DCExt Sistemas Embarcados Poli/UPE 2024.2

Tutorial Experimental: Projeto 4 - I2C

Alunos: Denilson José do Bom Jesus Silva de Lima
Rennan de Abreu e Lima Januario Silva
Rodrigo Luiz da Silva
Thales Mateus Lima Evaristo

Recife, Outubro de 2024

SUMÁRIO

1. Introdução

Neste exercício, iremos desenvolver um sistema de comunicação entre cliente e servidor, em que a placa STM NUCLEO-H753ZI atuará como cliente e a placa de sensores PCF8591 como servidor. O usuário irá enviar instruções ao PCF8591 para obter leituras analógicas dos sensores conectados a ele ou para controlar sua saída analógica, dependendo do comando recebido. A interação entre as duas placas será realizada através de um protocolo apropriado, possibilitando ao usuário requisitar informações dos sensores ou enviar valores para controlar a saída analógica do servidor.

Este exemplo prático tem como objetivo analisar a conexão entre aparelhos em um sistema embarcado, empregando a NUCLEO-H753ZI para a comunicação e o PCF8591 para coletar dados dos sensores e regular as saídas analógicas. O repositório desta prática pode ser encontrado em [https://github.com/RodrigoLuizz/DCExt_SE_UPE_poli_2024_2.git], seção Project-4-12C.

2. Requisitos necessários

Para a realização da prática proposta, são necessários os seguintes requisitos:

Placa de Desenvolvimento NUCLEO-H753ZI:

- Modelo: NUCLEO-H753ZI
- Funcionalidade: Atuar como cliente I2C e exibir os dados no terminal virtual via USART3.

Placa PCF8591 (Sensor I2C):

- Modelo: PCF8591
- Funcionalidade: Atuar como servidor I2C, com canais de entrada analógica e saída analógica configurável.

Software de Desenvolvimento STM32CubeIDE (Versão 1.16.1):

- IDE utilizada para o desenvolvimento e depuração do código na placa NUCLEO-H753ZI.

Software STM32CubeMX (Versão 6.12.1):

- Utilizado para configurar o projeto, as interfaces I2C e USART, além das interrupções.

Cabo USB-MicroUSB:

- Utilizado para a comunicação entre a placa NUCLEO-H753ZI e o PC, suportando tanto o carregamento quanto a transferência de dados.

Conhecimentos Básicos

- STM32
- Protocolo de comunicação I2C
- Comunicação serial com UART.

3. Descrição do Hardware

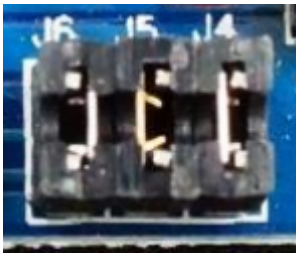
3.1. Hardware do NUCLEO-H753ZI

A placa **NUCLEO-H753ZI** é uma placa de desenvolvimento baseada no microcontrolador STM32H753ZI da STMicroelectronics. Para a atividade, será usado o seguinte recurso:

- **Botão Azul (USER_BUTTON)**: Conectado ao pino **PC13**, utilizado para gerar interrupções EXTI.

3.2. Hardware do PCF8591

A placa **PCF8591** também é uma placa de desenvolvimento baseada no microcontrolador STM32H753ZI da STMicroelectronics. Ela contém interface I2C com 4 entradas e 1 saída analógica. Entre os principais componentes eletrônicos, ela apresenta:



3 Jumpers instalados no módulo que pode se conectar a um LDR integrado, Termistor e Potenciômetro em 3 dessas 4 entradas. Os jumpers usados são:

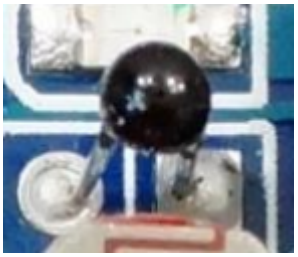
J4 - Conecta Termistor ao canal 1 (AIN1)

J5 - Conecta LDR ao canal 0 (AIN0)

J6 - Conecta Potenciômetro ao canal 3 (AIN3)

Nota-se que o canal 2 (AIN2) não possui jumper, pois não tem propósito alternativo de funcionamento

LDR (Light Detecting Resistor) detecta mudanças básicas na intensidade da luz irradiada sobre ela. Retorna um valor relativo, não a intensidade real da luz emitida. Útil para aplicações como verificar ciclo de dia e noite, se há luz ligada em certos cômodos e etc. Sua saída é transmitida no canal 0.



Termistor é um resistor que muda sua resistência de acordo com a temperatura. Não retorna valores precisos de temperatura, sendo mais útil para detectar variações de temperatura, podendo ser correlacionados a certas temperaturas por teste empírico. Sua saída é transmitida no canal 1.

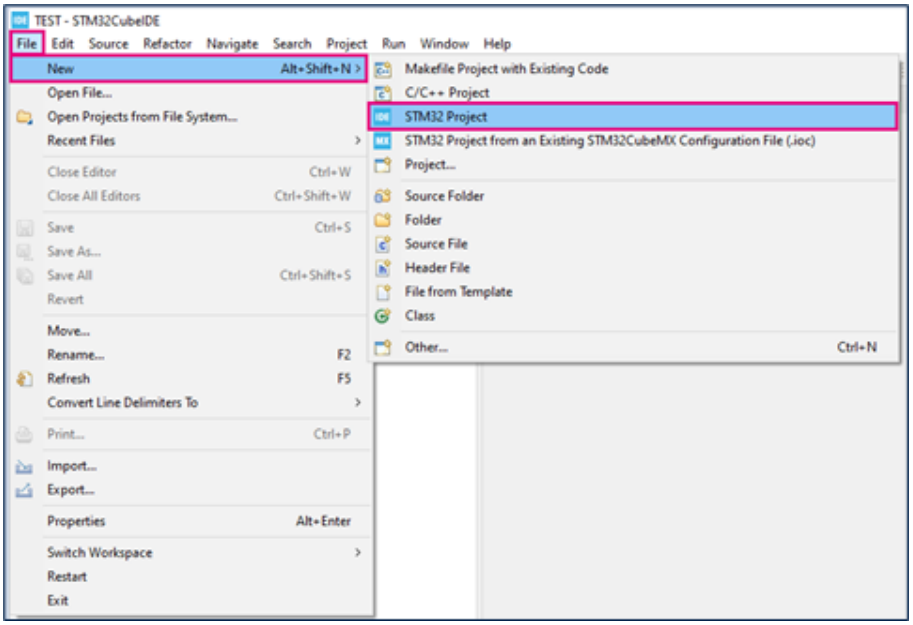
Potenciômetro é conectado entre o Vcc e Gnd, com o wiper conectado ao canal 3.Contém o valor de 10K. Pode ser usado como entrada de controle analógica de propósito geral. Rotacionar o potenciômetro variará a tensão entre 0V e Vcc.

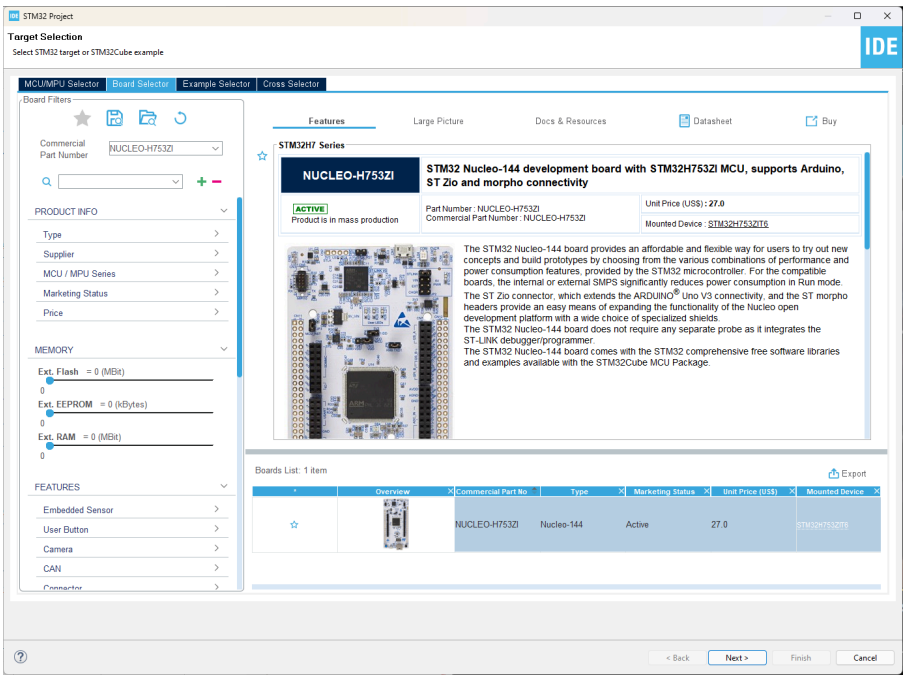
4. Implementação do Código

4.1. Configuração Inicial

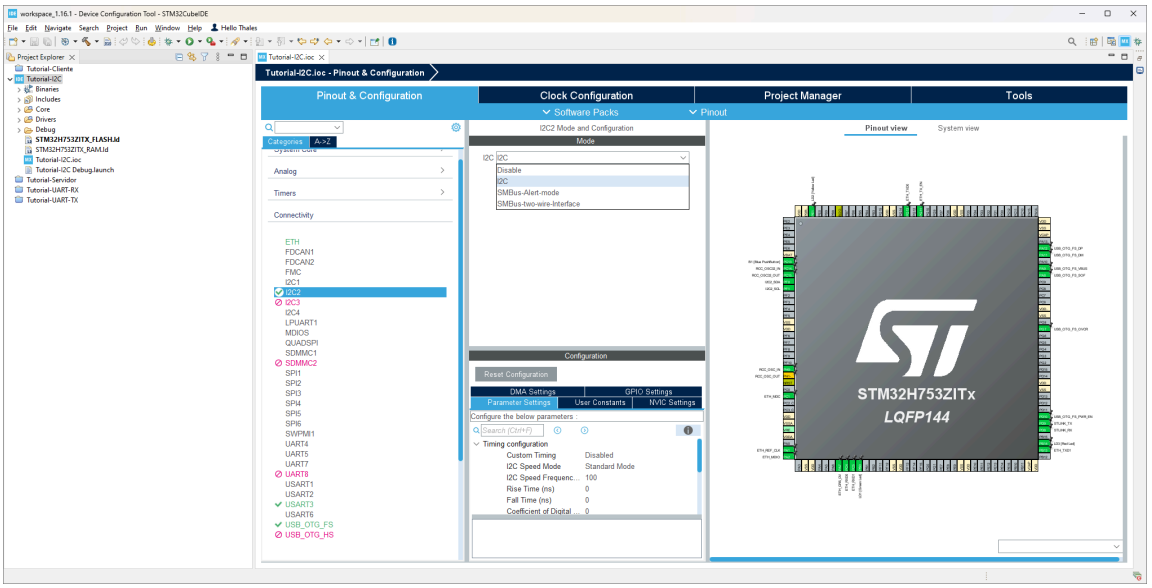
Observação: Em caso de dúvidas de como fazer os próximos passos consulte o primeiro tutorial.

- Abra o **STM32CubeIDE** e crie um novo projeto para a placa **NUCLEO-H753ZI**.

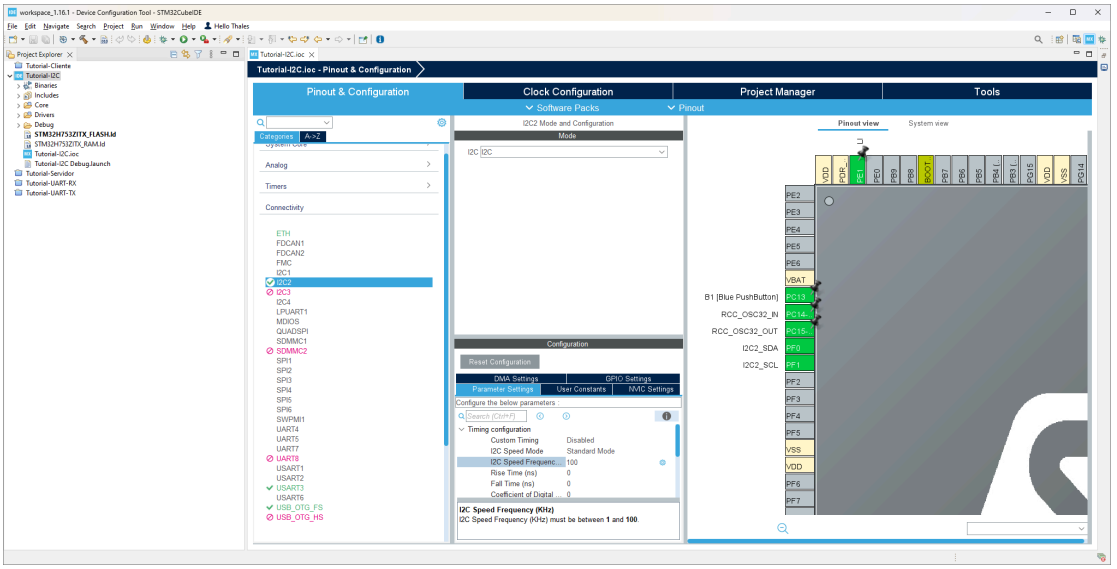




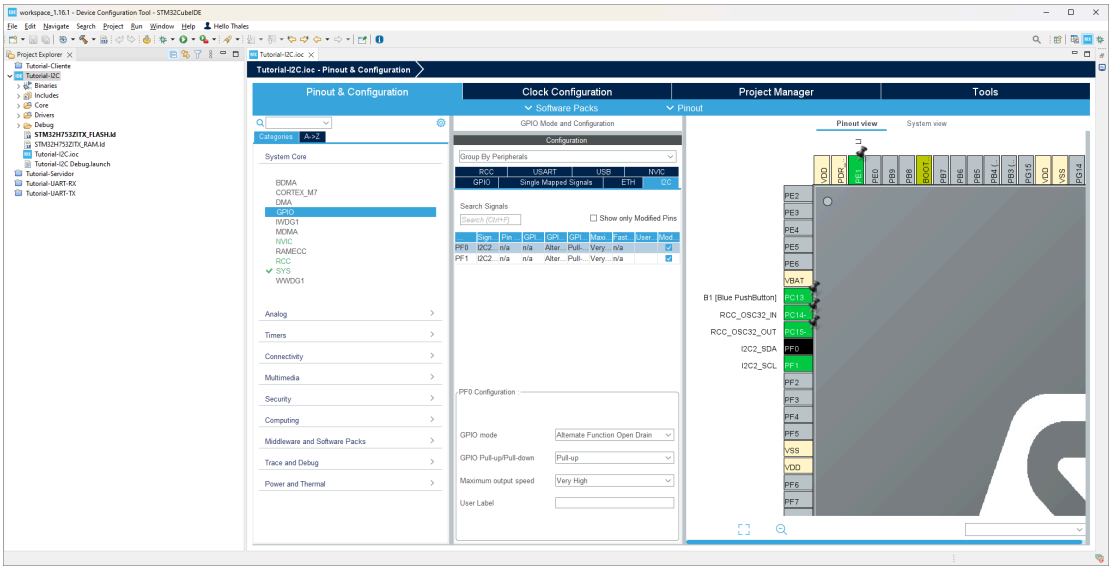
- Habilite o **I2C2,** por padrão serão definidos os pinos **PF0: SDA** e o **PF1:SCL**.



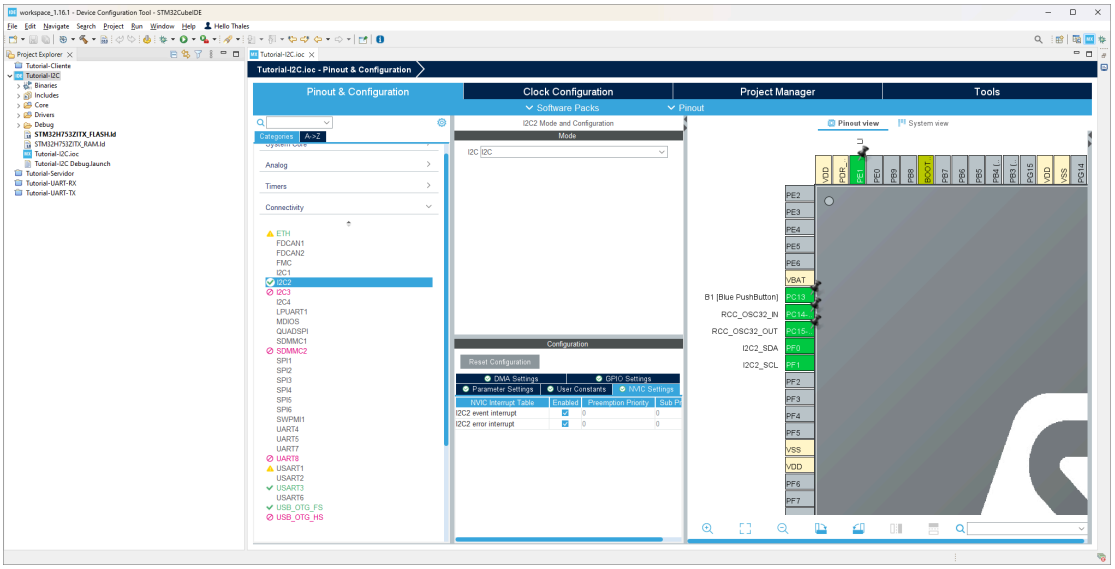
- Defina a **frequência do I2C** para **100Khz**.



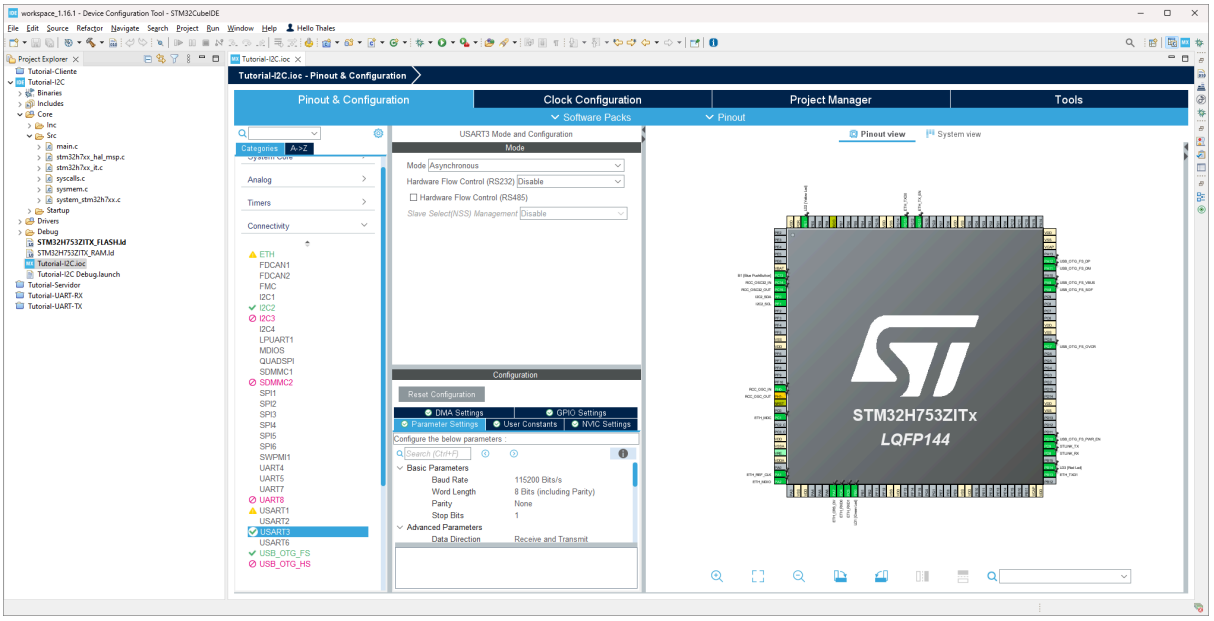
- Na aba de **configurações do GPIO** defina no **GPIO Pull-up/Pull-down** para **Pull-up** para os pinos **PF0** e **PF1**, e coloque a **frequência máxima de saída** de ambos os pinos para o **Very High.**



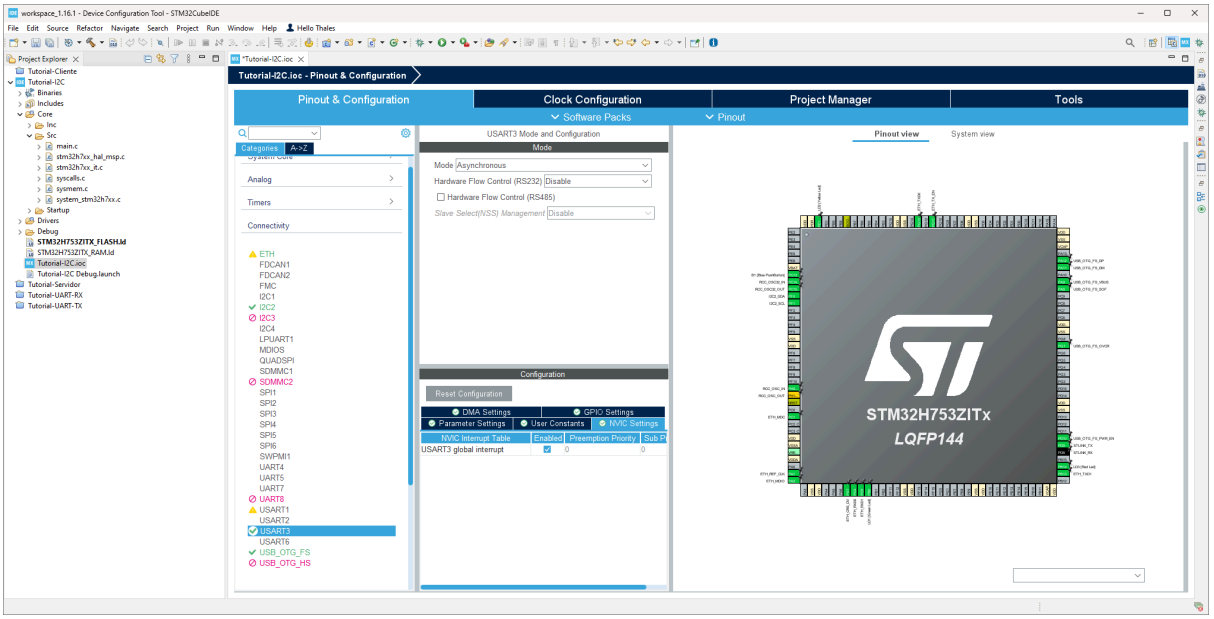
- **Habilite** a **Interrupção** para o **I2C2.**



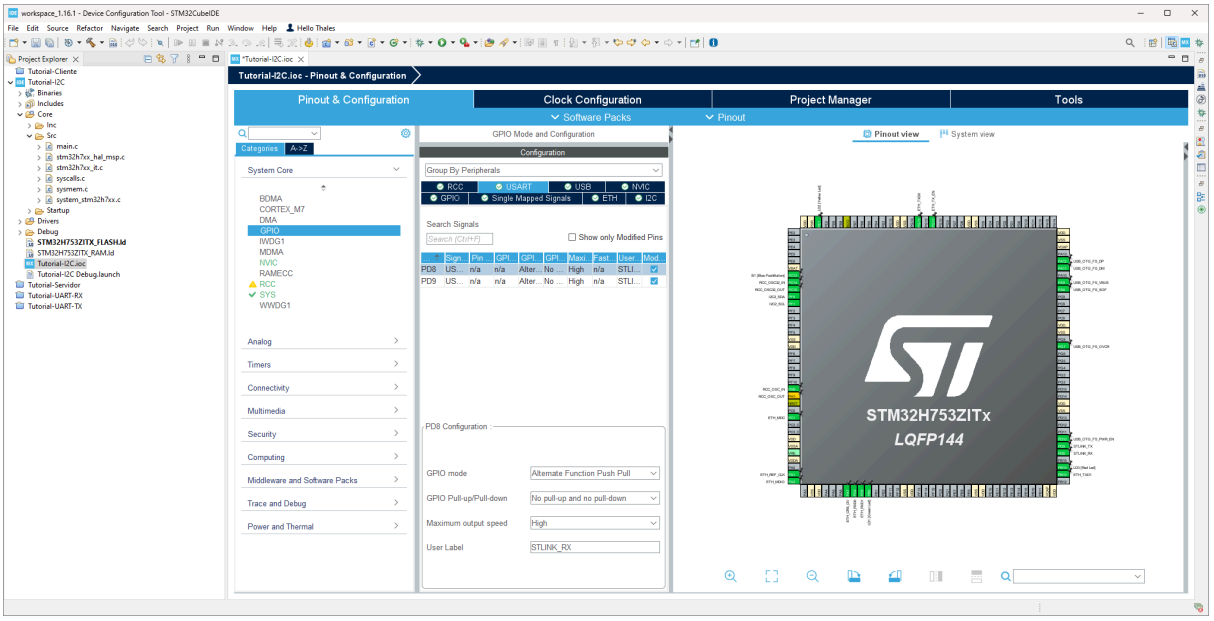
- **Habilite** a **UART3** no modo Assíncrono, e defina os parâmetros de **Baud Rate** como **115200**, o **Comprimento da Palavra** como **8 bits, sem bits de paridade,** e com **1 bit de parada.**



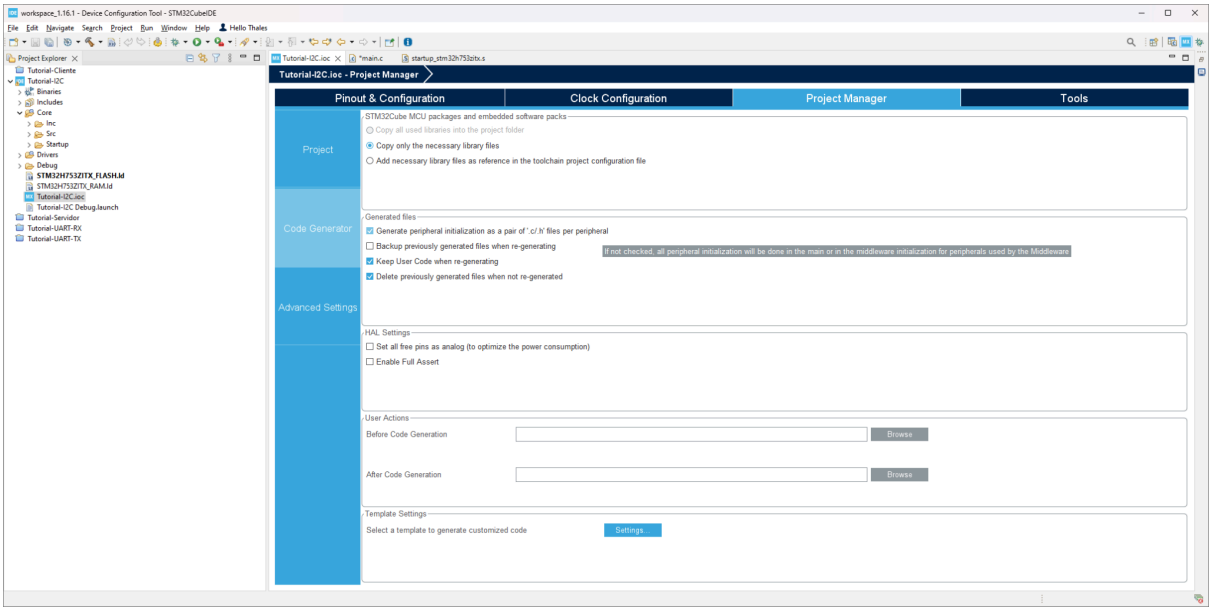
- **Habilite** a **Interrupção** para a **UART3.**



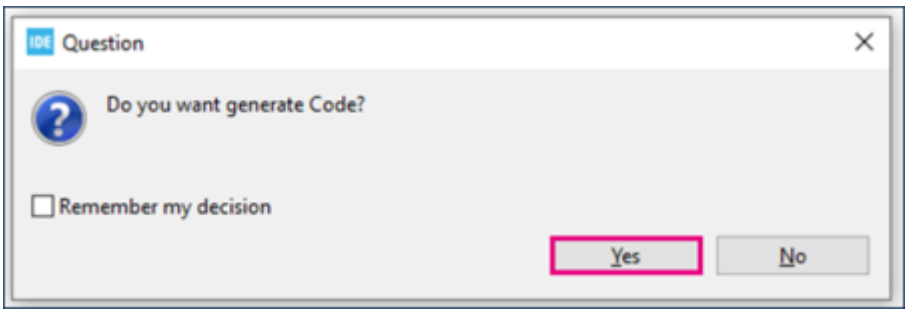
- Na aba de **configurações do GPIO,** defina para os pinos **PF0** e **PF1**, a **frequência máxima de saída** para o **High.**



- **Opcional:** Habilite a geração da inicialização dos periféricos em arquivos .c/.h separados do main (isso reduz a quantidade de código no main, facilitando a visualização do código que será implementado).



- Gere o código base clicando **“Ctrl+S”**.



4.2. Código-Fonte

No código abaixo, as linhas de cor cinza são parte gerada automaticamente pelo STM32CubeMX. As partes destacadas são as linhas adicionadas e/ou importantes para o entendimento das funcionalidades do código. As demais explicações dos trechos em destaque estão escritas em negrito.

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.c
 * @brief          : Main program body
 ******************************************************************************
 * @attention
 *
 * Copyright (c) 2024 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 ******************************************************************************
 */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "eth.h"
#include "i2c.h"
#include "memorymap.h"
#include "usart.h"
#include "usb_otg.h"
#include "gpio.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
// Definindo explicitamente para usar o snprintf
#include <stdio.h>
/* USER CODE END Includes */
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* *
* Esse deslocamento é necessário porque o protocolo I2C usa o endereço em 7 bits,
*  mas o código precisa do endereço em 8 bits para incluir o bit de leitura/escrita.
* */
#define PCF8591_ADDRESS 0x48 << 1  // Endereço do PCF8591 no barramento I2C
/* USER CODE END PD */
/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
/* USER CODE END PV */
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
```

```
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
// Buffers
uint8_t uart_rx_buffer[11];    // Buffer para o comando recebido
uint8_t uart_tx_buffer[20];    // Buffer para a resposta enviada
// Variáveis dos Sensores
uint8_t ldr_sensor = 0;
uint8_t temperature_sensor = 0;
uint8_t potenciometer = 0;
// Variáveis de Dados
uint8_t analog_data[2]; // Buffer para receber dados I2C
uint8_t channel_requested = 0; // Canal solicitado para leitura
void PCF8591_ReadAnalog_IT(uint8_t channel) {
   uint8_t config_byte = 0x40 | (channel & 0x03);
   channel_requested = channel; // Define o canal solicitado
   // Envia o byte de configuração em modo de interrupção
   HAL_I2C_Master_Transmit_IT(&hi2c2, PCF8591_ADDRESS, &config_byte, 1);
}
// Função para definir o valor do DAC no PCF8591
void Set_DAC_Value(uint8_t value) {
    uint8_t dac_command[2] = {0x40, value};  // Comando para o DAC seguido do
valor
         HAL_I2C_Master_Transmit_IT(&hi2c2,   PCF8591_ADDRESS,   dac_command,
sizeof(dac_command));
}
/* USER CODE END 0 */
/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void)
{
 /* USER CODE BEGIN 1 */
 /* USER CODE END 1 */
                                    /*                                    MCU
Configuration--------------------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
 HAL_Init();
 /* USER CODE BEGIN Init */
 /* USER CODE END Init */
 /* Configure the system clock */
 SystemClock_Config();
 /* USER CODE BEGIN SysInit */
 /* USER CODE END SysInit */
 /* Initialize all configured peripherals */
 MX_GPIO_Init();
 MX_ETH_Init();
 MX_USART3_UART_Init();
 MX_USB_OTG_FS_PCD_Init();
 MX_I2C2_Init();
```

```
  /* USER CODE BEGIN 2 */
  // Mensagem inicial para o terminal
   char start_msg[] = "Comandos suportados: Read_AIN0, Read_AIN1, Read_AIN3,
Set_DAC_<valor>\n";
  HAL_UART_Transmit_IT(&huart3, (uint8_t*)start_msg, sizeof(start_msg) - 1);
  // Inicia recepção de dados em modo interrupção
  HAL_UART_Receive_IT(&huart3, uart_rx_buffer, sizeof(uart_rx_buffer));
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
    // Atualiza os valores dos sensores
    PCF8591_ReadAnalog_IT(0); // Lê o canal 0 [LDR]
    HAL_Delay(500);
    PCF8591_ReadAnalog_IT(1); // Lê o canal 1 [TEMP]
    HAL_Delay(500);
    PCF8591_ReadAnalog_IT(3); // Lê o canal 3 [POT]
    HAL_Delay(500); // Delay de 500 ms
    __NOP(); // Apenas para debug da captura dos sensores
  }
  /* USER CODE END 3 */
}
/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  /** Supply configuration update enable
  */
  HAL_PWREx_ConfigSupply(PWR_LDO_SUPPLY);
  /** Configure the main internal regulator output voltage
  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  while(!__HAL_PWR_GET_FLAG(PWR_FLAG_VOSRDY)) {}
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_BYPASS;
  RCC_OscInitStruct.HSIState = RCC_HSI_DIV1;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 24;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  RCC_OscInitStruct.PLL.PLLR = 2;
  RCC_OscInitStruct.PLL.PLLRGE = RCC_PLL1VCIRANGE_3;
  RCC_OscInitStruct.PLL.PLLVCOSEL = RCC_PLL1VCOWIDE;
  RCC_OscInitStruct.PLL.PLLFRACN = 0;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2
                              |RCC_CLOCKTYPE_D3PCLK1|RCC_CLOCKTYPE_D1PCLK1;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.SYSCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB3CLKDivider = RCC_APB3_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_APB1_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_APB2_DIV1;
  RCC_ClkInitStruct.APB4CLKDivider = RCC_APB4_DIV1;
```

```
 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
 {
   Error_Handler();
 }
}
/* USER CODE BEGIN 4 */
// Função de Callback para receber comandos via UART
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    if (huart->Instance == USART3) {  // Verifica se a interrupção veio da
UART3
        // Processa o comando recebido
        if (strncmp((char*)uart_rx_buffer, "Read_AIN0", 9) == 0) {
                snprintf((char*)uart_tx_buffer, sizeof(uart_tx_buffer), "AIN0:
%d\n", ldr_sensor);
        }
        else if (strncmp((char*)uart_rx_buffer, "Read_AIN1", 9) == 0) {
                snprintf((char*)uart_tx_buffer, sizeof(uart_tx_buffer), "AIN1:
%d\n", temperature_sensor);
        }
        else if (strncmp((char*)uart_rx_buffer, "Read_AIN3", 9) == 0) {
                snprintf((char*)uart_tx_buffer, sizeof(uart_tx_buffer), "AIN3:
%d\n", potenciometer);
        }
        else if (strncmp((char*)uart_rx_buffer, "Set_DAC_", 8) == 0) {
             uint8_t value = atoi((char*)&uart_rx_buffer[8]);  // Pega o valor
após "Set_DAC_"
            Set_DAC_Value(value);
              snprintf((char*)uart_tx_buffer, sizeof(uart_tx_buffer), "DAC set
to: %d\n", value);
        } else {
              snprintf((char*)uart_tx_buffer, sizeof(uart_tx_buffer), "Comando
inválido\n");
        }
        // Envia a resposta
                                  HAL_UART_Transmit(&huart3,     uart_tx_buffer,
strlen((char*)uart_tx_buffer), HAL_MAX_DELAY);
        // Reinicia a recepção para novos comandos
        HAL_UART_Receive_IT(&huart3, uart_rx_buffer, sizeof(uart_rx_buffer));
    }
}
void HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef *hi2c) {
    if (hi2c->Instance == I2C2) {
        // Inicia a recepção após a transmissão do byte de configuração
        HAL_I2C_Master_Receive_IT(&hi2c2, PCF8591_ADDRESS, analog_data, 2);
    }
}
void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c) {
    if (hi2c->Instance == I2C2) {
        // Atualiza o valor lido conforme o canal solicitado
        switch(channel_requested) {
            case 0:
                ldr_sensor = analog_data[1];
                break;
            case 1:
                temperature_sensor = analog_data[1];
```

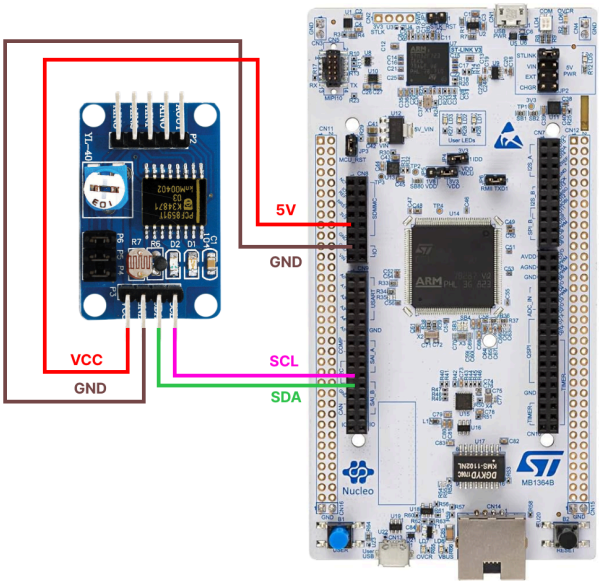
```
                break;
            case 3:
                potenciometer = analog_data[1];
                break;
        }
    }
}
/* USER CODE END 4 */
/**
 * @brief  This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
 /* USER CODE BEGIN Error_Handler_Debug */
 /* User can add his own implementation to report the HAL error return state
*/
//  __disable_irq();
 // Função de tratamento de erro
 while (1) {
     HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0);
     HAL_Delay(250);
 }
 /* USER CODE END Error_Handler_Debug */
}
#ifdef  USE_FULL_ASSERT
/**
 * @brief  Reports the name of the source file and the source line number
 *         where the assert_param error has occurred.
 * @param  file: pointer to the source file name
 * @param  line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
 /* USER CODE BEGIN 6 */
 /* User can add his own implementation to report the file name and line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
 /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

5. Demonstração Experimental

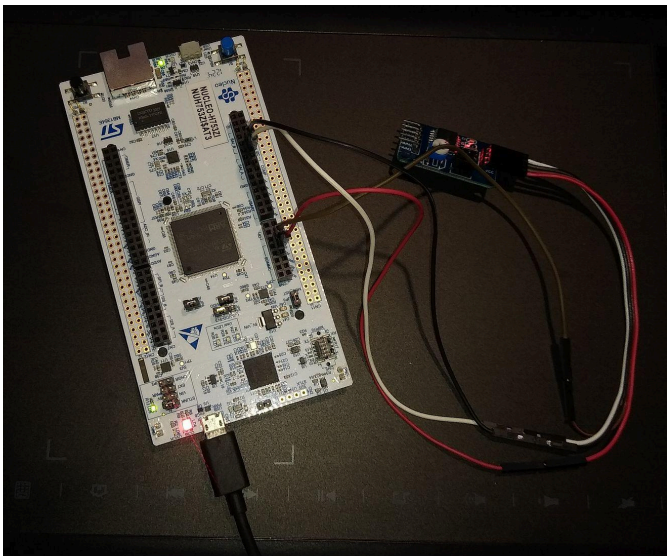
Nesta etapa, vamos explicar como carregar o código na placa **NUCLEO-H753ZI** e verificar se o projeto está funcionando conforme o esperado. Siga o passo a passo abaixo para a execução:

5.1. Conexão e Programação

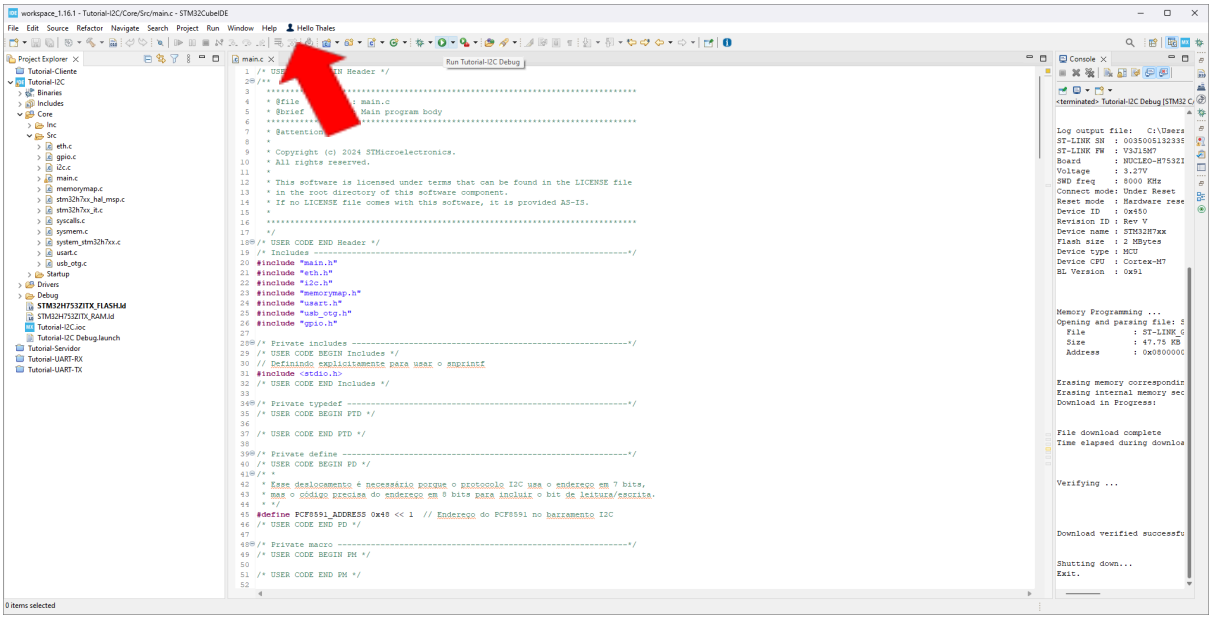
Conecte a Placa PCF8591 a placa NUCLEO H753ZI seguindo o diagrama de conexões abaixo:



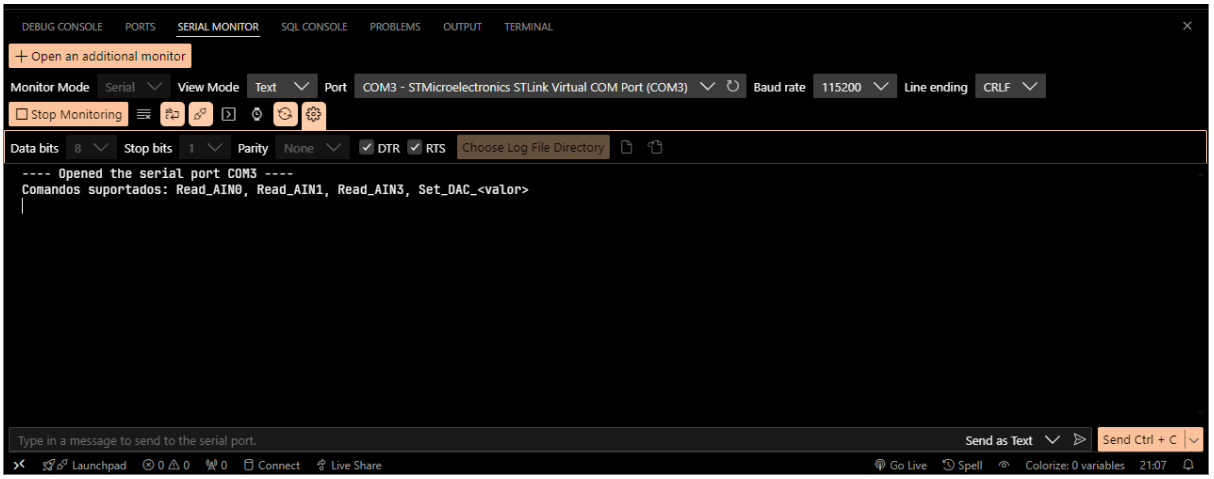
Conecte a placa NUCLEO-H753ZI ao seu computador usando o cabo USB-microUSB, como mostra a imagem abaixo:



Com a placa conectada ao computador, programe-a utilizando o **STM32CubeIDE**. Para isso, clique no ícone conforme mostrado na imagem abaixo:



Para enviar comandos a placa defina no **terminal serial**, o **Baud Rate** de **115200**, com **8 bits de Dados**, **sem bits de paridade**, e com **1 bit de parada**, lembrando de utilizar o **Line ending** como **CRLF** (CR+LF).

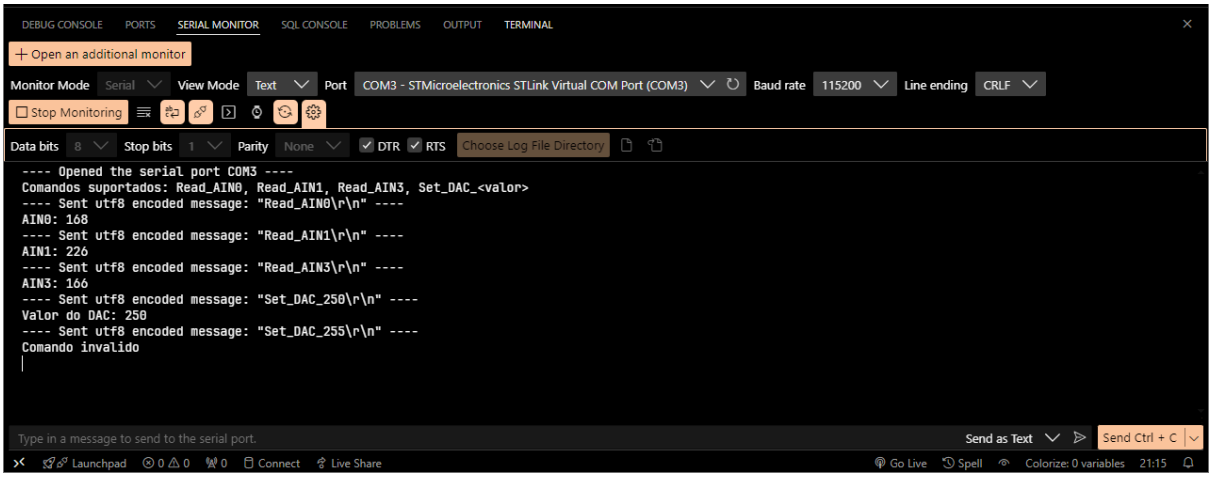


5.2. Análise de Resultados

1. Estado Inicial:
    - Após o upload, a placa envia a mensagem *"Comandos suportados: Read_AIN0, Read_AIN1, Read_AIN3, Set_DAC_<valor>"* ao terminal.

2. Comportamento Esperado:
    - A NUCLEO-H753ZI, atuando como mestre I2C, enviará comandos para o sensor PCF8591 para realizar as seguintes ações:
        - Leitura dos canais analógicos AIN0, AIN1 e AIN3.
        - Controle do valor de saída do DAC, enviando um comando com um valor entre 0 e 255 para ajustar a tensão de saída do PCF8591.
    - Quando a NUCLEO-H753ZI receber os dados do sensor via I2C, os valores serão exibidos no terminal virtual da seguinte maneira:
        - **AIN0**: <valor>
        - **AIN1**: <valor>
        - **AIN3**: <valor>
        - **Valor do DAC**: <valor>
    - A comunicação I2C ocorrerá de forma não bloqueante, ou seja, as transmissões serão feitas através de interrupções, permitindo que a NUCLEO continue processando outras tarefas enquanto aguarda a resposta do sensor.
    - A NUCLEO estará aguardando novos comandos no terminal virtual (USART3) para executar novas leituras ou ajustes no DAC.

3. Comportamento Apresentado:



A NUCLEO falha ao receber o comando para ajustar o valor no DAC.

Observação: Esse comportamento pode ser visto no vídeo postado no repositório deste projeto.

6. Conclusão e Reflexão

Este projeto demonstrou o uso de comandos I2C mediante a placa NUCLEO-H753ZI, junto à placa PCF8591, para replicar o comportamento de uma relação cliente-servidor. Neste sentido, as placas NUCLEO-H753ZI (cliente) e PCF8591 (servidor) se conectam via protocolo I2C, onde o cliente acessa os canais de entrada analógica e controla a saída analógica. A transmissão dos dados dos sensores são feitas por interrupção.

Na placa PCF8591, há três entradas analógicas que podem ser acessadas através dos comandos “Read_AIN0”, “Read_AIN1” e “Read_AIN3” (AIN2, por não ser configurável, não possui jumper). Para a saída analógica, o cliente pode definir um valor com o comando “Set_DAC_{valor}”, para um valor entre 0 e 255 (8 bits).

Os dados então são transmitidos ao computador via UART, via ST LINK, onde é exibido os dados no terminal virtual.

Ao final, o projeto não alcançou completamente o objetivo de controlar e monitorar os sensores de forma eficiente, mas proporcionou uma experiência prática valiosa na implementação de protocolos de comunicação em sistemas embarcados. A aplicação dos conceitos aprendidos permitiu resolver problemas práticos de comunicação e processamento de dados, mostrando como esses sistemas podem ser expandidos para soluções mais complexas envolvendo múltiplos dispositivos e controle em tempo real.

Este exercício demonstrou a importância de um bom entendimento da configuração de periféricos e protocolos de comunicação, além de ressaltar a relevância de práticas como a utilização de interrupções para otimizar a performance de sistemas embarcados.

7. Referências

- **Página de download do STM32CubeMX:**

  STMicroelectronics. (2024). *STM32Cube Initialization Code Generator*. Disponível em: https://www.st.com/en/development-tools/stm32cubemx.html. Acesso em: 29 de setembro de 2024.

- **Página de download do STM32CubeIDE:**

  STMicroelectronics. (2024). *Integrated Development Environment for STM32*. Disponível em: https://www.st.com/en/development-tools/stm32cubeide.html. Acesso em: 29 de setembro de 2024.

- **Especificações do H753**

  STMicroelectronics. (2024). *STM32H743/753*. Disponível em: https://www.st.com/en/microcontrollers-microprocessors/stm32h743-753.html. Acesso em: 29 de setembro de 2024.

- **Especificações do STM32H753ZI**

  STMicroelectronics. (2024). *High-performance and DSP with DP-FPU, Arm Cortex-M7 MCU with 2MBytes of Flash memory, 1MB RAM, 480 MHz CPU, L1 cache, external memory interface, JPEG codec, HW crypto, large set of peripherals*. Disponível em: https://www.st.com/en/microcontrollers-microprocessors/stm32h753zi.html. Acesso em: 29 de setembro de 2024.

- **Funcionamento do I2C no STM32**

  STMicroelectronics. *Getting started with I2C*. Disponível em: https://wiki.st.com/stm32mcu/wiki/Getting_started_with_I2C. Acesso em: 12 nov. 2024.

- **Funcionamento do UART no STM32**

  STMicroelectronics. *Getting started with UART*. Disponível em: https://wiki.st.com/stm32mcu/wiki/Getting_started_with_UART. Acesso em: 12 nov. 2024.