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:

```
/*
* @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 the Analog to Digital Converter (ADC):

```c
/*
 * @brief Initialise the ADC to POT0
 * @params None
 * @retval None
 */
static void init_ADC(void) {
  // Enable the ADC clock in the RCC
  RCC->APB2ENR |= RCC_APB2ENR_ADCEN;

  // Select ADC channel 5 for POT0
  ADC1->CHSELR |= ADC_CHSELR_CHSEL5;

  // Enable the ADC peripheral
  ADC1->CR |= ADC_CR_ADEN;

  // Wait for the ADC to become ready
  while (!(ADC1->ISR & ADC_ISR_ADRDY)) {
    __asm("nop");
  }
}
```

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
}
```

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

```c
/*
 * @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
}
```

(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:

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

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

(c)

Below are the functions to monitor the battery voltage:

```c
/*
 * @brief Kick off and grab an ADC conversion
 * @params None
 * @retval None
 */
static uint16_t getADC(void) {
  // Start a conversion
  ADC1->CR |= ADC_CR_ADSTART;

  // Wait for the conversion to finish
  while (!(ADC1->ISR & ADC_ISR_EOC)) {
    __asm("nop");
  }

  // Return the result of the conversion
  return (uint16_t)(ADC1->DR);
}

/*
 * @brief Check the "battery voltage" and display it
 * @params None
 * @retval None
 */
static void check_battery(void) {
  // Grab the ADC value, convert to uV and then to battery voltage
  uint16_t adcVal = getADC();
  uint32_t uVoltage = adcVal * ADC_GRAIN;
  batVoltage = 7.21*(uVoltage/ADC_MULTIPLIER);

  // Check for voltage threshold and change the LED accordingly
  if (batVoltage <= BAT_THRESHOLD) {
    GPIOB->ODR &= ~(1 << 11);
    GPIOB->ODR |= (1 << 10);
  } else {
    GPIOB->ODR &= ~(1 << 10);
    GPIOB->ODR |= (1 << 11);
  }
}
```

(d)

Below is the EXTI handler for lines 0 and 1:

```c
/*
 * @brief Interrupt Request Handler for EXTI Lines 2 and 3 (PB0 and PB1)
 * @params None
 * @retval None
 */
void EXTI0_1_IRQHandler(void) {
  // Check which button generated the interrupt
  if (getSW(0)) {
    // Check the state of the program
    switch (programState) {
    case PROG_STATE_WAIT_FOR_SW0:
      // If we were waiting for SW0, display the menu
      display(DISP_MENU, 0);

      // Change program state
      programState = PROG_STATE_WAIT_FOR_BUTTON;
      break;
    default:
      break;
    }
  } else if (getSW(1)) {
    // Check the state of the program
    switch (programState) {
    case PROG_STATE_WAIT_FOR_BUTTON:
      // If we were waiting for another button:
      rainCounter++; // Increment the rain counter
      display(DISP_RAIN_BUCKET, 0); // Notify the user
      break;
    default:
      break;
    }
  }

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

(e)

Below is the function to convert floats to BCD:

```c
/*
 * @brief Convert the float given to a string
 * @params rain: Rain in mm
 *         dec: Number of digits to the left of the decimal point
 *         frac: Number of decimal places (precision)
 * @retval Pointer to the converted string
 * @note String must be freed after use
 */
static uint8_t *ConverttoBCD(float number, uint8_t dec, uint8_t frac) {
  uint8_t *string; // Pointer to the resulting string
  uint32_t rainDec = number*pow(10,frac); // Shift all digits to be used onto the left side of the decimal point
  uint32_t strLength = (dec + frac + 2)*sizeof(uint8_t); // Calculate the length of the require string given the accuracy parameters
  string = malloc(strLength); // Allocate space for the resulting string
  memset(string, '0', strLength); // Set all characters in the string to zeroes

  // Loop through the digits in the newly formed integer number and place the digits in the string
  int pos = 0;
  int dig = 0;
  for (pos = 0; pos < strLength; pos++) {
    // If we reach the end of the decimal part of the number, skip a position for placement of the decimal point
    if (pos == dec) {
      pos++;
    }

    // Extract the digit from the newly formed integer number based on the position
    uint32_t multiplier = pow(10, strLength-dig-3);
    uint32_t digit = (uint32_t)(rainDec/multiplier);
    string[pos] = (uint8_t)(digit + 48); // Convert the number to ASCII by adding 48 to it
    rainDec -= digit*multiplier; // Subtract the extracted digit from the integer number

    // Increment the digit number
    dig++;
  }

  // Place the decimal point and the null terminator in the correct positions
  string[dec] = '.';
  string[strLength - 1] = '\0';

  // Return the pointer to the converted string
  return string;
}
```

(e)

Below is the function to display values on the LCD:

```c
/*
 * @brief Display the specified data on the screen
 * @params displayType: What to display on the screen
 *         ...: Data to display for the given type
 * @retval None
 */
void display(displayType_t displayType, float data) {
  // Switch on what needs to be displayed
  switch (displayType) {
  case DISP_BAT: {
    // Display the battery voltage on the LCD
    lcd_command(CLEAR);
    lcd_command(CURSOR_HOME);
    lcd_putstring("Battery:");
    lcd_command(LINE_TWO);

    // Generate the string with the batter voltage
    uint8_t *string = ConverttoBCD(data, 2, 3);
    lcd_putstring(string);
    lcd_putstring(" V");

    // De-allocate the memory used for the battery string
    free(string);
    break;
  }
  case DISP_RAINFALL: {
    // Display the rainfall amount on the LCD
    lcd_command(CLEAR);
    lcd_command(CURSOR_HOME);
    lcd_putstring("Rainfall:");
    lcd_command(LINE_TWO);

    // Fetch and convert the rainfall to a string
    float rain = 0.2*data;
    uint8_t *string = ConverttoBCD(rain, 4, 1);
    lcd_putstring(string);
    lcd_putstring(" mm");

    // De-allocate the memory used for the rainfall string
    free(string);
    break;
  }
  case DISP_RAIN_BUCKET:
    // Display the bucket tip notification LCD
    lcd_put2String("Rain bucket tip", "");
    break;
  case DISP_MENU:
    // Display the menu on the LCD
    lcd_put2String("Weather Station", "Press SW2 or SW3");
    break;
  case DISP_WELCOME:
    // Display the welcome on the LCD
    lcd_put2String("EEE3017W Prac 6", "Sean & Sean");
    break;
  default:
    break;
  }
}
```

(f)

Below is the completed code as tested:

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

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <stm32f0xx.h>
#include "math.h"
#include "diag/Trace.h"
#include "lcd_stm32f0.h"

#define TRUE            1
#define FALSE           0

#define DEBOUNCE_MS     20
#define ADC_GRAIN       806 // ADC uV per bits
#define ADC_MULTIPLIER  (float)1000000 // Grain multiplier
#define BAT_THRESHOLD   (float)14.0 // Low battery threshold

// == Type Definitions
typedef enum {
  PROG_STATE_INIT,
  PROG_STATE_WAIT_FOR_SW0,
  PROG_STATE_WAIT_FOR_BUTTON,
  PROG_STATE_BUCKET_TIP
} programState_t;

typedef enum {
  DISP_RAIN_BUCKET,
  DISP_RAINFALL,
  DISP_BAT,
  DISP_WELCOME,
  DISP_MENU
} displayType_t;

// == Global Variables
programState_t programState; // To keep track of the program state throughout execution
uint32_t rainCounter; // Keep track of the rain [0.2mm]
float batVoltage; // Battery voltage from 1Hz samples

// == Function Prototypes
static void init_ports(void);
static void init_ADC(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 getSW(uint8_t pb);
```

```c
static void check_battery(void);
static uint16_t getADC(void);
static void display(displayType_t displayType, float data);
static uint8_t *ConverttoBCD(float number, uint8_t dec, uint8_t frac);

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

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

  display(DISP_WELCOME, 0);
  programState = PROG_STATE_WAIT_FOR_SW0;

  // 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
```

```
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
      ↪  trigger to falling edge
148   EXTI->IMR |= EXTI_IMR_MR0 | EXTI_IMR_MR1 | EXTI_IMR_MR2 | EXTI_IMR_MR3; // Umask the
      ↪  interrupts
149 }
150
```

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

  // Set the frequency to 1Hz
  TIM14->PSC = 4800;
  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;

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


/*
 * @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) {
  lcd_command(CURSOR_HOME);
  lcd_command(CLEAR);
  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 getSW(uint8_t pb) {
  uint8_t pbBit;

  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;
  }

  if (!(GPIOA->IDR & pbBit)) {
    delay(DEBOUNCE_MS * 1000);
    if (!(GPIOA->IDR & pbBit)) {
      return TRUE;
    } else {
      return FALSE;
    }
  } else {
    return FALSE;
  }
}

/*
 * @brief Kick off and grab an ADC conversion
 * @params None
 * @retval None
 */
static uint16_t getADC(void) {
  // Start a conversion
  ADC1->CR |= ADC_CR_ADSTART;

  // Wait for the conversion to finish
  while (!(ADC1->ISR & ADC_ISR_EOC)) {
    __asm("nop");
  }

  // Return the result of the conversion
  return (uint16_t)(ADC1->DR);
}

/*
 * @brief Interrupt Request Handler for EXTI Lines 2 and 3 (PB0 and PB1)
 * @params None
 * @retval None
 */
void EXTI0_1_IRQHandler(void) {
```

```
  // Check which button generated the interrupt
  if (getSW(0)) {
    // Check the state of the program
    switch (programState) {
    case PROG_STATE_WAIT_FOR_SW0:
      // If we were waiting for SW0, display the menu
      display(DISP_MENU, 0);

      // Change program state
      programState = PROG_STATE_WAIT_FOR_BUTTON;
      break;
    default:
      break;
    }
  } else if (getSW(1)) {
    // Check the state of the program
    switch (programState) {
    case PROG_STATE_WAIT_FOR_BUTTON:
      // If we were waiting for another button:
      rainCounter++; // Increment the rain counter
      display(DISP_RAIN_BUCKET, 0); // Notify the user
      break;
    default:
      break;
    }
  }

  // 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 (getSW(2)) {
    switch (programState) {
    case PROG_STATE_WAIT_FOR_BUTTON:
      display(DISP_RAINFALL, rainCounter);
      break;
    default:
      break;
    }
  } else if (getSW(3)) {
    switch (programState) {
    case PROG_STATE_WAIT_FOR_BUTTON:
      display(DISP_BAT, batVoltage);
      break;
```

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

/*
 * @brief Interrupt Request Handler for TIM14
 * @params None
 * @retval None
 */
void TIM14_IRQHandler(void) {
  // Check the battery voltage
  check_battery();

  // Clear the interrupt pending bit
  TIM14->SR &= ~TIM_SR_UIF;
}

/*
 * @brief Check the "battery voltage" and display it
 * @params None
 * @retval None
 */
static void check_battery(void) {
  // Grab the ADC value, convert to uV and then to battery voltage
  uint16_t adcVal = getADC();
  uint32_t uVoltage = adcVal * ADC_GRAIN;
  batVoltage = 7.21*(uVoltage/ADC_MULTIPLIER);

  // Check for voltage threshold and change the LED accordingly
  if (batVoltage <= BAT_THRESHOLD) {
    GPIOB->ODR &= ~(1 << 11);
    GPIOB->ODR |= (1 << 10);
  } else {
    GPIOB->ODR &= ~(1 << 10);
    GPIOB->ODR |= (1 << 11);
  }
}

/*
 * @brief Display the specified data on the screen
 * @params displayType: What to display on the screen
 *         ...: Data to display for the given type
 * @retval None
 */
void display(displayType_t displayType, float data) {
  // Switch on what needs to be displayed
  switch (displayType) {
```

```
  case DISP_BAT: {
    // Display the battery voltage on the LCD
    lcd_command(CLEAR);
    lcd_command(CURSOR_HOME);
    lcd_putstring("Battery:");
    lcd_command(LINE_TWO);

    // Generate the string with the batter voltage
    uint8_t *string = ConverttoBCD(data, 2, 3);
    lcd_putstring(string);
    lcd_putstring(" V");

    // De-allocate the memory used for the battery string
    free(string);
    break;
  }
  case DISP_RAINFALL: {
    // Display the rainfall amount on the LCD
    lcd_command(CLEAR);
    lcd_command(CURSOR_HOME);
    lcd_putstring("Rainfall:");
    lcd_command(LINE_TWO);

    // Fetch and convert the rainfall to a string
    float rain = 0.2*data;
    uint8_t *string = ConverttoBCD(rain, 4, 1);
    lcd_putstring(string);
    lcd_putstring(" mm");

    // De-allocate the memory used for the rainfall string
    free(string);
    break;
  }
  case DISP_RAIN_BUCKET:
    // Display the bucket tip notification LCD
    lcd_put2String("Rain bucket tip", "");
    break;
  case DISP_MENU:
    // Display the menu on the LCD
    lcd_put2String("Weather Station", "Press SW2 or SW3");
    break;
  case DISP_WELCOME:
    // Display the welcome on the LCD
    lcd_put2String("EEE3017W Prac 6", "Sean & Sean");
    break;
  default:
    break;
  }
}

```

```
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
      ↪  of the decimal point
412   uint32_t strLength = (dec + frac + 2)*sizeof(uint8_t); // Calculate the length of the
      ↪  require string given the accuracy parameters
413   string = malloc(strLength); // Allocate space for the resulting string
414   memset(string, '0', strLength); // Set all characters in the string to zeroes
415 
416   // Loop through the digits in the newly formed integer number and place the digits in the
      ↪  string
417   int pos = 0;
418   int dig = 0;
419   for (pos = 0; pos < strLength; pos++) {
420     // If we reach the end of the decimal part of the number, skip a position for placement
        ↪  of the decimal point
421     if (pos == dec) {
422       pos++;
423     }
424 
425     // Extract the digit from the newly formed integer number based on the position
426     uint32_t multiplier = pow(10, strLength-dig-3);
427     uint32_t digit = (uint32_t)(rainDec/multiplier);
428     string[pos] = (uint8_t)(digit + 48); // Convert the number to ASCII by adding 48 to it
429     rainDec -= digit*multiplier; // Subtract the extracted digit from the integer number
430 
431     // Increment the digit number
432     dig++;
433   }
434 
435   // Place the decimal point and the null terminator in the correct positions
436   string[dec] = '.';
437   string[strLength - 1] = '\0';
438 
439   // Return the pointer to the converted string
440   return string;
441 }
442 
443 
444 // ----------------------------------------------------------------------------
```