./

LEARNING REPORT



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **To be Approved** | **Remarks/Revision Details** |
| 1 | 04/01/2021 | Ruchitha Pagadala |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Details**

Table of Contents

[1 ACTIVITY 4](#_Toc60587764)

[1.1 Object File 4](#_Toc60587765)

[1.2 Linker Script 4](#_Toc60587766)

[1.3 StartUp code 4](#_Toc60587767)

[1.4 Makefile 5](#_Toc60587768)

[2 MINI PROJECT 6](#_Toc60587771)

[2.1 About Project in Brief 6](#_Toc60587772)

[2.2 Software and Hardware required 6](#_Toc60587773)

[2.3 Block Diagram 7](#_Toc60587774)

[2.4 Working 7](#_Toc60587775)

[2.5 Results 8](#_Toc60587776)

# ACTIVITY

## 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.

## 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.

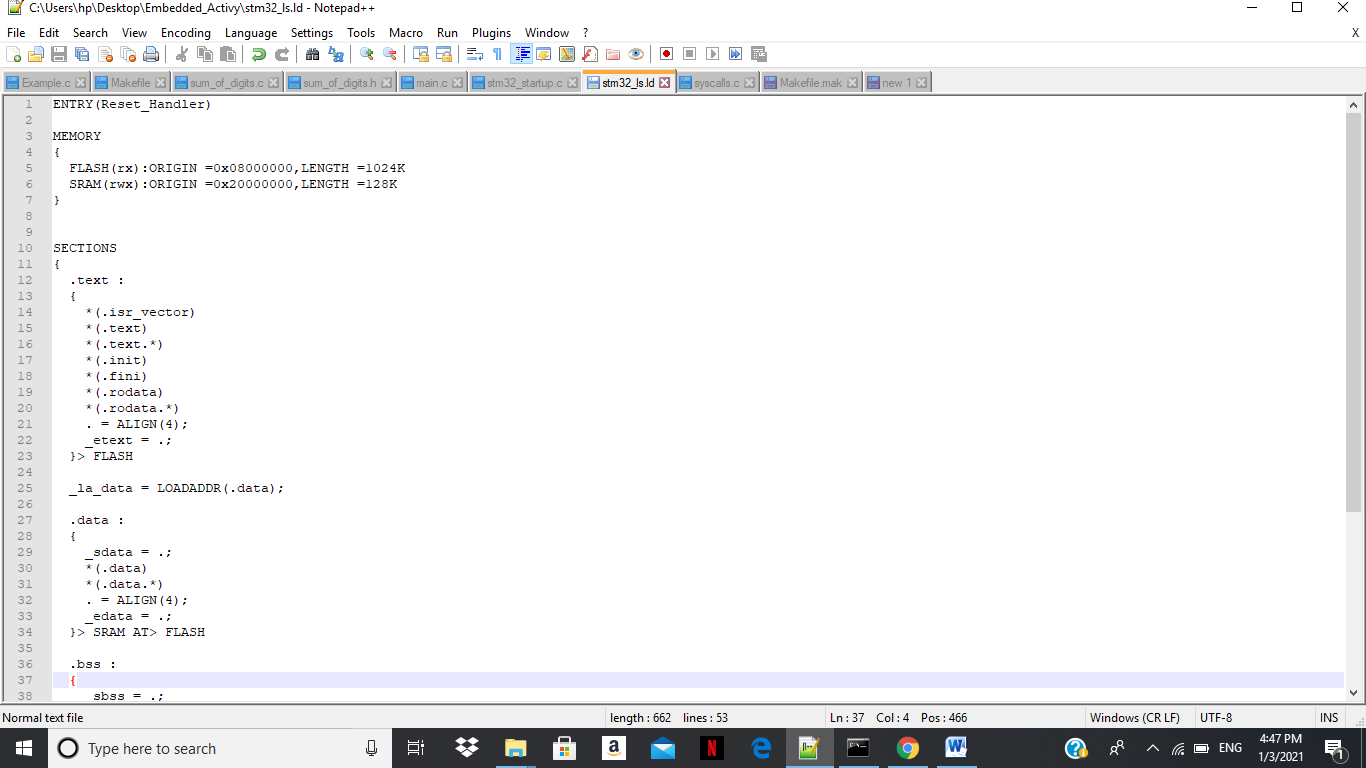


Fig. Linker Script

## 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.

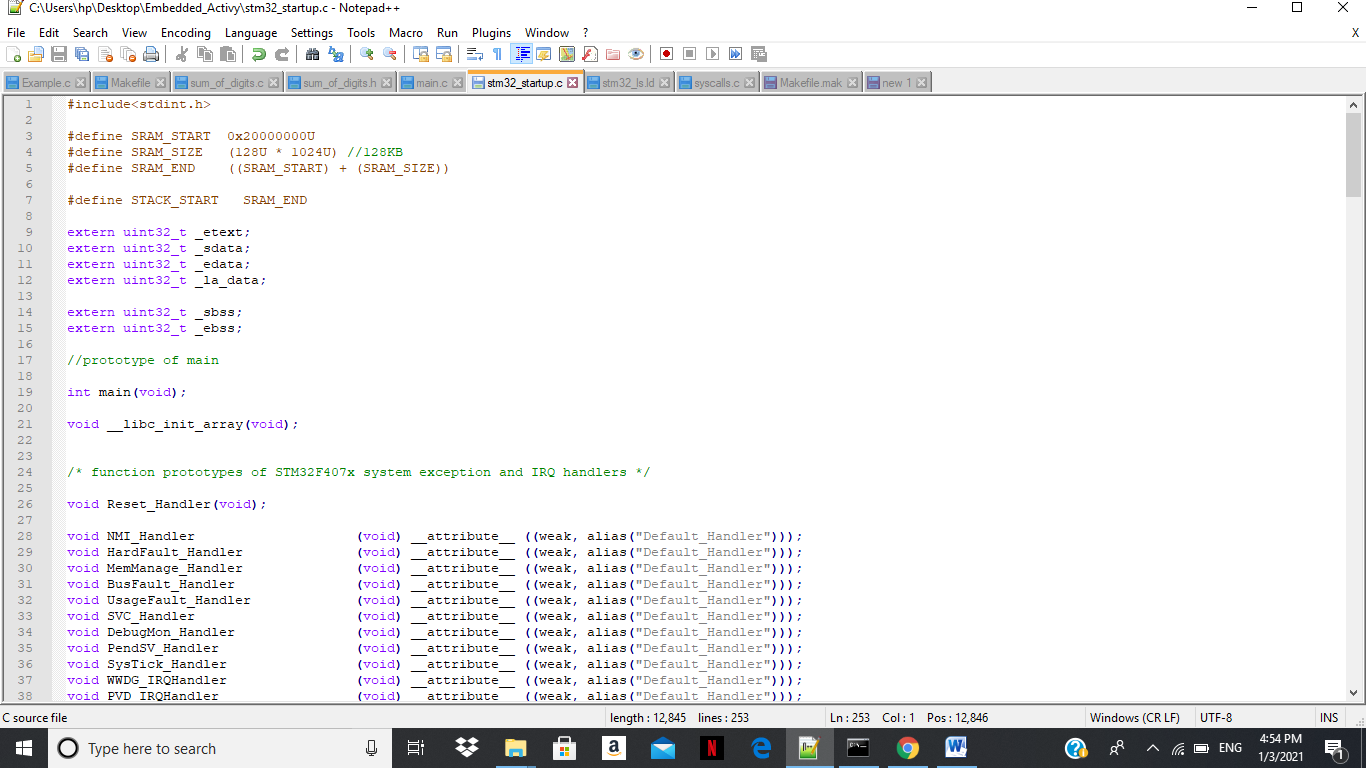


Fig. Startup code

## 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.

## 

Fig. Makefile

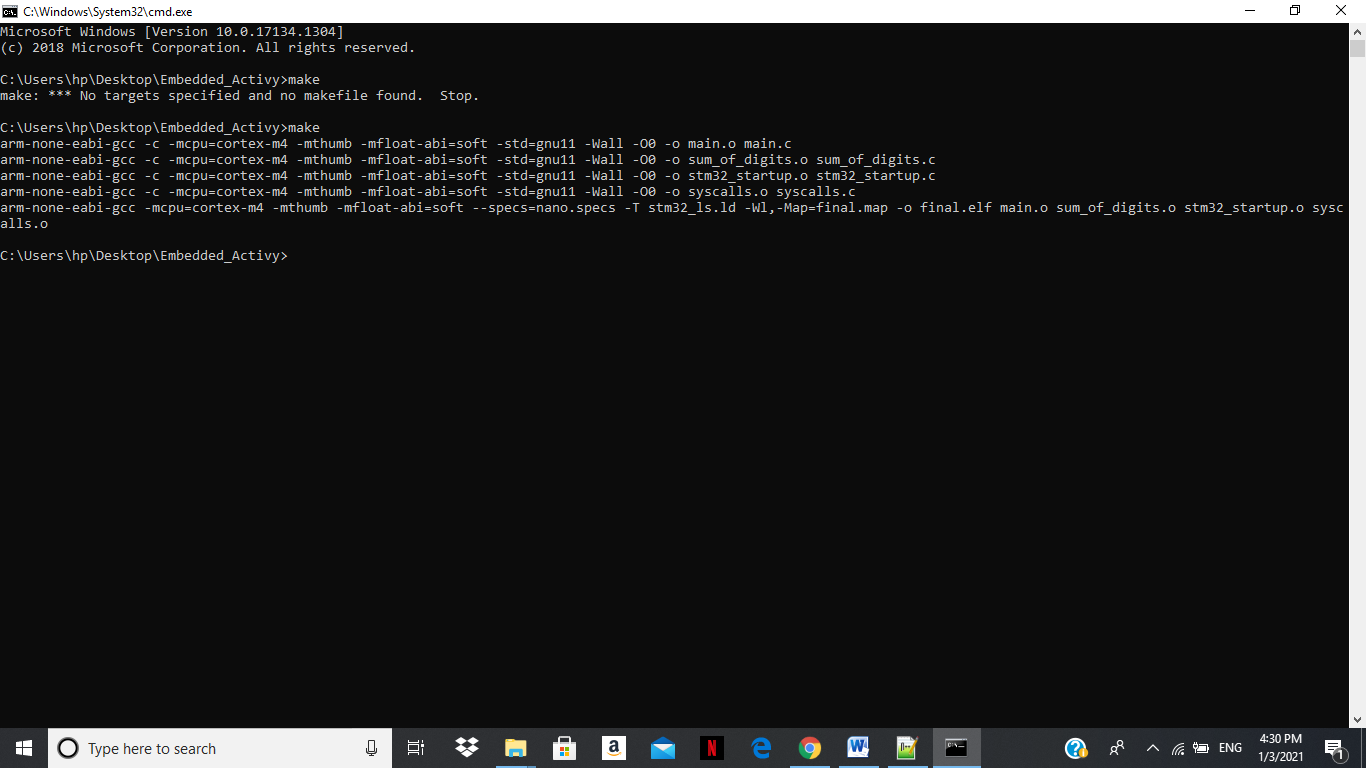


Fig. Executing Make command in cmd

# MINI PROJECT

## 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.

## Software and Hardware required

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

## Block Diagram

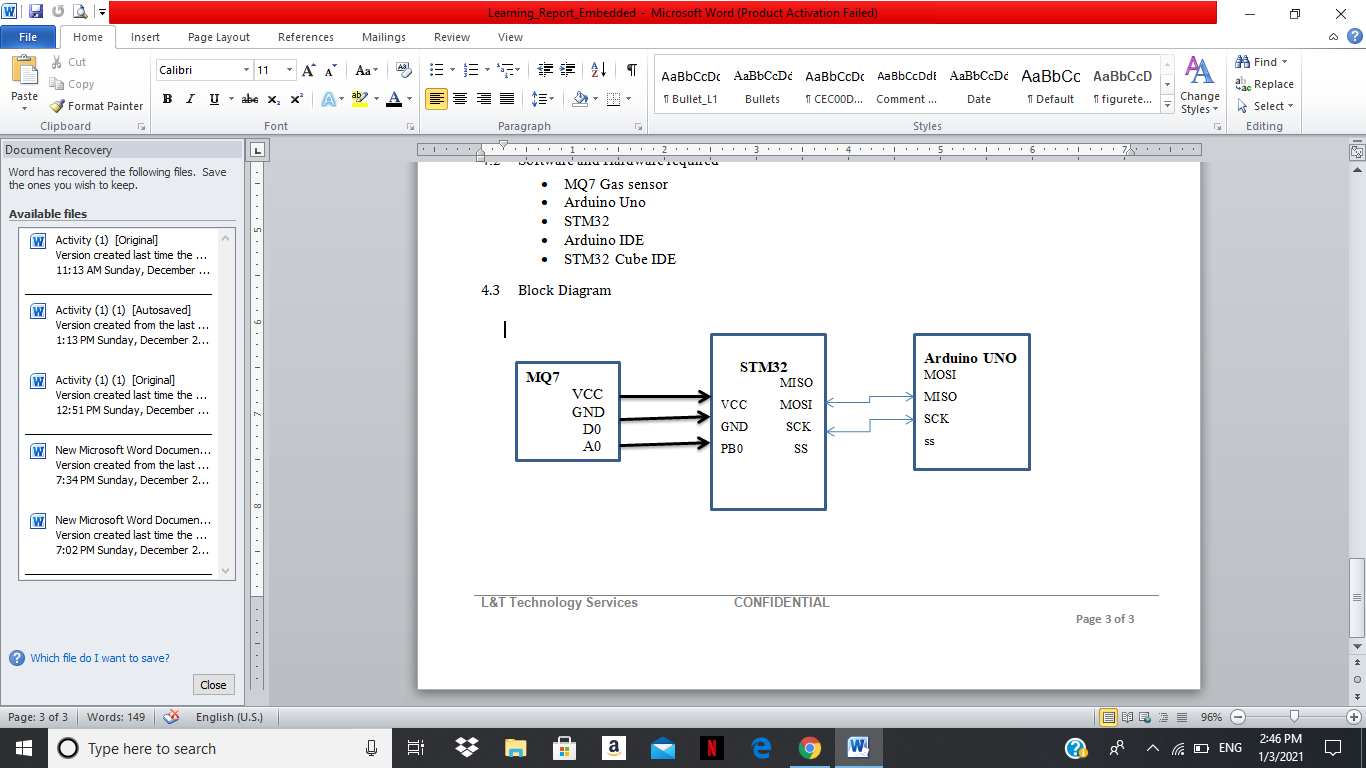
\

Fig. Block Diagram of Smoke Level Detector

## 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

## Results

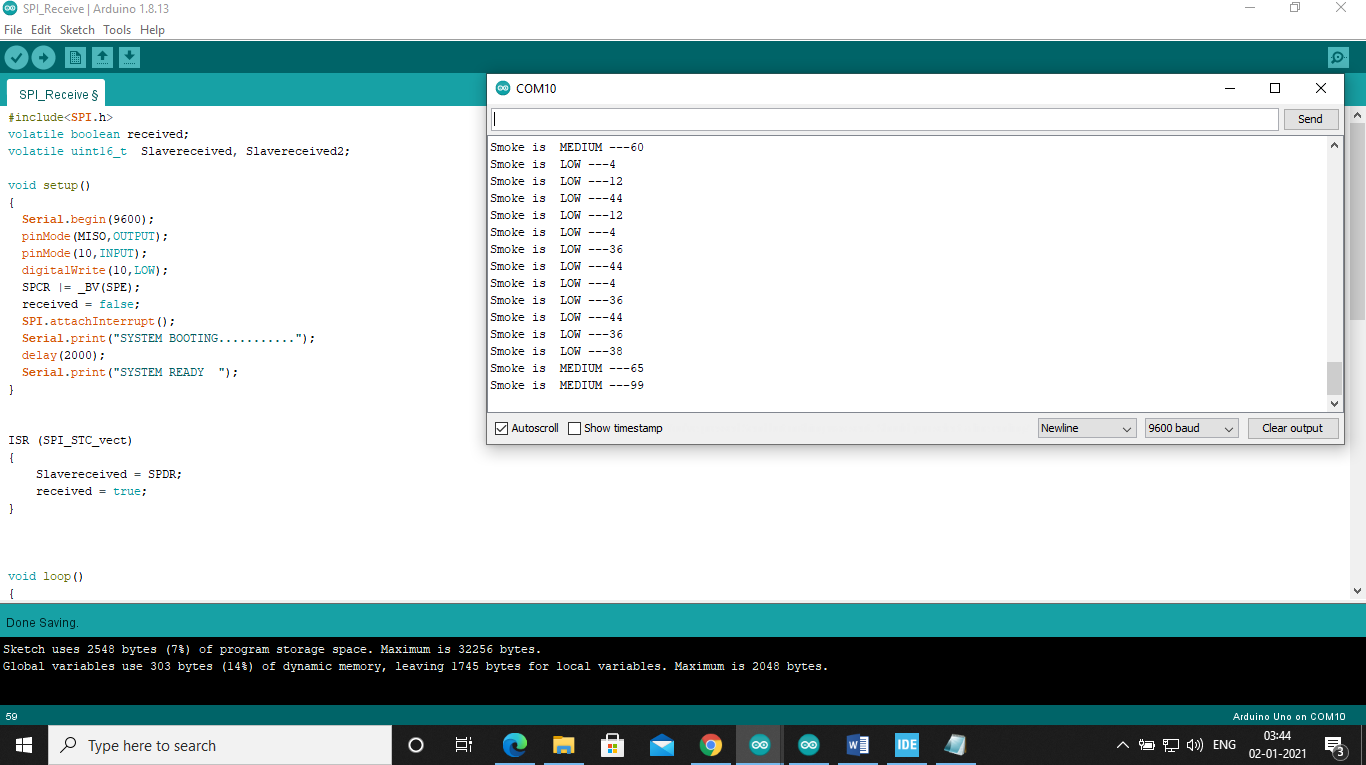


Fig. Arduino Receiving data using SPI

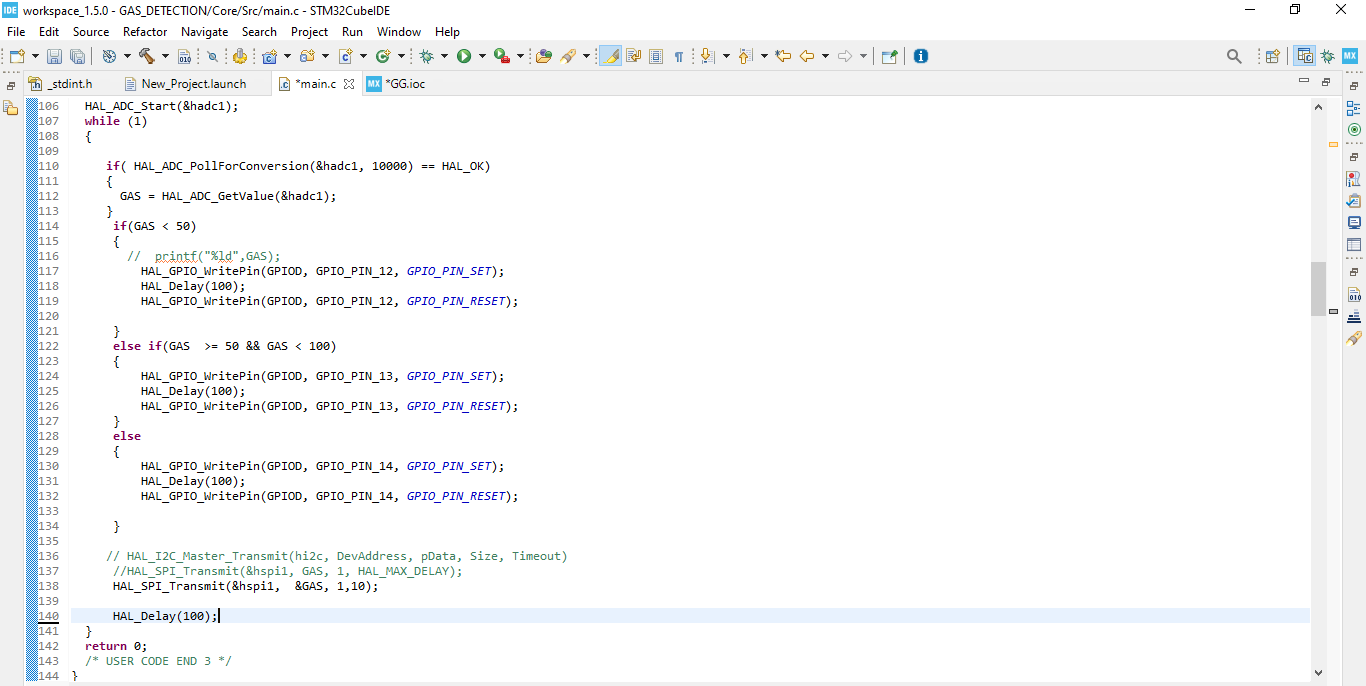


Fig. STM32 transmitting data

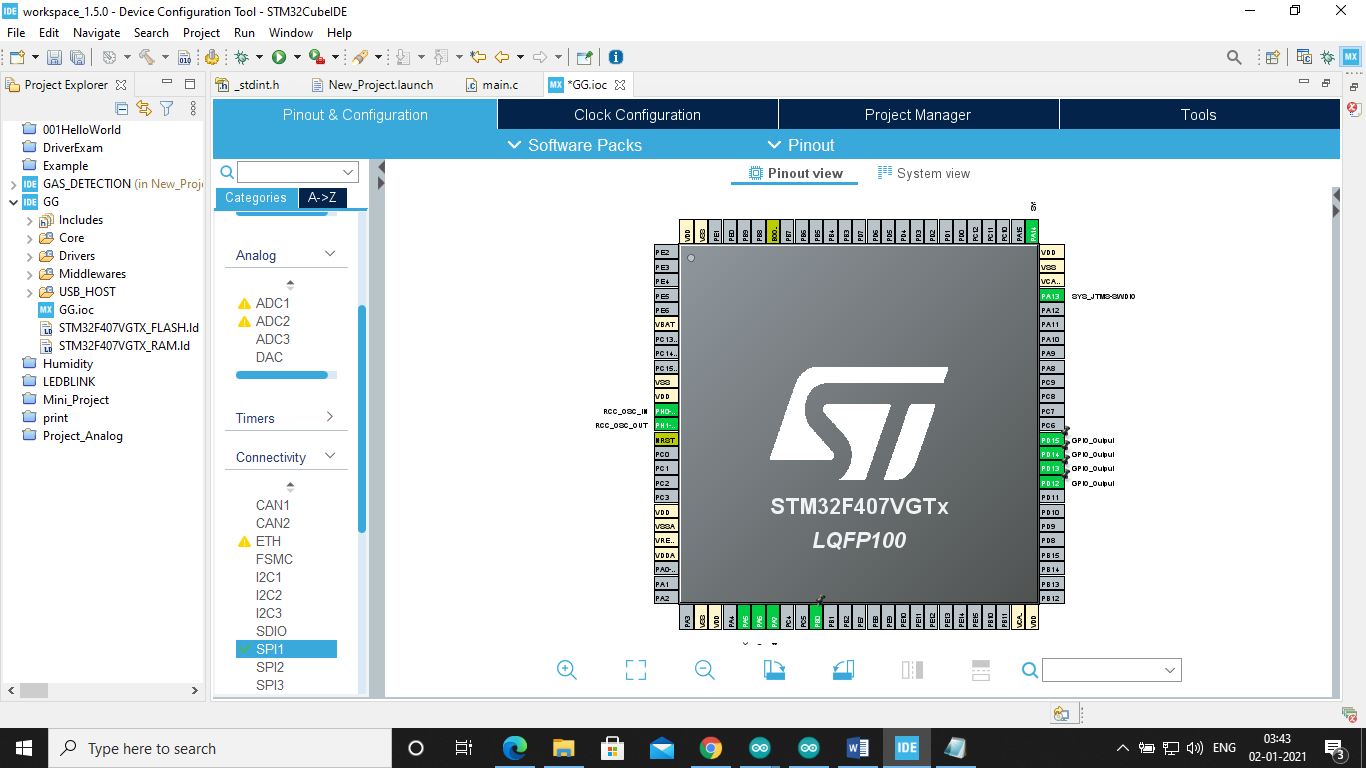


Fig. GUI of Project

GitHub Link: [99003158/Embedded\_C (github.com)](https://github.com/99003158/Embedded_C)