# Modbus Reading and GSM Data Logging for Schneider PowerLogic PM8000

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

## Arduino Implementation:

This project uses an Arduino to connect to Modbus devices, read information, and transmit it via GSM to a remote server.

### Hardware needed:

1. Arduino Board Recommended: Arduino MEGA 2560 (for more memory and I/O pins) or Arduino UNO (for simpler projects).
2. RS485 to TTL Module Allows communication between the Arduino and Modbus devices using the RS485 protocol.
3. 800C GSM Module 3.3/5V Enables the Arduino to send data over cellular networks.
4. Power Supply To power the Arduino and connected peripherals. A dedicated 2A power supply is required for the GSM module.
5. LED Indicators Two LEDs for status indication.
6. Capacitors 100uF and 10uF capacitors for power supply stabilization.

## Wiring

#### Wiring Diagram

```
Arduino Mega/Uno  800C GSM Module     Description
----------------  ---------------      -----------
      5V      ---->    VCC            Power supply
      GND     ---->    GND            Ground
      2 (RX)  <----    TX             GSM TX to Arduino RX
      3 (TX)  ---->    RX             Arduino TX to GSM RX

      7       ---->    DI (RS485)     RS485 Driver Input
      8       <----    RO (RS485)     RS485 Receiver Output
      4       ---->    DE/RE (RS485)  RS485 Driver Enable/Receiver Enable

      3       ---->    LED A          Status LED A
      5       ---->    LED B          Status LED B

Power Supply
------------
      VCC     ---->    + (2A)         Dedicated 2A power supply positive
      GND     ---->    - (GND)        Dedicated 2A power supply ground

Capacitors
----------
    VCC ----|(---- GND               100uF capacitor
    VCC ---||---- GND                10uF capacitor

                   Antenna
                      |
                      |
              [GSM Module 800C]
```

#### Wiring Notes:

1. Ensure the power supply can provide 2A current.
2. Place the 100uF and 10uF capacitors as close to the GSM module’s power pins as possible.
3. The RS485 connections are optional and depend on your specific requirements.
4. LEDs should be connected with appropriate current-limiting resistors (not shown in diagram).
5. The GSM module’s antenna should be connected securely.
6. Double-check all connections before powering on the system.

## Software

- Modbus Library: ModbusMaster

- GSM Library: Built-in SoftwareSerial
- NeoSWSerial: For better latency on software serial communication with Modbus

### Implementation Details

1. Modbus Configuration:
    - Slave ID: 101
    - Baud Rate: 9600
    - Register map: Defined in separate "register_map_pm8000.h" file
2. Data Logging and Transmission:
    - Frequency: Readings taken and transmitted every minute
    - Data Format: CSV (Comma-Separated Values) string
    - Data Structure: Timestamp (millis), followed by register values
    - Header Row: Includes register addresses for easy identification
3. Register Types Supported:
    - Float (32-bit)
    - Integer (32-bit)
    - String (up to 20 characters)
4. Error Handling and Status Indication:
    - LED A: Indicates successful data transmission
    - LED B: Indicates errors (e.g., GSM issues, Modbus communication errors)
    - Serial output for debugging (9600 baud)
5. Special Features:
    - Robust error handling for GSM and Modbus communication
    - Header sent once at the beginning of each session
    - Configurable APN and server URL

### Programming Workflow

1. Initialize hardware (GSM module, RS485 module)
2. Set up Modbus communication parameters
3. Enter main loop:
    - Read data from Modbus registers
    - Format data into CSV string
    - Send data via GSM to server
    - Handle any errors and provide status indication via LEDs
    - Delay for 1 minute before next reading and transmission

## 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.
- Ensure your GSM data plan can handle the amount of data being transmitted.
- Regularly check your server to ensure data is being received correctly.

## Important AT Commands

Here are some important AT commands used in this project:

1. `AT+XISP=0`: Set to use internal TCP/IP protocol stack.
2. `AT+CGDCONT=1,"IP","APN_NAME"`: Set the APN for your cellular provider.
3. `AT+XGAUTH=1,1,"username","password"`: Set authentication for private networks if needed.
4. `AT+XIIC=1`: Establish PPP link.
5. `AT+TCPSETUP=0,server_ip,port`: Establish TCP connection.
6. `AT+TCPSEND=0,data_length`: Send data over TCP connection.
7. `AT+TCPCLOSE=0`: Close TCP connection.
8. `AT+ENPWRSAVE=1`: Enable power-saving mode (if needed).

Remember to replace "APN_NAME", "username", "password", "server_ip", and "port" with your specific values.

## Troubleshooting

1. If you encounter communication issues, double-check your wiring and ensure all connections are secure.
2. Verify that your APN settings are correct for your cellular provider.
3. If the module isn't responding, try resetting it and check your power supply.
4. Use AT commands like `AT+CSQ` to check signal strength and `AT+CREG?` to check network registration status.
5. If data isn't being sent, verify your TCP connection settings and ensure you have an active data plan.

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