./

GENESIS

**Learning Report on**

**Embedded C – Hardware + Programming + Testing**

(MBSE)



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **To be Approved** | **Remarks/Revision Details** |
| 1. | 27-2-2021 | Reeshav Rout |  |  |  |

**Details**

TABLE OF CONTENTS

[TABLE OF CONTENTS 3](#_Toc65331648)

[ACTIVITY 1 5](#_Toc65331649)

[ACTIVITY 2 6](#_Toc65331652)

[ACTIVITY 3 9](#_Toc65331654)

[ACTIVITY 4 10](#_Toc65331655)

[**ACTIVITY 5 : IMPLEMENTATION OF GPIO, EXTI, ADC, SPI AND UART USING STM IDE…………………**](#_Toc64406844)

[5.1 LED Toggle using Push Pull Configuration on STM32 Board. 11](#_Toc65331656)

[5.2 LED Toggle using Open Drain configuration and enabling Internal Pull Up on STM32 Board. 13](#_Toc65331657)

[5.3 LED Toggle using open drain configuration and external pull up on STM32 Board. 14](#_Toc65331658)

[5.4 Toggle External LED using push pull configuration on STM32 Board. 15](#_Toc65331659)

[5.4 Enable internal button and toggle the LED using STM32 Board. 16](#_Toc65331660)

[5.6 Use external switch to toggle the LED on STM32 Board. 18](#_Toc65331661)

[5.7 HAL INTERRUPT 20](#_Toc65331662)

[5.8 HAL ADC 23](#_Toc65331663)

[5.9 HAL SPI 25](#_Toc65331664)

[5.10 HAL UART 26](#_Toc65331666)

**LIST OF FIGURES**

[Figure 1 : MakeFile command **Error! Bookmark not defined.**](#_Toc65328803)

[Figure 2 : Makefile code **Error! Bookmark not defined.**](#_Toc65328804)

[Figure 3: StartUp file **Error! Bookmark not defined.**](#_Toc65328805)

[Figure 4: Linker Script **Error! Bookmark not defined.**](#_Toc65328806)

[Figure 5: LED toggle code 11](#_Toc65328807)

[Figure 6: LED toggle output 12](#_Toc65328808)

[Figure 7: open drain internal pull up code 13](#_Toc65328809)

[Figure 8:open drain internal pull up output 13](#_Toc65328810)

[Figure 9:open drain external pull up code 14](#_Toc65328811)

[Figure 10: open drain external pull up output 14](#_Toc65328812)

[Figure 11: External LED Push pull code 15](#_Toc65328813)

[Figure 12: External LED Push Pull output 15](#_Toc65328814)

[Figure 13: internal button led toggle code 16](#_Toc65328815)

[Figure 14:internal button led toggle code 17](#_Toc65328816)

[Figure 15:internal button led toggle output 17](#_Toc65328817)

[Figure 16:external switch led toggle 18](#_Toc65328818)

[Figure 17:external switch led toggle code 19](#_Toc65328819)

[Figure 18: external switch led toggle output 19](#_Toc65328820)

[Figure 19: HAL interrupt 20](#_Toc65328821)

[Figure 20: HAL interrupt 20](#_Toc65328822)

[Figure 21: HAL interrupt 21](#_Toc65328823)

[Figure 22: HAL interrupt 21](#_Toc65328824)

[Figure 23: HAL interrupt 22](#_Toc65328825)

[Figure 24: HAL interrupt 22](#_Toc65328826)

[Figure 25: HAL ADC 23](#_Toc65328827)

[Figure 26: HAL ADC 23](#_Toc65328828)

[Figure 27: HAL ADC 24](#_Toc65328829)

[Figure 28: HAL SPI 25](#_Toc65328830)

[Figure 29: Output of SPI 26](#_Toc65328831)

[Figure 30: Output of UART 26](#_Toc65328832)

# ACTIVITY 1

**CREATE A MAKEFILE**

### MAKEFILE

Make is UNIX utility that is designed to start execution of a make file. A makefile is a special file, containing shell commands that user creates and name makefile (or Makefile depending upon the system). While in the directory containing this makefile, user needs to type *make* and the commands in the makefile will be executed.

Make keeps track of the last time files (normally object files) were updated and only updates those files which are required (ones containing changes) to keep the source file up-to-date. If you have a large program with many source and/or header files, when you change a file on which others depend, you must recompile all the dependent files. Without a makefile, this is an extremely time-consuming task.

As a makefile is a list of shell commands, it must be written for the shell which will process the makefile. A makefile that works well in one shell may not execute properly in another shell.

The makefile contains a list of *rules*. These rules tell the system what commands you want to be executed. Most times, these rules are commands to compile (or recompile) a series of files. The rules, which must begin in column 1, are in two parts. The first line is called a *dependency* line and the subsequent line(s) are called *actions* or *commands*. The action line(s) must be indented with a tab.

### 1.2 CREATING A MAKEFILE:

Procedure done:

1. Dividing the program into multi files (.c and .h)
2. Object file designed by the command : -o
3. The command : arm-none-eabi-gcc main.c –o main.o -

It takes the input file main.c and creates the relocation object file (main.o).

1. arm-none-eabi-gcc –c –mcpu=cortex-m4-mthumb main.c –o main.o –

This removes the error in assembler stage.

The makefile command:

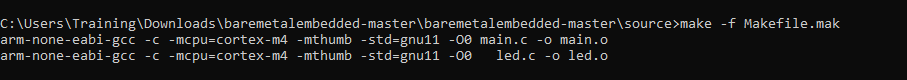


Figure 1: Makefile command

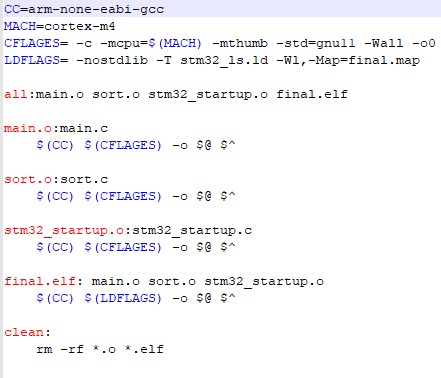


Figure 2: Code snippet of Makefile

# ACTIVITY 2

**CREATE A STARTUP FILE**

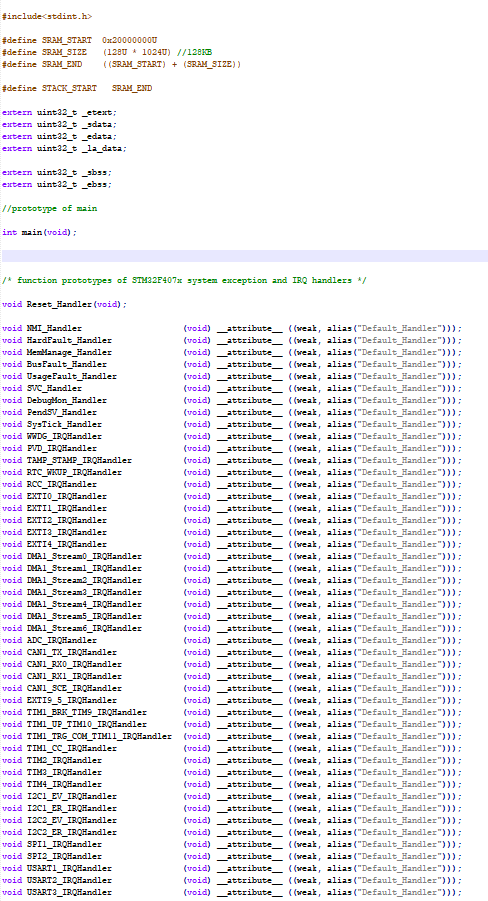
**2.1 START UP FILE:**

A start up file is a piece of code written in assembly or C language that executes before the main () function of our embedded application. It performs various initialization steps by setting up the hardware of the microcontroller so that the user application can run. Therefore, a start-up file always runs before the main () code of our embedded application.

### 2.2 CREATING A START UP FILE:

Following are the main functions of a start-up file:

1. Disable all interrupts.
2. Copying initialized global, global static, and local static variable data from flash to .data section RAM memory of a microcontroller
3. Copying uninitialized global, global static, and local static variable data from flash to .bss section of RAM memory and initialize .bss section of RAM to zero.
4. Allocate space for the stack and initialize the stack pointer
5. It also contains an array of function pointers (interrupt vector table) that point to various interrupt vector routines such as interrupts and exceptions. The start-up file also contains definitions of these interrupt or exception routines such as reset handler, NMI handler, bus fault handler, etc.
6. Enable interrupts
7. Calls the main function





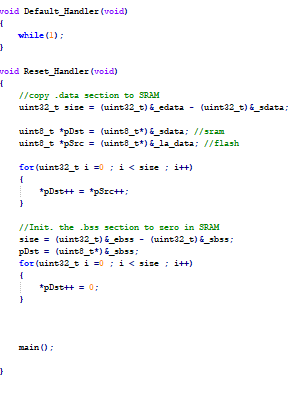


Figure 3: Code snippet of start up file

# ACTIVITY 3

**CREATE A LINKER SCRIPT**

**3.1 LINKER SCRIPT:**

The Linker Script is a text file made up of a series of Linker directives which tell the Linker where the available memory is and how it should be used. Thus, they reflect exactly the memory resources and memory map of the target microcontroller.

**3.2 CREATING A LINKER SCRIPT :**

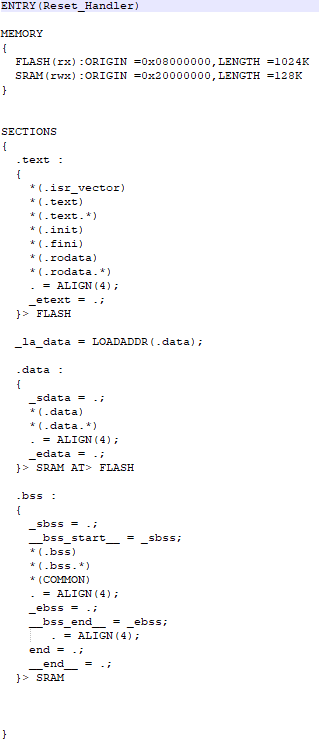
****

Figure 4: Code snippet of Linker script

# ACTIVITY 4

**DEBUGGING TECHNIQUES**

Debugging is the process of detecting and removing of existing and potential errors (also called as ‘bugs’) in a software code that can cause it to behave unexpectedly or crash. To prevent incorrect operation of a software or system, debugging is used to find and resolve bugs or defects. When various subsystems or modules are tightly coupled, debugging becomes harder as any change in one module may cause more bugs to appear in another.

Executable Download:

Connect the target to the PC by debug adapter, instituting programming. The host protocol to the native target protocol.

Programming Adapters:

It is used to get access to debug interface of the target with native protocol. It helps to download and debug code.

Eg: SEGGER - JLINK EDU

Open OCD:

Procedure:

1. Open OCD connects to debug adopter over its driver.

2. Open OCd sends USB packet as a target interface.

3. You can use JTAG pins or SWD. SWD has two pins. i.e., Clock and IO pins. IO is a data line> It is controlled by debug adopter and it bidirectional.

Various bus interface can be accessed such as ABP. There are various access points in the processors to talk to memory and core.

**ACTIVITY 5**

**TOGGLE LED**

# 5.1 LED Toggle using Push Pull Configuration on STM32 Board.

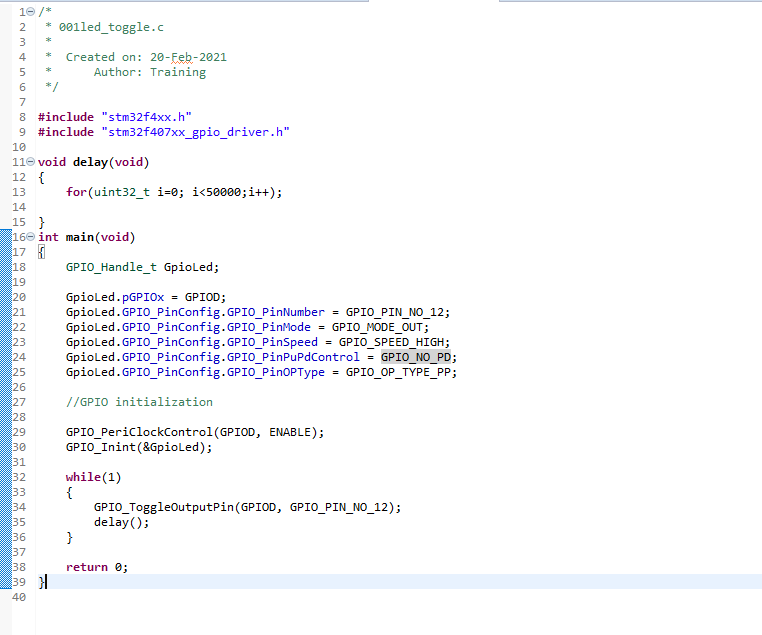


Figure 1: LED toggle code

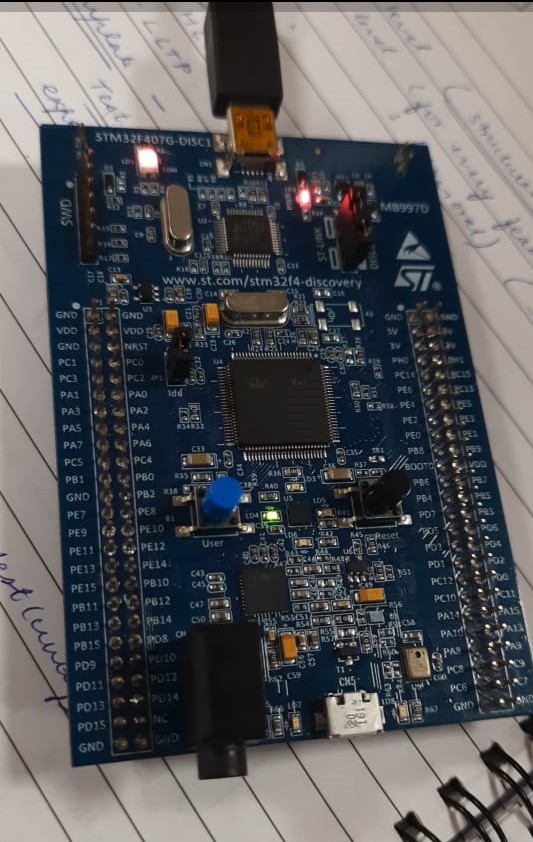


Figure 2: LED toggle output

# 5.2 LED Toggle using Open Drain configuration and enabling Internal Pull Up on STM32 Board.

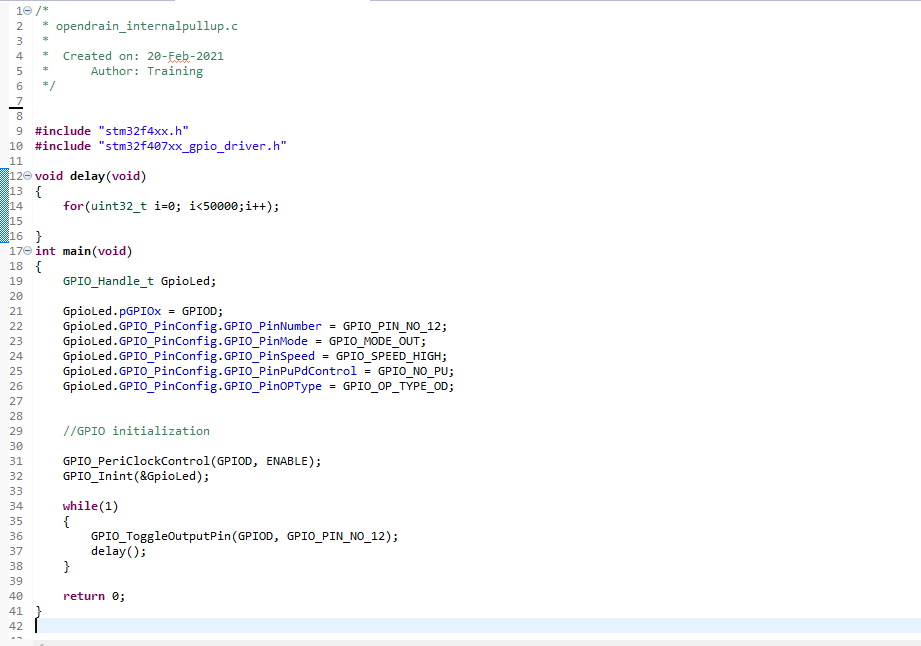


Figure 3: open drain internal pull up code

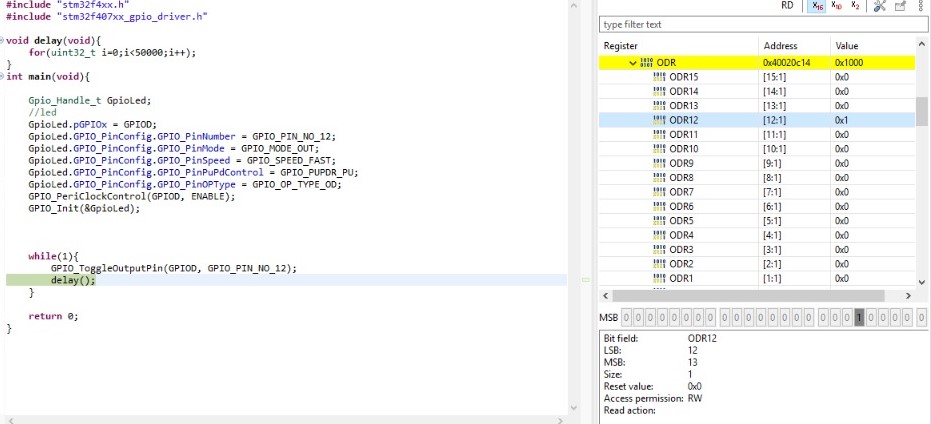
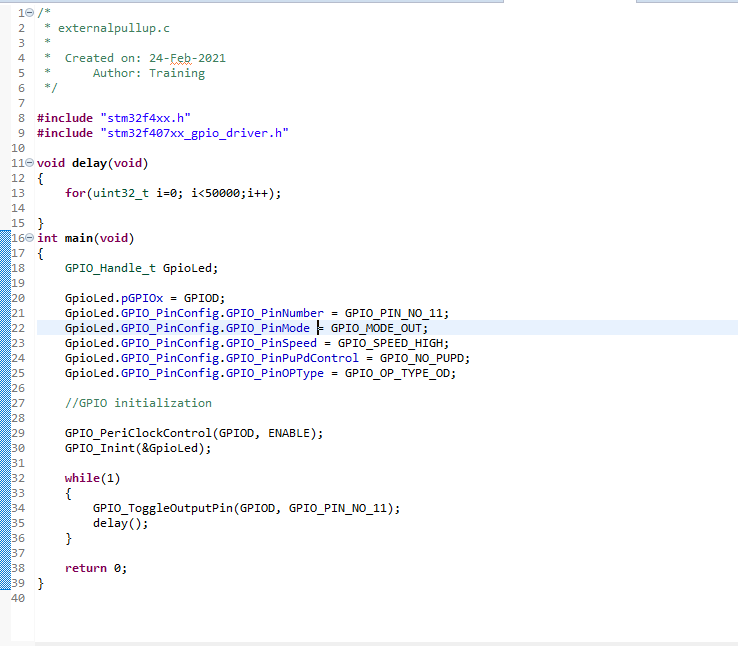


Figure 4: open drain internal pull up output

# 5.3 LED Toggle using open drain configuration and external pull up on STM32 Board.



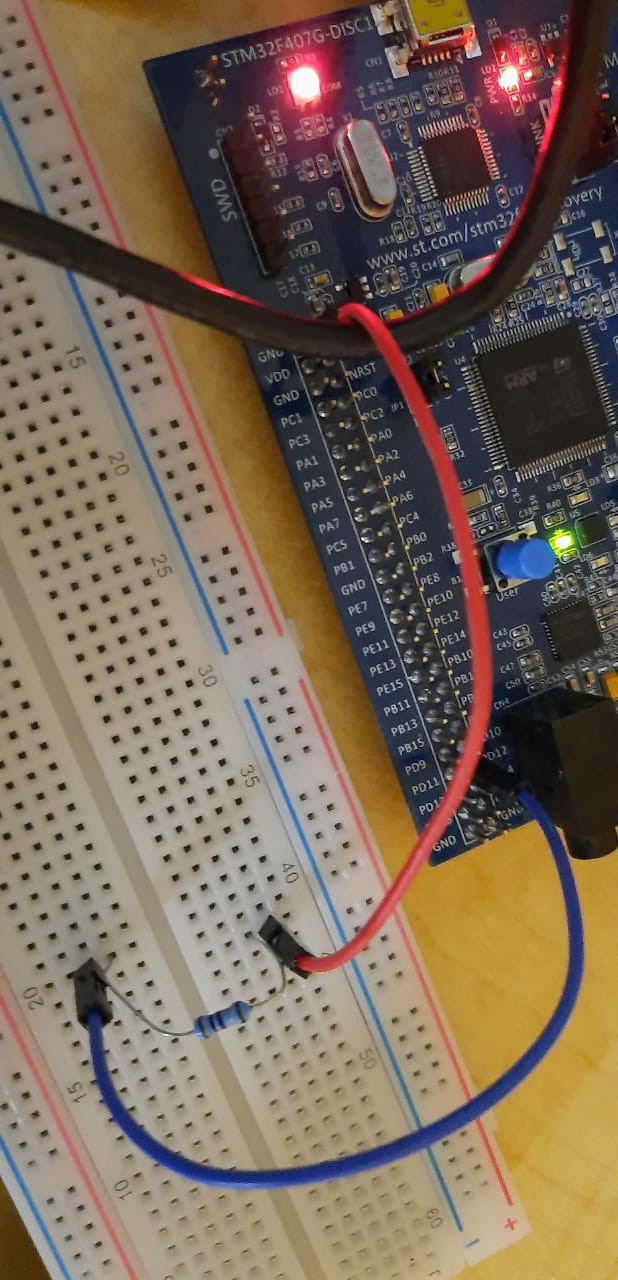
Figure 5: open drain external pull up code

Figure 6: open drain external pull up output

# 5.4 Toggle External LED using push pull configuration on STM32 Board.

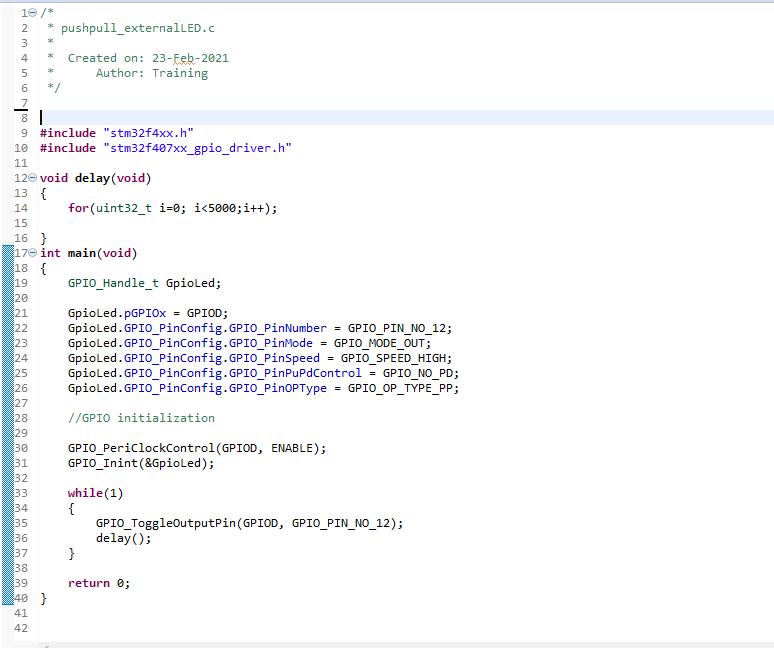


Figure 7: External LED Push pull code

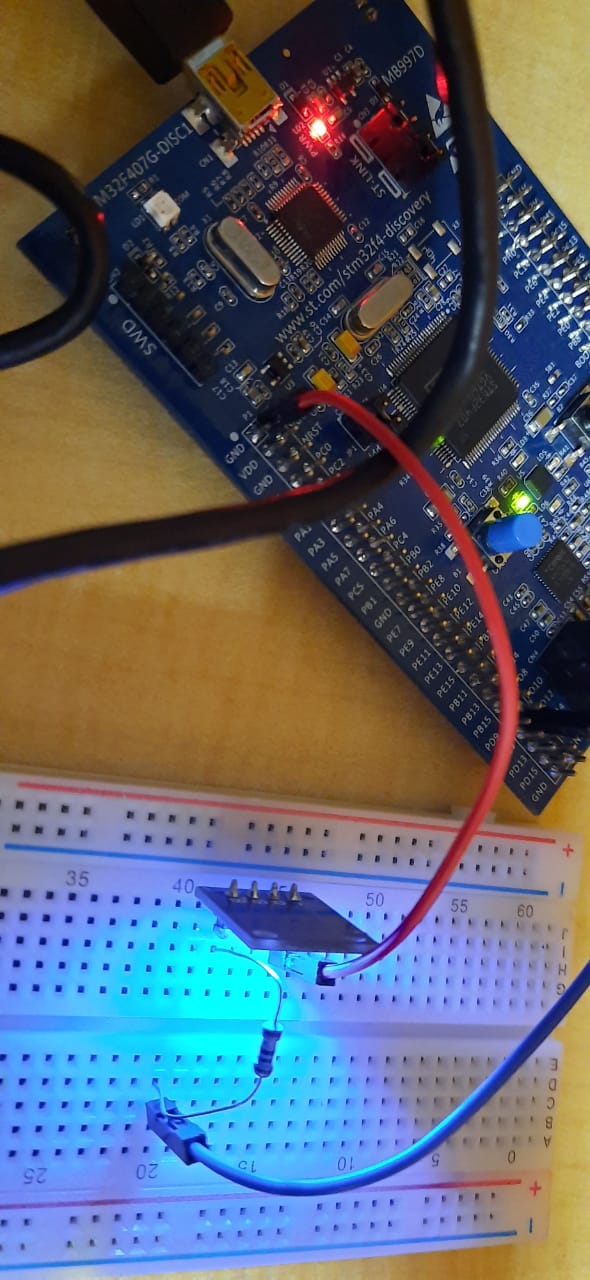


Figure 8: External LED Push Pull output

# 

# 5.4 Enable internal button and toggle the LED using STM32 Board.

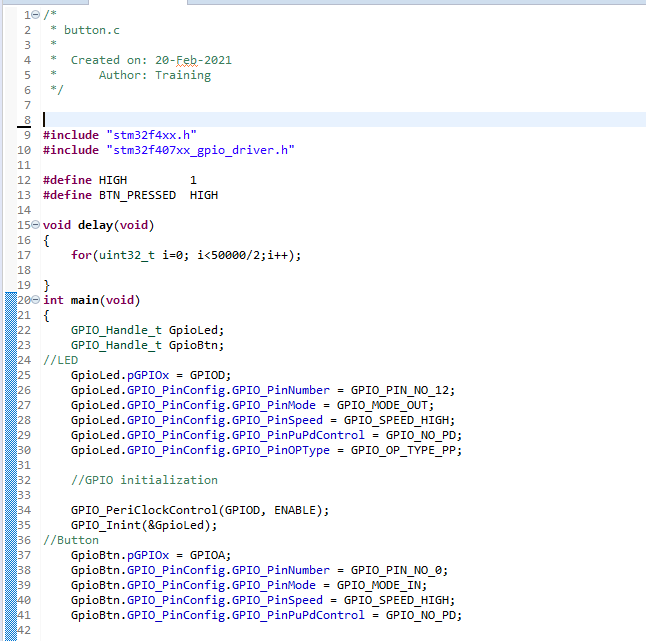


Figure 9: internal button led toggle code

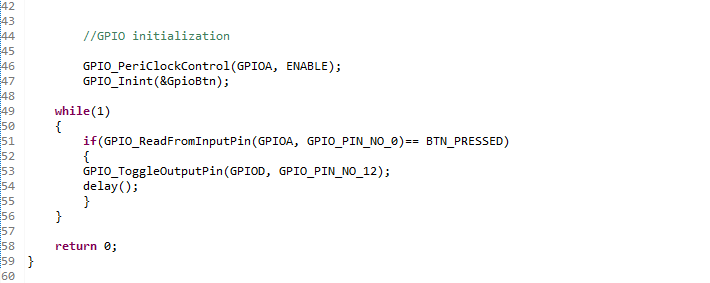


Figure 10: internal button led toggle code

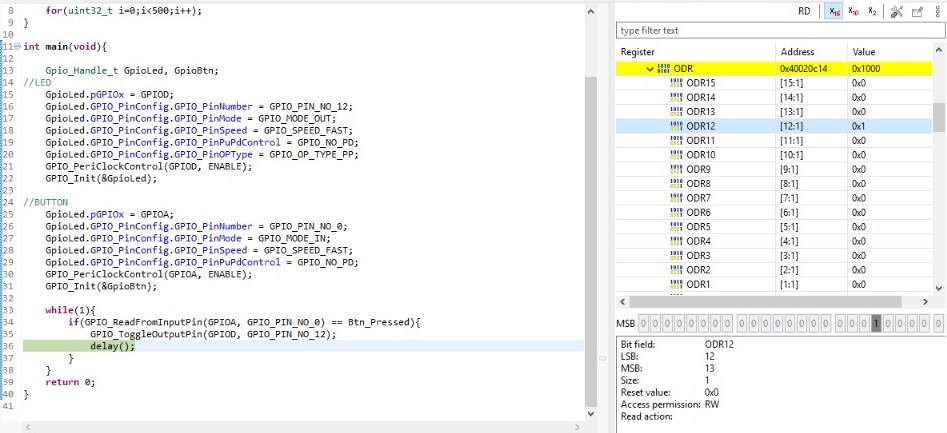


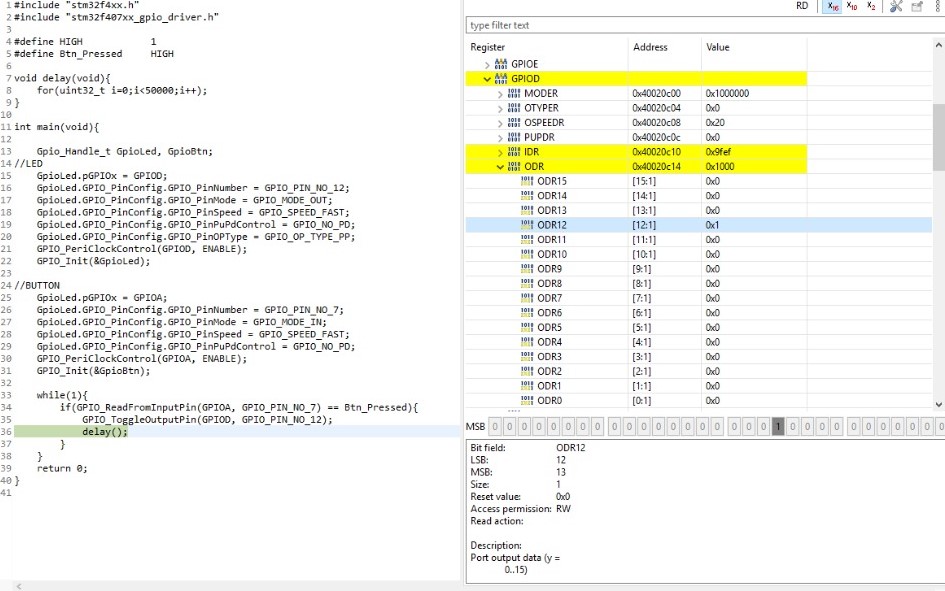
Figure 11: internal button led toggle output

# 

# 5.6 Use external switch to toggle the LED on STM32 Board.



Figure 12: external switch led toggle



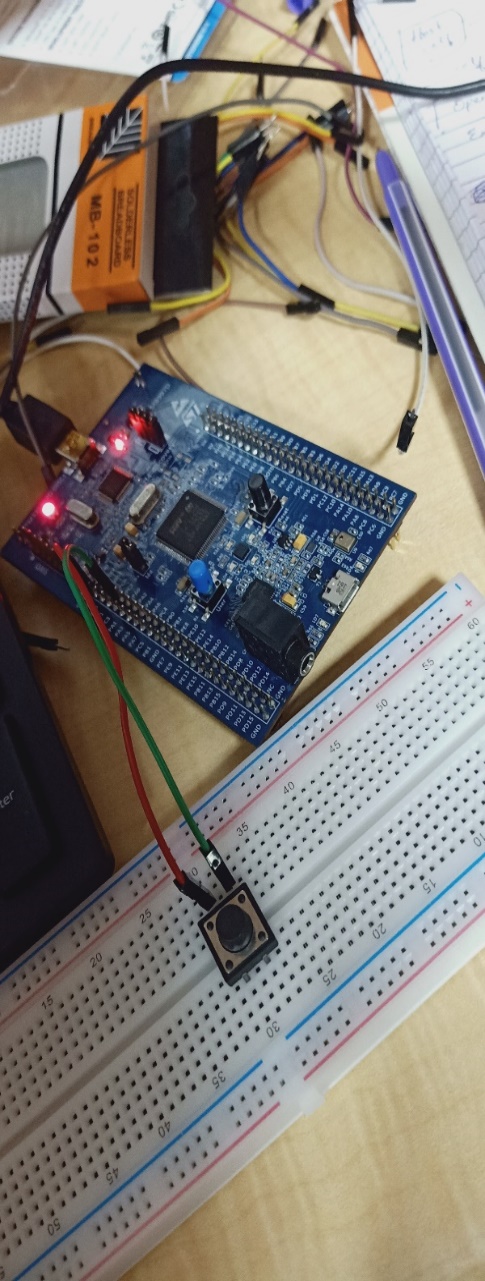
Figure 13: external switch led toggle code

Figure 14: external switch led toggle output

# 5.7 HAL INTERRUPT

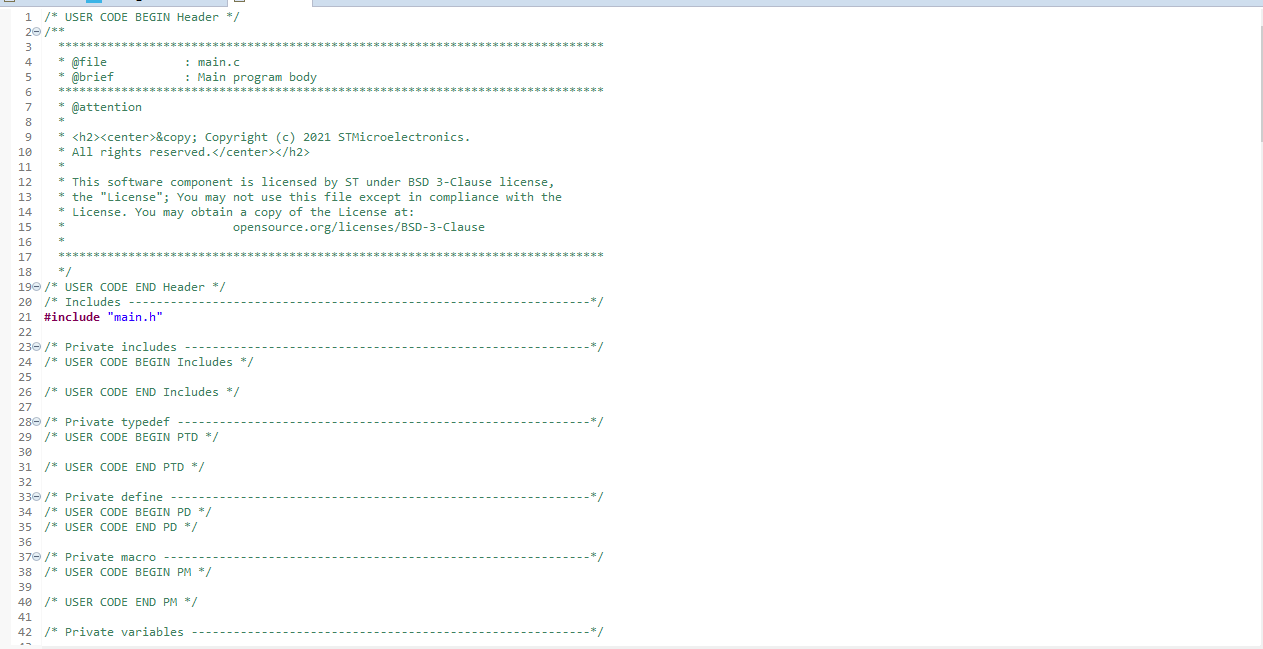


Figure 15: HAL interrupt

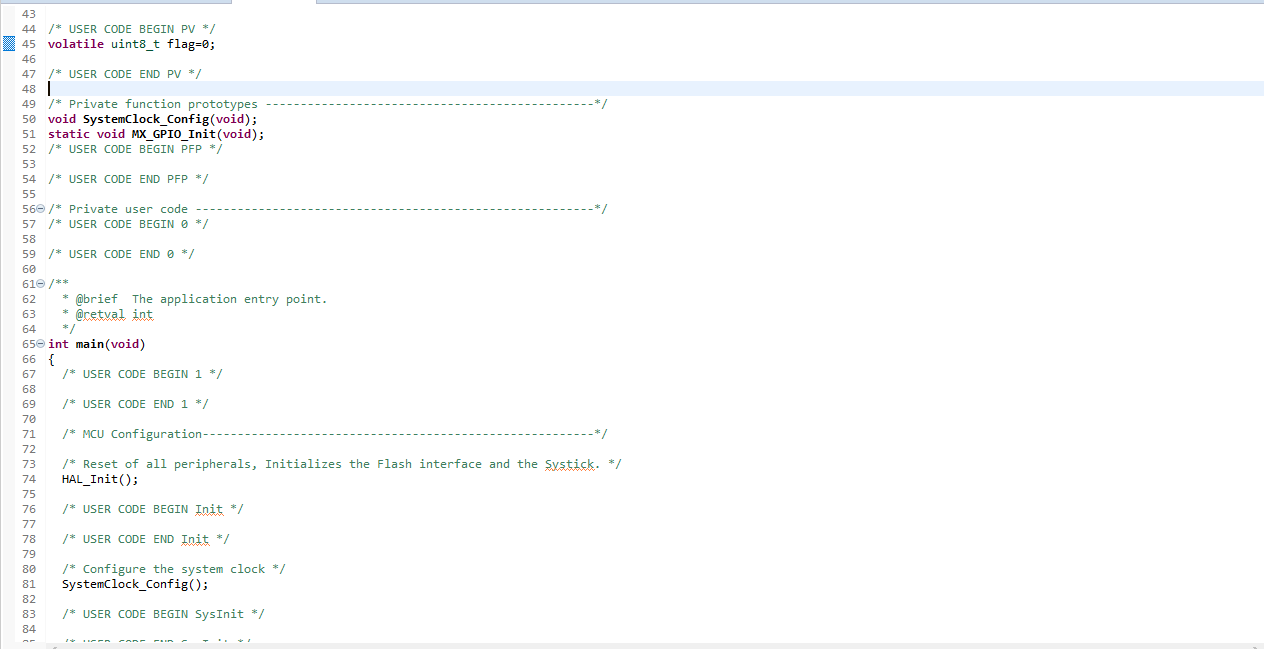


Figure 16: HAL interrupt

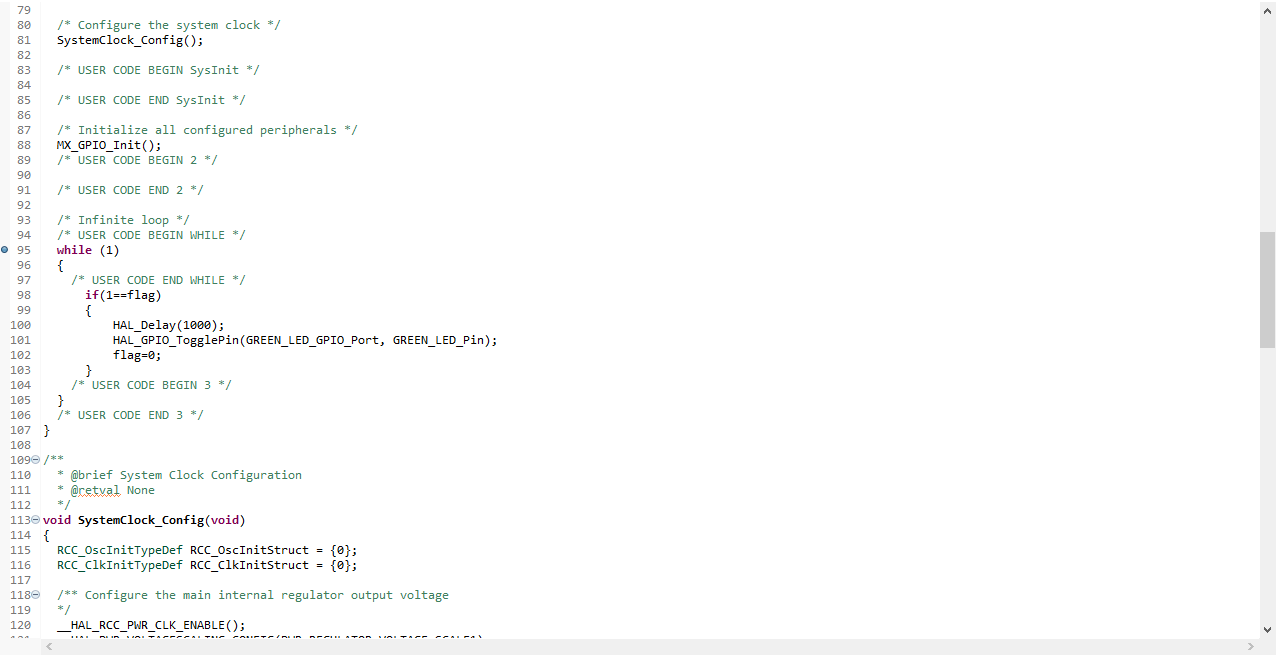


Figure 17: HAL interrupt

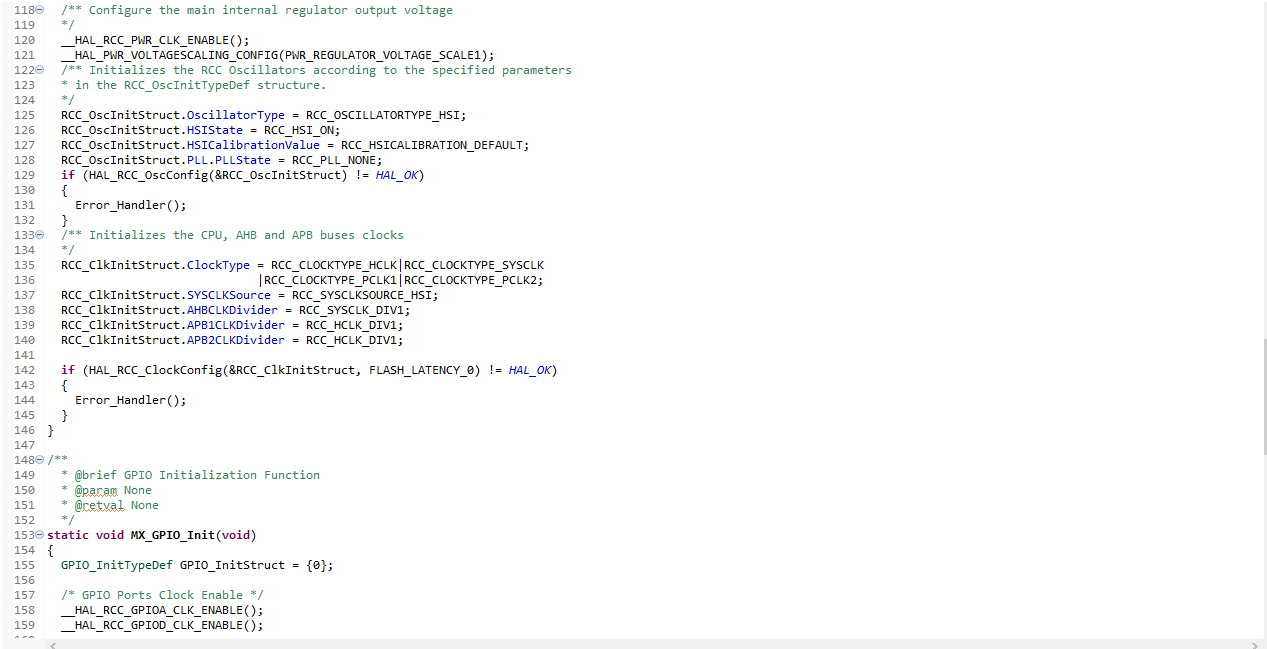
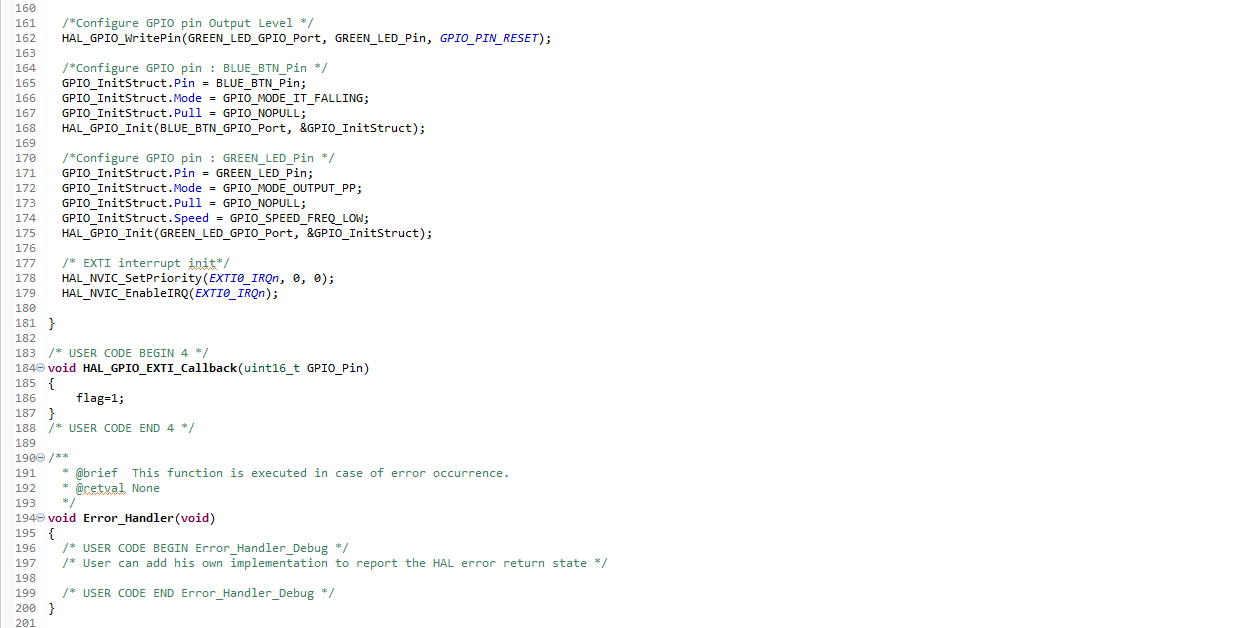


Figure 18: HAL interrupt



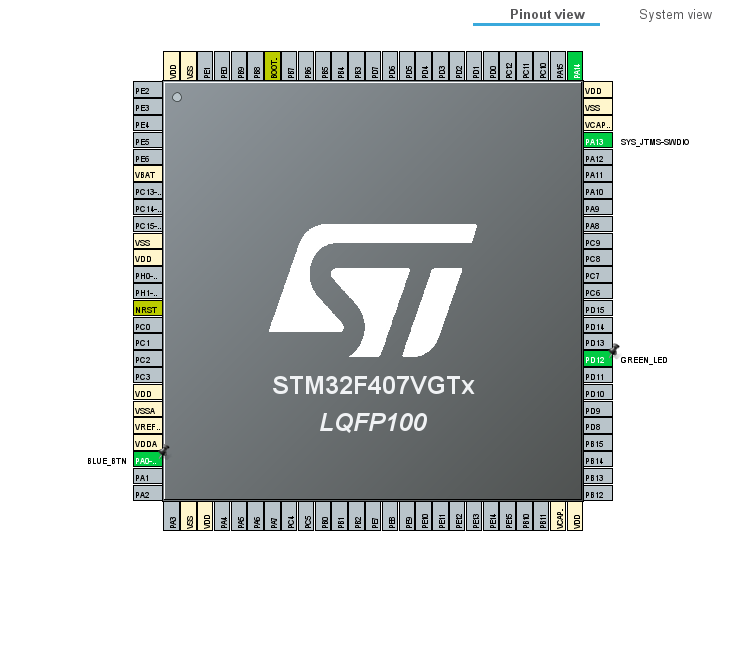
Figure 19: HAL interrupt

Figure 20: HAL interrupt

# 5.8 HAL ADC



Figure 21: HAL ADC

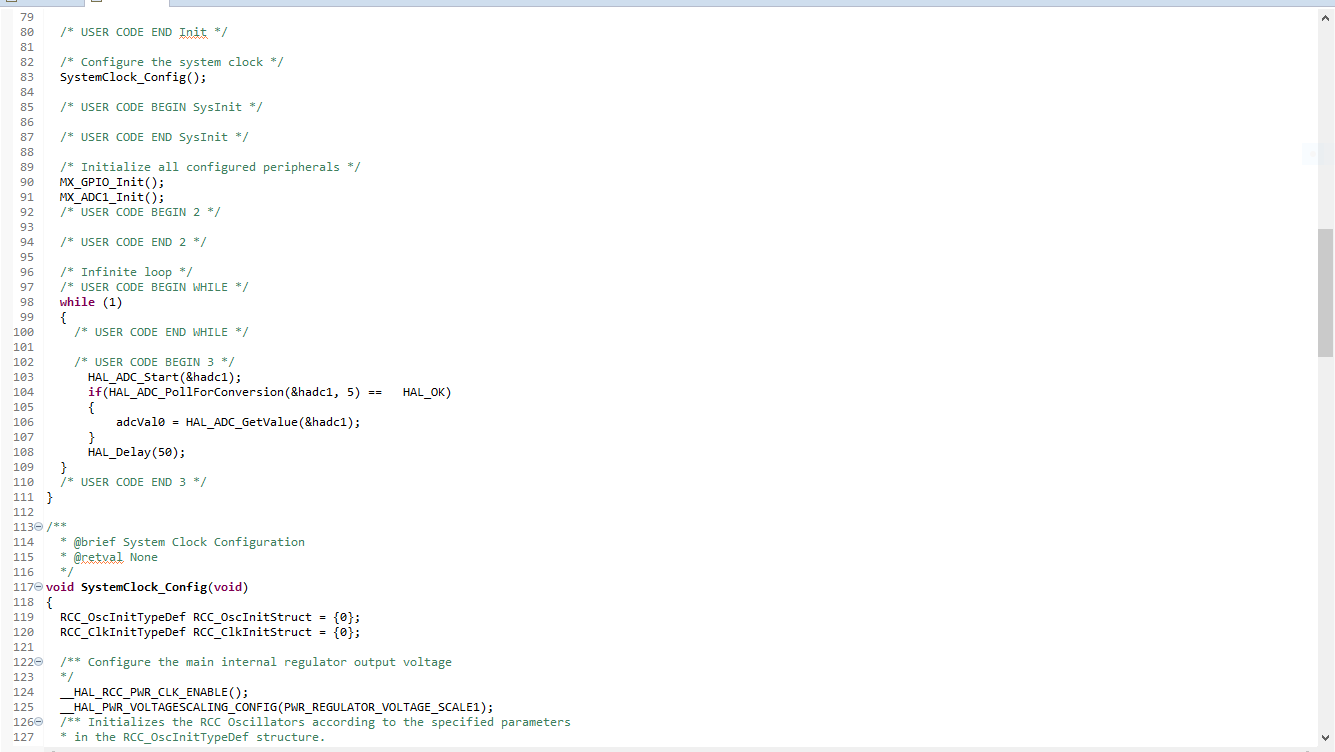


Figure 22: HAL ADC

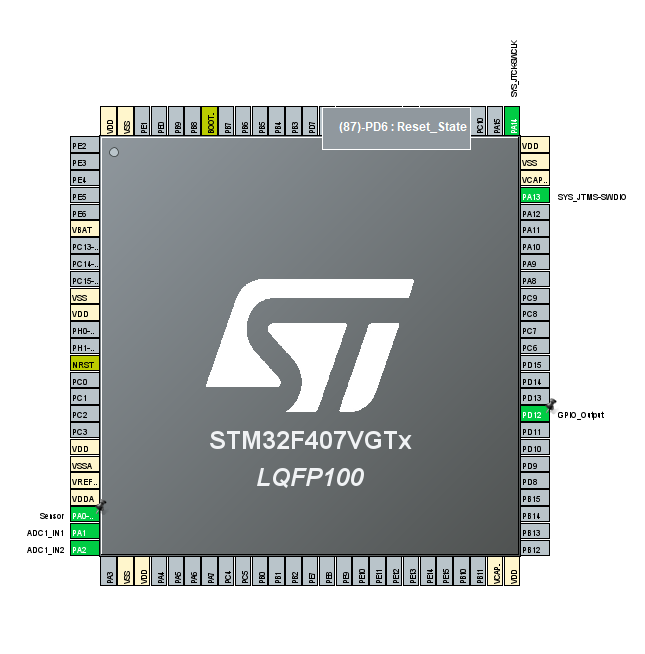


Figure 23: HAL ADC

# 

# 5.9 HAL SPI

# 

Figure 24: HAL SPI

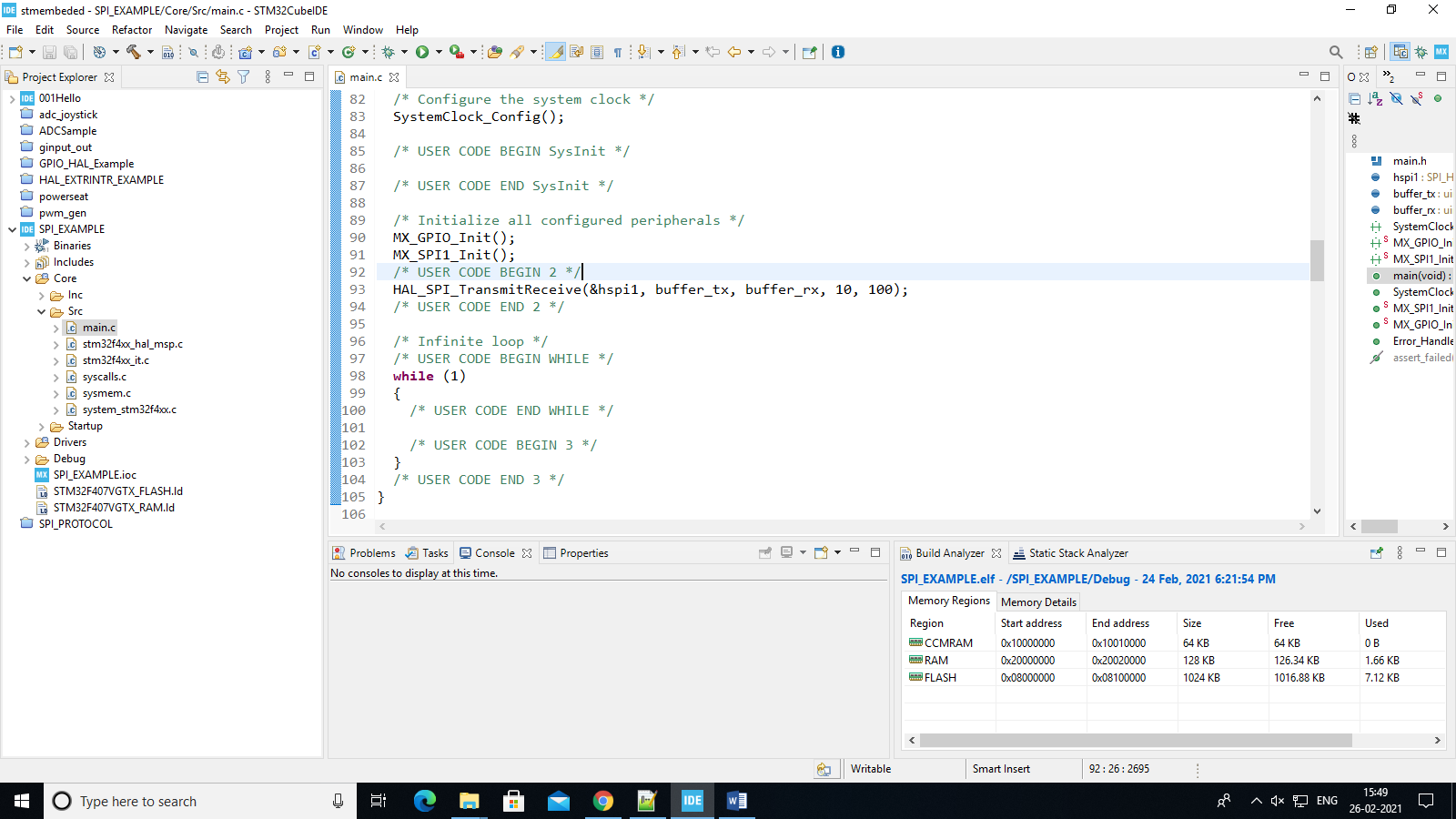


Figure 25: Output of SPI

# 5.10 HAL UART

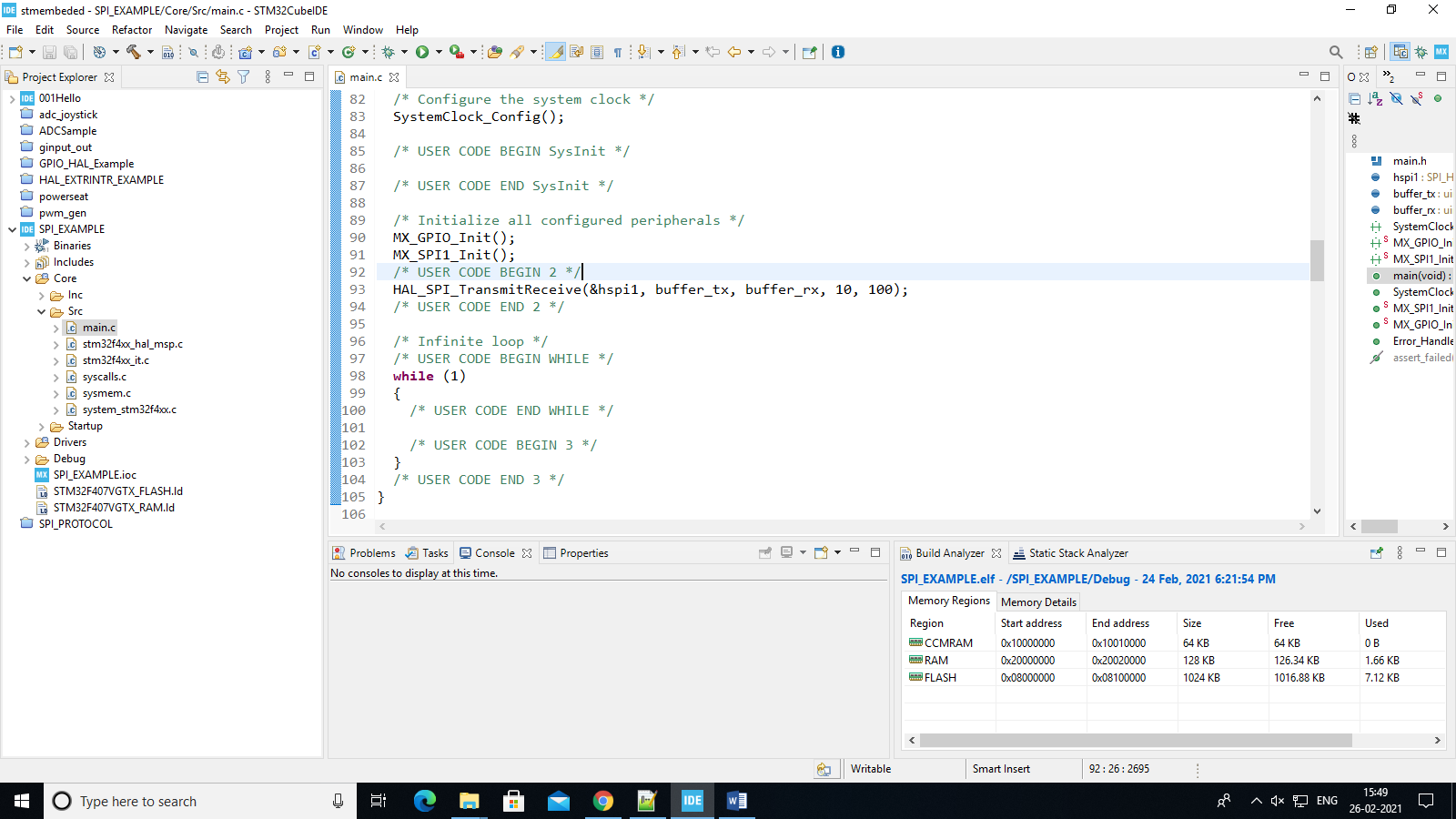


Figure 26: Output of UART