Team members:

Priya Wilkhu
Elizabeth Sotomayor
Matthew Sharp
Bekhzodbek Moydinboyev

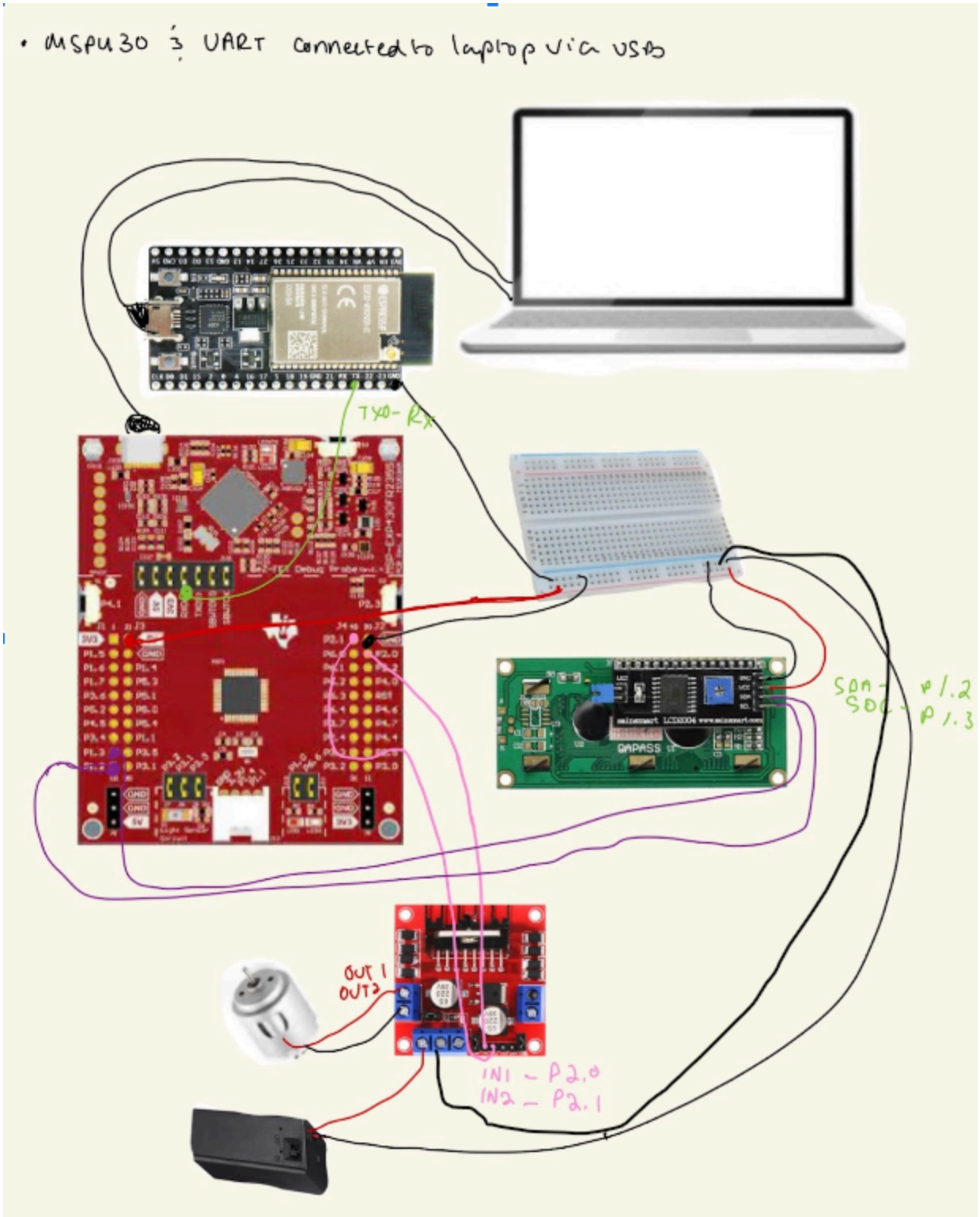
Project description:

The project involves designing a system that leverages website for user input (CS&CE) to control a PWM output signal for a 12 VDC electric motor. This system allows users to adjust the motor's speed via Wifi. The app features a user-friendly interface displaying the motor speed as a percentage, ranging from 0 to 100%. By sliding a control or entering a value on the website, users can seamlessly increase or decrease the motor's speed in real time. The wifi connection ensures wireless and convenient operation, making the system suitable for various applications where precise motor speed control is essential.

Wiring Diagram:

ESP and MSP both get their power from the laptop.
ESP, MSP, LCD, H-bridge, battery all share common ground.

• MSPM30 ; UART connected to laptop via USB



Video: https://youtu.be/KqjSwryjNBI

Pseudocode:

(MSP)

```
// Include necessary libraries and header files

// Define global variables
// Buffer to store serial data

// Main function
int main(void) {
    // Stop watchdog timer

    // Configure UART for communication
    // Configure motor control pins and direction
    // Configure timer for motor control
    // Initialize LCD and display initial duty cycle

    // Main loop
    while (1) {
        // Check if serial input is complete
        if (finished == 1) {
            // Process and display duty cycle
            // Adjust motor control based on duty cycle
            // Special case for 100% duty cycle
            // Reset counting variables for next input
        }
    }
}

// Interrupt Service Routine for Timer CCR0
    // Turn on motor if duty cycle is not 0
}

// Interrupt Service Routine for Timer CCR1
    // Turn off motor
}

// Interrupt Service Routine for UART
__interrupt void USCI_A1_ISR(void) {
    // Read and process incoming UART data
    // if it is just a character, add it to buffer
```

```
// if it is newline char, then we reached end of serial data, so signal it to main loop
}
```

(ESP)

```
// Include necessary libraries
// Define pins and baudrate
// Wi-Fi credentials
// HTML content for the web page

// Current slider value

// choose UART1
// AsyncWebServer setup

void setup(){
  MotorSpeedTransmit.begin(BAUDRATE, SERIAL_8N1, RxPin, TxPin);  // Initialize UART
communication
  Serial.begin(BAUDRATE);  // Initialize Serial for debugging

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  // Print ESP8266 IP address once connected
  // Configure routes for web server

  server.on("/slider", HTTP_GET, [] (AsyncWebServerRequest *request) {
    // get input number
    // Ensure that value is always 3 digits
    // Send sliderValue over UART character by character with delays in between
    // Add newline character to mark end of serial data
  });

  // Start server
  server.begin();
}
```

Code:

(MSP)

```
#include <msp430.h>
#include <stdio.h>
#include "LiquidCrystal_I2C.h"
```

```
char uartBuffer[8] = {0}; // buffer to store serial data
unsigned int idx = 0;  // index for the UART buffer
unsigned long input = 0;       // to store the final dutyCycle
unsigned char finished = 0;   // set to 1 once we reach \n, which marks end of serial data
unsigned int dutyCycle = 0;


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

   // UART Configuration
   UCA1CTLW0 |= UCSWRST;                     // Put eUSCI in reset
   UCA1CTLW0 |= UCSSEL__SMCLK;               // Clock source SMCLK
   UCA1BR0 = 8;                              // 1000000/115200 = 8.68
   UCA1MCTLW = 0xD600;                       // 1000000/115200 -
INT(1000000/115200)=0.68
   P4SEL1 &= ~BIT2;                          // Configure UART pins
   P4SEL0 |= BIT2;
   P4SEL1 &= ~BIT3;                          // Configure UART pins
   P4SEL0 |= BIT3;

   PM5CTL0 &= ~LOCKLPM5;                     // Disable the GPIO power-on default
high-impedance mode
   UCA1CTLW0 &= ~UCSWRST;                    // Initialize eUSCI
   UCA1IE |= UCRXIE;                         // Enable USCI_A1 RX interrupt
   __enable_interrupt();                     // Enable global interrupt

   // Motor config
   P2DIR |= BIT0; // using P2.0 to send signal to H-Bridge
   P2OUT &= ~BIT0; // ensure the motor is off initially
   P2DIR |= BIT1;  // using P2.1 for motor direction

   // Timer config
   TB3CTL = TBSSEL__ACLK | MC__UP | TBCLR; // ACLK, up mode, clear TBR
   TB3CCR0 = 655; // period for motor
   TB3CCR1 = 0;   // 0% duty cycle
```

```
    // interrupts
    TB3CCTL0 |= CCIE; // enable interrupt
    TB3CCTL1 |= CCIE; // enable interrupt
    TB3CCTL0 &= ~CCIFG; // lower flag
    TB3CCTL1 &= ~CCIFG; // lower flag

    // LCD Configuration
    I2C_Init(0x27);
    LCD_Setup();               // Initialize the LCD
    LCD_SetCursor(4, 0);       // set initial cursor to start of screen
    LCD_ClearDisplay();        // Clear Display
    LCD_Write("Duty Cycle: ");
    LCD_WriteNum(dutyCycle);

    while (1) {
        if (finished == 1) {
            // Process the input value
            LCD_ClearDisplay();
            LCD_Write("Duty Cycle: ");
            LCD_WriteNum(dutyCycle);
            if (dutyCycle == 100) {
                // Special case for 100% duty cycle
                P2OUT |= BIT0; // Keep motor on continuously
                TB3CCTL0 &= ~CCIE; // Disable Timer interrupt
                TB3CCTL1 &= ~CCIE; // Disable Timer interrupt
            } else {
                input = (dutyCycle * 655) / 100; // Calculation for Duty Cycle
                TB3CCR1 = input; // Duty Cycle to Motor
                TB3CCTL0 |= CCIE; // Enable Timer interrupt
                TB3CCTL1 |= CCIE; // Enable Timer interrupt
            }
            // Reset
            idx = 0;
            finished = 0;
        }
    }
}

/*****************************
*           ISR           *
```

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

#pragma vector = TIMER3_B0_VECTOR
__interrupt void ISR_TB3_CCR0(void) {
   if (TB3CCR1 != 0) {
      P2OUT |= BIT0;  // turn on motor
   }
   TB3CCTL0 &= ~CCIFG; // lower flag
}

#pragma vector = TIMER3_B1_VECTOR
__interrupt void ISR_TB3_CCR1(void) {
   P2OUT &= ~BIT0; // turns motor off
   TB3CCTL1 &= ~CCIFG; // lower flag
}

#pragma vector = USCI_A1_VECTOR
__interrupt void USCI_A1_ISR(void)
{
   unsigned char rxData = UCA1RXBUF;

   if (rxData == '\n') {
      uartBuffer[idx] = '\0';  // Null-terminate the string
      dutyCycle = atoi(uartBuffer);       // Convert the string to an integer
      finished = 1;                  // Set the finished flag
      idx = 0;                // Reset buffer index for next input
   } else if (rxData >= '0' && rxData <= '9' && idx < sizeof(uartBuffer) - 1) {
      uartBuffer[idx++] = rxData;  // Store received character in buffer
   }
}
```

(ESP):

```
// Load Wi-Fi library
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <HardwareSerial.h>
```

```
#define RxPin         16
#define TxPin         17
#define BAUDRATE      115200
#define SER_BUF_SIZE  1024

const char* ssid = "TheInterWeb";
const char* password = "rewardbasket657";

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>ESP Web Server</title>
  <style>
    html {font-family: Arial; display: inline-block; text-align: center;}
    h2 {font-size: 2rem;}
    p {font-size: 1.7rem;}
    body {max-width: 400px; margin:0px auto; padding-bottom: 25px;}
    .slider { -webkit-appearance: none; margin: 14px; padding: 1px; width: 360px; height:
25px; background: #FF5C35;
      outline: none;}
    .slider::-webkit-slider-thumb {-webkit-appearance: none; appearance: none; width:
22px; height: 22px; background: #FFFFFF; cursor: pointer;}
    .slider::-moz-range-thumb { width: 22px; height: 22px; background: #FFFFFF; cursor:
pointer; }
  </style>
</head>
<body>
  <h2>ESP Web Server</h2>
  <p>Slide to control motor speed</p>
  <p><input type="range" onchange="updateSliderPWM(this)" id="pwmSlider" min="0"
max="100" value="%SLIDERVALUE%" step="1" class="slider"></p>
  <p><span id="textSliderValue">%SLIDERVALUE%</span></p>
<script>
function updateSliderPWM(element) {
  var slider = document.getElementById("pwmSlider");
  var sliderValue = slider.value;
  document.getElementById("textSliderValue").innerHTML = sliderValue;
  slider.disabled = true;
```

```
  var xhr = new XMLHttpRequest();
  xhr.open("GET", "/slider?value="+sliderValue, true);
  xhr.send();
  // Re-enable the slider after 5 seconds
  setTimeout(function() {
    slider.disabled = false;
  }, 5000);
}
</script>
</body>
</html>
)rawliteral";

String sliderValue = "0";
const char* PARAM_INPUT = "value";

HardwareSerial MotorSpeedTransmit(1);       // Assign UART1
// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

// Replaces placeholder with button section in your web page
String processor(const String& var){
  if (var == "SLIDERVALUE"){
    return sliderValue;
  }
  return String();
}

void setup(){
  MotorSpeedTransmit.begin(BAUDRATE, SERIAL_8N1, RxPin, TxPin);
  // Serial port for debugging purposes
  Serial.begin(BAUDRATE);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }
```

```
// Print ESP Local IP Address
Serial.println(WiFi.localIP());

// Route for root / web page
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/html", index_html, processor);
});

// Send a GET request to <ESP_IP>/slider?value=<inputMessage>
server.on("/slider", HTTP_GET, [] (AsyncWebServerRequest *request) {
  String inputMessage;
  // GET input1 value on <ESP_IP>/slider?value=<inputMessage>
  if (request->hasParam(PARAM_INPUT)) {
    inputMessage = request->getParam(PARAM_INPUT)->value();
    sliderValue = inputMessage;
  }
  else {
    inputMessage = "No message sent";
  }

  // ensure input is always 3 digits
  if (sliderValue.length() == 1) {
    sliderValue = "00" + sliderValue;
  } else if (sliderValue.length() == 2) {
    sliderValue = "0" + sliderValue;
  }

  // send it over UART
  for (int i = 0; i < sliderValue.length(); i++) {
    char valueChar = sliderValue[i];
    MotorSpeedTransmit.write(valueChar);  // Send each character over UART
    Serial.write(valueChar);              // Print each character for debug
    delay(1000);
  }
  MotorSpeedTransmit.write('\n');  // Add newline to mark the end
  Serial.println();

  request->send(200, "text/plain", "OK");
});
```

```
  // Start server
  server.begin();
}
// because it is async web server, we can keep it empty
void loop() {
}
```