Lab 3: GPIO Control and LED Blinking

Embedded Systems Programming

John Doe October 08, 2025

Assignment Overview

Document Contents:

Assignment Details:

- **Course:** ECSE 303
- **Instructor:** Prof. Smith
- **Due Date:** September 19, 2025
- **Hardware:** Raspberry Pi 4, LED, 220Ω resistor, breadboard, jumper wires
- **Software:** Python (RPi.GPIO), C (WiringPi)
- **Duration:** ~3 hours
- **Lab Number:** Lab 3

Introduction

This lab explores the fundamentals of General Purpose Input/Output (GPIO) control on the Raspberry Pi platform. We will implement a simple LED blinking program that demonstrates basic digital output control and timing mechanisms.

The objectives of this lab are:
- Understand GPIO pin configuration and control
- Implement basic timing functions
- Compare Python and C implementations
- Analyze performance differences between programming languages

Background

GPIO pins allow microcontrollers and single-board computers to interface with external devices. The Raspberry Pi provides 40 GPIO pins that can be configured as either inputs or outputs, with configurable pull-up/pull-down resistors and interrupt capabilities.

Methodology

Hardware Setup

The following components were used:
- Raspberry Pi 4 Model B
- Red LED (2.1V forward voltage, 20mA forward current)
- 220Ω current-limiting resistor
- Breadboard for prototyping
- Jumper wires for connections

The LED was connected between GPIO pin 18 and ground, with the current-limiting resistor in series.

Software Implementation

Two implementations were developed:

1. Python Implementation: Using the `RPi.GPIO` library
2. C Implementation: Using the `WiringPi` library

Both programs implement the same functionality: blinking an LED at 1Hz (500ms on, 500ms off).

Code Examples

The Python implementation:

```python
import RPi.GPIO as GPIO
import time

# Set up GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)

try:
    while True:
        GPIO.output(18, GPIO.HIGH)  # Turn LED on
        time.sleep(0.5)
        GPIO.output(18, GPIO.LOW)   # Turn LED off
        time.sleep(0.5)
except KeyboardInterrupt:
    GPIO.cleanup()
```

The C implementation:

```c
#include <wiringPi.h>
#include <stdio.h>

#define LED_PIN 18

int main(void) {
    wiringPiSetupGpio();
    pinMode(LED_PIN, OUTPUT);

    while (1) {
        digitalWrite(LED_PIN, HIGH);
        delay(500);
        digitalWrite(LED_PIN, LOW);
        delay(500);
    }

    return 0;
}
```

Results

Both implementations successfully controlled the LED with the following observations:
- The LED blinked consistently at 1Hz
- Visual timing appeared identical between implementations
- The C implementation showed slightly more precise timing
- Python implementation was easier to develop and debug

Performance Analysis

Timing measurements were conducted using a logic analyzer:

The C implementation demonstrated more consistent timing, likely due to reduced overhead compared to Python's interpreted execution.

Discussion

The lab successfully demonstrated basic GPIO control on the Raspberry Pi. Key findings include:

- GPIO configuration is straightforward with both RPi.GPIO and WiringPi libraries
- Hardware setup requires attention to current-limiting resistors
- C implementations offer better timing precision for real-time applications
- Python provides faster development cycles for prototyping

Conclusion

This lab provided hands-on experience with GPIO control on embedded systems. The successful implementation of LED blinking in both Python and C demonstrates the versatility of the Raspberry Pi platform for embedded programming education.

Future work could explore:

- PWM control for LED brightness modulation
- Interrupt-driven input handling
- Multi-threaded applications
- Real-time operating system integration