Learning Report – Embedded C – Hardware + Programming + Testing

LTTS
GLOBAL
ENGINEERING
ACADEMY

L&T Technology Services

Details

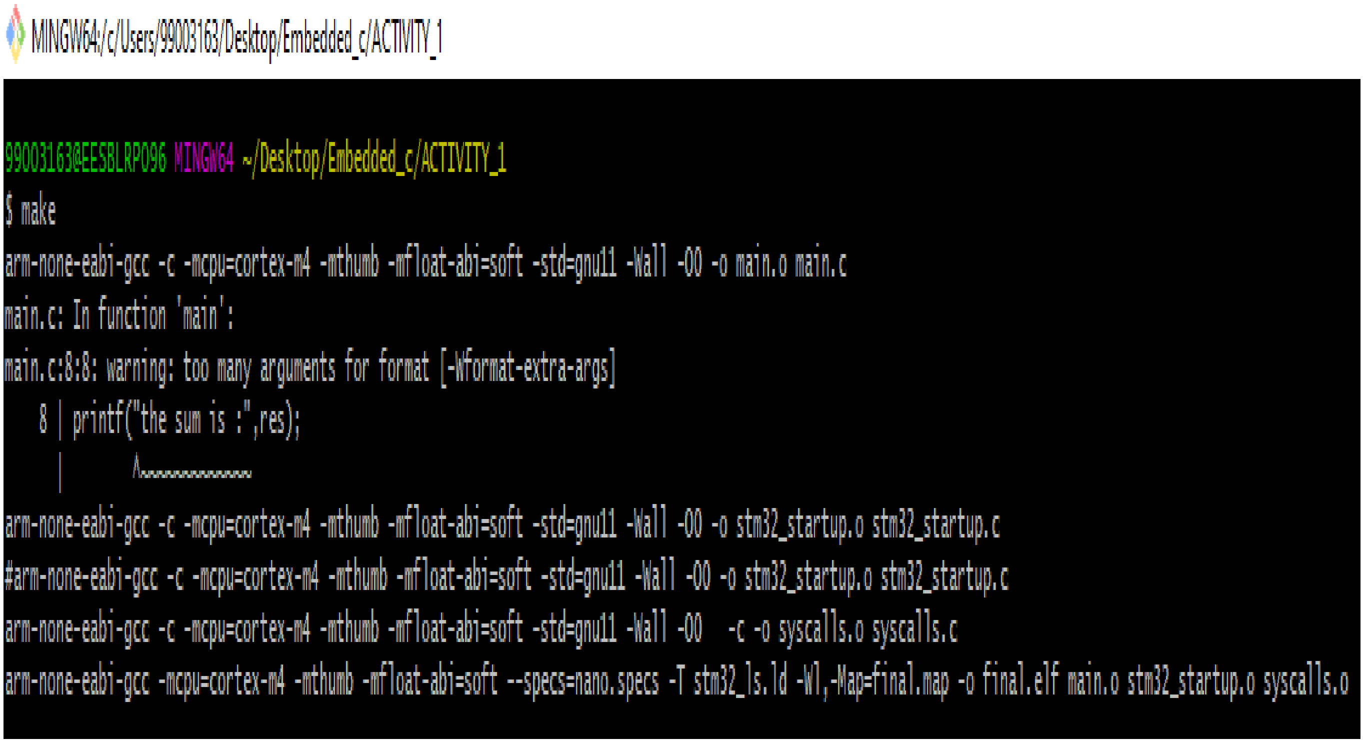
Ver. Rel. No.	Release Date	Prepared. By	Reviewed By	To be Approved	Remarks/Revision Details
1.0	29/12/2020	SHREYA KORI			
1.1	03/01/2021	SHREYA KORI			

Table of Contents

1. ACTIVITY 1: BUILD PROCESS

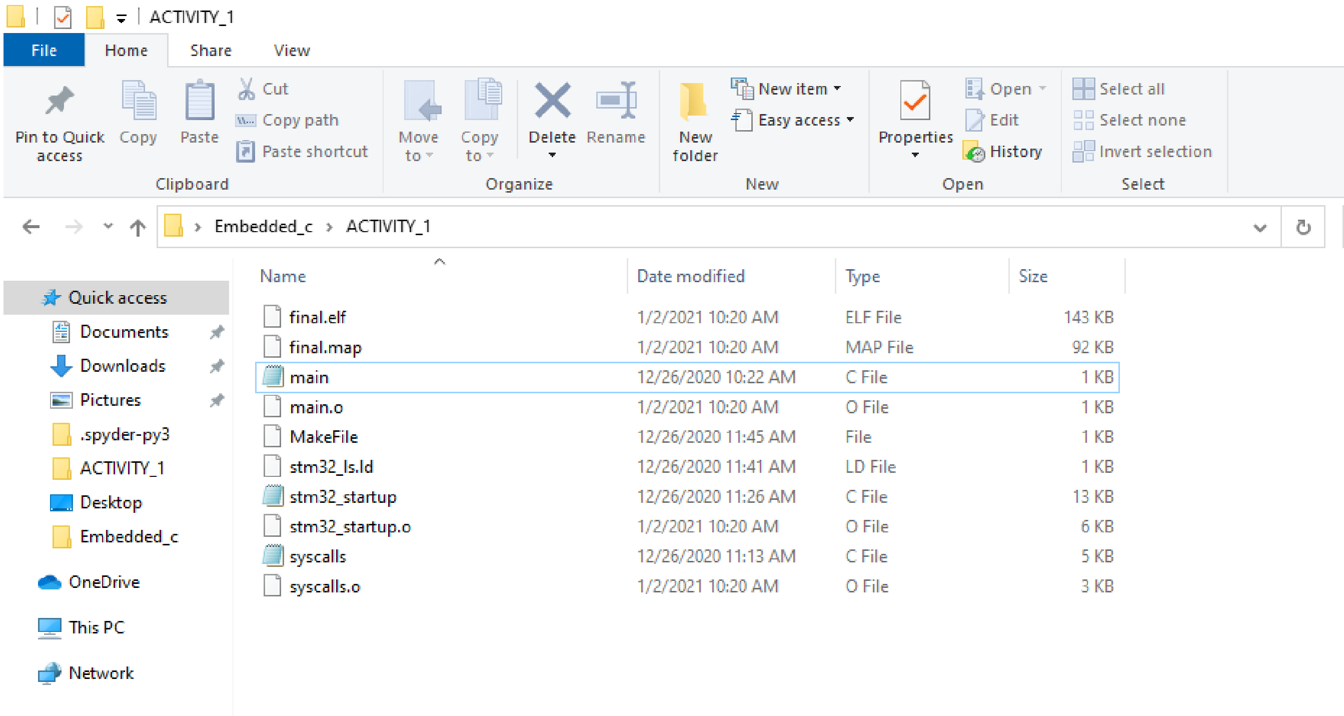
1.1. main.c

```
#include<stdio.h>
int main()
{
int num1=5;
int num2=10;
int addition;
addition = num1+num2;
printf("the sum is :",addition);
return 0;
}
```



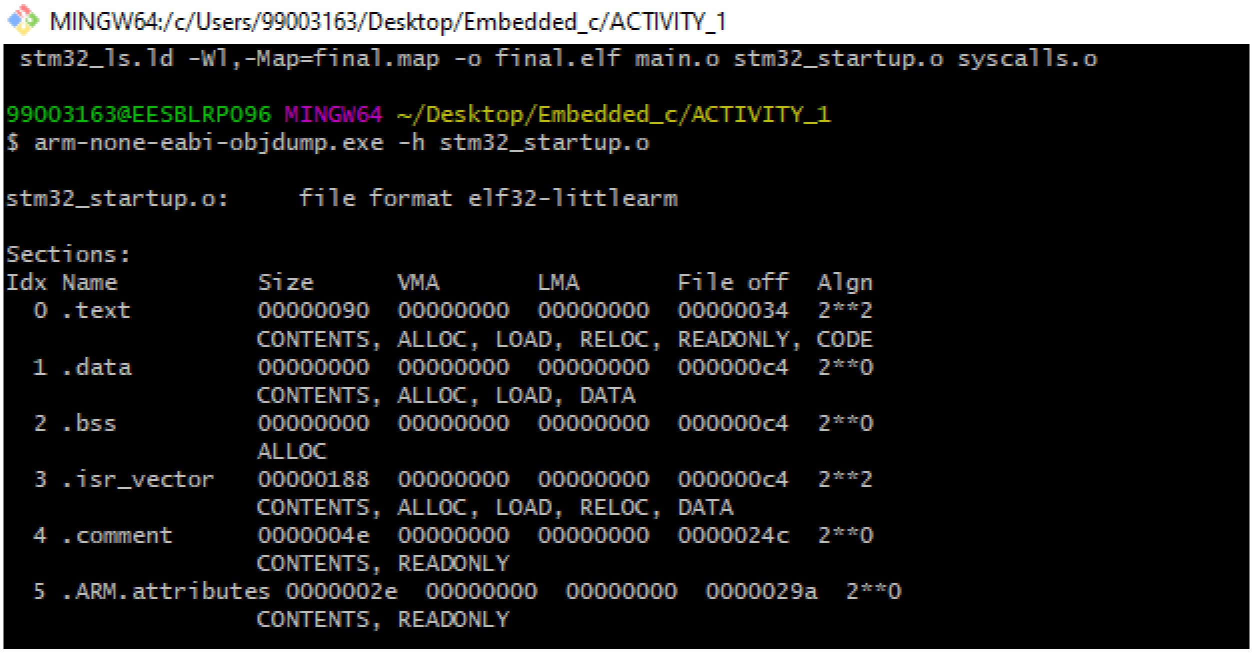
```
MINGW64:/c/Users/99003163/Desktop/Embedded_c/ACTIVITY_1

99003163@EESBLRP096 MINGW64 ~/Desktop/Embedded_c/ACTIVITY_1
$ make
arm-none-eabi-gcc -c -mcpu=cortex-m4 -mthumb -mfloat-abi=soft -std=gnu11 -Wall -O0 -o main.o main.c
main.c: In function 'main':
main.c:8:8: warning: too many arguments for format [-Wformat-extra-args]
    8 | printf("the sum is :",res);
      |        ^~~~~~~~~~~~~~
arm-none-eabi-gcc -c -mcpu=cortex-m4 -mthumb -mfloat-abi=soft -std=gnu11 -Wall -O0 -o stm32_startup.o stm32_startup.c
#arm-none-eabi-gcc -c -mcpu=cortex-m4 -mthumb -mfloat-abi=soft -std=gnu11 -Wall -O0 -o stm32_startup.o stm32_startup.c
arm-none-eabi-gcc -c -mcpu=cortex-m4 -mthumb -mfloat-abi=soft -std=gnu11 -Wall -O0   -c -o syscalls.o syscalls.c
arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -mfloat-abi=soft --specs=nano.specs -T stm32_ls.ld -Wl,-Map=final.map -o final.elf main.o stm32_startup.o syscalls.o
```

Fig. 1: Make using GIT Bash



Name	Date modified	Type	Size
final.elf	1/2/2021 10:20 AM	ELF File	143 KB
final.map	1/2/2021 10:20 AM	MAP File	92 KB
main	12/26/2020 10:22 AM	C File	1 KB
main.o	1/2/2021 10:20 AM	O File	1 KB
MakeFile	12/26/2020 11:45 AM	File	1 KB
stm32_ls.ld	12/26/2020 11:41 AM	LD File	1 KB
stm32_startup	12/26/2020 11:26 AM	C File	13 KB
stm32_startup.o	1/2/2021 10:20 AM	O File	6 KB
syscalls	12/26/2020 11:13 AM	C File	5 KB
syscalls.o	1/2/2021 10:20 AM	O File	3 KB

Fig. 2: Activity Folder



```
MINGW64:/c/Users/99003163/Desktop/Embedded_c/ACTIVITY_1
 stm32_ls.ld -Wl,-Map=final.map -o final.elf main.o stm32_startup.o syscalls.o

99003163@EESBLRP096 MINGW64 ~/Desktop/Embedded_c/ACTIVITY_1
$ arm-none-eabi-objdump.exe -h stm32_startup.o

stm32_startup.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000090  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  000000c4  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000c4  2**0
                  ALLOC
  3 .isr_vector   00000188  00000000  00000000  000000c4  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, DATA
  4 .comment      0000004e  00000000  00000000  0000024c  2**0
                  CONTENTS, READONLY
  5 .ARM.attributes 0000002e  00000000  00000000  0000029a  2**0
                  CONTENTS, READONLY
```

Fig. 3: Memory Organization

2. ACTIVITY 2: LINKER SCRIPT

```
ENTRY(Reset_Handler)

MEMORY
{
  FLASH(rx):ORIGIN =0x08000000,LENGTH =1024K
  SRAM(rwx):ORIGIN =0x20000000,LENGTH =128K
}
SECTIONS
{
  .text :
  {
    *(.isr_vector)
    *(.text)
      *(.text.*)
      *(.init)
      *(.fini)
      *(.rodata)
      *(.rodata.*)
      . = ALIGN(4);
      _etext = .;
  }> FLASH

  _la_data = LOADADDR(.data);

  .data :
  {
    _sdata = .;
      *(.data)
      *(.data.*)
      . = ALIGN(4);
      _edata = .;
  }> SRAM AT> FLASH

  .bss :
  {
    _sbss = .;
      __bss_start__ = _sbss;
      *(.bss)
      *(.bss.*)
      *(COMMON)
      . = ALIGN(4);
      _ebss = .;
      __bss_end__ = _ebss;
        . = ALIGN(4);
        end = .;
        __end__ = .;
     }> SRAM
   }
```

3. ACTIVITY 3: HEADER FILE

3.1 MCU- Specific Header File

```
#include <stdint.h>
#ifndef INC_STM32F4XX_H_
#define INC_STM32F4XX_H_
#define __vo volatile

/* other definitions*/
#define ENABLE                  1
#define DISABLE                 0
#define GPIO_PIN_SET            ENABLE
#define GPIO_PIN_RESET          DISABLE

/* Defining macros for the various memory */
#define FLASH_ADDR             0x80000000U
#define SRAM1_ADDR             0x20000000U
#define SRAM2_ADDR             0x2001C000U
#define ROM_ADDR               0X1FFF0000U
#define SRAM_ADDR              SRAM1_ADDR

/* Defining macros for bus system */
#define AHB1_ADDR               0x40020000U
#define AHB2_ADDR               0x50000000U
#define APB1_ADDR               0x40000000U
#define APB2_ADDR               0x40010000U
#define PERI_ADDR               APB1_ADDR

/* Defining macros for peripherals hanging on AHB1 Bus */
#define GPIOA_ADDR              (AHB1_ADDR + 0x0000U)
#define GPIOB_ADDR              (AHB1_ADDR + 0x0400U)
#define GPIOC_ADDR              (AHB1_ADDR + 0x0800U)
#define GPIOD_ADDR              (AHB1_ADDR + 0x0C00U)
#define GPIOE_ADDR              (AHB1_ADDR + 0x1000U)
#define GPIOF_ADDR              (AHB1_ADDR + 0x1400U)
#define GPIOG_ADDR              (AHB1_ADDR + 0x1800U)
#define GPIOH_ADDR              (AHB1_ADDR + 0x1C00U)
#define GPIOI_ADDR              (AHB1_ADDR + 0x2000U)
#define RCC_ADDR                (AHB1_ADDR + 0x3800U)

/*Defining the macros for peripherals which are hanging on to APB1 bus*/

#define SPI2_I2S2_ADDR                (APB1_ADDR    + 0X3800U)
#define SPI3_I2S3_ADDR                (APB1_ADDR    + 0X3C00U)

#define USART2_ADDR                   (APB1_ADDR    + 0X4400U)
#define USART3_ADDR                   (APB1_ADDR    + 0X4800U)
```

```
#define UART4_ADDR                  (APB1_ADDR   + 0X4C00U)
#define UART5_ADDR                  (APB1_ADDR   + 0X5000U)

#define I2C1_ADDR                   (APB1_ADDR   + 0X5400U)
#define I2C2_ADDR                   (APB1_ADDR   + 0X5800U)
#define I2C3_ADDR                   (APB1_ADDR   + 0X5C00U)

#define CAN1_ADDR                   (APB1_ADDR   + 0X6400U)
#define CAN2_ADDR                   (APB1_ADDR   + 0X6800U)

/*Defining the macros for peripherals which are hanging on to APB2 bus*/

#define USART1_ADDR                 (APB2_ADDR   + 0X1000U)
#define USART6_ADDR                 (APB2_ADDR   + 0X1400U)
#define SPI1_ADDR                   (APB2_ADDR   + 0X3000U)

#define RCC ((RCC_GPIO_Reg_def_t*)RCC_ADDR)
/* GPIO Peripheral Registers */
typedef struct
{
        uint32_t MODER;
        uint32_t OTYPER;
        uint32_t OSPEEDR;
        uint32_t PUPDR;
        uint32_t IDR;
        uint32_t ODR;
        uint32_t BSRR;
        uint32_t LCKR;
        uint32_t AFR[2];         //AFRL[0](Low Register) & AFRH[1](High Register)
}GPIO_Reg_def_t;

/* RCC Registers*/
typedef struct
{
        __vo uint32_t RCC_CR;
        __vo uint32_t RCC_PLLCFGR;
        __vo uint32_t RCC_CFGR;
        __vo uint32_t RCC_CIR;
        __vo uint32_t RCC_AHB1RSTR;
        __vo uint32_t RCC_AHB2RSTR;
        __vo uint32_t RCC_AHB3RSTR;
        uint32_t RESERVED0;
        __vo uint32_t RCC_APB1RSTR;
        __vo uint32_t RCC_APB2RSTR;
        uint32_t RESERVED1[2];
        __vo uint32_t RCC_AHB1ENR;
        __vo uint32_t RCC_AHB2ENR;
        uint32_t RESERVED2[2];
        __vo uint32_t RCC_AHB3ENR;
        __vo uint32_t RCC_APB1ENR;
        __vo uint32_t RCC_APB2ENR;
        __vo uint32_t RCC_AHB1LPENR;
        __vo uint32_t RCC_AHB2LPENR;
        __vo uint32_t RCC_AHB3LPENR;
        __vo uint32_t RCC_APB1LPENR;
        __vo uint32_t RCC_APB2LPENR;
        __vo uint32_t RCC_BDCR;
        __vo uint32_t RCC_CSR;
        __vo uint32_t RCC_SSCGR;
        __vo uint32_t RCC_PLLI2SCFGR;
        __vo uint32_t RCC_PLLSAICFGR;
        __vo uint32_t RCC_DCKCFGR;
} RCC_GPIO_Reg_def_t;


/* GPIO Clock Enable */
#define GPIOA_PE_CLOCK_ENABLE()        RCC->RCC_AHB1ENR |= 1<<0
#define GPIOB_PE_CLOCK_ENABLE()        RCC->RCC_AHB1ENR |= 1<<1
#define GPIOC_PE_CLOCK_ENABLE()        RCC->RCC_AHB1ENR |= 1<<2
#define GPIOD_PE_CLOCK_ENABLE()        RCC->RCC_AHB1ENR |= 1<<3
#define GPIOE_PE_CLOCK_ENABLE()        RCC->RCC_AHB1ENR |= 1<<4
#define GPIOF_PE_CLOCK_ENABLE()        RCC->RCC_AHB1ENR |= 1<<5
#define GPIOG_PE_CLOCK_ENABLE()        RCC->RCC_AHB1ENR |= 1<<6
#define GPIOH_PE_CLOCK_ENABLE()        RCC->RCC_AHB1ENR |= 1<<7
#define GPIOI_PE_CLOCK_ENABLE()        RCC->RCC_AHB1ENR |= 1<<8

/* GPIO Clock Disable
#define GPIOA_PE_CLOCK_DISABLE() RCC->RCC_AHB1ENR &= ~(1<<0)
#define GPIOB_PE_CLOCK_DISABLE() RCC->RCC_AHB1ENR &= ~(1<<0)
#define GPIOC_PE_CLOCK_DISABLE() RCC->RCC_AHB1ENR &= ~(1<<0)
#define GPIOD_PE_CLOCK_DISABLE() RCC->RCC_AHB1ENR &= ~(1<<0)
#define GPIOE_PE_CLOCK_DISABLE() RCC->RCC_AHB1ENR &= ~(1<<0)
#define GPIOF_PE_CLOCK_DISABLE() RCC->RCC_AHB1ENR &= ~(1<<0)
#define GPIOG_PE_CLOCK_DISABLE() RCC->RCC_AHB1ENR &= ~(1<<0)
#define GPIOH_PE_CLOCK_DISABLE() RCC->RCC_AHB1ENR &= ~(1<<0)
#define GPIOI_PE_CLOCK_DISABLE() RCC->RCC_AHB1ENR &= ~(1<<0)*/

/* GPIO Clock Reset */
#define GPIOA_PE_CLOCK_RESET()         do{RCC->RCC_AHB1RSTR |= (1<<0);RCC->RCC_AHB1RSTR &=
~(1<<0);}while(DISABLE)
#define GPIOB_PE_CLOCK_RESET()         do{RCC->RCC_AHB1RSTR |= (1<<1);RCC->RCC_AHB1RSTR &=
~(1<<1);}while(DISABLE)
#define GPIOC_PE_CLOCK_RESET()         do{RCC->RCC_AHB1RSTR |= (1<<2);RCC->RCC_AHB1RSTR &=
~(1<<2);}while(DISABLE)
#define GPIOD_PE_CLOCK_RESET()         do{RCC->RCC_AHB1RSTR |= (1<<3);RCC->RCC_AHB1RSTR &=
~(1<<3);}while(DISABLE)
#define GPIOE_PE_CLOCK_RESET()         do{RCC->RCC_AHB1RSTR |= (1<<4);RCC->RCC_AHB1RSTR &=
~(1<<4);}while(DISABLE)
#define GPIOF_PE_CLOCK_RESET()         do{RCC->RCC_AHB1RSTR |= (1<<5);RCC->RCC_AHB1RSTR &=
~(1<<5);}while(DISABLE)
#define GPIOG_PE_CLOCK_RESET()         do{RCC->RCC_AHB1RSTR |= (1<<6);RCC->RCC_AHB1RSTR &=
~(1<<6);}while(DISABLE)
```

```
#define GPIOH_PE_CLOCK_RESET()          do{RCC->RCC_AHB1RSTR |= (1<<7);RCC->RCC_AHB1RSTR &=
~(1<<7);}while(DISABLE)
#define GPIOI_PE_CLOCK_RESET()          do{RCC->RCC_AHB1RSTR |= (1<<8);RCC->RCC_AHB1RSTR &=
~(1<<8);}while(DISABLE)

/* GPIO Peripheral */
#define GPIOA ((GPIO_Reg_def_t*)GPIOA_ADDR)
#define GPIOB ((GPIO_Reg_def_t*)GPIOB_ADDR)
#define GPIOC ((GPIO_Reg_def_t*)GPIOC_ADDR)
#define GPIOD ((GPIO_Reg_def_t*)GPIOD_ADDR)
#define GPIOE ((GPIO_Reg_def_t*)GPIOE_ADDR)
#define GPIOF ((GPIO_Reg_def_t*)GPIOF_ADDR)
#define GPIOG ((GPIO_Reg_def_t*)GPIOG_ADDR)
#define GPIOH ((GPIO_Reg_def_t*)GPIOH_ADDR)
#define GPIOI ((GPIO_Reg_def_t*)GPIOI_ADDR)

   #endif /* INC_STM32F4XX_H_ */
```

3.2 GPIO-Specific Header File

```
#include "stm32f4xx.h"
#ifndef INC_STM32F4XX_GPIO_DRIVER_H_
#define INC_STM32F4XX_GPIO_DRIVER_H_

/* GPIO Pin Configuration*/
typedef struct
{
      uint8_t GPIO_Pin_Number;
      uint8_t GPIO_PinMode;
      uint8_t GPIO_Pin_Speed;
      uint8_t GPIO_Pin_PuPd_Control;
      uint8_t GPIO_Pin_OP_Type;
      uint8_t GPIO_Pin_Alt_Fun_Mode;
}GPIO_PIN_CONFIG_T;

/* GPIO Handle Structure */
typedef struct
{
      GPIO_Reg_def_t *pGPIOx;
      GPIO_PIN_CONFIG_T PIN_CONFIG;
}GPIO_HANDLE_T;

/* GPIO pin numbering */
#define GPIO_PIN_NUMBER_0               0
#define GPIO_PIN_NUMBER_1               1
#define GPIO_PIN_NUMBER_2               2
#define GPIO_PIN_NUMBER_3               3
#define GPIO_PIN_NUMBER_4               4
#define GPIO_PIN_NUMBER_5               5
#define GPIO_PIN_NUMBER_6               6
#define GPIO_PIN_NUMBER_7               7
```

```
#define GPIO_PIN_NUMBER_8               8
#define GPIO_PIN_NUMBER_9               9
#define GPIO_PIN_NUMBER_10              10
#define GPIO_PIN_NUMBER_11              11
#define GPIO_PIN_NUMBER_12              12
#define GPIO_PIN_NUMBER_13              13
#define GPIO_PIN_NUMBER_14              14
#define GPIO_PIN_NUMBER_15              15

/* GPIO Operating Modes */
#define GPIO_PIN_MODE_IN                0
#define GPIO_PIN_MODE_OUT               1
#define GPIO_PIN_MODE_ALT               2
#define GPIO_PIN_MODE_ANALOG            3
#define GPIO_PIN_MODE_RT                4
#define GPIO_PIN_MODE_FT                5
#define GPIO_PIN_MODE_RFT               6

/* GPIO pin possible output speeds*/
#define GPIO_PIN_SPEED_LOW              0
#define GPIO_PIN_SPEED_MEDIUM           1
#define GPIO_PIN_SPEED_FAST                   2
#define GPIO_PIN_SPEED_HIGH                   3

/* GPIO*/
#define GPIO_PIN_PUPD_CONTROL_0         0
#define GPIO_PIN_PUPD_CONTROL_1         1
#define GPIO_PIN_PUPD_CONTROL_2         2
#define GPIO_PIN_PUPD_CONTROL_3         3

/* GPIO Pin Output Types */
#define GPIO_OP_TYPE_PP                       0
#define GPIO_OP_TYPE_OD                       1

/* GPIO pin pull up and pull down configuration */
#define GPIO_NO_PUPD                          0
#define GPIO_PIN_PU                           1
#define GPIO_PIN_PD                           2

void GPIO_PeriClockControl(GPIO_Reg_def_t *pGPIOx,uint8_t EnorDi);

void GPIO_Init(GPIO_HANDLE_T *pGPIOHandle);
void GPIO_DeInit(GPIO_Reg_def_t *pGIOx);

uint8_t GPIO_ReadFromInputPin(GPIO_Reg_def_t *pGPIOx,uint8_t PinNumber);
uint16_t GPIO_ReadFromInputPort(GPIO_Reg_def_t *pGPIOx);


void GPIO_WriteToOutputPin(GPIO_Reg_def_t *pGPIOx,uint8_t PinNumber,uint8_t value);
void GPIO_WriteToOutputPort(GPIO_Reg_def_t *pGPIOx,uint8_t value);
void GPIO_ToggleOutputPin(GPIO_Reg_def_t *pGPIOx,uint8_t PinNumber);
```

```
#endif /* INC_STM32F4XX_GPIO_DRIVER_H_ */
```

3.3 Source File:

```
#include "stm32f4XX_GPIO_driver.h"

void GPIO_PeriClockControl(GPIO_Reg_def_t *pGPIOx,uint8_t EnorDi)
{
        if(EnorDi == ENABLE)
        {
                if(pGPIOx == GPIOA)
                {
                        GPIOA_PE_CLOCK_ENABLE();
                }
                else if(pGPIOx == GPIOB)
                {
                        GPIOB_PE_CLOCK_ENABLE();
                }
                else if(pGPIOx == GPIOC)
                {
                        GPIOC_PE_CLOCK_ENABLE();
                }
                else if(pGPIOx == GPIOD)
                {
                        GPIOD_PE_CLOCK_ENABLE();
                }
                else if(pGPIOx == GPIOE)
                {
                        GPIOE_PE_CLOCK_ENABLE();
                }
                else if(pGPIOx == GPIOF)
                {
                        GPIOF_PE_CLOCK_ENABLE();
                }
                else if(pGPIOx == GPIOG)
                {
                        GPIOG_PE_CLOCK_ENABLE();
                }
                else if(pGPIOx == GPIOH)
                {
                        GPIOH_PE_CLOCK_ENABLE();
                }
                else if(pGPIOx == GPIOI)
                {
                        GPIOI_PE_CLOCK_ENABLE();
                }
        }
        else
        {
                if(pGPIOx == GPIOA)
                {
                        GPIOA_PE_CLOCK_RESET();
                }
                else if(pGPIOx == GPIOB)
                {
                        GPIOB_PE_CLOCK_RESET();
                }
                else if(pGPIOx == GPIOC)
                {
                        GPIOC_PE_CLOCK_RESET();
                }
                else if(pGPIOx == GPIOD)
                {
                        GPIOD_PE_CLOCK_RESET();
                }
                else if(pGPIOx == GPIOE)
                {
                        GPIOE_PE_CLOCK_RESET();
                }
                else if(pGPIOx == GPIOF)
                {
                        GPIOF_PE_CLOCK_RESET();
                }
                else if(pGPIOx == GPIOG)
                {
                        GPIOG_PE_CLOCK_RESET();
                }
                else if(pGPIOx == GPIOH)
                {
                        GPIOH_PE_CLOCK_RESET();
                }
                else if(pGPIOx == GPIOI)
                {
                        GPIOI_PE_CLOCK_RESET();
                }
        }
    }

    void GPIO_Init(GPIO_HANDLE_T *pGPIOHandle)
    {
        //1. configuring the mode
        uint32_t temp=0;
                if(pGPIOHandle->PIN_CONFIG.GPIO_PinMode <= GPIO_PIN_MODE_ANALOG )//non
    interrupt modes
                {
                        temp = pGPIOHandle->PIN_CONFIG.GPIO_PinMode<<(2*pGPIOHandle-
    >PIN_CONFIG.GPIO_Pin_Number);
                        pGPIOHandle->pGPIOx->MODER |= temp;
                }
                else
                {
                           //interrupt mode FT, RT ,FTRT
                }
        //2. configuring the speed
        uint32_t temp1=0;
```

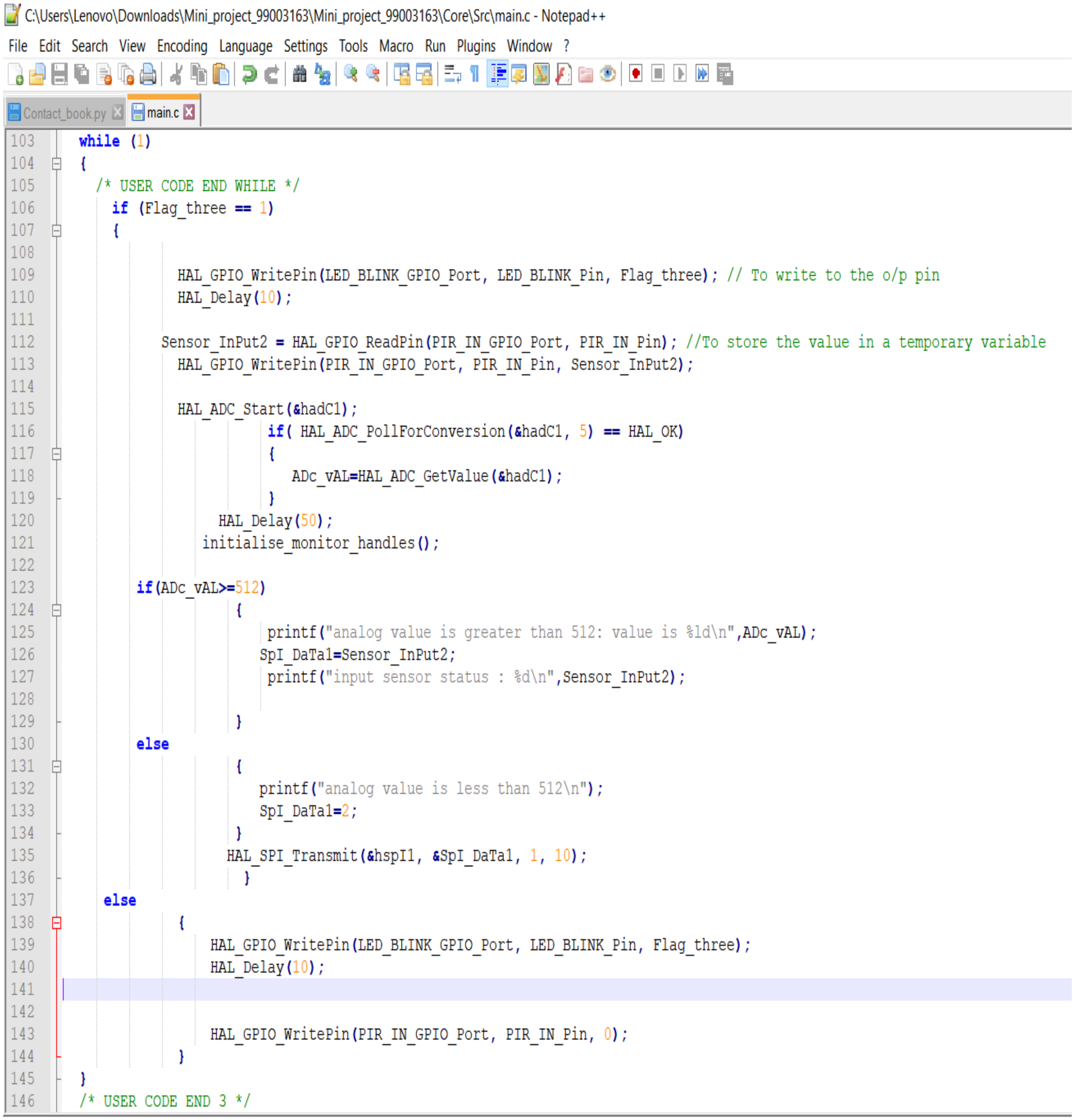
```
        temp1 = pGPIOHandle->PIN_CONFIG.GPIO_Pin_Speed<<(2*pGPIOHandle-
>PIN_CONFIG.GPIO_Pin_Number);
        pGPIOHandle->pGPIOx->OSPEEDR |= temp1;

        //3. configuring the pu pd control
        uint32_t temp2=0;
        temp2 = pGPIOHandle->PIN_CONFIG.GPIO_Pin_PuPd_Control<<(2*pGPIOHandle-
>PIN_CONFIG.GPIO_Pin_Number);
        pGPIOHandle->pGPIOx->PUPDR |= temp2;

        //4. configuring the output type
        uint32_t temp3=0;
        temp3 = pGPIOHandle->PIN_CONFIG.GPIO_Pin_OP_Type<<(pGPIOHandle-
>PIN_CONFIG.GPIO_Pin_Number);
        pGPIOHandle->pGPIOx->OTYPER |= temp3;

        uint32_t tempA = pGPIOHandle->PIN_CONFIG.GPIO_Pin_Number /8;
        uint32_t tempB = pGPIOHandle->PIN_CONFIG.GPIO_Pin_Number %8;
        pGPIOHandle->pGPIOx->AFR[tempA] |= pGPIOHandle->PIN_CONFIG.GPIO_Pin_Alt_Fun_Mode <<
(4*tempB);
        if(tempA == 0)
        {
                if(tempB == 0);
        }
}

void GPIO_DeInit(GPIO_Reg_def_t *pGIOx)
{

        if(EnorDi == ENABLE)
                {
                        if(pGPIOx == GPIOA)
                        {
                                GPIOA_PE_CLOCK_ENABLE();
                        }
                        else if(pGPIOx == GPIOB)
                        {
                                GPIOB_PE_CLOCK_ENABLE();
                        }
                        else if(pGPIOx == GPIOC)
                        {
                                GPIOC_PE_CLOCK_ENABLE();
                        }
                        else if(pGPIOx == GPIOD)
                        {
                                GPIOD_PE_CLOCK_ENABLE();
                        }
                        else if(pGPIOx == GPIOE)
                        {
                                GPIOE_PE_CLOCK_ENABLE();
                        }
                        else if(pGPIOx == GPIOF)
```

```
            {
                GPIOF_PE_CLOCK_ENABLE();
            }
            else if(pGPIOx == GPIOG)
            {
                GPIOG_PE_CLOCK_ENABLE();
            }
            else if(pGPIOx == GPIOH)
            {
                GPIOH_PE_CLOCK_ENABLE();
            }
            else if(pGPIOx == GPIOI)
            {
                GPIOI_PE_CLOCK_ENABLE();
            }
        }
        else
        {
            if(pGPIOx == GPIOA)
                    {
                        GPIOA_PE_CLOCK_ENABLE();
                    }
                    else if(pGPIOx == GPIOB)
                    {
                        GPIOB_PE_CLOCK_ENABLE();
                    }
                    else if(pGPIOx == GPIOC)
                    {
                        GPIOC_PE_CLOCK_ENABLE();
                    }
                    else if(pGPIOx == GPIOD)
                    {
                        GPIOD_PE_CLOCK_ENABLE();
                    }
                    else if(pGPIOx == GPIOE)
                    {
                        GPIOE_PE_CLOCK_ENABLE();
                    }
                    else if(pGPIOx == GPIOF)
                    {
                        GPIOF_PE_CLOCK_ENABLE();
                    }
                    else if(pGPIOx == GPIOG)
                    {
                        GPIOG_PE_CLOCK_ENABLE();
                    }
                    else if(pGPIOx == GPIOH)
                    {
                        GPIOH_PE_CLOCK_ENABLE();
                    }
                    else if(pGPIOx == GPIOI)
                    {
                        GPIOI_PE_CLOCK_ENABLE();
                    }// To be done
        }

}

uint8_t GPIO_ReadFromInputPin(GPIO_Reg_def_t *pGPIOx,uint8_t PinNumber)
{
    uint8_t value;
    value = (uint8_t)((pGPIOx->IDR >> PinNumber) * (0x00000001));
    return value;
}

uint16_t GPIO_ReadFromInputPort(GPIO_Reg_def_t *pGPIOx)
{
    uint16_t value;
    value = (uint16_t)(pGPIOx->IDR);
    return value;
}

void GPIO_WriteToOutputPin(GPIO_Reg_def_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_Reg_def_t *pGPIOx,uint8_t value)
{
    pGPIOx->ODR = value;
}

void GPIO_ToggleOutputPin(GPIO_Reg_def_t *pGPIOx,uint8_t PinNumber)
{
    pGPIOx->ODR ^= (1<<PinNumber);
}
```

4. ACTIVITY 4:

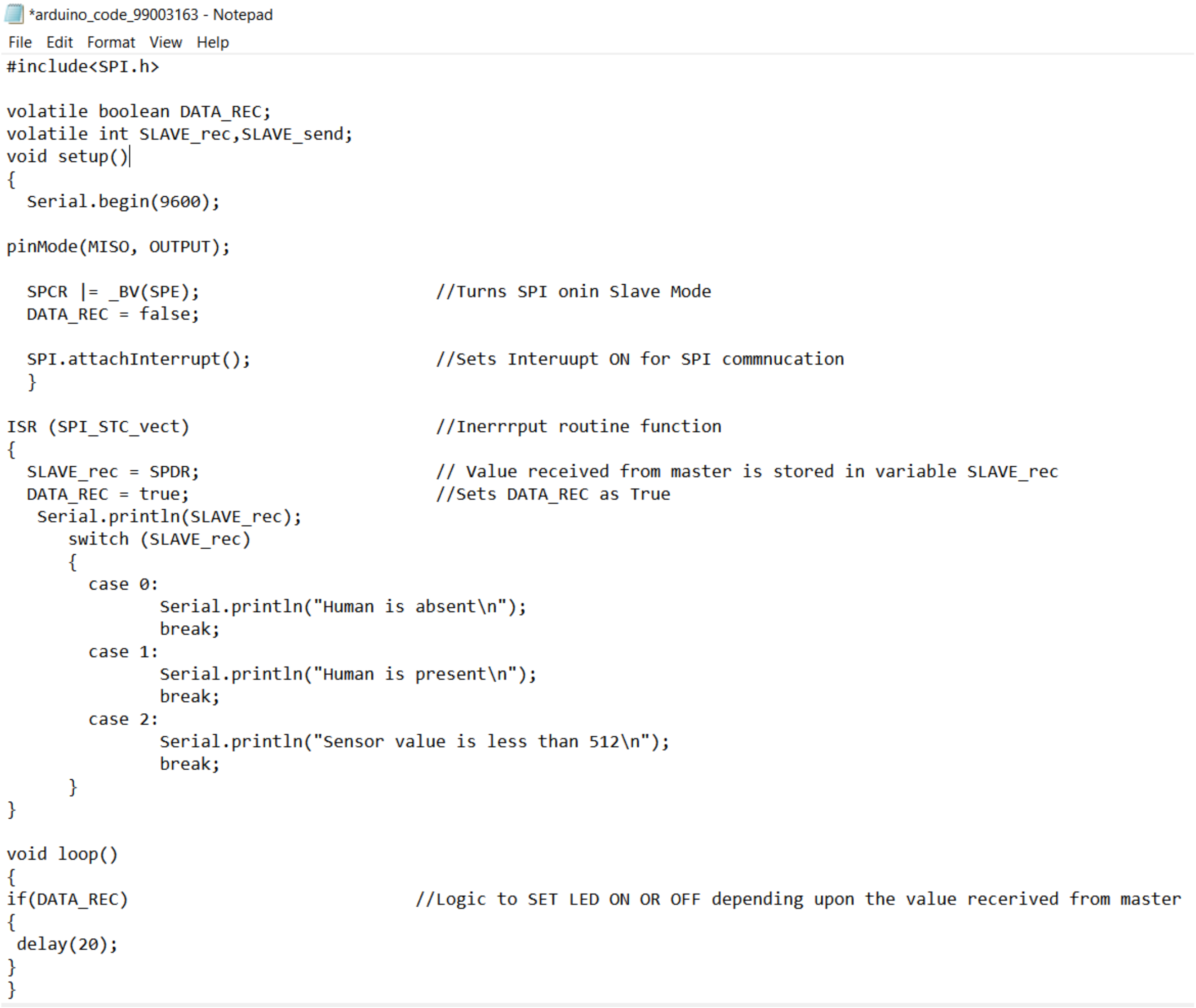
GITHUB LINK: https://github.com/99003163/Embedded_C

4.1 LOGIC CODE


C:\Users\Lenovo\Downloads\Mini_project_99003163\Mini_project_99003163\Core\Src\main.c - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Contact_book.py
main.c

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

            HAL_GPIO_WritePin(LED_BLINK_GPIO_Port, LED_BLINK_Pin, Flag_three); // To write to the o/p pin
            HAL_Delay(10);

          Sensor_InPut2 = HAL_GPIO_ReadPin(PIR_IN_GPIO_Port, PIR_IN_Pin); //To store the value in a temporary variable
            HAL_GPIO_WritePin(PIR_IN_GPIO_Port, PIR_IN_Pin, Sensor_InPut2);

            HAL_ADC_Start(&hadC1);
                       if( HAL_ADC_PollForConversion(&hadC1, 5) == HAL_OK)
                       {
                          ADc_vAL=HAL_ADC_GetValue(&hadC1);
                       }
                 HAL_Delay(50);
               initialise_monitor_handles();

       if(ADc_vAL>=512)
                   {
                       printf("analog value is greater than 512: value is %ld\n",ADc_vAL);
                      SpI_DaTa1=Sensor_InPut2;
                       printf("input sensor status : %d\n",Sensor_InPut2);

                   }
       else
                   {
                      printf("analog value is less than 512\n");
                      SpI_DaTa1=2;
                   }
                  HAL_SPI_Transmit(&hspI1, &SpI_DaTa1, 1, 10);
                    }
   else
            {
                HAL_GPIO_WritePin(LED_BLINK_GPIO_Port, LED_BLINK_Pin, Flag_three);
                HAL_Delay(10);


                HAL_GPIO_WritePin(PIR_IN_GPIO_Port, PIR_IN_Pin, 0);
            }
}
/* USER CODE END 3 */
```

4.2 ARDUINO CODE



```
#include<SPI.h>

volatile boolean DATA_REC;
volatile int SLAVE_rec,SLAVE_send;
void setup()
{
  Serial.begin(9600);

pinMode(MISO, OUTPUT);

  SPCR |= _BV(SPE);                       //Turns SPI onin Slave Mode
  DATA_REC = false;

  SPI.attachInterrupt();                  //Sets Interuupt ON for SPI commnucation
  }

ISR (SPI_STC_vect)                        //Inerrrput routine function
{
  SLAVE_rec = SPDR;                       // Value received from master is stored in variable SLAVE_rec
  DATA_REC = true;                        //Sets DATA_REC as True
   Serial.println(SLAVE_rec);
      switch (SLAVE_rec)
      {
        case 0:
               Serial.println("Human is absent\n");
               break;
        case 1:
               Serial.println("Human is present\n");
               break;
        case 2:
               Serial.println("Sensor value is less than 512\n");
               break;
      }
}

void loop()
{
if(DATA_REC)                            //Logic to SET LED ON OR OFF depending upon the value recerived from master
{
 delay(20);
}
}
```

REFEERENCES:

[1]. http://web.cs.iastate.edu/~smkautz/cs227s13/labs/lab6/page04.html
[2]. https://youtu.be/2Hm8eEHsgls
[3].https://youtu.be/Bsq6P1B8JqI