GENESIS

Learning Report on

Embedded C – Hardware + Programming + Testing

L&T Technology Services

LTTS
GLOBAL
ENGINEERING
ACADEMY

Details

Ver. Rel. No.	Release Date	Prepared. By	Reviewed By	To be Approved	Remarks/Revision Details
1.	27-2-2021	Sneha Anand			

TABLE OF CONTENTS

TABLE OF CONTENTS **3**

ACTIVITY 1 ..... **5**

1.1 MAKEFILE 5
1.2 CREATING A MAKEFILE: 5

ACTIVITY 2 ..... **6**

2.2 CREATING A START UP FILE: 6

ACTIVITY 3 ..... **9**

ACTIVITY 4 ..... **10**

ACTIVITY 5 : IMPLEMENTATION OF GPIO, EXTI, ADC, SPI AND UART USING STM IDE…….

5.1 LED TOGGLE USING PUSH PULL CONFIGURATION ON STM32 BOARD. **11**

5.2 LED TOGGLE USING OPEN DRAIN CONFIGURATION AND ENABLING INTERNAL PULL UP ON STM32 BOARD **13**

5.3 LED TOGGLE USING OPEN DRAIN CONFIGURATION AND EXTERNAL PULL UP ON STM32 BOARD. **14**

5.4 TOGGLE EXTERNAL LED USING PUSH PULL CONFIGURATION ON STM32 BOARD. **15**

5.4 ENABLE INTERNAL BUTTON AND TOGGLE THE LED USING STM32 BOARD. **16**

5.6 USE EXTERNAL SWITCH TO TOGGLE THE LED ON STM32 BOARD **18**

5.7 HAL INTERRUPT **20**

5.8 HAL ADC **23**

5.9 HAL SPI **25**

5.10 HAL UART **26**

LIST OF FIGURES

FIGURE 1 : MAKEFILE COMMAND ..... **ERROR! BOOKMARK NOT DEFINED.**
FIGURE 2 : MAKEFILE CODE ..... **ERROR! BOOKMARK NOT DEFINED.**
FIGURE 3: STARTUP FILE ..... **ERROR! BOOKMARK NOT DEFINED.**
FIGURE 4: LINKER SCRIPT ..... **ERROR! BOOKMARK NOT DEFINED.**
FIGURE 5: LED TOGGLE CODE 11
FIGURE 6: LED TOGGLE OUTPUT 12
FIGURE 7: OPEN DRAIN INTERNAL PULL UP CODE 13
FIGURE 8:OPEN DRAIN INTERNAL PULL UP OUTPUT 13
FIGURE 9:OPEN DRAIN EXTERNAL PULL UP CODE 14
FIGURE 10: OPEN DRAIN EXTERNAL PULL UP OUTPUT 14
FIGURE 11: EXTERNAL LED PUSH PULL CODE 15
FIGURE 12: EXTERNAL LED PUSH PULL OUTPUT 15
FIGURE 13: INTERNAL BUTTON LED TOGGLE CODE 16
FIGURE 14:INTERNAL BUTTON LED TOGGLE CODE 17
FIGURE 15:INTERNAL BUTTON LED TOGGLE OUTPUT 17
FIGURE 16:EXTERNAL SWITCH LED TOGGLE 18
FIGURE 17:EXTERNAL SWITCH LED TOGGLE CODE 19
FIGURE 18: EXTERNAL SWITCH LED TOGGLE OUTPUT 19
FIGURE 19: HAL INTERRUPT 20
FIGURE 20: HAL INTERRUPT 20
FIGURE 21: HAL INTERRUPT 21
FIGURE 22: HAL INTERRUPT 21
FIGURE 23: HAL INTERRUPT 22
FIGURE 24: HAL INTERRUPT 22
FIGURE 25: HAL ADC 23
FIGURE 26: HAL ADC 23
FIGURE 27: HAL ADC 24
FIGURE 28: HAL SPI 25
FIGURE 29: OUTPUT OF SPI 26
FIGURE 30: OUTPUT OF UART 26

ACTIVITY 1

CREATE A MAKEFILE

1.1 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).
4. arm-none-eabi-gcc –c –mcpu=cortex-m4-mthumb main.c –o main.o –
   This removes the error in assembler stage.

The makefile command:

```
C:\Users\Training\Downloads\baremetalembedded-master\baremetalembedded-master\source>make -f Makefile.mak
arm-none-eabi-gcc -c -mcpu=cortex-m4 -mthumb -std=gnu11 -O0 main.c -o main.o
arm-none-eabi-gcc -c -mcpu=cortex-m4 -mthumb -std=gnu11 -O0   led.c -o led.o
```

Figure 1: Makefile command

```
CC=arm-none-eabi-gcc
MACH=cortex-m4
CFLAGES= -c -mcpu=$(MACH) -mthumb -std=gnu11 -Wall -O0
LDFLAGS= -nostdlib -T stm32_ls.ld -Wl,-Map=final.map

all:main.o sort.o stm32_startup.o final.elf

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

sort.o:sort.c
    $(CC) $(CFLAGES) -o $@ $^

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

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

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

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

```
#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);


/* 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")));
void TAMP_STAMP_IRQHandler          (void) __attribute__ ((weak, alias("Default_Handler")));
void RTC_WKUP_IRQHandler            (void) __attribute__ ((weak, alias("Default_Handler")));
void RCC_IRQHandler                 (void) __attribute__ ((weak, alias("Default_Handler")));
void EXTI0_IRQHandler               (void) __attribute__ ((weak, alias("Default_Handler")));
void EXTI1_IRQHandler               (void) __attribute__ ((weak, alias("Default_Handler")));
void EXTI2_IRQHandler               (void) __attribute__ ((weak, alias("Default_Handler")));
void EXTI3_IRQHandler               (void) __attribute__ ((weak, alias("Default_Handler")));
void EXTI4_IRQHandler               (void) __attribute__ ((weak, alias("Default_Handler")));
void DMA1_Stream0_IRQHandler        (void) __attribute__ ((weak, alias("Default_Handler")));
void DMA1_Stream1_IRQHandler        (void) __attribute__ ((weak, alias("Default_Handler")));
void DMA1_Stream2_IRQHandler        (void) __attribute__ ((weak, alias("Default_Handler")));
void DMA1_Stream3_IRQHandler        (void) __attribute__ ((weak, alias("Default_Handler")));
void DMA1_Stream4_IRQHandler        (void) __attribute__ ((weak, alias("Default_Handler")));
void DMA1_Stream5_IRQHandler        (void) __attribute__ ((weak, alias("Default_Handler")));
void DMA1_Stream6_IRQHandler        (void) __attribute__ ((weak, alias("Default_Handler")));
void ADC_IRQHandler                 (void) __attribute__ ((weak, alias("Default_Handler")));
void CAN1_TX_IRQHandler             (void) __attribute__ ((weak, alias("Default_Handler")));
void CAN1_RX0_IRQHandler            (void) __attribute__ ((weak, alias("Default_Handler")));
void CAN1_RX1_IRQHandler            (void) __attribute__ ((weak, alias("Default_Handler")));
void CAN1_SCE_IRQHandler            (void) __attribute__ ((weak, alias("Default_Handler")));
void EXTI9_5_IRQHandler             (void) __attribute__ ((weak, alias("Default_Handler")));
void TIM1_BRK_TIM9_IRQHandler       (void) __attribute__ ((weak, alias("Default_Handler")));
void TIM1_UP_TIM10_IRQHandler       (void) __attribute__ ((weak, alias("Default_Handler")));
void TIM1_TRG_COM_TIM11_IRQHandler  (void) __attribute__ ((weak, alias("Default_Handler")));
void TIM1_CC_IRQHandler             (void) __attribute__ ((weak, alias("Default_Handler")));
void TIM2_IRQHandler                (void) __attribute__ ((weak, alias("Default_Handler")));
void TIM3_IRQHandler                (void) __attribute__ ((weak, alias("Default_Handler")));
void TIM4_IRQHandler                (void) __attribute__ ((weak, alias("Default_Handler")));
void I2C1_EV_IRQHandler             (void) __attribute__ ((weak, alias("Default_Handler")));
void I2C1_ER_IRQHandler             (void) __attribute__ ((weak, alias("Default_Handler")));
void I2C2_EV_IRQHandler             (void) __attribute__ ((weak, alias("Default_Handler")));
void I2C2_ER_IRQHandler             (void) __attribute__ ((weak, alias("Default_Handler")));
void SPI1_IRQHandler                (void) __attribute__ ((weak, alias("Default_Handler")));
void SPI2_IRQHandler                (void) __attribute__ ((weak, alias("Default_Handler")));
void USART1_IRQHandler              (void) __attribute__ ((weak, alias("Default_Handler")));
void USART2_IRQHandler              (void) __attribute__ ((weak, alias("Default_Handler")));
void USART3_IRQHandler              (void) __attribute__ ((weak, alias("Default_Handler")));
```

```
uint32_t vectors[] __attribute__((section(".isr_vector")))   = {
    STACK_START,
    (uint32_t)Reset_Handler,
    (uint32_t)NMI_Handler,
    (uint32_t)HardFault_Handler,
    (uint32_t)MemManage_Handler,
    (uint32_t)BusFault_Handler,
    (uint32_t)UsageFault_Handler,
    0,
    0,
    0,
    0,
    (uint32_t)SVC_Handler,
    (uint32_t)DebugMon_Handler,
    0,
    (uint32_t)PendSV_Handler,
    (uint32_t)SysTick_Handler,
    (uint32_t)WWDG_IRQHandler,
    (uint32_t)PVD_IRQHandler,
    (uint32_t)TAMP_STAMP_IRQHandler,
    (uint32_t)RTC_WKUP_IRQHandler,
    0,
    (uint32_t)RCC_IRQHandler,
    (uint32_t)EXTI0_IRQHandler,
    (uint32_t)EXTI1_IRQHandler,
    (uint32_t)EXTI2_IRQHandler,
    (uint32_t)EXTI3_IRQHandler,
    (uint32_t)EXTI4_IRQHandler,
    (uint32_t)DMA1_Stream0_IRQHandler,
    (uint32_t)DMA1_Stream1_IRQHandler,
    (uint32_t)DMA1_Stream2_IRQHandler,
    (uint32_t)DMA1_Stream3_IRQHandler,
    (uint32_t)DMA1_Stream4_IRQHandler,
    (uint32_t)DMA1_Stream5_IRQHandler,
    (uint32_t)DMA1_Stream6_IRQHandler,
    (uint32_t)ADC_IRQHandler,
    (uint32_t)CAN1_TX_IRQHandler,
    (uint32_t)CAN1_RX0_IRQHandler,
    (uint32_t)CAN1_RX1_IRQHandler,
    (uint32_t)CAN1_SCE_IRQHandler,
    (uint32_t)EXTI9_5_IRQHandler,
    (uint32_t)TIM1_BRK_TIM9_IRQHandler,
    (uint32_t)TIM1_UP_TIM10_IRQHandler,
    (uint32_t)TIM1_TRG_COM_TIM11_IRQHandler,
    (uint32_t)TIM1_CC_IRQHandler,
    (uint32_t)TIM2_IRQHandler,
    (uint32_t)TIM3_IRQHandler,
    (uint32_t)TIM4_IRQHandler,
    (uint32_t)I2C1_EV_IRQHandler,
    (uint32_t)I2C1_ER_IRQHandler,
    (uint32_t)I2C2_EV_IRQHandler,
    (uint32_t)I2C2_ER_IRQHandler,
    (uint32_t)SPI1_IRQHandler,
    (uint32_t)SPI2_IRQHandler,
    (uint32_t)USART1_IRQHandler,
    (uint32_t)USART2_IRQHandler,
    (uint32_t)USART3_IRQHandler,
    (uint32_t)EXTI15_10_IRQHandler,
    (uint32_t)RTC_Alarm_IRQHandler,
    (uint32_t)OTG_FS_WKUP_IRQHandler,
    (uint32_t)TIM8_BRK_TIM12_IRQHandler,
    (uint32_t)TIM8_UP_TIM13_IRQHandler,
    (uint32_t)TIM8_TRG_COM_TIM14_IRQHandler,
```

```
void Default_Handler(void)
{
    while(1);
}

void Reset_Handler(void)
{
    //copy .data section to SRAM
    uint32_t size = (uint32_t)&_edata - (uint32_t)&_sdata;

    uint8_t *pDst = (uint8_t*)&_sdata; //sram
    uint8_t *pSrc = (uint8_t*)&_la_data; //flash

    for(uint32_t i =0 ; i < size ; i++)
    {
        *pDst++ = *pSrc++;
    }

    //Init. the .bss section to zero in SRAM
    size = (uint32_t)&_ebss - (uint32_t)&_sbss;
    pDst = (uint8_t*)&_sbss;
    for(uint32_t i =0 ; i < size ; i++)
    {
        *pDst++ = 0;
    }


    main();

}
```

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 :

```
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 = .;
      __bss_start__ = _sbss;
      *(.bss)
      *(.bss.*)
      *(COMMON)
      . = ALIGN(4);
      _ebss = .;
      __bss_end__ = _ebss;
      . = ALIGN(4);
      end = .;
      __end__ = .;
   }> SRAM
```

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.

```c
/*
 * 001led_toggle.c
 *
 *  Created on: 20-Feb-2021
 *      Author: Training
 */

#include "stm32f4xx.h"
#include "stm32f407xx_gpio_driver.h"

void delay(void)
{
    for(uint32_t i=0; i<50000;i++);

}
int main(void)
{
    GPIO_Handle_t GpioLed;

    GpioLed.pGPIOx = GPIOD;
    GpioLed.GPIO_PinConfig.GPIO_PinNumber = GPIO_PIN_NO_12;
    GpioLed.GPIO_PinConfig.GPIO_PinMode = GPIO_MODE_OUT;
    GpioLed.GPIO_PinConfig.GPIO_PinSpeed = GPIO_SPEED_HIGH;
    GpioLed.GPIO_PinConfig.GPIO_PinPuPdControl = GPIO_NO_PD;
    GpioLed.GPIO_PinConfig.GPIO_PinOPType = GPIO_OP_TYPE_PP;

    //GPIO initialization

    GPIO_PeriClockControl(GPIOD, ENABLE);
    GPIO_Inint(&GpioLed);

    while(1)
    {
        GPIO_ToggleOutputPin(GPIOD, GPIO_PIN_NO_12);
        delay();
    }

    return 0;
}

```

Figure 1: LED toggle code

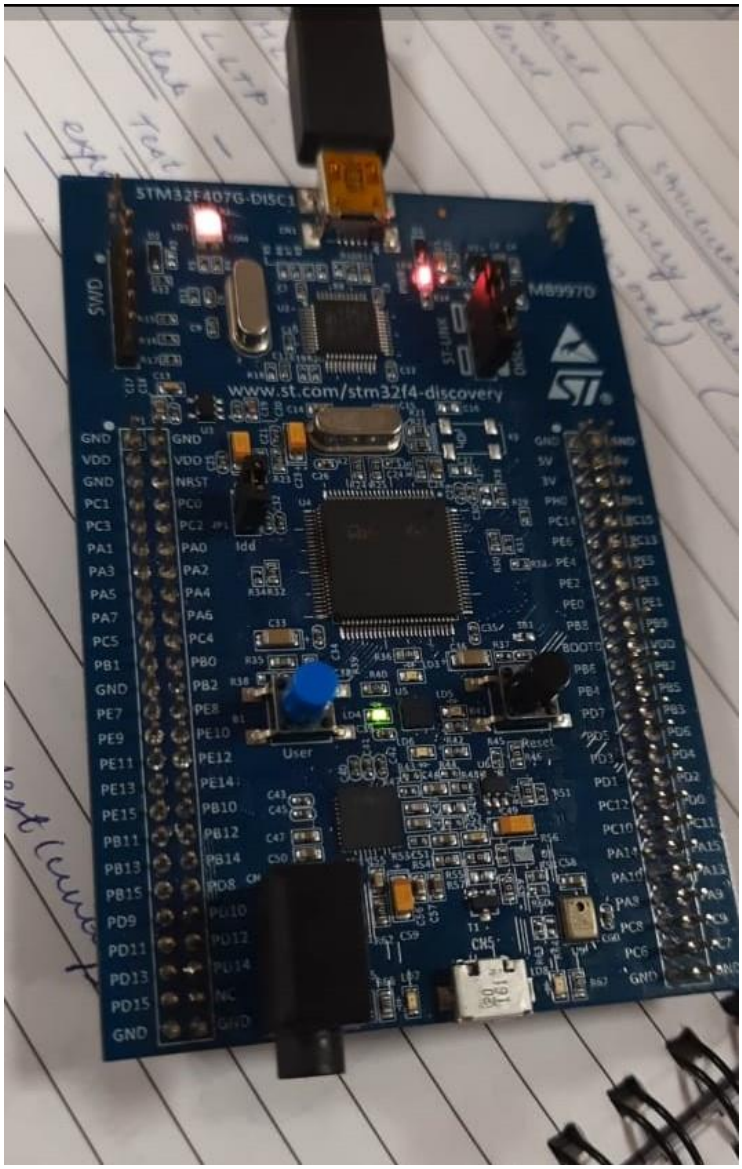

Figure 2: LED toggle output

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

```c
/*
 * opendrain_internalpullup.c
 *
 *  Created on: 20-Feb-2021
 *      Author: Training
 */


#include "stm32f4xx.h"
#include "stm32f407xx_gpio_driver.h"

void delay(void)
{
    for(uint32_t i=0; i<50000;i++);

}
int main(void)
{
    GPIO_Handle_t GpioLed;

    GpioLed.pGPIOx = GPIOD;
    GpioLed.GPIO_PinConfig.GPIO_PinNumber = GPIO_PIN_NO_12;
    GpioLed.GPIO_PinConfig.GPIO_PinMode = GPIO_MODE_OUT;
    GpioLed.GPIO_PinConfig.GPIO_PinSpeed = GPIO_SPEED_HIGH;
    GpioLed.GPIO_PinConfig.GPIO_PinPuPdControl = GPIO_NO_PU;
    GpioLed.GPIO_PinConfig.GPIO_PinOPType = GPIO_OP_TYPE_OD;


    //GPIO initialization

    GPIO_PeriClockControl(GPIOD, ENABLE);
    GPIO_Inint(&GpioLed);

    while(1)
    {
        GPIO_ToggleOutputPin(GPIOD, GPIO_PIN_NO_12);
        delay();
    }

    return 0;
}
```

Figure 3: open drain internal pull up code

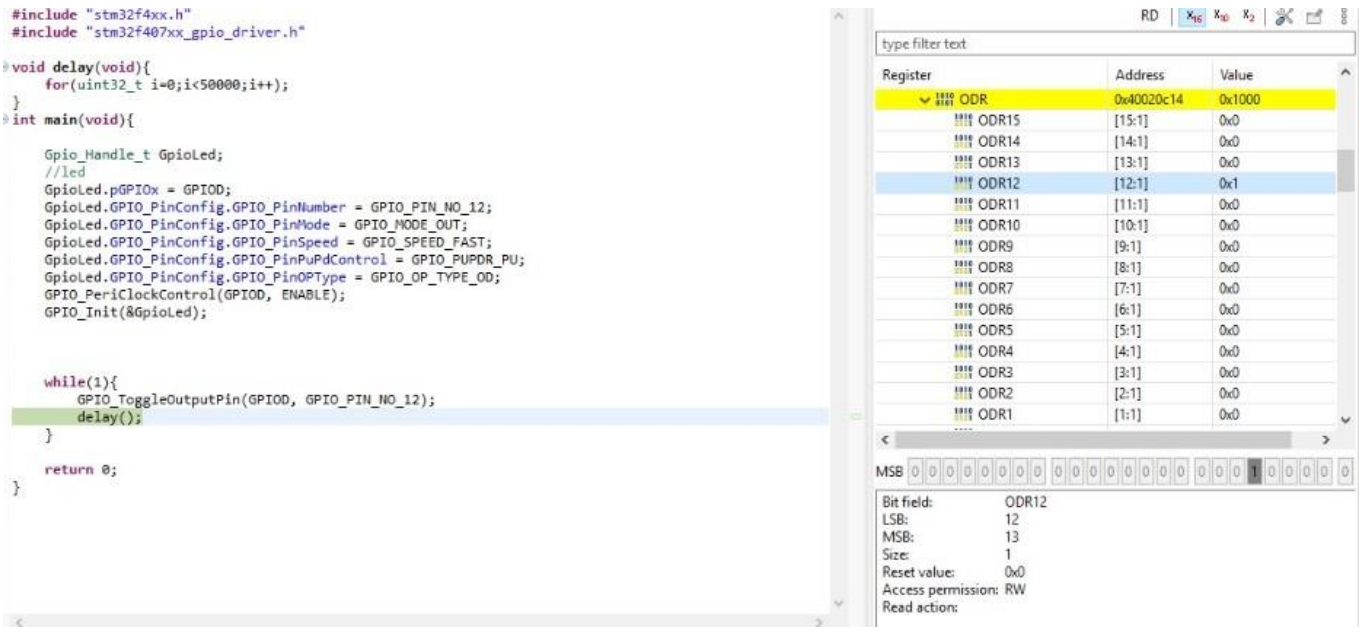

Figure 4: open drain internal pull up output

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

```c
/*
 * externalpullup.c
 *
 *  Created on: 24-Feb-2021
 *      Author: Training
 */

#include "stm32f4xx.h"
#include "stm32f407xx_gpio_driver.h"

void delay(void)
{
    for(uint32_t i=0; i<50000;i++);

}
int main(void)
{
    GPIO_Handle_t GpioLed;

    GpioLed.pGPIOx = GPIOD;
    GpioLed.GPIO_PinConfig.GPIO_PinNumber = GPIO_PIN_NO_11;
    GpioLed.GPIO_PinConfig.GPIO_PinMode = GPIO_MODE_OUT;
    GpioLed.GPIO_PinConfig.GPIO_PinSpeed = GPIO_SPEED_HIGH;
    GpioLed.GPIO_PinConfig.GPIO_PinPuPdControl = GPIO_NO_PUPD;
    GpioLed.GPIO_PinConfig.GPIO_PinOPType = GPIO_OP_TYPE_OD;

    //GPIO initialization

    GPIO_PeriClockControl(GPIOD, ENABLE);
    GPIO_Inint(&GpioLed);

    while(1)
    {
        GPIO_ToggleOutputPin(GPIOD, GPIO_PIN_NO_11);
        delay();
    }

    return 0;
}
```

Figure 5: open drain external pull up code

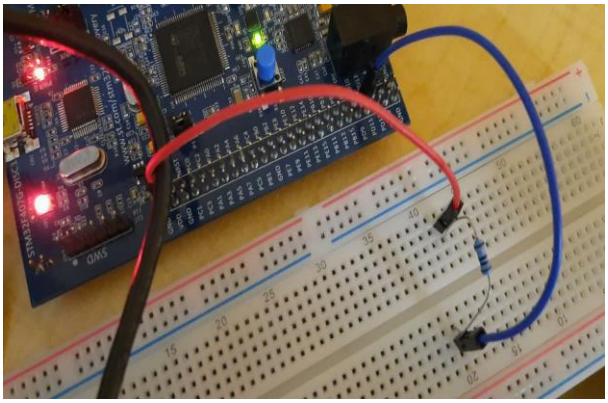

Figure 6: open drain external pull up output

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

```c
/*
 * pushpull_externalLED.c
 *
 *  Created on: 23-Feb-2021
 *      Author: Training
 */

#include "stm32f4xx.h"
#include "stm32f407xx_gpio_driver.h"

void delay(void)
{
    for(uint32_t i=0; i<5000;i++);

}
int main(void)
{
    GPIO_Handle_t GpioLed;

    GpioLed.pGPIOx = GPIOD;
    GpioLed.GPIO_PinConfig.GPIO_PinNumber = GPIO_PIN_NO_12;
    GpioLed.GPIO_PinConfig.GPIO_PinMode = GPIO_MODE_OUT;
    GpioLed.GPIO_PinConfig.GPIO_PinSpeed = GPIO_SPEED_HIGH;
    GpioLed.GPIO_PinConfig.GPIO_PinPuPdControl = GPIO_NO_PD;
    GpioLed.GPIO_PinConfig.GPIO_PinOPType = GPIO_OP_TYPE_PP;

    //GPIO initialization

    GPIO_PeriClockControl(GPIOD, ENABLE);
    GPIO_Inint(&GpioLed);

    while(1)
    {
        GPIO_ToggleOutputPin(GPIOD, GPIO_PIN_NO_12);
        delay();
    }

    return 0;
}
```

Figure 7: External LED Push pull code

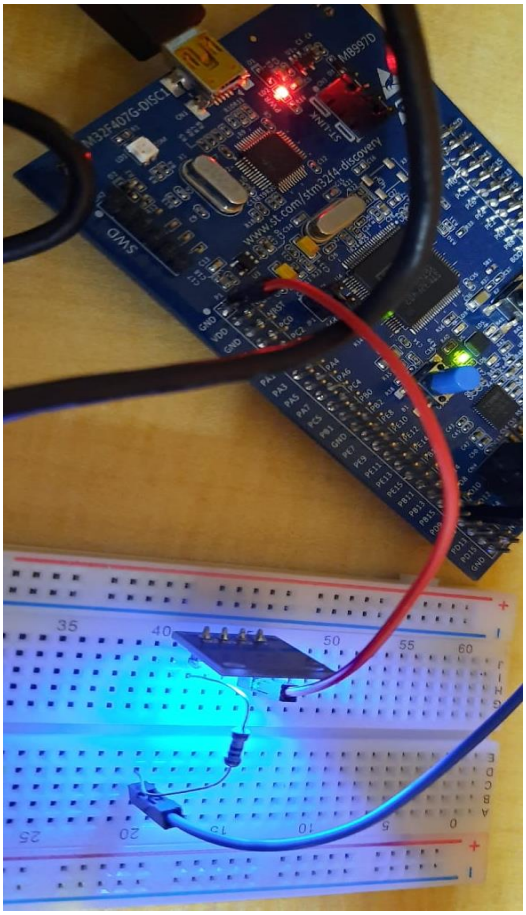

Figure 8: External LED Push Pull output

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

```c
/*
 * button.c
 *
 *  Created on: 20-Feb-2021
 *      Author: Training
 */

#include "stm32f4xx.h"
#include "stm32f407xx_gpio_driver.h"

#define HIGH         1
#define BTN_PRESSED  HIGH

void delay(void)
{
    for(uint32_t i=0; i<50000/2;i++);

}
int main(void)
{
    GPIO_Handle_t GpioLed;
    GPIO_Handle_t GpioBtn;
//LED
    GpioLed.pGPIOx = GPIOD;
    GpioLed.GPIO_PinConfig.GPIO_PinNumber = GPIO_PIN_NO_12;
    GpioLed.GPIO_PinConfig.GPIO_PinMode = GPIO_MODE_OUT;
    GpioLed.GPIO_PinConfig.GPIO_PinSpeed = GPIO_SPEED_HIGH;
    GpioLed.GPIO_PinConfig.GPIO_PinPuPdControl = GPIO_NO_PD;
    GpioLed.GPIO_PinConfig.GPIO_PinOPType = GPIO_OP_TYPE_PP;

    //GPIO initialization

    GPIO_PeriClockControl(GPIOD, ENABLE);
    GPIO_Inint(&GpioLed);
//Button
    GpioBtn.pGPIOx = GPIOA;
    GpioBtn.GPIO_PinConfig.GPIO_PinNumber = GPIO_PIN_NO_0;
    GpioBtn.GPIO_PinConfig.GPIO_PinMode = GPIO_MODE_IN;
    GpioBtn.GPIO_PinConfig.GPIO_PinSpeed = GPIO_SPEED_HIGH;
    GpioBtn.GPIO_PinConfig.GPIO_PinPuPdControl = GPIO_NO_PD;

```

Figure 9: internal button led toggle code

```
42
43
44          //GPIO initialization
45
46          GPIO_PeriClockControl(GPIOA, ENABLE);
47          GPIO_Inint(&GpioBtn);
48
49      while(1)
50      {
51          if(GPIO_ReadFromInputPin(GPIOA, GPIO_PIN_NO_0)== BTN_PRESSED)
52          {
53          GPIO_ToggleOutputPin(GPIOD, GPIO_PIN_NO_12);
54          delay();
55          }
56      }
57
58      return 0;
59  }
60
```

Figure 10: internal button led toggle code

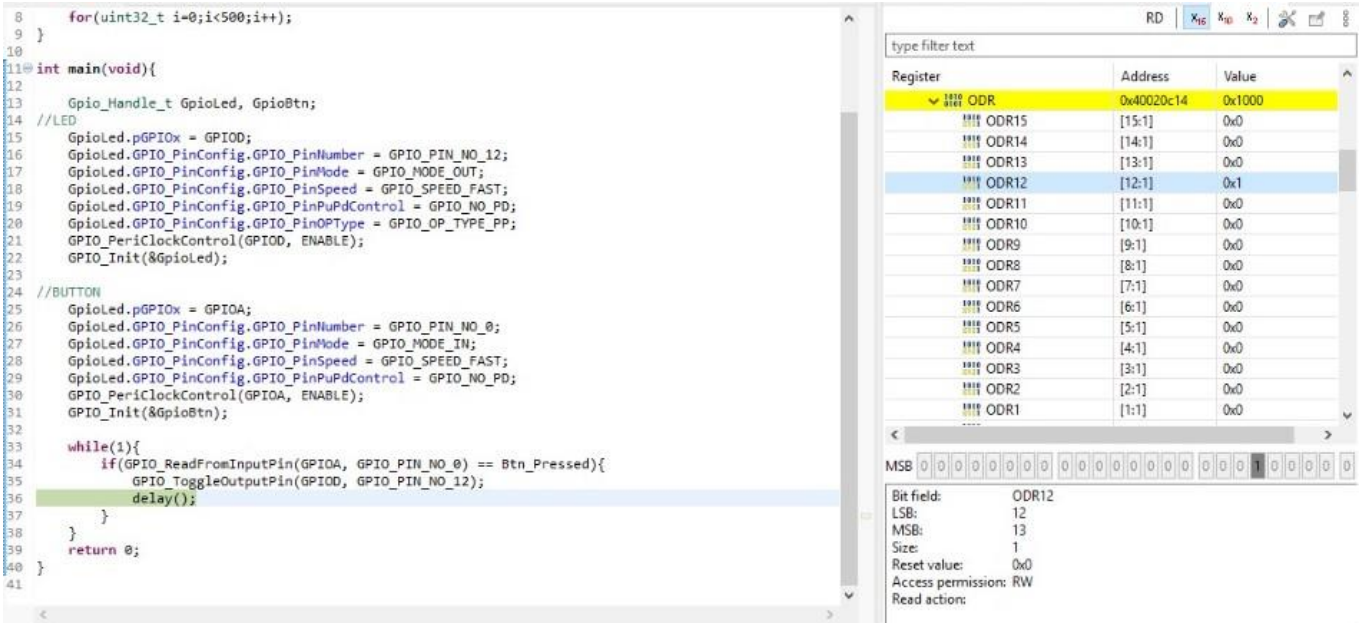

Figure 11: internal button led toggle output

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

```c
/*
 * ExternalSwitch_LED.c
 *
 *  Created on: 23-Feb-2021
 *      Author: Training
 */

#include "stm32f4xx.h"
#include "stm32f407xx_gpio_driver.h"

#define HIGH         1
#define BTN_PRESSED  HIGH

void delay(void)
{
    for(uint32_t i=0; i<50000/2;i++);

}
int main(void)
{
    GPIO_Handle_t GpioLed;
    GPIO_Handle_t GpioBtn;
//LED
    GpioLed.pGPIOx = GPIOD;
    GpioLed.GPIO_PinConfig.GPIO_PinNumber = GPIO_PIN_NO_12;
    GpioLed.GPIO_PinConfig.GPIO_PinMode = GPIO_MODE_OUT;
    GpioLed.GPIO_PinConfig.GPIO_PinSpeed = GPIO_SPEED_HIGH;
    GpioLed.GPIO_PinConfig.GPIO_PinPuPdControl = GPIO_NO_PD;
    GpioLed.GPIO_PinConfig.GPIO_PinOPType = GPIO_OP_TYPE_PP;

    //GPIO initialization

    GPIO_PeriClockControl(GPIOD, ENABLE);
    GPIO_Inint(&GpioLed);
//Button
    GpioBtn.pGPIOx = GPIOA;
    GpioBtn.GPIO_PinConfig.GPIO_PinNumber = GPIO_PIN_NO_0;
    GpioBtn.GPIO_PinConfig.GPIO_PinMode = GPIO_MODE_IN;
    GpioBtn.GPIO_PinConfig.GPIO_PinSpeed = GPIO_SPEED_HIGH;
    GpioBtn.GPIO_PinConfig.GPIO_PinPuPdControl = GPIO_NO_PUPD;

```

Figure 12: external switch led toggle

```c
#include "stm32f4xx.h"
#include "stm32f407xx_gpio_driver.h"

#define HIGH            1
#define Btn_Pressed     HIGH

void delay(void){
    for(uint32_t i=0;i<50000;i++);
}

int main(void){

    Gpio_Handle_t GpioLed, GpioBtn;
//LED
    GpioLed.pGPIOx = GPIOD;
    GpioLed.GPIO_PinConfig.GPIO_PinNumber = GPIO_PIN_NO_12;
    GpioLed.GPIO_PinConfig.GPIO_PinMode = GPIO_MODE_OUT;
    GpioLed.GPIO_PinConfig.GPIO_PinSpeed = GPIO_SPEED_FAST;
    GpioLed.GPIO_PinConfig.GPIO_PinPuPdControl = GPIO_NO_PD;
    GpioLed.GPIO_PinConfig.GPIO_PinOPType = GPIO_OP_TYPE_PP;
    GPIO_PeriClockControl(GPIOD, ENABLE);
    GPIO_Init(&GpioLed);

//BUTTON
    GpioLed.pGPIOx = GPIOA;
    GpioLed.GPIO_PinConfig.GPIO_PinNumber = GPIO_PIN_NO_7;
    GpioLed.GPIO_PinConfig.GPIO_PinMode = GPIO_MODE_IN;
    GpioLed.GPIO_PinConfig.GPIO_PinSpeed = GPIO_SPEED_FAST;
    GpioLed.GPIO_PinConfig.GPIO_PinPuPdControl = GPIO_NO_PD;
    GPIO_PeriClockControl(GPIOA, ENABLE);
    GPIO_Init(&GpioBtn);

    while(1){
        if(GPIO_ReadFromInputPin(GPIOA, GPIO_PIN_NO_7) == Btn_Pressed){
            GPIO_ToggleOutputPin(GPIOD, GPIO_PIN_NO_12);
            delay();
        }
    }
    return 0;
}
```

RD

type filter text

Register	Address	Value
GPIOE		
GPIOD		
MODER	0x40020c00	0x1000000
OTYPER	0x40020c04	0x0
OSPEEDR	0x40020c08	0x20
PUPDR	0x40020c0c	0x0
IDR	0x40020c10	0x9fef
ODR	0x40020c14	0x1000
ODR15	[15:1]	0x0
ODR14	[14:1]	0x0
ODR13	[13:1]	0x0
ODR12	[12:1]	0x1
ODR11	[11:1]	0x0
ODR10	[10:1]	0x0
ODR9	[9:1]	0x0
ODR8	[8:1]	0x0
ODR7	[7:1]	0x0
ODR6	[6:1]	0x0
ODR5	[5:1]	0x0
ODR4	[4:1]	0x0
ODR3	[3:1]	0x0
ODR2	[2:1]	0x0
ODR1	[1:1]	0x0
ODR0	[0:1]	0x0

MSB 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

Bit field: ODR12
LSB: 12
MSB: 13
Size: 1
Reset value: 0x0
Access permission: RW
Read action:

Description:
Port output data (y = 0..15)

Figure 13: external switch led toggle code

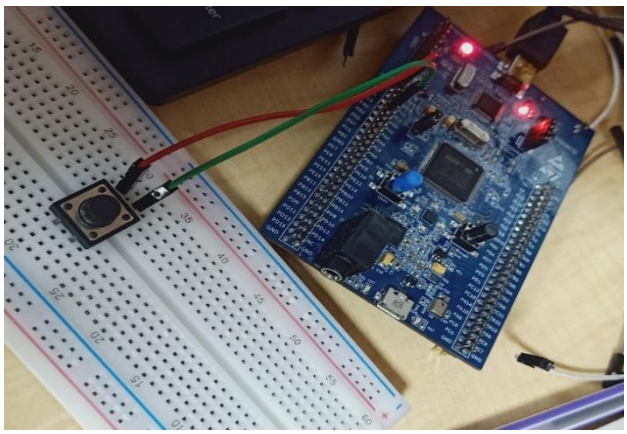

Figure 14: external switch led toggle output

5.7 HAL INTERRUPT

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
```

Figure 15: HAL interrupt

```

/* USER CODE BEGIN PV */
volatile uint8_t flag=0;

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */
```

Figure 16: HAL interrupt

```
  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
      if(1==flag)
      {
          HAL_Delay(1000);
          HAL_GPIO_TogglePin(GREEN_LED_GPIO_Port, GREEN_LED_Pin);
          flag=0;
      }
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
```

Figure 17: HAL interrupt

```
  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
```

Figure 18: HAL interrupt

```
160
161   /*Configure GPIO pin Output Level */
162   HAL_GPIO_WritePin(GREEN_LED_GPIO_Port, GREEN_LED_Pin, GPIO_PIN_RESET);
163
164   /*Configure GPIO pin : BLUE_BTN_Pin */
165   GPIO_InitStruct.Pin = BLUE_BTN_Pin;
166   GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
167   GPIO_InitStruct.Pull = GPIO_NOPULL;
168   HAL_GPIO_Init(BLUE_BTN_GPIO_Port, &GPIO_InitStruct);
169
170   /*Configure GPIO pin : GREEN_LED_Pin */
171   GPIO_InitStruct.Pin = GREEN_LED_Pin;
172   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
173   GPIO_InitStruct.Pull = GPIO_NOPULL;
174   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
175   HAL_GPIO_Init(GREEN_LED_GPIO_Port, &GPIO_InitStruct);
176
177   /* EXTI interrupt init*/
178   HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
179   HAL_NVIC_EnableIRQ(EXTI0_IRQn);
180
181 }
182
183 /* USER CODE BEGIN 4 */
184 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
185 {
186     flag=1;
187 }
188 /* USER CODE END 4 */
189
190 /**
191   * @brief  This function is executed in case of error occurrence.
192   * @retval None
193   */
194 void Error_Handler(void)
195 {
196   /* USER CODE BEGIN Error_Handler_Debug */
197   /* User can add his own implementation to report the HAL error return state */
198
199   /* USER CODE END Error_Handler_Debug */
200 }
201
```

Figure 19: HAL interrupt

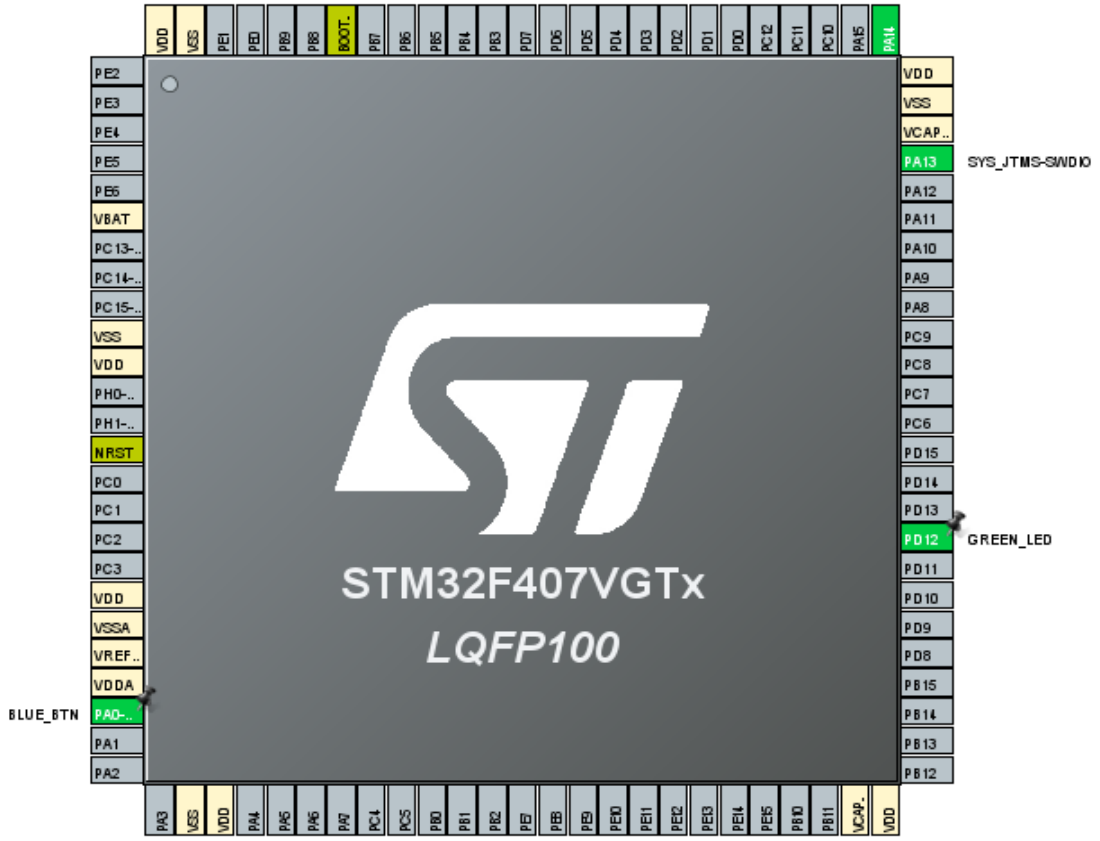

Figure 20: HAL interrupt

5.8 HAL ADC

```
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint32_t adcVal1, adcVal2;
ADC_ChannelConfTypeDef sConfig ;
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */
```

Figure 21: HAL ADC

```
  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      HAL_ADC_Start(&hadc1);
      if(HAL_ADC_PollForConversion(&hadc1, 5) ==   HAL_OK)
      {
          adcVal0 = HAL_ADC_GetValue(&hadc1);
      }
      HAL_Delay(50);
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
```

Figure 22: HAL ADC

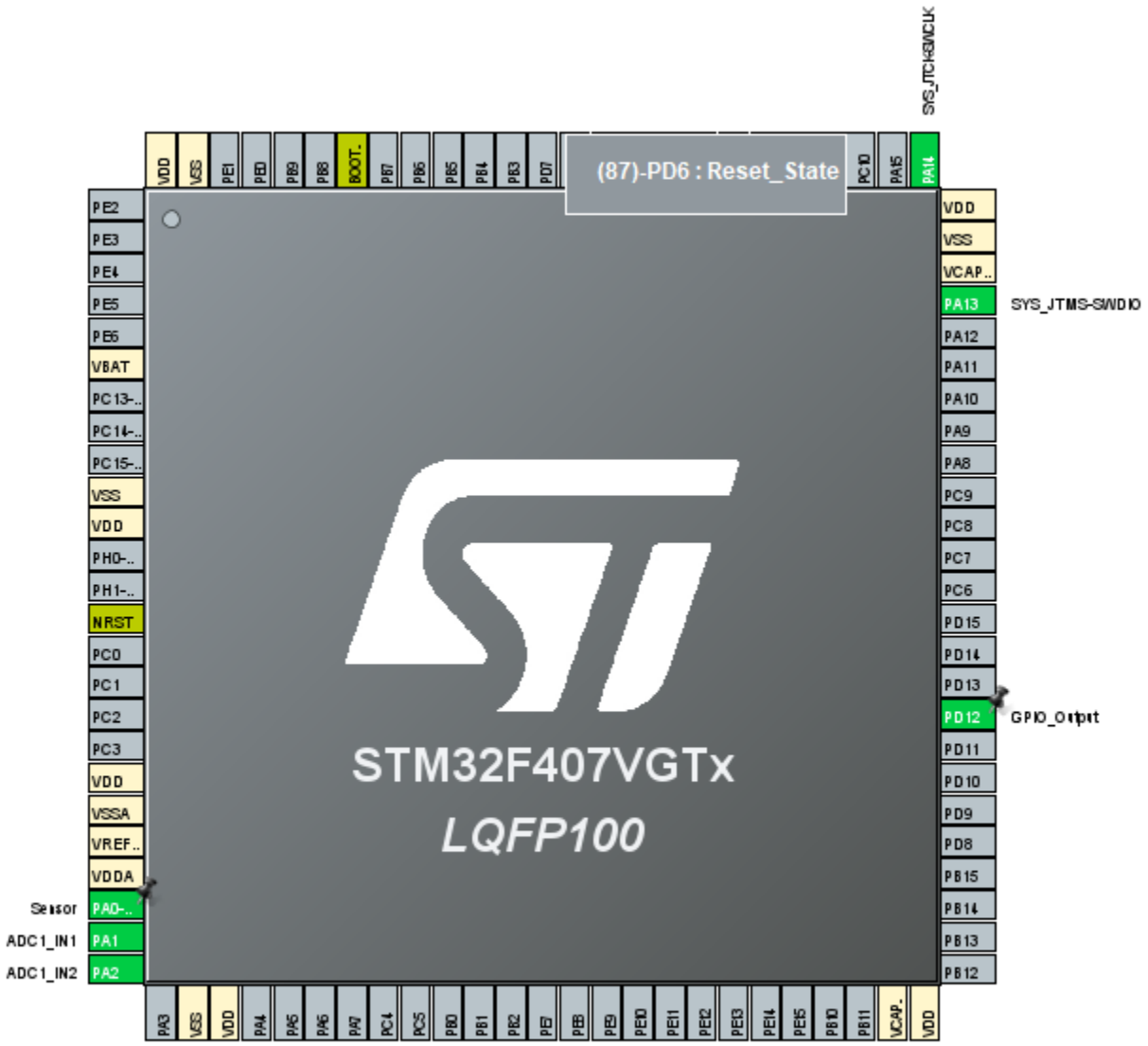

Figure 23: HAL ADC

5.9 HAL SPI

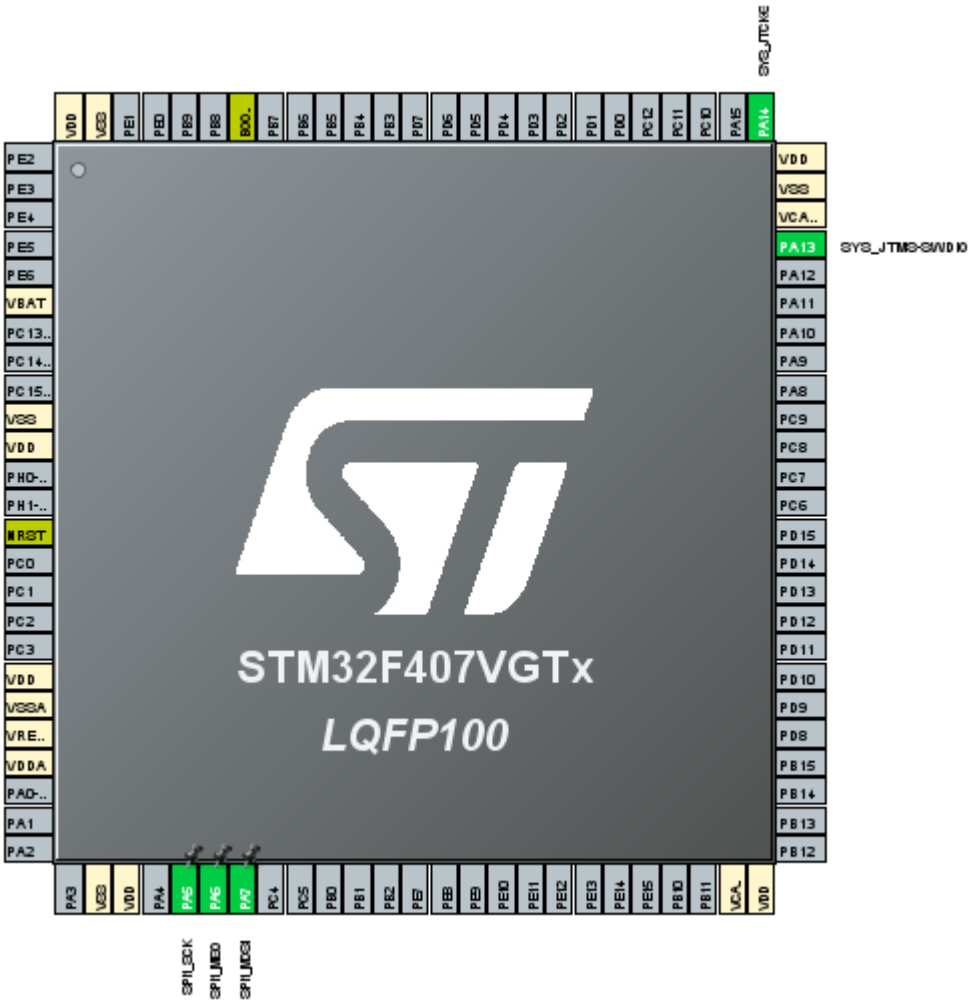

Figure 24: HAL SPI

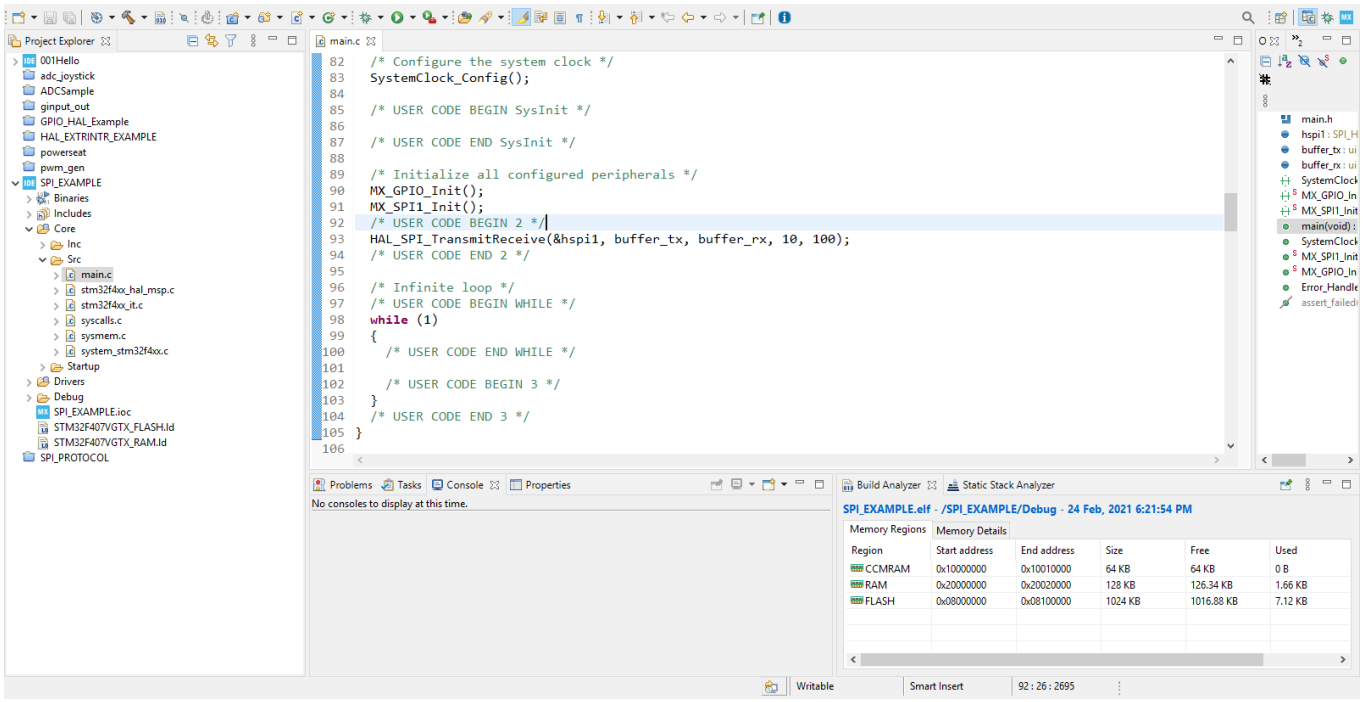

Figure 25: Output of SPI

5.10 HAL UART

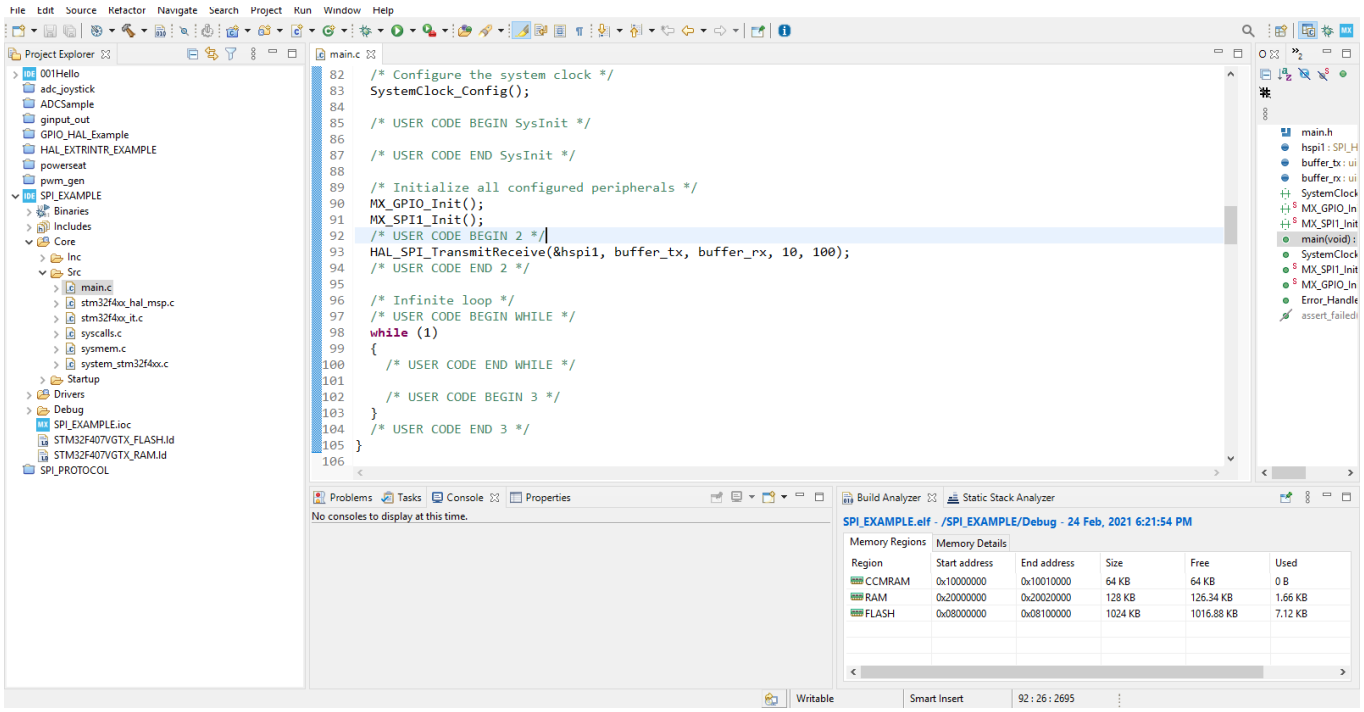

Figure 26: Output of UART