mcu-uuid-modbus

Simon Arlott

Apr 09, 2024

CONTENTS

CHAPTER

ONE

DESCRIPTION

Microcontroller asynchronous Modbus library

CHAPTER

TWO

PURPOSE

Provides an asynchronous client for communicating with Modbus devices. This library is for single threaded applications and cannot be used from an interrupt context.

CHAPTER

THREE

DEPENDENCIES

- mcu-uuid-common
- mcu-uuid-log

Refer to the `library.json` file for more details.

CHAPTER

FOUR

CONTENTS

4.1 Usage

```
#include <uuid/modbus.h>
```

Create a `uuid::modbus::SerialClient` with a configured `HardwareSerial` device and call `loop()` on the instance regularly.

Use the functions for reading/writing registers to initiate a request and then call `done()` on the returned response object to check for completion. Progress can only be made when the `loop()` function is called.

Call `success()` to find out if communication was successful and then read response data using `data()`.

4.1.1 Example

```
#include <Arduino.h>
#include <uuid/common.h>
#include <uuid/log.h>
#include <uuid/modbus.h>

static uuid::log::PrintHandler log_handler{Serial};
static uuid::modbus::SerialClient client{Serial1};

void setup() {
        Serial.begin(115200);
        Serial1.begin(19200, SERIAL_8E1);
        uuid::log::Logger::register_handler(&log_handler, uuid::log::Level::ALL);
}

void loop() {
        static uuid::log::Logger logger{F("example")};
        static std::shared_ptr<const uuid::modbus::RegisterDataResponse> response;
        static uint16_t address = 0x00A0;

        uuid::loop();
        client.loop();

        if (!response) {
                if (address < 0x00C0) {
                        logger.info(F("Reading from device at address %04X"), address);
```

(continues on next page)

(continued from previous page)

```
                        response = client.read_input_registers(123, address, 4);
                        address += 4;
                }
        } else if (response->done()) {
                if (response->success()) {
                        if (response->data().size() == 4) {
                                logger.info(F("Data: %04X %04X %04X %04X"),
                                        response->data()[0],
                                        response->data()[1],
                                        response->data()[2],
                                        response->data()[3]);
                        } else {
                                logger.err(F("Invalid number of registers in response: %u
↪"),
                                        response->data().size());
                        }
                } else {
                        logger.err(F("Failed"));
                }

                response.reset();
                log_handler.loop();
                Serial.println();
        }

        log_handler.loop();
}
```

Output

```
000+00:00:00.000 I [example] Reading from device at address 00A0
000+00:00:00.001 T [modbus] -> 7B 04'00 A0 00 04'FA 71
000+00:00:00.077 T [modbus] <- 7B 04'08 12 34 56 78 90 AB CD EF'BF C2
000+00:00:00.077 I [example] Data: 1234 5678 90AB CDEF

000+00:00:00.078 I [example] Reading from device at address 00A4
000+00:00:00.079 T [modbus] -> 7B 04'00 A4 00 04'BB B0
000+00:00:00.155 T [modbus] <- 7B 04'08 23 45 67 89 0A BC DE F1'76 0D
000+00:00:00.155 I [example] Data: 2345 6789 0ABC DEF1

000+00:00:00.156 I [example] Reading from device at address 00A8
000+00:00:00.157 T [modbus] -> 7B 04'00 A8 00 04'7B B3
000+00:00:00.233 T [modbus] <- 7B 04'08 34 56 78 90 AB CD EF 12'2C 75
000+00:00:00.233 I [example] Data: 3456 7890 ABCD EF12

000+00:00:00.234 I [example] Reading from device at address 00AC
000+00:00:00.235 T [modbus] -> 7B 04'00 AC 00 04'3A 72
000+00:00:00.311 T [modbus] <- 7B 04'08 45 67 89 0A BC DE F1 23'EE FF
000+00:00:00.311 E [modbus] Received frame with invalid CRC FFEE from device 123 with␣
↪function 04, expected 66BB
```

(continues on next page)

(continued from previous page)

```
000+00:00:00.311 E [example] Failed

000+00:00:00.312 I [example] Reading from device at address 00B0
000+00:00:00.313 T [modbus] -> 7B 04'00 B0 00 04'FB B4
000+00:00:00.381 T [modbus] <- 7B 84'02'E3 18
000+00:00:00.381 N [modbus] Exception code 02 for function 04 from device 123
000+00:00:00.381 E [example] Failed

000+00:00:00.382 I [example] Reading from device at address 00B4
000+00:00:00.383 T [modbus] -> 7B 04'00 B4 00 04'BA 75
000+00:00:00.457 T [modbus] <- 7B 04'06 56 78 9A BC DE F0'61 15
000+00:00:00.457 E [example] Invalid number of registers in response: 3

000+00:00:00.458 I [example] Reading from device at address 00B8
000+00:00:00.459 T [modbus] -> 7B 04'00 B8 00 04'7A 76
000+00:00:00.537 T [modbus] <- 7B 04'08 67 89 AB CD EF 01 23 45 67 89'49 92
000+00:00:00.537 E [modbus] Length mismatch for function 04 from device 123, expected 11␣
↪received 13
000+00:00:00.537 E [example] Failed

000+00:00:00.538 I [example] Reading from device at address 00BC
000+00:00:00.539 T [modbus] -> 7B 04'00 BC 00 04'3B B7
000+00:00:10.547 N [modbus] Timeout waiting for response to function 04 from device 123
000+00:00:10.547 E [example] Failed
```

CHAPTER

FIVE

RESOURCES

5.1 Change log

5.1.1 Unreleased

Changed

- Use `PSTR_ALIGN` for flash strings.

5.1.2 0.2.0 – 2022-02-10

Support configuring the default timeout for requests.

Added

- Configuration of the default timeout for requests.

5.1.3 0.1.1 – 2022-02-06

Fix for register reads.

Fixed

- Handling of responses to the following Modbus functions:
  - Read Holding Registers
  - Read Input Registers

5.1.4 0.1.0 – 2022-01-30

First stable release.

Added

- Serial device client support for the following Modbus functions:
  - Read Holding Registers
  - Read Input Registers
  - Write Single Register
  - Read Exception Status