UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA

DISCIPLINA SISTEMAS MICROPROCESSADOS

RELÓGIO ALARME COM STM32F103C6

Equipe:

1. John Vasconcelos dos Santos — Matrícula: 414953
2. Renato Avelino — Matrícula: 485369
3. Rayane Gadelha Melo de Lima — Matrícula: 368610

Disciplina: Sistemas Microprocessados.
Data de Entrega do Projeto: 11 de abril de 2021

Fortaleza
2021

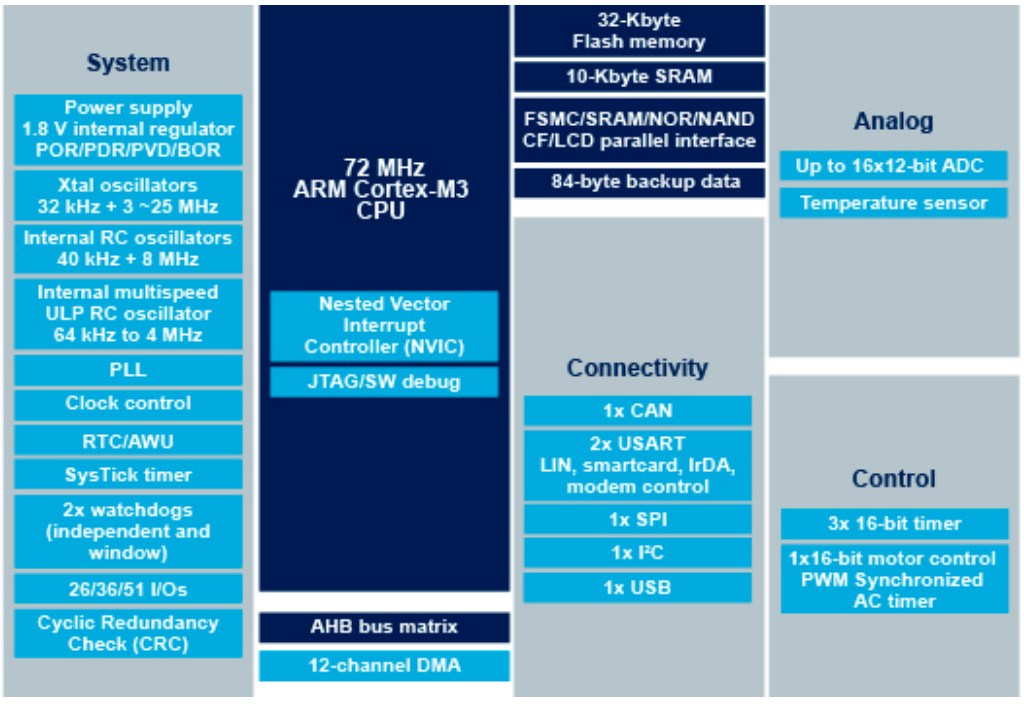
1. INTRODUÇÃO

1.1. Microprocessador STM32F103C6:

Os microprocessadores STM32F103x4 e STM32F103x6 incorpora o núcleo RISC de 32 bits ARM® Cortex ™ -M3, operando a uma frequência de 72 MHz, memória Flash incorporada de até 32 Kbytes e SRAM de até 6 Kbytes), e uma ampla gama de I / Os e periféricos aprimorados conectados a dois barramentos APB. Todos os dispositivos oferecem dois ADCs de 12 bits, três temporizadores de 16 bits de uso geral e um temporizador PWM, bem como interfaces de comunicação padrão: até dois I2Cs e SPIs, três USARTs, um USB e um CAN. A família de linhas de desempenho de baixa densidade STM32F103xx opera com uma fonte de alimentação de 2,0 a 3,6 V. Ele está disponível na faixa de temperatura de –40 a +85 ° C e na faixa de temperatura estendida de –40 a +105 ° C. Um conjunto abrangente de modo de economia de energia permite o design de aplicativos de baixo consumo de energia.

Algumas das aplicações dos microprocessadores STM32F103xx são drives de motores, controle de aplicativos, equipamentos médicos e portáteis, periféricos de PC e jogos, plataformas GPS, aplicações industriais, PLCs, inversores, impressoras, scanners, sistemas de alarme, intercomunicadores de vídeo e HVACs.

Figura 01-Caracteristicas do Stm32F103XX



1.2. Real Time Clock (RTC):

Um Real Time Clock (RTC) é um elemento de tempo dedicado para cronometragem de tempo, utilizado especialmente em operações que necessitam de um temporizador com alta precisão. Exemplos de tais aplicações além de relógios digitais, pode-se incluir máquinas de lavar, dispensadores de remédios, registradores de dados, etc. Na maioria dos

MCUs de 8 bits, como os PICs e AVRs regulares, não há módulos RTC embutidos e, portanto, precisa usar chips RTC dedicados como o popular DS1302 ou PCF8563. Os MCUs STM32 vêm com módulos RTC integrados que não requerem suporte de hardware adicional.

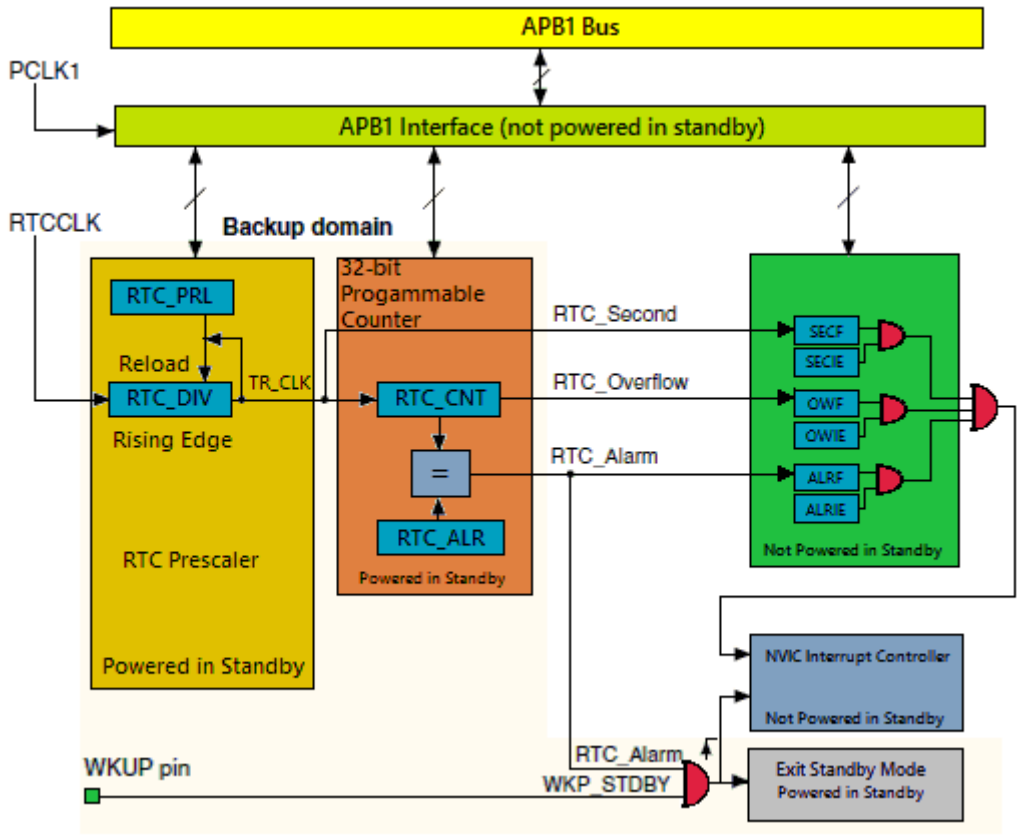
O RTC embutido de um micro STM32 é um contador de cronômetro independente com codificação binária decimal (BCD). O núcleo RTC consiste em contadores, prescalers, divisores de relógio, registros de dados de alarme, etc. Como com qualquer chip RTC padrão, o RTC integrado pode ser usado para fornecer um calendário baseado em software completo junto com funções de alarme. Os principais recursos do RTC embutido STM32 são destacados abaixo:

- Prescaler programável
- Contador programável de 32 bits para medição de longo prazo
- Dois relógios separados: PCLK1 para a interface APB1 e relógio RTC.
- A fonte de relógio RTC pode ser qualquer uma das seguintes: HSE clock divided by 128, LSE oscillator clock E LSI oscillator clock.
- Três linhas de interrupção dedicadas: Interrupção de alarme, para gerar uma interrupção de alarme programável por software, Interrupção de segundos, para gerar um sinal de interrupção periódica com uma duração de período programável (até 1 segundo), Interrupção de estouro, para detectar quando o contador programável interno chega a zero.

O RTC do STM32 é composto por dois componentes principais. A primeira é a interface de barramento APB1. A interface do barramento APB1 consiste em divisores de clock e é usada para se conectar ao barramento APB1. Essa interface também consiste em um conjunto de registradores de 16 bits que podem ser lidos / gravados por operações de barramento APB1.

O outro componente é o núcleo RTC. Consiste em um grupo de contadores programáveis divididos em dois módulos. O primeiro módulo é o módulo prescaler RTC. Este módulo gera uma base de tempo programável de um segundo (TR_CLK) usando um divisor programável de 20 bits. O outro módulo é um contador programável de 32 bits usado para manter a contagem dos segundos. Como tem 32 bits de largura, com um período de TR_CLK de um segundo, ele pode registrar até 4.294.967.296 segundos ou cerca de um século.

Figura 02-RTC interno Stm32F103XX



O núcleo RTC é totalmente independente da interface RTC APB1, assim como o Independent Watchdog Timer (IWDG). Os registros, contadores, alarmes e sinalizadores RTC são acessíveis por meio da interface APB1, mas os registros do contador são atualizados por um relógio RTC separado. O núcleo RTC continua funcionando mesmo se o barramento APB1 estiver sem energia. Isso só é possível com backup de fonte de alimentação usando uma bateria ou um supercapacitor. A Maioria dos registradores RTC tem 16 bits de largura. Existem dois registradores de controle RTC que definem as propriedades RTC.

2. OBJETIVOS

Projeta e simular no Proteus de um relógio digital com alarme utilizado o microprocessador STM32F103C6.

3. MATERIAIS

- Simulador de circuitos digitais Proteus;
- Microprocessador STM32F103C6;
- Chave digital micro Switch;
- 07 chaves Botões circuitos integrais e digitais;
- 06 Display 7 segmentos;
- 01 Led vermelho;
- 07 Resistores de 10kΩ;
- 08 Resistores de 330kΩ.

4. PROCEDIMENTOS

4.1. Código:

Inicialmente, foi configurado as portas do microprocessador STM32F103C6 no STM32Club versão 1.6.0. Posteriormente, foram configurados o RCC_OSC interno, o RTC e as interrupções. Compilou as configurações gerando um código fonte pré configurados, adicionando funções conforme a necessidade do projeto.

4.2. Montagem e simulação no Proteus:

Foi realizada a montagem do circuito no Proteus utilizado os matérias listados, verificado e ajustado o código fonte para melhor aplicabilidade do projeto.

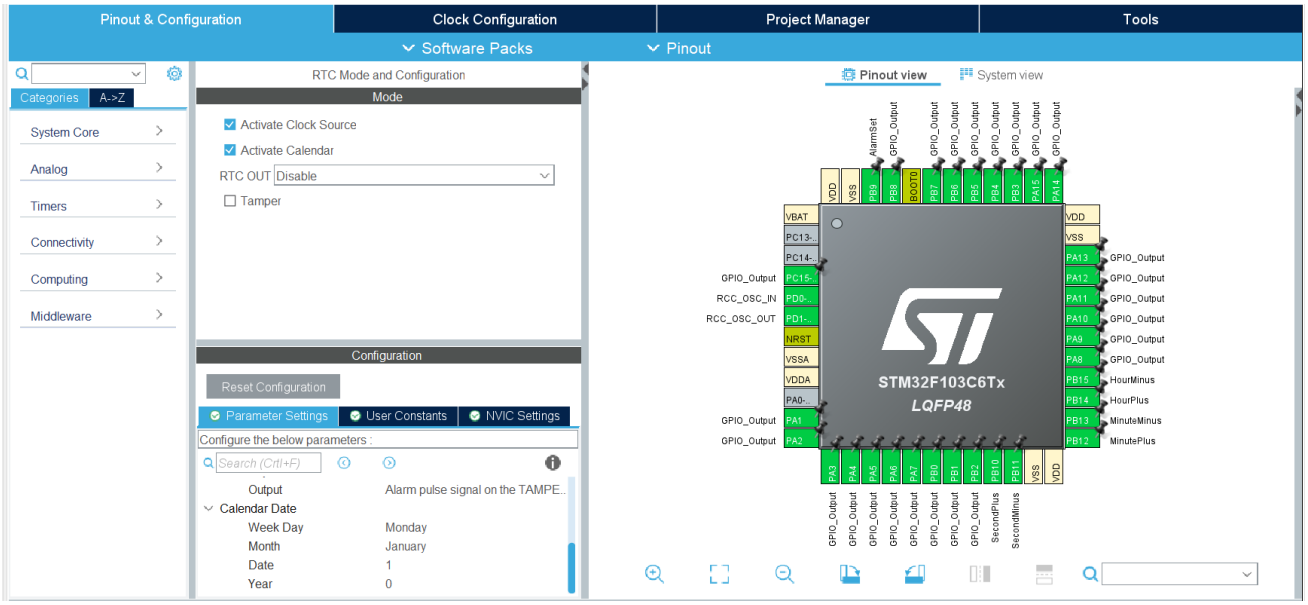
5. CONCLUSÃO E RESULTADOS

5.1. Código:

Inicialmente, foi configurado as portas do microprocessador STM32F103C6 no STM32Club versão 1.6.0.

Porta	*Configuração*	*Utilização*
PA01-PA15	GPIO_OUTPUT	Representação das horas, minutos e segundos no Display 7 segmentos.
PB0-PB08	GPIO_OUTPUT	Representação das horas, minutos e segundos no Display 7 segmentos.
PB009-PB015	GPIO_EXTI	Interrupção da função principal através de chaves digitais.
PC15	GPIO_OUTPUT	Acendimento de um LED quando o tempo do relógio for igual do Alarme.
PD0-PD01	RCC_OSC	Cristal ressonador interno

Posteriormente, foi escolhido o RCC_OSC interno, ativado o RTC, com ativação do Clock source, Calendar, interrupção global do RTC, interrupção alarm do RTC e configurado hora e data atuais.



Compilou as configurações e gerou um código fonte pré configurado, adicionou as seguintes funções:

/* **Declaração de variáveis globais do RTC** */

```
/* Private variables ---------------------------------------------------------*/
RTC_HandleTypeDef hrtc;

/* USER CODE BEGIN PV */

uint8_t alarmflag = 0;
RTC_TimeTypeDef sTime = {0};
RTC_AlarmTypeDef sAlarm = {0};

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
```

/* **SET do valor inicial do alarme** */

```
  /* Infinite loop */

  sAlarm.AlarmTime.Hours = 9;
  sAlarm.AlarmTime.Minutes = 16;
  sAlarm.AlarmTime.Seconds = 20;
  sAlarm.Alarm = RTC_ALARM_A;

  /* USER CODE BEGIN WHILE */
```

/* **Leitura do valor contido no Time do RTC e no Alarme e transformação de binária decimal (BCD) em binário para o Display 7 segmentos***/

```
  while (1)
    {
     /* USER CODE END WHILE */

     /* USER CODE BEGIN 3 */
     HAL_Delay(100);
     RTC_TimeTypeDef tmpTime;
     RTC_AlarmTypeDef tmpAlarm;
     HAL_RTC_GetTime(&hrtc, &tmpTime, RTC_FORMAT_BIN);
     HAL_RTC_GetAlarm(&hrtc, &tmpAlarm, RTC_ALARM_A, RTC_FORMAT_BIN);

     if(alarmflag == 0) {
            uint8_t secFirstDigit = tmpTime.Seconds%10;
            uint8_t secSecondDigit = tmpTime.Seconds/10;

            //uint8_t secFirstDigit =
alarmflag?(tmpAlarm.AlarmTime.Seconds%10):(tmpTime.Seconds%10);
            //uint8_t secSecondDigit =
alarmflag?(tmpAlarm.AlarmTime.Seconds/10):(tmpTime.Seconds/10);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, secFirstDigit & 0x01);
                  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, secFirstDigit & 0x02);
                  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, secFirstDigit & 0x04);
                  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, secFirstDigit & 0x08);

                  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, secSecondDigit & 0x01);
                  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, secSecondDigit & 0x02);
                  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, secSecondDigit & 0x04);
                  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, secSecondDigit & 0x08);

            uint8_t minFirstDigit = tmpTime.Minutes%10;
            uint8_t minSecondDigit = tmpTime.Minutes/10;

            //uint8_t minFirstDigit =
alarmflag?(tmpAlarm.AlarmTime.Minutes%10):(tmpTime.Minutes%10);
            //uint8_t minSecondDigit =
alarmflag?(tmpAlarm.AlarmTime.Minutes/10):(tmpTime.Minutes/10);

            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, minFirstDigit & 0x01);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, minFirstDigit & 0x02);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, minFirstDigit & 0x04);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, minFirstDigit & 0x08);

            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, minSecondDigit & 0x01);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, minSecondDigit & 0x02);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, minSecondDigit & 0x04);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, minSecondDigit & 0x08);

            uint8_t hourFirstDigit = tmpTime.Hours%10;
```

```
		uint8_t hourSecondDigit = tmpTime.Hours/10;

		//uint8_t hourFirstDigit =
alarmflag?(tmpAlarm.AlarmTime.Hours%10):(tmpTime.Hours%10);
		//uint8_t hourSecondDigit =
alarmflag?(tmpAlarm.AlarmTime.Hours/10):(tmpTime.Hours/10);
		HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, hourFirstDigit & 0x01);
		HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, hourFirstDigit & 0x02);
		HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, hourFirstDigit & 0x04);
		HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, hourFirstDigit & 0x08);

		HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, hourSecondDigit & 0x01);
		HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, hourSecondDigit & 0x02);
		HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, hourSecondDigit & 0x04);
		HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, hourSecondDigit & 0x08);
	}
	else if (alarmflag == 1) {
		uint8_t secFirstDigit = tmpAlarm.AlarmTime.Seconds%10;
		uint8_t secSecondDigit = tmpAlarm.AlarmTime.Seconds/10;

		//uint8_t secFirstDigit =
alarmflag?(tmpAlarm.sAlarm.AlarmTime.Seconds%10):(tmpTime.Seconds%10);
		//uint8_t secSecondDigit =
alarmflag?(tmpAlarm.AlarmTime.Seconds/10):(tmpTime.Seconds/10);

		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, secFirstDigit & 0x01);
		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, secFirstDigit & 0x02);
		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, secFirstDigit & 0x04);
		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, secFirstDigit & 0x08);

		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, secSecondDigit & 0x01);
		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, secSecondDigit & 0x02);
		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, secSecondDigit & 0x04);
		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, secSecondDigit & 0x08);

		uint8_t minFirstDigit = tmpAlarm.AlarmTime.Minutes%10;
		uint8_t minSecondDigit = tmpAlarm.AlarmTime.Minutes/10;

		//uint8_t minFirstDigit =
alarmflag?(tmpAlarm.AlarmTime.Minutes%10):(tmpTime.Minutes%10);
		//uint8_t minSecondDigit =
alarmflag?(tmpAlarm.AlarmTime.Minutes/10):(tmpTime.Minutes/10);

		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, minFirstDigit & 0x01);
			HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, minFirstDigit & 0x02);
			HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, minFirstDigit & 0x04);
			HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, minFirstDigit & 0x08);

			HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, minSecondDigit &
0x01);
			HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, minSecondDigit &
0x02);
			HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, minSecondDigit &
0x04);
			HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, minSecondDigit & 0x08);
```

```
            uint8_t hourFirstDigit = tmpAlarm.AlarmTime.Hours%10;
            uint8_t hourSecondDigit = tmpAlarm.AlarmTime.Hours/10;

            //uint8_t hourFirstDigit =
alarmflag?(tmpAlarm.AlarmTime.Hours%10):(tmpTime.Hours%10);
            //uint8_t hourSecondDigit =
alarmflag?(tmpAlarm.AlarmTime.Hours/10):(tmpTime.Hours/10);


            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, hourFirstDigit & 0x01);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, hourFirstDigit & 0x02);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, hourFirstDigit & 0x04);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, hourFirstDigit & 0x08);

            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, hourSecondDigit & 0x01);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, hourSecondDigit & 0x02);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, hourSecondDigit & 0x04);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, hourSecondDigit & 0x08);
      }
```

/* **Função comparativa entre Time e alarme, se for igual faz piscar um Led na Porta C Pino 15***/

```
      if (tmpAlarm.AlarmTime.Hours == tmpTime.Hours) {
                  if (tmpAlarm.AlarmTime.Minutes == tmpTime.Minutes) {
                        if(tmpAlarm.AlarmTime.Seconds == tmpTime.Seconds) {
                              HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_15);
                              HAL_Delay(200);



                        }
                  }
            }
       }
   /* USER CODE END 3 */
  }


  /** Initialize RTC and set the Time and Date  */
    sTime.Hours = 0x9;
    sTime.Minutes = 0x16;
    sTime.Seconds = 0x20;

    if (HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BCD) != HAL_OK)
    {
      Error_Handler();
    }
    DateToUpdate.WeekDay = RTC_WEEKDAY_MONDAY;
    DateToUpdate.Month = RTC_MONTH_JANUARY;
    DateToUpdate.Date = 0x1;
    DateToUpdate.Year = 0x0;
```

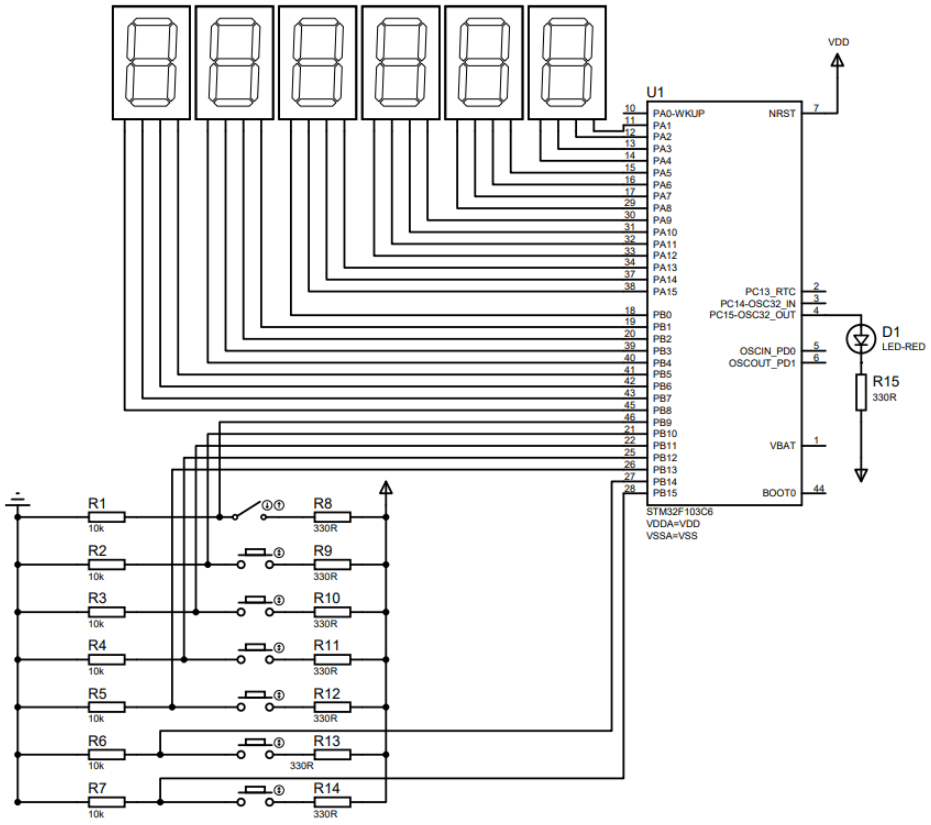
/ Interrupção do Time do RTC através da chave no Porta B Pino 09, adicionando ou subtraindo valores conforme o botão acionado */**

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {

  UNUSED(GPIO_Pin);
  switch(GPIO_Pin) {
        case AlarmSet_Pin: if (alarmflag) alarmflag = 0; else alarmflag = 1; break;
        case SecondPlus_Pin: if (alarmflag) sAlarm.AlarmTime.Seconds++; break;
        case MinutePlus_Pin: if (alarmflag) sAlarm.AlarmTime.Minutes++; break;
        case HourPlus_Pin: if (alarmflag) sAlarm.AlarmTime.Hours++; break;
        case SecondMinus_Pin: if (alarmflag) sAlarm.AlarmTime.Seconds--; break;
        case MinuteMinus_Pin: if (alarmflag) sAlarm.AlarmTime.Minutes--; break;
        case HourMinus_Pin: if (alarmflag) sAlarm.AlarmTime.Hours--; break;
  }
  if(alarmflag) HAL_RTC_SetAlarm(&hrtc, &sAlarm, RTC_FORMAT_BIN);

}
```

5.2. Montagem e simulação no Proteus:



Foi realizada a montagem do circuito no Proteus utilizado os matérias listados, verificado e ajustado o código fonte para melhor aplicabilidade do projeto.

Figura 02-Projeto Relógio alarme

Dispositivo	*Configuração*	*Utilização*
Display 01	Interligado nas PA01-PA04	segundos
Display 02	Interligado nas PA05-PA08	segundos
Display 03	Interligado nas PA09-PA12	Minutos
Display 04	Interligado nas PA13-PB0	Minutos
Display 05	Interligado nas PB01-PA04	Horas
Display 05	Interligado nas PB05-PA08	Horas
SWITCH	Interligado nas PB09	Interrupção da contagem de tempo para set alarme
Botão 01	Interligado nas PB10	set segundos alarme para mais
Botão 02	Interligado nas PB11	set segundos alarme para menos

Dispositivo	Configuração	Utilização
Botão 03	Interligado nas PB12	set minutos alarme para mais
Botão 04	Interligado nas PB13	set minutos alarme para menos
Botão 05	Interligado nas PB14	set horas alarme para mais
Botão 06	Interligado nas PB15	set horas alarme para menos
Led	Interligado nas PC15	Pisca quando alarme = Time

6. REFERÊNCIAS

ControllersTech. **Internal RTC in STM32.** Disponível em: https://controllerstech.com/internal-rtc-in-stm32/. Acessado em 05/04/2021.

Embedded Lab. **STM32'S INTERNAL RTC.** Disponível em: http://embedded-lab.com/blog/stm32s-internal-rtc . Acessado em 05/04/2021.

STMicroelectronics. **Using the hardware real-time clock (RTC) and the tamper management unit (TAMP) with STM32 microcontrollers**. AN4759 Application note. May, 2017.

STMicroelectronics. **Reference manual stm32f101xx stm32f102xx stm32f103xx stm32f105xx and stm32f107xx advanced ARM-based 32-bit MCUs** rev. 14, 2011.

Anexo A0

Código Fonte

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *

******************************************************************************
 */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

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

/* USER CODE END PTD */

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

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

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
RTC_HandleTypeDef hrtc;
```

```
/* USER CODE BEGIN PV */
uint8_t alarmflag = 0;
RTC_TimeTypeDef sTime = {0};
RTC_AlarmTypeDef sAlarm = {0};
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_RTC_Init(void);
static void MX_NVIC_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* 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_RTC_Init();

  /* Initialize interrupts */
  MX_NVIC_Init();
```

```
 /* USER CODE BEGIN 2 */

 /* USER CODE END 2 */

 /* Infinite loop */
 /* USER CODE BEGIN WHILE */

 sAlarm.AlarmTime.Hours = 9;
 sAlarm.AlarmTime.Minutes = 16;
 sAlarm.AlarmTime.Seconds = 20;
 sAlarm.Alarm = RTC_ALARM_A;
 while (1)
 {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
          HAL_Delay(100);
          RTC_TimeTypeDef tmpTime;
          RTC_AlarmTypeDef tmpAlarm;
          HAL_RTC_GetTime(&hrtc, &tmpTime, RTC_FORMAT_BIN);
          HAL_RTC_GetAlarm(&hrtc, &tmpAlarm, RTC_ALARM_A, RTC_FORMAT_BIN);

          if(alarmflag == 0) {
                  uint8_t secFirstDigit = tmpTime.Seconds%10;
                  uint8_t secSecondDigit = tmpTime.Seconds/10;

                  //uint8_t secFirstDigit =
alarmflag?(tmpAlarm.AlarmTime.Seconds%10):(tmpTime.Seconds%10);
                  //uint8_t secSecondDigit =
alarmflag?(tmpAlarm.AlarmTime.Seconds/10):(tmpTime.Seconds/10);
                  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, secFirstDigit & 0x01);
                          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, secFirstDigit & 0x02);
                          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, secFirstDigit & 0x04);
                          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, secFirstDigit & 0x08);

                          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, secSecondDigit & 0x01);
                          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, secSecondDigit & 0x02);
                          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, secSecondDigit & 0x04);
                          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, secSecondDigit & 0x08);

                  uint8_t minFirstDigit = tmpTime.Minutes%10;
                  uint8_t minSecondDigit = tmpTime.Minutes/10;

                  //uint8_t minFirstDigit =
alarmflag?(tmpAlarm.AlarmTime.Minutes%10):(tmpTime.Minutes%10);
                  //uint8_t minSecondDigit =
alarmflag?(tmpAlarm.AlarmTime.Minutes/10):(tmpTime.Minutes/10);

                  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, minFirstDigit & 0x01);
                  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, minFirstDigit & 0x02);
                  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, minFirstDigit & 0x04);
                  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, minFirstDigit & 0x08);
```

```
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, minSecondDigit & 0x01);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, minSecondDigit & 0x02);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, minSecondDigit & 0x04);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, minSecondDigit & 0x08);

                uint8_t hourFirstDigit = tmpTime.Hours%10;
                uint8_t hourSecondDigit = tmpTime.Hours/10;

                //uint8_t hourFirstDigit =
alarmflag?(tmpAlarm.AlarmTime.Hours%10):(tmpTime.Hours%10);
                //uint8_t hourSecondDigit =
alarmflag?(tmpAlarm.AlarmTime.Hours/10):(tmpTime.Hours/10);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, hourFirstDigit & 0x01);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, hourFirstDigit & 0x02);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, hourFirstDigit & 0x04);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, hourFirstDigit & 0x08);

                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, hourSecondDigit & 0x01);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, hourSecondDigit & 0x02);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, hourSecondDigit & 0x04);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, hourSecondDigit & 0x08);
        }
        else if (alarmflag == 1) {
                uint8_t secFirstDigit = tmpAlarm.AlarmTime.Seconds%10;
                uint8_t secSecondDigit = tmpAlarm.AlarmTime.Seconds/10;

                //uint8_t secFirstDigit =
alarmflag?(tmpAlarm.sAlarm.AlarmTime.Seconds%10):(tmpTime.Seconds%10);
                //uint8_t secSecondDigit =
alarmflag?(tmpAlarm.AlarmTime.Seconds/10):(tmpTime.Seconds/10);

                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, secFirstDigit & 0x01);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, secFirstDigit & 0x02);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, secFirstDigit & 0x04);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, secFirstDigit & 0x08);

                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, secSecondDigit & 0x01);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, secSecondDigit & 0x02);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, secSecondDigit & 0x04);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, secSecondDigit & 0x08);

                uint8_t minFirstDigit = tmpAlarm.AlarmTime.Minutes%10;
                uint8_t minSecondDigit = tmpAlarm.AlarmTime.Minutes/10;

                //uint8_t minFirstDigit =
alarmflag?(tmpAlarm.AlarmTime.Minutes%10):(tmpTime.Minutes%10);
                //uint8_t minSecondDigit =
alarmflag?(tmpAlarm.AlarmTime.Minutes/10):(tmpTime.Minutes/10);

                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, minFirstDigit & 0x01);
                        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, minFirstDigit & 0x02);
```

```
                        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, minFirstDigit & 0x04);
                        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, minFirstDigit & 0x08);

                        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_13, minSecondDigit & 0x01);
                        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_14, minSecondDigit & 0x02);
                        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, minSecondDigit & 0x04);
                        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, minSecondDigit & 0x08);
                uint8_t hourFirstDigit = tmpAlarm.AlarmTime.Hours%10;
                uint8_t hourSecondDigit = tmpAlarm.AlarmTime.Hours/10;

                //uint8_t hourFirstDigit =
alarmflag?(tmpAlarm.AlarmTime.Hours%10):(tmpTime.Hours%10);
                //uint8_t hourSecondDigit =
alarmflag?(tmpAlarm.AlarmTime.Hours/10):(tmpTime.Hours/10);


                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, hourFirstDigit & 0x01);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, hourFirstDigit & 0x02);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, hourFirstDigit & 0x04);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, hourFirstDigit & 0x08);

                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, hourSecondDigit & 0x01);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, hourSecondDigit & 0x02);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, hourSecondDigit & 0x04);
                HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, hourSecondDigit & 0x08);
        }
        if (tmpAlarm.AlarmTime.Hours == tmpTime.Hours) {
                        if (tmpAlarm.AlarmTime.Minutes == tmpTime.Minutes) {
                                if(tmpAlarm.AlarmTime.Seconds == tmpTime.Seconds) {
                                        HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_15);
                                        HAL_Delay(200);



                                }
                        }
                }
         }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
```

```
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_LSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.LSIState = RCC_LSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI_DIV2;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV2;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_RTC;
  PeriphClkInit.RTCClockSelection = RCC_RTCCLKSOURCE_LSI;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief NVIC Configuration.
  * @retval None
  */
static void MX_NVIC_Init(void)
{
  /* RTC_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(RTC_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(RTC_IRQn);
  /* RCC_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(RCC_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(RCC_IRQn);
  /* RTC_Alarm_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(RTC_Alarm_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(RTC_Alarm_IRQn);
}

/**
```

```
  * @brief RTC Initialization Function
  * @param None
  * @retval None
  */
static void MX_RTC_Init(void)
{

  /* USER CODE BEGIN RTC_Init 0 */

  /* USER CODE END RTC_Init 0 */

  RTC_TimeTypeDef sTime = {0};
  RTC_DateTypeDef DateToUpdate = {0};

  /* USER CODE BEGIN RTC_Init 1 */

  /* USER CODE END RTC_Init 1 */
  /** Initialize RTC Only
  */
  hrtc.Instance = RTC;
  hrtc.Init.AsynchPrediv = RTC_AUTO_1_SECOND;
  hrtc.Init.OutPut = RTC_OUTPUTSOURCE_ALARM;
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
  {
    Error_Handler();
  }

  /* USER CODE BEGIN Check_RTC_BKUP */

  /* USER CODE END Check_RTC_BKUP */

  /** Initialize RTC and set the Time and Date
  */
  sTime.Hours = 0x9;
  sTime.Minutes = 0x16;
  sTime.Seconds = 0x20;

  if (HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BCD) != HAL_OK)
  {
    Error_Handler();
  }
  DateToUpdate.WeekDay = RTC_WEEKDAY_MONDAY;
  DateToUpdate.Month = RTC_MONTH_JANUARY;
  DateToUpdate.Date = 0x1;
  DateToUpdate.Year = 0x0;

  if (HAL_RTC_SetDate(&hrtc, &DateToUpdate, RTC_FORMAT_BCD) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN RTC_Init 2 */
```

```
  /* USER CODE END RTC_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_15, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4
                    |GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8
                    |GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12
                    |GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
                    |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7
                    |GPIO_PIN_8, GPIO_PIN_RESET);

  /*Configure GPIO pin : PC15 */
  GPIO_InitStruct.Pin = GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : PA1 PA2 PA3 PA4
                          PA5 PA6 PA7 PA8
                          PA9 PA10 PA11 PA12
                          PA13 PA14 PA15 */
  GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4
                    |GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8
                    |GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12
                    |GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

```
  /*Configure GPIO pins : PB0 PB1 PB2 PB3
                         PB4 PB5 PB6 PB7
                         PB8 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
                        |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7
                        |GPIO_PIN_8;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pins : SecondPlus_Pin SecondMinus_Pin MinutePlus_Pin MinuteMinus_Pin
                         HourPlus_Pin HourMinus_Pin AlarmSet_Pin */
  GPIO_InitStruct.Pin = SecondPlus_Pin|SecondMinus_Pin|MinutePlus_Pin|MinuteMinus_Pin
                        |HourPlus_Pin|HourMinus_Pin|AlarmSet_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

}

/* USER CODE BEGIN 4 */

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {

        UNUSED(GPIO_Pin);
        switch(GPIO_Pin) {
                case AlarmSet_Pin: if (alarmflag) alarmflag = 0; else alarmflag = 1; break;
                case SecondPlus_Pin: if (alarmflag) sAlarm.AlarmTime.Seconds++; break;
                case MinutePlus_Pin: if (alarmflag) sAlarm.AlarmTime.Minutes++; break;
                case HourPlus_Pin: if (alarmflag) sAlarm.AlarmTime.Hours++; break;
                case SecondMinus_Pin: if (alarmflag) sAlarm.AlarmTime.Seconds--; break;
                case MinuteMinus_Pin: if (alarmflag) sAlarm.AlarmTime.Minutes--; break;
                case HourMinus_Pin: if (alarmflag) sAlarm.AlarmTime.Hours--; break;
        }
        if(alarmflag) HAL_RTC_SetAlarm(&hrtc, &sAlarm, RTC_FORMAT_BIN);

}

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
```

```
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */

  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```