

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

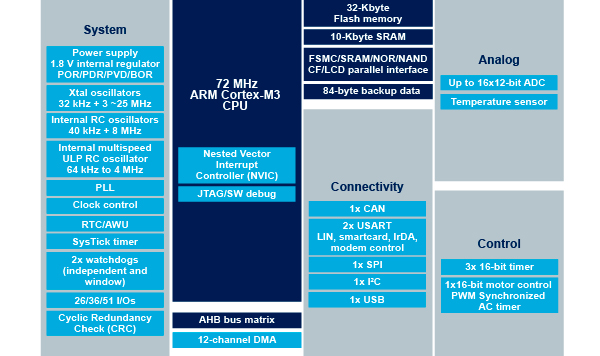
# INTRODUÇÃO

* 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. 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.

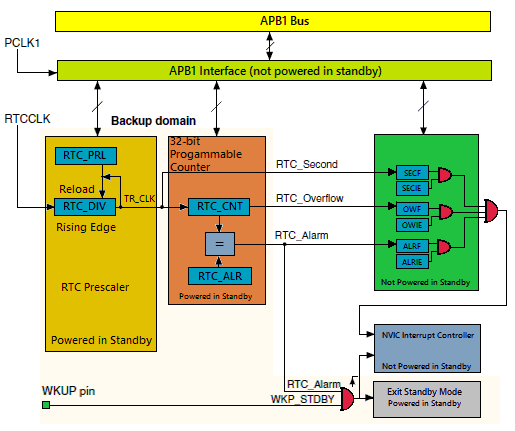
O RTC embutido de um micro STM32 é um contador de cronômetro independente com codificação binária decimal (BCD). O núcleo RTC consiste em contadores, prescalers, divisores de relógio, registros de dados de alarme, etc. Como com qualquer chip RTC padrão, o RTC integrado pode ser usado para fornecer um 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.

# OBJETIVOS

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

# 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.

# PROCEDIMENTOS

# 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.

# 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.

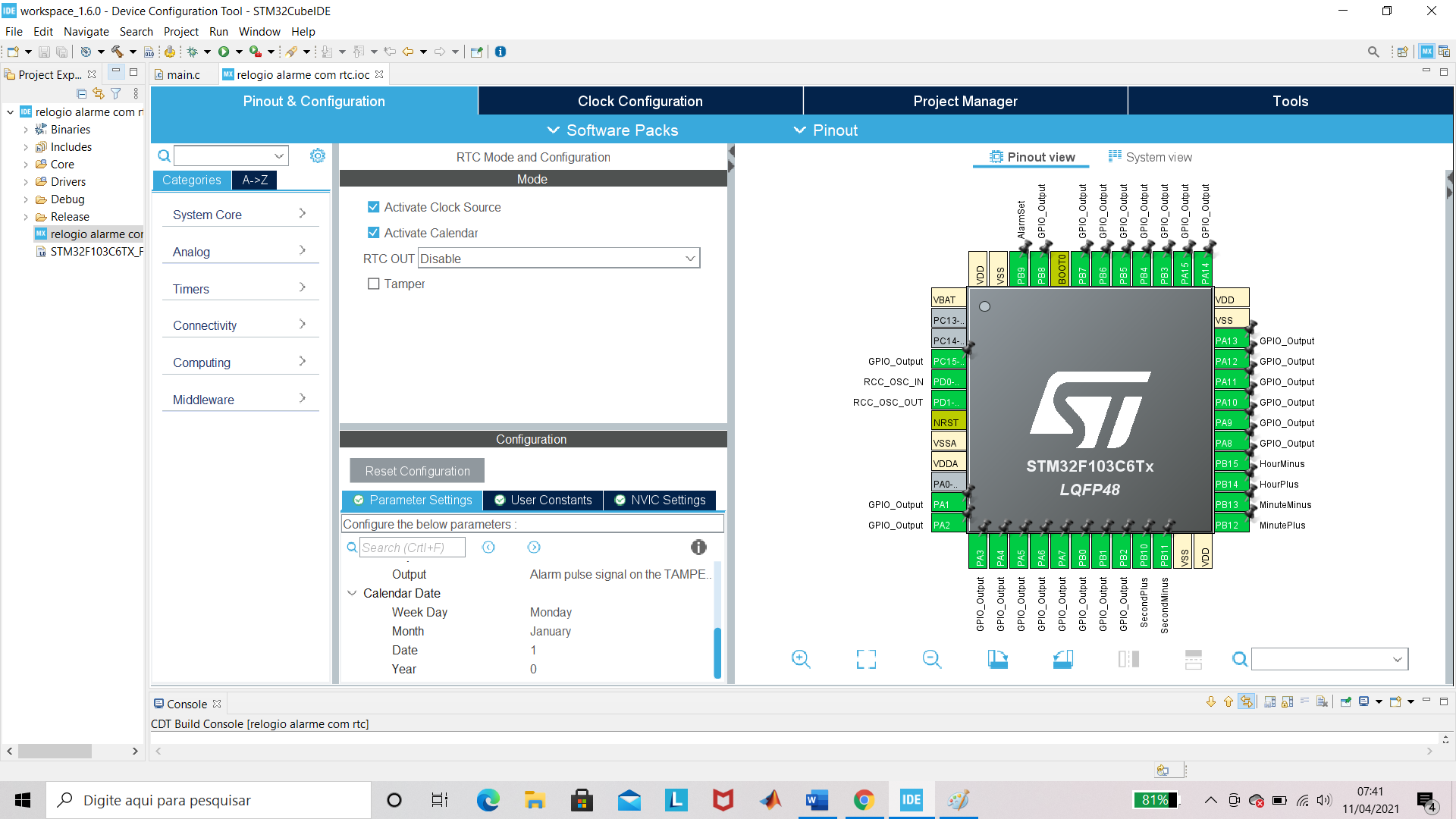
# CONCLUSÃO E RESULTADOS

# 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);

}

# 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 |

# 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\*\*\*\*/