```
import spidev
import time
import threading
import RPi.GPIO as GPIO
import paho.mqtt.client as mqtt
import logging

# ---Logging Setup---#
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

# ---MQTT Setup---#
MQTT_BROKER = "iot.kmitlnext.com"
MQTT_PORT = 9001
MQTT_TOPIC_R2 = "phon/log"
MQTT_TOPIC_LDR = "phon/ldr"
MQTT_TOPIC_DIM = "phon/dim"
MQTT_TOPIC_CONTROL = "led/control"
MQTT_USERNAME = "kmitliot"
MQTT_PASSWORD = "KMITL@iot1234"

client = mqtt.Client(transport="websockets")
client.username_pw_set(MQTT_USERNAME, MQTT_PASSWORD)

# ---SPI Setup---#
spi = spidev.SpiDev()
spi.open(0, 0)
```

```
spi.max_speed_hz = 1000000

# ---GPIO Setup---#
GPIO.setmode(GPIO.BCM)
GREEN_LED = 17
RED_LED = 18
GPIO.setup(GREEN_LED, GPIO.OUT)
GPIO.setup(RED_LED, GPIO.OUT)

# Initialize PWM for Red LED
pwm = GPIO.PWM(RED_LED, 1000)
pwm.start(0)

# ---Global Variables---#
previous_r2 = None
previous_green_led = None
previous_duty_cycle = None

# ---Helper Functions---#
def read_spi(channel):
    raw = spi.xfer2([1, (channel << 4) | 0x80, 0])
    return ((raw[1] & 3) << 8) | raw[2]

def calculate_voltage(adc_value, v_ref=3.3, resolution=1024):
    return (adc_value / resolution) * v_ref
```

```
# ---MQTT Handlers---#
def on_connect(client, userdata, flags, rc):
    logging.debug(f"Connected to MQTT broker with result code {rc}")
    client.subscribe(MQTT_TOPIC_CONTROL)

def on_message(client, userdata, msg):
    try:
        message = msg.payload.decode()
        logging.debug(f"Received message on {msg.topic}: {message}")

        if msg.topic == MQTT_TOPIC_CONTROL:
            if message.lower() == "green_on":
                GPIO.output(GREEN_LED, 1)
                logging.debug("Green LED turned ON via MQTT")
            elif message.lower() == "green_off":
                GPIO.output(GREEN_LED, 0)
                logging.debug("Green LED turned OFF via MQTT")
            elif message.startswith("dim:"):
                try:
                    brightness = float(message.split(":")[1])
                    pwm.ChangeDutyCycle(brightness)
                    logging.debug(f"Red LED brightness set to {brightness}% via MQTT")
                except ValueError:
                    logging.error("Invalid brightness value received in MQTT message")
    except Exception as e:
        logging.error(f"Error processing MQTT message: {e}")
```

```
def mqtt_loop():
    client.on_connect = on_connect
    client.on_message = on_message
    client.connect(MQTT_BROKER, MQTT_PORT)
    client.loop_forever()

# ---Thread Functions---#

# In the lab_1_r2 function
def lab_1_r2():
    global previous_r2
    ch = 0
    while True:
        adc_value = read_spi(ch)
        voltage = calculate_voltage(adc_value)
        resistance = ((voltage * 1000) / (3.3 - voltage)) *10  # R2 = (Vout * R1) / (Vin - Vout)
        current = voltage / resistance

        # Log the details
         logging.debug(f"Voltage: {voltage:.2f} V | Resistance: {resistance:.2f} Ohm | Current: {current}
mA")

        client.publish(MQTT_TOPIC_R2, f"Resistance: {resistance:.2f} Ohm")

        previous_r2 = resistance
```

```
        # Check if the resistance has changed by more than 1kÎ© or if it is the first reading
        if previous_r2 is None or abs(resistance - previous_r2) > 1000:
            logging.debug(f"Resistance change detected: {resistance:.2f} Ohm")
            # Publish new resistance value when it changes by more than 1kÎ©
            client.publish(MQTT_TOPIC_R2, f"Resistance: {resistance:.2f} Ohm")
            previous_r2 = resistance

        # Sleep before next reading
        time.sleep(2)

def lab_2_ldr():
    global previous_green_led
    ch_ldr = 1
    while True:
        adc_value = read_spi(ch_ldr)
        voltage = calculate_voltage(adc_value)
        led_status = 1 if voltage > 2 else 0
        GPIO.output(GREEN_LED, led_status)

        if previous_green_led is None or previous_green_led != led_status:
            status = "ON" if led_status else "OFF"
            logging.debug(f"Green LED status changed to {status}")
            # Publish the LED status change (ON/OFF)
            client.publish(MQTT_TOPIC_LDR, f"{status}")
            previous_green_led = led_status
```

```
        time.sleep(2)


def lab_3_potentiometer():
    global previous_duty_cycle
    ch_pot = 2
    while True:
        adc_value = read_spi(ch_pot)
        voltage = calculate_voltage(adc_value)
        duty_cycle = (adc_value / 1023) * 100
        pwm.ChangeDutyCycle(duty_cycle)

        logging.debug(f"Potentiometer: Voltage = {voltage:.2f} V | Duty Cycle = {duty_cycle:.2f}%")

        if previous_duty_cycle is None or abs(duty_cycle - previous_duty_cycle) > 1:
            logging.debug(f"Red LED brightness change detected: {duty_cycle:.2f}%")
            # Publish new brightness value when it changes by more than 1%
            client.publish(MQTT_TOPIC_DIM, f"Red LED Brightness: {duty_cycle:.2f}%")
            previous_duty_cycle = duty_cycle

        time.sleep(1)

# ---Main Function---#
if __name__ == '__main__':
    try:
        # Start MQTT in a separate thread
        thread_mqtt = threading.Thread(target=mqtt_loop, daemon=True)
```

```
    thread_mqtt.start()

    # Create and start other threads
    thread_lab1 = threading.Thread(target=lab_1_r2, daemon=True)
    thread_lab2 = threading.Thread(target=lab_2_ldr, daemon=True)
    thread_lab3 = threading.Thread(target=lab_3_potentiometer, daemon=True)

    thread_lab1.start()
    thread_lab2.start()
    thread_lab3.start()

    # Keep the main program running
    while True:
        time.sleep(0.1)

except KeyboardInterrupt:
    spi.close()
    pwm.stop()
    GPIO.cleanup()
    client.disconnect()
    logging.info("SPI closed, MQTT disconnected, and GPIO cleaned up. Exiting...")
```