

Modbus Reading for Schneider PowerLogic PM8000

This is a specification and implementation of the Arduino-based Modbus data logger for the Schneider PowerLogic PM8000. This software is designed for Vivarox EMS and only Vivarox has right to use and modify this software.

Arduino Implementation:

This project uses an Arduino to connect to Modbus devices, read information, and log it onto an SD card with timestamps.

Hardware needed:

1. Arduino Board Recommended: Arduino MEGA 2560 (for more memory and I/O pins) or Arduino UNO (for simpler projects).
 - [Arduino MEGA @ R377.20](#)
 - [UNO R3 with 16U2 USB Interface @ R151.00](#)
2. RS485 to TTL Module Allows communication between the Arduino and Modbus devices using the RS485 protocol.
 - [RS485 Module \(TTL -> RS485\) @ R25.30](#)
 - [MAX485 Bus Transceiver \(4 Pack\) @ R16.00](#)
3. SD Card Module Allows the Arduino to read from and write data to an SD card.
 - [Micro SD Card Module @ R25.00](#)
4. RTC Module To keep track of the current date and time, even when the Arduino is powered off.
 - [DS3231 Real Time Clock Module @ R55.20](#)
5. Power Supply To power the Arduino and connected peripherals.
 - [AC Adapter 9V with barrel jack @ R60](#)
6. LED Indicators Two LEDs for status indication (not included in original cost estimate).

Wiring

RS485 Module to Arduino:

1. RO (Receiver Output) to Arduino RX (pin 8)
2. DI (Driver Input) to Arduino TX (pin 7)
3. DE (Driver Enable) & RE (Receiver Enable) to Arduino digital pin 4
4. VCC to 5V on Arduino
5. GND to GND on Arduino
6. A & B (RS485 differential pair) to Modbus device

SD Card Module to Arduino:

1. VCC to 5V on Arduino
2. GND to GND on Arduino
3. MOSI to MOSI (pin 51 on MEGA, pin 11 on UNO)
4. MISO to MISO (pin 50 on MEGA, pin 12 on UNO)
5. SCK to SCK (pin 52 on MEGA, pin 13 on UNO)
6. CS (Chip Select) to digital pin 10

RTC Module to Arduino:

1. VCC to 5V on the Arduino
2. GND to GND on the Arduino
3. SDA to SDA (pin 20 on MEGA, pin A4 on UNO)
4. SCL to SCL (pin 21 on MEGA, pin A5 on UNO)

LED Indicators:

1. LED A to digital pin 3
2. LED B to digital pin 5

Software

- Modbus Library: ModbusMaster
- SD Library: SdFat (more advanced than the standard SD library)
- RTC Library: RTCLib by Adafruit
- NeoSWSerial: For better latency on software serial communication

Implementation Details

1. Modbus Configuration:
 - Slave ID: 101
 - Baud Rate: 9600
 - Register map: Defined in separate “register_map_pm8000.h” file
2. Data Logging:
 - Frequency: Readings taken every second
 - File Format: CSV (Comma-Separated Values)
 - Filename: “pm8k_YYYYMMDD.csv” (generated daily based on current date)
 - Data Structure: Timestamp, followed by register values
 - Header Row: Includes register addresses for easy identification
3. Register Types Supported:
 - Float (32-bit)
 - Integer (32-bit)
 - Long (64-bit)
 - String (up to 20 characters)
4. Error Handling and Status Indication:
 - LED A: Indicates successful data writing and transmission
 - LED B: Indicates errors (e.g., SD card issues, RTC problems, Modbus communication errors)
 - Serial output for debugging (9600 baud)
5. Special Features:
 - Automatic creation of new log file on date change
 - Header row written only once per file
 - Robust error handling for SD card, RTC, and Modbus communication

Programming Workflow

1. Initialize hardware (RTC, SD card, RS485 module)
2. Set up Modbus communication parameters
3. Enter main loop:
 - Read current time from RTC
 - Read data from Modbus registers
 - Write timestamped data to SD card
 - Handle any errors and provide status indication via LEDs
 - Delay for 1 second before next reading

Costs

Estimated cost of the hardware from suppliers like Micro Robotics, excluding labor to assemble:

- R617.00 per unit using the Arduino MEGA
- R374.50 per unit using the Arduino UNO

Note: These costs do not include the additional LEDs for status indication.

Additional Notes

- The system is designed to reset and write logs to newly inserted SD cards automatically.
- Error handling includes visual feedback via LED indicators and detailed serial output for debugging.
- The modular design allows for easy expansion of register types and Modbus devices.

For more detailed implementation, refer to the [Source Code](#).

Memory Limitations and Register Customization

Memory Constraints

The Arduino, particularly models like the UNO and MEGA, has limited memory available for storing program code and variables. This limitation affects the number of Modbus registers that can be defined and read in a single project.

- Arduino UNO: 32 KB Flash (program storage), 2 KB SRAM
- Arduino MEGA: 256 KB Flash, 8 KB SRAM

Due to these constraints, the number of registers that can be defined in the `register_map_pm8000.h` file is not unlimited. The exact number will depend on the complexity of your code and other libraries used.

Customizing the Register Map

To adapt this project to your specific needs, you can modify the `register_map_pm8000.h` file. This file contains the definitions of Modbus registers to be read by the Arduino.

To customize the register map:

1. Open the `register_map_pm8000.h` file in your Arduino IDE or text editor.
2. Locate the `registers` array in the file. It should look something like this:

```
const RegisterInfo registers[] PROGMEM = {
  {40001, 2}, // Example register
  {40003, 1},
  // ... other registers ...
};
```

1. To remove a register, simply comment out its line by adding `//` at the beginning:

```
const RegisterInfo registers[] PROGMEM = {
  {40001, 2}, // Example register
  // {40003, 1}, // This register is now commented out and won't be read
  // ... other registers ...
};
```

1. To add a new register, add a new line to the array with the register address and type:

```
const RegisterInfo registers[] PROGMEM = {
  {40001, 2}, // Example register
  {40003, 1},
  {40005, 2}, // New register added
  // ... other registers ...
};
```

1. Remember to keep the array syntax correct, with commas between entries and a semicolon at the end of the array.

Best Practices

- Start by commenting out registers you don't need before adding new ones.
- If you're using an Arduino UNO, you may need to be more selective about which registers to include due to memory constraints.
- Test your modifications incrementally to ensure the Arduino can handle the memory load.
- If you need to read a large number of registers, consider using an Arduino MEGA or a more powerful microcontroller.

By carefully managing the registers in the `register_map_pm8000.h` file, you can customize this Modbus reader to suit your specific requirements while staying within the memory limitations of your Arduino board.