PWM signals in Arduino.

Noura khdour-1212072

1212072@student.birzeit.edu

Abstract- **This task demonstrates the design of a PWM signal generator using the Arduino Uno and its 16-bit Timer1 module. In Part A, the PWM signal is generated on digital pin 9 with a fixed frequency of 1 kHz and a duty cycle adjustable from 0% to 100% in 10% steps. The system uses Fast PWM mode (Mode 14), where the ICR1 register sets the TOP value, allowing accurate frequency control while OCR1A adjusts the duty cycle dynamically through Serial input. Part B explores all Timer1 modes—Normal, CTC, Fast PWM, Phase Correct PWM and Phase and Frequency Correct PWM -detailing how each works and its common applications. Fast PWM was chosen allows high-resolution control and stable frequency. Making it ideal for real-time PWM applications like LED dimming or motor speed control.**

Keywords- PWM, TOP.

I. INTRODUCTION

Pulse Width Modulation is a technique to get variable voltage in terms of Digital Input. PWM device generates ON and OFF pulses according to the control signal, which determines the desired voltage level. PWM is used to control the amount of power delivered to the load. It is commonly used for controlling the brightness of LED, the speed of motors, etc. Arduino Uno R3 has 6 PWM pins that are 3, 5, 6, 9, 10, and 11. These pins are marked with the negation sign "∼". These pins can generate a pulse as per the given inputs. Arduino supports an 8-bit wide pulse that can have 256 possible levels (0 to 255).[1]

II. PROCEDURE

A. Design a PWM signal generator using Arduino Uno and Timer1

1) Timer1 Overview: Timer1 is a 16-bit timer on the (Arduino Uno), capable of generating PWM signals with high resolution. It supports Fast PWM mode, which is ideal for generating a fixed-frequency PWM signal. Pin 9 (OC1A) is used as the PWM output pin, controlled by Timer1's Output Compare Register A (OCR1A).

2) Frequency Calculation:Target PWM frequency: 1 kHz. Arduino Uno clock frequency: 16 MHz. To achieve 1 kHz, we use a prescaler and calculate the TOP value (maximum counter value).

Formula: f_pwm= $\frac{f\_CPU}{prescaler \times (TOP+1)}$

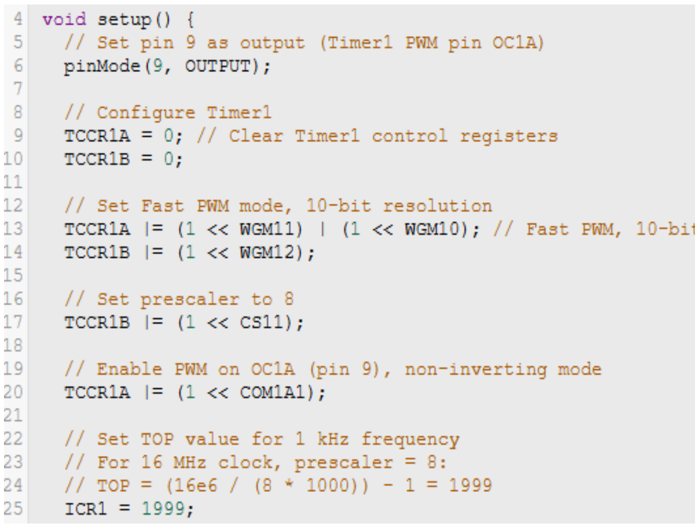
where *f_CPU* is 16 MHz, *Prescaler* is 8,so the *TOP* is 1999. We set ICR1 = 1999 to define the TOP value, giving a 10-bit resolution PWM.

3) Timer1 Configuration: Fast PWM Mode (10-bit): Set using WGM13:0 = 1110 (WGM11 and WGM10 in TCCR1A, WGM12 in TCCR1B, WGM13 = 0). This mode counts from 0 to ICR1 (1999) and resets, creating a 1 kHz signal. Prescaler: Set to 8 by enabling CS11 in TCCR1B (TCCR1B |= (1 << CS11)). PWM Output: Non-inverting mode on OC1A (pin 9) is enabled by setting COM1A1 in TCCR1A, making the pin high when the counter is below OCR1A and low when above.

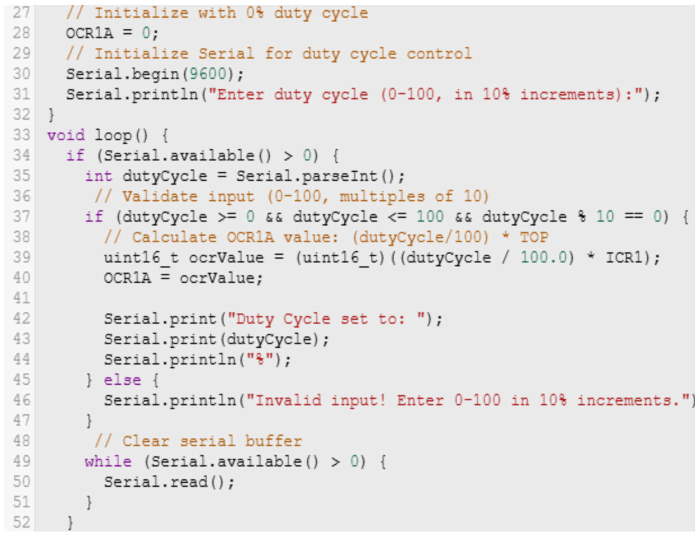
4) Duty Cycle Control: the duty cycle is controlled by setting OCR1A, which ranges from 0 (0% duty cycle) to ICR1 (1999, 100% duty cycle). For a given duty cycle percentage (e.g., 50%), $OCR1A = \left(\frac{duty\ cycle}{100}\right) * ICR1$

For 50%, OCR1A = 0.5 * 1999 ≈ 999. The code accepts user input via Serial (0, 10, 20, ..., 100) and calculates the corresponding OCR1A value.

5) Arduino code:



```
void setup() {
  // Set pin 9 as output (Timer1 PWM pin OC1A)
  pinMode(9, OUTPUT);

  // Configure Timer1
  TCCR1A = 0; // Clear Timer1 control registers
  TCCR1B = 0;

  // Set Fast PWM mode, 10-bit resolution
  TCCR1A |= (1 << WGM11) | (1 << WGM10); // Fast PWM, 10-bi
  TCCR1B |= (1 << WGM12);

  // Set prescaler to 8
  TCCR1B |= (1 << CS11);

  // Enable PWM on OC1A (pin 9), non-inverting mode
  TCCR1A |= (1 << COM1A1);

  // Set TOP value for 1 kHz frequency
  // For 16 MHz clock, prescaler = 8:
  // TOP = (16e6 / (8 * 1000)) - 1 = 1999
  ICR1 = 1999;
```



```
  // Initialize with 0% duty cycle
  OCR1A = 0;
  // Initialize Serial for duty cycle control
  Serial.begin(9600);
  Serial.println("Enter duty cycle (0-100, in 10% increments):");
}
void loop() {
  if (Serial.available() > 0) {
    int dutyCycle = Serial.parseInt();
     // Validate input (0-100, multiples of 10)
    if (dutyCycle >= 0 && dutyCycle <= 100 && dutyCycle % 10 == 0) {
      // Calculate OCR1A value: (dutyCycle/100) * TOP
      uint16_t ocrValue = (uint16_t)((dutyCycle / 100.0) * ICR1);
      OCR1A = ocrValue;

      Serial.print("Duty Cycle set to: ");
      Serial.print(dutyCycle);
      Serial.println("%");
    } else {
      Serial.println("Invalid input! Enter 0-100 in 10% increments.")
    }
     // Clear serial buffer
    while (Serial.available() > 0) {
      Serial.read();
    }
  }
```

Figure 1:Arduino code

B. Testing the PWM Signal

1.LED with Resistor(300Ω)

Purpose: Visually observe the PWM signal by varying the brightness of an LED.

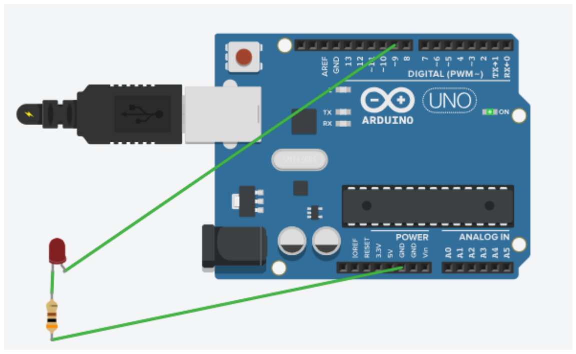


Figure 2: Led with resistor circuit at D=0%

-Note that we connect Anode of LED to Arduino pin 9.

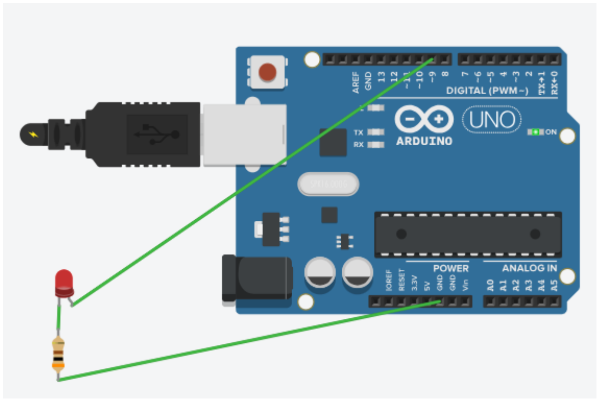


Figure 3: LED brightness when D=10%

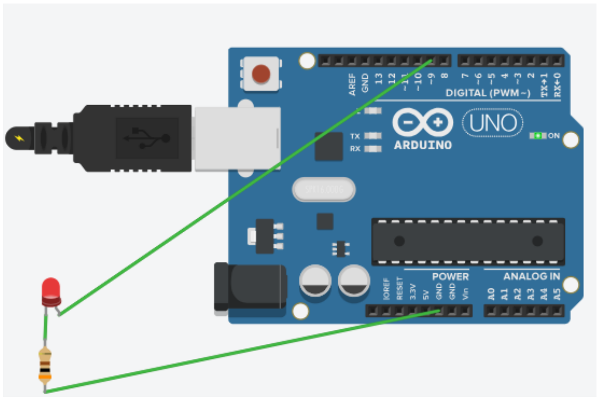


Figure 3: LED brightness when D=50%

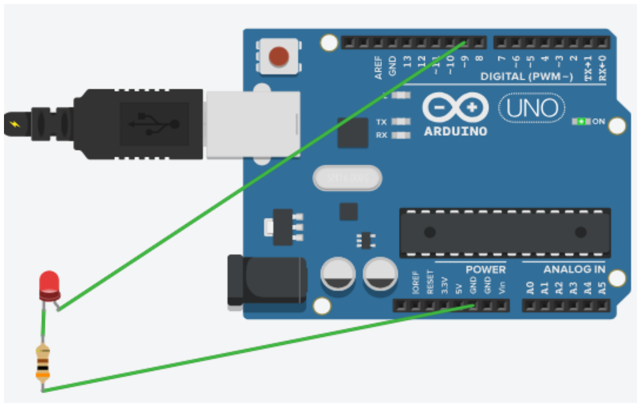


Figure 4: LED brightness when D=100%

The PWM signal modulates the LED's brightness. At 0% duty cycle, the LED is off; at 100%, it's fully on; at 50%, it's half brightness.

2- Oscilloscope (Signal Verification)

Purpose: Measure and visualize the PWM signal's frequency (period = 1 ms) and duty cycle.

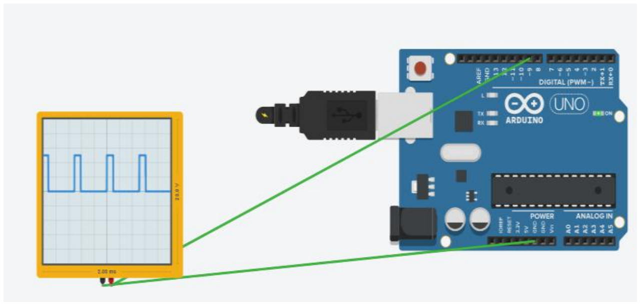


Figure 5: PWM signal when D=10%

High time is about 10% of 1 ms and the low time is about 90% of 1 ms.

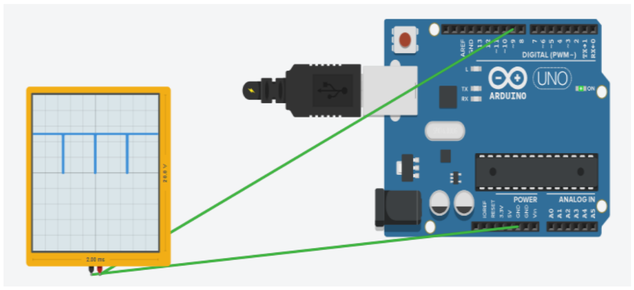


Figure 6: PWM signal when D=50%

The HIGH and LOW durations are both around 0.5 ms.

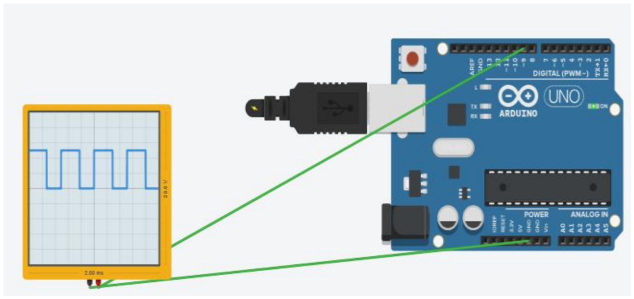


Figure 7: PWM signal when D=80%

High time is about 80% of 1 ms and the low time is about 20% of 1 ms.

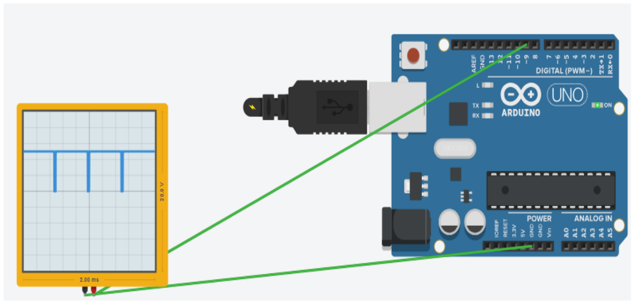


Figure 8: PWM signal when D=100%

The signal is always HIGH — no visible LOW time at all.

C. available modes of Timer1 on the Arduino Uno

Timer1 on the Arduino Uno (ATmega328P) is a 16-bit timer with multiple operating modes controlled by the Waveform Generation Mode (WGM) bits in the TCCR1A and TCCR1B registers.

1. *Normal Mode (WGM13:0 = 0000)*

How It Works: The timer counts from 0 to 65,535 (0xFFFF), then overflows and resets to 0. No special waveform generation occurs.

Applications: General-purpose timing (e.g., delays, event counting), measuring time intervals.

Behavior Impact: Free-running with no PWM output; relies on interrupts (e.g., TOV1) for timing tasks. Offers maximum count range but lacks waveform control.

2. *Clear Timer on Compare Match (CTC) Mode (WGM13:0 = 0100 or 1100)*

How It Works: The timer counts up to a user-defined TOP value (OCR1A or ICR1), then resets to 0. Output Compare pins (OC1A/OC1B) can toggle, set, or clear on match.

Applications: Precise frequency generation (e.g., square waves), triggering interrupts at specific intervals.

Behavior Impact: Provides accurate period control without inherent PWM; useful for fixed-frequency signals (e.g., 50% duty cycle) but requires manual duty cycle adjustment.

3. *Fast PWM Mode (WGM13:0 = 0101, 0110, 0111, 1110, 1111)*

How It Works: The timer counts up to a TOP value (8/9/10-bit, OCR1A, or ICR1), then resets to 0. PWM output on OC1A/OC1B is high until OCR1A/OCR1B, then low until TOP.

Applications: LED dimming, motor speed control, power regulation, high-frequency signal generation.

Behavior Impact: High-frequency PWM with variable duty cycle, offering up to 16-bit resolution (with ICR1). Ideal for applications needing constant frequency.

4. *Phase-Correct PWM Mode (WGM13:0 = 0001, 0010, 0011, 1010, 1011)*

How It Works: The timer counts up to TOP, then down to 0. PWM output is high during the up-count and low during the down-count, centered around the compare match.

Applications: Motor control (smoother transitions), audio signal generation, glitch-free PWM.

Behavior Impact: Symmetrical waveform reduces glitches, but half the frequency of Fast PWM for the same TOP, limiting maximum frequency.

5. Phase and Frequency Correct PWM Mode (WGM13:0 = 1000, 1001)

How It Works: Similar to Phase-Correct PWM, but compare values update at the bottom (0) for glitch-free transitions. TOP is set by ICR1 or OCR1A.

Applications: High-precision motor control, inverter circuits, frequency-stable PWM.

Behavior Impact: Ensures phase and frequency accuracy with dynamic TOP updates, but adds slight overhead due to update timing.

D. Justification for Choosing Fast PWM Mode in Part A.

Requirement: The PWM signal in Part A needs a 1 kHz frequency with a duty cycle adjustable from 0% to 100% in 10% increments on pin 9 (OC1A).

Choice: Fast PWM mode (WGM13:0 = 1110, 10-bit) was selected.

Frequency Match: With ICR1 = 1999 and prescaler = 8, meeting the 1 kHz target.

Resolution: 10-bit resolution (2000 steps) supports precise 10% increments (200 steps each), suitable for duty cycle control.

Simplicity: Fast PWM is straightforward to configure with TCCR1A and TCCR1B, requiring minimal code for fixed-frequency PWM.

Output: Directly generates PWM on OC1A (pin 9), aligning with the code's design.

Table1: Comparison of PWM Modes

E. Advantages and disadvantages of using different modes for PWM generation.

Mode	Advantages	Disadvantages
Fast PWM (Mode 14)	Precise frequency control withICR1. Higher frequency. High resolution (16-bit). Simple to configure.	Single-slope waveform may introduce glitches. less suitable for symmetrical signals.
Phase-Correct PWM	Symmetrical waveform reduces glitches. ideal for motor control or audio where smooth transitions matter.	Half the frequency (e.g., 500 Hz for the same TOP and pre-scaler). Reducing max frequency capability.
Phase and Frequency Correct PWM	Glitch-free updates and frequency stability. useful for dynamic frequency adjustments in precision applications.	Lower frequency (half of Fast PWM). increased complexity due to bottom-update timing. Higher overhead.
Fast PWM (Modes 5, 6, 7)	Simpler for low-resolution applications. Fixed TOP values reduceconfiguration complexity.	Limited resolution (8,9,10 bit). Less flexible for frequency control.
Fast PWM (Mode 15)	Uses OCR1A for TOP, freeing ICR1. High resolution.	limiting multi-channel PWM.

III. CONCLUSION.

In conclusion, Pulse Width Modulation (PWM) is a powerful technique used to simulate analog output using digital signals by varying the width (duration) of the high pulses in a square wave. we successfully generated a 1 kHz PWM signal using Timer1 on the Arduino Uno. By adjusting the duty cycle in 10% increments, we demonstrated how PWM can effectively control output power. his allows us to control devices like LEDs or motors with high precision, making PWM a key feature in many embedded applications.

IV- REFERENCE

[1] Arduino, "Secrets of Arduino PWM," *Arduino Documentation*, [Online]. Available: https://docs.arduino.cc/tutorials/generic/secrets-of-arduino-pwm/.

[2] Electronic Wings, "PWM in Arduino," *ElectronicWings*, [Online]. Available: https://www.electronicwings.com/arduino/pwm-in-arduino.