LEARNING REPORT
Embedded C

LTTS GLOBAL ENGINEERING ACADEMY

L&T Technology Services

Sana MD
99003180

Details

Ver. Rel. No.	Release Date	Prepared. By	Reviewed By	To be Approved	Remarks/Revision Details
1	04/01/2021	Sana Md			

Table of Contents

List of Figures

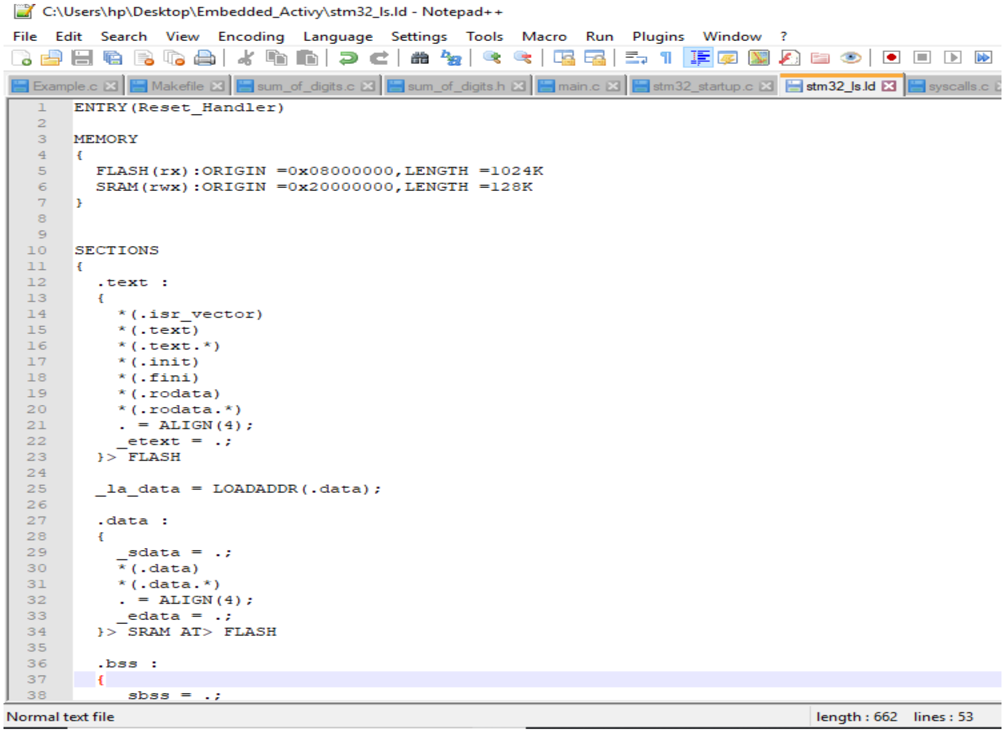
1 ACTIVITY

1.1 Object File

An object file is a computer file containing object code, i.e., machine code output of an assembler or complier. The object code is usually relocatable and not usually directly exectable. GCC will generate files with a .o extension.

1.2 Linker Script

When compiling the program, it is necessary to perform a few extra steps to ensure that the program is ready to be loaded and run by the boot code. The last step in compiling a program is to link all of the object files together, possibly also including some object files from system libraries. The default linker script used by GCC creates an ELF executable file, which includes startup code from the C library and also includes information which tells the loader where the various sections reside in memory.



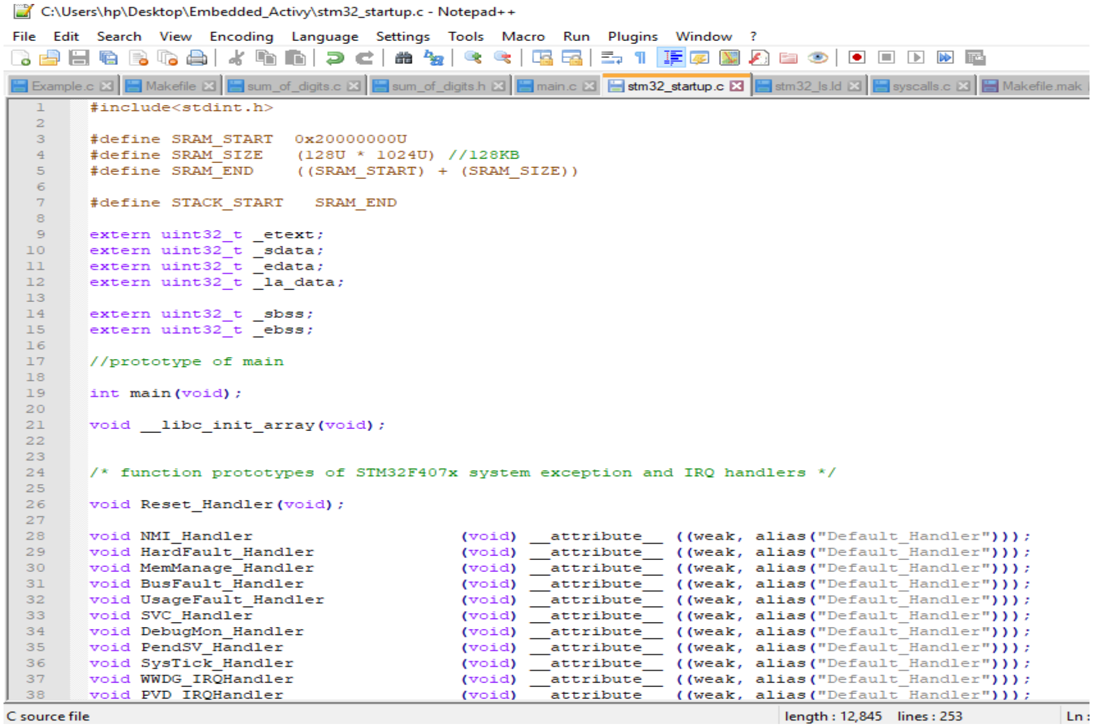
C:\Users\hp\Desktop\Embedded_Activy\stm32_ls.ld - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example.c
Makefile
sum_of_digits.c
sum_of_digits.h
main.c
stm32_startup.c
stm32_ls.ld
syscalls.c
Normal text file
length : 662 lines : 53

```
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 = .;
```

Fig. 1 Linker Script

1.3 StartUp code

The startup code provides the reset vector, initial stack pointer value, bus configuration registers and a symbol for each of the interrupt vectors. When the processor starts, it will initialize the MSP by loading the value stored in the first 4 bytes of the vector table. Then it will jump to the reset handler.

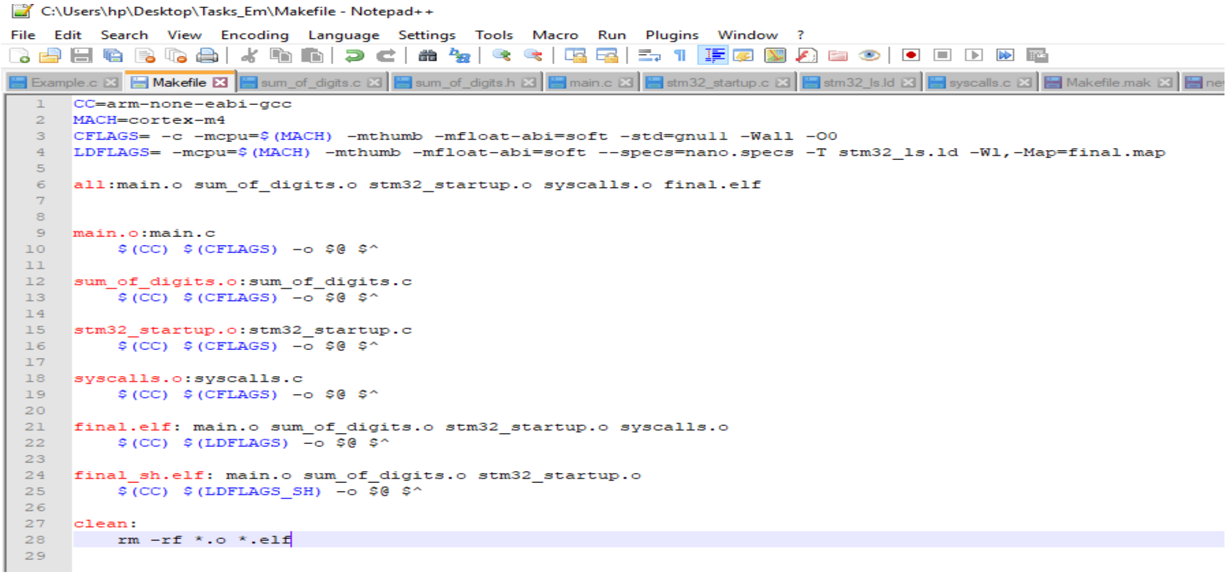

C:\Users\hp\Desktop\Embedded_Activy\stm32_startup.c - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example.c
Makefile
sum_of_digits.c
sum_of_digits.h
main.c
stm32_startup.c
stm32_ls.ld
syscalls.c
Makefile.mak
C source file
length : 12,845 lines : 253
Ln :

```c
#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")));
```

Fig. 2 Startup code

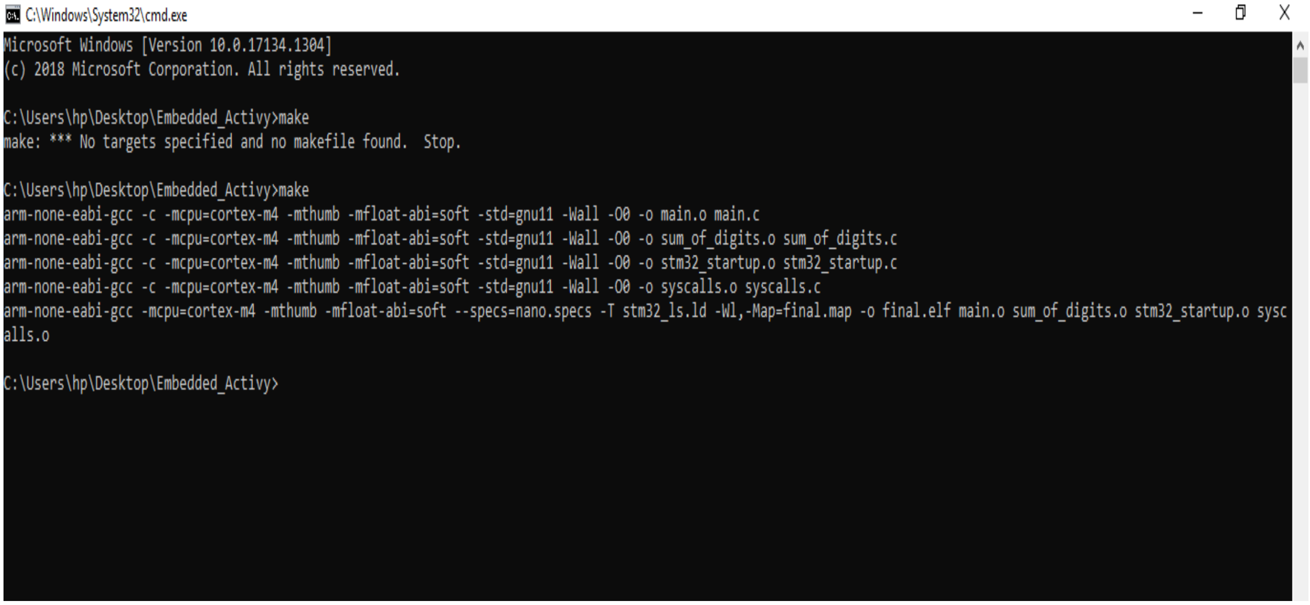
1.4 Makefile

Makefile is a set of commands with variable names and targets to create object file and to remove them. In a single make file we can create multiple targets to compile and to remove object, binary files.


C:\Users\hp\Desktop\Tasks_Em\Makefile - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example.c
Makefile
sum_of_digits.c
sum_of_digits.h
main.c
stm32_startup.c
stm32_ls.ld
syscalls.c
Makefile.mak
ne

```makefile
CC=arm-none-eabi-gcc
MACH=cortex-m4
CFLAGS= -c -mcpu=$(MACH) -mthumb -mfloat-abi=soft -std=gnu11 -Wall -O0
LDFLAGS= -mcpu=$(MACH) -mthumb -mfloat-abi=soft --specs=nano.specs -T stm32_ls.ld -Wl,-Map=final.map

all:main.o sum_of_digits.o stm32_startup.o syscalls.o final.elf


main.o:main.c
	$(CC) $(CFLAGS) -o $@ $^

sum_of_digits.o:sum_of_digits.c
	$(CC) $(CFLAGS) -o $@ $^

stm32_startup.o:stm32_startup.c
	$(CC) $(CFLAGS) -o $@ $^

syscalls.o:syscalls.c
	$(CC) $(CFLAGS) -o $@ $^

final.elf: main.o sum_of_digits.o stm32_startup.o syscalls.o
	$(CC) $(LDFLAGS) -o $@ $^

final_sh.elf: main.o sum_of_digits.o stm32_startup.o
	$(CC) $(LDFLAGS_SH) -o $@ $^

clean:
	rm -rf *.o *.elf
```

Fig. 3 Makefile



```
Microsoft Windows [Version 10.0.17134.1304]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\hp\Desktop\Embedded_Activy>make
make: *** No targets specified and no makefile found.  Stop.

C:\Users\hp\Desktop\Embedded_Activy>make
arm-none-eabi-gcc -c -mcpu=cortex-m4 -mthumb -mfloat-abi=soft -std=gnu11 -Wall -O0 -o main.o main.c
arm-none-eabi-gcc -c -mcpu=cortex-m4 -mthumb -mfloat-abi=soft -std=gnu11 -Wall -O0 -o sum_of_digits.o sum_of_digits.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 -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 sum_of_digits.o stm32_startup.o sysc
alls.o

C:\Users\hp\Desktop\Embedded_Activy>
```

Fig. 4 Executing Make command in cmd

2 MINI PROJECT

2.1 About Project in Brief

The major parts of the system are MQ7 Gas Sensor, Arduino Uno and STM32F40VGT6 microcontroller. Arduino and STM32 are connected to a computer by a USB link. The sensors' analog pin (A0) is connected to Port B0 (PB0) of STM32. Arduino is connected to STM32 using SPI communication protocol. MQ7 senses gases present in atmosphere and same are sent to STM32. Using SPI communication, the output values can be viewed in serial monitor of Arduino.

2.2 Software and Hardware required

- MQ7 Gas sensor
- Arduino Uno
- STM32
- Arduino IDE
- STM32 Cube IDE
- USB Cables

2.3 Block Diagram

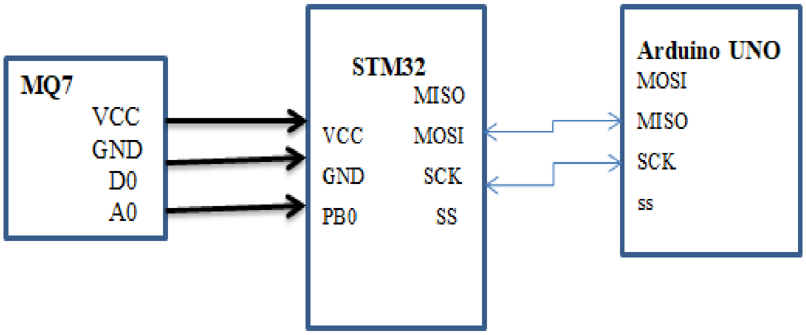


Fig. 5 Block Diagram of Smoke Level Detector

2.4 Working

MQ7 sensor detects carbon monoxide and the sensed values are processed by STM32 Controller. Different threshold values are set and when sensed value reaches a threshold, corresponding LED glows, indicating the level of Gases. The same values are displayed in serial monitor of Arduino. STM32 reads and produces output in analog value which is not a very useful parameter for gas concentration reading. This data must be converted to PPM (parts permillion) values. First of all, conversion of the analog values (0-1023) to corresponding voltage values (Vout)(0-5V) is done using:

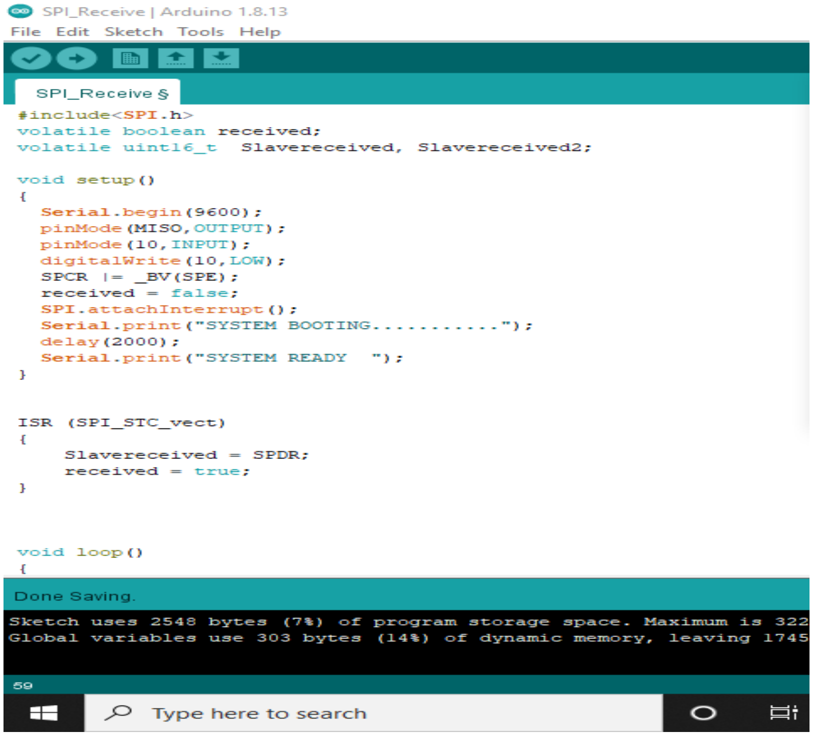
$$Vout = (AnalogValue*5)/1023$$

Resistance of sensor (RS) is defined in the datasheet of MQ135 as:

$$Rs \;=\; Vcc\ Vout - 1 * RL$$

$$= 1023\ AnalogValue - 1 * RL$$

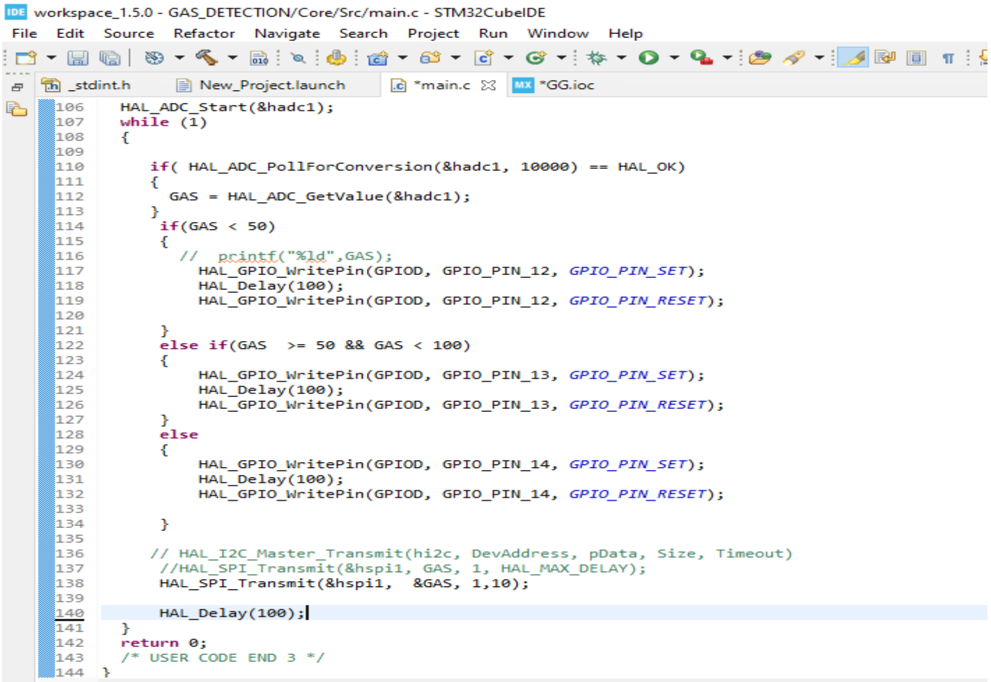
2.5 Results



```
#include<SPI.h>
volatile boolean received;
volatile uint16_t  Slavereceived, Slavereceived2;

void setup()
{
  Serial.begin(9600);
  pinMode(MISO,OUTPUT);
  pinMode(10,INPUT);
  digitalWrite(10,LOW);
  SPCR |= _BV(SPE);
  received = false;
  SPI.attachInterrupt();
  Serial.print("SYSTEM BOOTING...........");
  delay(2000);
  Serial.print("SYSTEM READY  ");
}


ISR (SPI_STC_vect)
{
    Slavereceived = SPDR;
    received = true;
}


void loop()
{
```

```
Done Saving.
Sketch uses 2548 bytes (7%) of program storage space. Maximum is 322
Global variables use 303 bytes (14%) of dynamic memory, leaving 1745

59
```

Fig. 6 Arduino Receiving data using SPI


workspace_1.5.0 - GAS_DETECTION/Core/Src/main.c - STM32CubeIDE
File Edit Source Refactor Navigate Search Project Run Window Help
_stdint.h
New_Project.launch
*main.c
*GG.ioc

```
  HAL_ADC_Start(&hadc1);
  while (1)
  {

      if( HAL_ADC_PollForConversion(&hadc1, 10000) == HAL_OK)
      {
        GAS = HAL_ADC_GetValue(&hadc1);
      }
       if(GAS < 50)
       {
         //  printf("%ld",GAS);
           HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
           HAL_Delay(100);
           HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);

       }
       else if(GAS  >= 50 && GAS < 100)
       {
           HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);
           HAL_Delay(100);
           HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
       }
       else
       {
           HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
           HAL_Delay(100);
           HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);

       }

      // HAL_I2C_Master_Transmit(hi2c, DevAddress, pData, Size, Timeout)
       //HAL_SPI_Transmit(&hspi1, GAS, 1, HAL_MAX_DELAY);
       HAL_SPI_Transmit(&hspi1,  &GAS, 1,10);

       HAL_Delay(100);
  }
  return 0;
  /* USER CODE END 3 */
}
```

Fig. 7 STM32 transmitting data

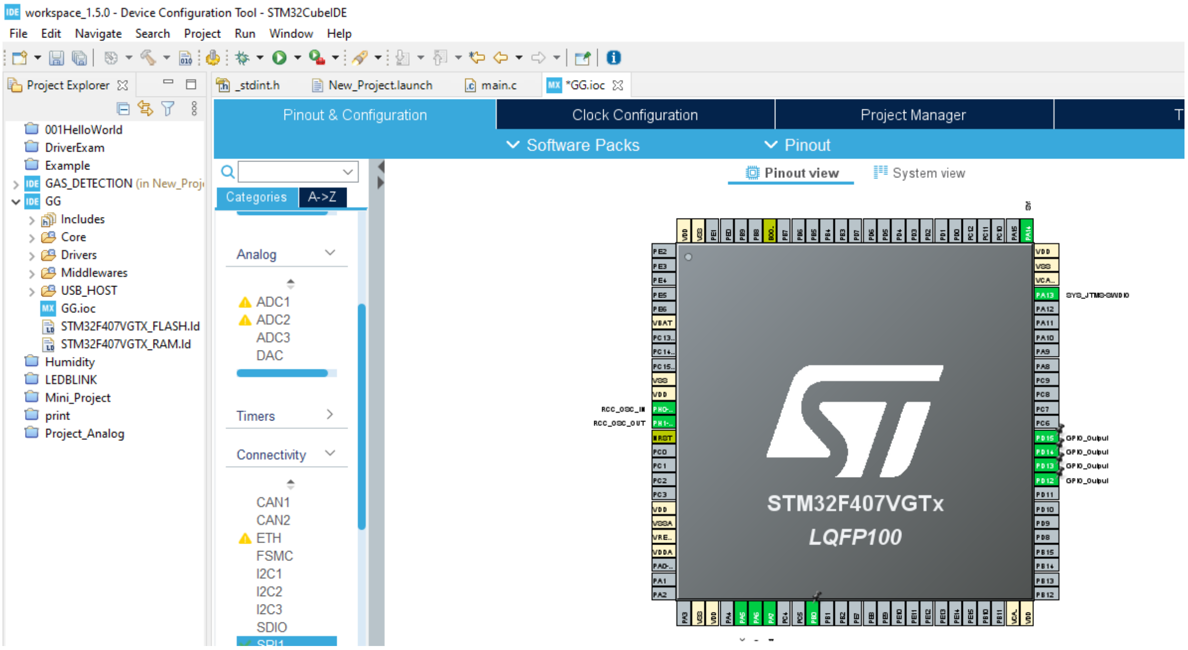


Fig. 8 GUI of Project

GitHub Link: https://github.com/99003180/Embedded_Project.git