Soil Rain Gauge Sensor with Raspberry Pi Pico

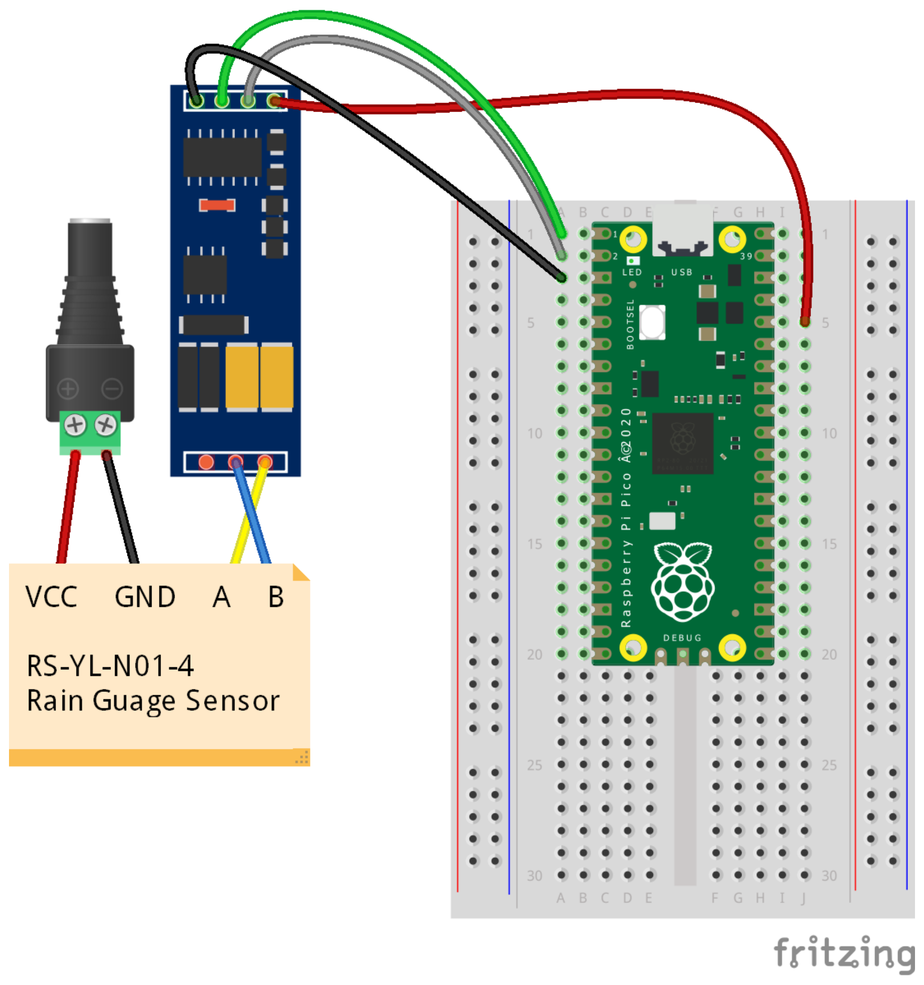
Requirement

- RS-YL-N01-4
- Raspberry Pi Pico
- RS485 Auto Direction Module
- 12 V DC Adapter
- แจ็กแยกขั้วไฟ ตัวเมีย

Language: Micropython

IDE: Thonny

Wiring Diagram

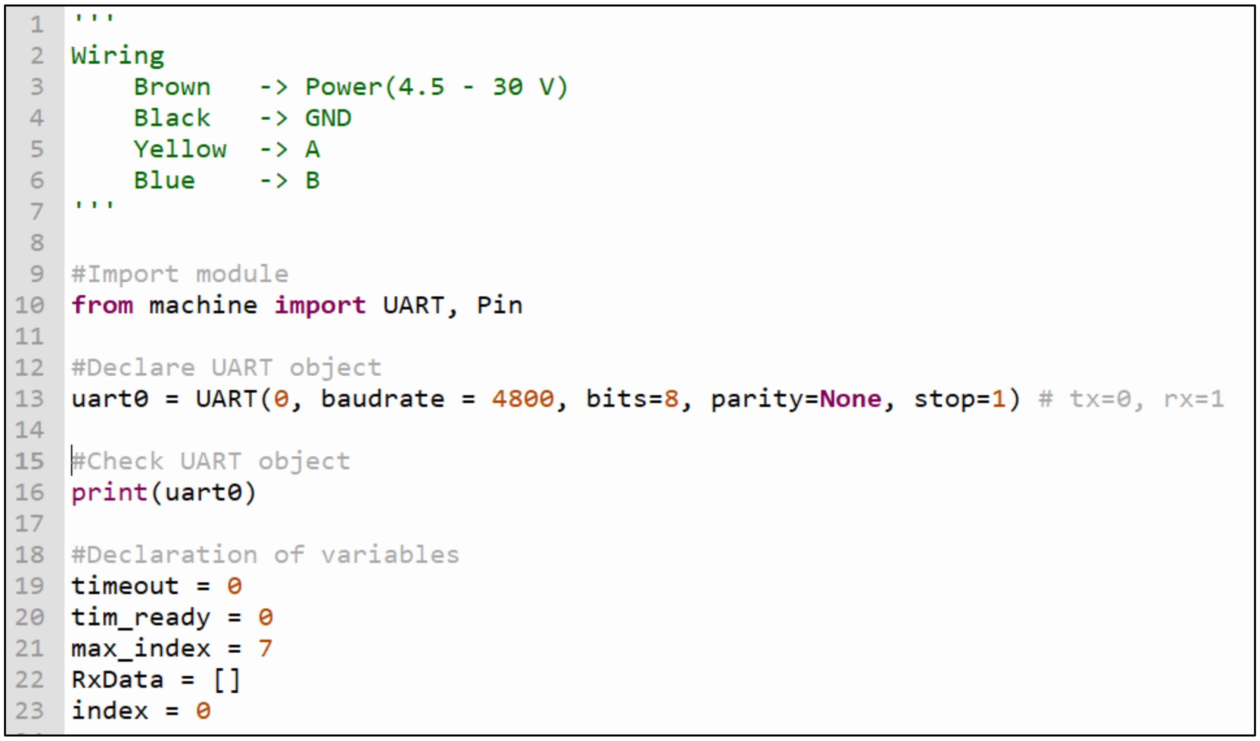


Pico		**RS485**		**PH Sensor**		**12 VDC Adapter**
3V3	↔	VCC		VCC	↔	+
UART0 TX	↔	TXD		GND	↔	-
UART0 RX	↔	RXD				
GND	↔	GND				
		A	↔	A		
		B	↔	B		

Coding

The example program will show the value of rainfall every 1 second and clear holding data every 10 second.

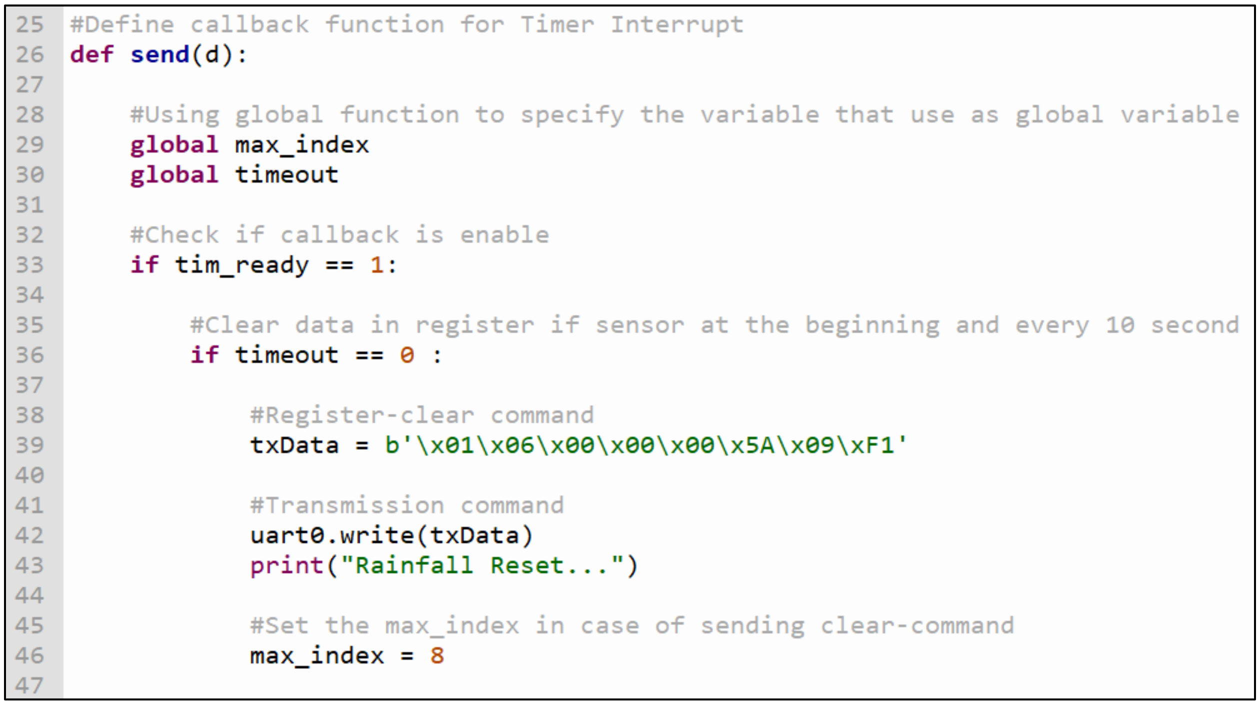
1. Import the module that we use in this program. In this case we only use “machine” as module. Import “UART” and “Pin” to use UART and GPIO of our Pico. Declare objects that we use. In this program, we will name in as “RxData”, “index”, “max_index” and “tim_ready”. RxData” is the empty list that we 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.



```
'''
Wiring
    Brown   -> Power(4.5 - 30 V)
    Black   -> GND
    Yellow  -> A
    Blue    -> B
'''

#Import module
from machine import UART, Pin

#Declare UART object
uart0 = UART(0, baudrate = 4800, bits=8, parity=None, stop=1) # tx=0, rx=1

#Check UART object
print(uart0)

#Declaration of variables
timeout = 0
tim_ready = 0
max_index = 7
RxData = []
index = 0
```

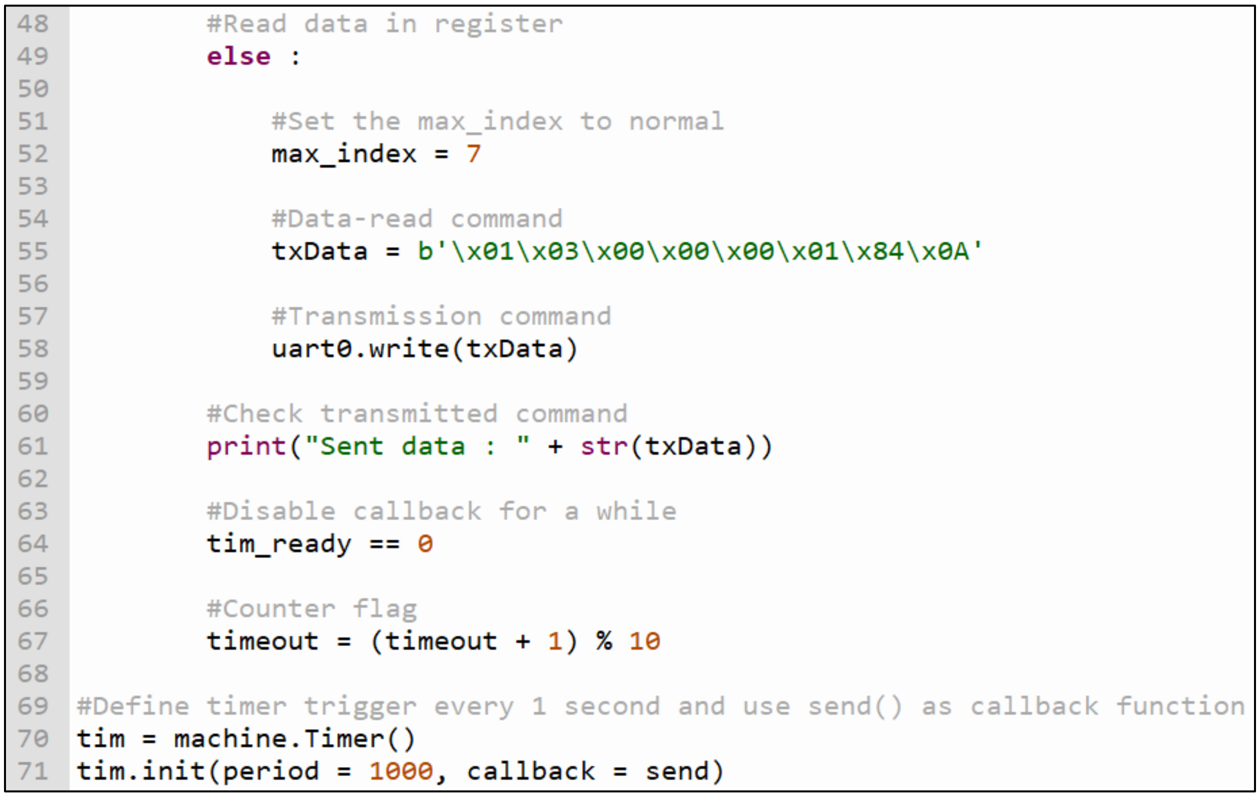
(Pic.1: Module import and object declaration)

2. To get or set 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 and register-clear command every 10 second. In case of sending register-clear command, we will receive 8 bytes of data instead of 7. So, we have to set the “max_index” to 8 to prevent buffer overload in object “RxData”. 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).



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

    #Using global function to specify the variable that use as global variable
    global max_index
    global timeout

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

        #Clear data in register if sensor at the beginning and every 10 second
        if timeout == 0 :

            #Register-clear command
            txData = b'\x01\x06\x00\x00\x00\x5A\x09\xF1'

            #Transmission command
            uart0.write(txData)
            print("Rainfall Reset...")

            #Set the max_index in case of sending clear-command
            max_index = 8

```

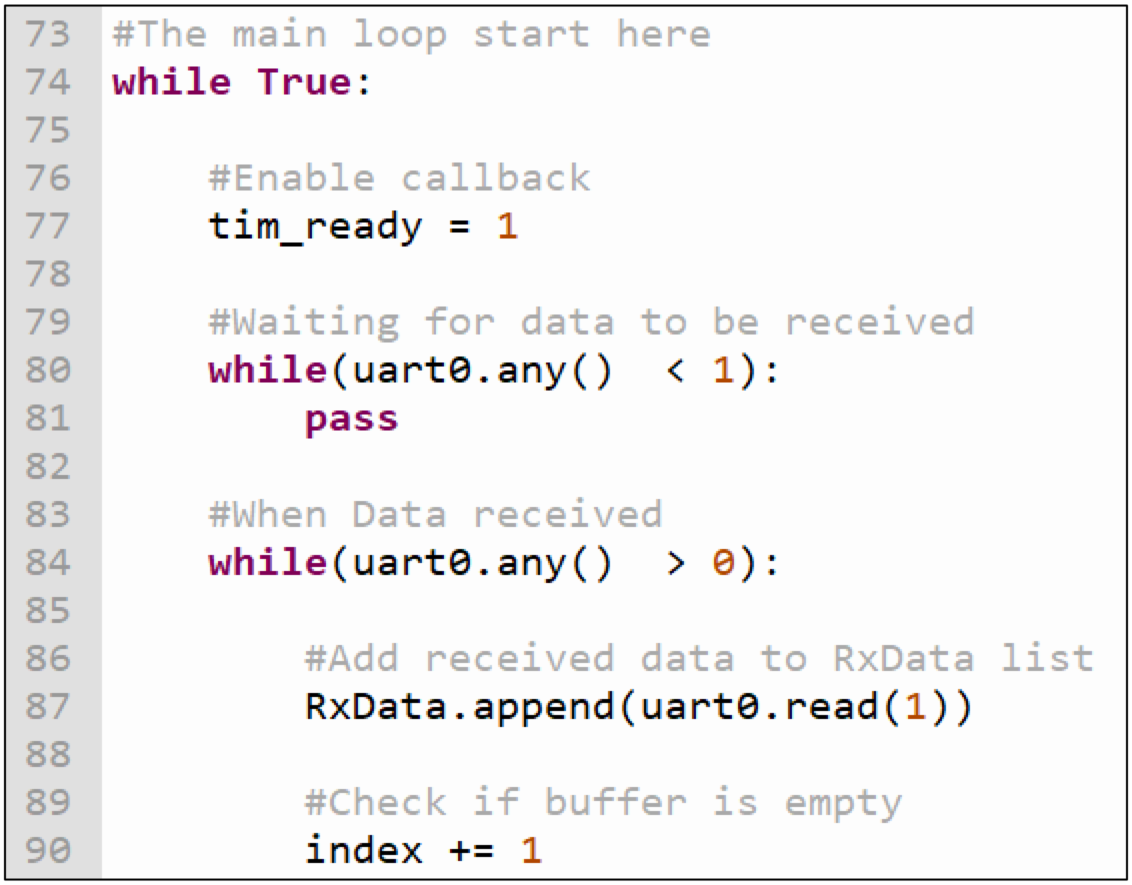
(Pic.2: Callback function definition and timer-interrupt object declaration 1)



```python
        #Read data in register
        else :

            #Set the max_index to normal
            max_index = 7

            #Data-read command
            txData = b'\x01\x03\x00\x00\x00\x01\x84\x0A'

            #Transmission command
            uart0.write(txData)

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

        #Disable callback for a while
        tim_ready == 0

        #Counter flag
        timeout = (timeout + 1) % 10

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

(Pic.3: Callback function definition and timer-interrupt object declaration 2)

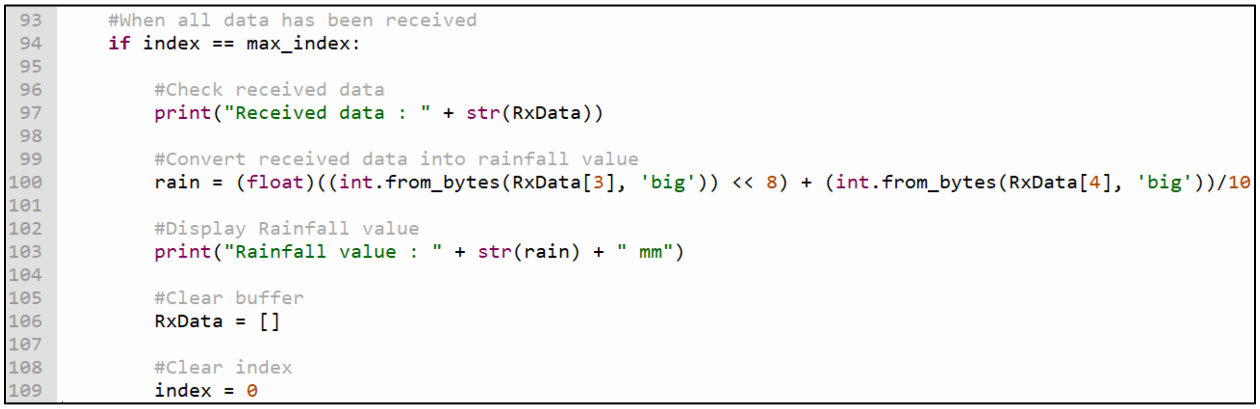
3. Create an infinity loop as a main program (all operation will run in the loop). In the picture below, we will set “tim_ready” to 1 to announce that program is ready to request data, then use loop (in line 80) to wait for data. If the data has been received, Program will be running to another loop (in line 84). This loop will add the data buffer into “RxData” and “index” will increase.



```
73 #The main loop start here
74 while True:
75 
76     #Enable callback
77     tim_ready = 1
78 
79     #Waiting for data to be received
80     while(uart0.any()  < 1):
81         pass
82 
83     #When Data received
84     while(uart0.any()  > 0):
85 
86         #Add received data to RxData list
87         RxData.append(uart0.read(1))
88 
89         #Check if buffer is empty
90         index += 1
```

(Pic.4: Inside the main loop)

4. In the picture below, rainfall value will be calculated after we receive all bytes of message. According to datasheet, the actual value of rainfall is the 3rd and 4th byte that we received and the raw data is greater than actual data by 10. In order to calculate the actual value of rainfall, sum all data after convert byte data into int and use bitwise operation then divide it by 10.



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

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

        #Convert received data into rainfall value
        rain = (float)((int.from_bytes(RxData[3], 'big')) << 8) + (int.from_bytes(RxData[4], 'big'))/10

        #Display Rainfall value
        print("Rainfall value : " + str(rain) + " mm")

        #Clear buffer
        RxData = []

        #Clear index
        index = 0
```

(Pic.5: Rainfall calculation)