Final Report - Hand Slapper

Michael Han, Thomas Schmitz, Kaustabh Paul
Advanced Mechatronics Design - Final Project
December 7th, 2023

Summary

The Hand Slapper is a gaming device that interacts with the users through the console as well as the swatter. The base logic behind the machine is that the user places their hand on the platform and once the "play" condition is met, the swatter slaps the user's hand. For the user to win a point, the user has to avoid being slapped, and if the swatter ends up slapping the user, the computer wins a point. This idea came from looking at simple and enjoyable arcade games. Throughout the project, the team has used their mechatronic skills learned in "Advanced Mechatronics Design" such as configuring Analog Digital Converter, Motor Control, Circuit Design, and more to implement the necessary functionalities of the hand slapper

Mechanical Design

The team has designed the software as well as the hardware of the machine. Other than the swatter itself, everything was self-made through 3D printing for laser cutting.

The parts that are 3D printed are the following

- IR Sensor Casing

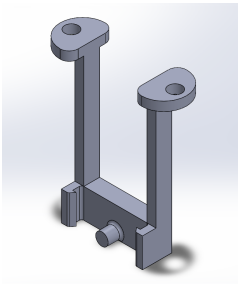


Figure 1

Since the machine performs and requires dynamical movements, IR sensors, which interact with the user, had to stay at its place. Therefore, IR sensor Casing was designed to secure the IR sensor's position in the machine. It was 3D printed with PLA and was screw tightened to the platform with M3 screws and bolts.

- Motor Casing

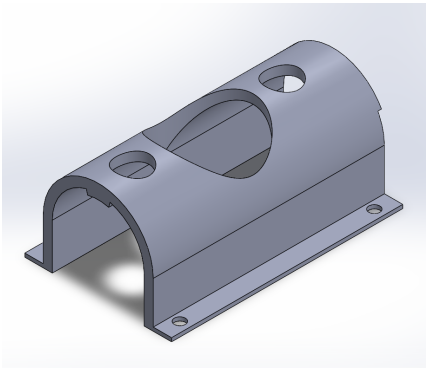
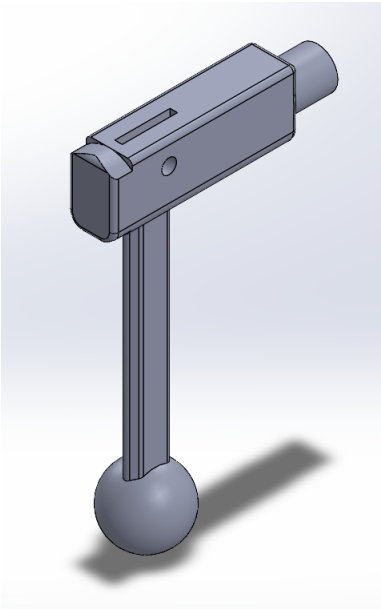


Figure 2

The motor was in charge of rotating the slapper which required the motor to be stable to perform the action. As a result, the motor casing was made to fix the motor's position as well as the rotation. The casing was built to be well ventilated through the holes on top. To restrict any movement and rotation of the motor, concise extrusion features with respect to the motor's exterior dimensions were designed.

- Motor Shaft Connection

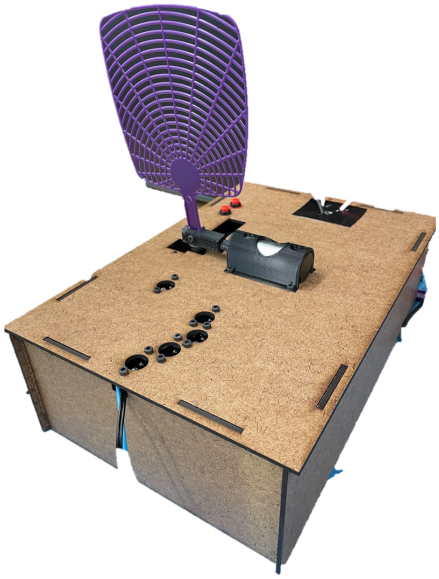


The motor shaft had to be connected to the swatter which needed to rotate in order to slap the user. The team made the motor shaft connection which connects the motor shaft and the swatter to resolve the problem. Interior of the side of the connection has the same dimensions as the motor shaft which enabled the connection between the motor shaft and the shaft connection and the swatter was placed on the top part of the motor shaft connection. However, the motor was not powerful enough to bring back the swatter from its horizontal position to the vertical position. Therefore, the team decided to add some weight to the opposite side of the swatter with respect to the rotational axis which is represented as a sphere at the end of a rod shape extrusion in the design.

Figure 3

The parts that are designed through laser cutting are following

- Platform



The platform consists of five wooden pieces that have been designed through laser cutting. The piece on the very top accounts for all the dimensions that need to be considered for the existing parts. The piece on top also considers the rotation of the motor shaft connection as well. The side pieces were designed to give room for the motor shaft connection when rotating as well as the microcontroller and the circuit board. As a result, all the components are either attached to the platform or inside the platform other than the power lines that need to be connected to the power source of the computer.

Figure 4

Mechatronic system design

The machine uses a single motor but multiple sensors. The components are essential parts to interact with the user since the motor runs to slap the user and the sensors react to the user's hand.

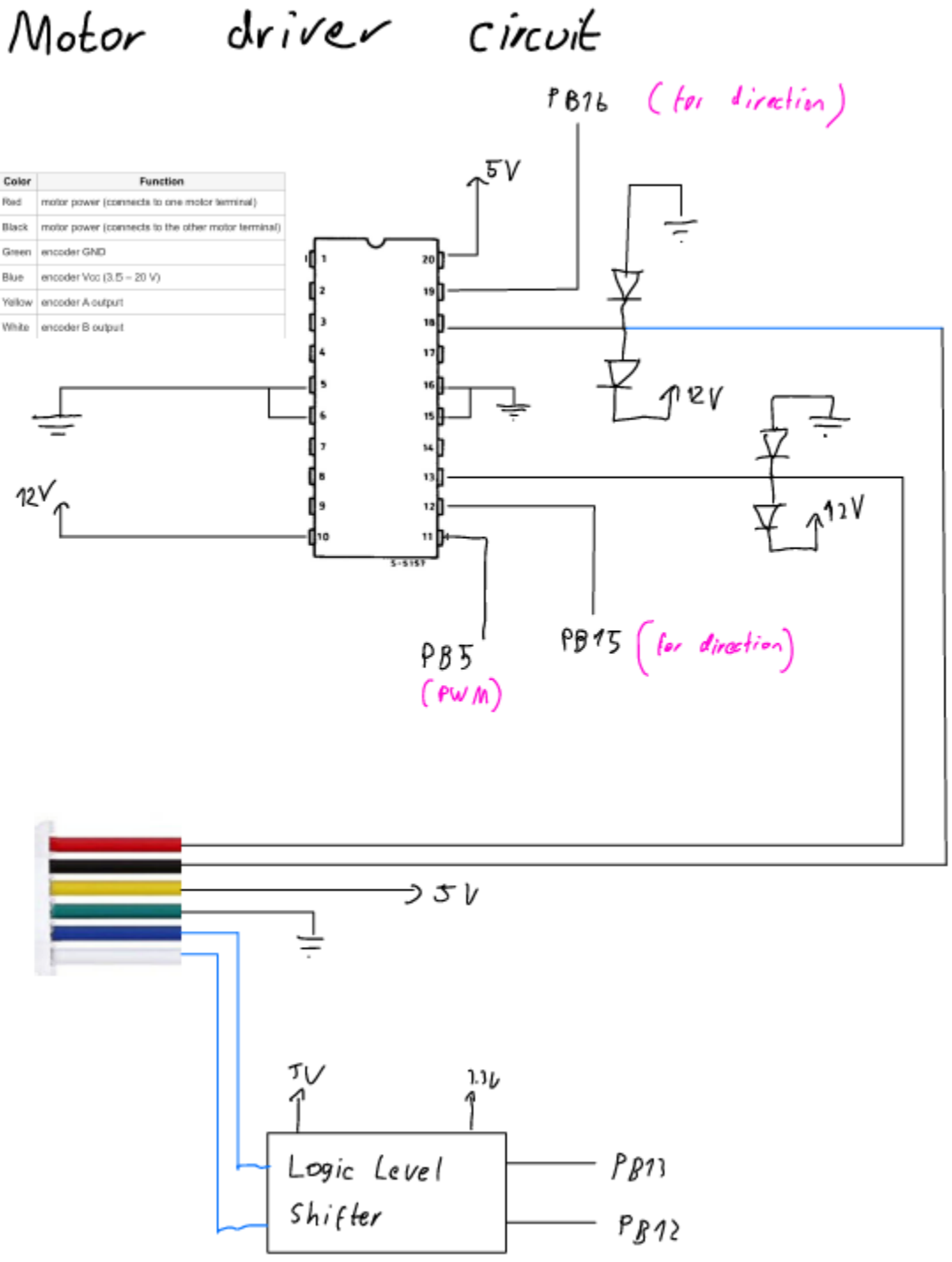
The motor that has been used for the machine is 'Pololu #4680'. The motor is a 12V DC Brushed Motor with 48 CPR encoder included. For the motor driver, the team used L293B to drive the motor. The purpose of an actuator for the machine was position control and since DC motors are not suitable for precise position control, the team used PID control and mechanical brakes to control the position of the motor. The team constantly followed the angle of the motor by reading the encoder counting and based on the angle, the motor changed its position with respect to the current status of the machine.

The sensors were used to interact with the user as well. The sensors(electrical components) used in the machine are IR distance sensors, Force Resistive Sensor(FSR), and buttons. The IR distance sensors were used to detect if the user has their hand on the platform. 5 IR sensors were used and each one was to detect each finger of the user. The IR sensors output high and low values with respect to the distance that they measure. Therefore, if one of the IR sensors does not detect the user's hand, meaning that the user has their hand off the platform, the game stops and requires the user to place their hand back on the platform.

The purpose of FSR was similar to the IR sensors. In order for the game to continue, the user needed to place a certain amount of force on to the FSR. If FSR does not read a certain value, the game stops and asks the user to place their hand on the platform again. The use of IR sensors and FSR are to make sure that the user has their hand on the platform and prevent cheating from the user at the same time.

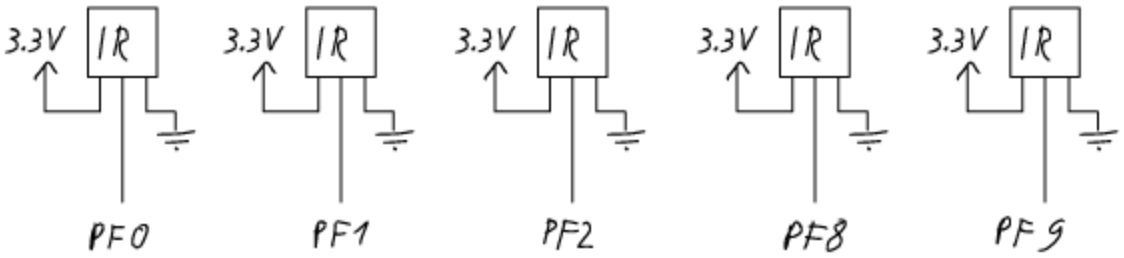
Lastly, a button was used to start and stop the game. If the input on the button reads high, then the game starts once the user places their hand on the platform and vice versa. The button acts as an entrance and the exit to the game. As a result, no button press means no game play while button press is the entering task to the game.

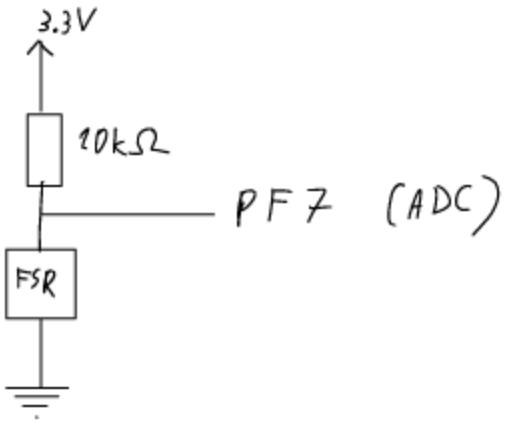
Final schematics

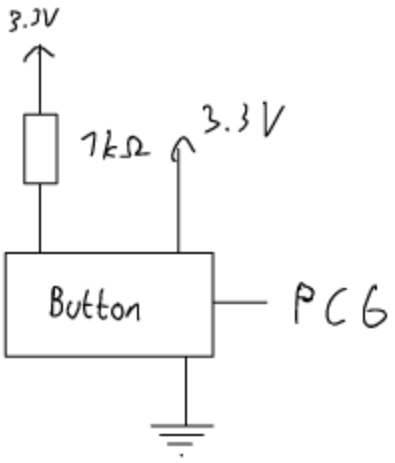


Color	Function
Red	motor power (connects to one motor terminal)
Black	motor power (connects to the other motor terminal)
Green	encoder GND
Blue	encoder Vcc (3.5 – 20 V)
Yellow	encoder A output
White	encoder B output

Sensors & Button Circuit







Digitally debounced

Bill of Materials

	12V DC Motor	Provided
	Swatter	$5.49
	IR sensors	$10.99
	FSR	Provided
	Button	$4.99
	Motor Driver	Provided

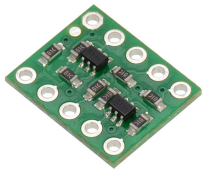
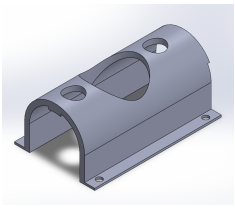
	Logic Level Shifter	Provided
	Diode	Provided
	Resister	Provided
	Wire	Provided
	Wooden Piece	Provided
	PLA material	Provided

Table 1

State Machine Design

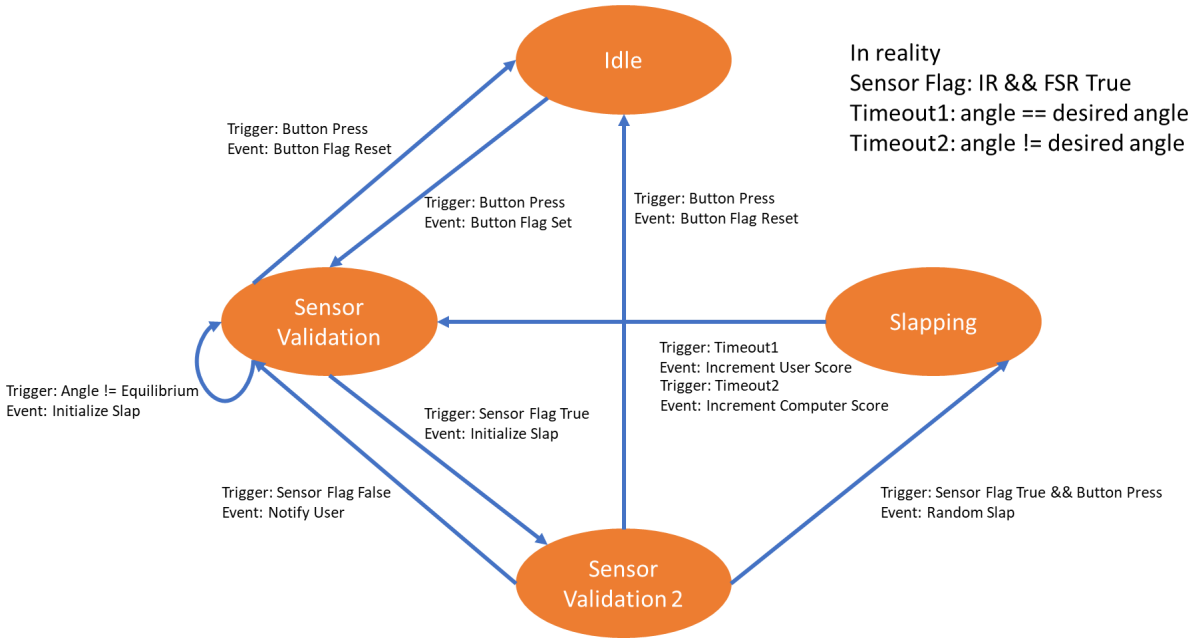


Figure 5

All header and code files and description

filename	Description
debug_mort.h/cpp	Contains print statements
hand_detector.h/c	Contains all the functionality
hardware_stm_adc.h/c	Initializes and reads from the ADC
hardware_stm_dma.h/c	Initializes direct memory access for ADC
hardware_stm_gpio.h/c	Initializes, reads, and writes to the GPIO pins
hardware_stm_interruptcontroller.h/c	Configures and services the interrupts
hardware_stm_timer3.h/c	Configures timer 3
main.cpp	The main file of the project
main.h	Contains global variables and function declarations
nucleo_led.c	Initializes and modifies the nucleo LEDs

stm32f4xx_rcc_mort.h/c	Starts the peripheral clocks

Source code

debug_mort.cpp

```
#include "mbed.h"
#include "debug_mort.h"

void debugprint(uint16_t number)
{
    printf("ADC Value: %u\n",number);
}

void debugprint32(uint32_t number)
{
    printf("Got to %u\n",number);
}

void debugprintRed( void )
{
    printf("Red Bear!!!\n");
}

void debugprintOrange( void )
{
    printf("Button Press\n");
}

void debugprintWhite( void )
{
    printf("White Bear!!!\n");
}

void debugprintGreen( void )
{
    printf("Green Bear!!!\n");
}

void debugprintHelloWorld( void )
```

```
{
    printf("I'm aliiiiiivveeee!!!\n");
}
```

debug_mort.h

```
#ifndef __DEBUG_MORT_H_
#define __DEBUG_MORT_H_

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes
------------------------------------------------------------------*/
#include "main.h"

/*Function
definitions-------------------------------------------------------------*/
void debugprint(uint16_t number);
void debugprint32(uint32_t number);
void debugprintRed( void );
void debugprintOrange( void );
void debugprintWhite( void );
void debugprintGreen( void );
void debugprintHelloWorld( void );

#ifdef __cplusplus
}
#endif

#endif /*__DEBUG_MORT_H */
```

hand_detector.c

```
/**

******************************************************************************
****
  * @file    hardware_stm_gpio.c
```

```
  * @author  mortamar@andrew.cmu.edu
  * @version 1.0
  * @date    Septembr-2021
  * @brief   Controls STM32F446ze GPIO

******************************************************************************
****
  */


#include "debug_mort.h"
#include "hardware_stm_gpio.h"
#include "hardware_stm_dma.h"
#include "stm32f4xx_rcc_mort.h"
#include "hardware_stm_adc.h"
#include "hardware_stm_interruptcontroller.h"
#include <cstdint>


uint16_t pin0 = 0;
uint16_t pin1 = 1;
uint16_t pin2 = 2;
uint16_t pin8 = 8;
uint16_t pin9 = 9;
uint32_t val;
uint32_t oldval=0;
uint16_t w=0;
uint32_t i, j, k, z;
uint16_t ButtonPress; //variable to store button state
int16_t des=18;
int16_t desequ = 0; //variable to store desired equlibrium
int16_t ierr=0;
//Initialization of userscore and slapperscore to keep track of points
uint16_t userscore=0;
uint16_t slapperscore=0;

//creates random delay
void delay2(uint16_t v){

    uint16_t j=0;
```

```
    while(j!=v){
        while(readcount()<15){ //once timer value is beneath 15 increment
            j=j+1;
        }

    }

}

//PD controller
int16_t pcontrol(int16_t kp, int16_t kd, int16_t ki){

    int16_t err;
    int16_t derr;
    err=des-angle; //calculates control error
    if(err==0){
        derr=0;
    }
    else{
        derr=(angle-oldangle)/delT;
    }
    ierr=ierr+delT*err;
    oldangle=angle;
    // PD controller for the new PWM
    return kp*err + kd*derr;
}

void initializeADCforFSR(void){
    // initialize ADC for GPIOF7
    initADC3_5();
    // Start ADC Conversion
    startADCConversion();
}

void initializeGPIOF01289AsInput(void){
    // initialize GPIOF0 as input
    initGpioFAsInput(pin0);
    // initialize GPIOF1 as input
    initGpioFAsInput(pin1);
    // initialize GPIOF2 as input
```

```
    initGpioFAsInput(pin2);
    // ínitialize GPIOF8 as input
    initGpioFAsInput(pin8);
    // ínitialize GPIOF9 as input
    initGpioFAsInput(pin9);
}

uint16_t CheckIRSensors(void){
    //Checks all 5 IR sensor outputs at Pins F0,F1,F2,F8,F9
    //IR sensor returns 0 if obstacle is within treshold, 1 else
    //returns 1 if all IR sensor outputs are 0
    uint16_t F0, F1, F2, F8, F9;

    uint16_t sensorcheck = 1;
    F0 = checkGPIOF(pin0);
    F1 = checkGPIOF(pin1);
    F2 = checkGPIOF(pin2);
    F8 = checkGPIOF(pin8);
    F9 = checkGPIOF(pin9);



    if (F0 || F1 || F2 || F8 || F9){
        sensorcheck = 0;
    }else{
        sensorcheck=1;
    }
    return sensorcheck;
}

uint16_t CheckFSR(void){
    //Reads FSR value through ADC
    //FSR is in series with 10k resistor, ADC pin reads voltage at ADC
    //If no force is applied FSR (resistance is very high >10MOhm) pin
will read 3.3V, ADC 2^16
    //

    //returns 1 if ADC reads value below FSRLimit, this means FSR is
reading adequate force
    uint16_t FSR;
```

```
    uint16_t FSRLimit = 1050;
    uint16_t sensorcheck = 1;
    FSR = returnADC3StoredValueforone();
    //printf("ADC value: %u \n", FSR);
    if (FSR > FSRLimit){ // when FSR is not pressed
        sensorcheck = 0;
    }else{
       sensorcheck=1;
   }
    return sensorcheck;
}

//Resets scores in order to start another game
void resetscore(void){
    userscore = 0;
    slapperscore = 0;
}
void printscore(void){
    printf("Your Score is: %u Computer Score is: %u \n", userscore,
slapperscore);
}
void printfinalscore (void){
    if (userscore > slapperscore){
        printscore();
        printf("Your Won \n");
    }
    else{
        printscore();
        printf("Your Lost \n");
    }
}

//Moves slapper to upright position, referenced as 1
//Function no longer used because angle=0 doesn't always result in upright
position
void moveup(void){
    des=desequ;//0; // desire angle
    int16_t result=1;
    int16_t oldangle=-20;
    uint32_t j=0;
```

```
    while(1){
        // update the value for PWM Cycle
        result=pcontrol(25,1,0.4);
        // control the motor to acheive desire angle
        pwm(result);
        if (des==oldangle && angle==des){ // if the desire angle is met,
leave the loop
            //printf("angle: %d \n", angle);
            break;
        }
        j++;
        if(j==1000000){ //Check if the desire angle is met
        oldangle=angle;
        j=0;
        }
    }
    // beark the motor and leave the motor at the desired position
    // initialize GPIOB15 as Output high
    initGpioBAsOut(15, 1);
    // initialize GPIOB6 as Output high
    initGpioBAsOut(6, 1);
    pwm(0);
}

//Generates random uint16_t
uint16_t getrand(void){
    uint16_t val= readcount5();
    while(val>500){
        val=val/10;
    }
    return val;
}

//Moves slapper to upright position by determining equlibrium
//replaces move up
void initslapper(void){
    des= -10;
    int16_t result;
    int16_t oldangle=-20;
    uint32_t j = 0;
```

```
    while(1){
        result=pcontrol(28,0.3,0.4);
        // control the motor to acheive desire angle
        pwm(result);

        j++;
        if(j==1000000){ //Check if the desire angle is met
            oldangle=angle;
            j=0;
        }

        if(oldangle==angle){
            desequ=angle; //store newly found equlibrium angle
            break;

        }

    }
    printf("Slapper Initialization complete \n");

}

void slap(void){

    des=desequ+27; // desire angle
    int16_t result=1;
    int16_t oldangle=-20;
    int16_t slappedanlge = -20; //another version of oldangle that is
updated more frequently
    uint32_t j = 0;
    uint32_t k = 0;
    uint16_t IR = CheckIRSensors();
    uint16_t FSR = CheckFSR();
    while(1){
        // update the value for PWM Cycle
        result=pcontrol(28,0.3,0.4);
        // control the motor to acheive desire angle
        pwm(result);
        if (des==oldangle && angle==des){ // if the desire angle is met,
leave the loop
```

```
            //If desired angle has been reached there was no obstacle
hence user was able to avoid being hit
            userscore=userscore+1;
            break;
        }
        if (angle != des && angle <= slappedanlge){
            //Slapper unable to reachdesired angle-->obstacle-->user's
hand has been slapped
            printf("you have been slapped \n");
            slapperscore= slapperscore+1;
            break;
        }
        j++;
        if(j==1000000){ //Check if the desire angle is met
            oldangle=angle;
            j=0;
        }
        k++;
        if(k == 500000){
            slappedanlge = angle;
            k = 0;
        }
    }
    // beark the motor and leave the motor at the desired position
    // initialize GPIOB15 as Output high
    initGpioBAsOut(15, 1);
    // initialize GPIOB66 as Output high
    initGpioBAsOut(6, 1);
    pwm(0);

}

void checkButton (void){
    //Checks if button was pressed by using digital debounce using delays
    //changes global variable ButtonPress
    val = checkGPIOC6();
    if(val!=oldval){
        //debugprintOrange();//Button is pressed

        if(w==0){// if button pressed for play
```

```
            ButtonPress = 1;
            w=1;
        }
        else{// if button pressed for stop
            ButtonPress = 0;
            w=0;
        }

        //delay2(20);
        // for loop for delay due to the noise of the button press
        for (i = 0; i < 100000000; i++)
        {
            for (k = 0; k < 200000000; k++)
                j = j + 1; // tiny delay
        }
        oldval=val;
    }
    //printf("button function read \n");
}

void PlayNStop(void){
    //Reads button and sensorsand begins randomly slapping
    // If conditions are not met, notifies user what he needs to do (e.g.
cover sensor or press button)
    uint16_t IR = CheckIRSensors();
    uint16_t FSR = CheckFSR();
    uint16_t Button;
    uint16_t val;

    if (ButtonPress && IR && FSR) { //Start slapping phase
        //debugprint(ButtonPress);
        printf("Start! \n");
        while(ButtonPress){
            //Wait until button is pressed
            checkButton();
            if (ButtonPress == 0){
                break;
            }
            val=readcount();
            val=val%2;
```

```
            val=val+2; //generates random number between 2 and 3
            initslapper();
            if((CheckIRSensors() && CheckFSR())==0){
                printf("Please place your hands on the sensors ! \n");
                printf("Current readings: IR=%u  FSR=%u ", IR, FSR);
                break;
            }
            if (ButtonPress == 0){
                break;
            }
            delay2(val*10); //random delay
            slap();
        }
    }
    else{
        if(angle != 0){
            //moves back to upright position
            printf("Stop! \n");
            moveup();
        }
        if(ButtonPress){
            printf("Please place your hands on the sensors ! \n");
            printf("Current readings: IR=%u  FSR=%u ", IR, FSR);
            printf("\n");
        }
        else{
            printf("Please press button to start \n");
            printscore();
            printf("\n");
        }
    }
    //printf("PlayNStop function read \n");
}
```

hand_detector.h

```
/* Define to prevent recursive inclusion
-------------------------------------*/
#ifndef __HAND_DETECTOR_H_
#define __HAND_DETECTOR_H_
```

```
#ifdef __cplusplus
 extern "C" {
#endif

/* Includes
------------------------------------------------------------------*/
#include "main.h"


/*Function
definitions-----------------------------------------------------------*/
void delay2(uint16_t v);
void initializeADCforFSR(void);
int16_t pcontrol(int16_t kp, int16_t kd, int16_t ki);
void initializeGPIOF01289AsInput(void);
uint16_t CheckIRSensors(void);
uint16_t CheckFSR(void);
void checkButton (void);
void moveup(void);
void slap(void);
void printscore(void);
void printfinalscore(void);
void initslapper(void);
void resetscore(void);
void PlayNStop(void);
#ifdef __cplusplus
}
#endif

#endif /*__GPIO_H */
```

hardware_stm_adc.c

```
/**

*****************************************************************************
****
  * @file    hardware_stm_gpio.c
  * @author  mortamar@andrew.cmu.edu
```

```
  * @version 1.0
  * @date    Septembr-2021
  * @brief   Controls STM32F446ze GPIO

******************************************************************************
****
  */

#include "hardware_stm_gpio.h"
#include "hardware_stm_dma.h"
#include "stm32f4xx_rcc_mort.h"
#include <cstdint>



/* MACRO
definitions-----------------------------------------------------------*/
#define SYSTEM_CONTROL_BASE_ADDRESS (0xE000E000)
#define NVIC_BASE_ADDRESS (SYSTEM_CONTROL_BASE_ADDRESS + 0x100)
#define NVIC_INTERRUPT_SET_ENABLE_REGISTER_0_31 (NVIC_BASE_ADDRESS)
#define ADC_INTERRUPT_BIT ((uint32_t)0x40000)

#define ADC123_BASE_ADDRESS ((uint32_t)0x40012000)
#define ADC1_BASE_ADDRESS (ADC123_BASE_ADDRESS + 0x000)
#define ADC2_BASE_ADDRESS (ADC123_BASE_ADDRESS + 0x100)
#define ADC3_BASE_ADDRESS (ADC123_BASE_ADDRESS + 0x200)

// status register
#define ADC1_SR_REGISTER (ADC1_BASE_ADDRESS + 0x00)
#define ADC2_SR_REGISTER (ADC2_BASE_ADDRESS + 0x00)
#define ADC3_SR_REGISTER (ADC3_BASE_ADDRESS + 0x00)

// control register 1
#define ADC1_CR1_REGISTER (ADC1_BASE_ADDRESS + 0x04)
#define ADC2_CR1_REGISTER (ADC3_BASE_ADDRESS + 0x04)
#define ADC3_CR1_REGISTER (ADC3_BASE_ADDRESS + 0x04)

// control register 2
#define ADC1_CR2_REGISTER (ADC1_BASE_ADDRESS + 0x08)
#define ADC2_CR2_REGISTER (ADC3_BASE_ADDRESS + 0x08)
```

```
#define ADC3_CR2_REGISTER (ADC3_BASE_ADDRESS + 0x08)

// sample time register 1
#define ADC1_SMPR1_REGISTER (ADC1_BASE_ADDRESS + 0x0C)
#define ADC2_SMPR1_REGISTER (ADC2_BASE_ADDRESS + 0x0C)
#define ADC3_SMPR1_REGISTER (ADC3_BASE_ADDRESS + 0x0C)

// sample time register 2
#define ADC1_SMPR2_REGISTER (ADC1_BASE_ADDRESS + 0x10)
#define ADC2_SMPR2_REGISTER (ADC2_BASE_ADDRESS + 0x10)
#define ADC3_SMPR2_REGISTER (ADC3_BASE_ADDRESS + 0x10)

// injected channel data offset register x

// watchdog higher threshold register
#define ADC1_HTR_REGISTER (ADC1_BASE_ADDRESS + 0x24)
#define ADC2_HTR_REGISTER (ADC2_BASE_ADDRESS + 0x24)
#define ADC3_HTR_REGISTER (ADC3_BASE_ADDRESS + 0x24)

// watchdog higher threshold register
#define ADC1_LTR_REGISTER (ADC1_BASE_ADDRESS + 0x28)
#define ADC2_LTR_REGISTER (ADC2_BASE_ADDRESS + 0x28)
#define ADC3_LTR_REGISTER (ADC3_BASE_ADDRESS + 0x28)

// regular sequence 1 register
#define ADC1_SQR1_REGISTER (ADC1_BASE_ADDRESS + 0x2C)
#define ADC2_SQR1_REGISTER (ADC2_BASE_ADDRESS + 0x2C)
#define ADC3_SQR1_REGISTER (ADC3_BASE_ADDRESS + 0x2C)

// regular sequence 2 register
#define ADC1_SQR2_REGISTER (ADC1_BASE_ADDRESS + 0x30)
#define ADC2_SQR2_REGISTER (ADC2_BASE_ADDRESS + 0x30)
#define ADC3_SQR2_REGISTER (ADC3_BASE_ADDRESS + 0x30)

// regular sequence 3 register
#define ADC1_SQR3_REGISTER (ADC1_BASE_ADDRESS + 0x34)
#define ADC2_SQR3_REGISTER (ADC2_BASE_ADDRESS + 0x34)
#define ADC3_SQR3_REGISTER (ADC3_BASE_ADDRESS + 0x34)

// injected sequence register
```

```
#define ADC1_JSQR_REGISTER (ADC1_BASE_ADDRESS + 0x38)
#define ADC2_JSQR_REGISTER (ADC2_BASE_ADDRESS + 0x38)
#define ADC3_JSQR_REGISTER (ADC3_BASE_ADDRESS + 0x38)

// injected data register x

// regular data register
#define ADC1_DR_REGISTER (ADC1_BASE_ADDRESS + 0x4C)
#define ADC2_DR_REGISTER (ADC2_BASE_ADDRESS + 0x4C)
#define ADC3_DR_REGISTER (ADC3_BASE_ADDRESS + 0x4C)

// common status register
#define ADC_CSR_REGISTER (ADC1_BASE_ADDRESS + 0x300)

// common control register
#define ADC_COMMON_CCR_REGISTER (ADC_CSR_REGISTER + 0x04)

// common regular data register for dual and triple modes
#define ADC_CDR_REGISTER (ADC_CSR_REGISTER + 0x08)

// ADC_CCR_PRESCALER
#define ADC_PRESCALER_4 ((uint32_t)0x10000)

// ADC_SR_EOC
#define ADC_EOC ((uint32_t)0b10)

// ADC_CR1
#define ADC_SCAN ((uint32_t)0x100)
#define ADC_EOCIE ((uint32_t)0x20)

// ADC_CR2
#define ADC_EOCS ((uint32_t)0x400)
#define ADC_CONT ((uint32_t)0x2)
#define ADC_DDS ((uint32_t)0x200)
#define ADC_DMA ((uint32_t)0x100)
#define ADC_SWSTART ((uint32_t)0x40000000)

// conversions
#define ADC_1_CONVERSIONS ((uint32_t)0b0000)
#define ADC_3_CONVERSIONS (((uint32_t)0b0010) << 20)
```

```
// SMPR2 Cycle
#define ADC_SMP_5_MX ((uint32_t)0x38000)
#define ADC_SMP_6_MX (((uint32_t)0b111) << 18)
#define ADC_SMP_7_MX (((uint32_t)0b111) << 21)

// SQR3 conversion
#define ADC_CHANNEL_5_MORT2 ((uint32_t)0b101)
#define ADC_CHANNEL_6_MORT2 ((uint32_t)0b110)
#define ADC_CHANNEL_7_MORT2 ((uint32_t)0b111)

// CR2
#define ADC_ADON ((uint32_t)0b1)

uint16_t adcvalue[3];
uint8_t adcnewvalue;
uint8_t adcindex;

/* function
definitions----------------------------------------------------------------*/
void enableNVIC_ADC( void )
{
    uint32_t * reg_pointer_32;
    reg_pointer_32 = (uint32_t *)NVIC_INTERRUPT_SET_ENABLE_REGISTER_0_31;
    *reg_pointer_32 = ADC_INTERRUPT_BIT;
}

void initADC3_567( void )
{
    uint32_t * reg_pointer;
    /* Initialize adcvalues */
    adcvalue[0] = 0;
    adcvalue[1] = 0;
    adcvalue[2] = 0;
    adcindex = 0;
    adcnewvalue = 0;
    /* Turn on ADC3 bus clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC3, ENABLE);
    /* Initialize GPIO F 7 8 9 as analog */
    initGpioF789AsAnalog();
```

```
    enableNVIC_ADC();
    /*Setup the clock Prescalers*/
    reg_pointer = (uint32_t *)ADC_COMMON_CCR_REGISTER;
    *reg_pointer = ADC_PRESCALER_4;
    /* configure ADC 12bit resolution, End of conversion interrupt
Enabled,
    SCAN Mode enable to be able to scan a group of channels */
    reg_pointer = (uint32_t *)ADC3_CR1_REGISTER;
    *reg_pointer = ADC_EOCIE + ADC_SCAN;
    /* configure ADC External trigger disabled, right data alignment, no
DMA,
    EOC is set at the end of each regular conversion, conitnuous
conversion disabled */
    reg_pointer = (uint32_t *)ADC3_CR2_REGISTER;
    *reg_pointer = ADC_EOCS;
    /* There will be 3 channels in the sequence of conversions */
    reg_pointer = (uint32_t *)ADC3_SQR1_REGISTER;
    *reg_pointer = ADC_3_CONVERSIONS;
    /* There Channels 5,6,7 to max sampling times (480 cycles) */
    reg_pointer = (uint32_t *)ADC3_SMPR2_REGISTER;
    *reg_pointer = ADC_SMP_5_MX + ADC_SMP_6_MX + ADC_SMP_7_MX;
    /* Configure the sequence of conversion for the ADC 5,6,7 */
    reg_pointer = (uint32_t *)ADC3_SQR3_REGISTER;
    *reg_pointer = ADC_CHANNEL_5_MORT2 + (ADC_CHANNEL_6_MORT2 << 5) +
(ADC_CHANNEL_7_MORT2 << 10);
    /* Enable the ADC3 */
    reg_pointer = (uint32_t *)ADC3_CR2_REGISTER;
    *reg_pointer = *reg_pointer | ADC_ADON;
}

void initADC3_5( void )
{
    uint32_t * reg_pointer;
    /* Initialize adcvalues */
    adcvalue[0] = 0;
    /* Turn on ADC3 bus clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC3, ENABLE);
    /* Initialize GPIO F 7 8 9 as analog */
    initGpioF7AsAnalog();
    initDMAForADC3_1channel();
```

```
    enableDMAForAdc3_3channels();
    /*Setup the clock Prescalers*/
    reg_pointer = (uint32_t *)ADC_COMMON_CCR_REGISTER;
    *reg_pointer = ADC_PRESCALER_4;
    /* configure ADC 12bit resolution, End of conversion interrupt
Enabled,
    SCAN Mode enable to be able to scan a group of channels */
    reg_pointer = (uint32_t *)ADC3_CR1_REGISTER;
    *reg_pointer = ADC_SCAN;
    /* configure ADC External trigger disabled, right data alignment, DMA,
    EOC is set at the end of each regular conversion, conitnuous
conversion enabled */
    reg_pointer = (uint32_t *)ADC3_CR2_REGISTER;
    *reg_pointer = ADC_EOCS + ADC_CONT + ADC_DDS + ADC_DMA;
    /* There will be 1 channels in the sequence of conversions */
    reg_pointer = (uint32_t *)ADC3_SQR1_REGISTER;
    *reg_pointer = ADC_1_CONVERSIONS;
    /* There Channels 5 to max sampling times (480 cycles) */
    reg_pointer = (uint32_t *)ADC3_SMPR2_REGISTER;
    *reg_pointer = ADC_SMP_5_MX;
    /* Configure the sequence of conversion for the ADC 5,6,7 */
    reg_pointer = (uint32_t *)ADC3_SQR3_REGISTER;
    *reg_pointer = ADC_CHANNEL_5_MORT2;
    /* Enable the ADC3 */
    reg_pointer = (uint32_t *)ADC3_CR2_REGISTER;
    *reg_pointer = *reg_pointer | ADC_ADON;
}

void initADC3_567_withDMA( void )
{
    uint32_t * reg_pointer;
    /* Turn on ADC3 bus clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC3, ENABLE);
    /* Initialize GPIO F 7 8 9 as analog */
    initGpioF789AsAnalog();
    initDMAForADC3_3channels();
    enableDMAForAdc3_3channels();
    /*Setup the clock Prescalers*/
    reg_pointer = (uint32_t *)ADC_COMMON_CCR_REGISTER;
    *reg_pointer = ADC_PRESCALER_4;
```

```
    /* configure ADC 12bit resolution, End of conversion interrupt
Enabled,
    SCAN Mode enable to be able to scan a group of channels */
    reg_pointer = (uint32_t *)ADC3_CR1_REGISTER;
    *reg_pointer = ADC_SCAN;
    /* configure ADC External trigger disabled, right data alignment, DMA,
    EOC is set at the end of each regular conversion, conitnuous
conversion enabled */
    reg_pointer = (uint32_t *)ADC3_CR2_REGISTER;
    *reg_pointer = ADC_EOCS + ADC_CONT + ADC_DDS + ADC_DMA;
    /* There will be 3 channels in the sequence of conversions */
    reg_pointer = (uint32_t *)ADC3_SQR1_REGISTER;
    *reg_pointer = ADC_3_CONVERSIONS;
    /* There Channels 5,6,7 to max sampling times (480 cycles) */
    reg_pointer = (uint32_t *)ADC3_SMPR2_REGISTER;
    *reg_pointer = ADC_SMP_5_MX + ADC_SMP_6_MX + ADC_SMP_7_MX;
    /* Configure the sequence of conversion for the ADC 5,6,7 */
    reg_pointer = (uint32_t *)ADC3_SQR3_REGISTER;
    *reg_pointer = ADC_CHANNEL_5_MORT2 + (ADC_CHANNEL_6_MORT2 << 5) +
(ADC_CHANNEL_7_MORT2 << 10);
    /* Enable the ADC3 */
    reg_pointer = (uint32_t *)ADC3_CR2_REGISTER;
    *reg_pointer = *reg_pointer | ADC_ADON;
}

void storeNewADCValueInSequence( void )
{
    uint32_t *reg_pointer_32;
    /*Get value stored in data register*/
    reg_pointer_32 = (uint32_t *)ADC3_DR_REGISTER;
    /*Store it in an array*/
    adcvalue[adcindex] = *reg_pointer_32 & 0xFFF;
    /*Increase the index of the array*/
    adcindex = adcindex + 1;
    if (adcindex > 2)
    {
        adcindex = 0; //if index is past the max value
    }
}
```

```
void startADCConversion ( void )
{
    uint32_t *reg_pointer_32;
    /* Clear any pending flags in the status register */
    reg_pointer_32 = (uint32_t *)ADC3_SR_REGISTER;
    *reg_pointer_32 = 0;
    /* start conversion of regular channels */
    reg_pointer_32 = (uint32_t *)ADC3_CR2_REGISTER;
    *reg_pointer_32 = *reg_pointer_32 | ADC_SWSTART;
}

uint16_t returnNewADCValue( void )
{
    return  adcvalue[adcindex];
}

uint8_t returnIfADChasnewValue( void )
{
    return adcnewvalue;
}

void clearIfADChasnewValue( void )
{
    adcnewvalue = 0;
}

void ADC_IRQHanlder( void )
{
    uint32_t *reg_pointer_32_sr;
    uint32_t *reg_pointer32_cr1;
    uint32_t value;

    reg_pointer_32_sr = (uint32_t *) ADC3_SR_REGISTER;
    reg_pointer32_cr1 = (uint32_t *) ADC3_CR1_REGISTER;
    value = *reg_pointer_32_sr;

    if (( (value & ADC_EOC) > 0) && ((*reg_pointer32_cr1 & ADC_EOCIE)))
    { // if we wanted the end of conversion interrupt and that is the one
that triggered
        /* store the Data value */
```

```
        storeNewADCValueInSequence(); // reading the value from the data
register clears the interrupt
        adcnewvalue = adcnewvalue + 1; // we have one more new value
    }
}
```

hardware_stm_adc.h

```
/* Define to prevent recursive inclusion
-------------------------------------*/
#ifndef __HARDWARE_STM_ADC_H_
#define __HARDWARE_STM_ADC_H_

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes
------------------------------------------------------------------*/
#include "main.h"


/*Function
definitions-----------------------------------------------------------*/
void enableNVIC_ADC( void );
void initADC3_567( void );
void initADC3_5( void );
void initADC3_567_withDMA( void );
void storeNewADCValueInSequence( void );
void startADCConversion ( void );
uint16_t returnNewADCValue( void );
uint8_t returnIfADChasnewValue( void );
void clearIfADChasnewValue( void );
void ADC_IRQHanlder( void );


#ifdef __cplusplus
}
#endif

#endif /*__GPIO_H */
```

hardware_stm_dma.c

```
/**

******************************************************************************
****
  * @file    hardware_stm_gpio.c
  * @author  mortamar@andrew.cmu.edu
  * @version 1.0
  * @date    Septembr-2021
  * @brief   Controls STM32F446ze GPIO

******************************************************************************
****
  */

#include "hardware_stm_gpio.h"
#include "stm32f4xx_rcc_mort.h"
#include <cstdint>


//led 1 is connected to PB0.
// GPIO B addresses: 0x4002 0400 - 0x4002 07FF
// GPIO C addresses: 0x4002 0800 - 0x4002 0BFF


/* MACRO
definitions-----------------------------------------------------------*/
#define DMA2_BASE_ADDRESS ((uint32_t)0x40026400)
#define DMA2_LISR_REGISTER (DMA2_BASE_ADDRESS + 0x00)
#define DMA2_HISR_REGISTER (DMA2_BASE_ADDRESS + 0x04)
#define DMA2_LIFCR_REGISTER (DMA2_BASE_ADDRESS + 0x08)
#define DMA2_HIFSR_REGISTER (DMA2_BASE_ADDRESS + 0x0C)
#define DMA2_S0CR_REGISTER (DMA2_BASE_ADDRESS + 0x10)
#define DMA2_S0NDTR_REGISTER (DMA2_BASE_ADDRESS + 0x14)
#define DMA2_S0PAR_REGISTER (DMA2_BASE_ADDRESS + 0x18)
#define DMA2_S0M0AR_REGISTER (DMA2_BASE_ADDRESS + 0x1C)
#define DMA2_S0M1AR_REGISTER (DMA2_BASE_ADDRESS + 0x20)
```

```
#define DMA2_S0FCR_REGISTER (DMA2_BASE_ADDRESS + 0x24)

// bits and flags
// DMA_SxCR: Dma Stream x configuration register
#define DMA_SxCR_CHANNEL_2_SELECT (((uint32_t)2) << 25)
#define DMA_SxCR_MSIZE_HALF_WORD (((uint32_t)1) << 13)
#define DMA_SxCR_PSIZE_HALF_WORD (((uint32_t)1) << 11)
#define DMA_SxCR_MINC_INCREMENT (((uint32_t)1) << 10)
#define DMA_SxCR_CIRC_ENABLE (((uint32_t)1) << 8)
#define DMA_SxCR_DIR_PERTOMEM 0
#define DMA_SxCR_STREAM_ENABLE 1

#define ADC123_BASE_ADDRESS ((uint32_t)0x40012000)
#define ADC1_BASE_ADDRESS (ADC123_BASE_ADDRESS + 0x000)
#define ADC2_BASE_ADDRESS (ADC123_BASE_ADDRESS + 0x100)
#define ADC3_BASE_ADDRESS (ADC123_BASE_ADDRESS + 0x200)

// regular data register
#define ADC1_DR_REGISTER (ADC1_BASE_ADDRESS + 0x4C)
#define ADC2_DR_REGISTER (ADC2_BASE_ADDRESS + 0x4C)
#define ADC3_DR_REGISTER (ADC3_BASE_ADDRESS + 0x4C)

/* function
definitions----------------------------------------------------------------*/
uint16_t adcDmaDataStorageBuffer[3];
uint16_t adcDmaDataStorageBufferforone;

void initDMAForADC3_1channel( void )
{
    uint32_t * reg_pointer;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);

    /* Configure Stream 0 to use channel 2 (ADC3) */
    reg_pointer = (uint32_t *)DMA2_S0CR_REGISTER;
    *reg_pointer = DMA_SxCR_CHANNEL_2_SELECT + DMA_SxCR_MSIZE_HALF_WORD
    + DMA_SxCR_PSIZE_HALF_WORD  + DMA_SxCR_DIR_PERTOMEM
    + DMA_SxCR_CIRC_ENABLE;

    /* we will transfer 1 data registers for 1 channel of ADC */
```

```
    reg_pointer = (uint32_t *)DMA2_S0NDTR_REGISTER;
    *reg_pointer = 1;

    /* we will transfer the ADC3 data register */
    reg_pointer = (uint32_t *)DMA2_S0PAR_REGISTER;
    *reg_pointer = ADC3_DR_REGISTER;

    /* we will transfer to the adcDmaDataStroageBuffer we just created */
    reg_pointer = (uint32_t *) DMA2_S0M0AR_REGISTER;
    *reg_pointer = (uint32_t) &adcDmaDataStorageBufferforone;
}

void initDMAForADC3_3channels( void )
{
    uint32_t * reg_pointer;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);

    /* Configure Stream 0 to use channel 2 (ADC3) */
    reg_pointer = (uint32_t *)DMA2_S0CR_REGISTER;
    *reg_pointer = DMA_SxCR_CHANNEL_2_SELECT + DMA_SxCR_MSIZE_HALF_WORD
    + DMA_SxCR_PSIZE_HALF_WORD + DMA_SxCR_MINC_INCREMENT +
DMA_SxCR_DIR_PERTOMEM
    + DMA_SxCR_CIRC_ENABLE;

    /* we will transfer 3 data registers for 3 channels of ADC */
    reg_pointer = (uint32_t *)DMA2_S0NDTR_REGISTER;
    *reg_pointer = 3;

    /* we will transfer the ADC3 data register */
    reg_pointer = (uint32_t *)DMA2_S0PAR_REGISTER;
    *reg_pointer = ADC3_DR_REGISTER;

    /* we will transfer to the adcDmaDataStroageBuffer we just created */
    reg_pointer = (uint32_t *) DMA2_S0M0AR_REGISTER;
    *reg_pointer = (uint32_t) & adcDmaDataStorageBuffer[0];
}

void enableDMAForAdc3_3channels( void )
{
```

```
    uint32_t * reg_pointer;
    reg_pointer = (uint32_t *) DMA2_S0CR_REGISTER;
    *reg_pointer = *reg_pointer | DMA_SxCR_STREAM_ENABLE;
}

uint16_t returnADC3StoredValue(uint8_t index)
{
    return adcDmaDataStorageBuffer[index];
}

uint16_t returnADC3StoredValueforone(void)
{
    return adcDmaDataStorageBufferforone;
}
```

hardware_stm_dma.h

```
/* Define to prevent recursive inclusion
-------------------------------------*/
#ifndef __HARDWARE_STM_DMA_H_
#define __HARDWARE_STM_DMA_H_

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes
------------------------------------------------------------------*/
#include "main.h"


/*Function
definitions-----------------------------------------------------------*/
void initDMAForADC3_1channel( void );
void initDMAForADC3_3channels( void );
void enableDMAForAdc3_3channels( void );
uint16_t returnADC3StoredValue(uint8_t index);
uint16_t returnADC3StoredValueforone(void);


#ifdef __cplusplus
```

```
}
#endif

#endif /*__GPIO_H */
```

hardware_stm_gpio.c

```
/**

******************************************************************************
  * @file    hardware_stm_gpio.c
  * @author  mortamar@andrew.cmu.edu
  * @version 1.0
  * @date    Septembr-2021
  * @brief   Controls STM32F446ze GPIO

******************************************************************************
  */

#include "hardware_stm_gpio.h"
#include "stm32f4xx_rcc_mort.h"


//led 1 is connected to PB0.
// GPIO B addresses: 0x4002 0400 - 0x4002 07FF
// GPIO C addresses: 0x4002 0800 - 0x4002 0BFF


/* MACRO
definitions-----------------------------------------------------------*/
//Port B addresses:
#define PORTB_BASE_ADDRESS ((uint32_t)0x40020400)        //The first
address in memory corresponding to Port B (this is in the user manual!)
// I gave you the first one, now you fill in the rest, check in the user
manual what is the offset from the base address for each register!
#define PORTB_MODER_REGISTER (PORTB_BASE_ADDRESS + 0x00) //replace the
question mark with the correct offset!
```

```
#define PORTB_OTYPER_REGISTER (PORTB_BASE_ADDRESS + 0x04)
#define PORTB_OSPEEDR_REGISTER (PORTB_BASE_ADDRESS + 0x08)
#define PORTB_PUPDR_REGISTER (PORTB_BASE_ADDRESS + 0x0C)
#define PORTB_IDR_REGISTER (PORTB_BASE_ADDRESS + 0x10)
#define PORTB_ODR_REGISTER (PORTB_BASE_ADDRESS + 0x14)
#define PORTB_BSRRL_REGISTER (PORTB_BASE_ADDRESS + 0x18)
#define PORTB_BSRR_REGISTER (PORTB_BASE_ADDRESS + 0x18)
#define PORTB_BSRRH_REGISTER (PORTB_BASE_ADDRESS + 0x1A)
#define PORTB_LCKR_REGISTER (PORTB_BASE_ADDRESS + 0x1C)
#define PORTB_AFR1_REGISTER (PORTB_BASE_ADDRESS + 0x20)
#define PORTB_AFR2_REGISTER (PORTB_BASE_ADDRESS + 0x24)
//#define PORTB_OSPEEDR_REGISTER (PORTB_BASE_ADDRESS + 0x08)

//Port C addresses:
#define PORTC_BASE_ADDRESS ((uint32_t)0x40020800)
#define PORTC_MODER_REGISTER (PORTC_BASE_ADDRESS + 0x00)
#define PORTC_OTYPER_REGISTER (PORTC_BASE_ADDRESS + 0x04)
#define PORTC_OSPEEDR_REGISTER (PORTC_BASE_ADDRESS + 0x08)
#define PORTC_PUPDR_REGISTER (PORTC_BASE_ADDRESS + 0x0C)
#define PORTC_IDR_REGISTER (PORTC_BASE_ADDRESS + 0x10)
#define PORTC_ODR_REGISTER (PORTC_BASE_ADDRESS + 0x14)
//#define PORTC_BSRRL_REGISTER (PORTB_BASE_ADDRESS + 0x18)
#define PORTC_BSRR_REGISTER (PORTC_BASE_ADDRESS + 0x18)
//#define PORTC_BSRRH_REGISTER (PORTB_BASE_ADDRESS + 0x1A)
#define PORTC_LCKR_REGISTER (PORTC_BASE_ADDRESS + 0x1C)
#define PORTC_AFRL_REGISTER (PORTC_BASE_ADDRESS + 0x20)
#define PORTC_AFRH_REGISTER (PORTC_BASE_ADDRESS + 0x24)
//#define PORTC_OSPEEDR_REGISTER (PORTB_BASE_ADDRESS + 0x08)

//Port F addresses:
#define PORTF_BASE_ADDRESS ((uint32_t)0x40021400)
#define PORTF_MODER_REGISTER (PORTF_BASE_ADDRESS + 0x00)
#define PORTF_OTYPER_REGISTER (PORTF_BASE_ADDRESS + 0x04)
#define PORTF_OSPEEDR_REGISTER (PORTF_BASE_ADDRESS + 0x08)
#define PORTF_PUPDR_REGISTER (PORTF_BASE_ADDRESS + 0x0C)
#define PORTF_IDR_REGISTER (PORTF_BASE_ADDRESS + 0x10)
#define PORTF_ODR_REGISTER (PORTF_BASE_ADDRESS + 0x14)
#define PORTF_BSRR_REGISTER (PORTF_BASE_ADDRESS + 0x18)
#define PORTF_LCKR_REGISTER (PORTF_BASE_ADDRESS + 0x1C)
#define PORTF_AFRL_REGISTER (PORTF_BASE_ADDRESS + 0x20)
```

```
#define PORTF_AFRH_REGISTER (PORTF_BASE_ADDRESS + 0x24)

//flags MODER Register:
#define PIN_0_INP 0b00
#define PIN_0_OUTP 0b01
#define PIN_0_ALTFUN 0b10
#define PIN_0_ANALOG 0b11

#define PIN_1_INP (uint32_t)(PIN_0_INP<<2)
#define PIN_1_OUTP (uint32_t)(PIN_0_OUTP<<2)
#define PIN_1_ALTFUN (uint32_t)(PIN_0_ALTFUN<<2)
#define PIN_1_ANALOG (uint32_t)(PIN_0_ANALOG<<2)

#define PIN_5_INP (uint32_t)(PIN_0_INP<<10)
#define PIN_5_OUTP (uint32_t)(PIN_0_OUTP<<10)
#define PIN_5_ALTFUN (uint32_t)(PIN_0_ALTFUN<<10)
#define PIN_5_ANALOG (uint32_t)(PIN_0_ANALOG<<10)

#define PIN_6_INP 0x0000
#define PIN_6_OUTP 0x1000
#define PIN_6_ALTFUN 0x2000
#define PIN_6_ANALOG 0x3000

#define PIN_7_INP (uint32_t)(PIN_0_INP<<14)
#define PIN_7_OUTP (uint32_t)(PIN_0_OUTP<<14)
#define PIN_7_ALTFUN (uint32_t)(PIN_0_ALTFUN<<14)
#define PIN_7_ANALOG (uint32_t)(PIN_0_ANALOG<<14)

#define PIN_8_INP (uint32_t)(PIN_0_INP<<16)
#define PIN_8_OUTP (uint32_t)(PIN_0_OUTP<<16)
#define PIN_8_ALTFUN (uint32_t)(PIN_0_ALTFUN<<16)
#define PIN_8_ANALOG (uint32_t)(PIN_0_ANALOG<<16)

#define PIN_9_INP (uint32_t)(PIN_0_INP<<18)
#define PIN_9_OUTP (uint32_t)(PIN_0_OUTP<<18)
#define PIN_9_ALTFUN (uint32_t)(PIN_0_ALTFUN<<18)
#define PIN_9_ANALOG (uint32_t)(PIN_0_ANALOG<<18)

#define PIN_12_INP (uint32_t)(PIN_0_INP<<24)
#define PIN_12_OUTP (uint32_t)(PIN_0_OUTP<<24)
```

```
#define PIN_12_ALTFUN (uint32_t)(PIN_0_ALTFUN<<24)
#define PIN_12_ANALOG (uint32_t)(PIN_0_ANALOG<<24)

#define PIN_13_INP (uint32_t)(PIN_0_INP<<26)
#define PIN_13_OUTP (uint32_t)(PIN_0_OUTP<<26)
#define PIN_13_ALTFUN (uint32_t)(PIN_0_ALTFUN<<26)
#define PIN_13_ANALOG (uint32_t)(PIN_0_ANALOG<<26)

//flags OTYPER Register:
#define PIN_0_OUTP_PUSHPULL 0b0
#define PIN_0_OUTP_OPENDRAIN 0b1

#define PIN_5_OUTP_PUSHPULL (uint32_t)(0b0<<5)
#define PIN_5_OUTP_OPENDRAIN (uint32_t)(0b1<<5)

#define PIN_6_OUTP_PUSHPULL 0x00
#define PIN_6_OUTP_OPENDRAIN (uint32_t)(0b1<<6)

#define PIN_7_OUTP_PUSHPULL (uint32_t)(0b0<<7)
#define PIN_7_OUTP_OPENDRAIN (uint32_t)(0b1<<7)

#define PIN_8_OUTP_PUSHPULL (uint32_t)(0b0<<8)
#define PIN_8_OUTP_OPENDRAIN (uint32_t)(0b1<<8)

#define PIN_9_OUTP_PUSHPULL (uint32_t)(0b0<<9)
#define PIN_9_OUTP_OPENDRAIN (uint32_t)(0b1<<9)

#define PIN_12_OUTP_PUSHPULL (uint32_t)(0b0<<12)
#define PIN_12_OUTP_OPENDRAIN (uint32_t)(0b1<<12)

#define PIN_13_OUTP_PUSHPULL (uint32_t)(0b0<<13)
#define PIN_13_OUTP_OPENDRAIN (uint32_t)(0b1<<13)

//flags OSPEEDR Register:
#define PIN_0_HIGHSPEED 0b11

#define PIN_5_HIGHSPEED (uint32_t)(PIN_0_HIGHSPEED << 10)

#define PIN_6_HIGHSPEED 0x3000
```

```
#define PIN_7_HIGHSPEED (uint32_t)(PIN_0_HIGHSPEED << 14)

#define PIN_12_HIGHSPEED (uint32_t)(PIN_0_HIGHSPEED << 24)

#define PIN_13_HIGHSPEED (uint32_t)(PIN_0_HIGHSPEED << 26)

//flags PUPDR Register:
#define PIN_0_NO 0b00
#define PIN_0_PULLUP 0b01
#define PIN_0_PULLDOWN 0b10

#define PIN_3_NO (uint32_t)(0b00<<6)
#define PIN_3_PULLUP (uint32_t)(0b01 << 6)
#define PIN_3_PULLDOWN (uint32_t)(0b10 << 6)

#define PIN_5_NO (uint32_t)(0b00<<10)
#define PIN_5_PULLUP (uint32_t)(0b01 << 10)
#define PIN_5_PULLDOWN (uint32_t)(0b10 << 10)

#define PIN_6_NO (uint32_t)(0b00<<12)
#define PIN_6_PULLUP (uint32_t)(0b01 << 12)
#define PIN_6_PULLDOWN 0x2000

#define PIN_7_NO (uint32_t)(0b00<<14)
#define PIN_7_PULLUP (uint32_t)(0b01 << 14)
#define PIN_7_PULLDOWN (uint32_t)(0b10 << 14)

#define PIN_8_NO (uint32_t)(0b00<<16)
#define PIN_8_PULLUP (uint32_t)(0b01 << 16)
#define PIN_8_PULLDOWN (uint32_t)(0b10 << 16)

#define PIN_9_NO (uint32_t)(0b00<<18)
#define PIN_9_PULLUP (uint32_t)(0b01 << 18)
#define PIN_9_PULLDOWN (uint32_t)(0b10 << 18)

#define PIN_12_NO (uint32_t)(0b00<<24)
#define PIN_12_PULLUP (uint32_t)(0b01 << 24)
#define PIN_12_PULLDOWN (uint32_t)(0b10 << 24)

#define PIN_13_NO (uint32_t)(0b00<<26)
```

```
#define PIN_13_PULLUP (uint32_t)(0b01 << 26)
#define PIN_13_PULLDOWN (uint32_t)(0b10 << 26)

#define PIN_15_NO (uint32_t)(0b00<<30)
#define PIN_15_PULLUP (uint32_t)(0b01 << 30)
#define PIN_15_PULLDOWN (uint32_t)(0b10 << 30)

//input data register:
#define PIN_0_IDR_HIGH 0b1
#define PIN_0_IDR_LOW 0b0

#define PIN_3_IDR_HIGH (uint32_t)(0b1 << 3)
#define PIN_3_IDR_LOW (uint32_t)(0b0 << 3)

#define PIN_6_IDR_HIGH 0x40
#define PIN_6_IDR_LOW (uint32_t)(0b0 << 6)

#define PIN_7_IDR_HIGH (uint32_t)(0b1 << 7)
#define PIN_7_IDR_LOW (uint32_t)(0b0 << 7)

#define PIN_12_IDR_HIGH (uint32_t)(0b1 << 12)
#define PIN_12_IDR_LOW (uint32_t)(0b0 << 12)

#define PIN_13_IDR_HIGH (uint32_t)(0b1 << 13)
#define PIN_13_IDR_LOW (uint32_t)(0b0 << 13)

//flags AFR1 Register:
//#define AFRL1_AFX = 0b(AFX) << 4
#define AFRL0_CLEAR 0b1111
#define AFRL0_AF2 0b0010

#define AFRL5_CLEAR (uint32_t)(AFRL0_CLEAR << 20)
#define AFRL5_AF2 (uint32_t)(AFRL0_AF2 << 20)

#define AFRL6_CLEAR (uint32_t)(0b1111<< 24)
//#define AFRL6_AF2 AFRL0_AF2 << 24
#define AFRL6_AF2 0x2000000

//flags ODR Register:
#define PIN_0_ODR_HIGH 0b1
```

```
#define PIN_0_ODR_LOW 0b0

#define PIN_5_ODR_HIGH (uint32_t)(0b1 << 5)
#define PIN_5_ODR_LOW (uint32_t)(0b0 << 5)

#define PIN_6_ODR_HIGH (uint32_t)(0b1 << 6)
#define PIN_6_ODR_LOW (uint32_t)(0b0 << 6)

#define PIN_12_ODR_HIGH (uint32_t)(0b1 << 12)
#define PIN_12_ODR_LOW (uint32_t)(0b0 << 12)

#define PIN_13_ODR_HIGH (uint32_t)(0b1 << 13)
#define PIN_13_ODR_LOW (uint32_t)(0b0 << 13)

//pin clear
#define PIN_0_2B_CLEAR 0b11
#define PIN_0_1B_CLEAR 0b1

#define PIN_3_2B_CLEAR (uint32_t)(PIN_0_2B_CLEAR << 6)
#define PIN_3_1B_CLEAR (uint32_t)(PIN_0_1B_CLEAR << 3)

#define PIN_5_2B_CLEAR (uint32_t)(PIN_0_2B_CLEAR << 10)
#define PIN_5_1B_CLEAR (uint32_t)(PIN_0_1B_CLEAR << 5)

#define PIN_6_2B_CLEAR (uint32_t)(PIN_0_2B_CLEAR << 12)
#define PIN_6_1B_CLEAR (uint32_t)(PIN_0_1B_CLEAR << 6)

#define PIN_7_2B_CLEAR (uint32_t)(PIN_0_2B_CLEAR << 14)
#define PIN_7_1B_CLEAR (uint32_t)(PIN_0_1B_CLEAR << 7)

#define PIN_12_2B_CLEAR (uint32_t)(PIN_0_2B_CLEAR << 24)
#define PIN_12_1B_CLEAR (uint32_t)(PIN_0_1B_CLEAR << 12)

#define PIN_13_2B_CLEAR (uint32_t)(PIN_0_2B_CLEAR << 26)
#define PIN_13_1B_CLEAR (uint32_t)(PIN_0_1B_CLEAR << 13)

#define PIN_15_2B_CLEAR (uint32_t)(PIN_0_2B_CLEAR << 30)
#define PIN_15_1B_CLEAR (uint32_t)(PIN_0_1B_CLEAR << 15)

#define SET_BIT(BITFIELD, N) (BITFIELD |= ((uint32_t)0x1 << N))
```

```
#define SET_BITS(BITFIELD, N, M) (BITFIELD |= ((uint32_t)0x1 << N) |
((uint32_t)0x1 << M))
#define CLEAR_BIT(BITFIELD, N) (BITFIELD &= ~((uint32_t)0x1 << N))
#define CLEAR_BITS(BITFIELD, N, M) (BITFIELD &= ~(((uint32_t)0x1 << N)) |
((uint32_t)0x1 << M))
#define EXTRACT_BIT(BITFIELD,N) ((BITFIELD &(((uint32_t)0x1 << N)))>>N)

#define SET_BIT16(BITFIELD, N) (BITFIELD |= ((uint16_t)0x1 << N))
#define SET_BITS16(BITFIELD, N, M) (BITFIELD |= ((uint16_t)0x1 << N) |
((uint16_t)0x1 << M))
#define CLEAR_BIT16(BITFIELD, N) (BITFIELD &= ~((uint16_t)0x1 << N))
#define CLEAR_BITS16(BITFIELD, N, M) (BITFIELD &= ~(((uint16_t)0x1 << N))
| ((uint16_t)0x1 << M))

/* function
definitions----------------------------------------------------------------*/

void initGpioB12AsInput( void )
{
      uint32_t  * reg_pointer;
      /* GPIOB Peripheral clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

    /* GPIOB Pin 12 as input*/
    reg_pointer = (uint32_t *)PORTB_MODER_REGISTER;
    // clear pin 12
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_12_2B_CLEAR));
    // Pin 12 as input
    *reg_pointer = *reg_pointer | PIN_12_INP;

    /*PUSH-PULL Pin*/
    reg_pointer = (uint32_t*)PORTB_OTYPER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_12_1B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_12_OUTP_PUSHPULL;

    /*GPIOB pin 12 high speed */
    reg_pointer = (uint32_t*)PORTB_OSPEEDR_REGISTER;
    *reg_pointer = *reg_pointer | PIN_12_HIGHSPEED;

    /*Configure pulled-down*/
```

```
    reg_pointer = (uint32_t*)PORTB_PUPDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_12_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_12_PULLDOWN;
}

void initGpioB13AsInput( void )
{
      uint32_t  * reg_pointer;
      /* GPIOB Peripheral clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

    /* GPIOB Pin 13 as input*/
    reg_pointer = (uint32_t *)PORTB_MODER_REGISTER;
    // clear pin 13
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_13_2B_CLEAR));
    // Pin 13 as input
    *reg_pointer = *reg_pointer | PIN_13_INP;

    /*PUSH-PULL Pin*/
    reg_pointer = (uint32_t*)PORTB_OTYPER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_13_1B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_13_OUTP_PUSHPULL;

    /*GPIOB pin 13 high speed */
    reg_pointer = (uint32_t*)PORTB_OSPEEDR_REGISTER;
    *reg_pointer = *reg_pointer | PIN_13_HIGHSPEED;

    /*Configure pulled-down*/
    reg_pointer = (uint32_t*)PORTB_PUPDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_13_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_13_PULLDOWN;
}

void initGpioC6AsInput( void )
{
      uint32_t  * reg_pointer;
      /* GPIOC Peripheral clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

    /* GPIOC Pin 6 as input*/
```

```
    reg_pointer = (uint32_t *)PORTC_MODER_REGISTER;
    //clear pin 6
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_6_2B_CLEAR));
    // Pin 6 as input
    *reg_pointer = *reg_pointer | PIN_6_INP;

    /*PUSH-PULL Pin*/
    reg_pointer = (uint32_t*)PORTC_OTYPER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_6_1B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_6_OUTP_PUSHPULL;

    /*GPIOC pin 6 high speed */
    reg_pointer = (uint32_t*)PORTC_OSPEEDR_REGISTER;
    *reg_pointer = *reg_pointer | PIN_6_HIGHSPEED;

    /*Configure pulled-down*/
    reg_pointer = (uint32_t*)PORTC_PUPDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_6_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_6_PULLDOWN;
}

void initGpioC7AsInput( void )
{
      uint32_t  * reg_pointer;
      /* GPIOC Peripheral clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

    /* GPIOC Pin 7 as input*/
    reg_pointer = (uint32_t *)PORTC_MODER_REGISTER;
    //clear pin 7
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_7_2B_CLEAR));
    // Pin 7 as input
    *reg_pointer = *reg_pointer | PIN_7_INP;

    /*PUSH-PULL Pin*/
    reg_pointer = (uint32_t*)PORTC_OTYPER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_7_1B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_7_OUTP_PUSHPULL;

    /*GPIOC pin 7 high speed */
```

```
    reg_pointer = (uint32_t*)PORTC_OSPEEDR_REGISTER;
    *reg_pointer = *reg_pointer | PIN_7_HIGHSPEED;

    /*Configure pulled-down*/
    reg_pointer = (uint32_t*)PORTC_PUPDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_7_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_7_PULLDOWN;
}

void initGpioFAsInput(uint16_t v)
{
    if (v > 15){
        return;
    }
    else {
          uint32_t  * reg_pointer;
          /* GPIOF Peripheral clock enable */
        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);

        /* GPIOF configured as input */
        reg_pointer = (uint32_t*)PORTF_MODER_REGISTER;
        CLEAR_BITS(*reg_pointer, v*2, v*2+1);

        /*GPIOF configured as push-pull */
        reg_pointer = (uint32_t*)PORTF_OTYPER_REGISTER;
        CLEAR_BIT(*reg_pointer, v);

        /* GPIOF as high speed */
        reg_pointer = (uint32_t*)PORTF_OSPEEDR_REGISTER;
        CLEAR_BITS(*reg_pointer, v*2, v*2+1);
        SET_BITS(*reg_pointer, v*2, v*2+1);
    }
}

uint32_t checkGPIOF(uint16_t v){
 uint32_t valueF;
 uint32_t * reg_pointer, PIN_IDR_HIGH;
 reg_pointer = (uint32_t*)PORTF_IDR_REGISTER;
 PIN_IDR_HIGH  = (uint32_t)(0b1 << v);
 valueF = *reg_pointer & PIN_IDR_HIGH;
```

```
 return valueF;
}

void initGpioBAsOut(uint16_t v, uint16_t hl)
{
    if (v > 15){
        return;
    }
    else {
          uint32_t  * reg_pointer;
          /* GPIOB Peripheral clock enable */
        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

        /* GPIOB0 configured as output */
        reg_pointer = (uint32_t*)PORTB_MODER_REGISTER;
        CLEAR_BITS(*reg_pointer, v*2, v*2+1);
        SET_BIT(*reg_pointer, v*2);

        /*GPIOB0 configured as push-pull */
        reg_pointer = (uint32_t*)PORTB_OTYPER_REGISTER;
        CLEAR_BIT(*reg_pointer, v);

        /*GPIOB0 configured floating */
        reg_pointer = (uint32_t*)PORTB_PUPDR_REGISTER;
        CLEAR_BITS(*reg_pointer, v*2, v*2+1);

        /* GPIOB0 driven high to start out with */
        reg_pointer = (uint32_t*)PORTB_ODR_REGISTER;
        CLEAR_BIT(*reg_pointer, v);
        if (hl){
            SET_BIT(*reg_pointer, v);
        }
    }
}

void writeGPIOB(uint16_t v, uint16_t hl){
    uint32_t  * reg_pointer;
    reg_pointer = (uint32_t*)PORTB_ODR_REGISTER;
    CLEAR_BIT(*reg_pointer, v);
    if (hl){
```

```
        SET_BIT(*reg_pointer, v);
    }
}

void initGpioB0AsOutput( void )
{
      uint32_t  * reg_pointer;
      /* GPIOB Peripheral clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

    /* GPIOB0 configured as output */
    reg_pointer = (uint32_t*)PORTB_MODER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_0_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_0_OUTP;

    /*GPIOB0 configured as push-pull */
    reg_pointer = (uint32_t*)PORTB_OTYPER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_0_1B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_0_OUTP_PUSHPULL;

    /*GPIOB0 configured floating */
    reg_pointer = (uint32_t*)PORTB_PUPDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_0_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_0_NO;

    /* GPIOB0 driven high to start out with */
    reg_pointer = (uint32_t*)PORTB_ODR_REGISTER;
    *reg_pointer = *reg_pointer | PIN_0_ODR_HIGH;
}


void initGpioB5AsOutput( void )
{
      uint32_t  * reg_pointer;
      /* GPIOB Peripheral clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

    /* GPIOB5 configured as output */
    reg_pointer = (uint32_t*)PORTB_MODER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_5_2B_CLEAR));
```

```
    *reg_pointer = *reg_pointer | PIN_5_OUTP;

    /*GPIOB5 configured as push-pull */
    reg_pointer = (uint32_t*)PORTB_OTYPER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_5_1B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_5_OUTP_PUSHPULL;

    /*GPIOB5 configured floating */
    reg_pointer = (uint32_t*)PORTB_PUPDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_5_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_5_NO;

    /* GPIOB5 driven High to start out with */
    reg_pointer = (uint32_t*)PORTB_ODR_REGISTER;
    *reg_pointer = *reg_pointer | PIN_5_ODR_HIGH;
}

void initGpioB12AsOutput( void )
{
      uint32_t  * reg_pointer;
      /* GPIOB Peripheral clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

    /* GPIOB12 configured as output */
    reg_pointer = (uint32_t*)PORTB_MODER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_12_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_12_OUTP;

    /*GPIOB12 configured as push-pull */
    reg_pointer = (uint32_t*)PORTB_OTYPER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_12_1B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_12_OUTP_PUSHPULL;

    /*GPIOB12 configured floating */
    reg_pointer = (uint32_t*)PORTB_PUPDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_12_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_12_NO;

    /* GPIOB12 driven High to start out with */
    reg_pointer = (uint32_t*)PORTB_ODR_REGISTER;
```

```
    *reg_pointer = *reg_pointer | PIN_12_ODR_HIGH;
}

void initGpioB13AsOutput( void )
{
      uint32_t  * reg_pointer;
      /* GPIOB Peripheral clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

    /* GPIOB13 configured as output */
    reg_pointer = (uint32_t*)PORTB_MODER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_13_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_13_OUTP;

    /*GPIOB13 configured as push-pull */
    reg_pointer = (uint32_t*)PORTB_OTYPER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_13_1B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_13_OUTP_PUSHPULL;

    /*GPIOB13 configured floating */
    reg_pointer = (uint32_t*)PORTB_PUPDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_13_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_13_NO;

    /* GPIOB13 driven High to start out with */
    reg_pointer = (uint32_t*)PORTB_ODR_REGISTER;
    *reg_pointer = *reg_pointer | PIN_13_ODR_LOW;
}

void initGpioF7AsAnalog( void )
{
      uint32_t  * reg_pointer;
      /* GPIOB Peripheral clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);

    /* GPIOB0 configured as Alternative Func */
    reg_pointer = (uint32_t*)PORTF_MODER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_7_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_7_ANALOG;
```

```
    /*GPIOB0 configured as push-pull */
    reg_pointer = (uint32_t*)PORTF_OTYPER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_7_1B_CLEAR));
    *reg_pointer = *reg_pointer & ((PIN_7_OUTP_PUSHPULL));

    /*GPIOB0 configured floating */
    reg_pointer = (uint32_t*)PORTF_PUPDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_7_2B_CLEAR));
    *reg_pointer = *reg_pointer & ((PIN_7_NO));
}

void initGpioF789AsAnalog( void )
{
      uint32_t  * reg_pointer;
      /* GPIOB Peripheral clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);

    /* GPIOB0 configured as Alternative Func */
    reg_pointer = (uint32_t*)PORTF_MODER_REGISTER;
    *reg_pointer = *reg_pointer | (PIN_7_ANALOG + PIN_8_ANALOG +
PIN_9_ANALOG);

    /*GPIOB0 configured as push-pull */
    reg_pointer = (uint32_t*)PORTF_OTYPER_REGISTER;
    *reg_pointer = *reg_pointer & ((PIN_7_OUTP_PUSHPULL +
PIN_8_OUTP_PUSHPULL + PIN_9_OUTP_PUSHPULL));

    /*GPIOB0 configured floating */
    reg_pointer = (uint32_t*)PORTF_PUPDR_REGISTER;
    *reg_pointer = *reg_pointer & ((PIN_7_NO + PIN_8_NO + PIN_9_NO));
}

void toggleGPIOB0( void )
{
    uint32_t value;
    uint32_t  * reg_pointer;
    //get the current value of the pin
    reg_pointer = (uint32_t*)PORTB_ODR_REGISTER;
    value = *reg_pointer & PIN_0_ODR_HIGH;
```

```
    if (value > 0)
    {
        //if high, clear the bit
        *reg_pointer = *reg_pointer & (~((uint32_t)PIN_0_ODR_HIGH));
    }
    else
    {
        //if low, set the bit
       *reg_pointer = *reg_pointer | PIN_0_ODR_HIGH;
    }
}

void setGPIOB0( void )
{
    uint32_t  * reg_pointer;
    /* GPIOB Pin 0 high */
    reg_pointer = (uint32_t*)PORTB_ODR_REGISTER;
    *reg_pointer = *reg_pointer | PIN_0_ODR_HIGH;
}

void clearGPIOB0( void )
{
    uint32_t  * reg_pointer;
    /* GPIOB Pin 0 low */
    reg_pointer = (uint32_t*)PORTB_ODR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t) 0b01));
}

void setGPIOB5( void )
{
    uint32_t  * reg_pointer;
    /* GPIOB Pin 0 high */
    reg_pointer = (uint32_t*)PORTB_ODR_REGISTER;
    *reg_pointer = *reg_pointer | PIN_5_ODR_HIGH;
}

void clearGPIOB5( void )
{
    uint32_t  * reg_pointer;
    /* GPIOB Pin 0 low */
```

```
    reg_pointer = (uint32_t*)PORTB_ODR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t) PIN_5_ODR_HIGH));
}

uint32_t checkGPIOC6(void)
{
 uint32_t valueC6;
 uint32_t * reg_pointer;
 reg_pointer = (uint32_t*)PORTC_IDR_REGISTER;
 //*reg_pointer = *reg_pointer & 0b1000000;
 valueC6 = *reg_pointer & PIN_6_IDR_HIGH;
 //valueC6 = *reg_pointer >> 6;
 return valueC6;
}

uint32_t checkGPIOC7(void)
{
 uint32_t valueC7;
 uint32_t * reg_pointer;
 reg_pointer = (uint32_t*)PORTC_IDR_REGISTER;
 //*reg_pointer = *reg_pointer & 0b1000000;
 valueC7 = *reg_pointer & PIN_7_IDR_HIGH;
 //valueC6 = *reg_pointer >> 6;
 return valueC7;
}

void ButtonPressC6C7( void ){
    uint32_t StartButton = 0;
    uint32_t StopButton = 0;
    uint16_t pin15 = 15;
    uint16_t pin6 = 6;
    StartButton = checkGPIOC6();
    //StopButton = checkGPIOC7();

    if (StartButton == 0){
        // initialize GPIOB15 as Output high
        initGpioBAsOut(pin15, 1);
        // initialize GPIOB3 as Output high
        initGpioBAsOut(pin6, 1);
    }
```

```
    else if (StartButton == PIN_6_IDR_HIGH){
        // initialize GPIOB15 as Output high
        initGpioBAsOut(pin15, 1);
        // initialize GPIOB3 as Output low
        initGpioBAsOut(pin6, 0);
    }

}

uint32_t checkGPIOB13(void)
{
 uint32_t valueB13;
 uint32_t * reg_pointer;
 reg_pointer = (uint32_t*)PORTB_IDR_REGISTER;
 //*reg_pointer = *reg_pointer & 0b1000000;
 valueB13 = *reg_pointer & PIN_13_IDR_HIGH;
 //valueC6 = *reg_pointer >> 6;
 return valueB13;
}

uint32_t checkGPIOB12(void)
{
 uint32_t valueB12;
 uint32_t * reg_pointer;
 reg_pointer = (uint32_t*)PORTB_IDR_REGISTER;
 //*reg_pointer = *reg_pointer & 0b1000000;
 valueB12 = *reg_pointer & PIN_12_IDR_HIGH;
 //valueC6 = *reg_pointer >> 6;
 return valueB12;
}

void initGpioB0MappedToTim3Chn3AsOutput( void )
{
      uint32_t  * reg_pointer;
      /* GPIOB Peripheral clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

    /* GPIOB0 configured as Alternative Func */
    reg_pointer = (uint32_t*)PORTB_MODER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_0_2B_CLEAR));
```

```
    *reg_pointer = *reg_pointer | PIN_0_ALTFUN;

    /*GPIOB0 configured as push-pull */
    reg_pointer = (uint32_t*)PORTB_OTYPER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_0_1B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_0_OUTP_PUSHPULL;

    /*GPIOB0 configured as High speed port */
    reg_pointer = (uint32_t*)PORTB_OSPEEDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_0_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_0_HIGHSPEED;

    /*GPIOB0 configured floating */
    reg_pointer = (uint32_t*)PORTB_PUPDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_0_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_0_NO;

    /*GPIOB0 configured AF2 for AFRL register */
    reg_pointer = (uint32_t*)PORTB_AFR1_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)AFRL0_CLEAR));
    *reg_pointer = *reg_pointer | AFRL0_AF2;
}

void initGpioB5MappedToTim3Chn3AsOutput( void )
{
      uint32_t  * reg_pointer;
      /* GPIOB Peripheral clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

    /* GPIOB5 configured as Alternative Func */
    reg_pointer = (uint32_t*)PORTB_MODER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_5_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_5_ALTFUN;

    /*GPIOB5 configured as push-pull */
    reg_pointer = (uint32_t*)PORTB_OTYPER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_5_1B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_5_OUTP_PUSHPULL;

    /*GPIOB5 configured as High speed port */
```

```
    reg_pointer = (uint32_t*)PORTB_OSPEEDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_5_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_5_HIGHSPEED;

    /*GPIOB5 configured floating */
    reg_pointer = (uint32_t*)PORTB_PUPDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_5_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_5_NO;

    /*GPIOB5 configured AF2 for AFRL register */
    reg_pointer = (uint32_t*)PORTB_AFR1_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)AFRL5_CLEAR));
    *reg_pointer = *reg_pointer | AFRL5_AF2;
}

void initGpioC6MappedToTim3Chn1AsInputCapture( void )
{
      uint32_t  * reg_pointer;
      /* GPIOC Peripheral clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

    /* GPIOC6 configured as Alternative Func */
    reg_pointer = (uint32_t*)PORTC_MODER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_6_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_6_ALTFUN;

    /*GPIOC6 configured as push-pull */
    reg_pointer = (uint32_t*)PORTC_OTYPER_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_6_1B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_6_OUTP_PUSHPULL;

    /*GPIOC6 configured as High speed port */
    reg_pointer = (uint32_t*)PORTC_OSPEEDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_6_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_6_HIGHSPEED;

    /*GPIOC6 configured pulldown so when C6 low in general */
    reg_pointer = (uint32_t*)PORTC_PUPDR_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)PIN_6_2B_CLEAR));
    *reg_pointer = *reg_pointer | PIN_6_PULLDOWN;
```

```
    /*GPIOC6 configured AF2 for AFRL register */
    reg_pointer = (uint32_t*)PORTC_AFRL_REGISTER;
    *reg_pointer = *reg_pointer & (~((uint32_t)AFRL6_CLEAR));
    *reg_pointer = *reg_pointer | AFRL6_AF2;
}
```

hardware_stm_gpio.h

```
/* Define to prevent recursive inclusion
-------------------------------------*/
#ifndef __HARDWARE_STM_GPIO_H_
#define __HARDWARE_STM_GPIO_H_

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes
------------------------------------------------------------------*/
#include "main.h"


/* Macros for
Everyone-----------------------------------------------------------*/
#define PIN_0   0
#define PIN_1   1
#define PIN_2   2
#define PIN_3   3
#define PIN_4   4
#define PIN_5   5
#define PIN_6   6
#define PIN_7   7
#define PIN_8   8
#define PIN_9   9
#define PIN_10  10
```

```
/*Function
definitions----------------------------------------------------------------*/
void initGpioBAsOut(uint16_t v, uint16_t hl);
void initGpioFAsInput(uint16_t v);
uint32_t checkGPIOF(uint16_t v);
void writeGPIOB(uint16_t v, uint16_t hl);
void initGpioB0AsOutput( void );
void initGpioB5AsOutput( void );
void initGpioB12AsOutput( void );
void initGpioB13AsOutput( void );
void initGpioB12AsInput( void );
void initGpioB13AsInput( void );
void toggleGPIOB0( void );
void setGPIOB0( void );
void setGPIOB5( void );
void clearGPIOB0( void );
void clearGPIOB5( void );
void initGpioC6AsInput( void );
void initGpioC7AsInput( void );
void initGpioB0MappedToTim3Chn3AsOutput(void);
void initGpioB5MappedToTim3Chn3AsOutput( void );
void initGpioC6MappedToTim3Chn1AsInputCapture( void );
uint32_t checkGPIOC6(void);
uint32_t checkGPIOC7(void);
void ButtonPressC6C7( void );
uint32_t checkGPIOB12(void);
uint32_t checkGPIOB13(void);
void initGpioF7AsAnalog( void );
void initGpioF789AsAnalog( void );

#ifdef __cplusplus
}
#endif

#endif /*__GPIO_H */
```

hardware_stm_interruptcontroller.c

```
/*#include "debug_mort.h"
//#include "Lab5 Part2/debug_mort.h"
```

```
#include "hardware_stm_gpio.h"
#include "main.h"
//#include
"mbed2/299/TARGET_NUCLEO_F446ZE/TARGET_STM/TARGET_STM32F4/TARGET_NUCLEO_F4
46ZE/device/stm32f4xx.h"
#include "stm32f4xx_rcc_mort.h"
#include "debug_mort.h"
#include "hardware_stm_adc.h"
#include "hardware_stm_dma.h"
#include <cstdint>
#include "mbed.h"
#include "hardware_stm_interruptcontroller.h"*/

#include "debug_mort.h"
#include "hardware_stm_gpio.h"
#include "main.h"
#include "stm32f4xx_rcc_mort.h"
#include "debug_mort.h"
#include "hardware_stm_adc.h"
#include "hardware_stm_dma.h"

/* MACRO
definitions-----------------------------------------------------------*/
#define TIM2_BASE_ADDRESS ((uint32_t)0x40000000)

#define TIM2_CR1_REGISTER_1 (TIM2_BASE_ADDRESS + 0x00)
#define TIM2_CR2_REGISTER_1 (TIM2_BASE_ADDRESS + 0x04)
// Timer 2 status register
#define TIM2_STATUS_REGISTER (TIM2_BASE_ADDRESS + 0x10)
//timer 2 interrupt enable register
#define TIM2_INTERRUPT_ENABLE_REGISTER (TIM2_BASE_ADDRESS + 0x0C)
//flags for interrupt enable register:
//Capture compare enable register
#define TIM2_CAPTURE_COMPARE_ENABLE_REGISTER (TIM2_BASE_ADDRESS + 0x20)

//Capture compare mode registers
#define TIM2_CAPTURE_COMPARE_MODE_1_REGISTER (TIM2_BASE_ADDRESS + 0x18)
#define TIM2_CAPTURE_COMPARE_MODE_2_REGISTER (TIM2_BASE_ADDRESS + 0x1C)

#define TIM2_CNT (TIM2_BASE_ADDRESS + 0x24)
```

```
// Compare, autoreload and Prescaler registers
#define TIM2_COMPARE_1_REGISTER (TIM2_BASE_ADDRESS + 0x34)
#define TIM2_COMPARE_2_REGISTER (TIM2_BASE_ADDRESS + 0x38)
#define TIM2_COMPARE_3_REGISTER (TIM2_BASE_ADDRESS + 0x3C)
#define TIM2_COMPARE_4_REGISTER (TIM2_BASE_ADDRESS + 0x40)
#define TIM2_PRESCALER_REGISTER (TIM2_BASE_ADDRESS + 0x28)
#define TIM2_AUTORELOAD_REGISTER (TIM2_BASE_ADDRESS + 0X2C)

#define TIM2_INTERRUPT_BIT (0x10000000)

#define TIM_CCMR2_OC3M_ACTIVE_MATCH (0x10)

#define TIM3_BASE_ADDRESS ((uint32_t)0x40000400)
// Timer 3 control register 1
#define TIM3_CR1_REGISTER_1 (TIM3_BASE_ADDRESS + 0x00)
#define TIM3_CR2_REGISTER_1 (TIM3_BASE_ADDRESS + 0x04)
//flags for CR1 register:
#define COUNTER_ENABLE_BIT (uint16_t)0x01
// Timer 3 status register
#define TIM3_STATUS_REGISTER (TIM3_BASE_ADDRESS + 0x10)
//flags for Status register:
#define TIM_UIF 0x01 //timer 3 overflow flag
#define TIM_CH1_CC1IF 0x02 //timer channel 1 capture/compare event
#define TIM_CH3_CC3IF 0x8 //timer channel 3 capture/compare event
//timer 3 interrupt enable register
#define TIM3_INTERRUPT_ENABLE_REGISTER (TIM3_BASE_ADDRESS + 0x0C)
//flags for interrupt enable register:
#define TIM_CH3_CC_INTERRUPT_ENABLE 0x8 //timer channel 3 capture/compare
interrupt
#define TIM_UPDATE_INTERRUPT_ENABLE 0x1 //timer overflow or event
interrupt
//Capture compare enable register
#define TIM3_CAPTURE_COMPARE_ENABLE_REGISTER (TIM3_BASE_ADDRESS + 0x20)
//flags for TIM3_CCER registers for output:
#define TIM3_CCER_CC3E (0x0100)
#define TIM3_CCER_CC2E (0x10)
//Capture compare mode registers
#define TIM3_CAPTURE_COMPARE_MODE_1_REGISTER (TIM3_BASE_ADDRESS + 0x18)
#define TIM3_CAPTURE_COMPARE_MODE_2_REGISTER (TIM3_BASE_ADDRESS + 0x1C)
```

```
//flags for Capture compare mode register
#define TIM_CCMR13_OCPE (0b00001000) // enable preload register channels 1
and 3
#define TIM_CCMR1_OC2PE (0x800)
// Compare, autoreload and Prescaler registers
#define TIM3_COMPARE_1_REGISTER (TIM3_BASE_ADDRESS + 0x34)
#define TIM3_COMPARE_2_REGISTER (TIM3_BASE_ADDRESS + 0x38)
#define TIM3_COMPARE_3_REGISTER (TIM3_BASE_ADDRESS + 0x3C)
#define TIM3_COMPARE_4_REGISTER (TIM3_BASE_ADDRESS + 0x40)
#define TIM3_PRESCALER_REGISTER (TIM3_BASE_ADDRESS + 0x28)
#define TIM3_AUTORELOAD_REGISTER (TIM3_BASE_ADDRESS + 0X2C)

#define TIM3_CNT (TIM3_BASE_ADDRESS +0x24)

#define TIM4_BASE_ADDRESS ((uint32_t)0x40000800)
// Timer 4 status register
#define TIM4_CR1_REGISTER_1 (TIM4_BASE_ADDRESS + 0x00)
#define TIM4_STATUS_REGISTER (TIM4_BASE_ADDRESS + 0x10)
#define TIM4_PRESCALER_REGISTER (TIM4_BASE_ADDRESS + 0x28)
#define TIM4_AUTORELOAD_REGISTER (TIM4_BASE_ADDRESS + 0X2C)
#define TIM4_CAPTURE_COMPARE_MODE_2_REGISTER (TIM4_BASE_ADDRESS + 0x1C)
#define TIM4_COMPARE_3_REGISTER (TIM4_BASE_ADDRESS + 0x3C)
//Capture compare enable register
#define TIM4_CAPTURE_COMPARE_ENABLE_REGISTER (TIM4_BASE_ADDRESS + 0x20)
#define TIM4_CNT (TIM4_BASE_ADDRESS + 0x24)
#define TIM4_CCR3 (TIM4_BASE_ADDRESS + 0x3C)

//TIMER5 stuff
#define TIM5_BASE_ADDRESS ((uint32_t)0x40000C00)

#define TIM5_CR1_REGISTER_1 (TIM5_BASE_ADDRESS + 0x00)
#define TIM5_STATUS_REGISTER (TIM5_BASE_ADDRESS + 0x10)
#define TIM5_PRESCALER_REGISTER (TIM5_BASE_ADDRESS + 0x28)
#define TIM5_AUTORELOAD_REGISTER (TIM5_BASE_ADDRESS + 0X2C)
#define TIM5_CAPTURE_COMPARE_MODE_1_REGISTER (TIM5_BASE_ADDRESS + 0x18)
#define TIM5_CAPTURE_COMPARE_MODE_2_REGISTER (TIM5_BASE_ADDRESS + 0x1C)
#define TIM5_COMPARE_1_REGISTER (TIM5_BASE_ADDRESS + 0x34)
#define TIM5_COMPARE_3_REGISTER (TIM5_BASE_ADDRESS + 0x3C)
//Capture compare enable register
#define TIM5_CAPTURE_COMPARE_ENABLE_REGISTER (TIM5_BASE_ADDRESS + 0x20)
```

```
#define TIM5_CNT (TIM5_BASE_ADDRESS + 0x24)
#define TIM5_CCR3 (TIM5_BASE_ADDRESS + 0x3C)
#define TIM5_INTERRUPT_BIT (0x40000)
#define TIM5_INTERRUPT_ENABLE_REGISTER (TIM5_BASE_ADDRESS + 0x0C)


#define TIM6_BASE_ADDRESS ((uint32_t)0x40001000)

#define TIM6_CR1_REGISTER_1 (TIM6_BASE_ADDRESS + 0x00)
#define TIM6_STATUS_REGISTER (TIM6_BASE_ADDRESS + 0x10)
#define TIM6_PRESCALER_REGISTER (TIM6_BASE_ADDRESS + 0x28)
#define TIM6_AUTORELOAD_REGISTER (TIM6_BASE_ADDRESS + 0X2C)
#define TIM6_CAPTURE_COMPARE_MODE_1_REGISTER (TIM6_BASE_ADDRESS + 0x18)
#define TIM6_CAPTURE_COMPARE_MODE_2_REGISTER (TIM6_BASE_ADDRESS + 0x1C)
#define TIM6_COMPARE_1_REGISTER (TIM6_BASE_ADDRESS + 0x34)
#define TIM6_COMPARE_3_REGISTER (TIM6_BASE_ADDRESS + 0x3C)
//Capture c6mpare enable register
#define TIM6_CAPTURE_COMPARE_ENABLE_REGISTER (TIM6_BASE_ADDRESS + 0x20)
#define TIM6_CNT (TIM6_BASE_ADDRESS + 0x24)
#define TIM6_CCR3 (TIM6_BASE_ADDRESS + 0x3C)
#define TIM6_INTERRUPT_BIT (0x40000)
#define TIM6_INTERRUPT_ENABLE_REGISTER (TIM6_BASE_ADDRESS + 0x0C)


#define TIM7_BASE_ADDRESS ((uint32_t)0x40001400)

#define TIM7_CR1_REGISTER_1 (TIM6_BASE_ADDRESS + 0x00)
#define TIM7_STATUS_REGISTER (TIM6_BASE_ADDRESS + 0x10)
#define TIM7_PRESCALER_REGISTER (TIM6_BASE_ADDRESS + 0x28)
#define TIM7_AUTORELOAD_REGISTER (TIM6_BASE_ADDRESS + 0X2C)
#define TIM7_CAPTURE_COMPARE_MODE_1_REGISTER (TIM6_BASE_ADDRESS + 0x18)
#define TIM7_CAPTURE_COMPARE_MODE_2_REGISTER (TIM6_BASE_ADDRESS + 0x1C)
#define TIM7_COMPARE_1_REGISTER (TIM6_BASE_ADDRESS + 0x34)
#define TIM7_COMPARE_3_REGISTER (TIM6_BASE_ADDRESS + 0x3C)
//Capture c7mpare enable register
#define TIM7_CAPTURE_COMPARE_ENABLE_REGISTER (TIM6_BASE_ADDRESS + 0x20)
#define TIM7_CNT (TIM6_BASE_ADDRESS + 0x24)
#define TIM7_CCR3 (TIM6_BASE_ADDRESS + 0x3C)
#define TIM7_INTERRUPT_BIT (0x40000)
#define TIM7_INTERRUPT_ENABLE_REGISTER (TIM6_BASE_ADDRESS + 0x0C)
```

```
/* MACRO
definitions----------------------------------------------------------------*/
#define SYSTEM_CONTROL_BASE_ADDRESS (0xE000E000)
#define NVIC_BASE_ADDRESS (SYSTEM_CONTROL_BASE_ADDRESS + 0x100)
#define NVIC_INTERRUPT_SET_ENABLE_REGISTER_0_31 (NVIC_BASE_ADDRESS)
#define NVIC_INTERRUPT_SET_ENABLE_REGISTER_32_63 (NVIC_BASE_ADDRESS+0x4)
#define NVIC_INTERRUPT_SET_ENABLE_REGISTER_64_95 (NVIC_BASE_ADDRESS+0x8)
#define TIM3_INTERRUPT_BIT (0x20000000)

/* definitions */
#define CCMR_OC3M_PWM_MODE1 (0b1100000)
#define CCMR_OC2M_PWM_MODE1 (0x6000)
#define TIM_CCMR13_OC3M_0 (0b00010000)
#define TIM_CCMR13_OC3M_1 (0b00100000)
#define TIM_CCMR13_OC3M_2 (0b01000000)
#define TIM_CCMR13_OUTPUT (0x00)
#define CCMR_CC3s_OUTPUT (0b00000000)
#define CCMR_CC2s_OUTPUT (0b0000000000)

/* MACRO
definitions----------------------------------------------------------------*/
#define SYSTEM_CONTROL_BASE_ADDRESS (0xE000E000)
#define NVIC_BASE_ADDRESS (SYSTEM_CONTROL_BASE_ADDRESS + 0x100)
#define NVIC_INTERRUPT_SET_ENABLE_REGISTER_0_31 (NVIC_BASE_ADDRESS)
#define NVIC_INTERRUPT_SET_ENABLE_REGISTER_32_63 (NVIC_BASE_ADDRESS+0x4)
#define NVIC_INTERRUPT_SET_ENABLE_REGISTER_64_95 (NVIC_BASE_ADDRESS+0x8)
#define NVIC_INTERRUPT_CLEAR_ENABLE_REGISTER_0_31 (NVIC_BASE_ADDRESS +
0x80)
#define NVIC_INTERRUPT_CLEAR_ENABLE_REGISTER_32_63
(NVIC_INTERRUPT_CLEAR_ENABLE_REGISTER_0_31 + 0x4)
#define NVIC_INTERRUPT_CLEAR_ENABLE_REGISTER_64_95
(NVIC_INTERRUPT_CLEAR_ENABLE_REGISTER_0_31 + 0x8)
#define NVIC_INTERRUPT_SET_PENDING_REGISTER_0_31 (NVIC_BASE_ADDRESS +
0x100)
#define NVIC_INTERRUPT_SET_PENDING_REGISTER_32_63
(NVIC_INTERRUPT_SET_PENDING_REGISTER_0_31 + 0x4)
#define NVIC_INTERRUPT_SET_PENDING_REGISTER_64_95
(NVIC_INTERRUPT_SET_PENDING_REGISTER_0_31 + 0x8)
#define NVIC_INTERRUPT_CLEAR_PENDING_REGISTER_0_31 (NVIC_BASE_ADDRESS +
0x180)
```

```
#define NVIC_INTERRUPT_CLEAR_PENDING_REGISTER_32_63
(NVIC_INTERRUPT_CLEAR_PENDING_REGISTER_0_31 + 0x4)
#define NVIC_INTERRUPT_CLEAR_PENDING_REGISTER_64_95
(NVIC_INTERRUPT_CLEAR_PENDING_REGISTER_0_31 + 0x8)
#define TIM3_INTERRUPT_BIT (0x20000000)
#define EXTI9_5_INTERRUPT_BIT (0x800000)
//For external interrupts:
#define SYSCFG_BASE_ADDRESS ((uint32_t)(0x40013800))
#define SYSCFG_EXTERNAL_INTERRUPT_REGISTER_2 (SYSCFG_BASE_ADDRESS + 0x0C)
#define SYSCFG_EXTERNAL_INTERRUPT_6_BITS ((uint32_t)0xF00) //flags for
External interrupt register 2
#define SYSCFG_EXTERNAL_INTERRUPT_7_BITS ((uint32_t)0xF000)
#define SYSCFG_EXTERNAL_INTERRUPT_6_PORTC ((uint32_t)0x200)
#define SYSCFG_EXTERNAL_INTERRUPT_7_PORTC ((uint32_t)0x2000)
#define SYSCFG_EXTERNAL_INTERRUPT_REGISTER_4 (SYSCFG_BASE_ADDRESS + 0x14)
#define SYSCFG_EXTERNAL_INTERRUPT_12_BITS ((uint32_t)0xF)
#define SYSCFG_EXTERNAL_INTERRUPT_13_BITS ((uint32_t)0xF0)
#define SYSCFG_EXTERNAL_INTERRUPT_12_PORTB ((uint32_t)0x1)
#define SYSCFG_EXTERNAL_INTERRUPT_13_PORTB ((uint32_t)0x10)
//External interrupt controller :
#define EXTERNAL_INTERRUPT_CONTROLLER_BASE_ADDRESS
((uint32_t)(0x40013C00))
#define EXTERNAL_INTERRUPT_CONTROLLER_MASK_REGISTER
(EXTERNAL_INTERRUPT_CONTROLLER_BASE_ADDRESS)
#define EXTERNAL_INTERRUPT_CONTROLLER_MASK_REGISTER_EXTI6 ((uint32_t)0x40)
//flags for external interrupt controller mask register
#define EXTERNAL_INTERRUPT_CONTROLLER_MASK_REGISTER_EXTI7 ((uint32_t)0x80)
#define EXTERNAL_INTERRUPT_CONTROLLER_RTSR
(EXTERNAL_INTERRUPT_CONTROLLER_BASE_ADDRESS+0x08)
#define EXTERNAL_INTERRUPT_CONTROLLER_FTSR
(EXTERNAL_INTERRUPT_CONTROLLER_BASE_ADDRESS+0x0C)
#define EXTERNAL_INTERRUPT_CONTROLLER_RTSR_EXTI6 ((uint32_t)0x40)
#define EXTERNAL_INTERRUPT_CONTROLLER_FTSR
(EXTERNAL_INTERRUPT_CONTROLLER_BASE_ADDRESS+0x0C)
#define EXTERNAL_INTERRUPT_CONTROLLER_FTSR_EXTI6 ((uint32_t)0x40)
#define EXTERNAL_INTERRUPT_CONTROLLER_PENDING_REGISTER
(EXTERNAL_INTERRUPT_CONTROLLER_BASE_ADDRESS+0x14)
#define EXTERNAL_INTERRUPT_CONTROLLER_PENDING_EXTI6 ((uint32_t)0x40)

#define SET_BIT(BITFIELD, N) (BITFIELD |= ((uint32_t)0x1 << N))
```

```
#define SET_BITS(BITFIELD, N, M) (BITFIELD |= ((uint32_t)0x1 << N) |
((uint32_t)0x1 << M))
#define CLEAR_BIT(BITFIELD, N) (BITFIELD &= ~((uint32_t)0x1 << N))
#define CLEAR_BITS(BITFIELD, N, M) (BITFIELD &= ~(((uint32_t)0x1 << N)) |
((uint32_t)0x1 << M))
#define EXTRACT_BIT(BITFIELD,N) ((BITFIELD &(((uint32_t)0x1 << N)))>>N)

#define SET_BIT16(BITFIELD, N) (BITFIELD |= ((uint16_t)0x1 << N))
#define SET_BITS16(BITFIELD, N, M) (BITFIELD |= ((uint16_t)0x1 << N) |
((uint16_t)0x1 << M))
#define CLEAR_BIT16(BITFIELD, N) (BITFIELD &= ~((uint16_t)0x1 << N))
#define CLEAR_BITS16(BITFIELD, N, M) (BITFIELD &= ~(((uint16_t)0x1 << N))
| ((uint16_t)0x1 << M))

#define PORTB_BASE_ADDRESS ((uint32_t)0x40020400)        //The first
address in memory corresponding to Port B (this is in the user manual!)
#define PORTB_IDR_REGISTER (PORTB_BASE_ADDRESS + 0x10)
#define PIN_13_IDR_HIGH (uint32_t)(0b1 << 13)
#define PIN_12_IDR_HIGH (uint32_t)(0b1 << 12)
#define PORTC_BASE_ADDRESS ((uint32_t)0x40020800)
#define PORTC_IDR_REGISTER (PORTC_BASE_ADDRESS + 0x10)
#define BIT(N) ((uint32_t)0x1<<N)

// global variable
uint16_t ButtonClick = 0;
uint16_t maxDMA = 4100;

void enableNVIC_Timer3(void)
{
    uint32_t * reg_pointer_32;
    reg_pointer_32 = (uint32_t *)NVIC_INTERRUPT_SET_ENABLE_REGISTER_0_31;
    *reg_pointer_32 = TIM3_INTERRUPT_BIT;
}

void enableNVIC_Timer2(void)
{
    uint32_t * reg_pointer_32;
    reg_pointer_32 = (uint32_t *)NVIC_INTERRUPT_SET_ENABLE_REGISTER_0_31;
    *reg_pointer_32 = TIM2_INTERRUPT_BIT;
}
```

```
void enableNVIC_Timer5(void)
{
    uint32_t * reg_pointer_32;
    reg_pointer_32 = (uint32_t *)NVIC_INTERRUPT_SET_ENABLE_REGISTER_32_63;
    *reg_pointer_32 = TIM5_INTERRUPT_BIT;
}

void initTimer3ToPWMMode1( void )
{
    uint16_t * reg_pointer_16;
    uint16_t prescalervalue2, autoreloadvalue;
    /* Timer 3 APB clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
    /*enable the interrupt that would go to timer 3*/
    enableNVIC_Timer3();
    /* Compute Prescale and Autorreload */
    // 30kHz = 90MHz / ((prescalervalue+1)*(autoreloadvalue))
    prescalervalue2 = 99; //Frequency of clock is 90 MHz
    autoreloadvalue = 30;
    /* Clear any pending flags in the status register */
    reg_pointer_16 = (uint16_t *)TIM3_STATUS_REGISTER;
    *reg_pointer_16 = 0;
    /* Set Prescale and Autorreload */
    reg_pointer_16 = (uint16_t *)TIM3_PRESCALER_REGISTER;
    *reg_pointer_16 = prescalervalue2;
    reg_pointer_16 = (uint16_t *)TIM3_AUTORELOAD_REGISTER;
    *reg_pointer_16 = autoreloadvalue;
    /* Capture Compare Register 2 OC3M to PWM Mode 1 */
    reg_pointer_16 = (uint16_t *)TIM3_CAPTURE_COMPARE_MODE_1_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | CCMR_OC2M_PWM_MODE1;
    /* Capture Compare Register 2 CC3S to OUTPUT */
    reg_pointer_16 = (uint16_t *)TIM3_CAPTURE_COMPARE_MODE_2_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | CCMR_CC2s_OUTPUT;
    /* Set Compare Value input duty cycle*/
    reg_pointer_16 = (uint16_t *)TIM3_COMPARE_2_REGISTER;
    *reg_pointer_16 = 0; //autoreloadvalue/1.5; //(maxDMA /
(returnADC3StoredValueforone()+1));
    /* Enable Preload Register (Don't HAVE to, but good practice) */
    reg_pointer_16 = (uint16_t *)TIM3_CAPTURE_COMPARE_MODE_1_REGISTER;
```

```
    *reg_pointer_16 = *reg_pointer_16 | TIM_CCMR1_OC1PE;
    /* enable the TIM3 channel 3 counter and keep the default
configuration for channel polarity */
    reg_pointer_16 = (uint16_t *)TIM3_CAPTURE_COMPARE_ENABLE_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM3_CCER_CC2E;
    /* enable interrupt on capture compare channel 3 */
    //reg_pointer_16 = (uint16_t *)TIM3_INTERRUPT_ENABLE_REGISTER;
    //*reg_pointer_16 = (TIM_CH3_CC_INTERRUPT_ENABLE |
TIM_UPDATE_INTERRUPT_ENABLE);
    /* enable timer 3 */
    reg_pointer_16 = (uint16_t *)TIM3_CR1_REGISTER_1;
    *reg_pointer_16 = *reg_pointer_16 | COUNTER_ENABLE_BIT;
}

void setdirection(uint16_t d){
    //sets spiinning direction of motor depending on value of input d
    uint16_t pin15 = 15;
    uint16_t pin6 = 6;
    if(d){
        initGpioBAsOut(pin6, 0);
        initGpioBAsOut(pin15, 1);
    }
    else{
        initGpioBAsOut(pin15, 0);
        initGpioBAsOut(pin6, 1);
    }

}

void pwm(int16_t val){
    //Sets pwm frequency to motor
    //Input val is a number between -100 and 100
    if(val<0){
        //If val is negative make it positive and chnge motor direction
        val=val*(-1);
        setdirection(1);
    }
    else{
        //If val is positive set other direction
        setdirection(0);
```

```
    }
    if(val>85){
        //setting max. allowed pwm frequency to 85% to protect motor
        val=85;
    }
    delT=30*val/100;
    val=100/val; // 1/val is value in percentage

    uint16_t * reg_pointer_16;
    reg_pointer_16 = (uint16_t *)TIM3_COMPARE_2_REGISTER;
    uint16_t autoreloadvalue = 30;
    *reg_pointer_16 = autoreloadvalue/val; //sets ne compare value which
is certain percentage of ARR
}


void initTimer3ToInterruptNOutputCompCh3( void )
{
    //Code from lab4, not used any further

    uint16_t * reg_pointer_16;
    uint16_t prescalervalue2, autoreloadvalue;
    /* Timer 3 APB clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
    /*enable the interrupt that would go to timer 3*/
    enableNVIC_Timer3();
    /* Compute Prescale and Autorreload */
    // LED to 0.25Hz
    // 1Hz = 90MHz / ((prescalervalue+1)*(autoreloadvalue))
    prescalervalue2 = 8999; //Frequency of clock is 90 MHz
    autoreloadvalue = 40000;
    /* Clear any pending flags in the status register */
    reg_pointer_16 = (uint16_t *)TIM3_STATUS_REGISTER;
    *reg_pointer_16 = 0;
    /* Set Prescale and Autorreload */
    reg_pointer_16 = (uint16_t *)TIM3_PRESCALER_REGISTER;
    *reg_pointer_16 = prescalervalue2;
    reg_pointer_16 = (uint16_t *)TIM3_AUTORELOAD_REGISTER;
    *reg_pointer_16 = autoreloadvalue;
    /* Set Compare Value */
```

```
    reg_pointer_16 = (uint16_t *)TIM3_COMPARE_3_REGISTER;
    *reg_pointer_16 = autoreloadvalue/4;
    /* Enable Preload Register (Don't HAVE to, but good practice) */
    reg_pointer_16 = (uint16_t *)TIM3_CAPTURE_COMPARE_MODE_2_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM_CCMR13_OCPE;
    /*enable the TIM3 channel 3 OC3M to active level on match mode for
output compare mode*/
    reg_pointer_16 = (uint16_t *)TIM3_CAPTURE_COMPARE_MODE_2_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM_CCMR13_OUTPUT;
    /*enable the TIM3 channel 3 counter and keep the default configuration
for channel polarity*/
    reg_pointer_16 = (uint16_t *)TIM3_CAPTURE_COMPARE_ENABLE_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM3_CCER_CC3E;
    /*enable interrupt on capture compare channel 3*/
    reg_pointer_16 = (uint16_t *)TIM3_INTERRUPT_ENABLE_REGISTER;
    *reg_pointer_16 = (TIM_CH3_CC_INTERRUPT_ENABLE |
TIM_UPDATE_INTERRUPT_ENABLE);
    /*enable timer 3*/
    reg_pointer_16 = (uint16_t *)TIM3_CR1_REGISTER_1;
    *reg_pointer_16 = *reg_pointer_16 | COUNTER_ENABLE_BIT;
}

uint16_t readcount4(void){
    //to measure time between encoder ticks
    //used for delT in D controller
    uint16_t* reg;
    uint16_t TIM4Val = 0;
    reg= (uint16_t *) TIM4_CNT;
    TIM4Val = *reg;
    *reg = 0; //count value
    return TIM4Val;
}

void initTimer4Ch3( void )
{
    //Initializes TIMER4 which is used for D controller

    uint16_t * reg_pointer_16;
    uint16_t prescalervalue2, autoreloadvalue;
    /* Timer 4 APB clock enable */
```

```
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
    /* Compute Prescale and Autorreload */
    // if we want LED freq to be 1Hz
    // 1Hz = 90MHz / ((prescalervalue+1)*(autoreloadvalue))
    // Frequency of system clock is 90 MHz
    prescalervalue2 = 8999;
    autoreloadvalue = 65000;
    /* Clear any pending flags in the status register */
    reg_pointer_16 = (uint16_t *)TIM4_STATUS_REGISTER;
    *reg_pointer_16 = 0;
    /* Set Prescale and Autorreload */
    reg_pointer_16 = (uint16_t *)TIM4_PRESCALER_REGISTER;
    *reg_pointer_16 = prescalervalue2;
    reg_pointer_16 = (uint16_t *)TIM4_AUTORELOAD_REGISTER;
    *reg_pointer_16 = autoreloadvalue;
    /* Capture Compare Register 2 OC3M to frozen mode */
    reg_pointer_16 = (uint16_t *)TIM4_CAPTURE_COMPARE_MODE_2_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM_CCMR13_OUTPUT;
    /* Set Compare Value */
    reg_pointer_16 = (uint16_t *)TIM4_COMPARE_3_REGISTER;
    *reg_pointer_16 = autoreloadvalue/2; // duty cycle
    reg_pointer_16 = (uint16_t *)TIM4_CR1_REGISTER_1;
    *reg_pointer_16 = *reg_pointer_16 | COUNTER_ENABLE_BIT;
}

void initTimer2ToPWMMode1( void )
{
    uint16_t * reg_pointer_16;
    uint16_t prescalervalue2, autoreloadvalue;
    /* Timer 3 APB clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
    /*enable the interrupt that would go to timer 3*/
    enableNVIC_Timer2();
    /* Compute Prescale and Autorreload */
    // 30kHz = 90MHz / ((prescalervalue+1)*(autoreloadvalue))
    prescalervalue2 = 9999; //Frequency of clock is 90 MHz
    autoreloadvalue = 40000;
    /* Clear any pending flags in the status register */
    reg_pointer_16 = (uint16_t *)TIM2_STATUS_REGISTER;
    *reg_pointer_16 = 0;
```

```
    /* Set Prescale and Autorreload */
    reg_pointer_16 = (uint16_t *)TIM2_PRESCALER_REGISTER;
    *reg_pointer_16 = prescalervalue2;
    reg_pointer_16 = (uint16_t *)TIM2_AUTORELOAD_REGISTER;
    *reg_pointer_16 = autoreloadvalue;
    /* Capture Compare Register 2 OC3M to PWM Mode 1 */
    reg_pointer_16 = (uint16_t *)TIM3_CAPTURE_COMPARE_MODE_1_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | CCMR_OC2M_PWM_MODE1;
    /* Capture Compare Register 2 CC3S to OUTPUT */
    reg_pointer_16 = (uint16_t *)TIM2_CAPTURE_COMPARE_MODE_2_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | CCMR_CC2s_OUTPUT;
    /* Set Compare Value input duty cycle*/
    reg_pointer_16 = (uint16_t *)TIM2_COMPARE_2_REGISTER;
    *reg_pointer_16 = autoreloadvalue/2; //autoreloadvalue/1.5; //(maxDMA
/ (returnADC3StoredValueforone()+1));
    /* Enable Preload Register (Don't HAVE to, but good practice) */
    reg_pointer_16 = (uint16_t *)TIM2_CAPTURE_COMPARE_MODE_2_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM_CCMR1_OC1PE;
    /* enable the TIM3 channel 3 counter and keep the default
configuration for channel polarity */
    reg_pointer_16 = (uint16_t *)TIM2_CAPTURE_COMPARE_ENABLE_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM3_CCER_CC2E;
    /* enable interrupt on capture compare channel 3 */
    //reg_pointer_16 = (uint16_t *)TIM3_INTERRUPT_ENABLE_REGISTER;
    //*reg_pointer_16 = (TIM_CH3_CC_INTERRUPT_ENABLE |
TIM_UPDATE_INTERRUPT_ENABLE);
    /* enable timer 3 */
    reg_pointer_16 = (uint16_t *)TIM2_CR1_REGISTER_1;
    *reg_pointer_16 = *reg_pointer_16 | COUNTER_ENABLE_BIT;
}


void initTimer5Ch3( void ) //Not used, was written for testing
{
    uint16_t * reg_pointer_16;
    uint16_t prescalervalue2, autoreloadvalue;
    /* Timer 4 APB clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM5, ENABLE);
    /* Compute Prescale and Autorreload */
    // if we want LED freq to be 1Hz
```

```
    // 1Hz = 90MHz / ((prescalervalue+1)*(autoreloadvalue))
    // Frequency of system clock is 90 MHz
    enableNVIC_Timer5();
    prescalervalue2 = 8999;
    autoreloadvalue = 65000;
    /* Clear any pending flags in the status register */
    reg_pointer_16 = (uint16_t *)TIM5_STATUS_REGISTER;
    *reg_pointer_16 = 0;
    /* Set Prescale and Autorreload */
    reg_pointer_16 = (uint16_t *)TIM5_PRESCALER_REGISTER;
    *reg_pointer_16 = prescalervalue2;
    reg_pointer_16 = (uint16_t *)TIM5_AUTORELOAD_REGISTER;
    *reg_pointer_16 = autoreloadvalue;
    /* Capture Compare Register 2 OC3M to frozen mode */
    reg_pointer_16 = (uint16_t *)TIM5_CAPTURE_COMPARE_MODE_1_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM_CCMR13_OUTPUT;
    /* Set Compare Value */
    reg_pointer_16 = (uint16_t *)TIM5_COMPARE_1_REGISTER;
    *reg_pointer_16 = autoreloadvalue/2; // duty cycle
    reg_pointer_16 = (uint16_t *)TIM5_CR1_REGISTER_1;
    *reg_pointer_16 = *reg_pointer_16 | COUNTER_ENABLE_BIT;

}

void initTimer6Ch3( void ) //Not used, was written for testing
{
    uint16_t * reg_pointer_16;
    uint16_t prescalervalue2, autoreloadvalue;
    /* Timer 4 APB clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, ENABLE);
    /* Compute Prescale and Autorreload */
    // if we want LED freq to be 1Hz
    // 1Hz = 90MHz / ((prescalervalue+1)*(autoreloadvalue))
    // Frequency of system clock is 90 MHz
    //enableNVIC_Timer5();
    prescalervalue2 = 8999;
    autoreloadvalue = 65000;
    /* Clear any pending flags in the status register */
    reg_pointer_16 = (uint16_t *)TIM6_STATUS_REGISTER;
    *reg_pointer_16 = 0x2;
```

```
    /* Set Prescale and Autorreload */
    reg_pointer_16 = (uint16_t *)TIM6_PRESCALER_REGISTER;
    *reg_pointer_16 = prescalervalue2;
    reg_pointer_16 = (uint16_t *)TIM6_AUTORELOAD_REGISTER;
    *reg_pointer_16 = autoreloadvalue;
    /* Capture Compare Register 2 OC3M to frozen mode */
    reg_pointer_16 = (uint16_t *)TIM6_CAPTURE_COMPARE_MODE_2_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM_CCMR13_OUTPUT | 0x35;
    /* Set Compare Value */
    reg_pointer_16 = (uint16_t *)TIM6_COMPARE_1_REGISTER;
    *reg_pointer_16 = autoreloadvalue/2; // duty cycle
    reg_pointer_16 = (uint16_t *)TIM6_CR1_REGISTER_1;
    *reg_pointer_16 = *reg_pointer_16 | COUNTER_ENABLE_BIT;

}

void initTimer7Ch3( void ) //Not used, was written for testing
{
    uint16_t * reg_pointer_16;
    uint16_t prescalervalue2, autoreloadvalue;
    /* Timer 4 APB clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM7, ENABLE);
    /* Compute Prescale and Autorreload */
    // if we want LED freq to be 1Hz
    // 1Hz = 90MHz / ((prescalervalue+1)*(autoreloadvalue))
    // Frequency of system clock is 90 MHz
    //enableNVIC_Timer5();
    prescalervalue2 = 8999;
    autoreloadvalue = 65000;
    /* Clear any pending flags in the status register */
    reg_pointer_16 = (uint16_t *)TIM7_STATUS_REGISTER;
    *reg_pointer_16 = 0x2;
    /* Set Prescale and Autorreload */
    reg_pointer_16 = (uint16_t *)TIM7_PRESCALER_REGISTER;
    *reg_pointer_16 = prescalervalue2;
    reg_pointer_16 = (uint16_t *)TIM7_AUTORELOAD_REGISTER;
    *reg_pointer_16 = autoreloadvalue;
    /* Capture Compare Register 2 OC3M to frozen mode */
    reg_pointer_16 = (uint16_t *)TIM7_CAPTURE_COMPARE_MODE_2_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM_CCMR13_OUTPUT | 0x35;
```

```
    /* Set Compare Value */
    reg_pointer_16 = (uint16_t *)TIM7_COMPARE_1_REGISTER;
    *reg_pointer_16 = autoreloadvalue/2; // duty cycle
    reg_pointer_16 = (uint16_t *)TIM7_CR1_REGISTER_1;
    *reg_pointer_16 = *reg_pointer_16 | COUNTER_ENABLE_BIT;

}

void initTimer2Ch3( void ) //Not used, was written for testing
{
    uint16_t * reg_pointer_16;
    uint16_t prescalervalue2, autoreloadvalue;
    /* Timer 4 APB clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
    /* Compute Prescale and Autorreload */
    // if we want LED freq to be 1Hz
    // 1Hz = 90MHz / ((prescalervalue+1)*(autoreloadvalue))
    // Frequency of system clock is 90 MHz
    prescalervalue2 = 8999;
    autoreloadvalue = 65000;
    /* Clear any pending flags in the status register */
    reg_pointer_16 = (uint16_t *)TIM2_STATUS_REGISTER;
    *reg_pointer_16 = 0;
    /* Set Prescale and Autorreload */
    reg_pointer_16 = (uint16_t *)TIM2_PRESCALER_REGISTER;
    *reg_pointer_16 = prescalervalue2;
    reg_pointer_16 = (uint16_t *)TIM2_AUTORELOAD_REGISTER;
    *reg_pointer_16 = autoreloadvalue;
    /* Capture Compare Register 2 OC3M to frozen mode */
    reg_pointer_16 = (uint16_t *)TIM2_CAPTURE_COMPARE_MODE_2_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM_CCMR13_OUTPUT;
    /* Set Compare Value */
    reg_pointer_16 = (uint16_t *)TIM2_COMPARE_3_REGISTER;
    *reg_pointer_16 = autoreloadvalue/2; // duty cycle
    reg_pointer_16 = (uint16_t *)TIM2_CR1_REGISTER_1;
    *reg_pointer_16 = *reg_pointer_16 | COUNTER_ENABLE_BIT;
}

uint16_t readcount(void){ //To read current timer3 count
    uint16_t* reg;
```

```
    reg= (uint16_t *) TIM3_CNT;
    return *reg;

}

uint16_t readcount2(void){ //To read current timer2 count
    uint16_t* reg;
    reg= (uint16_t *) TIM2_CNT;
    return *reg;
}

uint16_t readcount5(void){ //To read current timer5 count
    uint16_t* reg;
    reg= (uint16_t *) TIM5_CNT;
    return *reg;

}

uint16_t readcount6(void){ //To read current timer6 count
    uint16_t* reg;
    reg= (uint16_t *) TIM6_CNT;
    return *reg;

}


uint16_t readcount7(void){ //To read current timer7 count
    uint16_t* reg;
    reg= (uint16_t *) TIM7_CNT;
    return *reg;

}


void TIM3_IRQHandler( void )
{
    uint16_t *reg_pointer_16_sr;
    uint16_t *reg_pointer_16_dier;
    reg_pointer_16_sr = (uint16_t*)TIM3_STATUS_REGISTER;
    reg_pointer_16_dier = (uint16_t*)TIM3_INTERRUPT_ENABLE_REGISTER;
```

```
    // check which interrupts fired and if they were supposed to fire,
then clear the flags
    // sp they don't keep firing, then perform actions according to these
interrupts

    //chcek if Output Compare 3 triggered the interrupt:
    if (((*reg_pointer_16_sr & TIM_CH3_CC3IF) == TIM_CH3_CC3IF) &&
((*reg_pointer_16_dier & TIM_CH3_CC_INTERRUPT_ENABLE) ==
TIM_CH3_CC_INTERRUPT_ENABLE))
    {
        //clear interrupt
        *reg_pointer_16_sr = ~((uint16_t)TIM_CH3_CC3IF);
    }
    // check if Overflow triggered the interrupt: i.e. Timer Counter 3 >=
Autoreload value
    if (((*reg_pointer_16_sr & TIM_UIF) == TIM_UIF) &&
((*reg_pointer_16_dier & TIM_UPDATE_INTERRUPT_ENABLE) ==
TIM_UPDATE_INTERRUPT_ENABLE))
    {
        //clear interrupt
        *reg_pointer_16_sr = ~((uint16_t)TIM_UIF);
        //perform action
    }
}



void TIM2_IRQHandler( void ) //Not used, written for testing
{
    debugprint(5);
    uint16_t *reg_pointer_16_sr;
    uint16_t *reg_pointer_16_dier;
    uint16_t *reg_pointer_16;
    uint16_t TIM2Value = 0;
    reg_pointer_16_sr = (uint16_t*)TIM2_STATUS_REGISTER;
    reg_pointer_16_dier = (uint16_t*)TIM2_INTERRUPT_ENABLE_REGISTER;
    reg_pointer_16 = (uint16_t*) TIM2_CNT;
    // check which interrupts fired and if they were supposed to fire,
then clear the flags
    // sp they don't keep firing, then perform actions according to these
```

```
interrupts
    //chcek if Output Compare 3 triggered the interrupt:
    if (((*reg_pointer_16_sr & TIM_CH3_CC3IF) == TIM_CH3_CC3IF) &&
((*reg_pointer_16_dier & TIM_CH3_CC_INTERRUPT_ENABLE) ==
TIM_CH3_CC_INTERRUPT_ENABLE))
    {
        //clear interrupt
        *reg_pointer_16_sr = ~((uint16_t)TIM_CH3_CC3IF);
        clearGPIOB0();
    }
    // check if Overflow triggered the interrupt: i.e. Timer Counter 3 >=
Autoreload value
    if (((*reg_pointer_16_sr & TIM_UIF) == TIM_UIF) &&
((*reg_pointer_16_dier & TIM_UPDATE_INTERRUPT_ENABLE) ==
TIM_UPDATE_INTERRUPT_ENABLE))
    {
        //clear interrupt
        *reg_pointer_16_sr = ~((uint16_t)TIM_UIF);
        //perform action
        setGPIOB0();
        //debugprintHelloWorld();
        TIM2Value = *reg_pointer_16;
        //debugprint(TIM2Value);
    }
}

void TIM5_IRQHandler( void )  //Not used, written for testing
{
    uint16_t *reg_pointer_16_sr;
    uint16_t *reg_pointer_16_dier;
    reg_pointer_16_sr = (uint16_t*)TIM5_STATUS_REGISTER;
    reg_pointer_16_dier = (uint16_t*)TIM5_INTERRUPT_ENABLE_REGISTER;
    // check which interrupts fired and if they were supposed to fire,
then clear the flags
    // sp they don't keep firing, then perform actions according to these
interrupts

    //chcek if Output Compare 3 triggered the interrupt:
    if (((*reg_pointer_16_sr & TIM_CH3_CC3IF) == TIM_CH3_CC3IF) &&
((*reg_pointer_16_dier & TIM_CH3_CC_INTERRUPT_ENABLE) ==
```

```
TIM_CH3_CC_INTERRUPT_ENABLE))
    {
        //clear interrupt
        *reg_pointer_16_sr = ~((uint16_t)TIM_CH3_CC3IF);
    }
    // check if Overflow triggered the interrupt: i.e. Timer Counter 3 >=
Autoreload value
    if (((*reg_pointer_16_sr & TIM_UIF) == TIM_UIF) &&
((*reg_pointer_16_dier & TIM_UPDATE_INTERRUPT_ENABLE) ==
TIM_UPDATE_INTERRUPT_ENABLE))
    {
        //clear interrupt
        debugprint(readcount5());
        *reg_pointer_16_sr = ~((uint16_t)TIM_UIF);
        //perform action
    }
}

void enableEXTI6OnPortC(void)
{
    uint32_t * reg_pointer_32;
    /*Init GPIO 6 C as input*/
    initGpioC6AsInput();
    /*As a test, Init GPIO B0 as output for debugging*/
    initGpioB0AsOutput();
    /* Enable SYSCFG clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
    /*map EXTI6 to port C bit 6*/
    reg_pointer_32 = (uint32_t *)SYSCFG_EXTERNAL_INTERRUPT_REGISTER_2;
    //clear EXTI6
    *reg_pointer_32 = *reg_pointer_32 & ~SYSCFG_EXTERNAL_INTERRUPT_6_BITS;
    //set EXTI6 to Port C
    *reg_pointer_32 = *reg_pointer_32 | SYSCFG_EXTERNAL_INTERRUPT_6_PORTC;
    /*un-mask EXTI6*/
    reg_pointer_32 = (uint32_t
*)EXTERNAL_INTERRUPT_CONTROLLER_MASK_REGISTER;
    *reg_pointer_32 = *reg_pointer_32 |
EXTERNAL_INTERRUPT_CONTROLLER_MASK_REGISTER_EXTI6;
    /*trigger on rising edge*/
    reg_pointer_32 = (uint32_t *)EXTERNAL_INTERRUPT_CONTROLLER_RTSR;
```

```
    *reg_pointer_32 = *reg_pointer_32 |
EXTERNAL_INTERRUPT_CONTROLLER_RTSR_EXTI6;
    /* set the NVIC to respond to EXTI9_5*/
    reg_pointer_32 = (uint32_t *)NVIC_INTERRUPT_SET_ENABLE_REGISTER_0_31;
    *reg_pointer_32 = EXTI9_5_INTERRUPT_BIT;
}

void enableEXTI7OnPortC(void)
{
    //originally written such that another microcontroller running with
STMCube IDE can signal this uController to start
    //Concept not used in final implementation
    //External interrupt functions as intended

    uint32_t * reg_pointer_32;
    /*Init GPIO 7 C as input*/
    initGpioC7AsInput();
    /*As a test, Init GPIO B0 as output for debugging*/
    /* Enable SYSCFG clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
    /*map EXTI7 to port C bit 6*/
    reg_pointer_32 = (uint32_t *)SYSCFG_EXTERNAL_INTERRUPT_REGISTER_2;
    //clear EXTI7
    *reg_pointer_32 = *reg_pointer_32 & ~SYSCFG_EXTERNAL_INTERRUPT_7_BITS;
    //set EXTI7 to Port C
    *reg_pointer_32 = *reg_pointer_32 | SYSCFG_EXTERNAL_INTERRUPT_7_PORTC;
    /*un-mask EXTI7*/
    reg_pointer_32 = (uint32_t
*)EXTERNAL_INTERRUPT_CONTROLLER_MASK_REGISTER;
    *reg_pointer_32 = *reg_pointer_32 |
EXTERNAL_INTERRUPT_CONTROLLER_MASK_REGISTER_EXTI6;
    SET_BIT(*reg_pointer_32, 7);
    /*trigger on rising edge*/
    reg_pointer_32 = (uint32_t *)EXTERNAL_INTERRUPT_CONTROLLER_RTSR;
    SET_BIT(*reg_pointer_32,7);
    reg_pointer_32 = (uint32_t *)EXTERNAL_INTERRUPT_CONTROLLER_FTSR;
    SET_BIT(*reg_pointer_32,7);
    /* set the NVIC to respond to EXTI9_5*/
    reg_pointer_32 = (uint32_t *)NVIC_INTERRUPT_SET_ENABLE_REGISTER_0_31;
    *reg_pointer_32 = EXTI9_5_INTERRUPT_BIT;
```

```
}

void EXTI9_5_IRQHandler(void)
{
    uint32_t *reg_pointer_32;
    uint32_t *reg_pointer;
    uint32_t value;
    reg_pointer_32 = (uint32_t
*)EXTERNAL_INTERRUPT_CONTROLLER_PENDING_REGISTER;
    reg_pointer = (uint32_t*)PORTC_IDR_REGISTER;
    //check which interrupt fired:
    if ((*reg_pointer_32 & BIT(7)) > 0)
    {
        //clear the interrupt:
        SET_BIT(*reg_pointer_32, 7);
        value = *reg_pointer & BIT(7);
        printf("value is: %u\n", value);
        if(value){
            printf("Starting\n");
        }else
        {
            printf("stopping\n");
        }

    }



    }


void enableEXTI12OnPortB(void)
{
    uint32_t * reg_pointer_32;
    /*Init GPIO B 12 as input*/
    initGpioB12AsInput();
    /*As a test, Init GPIO B0 as output for debugging*/
    //initGpioB0AsOutput();
    /* Enable SYSCFG clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
    /*map EXTI12 to port B bit 12*/
```

```
    reg_pointer_32 = (uint32_t *)SYSCFG_EXTERNAL_INTERRUPT_REGISTER_4;
    //clear EXTI6
    *reg_pointer_32 = *reg_pointer_32 &
~SYSCFG_EXTERNAL_INTERRUPT_12_BITS;
    //set EXTI6 to Port C
    *reg_pointer_32 = *reg_pointer_32 |
SYSCFG_EXTERNAL_INTERRUPT_12_PORTB;
    /*un-mask EXTI6*/
    reg_pointer_32 = (uint32_t
*)EXTERNAL_INTERRUPT_CONTROLLER_MASK_REGISTER;
    SET_BIT(*reg_pointer_32,12);
    /*trigger on rising edge and falling edge*/
    reg_pointer_32 = (uint32_t *)EXTERNAL_INTERRUPT_CONTROLLER_RTSR;
    SET_BIT(*reg_pointer_32,12);
    reg_pointer_32 = (uint32_t *)EXTERNAL_INTERRUPT_CONTROLLER_FTSR;
    SET_BIT(*reg_pointer_32,12);
    /* set the NVIC to respond to EXTI15_10*/
    //40-32=8
    reg_pointer_32 = (uint32_t *)NVIC_INTERRUPT_SET_ENABLE_REGISTER_32_63;
    SET_BIT(*reg_pointer_32, 8);
}

void enableEXTI13OnPortB(void)
{
    uint32_t * reg_pointer_32;
    /*Init GPIO B 12 as input*/
    initGpioB13AsInput();
    /*As a test, Init GPIO B0 as output for debugging*/
    //initGpioB0AsOutput();
    /* Enable SYSCFG clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
    /*map EXTI12 to port B bit 12*/
    reg_pointer_32 = (uint32_t *)SYSCFG_EXTERNAL_INTERRUPT_REGISTER_4;
    //clear EXTI6
    *reg_pointer_32 = *reg_pointer_32 &
~SYSCFG_EXTERNAL_INTERRUPT_13_BITS;
    //set EXTI6 to Port C
    *reg_pointer_32 = *reg_pointer_32 |
SYSCFG_EXTERNAL_INTERRUPT_13_PORTB;
    /*un-mask EXTI6*/
```

```
    reg_pointer_32 = (uint32_t
*)EXTERNAL_INTERRUPT_CONTROLLER_MASK_REGISTER;
    SET_BIT(*reg_pointer_32,13);
    /*trigger on rising edge and falling edge*/
    reg_pointer_32 = (uint32_t *)EXTERNAL_INTERRUPT_CONTROLLER_RTSR;
    SET_BIT(*reg_pointer_32,13);
    //reg_pointer_32 = (uint32_t *)EXTERNAL_INTERRUPT_CONTROLLER_FTSR;
    //SET_BIT(*reg_pointer_32,13);
    /* set the NVIC to respond to EXTI15_10*/
    reg_pointer_32 = (uint32_t *)NVIC_INTERRUPT_SET_ENABLE_REGISTER_32_63;
    SET_BIT(*reg_pointer_32, 8);
}

void checkchange(void){
//written for testing purposes, no longer used
    if(a_curr!=a_old){
        if(b_curr!=a_curr){
            angle++;
        } else{
            angle--;
        }
    }
    a_old=a_curr;
    // debugprint(angle);

}

void updateT(void){
    //wrapper function to update global variable delT which is used for D
controller
    //delT is the time between the last angle change
    delT=readcount4();
}

void EXTI15_10_IRQHandler(void)
{
    //triggered on rising edge
    uint32_t *reg_pointer_32;
    uint32_t valueB13 = 0;
    uint32_t valueB12 = 0;
```

```
    uint32_t *reg_pointer;

    reg_pointer_32 = (uint32_t
*)EXTERNAL_INTERRUPT_CONTROLLER_PENDING_REGISTER;
    reg_pointer = (uint32_t*)PORTB_IDR_REGISTER;
    //check which interrupt fired:
    if ((*reg_pointer_32 & BIT(12)) > 0)
    {
        //clear the interrupt:
        SET_BIT(*reg_pointer_32, 12);
        //a_curr=checkGPIOB12();

        valueB13 = *reg_pointer & PIN_13_IDR_HIGH;
        valueB12 = *reg_pointer & PIN_12_IDR_HIGH;

        //logic to detect whether angle is incrementing or decrementing
        //based on the 2 phase shifted square waves generated by encoder
        if((valueB13 > 0) && (valueB12 ==0)){
            a_curr=1;
            //angle = angle + 1;
        }
        else if((valueB13 == 0) && (valueB12 > 0)){
            b_curr=1;
            //angle = angle - 1;
        }
        else if((valueB13 > 0) && (valueB12 >0)){
            a_curr=0;
        }
        else if((valueB13 == 0) && (valueB12 ==0)){
            b_curr=0;
        }

        if((a_curr==1) && (b_curr==1)){
            angle = angle + 1;
            updateT();
        }
        else if((a_curr == 0) && (b_curr==0)){
            angle = angle - 1;
            updateT();
        }
```

```
    }

}
```

hardware_stm_interruptcontroller.h

```
/* Define to prevent recursive inclusion
-------------------------------------*/
#ifndef __HARDWARE_STM_INTERRUPTCONTROLLER_H_
#define __HARDWARE_STM_INTERRUPTCONTROLLER_H_

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes
------------------------------------------------------------------*/
#include "main.h"

/*Function
definitions-------------------------------------------------------------*/
void enableNVIC_Timer3(void);
void enableNVIC_Timer2(void);
void enableNVIC_Timer5(void);
void initTimer3ToPWMMode1( void );
void setdirection(uint16_t d);
void pwm(int16_t val);
void initTimer3ToInterruptNOutputCompCh3( void );
void initTimer4Ch3( void );
void initTimer2ToPWMMode1( void );
void initTimer5Ch3( void );
void initTimer2Ch3( void );
void initTimer6Ch3( void );
void initTimer7Ch3( void );
uint16_t readcount(void);
uint16_t readcount4(void);
uint16_t readcount5(void);
uint16_t readcount2(void);
uint16_t readcount6(void);
uint16_t readcount7(void);
```

```
void updateT(void);
void TIM3_IRQHandler( void );
void TIM2_IRQHandler( void );
void TIM5_IRQHandler( void );
void enableEXTI6OnPortC(void);
void enableEXTI7OnPortC(void);
void EXTI9_5_IRQHandler(void);

void enableEXTI12OnPortB(void);
void enableEXTI13OnPortB(void);
void checkchange(void);
void EXTI15_10_IRQHandler(void);

#ifdef __cplusplus
}
#endif

#endif /*__TIMER3_H_ */
```

hardware_stm_timer3.c

```
#include "hardware_stm_gpio.h"
#include "stm32f4xx_rcc_mort.h"

/* MACRO
definitions-----------------------------------------------------------*/
#define TIM3_BASE_ADDRESS ((uint32_t)0x40000400)
// Timer 3 control register 1
#define TIM3_CR1_REGISTER_1 (TIM3_BASE_ADDRESS + 0x00)
//flags for CR1 register:
#define COUNTER_ENABLE_BIT (uint16_t)0x01
// Timer 3 status register
#define TIM3_STATUS_REGISTER (TIM3_BASE_ADDRESS + 0x10)
//flags for Status register:
#define TIM_UIF 0x01 //timer 3 overflow flag
#define TIM_CH1_CC1IF 0x2 //timer channel 1 capture/compare event
#define TIM_CH3_CC3IF 0x8 //timer channel 3 capture/compare event
//timer 3 interrupt enable register
#define TIM3_INTERRUPT_ENABLE_REGISTER (TIM3_BASE_ADDRESS + 0x0C)
```

```
//flags for interrupt enable register:
#define TIM_CH3_CC_INTERRUPT_ENABLE 0x8 //timer channel 3 capture/compare
interrupt
#define TIM_CH1_CC_INTERRUPT_ENABLE 0x2 //timer channel 1 capture/compare
interrupt
#define TIM_UPDATE_INTERRUPT_ENABLE 0x1 //timer overflow or event
interrupt
//Capture compare enable register
#define TIM3_CAPTURE_COMPARE_ENABLE_REGISTER (TIM3_BASE_ADDRESS + 0x20)
//flags for TIM3_CCER registers for output:
#define TIM3_CCER_CC3E (0x0100)
#define TIM3_CCER_CC1E (0b1)
//Capture compare mode registers
#define TIM3_CAPTURE_COMPARE_MODE_1_REGISTER (TIM3_BASE_ADDRESS + 0x18)
#define TIM3_CAPTURE_COMPARE_MODE_2_REGISTER (TIM3_BASE_ADDRESS + 0x1C)
//flags for Capture compare mode register
#define TIM_CCMR13_OC3M_0 (0b00010000)
#define TIM_CCMR13_OC3M_1 (0b00100000)
#define TIM_CCMR13_OC3M_2 (0b01000000)
#define TIM_CCMR13_OCPE (0b00001000) // enable preload register channels 1
and 3
#define TIM_CCMR13_OUTPUT (0x00)
#define TIM_CCMR1_CC1S_INPUT_TI1 (0b01)
// Compare, autoreload and Prescaler registers
#define TIM3_COMPARE_1_REGISTER (TIM3_BASE_ADDRESS + 0x34)
#define TIM3_COMPARE_2_REGISTER (TIM3_BASE_ADDRESS + 0x38)
#define TIM3_COMPARE_3_REGISTER (TIM3_BASE_ADDRESS + 0x3C)
#define TIM3_COMPARE_4_REGISTER (TIM3_BASE_ADDRESS + 0x40)
#define TIM3_PRESCALER_REGISTER (TIM3_BASE_ADDRESS + 0x28)
#define TIM3_AUTORELOAD_REGISTER (TIM3_BASE_ADDRESS + 0X2C)
// Timer counter value
#define TIM3_CNT (TIM3_BASE_ADDRESS + 0x24)

/* MACRO
definitions-------------------------------------------------------------*/
#define SYSTEM_CONTROL_BASE_ADDRESS (0xE000E000)
#define NVIC_BASE_ADDRESS (SYSTEM_CONTROL_BASE_ADDRESS + 0x100)
#define NVIC_INTERRUPT_SET_ENABLE_REGISTER_0_31 (NVIC_BASE_ADDRESS)
#define NVIC_INTERRUPT_SET_ENABLE_REGISTER_32_63 (NVIC_BASE_ADDRESS+0x4)
#define NVIC_INTERRUPT_SET_ENABLE_REGISTER_64_95 (NVIC_BASE_ADDRESS+0x8)
```

```
#define TIM3_INTERRUPT_BIT (0x20000000)

/* definitions */
#define CCMR_OC3M_PWM_MODE1 (0b01100000)
#define CCMR_CC3s_OUTPUT (0b00000000)


void initTimer3Channel3( void )
{
    uint16_t * reg_pointer_16;
    uint16_t prescalervalue2, autoreloadvalue;
    /* Timer 3 APB clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
    /* Compute Prescale and Autorreload */
    // if we want LED freq to be 1Hz
    // 1Hz = 90MHz / ((prescalervalue+1)*(autoreloadvalue))
    // Frequency of system clock is 90 MHz
    prescalervalue2 = 8999;
    autoreloadvalue = 10000;
    /* Clear any pending flags in the status register */
    reg_pointer_16 = (uint16_t *)TIM3_STATUS_REGISTER;
    *reg_pointer_16 = 0;
    /* Set Prescale and Autorreload */
    reg_pointer_16 = (uint16_t *)TIM3_PRESCALER_REGISTER;
    *reg_pointer_16 = prescalervalue2;
    reg_pointer_16 = (uint16_t *)TIM3_AUTORELOAD_REGISTER;
    *reg_pointer_16 = autoreloadvalue;
    /* Capture Compare Register 2 OC3M to toggle mode */
    reg_pointer_16 = (uint16_t *)TIM3_CAPTURE_COMPARE_MODE_2_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM_CCMR13_OC3M_1 |
TIM_CCMR13_OC3M_0 | TIM_CCMR13_OUTPUT;
    /* Set Compare Value */
    reg_pointer_16 = (uint16_t *)TIM3_COMPARE_3_REGISTER;
    *reg_pointer_16 = autoreloadvalue/2; // duty cycle
    /* Enable Preload Register (Don't HAVE to, but good practice) */
    reg_pointer_16 = (uint16_t *)TIM3_CAPTURE_COMPARE_MODE_2_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM_CCMR13_OCPE;
    /*enable the TIM3 channel 3 counter and keep the default configuration
for channel polarity*/
    reg_pointer_16 = (uint16_t *)TIM3_CAPTURE_COMPARE_ENABLE_REGISTER;
```

```
    *reg_pointer_16 = *reg_pointer_16 | TIM3_CCER_CC3E;
    /*enable timer 3*/
    reg_pointer_16 = (uint16_t *)TIM3_CR1_REGISTER_1;
    *reg_pointer_16 = *reg_pointer_16 | COUNTER_ENABLE_BIT;
}

void initTimer3Channel1ToInputCapture( void )
{
    uint16_t * reg_pointer_16;
    uint16_t prescalervalue2, autoreloadvalue;
    /* Timer 3 APB clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
    /* Compute Prescale and Autorreload */
    // if we want LED freq to be 1Hz
    // 1Hz = 90MHz / ((prescalervalue+1)*(autoreloadvalue))
    // Frequency of system clock is 90 MHz
    prescalervalue2 = 8999;
    autoreloadvalue = 40000;
    /* Clear any pending flags in the status register */
    reg_pointer_16 = (uint16_t *)TIM3_STATUS_REGISTER;
    *reg_pointer_16 = 0;
    /* Set Prescale and Autorreload */
    reg_pointer_16 = (uint16_t *)TIM3_PRESCALER_REGISTER;
    *reg_pointer_16 = prescalervalue2;
    reg_pointer_16 = (uint16_t *)TIM3_AUTORELOAD_REGISTER;
    *reg_pointer_16 = autoreloadvalue;
    /* Capture Compare Register 1 CC1S to input mode, IC1 mapped to TI1 */
    reg_pointer_16 = (uint16_t *)TIM3_CAPTURE_COMPARE_MODE_1_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM_CCMR1_CC1S_INPUT_TI1;
    /* Enable Preload Register (Don't HAVE to, but good practice) */
    reg_pointer_16 = (uint16_t *)TIM3_CAPTURE_COMPARE_MODE_2_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM_CCMR13_OCPE;
    /*enable the TIM3 channel 1 counter and keep the default configuration
for channel polarity*/
    reg_pointer_16 = (uint16_t *)TIM3_CAPTURE_COMPARE_ENABLE_REGISTER;
    *reg_pointer_16 = *reg_pointer_16 | TIM3_CCER_CC1E;
    /*enable timer 3*/
    reg_pointer_16 = (uint16_t *)TIM3_CR1_REGISTER_1;
    *reg_pointer_16 = *reg_pointer_16 | COUNTER_ENABLE_BIT;
}
```

```
uint16_t ReadInputCaputreFlagNTakeAction( void )
{
    uint16_t TIM3_Value = 0;
    uint16_t *reg_pointer_16_sr;
    uint16_t *reg_pointer_32;
    // read TIM3_SR value & clear by reading
    reg_pointer_16_sr = (uint16_t *)TIM3_STATUS_REGISTER;
    //TIM3SR_Value = *reg_pointer_16;
    // check if the flag is triggered
    if ((*reg_pointer_16_sr & TIM_CH1_CC1IF) == TIM_CH1_CC1IF)
    {
        // get timer value form cap/com register ch1
        reg_pointer_32 = (uint16_t *)TIM3_COMPARE_1_REGISTER;
        TIM3_Value = *reg_pointer_32;
    }
    return TIM3_Value;
}

void ReadInputCaputreFlagNReadInput( void )
{
    uint16_t TIM3_Value = 0;
    uint16_t *reg_pointer_16_sr;
    uint16_t *reg_pointer_32;
    // read TIM3_SR value & clear by reading
    reg_pointer_16_sr = (uint16_t *)TIM3_STATUS_REGISTER;
    //TIM3SR_Value = *reg_pointer_16;
    // check if the flag is triggered
    if ((*reg_pointer_16_sr & TIM_CH1_CC1IF) == TIM_CH1_CC1IF)
    {
        // checkGPIOB13
        // checkGPIOB12
        // checkGPIOC7
    }
}
```

hardware_stm_timer3.h

```
/* Define to prevent recursive inclusion
-------------------------------------*/
#ifndef __HARDWARE_STM_TIMER3_H_
```

```
#define __HARDWARE_STM_TIMER3_H_

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes
------------------------------------------------------------------*/
#include "main.h"

/*Function
definitions-----------------------------------------------------------*/
void initTimer3Channel3( void );
//uint32_t ValueTimer3Channel3( void );
void initTimer3Channel1ToInputCapture( void );
uint16_t ReadInputCaputreFlagNTakeAction( void );
void ReadInputCaputreFlagNReadInput( void );

#ifdef __cplusplus
}
#endif

#endif /*__TIMER3_H_ */
```

main.cpp

```
#include "main.h"
#include "mbed.h"
#include "debug_mort.h"
#include "nucleo_led.h"
#include "hardware_stm_gpio.h"
#include "hardware_stm_timer3.h"
#include "hardware_stm_adc.h"
#include "hardware_stm_dma.h"
#include "hardware_stm_interruptcontroller.h"
#include "hand_detector.h"
#include <cstdint>

// Global variables
```

```
volatile uint16_t a_curr=0;
volatile uint16_t a_old=0;
volatile uint16_t b_curr=0;
volatile uint32_t ovf=0;

volatile int16_t angle=0; //global variable to store angle
volatile uint16_t oldT=0;
volatile uint32_t delT=0; //global variable to store time difference
between encode ticks

volatile int16_t oldangle=0;


int main (void)
{
    uint16_t previousADCValue = 0;
    uint16_t currentADCValue = 0;
    uint16_t pin15 = 15;
    uint16_t pin6 = 6;
    uint16_t a=0;

    // initialize GPIOC6 as input for button
    initGpioC6AsInput();
    // initialize GPIOB15 as Output high
    initGpioBAsOut(pin15, 1);
    // initialize GPIOB6 as Output low
    initGpioBAsOut(pin6, 0);
    // initialize GPIOB5 as AF2 Output
    initGpioB5MappedToTim3Chn3AsOutput();
    // initialize GPIOB12 as EXTI input
    enableEXTI12OnPortB();
    // initialize GPIOB13 as input
    initGpioB13AsInput();
    // initialize timer 4
    initTimer4Ch3();
    // initialize PWM mode 1
    initTimer3ToPWMMode1();
    //
    initializeADCforFSR();
    //
```

```
    initializeGPIOF01289AsInput();
    //initTimer5Ch3();
    initTimer2Ch3();
    // enable external interrupt
    enableEXTI7OnPortC();
    uint16_t kk=0;
    while(1){
        //check if GPIOC6 value
        checkButton();
        PlayNStop();
        //kk=kk+1;
        /*if (kk==5){
            printfinalscore();
            kk=0;
            printf("Starting new game \n");
            printf("\n");
            resetscore();

        }*/

    }
}
```

main.h

```
#include "main.h"
#include "mbed.h"
#include "debug_mort.h"
#include "nucleo_led.h"
#include "hardware_stm_gpio.h"
#include "hardware_stm_timer3.h"
#include "hardware_stm_adc.h"
#include "hardware_stm_dma.h"
#include "hardware_stm_interruptcontroller.h"
#include "hand_detector.h"
#include <cstdint>

// Global variables
volatile uint16_t a_curr=0;
volatile uint16_t a_old=0;
volatile uint16_t b_curr=0;
```

```
volatile uint32_t ovf=0;

volatile int16_t angle=0; //global variable to store angle
volatile uint16_t oldT=0;
volatile uint32_t delT=0; //global variable to store time difference
between encode ticks

volatile int16_t oldangle=0;


int main (void)
{
    uint16_t previousADCValue = 0;
    uint16_t currentADCValue = 0;
    uint16_t pin15 = 15;
    uint16_t pin6 = 6;
    uint16_t a=0;

    // initialize GPIOC6 as input for button
    initGpioC6AsInput();
    // initialize GPIOB15 as Output high
    initGpioBAsOut(pin15, 1);
    // initialize GPIOB6 as Output low
    initGpioBAsOut(pin6, 0);
    // initialize GPIOB5 as AF2 Output
    initGpioB5MappedToTim3Chn3AsOutput();
    // initialize GPIOB12 as EXTI input
    enableEXTI12OnPortB();
    // initialize GPIOB13 as input
    initGpioB13AsInput();
    // initialize timer 4
    initTimer4Ch3();
    // initialize PWM mode 1
    initTimer3ToPWMMode1();
    //
    initializeADCforFSR();
    //
    initializeGPIOF01289AsInput();
    //initTimer5Ch3();
    initTimer2Ch3();
```

```
    // enable external interrupt
    enableEXTI7OnPortC();
    uint16_t kk=0;
    while(1){
        //check if GPIOC6 value
        checkButton();
        PlayNStop();
        //kk=kk+1;
        /*if (kk==5){
            printfinalscore();
            kk=0;
            printf("Starting new game \n");
            printf("\n");
            resetscore();

        }*/

    }
}
```

nucleo_led.c

```
/**

******************************************************************************
****
  * @file    nucleo_led.c
  * @author  mortamar@andrew.cmu.edu
  * @version 1.0
  * @date    Septembr-2021
  * @brief   Controls the LED's on the nucleo board

******************************************************************************
****
  */

#include "hardware_stm_gpio.h"
#include "hardware_stm_timer3.h"
#include "nucleo_led.h"

/***********************************
```

```
* Initializes LED1 on the nucleo Board which is connected to Port B Pin 0
**********************************/
void init_LED1( void )
{
    // Call something from hardware_stm_gpio
    initGpioB0AsOutput();
}
/**********************************
* Toggles LED1
**********************************/
void toggle_LED1( void )
{
    // Call something else from hardware_stm_gpio
    toggleGPIOB0();
}

/* Function that turns the LED on or off */
void LED_ONOROFF_FUNCTION( void )
{
    uint32_t valueC6 = 0;
    // check GPIOC PIN 6
    valueC6 = checkGPIOC6();
    // when pin 6 high
    if (valueC6 == 64)
    {
        // turn on LED
        setGPIOB0();

    }
    // when pin 6 low
    else
    {
        // turn off LED
        clearGPIOB0();
    }
}

//void LED_ONOROFF_TIMER3_FUNCTION( void )
//{
//    //initialize pin6 as input
```

```
//    initGpioC6AsInput();
//    //initialize timer3 channel3
//    initTimer3Channel3();
//    uint32_t valueT3C3;
//    // check GPIOC PIN 6
//    valueT3C3 = ValueTimer3Channel3();
//    // when pin 6 low
//    if (valueT3C3 == 0b0)
//    {
//        // turn off LED
//        clearGPIOB0();
//    }
//    // when pin 6 high
//    else if (valueT3C3 == 0b1)
//    {
//        // turn on LED
//        setGPIOB0();
//    }
//}
```

nucleo_led.h

```
/* Define to prevent recursive inclusion
-------------------------------------*/
#ifndef __LED1_H_
#define __LED1_H_

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes
------------------------------------------------------------------*/
#include "main.h"

/* Macros for
Everyone----------------------------------------------------------*/
```

```
/*Function
definitions----------------------------------------------------------------*/

void init_LED1(void);
void toggle_LED1(void);
void LED_ONOROFF_FUNCTION(void);
//void LED_ONOROFF_TIMER3_FUNCTION(void);

#ifdef __cplusplus
}
#endif

#endif /*__LED1_H */
```

Lesson Learned

- Testing of electronic components and checking their performances (e.g. button didn't work)
- Controlling motor with PWM signal and PID controller with generalized code
- Implementing logic (such as slap) into real life
- It's important to leave buffer time to do final testing and find bugs in the code
- Keep it simple (e.g. not implement unnecessary timers)
- Don't deviate from core concepts couple days before deadline (e.g. using 2 microcontrollers)
- Make sure everyone on the team is competent with git

Citation

- *Arcade machine game cartoon vector icon illustration. holiday technology icon isolated premium flat Free Vector*. (2023, February 28). Freepik. https://www.freepik.com/vectors/arcade
- *Amazon.com: 10PCS IR Infrared Obstacle Avoidance Sensor Module for Smart Car Robot : Industrial & Scientific*. (n.d.). https://www.amazon.com/dp/B08ZYDMZMD?psc=1&ref=ppx_yo2ov_dt_b_produ

ct_details

- *Amazon.com: Rebower Push Button Switch NO Latching Round Button, [for Light Relay] - 10mm / red / 2 pcs : Industrial & Scientific*. (n.d.). https://www.amazon.com/dp/B0BXJ6GQ9S?psc=1&ref=ppx_yo2ov_dt_b_product_details
- *Force Sensitive Resistor 0.5"*. (n.d.). SEN-09375 - SparkFun Electronics. https://www.sparkfun.com/products/9375
- *Pololu - MP 12V Motor with 48 CPR Encoder for 25D mm Metal Gearmotors (No Gearbox)*. (n.d.). https://www.pololu.com/product/4860
- *STMicroElectronics L293B*. (n.d.). Mouser Electronics. https://www.mouser.com/ProductDetail/STMicroelectronics/L293B?qs=gr8Zi5OG3Mhln2B1GrZirA%3D%3D
- *Pololu - Logic Level Shifter, 4-Channel, Bidirectional*. (n.d.). https://www.pololu.com/product/2595
- *Diodes - national instruments*. (n.d.). https://education.ni.com/teach/resources/926/diodes
- *Resistors & Capacitors*. (n.d.). indiamart.com. https://www.indiamart.com/proddetail/resistors-capacitors-4451967462.html
- Chernikhovska, V. (2020, December 6). Electrical Wiring Color Code Standards.

Nassau National Cable.

https://nassaunationalcable.com/blogs/blog/electrical-wiring-color-code-standards