Contents

1 ModbusSlave

1.0.0.0.1 ModbusSlave library for Arduino This Modbus RTU slave library uses callbacks to handle modbus requests for one or multiple slave ids. Handler functions are called on modbus a request, and the users can implement them within their sketch.

1.0.1 ModbusSlave is fun and easy to use

Register a handler function:

```
slave.cbVector[CB_READ_INPUT_REGISTERS] = ReadAnalogIn;
```

Implement it:

```
void ReadAnalogIn(uint8_t fc, uint16_t address, uint16_t length, void *callbackContext) {
    for (int i = 0; i < length; i++)
        slave.writeRegisterToBuffer(i, analogRead(address + i));
}
```

And thats it, your sketch is modbus enabled. (see the full examples for more detail)

- Install
- Compatibility
- Callback vector - Multiple Slaves - Slots - Handler function - Function codes - Reading and writing to the request buffer
- Examples - handle “Force Single Coil” as arduino digitalWrite - handle “Read Input Registers” as arduino analogRead

---

1.0.2 Install

Download the zip package, and install it into your Arduino IDE. See the Arduino tutorial about installing 3rd party libraries: https://www.arduino.cc/en/Guide/Libraries#toc4

1.0.3 Compatibility

This class implements:

- FC1 “Read Coil Status”
- FC2 “Read Input Status”

- FC3 “Read Holding Registers”
- FC4 “Read Input Registers”
- FC5 “Force Single Coil”
- FC6 “Preset Single Register”
- FC15 “Force Multiple Coils”
- FC16 “Preset Multiple Registers”

1.0.4 Serial port

- The default serial port is Serial, but any class that inherits from the Stream class can be used. To set a different Serial class, explicitly pass the Stream in the Modbus class constuctor.

1.0.5 Callback vector

Users register handler functions into the callback vector of the slave.

Multiple Slaves

This can be done independently for one or multiple slaves with different IDs.

Slots

The callback vector has 7 slots for request handlers:

- slave.cbVector[CB_READ_COILS] - called on FC1
- slave.cbVector[CB_READ_DISCRETE_INPUTS] - called on FC2
- slave.cbVector[CB_READ_HOLDING_REGISTERS] - called on FC3
- slave.cbVector[CB_READ_INPUT_REGISTERS] - called on FC4
- slave.cbVector[CB_WRITE_COILS] - called on FC5 and FC15
- slave.cbVector[CB_WRITE_HOLDING_REGISTERS] - called on FC6 and FC16
- slave.cbVector[CB_READ_EXCEPTION_STATUS] - called on FC7

Handler function

A handler functions must return an uint8_t code and take the following as parameters:

- uint8_t fc - request function code
- uint16_t address - first register / first coil address
- uint16_t length - length of data
- void *callbackContext - callback context

Usable return codes:

- STATUS_OK = 0,
- STATUS_ILLEGAL_FUNCTION,
- STATUS_ILLEGAL_DATA_ADDRESS,
- STATUS_ILLEGAL_DATA_VALUE,
- STATUS_SLAVE_DEVICE_FAILURE,
- STATUS_ACKNOWLEDGE,
- STATUS_SLAVE_DEVICE_BUSY,

- STATUS_NEGATIVE_ACKNOWLEDGE,
- STATUS_MEMORY_PARITY_ERROR,
- STATUS_GATEWAY_PATH_UNAVAILABLE,
- STATUS_GATEWAY_TARGET_DEVICE_FAILED_TO_RESPOND

Function codes

- FC_READ_COILS = 1
- FC_READ_DISCRETE_INPUT = 2
- FC_READ_REGISTERS = 3
- FC_READ_INPUT_REGISTERS = 4
- FC_WRITE_COIL = 5
- FC_WRITE_REGISTER = 6
- FC_READ_EXCEPTION_STATUS = 7
- FC_WRITE_MULTIPLE_COILS = 15
- FC_WRITE_MULTIPLE_REGISTERS = 16

Reading and writing to the request / response buffer

- bool readCoilFromBuffer(int offset) : read one coil value from the request buffer.
- uint16_t readRegisterFromBuffer(int offset) : read one register value from the request buffer.
- uint8_t writeExceptionStatusToBuffer(int offset, bool status) : write an exception status into the response buffer.
- uint8_t writeCoilToBuffer(int offset, int state) : write one coil state into the response buffer.
- uint8_t writeDiscreteInputToBuffer(int offset, bool state) : write one discrete input value into the response buffer.
- uint8_t writeRegisterToBuffer(int offset, uint16_t value) : write one register value into the response buffer.
- uint8_t writeArrayToBuffer(int offset, uint16_t *str, uint8_t length); : writes an array of data into the response register.

1.0.6 Examples

Handle “Force Single Coil” and write the received value to digitalWrite()

```
#include <ModbusSlave.h>

// Implicitly set stream to use the Serial serialport.
Modbus slave(1, 8); // [stream = Serial,] slave id = 1, rs485 control-pin = 8

void setup() {
    // Register functions to call when a certain function code is received.
    // If there is no handler assigned to the function code a valid but empty message will be replied.
    slave.cbVector[CB_WRITE_COILS] = writeDigitalOut;

    // Start the slave at a baudrate of 9600bps on the Serial port.
    Serial.begin(9600);
```

```
    slave.begin(9600);
}

void loop() {
    // Listen for modbus requests on the serial port.
    // When a request is received it's going to get validated.
    // And if there is a function registered to the received function code, this function will be executed.
    slave.poll();
}

// Handel Force Single Coil (FC=05).
uint8_t writeDigitalOut(uint8_t fc, uint16_t address, uint16_t length, void *callbackContext) {
    if (slave.readCoilFromBuffer(0) == HIGH)
    {
        digitalWrite(address, HIGH);
    }
    else
    {
        digitalWrite(address, LOW);
    }
    return STATUS_OK;
}
```

Handle “Read Input Registers” and return analogRead()

```
#include <ModbusSlave.h>

// Explicitly set a stream to use the Serial port.
Modbus slave(Serial, 1, 8); // stream = Serial, slave id = 1, rs485 control-pin = 8

void setup() {
    // Register functions to call when a certain function code is received.
    // If there is no handler assigned to the function code a valid but empty message will be replied.
    slave.cbVector[CB_WRITE_COILS] = readAnalogIn;

    // Start the slave at a baudrate of 9600bps on the Serial port.
    Serial.begin(9600);
    slave.begin(9600);
}

void loop() {
    // Listen for modbus requests on the serial port.
    // When a request is received it's going to get validated.
    // And if there is a function registered to the received function code, this function will be executed.
```

```
    slave.poll();
}

// Handle Read Input Registers (FC=04).
uint8_t readAnalogIn(uint8_t fc, uint16_t address, uint16_t length, void *callbackContext) {
    // Write the result of analogRead() into the response buffer.
    for (int i = 0; i < length; i++) {
      slave.writeRegisterToBuffer(i, analogRead(address + i));
    }
    return STATUS_OK;
}
```