MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

SENSOR DE CORRENTE ACS712 NA STM32MP1 DK1

PROFESSOR:
Nicolas Souza de Melo Miranda de Oliveira

ALUNOS: SCARPINI, FEDATO e MATHEUS ARAUJO

PROGRAMAÇÃO APLICADA

RIO DE JANEIRO

2025

Sumário

0.1 Introdução

O presente trabalho tem como objetivo o desenvolvimento de um sistema embarcado capaz de realizar a leitura de corrente elétrica utilizando o sensor **ACS712**, processando os sinais por meio da placa **STM32MP1 DK1**. O sistema foi projetado para demonstrar o fluxo completo de aquisição, conversão e transmissão de dados de corrente, bem como sua visualização em uma interface gráfica.

O projeto integra conceitos de eletrônica analógica e digital, programação orientada a objetos em `C++`, comunicação em rede via protocolo `UDP`, e exibição dos resultados em um painel desenvolvido em `Python` com bibliotecas gráficas.

O objetivo principal é mostrar a integração entre hardware e software em um ambiente embarcado, de forma modular e escalável para futuras aplicações industriais ou acadêmicas.

0.2 Fluxograma

O fluxograma da Figura 1 representa o processo de leitura, conversão e transmissão da corrente elétrica no sistema embarcado.

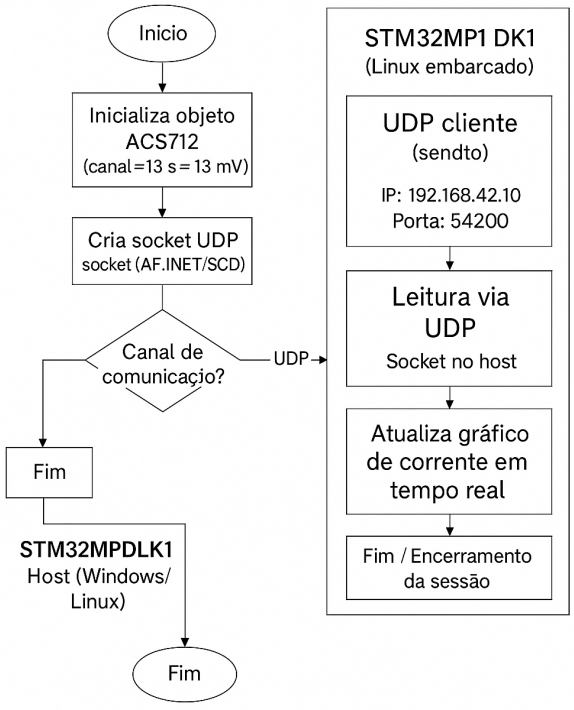

Figura 1 – Fluxo de funcionamento do sistema

Figura 1 – Fluxo de funcionamento do sistema

O processo inicia com a leitura do ADC via sysfs, passa pela conversão em ampères, exibição no terminal e envio dos valores por UDP.

0.3 Diagrama de Classes

O sistema foi implementado com programação orientada a objetos, utilizando a classe `ACS712` para encapsular as funções de leitura e conversão de corrente.

Classe: ACS712
- channel : int - sensitivity : float - scale : float - basePath : string - rawPath : string
+ ACS712(channel, sensitivity) + readCurrent() : float - readIntFromFile(path : string) : int

0.4 Instruções de Compilação e Uso

0.4.1 Compilação cruzada

Para compilar o código para a STM32MP1 DK1, utilize o compilador ARM:

```
arm-linux-gnueabihf-g++ acs712_reader.cpp -o acs712_reader
arm-linux-gnueabihf-g++ acs712_reader_com_UDP.cpp -o acs712_udp
```

0.4.2 Execução

Envie o binário para a placa (via SCP ou pendrive) e execute:

```
./acs712_udp
```

No computador, abra um terminal para receber os pacotes UDP:

```
nc -ul 45200
```

0.5 Documentação do Projeto, do Código e do Protocolo

A documentação automática foi gerada com o **Doxygen**, conforme o arquivo `DoxyFileTrab3.txt`. O protocolo UDP usa o endereço `192.168.42.10` e a porta `45200` para envio dos dados.

0.6 Descrição do Sensor e Funcionamento

O **ACS712** é um sensor de corrente baseado no **efeito Hall**. Quando uma corrente passa por seu condutor interno, cria-se um campo magnético. Esse campo é detectado por uma célula Hall, que gera uma tensão proporcional. Essa tensão é convertida em valor digital pelo ADC da STM32MP1 e, a partir dela, é calculada a corrente elétrica:

$$I = \frac{V_{sensor} - 2,5V}{185\,mV/A}$$

0.7 Capturas de Tela da Interface

As Figuras 2 e 3 mostram o dashboard desenvolvido em Python para monitorar os dados recebidos via UDP.

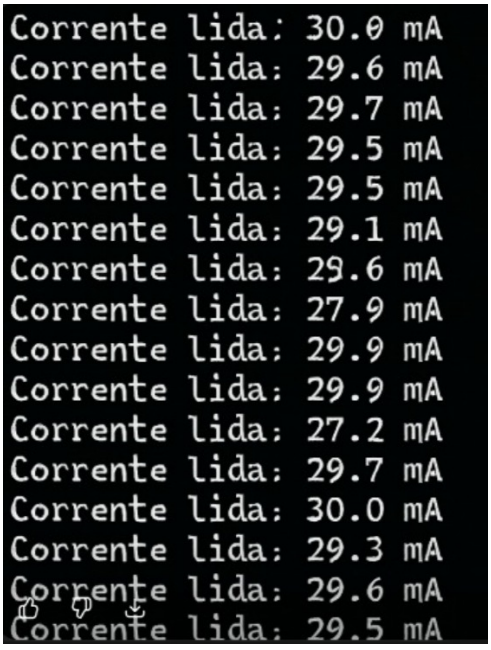

Figura 2 – Leituras de corrente exibidas no terminal

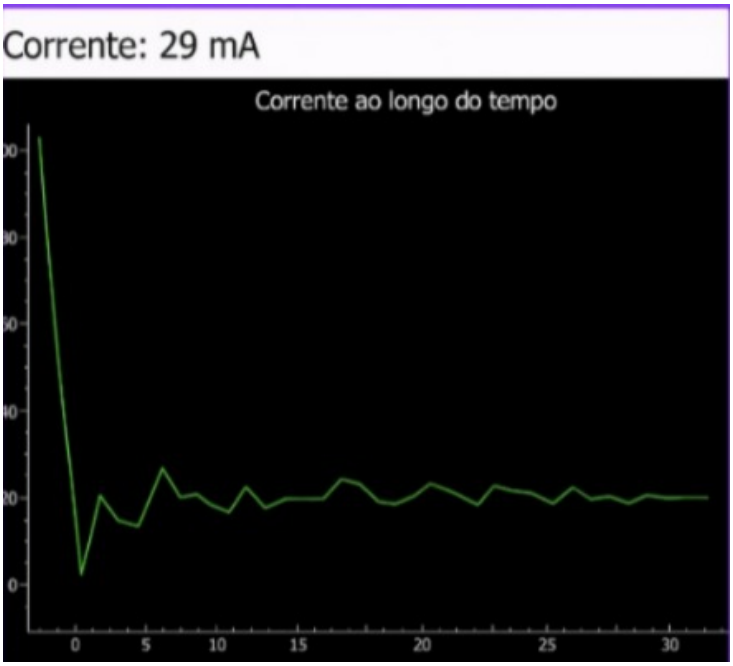

Figura 3 – Interface gráfica em PyQt5 e PyQtGraph mostrando corrente em tempo real

0.8 Análise dos Valores Medidos

Durante os testes com alimentação de 3 V e resistor de 1 kΩ, o sistema apresentou leituras estáveis entre 29 mA e 30 mA. A comunicação UDP foi validada por meio do **Wireshark** e do **ncat**, mostrando pacotes transmitidos com frequência de 100 ms.

0.9 Código-Fonte Principal (acs712_reader.cpp)

```
#include <iostream>
#include <fstream>
#include <string>
#include <stdexcept>
#include <thread>
#include <chrono>

class ACS712 {
public:
    ACS712(int channel, float sensitivity_mV_per_A)
        : channel(channel), sensitivity(sensitivity_mV_per_A) {
        basePath = "/sys/bus/iio/devices/iio:device0/";
        rawPath = basePath + "in_voltage" + std::to_string(channel) + "
            _raw";
        scale = 1.0;
    }
```

```
16
17      float readCurrent() {
18          int raw = readIntFromFile(rawPath);
19          float voltage_mV = raw * scale;
20          float zeroCurrentVoltage = 2500.0;
21          float current = (voltage_mV - zeroCurrentVoltage) / sensitivity;
22          return current;
23      }
24
25  private:
26      int channel;
27      float sensitivity;
28      float scale;
29      std::string basePath;
30      std::string rawPath;
31
32      int readIntFromFile(const std::string &path) {
33          std::ifstream file(path);
34          if (!file.is_open())
35              throw std::runtime_error("Erro ao abrir " + path);
36          int value;
37          file >> value;
38          return value;
39      }
40  };
41
42  int main() {
43      try {
44          ACS712 sensor(13, 185.0);
45          while (true) {
46              float current = sensor.readCurrent();
47              std::cout << "Corrente lida: " << current << " A" << std::
                    endl;
48              std::this_thread::sleep_for(std::chrono::milliseconds(100));
49          }
50      } catch (const std::exception &e) {
51          std::cerr << "Erro: " << e.what() << std::endl;
52          return 1;
53      }
54      return 0;
55  }
```

0.10 Código com Comunicação UDP em Formato JSON

```
1  #include <iostream>
```

```cpp
#include <fstream>
#include <string>
#include <stdexcept>
#include <thread>
#include <chrono>
#include <cstring>
#include <arpa/inet.h>
#include <unistd.h>

class ACS712 {
public:
    ACS712(int channel, float sensitivity_mV_per_A)
        : channel(channel), sensitivity(sensitivity_mV_per_A) {
        basePath = "/sys/bus/iio/devices/iio:device0/";
        rawPath = basePath + "in_voltage" + std::to_string(channel) + "
            _raw";
        scale = 1.0;
    }

    float readCurrent() {
        int raw = readIntFromFile(rawPath);
        float voltage_mV = raw * scale;
        float zeroCurrentVoltage = 2500.0;
        float current = (voltage_mV - zeroCurrentVoltage) / sensitivity;
        return current;
    }

private:
    int channel;
    float sensitivity;
    float scale;
    std::string basePath;
    std::string rawPath;

    int readIntFromFile(const std::string &path) {
        std::ifstream file(path);
        if (!file.is_open())
            throw std::runtime_error("Erro ao abrir " + path);
        int value;
        file >> value;
        return value;
    }
};

int main() {
    try {
        ACS712 sensor(13, 185.0);
```

```
48          int sock = socket(AF_INET, SOCK_DGRAM, 0);
49          if (sock < 0)
50              throw std::runtime_error("Erro ao criar socket UDP.");
51
52          sockaddr_in serverAddr{};
53          serverAddr.sin_family = AF_INET;
54          serverAddr.sin_port = htons(45200);
55          inet_pton(AF_INET, "192.168.42.10", &serverAddr.sin_addr);
56
57          std::cout << "Iniciando envio de dados em formato JSON..." <<
                std::endl;
58
59          while (true) {
60              float current = sensor.readCurrent();
61              std::string message = "{\"corrente\": " + std::to_string(
                    current) + "}";
62              sendto(sock, message.c_str(), message.size(), 0,
63                     (sockaddr*)&serverAddr, sizeof(serverAddr));
64              std::cout << "Enviado (JSON): " << message << std::endl;
65              std::this_thread::sleep_for(std::chrono::milliseconds(1000))
                    ;
66          }
67
68          close(sock);
69      } catch (const std::exception &e) {
70          std::cerr << "Erro: " << e.what() << std::endl;
71          return 1;
72      }
73      return 0;
74  }
```