Controle de temperatura

Gerado por Doxygen 1.9.2

Capítulo 1

Índice dos Módulos

1.1 Módulos

Esta é a lista de todos os módulos:

Capítulo 2

Índice dos Arquivos

2.1 Lista de Arquivos

Esta é a lista de todos os arquivos documentados e suas respectivas descrições:

Capítulo 3

Módulos

3.1 CMSIS

Módulos

- Stm32f1xx_system

3.1.1 Descrição detalhada

3.2 Stm32f1xx_system

Módulos

- STM32F1xx_System_Private_Includes
- STM32F1xx_System_Private_TypesDefinitions
- STM32F1xx_System_Private_Defines
- STM32F1xx_System_Private_Macros
- STM32F1xx_System_Private_Variables
- STM32F1xx_System_Private_FunctionPrototypes
- STM32F1xx_System_Private_Functions

3.2.1 Descrição detalhada

3.3 STM32F1xx_System_Private_Includes

3.4 STM32F1xx_System_Private_TypesDefinitions

3.5 STM32F1xx_System_Private_Defines

Definições e Macros

- #define HSE_VALUE 8000000U
- #define HSI_VALUE 8000000U

3.5.1 Descrição detalhada

3.5.2 Definições e macros

3.5.2.1 HSE_VALUE

```
#define HSE_VALUE 8000000U
```

Default value of the External oscillator in Hz. This value can be provided and adapted by the user application.

Definição na linha 79 do arquivo system_stm32f1xx.c.

3.5.2.2 HSI_VALUE

```
#define HSI_VALUE 8000000U
```

Default value of the Internal oscillator in Hz. This value can be provided and adapted by the user application.

Definição na linha 84 do arquivo system_stm32f1xx.c.

3.6 STM32F1xx_System_Private_Macros

3.7 STM32F1xx_System_Private_Variables

Variáveis

- uint32_t SystemCoreClock = 16000000
- const uint8_t AHBPrescTable [16U] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}
- const uint8_t APBPrescTable [8U] = {0, 0, 0, 0, 1, 2, 3, 4}

3.7.1 Descrição detalhada

3.7.2 Variáveis

3.7.2.1 AHBPrescTable

```
const uint8_t AHBPrescTable[16U] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}
```

Definição na linha 143 do arquivo system_stm32f1xx.c.

3.7.2.2 APBPrescTable

```
const uint8_t APBPrescTable[8U] = {0, 0, 0, 0, 1, 2, 3, 4}
```

Definição na linha 144 do arquivo system_stm32f1xx.c.

3.7.2.3 SystemCoreClock

```
uint32_t SystemCoreClock = 16000000
```

Definição na linha 142 do arquivo system_stm32f1xx.c.

3.8 STM32F1xx_System_Private_FunctionPrototypes

3.9 STM32F1xx_System_Private_Functions

Funções

- void SystemInit (void)

  Setup the microcontroller system Initialize the Embedded Flash Interface, the PLL and update the SystemCoreClock variable.
- void SystemCoreClockUpdate (void)

  Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

3.9.1 Descrição detalhada

3.9.2 Funções

3.9.2.1 SystemCoreClockUpdate()

```
void SystemCoreClockUpdate (
            void  )
```

Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

Observação

Each time the core clock (HCLK) changes, this function must be called to update SystemCoreClock variable value. Otherwise, any configuration based on this variable will be incorrect.

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:

- If SYSCLK source is HSI, SystemCoreClock will contain the HSI_VALUE(∗)
- If SYSCLK source is HSE, SystemCoreClock will contain the HSE_VALUE(∗∗)
- If SYSCLK source is PLL, SystemCoreClock will contain the HSE_VALUE(∗∗) or HSI_VALUE(∗) multiplied by the PLL factors.

(∗) HSI_VALUE is a constant defined in stm32f1xx.h file (default value 8 MHz) but the real value may vary depending on the variations in voltage and temperature.

(∗∗) HSE_VALUE is a constant defined in stm32f1xx.h file (default value 8 MHz or 25 MHz, depending on the product used), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.

- The result of this function could be not correct when using fractional value for HSE crystal.

Parâmetros

None	

Valores Retornados

None	

Definição na linha 225 do arquivo system_stm32f1xx.c.

```
{
  uint32_t tmp = 0U, pllmull = 0U, pllsource = 0U;

#if defined(STM32F105xC) || defined(STM32F107xC)
  uint32_t prediv1source = 0U, prediv1factor = 0U, prediv2factor = 0U, pll2mull = 0U;
#endif /* STM32F105xC */

#if defined(STM32F100xB) || defined(STM32F100xE)
  uint32_t prediv1factor = 0U;
#endif /* STM32F100xB or STM32F100xE */

  /* Get SYSCLK source -------------------------------------------------------*/
  tmp = RCC->CFGR & RCC_CFGR_SWS;

  switch (tmp)
  {
    case 0x00U:  /* HSI used as system clock */
      SystemCoreClock = HSI_VALUE;
      break;
    case 0x04U:  /* HSE used as system clock */
      SystemCoreClock = HSE_VALUE;
      break;
    case 0x08U:  /* PLL used as system clock */

      /* Get PLL clock source and multiplication factor ----------------------*/
      pllmull = RCC->CFGR & RCC_CFGR_PLLMULL;
      pllsource = RCC->CFGR & RCC_CFGR_PLLSRC;

#if !defined(STM32F105xC) && !defined(STM32F107xC)
```

```
      pllmull = ( pllmull » 18U) + 2U;

      if (pllsource == 0x00U)
      {
        /* HSI oscillator clock divided by 2 selected as PLL clock entry */
        SystemCoreClock = (HSI_VALUE » 1U) * pllmull;
      }
      else
      {
 #if defined(STM32F100xB) || defined(STM32F100xE)
       prediv1factor = (RCC->CFGR2 & RCC_CFGR2_PREDIV1) + 1U;
       /* HSE oscillator clock selected as PREDIV1 clock entry */
       SystemCoreClock = (HSE_VALUE / prediv1factor) * pllmull;
 #else
        /* HSE selected as PLL clock entry */
        if ((RCC->CFGR & RCC_CFGR_PLLXTPRE) != (uint32_t)RESET)
        {/* HSE oscillator clock divided by 2 */
          SystemCoreClock = (HSE_VALUE » 1U) * pllmull;
        }
        else
        {
          SystemCoreClock = HSE_VALUE * pllmull;
        }
 #endif
      }
#else
      pllmull = pllmull » 18U;

      if (pllmull != 0x0DU)
      {
         pllmull += 2U;
      }
      else
      { /* PLL multiplication factor = PLL input clock * 6.5 */
        pllmull = 13U / 2U;
      }

      if (pllsource == 0x00U)
      {
        /* HSI oscillator clock divided by 2 selected as PLL clock entry */
        SystemCoreClock = (HSI_VALUE » 1U) * pllmull;
      }
      else
      {/* PREDIV1 selected as PLL clock entry */

        /* Get PREDIV1 clock source and division factor */
        prediv1source = RCC->CFGR2 & RCC_CFGR2_PREDIV1SRC;
        prediv1factor = (RCC->CFGR2 & RCC_CFGR2_PREDIV1) + 1U;

        if (prediv1source == 0U)
        {
          /* HSE oscillator clock selected as PREDIV1 clock entry */
          SystemCoreClock = (HSE_VALUE / prediv1factor) * pllmull;
        }
        else
        {/* PLL2 clock selected as PREDIV1 clock entry */

          /* Get PREDIV2 division factor and PLL2 multiplication factor */
          prediv2factor = ((RCC->CFGR2 & RCC_CFGR2_PREDIV2) » 4U) + 1U;
          pll2mull = ((RCC->CFGR2 & RCC_CFGR2_PLL2MUL) » 8U) + 2U;
          SystemCoreClock = (((HSE_VALUE / prediv2factor) * pll2mull) / prediv1factor) * pllmull;
        }
      }
#endif /* STM32F105xC */
      break;

    default:
      SystemCoreClock = HSI_VALUE;
      break;
  }

  /* Compute HCLK clock frequency ----------------*/
  /* Get HCLK prescaler */
  tmp = AHBPrescTable[((RCC->CFGR & RCC_CFGR_HPRE) » 4U)];
  /* HCLK clock frequency */
  SystemCoreClock »= tmp;
}
```

3.9.2.2 SystemInit()

```
void SystemInit (
void )
```

Setup the microcontroller system Initialize the Embedded Flash Interface, the PLL and update the SystemCoreClock variable.

Observação

This function should be used only after reset.

Parâmetros

None	

Valores Retornados

None	

Definição na linha 176 do arquivo system_stm32f1xx.c.

```
{
#if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) || defined(STM32F103xE) ||
 defined(STM32F103xG)
  #ifdef DATA_IN_ExtSRAM
    SystemInit_ExtMemCtl();
  #endif /* DATA_IN_ExtSRAM */
#endif

  /* Configure the Vector Table location -------------------------------------*/
#if defined(USER_VECT_TAB_ADDRESS)
  SCB->VTOR = VECT_TAB_BASE_ADDRESS | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM. */
#endif /* USER_VECT_TAB_ADDRESS */
}
```

Capítulo 4

Arquivos

4.1 Referência do Arquivo D:/BACKUP/Faculdade/16_Embarcados/↩ Controlador_Temperatura/Controle-de-temperatura/↩ Codigos/controle_temperatura_RTOS/Core/Src/FIRFilter.c

Implementação do filtro FIR.

```
#include "FIRFilter.h"
```

Funções

- void FIRFilter_Init (FIRFilter ∗fir)
- float FIRFilter_Update (FIRFilter ∗fir, float inp)

4.1.1 Descrição detalhada

Implementação do filtro FIR.

Autor

Arthur Damasceno

Atenção

Este código apresenta a implementação de filtro FIR com coeficientes calculados previamente

Definição no arquivo FIRFilter.c.

4.1.2 Funções

4.1.2.1 FIRFilter_Init()

```
void FIRFilter_Init (
             FIRFilter * fir )
```

Definição na linha 19 do arquivo FIRFilter.c.

```
00019                                     {
00020 
00021     /* Reinicia o buffer do filtro */
00022     for(uint8_t n=0; n<FIR_FILTER_LENGTH;n++){
00023 
00024         fir->buf[n] = 0.0f;
00025 
00026     }
00027 
00028     /* Reseta o index do buffer */
00029     fir->bufindex = 0;
00030 
00031     /* Reseta a saída do filtro */
00032     fir->out = 0.0f;
00033 }
```

4.1.2.2 FIRFilter_Update()

```
float FIRFilter_Update (
             FIRFilter * fir,
             float inp )
```

Definição na linha 35 do arquivo FIRFilter.c.

```
00035                                                   {
00036 
00037     /* Guarda a entrada no buffer */
00038     fir->buf[fir->bufindex] = inp;
00039 
00040     /* Incrementa o index do buffer e reinicia se necessário */
00041     fir->bufindex++;
00042 
00043     if(fir->bufindex == FIR_FILTER_LENGTH){
00044 
00045         fir->bufindex = 0;
00046 
00047     }
00048 
00049     /* Calcula nova saída via convolução */
00050     fir->out = 0.0f;
00051 
00052     uint8_t sumIndex = fir->bufindex;
00053 
00054     for(uint8_t n=0; n<FIR_FILTER_LENGTH;n++){
00055 
00056         /* Decrementa o index e reseta se necessary */
00057         if(sumIndex>0){
00058 
00059             sumIndex--;
00060 
00061         }else{
00062 
00063             sumIndex = FIR_FILTER_LENGTH -1;
00064 
00065         }
00066 
00067         /* Multiplica resposta ao impulso com a entrada deslocada e soma a saída */
00068         fir->out += FIR_IMPULSE_RESPONSE[n] * fir->buf[sumIndex];
00069     }
00070 
00071     /* Returna saída filtrada */
00072     return fir->out;
00073 
00074 }
```

4.2 FIRFilter.c

Vá para a documentação desse arquivo.

```c

#include "FIRFilter.h"

/* Coeficientes do filtro */
static float FIR_IMPULSE_RESPONSE[FIR_FILTER_LENGTH] = {0.02840647f, 0.23700821f, 0.46917063f, 0.23700821f, 0.02840647f};

void FIRFilter_Init(FIRFilter *fir){

    /* Reinicia o buffer do filtro */
    for(uint8_t n=0; n<FIR_FILTER_LENGTH;n++){

        fir->buf[n] = 0.0f;

    }

    /* Reseta o index do buffer */
    fir->bufindex = 0;

    /* Reseta a saída do filtro */
    fir->out = 0.0f;
}

float FIRFilter_Update(FIRFilter *fir, float inp){

    /* Guarda a entrada no buffer */
    fir->buf[fir->bufindex] = inp;

    /* Incrementa o index do buffer e reinicia se necessário */
    fir->bufindex++;

    if(fir->bufindex == FIR_FILTER_LENGTH){

        fir->bufindex = 0;

    }

    /* Calcula nova saída via convolução */
    fir->out = 0.0f;

    uint8_t sumIndex = fir->bufindex;

    for(uint8_t n=0; n<FIR_FILTER_LENGTH;n++){

        /* Decrementa o index e reseta se necessary */
        if(sumIndex>0){

            sumIndex--;

        }else{

            sumIndex = FIR_FILTER_LENGTH -1;

        }

        /* Multiplica resposta ao impulso com a entrada deslocada e soma a saída */
        fir->out += FIR_IMPULSE_RESPONSE[n] * fir->buf[sumIndex];
    }

    /* Returna saída filtrada */
    return fir->out;

}
```

4.3 freertos.c

```c
/* USER CODE BEGIN Header */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "FreeRTOS.h"
#include "task.h"
#include "main.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 ---------------------------------------------------------*/
/* USER CODE BEGIN Variables */

/* USER CODE END Variables */

/* Private function prototypes -----------------------------------------------*/
/* USER CODE BEGIN FunctionPrototypes */

/* USER CODE END FunctionPrototypes */

/* GetIdleTaskMemory prototype (linked to static allocation support) */
void vApplicationGetIdleTaskMemory( StaticTask_t **ppxIdleTaskTCBBuffer, StackType_t **ppxIdleTaskStackBuffer, uint32_t *pulIdleTaskStackSize );

/* USER CODE BEGIN GET_IDLE_TASK_MEMORY */
static StaticTask_t xIdleTaskTCBBuffer;
static StackType_t xIdleStack[configMINIMAL_STACK_SIZE];

void vApplicationGetIdleTaskMemory( StaticTask_t **ppxIdleTaskTCBBuffer, StackType_t **ppxIdleTaskStackBuffer, uint32_t *pulIdleTaskStackSize )
{
  *ppxIdleTaskTCBBuffer = &xIdleTaskTCBBuffer;
  *ppxIdleTaskStackBuffer = &xIdleStack[0];
  *pulIdleTaskStackSize = configMINIMAL_STACK_SIZE;
  /* place for user code */
}
/* USER CODE END GET_IDLE_TASK_MEMORY */

/* Private application code --------------------------------------------------*/
/* USER CODE BEGIN Application */

/* USER CODE END Application */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

4.4 Referência do Arquivo D:/BACKUP/Faculdade/16_Embarcados/Controlador_Temperatura/Controle-de-temperatura/Codigos/controle_temperatura_RTOS/Core/Src/main.c

Corpo principal do programa.

```
#include "main.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "string.h"
#include "stdio.h"
#include "stdlib.h"
#include "FIRFilter.h"
```

Definições e Macros

- #define r 0.000210115034038755f

- #define p 1.0f
- #define k 0.0638221651196478f
- #define ksat 0.1f
- #define CSen HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_RESET);
- #define CSdis HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
- #define SCK_H HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
- #define SCK_L HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);

Funções

- void SystemClock_Config (void)

  Configuração do clock do sistema.
- void Temp_taskF (void ∗pvParameters)

  Tarefa de leitura da temperatura.
- void Filter_taskF (void ∗pvParameters)

  Tarefa de filtro da variável temperatura.
- void Control_taskF (void ∗pvParameters)

  Tarefa de atualização da referencia, calculo e execução da lei de controle.
- void Display_taskF (void ∗pvParameters)

  Tarefa de atualização das variáveis no display.
- int main (void)

  ponto de entrada da aplicação.
- void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef ∗htim)

  chamada da função de período
- void Error_Handler (void)

  Função executada em caso de erro na aplicação.

Variáveis

- ADC_HandleTypeDef hadc1
- TIM_HandleTypeDef htim1
- UART_HandleTypeDef huart1
- TaskHandle_t Temp_Task
- TaskHandle_t Filter_Task
- TaskHandle_t Control_Task
- TaskHandle_t Display_Task
- QueueHandle_t tempQueue
- QueueHandle_t filteredTempQueue
- FIRFilter tempFilter
- float filteredTemp = 0
- float ref = 0
- float dutyCycle = 0

4.4.1 Descrição detalhada

Corpo principal do programa.

Autor

Arthur Damasceno

Mateus Piccinin

Atenção

Controlador de temperatura

Este código apresenta a implementação de um controlador de temperatura utilizando a interface FreeRTOS.

Definição no arquivo main.c.

4.4.2 Definições e macros

4.4.2.1 CSdis

```
#define CSdis HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
```

Definição na linha 36 do arquivo main.c.

4.4.2.2 CSen

```
#define CSen HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_RESET);
```

Definição na linha 35 do arquivo main.c.

4.4.2.3 k

```
#define k 0.0638221651196478f
```

Definição na linha 32 do arquivo main.c.

4.4.2.4 ksat

```
#define ksat 0.1f
```

Definição na linha 33 do arquivo main.c.

4.4.2.5 p

```
#define p 1.0f
```

Definição na linha 31 do arquivo main.c.

4.4.2.6 r

```
#define r 0.000210115034038755f
```

Definição na linha 30 do arquivo main.c.

4.4.2.7 SCK_H

```
#define SCK_H HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
```

Definição na linha 38 do arquivo main.c.

4.4.2.8 SCK_L

```
#define SCK_L HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);
```

Definição na linha 39 do arquivo main.c.

4.4.3 Funções

4.4.3.1 Control_taskF()

```
void Control_taskF (
             void * pvParameters )
```

Tarefa de atualização da referencia, calculo e execução da lei de controle.

Observação

> Esta tarefa executa o calculo da lei de controle, atualizando o valor de referencia setando a razão cíclica da chave de saída ou ativando a ventoinha de resfrianmento

Parâmetros

∗pvParameters	(não utilizado) permite iniciar a função com valor inical

Valores Retornados

None	

Definição na linha 386 do arquivo main.c.

```
00386                                         {
00387     while (1) {
00388         float rx_filteredTemp;
00389         /* Recebe da fila filteredTempQueue */
00390         if (xQueueReceive(filteredTempQueue, &rx_filteredTemp, 10)) {
00391             /* Leitura da entrada analógica para calculo de referencia */
00392             HAL_ADC_PollForConversion(&hadc1, 10);
00393             ref = (float) HAL_ADC_GetValue(&hadc1) / 27.3; // leitura do potenciometro convertido em
       ref até 150°C
00394 
00395             /* Lei de controle */
00396             float u;
00397             static float up, uint;
00398             int flag_sat;
00399             float ek = ref - rx_filteredTemp;
00400 
00401             /* Controlador bang-bang ventoinha */
00402             if (ek < -15.0) {
00403                 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_SET);
00404             } else {
00405                 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET);
00406             }
00407 
00408             /* Anti-windup integrador */
00409             if (!flag_sat) {
00410                 uint = uint * p + r * ek;
00411 
00412             } else if (flag_sat) {
00413                 uint = (uint * p + r * ek) * ksat;
00414             }
00415 
00416             /* Proporcional */
00417             up = k * ek;
00418 
00419             /* Ação de controle */
00420             u = up + uint;
00421 
00422             /* Conversão período PWM */
00423             u = u * 4500.0;
00424 
00425             /* Limites de saturação de PWM */
00426             if (u > 18000.0) {
00427                 u = 18000.0;
00428                 flag_sat = 1;
00429             } else if (u < 0) {
00430                 u = 0;
00431                 flag_sat = 1;
00432             } else {
00433                 u = u;
00434                 flag_sat = 0;
00435             }
00436 
00437             /* Converte periodo do timer em razão cíclica */
00438             dutyCycle = u / 180.0;
00439 
00440             /* Seta periférico PWM */
00441             __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, (uint32_t ) u);
00442 
00443         }
00444 
00445     }
00446 }
```

4.4.3.2 Display_taskF()

```
void Display_taskF (
               void * pvParameters )
```

Tarefa de atualização das variáveis no display.

Observação

> Esta tarefa envia ao display TFT via UART os valores atualizados de referência, variável manipulada(razão cíclica) e variável de processo (temperatura)

Parâmetros

**pvParameters*	(não utilizado) permite iniciar a função com valor inical

Valores Retornados

None	

Definição na linha 456 do arquivo main.c.

```
00456                                      {
00457     while (1) {
00458         char str[100];
00459         /* Fim de comando definido pela API do display */
00460         uint8_t Cmd_End[3] = { 0xFF, 0xFF, 0xFF };
00461 
00462         /* Atualiza valor do setpoint */
00463         int32_t number = ref * 100;
00464         sprintf(str, "setPoint.val=%ld", number);
00465         HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
00466         HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);
00467 
00468         /* Atualiza valor da variável de processo */
00469         number = filteredTemp * 100;
00470         sprintf(str, "filteredTemp.val=%ld", number);
00471         HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
00472         HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);
00473 
00474         /* Atualiza valor da variável manipulada */
00475         number = dutyCycle * 100;
00476         sprintf(str, "dutyCycle.val=%ld", number);
00477         HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
00478         HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);
00479 
00480         /* Atraso para definição do período da tarefa */
00481         vTaskDelay(1000); /*1Hz frequency*/
00482     }
00483 }
```

4.4.3.3 Error_Handler()

```
void Error_Handler (
               void )
```

Função executada em caso de erro na aplicação.

Valores Retornados

None	

Definição na linha 501 do arquivo main.c.

```
00501                       {
00502     __disable_irq();
00503     while (1) {
00504     }
00505 }
```

4.4.3.4 Filter_taskF()

```
void Filter_taskF (
            void * pvParameters )
```

Tarefa de filtro da variável temperatura.

Observação

> Esta tarefa executa a chamada para o filtro FIR, após 5 atualizações o valor é adicionado a fila filteredTempQueue

Parâmetros

**pvParameters*	(não utilizado) permite iniciar a função com valor inical

Valores Retornados

None	

Definição na linha 357 do arquivo main.c.

```
00357                                  {
00358     uint8_t aux = 0;
00359     while (1) {
00360         float rx_temp;
00361         /* Recebe da fila filteredTempQueue */
00362         if (xQueueReceive(tempQueue, &rx_temp, 10)) {
00363             aux++;
00364             /* Chamada do filtro FIR */
00365             filteredTemp = FIRFilter_Update(&tempFilter, rx_temp);
00366 
00367         }
00368         if (aux == 5) {
00369 
00370             aux = 0;
00371             /* Adiciona a fila filteredTempQueue */
00372             if (xQueueSend(filteredTempQueue, &filteredTemp, 10) == pdPASS) {
00373             }
00374         }
00375     }
00376 }
```

4.4.3.5 HAL_TIM_PeriodElapsedCallback()

```
void HAL_TIM_PeriodElapsedCallback (
            TIM_HandleTypeDef * htim )
```

chamada da função de período

Observação

> Esta função atualiza o valor de "uwTick" utilizado como base de tempo do sistema

Parâmetros

htim	: TIM handle

Valores Retornados

None	

Definição na linha 491 do arquivo main.c.

```
00491                                                                {
00492     if (htim->Instance == TIM4) {
00493         HAL_IncTick();
00494     }
00495 }
```

4.4.3.6 main()

```
int main (
              void  )
```

ponto de entrada da aplicação.

Valores Retornados

int	

Definição na linha 76 do arquivo main.c.

```
00076                 {
00077     /* Reinicia todos os periféricos, inicializa a interface flash e o systick */
00078     HAL_Init();
00079 
00080     /* Configura o clock do sistema */
00081     SystemClock_Config();
00082 
00083     /* Inicializa todos os periféricos configurados */
00084     MX_GPIO_Init();
00085     MX_ADC1_Init();
00086     MX_TIM1_Init();
00087     MX_USART1_UART_Init();
00088 
00089     /* Inicializa o conversor AD */
00090     if (HAL_ADC_Start(&hadc1) != HAL_OK) {
00091         Error_Handler();
00092     }
00093 
00094     /* Inicializa o timer1 em modo PWM */
00095     if (HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1) != HAL_OK) {
00096         Error_Handler();
00097     }
00098 
00099     /* Inicializa o filtro FIR */
00100     FIRFilter_Init(&tempFilter);
00101 
00102     /* Cria a fila de leituras de temperatura bruta */
00103     tempQueue = xQueueCreate(1, sizeof(float));
00104     if (tempQueue == 0) {
00105         Error_Handler();
00106     }
00107 
00108     /* Cria a fila de leituras de temperatura filtrada */
00109     filteredTempQueue = xQueueCreate(1, sizeof(float));
00110     if (tempQueue == 0) {
00111         Error_Handler();
00112     }
00113 
00114     /* Cria tasks na pilha do sistema */
00115     xTaskCreate(Temp_taskF, "TempTask", 128, NULL, 3, &Temp_Task);
00116     xTaskCreate(Filter_taskF, "FilterTask", 128, NULL, 2, &Filter_Task);
00117     xTaskCreate(Control_taskF, "ControlTask", 128, NULL, 4, &Control_Task);
00118     xTaskCreate(Display_taskF, "DisplayTask", 128, NULL, 1, &Display_Task);
00119 
00120     /* Inicializa o escalonador */
00121     vTaskStartScheduler();
00122 
00123     /* Loop infinito */
00124     while (1) {
00125     }
00126 }
```

4.4.3.7 SystemClock_Config()

```
void SystemClock_Config (
            void  )
```

Configuração do clock do sistema.

Valores Retornados

None	

Definição na linha 132 do arquivo main.c.

```
00132                                     {
00133     RCC_OscInitTypeDef RCC_OscInitStruct = { 0 };
00134     RCC_ClkInitTypeDef RCC_ClkInitStruct = { 0 };
00135     RCC_PeriphCLKInitTypeDef PeriphClkInit = { 0 };
00136 
00137     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
00138     RCC_OscInitStruct.HSEState = RCC_HSE_ON;
00139     RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
00140     RCC_OscInitStruct.HSIState = RCC_HSI_ON;
00141     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
00142     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
00143     RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
00144     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
00145         Error_Handler();
00146     }
00147 
00148     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK
00149             | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
00150     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
00151     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
00152     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
00153     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
00154 
00155     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK) {
00156         Error_Handler();
00157     }
00158     PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
00159     PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV6;
00160     if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK) {
00161         Error_Handler();
00162     }
00163 }
```

4.4.3.8 Temp_taskF()

```
void Temp_taskF (
            void * pvParameters )
```

Tarefa de leitura da temperatura.

Observação

Esta tarefa executa a leitura da temperatura armazenada na memória do módulo MAX6675 atravez de um bitbanging do protocolo SPI, ao fim da conversão o valor é adicionado a fila tempQueue

Parâmetros

∗*pvParameters*	(não utilizado) permite iniciar a função com valor inical

Valores Retornados

None	

Definição na linha 315 do arquivo main.c.

```
00315                                 {
00316     while (1) {
00317         uint8_t tempdata[16];
00318         uint16_t temp16 = 0;
00319 
00320         /* bitbanging protocolo SPI */
00321         CSen
00322         for (int i = 0; i < 16; i++) {
00323             SCK_H
00324             tempdata[i] = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_4);
00325             SCK_L
00326         }
00327         CSdis
00328 
00329         /* Conversão temperatura */
00330         if (tempdata[13] == 0) {
00331 
00332             for (int n = 1; n < 13; n++) {
00333                 temp16 += tempdata[n] * (2048 / (1 « (n - 1)));
00334             }
00335 
00336         }
00337 
00338         float temp = (float) temp16 / 4;
00339 
00340         /* Adiciona a fila tempQueue */
00341         if (xQueueSend(tempQueue, &temp, 10) == pdPASS) {
00342         }
00343 
00344         /* Atraso para definição do período da tarefa */
00345         vTaskDelay(200); /*5Hz frequency*/
00346     }
00347 }
```

4.4.4 Variáveis

4.4.4.1 Control_Task

```
TaskHandle_t Control_Task
```

Definição na linha 49 do arquivo main.c.

4.4.4.2 Display_Task

```
TaskHandle_t Display_Task
```

Definição na linha 50 do arquivo main.c.

4.4.4.3 dutyCycle

```
float dutyCycle = 0
```

Definição na linha 59 do arquivo main.c.

4.4.4.4 Filter_Task

```
TaskHandle_t Filter_Task
```

Definição na linha 48 do arquivo main.c.

4.4.4.5 filteredTemp

```
float filteredTemp = 0
```

Definição na linha 57 do arquivo main.c.

4.4.4.6 filteredTempQueue

```
QueueHandle_t filteredTempQueue
```

Definição na linha 53 do arquivo main.c.

4.4.4.7 hadc1

```
ADC_HandleTypeDef hadc1
```

Definição na linha 41 do arquivo main.c.

4.4.4.8 htim1

```
TIM_HandleTypeDef htim1
```

Definição na linha 43 do arquivo main.c.

4.4.4.9 huart1

```
UART_HandleTypeDef huart1
```

Definição na linha 45 do arquivo main.c.

4.4.4.10 ref

```
float ref = 0
```

Definição na linha 58 do arquivo main.c.

4.4.4.11 Temp_Task

```
TaskHandle_t Temp_Task
```

Definição na linha 47 do arquivo main.c.

4.4.4.12 tempFilter

```
FIRFilter tempFilter
```

Definição na linha 55 do arquivo main.c.

4.4.4.13 tempQueue

```
QueueHandle_t tempQueue
```

Definição na linha 52 do arquivo main.c.

4.5 main.c

Vá para a documentação desse arquivo.

```c

#include "main.h"

#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"

#include "string.h"
#include "stdio.h"
#include "stdlib.h"

#include "FIRFilter.h"

#define r 0.000210115034038755f
#define p 1.0f
#define k 0.0638221651196478f
#define ksat 0.1f

#define CSen HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_RESET);
#define CSdis HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);

#define SCK_H HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
#define SCK_L HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);

ADC_HandleTypeDef hadc1;

TIM_HandleTypeDef htim1;

UART_HandleTypeDef huart1;

TaskHandle_t Temp_Task;
TaskHandle_t Filter_Task;
TaskHandle_t Control_Task;
TaskHandle_t Display_Task;

QueueHandle_t tempQueue;
QueueHandle_t filteredTempQueue;

FIRFilter tempFilter;

float filteredTemp = 0;
float ref = 0;
float dutyCycle = 0;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM1_Init(void);
static void MX_USART1_UART_Init(void);

void Temp_taskF(void *pvParameters);
void Filter_taskF(void *pvParameters);
void Control_taskF(void *pvParameters);
void Display_taskF(void *pvParameters);

int main(void) {
    /* Reinicia todos os periféricos, inicializa a interface flash e o systick */
    HAL_Init();

    /* Configura o clock do sistema */
    SystemClock_Config();

    /* Inicializa todos os periféricos configurados */
    MX_GPIO_Init();
    MX_ADC1_Init();
    MX_TIM1_Init();
    MX_USART1_UART_Init();

    /* Inicializa o conversor AD */
    if (HAL_ADC_Start(&hadc1) != HAL_OK) {
        Error_Handler();
    }

    /* Inicializa o timer1 em modo PWM */
    if (HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1) != HAL_OK) {
        Error_Handler();
    }

    /* Inicializa o filtro FIR */
    FIRFilter_Init(&tempFilter);

    /* Cria a fila de leituras de temperatura bruta */
    tempQueue = xQueueCreate(1, sizeof(float));
    if (tempQueue == 0) {
        Error_Handler();
    }

    /* Cria a fila de leituras de temperatura filtrada */
    filteredTempQueue = xQueueCreate(1, sizeof(float));
    if (tempQueue == 0) {
        Error_Handler();
    }

    /* Cria tasks na pilha do sistema */
    xTaskCreate(Temp_taskF, "TempTask", 128, NULL, 3, &Temp_Task);
    xTaskCreate(Filter_taskF, "FilterTask", 128, NULL, 2, &Filter_Task);
    xTaskCreate(Control_taskF, "ControlTask", 128, NULL, 4, &Control_Task);
    xTaskCreate(Display_taskF, "DisplayTask", 128, NULL, 1, &Display_Task);

    /* Inicializa o escalonador */
    vTaskStartScheduler();

    /* Loop infinito */
    while (1) {
    }
}

void SystemClock_Config(void) {
    RCC_OscInitTypeDef RCC_OscInitStruct = { 0 };
    RCC_ClkInitTypeDef RCC_ClkInitStruct = { 0 };
    RCC_PeriphCLKInitTypeDef PeriphClkInit = { 0 };

    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
        Error_Handler();
    }

    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK
            | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK) {
        Error_Handler();
    }
    PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
    PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV6;
    if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK) {
        Error_Handler();
    }
}

static void MX_ADC1_Init(void) {

    ADC_ChannelConfTypeDef sConfig = { 0 };

    hadc1.Instance = ADC1;
    hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
    hadc1.Init.ContinuousConvMode = ENABLE;
    hadc1.Init.DiscontinuousConvMode = DISABLE;
    hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
    hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc1.Init.NbrOfConversion = 1;
    if (HAL_ADC_Init(&hadc1) != HAL_OK) {
        Error_Handler();
    }

    sConfig.Channel = ADC_CHANNEL_0;
    sConfig.Rank = ADC_REGULAR_RANK_1;
    sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES_5;
    if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) {
        Error_Handler();
    }
}

static void MX_TIM1_Init(void) {

    TIM_MasterConfigTypeDef sMasterConfig = { 0 };
    TIM_OC_InitTypeDef sConfigOC = { 0 };
    TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = { 0 };

    htim1.Instance = TIM1;
    htim1.Init.Prescaler = 1;
    htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim1.Init.Period = 18000 - 1;
    htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim1.Init.RepetitionCounter = 0;
    htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_PWM_Init(&htim1) != HAL_OK) {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig)
            != HAL_OK) {
        Error_Handler();
    }
    sConfigOC.OCMode = TIM_OCMODE_PWM1;
    sConfigOC.Pulse = 0;
    sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
    sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
    sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
    sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
    sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
    if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1)
            != HAL_OK) {
        Error_Handler();
    }
    sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
    sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
    sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
    sBreakDeadTimeConfig.DeadTime = 0;
    sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
    sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
    sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
    if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig)
            != HAL_OK) {
        Error_Handler();
    }

    HAL_TIM_MspPostInit(&htim1);
}

static void MX_USART1_UART_Init(void) {

    huart1.Instance = USART1;
    huart1.Init.BaudRate = 115200;
    huart1.Init.WordLength = UART_WORDLENGTH_8B;
    huart1.Init.StopBits = UART_STOPBITS_1;
    huart1.Init.Parity = UART_PARITY_NONE;
    huart1.Init.Mode = UART_MODE_TX;
    huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart1.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart1) != HAL_OK) {
        Error_Handler();
    }
}

static void MX_GPIO_Init(void) {
    GPIO_InitTypeDef GPIO_InitStruct = { 0 };

    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOD_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_RESET);

    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);

    GPIO_InitStruct.Pin = GPIO_PIN_15;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_3;
    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);

    GPIO_InitStruct.Pin = GPIO_PIN_4;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_15;
    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);
}

void Temp_taskF(void *pvParameters) {
    while (1) {
        uint8_t tempdata[16];
        uint16_t temp16 = 0;

        /* bitbanging protocolo SPI */
        CSen
        for (int i = 0; i < 16; i++) {
            SCK_H
            tempdata[i] = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_4);
            SCK_L
        }
        CSdis

        /* Conversão temperatura */
        if (tempdata[13] == 0) {

            for (int n = 1; n < 13; n++) {
                temp16 += tempdata[n] * (2048 / (1 « (n - 1)));
            }

        }

        float temp = (float) temp16 / 4;

        /* Adiciona a fila tempQueue */
        if (xQueueSend(tempQueue, &temp, 10) == pdPASS) {
        }

        /* Atraso para definição do período da tarefa */
        vTaskDelay(200); /*5Hz frequency*/
    }
}

void Filter_taskF(void *pvParameters) {
    uint8_t aux = 0;
    while (1) {
        float rx_temp;
        /* Recebe da fila filteredTempQueue */
        if (xQueueReceive(tempQueue, &rx_temp, 10)) {
            aux++;
            /* Chamada do filtro FIR */
            filteredTemp = FIRFilter_Update(&tempFilter, rx_temp);

        }
        if (aux == 5) {

            aux = 0;
            /* Adiciona a fila filteredTempQueue */
            if (xQueueSend(filteredTempQueue, &filteredTemp, 10) == pdPASS) {
            }
        }
    }
}

void Control_taskF(void *pvParameters) {
    while (1) {
        float rx_filteredTemp;
        /* Recebe da fila filteredTempQueue */
        if (xQueueReceive(filteredTempQueue, &rx_filteredTemp, 10)) {
            /* Leitura da entrada analógica para calculo de referencia */
            HAL_ADC_PollForConversion(&hadc1, 10);
            ref = (float) HAL_ADC_GetValue(&hadc1) / 27.3; // leitura do potenciometro convertido em ref até 150°C

            /* Lei de controle */
            float u;
            static float up, uint;
            int flag_sat;
            float ek = ref - rx_filteredTemp;

            /* Controlador bang-bang ventoinha */
            if (ek < -15.0) {
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_SET);
            } else {
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET);
            }

            /* Anti-windup integrador */
            if (!flag_sat) {
                uint = uint * p + r * ek;

            } else if (flag_sat) {
                uint = (uint * p + r * ek) * ksat;
            }

            /* Proporcional */
            up = k * ek;

            /* Ação de controle */
            u = up + uint;

            /* Conversão período PWM */
            u = u * 4500.0;

            /* Limites de saturação de PWM */
            if (u > 18000.0) {
                u = 18000.0;
                flag_sat = 1;
            } else if (u < 0) {
                u = 0;
                flag_sat = 1;
            } else {
                u = u;
                flag_sat = 0;
            }

            /* Converte periodo do timer em razão cíclica */
            dutyCycle = u / 180.0;

            /* Seta periférico PWM */
            __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, (uint32_t ) u);

        }

    }
}

void Display_taskF(void *pvParameters) {
    while (1) {
        char str[100];
        /* Fim de comando definido pela API do display */
        uint8_t Cmd_End[3] = { 0xFF, 0xFF, 0xFF };

        /* Atualiza valor do setpoint */
        int32_t number = ref * 100;
        sprintf(str, "setPoint.val=%ld", number);
        HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
        HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);

        /* Atualiza valor da variável de processo */
        number = filteredTemp * 100;
        sprintf(str, "filteredTemp.val=%ld", number);
        HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
        HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);

        /* Atualiza valor da variável manipulada */
        number = dutyCycle * 100;
        sprintf(str, "dutyCycle.val=%ld", number);
        HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
        HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);

        /* Atraso para definição do período da tarefa */
        vTaskDelay(1000); /*1Hz frequency*/
    }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM4) {
        HAL_IncTick();
    }
}

void Error_Handler(void) {
    __disable_irq();
    while (1) {
    }
}

#ifdef  USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line)
{
}
#endif
```

4.6 Referência do Arquivo D:/BACKUP/Faculdade/16_Embarcados/Controlador_Temperatura/↩ Controle-de-temperatura/Codigos/controle_temperatura_RTOS/↩ Core/Src/stm32f1xx_hal_msp.c

This file provides code for the MSP Initialization and de-Initialization codes.

```
#include "main.h"
```

Funções

- void HAL_TIM_MspPostInit (TIM_HandleTypeDef ∗htim)
- void HAL_MspInit (void)
- void HAL_ADC_MspInit (ADC_HandleTypeDef ∗hadc)

  ADC MSP Initialization This function configures the hardware resources used in this example.
- void HAL_ADC_MspDeInit (ADC_HandleTypeDef ∗hadc)

  ADC MSP De-Initialization This function freeze the hardware resources used in this example.
- void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef ∗htim_oc)

  TIM_OC MSP Initialization This function configures the hardware resources used in this example.
- void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef ∗htim_oc)

  TIM_OC MSP De-Initialization This function freeze the hardware resources used in this example.
- void HAL_UART_MspInit (UART_HandleTypeDef ∗huart)

  UART MSP Initialization This function configures the hardware resources used in this example.
- void HAL_UART_MspDeInit (UART_HandleTypeDef ∗huart)

  UART MSP De-Initialization This function freeze the hardware resources used in this example.

4.6.1 Descrição detalhada

This file provides code for the MSP Initialization and de-Initialization codes.

Atenção

Definição no arquivo stm32f1xx_hal_msp.c.

4.6.2 Funções

4.6.2.1 HAL_ADC_MspDeInit()

```
void HAL_ADC_MspDeInit (
            ADC_HandleTypeDef ∗ hadc )
```

ADC MSP De-Initialization This function freeze the hardware resources used in this example.

Parâmetros

hadc	ADC handle pointer

Valores Retornados

None	

ADC1 GPIO Configuration PA0-WKUP ------> ADC1_IN0

Definição na linha 126 do arquivo stm32f1xx_hal_msp.c.

```
{
  if(hadc->Instance==ADC1)
  {
  /* USER CODE BEGIN ADC1_MspDeInit 0 */

  /* USER CODE END ADC1_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_ADC1_CLK_DISABLE();

    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_0);

  /* USER CODE BEGIN ADC1_MspDeInit 1 */

  /* USER CODE END ADC1_MspDeInit 1 */
  }

}
```

4.6.2.2 HAL_ADC_MspInit()

```
void HAL_ADC_MspInit (
              ADC_HandleTypeDef * hadc )
```

ADC MSP Initialization This function configures the hardware resources used in this example.

Parâmetros

hadc	ADC handle pointer

Valores Retornados

None	

ADC1 GPIO Configuration PA0-WKUP ------> ADC1_IN0

Definição na linha 94 do arquivo stm32f1xx_hal_msp.c.

```
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(hadc->Instance==ADC1)
  {
  /* USER CODE BEGIN ADC1_MspInit 0 */

  /* USER CODE END ADC1_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_ADC1_CLK_ENABLE();

    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = GPIO_PIN_0;
00110     GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
00111     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00112
00113   /* USER CODE BEGIN ADC1_MspInit 1 */
00114
00115   /* USER CODE END ADC1_MspInit 1 */
00116   }
00117
00118 }
```

4.6.2.3 HAL_MspInit()

```
void HAL_MspInit (
              void  )
```

Initializes the Global MSP. DISABLE: JTAG-DP Disabled and SW-DP Disabled

Definição na linha 66 do arquivo stm32f1xx_hal_msp.c.

```
00067 {
00068   /* USER CODE BEGIN MspInit 0 */
00069
00070   /* USER CODE END MspInit 0 */
00071
00072   __HAL_RCC_AFIO_CLK_ENABLE();
00073   __HAL_RCC_PWR_CLK_ENABLE();
00074
00075   /* System interrupt init*/
00076   /* PendSV_IRQn interrupt configuration */
00077   HAL_NVIC_SetPriority(PendSV_IRQn, 15, 0);
00078
00081   __HAL_AFIO_REMAP_SWJ_DISABLE();
00082
00083   /* USER CODE BEGIN MspInit 1 */
00084
00085   /* USER CODE END MspInit 1 */
00086 }
```

4.6.2.4 HAL_TIM_MspPostInit()

```
void HAL_TIM_MspPostInit (
              TIM_HandleTypeDef * htim )
```

TIM1 GPIO Configuration PA8 ------> TIM1_CH1

Definição na linha 170 do arquivo stm32f1xx_hal_msp.c.

```
00171 {
00172   GPIO_InitTypeDef GPIO_InitStruct = {0};
00173   if(htim->Instance==TIM1)
00174   {
00175   /* USER CODE BEGIN TIM1_MspPostInit 0 */
00176
00177   /* USER CODE END TIM1_MspPostInit 0 */
00178
00179     __HAL_RCC_GPIOA_CLK_ENABLE();
00183     GPIO_InitStruct.Pin = GPIO_PIN_8;
00184     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
00185     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00186     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00187
00188   /* USER CODE BEGIN TIM1_MspPostInit 1 */
00189
00190   /* USER CODE END TIM1_MspPostInit 1 */
00191   }
00192
00193 }
```

4.6.2.5 HAL_TIM_PWM_MspDeInit()

```
void HAL_TIM_PWM_MspDeInit (
            TIM_HandleTypeDef * htim_oc )
```

TIM_OC MSP De-Initialization This function freeze the hardware resources used in this example.

Parâmetros

htim_oc	TIM_OC handle pointer

Valores Retornados

None	

Definição na linha 200 do arquivo stm32f1xx_hal_msp.c.

```
{
  if(htim_oc->Instance==TIM1)
  {
  /* USER CODE BEGIN TIM1_MspDeInit 0 */

  /* USER CODE END TIM1_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_TIM1_CLK_DISABLE();
  /* USER CODE BEGIN TIM1_MspDeInit 1 */

  /* USER CODE END TIM1_MspDeInit 1 */
  }

}
```

4.6.2.6 HAL_TIM_PWM_MspInit()

```
void HAL_TIM_PWM_MspInit (
            TIM_HandleTypeDef * htim_oc )
```

TIM_OC MSP Initialization This function configures the hardware resources used in this example.

Parâmetros

htim_oc	TIM_OC handle pointer

Valores Retornados

None	

Definição na linha 154 do arquivo stm32f1xx_hal_msp.c.

```
{
  if(htim_oc->Instance==TIM1)
  {
  /* USER CODE BEGIN TIM1_MspInit 0 */

  /* USER CODE END TIM1_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_TIM1_CLK_ENABLE();
  /* USER CODE BEGIN TIM1_MspInit 1 */

  /* USER CODE END TIM1_MspInit 1 */
  }

}
```

4.6.2.7 HAL_UART_MspDeInit()

```
void HAL_UART_MspDeInit (
            UART_HandleTypeDef * huart )
```

UART MSP De-Initialization This function freeze the hardware resources used in this example.

Parâmetros

huart	UART handle pointer

Valores Retornados

None	

USART1 GPIO Configuration PA9 ---—> USART1_TX PA10 ---—> USART1_RX

Definição na linha 261 do arquivo stm32f1xx_hal_msp.c.

```
{
  if(huart->Instance==USART1)
  {
  /* USER CODE BEGIN USART1_MspDeInit 0 */

  /* USER CODE END USART1_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_USART1_CLK_DISABLE();

    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_9|GPIO_PIN_10);

  /* USER CODE BEGIN USART1_MspDeInit 1 */

  /* USER CODE END USART1_MspDeInit 1 */
  }

}
```

4.6.2.8 HAL_UART_MspInit()

```
void HAL_UART_MspInit (
            UART_HandleTypeDef * huart )
```

UART MSP Initialization This function configures the hardware resources used in this example.

Parâmetros

huart	UART handle pointer

Valores Retornados

None	

USART1 GPIO Configuration PA9 ------> USART1_TX PA10 ------> USART1_RX

Definição na linha 222 do arquivo stm32f1xx_hal_msp.c.

```
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(huart->Instance==USART1)
  {
  /* USER CODE BEGIN USART1_MspInit 0 */

  /* USER CODE END USART1_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_USART1_CLK_ENABLE();

    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = GPIO_PIN_9;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_10;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* USER CODE BEGIN USART1_MspInit 1 */

  /* USER CODE END USART1_MspInit 1 */
  }

}
```

4.7 stm32f1xx_hal_msp.c

Vá para a documentação desse arquivo.

```
/* USER CODE BEGIN Header */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN TD */

/* USER CODE END TD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN Define */

/* USER CODE END Define */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN Macro */

/* USER CODE END Macro */

/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* External functions --------------------------------------------------------*/
/* USER CODE BEGIN ExternalFunctions */

/* USER CODE END ExternalFunctions */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);
void HAL_MspInit(void)
{
  /* USER CODE BEGIN MspInit 0 */

  /* USER CODE END MspInit 0 */

  __HAL_RCC_AFIO_CLK_ENABLE();
  __HAL_RCC_PWR_CLK_ENABLE();

  /* System interrupt init*/
  /* PendSV_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(PendSV_IRQn, 15, 0);

  __HAL_AFIO_REMAP_SWJ_DISABLE();

  /* USER CODE BEGIN MspInit 1 */

  /* USER CODE END MspInit 1 */
}

void HAL_ADC_MspInit(ADC_HandleTypeDef* hadc)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(hadc->Instance==ADC1)
  {
  /* USER CODE BEGIN ADC1_MspInit 0 */

  /* USER CODE END ADC1_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_ADC1_CLK_ENABLE();

    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* USER CODE BEGIN ADC1_MspInit 1 */

  /* USER CODE END ADC1_MspInit 1 */
  }

}

void HAL_ADC_MspDeInit(ADC_HandleTypeDef* hadc)
{
  if(hadc->Instance==ADC1)
  {
  /* USER CODE BEGIN ADC1_MspDeInit 0 */

  /* USER CODE END ADC1_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_ADC1_CLK_DISABLE();

    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_0);

  /* USER CODE BEGIN ADC1_MspDeInit 1 */

  /* USER CODE END ADC1_MspDeInit 1 */
  }

}

void HAL_TIM_PWM_MspInit(TIM_HandleTypeDef* htim_oc)
{
  if(htim_oc->Instance==TIM1)
  {
  /* USER CODE BEGIN TIM1_MspInit 0 */

  /* USER CODE END TIM1_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_TIM1_CLK_ENABLE();
  /* USER CODE BEGIN TIM1_MspInit 1 */

  /* USER CODE END TIM1_MspInit 1 */
  }

}

void HAL_TIM_MspPostInit(TIM_HandleTypeDef* htim)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(htim->Instance==TIM1)
  {
  /* USER CODE BEGIN TIM1_MspPostInit 0 */

  /* USER CODE END TIM1_MspPostInit 0 */

    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = GPIO_PIN_8;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* USER CODE BEGIN TIM1_MspPostInit 1 */

  /* USER CODE END TIM1_MspPostInit 1 */
  }

}
void HAL_TIM_PWM_MspDeInit(TIM_HandleTypeDef* htim_oc)
{
  if(htim_oc->Instance==TIM1)
  {
  /* USER CODE BEGIN TIM1_MspDeInit 0 */

  /* USER CODE END TIM1_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_TIM1_CLK_DISABLE();
  /* USER CODE BEGIN TIM1_MspDeInit 1 */

  /* USER CODE END TIM1_MspDeInit 1 */
  }

}

void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(huart->Instance==USART1)
  {
  /* USER CODE BEGIN USART1_MspInit 0 */

  /* USER CODE END USART1_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_USART1_CLK_ENABLE();

    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = GPIO_PIN_9;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_10;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* USER CODE BEGIN USART1_MspInit 1 */

  /* USER CODE END USART1_MspInit 1 */
  }

}

void HAL_UART_MspDeInit(UART_HandleTypeDef* huart)
{
  if(huart->Instance==USART1)
  {
  /* USER CODE BEGIN USART1_MspDeInit 0 */

  /* USER CODE END USART1_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_USART1_CLK_DISABLE();

    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_9|GPIO_PIN_10);

  /* USER CODE BEGIN USART1_MspDeInit 1 */

  /* USER CODE END USART1_MspDeInit 1 */
  }

}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

4.8 Referência do Arquivo D:/BACKUP/Faculdade/16_Embarcados/Controlador_Temperatura/↩ Controle-de-temperatura/Codigos/controle_temperatura_RTOS/↩ Core/Src/stm32f1xx_hal_timebase_tim.c

HAL time base based on the hardware TIM.

```
#include "stm32f1xx_hal.h"
#include "stm32f1xx_hal_tim.h"
```

Funções

- HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)

  This function configures the TIM4 as a time base source. The time source is configured to have 1ms time base with a dedicated Tick interrupt priority.
- void HAL_SuspendTick (void)

  Suspend Tick increment.
- void HAL_ResumeTick (void)

  Resume Tick increment.

Variáveis

- TIM_HandleTypeDef htim4

4.8.1 Descrição detalhada

HAL time base based on the hardware TIM.

Atenção

Definição no arquivo stm32f1xx_hal_timebase_tim.c.

4.8.2 Funções

4.8.2.1 HAL_InitTick()

```
HAL_StatusTypeDef HAL_InitTick (
             uint32_t TickPriority )
```

This function configures the TIM4 as a time base source. The time source is configured to have 1ms time base with a dedicated Tick interrupt priority.

Observação

> This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is configured, by HAL_RCC_ClockConfig().

Parâmetros

TickPriority	Tick interrupt priority.

Valores Retornados

HAL	status

Definição na linha 42 do arquivo stm32f1xx_hal_timebase_tim.c.

```
00043 {
00044   RCC_ClkInitTypeDef    clkconfig;
00045   uint32_t              uwTimclock = 0;
00046   uint32_t              uwPrescalerValue = 0;
00047   uint32_t              pFLatency;
00048   /*Configure the TIM4 IRQ priority */
00049   HAL_NVIC_SetPriority(TIM4_IRQn, TickPriority ,0);
00050 
00051   /* Enable the TIM4 global Interrupt */
00052   HAL_NVIC_EnableIRQ(TIM4_IRQn);
00053   /* Enable TIM4 clock */
00054   __HAL_RCC_TIM4_CLK_ENABLE();
00055 
00056   /* Get clock configuration */
00057   HAL_RCC_GetClockConfig(&clkconfig, &pFLatency);
00058 
00059   /* Compute TIM4 clock */
00060   uwTimclock = 2*HAL_RCC_GetPCLK1Freq();
00061   /* Compute the prescaler value to have TIM4 counter clock equal to 1MHz */
00062   uwPrescalerValue = (uint32_t) ((uwTimclock / 1000000U) - 1U);
00063 
00064   /* Initialize TIM4 */
00065   htim4.Instance = TIM4;
00066 
00067   /* Initialize TIMx peripheral as follow:
00068   + Period = [(TIM4CLK/1000) - 1]. to have a (1/1000) s time base.
00069   + Prescaler = (uwTimclock/1000000 - 1) to have a 1MHz counter clock.
00070   + ClockDivision = 0
00071   + Counter direction = Up
00072   */
00073   htim4.Init.Period = (1000000U / 1000U) - 1U;
00074   htim4.Init.Prescaler = uwPrescalerValue;
00075   htim4.Init.ClockDivision = 0;
00076   htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
00077   if(HAL_TIM_Base_Init(&htim4) == HAL_OK)
00078   {
00079     /* Start the TIM time Base generation in interrupt mode */
00080     return HAL_TIM_Base_Start_IT(&htim4);
00081   }
00082 
00083   /* Return function status */
00084   return HAL_ERROR;
00085 }
```

4.8.2.2 HAL_ResumeTick()

```
void HAL_ResumeTick (
             void  )
```

Resume Tick increment.

Observação

Enable the tick increment by Enabling TIM4 update interrupt.

Parâmetros

None	

Valores Retornados

None	

Definição na linha 105 do arquivo stm32f1xx_hal_timebase_tim.c.

```
00106 {
00107   /* Enable TIM4 Update interrupt */
00108   __HAL_TIM_ENABLE_IT(&htim4, TIM_IT_UPDATE);
00109 }
```

4.8.2.3 HAL_SuspendTick()

```
void HAL_SuspendTick (
             void  )
```

Suspend Tick increment.

Observação

Disable the tick increment by disabling TIM4 update interrupt.

Parâmetros

None	

Valores Retornados

None	

Definição na linha 93 do arquivo stm32f1xx_hal_timebase_tim.c.

```
00094 {
00095   /* Disable TIM4 update Interrupt */
00096   __HAL_TIM_DISABLE_IT(&htim4, TIM_IT_UPDATE);
00097 }
```

4.8.3 Variáveis

4.8.3.1 htim4

```
TIM_HandleTypeDef htim4
```

Definição na linha 29 do arquivo stm32f1xx_hal_timebase_tim.c.

4.9 stm32f1xx_hal_timebase_tim.c

Vá para a documentação desse arquivo.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Includes ------------------------------------------------------------------*/
00022 #include "stm32f1xx_hal.h"
00023 #include "stm32f1xx_hal_tim.h"
00024
00025 /* Private typedef -----------------------------------------------------------*/
00026 /* Private define ------------------------------------------------------------*/
00027 /* Private macro -------------------------------------------------------------*/
00028 /* Private variables ---------------------------------------------------------*/
00029 TIM_HandleTypeDef        htim4;
00030 /* Private function prototypes -----------------------------------------------*/
00031 /* Private functions ---------------------------------------------------------*/
00032
00042 HAL_StatusTypeDef HAL_InitTick(uint32_t TickPriority)
00043 {
00044   RCC_ClkInitTypeDef    clkconfig;
00045   uint32_t              uwTimclock = 0;
00046   uint32_t              uwPrescalerValue = 0;
00047   uint32_t              pFLatency;
00048   /*Configure the TIM4 IRQ priority */
00049   HAL_NVIC_SetPriority(TIM4_IRQn, TickPriority ,0);
00050
00051   /* Enable the TIM4 global Interrupt */
00052   HAL_NVIC_EnableIRQ(TIM4_IRQn);
00053   /* Enable TIM4 clock */
00054   __HAL_RCC_TIM4_CLK_ENABLE();
00055
00056   /* Get clock configuration */
00057   HAL_RCC_GetClockConfig(&clkconfig, &pFLatency);
00058
00059   /* Compute TIM4 clock */
00060   uwTimclock = 2*HAL_RCC_GetPCLK1Freq();
00061   /* Compute the prescaler value to have TIM4 counter clock equal to 1MHz */
00062   uwPrescalerValue = (uint32_t) ((uwTimclock / 1000000U) - 1U);
00063
00064   /* Initialize TIM4 */
00065   htim4.Instance = TIM4;
00066
00067   /* Initialize TIMx peripheral as follow:
00068   + Period = [(TIM4CLK/1000) - 1]. to have a (1/1000) s time base.
00069   + Prescaler = (uwTimclock/1000000 - 1) to have a 1MHz counter clock.
00070   + ClockDivision = 0
00071   + Counter direction = Up
00072   */
00073   htim4.Init.Period = (1000000U / 1000U) - 1U;
00074   htim4.Init.Prescaler = uwPrescalerValue;
00075   htim4.Init.ClockDivision = 0;
00076   htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
00077   if(HAL_TIM_Base_Init(&htim4) == HAL_OK)
00078   {
00079     /* Start the TIM time Base generation in interrupt mode */
00080     return HAL_TIM_Base_Start_IT(&htim4);
00081   }
00082
00083   /* Return function status */
00084   return HAL_ERROR;
00085 }
00086
00093 void HAL_SuspendTick(void)
```

```
{
  /* Disable TIM4 update Interrupt */
  __HAL_TIM_DISABLE_IT(&htim4, TIM_IT_UPDATE);
}

void HAL_ResumeTick(void)
{
  /* Enable TIM4 Update interrupt */
  __HAL_TIM_ENABLE_IT(&htim4, TIM_IT_UPDATE);
}

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

4.10 Referência do Arquivo D:/BACKUP/Faculdade/16_Embarcados/Controlador_Temperatura/Controle-de-temperatura/Codigos/controle_temperatura_RTOS/Core/Src/stm32f1xx_it.c

Interrupt Service Routines.

```
#include "main.h"
#include "stm32f1xx_it.h"
```

Funções

- void NMI_Handler (void)

  This function handles Non maskable interrupt.
- void HardFault_Handler (void)

  This function handles Hard fault interrupt.
- void MemManage_Handler (void)

  This function handles Memory management fault.
- void BusFault_Handler (void)

  This function handles Prefetch fault, memory access fault.
- void UsageFault_Handler (void)

  This function handles Undefined instruction or illegal state.
- void DebugMon_Handler (void)

  This function handles Debug monitor.
- void TIM4_IRQHandler (void)

  This function handles TIM4 global interrupt.

Variáveis

- TIM_HandleTypeDef htim4

4.10.1 Descrição detalhada

Interrupt Service Routines.

Atenção

Definição no arquivo stm32f1xx_it.c.

4.10.2 Funções

4.10.2.1 BusFault_Handler()

```
void BusFault_Handler (
             void  )
```

This function handles Prefetch fault, memory access fault.

Definição na linha 116 do arquivo stm32f1xx_it.c.

```
{
  /* USER CODE BEGIN BusFault_IRQn 0 */

  /* USER CODE END BusFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_BusFault_IRQn 0 */
    /* USER CODE END W1_BusFault_IRQn 0 */
  }
}
```

4.10.2.2 DebugMon_Handler()

```
void DebugMon_Handler (
             void  )
```

This function handles Debug monitor.

Definição na linha 146 do arquivo stm32f1xx_it.c.

```
{
  /* USER CODE BEGIN DebugMonitor_IRQn 0 */

  /* USER CODE END DebugMonitor_IRQn 0 */
  /* USER CODE BEGIN DebugMonitor_IRQn 1 */

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

4.10.2.3 HardFault_Handler()

```
void HardFault_Handler (
            void )
```

This function handles Hard fault interrupt.

Definição na linha 86 do arquivo stm32f1xx_it.c.

```
00087 {
00088   /* USER CODE BEGIN HardFault_IRQn 0 */
00089 
00090   /* USER CODE END HardFault_IRQn 0 */
00091   while (1)
00092   {
00093     /* USER CODE BEGIN W1_HardFault_IRQn 0 */
00094     /* USER CODE END W1_HardFault_IRQn 0 */
00095   }
00096 }
```

4.10.2.4 MemManage_Handler()

```
void MemManage_Handler (
            void )
```

This function handles Memory management fault.

Definição na linha 101 do arquivo stm32f1xx_it.c.

```
00102 {
00103   /* USER CODE BEGIN MemoryManagement_IRQn 0 */
00104 
00105   /* USER CODE END MemoryManagement_IRQn 0 */
00106   while (1)
00107   {
00108     /* USER CODE BEGIN W1_MemoryManagement_IRQn 0 */
00109     /* USER CODE END W1_MemoryManagement_IRQn 0 */
00110   }
00111 }
```

4.10.2.5 NMI_Handler()

```
void NMI_Handler (
            void )
```

This function handles Non maskable interrupt.

Definição na linha 71 do arquivo stm32f1xx_it.c.

```
00072 {
00073   /* USER CODE BEGIN NonMaskableInt_IRQn 0 */
00074 
00075   /* USER CODE END NonMaskableInt_IRQn 0 */
00076   /* USER CODE BEGIN NonMaskableInt_IRQn 1 */
00077   while (1)
00078   {
00079   }
00080   /* USER CODE END NonMaskableInt_IRQn 1 */
00081 }
```

4.10.2.6 TIM4_IRQHandler()

```
void TIM4_IRQHandler (
            void  )
```

This function handles TIM4 global interrupt.

Definição na linha 166 do arquivo stm32f1xx_it.c.

```
{
  /* USER CODE BEGIN TIM4_IRQn 0 */

  /* USER CODE END TIM4_IRQn 0 */
  HAL_TIM_IRQHandler(&htim4);
  /* USER CODE BEGIN TIM4_IRQn 1 */

  /* USER CODE END TIM4_IRQn 1 */
}
```

4.10.2.7 UsageFault_Handler()

```
void UsageFault_Handler (
            void  )
```

This function handles Undefined instruction or illegal state.

Definição na linha 131 do arquivo stm32f1xx_it.c.

```
{
  /* USER CODE BEGIN UsageFault_IRQn 0 */

  /* USER CODE END UsageFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_UsageFault_IRQn 0 */
    /* USER CODE END W1_UsageFault_IRQn 0 */
  }
}
```

4.10.3 Variáveis

4.10.3.1 htim4

```
TIM_HandleTypeDef htim4  [extern]
```

Definição na linha 29 do arquivo stm32f1xx_hal_timebase_tim.c.

4.11 stm32f1xx_it.c

Vá para a documentação desse arquivo.

```
/* USER CODE BEGIN Header */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f1xx_it.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN TD */

/* USER CODE END TD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* 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 -----------------------------------------------*/
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/* External variables --------------------------------------------------------*/
extern TIM_HandleTypeDef htim4;

/* USER CODE BEGIN EV */

/* USER CODE END EV */

/******************************************************************************/
/*           Cortex-M3 Processor Interruption and Exception Handlers          */
/******************************************************************************/
void NMI_Handler(void)
{
  /* USER CODE BEGIN NonMaskableInt_IRQn 0 */

  /* USER CODE END NonMaskableInt_IRQn 0 */
  /* USER CODE BEGIN NonMaskableInt_IRQn 1 */
  while (1)
  {
  }
  /* USER CODE END NonMaskableInt_IRQn 1 */
}

void HardFault_Handler(void)
{
  /* USER CODE BEGIN HardFault_IRQn 0 */

  /* USER CODE END HardFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_HardFault_IRQn 0 */
    /* USER CODE END W1_HardFault_IRQn 0 */
  }
}

void MemManage_Handler(void)
{
  /* USER CODE BEGIN MemoryManagement_IRQn 0 */

  /* USER CODE END MemoryManagement_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_MemoryManagement_IRQn 0 */
    /* USER CODE END W1_MemoryManagement_IRQn 0 */
  }
}

void BusFault_Handler(void)
{
  /* USER CODE BEGIN BusFault_IRQn 0 */

  /* USER CODE END BusFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_BusFault_IRQn 0 */
    /* USER CODE END W1_BusFault_IRQn 0 */
  }
}

void UsageFault_Handler(void)
{
  /* USER CODE BEGIN UsageFault_IRQn 0 */

  /* USER CODE END UsageFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_UsageFault_IRQn 0 */
    /* USER CODE END W1_UsageFault_IRQn 0 */
  }
}

void DebugMon_Handler(void)
{
  /* USER CODE BEGIN DebugMonitor_IRQn 0 */

  /* USER CODE END DebugMonitor_IRQn 0 */
  /* USER CODE BEGIN DebugMonitor_IRQn 1 */

  /* USER CODE END DebugMonitor_IRQn 1 */
}

/******************************************************************************/
/* STM32F1xx Peripheral Interrupt Handlers                                    */
/* Add here the Interrupt Handlers for the used peripherals.                  */
/* For the available peripheral interrupt handler names,                      */
/* please refer to the startup file (startup_stm32f1xx.s).                    */
/******************************************************************************/

void TIM4_IRQHandler(void)
{
  /* USER CODE BEGIN TIM4_IRQn 0 */

  /* USER CODE END TIM4_IRQn 0 */
  HAL_TIM_IRQHandler(&htim4);
  /* USER CODE BEGIN TIM4_IRQn 1 */

  /* USER CODE END TIM4_IRQn 1 */
}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

4.12 Referência do Arquivo D:/BACKUP/Faculdade/16_Embarcados/Controlador_Temperatura/Controle-de-temperatura/Codigos/controle_temperatura_RTOS/Core/Src/syscalls.c

STM32CubeIDE Minimal System calls file.

```
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <signal.h>
#include <time.h>
#include <sys/time.h>
#include <sys/times.h>
```

Funções

- int **__io_putchar** (int ch) __attribute__((weak))
- int __io_getchar (void)
- void initialise_monitor_handles ()
- int _getpid (void)
- int _kill (int pid, int sig)
- void _exit (int status)
- __attribute__ ((weak))
- int _close (int file)
- int _fstat (int file, struct stat ∗st)
- int _isatty (int file)
- int _lseek (int file, int ptr, int dir)
- int _open (char ∗path, int flags,...)
- int _wait (int ∗status)
- int _unlink (char ∗name)
- int _times (struct tms ∗buf)
- int _stat (char ∗file, struct stat ∗st)
- int _link (char ∗old, char ∗new)
- int _fork (void)
- int _execve (char ∗name, char ∗∗argv, char ∗∗env)

Variáveis

- int **errno**
- char ∗∗ environ = __env

4.12.1 Descrição detalhada

STM32CubeIDE Minimal System calls file.

Autor

Auto-generated by STM32CubeIDE

```
For more information about which c-functions
need which of these lowlevel functions
please consult the Newlib libc-manual
```

Atenção

Definição no arquivo syscalls.c.

4.12.2 Funções

4.12.2.1 __attribute__()

```
__attribute__ (
            (weak)  )
```

Definição na linha 69 do arquivo syscalls.c.

```
{
    int DataIdx;

    for (DataIdx = 0; DataIdx < len; DataIdx++)
    {
        *ptr++ = __io_getchar();
    }

return len;
}
```

4.12.2.2 __io_getchar()

```
int __io_getchar (
            void  )
```

Definição na linha 39 do arquivo syscalls.c.

```
                  { 0 };
```

4.12.2.3 _close()

```
int _close (
            int file )
```

Definição na linha 92 do arquivo syscalls.c.

```
{
    return -1;
}
```

4.12.2.4 _execve()

```
int _execve (
            char * name,
            char ** argv,
            char ** env )
```

Definição na linha 155 do arquivo syscalls.c.

```
{
    errno = ENOMEM;
    return -1;
}
```

4.12.2.5 _exit()

```
void _exit (
            int status )
```

Definição na linha 63 do arquivo syscalls.c.

```
{
    _kill(status, -1);
    while (1) {}        /* Make sure we hang here */
}
```

4.12.2.6 _fork()

```
int _fork (
            void  )
```

Definição na linha 149 do arquivo syscalls.c.

```
{
    errno = EAGAIN;
    return -1;
}
```

4.12.2.7 _fstat()

```
int _fstat (
            int file,
            struct stat * st )
```

Definição na linha 98 do arquivo syscalls.c.

```
{
    st->st_mode = S_IFCHR;
    return 0;
}
```

4.12.2.8 _getpid()

```
int _getpid (
            void  )
```

Definição na linha 52 do arquivo syscalls.c.

```
{
    return 1;
}
```

4.12.2.9 _isatty()

```
int _isatty (
             int file )
```

Definição na linha 104 do arquivo syscalls.c.

```
00105 {
00106     return 1;
00107 }
```

4.12.2.10 _kill()

```
int _kill (
             int pid,
             int sig )
```

Definição na linha 57 do arquivo syscalls.c.

```
00058 {
00059     errno = EINVAL;
00060     return -1;
00061 }
```

4.12.2.11 _link()

```
int _link (
             char * old,
             char * new )
```

Definição na linha 143 do arquivo syscalls.c.

```
00144 {
00145     errno = EMLINK;
00146     return -1;
00147 }
```

4.12.2.12 _lseek()

```
int _lseek (
             int file,
             int ptr,
             int dir )
```

Definição na linha 109 do arquivo syscalls.c.

```
00110 {
00111     return 0;
00112 }
```

4.12.2.13 _open()

```
int _open (
            char * path,
            int flags,
             ... )
```

Definição na linha 114 do arquivo syscalls.c.

```
{
    /* Pretend like we always fail */
    return -1;
}
```

4.12.2.14 _stat()

```
int _stat (
            char * file,
            struct stat * st )
```

Definição na linha 137 do arquivo syscalls.c.

```
{
    st->st_mode = S_IFCHR;
    return 0;
}
```

4.12.2.15 _times()

```
int _times (
            struct tms * buf )
```

Definição na linha 132 do arquivo syscalls.c.

```
{
    return -1;
}
```

4.12.2.16 _unlink()

```
int _unlink (
            char * name )
```

Definição na linha 126 do arquivo syscalls.c.

```
{
    errno = ENOENT;
    return -1;
}
```

4.12.2.17 _wait()

```
int _wait (
            int * status )
```

Definição na linha 120 do arquivo syscalls.c.

```
00121 {
00122     errno = ECHILD;
00123     return -1;
00124 }
```

4.12.2.18 initialise_monitor_handles()

```
void initialise_monitor_handles ( )
```

Definição na linha 48 do arquivo syscalls.c.

```
00049 {
00050 }
```

4.12.3 Variáveis

4.12.3.1 environ

```
char** environ = __env
```

Definição na linha 44 do arquivo syscalls.c.

4.13 syscalls.c

Vá para a documentação desse arquivo.

```
00001
00024 /* Includes */
00025 #include <sys/stat.h>
00026 #include <stdlib.h>
00027 #include <errno.h>
00028 #include <stdio.h>
00029 #include <signal.h>
00030 #include <time.h>
00031 #include <sys/time.h>
00032 #include <sys/times.h>
00033
00034
00035 /* Variables */
00036 //#undef errno
00037 extern int errno;
00038 extern int __io_putchar(int ch) __attribute__((weak));
00039 extern int __io_getchar(void) __attribute__((weak));
00040
00041 register char * stack_ptr asm("sp");
00042
00043 char *__env[1] = { 0 };
00044 char **environ = __env;
00045
00046
00047 /* Functions */
00048 void initialise_monitor_handles()
00049 {
}

int _getpid(void)
{
    return 1;
}

int _kill(int pid, int sig)
{
    errno = EINVAL;
    return -1;
}

void _exit (int status)
{
    _kill(status, -1);
    while (1) {}        /* Make sure we hang here */
}

__attribute__((weak)) int _read(int file, char *ptr, int len)
{
    int DataIdx;

    for (DataIdx = 0; DataIdx < len; DataIdx++)
    {
        *ptr++ = __io_getchar();
    }

return len;
}

__attribute__((weak)) int _write(int file, char *ptr, int len)
{
    int DataIdx;

    for (DataIdx = 0; DataIdx < len; DataIdx++)
    {
        __io_putchar(*ptr++);
    }
    return len;
}

int _close(int file)
{
    return -1;
}


int _fstat(int file, struct stat *st)
{
    st->st_mode = S_IFCHR;
    return 0;
}

int _isatty(int file)
{
    return 1;
}

int _lseek(int file, int ptr, int dir)
{
    return 0;
}

int _open(char *path, int flags, ...)
{
    /* Pretend like we always fail */
    return -1;
}

int _wait(int *status)
{
    errno = ECHILD;
    return -1;
}

int _unlink(char *name)
{
    errno = ENOENT;
    return -1;
}

int _times(struct tms *buf)
{
    return -1;
}

```

```
int _stat(char *file, struct stat *st)
{
    st->st_mode = S_IFCHR;
    return 0;
}

int _link(char *old, char *new)
{
    errno = EMLINK;
    return -1;
}

int _fork(void)
{
    errno = EAGAIN;
    return -1;
}

int _execve(char *name, char **argv, char **env)
{
    errno = ENOMEM;
    return -1;
}
```

4.14 Referência do Arquivo D:/BACKUP/Faculdade/16_Embarcados/Controlador_Temperatura/Controle-de-temperatura/Codigos/controle_temperatura_RTOS/Core/Src/sysmem.c

STM32CubeIDE System Memory calls file.

```
#include <errno.h>
#include <stdint.h>
```

Funções

- void * _sbrk (ptrdiff_t incr)

  _sbrk() allocates memory to the newlib heap and is used by malloc and others from the C library

4.14.1 Descrição detalhada

STM32CubeIDE System Memory calls file.

Autor

Generated by STM32CubeIDE

```
For more information about which C functions
need which of these lowlevel functions
please consult the newlib libc manual
```

Atenção

Definição no arquivo sysmem.c.

4.14.2 Funções

4.14.2.1 _sbrk()

```
void * _sbrk (
            ptrdiff_t incr )
```

_sbrk() allocates memory to the newlib heap and is used by malloc and others from the C library

```
* ###############################################################################
* #   .data   #   .bss   #       newlib heap       #          MSP stack          #
* #           #          #                         # Reserved by _Min_Stack_Size #
* ###############################################################################
* ^-- RAM start      ^-- _end                             _estack, RAM end --^
*
```

This implementation starts allocating at the '_end' linker symbol The '_Min_Stack_Size' linker symbol reserves a memory for the MSP stack The implementation considers '_estack' linker symbol to be RAM end NOTE: If the MSP stack, at any point during execution, grows larger than the reserved size, please increase the '_Min_Stack_Size'.

Parâmetros

incr	Memory size

Retorna

Pointer to allocated memory

Definição na linha 54 do arquivo sysmem.c.

```
00055 {
00056   extern uint8_t _end; /* Symbol defined in the linker script */
00057   extern uint8_t _estack; /* Symbol defined in the linker script */
00058   extern uint32_t _Min_Stack_Size; /* Symbol defined in the linker script */
00059   const uint32_t stack_limit = (uint32_t)&_estack - (uint32_t)&_Min_Stack_Size;
00060   const uint8_t *max_heap = (uint8_t *)stack_limit;
00061   uint8_t *prev_heap_end;
00062 
00063   /* Initialize heap end at first call */
00064   if (NULL == __sbrk_heap_end)
00065   {
00066     __sbrk_heap_end = &_end;
00067   }
```

```
  /* Protect heap from growing into the reserved MSP stack */
  if (__sbrk_heap_end + incr > max_heap)
  {
    errno = ENOMEM;
    return (void *)-1;
  }

  prev_heap_end = __sbrk_heap_end;
  __sbrk_heap_end += incr;

  return (void *)prev_heap_end;
}
```

4.15 sysmem.c

Vá para a documentação desse arquivo.

```
/* Includes */
#include <errno.h>
#include <stdint.h>

static uint8_t *__sbrk_heap_end = NULL;

void *_sbrk(ptrdiff_t incr)
{
  extern uint8_t _end; /* Symbol defined in the linker script */
  extern uint8_t _estack; /* Symbol defined in the linker script */
  extern uint32_t _Min_Stack_Size; /* Symbol defined in the linker script */
  const uint32_t stack_limit = (uint32_t)&_estack - (uint32_t)&_Min_Stack_Size;
  const uint8_t *max_heap = (uint8_t *)stack_limit;
  uint8_t *prev_heap_end;

  /* Initialize heap end at first call */
  if (NULL == __sbrk_heap_end)
  {
    __sbrk_heap_end = &_end;
  }

  /* Protect heap from growing into the reserved MSP stack */
  if (__sbrk_heap_end + incr > max_heap)
  {
    errno = ENOMEM;
    return (void *)-1;
  }

  prev_heap_end = __sbrk_heap_end;
  __sbrk_heap_end += incr;

  return (void *)prev_heap_end;
}
```

4.16 Referência do Arquivo D:/BACKUP/Faculdade/16_Embarcados/Controlador_Temperatura/Controle-de-temperatura/Codigos/controle_temperatura_RTOS/Core/Src/system_stm32f1xx.c

CMSIS Cortex-M3 Device Peripheral Access Layer System Source File.

```
#include "stm32f1xx.h"
```

Definições e Macros

- #define HSE_VALUE 8000000U
- #define HSI_VALUE 8000000U

Funções

- void SystemInit (void)

  Setup the microcontroller system Initialize the Embedded Flash Interface, the PLL and update the SystemCoreClock variable.
- void SystemCoreClockUpdate (void)

  Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

Variáveis

- uint32_t SystemCoreClock = 16000000
- const uint8_t AHBPrescTable [16U] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}
- const uint8_t APBPrescTable [8U] = {0, 0, 0, 0, 1, 2, 3, 4}

4.16.1 Descrição detalhada

CMSIS Cortex-M3 Device Peripheral Access Layer System Source File.

Autor

MCD Application Team

1. This file provides two functions and one global variable to be called from user application:
   - SystemInit(): Setups the system clock (System clock source, PLL Multiplier factors, AHB/APBx prescalers and Flash settings). This function is called at startup just after reset and before branch to main program. This call is made inside the "startup_stm32f1xx_xx.s" file.
   - SystemCoreClock variable: Contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.
   - SystemCoreClockUpdate(): Updates the variable SystemCoreClock and must be called whenever the core clock is changed during program execution.
2. After each device reset the HSI (8 MHz) is used as system clock source. Then SystemInit() function is called, in "startup_stm32f1xx_xx.s" file, to configure the system clock before to branch to main program.
3. The default value of HSE crystal is set to 8 MHz (or 25 MHz, depending on the product used), refer to "HSE↩_VALUE". When HSE is used as system clock source, directly or through PLL, and you are using different crystal you have to adapt the HSE value to your own configuration.

Atenção

Definição no arquivo system_stm32f1xx.c.

4.17 system_stm32f1xx.c

Vá para a documentação desse arquivo.

```
#include "stm32f1xx.h"

#if !defined  (HSE_VALUE)
  #define HSE_VALUE               8000000U
#endif /* HSE_VALUE */

#if !defined  (HSI_VALUE)
  #define HSI_VALUE               8000000U
#endif /* HSI_VALUE */

#if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) || defined(STM32F103xE) ||
 defined(STM32F103xG)
/* #define DATA_IN_ExtSRAM */
#endif /* STM32F100xE || STM32F101xE || STM32F101xG || STM32F103xE || STM32F103xG */

/* Note: Following vector table addresses must be defined in line with linker
         configuration. */
/* #define USER_VECT_TAB_ADDRESS */

#if defined(USER_VECT_TAB_ADDRESS)
/* #define VECT_TAB_SRAM */
#if defined(VECT_TAB_SRAM)
#define VECT_TAB_BASE_ADDRESS   SRAM_BASE
#define VECT_TAB_OFFSET         0x00000000U
#else
#define VECT_TAB_BASE_ADDRESS   FLASH_BASE
#define VECT_TAB_OFFSET         0x00000000U
#endif /* VECT_TAB_SRAM */
#endif /* USER_VECT_TAB_ADDRESS */

/******************************************************************************/

  /* This variable is updated in three ways:
      1) by calling CMSIS function SystemCoreClockUpdate()
      2) by calling HAL API function HAL_RCC_GetHCLKFreq()
      3) each time HAL_RCC_ClockConfig() is called to configure the system clock frequency
         Note: If you use this function to configure the system clock; then there
               is no need to call the 2 first functions listed above, since SystemCoreClock
               variable is updated automatically.
  */
uint32_t SystemCoreClock = 16000000;
const uint8_t AHBPrescTable[16U] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9};
const uint8_t APBPrescTable[8U] =  {0, 0, 0, 0, 1, 2, 3, 4};

#if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) || defined(STM32F103xE) ||
 defined(STM32F103xG)
#ifdef DATA_IN_ExtSRAM
  static void SystemInit_ExtMemCtl(void);
#endif /* DATA_IN_ExtSRAM */
#endif /* STM32F100xE || STM32F101xE || STM32F101xG || STM32F103xE || STM32F103xG */

void SystemInit (void)
{
#if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) || defined(STM32F103xE) ||
 defined(STM32F103xG)
  #ifdef DATA_IN_ExtSRAM
    SystemInit_ExtMemCtl();
  #endif /* DATA_IN_ExtSRAM */
#endif

  /* Configure the Vector Table location -------------------------------------*/
#if defined(USER_VECT_TAB_ADDRESS)
  SCB->VTOR = VECT_TAB_BASE_ADDRESS | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM. */
#endif /* USER_VECT_TAB_ADDRESS */
}

void SystemCoreClockUpdate (void)
{
  uint32_t tmp = 0U, pllmull = 0U, pllsource = 0U;

#if defined(STM32F105xC) || defined(STM32F107xC)
  uint32_t prediv1source = 0U, prediv1factor = 0U, prediv2factor = 0U, pll2mull = 0U;
#endif /* STM32F105xC */

#if defined(STM32F100xB) || defined(STM32F100xE)
  uint32_t prediv1factor = 0U;
#endif /* STM32F100xB or STM32F100xE */

  /* Get SYSCLK source -------------------------------------------------------*/
  tmp = RCC->CFGR & RCC_CFGR_SWS;

  switch (tmp)
  {
    case 0x00U:  /* HSI used as system clock */
      SystemCoreClock = HSI_VALUE;
      break;
    case 0x04U:  /* HSE used as system clock */
      SystemCoreClock = HSE_VALUE;
      break;
    case 0x08U:  /* PLL used as system clock */

      /* Get PLL clock source and multiplication factor ----------------------*/
      pllmull = RCC->CFGR & RCC_CFGR_PLLMULL;
      pllsource = RCC->CFGR & RCC_CFGR_PLLSRC;

#if !defined(STM32F105xC) && !defined(STM32F107xC)
      pllmull = ( pllmull » 18U) + 2U;

      if (pllsource == 0x00U)
      {
        /* HSI oscillator clock divided by 2 selected as PLL clock entry */
        SystemCoreClock = (HSI_VALUE » 1U) * pllmull;
      }
      else
      {
 #if defined(STM32F100xB) || defined(STM32F100xE)
       prediv1factor = (RCC->CFGR2 & RCC_CFGR2_PREDIV1) + 1U;
       /* HSE oscillator clock selected as PREDIV1 clock entry */
       SystemCoreClock = (HSE_VALUE / prediv1factor) * pllmull;
 #else
        /* HSE selected as PLL clock entry */
        if ((RCC->CFGR & RCC_CFGR_PLLXTPRE) != (uint32_t)RESET)
        {/* HSE oscillator clock divided by 2 */
          SystemCoreClock = (HSE_VALUE » 1U) * pllmull;
        }
        else
        {
          SystemCoreClock = HSE_VALUE * pllmull;
        }
 #endif
      }
#else
      pllmull = pllmull » 18U;

      if (pllmull != 0x0DU)
      {
         pllmull += 2U;
      }
      else
      { /* PLL multiplication factor = PLL input clock * 6.5 */
        pllmull = 13U / 2U;
      }

      if (pllsource == 0x00U)
      {
        /* HSI oscillator clock divided by 2 selected as PLL clock entry */
        SystemCoreClock = (HSI_VALUE » 1U) * pllmull;
      }
      else
      {/* PREDIV1 selected as PLL clock entry */

        /* Get PREDIV1 clock source and division factor */
        prediv1source = RCC->CFGR2 & RCC_CFGR2_PREDIV1SRC;
        prediv1factor = (RCC->CFGR2 & RCC_CFGR2_PREDIV1) + 1U;

        if (prediv1source == 0U)
        {
          /* HSE oscillator clock selected as PREDIV1 clock entry */
          SystemCoreClock = (HSE_VALUE / prediv1factor) * pllmull;
        }
        else
        {/* PLL2 clock selected as PREDIV1 clock entry */

          /* Get PREDIV2 division factor and PLL2 multiplication factor */
          prediv2factor = ((RCC->CFGR2 & RCC_CFGR2_PREDIV2) » 4U) + 1U;
          pll2mull = ((RCC->CFGR2 & RCC_CFGR2_PLL2MUL) » 8U) + 2U;
          SystemCoreClock = (((HSE_VALUE / prediv2factor) * pll2mull) / prediv1factor) * pllmull;

        }
      }
#endif /* STM32F105xC */
      break;

    default:
      SystemCoreClock = HSI_VALUE;
      break;
  }

  /* Compute HCLK clock frequency ----------------*/
  /* Get HCLK prescaler */
  tmp = AHBPrescTable[((RCC->CFGR & RCC_CFGR_HPRE) » 4U)];
  /* HCLK clock frequency */
  SystemCoreClock »= tmp;
}

#if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) || defined(STM32F103xE) || defined(STM32F103xG)
#ifdef DATA_IN_ExtSRAM
void SystemInit_ExtMemCtl(void)
{
  __IO uint32_t tmpreg;
  /* Enable FSMC clock */
  RCC->AHBENR = 0x00000114U;

  /* Delay after an RCC peripheral clock enabling */
  tmpreg = READ_BIT(RCC->AHBENR, RCC_AHBENR_FSMCEN);

  /* Enable GPIOD, GPIOE, GPIOF and GPIOG clocks */
  RCC->APB2ENR = 0x000001E0U;

  /* Delay after an RCC peripheral clock enabling */
  tmpreg = READ_BIT(RCC->APB2ENR, RCC_APB2ENR_IOPDEN);

  (void)(tmpreg);

/* ---------------  SRAM Data lines, NOE and NWE configuration ---------------*/
/*----------------  SRAM Address lines configuration -------------------------*/
/*----------------  NOE and NWE configuration --------------------------------*/
/*----------------  NE3 configuration ----------------------------------------*/
/*----------------  NBL0, NBL1 configuration ---------------------------------*/

  GPIOD->CRL = 0x44BB44BBU;
  GPIOD->CRH = 0xBBBBBBBBU;

  GPIOE->CRL = 0xB44444BBU;
  GPIOE->CRH = 0xBBBBBBBBU;

  GPIOF->CRL = 0x44BBBBBBU;
  GPIOF->CRH = 0xBBBB4444U;

  GPIOG->CRL = 0x44BBBBBBU;
  GPIOG->CRH = 0x444B4B44U;

/*----------------  FSMC Configuration ---------------------------------------*/
/*----------------  Enable FSMC Bank1_SRAM Bank ------------------------------*/

  FSMC_Bank1->BTCR[4U] = 0x00001091U;
  FSMC_Bank1->BTCR[5U] = 0x00110212U;
}
#endif /* DATA_IN_ExtSRAM */
#endif /* STM32F100xE || STM32F101xE || STM32F101xG || STM32F103xE || STM32F103xG */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Índice Remissivo