



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:

```

1  /*
2   * @brief Initialise the GPIO ports for pushbuttons, LEDs and the ADC
3   * @params None
4   * @retval None
5   */
6  static void init_ports(void) {
7      // Enable the clock for ports used
8      RCC->AHBENR |= RCC_AHBENR_GPIOBEN | RCC_AHBENR_GPIOAEN;
9
10     // Initialise PB0 - PB7, PB10 and PB11 for RG Led
11     GPIOB->MODER |= GPIO_MODER_MODER0_0 | GPIO_MODER_MODER1_0 |
12                   GPIO_MODER_MODER2_0 | GPIO_MODER_MODER3_0 |
13                   GPIO_MODER_MODER4_0 | GPIO_MODER_MODER5_0 |
14                   GPIO_MODER_MODER6_0 | GPIO_MODER_MODER7_0 |
15                   GPIO_MODER_MODER10_0 | GPIO_MODER_MODER11_0;
16     GPIOB->ODR &= ~(GPIO_ODR_10 | GPIO_ODR_11); // Make sure they are not on
17
18     // Initialise PA0, PA1, PA2 and PA3 for Sw0, Sw1, Sw2 and Sw3
19     GPIOA->MODER &= ~(GPIO_MODER_MODER0 | GPIO_MODER_MODER1 | GPIO_MODER_MODER2
20                     | GPIO_MODER_MODER3);
21     GPIOA->PUPDR |= GPIO_PUPDR_PUPDR0_0 | GPIO_PUPDR_PUPDR1_0
22                   | GPIO_PUPDR_PUPDR2_0 | GPIO_PUPDR_PUPDR3_0; // Enable pullup resistors
23
24     // Initialise PA5 for ADC1
25     GPIOA->MODER |= GPIO_MODER_MODER5;
26 }

```

Below is the initialization function for TIM14:

```

1  /*
2   * @brief Initialise the TIM14
3   * @params None
4   * @retval None
5   */
6  static void init_TIM14(void) {
7      // Enable the clock for TIM14
8      RCC->APB1ENR |= RCC_APB1ENR_TIM14EN;
9
10     // Set the frequency to 100Hz
11     TIM14->PSC = 48;
12     TIM14->ARR = 10000;
13
14     // Enable the interrupt
15     TIM14->DIER |= 0x1; // Enable the UIE (Update Interrupt Enable)
16     TIM14->CR1 &= ~(1 << 2); // Make sure the interrupt is not disabled in the Control
17                               ↪ Register 1
18
19     // Make sure the counter is at zero
20     TIM14->CNT = 0;
21 }

```

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

```
1  /*
2   * @brief Initialise the NVIC for pushbutton interrupts
3   * @params None
4   * @retval None
5   */
6  static void init_NVIC(void) {
7      NVIC_EnableIRQ(EXTI0_1_IRQn); // For lines 0 and 1
8      NVIC_EnableIRQ(EXTI2_3_IRQn); // For lines 2 and 3
9      NVIC_EnableIRQ(TIM14_IRQn); // For TIM14
10 }
```

(c)

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

```
1  /*
2   * @brief Interrupt Request Handler for TIM14
3   * @params None
4   * @retval None
5   */
6  void TIM14_IRQHandler(void) {
7      if (programState != PROG_STATE_STOP) {
8          // If we are counting either in LAP mode or in COUNTING mode, increment the time
9          timer++;
10         if (programState == PROG_STATE_COUNTING) {
11             // If we are in COUNTING mode, display the timer on the screen
12             display(TIME, timer);
13         }
14     }
15
16     // Clear the interrupt pending bit
17     TIM14->SR &= ~(1 << 0);
18 }
```

(d)

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

```
1  /*
2   * @brief Get the state of the specified switch, with debouncing of predefined length
3   * @params pb: Pushbutton number
4   * @retval True or false when pressed and not pressed resp.
5   */
6  static uint8_t check_pb(uint8_t pb) {
7      uint8_t pbBit;
8
9      // Check which PB needs to be checked
10     switch (pb) {
11     case 0:
12         pbBit = GPIO_IDR_0;
13         break;
14     case 1:
15         pbBit = GPIO_IDR_1;
16         break;
17     case 2:
18         pbBit = GPIO_IDR_2;
19         break;
20     case 3:
21         pbBit = GPIO_IDR_3;
22         break;
23     default:
24         return FALSE;
25     }
26
27     // Debounce and check again - return the result
28     if (!(GPIOA->IDR & pbBit)) {
29         delay(DEBOUNCE_MS * 1000);
30         if (!(GPIOA->IDR & pbBit)) {
31             return TRUE;
32         } else {
33             return FALSE;
34         }
35     } else {
36         return FALSE;
37     }
38 }
```

```

1  /*
2   * @brief Interrupt Request Handler for EXTI Lines 2 and 3 (PB0 and PB1)
3   * @params None
4   * @retval None
5   */
6  void EXTI0_1_IRQHandler(void) {
7      if (check_pb(0)) {
8          if (programState == PROG_STATE_STOP || programState == PROG_STATE_LAP) {
9              // Put the program into COUNTING mode and set the appropriate LED
10             display(TIME, timer);
11             programState = PROG_STATE_COUNTING;
12             GPIOB->ODR = (1 << 0);
13         }
14     } else if (check_pb(1)) {
15         if (programState == PROG_STATE_COUNTING) {
16             // Update program state to LAP mode
17             programState = PROG_STATE_LAP;
18
19             // Capture the lap time, display on the LCD and set the appropriate LED
20             lapValue = timer;
21             display(TIME, timer);
22             GPIOB->ODR = (1 << 1);
23         }
24     }
25
26     // Clear the interrupt pending bit
27     EXTI->PR |= EXTI_PR_PR0 | EXTI_PR_PR1;
28 }
29
30 /*
31 * @brief Interrupt Request Handler for EXTI Lines 2 and 3 (PB2 and PB3)
32 * @params None
33 * @retval None
34 */
35 void EXTI2_3_IRQHandler(void) {
36     if (check_pb(2)) {
37         if (programState == PROG_STATE_COUNTING) {
38             // Put the program into STOP mode and set the appropriate LED
39             programState = PROG_STATE_STOP;
40             GPIOB->ODR = (1 << 2);
41         }
42     } else if (check_pb(3)) {
43         // Zero the timer, update the program state, display the welcome screen and set the
44         // appropriate LED
45         timer = 0;
46         programState = PROG_STATE_STOP;
47         display(WELCOME, 0);
48         GPIOB->ODR = (1 << 3);
49     }
50
51     // Clear the interrupt pending bit
52     EXTI->PR |= EXTI_PR_PR2 | EXTI_PR_PR3;
53 }

```

(e)

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

```
1  /*
2   * @brief Display the specified data on the screen
3   * @params displayType: What to display on the screen
4   *         value: Data to display for the given type
5   * @retval None
6   */
7  void display(displayType_t displayType, uint32_t value) {
8      // Check for what we need to display
9      switch (displayType) {
10         case TIME:
11             if (programState != PROG_STATE_COUNTING) {
12                 // Only clear the screen if we know that the first line is going to change
13                 lcd_command(CLEAR);
14                 lcd_putstring("Time");
15             }
16
17             // Convert the time to the string format and display it on the LCD
18             lcd_command(LINE_TWO);
19             uint8_t *string = time2String(value);
20             lcd_putstring(string);
21             free(string); // Make sure we de-allocate the string!
22             break;
23         case WELCOME:
24             // Display the welcome message
25             lcd_put2String("Stop Watch", "Press SW0...");
26             break;
27         default:
28             break;
29     }
30 }
```



(f)

Below is the completed code as tested:

```

1 // -----
2
3 // == Includes ==
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <stm32f0xx.h>
7 #include "diag/Trace.h"
8 #include "lcd_stm32f0.h"
9
10 #define TRUE          1
11 #define FALSE         0
12
13 #define DEBOUNCE_MS   20
14
15 // == Type Definitions ==
16
17 // States the program could be in
18 typedef enum {
19     PROG_STATE_INIT,
20     PROG_STATE_STOP,
21     PROG_STATE_COUNTING,
22     PROG_STATE_LAP
23 } programState_t;
24
25 // Types of things to display
26 typedef enum {
27     TIME,
28     WELCOME
29 } displayType_t;
30
31 // == Global Variables ==
32 programState_t programState; // To keep track of the program state throughout execution
33 uint32_t timer = 0; // ms Timer
34 uint32_t lapValue = 0; // Variable to store the lap time
35
36 // == Function Prototypes ==
37 static void init_ports(void);
38 static void init_NVIC(void);
39 static void init_EXTI(void);
40 static void init_TIM14(void);
41
42 static void lcd_put2String(uint8_t *string1, uint8_t *string2);
43 void delay(unsigned int microseconds);
44
45 static uint8_t check_pb(uint8_t pb);
46 static void display(displayType_t displayType, uint32_t value);
47 static uint8_t *time2String(uint32_t time);
48
49 // == Program Code ==
50 int main(int argc, char* argv[]) {

```

```

51 // Initialisations
52 programState = PROG_STATE_INIT;
53
54 init_LCD();
55 init_ports();
56 init_EXTI();
57 init_NVIC();
58 init_TIM14();
59
60 programState = PROG_STATE_STOP;
61
62 // Enable the timer
63 TIM14->CR1 |= 0x1;
64
65 // Display the welcome message
66 display(WELCOME, 0);
67 GPIOB->ODR = (1 << 3);
68
69 // Infinite loop
70 while (1) {
71     __asm("nop");
72 }
73 }
74
75 // == Function Definitions ==
76
77 /*
78  * @brief Initialise the GPIO ports for pushbuttons, LEDs and the ADC
79  * @params None
80  * @retval None
81  */
82 static void init_ports(void) {
83     // Enable the clock for ports used
84     RCC->AHBENR |= RCC_AHBENR_GPIOBEN | RCC_AHBENR_GPIOAEN;
85
86     // Initialise PB0 - PB7, PB10 and PB11 for RG Led
87     GPIOB->MODER |= GPIO_MODER_MODER0_0 | GPIO_MODER_MODER1_0 |
88                   GPIO_MODER_MODER2_0 | GPIO_MODER_MODER3_0 |
89                   GPIO_MODER_MODER4_0 | GPIO_MODER_MODER5_0 |
90                   GPIO_MODER_MODER6_0 | GPIO_MODER_MODER7_0 |
91                   GPIO_MODER_MODER10_0 | GPIO_MODER_MODER11_0;
92     GPIOB->ODR &= ~(GPIO_ODR_10 | GPIO_ODR_11); // Make sure they are not on
93
94     // Initialise PA0, PA1, PA2 and PA3 for SW0, SW1, SW2 and SW3
95     GPIOA->MODER &= ~(GPIO_MODER_MODER0 | GPIO_MODER_MODER1 | GPIO_MODER_MODER2
96                     | GPIO_MODER_MODER3);
97     GPIOA->PUPDR |= GPIO_PUPDR_PUPDR0_0 | GPIO_PUPDR_PUPDR1_0
98                   | GPIO_PUPDR_PUPDR2_0 | GPIO_PUPDR_PUPDR3_0; // Enable pullup resistors
99
100    // Initialise PA5 for ADC1

```

```

101  GPIOA->MODER |= GPIO_MODER_MODER5;
102  }
103
104  /*
105   * @brief Initialise the TIM14
106   * @params None
107   * @retval None
108   */
109  static void init_TIM14(void) {
110      // Enable the clock for TIM14
111      RCC->APB1ENR |= RCC_APB1ENR_TIM14EN;
112
113      // Set the frequency to 100Hz
114      TIM14->PSC = 48;
115      TIM14->ARR = 10000;
116
117      // Enable the interrupt
118      TIM14->DIER |= 0x1; // Enable the UIE (Update Interrupt Enable)
119      TIM14->CR1 &= ~(1 << 2); // Make sure the interrupt is not disabled in the Control Register
120      ↪ 1
121
122      // Make sure the counter is at zero
123      TIM14->CNT = 0;
124  }
125
126  /*
127   * @brief Initialise the NVIC for pushbutton interrupts
128   * @params None
129   * @retval None
130   */
131  static void init_NVIC(void) {
132      NVIC_EnableIRQ(EXTI0_1_IRQn); // For lines 0 and 1
133      NVIC_EnableIRQ(EXTI2_3_IRQn); // For lines 2 and 3
134      NVIC_EnableIRQ(TIM14_IRQn); // For TIM14
135  }
136
137  /*
138   * @brief Initialise the EXTI lines for pushbutton interrupts
139   * @params None
140   * @retval None
141   */
142  static void init_EXTI(void) {
143      RCC->APB2ENR |= RCC_APB2ENR_SYSCFGCOMPEN; // Enable the SYSCFG and COMP RCC clock
144      SYSCFG->EXTICR[1] &= ~(0xFFFF); // Map PA0 and PA1 to external interrupt lines
145
146      EXTI->FTSR |= EXTI_FTSR_TR0 | EXTI_FTSR_TR1 | EXTI_FTSR_TR2 | EXTI_FTSR_TR3; // Configure
147      ↪ trigger to falling edge
148      EXTI->IMR |= EXTI_IMR_MR0 | EXTI_IMR_MR1 | EXTI_IMR_MR2 | EXTI_IMR_MR3; // Unmask the
149      ↪ interrupts
150  }
151
152  /*
153   * @brief Rational addition of a safe 2 line write to the LCD

```

```
151  * @params *string1: Pointer to the string to be written to line 1
152  *          *string2: Pointer to the string to be written to line 2
153  * @retval None
154  */
155  static void lcd_put2String(uint8_t *string1, uint8_t *string2) {
156      // Clear the LCD
157      lcd_command(CLEAR);
158
159      // Write the strings to the LCD
160      lcd_putstr(string1);
161      lcd_command(LINE_TWO);
162      lcd_putstr(string2);
163  }
164
165  /*
166   * @brief Get the state of the specified switch, with debouncing of predefined length
167   * @params pb: Pushbutton number
168   * @retval True or false when pressed and not pressed resp.
169   */
170  static uint8_t check_pb(uint8_t pb) {
171      uint8_t pbBit;
172
173      // Check which PB needs to be checked
174      switch (pb) {
175          case 0:
176              pbBit = GPIO_IDR_0;
177              break;
178          case 1:
179              pbBit = GPIO_IDR_1;
180              break;
181          case 2:
182              pbBit = GPIO_IDR_2;
183              break;
184          case 3:
185              pbBit = GPIO_IDR_3;
186              break;
187          default:
188              return FALSE;
189      }
190
191      // Debounce and check again - return the result
192      if (!(GPIOA->IDR & pbBit)) {
193          delay(DEBOUNCE_MS * 1000);
194          if (!(GPIOA->IDR & pbBit)) {
195              return TRUE;
196          } else {
197              return FALSE;
198          }
199      } else {
200          return FALSE;
```

```
201     }
202 }
203
204 /*
205  * @brief Interrupt Request Handler for TIM14
206  * @params None
207  * @retval None
208  */
209 void TIM14_IRQHandler(void) {
210     if (programState != PROG_STATE_STOP) {
211         // If we are counting either in LAP mode or in COUNTING mode, increment the time
212         timer++;
213         if (programState == PROG_STATE_COUNTING) {
214             // If we are in COUNTING mode, display the timer on the screen
215             display(TIME, timer);
216         }
217     }
218
219     // Clear the interrupt pending bit
220     TIM14->SR &= ~(1 << 0);
221 }
222
223 /*
224  * @brief Interrupt Request Handler for EXTI Lines 2 and 3 (PB0 and PB1)
225  * @params None
226  * @retval None
227  */
228 void EXTI0_1_IRQHandler(void) {
229     if (check_pb(0)) {
230         if (programState == PROG_STATE_STOP || programState == PROG_STATE_LAP) {
231             // Put the program into COUNTING mode and set the appropriate LED
232             display(TIME, timer);
233             programState = PROG_STATE_COUNTING;
234             GPIOB->ODR = (1 << 0);
235         }
236     } else if (check_pb(1)) {
237         if (programState == PROG_STATE_COUNTING) {
238             // Update program state to LAP mode
239             programState = PROG_STATE_LAP;
240
241             // Capture the lap time, display on the LCD and set the appropriate LED
242             lapValue = timer;
243             display(TIME, timer);
244             GPIOB->ODR = (1 << 1);
245         }
246     }
247
248     // Clear the interrupt pending bit
249     EXTI->PR |= EXTI_PR_PR0 | EXTI_PR_PR1;
250 }
```

```
251
252 /*
253  * @brief Interrupt Request Handler for EXTI Lines 2 and 3 (PB2 and PB3)
254  * @params None
255  * @retval None
256  */
257 void EXTI2_3_IRQHandler(void) {
258     if (check_pb(2)) {
259         if (programState == PROG_STATE_COUNTING) {
260             // Put the program into STOP mode and set the appropriate LED
261             programState = PROG_STATE_STOP;
262             GPIOB->ODR = (1 << 2);
263         }
264     } else if (check_pb(3)) {
265         // Zero the timer, update the program state, display the welcome screen and set the
266         // appropriate LED
267         timer = 0;
268         programState = PROG_STATE_STOP;
269         display(WELCOME, 0);
270         GPIOB->ODR = (1 << 3);
271     }
272     // Clear the interrupt pending bit
273     EXTI->PR |= EXTI_PR_PR2 | EXTI_PR_PR3;
274 }
275
276
277 /*
278  * @brief Display the specified data on the screen
279  * @params displayType: What to display on the screen
280  *         value: Data to display for the given type
281  * @retval None
282  */
283 void display(displayType_t displayType, uint32_t value) {
284     // Check for what we need to display
285     switch (displayType) {
286     case TIME:
287         if (programState != PROG_STATE_COUNTING) {
288             // Only clear the screen if we know that the first line is going to change
289             lcd_command(CLEAR);
290             lcd_putstr("Time");
291         }
292
293         // Convert the time to the string format and display it on the LCD
294         lcd_command(LINE_TWO);
295         uint8_t *string = time2String(value);
296         lcd_putstr(string);
297         free(string); // Make sure we de-allocate the string!
298         break;
299     case WELCOME:
300         // Display the welcome message
```

```
301     lcd_put2String("Stop Watch", "Press SW0...");
302     break;
303 default:
304     break;
305 }
306 }
307
308 /*
309  * @brief Convert the time from ms into a displayable string
310  * @params time: The time in ms
311  * @retval Pointer to a string
312  * @Note: The string must be deallocated after use
313  */
314 static uint8_t *time2String(uint32_t time) {
315     uint32_t timeVal = time;
316     uint8_t *string;
317     uint8_t strLength = 9*sizeof(uint8_t); // Calculate the string length
318     string = malloc(strLength); // Allocate the correct amount of memory for the string
319
320     // Extract the minutes, seconds and milliseconds
321     uint8_t minutes = timeVal/6000;
322     timeVal -= minutes*6000;
323
324     uint8_t seconds = timeVal/100;
325     timeVal -= seconds*100;
326
327     uint8_t ms = timeVal;
328
329     // Format the output string
330     sprintf(string, "%02d:%02d.%02d\0", minutes, seconds, ms);
331
332     // Return a pointer to the string
333     return string;
334 }
335
336 // -----
```