./

Learning Report

Course Code: EMBEDDED C



Version Number:

Name: PAVAN YADAV A

PS No: 99004494

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **Approved By** | **Remarks/Revision Details** |
|  | 16/06/2021 | PAVAN YADAV A |  |  |  |
|  | 17/06/2021 | PAVAN YADAV A |  |  |  |
|  | 18/06/2021 | PAVAN YADAV A |  |  |  |
|  | 19/06/2021 | PAVAN YADAV A |  |  |  |
|  | 20/06/2021 | PAVAN YADAV A |  |  |  |
|  | 21/06/2021 | PAVAN YADAV A |  |  |  |

Contents

[Activity 1 - Introduction to ARM Cortex-M and its Architecture 4](#_Toc75715878)

[**1.1 INTRODUCTION TO THE SOFTWARE: STM32CUBEIDE** 4](#_Toc75715879)

[**1.2 GETTING STARTED WITH THE SOFTWARE:** 4](#_Toc75715880)

[**1.3 BASICS OF STM32F407 DISCOVERY BOARD** 7](#_Toc75715881)

[Activity 2 – COMPILATION APROACH 7](#_Toc75715882)

[**2.1** **MAKE FILE** 7](#_Toc75715883)

[**2.2** **STARTUP CODE** 9](#_Toc75715884)

[**2.3** **LINKER SCRIPT** 11](#_Toc75715885)

[**2.4** **DEBUGGING TECHNIQUES** 12](#_Toc75715886)

[ACTIVITY 3- 13](#_Toc75715887)

[**3.1 Explore the base addresses of different types of Memory** 13](#_Toc75715888)

[**3.2 Explore base addresses of AHBx and APBx bus peripherals** 13](#_Toc75715889)

[**3.3 Explore base addresses of peripherals hanging on AHB1 bus** 13](#_Toc75715890)

[**3.4 Explore base addresses of peripherals hanging on APB1 bus** 13](#_Toc75715891)

[**3.5 Explore base addresses of peripherals hanging on APB2 bus** 14](#_Toc75715892)

[**3.6 Peripheral register definitions** 14](#_Toc75715893)

[**3.7 Simulator-QEMU** 15](#_Toc75715894)

[**3.8 Rule of optimization and its flags** 16](#_Toc75715895)

# Activity 1 - Introduction to ARM Cortex-M and its Architecture

## **1.1 INTRODUCTION TO THE SOFTWARE: STM32CUBEIDE**

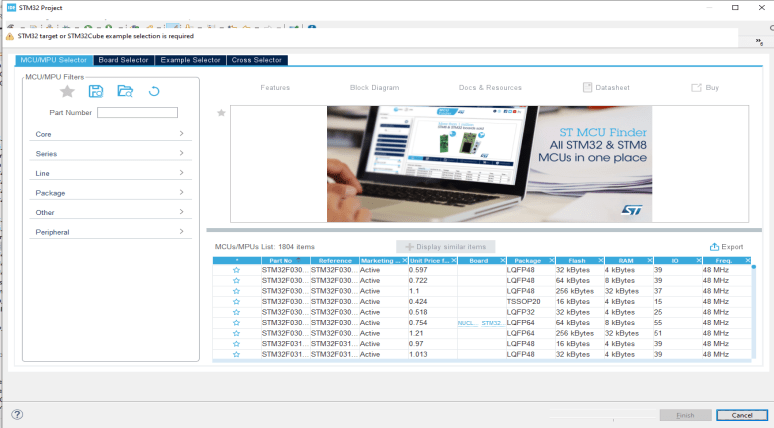
STM32CubeIDE is a multi-OS development tool, which is part of the STM32Cube system program. STM32CubeIDE is a state-of-the-art C / C ++ development platform with boundary modification, code production, code integration, and debugging features for STM32 microcontrollers and microprocessors. Based on the Eclipse® / CDT framework and the GCC development tools, as well as the GDB debugging. Allows integration of hundreds of existing plugins that complete the Eclipse® IDE features.

STM32CubeIDE combines STM32 configuration and project creation functionality from STM32CubeMX to provide the same tools experience and save installation time and development time. After the selection of the STM32 MCU or blank MPU, or a customized microcontroller or microprocessor from the board selection or model selection, the project is created, and a code is generated. At any time during development, the user can revert to implementation and configuration or middleware and update the startup code without affecting the user code.STM32CubeIDE includes build and layout analysts that provide the user with useful information about project status and memory requirements.STM32CubeIDE also includes advanced and advanced debugging features including basic CPU registers, memory, and road-related registers, as well as a flexible live clock, Serial Wire Viewer interface, or error analyzer.

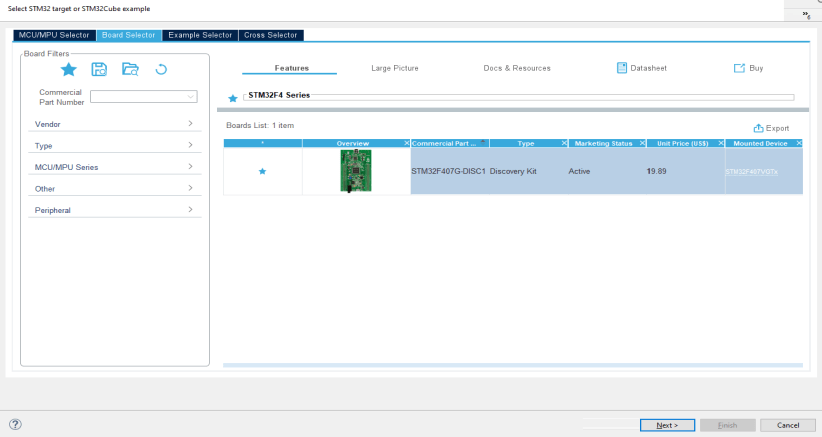
## **1.2 GETTING STARTED WITH THE SOFTWARE:**

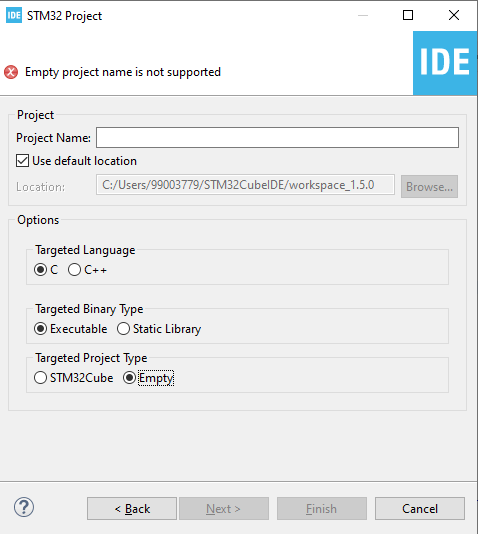
Start the software:

1. Go to file a New a STM32 Project.



1. Select the board and star it for frequent uses and start with the project.





1. Type the name of the project and start with the coding.

## **1.3 BASICS OF STM32F407 DISCOVERY BOARD**

1. Board name: STM32F4 DISCOVERY board
2. Microcontroller name: STM32F407VG MCU
3. Processor name: ARM Cortex –M4 with FPU
4. Range of addresses from which D-bus pick: 0x00000000 to 0x1FFFFFFF
5. Range of addresses from which I-bus pick: 0x00000000 to 0x1FFFFFFF
6. Range of addresses from which S-bus pick: 0x20000000 to 0xDFFFFFFF and 0xE0100000 to 0xFFFFFFFF
7. Number of total pins in Microcontroller:100
8. Number of ports in Microcontroller:5
9. Number of pins on each port:16

# Activity 2 – COMPILATION APROACH

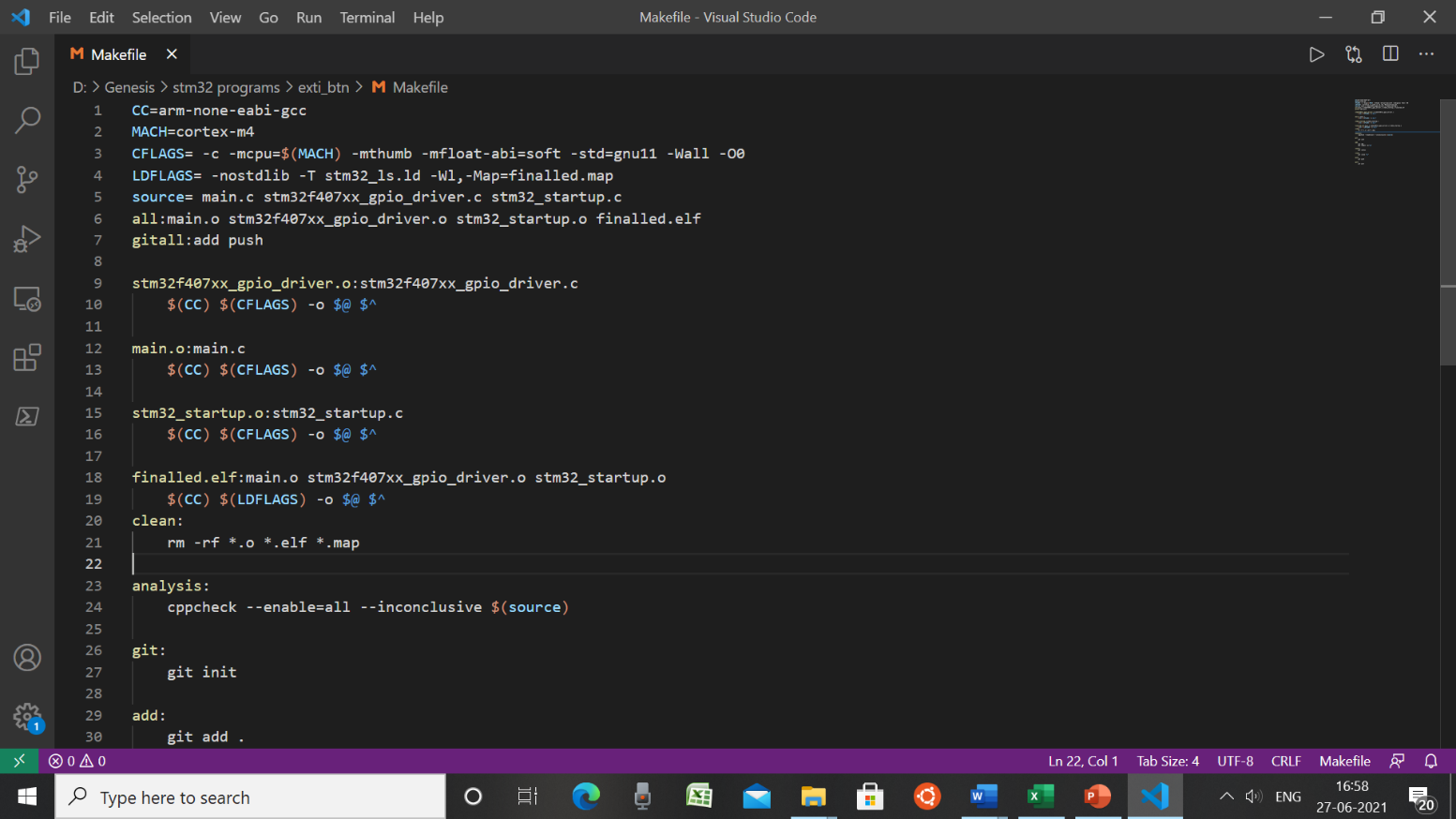
This is a complete process for integrating the ARM Cortex Mx processor sample system.

The following are the integration stages of plan C:

1. Preprocessor stage
2. Compilation stage
3. Assembly stage
4. Linking stage

## **MAKE FILE**

Below is the make file for the sample program:



**Fig 1. Make file.**

* arm-none-eabi-gcc: invokes cross compiler.
* -mcpu = cortex-m4 used to select our cortex-m4 processor used
* -O0 : optimization for compilation time
* -std=gnu11: gnu standard
* -Wall: turns on all most used compiler warnings
* -mthumb: used to generate output code in ARM state
* main. o: redirect file
* main.c stm32f407xx\_gpio\_driver.c stm32\_startup.c: source files
* finalled.elf: generated binary file

The command to run this make file in the command prompt is:

A picture containing text, electronics, computer, screenshot

Description automatically generated

**Fig 2. Run make file in command prompt**

## **STARTUP CODE**

• The startup file is responsible for setting the correct code usage fields in the main.c file.

• Part of the start code depends on the direction (processed)

• Role of startup file:

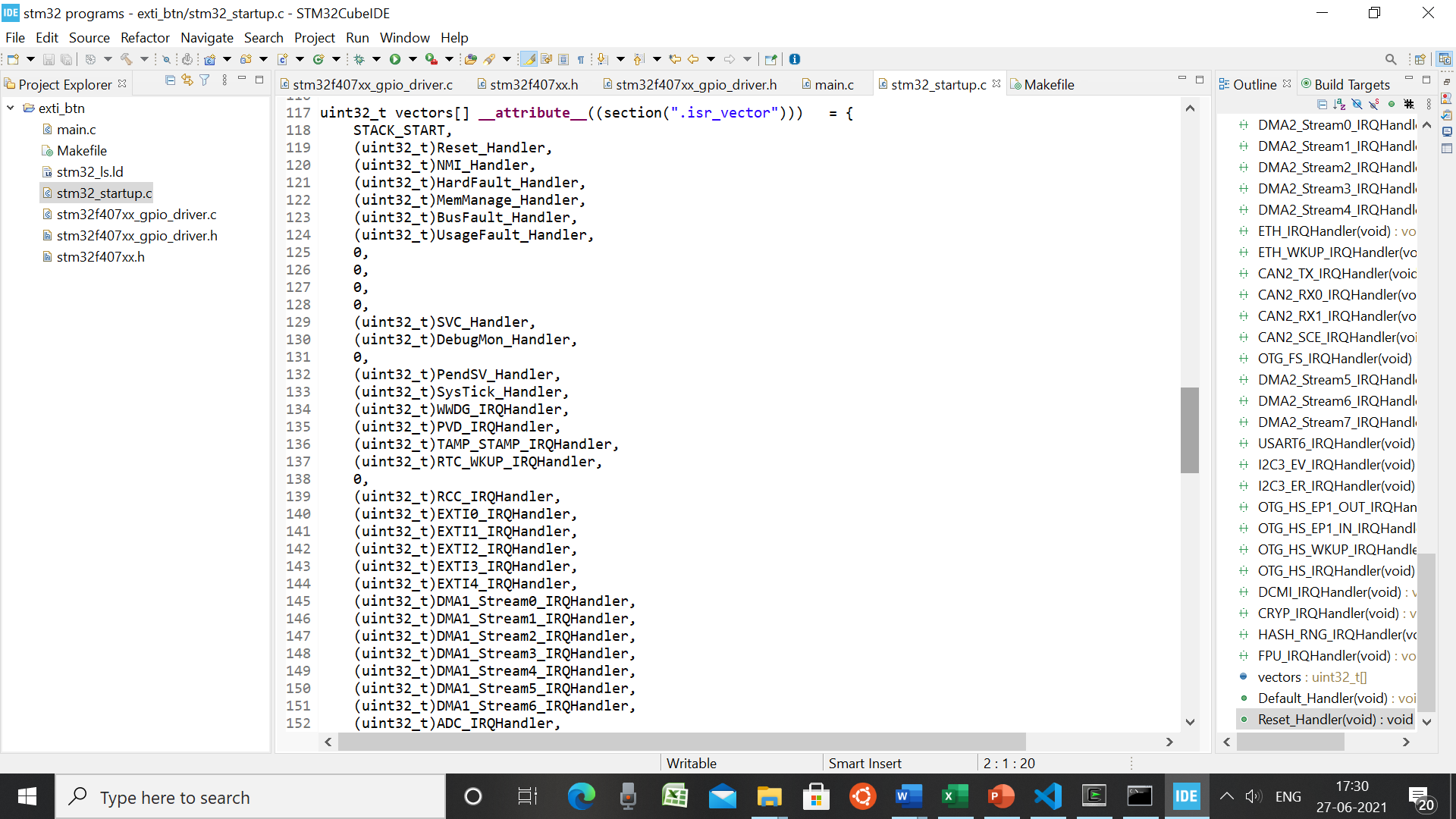
1. Create a special MCU microcontroller vector table.

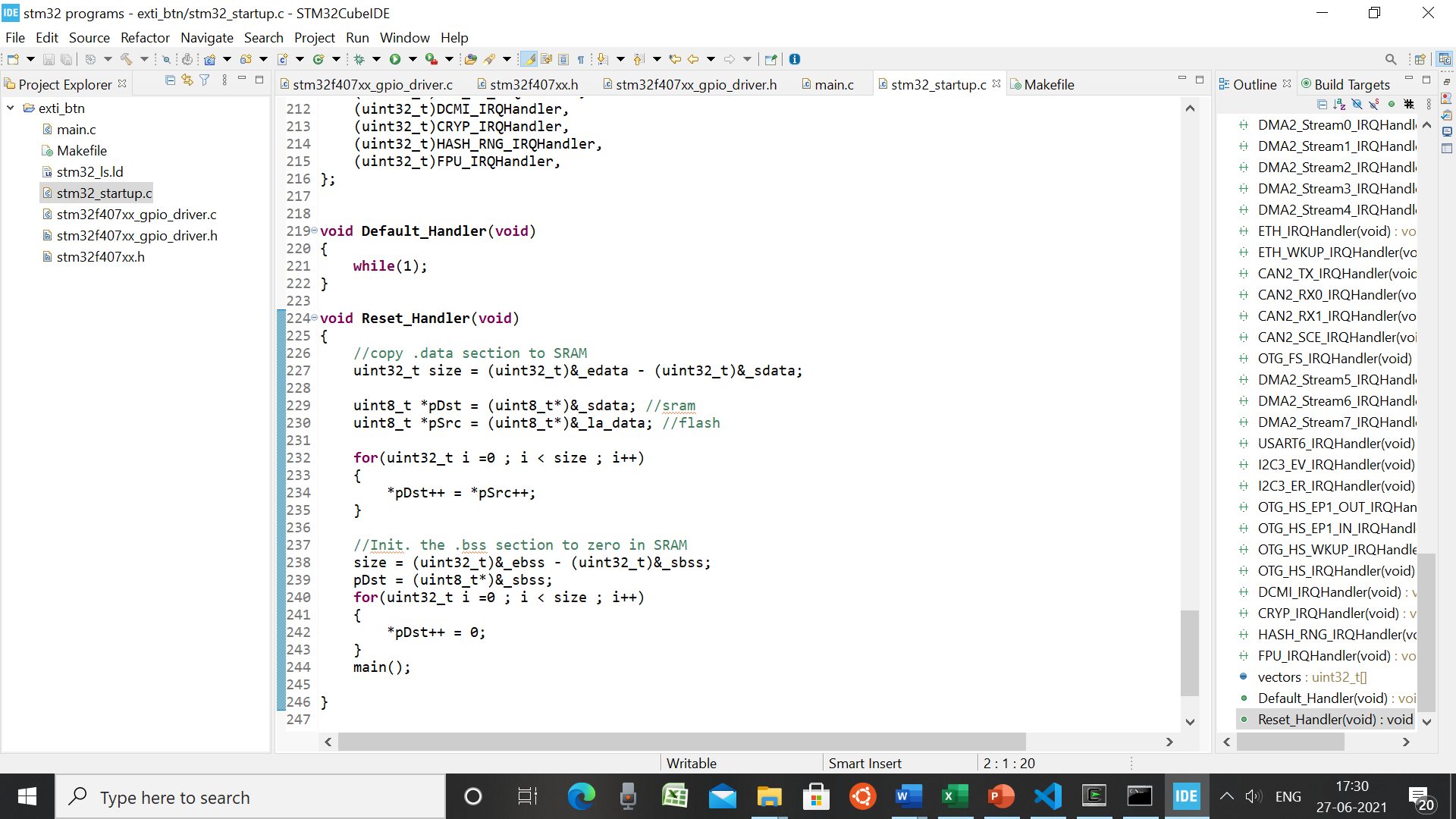
2. Write the startup code that starts the data section and. bss in SRAM.

3. Call the main ()

Graphical user interface, text

Description automatically generated





**Fig 3. Startup code**

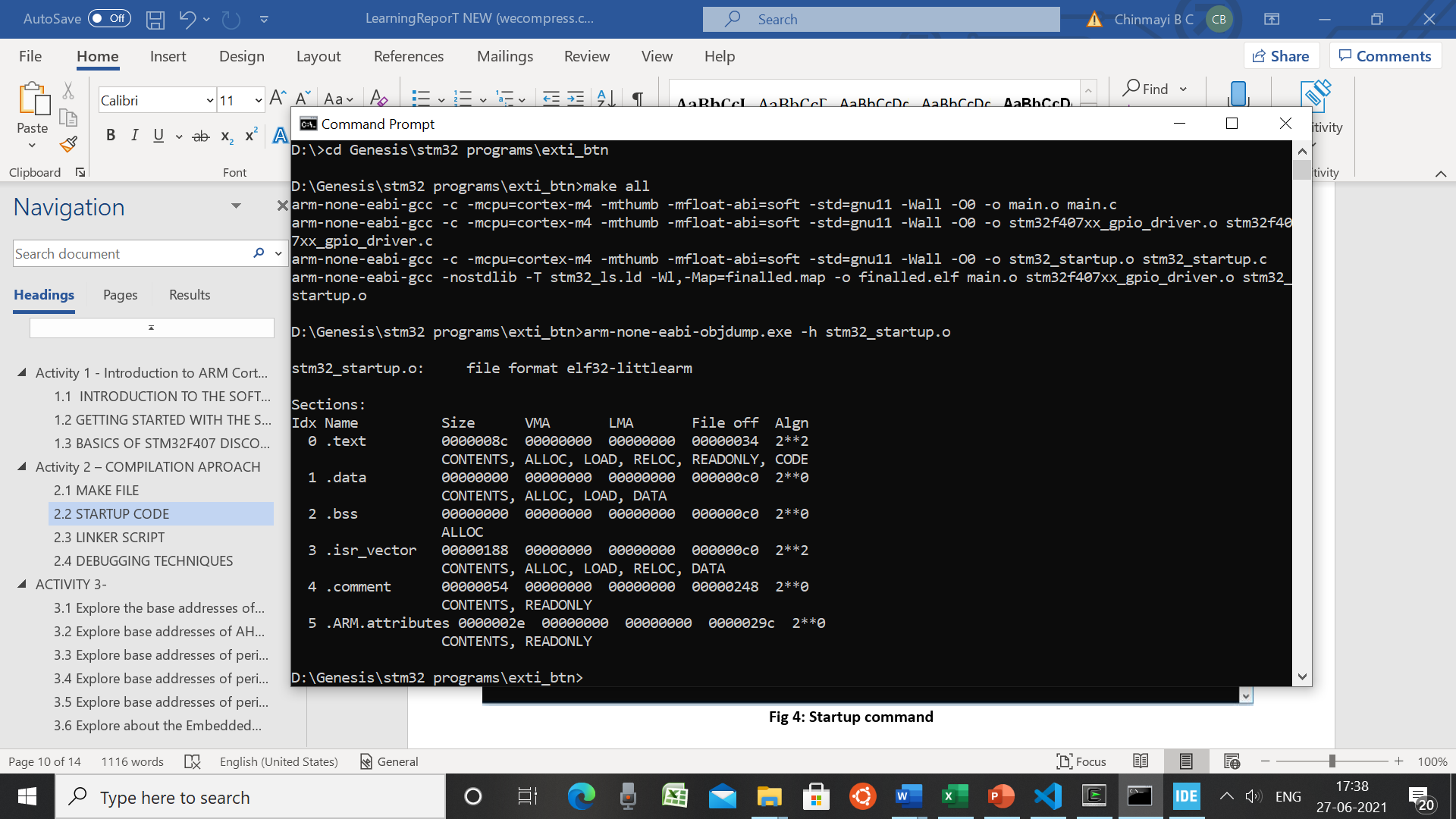
In the startup code we use the attributes to keep the variable in the user-defined function.

**Job qualifications:**

• **Weakness**: Allows the app to override a previously defined weak function (dummy function) with the same function name.

• **Alias**: Lets programmer provides any alias name for the same function.

A startup. o file made in usable format, the various components of which are shown below:

****

**Fig 4: Startup command**

## **LINKER SCRIPT**

• Linkers take one or more object files or libraries as input and combine them to create a single usable file as output.

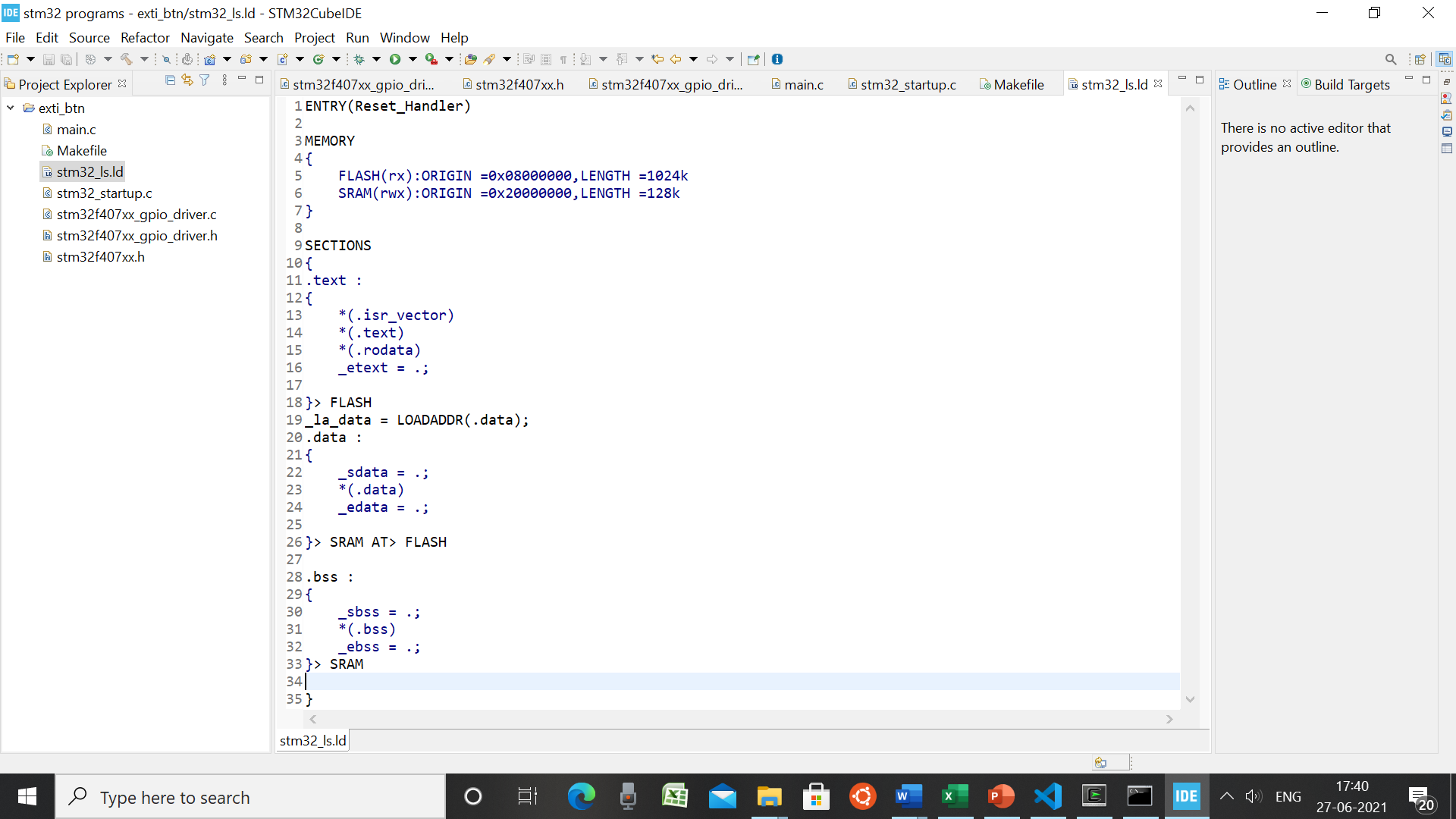
• Linker scripts determine how different categories of object files should be compiled to create an output file.

• Reset holder is the login of the application

• The login command is used to set the "login address" information in the last elf file file created.

Syntax: Login (brand sign)

Login (Reset\_Manager)



**Fig 5: Linker script**

## **DEBUGGING TECHNIQUES**

• STM32F407VG embedded in chip debugger to fix code error.

• OCD ON-Chip Debugger aims to provide bug fixes, program settings and test parameters for embedded targeted devices.

• OCD is a free and resource-based application that allows you to edit, correct, and analyze your plans using GDB.

• Supports various target boards based on different processor properties.

# ACTIVITY 3-

## **3.1 Explore the base addresses of different types of Memory**

#define FLASH\_BASEADDR 0x08000000U // flash memory base address

#define SRAM1\_BASEADDR 0x20000000U // sram1 memory base address

#define SRAM2\_BASEADDR 0x2001C000U // sram2 memory base address

#define ROM\_BASEADDR 0x1FFF0000U // ROM memory base address

#define SRAM\_BASEADDR SRAM1\_BASEADDR // SRAM memory base address

## **3.2 Explore base addresses of AHBx and APBx bus peripherals**

#define PERIPH\_BASEADDR 0x40000000U

#define APB1PERIPH\_BASEADDR PERIPH\_BASEADDR

#define APB2PERIPH\_BASEADDR 0x40010000U

#define AHB1PERIPH\_BASEADDR 0x40020000U

#define AHB2PERIPH\_BASEADDR 0x50000000U

## **3.3 Explore base addresses of peripherals hanging on AHB1 bus**

#define GPIOA\_BASEADDR (AHB1PERIPH\_BASEADDR + 0x0000)

#define GPIOB\_BASEADDR (AHB1PERIPH\_BASEADDR + 0x0400)

#define GPIOC\_BASEADDR (AHB1PERIPH\_BASEADDR + 0x0800)

#define GPIOD\_BASEADDR (AHB1PERIPH\_BASEADDR + 0x0C00)

#define GPIOE\_BASEADDR (AHB1PERIPH\_BASEADDR + 0x1000)

#define GPIOF\_BASEADDR (AHB1PERIPH\_BASEADDR + 0x1400)

#define GPIOG\_BASEADDR (AHB1PERIPH\_BASEADDR + 0x1800)

#define GPIOH\_BASEADDR (AHB1PERIPH\_BASEADDR+ 0x1C00)

#define GPIOI\_BASEADDR (AHB1PERIPH\_BASEADDR + 0x2000)

#define RCC\_BASEADDR (AHB1PERIPH\_BASEADDR + 0x3800)

## **3.4 Explore base addresses of peripherals hanging on APB1 bus**

#define I2C1\_BASEADDR (APB1PERIPH\_BASE+ 0x5400)

#define I2C2\_BASEADDR (APB1PERIPH\_BASE+ 0x5800)

#define I2C3\_BASEADDR (APB1PERIPH\_BASE+ 0x5C00)

#define SPI2\_BASEADDR (APB1PERIPH\_BASE+ 0x3800)

#define SPI3\_BASEADDR (APB1PERIPH\_BASE+ 0x3C00)

#define USART2\_BASEADDR (APB1PERIPH\_BASE+ 0x4400)

#define USART3\_BASEADDR (APB1PERIPH\_BASE+ 0x4800)

#define UART4\_BASEADDR (APB1PERIPH\_BASE+ 0x4C00)

#define UART5\_BASEADDR (APB1PERIPH\_BASE+ 0x5000)

## **3.5 Explore base addresses of peripherals hanging on APB2 bus**

#define EXTI\_BASEADDR (APB2PERIPH\_BASE+ 0x3C00)

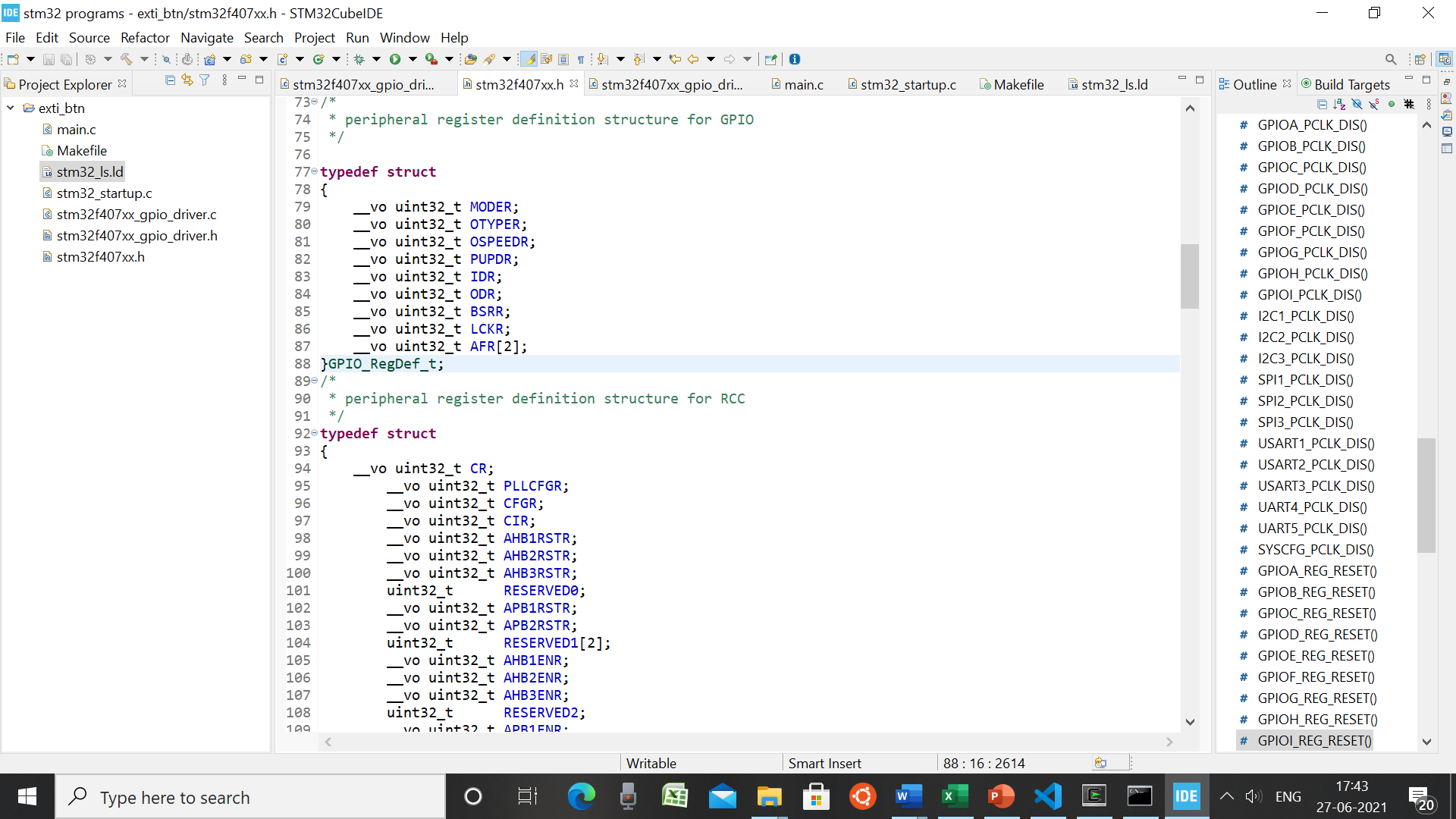
#define SPI1\_BASEADDR (APB2PERIPH\_BASE+ 0x3000)

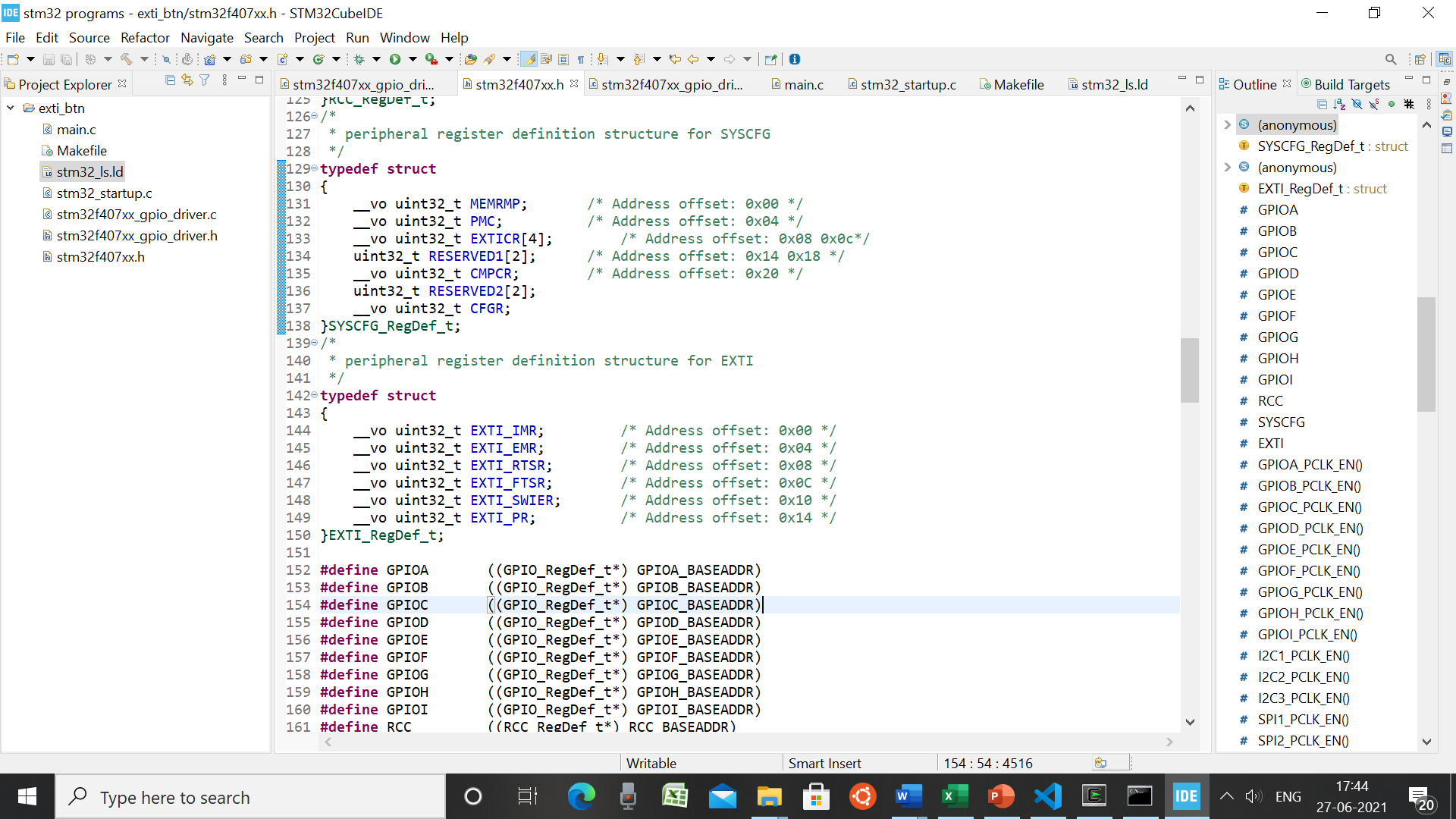
#define USART1\_BASEADDR (APB2PERIPH\_BASE+ 0x1000)

#define USART6\_BASEADDR (APB2PERIPH\_BASE+ 0x1400)

#define SYSCFG\_BASEADDR (APB2PERIPH\_BASE+ 0x3800)

## **3.6 Peripheral register definitions**





**Fig 6: Peripheral register definition**

## **3.7 Simulator-QEMU**

Command to dump code on QEMU emulatorA screenshot of a computer

Description automatically generated

**Fig 6: Command to open virtual simulator.**

**A picture containing text, electronics, circuit

Description automatically generated**

**Fig 7: Output on virtual board**

## **3.8 Rule of optimization and its flags**

gcc -O sets the compiler's *optimization* level.

Graphical user interface, text, application

Description automatically generated

**Fig 8: GCC optimization options**

These options are used to set compiler’s optimization level. +increase ++increase more +++increase even more -reduce --reduce more ---reduce even more.