CPE-X. Project 2566.3

Class 1/5: ESP-32 and Arduino Cloud Platform

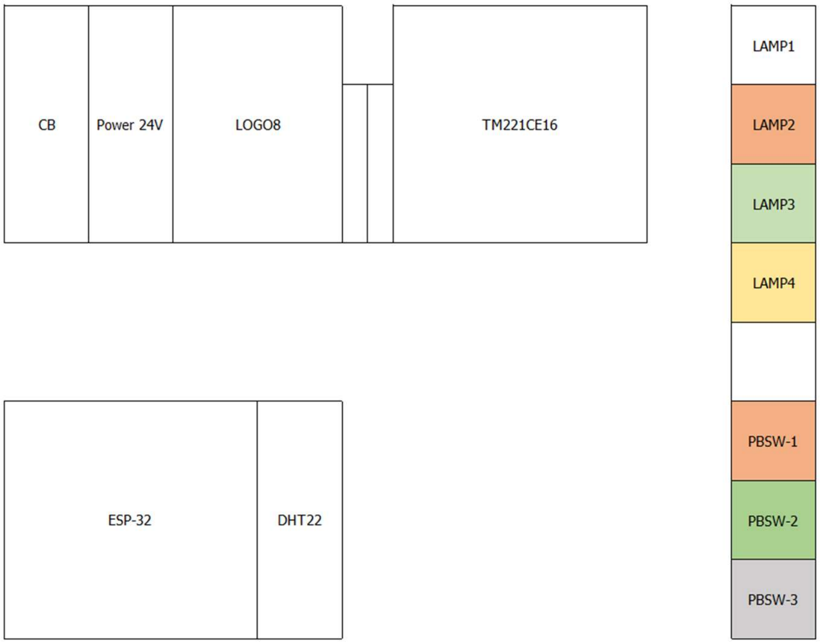
สมาชิก 1: B6309237 นายอนุวัฒน์ ปัสสาพันธ์

สมาชิก 2: B6310646 นางสาวสุภานัน เรืองสุข

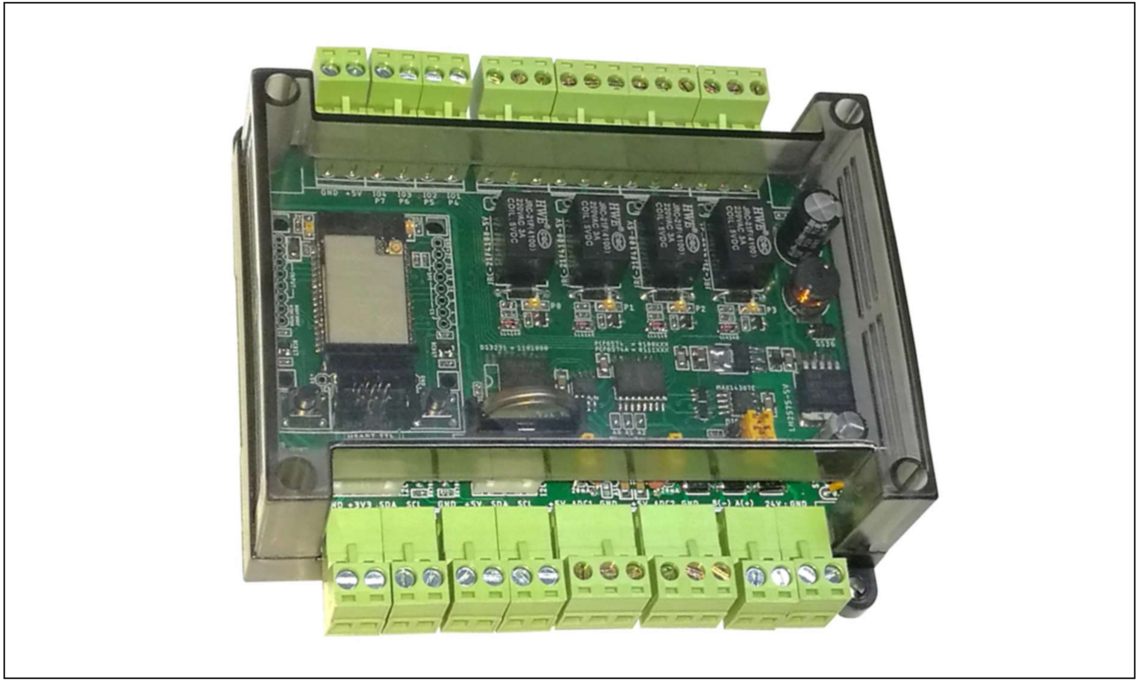
สมาชิก 3: B6321697 นางสาววิจิตรา แซ่เอีย

Week01,02 – ESP32 I/O and IoTs Remote I/O by Arduino Cloud Platform

1. ประกอบอุปกรณ์ต่างๆ บนแผงพลาสติก และเขียนโปรแกรมให้ ESP32 ควบคุมการทำงาน

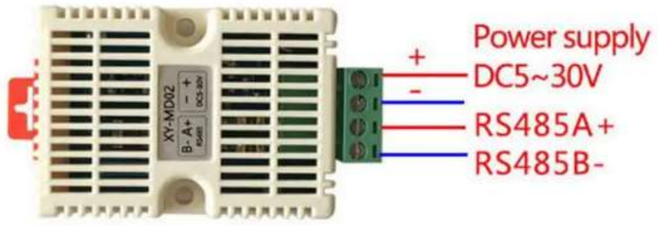


2. แนะนำ ESP32 Board >>
    - https://www.etteam.com/productI2C_RS485/ET-ESP32-RS485_V2/index.html
    - https://www.etteam.com/productI2C_RS485/ET-ESP32-RS485_V2/man-th-et-esp32-wrover-rs485-v2.pdf

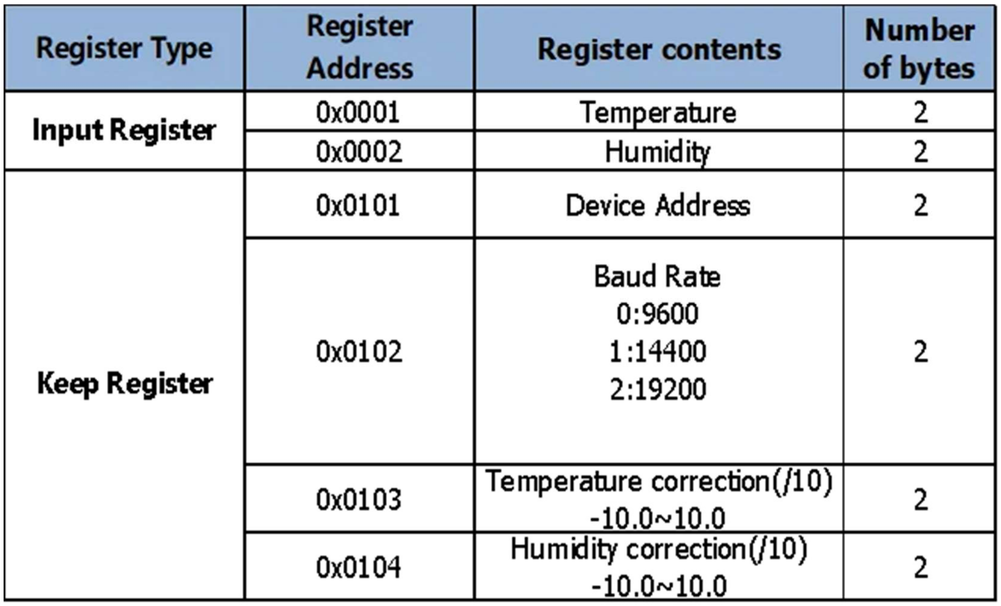


3. ทดสอบการทำงานร่วมกับ Arduino IDE ด้วยโปรแกรม Blink
4. แนะนำ Arduino IoT Platform: https://cloud.arduino.cc/

5. แนะนำ DHT22 : https://www.cybertice.com/p/4530
6. ทดสอบการอ่านค่า DHT22



RS485 communication distance up to 1000 meters.



Register Type	Register Address	Register contents	Number of bytes
Input Register	0x0001	Temperature	2
	0x0002	Humidity	2
Keep Register	0x0101	Device Address	2
	0x0102	Baud Rate 0:9600 1:14400 2:19200	2
	0x0103	Temperature correction(/10) -10.0~10.0	2
	0x0104	Humidity correction(/10) -10.0~10.0	2

7. แนะนำการใช้งาน 24V Pilot Lamp
8. แนะนำการใช้งานสวิตซ์แบบกดติดปล่อยดับ แบบ NO{สีดำ, สีเขียว} และแบบ NC{สีแดง}

9. ทดสอบต่อ Lamp และ สวิตซ์กับ ESP32

10. ต่อวงจรและโปรแกรมระบบให้ทำงานดังนี้

รวมโค้ดทั้งหมดในนี้

https://github.com/miaw88/CPE_X_PJ.git

10.1 อ่านค่าอุณหภูมิและความชื้นแสดงที่ Smart Phone

Code Main Part

```
#define BLYNK_TEMPLATE_ID "TMPL6g3pjkHqp"
#define BLYNK_TEMPLATE_NAME "TempHum"
#define BLYNK_AUTH_TOKEN "74O-g_0xMdV_5Aly2IUbKDoSXIm3AKtn"

#define BLYNK_PRINT Serial

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
char ssid[] = "EM Anu";
char pass[] = "anuwat11";

/******************************************************************************
 * ET-ESP32(WROVER) RS485 V2
 * Tools->Board:"ESP32 Wrover Module"
 *****************************************************************************
 * I2C Interface & I2C Bus
 * -> IO22              = I2C_SCL
 * -> IO21              = I2C_SDA
 * -> I2C RTC:DS3231       = I2C Address : 0x68:1100100(x)
 * -> I2C EEPROM 24LC16    = I2C Address : 0x50:1010000(x)
 * -> I2C ADC MCP3423      = I2C Address : 0x6D:1100101(x)
 * -> I2C Sensor:BME280    = I2C Address : 0x76:1110110(x)
 * -> I2C Sebsor:SHT31     = I2C Address : 0x44:1000100(x)/0x45:1010101(x)
 * SPI Interface SD Card
 * -> SD_CS              = IO4
 * -> SPI_MISO           = IO19S
 * -> SPI_MOSI           = IO23
 * -> SPI_SCK            = IO18
 * UART2 RS485 Half Duplex Auto Direction
 * -> IO26              = RX2
 * -> IO27               = TX2
 * User Switch
 * -> IO36               = USER_SW
 * RTC Interrupt
 * -> IO39               = RTC_INT#
 ******************************************************************************/
#include <Wire.h>
#include <HardwareSerial.h>
#define SerialDebug Serial  // USB Serial(Serial0)
#define SerialRS485_RX_PIN 26
#define SerialRS485_TX_PIN 27
#define SerialRS485 Serial2  // Serial2(IO27=TXD,IO26=RXD)
#define SerialLora_RX_PIN 14
#define SerialLora_TX_PIN 13
#define SerialLora Serial1  // Serial1(IO13=TXD,IO14=RXD)
#define LORA_RES_PIN 33  // ESP32-WROVER :IO33(LoRa-RESET)
#define LORA_RES_PRESS LOW
#define LORA_RES_RELEASE HIGH
#define I2C_SCL_PIN 22  // ESP32-WROVER : IO22(SCL1)
#define I2C_SDA_PIN 21  // ESP32-WROVER : IO21(SDA1)
#define LED_PIN 2  // ESP-WROVER  : IO2
#define LedON 1
#define LedOFF 0
#define USER_SW_PIN 36  // ESP32-WROVER :IO36
#define SW_PRESS LOW
#define SW_RELEASE HIGH
#define RTC_INT_PIN 39  // ESP32-WROVER :IO39
#define RTC_INT_ACTIVE LOW
#define RTC_INT_DEACTIVE HIGH
// End of Default Hardware : ET-ESP32(WROVER) RS485 V2
// Demo RS485 Modbus RTU Interface Soil Moisture Sensor(SOIL MOISTURE-H MODBUS RTU)
// Red    = +5V or 24V(3.6-30VDC)
// Black  = GND
// White  = RS485(B)
// Yellow = RS485(A)
// Green  = NC
```

```
//============================================================================================
// InputRegister[1] = Soil Moisture INT16 Value
// HoldingRegister[512] = Soil Moisture Sensor Slave ID
#include "ModbusMaster.h"  // https://github.com/4-20ma/ModbusMaster
ModbusMaster node;  // instantiate ModbusMaster object
uint8_t result;
float soil_moisture_float_value;

void setup() {
  // Start of Initial Default Hardware : ET-ESP32(WROVER) RS485 V2
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LedOFF);
  pinMode(USER_SW_PIN, INPUT_PULLUP);
  pinMode(RTC_INT_PIN, INPUT_PULLUP);
  Wire.begin(I2C_SDA_PIN, I2C_SCL_PIN);
  SerialDebug.begin(115200);
  while (!SerialDebug)
    ;
  // End of Initial Default Hardware : ET-ESP32(WROVER) RS485 V2
  SerialDebug.println();
  SerialDebug.println("ET-ESP32(WROVER)RS485 V2.....Ready");
  SerialDebug.println();
  SerialDebug.println("ET-ESP32(WROVER)RS485 V2...Demo RS485 Modbus Master Library");
  SerialDebug.println("Interface...Soil Moisture-H Modbus RTU");
  SerialRS485.begin(9600, SERIAL_8N1, SerialRS485_RX_PIN, SerialRS485_TX_PIN);
  while (!SerialRS485)
    ;
  node.begin(1, SerialRS485);  // Soil Moisture = Modbus slave ID 1
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
}
void loop() {
  uint8_t result;
  uint16_t data[2];
  Blynk.run();
  Serial.println("get data");
  result = node.readInputRegisters(1, 2);
  if (result == node.ku8MBSuccess) {
    Serial.print("Temp: ");
    Serial.println(node.getResponseBuffer(0) / 10.0f);
    Serial.print("Humi: ");
    Serial.println(node.getResponseBuffer(1) / 10.0f);
    Serial.println();
    Blynk.virtualWrite(V1, node.getResponseBuffer(0) / 10.0f);
    Blynk.virtualWrite(V2, node.getResponseBuffer(1) / 10.0f);
  }
  delay(1000);
}
```

ModbusMaster.cpp file

```
#include "ModbusMaster.h"
ModbusMaster::ModbusMaster(void)
{
  _idle = 0;
  _preTransmission = 0;
  _postTransmission = 0;
}
void ModbusMaster::begin(uint8_t slave, Stream &serial)
{
//  txBuffer = (uint16_t*) calloc(ku8MaxBufferSize, sizeof(uint16_t));
  _u8MBSlave = slave;
  _serial = &serial;
  _u8TransmitBufferIndex = 0;
  u16TransmitBufferLength = 0;

#if __MODBUSMASTER_DEBUG__
  pinMode(__MODBUSMASTER_DEBUG_PIN_A__, OUTPUT);
  pinMode(__MODBUSMASTER_DEBUG_PIN_B__, OUTPUT);
#endif
}
void ModbusMaster::beginTransmission(uint16_t u16Address)
{
  _u16WriteAddress = u16Address;
  _u8TransmitBufferIndex = 0;
  u16TransmitBufferLength = 0;
}
// eliminate this function in favor of using existing MB request functions
uint8_t ModbusMaster::requestFrom(uint16_t address, uint16_t quantity)
{
  uint8_t read;
  // clamp to buffer length
  if (quantity > ku8MaxBufferSize)
  {
    quantity = ku8MaxBufferSize;
  }
  // set rx buffer iterator vars
  _u8ResponseBufferIndex = 0;
  _u8ResponseBufferLength = read;
  return read;
}
void ModbusMaster::sendBit(bool data)
{
  uint8_t txBitIndex = u16TransmitBufferLength % 16;
  if ((u16TransmitBufferLength >> 4) < ku8MaxBufferSize)
  {
    if (0 == txBitIndex)
    {
      _u16TransmitBuffer[_u8TransmitBufferIndex] = 0;
    }
    bitWrite(_u16TransmitBuffer[_u8TransmitBufferIndex], txBitIndex, data);
    u16TransmitBufferLength++;
    _u8TransmitBufferIndex = u16TransmitBufferLength >> 4;
  }
}
void ModbusMaster::send(uint16_t data)
{
  if (_u8TransmitBufferIndex < ku8MaxBufferSize)
  {
    _u16TransmitBuffer[_u8TransmitBufferIndex++] = data;
    u16TransmitBufferLength = _u8TransmitBufferIndex << 4;
  }
}
```

```
void ModbusMaster::send(uint32_t data)
{
  send(lowWord(data));
  send(highWord(data));
}
void ModbusMaster::send(uint8_t data)
{
  send(word(data));
}
uint8_t ModbusMaster::available(void)
{
  return _u8ResponseBufferLength - _u8ResponseBufferIndex;
}
uint16_t ModbusMaster::receive(void)
{
  if (_u8ResponseBufferIndex < _u8ResponseBufferLength)
  {
    return _u16ResponseBuffer[_u8ResponseBufferIndex++];
  }
  else
  {
    return 0xFFFF;
  }
}
void ModbusMaster::idle(void (*idle)())
{
  _idle = idle;
}
void ModbusMaster::preTransmission(void (*preTransmission)())
{
  _preTransmission = preTransmission;
}
void ModbusMaster::postTransmission(void (*postTransmission)())
{
  _postTransmission = postTransmission;
}
uint16_t ModbusMaster::getResponseBuffer(uint8_t u8Index)
```

```
{
  if (u8Index < ku8MaxBufferSize)
  {
    return _u16ResponseBuffer[u8Index];
  }
  else
  {
    return 0xFFFF;
  }
}
void ModbusMaster::clearResponseBuffer()
{
  uint8_t i;

  for (i = 0; i < ku8MaxBufferSize; i++)
  {
    _u16ResponseBuffer[i] = 0;
  }
}
uint8_t ModbusMaster::setTransmitBuffer(uint8_t u8Index, uint16_t u16Value)
{
  if (u8Index < ku8MaxBufferSize)
  {
    _u16TransmitBuffer[u8Index] = u16Value;
    return ku8MBSuccess;
  }
  else
  {
    return ku8MBIllegalDataAddress;
  }
}
void ModbusMaster::clearTransmitBuffer()
{
  uint8_t i;

  for (i = 0; i < ku8MaxBufferSize; i++)
  {
    _u16TransmitBuffer[i] = 0;
  }
}
uint8_t ModbusMaster::readCoils(uint16_t u16ReadAddress, uint16_t u16BitQty)
{
  _u16ReadAddress = u16ReadAddress;
  _u16ReadQty = u16BitQty;
  return ModbusMasterTransaction(ku8MBReadCoils);
}
uint8_t ModbusMaster::readDiscreteInputs(uint16_t u16ReadAddress,
  uint16_t u16BitQty)
{
  _u16ReadAddress = u16ReadAddress;
  _u16ReadQty = u16BitQty;
  return ModbusMasterTransaction(ku8MBReadDiscreteInputs);
}
uint8_t ModbusMaster::readHoldingRegisters(uint16_t u16ReadAddress,
  uint16_t u16ReadQty)
{
  _u16ReadAddress = u16ReadAddress;
  _u16ReadQty = u16ReadQty;
  return ModbusMasterTransaction(ku8MBReadHoldingRegisters);
}
uint8_t ModbusMaster::readInputRegisters(uint16_t u16ReadAddress,
  uint8_t u16ReadQty)
{
  _u16ReadAddress = u16ReadAddress;
  _u16ReadQty = u16ReadQty;
  return ModbusMasterTransaction(ku8MBReadInputRegisters);
}
uint8_t ModbusMaster::writeSingleCoil(uint16_t u16WriteAddress, uint8_t u8State)
{
  _u16WriteAddress = u16WriteAddress;
  _u16WriteQty = (u8State ? 0xFF00 : 0x0000);
  return ModbusMasterTransaction(ku8MBWriteSingleCoil);
}
uint8_t ModbusMaster::writeSingleRegister(uint16_t u16WriteAddress,
  uint16_t u16WriteValue)
{
  _u16WriteAddress = u16WriteAddress;
  _u16WriteQty = 0;
  _u16TransmitBuffer[0] = u16WriteValue;
  return ModbusMasterTransaction(ku8MBWriteSingleRegister);
}
uint8_t ModbusMaster::writeMultipleCoils(uint16_t u16WriteAddress,
  uint16_t u16BitQty)
{
  _u16WriteAddress = u16WriteAddress;
  _u16WriteQty = u16BitQty;
  return ModbusMasterTransaction(ku8MBWriteMultipleCoils);
}
uint8_t ModbusMaster::writeMultipleCoils()
{
  _u16WriteQty = u16TransmitBufferLength;
  return ModbusMasterTransaction(ku8MBWriteMultipleCoils);
}
uint8_t ModbusMaster::writeMultipleRegisters(uint16_t u16WriteAddress,
  uint16_t u16WriteQty)
{
  _u16WriteAddress = u16WriteAddress;
  _u16WriteQty = u16WriteQty;
  return ModbusMasterTransaction(ku8MBWriteMultipleRegisters);
}

// new version based on Wire.h
uint8_t ModbusMaster::writeMultipleRegisters()
{
  _u16WriteQty = _u8TransmitBufferIndex;
  return ModbusMasterTransaction(ku8MBWriteMultipleRegisters);
}
uint8_t ModbusMaster::maskWriteRegister(uint16_t u16WriteAddress,
  uint16_t u16AndMask, uint16_t u16OrMask)
{
  _u16WriteAddress = u16WriteAddress;
  _u16TransmitBuffer[0] = u16AndMask;
  _u16TransmitBuffer[1] = u16OrMask;
  return ModbusMasterTransaction(ku8MBMaskWriteRegister);
}
uint8_t ModbusMaster::readWriteMultipleRegisters(uint16_t u16ReadAddress,
  uint16_t u16ReadQty, uint16_t u16WriteAddress, uint16_t u16WriteQty)
{
  _u16ReadAddress = u16ReadAddress;
  _u16ReadQty = u16ReadQty;
  _u16WriteAddress = u16WriteAddress;
  _u16WriteQty = u16WriteQty;
  return ModbusMasterTransaction(ku8MBReadWriteMultipleRegisters);
}
uint8_t ModbusMaster::readWriteMultipleRegisters(uint16_t u16ReadAddress,
  uint16_t u16ReadQty)
{
  _u16ReadAddress = u16ReadAddress;
  _u16ReadQty = u16ReadQty;
  _u16WriteQty = _u8TransmitBufferIndex;
  return ModbusMasterTransaction(ku8MBReadWriteMultipleRegisters);
}
uint8_t ModbusMaster::ModbusMasterTransaction(uint8_t u8MBFunction)
{
  uint8_t u8ModbusADU[256];
  uint8_t u8ModbusADUSize = 0;
  uint8_t i, u8Qty;
  uint16_t u16CRC;
  uint32_t u32StartTime;
  uint8_t u8BytesLeft = 8;
  uint8_t u8MBStatus = ku8MBSuccess;
  // assemble Modbus Request Application Data Unit
  u8ModbusADU[u8ModbusADUSize++] = _u8MBSlave;
  u8ModbusADU[u8ModbusADUSize++] = u8MBFunction;
  switch(u8MBFunction)
  {
    case ku8MBReadCoils:
    case ku8MBReadDiscreteInputs:
```

```
  case ku8MBReadInputRegisters:
  case ku8MBReadHoldingRegisters:
  case ku8MBReadWriteMultipleRegisters:
    u8ModbusADU[u8ModbusADUSize++] = highByte(_u16ReadAddress);
    u8ModbusADU[u8ModbusADUSize++] = lowByte(_u16ReadAddress);
    u8ModbusADU[u8ModbusADUSize++] = highByte(_u16ReadQty);
    u8ModbusADU[u8ModbusADUSize++] = lowByte(_u16ReadQty);
    break;
}
switch(u8MBFunction)
{
  case ku8MBWriteSingleCoil:
  case ku8MBMaskWriteRegister:
  case ku8MBWriteMultipleCoils:
  case ku8MBWriteSingleRegister:
  case ku8MBWriteMultipleRegisters:
  case ku8MBReadWriteMultipleRegisters:
    u8ModbusADU[u8ModbusADUSize++] = highByte(_u16WriteAddress);
    u8ModbusADU[u8ModbusADUSize++] = lowByte(_u16WriteAddress);
    break;
}
switch(u8MBFunction)
{
  case ku8MBWriteSingleCoil:
    u8ModbusADU[u8ModbusADUSize++] = highByte(_u16WriteQty);
    u8ModbusADU[u8ModbusADUSize++] = lowByte(_u16WriteQty);
    break;
  case ku8MBWriteSingleRegister:
    u8ModbusADU[u8ModbusADUSize++] = highByte(_u16TransmitBuffer[0]);
    u8ModbusADU[u8ModbusADUSize++] = lowByte(_u16TransmitBuffer[0]);
    break;
  case ku8MBWriteMultipleCoils:
    u8ModbusADU[u8ModbusADUSize++] = highByte(_u16WriteQty);
    u8ModbusADU[u8ModbusADUSize++] = lowByte(_u16WriteQty);
    u8Qty = (_u16WriteQty % 8) ? ((_u16WriteQty >> 3) + 1) : (_u16WriteQty >> 3);
    u8ModbusADU[u8ModbusADUSize++] = u8Qty;
    for (i = 0; i < u8Qty; i++)
    {
      switch(i % 2)
      {
        case 0: // i is even
          u8ModbusADU[u8ModbusADUSize++] = lowByte(_u16TransmitBuffer[i >> 1]);
          break;

        case 1: // i is odd
          u8ModbusADU[u8ModbusADUSize++] = highByte(_u16TransmitBuffer[i >> 1]);
          break;
      }
    }
    break;
  case ku8MBWriteMultipleRegisters:
  case ku8MBReadWriteMultipleRegisters:
    u8ModbusADU[u8ModbusADUSize++] = highByte(_u16WriteQty);
    u8ModbusADU[u8ModbusADUSize++] = lowByte(_u16WriteQty);
    u8ModbusADU[u8ModbusADUSize++] = lowByte(_u16WriteQty << 1);
    for (i = 0; i < lowByte(_u16WriteQty); i++)
    {
      u8ModbusADU[u8ModbusADUSize++] = highByte(_u16TransmitBuffer[i]);
      u8ModbusADU[u8ModbusADUSize++] = lowByte(_u16TransmitBuffer[i]);
    }
    break;
  case ku8MBMaskWriteRegister:
    u8ModbusADU[u8ModbusADUSize++] = highByte(_u16TransmitBuffer[0]);
    u8ModbusADU[u8ModbusADUSize++] = lowByte(_u16TransmitBuffer[0]);
    u8ModbusADU[u8ModbusADUSize++] = highByte(_u16TransmitBuffer[1]);
    u8ModbusADU[u8ModbusADUSize++] = lowByte(_u16TransmitBuffer[1]);
    break;
}
// append CRC
u16CRC = 0xFFFF;
for (i = 0; i < u8ModbusADUSize; i++)
{
  u16CRC = crc16_update(u16CRC, u8ModbusADU[i]);
}
```

```
  u8ModbusADU[u8ModbusADUSize++] = lowByte(u16CRC);
  u8ModbusADU[u8ModbusADUSize++] = highByte(u16CRC);
  u8ModbusADU[u8ModbusADUSize] = 0;
  // flush receive buffer before transmitting request
  while (_serial->read() != -1);
  // transmit request
  if (_preTransmission)
  {
    _preTransmission();
  }
  for (i = 0; i < u8ModbusADUSize; i++)
  {
    _serial->write(u8ModbusADU[i]);
  }
  u8ModbusADUSize = 0;
  _serial->flush();    // flush transmit buffer
  if (_postTransmission)
  {
    _postTransmission();
  }
  // loop until we run out of time or bytes, or an error occurs
  u32StartTime = millis();
  while (u8BytesLeft && !u8MBStatus)
  {
    if (_serial->available())
    {
#if __MODBUSMASTER_DEBUG__
      digitalWrite(__MODBUSMASTER_DEBUG_PIN_A__, true);
#endif
      u8ModbusADU[u8ModbusADUSize++] = _serial->read();
      u8BytesLeft--;
#if __MODBUSMASTER_DEBUG__
      digitalWrite(__MODBUSMASTER_DEBUG_PIN_A__, false);
#endif
    }
    else
    {
```

```
#if __MODBUSMASTER_DEBUG__
      digitalWrite(__MODBUSMASTER_DEBUG_PIN_B__, true);
#endif
      if (_idle)
      {
        _idle();
      }
#if __MODBUSMASTER_DEBUG__
      digitalWrite(__MODBUSMASTER_DEBUG_PIN_B__, false);
#endif
    }
    // evaluate slave ID, function code once enough bytes have been read
    if (u8ModbusADUSize == 5)
    {
      // verify response is for correct Modbus slave
      if (u8ModbusADU[0] != _u8MBSlave)
      {
        u8MBStatus = ku8MBInvalidSlaveID;
        break;
      }
      // verify response is for correct Modbus function code (mask exception bit 7)
      if ((u8ModbusADU[1] & 0x7F) != u8MBFunction)
      {
        u8MBStatus = ku8MBInvalidFunction;
        break;
      }
      // check whether Modbus exception occurred; return Modbus Exception Code
      if (bitRead(u8ModbusADU[1], 7))
      {
        u8MBStatus = u8ModbusADU[2];
        break;
      }
      // evaluate returned Modbus function code
      switch(u8ModbusADU[1])
      {
        case ku8MBReadCoils:
        case ku8MBReadDiscreteInputs:
```

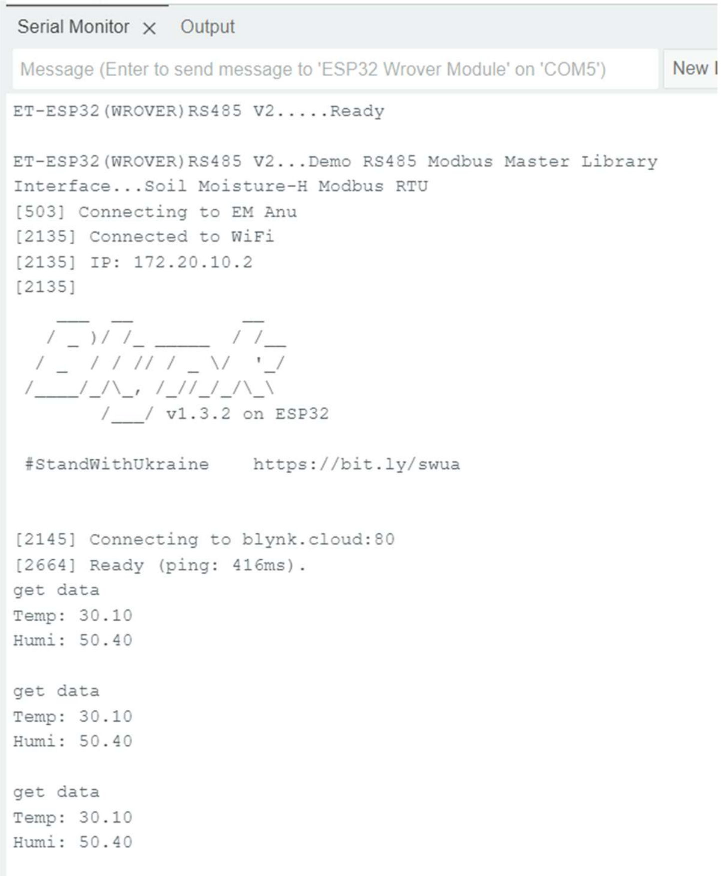
```
      case ku8MBReadInputRegisters:
      case ku8MBReadHoldingRegisters:
      case ku8MBReadWriteMultipleRegisters:
        u8BytesLeft = u8ModbusADU[2];
        break;
      case ku8MBWriteSingleCoil:
      case ku8MBWriteMultipleCoils:
      case ku8MBWriteSingleRegister:
      case ku8MBWriteMultipleRegisters:
        u8BytesLeft = 3;
        break;
      case ku8MBMaskWriteRegister:
        u8BytesLeft = 5;
        break;
    }
  }
  if ((millis() - u32StartTime) > ku16MBResponseTimeout)
  {
    u8MBStatus = ku8MBResponseTimedOut;
  }
}
// verify response is large enough to inspect further
if (!u8MBStatus && u8ModbusADUSize >= 5)
{
  // calculate CRC
  u16CRC = 0xFFFF;
  for (i = 0; i < (u8ModbusADUSize - 2); i++)
  {
    u16CRC = crc16_update(u16CRC, u8ModbusADU[i]);
  }
  // verify CRC
  if (!u8MBStatus && (lowByte(u16CRC) != u8ModbusADU[u8ModbusADUSize - 2] ||
    highByte(u16CRC) != u8ModbusADU[u8ModbusADUSize - 1]))
  {
    u8MBStatus = ku8MBInvalidCRC;
  }
}
```

```
// disassemble ADU into words
if (!u8MBStatus)
{
  // evaluate returned Modbus function code
  switch(u8ModbusADU[1])
  {
    case ku8MBReadCoils:
    case ku8MBReadDiscreteInputs:
      // load bytes into word; response bytes are ordered L, H, L, H, ...
      for (i = 0; i < (u8ModbusADU[2] >> 1); i++)
      {
        if (i < ku8MaxBufferSize)
        {
          _u16ResponseBuffer[i] = word(u8ModbusADU[2 * i + 4], u8ModbusADU[2 * i + 3]);
        }
        _u8ResponseBufferLength = i;
      }
      // in the event of an odd number of bytes, load last byte into zero-padded word
      if (u8ModbusADU[2] % 2)
      {
        if (i < ku8MaxBufferSize)
        {
          _u16ResponseBuffer[i] = word(0, u8ModbusADU[2 * i + 3]);
        }
        _u8ResponseBufferLength = i + 1;
      }
      break;
    case ku8MBReadInputRegisters:
    case ku8MBReadHoldingRegisters:
    case ku8MBReadWriteMultipleRegisters:
      // load bytes into word; response bytes are ordered H, L, H, L, ...
      for (i = 0; i < (u8ModbusADU[2] >> 1); i++)
      {
        if (i < ku8MaxBufferSize)          {
          _u16ResponseBuffer[i] = word(u8ModbusADU[2 * i + 3], u8ModbusADU[2 * i + 4]);
        }
        _u8ResponseBufferLength = i;        }
        break;    }0  }
  _u8TransmitBufferIndex = 0;
  u16TransmitBufferLength = 0;
  _u8ResponseBufferIndex = 0;
  return u8MBStatus;
}
```

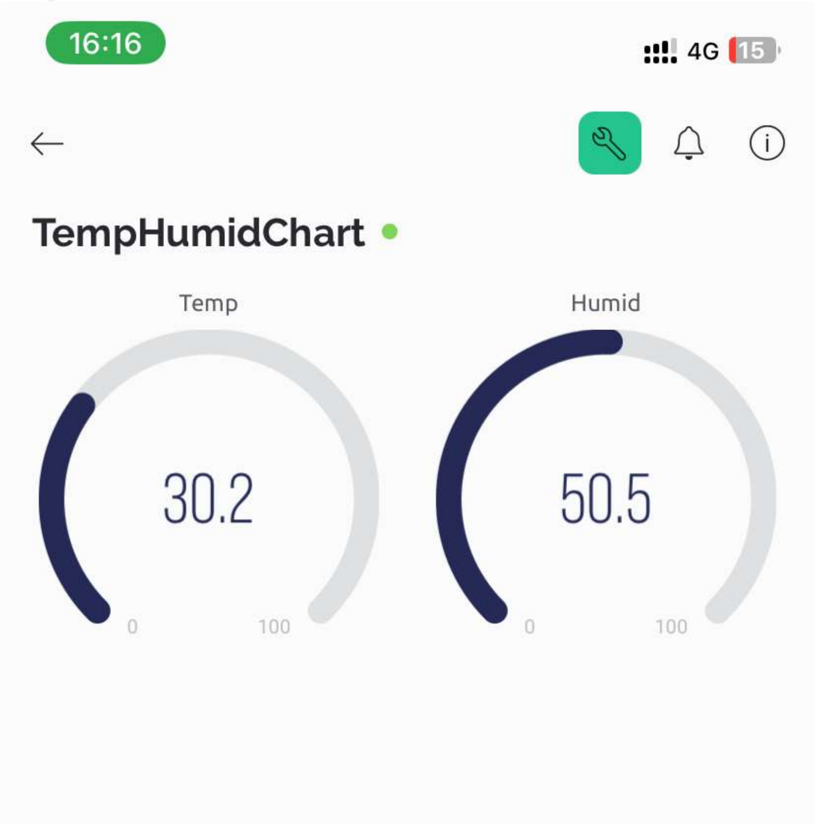
ModbusMaster.h   file

```
#ifndef ModbusMaster_h
#define ModbusMaster_h
#define __MODBUSMASTER_DEBUG__ (0)
#define __MODBUSMASTER_DEBUG_PIN_A__ 4
#define __MODBUSMASTER_DEBUG_PIN_B__ 5
#include "Arduino.h"
#include "util/crc16.h"
#include "util/word.h"
class ModbusMaster
{
  public:
    ModbusMaster();
    void begin(uint8_t, Stream &serial);
    void idle(void (*)());
    void preTransmission(void (*)());
    void postTransmission(void (*)());
    // Modbus exception codes
    static const uint8_t ku8MBIllegalFunction            = 0x01;
    static const uint8_t ku8MBIllegalDataAddress         = 0x02;
    static const uint8_t ku8MBIllegalDataValue           = 0x03;
    static const uint8_t ku8MBSlaveDeviceFailure         = 0x04;
    static const uint8_t ku8MBSuccess                    = 0x00;
    static const uint8_t ku8MBInvalidSlaveID             = 0xE0;
    static const uint8_t ku8MBInvalidFunction            = 0xE1;
    static const uint8_t ku8MBResponseTimedOut           = 0xE2;
    static const uint8_t ku8MBInvalidCRC                 = 0xE3;
    uint16_t getResponseBuffer(uint8_t);
    void     clearResponseBuffer();
    uint8_t  setTransmitBuffer(uint8_t, uint16_t);
    void     clearTransmitBuffer();
    void beginTransmission(uint16_t);
    uint8_t requestFrom(uint16_t, uint16_t);
    void sendBit(bool);
    void send(uint8_t);
    void send(uint16_t);
    void send(uint32_t);
    uint8_t available(void);
    uint16_t receive(void);
    uint8_t  readCoils(uint16_t, uint16_t);
    uint8_t  readDiscreteInputs(uint16_t, uint16_t);
    uint8_t  readHoldingRegisters(uint16_t, uint16_t);
    uint8_t  readInputRegisters(uint16_t, uint8_t);
    uint8_t  writeSingleCoil(uint16_t, uint8_t);
    uint8_t  writeSingleRegister(uint16_t, uint16_t);
    uint8_t  writeMultipleCoils(uint16_t, uint16_t);
    uint8_t  writeMultipleCoils();
    uint8_t  writeMultipleRegisters(uint16_t, uint16_t);
    uint8_t  writeMultipleRegisters();
    uint8_t  maskWriteRegister(uint16_t, uint16_t, uint16_t);
    uint8_t  readWriteMultipleRegisters(uint16_t, uint16_t, uint16_t, uint16_t);
    uint8_t  readWriteMultipleRegisters(uint16_t, uint16_t);
  private:
    Stream* _serial;                                         ///< reference to serial port object
    uint8_t  _u8MBSlave;                                       ///< Modbus slave (1..255) initialized in begin()
    static const uint8_t ku8MaxBufferSize              = 64;   ///< size of response/transmit buffers
    uint16_t _u16ReadAddress;                                    ///< slave register from which to read
    uint16_t _u16ReadQty;                                       ///< quantity of words to read
    uint16_t _u16ResponseBuffer[ku8MaxBufferSize];              ///< buffer to store Modbus slave
response; read via GetResponseBuffer()
    uint16_t _u16WriteAddress;                                   ///< slave register to which to write
    uint16_t _u16WriteQty;                                      ///< quantity of words to write
    uint16_t _u16TransmitBuffer[ku8MaxBufferSize];              ///< buffer containing data to transmit to
Modbus slave; set via SetTransmitBuffer()
    uint16_t* txBuffer; // from Wire.h -- need to clean this up Rx
    uint8_t _u8TransmitBufferIndex;
    uint16_t u16TransmitBufferLength;
    uint16_t* rxBuffer; // from Wire.h -- need to clean this up Rx
    uint8_t _u8ResponseBufferIndex;
    uint8_t _u8ResponseBufferLength;
    // Modbus function codes for bit access
    static const uint8_t ku8MBReadCoils                  = 0x01; ///< Modbus function 0x01 Read Coils
    static const uint8_t ku8MBReadDiscreteInputs         = 0x02; ///< Modbus function 0x02 Read Discrete
Inputs
    static const uint8_t ku8MBWriteSingleCoil            = 0x05; ///< Modbus function 0x05 Write Single
Coil
    static const uint8_t ku8MBWriteMultipleCoils         = 0x0F; ///< Modbus function 0x0F Write Multiple
Coils
    // Modbus function codes for 16 bit access
    static const uint8_t ku8MBReadHoldingRegisters       = 0x03; ///< Modbus function 0x03 Read
Holding Registers
    static const uint8_t ku8MBReadInputRegisters         = 0x04; ///< Modbus function 0x04 Read Input
Registers
    static const uint8_t ku8MBWriteSingleRegister        = 0x06; ///< Modbus function 0x06 Write Single
Register
    static const uint8_t ku8MBWriteMultipleRegisters     = 0x10; ///< Modbus function 0x10 Write
Multiple Registers
    static const uint8_t ku8MBMaskWriteRegister          = 0x16; ///< Modbus function 0x16 Mask Write
Register
    static const uint8_t ku8MBReadWriteMultipleRegisters = 0x17; ///< Modbus function 0x17 Read Write
Multiple Registers
    // Modbus timeout [milliseconds]
    static const uint16_t ku16MBResponseTimeout          = 2000; ///< Modbus timeout [milliseconds]
    // master function that conducts Modbus transactions
    uint8_t ModbusMasterTransaction(uint8_t u8MBFunction);
    // idle callback function; gets called during idle time between TX and RX
    void (*_idle)();
    // preTransmission callback function; gets called before writing a Modbus message
    void (*_preTransmission)();
    // postTransmission callback function; gets called after a Modbus message has been sent
    void (*_postTransmission)();
};
#endif
```

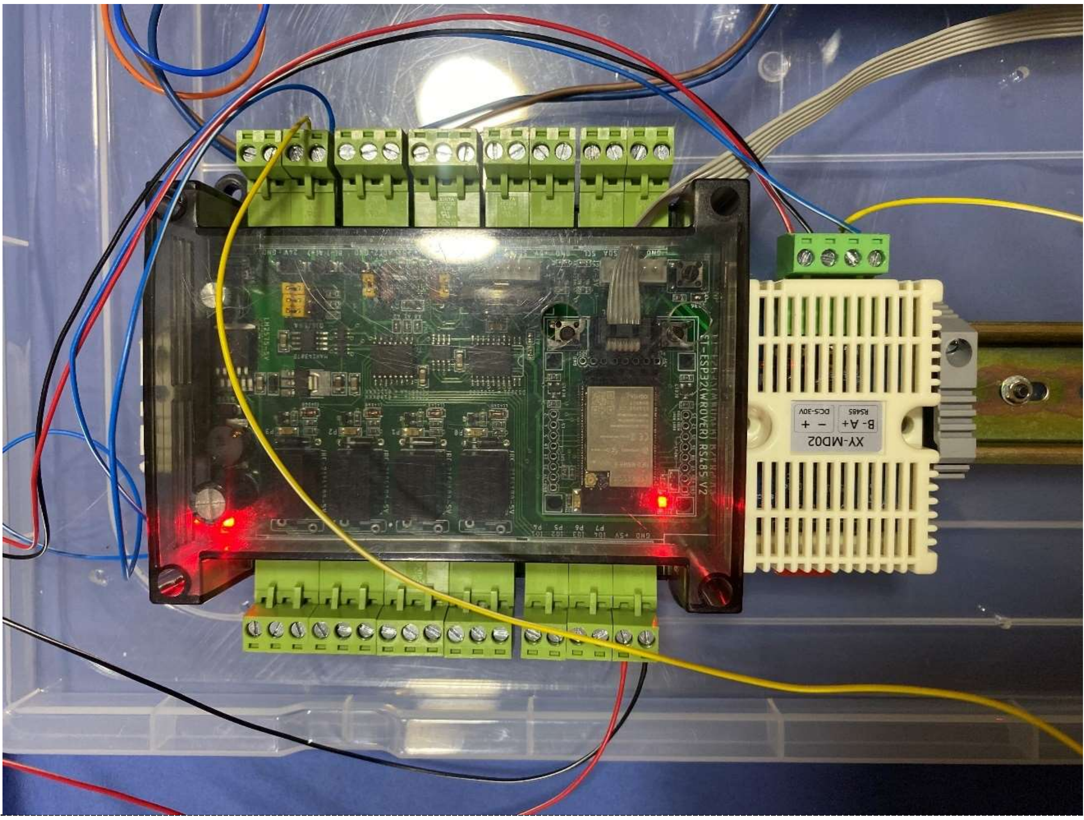
โปรแกรมที่ใช้ทดสอบ


Serial Monitor ×
Output
Message (Enter to send message to 'ESP32 Wrover Module' on 'COM5')
New l

```
ET-ESP32(WROVER)RS485 V2.....Ready

ET-ESP32(WROVER)RS485 V2...Demo RS485 Modbus Master Library
Interface...Soil Moisture-H Modbus RTU
[503] Connecting to EM Anu
[2135] Connected to WiFi
[2135] IP: 172.20.10.2
[2135]
    ___  __          __
   / _ )/ /_ _____  / /__
  / _  / / // / _ \/  '_/
 /____/_/\_, /_//_/_/\_\
        /___/ v1.3.2 on ESP32

 #StandWithUkraine    https://bit.ly/swua


[2145] Connecting to blynk.cloud:80
[2664] Ready (ping: 416ms).
get data
Temp: 30.10
Humi: 50.40

get data
Temp: 30.10
Humi: 50.40

get data
Temp: 30.10
Humi: 50.40
```



วงจรที่ใช้ในการทดสอบ



Link

https://youtu.be/zTyW5YI6BEU?si=g3w_SVQ4wnxRU_Nz

10.2 ควบการปิดเปิด White Lamp ผ่าน Smart Phone

Code Main Part

```
xxccc#define BLYNK_TEMPLATE_ID "TMPL6g3pjkHqp"
#define BLYNK_TEMPLATE_NAME "TempHum"
#define BLYNK_AUTH_TOKEN "74O-g_0xMdV_5Aly2IUbKDoSXIm3AKtn"
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include "Arduino.h"
#include "PCF8574.h"  // https://github.com/xreef/PCF8574_library
#define I2C_Address 0x20
#define I2C_SDA_Pin 21
#define I2C_SCL_Pin 22
// Instantiate Wire for generic use at 100kHz
TwoWire I2Ctwo = TwoWire(1);
// Set i2c address
PCF8574 pcf8574(&I2Ctwo, I2C_Address, I2C_SDA_Pin, I2C_SCL_Pin);
// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "EM Anu";
char pass[] = "anuwat11";
/********************************************************************************
 * ET-ESP32(WROVER) RS485 V2
 * Tools->Board:"ESP32 Wrover Module"
 ********************************************************************************
 * I2C Interface & I2C Bus
 * -> IO22               = I2C_SCL
 * -> IO21               = I2C_SDA
 * -> I2C RTC:DS3231       = I2C Address : 0x68:1100100(x)
 * -> I2C EEPROM 24LC16    = I2C Address : 0x50:1010000(x)
 * -> I2C ADC MCP3423      = I2C Address : 0x6D:1100101(x)
 * -> I2C Sensor:BME280    = I2C Address : 0x76:1110110(x)
 * -> I2C Sebsor:SHT31     = I2C Address : 0x44:1000100(x)/0x45:1010101(x)
 * SPI Interface SD Card
 * -> SD_CS               = IO4
 * -> SPI_MISO             = IO19S
 * -> SPI_MOSI            = IO23
 * -> SPI_SCK             = IO18
 * UART2 RS485 Half Duplex Auto Direction
 * -> IO26                = RX2
 * -> IO27                = TX2
 * User Switch
 * -> IO36                = USER_SW
 * RTC Interrupt
 * -> IO39                = RTC_INT#
 ******************************************************************************/
#include <Wire.h>
#include <HardwareSerial.h>
#define SerialDebug Serial  // USB Serial(Serial0)
#define SerialRS485_RX_PIN 26
#define SerialRS485_TX_PIN 27
#define SerialRS485 Serial2  // Serial2(IO27=TXD,IO26=RXD)
#define SerialLora_RX_PIN 14
#define SerialLora_TX_PIN 13
#define SerialLora Serial1  // Serial1(IO13=TXD,IO14=RXD)
#define LORA_RES_PIN 33  // ESP32-WROVER :IO33(LoRa-RESET)
#define LORA_RES_PRESS LOW
#define LORA_RES_RELEASE HIGH
#define I2C_SCL_PIN 22  // ESP32-WROVER : IO22(SCL1)
#define I2C_SDA_PIN 21  // ESP32-WROVER : IO21(SDA1)
#define LED_PIN 2  // ESP-WROVER  : IO2
#define LedON 1
#define LedOFF 0
#define USER_SW_PIN 36  // ESP32-WROVER :IO36
#define SW_PRESS LOW
#define SW_RELEASE HIGH
#define RTC_INT_PIN 39  // ESP32-WROVER :IO39
#define RTC_INT_ACTIVE LOW
#define RTC_INT_DEACTIVE HIGH
// Demo RS485 Modbus RTU Interface Soil Moisture Sensor(SOIL MOISTURE-H MODBUS RTU)
// Red    = +5V or 24V(3.6-30VDC)
// Black  = GND
// White  = RS485(B)
// Yellow = RS485(A)
// Green  = NC
// InputRegister[1] = Soil Moisture INT16 Value
// HoldingRegister[512] = Soil Moisture Sensor Slave ID
#include "ModbusMaster.h"  // https://github.com/4-20ma/ModbusMaster
ModbusMaster node;  // instantiate ModbusMaster object
uint8_t result;
float soil_moisture_float_value;
void setup() {
  // Start of Initial Default Hardware : ET-ESP32(WROVER) RS485 V2
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LedOFF);
  pinMode(USER_SW_PIN, INPUT_PULLUP);
  pinMode(RTC_INT_PIN, INPUT_PULLUP);
  Wire.begin(I2C_SDA_PIN, I2C_SCL_PIN);
  SerialDebug.begin(115200);
  while (!SerialDebug);
  // End of Initial Default Hardware : ET-ESP32(WROVER) RS485 V2
  SerialDebug.println();
  SerialDebug.println("ET-ESP32(WROVER)RS485 V2.....Ready");
  SerialDebug.println();
  SerialDebug.println("ET-ESP32(WROVER)RS485 V2...Demo RS485 Modbus Master Library");
  SerialDebug.println("Interface...Soil Moisture-H Modbus RTU");
  SerialRS485.begin(9600, SERIAL_8N1, SerialRS485_RX_PIN, SerialRS485_TX_PIN);
  while (!SerialRS485);
  node.begin(1, SerialRS485);  // Soil Moisture = Modbus slave ID 1
  pcf8574.pinMode(0, OUTPUT);
  pcf8574.pinMode(1, OUTPUT);
  pcf8574.pinMode(2, OUTPUT);
  pcf8574.pinMode(3, OUTPUT);
  pcf8574.pinMode(4, INPUT_PULLUP);
  pcf8574.pinMode(5, INPUT_PULLUP);
  pcf8574.pinMode(6, INPUT_PULLUP);
  pcf8574.pinMode(7, INPUT_PULLUP);
  pcf8574.begin();
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);}
int Counter = 0;
```

```
void loop() {
  uint8_t result;
  uint16_t data[2];
  Blynk.run();
  Serial.println("get data");
  result = node.readInputRegisters(1, 2);
  if (result == node.ku8MBSuccess) {
    Serial.print("Temp: ");
    Serial.println(node.getResponseBuffer(0) / 10.0f);
    Serial.print("Humi: ");
    Serial.println(node.getResponseBuffer(1) / 10.0f);
    Serial.println();
    Blynk.virtualWrite(V1, node.getResponseBuffer(0) / 10.0f);
    Blynk.virtualWrite(V2, node.getResponseBuffer(1) / 10.0f);
  }
  if (pcf8574.digitalRead(P4) == LOW) {
    delay(20);
    while (pcf8574.digitalRead(P4) == LOW)
      delay(50);
    Counter++;
    delay(10);
    Serial.println(Counter);
    pcf8574.digitalWrite(P0, Counter % 2);
  }
  delay(500);
}
BLYNK_WRITE(V3)
{
  int pinValue = param.asInt();  // assigning incoming value from pin V1 to a variable
  pcf8574.digitalWrite(P0, !pinValue);
  if (pinValue == 1) {
    // do something when button is pressed;
    Serial.println("White Lamp is ON");
  } else if (pinValue == 0) {
    Serial.println("White Lamp is OFF");
  }
}
```

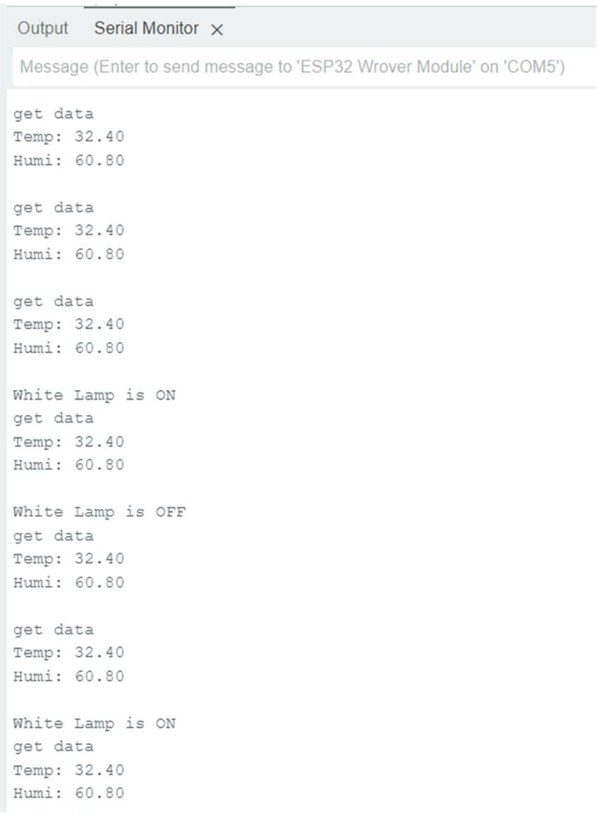
ModbusMaster.cpp file

#Same code at 10.1 or https://github.com/miaw88/CPE_X_PJ/blob/main/meet-1of5/10.2/10.2/ModbusMaster.cpp

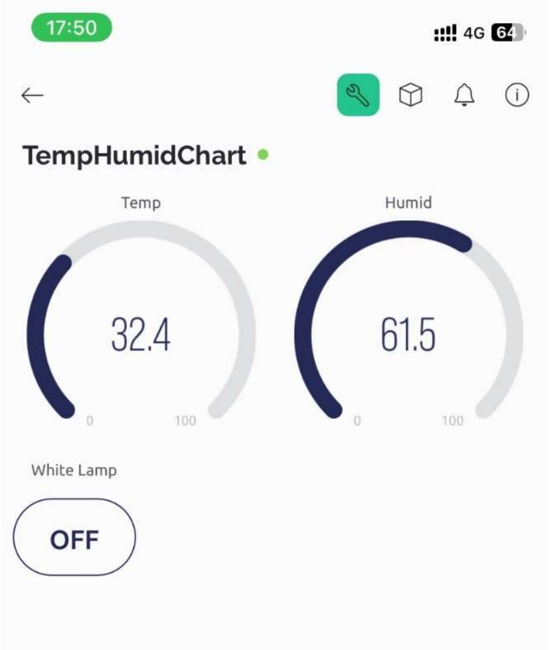
ModbusMaster.h file

#Same code at 10.1 or https://github.com/miaw88/CPE_X_PJ/blob/main/meet-1of5/10.2/10.2/ModbusMaster.h

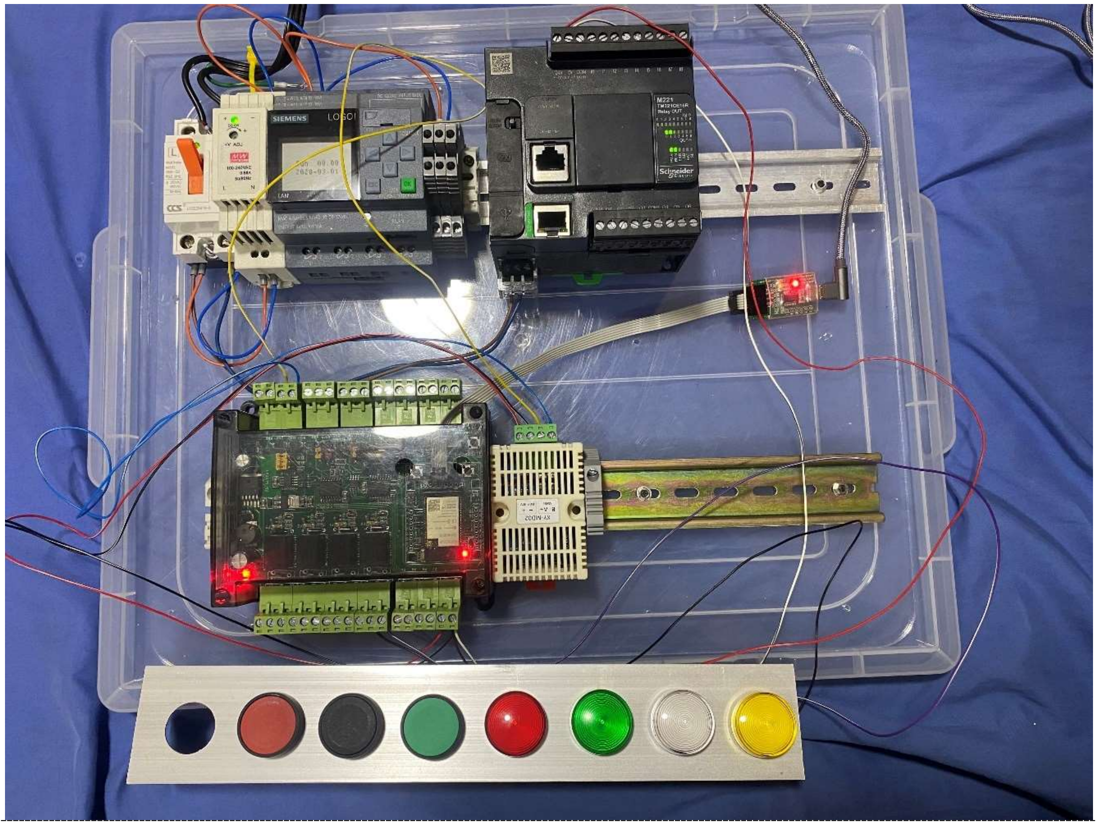
โปรแกรมที่ใช้ทดสอบ


Output
Serial Monitor ×
Message (Enter to send message to 'ESP32 Wrover Module' on 'COM5')

```
get data
Temp: 32.40
Humi: 60.80

get data
Temp: 32.40
Humi: 60.80

get data
Temp: 32.40
Humi: 60.80

White Lamp is ON
get data
Temp: 32.40
Humi: 60.80

White Lamp is OFF
get data
Temp: 32.40
Humi: 60.80

get data
Temp: 32.40
Humi: 60.80

White Lamp is ON
get data
Temp: 32.40
Humi: 60.80
```



วงจรที่ใช้ในการทดสอบ



Link

https://youtu.be/nrwx0CIhYlo?si=1n7Dhr_pUMiVF7DE

10.3 ควบการปิดเปิด Green Lamp ผ่าน Smartt Phone และ Green Switch โดย

- ปิดด้วย Smartt Phone
- เปิดด้วย Smartt Phone
- หาก Lamp On กดสวิตซ์จะ Off
- หาก Lamp Off กดสวิตซ์จะ On

Code Main Part

```
#define BLYNK_TEMPLATE_ID "TMPL6g3pjkHqp"
#define BLYNK_TEMPLATE_NAME "TempHum"
#define BLYNK_AUTH_TOKEN "74O-g_0xMdV_5Aly2IUbKDoSXIm3AKtn"
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include "Arduino.h"
#include "PCF8574.h"  // https://github.com/xreef/PCF8574_library
#define I2C_Address 0x20
#define I2C_SDA_Pin 21
#define I2C_SCL_Pin 22
// Instantiate Wire for generic use at 100kHz
TwoWire I2Ctwo = TwoWire(1);
// Set i2c address
PCF8574 pcf8574(&I2Ctwo, I2C_Address, I2C_SDA_Pin, I2C_SCL_Pin);
// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "EM Anu";
char pass[] = "anuwat11";
/******************************************************************************
 * ET-ESP32(WROVER) RS485 V2
 * Tools->Board:"ESP32 Wrover Module"
 ******************************************************************************
 * I2C Interface & I2C Bus
 * -> IO22                = I2C_SCL
 * -> IO21                = I2C_SDA
 * -> I2C RTC:DS3231       = I2C Address : 0x68:1100100(x)
 * -> I2C EEPROM 24LC16    = I2C Address : 0x50:1010000(x)
 * -> I2C ADC MCP3423      = I2C Address : 0x6D:1100101(x)
 * -> I2C Sensor:BME280    = I2C Address : 0x76:1110110(x)
 * -> I2C Sebsor:SHT31     = I2C Address : 0x44:1000100(x)/0x45:1010101(x)
 * SPI Interface SD Card
 * -> SD_CS                = IO4
 * -> SPI_MISO             = IO19S
 * -> SPI_MOSI             = IO23
 * -> SPI_SCK              = IO18
 * UART2 RS485 Half Duplex Auto Direction
 * -> IO26                 = RX2
 * -> IO27                 = TX2
 * User Switch
 * -> IO36                 = USER_SW
 * RTC Interrupt
 * -> IO39                 = RTC_INT#
 ********************************************************************************/
#include <Wire.h>
#include <HardwareSerial.h>
#define SerialDebug Serial  // USB Serial(Serial0)
#define SerialRS485_RX_PIN 26
#define SerialRS485_TX_PIN 27
#define SerialRS485 Serial2  // Serial2(IO27=TXD,IO26=RXD)
#define SerialLora_RX_PIN 14
#define SerialLora_TX_PIN 13
#define SerialLora Serial1  // Serial1(IO13=TXD,IO14=RXD)
#define LORA_RES_PIN 33  // ESP32-WROVER :IO33(LoRa-RESET)
#define LORA_RES_PRESS LOW
#define LORA_RES_RELEASE HIGH
#define I2C_SCL_PIN 22  // ESP32-WROVER : IO22(SCL1)
#define I2C_SDA_PIN 21  // ESP32-WROVER : IO21(SDA1)
#define LED_PIN 2  // ESP-WROVER  : IO2
#define LedON 1
#define LedOFF 0
#define USER_SW_PIN 36  // ESP32-WROVER :IO36
#define SW_PRESS LOW
#define SW_RELEASE HIGH
#define RTC_INT_PIN 39  // ESP32-WROVER :IO39
#define RTC_INT_ACTIVE LOW
#define RTC_INT_DEACTIVE HIGH
```

```
// End of Default Hardware : ET-ESP32(WROVER) RS485 V2
// Demo RS485 Modbus RTU Interface Soil Moisture Sensor(SOIL MOISTURE-H MODBUS RTU)
// Red    = +5V or 24V(3.6-30VDC)
// Black  = GND
// White  = RS485(B)
// Yellow = RS485(A)
// Green  = NC
// InputRegister[1] = Soil Moisture INT16 Value
// HoldingRegister[512] = Soil Moisture Sensor Slave ID
#include "ModbusMaster.h"  // https://github.com/4-20ma/ModbusMaster
ModbusMaster node;  // instantiate ModbusMaster object
uint8_t result;
float soil_moisture_float_value;
void setup() {
  // Start of Initial Default Hardware : ET-ESP32(WROVER) RS485 V2
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LedOFF);
  pinMode(USER_SW_PIN, INPUT_PULLUP);
  pinMode(RTC_INT_PIN, INPUT_PULLUP);
  Wire.begin(I2C_SDA_PIN, I2C_SCL_PIN);
  SerialDebug.begin(115200);
  while (!SerialDebug)    ;
  SerialDebug.println();
  SerialDebug.println("ET-ESP32(WROVER)RS485 V2.....Ready");
  SerialDebug.println();
  SerialDebug.println("ET-ESP32(WROVER)RS485 V2...Demo RS485 Modbus Master Library");
  SerialDebug.println("Interface...Soil Moisture-H Modbus RTU");
  SerialRS485.begin(9600, SERIAL_8N1, SerialRS485_RX_PIN, SerialRS485_TX_PIN);
  while (!SerialRS485)    ;
  node.begin(1, SerialRS485);  // Soil Moisture = Modbus slave ID 1
  pcf8574.pinMode(0, OUTPUT);
  pcf8574.pinMode(1, OUTPUT);
  pcf8574.pinMode(2, OUTPUT);
  pcf8574.pinMode(3, OUTPUT);
  pcf8574.pinMode(4, INPUT_PULLUP);
  pcf8574.pinMode(5, INPUT_PULLUP);
  pcf8574.pinMode(6, INPUT_PULLUP);
  pcf8574.pinMode(7, INPUT_PULLUP);
  pcf8574.begin();
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);}
int Counter = 0;
void loop() {
  uint8_t result;
  uint16_t data[2];
  Blynk.run();
  Serial.println("get data");
  result = node.readInputRegisters(1, 2);
  if (result == node.ku8MBSuccess) {
    Serial.print("Temp: ");
    Serial.println(node.getResponseBuffer(0) / 10.0f);
    Serial.print("Humi: ");
    Serial.println(node.getResponseBuffer(1) / 10.0f);
    Serial.println();
    Blynk.virtualWrite(V1, node.getResponseBuffer(0) / 10.0f);
    Blynk.virtualWrite(V2, node.getResponseBuffer(1) / 10.0f);  }
  if (pcf8574.digitalRead(P4) == LOW) {
    delay(20);
    while (pcf8574.digitalRead(P4) == LOW)
      delay(50);
    Counter++;
    delay(10);
    Serial.println(Counter);
    pcf8574.digitalWrite(P1, Counter % 2);  }
  delay(250);}
BLYNK_WRITE(V3)
{
  int pinValue = param.asInt();  // assigning incoming value from pin V1 to a variable
  pcf8574.digitalWrite(P0, !pinValue);
  if (pinValue == 1) {
    // do something when button is pressed;
    Serial.println("White Lamp is ON");
  } else if (pinValue == 0) {
    Serial.println("White Lamp is OFF");  }
}
```

```
BLYNK_WRITE(V4)
{
  int pinValue = param.asInt();  // assigning incoming value from pin V1 to a variable
  pcf8574.digitalWrite(P1, !pinValue);
  if (pinValue == 1) {
    Counter = 0;
    Serial.println("Green Lamp is ON");
  } else if (pinValue == 0) {
    Counter = 1;
    Serial.println("Green Lamp is OFF");  }}
```

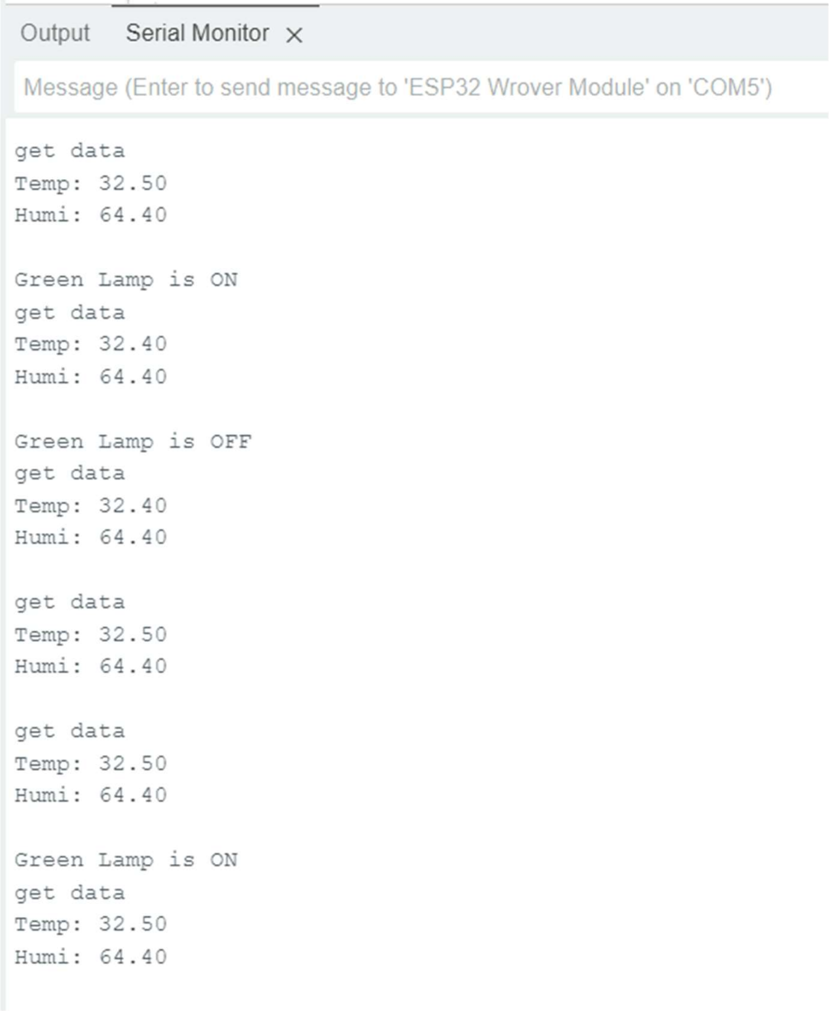
ModbusMaster.cpp   file

#Same code at 10.1 or https://github.com/miaw88/CPE_X_PJ/blob/main/meet-1of5/10.3/10.3/ModbusMaster.cpp

ModbusMaster.h   file

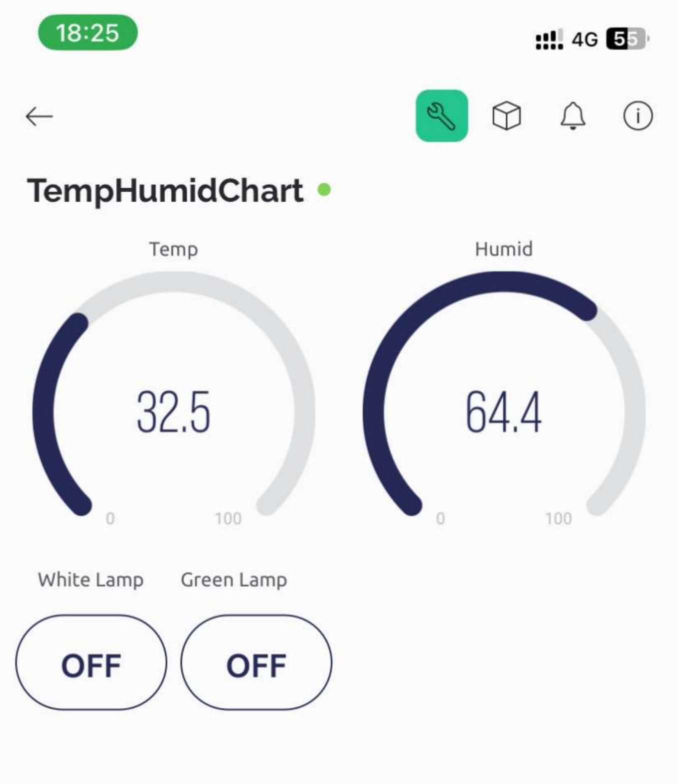
#Same code at 10.1 or https://github.com/miaw88/CPE_X_PJ/blob/main/meet-1of5/10.3/10.3/ModbusMaster.h

โปรแกรมที่ใช้ทดสอบ

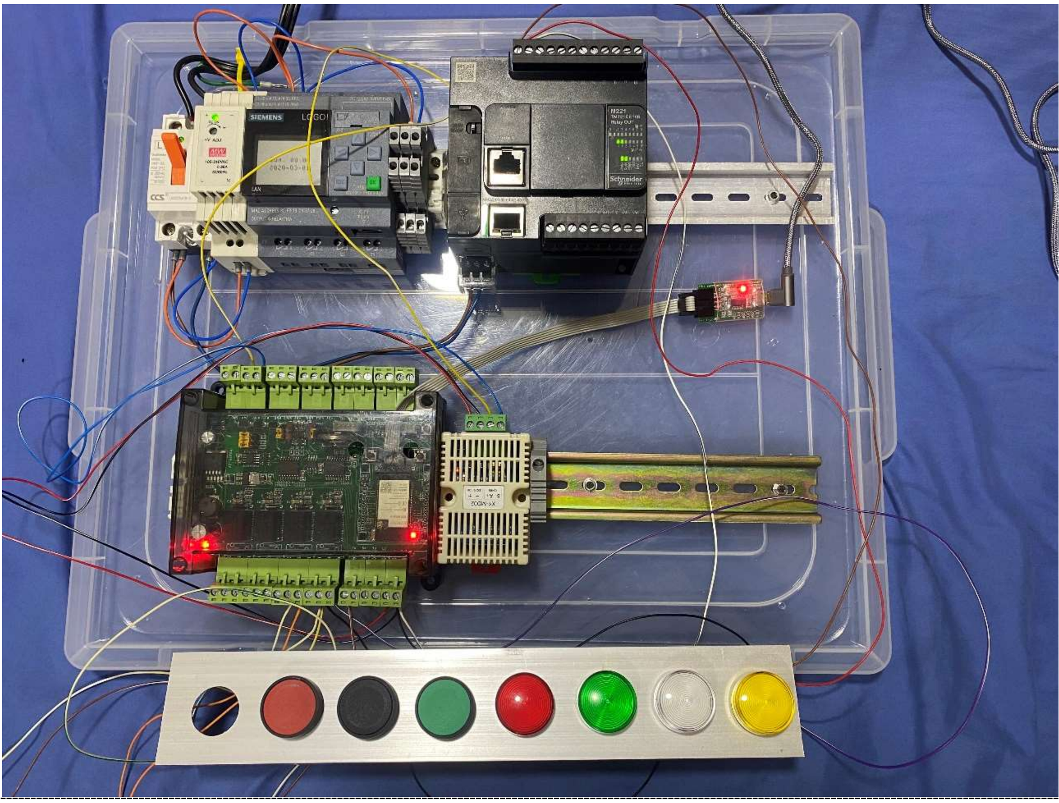


Message (Enter to send message to 'ESP32 Wrover Module' on 'COM5')

```
get data
Temp: 32.50
Humi: 64.40

Green Lamp is ON
get data
Temp: 32.40
Humi: 64.40

Green Lamp is OFF
get data
Temp: 32.40
Humi: 64.40

get data
Temp: 32.50
Humi: 64.40

get data
Temp: 32.50
Humi: 64.40

Green Lamp is ON
get data
Temp: 32.50
Humi: 64.40
```



วงจรที่ใช้ในการทดสอบ



Link

https://youtu.be/ixZXoYj2IxE?si=aoEBXi1kiM-TT2-u

10.4 ควบการปิดเปิด Yellow Lamp ผ่าน Smartt Phone และ Black Switch

Code Main Part

```
#define BLYNK_TEMPLATE_ID "TMPL6g3pjkHqp"
#define BLYNK_TEMPLATE_NAME "TempHum"
#define BLYNK_AUTH_TOKEN "74O-g_0xMdV_5Aly2IUbKDoSXIm3AKtn"
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include "Arduino.h"
#include "PCF8574.h"  // https://github.com/xreef/PCF8574_library
#define I2C_Address 0x20
#define I2C_SDA_Pin 21
#define I2C_SCL_Pin 22
// Instantiate Wire for generic use at 100kHz
TwoWire I2Ctwo = TwoWire(1);
// Set i2c address
PCF8574 pcf8574(&I2Ctwo, I2C_Address, I2C_SDA_Pin, I2C_SCL_Pin);
// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "EM Anu";
char pass[] = "anuwat11";
/******************************************************************************
 * ET-ESP32(WROVER) RS485 V2
 * Tools->Board:"ESP32 Wrover Module"
 ******************************************************************************
 * I2C Interface & I2C Bus
 * -> IO22                 = I2C_SCL
 * -> IO21                 = I2C_SDA
 * -> I2C RTC:DS3231       = I2C Address : 0x68:1100100(x)
 * -> I2C EEPROM 24LC16    = I2C Address : 0x50:1010000(x)
 * -> I2C ADC MCP3423      = I2C Address : 0x6D:1100101(x)
 * -> I2C Sensor:BME280    = I2C Address : 0x76:1110110(x)
 * -> I2C Sebsor:SHT31     = I2C Address : 0x44:1000100(x)/0x45:1010101(x)
 * SPI Interface SD Card
 * -> SD_CS                = IO4
 * -> SPI_MISO             = IO19S
 * -> SPI_MOSI            = IO23
 * -> SPI_SCK             = IO18
 * UART2 RS485 Half Duplex Auto Direction
 * -> IO26                = RX2
 * -> IO27                = TX2
 * User Switch
 * -> IO36                = USER_SW
 * RTC Interrupt
 * -> IO39                = RTC_INT#
 ********************************************************************************/
#include <Wire.h>
#include <HardwareSerial.h>
#define SerialDebug Serial  // USB Serial(Serial0)
#define SerialRS485_RX_PIN 26
#define SerialRS485_TX_PIN 27
#define SerialRS485 Serial2  // Serial2(IO27=TXD,IO26=RXD)
#define SerialLora_RX_PIN 14
#define SerialLora_TX_PIN 13
#define SerialLora Serial1  // Serial1(IO13=TXD,IO14=RXD)
#define LORA_RES_PIN 33  // ESP32-WROVER :IO33(LoRa-RESET)
#define LORA_RES_PRESS LOW
#define LORA_RES_RELEASE HIGH
#define I2C_SCL_PIN 22  // ESP32-WROVER : IO22(SCL1)
#define I2C_SDA_PIN 21  // ESP32-WROVER : IO21(SDA1)
#define LED_PIN 2  // ESP-WROVER  : IO2
#define LedON 1
#define LedOFF 0
#define USER_SW_PIN 36  // ESP32-WROVER :IO36
#define SW_PRESS LOW
#define SW_RELEASE HIGH
#define RTC_INT_PIN 39  // ESP32-WROVER :IO39
#define RTC_INT_ACTIVE LOW
#define RTC_INT_DEACTIVE HIGH
// End of Default Hardware : ET-ESP32(WROVER) RS485 V2
// Demo RS485 Modbus RTU Interface Soil Moisture Sensor(SOIL MOISTURE-H MODBUS RTU)
// Red    = +5V or 24V(3.6-30VDC)
// Black  = GND
// White  = RS485(B)
// Yellow = RS485(A)
// Green  = NC
// InputRegister[1] = Soil Moisture INT16 Value
// HoldingRegister[512] = Soil Moisture Sensor Slave ID
#include "ModbusMaster.h"  // https://github.com/4-20ma/ModbusMaster
ModbusMaster node;  // instantiate ModbusMaster object
uint8_t result;
float soil_moisture_float_value;
void setup() {
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LedOFF);
  pinMode(USER_SW_PIN, INPUT_PULLUP);
  pinMode(RTC_INT_PIN, INPUT_PULLUP);
  Wire.begin(I2C_SDA_PIN, I2C_SCL_PIN);
  SerialDebug.begin(115200);
  while (!SerialDebug)
    ;
  SerialDebug.println();
  SerialDebug.println("ET-ESP32(WROVER)RS485 V2.....Ready");
  SerialDebug.println();
  SerialDebug.println("ET-ESP32(WROVER)RS485 V2...Demo RS485 Modbus Master Library");
  SerialDebug.println("Interface...Soil Moisture-H Modbus RTU");
  SerialRS485.begin(9600, SERIAL_8N1, SerialRS485_RX_PIN, SerialRS485_TX_PIN);
  while (!SerialRS485)
    ;
  node.begin(1, SerialRS485);  // Soil Moisture = Modbus slave ID 1
  pcf8574.pinMode(0, OUTPUT);
  pcf8574.pinMode(1, OUTPUT);
  pcf8574.pinMode(2, OUTPUT);
  pcf8574.pinMode(3, OUTPUT);
  pcf8574.pinMode(4, INPUT_PULLUP);
  pcf8574.pinMode(5, INPUT_PULLUP);
  pcf8574.pinMode(6, INPUT_PULLUP);
  pcf8574.pinMode(7, INPUT_PULLUP);
  pcf8574.begin();
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);}
```

```
int Counter = 0;
int Counter2 = 0;
void loop() {
  uint8_t result;
  uint16_t data[2];
  Blynk.run();
  Serial.println("get data");
  result = node.readInputRegisters(1, 2);
  if (result == node.ku8MBSuccess) {
    Serial.print("Temp: ");
    Serial.println(node.getResponseBuffer(0) / 10.0f);
    Serial.print("Humi: ");
    Serial.println(node.getResponseBuffer(1) / 10.0f);
    Serial.println();
    Blynk.virtualWrite(V1, node.getResponseBuffer(0) / 10.0f);
    Blynk.virtualWrite(V2, node.getResponseBuffer(1) / 10.0f);
  }
  if (pcf8574.digitalRead(P4) == LOW) {
    delay(20);
    while (pcf8574.digitalRead(P4) == LOW)
      delay(50);
    Counter++;
    delay(10);
    Serial.println(Counter);
    pcf8574.digitalWrite(P1, Counter % 2);
  }
  if (pcf8574.digitalRead(P5) == LOW) {
    delay(20);
    while (pcf8574.digitalRead(P5) == LOW)
      delay(50);
    Counter2++;
    delay(10);
    Serial.println(Counter2);
    pcf8574.digitalWrite(P2, Counter2 % 2);
  }
  delay(250);
}
```

```
BLYNK_WRITE(V3)
{
  int pinValue = param.asInt();  // assigning incoming value from pin V1 to a variable
  pcf8574.digitalWrite(P0, !pinValue);
  if (pinValue == 1) {
    // do something when button is pressed;
    Serial.println("White Lamp is ON");
  } else if (pinValue == 0) {
    Serial.println("White Lamp is OFF");
  }
}
BLYNK_WRITE(V4)
{
  int pinValue = param.asInt();  // assigning incoming value from pin V1 to a variable
  pcf8574.digitalWrite(P1, !pinValue);
  if (pinValue == 1) {
    Counter = 0;
    Serial.println("Green Lamp is ON");
  } else if (pinValue == 0) {
    Counter = 1;
    Serial.println("Green Lamp is OFF");
  }
}
BLYNK_WRITE(V5)
{
  int pinValue = param.asInt();  // assigning incoming value from pin V1 to a variable
  pcf8574.digitalWrite(P2, !pinValue);
  if (pinValue == 1) {
    Counter2 = 0;
    Serial.println("Yellow Lamp is ON");
  } else if (pinValue == 0) {
    Counter2 = 1;
    Serial.println("Yellow Lamp is OFF");
  }
}
```

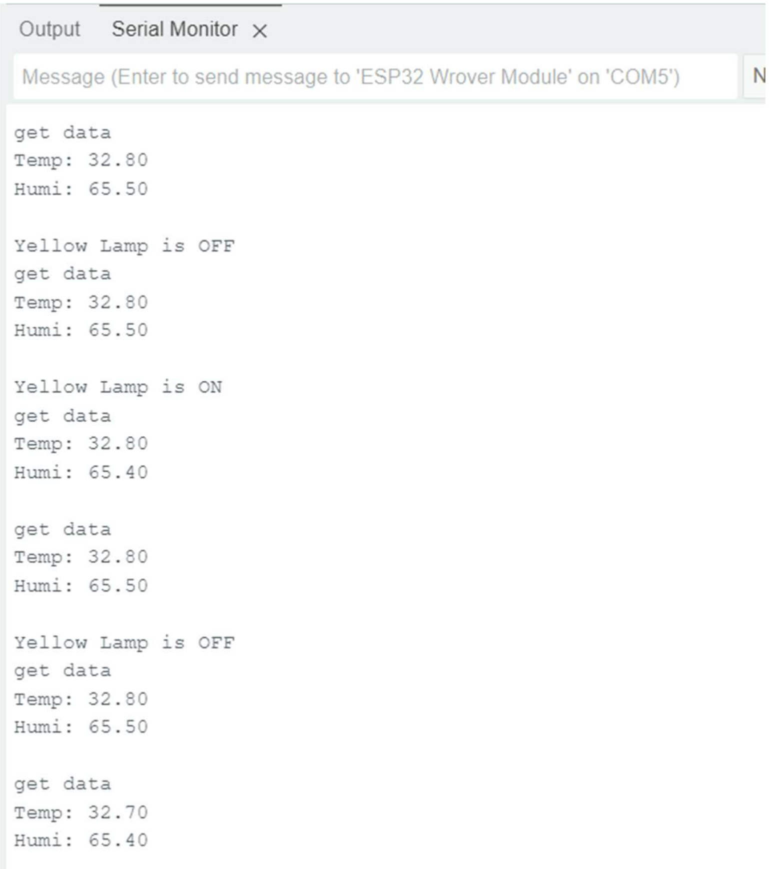
ModbusMaster.cpp file

#Same code at 10.1 or https://github.com/miaw88/CPE_X_PJ/blob/main/meet-1of5/10.4/10.4/ModbusMaster.cpp

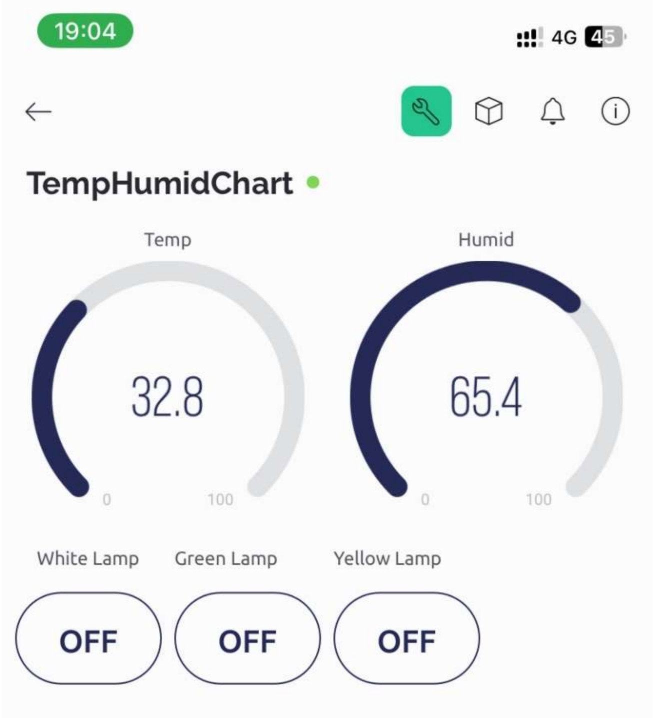
ModbusMaster.h file

#Same code at 10.1 or https://github.com/miaw88/CPE_X_PJ/blob/main/meet-1of5/10.4/10.4/ModbusMaster.h

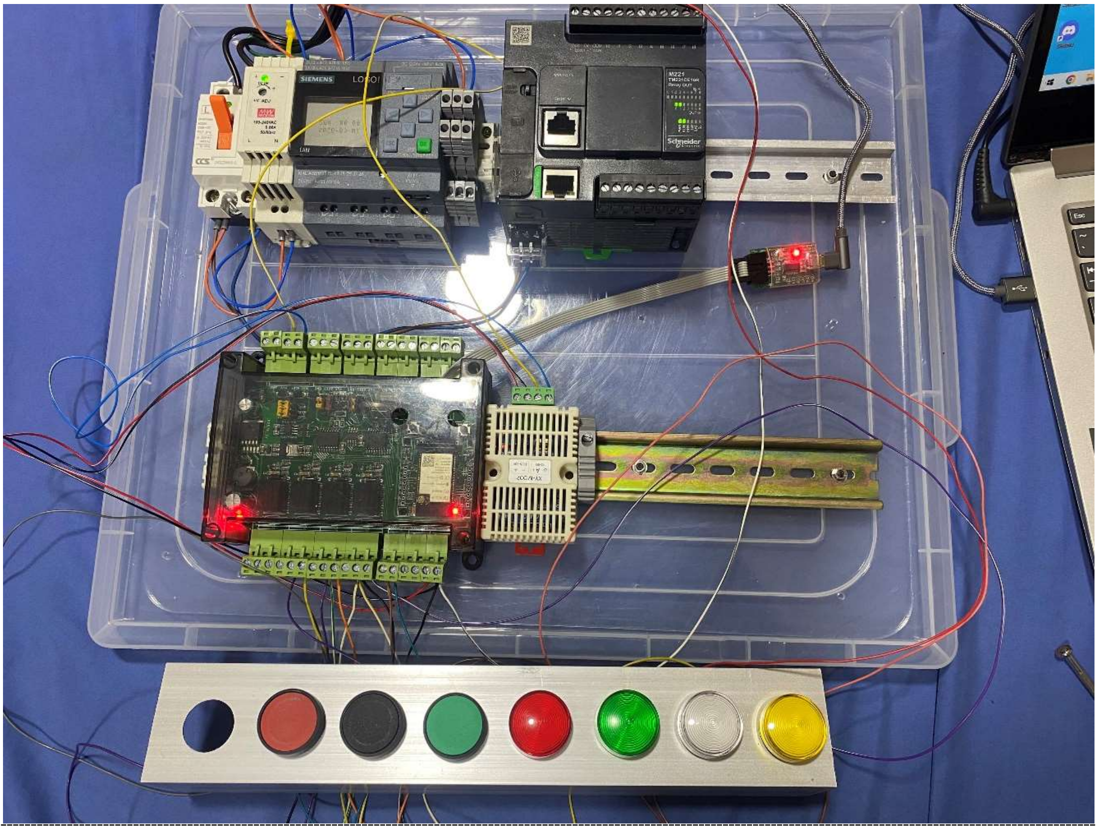
โปรแกรมที่ใช้ทดสอบ


Output
Serial Monitor ×
Message (Enter to send message to 'ESP32 Wrover Module' on 'COM5')
N

```
get data
Temp: 32.80
Humi: 65.50

Yellow Lamp is OFF
get data
Temp: 32.80
Humi: 65.50

Yellow Lamp is ON
get data
Temp: 32.80
Humi: 65.40

get data
Temp: 32.80
Humi: 65.50

Yellow Lamp is OFF
get data
Temp: 32.80
Humi: 65.50

get data
Temp: 32.70
Humi: 65.40
```



วงจรที่ใช้ในการทดสอบ



Link

https://youtu.be/XdzcEci5PZ8?si=nG0VbgYjEdBBtquc

10.5 ควบการปิดเปิด Red Lamp ผ่าน Smartt Phone และ Red Switch

Code Main Part

```
#define BLYNK_TEMPLATE_ID "TMPL6g3pjkHqp"
#define BLYNK_TEMPLATE_NAME "TempHum"
#define BLYNK_AUTH_TOKEN "74O-g_0xMdV_5Aly2IUbKDoSXIm3AKtn"
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include "Arduino.h"
#include "PCF8574.h"  // https://github.com/xreef/PCF8574_library
#define I2C_Address 0x20
#define I2C_SDA_Pin 21
#define I2C_SCL_Pin 22
// Instantiate Wire for generic use at 100kHz
TwoWire I2Ctwo = TwoWire(1);
// Set i2c address
PCF8574 pcf8574(&I2Ctwo, I2C_Address, I2C_SDA_Pin, I2C_SCL_Pin);
// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "EM Anu";
char pass[] = "anuwat11";
/********************************************************************************
 * ET-ESP32(WROVER) RS485 V2
 * Tools->Board:"ESP32 Wrover Module"
 ********************************************************************************
 * I2C Interface & I2C Bus
 * -> IO22                  = I2C_SCL
 * -> IO21                  = I2C_SDA
 * -> I2C RTC:DS3231        = I2C Address : 0x68:1100100(x)
 * -> I2C EEPROM 24LC16     = I2C Address : 0x50:1010000(x)
 * -> I2C ADC MCP3423       = I2C Address : 0x6D:1100101(x)
 * -> I2C Sensor:BME280     = I2C Address : 0x76:1110110(x)
 * -> I2C Sebsor:SHT31      = I2C Address : 0x44:1000100(x)/0x45:1010101(x)
 * SPI Interface SD Card
 * -> SD_CS                 = IO4
 * -> SPI_MISO              = IO19S
 * -> SPI_MOSI            = IO23
 * -> SPI_SCK             = IO18
 * UART2 RS485 Half Duplex Auto Direction
 * -> IO26                = RX2
 * -> IO27                = TX2
 * User Switch
 * -> IO36                = USER_SW
 * RTC Interrupt
 * -> IO39                = RTC_INT#
 ******************************************************************************/
#include <Wire.h>
#include <HardwareSerial.h>
#define SerialDebug Serial  // USB Serial(Serial0)
#define SerialRS485_RX_PIN 26
#define SerialRS485_TX_PIN 27
#define SerialRS485 Serial2  // Serial2(IO27=TXD,IO26=RXD)
#define SerialLora_RX_PIN 14
#define SerialLora_TX_PIN 13
#define SerialLora Serial1  // Serial1(IO13=TXD,IO14=RXD)
#define LORA_RES_PIN 33  // ESP32-WROVER :IO33(LoRa-RESET)
#define LORA_RES_PRESS LOW
#define LORA_RES_RELEASE HIGH
#define I2C_SCL_PIN 22  // ESP32-WROVER : IO22(SCL1)
#define I2C_SDA_PIN 21  // ESP32-WROVER : IO21(SDA1)
#define LED_PIN 2  // ESP-WROVER  : IO2
#define LedON 1
#define LedOFF 0
#define USER_SW_PIN 36  // ESP32-WROVER :IO36
#define SW_PRESS LOW
#define SW_RELEASE HIGH
#define RTC_INT_PIN 39  // ESP32-WROVER :IO39
#define RTC_INT_ACTIVE LOW
#define RTC_INT_DEACTIVE HIGH
// End of Default Hardware : ET-ESP32(WROVER) RS485 V2
// Demo RS485 Modbus RTU Interface Soil Moisture Sensor(SOIL MOISTURE-H MODBUS RTU)
// Red    = +5V or 24V(3.6-30VDC)
// Black  = GND
// White  = RS485(B)
// Yellow = RS485(A)
// Green  = NC
//==========================================================================
======================
// InputRegister[1] = Soil Moisture INT16 Value
// HoldingRegister[512] = Soil Moisture Sensor Slave ID
#include "ModbusMaster.h"  // https://github.com/4-20ma/ModbusMaster
ModbusMaster node;  // instantiate ModbusMaster object
uint8_t result;
float soil_moisture_float_value;
void setup() {
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LedOFF);
  pinMode(USER_SW_PIN, INPUT_PULLUP);
  pinMode(RTC_INT_PIN, INPUT_PULLUP);
  Wire.begin(I2C_SDA_PIN, I2C_SCL_PIN);
  SerialDebug.begin(115200);
  while (!SerialDebug)    ;
  SerialDebug.println();
  SerialDebug.println("ET-ESP32(WROVER)RS485 V2.....Ready");
  SerialDebug.println();
  SerialDebug.println("ET-ESP32(WROVER)RS485 V2...Demo RS485 Modbus Master Library");
  SerialDebug.println("Interface...Soil Moisture-H Modbus RTU");
  SerialRS485.begin(9600, SERIAL_8N1, SerialRS485_RX_PIN, SerialRS485_TX_PIN);
  while (!SerialRS485)    ;
  node.begin(1, SerialRS485);  // Soil Moisture = Modbus slave ID 1
  pcf8574.pinMode(0, OUTPUT);
  pcf8574.pinMode(1, OUTPUT);
  pcf8574.pinMode(2, OUTPUT);
  pcf8574.pinMode(3, OUTPUT);
  pcf8574.pinMode(4, INPUT_PULLUP);
  pcf8574.pinMode(5, INPUT_PULLUP);
  pcf8574.pinMode(6, INPUT_PULLUP);
  pcf8574.pinMode(7, INPUT_PULLUP);
  pcf8574.begin();
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);}
```

```
int Counter = 0;
int Counter2 = 0;
int Counter3 = 0;
void loop() {
  uint8_t result;
  uint16_t data[2];
  Blynk.run();
  Serial.println("get data");
  result = node.readInputRegisters(1, 2);
  if (result == node.ku8MBSuccess) {
    Serial.print("Temp: ");
    Serial.println(node.getResponseBuffer(0) / 10.0f);
    Serial.print("Humi: ");
    Serial.println(node.getResponseBuffer(1) / 10.0f);
    Serial.println();
    Blynk.virtualWrite(V1, node.getResponseBuffer(0) / 10.0f);
    Blynk.virtualWrite(V2, node.getResponseBuffer(1) / 10.0f);
  }
  if (pcf8574.digitalRead(P4) == LOW) {
    delay(20);
    while (pcf8574.digitalRead(P4) == LOW)
      delay(50);
    Counter++;
    delay(10);
    Serial.println(Counter);
    pcf8574.digitalWrite(P1, Counter % 2);
  }
  if (pcf8574.digitalRead(P5) == LOW) {
    delay(20);
    while (pcf8574.digitalRead(P5) == LOW)
      delay(50);
    Counter2++;
    delay(10);
    Serial.println(Counter2);
    pcf8574.digitalWrite(P2, Counter2 % 2);
  }
  if (pcf8574.digitalRead(P6) == HIGH) {
    delay(20);
    while (pcf8574.digitalRead(P6) == HIGH)
      delay(50);
    Counter3++;
    delay(10);
    Serial.println(Counter3);
    pcf8574.digitalWrite(P3, Counter3 % 2);
  }
  delay(250);
}
BLYNK_WRITE(V3)
{
  int pinValue = param.asInt();  // assigning incoming value from pin V1 to a variable
  pcf8574.digitalWrite(P0, !pinValue);
  if (pinValue == 1) {
    // do something when button is pressed;
    Serial.println("White Lamp is ON");
  } else if (pinValue == 0) {
    Serial.println("White Lamp is OFF");
  }
}
BLYNK_WRITE(V4)
{
  int pinValue = param.asInt();  // assigning incoming value from pin V1 to a variable
  pcf8574.digitalWrite(P1, !pinValue);
  if (pinValue == 1) {
    Counter = 0;
    Serial.println("Green Lamp is ON");
  } else if (pinValue == 0) {
    Counter = 1;
    Serial.println("Green Lamp is OFF");
  }
}
BLYNK_WRITE(V5)
{
  int pinValue = param.asInt();  // assigning incoming value from pin V1 to a variable
  pcf8574.digitalWrite(P2, !pinValue);
  if (pinValue == 1) {
    Counter2 = 0;
    Serial.println("Yellow Lamp is ON");
  } else if (pinValue == 0) {
    Counter2 = 1;
    Serial.println("Yellow Lamp is OFF");
  }
}
BLYNK_WRITE(V6)
{
  int pinValue = param.asInt();  // assigning incoming value from pin V1 to a variable
  pcf8574.digitalWrite(P3, !pinValue);
  if (pinValue == 1) {
    Counter3 = 0;
    Serial.println("Red Lamp is ON");
  } else if (pinValue == 0) {
    Counter3 = 1;
    Serial.println("Red Lamp is OFF");
  }
}
```

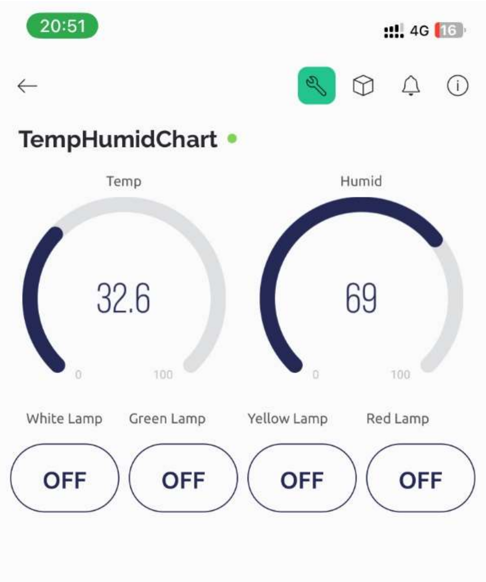
ModbusMaster.cpp file

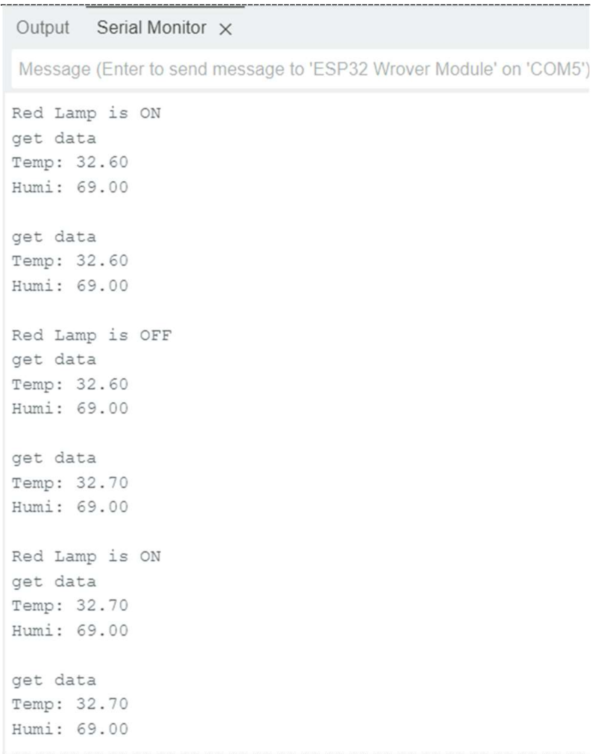
#Same code at 10.1 or https://github.com/miaw88/CPE_X_PJ/blob/main/meet-1of5/10.5/10.5/ModbusMaster.cpp

ModbusMaster.h file

#Same code at 10.1 or https://github.com/miaw88/CPE_X_PJ/blob/main/meet-1of5/10.5/10.5/ModbusMaster.h

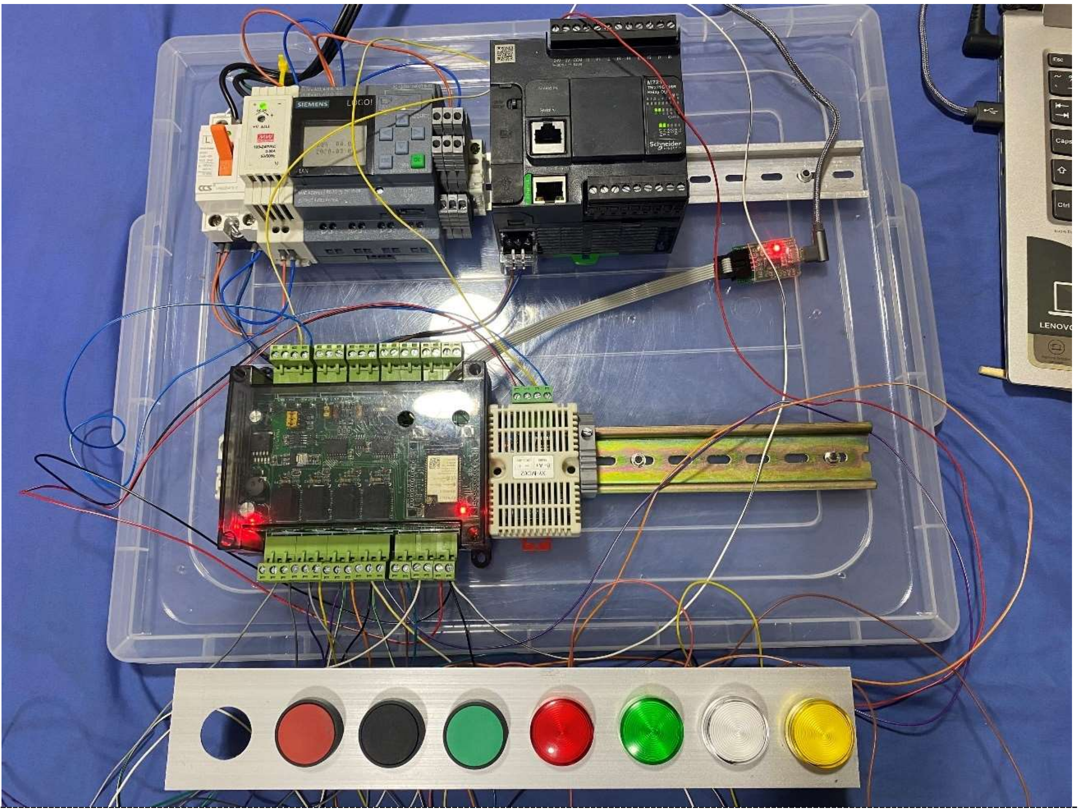
โปรแกรมที่ใช้ทดสอบ




Output
Serial Monitor ×
Message (Enter to send message to 'ESP32 Wrover Module' on 'COM5')

```
Red Lamp is ON
get data
Temp: 32.60
Humi: 69.00

get data
Temp: 32.60
Humi: 69.00

Red Lamp is OFF
get data
Temp: 32.60
Humi: 69.00

get data
Temp: 32.70
Humi: 69.00

Red Lamp is ON
get data
Temp: 32.70
Humi: 69.00

get data
Temp: 32.70
Humi: 69.00
```

วงจรที่ใช้ในการทดสอบ



Link

https://youtu.be/N57kLhYdG2U?si=-FNAAj9vJvM6T0yT

10.6 สไลด์นำเสนอ พร้อมวิดีโอ

Canvas https://www.canva.com/design/DAGA34QUmyl/YziMwwwRb4ZYV8B_K8kRGQ/edit?fbclid=IwAR3DGbLMgmJdxcR9Dzil9LTFdmm1p8KbOfSH0I7WExRDUMFQ-KnVlhixCRk_aem_AU9wPQ_H8RKEH-he_DVMWAScokq4LRRZvLLsrbv7YkZXDe7tqi4YrCSw5bgjkLquQt1kol4BlxD7PKzrflWCG2QV
Youtube **https://youtu.be/4diO4QMjdso?si=3dKldc-K57vROI7R**

11. เติมรูปถ่ายในการทำงาน โปรแกรมที่ใช้ในการทดสอบ วงจรที่ใช้ในการทดสอบ ในเอกสารนี้แล้ว Save As เป็น pdf ไฟล์
12. สมาชิกกลุ่มเข้า ZOOM Meeting ประชุมและบันทึกการประชุม
    12.1 นำเสนอว่า โปรเจ็ต ทำอะไรไปบ้าง จะทำอะไร ปัญหาที่พบ แนวทางแก้ไข
    12.2 นำเสนอว่า แต่ละคน ทำอะไรไปบ้าง จะทำอะไร ปัญหาที่พบ แนวทางแก้ไข
    12.3 ความยาวระหว่าง 8-10 นาที ตัดต่อตามสมควรแล้ว Upload ขึ้น YouTube
13. ส่งงานที่ Link ด้วยการ Upload pdf Report และ YouTube Link ก่อน 20240330-0600
14. ส่งงานที่ < จะแจ้งทาง FB Group อีกครั้ง >

Week03,04 - TM221CE16 I/O and IoTs Remote I/O

Week05,06 - LOGO8 I/O and IoTs Remote I/O

Week07,08 – Raspberry Pi + Node-RED

Week09,10 – Remote Control and Monitor with IoTs and RUT200