and every time I saw that the start and stop times I used to clock the functions were the same value every time when it came time to print them. I attempted to find a better way to clock the functions to no avail, most of the HighRes timing stuff is in C++ and this project is basically limited by the 100MHz clock when it comes to timing.

A
VHDL Code

```
-- ECE532 Final Exam
-- Zackary McClamma
-- 11-DEC-2019

library ieee;
  use ieee.std_logic_1164.all;

entity final_exam is
  port
  (
                i_clk          : in  std_logic := 'X'; -- clk
                i_reset_n      : in  std_logic := 'X'; -- reset_n
        i_uart_rxd       : in  std_logic := 'X'; -- rxd
        o_uart_txd       : out std_logic;         -- txd
        b_i2c_scl        : inout std_logic;
        b_i2c_sda        : inout std_logic;
        green_led        : out std_logic;
        red_led          : out std_logic;
                o_sdram_addr       : out   std_logic_vector(12 downto 0);
-- addr
                o_sdram_ba         : out   std_logic_vector(1 downto 0);
-- ba
                o_sdram_cas_n      : out   std_logic;
-- cas_n
                o_sdram_cke        : out   std_logic;
-- cke
                o_sdram_cs_n       : out   std_logic;
-- cs_n
                b_sdram_dq         : inout std_logic_vector(31 downto 0) := (oth
                o_sdram_dqm        : out   std_logic_vector(3 downto 0);
-- dqm
                o_sdram_ras_n      : out   std_logic;
-- ras_n
                o_sdram_we_n       : out   std_logic;
                o_sdram_clk        : out   std_logic;
                display       : out std_logic_vector(6 downto 0);
                button : in std_logic

  );
end final_exam;

architecture sch of final_exam is

  component final_cpu is
        port (
      clk_clk            : in  std_logic := 'X'; -- clk
```

```
        reset_reset_n      : in  std_logic := 'X'; -- reset_n
                  uart_rxd         : in  std_logic := 'X';
                  uart_txd     : out std_logic;
        i2c_sda_in       : in std_logic;
        i2c_scl_in       : in std_logic;
        i2c_sda_oe       : out std_logic;
        i2c_scl_oe       : out std_logic;
        green_led_export : out std_logic;
        red_led_export   : out std_logic;
                  sdram_addr       : out   std_logic_vector(12 downto 0);
                  sdram_ba         : out   std_logic_vector(1 downto 0);
                  sdram_cas_n      : out   std_logic;
                  sdram_cke        : out   std_logic;
                  sdram_cs_n       : out   std_logic;
                  sdram_dq         : inout std_logic_vector(31 downto 0) := (other
                  sdram_dqm        : out   std_logic_vector(3 downto 0);
                  sdram_ras_n      : out   std_logic;
                  sdram_we_n       : out   std_logic;
                  sdram_clk_clk    : out   std_logic;
                  button_export : in std_logic;
                  display_export       : out std_logic_vector(6 downto 0)
-- export

        );
  end component final_cpu;

  signal w_i2c_sda_in      : std_logic;
  signal w_i2c_scl_in      : std_logic;
  signal w_i2c_sda_oe      : std_logic;
  signal w_i2c_scl_oe      : std_logic;
  signal w_sdram_clk  : std_logic;

begin

    b_i2c_scl <= '0' when w_i2c_scl_oe = '1' else 'Z';
    b_i2c_sda <= '0' when w_i2c_sda_oe = '1' else 'Z';
    w_i2c_scl_in <= b_i2c_scl;
    w_i2c_sda_in <= b_i2c_sda;
         o_sdram_clk <= w_sdram_clk;

    u0 : component final_cpu
        port map (
      clk_clk            => i_clk,            --         clk.clk
      reset_reset_n      => i_reset_n,      --       reset.reset_n
                  uart_rxd     => i_uart_rxd,
                  uart_txd     => o_uart_txd,
      green_led_export => green_led,
      red_led_export => red_led,
      i2c_sda_in => w_i2c_sda_in,
      i2c_scl_in => w_i2c_scl_in,
```

```
        i2c_sda_oe => w_i2c_sda_oe,
        i2c_scl_oe => w_i2c_scl_oe,
                  sdram_addr       => o_sdram_addr,
                  sdram_ba         => o_sdram_ba,
                  sdram_cas_n      => o_sdram_cas_n,
                  sdram_cke        => o_sdram_cke,
                  sdram_cs_n       => o_sdram_cs_n,
                  sdram_dq         => b_sdram_dq,
                  sdram_dqm        => o_sdram_dqm,
                  sdram_ras_n      => o_sdram_ras_n,
                  sdram_we_n       => o_sdram_we_n,
                  sdram_clk_clk    => w_sdram_clk,
                  button_export    => button,
                  display_export => display

        );
end sch;
```

B C Code

B.1 Headers

```
/*
 * Name: Zackary McClamma
 * Course: ECE 532
 * Assignment: Final Exam
 * Date: 11 DEC 2019
 * File: hw5.h
 *
 * */


#ifndef FINAL_H_
#define FINAL_H_

#include <sys/alt_irq.h>
#include <sys/alt_timestamp.h>
#include "altera_avalon_pio_regs.h"
#include <time.h>
#include "alt_types.h"
#include "sys/alt_irq.h"

#define I2C_BASE 0x80000
#define UART_BASE 0x60000
#define RED_LED_BASE 0x90000
#define GREEN_LED_BASE 0xA0000
#define SDRAM_BASE 0x10000000
#define SDRAM_SIZE_WORDS 32000000
```

```
#define DISPLAY_BASE 0x000D0000
#define BUTTON_BASE 0x000E0000
#define DMA_BASE 0x000c0000


typedef struct str_timer_regs{
        unsigned int stats;
        unsigned int control;
        unsigned int periodl;
        unsigned int periodh;
        unsigned int snapl;
        unsigned int snaph;
}timer_regs;

typedef volatile struct{
        unsigned int uart_rxdata;
        unsigned int uart_txdata;
        unsigned int uart_status;
        unsigned int uart_control;
        unsigned int uart_divisor;
        unsigned int uart_eop;
}uart_reg;

typedef volatile struct{
        unsigned int i2c_tfr_cmd;
        unsigned int i2c_rxdata;
        unsigned int i2c_ctrl;
        unsigned int i2c_iser;
        unsigned int i2c_isr;
        unsigned int i2c_status;
        unsigned int i2c_tfr_cmd_fifo_lvl;
        unsigned int i2c_rx_data_fifo_lvl;
        unsigned int i2c_scl_low;
        unsigned int i2c_scl_high;
        unsigned int i2c_sda_hold;
}i2c_reg;

typedef volatile struct{
        unsigned int data;
        unsigned int dir;
        unsigned int intmask;
        unsigned int edge;
        unsigned int outset;
        unsigned int outclear;
}gpio_regs;

typedef volatile struct{
        unsigned int dma_status;
        unsigned int dma_read_addr;
        unsigned int dma_write_addr;
```

```
        unsigned int dma_length;
        unsigned int dma_res1;
        unsigned int dma_res2;
        unsigned int dma_control;
        unsigned int dma_res3;
}dma_reg;

int display_values[] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x00, 0
volatile int edge_capture = 0;
volatile int counter = 0;


void i2c_init(void);
void eep_write(unsigned short addr, unsigned char data);
unsigned char eep_read(unsigned short addr);
void printMenu(void);

void led_on(int* base);
void led_off(int* base);

void timer_isr(void);
int sdram_test_ones(void);
int sdram_test_zeros(void);
int sdram_test_increment(void);

void dma_start(unsigned int raddr, unsigned int waddr, unsigned int len);
void dma_wait();
void dma_check(unsigned int start_addr);
void dma_init_block();

void increment_display();
void init_button();
void button_isr(void* context, alt_u32 id);
#endif /* FINAL_H_ */
```

B.2 Source

```
/*
 * Name: Zackary McClamma
 * Course: ECE 532
 * Assignment: Final Exam
 * Date: 11 DEC 2019
 * File: main.c
 *
 * */
#include <stdio.h>
#include <system.h>
#include "final.h"
#include "includes.h"
```

```
OS_EVENT *sem;


/* Definition of Task Stacks */
#define   TASK_STACKSIZE       2048
OS_STK    task1_stk[TASK_STACKSIZE];
OS_STK    task2_stk[TASK_STACKSIZE];
OS_STK    task3_stk[TASK_STACKSIZE];

/* Definition of Task Priorities */

#define TASK1_PRIORITY      1
#define TASK2_PRIORITY      3
#define TASK3_PRIORITY          2

/* Toggles the Green LED every 500ms */
void task1(void* pdata)
{
        while(1){
          INT8U err;
          //OSMutexPend(tex,0,&err);
          unsigned int* gled = GREEN_LED_BASE;

                  if(*gled == 0x01){
                          led_off(GREEN_LED_BASE);
                  }
                  else{
                        led_on(GREEN_LED_BASE);
                  }
          //OSMutexPost(tex);
          OSTimeDlyHMSM(0, 0, 0, 500);
        }
}

/*Increment 7-Segment display*/
void task2(void* pdata)
{
        int* display = DISPLAY_BASE;
        *display = display_values[0];
        INT16U timeout = 0;
        INT8U err;
        while(1)
        {
                OSSemPend(sem, timeout, err);
                if (edge_capture !=0)
                {
                        increment_display();
                        edge_capture = 0;
                }
        }
```

```
}

/*Display Menu and facilitate user selections*/
void task3(void* pdata)
{
          INT8U err;
          i2c_init();
          while (1)
          {
                int start, stop, opt, addr, data, addrSet, dataSet = 0;
                unsigned int dma_length = 1024*1024;
                unsigned char read;
                //display menu
                printf("Options:\r\n");
                printf("1) Write EEPROM\r\n");
                printf("2) Read EEPROM\r\n");
                printf("3) Turn on Red LED\r\n");
                printf("4) Turn off Red LED\r\n");
                printf("5) DMA Test\r\n");
                printf("6) SDRAM Test\r\n");
                scanf("%d", &opt);
                fflush(stdin);
                printf("You chose option %d\r\n",opt);
                switch(opt){
                //Write EEPROM
                case 1:
                        addrSet = 0;
                        dataSet = 0;
                        while(addrSet == 0)
                        {
                                printf("Enter Address: ");
                                scanf("%d", &addr);
                                fflush(stdin);
                                printf(" %d\r\n", addr);

                                if((addr < 0) || (addr > 127))
                                {
                                        printf("Invalid input try again\r\n");
                                }
                                else
                                {
                                        addrSet = 1;
                                }
                        }
                        while(dataSet == 0)
                        {
                                printf("Enter Data: ");
                                scanf("%d", &data);
                                fflush(stdin);
                                printf(" %d\r\n", data);
```

```
                if (data > 255)
                {
                        printf("Invalid input try again\r\n");
                }
                else
                {
                        dataSet = 1;
                }
        }
        eep_write(addr, data);
        OSTimeDlyHMSM(0, 0, 1, 0);
        break;
//Read EEPROM
case 2:
        addrSet = 0;
        while(addrSet == 0)
        {
                printf("Enter Address: ");
                scanf("%d", &addr);
                fflush(stdin);
                printf("\r\n");
                if((addr < 0) || (addr > 127))
                {
                        printf("Invalid input try again\r\n");
                }
                else
                {
                        addrSet = 1;
                }
        }
        read = eep_read(addr);
        printf("Value at address %d is %d\r\n", addr, (int)read
        OSTimeDlyHMSM(0, 0, 1, 0);
        break;
//Turn Red LED on
case 3:
        led_on(RED_LED_BASE);
        OSTimeDlyHMSM(0, 0, 1, 0);
        break;
//Turn Red LED off
case 4:
        led_off(RED_LED_BASE);
        OSTimeDlyHMSM(0, 0, 1, 0);
        break;
//DMA Test
case 5:


        printf("Initializing 1MB block to 0x000A0000.\r\n");
```

```
                        dma_init_block();

                        printf("Copying 1MB block to 0x10100000...");
                        start = OSTimeGet();
                        dma_start(SDRAM_BASE, 0x10100000, dma_length);
                        stop = OSTimeGet();
                        printf("Process took %f seconds\r\n", ((float)stop-(flo
                        dma_check(0x10100000);

                        printf("Copying 1MB block to 0x10200000...");
                        start = OSTimeGet();
                        dma_start(SDRAM_BASE, 0x10200000, dma_length);
                        //dma_wait();
                        stop = OSTimeGet();
                        printf("Process took %f seconds\r\n", ((float)stop-(flo
                        dma_check(0x10200000);
                        break;
                //SDRAM Test
                case 6:
                        sdram_test_ones();
                        sdram_test_zeros();
                        sdram_test_increment();
                        break;
                default:
                        printf("Invalid option please try again.\r\n");
                        break;

                }
             // OSTimeDlyHMSM(0, 0, 1, 0);
                //OSMutexPost(tex);
                //OSTimeDlyHMSM(0, 0, 0, 500);
          }
}
/* The main function creates two task and starts multi-tasking */
int main(void)
{

  INT16U initSem = 1;
  sem = OSSemCreate(initSem);
  int tst;
  init_button();
  tst = alt_ic_irq_enable(0, 5);

  //Check if Interrupt was enabled successfully
  if (tst<0) printf("\nFailed to enable interrupt, enable returned %d\r\n", tst
  else printf("\n Interrupt Enabled.\r\n");
  //Initialize display to 0
  int* display = DISPLAY_BASE;
   *display = display_values[0];
```

```
  OSTaskCreateExt(task1,
                  NULL,
                  (void *)&task1_stk[TASK_STACKSIZE-1],
                  TASK1_PRIORITY,
                  TASK1_PRIORITY,
                  task1_stk,
                  TASK_STACKSIZE,
                  NULL,
                  0);


  OSTaskCreateExt(task2,
                  NULL,
                  (void *)&task2_stk[TASK_STACKSIZE-1],
                  TASK2_PRIORITY,
                  TASK2_PRIORITY,
                  task2_stk,
                  TASK_STACKSIZE,
                  NULL,
                  0);


  OSTaskCreateExt(task3,
                  NULL,
                  (void *)&task3_stk[TASK_STACKSIZE-1],
                  TASK3_PRIORITY,
                  TASK3_PRIORITY,
                  task3_stk,
                  TASK_STACKSIZE,
                  NULL,
                  0);

  OSStart();
  return 0;
}

void led_on(int* base){
//      INT8U err;
        //OSMutexPend(tex,0,&err);
        *base = 0x01;
        //OSMutexPost(tex);
        return;
}

void led_off(int* base){
//      INT8U err;
        //OSMutexPend(tex,0,&err);
        *base = 0x00;
        return;
}
```

```
void i2c_init(void){
        i2c_reg *reg = I2C_0_BASE;

        reg->i2c_scl_low = 1000;
        reg->i2c_scl_high = 1000;
        reg->i2c_sda_hold = 500;
        reg->i2c_tfr_cmd_fifo_lvl = 8;
        reg->i2c_rx_data_fifo_lvl = 8;
}

void eep_write(unsigned short addr, unsigned char data){
        i2c_reg* reg = I2C_0_BASE;
//      INT8U err;
        unsigned char addr_low = addr >> 8;
        unsigned char addr_high = addr & 0x00FF;
        //OSMutexPend(tex,0,&err);
        reg->i2c_ctrl = 0x1;
        reg->i2c_tfr_cmd = 0x2A0;
        reg->i2c_tfr_cmd = addr_high;
        reg->i2c_tfr_cmd = addr_low;
        reg->i2c_tfr_cmd = data | 0x100;

        while(reg->i2c_status != 0);
        reg->i2c_ctrl = 0x0;
        //OSMutexPost(tex);
        OSTimeDlyHMSM(0, 0, 0, 5);
}

unsigned char eep_read(unsigned short addr){
        i2c_reg* reg = I2C_0_BASE;
        unsigned char addr_low = addr >> 8;
        unsigned char addr_high = addr & 0x00FF;
        unsigned char data;
        unsigned char tempData = 0;
        //OSMutexPend(tex,0,&err);

        reg->i2c_ctrl = 0x1;
        reg->i2c_tfr_cmd = 0x2A0;
        reg->i2c_tfr_cmd = addr_high;
        reg->i2c_tfr_cmd = addr_low;
        reg->i2c_tfr_cmd = 0x2A1;
        reg->i2c_tfr_cmd = tempData | 0x100;


        while(reg->i2c_status != 0);
        data = reg->i2c_rxdata;
        reg->i2c_ctrl = 0x0;
        //OSMutexPost(tex);
        return data;
```

```
}

int sdram_test_ones(void){
        unsigned int idx;
        unsigned int *sdram=SDRAM_BASE;
        unsigned int start, stop;

        printf("\r\nWriting ones to SDRAM...");
        start = OSTimeGet();
        for(idx = 0;idx<SDRAM_SIZE_WORDS;idx++)
        {
                sdram[idx] = 0xFFFFFFFF;
        }
        stop=OSTimeGet();
        printf("done.\r\n");
        printf("Process took %u seconds\r\n", (stop-start)/100);

        printf("Verifying ones...");
        start=OSTimeGet();
        for(idx=0;idx<SDRAM_SIZE_WORDS;idx++)
        {
                if(sdram[idx] != 0xFFFFFFFF)
                {
                        stop=OSTimeGet();
                        printf("MEMORY TEST FAILED!\r\n");
                        printf("Process took %u seconds \r\n",(stop-start)/100)
                        return 0;
                }
        }
        stop=OSTimeGet();
        printf("done.\r\n");
        printf("Process took %u seconds\r\n", (stop-start)/100);
        printf("Memory Test PASSED\r\n");
        return 0;
}

int sdram_test_zeros(void){
        unsigned int idx;
        unsigned int *sdram=SDRAM_BASE;
        unsigned int start, stop;

        printf("\r\nWriting zeros to SDRAM...");
        start = OSTimeGet();
        for(idx = 0;idx<SDRAM_SIZE_WORDS;idx++)
        {
                sdram[idx] = 0x00000000;
        }
        stop=OSTimeGet();
        printf("done.\r\n");
        printf("Process took %u seconds\r\n", (stop-start)/100);
```

```
        printf("Verifying zeros...");
        start=OSTimeGet();
        for(idx=0;idx<SDRAM_SIZE_WORDS;idx++)
        {
                if(sdram[idx] != 0x00000000)
                {
                        stop=OSTimeGet();
                        printf("MEMORY TEST FAILED!\r\n");
                        printf("Process took %u seconds \r\n",(stop-start)/100)
                        return 0;
                }
        }
        stop=OSTimeGet();
        printf("done.\r\n");
        printf("Process took %u seconds\r\n", (stop-start)/100);
        printf("Memory Test PASSED\r\n");
        return 0;
}

int sdram_test_increment(void)
{
        unsigned int idx;
        unsigned int *sdram=SDRAM_BASE;
        unsigned int start, stop;

        printf("\r\nWriting incrementing values to SDRAM...");
        start = OSTimeGet();
        for(idx = 0;idx<SDRAM_SIZE_WORDS;idx++)
        {
                sdram[idx] = idx;
        }
        stop=OSTimeGet();
        printf("done.\r\n");
        printf("Process took %u seconds\r\n", (stop-start)/100);

        printf("Verifying incremented values...");
        start=OSTimeGet();
        for(idx=0;idx<SDRAM_SIZE_WORDS;idx++)
        {
                if(sdram[idx] != idx)
                {
                        stop=OSTimeGet();
                        printf("MEMORY TEST FAILED!\r\n");
                        printf("Process took %u seconds \r\n",(stop-start)/100)
                        return 0;
                }
        }
        stop=OSTimeGet();
        printf("done.\r\n");
```

```
        printf("Process took %u seconds\r\n", (stop-start)/100);
        printf("Memory Test PASSED\r\n");
        return 0;
}

void button_isr(void* context, alt_u32 id){
        //Set context to edge_capture_ptr value (global value registered with I
        volatile int* edge_capture_ptr = (volatile int*) context;
        //Read edge capture register of Button
        *edge_capture_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_BASE);
        OSSemPost(sem);
        //Write 1 to edge capture register
        IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_BASE, 0x1);

        return;
}

void init_button()
{
        //Type cast edge_capture variable so it can be passed to register funct
        void* edge_capture_ptr = (void*) &edge_capture;

        //Write 1 to IRQ Mask
        IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON_BASE, 0x1);
        //Write 0 to edge capture of Button
        IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_BASE, 0x0);

        //Register Interrupt
        alt_ic_isr_register(0,
                                                5,
                                                (void*)button_isr,
                                                edge_capture_ptr,
                                                0x0);
}

void increment_display()
{
        int* display = DISPLAY_BASE;
        counter++;
        if (counter > 9)
        {
                counter = 0;
        }
        *display = display_values[counter];
}

void dma_start(unsigned int raddr, unsigned int waddr, unsigned int len)
{
        dma_reg *dmaPtr = DMA_BASE;
        dmaPtr->dma_read_addr = raddr;
```

```
        dmaPtr->dma_write_addr = waddr;
        dmaPtr->dma_length = 2^20;
        dma_wait();

}

void dma_wait(void)
{
        dma_reg *dmaPtr = DMA_BASE;
        dmaPtr->dma_control = 0x31;
        while((dmaPtr->dma_status >> 0) & 1);
        dmaPtr->dma_status &= 0x0;
        //dmaPtr->dma_control = 0x00000000;

}

void dma_check(unsigned int start_addr)
{
        unsigned int idx = 0;
        unsigned int *sdram=SDRAM_BASE;
        unsigned int stop, start;
        printf("Verifying DMA Copy...");
        start = OSTimeGet();
                for(idx=0;idx<((1024*1024)/4);idx++)
                        {
                                if(sdram[start_addr + idx] != 0x000A0000)
                                {
                                        stop = OSTimeGet();
                                        printf("Process took %g seconds\r\n", (
                                        printf("DMA Copy FAILED!\r\n");
                                        return;
                                }
                        }
                stop = OSTimeGet();
                printf("done.\r\n");
                printf("Process took %g seconds\r\n", ((float)stop-(float)start
                printf("DMA Copy Successful.\r\n");
}

void dma_init_block()
{
        unsigned int idx;
        unsigned int *sdram=SDRAM_BASE;
        for(idx = 0;idx<((1024*1024)/4);idx++)
        {
                sdram[idx] = 0x000A0000;
        }
}
```