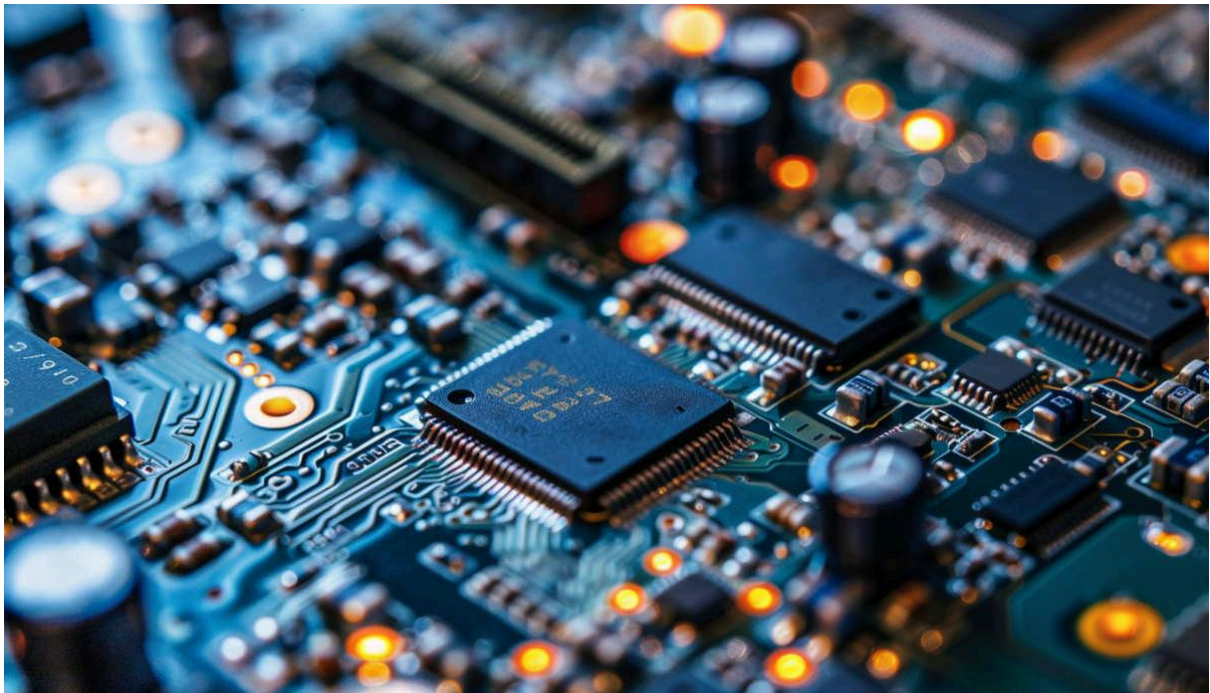


Embedded C Assignment Circuit

Najib Abdi

ACDA4013

Generating a program for a specific embedded requirement in C.



<https://datascientest.com/en/all-about-embedded-systems>

Introduction

For the project, I created an embedded C program to drive a DC motor, read analog input using an ADC and output to an LCD and also generate dynamic LED light effects. I had no physical hardware to test on, so I tackled the assignment by simulating and coding keeping in mind to how actual microcontroller systems would behave in the real world for every line of code that was written. I tried to use general embedded programming concepts like writing code in modular format, input handling properly, and proper feedback to the user using display systems throughout the process. I outline below a summary of the procedures I used while developing the program and decisions I made along the way in addition to the concepts I learned on a theoretical level.

Design and implementation

The objective of the assignment was to write C code that could interact with connected devices such as push-buttons, LEDs, motors, and an LCD screen, through digital and analog I/O ports of a PIC microcontroller. The design was structured around four operational modes, each triggered by a specific switch SW1A to SW1D, with more input provided by a push-button and two analog channels. The code had to be logical, responsive, and modular, even without access to physical test benches.

The first step in the development process involved defining the required hardware interface assumptions, no physical connections were made but the coding logic was written with standard PIC microcontroller conventions in mind.

The program uses port bits such as `portbit.RB0` and `latchbits.LATC0` to detect switch inputs and outputs. `Portbit.RB0` for switch input and `latchbits.LATC0` for output pins. Also `LATC0` to control devices such as motors and `TRIS` registers were used to define the data directions of ports such as `TRISB=0x1F` ensures that the first five pins of `portb` are set as inputs. Which prepares them for communication with virtual switches in simulation. These configurations are important in microcontroller programming and even without running the code on physical devices. Being detailed at this level would determine whether the code would function properly in practice.

Code development and Operational Modes

In the program each mode of operation was associated with a specific task. Mode A is programmed to detect the state of a button and use it to control a motor's clockwise operations, echoing Pressed or Released on the LCD and to the virtual terminal. I did this using state tracking with a static `bool prev_botton` which is allowing the program to only respond when the button state changes and not on every loop iteration. The next hurdle was edge detection which is responding to changes in input state rather than constant values. This is important in embedded systems to prevent jitter and misreads due to signal bouncing or looping. This makes it efficient and avoids resource wasting in physical systems and optimises it.

ADC interaction is focused on by Mode B, the code basically reads an analog voltage using `adc_read(0)` after getting the value from a potentiometer on channel 0. And then shifts the value right by two bits so we can reduce the ten bit value to a eight bit value and outputs it to a LED array connected to `LATD`. I do this in order to be efficient and for mapping resolution constrained data into a hardware display like eight LEDs. I also formatted the ADC value and it is shown on the LCD which is necessary in monitoring real time analog data such as the sensors and voltage levels. This part of my program reinforces my knowledge of bitwise operations, data scaling and digital to human readable output handling.

ADC channel 1 is used for Mode C to find the direction of the motor. For example if the analog bumper is below ninety five the motor should run counterclockwise and if it is above one hundred and sixty five it will run clockwise or it stops. This shows how real life systems make decisions by sensor thresholds. For example using temperature sensors and light levels or rotation speeds. For multilayering information we use a display with multiple lines on the LCD here we use line one for the ADC value and line two for the motor direction which shows how embedded systems communicate information to the user in a minimal interface setup.

For Mode D it shows the knight rider LED pattern which basically creates a sweeping light demonstration. I did this by writing one hot value to the LED port and showing through the LED position. However there is a unique feature which is SW1D for each looping session. It is for if the mode is changing during the pattern and allows for an exit from it. I believe this shows a good design and planning of the code because it stays responsive even while doing tasks which is a norm in modern day embedded systems.

Code structure

As across all modes the LCD function is vital in user feedback. The function named `display_centered` calculates where to start the string on the LCD as to make it centered which was important to keep in mind because it allows for the code to be read easily and to be reused and because it's better than reusing cursor calculations. It's important because it allows the program to reuse the function across various LCD lines and display messages. In the planning stage I made sure to keep in mind the separation of the display logic from core functionality.

In relation to the skills involved it was important to consider how the logic flow was handled. A single infinite loop in main file checks each of the switches state in order and calls the required mode function when it needs to. This structure is effective for systems with minimal concurrency or where the tasks are mutually exclusive. In control flow within each mode it shows solid reasoning skills such as in Mode A where rather than checking the button state each loop and updating displays or the motor, the code compares the previous and current loop states. This decreases unnecessary processing and allows the code to do more power sensitive application where the efficiency is vital.

In terms of the code it was structured to be readable and maintainable and all the functions are declared on the top and most are implemented with single responsibilities. I also named the function very clearly such as `run_motor_cw` and `stop_motor_` which makes the code easy to read and extend. Also my commenting is clear and understandable. All the functions have a header comment which explains

the purpose which is important for individual and team collaborations if i worked with others such as for debugging or to expand on the code.

learning curve

This project helped me to build confidence in my programming skills and even though there are more complex models i could have used it was important to gain a foundation to build my knowledge on. If i was to expand on this project in the future i would make it a priority to learn interrupt handling, multitasking and communication protocols. For the PWM motor i could use instead of a simple on and off logic a I2C communication for the LCD after i gain the knowledge.

This project helped me to gain a foundation and learn more principles in embedded systems without the problems of physical hardware such as debounce handling and voltage level mismatches or even signal noise. But there were challenges such as learning how microcontrollers work internally and the different kinds such as STM32 and others which i haven't had the pleasure of learning as of yet but i did accidentally learn a bit of STM32. Hardships included how to simulate time delays and how each I/O register affects physical behaviour which harnessed my skills in system modeling and digital logic.

The transferrable skills i learned include structured logic, modular design and consistent testing. Also clear documentation which are very important in software careers I also improved my skills in task management because each mode had to be tested independently and then integrated properly. The skills such as abstract design problems such as how ADC values would translate with motor direction are important in most engineering fields.

Conclusion

In conclusion I gained valuable experience in embedded programming design such as in input reading and motor control to data display and visual effect such as the knight rider effect. By writing clean, logically sound code allowed me to develop skills in physical prototyping. This project increased my skill in many areas including digital I/O, analog to digital interfacing, motor direction control and user feedback, i made sure the code is sound and ready for real world physically testing.

C

```
/******
```

```
 * Embedded C Assignment – Motor, ADC, LED, LCD, and Knight  
Rider Control
```

```
 * Author: [Najib Abdi]
```

```
 * Date: [03/05/2025]
```

```
 * Description:
```

```
 * Controls Motor A LEDs and LCD based on SW1-A to SW1-D.
```

```
 * Demonstrates the use of ADC digital IO and LCD interfacing.
```

```
*****/
```

```
#include <xc.h>                // Device header (Make sure to  
adjust for your MCU)
```

```
#include <stdint.h>
```

```
#include <stdbool.h>
```

```
#include "lcd.h"                // Assumed LCD library
```

```
#include "adc.h"                // Assumed ADC library
```

```
#include "delay.h"              // Assumed delay library
```

```
// --- Hardware Definitions (adjust) ---
```

```
#define SW1A    PORTBbits.RB0
```

```
#define SW1B    PORTBbits.RB1
```

```
#define SW1C    PORTBbits.RB2
```

```
#define SW1D    PORTBbits.RB3
```

```
#define BUTTON  PORTBbits.RB4
```

```
#define MOTOR_A_CW    LATCbits.LATC0
```

```
#define MOTOR_A_CCW    LATCbits.LATC1
```

```
#define LED_PORT    LATD        // 8-bit port for LEDs
```

```
// --- Function Prototypes ---
```

```
void init_hardware(void);
```

```

void mode_A(void);
void mode_B(void);
void mode_C(void);
void mode_D(void);
void display_centered(char* text, uint8_t line);
void knight_rider(void);
void stop_motor(void);
void run_motor_cw(void);
void run_motor_ccw(void);

// --- Main Program ---
int main(void) {
    init_hardware();
    lcd_init();
    adc_init();

    while (1) {
        if (SW1A) {
            mode_A();
        } else if (SW1B) {
            mode_B();
        } else if (SW1C) {
            mode_C();
        } else if (SW1D) {
            mode_D();
        } else {
            stop_motor();
            lcd_clear();
        }
    }
    return 0;
}

/*****
 * Function: init_hardware
 * -----
 * Configures IO ADC LCD and initial states.
 *****/

```

```

void init_hardware(void) {
    // Configure ports as input/output as appropriate
    TRISB = 0x1F;    // SW1A-D and BUTTON as inputs
    TRISC = 0x00;    // Motor control pins as outputs
    TRISD = 0x00;    // LEDs as outputs
    LATC = 0x00;
    LATD = 0x00;
}

/*****
 * Function: mode_A
 * -----
 * SW1-A: BUTTON controls Motor A (CW while pressed).
 * LCD line 3: "Pressed"/"Released" centered.
 * Virtual terminal: Echo status once per change.
 *****/
void mode_A(void) {
    static bool prev_button = false;
    bool curr_button = BUTTON;

    if (curr_button && !prev_button) {
        // Button just pressed
        run_motor_cw();
        display_centered("Pressed", 3);
        printf("Button Pressed\r\n"); // Virtual terminal
    } else if (!curr_button && prev_button) {
        // Button just released
        stop_motor();
        display_centered("Released", 3);
        printf("Button Released\r\n");
    } else if (curr_button) {
        run_motor_cw();
    } else {
        stop_motor();
    }
    prev_button = curr_button;
}

```

```

/*****
 * Function: mode_B
 * -----
 * SW1-B: Show ADC value from RV1 on LEDs display centered on
LCD line 1 echo to terminal.
 *****/
void mode_B(void) {
    uint16_t adc_val = adc_read(0); // Read from channel 0
(RV1)
    LED_PORT = adc_val >> 2;          // Show upper 8 bits on
LEDs

    char buf[17];
    sprintf(buf, "ADC: %3d", adc_val);
    display_centered(buf, 1);
    printf("ADC Value: %d\r\n", adc_val); // Virtual terminal
    delay_ms(200);
}

/*****
 * Function: mode_C
 * -----
 * SW1-C: Use ADC value to control Motor A direction.
 * <95: CCW, >165: CW, 96-164: Stop.
 * LCD line 1: ADC value, line 2: direction, both centered.
 *****/
void mode_C(void) {
    uint16_t adc_val = adc_read(1); // Read from channel 1
(RC1)
    char buf1[17], buf2[17];

    sprintf(buf1, "ADC: %3d", adc_val);

    if (adc_val < 95) {
        run_motor_ccw();
        sprintf(buf2, "AntiClockwise");
    } else if (adc_val > 165) {
        run_motor_cw();

```

```

        sprintf(buf2, "Clockwise");
    } else {
        stop_motor();
        sprintf(buf2, "Stopped");
    }

    display_centered(buf1, 1);
    display_centered(buf2, 2);

    printf("ADC: %d, Dir: %s\r\n", adc_val, buf2); // Virtual
terminal
    delay_ms(200);
}

/*****
 * Function: mode_D
 * -----
 * SW1-D: Knight Rider LED pattern at 300ms interval.
 * LCD line 4: "Kitt mode" centered, echo once to terminal.
 *****/
void mode_D(void) {
    static bool first_entry = true;
    if (first_entry) {
        display_centered("Kitt mode", 4);
        printf("Kitt mode\r\n");
        first_entry = false;
    }
    knight_rider();
}

/*****
 * Function: display_centered
 * -----
 * Displays text centered on specified LCD line (1-4).
 *****/
void display_centered(char* text, uint8_t line) {
    uint8_t len = strlen(text);
    uint8_t pos = (16 - len) / 2; // For 16x4 LCD

```

```

    lcd_set_cursor(line, pos);
    lcd_write_str(text);
}

/*****
 * Function: knight_rider
 * -----
 * Implements Knight Rider LED pattern, left-right and back.
 *****/
void knight_rider(void) {
    uint8_t i;
    // Left to right
    for (i = 0; i < 8; i++) {
        LED_PORT = (1 << i);
        delay_ms(300);
        if (!SW1D) return; // Exit if mode changed
    }
    // Right to left
    for (i = 6; i > 0; i--) {
        LED_PORT = (1 << i);
        delay_ms(300);
        if (!SW1D) return;
    }
}

/*****
 * Motor Control Helper Functions
 *****/
void run_motor_cw(void) {
    MOTOR_A_CW = 1;
    MOTOR_A_CCW = 0;
}
void run_motor_ccw(void) {
    MOTOR_A_CW = 0;
    MOTOR_A_CCW = 1;
}
void stop_motor(void) {
    MOTOR_A_CW = 0;

```

```
    MOTOR_A_CCW = 0;  
}
```

```
/******  
* End of Program  
*****/
```