Wyatt Tack
Joshua Lin
EE 329-01 F'24
Group
2024-Oct-11

EE 329 A4

This code is designed to validate the use of Timers, with multiple sections using/measuring outputs due to the onboard timers. The STM32L4 has multiple on board clocks, the one we were using was a 32-Bit timer TIM2, and its mid way capture register CC1. Using this timer we measured interrupt time, the smallest frequency countable on the interrupt, and then applied it into a reaction timer system. The reaction timer worked as waiting for an input button, prompting the user to get ready, randomly turning on an LED, and logging the decimal seconds taken to turn on the LED.

Link to YouTube Presentation:

https://youtu.be/eCX4bWneXJQ

Calculations:

Equation A4.a(a): Frequency Calculations:

Counts of 4MHz clock for 5KHz wave:

$$\text{Period} = \frac{4MHz}{15KHz} - 1 \rightarrow \textbf{Period = 799}$$

$$\text{25\% DC} = \frac{CC1\ Counts}{800} - 1 \rightarrow \textbf{CC1 Counts = 199}$$

Captures:

Figure A4.a(a): Drawn 4MHz and 5KHz clock sketches (Not to Scale)

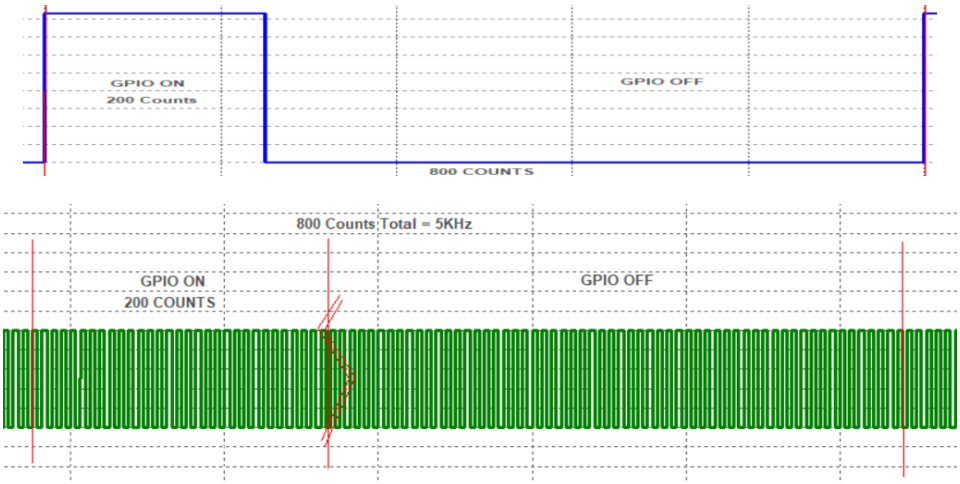


Figure A4.a(b): Oscilloscope 5KHz 25%DC Capture:

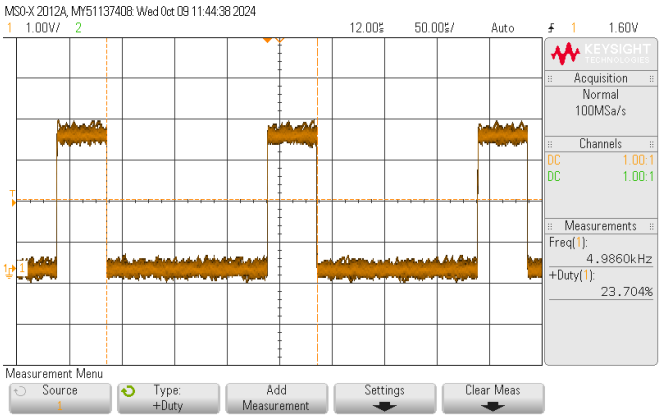


Figure A4.b(a): Oscilloscope MCO ISR Length Capture:

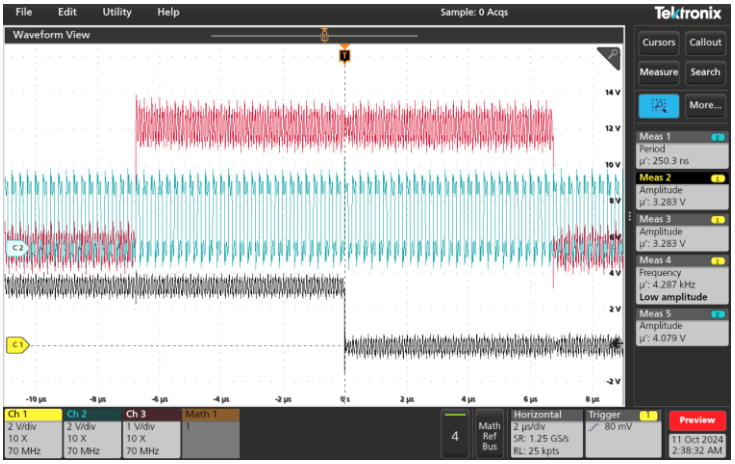


Figure A4.c(a): Oscilloscope CCR1 Smallest Output Capture:

Obtained Data:

Table A4.b(a): ISR MCO instruction length

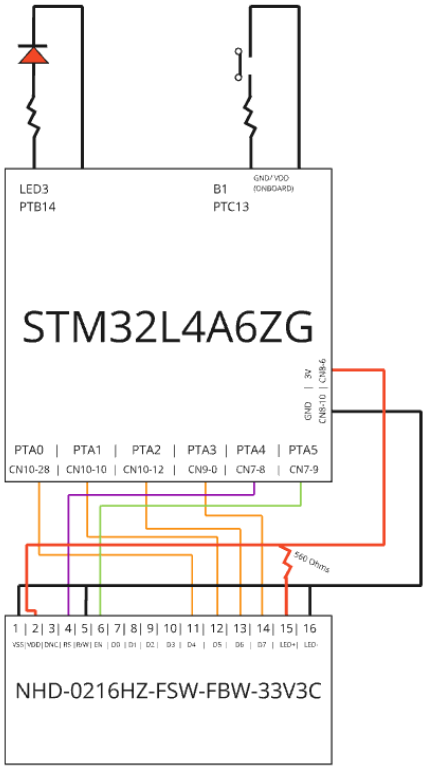
MCO CYCLES PER ISR	ISR TIME
55 Cycles per ISR	13.538 uS

Table A4.c(a): CCR1 Smallest Value

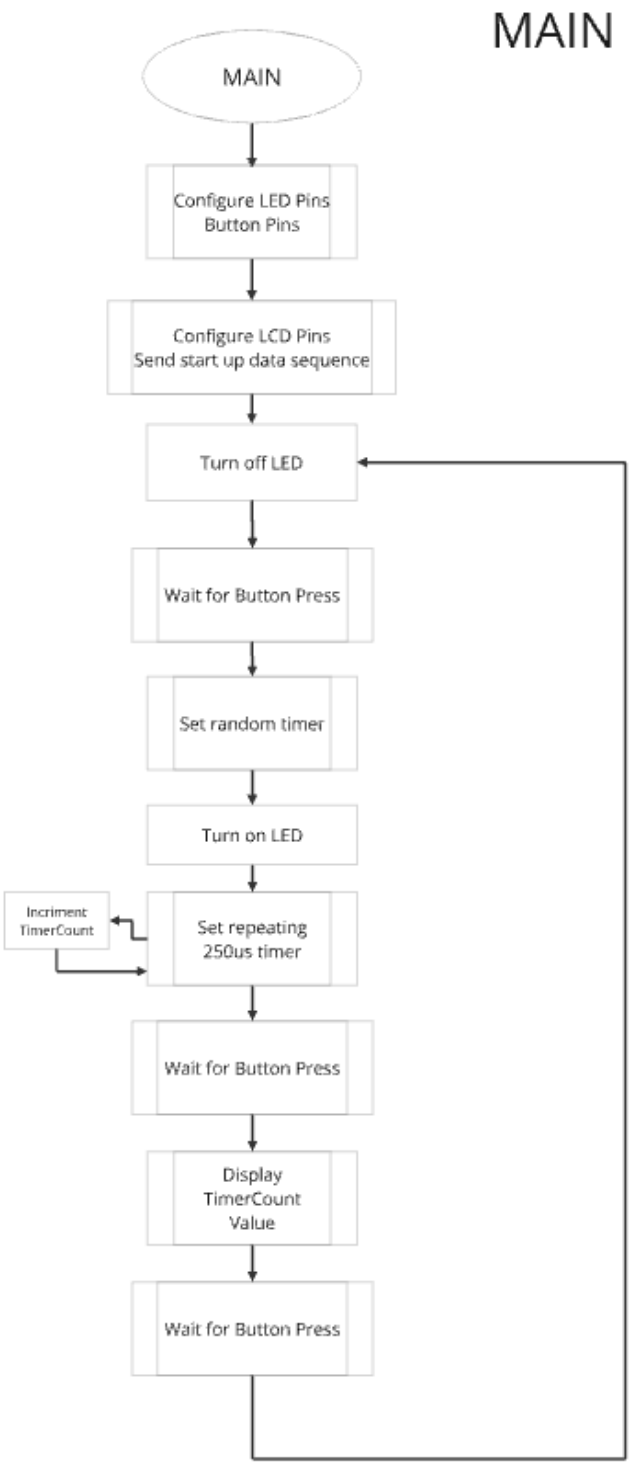
SMALLEST CC1 COUNT	SMALLEST ISR TIME
65 Cycles per ISR	13.597 uS

smallest CC1 count is at/close to cycles per ISR count

Figure A4(a): OnBoard LED/Button and 16x2 LCD Wiring



Psuedocode Flow Chart:



Formatted Source Code part B main.h:

```
/**
  ******************************************************************************
  * @file           : main.h
  * project                    : EE329 Lab A4
  * author                     : Wyatt Tack (wwt) - wtack@calpoly.edu
  * date                       : 10/9/2024
  * firmware           : ST-Link V1
  * @attention         : Copyright (c) 2024 STMicroelectronics. All rights
reserved.
  ******************************************************************************
  *
  *     main header for defines for C and stm32 headers/hal
  *
  ******************************************************************************
  */

#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Created defines and function prototypes -----------------------------------*/
#define PERIOD (0xFFFFFFFF);
#define DUTY_CYCLE (399);

void Led_Config(void);
void setup_TIM2( int iDutyCycle );
void TIM2_IRQHandler(void);
void setup_MCO_CLK(void);

/* Includes ------------------------------------------------------------------*/
#include "stm32l4xx_hal.h"

/* Exported functions prototypes ---------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);


#ifdef __cplusplus
}
#endif

#endif
```

Formatted Source Code part B main.c:

```
/**
  ******************************************************************************
  * @file                           : main.c
  * project                         : EE329 Lab A4
  * author                          : Wyatt Tack (wwt) - wtack@calpoly.edu
  * date                            : 10/9/2024
  * firmware                        : ST-Link V1
  * @attention                      : Copyright (c) 2024 STMicroelectronics. All rights reserved.
  ******************************************************************************
  *
  *                Device uses exclusively interupts to produce 5KHz 50%DC from internal Tim2
  *                at 4MHz. Uses GPIOC pin 0 to output 5KHz 50%DC wave, GPIOC Pin 1 to signal
  *                entering and leaving ISR (every 400 cycles of 4MHz), and GPIOA pin 8 to
  *                match 4MHz clock.
  *
  ******************************************************************************
  */

#include "main.h"

int main(void)
{
                  //Initialize clock and delay configs
                  HAL_Init();
                  SystemClock_Config();
                  Led_Config();
                  setup_MCO_CLK();
                  setup_TIM2 (DUTY_CYCLE); //400 ticks for each flip

                  // infinite loop to avoid break - program done in ISRs
  while (1){;}
}


//------------------------- ISR ----------------------------------------

void TIM2_IRQHandler(void) {
   GPIOC->BSRR |= (GPIO_PIN_1);       //toggle in ISR
   if (TIM2->SR & TIM_SR_CC1IF) {      // triggered by CCR1 event ...
      TIM2->SR &= ~(TIM_SR_CC1IF);     // manage the flag
      GPIOC->ODR ^= (GPIO_PIN_0);     // Flip GPIO each 400 ticks
      TIM2->CCR1 = TIM2->CNT + DUTY_CYCLE               // reset counter
   }
   if (TIM2->SR & TIM_SR_UIF) {        // triggered by ARR event ...
      TIM2->SR &= ~(TIM_SR_UIF);       // manage the flag
   }
   GPIOC->BRR |= (GPIO_PIN_1);                          //toggle out ISR
}


//-----------------------------------------------------------------------
void setup_TIM2( int iDutyCycle ) {
   RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;           // enable clock for TIM2
   TIM2->DIER |= (TIM_DIER_CC1IE | TIM_DIER_UIE);  // enable event gen, rcv CCR1
   TIM2->ARR = PERIOD;                             // ARR = T = counts @4MHz
   TIM2->CCR1 = iDutyCycle;                        // ticks for duty cycle
   TIM2->SR &= ~(TIM_SR_CC1IF | TIM_SR_UIF);       // clr IRQ flag in status reg
   NVIC->ISER[0] |= (1 << (TIM2_IRQn & 0x1F));     // set NVIC interrupt: 0x1F
   __enable_irq();                                 // global IRQ enable
   TIM2->CR1 |= TIM_CR1_CEN;                       // start TIM2 CR1
}

void setup_MCO_CLK(void) {
   // Enable MCO, select MSI (4 MHz source)
   RCC->CFGR = ((RCC->CFGR & ~(RCC_CFGR_MCOSEL)) | (RCC_CFGR_MCOSEL_0));
   // Configure MCO output on PA8
   RCC->AHB2ENR   |=  (RCC_AHB2ENR_GPIOAEN);
   GPIOA->MODER   &= ~(GPIO_MODER_MODE8);                // clear MODER bits
   GPIOA->MODER   |=  (GPIO_MODER_MODE8_1);              // set alternate function mode
   GPIOA->OTYPER  &= ~(GPIO_OTYPER_OT8);                 // Push-pull output
   GPIOA->PUPDR   &= ~(GPIO_PUPDR_PUPD8);                // no resistor
   GPIOA->OSPEEDR |=  (GPIO_OSPEEDR_OSPEED8);   // high speed
   GPIOA->AFR[1]  &= ~(GPIO_AFRH_AFSEL8);                // select MCO function
}


//-----------------------------------------------------------------------
void Led_Config(void)
{                                 // configure GPIO pins PC0-3 for:
                                  // output mode, push-pull, no pull up or pull down, high speed
  RCC->AHB2ENR   |=  (RCC_AHB2ENR_GPIOCEN);
  GPIOC->MODER   &= ~(GPIO_MODER_MODE0 | GPIO_MODER_MODE1);
  GPIOC->MODER   |=  (GPIO_MODER_MODE0_0 | GPIO_MODER_MODE1_0);
  GPIOC->OTYPER  &= ~(GPIO_OTYPER_OT0 | GPIO_OTYPER_OT1);
  GPIOC->PUPDR   &= ~(GPIO_PUPDR_PUPD0 | GPIO_PUPDR_PUPD1);
  GPIOC->OSPEEDR |=  ((3 << GPIO_OSPEEDR_OSPEED0_Pos) | (3 << GPIO_OSPEEDR_OSPEED1_Pos));
  GPIOC->BRR |= (GPIO_PIN_0 | GPIO_PIN_1);
}

// System
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

void Error_Handler(void)
{

  __disable_irq();
  while (1)
  {
  }

}

#ifdef  USE_FULL_ASSERT

void assert_failed(uint8_t *file, uint32_t line)
{

}
#endif
```

Formatted Source Code part D main.h:

```
/**
  ******************************************************************************
  * @file                    : main.h
  * project                  : EE329 Lab A4
  * author                   : Wyatt Tack (wwt) - wtack@calpoly.edu
  * date                     : 10/9/2024
  * firmware                 : ST-Link V1
  * @attention               : Copyright (c) 2024 STMicroelectronics. All rights reserved.
  ******************************************************************************
  *
  *       main header for defines for C and stm32 headers/hal
  *
  ******************************************************************************
  */

#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Created defines and function prototypes -----------------------------------*/
#define BUTTON_PORT GPIOC
#define LED_PORT GPIOB
#define BUTTON_PRESS ((((~(GPIOC->IDR)) & GPIO_PIN_13) >> 13))

void Button_Config(void);
void Button_Press(void);
void Write_Time(int timeTaken);
void Led_Config(void);
void setup_TIM2( int iDutyCycle );
void set_TIM2( int period );
void TIM2_IRQHandler(void);
void setup_MCO_CLK(void);
void setup_RNG(void);

/* Includes ------------------------------------------------------------------*/
#include "stm32l4xx_hal.h"

/* Exported functions prototypes ---------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);


#ifdef __cplusplus
}
#endif

#endif
```

Formatted Source Code part D main.c:

```
/**
  ******************************************************************************
  * @file           : main.c
  * @project        : EE329 Lab A4
  * @author         : Wyatt Tack (wat) - wtack@calpoly.edu
  * @board          : F5-disk V1
  * @firmware       : F5-disk V1
  * @attention      : Copyright (c) 2024 STMicroelectronics. All rights reserved.
  ******************************************************************************
  *
  * Device uses interrupts to act as a reaction timer game. User is prompted to
  * press button, and then press again once the LED lights after a randomly
  * generated number of seconds. Once pressed, interrupts are used to monitor
  * reaction speed.
  * LED defined at GPIOA PTB14 (on board LED3 RED)
  * Button defined at GPIOC PTC13 (on board B1)
  * LCD defined at GPIOA PTA8-3 (4 bit data), PTA4-5 (RS and EN)
  *
  ******************************************************************************
  */

#include "main.h"
#include "lcd.h"
#include "delay.h"

uint32_t usReaction = 0;
uint8_t myflag = 0;

int main(void)
{
                                        //Initialize clock and delay/LCD configs
                                        HAL_Init();
                                        SystemClock_Config();
                                        delay_init();
                                        Led_Config();
                                        Button_Config();
                                        LCD_init();
                                        setup_RNG();
                                        setup_TIM2(0xFFFFFFFF);
                                        //------------------------ LCD VARIABLES -----
                                        //new lcd_write/openings on LCD
                                        uint8_t origin[2] = {0,0};
                                        uint8_t line2[2] = {0,1};
                                        uint8_t openingLine1[16] = "RK 329 A4 REACT ";
                                        uint8_t openingLine2[16] = "PUSH SW TO THIS ";
                                        uint8_t openingLine3[16] = "    Ready?   ";
                                        uint8_t openingLine4[16] = "  Too Fast Dude ";
                                        uint32_t timeTaken = 0;
                                        uint8_t breakFlag;


                                        //---------------------- MAIN EXECUTION -----
                                        while (1){
//Opening Sequence
                                        myflag = 0;
                                        breakFlag = 0;
                                        GPIOA->ODR |= (GPIO_PIN_14);
                                        LCD_set_cursor(origin);
                                        LCD_write_string(openingLine1);
                                        LCD_set_cursor(line2);
                                        LCD_write_string(openingLine2);
//Button Pushed, get ready
                                        Button_Press();
                                        LCD_set_cursor(line2);
                                        LCD_write_string(openingLine3);
//Set random timer

                                        set_TIM ((((RNG->DR)% 0x3120C0) + 0x7A1200)); //Set timer for random 0-8Seconds
                                        while (myflag == 0){
                                            if(!BUTTON_PRESS){                                          //if pressed too early
                                                breakFlag = 1;
                                                LCD_set_cursor(line2);
                                                LCD_write_string(openingLine4);
                                                while(!BUTTON_PRESS);     //once released break and skip rest of code
                                                break;
                                            }
                                        }
                                        if(!breakFlag) //end sequence only if button wasn't early
//Random Time Finished
                                        LED_PORT->ODR |= (GPIO_PIN_14); //Light On
                                        usReaction = 0;
                                        set_TIM (990); //Set timer for counting
                                        while (BUTTON_PRESS) if(usReaction > 10000000) break; //wait for 12S or keypress
                                        timeTaken = usReaction;                             //log reaction time
                                        while (!BUTTON_PRESS);//wait for button release
                                        Write_Time(timeTaken);
                                        }
//Wait for button to replay
                                        Button_Press();
                                        }
}

//-----------------------------------------------------------
void Write_Time(int timeTaken){
                                        //Takes in time to press button in uS and parses it to data, then writes
                                        //results onto LCD screen.
                                        //Parse total time into small times
                                                                        uint8_t Sec = timeTaken/1000000;
                                                                        uint8_t MSecRun = timeTaken/100000 - 10*Sec;
                                                                        uint8_t MSecTen = timeTaken/10000 - 10*Sec - 10*MSecRun;
                                                                        uint8_t MSecOne = timeTaken/1000 - 1000*Sec - 100*MSecRun - 10*MSecTen;

                                        //Write Screen
                                                                        uint8_t line2[2] = {0,1};
                                                                        uint8_t endline1[16] = "TIME = 0.000 s  ";
                                                                        LCD_set_cursor(line2);
                                                                        LCD_write_string(endline1);

                                        //Write Data
                                                                        uint8_t clockSec[2] = {7,1};
                                                                        uint8_t clockMSecRun[2] = {9,1};
                                                                        uint8_t clockMSecTen[2] = {10,1};
                                                                        uint8_t clockMSecOne[2] = {11,1};
                                                                        uint8_t clockEnd[2] = {12,1};
                                                                        LCD_set_cursor(clockSec);
                                                                        LCD_write_char(Sec + 0x30);
                                                                        LCD_set_cursor(clockMSecRun);
                                                                        LCD_write_char(MSecRun + 0x30);
                                                                        LCD_set_cursor(clockMSecTen);
                                                                        LCD_write_char(MSecTen + 0x30);
                                                                        LCD_set_cursor(clockMSecOne);
                                                                        LCD_write_char(MSecOne + 0x30);
                                                                        LCD_set_cursor(clockEnd);
                                                                        LCD_write_char(' ');
                                                                        uint8_t endline2[16] = "  Too Slow Dude ";
                                                                        if(usReaction > 4000000){
                                                                            LCD_set_cursor(line2);
                                                                            LCD_write_string(endline2);
                                                                        }

}

//-------------------------- IRH -------------------------------------
void TIM2_IRQHandler(void) {
    if (TIM2->SR & TIM_SR_CC1IF) {      // triggered by CCR1 event ...
        TIM2->SR &= ~(TIM_SR_CC1IF);    // manage the flag
        //myFlag = 1;     // Flip GPIO each 400 ticks
        //usReaction = usReaction + 250;
        //TIM2->EGR |= TIM_EGR_UG;
    }

    if (TIM2->SR & TIM_SR_UIF) {        // triggered by ARR event ...
                                        TIM2->SR &= ~(TIM_SR_UIF);      // manage the flag
                                        usReaction = usReaction + 250;  //increment count
                                        myflag = 1;                                                                                 //set flag for main
                                        myflag = 1;
    }
}

//-----------------------------------------------------------
void setup_RNG(void) {
    RCC->AHB2ENR |= RCC_AHB2ENR_RNGEN;  // enable clock for RNG
    RCC->CR&CR |= (RCC_CR&CR_RNGPDMD);
    RNG->CR |= (RNG_CR_RNGEN | RNG_CR_IE);
    RNG->SR |= (RNG_SR_DRDY);           // enable RNG
}

void set_TIM( int period ) {
    RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;           // enable clock for TIM2
    TIM2->DIER |= (TIM_DIER_CC1IE | TIM_DIER_UIE);  // enable event gen, rcv CCR1
    TIM2->ARR = period;                                                                             // ticks for ARR Period
    TIM2->SR &= ~(TIM_SR_CC1IF | TIM_SR_UIF);      // clr IRQ flag in status reg
    NVIC->ISER[0] |= (1 << (TIM2_IRQn & 0x1F));     // set NVIC interrupt: 0x1F
     enable irq(1);                                 // global IRQ enable
    TIM2->CR1 |= TIM_CR1_CEN;                                                                                                       // start TIM2 CR1
}

void setup_TIM2( int period ) {
    RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;           // enable clock for TIM2
    TIM2->DIER |= (TIM_DIER_CC1IE | TIM_DIER_UIE);  // enable event gen, rcv CCR1
    TIM2->ARR = period;                                                                                                 // ARR = T = counts @4MHz
                                                                    //PERIOD;
    //TIM2->CCR1 = (duty*period);                   // ticks for duty cycle
    TIM2->SR &= ~(TIM_SR_CC1IF | TIM_SR_UIF);      // clr IRQ flag in status reg
    NVIC->ISER[0] |= (1 << (TIM2_IRQn & 0x1F));     // set NVIC interrupt: 0x1F
     enable irq(1);                                 // global IRQ enable
    TIM2->CR1 |= TIM_CR1_CEN;                                                                                                       // start TIM2 CR1
}
/*
void setup_MCO_CLK(void) {
  // Enable MCO, select MSI (4 MHz source)
  RCC->CFGR = ((RCC_CFGR_MCOSEL) | (RCC_CFGR_MCOSEL_0));
  // Configure MCO output on PA8
  RCC->AHB2ENR   |=  (RCC_AHB2ENR_GPIOAEN);
  GPIOA->MODER   &= ~(GPIO_MODER_MODE8);            // clear MODER bits
  GPIOA->MODER   |=  (GPIO_MODER_MODE8_1);          // set alternate function mode
  GPIOA->OTYPER  &= ~(GPIO_OTYPER_OT8);             // Push-pull output
  GPIOA->PUPDR   &= ~(GPIO_PUPDR_PUPD8);            // no resistor
  GPIOA->OSPEEDR |=  (GPIO_OSPEEDR_OSPEED8);        // high speed
  GPIOA->AFR[1]  &= ~(GPIO_AFRH_AFSEL8);            // select MCO function
}
*/

//-------------------------- GPIO ------------------------------------
void Button_Press(void){
                                        while (BUTTON_PRESS)delay_us(100);
                                        delay_us(100);
                                        while (!BUTTON_PRESS)delay_us(100);
                                        delay_us(100);
}

void Button_Config(void)
{
                                                                    // configure GPIO pin PC13 for:
                                                                    // Button pin initialize - input, on board PU
                                        RCC->AHB2ENR   |=  (RCC_AHB2ENR_GPIOCEN);
                                        BUTTON_PORT->MODER   &= ~(GPIO_MODER_MODE13);
                                        BUTTON_PORT->PUPDR   &= ~(GPIO_PUPDR_PUPD13);
}

void Led_Config(void)                                               // configure GPIO pin PB14 for:
{                                                                   // output mode, push-pull, no pull up or pull down, high speed

  RCC->AHB2ENR   |=  (RCC_AHB2ENR_GPIOBEN);
  LED_PORT->MODER   &= ~(GPIO_MODER_MODE14);
  LED_PORT->MODER   |=  (GPIO_MODER_MODE14_0);
  LED_PORT->OTYPER  &= ~(GPIO_OTYPER_OT14);
  LED_PORT->PUPDR   &= ~(GPIO_PUPDR_PUPD14);
  LED_PORT->OSPEEDR |=  (3 << GPIO_OSPEEDR_OSPEED14_Pos);
  LED_PORT->ODR |= (GPIO_PIN_14);
}
//-------------------------- SYSTEM ----------------------------------
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initialize the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

void Error_Handler(void)
{

    disable irq();
  while (1)
  {
  }

}

#ifdef  USE_FULL_ASSERT

void assert_failed(uint8_t *file, uint32_t line)
{

}
#endif
```

Formatted Source Code delay.h:

```
/**
  ******************************************************************************
  * @file               : delay.h
  * project             : EE329 Lab A3
  * author              : Wyatt Tack (wwt) - wtack@calpoly.edu
  * date                : 10/1/2024
  * firmware            : ST-Link V1
  * @attention          : Copyright (c) 2024 STMicroelectronics. All rights
reserved.
  ******************************************************************************
  *
  *   main header for defines for delay.h
  *
  ******************************************************************************
  */

#ifndef INC_DELAY_H_
#define INC_DELAY_H_

#include "stm32l4xx_hal.h"
void SysTick_Init(void);
void delay_us(const uint32_t time_us);

#endif /* INC_DELAY_H_ */
```

Formatted Source Code delay.c:

```
/**
  ******************************************************************************
  * @file               : delay.c
  * project             : EE329 Lab A3
  * author              : Wyatt Tack (wwt) - wtack@calpoly.edu
  * date                : 10/1/2024
  * firmware            : ST-Link V1
  * @attention          : Copyright (c) 2024 STMicroelectronics. All rights
reserved.
  ******************************************************************************
  *
  *   Functions for using SysTick clock for software delays. Provided
  *   on behalf of EE329 lab manual.
  *
  ******************************************************************************
  */

#include "delay.h"

// ------------------------------------------------- delay.c w/o #includes ---
// configure SysTick timer for use with delay_us().
// warning: breaks HAL_delay() by disabling interrupts for shorter delay timing.
void SysTick_Init(void) {
      SysTick->CTRL |= (SysTick_CTRL_ENABLE_Msk |           // enable SysTick
Timer
                        SysTick_CTRL_CLKSOURCE_Msk);    // select CPU clock
      SysTick->CTRL &= ~(SysTick_CTRL_TICKINT_Msk);   // disable interrupt
}

// delay in microseconds using SysTick timer to count CPU clock cycles
// do not call with 0 : error, maximum delay.
// careful calling with small nums : results in longer delays than specified:
//       e.g. @4MHz, delay_us(1) = 10=15 us delay.
void delay_us(const uint32_t time_us) {
      // set the counts for the specified delay
      SysTick->LOAD = (uint32_t)((time_us * (SystemCoreClock / 1000000)) - 1);
      SysTick->VAL = 0;                                          // clear timer
count
      SysTick->CTRL &= ~(SysTick_CTRL_COUNTFLAG_Msk);        // clear count flag
      while (!(SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk)); // wait for flag
}
```

Formatted Source Code lcd.h:

```
/**
  ******************************************************************************
  * @file               : lcd.h
  * project             : EE329 Lab A3
  * author              : Wyatt Tack (wwt) - wtack@calpoly.edu
  * date                : 10/1/2024
  * firmware            : ST-Link V1
  * @attention          : Copyright (c) 2024 STMicroelectronics. All rights
reserved.
  ******************************************************************************
  *
  *   main header for defines for lcd.h
  *
  ******************************************************************************
  */

#ifndef INC_LCD_H_
#define INC_LCD_H_

#include "delay.h"
#include "stm32l4xx_hal.h"

#define LCD_MODER (0x03FFF)
#define LCD_MODER_0 (0x01555)
#define LCD_OTYPER (0x07F)
#define LCD_PUPDR (0x03FFF)
#define LCD_OSPEEDR (0x03FFF)
#define LCD_PORT GPIOA
#define LCD_EN GPIO_PIN_5     //Pin 5
#define LCD_RS GPIO_PIN_4     //Pin 4
#define LCD_DATA_BITS (GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3)
                                        //Pins 11-14
                                        //pull down R/W (set as only outputs)
void LCD_init( void );
void LCD_pulse_ENA( void );
void LCD_4b_command( uint8_t command );
void LCD_command( uint8_t command );
void LCD_write_char( uint8_t letter );
void LCD_set_cursor( uint8_t position[2]);
void LCD_write_string( uint8_t writeData[] );

#endif /* INC_LCD_H_ */
```

Formatted Source Code lcd.c:

```
/**
  ******************************************************************************
  * @file                    : lcd.c
  * project                  : EE329 Lab A3
  * author                   : Wyatt Tack (wwt) - wtack@calpoly.edu
  * date                     : 10/1/2024
  * firmware                 : ST-Link V1
  * @attention               : Copyright (c) 2024 STMicroelectronics. All rights reserved.
  ******************************************************************************
  *
  *            Functions for interfacing and communicating to LCD display through
  *            nibble mode. Provided on behalf of EE329 lab manual. Configured to
  *            be wired through GPIO PORT A Pin0-3 as D4-D7; Pin4 as RS; Pin5 as EN
  *
  ******************************************************************************
  */

#include "lcd.h"

// ------------------------------------------------------ excerpt from lcd.c ---
void LCD_init( void )  {
                //Port clock initialize
                RCC->AHB2ENR   |=  (RCC_AHB2ENR_GPIOAEN);
                //LCD pin initialize - Push Pull, no PU/PD, high speed
                LCD_PORT->MODER   &= ~(LCD_MODER);
                LCD_PORT->MODER   |=  (LCD_MODER_0);
                LCD_PORT->OTYPER  &= ~(LCD_OTYPER);
                LCD_PORT->PUPDR   &= ~(LCD_PUPDR);
                LCD_PORT->OSPEEDR |=  (LCD_OSPEEDR);
              delay_us( 80000 );                        // power-up wait 80 ms
              LCD_PORT->ODR   &= ~(LCD_RS);                          // clear RS bit
              for ( int idx = 0; idx < 3; idx++ ) {  // wake up 1,2,3: DATA = 0011 XXXX
     LCD_4b_command( 0x30 );// HI 4b of 8b cmd, low nibble = X
     delay_us( 5000 );
   }
   LCD_4b_command( 0x20 ); // fcn set #4: 4b cmd set 4b mode - next 0x28:2-line
   delay_us(3000);
   LCD_command( 0x28 );       // fcn set 4b mode and 2x28 line
   delay_us( 300 );
   LCD_command( 0x10 );       //Set cursor
   delay_us( 300 );
   LCD command( 0x0F ); // display, cursor, blink on
   delay_us( 300 );
   LCD command(0x06);         //Entry mode
   delay_us( 300 );
   LCD_command(0x01);         //clear display
   delay_us( 300 );
   LCD_command(0x80);         // set cursor home
   delay_us( 300 );

}

void LCD_pulse_ENA( void )  {
// ENAble line sends command on falling edge
// set to restore default then clear to trigger
   LCD PORT->ODR   |= ( LCD EN );            // ENABLE = HI
   delay us( 25 );                          // TDDR > 320 ns
   LCD_PORT->ODR   &= ~( LCD_EN );        // ENABLE = LOW
   delay_us( 20 );                          // low values flakey, see A3:p.1
}

void LCD_4b_command( uint8_t command )  {
// LCD command using high nibble only - used for 'wake-up' 0x30 commands
   LCD_PORT->ODR   &= ~( LCD_DATA_BITS );    // clear DATA bits
   LCD_PORT->ODR   |= ( command >> 4 );   // DATA = command
   delay_us( 15 );
   LCD pulse ENA( );
}

void LCD_command( uint8_t command )  {
// send command to LCD in 4-bit instruction mode
// HIGH nibble then LOW nibble, timing sensitive
   LCD_PORT->ODR   &= ~( LCD_DATA_BITS );              // isolate cmd bits
   LCD_PORT->ODR   |= ( (command>>4) & LCD_DATA_BITS ); // HIGH shifted low
   delay_us( 15 );
   LCD_pulse_ENA( );                                   // latch HIGH NIBBLE

   LCD PORT->ODR   &= ~( LCD DATA BITS );              // isolate cmd bits
   LCD PORT->ODR   |= ( command & LCD DATA BITS );     // LOW nibble
   delay_us( 15 );
   LCD pulse ENA( );                                   // latch LOW NIBBLE
}

void LCD_write_char( uint8_t letter )  {
// calls LCD_command() w/char data; assumes all ctrl bits set LO in LCD_init()
   LCD_PORT->ODR   |= (LCD_RS);       // RS = HI for data to address
   delay_us( 15 );
   LCD_command( letter );             // character to print
   LCD_PORT->ODR   &= ~(LCD_RS);      // RS = LO
}

void LCD_set_cursor( uint8_t position[2])    {
//              calls LCD_command to change cursor position
                //position formatted as [row,col] (zero indexed)
                //sets ddram address for cursor set
                LCD_PORT->ODR   &= ~(LCD_RS);
                uint8_t ddRamAdd =  (40 * position[1]);
                ddRamAdd |= (0x80);
                LCD_command(ddRamAdd);
                delay_us( 500 );
                for (int col = 0; col < position[0]; col++){
                                LCD_command( 0x14 );
                                delay us( 300 );
                                }
                //set address as RS_Low, data as [0x80 | address]
                //address defined as 0x00-0x0F, 0x40-0x4F
}

void LCD_write_string( uint8_t              writeData[])   {
//              calls LCD_write_char in row long for loop
                for (uint8_t indexCol = 0; indexCol < 16; indexCol++ ){
                                                LCD_write_char(writeData[indexCol]);
                                                delay us(60);
                                }
                }
```