EE351

None

Qijun Han

None

Table of contents

1. EE351

EE351“ ”24Fall

4B Raspberry Pi OS

1.1

-
-
-
- PCF8691
-
-
-
- PS2
-
-

:

2.

2.1 Lab1 Raspberry Pi SSH VNC

2.1.1

SD Raspberry Pi Imager SSH VNC SD

2.1.2

- SD Formmater SD Raspberry Pi Imager Win32DiskImager SD
- SSH Secure Shell
- VNC Virtual Network Computing VNC Viewer VNC

2.1.3

- SD Formmater SD
- Raspberry Pi Imager
- SD config.txt
- SD
-
- SSH VNC IP
- Putty MobaXterm SSH
- VNC Viewer xrdp Windows

2.2

Lab2 LED

Raspberry Pi IO wiringPi RPi.GPIO Mu Geany IDE Python C/C++
LED

1. **Raspberry Pi IO**
2. Raspberry Pi 40 GPIO wiringPi BCM2837 SOC
3. BCM
4. **wiringPi**
5. wiringPi C/C++ GPIO Raspberry Pi GPIO GPIO
6. **RPi.GPIO**
7. RPi.GPIO Python Python Raspberry Pi GPIO Raspbian API
8. **Mu Geany IDE**
9. Mu Python IDE
10. Geany IDE C/C++
11. **LED**
12. LED LED

1.
2. LED S Raspberry Pi GPIO GND Raspberry Pi GPIO BCM GPIO19, GPIO20, GND
3.
4. Mu Python LED GPIO / LED
5. C/C++ Geany GPIO wiringPi
6.
7. Python C/C++ LED
8.
9.
10.

PYTHON

Python LED

```
import RPi.GPIO as GPIO
import time

# Define GPIO pins for the LED (BCM numbering)
RED_PIN = 19  # Red part of the dual-color LED
GREEN_PIN = 20  # Green part of the dual-color LED

# Setup GPIO mode and pin directions
GPIO.setmode(GPIO.BCM)
GPIO.setup(RED_PIN, GPIO.OUT)
GPIO.setup(GREEN_PIN, GPIO.OUT)

try:
```

```
    while True:
        # Turn on red LED
        GPIO.output(RED_PIN, GPIO.HIGH)
        GPIO.output(GREEN_PIN, GPIO.LOW)
        print("Red LED is ON")
        time.sleep(1)  # Wait for 1 second

        # Turn on green LED
        GPIO.output(RED_PIN, GPIO.LOW)
        GPIO.output(GREEN_PIN, GPIO.HIGH)
        print("Green LED is ON")
        time.sleep(1)  # Wait for 1 second

except KeyboardInterrupt:
    print("Program stopped by user")

finally:
    # Clean up GPIO settings before exiting
    GPIO.cleanup()
```

LED - - - 1

2.3

Lab3

Raspberry Pi GPIO LED

1.
2. S 0V 3.3V 5V
3. **GPIO**
4. Raspberry Pi GPIO GPIO
5. **LED**
6. LED
7.
8. RC

1.
2. SIG(S) Raspberry Pi GPIO17 VCC 5V GND LED
3. LED T GPIO19, GPIO20, GND LED GPIO27 GND
4.
5. Python RPi.GPIO GPIO
6. GPIO17
7.
8. LED
9.
10. Python LED
11. LED
12.
13.
14.
15.
16. GPIO

2.4 PCF8691

Lab5 PCF8591

PCF8591 Analog-to-Digital Converter, ADC Raspberry Pi PCF8591 8 ADC I2C PCF8591 LED

1. **PCF8591**
2. PCF8591 CMOS 8 A/D 8 D/A
3. I2C 0x48 A0, A1, A2 8 I2C
4. **I2C**
5. I2C SDA SCL Raspberry Pi PCF8591
6.
7. AIN0 AOUT LED LED
8. NTC

1.
2. Raspberry Pi T PCF8591 SDA SCL VCC GND
3. LED PCF8591 AOUT GND LED GND GPIO GPIO17
4. **I2C**
5. Raspberry Pi Preferences -> Raspberry Pi Configuration
6. Interfaces I2C OK
7.
8. Python `smbus` I2C
9. SMBus PCF8591 AIN0 AOUT LED
10.

```
import smbus
import time

# Define the I2C address of the PCF8591 and control bits
address = 0x48  # Default address for PCF8591
control_bit = 0x40  # Command to start conversion on channel 0 (AIN0)

# Initialize the SMBus library
bus = smbus.SMBus(1)  # Use I2C bus 1

try:
    while True:
        # Write the control byte to initiate an A/D conversion on channel 0
        bus.write_byte(address, control_bit)

        # Read back the converted value from the PCF8591
        analog_value = bus.read_byte(address)

        # Print out the raw analog value
        print("Analog Value:", analog_value)

        # Map the analog value to a range suitable for controlling LED brightness
        led_brightness = int((analog_value / 255.0) * 100)

        # Here you would add code to set the LED brightness using PWM or similar method.
        # For demonstration purposes, we'll just print the calculated brightness.
        print("LED Brightness (%):", led_brightness)

        time.sleep(0.1)  # Small delay between readings
```

```
except KeyboardInterrupt:
    pass  # Allow the program to exit cleanly with Ctrl+C
```

1.
2. Python LED
3.
4.
5.

2.5

Lab5

NTC Raspberry Pi PCF8591 Raspberry Pi
/

1. **NTC**
2. NTC
3. Steinhart-Hart -
4.
5. NTC 10kΩ
6. PCF8591 AIN0
7.
8. PCF8591 A/D 5V ADC 8 0~255 0~5V
9. Steinhart-Hart T
10. **Steinhart-Hart**
11. Steinhart-Hart (\frac{1}{T} = A + B\ln(R) + C(\ln(R))^3) (T) (R) (A), (B), (C)
(R_0) 10kΩ B 3950K

1.
2. Raspberry Pi T PCF8591 SDA SCL VCC GND
3. AO PCF8591 AIN0 DO VCC 5V GND
4. **I2C**
5. Raspberry Pi Preferences -> Raspberry Pi Configuration
6. Interfaces I2C OK
7.
8. Python `smbus` I2C
9. SMBus PCF8591 AIN0
10.

```
import smbus
import math
import time

# Define the I2C address of the PCF8591 and control bits
address = 0x48  # Default address for PCF8591
control_bit = 0x40  # Command to start conversion on channel 0 (AIN0)

# Constants for the thermistor calculation
R0 = 10000  # Resistance at 25°C in ohms
B = 3950  # Thermistor constant in Kelvin
T0 = 298.15  # Standard temperature in Kelvin (25°C)
Vcc = 5.0  # Supply voltage in volts

# Initialize the SMBus library
bus = smbus.SMBus(1)  # Use I2C bus 1

def read_temperature():
    try:
        # Write the control byte to initiate an A/D conversion on channel 0
        bus.write_byte(address, control_bit)

        # Read back the converted value from the PCF8591
```

```
        analog_value = bus.read_byte(address)

        # Calculate the analog voltage
        Vr = (analog_value / 255.0) * Vcc

        # Calculate the resistance of the thermistor
        Rt = R0 * Vr / (Vcc - Vr)

        # Apply the Steinhart-Hart equation to calculate temperature
        temp_kelvin = 1 / (math.log(Rt / R0) / B + 1 / T0)
        temp_celsius = temp_kelvin - 273.15

        return round(temp_celsius, 2)

    except Exception as e:
        print("Error reading temperature:", str(e))
        return None

try:
    while True:
        temperature = read_temperature()
        if temperature is not None:
            print(f"Temperature: {temperature}°C")
        else:
            print("Failed to read temperature.")

        time.sleep(1)  # Small delay between readings

except KeyboardInterrupt:
    pass  # Allow the program to exit cleanly with Ctrl+C
```

1.
2. Python
3.
4.
5.

2.6 Lab 6

2.6.1

2.6.2

2.6.3

Lab6

HC-SR04 Raspberry Pi HC-SR04
Python

1.
2. Trig 10 8 40kHz
3. Echo Echo
4. 343 / 20 Echo
5.
6. **VCC** 5V
7. **Trig**
8. **Echo**
9. **GND**
10.
11. GPIO 3.3V Echo 5V Echo

1.
2. Raspberry Pi T VCC Trig Echo GND
3. Trig Raspberry Pi GPIO17 BCM Echo GPIO18 BCM VCC 5V GND
4.
5. Python RPi.GPIO GPIO
6. `get_distance()` Trig 10 Echo Echo

7.

```
import RPi.GPIO as GPIO
import time

# Define GPIO pins for the ultrasonic sensor
TRIG = 17  # BCM numbering
ECHO = 18  # BCM numbering

# Setup GPIO mode and pin directions
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

def get_distance():
    # Ensure TRIG is low initially
    GPIO.output(TRIG, False)
    time.sleep(0.2)

    # Send a 10us pulse to TRIG
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
```

```
    GPIO.output(TRIG, False)

    # Wait for ECHO to go high
    while GPIO.input(ECHO) == 0:
        pulse_start = time.time()

    # Wait for ECHO to go low again
    while GPIO.input(ECHO) == 1:
        pulse_end = time.time()

    # Calculate the duration of the pulse
    pulse_duration = pulse_end - pulse_start

    # Convert pulse duration to distance in centimeters
    distance = pulse_duration * 17150  # Speed of sound in cm/s divided by 2 (round trip)
    distance = round(distance, 2)

    return distance

try:
    print("Measuring distance...")
    while True:
        dist = get_distance()
        print(f"Distance: {dist} cm")
        time.sleep(1)

except KeyboardInterrupt:
    print("Measurement stopped by user")

finally:
    GPIO.cleanup()  # Clean up GPIO settings before exiting
```

1.
2. Python
3.
4.
5.

2.7

Lab7

Raspberry Pi

1.
2.
3. GPIO
4.
5.
6. PFM Pulse-Frequency Modulation /
7. PWM Pulse-Width Modulation
8.
9. GPIO
10. PWM

1

1.
2. Raspberry Pi T I/O VCC GND
3. 3.3V
4.
5. Python RPi.GPIO GPIO
6. `play_tone()` GPIO
7.

```
import RPi.GPIO as GPIO
import time

# Define GPIO pin for the buzzer (BCM numbering)
BUZZER_PIN = 17  # BCM 17, physical pin 11

# Setup GPIO mode and pin direction
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUZZER_PIN, GPIO.OUT)

def play_tone(duration=0.5):
    """Play a tone using the active buzzer."""
    try:
        # Turn on the buzzer (low level trigger)
        GPIO.output(BUZZER_PIN, GPIO.LOW)
        time.sleep(duration)

        # Turn off the buzzer
        GPIO.output(BUZZER_PIN, GPIO.HIGH)
        time.sleep(0.1)  # Short pause between tones

    except KeyboardInterrupt:
        print("Stopped by user")

finally:
    GPIO.cleanup()  # Clean up GPIO settings before exiting

if __name__ == "__main__":
    print("Playing tone...")
    while True:
        play_tone()
```

2

1.
2. Raspberry Pi T I/O VCC GND
3. PWM GPIO GPIO18 BCM
4.
5. Python RPi.GPIO pigpio PWM
6. `play_music()`
7.

```
import RPi.GPIO as GPIO
import pigpio
import time

# Define GPIO pin for the passive buzzer (BCM numbering)
BUZZER_PIN = 18  # BCM 18, physical pin 12

# Initialize pigpio library
pi = pigpio.pi()

# Notes and their frequencies in Hz
NOTES = {
    'C4': 262, 'D4': 294, 'E4': 330, 'F4': 349, 'G4': 392, 'A4': 440, 'B4': 494,
    'C5': 523, 'D5': 587, 'E5': 659, 'F5': 698, 'G5': 784, 'A5': 880, 'B5': 988,
}

# A simple melody to play
MELODY = ['C4', 'D4', 'E4', 'C4', 'E4', 'D4', 'C4']

# Function to set frequency of the passive buzzer
def set_frequency(freq):
    pi.hardware_PWM(BUZZER_PIN, freq, 500000)  # Frequency, Duty cycle (50%)

def play_music(melody):
    try:
        for note in melody:
            if note in NOTES:
                set_frequency(NOTES[note])
                time.sleep(0.5)  # Duration of each note
                set_frequency(0)  # Stop sound between notes
                time.sleep(0.1)  # Short pause between notes

    except KeyboardInterrupt:
        print("Music stopped by user")

finally:
    pi.stop()  # Clean up pigpio resources
    GPIO.cleanup()  # Clean up GPIO settings before exiting

if __name__ == "__main__":
    print("Playing music...")
    play_music(MELODY)
```

1.
2. Python
3.
4.
5.

2.8 PS2

Lab8 PS2

PS2 Raspberry Pi LED PS2 X Y Z
PCF8591 LED

1. **PS2**
2. PS2 X Y 0V 5V 2.5V
3. SW 0V
4.
5. PS2 X VRX Y VRY PCF8591 AIN0 AIN1 SW GPIO
6. PCF8591 Raspberry Pi
7.
8. PCF8591 X Y
9. PCF8591 AOUT LED LED LED

1.
2. Raspberry Pi T PCF8591 PS2 SDA SCL VCC GND VRX VRY SW
3. PS2 VRX PCF8591 AIN0 VRY AIN1 SW GPIO VCC 5V GND
4. **I2C**
5. Raspberry Pi Preferences -> Raspberry Pi Configuration
6. Interfaces I2C OK
7.
8. Python `smbus` I2C
9. SMBus PCF8591 AIN0 AIN1 LED
10.

```
import smbus
import time

# Define the I2C address of the PCF8591 and control bits
address = 0x48  # Default address for PCF8591
control_bit_x = 0x40  # Command to start conversion on channel 0 (AIN0, X-axis)
control_bit_y = 0x41  # Command to start conversion on channel 1 (AIN1, Y-axis)

# Initialize the SMBus library
bus = smbus.SMBus(1)  # Use I2C bus 1

def read_joystick(axis='x'):
    """Read joystick position from specified axis."""
    if axis.lower() == 'x':
        control_bit = control_bit_x
    elif axis.lower() == 'y':
        control_bit = control_bit_y
    else:
        raise ValueError("Invalid axis. Choose 'x' or 'y'.")

    try:
        # Write the control byte to initiate an A/D conversion on selected channel
        bus.write_byte(address, control_bit)

        # Read back the converted value from the PCF8591
        analog_value = bus.read_byte(address)

        return analog_value
```

```
    except Exception as e:
        print(f"Error reading {axis}-axis:", str(e))
        return None

def map_to_brightness(value, in_min=0, in_max=255, out_min=0, out_max=100):
    """Map joystick value to LED brightness percentage."""
    return int((value - in_min) * (out_max - out_min) / (in_max - in_min) + out_min)

try:
    while True:
        x_value = read_joystick('x')
        y_value = read_joystick('y')

        if x_value is not None and y_value is not None:
            print(f"X-axis: {x_value}, Y-axis: {y_value}")

            # Calculate LED brightness based on joystick position
            led_brightness_x = map_to_brightness(x_value)
            led_brightness_y = map_to_brightness(y_value)

            # Here you would add code to set the LED brightness using PWM or similar method.
            # For demonstration purposes, we'll just print the calculated brightness.
            print(f"LED Brightness X (%): {led_brightness_x}, Y (%): {led_brightness_y}")

        time.sleep(0.1)  # Small delay between readings

except KeyboardInterrupt:
    pass  # Allow the program to exit cleanly with Ctrl+C
```

1.
2. Python LED
3.
4.
5.

2.9

Lab9

LIRC Raspberry Pi
Raspberry Pi

1.
2. LED PIN
3.
4. PIN
5.
6. **LIRC**
7. LIRC Linux Infrared Remote Control LIRC

1. **LIRC**
2. LIRC `bash`
   ```
   sudo apt-get update
   sudo apt-get install lirc
   ```
3. `/boot/config.txt` GPIO 22 23 `text`
   ```
   dtoverlay=gpio-ir,gpio_pin=22
   dtoverlay=gpio-ir-tx,gpio_pin=23
   ```
4.
5. `/etc/lirc/lirc_options.conf` LIRC `bash`
   ```
   sudo nano /etc/lirc/lirc_options.conf
   ```
   `text`
   ```
   driver = default
   device = /dev/lirc0
   ```
6.
7. Raspberry Pi `bash`
   ```
   sudo reboot
   ```
8. **IR**
9. `irw` `bash`
   ```
   irw
   ```
10.
11.
12. Python LIRC
13. Python

```
import subprocess

def listen_to_remote():
    try:
        process = subprocess.Popen(['irw'], stdout=subprocess.PIPE)

        while True:
            line = process.stdout.readline().decode('utf-8').strip()
            if not line:
                break

            print("Received IR command:", line)
```

```
    except KeyboardInterrupt:
        print("\nListening stopped.")

if __name__ == "__main__":
    print("Listening for IR commands...")
    listen_to_remote()
```

1.
2. Python
3.
4.
5.

2.10

Lab10

Raspberry Pi GPIO
LED

1.
2. `GPIO.add_event_detect()` GPIO
   - `channel` GPIO
   - `edge` `GPIO.RISING` `GPIO.FALLING` `GPIO.BOTH`
   - `callback`
   - `bouncetime`
3.
4. `GPIO.wait_for_edge()` CPU
5.
6. “ ”

1.
2. Raspberry Pi T SIG(S) VCC GND
3. SIG(S) Raspberry Pi GPIO23 BCM VCC 5V GND
4. LED LED GPIO17 LED GPIO27
5.
6. Python RPi.GPIO GPIO
7. `setup_gpio()` GPIO
8. `button_pressed_callback()` LED
9.

```
import RPi.GPIO as GPIO
import time

# Define GPIO pins for the LED and button (BCM numbering)
RED_LED_PIN = 17  # BCM 17, physical pin 11
GREEN_LED_PIN = 27  # BCM 27, physical pin 13
BUTTON_PIN = 23  # BCM 23, physical pin 16

def setup_gpio():
    """Setup GPIO mode and pin directions."""
    GPIO.setmode(GPIO.BCM)

    # Setup LEDs as output
    GPIO.setup(RED_LED_PIN, GPIO.OUT)
    GPIO.setup(GREEN_LED_PIN, GPIO.OUT)

    # Setup button as input with pull-up resistor
    GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def button_pressed_callback(channel):
    """Callback function called when the button is pressed."""
    if channel == BUTTON_PIN:
        print("Button pressed!")

        # Toggle between red and green LED
        if GPIO.input(RED_LED_PIN):
            GPIO.output(RED_LED_PIN, GPIO.LOW)
            GPIO.output(GREEN_LED_PIN, GPIO.HIGH)
        else:
```

```
            GPIO.output(RED_LED_PIN, GPIO.HIGH)
            GPIO.output(GREEN_LED_PIN, GPIO.LOW)

try:
    setup_gpio()

    # Add event detection on the button pin with debouncing
    GPIO.add_event_detect(BUTTON_PIN, GPIO.FALLING, callback=button_pressed_callback, bouncetime=200)

    print("Waiting for button press...")
    while True:
        time.sleep(1)  # Keep script running to allow callbacks to work

except KeyboardInterrupt:
    print("\nProgram stopped by user")

finally:
    GPIO.cleanup()  # Clean up GPIO settings before exiting
```

1.
2. Python LED
3. `bouncetime`
4.
5. ...
6.
7.
8.