Soil PH Sensor with Raspberry Pi Pico
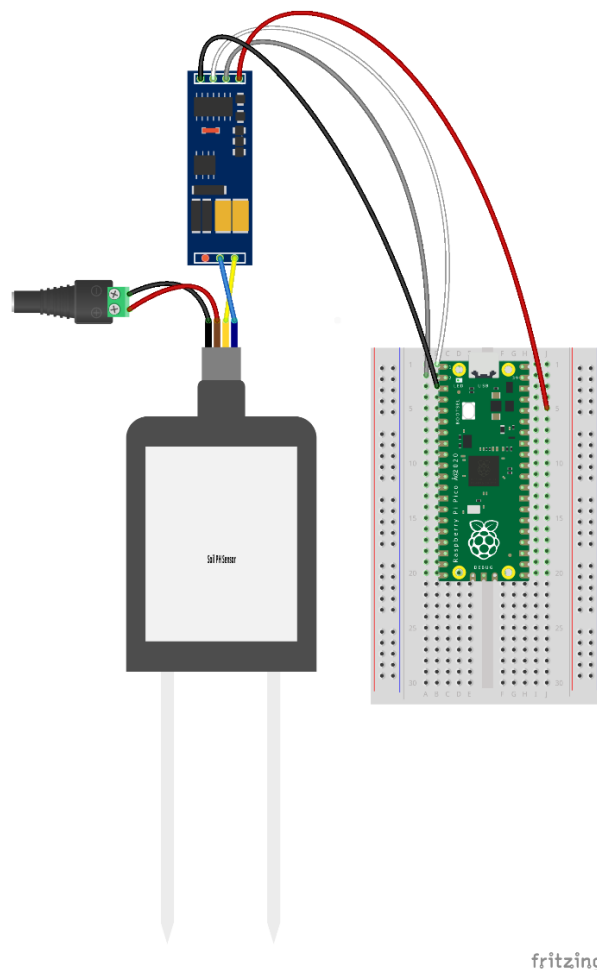
Requirement
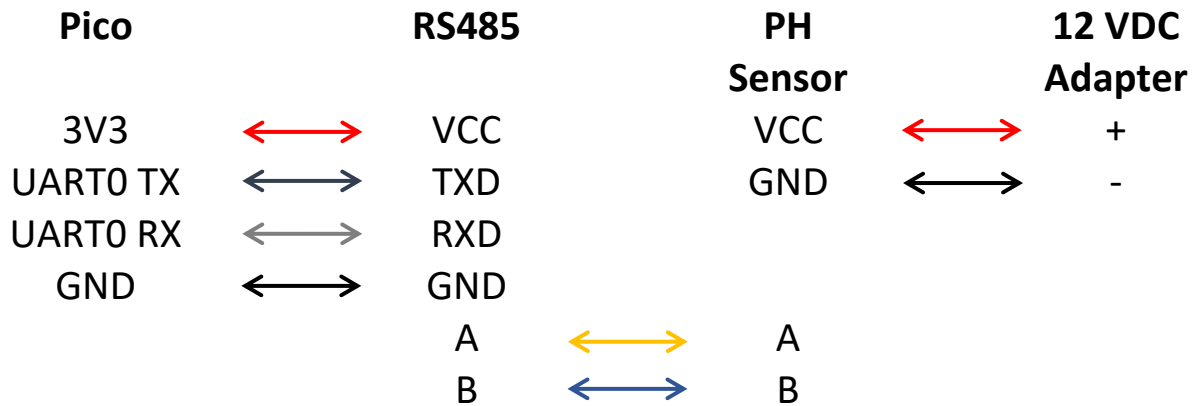
- RS-PH-N01-TR-1
- Raspberry Pi Pico
- RS485 Auto Direction Module
- 12 V DC Adapter
- แจ็กแยกขั้วไฟ ตัวเมีย

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 PH 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. The declaration of pin is already show in the picture below.

```
1  '''
2  Wiring
3      Brown   -> Power(5 - 30 VDC)
4      Black   -> GND
5      Yellow  -> A
6      Blue    -> B
7  '''
8
9  #Import module
10 from machine import UART, Pin
11
12 #Declare UART object
13 uart0 = UART(0, baudrate = 4800, bits=8, parity=None, stop=1) # tx=0, rx=1
14
15 #Check UART object
16 print(uart0)
```

(Pic.1: Module import and UART object declaration)

2. To get data from the sensor via Modbus RTU Protocol, we need to send the command to request data every time. This program will use timer interrupt to send 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).

```python
18  #Define callback function for Timer Interrupt
19  def send(d):
20
21      #Check if callback is enable
22      if tim_ready == 1:
23
24          #Data-read command
25          txData = b'\x01\x03\x00\x00\x00\x01\x84\x0A'
26
27          #Transmission command
28          uart0.write(txData)
29
30          #Check transmitted command
31          print("Sent data : " + str(txData))
32
33          #Disable callback for a while
34          tim_ready == 0
35
36  #Define timer trigger every 1 second and use send() as callback function
37  tim = machine.Timer()
38  tim.init(period = 1000, callback = send)
```

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

3. 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 PH sensor, 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.

```
40  #Declaration of variable objects
41  #Each sensor has maximum data buffer, edit the max_index as your desire
42  RxData = []
43  index = 0
44  max_index = 7
45  tim_ready = 0
```

(Pic.3: Variable object declaration)

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 54) to wait for data. If the data has been received, Program will be running to another loop (in line 58). This loop will add the data buffer into "RxData" and "index" will increase.

```
47  #The main loop start here
48  while True:
49
50      #Enable callback
51      tim_ready = 1
52
53      #Waiting for data to be received
54      while(uart0.any()  < 1):
55          pass
56
57      #When Data received
58      while(uart0.any()  > 0):
59
60          #Add received data to RxData list
61          RxData.append(uart0.read(1))
62
63          #Check if buffer is empty
64          index += 1
```

(Pic.4: Inside the main loop)

5. In the picture below, PH value will be calculated after we receive all bytes of message. According to datasheet, the actual value of PH 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 PH, sum all data after convert byte data into int and use bitwise operation then divide it by 10.

```
66      #When all data has been received
67      if index == max_index:
68
69          #Check received data
70          print("Received data : " + str(RxData))
71
72          #Convert received data into ph value
73          ph = (float)((int.from_bytes(RxData[3], 'big')) << 8) + (int.from_bytes(RxData[4], 'big'))/10
74
75          #Display ph value
76          print("PH : " + str(ph))
77
78          #Clear buffer
79          RxData = []
80
81          #Clear index
82          index = 0
```

(Pic.5: PH calculation)