Lab2 Write Up

Airi Kokuryo

November 1, 2023

(somehow the LaTeX date time is not set properly, but it is October 31, 2023)

1 Approach to Sensor Connection

In the process of approaching a connection between the Raspberry Pi and the 3 various sensors, the first step was to read the document thoroughly to understand which sensor cords belonged to either Ground, VCC, I2C SCL(Serial Clock), or I2C SDA(Serial Data). The information about what the sensors were used for and how they output information had been helpful in the course of understanding the sensor connections as well. This approach has succeeded in starting with a decent guess on how the sensors are connected. Writing down the important information about each sensor has helped shorten the time consumed in brainstorming.

a. The documentation was missing color information about the Soil Sensor cords, however, I managed to find out which belonged to I2C SDA and SCL from the general knowledge that the clock uses green and by actually trying it out with the test Python program. Since it was my first attempt at using the breadboard, I also had to research how the breadboard works.

   - Reference1: How to use an wind sensor
   - Reference2: Breadboard tutorial
   - Reference3: Breadboard YouTube tutorial

b. Picture of the Wiring

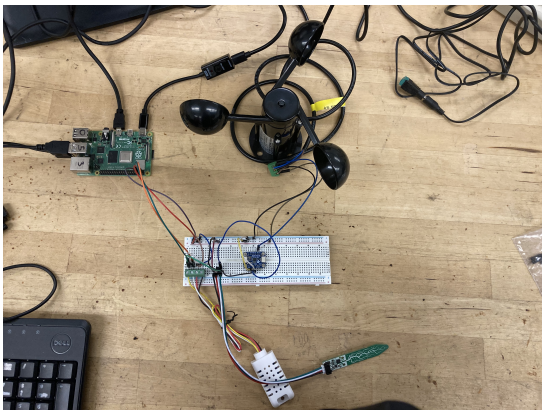

Figure 1: Wiring on the Breadboard

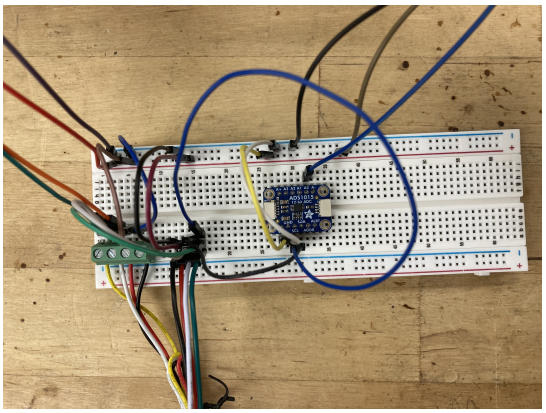

Figure 2: Wiring Raspberry Pi to Multiple Sensors

2 Interface Pi to Sensors

- How the Pi is interfaced with the Sensors

  The Raspberry Pi is connected to the sensors through the 40-pin GPIO headers. Pins 3,4,5,6 from [Figure 1] are each used for SDA, VCC, SCL, and Ground, and this would be connected to the same cable type as the other sensors using the female-to-male jumper wires and the breadboard.

- Pros and cons of I2C protocol compared to the SPI and UART communication protocols

  Comparing I2C to SPI, I2C is a two-wired interface while SPI is a 4-wired interface. This simplicity is the advantage over SPI, especially when

adding multiple sensor devices to the bus. Meanwhile, SPI is the best speed out of the three communication protocols.

I2C to UART are both a two-wire interface with an error handling system. The comparison is in synchronicity, as I2C is a synchronous protocol and UART is an asynchronous protocol. Also, I2C works faster than UART but shorter in distance.

a Sensor I2C address and dealing with addressing conflicts

The soil sensor has the default i2C address, 0x36, and a selection between 0x36 and 0x39 when shorting AD0 and/or AD1. The temperature and humidity sensor has the 0x44 default address and a 0x45 address when ADDR is connected to logic high. The wind sensor does not have its own i2C address, however, the ADS1015 has a different slave address for each Ground(1001000), VDD(1001001), SDA(1001010), and SCL(1001011). If there are addressing conflicts, this could be avoided by using the other address which is given by shorting.

Reference: I2C vs SPI vs UART

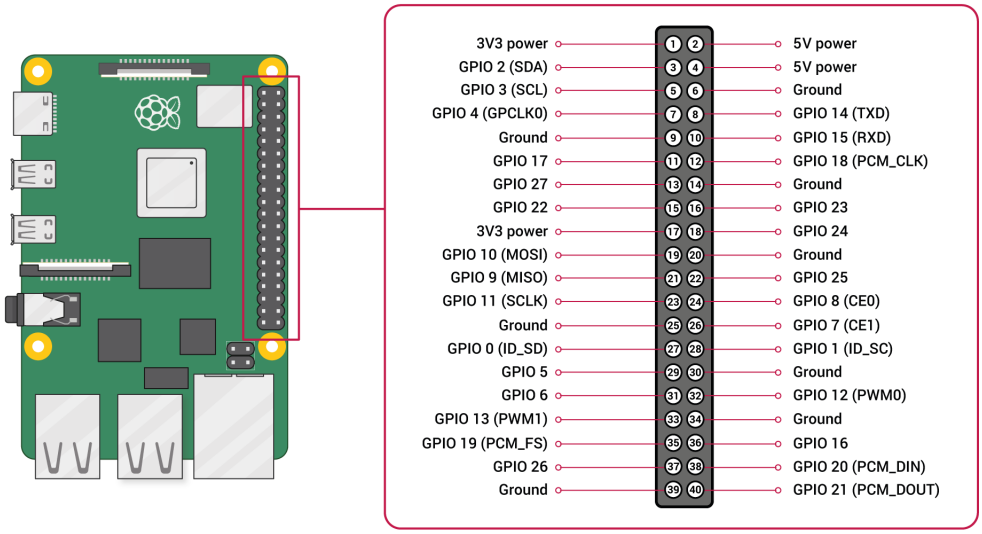

Figure 3: Raspberry Pi 40-Pin GPIO Headers

3 Sensor Polling

The implementation of serial polling was done by connecting each cable with the same input/output. For example, the SDA cables would be connected in the same row with the electric current, powered by VCC cables. Ground and SCL would also be connected in the same way, creating a serial bus including the Raspberry Pi and the sensors. One of the difficulties I encountered was coming from the insecure wire connections caused by the breadboard. Some parts of the breadboard were not working properly but were found and fixed by debugging with logs, connecting each sensor one at a time, and changing devices. The sensor polling was performed by running the Python program on the Raspberry Pi, which used the I2C protocol for getting serial data and clocks. Adafruit libraries(SHT30, seesaw, ADS1x15) were used for getting sensor information, and SimpleIO library was used for the wind sensor. Soil sensor used board to get the sensor information, as where the other two sensors used busio for I2C outputs. The codes are shown below:

```
import time
import board
import busio

from adafruit_seesaw.seesaw import Seesaw
import adafruit_sht31d
import adafruit_ads1x15.ads1015 as ADS
from adafruit_ads1x15.analog_in import AnalogIn

import simpleio

from datetime import datetime
from datetime import date

I2C = busio.I2C(board.SCL, board.SDA)

#SoilMoisture.py
def SoilMoisture():
    i2c = board.I2C()
    ss = Seesaw(i2c, addr=0x36)
    temperature = ss.get_temp()
    moisture = ss.moisture_read()
    soilMoisture = "Soil Moisture:" + str(moisture
        ) + "\n Soil Temperature:" + str(
        temperature) + "\n"
#     print('Soil Moisture:'+ str(moisture))
#     print('Soil Temperature:'+ str(temperature))
    return soilMoisture
```

```
#getting information from SHT30 Temperature/
    Humidity sensor
def TempHumidity():
    sensor = adafruit_sht31d.SHT31D(I2C)
#     print('Humidity: {0}%'.format(sensor.
    relative_humidity))
#     print('Temperature: {0}C'.format(sensor.
    temperature))
    tempHumidity = "Temperature:" + str(sensor.
        temperature) + "\n" + "Humidity:" + str(
        sensor.relative_humidity) + "\n"
    return tempHumidity

#getting information from the ADS1015 12-bit ADC
def WindSpeed():
     ads = ADS.ADS1015(I2C)
     chan = AnalogIn(ads, ADS.P0)
     map_speed = simpleio.map_range(chan.voltage,
         0.40, 0.78, 0, 32) #map min and max
     #print('Wind Speed:' + str(wind_speed))
     windSpeed = str(map_speed)
     return windSpeed

#getting time and date
def DateTime():
    now = datetime.now()
    current_time = now.strftime("%H:%M:%S")
    current_date = now.strftime("%Y-%m-%d")
    dateTime = current_date + "\n" + current_time
        + "\n"
    return dateTime

def main():
    with open('/home/akokuryo/Documents/polling-
        log.txt', 'w') as f:
         f.write("AiriKokuryo")

    while True:
        print("write start")
        returnDateTime = DateTime()
        returnSoilMoisture = SoilMoisture()
        returnTempHumidity = TempHumidity()
        returnWindSpeed = WindSpeed()
        time.sleep(5)
        print("complete")
```

```
        with open('/home/akokuryo/Documents/
            polling-log.txt', 'a') as f:
             f.write(returnDateTime)
             f.write(returnSoilMoisture)
             f.write(returnTempHumidity)
             f.write(returnWindSpeed)
             f.write("\n ")

main()
```

a Differences between each sensor and how these underlying behaviors influence the implementation of the polling method

The wind sensor has an analog output, whereas other sensors do not. Therefore the wind sensor must be connected to the ADS1015, which will convert this input to I2C SDA and SCL outputs. Meanwhile, the soil sensor and the temperature/humidity sensor have an I2C output so these could be connected directly to the pi without conversion of the data type.

b Does this method poll all sensors at exactly the same time?

No, because all the sensors are polled asynchronously, especially the wind sensor which needs the conversion time to convert its data from analog to digital.

4 Implementing Cooperative Multitasking

- How to implement cooperative multitasking

The cooperative multitasking is implemented by using the asyncio package from Python. The program uses an event loop responsible for managing the asynchronous functions and uses the asyncio.gather() function to collect and run multiple concurrent functions. The time is scheduled using the asyincio.sleep(interval). The data is logged through using loggers and configuring it to log messages to a file.

a How does cooperative multitasking compare to parallelism? How does it differ from it? Who runs the tasks in cooperative multitasking, and can two tasks run simultaneously?

Cooperative multitasking runs in a single thread whereas parallelism runs in multiple threads. This means that parallel processing allows a situation where multiple tasks proceed simultaneously whereas cooperative multitasking runs only one task at any given moment. Therefore in cooperative multitasking, the scheduler determines which tasks to run next, and can

not run two tasks simultaneously. On the other hand, cooperative multitasking may be more simple as it does not involve complex synchronization mechanisms or context switching.

b What projects would benefit from cooperative multitasking?

From the point of simplicity, cooperative multitasking would have a benefit over projects such as embedded systems and IoT devices. These projects have limited resources and real-time constraints, and also need to be managed without the overhead of the operating system.

5 Cooperative Multitasking Within Several Different Machines

- Consider if you extended cooperative multitasking to be across several different machines.

a Why we need a timer process

We might need a timer process to centralize the timing control in highly distributed systems where variable latency occurs due to network communication or different machine capabilities. Also, this is important in ensuring that the consistency is kept by operating based on the same time standard.

b What advantages and disadvantages does cooperative multitasking have compared to serial polling in our lab setup?

Cooperative multitasking can be advantageous over serial polling to the point where serial polling has to make a request every time to get the data. This may slow down the process Cooperative multitasking does not require any request and will work under the scheduling scheme. Also, it may take time to poll when there are too many devices to check. Oppositely, Cooperative multitasking may have a disadvantage over serial polling when the task does not yield control due to errors, since it would cause the CPU to monopolize and prevent other tasks from executing.

6 Conclusion

In part 1, I assembled and connected several sensors to the Raspberry Pi using the I2C communication protocol. Then in part 2, I created the Python script to periodically poll all sensors every 5 seconds and log their measurements into a file. At last in part 3, I was introduced to cooperative multitasking using asyncio and also created the script to read data from each sensor and log it out using the logger.