Arquitetura da Biblioteca Arduino para Medidor Trifásico com CS5463 e ESP32

Este documento descreve a arquitetura proposta para uma biblioteca Arduino destinada a facilitar a criação de um medidor de energia trifásico utilizando três CIs CS5463 e um microcontrolador ESP32. A biblioteca visa encapsular a comunicação com os CIs, o cálculo das medições elétricas, a implementação dos alarmes solicitados e a preparação dos dados para envio via MQTT.

1. Visão Geral

A biblioteca será estruturada em torno de duas classes principais:

1. `CS5463_Sensor`: Representa e gerencia a comunicação e as leituras de um único chip CS5463.
2. `ThreePhaseMeter`: Orquestra três instâncias da classe `CS5463_Sensor` (uma para cada fase), agrega os dados, implementa a lógica de alarme trifásico e formata os dados para MQTT.

O usuário da biblioteca interagirá primariamente com a classe `ThreePhaseMeter`.

2. Comunicação SPI

- O ESP32 se comunicará com cada um dos três CIs CS5463 via SPI.
- Será utilizada a interface SPI padrão do ESP32 (VSPI ou HSPI).
- Cada CI CS5463 deverá ser conectado a um pino de Chip Select (CS) distinto no ESP32. Os pinos CS serão configuráveis pelo usuário ao instanciar a classe `ThreePhaseMeter`.
- A classe `CS5463_Sensor` encapsulará as transações SPI de baixo nível (leitura e escrita de registradores, envio de comandos) para um único chip, gerenciando a ativação/desativação do CS correspondente.
- A biblioteca utilizará a biblioteca SPI padrão do Arduino (`SPI.h`). Serão gerenciadas as configurações de SPI (velocidade, modo) adequadas para o CS5463, conforme o datasheet.

3. Classe `CS5463_Sensor`

Esta classe será responsável por interagir com um único chip CS5463.

Atributos Principais:

- Pino CS associado.
- Objeto SPI (referência à instância SPI utilizada).
- Constantes de calibração (ganho, offset para V e I, offset de potência, etc.).
- Configurações do chip (ganho do PGA, filtros, etc.).
- Últimas leituras (para acesso rápido e cálculo de frequência).

Métodos Principais:

- `begin(spi_bus, cs_pin, mclk_freq)` : Inicializa a comunicação SPI, reseta o chip, aplica configurações iniciais e verifica a comunicação.
- `readRegister(address)` : Lê um registrador de 24 bits.
- `writeRegister(address, value)` : Escreve em um registrador de 24 bits.
- `sendCommand(command)` : Envia um comando (ex: iniciar conversão, calibração).
- `readMeasurements()` : Lê todos os registradores de medição relevantes (V, I, P, Q, S, PF, Vrms, Irms, Vpeak, Ipeak, Temp, etc.) em uma única sequência para otimizar a comunicação SPI.
- `getInstantaneousVoltage()` , `getInstantaneousCurrent()` , `getInstantaneousPower()` : Retornam os valores instantâneos.
- `getVRMS()` , `getIRMS()` : Retornam os valores RMS.
- `getActivePower()` , `getReactivePower()` , `getApparentPower()` : Retornam as potências.
- `getPowerFactor()` : Retorna o fator de potência.
- `getPeakVoltage()` , `getPeakCurrent()` : Retornam os picos.
- `getTemperature()` : Retorna a temperatura interna.
- `getFundamentalActivePower()` , `getFundamentalReactivePower()` , `getHarmonicActivePower()` : Retornam as potências fundamental e harmônica.
- `getStatusRegister()` : Lê o registrador de status para verificar flags (DRDY, CRDY, erros).
- `performCalibration(type)` : Inicia sequências de calibração (DC offset, AC offset, DC gain, AC gain).
- `setCalibrationConstants(...)` : Define os valores de calibração a serem usados.
- `configure(...)` : Permite ajustar configurações como ganho do PGA, filtros HPF/IIR, contagem de ciclos (N).

4. Classe `ThreePhaseMeter`

Esta classe gerencia o sistema trifásico como um todo.

Atributos Principais:

- Três instâncias de `CS5463_Sensor` (fase A, B, C).
- Configurações de alarme (limites Vmin, Vmax, Imax, Vloss, durações mínimas).
- Status dos alarmes para cada fase.
- Tópicos MQTT base para publicação.
- Configurações de cálculo (ex: frequência da rede nominal para cálculo de frequência real).
- Últimas leituras agregadas.

Métodos Principais:

- `begin(spi_bus, cs_pin_A, cs_pin_B, cs_pin_C, mclk_freq)` : Inicializa as três instâncias `CS5463_Sensor` .
- `configureAlarms(vmin, vmax, imax, vloss, duration)` : Define os limites e durações para os alarmes.
- `configureMQTT(base_topic)` : Define o tópico MQTT base.
- `readAllPhases()` : Chama `readMeasurements()` para cada uma das três fases.
- `calculateFrequency(phase)` : Calcula a frequência da rede para uma fase específica (requer lógica adicional baseada em cruzamentos por zero ou análise do ciclo de cálculo).
- `checkAlarms()` : Após `readAllPhases()` , verifica as condições de alarme (V fora da faixa, I alta, falta de fase) para cada fase, atualizando os status internos.
- `getPhaseData(phase)` : Retorna uma estrutura ou objeto contendo todas as medições da fase especificada.
- `getAlarmStatus(phase)` : Retorna o status dos alarmes para a fase especificada.
- `getJsonPayload(phase)` : Formata os dados de medição e status de alarme da fase especificada em uma string JSON pronta para publicação MQTT. O JSON incluirá campos como `v` , `i` , `p` , `q` , `s` , `pf` , `freq` , `temp` , `vpeak` , `ipeak` , `alarm_vmin` , `alarm_vmax` , `alarm_imax` , `alarm_phaseloss` .
- `performFullCalibration()` : Guia o usuário ou executa um processo de calibração para as três fases.
- `applyCalibration(...)` : Aplica constantes de calibração para as três fases.

5. Integração MQTT

A biblioteca **não** incluirá um cliente MQTT completo. Em vez disso, ela fornecerá os dados formatados (via `getJsonPayload()`) para serem publicados pelo código principal do Arduino (sketch), que será responsável por:

- Incluir uma biblioteca MQTT (ex: `PubSubClient`).
- Estabelecer a conexão Wi-Fi.
- Conectar-se ao broker MQTT.
- Periodicamente chamar `readAllPhases()`, `checkAlarms()`.
- Obter os payloads JSON usando `getJsonPayload(phase)` para cada fase.
- Publicar os payloads nos tópicos MQTT apropriados (ex: `base_topic/phaseA`, `base_topic/phaseB`, `base_topic/phaseC`).

6. Fluxo de Uso Típico (no Sketch Arduino)

1. Incluir a biblioteca `ThreePhaseMeter.h`.
2. Instanciar `ThreePhaseMeter` com os pinos CS corretos.
3. No `setup()`:
    - Iniciar Serial.
    - Iniciar SPI.
    - Chamar `meter.begin(...)`.
    - Aplicar constantes de calibração (`meter.applyCalibration(...)`).
    - Configurar alarmes (`meter.configureAlarms(...)`).
    - Configurar MQTT (`meter.configureMQTT(...)`).
    - Conectar ao Wi-Fi e ao broker MQTT.
4. No `loop()`:
    - Manter conexão MQTT.
    - Chamar `meter.readAllPhases()`.
    - Chamar `meter.checkAlarms()`.
    - Para cada fase (A, B, C):
        - Obter payload JSON: `String payload = meter.getJsonPayload(phase);`
        - Publicar no tópico MQTT correspondente.
    - Adicionar um delay apropriado.

Esta arquitetura visa modularidade, encapsulamento e facilidade de uso, separando a interação de baixo nível com o hardware (CS5463) da lógica de aplicação trifásica e da comunicação externa (MQTT).