

**EE242-MICROPROCESSOR SYSTEMS**

**Project Report**

**Implementation of Alarm Clock by Using Stm32F103**

by

Mustafa AKSOY

20160701106



Faculty of Engineering

Department of Electrical and Electronics Engineering

2020 Fall

# 1 .PURPOSE OF THE PROJECT

The aim of this project is to make an alarm clock with a stm32f407vg microprocessor. The stm32f407vg software language is C. An LCD screen is used to display the time. The buttons are used to adjust the time. Buzzer is used to make a sound when the time comes.

## 2. METHODOLOGY

### 2.1 Harware System

In this project, stm32f407vg microprocessor, 16x2 LCD Display, buzzer and button are used.

#### 2.1.1 STM32F103 Microprocessor

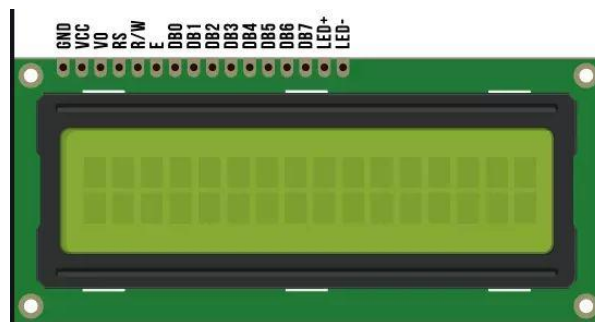
The STM32F103xx medium-density performance line family incorporates the high-performance ARM<sup>®</sup>Cortex<sup>®</sup>-M3 32-bit RISC core operating at a 72 MHz frequency, high-speed embedded memories (Flash memory up to 128 Kbytes and SRAM up to 20 Kbytes), and an extensive range of enhanced I/Os and peripherals connected to two APB buses. All devices offer two 12-bit ADCs, three general purpose 16-bit timers plus one PWM timer, as well as standard and advanced communication interfaces: up to two I<sup>2</sup>Cs and SPIs, three USARTs, an USB and a CAN.



**Figure 2. 1:** STM32F103

### 2.1.2 16x2 LCD Screen

It has white writing color on a blue background. It works with 5V. It does not need any other power source. Compatible with development boards. There are 16 pins on it. The number of pins can be reduced by using a converter. This project is also used to show the time and the alarm.



**Figure 2. 2:** 16x2 LCD Screen

### 2.1.3 Buzzer

Buzzer; It is a type of auditory warning device that works based on mechanical, electromechanical or piezoelectric principles. Buzzers, which have a lot of usage areas, generally work with the piezoelectric principle. Buzzers can be used in functions such as alarm, timer, confirmation response alert, depending on their area of use. As a matter of fact, as mentioned in the definition, buzzers are types of auditory warning devices. They have types such as illuminated buzzer, non-light buzzer, passive buzzer and active buzzer. Buzzer devices have two legs in total. One of the legs is (+) and the other is (-) leg. Therefore, attention should be paid to the legs of the buzzer when connecting the buzzer. Buzzers usually have (+) or (-) indicators on the legs to help you distinguish their legs. In this project, buzzer was used to wake up.



**Figure 2.3 : Buzzer**

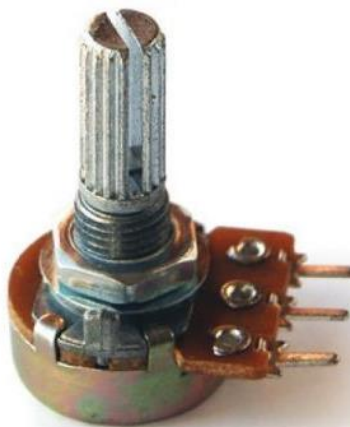
#### **2.1.4 Button**

It is available with 4 legs or 2 legs. It is powered by 5V. It allows us to keep it constant while push up and pull down logic is at 0. In this project, the hour, minute, second changes as the button is pressed. That's why 3 were used.



**Figure 2. 4 : Button**

#### **2.1.5 Potansiyometre**



## 2.2 Code

```
1  /* USER CODE BEGIN Header */
2  /**
3   * @file
4   * @brief : Main program body
5   * @attention
6   *
7   * <h2><center>copy; Copyright (c) 2021 STMicroelectronics.
8   * All rights reserved.</center></h2>
9   *
10  * This software component is licensed by ST under BSD 3-Clause license,
11  * the "License"; You may not use this file except in compliance with the
12  * License. You may obtain a copy of the License at:
13  *
14  * opensource.org/licenses/BSD-3-Clause
15  *
16  */
17  /* USER CODE END Header */
18  /* Includes -----*/
19  #include "main.h"
20  #include "stm32f10x.h"
21  /* Private includes -----*/
22  /* USER CODE BEGIN Includes */
23  /* USER CODE END Includes */
24  /* Private typedef -----*/
25  /* USER CODE BEGIN PTD */
26  /* USER CODE END PTD */
27  /* Private define -----*/
28  /* USER CODE BEGIN PD */
29  /* USER CODE END PD */
30  /* Private macro -----*/
31  /* USER CODE BEGIN PM */
32
33  char i1kstatr[16]=" ";
34  char i1kstatr[16]=" ";
35  #include "time.h"
36  /* USER CODE END PM */
37
38  /* Private macro -----*/
39  /* USER CODE BEGIN PM */
40  /* USER CODE END PM */
41
42  /* Private variables -----*/
43  RTC_HandleTypeDef hrtc;
44  TIM_HandleTypeDef htimg;
45
46  /* USER CODE BEGIN PV */
47  uint8_t Hour;
48  uint8_t Sec=0;
49  uint8_t Min;
50  uint8_t Year;
51  uint8_t Month;
52  uint8_t Date;
53  uint8_t selection;
54  uint8_t select_som;
55  RTC_HandleTypeDef hrtc;
56  RTC_DateTypeDef RTC_Date;
57
58  /* USER CODE END PV */
59
60  /* Private function prototypes -----*/
61  void SystemClock_Config(void);
62  static void MX_GPIO_Init(void);
63  static void MX_RTC_Init(void);
64  static void MX_TIM1_Init(void);
65  /* USER CODE BEGIN PFP */
66  /* USER CODE END PFP */
67
68  #include "stdio.h"
69  #include "LCD.h"
70
71  /* USER CODE END PFP */
72  /* Private user code -----*/
```

```
main.c
85 /* USER CODE END FFP */
86
87 /* Private user code -----*/
88 /* USER CODE BEGIN 0 */
89
90
91
92
93
94 /* USER CODE END 0 */
95
96 /**
97  * @brief The application entry point.
98  * @retval int
99  */
100 int main(void)
101 {
102     /* USER CODE BEGIN 1 */
103
104     int second = 0, minute = 0, hour = 0, ahour = 0, amin = 0;
105
106     int currentHour=0;
107     int currentMin=0;
108     int screenHour=0;
109     int screenMin=0;
110     int alarmHour=0;
111     int alarmMin=0;
112     bool isChanged=false;
113     bool isAlarmBuzed=false;
114
115
116     int hour2=0;
117     int min2=0;
118
119     GPIO_PinState mypush;
120
121
122
123     /* USER CODE END 1 */
124
125     /* MCU Configuration-----*/
126
127     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
128     HAL_Init();
129
```

```
main.c
127 /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
128 HAL_Init();
129
130 /* USER CODE BEGIN Init */
131
132 /* USER CODE END Init */
133
134 /* Configure the system clock */
135 SystemClock_Config();
136
137 /* USER CODE BEGIN SysInit */
138
139 /* USER CODE END SysInit */
140
141 /* Initialize all configured peripherals */
142 MX_GPIO_Init();
143 MX_RTC_Init();
144 MX_TIM1_Init();
145 /* USER CODE BEGIN 2 */
146 lcd_init(_LCD_4BIT, _LCD_FONT_5x8, _LCD_2LINE);
147
148 HAL_RTC_GetTime(&rtc, &RTC_Time, RTC_FORMAT_BIN);
149
150
151
152
153
154
155 /*
156  * If (HAL_GPIO_ReadPin(GPIOB,button_Pin)==1 && HAL_GPIO_ReadPin(GPIOB,button1_Pin)==1)
157  * {
158  *     ahour=Hour;
159  *     amin= Min;
160  *
161  *     sprintf(ikksatir,"Guncel Alarm" );
162  *     lcd_print(1,1,ikksatir);
163  *     sprintf(ikksatir,"%02d:%02d:%02d", ahour, amin, Sec);
164  *     lcd_print(2,1,ikksatir);
165  *     HAL_Delay(5000);
166  * }
167  */
168
169
170
171
```

```
main.c
172 */
173
174
175
176
177 /* USER CODE END 2 */
178
179 /* Infinite loop */
180 /* USER CODE BEGIN WHILE */
181
182
183
184
185
186 while (1)
187 {
188     /* USER CODE END WHILE */
189
190     /* USER CODE BEGIN 3 */
191
192
193     HAL_RTC_GetTime(&rtc, &RTC_Time, RTC_FORMAT_BIN);
194     HAL_RTC_GetDate(&rtc, &RTC_Date, RTC_FORMAT_BIN);
195
196
197     currentHour=RTC_Time.Hours;
198     currentMin=RTC_Time.Minutes;
199     if(!isChanged){
200         screenHour=currentHour;
201         screenMin=currentMin;
202     }
203
204
205
206     Hour = RTC_Time.Hours + hour2;
207     Min = RTC_Time.Minutes + min2;
208     Sec = RTC_Time.Seconds;
209
210     Date = RTC_Date.Date;
211     Month = RTC_Date.Month;
212     Year = RTC_Date.Year;
213
214
215     sprintf(ikksatir,"Mustafa Aksoy" );
216     sprintf(ikksatir,"%02d:%02d:%02d", currentHour, currentMin, Sec);
217
```

```

main.c
214
215 sprintf(ikksatir,"Mustafa Aksoy");
216 sprintf(ikksatir,"%02d:02d:02d", currentHour, currentMin, Sec);
217
218 lcd_print(1,1,ikksatir);
219 lcd_print(2,1,ikksatir);
220 HAL_Delay(200);
221
222 lcd_clear();
223
224
225
226 //alarm
227
228
229 if (HAL_GPIO_ReadPin(GPIOB,button_Pin)==1 && HAL_GPIO_ReadPin(GPIOB,button1_Pin)==1)
230 {
231
232
233 alarmHour=screenHour;
234 alarmMin=screenMin;
235 isAlarmBuzed=false;
236
237
238
239
240 ahour=hour2 ;
241 amin=min2 ;
242
243
244
245 sprintf(ikksatir,"Guncel Alarm");
246 lcd_print(1,1,ikksatir);
247 sprintf(ikksatir,"%02d:02d", alarmHour, alarmMin);
248 lcd_print(2,1,ikksatir);
249 HAL_Delay(5000);
250
251
252 }
253
254
255
256 //alarm buzzer
257 /*
258

```

```

main.c
256
257 //alarm buzzer
258 /*
259 if (ahour==Hour && amin==Min)
260 {
261
262
263 sprintf(ikksatir,"Alarm");
264 lcd_print(1,1,ikksatir);
265 lcd_clear();
266
267 HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_SET);
268 HAL_Delay(500);
269 HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_RESET);
270
271 sprintf(ikksatir,"Alarm");
272 lcd_print(1,1,ikksatir);
273 lcd_clear();
274
275 HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_SET);
276 HAL_Delay(500);
277 HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_RESET);
278
279 sprintf(ikksatir,"Alarm");
280 lcd_print(1,1,ikksatir);
281 lcd_clear();
282
283 HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_SET);
284 HAL_Delay(500);
285 HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_RESET);
286
287 sprintf(ikksatir,"Alarm");
288 lcd_print(1,1,ikksatir);
289 lcd_clear();
290
291 HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_SET);
292 HAL_Delay(500);
293 HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_RESET);
294
295 sprintf(ikksatir,"Alarm");
296 lcd_print(1,1,ikksatir);
297 lcd_clear();
298
299
300

```

```

main.c
301
302
303 */
304 //hour++
305 if (HAL_GPIO_ReadPin(GPIOB,button_Pin)==1)
306 {
307 HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_SET);
308 HAL_Delay(500);
309 HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_RESET);
310
311
312 if (screenHour==23) {
313 screenHour=0;
314 }
315 else {
316 screenHour++;
317 //Hour++
318 }
319 isChanged=true;
320 lcd_clear();
321 sprintf(ikksatir,"%02d: %02d: %02d", screenHour, screenMin, screenSec);
322 lcd_print(1,1,ikksatir);
323 HAL_Delay(3000);
324
325
326
327
328
329
330
331 //hour -
332
333
334 if (HAL_GPIO_ReadPin(GPIOB,button1_Pin)==1)
335 {
336 HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_SET);
337 HAL_Delay(500);
338 HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_RESET);
339
340
341 if (screenHour==0) {
342 screenHour=23;
343 }
344 else {
345 screenHour--;
346

```

```

main.c
343 }
344 else{
345     screenHour--;
346     //Hour--;
347 }
348 }
349 isChanged=true;
350 lcd_clear();
351 sprintf(ikisatir,"%02d: hour -degist1", screenHour );
352 lcd_print(2,1,ikisatir);
353 HAL_Delay(3000);
354 }
355 }
356 }
357 }
358 //min ++
359 }
360 }
361 }
362 if(HAL_GPIO_ReadPin(GPIOB,button2_Pin)==1)
363 {
364     HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_SET);
365     HAL_Delay(500);
366     HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_RESET);
367 }
368 }
369 }
370 if(screenMin==59){
371     screenMin=0;
372 }
373 else{
374     screenMin++;
375     //Hour--;
376 }
377 }
378 }
379 isChanged=true;
380 lcd_clear();
381 sprintf(ikisatir,"%02d: min +degist1", screenMin);
382 lcd_print(2,1,ikisatir);
383 HAL_Delay(3000);
384 }
385 }
386 //min--
387 }

```

```

main.c
385 }
386 //min--
387 }
388 }
389 }
390 if(HAL_GPIO_ReadPin(GPIOB,button3_Pin)==1)
391 {
392     HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_SET);
393     HAL_Delay(500);
394     HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_RESET);
395 }
396 }
397 if(screenMin==0){
398     screenMin=59;
399 }
400 else{
401     screenMin--;
402     //Min--;
403 }
404 }
405 isChanged=true;
406 lcd_clear();
407 sprintf(ikisatir,"%02d: min -degist1", screenMin );
408 lcd_print(2,1,ikisatir);
409 HAL_Delay(3000);
410 }
411 }
412 }
413 }
414 //
415 }
416 }
417 }
418 }
419 }
420 if(currentHour==alarmHour && currentMin==alarmMin)
421 {
422     if(!isAlarmBuzzed){
423         sprintf(ikisatir,"Alarm ");
424         lcd_print(1,1,ikisatir);
425         lcd_clear();
426     }
427     HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_SET);
428     HAL_Delay(2000);
429     HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_RESET);

```

```

main.c
427 HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_SET);
428 HAL_Delay(2000);
429 HAL_GPIO_WritePin(buzzer_GPIO_Port,buzzer_Pin,GPIO_PIN_RESET);
430 isAlarmBuzzed=true;
431 isChanged=false;
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 /* USER CODE END 3 */
440 }
441 }
442 /**
443  * Brief System Clock Configuration
444  * @retval None
445  */
446 void SystemClock_Config(void)
447 {
448     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
449     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
450     RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
451 }
452 /**
453  * Initializes the RCC Oscillators according to the specified parameters
454  * in the RCC_OscInitTypeDef structure.
455  */
456 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE|RCC_OSCILLATORTYPE_LSE;
457 RCC_OscInitStruct.HSEState = RCC_HSE_ON;
458 RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
459 RCC_OscInitStruct.LSEState = RCC_LSE_ON;
460 RCC_OscInitStruct.HSIState = RCC_HSI_ON;
461 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
462 RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
463 RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL2;
464 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
465 {
466     Error_Handler();
467 }
468 /**
469  * Initializes the CPU, AHB and APB buses clocks
470  */
471 RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYCLK
472 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
473 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;

```



```

main.c
469 RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
470 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
471 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
472 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
473 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
474 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
475
476 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
477 {
478     Error_Handler();
479 }
480 PeriphCkInit.PeriphClockSelection = RCC_PERIPHCLK_RTC;
481 PeriphCkInit.RTCClockSelection = RCC_RTCCLKSOURCE_LSE;
482 if (HAL_RCCEx_PeriphCLKConfig(&PeriphCkInit) != HAL_OK)
483 {
484     Error_Handler();
485 }
486 }
487
488 /**
489  * Brief RTC Initialization Function
490  * @param None
491  * @retval None
492  */
493 static void MX_RTC_Init(void)
494 {
495     /* USER CODE BEGIN RTC_Init 0 */
496
497     /* USER CODE END RTC_Init 0 */
498
499     RTC_TimeTypeDef sTime = {0};
500     RTC_DateTypeDef DateToUpdate = {0};
501     RTC_AlarmTypeDef sAlarm = {0};
502
503     /* USER CODE BEGIN RTC_Init 1 */
504
505     /* USER CODE END RTC_Init 1 */
506
507     /* Initialize RTC Only
508     */
509     hrtc.Instance = RTC;
510     hrtc.Init.AsynchPresdiv = RTC_AUTO_1_SECOND;
511     hrtc.Init.OutPut = RTC_OUTPUTSOURCE_ALARM;
512     if (HAL_RTC_Init(&hrtc) != HAL_OK)
513     {

```

```

main.c
514     Error_Handler();
515     }
516
517     /* USER CODE BEGIN Check_RTC_BKUP */
518
519     /* USER CODE END Check_RTC_BKUP */
520
521     /* Initialize RTC and set the Time and Date
522     */
523     sTime.Hours = 22;
524     sTime.Minutes = 10;
525     sTime.Seconds = 45;
526
527     if (HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BIN) != HAL_OK)
528     {
529         Error_Handler();
530     }
531     DateToUpdate.WeekDay = RTC_WEEKDAY_SATURDAY;
532     DateToUpdate.Month = RTC_MONTH_JANUARY;
533     DateToUpdate.Date = 16;
534     DateToUpdate.Year = 21;
535
536     if (HAL_RTC_SetDate(&hrtc, &DateToUpdate, RTC_FORMAT_BIN) != HAL_OK)
537     {
538         Error_Handler();
539     }
540     /* Enable the Alarm A
541     */
542     sAlarm.AlarmTime.Hours = 0;
543     sAlarm.AlarmTime.Minutes = 0;
544     sAlarm.AlarmTime.Seconds = 0;
545     sAlarm.Alarm = RTC_ALARM_A;
546     if (HAL_RTC_SetAlarm_IT(&hrtc, &sAlarm, RTC_FORMAT_BIN) != HAL_OK)
547     {
548         Error_Handler();
549     }
550     /* USER CODE BEGIN RTC_Init 2 */
551
552     /* USER CODE END RTC_Init 2 */
553
554 }
555
556 /**
557  * Brief TIM1 Initialization Function
558  * @param None
559  * @retval None
560  */
561 static void MX_TIM1_Init(void)
562 {
563     /* USER CODE BEGIN TIM1_Init 0 */
564
565     /* USER CODE END TIM1_Init 0 */
566
567     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
568     TIM_MasterConfigTypeDef sMasterConfig = {0};
569
570     /* USER CODE BEGIN TIM1_Init 1 */
571
572     /* USER CODE END TIM1_Init 1 */
573     htim1.Instance = TIM1;
574     htim1.Init.Prescaler = 0;
575     htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
576     htim1.Init.Period = 65535;
577     htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
578     htim1.Init.RepetitionCounter = 0;
579     htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
580     if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
581     {
582         Error_Handler();
583     }
584
585     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
586     if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
587     {
588         Error_Handler();
589     }
590     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
591     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
592     if (HAL_TIMEX_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
593     {
594         Error_Handler();
595     }
596     /* USER CODE BEGIN TIM1_Init 2 */
597
598     /* USER CODE END TIM1_Init 2 */
599
600 }

```

```

main.c
556 /**
557  * Brief TIM1 Initialization Function
558  * @param None
559  * @retval None
560  */
561 static void MX_TIM1_Init(void)
562 {
563     /* USER CODE BEGIN TIM1_Init 0 */
564
565     /* USER CODE END TIM1_Init 0 */
566
567     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
568     TIM_MasterConfigTypeDef sMasterConfig = {0};
569
570     /* USER CODE BEGIN TIM1_Init 1 */
571
572     /* USER CODE END TIM1_Init 1 */
573     htim1.Instance = TIM1;
574     htim1.Init.Prescaler = 0;
575     htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
576     htim1.Init.Period = 65535;
577     htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
578     htim1.Init.RepetitionCounter = 0;
579     htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
580     if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
581     {
582         Error_Handler();
583     }
584
585     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
586     if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
587     {
588         Error_Handler();
589     }
590     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
591     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
592     if (HAL_TIMEX_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
593     {
594         Error_Handler();
595     }
596     /* USER CODE BEGIN TIM1_Init 2 */
597
598     /* USER CODE END TIM1_Init 2 */
599
600 }

```

```

main.c
601
602 /**
603  * @brief GPIO Initialization Function
604  * @param None
605  * @retval None
606  */
607 static void MX_GPIO_Init(void)
608 {
609     GPIO_InitTypeDef GPIO_InitStruct = {0};
610
611     /* GPIO Ports Clock Enable */
612     __HAL_RCC_GPIOC_CLK_ENABLE();
613     __HAL_RCC_GPIOB_CLK_ENABLE();
614     __HAL_RCC_GPIOA_CLK_ENABLE();
615     __HAL_RCC_GPIOH_CLK_ENABLE();
616
617     /*Configure GPIO pin Output Level */
618     HAL_GPIO_WritePin(GPIOA, LCD_EN_Pin|LCD_RS_Pin|LCD_D4_Pin|LCD_D5_Pin
619                       |LCD_D6_Pin|LCD_D7_Pin, GPIO_PIN_RESET);
620
621     /*Configure GPIO pin Output Level */
622     HAL_GPIO_WritePin(buzzer_GPIO_Port, buzzer_Pin, GPIO_PIN_RESET);
623
624     /*Configure GPIO pins : LCD_EN_Pin LCD_RS_Pin LCD_D4_Pin LCD_D5_Pin
625                        LCD_D6_Pin LCD_D7_Pin */
626     GPIO_InitStruct.Pin = LCD_EN_Pin|LCD_RS_Pin|LCD_D4_Pin|LCD_D5_Pin
627                       |LCD_D6_Pin|LCD_D7_Pin;
628     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
629     GPIO_InitStruct.Pull = GPIO_NOPULL;
630     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
631     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
632
633     /*Configure GPIO pin : button_Pin */
634     GPIO_InitStruct.Pin = button_Pin;
635     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
636     GPIO_InitStruct.Pull = GPIO_PULLDOWN;
637     HAL_GPIO_Init(button_GPIO_Port, &GPIO_InitStruct);
638
639     /*Configure GPIO pins : button1_Pin button2_Pin button3_Pin */
640     GPIO_InitStruct.Pin = button1_Pin|button2_Pin|button3_Pin;
641     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
642     GPIO_InitStruct.Pull = GPIO_NOPULL;
643     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
644
645     /*Configure GPIO pin : buzzer_Pin */

```

```

main.c
644
645 /*Configure GPIO pin : buzzer_Pin */
646 GPIO_InitStruct.Pin = buzzer_Pin;
647 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
648 GPIO_InitStruct.Pull = GPIO_PULLDOWN;
649 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
650 HAL_GPIO_Init(buzzer_GPIO_Port, &GPIO_InitStruct);
651
652 }
653
654 /* USER CODE BEGIN 4 */
655
656 /* USER CODE END 4 */
657
658 /**
659  * @brief This function is executed in case of error occurrence.
660  * @retval None
661  */
662 void Error_Handler(void)
663 {
664     /* USER CODE BEGIN Error_Handler_Debug */
665     /* User can add his own implementation to report the HAL error return state */
666
667     /* USER CODE END Error_Handler_Debug */
668
669 #ifndef USE_FULL_ASSERT
670 /**
671  * @brief Reports the name of the source file and the source line number
672  * where the assert_param error has occurred.
673  * @param file: pointer to the source file name
674  * @param line: assert_param error line source number
675  * @retval None
676  */
677 void assert_failed(uint8_t *file, uint32_t line)
678 {
679     /* USER CODE BEGIN 6 */
680     /* User can add his own implementation to report the file name and line number,
681     tex: printf("Wrong parameters value: file %s on line %d\n", file, line) */
682     /* USER CODE END 6 */
683 }
684 #endif /* USE_FULL_ASSERT */
685
686
687 /***** (C) COPYRIGHT STMicroelectronics *****/
688

```

```

659  * @brief This function is executed in case of error occurrence.
660  * @retval None
661  */
662 void Error_Handler(void)
663 {
664     /* USER CODE BEGIN Error_Handler_Debug */
665     /* User can add his own implementation to report the HAL error return state */
666
667     /* USER CODE END Error_Handler_Debug */
668
669 #ifndef USE_FULL_ASSERT
670 /**
671  * @brief Reports the name of the source file and the source line number
672  * where the assert_param error has occurred.
673  * @param file: pointer to the source file name
674  * @param line: assert_param error line source number
675  * @retval None
676  */
677 void assert_failed(uint8_t *file, uint32_t line)
678 {
679     /* USER CODE BEGIN 6 */
680     /* User can add his own implementation to report the file name and line number,
681     tex: printf("Wrong parameters value: file %s on line %d\n", file, line) */
682     /* USER CODE END 6 */
683 }
684 #endif /* USE_FULL_ASSERT */
685
686
687 /***** (C) COPYRIGHT STMicroelectronics *****/
688

```

Project Books Func... Temp...

Build Output

.../Core/Std/main.c: 6 warnings, 0 errors

linking...

Program Size: Code=8248 RO-data=292 RW-data=64 ZI-data=1120

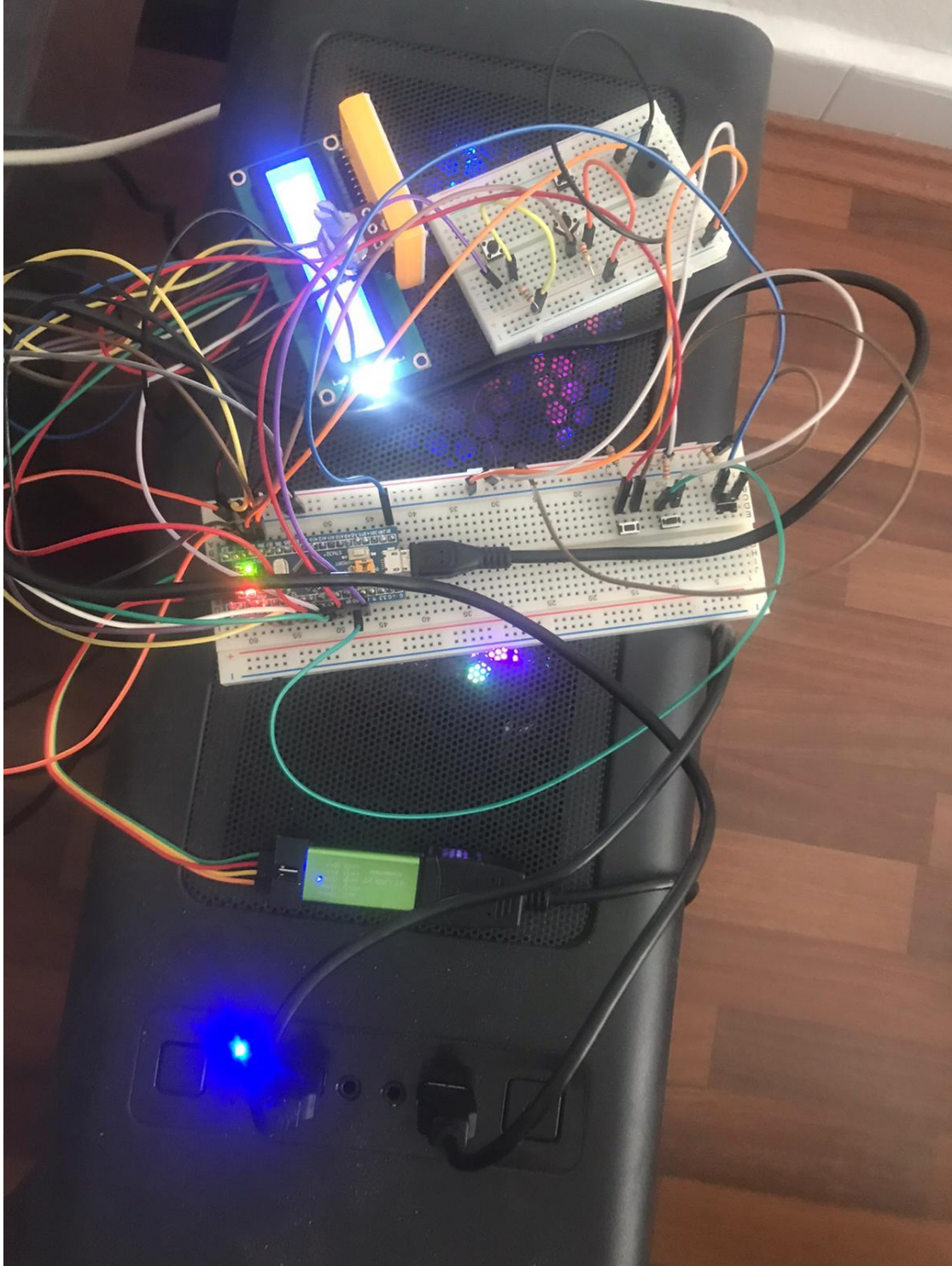
FromELF: creating hex file...

"embed\_project\embed\_proj.axf" - 0 Error(s), 6 Warning(s).

Build Time Elapsed: 00:00:01

ST-Link Debugger

### 2.3 Total System



**Project link :**  
**<https://www.youtube.com/watch?v=rWvaX9QZEO4&list=LL&index=1>**

