DCExt Sistemas Embarcados Poli/UPE 2024.2

Tutorial Experimental: Controle da cadência de piscar de um LED utilizando EXTI e botão de usuário

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

Nesta prática, vamos explorar a implementação de um sistema utilizando a placa STM NUCLEO-H753ZI, focando no uso do periférico EXTI (External Interrupt) para controlar um LED. Através deste exemplo prático, o LED será programado para piscar com uma frequência variável, onde a velocidade será controlada pelo botão de usuário (botão azul) da placa.

O periférico EXTI permite que interrupções externas disparem eventos, tornando-o ideal para interações com dispositivos como botões. Ao pressionar o botão de usuário, a interrupção será ativada, alterando dinamicamente a frequência de piscar do LED. O repositório desta prática pode ser encontrada em [https://github.com/RodrigoLuizz/DCExt_SE_UPE_poli_2024_2.git], seção Project-2-EXTI.

2. Requisitos necessários

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

- Placa de Desenvolvimento **NUCLEO-H753ZI**;
- Software de desenvolvimento **STM32CubeIDE** (Versão utilizada **1.16.1**);
- Software de desenvolvimento **STM32CubeMX** (Versão utilizada **6.12.1**);
- Cabo **USB-microUSB** (Com suporte a carregamento e transferência de dados)
- **Conhecimentos básicos** em STM32, interrupções (EXTI), e controle de LEDs.

3. Descrição do Hardware

A placa **NUCLEO-H753ZI** é uma placa de desenvolvimento baseada no microcontrolador STM32H753ZI da STMicroelectronics. Ela conta com vários recursos, incluindo:

- **LED Vermelho (LD3)**: Conectado ao pino **PB0** da placa, será utilizado neste projeto para indicar as piscadas controladas pelo botão.
- **Botão Azul (USER_BUTTON)**: Conectado ao pino **PC13**, utilizado para gerar interrupções EXTI.

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**.
- No **STM32CubeMX**, configure o pino **PB0** como **Output** (saída), que será usado para controlar o LED.
- Configure o pino **PC13** como **GPIO_EXTI13**, que ativará uma interrupção ao pressionar o botão azul.
- Gere o código base clicando em **Generate Code**.

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 END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "string.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */
/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables ---------------------------------------------------------*/
#if defined ( __ICCARM__ ) /*!< IAR Compiler */
#pragma location=0x30040000
ETH_DMADescTypeDef  DMARxDscrTab[ETH_RX_DESC_CNT]; /* Ethernet Rx DMA Descriptors */
#pragma location=0x30040060
ETH_DMADescTypeDef  DMATxDscrTab[ETH_TX_DESC_CNT]; /* Ethernet Tx DMA Descriptors */
#elif defined ( __CC_ARM )  /* MDK ARM Compiler */
__attribute__((at(0x30040000))) ETH_DMADescTypeDef  DMARxDscrTab[ETH_RX_DESC_CNT]; /* Ethernet Rx DMA Descriptors */
__attribute__((at(0x30040060))) ETH_DMADescTypeDef  DMATxDscrTab[ETH_TX_DESC_CNT]; /* Ethernet Tx DMA Descriptors */
#elif defined ( __GNUC__ ) /* GNU Compiler */
ETH_DMADescTypeDef DMARxDscrTab[ETH_RX_DESC_CNT] __attribute__((section(".RxDecripSection"))); /* Ethernet Rx DMA Descriptors */
ETH_DMADescTypeDef DMATxDscrTab[ETH_TX_DESC_CNT] __attribute__((section(".TxDecripSection")));   /* Ethernet Tx DMA Descriptors */
#endif
ETH_TxPacketConfig TxConfig;
ETH_HandleTypeDef heth;
UART_HandleTypeDef huart3;
PCD_HandleTypeDef hpcd_USB_OTG_FS;
/* USER CODE BEGIN PV */
/* USER CODE END PV */
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ETH_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_USB_OTG_FS_PCD_Init(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code ---------------------------------------------------------*/
```

Nesta linha, é declarada a variável para o tempo (inicialmente 1000 ms)

```
/* USER CODE BEGIN 0 */
uint32_t timer = 1000;
/* 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();
 /* USER CODE BEGIN 2 */
 /* USER CODE END 2 */
 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {
```

Aqui fica o comportamento padrão do laço, antes da interrupção

```
  /* USER CODE END WHILE */
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET);
                HAL_Delay(timer);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_RESET);
                HAL_Delay(timer);
  /* USER CODE BEGIN 3 */
 }
 /* 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();
  }
}
/**
  * @brief ETH Initialization Function
  * @param None
  * @retval None
  */
static void MX_ETH_Init(void)
{
  /* USER CODE BEGIN ETH_Init 0 */
  /* USER CODE END ETH_Init 0 */
   static uint8_t MACAddr[6];
  /* USER CODE BEGIN ETH_Init 1 */
  /* USER CODE END ETH_Init 1 */
  heth.Instance = ETH;
  MACAddr[0] = 0x00;
  MACAddr[1] = 0x80;
  MACAddr[2] = 0xE1;
  MACAddr[3] = 0x00;
  MACAddr[4] = 0x00;
  MACAddr[5] = 0x00;
  heth.Init.MACAddr = &MACAddr[0];
  heth.Init.MediaInterface = HAL_ETH_RMII_MODE;
  heth.Init.TxDesc = DMATxDscrTab;
  heth.Init.RxDesc = DMARxDscrTab;
  heth.Init.RxBuffLen = 1524;
  /* USER CODE BEGIN MACADDRESS */
  /* USER CODE END MACADDRESS */
  if (HAL_ETH_Init(&heth) != HAL_OK)
  {
    Error_Handler();
  }
  memset(&TxConfig, 0 , sizeof(ETH_TxPacketConfig));
  TxConfig.Attributes = ETH_TX_PACKETS_FEATURES_CSUM | ETH_TX_PACKETS_FEATURES_CRCPAD;
  TxConfig.ChecksumCtrl = ETH_CHECKSUM_IPHDR_PAYLOAD_INSERT_PHDR_CALC;
  TxConfig.CRCPadCtrl = ETH_CRC_PAD_INSERT;
  /* USER CODE BEGIN ETH_Init 2 */
  /* USER CODE END ETH_Init 2 */
}aoao
/**
  * @brief USART3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART3_UART_Init(void)
{
  /* USER CODE BEGIN USART3_Init 0 */
  /* USER CODE END USART3_Init 0 */
  /* USER CODE BEGIN USART3_Init 1 */
  /* USER CODE END USART3_Init 1 */
  huart3.Instance = USART3;
  huart3.Init.BaudRate = 115200;
  huart3.Init.WordLength = UART_WORDLENGTH_8B;
  huart3.Init.StopBits = UART_STOPBITS_1;
  huart3.Init.Parity = UART_PARITY_NONE;
  huart3.Init.Mode = UART_MODE_TX_RX;
  huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart3.Init.OverSampling = UART_OVERSAMPLING_16;
  huart3.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart3.Init.ClockPrescaler = UART_PRESCALER_DIV1;
  huart3.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart3) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_UARTEx_SetTxFifoThreshold(&huart3, UART_TXFIFO_THRESHOLD_1_8) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_UARTEx_SetRxFifoThreshold(&huart3, UART_RXFIFO_THRESHOLD_1_8) != HAL_OK)
  {
    Error_Handler();
```

```
  }
  if (HAL_UARTEx_DisableFifoMode(&huart3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART3_Init 2 */
  /* USER CODE END USART3_Init 2 */
}
/**
  * @brief USB_OTG_FS Initialization Function
  * @param None
  * @retval None
  */
static void MX_USB_OTG_FS_PCD_Init(void)
{
  /* USER CODE BEGIN USB_OTG_FS_Init 0 */
  /* USER CODE END USB_OTG_FS_Init 0 */
  /* USER CODE BEGIN USB_OTG_FS_Init 1 */
  /* USER CODE END USB_OTG_FS_Init 1 */
  hpcd_USB_OTG_FS.Instance = USB_OTG_FS;
  hpcd_USB_OTG_FS.Init.dev_endpoints = 9;
  hpcd_USB_OTG_FS.Init.speed = PCD_SPEED_FULL;
  hpcd_USB_OTG_FS.Init.dma_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.phy_itface = PCD_PHY_EMBEDDED;
  hpcd_USB_OTG_FS.Init.Sof_enable = ENABLE;
  hpcd_USB_OTG_FS.Init.low_power_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.lpm_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.battery_charging_enable = ENABLE;
  hpcd_USB_OTG_FS.Init.vbus_sensing_enable = ENABLE;
  hpcd_USB_OTG_FS.Init.use_dedicated_ep1 = DISABLE;
  if (HAL_PCD_Init(&hpcd_USB_OTG_FS) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USB_OTG_FS_Init 2 */
  /* USER CODE END USB_OTG_FS_Init 2 */
}
/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
 /* USER CODE BEGIN MX_GPIO_Init_1 */
 /* USER CODE END MX_GPIO_Init_1 */
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOG_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|LD3_Pin, GPIO_PIN_RESET);
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(USB_OTG_FS_PWR_EN_GPIO_Port, USB_OTG_FS_PWR_EN_Pin, GPIO_PIN_RESET);
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
  /*Configure GPIO pins : PE2 PE3 PE4 PE5
                           PE6 PE7 PE8 PE9
                           PE10 PE11 PE12 PE13
                           PE14 PE15 PE0 */
  GPIO_InitStruct.Pin = GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5
                        |GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9
                        |GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13
                        |GPIO_PIN_14|GPIO_PIN_15|GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
  /*Configure GPIO pins : PF0 PF1 PF2 PF3
                           PF4 PF5 PF6 PF7
                           PF8 PF9 PF10 PF11
                           PF12 PF13 PF14 PF15 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
                        |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7
                        |GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11
                        |GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
  HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);
  /*Configure GPIO pins : PC0 PC2 PC3 PC6
                           PC7 PC8 PC9 PC10
                           PC11 PC12 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_6
                        |GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10
                        |GPIO_PIN_11|GPIO_PIN_12;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
  /*Configure GPIO pins : PA0 PA3 PA4 PA5
                           PA6 PA10 PA13 PA14
                           PA15 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5
                        |GPIO_PIN_6|GPIO_PIN_10|GPIO_PIN_13|GPIO_PIN_14
                        |GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  /*Configure GPIO pins : PB0 LD3_Pin */
  GPIO_InitStruct.Pin = GPIO_PIN_0|LD3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  /*Configure GPIO pins : PB1 PB2 PB10 PB11
                           PB12 PB15 PB3 PB4
                           PB5 PB6 PB7 PB8
                           PB9 */
  GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_10|GPIO_PIN_11
                        |GPIO_PIN_12|GPIO_PIN_15|GPIO_PIN_3|GPIO_PIN_4
                        |GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8
                        |GPIO_PIN_9;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  /*Configure GPIO pins : PG0 PG1 PG2 PG3
                           PG4 PG5 PG6 PG8
                           PG9 PG10 PG12 PG14
                           PG15 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
                        |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_8
                        |GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_12|GPIO_PIN_14
                        |GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
  /*Configure GPIO pin : USB_OTG_FS_PWR_EN_Pin */
  GPIO_InitStruct.Pin = USB_OTG_FS_PWR_EN_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(USB_OTG_FS_PWR_EN_GPIO_Port, &GPIO_InitStruct);
  /*Configure GPIO pins : PD11 PD12 PD13 PD14
                           PD15 PD0 PD1 PD2
                           PD3 PD4 PD5 PD6
                           PD7 */
  GPIO_InitStruct.Pin = GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14
                        |GPIO_PIN_15|GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2
                        |GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6
                        |GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
  /*Configure GPIO pin : USB_OTG_FS_OVCR_Pin */
  GPIO_InitStruct.Pin = USB_OTG_FS_OVCR_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(USB_OTG_FS_OVCR_GPIO_Port, &GPIO_InitStruct);
  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
 /* USER CODE BEGIN MX_GPIO_Init_2 */
 /* USER CODE END MX_GPIO_Init_2 */
 }
```

Neste trecho, é declarada a função de interrupção chamada ao apertar o botão. Neste caso, ela altera o valor da variável de tempo para 500 ms (1 / 0,5 s = 2 Hz)

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
	if(GPIO_Pin == GPIO_PIN_13) timer = 500;
	else __NOP();
}
/* 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();
 while (1)
 {
 }
 /* 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 */
```

***ATENÇÃO* Não se deve tentar rodar um laço na função de interrupção do GPIO, pois a lógica da função deve ser de tratamento/interação com o usuário, não para "substituir" o laço principal.**

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

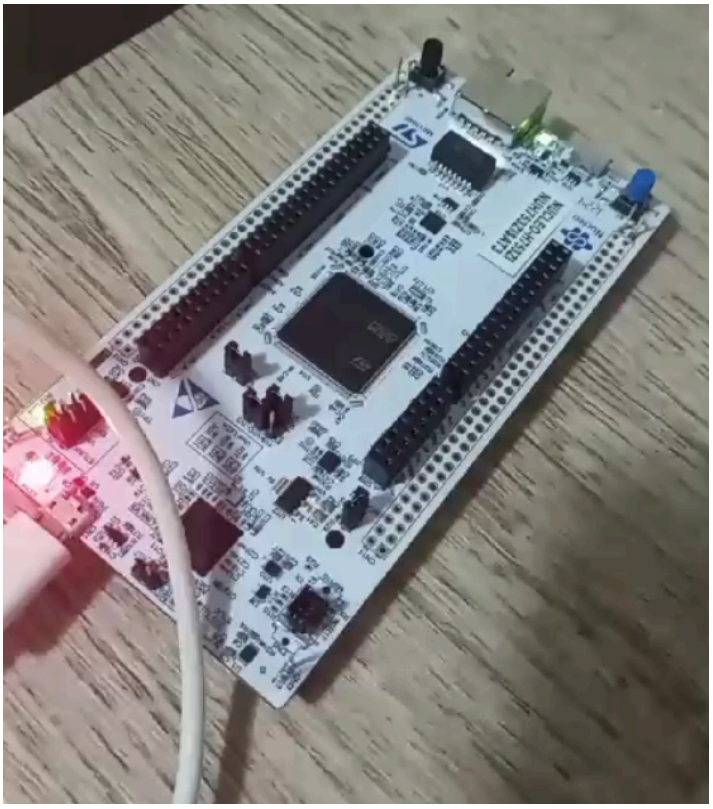

Imagem 01: Placa conectada.

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

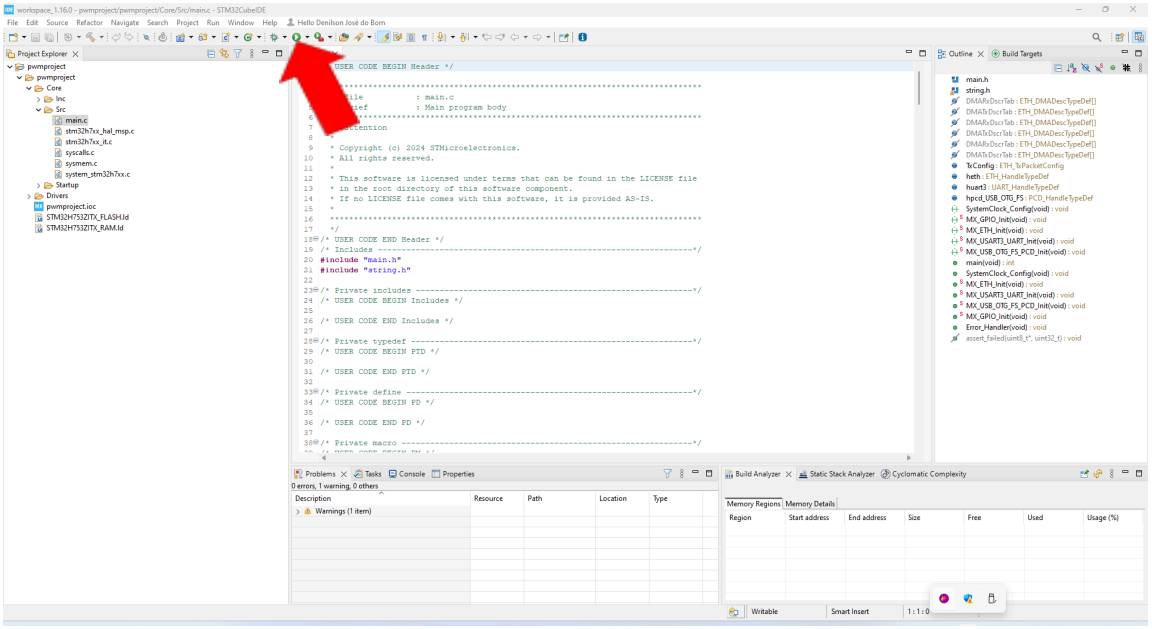

Imagem 02: STM32CubeIDE com o código do projeto aberto.

5.2. Análise de Resultados

1. Estado Inicial:
    - Após o upload, o LED vermelho começará a piscar a uma frequência de **1 Hz**.

2. Comportamento Esperado:
    - O LED vermelho irá piscar inicialmente com uma cadência de **1 Hz** (uma vez por segundo);
    - Quando o botão azul da placa for pressionado, uma interrupção EXTI deve ser gerada;
    - A interrupção deve alterar a cadência do LED para **2 Hz** (duas piscadas por segundo);
    - Cada nova interrupção EXTI gerada pelo botão deve alternar entre a cadência de **1 Hz** e **2 Hz**.

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

6. Conclusão e Reflexão

Este projeto demonstrou o uso de interrupções EXTI em conjunto com a placa NUCLEO-H753ZI para controlar o comportamento de um LED, alternando sua frequência de piscadas por meio de interrupções externas acionadas pelo botão de usuário. A implementação ofereceu uma maneira eficaz de responder a eventos de hardware com baixa latência, permitindo que o LED piscasse em frequências variáveis.

Durante o desenvolvimento, enfrentamos desafios na configuração correta do EXTI para detectar a interrupção do botão, o que ajudou a consolidar conceitos de hardware e software e aplicou o conhecimento teórico aprendido em sala. Esse exemplo pode ser expandido para sistemas mais complexos, envolvendo múltiplos periféricos e diferentes formas de controle.

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.