```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "stdbool.h"
/* USER CODE END Includes */

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

// Estado do Cursor
#define CURSOR_OFF 0x0C   // Apagado
#define CURSOR_ON 0x0E    // Ligado
#define CURSOR_BLINK 0x0F // Piscante

// Estado dos pinos de Controle...
#define RS_0 GPIOA -> BRR =  1<<9 //PA9
#define RS_1 GPIOA -> BSRR = 1<<9 //PA9
#define EN_0 GPIOC -> BRR =  1<<7 //PC7
#define EN_1 GPIOC -> BSRR = 1<<7 //PC7

// Estado dos pinos do Barramento do LCD...
#define D7_0 GPIOA -> BRR  = 1<<8 //PA8
#define D7_1 GPIOA -> BSRR = 1<<8 //PA8

#define D6_0 GPIOB -> BRR  = 1<<10 //PB10
#define D6_1 GPIOB -> BSRR = 1<<10 //PB10

#define D5_0 GPIOB -> BRR  = 1<<4 //PB4
#define D5_1 GPIOB -> BSRR = 1<<4 //PB4

#define D4_0 GPIOB -> BRR  = 1<<5 //PB5
#define D4_1 GPIOB -> BSRR = 1<<5 //PB5

//PARA O USO DA UART
#define NO_LCD 1
#define NA_SERIAL 2
/* USER CODE END PTD */

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

/* USER CODE END PD */

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

```

```
70  /* USER CODE END PM */
71
72  /* Private variables ---------------------------------------------------------*/
73  I2C_HandleTypeDef hi2c1;
74
75  RTC_HandleTypeDef hrtc;
76
77  TIM_HandleTypeDef htim1;
78
79  UART_HandleTypeDef huart2;
80
81  /* USER CODE BEGIN PV */
82
83  /* USER CODE END PV */
84
85  /* Private function prototypes -----------------------------------------------*/
86  void SystemClock_Config(void);
87  static void MX_GPIO_Init(void);
88  static void MX_USART2_UART_Init(void);
89  static void MX_I2C1_Init(void);
90  static void MX_RTC_Init(void);
91  static void MX_TIM1_Init(void);
92  /* USER CODE BEGIN PFP */
93  //-------FUNCOES PADRAO PARA O FUNCIONAMENTO DO LCD----------//
94  void udelay(void);
95  void delayus(int tempo);
96  void lcd_wrcom4 (uint8_t com4);
97  void lcd_wrcom(uint8_t com);
98  void lcd_wrchar(char ch);
99  void lcd_init(uint8_t cursor);
100 void lcd_wrstr(char *str);
101 void lcd_wr2dig(uint8_t valor);
102 void lcd_senddata(uint8_t data);
103 void lcd_clear(void);
104 void lcd_progchar(uint8_t n);
105 void lcd_goto(uint8_t x, uint8_t y);
106 int __io_putschar(int ch);
107 //----------------Delay-------------------------
108 void delay(uint16_t us);
109 //----------------Debug-------------------------
110 void debugI2c(void);
111 void Scan_I2C_Address(I2C_HandleTypeDef *hi2c);
112 void print_mem(void);
113 //----------------Basicas Memoria---------------------------
114 void clean_mem(void);
115 void save_in_mem(int *pos_in, char *str);
116 void indentificador_de_mem_principal(HAL_StatusTypeDef ret);
117 void le_uart(char *str);
118 int senha_alarme_func(void);// Recupera a senha pra desligar o alarme
119 //----------------Memoria-------------------------
120 void check_mem_load(void);
121 int save_logs(void);
122 int verify_acess(char *nome, char *senha);// Se precionado o botao, esige que a pessoa
    ponha seu nome e usuario pra liberar a entrada
123 int busca_nome_mp(int *pos, char *nome_mp);
124 void concede_acesso(void);
125
126 //Variavel de controle de maquina de estados
127 volatile int corrente = 1;
128
129 /* USER CODE END PFP */
130
131 /* Private user code ---------------------------------------------------------*/
132 /* USER CODE BEGIN 0 */
133 TIM_OC_InitTypeDef sConfig = {0};
134
135 int ret_error=0;
136 int erro = 0;
137
```

```c
char AONDE=NO_LCD;
char senha_mp[5] = {0};
char senha_alarme[5] = {0};
char nome_mp[5] = {0};

// ----------- Variaveis globais --------------------
int __io_putchar(int ch){
    if (AONDE == NO_LCD){
        if (ch != '\n') lcd_wrchar(ch);
    }
    if (AONDE == NA_SERIAL){
        HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, 100);
    }
    return ch;
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
    if((GPIO_Pin == GPIO_PIN_1)){
        delay(100);
        if((GPIOC->IDR & (1<<1))==0){corrente = 0;}// Pressiona Botao pra digitar log e senha e entrar
        if((GPIOC->IDR & (1<<1))!=0){corrente = 1;}// Des pressiona Botao
    }
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_I2C1_Init();
  MX_RTC_Init();
  MX_TIM1_Init();
  /* USER CODE BEGIN 2 */
  //-------------------------Inits-----------------------------------
    HAL_RTC_WaitForSynchro(&hrtc);
    HAL_TIM_Base_Init(&htim1);
    HAL_TIM_Base_Start(&htim1);
    HAL_UART_Init(&huart2);
    HAL_I2C_Init(&hi2c1);
    HAL_RTC_Init(&hrtc);
    lcd_init(CURSOR_OFF);

//  HAL_Delay(10);
    debugI2c();
```

```
206   //  print_mem();
207   //  delay(1000);
208   //  AONDE=NA_SERIAL;
209   //  printf("\rALARME RESIDENCIAL\r\n");
210   //  check_mem_load();
211       AONDE = NA_SERIAL;
212       print_mem();
213   //  printf("%s\r\n", senha_alarme);
214   //  print_mem();
215       Scan_I2C_Address(&hi2c1);
216
217     /* USER CODE END 2 */
218
219     /* Infinite loop */
220     /* USER CODE BEGIN WHILE */
221     while (1)
222     {
223       /* USER CODE END WHILE */
224
225       /* USER CODE BEGIN 3 */
226     }
227     /* USER CODE END 3 */
228   }
229
230   /**
231     * @brief System Clock Configuration
232     * @retval None
233     */
234   void SystemClock_Config(void)
235   {
236     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
237     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
238     RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
239
240     /** Initializes the RCC Oscillators according to the specified parameters
241     * in the RCC_OscInitTypeDef structure.
242     */
243     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_HSI48
244                                       |RCC_OSCILLATORTYPE_LSI;
245     RCC_OscInitStruct.HSIState = RCC_HSI_ON;
246     RCC_OscInitStruct.HSI48State = RCC_HSI48_ON;
247     RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
248     RCC_OscInitStruct.LSIState = RCC_LSI_ON;
249     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
250     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
251     {
252       Error_Handler();
253     }
254
255     /** Initializes the CPU, AHB and APB buses clocks
256     */
257     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
258                                   |RCC_CLOCKTYPE_PCLK1;
259     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI48;
260     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
261     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
262
263     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
264     {
265       Error_Handler();
266     }
267     PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART2|RCC_PERIPHCLK_I2C1
268                                         |RCC_PERIPHCLK_RTC;
269     PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
270     PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_HSI;
271     PeriphClkInit.RTCClockSelection = RCC_RTCCLKSOURCE_LSI;
272     if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
273     {
274       Error_Handler();
```

```
275     }
276   }
277
278   /**
279     * @brief I2C1 Initialization Function
280     * @param None
281     * @retval None
282     */
283   static void MX_I2C1_Init(void)
284   {
285
286     /* USER CODE BEGIN I2C1_Init 0 */
287
288     /* USER CODE END I2C1_Init 0 */
289
290     /* USER CODE BEGIN I2C1_Init 1 */
291
292     /* USER CODE END I2C1_Init 1 */
293     hi2c1.Instance = I2C1;
294     hi2c1.Init.Timing = 0x2000090E;
295     hi2c1.Init.OwnAddress1 = 0;
296     hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
297     hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
298     hi2c1.Init.OwnAddress2 = 0;
299     hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
300     hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
301     hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
302     if (HAL_I2C_Init(&hi2c1) != HAL_OK)
303     {
304       Error_Handler();
305     }
306
307     /** Configure Analogue filter
308     */
309     if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
310     {
311       Error_Handler();
312     }
313
314     /** Configure Digital filter
315     */
316     if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
317     {
318       Error_Handler();
319     }
320     /* USER CODE BEGIN I2C1_Init 2 */
321
322     /* USER CODE END I2C1_Init 2 */
323
324   }
325
326   /**
327     * @brief RTC Initialization Function
328     * @param None
329     * @retval None
330     */
331   static void MX_RTC_Init(void)
332   {
333
334     /* USER CODE BEGIN RTC_Init 0 */
335
336     /* USER CODE END RTC_Init 0 */
337
338     /* USER CODE BEGIN RTC_Init 1 */
339
340     /* USER CODE END RTC_Init 1 */
341
342     /** Initialize RTC Only
343     */
```

```
344     hrtc.Instance = RTC;
345     hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
346     hrtc.Init.AsynchPrediv = 127;
347     hrtc.Init.SynchPrediv = 255;
348     hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
349     hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
350     hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
351     if (HAL_RTC_Init(&hrtc) != HAL_OK)
352     {
353       Error_Handler();
354     }
355     /* USER CODE BEGIN RTC_Init 2 */
356
357     /* USER CODE END RTC_Init 2 */
358
359   }
360
361   /**
362     * @brief TIM1 Initialization Function
363     * @param None
364     * @retval None
365     */
366   static void MX_TIM1_Init(void)
367   {
368
369     /* USER CODE BEGIN TIM1_Init 0 */
370
371     /* USER CODE END TIM1_Init 0 */
372
373     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
374     TIM_MasterConfigTypeDef sMasterConfig = {0};
375
376     /* USER CODE BEGIN TIM1_Init 1 */
377
378     /* USER CODE END TIM1_Init 1 */
379     htim1.Instance = TIM1;
380     htim1.Init.Prescaler = 0;
381     htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
382     htim1.Init.Period = 65535;
383     htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
384     htim1.Init.RepetitionCounter = 0;
385     htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
386     if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
387     {
388       Error_Handler();
389     }
390     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
391     if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
392     {
393       Error_Handler();
394     }
395     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
396     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
397     if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
398     {
399       Error_Handler();
400     }
401     /* USER CODE BEGIN TIM1_Init 2 */
402
403     /* USER CODE END TIM1_Init 2 */
404
405   }
406
407   /**
408     * @brief USART2 Initialization Function
409     * @param None
410     * @retval None
411     */
412   static void MX_USART2_UART_Init(void)
```

```
413   {
414
415     /* USER CODE BEGIN USART2_Init 0 */
416
417     /* USER CODE END USART2_Init 0 */
418
419     /* USER CODE BEGIN USART2_Init 1 */
420
421     /* USER CODE END USART2_Init 1 */
422     huart2.Instance = USART2;
423     huart2.Init.BaudRate = 9600;
424     huart2.Init.WordLength = UART_WORDLENGTH_8B;
425     huart2.Init.StopBits = UART_STOPBITS_1;
426     huart2.Init.Parity = UART_PARITY_NONE;
427     huart2.Init.Mode = UART_MODE_TX_RX;
428     huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
429     huart2.Init.OverSampling = UART_OVERSAMPLING_16;
430     huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
431     huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
432     if (HAL_UART_Init(&huart2) != HAL_OK)
433     {
434       Error_Handler();
435     }
436     /* USER CODE BEGIN USART2_Init 2 */
437
438     /* USER CODE END USART2_Init 2 */
439
440   }
441
442   /**
443     * @brief GPIO Initialization Function
444     * @param None
445     * @retval None
446     */
447   static void MX_GPIO_Init(void)
448   {
449     GPIO_InitTypeDef GPIO_InitStruct = {0};
450   /* USER CODE BEGIN MX_GPIO_Init_1 */
451   /* USER CODE END MX_GPIO_Init_1 */
452
453     /* GPIO Ports Clock Enable */
454     __HAL_RCC_GPIOC_CLK_ENABLE();
455     __HAL_RCC_GPIOF_CLK_ENABLE();
456     __HAL_RCC_GPIOA_CLK_ENABLE();
457     __HAL_RCC_GPIOB_CLK_ENABLE();
458
459     /*Configure GPIO pin Output Level */
460     HAL_GPIO_WritePin(GPIOA, LD2_Pin|GPIO_PIN_8|GPIO_PIN_9, GPIO_PIN_RESET);
461
462     /*Configure GPIO pin Output Level */
463     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10|GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);
464
465     /*Configure GPIO pin Output Level */
466     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, GPIO_PIN_RESET);
467
468     /*Configure GPIO pins : B1_Pin PC1 */
469     GPIO_InitStruct.Pin = B1_Pin|GPIO_PIN_1;
470     GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
471     GPIO_InitStruct.Pull = GPIO_NOPULL;
472     HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
473
474     /*Configure GPIO pins : LD2_Pin PA8 PA9 */
475     GPIO_InitStruct.Pin = LD2_Pin|GPIO_PIN_8|GPIO_PIN_9;
476     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
477     GPIO_InitStruct.Pull = GPIO_NOPULL;
478     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
479     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
480
481     /*Configure GPIO pins : PB10 PB4 PB5 */
```

```
482     GPIO_InitStruct.Pin = GPIO_PIN_10|GPIO_PIN_4|GPIO_PIN_5;
483     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
484     GPIO_InitStruct.Pull = GPIO_NOPULL;
485     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
486     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
487
488     /*Configure GPIO pin : PC7 */
489     GPIO_InitStruct.Pin = GPIO_PIN_7;
490     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
491     GPIO_InitStruct.Pull = GPIO_NOPULL;
492     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
493     HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
494
495     /* EXTI interrupt init*/
496     HAL_NVIC_SetPriority(EXTI0_1_IRQn, 0, 0);
497     HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
498
499   /* USER CODE BEGIN MX_GPIO_Init_2 */
500   /* USER CODE END MX_GPIO_Init_2 */
501   }
502
503   /* USER CODE BEGIN 4 */
504   //---------------------------------------- FUNCAO
      DELAY-------------------------------------------------------------
505   void delay(uint16_t us){
506       uint16_t start = __HAL_TIM_GET_COUNTER(&htim1); // Lê o valor atual do contador
507       while ((__HAL_TIM_GET_COUNTER(&htim1) - start) < us); // Aguarda até que a diferença
          atinja 'us'
508   }
509
510   //---------------------------------------- FUNCOES DE DEBUG
      ------------------------------------------------------
511   void Scan_I2C_Address(I2C_HandleTypeDef *hi2c){
512       uint8_t address;
513       HAL_StatusTypeDef res;
514       printf("Iniciando a varredura do barramento I2C...\n");
515
516       for(address = 1; address < 128; address++)
517       {
518           res = HAL_I2C_IsDeviceReady(hi2c, (uint16_t)(address << 1), 10, 100);
519           if(res == HAL_OK)
520           {
521               printf("\rDispositivo encontrado no endereço I2C: 0x%X\n", address);
522               return;
523           }
524       }
525       printf("\rDispositivo não encontrado\n");
526
527   }
528
529   void debugI2c(void){/*
      DEBUG                                          */
530       char t = 0;
531       AONDE= NA_SERIAL;
532       HAL_StatusTypeDef ret;
533       for(uint8_t k=0; k<25; k++){
534           //t = t + 1;
535           ret =HAL_I2C_Mem_Write(&hi2c1, 0xA0, k, 1, (uint8_t*)&t, 1, 1000);
536           //HAL_Delay(100);
537           if ( ret != HAL_OK ) {
538               printf(" nao consegue trensferir\r\n");
539           }
540           else {
541               printf(" consegue trensferir\r\n");
542           }
543           //pw_in[k]= ch;
544           //printf("%c\r\n", t);
545           //printf("%c\r\n",(char)pw_in[k]);
546           //if((k+1)==4)pw_in[4]='\0';
```

```
547         }
548         t = 0;
549         printf("-----------------------------------------------------\r\n");
550         for(uint8_t k=0; k<26; k++){
551             ret = HAL_I2C_Mem_Read(&hi2c1, 0xA1, k, 1, (uint8_t*)&t, 1, 1000);
552             //HAL_Delay(1);
553             //pw_in[k]= ch;
554             if ( ret != HAL_OK ) {
555                 printf(" nao consegue ler\r\n");
556             }
557             else {
558                 printf(" consegue ler\r\n");
559             }
560             printf("%c\r\n", t);
561             //printf("%c\r\n",(char)pw_in[k]);
562             //if((k+1)==4)pw_in[4]='\0';
563         }
564         /*                                      DEBUG                                   */
565     }
566     void print_mem(void){
567         char t = 0;
568         HAL_StatusTypeDef ret;
569         uint8_t k=0;
570         AONDE = NA_SERIAL;
571         for(uint8_t k=0; k<33; k++){
572             ret = HAL_I2C_Mem_Read(&hi2c1, 0xA1, k, 1, (uint8_t*)&t, 1, 1000);
573     //      if ( ret != HAL_OK ) {
574     //          printf(" nao consegue ler\r\n");
575     //      }
576     //      else {
577     //          printf(" consegue ler\r\n");
578     //      }
579              if(ret == HAL_ERROR)   printf("|%d|%c,%dHAL_ERROR\n\r", k, t,(int)t);
580              if(ret == HAL_BUSY)    printf("|%d|%c,%dHAL_BUSY\n\r", k, t,(int)t);
581              if(ret ==HAL_TIMEOUT)  printf("|%d|%c,%dHAL_TIMEOUT\n\r", k, t,(int)t);
582              if(ret == HAL_OK) printf("|%d|%c,%dHAL_OK\n\r", k, t,(int)t);
583             //printf("%c\r\n",(char)pw_in[k]);
584             //if((k+1)==4)pw_in[4]='\0'; printf("|%d|%c %d", k, t,(int)t);
585         }
586     //  while(ret){
587     //      ret = HAL_I2C_Mem_Read(&hi2c1, 0xA1, k, 1, (uint8_t*)&t, 1, 1000);
588     //      HAL_Delay(1);
589     ////          if ( ret != HAL_OK ) {
590     ////              printf(" nao consegue ler\r\n");
591     ////          }
592     ////          else {
593     ////              printf(" consegue ler\r\n");
594     ////          }
595     //       if(ret == HAL_ERROR)   printf("|%d|%c,%d HAL_ERROR\n\r", k, t,(int)t);
596     //       if(ret == HAL_BUSY)    printf("|%d|%c,%d HAL_BUSY\n\r", k, t,(int)t);
597     //       if(ret ==HAL_TIMEOUT)  printf("|%d|%c,%d HAL_TIMEOUT\n\r", k, t,(int)t);
598     //       if(ret == HAL_OK) printf("|%d|%c,%dHAL_OK\n\r", k, t,(int)t);
599     //      //printf("%c\r\n",(char)pw_in[k]);
600     //      //if((k+1)==4)pw_in[4]='\0';
601     //       k+=1;
602     //  }
603     }
604     //------------------------------------------ FUNCOES BASICAS DE MEMORIA
        ------------------------------------------------------------
605     void clean_mem(void){
606         char t = 0;
607         AONDE= NA_SERIAL;
608         HAL_StatusTypeDef ret;
609         for(uint8_t k=0; k<33; k++){
610             //t = t + 1;
611             ret =HAL_I2C_Mem_Write(&hi2c1, 0xA0, k, 1, (uint8_t*)&t, 1, 1000);
612             HAL_Delay(10);
613             if ( ret != HAL_OK ) {
614                 //printf(" nao consegue trensferir\r\n");
```

```
615             }
616             else {
617                 //printf(" consegue trensferir\r\n");
618             }
619 
620         }
621         printf("-----------------------------------------------------\r\n");
622         for(uint8_t k=0; k<33; k++){
623             ret = HAL_I2C_Mem_Read(&hi2c1, 0xA1, k, 1, (uint8_t*)&t, 1, 1000);
624             //HAL_Delay(1);
625             //pw_in[k]= ch;
626             if ( ret != HAL_OK ) {
627                 //printf(" nao consegue ler\r\n");
628             }
629             else {
630                 //printf(" consegue ler\r\n");
631             }
632             printf("|%d|%c %d\n\r", k, t,(int)t);
633         }
634 
635     }
636 
637     void indentificador_de_mem_principal(HAL_StatusTypeDef ret){// Verifica se a memoria
        principal esta presente
638         if(ret != HAL_OK){
639             ret_error =1;
640         }
641     }
642     void le_uart(char *str){// Le a string digitada pelo usuario, com tamanho de 4 caracteres
643         char ch=0;
644         int i=0;
645         char str2[5];
646         for(i=0;i<4;i++){
647             do{
648                 erro = HAL_UART_Receive(&huart2, (uint8_t*)&ch, 1, 10);
649             } while (erro != HAL_UART_ERROR_NONE);
650             HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, 2);//ECO
651             str[i]=ch;
652             ch=0;
653         }
654         str[4]='\0';
655         strcpy(str, str2);
656         AONDE = NA_SERIAL;
657         printf("\r\n\r");
658     }
659     void save_in_mem(int *pos_in, char *str){// Salva na memoria principal
660         HAL_StatusTypeDef ret;
661         int pos= *pos_in;
662         int pos2 =pos;
663         AONDE=NA_SERIAL;
664         int ch;
665         printf("%d\n\r", pos);
666         for(int k=0; k<4; k++){// Salva na memoria
667             ch = str[k];
668             ret = HAL_I2C_Mem_Write(&hi2c1, 0xA0, pos, 1, (uint8_t*)&ch, 1, 1000);
669             HAL_Delay(100);
670             pos += 1;
671             printf("|%d|%c %d\n\r", k, str[k],(int)str[k]);
672         }
673         char confere[5]={0};
674         for(int k=0; k<4; k++){
675             ret = HAL_I2C_Mem_Read(&hi2c1, 0xA1, pos2, 1, (uint8_t*)&confere[k], 1, 1000);
676             HAL_Delay(100);
677             pos += 1;
678             if ( ret != HAL_OK ) {
679                 //printf(" nao consegue ler\r\n");
680             }
681             else {
682                 //printf(" consegue ler\r\n");
```

```
683             }
684             printf("|%d|%c %d\n\r", k, confere[k],confere[k]);
685         }
686 
687         *pos_in = pos;// Retorna a posicao atual da memoria principal
688     }
689     int senha_alarme_func(void){// Salva a senha do alarme da memória principal para a
    memória do programa
690         AONDE=NA_SERIAL;
691         char senha[5]={0};
692         int pos = 0x07;
693         HAL_StatusTypeDef ret;
694         for(int i = 0; i<4; i++){// Salva o nome presente no cartão!
695             ret = HAL_I2C_Mem_Read(&hi2c1, 0xA1, pos + i, 1, (uint8_t*)&senha[i], 1, 1000);
696             //HAL_Delay(100);
697     //      if (ret != HAL_OK ){
698     //          return 2;// Retorna pro display, a função setornara no código principal para
    a variável corrente
699     //      }
700         }
701         senha[4] = '\0';
702         strcpy(senha_alarme, senha);// 'senha_alarme' é uma variavel globar para salvar a
    senha para desligar o alarme
703         return 0; // Talvez mudar isso
704     }
705     //---------------------------------------- FUNCOES BASICAS DE MEMORIA
    ------------------------------------------------------
706     //---------------------------------------- FUNCOES DE
    MEMORIA------------------------------------------------------------
707     void check_mem_load(void){// Verifica se a memoria esta carregada ou vazia
708         uint8_t CH=0;
709         AONDE=NA_SERIAL;
710         HAL_StatusTypeDef ret;
711         HAL_Delay(1);
712         ret = HAL_I2C_Mem_Read(&hi2c1, 0xA1, 1, 1, (uint8_t*)&CH, 1, 100);
713         if( ret!= HAL_OK ){
714             printf(" Memoria comprometida\r\n");
715                 return;
716         }
717         else{
718             if(CH == 0x1B){
719                 printf("Memoria carregada\r\n");
720             }
721             else{
722                 printf("Memoria vazia\r\n");
723                 clean_mem();
724                 save_logs();
725             }
726             senha_alarme_func();
727         }
728     }
729     int save_logs(void){// Salva os usuários e suas senhas na memória principal, com uma
    estrutura de partição
730         AONDE=NA_SERIAL;
731         HAL_StatusTypeDef ret;
732       /*//------------ Com o formato --------\\
733         // |-Pos-| ---- Dados da Particao ----|
734         // | *0* | (0x1B-Indica se esta cheia)|
735         // | *1* | (--0xA0-Indica o nome 0--) |
736         // |02-05| (---Nome--4 Bytes--------) |
737         // | *6* | (--0xB0-Indica a senha 0-) |
738         // |07-10| (--Senha--4 Bytes--------) |
739         // | 11* | (--0xA1-Indica o nome 1--) |
740         // |12-15| (---Nome--4 Bytes--------) |
741         // | 16* | (--0xB1-Indica a senha 1-) |
742         // |17-20| (--Senha--4 Bytes--------) |
743         // | 21* | (--0xA2-Indica o nome 2--) |
744         // |22-25| (---Nome--4 Bytes--------) |
745         // | 26* | (--0xB2-Indica a senha 2-) |
```

```
746         // |27-30| (--Senha--4 Bytes--------) |
747         // ..................................|*/
748         char nome[5]={0};
749         char senhax[5]={0};
750         int pos=2;
751         AONDE = NA_SERIAL;
752         uint8_t particao = 0xA0; // O primeiro se refere a se é nome(A) ou senha(B) e o
            segundo se refere user sendo '0' o usuario coringa;
753         ret_error = 0;
754         HAL_Delay(100);
755         ////////////////////////////////////////////////////////////////////////////////
756         printf("Escrava o nome do Usuario Coringa Com 4 Letras\r\n");
757         memset(nome, 0, sizeof(nome));// Limpa a variavel
758         le_uart(nome);// Le a string digitada pelo usuario, com tamanho de 4 caracteres
759         particao = 0xA0;// Indica que o nome 0 esta a seguir
760         ret = HAL_I2C_Mem_Write(&hi2c1, 0xA0, pos, 1, (uint8_t*)&particao, 1, 1000);
761         HAL_Delay(100);
762         pos+=1;
763         save_in_mem(&pos, nome);// Salva na memoria principal a parrticao 0xA0
764         indentificador_de_mem_principal(ret);// Verifica se a memoria principal esta presente
765         /*-------------------------------------------------------------------------------*/
766         printf("Escrava o senha alarme Com 4 Letras, para desativar os alarmes\r\n");
767         memset(senhax, 0, sizeof(senhax));// Limpa a variavel
768         le_uart(senhax);// Le a string digitada pelo usuario, com tamanho de 4, mas somente
            aceita numeros e ignora outros chars
769         particao = 0xB0;// Indica que a senha 0 esta a seguir
770         ret = HAL_I2C_Mem_Write(&hi2c1, 0xA0, pos, 1, (uint8_t*)&particao, 1, 1000);
771         HAL_Delay(100);
772         pos+=1;
773         save_in_mem(&pos, senhax);// Salva na memoria principal a parrticao 0xB0
774         indentificador_de_mem_principal(ret);// Verifica se a memoria principal esta presente
775         ////////////////////////////////////////////////////////////////////////////////
776
777
778         return 0;
779     }
780     int verify_acess(char *nome, char *senha){
781         AONDE=NA_SERIAL;
782         HAL_StatusTypeDef ret;
783         ////////////////////////////////////////////////////////////////////
784         char nome_user[5]={0};// Variavel para salvar o nome lido do teclado
785         char senha_userr[5]={0};
786         AONDE=NA_SERIAL;
787         printf("Digite o nome de usuario com 4 characteres \r\n");
788         le_uart(nome_user);//PEDE AO USER O NOME E SENHA
789         printf("Digite a senha com 4 characteres \r\n");
790         le_uart(senha_userr);
791         ////////////////////////////////////////////////////////////////////
792         //Agora vai tentar procurar esse nome na memoria principal
793         char nome_mp[5]={0};
794         int pos = 0;
795         while(1){
796             memset(nome_mp, 0, sizeof(nome_mp));
797             busca_nome_mp(&pos,nome_mp);// Busca o nome na memoria principal
798             if((strcmp(nome_mp, nome_user)==0)){// Se o nome lido do cartao for igual ao
                nome que esta na memoria principal
799                 printf("Nome encontrado\r\n");
800                 ////////////////////////////////////////////////////////////////////
801                 //Agora vai salvar a senha principal na variavel senha_mp
802                 printf("Verifincado senha\r\n");
803
804                 char senha[5]={0};
805                 for(int i = 0; i<4; i++){
806                     ret = HAL_I2C_Mem_Read(&hi2c1, 0xA1, pos +i, 1, (uint8_t*)&senha[i], 1,
                    1000);
807                     if (ret != HAL_OK ){
808                         printf(" Erro em ler a senha, tente  mais tarde\r\n");
809                         return 2;// Retorna pro display, a função setornara no código
                        principal para a variável corrente
```

```
                }
            }
            senha[4] = '\0';// Senha recuuperada da memoria principal com sucesso
            if((strcmp(senha_userr, senha))==0){
                    AONDE = NA_SERIAL;
                    printf("\rEntrada user %s\r\n", nome_mp);
                    //catraca();
                    lcd_clear();
                    AONDE = NO_LCD;
                    lcd_goto(0,0);
                    printf("Entrada user\n");
                    lcd_goto(1,0);
                    printf("%s\n", nome_mp);
                    HAL_Delay(1500);
                }
                else{
                    AONDE = NA_SERIAL;
                    printf("\rAcesso negado!\r\n");
                    lcd_clear();
                    AONDE = NO_LCD;
                    lcd_goto(0,0);
                    printf("Acesso negado!\n");
                    HAL_Delay(1000);
                }
                lcd_clear();
                corrente = 2;

            // 'senha' é uma variavel para salvar a senha referente ao nome que estava na mp,
            // para depois conferir com a senha que o user digitou para liberer a catraca
        }
        ////////////////////////////////////////////////////////////////////////
        else printf("Procurando\r\n");
        if( pos > 18 ){//procurrar ate a posicao 18, no caso ele vai somar mais 5 e vai dar 23
            printf("Nome não encontrado na Memoria principal\r\n");
            return 2;
        }
    }
}
int busca_nome_mp(int *pos_out, char *nome_mp){// Busca o nome na memoria principal
    AONDE=NA_SERIAL;
    uint8_t ch;
    char nome[5]={0};
    int pos = *pos_out;
    HAL_StatusTypeDef ret;
    do{
        HAL_I2C_Mem_Read(&hi2c1, 0xA1, pos , 1, (uint8_t*)&ch, 1, 1000);
        pos = pos +1;
    }while(ch!=0xA0||ch!=0xA1||ch!=0xA2);
    if(ch==0xA0||ch==0xA1||ch==0xA2){
        for(int i = 0; i<4; i++){// Salva o nome presente no cartão!
            ret = HAL_I2C_Mem_Read(&hi2c1, 0xA1, pos +i, 1, (uint8_t*)&nome[i], 1, 1000);
            //HAL_Delay(100);
            if (ret != HAL_OK ){
                printf(" Erro em ler o nome, retire o cartao\r\n");
                return 2;// Retorna pro display, a função setornara no código principal para a variável corrente
            }
        }
        nome[4] = '\0';
    }
    *pos_out = pos + 5;// Soma 5 para ir para a proxima posicao
    strcpy(nome_mp, nome);// Salva o nome na variavel global
    return 5;// Sera decidido depois
}
//--------------------------------LCD----------------------------
void lcd_backlight (uint8_t light){
    if(light == 0){
```

```
        GPIOB -> BRR = 1<<3;
    } else {
        GPIOB -> BSRR= 1<<3;
    }
}
void lcd_init(uint8_t cursor){
    lcd_wrcom4(3);
    lcd_wrcom4(3);
    lcd_wrcom4(3);
    lcd_wrcom4(2);
    lcd_wrcom(0x28);
    lcd_wrcom(cursor);
    lcd_wrcom(0x06);
    lcd_wrcom(0x01);
}
void lcd_wrcom4(uint8_t com4){
    lcd_senddata(com4); //D4...d0
    RS_0;
    EN_1;
    delayus(5);
    EN_0;
    HAL_Delay(5);
}
void lcd_wrcom(uint8_t com){
    lcd_senddata(com>>4); //0000D7...D4
    RS_0;
    EN_1;
    delayus(5);
    EN_0;
    delayus(5);

    lcd_senddata(com & 0x0F); //0000D3...d0
    EN_1;
    delayus(5);
    EN_0;
    HAL_Delay(5);
}
void lcd_clear(void){
    lcd_wrcom(0x01);
}
//goto para 16x2
void lcd_goto(uint8_t x, uint8_t y){
    uint8_t com = 0x80;
    if (x==0 && y<16) com = 0x80 + y;
    else if (x==1 && y<16) com = 0xC0 + y;
    else com = 0x80;
    lcd_wrcom(com);
}
void lcd_wrchar(char ch){
    lcd_senddata(ch>>4); //D7...D4
    RS_1;
    EN_1;
    delayus(5);
    EN_0;
    delayus(5);

    lcd_senddata(ch & 0x0F); //D3...D0
    RS_1;
    EN_1;
    delayus(5);
    EN_0;
    HAL_Delay(5);
}

void lcd_wrstr(char *str){
    while(*str) lcd_wrchar(*str++);
}
```

```
945  void udelay(void){
946      int tempo = 7;
947      while(tempo--);
948  }
949
950  void delayus(int tempo){
951      while(tempo--) udelay();
952  }
953
954  void lcd_wr2dig(uint8_t valor){
955      lcd_wrchar(valor/10 + '0'); // ou +48 -> dezena
956      lcd_wrchar(valor%10 + '0'); // ou +48 -> unidade
957  }
958
959  void lcd_senddata(uint8_t data){
960      if((data & (1<<3))==0) D7_0; else D7_1;
961      if((data & (1<<2))==0) D6_0; else D6_1;
962      if((data & (1<<1))==0) D5_0; else D5_1;
963      if((data & (1<<0))==0) D4_0; else D4_1;
964  }
965  //-----------------------------------------------------------------
966  /* USER CODE END 4 */
967
968  /**
969    * @brief  This function is executed in case of error occurrence.
970    * @retval None
971    */
972  void Error_Handler(void)
973  {
974    /* USER CODE BEGIN Error_Handler_Debug */
975    /* User can add his own implementation to report the HAL error return state */
976    __disable_irq();
977    while (1)
978    {
979    }
980    /* USER CODE END Error_Handler_Debug */
981  }
982
983  #ifdef  USE_FULL_ASSERT
984  /**
985    * @brief  Reports the name of the source file and the source line number
986    *         where the assert_param error has occurred.
987    * @param  file: pointer to the source file name
988    * @param  line: assert_param error line source number
989    * @retval None
990    */
991  void assert_failed(uint8_t *file, uint32_t line)
992  {
993    /* USER CODE BEGIN 6 */
994    /* User can add his own implementation to report the file name and line number,
995       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
996    /* USER CODE END 6 */
997  }
998  #endif /* USE_FULL_ASSERT */
999
```