Embedded Systems- Bluetooth Controlled Motor Speed System

Final Project Report

Bindu Sagar Aluri (U96935703)

Objective: To Create a system that controls an electric motor using a PWM output signal from your smartphone as the input (12 VDC). Your smartphone app (requires Bluetooth connectivity) must display the speed % (0 to 100).

Problem Description:

- Here, we use C on a microcontroller and use PWM (Pulse Width Modulation) to regulate the motor speed while displaying the speed on an LCD screen.
- A Bluetooth connection is provided by the application for the user to change the motor speed. Initialization of the required peripherals, including the Serial Communication Ports, the I2C mode, and the GPIO pins for the LCD, is done by the application.
- The LCD screen and the PWM object are both initialized as well.
- The software waits for an input push on Bluetooth data in the main loop.
- When a key is pushed, the value in a buffer and the digits variable reflect the new value. The application modifies the PWM's duty cycle based on the user's input digits if the end key is depressed.
- The application additionally checks for and modifies the duty cycle received over Bluetooth if it is more than 100%.
- Following that, the LCD panel shows the current motor speed.
- In general, the application offers a straightforward user interface for controlling the motor speed through Bluetooth or a keyboard.

Pseudo code:

- Initialization:
  - Initialize the watchdog timer.
  - Initialize the Bluetooth module for communication.
  - Initialize the LCD display.
  - Set motor control pins as output.
- Enable Interrupts and Setup:
  - Enable interrupts and clear low-power mode lock.
  - Set up UART communication for Bluetooth.
- Main Loop:
  - Continuously check for new messages from Bluetooth.
  - If a new message is received:
    - Extract the PWM (Pulse Width Modulation) value from the message.
    - Display the received PWM value on the LCD.
    - Turn the motor on if the PWM value is greater than zero; otherwise, turn it off.
- UART Interrupt Service Routine:
  - Upon receiving a character via UART (Bluetooth):

- Read the received character.
- If the character is a newline:
  Terminate the message string.
  Set a flag indicating a new message.
- Increment the message counter.

- Motor Control Functions:
  – Function to turn the motor on by setting appropriate pins.
  – Function to turn the motor off by clearing appropriate pins.
- Delay Function:
  – Function to wait for a specified number of milliseconds by counting clock cycles.

C code:

```
#include <msp430.h>
#include <stdbool.h>
#include <string.h>
#include "LiquidCrystal_I2C.h" // Include header file for I2C LCD library

#define MAX_MESSAGE_LENGTH 12 // Define maximum length for received message

volatile char message[MAX_MESSAGE_LENGTH]; // Define buffer for received message
volatile unsigned int msg_counter = 0;      // Initialize message counter
volatile unsigned int new_msg = 0;          // Initialize flag for new message
volatile unsigned int DutyCycle = 0;        // Initialize variable for duty cycle

void bluetooth_init() {
    // Configure HC-05 Bluetooth module
    // Setup UART communication with appropriate baud rate
    // Configure GPIO pins for communication with HC-05
    UCA1CTLW0 |= UCSWRST;       // Put USCI in reset state
    UCA1CTLW0 |= UCSSEL__ACLK;  // Select ACLK as clock source
    UCA1BR0 = 3;                // Set Baud Rate Control Registers
    UCA1BR1 = 0;
    UCA1MCTLW = 0x9200;         // Set modulation control
    P4SEL1 &= ~BIT2;            // Configure P4.2 as TX
    P4SEL0 |= BIT2;
    UCA1CTLW0 &= ~UCSWRST;      // Release USCI from reset state
    UCA1IE |= UCRXIE;           // Enable USCI A1 RX interrupt
}

void lcd_init() {
    // Initialize and configure LCD display
    I2C_Init(0x27);         // Initialize I2C with appropriate LCD address
    LCD_Setup();            // Setup LCD
    LCD_ClearDisplay();     // Clear LCD display
    LCD_SetCursor(0, 0);        // Set cursor to first line, first column
    LCD_Write("SEND PWM VALUE");    // Display initial message on LCD
}

void display_message(const char *msg) {
    LCD_ClearDisplay();             // Clear LCD display
    LCD_SetCursor(0, 0);            // Set cursor to first line, first column
```

```
    LCD_Write("PWM VALUE:");        // Display label
    LCD_SetCursor(0, 1);            // Set cursor to second line, first column
    LCD_Write(msg);                 // Display received PWM value on LCD
}

void motor_on(void) {
    P4OUT |= BIT0 | BIT6; // Turn on motor by setting appropriate pins
    P4OUT &= ~BIT7;       // Set direction (if needed)
}

void motor_off(void) {
    P4OUT &= ~(BIT0 | BIT6); // Turn off motor by clearing appropriate pins
}

void delay_ms(int ms) {
    while (ms--) {
        __delay_cycles(1000); // Delay 1 ms assuming 1 MHz clock
    }
}

int main(void) {
    WDTCTL = WDTPW | WDTHOLD;       // Stop watchdog timer

    bluetooth_init();               // Initialize Bluetooth communication
    lcd_init();                      // Initialize LCD display

    P4DIR |= BIT0 | BIT6 | BIT7; // Set motor control pins as output

    PM5CTL0 &= ~LOCKLPM5;       // Unlock GPIO
    UCA1CTLW0 &= ~UCSWRST;      // Release USCI from reset state
    __enable_interrupt();       // Enable interrupts

    while (1) {
        if (new_msg) {                      // Check if new message is received
            DutyCycle = atoi(message);      // Extract PWM value from message
            display_message(message);       // Display received PWM value on LCD


            if (DutyCycle > 0) {            // If PWM value is greater than zero
                motor_on();                 // Turn the motor on
            } else {                        // If PWM value is zero or negative
                motor_off();                // Turn the motor off
            }

            new_msg = 0;                    // Clear new message flag
        }
    }
    return 0;
}

#pragma vector=USCI_A1_VECTOR                   // Interrupt service routine for UART
communication
__interrupt void USCI_A1_ISR(void) {
    switch (UCA1IV) {                       // Check interrupt vector
        case USCI_NONE: break;              // No interrupt
```

```
        case USCI_UART_UCRXIFG:             // Receive interrupt
            message[msg_counter++] = UCA1RXBUF;             // Read received
character
            if (message[msg_counter - 1] == '\n') {         // If newline character
is received
                message[msg_counter - 1] = '\0';            // Terminate string
                msg_counter = 0;                        // Reset message counter
                new_msg = 1;                    // Set new message flag
            }
            break;
        case USCI_UART_UCTXIFG: break;          // Transmit interrupt
        case USCI_UART_UCSTTIFG: break;         // Start bit interrupt
        case USCI_UART_UCTXCPTIFG: break;       // Transmit complete interrupt
    }
}
```

Wiring diagram: