# 20WSC355 Interfacing Using a Microcontroller Submission Portfolio

This portfolio should contain outputs that are generated as a result of completing laboratory sessions, plus and extra task to complete in your own time.

**Note:** When asked for code, only insert an excerpt of you code for the specific functions requested. Ensure code is correctly annotated with you own description of functions.

## Timers and GPIO Interfacing

### Initialisation Code

**From Laboratory 1: Task 2.1.** Insert below the code used to initialise the GPIO ports

#include "stm32f3xx.h" // Device header

void delay(int a); // prototype for delay function

int main()

{

RCC->AHBENR |= RCC\_AHBENR\_GPIOEEN;// Enable clock on GPIO port E

//PE.8 and PE.12 set to output mode "01"

GPIOE->MODER = (GPIOE->MODER & ~0x3030000) | (0x1010000);

GPIOE->OTYPER |= 0x100; // PE.8 set to open drain

GPIOE->OTYPER &= ~0x1000; // PE.12 set to push/pull

//-------------Set PE.8 to Pull-Up and set PE.12 to No pull-up,pull-down

GPIOE->PUPDR = (GPIOE->PUPDR & ~0x3030000) | 0x10000;

//----------------------------------------------------------------------

while (1)

{

GPIOE->BSRRL = 0x1100;// turn on LED on PE.8,PE.12

delay(555555 \* 2);// wait for 1 second

GPIOE->BSRRH = 0x1100; // turn off LED on PE.8,PE.12

delay(555555 \* 2);// wait for 1 second

}

}

// Delay function to occupy processor

void delay(int a)

{

volatile int i;

for (i = 0; i < a; i++)

{}

}

### Circuit Diagram

**From Laboratory 1: Task 2.1.** Include the circuit diagram of the connections on Port E pins 8 and 12.

*Diagram

Description automatically generated*

### Write Values

**From Laboratory 1: Task 2.3.** Include the code used to both increment the count and to display the output on the LEDs.

#include "stm32f3xx.h"

void delay(int a); // prototype for delay function

int main(void)

{

RCC->AHBENR |= RCC\_AHBENR\_GPIOEEN; // Enable clock on GPIO port E

//-----------------ALL LED PINS SET TO PUSH/PULL WITH NO No pull-up, pull-down

GPIOE->MODER = (GPIOE->MODER & ~(0xFFFF0000)) | 0x55550000;// output mode "01"

//for pins(8-15)

GPIOE->OTYPER &= ~(0xFF00); // push/pull "00" for pins(8-15)

GPIOE->PUPDR &= ~(0xFFFF0000); // no pullup, pull-down for pins(8-15)

//--------------------------------------------------------------------------------

int count = 0;// = 0b00000000

while (1)

{

while (count < (255 + 1)) // run max value + 1 times, (2^8)-1 +1= 255+1

{

GPIOE->BSRRL = (count << 8);// turn on LED by shifting bits to //the left

delay(555555 \* 2);// wait for 1 second

GPIOE->BSRRH = (count << 8); // turn off LED

++count; // increment

}

GPIOE->BSRRH = 0xFF00; // turn off all LEDs

count = 0;// reset count

delay(555555 \* 2);// wait for 1 second to 'see' all LED OFF

}

}

// Delay function to occupy processor

void delay(int a)

{

volatile int i;

for (i = 0; i < a; i++)

{

}

}

### Timing Calculations

**From Laboratory 1: Task 3.1.** Show the calculation used to decide a value of PSC and ARR for an ‘Update’ interrupt to occur once a second.

Answer: The maximum value of PSC is chosen then the value of ARR is calculated from there with an interrupt frequency of 1Hz.

Text

Description automatically generated with medium confidence

### Initialisation Code

**From Laboratory 1: Task 3.2-3.3.** Insert below the lines of code created to initialise an onboard timer to generate an interrupt.

#include "stm32f3xx.h" // Device header

#include <stdbool.h>

static volatile bool flag = false;

void TIM3\_IRQHandler()

{

if ((TIM3->SR & TIM\_SR\_UIF) != 0) // Check interrupt source is from the //'Update' interrupt flag

{

if (flag)// if LED is on

{

GPIOE->BSRRH = 0x200; // turn off

flag = false;// change flag

}

else if (!flag)// if LED is off

{

GPIOE->BSRRL = 0x200;// turn on LED1

flag = true; // change flag to go back to the first if statement

}

}

TIM3->SR &= ~TIM\_SR\_UIF; // Reset 'update' interrupt flag in the SR register

}

int main(void)

{

// Enable clock on GPIO port E

RCC->AHBENR |= RCC\_AHBENR\_GPIOEEN;

//-----------------ALL LED PINS SET TO PUSH/PULL WITH NO No pull-up, pull-down

GPIOE->MODER = (GPIOE->MODER & ~(0xFFFF0000)) | 0x55550000;// output mode "01" //for pins(8-15)

GPIOE->OTYPER &= ~(0xFF00); // push/pull "00" for pins(8-15)

GPIOE->PUPDR &= ~(0xFFFF0000); // no pullup, pull-down for pins(8-15)

//--------------------------------------------------------------------------------

RCC->APB1ENR |= RCC\_APB1ENR\_TIM3EN; //Define the clock pulse toTIM3

TIM3->PSC = 65535;

TIM3->ARR = (int)121.0703;

TIM3->CR1 |= TIM\_CR1\_CEN;//Set Timer Control Register to start timer

TIM3->DIER |= TIM\_DIER\_UIE; // Set DIER register to watch out for an 'Update' //Interrupt Enable (UIE) – or 0x00000001

NVIC\_EnableIRQ(TIM3\_IRQn); // Enable Timer 3 interrupt request in NVIC

}

### Interrupt Service Routine

**From Laboratory 2:** Insert below the ISR created to write the new values of a count on to the LEDs

#include "stm32f3xx.h"

static int count = 0;

void counter();

void TIM3\_IRQHandler()

{

if ((TIM3->SR & TIM\_SR\_UIF) != 0) // Check interrupt source is from the //'Update' interrupt flag

{

counter(); // call the binary counter function

}

TIM3->SR &= ~TIM\_SR\_UIF; // Reset 'update' interrupt flag in the SR register

}

int main(void)

{

// Enable clock on GPIO port E

RCC->AHBENR |= RCC\_AHBENR\_GPIOEEN;

//-----------------ALL LED PINS SET TO PUSH/PULL WITH NO No pull-up, pull-down

GPIOE->MODER = (GPIOE->MODER & ~(0xFFFF0000)) | 0x55550000;// output mode "01" //for pins(8-15)

GPIOE->OTYPER &= ~(0xFF00); // push/pull "00" for pins(8-15)

GPIOE->PUPDR &= ~(0xFFFF0000); // no pullup, pull-down for pins(8-15)

//--------------------------------------------------------------------------------

RCC->APB1ENR |= RCC\_APB1ENR\_TIM3EN; //Define the clock pulse toTIM3

TIM3->PSC = 65535;

TIM3->ARR = (int)121.0703;

TIM3->CR1 |= TIM\_CR1\_CEN;//Set Timer Control Register to start timer

TIM3->DIER |= TIM\_DIER\_UIE; // Set DIER register to watch out for an 'Update' //Interrupt Enable (UIE) – or 0x00000001

NVIC\_EnableIRQ(TIM3\_IRQn); // Enable Timer 3 interrupt request in NVIC

}

void counter()

{

if (count > 255) { count = 0; } // reset counter if max value is reached

GPIOE->BSRRH = 0xFF00; // turn off LEDs

GPIOE->BSRRL = count << 8; // shift bits to turn on LEDs

++count;// increment counter

}

### Observations

**From Laboratory 1:** briefly comment on the two methods used to manage the timing aspect of the LED control

Answer: The delay function is a CPU blocking delay while the other option uses a timer based interrupt.

The delay function blocks the CPU preventing it from executing any other code, therefore wasting the CPU-time. It also requires some trial-error to work out the number required for exact loop time. However,it is easier to comprehend and implement.

The time based interrupt requires some calculations to work out the prescaler and auto-reload registor values for a specific interrupt frequency. This has advantages such as allowing the CPU to perform other tasks and can be used for generating signals and sampling sensors. With the only disadvantage being the complexity require to initialise the interrupt correctly.

## DAC and ADC

### Initialisation Code

**From Laboratory 2: Task 1.1.** Insert below both the code used to configure the DAC, andthe Interrupt Service Routine used output an analogue voltage

#include "stm32f3xx.h" // Device header

static int count = 0;//0b01

void counter();

void TIM3\_IRQHandler()

{

if ((TIM3->SR & TIM\_SR\_UIF) != 0) // Check interrupt source is from the //'Update' interrupt flag

{

counter();

}

TIM3->SR &= ~TIM\_SR\_UIF; // Reset 'update' interrupt flag in the SR register

}

int main(void)

{

// Enable clock on GPIO port E

RCC->AHBENR |= RCC\_AHBENR\_GPIOEEN;

// Configure ports for DAC

GPIOA->MODER |= 0x300; // PA.4 set to analogue mode

//-----------------ALL LED PINS SET TO PUSH/PULL WITH NO No pull-up, pull-down

GPIOE->MODER = (GPIOE->MODER & ~(0xFFFF0000)) | 0x55550000;// output mode "01" //for pins(8-15)

GPIOE->OTYPER &= ~(0xFF00); // push/pull "00" for pins(8-15)

GPIOE->PUPDR &= ~(0xFFFF0000); // no pullup, pull-down for pins(8-15)

//--------------------------------------------------------------------------------

RCC->APB1ENR |= RCC\_APB1ENR\_TIM3EN; //Define the clock pulse toTIM3

TIM3->PSC = 65535;

TIM3->ARR = (int)11.2070;

TIM3->CR1 |= TIM\_CR1\_CEN;//Set Timer Control Register to start timer

TIM3->DIER |= TIM\_DIER\_UIE; // Set DIER register to watch out for an 'Update' Interrupt Enable (UIE) – or 0x00000001

NVIC\_EnableIRQ(TIM3\_IRQn); // Enable Timer 3 interrupt request in NVIC

//--------------------DAC configuration

RCC->APB1ENR |= RCC\_APB1ENR\_DAC1EN;

DAC1->CR |= DAC\_CR\_BOFF1;

DAC1->CR |= DAC\_CR\_EN1;

//-------------------------------------

}

void counter()

{

if (count > 255) { count = 0; }

DAC1->DHR12R1 = count;// write counter into DAC

GPIOE->BSRRH = 0xFF00; // turn OFF Leds

GPIOE->BSRRL = (DAC1->DHR12R1) << 8; // turn ON Leds by shifting bits

++count;

}

### Maximum Voltage

**From Laboratory 2: Task 1.1.** Show your calculations for the maximum anticipated voltage output from the DAC.

*Text

Description automatically generated with medium confidence*

Using the supply voltage of 3v results in the maximum DAC signal voltage of 186.8mV however other online sources suggest using 3.3v which results in the max voltage to be 205mV.

### ADC Initialisation Code

**From Laboratory 2: Task 2.1.** Insert below the sub-function created to initialise the onboard ADC peripheral.

#include "stm32f3xx.h"

static int count = 0;

void wait(int a); // delay function

void ADC\_init(); // ADC function prototype

void DAC\_init();// DAC function prototype

void LEDS\_ALL\_init(); // initialise all GPIOE LEDs

void TIM3\_init();// Initialise the timer at 100Hz

void TIM3\_IRQHandler()

{

if ((TIM3->SR & TIM\_SR\_UIF) != 0) // Check interrupt source is from the //'Update' interrupt flag

{

++count;

if (count > 255) { count = 0; }// reset count to 1 if max value reached

}

TIM3->SR &= ~TIM\_SR\_UIF; // Reset 'update' interrupt flag in the SR register

}

int main(void)

{

LEDS\_ALL\_init();// initialise all LEDs

TIM3\_init();// initialise the timer interrupt

DAC\_init(); // initialise the DAC

ADC\_init();// initialise the ADC

while (1)

{

DAC1->DHR12R1 = count; // write the value into the DAC

ADC1->CR |= 0x4; // enable ADC

while (!(ADC1->ISR & 0x4)) {}// wait for EOC flag to go high

GPIOE->BSRRH = 0xFF00; // turn OFF Leds

GPIOE->BSRRL = (ADC1->DR) << 8; // turn ON Leds by shifting bits

}

}

void wait(int c)

{

int ticks = 0;

while (ticks < c)// exit when ticks reach the count

{

++ticks;

}

}

void LEDS\_ALL\_init() // initialise all GPIOE LEDs

{

// Enable clock on GPIO port E

RCC->AHBENR |= RCC\_AHBENR\_GPIOEEN;

//-----------------ALL LED PINS SET TO PUSH/PULL WITH NO No pull-up, pull-down

GPIOE->MODER = (GPIOE->MODER & ~(0xFFFF0000)) | 0x55550000;// output mode "01" //for pins(8-15)

GPIOE->OTYPER &= ~(0xFF00); // push/pull "00" for pins(8-15)

GPIOE->PUPDR &= ~(0xFFFF0000); // no pullup, pull-down for pins(8-15)

//--------------------------------------------------------------------------------

}

void TIM3\_init()

{

RCC->APB1ENR |= RCC\_APB1ENR\_TIM3EN; //Define the clock pulse toTIM3

TIM3->PSC = 65535;

TIM3->ARR = (int)11.2070;

TIM3->CR1 |= TIM\_CR1\_CEN;//Set Timer Control Register to start timer

TIM3->DIER |= TIM\_DIER\_UIE; // Set DIER register to watch out for an 'Update' Interrupt Enable (UIE) – or 0x00000001

NVIC\_EnableIRQ(TIM3\_IRQn); // Enable Timer 3 interrupt request in NVIC

}

void DAC\_init()

{

//DAC configuration

RCC->APB1ENR |= RCC\_APB1ENR\_DAC1EN;

GPIOA->MODER |= 0x300; // PA.4 set to analogue mode

DAC1->CR |= DAC\_CR\_BOFF1;

DAC1->CR |= DAC\_CR\_EN1;

//-------------------------------------

}

void ADC\_init()

{

ADC1->CR &= ~(0x30000000); // 0b00 for the ADVREGEN = reset

ADC1->CR |= (0x10000000);// 0b01 for the ADVREGEN = enable

RCC->AHBENR |= RCC\_AHBENR\_GPIOFEN;// enable clock on GPIOF

GPIOF->MODER |= 0x30; // analogue mode on PF.2, "11"

wait(100);// 10us delay function

ADC1->CR &= ~(0x40000000);// 0b0 0 single-ended calibration

ADC1->CR |= 0x80000000;// begin calibration

while (ADC1->CR & 0x80000000) {} // wait for ADCAL to return 0

//------------------Enabling clock peripherials

RCC->CFGR2 = (RCC->CFGR2 & ~0x1F0) | RCC\_CFGR2\_ADCPRE12\_DIV2;// clear registor //first then 10001: PLL clock divided by 2

RCC->AHBENR |= RCC\_AHBENR\_ADC12EN;

ADC1\_2\_COMMON->CCR |= 0x00010000;

//---------------------------------------------

ADC1->CR &= ~(0xC);// set ADSTART and JADSTART to 0, "00 00"

ADC1->CFGR = (ADC1->CFGR & ~0x18) | (0x10);// 8 bit resolution

ADC1->CFGR &= ~(0x20);// align right

ADC1->CFGR &= ~(0x2000);// no continous

ADC1->SQR1 = (ADC1->SQR1 & ~0x3C0) | 0x280;// channel 10 in SQR1, "01010"

ADC1->SQR1 &= ~(0xF);// only one conversion, L is "0000"

ADC1->SMPR1 |= 0x38; // 601.5 ADC clock cycles

ADC1->CR |= 0x1; // ADEN set to high to enable ADC

while (!(ADC1->ISR & 0x01)) {}// wait for ARDY flag to go high

}

### Observations

**From Laboratory 2: Task 2.3.** Comment on the maximum bitwise and voltage value obtained in the ADC. Comment on the clarity of the bitwise counter output.

Answer:

The DAC is configured to be 12-bit right aligned while the ADC is 8-bit. Since we are using only 8-bits of the DAC, the ADC will be able to read the 8 most significant bits(MSB) resulting in an attenuation of 1/16. Therefore the maximum value the ADC is able to read is 0b1111 or 16, turning on the first 4 LEDs but not all of them.

Moreover, it ‘looks’ like the frequency of the ADC has slowed down but this is again due to the fact it reads the 8 MSB.

The attenuation of the signal would be 24.1 dB (-24.1 dB gain).

Graphical user interface, text, application

Description automatically generated

### Signal Chain

**From Laboratory 2: Task 3.1/3.2.** Include the diagram of the signal chain and the OpAmp gain

Diagram, text

Description automatically generated

### Observations

**From Laboratory 2: Task 2.3.** Comment on the overall quality of the output count value.

The Op-Amp is configured to provide a gain of 16, which effectiyely means a left bitwise shift of 4. This is enough to nullify the effect of previous attenuation effect created by using a 8-bit. Now the value read by the ADC becomes 0b1111 1111 which turns on all LEDs providing the expected results. An alternate way would be to simply use a 12-bit ADC which would not require an Op-Amp at all.

## Pulses

### Initialisation

**From Laboratory 3: Task 1.1.** Include your code used to conifigure Timer 1 to output a PWM on a chosen pin.

#include "stm32f3xx.h"

int main()

{

RCC->APB2ENR |= RCC\_APB2ENR\_TIM1EN;// enable clock connection for TIMER1

RCC->AHBENR |= RCC\_AHBENR\_GPIOEEN;// enable clock on GPIOE

GPIOE->MODER = (GPIOE->MODER & (~0x3C0C0000)) | 0x28080000; // PE.9,PE13,PE14 configured //to alternate mode

GPIOE->OTYPER &= ~(0x6200); // Push/pull for PE9,PE13,PE14

GPIOE->PUPDR &= ~(0x3C0C0000); // No pull-up,pull down for PE9,PE13,PE14

GPIOE->AFR[1] = (GPIOE->AFR[1] & ~(0xFF000F0)) | 0x2200020; // PE.9,PE.13 and PE.14

TIM1->PSC = 19;

TIM1->ARR = 3999; // around 100Hz or 0.01s

TIM1->CCMR1 |= 0x00000060; // channel 1 for PE.9

TIM1->CCMR2 = (TIM1->CCMR2 & ~0xF0F0) | 0x6060;// channel 3 and 4(PE.13 and PE.14 )

TIM1->CCR1 = 400; //determine the duty cycle, 10%

TIM1->CCR3 = 2000; //determine the duty cycle, 50%

TIM1->CCR4 = 3600; //determine the duty cycle, 90%

//Enable the Channel chosen to be output to the GPIO pin

TIM1->BDTR |= TIM\_BDTR\_MOE;// 0x00008000

TIM1->CCER |= 0x1101; // channel 1,3 and channel 4,{capture/compare output "enable"}

//------------------------------------------------------

TIM1->CR1 |= TIM\_CR1\_CEN; // enable timer

}

### Calculations

**From Laboratory 3: Task 1.2.** Include your calculations to show the PWM pulse frequency and the possible resolution of output pulses. Why did you choose these values?

A higher value of ARR results in higher resolution. High resolutions allows us to precisely control the duty cycle which set by the CCR registor closely related to the ARR as seen from the equation. The resolution is calculated by using log base 2 of (ARR+1) which in our case is 11.96 bits with the PWM frequency of 100Hz. A high value of ARR is chosen as it provides a high resolution and a sensible non-decimal value of PSC.

Text

Description automatically generated

### External Interrupt Initialisation

**From Laboratory 3: Task 2.1.** Include your code used to configure the USR button to generate an Interrupt Service Routine.

#include "stm32f3xx.h"

static int index = 1;// starting index, could be anything 0-4

int arr[5] = { 0,1000,2000,3000,4000 }; // calculated values for CCR1

void PWM\_init(int); // initialise the PWM

void interrupt\_init(void);// initialise the external interrupt

void setDutyCycle(int); // function prototype for cycling thro duty cycles

void EXTI0\_IRQHandler() // external interrupt handler function

{

if (EXTI->PR & EXTI\_PR\_PR0) // check source

{

EXTI->PR |= EXTI\_PR\_PR0; // clear flag\*

if (index > 4) { index = 0; }// reset index if max value reached

setDutyCycle(arr[index]);

++index; // increment dutyCycle by 25%

}

};

int main()

{

interrupt\_init(); // initialise the external interrupt for USR button

PWM\_init(0); // initialise the PWM

}

void interrupt\_init()// code to initialise USR button to generate ISR

{

//Enable the system configuration controller to be connected to a system cloc

RCC->APB2ENR |= RCC\_APB2ENR\_SYSCFGEN;

//Remove the mask to enable an interrupt to be generated using the EXTI\_IMR register

EXTI->IMR |= EXTI\_IMR\_MR0;

//Set interrupt trigger to be rising edge

EXTI->RTSR |= EXTI\_RTSR\_TR0;

//The USR push button (blue button on the STM32F3discovery board) is connected to pin //PA.0.

SYSCFG->EXTICR[0] |= SYSCFG\_EXTICR1\_EXTI0\_PA;

NVIC\_EnableIRQ(EXTI0\_IRQn); // set the nvic

NVIC\_SetPriority(EXTI0\_IRQn, 0);// set priority to 0(highest)

};

void PWM\_init(int dutyCycle)

{

RCC->APB2ENR |= RCC\_APB2ENR\_TIM1EN;// enable clock connection for TIMER1

RCC->AHBENR |= RCC\_AHBENR\_GPIOEEN;// enable clock on GPIOE

GPIOE->MODER = (GPIOE->MODER & (~0x3C0C0000)) | 0x28080000; // PE.9,PE13,PE14 configured //to alternate mode

GPIOE->OTYPER &= ~(0x6200); // Push/pull for PE9,PE13,PE14

GPIOE->PUPDR &= ~(0x3C0C0000); // No pull-up,pull down for PE9,PE13,PE14

GPIOE->AFR[1] = (GPIOE->AFR[1] & ~(0xFF000F0)) | 0x2200020; // PE.9,PE.13 and PE.14

TIM1->PSC = 19;

TIM1->ARR = 3999; // around 100Hz or 0.01s

TIM1->CCMR1 |= 0x00000060; // channel 1 for PE.9

TIM1->CCMR2 = (TIM1->CCMR2 & ~0xF0F0) | 0x6060;// channel 3 and 4(PE.13 and PE.14 )

TIM1->CCR1 = dutyCycle; //determine the duty cycle, (CCR1/(ARR+1))\*100 == Duty\_cycle%

TIM1->CCR3 = dutyCycle; //determine the duty cycle

TIM1->CCR4 = dutyCycle; //determine the duty cycle

//Enable the Channel chosen to be output to the GPIO pin

TIM1->BDTR |= TIM\_BDTR\_MOE;// 0x00008000

TIM1->CCER |= 0x1101; // channel 1,3 and channel 4

//------------------------------------------------------

TIM1->CR1 |= TIM\_CR1\_CEN;

}

void setDutyCycle(int dutyCycle)

{

TIM1->CCR1 = dutyCycle; //determine the duty cycle, (CCR1/(ARR+1))\*100 == Duty\_cycle%

TIM1->CCR3 = dutyCycle; //determine the duty cycle

TIM1->CCR4 = dutyCycle; //determine the duty cycle

}

### Encoder Emulation

**From Laboratory 3: Task 3.3.** Include your schematics for the key internal and external components including:

* + The output transistor arrangements
  + The resistor connections
  + The input configuration
  + The LED connections

The high level reprentation:

*Graphical user interface, application, table

Description automatically generated*

The encoder block:

*Diagram

Description automatically generated*

The LED counter block:

Diagram, schematic

Description automatically generated

### Encoder Emulation (2)

**From Laboratory 3: Task 3.2.** Include the interrupt service routine(s) that responds to the encoder pulses and keeps the count updated.

#include <stdbool.h>

#include <stdlib.h>

#include "stm32f3xx.h"

//-------------------------------------------------ENCODER STATE VARIABLES & FUNCTIONS

static volatile bool direction = false; // true = clockwise, false = anti-clockwise

const int states[4] = { 0,2,3,1 }; // states of the encoder stored in an integer //array,{0b00,0b10,0b11,0b01}

static int state = 0; // state of the encoder

void timer\_init(void);// initialise timer based interrupt

void encoder\_signal(void); // emulate the encoder signal

void encoder\_pos(void); // CHA = PE.9 connected to PA.0, CHB= PE.8 connected to PA.1

static volatile int current\_state[2] = { 0,0 }; // index 0 = CHA, index 1= CHB

static volatile int last\_state[2] = { 0,0 };// index 1 = CHB, index 1 = CHB

//-------------------------------------------------------------------------

//------------------------------------------------------DISPLAY ENCODER PULSES ON LEDS

void ext\_interrupt1(void); // initialise external interrupts

void ext\_interrupt2(void); // initialise external interrupts

void interrupt\_pins\_init(void);// initialise the input external interrupt pins PA0,PA1

volatile int encoderCount = 0; // counter for encoder pulses

void LED\_init(void); // initialise LEDs on PE8,9,11-14

void displayLED(int); // display encoder count on the LEDs

//------------------------------------------------------------------------------------

void EXTI0\_IRQHandler() // external interrupt channel 0

{

if (EXTI->PR & EXTI\_PR\_PR0) // check source

{

EXTI->PR |= EXTI\_PR\_PR0; // clear flag\*

encoder\_pos(); // function to determine direction

if (abs(encoderCount) > 15)

{

encoderCount = 0;

}// reset when max 4-bit value is reached

displayLED(encoderCount);// display the count on the LED PE.11-14

}

};

void EXTI1\_IRQHandler() // external interrupt channel 1

{

if (EXTI->PR & EXTI\_PR\_PR1) // check source

{

EXTI->PR |= EXTI\_PR\_PR1; // clear flag\*

encoder\_pos();// function to determine direction

if (abs(encoderCount) > 15)

{

encoderCount = 0;

}// reset when max 4-bit value is reached

displayLED(encoderCount);// display the count on the LED PE.11-14

}

}

void TIM3\_IRQHandler()// Timer based interrupt

{

if ((TIM3->SR & TIM\_SR\_UIF) != 0) // Check interrupt source is from the 'Update' interrupt //flag

{

encoder\_signal();// generate singal using the state machine mechanism

}

TIM3->SR &= ~TIM\_SR\_UIF; // Reset 'update' interrupt flag in the SR register

}

int main(void)

{

LED\_init(); // initialise LEDs on PE8,9,11-14

interrupt\_pins\_init(); // initialise the input external interrupt pins PA0,PA1

ext\_interrupt1(); // initialise external interrupt on PA.0

ext\_interrupt2();// initialise external interrupt on PA.1

timer\_init(); // generate encoder signal on PE8,PE9 at 1Hz square wave

}

void encoder\_signal()

{

GPIOE->BSRRH = 0x300; // 0b11<<8, this turns OFF leds on PE8,9 to visualize the encoder //signal

if (!direction)// anti-clockwise direction

{

switch (state)

{

case 0:

GPIOE->BSRRL = (states[0] << 8); // 0b00 << 8

state = 1; // move on to the next state

break;

case 1:

GPIOE->BSRRL = (states[3] << 8); // 0b01 << 8

state = 2; // move on to the next state

break;

case 2:

GPIOE->BSRRL = (states[2] << 8); // 0b11 << 8

state = 3; // move on to the next state

break;

case 3:

GPIOE->BSRRL = (states[1] << 8); // 0b10 << 8

state = 0; // move on to the next state

break;

}

}

else if (direction) // clockwise direction

{

switch (state)

{

case 0:

GPIOE->BSRRL = (states[0] << 8); // 0b00 << 8

state = 1; // move on to the next state

break;

case 1:

GPIOE->BSRRL = (states[1] << 8); // 0b10 << 8

state = 2; // move on to the next state

break;

case 2:

GPIOE->BSRRL = (states[2] << 8); // 0b11 << 8

state = 3; // move on to the next state

break;

case 3:

GPIOE->BSRRL = (states[3] << 8); // 0b01 << 8

state = 0; // move on to the next state

break;

}

}

}

void ext\_interrupt1()

{

//Enable the system configuration controller to be connected to a system cloc

RCC->APB2ENR |= RCC\_APB2ENR\_SYSCFGEN;

//Remove the mask to enable an interrupt to be generated using the EXTI\_IMR //register

EXTI->IMR |= EXTI\_IMR\_MR0;

EXTI->RTSR |= EXTI\_RTSR\_TR0; // trigger on rising edge

EXTI->FTSR |= EXTI\_FTSR\_TR0; // trigger on falling edge

//The USR push button (blue button on the STM32F3discovery board) is connected to pin PA.0.

SYSCFG->EXTICR[0] |= SYSCFG\_EXTICR1\_EXTI0\_PA;

NVIC\_EnableIRQ(EXTI0\_IRQn); // set the nvic

NVIC\_SetPriority(EXTI0\_IRQn, 0);// set priority to 0

}

void ext\_interrupt2()

{

//Enable the system configuration controller to be connected to a system cloc

RCC->APB2ENR |= RCC\_APB2ENR\_SYSCFGEN;

//Remove the mask to enable an interrupt to be generated using the EXTI\_IMR //register

EXTI->IMR |= EXTI\_IMR\_MR1;

EXTI->RTSR |= EXTI\_RTSR\_TR1; // trigger on rising edge

EXTI->FTSR |= EXTI\_FTSR\_TR1; // trigger on falling edge

// Configure the second channel, PinA.1

SYSCFG->EXTICR[0] |= SYSCFG\_EXTICR1\_EXTI1\_PA;

NVIC\_EnableIRQ(EXTI1\_IRQn); // set the nvic

NVIC\_SetPriority(EXTI1\_IRQn, 0);// set priority to 1

}

void displayLED(int encoder\_count)

{

GPIOE->BSRRH = (0xF) << 11; // turn off all LEDs on PE11-14

GPIOE->BSRRL = abs(encoder\_count) << 11; // turn on LEDs by shifting bit to match PE11-14

}

void encoder\_pos()// function to determine direction

{

current\_state[0] = (GPIOA->IDR & 0x1); //Store Current State of CHA

current\_state[1] = (GPIOA->IDR & 0x2) >> 1; //Store Current State of CHB

if (current\_state[0] != last\_state[0]) // CHA triggered the interrupt

{

if (current\_state[0] ^ last\_state[1]) // CHA\_new XOR CHB\_old

{

++encoderCount;

}

else

{

--encoderCount;

}

last\_state[0] = current\_state[0]; // update CHA state

}

else if (current\_state[1] != last\_state[1])// CHB triggered the interrupt

{

if (last\_state[0] ^ current\_state[1]) // CHA\_old XOR CHB\_new

{

--encoderCount;

}

else

{

++encoderCount;

}

last\_state[1] = current\_state[1]; // update CHB state

}

}

void LED\_init() // initialise LEDs on PE8,9,11-14

{

RCC->AHBENR |= RCC\_AHBENR\_GPIOEEN;// Enable clock on GPIO port E

//-----------------LED Pins 8,9 set to output mode

GPIOE->MODER = (GPIOE->MODER & ~(0xF0000)) | 0x50000;// output mode "01" for pins(8,9)

GPIOE->OTYPER &= ~(0x300); // push/pull "00" for pins(8,9)

GPIOE->PUPDR &= ~(0xF0000); // no pullup, pull-down for pins(8,9)

//--------------------------------------------------------------------------------

GPIOE->MODER = (GPIOE->MODER & ~(0x3FC00000)) | (0x15400000); // PE.11-14 set to output //mode

GPIOE->OTYPER &= ~(0x7800); // Push/pull mode set for PE.11-14

GPIOE->PUPDR &= ~(0x3FC00000);// no pullup, pull down for PE.11-14

}

void interrupt\_pins\_init()

{

RCC->AHBENR |= RCC\_AHBENR\_GPIOAEN;// Enable clock on GPIO port A

GPIOA->MODER &= ~(0xF); // pins A.0+A.1 both set to input mode

}

void timer\_init()// initialise timer based interrupt

{

RCC->APB1ENR |= RCC\_APB1ENR\_TIM3EN; //Define the clock pulse toTIM3

TIM3->PSC = 65535;

TIM3->ARR = (int)121.0703; // every 1 second the interrupt is called

TIM3->CR1 |= TIM\_CR1\_CEN;//Set Timer Control Register to start timer

TIM3->DIER |= TIM\_DIER\_UIE; // Set DIER register to watch out for an 'Update' Interrupt //Enable (UIE) – or 0x00000001

NVIC\_EnableIRQ(TIM3\_IRQn); // Enable Timer 3 interrupt request in NVIC

}

## Position Measurement Exercise

This exercise brings together all the aspects of the prior labs to create a functional system. The aim is to create a dual-sensor position measurement using your emulated functions created previously. Remember that the code submission should be your own, and this should be reflected in commentary.

### Code Submission

Submit your code BOTH by copying in below and as an additional ‘.c’ file.

#include <stdbool.h>

#include <stdlib.h>

#include <math.h>

#include "stm32f3xx.h" // Device header

//#include "Throttle\_pedal.h"

#define PRESCALER1 35 // DAC/triangle wave incrementing in steps of 10, ISR frequency of 730 Hz

#define ARR1 303

#define PRESCALER2 85 // triangluar wave simuation occurs at 226 Hz

#define ARR2 407

#define MAX\_AMPLITUDE 3640 // triangle wave goes up to 3640 -> 0 -> 3640

//---------------------------------------FUNCTION PROTOTYYPES---------------------------------

void CountLEDs\_init(void); // Initialise ALL LEDs

//-------------------------------------------------POTENTIOMETER INIT

void DAC\_init(void); // initialise the DAC with a timer-based interrupt, PA.4

void triangle\_emulator(void); // write DAC values which output an analogue triangular signal

//-------------------------------------------------------------------

//-------------------------------------------------ENCODER EMULATOR

void encoder\_signal(void); // emulate the encoder signal on PB.12 and PB.13

void encoder\_init(void); // initialise the interrupt required to generate the encoder signal

void ext\_interrupt1\_init(void); // initialise interrupt pins, PA.1(HIGHER priority)

void ext\_interrupt2\_init(void);// initialise interrupt pins , PC.3

void encoder\_pos(void); // increment CW, decrement CCW

//--------------------------------------------------------------------

//-------------------------------------------------ADC INIT

void ADC\_init(void);// PF.2 as input

void wait(int);

void store\_ADC(void); // a function to store ADC values in an array

//--------------------------------------------------------------------

//--------------------------------------Button Interrupt PA.0, EXTI0 and STATES

void interrupt\_init(void);// button interrupt, PA.0

void test\_options(void);

enum tests{POTENTIOMETER,ENCODER,COMBINED\_TEST};

//-----------------------------------------------------------------------------

void writeLEDs(void); // MAIN function to run inside the int main() while loop

//-------------------------------------------------POTENTIOMETER INIT

static unsigned long count = 0; // counter variable for producing triangle wave

const float scaler1 = 117.419; // 3640/31 = scaling factor for DAC triangle wave

//-------------------------------------------------------------------

//-------------------------------------------------ENCODER EMULATOR

static volatile bool direction = false; // true = clockwise, false = anti-clockwise

const int states[4] = {0,2,3,1}; // states of the encoder stored in an integer array,{0b00,0b10,0b11,0b01}

static int state = 0; // state of the encoder

static int current\_state[2]={0,0}; // for position measurement

static int last\_state[2]={0,0};// for position

static int encoderCount = 113; // counter for encoder pulses

static int encoder\_dir\_counter = 0; // counter for inverting encoder direction to match triangular wave

const float scaler2 = 3.645; // encoderCounts converted to 5-bit, 113/31

//--------------------------------------------------------------------

//---------------------------------------------------------COMPUTING 5-POINT MOVING AVG

static int j=0;// index for cycling through array

float storage[5]={0};// store values from ADC

float sum=0,avg=0; // variables to find avg

//-------------------------------------------------------------------------------------

int main(void)

{

ADC\_init();// PF.2 as input

interrupt\_init();// button interrupt, PA.0

CountLEDs\_init();// init ALL LEDS(PE.8-PE.15)

ext\_interrupt1\_init(); // init interrupt on PA.1

ext\_interrupt2\_init();// init interrupt on PC.3

encoder\_init(); // initialise the encoder emulator on PB.12 and PB.13

DAC\_init(); // generate triangle wave at 1Hz frequency, PA.4

while(1)

{

writeLEDs(); // run the main function

}

}

//----------------------------------------------------------------ALL LEDS init---------------------------------

void CountLEDs\_init()

{

RCC->AHBENR |= RCC\_AHBENR\_GPIOEEN;// Enable clock on GPIO port C

GPIOE->MODER = (GPIOE->MODER & ~(0xFFFF0000))| 0x55550000;// output mode "01" for pins(8-15)

GPIOE->OTYPER &= ~(0xFF00); // push/pull "00" for pins(8-15)

GPIOE->PUPDR &= ~(0xFFFF0000); // no pullup, pull-down for pins(8-15)

}

//---------------------------------------------------------------POTENTIOMETER INIT--------------------------

void DAC\_init(void)

{

RCC->APB1ENR |= RCC\_APB1ENR\_TIM3EN; //Define the clock pulse to TIM3

TIM3->PSC = PRESCALER1;

TIM3->ARR = ARR1;// generate the triangle wave at 1Hz frequency

TIM3->CR1 |= TIM\_CR1\_CEN;//Set Timer Control Register to start timer

TIM3->DIER |= TIM\_DIER\_UIE; // Set DIER register to watch out for an 'Update' Interrupt Enable (UIE) – or 0x00000001

NVIC\_EnableIRQ(TIM3\_IRQn); // Enable Timer 3 interrupt request in NVIC

//--------------------DAC configuration

RCC->APB1ENR |= RCC\_APB1ENR\_DAC1EN;

GPIOA->MODER |= 0x300; // PA.4 set to analogue mode,Configure ports for DAC

DAC1->CR |= DAC\_CR\_BOFF1;

DAC1->CR |= DAC\_CR\_EN1;

//-------------------------------------

}

void TIM3\_IRQHandler() // Timer interrupt 1

{

if ((TIM3->SR & TIM\_SR\_UIF) !=0) // Check interrupt source is from the 'Update' interrupt flag

{

triangle\_emulator(); // generate the triangle wave

}

TIM3->SR &= ~TIM\_SR\_UIF; // Reset 'update' interrupt flag in the SR register

}

void triangle\_emulator(void)

{

DAC1->DHR12R1 = abs((count % (2\*MAX\_AMPLITUDE))-MAX\_AMPLITUDE); // triangle wave writing 3640-0-3640 to the DAC

count += 10;

// if (j>4){j=0;}// cycle through the storage array

// storage[j++]= (DAC1->DHR12R1);// store results in the array

store\_ADC();

}

//-------------------------------------------------------------------------------------------------------

//------------------------------------------------------------------------------ENCODER EMULATOR---------

void encoder\_init(void) // initialise the interrupt for timer interrupt 2

{

// Enable clock on GPIO port B

RCC->AHBENR |= RCC\_AHBENR\_GPIOBEN;

RCC->APB1ENR |= RCC\_APB1ENR\_TIM2EN; // enable clock for TIM2

GPIOB->MODER = (GPIOB->MODER & ~(0xF000000))| 0x5000000;// output mode "01" for PB(12,13)

GPIOB->OTYPER &= ~(0x3000); // push/pull "00" for pins(12,13)

GPIOB->PUPDR &= ~(0xF000000); // no pullup, pull-down for pins(12,13)

TIM2->PSC = PRESCALER2;

TIM2->ARR = ARR2;

TIM2->CR1 |= TIM\_CR1\_CEN;// timer action is set in motion with the ‘enable’ command

TIM2->DIER |= TIM\_DIER\_UIE; // Set DIER register to watch out for an 'Update' Interrupt Enable (UIE) – or 0x00000001

NVIC\_EnableIRQ(TIM2\_IRQn); // Enable Timer 3 interrupt request in NVIC

}

void ext\_interrupt1\_init(void) // initialise interrupt on PA.1

{

GPIOA->MODER &= ~(0xC); // pins A.1 set to input mode

GPIOA->PUPDR = (GPIOA->PUPDR & ~(0xC))|(0x4); // PA.1 with pullup enabled

RCC->APB2ENR |= RCC\_APB2ENR\_SYSCFGEN;//Enable the system configuration controller to be connected to a system cloc

RCC->AHBENR |= RCC\_AHBENR\_GPIOAEN;// Enable clock on GPIO port A

//Remove the mask to enable an interrupt to be generated using the EXTI\_IMR register

EXTI->IMR |= EXTI\_IMR\_MR1;

EXTI->RTSR |= EXTI\_RTSR\_TR1;//Set interrupt trigger to be rising edge

EXTI->FTSR |= EXTI\_RTSR\_TR1;//Set interrupt trigger to be rising edge

SYSCFG->EXTICR[0] |= SYSCFG\_EXTICR1\_EXTI1\_PA;// PA.1

NVIC\_EnableIRQ(EXTI1\_IRQn); // set the nvic

NVIC\_SetPriority(EXTI1\_IRQn,0);// set priority to 1, second highest priority

}

void EXTI1\_IRQHandler() // ext interrupt on PA.1

{

if (EXTI->PR & EXTI\_PR\_PR1) // check source

{

EXTI->PR |= EXTI\_PR\_PR1; // clear flag\*

encoder\_pos();

}

};

void ext\_interrupt2\_init(void)// initialise interrupt on PC.3

{

GPIOC->MODER &= ~(0xC0); // PC.3 set to input mode

GPIOC->PUPDR = (GPIOC->PUPDR & ~(0xC0))|(0x40); // PC.3 with pullup enabled

RCC->APB2ENR |= RCC\_APB2ENR\_SYSCFGEN;//Enable the system configuration controller to be connected to a system cloc

RCC->AHBENR |= RCC\_AHBENR\_GPIOCEN;// Enable clock on GPIO port C

EXTI->IMR |= EXTI\_IMR\_MR3;//Remove the mask to enable an interrupt to be generated using the EXTI\_IMR register

EXTI->RTSR |= EXTI\_RTSR\_TR3;//Set interrupt trigger to be rising edge

EXTI->FTSR |= EXTI\_RTSR\_TR3;//Set interrupt trigger to be rising edge

SYSCFG->EXTICR[0] |= SYSCFG\_EXTICR1\_EXTI3\_PC;

NVIC\_EnableIRQ(EXTI3\_IRQn); // set the nvic

NVIC\_SetPriority(EXTI3\_IRQn,0);// set priority to 0, highest priority

}

void EXTI3\_IRQHandler() // ext interrupt on PC.3

{

if (EXTI->PR & EXTI\_PR\_PR3) // check source

{

EXTI->PR |= EXTI\_PR\_PR3; // clear flag\*

encoder\_pos();

}

};

void TIM2\_IRQHandler()

{

if ((TIM2->SR & TIM\_SR\_UIF) !=0) // Check interrupt source is from the ‘Update’ interrupt flag

{

encoder\_signal();

}

TIM2->SR &= ~TIM\_SR\_UIF; // Reset ‘Update’ interrupt flag in the SR register

}

void encoder\_signal() // emulates the encoder signal using state machine mechanism

{

if (encoder\_dir\_counter > 112) // reset when 113 encoder counts have occurred

{

encoder\_dir\_counter =0 ;// reset

direction = !direction; // toggle direction

state = 0; // reset state

}

GPIOB -> BSRRH = 0x3000; // 0b11<<12, making both of the signals low to emalate a square wave signal

if (!direction)// anti-clockwise direction

{

switch(state)

{

case 0:

GPIOB->BSRRL = (states[0]<<12); // 0b00 << 12

state = 1; // move on to the next state

break;

case 1:

GPIOB->BSRRL = (states[3]<<12); // 0b01 << 12

state = 2; // move on to the next state

break;

case 2:

GPIOB->BSRRL = (states[2]<<12); // 0b11 << 12

state = 3; // move on to the next state

break;

case 3:

GPIOB->BSRRL = (states[1]<<12); // 0b10 << 12

state = 0; // move on to the next state

break;

}

}

else if(direction) // clockwise direction

{

switch(state)

{

case 0:

GPIOB->BSRRL = (states[0]<<12); // 0b00 << 12

state = 1; // move on to the next state

break;

case 1:

GPIOB->BSRRL = (states[1]<<12); // 0b10 << 12

state = 2; // move on to the next state

break;

case 2:

GPIOB->BSRRL = (states[2]<<12); // 0b11 << 12

state = 3; // move on to the next state

break;

case 3:

GPIOB->BSRRL = (states[3]<<12); // 0b01 << 12

state = 0; // move on to the next state

break;

}

}

}

void encoder\_pos() // function to determine direction of encoder from external interrupts

{

current\_state[0] = (GPIOA -> IDR & 0x2)>>1; //Store Current State of CHA

current\_state[1] = (GPIOC -> IDR & 0x8)>>3; //Store Current State of CHB

if (current\_state[0]!= last\_state[0]) // CHA triggered the interrupt

{

if (current\_state[0] ^ last\_state[1]) // CHA\_new XOR CHB\_old

{

++encoderCount;

}

else

{

--encoderCount;

}

last\_state[0] = current\_state[0]; // update CHA state

}

else if (current\_state[1] != last\_state[1])// CHB triggered the interrupt

{

if (last\_state[0] ^ current\_state[1]) // CHA\_old XOR CHB\_new

{

--encoderCount;

}

else

{

++encoderCount;

}

last\_state[1] = current\_state[1]; // update CHB state

}

++encoder\_dir\_counter; // increment variable for changing direction

}

//---------------------------------------------------------------------------------------------------------------

//---------------------------------------------------------------------------------------ADC INIT-----

void wait(int c)

{

int ticks=0;

while (ticks<c)// exit when ticks reach the count

{

++ticks;

}

}

void ADC\_init()

{

ADC1->CR &= ~(0x30000000); // 0b00 for the ADVREGEN = reset

ADC1->CR |= (0x10000000);// 0b01 for the ADVREGEN = enable

RCC->AHBENR |= RCC\_AHBENR\_GPIOFEN;// enable clock on GPIOF

GPIOF->MODER |= 0x30; // analogue mode on PF.2, "11"

wait(100);// 10us delay function

ADC1->CR &= ~(0x40000000);// 0b0 0 single-ended calibration

ADC1->CR |= 0x80000000;// begin calibration

while (ADC1->CR & 0x80000000) {} // wait for ADCAL to return 0

//------------------Enabling clock peripherials

RCC->CFGR2 = (RCC->CFGR2 & ~0x1F0)|RCC\_CFGR2\_ADCPRE12\_DIV2;// clear registor first then 10001: PLL clock divided by 2

RCC->AHBENR |= RCC\_AHBENR\_ADC12EN;

ADC1\_2\_COMMON->CCR |= 0x00010000;

//---------------------------------------------

ADC1->CR &= ~(0xC);// set ADSTART and JADSTART to 0, "00 00"

// ADC1->CFGR = (ADC1->CFGR & ~0x18)|(0x10);// 8 bit resolution

ADC1->CFGR &= ~(0x18); // 12 bit resolution

ADC1->CFGR &= ~(0x20);// align right

ADC1->CFGR &= ~(0x2000);// no continous

ADC1->SQR1 = (ADC1->SQR1& ~0x3C0)|0x280;// channel 10 in SQR1, "01010"

ADC1->SQR1 &= ~(0xF);// only one conversion, L is "0000"

ADC1->SMPR1 |= 0x38; // 601.5 ADC clock cycles

ADC1->CR |= 0x1; // ADEN set to high to enable ADC

while (!(ADC1->ISR & 0x01)) {}// wait for ARDY flag to go high

}

int ADC\_read() // a function to continously read from the ADC, return the integer value of the ADC

{

ADC1->CR |= 0x4; // enable ADC

while (!(ADC1->ISR & 0x4)) {}// wait for EOC flag to go high

// float returner = round((ADC1->DR)/scaler1); // variable to scale the traingle wave values

return (ADC1->DR);

}

void store\_ADC() // store triangle wave inside an array for 5-point moving average

{

if (j>4){j=0;}// cycle through the storage array

storage[j++]= ADC\_read();// store results in the array

}

//-------------------------------------------------------------------------------------------------------------------------------

//------------------------------------------------BUTTON INTERRUPT INIT----------------------------------------------------------

void interrupt\_init(void)

{

//Enable the system configuration controller to be connected to a system cloc

RCC->APB2ENR |= RCC\_APB2ENR\_SYSCFGEN;

//Remove the mask to enable an interrupt to be generated using the EXTI\_IMR register

EXTI->IMR |= EXTI\_IMR\_MR0;

//Set interrupt trigger to be rising edge

EXTI->RTSR |= EXTI\_RTSR\_TR0;

//The USR push button (blue button on the STM32F3discovery board) is connected to pin PA.0.

SYSCFG->EXTICR[0] |= SYSCFG\_EXTICR1\_EXTI0\_PA;

NVIC\_EnableIRQ(EXTI0\_IRQn); // set the nvic

NVIC\_SetPriority(EXTI0\_IRQn,0);// set priority to 0, highest priority

};

void EXTI0\_IRQHandler() // button interrupt on PA.0

{

if (EXTI->PR & EXTI\_PR\_PR0) // check source

{

EXTI->PR |= EXTI\_PR\_PR0; // clear flag\*

test\_options(); // switch state

}

};

enum tests testing = POTENTIOMETER;

void test\_options() // state machine mechanism to respond to blue push button

{

switch (testing)

{

case POTENTIOMETER:

testing = ENCODER;

break;

case ENCODER:

testing = COMBINED\_TEST;

break;

case COMBINED\_TEST:

testing = POTENTIOMETER;

break;

}

}

//--------------------------------------------------MAIN FUNCTION TO RUN INSIDE int main()-----

void writeLEDs()

{

switch (testing)

{

case POTENTIOMETER:

GPIOE->BSRRH = (0xFE00); // turn OFF Leds PE.9-PE.15

if(!(GPIOE->IDR & 0x100)) // turn ON Led PE.8 if OFF

{

GPIOE->BSRRL = 0x100;

}

// the following scales the ADC value to 5-bit then converts that float into integer

GPIOE->BSRRL = ((int)round((ADC\_read()/scaler1))) << 11; // turn ON Leds by shifting bits, the ADC is scaled to 5-bits

break;

case ENCODER:

GPIOE->BSRRH = (0xFD00); // turn OFF Leds PE.8, PE.10-15

if(!(GPIOE->IDR & 0x200)) // turn ON Led PE.9 if OFF

{

GPIOE->BSRRL = 0x200;

}

// scale the encoder count to 5-bit then cast to integer

GPIOE->BSRRL = (int)(round(abs(encoderCount)/scaler2)) << 11; // turn on LEDs by shifting bit to match PE11-15

break;

case COMBINED\_TEST:

GPIOE->BSRRH = (0xFB00); // turn OFF Leds PE.8-9, PE.11-15

if(!(GPIOE->IDR & 0x400)) // turn ON Led PE.10 if OFF

{

GPIOE->BSRRL = 0x400;

}

\_\_disable\_irq(); // disable interrupts

for (size\_t i=0;i<5;++i)

{

sum += storage[i]/scaler1;// scale the array values to fit 5-bit

}

avg = ((sum/5)+(abs(encoderCount)/scaler2))/2; // sum/5 = avg of POT , (avg of POT + encoder)/2

\_\_enable\_irq(); // enable interrupts

GPIOE->BSRRL = ((int)round(avg)) << 11; // cast the decimal value to int after rounding

sum = 0;// reset sum to zero for next iteration

break;

}

}

//-----------------------------------------------------------------------------------------------------------