



UNIVERSITY OF CAPETOWN

Department of Electrical Engineering

EEE3017W - Digitals

Practical 6 Report

Sean Wood - WDXSEA003 | Sean Woodgate - WDGSEA001

1 Introduction

This report covers the system design and implementation of a rain-gauge on the STM32F051C6 Development Board as part of the EEE3017W Digitals Practical 6.

System Design

(a)

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 the Analog to Digital Converter (ADC):

```

1  /*
2   * @brief Initialise the ADC to POT0
3   * @params None
4   * @retval None
5   */
6  static void init_ADC(void) {
7      // Enable the ADC clock in the RCC
8      RCC->APB2ENR |= RCC_APB2ENR_ADCEN;
9
10     // Select ADC channel 5 for POT0
11     ADC1->CHSELR |= ADC_CHSELR_CHSEL5;
12
13     // Enable the ADC peripheral
14     ADC1->CR |= ADC_CR_ADEN;
15
16     // Wait for the ADC to become ready
17     while (!(ADC1->ISR & ADC_ISR_ADRDY)) {
18         __asm("nop");
19     }
20 }

```

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 }

```

Below is the initialization function for the External Interrupt controller (EXTI):

```

1  /*
2   * @brief Initialise the EXTI lines for pushbutton interrupts
3   * @params None
4   * @retval None
5   */
6  static void init_EXTI(void) {
7      RCC->APB2ENR |= RCC_APB2ENR_SYSCFGCOMPEN; // Enable the SYSCFG and COMP RCC clock
8      SYSCFG->EXTICR[1] &= ~(0xFFFF); // Map PA0 and PA1 to external interrupt lines
9
10     EXTI->FTSR |= EXTI_FTSR_TR0 | EXTI_FTSR_TR1 | EXTI_FTSR_TR2 | EXTI_FTSR_TR3; // Configure
11     ↪ trigger to falling edge
12     EXTI->IMR |= EXTI_IMR_MR0 | EXTI_IMR_MR1 | EXTI_IMR_MR2 | EXTI_IMR_MR3; // Unmask the
13     ↪ interrupts
14 }

```

(b)

The rain-gauge requires a low voltage detection system to detect when the battery falls below a certain threshold. Given that the maximum battery voltage is 24V and the threshold is set to 14V, the digital ADC value at the threshold will be the following:

$$\begin{aligned} \text{ADC Analog Value} \Big|_{\text{at threshold}} &= \frac{V_{ADC_{max}}}{V_{bat_{max}}} \times V_{thres} \\ &= \frac{3}{24}(14) \\ &= 1.75V \end{aligned}$$

$$\begin{aligned} \text{ADC} \Big|_{1.75 \text{ V}} &= \frac{\text{12-bit maximim}}{V_{ADC_{max}}} \times \text{ADC Analog Value} \\ &= \frac{4095}{3}(1.75) \\ &= 2399_{///} \end{aligned}$$

(c)

Below are the functions to monitor the battery voltage:

```
1  /*
2  * @brief Kick off and grab an ADC conversion
3  * @params None
4  * @retval None
5  */
6  static uint16_t getADC(void) {
7      // Start a conversion
8      ADC1->CR |= ADC_CR_ADSTART;
9
10     // Wait for the conversion to finish
11     while (!(ADC1->ISR & ADC_ISR_EOC)) {
12         __asm("nop");
13     }
14
15     // Return the result of the conversion
16     return (uint16_t)(ADC1->DR);
17 }
18
19 /*
20 * @brief Check the "battery voltage" and display it
21 * @params None
22 * @retval None
23 */
24 static void check_battery(void) {
25     // Grab the ADC value, convert to uV and then to battery voltage
26     uint16_t adcVal = getADC();
27     uint32_t uVoltage = adcVal * ADC_GRAIN;
28     batVoltage = 7.21*(uVoltage/ADC_MULTIPLIER);
29
30     // Check for voltage threshold and change the LED accordingly
31     if (batVoltage <= BAT_THRESHOLD) {
32         GPIOB->ODR &= ~(1 << 11);
33         GPIOB->ODR |= (1 << 10);
34     } else {
35         GPIOB->ODR &= ~(1 << 10);
36         GPIOB->ODR |= (1 << 11);
37     }
38 }
```

(d)

Below is the EXTI handler for lines 0 and 1:

```

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      // Check which button generated the interrupt
8      if (getSW(0)) {
9          // Check the state of the program
10         switch (programState) {
11             case PROG_STATE_WAIT_FOR_SW0:
12                 // If we were waiting for SW0, display the menu
13                 display(DISPLAY_MENU, 0);
14
15                 // Change program state
16                 programState = PROG_STATE_WAIT_FOR_BUTTON;
17                 break;
18             default:
19                 break;
20         }
21     } else if (getSW(1)) {
22         // Check the state of the program
23         switch (programState) {
24             case PROG_STATE_WAIT_FOR_BUTTON:
25                 // If we were waiting for another button:
26                 rainCounter++; // Increment the rain counter
27                 display(DISPLAY_RAIN_BUCKET, 0); // Notify the user
28                 break;
29             default:
30                 break;
31         }
32     }
33
34     // Clear the interrupt pending bit
35     EXTI->PR |= EXTI_PR_PR0 | EXTI_PR_PR1;
36 }

```

(e)

Below is the function to convert floats to BCD:

```

1  /*
2  * @brief Convert the float given to a string
3  * @params rain: Rain in mm
4  *         dec: Number of digits to the left of the decimal point
5  *         frac: Number of decimal places (precision)
6  * @retval Pointer to the converted string
7  * @note String must be freed after use
8  */
9  static uint8_t *ConverttoBCD(float number, uint8_t dec, uint8_t frac) {
10     uint8_t *string; // Pointer to the resulting string
11     uint32_t rainDec = number*pow(10,frac); // Shift all digits to be used onto the left side
12     ↪ of the decimal point
13     uint32_t strLength = (dec + frac + 2)*sizeof(uint8_t); // Calculate the length of the
14     ↪ require string given the accuracy parameters
15     string = malloc(strLength); // Allocate space for the resulting string
16     memset(string, '0', strLength); // Set all characters in the string to zeroes
17
18     // Loop through the digits in the newly formed integer number and place the digits in the
19     ↪ string
20     int pos = 0;
21     int dig = 0;
22     for (pos = 0; pos < strLength; pos++) {
23         // If we reach the end of the decimal part of the number, skip a position for placement
24         ↪ of the decimal point
25         if (pos == dec) {
26             pos++;
27         }
28
29         // Extract the digit from the newly formed integer number based on the position
30         uint32_t multiplier = pow(10, strLength-dig-3);
31         uint32_t digit = (uint32_t)(rainDec/multiplier);
32         string[pos] = (uint8_t)(digit + 48); // Convert the number to ASCII by adding 48 to it
33         rainDec -= digit*multiplier; // Subtract the extracted digit from the integer number
34
35         // Increment the digit number
36         dig++;
37     }
38
39     // Place the decimal point and the null terminator in the correct positions
40     string[dec] = '.';
41     string[strLength - 1] = '\0';
42
43     // Return the pointer to the converted string
44     return string;
45 }

```

(e)

Below is the function to display values on the LCD:

```
1  /*
2   * @brief Display the specified data on the screen
3   * @params displayType: What to display on the screen
4   *         ...: Data to display for the given type
5   * @retval None
6   */
7  void display(displayType_t displayType, float data) {
8      // Switch on what needs to be displayed
9      switch (displayType) {
10         case DISP_BAT: {
11             // Display the battery voltage on the LCD
12             lcd_command(CLEAR);
13             lcd_command(CURSOR_HOME);
14             lcd_putstr("Battery:");
15             lcd_command(LINE_TWO);
16
17             // Generate the string with the batter voltage
18             uint8_t *string = ConverttoBCD(data, 2, 3);
19             lcd_putstr(string);
20             lcd_putstr(" V");
21
22             // De-allocate the memory used for the battery string
23             free(string);
24             break;
25         }
26         case DISP_RAINFALL: {
27             // Display the rainfall amount on the LCD
28             lcd_command(CLEAR);
29             lcd_command(CURSOR_HOME);
30             lcd_putstr("Rainfall:");
31             lcd_command(LINE_TWO);
32
33             // Fetch and convert the rainfall to a string
34             float rain = 0.2*data;
35             uint8_t *string = ConverttoBCD(rain, 4, 1);
36             lcd_putstr(string);
37             lcd_putstr(" mm");
38
39             // De-allocate the memory used for the rainfall string
40             free(string);
41             break;
42         }
43         case DISP_RAIN_BUCKET:
44             // Display the bucket tip notification LCD
45             lcd_put2String("Rain bucket tip", "");
46             break;
47         case DISP_MENU:
48             // Display the menu on the LCD
49             lcd_put2String("Weather Station", "Press SW2 or SW3");
50             break;
51         case DISP_WELCOME:
52             // Display the welcome on the LCD
53             lcd_put2String("EEE3017W Prac 6", "Sean & Sean");
54             break;
55         default:
56             break;
57     }
58 }
```


(f)

Below is the completed code as tested:

```

1 // -----
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <stdarg.h>
6 #include <stm32f0xx.h>
7 #include "math.h"
8 #include "diag/Trace.h"
9 #include "Lcd_stm32f0.h"
10
11 #define TRUE          1
12 #define FALSE         0
13
14 #define DEBOUNCE_MS    20
15 #define ADC_GRAIN      806 // ADC uV per bits
16 #define ADC_MULTIPLIER (float)1000000 // Grain multiplier
17 #define BAT_THRESHOLD (float)14.0 // Low battery threshold
18
19 // == Type Definitions
20 typedef enum {
21     PROG_STATE_INIT,
22     PROG_STATE_WAIT_FOR_SW0,
23     PROG_STATE_WAIT_FOR_BUTTON,
24     PROG_STATE_BUCKET_TIP
25 } programState_t;
26
27 typedef enum {
28     DISP_RAIN_BUCKET,
29     DISP_RAINFALL,
30     DISP_BAT,
31     DISP_WELCOME,
32     DISP_MENU
33 } displayType_t;
34
35 // == Global Variables
36 programState_t programState; // To keep track of the program state throughout execution
37 uint32_t rainCounter; // Keep track of the rain [0.2mm]
38 float batVoltage; // Battery voltage from 1Hz samples
39
40 // == Function Prototypes
41 static void init_ports(void);
42 static void init_ADC(void);
43 static void init_NVIC(void);
44 static void init_EXTI(void);
45 static void init_TIM14(void);
46
47 static void lcd_put2String(uint8_t *string1, uint8_t *string2);
48 void delay(unsigned int microseconds);
49
50 static uint8_t getSw(uint8_t pb);

```

```

51 static void check_battery(void);
52 static uint16_t getADC(void);
53 static void display(displayType_t displayType, float data);
54 static uint8_t *ConverttoBCD(float number, uint8_t dec, uint8_t frac);
55
56 // == Program Code
57 int main(int argc, char* argv[]) {
58     // Initialisations
59     programState = PROG_STATE_INIT;
60
61     init_LCD();
62     init_ports();
63     init_EXTI();
64     init_NVIC();
65     init_ADC();
66     init_TIM14();
67
68     display(DISP_WELCOME, 0);
69     programState = PROG_STATE_WAIT_FOR_SW0;
70
71     // Infinite Loop
72     while (1) {
73         __asm("nop");
74     }
75 }
76
77 // == Function Definitions
78
79 /*
80  * @brief Initialise the GPIO ports for pushbuttons, LEDs and the ADC
81  * @params None
82  * @retval None
83  */
84 static void init_ports(void) {
85     // Enable the clock for ports used
86     RCC->AHBENR |= RCC_AHBENR_GPIOBEN | RCC_AHBENR_GPIOAEN;
87
88     // Initialise PB0 - PB7, PB10 and PB11 for RG Led
89     GPIOB->MODER |= GPIO_MODER_MODER0_0 | GPIO_MODER_MODER1_0 |
90                   GPIO_MODER_MODER2_0 | GPIO_MODER_MODER3_0 |
91                   GPIO_MODER_MODER4_0 | GPIO_MODER_MODER5_0 |
92                   GPIO_MODER_MODER6_0 | GPIO_MODER_MODER7_0 |
93                   GPIO_MODER_MODER10_0 | GPIO_MODER_MODER11_0;
94     GPIOB->ODR &= ~(GPIO_ODR_10 | GPIO_ODR_11); // Make sure they are not on
95
96     // Initialise PA0, PA1, PA2 and PA3 for SW0, SW1, SW2 and SW3
97     GPIOA->MODER &= ~(GPIO_MODER_MODER0 | GPIO_MODER_MODER1 | GPIO_MODER_MODER2
98                     | GPIO_MODER_MODER3);
99     GPIOA->PUPDR |= GPIO_PUPDR_PUPDR0_0 | GPIO_PUPDR_PUPDR1_0
100                  | GPIO_PUPDR_PUPDR2_0 | GPIO_PUPDR_PUPDR3_0; // Enable pullup resistors

```

```

101
102 // Initialise PA5 for ADC1
103 GPIOA->MODER |= GPIO_MODER_MODER5;
104 }
105
106 /*
107  * @brief Initialise the ADC to POT0
108  * @params None
109  * @retval None
110  */
111 static void init_ADC(void) {
112     // Enable the ADC clock in the RCC
113     RCC->APB2ENR |= RCC_APB2ENR_ADCEN;
114
115     // Select ADC channel 5 for POT0
116     ADC1->CHSELR |= ADC_CHSELR_CHSEL5;
117
118     // Enable the ADC peripheral
119     ADC1->CR |= ADC_CR_ADEN;
120
121     // Wait for the ADC to become ready
122     while (!(ADC1->ISR & ADC_ISR_ADRDY)) {
123         __asm("nop");
124     }
125 }
126
127 /*
128  * @brief Initialise the NVIC for pushbutton interrupts
129  * @params None
130  * @retval None
131  */
132 static void init_NVIC(void) {
133     NVIC_EnableIRQ(EXTI0_1_IRQn); // For lines 0 and 1
134     NVIC_EnableIRQ(EXTI2_3_IRQn); // For lines 2 and 3
135     NVIC_EnableIRQ(TIM14_IRQn); // For TIM14
136 }
137
138 /*
139  * @brief Initialise the EXTI lines for pushbutton interrupts
140  * @params None
141  * @retval None
142  */
143 static void init_EXTI(void) {
144     RCC->APB2ENR |= RCC_APB2ENR_SYSCFGCOMPEN; // Enable the SYSCFG and COMP RCC clock
145     SYSCFG->EXTICR[1] &= ~(0xFFFF); // Map PA0 and PA1 to external interrupt lines
146
147     EXTI->FTSR |= EXTI_FTSR_TR0 | EXTI_FTSR_TR1 | EXTI_FTSR_TR2 | EXTI_FTSR_TR3; // Configure
148     ↪ trigger to falling edge
149     EXTI->IMR |= EXTI_IMR_MR0 | EXTI_IMR_MR1 | EXTI_IMR_MR2 | EXTI_IMR_MR3; // Unmask the
150     ↪ interrupts

```

```
151 /*
152  * @brief Initialise TIM14 for battery checking
153  * @params None
154  * @retval None
155  */
156 static void init_TIM14(void) {
157     // Enable the clock for TIM14
158     RCC->APB1ENR |= RCC_APB1ENR_TIM14EN;
159
160     // Set the frequency to 1Hz
161     TIM14->PSC = 4800;
162     TIM14->ARR = 10000;
163
164     // Enable the interrupt
165     TIM14->DIER |= 0x1; // Enable the UIE (Update Interrupt Enable)
166     TIM14->CR1 &= ~(1 << 2); // Make sure the interrupt is not disabled in the Control Register
167     ↪ 1
168
169     // Make sure the counter is at zero
170     TIM14->CNT = 0;
171
172     // Enable the timer
173     TIM14->CR1 |= 0x1;
174 }
175
176 /*
177  * @brief Rational addition of a safe 2 line write to the LCD
178  * @params *string1: Pointer to the string to be written to line 1
179  *          *string2: Pointer to the string to be written to line 2
180  * @retval None
181  */
182 static void lcd_put2String(uint8_t *string1, uint8_t *string2) {
183     lcd_command(CURSOR_HOME);
184     lcd_command(CLEAR);
185     lcd_putstring(string1);
186     lcd_command(LINE_TWO);
187     lcd_putstring(string2);
188 }
189
190 /*
191  * @brief Get the state of the specified switch, with debouncing of predefined length
192  * @params pb: Pushbutton number
193  * @retval True or false when pressed and not pressed rsp.
194  */
195 static uint8_t getSW(uint8_t pb) {
196     uint8_t pbBit;
197
198     switch (pb) {
199     case 0:
200         pbBit = GPIO_IDR_0;
```

```
201     break;
202 case 1:
203     pbBit = GPIO_IDR_1;
204     break;
205 case 2:
206     pbBit = GPIO_IDR_2;
207     break;
208 case 3:
209     pbBit = GPIO_IDR_3;
210     break;
211 default:
212     return FALSE;
213 }
214
215 if (!(GPIOA->IDR & pbBit)) {
216     delay(DEBOUNCE_MS * 1000);
217     if (!(GPIOA->IDR & pbBit)) {
218         return TRUE;
219     } else {
220         return FALSE;
221     }
222 } else {
223     return FALSE;
224 }
225 }
226
227 /*
228  * @brief Kick off and grab an ADC conversion
229  * @params None
230  * @retval None
231  */
232 static uint16_t getADC(void) {
233     // Start a conversion
234     ADC1->CR |= ADC_CR_ADSTART;
235
236     // Wait for the conversion to finish
237     while (!(ADC1->ISR & ADC_ISR_EOC)) {
238         __asm("nop");
239     }
240
241     // Return the result of the conversion
242     return (uint16_t)(ADC1->DR);
243 }
244
245 /*
246  * @brief Interrupt Request Handler for EXTI Lines 2 and 3 (PB0 and PB1)
247  * @params None
248  * @retval None
249  */
250 void EXTI0_1_IRQHandler(void) {
```

```
251 // Check which button generated the interrupt
252 if (getSW(0)) {
253     // Check the state of the program
254     switch (programState) {
255         case PROG_STATE_WAIT_FOR_SW0:
256             // If we were waiting for SW0, display the menu
257             display(DISPLAY_MENU, 0);
258
259             // Change program state
260             programState = PROG_STATE_WAIT_FOR_BUTTON;
261             break;
262         default:
263             break;
264     }
265 } else if (getSW(1)) {
266     // Check the state of the program
267     switch (programState) {
268         case PROG_STATE_WAIT_FOR_BUTTON:
269             // If we were waiting for another button:
270             rainCounter++; // Increment the rain counter
271             display(DISPLAY_RAIN_BUCKET, 0); // Notify the user
272             break;
273         default:
274             break;
275     }
276 }
277
278 // Clear the interrupt pending bit
279 EXTI->PR |= EXTI_PR_PR0 | EXTI_PR_PR1;
280 }
281
282 /*
283  * @brief Interrupt Request Handler for EXTI Lines 2 and 3 (PB2 and PB3)
284  * @params None
285  * @retval None
286  */
287 void EXTI2_3_IRQHandler(void) {
288     if (getSW(2)) {
289         switch (programState) {
290             case PROG_STATE_WAIT_FOR_BUTTON:
291                 display(DISPLAY_RAINFALL, rainCounter);
292                 break;
293             default:
294                 break;
295         }
296     } else if (getSW(3)) {
297         switch (programState) {
298             case PROG_STATE_WAIT_FOR_BUTTON:
299                 display(DISPLAY_BAT, batVoltage);
300                 break;
```

```
301     default:
302         break;
303     }
304 }
305 EXTI->PR |= EXTI_PR_PR2 | EXTI_PR_PR3; // Clear the interrupt pending bit
306 }
307
308 /*
309  * @brief Interrupt Request Handler for TIM14
310  * @params None
311  * @retval None
312  */
313 void TIM14_IRQHandler(void) {
314     // Check the battery voltage
315     check_battery();
316
317     // Clear the interrupt pending bit
318     TIM14->SR &= ~TIM_SR_UIF;
319 }
320
321 /*
322  * @brief Check the "battery voltage" and display it
323  * @params None
324  * @retval None
325  */
326 static void check_battery(void) {
327     // Grab the ADC value, convert to uV and then to battery voltage
328     uint16_t adcVal = getADC();
329     uint32_t uVoltage = adcVal * ADC_GRAIN;
330     batVoltage = 7.21*(uVoltage/ADC_MULTIPLIER);
331
332     // Check for voltage threshold and change the LED accordingly
333     if (batVoltage <= BAT_THRESHOLD) {
334         GPIOB->ODR &= ~(1 << 11);
335         GPIOB->ODR |= (1 << 10);
336     } else {
337         GPIOB->ODR &= ~(1 << 10);
338         GPIOB->ODR |= (1 << 11);
339     }
340 }
341
342 /*
343  * @brief Display the specified data on the screen
344  * @params displayType: What to display on the screen
345  *         ...: Data to display for the given type
346  * @retval None
347  */
348 void display(displayType_t displayType, float data) {
349     // Switch on what needs to be displayed
350     switch (displayType) {
```

```
351 case DISP_BAT: {
352     // Display the battery voltage on the LCD
353     lcd_command(CLEAR);
354     lcd_command(CURSOR_HOME);
355     lcd_putstr("Battery:");
356     lcd_command(LINE_TWO);
357
358     // Generate the string with the batter voltage
359     uint8_t *string = ConverttoBCD(data, 2, 3);
360     lcd_putstr(string);
361     lcd_putstr(" V");
362
363     // De-allocate the memory used for the battery string
364     free(string);
365     break;
366 }
367 case DISP_RAINFALL: {
368     // Display the rainfall amount on the LCD
369     lcd_command(CLEAR);
370     lcd_command(CURSOR_HOME);
371     lcd_putstr("Rainfall:");
372     lcd_command(LINE_TWO);
373
374     // Fetch and convert the rainfall to a string
375     float rain = 0.2*data;
376     uint8_t *string = ConverttoBCD(rain, 4, 1);
377     lcd_putstr(string);
378     lcd_putstr(" mm");
379
380     // De-allocate the memory used for the rainfall string
381     free(string);
382     break;
383 }
384 case DISP_RAIN_BUCKET:
385     // Display the bucket tip notification LCD
386     lcd_put2String("Rain bucket tip", "");
387     break;
388 case DISP_MENU:
389     // Display the menu on the LCD
390     lcd_put2String("Weather Station", "Press SW2 or SW3");
391     break;
392 case DISP_WELCOME:
393     // Display the welcome on the LCD
394     lcd_put2String("EEE3017W Prac 6", "Sean & Sean");
395     break;
396 default:
397     break;
398 }
399 }
400
```



```
401 /*
402  * @brief Convert the float given to a string
403  * @params rain: Rain in mm
404  *          dec: Number of digits to the left of the decimal point
405  *          frac: Number of decimal places (precision)
406  * @retval Pointer to the converted string
407  * @note String must be freed after use
408  */
409 static uint8_t *ConverttoBCD(float number, uint8_t dec, uint8_t frac) {
410     uint8_t *string; // Pointer to the resulting string
411     uint32_t rainDec = number*pow(10,frac); // Shift all digits to be used onto the left side
412     ↪ of the decimal point
413     uint32_t strLength = (dec + frac + 2)*sizeof(uint8_t); // Calculate the length of the
414     ↪ require string given the accuracy parameters
415     string = malloc(strLength); // Allocate space for the resulting string
416     memset(string, '0', strLength); // Set all characters in the string to zeroes
417
418     // Loop through the digits in the newly formed integer number and place the digits in the
419     ↪ string
420     int pos = 0;
421     int dig = 0;
422     for (pos = 0; pos < strLength; pos++) {
423         // If we reach the end of the decimal part of the number, skip a position for placement
424         ↪ of the decimal point
425         if (pos == dec) {
426             pos++;
427         }
428
429         // Extract the digit from the newly formed integer number based on the position
430         uint32_t multiplier = pow(10, strLength-dig-3);
431         uint32_t digit = (uint32_t)(rainDec/multiplier);
432         string[pos] = (uint8_t)(digit + 48); // Convert the number to ASCII by adding 48 to it
433         rainDec -= digit*multiplier; // Subtract the extracted digit from the integer number
434
435         // Increment the digit number
436         dig++;
437     }
438
439     // Place the decimal point and the null terminator in the correct positions
440     string[dec] = '.';
441     string[strLength - 1] = '\0';
442
443     // Return the pointer to the converted string
444     return string;
445 }
```