Temperature and Humidity Sensor with Raspberry Pi Pico

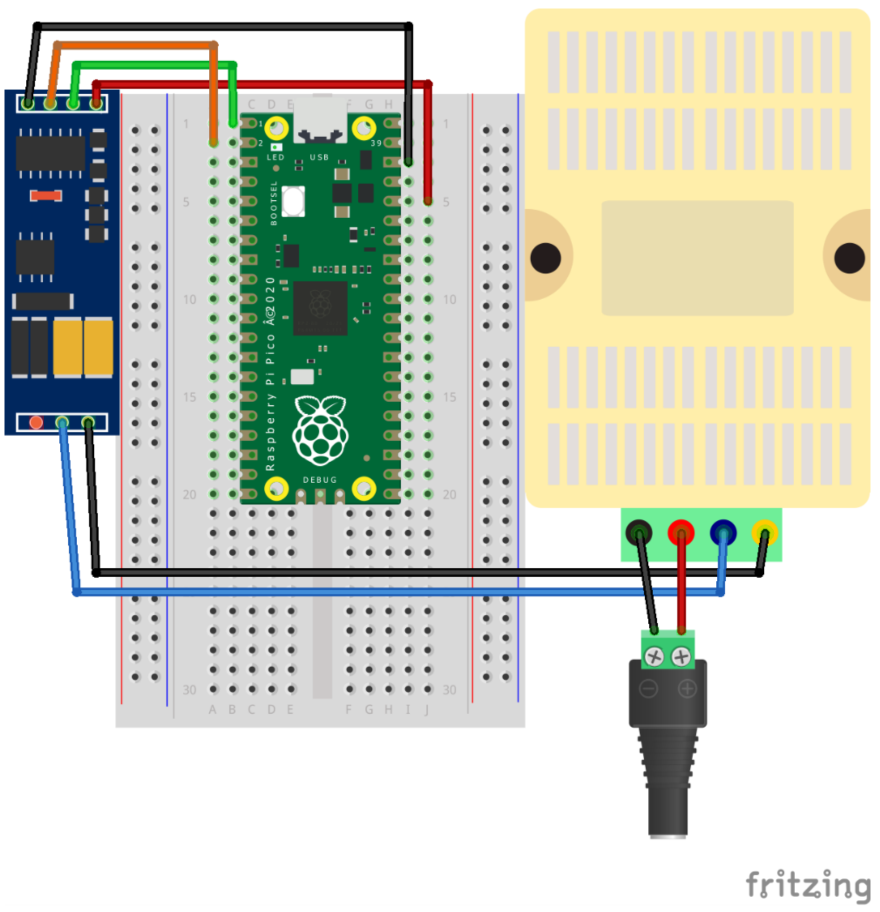
Requirement

- PKTH100B-CZ1
- Raspberry Pi Pico
- RS485 Auto Direction Module
- 12 V DC Adapter
- แจ็กแยกขั้วไฟ ตัวเมีย

Language: Micropython

IDE: Thonny

Wiring Diagram

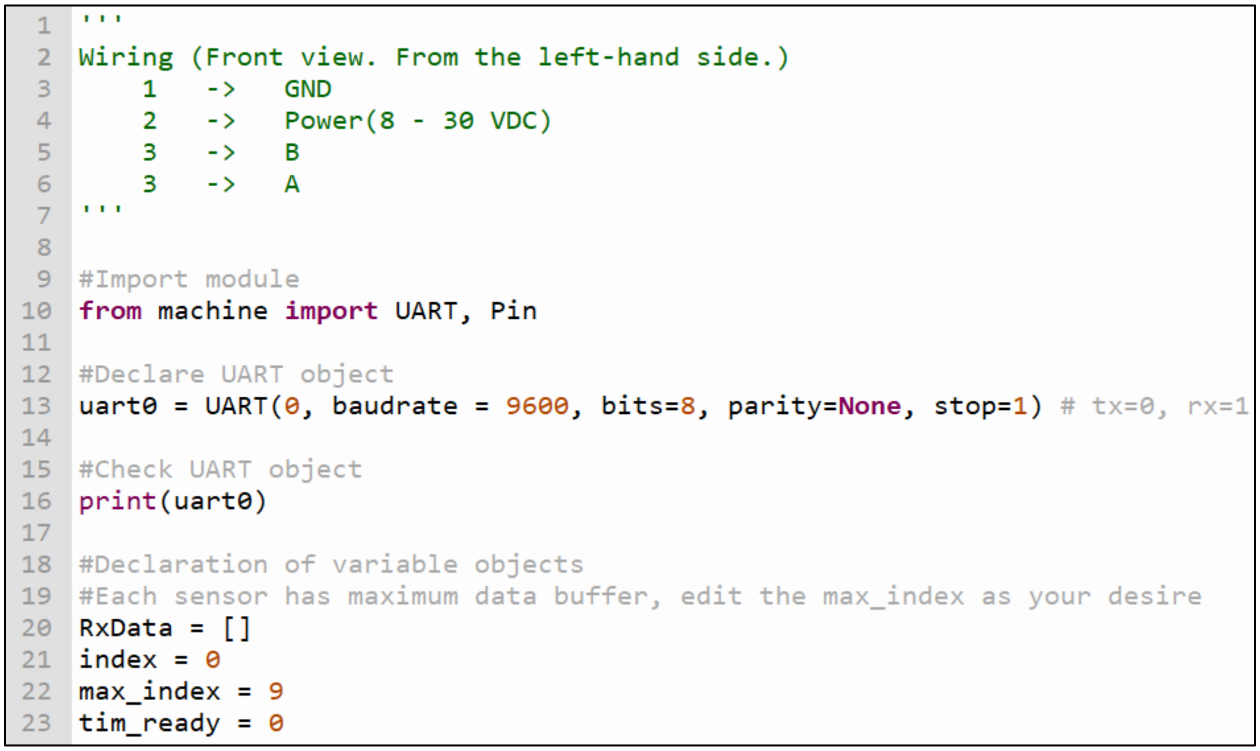


Pico		**RS485**		**Temp & Hum**		**12 VDC Adapter**
3V3	↔	VCC		VCC	↔	+
UART0 TX	↔	TXD		GND	↔	-
UART0 RX	↔	RXD				
GND	↔	GND				
		A	↔	A		
		B	↔	B		

Coding

The example program will show the value of temperature and humidity every 1 second.

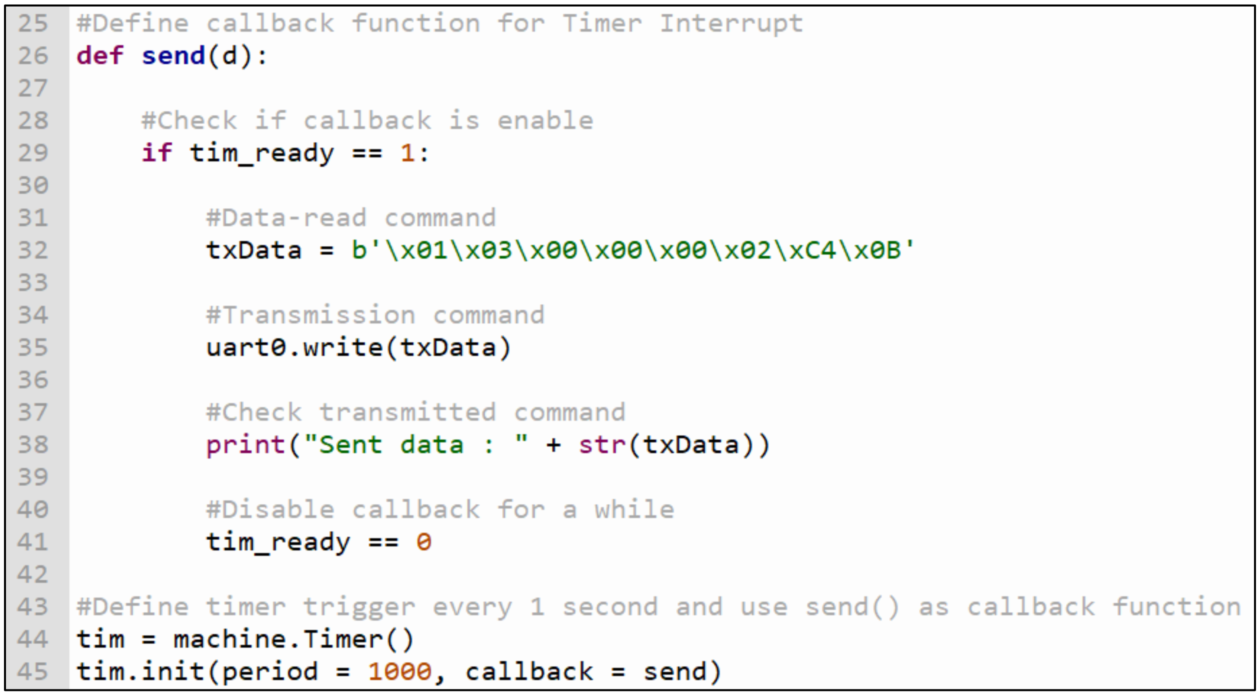
1. Import the module that we use in this program. In this case we only use “machine” as module. Import “UART” and “Pin” to use UART and GPIO of our Pico. Declare objects that we use. In this program, we will name in as “RxData”, “index”, “max_index” and “tim_ready”. RxData” is the empty list that we use to store the buffer that sensor has sent. “index” is the counting variable that use to check and limit data that Pico receive. “max_index” is variable that use to define the maximum value that “index” can reach. For this Soil Moisture sensor, normal maximum index is equal to 7 according to datasheet. “tim_ready” is variable that use to check if program is ready or not. 1 means the program is ready to request data. 0 means the program is not ready to request data. “timeout” is variable that use to check if the time has passed for 10 second. All declaration is already show in the picture below.



```python
'''
Wiring (Front view. From the left-hand side.)
    1   ->   GND
    2   ->   Power(8 - 30 VDC)
    3   ->   B
    3   ->   A
'''

#Import module
from machine import UART, Pin

#Declare UART object
uart0 = UART(0, baudrate = 9600, bits=8, parity=None, stop=1) # tx=0, rx=1

#Check UART object
print(uart0)

#Declaration of variable objects
#Each sensor has maximum data buffer, edit the max_index as your desire
RxData = []
index = 0
max_index = 9
tim_ready = 0
```

(Pic.1: Module import and object declaration)

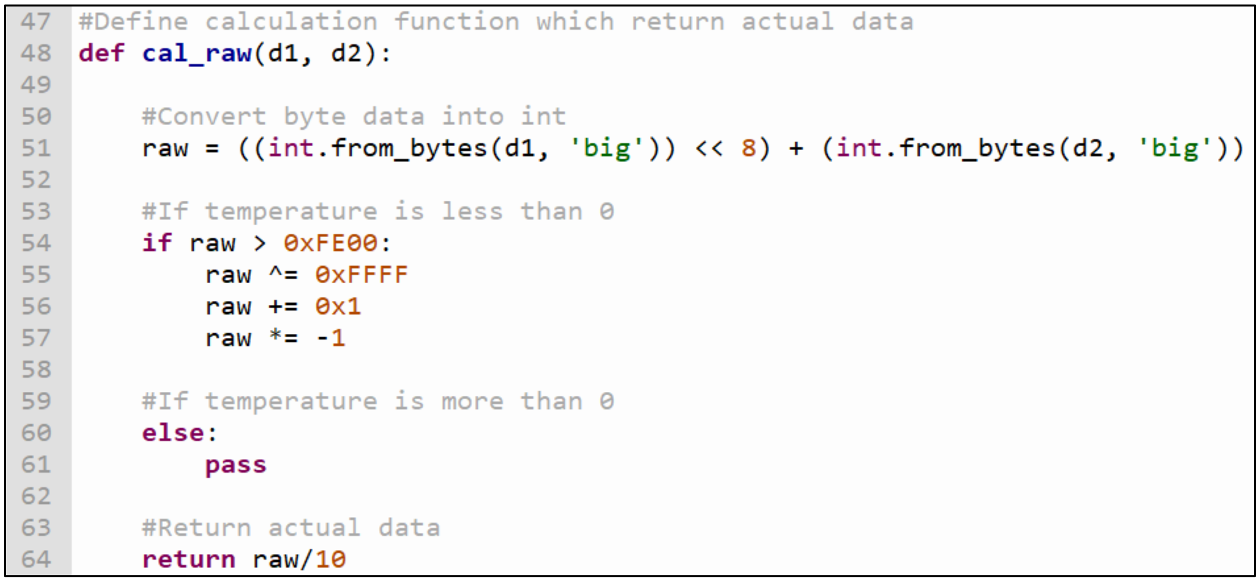
2. To get data from the sensor via Modbus RTU Protocol, we need to send the command to request every time. This program will use timer interrupt to send data-request command every 1 second. Remember that every interrupt needs a callback function. So, we define the callback function of timer interrupt and then declare timer object as the picture below (Note that every callback function needs 1 parameter).



```
#Define callback function for Timer Interrupt
def send(d):

    #Check if callback is enable
    if tim_ready == 1:

        #Data-read command
        txData = b'\x01\x03\x00\x00\x00\x02\xC4\x0B'

        #Transmission command
        uart0.write(txData)

        #Check transmitted command
        print("Sent data : " + str(txData))

        #Disable callback for a while
        tim_ready == 0

#Define timer trigger every 1 second and use send() as callback function
tim = machine.Timer()
tim.init(period = 1000, callback = send)
```

(Pic.2: Callback function definition and timer-interrupt object declaration)

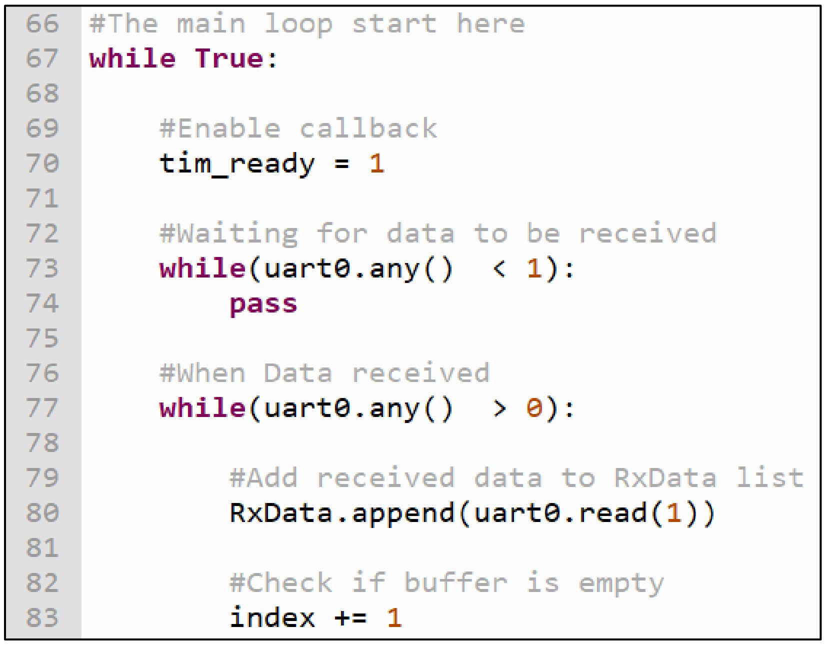
3. Define calculation function name “cal_raw” which return actual value of temperature and humidity when we pass parameters to function. According to datasheet, there are two cases of temperature value. First is when value is between 0 and 120 degree Celsius. In this case we can calculate by sum all data after convert byte data into int and use bitwise operation then divide it by 10. And another case is when value is under 0 degree Celsius. In this case, data we received will express in tw0’s complement form. First, sum all data after convert byte data into int and use bitwise operation. Next, do two’s complement. Then, multiply value by -1 and divide by 10. For humidity, calculate like the first case of temperature calculation. Coding procedure is shown in the picture below



```
47 #Define calculation function which return actual data
48 def cal_raw(d1, d2):
49
50     #Convert byte data into int
51     raw = ((int.from_bytes(d1, 'big')) << 8) + (int.from_bytes(d2, 'big'))
52
53     #If temperature is less than 0
54     if raw > 0xFE00:
55         raw ^= 0xFFFF
56         raw += 0x1
57         raw *= -1
58
59     #If temperature is more than 0
60     else:
61         pass
62
63     #Return actual data
64     return raw/10
```

(Pic.3: Conversion function definition and data calculation)

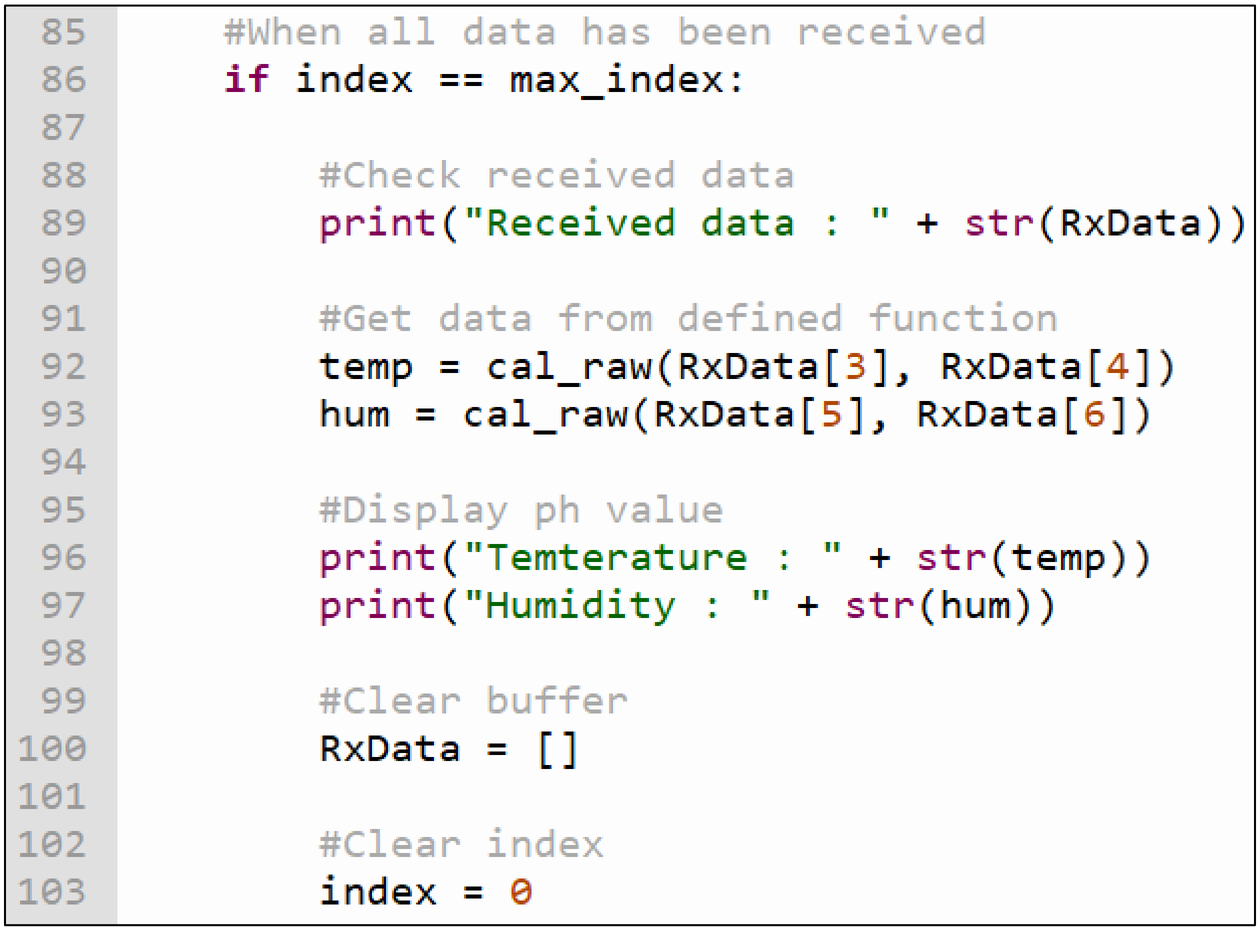
4. Create an infinity loop as a main program (all operation will run in the loop). In the picture below, we will set “tim_ready” to 1 to announce that program is ready to request data, then use loop (in line 65) to wait for data. If the data has been received, Program will be running to another loop (in line 69). This loop will add the data buffer into “RxData” and “index” will increase.



```
66 #The main loop start here
67 while True:
68
69     #Enable callback
70     tim_ready = 1
71
72     #Waiting for data to be received
73     while(uart0.any()  < 1):
74         pass
75
76     #When Data received
77     while(uart0.any()  > 0):
78
79         #Add received data to RxData list
80         RxData.append(uart0.read(1))
81
82         #Check if buffer is empty
83         index += 1
```

(Pic.4: Inside the main loop)

5. In the picture below, temperature and humidity value will be calculated after we receive all bytes of message. According to datasheet, the actual value of temperature is the 3rd and 4th byte, actual value of humidity is the 5th and 6th byte we received. Pass the data into "cal_raw" to get the actual value of temperature and humidity.



```
    #When all data has been received
    if index == max_index:

        #Check received data
        print("Received data : " + str(RxData))

        #Get data from defined function
        temp = cal_raw(RxData[3], RxData[4])
        hum = cal_raw(RxData[5], RxData[6])

        #Display ph value
        print("Temterature : " + str(temp))
        print("Humidity : " + str(hum))

        #Clear buffer
        RxData = []

        #Clear index
        index = 0
```

(Pic.5: Getting actual data from defined functions)