TITLE AND DATE

Document name: PWM Motor Driver
Document reference: *Davenport, Robert*.cmpe311.fall25.*Lab-FreeRTOS-Motor-Driver*
Date of publication: 12*/9/2025*

LEAD ENGINEER: *Robert Davenport, UMBC CSEE, Catonsville, Maryland, USA*

STAKEHOLDERS:

Prof Kidd, Instructor, University of Maryland Baltimore County, Baltimore, Maryland, USA
MD Safwan Zaman, TA, University of Maryland Baltimore County, Baltimore, Maryland, USA

HIGH-LEVEL DESCRIPTION: This document outlines requirements for project #5. These include customer, technical, and testing requirements

DESCRIPTION: *Design and implement an embedded system using an Arduino UNO compatible development board. Perform identical to Project #4 outlined in Davenport, Robert*.cmpe311.fall25.*Lab-FreeRTOS-Motor-Driver*.pdf*. The system MUST use a task manager based upon the FreeRTOS library using various function calls to handle input and output.*

RESULT SUMMARY: The project was successful, and the design proposed meets all requirements outlined

REFERENCES AND GLOSSARY

REFERENCES:

- projectAddDutyCycle.pdf – *Original project outline*
- *Davenport, Robert*.cmpe311.fall25.*Lab-FreeRTOS-Motor-Driver*.pdf– *Last project on which this project was based upon*
- https://forum.arduino.cc/t/using-freertos-along-with-arduino/1254126/2 - using FreeRTOS
- Arduino UNO R3 Product Reference Manual SKU A000066 12/03/2024
- https://www.alldatasheet.com/datasheet-pdf/pdf/22400/STMICROELECTRONICS/IRF740.html - IRF740 Mosfet Datasheet
- driveMotorFromDigitalPin.pdf – Sample circuit for simple PWM motor driver and calculations

DEFINITIONS AND ABBREVIATIONS:

Arduino – an Italian open-source hardware and software company; also refers to a development board created by the company
arduino.h – header for a library of convenience functions specific to the Arduino development platform
AVR – A family of microcontrollers, originally developed by Atmel, and currently owned by Microchip Technology

ELEGOO – A Chinese company that develops and markets 3D printers and accessories
IDE – Integrated Development Environment
gcc – front end for the GNU Compiler Collection
Github – A widely used distributed SVC (Software Version Control) system
LED – Light Emitting Diode
Msec – Millisecond time interval used by IDE

REQUIREMENTS

CONVENTIONS:

Must, shall or will – your design must satisfy the requirement
May – your design may satisfy the requirement but doesn't have to
Informative – the intent of the following description is to make the requirement more understandable
All customer requirements are started with "C.#".
All high-level requirements are started with "HL.#".
All testing/validation requirements ar5e started with "T.#"

CUSTOMER REQUIREMENTS:

C1. The User must be able to adjust the duty cycle of the motor with a button push
C2. The User must be able to update the duty cycle with subsequent button pushes
C3. The System must run upon an Arduino Uno R3 compatible development board

HIGH-LEVEL TECHNICAL REQUIREMENTS:

HL.1 The System must a limiting resistor and a pull-down resistor
HL.2 The System must include the IRF740 Mosfet, flyback diode as well as a provided motor
HL.3 The System must use a standard Arduino compatible development board (e.g. the provided ELEGOO Uno R3)
HL.4 The System must interpret user button inputs asynchronously, and send updates using the PWM pin on the Arduino (or equivalent board)
HL.5 The User must be able to adjust the duty cycle of the motor with subsequent button presses
HL.6 The set duty cycle must stay constant until the user input is detected
HL7 The System must use a FreeRTOS task manager with function calls to handle all I/O

TESTING AND VALIDATION REQUIREMENTS:

T.1 Motor RPM is at 0 initially
T.2 Motor runs 2/8 duty cycle with 1 button push
    T.2.1 Subsequent button pushes increase duty cycle 2/8, up to its maximum duty cycle
T.3 After Motor reaches maximum duty cycle, subsequent button pushes will decrease duty cycle, back down to 0
T.4 System must be able to increase and decrease duty cycle automatically as user continues to push button

DESIGN AND TESTING

DESIGN

Materials:

- Arduino Uno R3 or similar development board
- Arduino IDE 2.3.3 or newer
- Cables & Breadboard
- 1 flyback diode
- 1 IRF740 Mosfet
- An DC electric motor
- 2 resistors, 1 limiting and one pull down
- 9V battery

Schematic: Located under Appendix A.

Design code: Located under Appendix B.

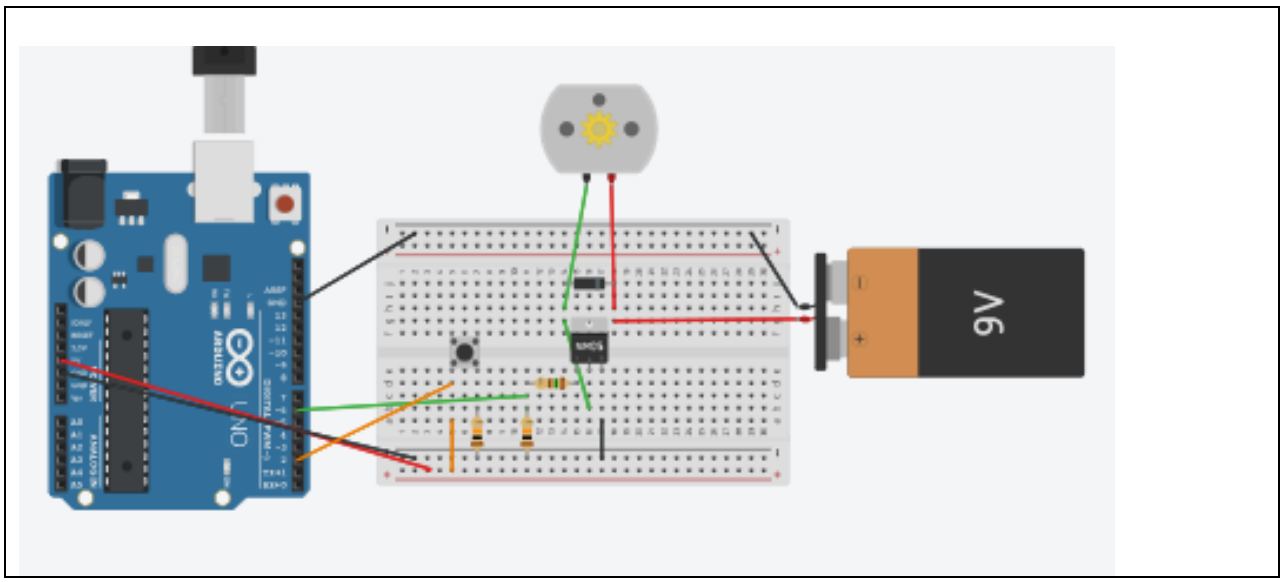
Solutions: Located under Appendix C.

TESTING RESULTS:

Test # (See Testing and Validation requirements)	Result (See video of test)
T.1	Passed
T.2	Passed
T.2.1	Passed
T.3	Passed
T.4	Passed

Appendix A. Schematic

Testbed Schematic. PWM pin at 6, Button input at pin 2.



Appendix B. Design Code

```
#include <Arduino_FreeRTOS.h>

// Pin definitions
const int button = 2;
const int pwm = 6;

// Button state tracking variables
int buttonState = LOW;
int lastButtonState = LOW;

// Motor speed state variables
int state_count = 0;
const int maxDuty = 255;
bool decend = false;

// Debounce timing variables
unsigned long lastDebounceTime = 0;
unsigned long debounceDelay = 50;

// Task function prototypes
void TaskButtonDebounce(void *pvParameters);
void TaskMotorControl(void *pvParameters);

void setup() {
  Serial.begin(9600); // for debugging

  pinMode(pwm, OUTPUT);
  pinMode(button, INPUT);
  analogWrite(pwm, 0);

  // FreeRTOS tasks
  xTaskCreate(
```

```
    TaskButtonDebounce,           // Task function
    "ButtonDebounce",             // Task name
    128,                          // Stack size (words)
    NULL,                         // Task parameters
    1,                            // Task priority
    NULL                          // Task handle
  );

  xTaskCreate(
    TaskMotorControl,             // Task function
    "MotorControl",               // Task name
    128,                          // Stack size (words)
    NULL,                         // Task parameters
    1,                            // Task priority
    NULL                          // Task handle
  );

  // Start the FreeRTOS scheduler
  vTaskStartScheduler();
}

void loop() {
  // Empty loop - FreeRTOS scheduler takes over
}

// Task: Button debounce and state management
void TaskButtonDebounce(void *pvParameters) {
  (void) pvParameters;  // Unused parameter

  // Task loop runs indefinitely
  for (;;) {
    // Read current button state
    int buttonRead = digitalRead(button);

    // Debug output to serial monitor
    Serial.print("Button Read: ");
    Serial.print(buttonRead);
    Serial.print(" | Button State: ");
    Serial.print(buttonState);
    Serial.print(" | State Count: ");
    Serial.println(state_count);

    // Check if button reading has changed
    if (buttonRead != lastButtonState) {
      lastDebounceTime = millis();  // Reset debounce timer
    }

    // Check if enough time has passed since last state change
    if ((millis() - lastDebounceTime) > debounceDelay) {
      // Check if the stable reading is different from current button
state
      if (buttonRead != buttonState) {
        buttonState = buttonRead;  // Update button state to new stable
value
```

```
        // update motor speed on button PRESS
        if (buttonState == HIGH) {
          //increase speed from 0 to 8
          if (decend == false) {
            state_count = state_count + 2;  // Increment speed state by
2
            if (state_count >= 8) {         // Check if maximum reached
              state_count = 8;              // Cap at maximum
              decend = true;                // Switch to descending mode
            }
          }
          // decrease speed from 8 to 0
          else {
            state_count = state_count - 2;  // Decrement speed state by
2
            if (state_count <= 0) {         // Check if minimum reached
              state_count = 0;              // Cap at minimum
              decend = false;               // Switch back to ascending
mode
            }
          }
        }
      }
    }

    // Save current reading for next iteration
    lastButtonState = buttonRead;

    vTaskDelay(100 / portTICK_PERIOD_MS);  // 100ms delay
  }
}

// Task: Motor PWM control
void TaskMotorControl(void *pvParameters) {
  (void) pvParameters;  // Unused parameter

  // Task loop runs indefinitely
  for (;;) {
    // Calculate PWM duty cycle based on state_count
    // Maps state_count (0-8) to duty cycle (0-255)
    int duty = (maxDuty * state_count) / 8;

    // Apply the calculated duty cycle to the motor
    analogWrite(pwm, duty);

    // FreeRTOS delay - yields to other tasks
    vTaskDelay(10 / portTICK_PERIOD_MS);  // 10ms update rate
  }
}
```

END OF DOCUMENT