UNIVERSITY OF CAPETOWN

Department of Electrical Engineering

EEE3017W - Digitals

Practical 7 Report

Sean Wood - WDXSEA003 | Sean Woodgate - WDGSEA001

1 Introduction

This report covers the system design and implementation of a simple stopwatch which has lap functionality.

System Design

(a)

Below are the calculations used to find the timer parameters to result in a 0.01s period. The calculations do not follow from the practical instructions.

$$\begin{aligned} f_{cnt} &= \frac{f_{clk}}{\text{Prescaler}} \\ \text{Prescaler} &= 48 \\ \therefore f_{cnt} &= \frac{48000000}{48} = 1000000\text{Hz} \\ \text{ARR} &= T_{tim} \times f_{cnt} \\ &= 0.011000000 \\ &= 10000_{///} \end{aligned} \tag{1}$$

(b)

Below is the initialization function for the ports used in the project:

```c
  /*
   * @brief Initialise the GPIO ports for pushbuttons, LEDs and the ADC
   * @params None
   * @retval None
   */
  static void init_ports(void) {
    // Enable the clock for ports used
    RCC->AHBENR |= RCC_AHBENR_GPIOBEN | RCC_AHBENR_GPIOAEN;

    // Initialise PB0 - PB7, PB10 and PB11 for RG Led
    GPIOB->MODER |= GPIO_MODER_MODER0_0 | GPIO_MODER_MODER1_0 |
                    GPIO_MODER_MODER2_0 | GPIO_MODER_MODER3_0 |
                    GPIO_MODER_MODER4_0 | GPIO_MODER_MODER5_0 |
                    GPIO_MODER_MODER6_0 | GPIO_MODER_MODER7_0 |
                    GPIO_MODER_MODER10_0 | GPIO_MODER_MODER11_0;
    GPIOB->ODR &= ~(GPIO_ODR_10 | GPIO_ODR_11); // Make sure they are not on

    // Initialise PA0, PA1, PA2 and PA3 for SW0, SW1, SW2 and SW3
    GPIOA->MODER &= ~(GPIO_MODER_MODER0 | GPIO_MODER_MODER1 | GPIO_MODER_MODER2
        | GPIO_MODER_MODER3);
    GPIOA->PUPDR |= GPIO_PUPDR_PUPDR0_0 | GPIO_PUPDR_PUPDR1_0
        | GPIO_PUPDR_PUPDR2_0 | GPIO_PUPDR_PUPDR3_0; // Enable pullup resistors

    // Initialise PA5 for ADC1
    GPIOA->MODER |= GPIO_MODER_MODER5;
  }
```

Below is the initialization function for TIM14:

```c
/*
 * @brief Initialise the TIM14
 * @params None
 * @retval None
 */
static void init_TIM14(void) {
  // Enable the clock for TIM14
  RCC->APB1ENR |= RCC_APB1ENR_TIM14EN;

  // Set the frequency to 100Hz
  TIM14->PSC = 48;
  TIM14->ARR = 10000;

  // Enable the interrupt
  TIM14->DIER |= 0x1; // Enable the UIE (Update Interrupt Enable)
  TIM14->CR1 &= ~(1 << 2); // Make sure the interrupt is not disabled in the Control
   ↪  Register 1

  // Make sure the counter is at zero
  TIM14->CNT = 0;
}
```

Below is the initialization function for the Nested Vector Interrupt Controller (NVIC):

```c
/*
 * @brief Initialise the NVIC for pushbutton interrupts
 * @params None
 * @retval None
 */
static void init_NVIC(void) {
  NVIC_EnableIRQ(EXTI0_1_IRQn); // For lines 0 and 1
  NVIC_EnableIRQ(EXTI2_3_IRQn); // For lines 2 and 3
  NVIC_EnableIRQ(TIM14_IRQn); // For TIM14
}
```

(c)

Below is the TIM14 interrupt handler which simply increments a global variable which is keeping the time:

```c
/*
 * @brief Interrupt Request Handler for TIM14
 * @params None
 * @retval None
 */
void TIM14_IRQHandler(void) {
  if (programState != PROG_STATE_STOP) {
    // If we are counting either in LAP mode or in COUNTING mode, increment the time
    timer++;
    if (programState == PROG_STATE_COUNTING) {
      // If we are in COUNTING mode, display the timer on the screen
      display(TIME, timer);
    }
  }

  // Clear the interrupt pending bit
  TIM14->SR &= ~(1 << 0);
}
```

(d)

Below is the function to check for buttons. This function is used in conjunction with interrupts on the EXTI lines:

```c
/*
 * @brief Get the state of the specified switch, with debouncing of predefined length
 * @params pb: Pushbutton number
 * @retval True or false when pressed and not pressed rsp.
 */
static uint8_t check_pb(uint8_t pb) {
  uint8_t pbBit;

  // Check which PB needs to be checked
  switch (pb) {
  case 0:
    pbBit = GPIO_IDR_0;
    break;
  case 1:
    pbBit = GPIO_IDR_1;
    break;
  case 2:
    pbBit = GPIO_IDR_2;
    break;
  case 3:
    pbBit = GPIO_IDR_3;
    break;
  default:
    return FALSE;
  }

  // Debounce and check again - return the result
  if (!(GPIOA->IDR & pbBit)) {
    delay(DEBOUNCE_MS * 1000);
    if (!(GPIOA->IDR & pbBit)) {
      return TRUE;
    } else {
      return FALSE;
    }
  } else {
    return FALSE;
  }
}
```

```c
/*
 * @brief Interrupt Request Handler for EXTI Lines 2 and 3 (PB0 and PB1)
 * @params None
 * @retval None
 */
void EXTI0_1_IRQHandler(void) {
  if (check_pb(0)) {
    if (programState == PROG_STATE_STOP || programState == PROG_STATE_LAP) {
      // Put the program into COUNTING mode and set the appropriate LED
      display(TIME, timer);
      programState = PROG_STATE_COUNTING;
      GPIOB->ODR = (1 << 0);
    }
  } else if (check_pb(1)) {
    if (programState == PROG_STATE_COUNTING) {
      // Update program state to LAP mode
      programState = PROG_STATE_LAP;

      // Capture the lap time, display on the LCD and set the appropriate LED
      lapValue = timer;
      display(TIME, timer);
      GPIOB->ODR = (1 << 1);
    }
  }

  // Clear the interrupt pending bit
  EXTI->PR |= EXTI_PR_PR0 | EXTI_PR_PR1;
}

/*
 * @brief Interrupt Request Handler for EXTI Lines 2 and 3 (PB2 and PB3)
 * @params None
 * @retval None
 */
void EXTI2_3_IRQHandler(void) {
  if (check_pb(2)) {
    if (programState == PROG_STATE_COUNTING) {
      // Put the program into STOP mode and set the appropriate LED
      programState = PROG_STATE_STOP;
      GPIOB->ODR = (1 << 2);
    }
  } else if (check_pb(3)) {
    // Zero the timer, update the program state, display the welcome screen and set the
     ↪  appropriate LED
    timer = 0;
    programState = PROG_STATE_STOP;
    display(WELCOME, 0);
    GPIOB->ODR = (1 << 3);
  }

  // Clear the interrupt pending bit
  EXTI->PR |= EXTI_PR_PR2 | EXTI_PR_PR3;
}
```

(e)

Below is the function to display certain data on the LCD:

```c
/*
 * @brief Display the specified data on the screen
 * @params displayType: What to display on the screen
 *         value: Data to display for the given type
 * @retval None
 */
void display(displayType_t displayType, uint32_t value) {
  // Check for what we need to display
  switch (displayType) {
  case TIME:
    if (programState != PROG_STATE_COUNTING) {
      // Only clear the screen if we know that the first line is going to change
      lcd_command(CLEAR);
      lcd_putstring("Time");
    }

    // Convert the time to the string format and display it on the LCD
    lcd_command(LINE_TWO);
    uint8_t *string = time2String(value);
    lcd_putstring(string);
    free(string); // Make sure we de-allocate the string!
    break;
  case WELCOME:
    // Display the welcome message
    lcd_put2String("Stop Watch", "Press SW0...");
    break;
  default:
    break;
  }
}
```

(f)

Below is the completed code as tested:

```
// ----------------------------------------------------------------------------

// == Includes ==
#include <stdio.h>
#include <stdlib.h>
#include <stm32f0xx.h>
#include "diag/Trace.h"
#include "lcd_stm32f0.h"

#define TRUE           1
#define FALSE          0

#define DEBOUNCE_MS    20

// == Type Definitions ==

// States the program could be in
typedef enum {
  PROG_STATE_INIT,
  PROG_STATE_STOP,
  PROG_STATE_COUNTING,
  PROG_STATE_LAP
} programState_t;

// Types of things to display
typedef enum {
  TIME,
  WELCOME
} displayType_t;

// == Global Variables ==
programState_t programState; // To keep track of the program state throughout execution
uint32_t timer = 0; // ms Timer
uint32_t lapValue = 0; // Variable to store the lap time

// == Function Prototypes ==
static void init_ports(void);
static void init_NVIC(void);
static void init_EXTI(void);
static void init_TIM14(void);

static void lcd_put2String(uint8_t *string1, uint8_t *string2);
void delay(unsigned int microseconds);

static uint8_t check_pb(uint8_t pb);
static void display(displayType_t displayType, uint32_t value);
static uint8_t *time2String(uint32_t time);

// == Program Code ==
int main(int argc, char* argv[]) {
  // Initialisations
  programState = PROG_STATE_INIT;

  init_LCD();
  init_ports();
  init_EXTI();
  init_NVIC();
  init_TIM14();

  programState = PROG_STATE_STOP;

  // Enable the timer
  TIM14->CR1 |= 0x1;

  // Display the welcome message
  display(WELCOME, 0);
  GPIOB->ODR = (1 << 3);

  // Infinite loop
  while (1) {
    __asm("nop");
  }
}

// == Function Definitions ==

/*
 * @brief Initialise the GPIO ports for pushbuttons, LEDs and the ADC
 * @params None
 * @retval None
 */
static void init_ports(void) {
  // Enable the clock for ports used
  RCC->AHBENR |= RCC_AHBENR_GPIOBEN | RCC_AHBENR_GPIOAEN;

  // Initialise PB0 - PB7, PB10 and PB11 for RG Led
  GPIOB->MODER |= GPIO_MODER_MODER0_0 | GPIO_MODER_MODER1_0 |
                  GPIO_MODER_MODER2_0 | GPIO_MODER_MODER3_0 |
                  GPIO_MODER_MODER4_0 | GPIO_MODER_MODER5_0 |
                  GPIO_MODER_MODER6_0 | GPIO_MODER_MODER7_0 |
                  GPIO_MODER_MODER10_0 | GPIO_MODER_MODER11_0;
  GPIOB->ODR &= ~(GPIO_ODR_10 | GPIO_ODR_11); // Make sure they are not on

  // Initialise PA0, PA1, PA2 and PA3 for SW0, SW1, SW2 and SW3
  GPIOA->MODER &= ~(GPIO_MODER_MODER0 | GPIO_MODER_MODER1 | GPIO_MODER_MODER2
      | GPIO_MODER_MODER3);
  GPIOA->PUPDR |= GPIO_PUPDR_PUPDR0_0 | GPIO_PUPDR_PUPDR1_0
      | GPIO_PUPDR_PUPDR2_0 | GPIO_PUPDR_PUPDR3_0; // Enable pullup resistors

  // Initialise PA5 for ADC1
```

```c
  GPIOA->MODER |= GPIO_MODER_MODER5;
}

/*
 * @brief Initialise the TIM14
 * @params None
 * @retval None
 */
static void init_TIM14(void) {
  // Enable the clock for TIM14
  RCC->APB1ENR |= RCC_APB1ENR_TIM14EN;

  // Set the frequency to 100Hz
  TIM14->PSC = 48;
  TIM14->ARR = 10000;

  // Enable the interrupt
  TIM14->DIER |= 0x1; // Enable the UIE (Update Interrupt Enable)
  TIM14->CR1 &= ~(1 << 2); // Make sure the interrupt is not disabled in the Control Register 1

  // Make sure the counter is at zero
  TIM14->CNT = 0;
}

/*
 * @brief Initialise the NVIC for pushbutton interrupts
 * @params None
 * @retval None
 */
static void init_NVIC(void) {
  NVIC_EnableIRQ(EXTI0_1_IRQn); // For lines 0 and 1
  NVIC_EnableIRQ(EXTI2_3_IRQn); // For lines 2 and 3
  NVIC_EnableIRQ(TIM14_IRQn); // For TIM14
}

/*
 * @brief Initialise the EXTI lines for pushbutton interrupts
 * @params None
 * @retval None
 */
static void init_EXTI(void) {
  RCC->APB2ENR |= RCC_APB2ENR_SYSCFGCOMPEN; // Enable the SYSCFG and COMP RCC clock
  SYSCFG->EXTICR[1] &= ~(0xFFFF); // Map PA0 and PA1 to external interrupt lines

  EXTI->FTSR |= EXTI_FTSR_TR0 | EXTI_FTSR_TR1 | EXTI_FTSR_TR2 | EXTI_FTSR_TR3; // Configure trigger to falling edge
  EXTI->IMR |= EXTI_IMR_MR0 | EXTI_IMR_MR1 | EXTI_IMR_MR2 | EXTI_IMR_MR3; // Umask the interrupts
}

/*
 * @brief Rational addition of a safe 2 line write to the LCD
 * @params *string1: Pointer to the string to be written to line 1
 *         *string2: Pointer to the string to be written to line 2
 * @retval None
 */
static void lcd_put2String(uint8_t *string1, uint8_t *string2) {
  // Clear the LCD
  lcd_command(CLEAR);

  // Write the strings to the LCD
  lcd_putstring(string1);
  lcd_command(LINE_TWO);
  lcd_putstring(string2);
}

/*
 * @brief Get the state of the specified switch, with debouncing of predefined length
 * @params pb: Pushbutton number
 * @retval True or false when pressed and not pressed rsp.
 */
static uint8_t check_pb(uint8_t pb) {
  uint8_t pbBit;

  // Check which PB needs to be checked
  switch (pb) {
  case 0:
    pbBit = GPIO_IDR_0;
    break;
  case 1:
    pbBit = GPIO_IDR_1;
    break;
  case 2:
    pbBit = GPIO_IDR_2;
    break;
  case 3:
    pbBit = GPIO_IDR_3;
    break;
  default:
    return FALSE;
  }

  // Debounce and check again - return the result
  if (!(GPIOA->IDR & pbBit)) {
    delay(DEBOUNCE_MS * 1000);
    if (!(GPIOA->IDR & pbBit)) {
      return TRUE;
    } else {
      return FALSE;
    }
  } else {
    return FALSE;
  }
}

/*
 * @brief Interrupt Request Handler for TIM14
 * @params None
 * @retval None
 */
void TIM14_IRQHandler(void) {
  if (programState != PROG_STATE_STOP) {
    // If we are counting either in LAP mode or in COUNTING mode, increment the time
    timer++;
    if (programState == PROG_STATE_COUNTING) {
      // If we are in COUNTING mode, display the timer on the screen
      display(TIME, timer);
    }
  }

  // Clear the interrupt pending bit
  TIM14->SR &= ~(1 << 0);
}

/*
 * @brief Interrupt Request Handler for EXTI Lines 2 and 3 (PB0 and PB1)
 * @params None
 * @retval None
 */
void EXTI0_1_IRQHandler(void) {
  if (check_pb(0)) {
    if (programState == PROG_STATE_STOP || programState == PROG_STATE_LAP) {
      // Put the program into COUNTING mode and set the appropriate LED
      display(TIME, timer);
      programState = PROG_STATE_COUNTING;
      GPIOB->ODR = (1 << 0);
    }
  } else if (check_pb(1)) {
    if (programState == PROG_STATE_COUNTING) {
      // Update program state to LAP mode
      programState = PROG_STATE_LAP;

      // Capture the lap time, display on the LCD and set the appropriate LED
      lapValue = timer;
      display(TIME, timer);
      GPIOB->ODR = (1 << 1);
    }
  }

  // Clear the interrupt pending bit
  EXTI->PR |= EXTI_PR_PR0 | EXTI_PR_PR1;
}
```

```

/*
 * @brief Interrupt Request Handler for EXTI Lines 2 and 3 (PB2 and PB3)
 * @params None
 * @retval None
 */
void EXTI2_3_IRQHandler(void) {
  if (check_pb(2)) {
    if (programState == PROG_STATE_COUNTING) {
      // Put the program into STOP mode and set the appropriate LED
      programState = PROG_STATE_STOP;
      GPIOB->ODR = (1 << 2);
    }
  } else if (check_pb(3)) {
    // Zero the timer, update the program state, display the welcome screen and set the
     ↪  appropriate LED
    timer = 0;
    programState = PROG_STATE_STOP;
    display(WELCOME, 0);
    GPIOB->ODR = (1 << 3);
  }

  // Clear the interrupt pending bit
  EXTI->PR |= EXTI_PR_PR2 | EXTI_PR_PR3;
}


/*
 * @brief Display the specified data on the screen
 * @params displayType: What to display on the screen
 *         value: Data to display for the given type
 * @retval None
 */
void display(displayType_t displayType, uint32_t value) {
  // Check for what we need to display
  switch (displayType) {
  case TIME:
    if (programState != PROG_STATE_COUNTING) {
      // Only clear the screen if we know that the first line is going to change
      lcd_command(CLEAR);
      lcd_putstring("Time");
    }

    // Convert the time to the string format and display it on the LCD
    lcd_command(LINE_TWO);
    uint8_t *string = time2String(value);
    lcd_putstring(string);
    free(string); // Make sure we de-allocate the string!
    break;
  case WELCOME:
    // Display the welcome message
    lcd_put2String("Stop Watch", "Press SW0...");
    break;
  default:
    break;
  }
}

/*
 * @brief Convert the time from ms into a displayable string
 * @params time: The time in ms
 * @retval Pointer to a string
 * @Note: The string must be deallocated after use
 */
static uint8_t *time2String(uint32_t time) {
  uint32_t timeVal = time;
  uint8_t *string;
  uint8_t strLength = 9*sizeof(uint8_t); // Calculate the string length
  string = malloc(strLength); // Allocate the correct amount of memory for the string

  // Extract the minutes, seconds and milliseconds
  uint8_t minutes = timeVal/6000;
  timeVal -= minutes*6000;

  uint8_t seconds = timeVal/100;
  timeVal -= seconds*100;

  uint8_t ms = timeVal;

  // Format the output string
  sprintf(string, "%02d:%02d.%02d\0", minutes, seconds, ms);

  // Return a pointer to the string
  return string;
}

// ----------------------------------------------------------------------------
```