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.

```
00226 {
00227   uint32_t tmp = 0U, pllmull = 0U, pllsource = 0U;
00228
00229 #if defined(STM32F105xC) || defined(STM32F107xC)
00230   uint32_t prediv1source = 0U, prediv1factor = 0U, prediv2factor = 0U, pll2mull = 0U;
00231 #endif /* STM32F105xC */
00232
00233 #if defined(STM32F100xB) || defined(STM32F100xE)
00234   uint32_t prediv1factor = 0U;
00235 #endif /* STM32F100xB or STM32F100xE */
00236
00237   /* Get SYSCLK source -------------------------------------------------------*/
00238   tmp = RCC->CFGR & RCC_CFGR_SWS;
00239
00240   switch (tmp)
00241   {
00242     case 0x00U:  /* HSI used as system clock */
00243       SystemCoreClock = HSI_VALUE;
00244       break;
00245     case 0x04U:  /* HSE used as system clock */
00246       SystemCoreClock = HSE_VALUE;
00247       break;
00248     case 0x08U:  /* PLL used as system clock */
00249
00250       /* Get PLL clock source and multiplication factor ----------------------*/
00251       pllmull = RCC->CFGR & RCC_CFGR_PLLMULL;
00252       pllsource = RCC->CFGR & RCC_CFGR_PLLSRC;
00253
00254 #if !defined(STM32F105xC) && !defined(STM32F107xC)
```

```
00255       pllmull = ( pllmull » 18U) + 2U;
00256
00257       if (pllsource == 0x00U)
00258       {
00259         /* HSI oscillator clock divided by 2 selected as PLL clock entry */
00260         SystemCoreClock = (HSI_VALUE » 1U) * pllmull;
00261       }
00262       else
00263       {
00264  #if defined(STM32F100xB) || defined(STM32F100xE)
00265        prediv1factor = (RCC->CFGR2 & RCC_CFGR2_PREDIV1) + 1U;
00266        /* HSE oscillator clock selected as PREDIV1 clock entry */
00267        SystemCoreClock = (HSE_VALUE / prediv1factor) * pllmull;
00268  #else
00269         /* HSE selected as PLL clock entry */
00270         if ((RCC->CFGR & RCC_CFGR_PLLXTPRE) != (uint32_t)RESET)
00271         {/* HSE oscillator clock divided by 2 */
00272           SystemCoreClock = (HSE_VALUE » 1U) * pllmull;
00273         }
00274         else
00275         {
00276           SystemCoreClock = HSE_VALUE * pllmull;
00277         }
00278  #endif
00279       }
00280 #else
00281       pllmull = pllmull » 18U;
00282
00283       if (pllmull != 0x0DU)
00284       {
00285          pllmull += 2U;
00286       }
00287       else
00288       { /* PLL multiplication factor = PLL input clock * 6.5 */
00289         pllmull = 13U / 2U;
00290       }
00291
00292       if (pllsource == 0x00U)
00293       {
00294         /* HSI oscillator clock divided by 2 selected as PLL clock entry */
00295         SystemCoreClock = (HSI_VALUE » 1U) * pllmull;
00296       }
00297       else
00298       {/* PREDIV1 selected as PLL clock entry */
00299
00300         /* Get PREDIV1 clock source and division factor */
00301         prediv1source = RCC->CFGR2 & RCC_CFGR2_PREDIV1SRC;
00302         prediv1factor = (RCC->CFGR2 & RCC_CFGR2_PREDIV1) + 1U;
00303
00304         if (prediv1source == 0U)
00305         {
00306           /* HSE oscillator clock selected as PREDIV1 clock entry */
00307           SystemCoreClock = (HSE_VALUE / prediv1factor) * pllmull;
00308         }
00309         else
00310         {/* PLL2 clock selected as PREDIV1 clock entry */
00311
00312           /* Get PREDIV2 division factor and PLL2 multiplication factor */
00313           prediv2factor = ((RCC->CFGR2 & RCC_CFGR2_PREDIV2) » 4U) + 1U;
00314           pll2mull = ((RCC->CFGR2 & RCC_CFGR2_PLL2MUL) » 8U) + 2U;
00315           SystemCoreClock = (((HSE_VALUE / prediv2factor) * pll2mull) / prediv1factor) * pllmull;

00316         }
00317       }
00318 #endif /* STM32F105xC */
00319       break;
00320
00321     default:
00322       SystemCoreClock = HSI_VALUE;
00323       break;
00324   }
00325
00326   /* Compute HCLK clock frequency ----------------*/
00327   /* Get HCLK prescaler */
00328   tmp = AHBPrescTable[((RCC->CFGR & RCC_CFGR_HPRE) » 4U)];
00329   /* HCLK clock frequency */
00330   SystemCoreClock »= tmp;
00331 }
```

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.

```
00177 {
00178 #if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) || defined(STM32F103xE) ||
       defined(STM32F103xG)
00179   #ifdef DATA_IN_ExtSRAM
00180     SystemInit_ExtMemCtl();
00181   #endif /* DATA_IN_ExtSRAM */
00182 #endif
00183
00184   /* Configure the Vector Table location ----------------------------------*/
00185 #if defined(USER_VECT_TAB_ADDRESS)
00186   SCB->VTOR = VECT_TAB_BASE_ADDRESS | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM. */
00187 #endif /* USER_VECT_TAB_ADDRESS */
00188 }
```

Capítulo 4

Arquivos

4.1 FIRFilter.c

```
00001 #include "FIRFilter.h"
00002
00003 /* Designed filter coefficients*/
00004 static float FIR_IMPULSE_RESPONSE[FIR_FILTER_LENGTH] = {-0.01238356f, 0.10332170f, 0.81812371f,
       0.10332170f, -0.01238356f};
00005
00006 void FIRFilter_Init(FIRFilter *fir){
00007
00008     /* Clear filter buffer */
00009     for(uint8_t n=0; n<FIR_FILTER_LENGTH;n++){
00010
00011         fir->buf[n] = 0.0f;
00012
00013     }
00014
00015     /* Reset buffer index */
00016     fir->bufindex = 0;
00017
00018     /* Clear filter output */
00019     fir->out = 0.0f;
00020 }
00021
00022 float FIRFilter_Update(FIRFilter *fir, float inp){
00023
00024     /* Store latest sample in buffer */
00025     fir->buf[fir->bufindex] = inp;
00026
00027     /* Increment buffer index and wrap around if necessary */
00028     fir->bufindex++;
00029
00030     if(fir->bufindex == FIR_FILTER_LENGTH){
00031
00032         fir->bufindex = 0;
00033
00034     }
00035
00036     /* Compute new output sample (via convolution) */
00037     fir->out = 0.0f;
00038
00039     uint8_t sumIndex = fir->bufindex;
00040
00041     for(uint8_t n=0; n<FIR_FILTER_LENGTH;n++){
00042
00043         /* Decrement index and wrap if necessary */
00044         if(sumIndex>0){
00045
00046             sumIndex--;
00047
00048         }else{
00049
00050             sumIndex = FIR_FILTER_LENGTH -1;
00051
00052         }
00053
00054         /* Multiply impulse response with shifted input sample and add to output */
00055         fir->out += FIR_IMPULSE_RESPONSE[n] * fir->buf[sumIndex];
00056     }
00057
```

```
00058     /* Return filtered output */
00059     return fir->out;
00060
00061 }
```

4.2 freertos.c

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Includes ------------------------------------------------------------------*/
00022 #include "FreeRTOS.h"
00023 #include "task.h"
00024 #include "main.h"
00025
00026 /* Private includes ----------------------------------------------------------*/
00027 /* USER CODE BEGIN Includes */
00028
00029 /* USER CODE END Includes */
00030
00031 /* Private typedef -----------------------------------------------------------*/
00032 /* USER CODE BEGIN PTD */
00033
00034 /* USER CODE END PTD */
00035
00036 /* Private define ------------------------------------------------------------*/
00037 /* USER CODE BEGIN PD */
00038
00039 /* USER CODE END PD */
00040
00041 /* Private macro -------------------------------------------------------------*/
00042 /* USER CODE BEGIN PM */
00043
00044 /* USER CODE END PM */
00045
00046 /* Private variables ---------------------------------------------------------*/
00047 /* USER CODE BEGIN Variables */
00048
00049 /* USER CODE END Variables */
00050
00051 /* Private function prototypes -----------------------------------------------*/
00052 /* USER CODE BEGIN FunctionPrototypes */
00053
00054 /* USER CODE END FunctionPrototypes */
00055
00056 /* GetIdleTaskMemory prototype (linked to static allocation support) */
00057 void vApplicationGetIdleTaskMemory( StaticTask_t **ppxIdleTaskTCBBuffer, StackType_t
      **ppxIdleTaskStackBuffer, uint32_t *pulIdleTaskStackSize );
00058
00059 /* USER CODE BEGIN GET_IDLE_TASK_MEMORY */
00060 static StaticTask_t xIdleTaskTCBBuffer;
00061 static StackType_t xIdleStack[configMINIMAL_STACK_SIZE];
00062
00063 void vApplicationGetIdleTaskMemory( StaticTask_t **ppxIdleTaskTCBBuffer, StackType_t
      **ppxIdleTaskStackBuffer, uint32_t *pulIdleTaskStackSize )
00064 {
00065   *ppxIdleTaskTCBBuffer = &xIdleTaskTCBBuffer;
00066   *ppxIdleTaskStackBuffer = &xIdleStack[0];
00067   *pulIdleTaskStackSize = configMINIMAL_STACK_SIZE;
00068   /* place for user code */
00069 }
00070 /* USER CODE END GET_IDLE_TASK_MEMORY */
00071
00072 /* Private application code --------------------------------------------------*/
00073 /* USER CODE BEGIN Application */
00074
00075 /* USER CODE END Application */
00076
00077 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

4.3 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 r1 0.00021012f
- #define p1 1.00000000f
- #define k 0.06382217f
- #define ksat 0.10000000f
- #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);
- #define DWT_CTRL (∗(volatile uint32_t∗) 0XE0001000)

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

4.3.1 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.

4.3.2 Definições e macros

4.3.2.1 CSdis

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

Definição na linha 36 do arquivo main.c.

4.3.2.2 CSen

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

Definição na linha 35 do arquivo main.c.

4.3.2.3 DWT_CTRL

```
#define DWT_CTRL (*(volatile uint32_t*) 0XE0001000)
```

Definição na linha 41 do arquivo main.c.

4.3.2.4 k

```
#define k 0.06382217f
```

Definição na linha 32 do arquivo main.c.

4.3.2.5 ksat

```
#define ksat 0.10000000f
```

Definição na linha 33 do arquivo main.c.

4.3.2.6 p1

```
#define p1 1.00000000f
```

Definição na linha 31 do arquivo main.c.

4.3.2.7 r1

```
#define r1 0.00021012f
```

Definição na linha 30 do arquivo main.c.

4.3.2.8 SCK_H

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

Definição na linha 38 do arquivo main.c.

4.3.2.9 SCK_L

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

Definição na linha 39 do arquivo main.c.

4.3.3 Funções

4.3.3.1 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 393 do arquivo main.c.

```
00393                                           {
00394     while (1) {
00395         float rx_filteredTemp;
00396         /* Recebe da fila filteredTempQueue */
00397         if (xQueueReceive(filteredTempQueue, &rx_filteredTemp, 10)) {
00398             /* Leitura da entrada analógica para calculo de referencia */
00399             HAL_ADC_PollForConversion(&hadc1, 10);
00400             ref = (float) HAL_ADC_GetValue(&hadc1) / 27.3; // leitura do potenciometro convertido em
      ref até 150°C
00401
00402             /* Lei de controle */
00403             float u;
00404             static float up, uint;
00405             int flag_sat;
00406             float ek = ref - rx_filteredTemp;
00407
00408             /* Controlador bang-bang ventoinha */
00409             if (ek < -15.0) {
00410                 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_SET);
00411             } else {
00412                 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET);
00413             }
00414
00415             /* Anti-windup integrador */
00416             if (!flag_sat) {
00417                 uint = uint * p1 + r1 * ek;
00418
00419             } else if (flag_sat) {
00420                 uint = (uint * p1 + r1 * ek) * ksat;
00421             }
00422
00423             /* Proporcional */
00424             up = k * ek;
00425
00426
00427             /* Ação de controle */
00428             u = up + uint;
00429
00430             /* Conversão período PWM */
00431             u = u * 4500.0;
00432
00433             /* Limites de saturação de PWM */
00434             if (u > 18000.0) {
00435                 u = 18000.0;
00436                 flag_sat = 1;
00437             } else if (u < 0.0) {
00438                 u = 0.0;
00439                 flag_sat = 1;
00440             } else {
00441                 u = u;
00442                 flag_sat = 0;
00443             }
00444
00445             /* Converte periodo do timer em razão cíclica */
00446             dutyCycle = u / 180.0;
00447
00448             /* Seta periférico PWM */
00449             __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, (uint32_t ) u);
00450
00451         }
00452
00453     }
00454 }
```

4.3.3.2 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 464 do arquivo main.c.

```
00464                                       {
00465     while (1) {
00466         char str[100];
00467         /* Fim de comando definido pela API do display */
00468         uint8_t Cmd_End[3] = { 0xFF, 0xFF, 0xFF };
00469
00470         /* Atualiza valor do setpoint */
00471         int32_t number = ref * 100;
00472         sprintf(str, "setPoint.val=%ld", number);
00473         HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
00474         HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);
00475
00476         /* Atualiza valor da variável de processo */
00477         number = filteredTemp * 100;
00478         sprintf(str, "filteredTemp.val=%ld", number);
00479         HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
00480         HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);
00481
00482         /* Atualiza valor da variável manipulada */
00483         number = dutyCycle * 100;
00484         sprintf(str, "dutyCycle.val=%ld", number);
00485         HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
00486         HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);
00487
00488         /* Atraso para definição do período da tarefa */
00489         vTaskDelay(1000); /*1Hz frequency*/
00490     }
00491 }
```

4.3.3.3 Error_Handler()

```
void Error_Handler (
             void  )
```

Função executada em caso de erro na aplicação.

Valores Retornados

None	

Definição na linha 509 do arquivo main.c.

```
00509                           {
00510     __disable_irq();
00511     while (1) {
00512     }
00513 }
```

4.3.3.4 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 filteredTemp↩ Queue

Parâmetros

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

Valores Retornados

None	

Definição na linha 364 do arquivo main.c.

```
00364                                       {
00365     uint8_t aux = 0;
00366     while (1) {
00367         float rx_temp;
00368         /* Recebe da fila filteredTempQueue */
00369         if (xQueueReceive(tempQueue, &rx_temp, 10)) {
00370             aux++;
00371             /* Chamada do filtro FIR */
00372             filteredTemp = FIRFilter_Update(&tempFilter, rx_temp);
00373 
00374         }
00375         if (aux == 5) {
00376 
00377             aux = 0;
00378             /* Adiciona a fila filteredTempQueue */
00379             if (xQueueSend(filteredTempQueue, &filteredTemp, 10) == pdPASS) {
00380             }
00381         }
00382     }
00383 }
```

4.3.3.5 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 499 do arquivo main.c.

```
00499                                                   {
00500     if (htim->Instance == TIM4) {
00501         HAL_IncTick();
00502     }
00503 }
```

4.3.3.6 main()

```
int main (
              void  )
```

ponto de entrada da aplicação.

Valores Retornados

int	

Definição na linha 78 do arquivo main.c.

```
00078               {
00079     /* Reinicia todos os periféricos, inicializa a interface flash e o systick */
00080     HAL_Init();
00081
00082     /* Configura o clock do sistema */
00083     SystemClock_Config();
00084
00085     /* Inicializa todos os periféricos configurados */
00086     MX_GPIO_Init();
00087     MX_ADC1_Init();
00088     MX_TIM1_Init();
00089     MX_USART1_UART_Init();
00090
00091     DWT_CTRL |= (1<<0);
00092
00093     /* Inicializa o conversor AD */
00094     if (HAL_ADC_Start(&hadc1) != HAL_OK) {
00095         Error_Handler();
00096     }
00097
00098     /* Inicializa o timer1 em modo PWM */
00099     if (HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1) != HAL_OK) {
00100         Error_Handler();
00101     }
00102
00103     /* Inicializa o filtro FIR */
00104     FIRFilter_Init(&tempFilter);
00105
00106     SEGGER_SYSVIEW_Conf();
00107     SEGGER_SYSVIEW_Start();
00108
00109     /* Cria a fila de leituras de temperatura bruta */
```

```
00110     tempQueue = xQueueCreate(1, sizeof(float));
00111     if (tempQueue == 0) {
00112         Error_Handler();
00113     }
00114
00115     /* Cria a fila de leituras de temperatura filtrada */
00116     filteredTempQueue = xQueueCreate(1, sizeof(float));
00117     if (tempQueue == 0) {
00118         Error_Handler();
00119     }
00120
00121     /* Cria tasks na pilha do sistema */
00122     xTaskCreate(Temp_taskF, "TempTask", 128, NULL, 3, &Temp_Task);
00123     xTaskCreate(Filter_taskF, "FilterTask", 128, NULL, 2, &Filter_Task);
00124     xTaskCreate(Control_taskF, "ControlTask", 128, NULL, 4, &Control_Task);
00125     xTaskCreate(Display_taskF, "DisplayTask", 128, NULL, 1, &Display_Task);
00126
00127     /* Inicializa o escalonador */
00128     vTaskStartScheduler();
00129
00130     /* Loop infinito */
00131     while (1) {
00132     }
00133 }
```

4.3.3.7 SystemClock_Config()

```
void SystemClock_Config (
             void  )
```

Configuração do clock do sistema.

Valores Retornados

None	

Definição na linha 139 do arquivo main.c.

```
00139                                 {
00140     RCC_OscInitTypeDef RCC_OscInitStruct = { 0 };
00141     RCC_ClkInitTypeDef RCC_ClkInitStruct = { 0 };
00142     RCC_PeriphCLKInitTypeDef PeriphClkInit = { 0 };
00143
00144     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
00145     RCC_OscInitStruct.HSEState = RCC_HSE_ON;
00146     RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
00147     RCC_OscInitStruct.HSIState = RCC_HSI_ON;
00148     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
00149     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
00150     RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
00151     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
00152         Error_Handler();
00153     }
00154
00155     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK
00156             | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
00157     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
00158     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
00159     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
00160     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
00161
00162     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK) {
00163         Error_Handler();
00164     }
00165     PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
00166     PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV6;
00167     if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK) {
00168         Error_Handler();
00169     }
00170 }
```

4.3.3.8 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 322 do arquivo main.c.

```
00322                                   {
00323     while (1) {
00324         uint8_t tempdata[16];
00325         uint16_t temp16 = 0;
00326 
00327         /* bitbanging protocolo SPI */
00328         CSen
00329         for (int i = 0; i < 16; i++) {
00330             SCK_H
00331             tempdata[i] = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_4);
00332             SCK_L
00333         }
00334         CSdis
00335 
00336         /* Conversão temperatura */
00337         if (tempdata[13] == 0) {
00338 
00339             for (int n = 1; n < 13; n++) {
00340                 temp16 += tempdata[n] * (2048 / (1 « (n - 1)));
00341             }
00342 
00343         }
00344 
00345         float temp = (float) temp16 / 4;
00346 
00347         /* Adiciona a fila tempQueue */
00348         if (xQueueSend(tempQueue, &temp, 10) == pdPASS) {
00349         }
00350 
00351         /* Atraso para definição do período da tarefa */
00352         vTaskDelay(200); /*5Hz frequency*/
00353     }
00354 }
```

4.3.4 Variáveis

4.3.4.1 Control_Task

```
TaskHandle_t Control_Task
```

Definição na linha 51 do arquivo main.c.

4.3.4.2 Display_Task

```
TaskHandle_t Display_Task
```

Definição na linha 52 do arquivo main.c.

4.3.4.3 dutyCycle

```
float dutyCycle = 0
```

Definição na linha 61 do arquivo main.c.

4.3.4.4 Filter_Task

```
TaskHandle_t Filter_Task
```

Definição na linha 50 do arquivo main.c.

4.3.4.5 filteredTemp

```
float filteredTemp = 0
```

Definição na linha 59 do arquivo main.c.

4.3.4.6 filteredTempQueue

```
QueueHandle_t filteredTempQueue
```

Definição na linha 55 do arquivo main.c.

4.3.4.7 hadc1

```
ADC_HandleTypeDef hadc1
```

Definição na linha 43 do arquivo main.c.

4.3.4.8 htim1

```
TIM_HandleTypeDef htim1
```

Definição na linha 45 do arquivo main.c.

4.3.4.9 huart1

```
UART_HandleTypeDef huart1
```

Definição na linha 47 do arquivo main.c.

4.3.4.10 ref

```
float ref = 0
```

Definição na linha 60 do arquivo main.c.

4.3.4.11 Temp_Task

```
TaskHandle_t Temp_Task
```

Definição na linha 49 do arquivo main.c.

4.3.4.12 tempFilter

```
FIRFilter tempFilter
```

Definição na linha 57 do arquivo main.c.

4.3.4.13 tempQueue

```
QueueHandle_t tempQueue
```

Definição na linha 54 do arquivo main.c.

4.4 main.c

Vá para a documentação desse arquivo.

```
00001
00018 #include "main.h"
00019
00020 #include "FreeRTOS.h"
00021 #include "task.h"
00022 #include "queue.h"
00023
00024 #include "string.h"
00025 #include "stdio.h"
00026 #include "stdlib.h"
00027
00028 #include "FIRFilter.h"
00029
00030 #define r1 0.00021012f
00031 #define p1 1.00000000f
00032 #define k 0.06382217f
00033 #define ksat 0.10000000f
00034
00035 #define CSen HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_RESET);
00036 #define CSdis HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
00037
00038 #define SCK_H HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
00039 #define SCK_L HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);
00040
00041 #define DWT_CTRL (*(volatile uint32_t*) 0XE0001000)
00042
00043 ADC_HandleTypeDef hadc1;
00044
00045 TIM_HandleTypeDef htim1;
00046
00047 UART_HandleTypeDef huart1;
00048
00049 TaskHandle_t Temp_Task;
00050 TaskHandle_t Filter_Task;
00051 TaskHandle_t Control_Task;
00052 TaskHandle_t Display_Task;
00053
00054 QueueHandle_t tempQueue;
00055 QueueHandle_t filteredTempQueue;
00056
00057 FIRFilter tempFilter;
00058
00059 float filteredTemp = 0;
00060 float ref = 0;
00061 float dutyCycle = 0;
00062
00063 void SystemClock_Config(void);
00064 static void MX_GPIO_Init(void);
00065 static void MX_ADC1_Init(void);
00066 static void MX_TIM1_Init(void);
00067 static void MX_USART1_UART_Init(void);
00068
00069 void Temp_taskF(void *pvParameters);
00070 void Filter_taskF(void *pvParameters);
00071 void Control_taskF(void *pvParameters);
00072 void Display_taskF(void *pvParameters);
00073
00078 int main(void) {
00079     /* Reinicia todos os periféricos, inicializa a interface flash e o systick */
00080     HAL_Init();
00081
00082     /* Configura o clock do sistema */
00083     SystemClock_Config();
00084
00085     /* Inicializa todos os periféricos configurados */
00086     MX_GPIO_Init();
00087     MX_ADC1_Init();
00088     MX_TIM1_Init();
00089     MX_USART1_UART_Init();
00090
00091     DWT_CTRL |= (1<<0);
00092
00093     /* Inicializa o conversor AD */
00094     if (HAL_ADC_Start(&hadc1) != HAL_OK) {
00095         Error_Handler();
00096     }
00097
00098     /* Inicializa o timer1 em modo PWM */
00099     if (HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1) != HAL_OK) {
00100         Error_Handler();
00101     }
00102
```

```
00103     /* Inicializa o filtro FIR */
00104     FIRFilter_Init(&tempFilter);
00105
00106     SEGGER_SYSVIEW_Conf();
00107     SEGGER_SYSVIEW_Start();
00108
00109     /* Cria a fila de leituras de temperatura bruta */
00110     tempQueue = xQueueCreate(1, sizeof(float));
00111     if (tempQueue == 0) {
00112         Error_Handler();
00113     }
00114
00115     /* Cria a fila de leituras de temperatura filtrada */
00116     filteredTempQueue = xQueueCreate(1, sizeof(float));
00117     if (tempQueue == 0) {
00118         Error_Handler();
00119     }
00120
00121     /* Cria tasks na pilha do sistema */
00122     xTaskCreate(Temp_taskF, "TempTask", 128, NULL, 3, &Temp_Task);
00123     xTaskCreate(Filter_taskF, "FilterTask", 128, NULL, 2, &Filter_Task);
00124     xTaskCreate(Control_taskF, "ControlTask", 128, NULL, 4, &Control_Task);
00125     xTaskCreate(Display_taskF, "DisplayTask", 128, NULL, 1, &Display_Task);
00126
00127     /* Inicializa o escalonador */
00128     vTaskStartScheduler();
00129
00130     /* Loop infinito */
00131     while (1) {
00132     }
00133 }
00134
00139 void SystemClock_Config(void) {
00140     RCC_OscInitTypeDef RCC_OscInitStruct = { 0 };
00141     RCC_ClkInitTypeDef RCC_ClkInitStruct = { 0 };
00142     RCC_PeriphCLKInitTypeDef PeriphClkInit = { 0 };
00143
00144     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
00145     RCC_OscInitStruct.HSEState = RCC_HSE_ON;
00146     RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
00147     RCC_OscInitStruct.HSIState = RCC_HSI_ON;
00148     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
00149     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
00150     RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
00151     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
00152         Error_Handler();
00153     }
00154
00155     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK
00156             | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
00157     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
00158     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
00159     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
00160     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
00161
00162     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK) {
00163         Error_Handler();
00164     }
00165     PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
00166     PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV6;
00167     if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK) {
00168         Error_Handler();
00169     }
00170 }
00171
00177 static void MX_ADC1_Init(void) {
00178
00179     ADC_ChannelConfTypeDef sConfig = { 0 };
00180
00181     hadc1.Instance = ADC1;
00182     hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
00183     hadc1.Init.ContinuousConvMode = ENABLE;
00184     hadc1.Init.DiscontinuousConvMode = DISABLE;
00185     hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
00186     hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
00187     hadc1.Init.NbrOfConversion = 1;
00188     if (HAL_ADC_Init(&hadc1) != HAL_OK) {
00189         Error_Handler();
00190     }
00191
00192     sConfig.Channel = ADC_CHANNEL_0;
00193     sConfig.Rank = ADC_REGULAR_RANK_1;
00194     sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES_5;
00195     if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) {
00196         Error_Handler();
00197     }
00198 }
```

```
00199
00205 static void MX_TIM1_Init(void) {
00206
00207     TIM_MasterConfigTypeDef sMasterConfig = { 0 };
00208     TIM_OC_InitTypeDef sConfigOC = { 0 };
00209     TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = { 0 };
00210
00211     htim1.Instance = TIM1;
00212     htim1.Init.Prescaler = 1;
00213     htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
00214     htim1.Init.Period = 18000 - 1;
00215     htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
00216     htim1.Init.RepetitionCounter = 0;
00217     htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
00218     if (HAL_TIM_PWM_Init(&htim1) != HAL_OK) {
00219         Error_Handler();
00220     }
00221     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
00222     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
00223     if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig)
00224             != HAL_OK) {
00225         Error_Handler();
00226     }
00227     sConfigOC.OCMode = TIM_OCMODE_PWM1;
00228     sConfigOC.Pulse = 0;
00229     sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
00230     sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
00231     sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
00232     sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
00233     sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
00234     if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1)
00235             != HAL_OK) {
00236         Error_Handler();
00237     }
00238     sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
00239     sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
00240     sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
00241     sBreakDeadTimeConfig.DeadTime = 0;
00242     sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
00243     sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
00244     sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
00245     if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig)
00246             != HAL_OK) {
00247         Error_Handler();
00248     }
00249
00250     HAL_TIM_MspPostInit(&htim1);
00251 }
00252
00258 static void MX_USART1_UART_Init(void) {
00259
00260     huart1.Instance = USART1;
00261     huart1.Init.BaudRate = 115200;
00262     huart1.Init.WordLength = UART_WORDLENGTH_8B;
00263     huart1.Init.StopBits = UART_STOPBITS_1;
00264     huart1.Init.Parity = UART_PARITY_NONE;
00265     huart1.Init.Mode = UART_MODE_TX;
00266     huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
00267     huart1.Init.OverSampling = UART_OVERSAMPLING_16;
00268     if (HAL_UART_Init(&huart1) != HAL_OK) {
00269         Error_Handler();
00270     }
00271 }
00272
00278 static void MX_GPIO_Init(void) {
00279     GPIO_InitTypeDef GPIO_InitStruct = { 0 };
00280
00281     __HAL_RCC_GPIOC_CLK_ENABLE();
00282     __HAL_RCC_GPIOD_CLK_ENABLE();
00283     __HAL_RCC_GPIOA_CLK_ENABLE();
00284     __HAL_RCC_GPIOB_CLK_ENABLE();
00285
00286     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_RESET);
00287
00288     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
00289
00290     GPIO_InitStruct.Pin = GPIO_PIN_15;
00291     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00292     GPIO_InitStruct.Pull = GPIO_NOPULL;
00293     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00294     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00295
00296     GPIO_InitStruct.Pin = GPIO_PIN_3;
00297     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00298     GPIO_InitStruct.Pull = GPIO_NOPULL;
00299     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00300     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

```
00301
00302     GPIO_InitStruct.Pin = GPIO_PIN_4;
00303     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
00304     GPIO_InitStruct.Pull = GPIO_NOPULL;
00305     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00306
00307     GPIO_InitStruct.Pin = GPIO_PIN_15;
00308     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00309     GPIO_InitStruct.Pull = GPIO_NOPULL;
00310     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00311     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00312 }
00313
00322 void Temp_taskF(void *pvParameters) {
00323     while (1) {
00324         uint8_t tempdata[16];
00325         uint16_t temp16 = 0;
00326
00327         /* bitbanging protocolo SPI */
00328         CSen
00329         for (int i = 0; i < 16; i++) {
00330             SCK_H
00331             tempdata[i] = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_4);
00332             SCK_L
00333         }
00334         CSdis
00335
00336         /* Conversão temperatura */
00337         if (tempdata[13] == 0) {
00338
00339             for (int n = 1; n < 13; n++) {
00340                 temp16 += tempdata[n] * (2048 / (1 « (n - 1)));
00341             }
00342
00343         }
00344
00345         float temp = (float) temp16 / 4;
00346
00347         /* Adiciona a fila tempQueue */
00348         if (xQueueSend(tempQueue, &temp, 10) == pdPASS) {
00349         }
00350
00351         /* Atraso para definição do período da tarefa */
00352         vTaskDelay(200); /*5Hz frequency*/
00353     }
00354 }
00355
00364 void Filter_taskF(void *pvParameters) {
00365     uint8_t aux = 0;
00366     while (1) {
00367         float rx_temp;
00368         /* Recebe da fila filteredTempQueue */
00369         if (xQueueReceive(tempQueue, &rx_temp, 10)) {
00370             aux++;
00371             /* Chamada do filtro FIR */
00372             filteredTemp = FIRFilter_Update(&tempFilter, rx_temp);
00373
00374         }
00375         if (aux == 5) {
00376
00377             aux = 0;
00378             /* Adiciona a fila filteredTempQueue */
00379             if (xQueueSend(filteredTempQueue, &filteredTemp, 10) == pdPASS) {
00380             }
00381         }
00382     }
00383 }
00384
00393 void Control_taskF(void *pvParameters) {
00394     while (1) {
00395         float rx_filteredTemp;
00396         /* Recebe da fila filteredTempQueue */
00397         if (xQueueReceive(filteredTempQueue, &rx_filteredTemp, 10)) {
00398             /* Leitura da entrada analógica para calculo de referencia */
00399             HAL_ADC_PollForConversion(&hadc1, 10);
00400             ref = (float) HAL_ADC_GetValue(&hadc1) / 27.3; // leitura do potenciometro convertido em
      ref até 150°C
00401
00402             /* Lei de controle */
00403             float u;
00404             static float up, uint;
00405             int flag_sat;
00406             float ek = ref - rx_filteredTemp;
00407
00408             /* Controlador bang-bang ventoinha */
00409             if (ek < -15.0) {
00410                 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_SET);
```

```
00411             } else {
00412                 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET);
00413             }
00414 
00415             /* Anti-windup integrador */
00416             if (!flag_sat) {
00417                 uint = uint * p1 + r1 * ek;
00418 
00419             } else if (flag_sat) {
00420                 uint = (uint * p1 + r1 * ek) * ksat;
00421             }
00422 
00423             /* Proporcional */
00424             up = k * ek;
00425 
00426 
00427             /* Ação de controle */
00428             u = up + uint;
00429 
00430             /* Conversão período PWM */
00431             u = u * 4500.0;
00432 
00433             /* Limites de saturação de PWM */
00434             if (u > 18000.0) {
00435                 u = 18000.0;
00436                 flag_sat = 1;
00437             } else if (u < 0.0) {
00438                 u = 0.0;
00439                 flag_sat = 1;
00440             } else {
00441                 u = u;
00442                 flag_sat = 0;
00443             }
00444 
00445             /* Converte periodo do timer em razão cíclica */
00446             dutyCycle = u / 180.0;
00447 
00448             /* Seta periférico PWM */
00449             __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, (uint32_t ) u);
00450 
00451         }
00452 
00453     }
00454 }
00455 
00464 void Display_taskF(void *pvParameters) {
00465     while (1) {
00466         char str[100];
00467         /* Fim de comando definido pela API do display */
00468         uint8_t Cmd_End[3] = { 0xFF, 0xFF, 0xFF };
00469 
00470         /* Atualiza valor do setpoint */
00471         int32_t number = ref * 100;
00472         sprintf(str, "setPoint.val=%ld", number);
00473         HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
00474         HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);
00475 
00476         /* Atualiza valor da variável de processo */
00477         number = filteredTemp * 100;
00478         sprintf(str, "filteredTemp.val=%ld", number);
00479         HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
00480         HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);
00481 
00482         /* Atualiza valor da variável manipulada */
00483         number = dutyCycle * 100;
00484         sprintf(str, "dutyCycle.val=%ld", number);
00485         HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
00486         HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);
00487 
00488         /* Atraso para definição do período da tarefa */
00489         vTaskDelay(1000); /*1Hz frequency*/
00490     }
00491 }
00492 
00499 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
00500     if (htim->Instance == TIM4) {
00501         HAL_IncTick();
00502     }
00503 }
00504 
00509 void Error_Handler(void) {
00510     __disable_irq();
00511     while (1) {
00512     }
00513 }
00514 
00515 #ifdef  USE_FULL_ASSERT
```

```
00522 void assert_failed(uint8_t *file, uint32_t line)
00523 {
00524 }
00525 #endif
```

4.5 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"
#include "FreeRTOS.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.

4.5.1 Descrição detalhada

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

Atenção

Definição no arquivo stm32f1xx_hal_msp.c.

4.5.2 Funções

4.5.2.1 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 124 do arquivo stm32f1xx_hal_msp.c.

```
00125 {
00126   if(hadc->Instance==ADC1)
00127   {
00128   /* USER CODE BEGIN ADC1_MspDeInit 0 */
00129
00130   /* USER CODE END ADC1_MspDeInit 0 */
00131     /* Peripheral clock disable */
00132     __HAL_RCC_ADC1_CLK_DISABLE();
00133
00137     HAL_GPIO_DeInit(GPIOA, GPIO_PIN_0);
00138
00139   /* USER CODE BEGIN ADC1_MspDeInit 1 */
00140
00141   /* USER CODE END ADC1_MspDeInit 1 */
00142   }
00143
00144 }
```

4.5.2.2 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 92 do arquivo stm32f1xx_hal_msp.c.

```
00093 {
00094   GPIO_InitTypeDef GPIO_InitStruct = {0};
00095   if(hadc->Instance==ADC1)
00096   {
00097   /* USER CODE BEGIN ADC1_MspInit 0 */
00098
00099   /* USER CODE END ADC1_MspInit 0 */
00100     /* Peripheral clock enable */
00101     __HAL_RCC_ADC1_CLK_ENABLE();
00102
00103     __HAL_RCC_GPIOA_CLK_ENABLE();
00107     GPIO_InitStruct.Pin = GPIO_PIN_0;
00108     GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
00109     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00110
00111   /* USER CODE BEGIN ADC1_MspInit 1 */
00112
00113   /* USER CODE END ADC1_MspInit 1 */
00114   }
00115
00116 }
```

4.5.2.3 HAL_MspInit()

```
void HAL_MspInit (
             void  )
```

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

Definição na linha 64 do arquivo stm32f1xx_hal_msp.c.

```
00065 {
00066   /* USER CODE BEGIN MspInit 0 */
00067
00068   /* USER CODE END MspInit 0 */
00069
00070   __HAL_RCC_AFIO_CLK_ENABLE();
00071   __HAL_RCC_PWR_CLK_ENABLE();
00072
00073   /* System interrupt init*/
00074   /* PendSV_IRQn interrupt configuration */
00075   HAL_NVIC_SetPriority(PendSV_IRQn, 15, 0);
00076
00079   __HAL_AFIO_REMAP_SWJ_NOJTAG();
00080
00081   /* USER CODE BEGIN MspInit 1 */
00082
00083   /* USER CODE END MspInit 1 */
00084 }
```

4.5.2.4 HAL_TIM_MspPostInit()

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

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

Definição na linha 168 do arquivo stm32f1xx_hal_msp.c.

```
00169 {
00170   GPIO_InitTypeDef GPIO_InitStruct = {0};
00171   if(htim->Instance==TIM1)
00172   {
00173   /* USER CODE BEGIN TIM1_MspPostInit 0 */
00174
```

```
00175   /* USER CODE END TIM1_MspPostInit 0 */
00176
00177     __HAL_RCC_GPIOA_CLK_ENABLE();
00181     GPIO_InitStruct.Pin = GPIO_PIN_8;
00182     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
00183     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00184     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00185
00186   /* USER CODE BEGIN TIM1_MspPostInit 1 */
00187
00188   /* USER CODE END TIM1_MspPostInit 1 */
00189   }
00190
00191 }
```

4.5.2.5 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 198 do arquivo stm32f1xx_hal_msp.c.

```
00199 {
00200   if(htim_oc->Instance==TIM1)
00201   {
00202   /* USER CODE BEGIN TIM1_MspDeInit 0 */
00203
00204   /* USER CODE END TIM1_MspDeInit 0 */
00205     /* Peripheral clock disable */
00206     __HAL_RCC_TIM1_CLK_DISABLE();
00207   /* USER CODE BEGIN TIM1_MspDeInit 1 */
00208
00209   /* USER CODE END TIM1_MspDeInit 1 */
00210   }
00211
00212 }
```

4.5.2.6 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 152 do arquivo stm32f1xx_hal_msp.c.

```
00153 {
00154   if(htim_oc->Instance==TIM1)
00155   {
00156   /* USER CODE BEGIN TIM1_MspInit 0 */
00157
00158   /* USER CODE END TIM1_MspInit 0 */
00159     /* Peripheral clock enable */
00160     __HAL_RCC_TIM1_CLK_ENABLE();
00161   /* USER CODE BEGIN TIM1_MspInit 1 */
00162     vInitPrioGroupValue();
00163   /* USER CODE END TIM1_MspInit 1 */
00164   }
00165
00166 }
```

4.5.2.7 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 259 do arquivo stm32f1xx_hal_msp.c.

```
00260 {
00261   if(huart->Instance==USART1)
00262   {
00263   /* USER CODE BEGIN USART1_MspDeInit 0 */
00264
00265   /* USER CODE END USART1_MspDeInit 0 */
00266     /* Peripheral clock disable */
00267     __HAL_RCC_USART1_CLK_DISABLE();
00268
00273     HAL_GPIO_DeInit(GPIOA, GPIO_PIN_9|GPIO_PIN_10);
00274
00275   /* USER CODE BEGIN USART1_MspDeInit 1 */
00276
00277   /* USER CODE END USART1_MspDeInit 1 */
00278   }
00279
00280 }
```

4.5.2.8 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 220 do arquivo stm32f1xx_hal_msp.c.

```
00221 {
00222   GPIO_InitTypeDef GPIO_InitStruct = {0};
00223   if(huart->Instance==USART1)
00224   {
00225   /* USER CODE BEGIN USART1_MspInit 0 */
00226
00227   /* USER CODE END USART1_MspInit 0 */
00228     /* Peripheral clock enable */
00229     __HAL_RCC_USART1_CLK_ENABLE();
00230
00231     __HAL_RCC_GPIOA_CLK_ENABLE();
00236     GPIO_InitStruct.Pin = GPIO_PIN_9;
00237     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
00238     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
00239     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00240
00241     GPIO_InitStruct.Pin = GPIO_PIN_10;
00242     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
00243     GPIO_InitStruct.Pull = GPIO_NOPULL;
00244     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00245
00246   /* USER CODE BEGIN USART1_MspInit 1 */
00247
00248   /* USER CODE END USART1_MspInit 1 */
00249   }
00250
00251 }
```

4.6 stm32f1xx_hal_msp.c

Vá para a documentação desse arquivo.

```
00001 /* USER CODE BEGIN Header */
00020 /* USER CODE END Header */
00021
00022 /* Includes ------------------------------------------------------------------*/
00023 #include "main.h"
00024 /* USER CODE BEGIN Includes */
00025 #include "FreeRTOS.h"
00026 /* USER CODE END Includes */
00027
00028 /* Private typedef -----------------------------------------------------------*/
00029 /* USER CODE BEGIN TD */
00030
00031 /* USER CODE END TD */
00032
00033 /* Private define ------------------------------------------------------------*/
00034 /* USER CODE BEGIN Define */
00035
00036 /* USER CODE END Define */
00037
00038 /* Private macro -------------------------------------------------------------*/
00039 /* USER CODE BEGIN Macro */
00040
00041 /* USER CODE END Macro */
00042
00043 /* Private variables ---------------------------------------------------------*/
00044 /* USER CODE BEGIN PV */
00045
00046 /* USER CODE END PV */
```

```
00047
00048 /* Private function prototypes -----------------------------------------------*/
00049 /* USER CODE BEGIN PFP */
00050
00051 /* USER CODE END PFP */
00052
00053 /* External functions --------------------------------------------------------*/
00054 /* USER CODE BEGIN ExternalFunctions */
00055
00056 /* USER CODE END ExternalFunctions */
00057
00058 /* USER CODE BEGIN 0 */
00059 void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);
00060 /* USER CODE END 0 */
00064 void HAL_MspInit(void)
00065 {
00066   /* USER CODE BEGIN MspInit 0 */
00067
00068   /* USER CODE END MspInit 0 */
00069
00070   __HAL_RCC_AFIO_CLK_ENABLE();
00071   __HAL_RCC_PWR_CLK_ENABLE();
00072
00073   /* System interrupt init*/
00074   /* PendSV_IRQn interrupt configuration */
00075   HAL_NVIC_SetPriority(PendSV_IRQn, 15, 0);
00076
00079   __HAL_AFIO_REMAP_SWJ_NOJTAG();
00080
00081   /* USER CODE BEGIN MspInit 1 */
00082
00083   /* USER CODE END MspInit 1 */
00084 }
00085
00092 void HAL_ADC_MspInit(ADC_HandleTypeDef* hadc)
00093 {
00094   GPIO_InitTypeDef GPIO_InitStruct = {0};
00095   if(hadc->Instance==ADC1)
00096   {
00097   /* USER CODE BEGIN ADC1_MspInit 0 */
00098
00099   /* USER CODE END ADC1_MspInit 0 */
00100     /* Peripheral clock enable */
00101     __HAL_RCC_ADC1_CLK_ENABLE();
00102
00103     __HAL_RCC_GPIOA_CLK_ENABLE();
00107     GPIO_InitStruct.Pin = GPIO_PIN_0;
00108     GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
00109     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00110
00111   /* USER CODE BEGIN ADC1_MspInit 1 */
00112
00113   /* USER CODE END ADC1_MspInit 1 */
00114   }
00115
00116 }
00117
00124 void HAL_ADC_MspDeInit(ADC_HandleTypeDef* hadc)
00125 {
00126   if(hadc->Instance==ADC1)
00127   {
00128   /* USER CODE BEGIN ADC1_MspDeInit 0 */
00129
00130   /* USER CODE END ADC1_MspDeInit 0 */
00131     /* Peripheral clock disable */
00132     __HAL_RCC_ADC1_CLK_DISABLE();
00133
00137     HAL_GPIO_DeInit(GPIOA, GPIO_PIN_0);
00138
00139   /* USER CODE BEGIN ADC1_MspDeInit 1 */
00140
00141   /* USER CODE END ADC1_MspDeInit 1 */
00142   }
00143
00144 }
00145
00152 void HAL_TIM_PWM_MspInit(TIM_HandleTypeDef* htim_oc)
00153 {
00154   if(htim_oc->Instance==TIM1)
00155   {
00156   /* USER CODE BEGIN TIM1_MspInit 0 */
00157
00158   /* USER CODE END TIM1_MspInit 0 */
00159     /* Peripheral clock enable */
00160     __HAL_RCC_TIM1_CLK_ENABLE();
00161   /* USER CODE BEGIN TIM1_MspInit 1 */
00162     vInitPrioGroupValue();
```

```
00163   /* USER CODE END TIM1_MspInit 1 */
00164   }
00165
00166 }
00167
00168 void HAL_TIM_MspPostInit(TIM_HandleTypeDef* htim)
00169 {
00170   GPIO_InitTypeDef GPIO_InitStruct = {0};
00171   if(htim->Instance==TIM1)
00172   {
00173   /* USER CODE BEGIN TIM1_MspPostInit 0 */
00174
00175   /* USER CODE END TIM1_MspPostInit 0 */
00176
00177     __HAL_RCC_GPIOA_CLK_ENABLE();
00181     GPIO_InitStruct.Pin = GPIO_PIN_8;
00182     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
00183     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00184     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00185
00186   /* USER CODE BEGIN TIM1_MspPostInit 1 */
00187
00188   /* USER CODE END TIM1_MspPostInit 1 */
00189   }
00190
00191 }
00198 void HAL_TIM_PWM_MspDeInit(TIM_HandleTypeDef* htim_oc)
00199 {
00200   if(htim_oc->Instance==TIM1)
00201   {
00202   /* USER CODE BEGIN TIM1_MspDeInit 0 */
00203
00204   /* USER CODE END TIM1_MspDeInit 0 */
00205     /* Peripheral clock disable */
00206     __HAL_RCC_TIM1_CLK_DISABLE();
00207   /* USER CODE BEGIN TIM1_MspDeInit 1 */
00208
00209   /* USER CODE END TIM1_MspDeInit 1 */
00210   }
00211
00212 }
00213
00220 void HAL_UART_MspInit(UART_HandleTypeDef* huart)
00221 {
00222   GPIO_InitTypeDef GPIO_InitStruct = {0};
00223   if(huart->Instance==USART1)
00224   {
00225   /* USER CODE BEGIN USART1_MspInit 0 */
00226
00227   /* USER CODE END USART1_MspInit 0 */
00228     /* Peripheral clock enable */
00229     __HAL_RCC_USART1_CLK_ENABLE();
00230
00231     __HAL_RCC_GPIOA_CLK_ENABLE();
00236     GPIO_InitStruct.Pin = GPIO_PIN_9;
00237     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
00238     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
00239     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00240
00241     GPIO_InitStruct.Pin = GPIO_PIN_10;
00242     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
00243     GPIO_InitStruct.Pull = GPIO_NOPULL;
00244     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00245
00246   /* USER CODE BEGIN USART1_MspInit 1 */
00247
00248   /* USER CODE END USART1_MspInit 1 */
00249   }
00250
00251 }
00252
00259 void HAL_UART_MspDeInit(UART_HandleTypeDef* huart)
00260 {
00261   if(huart->Instance==USART1)
00262   {
00263   /* USER CODE BEGIN USART1_MspDeInit 0 */
00264
00265   /* USER CODE END USART1_MspDeInit 0 */
00266     /* Peripheral clock disable */
00267     __HAL_RCC_USART1_CLK_DISABLE();
00268
00273     HAL_GPIO_DeInit(GPIOA, GPIO_PIN_9|GPIO_PIN_10);
00274
00275   /* USER CODE BEGIN USART1_MspDeInit 1 */
00276
00277   /* USER CODE END USART1_MspDeInit 1 */
00278   }
```

```
00279
00280 }
00281
00282 /* USER CODE BEGIN 1 */
00283
00284 /* USER CODE END 1 */
00285
00286 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

4.7 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

4.7.1 Descrição detalhada

HAL time base based on the hardware TIM.

Atenção

Definição no arquivo stm32f1xx_hal_timebase_tim.c.

4.7.2 Funções

4.7.2.1 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 }
```

4.7.2.2 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 }
```

4.7.2.3 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 }
```

4.7.3 Variáveis

4.7.3.1 htim4

```
TIM_HandleTypeDef htim4
```

Definição na linha 29 do arquivo stm32f1xx_hal_timebase_tim.c.

4.8 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)
00094 {
00095   /* Disable TIM4 update Interrupt */
00096   __HAL_TIM_DISABLE_IT(&htim4, TIM_IT_UPDATE);
00097 }
00098
00105 void HAL_ResumeTick(void)
00106 {
00107   /* Enable TIM4 Update interrupt */
00108   __HAL_TIM_ENABLE_IT(&htim4, TIM_IT_UPDATE);
00109 }
00110
00111 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

4.9 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

4.9.1 Descrição detalhada

Interrupt Service Routines.

Atenção

Definição no arquivo stm32f1xx_it.c.

4.9.2 Funções

4.9.2.1 BusFault_Handler()

```
void BusFault_Handler (
              void  )
```

This function handles Prefetch fault, memory access fault.

Definição na linha 116 do arquivo stm32f1xx_it.c.

```
00117 {
00118   /* USER CODE BEGIN BusFault_IRQn 0 */
00119
00120   /* USER CODE END BusFault_IRQn 0 */
00121   while (1)
00122   {
00123     /* USER CODE BEGIN W1_BusFault_IRQn 0 */
00124     /* USER CODE END W1_BusFault_IRQn 0 */
00125   }
00126 }
```

4.9.2.2 DebugMon_Handler()

```
void DebugMon_Handler (
              void  )
```

This function handles Debug monitor.

Definição na linha 146 do arquivo stm32f1xx_it.c.

```
00147 {
00148   /* USER CODE BEGIN DebugMonitor_IRQn 0 */
00149
00150   /* USER CODE END DebugMonitor_IRQn 0 */
00151   /* USER CODE BEGIN DebugMonitor_IRQn 1 */
00152
00153   /* USER CODE END DebugMonitor_IRQn 1 */
00154 }
```

4.9.2.3 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 }
```

4.9.2.4 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 }
```

4.9.2.5 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 }
```

4.9.2.6 TIM4_IRQHandler()

```
void TIM4_IRQHandler (
             void  )
```

This function handles TIM4 global interrupt.

Definição na linha 166 do arquivo stm32f1xx_it.c.

```
00167 {
00168   /* USER CODE BEGIN TIM4_IRQn 0 */
00169
00170   /* USER CODE END TIM4_IRQn 0 */
00171   HAL_TIM_IRQHandler(&htim4);
00172   /* USER CODE BEGIN TIM4_IRQn 1 */
00173
00174   /* USER CODE END TIM4_IRQn 1 */
00175 }
```

4.9.2.7 UsageFault_Handler()

```
void UsageFault_Handler (
             void  )
```

This function handles Undefined instruction or illegal state.

Definição na linha 131 do arquivo stm32f1xx_it.c.

```
00132 {
00133   /* USER CODE BEGIN UsageFault_IRQn 0 */
00134
00135   /* USER CODE END UsageFault_IRQn 0 */
00136   while (1)
00137   {
00138     /* USER CODE BEGIN W1_UsageFault_IRQn 0 */
00139     /* USER CODE END W1_UsageFault_IRQn 0 */
00140   }
00141 }
```

4.9.3 Variáveis

4.9.3.1 htim4

```
TIM_HandleTypeDef htim4  [extern]
```

Definição na linha 29 do arquivo stm32f1xx_hal_timebase_tim.c.

4.10 stm32f1xx_it.c

Vá para a documentação desse arquivo.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Includes ------------------------------------------------------------------*/
00022 #include "main.h"
00023 #include "stm32f1xx_it.h"
00024 /* Private includes ----------------------------------------------------------*/
00025 /* USER CODE BEGIN Includes */
00026 /* USER CODE END Includes */
00027
00028 /* Private typedef -----------------------------------------------------------*/
00029 /* USER CODE BEGIN TD */
00030
00031 /* USER CODE END TD */
00032
00033 /* Private define ------------------------------------------------------------*/
00034 /* USER CODE BEGIN PD */
00035
00036 /* USER CODE END PD */
00037
00038 /* Private macro -------------------------------------------------------------*/
00039 /* USER CODE BEGIN PM */
00040
00041 /* USER CODE END PM */
00042
00043 /* Private variables ---------------------------------------------------------*/
00044 /* USER CODE BEGIN PV */
00045
00046 /* USER CODE END PV */
00047
00048 /* Private function prototypes -----------------------------------------------*/
00049 /* USER CODE BEGIN PFP */
00050
00051 /* USER CODE END PFP */
00052
00053 /* Private user code ---------------------------------------------------------*/
00054 /* USER CODE BEGIN 0 */
00055
00056 /* USER CODE END 0 */
00057
00058 /* External variables --------------------------------------------------------*/
00059 extern TIM_HandleTypeDef htim4;
00060
00061 /* USER CODE BEGIN EV */
00062
00063 /* USER CODE END EV */
00064
00065 /******************************************************************************/
00066 /*           Cortex-M3 Processor Interruption and Exception Handlers          */
00067 /******************************************************************************/
00071 void NMI_Handler(void)
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 }
00082
00086 void HardFault_Handler(void)
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 }
00097
00101 void MemManage_Handler(void)
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 }
00112
00116 void BusFault_Handler(void)
00117 {
00118   /* USER CODE BEGIN BusFault_IRQn 0 */
00119
00120   /* USER CODE END BusFault_IRQn 0 */
00121   while (1)
00122   {
00123     /* USER CODE BEGIN W1_BusFault_IRQn 0 */
00124     /* USER CODE END W1_BusFault_IRQn 0 */
00125   }
00126 }
00127
00131 void UsageFault_Handler(void)
00132 {
00133   /* USER CODE BEGIN UsageFault_IRQn 0 */
00134
00135   /* USER CODE END UsageFault_IRQn 0 */
00136   while (1)
00137   {
00138     /* USER CODE BEGIN W1_UsageFault_IRQn 0 */
00139     /* USER CODE END W1_UsageFault_IRQn 0 */
00140   }
00141 }
00142
00146 void DebugMon_Handler(void)
00147 {
00148   /* USER CODE BEGIN DebugMonitor_IRQn 0 */
00149
00150   /* USER CODE END DebugMonitor_IRQn 0 */
00151   /* USER CODE BEGIN DebugMonitor_IRQn 1 */
00152
00153   /* USER CODE END DebugMonitor_IRQn 1 */
00154 }
00155
00156 /******************************************************************************/
00157 /* STM32F1xx Peripheral Interrupt Handlers                                    */
00158 /* Add here the Interrupt Handlers for the used peripherals.                  */
00159 /* For the available peripheral interrupt handler names,                      */
00160 /* please refer to the startup file (startup_stm32f1xx.s).                    */
00161 /******************************************************************************/
00162
00166 void TIM4_IRQHandler(void)
00167 {
00168   /* USER CODE BEGIN TIM4_IRQn 0 */
00169
00170   /* USER CODE END TIM4_IRQn 0 */
00171   HAL_TIM_IRQHandler(&htim4);
00172   /* USER CODE BEGIN TIM4_IRQn 1 */
00173
00174   /* USER CODE END TIM4_IRQn 1 */
00175 }
00176
00177 /* USER CODE BEGIN 1 */
00178
00179 /* USER CODE END 1 */
00180 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

4.11 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

4.11.1 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.

4.11.2 Funções

4.11.2.1 __attribute__()

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

Definição na linha 69 do arquivo syscalls.c.

```
00070 {
00071     int DataIdx;
00072
00073     for (DataIdx = 0; DataIdx < len; DataIdx++)
00074     {
00075         *ptr++ = __io_getchar();
00076     }
00077
00078 return len;
00079 }
```

4.11.2.2 __io_getchar()

```
int __io_getchar (
              void  )
```

Definição na linha 39 do arquivo syscalls.c.

```
00043                 { 0 };
```

4.11.2.3 _close()

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

Definição na linha 92 do arquivo syscalls.c.

```
00093 {
00094     return -1;
00095 }
```

4.11.2.4 _execve()

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

Definição na linha 155 do arquivo syscalls.c.

```
00156 {
00157     errno = ENOMEM;
00158     return -1;
00159 }
```

4.11.2.5 _exit()

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

Definição na linha 63 do arquivo syscalls.c.

```
00064 {
00065     _kill(status, -1);
00066     while (1) {}        /* Make sure we hang here */
00067 }
```

4.11.2.6 _fork()

```
int _fork (
            void  )
```

Definição na linha 149 do arquivo syscalls.c.

```
00150 {
00151     errno = EAGAIN;
00152     return -1;
00153 }
```

4.11.2.7 _fstat()

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

Definição na linha 98 do arquivo syscalls.c.

```
00099 {
00100     st->st_mode = S_IFCHR;
00101     return 0;
00102 }
```

4.11.2.8 _getpid()

```
int _getpid (
            void  )
```

Definição na linha 52 do arquivo syscalls.c.

```
00053 {
00054     return 1;
00055 }
```

4.11.2.9 _isatty()

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

Definição na linha 104 do arquivo syscalls.c.

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

4.11.2.10 _kill()

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

Definição na linha 57 do arquivo syscalls.c.

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

4.11.2.11 _link()

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

Definição na linha 143 do arquivo syscalls.c.

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

4.11.2.12 _lseek()

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

Definição na linha 109 do arquivo syscalls.c.

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

4.11.2.13 _open()

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

Definição na linha 114 do arquivo syscalls.c.

```
00115 {
00116     /* Pretend like we always fail */
00117     return -1;
00118 }
```

4.11.2.14 _stat()

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

Definição na linha 137 do arquivo syscalls.c.

```
00138 {
00139     st->st_mode = S_IFCHR;
00140     return 0;
00141 }
```

4.11.2.15 _times()

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

Definição na linha 132 do arquivo syscalls.c.

```
00133 {
00134     return -1;
00135 }
```

4.11.2.16 _unlink()

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

Definição na linha 126 do arquivo syscalls.c.

```
00127 {
00128     errno = ENOENT;
00129     return -1;
00130 }
```

4.11.2.17 _wait()

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

Definição na linha 120 do arquivo syscalls.c.

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

4.11.2.18 initialise_monitor_handles()

```
void initialise_monitor_handles ( )
```

Definição na linha 48 do arquivo syscalls.c.

```
00049 {
00050 }
```

4.11.3 Variáveis

4.11.3.1 environ

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

Definição na linha 44 do arquivo syscalls.c.

4.12 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 {
```

```
00050 }
00051
00052 int _getpid(void)
00053 {
00054     return 1;
00055 }
00056
00057 int _kill(int pid, int sig)
00058 {
00059     errno = EINVAL;
00060     return -1;
00061 }
00062
00063 void _exit (int status)
00064 {
00065     _kill(status, -1);
00066     while (1) {}        /* Make sure we hang here */
00067 }
00068
00069 __attribute__((weak)) int _read(int file, char *ptr, int len)
00070 {
00071     int DataIdx;
00072
00073     for (DataIdx = 0; DataIdx < len; DataIdx++)
00074     {
00075         *ptr++ = __io_getchar();
00076     }
00077
00078 return len;
00079 }
00080
00081 __attribute__((weak)) int _write(int file, char *ptr, int len)
00082 {
00083     int DataIdx;
00084
00085     for (DataIdx = 0; DataIdx < len; DataIdx++)
00086     {
00087         __io_putchar(*ptr++);
00088     }
00089     return len;
00090 }
00091
00092 int _close(int file)
00093 {
00094     return -1;
00095 }
00096
00097
00098 int _fstat(int file, struct stat *st)
00099 {
00100     st->st_mode = S_IFCHR;
00101     return 0;
00102 }
00103
00104 int _isatty(int file)
00105 {
00106     return 1;
00107 }
00108
00109 int _lseek(int file, int ptr, int dir)
00110 {
00111     return 0;
00112 }
00113
00114 int _open(char *path, int flags, ...)
00115 {
00116     /* Pretend like we always fail */
00117     return -1;
00118 }
00119
00120 int _wait(int *status)
00121 {
00122     errno = ECHILD;
00123     return -1;
00124 }
00125
00126 int _unlink(char *name)
00127 {
00128     errno = ENOENT;
00129     return -1;
00130 }
00131
00132 int _times(struct tms *buf)
00133 {
00134     return -1;
00135 }
00136
```

```
00137 int _stat(char *file, struct stat *st)
00138 {
00139     st->st_mode = S_IFCHR;
00140     return 0;
00141 }
00142
00143 int _link(char *old, char *new)
00144 {
00145     errno = EMLINK;
00146     return -1;
00147 }
00148
00149 int _fork(void)
00150 {
00151     errno = EAGAIN;
00152     return -1;
00153 }
00154
00155 int _execve(char *name, char **argv, char **env)
00156 {
00157     errno = ENOMEM;
00158     return -1;
00159 }
```

4.13 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

4.13.1 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.

4.13.2 Funções

4.13.2.1 _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   }
```

```
00068
00069   /* Protect heap from growing into the reserved MSP stack */
00070   if (__sbrk_heap_end + incr > max_heap)
00071   {
00072     errno = ENOMEM;
00073     return (void *)-1;
00074   }
00075
00076   prev_heap_end = __sbrk_heap_end;
00077   __sbrk_heap_end += incr;
00078
00079   return (void *)prev_heap_end;
00080 }
```

4.14 sysmem.c

Vá para a documentação desse arquivo.

```
00001
00024 /* Includes */
00025 #include <errno.h>
00026 #include <stdint.h>
00027
00031 static uint8_t *__sbrk_heap_end = NULL;
00032
00054 void *_sbrk(ptrdiff_t incr)
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   }
00068
00069   /* Protect heap from growing into the reserved MSP stack */
00070   if (__sbrk_heap_end + incr > max_heap)
00071   {
00072     errno = ENOMEM;
00073     return (void *)-1;
00074   }
00075
00076   prev_heap_end = __sbrk_heap_end;
00077   __sbrk_heap_end += incr;
00078
00079   return (void *)prev_heap_end;
00080 }
```

4.15 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}

4.15.1 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.

4.16 system_stm32f1xx.c

Vá para a documentação desse arquivo.

```
00001
00059 #include "stm32f1xx.h"
00060
00077 #if !defined  (HSE_VALUE)
00078   #define HSE_VALUE               8000000U
00080 #endif /* HSE_VALUE */
00081
00082 #if !defined  (HSI_VALUE)
00083   #define HSI_VALUE               8000000U
00085 #endif /* HSI_VALUE */
00086
00088 #if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) || defined(STM32F103xE) ||
       defined(STM32F103xG)
00089 /* #define DATA_IN_ExtSRAM */
00090 #endif /* STM32F100xE || STM32F101xE || STM32F101xG || STM32F103xE || STM32F103xG */
00091
00092 /* Note: Following vector table addresses must be defined in line with linker
00093          configuration. */
00097 /* #define USER_VECT_TAB_ADDRESS */
00098
00099 #if defined(USER_VECT_TAB_ADDRESS)
00102 /* #define VECT_TAB_SRAM */
00103 #if defined(VECT_TAB_SRAM)
00104 #define VECT_TAB_BASE_ADDRESS   SRAM_BASE
00106 #define VECT_TAB_OFFSET         0x00000000U
00108 #else
00109 #define VECT_TAB_BASE_ADDRESS   FLASH_BASE
00111 #define VECT_TAB_OFFSET         0x00000000U
00113 #endif /* VECT_TAB_SRAM */
00114 #endif /* USER_VECT_TAB_ADDRESS */
00115
00116 /******************************************************************************/
00117
00134   /* This variable is updated in three ways:
00135       1) by calling CMSIS function SystemCoreClockUpdate()
00136       2) by calling HAL API function HAL_RCC_GetHCLKFreq()
00137       3) each time HAL_RCC_ClockConfig() is called to configure the system clock frequency
00138          Note: If you use this function to configure the system clock; then there
00139                is no need to call the 2 first functions listed above, since SystemCoreClock
00140                variable is updated automatically.
00141   */
00142 uint32_t SystemCoreClock = 16000000;
00143 const uint8_t AHBPrescTable[16U] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9};
00144 const uint8_t APBPrescTable[8U] =  {0, 0, 0, 0, 1, 2, 3, 4};
00145
00154 #if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) || defined(STM32F103xE) ||
       defined(STM32F103xG)
00155 #ifdef DATA_IN_ExtSRAM
00156   static void SystemInit_ExtMemCtl(void);
00157 #endif /* DATA_IN_ExtSRAM */
00158 #endif /* STM32F100xE || STM32F101xE || STM32F101xG || STM32F103xE || STM32F103xG */
00159
00176 void SystemInit (void)
00177 {
00178 #if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) || defined(STM32F103xE) ||
       defined(STM32F103xG)
00179   #ifdef DATA_IN_ExtSRAM
00180     SystemInit_ExtMemCtl();
00181   #endif /* DATA_IN_ExtSRAM */
00182 #endif
00183
00184   /* Configure the Vector Table location ----------------------------------*/
00185 #if defined(USER_VECT_TAB_ADDRESS)
00186   SCB->VTOR = VECT_TAB_BASE_ADDRESS | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM. */
00187 #endif /* USER_VECT_TAB_ADDRESS */
00188 }
00189
00225 void SystemCoreClockUpdate (void)
00226 {
00227   uint32_t tmp = 0U, pllmull = 0U, pllsource = 0U;
00228
00229 #if defined(STM32F105xC) || defined(STM32F107xC)
00230   uint32_t prediv1source = 0U, prediv1factor = 0U, prediv2factor = 0U, pll2mull = 0U;
00231 #endif /* STM32F105xC */
00232
00233 #if defined(STM32F100xB) || defined(STM32F100xE)
00234   uint32_t prediv1factor = 0U;
00235 #endif /* STM32F100xB or STM32F100xE */
00236
00237   /* Get SYSCLK source -------------------------------------------------------*/
00238   tmp = RCC->CFGR & RCC_CFGR_SWS;
00239
```

```
00240   switch (tmp)
00241   {
00242     case 0x00U:  /* HSI used as system clock */
00243       SystemCoreClock = HSI_VALUE;
00244       break;
00245     case 0x04U:  /* HSE used as system clock */
00246       SystemCoreClock = HSE_VALUE;
00247       break;
00248     case 0x08U:  /* PLL used as system clock */
00249
00250       /* Get PLL clock source and multiplication factor ----------------------*/
00251       pllmull = RCC->CFGR & RCC_CFGR_PLLMULL;
00252       pllsource = RCC->CFGR & RCC_CFGR_PLLSRC;
00253
00254 #if !defined(STM32F105xC) && !defined(STM32F107xC)
00255       pllmull = ( pllmull >> 18U) + 2U;
00256
00257       if (pllsource == 0x00U)
00258       {
00259         /* HSI oscillator clock divided by 2 selected as PLL clock entry */
00260         SystemCoreClock = (HSI_VALUE >> 1U) * pllmull;
00261       }
00262       else
00263       {
00264  #if defined(STM32F100xB) || defined(STM32F100xE)
00265        prediv1factor = (RCC->CFGR2 & RCC_CFGR2_PREDIV1) + 1U;
00266        /* HSE oscillator clock selected as PREDIV1 clock entry */
00267        SystemCoreClock = (HSE_VALUE / prediv1factor) * pllmull;
00268  #else
00269        /* HSE selected as PLL clock entry */
00270        if ((RCC->CFGR & RCC_CFGR_PLLXTPRE) != (uint32_t)RESET)
00271        {/* HSE oscillator clock divided by 2 */
00272          SystemCoreClock = (HSE_VALUE >> 1U) * pllmull;
00273        }
00274        else
00275        {
00276          SystemCoreClock = HSE_VALUE * pllmull;
00277        }
00278  #endif
00279       }
00280 #else
00281       pllmull = pllmull >> 18U;
00282
00283       if (pllmull != 0x0DU)
00284       {
00285          pllmull += 2U;
00286       }
00287       else
00288       { /* PLL multiplication factor = PLL input clock * 6.5 */
00289         pllmull = 13U / 2U;
00290       }
00291
00292       if (pllsource == 0x00U)
00293       {
00294         /* HSI oscillator clock divided by 2 selected as PLL clock entry */
00295         SystemCoreClock = (HSI_VALUE >> 1U) * pllmull;
00296       }
00297       else
00298       {/* PREDIV1 selected as PLL clock entry */
00299
00300         /* Get PREDIV1 clock source and division factor */
00301         prediv1source = RCC->CFGR2 & RCC_CFGR2_PREDIV1SRC;
00302         prediv1factor = (RCC->CFGR2 & RCC_CFGR2_PREDIV1) + 1U;
00303
00304         if (prediv1source == 0U)
00305         {
00306           /* HSE oscillator clock selected as PREDIV1 clock entry */
00307           SystemCoreClock = (HSE_VALUE / prediv1factor) * pllmull;
00308         }
00309         else
00310         {/* PLL2 clock selected as PREDIV1 clock entry */
00311
00312           /* Get PREDIV2 division factor and PLL2 multiplication factor */
00313           prediv2factor = ((RCC->CFGR2 & RCC_CFGR2_PREDIV2) >> 4U) + 1U;
00314           pll2mull = ((RCC->CFGR2 & RCC_CFGR2_PLL2MUL) >> 8U) + 2U;
00315           SystemCoreClock = (((HSE_VALUE / prediv2factor) * pll2mull) / prediv1factor) * pllmull;

00316         }
00317       }
00318 #endif /* STM32F105xC */
00319       break;
00320
00321     default:
00322       SystemCoreClock = HSI_VALUE;
00323       break;
00324   }
00325
```

```
00326   /* Compute HCLK clock frequency ----------------*/
00327   /* Get HCLK prescaler */
00328   tmp = AHBPrescTable[((RCC->CFGR & RCC_CFGR_HPRE) » 4U)];
00329   /* HCLK clock frequency */
00330   SystemCoreClock »= tmp;
00331 }
00332
00333 #if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) || defined(STM32F103xE) ||
     defined(STM32F103xG)
00340 #ifdef DATA_IN_ExtSRAM
00350 void SystemInit_ExtMemCtl(void)
00351 {
00352   __IO uint32_t tmpreg;
00356   /* Enable FSMC clock */
00357   RCC->AHBENR = 0x00000114U;
00358
00359   /* Delay after an RCC peripheral clock enabling */
00360   tmpreg = READ_BIT(RCC->AHBENR, RCC_AHBENR_FSMCEN);
00361
00362   /* Enable GPIOD, GPIOE, GPIOF and GPIOG clocks */
00363   RCC->APB2ENR = 0x000001E0U;
00364
00365   /* Delay after an RCC peripheral clock enabling */
00366   tmpreg = READ_BIT(RCC->APB2ENR, RCC_APB2ENR_IOPDEN);
00367
00368   (void)(tmpreg);
00369
00370 /* ---------------  SRAM Data lines, NOE and NWE configuration ---------------*/
00371 /*----------------  SRAM Address lines configuration -------------------------*/
00372 /*----------------  NOE and NWE configuration ---------------------------------*/
00373 /*----------------  NE3 configuration -----------------------------------------*/
00374 /*----------------  NBL0, NBL1 configuration ----------------------------------*/
00375
00376   GPIOD->CRL = 0x44BB44BBU;
00377   GPIOD->CRH = 0xBBBBBBBBU;
00378
00379   GPIOE->CRL = 0xB44444BBU;
00380   GPIOE->CRH = 0xBBBBBBBBU;
00381
00382   GPIOF->CRL = 0x44BBBBBBU;
00383   GPIOF->CRH = 0xBBBB4444U;
00384
00385   GPIOG->CRL = 0x44BBBBBBU;
00386   GPIOG->CRH = 0x444B4B44U;
00387
00388 /*----------------  FSMC Configuration ----------------------------------------*/
00389 /*----------------  Enable FSMC Bank1_SRAM Bank -------------------------------*/
00390
00391   FSMC_Bank1->BTCR[4U] = 0x00001091U;
00392   FSMC_Bank1->BTCR[5U] = 0x00110212U;
00393 }
00394 #endif /* DATA_IN_ExtSRAM */
00395 #endif /* STM32F100xE || STM32F101xE || STM32F101xG || STM32F103xE || STM32F103xG */
00396
00408 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Índice Remissivo