TAZROUT

IoT Irrigation System

Day 6 Learning Report: PWM LED Control

Developer: KHENFRI Moussa

Date: February 6, 2026
Week: 1 – ESP32 Fundamentals
Session: 3h 25m / 4h
Status: `Completed - Excellent!`

Module: ESP32 Sensor & Network Layer
Platform: Wokwi ESP32 Simulator
Project: TAZROUT IoT Irrigation System

Prepared as part of the 8-week ESP32 development plan
Personal learning journal and technical documentation

Contents

1 Executive Summary

Day 6 completes the input-output cycle I started building throughout Week 1. Today I combined Day 5's analog input (potentiometer) with PWM analog output (LED brightness control), creating my first complete sensor-to-actuator system. This is the fundamental pattern of all embedded control systems, including TAZROUT's irrigation logic.

1.1 Session Overview

Metric	Value
Planned Time	3-4 hours
Actual Time Spent	3 hours 25 minutes
Tasks Planned	8
Tasks Completed	8
Wokwi Projects Created	2 (PWM tests)
Lines of Code Written	∼210
Components Used	Potentiometer + LED
Coffee Consumed	2 cups

Table 1: Day 6 Session Metrics

1.2 Primary Achievement

Successfully implemented a complete analog control system featuring:

- ESP32 LEDC (LED Control) peripheral configuration
- PWM signal generation at 5kHz, 8-bit resolution
- Real-time potentiometer-to-LED brightness mapping
- Smooth transitions using moving average filter
- Professional multi-format status display

Success Achieved

Major Milestone - Complete Control Loop:

- **INPUT:** Read potentiometer (Day 5 skill)
- **PROCESS:** Filter and map values
- **OUTPUT:** Control LED brightness with PWM
- **FEEDBACK:** Visual confirmation of control
- Ready for Week 5 pump control implementation

This is the complete embedded systems pattern: Sense → Process → Actuate!

1.3 The "Analog Output" Revelation

Understanding PWM

Yesterday's Question: Can digital pins create analog output?
Today's Answer: Yes! Through PWM (Pulse Width Modulation)
How it works:

- Digital pins can only be HIGH or LOW
- But we can switch them very fast (5000 times/second!)
- By varying the ON/OFF ratio, we simulate analog voltage
- Human eye (or LED) averages the rapid switching
- Result: Smooth brightness control from 0-100%

Mind-blown moment: Digital can fake analog through speed!

2 Detailed Learning Journey

2.1 Phase 1: Understanding PWM (40 minutes)

This was the most conceptually intensive part of the day.

2.1.1 The Digital Output Limitation

First, I needed to understand the problem PWM solves:

The Problem

What I knew from Days 1-4:
`digitalWrite(LED_PIN, HIGH);` → LED fully ON (3.3V)
`digitalWrite(LED_PIN, LOW);` → LED fully OFF (0V)
The limitation:
Only two states! Cannot set LED to "half brightness" or "25% brightness"
What I needed:
Variable brightness control, like a dimmer switch.
Solution Preview: PWM!

2.1.2 PWM Conceptual Understanding

I spent 25 minutes truly understanding this concept:

PWM = Pulse Width Modulation

Core Concept: Rapidly switch between HIGH and LOW, varying the ratio.

Visual Representation:

100% Duty Cycle (Always ON):

```
<- Always HIGH
```

Result: LED fully bright

50% Duty Cycle (Half ON, Half OFF):

```
<- 50% HIGH, 50% LOW
```

Result: LED appears half brightness

25% Duty Cycle (Quarter ON):

```
<- 25% HIGH, 75% LOW
```

Result: LED appears dim

0% Duty Cycle (Always OFF):

```
<- Always LOW
```

Result: LED off

The Magic: If switching happens fast enough (5000 times/second), human eye cannot see individual flashes - it perceives average brightness!

Understanding the Choices

Why 5000 Hz frequency?
Too slow (e.g., 10 Hz):

- 10 cycles/second = visible flicker
- LED flashes noticeably
- Annoying to human eye

Good range (1000-10000 Hz):

- Fast enough: no visible flicker
- Efficient: doesn't waste CPU
- Common choice for LED control

Too fast (e.g., 100 kHz):

- Wastes CPU cycles
- No perceptible benefit
- May cause EMI (electromagnetic interference)

We chose 5000 Hz: Sweet spot - smooth, efficient, standard.
Why 8-bit resolution?

- 8-bit = 2^8 = 256 brightness levels (0-255)
- More than enough for smooth LED dimming
- Simple to work with (standard byte size)
- Efficient memory and calculation

Could use 10-bit (1024 levels) or 12-bit (4096 levels), but 256 levels is plenty smooth for human perception!

2.1.4 ESP32 LEDC Peripheral

LEDC = LED Controller
ESP32's built-in hardware for PWM generation.
Key features:

- **16 independent channels** (0-15)
- Each channel controls one GPIO pin
- Configurable frequency and resolution
- **Hardware-based** - runs independently of CPU!

Why hardware-based matters:
Software PWM would require:

```
// CPU must do this 5000 times/second!
digitalWrite(LED, HIGH);
```

```
delayMicroseconds(duty_time);
digitalWrite(LED, LOW);
delayMicroseconds(off_time);
// Blocks all other code!
```

Hardware PWM:

```
ledcWrite(channel, brightness);
// Done! Hardware handles the rest.
// CPU is free for other tasks.
```

Benefit: Set brightness once, hardware maintains it automatically. CPU can do other things (like reading sensors, processing MQTT, etc.)!

P

2.2 Phase 2: Building the Circuit (25 minutes)

Today's circuit combines Day 5's input with new PWM output.

2.2.1 Circuit Design

bΦENTIOMETER (INPUT - from Day 5): Pot VCC -> ESP32 3V3 Pot GND -> ESP32 GND
Pot SIG -> ESP32 GPIO 34 (ADC input)
LED (PWM OUTPUT - new today): LED Anode (+) -> ESP32 GPIO 26 (PWM output)
LED Cathode (-) -> 220 ohm Resistor -> ESP32 GND

2.2.2 Why GPIO 26 for PWM?

GPIO Pin Selection

Good news: ALL ESP32 GPIO pins can do PWM!
Unlike some microcontrollers (like Arduino Uno with only 6 PWM pins), ESP32's LEDC peripheral can use any output-capable pin.
But some pins to avoid:

- GPIO 0: Boot mode selection
- GPIO 2: Built-in LED (we're using external)
- GPIO 12: Boot voltage selection
- GPIO 15: Boot mode selection

GPIO 26 is perfect because:

- No special boot functions
- Commonly used in examples
- Well-documented
- No conflicts with WiFi or other peripherals

2.2.3 Circuit Testing

Before coding, I verified:

1. Potentiometer connected correctly (reused Day 5 wiring)
2. LED polarity correct (anode to GPIO 26, cathode to resistor)
3. Resistor value appropriate (220 for LED protection)
4. No short circuits
5. All connections firm in Wokwi

Time saved: 10 minutes! Day 5 circuit was already half-built.

2.3 Phase 3: PWM Configuration (45 minutes)

Understanding the three key LEDC functions was crucial.

2.3.1 The Three Essential Functions

LEDC Function Trinity

1. ledcSetup() - Configure PWM channel

```
ledcSetup(channel, frequency, resolution);
```

Parameters:

- `channel`: 0-15 (which PWM channel to configure)
- `frequency`: Hz (how fast to pulse)
- `resolution`: 1-16 bits (how many brightness levels)

Example:

```
ledcSetup(0, 5000, 8);
// Channel 0, 5kHz, 8-bit (0-255)
```

2. ledcAttachPin() - Connect GPIO to channel

```
ledcAttachPin(pin, channel);
```

Parameters:

- `pin`: GPIO number
- `channel`: Previously configured channel

Example:

```
ledcAttachPin(26, 0);
// GPIO 26 now controlled by channel 0
```

3. ledcWrite() - Set brightness

```
ledcWrite(channel, dutyCycle);
```

Parameters:

- `channel`: Which channel to control
- `dutyCycle`: 0 to ($2^{resolution}$-1)

Example:

```
ledcWrite(0, 128);
// 128/255 = 50% brightness
```

2.3.2 First PWM Test - Automatic Fade

Started simple to verify PWM works:

Listing 1: Basic PWM Fade Test

```
const int LED_PIN = 26;
const int PWM_CHANNEL = 0;

void setup() {
  Serial.begin(115200);

  // Configure PWM: channel 0, 5kHz, 8-bit
  ledcSetup(PWM_CHANNEL, 5000, 8);

  // Attach GPIO 26 to channel 0
  ledcAttachPin(LED_PIN, PWM_CHANNEL);

  Serial.println("PWM Fade Test Started");
}

void loop() {
  // Fade from 0 to 255 (off to full bright)
  for (int brightness = 0; brightness <= 255; brightness++) {
    ledcWrite(PWM_CHANNEL, brightness);
    Serial.println(brightness);
    delay(10);
  }

  // Fade from 255 to 0 (full bright to off)
  for (int brightness = 255; brightness >= 0; brightness--) {
    ledcWrite(PWM_CHANNEL, brightness);
    Serial.println(brightness);
    delay(10);
  }
}
```

Test Results:

- LED smoothly faded in (0→255 over 2.55 seconds)
- LED smoothly faded out (255→0 over 2.55 seconds)
- No visible steps or flicker
- Continuous smooth transition

Success Moment

When I saw the LED smoothly fade:
This was magical! After days of ON/OFF digital control, seeing *gradual* brightness change felt like unlocking a new dimension of control.
Realization: This is how all dimmable lights, motor speed controllers, and pump flow regulators work!

2.4 Phase 4: Combining Input + Output (50 minutes)

The main event - creating a complete control system!

2.4.1 The Control Loop Architecture

Listing 2: Complete Control Flow

```
INPUT (Potentiometer):
  analogRead(POT_PIN)
  Range: 0-4095
         |
         v
FILTER (Moving Average):
  applyFilter(rawValue)
  Smooths noise
         |
         v
MAP (Range Conversion):
  map(filtered, 0, 4095, 0, 255)
  Converts: ADC range -> PWM range
         |
         v
CONSTRAIN (Safety):
  constrain(brightness, 0, 255)
  Ensures valid PWM value
         |
         v
OUTPUT (LED Brightness):
  ledcWrite(PWM_CHANNEL, brightness)
  Controls LED via PWM
```

2.4.2 The Beautiful Mapping

The Challenge:
Potentiometer gives: 0-4095 (12-bit ADC)
PWM needs: 0-255 (8-bit duty cycle)
The Solution: map() function

```
int brightness = map(potValue, 0, 4095, 0, 255);
```

How it works:
Pot at 0%: $0 \rightarrow 0$ $(LEDoff)$
Pot at 25%: $1024 \rightarrow 64$ $(LEDdim)$
Pot at 50%: $2048 \rightarrow 128$ $(LEDhalf)$
Pot at 75%: $3072 \rightarrow 192$ $(LEDbright)$
Pot at 100%: $4095 \rightarrow 255$ $(LEDmax)$
Beautiful property: Perfectly proportional! 50% potentiometer = 50% LED brightness.
c

2.4.3 Complete Implementation

$$\text{bnst int POT}_{P}IN = 34; constint LED_{P}IN = 26; constint PWM_{C}HANNEL = 0;$$
// Moving average filter (from Day 5!) const int
$$\text{FILTER}_{S}IZE = 5; intreadings[FILTER_{S}IZE]; intreadIndex = 0; inttotal = 0;$$
void setup() Serial.begin(115200);
// Configure PWM
$$\text{ledcSetup(PWM}_{C}HANNEL, 5000, 8); ledcAttachPin(LED_{P}IN, PWM_{C}HANNEL);$$
// Initialize filter for (int i = 0; i $<$ $\text{FILTER}_{S}IZE; i++)readings[i] = 0;$
Serial.println("Potentiometer controls LED brightness!");
void loop() // Read potentiometer int potValue = $\text{analogRead(POT}_{P}IN)$;
// Apply filter (smooth control) int filtered = applyFilter(potValue);
// Map to PWM range int brightness = map(filtered, 0, 4095, 0, 255); brightness = constrain(brightness, 0, 255);
// Set LED brightness $\text{ledcWrite(PWM}_{C}HANNEL, brightness)$;
// Calculate percentage int percentage = (brightness * 100) / 255;
// Display status Serial.print("Pot: "); Serial.print(potValue); Serial.print(" -> Brightness: "); Serial.print(percentage); Serial.println("
delay(100);
int applyFilter(int newValue) total = total - readings[readIndex]; readings[readIndex] = newValue; total = total + newValue; readIndex = (readIndex + 1) return total / $\text{FILTER}_{S}IZE$;

2.4.4 Testing Interactive Control

Test sequence I performed:

1. Set potentiometer to 0% → LED completely off
2. Slowly rotate to 25% → LED dims smoothly
3. Continue to 50% → LED at half brightness
4. Continue to 75% → LED bright
5. Continue to 100% → LED maximum brightness
6. Rotate backward → LED dims smoothly
7. Rapid changes → Filter smooths out jitter

The "It Works!" Moment

Pure Joy Moment:
When I rotated the potentiometer and watched the LED brightness follow perfectly in real-time...
This is control engineering!
I created a complete feedback system:

- Sensor (potentiometer) senses user input
- Processor (ESP32) filters and maps data
- Actuator (LED) responds to command
- Visual feedback (brightness) confirms control

This is the pattern used in:

- Car cruise control (speed sensor → throttle actuator)
- Thermostat (temperature sensor → heater/AC)
- **TAZROUT irrigation (moisture sensor → pump!)**

Professional embedded systems engineer feeling: 9/10!

2.5 Phase 5: Smooth Transitions (30 minutes)

Made the control feel professional with enhanced filtering.

2.5.1 Why Filtering Matters for PWM

Without filter:

```
Pot jitters: 2048, 2051, 2047, 2053, 2046...
PWM jitters:  128,  128,  127,  129,  127...
LED flickers slightly at constant position!
```

With moving average filter:

```
Pot jitters: 2048, 2051, 2047, 2053, 2046...
Filtered:    2048, 2049, 2049, 2050, 2049...
PWM stable:   128,  128,  128,  128,  128...
LED rock solid!
```

Professional Touch

Filter makes control feel premium:
Without filter:

- LED jitters even when pot is still
- Control feels "nervous"
- Unprofessional appearance

With filter:

- LED brightness stable when pot is still
- Smooth transitions when rotating
- Professional, polished feel

Lesson: Small details (like filtering) separate hobby projects from professional products!

2.6 Phase 6: Understanding Duty Cycle (20 minutes)

Deep dive into PWM theory.

2.6.1 Duty Cycle Mathematics

Definition:

Duty Cycle = Percentage of time signal is HIGH

Formula:

$$DutyCycle(\%) = \frac{PWMValue}{MaxValue} \times 100$$

Examples:

PWM = 0: $0_{\overline{255\times100=0\% \quad (LEDOFF)}}$

PWM = 64: $64_{\overline{255\times100=25\% \quad (LED25\%)}}$

PWM = 128: $128_{\overline{255\times100=50\% \quad (LEDhalf)}}$

PWM = 192: $192_{\overline{255\times100=75\% \quad (LEDbright)}}$

PWM = 255: $255_{\overline{255\times100=100\% \quad (LEDmax)}}$

Why PWM creates brightness perception:

LED at 25% duty cycle:

- ON for 0.05ms, OFF for 0.15ms, repeat 5000 times/second
- LED is physically ON only 25% of the time
- But switches too fast for eye to see individual flashes
- Eye integrates (averages) the light over time
- Perceives: 25% brightness

Key insight: PWM doesn't change the voltage - it changes the *time* the voltage is applied!

S

2.7 Phase 7: Professional Display Output (25 minutes)

Created comprehensive status displays.

2.7.1 Multi-Level Display Strategy

1. Compact Format (Every Reading):

bSerial.print("Pot: "); Serial.print(potValue); Serial.print(" | PWM: "); Serial.print(brightness); Serial.print(" | Brightness: "); Serial.print(percentage); Serial.println("

Output:

```
Pot: 2048 | PWM: 128 | Brightness: 50%
```

2. Visual Bar Graph (Every 10 Readings):

Listing 3: Bar Graph Visualization

```
void printBarGraph(int percentage) {
  Serial.print("Brightness: [");

  int bars = percentage / 5;  // 20 bars for 100%
  for (int i = 0; i < 20; i++) {
    if (i < bars) {
      Serial.print("   ");
    } else {
      Serial.print("   ");
    }
  }

  Serial.print("] ");
  Serial.print(percentage);
  Serial.println("%");
}
```

Output:

```
Brightness: [] 50%
```

3. Detailed Analysis (Every 30 Readings):

Includes:

- Input ADC value and percentage
- Filtered value
- PWM duty cycle (0-255)
- LED brightness percentage
- Status classification (OFF/DIM/MEDIUM/BRIGHT)

Professional Display Design

Why three display levels?
Same principle as Day 5:

1. **Compact:** Monitor operation continuously
2. **Visual:** Quick brightness assessment
3. **Detailed:** Deep analysis for troubleshooting

Professional systems provide information at multiple levels of detail!

3 Connection to TAZROUT Project

3.1 PWM Applications in TAZROUT

Day 6 Skill	TAZROUT Application	Week
PWM basics	Variable pump speed control	Week 5
Brightness control	Gradual irrigation flow changes	Week 5
Smooth transitions	Prevent water hammer in pipes	Week 5
Duty cycle	Water flow rate control	Week 5
Input-to-output mapping	Moisture level to pump speed	Week 5

Table 2: Day 6 Skills Applied to TAZROUT

3.2 Advanced Irrigation Control Vision

Basic ON/OFF Control (Week 5 minimum):

```
if (moisturePercent < 30) {
  digitalWrite(PUMP_PIN, HIGH);  // Pump ON
} else {
  digitalWrite(PUMP_PIN, LOW);   // Pump OFF
}
```

Advanced PWM Control (Week 5 optional):

```
// Proportional pump speed based on moisture deficit
int pumpSpeed = 0;

if (moisturePercent < 20) {
  pumpSpeed = 255;  // Very dry -> full speed
} else if (moisturePercent < 40) {
  pumpSpeed = 180;  // Dry -> medium-high speed
} else if (moisturePercent < 60) {
  pumpSpeed = 128;  // Somewhat dry -> medium speed
} else {
  pumpSpeed = 0;    // Adequate moisture -> off
}

ledcWrite(PUMP_CHANNEL, pumpSpeed);
```

Benefits of PWM pump control:

1. **Gentle irrigation:** Gradual water application prevents soil erosion
2. **Water hammer prevention:** Slow ramp-up/down protects pipes
3. **Energy efficiency:** Lower speed = less power consumption
4. **Proportional response:** More dry = more water (smarter!)

Today's LED control is exactly the same as future pump control!

5 Challenges and Solutions

5.1 Challenge Log

#	Challenge	Solution	Time
1	Understanding PWM concept	Watched video, drew duty cycle diagrams	25 min
2	Choosing frequency/resolution	Researched typical values, used 5kHz/8-bit	10 min
3	LED not changing	Verified ledcAttachPin() connected correct GPIO	8 min
4	Control felt jittery	Applied moving average filter from Day 5	5 min
5	Mapping ADC to PWM	Used map() function with correct ranges	5 min

Table 4: Day 6 Challenge Log

5.2 Most Interesting Challenge

Challenge: Understanding PWM Conceptually

The Conceptual Hurdle:
How can rapidly switching ON/OFF create the appearance of a middle voltage?
Initial confusion:

- Thought PWM somehow "reduced" the voltage
- Didn't understand how ON/OFF could make "half brightness"
- Couldn't visualize what 5000 Hz looks like

The Breakthrough:
Drew timeline on paper:

1. One period at 5kHz = 0.2 milliseconds
2. Drew 50% duty cycle: 0.1ms ON, 0.1ms OFF
3. Realized: In 1 second, LED is ON 2500 times, OFF 2500 times
4. Human eye integration time is 20ms
5. In 20ms, LED flashes 100 times (50 ON, 50 OFF)
6. Eye cannot see individual flashes - averages to 50% brightness!

Analogy that clicked:
PWM is like a strobe light, but SO fast you can't see individual flashes - you only see the average brightness.
Time spent: 25 minutes
Value gained: Deep understanding of fundamental embedded technique!

6 Skills Development Matrix

Skill	After Day 5	After Day 6	Growth
PWM Control	0/10	8/10	+8
LEDC Peripheral	0/10	8/10	+8
Analog Output	0/10	8/10	+8
Control Systems	3/10	8/10	+5
Input-Output Mapping	6/10	9/10	+3
Complete Feedback Loop	2/10	9/10	+7
ESP32 Knowledge	8/10	9/10	+1

Table 5: Skill Development Progress

Personal Reflection

Massive Skill Acquisition Day!
PWM Control: 0 → 8/10
From zero knowledge to implementing professional PWM control!
Complete Feedback Loop: 2 → 9/10
Combined 6 days of learning into one complete system: input, processing, filtering, mapping, output, feedback.
Control Systems: 3 → 8/10
Now understand how real-world control systems work. This is engineering!
Overall Assessment:
Day 6 synthesized everything from Week 1. I can now:

- Read sensors (ADC)
- Process data (filter, map)
- Control actuators (PWM)
- Create complete control loops

Ready for real-world embedded projects!

7 Reflections and Insights

7.1 The "Complete System" Revelation

The Synthesis Moment:

Around 2 hours into Day 6, when the complete system was working (potentiometer controlling LED smoothly), I had a profound realization:

"This is what embedded systems ARE!"

What clicked:

- Day 1-2: Output control (GPIO, Serial)
- Day 3-4: Logic (state machines)
- Day 5: Input sensing (ADC)
- **Day 6: Everything together!**

The complete embedded systems cycle:

$$SENSE \rightarrow PROCESS \rightarrow ACTUATE \rightarrow FEEDBACK \rightarrow SENSE...$$

This is how ALL control systems work:

- Car cruise control
- Home thermostat
- Drone flight controller
- **TAZROUT irrigation system!**

Everything I've learned in Week 1 came together today!

1. Hardware PWM is a Gift

Software PWM would block the CPU completely. Hardware PWM frees the CPU to do other tasks. This is why ESP32 is powerful - 16 independent PWM channels!

2. Filtering Makes Control Professional

Without filtering: Works, but feels amateur

With filtering: Smooth, polished, professional

Small detail, huge impact on user experience.

3. Mapping is Universal

`map()` appears everywhere in embedded systems. Mastering it opens many doors.

4. Duty Cycle is Physics

PWM doesn't change voltage - it changes TIME. Elegant solution using digital to create analog effect!

5. Complete Loops Need All Pieces

Input alone is useless. Output alone is blind. Processing connects them into useful systems.

Confidence Transformation

Before Day 6: "I can read sensors and control outputs separately"
After Day 6: "I can create complete sensor-to-actuator control systems"
What changed:
The pieces became a whole. I don't just know isolated techniques - I know how to combine them into functional systems.
Specific milestone:
When I saw the LED respond smoothly to potentiometer in real-time with filtered, mapped control... that's when I felt like a real embedded systems engineer.
Readiness for Week 2: Excellent
Readiness for TAZROUT irrigation: Getting very ready!

8 Week 1 Near-Complete Status

8.1 Days Completed

Day	Status	Achievement
Day 1	Complete	LED blink, GPIO, Serial
Day 2	Complete	Serial communication mastery
Day 3	Complete	State machine foundation
Day 4	Complete	Professional state machine
Day 5	Complete	Analog input, ADC, filtering
Day 6	Complete	PWM output, complete control
Day 7	Tomorrow	Git, documentation, review

Table 6: Week 1 Progress - 6/7 Days (86%)

Status: One day away from completing Week 1!

8.2 Cumulative Skills Acquired

Week 1 Skill Arsenal: 2

- GPIO digital I/O
- Serial communication
- State machines
- Non-blocking timing
- ADC operations
- PWM control
- Moving average filtering
- Sensor calibration
- Data mapping
- Circular buffers
- Complete control loops
- Professional coding

9 Deliverables and Documentation

9.1 Wokwi Project

Project Details

Project Name: ESP32 PWM LED Control - Potentiometer Day6
Status: Fully functional, production-quality
Features:

- Potentiometer input (GPIO 34)
- LED PWM output (GPIO 26)
- Moving average filter (smooth control)
- Real-time brightness mapping
- Three-level display output
- Professional status reporting

Last Updated: February 6, 2026, 19:40

9.2 Code Quality Assessment

Criterion	Score	Notes
Functionality	5/5	Perfect potentiometer-to-LED control
Code Organization	5/5	Clean, modular, well-structured
Documentation	5/5	Comprehensive comments
Efficiency	5/5	Hardware PWM, optimal filtering
Maintainability	5/5	Easy to modify or extend
Professional Standards	5/5	Production-quality code
Overall	**30/30**	**100% - Perfect!**

Table 7: Code Quality Self-Assessment

Quality Progression:

- Day 1: 84%
- Day 2: 96%
- Day 3: 97%
- Day 4: 98%
- Day 5: 100%
- Day 6: 100%

Achieved and maintained 100% quality! Week 1 mastery!

10 Time Management Analysis

Activity	Planned	Actual	Variance
PWM Understanding	25 min	40 min	+15 min
Circuit Building	20 min	25 min	+5 min
PWM Configuration	30 min	45 min	+15 min
Input-Output Integration	35 min	50 min	+15 min
Smooth Transitions	20 min	30 min	+10 min
Duty Cycle Study	15 min	20 min	+5 min
Display Output	20 min	25 min	+5 min
Testing	15 min	10 min	-5 min
Total	**3h 00m**	**3h 25m**	**+25 min**

Table 8: Day 6 Time Analysis

10.1 Why Time Variance Was Small

1. Built on solid Day 5 foundation (potentiometer already working)
2. PWM concept clicked faster than expected
3. Reused moving average filter code (no reinvention)
4. Testing went faster (experience from 5 previous days)

Assessment: Excellent time management! Only 25 minutes over.

11 Self-Assessment Against Objectives

Objective	Met?	Evidence
Understand PWM		Can explain duty cycle, frequency, hardware implementation
Configure ESP32 LEDC		Successfully configured channel, attached pin, set brightness
Control LED brightness		Smooth 0-100% brightness control achieved
Combine analog I/O		Potentiometer input controls PWM output perfectly
Implement smooth transitions		Moving average filter provides professional control feel

Table 9: Day 6 Objectives - 100% Complete

Perfect Achievement

All 5 Objectives Met!
Day 6 exceeded expectations. Created production-quality control system!

12 Conclusion and Personal Notes

12.1 Overall Assessment

Day 6: OUTSTANDING SUCCESS

Day 6 was the synthesis day - everything from Week 1 came together!
Major Achievements:

- Mastered PWM control from concept to implementation
- Created complete sensor-to-actuator control system
- Achieved professional-quality smooth control
- Synthesized 6 days of learning into working system
- Ready for TAZROUT advanced pump control

Confidence level: 9/10 (highest yet!)
Readiness for Day 7: Excellent
Readiness for Week 2: Completely ready!

12.2 What I'm Most Proud Of

1. **Complete system integration:** Input + Processing + Output working harmoniously
2. **Professional control feel:** Filtering makes it feel like commercial product
3. **Deep PWM understanding:** Not just using it, but truly understanding the physics
4. **Code quality:** 100% (30/30) for second consecutive day!

12.3 Personal Reflection

After 6 outstanding days, I'm more committed than ever to:

- Deep understanding over surface completion
- Professional quality in every project
- System thinking (how pieces work together)
- Continuous improvement and learning

Tomorrow (Day 7): Git, documentation, Week 1 reflection
Week 2 Preview: Real sensors (moisture, temperature, humidity)!

Signed (digitally),
KHENFRI Moussa
February 6, 2026

---

b

Next: Day 7 - Git Setup & Week 1 Review

TAZROUT ESP32 Module Development
Developer: KHENFRI Moussa