SM Energy FE102M Energy Monitor

Technical User Manual

Version 1.0

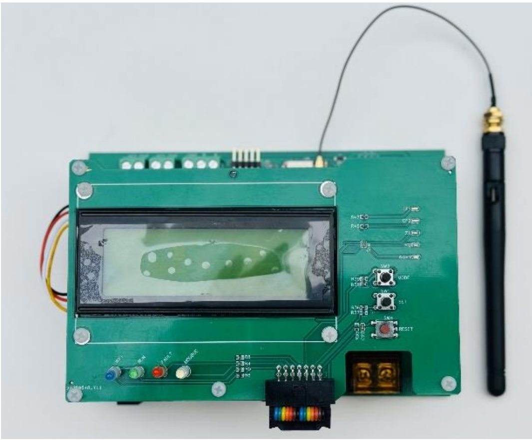

Table of Contents

---

1. Introduction

Thank you for purchasing the SM Energy FE102M Energy Monitor. This device is designed to provide accurate energy measurement capabilities for a wide range of applications, from home energy monitoring to industrial power management.

This technical manual provides comprehensive information on hardware installation, software configuration, calibration procedures, and programming examples to help you get the most out of your Energy Monitor.

---

2. Product Overview

The SM Energy FE102M Energy Monitor is built around the ESP32 WROOM 32 microcontroller and ATM90E26 energy monitoring IC. It offers high-precision measurement of electrical parameters including voltage, current, power (active, reactive, and apparent), energy consumption, power factor, and frequency.

Key Features:

- **High Accuracy**: ±0.1% for active energy and ±0.2% for reactive energy
- **Wide Dynamic Range**: 6000:1 measurement capability
- **Multiple Communication Protocols**: WiFi, Bluetooth, and MODBUS (RS485)
- **Current Measurement**: Compatible with various CT clamps
- **Environmental Sensing**: Onboard DHT22 temperature and humidity sensor
- **Display Capabilities**: I2C 20x4 LCD module interface
- **Programmable Functionality**: Arduino ESP32 compatible
- **Flexible Integration**: Energy pulse outputs, status LEDs, and user buttons

The device is fully compliant with IEC62052-11, IEC62053-21, and IEC62053-23 standards, ensuring reliable and accurate energy measurements.

3. Technical Specifications

Electrical Characteristics

Parameter	Specification
Input Voltage	AC Low Voltage (< 250V)
Power Supply	9-12V DC
Current Measurement	Compatible with various CT clamps (see Current Transformer Selection)
Measurement Accuracy	±0.1% for active energy, ±0.2% for reactive energy
Parameter Accuracy	Less than ±0.5% fiducial error for Vrms, Irms, power, frequency, power factor, phase angle
Temperature Coefficient	6 ppm/℃ (typical) for on-chip reference voltage

Connectivity

Feature	Specification
WiFi	Built-in ESP32 WiFi capability
Bluetooth	Built-in ESP32 Bluetooth capability
MODBUS	RS485 interface for industrial applications
Serial	USB interface for programming and debugging

Physical Characteristics

Parameter	Specification
Dimensions	150mm × 105mm × 35mm
Mounting	Multiple mounting holes for panel or enclosure mounting
Connectors	Screw terminals for power and CT connections

Environmental Conditions

Parameter	Specification
Operating Temperature	-10°C to +50°C
Storage Temperature	-20°C to +60°C
Humidity	10% to 90% RH, non-condensing

4. Hardware Installation

Safety Precautions

WARNING: Working with electrical systems can be dangerous. Always disconnect power before installation and ensure all connections are secure before restoring power.

- Always use appropriate safety equipment when working with electrical installations • Ensure the device is properly grounded
- Do not exceed the maximum rated voltage (250V AC)
- For current measurement, only use properly rated CT clamps

Mounting Instructions

1. Select a suitable location for mounting the Energy Monitor that provides: • Easy access for wiring and viewing the display

- Protection from dust and moisture
- Adequate ventilation to prevent overheating

2. Mount the device using the provided mounting holes.

Wiring Connections

Power Supply

- Connect a 9-12V DC power supply to the DC input terminals. • Polarity must be observed (marked as + and - on the PCB).

Voltage Measurement

- Connect the AC line voltage (< 250V) to the Line Input terminals. • For single-phase measurement, connect Line and Neutral.

Current Measurement

1. Identify the appropriate CT clamp for your application (see Current Transformer Selection).
2. For voltage output type CT sensors (SCT-013-030, SCT-013-050): • Ensure JP1 and JP2 jumpers are OPEN.
3. For current output type CT sensors:

- Ensure JP1 and JP2 jumpers are SHORTED.

4. Install the CT clamp around the conductor to be measured.

- Ensure the CT is installed in the correct orientation (arrow pointing toward the load).

5. Connect the CT clamp to the appropriate CT input terminals on the Energy Monitor.

Communication Connections

- For USB programming and debugging, connect a USB cable to the USB connector. • For MODBUS communication, connect to the MODBUS connector (GND, B, A).
- WiFi and Bluetooth are provided by the built-in ESP32 module.

5. Software Configuration

Setting Up the Arduino IDE

1. Install the Arduino IDE from arduino.cc.

2. Add ESP32 board support:

- Open Arduino IDE
- Go to File > Preferences
- Add `https://dl.espressif.com/dl/package_esp32_index.json`

to the "Additional Boards Manager URLs" field

- Go to Tools > Board > Boards Manager
- Search for "ESP32" and install the package

3. Select the correct board:

- Go to Tools > Board > ESP32 Arduino
- Select "ESP32 Dev Module" or "WEMOS D1 MINI ESP32" (or similar)
- Set Flash Frequency to 80MHz
- Set Upload Speed to 921600 for faster uploaded
- Set CPU Frequency to 240MHz

Required Libraries

Install the following libraries through the Arduino Library Manager (Tools > Manage Libraries):

1. SPI.h - Standard Arduino SPI library (included with ESP32 board package)
2. Wire.h - Standard Arduino I2C library (included with ESP32 board package)
3. DHT.h - Adafruit DHT sensor library
4. LiquidCrystal_I2C.h - For I2C LCD displays
5. WiFi.h - For WiFi connectivity (included with ESP32 board package)
6. ModbusMaster.h - For MODBUS communication

Initial Connection

1. Connect the FE102M Energy Monitor to your computer using a USB cable.
2. Select the correct COM port in the Arduino IDE (Tools > Port).
3. Upload a test sketch to verify communication.

Note: The Energy Monitor requires an external 12V DC power supply for full functionality, even when connected via USB.

---

6. ESP32 Pin Configuration

The SM Energy FE102M Energy Monitor utilizes the following ESP32 pin assignments:

ESP32 Pin No	Pin Name	Pin Function
4	(SENSOR_VP) GPIO36	SET input Switch
5	(SENSOR_VN) GPIO39	ATM90E26 IRQ Interrupt
6	GPIO34	ATM90E26 WarnOut: Fatal Error Warning Interrupt
7	GPIO35	ATM90E26 CF1: Active Energy Pulse Output Interrupt
8	GPIO32	ATM90E26 ZX: Voltage Zero-Crossing Output Interrupt
9	GPIO33	MODE input Switch
10	GPIO25	ATM90E26 CF2: Reactive Energy Pulse Output Interrupt
11	GPIO26	ATM90E26 UART RX pin (Should be selected from SW5 Switch to UART Mode)
12	GPIO27	ATM90E26 UART TX pin (Should be selected from SW5 Switch to UART Mode)
13	GPIO14	WiFi LED
14	GPIO12	RUN LED
16	GPIO13	MAX485 TXRX Control pin for MODBUS (RS485)
23	GPIO15	MODBUS LED
24	GPIO2	FAULT LED
26	GPIO4	DHT22 (AM2302) Sensor Data pin
27	GPIO16	MAX485 RO pin (RXD)
28	GPIO17	MAX485 DI pin (TXD)
29	GPIO5	ATM90E26 SPI Chip Select Pin
30	GPIO18	ATM90E26 SPI SCLK
31	GPIO19	ATM90E26 SPI MISO
33	GPIO21	I2C SDA
34	RXD0	Debug/Programming (USB) RX
35	TXD0	Debug/Programming (USB) TX
36	GPIO22	I2C SCL
37	GPIO23	ATM90E26 SPI MOSI

Front Panel Controls and Indicators

Component	Function
MODE Switch	User-programmable input for changing operating modes
SET Switch	User-programmable input for settings adjustment
RESET Switch	Resets both the Energy Meter and ESP32
WiFi Status LED	Indicates WiFi connection status
Meter Run Mode LED	Indicates the meter is in run mode
Meter Fault Status LED	Indicates a fault condition
MODBUS Status LED	Indicates MODBUS communication activity
ATM90E26 Energy Interrupt Status LEDs	Display status of various energy monitoring interrupts

7. ATM90E26 Energy Monitor IC

The ATM90E26 is a high-precision energy monitoring IC that forms the heart of the FE102M Energy Monitor. It provides accurate measurements of various electrical parameters.

Key Features of the ATM90E26

- Single-phase active and reactive energy metering
- Measurement of voltage, current, power, frequency, and power factor
- High accuracy (±0.1% for active energy, ±0.2% for reactive energy)
- Dynamic range of 6000:1
- Temperature compensation (6 ppm/℃ typical)
- Energy pulse output for active and reactive energy
- Voltage sag detection
- Zero-crossing detection
- SPI interface for communication

Register Map

ATM90E26 is controlled and read through a series of registers. The most important registers are:

System Control Registers

Register	Address	Description
SoftReset	0x00	Software Reset
SysStatus	0x01	System Status
FuncEn	0x02	Function Enable
SagTh	0x03	Voltage Sag Threshold
MMode	0x04	Metering Mode Configuration

Calibration Registers

Register	Address	Description
UgainA	0x08	Voltage RMS Gain
IgainA	0x09	Current RMS Gain
UoffsetA	0x0A	Voltage Offset
IoffsetA	0x0B	Current Offset
PoffsetA	0x0C	Power Offset
PPosA	0x0D	Power Positive Calibration
PNegA	0x0E	Power Negative Calibration
QoffsetA	0x0F	Reactive Power Offset
QPosA	0x10	Reactive Power Positive Calibration
QNegA	0x11	Reactive Power Negative Calibration

Measurement Registers

Register	Address	Description
UmeansA	0x49	Voltage RMS
ImeansA	0x48	Current RMS
PmeansA	0x4A	Active Power
QmeansA	0x4B	Reactive Power
SmeansA	0x4C	Apparent Power
PFmeansA	0x4D	Power Factor
PAnglesA	0x4E	Phase Angle
Freq	0x4F	Frequency

Energy Registers

Register	Address	Description
APenergyA	0x40	Active Positive Energy
ANenergyA	0x41	Active Negative Energy
RPenergyA	0x42	Reactive Positive Energy
RNenergyA	0x43	Reactive Negative Energy
SenergyA	0x45	Apparent Energy

Power Modes

The ATM90E26 has four power modes:

PM1 Value	Power Mode	Description
11	Normal Mode	Full functionality
10	Partial Measurement (M mode)	Reduced power consumption with limited functionality
01	Detection (D mode)	Low power, only detects presence of signals
00	Idle (I mode)	Minimal power consumption, most functions disabled

8. Current Transformer Selection

The FE102M Energy Monitor is compatible with a variety of current transformers (CT clamps). Select the appropriate CT based on your application's current range requirements.

Compatible Current Transformers

Model	Specification	Jumper Configuration
SCT-006	20A/25mA	JP1 and JP2 shorted
SCT-013-030	30A/1V	JP1 and JP2 open
SCT-013-050	50A/1V	JP1 and JP2 open
SCT-010	80A/26.6mA	JP1 and JP2 shorted
SCT-013-000	100A/50mA	JP1 and JP2 shorted
SCT-016	120A/40mA	JP1 and JP2 shorted
SCT-024	200A/100mA	JP1 and JP2 shorted
SCT-024	200A/50mA	JP1 and JP2 shorted

CT Types and Jumper Settings

The FE102M Energy Monitor supports two types of CT sensors:

1. **Current Output Type**: These CTs output a current proportional to the measured current.

- Examples: SCT-006, SCT-010, SCT-013-000, SCT-016, SCT-024
- **Jumper Configuration**: JP1 and JP2 should be SHORTED

2. **Voltage Output Type**: These CTs include a burden resistor and output a voltage proportional to the measured current.

- Examples: SCT-013-030, SCT-013-050
- **Jumper Configuration**: JP1 and JP2 should be OPEN

Important Note: Incorrect jumper configuration can result in inaccurate measurements or damage to the device.

CT Installation Guidelines

- Install the CT around a single conductor (Line or Neutral, but not both) • Ensure the CT is oriented correctly (arrow pointing toward the load)
- Do not install CTs on conductors that exceed the CT's rated current • Ensure the CT is fully closed and securely latched
- Keep the CT away from strong magnetic fields that could interfere with measurements

9. Calibration Procedure

To achieve the specified accuracy, the FE102M Energy Monitor requires proper calibration. This section outlines the calibration procedure for voltage, current, and power measurements.

Equipment Required

- Precision voltage source (AC)
- Precision current source or known load
- Reference energy meter or power analyzer • Digital multimeter
- Variable load (resistive, inductive, and capacitive)

Calibration Steps

1. Voltage Calibration

1. Connect a known reference voltage to the AC voltage input.
2. Measure the voltage with your reference meter.
3. Read the uncalibrated voltage from the FE102M.
4. Calculate the voltage gain calibration factor:

```
UgainA = (Reference Voltage / Measured Voltage) × Current UgainA Value
```

5. Update the UgainA register with the new value.
6. Verify the calibration by comparing the FE102M reading with your reference meter.

2. Current Calibration

1. Connect a known current through the CT clamp.
2. Measure the current with your reference meter.
3. Read the uncalibrated current from the FE102M.
4. Calculate the current gain calibration factor:

```
IgainA = (Reference Current / Measured Current) × Current IgainA Value
```

5. Update the IgainA register with the new value.
6. Verify the calibration by comparing the FE102M reading with your reference meter.

3. Power Calibration

1. Connect both voltage and current inputs with a known power factor load.
2. Measure the active power with your reference meter.
3. Read the uncalibrated active power from the FE102M.
4. Calculate the power gain calibration factor:

```
PPosA = (Reference Power / Measured Power) × Current PPosA Value
```

5. Update the PPosA register with the new value.
6. For reactive power calibration, repeat with an inductive or capacitive load.

Saving Calibration Values

Calibration values should be saved to the onboard EEPROM to ensure they persist after power cycles. The example code in the Programming Examples section includes functions for saving and loading calibration values.

Default Calibration Values

If you need to restore default calibration values, use the following starting points and adjust as needed:

cpp

```cpp
#define DEFAULT_UGAIN 0xF709
#define DEFAULT_IGAIN 0x7DFB
#define DEFAULT_LGAIN 0x1D39
#define DEFAULT_CRC1  0xAE70
#define DEFAULT_CRC2  0x3D1D
```

Note: These values are only starting points. Proper calibration is required for accurate measurements.

10. Arduino Library Implementation

The FE102M Energy Monitor can be programmed using Arduino IDE and C++. This section provides an overview of the library structure and implementation.

Library Overview

The FE102M Energy Monitor uses a customized library for the ATM90E26 energy monitor IC. The library provides functions for initializing the IC, reading measurements, and calibrating the device.

Library Files

EnergyATM90E26.h

cpp

```cpp
/*
 * SM Energy FE102M Energy Monitor Library
 * Based on ATM90E26 Energy Monitor IC
 */

#ifndef EnergyATM90E26_h
#define EnergyATM90E26_h

#include <Arduino.h>
#include <SPI.h>

/* ATM90E26 Register Addresses */
// System Registers
#define SoftReset    0x00  // Software Reset
#define SysStatus    0x01  // System Status
#define FuncEn       0x02  // Function Enable
#define SagTh        0x03  // Voltage Sag Threshold
#define MMode        0x04  // Metering Mode Configuration
#define CSOne        0x05  // Checksum 1
#define CSTwo        0x06  // Checksum 2
#define Gain         0x07  // Measurement Gain

// Calibration Registers
#define Ugain        0x08  // Voltage RMS Gain
#define IgainL       0x09  // L Line Current Gain
#define Uoffset      0x0A  // Voltage Offset
#define IoffsetL     0x0B  // L Line Current Offset
#define PoffsetL     0x0C  // L Line Active Power Offset
#define QoffsetL     0x0D  // L Line Reactive Power Offset
#define Lgain        0x0E  // L Line Calibration Gain

// Energy Registers
#define APenergy     0x40  // Forward Active Energy
#define ANenergy     0x41  // Reverse Active Energy
#define RPenergy     0x42  // Forward Reactive Energy
#define RNenergy     0x43  // Reverse Reactive Energy
#define ATenergy     0x44  // Absolute Active Energy
#define Rtenergy     0x45  // Absolute Reactive Energy

// Measurement Registers
#define Irms         0x48  // Current RMS
#define Urms         0x49  // Voltage RMS
#define Pmean        0x4A  // Mean Active Power
#define Qmean        0x4B  // Mean Reactive Power
#define Smean        0x4C  // Mean Apparent Power
#define PowerF       0x4D  // Power Factor
```

```
#define PAngle      0x4E  // Phase Angle
#define Freq        0x4F  // Line Frequency

// Configuration Registers
#define CalStart    0x20  // Calibration Start Command
#define PLconstH    0x21  // High Word of PL_Constant
#define PLconstL    0x22  // Low Word of PL_Constant
#define PStartTh    0x27  // Active Startup Power Threshold
#define QStartTh    0x28  // Reactive Startup Power Threshold
#define PNolTh      0x29  // Active No-Load Power Threshold
#define QNolTh      0x2A  // Reactive No-Load Power Threshold
#define AdjStart    0x30  // Measurement Calibration Start Command
#define EnStatus    0x31  // Metering Status
#define LSB         0x32  // LSB Value

class ATM90E26_SPI {
private:
  int _cs;
  unsigned short _lgain;
  unsigned short _ugain;
  unsigned short _igain;
  unsigned short _crc1;
  unsigned short _crc2;

public:
  // Constructor
  ATM90E26_SPI(int pin);

  // Initialization
  void InitEnergyIC();

  // Set calibration values
  void SetLGain(unsigned short lgain);
  void SetUGain(unsigned short ugain);
  void SetIGain(unsigned short igain);
  void SetCRC1(unsigned short crc1);
  void SetCRC2(unsigned short crc2);

  // Communication
  unsigned short CommEnergyIC(unsigned char RW, unsigned char address, unsigned short val);

  // Measurement functions
  double GetLineVoltage();
  double GetLineCurrent();
  double GetActivePower();
  double GetFrequency();
  double GetPowerFactor();
```

```
    double GetImportEnergy();
    double GetExportEnergy();
    double GetAbsActiveEnergy();
    double GetReactivefwdEnergy();
    double GetAbsReactiveEnergy();

    // Status functions
    unsigned short GetSysStatus();
    unsigned short GetMeterStatus();
    unsigned short GetCalStartStatus();
    unsigned short GetLSBStatus();
    unsigned short GetUGain();
    unsigned short GetLGain();
    unsigned short GetIGain();
    unsigned short GetMModeStatus();
    unsigned short GetCS1Status();
    unsigned short GetCS2Status();
    unsigned short GetCS1Calculated();
    unsigned short GetCS2Calculated();
};

#endif
```

EnergyATM90E26.cpp

This file contains the implementation of the functions declared in the header file. It provides methods for communicating with the ATM90E26 IC, reading measurements, and setting calibration values.

See the attached EnergyATM90E26.cpp file for the complete implementation.

Defaults and Calibration

The FE102M_Defaults.h file contains default calibration values and initialization routines. These values
provide a starting point for calibration.

See the attached FE102M_Defaults.h file for the complete implementation.

11. Programming Examples

This section provides example code for common tasks with the FE102M Energy Monitor.

Basic Energy Monitoring

cpp

```
/*
*    FE102M Energy Monitor - Basic Energy Monitoring Example
*
*    This example demonstrates basic energy monitoring using the SM Energy FE102M
*    Energy Monitor with ESP32 and ATM90E26 energy monitoring IC.
*
*    Reads and displays voltage, current, power, energy, and other electrical parameters.
*/

#include <SPI.h>
#include <Wire.h>
#include "EnergyATM90E26.h"

// Define chip select pin
#define CS_PIN 5  // GPIO5 - ATM90E26 SPI Chip Select

// Create an instance of the ATM90E26_SPI class
ATM90E26_SPI energy(CS_PIN);

void setup() {
// Initialize serial communication
Serial.begin(115200);
Serial.println("FE102M Energy Monitor - Basic Example");

// Initialize SPI for ATM90E26 communication
SPI.begin();

// Initialize the ATM90E26 IC
energy.InitEnergyIC();

// Check system status
unsigned short systemStatus = energy.GetSysStatus(); Serial.print("System Status: 0x"); Serial.println(systemStatus, HEX);

if (systemStatus & 0xC000) { Serial.println("Checksum 1 Error!");
}

if (systemStatus & 0x3000) { Serial.println("Checksum 2 Error!");
}
}
```

```
void loop() {
// Read energy parameters
float voltage = energy.GetLineVoltage();
float current = energy.GetLineCurrent();
float power = energy.GetActivePower();
float frequency = energy.GetFrequency();
float pf = energy.GetPowerFactor();
float importEnergy = energy.GetImportEnergy();
float exportEnergy = energy.GetExportEnergy();

// Print results
Serial.println("\n--- Energy Monitor Readings ---");
Serial.print("Voltage: ");
Serial.print(voltage, 2);
Serial.println(" V");

Serial.print("Current: ");
Serial.print(current, 3);
Serial.println(" A");

Serial.print("Active Power: ");
Serial.print(power, 2);
Serial.println(" W");

Serial.print("Frequency: ");
Serial.print(frequency, 2);
Serial.println(" Hz");

Serial.print("Power Factor: ");
Serial.println(pf, 3);

Serial.print("Import Energy: ");
Serial.print(importEnergy, 4);
Serial.println(" kWh");

Serial.print("Export Energy: ");
Serial.print(exportEnergy, 4);
Serial.println(" kWh");

Serial.println("          ---------------------------------------------------     ");

// Wait before next reading
delay(2000);
}
```

LCD Display Implementation

cpp

```cpp
/*
*     FE102M Energy Monitor - LCD Display Example
*
*     This example demonstrates how to display energy monitoring data
*     on a 20x4 LCD screen using I2C communication.
*/

#include <SPI.h>
#include<Wire.h>
#include<LiquidCrystal_I2C.h>
#include "EnergyATM90E26.h"

// Define pins
#define CS_PIN 5  // GPIO5 - ATM90E26 SPI Chip Select

// Create instances
ATM90E26_SPI energy(CS_PIN);
LiquidCrystal_I2C lcd(0x27, 20, 4);  // I2C address 0x27, 20x4 LCD

void setup() {
// Initialize serial communication
Serial.begin(115200);
Serial.println("FE102M Energy Monitor - LCD Example");

// Initialize SPI for ATM90E26 communication
SPI.begin();

// Initialize I2C LCD
Wire.begin();
lcd.init();

lcd.backlight();

lcd.clear();

lcd.setCursor(0, 0);
lcd.print("FE102M Energy Monitor");
lcd.setCursor(0, 1);
lcd.print("Initializing...");

// Initialize the ATM90E26 IC
energy.InitEnergyIC();

// Check system status
unsigned short systemStatus = energy.GetSysStatus();

if ((systemStatus & 0xC000) || (systemStatus & 0x3000)) { lcd.setCursor(0, 2);
```

```
lcd.print("Checksum Error!");
Serial.print("System Status: 0x"); Serial.println(systemStatus, HEX); delay(3000);
}

lcd.clear(); lcd.setCursor(0, 0);
lcd.print("FE102M Energy Monitor");
}

void loop() {
// Read energy parameters
float voltage = energy.GetLineVoltage();
float current = energy.GetLineCurrent();
float power = energy.GetActivePower();
float frequency = energy.GetFrequency();
float pf = energy.GetPowerFactor();
float importEnergy = energy.GetImportEnergy();

// Update LCD display
// Line 1 - Title (already set in setup)

// Line 2 - Voltage and Current
lcd.setCursor(0, 1);
lcd.print("V:");
lcd.print(voltage, 1);
lcd.print("V  I:");
lcd.print(current, 3);
lcd.print("A                ");

// Line 3 - Power and Frequency
lcd.setCursor(0, 2);
lcd.print("P:");
lcd.print(power, 0);
lcd.print("W  F:");
lcd.print(frequency, 1);
lcd.print("Hz               ");

// Line 4 - Power Factor and Energy
lcd.setCursor(0, 3);
lcd.print("PF:");
lcd.print(pf, 2);
lcd.print("  E:");
lcd.print(importEnergy, 3);
lcd.print("kWh");

// Wait before next update
```

```
delay(1000);
}
```

WiFi Data Logging Example

cpp

```
/*
*     FE102M Energy Monitor - WiFi Data Logging Example
*
*     This example demonstrates how to log energy monitoring data
*     to a web server using the ESP32's WiFi capabilities.
*/

#include <SPI.h>
#include <Wire.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include "EnergyATM90E26.h"

// Define pins
#define CS_PIN 5             // GPIO5 - ATM90E26 SPI Chip Select
#define WIFI_LED 14   // GPIO14 - WiFi Status LED

// WiFi credentials
const char* ssid = "YourWiFiSSID";
const char* password = "YourWiFiPassword";

// Web server details
const char* serverName = "http://example.com/api/energy_data";

// Create an instance of the ATM90E26_SPI class
ATM90E26_SPI energy(CS_PIN);

// Variable to track last upload time
unsigned long lastUploadTime = 0;
const unsigned long uploadInterval = 60000; // 1 minute interval

void setup() {
// Initialize serial communication
Serial.begin(115200);
Serial.println("FE102M Energy Monitor - WiFi Example");

// Initialize SPI for ATM90E26 communication
SPI.begin();

// Initialize WiFi LED
pinMode(WIFI_LED, OUTPUT);
digitalWrite(WIFI_LED, LOW); // LED off initially
```

```
//Initialize the ATM90E26 IC
energy.InitEnergyIC();
// Connect to WiFi
WiFi.begin(ssid, password);
Serial.print("Connecting to WiFi");

while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
digitalWrite(WIFI_LED, !digitalRead(WIFI_LED)); // Toggle LED
}

Serial.println();
Serial.println("Connected to WiFi");
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

// WiFi connected, keep LED on
digitalWrite(WIFI_LED, HIGH);
}

void loop() {
// Read energy parameters
float voltage = energy.GetLineVoltage();
float current = energy.GetLineCurrent();
float power = energy.GetActivePower();
float frequency = energy.GetFrequency();
float pf = energy.GetPowerFactor();
float importEnergy = energy.GetImportEnergy();

// Print to serial
Serial.println("\n--- Energy Monitor Readings ---");
Serial.print("Voltage: ");
Serial.print(voltage, 2);
Serial.println(" V");
Serial.print("Current: ");
Serial.print(current, 3);
Serial.println(" A");
Serial.print("Power: ");
Serial.print(power, 2);
Serial.println(" W");
```

```
// Check if it's time to upload data
if (millis() - lastUploadTime >= uploadInterval) { lastUploadTime = millis();
uploadData(voltage, current, power, frequency, pf, importEnergy);
}

delay(1000); // Wait 1 second between readings
}

void uploadData(float voltage, float current, float power, float frequency, float pf, float ene
// Check WiFi connection status
if (WiFi.status() == WL_CONNECTED) { HTTPClient http;

// Begin HTTP connection
http.begin(serverName);

// Specify content-type header
http.addHeader("Content-Type", "application/x-www-form-urlencoded");

// Prepare data to send
String httpRequestData = "voltage=" + String(voltage) +
"&current=" + String(current) + "&power=" + String(power) + "&frequency=" + String(frequency) + "&pf=" + String(pf) +
"&energy=" + String(energy);

// Send HTTP POST request
int httpResponseCode = http.POST(httpRequestData);

// Check response
if (httpResponseCode > 0) { Serial.print("HTTP Response code: "); Serial.println(httpResponseCode); String payload =
http.getString(); Serial.println(payload);

digitalWrite(WIFI_LED, HIGH); // Success, keep LED on
} else {
Serial.print("Error code: "); Serial.println(httpResponseCode);

// Indicate error with LED digitalWrite(WIFI_LED, LOW); delay(200); digitalWrite(WIFI_LED, HIGH);
}

// Free resources
http.end();
} else {
Serial.println("WiFi Disconnected");
// Try to reconnect
```

WiFi.reconnect();

// Indicate disconnection with LED
digitalWrite(WIFI_LED, LOW);
}
}

MODBUS RTU Implementation

cpp

```cpp
/*
 * FE102M Energy Monitor - MODBUS RTU Example
 *
 * This example demonstrates how to implement MODBUS RTU slave functionality
 * using the FE102M Energy Monitor.
 */

#include <SPI.h>
#include <Wire.h>
#include <ModbusMaster.h>
#include "EnergyATM90E26.h"

// Define pins
#define CS_PIN 5          // GPIO5 - ATM90E26 SPI Chip Select
#define MODBUS_LED 15     // GPIO15 - MODBUS LED
#define MAX485_DE 13      // GPIO13 - MAX485 DE/RE pin for MODBUS
#define MAX485_RO 16      // GPIO16 - MAX485 RO pin (RXD)
#define MAX485_DI 17      // GPIO17 - MAX485 DI pin (TXD)

// MODBUS slave ID
#define SLAVE_ID 1

// MODBUS register mapping
#define REG_VOLTAGE        0
#define REG_CURRENT        1
#define REG_ACTIVE_POWER   2
#define REG_REACTIVE_POWER 3
#define REG_APPARENT_POWER 4
#define REG_POWER_FACTOR   5
#define REG_FREQUENCY      6
#define REG_IMPORT_ENERGY  7
#define REG_EXPORT_ENERGY  8

// Create instances
ATM90E26_SPI energy(CS_PIN);
ModbusMaster modbus;

// Buffer for MODBUS register values (16-bit registers)
uint16_t modbusRegisters[20];

// Function prototypes
void preTransmission();
void postTransmission();
void updateModbusRegisters();

void setup() {
```

```
  // Initialize serial communication
  Serial.begin(115200);
  Serial.println("FE102M Energy Monitor - MODBUS RTU Example");

  // Initialize SPI for ATM90E26 communication
  SPI.begin();

  // Initialize MODBUS LED
  pinMode(MODBUS_LED, OUTPUT);
  digitalWrite(MODBUS_LED, LOW);

  // Initialize MAX485 control pin
  pinMode(MAX485_DE, OUTPUT);
  digitalWrite(MAX485_DE, LOW); // Receive mode

  // Initialize the ATM90E26 IC
  energy.InitEnergyIC();

  // Initialize MODBUS communication (Serial2 for ESP32)
  Serial2.begin(9600, SERIAL_8N1, MAX485_RO, MAX485_DI);

  // Initialize ModbusMaster
  modbus.begin(SLAVE_ID, Serial2);

  // Callbacks for RS485 flow control
  modbus.preTransmission(preTransmission);
  modbus.postTransmission(postTransmission);

  Serial.println("MODBUS RTU initialized as slave ID " + String(SLAVE_ID));
}

void loop() {
  // Read energy parameters and update MODBUS registers
  updateModbusRegisters();

  // Process MODBUS requests
  // Note: In a real implementation, use a MODBUS slave library
  // This example just shows the concept

  // Simulate MODBUS activity for demonstration
  digitalWrite(MODBUS_LED, HIGH);
  delay(50);
  digitalWrite(MODBUS_LED, LOW);

  // Print current values
  Serial.println("\n--- Energy Monitor Readings ---");
  Serial.print("Voltage: ");
```

```
  Serial.print(energy.GetLineVoltage(), 2);
  Serial.println(" V");
  Serial.print("Current: ");
  Serial.print(energy.GetLineCurrent(), 3);
  Serial.println(" A");
  Serial.print("Power: ");
  Serial.print(energy.GetActivePower(), 2);
  Serial.println(" W");

  delay(1000); // Wait 1 second between updates
}

// MODBUS pre-transmission callback
void preTransmission() {
  digitalWrite(MAX485_DE, HIGH); // Enable transmission
  delayMicroseconds(50);
}

// MODBUS post-transmission callback
void postTransmission() {
  delayMicroseconds(50);
  digitalWrite(MAX485_DE, LOW); // Enable reception
}

// Update MODBUS registers with current readings
void updateModbusRegisters() {
  // Read values from ATM90E26
  float voltage = energy.GetLineVoltage();
  float current = energy.GetLineCurrent();
  float activePower = energy.GetActivePower();
  float powerFactor = energy.GetPowerFactor();
  float frequency = energy.GetFrequency();
  float importEnergy = energy.GetImportEnergy();
  float exportEnergy = energy.GetExportEnergy();

  // Convert floating point values to fixed-point for MODBUS
  // Multiply by 100 to preserve 2 decimal places
  modbusRegisters[REG_VOLTAGE] = (uint16_t)(voltage * 100);
  modbusRegisters[REG_CURRENT] = (uint16_t)(current * 1000);
  modbusRegisters[REG_ACTIVE_POWER] = (uint16_t)activePower;
  modbusRegisters[REG_POWER_FACTOR] = (uint16_t)(powerFactor * 1000);
  modbusRegisters[REG_FREQUENCY] = (uint16_t)(frequency * 100);

  // Energy values need special handling (32-bit values split into two registers)
  uint32_t importEnergyFixed = (uint32_t)(importEnergy * 1000);
  modbusRegisters[REG_IMPORT_ENERGY] = (uint16_t)(importEnergyFixed >> 16); // High word
  modbusRegisters[REG_IMPORT_ENERGY + 1] = (uint16_t)(importEnergyFixed & 0xFFFF); // Low word

  uint32_t exportEnergyFixed = (uint32_t)(exportEnergy * 1000);
  modbusRegisters[REG_EXPORT_ENERGY] = (uint16_t)(exportEnergyFixed >> 16); // High word
  modbusRegisters[REG_EXPORT_ENERGY + 1] = (uint16_t)(exportEnergyFixed & 0xFFFF); // Low word
}
```

EEPROM Calibration Storage

cpp

```cpp
/*
 * FE102M Energy Monitor - EEPROM Calibration Storage Example
 *
 * This example demonstrates how to save and load calibration
 * values to/from the onboard 24C256 EEPROM.
 */

#include <SPI.h>
#include <Wire.h>
#include "EnergyATM90E26.h"

// Define pins
#define CS_PIN 5      // GPIO5 - ATM90E26 SPI Chip Select
#define SDA_PIN 21    // GPIO21 - I2C SDA
#define SCL_PIN 22    // GPIO22 - I2C SCL

// EEPROM address (24C256)
#define EEPROM_ADDR 0x50

// EEPROM addresses for calibration values
#define EEPROM_UGAIN_ADDR 0
#define EEPROM_IGAIN_ADDR 2
#define EEPROM_LGAIN_ADDR 4
#define EEPROM_CRC1_ADDR  6
#define EEPROM_CRC2_ADDR  8

// Default calibration values
#define DEFAULT_UGAIN 0xF709
#define DEFAULT_IGAIN 0x7DFB
#define DEFAULT_LGAIN 0x1D39
#define DEFAULT_CRC1  0xAE70
#define DEFAULT_CRC2  0x3D1D

// Create an instance of the ATM90E26_SPI class
ATM90E26_SPI energy(CS_PIN);

// Function prototypes
void saveCalibrationToEEPROM();
bool loadCalibrationFromEEPROM();
void writeEEPROM(int deviceAddress, unsigned int eeAddress, byte data);
byte readEEPROM(int deviceAddress, unsigned int eeAddress);
void writeWord(int deviceAddress, unsigned int eeAddress, unsigned short data);
unsigned short readWord(int deviceAddress, unsigned int eeAddress);

void setup() {
  // Initialize serial communication
```

```
  Serial.begin(115200);
  Serial.println("FE102M Energy Monitor - EEPROM Calibration Example");

  // Initialize I2C
  Wire.begin(SDA_PIN, SCL_PIN);

  // Initialize SPI for ATM90E26 communication
  SPI.begin();

  // Check if calibration exists in EEPROM
  Serial.println("Checking for calibration data in EEPROM...");
  if (loadCalibrationFromEEPROM()) {
    Serial.println("Calibration loaded from EEPROM successfully");
  } else {
    Serial.println("No valid calibration found, using defaults");
    // Set default calibration values
    energy.SetUGain(DEFAULT_UGAIN);
    energy.SetIGain(DEFAULT_IGAIN);
    energy.SetLGain(DEFAULT_LGAIN);
    energy.SetCRC1(DEFAULT_CRC1);
    energy.SetCRC2(DEFAULT_CRC2);

    // Save defaults to EEPROM
    saveCalibrationToEEPROM();
  }

  // Initialize the ATM90E26 IC
  energy.InitEnergyIC();

  // Check system status
  unsigned short systemStatus = energy.GetSysStatus();
  Serial.print("System Status: 0x");
  Serial.println(systemStatus, HEX);

  if (systemStatus & 0xC000) {
    Serial.println("Checksum 1 Error!");
  }

  if (systemStatus & 0x3000) {
    Serial.println("Checksum 2 Error!");
  }
}

void loop() {
  // Read energy parameters
  float voltage = energy.GetLineVoltage();
  float current = energy.GetLineCurrent();
```

```
  float power = energy.GetActivePower();

  // Print results
  Serial.println("\n--- Energy Monitor Readings ---");
  Serial.print("Voltage: ");
  Serial.print(voltage, 2);
  Serial.println(" V");

  Serial.print("Current: ");
  Serial.print(current, 3);
  Serial.println(" A");

  Serial.print("Active Power: ");
  Serial.print(power, 2);
  Serial.println(" W");

  // Print calibration values
  Serial.println("\n--- Calibration Values ---");
  Serial.print("UGain: 0x");
  Serial.println(energy.GetUGain(), HEX);
  Serial.print("IGain: 0x");
  Serial.println(energy.GetIGain(), HEX);
  Serial.print("LGain: 0x");
  Serial.println(energy.GetLGain(), HEX);

  Serial.println("------------------------------");

  // Wait before next reading
  delay(2000);
}

// Save calibration values to EEPROM
void saveCalibrationToEEPROM() {
  Serial.println("Saving calibration to EEPROM...");

  // Get current calibration values
  unsigned short ugain = energy.GetUGain();
  unsigned short igain = energy.GetIGain();
  unsigned short lgain = energy.GetLGain();
  unsigned short crc1 = energy.GetCS1Status();
  unsigned short crc2 = energy.GetCS2Status();

  // Write values to EEPROM
  writeWord(EEPROM_ADDR, EEPROM_UGAIN_ADDR, ugain);
  writeWord(EEPROM_ADDR, EEPROM_IGAIN_ADDR, igain);
  writeWord(EEPROM_ADDR, EEPROM_LGAIN_ADDR, lgain);
  writeWord(EEPROM_ADDR, EEPROM_CRC1_ADDR, crc1);
```

12. MODBUS Implementation

The FE102M Energy Monitor includes built-in MODBUS RTU capability through the RS485 interface. This allows for integration with industrial automation systems, building management systems, and other MODBUS-enabled devices.

MODBUS Hardware Interface

The RS485 interface is implemented using a MAX485 transceiver chip with the following connections:

- **GPIO13** (MAX485 DE/RE pin): Transmit/receive enable control
- **GPIO16** (MAX485 RO pin): Receive data
- **GPIO17** (MAX485 DI pin): Transmit data
- **GPIO15** (MODBUS LED): Indicates MODBUS communication activity

MODBUS Register Map

The following table defines the MODBUS register map for accessing energy monitoring data:

Register Address	Description	Format	Unit	Access
0x0000-0x0001	Voltage RMS	32-bit float	V	Read
0x0002-0x0003	Current RMS	32-bit float	A	Read
0x0004-0x0005	Active Power	32-bit float	W	Read
0x0006-0x0007	Reactive Power	32-bit float	VAR	Read
0x0008-0x0009	Apparent Power	32-bit float	VA	Read
0x000A-0x000B	Power Factor	32-bit float	-	Read
0x000C-0x000D	Frequency	32-bit float	Hz	Read
0x000E-0x000F	Active Import Energy	32-bit float	kWh	Read
0x0010-0x0011	Active Export Energy	32-bit float	kWh	Read
0x0012-0x0013	Reactive Import Energy	32-bit float	kVARh	Read
0x0014-0x0015	Reactive Export Energy	32-bit float	kVARh	Read
0x0016-0x0017	Apparent Energy	32-bit float	kVAh	Read
0x0018	System Status	16-bit uint	-	Read
0x0019	Device Control	16-bit uint	-	R/W
0x001A	MODBUS Address	16-bit uint	-	R/W
0x001B	Reset Energy Counters	16-bit uint	-	Write

MODBUS Configuration

The default MODBUS configuration is:

- Slave Address: 1 (configurable) • Baud Rate: 9600
- Data Bits: 8 • Parity: None • Stop Bits: 1
- Mode: RTU

To change the MODBUS configuration, use the programming examples provided or modify the MODBUS settings through the device's user interface.

MODBUS Implementation Notes

- The device responds to standard MODBUS function codes:
- 0x03 (Read Holding Registers)
- 0x04 (Read Input Registers)
- 0x06 (Write Single Register)
- 0x10 (Write Multiple Registers)
- 32-bit float values are stored in IEEE-754 format with most significant register first.
- Reading energy values too frequently may impact the overall performance of the device. For optimal performance, limit MODBUS requests to once per second or less.

13. WiFi Connectivity

The FE102M Energy Monitor includes built-in WiFi connectivity through the ESP32 microcontroller. This allows for remote monitoring, data logging to cloud services, integration with home automation systems, and over-the-air firmware updates.

WiFi Configuration

The device can be configured to connect to your WiFi network using the Arduino code examples provided. The WiFi connection status is indicated by the WiFi LED (GPIO14).

Web Server Implementation

The ESP32 can be programmed to function as a web server, providing a web interface for monitoring energy data and configuring the device. The WiFi Data Logging Example shows how to implement basic WiFi connectivity and data transmission.

Cloud Integration

The FE102M Energy Monitor can be integrated with various cloud platforms for data logging and analytics, including:

- ThingSpeak
- Adafruit IO
- AWS IoT
- Google Cloud IoT
- Home Assistant
- Node-RED

Example code for these integrations can be found in the GitHub repository.

Over-the-Air (OTA) Updates

The ESP32 supports over-the-air firmware updates, allowing you to update the device's firmware remotely without physical access. This is particularly useful for devices installed in hard-to-reach locations.

WiFi Security Considerations

- Always use WPA2 or higher encryption for your WiFi network
- Change default passwords and credentials
- Use TLS/SSL for secure data transmission
- Consider implementing a firewall or VLAN for IoT devices
- Regularly update firmware to address security vulnerabilities

14. LCD Integration

The FE102M Energy Monitor includes a connector for an I2C 20x4 LCD module, allowing for real-time display of energy monitoring data without requiring a computer or network connection.

LCD Hardware Connection

The LCD module connects to the following pins:

- **GPIO21** (I2C SDA)
- **GPIO22** (I2C SCL)
- 5V power supply
- GND

LCD Display Layout

The default LCD display layout is as follows:

```
FE102M Energy Monitor V:230.0V  I:1.200A P:276W  F:50.0Hz PF:0.98   E:42.5kWh
```

This layout provides a comprehensive overview of the most important electrical parameters: • Line

1: Device identification

- Line 2: Voltage and current readings
- Line 3: Active power and frequency
- Line 4: Power factor and energy consumption

Custom LCD Display

You can customize the LCD display layout and content by modifying the Arduino code. The LCD Display Implementation example shows how to configure and update the LCD display.

15. Troubleshooting

This section provides solutions to common issues that may be encountered when using the FE102M Energy Monitor.

Power and Startup Issues

Problem	Possible Cause	Solution
Device doesn't power on	Incorrect power supply	Ensure 9-12V DC power supply is connected with correct polarity
RUN LED not illuminated	Initialization failure	Check power supply voltage, press RESET button
FAULT LED illuminated	ATM90E26 error	Check system status register, verify calibration values

Measurement Issues

Problem	Possible Cause	Solution
Voltage reading incorrect	Incorrect calibration	Perform voltage calibration procedure
Current reading incorrect	Incorrect CT type or orientation	Verify CT type matches jumper settings, check CT orientation
Power reading zero or very low	CT not detecting current	Ensure CT is properly installed around conductor
Erratic readings	Electrical noise	Check grounding, add filtering, keep away from noise sources

Communication Issues

Problem	Possible Cause	Solution
WiFi connection fails	Incorrect credentials or weak signal	Verify SSID and password, check signal strength
MODBUS communication fails	Incorrect wiring or settings	Verify RS485 wiring, check MODBUS settings
USB connection fails	Driver issue	Install proper USB-UART driver for CP2102
I2C LCD not working	Incorrect address or wiring	Check I2C address (usually 0x27 or 0x3F), verify wiring

Software Issues

Problem	Possible Cause	Solution
Upload fails	ESP32 in boot loop	Hold BOOT button while initiating upload
Checksum errors	Invalid calibration values	Restore default calibration values
Energy values reset	Power cycle without saving	Use EEPROM to store energy values
ESP32 crashes	Memory issues or watchdog timer	Check for memory leaks, implement watchdog reset

Diagnostic LEDs

The status LEDs on the front panel provide valuable diagnostic information:

- **WiFi LED**: Indicates WiFi connection status
- **RUN LED**: Indicates normal operation
- **FAULT LED**: Indicates error condition
- **MODBUS LED**: Indicates MODBUS communication activity
- **ATM90E26 Status LEDs**: Indicate specific energy monitor interrupt states

Factory Reset

To restore the device to factory settings:

1. Power off the device
2. Press and hold both SET and MODE buttons
3. Power on the device while holding the buttons
4. Continue holding both buttons for 5 seconds
5. Release the buttons when all LEDs flash

16. Technical Support

Documentation and Resources

- **GitHub Repository**: FE102M-Energy-Monitor
- **Wiki**: Detailed technical documentation and guides
- **Issues Tracker**: Report bugs and request features
- **Discussion Forum**: Community support and discussions

Contact Information

For technical support, please contact:

Chamil Vithanage

Microcode Embedded Solutions, Australia

Email: microcode-eng@outlook.com

Website: www. microcodeeng.com

Warranty Information

The FE102M Energy Monitor comes with a limited 1-year warranty covering manufacturing defects and component failures. This warranty does not cover:

- Damage due to improper installation or use
- Damage from power surges or incorrect power supply
- Modifications or repairs by unauthorized personnel
- Normal wear and tear

For warranty claims, please contact the manufacturer with your purchase information and a description of the issue.

Note: All example code provided is open source and may be freely used or modified as needed. It is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. Always ensure proper testing before deployment in critical applications.