Learning Report – Embedded C

LTTS
GLOBAL
ENGINEERING
ACADEMY

L&T Technology Services

Document History

Ver. Rel. No.	Release Date	Prepared. By	Reviewed By	Approved By	Remarks/Revision Details
	18-02-2021	Saloni Adanna		Dr Vivek Kaundal & Bhargav N	
	19-02-2021	Saloni Adanna		Dr Vivek Kaundal & Bhargav N	
	20-02-2021	Saloni Adanna		Dr Vivek Kaundal & Bhargav N	
	21-02-2021	Saloni Adanna		Dr Vivek Kaundal & Bhargav N	
	22-02-2021	Saloni Adanna		Dr Vivek Kaundal & Bhargav N	
	23-02-2021	Saloni Adanna		Dr Vivek Kaundal & Bhargav N	

TABLE OF TABLES

Activity 1

1.1 Introduction

A sample c application program is taken and its cross compilation is done. Cross compilation is a process where compiling is staged and each stage produces a file which can be viewed. The cross-compilation process involves in converting a main code into.asm,. s and .elf files. These are the files produced at each stage of cross compilation. The below image shows the steps involved in cross compilation.

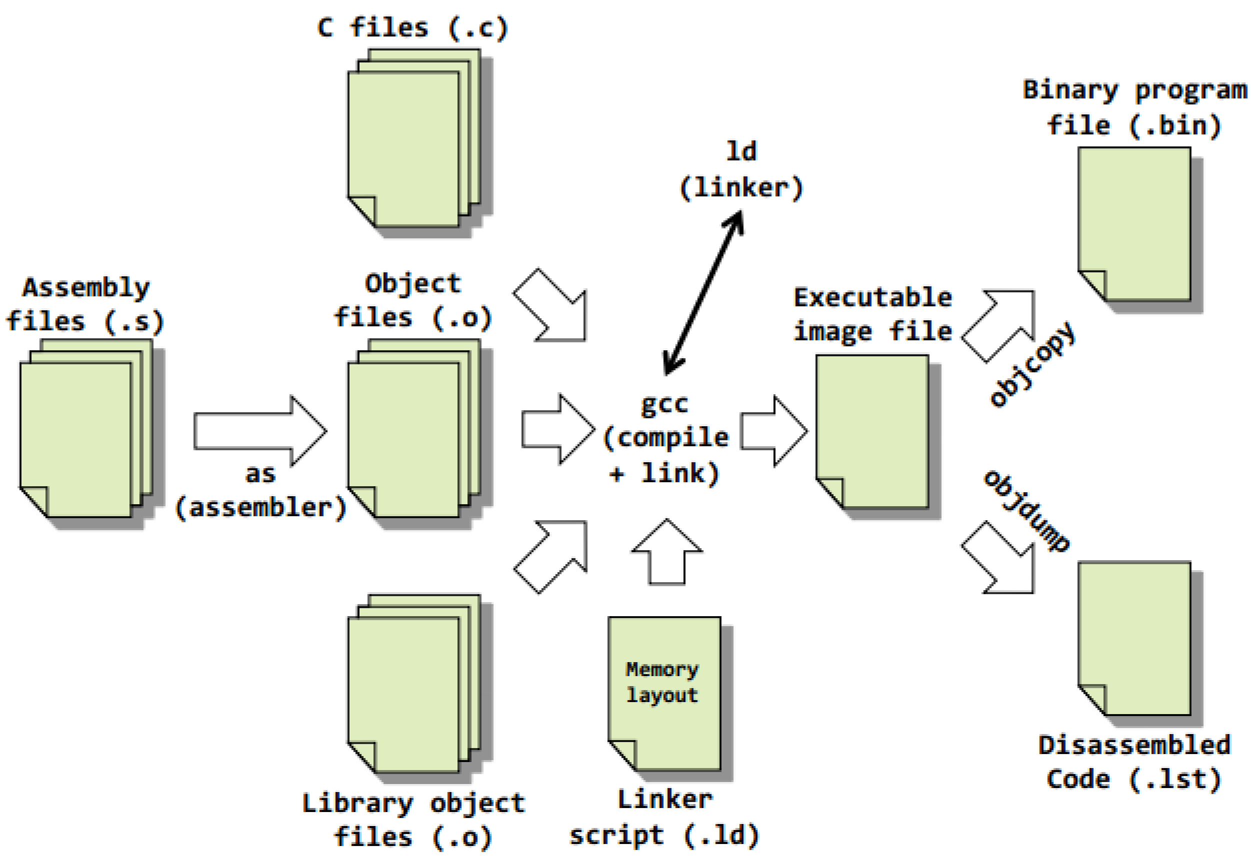


Figure 1:

1. Creating Makefile

Makefiles contains basically the shell comments. Compiling the source code files can be tiring, especially when you have to include several source files and type the compiling command every time you need to compile. Makefiles are the solution to simplify this task. Makefiles are special format files that help build and manage the projects automatically. Its acts as an automation tool.

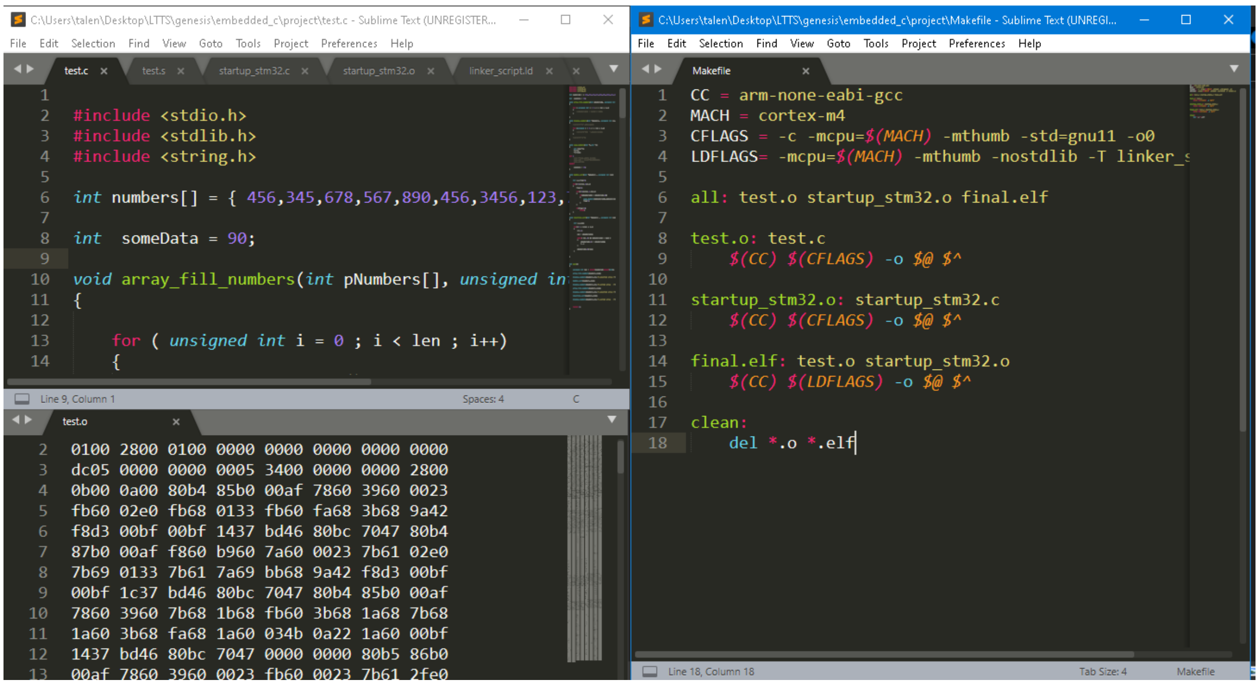


Figure 2:

The above figure 2 shows the sample c program along with make file which has generated an object file of the given program which is an intermediate compilation stage.

2. Startup File

Startup files are usually in your home directory. Their names begin with a period, which keeps the ls command from displaying them under normal circumstances. None of the files are required; all the affected programs are smart enough to use defaults when the file does not exist. Figure 3 shows the start-up file for the given application. It tells the computer to where to start the program from and also the address location. Figure 4 shows the object file created from the start-up file.

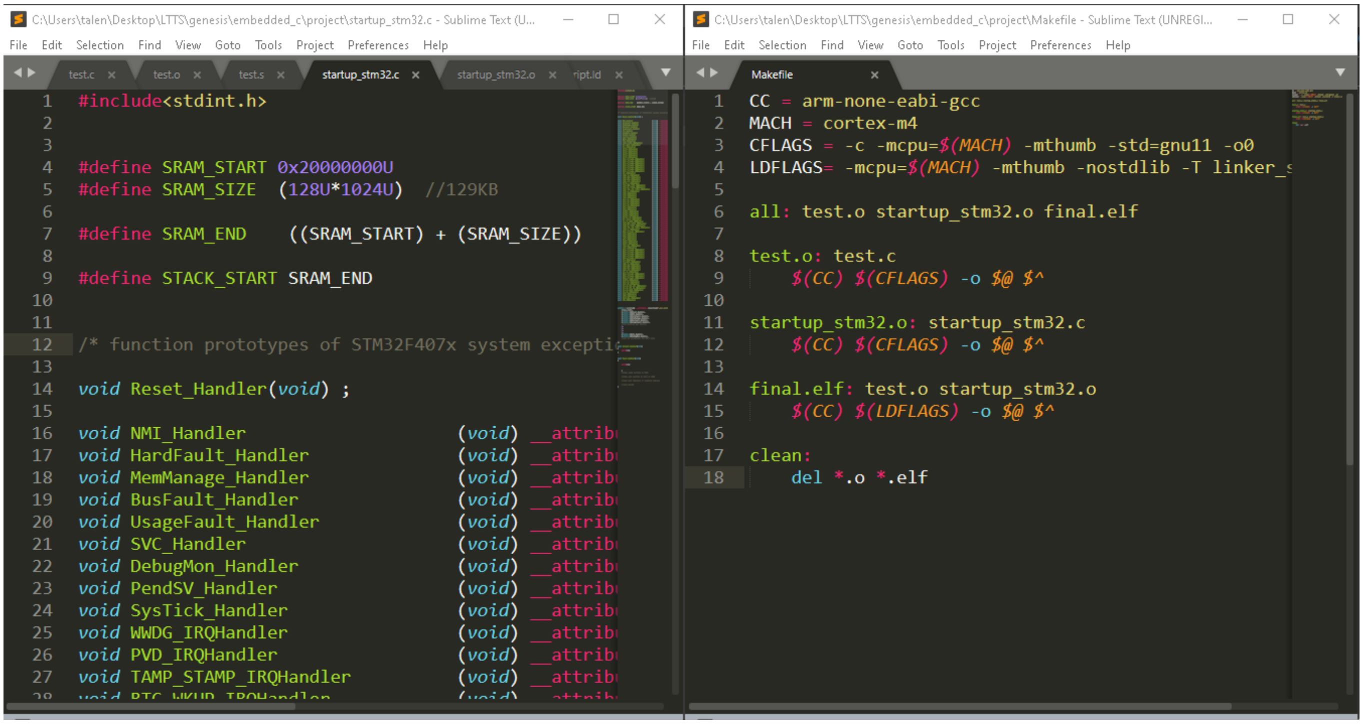


Figure 3:

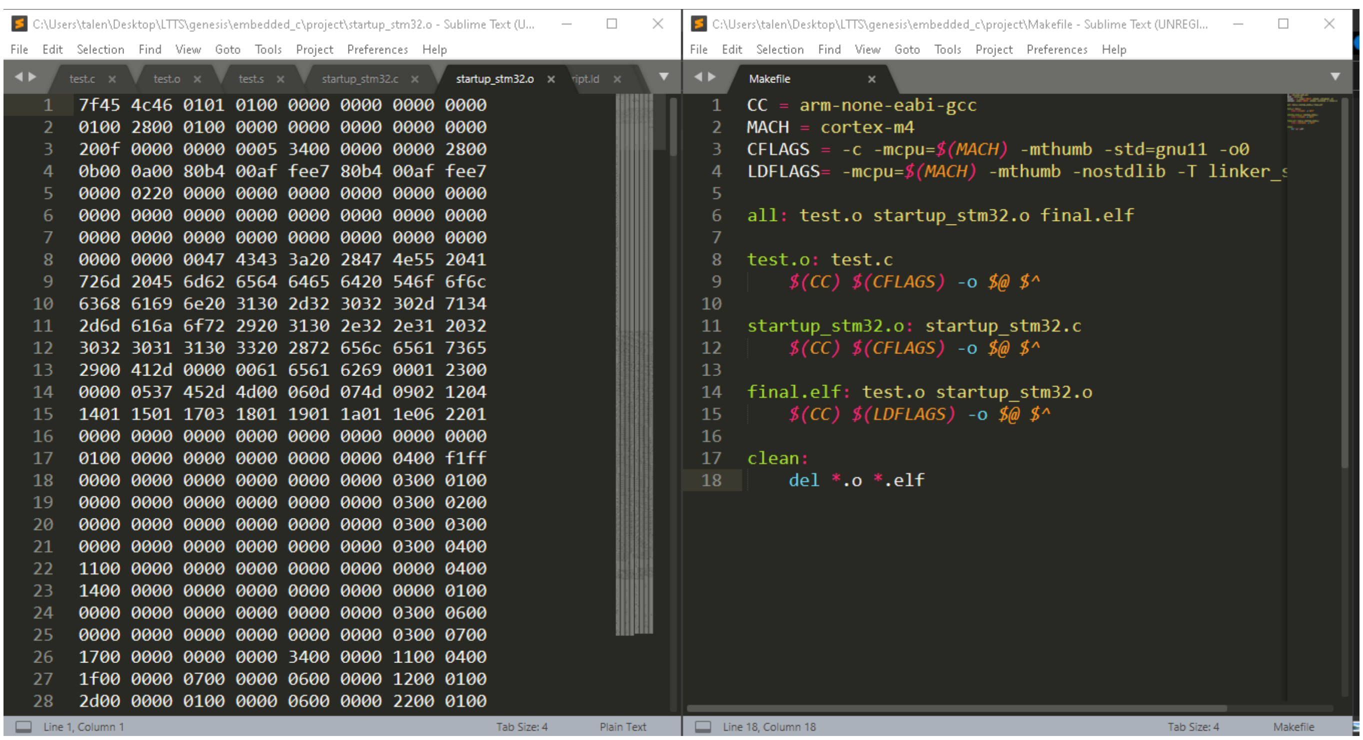


Figure 4:

3. Linker Scripts

Every link is controlled by a linker script. This script is written in the linker command language. The main purpose of the linker script is to describe how the sections in the input files should be mapped into the output file, and to control the memory layout of the output file. Most linker scripts do nothing more than this. However, when necessary, the linker script can also direct the linker to perform many other operations, using the commands described below. The linker always uses a linker script. If you do not supply one yourself, the linker will use a default script that is compiled into the linker executable. You can use the '--verbose' command-line option to display the default linker script. Certain command-line options, such as '-r' or '-N', will affect the default linker script.

You may supply your own linker script by using the '-T' command line option. When you do this, your linker script will replace the default linker script.

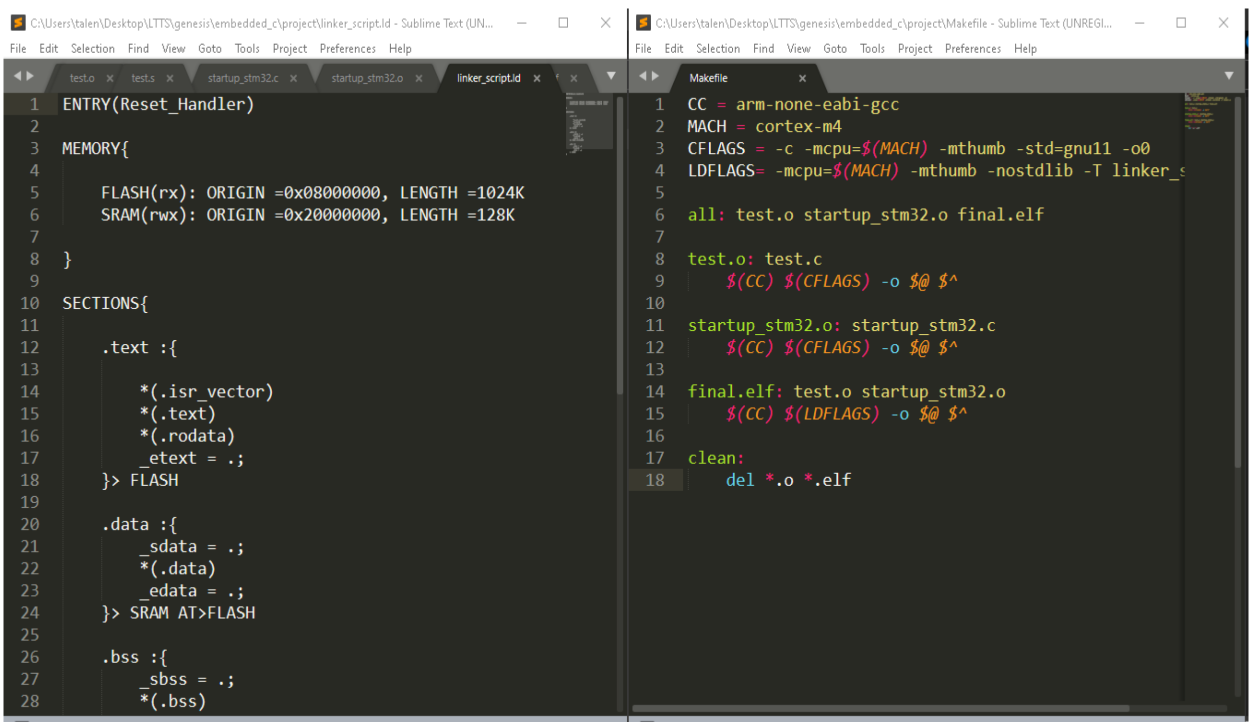


Figure 5:

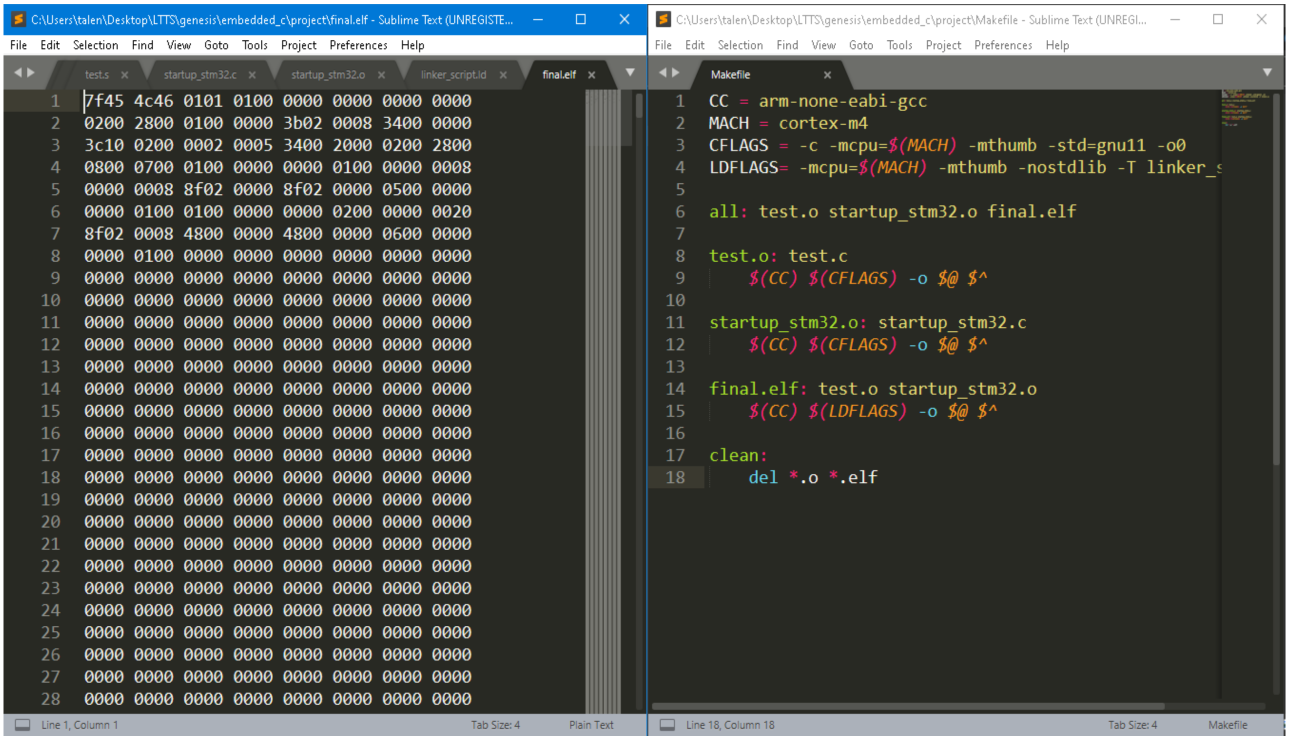


Figure 6

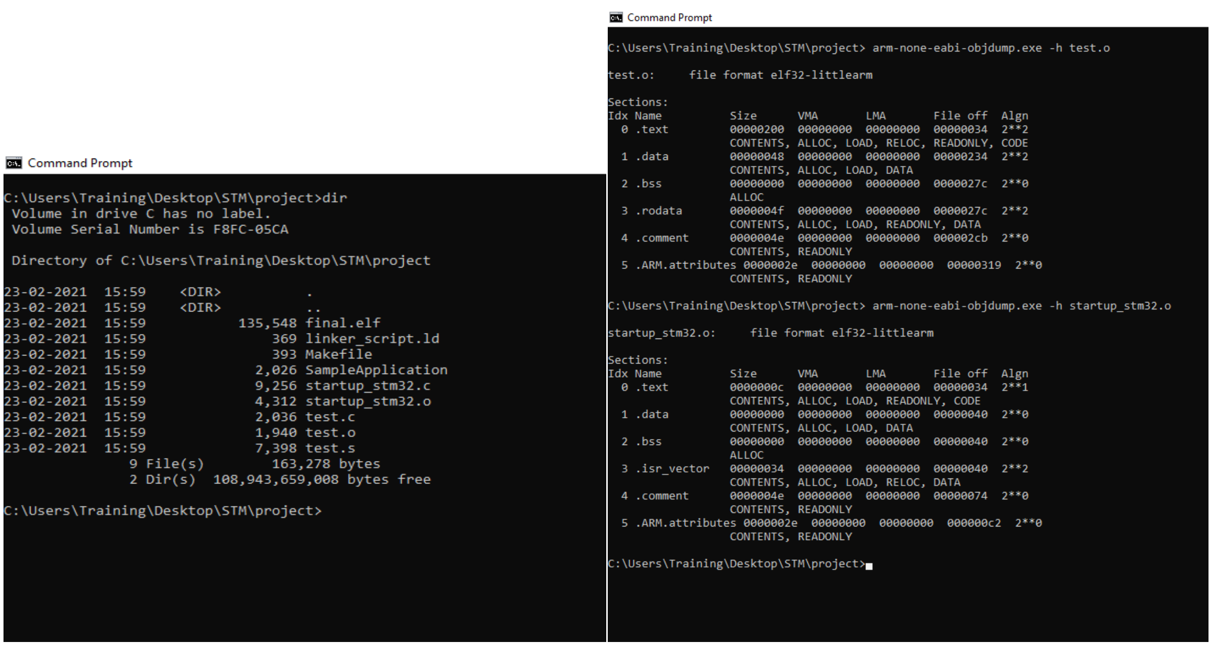


Figure 7

Activity 2

1. GPIO

GPIO'S also known as general purpose input output are the ports that are available on any micro-controller. These vary from one micro controller to other. GPIO's basically helps in controlling the input and output. But sometimes these ports are used for special functioning too. Depending on the size of micro controller they vary from 8 bit to 32,64 bits. In the below project, stm32f407, a 32-bit micro controller is used to toggle a LED at the output of one of the GPIO ports. This has been done using the stm IDE where the headers and main code is generated automatically.

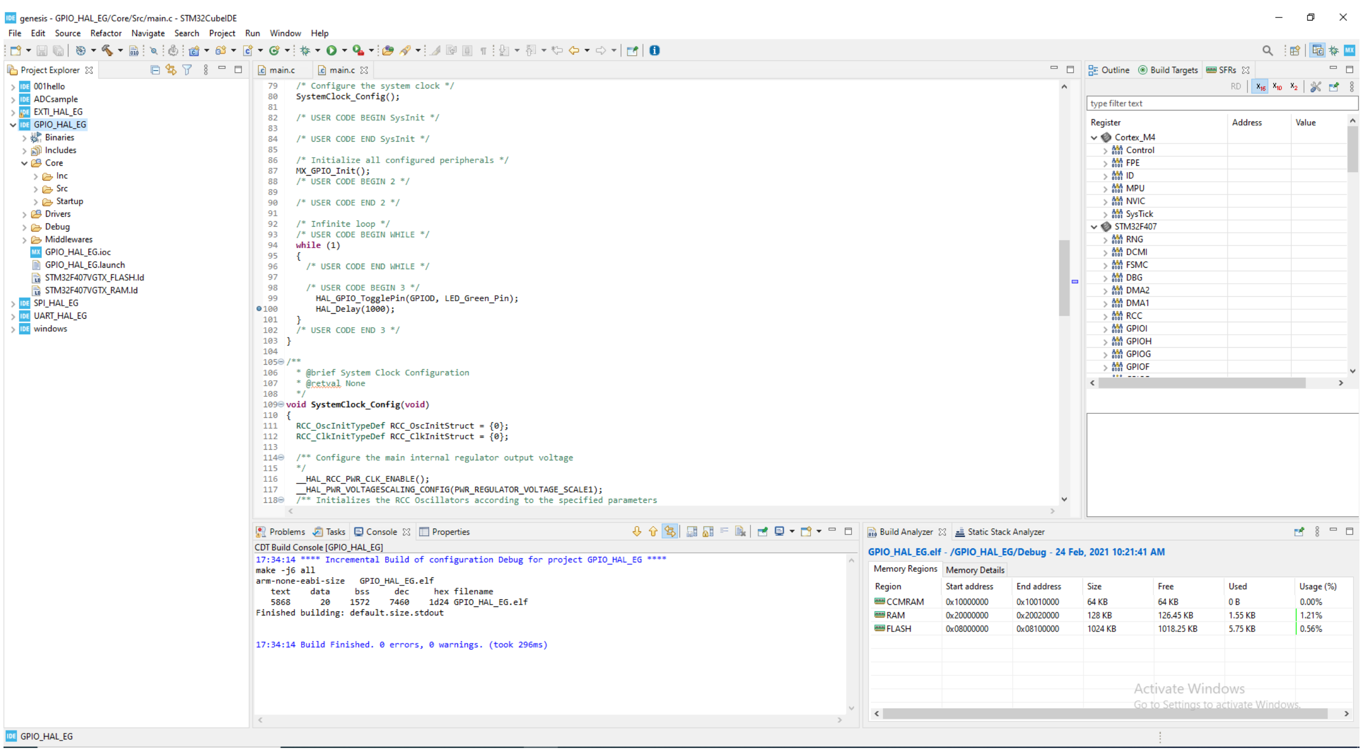


Figure 8

2. EXT INTERRUPTS

Interrupts are a piece of code which halts the main function and tells to execute this piece of code. Once the interrupt is executed, the main function continues from where it has started. Depending on the micro-controller interrupts are present.

There are two types of internals in a micro controller. Ones that control from inside are the internal interrupts and the ones that are controlled externally like buttons, switches are called external interrupts. In this stm32 board we are controlling the LED through an external interrupt.

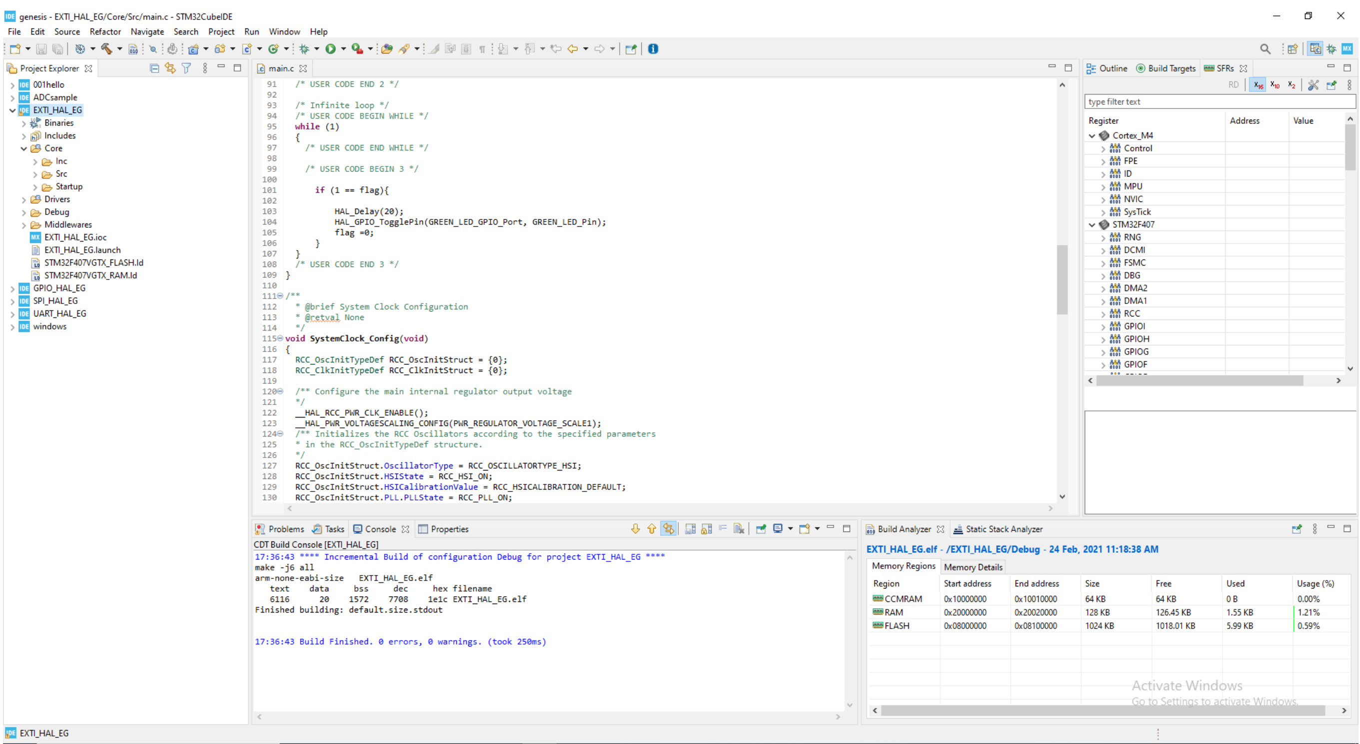
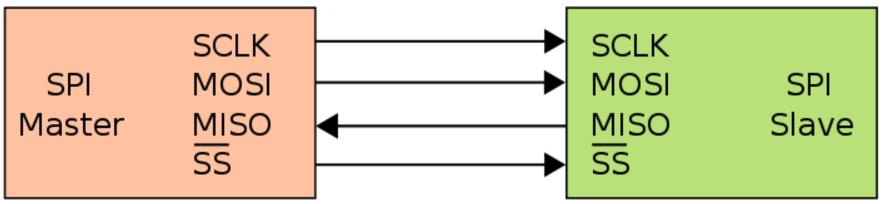
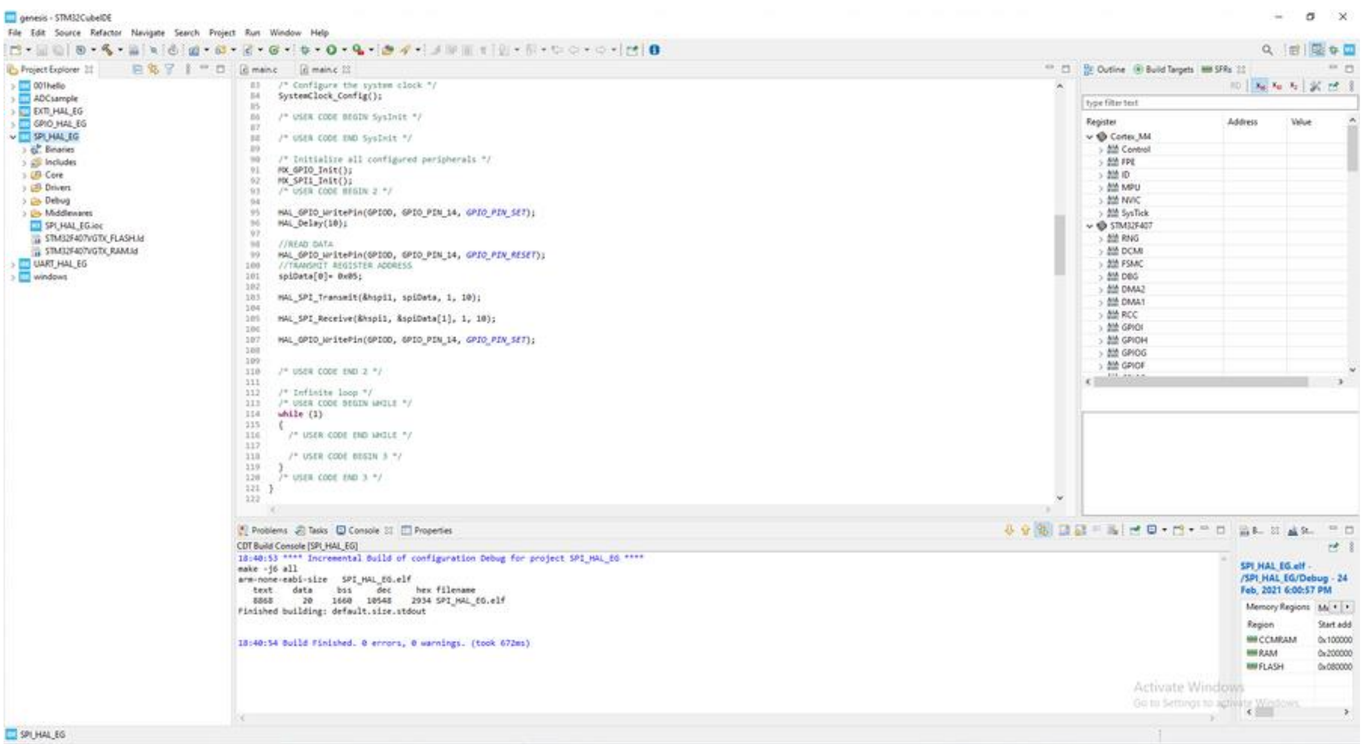


Figure 9

3. SPI

SPI is a serial communication protocol. The full form is Synchronous Data transfer Protocol. These are used for high -speed communication. SPI operation based upon shift registers. Master or Slave has an 8bit registers inside it. SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device originates the frame for reading and writing. Multiple slave-devices are supported through selection with individual slave select (SS), sometimes called chip select (CS), lines.

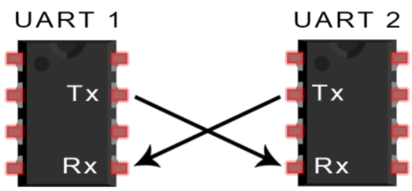


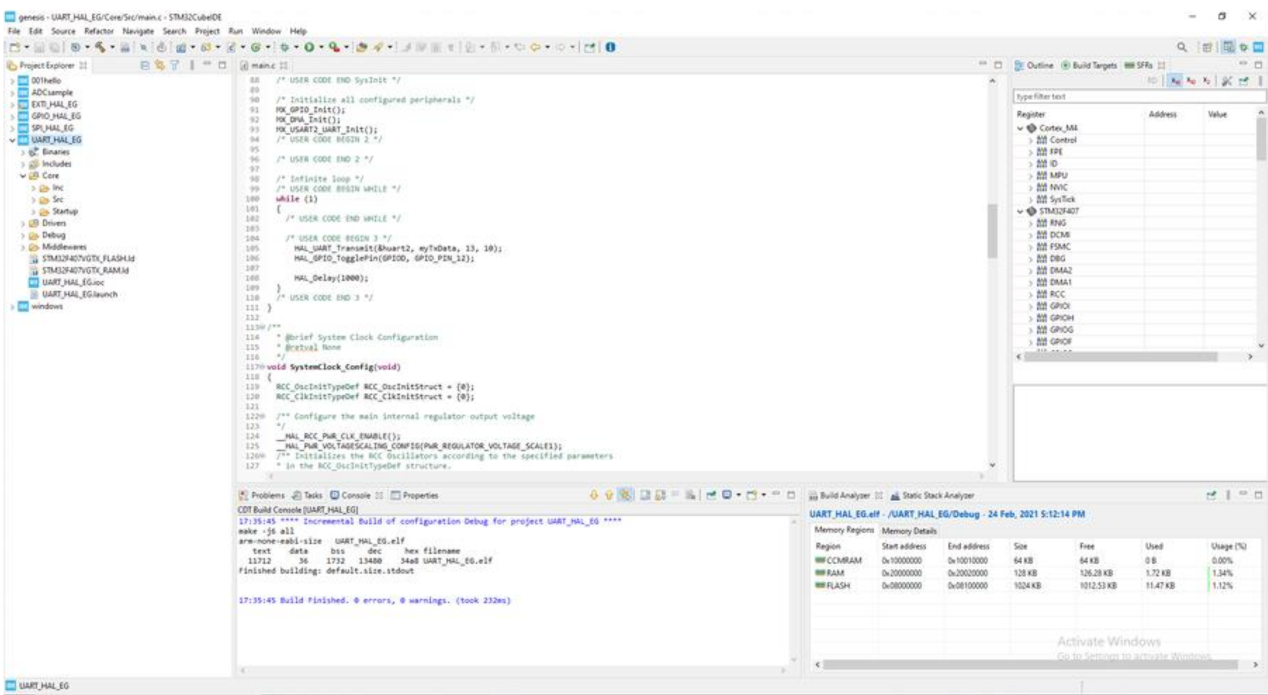


Here we are transmitting and receiving data using SPI

4. UART

In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART

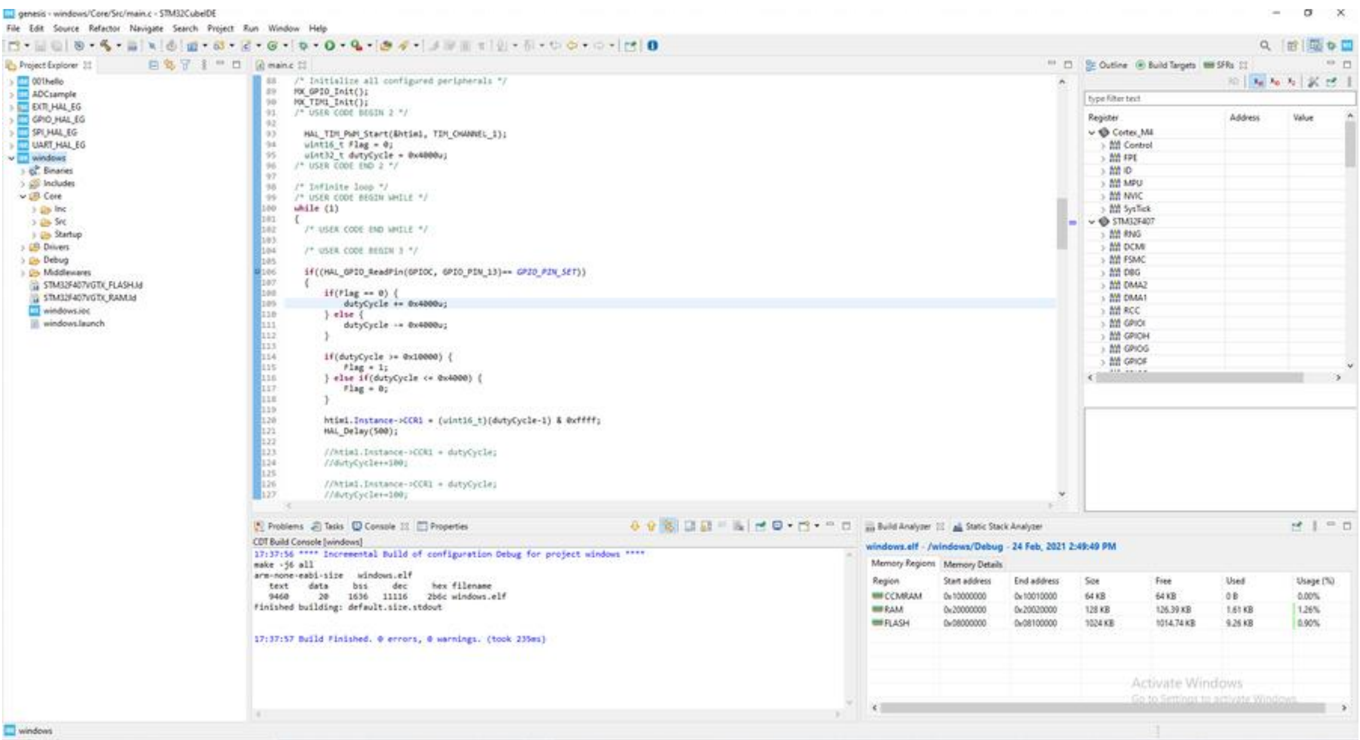




SPI interface of the STM32 devices using the STM32CubeMX HAL API.

5. PWM

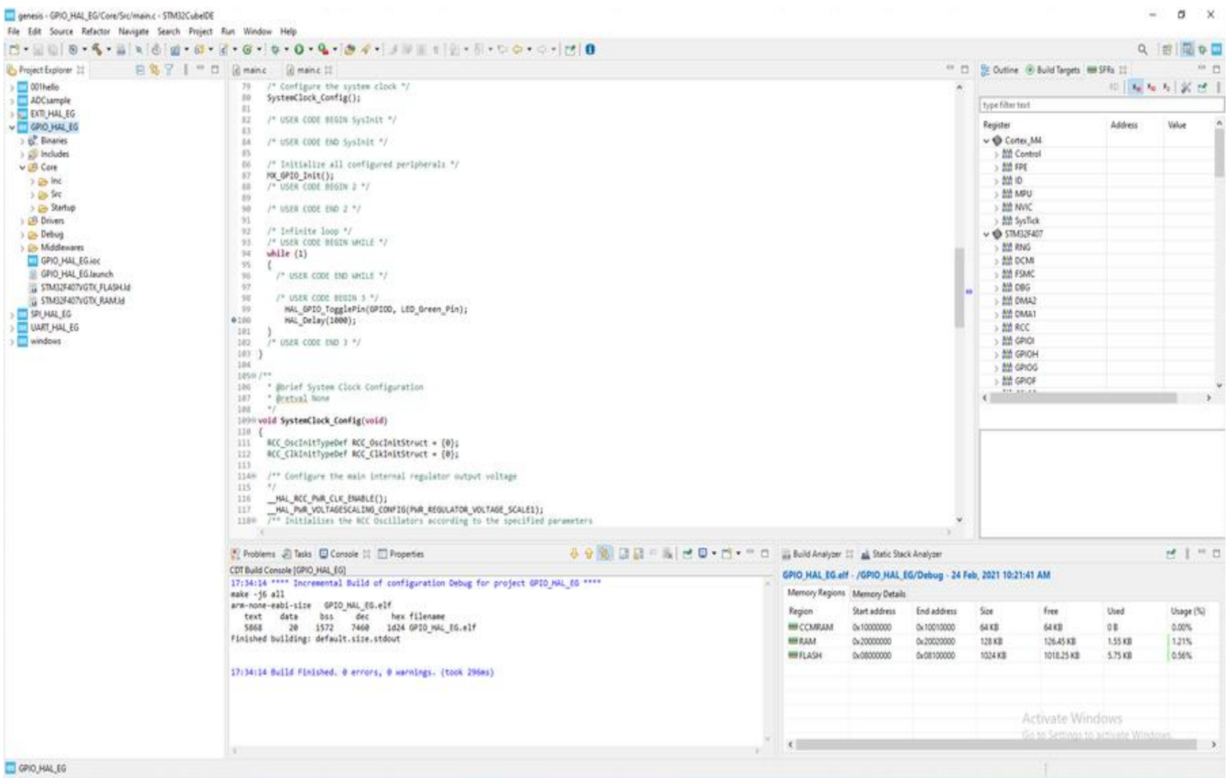
I used the STM32Cube initialization code generator to generate an initialized Timer function. To generate a fixed duty cycle PWM signal I added HAL_TIM_Base_Start ; //Starts the TIM Base generation and HAL_TIM_PWM_Start //Starts the PWM signal generation to the Timer initialization function as shown below.



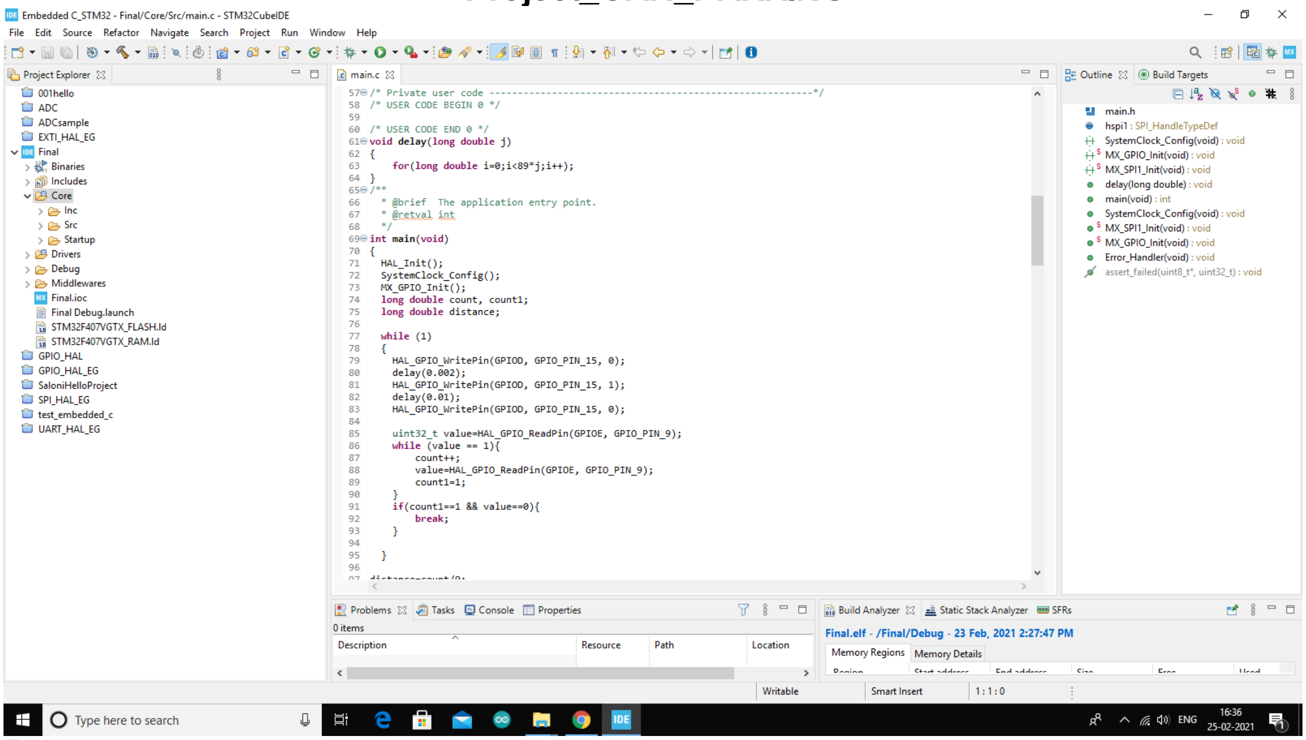
Reading GPIO and setting duty signal, we set flag and adjust PWM

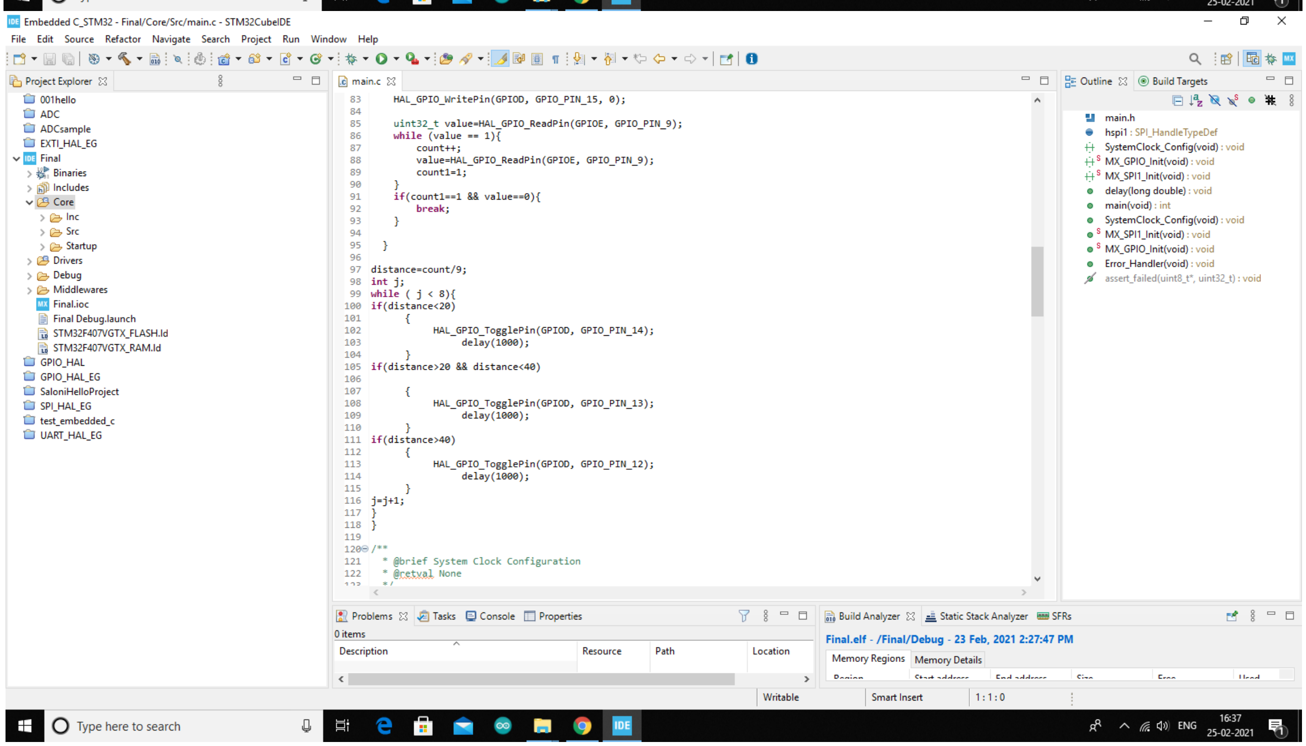
6. ADC

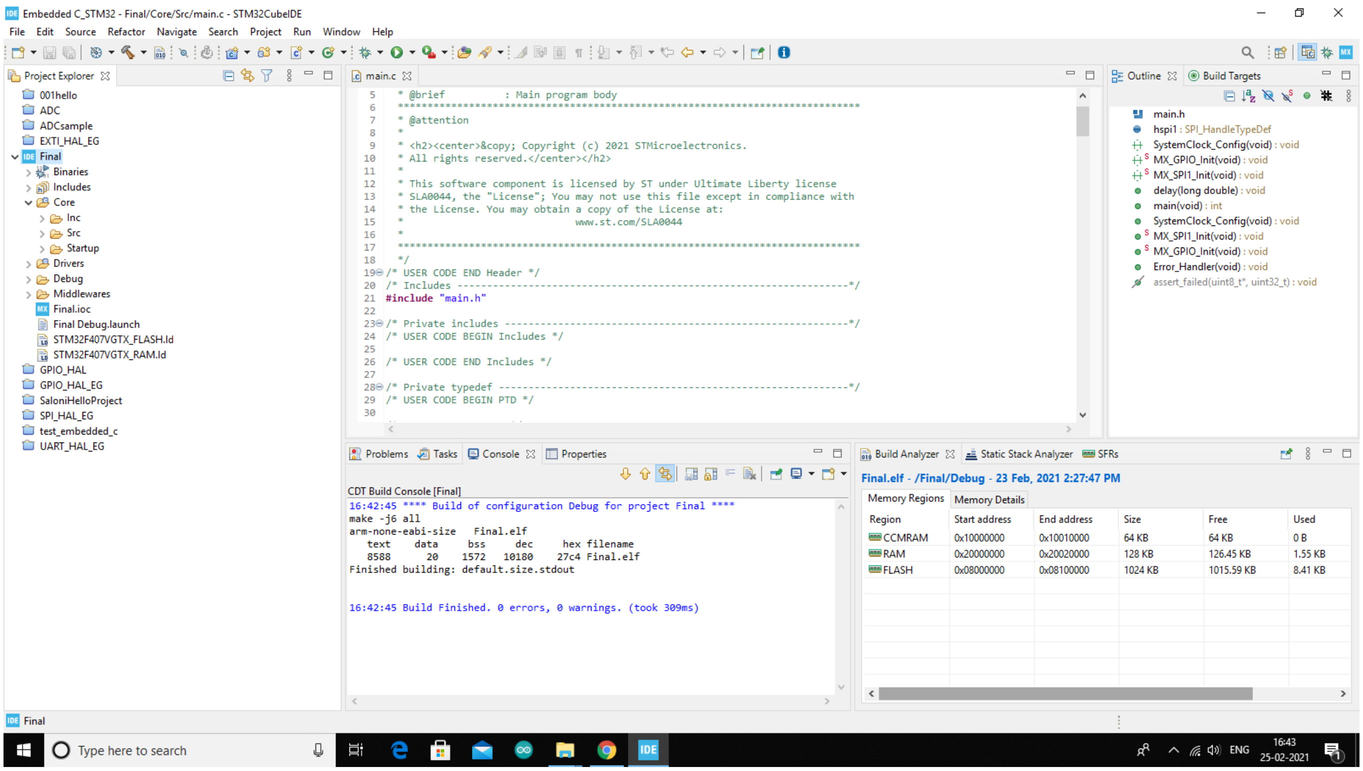
One of the most common peripherals on many modern microcontrollers is the analog-to-digital converter (ADC). These embedded devices read an analog voltage (usually somewhere between 0 V and the given reference voltage) and report it as a binary value. The exact implementation of the ADC can change among STM32 chips, as some use the successive-approximation register (SAR) technique while others rely on sigma-delta modulation for more resolution (but lower speeds).
ADC on the STM32L476 with STM32CubeIDE and HAL. Direct memory access (DMA) controller to demonstrate how you might handle moving (relatively) large amounts of data in your microcontroller.



Project_CAR_PARKING



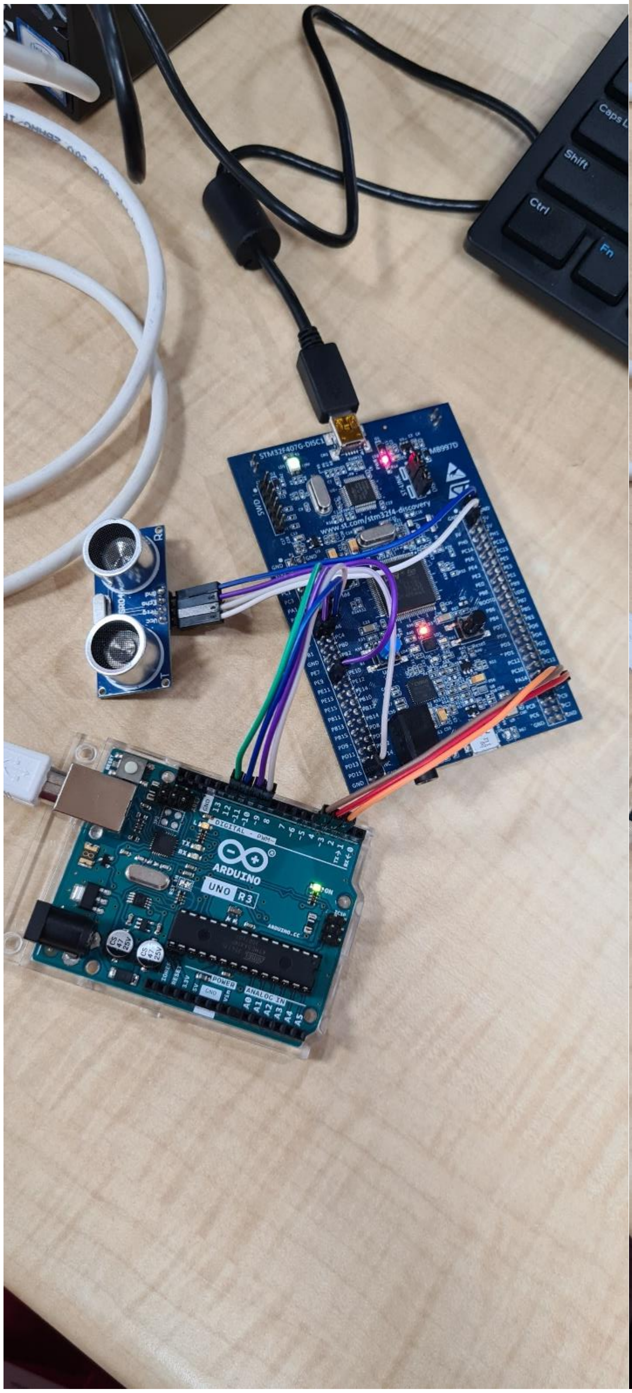


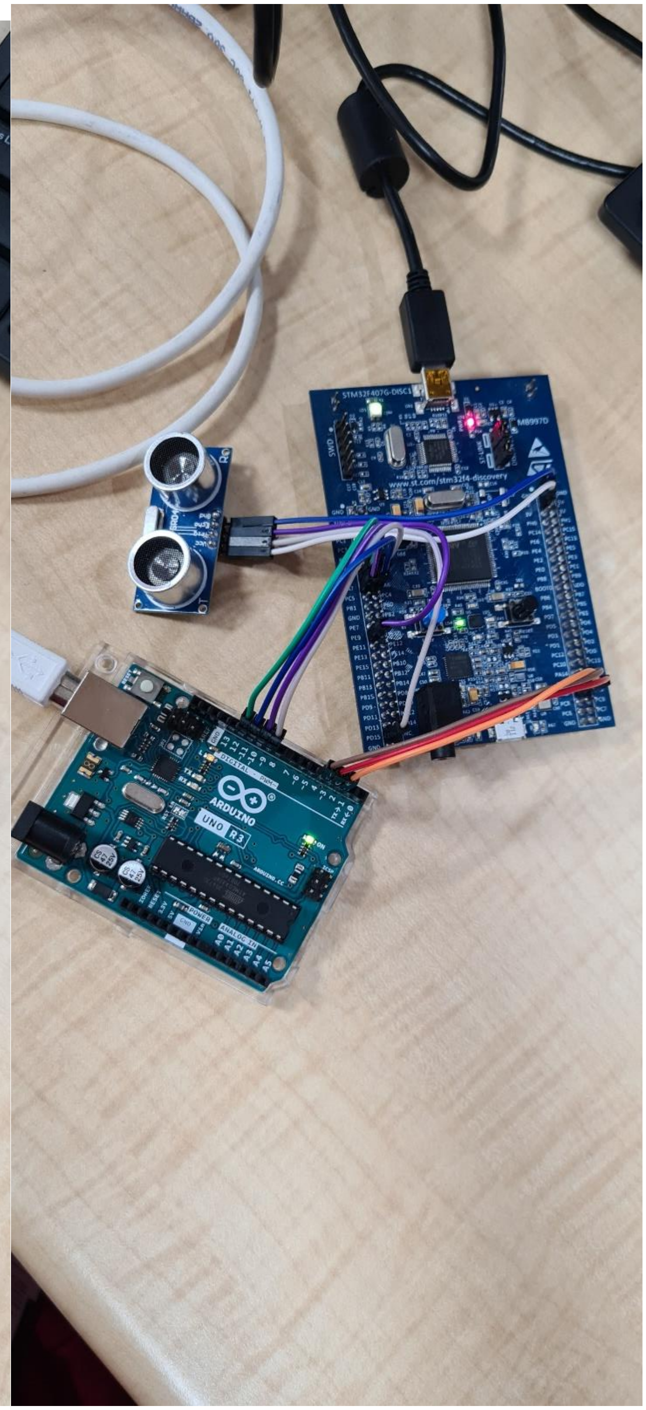


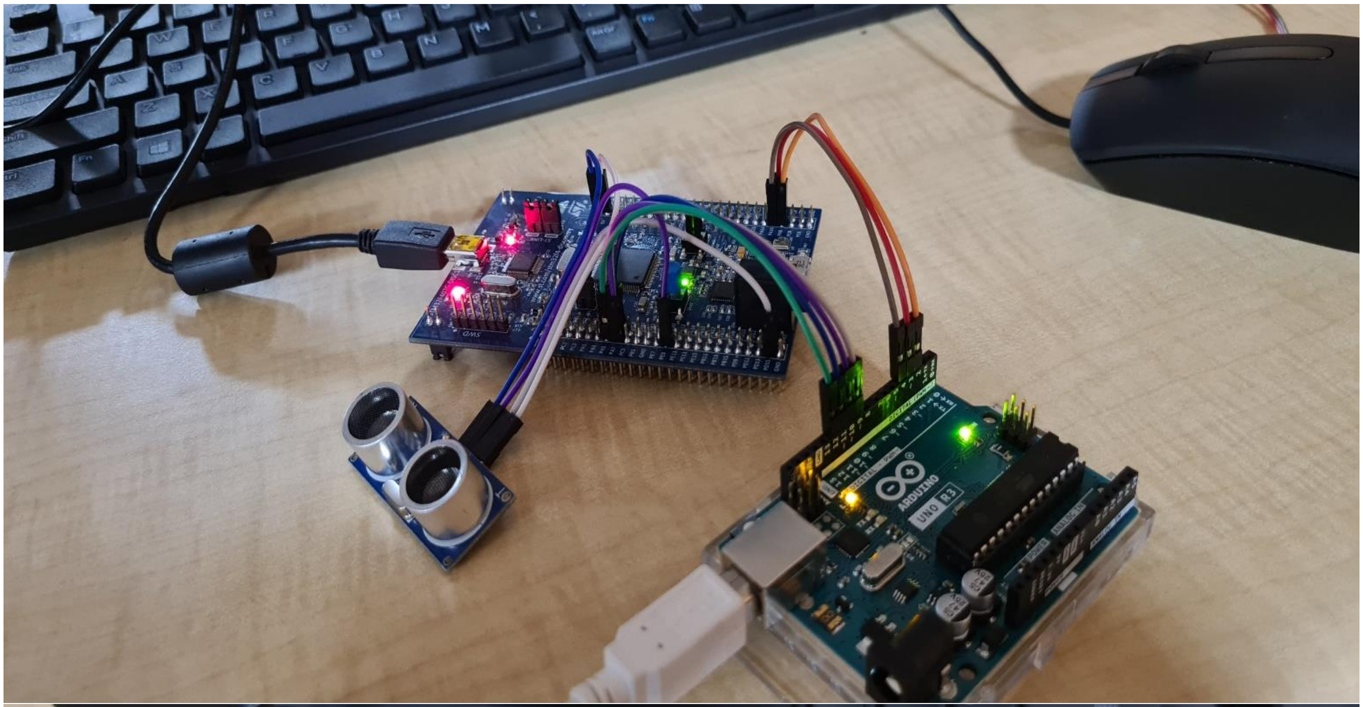
Here we are using ultrasonic sensor for measuring distance between object behind and Car. So that user can easily park car.

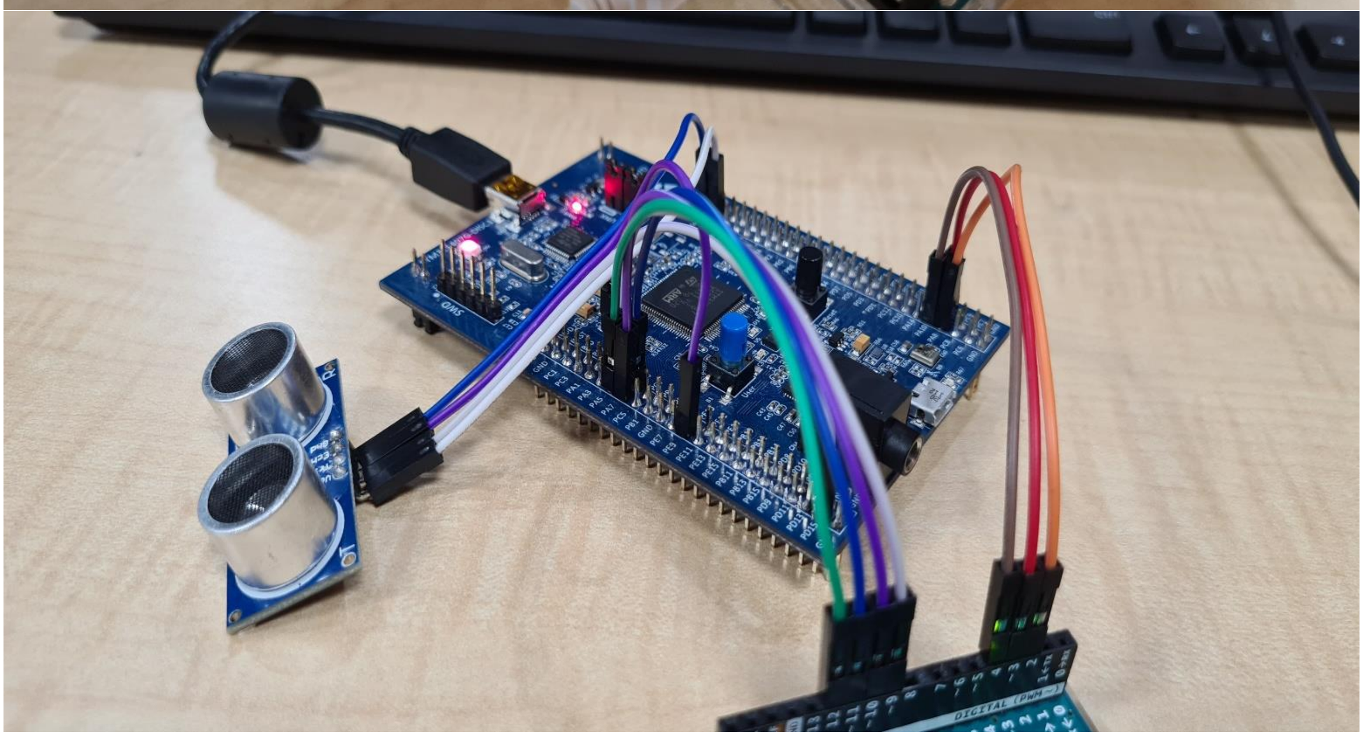
We are using three LEDs- Green , Orange , Red

Green – Nothing is there, Go fast and park
Orange – some distance is there between car and object , Be careful and Park
Red- Stop immediately, Object is so close


Shift
Ctrl
Fn
ARDUINO
UNO R3


ARDUINO
UNO R3


ARDUINO
UNO
R3


DIGITAL (PWM~)