DCExt Sistemas Embarcados Poli/UPE 2024.2

Tutorial Experimental: Controle de LEDs com PWM por Software em GPIO na Placa STM32H7 - NUCLEO-H753ZI

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, Setembro de 2024

SUMÁRIO

1. Introdução.....3
2. Requisitos necessários.....4
3. Configuração do Ambiente de Desenvolvimento.....5
3.1. Configurando o STM32CubeMX (Versão 6.12.1).....5
3.2. Configurando o STMCubeIDE (Versão 1.16.0).....10
4. Descrição do Hardware.....16
5. Implementação do Código.....17
6. Demonstração Experimental.....24
7. Conclusão e Reflexão.....25
8. Referências.....26

1. Introdução

O projeto experimental intitulado "Controle de LEDs com PWM por Software em GPIO na Placa STM32H7 - NUCLEO-H753ZI" explora a aplicação prática da modulação por largura de pulso (PWM) implementada via software. A placa STM32H7, da série de microcontroladores, foi utilizada para controlar individualmente três LEDs conectados às saídas GPIO, cada um com seu ciclo de trabalho (duty cycle) específico. A técnica de PWM, amplamente utilizada em sistemas embarcados, permite um controle preciso de dispositivos eletrônicos, ajustando a potência média entregue ao componente ao variar os tempos em que o sinal permanece em estado alto (on) e baixo (off).

Este projeto abrange conceitos como controle de hardware por meio de software, gerenciamento de temporizadores e eficiência energética. Através dele, foi possível avaliar a versatilidade do STM32H7 no desenvolvimento de aplicações que demandam controle de tempo crítico e interação direta com periféricos eletrônicos. O repositório desta prática pode ser encontrado em [https://github.com/RodrigoLuizz/DCExt_SE_UPE_poli_2024_2.git], seção Project-1-PWM.

2. Requisitos necessários

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

- **Placa de Desenvolvimento**: NUCLEO-H753ZI;
- STMCubeIDE (Versão utilizada **1.16.1**);
- STM32CubeMX (Versão utilizada **6.12.1**);
- Cabo USB-microUSB (Com suporte a carregamento e transferência de dados)

3. Configuração do Ambiente de Desenvolvimento

Nesta seção, será descrito o procedimento para configurar o ambiente de desenvolvimento. Abaixo estão os passos para a instalação do STM32CubeIDE, STM32CubeMX e a verificação da conexão com a placa NUCLEO-H753ZI.

3.1. Configurando o STM32CubeMX (Versão 6.12.1)

Na página inicial do STM32CubeMX, será necessário fazer login com uma conta ST para acessar todas as funcionalidades. Para isso, clique na parte superior central indicada pela seta vermelha na imagem abaixo. Você terá duas opções: fazer login, se já possuir uma conta, ou criar uma nova conta ST, caso ainda não tenha. Se preferir, pode criar a conta previamente através do link: [Criar Conta na ST]

Imagem 01: Tela Inicial do STM32CubeMX.

Na página inicial do STM32CubeMX, clique na opção indicada pela seta vermelha na imagem abaixo para acessar o seletor de placas.

Imagem 02: Tela Inicial do STM32CubeMX.

Na tela de seleção de placas, utilize a barra de pesquisa à esquerda (indicada pela seta) e digite "H753ZI". O modelo NUCLEO-H753ZI será exibido (apontado pela segunda seta vermelha), conforme mostrado na imagem abaixo.

Imagem 03: Tela de Seleção de Placa.

Na tela de configuração de pinos, usaremos os seguintes pinos: PB0 para o LED Verde, PE1 para o LED Amarelo e PB14 para o LED Vermelho. Eles estão localizados nas posições indicadas pelas setas vermelhas, como mostrado na imagem abaixo.

Imagem 04: Tela de Configuração dos Pinos do STM32CubeMX para a placa NUCLEO-H753ZI.

Na tela de configuração de pinos, ao selecionar um pino, será exibido um menu com as opções de operação para o pino escolhido. No nosso exemplo, utilizaremos o modo GPIO_Output para os pinos PB0, PB14 e PE1, conforme mostrado na imagem abaixo.

Imagem 05: Tela de Configuração dos Pinos do STM32CubeMX para a placa NUCLEO-H753ZI.

Depois de configurar os pinos, acesse a tela de Gerenciamento do Projeto clicando na aba "Project Manager", conforme indicado na imagem abaixo.

Imagem 06: Tela de Configuração dos Pinos do STM32CubeMX para a placa NUCLEO-H753ZI.

Na tela de Gerenciamento do Projeto, insira um nome para o projeto, escolha o diretório de armazenamento e altere o toolchain para STM32CubeIDE, ativando a opção "Generate Under Root", conforme mostrado na imagem abaixo.
Dica: Evite usar caracteres especiais ao definir o nome do projeto e o local de armazenamento, pois algumas IDEs ainda não oferecem suporte total a esses caracteres.

Imagem 07: Tela de Configuração do Projeto no STM32CubeMX.

Após concluir as etapas anteriores, clique no botão "Generate Code", indicado pela seta vermelha na imagem abaixo, para gerar o código.

Imagem 08: Tela de Configuração do Projeto no STM32CubeMX.

Para gerar o projeto, é necessário aceitar os termos da licença. Se desejar consultar os termos, clique no link destacado em azul na tela. Para aceitar, basta marcar a opção indicada pela seta na imagem abaixo.

Imagem 09: Tela de Concordância dos Termos de Licença.

Para concluir esta etapa, clique para abrir o projeto no STMCubeIDE, conforme mostrado na imagem abaixo.

Imagem 10: Tela de Projeto Configurado no STM32CubeMX.

3.2. Configurando o STMCubeIDE (Versão 1.16.0)

Na tela de configuração do workspace, clique em "Launch" para iniciar o STM32CubeIDE, conforme demonstrado na imagem abaixo.

Imagem 11: Tela de Configuração de Workspace do STMCubeIDE.

Com o STM32CubeIDE iniciado, clique em "Yes" para importar o projeto do STM32CubeMX, conforme mostrado na imagem abaixo.

Imagem 12: Tela para importar projeto do STM32CubeMX para o STMCubeIDE.

Clique em "Yes" para confirmar a importação do projeto no STM32CubeIDE, conforme exibido na imagem abaixo.

Imagem 13: Tela de confirmação de importação do projeto do STM32CubeMX para o STMCubeIDE.

Clique em "Ok" para remover arquivos do projeto STMCubeMX no STM32CubeIDE, conforme exibido na imagem abaixo.

Imagem 14: Tela de remoção de arquivos de projeto do STM32CubeMX.

Clique em "Continue" para confirmar a remoção dos arquivos do projeto STM32CubeMX no STMCubeIDE, conforme exibido na imagem abaixo.

Imagem 15: Tela de confirmação de remoção de arquivos de projeto do STM32CubeMX.

Na tela inicial do projeto no STM32CubeIDE, clique no projeto para expandi-lo, conforme indicado pela seta na imagem abaixo.

Imagem 16: Tela de projeto do STMCubeIDE.

Na tela inicial do projeto no STM32CubeIDE, clique na pasta do projeto para expandi-lo, conforme indicado pela seta na imagem abaixo.

Imagem 17: Tela de projeto do STMCubeIDE com projeto expandido.

Na tela inicial do projeto no STM32CubeIDE, clique na pasta “Core” do projeto para expandi-lo, conforme indicado pela seta na imagem abaixo.

Imagem 18: Tela de projeto do STMCubeIDE com pastas do projeto expandido.

Na tela inicial do projeto no STM32CubeIDE, clique na pasta “Src” (Source) do projeto para expandi-lo, conforme indicado pela seta na imagem abaixo.

Imagem 19: Tela de projeto do STMCubeIDE com pastas do projeto expandido.

Na tela inicial do STM32CubeIDE, clique no arquivo “main.c” do projeto para expandi-lo, conforme indicado pela seta na imagem abaixo.

Imagem 20: Tela de projeto do STMCubeIDE com pasta source do projeto expandido.

Estamos no passo final. Sua tela deve se assemelhar à imagem abaixo. Agora, basta seguir as etapas de implementação do código, conforme descrito nas próximas seções.

Imagem 21: Tela de projeto do STMCubeIDE com código main do projeto.

4. Descrição do Hardware

A placa STM32H7 NUCLEO-H753ZI integra o microcontrolador STM32H753 de 32 bits da família ARM Cortex-M7 com recursos aprimorados que combinam alto desempenho com eficiência energética. Operando em até 480 MHz, esta placa oferece capacidade de processamento para aplicações avançadas, como controle em tempo real e processamento de sinais digitais. Possui uma arquitetura onde cada processador possui suporte a instruções FPU e DSP, que o torna muito eficiente no tratamento de dados matemáticos e algoritmos de controle.

O NUCLEO-H753ZI também inclui uma variedade de periféricos, como vários temporizadores, interfaces de comunicação – UART, SPI, I2C e CAN – junto com controladores DMA para otimização de desempenho na transferência de dados sem carga extra no processador principal. A compatibilidade com Arduino é suportada por conectores integrados, sendo possível também agregar outros módulos e sensores. Inclui diversas portas GPIO, que podem ser configuradas com muitas funcionalidades, como entradas e saídas digitais e controle de dispositivos via PWM. O NUCLEO-H753ZI também inclui uma interface de depuração ST-LINK, que pode ser usada para desenvolver e depurar software incorporado na placa.

5. Implementação do Código

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 "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 */
```

Neste trecho, está descrita a função principal utilizada. Deve permanecer entre as tags de 'USER CODE BEGIN' para ser devidamente salvo em caso de alterações na configuração do projeto, que exige que o código seja gerado novamente.

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void software_pwm(uint16_t frequency, uint8_t duty_cycle)
{
```

Captura o estado do botão. Enquanto estiver pressionado, 'button_val' é igual a 1. Do contrário, é igual a 0.

```
    int button_val = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13);
    if(button_val == 1) {
```

Calcula tempo em milisegundos em função da frequência.

```
    uint32_t period_ms = 1000 / frequency;
```

Calcula o período de tempo ligado e desligado em função do duty cycle.

```
    uint32_t on_time = (period_ms * duty_cycle) / 100;
    uint32_t off_time = period_ms - on_time;
```

Liga os LEDs associados ao seu respectivo pino GPIO pelo tempo calculado a partir do duty cycle.

```
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_Delay(on_time);
```

Desliga os LEDs associados ao seu respectivo pino GPIO pelo tempo calculado a partir do duty cycle. Encerra a função e retorna o laço.

```
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_1, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
    HAL_Delay(off_time);
  } else {
```

Caso o botão não esteja pressionado (button_val == 0), os LEDs permanecem desligados

```
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_1, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  }
```

```
}
/* 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 */
```

Neste trecho, há a aplicação da função descrita anteriormente. Esta deve estar em um laço infinito, para que possa ser chamada indefinidamente.

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
            software_pwm(10, 50);
    /* 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 */

}
```

```
/**
  * @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
```

```
*/
```

O trecho abaixo é gerado automaticamente pelo SMT32CubeMX. Ele apenas reseta os pinos de GPIO. Perceba que há dois espaços para adição de código pelo usuário (neste caso, não será necessário adicionar nada).

```
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_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOG_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, LD1_Pin|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 pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD1_Pin LD3_Pin */
  GPIO_InitStruct.Pin = LD1_Pin|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 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 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);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* 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 */
```

6. Demonstração Experimental

6.1. Execução do projeto

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:

6.1.1 Conecte a placa ao computador via cabo USB-microUSB, como mostrado abaixo:

6.1.2 No ***STMCubeIDE***, compile e grave o código na placa.

6.2. Análise de Resultados

Após carregar o código, observe os seguintes comportamentos:

6.2.1 O LED deve piscar conforme o ciclo de trabalho (duty cycle) configurado para o PWM;

6.2.2 Ajuste o código, se necessário, para modificar a frequência ou o ciclo de trabalho do PWM, observando como essas mudanças afetam o comportamento dos LEDs.

7. Conclusão e Reflexão

O projeto permitiu ao grupo aplicar de forma prática os conceitos de PWM e controle de GPIO através de um código que possibilita o controle de LEDs utilizando PWM na placa STM32H7. Com a execução do projeto, e todo o estudo necessário para isso, foi possível entender com mais clareza como lidar com sinais digitais e o controle de suas temporizações.

A maior parte das dificuldades levantadas estava relacionada à precisão do PWM e à sincronização do sinal, que foram resolvidas por meio de alterações no código e estudos acerca do comportamento do sistema. O projeto ajudou a consolidar conceitos que dizem respeito ao funcionamento de hardware e software, e permitiu a aplicação de um conhecimento teórico aprendido em sala.

8. 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.