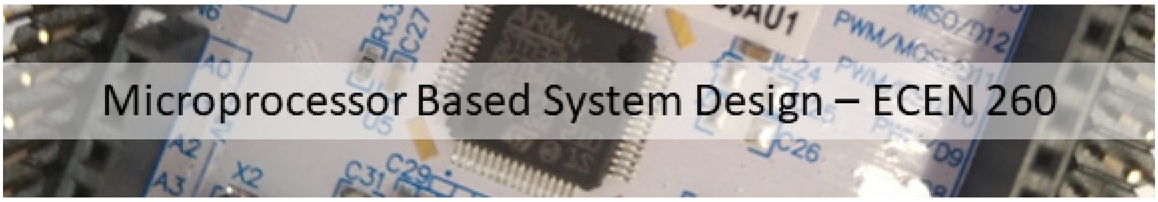

ECEN 260 - Final Project

Moving Chess Board

Seth Ricks

Instructor: Brother Allred

July 17, 2024

Contents

List of Tables

List of Figures

1 Project Overview

This report is for the final project in ECEN 260 (Microprocessor-based system design) at Brigham Young University–Idaho. The project is a Moving Chess Board, which is driven manually with a joystick and button.

Shown below are the specifications for the chess board system, the schematic of the circuit, the test plan and results, and appropriate code snippets to demonstrate the software functionality. The objectives of this project were the following:

1.1 Objectives

- To show what has been learned while in ECEN 260
- Integrate the following elements: Interrupts, ADC, PWM, Timers, I2C Display
- Create a project worthy of adding to a resume

2 Specifications

This system is designed to be a moving chess board, with the ability to move chess pieces manually without touching them. There are two inputs, which comes from a x/y joystick and a simple button. These two inputs together control the outputs of two rotational motors, and one servo motor. Another output is sent to a LCD screen, that shows the current direction of the joystick.

There are a few known limitations to this system. For one, the rotational motors only move at one speed. The button that controls the servo motor is also a little finicky, meaning that if the servo button is pushed rapidly it will not work as desired. There is also the possibility of the non-threaded rods getting stuck in their guides.

2.1 Parts List

- 1x STM32 Microcontroller
- 1x DROK DC Motor Driver
- 1x Breadboard
- 1x Button
- 1x Joystick
- 1x LCD Screen
- 1x Power Module
- 1x 100 Ω Resistors
- 2x AC-DC Power Adapter
- 2x 12v 600 RPM Gear Motor
- 1x Servo Motor
- 2x Threaded Rod
- 2x Non-Threaded Rod
- 4x Wood Stands
- 1x Base Board
- 1x Acrylic Chess Board
- 2x Demonstration Chess Pieces
- Assorted Male-Male/Male-Female Wires
- Various 3D Printed Connector Pieces

For specifics on the wiring setup, see the schematic in Section 3.'

3 Schematics

See Figure 1 for the schematic of the wiring for the Moving Chess Board.

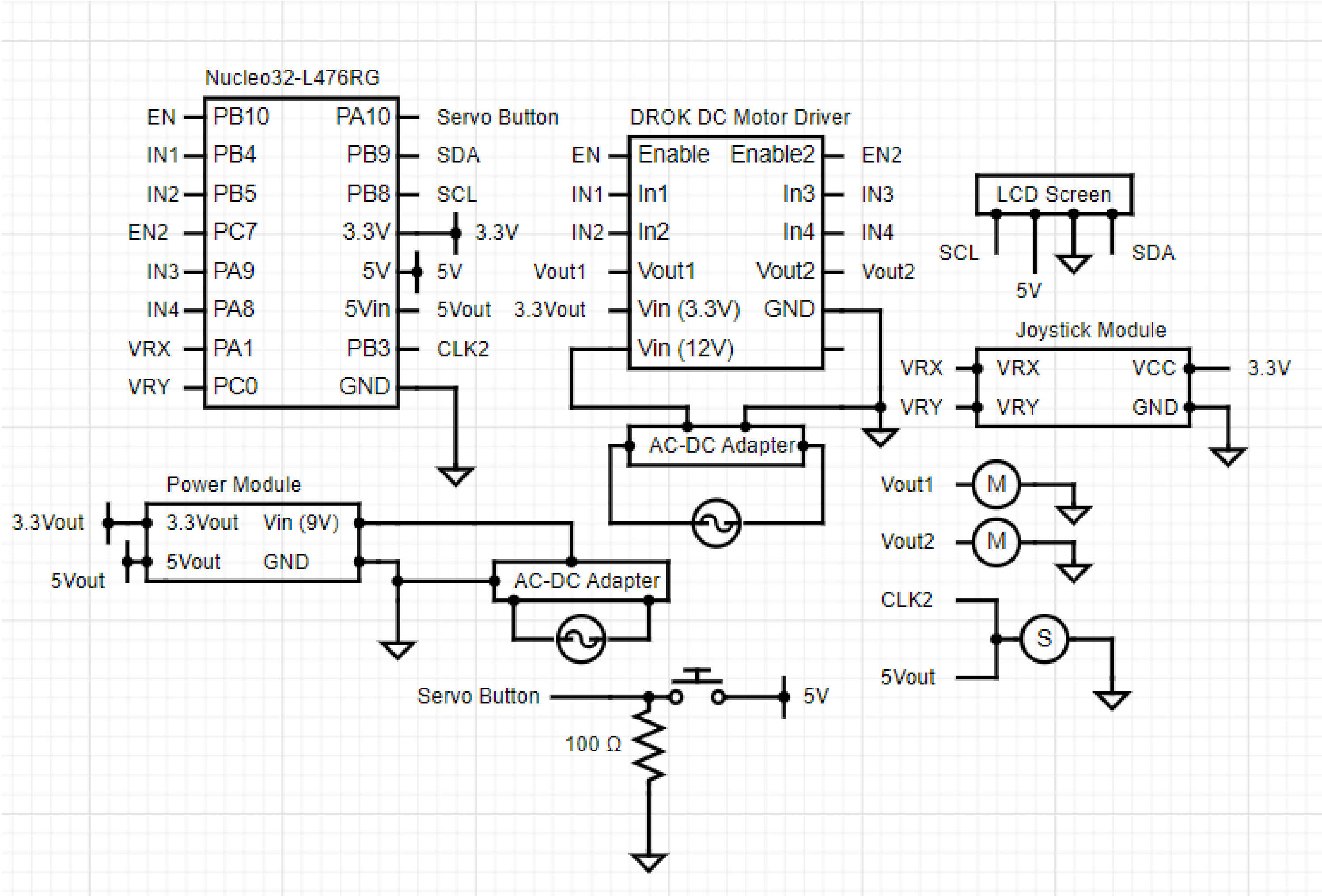

Figure 1: Schematic Diagram for the Moving Chess Board

4 Test Plan and Test Results

Section 4.1 below describes the test plan and results for the Moving Chess Board. The test plan includes several tests, each with precise steps to complete the test. The set of tests includes common cases as well as edge cases/error cases as appropriate. Each test also accurately indicates specific expected results. Then, the actual results are recorded after each test was completed.

4.1 Test Plan Procedure

This section includes the test procedure for each test.

- Test Scenario #1
  - Step 1: Turn the board on
  - Step 2: Wait for the LCD to load
  - Step 3: Move the joystick in all 8 directions, one at a time
  - Step 4: Press the servo arm control button
- Test Scenario #2
  - Step 1: Turn the board on
  - Step 2: Wait for the LCD to load
  - Step 3: Move the joystick around at random
  - Step 4: While still moving the joystick, push the servo button
- Test Scenario #3
  - Step 1: Turn the board on
  - Step 2: Wait for the LCD to load
  - Step 3: Rapidly press the servo arm control button
- Test Scenario #4
  - Step 1: Turn the board on
  - Step 2: Don't wait for the LCD to load, immediately start pressing the servo button and moving the joystick.

4.2 Expected and Observed Results

This section includes the the expected and actual results of each test.

- Test Scenario #1
  - Expected Result: The servo module moves in all 8 directions when prompted. When the module moves up or down, the LCD screen displays that accordingly. The same is so for the right and left directions. When the servo control button is pushed, the servo arm moves down.
  - Actual Result: Same as expected result
- Test Scenario #2
  - Expected Result: The same as test scenario #1. The servo motor moves as soon as the button is pushed, even when the other motors are moving as a result of the joystick being moved.
  - Actual Result: Same as expected result
- Test Scenario #3
  - Expected Result: When the servo control button is pushed rapidly, the servo arm moves up and down rapidly, according to the rate that the button is being pushed.
  - Actual Result: The servo arm doesn't actually move at the same rate that the button is pushed. Sometimes it follows the commands of the button, and other times it moves up and down at a different pace. It is unclear why this happens and requires further research.
- Test Scenario #4
  - Expected Result: The rotational shaft motors and the servo motor, along with the words on the LCD don't respond until the system is completely loaded.
  - Actual Result: Same as expected result

5 Code

The following is the code for the Moving Chess Board's STM32 microprocessor. There was a lot of code that was generated from the STM32 code, but I have only included the appropriate USER CODE. The following Section 5.1 has the appropriate code from "main.c".

5.1 Code for main.c

```c
#include "main.h"

#define I2C_ADDR 0x27 // I2C address of the PCF8574
#define RS_BIT 0 // Register select bit
#define EN_BIT 2 // Enable bit
#define BL_BIT 3 // Backlight bit
#define D4_BIT 4 // Data 4 bit
#define D5_BIT 5 // Data 5 bit
#define D6_BIT 6 // Data 6 bit
#define D7_BIT 7 // Data 7 bit
#define LCD_ROWS 2 // Number of rows on the LCD
#define LCD_COLS 16 // Number of columns on the LCD

//Initialize ADCs
ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;
ADC_HandleTypeDef hadc3;

//Initialize I1C and timers
I2C_HandleTypeDef hi2c1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim16;

//The backlight state is on
uint8_t backlight_state = 1;

//initialize all the directions
int DOWN = 0;
int RIGHT = 0;
int LEFT = 0;
int UP = 0;

//initialize if lcd message availability
uint8_t update_lcd = 0;

char lcd_buffer[2][16]; // Buffer to hold the LCD messages for each row

//servo motor status
int UP_servo = 1;

//Private functions
void SystemClock_Config(void);
void PeriphCommonClock_Config(void);
static void MX_GPIO_Init(void);
```

```
static void MX_ADC1_Init(void);
static void MX_ADC2_Init(void);
static void MX_ADC3_Init(void);
static void MX_TIM16_Init(void);
static void MX_I2C1_Init(void);
static void MX_TIM2_Init(void);

/* USER CODE BEGIN 0 */
//Function to write a nibble to the LCD
  void lcd_write_nibble(uint8_t nibble, uint8_t rs) {
   uint8_t data = nibble << D4_BIT;
   data |= rs << RS_BIT;
   data |= backlight_state << BL_BIT; // Include backlight state in data
   data |= 1 << EN_BIT;
   HAL_I2C_Master_Transmit(&hi2c1, I2C_ADDR << 1, &data, 1, 100);
   HAL_Delay(1);
   data &= ~(1 << EN_BIT);
   HAL_I2C_Master_Transmit(&hi2c1, I2C_ADDR << 1, &data, 1, 100);
  }
 //Function to send a command to the LCD
  void lcd_send_cmd(uint8_t cmd) {
   uint8_t upper_nibble = cmd >> 4;
   uint8_t lower_nibble = cmd & 0x0F;
   lcd_write_nibble(upper_nibble, 0);
   lcd_write_nibble(lower_nibble, 0);
   if (cmd == 0x01 || cmd == 0x02) {
   HAL_Delay(2);
   }
  }
 //Function to send data to the LCD
  void lcd_send_data(uint8_t data) {
   uint8_t upper_nibble = data >> 4;
   uint8_t lower_nibble = data & 0x0F;
   lcd_write_nibble(upper_nibble, 1);
   lcd_write_nibble(lower_nibble, 1);
  }
  void lcd_init() {
   HAL_Delay(50);
   lcd_write_nibble(0x03, 0);
   HAL_Delay(5);
   lcd_write_nibble(0x03, 0);
   HAL_Delay(1);
   lcd_write_nibble(0x03, 0);
   HAL_Delay(1);
   lcd_write_nibble(0x02, 0);
   lcd_send_cmd(0x28);
   lcd_send_cmd(0x0C);
   lcd_send_cmd(0x06);
   lcd_send_cmd(0x01);
   HAL_Delay(2);
  }
 //Funtion to write a string to the LCD
  void lcd_write_string(char *str) {
   while (*str) {
```

```c
    lcd_send_data(*str++);
    }
  }
 //Function to set the cursor
  void lcd_set_cursor(uint8_t row, uint8_t column) {
    uint8_t address;
    switch (row) {
    case 0:
    address = 0x00;
    break;
    case 1:
    address = 0x40;
    break;
    default:
    address = 0x00;
    }
    address += column;
    lcd_send_cmd(0x80 | address);
  }
 //Function to clear the LCD
  void lcd_clear(void) {
  lcd_send_cmd(0x01);
   HAL_Delay(2);
  }
 //Function to turn the backlight on and off
  void lcd_backlight(uint8_t state) {
    if (state) {
    backlight_state = 1;
    } else {
    backlight_state = 0;
    }
  }

/* USER CODE END 0 */

/* USER CODE BEGIN 2 */
     HAL_TIM_Base_Start_IT(&htim16); // Start Timer16
     // I2C pull-up resistors
     GPIOB->PUPDR |= 0b01 << (8*2);
     GPIOB->PUPDR |= 0b01 << (9*2);
     // Initialize the LCD
     lcd_init();
     lcd_backlight(1); // Turn on backlight

     HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);  // Start PWM on TIM2, Channel 2
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
     while (1)
     {
    //If there is data for the LCD to receive
       if (update_lcd)
       {
```

```
        update_lcd = 0; // Clear the flag
        lcd_set_cursor(0, 0); //cursor on first row, first column
        lcd_write_string(lcd_buffer[0]); //write the first message
        lcd_set_cursor(1, 0); //cursor on second row, first column
        lcd_write_string(lcd_buffer[1]); //write the second message
      }

      //move the first motor, so magnet moves up
      if (UP){
      HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, 1);
      HAL_GPIO_WritePin(ENABLE_GPIO_Port, ENABLE_Pin, 1);
      HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, 0);
      }

      //move the first motor, so magnet moves down
      else if (DOWN){
      HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, 0);
      HAL_GPIO_WritePin(ENABLE_GPIO_Port, ENABLE_Pin, 1);
      HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, 1);
      }

      //turn off the first motor
      else{
      HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, 0);
      HAL_GPIO_WritePin(ENABLE_GPIO_Port, ENABLE_Pin, 0);
      HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, 0);
      }

      //move the second motor, so magnet moves left
      if (LEFT){
      HAL_GPIO_WritePin(IN3_GPIO_Port, IN3_Pin, 1);
      HAL_GPIO_WritePin(ENABLE2_GPIO_Port, ENABLE2_Pin, 1);
      HAL_GPIO_WritePin(IN4_GPIO_Port, IN4_Pin, 0);
      }

      //move the second motor, so magnet moves right
      else if (RIGHT){
      HAL_GPIO_WritePin(IN3_GPIO_Port, IN3_Pin, 0);
      HAL_GPIO_WritePin(ENABLE2_GPIO_Port, ENABLE2_Pin, 1);
      HAL_GPIO_WritePin(IN4_GPIO_Port, IN4_Pin, 1);
      }

      //turn off the second motor
      else{
      HAL_GPIO_WritePin(IN3_GPIO_Port, IN3_Pin, 0);
      HAL_GPIO_WritePin(ENABLE2_GPIO_Port, ENABLE2_Pin, 0);
      HAL_GPIO_WritePin(IN4_GPIO_Port, IN4_Pin, 0);
      }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    }
  /* USER CODE END 3 */
}
```

```

/* USER CODE BEGIN 4 */
  // Callback: timer has rolled over
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  //if the clock that was triggerd is clock 16
  if (htim == &htim16)
  {
    int ADC_RANGE = 4096; // 2^12 (12-bit resolution)
    // Start ADC Conversions
    HAL_ADC_Start(&hadc2);
    HAL_ADC_Start(&hadc3);
    // Wait for ADC conversions to complete
    HAL_ADC_PollForConversion(&hadc2, HAL_MAX_DELAY);
    HAL_ADC_PollForConversion(&hadc3, HAL_MAX_DELAY);
    // Read ADC values
    uint16_t xjoy_measurement = HAL_ADC_GetValue(&hadc2);
    uint16_t yjoy_measurement = HAL_ADC_GetValue(&hadc3);
    // Convert ADC levels to a fraction of total
    float xjoy_value = ((float)xjoy_measurement) / ADC_RANGE;
    float yjoy_value = ((float)yjoy_measurement) / ADC_RANGE;

    //joystick is turned to the left
    if (yjoy_value < 0.3)
    {
      RIGHT = 1;
      LEFT = 0;
      //message to be sent to the display
      snprintf(lcd_buffer[1], 16, "LEFT        ");
    }
    //joystick is turned to the right
    else if (yjoy_value > 0.5)
    {
      RIGHT = 0;
      LEFT = 1;
      //message to be sent to the display
      snprintf(lcd_buffer[1], 16, "RIGHT         ");
    }
    //joystick is in the middle (not left or right)
    else
    {
    RIGHT = 0;
    LEFT = 0;
      //message to be sent to the display
      snprintf(lcd_buffer[1], 16, "NA            ");
    }

    //joystick is turned up
    if (xjoy_value < 0.3)
    {
      UP = 1;
      DOWN = 0;
      //message to be sent to the display
      snprintf(lcd_buffer[0], 16, "DOWN          ");
```

```
    }
    //joystick is turned down
    else if (xjoy_value > 0.5)
    {
      UP = 0;
      DOWN = 1;
      //message to be sent to the display
      snprintf(lcd_buffer[0], 16, "UP             ");
    }

    //joystick is in the middle (not up or down)
    else
    {
    UP = 0;
    DOWN = 0;
      //message to be sent to the display
      snprintf(lcd_buffer[0], 16, "NA             ");
    }

    //message ready to be sent to the LCD
    update_lcd = 1; // Set flag to update the LCD
  }
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  //if the servo button is pressed
  if (GPIO_Pin == SERVO_BUTTON_Pin){

    //if the motor is not already up
    if(!UP_servo){
      //move servo motor to the up position
      TIM2->CCR2 = 99;
      //The servo arm is now up
      UP_servo = 1;
    }

    //if the motor is already up, move it down
    else{
      //move servo motor to the down position
      TIM2->CCR2 = 50;
      //The servo arm is now down
      UP_servo = 0;
    }
  }
}
/* USER CODE END 4 */
```

6 Conclusion

The Moving Chess Board final project dealt with demonstrating understanding of everything that was taught in ECEN 260. In this project I deepened my understanding of several different concepts that were taught in class. This included PWM signals and how they are generated, clocks and how to configure them, Analog to Digital Converters, and 12C communication protocol. I accomplished the successful creation of the circuit and code using tools such as the STM32 IOC, and previous lab instructions. Other pieces of this project were 3D printed, whose design was created using CAD (Onshape).

In class before this final project, I learned about most of the material and code that went into the chess board. I saw PWM in actions as I made the code for the servo motor arm, and interrupts as I made the button for the servo motor. I applied my understanding of ADC to the movements of the joystick, and timers to the lapsed measurement of the joystick. A LCD screen was shown to demonstrated understanding of SPI communication protocol as well. I will be applying what I have learned about these various topics to my future job and personal projects, as I now have had a successful experience putting all these subjects together. This will boost my confidence of my capability in the future as I try to take on more complicated projects.

References