Projeto Final: Sistemas Digitais 2
MEDIDOR DE TEMPERATURA BLUETOOTH

Felipe Pereira Pontes
Departamento Engenharia Mecânica
Universidade Federal de Pernambuco
Recife, Pernambuco
felipe.ppontes@ufpe.br

***Abstract*—Final Project Report for the Digital Systems 2 Course (ME587), 2024.1, of the Mechanical Engineering Program at the Center for Technology and Geosciences of the Federal University of Pernambuco.**
***Professor* — José Rodrigues de Oliveira Neto.**

I. Introduction

O projeto escolhido é um Medidor de Temperatura Bluetooth, cuja principal funcionalidade é medir e exibir a temperatura em tempo real, além de transmitir esses mesmos dados para um smartphone via Bluetooth.

A. *Funcionalidades Principais:*

- **Medição da Temperatura:** A temperatura será medida periodicamente utilizando um sensor de temperatura.
- **Exibição da Temperatura:** A temperatura medida será exibida em tempo real no display de 7 segmentos disponível no Multifunction-Shield Arduino. *[Iremos substiruir provisóriamente o LM35 pelo potenciômetro da Shield por falta de alimentação do sensor].*
- **Transmissão dos Dados:** Os valores da temperatura serão enviados para um smartphone através de um módulo Bluetooth BLE V4.0 HM-10 Keyes.

"Fig. 1" Fluxograma do funcionamento do projeto.

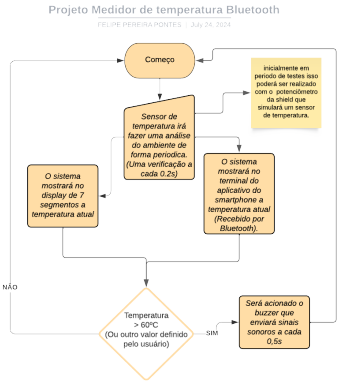

Fig. 1. Descrição da imagem. Fonte: LucidChart

Identify applicable funding agency here. If none, delete this.

B. *Funcionalidades Futuras:*

Para aprimorar o projeto do Medidor de Temperatura Bluetooth, diversas funcionalidades futuras podem ser implementadas. Estas melhorias visam aumentar a funcionalidade, robustez e a usabilidade do sistema. As principais adições são a conectividade à internet, a criação de uma estrutura física avançada e a construção de uma placa de circuito impresso específica.

- **Conectividade à Internet:** Integrar um módulo Wi-Fi para permitir a conexão à internet e enviar alertas via e-mail ou mensagem de texto quando uma temperatura crítica for atingida, com o objetivo de melhorar o monitoramento remoto e garantir reações rápidas a condições críticas.
- **Estrutura Física com Impressão 3D e Corte a Laser:** Desenvolver uma estrutura robusta utilizando impressão 3D e corte a laser, com um display LCD 16x2 para uma interface intuitiva, com o objetivo de facilitar o manuseio, tornando o dispositivo mais prático e atraente.
- **Placa de Circuito Impresso (PCB) Específica:** Projetar e construir uma PCB dedicada para integrar todos os componentes de forma compacta e eficiente, para tornar o sistema mais compacto, confiável e fácil de replicar e manter.

C. *Objetivo*

O objetivo do projeto é aplicar e reforçar os conhecimentos adquiridos durante a disciplina de Sistemas Digitais 2, especialmente no que diz respeito ao desenvolvimento de sistemas embarcados utilizando um sistema operacional de tempo real (RTOS), especificamente o FreeRTOS, em conjunto com o ambiente de desenvolvimento STM32CubeIDE. Através deste projeto, deve-se demonstrar sua capacidade de integrar sensores, exibir dados em tempo real e utilizar comunicação sem fio, nesse caso: conexão Bluetooth, para transmitir informações

.

II. Desenvolvimento

Para resolver o problema proposto, foram utilizados diversos componentes, incluindo uma placa de desenvolvimento, sensores, atuadores e principais periféricos. A seguir, detalho cada um dos elementos utilizados no projeto, a configuração inicial e como o firmware está estruturado em tasks, com subseções dedicadas a cada task.

A. *HARDWARE:*

- **Placa de Desenvolvimento:** STM32 (Blue Pill): Esta placa é baseada no microcontrolador STM32F103C8T6 da STMicroelectronics, que oferece diversas interfaces periféricas e é adequada para aplicações que requerem controle em tempo real. .

- **Sensores:** LM35 (Sensor de Temperatura): Este sensor é utilizado para medir a temperatura ambiente. Ele fornece uma saída linear e proporcional à temperatura em graus Celsius, com uma variação de 10 mV para cada grau Celsius.

- **Atuadores e Periféricos :**
  1) **Shield Multi-Functions:** Esta shield inclui um display de 7 segmentos e um buzzer, utilizados para exibir a temperatura e emitir alertas sonoros quando a temperatura ultrapassa um determinado valor.
  2) **Módulo Bluetooth BLE V4.0 HM-10:** Este módulo é utilizado para transmitir os dados de temperatura para um smartphone, permitindo o monitoramento remoto.

B. *Configuração Inicial*

- **Placa de desenvolvimento - BluePill:**
  “Fig. 2” Pinagem da Placa de desenvolvimento Bluepill

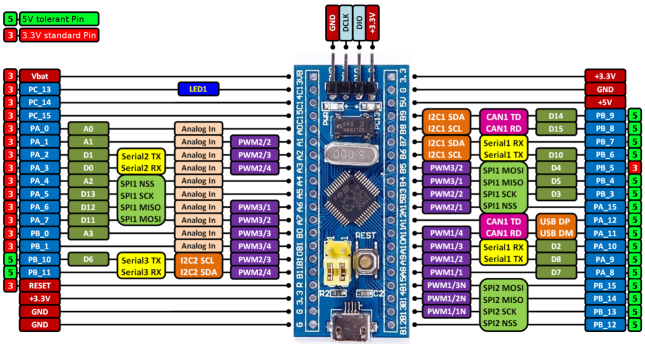

Fig. 2. Descrição da imagem. Fonte: [1]

A placa de desenvolvimento Bluepill, que possui o microcontrolador STM32F103C8T6, desempenha um papel central no projeto, sendo responsável pelo monitoramento e controle dos sensores e periféricos utilizados. Nesta aplicação, utilizaremos alguns pinos como entradas e saídas de sinais, sejam eles digitais (periféricos) ou analógicos (sensores). Para isso, é essencial consultar o esquema da Figura 1, que mostra a funcionalidade dos pinos, permitindo a escolha adequada conforme a necessidade da aplicação.
Por exemplo, será necessário utilizar entradas que suportem comunicação UART para a conexão com o módulo Bluetooth. Portanto, é importante verificar quais pinos são habilitados para esse tipo de comunicação. Neste projeto, utilizaremos os pinos PA9, atuando como transmissor, e PA10, atuando como receptor, para a comunicação.
Além disso, o uso do ST-Link é fundamental para a programação do microcontrolador STM32F103C8T6 na placa Bluepill. O ST-Link é uma interface de programação e depuração que permite a transferência de código do ambiente de desenvolvimento para o microcontrolador. Ele atua diretamente na Bluepill, conectando-se aos pinos de depuração do microcontrolador, permitindo a gravação do firmware e a depuração do código em tempo real.

“Fig. 3” Conexão entre ST-LINK V2 e placa Bluepill

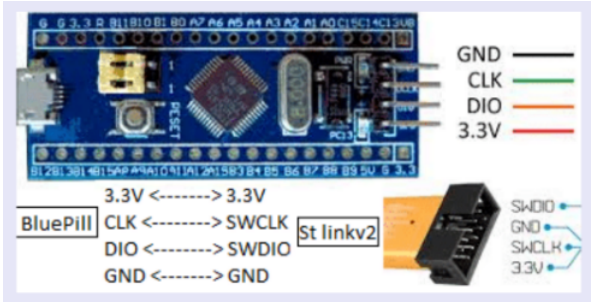

Fig. 3. Descrição da imagem. Fonte: [2]

Um dos motivos pelos quais a programação da Bluepill só pode ser realizada através do ST-Link e não pela entrada USB é que a USB na Bluepill não está configurada como uma interface de programação por padrão. A USB é utilizada principalmente para comunicação serial ou outras funções de interface de usuário. O ST-Link, por outro lado, fornece uma interface dedicada para a programação e depuração, garantindo uma transferência de código mais confiável e um processo de depuração.

- **Shield Multfunções:**
  O Shield Multifunções para Arduino é uma placa acessória que integra diversos componentes, facilitando o desenvolvimento de projetos. Ele integra displays de 7 segmentos, sensores de temperatura e luz, LEDs, botões de pressão, buzzer e um potenciômetro. Esses elementos permitem a exibição de informações, detecção de temperatura e luz, controle de funções,

emissão de sons e ajustes analógicos. Com conectores e pinos de expansão, ele simplifica a prototipagem rápida e a integração de funcionalidades, sendo útil tanto para iniciantes quanto para desenvolvedores mais experientes. No projeto, foram utilizados três componentes de saída da placa de multifunções para Arduino: o buzzer (PB5) e o LED D1(PB15), que servem para alertar o usuário quando um limite preestabelecido é atingido, e o display de 7 segmentos, que exibe os dados obtidos pelo conversor analógico-digital. O buzzer emite um alerta sonoro e o LED D1 acende para indicar que o limite foi ultrapassado, enquanto o display de 7 segmentos apresenta a leitura de tensão ou temperatura, permitindo uma visualização clara dos dados.

“Fig. 4” Placa Shield Multfunções para arduino.

Fig. 4. Descrição da imagem. Fonte: [3]

Mas essa placa foi projetada para ser compatível com a placa Arduino uno R3 e no projeto em questão precisamos trabalhar com a Bluepill, para isso acontecer foi usado durante a disciplina e o projeto a placa µPI, desenvolvida na UFABC, pelo professor João Ranhel [4]. "A placa µPI que é uma placa intermediária, uma espécie de placa-base onde, na parte de baixo, encaixamos o kit STM32F103C8T6 (blue pill). Na parte do topo da placa podemos encaixar qualquer shield Arduino." [5]. Abaixo está uma imagem de como ficou a organização dos dispositivos:

“Fig. 5” Placa µPI com o shield multifunções no soquete de shield Arduino.

Fig. 5. Descrição da imagem. Fonte: [5]

- **Conexão e Configuração do Sensor de Temperatura LM35:**

“Fig. 6” Caracteristicas físicas do sensor de temperatura LM35

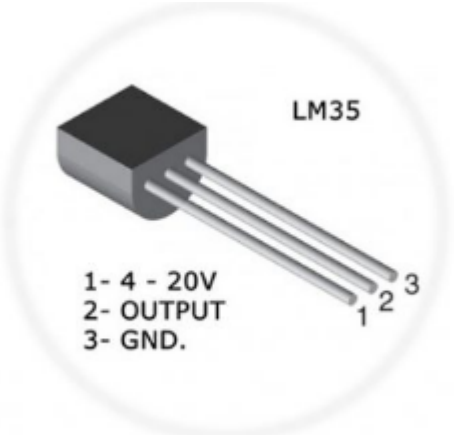

Fig. 6. Descrição da imagem. Fonte: [6]

O sensor LM35 será utilizado para monitorar a temperatura do ambiente. Este sensor envia sinais analógicos para a placa Bluepill, que converte esses dados para o formato digital. Após os cálculos realizados conforme o firmware, a temperatura registrada será exibida no display e enviada para um smartphone via Bluetooth. Veremos abaixo como isso irá funcionar:

1) **Pinos de Conexão:** O sensor LM35 está conectado ao pino PA4 do STM32, nesse pino, como mostra a figura [1], pode ser uma entrada de dados analógicos. A leitura do sensor é feita através do ADC (Conversor Analógico-Digital) do microcontrolador.

"Fig. 7", Esquemático da conexão entre LM35 e Bluepill, por meio da Shield Multifunções.

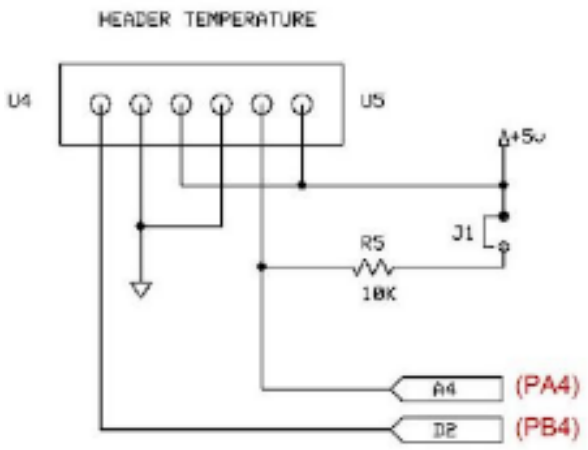

Fig. 7. Descrição da imagem. Fonte: [3]

2) **Configuração do ADC:** O ADC é configurado para ler o valor analógico do sensor e convertê-lo em uma temperatura em graus Celsius. A fórmula utilizada é mostrada na formula 1:

$$\text{TEMPERATURA} = \frac{\text{VALOR.ADC} \times 3300}{4096 \times 100} \quad (1)$$

- **Módulo Bluetooth HM-10 V4.0**
  O módulo Bluetooth HM-10 V4.0 BLE é uma interface de comunicação sem fio eficiente, ideal para projetos de Internet das Coisas (IoT) e dispositivos vestíveis. Operando na frequência de 2,4 GHz, ele permite a troca de dados a curtas distâncias com baixo consumo de energia. O HM-10 é configurável via comandos AT (Estes comandos permitem configurar e monitorar o dispositivo, alterando configurações como a velocidade de comunicação (baud rate), o nome do dispositivo, e o modo de operação (mestre/escravo). Eles também podem ser usados para verificar o status da conexão e outras funções operacionais do módulo) e é compatível com diversos microcontroladores, como a placa Bluepill com o STM32F103C8T6. No projeto, ele se conecta à Bluepill via UART, utilizando pinos como PA9 (TX) e PA10 (RX). Este módulo simplifica a comunicação sem fio em projetos, sendo configurável para modos mestre ou escravo conforme a necessidade.

"Fig. 8" Imagem Módulo Bluetooth HM-10

Fig. 8. Descrição da imagem. Fonte: [7]

Algumas de suas principais caracteristicas:

-Pinos: State, VCC, GND, TXD, RXD, BRK
-CI CC2541 (responsável pelo controle da alimentação do módulo)
-Versão do firmware: V544
-Bluetooth BLE (Bluetooth low energy) V4.0
-Modo de operação: Servidor (Mestre) ou cliente (escravo)
-Comunicação com o microcontrolador: Serial
-Dimensões: 40 x 17 x 3,5mm

1) **Pinos de Conexão:** O header na parte superior esquerda da shield, é onde se pode usar o módulo bluetooth, como mostra a figura [3].

"Fig. 9" Pinos de conexão para HM-10 na Shield.

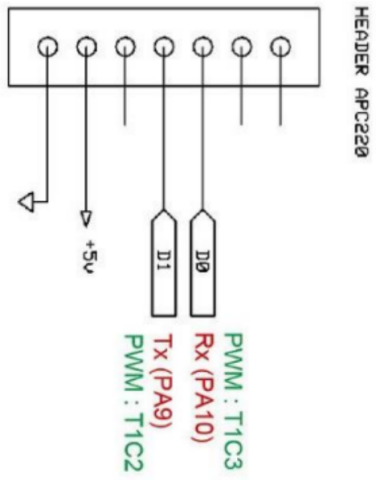

Fig. 9. Descrição da imagem. Fonte: [3]

Com essa facilidade que a shild nos fornece, Para usá-lo, basta conectar os PINOS (TX e RX) do microcontrolador e do módulo HM-10, e com isso fazer a comunicação serial normalmente,

onde o RX: Recepção/Upload (Dados enviados) e o TX: Transmissão/Download (Dados recebidos)

“Fig. 10”. Pinos do Módulo Bluetooth HM-10

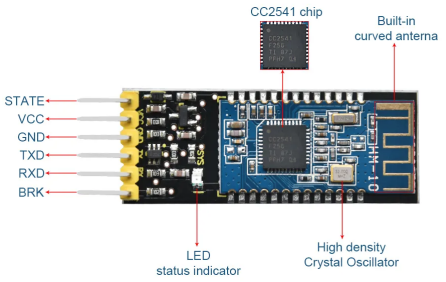

Fig. 10. Descrição da imagem. Fonte: [7]

2) **Comunicação Bluepill e HM-10:** A comunicação entre o módulo HM-10 (que é um módulo Bluetooth BLE) e a placa Bluepill (STM32F103C8) pode ser realizada utilizando o protocolo UART (Universal Asynchronous Receiver-Transmitter). Para precisará configurar o STM32 (Bluepill), atravez da stmcubeide, para comunicar-se com o HM-10 via UART. Para entender melhor veja os esquemáticos abaixo:

“Fig. 11”. Protocolo de comunicação envolvido.

Fig. 11. Descrição da imagem. Fonte: [**?**]

“Fig. 12”. Funcionamento da comunicação UART.

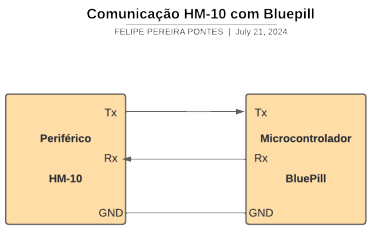

Fig. 12. Descrição da imagem. Fonte: [8]

3) **Configuração do protocolo UART na STMCubeIDE:** Para configurar corretamente o microcontrolador e habilitar a comunicação UART com o módulo HM10, siga estes passos:

a) Inicie a STMCubeIDE e abra o arquivo .IOC do seu projeto.
b) Na interface do STMCubeIDE, localize e selecione a aba "Pinout Configuration".
c) Dentro da aba "Pinout Configuration", encontre a categoria "Connectivity" e selecione a guia correspondente ao "USART1".
d) Na guia "Mode" selecione a opção Asynchronou
e) O *"Baud rate"* Configure para 9600 bits/s. Esta é a taxa de transferência de dados recomendada para o módulo HM-10 [9].

- **Aplicativo para Realizar Conexão Bluetooth com O Smartphone:** O app Bluetooth Serial Terminal, disponível na Google Play Store [10], facilita a comunicação sem fio entre dispositivos Android e módulos Bluetooth serial como HM-10 e HC-05. Ele permite conexões diretas via Bluetooth, suporta diferentes taxas de baud rate e oferece recursos avançados como configuração de terminadores de linha e visualização em tempo real dos dados recebidos. É ideal para projetos de IoT, automação e monitoramento de sensores, proporcionando uma interface amigável para interação eficiente com dispositivos Bluetooth serial pelo seu smartphone ou tablet Android.

“Fig. 13”. Logo do aplicativo Bluetooth Serial Terminal.

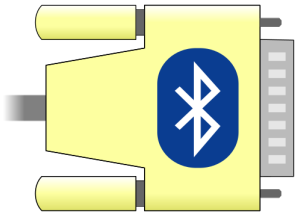

Fig. 13. Descrição da imagem. Fonte: [11]

Para **Configurar aplicativo** é só seguir o passo a passo abaixo:

1) Abrir o app e procurar a guia: Opções (3 barrinhas horizontais, no canto superior esquerdo da tela);
2) Selecionar opção: Devices, e procurar dispositivos disponíveis;

3) Com o bluetooth do smartphone ativado irá aparecer na tela a opção de se conectar a um dispositivo chamado HM. Selecione ele;
4) Após pareado, voltar ao terminal e já será possível estabelecer a conexão (receber ou enviar dados via bluetooth);

C. *FIRMWARE*

O firmware foi desenvolvido inteiramente em linguagem C, utilizando o ambiente de desenvolvimento STMCubeIDE [12] da STMicroelectronics. Este ambiente oferece uma ampla gama de recursos, incluindo uma ferramenta de configuração gráfica, um banco de dados abrangente com todos os modelos de microcontroladores da STMicroelectronics e diversas funcionalidades adicionais. Podemos obter mais informações sobre o Ambiente de desenvolvimento na apostila [5], informaçoes sobre as configurações iniciais, funcionalidades, recursos e importancia da STMCUBEIDE. O firmware é organizado em tasks, cada uma responsável por uma funcionalidade específica do projeto. A seguir, na seção III, apresento uma visão geral das tasks, seguida de subseções detalhadas para cada uma delas, asssim como as principais funções, biblioteca e lógica envolvida no processo

III. Resultados

Nesta seção será apresentado os resultados obtidos ao longo do desenvolvimento do projeto, e uma explicação do processo envolvido em cada etapa.

A. *Hardware*

Nesse tópico será mostrado os resultados finais obtidos em relação a toda parte de hardware do projeto. Será mostrado as conexões, alimentação e o comportamento do sistema de maneira geral.

"Fig. 14".Esquema elétrico das conexões do Hardware.

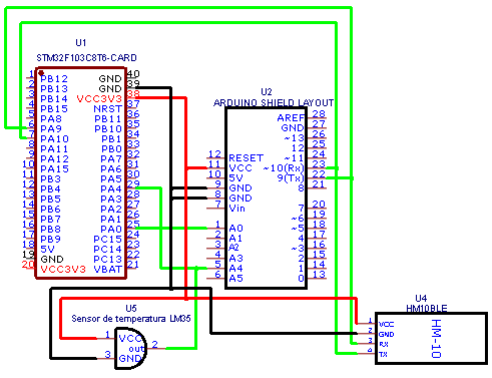

Fig. 14. Descrição da imagem. Fonte: EasyEDA.

Após toda fase de estudo e testes o resultado prático no projeto ficou assim Fig.15:

"Fig. 15". Imagem do Sistema montado e funcionando.

Fig. 15. Descrição da imagem. Fonte: Autoria própria.

B. *Software*

Aqui será mostrado, com texto e imagens, o resultado final dos processos envolvendo o Firmware, como: Passo a passo de como foi feita a configuração do código, fluxograma das funções principais, Código envolvido e outras questões. Os tópicos serão divididos por configurações iniciais e tasks.

- **Configurações iniciais**
  1) **Arquivo .ioc** Antes de iniciar a programação da função principal (main.c), é fundamental configurar o microcontrolador para habilitar as funcionalidades desejadas, como comunicação UART, conversão analógica-digital e outras. Para isso, o ambiente de desenvolvimento STM32CubeIDE oferece uma ferramenta gráfica acessível através do arquivo .ioc, conforme ilustrado na Fig.16.

"Fig. 16". Arquivo .ioc pra configurações do microcontrolador.

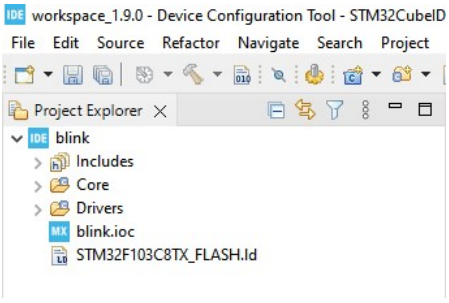

Fig. 16. Descrição da imagem. Fonte: STMCUBEIDE.

O arquivo .ioc será gerado no momento que criar o novo projeto. Após selecionar o Arquivo .ioc, o programador será direcionando a interface de configurações Pinout configuration. Outra informação importante: O arquivo .ioc tambem pode ser gerado em outro software da STMicroelectronics, o STM32CubeMX.
“Fig. 17”. Arquivo .ioc pra configurações do microcontrolador.

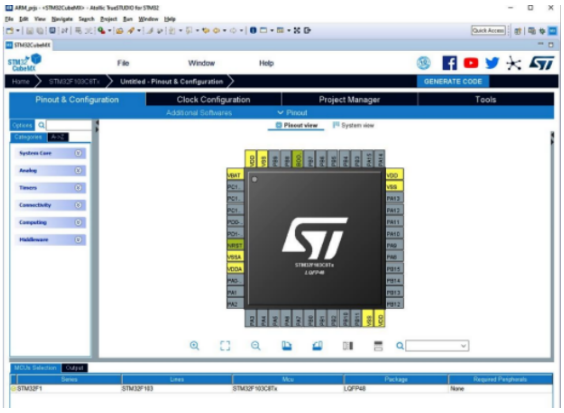

Fig. 17. Descrição da imagem. Fonte: [5].

Nesse momento podemos iniciar as configurações do projeto. O primeiro passo é configurar o clock do microcontrolador. Na guia System Core » RCC » Mode » High speed Clock » Crystal Ceramic e está habilitado o clock do sistema. Isso permitirá que você acesse a aba "Clock Configuration" e lá ajustar o HCLK (High-speed Clock), que é o clock principal do microcontrolador. O HCLK é crucial porque fornece o sinal de clock para a maioria dos periféricos e para o núcleo do processador. Nessa aplicação, foi configurado o HCLK para a frequência 72 MHz, conforme as necessidades do projeto. Veja como ficou na Fig.18.

“Fig. 18”.Configurações do Clock do sistema.

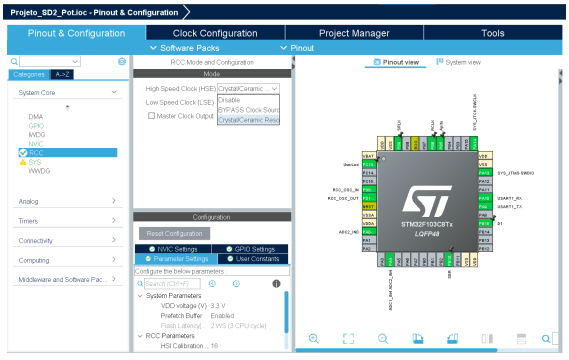

Fig. 18. Descrição da imagem. Fonte: STMCUBEIDE.

O segundo passo é configurar os pinos utilizados no projeto. Para isso, selecione os pinos diretamente na imagem do microcontrolador e defina suas funções, como entradas (input) ou saídas (output). Você também pode atribuir um apelido a cada pino utilizando a opção "User Label" disponível ao clicar com o botão direito do mouse sobre o pino. Neste projeto, todos os pinos foram configurados como saídas (output) e receberam apelidos que refletem suas funções específicas, conforme ilustrado na Fig.19.

“Fig. 19”.Configurações dos pinos.

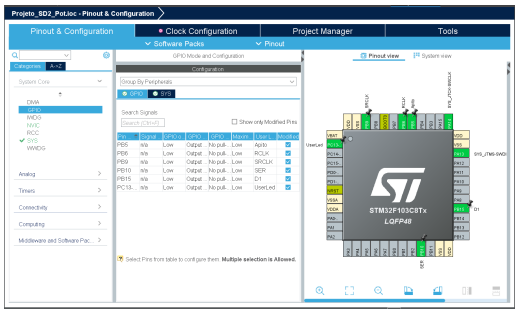

Fig. 19. Descrição da imagem. Fonte: STMCUBEIDE.

Após isso Ainda na guia System Core » SYS » Serial Wire que Habilita o SWD (Serial Wire Debug), para permitir a depuração eficiente com ferramentas como ST-Link. Como mostra a Fig.20.

“Fig. 20”. Habilita o Depurador ST-Link V2.

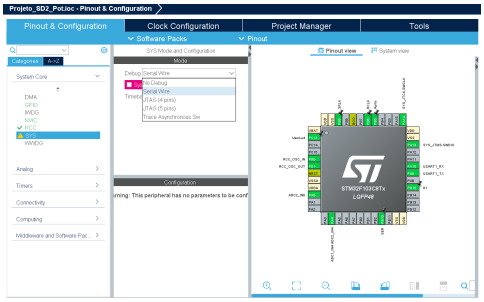

Fig. 20. Descrição da imagem. Fonte: STMCUBEIDE.

O proximo passo é habilitar a comunicação UART, para comunicar o microcontrolador com o módulo HM10 BLE. Para isso Deve-se ira na guia Connectivity » USART1 » Mode » Asynchronous, feito isso na aba Parameter Settings » Basic Parameters » Baud rate » Deve ser digitada a velocidade: 9600 Bits/s ela é a velocidade que deve ser escolhida Baud rate (Velocidade de transmissão de dados).

“Fig. 21”. Habilita a comunicação UART.

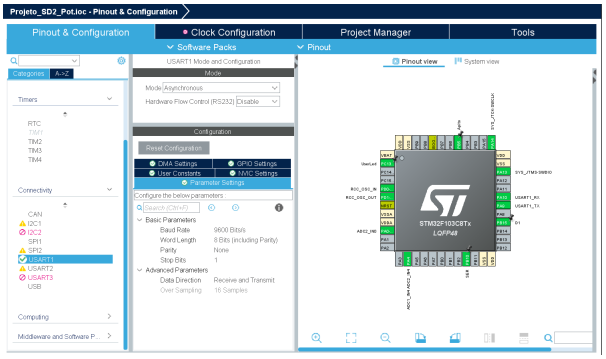

Fig. 21. Descrição da imagem. Fonte: STMCUBEIDE.

É preciso também habilitar a converção do ADC. Para isso, na guia Analog » ADC1 » IN4 (Pino analógico PA4), e ainda na guia Analog » ADC2 » IN0(Pino analógico PA0) e IN4.

“Fig. 22”. Habilita a Conversão Analógica-digital.

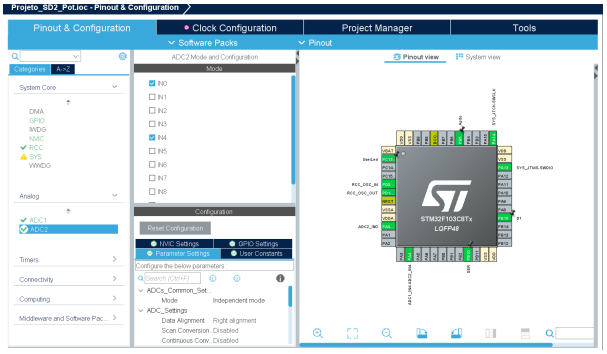

Fig. 22. Descrição da imagem. Fonte: STMCUBEIDE.

Além de tudo, uma das partes mais importante do projeto é a criação das tasks (tarefas) e Queue (fila). Porém, para criar as tasks (tarefas) no seu projeto, siga este caminho: vá para middleware and software » FreeRTOS » interface » CMSISV1 » TASKS AND QUEUES » add, como na Fig.23. Em seguida, insira o nome da task, defina sua prioridade como normal e forneça o nome da função associada a ela. Esse processo garantirá que suas tarefas sejam configuradas corretamente dentro do FreeRTOS, permitindo que o sistema operacional de tempo real gerencie a execução das suas funções conforme a prioridade e o nome que você especificou. Faça o mesmo Procedimento para Criar a Queue conforme Fig.23:

“Fig. 23”. Habilitando FREERTOS e Criando as tasks.

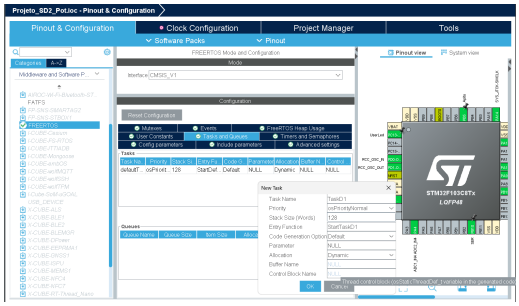

Fig. 23. Descrição da imagem. Fonte: STMCUBEIDE.

“Fig. 24”. Habilitando FREERTOS e Criando Queue.

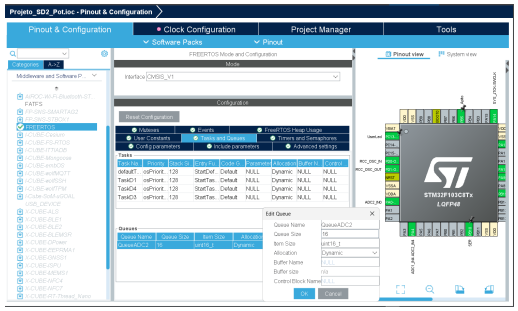

Fig. 24. Descrição da imagem. Fonte: STMCUBEIDE.

Outro Recurso utilizado no FREERTOS foi o uso de Semaforo. Para criar o semaforo "Recurso Compartilhado", como na Fig. 25, o passo a passo é semlhante ao das tasks e queues: middleware and software » FreeRTOS » timers and Semaphores » add, insira o nome.
“Fig. 25”. Habilitando FREERTOS e Criando Semaforo para Recurso Compartilhado.

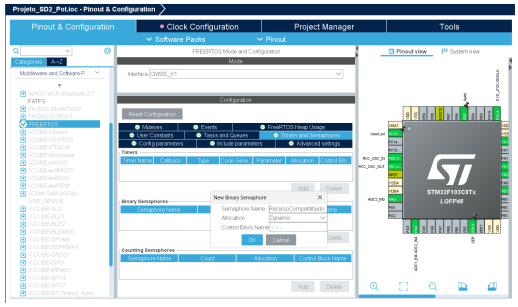

Fig. 25. Descrição da imagem. Fonte: STMCUBEIDE.

Ainda no contexto do FREERTOS, é preciso Habilitar a implementação de atrasos periódicos de maneira eficiente no seu sistema com a "vTaskDelayUntil" para isso: middleware and

software » FreeRTOS » include parameters » vTaskDelayUntil » Enable, como mostra a Fig.26

“Fig. 26”. Habilitando FREERTOS e o uso interrupções com vTaskDelayUntil.

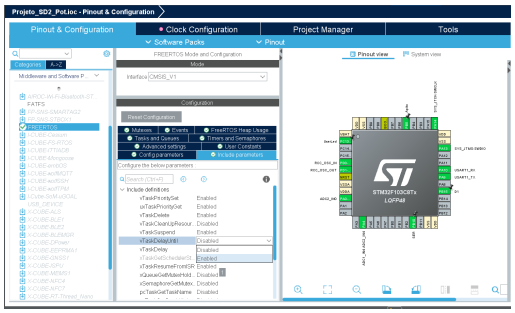

Fig. 26. Descrição da imagem. Fonte: STMCUBEIDE.

Para finalizar as configurações iniciais do arquivo .ioc, veja que o Clock configuration apresenta um erro, devido a configuração de atuação do ADC. Isso deve ser corrigido como mostra a Fig.27, escolhendo a opção /6 em ADC Prescaler.
“Fig. 27”. Reconfigurando o Clock do sistema.

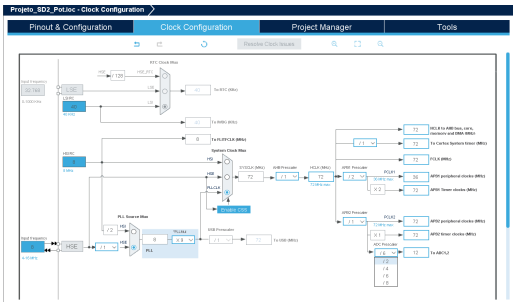

Fig. 27. Descrição da imagem. Fonte: STMCUBEIDE.

2) **Includes, Variáveis e funções**
Sobre os Includes desse código, temos:
<main.h>: Arquivo de cabeçalho principal que geralmente contém declarações de funções e definições globais usadas em main.c.
<cmsis_os.h>: Header do CMSIS-RTOS, que fornece a API para trabalhar com o sistema operacional de tempo real (RTOS) da ARM. Inclui funções para criação e manipulação de threads, semáforos, filas e outros recursos de RTOS.
<mx_prat_05_funcoes.h>: Biblioteca personalizada incluída para funções relacionadas ao display de 7 segmentos (Usada durante a Disciplina).
<stdio.h>: Biblioteca padrão C para entrada e saída. Usada para funções como sprintf.
<string.h>: Biblioteca padrão C para manipulação de strings. Usada para funções como strlen.

```
/* Includes ------------------------*/

#include "main.h"
#include "cmsis_os.h"

/* Private includes ----------------*/
/* USER CODE BEGIN Includes */

#include "mx_prat_05_funcoes.h"
#include <stdio.h>
#include <string.h>

/* USER CODE END Includes */
```

Sobre as variáveis desse código, temos: **int valor_adc**: Armazena o valor lido do ADC. É uma variável global acessada pelas diferentes tasks.
char buf_tx[10]: Buffer para armazenar a string formatada que será transmitida via UART.
int limite: Valor limítro do ADC usado para gerar um alarme se o valor lido exceder esse limite.

```
/* Private typedef -----------------*/
/* USER CODE BEGIN PTD */

int valor_adc = 0;
char buf_tx[10];
int limite = 1500 ;
/* USER CODE END PTD */
```

- **Tasks (Tarefas)**
  1) **DefaultTask** é uma tarefa padrão gerada automaticamente pelo FreeRTOS quando você habilita o suporte a RTOS (Real-Time Operating System) no STM32CubeMX. Nela foi definido que o pino PC13, onde está o led da bluepill, pisque 100ms a cada intervalo de 1s, como mostrado abaixo:

```
void StartDefaultTask(void const
* argument){
/* USER CODE BEGIN 5 */
TickType_t TaskTimeStamp;
TaskTimeStamp =  xTaskGetTickCount();
/* Infinite loop */
for(;;)
{
//MANTER O USERLED PISCANDO 0.1s A CADA 1s:
 HAL_GPIO_WritePin(UserLed_GPIO_Port,
 UserLed_Pin, GPIO_PIN_RESET);
 osDelayUntil(&TaskTimeStamp, 100);
 HAL_GPIO_WritePin(UserLed_GPIO_Port,
```

```
 UserLed_Pin, GPIO_PIN_SET);
 osDelayUntil(&TaskTimeStamp, 900);}
```

Essa task tem o objetivo de manter o usuário ciente que o código está rodando bem. O uso da função osDelayUntil substituindo a função Delay permite uma temporização precisa e regular. Além disso, é uma ferramenta poderosa para sincronização de tarefas em sistemas embarcados, Ela é essencial para aplicações que exigem consistência temporal rigorosa.

2) **TaskD1** Nessa tarefa, é realizada a leitura da variável que armazena os dados do sensor de temperatura/potenciômetro. Se forem detectados dados superiores ao definido pelo desenvolvedor na variável global *limite*, o LED D1 será ligado como forma de alerta.

```
void StartTaskD1(void const * argument)
{
/* USER CODE BEGIN StartTaskD1 */
/* Infinite loop */
for(;;)
{
 //Se a leitura do analógica superar o
 limite o LED D1 criará um alarme:

 if(valor_adc > limite){
 HAL_GPIO_TogglePin(GPIOB,
   GPIO_PIN_15);
 HAL_Delay(500);
 }
 // MANTER O BUZZER DESLIGADO
 ENQUANTO NÃO PASSOU DO LIMITE
  else{
  HAL_GPIO_WritePin(GPIOB,
  GPIO_PIN_15, GPIO_PIN_SET); }
 }
```

3) **TaskD3** Essa Task inicia o código com a conversão do ADC e obtém o valor convertido, que é escalado para representar uma tensão de 0 a 3.3V. Esse valor é formatado em uma string e transmitido via UART1. A cada iteração, de 0.2s do loop, se o valor lido for superior ao limite definido, o buzzer é ativado alternando o estado do pino GPIO. Caso contrário, o pino do buzzer é mantido desligado.

```
void StartTaskD3(void const
* argument)
{
for(;;)
{
HAL_ADC_Start_IT(&hadc2);
valor_adc=HAL_ADC_GetValue(&hadc2);
valor_adc=valor_adc * 3300 / 4095;
sprintf(buf_tx,"%u\r\n",valor_adc);
HAL_UART_Transmit(&huart1,
(uint8_t*)buf_tx, strlen(buf_tx),
HAL_MAX_DELAY);
HAL_Delay(200);

    if(valor_adc > limite) {
      HAL_GPIO_TogglePin(GPIOB,
      GPIO_PIN_5);
      HAL_Delay(250);
 }
    else {
      HAL_GPIO_WritePin(GPIOB,
      GPIO_PIN_5,
      GPIO_PIN_SET);
    }
  }
}
```

Assim como na taskD1, o usuário será alertado quando o valor preestabelecido de *limite* for atingido. No caso de taskD3, a tarefa realiza a leitura do valor do ADC, converte esse valor para uma tensão de 0 a 3.3V e o transmite via UART. Se o valor lido for superior ao limite definido, o buzzer é ativado, alertando o usuário. Caso contrário, o buzzer permanece desligado.

4) **TaskD4** A TaskD4 é uma tarefa que processa e exibe a leitura do valor ADC em um display de 7 segmentos. A tarefa lê mensagens da fila *QueueADC2Handle* para obter o valor mais recente do ADC. Esse valor é convertido para milivolts e dividido em dígitos (unidades, dezenas, centenas e milésimos) para serem exibidos. Cada dígito é enviado para um display de 7 segmentos específico usando o registro de deslocamento 74HC595. Dependendo do valor de cada dígito, a tarefa decide se deve exibir o dígito ou deixar o display apagado. Para o último display, um ponto decimal é ativado se o dígito das unidades for maior que zero. A tarefa continua em um loop infinito, atualizando os displays com os valores calculados.

```
void StartTaskD4(void const
* argument)
{
  int milADC = 0;
  uint16_t ultimoValor = 0;
  osEvent QreadState;
  uint16_t cenADC = 0;
  uint16_t dezADC = 0;
  uint16_t uniADC = 0;
  uint16_t val7seg = 0x00FF;
```

```
  uint16_t serial_data=0x01FF;
  uint32_t miliVolt = 0x0;

  for(;;)
  {
    QreadState = osMessageGet(
    QueueADC2Handle, 0);
    if(QreadState.status ==
    osEventMessage)
    {

ultimoValor = QreadState.value.v;
    }
    miliVolt = (ultimoValor * 3300)
    / 4095;
    uniADC = (miliVolt/1000) % 10;
    dezADC = (miliVolt/100) % 10;
    cenADC = (miliVolt/10) % 10;
    milADC = (miliVolt) % 10;

    serial_data = DISPLAY_1;
    val7seg = conv_7_seg(milADC);
    serial_data |= val7seg;
    serializar(serial_data);

    serial_data = DISPLAY_2;
    if(cenADC > 0 || dezADC > 0
    || uniADC > 0)
    {
      val7seg = conv_7_seg(cenADC);
    }
    else
    {
    val7seg = conv_7_seg(
    DIGITO_APAGADO);
    }
    serial_data |= val7seg;
    serializar(serial_data);

    serial_data = DISPLAY_3;
    if(dezADC > 0 || uniADC > 0)
    {
 val7seg = conv_7_seg(dezADC);
    }
    else
    {
    val7seg = conv_7_seg(
    DIGITO_APAGADO);
    }
    serial_data |= val7seg;
    serializar(serial_data);

    serial_data = DISPLAY_4;
    if(uniADC > 0)
    {
    val7seg = conv_7_seg(uniADC);
    val7seg &= DISPLAY_PONTO;
    }
    else
    {
    val7seg = conv_7_seg(
    DIGITO_APAGADO);
    }
    serial_data |= val7seg;
    serializar(serial_data);

  }
}
```

Então, a TaskD4 exibe os dados no display de 7 segmentos para o usuário acompanhar a tensão medida, seja de um potenciômetro ou de um sensor de temperatura. Esses mesmos dados também são transmitidos via Bluetooth e exibidos na tela do celular através da StartTaskD3. Isso proporciona uma maneira visual direta e uma interface remota para monitoramento dos dados.

"Fig. 28". Fluxograma do funcionamento do código de comunicação com módulo HM-10.

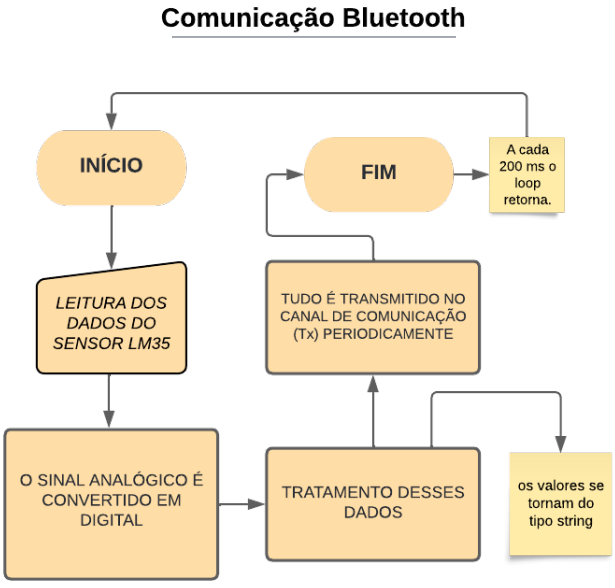

Fig. 28. Descrição da imagem. Fonte: LucidChart.

C. *Conclusão*

O projeto "Medidor de Temperatura Bluetooth", desenvolvido no âmbito da disciplina de Sistemas Digitais 2, teve como objetivo consolidar os conhecimentos adquiridos ao longo do curso, focando na implementação de sistemas embarcados com um sistema operacional de tempo real (RTOS), mais especificamente o FreeRTOS, utilizando a ferramenta STM32CubeIDE.

Durante o desenvolvimento do projeto, foi possível integrar o microcontrolador STM32103c8t6 com o sensor de temperatura LM35 ou o potenciômetro, e o módulo Bluetooth HM-10. A leitura da temperatura e a transmissão dos dados para um smartphone foram alcançadas, no entanto, os dados enviados não são dados confiáveis e precisos, devido a tensão de alimentação incompatível.Os testes realizados mostraram que a eficiencia da coleta de dados do sensor é afetada pelo fato dele está configurado para receber uma tensão de alimentação mínima de 4 volts, e a Bluepill só pode fornecer 3.3 volts.

O STM32CubeIDE se mostrou uma ferramenta essencial para o desenvolvimento, oferecendo um ambiente integrado para configuração do hardware, geração de código e depuração. A comunicação entre o módulo HM-10 e o smartphone foi configurada e testada, comprovando a eficácia da transmissão dos dados em tempo real.

A elaboração de um diagrama de tarefas e um cronograma de execução detalhado contribuiu significativamente para o planejamento e a organização do projeto. Estes elementos garantiram que as etapas fossem executadas dentro dos prazos estabelecidos e permitiram um controle eficaz do progresso.

A robustez da comunicação sem fio, validando a funcionalidade do Bluetooth. O projeto não apenas atingiu seus objetivos, mas também proporcionou uma experiência prática valiosa, reforçando a compreensão dos conceitos teóricos abordados na disciplina. A configuração e o uso do FreeRTOS permitiram a coordenação eficiente das várias tarefas necessárias, como a leitura dos dados do sensor, a comunicação via Bluetooth e a atualização dos displays e indicadores.

Em suma, o projeto demonstrou a eficácia da integração de hardware e software em sistemas embarcados e a utilidade do FreeRTOS para a gestão de tarefas em tempo real. Os resultados obtidos solidificam a base para futuros desafios na área de sistemas digitais e embarcados, evidenciando a aplicabilidade dos conhecimentos adquiridos e a viabilidade de soluções práticas para problemas complexos.

D. *cronograma de Execução*

Para organizar o projeto e evitar sobrecarga de trabalho em um curto período, foi elaborado um diagrama de tarefas utilizando a ferramenta Drawio. Além disso, foi desenvolvido um cronograma de execução, conforme ilustrado na imagem abaixo e detalhado na Tabela 1.

“Fig. 29”. Planejamento de atividades de pesquisa e práticas.

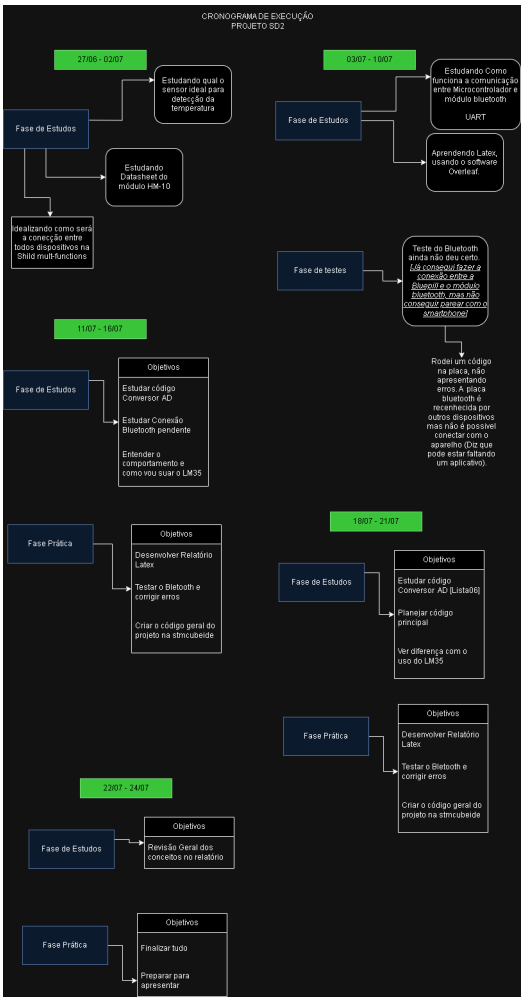

Fig. 29. Descrição da imagem. Fonte: Drawio.

Com essa estrutura, busco garantir que todas as tarefas sejam distribuídas de maneira equilibrada ao longo do tempo, permitindo uma melhor gestão do tempo e aumentando a eficiência na conclusão do projeto. Com isso a execução do projeto se deu como mostra a Tabela I:

TABLE I
INÍCIO: 27/06/2024
FIM: 24/07/2024

CRONOGRAMA DE EXECUÇÃO DO PROJETO				
Semanas	***27/06-03/07***	***04/07-10/07***	***11/07-17/07***	***18/07-24/07***
Estudo do tema	X	X		
Planejamento	X	X		
Pesquisa dos requesitos	X	X	X	
Execução		X	X	X
Testes			X	X
Documentação		X	X	X
Apresentação				X

Cronograma de execução atualizado.

REFERÊNCIAS

[1] armMBED, "Stm32f103c8t6 board, alias bluepill," 2016, Acessado em: 07/2024.

[2] EMBARCADOS, "Stm32cubeide: Primeiros passos e cmsis core com gpio," 2019, Acessado em: 07/2024.

[3] Prof. João Ranhel, *Sistemas Microprocessados*, UFABC, 2019, p.18. Apostila com práticas e foco nos processadores ARM CORTEX.

[4] Prof. João Ranhel, *Sistemas Microprocessados*, UFABC, 2019, Apostila com práticas e foco nos processadores ARM CORTEX.

[5] Prof. João Ranhel, *Sistemas Microprocessados*, UFABC, 2019, p.15. Apostila com práticas e foco nos processadores ARM CORTEX.

[6] HadwareLibre, "Lm35: informações completas sobre este sensor de temperatura," 2024, Acessado em: 06/2024.

[7] MAKER HERO, "Módulo bluetooth ble v4.0 hm-10 keyes," 2024, Disponível em: https://www.makerhero.com/produto/modulo-bluetooth-ble-v4-0-hm-10-keyes/.

[8] HUINFINITO, "Tutorial módulo bluetooth serial hm-10 ble 4.0," 2024, Disponível em: https://www.huinfinito.com.br/home/1550-modulo-bluetooth-serial-hm-10-ble-40-mestreescravo.html.

[9] DSD TECH, "Hm bluetooth module datasheet," 2024, Disponível em: https://www.deshide.com/product-details.html?pid=344835&_t=1665209850.

[10] GOOGLE PLAY, "Serial bluetooth terminal," 2024, Disponível em: https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&hl=pt_BR&pli=1.

[11] SOFTONIC, "Um programa gratuito para android, por kai morich," 2023, Disponível em: https://serial-bluetooth-terminal.softonic.com.br/android.

[12] STMicroelectronics, "Integrated development environment for stm32," 2024, Acessado em: 06/2024.