Final Project

SDRAM and DMA Controllers

Connor Prikkel

ECE 532: Embedded Systems

22 January 2026

University of Dayton

Contents

1 Introduction

In this final project, the student develops a comphrehensive embedded system that expands on the interactive menu developed for the previous assingment. The menu is complete with five subtasks that involve resources and peripherals utilized throughout the semester, including EEPROM, HEX displays, green LEDs, and message queues. However, the menu also adds functionality for two new ways of accessing and manipulating memory in 'SDRAM' and 'Direct Memory Access' (DMA). SDRAM is an external, large area of memory that can be accessed via an interface similar to EEPROM. DMA is an efficient type of data transfer that moves data between the Analog-to-Digital Controller (ADC), SDRAM, and Digital-to-Analog Controller (DAC). Both SDRAM and DMA give more options to manipulate and move data within an embedded system, each with their own upsides and downsides. Four other tasks alongside the menu are all implemented within μC/OS-II.

2 Theory of Operation

2.1 Embedded Hardware

Within Qsys, a Nios-II/e processor is added and connected to on-chip RAM. The RS-232 interface, a push button, eighteen switches, a timer, eight seven-segment displays, the I2C interface, eighteen red LEDs, the EEPROM interface, and nine green LEDs are all connected to the processor following the same procedure of previous assignments. To better show the wiring and functionality of this embedded system, its representation has been split into four images and summarized in Figure 1, Figure 2, Figure 3, and Figure 4 below.

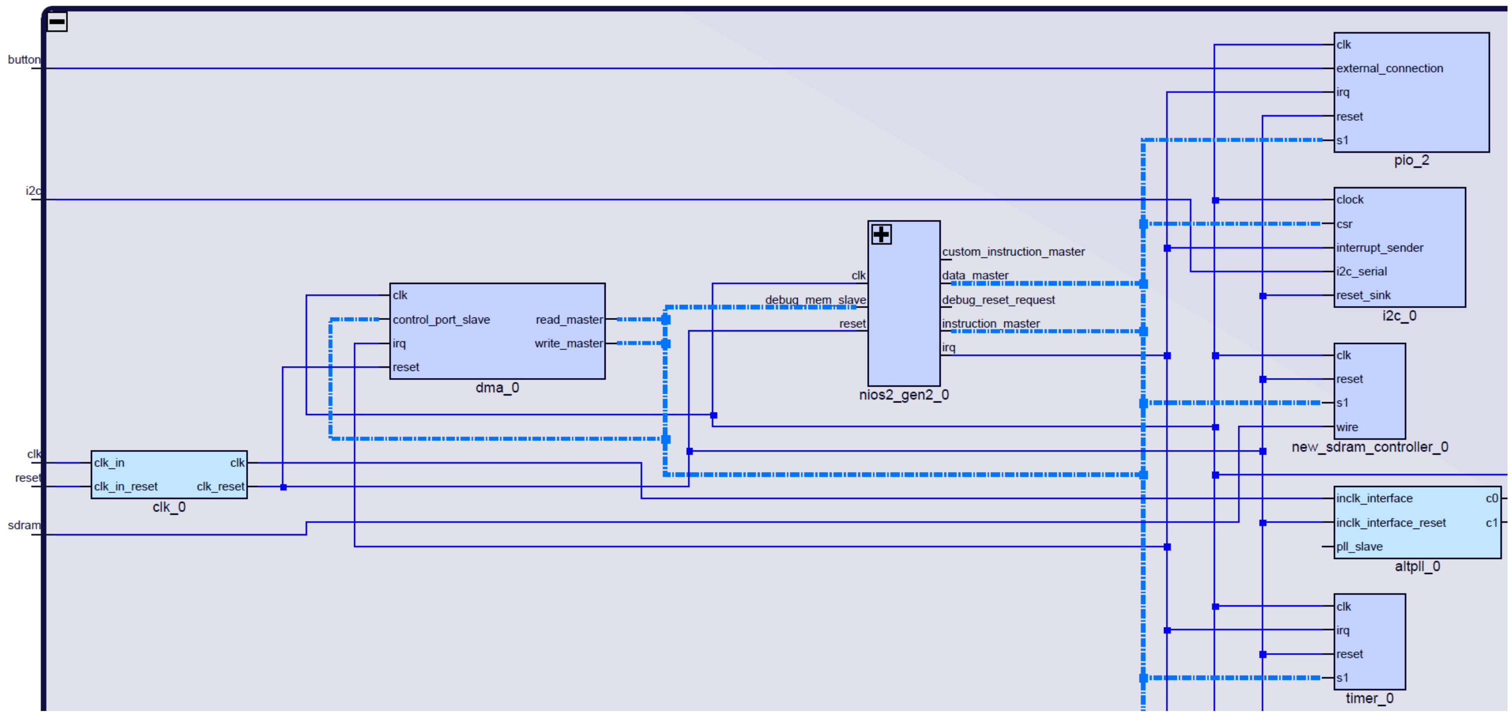


Figure 1: Embedded System Block Diagram A

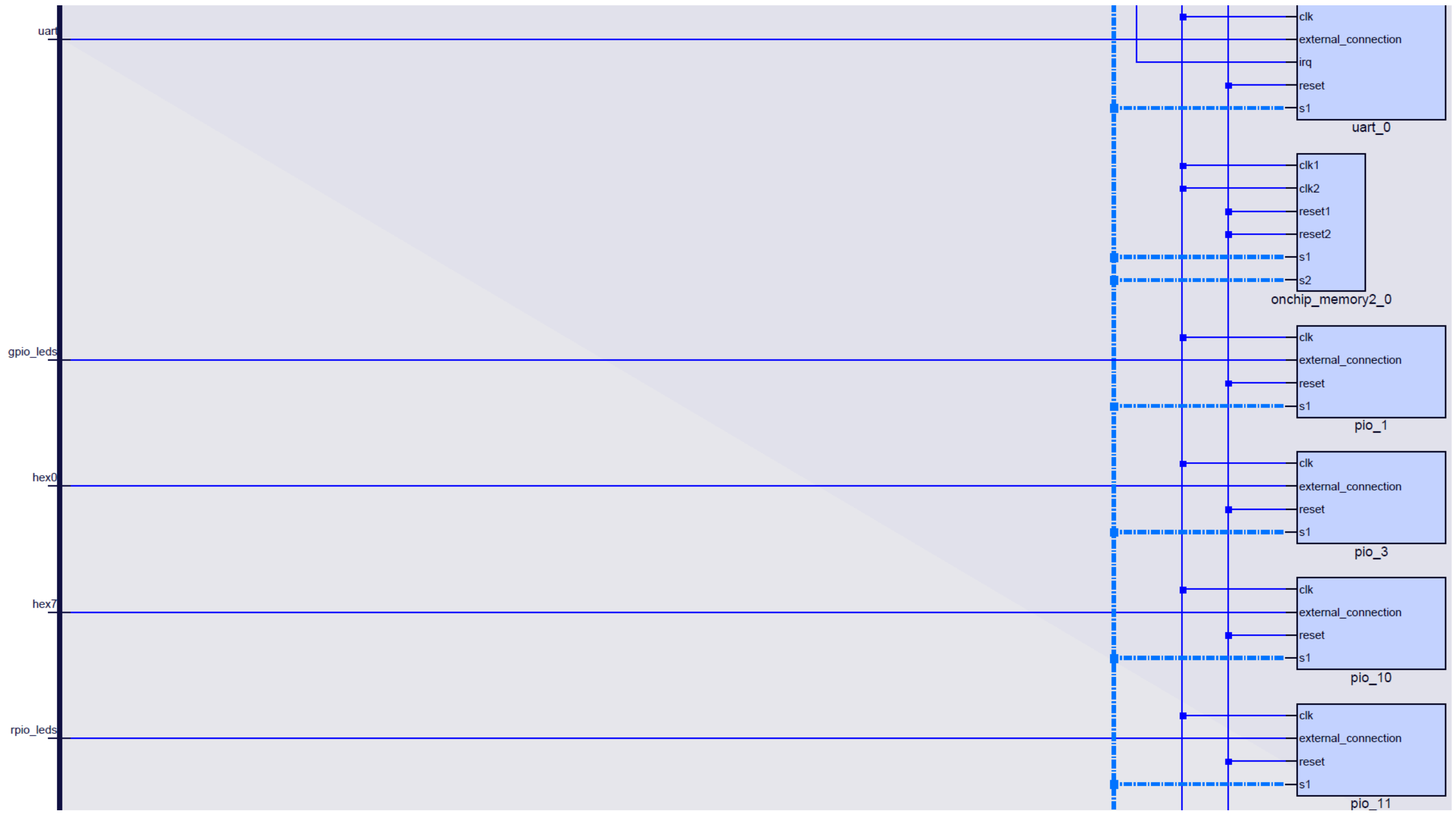


Figure 2: Embedded System Block Diagram B

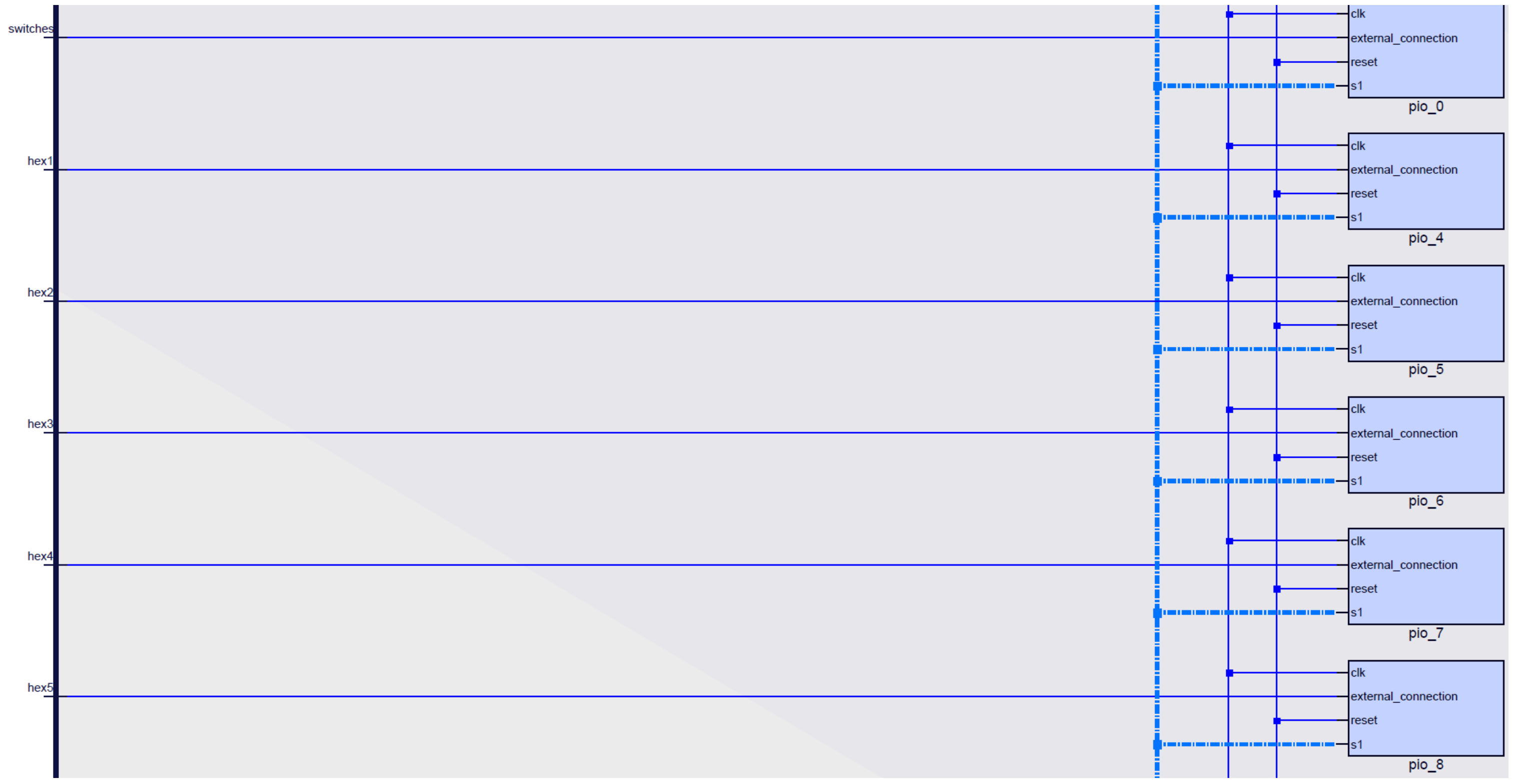


Figure 3: Embedded System Block Diagram C

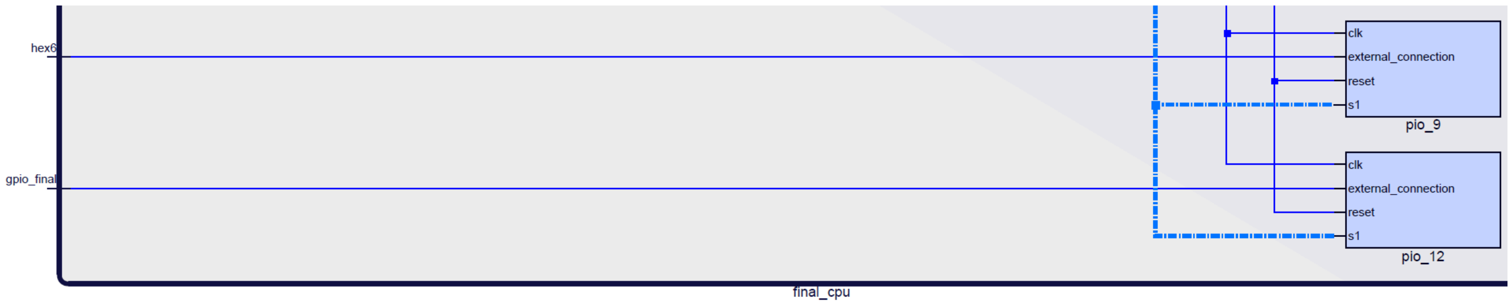


Figure 4: Embedded System Block Diagram D

The DMA controller and its signals are connected to the processor and all other I/Os as seen in Figure 1 to give them each direct access to memory. From here, the EEPROM is defined and connected to the I2C interface as illustrated in Figure 5 below.

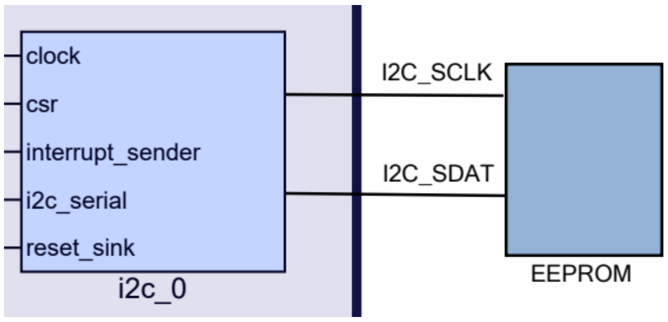


Figure 5: EEPROM Device Connection

The SDRAM interface requires some extra steps to be added to the system. First, an 'ALTPLL', which increases the clock speed to 100 MHz for the SDRAM chips, is added and used to drive the clocks of all other components. Next, an SDRAM controller is connected to this 'ALTPLL' and the external SDRAM through the signals visible below in Figure 6.

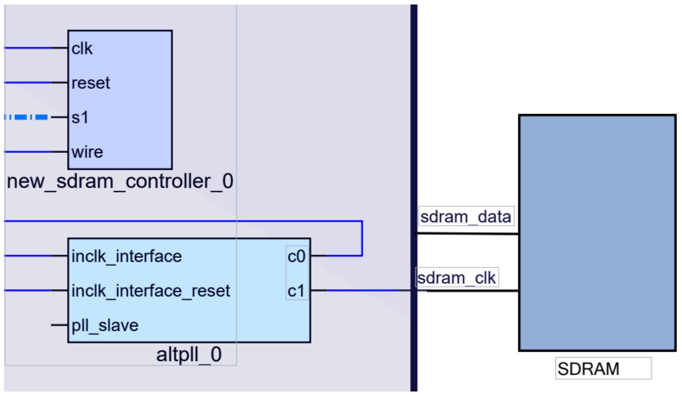


Figure 6: SDRAM Device Connection

The red LEDs are assigned the same behavior as before; they turn on if their corresponding switch is flipped or if set to 1 within the 'Knight Rider' task. The 4:7 decoder process is re-utilized from the previous assignment. All inputs and outputs are mapped to pins on the DE-2 board.

One other hardware adjustment to note is the addition of a timing constraint to the project. This timing constraint can be seen below in Section 4.1.3 and is added to prevent the board from stalling due to potential disparities in clock timing.

2.2 Embedded Software

The software for this assignment is split into five main tasks: a menu task, a 'Knight Rider' task, a task that updates HEX0 when a button is pushed, a task that gets a message from the menu regarding a green LED pattern to display, and a task that blinks the solitary green LED once per second.

The tasks controlling the Knight Rider pattern and the green LED pattern displayed function exactly the same as the previous assignment. The menu task now gives the user a choice between updating seven-segment displays, writing or reading to the EEPROM, writing, reading, or testing the SDRAM, posting a selected green LED pattern, and a DMA test. The seven-segment display, EEPROM, and green LED pattern subtasks remain unchanged from the previous assignment.

An interrupt is setup and enabled to work with a semaphore, or a special variable that effectively shares resources between processes, to increment the HEX0 display whenever the user pushes the button. When the task that handles this is ran, it pends on a semaphore and puts the task into a 'waiting' state. When the 'button_isr()' is called, the semaphore is posted and the task that is waiting is 'woken up' and ran.

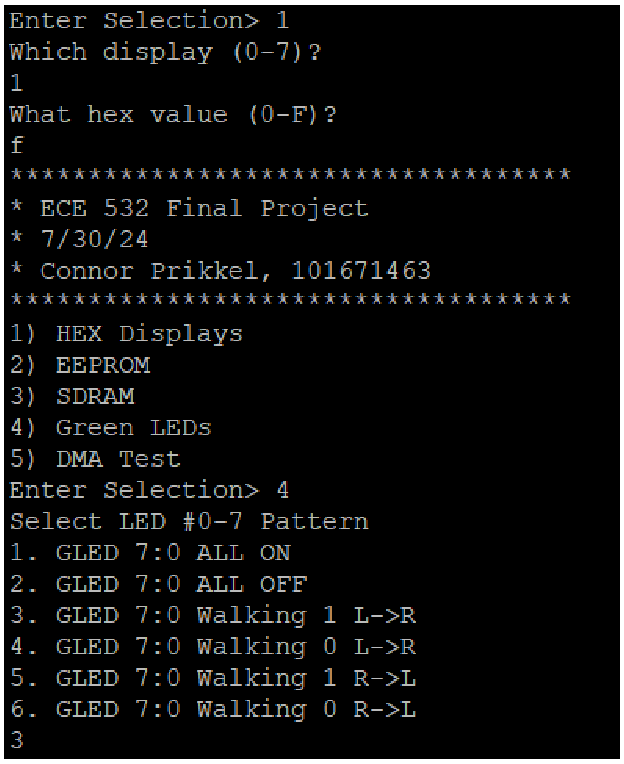
The solitary green LED blinking involves simply writing one to the data of the LED before delaying for a fraction of a second to appear as if blinking before zero is written and an argument of one second is passed to 'OSTimeDlyHMSM()'.

In the menu's third subtask, the user is prompted for an address and/or data to write and read from the SDRAM. A pointer is declared to that address and data is either written to or gotten from that pointer. The three SDRAM tests are slightly more complex to implement in code. In each test, a for loop is used to iterate over all addresses, with each value written dependent on the respective test being run. After all addresses are written to, each address is iterated over again to compare the values present against the intended value, and a 'Success!' or 'FAIL!' message is printed depending on the results. Before and after each of these steps, the 'OSTimeGet()' function is used to calculate how long each process took.

Finally, a new register map is defined for the DMA test. Ones are written to an arbitrary block in SDRAM. The contents of this block are then copied to two other blocks using the CPU and DMA, respectively. The time that it took for each process to copy the data is recorded and compared. In the case of DMA, the parameters for a transfer are defined and the 'status' register is polled until the transfer is complete.

3 Results

The code is built and ran, and each possible menu state is described below. Arbitrary user input to update a HEX display and choose a green LED pattern can be seen after selecting subtasks one and four, as shown below in Figure 7.



```
Enter Selection> 1
Which display (0-7)?
1
What hex value (0-F)?
f
*************************************
* ECE 532 Final Project
* 7/30/24
* Connor Prikkel, 101671463
*************************************
1) HEX Displays
2) EEPROM
3) SDRAM
4) Green LEDs
5) DMA Test
Enter Selection> 4
Select LED #0-7 Pattern
1. GLED 7:0 ALL ON
2. GLED 7:0 ALL OFF
3. GLED 7:0 Walking 1 L->R
4. GLED 7:0 Walking 0 L->R
5. GLED 7:0 Walking 1 R->L
6. GLED 7:0 Walking 0 R->L
3
```

Figure 7: Menu Subtasks 1 and 4

In Figure 7, the user chooses to update the first seven-segment display with a value of 'F' and to run the walking one pattern on the green LEDs. These choices are shown on the DE-2 board in Figure 8 below.

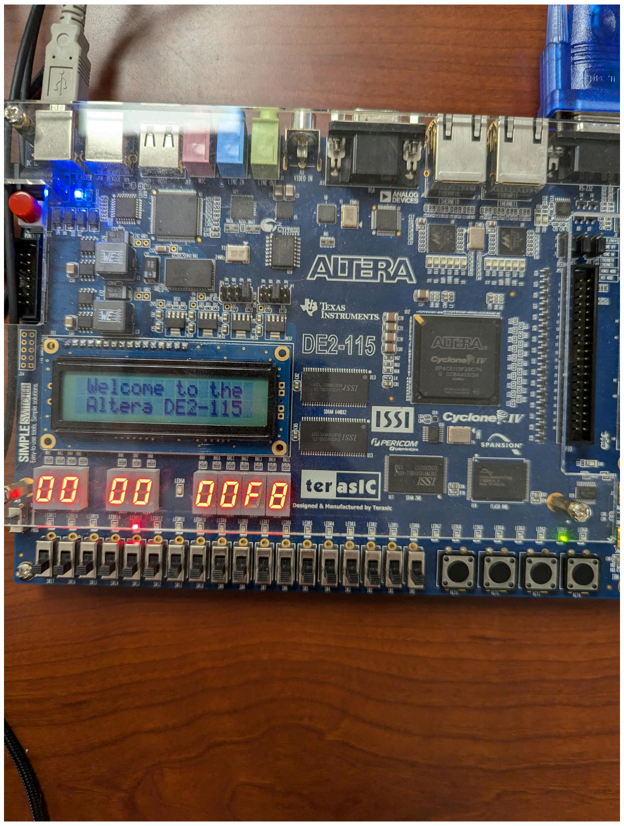
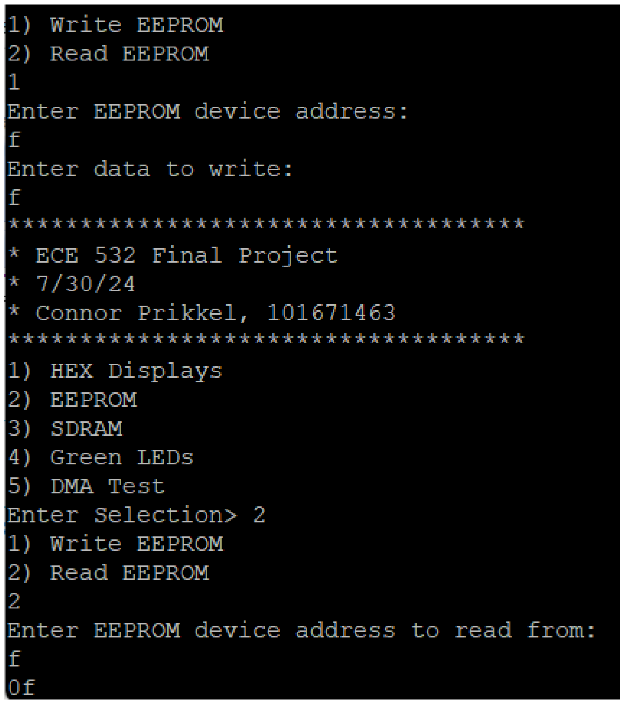


Figure 8: Updated Hex Display and Green LED Pattern

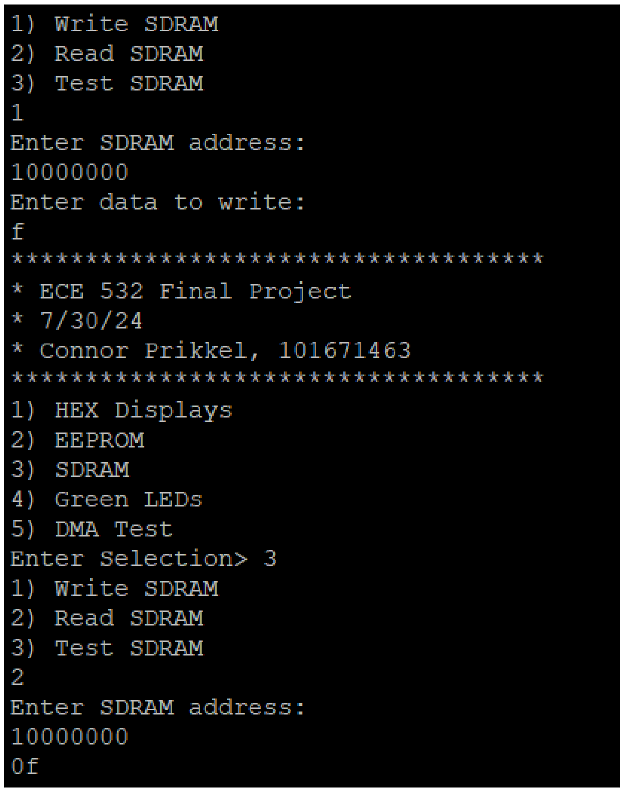
Figure 9 below reveals an example of the user writing an arbitrary value to an arbitrary HEX address and then reading that value from that address using EEPROM.



```
1) Write EEPROM
2) Read EEPROM
1
Enter EEPROM device address:
f
Enter data to write:
f
************************************
* ECE 532 Final Project
* 7/30/24
* Connor Prikkel, 101671463
************************************
1) HEX Displays
2) EEPROM
3) SDRAM
4) Green LEDs
5) DMA Test
Enter Selection> 2
1) Write EEPROM
2) Read EEPROM
2
Enter EEPROM device address to read from:
f
0f
```

Figure 9: Menu Subtask 2: EEPROM

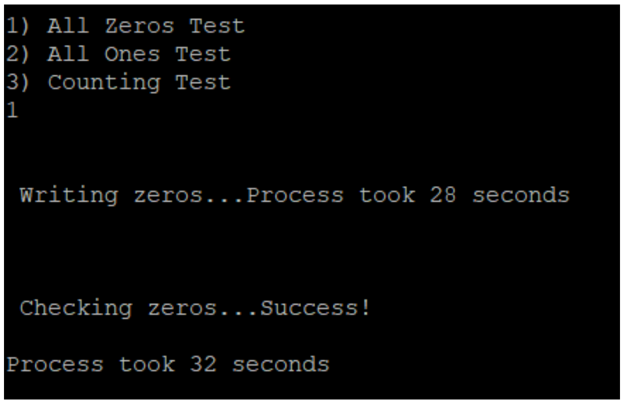
The address is read from successfully and displayed in HEX, as indicated by the '0f' printed to the display. Next, an arbitrary SDRAM address is written to and read from as revealed below in Figure 10.



```
1) Write SDRAM
2) Read SDRAM
3) Test SDRAM
1
Enter SDRAM address:
10000000
Enter data to write:
f
*************************************
* ECE 532 Final Project
* 7/30/24
* Connor Prikkel, 101671463
*************************************
1) HEX Displays
2) EEPROM
3) SDRAM
4) Green LEDs
5) DMA Test
Enter Selection> 3
1) Write SDRAM
2) Read SDRAM
3) Test SDRAM
2
Enter SDRAM address:
10000000
0f
```

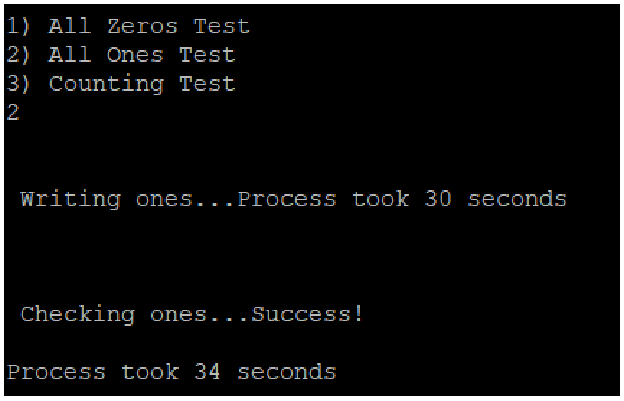
Figure 10: Menu Subtask 3: SDRAM Writing and Reading

Each of the three SDRAM tests are ran successfully as summarized below in Figure 11, Figure 12, and Figure 13. The time taken for writing and checking within each test is also given.



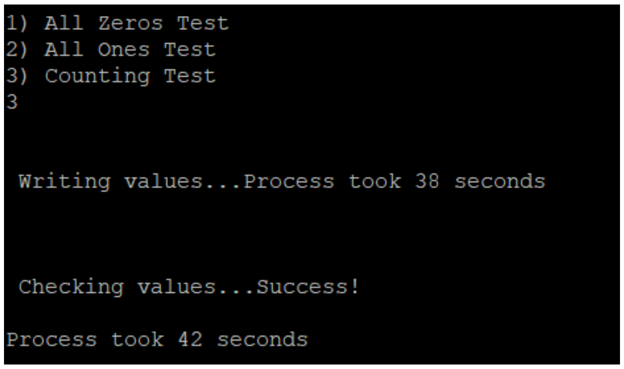
```
1) All Zeros Test
2) All Ones Test
3) Counting Test
1


 Writing zeros...Process took 28 seconds



 Checking zeros...Success!

Process took 32 seconds
```

Figure 11: Menu Subtask 3: SDRAM Test 1



```
1) All Zeros Test
2) All Ones Test
3) Counting Test
2


 Writing ones...Process took 30 seconds



 Checking ones...Success!

Process took 34 seconds
```

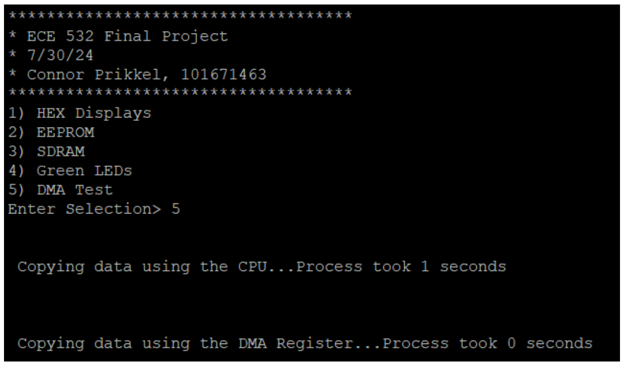
Figure 12: Menu Subtask 3: SDRAM Test 2



```
1) All Zeros Test
2) All Ones Test
3) Counting Test
3


 Writing values...Process took 38 seconds



 Checking values...Success!

Process took 42 seconds
```

Figure 13: Menu Subtask 3: SDRAM Test 3

The output of the DMA test is shown below in Figure 14. As expected, copying data via the DMA proved faster than the CPU.



```
************************************
* ECE 532 Final Project
* 7/30/24
* Connor Prikkel, 101671463
************************************
1) HEX Displays
2) EEPROM
3) SDRAM
4) Green LEDs
5) DMA Test
Enter Selection> 5


 Copying data using the CPU...Process took 1 seconds



 Copying data using the DMA Register...Process took 0 seconds
```

Figure 14: Menu Subtask 5: DMA Test

4 Conclusion

The culmination of this course in an embedded system featuring many 'external' devices, connected peripherals, and tasks that each contain their own timing constraints is developed. I learned how more memory can be added to the system with SDRAM, alleviating possible memory constraint issues. I also learned the power of DMA, as it frees up the CPU to be used for other tasks. Unlike ECE 501 where I was confused about how the VHDL code I was writing affected the hardware, ECE 532 was much more clear and filled in many of those gaps in my understanding.

The most challenging aspect of this course was bridging the gap between hardware and software, and needing to truly understand how an embedded system should be wired, which has never been a large focus in other classes I've taken up until this point. As someone whose built my own PC twice, the most rewarding aspect of this class was getting to implement SDRAM, as I've always wanted to understand how external memory works and interfaces with the central processor.

4.1 VHDL: Hardware Instantiation

4.1.1 'final.vhd'

```
-- ECE 532 Final Project Top Level
-- Connor Prikkel
-- 8/1/24

library ieee;
use ieee.std_logic_1164.all;

entity final is
  port
  (
   i_clk      : in    std_logic;
   i_rst_n    : in    std_logic;
  i_button    : in    std_logic;
  i_switches : in    std_logic_vector(17 downto 0);
  o_hex0     : out   std_logic_vector(6 downto 0);
  o_hex1     : out   std_logic_vector(6 downto 0);
  o_hex2     : out   std_logic_vector(6 downto 0);
  o_hex3     : out   std_logic_vector(6 downto 0);
  o_hex4     : out   std_logic_vector(6 downto 0);
  o_hex5     : out   std_logic_vector(6 downto 0);
  o_hex6     : out   std_logic_vector(6 downto 0);
  o_hex7     : out   std_logic_vector(6 downto 0);
  o_gpio_leds : out  std_logic_vector(7 downto 0);
  o_rpio_leds : out std_logic_vector(17 downto 0);
  -- for final green LED
  o_gpio_final : out std_logic;

  -- I2C pins (bidirectional)
  b_i2c_scl  : inout std_logic;
  b_i2c_sda  : inout std_logic;
  i_uart_rxd : in    std_logic;
  o_uart_txd : out   std_logic;

  -- SDRAM pins
  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);                    -- dq
   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

);
end final;

architecture sch of final is

--instantiate decoder for each hex display
component decoder is
 port (
  decode        : out std_logic_vector(6 downto 0);
  code          : in std_logic_vector(3 downto 0);
  clk         : in std_logic
);
end component decoder;

component final_cpu is
   port (
      clk_clk          : in  std_logic;
      reset_reset_n    : in  std_logic;
    button_export    : in  std_logic;
    gpio_final_export : out std_logic;
      switches_export  : in  std_logic_vector(17 downto 0);
    gpio_leds_export : out std_logic_vector(7 downto 0);
    rpio_leds_export : out std_logic_vector(17 downto 0);
      hex0_export      : out std_logic_vector(3 downto 0);
      hex1_export      : out std_logic_vector(3 downto 0);
      hex2_export      : out std_logic_vector(3 downto 0);
      hex3_export      : out std_logic_vector(3 downto 0);
      hex4_export      : out std_logic_vector(3 downto 0);
      hex5_export      : out std_logic_vector(3 downto 0);
      hex6_export      : out std_logic_vector(3 downto 0);
      hex7_export      : out std_logic_vector(3 downto 0);
    -- needs default val to avoid error
    uart_rxd         : in  std_logic      := 'X';
      uart_txd         : out std_logic;
    i2c_sda_in       : in std_logic        := 'X';
    i2c_scl_in       : in std_logic        := 'X';
    i2c_sda_oe       : out std_logic;
    i2c_scl_oe       : out std_logic;
    -- SDRAM pins
    sdram_addr       : out   std_logic_vector(12 downto 0);                    -- addr
    sdram_ba         : out   std_logic_vector(1 downto 0);                     -- ba
    sdram_cas_n      : out   std_logic;                                        -- cas_n
    sdram_cke        : out   std_logic;                                        -- cke
    sdram_cs_n       : out   std_logic;                                         -- cs_n
    sdram_dq         : inout std_logic_vector(31 downto 0) := (others => 'X'); -- dq
    sdram_dqm        : out   std_logic_vector(3 downto 0);                      -- dqm
    sdram_ras_n      : out   std_logic;                                         -- ras_n
    sdram_we_n       : out   std_logic;                                         -- we_n
    sdram_clk_clk    : out   std_logic
    );
    end component final_cpu;

   signal hex0_val    :  std_logic_vector(3 downto 0);
   signal hex1_val    :  std_logic_vector(3 downto 0);
   signal hex2_val    :  std_logic_vector(3 downto 0);
   signal hex3_val    :  std_logic_vector(3 downto 0);
   signal hex4_val    :  std_logic_vector(3 downto 0);
   signal hex5_val    :  std_logic_vector(3 downto 0);
   signal hex6_val    :  std_logic_vector(3 downto 0);
   signal hex7_val    :  std_logic_vector(3 downto 0);

   signal cpu_clk   :  std_logic;
   signal w_gpio_final : std_logic;
   signal w_gpio_leds  : std_logic_vector(7 downto 0);
   signal w_rpio_leds  : std_logic_vector(17 downto 0);

   --wiring signals hook up to controller
   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;




begin

  -- for eighteen red LEDs, OR w/ switch input so that switches can allow knight rider
  o_rpio_leds <= w_rpio_leds or i_switches;
  o_gpio_leds <= w_gpio_leds;
  o_gpio_final <= w_gpio_final;

  -- map signal clk to i_clk
  cpu_clk <= i_clk;

  -- bidirectional signals
  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;

  u0 : component final_cpu
  port map
  (
   clk_clk         => i_clk,
   reset_reset_n   => i_rst_n,
   button_export   => i_button,
  switches_export  => i_switches,
  gpio_leds_export => w_gpio_leds,
  rpio_leds_export => w_rpio_leds,
  gpio_final_export => w_gpio_final,
  --map exports to decoder's signals
  hex0_export   => hex0_val,
  hex1_export   => hex1_val,
  hex2_export   => hex2_val,
  hex3_export   => hex3_val,
  hex4_export   => hex4_val,
  hex5_export   => hex5_val,
  hex6_export   => hex6_val,
  hex7_export   => hex7_val,
  uart_rxd      => i_uart_rxd,
  uart_txd      => o_uart_txd,
  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
  sdram_addr       => o_sdram_addr,       --     sdram.addr
   sdram_ba         => o_sdram_ba,          --           .ba
   sdram_cas_n      => o_sdram_cas_n,       --           .cas_n
   sdram_cke        => o_sdram_cke,         --           .cke
   sdram_cs_n       => o_sdram_cs_n,        --           .cs_n
   sdram_dq         => b_sdram_dq,          --           .dq
   sdram_dqm        => o_sdram_dqm,         --           .dqm
   sdram_ras_n      => o_sdram_ras_n,       --           .ras_n
   sdram_we_n       => o_sdram_we_n,        --           .we_n
   sdram_clk_clk    => o_sdram_clk          -- sdram_clk.clk
  );

  --map from decoder.vhd to defs
  dec0 : component decoder
  port map (code => hex0_val, decode => o_hex0, clk => cpu_clk);
  dec1 : component decoder
  port map (code => hex1_val, decode => o_hex1, clk => cpu_clk);
  dec2 : component decoder
  port map (code => hex2_val, decode => o_hex2, clk => cpu_clk);
  dec3 : component decoder
  port map (code => hex3_val, decode => o_hex3, clk => cpu_clk);
  dec4 : component decoder
  port map (code => hex4_val, decode => o_hex4, clk => cpu_clk);
  dec5 : component decoder
  port map (code => hex5_val, decode => o_hex5, clk => cpu_clk);
  dec6: component decoder
  port map (code => hex6_val, decode => o_hex6, clk => cpu_clk);
  dec7 : component decoder
  port map (code => hex7_val, decode => o_hex7, clk => cpu_clk);


end sch;
```

4.1.2 'decoder.vhd'

```
-- ECE 532 Final Project Decoder Process
-- Connor Prikkel
-- 8/1/24

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decoder is
port (
  decode        : out std_logic_vector(6 downto 0);
  code          : in std_logic_vector(3 downto 0);
  clk         : in std_logic
);
end decoder;

architecture behavior of decoder is

begin

  decode_proc: process(clk, code)
   begin
      --update 7-segment only on rising_edge of clk
        if(rising_edge(clk)) then

        --matches hex switch statement in C
            case code is

          when "0000" =>

            --display switch statement val, let decoder handle binary -> hex
            --decode <= not std_logic_vector(x"3F");
            --each bit of output corresponds to segment-encoding 6,5,4,3,2,1,0
            --ex: for 0, only segment 6 is off, or "1"
            decode <= "1000000";
          when "0001" =>
            decode <= "1111001";
          when "0010" =>
            decode <= "0100100";
          when "0011" =>
            decode <= "0110000";
          when "0100" =>
            decode <= "0011001";
          when "0101" =>
            decode <= "0010010";
          when "0110" =>
            decode <= "0000010";
          when "0111" =>
            decode <= "1111000";
          when "1000" =>
            decode <= "0000000";
          when "1001" =>
            decode <= "0011000";
          when "1010" =>
            decode <= "0001000";
          when "1011" =>
            decode <= "0000011";
          when "1100" =>
            decode <= "1000110";
          when "1101" =>
            decode <= "0100001";
          when "1110" =>
            decode <= "0000110";
          when "1111" =>
            decode <= "0001110";
        end case;
        end if;
    end process decode_proc;
end behavior;
```

4.1.3 ‘SD1.sdc’

```
create_clock -period 20.0 -name i_clk [get_ports {i_clk}]
```

4.2 Software: C Code

4.2.1 'final.c'

```
/*
 * Connor Prikkel
 * ECE 532 Final Project
 * 7/30/24
 */

#include <stdio.h>
#include "includes.h"
#include "final.h"

// define global var to track green LED state
int gled_state;

// def global var to track HEX0 val
static signed int hex0_val = 0;

// set ptr to addr
eep_regs *ep = I2C_BASE;

// def other ptrs
gpio_regs *btn = BUTTON_BASE;
unsigned int *sdram = SDRAM_BASE;

/* 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];
OS_STK    task4_stk[TASK_STACKSIZE];
OS_STK    task5_stk[TASK_STACKSIZE];

/* definition of task priorities */
// assigned by freq, i.e. 1/period, frequent tasks should have numerically lower prio
// tasks cannot have the same prio
#define TASK1_PRIORITY      1
#define TASK2_PRIORITY      2
#define TASK3_PRIORITY      3
#define TASK4_PRIORITY      4
#define TASK5_PRIORITY      5

// define message queue event and param.
#define MSG_SIZE 16
void *msg_buf[MSG_SIZE];
OS_EVENT *msg;

// define semaphore event and param
OS_EVENT *sema;
INT16U cnt;

// menu task
void task1(void* pdata)
{

  // create a local buffer for msgs for subtask 3
  char txbuf[10];
  // task loop
  while (1)
  {
    // dummy var for user input
    int x;

    // print out main menus, allow users to select
    printf("************************************\n");
    printf("* ECE 532 Final Project \n");
    printf("* 7/30/24 \n");
    printf("* Connor Prikkel, 101671463\n");
    printf("************************************\n");
    printf("1) HEX Displays\n");
    printf("2) EEPROM\n");
    printf("3) SDRAM\n");
    printf("4) Green LEDs\n");
    printf("5) DMA Test\n");
    printf("Enter Selection> ");

    // read an int in from the user
    scanf("%d", &x);

    // write seperate fns. for submenu tasks
    // if subtask 1 selected
    if(x == 1){
      int hex_num;
      int hex_data;

      printf("Which display (0-7)?\n");
      scanf("%d", &hex_num);
      printf("What hex value (0-F)?\n");
      scanf("%x", &hex_data);

      // update correct hex display
      switch (hex_num) {

      case 0: hex_display(hex_data, HEX0_BASE);
              // keep track of hex0 for task 3
              hex0_val = hex_data;
              break;
      case 1: hex_display(hex_data, HEX1_BASE);
            break;
      case 2: hex_display(hex_data, HEX2_BASE);
            break;
      case 3: hex_display(hex_data, HEX3_BASE);
            break;
      case 4: hex_display(hex_data, HEX4_BASE);
            break;
      case 5: hex_display(hex_data, HEX5_BASE);
            break;
      case 6: hex_display(hex_data, HEX6_BASE);
            break;
      case 7: hex_display(hex_data, HEX7_BASE);
            break;

      }

    }

    // if subtask 2 selected
```

```
    if(x == 2) {

      // dummy var for EEPROM
      unsigned short addr;
      unsigned char data;

      int eep_int;
      printf("1) Write EEPROM\n");
      printf("2) Read EEPROM\n");

      scanf("%d", &eep_int);

      // setup i2c
      i2c_init();

      if(eep_int == 1){

        // ask user for addr + data in hex
        printf("Enter EEPROM device address: \n");
        scanf("%hx", &addr);

        printf("Enter data to write: \n");
        scanf("%hhx", &data);

        // write to eeprom
        eep_write(addr, data);
      }
      if(eep_int == 2){

        // ask user for addr + data in hex
        printf("Enter EEPROM device address to read from: \n");
        scanf("%hx", &addr);

        // read from eeprom
        unsigned char read_data;
        read_data = eep_read(addr);

        // print to display in hex
        printf("%02x\n", read_data);
      }


    }

    // if subtask 3 selected
    if(x == 3) {

      // dummy var for SDRAM
      unsigned short s_addr;
      unsigned char s_data;
      unsigned int *sd_ptr;

      // var for SDRAM tests
      int test_int;
      unsigned int idx;
      unsigned int t1, t2;

      int sdram_int;
      printf("1) Write SDRAM\n");
      printf("2) Read SDRAM\n");
      printf("3) Test SDRAM\n");

      scanf("%d", &sdram_int);

      switch(sdram_int) {
          case 1:

          // ask user for addr + data in hex
          printf("Enter SDRAM address: \n");
          scanf("%hx", &s_addr);

          // declare a ptr to that addr
          sd_ptr = (unsigned int*) s_addr;

          printf("Enter data to write: \n");
          scanf("%hhx", &s_data);

          // write to that ptr (dereference)
          *sd_ptr = s_data;

          break;

          case 2:
            printf("Enter SDRAM address: \n");
            scanf("%hx", &s_addr);

            // declare a ptr to that addr
            sd_ptr = (unsigned int*) s_addr;

            // get data from that addr
            s_data = *sd_ptr;

            // print to display in hex
            printf("%02x\n", s_data);
            break;


          case 3:
            printf("1) All Zeros Test\n");
            printf("2) All Ones Test\n");
            printf("3) Counting Test\n");
            scanf("%d", &test_int);

            switch (test_int) {

            // zeros test
            case 1:
              t1 = OSTimeGet();

              // set all mem locations to 0
              printf("\r\n Writing zeros...");
              for(idx = 0; idx < SDRAM_SIZE_WORDS; idx++) {
                sdram[idx] = 0;
              }
              t2 = OSTimeGet();
              printf("Process took %u seconds \r\n", (t2-t1)/100);

              // check locations
              printf("\r\n Checking zeros...");
              t1 = OSTimeGet();
```

```
  for(idx = 0; idx < SDRAM_SIZE_WORDS; idx++) {
    if(sdram[idx] != 0){
      t2 = OSTimeGet();
      printf("FAIL!\r\n");
      printf("Process took %u seconds\r\n", (t2-t1)/100);
      break;
    }
  }

  t2 = OSTimeGet();
  printf("Success! \r\n");
  printf("Process took %u seconds\r\n", (t2-t1)/100);
  break;

// ones test
case 2:
t1 = OSTimeGet();
// set all mem locations to 1
printf("\r\n Writing ones...");
for(idx=0; idx < SDRAM_SIZE_WORDS; idx++) {
  sdram[idx] = 1;
}
t2 = OSTimeGet();
printf("Process took %u seconds \r\n", (t2-t1)/100);

// check locations
printf("\r\n Checking ones...");
t1 = OSTimeGet();

for(idx=0; idx < SDRAM_SIZE_WORDS; idx++) {
  if(sdram[idx] != 1){
     t2 = OSTimeGet();
     printf("FAIL!\r\n");
     printf("Process took %u seconds\r\n", (t2-t1)/100);
     break;
  }
}

t2 = OSTimeGet();
printf("Success! \r\n");
printf("Process took %u seconds\r\n", (t2-t1)/100);
break;

// counting test
case 3:
t1 = OSTimeGet();

// each 32-bit word increments by 1
int incrementer = 0;
printf("\r\n Writing values...");
for(idx=0; idx < SDRAM_SIZE_WORDS; idx++) {
  sdram[idx] = incrementer;
  incrementer++;
}
t2 = OSTimeGet();
printf("Process took %u seconds \r\n", (t2-t1)/100);

// reset incrementer var for test
incrementer = 0;
```

```
        // check locations
        printf("\r\n Checking values...");
        t1 = OSTimeGet();

        for(idx=0; idx < SDRAM_SIZE_WORDS; idx++) {
          if(sdram[idx] != incrementer){
            t2 = OSTimeGet();
            printf("FAIL!\r\n");
            printf("Process took %u seconds\r\n", (t2-t1)/100);
            break;
          }
          incrementer++;
        }

        t2 = OSTimeGet();
        printf("Success! \r\n");
        printf("Process took %u seconds\r\n", (t2-t1)/100);
        break;

        }

  }


}

// if subtask 4 selected
if(x == 4) {

  int gled_pat;

  // get user input about desired LED pattern
  printf("Select LED #0-7 Pattern\n");
  printf("1. GLED 7:0 ALL ON\n");
  printf("2. GLED 7:0 ALL OFF\n");
  printf("3. GLED 7:0 Walking 1 L->R\n");
  printf("4. GLED 7:0 Walking 0 L->R\n");
  printf("5. GLED 7:0 Walking 1 R->L\n");
  printf("6. GLED 7:0 Walking 0 R->L\n");

  scanf("%d", &gled_pat);

  // print int to str
  sprintf(txbuf, "%d\n", gled_pat);

  // send an OSQ message to task 3
  OSQPost(msg, txbuf);

}

// if subtask 5 selected
if (x == 5) {

  // initialize block #1 with a constant value 1
  unsigned int idx;
  for (idx = SDRAM_BASE; idx < BLOCK_TWO; idx++){
    sdram[idx] = 1;
  }
```

```
      // copy block #1 -> #2 using CPU
      copy_data(SDRAM_BASE, BLOCK_TWO, 0x100000);

      // copy block #1 -> #3 using DMA
      dma_copy(SDRAM_BASE, BLOCK_THREE, 0x100000);

    }
    OSTimeDlyHMSM(0, 0, 0, 500);
  }
}

// "knight" rider task
void task2(void* pdata)
{

  while (1)
  {
    static unsigned int pattern = 0b100000000000000000;
    static unsigned int shift_right = 1;

    // ptr to red LEDs
      gpio_regs *re = RED_LEDS;

    // if not shifted all the way to the right
    if(shift_right){
      pattern = pattern >> 1;
    } else {
      pattern = pattern << 1;
    }

    re->data = pattern;

    // switch directions and start shifting the other way
    if (pattern == 0b100000000000000000) {
      shift_right = 1;
    }

    if (pattern == 0b000000000000000001) {
      shift_right = 0;
    }


    OSTimeDlyHMSM(0, 0, 0, 150);
    }
}

// hex0 task
void task3(void* pdata) {

  INT8U err;

  while(1){

    // pend on a semaphore- "waiting" state
    OSSemPend(sema, 0, &err);

    // increment hex display
    hex_display(hex0_val++, HEX0_BASE);

    // only 15 num poss
    if(hex0_val > 15){
      hex0_val = 0;
    }

  }

  // no delay as waiting on semaphore

}

// acts on message from task #1, updates at 4 Hz rate
void task4(void* pdata)
{
  INT8U err;

  // ptr for received msg
  char *rxbuf;

  // ptr to green LEDs
  gpio_regs *gr = GREEN_LEDS;

  while (1)
  {
    int gled_num;

    // wait on msg, don't wait forever (only 250 ms)
    rxbuf = OSQPend(msg, 25, &err);

    // check if msg timeout
    if(rxbuf != NULL) {

      // convert from char -> int based on ASCII codes (start at 48)
      // Ref: https://stackoverflow.com/questions/5029840/convert-char-to-int-in-c-and-c
      gled_num = rxbuf[0] - '0';
    }

    // ensure current pattern keeps running
    else {
      gled_num = gled_state;
    }

    // update global var in case timeout
    gled_state = gled_num;

    // led pattern pos
    static unsigned char off_pattern = 0b00000000;
    static unsigned char walking_one = 0b10000000;
    static unsigned char walking_one_right = 0b00000001;

    // switch statement for different green LED patterns
    switch (gled_num) {

    // all on
    case 1:
      gr->data = ~(off_pattern);
      break;
    // all off
    case 2:
      gr->data = off_pattern;
      break;
    // walking 1 L->R
    case 3:
      gr->data = walking_one;
      walking_one = walking_one >> 1;

      // handle wrap-around
      if(!walking_one) {
        walking_one = 0b10000000;
      }
      break;

    // walking 0 L->R
    case 4:
      gr->data = ~(walking_one);
      walking_one = walking_one >> 1;

      // handle wrap-around
      if(!walking_one) {
        walking_one = 0b10000000;
      }
      break;

    // walking 1 R->L
    case 5:
      gr->data = walking_one_right;
      walking_one_right = walking_one_right << 1;

      // handle wrap-around
      if(!walking_one_right) {
        walking_one_right = 0b000000001;
      }
      break;

    // walking 0 R->L
    case 6:
      gr->data = ~(walking_one_right);
      walking_one_right = walking_one_right << 1;

      // handle wrap-around
      if(!walking_one_right) {
        walking_one_right = 0b000000001;
      }
      break;

    }

    OSTimeDlyHMSM(0, 0, 0, 250);
    }
}

// green LED #8 blinks once per sec
void task5(void *pdata){

  while(1){

    // ptr to green LED #8
    gpio_regs *gr_f = FINAL_GREEN;

    gr_f->data = 1;
    // delay to appear as blinking
    usleep(10000);
    gr_f->data = 0;

    // delay 1 second
    OSTimeDlyHMSM(0, 0, 1, 0);
  }
}

/* create ten tasks using uC/OS-II */
int main(void)
{

  // create semaphore
  sema = OSSemCreate(cnt);

  // setup button interrupt for task 3
  alt_ic_isr_register(0, BUTTON_IRQ, button_isr, NULL, NULL);

  alt_ic_irq_enable(0, BUTTON_IRQ);

  // write to PIO_1 intmask reg for btn interrupt
  btn->intmask = 0x01;

  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);

  OSTaskCreateExt(task4,
                    NULL,
                    (void *)&task4_stk[TASK_STACKSIZE-1],
                    TASK4_PRIORITY,
                    TASK4_PRIORITY,
                    task4_stk,
                    TASK_STACKSIZE,
                    NULL,
                    0);

  OSTaskCreateExt(task5,
                    NULL,
                    (void *)&task5_stk[TASK_STACKSIZE-1],
                    TASK5_PRIORITY,
                    TASK5_PRIORITY,
                    task5_stk,
                    TASK_STACKSIZE,
                    NULL,
                    0);

  // create msg queue
  msg = OSQCreate(msg_buf, MSG_SIZE);

  OSStart();
  return 0;
}

// fn. to update hex display
void hex_display(unsigned int val, unsigned int HEX_BASE) {

    // define a general pointer to limit # of case statements required
    gpio_regs *hex = (gpio_regs*)HEX_BASE;

    // decoder should handle conversion from binary -> hex
    hex->data = val;
}

// handle setup for EEPROM device
void i2c_init(void){

  // disable i2c bus in case left on from previous run
  ep->ctrl = 0x00;

  // set setup/hold times for eeprom
  ep->sclhigh = 1000;
  ep->scllow = 1000;
  ep->sdahold = 500;

  // enable i2c bus
  ep->ctrl = 0x01;
}

void eep_write(unsigned short addr, unsigned char data){

  // set device addr
  ep->tfrcmd = 0xA0 | (1 << 9);

  // set high and low byte addresses
  ep->tfrcmd = (addr >> 8);
  ep->tfrcmd = (addr & 0xFF);

  // get data byte, write until stop bit
  ep->tfrcmd = data | (0x100);

  // poll status register until write complete
  while(ep->status) {
    // delay for at least 6 ms to allow EEPROM time to finish
    OSTimeDlyHMSM(0, 0, 0, 10);
  }

}
unsigned char eep_read(unsigned short addr){

  // dummy var to store read data
  unsigned char data;

  // set device addr to dummy write
  // must write before you can read
  ep->tfrcmd = 0xA0 | (1 << 9);

  // same high and low byte addr as write
  ep->tfrcmd = (addr >> 8);
  ep->tfrcmd = (addr & 0xFF);

  // set device addr to read
  ep->tfrcmd = 0xA1 | (1 << 9);

  // wait until stop bit
  ep->tfrcmd = 0x100;

  // poll status until read complete
  while(ep->status){

    OSTimeDlyHMSM(0, 0, 0, 10);
  }

  // read data
  data = ep->rxdata;

  return data;
}

// copy data using DMA controller
// block #1 -> #3
void dma_copy(unsigned short read_addr, unsigned short write_addr, unsigned int num_bytes){

  // initialize dma controller
  unsigned int t1, t2;
  dma_regs *dma = (dma_regs*)DMA_BASE;

  // write to control register
  dma->control = 0x8C;

  // def param for transfer
  dma->readaddress = read_addr;
  dma->writeaddress = write_addr;
  dma->length = num_bytes;

  printf("\r\n Copying data using the DMA Register...");
  t1 = OSTimeGet();

  // poll the status register, when status = "1" its done
  while(dma->status == 0x0){

    // wait until transfer complete
  }
  t2 = OSTimeGet();
  printf("Process took %u seconds \r\n", (t2-t1)/100);

  // clear the DONE bit
  dma->status = 0x0;

}

// copy data using CPU
// following format from slides
// copy data from block #1 -> #2
void copy_data(unsigned int *src, unsigned int *dst, unsigned int nwords) {
  unsigned int t1, t2;
  unsigned int k;


  printf("\r\n Copying data using the CPU...");
  t1 = OSTimeGet();

  for(k = 0; k < nwords; k++) {
    dst[k] = src[k];
  }

  t2 = OSTimeGet();
  printf("Process took %u seconds \r\n", (t2-t1)/100);
}

void button_isr(void) {

  // post semaphore - "wake up" task waiting
  OSSemPost(sema);

  // clear edge-capture reg
  btn->edge = 0x01;

}
```

4.2.2 'final.h'

```
// ECE 532 - Header for Final Project
// Connor Prikkel
// 7/30/24

// locations in memory
#define UART_BASE 0x50000
#define SWITCHES_BASE 0x60000
#define GREEN_LEDS 0x70000
#define BUTTON_BASE 0x80000
#define HEX0_BASE 0x90000
#define HEX1_BASE 0xA0000
#define HEX2_BASE 0xB0000
#define HEX3_BASE 0xC0000
#define HEX4_BASE 0xD0000
#define HEX5_BASE 0xE0000
#define HEX6_BASE 0xF0000
#define HEX7_BASE 0x1F0000
#define TIMER_BASE 0x2F0000
#define I2C_BASE 0x3F0000
#define RED_LEDS 0x4F0000
#define FINAL_GREEN 0x6F0000
```

```
// this is also the location of block 1 for sdram
#define SDRAM_BASE 0x10000000
#define BLOCK_TWO 0x10100000
#define BLOCK_THREE 0x10200000
#define DMA_BASE 0x7F0000

// interrupt defs
#define BUTTON_IRQ 1
#define TIMER_IRQ 2

// defs
#define TEN_MS_TICKS 500000
#define HUNDRED_MS_TICKS 5000000
#define TWO_HUNDRED_FIFTY_MS_TICKS 12500000
#define ONE_SEC_TICKS 50000000
// the sdram is size 2^(25)
#define SDRAM_SIZE_WORDS 33554432

// fn. prototypes
void task0(void* pdata);
void task1(void* pdata);
void task2(void* pdata);
void task3(void* pdata);
void task4(void* pdata);
void task5(void* pdata);
void button_isr(void);
void timer_isr(void);
void dma_copy(unsigned short read_addr, unsigned short write_addr, unsigned int num_bytes);
void copy_data(unsigned int *src, unsigned int *dst, unsigned int nwords);
void hex_display(unsigned int val, unsigned int HEX_BASE);
void i2c_init(void);
void eep_write(unsigned short addr, unsigned char data);
unsigned char eep_read(unsigned short addr);

// structs
typedef struct gr{
  unsigned int data;
  unsigned int dir;
  unsigned int intmask;
  unsigned int edge;
  unsigned int outset;
  unsigned int outclear;
} gpio_regs;

typedef struct ti{
  unsigned int status;
  unsigned int control;
  unsigned int periodl;
  unsigned int periodh;
  unsigned int snapl;
  unsigned int snaph;
} timer_regs;

// for uart display
typedef struct ua{
  unsigned int rxdata;
  unsigned int txdata;
  unsigned int status;
  unsigned int control;
  unsigned int divisor;
  unsigned int endofpacket;
} uart_regs;

// for eeprom
typedef struct ep{
  unsigned int tfrcmd;
  unsigned int rxdata;
  unsigned int ctrl;
  unsigned int iser;
  unsigned int isr;
  unsigned int status;
  unsigned int tfrcmdfifolvl;
  unsigned int rxdatafifolvl;
  unsigned int scllow;
  unsigned int sclhigh;
  unsigned int sdahold;
} eep_regs;

// for dma controller
typedef struct dma{
  unsigned int status;
  unsigned int readaddress;
  unsigned int writeaddress;
  unsigned int length;
  unsigned int control;
} dma_regs;
```

4.3 Video

Click the text below for a link to my video in Google Drive:

ECE_532_Prikkel_Final_Video