Guia para a confecção do hardware-in-the-loop do conversor boost

Professora: Vilma A. de Oliveira
Monitor: Deniver R. Schutz

Resumo

Este roteiro tem como objetivo fornecer um guia claro para o desenvolvimento da simulação hardware-in-the-loop (HIL) de um conversor boost controlado por um controlador proporcional integrativo (PI). Para realizar essa simulação, utilizaremos o *board* STM32F407-Discovery e a plataforma Cube-IDE para a programação do mesmo. Este guia é baseado no roteiro de projeto disponibilizado na disciplina SEL0328 - Laboratório de Controle de Sistemas.

Palavras-chave: conversor *Boost*, controle de tensão e corrente, controle em cascata, modulação PWM, *hardware-in-the-loop*.

1 Introdução

A estratégia experimental *hardware-in-the-loop* (HIL) trata-se de um processo de emulação de performance de baixo custo frequentemente utilizada em sistemas onde testes não podem ser realizados com facilidade, capaz se capturar e respostas e dinâmicas do sistema sobre diferentes pontos de operação, sem a necessidade do acionamento do sistema físico real. Além de permitir a avaliação individual de algoritmos, componentes e *softwares* em lógica embarcada [1].

Neste guia, utilizaremos o STM32F407-Discovery para a simulação hardware-in-the-loop (HIL), mas vale ressaltar que a programação pode ser adaptada para outros microcontroladores STM32. Para facilitar a demonstração, utilizaremos a arquitetura de uma única placa, mas a lógica de programação é a mesma para a arquitetura clássica, em que o controlador e o modelo são separados em duas placas distintas.

O software de programação CubeIDE pode ser obtido gratuitamente no site da fabricante `https://www.st.com/en/development-tools/stm32cubeide.html#get-software`.

2 Criando o projeto

Abrindo o CubeIDE crie um novo "STM32 Project". Seguindo os passos da Figura 1 selecione a placa desejada e clique em *"Next"*. Em seguida defina o nome do projeto, clique em *"Finish"* e *"Yes"* conforme a Figura 2.

Em seguida abrirá uma tela para a definição dos periféricos, conforme a Figura 3. Como pode ser observado a inicialização pela board apresentara alguns pinos já ativados devido a periféricos inerentes a placa, como por exemplo o *push botton* conectado ao "PA0"e os lED's interligados as portas "PD12"a "PD15". Nesse momento possuimos duas opções, podemos resetar todos os pinos indo em "Pinout"e em seguida "Clear Pinouts"ou seguir o projeto com os pinos ativados. **Para fins práticos de aprendizado seguiremos com os pinos ativado**s. Caso deseje desativar será necessário configurar os pinos referentes ao cristal oscilador em "RCC".

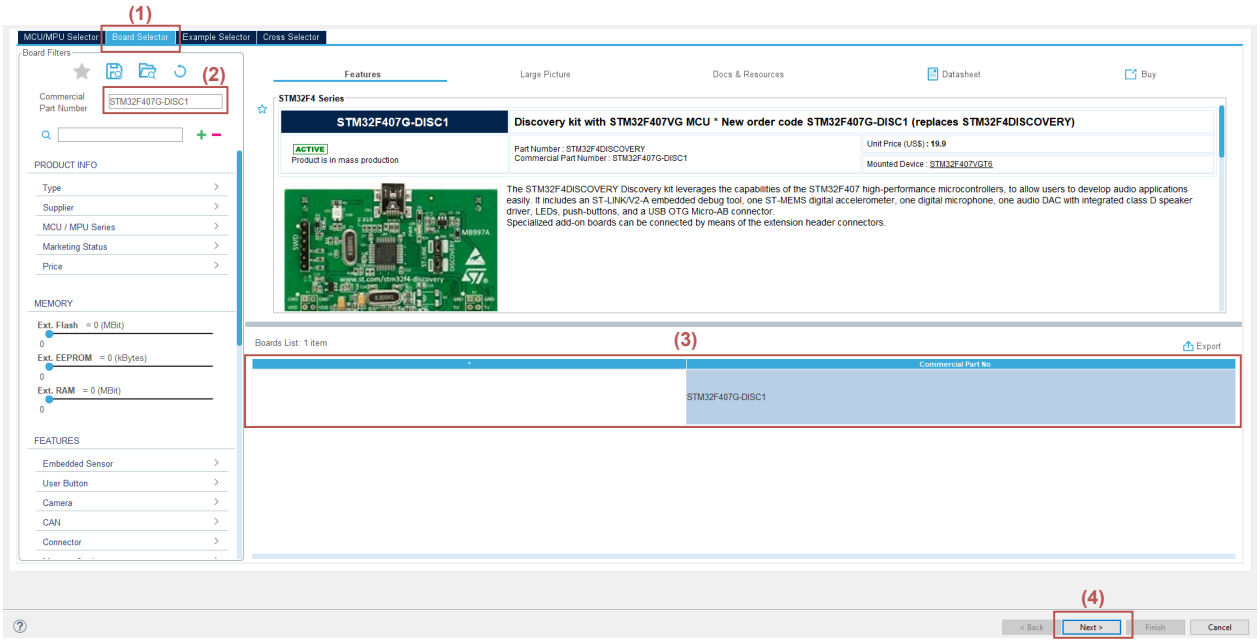

Figura 1: Criando Projeto

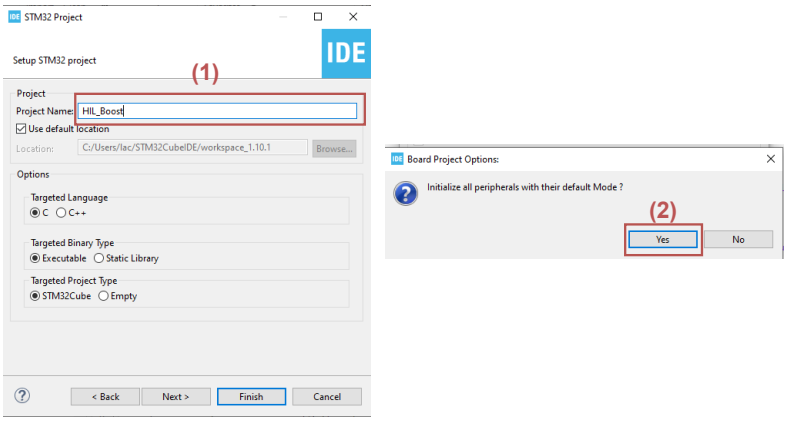

Figura 2: Criando Projeto

3 Definindo os periféricos

Para o módulo de controle precisaremos dos seguintes periféricos:

- Uma Saída PWM de controle e
- Duas entradas analógicas referentes aos valores de tensão e corrente do modelo.

Já para o módulo referente ao modelo do conversor boost precisamos de :

- Uma entrada digital para a leitura do estado do PWM e
- Duas saídas analógicas referentes ao valores de tensão e corrente do modelo.

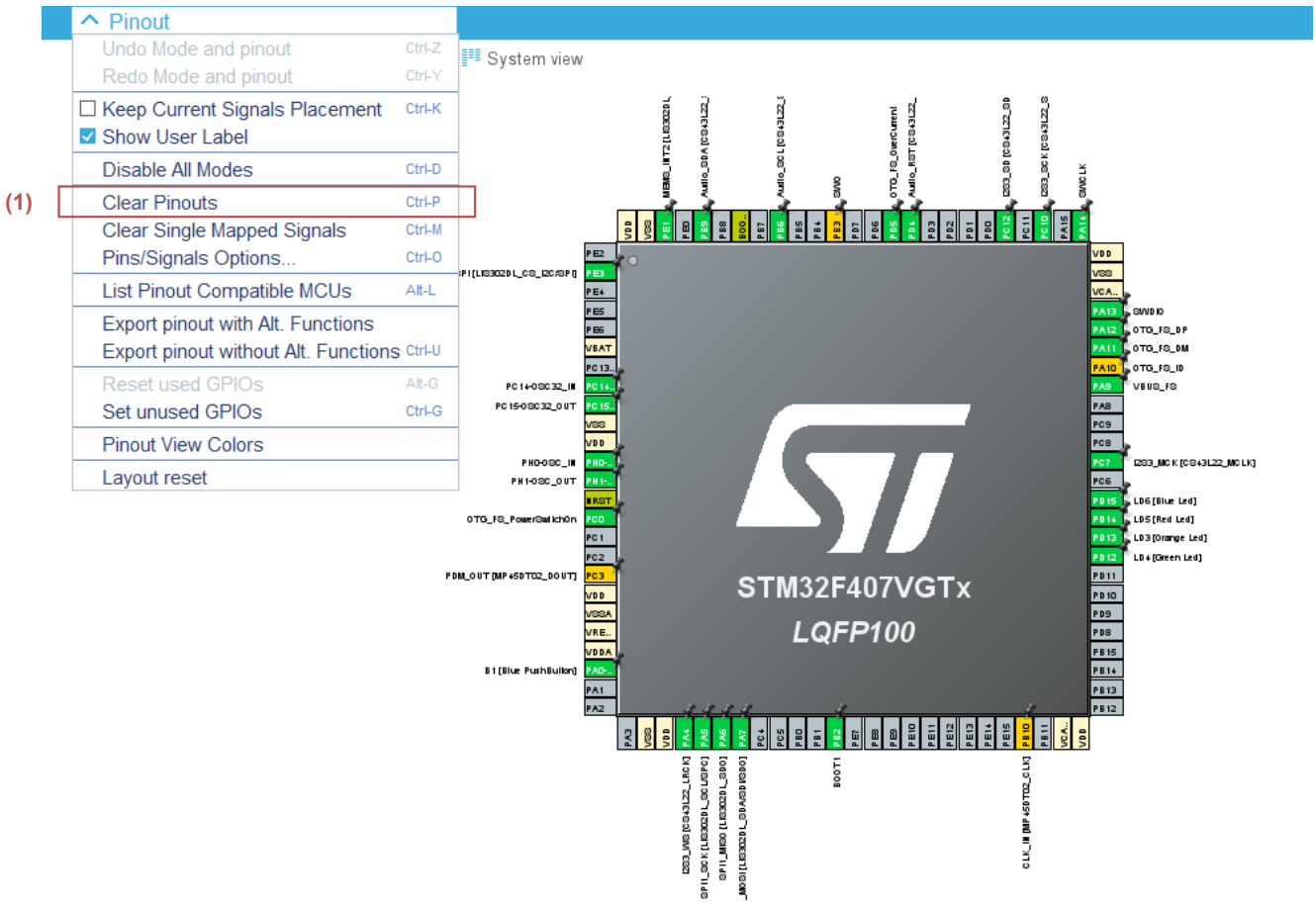

Figura 3: Tela de periféricos

3.1 Gerando o código

Para gerar o código C# a partir do CubeMx, página de configuração dos periféricos, basta seguir os passos da Figura 4. **Toda vez que alterar algo no CubeMx é necessário gerar o código novamente.**

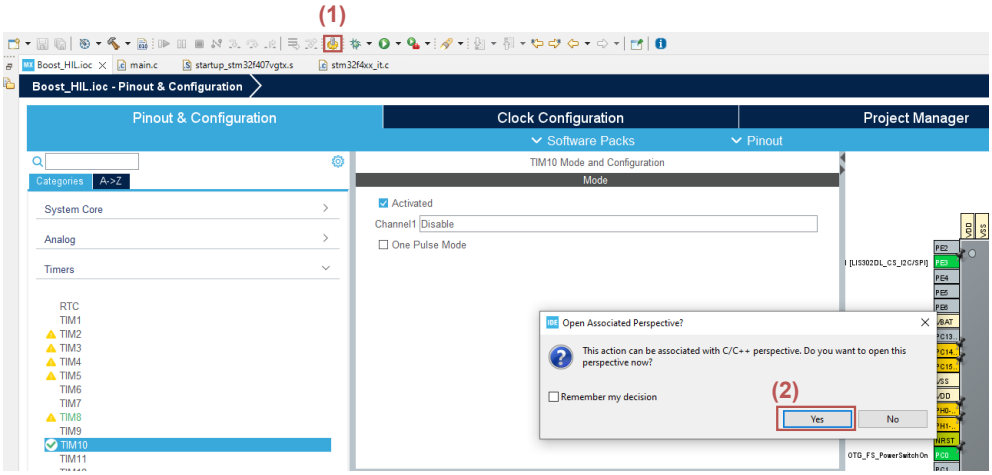

Figura 4: Gerando o código C#

3.2 Configurando a interrupção

Por se tratar de um sistema que opera em tempo discreto, é necessário estabelecer uma rotina de interrupção, conforme ilustrado na Figura 5. **É importante que o código seja escrito dentro dessa rotina, de modo que se respeite a taxa de amostragem a ser estabelecida.**

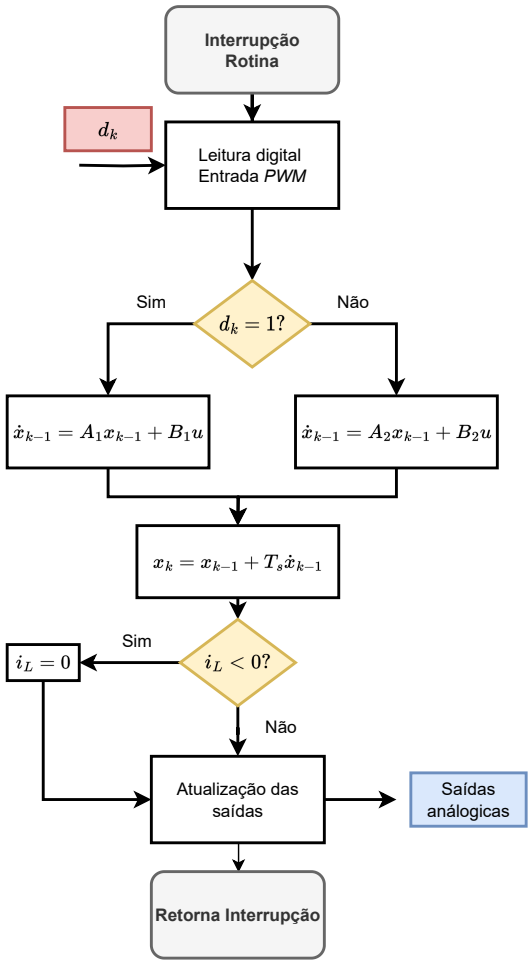

Figura 5: Rotina de interrupção. Adaptado de [2].

A primeira etapa consiste em configurar o clock do sistema. No guia "Configuração do Clock"(Clock Configuration), ajustamos o valor do HCLK para obter a máxima frequência de clock possível. No caso do STM32F407, esse valor é de 168 MHz. É importante que você anote os valores de APB1 e APB2, pois serão necessários para configurar o tempo de interrupção e o sinal PWM. Nesse caso, o APB1 está configurado para 84 MHz e o APB2 para 168 MHz, conforme a Figura 6.

Para gerar a configuração utilizaremos o Timer 10. selecione o *TIM10* e, em seguida, ative o timer conforme indicado na etapa (2) da Figura 7. Em seguida, configuraremos as interrupções para ocorrerem de acordo com a taxa de amostragem $T_s = 5 \times 10^{-6}$ (5 microssegundos). acordo com o datasheet do microcontrolador STM32F405xx [3], o Timer 10 (TIM10) está conectado ao barramento APB2, que anteriormente foi configurado para operar a uma frequência de 168 MHz.

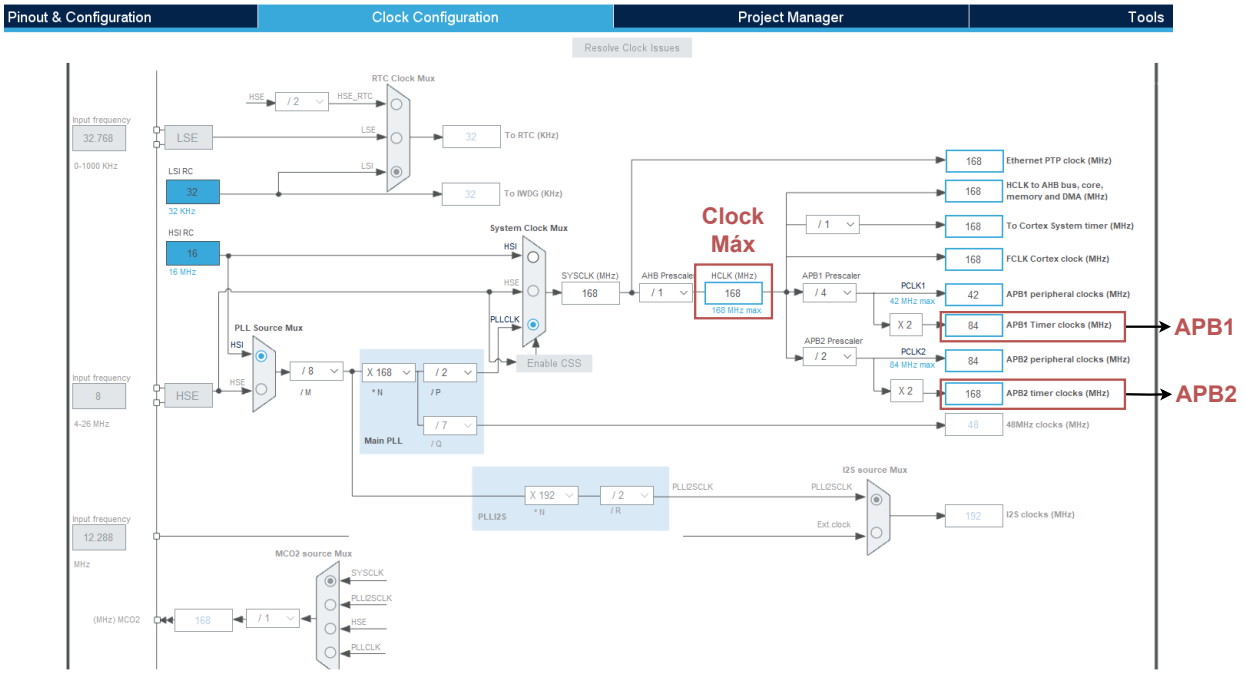

Figura 6: Tela de periféricos

Para configurarmos a interrupção a cada 5 microssegundos ajustamos de acordo com o passo (3) da Figura 7 e seguindo as seguintes equações:

$$Prescaler = \frac{APBx}{Frequência\ de\ Chavemanto} = \frac{168\ MHz}{200\ KHz} = 840 \tag{1}$$

Quando o contador do timer atingir o valor do "auto-reload register"(ARR), será gerada a interrupção. Como a frequência de chaveamento já atinge a taxa de amostragem desejada de $T_s = 5 \times 10^{-6}$, podemos definir o valor de ARR como 1. Nesse ponto, há várias combinações possíveis de ARR e do prescaler que podem atingir a taxa necessária. Cabe ao projetista determinar a melhor configuração.

Por fim, na guia "NVIC Settings"(Configurações do NVIC), habilitamos a interrupção global do TIM10. Dessa forma, o microcontrolador pausará todas as atividades e executará o código da interrupção quando ocorrer a interrupção do TIM10.

A Figura 8 apresenta a localização da função gerado de interrupção. No período configurado o microcontrolador para as demais ações e executa o código aqui apresentado. É essencial colocar esse código entre os comentários indicados por (5), pois, mesmo que seja colocado dentro da função, mas fora dos comentários, há o risco de o código ser apagado caso ocorra uma alteração no CubeMx e um novo código seja gerado.

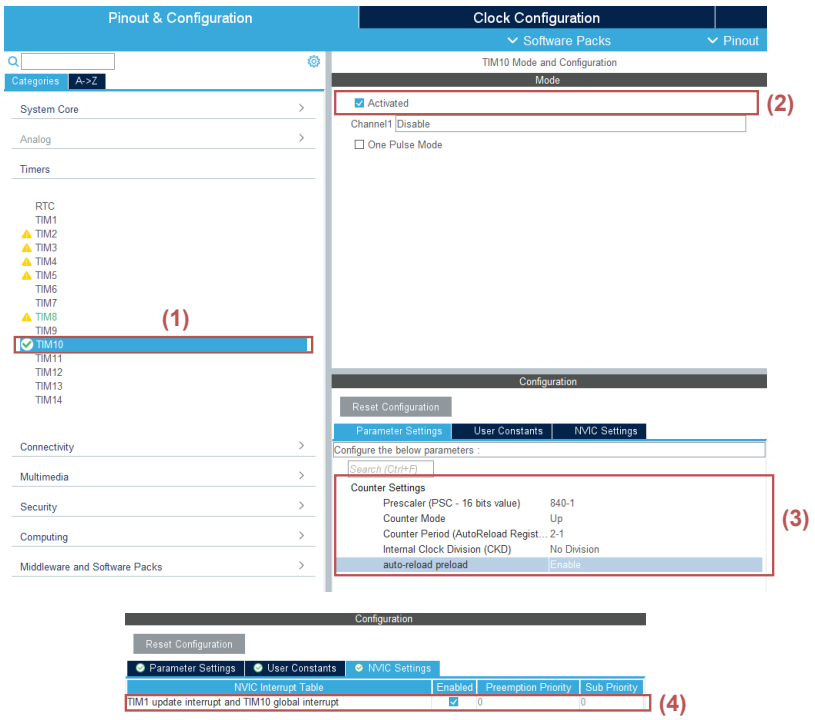

Figura 7: Configuração do Timer

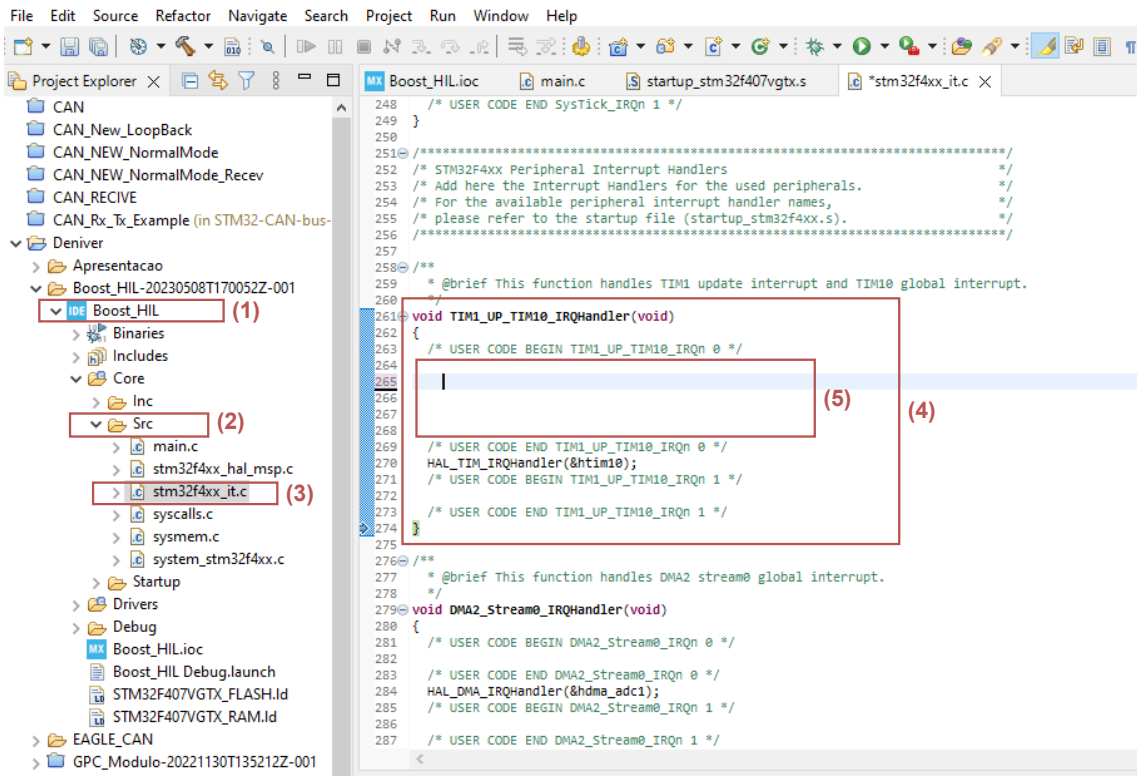

Figura 8: Função geradora de interrupção. (1) Arquivo de projeto gerado a partir do CubeMx, (2) códigos e funções, (3) configuração dos temporizadores, (4) função de interrupção e (5) local para inserir o código

```
void TIM1_UP_TIM10_IRQHandler(void)
 /* USER CODE BEGIN TIM1_UP_TIM10_IRQn 0 */
{

Neste local deve ser inserido o código a ser executado em cada interrupção.

 /* USER CODE END TIM1_UP_TIM10_IRQn 0 */
  HAL_TIM_IRQHandler(&htim10);
  /* USER CODE BEGIN TIM1_UP_TIM10_IRQn 1 */

  /* USER CODE END TIM1_UP_TIM10_IRQn 1 */
}
```

Em *main.c* é importante iniciar a contagem do timer conforme o código:

```
int main(void)
{
  /* USER CODE BEGIN 2 */

  HAL_TIM_Base_Start_IT(&htim10);

 /* USER CODE END 2 */
```

Vale destacar que as variáveis declaras no *main.c* a serem utilizadas dentro da interrupção, devem também serem declaradas no arquivo *stm32f4xxz_it.c* (arquivo de interrupção) como variáveis externas, por exemplo:

```
/* Private variables ---------------------------------------------*/
/* USER CODE BEGIN PV */
extern float Ts;
```

3.3 Configurando a saída PWM

Será necessário configurar uma saída PWM, pois esse sinal será utilizado como o sinal de controle para o nosso controlador PI. Nesse exemplo será utilizado o *TIM8* (timer 8) e uma frequência de PWM de 10 KHz. A Figura 9 apresenta Ativação do periférico como saída PWM. Para esse caso como foi escolhido o canal 4 do TIM8 o pino ativado foi o *PC9*, caso escolha outro canal e/ou outro timer o pino ativado será diferente.

Em seguida tem-se que configurar o timer de modo a geral a frequência estimada em projeto. A configuração é semelhante a já executada no timer de interrupção. Pelo datasheet o TIM8 está conectado ao APB2, logo a configuração do prescaler e do ARR é dado por:

$$Tim\ Clock = \frac{APBx}{Prescaler} = \frac{168\ MHz}{168} = 1\ MHz \tag{2}$$

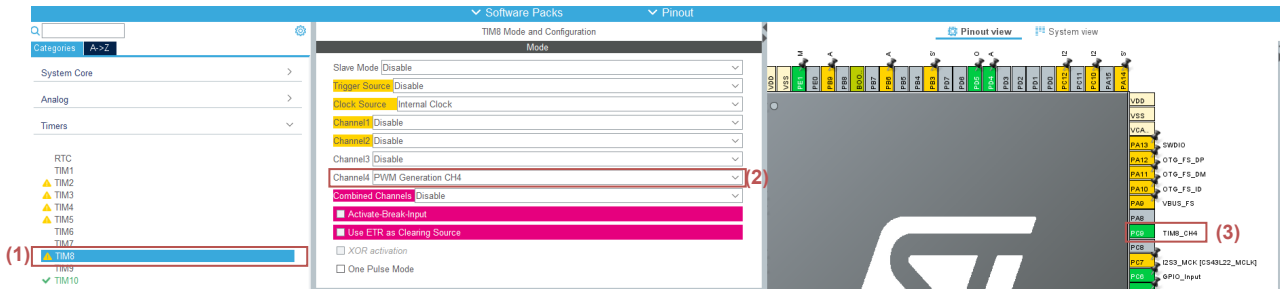

Figura 9: Ativação do periférico como saída PWM

$$Freq = \frac{Tim\ Clock}{ARR} = \frac{1\ MHz}{100} = 10\ KHz \tag{3}$$

É importante destacar o ARR (Auto-Reload Register), pois ele determinará a faixa de duty-cycle a ser adotada. No nosso caso, o duty-cycle será de 0 a 100. A configuração é apresentada na Figura 10.

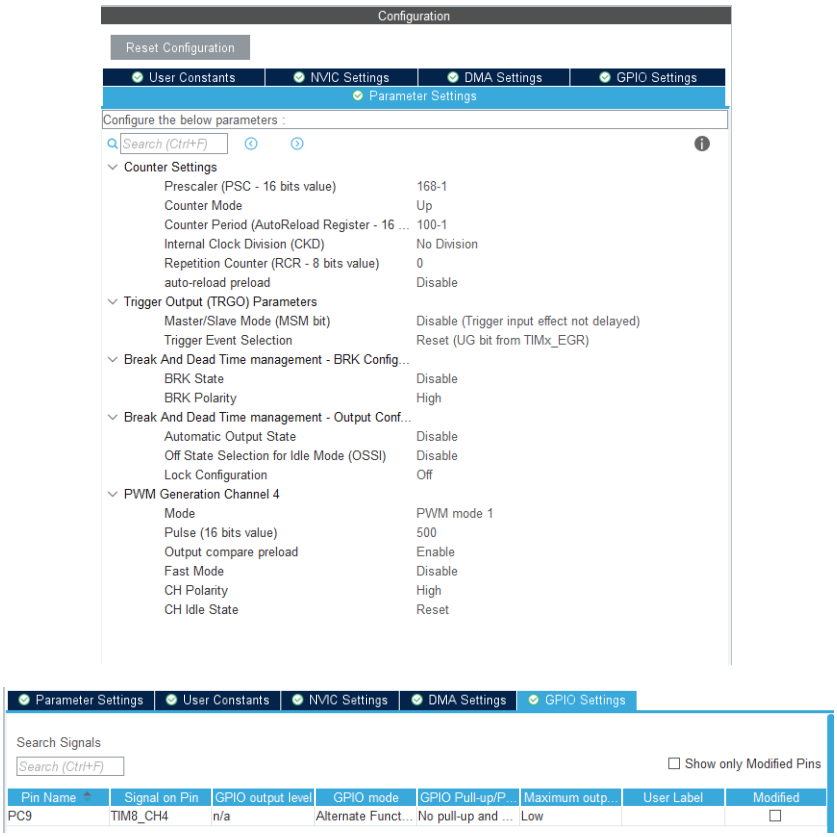

Figura 10: Configuração do PWM

A inicialização do PWM deve ser incuida no main.c void entre o /* USER CODE BEGIN 2 */, conforme o código:

```
/* USER CODE BEGIN 2 */
HAL_TIM_PWM_Start(&htim8,TIM_CHANNEL_4); // Inicia o PWM
```

```
  TIM8->CCR4 =30; //Duty=CCR/ARR -> Inicia um PWM de 30% de duty-cycle
 /* USER CODE END 2 */
```

Neste momento, é recomendável avaliar a saída PWM com o auxílio de um osciloscópio. Analise a frequência de chaveamento e verifique se está de acordo com o que foi programado.

3.4 Configuração da entrada digital

A entrada digital é utilizada pelo módulo do modelo matemático do motor de corrente contínua (CC) para ler o estado do sistema, ou seja, verificar se o sinal PWM está em nível alto ou baixo. Isso permite ao microcontrolador chavear o modelo de acordo com o estado atual do sinal PWM, garantindo o funcionamento adequado do conversor.

Para configurar uma entrada digital basta clicar com o botão direito do mouse sobre o Pino desejado e configura-lo como *GPIO_Input*, conforme a Figura 11.

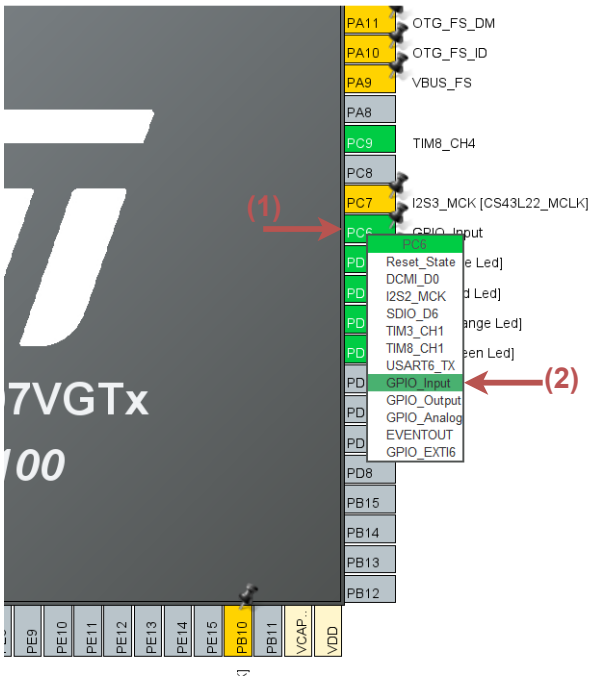

Figura 11: Configuração da entrada digital

Para esse exemplo configuramos o periférico *PC6*. No código para ler o estado dessa entra utilziamos o comando:

```
HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_6)
//Onde C e 6 devem ser trocadas de acordo com o seu periferico
// Por exemplo PD5 deve ser utilzado: HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_5)
```

3.5 Configuração da saída analógica - DAC

As saídas analógicas são utilizadas para representar as saídas da corrente do indutor e a tensão da carga no conversor CC. Essas saídas são importantes para o monitoramento e análise

do comportamento do sistema, permitindo que o controlador obtenha informações precisas sobre a corrente e a tensão do circuito, facilitando o controle e ajuste adequado do conversor.

DAC significa conversor digital para analógico e, como o nome sugere, converte o sinal digital em analógico. Conforme a Figura 12 clique sobre DAC (1) e habilite duas saídas digitais (2), note que o *Output Buffer* é ativado e o Trigger desativado. Ao finalizar dois periféricos serão ativados, no caso do exemplo *PA4* e*PA5*.

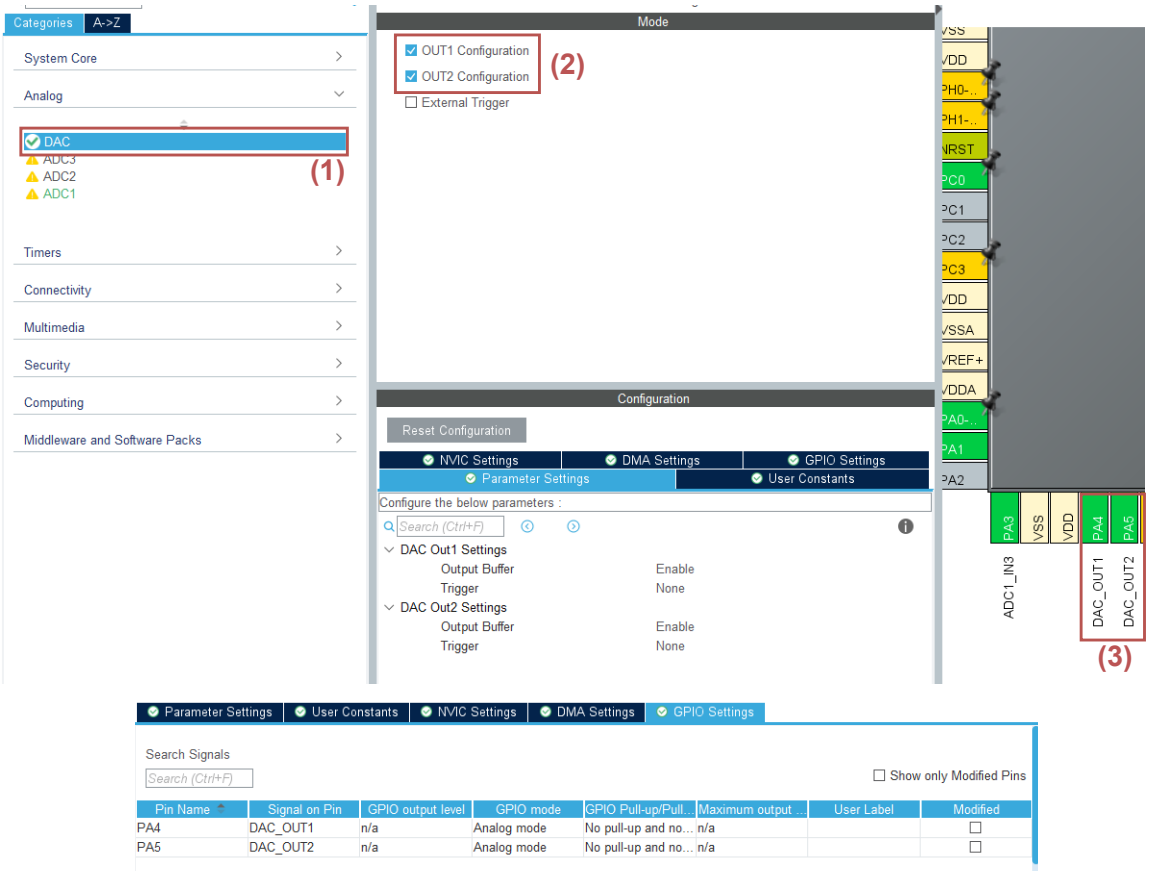

Figura 12: Configuração das saídas digitais

No código precisamos iniciar as saídas:

```
  /* USER CODE BEGIN 2 */
HAL_DAC_Start(&hdac, DAC_CHANNEL_1); //Inicia DAC chanel 1 (PA4)
HAL_DAC_Start(&hdac, DAC_CHANNEL_2); //Inicia DAC chanel 2 (PA5)
 /* USER CODE END 2 */
```

Correto, se os sinais enviados variam de 0 a 3.3V e estamos utilizando 12 bits de resolução, o range de envio deve ser convertido para o intervalo de 0 a 4095. Isso significa que os valores a serem enviados devem ser escalados proporcionalmente para esse novo range, a fim de utilizar toda a resolução disponível e obter uma representação mais precisa dos sinais analógicos. Pelos conhecimentos do modelo e da planta de controle inciamos as seguinte variáveis.

```
/* USER CODE BEGIN PV */
float iL=0; // Corrente no indutor
```

```
float vC=0; //Tensão na Carga
float iL_lim[2]={2,300}; // Valores limites de Corrente
float vC_lim[2]={2,300}; // Valores limites de Tensão
uint16_t sentIL =0; //Valor a ser enviado de corrente
uint16_t sentvC =0; //Valor a ser enviado de tensão
/* USER CODE END PV */
```

Os valores limites são estimados com base no conhecimento do modelo e são utilizados na conversão do range de atuação. A conversão pode ser feito por:

```
 float percentage=(iL-iL_lim[0])/(iL_lim[1]-iL_lim[0]);
 sentIL=percentage*(4095-0)+0;
percentage=(vC-vC_lim[0])/(vC_lim[1]-vC_lim[0]);
 sentvC=percentage*(4095-0)+0;

// Função para enviar os sinais analógicos
HAL_DAC_SetValue(&hdac, DAC1_CHANNEL_2, DAC_ALIGN_12B_R, sentIL);//iL
HAL_DAC_SetValue(&hdac, DAC1_CHANNEL_1, DAC_ALIGN_12B_R, sentvC);//vC
```

3.6 Configuração das entrada analógica - ADC

Para configurar as duas entradas analógicas e obter os valores de leitura de v_C (tensão do capacitor) e i_L (corrente do indutor), é recomendado utilizar o método DMA (Direct Memory Access). O DMA permite realizar a leitura analógica de forma não bloqueante, ou seja, enquanto o restante do programa é executado, o DMA continua buscando os valores em segundo plano.

Quando a conversão é concluída, os valores do ADC (Conversor Analógico-Digital) são salvos em um buffer pelo DMA. Dessa forma, podemos obter os valores lidos quando necessário, sem interromper a execução do programa principal.

O uso do DMA oferece vantagens em termos de desempenho e eficiência, permitindo que o programa continue executando outras tarefas enquanto as leituras analógicas são realizadas em segundo plano. Na Figura 13 é apresentada a sequência para a ativação dos periféricos. O exemplo apresenta a configuração de uma porta extra por precaução, porém somente duas são necessárias

Para a configuração seguimos o exemplo da Figura 14.

Na Figura 14 Como faremos a leitura analógicas continuamente ao longo do loop de controle selecionamos como *Enabled* a entrada *Continuos Conversion Mode* e como *Disabled* a entrada *Discontinuos Conversion Mode*. Se entrada *Scan mode* estiver ativado, a conversão não para no último canal do grupo selecionado, mas continua novamente a partir do primeiro canal do grupo selecionado. Para a utilização do DMA é necessário a habilitação do *Scan Mode* permite o armazenamento dos valores convertidos para posterior utilização pelo programa. Este modo será selecionado automaticamente se você estiver fazendo conversões para mais de 1 canal. Como *Resolution* usaremos 12 Bits, uma vez que foi configurado a mesma taxa pra as saídas analógicas anteriormente. Definimos o numero de conversões como 3 e o rank de prioridade conforme indicado. Por fim ativamos o DMA conforme a Figura 15.

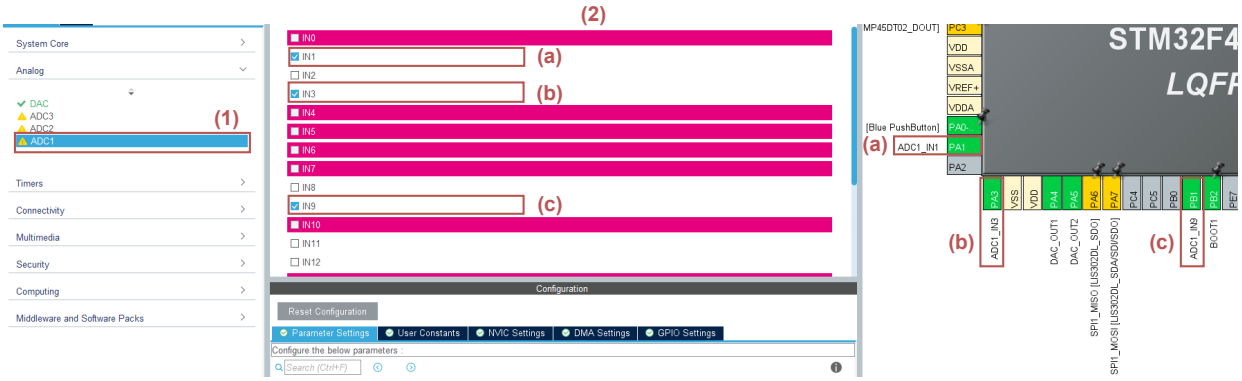

Figura 13: Ativação das entradas analógicas

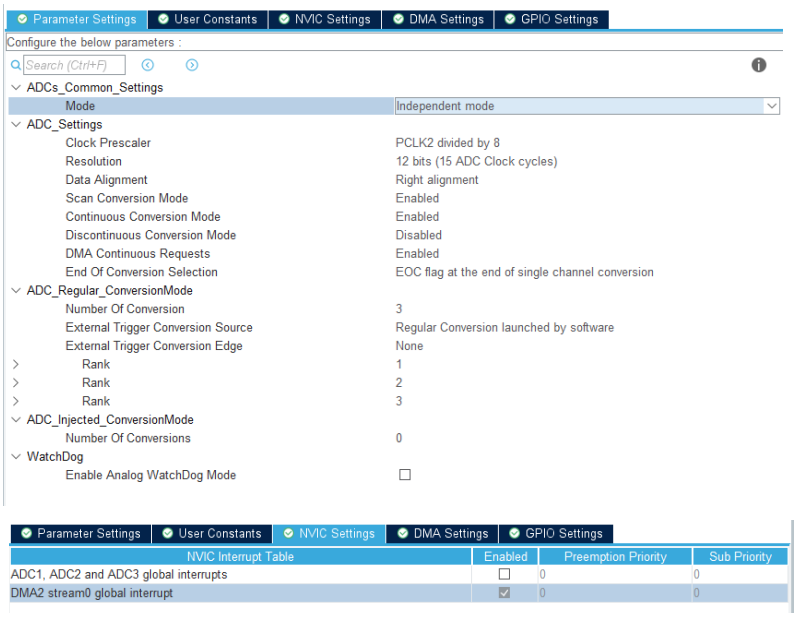

Figura 14: Configuração das entradas analógicas

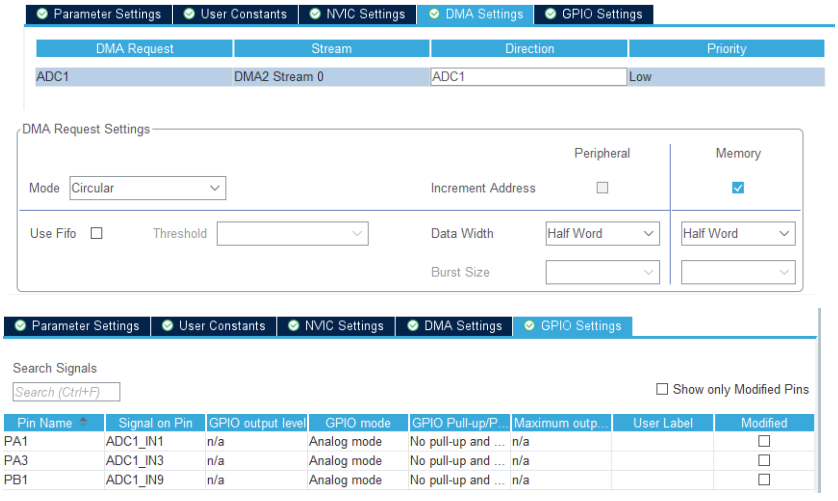

Figura 15: Configuração e ativação do DMA.

Ativa-se o DMA no modo *Circular*, isso garante que o DMA continue atuando durante todo o loop de controle. de modo geral, depois que a conversão é completa, o contador irá reiniciar e o DMA irá iniciar automaticamente. A entrada *Data Width* é selecionada como *Half WORD*.

No código criaremos um buffer que irá armazenar as variáveis lidas e definimos uma variável referente ao numero de canais ADC *NUMBER_ADC_CHANNEL* e uma variável que definira o tamanho do buffer *NUMBER_ADC_CHANNEL_AVERAGE_PER_CHANNEL*:

```
/* USER CODE BEGIN PD */
#define NUMBER_ADC_CHANNEL 3 //Número de canais
#define NUMBER_ADC_CHANNEL_AVERAGE_PER_CHANNEL 8 //Tamanho do Buffer
/* USER CODE END PD */
/* USER CODE BEGIN PV */
// Cria o Buffer
uint16_t ADC_DMA_BUFF[NUMBER_ADC_CHANNEL *
NUMBER_ADC_CHANNEL_AVERAGE_PER_CHANNEL]={0};
```

Em seguida, criamos uma função que, toda vez que solicitarmos os valores de tensão e corrente, fornecerá os últimos 8 valores de leitura de tensão e corrente entre a ocorrência da interrupção.

```
uint16_t ADC_DMA_AVERAGE(int channel)
{
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
        uint32_t adc_sum;
        int i;

        adc_sum=0;
        if(channel<NUMBER_ADC_CHANNEL)
        {
                for(i=0;i<NUMBER_ADC_CHANNEL_AVERAGE_PER_CHANNEL;i++)
                        adc_sum+=ADC_DMA_BUFF[channel+i*NUMBER_ADC_CHANNEL];
        }
        else
                return 1;

        return adc_sum/NUMBER_ADC_CHANNEL_AVERAGE_PER_CHANNEL;
}
/* USER CODE END 0 */
```

Iniciamos as entradas ADC e e o DMA:

```
HAL_ADC_Start(&hadc1);
HAL_ADC_Start_DMA(&hadc1, (uint32_t*)ADC_DMA_BUFF,
NUMBER_ADC_CHANNEL*NUMBER_ADC_CHANNEL_AVERAGE_PER_CHANNEL);
```

Para solicitar os valores de leitura do buffer basta "chamar"a função e converter os valores:

```
adciL=ADC_DMA_AVERAGE(0); //Solicita do primeiro buffer a corrente
adcvC=ADC_DMA_AVERAGE(1); //Solicita do segundo buffer a Tensão

// Converte os valores para a faixa de valores
adciL_aux=((float)(adciL)/4095)*(iL_lim[1]-iL_lim[0])+iL_lim[0];
adcvC_aux=((float)(adcvC)/4095)*(vC_lim[1]-vC_lim[0])+vC_lim[0];
```

3.7 Controlador PID

Para implementar um controlador em um microcontrolador, é necessário adaptá-lo para operar no domínio discreto, levando em consideração o período de tempo discreto. Assim, a equação do controlador PID do domínio Z deve ser transformado em termos de equações de diferenças.

$$C_{PID}(z) = \frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{4}$$

onde U(z) é a saída de controle e E(z) erro. Os coeficientes do controlador são escritos em termos da taxa de amostragem T_s:

$$\begin{aligned} b_0 &= K_p(1 + NT_s) + K_i T_s(1 + NT_s) + K_d N \\ b_1 &= -(K_p(2 + NT_s) + K_i T_s + 2K_d N) \\ b_2 &= K_p + K_d N \\ a_0 &= (1 + NT_s) \\ a_1 &= -(2 + NT_s) \\ a_2 &= 1 \end{aligned}$$

A partir de (4):

$$a_0 U(z) + a_1 z^{-1} U(z) + a_2 z^{-2} U(z) = +b_0 E(z) + b_1 z^{-1} E(z) + b_2 z^{-2} E(z) \tag{5}$$

$$a_0 U(z) = -a_1 z^{-1} U(z) - a_2 z^{-2} U(z) + b_0 E(z) + b_1 z^{-1} E(z) + b_2 z^{-2} E(z) \tag{6}$$

Logo a equação de diferenças do controlador a ser embarcada é dada por:

$$u[k] = -\frac{a_1}{a_0} u[k-1] - \frac{a_2}{a_0} u[k-2] + \frac{b_0}{a_0} e[k] + \frac{b_1}{a_0} e[k-1] + \frac{b_2}{a_0} e[k-2] \tag{7}$$

No código primeiramente definimos as variáveis:

```
/* USER CODE BEGIN PV */
//Variáveis do controlador de corrente
float N=20;
float kc_i=0.0144;  //Kc
```

```
float ti_i=4;       //Ki
float td_i=0;       //Kd
//Variáveis do controlador de tensão
float kc_v=0.0188;//Kc
float ti_v=32;   //Ki
float td_v=0;         //Kd

/* USER CODE END PV */
int main(void)
{
  /* USER CODE BEGIN 1 */
        a0=(1+N*Ts);
        a1=-(2+N*Ts);
        a2=1;
//Coeficientes do controlador de corrente
    b0_i=kc_i*(1+N*Ts)+ti_i*Ts*(1+N*Ts)+td_i*N;
    b1_i=-(kc_i*(2+N*Ts)+ti_i*Ts+2*td_i*N);
    b2_i=kc_i+td_i*N;
//Coeficientes do controlador de tensão
    b0_v=kc_v*(1+N*Ts)+ti_v*Ts*(1+N*Ts)+td_v*N;
    b1_v=-(kc_v*(2+N*Ts)+ti_v*Ts+2*td_v*N);
    b2_v=kc_v+td_v*N;
/* USER CODE END 1 */
```

Vale ressaltar que para o controle em cascata definem-se essas variáveis tanto para o controlador de corrente quanto para o de tensão. Uma dica é definir os coeficientes antes de iniciar o timer de interrupção, dessa forma diminui-se o tempo crítico de processamento, consequentemente relaxando a taxa de amostragem.

Por fim a equação de diferenças pode ser embarcada no loop como:

```
/* USER CODE BEGIN PV */
//Variáveis do loop de controle de corrente
    float UPID_i=0;
    float erro_i[3]={0,0,0};
    float u_i[2]={0,0};

// Variáveis do loop de controle de Tensão
    float u_i[2]={0,0};
    float u_v[2]={0,0};
    float UPID_v=0;

float ref; // Referência de tensão
/* USER CODE END PV */
```

No loop de controle:

```
 /* USER CODE BEGIN TIM1_UP_TIM10_IRQn 0 */

 //_______________________________PID Tensão_______________________________//

//Atualização das variáveis passadas
erro_v[2]=erro_v[1];
erro_v[1]=erro_v[0];
erro_v[0]=ref-adcvC_aux; //Cálcula o erro
u_v[1]=u_v[0];
u_v[0]=UPID_v;

// Equação de diferença
UPID_v=-(a1/a0)*u_v[0]-(a2/a0)*u_v[1]+(b0_v/a0)*erro_v[0]+(b1_v/a0)*erro_v[1]+
(b2_v/a0)*erro_v[2];

 //_______________________________PID Corrente_____________________________//

//Atualização das variáveis passadas
erro_i[2]=erro_i[1];
erro_i[1]=erro_i[0];
erro_i[0]=UPID_v-adciL_aux; //Erro em cascata
u_i[1]=u_i[0];
u_i[0]=UPID_i;

// Equação de diferença
UPID_i=-(a1/a0)*u_i[0]-(a2/a0)*u_i[1]+(b0_i/a0)*erro_i[0]+(b1_i/a0)*erro_i[1]+
(b2_i/a0)*erro_i[2];

 //_______________________________Saída PWM_______________________________//
// Definição do Duty-cycle
Duty=UPID_i*100; // Transforma pro range definido

// Limita o duty-cyle de 0 a 100%
if (Duty>98)
{
        Duty=98;
}
if (Duty<0)
{
        Duty=0;
}
// Envia o sinal PWM
TIM8->CCR4 = Duty; //Duty=CCR/ARR
```

3.8 Modelo matemático

O diagrama do circuito elétrico do conversor *boost* é apresentado na Figura 16. A configuração do modelo em linguagem C# deve seguir o diagrama de interrupção apresentado na Figura 5.

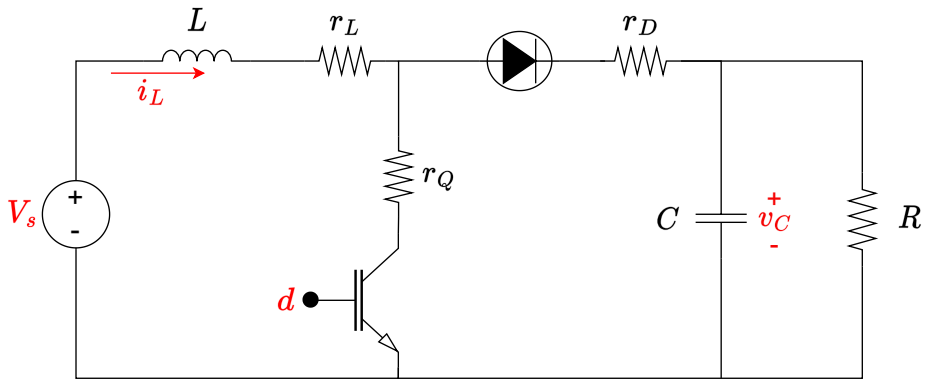

Figura 16: Diagrama elétrico do conversor CC-CC *Boost*.

onde L é a indutância, C a capacitância, V_s a tensão de entrada, R a resistência da carga, d a entrada PWM (pulse width modulation) e r_L, r_D e r_Q as perdas resistivas no indutor, diodo e interruptor de potência respectivamente. A dinâmica do modelo do conversor *boost* é derivada dos circuitos correspondentes dos estados de interrupção alto ($d = 1$) e baixo ($d = 0$) [2].As variáveis de espaço de estados do conversor boost podem ser descritas como [2]:

$$\begin{aligned} x &= [i_L \ v_C]^T, \\ u &= V_s \text{ e} \\ y &= x. \end{aligned}$$

O modelo de espaço do conversor com nível lógico do PWM alto é dado por:

$$\dot{x} = A_1 x + B_1 u \tag{8}$$

onde:

$$A_1 = \begin{bmatrix} -\frac{r_L + r_Q}{L} & 0 \\ 0 & -\frac{1}{RC} \end{bmatrix} \text{ e } B_1 = \begin{bmatrix} 1/L \\ 0 \end{bmatrix} \tag{9}$$

e para o nível lógico do PWM baixo é dado por:

$$A_1 = \begin{bmatrix} -\frac{r_L + r_Q}{L} & -\frac{1}{L} \\ \frac{1}{C} & -\frac{1}{RC} \end{bmatrix} \text{ e } B_1 = \begin{bmatrix} 1/L \\ 0 \end{bmatrix} \tag{10}$$

Para o código em C# iniciamos definindo as variáveis e as matrizes:

```
/* USER CODE BEGIN PV */
```

```
//Definição das variáveis do Modelo
    float Vs=100;   //Entrada de tensão
    float rL=0.1;   //Resistência Indutor
    float rD=0.1;   //Resistência Diodo
    float rQ=0.1;   //Resistência Power Switch
    float R=10;     //Load resistance
    float C=0.0004; //Capacit.
    float L=0.01;   //Induct.
    float Ts=0.00005;   //Taxa de amostragem

//Matrizes do estado 1
    float A1[2][2]={{0, 0},{0,0}};
    float B1[2]={0,0};
//Matrizes do estado 2
    float A2[2][2]={{0, 0},{0,0}};
    float B2[2]={0,0};
/* USER CODE END PV */
int main(void)
{
/* USER CODE BEGIN 1 */
//  Modelo estado 1
    A1[0][0]=-(rL+rQ)/L;
    A1[1][1]=-1/(R*C);
    B1[0]=1/L;
//  Modelo estado 2
    A2[0][0]=-(rL+rD)/L;
    A2[0][1]=-1/L;
    A2[1][1]=-1/(R*C);
    A2[1][0]=1/C;
    B2[0]=1/L;
```

No módulo contendo o modelo matemático os estados são configurados através do código:

```
//_____________________Modelo do Conversor Boost_____________________//
// Faz a Leitura do Estado
    if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_6))  //Se estado PWM alto
    {
        x_dot[0]=A1[0][0]*x[0]+A1[0][1]*x[1]+B1[0]*Vs; //Calculo iL
        x_dot[1]=A1[1][0]*x[0]+A1[1][1]*x[1]+B1[1]*Vs; //Calculo vC
    }
    else { //Se estado PWM baixo
    x_dot[0]=A2[0][0]*x[0]+A2[0][1]*x[1]+B2[0]*Vs; //Calculo iL
    x_dot[1]=A2[1][0]*x[0]+A2[1][1]*x[1]+B2[1]*Vs; //Calculo vC
    }
```

```
x[0]=x[0]+Ts*x_dot[0];
x[1]=x[1]+Ts*x_dot[1];

if (x[0]<0){
        x[0]=0;
}

iL=x[0]; // Corrente
vC=x[1]; // Tensão
//____________________Descreve as saídas analógicas____________________//
//Converte para 12 Bits
float percentage=(iL-iL_lim[0])/(iL_lim[1]-iL_lim[0]);
sentIL=percentage*(4095-0)+0;

 percentage=(vC-vC_lim[0])/(vC_lim[1]-vC_lim[0]);
sentvC=percentage*(4095-0)+0;

//Aciona as saídas
HAL_DAC_SetValue(&hdac, DAC1_CHANNEL_2, DAC_ALIGN_12B_R, sentIL);//iL
HAL_DAC_SetValue(&hdac, DAC1_CHANNEL_1, DAC_ALIGN_12B_R, sentvC);//vC
```

3.9 Compilando o código e observando as variáveis

Para carregar o código desenvolvido para o microcontrolador basta seguir os passos descritos na Figura 17.

Para observarmos as variáveis em tempo real podemos ativas o *Live expressions*. Depois de compilar o código, uma nova janela será aberta. Seguindo os passos da Figura 18 abrimos a aba do *Live expressions*.

Dentro da aba do *live expressions* adicionamos as variáveis que queremos observar, conforme a Figura 19.

Por fim, podemos rodar o código e observar o funcionamento do HIL projetado em tempo real, conforme a Figura 20.

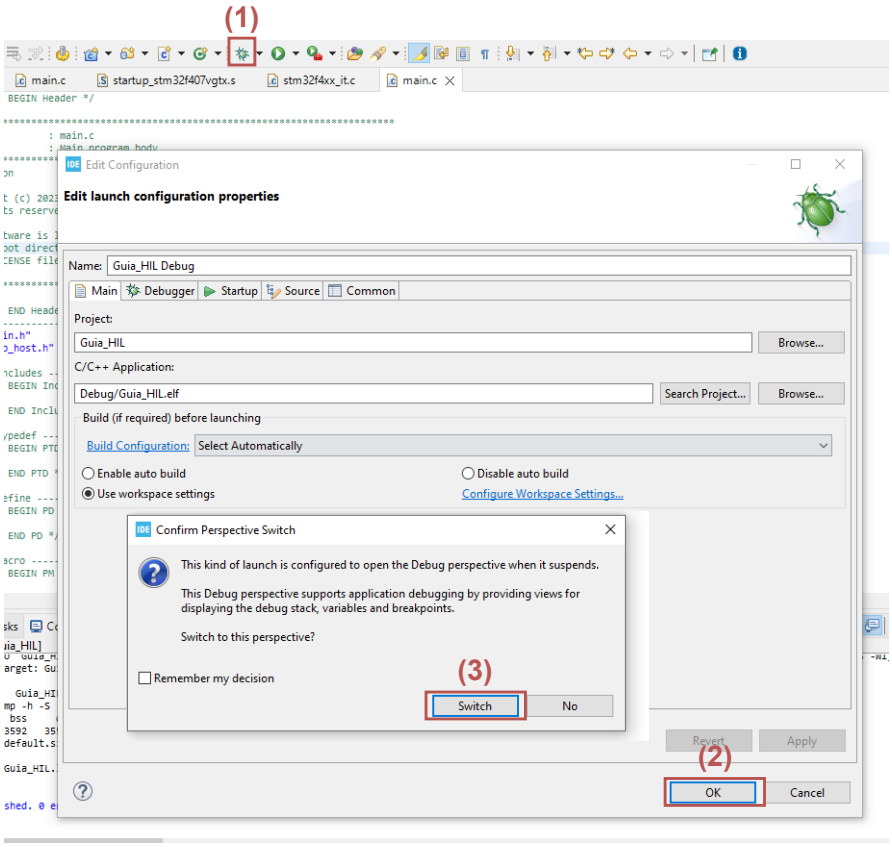

Figura 17: Com o microcontrolador conectado: Caregar o código

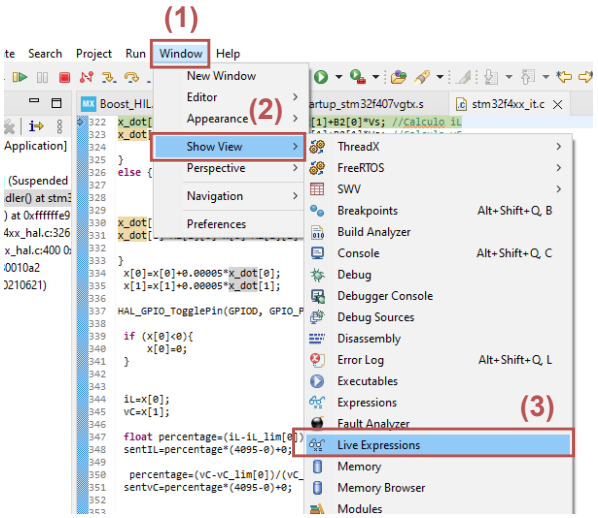

Figura 18: Ativando o *Live expressions*

3.10 Cube Monitor

Uma maneira mais eficiente de observar a resposta do HIL é através do CubeMonitor, o qual é uma ferramenta de analise gráfica distribuída gratuitamente pela própria STM32 `https://www.st.com/en/development-tools/stm32cubemonitor.html`. Primeiramente abra o CubeMonitor, na aba *Menu* em seguida *Import*, copie o texto do arquivo txt disponibilizado no link e

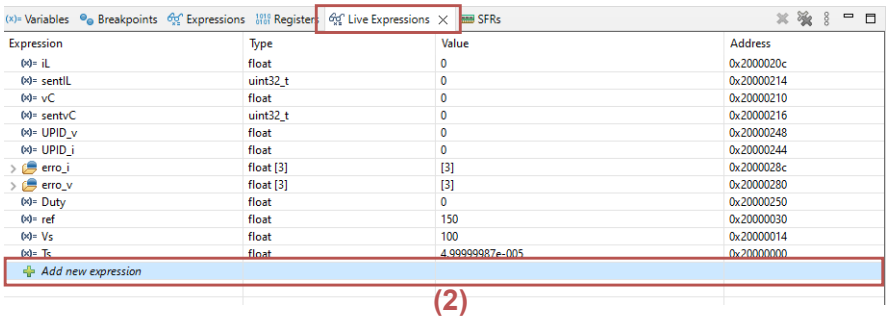

Figura 19: Inserindo variáveis

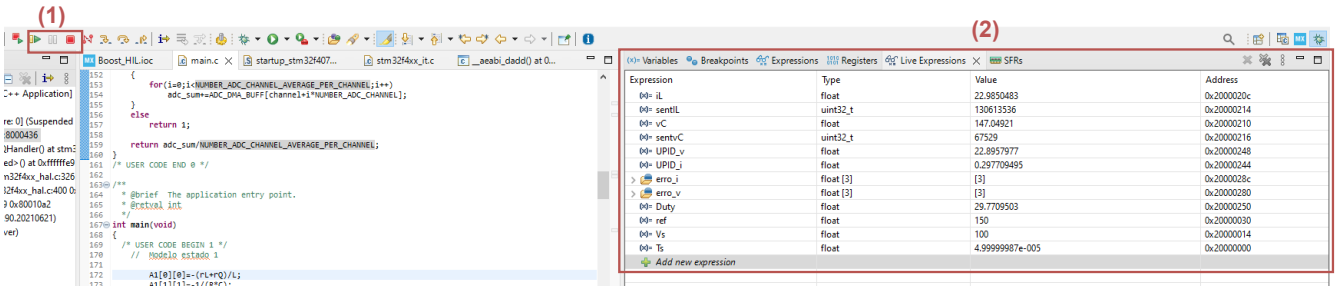

Figura 20: Em (1) na sequência podemos rodar, pausar e parar o loop do HIL, em (2) tempos a atualização da variáveis do HIL em tempo real.

cole no clipbord. `https://drive.google.com/file/d/1RzLj9tjEK93_EaM-8RMA1uKW7IaCej4u/view?usp=drive_link`

Primeiramente carregamos os arquivos *.elf* gerados a partir da compilação do código no CubeIDE. Na guia de design, seguindo a Figura 21, clique em MyVariables.

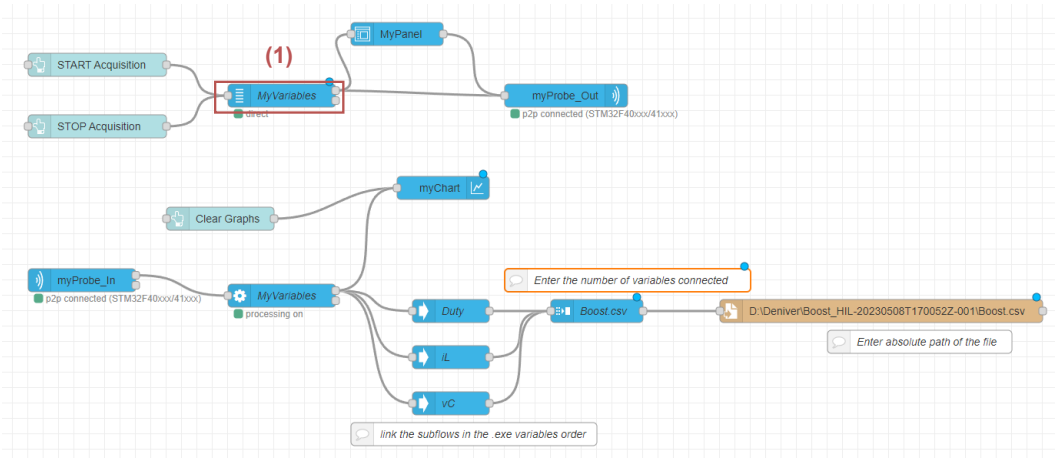

Figura 21: Designe carregado

Pela 22 em (2) colaca-se o local do arquivo criado apartir do CubeIde, em (3) seleciona-se o arquivo *elf* e por fim seleciona-se as variáveis desejadas, aqui selecionaremos a corrente i_L, a tensão v_C, a referência e o Duty-cycle gerado pela saída do controlador.

Com o STM32 conectado à entra USB iniciaremos a conexão com o STM32 através do STLink, conforme os passos da Figura 23

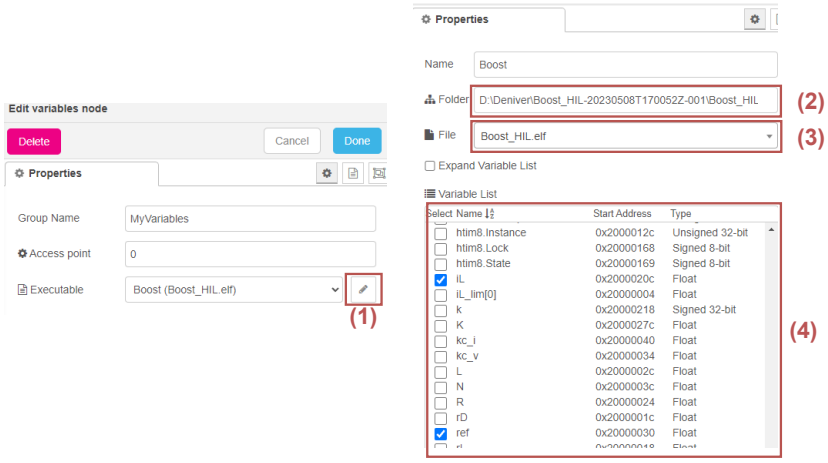

Figura 22: Selecionando as variáveis

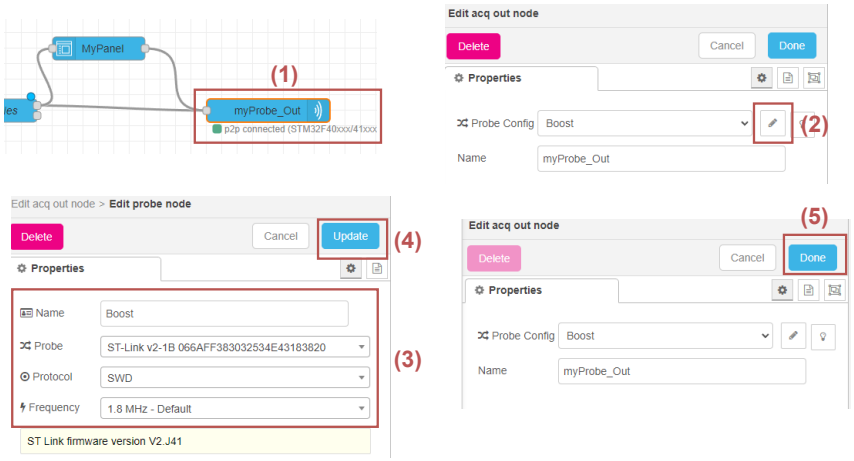

Figura 23: Coneção entre o STM e o Cube Monitor. *Probe* irá variar conforme sua conexão.

Se a conexão for estabelecida um ícone verde indicando a conexão p2p irá aparecer sobre *myProbe_Out*. Em *MyProbe_In* realize os mesmos passos para efetuar a conexão com STM32.

Por fim, iniciamos a comunicação seguindo os passos da Figura 24. **Se a conexão for bem sucedida um ícone verde sob *MyVariables* indicará a iniciação do processamento.**

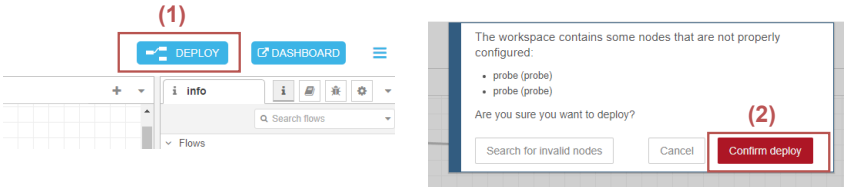

Figura 24: Coneção entre o STM e o Cube Monitor.

Em *Dashboard* ao lado do botão *Deploy* será gerado um *dashboard* gráfico para a avaliação das variáveis do HIL. Clicando em *Star Acquisition* pode-se iniciar a impressão gráfica das variáveis selecionadas. Mais abaixo pode-se variar a referência do controlador selecionando-se a variável de referência e clicando em *Write*. A Figura 25 apresenta a resposta do HIL projetado

nesse guia, sendo i_L a corrente no indutor, *Duty* do duty-cycle do sinal PWM e v_C a corrente na carga.

Figura 25: Resultado final do HIL projetado

Nessa resposta podemos ver que através do controle em cascata projetado o controlador segue a referência de tensão variando-se o duty-cycle do sinal PWM, atestando tanto o funcionamento do HIL projetado.

3.11 Considerações Finais

Nesse guia foi apresentado o passo a passo para o desenvolvimento de uma simulação *Hardwaere-in-the-loop* HIL de um conversor boost com tensão controlada por um controlador Proporcional Integrativo em cascata. Por se tratar de um guia especifico adaptações podem ser necessárias dependendo da aplicação.

Referências

[1] J. A. Ledin, "Hardware-in-the-loop simulation," *Embedded Systems Programming*, vol. 12, pp. 42–62, 1999.

[2] D. S. Castro, R. F. Magossi, R. F. Bastos, V. A. Oliveira, and R. Q. Machado, "Low-cost hardware in the loop implementation of a boost converter," in *2019 18th European Control Conference (ECC)*. IEEE, 2019, pp. 423–428.

[3] S. Microelectronics, "Stm32f405xx/stm32f407xx datasheet," Tech. rep., ST Microelectronics, Tech. Rep., 2013. [Online]. Available: https://pdf1.alldatasheet.com/datasheet-pdf/view/435286/STMICROELECTRONICS/STM32F407XX.html

Alguns sites que podem ajudar:

1. Timers: https://community.st.com/t5/stm32-mcus/how-to-generate-a-one-second-interru
   ta-p/49858
2. PWM: https://controllerstech.com/pwm-in-stm32/
3. DAC (Saída analógica): https://controllerstech.com/dac-in-stm32/
4. ADC (Entrada analógica): https://controllerstech.com/stm32-adc-single-channel/
5. Cube Monitor:https://wiki.st.com/stm32mcu/wiki/STM32CubeMonitor:How_to_log_
   data_in_a_.csv_file