

BENG PT Embedded Electronic Systems
Embedded C

Generating a program for a specified
embedded requirement in C

2111739

Joshua Ofosu Asare

ACDA4013

```

/** Author: 2111739 */

#include <main.h>
#define LCD_ENABLE_PIN PIN_D2
#define LCD_RS_PIN PIN_D0
#define LCD_RW_PIN PIN_D1
#define LCD_DATA4 PIN_D4
#define LCD_DATA5 PIN_D5
#define LCD_DATA6 PIN_D6
#define LCD_DATA7 PIN_D7

#include <lcd.c>

/** USER BEGIN defines */
#define MOTOR_A PIN_C0 // defining motor a as pin c0
#define MOTOR_B PIN_C1 // defining motor b as pin c0
#define BUTTON PIN_A1 // defining button as pin a1
/** USER END defines */

/** USER BEGIN Function Prototypes */
void analog(void);
void adc_motor(void);
void button_motor(void);
void kitt_mode(unsigned int *);
/** USER END Function Prototypes */

/** USER BEGIN Private Variables */
unsigned int adc; // stores the value for the potentiometer
unsigned int flag = 0;
/** USER END Private Variables */

/** USER BEGIN enum */
enum // This enum is used for brevity of code later on.
{
    RIGHT = 1,
    LEFT = 2
};
/** USER END enum */

/** USER BEGIN Code 1 */
void main()
{
    unsigned int switches;
    unsigned int previous_switches;
    static unsigned int led_bitfield;
    led_bitfield = 1;
    int array[12] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
    setup_adc_ports(SAN0);

```

```

setup_adc(ADC_CLOCK_INTERNAL | ADC_TAD_MUL_0); // setup adc operation

lcd_init(); // initialise the LCD

if (switches != previous_switches)
    printf(lcd_putc, "\f");

while (TRUE)
{
    switches = input_a();
    switches &= 0x3c;
    switches >>= 2;
    switch (switches)
    {
        case 0x01:
            analog();
            break;
        case 0x02:
            adc_motor();
            break;
        case 0x04:
            button_motor();
            break;
        case 0x08:
            kitt_mode(&led_bitfield);
            break;
        default:
            break;
    }
    previous_switches = switches;
}

while (TRUE)
{
    /** Loop forever
     * This should never be reached but is placed to guard the program
     * from executing arbitrary code in the case that a voltage dip which
     * is not long enough to cause a brownout occurs and skips an
     * instruction, to prevent the main() function from returning.
     * This has been added since motors are being controlled which are
     * potentially dangerous when executing arbitrary code
     */
}
}

/** USER END Code 1 */

/* Assignment part 1:

```

```

* Sets the ADC channel and waits for it to settle, reads a value from the
* ADC and outputs it directly to the LEDs. Prints to the terminal and LCD.
* */
void analog()
{
    set_adc_channel(0);
    delay_us(30);
    adc = read_adc();
    output_b(adc);

    lcd_gotoxy(1, 1);
    printf(lcd_putc, "          %03u", adc);
    printf("%u\n\r", adc);
}

/** Reads a value from the ADC, if it is above 175, the motor rotates
* anti-clockwise. If the value is below 75, it rotates clockwise.
* Otherwise it remains stationary */
void adc_motor()
{
    adc = read_adc();
    /** Rotate the motor clockwise if the adc returns a value less than 75
    * Print this to the screen
    */
    if (adc < 75)
    {
        output_high(MOTOR_A);
        output_low(MOTOR_B);
        lcd_gotoxy(1, 2);
        printf(lcd_putc, "%u :Clockwise      ", adc);
    }
    /** Drive both motor pins low to disable it if the adc is between 76 and
174
    * Print this to the screen
    */
    else if ((adc >= 76) && (adc <= 174))
    {
        output_low(MOTOR_A);
        output_low(MOTOR_B);
        // printf("Motor off\n\r");
        lcd_gotoxy(1, 2);
        printf(lcd_putc, "%u: Off          ", adc);
    }
    else if (adc > 175)
    {
        /** opposite of clockwise config
        * Print this to the screen
        */

```

```

        output_high(MOTOR_B);
        output_low(MOTOR_A);
        lcd_gotoxy(1, 2);
        printf(lcd_putc, "%u :Anti-clockwise ", adc);
    }
}

/** While the button is pressed, the motor rotates clockwise
 * The status is printed to the LCD and repeated on the terminal
 * only once.
 */
void button_motor()
{
    /** Flag "flag" to indicate if we have already printed to the terminal
    once
    * Flag "previous_flag" to containing the previous function calls flag to
    * work out if we should print a message to the terminal once.
    */
    static int flag;
    static int previous_flag;
    /** Variable to store the state of the button */
    static int button_state;
    button_state = input(BUTTON);
    /**
    * If the button has been pressed, print this to the screen
    */
    if (button_state == TRUE)
    {
        output_high(MOTOR_A);
        output_low(MOTOR_B);
        lcd_gotoxy(21, 1);
        printf(lcd_putc, "Button pressed\r\n");
        flag = TRUE;
    }
    else
    {
        output_low(MOTOR_A);
        output_low(MOTOR_B);
        flag = FALSE;
        lcd_gotoxy(21, 1);
        printf(lcd_putc, "Button released\r\n");
    }
    /** If there has been a change of state of the button, and it is pressed
    * or released, print to the screen the button has been pressed/released.
    */
    if ((previous_flag != flag) & button_state)
        printf("Button pressed\r\n");
    else if ((previous_flag != flag) & !button_state)

```

```

        printf("Button released\r\n");
    previous_flag = flag;
}

void kitt_mode(unsigned int *led_bitfield)
{
    static int direction;
    lcd_gotoxy(21, 2);
    printf(lcd_putc, "Kitt mode");
    switch (direction)
    {
        case LEFT:
            *led_bitfield <= 1;
            break;
        case RIGHT:
            *led_bitfield >= 1;
            break;
        default:
            break;
    }
    output_b(*led_bitfield);
    /** Instead of hard coding the values via a lookup table, and to allow
     * the kitt mode to be stopped early, we use a variable to keep track
     * of where we are in the loop, using boundary checks to change the
     * direction the leds propagate in
     */
    if (*led_bitfield == 0x80)
        direction = RIGHT;
    else if (*led_bitfield == 0x01)
        direction = LEFT;

    delay_ms(150);
}

```