

Controle de temperatura

Gerado por Doxygen 1.9.2

1 Índice dos Arquivos	1
1.1 Lista de Arquivos	1
2 Arquivos	3
2.1 Referência do Arquivo D:/BACKUP/Faculdade/16_Embarcados/Controlador_Temperatura/Controle- de-temperatura/Codigos/controle_temperatura_RTOS/Core/Src/main.c	3
2.1.1 Descrição detalhada	4
2.1.2 Definições e macros	5
2.1.2.1 CSdis	5
2.1.2.2 CSen	5
2.1.2.3 k	5
2.1.2.4 ksat	5
2.1.2.5 p	5
2.1.2.6 r	6
2.1.2.7 SCK_H	6
2.1.2.8 SCK_L	6
2.1.3 Funções	6
2.1.3.1 Control_taskF()	6
2.1.3.2 Display_taskF()	7
2.1.3.3 Error_Handler()	8
2.1.3.4 Filter_taskF()	8
2.1.3.5 HAL_TIM_PeriodElapsedCallback()	9
2.1.3.6 main()	10
2.1.3.7 SystemClock_Config()	11
2.1.3.8 Temp_taskF()	11
2.1.4 Variáveis	12
2.1.4.1 Control_Task	12
2.1.4.2 Display_Task	12
2.1.4.3 dutyCycle	13
2.1.4.4 Filter_Task	13
2.1.4.5 filteredTemp	13
2.1.4.6 filteredTempQueue	13
2.1.4.7 hadc1	13
2.1.4.8 htim1	13
2.1.4.9 huart1	14
2.1.4.10 ref	14
2.1.4.11 Temp_Task	14
2.1.4.12 tempFilter	14
2.1.4.13 tempQueue	14
2.2 main.c	15
Índice Remissivo	21

Capítulo 1

Índice dos Arquivos

1.1 Lista de Arquivos

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

D:/BACKUP/Faculdade/16_Embarcados/Controlador_Temperatura/Controle-de-temperatura/Codigos/controle↵ _temperatura_RTOS/Core/Src/ main.c	
Corpo principal do programa	3

Capítulo 2

Arquivos

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

Corpo principal do programa.

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

Definições e Macros

- #define **r** 0.000210115034038755f
- #define **p** 1.0f
- #define **k** 0.0638221651196478f
- #define **ksat** 0.1f
- #define **CSen** HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_RESET);
- #define **CSdis** HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
- #define **SCK_H** HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
- #define **SCK_L** HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);

Funções

- void **SystemClock_Config** (void)
Configuração do clock do sistema.
- void **Temp_taskF** (void *pvParameters)
Tarefa de leitura da temperatura.
- void **Filter_taskF** (void *pvParameters)

- Tarefa de filtro da variável temperatura.*
 - void [Control_taskF](#) (void *pvParameters)
- Tarefa de atualização da referencia, calculo e execução da lei de controle.*
 - void [Display_taskF](#) (void *pvParameters)
- Tarefa de atualização das variáveis no display.*
 - int [main](#) (void)
- ponto de entrada da aplicação.*
 - void [HAL_TIM_PeriodElapsedCallback](#) (TIM_HandleTypeDef *htim)
- chamada da função de período*
 - void [Error_Handler](#) (void)
- Função executada em caso de erro na aplicação.*

Variáveis

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

2.1.1 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](#).

2.1.2 Definições e macros

2.1.2.1 CSdis

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

Definição na linha 36 do arquivo [main.c](#).

2.1.2.2 CSen

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

Definição na linha 35 do arquivo [main.c](#).

2.1.2.3 k

```
#define k 0.0638221651196478f
```

Definição na linha 32 do arquivo [main.c](#).

2.1.2.4 ksat

```
#define ksat 0.1f
```

Definição na linha 33 do arquivo [main.c](#).

2.1.2.5 p

```
#define p 1.0f
```

Definição na linha 31 do arquivo [main.c](#).

2.1.2.6 r

```
#define r 0.000210115034038755f
```

Definição na linha 30 do arquivo [main.c](#).

2.1.2.7 SCK_H

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

Definição na linha 38 do arquivo [main.c](#).

2.1.2.8 SCK_L

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

Definição na linha 39 do arquivo [main.c](#).

2.1.3 Funções

2.1.3.1 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 resfriamento

Parâmetros

<i>*pvParameters</i>	(não utilizado) permite iniciar a função com valor inicial
----------------------	--

Valores Retornados

None	
------	--

Definição na linha 386 do arquivo main.c.

```

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

```

2.1.3.2 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

<i>*pvParameters</i>	(não utilizado) permite iniciar a função com valor inicial
----------------------	--

Valores Retornados

None	
------	--

Definição na linha 456 do arquivo [main.c](#).

```

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

2.1.3.3 Error_Handler()

```

void Error_Handler (
    void )
```

Função executada em caso de erro na aplicação.

Valores Retornados

None	
------	--

Definição na linha 501 do arquivo [main.c](#).

```

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

2.1.3.4 Filter_taskF()

```

void Filter_taskF (
    void * pvParameters )
```

Tarefa de filtro da variável temperatura.

Observação

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

Parâmetros

<i>*pvParameters</i>	(não utilizado) permite iniciar a função com valor inicial
----------------------	--

Valores Retornados

<i>None</i>	
-------------	--

Definição na linha 357 do arquivo main.c.

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

2.1.3.5 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

<i>htim</i>	: TIM handle
-------------	--------------

Valores Retornados

<i>None</i>	
-------------	--

Definição na linha 491 do arquivo [main.c](#).

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

2.1.3.6 main()

```
int main (
    void )
```

ponto de entrada da aplicação.

Valores Retornados

<i>int</i>	
------------	--

Definição na linha 76 do arquivo [main.c](#).

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

2.1.3.7 SystemClock_Config()

```
void SystemClock_Config (
    void )
```

Configuração do clock do sistema.

Valores Retornados

None	
------	--

Definição na linha 132 do arquivo main.c.

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

2.1.3.8 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 através 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 inicial
---------------	--

Valores Retornados

None	
------	--

Definição na linha 315 do arquivo [main.c](#).

```

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

```

2.1.4 Variáveis

2.1.4.1 Control_Task

TaskHandle_t Control_Task

Definição na linha 49 do arquivo [main.c](#).

2.1.4.2 Display_Task

TaskHandle_t Display_Task

Definição na linha 50 do arquivo [main.c](#).

2.1.4.3 dutyCycle

```
float dutyCycle = 0
```

Definição na linha 59 do arquivo [main.c](#).

2.1.4.4 Filter_Task

```
TaskHandle_t Filter_Task
```

Definição na linha 48 do arquivo [main.c](#).

2.1.4.5 filteredTemp

```
float filteredTemp = 0
```

Definição na linha 57 do arquivo [main.c](#).

2.1.4.6 filteredTempQueue

```
QueueHandle_t filteredTempQueue
```

Definição na linha 53 do arquivo [main.c](#).

2.1.4.7 hadc1

```
ADC_HandleTypeDef hadc1
```

Definição na linha 41 do arquivo [main.c](#).

2.1.4.8 htim1

```
TIM_HandleTypeDef htim1
```

Definição na linha 43 do arquivo [main.c](#).

2.1.4.9 huart1

```
UART_HandleTypeDef huart1
```

Definição na linha 45 do arquivo [main.c](#).

2.1.4.10 ref

```
float ref = 0
```

Definição na linha 58 do arquivo [main.c](#).

2.1.4.11 Temp_Task

```
TaskHandle_t Temp_Task
```

Definição na linha 47 do arquivo [main.c](#).

2.1.4.12 tempFilter

```
FIRFilter tempFilter
```

Definição na linha 55 do arquivo [main.c](#).

2.1.4.13 tempQueue

```
QueueHandle_t tempQueue
```

Definição na linha 52 do arquivo [main.c](#).

2.2 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 r 0.000210115034038755f
00031 #define p 1.0f
00032 #define k 0.0638221651196478f
00033 #define ksat 0.1f
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 ADC_HandleTypeDef hadc1;
00042
00043 TIM_HandleTypeDef htim1;
00044
00045 UART_HandleTypeDef huart1;
00046
00047 TaskHandle_t Temp_Task;
00048 TaskHandle_t Filter_Task;
00049 TaskHandle_t Control_Task;
00050 TaskHandle_t Display_Task;
00051
00052 QueueHandle_t tempQueue;
00053 QueueHandle_t filteredTempQueue;
00054
00055 FIRFilter tempFilter;
00056
00057 float filteredTemp = 0;
00058 float ref = 0;
00059 float dutyCycle = 0;
00060
00061 void SystemClock_Config(void);
00062 static void MX_GPIO_Init(void);
00063 static void MX_ADC1_Init(void);
00064 static void MX_TIM1_Init(void);
00065 static void MX_USART1_UART_Init(void);
00066
00067 void Temp_taskF(void *pvParameters);
00068 void Filter_taskF(void *pvParameters);
00069 void Control_taskF(void *pvParameters);
00070 void Display_taskF(void *pvParameters);
00071
00076 int main(void) {
00077     /* Reinicia todos os periféricos, inicializa a interface flash e o systick */
00078     HAL_Init();
00079
00080     /* Configura o clock do sistema */
00081     SystemClock_Config();
00082
00083     /* Inicializa todos os periféricos configurados */
00084     MX_GPIO_Init();
00085     MX_ADC1_Init();
00086     MX_TIM1_Init();
00087     MX_USART1_UART_Init();
00088
00089     /* Inicializa o conversor AD */
00090     if (HAL_ADC_Start(&hadc1) != HAL_OK) {
00091         Error_Handler();
00092     }
00093
00094     /* Inicializa o timer1 em modo PWM */
00095     if (HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1) != HAL_OK) {
00096         Error_Handler();
00097     }
00098
00099     /* Inicializa o filtro FIR */
00100     FIRFilter_Init(&tempFilter);
00101
00102     /* Cria a fila de leituras de temperatura bruta */

```

```

00103     tempQueue = xQueueCreate(1, sizeof(float));
00104     if (tempQueue == 0) {
00105         Error_Handler();
00106     }
00107
00108     /* Cria a fila de leituras de temperatura filtrada */
00109     filteredTempQueue = xQueueCreate(1, sizeof(float));
00110     if (tempQueue == 0) {
00111         Error_Handler();
00112     }
00113
00114     /* Cria tasks na pilha do sistema */
00115     xTaskCreate(Temp_taskF, "TempTask", 128, NULL, 3, &Temp_Task);
00116     xTaskCreate(Filter_taskF, "FilterTask", 128, NULL, 2, &Filter_Task);
00117     xTaskCreate(Control_taskF, "ControlTask", 128, NULL, 4, &Control_Task);
00118     xTaskCreate(Display_taskF, "DisplayTask", 128, NULL, 1, &Display_Task);
00119
00120     /* Inicializa o escalonador */
00121     vTaskStartScheduler();
00122
00123     /* Loop infinito */
00124     while (1) {
00125     }
00126 }
00127
00132 void SystemClock_Config(void) {
00133     RCC_OscInitTypeDef RCC_OscInitStruct = { 0 };
00134     RCC_ClkInitTypeDef RCC_ClkInitStruct = { 0 };
00135     RCC_PeriphCLKInitTypeDef PeriphClkInit = { 0 };
00136
00137     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
00138     RCC_OscInitStruct.HSEState = RCC_HSE_ON;
00139     RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
00140     RCC_OscInitStruct.HSIState = RCC_HSI_ON;
00141     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
00142     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
00143     RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
00144     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
00145         Error_Handler();
00146     }
00147
00148     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCCLK
00149         | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
00150     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
00151     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
00152     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
00153     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
00154
00155     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK) {
00156         Error_Handler();
00157     }
00158     PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
00159     PeriphClkInit.AdcClockSelection = RCC_ADCCLK2_DIV6;
00160     if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK) {
00161         Error_Handler();
00162     }
00163 }
00164
00170 static void MX_ADC1_Init(void) {
00171
00172     ADC_ChannelConfTypeDef sConfig = { 0 };
00173
00174     hadc1.Instance = ADC1;
00175     hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
00176     hadc1.Init.ContinuousConvMode = ENABLE;
00177     hadc1.Init.DiscontinuousConvMode = DISABLE;
00178     hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
00179     hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
00180     hadc1.Init.NbrOfConversion = 1;
00181     if (HAL_ADC_Init(&hadc1) != HAL_OK) {
00182         Error_Handler();
00183     }
00184
00185     sConfig.Channel = ADC_CHANNEL_0;
00186     sConfig.Rank = ADC_REGULAR_RANK_1;
00187     sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES_5;
00188     if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) {
00189         Error_Handler();
00190     }
00191 }
00192
00198 static void MX_TIM1_Init(void) {
00199
00200     TIM_MasterConfigTypeDef sMasterConfig = { 0 };
00201     TIM_OC_InitTypeDef sConfigOC = { 0 };
00202     TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = { 0 };
00203

```

```

00204     htim1.Instance = TIM1;
00205     htim1.Init.Prescaler = 1;
00206     htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
00207     htim1.Init.Period = 18000 - 1;
00208     htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
00209     htim1.Init.RepetitionCounter = 0;
00210     htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
00211     if (HAL_TIM_PWM_Init(&htim1) != HAL_OK) {
00212         Error_Handler();
00213     }
00214     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
00215     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
00216     if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig)
00217         != HAL_OK) {
00218         Error_Handler();
00219     }
00220     sConfigOC.OCMode = TIM_OCMODE_PWM1;
00221     sConfigOC.Pulse = 0;
00222     sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
00223     sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
00224     sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
00225     sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
00226     sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
00227     if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1)
00228         != HAL_OK) {
00229         Error_Handler();
00230     }
00231     sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
00232     sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
00233     sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
00234     sBreakDeadTimeConfig.DeadTime = 0;
00235     sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
00236     sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
00237     sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
00238     if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig)
00239         != HAL_OK) {
00240         Error_Handler();
00241     }
00242
00243     HAL_TIM_MspPostInit(&htim1);
00244 }
00245
00251 static void MX_USART1_UART_Init(void) {
00252
00253     huart1.Instance = USART1;
00254     huart1.Init.BaudRate = 115200;
00255     huart1.Init.WordLength = UART_WORDLENGTH_8B;
00256     huart1.Init.StopBits = UART_STOPBITS_1;
00257     huart1.Init.Parity = UART_PARITY_NONE;
00258     huart1.Init.Mode = UART_MODE_TX;
00259     huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
00260     huart1.Init.OverSampling = UART_OVERSAMPLING_16;
00261     if (HAL_UART_Init(&huart1) != HAL_OK) {
00262         Error_Handler();
00263     }
00264 }
00265
00271 static void MX_GPIO_Init(void) {
00272     GPIO_InitTypeDef GPIO_InitStruct = { 0 };
00273
00274     __HAL_RCC_GPIOC_CLK_ENABLE();
00275     __HAL_RCC_GPIOD_CLK_ENABLE();
00276     __HAL_RCC_GPIOA_CLK_ENABLE();
00277     __HAL_RCC_GPIOB_CLK_ENABLE();
00278
00279     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_RESET);
00280
00281     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_RESET);
00282
00283     GPIO_InitStruct.Pin = GPIO_PIN_15;
00284     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00285     GPIO_InitStruct.Pull = GPIO_NOPULL;
00286     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00287     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00288
00289     GPIO_InitStruct.Pin = GPIO_PIN_3;
00290     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00291     GPIO_InitStruct.Pull = GPIO_NOPULL;
00292     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00293     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00294
00295     GPIO_InitStruct.Pin = GPIO_PIN_4;
00296     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
00297     GPIO_InitStruct.Pull = GPIO_NOPULL;
00298     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00299
00300     GPIO_InitStruct.Pin = GPIO_PIN_15;

```

```

00301     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00302     GPIO_InitStruct.Pull = GPIO_NOPULL;
00303     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00304     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00305 }
00306
00315 void Temp_taskF(void *pvParameters) {
00316     while (1) {
00317         uint8_t tempdata[16];
00318         uint16_t temp16 = 0;
00319
00320         /* bitbanging protocolo SPI */
00321         CSen
00322         for (int i = 0; i < 16; i++) {
00323             SCK_H
00324             tempdata[i] = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_4);
00325             SCK_L
00326         }
00327         CSdis
00328
00329         /* Conversão temperatura */
00330         if (tempdata[13] == 0) {
00331             for (int n = 1; n < 13; n++) {
00332                 temp16 += tempdata[n] * (2048 / (1 << (n - 1)));
00333             }
00334         }
00335
00336         float temp = (float) temp16 / 4;
00337
00338         /* Adiciona a fila tempQueue */
00339         if (xQueueSend(tempQueue, &temp, 10) == pdPASS) {
00340         }
00341
00342         /* Atraso para definição do período da tarefa */
00343         vTaskDelay(200); /*5Hz frequency*/
00344     }
00345 }
00346
00357 void Filter_taskF(void *pvParameters) {
00358     uint8_t aux = 0;
00359     while (1) {
00360         float rx_temp;
00361         /* Recebe da fila filteredTempQueue */
00362         if (xQueueReceive(tempQueue, &rx_temp, 10)) {
00363             aux++;
00364             /* Chamada do filtro FIR */
00365             filteredTemp = FIRFilter_Update(&tempFilter, rx_temp);
00366         }
00367
00368         if (aux == 5) {
00369             aux = 0;
00370             /* Adiciona a fila filteredTempQueue */
00371             if (xQueueSend(filteredTempQueue, &filteredTemp, 10) == pdPASS) {
00372             }
00373         }
00374     }
00375 }
00376
00386 void Control_taskF(void *pvParameters) {
00387     while (1) {
00388         float rx_filteredTemp;
00389         /* Recebe da fila filteredTempQueue */
00390         if (xQueueReceive(filteredTempQueue, &rx_filteredTemp, 10)) {
00391             /* Leitura da entrada analógica para calculo de referencia */
00392             HAL_ADC_PollForConversion(&hadc1, 10);
00393             ref = (float) HAL_ADC_GetValue(&hadc1) / 27.3; // leitura do potenciometro convertido em
ref até 150°C
00394
00395             /* Lei de controle */
00396             float u;
00397             static float up, uint;
00398             int flag_sat;
00399             float ek = ref - rx_filteredTemp;
00400
00401             /* Controlador bang-bang ventoinha */
00402             if (ek < -15.0) {
00403                 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_SET);
00404             } else {
00405                 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET);
00406             }
00407
00408             /* Anti-windup integrador */
00409             if (!flag_sat) {
00410                 uint = uint * p + r * ek;

```

```

00411
00412     } else if (flag_sat) {
00413         uint = (uint * p + r * ek) * ksat;
00414     }
00415
00416     /* Proporcional */
00417     up = k * ek;
00418
00419     /* Ação de controle */
00420     u = up + uint;
00421
00422     /* Conversão período PWM */
00423     u = u * 4500.0;
00424
00425     /* Limites de saturação de PWM */
00426     if (u > 18000.0) {
00427         u = 18000.0;
00428         flag_sat = 1;
00429     } else if (u < 0) {
00430         u = 0;
00431         flag_sat = 1;
00432     } else {
00433         u = u;
00434         flag_sat = 0;
00435     }
00436
00437     /* Converte período do timer em razão cíclica */
00438     dutyCycle = u / 180.0;
00439
00440     /* Seta periférico PWM */
00441     __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, (uint32_t) u);
00442
00443 }
00444
00445 }
00446 }
00447
00456 void Display_taskF(void *pvParameters) {
00457     while (1) {
00458         char str[100];
00459         /* Fim de comando definido pela API do display */
00460         uint8_t Cmd_End[3] = { 0xFF, 0xFF, 0xFF };
00461
00462         /* Atualiza valor do setpoint */
00463         int32_t number = ref * 100;
00464         sprintf(str, "setPoint.val=%ld", number);
00465         HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
00466         HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);
00467
00468         /* Atualiza valor da variável de processo */
00469         number = filteredTemp * 100;
00470         sprintf(str, "filteredTemp.val=%ld", number);
00471         HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
00472         HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);
00473
00474         /* Atualiza valor da variável manipulada */
00475         number = dutyCycle * 100;
00476         sprintf(str, "dutyCycle.val=%ld", number);
00477         HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str), 10);
00478         HAL_UART_Transmit(&huart1, Cmd_End, 3, 10);
00479
00480         /* Atraso para definição do período da tarefa */
00481         vTaskDelay(1000); /*1Hz frequency*/
00482     }
00483 }
00484
00491 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
00492     if (htim->Instance == TIM4) {
00493         HAL_IncTick();
00494     }
00495 }
00496
00501 void Error_Handler(void) {
00502     __disable_irq();
00503     while (1) {
00504     }
00505 }
00506
00507 #ifdef USE_FULL_ASSERT
00514 void assert_failed(uint8_t *file, uint32_t line)
00515 {
00516 }
00517 #endif

```


Índice Remissivo

Control_Task
 main.c, [12](#)
Control_taskF
 main.c, [6](#)
CSdis
 main.c, [5](#)
CSen
 main.c, [5](#)
D:/BACKUP/Faculdade/16_Embarcados/Controlador_Temperatura/Controlador_Temperatura/Controle_de-temperatura/Codigos/controle_temperatura_RTOS/Controle_Tempera/main.c, [3](#), [15](#)
Display_Task
 main.c, [12](#)
Display_taskF
 main.c, [7](#)
dutyCycle
 main.c, [12](#)
Error_Handler
 main.c, [8](#)
Filter_Task
 main.c, [13](#)
Filter_taskF
 main.c, [8](#)
filteredTemp
 main.c, [13](#)
filteredTempQueue
 main.c, [13](#)
hadc1
 main.c, [13](#)
HAL_TIM_PeriodElapsedCallback
 main.c, [9](#)
htim1
 main.c, [13](#)
huart1
 main.c, [13](#)
k
 main.c, [5](#)
ksat
 main.c, [5](#)
main
 main.c, [10](#)
main.c
 Control_Task, [12](#)
 Control_taskF, [6](#)
CSdis, [5](#)
CSen, [5](#)
Display_Task, [12](#)
Display_taskF, [7](#)
dutyCycle, [12](#)
Error_Handler, [8](#)
Filter_Task, [13](#)
Filter_taskF, [8](#)
filteredTemp, [13](#)
filteredTempQueue, [13](#)
hadc1, [13](#)
HAL_TIM_PeriodElapsedCallback, [9](#)
htim1, [13](#)
huart1, [13](#)
k, [5](#)
ksat, [5](#)
main, [10](#)
p, [5](#)
r, [5](#)
ref, [14](#)
SCK_H, [6](#)
SCK_L, [6](#)
SystemClock_Config, [10](#)
Temp_Task, [14](#)
Temp_taskF, [11](#)
tempFilter, [14](#)
tempQueue, [14](#)
p
 main.c, [5](#)
r
 main.c, [5](#)
ref
 main.c, [14](#)
SCK_H
 main.c, [6](#)
SCK_L
 main.c, [6](#)
SystemClock_Config
 main.c, [10](#)
Temp_Task
 main.c, [14](#)
Temp_taskF
 main.c, [11](#)
tempFilter
 main.c, [14](#)
tempQueue
 main.c, [14](#)