IDEC

Application Note

FC6A Series PLC Communicating with an Arduino Uno over Modbus RTU

1. Overview

In this application, we will cover how to connect an FC6A series PLC to an Arduino Uno using Modbus RTU communications.

We will be using an FC6A Plus PLC with an FC6A-HPH1 expansion cartridge base module. An FC6A-PC3 (RS485 cartridge) will be installed in the bottom slot of the FC6A-HPH1. This will be configured as serial communications port 2 in WindLDR, with the FC6A Plus PLC set up as the Modbus RTU master:

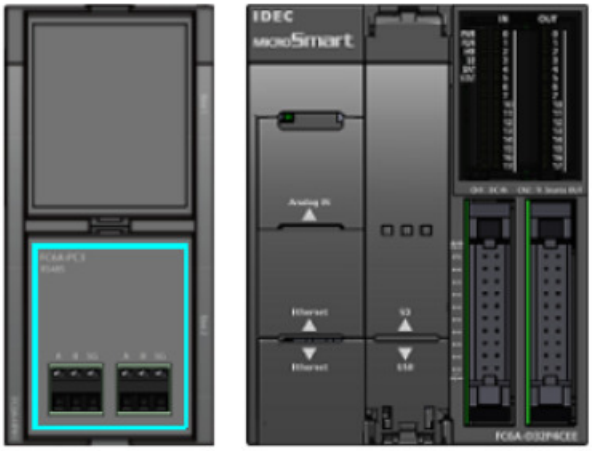


Figure 1. IDEC FC6A Plus PLC with FC6A-HPH1 and FC6A-PC3 Installed

We will be using an Arduino Uno R3 as the Modbus RTU slave:

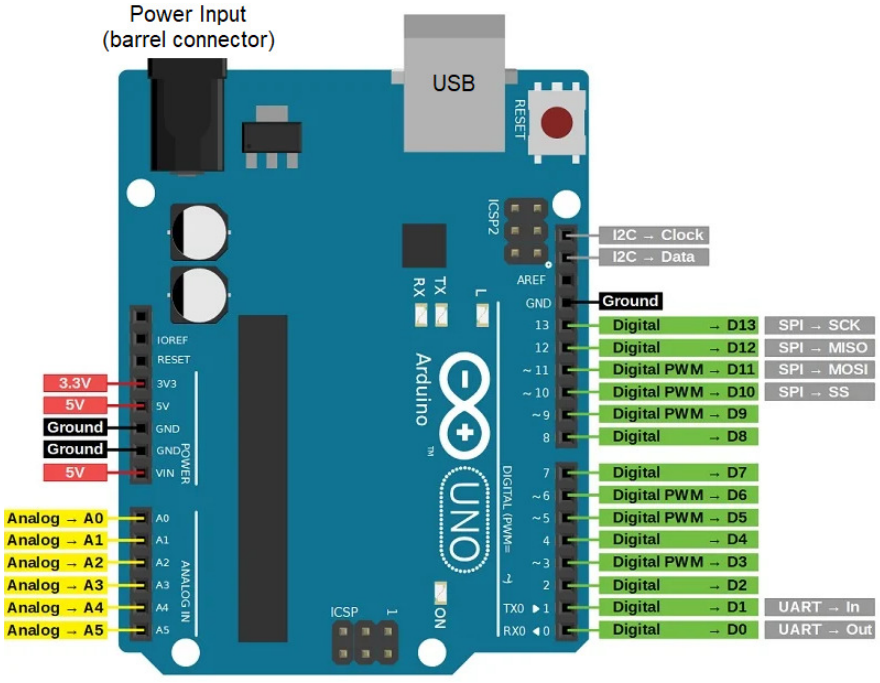


Figure 2. Arduino Uno R3 Pin Information

As the UART (serial port) on the Arduino supports TTL logic, we will also need to use a converter to transform the TTL signals into RS485:

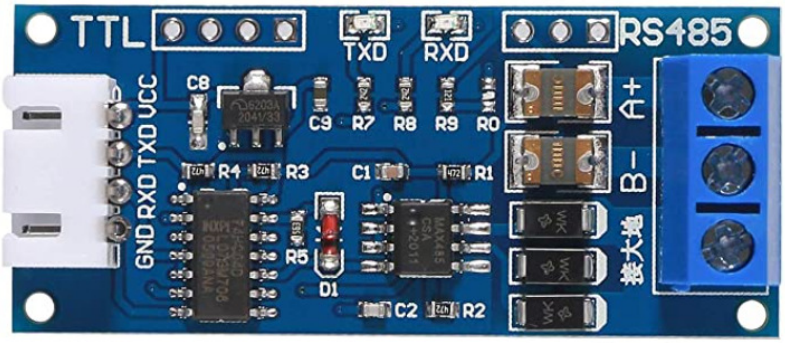


Figure 3. TTL-to-RS485 Converter

2. Wiring Connections

The Arduino is being powered via the USB connection to the computer. Note that it could have also been powered externally via a power supply connected to the power input barrel connector or to the 5V power input pin and one of the ground pins in the POWER pin section of the Arduino.

The TTL-to-RS485 converter that we are using is also powered via 5V DC.

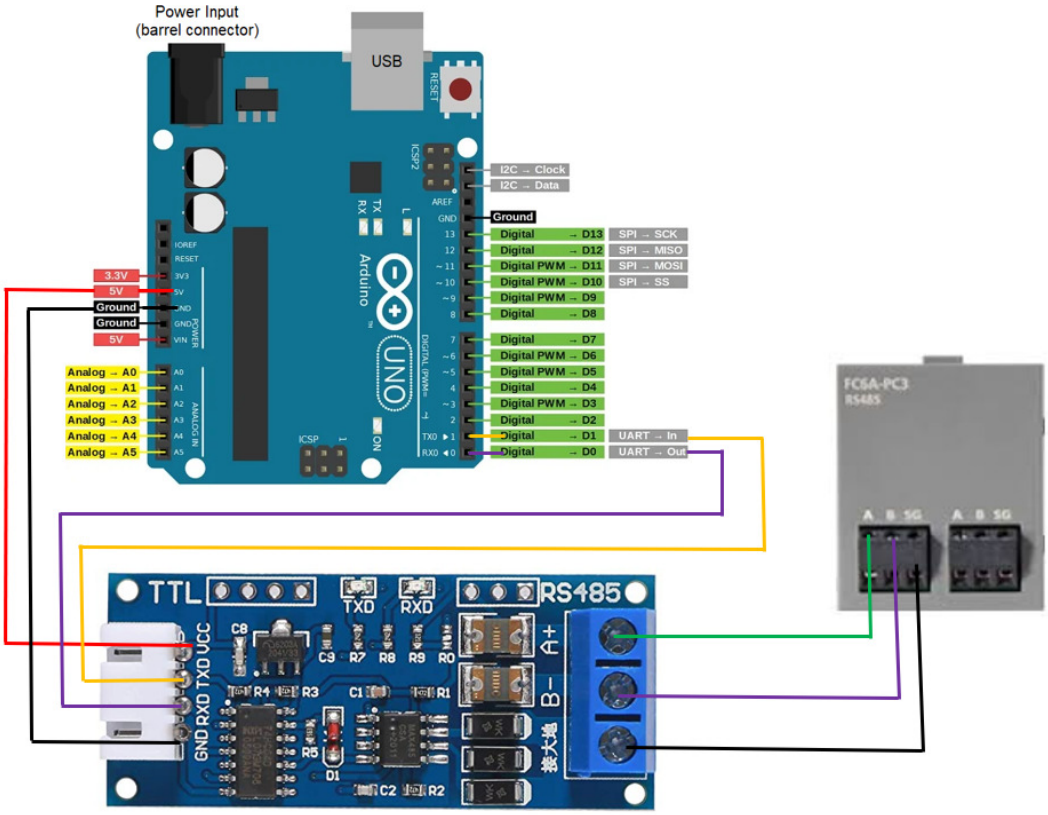


Figure 4. Wiring Diagram

3. Setting up the FC6A as a Modbus RTU Master

Now that we have everything wired together, we need to configure the FC6A to be a Modbus RTU master.

In WindLDR, click on ***Configuration → Comm Ports***:

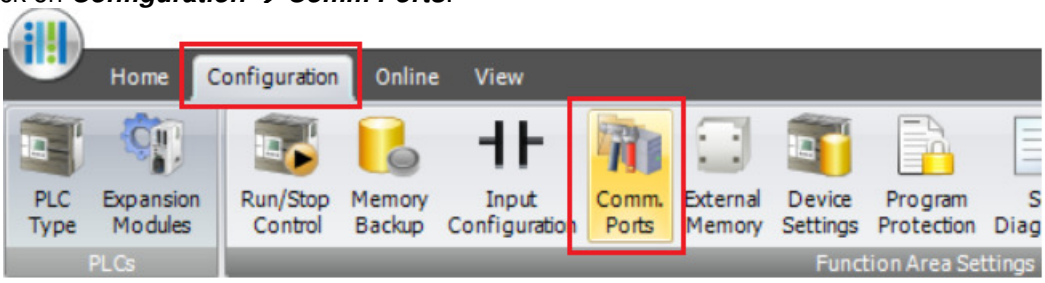


Figure 5. Selecting the Serial Communication Port

This will bring up the Communication Ports dialog box. Beside Port 2, click where it says Maintenance Protocol. A drop-down will appear where we can select Modbus RTU Master:

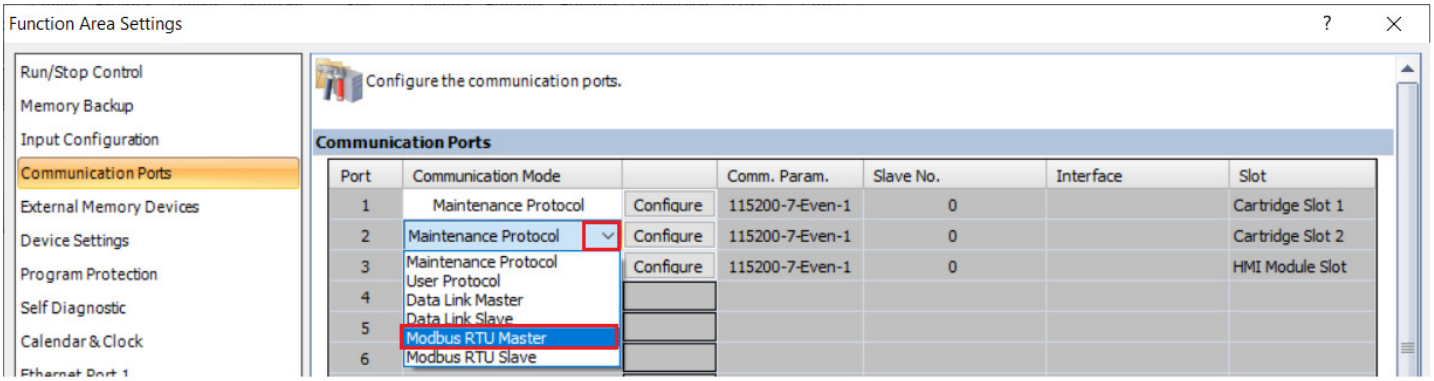


Figure 6. Configuring Port 2 as a Modbus RTU Master

This will bring up the Modbus RTU Master Request Table:

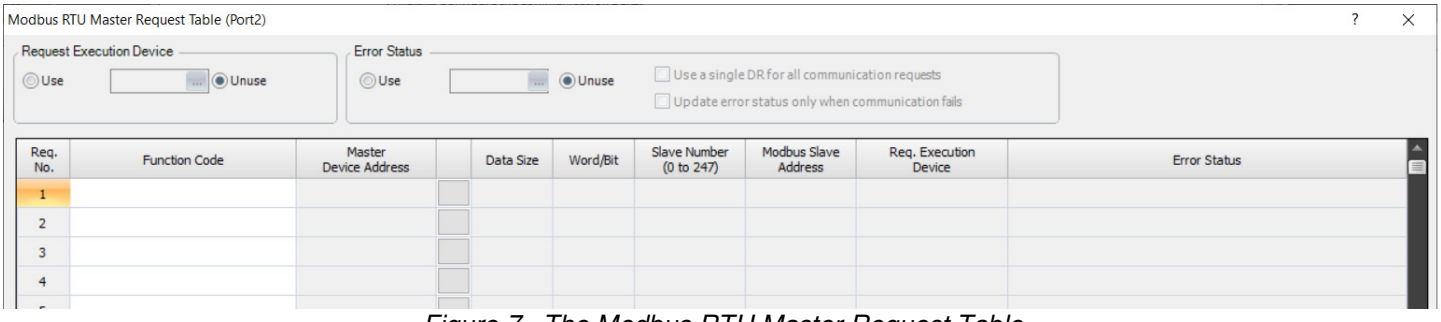


Figure 7. The Modbus RTU Master Request Table

For this application, we are going to set it up so that the PLC writes 10 registers and 10 bits to the Arduino. The Arduino will read this information and send it back to the PLC in 10 different registers and 10 different bits. This way, we can confirm that the data we send to the Arduino is indeed being received.

Read 10 Registers from the Arduino to the PLC

For the first request, we'll use a function code of *04 Read Input Registers* to read 10 registers from the Arduino into the PLC and store them in consecutive data registers starting with D0 (the data being read from the Arduino will be stored in D0-D9 in the PLC).

Read 10 Bits from the Arduino to the PLC

For the second request, we'll use a function code of *02 Read Input Status* to read 10 bits from the Arduino into the PLC and store them in consecutive internal relays starting with M0 (the data being read from the Arduino will be stored in M0-M7 and M10-M11 in the PLC – remember, internal relays use octal-based addressing, so no 8's or 9's in their addresses!).

Write 10 Registers from the PLC to the Arduino

For the third request, we'll use a function code of *16 Preset Multiple Registers* to write the values of 10 consecutive data registers starting from D50 in the PLC into the Arduino. We'll write these into holding registers starting with Modbus address 400001.

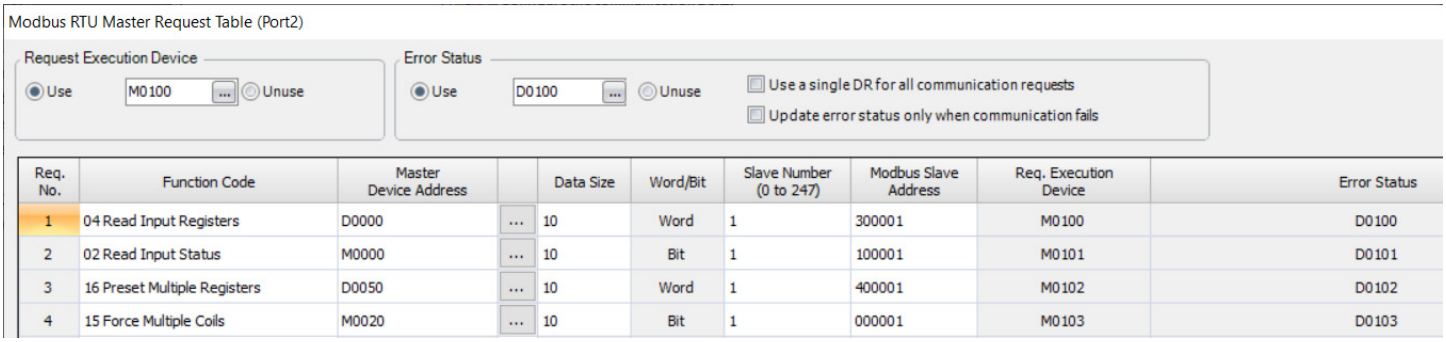
Write 10 Bits from the PLC to the Arduino

For the last request, we'll use a function code of *15 Force Multiple Coils* to write the values of 10 consecutive internal relay starting from M20 from the PLC into the Arduino. We'll write these into coils starting with Modbus address 000001.

When we get to the Arduino code, we'll assign it a Modbus slave address of 1. This needs to be referenced in the Slave Number column in the Modbus RTU Master Request Table for each request.

We'll use Request Execution Devices to control when each Modbus request gets executed. We'll start the addressing of these devices at M100, so with four Modbus requests, we'll be using M100, M101, M102 and M103. When these bits turn on, the corresponding Modbus request will be executed.

We'll also use the Error Status registers so that we can monitor the communications status of each request to make sure they are working as expected. We'll start the addressing for this at D100. With four Modbus requests, we'll be using D100, D101, D102 and D103.


Modbus RTU Master Request Table (Port2)

Request Execution Device: Use M0100 ... Unuse

Error Status: Use D0100 ... Unuse

Use a single DR for all communication requests

Update error status only when communication fails

Req. No.	Function Code	Master Device Address		Data Size	Word/Bit	Slave Number (0 to 247)	Modbus Slave Address	Req. Execution Device	Error Status
1	04 Read Input Registers	D0000	...	10	Word	1	300001	M0100	D0100
2	02 Read Input Status	M0000	...	10	Bit	1	100001	M0101	D0101
3	16 Preset Multiple Registers	D0050	...	10	Word	1	400001	M0102	D0102
4	15 Force Multiple Coils	M0020	...	10	Bit	1	000001	M0103	D0103

Figure 8. Modbus RTU Master Request Table with Four Requests

Communication Parameters

The last item that we need to configure in this table are the communication parameters – the baud rate, parity and stop bits that we want to use for Modbus communication. To set this up, click on the **Communication Settings** button in the bottom left-hand corner of the table.

We want to set the Baud Rate to 9600, the Parity to None and the Stop Bit to 1:

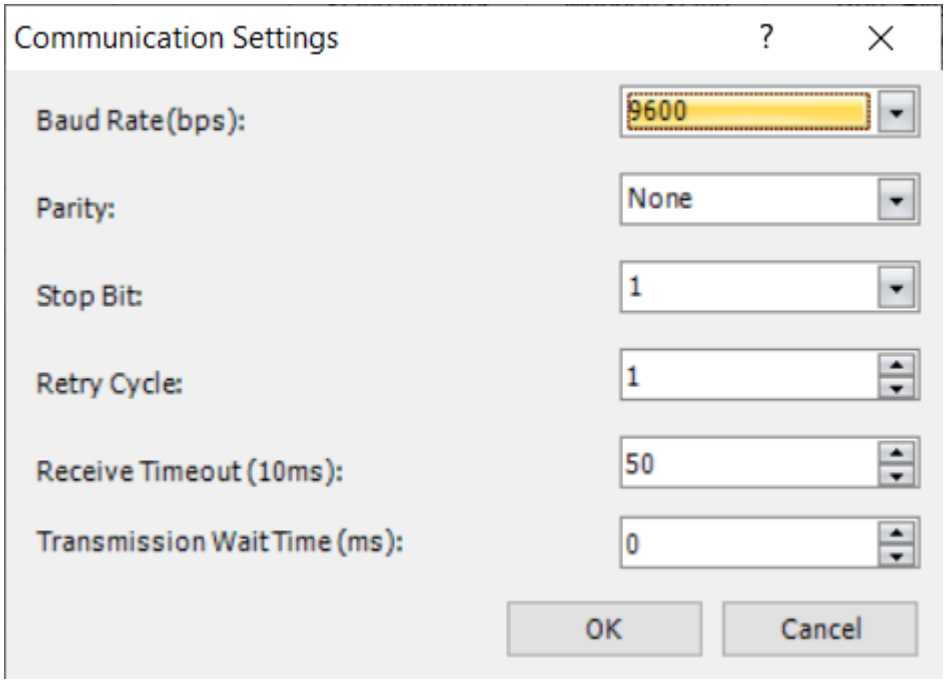


Figure 9. FC6A Modbus RTU Communication Settings

Once this is configured, click ***OK*** in the Communication Settings dialog box and click ***OK*** in the Modbus RTU Master Request Table. This will put us back in the Communications Ports dialog box, where we will see the Port 2 has been configured as a Modbus RTU Master with communication parameters of 9600 baud, 8 data bits, no parity and one stop bit:

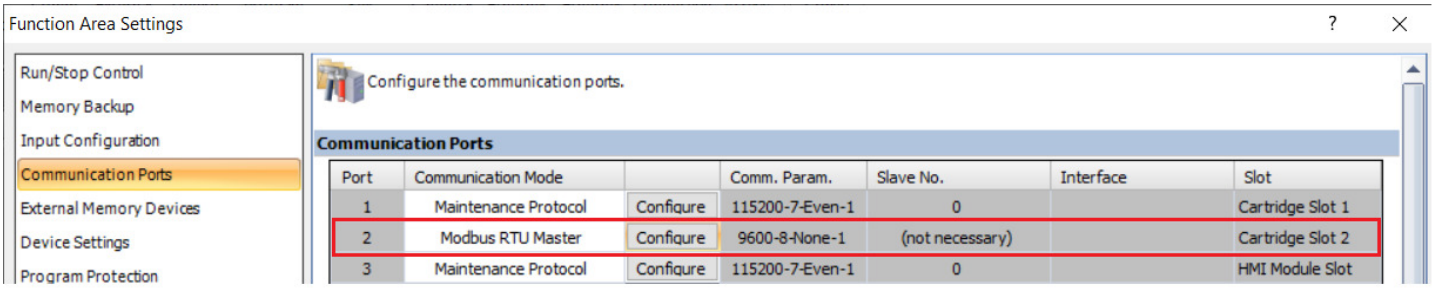


Figure 10. Communication Ports Dialog Box with Setup Completed

We can click ***OK*** in this dialog box to confirm everything.

Now that we have the Modbus communications set up, we need to write a small PLC program to control when the Modbus Request Execution Devices get turned on.

Let's say we want to execute these four Modbus requests roughly every 100 ms. We can set up a self-resetting timer to give us a pulse every 100 ms to turn on the four internal relays that we're using as the Request Execution Devices.

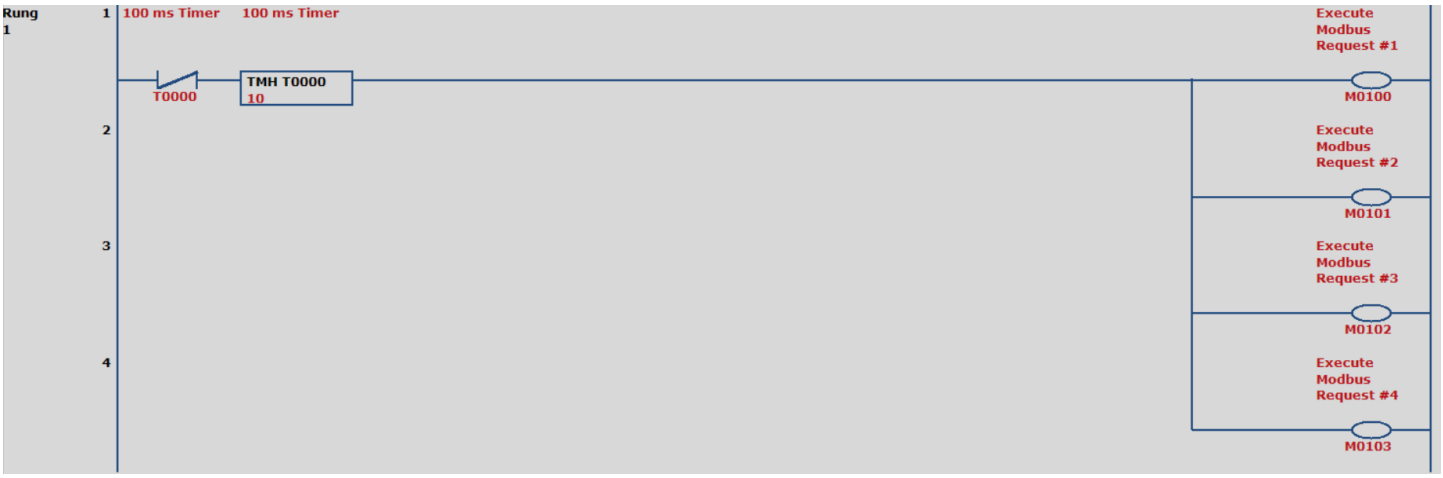


Figure 11. PLC Program

Once we have this program completed, we can download it to the PLC.

4. Setting Up the Arduino as a Modbus RTU Slave

The program for the Arduino was written using the Arduino IDE, which can be downloaded from Arduino's website: https://www.arduino.cc/en/software

Once the IDE is installed and launched, the first thing we need to do is to install the libraries that we are going to be using. For this application, we are using the ArduinoModbus and ArduinoRS485 libraries.

In the Arduino IDE, go to ***Tools → Manage Libraries***:

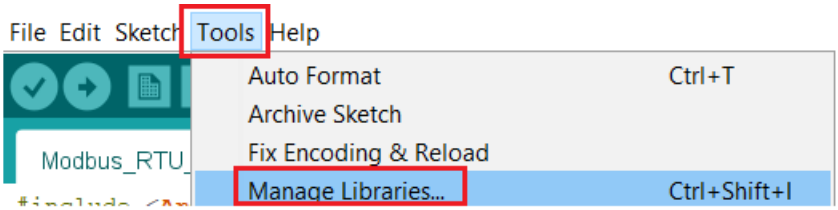


Figure 12. Manage Libraries

This will bring up the Library Manager. If you search for *ArduinoRS485*, this will bring up both libraries that we need. Click on each one and click the Install button to install each library.

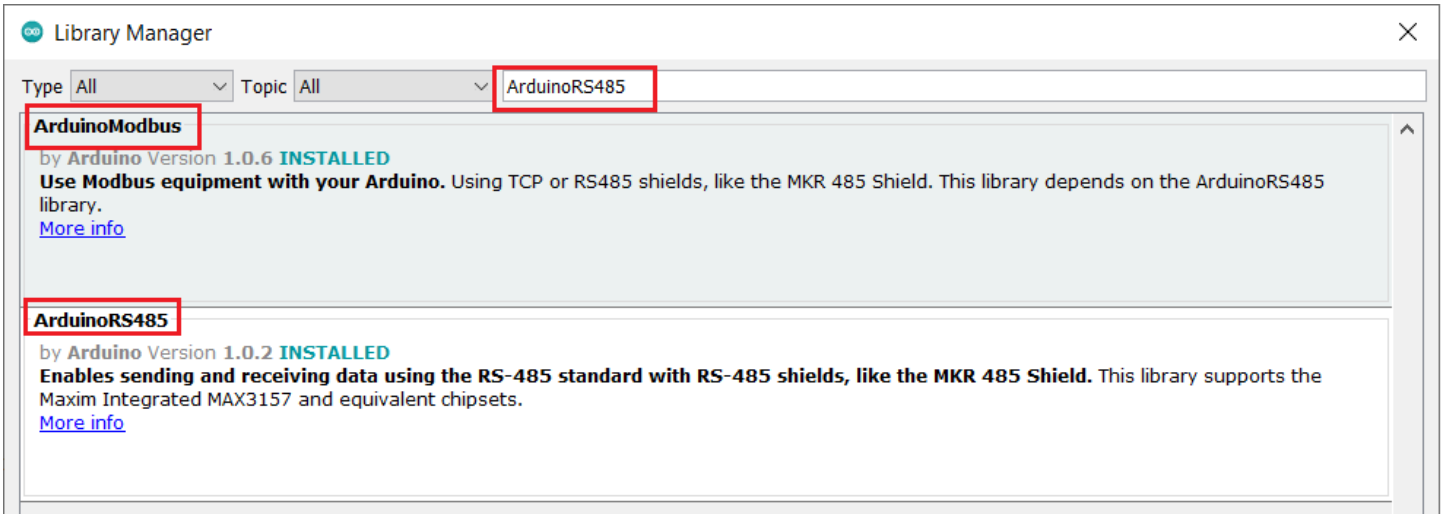


Figure 13. ArduinoModbus and ArduinoRS485 Libraries

Now, let's write the Arduino code!

First, we need to include the two libraries that we need to use:

```
#include <ArduinoRS485.h>
#include <ArduinoModbus.h>
```

Next, let's name digital pin 13 on the Arduino LED_PIN. We'll use this later in the code:

```
#define LED_PIN 13
```

Let's set up some variables to define the number of coils, discrete inputs, holding registers and input registers we'll be using. Let's also set up a variable called LED_CONTROL – we'll use this to control digital pin 13 later in the program.

```
const int numCoils = 10;
const int numDiscreteInputs = 10;
const int numHoldingRegisters = 10;
const int numInputRegisters = 10;
long LED_CONTROL = 0;
```

In the Arduino, there are two primary functions that are used – *setup* and *loop*. The setup function is where you do initialization – things you only want to happen once. The loop function is where you write the main section of your program – what you write in here will execute over and over continually.

In the setup function, we'll define pin 13 – our LED_PIN – as an output so that we can write to it. We'll also enable the serial port for 9600 baud and start the Modbus RTU server:

```
void setup() {

  pinMode(LED_PIN, OUTPUT);

  Serial.begin(9600);
  while (!Serial);
  Serial.println("Arduino as a Modbus RTU Server");

  // start the Modbus RTU server, with slave ID of 1
  if (!ModbusRTUServer.begin(1, 9600))
  {
    Serial.println("Failed to start Modbus RTU Server!");
    while (1);
  }
```

We'll also configure the Modbus coils, discrete inputs, holding registers and input registers and have them all start at the first address:

```
  // configure coils at address 0x00
  ModbusRTUServer.configureCoils(0x00, numCoils);

  // configure discrete inputs at address 0x00
  ModbusRTUServer.configureDiscreteInputs(0x00, numDiscreteInputs);

  // configure holding registers at address 0x00
  ModbusRTUServer.configureHoldingRegisters(0x00, numHoldingRegisters);

  // configure input registers at address 0x00
  ModbusRTUServer.configureInputRegisters(0x00, numInputRegisters);
}
```

In the loop function, we'll poll for Modbus RTU requests:

```
void loop() {
  // poll for Modbus RTU requests
  ModbusRTUServer.poll();
```

We'll map the coils (what we are writing to with the PLC) into discrete inputs to be read by the PLC. Similarly, we'll map the holding registers (what we are writing to with the PLC) into input registers to be read by the PLC.

We'll also read the value of the first holding register into our LED_CONTROL variable:

```
// map the coil values to the discrete input values (what we write to the Arduino coils is echoed back into the discrete inputs on the PLC)
  for (int i = 0; i < numCoils; i++) {
    int coilValue = ModbusRTUServer.coilRead(i);
    ModbusRTUServer.discreteInputWrite(i, coilValue);
  }

  // map the holding register values to the input register values (what we write to the Arduino holding registers is echoed back into the input registers on the PLC)
  for (int i = 0; i < numHoldingRegisters; i++) {
    long holdingRegisterValue = ModbusRTUServer.holdingRegisterRead(i);
    long LED_CONTROL = ModbusRTUServer.holdingRegisterRead(0);
    ModbusRTUServer.inputRegisterWrite(i, holdingRegisterValue);
```

Finally, we'll control the LED built-in to the Arduino that is controlled by pin 13. Whenever the first holding register – what we read into the variable LED_CONTROL – is equal to 1, we'll turn on pin 13, which will turn on the LED. Any other value will turn pin 13 and the LED off:

```
  // control the LED attached to pin 13 based on the value of the first holding register coming from the PLC (Modbus RTU master)
    if (LED_CONTROL == 1) {
      digitalWrite(LED_PIN, HIGH);
     }
    else {
      digitalWrite(LED_PIN, LOW);
    }
  }
}
```

That's it! We can now transfer our program to the Arduino by clicking on the Upload button in the icon bar:

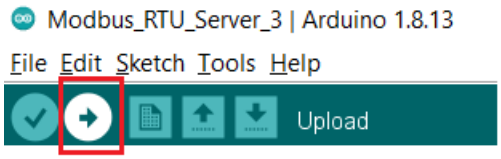


Figure 14. Transfer Program to the Arduino

5. Checking Connectivity and Testing Reading/Writing

The first thing we want to check is to make sure the PLC and the Arduino are communicating. To do this, we can monitor our error status registers that we set up for our four Modbus requests. We'll monitor these as HEX(W) values. For successful communication, we should see the slave number as the first two digits, with 00 in the last two digits. Any other numbers in the last two digits indicates that something isn't quite right.

We'll set up a custom monitor dialog box in the PLC program to check this:

Arduino Monitor - Overall Comms

Write | Close | Save

Device		Device Address	Monitor Type	Device Range	Current Value	Preset Value	Comment
D0100	...	D0100	HEX (W)	0	0100		Modbus Read Input Registers
D0101	...	D0101	HEX (W)	0	0100		Modbus Read Input Status
D0102	...	D0102	HEX (W)	0	0100		Modbus Preset Multiple Registers
D0103	...	D0103	HEX (W)	0	0100		Modbus Force Multiple Coils

Figure 15. Monitoring Communications for all Four Modbus Requests

With all four Modbus requests executing correctly, we can now create another custom monitor dialog box to see if what we are writing to the Arduino (in data registers D50-D59) is getting copied into the input registers in the Arduino and then read back into the PLC in data registers D0-D9.

Good news! The values that we write in D50-D59 are indeed showing up in D0-D9:

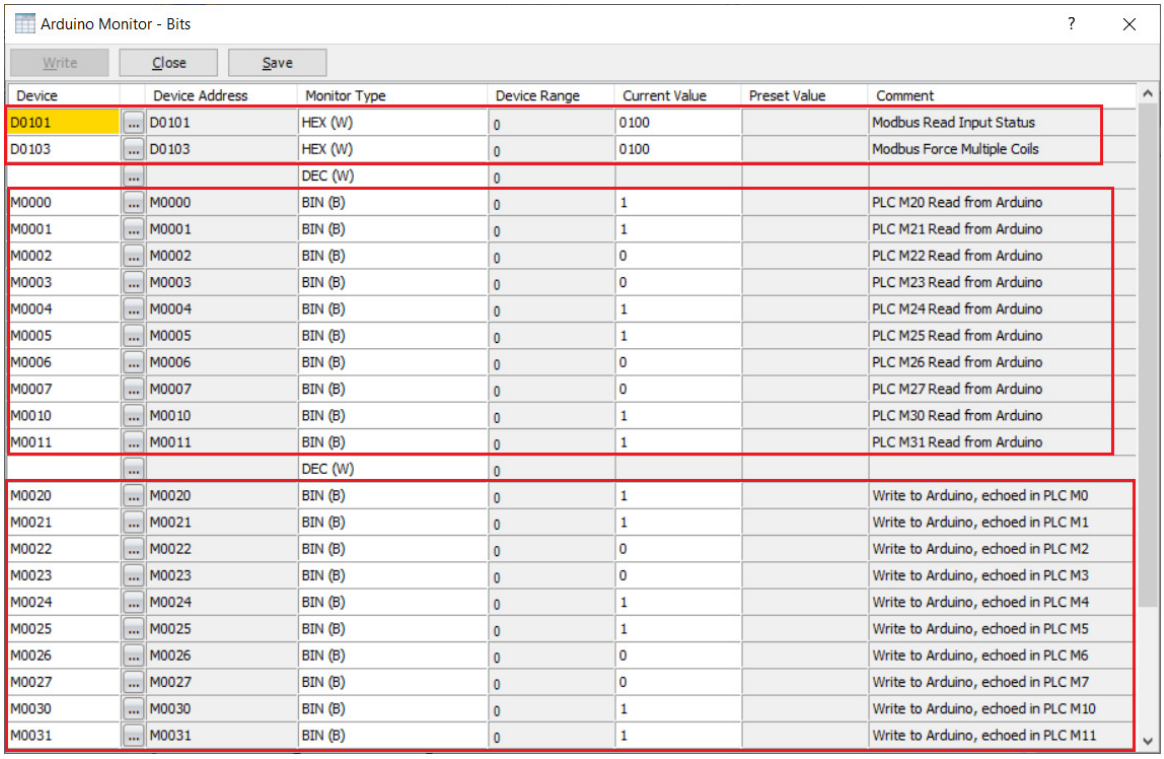
Arduino Monitor - Registers

Write | Close | Save

Device		Device Address	Monitor Type	Device Range	Current Value	Preset Value	Comment
D0100	...	D0100	HEX (W)	0	0100		Modbus Read Input Registers
D0102	...	D0102	HEX (W)	0	0100		Modbus Preset Multiple Registers
	...		DEC (W)	0			
D0000	...	D0000	DEC (W)	0	1		PLC D50 Read from Arduino
D0001	...	D0001	DEC (W)	0	0		PLC D51 Read from Arduino
D0002	...	D0002	DEC (W)	0	12		PLC D52 Read from Arduino
D0003	...	D0003	DEC (W)	0	755		PLC D53 Read from Arduino
D0004	...	D0004	DEC (W)	0	80		PLC D54 Read from Arduino
D0005	...	D0005	DEC (W)	0	19		PLC D55 Read from Arduino
D0006	...	D0006	DEC (W)	0	5		PLC D56 Read from Arduino
D0007	...	D0007	DEC (W)	0	1		PLC D57 Read from Arduino
D0008	...	D0008	DEC (W)	0	0		PLC D58 Read from Arduino
D0009	...	D0009	DEC (W)	0	1		PLC D59 Read from Arduino
	...		DEC (W)	0			
D0050	...	D0050	DEC (W)	0	1		Write to Arduino, echoed in PLC D0
D0051	...	D0051	DEC (W)	0	0		Write to Arduino, echoed in PLC D1
D0052	...	D0052	DEC (W)	0	12		Write to Arduino, echoed in PLC D2
D0053	...	D0053	DEC (W)	0	755		Write to Arduino, echoed in PLC D3
D0054	...	D0054	DEC (W)	0	80		Write to Arduino, echoed in PLC D4
D0055	...	D0055	DEC (W)	0	19		Write to Arduino, echoed in PLC D5
D0056	...	D0056	DEC (W)	0	5		Write to Arduino, echoed in PLC D6
D0057	...	D0057	DEC (W)	0	1		Write to Arduino, echoed in PLC D7
D0058	...	D0058	DEC (W)	0	0		Write to Arduino, echoed in PLC D8
D0059	...	D0059	DEC (W)	0	1		Write to Arduino, echoed in PLC D9

Figure 16. Reading and Writing Register Data

Now let's check to see if the bits we are writing into M20-M27 and M30-M31 are getting received by the Arduino and mapped into the discrete inputs (read back into the PLC in M0-M7 and M10-M11):



Device	Device Address	Monitor Type	Device Range	Current Value	Preset Value	Comment
D0101	D0101	HEX (W)	0	0100		Modbus Read Input Status
D0103	D0103	HEX (W)	0	0100		Modbus Force Multiple Coils
		DEC (W)	0			
M0000	M0000	BIN (B)	0	1		PLC M20 Read from Arduino
M0001	M0001	BIN (B)	0	1		PLC M21 Read from Arduino
M0002	M0002	BIN (B)	0	0		PLC M22 Read from Arduino
M0003	M0003	BIN (B)	0	0		PLC M23 Read from Arduino
M0004	M0004	BIN (B)	0	1		PLC M24 Read from Arduino
M0005	M0005	BIN (B)	0	1		PLC M25 Read from Arduino
M0006	M0006	BIN (B)	0	0		PLC M26 Read from Arduino
M0007	M0007	BIN (B)	0	0		PLC M27 Read from Arduino
M0010	M0010	BIN (B)	0	1		PLC M30 Read from Arduino
M0011	M0011	BIN (B)	0	1		PLC M31 Read from Arduino
		DEC (W)	0			
M0020	M0020	BIN (B)	0	1		Write to Arduino, echoed in PLC M0
M0021	M0021	BIN (B)	0	1		Write to Arduino, echoed in PLC M1
M0022	M0022	BIN (B)	0	0		Write to Arduino, echoed in PLC M2
M0023	M0023	BIN (B)	0	0		Write to Arduino, echoed in PLC M3
M0024	M0024	BIN (B)	0	1		Write to Arduino, echoed in PLC M4
M0025	M0025	BIN (B)	0	1		Write to Arduino, echoed in PLC M5
M0026	M0026	BIN (B)	0	0		Write to Arduino, echoed in PLC M6
M0027	M0027	BIN (B)	0	0		Write to Arduino, echoed in PLC M7
M0030	M0030	BIN (B)	0	1		Write to Arduino, echoed in PLC M10
M0031	M0031	BIN (B)	0	1		Write to Arduino, echoed in PLC M11

Figure 17. Reading and Writing Bit Data

This is working correctly as well.

The final item we need to check is to see if the LED on-board the Arduino that is controlled by pin 13 turns on when we write a value of 1 into D50 and turns off when we write any other value. A quick test of this shows the LED is working correctly, so all of the code that we wrote for the PLC and for the Arduino is functioning as desired!

6. Entire Arduino Program

In case you wanted to copy the Arduino code, it is included below in its entirety:

```
#include <ArduinoRS485.h>
#include <ArduinoModbus.h>

#define LED_PIN 13

const int numCoils = 10;
const int numDiscreteInputs = 10;
const int numHoldingRegisters = 10;
const int numInputRegisters = 10;
long LED_CONTROL = 0;

void setup() {

  pinMode(LED_PIN, OUTPUT);

  Serial.begin(9600);
  while (!Serial);
  Serial.println("Arduino as a Modbus RTU Server");

  // start the Modbus RTU server, with slave ID of 1
  if (!ModbusRTUServer.begin(1, 9600))
  {
    Serial.println("Failed to start Modbus RTU Server!");
    while (1);
  }

  // configure coils at address 0x00
  ModbusRTUServer.configureCoils(0x00, numCoils);

  // configure discrete inputs at address 0x00
  ModbusRTUServer.configureDiscreteInputs(0x00, numDiscreteInputs);

  // configure holding registers at address 0x00
  ModbusRTUServer.configureHoldingRegisters(0x00, numHoldingRegisters);

  // configure input registers at address 0x00
  ModbusRTUServer.configureInputRegisters(0x00, numInputRegisters);
}

void loop() {
  // poll for Modbus RTU requests
  ModbusRTUServer.poll();

  // map the coil values to the discrete input values (what we write to the Arduino coils is echoed back into the
discrete inputs on the PLC)
  for (int i = 0; i < numCoils; i++) {
    int coilValue = ModbusRTUServer.coilRead(i);
    ModbusRTUServer.discreteInputWrite(i, coilValue);
  }
```

```
  // map the holding register values to the input register values (what we write to the Arduino holding registers is
echoed back into the input registers on the PLC)
  for (int i = 0; i < numHoldingRegisters; i++) {
    long holdingRegisterValue = ModbusRTUServer.holdingRegisterRead(i);
    long LED_CONTROL = ModbusRTUServer.holdingRegisterRead(0);
    ModbusRTUServer.inputRegisterWrite(i, holdingRegisterValue);

  // control the LED attached to pin 13 based on the value of the first holding register coming from the PLC
(Modbus RTU master)
    if (LED_CONTROL == 1) {
      digitalWrite(LED_PIN, HIGH);
     }
    else {
      digitalWrite(LED_PIN, LOW);
    }
  }
}
```