Course: CMPE311-Project 3
DATE: Dec 1, 2025
TO: Professor Kidd
FROM: Abhinna Das
SUBJECT: Duty cycle project

Part 1

Part I: (PWM) Build the embedded system to change the motor RPM as described above. For this part, instead of driving an actual motor, use the motor digital control signal to control an LED's intensity instead of the speed of the motor.

Part 2

Part II: (Driver) The embedded system (i.e. the Arduino) is not needed for this part. Build and test a MOSFET-based motor driver circuit. See Table A for the DC motor's specification. Test the circuit by connecting the motor and setting the control signal input voltage to 0Vdc (0 RPM) and two 1.5Vdc batteries in series. Use one to verify that the motor runs. Make sure you place a resistor in series to limit the current draw of the motor.

Part 3

(Integration) Connect your embedded system to the driver circuit and demonstrate its operation.
Rubric (Demonstration of successful operation): (1) Pressing the button cycles the speed (RPMs) of the motor and fan from off to maximum and back down. (2) This operation operates asynchronously with the blinking LEDs from Project #2.

Question answering

PROBLEM#1: Calculate the value of the resistor R1 to limit the current through the gate to the maximum acceptable for an Arduino Uno digital pin. (Note that R1 only comes into play if there the MOSFET has failed resulting in a short.)

Answer: The max current output for the Arduino pin is 40 mA howeve,r it is recommended to only output 20 mA as such the resistor to limit the current to the gate needs to be at least 40mA as such using ohms law we have V = IR since where V = 5 V and I = 40 mA for the max and 20 mA for the recommended. As such, the minimum resistance for R1 is 125 ohms; anything higher would still protect the Arduino.

PROBLEM#2: What constraints must you be aware of in determining an acceptable value for R2?

Course: CMPE311-Project 3
DATE: Dec 1, 2025
TO: Professor Kidd
FROM: Abhinna Das
SUBJECT: Duty cycle project

Answer: You want to make sure that the resistor value is not big enough to limit the draw a lot of current from the Arduino.

PROBLEM#3: What is the minimum response time theoretically possible by the ATmega328p running at 16MHz?

Answer: Theoretically, the minimum is 1/16MHz = 62.5 Ns, or the period of the clock.

Video:
https://drive.google.com/file/d/14IUBa9NWl915gmQ3GhyOmCS3pN6i_tON/view?usp=drive_link

Code section: `// --- Hardware Definitions (Merged) ---`

```
// --- Hardware Definitions (Merged) ---
const int PWM_PIN = 9;      // The pin driving the MOSFET/LED (Timer1)
const int BUTTON_PIN = 2;   // The pin connected to the button
#define LED_1 5             // Serial Controlled LED 1
#define LED_2 6             // Serial Controlled LED 2

// --- PWM Logic Constants & Globals ---
const unsigned int PWM_PERIOD_TICKS = 20000;
const unsigned long DEBOUNCE_DELAY = 20;

volatile unsigned int pwmDutyTicks = 0;
unsigned long lastDebounceTime = 0;
int buttonState;
int lastButtonState = LOW;
int sequenceIndex = 0;

// Duty Cycle Sequence: 0%, 25%, 50%, 75%, 100%, 75%, 50%, 25%
const float dutyCycleSequence[] = {0.0, 0.25, 0.50, 0.75, 1.0, 0.75, 0.50,
0.25};
const int SEQUENCE_LENGTH = 8;

// --- LED Blinking Logic Globals ---
byte LED1State = LOW;
byte LED2State = LOW;
unsigned long prevTimeLED1 = 0, LED1BlinkDelay = 0;
unsigned long prevTimeLED2 = 0, LED2BlinkDelay = 0;
```

Course: CMPE311-Project 3
DATE: Dec 1, 2025
TO: Professor Kidd
FROM: Abhinna Das
SUBJECT: Duty cycle project

```
// Variables for User Input
int chosenLED = 0;
int chosenDelay = 0;

typedef void (*cycleCode)();

// Forward declarations
void GetAndSet();
void blink();
void handlePWMButton();
void setupTimer1PWM();
void advanceDutyCycle();
int pickvalue(const char* printout);

// The Task Array
// We add handlePWMButton here so it runs in the cycle
cycleCode tasks[] = {GetAndSet, blink, handlePWMButton};

int count = 0;
int task_count = sizeof(tasks)/sizeof(tasks[0]);

void setup() {
  // Configure Pins
  pinMode(PWM_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT); // Assumes external pull-down
  pinMode(LED_1, OUTPUT);
  pinMode(LED_2, OUTPUT);

  // Configure Serial
  Serial.begin(9600);
  Serial.println("System Starting: Cyclic Executive Active");

  // Configure Hardware Timer for PWM (Runs in background via ISR)
  setupTimer1PWM();
}
```

Course: CMPE311-Project 3
DATE: Dec 1, 2025
TO: Professor Kidd
FROM: Abhinna Das
SUBJECT: Duty cycle project

```
void loop() {
  // Execute the current task in the array
  tasks[count % task_count]();

  // Move to next task
  count++;
}

void GetAndSet() {
  // Initial prompt logic
  if (chosenLED == 0 && chosenDelay == 0) {
    Serial.print('\n');
    Serial.println("To get started: Select LED (1,2) and interval.");
    // We force a start state to enter the logic below
    chosenLED = 1;
  }

  // Check if data is waiting to avoid blocking the main loop
unnecessarily
  if (Serial.available() == 0) {
    chosenLED = pickvalue("What LED? (1 or 2)");

    Serial.print("Picked LED: ");
    Serial.println(chosenLED);

    if (chosenLED != 1 && chosenLED != 2) {
      Serial.println("LED Picked is not available");
      return;
    }

    chosenDelay = pickvalue("What interval (in msec)?");
    Serial.print("Blink Rate: ");
    Serial.println(chosenDelay);
  }

  // Assign the rate
  if (chosenLED == 1) {
```

Course: CMPE311-Project 3
DATE: Dec 1, 2025
TO: Professor Kidd
FROM: Abhinna Das
SUBJECT: Duty cycle project

```
    LED1BlinkDelay = chosenDelay;
  } else if (chosenLED == 2) {
    LED2BlinkDelay = chosenDelay;
  }
}

// Helper to keep system alive while waiting for Serial input
void runBackgroundTasks() {
    blink();             // Keep LEDs blinking
    handlePWMButton();   // Keep PWM button responsive
}

int pickvalue(const char* printout) {
  Serial.println(printout);

  while (Serial.available() == 0) {
    runBackgroundTasks();
  }

  int number = Serial.parseInt();

  // Clear buffer
  while (Serial.available() != 0) {
    Serial.read();
  }
  return number;
}

void blink() {
  unsigned long timeNow = millis();

  // Handle LED 1
  if (LED1BlinkDelay > 0 && (timeNow - prevTimeLED1 > LED1BlinkDelay)) {
    prevTimeLED1 += LED1BlinkDelay; // Keep accurate timebase
    LED1State = !LED1State;         // Toggle
    digitalWrite(LED_1, LED1State);
  }
```

Course: CMPE311-Project 3
DATE: Dec 1, 2025
TO: Professor Kidd
FROM: Abhinna Das
SUBJECT: Duty cycle project

```
  // Handle LED 2
  if (LED2BlinkDelay > 0 && (timeNow - prevTimeLED2 > LED2BlinkDelay)) {
    prevTimeLED2 += LED2BlinkDelay; // Keep accurate timebase
    LED2State = !LED2State;         // Toggle
    digitalWrite(LED_2, LED2State);
  }
}

void handlePWMButton() {
  int reading = digitalRead(BUTTON_PIN);

  // If the switch changed, due to noise or pressing:
  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > DEBOUNCE_DELAY) {
    // If state has stabilized
    if (reading != buttonState) {
      buttonState = reading;

      // Detect RISING EDGE (LOW to HIGH)
      if (buttonState == HIGH) {
        advanceDutyCycle();
      }
    }
  }
  lastButtonState = reading;
}
void advanceDutyCycle() {
  // 1. Increment Index
  sequenceIndex++;
  if (sequenceIndex >= SEQUENCE_LENGTH) {
    sequenceIndex = 0;
  }
  // 2. Calculate new PWM ticks
```

Course: CMPE311-Project 3
DATE: Dec 1, 2025
TO: Professor Kidd
FROM: Abhinna Das
SUBJECT: Duty cycle project

```
  float currentRatio = dutyCycleSequence[sequenceIndex];
  noInterrupts();
  pwmDutyTicks = (unsigned int)(PWM_PERIOD_TICKS * currentRatio);
  // Update Compare Register (Offset by TCNT start)
  OCR1A = (65536 - PWM_PERIOD_TICKS) + pwmDutyTicks;
  interrupts();

  Serial.print("PWM Button Pressed. Duty: ");
  Serial.print(currentRatio * 100);
  Serial.println("%");
}

void setupTimer1PWM() {
  noInterrupts();
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1  = 0;
  OCR1A = 0; // Start at 0%
  // Prescaler 8 (16MHz/8 = 2MHz). 20000 ticks = 10ms (100Hz)
  TCCR1B |= (1 << CS11);
  // Enable Output Compare A and Overflow Interrupts
  TIMSK1 |= (1 << OCIE1A) | (1 << TOIE1);
  interrupts();
}
// ISR: Timer Overflow (Start of Period)
ISR(TIMER1_OVF_vect) {
  TCNT1 = 65536 - PWM_PERIOD_TICKS; // Reset counter
  if (pwmDutyTicks > 0) {
    digitalWrite(PWM_PIN, HIGH);
  }
}
// ISR: Compare Match (End of Duty Cycle)
ISR(TIMER1_COMPA_vect) {
  if (pwmDutyTicks < PWM_PERIOD_TICKS) {
    digitalWrite(PWM_PIN, LOW);
  }
}
```