Controle de temperatura

Gerado por Doxygen 1.9.2

Capítulo 1

Índice dos Arquivos

1.1 Lista de Arquivos

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

Capítulo 2

Arquivos

2.1 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

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 DWT_CTRL

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

Definição na linha 41 do arquivo main.c.

2.1.2.4 k

```
#define k 0.06382217f
```

Definição na linha 32 do arquivo main.c.

2.1.2.5 ksat

```
#define ksat 0.10000000f
```

Definição na linha 33 do arquivo main.c.

2.1.2.6 p1

```
#define p1 1.00000000f
```

Definição na linha 31 do arquivo main.c.

2.1.2.7 r1

```
#define r1 0.00021012f
```

Definição na linha 30 do arquivo main.c.

2.1.2.8 SCK_H

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

Definição na linha 38 do arquivo main.c.

2.1.2.9 SCK_L

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

Definição na linha 39 do arquivo main.c.

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 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 }
```

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

∗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 }
```

2.1.3.3 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 }
```

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 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 }
```

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

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 }
```

2.1.3.6 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 }
```

2.1.3.7 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 }
```

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 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 }
```

2.1.4 Variáveis

2.1.4.1 Control_Task

```
TaskHandle_t Control_Task
```

Definição na linha 51 do arquivo main.c.

2.1.4.2 Display_Task

```
TaskHandle_t Display_Task
```

Definição na linha 52 do arquivo main.c.

2.1.4.3 dutyCycle

```
float dutyCycle = 0
```

Definição na linha 61 do arquivo main.c.

2.1.4.4 Filter_Task

```
TaskHandle_t Filter_Task
```

Definição na linha 50 do arquivo main.c.

2.1.4.5 filteredTemp

```
float filteredTemp = 0
```

Definição na linha 59 do arquivo main.c.

2.1.4.6 filteredTempQueue

```
QueueHandle_t filteredTempQueue
```

Definição na linha 55 do arquivo main.c.

2.1.4.7 hadc1

```
ADC_HandleTypeDef hadc1
```

Definição na linha 43 do arquivo main.c.

2.1.4.8 htim1

```
TIM_HandleTypeDef htim1
```

Definição na linha 45 do arquivo main.c.

2.1.4.9 huart1

```
UART_HandleTypeDef huart1
```

Definição na linha 47 do arquivo main.c.

2.1.4.10 ref

```
float ref = 0
```

Definição na linha 60 do arquivo main.c.

2.1.4.11 Temp_Task

```
TaskHandle_t Temp_Task
```

Definição na linha 49 do arquivo main.c.

2.1.4.12 tempFilter

```
FIRFilter tempFilter
```

Definição na linha 57 do arquivo main.c.

2.1.4.13 tempQueue

```
QueueHandle_t tempQueue
```

Definição na linha 54 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 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
```

Índice Remissivo