./

Learning Report –EMBEDDED

99003588



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **To be approved By** | **Remarks/Revision Details** |
| 1 |  | SANJANA GS |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Document History**

Contents

1. CREATE A MAKE FILE………………………………………………………………………………………………………………………
2. CREATE A STARTUP FILE…………………………………………………………………………………………………………………
3. CREATE A LINKUP FILE…………………………………………………………………………………………………………………..
4. DEBUGGING TECHNIQUES…………………………………………………………………………………………………………….
5. BLINKING OF AN LED IN TOGGLING MODE…………………………………………………………………………………….
6. BLINKING OF AN LED IN INTERNAL PUSHPULL MODE…………………………………………………………………….
7. BLINKING OF AN LED IN OPEN DRIVE USING EXTERNAL PULLUP……………………………………………………..
8. BLINKING OF AN LED USING EXTERNAL BUTTON……………………………………………………………………………
9. GPIO INTERFACE USING HAL\_DRIVERS…………………………………………………………………………………………..
10. EXTERNAL INTERUPPTS USING HAL\_DRIVERS……………………………………………………………………………………
11. ADC USING HAL\_DRIVERS……………………………………………………………………………………………………………..
12. SPI USING HAL\_DRIVERS……………………………………………………………………………………………………………..
13. PWM USING HAL\_DRIVERS………………………………………………………………………………………………………….
14. USART USING HAL\_DRIVERS………………………………………………………………………………………………………..
15. SPI INTERFACE USING STM32F407 BOARD……………………………………………………………………………………..

**//Make file**

CC=arm-none-eabi-gcc

MACH=cortex-m4

CFLAGS= -c -mcpu=$(MACH) -mthumb -std=gnu11 -Wall -o0

LDFLAGS= -nostdlib -T stm.ld -Wl,-Map=final.map

all:Application.o startup.o final.elf

Application.o:Application.c

$(CC) $(CFLAGS) -o $@ $^

startup.o:startup.c

$(CC) $(CFLAGS) -o $@ $^

final.elf: Application.o startup.o

$(CC) $(LDFLAGS) -o $@ $^

clean:

rm -rf \*.o \*.elf

**//startup file**

#include<stdint.h>

#define SRAM\_START 0x20000000U

#define SRAM\_SIZE (128U \* 1024U) //128KB

#define SRAM\_END ((SRAM\_START) + (SRAM\_SIZE))

#define STACK\_START SRAM\_END

extern uint32\_t \_etext;

extern uint32\_t \_sdata;

extern uint32\_t \_edata;

extern uint32\_t \_la\_data;

extern uint32\_t \_sbss;

extern uint32\_t \_ebss;

//prototype of main

int main(void);

//void \_\_libc\_init\_array(void);

/\* function prototypes of STM32F407x system exception and IRQ handlers \*/

void Reset\_Handler(void);

void NMI\_Handler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void HardFault\_Handler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void MemManage\_Handler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void BusFault\_Handler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void UsageFault\_Handler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void SVC\_Handler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DebugMon\_Handler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void PendSV\_Handler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void SysTick\_Handler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void WWDG\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void PVD\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TAMP\_STAMP\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void RTC\_WKUP\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void RCC\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void EXTI0\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void EXTI1\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void EXTI2\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void EXTI3\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void EXTI4\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA1\_Stream0\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA1\_Stream1\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA1\_Stream2\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA1\_Stream3\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA1\_Stream4\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA1\_Stream5\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA1\_Stream6\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void ADC\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void CAN1\_TX\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void CAN1\_RX0\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void CAN1\_RX1\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void CAN1\_SCE\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void EXTI9\_5\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TIM1\_BRK\_TIM9\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TIM1\_UP\_TIM10\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TIM1\_TRG\_COM\_TIM11\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TIM1\_CC\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TIM2\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TIM3\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TIM4\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void I2C1\_EV\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void I2C1\_ER\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void I2C2\_EV\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void I2C2\_ER\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void SPI1\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void SPI2\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void USART1\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void USART2\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void USART3\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void EXTI15\_10\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void RTC\_Alarm\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void OTG\_FS\_WKUP\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TIM8\_BRK\_TIM12\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TIM8\_UP\_TIM13\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TIM8\_TRG\_COM\_TIM14\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TIM8\_CC\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA1\_Stream7\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void FSMC\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void SDIO\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TIM5\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void SPI3\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void UART4\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void UART5\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TIM6\_DAC\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void TIM7\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA2\_Stream0\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA2\_Stream1\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA2\_Stream2\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA2\_Stream3\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA2\_Stream4\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void ETH\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void ETH\_WKUP\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void CAN2\_TX\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void CAN2\_RX0\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void CAN2\_RX1\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void CAN2\_SCE\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void OTG\_FS\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA2\_Stream5\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA2\_Stream6\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DMA2\_Stream7\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void USART6\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void I2C3\_EV\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void I2C3\_ER\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void OTG\_HS\_EP1\_OUT\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void OTG\_HS\_EP1\_IN\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void OTG\_HS\_WKUP\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void OTG\_HS\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void DCMI\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void CRYP\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void HASH\_RNG\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

void FPU\_IRQHandler (void) \_\_attribute\_\_ ((weak, alias("Default\_Handler")));

uint32\_t vectors[] \_\_attribute\_\_((section(".isr\_vector"))) = {

STACK\_START,

(uint32\_t)Reset\_Handler,

(uint32\_t)NMI\_Handler,

(uint32\_t)HardFault\_Handler,

(uint32\_t)MemManage\_Handler,

(uint32\_t)BusFault\_Handler,

(uint32\_t)UsageFault\_Handler,

0,

0,

0,

0,

(uint32\_t)SVC\_Handler,

(uint32\_t)DebugMon\_Handler,

0,

(uint32\_t)PendSV\_Handler,

(uint32\_t)SysTick\_Handler,

(uint32\_t)WWDG\_IRQHandler,

(uint32\_t)PVD\_IRQHandler,

(uint32\_t)TAMP\_STAMP\_IRQHandler,

(uint32\_t)RTC\_WKUP\_IRQHandler,

0,

(uint32\_t)RCC\_IRQHandler,

(uint32\_t)EXTI0\_IRQHandler,

(uint32\_t)EXTI1\_IRQHandler,

(uint32\_t)EXTI2\_IRQHandler,

(uint32\_t)EXTI3\_IRQHandler,

(uint32\_t)EXTI4\_IRQHandler,

(uint32\_t)DMA1\_Stream0\_IRQHandler,

(uint32\_t)DMA1\_Stream1\_IRQHandler,

(uint32\_t)DMA1\_Stream2\_IRQHandler,

(uint32\_t)DMA1\_Stream3\_IRQHandler,

(uint32\_t)DMA1\_Stream4\_IRQHandler,

(uint32\_t)DMA1\_Stream5\_IRQHandler,

(uint32\_t)DMA1\_Stream6\_IRQHandler,

(uint32\_t)ADC\_IRQHandler,

(uint32\_t)CAN1\_TX\_IRQHandler,

(uint32\_t)CAN1\_RX0\_IRQHandler,

(uint32\_t)CAN1\_RX1\_IRQHandler,

(uint32\_t)CAN1\_SCE\_IRQHandler,

(uint32\_t)EXTI9\_5\_IRQHandler,

(uint32\_t)TIM1\_BRK\_TIM9\_IRQHandler,

(uint32\_t)TIM1\_UP\_TIM10\_IRQHandler,

(uint32\_t)TIM1\_TRG\_COM\_TIM11\_IRQHandler,

(uint32\_t)TIM1\_CC\_IRQHandler,

(uint32\_t)TIM2\_IRQHandler,

(uint32\_t)TIM3\_IRQHandler,

(uint32\_t)TIM4\_IRQHandler,

(uint32\_t)I2C1\_EV\_IRQHandler,

(uint32\_t)I2C1\_ER\_IRQHandler,

(uint32\_t)I2C2\_EV\_IRQHandler,

(uint32\_t)I2C2\_ER\_IRQHandler,

(uint32\_t)SPI1\_IRQHandler,

(uint32\_t)SPI2\_IRQHandler,

(uint32\_t)USART1\_IRQHandler,

(uint32\_t)USART2\_IRQHandler,

(uint32\_t)USART3\_IRQHandler,

(uint32\_t)EXTI15\_10\_IRQHandler,

(uint32\_t)RTC\_Alarm\_IRQHandler,

(uint32\_t)OTG\_FS\_WKUP\_IRQHandler,

(uint32\_t)TIM8\_BRK\_TIM12\_IRQHandler,

(uint32\_t)TIM8\_UP\_TIM13\_IRQHandler,

(uint32\_t)TIM8\_TRG\_COM\_TIM14\_IRQHandler,

(uint32\_t)TIM8\_CC\_IRQHandler,

(uint32\_t)DMA1\_Stream7\_IRQHandler,

(uint32\_t)FSMC\_IRQHandler,

(uint32\_t)SDIO\_IRQHandler,

(uint32\_t)TIM5\_IRQHandler,

(uint32\_t)SPI3\_IRQHandler,

(uint32\_t)UART4\_IRQHandler,

(uint32\_t)UART5\_IRQHandler,

(uint32\_t)TIM6\_DAC\_IRQHandler,

(uint32\_t)TIM7\_IRQHandler,

(uint32\_t)DMA2\_Stream0\_IRQHandler,

(uint32\_t)DMA2\_Stream1\_IRQHandler,

(uint32\_t)DMA2\_Stream2\_IRQHandler,

(uint32\_t)DMA2\_Stream3\_IRQHandler,

(uint32\_t)DMA2\_Stream4\_IRQHandler,

(uint32\_t)ETH\_IRQHandler,

(uint32\_t)ETH\_WKUP\_IRQHandler,

(uint32\_t)CAN2\_TX\_IRQHandler,

(uint32\_t)CAN2\_RX0\_IRQHandler,

(uint32\_t)CAN2\_RX1\_IRQHandler,

(uint32\_t)CAN2\_SCE\_IRQHandler,

(uint32\_t)OTG\_FS\_IRQHandler,

(uint32\_t)DMA2\_Stream5\_IRQHandler,

(uint32\_t)DMA2\_Stream6\_IRQHandler,

(uint32\_t)DMA2\_Stream7\_IRQHandler,

(uint32\_t)USART6\_IRQHandler,

(uint32\_t)I2C3\_EV\_IRQHandler,

(uint32\_t)I2C3\_ER\_IRQHandler,

(uint32\_t)OTG\_HS\_EP1\_OUT\_IRQHandler,

(uint32\_t)OTG\_HS\_EP1\_IN\_IRQHandler,

(uint32\_t)OTG\_HS\_WKUP\_IRQHandler,

(uint32\_t)OTG\_HS\_IRQHandler,

(uint32\_t)DCMI\_IRQHandler,

(uint32\_t)CRYP\_IRQHandler,

(uint32\_t)HASH\_RNG\_IRQHandler,

(uint32\_t)FPU\_IRQHandler,

};

void Default\_Handler(void)

{

while(1);

}

void Reset\_Handler(void)

{

//copy .data section to SRAM

uint32\_t size = (uint32\_t)&\_edata - (uint32\_t)&\_sdata;

uint8\_t \*pDst = (uint8\_t\*)&\_sdata; //sram

uint8\_t \*pSrc = (uint8\_t\*)&\_la\_data; //flash

for(uint32\_t i =0 ; i < size ; i++)

{

\*pDst++ = \*pSrc++;

}

//Init. the .bss section to zero in SRAM

size = (uint32\_t)&\_ebss - (uint32\_t)&\_sbss;

pDst = (uint8\_t\*)&\_sbss;

for(uint32\_t i =0 ; i < size ; i++)

{

\*pDst++ = 0;

}

//\_\_libc\_init\_array ();

main();

}

**//link up file**

#include <stdio.h>

#include<stdlib.h>

#include<string.h>

int numbers[] = { 456,345,678,567,890,456,3456,123,765,456,896,456,678,987,000,145,90};

int someData = 90;

void array\_fill\_numbers(int pNumbers[], unsigned int len)

{

for ( unsigned int i = 0 ; i < len ; i++)

{

pNumbers[i] = numbers[i];//rand() % 1000;

}

}

void display\_numbers(int \*pNumbers, unsigned int len, char \*pMessage)

{

//printf("%s",pMessage);

for (unsigned i = 0 ; i < len ; i++)

{

// printf("%5d ",pNumbers[i]);

}

// printf("\n");

}

void swap\_numbers(int \*x,int \*y)

{

int temp=\*x;

\*x=\*y;

\*y=temp;

#if 0

void (\*jump\_addr) (void);

jump\_addr = (void\*)0x20000009;

jump\_addr();

#endif

someData = 10;

}

void bubble\_sort(int \*pNumbers , unsigned int len)

{

int i,j,flag=0;

for(i=0;i<len-1;i++)

{

flag=0;

for(j=0;j<len-1-i;j++)

{

if(pNumbers[j] > pNumbers[j+1])

{

swap\_numbers(&pNumbers[j],&pNumbers[j+1]);

flag=1;

}

}

if(flag==0)

break;

}

}

void insertion\_sort(int \*pNumbers , unsigned int len)

{

int i,j,num;

for(i=1 ; i<len ; i++)

{

j=i-1;

num = pNumbers[i];

while( (j>-1) && (pNumbers[j] > num) )

{

pNumbers[j+1] = pNumbers[j];

j--;

}

pNumbers[j+1]=num;

}

}

int main()

{

unsigned int len = sizeof(numbers)/sizeof(int);

array\_fill\_numbers(numbers,len);

display\_numbers(numbers,len,"B-unsorted array :");

bubble\_sort(numbers,len);

display\_numbers(numbers,len,"B-sorted array :");

array\_fill\_numbers(numbers,len);

display\_numbers(numbers,len,"I-unsorted array :");

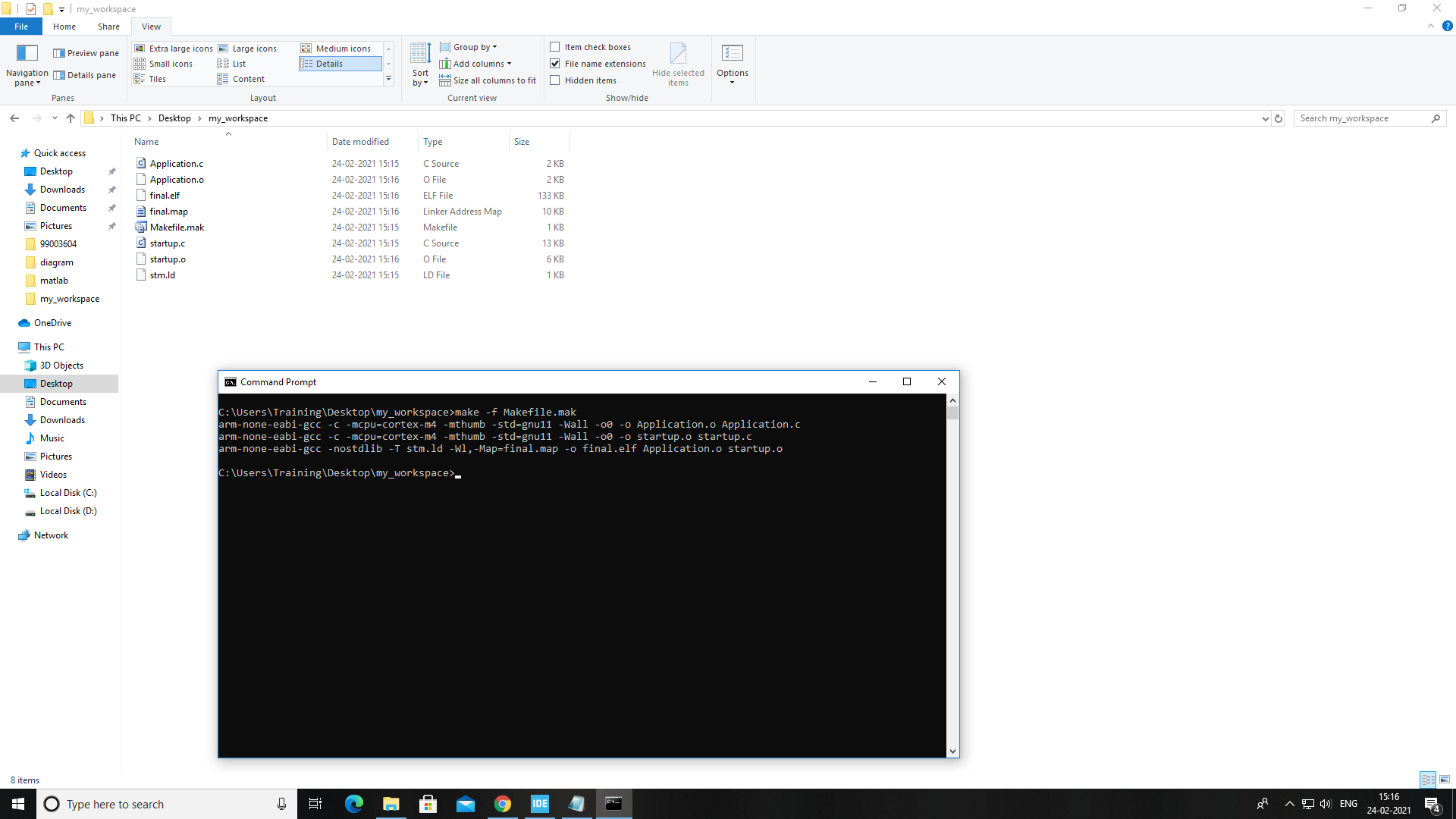
insertion\_sort(numbers,len);

display\_numbers(numbers,len,"I-sorted array :");

return 0;

}

//All created make file output:

****

**//MCU specific Header file**

#ifndef \_STM32F4XX\_H

#define \_STM32F4XX\_H

**#include<stdint.h>**

#define \_vo volatile

**// DEFINING MEMORY BASE ADDRESSES**

#define SRAM1\_BASEADDR 0x20000000U

#define SRAM2\_BASEADDR 0x2001C000U

#define FLASH\_BASEADDR 0x08000000U

#define ROM\_BASEADDR 0x1FFF0000U

**//DEFINING PHERIPHERAL BASE ADDRESSES(BUSES)**

#define AHB1\_BASEADDR 0x40020000U

#define AHB2\_BASEADDR 0x50000000U

#define APB1\_BASEADDR 0x40000000U

#define APB2\_BASEADDR 0x40010000U

**//DEFINING PERIPHERALS BASE ADRESSES WHICH IS HANGING ON TO AHB1 BUS //**

#define GPIOA\_BASEADDR (AHB1\_BASEADDR + 0x0U) //0x40020000U

#define GPIOB\_BASEADDR (AHB1\_BASEADDR + 0x0400U) //0x40020400U

#define GPIOC\_BASEADDR (AHB1\_BASEADDR + 0x0800U)

#define GPIOD\_BASEADDR (AHB1\_BASEADDR + 0x0C00U)

#define GPIOE\_BASEADDR (AHB1\_BASEADDR + 0x1000U)

#define GPIOF\_BASEADDR (AHB1\_BASEADDR + 0x1400U)

#define GPIOG\_BASEADDR (AHB1\_BASEADDR + 0x1800U)

#define GPIOH\_BASEADDR (AHB1\_BASEADDR + 0x1C00U)

#define GPIOI\_BASEADDR (AHB1\_BASEADDR + 0x2000U)

#define GPIOJ\_BASEADDR (AHB1\_BASEADDR + 0x2400U)

#define GPIOK\_BASEADDR (AHB1\_BASEADDR + 0x2800U)

**//AHB2**

#define USB\_BASEADDR (AHB2\_BASEADDR + 0x03FF)

**//APB1**

#define TIM2\_BASEADDR (APB1\_BASEADDR + 0x03FF)

**//I2S**

#define I2S2ext\_BASEADDR (APB1\_BASEADDR + 0x37FF)

#define I2S3ext\_BASEADDR (APB1\_BASEADDR + 0x43FF)

**//SPI**

#define SPI2\_I2S2\_BASEADDR (APB1\_BASEADDR + 0x3BFF)

#define SPI3\_I2S3\_BASEADDR (APB1\_BASEADDR + 0x3FFF)

#define SPI3\_I2S3\_BASEADDR (APB1\_BASEADDR + 0x3FFF)

**//USART**

#define USART2\_BASEADDR (APB1\_BASEADDR + 0x47FF)

#define USART3\_BASEADDR (APB1\_BASEADDR + 0x4BFF)

**//UART**

#define UART3\_BASEADDR (APB1\_BASEADDR + 0x4BFF)

**//APB2**

#define TIM1\_BASEADDR (APB2\_BASEADDR + 0x03FF)

//define structures

typedef struct

{

\_vo uint32\_t MODER;

\_vo uint32\_t OTYPER;

\_vo uint32\_t OSPEEDR;

\_vo uint32\_t PUPDR;

\_vo uint32\_t IDR;

\_vo uint32\_t ODR;

\_vo uint32\_t BSRR;

\_vo uint32\_t LCKR;

\_vo uint32\_t AFR[2];

} Gpio\_RegDef\_t; **// GPIO registers type//**

#define GPIOA ((Gpio\_RegDef\_t\*)GPIOA\_BASEADDR)

#define GPIOB ((Gpio\_RegDef\_t\*)GPIOB\_BASEADDR)

#define GPIOC ((Gpio\_RegDef\_t\*)GPIOC\_BASEADDR)

#define GPIOD ((Gpio\_RegDef\_t\*)GPIOD\_BASEADDR)

#define GPIOE ((Gpio\_RegDef\_t\*)GPIOE\_BASEADDR)

// Gpio\_RegDef\_t \*GPIOP[]={((Gpio\_RegDef\_t\*)GPIOA\_BASEADDR), ((Gpio\_RegDef\_t\*)GPIOB\_BASEADDR) , ((Gpio\_RegDef\_t\*)GPIOC\_BASEADDR) , ((Gpio\_RegDef\_t\*)GPIOD\_BASEADDR), ((Gpio\_RegDef\_t\*)GPIOE\_BASEADDR)};

**//ENABLE CLOCK**

//rcc base address

#define RCC\_BASEADDR (AHB1\_BASEADDR + 0x3800U)

typedef struct

{

uint32\_t RCC\_CR;

uint32\_t PLLCFGR;

uint32\_t CFGR;

uint32\_t CIR;

uint32\_t AHB1RSTR;

uint32\_t AHB2RSTR;

uint32\_t AHB3RSTR;

uint32\_t Reserved0;

uint32\_t APB1RSTR;

uint32\_t APB2RSTR;

uint32\_t Reserved1;

uint32\_t Reserved2;

uint32\_t AHB1ENR;

uint32\_t AHB2ENR;

uint32\_t AHB3ENR;

uint32\_t Reserved3;

uint32\_t APB1ENR;

uint32\_t APB2ENR;

uint32\_t Reserved4;

uint32\_t Reserved5;

uint32\_t AHB1LPENR;

uint32\_t AHB2LPENR;

uint32\_t AHB3LPENR;

uint32\_t Reserved6;

uint32\_t APB1LPENR;

uint32\_t APB2LPENR;

uint32\_t Reserved7;

uint32\_t Reserved8;

uint32\_t BDCR;

uint32\_t CSR;

uint32\_t Reserved9;

uint32\_t Reserved10;

uint32\_t SSCGR;

uint32\_t PLLI2SCFGR;

}RCC\_RegDef\_t;

#define RCC ((RCC\_RegDef\_t\*)RCC\_BASEADDR)

**//PHERIFERAL CLOCK ENABLE**

#define GPIOA\_PCLK\_EN() (RCC -> AHB1ENR |= (1<<0))

#define GPIOB\_PCLK\_EN() (RCC -> AHB1ENR |= (1<<1))

#define GPIOC\_PCLK\_EN() (RCC -> AHB1ENR |= (1<<2))

#define GPIOD\_PCLK\_EN() (RCC -> AHB1ENR |= (1<<3))

#define GPIOE\_PCLK\_EN() (RCC -> AHB1ENR |= (1<<4))

**//PHERIFERAL CLOCK DISABLE**

**//TODO**

#define GPIOA\_PCLK\_DS() (RCC -> AHB1RSTR &= ~(1<<0)) //todo

#define GPIOB\_PCLK\_DS() (RCC -> AHB1RSTR &= ~(1<<1))

#define GPIOC\_PCLK\_DS() (RCC -> AHB1RSTR &= ~(1<<2))

#define GPIOD\_PCLK\_DS() (RCC -> AHB1RSTR &= ~(1<<3))

#define GPIOE\_PCLK\_DS() (RCC -> AHB1RSTR &= ~(1<<4))

**// SOME GENERIC MACROS**

#define ENABLE 1

#define DISABLE 0

#define SET ENABLE

#define RESET DISABLE

#define GPIO\_PIN\_SET ENABLE

#define GPIO\_PIN\_RESET DISABLE

**//RCC RESET PHERIFERALS**

#define GPIOA\_REG\_RESET()

do

{ (RCC->AHB1RSTR |= (1<<0)); (RCC->AHB1RSTR &= ~(1 << 0));

} while(0)

#define GPIOB\_REG\_RESET()

do{ (RCC->AHB1RSTR |= (1<<1)); (RCC->AHB1RSTR &= ~(1 << 1));

} while(0)

#define GPIOC\_REG\_RESET()

do{ (RCC->AHB1RSTR |= (1<<2)); (RCC->AHB1RSTR &= ~(1 << 2));

} while(0)

#define GPIOD\_REG\_RESET()

do{ (RCC->AHB1RSTR |= (1<<3)); (RCC->AHB1RSTR &= ~(1 << 3));

} while(0)

#define GPIOE\_REG\_RESET()

do{ (RCC->AHB1RSTR |= (1<<4)); (RCC->AHB1RSTR &= ~(1 << 4));

} while(0)

#endif

**Drivers specific header file:**

#ifndef \_GPIO\_DRIVER\_H

#define \_GPIO\_DRIVER\_H

#include "stm32f4xx.h"

typedef struct

{

uint8\_t GPIO\_PinNumber;

uint8\_t GPIO\_PinMode;

uint8\_t GPIO\_PinSpeed;

uint8\_t GPIO\_PinPuPdControl;

uint8\_t GPIO\_PinOPType;

uint8\_t GPIO\_PinAltFuncMode;

} GPIO\_Pinconfig\_t;

typedef struct

{

Gpio\_RegDef\_t \*pGPIOx;

GPIO\_Pinconfig\_t GPIO\_PinConfig;

}GPIO\_Handle\_t;

**// PIN NUMBERS**

#define GPIO\_PIN\_NO\_O 0

#define GPIO\_PIN\_NO\_1 1

#define GPIO\_PIN\_NO\_2 2

#define GPIO\_PIN\_NO\_3 3

#define GPIO\_PIN\_NO\_4 4

#define GPIO\_PIN\_NO\_5 5

#define GPIO\_PIN\_NO\_6 6

#define GPIO\_PIN\_NO\_7 7

#define GPIO\_PIN\_NO\_8 8

#define GPIO\_PIN\_NO\_9 9

#define GPIO\_PIN\_NO\_10 10

#define GPIO\_PIN\_NO\_11 11

#define GPIO\_PIN\_NO\_12 12

#define GPIO\_PIN\_NO\_13 13

#define GPIO\_PIN\_NO\_14 14

#define GPIO\_PIN\_NO\_15 15

**//MODES OF GPIO'S**

#define GPIO\_MODE\_IN 0

#define GPIO\_MODE\_OUT 1

#define GPIO\_MODE\_ALT\_FUNC 2

#define GPIO\_MODE\_ANALOG 3

**//OUTPUT TYPE**

#define GPIO\_OP\_TYPE\_PP 0

#define GPIO\_OP\_TYPE\_OD 1

**//SPEED OG GPIO**

#define GPIO\_SPEED\_LOW 0

#define GPIO\_SPEED\_MEDIUM 1

#define GPIO\_SPEED\_HIGH 2

#define GPIO\_SPEED\_VERY\_HIGH 3

**//GPIO PIN FOR PULL UP AND PULL DOWN**

#define GPIO\_N0\_PUPD 0

#define GPIO\_PU 1

#define GPIO\_PD 2

**/\***

**\* To control clock setting**

**\*/**

void GPIO\_PeriClockControl(Gpio\_RegDef\_t \*pGPIOx, uint8\_t EnorDi);

**/\***

**\* To do initialization**

**\*/**

void GPIO\_Init(GPIO\_Handle\_t \*pGPIOHandle);

void GPIO\_DeInit(Gpio\_RegDef\_t \*pGPIOx);

**/\***

**\* read from input pin**

**\*/**

uint8\_t GPIO\_ReadFromInputPin(Gpio\_RegDef\_t \*pGPIOx, uint8\_t PinNumber);

**/\***

**\* read from input port**

**\*/**

uint16\_t GPIO\_ReadFromInputPort(Gpio\_RegDef\_t \*pGPIOx);

**/\***

**\* write to output pin**

**\*/**

void GPIO\_WriteToOutputPin(Gpio\_RegDef\_t \*pGPIOx,uint8\_t PinNumber,uint8\_t value);

**/\***

**\* read from output port**

**\*/**

void GPIO\_WriteToOutputPort(Gpio\_RegDef\_t \*pGPIOx,uint16\_t value);

**/\***

**\* to do**

**\*/**

void GPIO\_ToggleOutputPin(Gpio\_RegDef\_t \*pGPIOx,uint8\_t PinNumber);

#endif

**Driver’s source file:**

#include "stm32f4xx.h"

#include "gpio\_driver.h"

void GPIO\_PeriClockControl(Gpio\_RegDef\_t \*pGPIOx, uint8\_t EnorDi)

{

RCC\_RegDef\_t \*rcc\_ahd1en = &(RCC->AHB1ENR);

if(EnorDi== ENABLE)

{

if(pGPIOx == GPIOA)

{

GPIOA\_PCLK\_EN();

}

else if(pGPIOx == GPIOB)

{

GPIOB\_PCLK\_EN();

}

else if(pGPIOx == GPIOC)

{

GPIOC\_PCLK\_EN();

}

else if(pGPIOx == GPIOD)

{

GPIOD\_PCLK\_EN();

}

else if(pGPIOx == GPIOE)

{

GPIOE\_PCLK\_EN();

}

}

else

{

if(pGPIOx == GPIOA)

{

GPIOA\_PCLK\_DS();

}

else if(pGPIOx == GPIOB)

{

GPIOB\_PCLK\_DS();

}

else if(pGPIOx == GPIOC)

{

GPIOC\_PCLK\_DS();

}

else if(pGPIOx == GPIOD)

{

GPIOD\_PCLK\_DS();

}

else if(pGPIOx == GPIOE)

{

GPIOE\_PCLK\_DS();

}

}

}

**/\***

**\* To do**

**\*/**

void GPIO\_Init(GPIO\_Handle\_t \*pGPIOHandle)

{

**//1. CONFIGURATION OF THE MODE OF GPIOX**

uint32\_t temp=0;

temp=(pGPIOHandle->GPIO\_PinConfig.GPIO\_PinMode<< (2\*pGPIOHandle>GPIO\_PinConfig.GPIO\_PinNumber));

pGPIOHandle->pGPIOx->MODER&=~(0x3<<(2\*pGPIOHandle->GPIO\_PinConfig.GPIO\_PinNumber));

pGPIOHandle->pGPIOx->MODER |=temp; //set the value

**//2. CONIGURE THE SPEED**

temp=( pGPIOHandle->GPIO\_PinConfig.GPIO\_PinSpeed<< (2\*pGPIOHandle->GPIO\_PinConfig.GPIO\_PinNumber));

pGPIOHandle->pGPIOx->OSPEEDR &=~(0x3<<(2\*pGPIOHandle->GPIO\_PinConfig.GPIO\_PinNumber));

pGPIOHandle->pGPIOx->OSPEEDR |=temp;

**//3. PULL UP AND PULL DOWN REGISTER**

temp=(pGPIOHandle->GPIO\_PinConfig.GPIO\_PinPuPdControl<< (2\*pGPIOHandle->GPIO\_PinConfig.GPIO\_PinNumber));

pGPIOHandle->pGPIOx->PUPDR &=~(0x3<<(2\*pGPIOHandle->GPIO\_PinConfig.GPIO\_PinNumber));

pGPIOHandle->pGPIOx->PUPDR |=temp;

**//4. OUTPUT TYPE REGISTER**

temp=( pGPIOHandle->GPIO\_PinConfig.GPIO\_PinOPType << (pGPIOHandle->GPIO\_PinConfig.GPIO\_PinNumber));

pGPIOHandle->pGPIOx->OTYPER&=~(0x1<<(pGPIOHandle->GPIO\_PinConfig.GPIO\_PinNumber));

pGPIOHandle->pGPIOx->OTYPER |=temp;

**//5. ALTERNATING FUNCTION LOW REGISTER**

if(pGPIOHandle->GPIO\_PinConfig.GPIO\_PinMode== GPIO\_MODE\_ALT\_FUNC)

{

uint8\_t temp1,temp2;

temp1=(pGPIOHandle->GPIO\_PinConfig.GPIO\_PinNumber)/8;

temp2=(pGPIOHandle->GPIO\_PinConfig.GPIO\_PinNumber)%8;

pGPIOHandle->pGPIOx->AFR[temp1] &= ~(0xf<<(4\*temp2));

pGPIOHandle->pGPIOx->AFR[temp1] |= pGPIOHandle->GPIO\_PinConfig.GPIO\_PinAltFuncMode << (4\*temp2);

}

void GPIO\_DeInit(Gpio\_RegDef\_t \*pGPIOx)

{

if(pGPIOx == GPIOA)

{

GPIOA\_REG\_RESET();

}else if (pGPIOx == GPIOB)

{

GPIOB\_REG\_RESET();

}else if (pGPIOx == GPIOC)

{

GPIOC\_REG\_RESET();

}else if (pGPIOx == GPIOD)

{

GPIOD\_REG\_RESET();

}else if (pGPIOx == GPIOE)

{

GPIOE\_REG\_RESET();

}

}

/\*

\*

\*/

uint8\_t GPIO\_ReadFromInputPin(Gpio\_RegDef\_t \*pGPIOx, uint8\_t PinNumber)

{

uint8\_t value;

//value=(uint8\_t)pGPIOx->IDR;

value=(uint8\_t) ((pGPIOx->IDR >> PinNumber)& 0x00000001); //shift the the value to 0th bit and and it with 1 .so we can get the value

return(value);

}

uint16\_t GPIO\_ReadFromInputPort(Gpio\_RegDef\_t \*pGPIOx)

{

uint16\_t value;

value = pGPIOx->IDR;

return(value);

}

/\*

\*

\*/

void GPIO\_WriteToOutputPin(Gpio\_RegDef\_t \*pGPIOx,uint8\_t PinNumber,uint8\_t value)

{

if(value == GPIO\_PIN\_SET)

{

pGPIOx->ODR |= (1<<PinNumber);

}

else

{

pGPIOx->ODR &= ~(1<<PinNumber);

}

}

void GPIO\_WriteToOutputPort(Gpio\_RegDef\_t \*pGPIOx,uint16\_t value)

{

}

/\*

\* TO DO

\*/

void GPIO\_ToggleOutputPin(Gpio\_RegDef\_t \*pGPIOx,uint8\_t PinNumber)

{

pGPIOx->ODR ^= (1<<PinNumber);

}

**LED TOGGLE**

**MAIN FILE:**

#include "stm32f407xx.h"

void delay(void)

{

for(uint32\_t i = 0 ; i < 500000 ; i ++);

}

int main(void)

{

GPIO\_Handle\_t GpioLed;

GpioLed.pGPIOx = GPIOD;

GpioLed.GPIO\_PinConfig.GPIO\_PinNumber = GPIO\_PIN\_NO\_12;

GpioLed.GPIO\_PinConfig.GPIO\_PinMode = GPIO\_MODE\_OUT;

GpioLed.GPIO\_PinConfig.GPIO\_PinSpeed = GPIO\_SPEED\_FAST;

GpioLed.GPIO\_PinConfig.GPIO\_PinOPType = GPIO\_OP\_TYPE\_OD;

GpioLed.GPIO\_PinConfig.GPIO\_PinPuPdControl = GPIO\_NO\_PUPD;

GPIO\_PeriClockControl(GPIOD,ENABLE);

GPIO\_Init(&GpioLed);

while(1)

{

GPIO\_ToggleOutputPin(GPIOD,GPIO\_PIN\_NO\_12);

delay();

}

return 0;

}

**LED BUTTON**:

#include "stm32f407xx.h"

#define HIGH 1

#define BTN\_PRESSED HIGH

void delay(void)

{

for(uint32\_t i = 0 ; i < 500000/2 ; i ++);

}

int main(void)

{

GPIO\_Handle\_t GpioLed, GPIOBtn;

//this is led gpio configuration

GpioLed.pGPIOx = GPIOD;

GpioLed.GPIO\_PinConfig.GPIO\_PinNumber = GPIO\_PIN\_NO\_12;

GpioLed.GPIO\_PinConfig.GPIO\_PinMode = GPIO\_MODE\_OUT;

GpioLed.GPIO\_PinConfig.GPIO\_PinSpeed = GPIO\_SPEED\_FAST;

GpioLed.GPIO\_PinConfig.GPIO\_PinOPType = GPIO\_OP\_TYPE\_PP;

GpioLed.GPIO\_PinConfig.GPIO\_PinPuPdControl = GPIO\_NO\_PUPD;

GPIO\_PeriClockControl(GPIOD,ENABLE);

GPIO\_Init(&GpioLed);

//this is btn gpio configuration

GPIOBtn.pGPIOx = GPIOA;

GPIOBtn.GPIO\_PinConfig.GPIO\_PinNumber = GPIO\_PIN\_NO\_0;

GPIOBtn.GPIO\_PinConfig.GPIO\_PinMode = GPIO\_MODE\_IN;

GPIOBtn.GPIO\_PinConfig.GPIO\_PinSpeed = GPIO\_SPEED\_FAST;

GPIOBtn.GPIO\_PinConfig.GPIO\_PinPuPdControl = GPIO\_NO\_PUPD;

GPIO\_PeriClockControl(GPIOA,ENABLE);

GPIO\_Init(&GPIOBtn);

while(1)

{

if(GPIO\_ReadFromInputPin(GPIOA,GPIO\_PIN\_NO\_0) == BTN\_PRESSED)

{

//delay();

GPIO\_ToggleOutputPin(GPIOD,GPIO\_PIN\_NO\_12);

}

}

return 0;

}

**LED WITH EXTERNAL BUTTON:**

#include "stm32f407xx.h"

#define HIGH 1

#define LOW 0

#define BTN\_PRESSED LOW

void delay(void)

{

for(uint32\_t i = 0 ; i < 500000/2 ; i ++);

}

int main(void)

{

GPIO\_Handle\_t GpioLed, GPIOBtn;

//this is led gpio configuration

GpioLed.pGPIOx = GPIOA;

GpioLed.GPIO\_PinConfig.GPIO\_PinNumber = GPIO\_PIN\_NO\_8;

GpioLed.GPIO\_PinConfig.GPIO\_PinMode = GPIO\_MODE\_OUT;

GpioLed.GPIO\_PinConfig.GPIO\_PinSpeed = GPIO\_SPEED\_FAST;

GpioLed.GPIO\_PinConfig.GPIO\_PinOPType = GPIO\_OP\_TYPE\_PP;

GpioLed.GPIO\_PinConfig.GPIO\_PinPuPdControl = GPIO\_NO\_PUPD;

GPIO\_PeriClockControl(GPIOA,ENABLE);

GPIO\_Init(&GpioLed);

//this is btn gpio configuration

GPIOBtn.pGPIOx = GPIOB;

GPIOBtn.GPIO\_PinConfig.GPIO\_PinNumber = GPIO\_PIN\_NO\_12;

GPIOBtn.GPIO\_PinConfig.GPIO\_PinMode = GPIO\_MODE\_IN;

GPIOBtn.GPIO\_PinConfig.GPIO\_PinSpeed = GPIO\_SPEED\_FAST;

GPIOBtn.GPIO\_PinConfig.GPIO\_PinPuPdControl = GPIO\_PIN\_PU;

GPIO\_PeriClockControl(GPIOB,ENABLE);

GPIO\_Init(&GPIOBtn);

while(1)

{

if(GPIO\_ReadFromInputPin(GPIOB,GPIO\_PIN\_NO\_12) == BTN\_PRESSED)

{

delay();

GPIO\_ToggleOutputPin(GPIOA,GPIO\_PIN\_NO\_8);

}

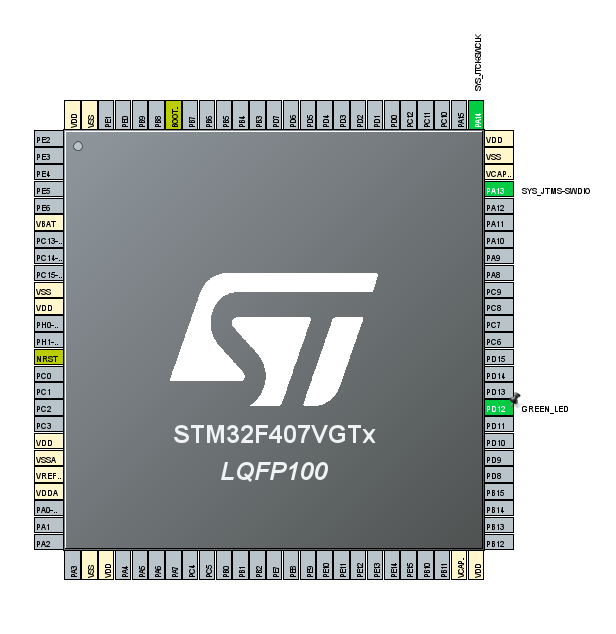
}

return 0;

}

**STM IDE TASKS:**

1. **GPIO INTERFACE USING HAL\_LIBRARY**

****

/\* 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 ---------------------------------------------------------\*/

/\* USER CODE BEGIN PV \*/

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_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();

/\* USER CODE BEGIN 2 \*/

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

HAL\_GPIO\_TogglePin(GREEN\_LED\_GPIO\_Port, GREEN\_LED\_Pin);

HAL\_Delay(500);

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Configure the main internal regulator output voltage

\*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

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

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

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\_HSI;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

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();

}

}

/\*\*

\* @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\_GPIOD\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GREEN\_LED\_GPIO\_Port, GREEN\_LED\_Pin, GPIO\_PIN\_SET);

/\*Configure GPIO pin : GREEN\_LED\_Pin \*/

GPIO\_InitStruct.Pin = GREEN\_LED\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_PULLUP;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_HIGH;

HAL\_GPIO\_Init(GREEN\_LED\_GPIO\_Port, &GPIO\_InitStruct);

}

/\* USER CODE BEGIN 4 \*/

/\* 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 \*/

\_\_disable\_irq();

while (1)

{

}

/\* 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,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

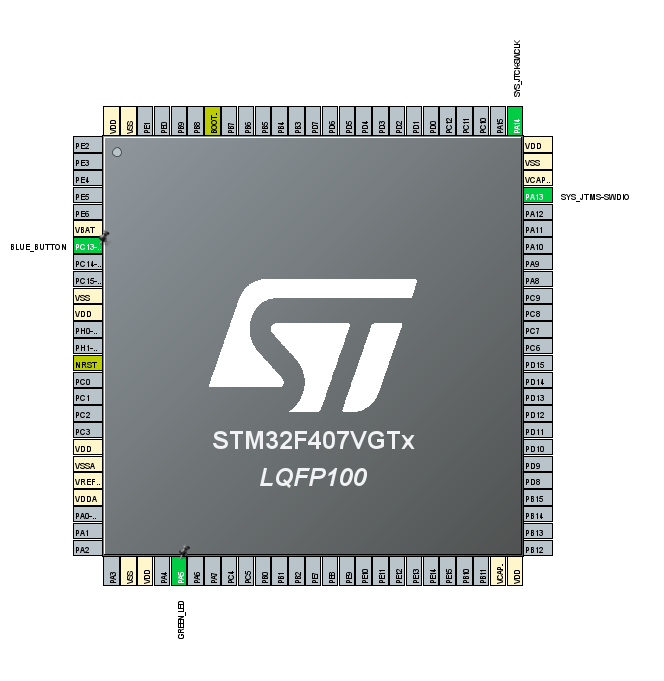
/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/

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

1. **EXTERNAL INTERUPPT USING HAL\_LIBRARY**

****

/\* 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 ---------------------------------------------------------\*/

/\* USER CODE BEGIN PV \*/

uint8\_t flag=0;

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_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();

/\* USER CODE BEGIN 2 \*/

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

if(flag==1)

{

HAL\_GPIO\_TogglePin(GREEN\_LED\_GPIO\_Port,GREEN\_LED\_Pin);

flag=0;

}

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Configure the main internal regulator output voltage

\*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

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

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

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\_HSI;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

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();

}

}

/\*\*

\* @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\_GPIOA\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GREEN\_LED\_GPIO\_Port, GREEN\_LED\_Pin, GPIO\_PIN\_RESET);

/\*Configure GPIO pin : BLUE\_BUTTON\_Pin \*/

GPIO\_InitStruct.Pin = BLUE\_BUTTON\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_IT\_FALLING;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

HAL\_GPIO\_Init(BLUE\_BUTTON\_GPIO\_Port, &GPIO\_InitStruct);

/\*Configure GPIO pin : GREEN\_LED\_Pin \*/

GPIO\_InitStruct.Pin = GREEN\_LED\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GREEN\_LED\_GPIO\_Port, &GPIO\_InitStruct);

/\* EXTI interrupt init\*/

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)

{

flag=1;

}

/\* 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 \*/

\_\_disable\_irq();

while (1)

{

}

/\* 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,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

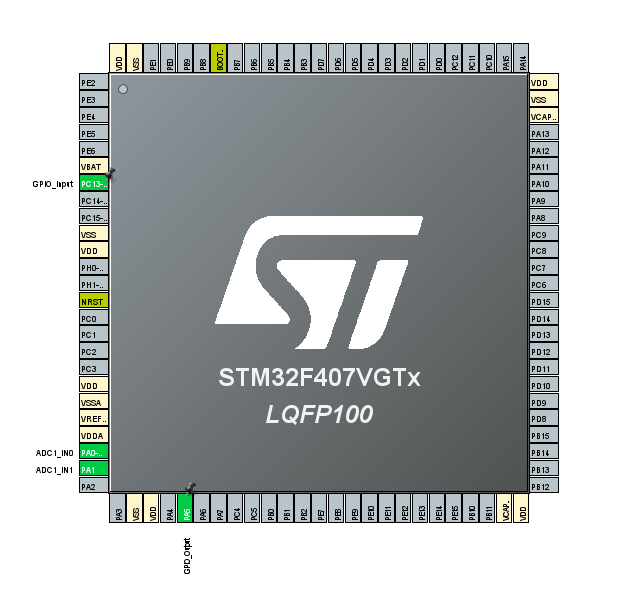
/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/

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

1. **ADC USING HAL\_LIBRARY**

****

/\* 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 ---------------------------------------------------------\*/

ADC\_HandleTypeDef hadc1;

/\* USER CODE BEGIN PV \*/

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_ADC1\_Init(void);

/\* USER CODE BEGIN PFP \*/

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

uint32\_t adcVal0,adcVal1;

/\* USER CODE END 0 \*/

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

ADC\_ChannelConfTypeDef sConfig = {0};

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\_ADC1\_Init();

/\* USER CODE BEGIN 2 \*/

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

sConfig.Channel=ADC\_CHANNEL\_0;

HAL\_ADC\_ConfigChannel(&hadc1, &sConfig);

HAL\_ADC\_Start(&hadc1);

if( HAL\_ADC\_PollForConversion(&hadc1,5)==HAL\_OK)

{

adcVal0=HAL\_ADC\_GetValue(&hadc1);

}

sConfig.Channel=ADC\_CHANNEL\_1;

HAL\_ADC\_ConfigChannel(&hadc1, &sConfig);

HAL\_ADC\_Start(&hadc1);

if( HAL\_ADC\_PollForConversion(&hadc1,5)==HAL\_OK)

{

adcVal1=HAL\_ADC\_GetValue(&hadc1);

}

HAL\_Delay(50);

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Configure the main internal regulator output voltage

\*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

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

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

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\_HSI;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

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();

}

}

/\*\*

\* @brief ADC1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_ADC1\_Init(void)

{

/\* USER CODE BEGIN ADC1\_Init 0 \*/

/\* USER CODE END ADC1\_Init 0 \*/

/\* USER CODE BEGIN ADC1\_Init 1 \*/

/\* USER CODE END ADC1\_Init 1 \*/

/\*\* Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)

\*/

hadc1.Instance = ADC1;

hadc1.Init.ClockPrescaler = ADC\_CLOCK\_SYNC\_PCLK\_DIV2;

hadc1.Init.Resolution = ADC\_RESOLUTION\_8B;

hadc1.Init.ScanConvMode = DISABLE;

hadc1.Init.ContinuousConvMode = DISABLE;

hadc1.Init.DiscontinuousConvMode = DISABLE;

hadc1.Init.ExternalTrigConvEdge = ADC\_EXTERNALTRIGCONVEDGE\_NONE;

hadc1.Init.ExternalTrigConv = ADC\_SOFTWARE\_START;

hadc1.Init.DataAlign = ADC\_DATAALIGN\_RIGHT;

hadc1.Init.NbrOfConversion = 1;

hadc1.Init.DMAContinuousRequests = DISABLE;

hadc1.Init.EOCSelection = ADC\_EOC\_SINGLE\_CONV;

if (HAL\_ADC\_Init(&hadc1) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

\*/

sConfig.Channel = ADC\_CHANNEL\_0;

sConfig.Rank = 1;

sConfig.SamplingTime = ADC\_SAMPLETIME\_28CYCLES;

if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN ADC1\_Init 2 \*/

/\* USER CODE END ADC1\_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\_GPIOA\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_5, GPIO\_PIN\_RESET);

/\*Configure GPIO pin : PC13 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_13;

GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

HAL\_GPIO\_Init(GPIOC, &GPIO\_InitStruct);

/\*Configure GPIO pin : PA5 \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_5;

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);

}

/\* USER CODE BEGIN 4 \*/

/\* 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 \*/

\_\_disable\_irq();

while (1)

{

}

/\* 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,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

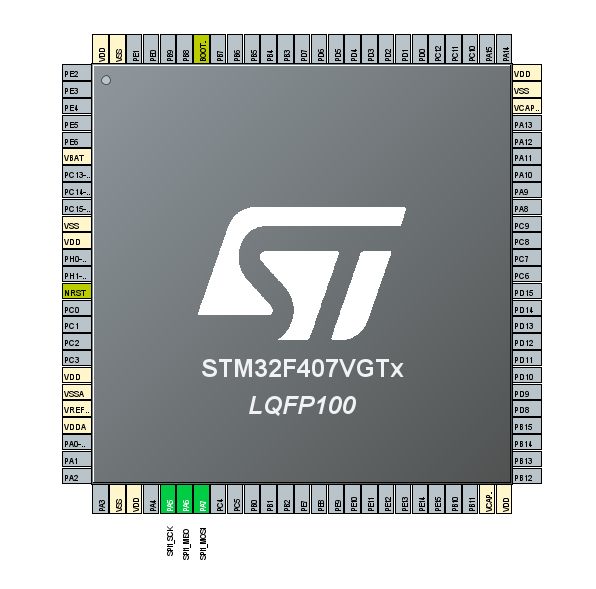
/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/

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

1. **SPI USING HAL\_LIBRARY**



/\* 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 ---------------------------------------------------------\*/

SPI\_HandleTypeDef hspi1;

/\* USER CODE BEGIN PV \*/

uint8\_t buffer\_tx[10]={30,31,32,33,34,35,36,37,38,39};

uint8\_t buffer\_rx[10];

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_SPI1\_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\_SPI1\_Init();

/\* USER CODE BEGIN 2 \*/

HAL\_SPI\_TransmitReceive(&hspi1,buffer\_tx,buffer\_rx,10,100);

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Configure the main internal regulator output voltage

\*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

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

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

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\_HSI;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

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();

}

}

/\*\*

\* @brief SPI1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_SPI1\_Init(void)

{

/\* USER CODE BEGIN SPI1\_Init 0 \*/

/\* USER CODE END SPI1\_Init 0 \*/

/\* USER CODE BEGIN SPI1\_Init 1 \*/

/\* USER CODE END SPI1\_Init 1 \*/

/\* SPI1 parameter configuration\*/

hspi1.Instance = SPI1;

hspi1.Init.Mode = SPI\_MODE\_MASTER;

hspi1.Init.Direction = SPI\_DIRECTION\_2LINES;

hspi1.Init.DataSize = SPI\_DATASIZE\_8BIT;

hspi1.Init.CLKPolarity = SPI\_POLARITY\_LOW;

hspi1.Init.CLKPhase = SPI\_PHASE\_1EDGE;

hspi1.Init.NSS = SPI\_NSS\_SOFT;

hspi1.Init.BaudRatePrescaler = SPI\_BAUDRATEPRESCALER\_8;

hspi1.Init.FirstBit = SPI\_FIRSTBIT\_MSB;

hspi1.Init.TIMode = SPI\_TIMODE\_DISABLE;

hspi1.Init.CRCCalculation = SPI\_CRCCALCULATION\_DISABLE;

hspi1.Init.CRCPolynomial = 10;

if (HAL\_SPI\_Init(&hspi1) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN SPI1\_Init 2 \*/

/\* USER CODE END SPI1\_Init 2 \*/

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_GPIO\_Init(void)

{

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

}

/\* USER CODE BEGIN 4 \*/

/\* 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 \*/

\_\_disable\_irq();

while (1)

{

}

/\* 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,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

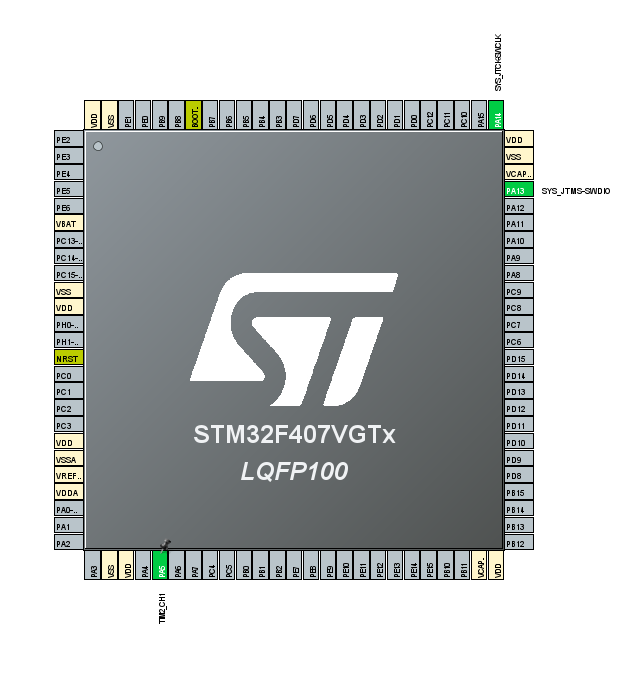
/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/

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

1. **PWM USING HAL\_LIBRARY**

****

/\* 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 ---------------------------------------------------------\*/

TIM\_HandleTypeDef htim2;

/\* USER CODE BEGIN PV \*/

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_TIM2\_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\_TIM2\_Init();

/\* USER CODE BEGIN 2 \*/

HAL\_TIM\_PWM\_Start(&htim2, TIM\_CHANNEL\_1);

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while (1)

{

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Configure the main internal regulator output voltage

\*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

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

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

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\_HSI;

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();

}

}

/\*\*

\* @brief TIM2 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_TIM2\_Init(void)

{

/\* USER CODE BEGIN TIM2\_Init 0 \*/

/\* USER CODE END TIM2\_Init 0 \*/

TIM\_ClockConfigTypeDef sClockSourceConfig = {0};

TIM\_MasterConfigTypeDef sMasterConfig = {0};

TIM\_OC\_InitTypeDef sConfigOC = {0};

/\* USER CODE BEGIN TIM2\_Init 1 \*/

/\* USER CODE END TIM2\_Init 1 \*/

htim2.Instance = TIM2;

htim2.Init.Prescaler = 127;

htim2.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim2.Init.Period = 62499;

htim2.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim2.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

if (HAL\_TIM\_Base\_Init(&htim2) != HAL\_OK)

{

Error\_Handler();

}

sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;

if (HAL\_TIM\_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL\_OK)

{

Error\_Handler();

}

if (HAL\_TIM\_PWM\_Init(&htim2) != HAL\_OK)

{

Error\_Handler();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

if (HAL\_TIMEx\_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL\_OK)

{

Error\_Handler();

}

sConfigOC.OCMode = TIM\_OCMODE\_PWM1;

sConfigOC.Pulse = 0;

sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;

sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;

if (HAL\_TIM\_PWM\_ConfigChannel(&htim2, &sConfigOC, TIM\_CHANNEL\_1) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM2\_Init 2 \*/

/\* USER CODE END TIM2\_Init 2 \*/

HAL\_TIM\_MspPostInit(&htim2);

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_GPIO\_Init(void)

{

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

}

/\* USER CODE BEGIN 4 \*/

/\* 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 \*/

\_\_disable\_irq();

while (1)

{

}

/\* 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,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

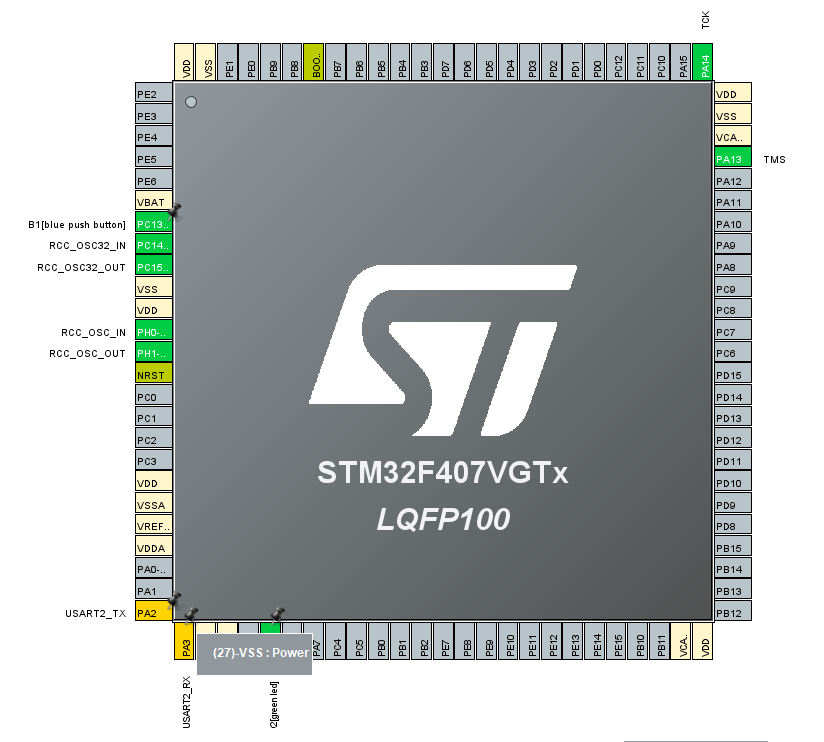
/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/

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

1. **USART USING HAL\_DRIVERS**

****

#include<stdio.h>

#include<string.h>

#include "stm32f407xx.h"

char msg[1024] = "UART Tx testing...\n\r";

USART\_Handle\_t usart2\_handle;

void USART2\_Init(void)

{

usart2\_handle.pUSARTx = USART2;

usart2\_handle.USART\_Config.USART\_Baud = USART\_STD\_BAUD\_115200;

usart2\_handle.USART\_Config.USART\_HWFlowControl = USART\_HW\_FLOW\_CTRL\_NONE;

usart2\_handle.USART\_Config.USART\_Mode = USART\_MODE\_ONLY\_TX;

usart2\_handle.USART\_Config.USART\_NoOfStopBits = USART\_STOPBITS\_1;

usart2\_handle.USART\_Config.USART\_WordLength = USART\_WORDLEN\_8BITS;

usart2\_handle.USART\_Config.USART\_ParityControl = USART\_PARITY\_DISABLE;

USART\_Init(&usart2\_handle);

}

void USART2\_GPIOInit(void)

{

GPIO\_Handle\_t usart\_gpios;

usart\_gpios.pGPIOx = GPIOA;

usart\_gpios.GPIO\_PinConfig.GPIO\_PinMode = GPIO\_MODE\_ALTFN;

usart\_gpios.GPIO\_PinConfig.GPIO\_PinOPType = GPIO\_OP\_TYPE\_PP;

usart\_gpios.GPIO\_PinConfig.GPIO\_PinPuPdControl = GPIO\_PIN\_PU;

usart\_gpios.GPIO\_PinConfig.GPIO\_PinSpeed = GPIO\_SPEED\_FAST;

usart\_gpios.GPIO\_PinConfig.GPIO\_PinAltFunMode =7;

//USART2 TX

usart\_gpios.GPIO\_PinConfig.GPIO\_PinNumber = GPIO\_PIN\_NO\_2;

GPIO\_Init(&usart\_gpios);

//USART2 RX

usart\_gpios.GPIO\_PinConfig.GPIO\_PinNumber = GPIO\_PIN\_NO\_3;

GPIO\_Init(&usart\_gpios);

}

void GPIO\_ButtonInit(void)

{

GPIO\_Handle\_t GPIOBtn,GpioLed;

//this is btn gpio configuration

GPIOBtn.pGPIOx = GPIOA;

GPIOBtn.GPIO\_PinConfig.GPIO\_PinNumber = GPIO\_PIN\_NO\_0;

GPIOBtn.GPIO\_PinConfig.GPIO\_PinMode = GPIO\_MODE\_IN;

GPIOBtn.GPIO\_PinConfig.GPIO\_PinSpeed = GPIO\_SPEED\_FAST;

GPIOBtn.GPIO\_PinConfig.GPIO\_PinPuPdControl = GPIO\_NO\_PUPD;

GPIO\_Init(&GPIOBtn);

//this is led gpio configuration

GpioLed.pGPIOx = GPIOD;

GpioLed.GPIO\_PinConfig.GPIO\_PinNumber = GPIO\_PIN\_NO\_12;

GpioLed.GPIO\_PinConfig.GPIO\_PinMode = GPIO\_MODE\_OUT;

GpioLed.GPIO\_PinConfig.GPIO\_PinSpeed = GPIO\_SPEED\_FAST;

GpioLed.GPIO\_PinConfig.GPIO\_PinOPType = GPIO\_OP\_TYPE\_OD;

GpioLed.GPIO\_PinConfig.GPIO\_PinPuPdControl = GPIO\_NO\_PUPD;

GPIO\_PeriClockControl(GPIOD,ENABLE);

GPIO\_Init(&GpioLed);

}

void delay(void)

{

for(uint32\_t i = 0 ; i < 500000/2 ; i ++);

}

int main(void)

{

GPIO\_ButtonInit();

USART2\_GPIOInit();

USART2\_Init();

USART\_PeripheralControl(USART2,ENABLE);

while(1)

{

//wait till button is pressed

while( ! GPIO\_ReadFromInputPin(GPIOA,GPIO\_PIN\_NO\_0) );

//to avoid button de-bouncing related issues 200ms of delay

delay();

USART\_SendData(&usart2\_handle,(uint8\_t\*)msg,strlen(msg));

}

return 0;

}

**SERIAL PERIPHERAL INTERFACE(SPI):**

The SPI interface provides two main functions, supporting either the SPI protocol or the I2Saudio protocol. By default, it is the SPI function that is selected. It is possible to switch the interface from SPI to I2S by software.

The serial peripheral interface (SPI) allows half/ full-duplex, synchronous, serial

communication with external devices. The interface can be configured as the master and in this case it provides the communication clock (SCK) to the external slave device. The

interface is also capable of operating in multi-master configuration.

It may be used for a variety of purposes, including simplex synchronous transfers on two

lines with a possible bidirectional data line or reliable communication using CRC checking.

The I2S is also a synchronous serial communication interface. It can address four different audio standards including the I2S Philips standard, the MSB- and LSB-justified standards, and the PCM standard. It can operate as a slave or a master device in full-duplex mode (using 4 pins) or in half-duplex mode (using 3 pins). Master clock can be provided by the interface to an external slave component when the I2S is configured as the communication

master.

**SPI features**

• Full-duplex synchronous transfers on three lines

• Simplex synchronous transfers on two lines with or without a bidirectional data line

• 8- or 16-bit transfer frame format selection

• Master or slave operation

• Multi-master mode capability

• 8 master mode baud rate pre-scalers (fPCLK/2 max.)

• Slave mode frequency (fPCLK/2 max)

• Faster communication for both master and slave

• NSS management by hardware or software for both master and slave: dynamic change

of master/slave operations

• Programmable clock polarity and phase

• Programmable data order with MSB-first or LSB-first shifting

• Dedicated transmission and reception flags with interrupt capability

• SPI bus busy status flag

• SPI TI mode

• Hardware CRC feature for reliable communication:

– CRC value can be transmitted as last byte in Tx mode

– Automatic CRC error checking for last received byte

• Master mode fault, overrun and CRC error flags with interrupt capability

• 1-byte transmission and reception buffer with DMA capability: Tx and Rx requests

**Usually, the SPI is connected to external devices through four pins**:

• MISO: Master In / Slave Out data. This pin can be used to transmit data in slave mode

and receive data in master mode.

• MOSI: Master Out / Slave In data. This pin can be used to transmit data in master

mode and receive data in slave mode.

• SCK: Serial Clock output for SPI masters and input for SPI slaves.

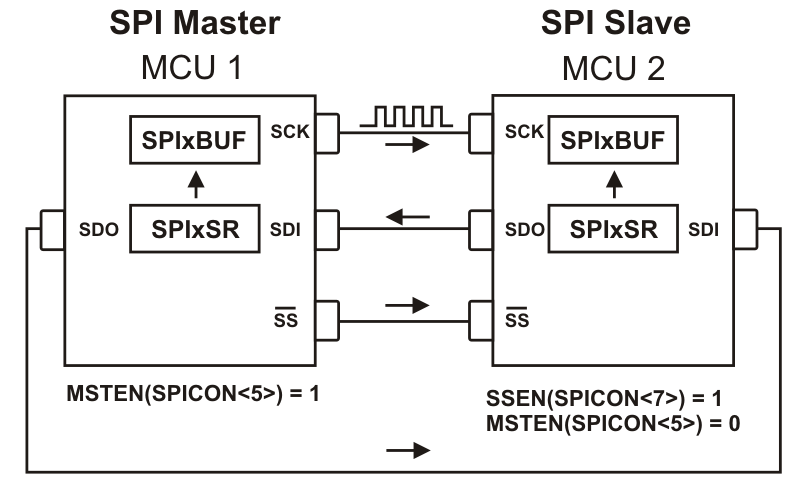
• NSS: Slave select. This is an optional pin to select a slave device. This pin acts as a

‘chip select’ to let the SPI master communicate with slaves individually and to avoid

contention on the data lines. Slave NSS inputs can be driven by standard IO ports on

the master device. The NSS pin may also be used as an output if enabled (SSOE bit)

and driven low if the SPI is in master configuration.



**SPI DRIVERS SPECIFIC:**

typedef struct

{

uint8\_t SPI\_DeviceMode;

uint8\_t SPI\_BusConfig;

uint8\_t SPI\_SclkSpeed;

uint8\_t SPI\_DFF;

uint8\_t SPI\_CPOL;

uint8\_t SPI\_CPHA;

uint8\_t SPI\_SSM;

}SPI\_Config\_t;

**/\***

**\*Handle structure for SPIx peripheral**

**\*/**

typedef struct

{

SPI\_RegDef\_t \*pSPIx; /\*!< This holds the base address of SPIx(x:0,1,2) peripheral >\*/

SPI\_Config\_t SPIConfig;

uint8\_t \*pTxBuffer; /\* !< To store the app. Tx buffer address > \*/

uint8\_t \*pRxBuffer; /\* !< To store the app. Rx buffer address > \*/

uint32\_t TxLen; /\* !< To store Tx len > \*/

uint32\_t RxLen; /\* !< To store Tx len > \*/

uint8\_t TxState; /\* !< To store Tx state > \*/

uint8\_t RxState; /\* !< To store Rx state > \*/

}SPI\_Handle\_t;

/\*

\* SPI application states

\*/

#define SPI\_READY 0

#define SPI\_BUSY\_IN\_RX 1

#define SPI\_BUSY\_IN\_TX 2

**/\***

**\* Possible SPI Application events**

**\*/**

#define SPI\_EVENT\_TX\_CMPLT 1

#define SPI\_EVENT\_RX\_CMPLT 2

#define SPI\_EVENT\_OVR\_ERR 3

#define SPI\_EVENT\_CRC\_ERR 4

**/\***

**\* @SPI\_DeviceMode**

**\*/**

#define SPI\_DEVICE\_MODE\_MASTER 1

#define SPI\_DEVICE\_MODE\_SLAVE 0

**/\***

**\* @SPI\_BusConfig**

**\*/**

#define SPI\_BUS\_CONFIG\_FD 1

#define SPI\_BUS\_CONFIG\_HD 2

#define SPI\_BUS\_CONFIG\_SIMPLEX\_RXONLY 3

/\*

\* @SPI\_SclkSpeed

\*/

#define SPI\_SCLK\_SPEED\_DIV2 0

#define SPI\_SCLK\_SPEED\_DIV4 1

#define SPI\_SCLK\_SPEED\_DIV8 2

#define SPI\_SCLK\_SPEED\_DIV16 3

#define SPI\_SCLK\_SPEED\_DIV32 4

#define SPI\_SCLK\_SPEED\_DIV64 5

#define SPI\_SCLK\_SPEED\_DIV128 6

#define SPI\_SCLK\_SPEED\_DIV256 7

**/\***

**\* @SPI\_DFF**

**\*/**

#define SPI\_DFF\_8BITS 0

#define SPI\_DFF\_16BITS 1

**/\***

**\* @CPOL**

**\*/**

#define SPI\_CPOL\_HIGH 1

#define SPI\_CPOL\_LOW 0

**/\***

**\* @CPHA**

**\*/**

#define SPI\_CPHA\_HIGH 1

#define SPI\_CPHA\_LOW 0

**/\***

**\* @SPI\_SSM**

**\*/**

#define SPI\_SSM\_EN 1

#define SPI\_SSM\_DI 0

**/\***

**\* SPI related status flags definitions**

**\*/**

#define SPI\_TXE\_FLAG ( 1 << SPI\_SR\_TXE)

#define SPI\_RXNE\_FLAG ( 1 << SPI\_SR\_RXNE)

#define SPI\_BUSY\_FLAG ( 1 << SPI\_SR\_BSY)

**/\***

**\* Peripheral Clock setup**

**\*/**

void SPI\_PeriClockControl(SPI\_RegDef\_t \*pSPIx, uint8\_t EnorDi);

**/\***

**\* Init and De-init**

**\*/**

void SPI\_Init(SPI\_Handle\_t \*pSPIHandle);

void SPI\_DeInit(SPI\_RegDef\_t \*pSPIx);

**/\***

**\* Data Send and Receive**

**\*/**

void SPI\_SendData(SPI\_RegDef\_t \*pSPIx,uint8\_t \*pTxBuffer, uint32\_t Len);

void SPI\_ReceiveData(SPI\_RegDef\_t \*pSPIx, uint8\_t \*pRxBuffer, uint32\_t Len);

uint8\_t SPI\_SendDataIT(SPI\_Handle\_t \*pSPIHandle,uint8\_t \*pTxBuffer, uint32\_t Len);

uint8\_t SPI\_ReceiveDataIT(SPI\_Handle\_t \*pSPIHandle, uint8\_t \*pRxBuffer, uint32\_t Len);

**/\***

**\* IRQ Configuration and ISR handling**

**\*/**

void SPI\_IRQInterruptConfig(uint8\_t IRQNumber, uint8\_t EnorDi);

void SPI\_IRQPriorityConfig(uint8\_t IRQNumber, uint32\_t IRQPriority);

void SPI\_IRQHandling(SPI\_Handle\_t \*pHandle);

**/\***

**\* Other Peripheral Control APIs**

**\*/**

void SPI\_PeripheralControl(SPI\_RegDef\_t \*pSPIx, uint8\_t EnOrDi);

void SPI\_SSIConfig(SPI\_RegDef\_t \*pSPIx, uint8\_t EnOrDi);

void SPI\_SSOEConfig(SPI\_RegDef\_t \*pSPIx, uint8\_t EnOrDi);

uint8\_t SPI\_GetFlagStatus(SPI\_RegDef\_t \*pSPIx , uint32\_t FlagName);

void SPI\_ClearOVRFlag(SPI\_RegDef\_t \*pSPIx);

void SPI\_CloseTransmisson(SPI\_Handle\_t \*pSPIHandle);

void SPI\_CloseReception(SPI\_Handle\_t \*pSPIHandle);

uint8\_t I2C\_DeviceMode(I2C\_RegDef\_t \*I2Cx);

**/\***

**\* Application callback**

**\*/**

void SPI\_ApplicationEventCallback(SPI\_Handle\_t \*pSPIHandle,uint8\_t AppEv);

#endif **/\* INC\_STM32F407XX\_SPI\_DRIVER\_H\_ \*/**

**DRIVERS MAIN FILE:**

static void spi\_txe\_interrupt\_handle(SPI\_Handle\_t \*pSPIHandle);

static void spi\_rxne\_interrupt\_handle(SPI\_Handle\_t \*pSPIHandle);

static void spi\_ovr\_err\_interrupt\_handle(SPI\_Handle\_t \*pSPIHandle);

void SPI\_PeriClockControl(SPI\_RegDef\_t \*pSPIx, uint8\_t EnorDi)

{

if(EnorDi == ENABLE)

{

if(pSPIx == SPI1)

{

SPI1\_PCLK\_EN();

}else if (pSPIx == SPI2)

{

SPI2\_PCLK\_EN();

}else if (pSPIx == SPI3)

{

SPI3\_PCLK\_EN();

}

}

else

{

//TODO

}

}

void SPI\_Init(SPI\_Handle\_t \*pSPIHandle)

{

**//peripheral clock enable**

SPI\_PeriClockControl(pSPIHandle->pSPIx, ENABLE);

//first lets configure the SPI\_CR1 register

uint32\_t tempreg = 0;

**//1. configure the device mode**

tempreg |= pSPIHandle->SPIConfig.SPI\_DeviceMode << SPI\_CR1\_MSTR ;

**//2. Configure the bus config**

if(pSPIHandle->SPIConfig.SPI\_BusConfig == SPI\_BUS\_CONFIG\_FD)

{

**//bidi mode should be cleared**

tempreg &= ~( 1 << SPI\_CR1\_BIDIMODE);

}else if (pSPIHandle->SPIConfig.SPI\_BusConfig == SPI\_BUS\_CONFIG\_HD)

{

//bidi mode should be set

tempreg |= ( 1 << SPI\_CR1\_BIDIMODE);

}else if (pSPIHandle->SPIConfig.SPI\_BusConfig == SPI\_BUS\_CONFIG\_SIMPLEX\_RXONLY)

{

//BIDI mode should be cleared

tempreg &= ~( 1 << SPI\_CR1\_BIDIMODE);

//RXONLY bit must be set

tempreg |= ( 1 << SPI\_CR1\_RXONLY);

}

// 3. Configure the spi serial clock speed (baud rate)

tempreg |= pSPIHandle->SPIConfig.SPI\_SclkSpeed << SPI\_CR1\_BR;

//4. Configure the DFF

tempreg |= pSPIHandle->SPIConfig.SPI\_DFF << SPI\_CR1\_DFF;

//5. configure the CPOL

tempreg |= pSPIHandle->SPIConfig.SPI\_CPOL << SPI\_CR1\_CPOL;

//6 . configure the CPHA

tempreg |= pSPIHandle->SPIConfig.SPI\_CPHA << SPI\_CR1\_CPHA;

tempreg |= pSPIHandle->SPIConfig.SPI\_SSM << SPI\_CR1\_SSM;

pSPIHandle->pSPIx->CR1 = tempreg;

}

void SPI\_DeInit(SPI\_RegDef\_t \*pSPIx)

{

//todo

}

uint8\_t SPI\_GetFlagStatus(SPI\_RegDef\_t \*pSPIx , uint32\_t FlagName)

{

if(pSPIx->SR & FlagName)

{

return FLAG\_SET;

}

return FLAG\_RESET;

}

void SPI\_SendData(SPI\_RegDef\_t \*pSPIx,uint8\_t \*pTxBuffer, uint32\_t Len)

{

while(Len > 0)

{

//1. wait until TXE is set

while(SPI\_GetFlagStatus(pSPIx,SPI\_TXE\_FLAG) == FLAG\_RESET );

//2. check the DFF bit in CR1

if( (pSPIx->CR1 & ( 1 << SPI\_CR1\_DFF) ) )

{

//16 bit DFF

//1. load the data in to the DR

pSPIx->DR = \*((uint16\_t\*)pTxBuffer);

Len--;

Len--;

(uint16\_t\*)pTxBuffer++;

}else

{

//8 bit DFF

pSPIx->DR = \*pTxBuffer;

Len--;

pTxBuffer++;

}

}

}

void SPI\_ReceiveData(SPI\_RegDef\_t \*pSPIx, uint8\_t \*pRxBuffer, uint32\_t Len)

{

while(Len > 0)

{

//1. wait until RXNE is set

while(SPI\_GetFlagStatus(pSPIx,SPI\_RXNE\_FLAG) == (uint8\_t)FLAG\_RESET );

//2. check the DFF bit in CR1

if( (pSPIx->CR1 & ( 1 << SPI\_CR1\_DFF) ) )

{

//16 bit DFF

//1. load the data from DR to Rxbuffer address

\*((uint16\_t\*)pRxBuffer) = pSPIx->DR ;

Len--;

Len--;

(uint16\_t\*)pRxBuffer++;

}else

{

//8 bit DFF

\*(pRxBuffer) = pSPIx->DR ;

Len--;

pRxBuffer++;

}

}

}

void SPI\_PeripheralControl(SPI\_RegDef\_t \*pSPIx, uint8\_t EnOrDi)

{

if(EnOrDi == ENABLE)

{

pSPIx->CR1 |= (1 << SPI\_CR1\_SPE);

}else

{

pSPIx->CR1 &= ~(1 << SPI\_CR1\_SPE);

}

}

void SPI\_SSIConfig(SPI\_RegDef\_t \*pSPIx, uint8\_t EnOrDi)

{

if(EnOrDi == ENABLE)

{

pSPIx->CR1 |= (1 << SPI\_CR1\_SSI);

}else

{

pSPIx->CR1 &= ~(1 << SPI\_CR1\_SSI);

}

}

void SPI\_SSOEConfig(SPI\_RegDef\_t \*pSPIx, uint8\_t EnOrDi)

{

if(EnOrDi == ENABLE)

{

pSPIx->CR2 |= (1 << SPI\_CR2\_SSOE);

}else

{

pSPIx->CR2 &= ~(1 << SPI\_CR2\_SSOE);

}

}

void SPI\_IRQInterruptConfig(uint8\_t IRQNumber, uint8\_t EnorDi)

{

if(EnorDi == ENABLE)

{

if(IRQNumber <= 31)

{

//program ISER0 register

\*NVIC\_ISER0 |= ( 1 << IRQNumber );

}else if(IRQNumber > 31 && IRQNumber < 64 ) //32 to 63

{

//program ISER1 register

\*NVIC\_ISER1 |= ( 1 << (IRQNumber % 32) );

}

else if(IRQNumber >= 64 && IRQNumber < 96 )

{

//program ISER2 register //64 to 95

\*NVIC\_ISER3 |= ( 1 << (IRQNumber % 64) );

}

}else

{

if(IRQNumber <= 31)

{

//program ICER0 register

\*NVIC\_ICER0 |= ( 1 << IRQNumber );

}else if(IRQNumber > 31 && IRQNumber < 64 )

{

//program ICER1 register

\*NVIC\_ICER1 |= ( 1 << (IRQNumber % 32) );

}

else if(IRQNumber >= 6 && IRQNumber < 96 )

{

//program ICER2 register

\*NVIC\_ICER3 |= ( 1 << (IRQNumber % 64) );

}

}

}

void SPI\_IRQPriorityConfig(uint8\_t IRQNumber,uint32\_t IRQPriority)

{

//1. first lets find out the ipr register

uint8\_t iprx = IRQNumber / 4;

uint8\_t iprx\_section = IRQNumber %4 ;

uint8\_t shift\_amount = ( 8 \* iprx\_section) + ( 8 - NO\_PR\_BITS\_IMPLEMENTED) ;

\*( NVIC\_PR\_BASE\_ADDR + iprx ) |= ( IRQPriority << shift\_amount );

}

uint8\_t SPI\_SendDataIT(SPI\_Handle\_t \*pSPIHandle,uint8\_t \*pTxBuffer, uint32\_t Len)

{

uint8\_t state = pSPIHandle->TxState;

if(state != SPI\_BUSY\_IN\_TX)

{

//1 . Save the Tx buffer address and Len information in some global variables

pSPIHandle->pTxBuffer = pTxBuffer;

pSPIHandle->TxLen = Len;

//2. Mark the SPI state as busy in transmission so that

// no other code can take over same SPI peripheral until transmission is over

pSPIHandle->TxState = SPI\_BUSY\_IN\_TX;

//3. Enable the TXEIE control bit to get interrupt whenever TXE flag is set in SR

pSPIHandle->pSPIx->CR2 |= ( 1 << SPI\_CR2\_TXEIE );

}

return state;

}

uint8\_t SPI\_ReceiveDataIT(SPI\_Handle\_t \*pSPIHandle, uint8\_t \*pRxBuffer, uint32\_t Len)

{

uint8\_t state = pSPIHandle->RxState;

if(state != SPI\_BUSY\_IN\_RX)

{

//1 . Save the Rx buffer address and Len information in some global variables

pSPIHandle->pRxBuffer = pRxBuffer;

pSPIHandle->RxLen = Len;

//2. Mark the SPI state as busy in reception so that

// no other code can take over same SPI peripheral until reception is over

pSPIHandle->RxState = SPI\_BUSY\_IN\_RX;

//3. Enable the RXNEIE control bit to get interrupt whenever RXNEIE flag is set in SR

pSPIHandle->pSPIx->CR2 |= ( 1 << SPI\_CR2\_RXNEIE );

}

return state;

}

void SPI\_IRQHandling(SPI\_Handle\_t \*pHandle)

{

uint8\_t temp1 , temp2;

//first lets check for TXE

temp1 = pHandle->pSPIx->SR & ( 1 << SPI\_SR\_TXE);

temp2 = pHandle->pSPIx->CR2 & ( 1 << SPI\_CR2\_TXEIE);

if( temp1 && temp2)

{

//handle TXE

spi\_txe\_interrupt\_handle(pHandle);

}

// check for RXNE

temp1 = pHandle->pSPIx->SR & ( 1 << SPI\_SR\_RXNE);

temp2 = pHandle->pSPIx->CR2 & ( 1 << SPI\_CR2\_RXNEIE);

if( temp1 && temp2)

{

//handle RXNE

spi\_rxne\_interrupt\_handle(pHandle);

}

// check for ovr flag

temp1 = pHandle->pSPIx->SR & ( 1 << SPI\_SR\_OVR);

temp2 = pHandle->pSPIx->CR2 & ( 1 << SPI\_CR2\_ERRIE);

if( temp1 && temp2)

{

//handle ovr error

spi\_ovr\_err\_interrupt\_handle(pHandle);

}

}

//some helper function implementations

static void spi\_txe\_interrupt\_handle(SPI\_Handle\_t \*pSPIHandle)

{

// check the DFF bit in CR1

if( (pSPIHandle->pSPIx->CR1 & ( 1 << SPI\_CR1\_DFF) ) )

{

//16 bit DFF

//1. load the data in to the DR

pSPIHandle->pSPIx->DR = \*((uint16\_t\*)pSPIHandle->pTxBuffer);

pSPIHandle->TxLen--;

pSPIHandle->TxLen--;

(uint16\_t\*)pSPIHandle->pTxBuffer++;

}else

{

//8 bit DFF

pSPIHandle->pSPIx->DR = \*pSPIHandle->pTxBuffer;

pSPIHandle->TxLen--;

pSPIHandle->pTxBuffer++;

}

if(! pSPIHandle->TxLen)

{

//TxLen is zero , so close the spi transmission and inform the application that

//TX is over.

//this prevents interrupts from setting up of TXE flag

SPI\_CloseTransmisson(pSPIHandle);

SPI\_ApplicationEventCallback(pSPIHandle,SPI\_EVENT\_TX\_CMPLT);

}

}

static void spi\_rxne\_interrupt\_handle(SPI\_Handle\_t \*pSPIHandle)

{

//do rxing as per the dff

if(pSPIHandle->pSPIx->CR1 & ( 1 << 11))

{

//16 bit

\*((uint16\_t\*)pSPIHandle->pRxBuffer) = (uint16\_t) pSPIHandle->pSPIx->DR;

pSPIHandle->RxLen -= 2;

pSPIHandle->pRxBuffer++;

pSPIHandle->pRxBuffer++;

}

else

{

//8 bit

\*(pSPIHandle->pRxBuffer) = (uint8\_t) pSPIHandle->pSPIx->DR;

pSPIHandle->RxLen--;

pSPIHandle->pRxBuffer++;

}

if(! pSPIHandle->RxLen)

{

//reception is complete

SPI\_CloseReception(pSPIHandle);

SPI\_ApplicationEventCallback(pSPIHandle,SPI\_EVENT\_RX\_CMPLT);

}

}

static void spi\_ovr\_err\_interrupt\_handle(SPI\_Handle\_t \*pSPIHandle)

{

uint8\_t temp;

//1. clear the ovr flag

if(pSPIHandle->TxState != SPI\_BUSY\_IN\_TX)

{

temp = pSPIHandle->pSPIx->DR;

temp = pSPIHandle->pSPIx->SR;

}

(void)temp;

//2. inform the application

SPI\_ApplicationEventCallback(pSPIHandle,SPI\_EVENT\_OVR\_ERR);

}

void SPI\_CloseTransmisson(SPI\_Handle\_t \*pSPIHandle)

{

pSPIHandle->pSPIx->CR2 &= ~( 1 << SPI\_CR2\_TXEIE);

pSPIHandle->pTxBuffer = NULL;

pSPIHandle->TxLen = 0;

pSPIHandle->TxState = SPI\_READY;

}

void SPI\_CloseReception(SPI\_Handle\_t \*pSPIHandle)

{

pSPIHandle->pSPIx->CR2 &= ~( 1 << SPI\_CR2\_RXNEIE);

pSPIHandle->pRxBuffer = NULL;

pSPIHandle->RxLen = 0;

pSPIHandle->RxState = SPI\_READY;

}

void SPI\_ClearOVRFlag(SPI\_RegDef\_t \*pSPIx)

{

uint8\_t temp;

temp = pSPIx->DR;

temp = pSPIx->SR;

(void)temp;

}

\_\_weak void SPI\_ApplicationEventCallback(SPI\_Handle\_t \*pSPIHandle,uint8\_t AppEv)

{

}

**SPI\_TESTING:**

#include<string.h>

#include "stm32f407xx.h"

/\*

\* PB14 --> SPI2\_MISO

\* PB15 --> SPI2\_MOSI

\* PB13 -> SPI2\_SCLK

\* PB12 --> SPI2\_NSS

\* ALT function mode : 5

\*/

void SPI2\_GPIOInits(void)

{

GPIO\_Handle\_t SPIPins;

SPIPins.pGPIOx = GPIOB;

SPIPins.GPIO\_PinConfig.GPIO\_PinMode = GPIO\_MODE\_ALTFN;

SPIPins.GPIO\_PinConfig.GPIO\_PinAltFunMode = 5;

SPIPins.GPIO\_PinConfig.GPIO\_PinOPType = GPIO\_OP\_TYPE\_PP;

SPIPins.GPIO\_PinConfig.GPIO\_PinPuPdControl = GPIO\_NO\_PUPD;

SPIPins.GPIO\_PinConfig.GPIO\_PinSpeed = GPIO\_SPEED\_FAST;

//SCLK

SPIPins.GPIO\_PinConfig.GPIO\_PinNumber = GPIO\_PIN\_NO\_13;

GPIO\_Init(&SPIPins);

//MOSI

SPIPins.GPIO\_PinConfig.GPIO\_PinNumber = GPIO\_PIN\_NO\_15;

GPIO\_Init(&SPIPins);

//MISO//

SPIPins.GPIO\_PinConfig.GPIO\_PinNumber = GPIO\_PIN\_NO\_14;

GPIO\_Init(&SPIPins);

//NSS//

SPIPins.GPIO\_PinConfig.GPIO\_PinNumber = GPIO\_PIN\_NO\_12;

GPIO\_Init(&SPIPins);

}

void SPI2\_Inits(void)

{

SPI\_Handle\_t SPI2handle;

SPI2handle.pSPIx = SPI2;

SPI2handle.SPIConfig.SPI\_BusConfig = SPI\_BUS\_CONFIG\_FD;

SPI2handle.SPIConfig.SPI\_DeviceMode = SPI\_DEVICE\_MODE\_MASTER;

SPI2handle.SPIConfig.SPI\_SclkSpeed = SPI\_SCLK\_SPEED\_DIV2;

//generates sclk of 8MHz

SPI2handle.SPIConfig.SPI\_DFF = SPI\_DFF\_8BITS;

SPI2handle.SPIConfig.SPI\_CPOL = SPI\_CPOL\_HIGH;

SPI2handle.SPIConfig.SPI\_CPHA = SPI\_CPHA\_LOW;

SPI2handle.SPIConfig.SPI\_SSM = SPI\_SSM\_EN; //software slave management enabled for NSS pin

SPI\_Init(&SPI2handle);

}

int main(void)

{

char user\_data[] = "Hello world";

//this function is used to initialize the GPIO pins to behave as SPI2 pins

SPI2\_GPIOInits();

//This function is used to initialize the SPI2 peripheral parameters

SPI2\_Inits();

//this makes NSS signal internally high and avoids MODF error

SPI\_SSIConfig(SPI2,ENABLE);

//enable the SPI2 peripheral

SPI\_PeripheralControl(SPI2,ENABLE);

//to send data

SPI\_SendData(SPI2,(uint8\_t\*)user\_data,strlen(user\_data));

//lets confirm SPI is not busy

while( SPI\_GetFlagStatus(SPI2,SPI\_BUSY\_FLAG) );

//Disable the SPI2 peripheral

SPI\_PeripheralControl(SPI2,DISABLE);

while(1);

return 0;

}