Birzeit University

Electrical and Computer Engineering Department

ENCS4380 - INTERFACING TECHNIQUES

PWM Signals in Arduino

Prepared by: Lana Musaffer - 1210455

Section: 1

Instructor: Dr. Wasel Ghanem

Date: 1/6/2025

1 Overview

Timer1 in the Arduino Uno (ATmega328P) is a 16-bit timer/counter capable of generating various timing and waveform functions. It supports multiple modes configured using the Waveform Generation Mode (WGM) bits in the `TCCR1A` and `TCCR1B` registers. These modes determine the behavior of the timer, including PWM generation, frequency control, and interrupt triggering.

2 PART A: PWM Signal Generation Using Timer1

The LED is connected to pin 9 (OC1A) of the Arduino Uno through a 220Ω resistor. Pin 9 is used as the PWM output generated via Timer1. The circuit schematic is shown in Figure 1.

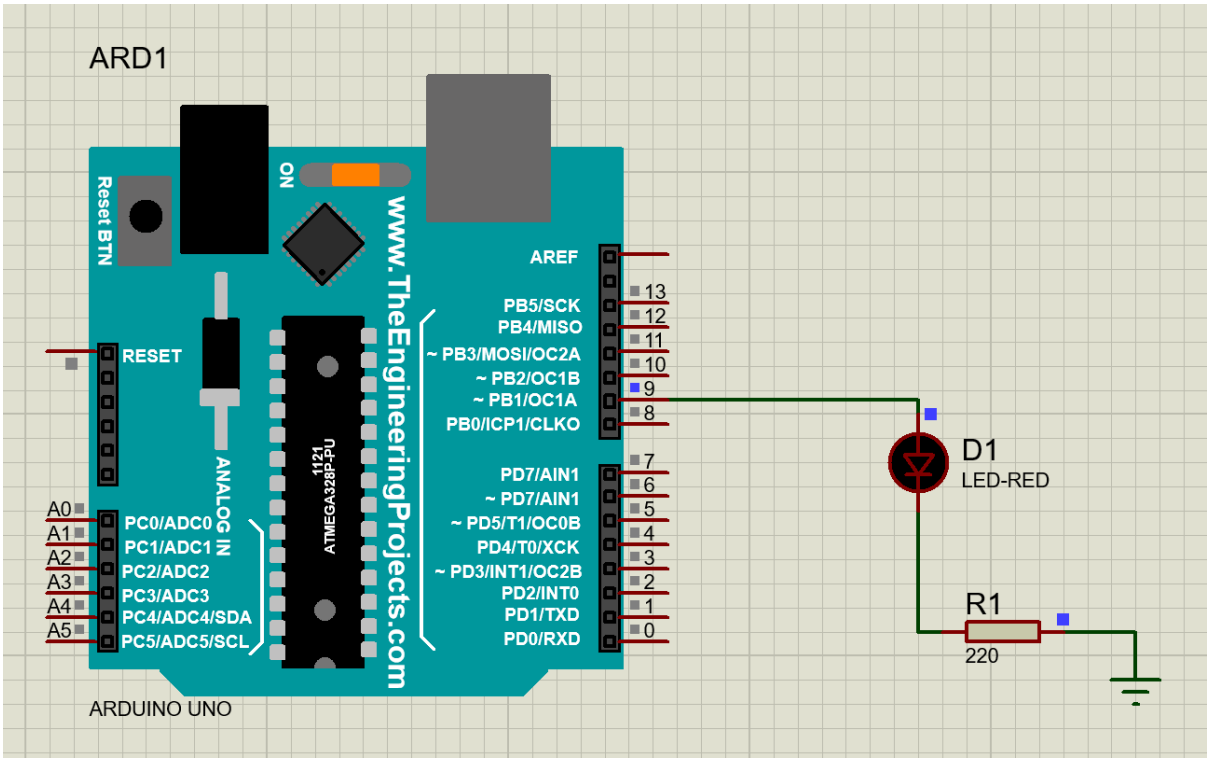
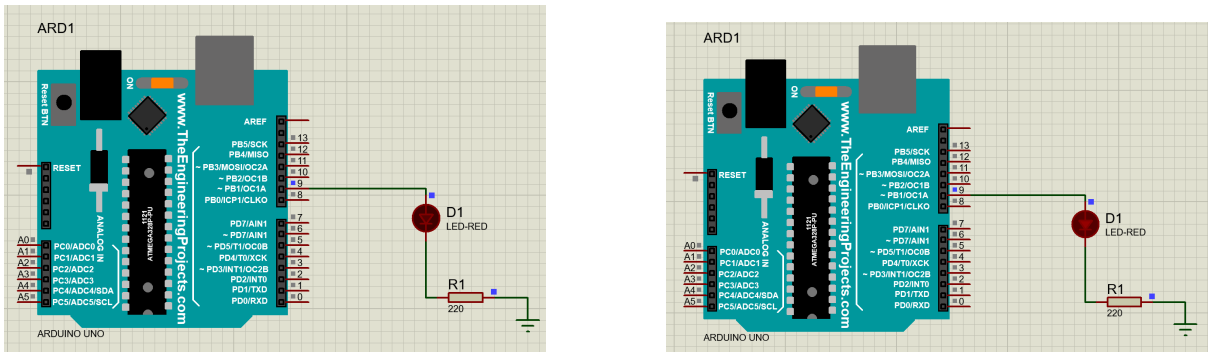


Figure 1: PWM LED Circuit connected to Pin 9 (OC1A) using Timer1

2.1 Procedure Summary

- Timer1 was configured in Fast PWM mode (Mode 14) using ICR1.
- A frequency of 1 kHz was achieved by setting ICR1 = 1999 with a prescaler of 8.
- The duty cycle was varied from 0% to 100% and back in 10% steps every second.
- The PWM output was observed on pin 9 and verified by LED brightness.

2.2 Results



(a) Duty cycle at a lower value

(b) Duty cycle at a higher value

Figure 2: LED brightness variations with changing PWM duty cycle

The LED connected to OC1A (Pin 9) was tested using Proteus simulation. The duty cycle was observed to increase and decrease in 10% steps as expected. The LED brightness changed accordingly, and the Virtual Terminal confirmed the PWM signal generation.

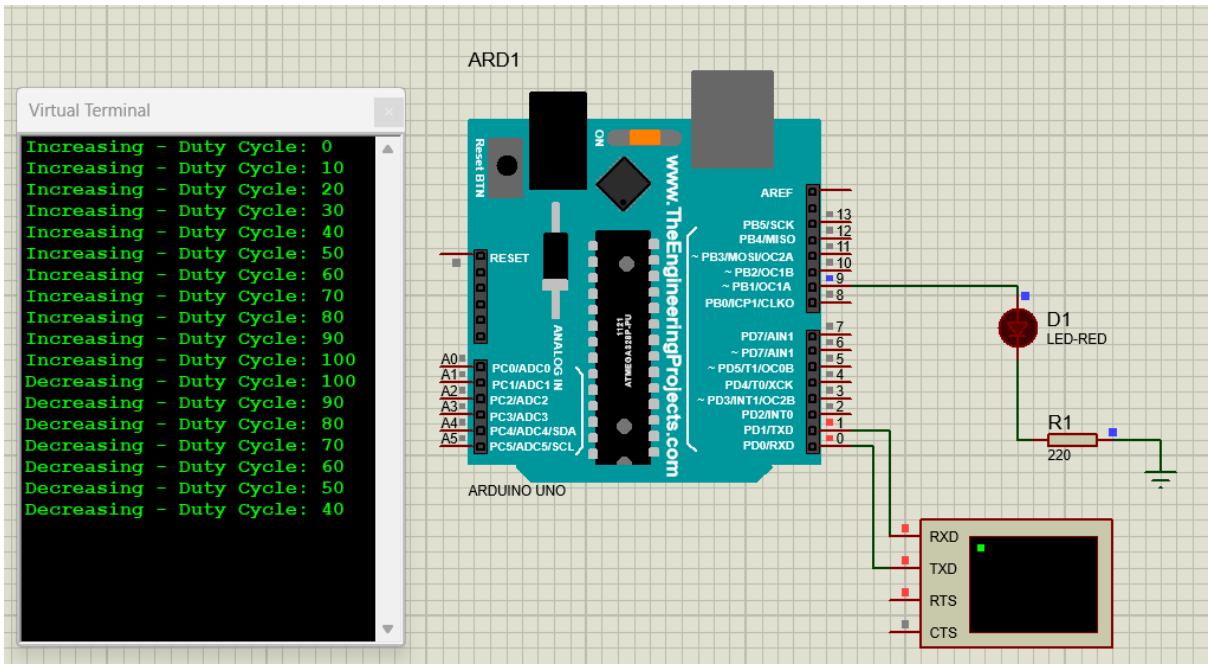


Figure 3: Working Proteus simulation showing PWM output and LED brightness variation on Pin 9

3 PART B: Available Timer1 Modes

3.1 Normal Mode (WGM13:0 = 0)

- **Description:** Counts from 0 to 65535. Triggers an interrupt on overflow.
- **Applications:** Time measurement, event counting.
- **Behavior:** No PWM output. Overflow handled manually.
- **Limitation:** Not suitable for PWM signal generation.

3.2 CTC Mode – Clear Timer on Compare Match (WGM13:0 = 4)

- **Description:** Resets to 0 when matching `OCR1A`.
- **Applications:** Frequency generation, square wave outputs.
- **Behavior:** Precise timing; limited duty cycle control.
- **Limitation:** Cannot produce smooth PWM; toggling only.

3.3 Fast PWM Mode using ICR1 (WGM13:0 = 14)

- **Description:** Counts from 0 to `ICR1`. Duty cycle via `OCR1A`.
- **Applications:** LED dimming, motor speed control, precise PWM.
- **Behavior:** High-frequency PWM with adjustable duty cycle.
- **Advantage:** Precise frequency and full 0–100% duty cycle control.
- **Note that this mode was chosen in part A for generating 1 kHz PWM with adjustable duty cycle.**

3.4 Phase Correct PWM (WGM13:0 = 8 or 10)

- **Description:** Counts up to TOP then back down (center-aligned).
- **Applications:** Motor control, audio (symmetrical waveform).
- **Behavior:** Stable frequency, slower updates.
- **Limitation:** Less frequency resolution than fast PWM.

3.5 Phase and Frequency Correct PWM (WGM13:0 = 9 or 11)

- **Description:** Like phase correct PWM with precise freq. control via `ICR1`.
- **Applications:** High-precision motor control.
- **Behavior:** Balanced waveform, lower frequency range.
- **Limitation:** Complex setup and lower max frequency.

4 Comparison of Timer1 Modes

Mode	Frequency Control	Duty Cycle Control	Applications
1. Normal Mode	No (manual only)	No	Delays, counters
2. CTC Mode	Yes (via OCR1A)	Limited (toggle)	Tone generation, timing
3. Fast PWM (ICR1)	Yes (via ICR1)	Full (0–100%)	**PWM signals**, LED, motors
4. Phase Correct PWM	Yes (moderate)	Full (0–100%)	Symmetric PWM, motors
5. Phase/Frequency Correct PWM	Yes (precise)	Full (0–100%)	Motor control, audio

Table 1: Comparison of Timer1 Modes on Arduino Uno

5 Justification for Mode Used in Part A

The **Fast PWM Mode using ICR1 (Mode 14)** was selected for generating the 1 kHz PWM signal in Part A because:

- It allows precise frequency control using `ICR1`.
- It supports full-range duty cycle adjustment from 0% to 100%.
- It uses hardware PWM on OC1A (Pin 9), reducing CPU load and improving accuracy.
- The mode is optimal for applications that require consistent and high-resolution PWM output.

6 Conclusion

Timer1 on Arduino Uno offers multiple modes for various timing and waveform generation tasks. For PWM applications requiring precise frequency and full duty cycle control, **Fast PWM using ICR1** stands out as the most versatile and accurate mode. While other modes like CTC or Phase Correct PWM have their specific advantages, they are less suitable for applications like real-time LED dimming or motor speed control.

Appendix: Arduino Code for PWM Generation

```
#define PWM_PIN 9  // OC1A - Timer1 PWM Output
int duty = 0; // Initial duty cycle percentage

void setup() {
  pinMode(PWM_PIN, OUTPUT);
  Serial.begin(9600);

  // Stop Timer1
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1 = 0;

  // Set Fast PWM mode using ICR1 as TOP (Mode 14)
  TCCR1A |= (1 << COM1A1);
  TCCR1A |= (1 << WGM11);
  TCCR1B |= (1 << WGM13) | (1 << WGM12);

  ICR1 = 1999; // 1 kHz
  OCR1A = (duty * ICR1) / 100;
  TCCR1B |= (1 << CS11);  // Prescaler = 8
}

void loop() {
  // Increase duty cycle
  for (duty = 0; duty <= 100; duty += 10) {
    OCR1A = (duty * ICR1) / 100;

    // Log to Serial Monitor
    Serial.print("Increasing - Duty Cycle: ");
    Serial.println(duty);

    delay(1000);
  }

  // Decrease duty cycle
  for (duty = 100; duty >= 0; duty -= 10) {
    OCR1A = (duty * ICR1) / 100;

    // Log to Serial Monitor
    Serial.print("Decreasing - Duty Cycle: ");
    Serial.println(duty);

    delay(1000);
  }
}
```

Listing 1: Arduino Code to Generate 1 kHz PWM with Variable Duty Cycle using Timer1