TITLE AND DATE

Document name: Project#4 FreeRTOS Project: Lab Report
Document reference: kiddKid.cmpe311.fall25.project# PWM
Date of publication: December 9, 2025
GITHUB REPO LINK: LINK
VIDEO LINK: VIDEO LIN K

LEAD ENGINEER: Shariq Moghees, UMBC

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 is the project document for Project#4 of the CMPE311 class. It contains customer, technical, and testing requirements as well as the design and results of the validation testing.

DESCRIPTION: This design implements an embedded multitasking system on the Arduino Uno R3 using the FreeRTOS real-time operating system. The system controls LED blink intervals and motor speed concurrently by running each function as an independent FreeRTOS task. LED behavior is controlled through user-defined blink intervals, while motor output is driven using a Pulse Width Modulation (PWM) signal whose duty cycle is adjusted via button input.

FreeRTOS replaces the Cyclic Executive (CE) structure used in Project 2 by providing preemptive task scheduling, deterministic timing, and clean task separation. Dedicated tasks handle LED blinking, PWM generation, and button input processing. The PWM duty cycle increases or decreases in fixed steps with each button press, cycling smoothly between 0% and 100%. The LED blink intervals can be updated at runtime through serial input without interrupting other system behaviors.

This project demonstrates proper use of real-time scheduling, inter-task communication through shared global variables, and modular task-based design. The document includes the overall system logic, task structure, hardware validation, and demonstration of FreeRTOS-based concurrency and control.

RESULT SUMMARY: The project was a success, the embedded system design meeting all testing and high-level requirements.

REFERENCES AND GLOSSARY

<to eliminate any ambiguity, include any specific terms or entities, any unique words, any clarification, and any abbreviations>

REFERENCES:

- projectTaskMgmtFreeRTOS.pdf– The definition of the project this document addresses
- Arduino UNO R3 Product Reference Manual SKU A000066, 12/03/2024
- Async Programming in Arduino: Unleashing the Power of Non-Blocking Code, Mahdi Valizadeh, medium.com, 4/8/2024

***DEFINITIONS*:**

"The User" – The person operating (not programming) the embedded system
"The System" – The embedded system being operated by The User
"The Customer" – The person(s) paying for the embedded system being designed and built
"The Developer" – The person(s) designing and building the System
"The Evaluator" – The person(s) that determine whether or not The System satisfies The Customer-requirements.
"The Customer-requirements" – The requirements defined by The Customer as satisfying The Contract.
"The Requirements" – The System's high-level technical requirements derived from The Customer-requirements.
"The Educational-constraints" – Requirements imposed by the instructor unrelated to the embedded system that allow The System to be evaluated.
"The Company" – The organization The Customer has contracted with to build The System.
"The Contract" – The business document that legally binds The Company to provide some service or product to The Customer.
"serial-monitor" – The serial port used by the Arduino IDE to communicate with The User.
"The Reference-platform" – The configuration of The System used by The Developer to test and validate The System. For this class, The System is the Arduino compatible ELEGOO Uno R3 development board.

ACRONYMS 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
PWM- Pulse Width Modulation

REQUIREMENTS

<customer, high-level and testing 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: *<list the customer requirements which I gave you>*

C1. The user must be able to vary the PWM duty cycle in discrete steps using a button input.
C2. The PWM signal must control an LED's brightness proportional to duty cycle (0–100%).
C3. The LED brightness must reflect changes immediately upon button press.
C4. The system must run on an Arduino Uno R3-compatible platform.
C5. The button input must be debounced in software to ensure reliable transitions.
C6. The Asynchronous LED blinking from Project #2 must also work at the same time
C7. The program must operate on a FreeRTOS task scheduling system, replacing the Cyclic Executive from the previous project

HIGH-LEVEL TECHNICAL REQUIREMENTS: *<list the derived high-level technical requirements>*

HL.1. The system must use an interrupt-based PWM generator.
HL.1.1. The PWM duty cycle must be adjustable in five steps (0%, 20%, 40%, 60%, 80%, 100%).
HL.2. The system must use a Free RTOS task scheduling system to manage independent tasks, including PWM updates and button processing.
HL.3. Logic must be handled asynchronously by the CE task loop.
HL.4. The system must run deterministically and handle debouncing without blocking execution.
HL.5. The PWM output must be compatible with hardware-based control of a DC motor via MOSFET from the prior project.

DESIGN:

The system design uses one LED connected to a PWM-capable digital pin (D9).
A pushbutton is connected to an external interrupt pin (D2) configured as active LOW.
An ISR captures button events and sets a flag that is processed in the taskButton() function to update brightness level. A separate taskPWM() handles PWM signal updates, and a taskStatus() prints current duty cycle status over serial. All tasks are scheduled through a Cyclic Executive array and execute asynchronously in a round-robin fashion. This modular structure allows Part III (motor control via MOSFET) to use the same embedded logic with only hardware wiring changes.The design fulfills all the following high-level requirements:

- *Independent LED control (HL.1.1).*
- *Non-blocking, asynchronous execution (HL.1.2).*
- *Serial Monitor-based interface (HL.2).*
- *Runs on Arduino Uno R3 platform (HL.3).*
- *Task Management via Cyclic Executive (ER.1)*

PART 2: DRIVER CIRCUIT DESIGN CALCULATION AND JUSTIFICATON:

SEE APPENDIX A TO SEE THE SKETCH OF THE DRIVER CIRCUIT SCHEMATIC

CALCULATION OF R1:

Motor: 6 Vdc
Max Current of Arduino Pin: 20 mA

R1 = V/A = 6 V / 20mA **= 300 Ohms, Choose 460 Ohm Resistor for R1 (330 Ohm resistors were not provided in class)**

CALCULATION OF R2:

Let R2 be 10x R1: 300 X 10 = **3k Ohms, Choose 10k Ohm Resistor**

R2 is a pull-down resistor, it must be a great deal larger than R1 in case something fails

JUSTIFICATION OF FLYBACK DIODE:

The DC Motor acts as an inductive load, meaning when current is immediately cut off, the inductor-like behavior will cause a **large voltage spike** to resist the change in current. This can potentially damage components that are not meant to handle that level of voltage. The flyback diode allows the large voltage being pushed back by the motor to be

looped around and fed back into the motor **to create a safe outlet for the extra voltage to be dissipated,** instead of through out the circuit and frying components.

DESIGN PRE-REQUISITES: *<include any equipment or other things you are required to build your design upon such as the configuration of the development platform>*

1. ELEGOO Arduino Uno R3 clone
2. Arduino IDE 2.3.3 or better
3. Two External LEDs for Asynchronous blinking from Project 2
4. Two 330 Ohm resistors for each LED from Project 2
5. One 460 Ohm Resistor for R1 and two 10k Ohm resistors for R2 and the push button
6. One Diode
7. One 12V battery Pack with snap-on connector
8. One 6 Vdc DC Motor
9. One N-Channel MOSFET transistor
10. One Push-Button
11. One breadboard
12. Jumper wires for creating connections from Arduino board pins 2 & 6, to the resistor. and finally LED.

DEVELOPMENT PLATFORM: *<include any equipment, SW, etc necessary to reproduce your design environment>*

1. See DESIGN PRE-REQUISITES above

ANY ADDITIONAL DESIGN CONSIDERATIONS: *<include anything else needed for someone to understand and reproduce your design>*

1. This motor driver circuit is implemented on top of the previous project, which contains Asynchronously blinking LED lights which can be changed at will separately via the Command console.
2. The program is developed using a library called FreeRTOS by Richard Berry.

See Appendix A for the **Design Code**

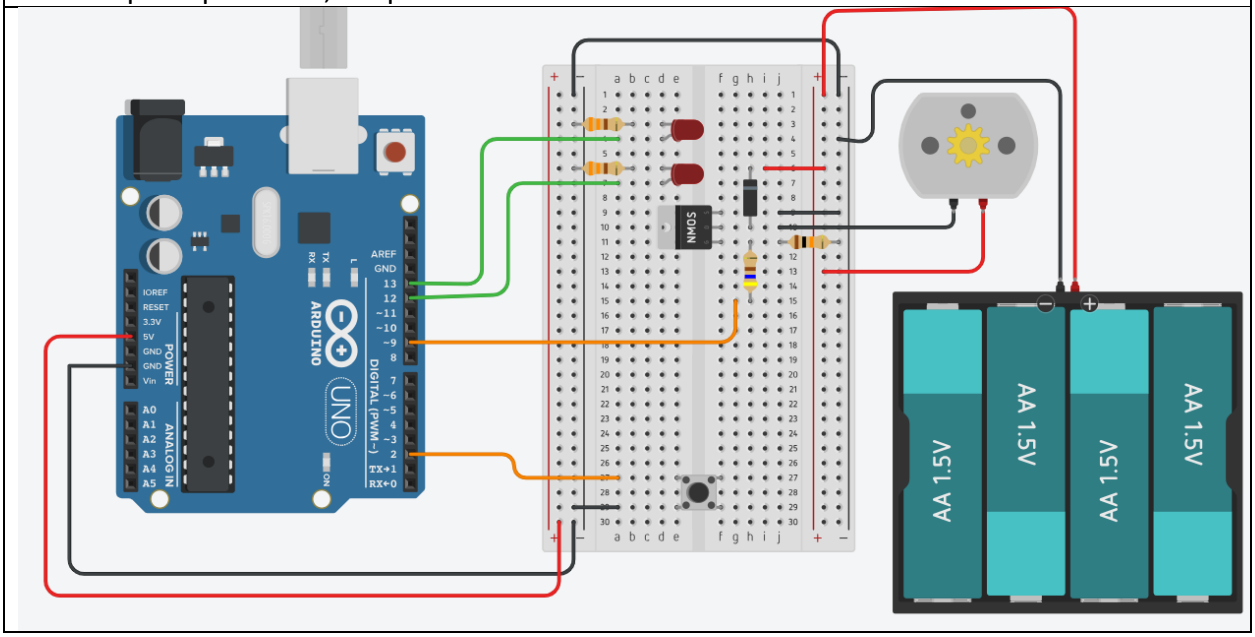
TESTING AND VALIDATION

*The testbed setup is shown in Figure 1, with two LEDs connected to pins 12 and 13 for the asynchronous blinking of Project 2. The push button is connected to ARDUINO pin 2. The Driver circuit's **PWM signal is driven by ARDUINO pin 9**. The system was tested by entering commands into the Serial Monitor as well as pressing the push button. Each test involved setting a new interval for one LED while observing that the other LED's blink rate remained unaffected. While pressing the push button will see the PWM signal instensity increment/decrement and wither change the instensity of an LED (part 1) or the speed of the motor (part 3)*

The required dialog was reproduced successfully (Table 1), and repeated trials were used to verify stability (Table 2). Validation confirmed that inputs in milliseconds were accepted, intervals updated correctly, and LEDs blinked asynchronously.

DESCRIPTION: The tests that have to be performed and validated are shown in Table xxx. These tests were performed on the testbed shown Figure 1. Table 1 shows a dialog that must be successfully performed by the embedded system. The results of testing are shown in Table 3. When necessary, a video of the test is provided along with this report.

Figure 1. Testbed Setup. LEDs connected to digital pins 12 & 13. Button connected to Pin 2. PWM signal for motor driver circuit connected to pin 9. The pushbutton uses the arduino's internal pull-up resistor, as specified within the code.



TESTING PLATFORM: *<include any equipment, SW, etc necessary to reproduce your design environment>*

1. ELEGOO Arduino Uno R3 clone
2. Arduino IDE 2.3.3 or better
3. Power: Testing platform powered through USB cable, LEDs connected directly through Digital Outputs
4. One N-channel MOSFET transistor
5. One Diode
6. Two external LED
7. Two 330 Ohm resistors, One 460 Ohm , and two 10k Ohm resistors
8. One DC motor
9. Jumper wires for connections

Table 1. Required Test Dialog (***blue*** are the user inputs).

Serial port I/O	**Notes**
"Format: <LED#> <interval_ms>"	System initialized. LED/motor starts at 0% duty cycle (off). Awaiting first button press. Asynchronous blinking LEDs start at a generic blink rate. Specifies how to input to set an LEDs blink rate.
User presses button once "Brightness: 1"	LED brightness/Motor Speed increases to 20%
(User continues pressing button) "Brightness: 2" "Brightness: 3" "Brightness: 4" "Brightness: 5" "Brightness: 4" "Brightness: 3" .. and so on until 0	LED brightness steps up to amx and then back down through 80%, 60%, 40%, 20%, 0%. Cycle reverses direction at each limit.
User Enters "1 200" into Serial monitor command line "LED1 rate updated to 200 ms"	LED 1 changes its interval asynchronously to blink every 200 nanoseconds

TESTING AND VALIDATION REQUIREMENTS: *<include all tests using appropriate language, e.g. must, so that there is no ambiguity>*

Table 2. Testing and Validation Requirements

T.0 All testing and validation must be done on the testbed shown in Figure 1 (Arduino UNO + MOSFET + LED + Button).

T.1 System must initialize with LED/Motor off (PWM = 0%).
T.1A The Serial Monitor must display initial status text on startup ("Project 3 PWM Driver…").T.1.1 Setting of the blink rate (and LED#) must be through the IDE serial monitor

T.2 After reaching 100% duty cycle, the next press must reverse direction and begin decreasing brightness/speed.

T.3 After reaching 0%, the system must reverse direction again and increase brightness/Motor on the next press

T.4 The Serial Monitor output must match the current brightness/speed level and duty cycle after each press.
T.5. PWM output must function on hardware pin 6 and be verified by either LED brightness or motor speed response.

T.5 System must change its blink rate of either one of the two LEDs based on the serial monitor input given by the user

TESTING RESULTS: *<include any relevant tests, data, videos, etc that are used to confirm the tests>*

All tests in Table 2 were executed on the ELEGOO Uno R3 clone. Observations confirmed that the blink rates matched user input, changes were isolated to the correct LED, and no blocking occurred when handling Serial input. Videos of select tests were recorded for documentation. Results are summarized in Table 3, all marked as satisfied.

Table 3. Results of tests

Test performed	Results
T.1.1	Satisfied. See video of test.
T.1	Satisfied
T.2	Satisfied. See video of test.
T.3	Satisfied
T.4	Satisfied
T.4.1	Satisfied. See video of test.
T.5.1	Satisfied. See video of test.
T.5.2	Satisfied. See video of test.

APPENDIX A. DESIGN CODE

```
//Filename: 311_FreeRTOS_project.ino
//Author: Shariq Moghees (OL41895)
//Date: 12/9/25
//Class: CMPE 311 Dr. Kidd Fall 2025
//Desc: A program that takes the Asynchronous LED + motor driver circuit from the
last project and converts it into a
// FreeRTOS task scheduling system using an imported Library instead of using a
Cyclic Executive.
//-------------------------------------------------------------------------------
-------------------------------------

#include <Arduino_FreeRTOS.h>
#include "task.h"

#define PIN_LED1   13   //led 1
#define PIN_LED2   12   //led 2
#define PIN_PWM     9   //PWM signal will be coming from pin 9
#define PIN_BUTTON  2   // Using internal pull-up. active LOW push button

// global variables for LED and PWM dirver circuit tasks

volatile uint32_t intervalLed[3] = {0, 1000, 500};  // default intervals
volatile int ledOn1 = 0;
volatile int ledOn2 = 0;
```

```
volatile int brightness = 0;     // 0-5
volatile bool decreasing = false;

//TASK DECLARATIONS
void taskLED1(void *pv);
void taskLED2(void *pv);
void taskPWM(void *pv);
void taskButton(void *pv);
void taskSerial(void *pv);

//SETUP
void setup() {

  Serial.begin(9600);

  pinMode(PIN_LED1, OUTPUT);
  pinMode(PIN_LED2, OUTPUT);
  pinMode(PIN_PWM, OUTPUT);
  pinMode(PIN_BUTTON, INPUT_PULLUP); //internal arduino pull-up resistor for button

  digitalWrite(PIN_LED1, LOW);
  digitalWrite(PIN_LED2, LOW);
  analogWrite(PIN_PWM, 0); //we initialize the Motor to be off initially (level 0)

  Serial.println("FreeRTOS Integrated System Ready");
  Serial.println("Format: <LED#> <interval_ms>");

  //creat the tasks using FreeRTOS library syntax
  xTaskCreate(taskLED1,   "LED1",   110, NULL, 1, NULL);
  xTaskCreate(taskLED2,   "LED2",   110, NULL, 1, NULL);
  xTaskCreate(taskPWM,    "PWM",    90,  NULL, 1, NULL);
  xTaskCreate(taskButton, "BTN",    90,  NULL, 2, NULL); //set the serial and button
input tasks at a higher priority
  xTaskCreate(taskSerial, "SER",    180, NULL, 2, NULL);

  vTaskStartScheduler();
}

void loop() {} //loop dosnt need anything in it becasue FreeRTOS does all our task
scheduling for us

//LED TASKS

void taskLED1(void *pv) {

  TickType_t lastWake = xTaskGetTickCount();
  pinMode(PIN_LED1, OUTPUT);

  for (;;) {
    ledOn1 = !ledOn1;
    digitalWrite(PIN_LED1, ledOn1);

    // EXACT periodic timing
    vTaskDelayUntil(&lastWake, intervalLed[1] / portTICK_PERIOD_MS);
  }
}

void taskLED2(void *pv) {

  TickType_t lastWake = xTaskGetTickCount();
  pinMode(PIN_LED2, OUTPUT);

  for (;;) {
```

```
    ledOn2 = !ledOn2;
    digitalWrite(PIN_LED2, ledOn2);

    vTaskDelayUntil(&lastWake, intervalLed[2] / portTICK_PERIOD_MS);
  }
}

//PWM TASK
void taskPWM(void *pv) {
  pinMode(PIN_PWM, OUTPUT);

  for (;;) {
    analogWrite(PIN_PWM, brightness * 51);  // 0-255

    vTaskDelay(20 / portTICK_PERIOD_MS);
  }
}

//PUSH BUTTON INPUT TASK

void taskButton(void *pv) {

  for (;;) {

    if (digitalRead(PIN_BUTTON) == LOW) {  // pressed (active LOW)

      if (!decreasing) { //step up
        brightness++;
        if (brightness >= 5) {
          brightness = 5;
          decreasing = true;
        }
      } else {    //step down
        brightness--;
        if (brightness <= 0) {
          brightness = 0;
          decreasing = false;
        }
      }

      Serial.print("Brightness: ");
      Serial.println(brightness);

      vTaskDelay(200 / portTICK_PERIOD_MS); // debounce/edge detect for the push
button
    }

    vTaskDelay(30 / portTICK_PERIOD_MS);
  }
}

//SERIAL INPUT TASK (for changing the asynchronous LED blinking rate)
//I'm using chars intead of strings here to reduce the amount of memory used. I was
running into a memory issue with stack space and RTOS tasks.
void taskSerial(void *pv) {

  char buffer[16];
  uint8_t idx = 0;

  for (;;) {

    while (Serial.available()) {
      char c = Serial.read();
```

```
      if (c == '\n' || c == '\r') {
        buffer[idx] = 0;  // null-terminate string

        int led = atoi(strtok(buffer, " "));
        int val = atoi(strtok(NULL, " "));

        if (led >= 1 && led <= 2 && val > 0) {
          intervalLed[led] = val;

          Serial.print("LED");
          Serial.print(led);
          Serial.print(" rate updated to ");
          Serial.print(val);
          Serial.println(" ms");
        }

        idx = 0;
      }
      else if (idx < 15) {
        buffer[idx++] = c;
      }
    }

    vTaskDelay(20 / portTICK_PERIOD_MS);
  }
}
```

END OF DOCUMENT