



# Learning Report – Embedded C



*L&T Technology Services*



## Document History

Ver. Rel. No.	Release Date	Prepared. By	Reviewed By	Approved By	Remarks/Revision Details
	23/02/2021	Srushti Wilson Rangari		Dr.Vivek Kaundal, Bhargav N	First Draft
	25/02/2021	Srushti Wilson Rangari		Dr.Vivek Kaundal, Bhargav N	Second Draft
	27/02/2021	Srushti Wilson Rangari		Dr.Vivek Kaundal, Bhargav N	Final Draft

## Contents

<b>ACTIVITY 1 – COMPILATION APPROACH.....</b>	<b>4</b>
1.1- MAKE FILE .....	<b>4ERROR! BOOKMARK NOT DEFINED.</b>
1.2- STARTUP CODE.....	5
1.3- LINKER FILE.....	8
1.4- DEBUGGING TECHNIQUES.....	9
 <b>ACTIVITY 2 – IMPLEMENTATION OF PROTOCOLS USING STM IDE .....</b>	 <b>10</b>
2.1 GPIO.....	10
2.2 EXTI.....	12
2.3 ADC.....	14
2.4 SPI.....	16
2.5 UART.....	19
 <b>ACTIVITY 3 – PROJECT ON BCM MODULE .....</b>	 <b>21</b>

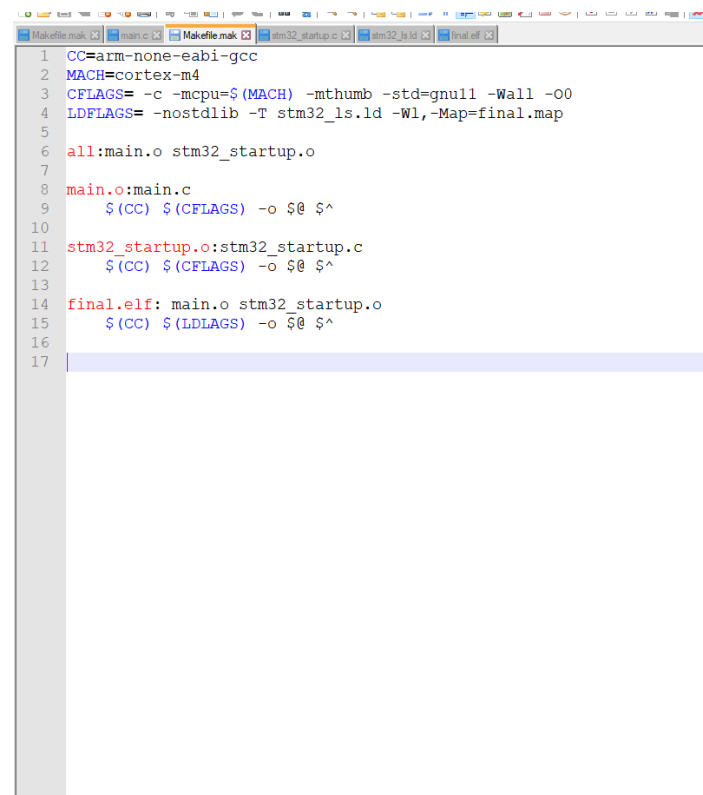
## Activity 1 – COMPILATION APPROACH

This is the complete compilation process of the sample program for ARM Cortex Mx processor based boards. Following are the compilation stages of a C program:

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

### 1.1- MAKE FILE


Below is the make file for the sample program:



```
1 CC=arm-none-eabi-gcc
2 MACH=cortex-m4
3 CFLAGS= -c -mcpu=$(MACH) -mthumb -std=gnu11 -Wall -O0
4 LDFLAGS= -nostdlib -T stm32_1s.ld -Wl,-Map=final.map
5
6 all:main.o stm32_startup.o
7
8 main.o:main.c
9     $(CC) $(CFLAGS) -o $@ $^
10
11 stm32_startup.o:stm32_startup.c
12     $(CC) $(CFLAGS) -o $@ $^
13
14 final.elf: main.o stm32_startup.o
15     $(CC) $(LDFLAGS) -o $@ $^
16
17
```

Fig 1.1.1 make file

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



```

C:\Windows\System32\cmd.exe

:\Users\Training\Desktop\Activity>make -f Makefile.mak
arm-none-eabi-gcc -c -mcpu=cortex-m4 -mthumb -std=gnu11 -Wall -O0 -o main.o main.c

:\Users\Training\Desktop\Activity>arm-none-eabi-gcc -nostdlib -T stm32_ls.ld *.o -o final.elf

:\Users\Training\Desktop\Activity>
```

Fig 1.1.2 Make command

- **-mcpu=cortex-m4** is used to select our cortex-m4 processor which is used
- **-mthumb** is used to generate the code that executes in ARM state
- **main.o** is the target file
- **main.c** is the dependency

## 1.2- STARTUP CODE

- The startup file is responsible for setting up the right environments to run the code in main.c file.
- Some part of the startup code is target (processor) dependent.
- Role of startup file:
  1. Create a MCU specific vector table for microcontroller.
  2. To write a startup code which initializes .data and .bss section in SRAM.
  3. Call main()

```

1 #include<stdint.h>
2
3 #define SRAM_START 0x20000000U
4 #define SRAM_SIZE (128U * 1024U) //128KB
5 #define SRAM_END ((SRAM_START) + (SRAM_SIZE))
6
7 #define STACK_START SRAM_END
8
9 extern uint32_t _etext;
10 extern uint32_t _edata;
11 extern uint32_t _e_data;
12 extern uint32_t _la_data;
13
14 extern uint32_t _ebss;
15 extern uint32_t _ebss;
16
17 //prototype of main
18
19 int main(void);
20
21 void __libc_init_array(void);
22
23
24 /* function prototypes of STM32F407x system exception and IRQ handlers */
25
26 void Reset_Handler(void);
27
28 void NMI_Handler (void) __attribute__((weak, alias("Default_Handler")));
29 void HardFault_Handler (void) __attribute__((weak, alias("Default_Handler")));
30 void MemManage_Handler (void) __attribute__((weak, alias("Default_Handler")));
31 void BusFault_Handler (void) __attribute__((weak, alias("Default_Handler")));
32 void UsageFault_Handler (void) __attribute__((weak, alias("Default_Handler")));
33 void SVC_Handler (void) __attribute__((weak, alias("Default_Handler")));
34 void DebugMon_Handler (void) __attribute__((weak, alias("Default_Handler")));
35 void PendSV_Handler (void) __attribute__((weak, alias("Default_Handler")));
36 void SysTick_Handler (void) __attribute__((weak, alias("Default_Handler")));
37 void WWDG_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
38 void PVD_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
39 void TAMP_STAMP_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
40 void RTC_WKUP_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
41 void RCC_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
42 void EXTI0_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
43 void EXTI1_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
44 void EXTI2_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
45 void EXTI3_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
46 void EXTI4_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
47 void DMA1_Stream0_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
48 void DMA1_Stream1_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
49 void DMA1_Stream2_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
50 void DMA1_Stream3_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
51 void DMA1_Stream4_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
52 void DMA1_Stream5_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
53 void DMA1_Stream6_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
54 void ADC_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
55 void CAN1_TX_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
56 void CAN1_RX0_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
57 void CAN1_RX1_IRQHandler (void) __attribute__((weak, alias("Default_Handler")));
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252

```

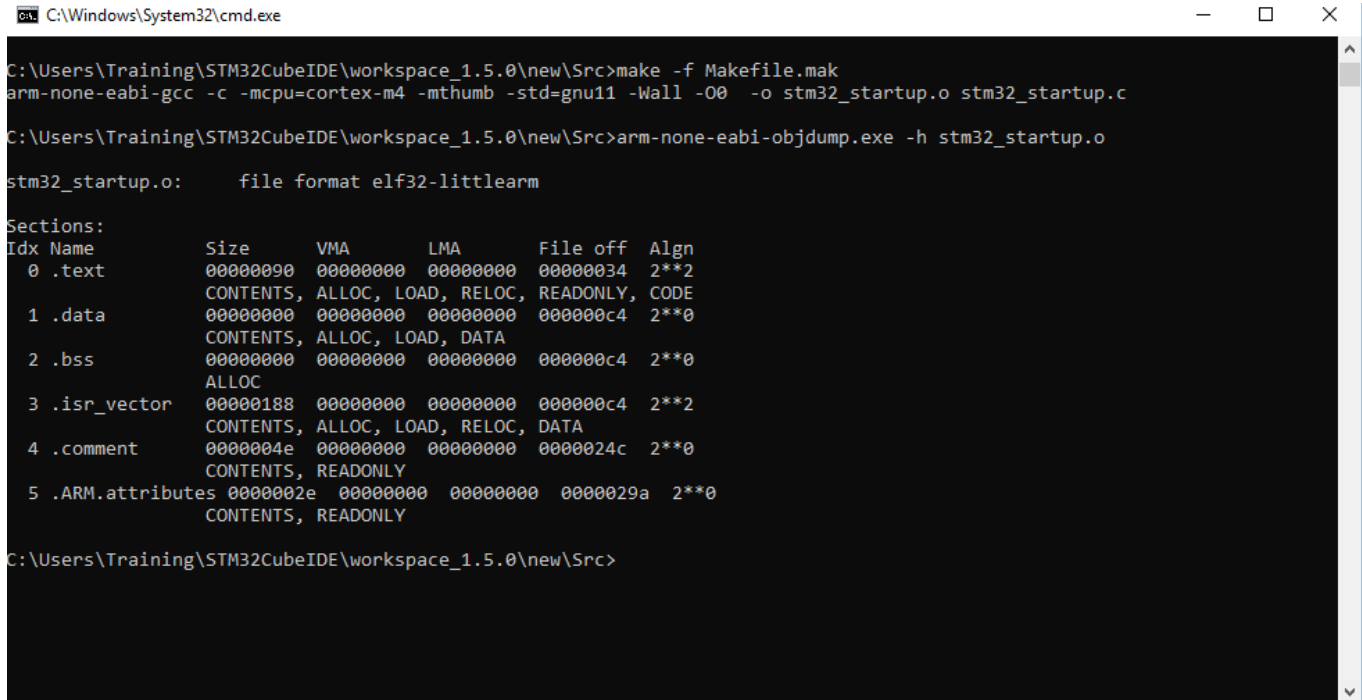
Fig 1.2.1 Startup code

In startup code we use variable attributes to store some variables in the user defined function.

Function attributes:

- Weak: Lets programmer override already defined weak function (dummy function) with the same function name.
- Alias: Lets programmer give any alias name for same function.

The startup.o file generated is of elf executable format, various sections of which are shown below:



```

C:\Windows\System32\cmd.exe

C:\Users\Training\STM32CubeIDE\workspace_1.5.0\new\Src>make -f Makefile.mak
arm-none-eabi-gcc -c -mcpu=cortex-m4 -mthumb -std=gnu11 -Wall -O0 -o stm32_startup.o stm32_startup.c

C:\Users\Training\STM32CubeIDE\workspace_1.5.0\new\Src>arm-none-eabi-objdump.exe -h stm32_startup.o

stm32_startup.o:      file format elf32-littlearm

Sections:
Idx Name              Size      VMA           LMA           File off  Algn
 0 .text              00000090  00000000  00000000  00000034  2**2
   CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
 1 .data              00000000  00000000  00000000  000000c4  2**0
   CONTENTS, ALLOC, LOAD, DATA
 2 .bss               00000000  00000000  00000000  000000c4  2**0
   ALLOC
 3 .isr_vector        00000188  00000000  00000000  000000c4  2**2
   CONTENTS, ALLOC, LOAD, RELOC, DATA
 4 .comment           0000004e  00000000  00000000  0000024c  2**0
   CONTENTS, READONLY
 5 .ARM.attributes    0000002e  00000000  00000000  0000029a  2**0
   CONTENTS, READONLY

C:\Users\Training\STM32CubeIDE\workspace_1.5.0\new\Src>

```

Fig 1.2.2: Startup command

### 1.3- LINKER SCRIPT

- Linkers take one or more object files or libraries as input and combines them to create a single executable file as output.
- Linker scripts decide how different sections of object file should be merged to create an output file.
- Reset handler is the entry point to the application
- Entry command is used to set the “Entry point address” information in the header of final elf file generated.

Syntax: Entry(symbol\_name)

Entry(Reset\_Handler)

```
4
5 {
6   FLASH(cx): ORIGIN = 0x00000000, LENGTH = 1024K
7   BRAM(lmx): ORIGIN = 0x20000000, LENGTH = 128K
8 }
9
10 SECTIONS
11 {
12   .text :
13   {
14     *(.isr_vector)
15     *(.text)
16     *(.text.*)
17     *(.init)
18     *(.fini)
19     *(.rodata)
20     *(.rodata.*)
21     . = ALIGN(4);
22     _etext = .;
23   } > FLASH
24
25   _la_data = LOADADDR(.data);
26
27   .data :
28   {
29     _edata = .;
30     *(.data)
31     *(.data.*)
32     . = ALIGN(4);
33     _edata = .;
34   } > BRAM AT> FLASH
35
36   .bss :
37   {
38     _bss = .;
39     __bss_start__ = _bss;
40     *(.bss)
41     *(.bss.*)
42     *(COMMON)
43     . = ALIGN(4);
44     _ebss = .;
45     __bss_end__ = _ebss;
46     . = ALIGN(4);
47     end = .;
48     __end__ = .;
49   } > BRAM
50
51 }
```



```
C:\Windows\System32\cmd.exe
C:\Users\Training\Documents\Embedded C\STM project\Activity1>arm-none-eabi-gcc -nostdlib -T stm32_ls.ld *.o -o final.elf
c:/program files (x86)/gnu arm embedded toolchain/10 2020-q4-major/bin/../lib/gcc/arm-none-eabi/10.2.1/../../../../arm-none-eabi/bin/ld.exe: stm3
2_startup.o: in function `Reset_Handler':
stm32_startup.c:(.text+0x80): undefined reference to `_la_data'
collect2.exe: error: ld returned 1 exit status

C:\Users\Training\Documents\Embedded C\STM project\Activity1>arm-none-eabi-gcc -nostdlib -T stm32_ls.ld *.o -o final.elf

C:\Users\Training\Documents\Embedded C\STM project\Activity1>dir
Volume in drive C has no label.
Volume Serial Number is F8FC-05CA

Directory of C:\Users\Training\Documents\Embedded C\STM project\Activity1

23-02-2021  14:29    <DIR>          .
23-02-2021  14:29    <DIR>          ..
23-02-2021  14:29             135,752 final.elf
23-02-2021  14:24             2,034 main.c
23-02-2021  14:24             1,940 main.o
23-02-2021  11:35              220 Makefile.mak
23-02-2021  14:28              662 stm32_ls.ld
23-02-2021  14:27            12,847 stm32_startup.c
23-02-2021  14:24             5,688 stm32_startup.o
                7 File(s)          159,143 bytes
                2 Dir(s)        125,771,227,136 bytes free

C:\Users\Training\Documents\Embedded C\STM project\Activity1>
```

Fig 1.3.1 command to generate final.elf file

## 1.4- DEBUGGING TECHNIQUES

- The STM32F407VG is embedded with on chip debugger for debugging the code.
- The OCD ON-Chip Debugger aims to provide debugging, in system programming and boundary scan testing for embedded target devices.
- OCD is a free and opensource host application allows you to program, debug, and analyze your applications using GDB.
- It supports various target boards based on different processor architecture.

## Activity 2 – IMPLEMENTATION OF PROTOCOLS USING STM IDE

Implementation of protocols for STM32F407VG microcontroller featuring ARM32 bit ARM-cortex - M4 with FPU core using HAL library.

### 2.1 GPIO:

Toggling LED at pin PD12 at GREEN\_LED\_GPIO\_PORT. Serial wire is enabled at pin PA13.

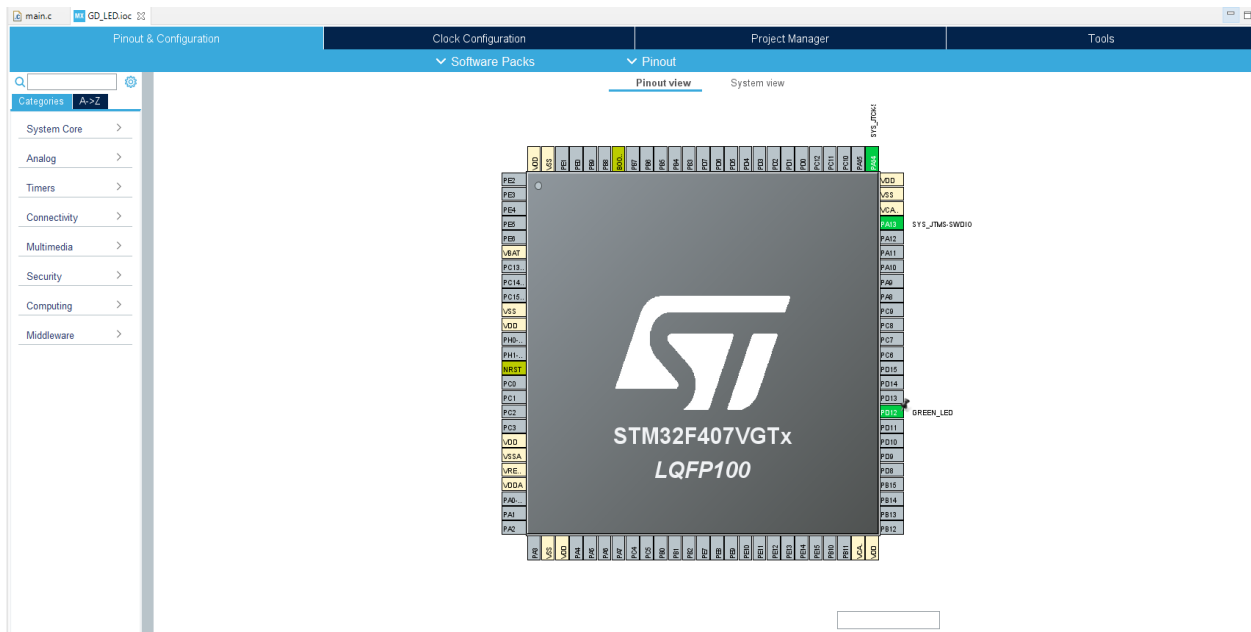


Fig: 2.1.1 GPIO pin configuration

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

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    HAL_GPIO_TogglePin(GREEN_LED_GPIO_Port, GREEN_LED_Pin);
    HAL_Delay(500);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}
```

Fig: 2.1.2 GPIO configuration code

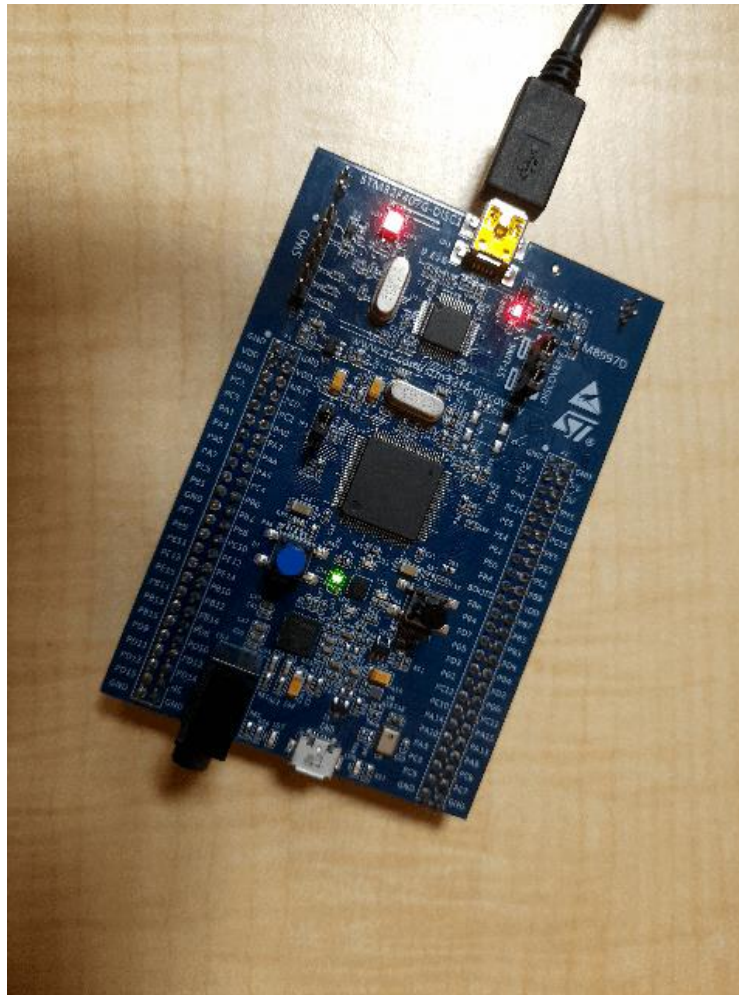


Fig: 2.1.2 LED toggling

## 2.2 EXTI:

Blue button at PA0 works as an external interrupt.

When the blue button is pressed the Green LED at pin PD12 toggles.

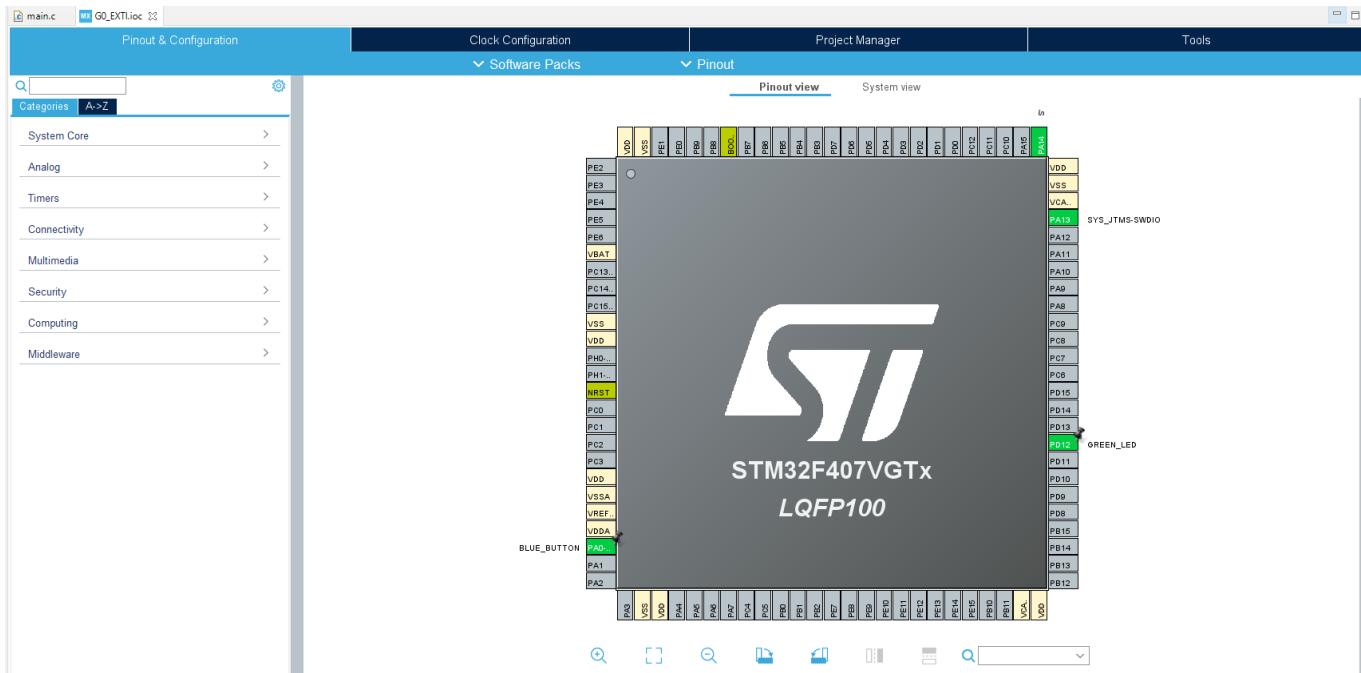


Fig: 2.2.1 EXTI pin configuration

In the main.c file a flag is initialized and if the flag == 1, the condition under the if loop executed to toggle the LED at PD12.

```

43
44 /* USER CODE BEGIN PV */
45 uint8_t flag = 0;
46 /* USER CODE END PV */
47
48 /* Private function prototypes -----*/
49 void SystemClock_Config(void);
50 static void MX_GPIO_Init(void);
51 /* USER CODE BEGIN PFP */
52
53 /* USER CODE END PFP */
54
55 /* Private user code -----*/
56 /* USER CODE BEGIN 0 */
57
58 /* USER CODE END 0 */
59
60 /**
61  * @brief The application entry point.
62  * @retval int
63  */
64 int main(void)
65 {
66     /* USER CODE BEGIN 1 */
67
68     /* USER CODE END 1 */
69
70     /* MCU Configuration -----*/
71
72     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
73     HAL_Init();
74
75     /* USER CODE BEGIN Init */
76
77     /* USER CODE END Init */
78
79     /* Configure the system clock */
80     SystemClock_Config();
81
82     /* USER CODE BEGIN SysInit */
83
84     /* USER CODE END SysInit */
85
86     /* Initialize all configured peripherals */
87     MX_GPIO_Init();
88     /* USER CODE BEGIN 2 */
89
90     /* USER CODE END 2 */
91
92     /* Infinite loop */
93     /* USER CODE BEGIN WHILE */
94     while (1)
95     {
96         /* USER CODE END WHILE */
97
98         if(!flag){
99             HAL_GPIO_TogglePin(GREEN_LED_GPIO_Port, GREEN_LED_Pin);
100             HAL_Delay(500);
101             flag = 0;
102         }
103         /* USER CODE BEGIN 3 */
104
105     }
106 }

```

Fig: 2.2.1 EXTI configuration code

Fig: 2.3.1 ADC pin configuration

```
*main.c
19 /* USER CODE END Header */
20 /* Includes -----*/
21 #include "main.h"
22 #include "usb_host.h"
23
24
25 ADC_HandleTypeDef hadc1;
26
27 I2C_HandleTypeDef hi2c1;
28
29 I2S_HandleTypeDef hi2s3;
30
31 SPI_HandleTypeDef hspi1;
32
33
34 /* Private function prototypes -----*/
35 void SystemClock_Config(void);
36 static void MX_GPIO_Init(void);
37 static void MX_I2C1_Init(void);
38 static void MX_I2S3_Init(void);
39 static void MX_SPI1_Init(void);
40 static void MX_ADC1_Init(void);
41 void MX_USB_HOST_Process(void);
42
43
44 uint32_t adc_value;
45
46 int main(void)
47 {
48     HAL_Init();
49     SystemClock_Config();
50     MX_GPIO_Init();
51     MX_I2C1_Init();
52     MX_I2S3_Init();
53
54     MX_GPIO_Init();
55     MX_I2C1_Init();
56     MX_I2S3_Init();
57
58     MX_GPIO_Init();
59     MX_I2C1_Init();
60     MX_I2S3_Init();
61     MX_SPI1_Init();
62     MX_USB_HOST_Init();
63     MX_ADC1_Init();
64
65     while (1)
66     {
67         MX_USB_HOST_Process();
68
69         HAL_ADC_Start(&hadc1);
70
71         if(HAL_ADC_PollForConversion(&hadc1, 5) == HAL_OK){
72             adc_value= HAL_ADC_GetValue(&hadc1);
73         }
74
75         HAL_ADC_Stop(&hadc1);
76         HAL_Delay(100);
77     }
78 }
79
80
81
82
83
84 void SystemClock_Config(void)
85 {
86     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
87     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
88     RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};
```

Fig: 2.3.2 ADC configuration code

## 2.4 SPI

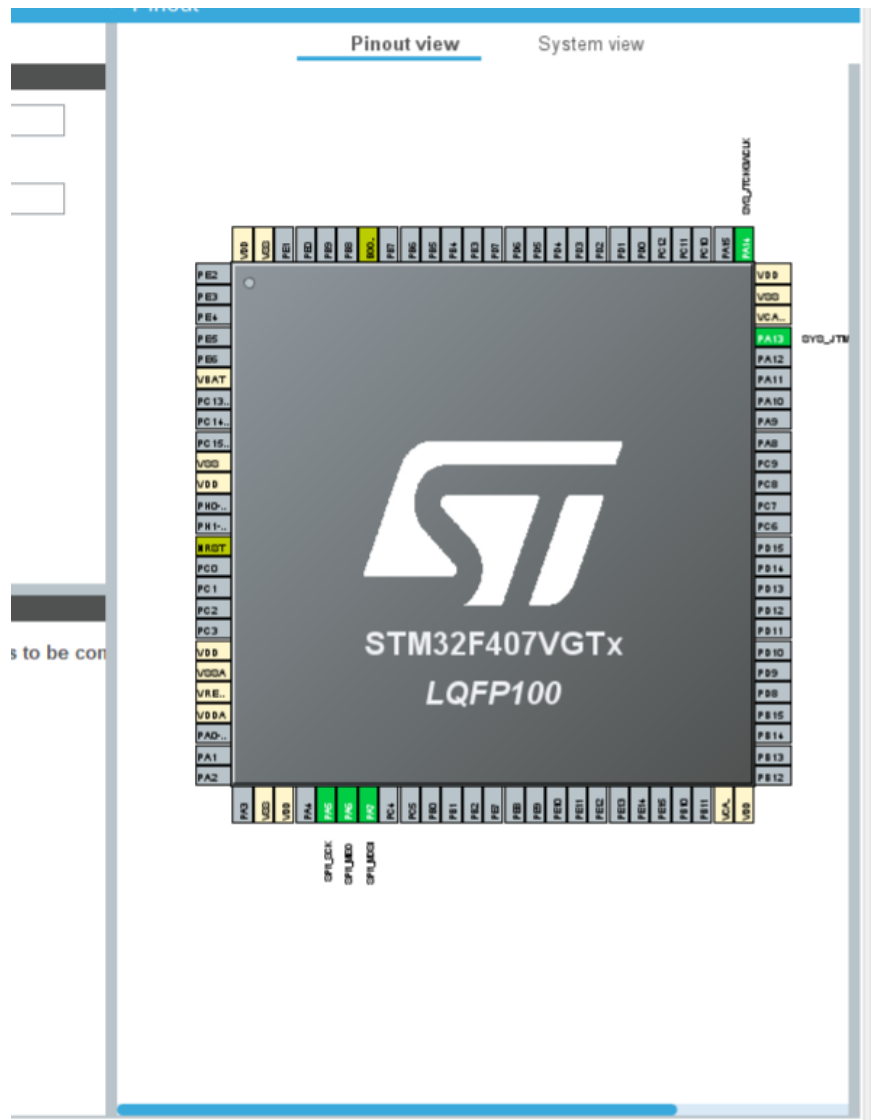
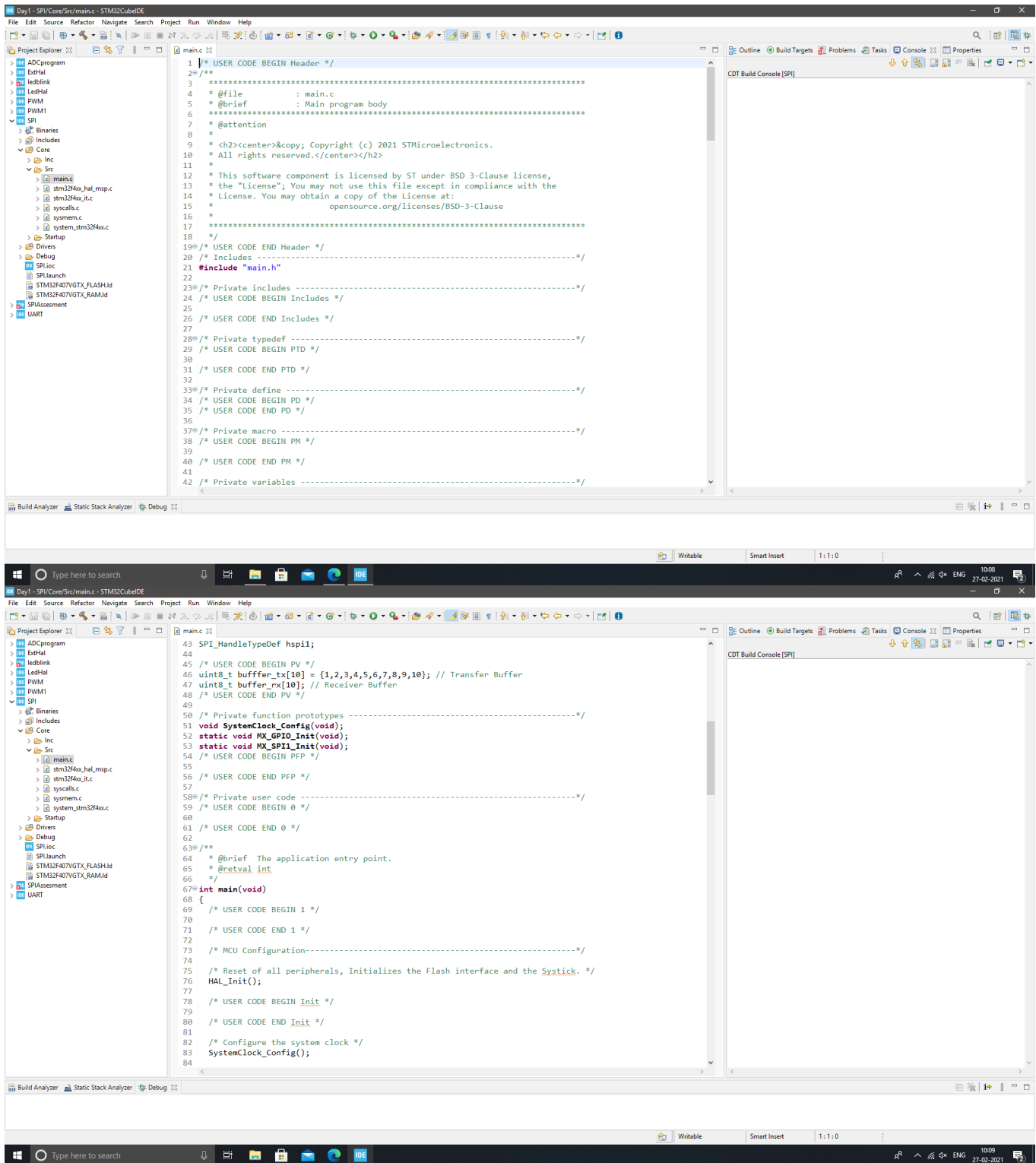


Fig: 2.4.1 SPI Pin configuration





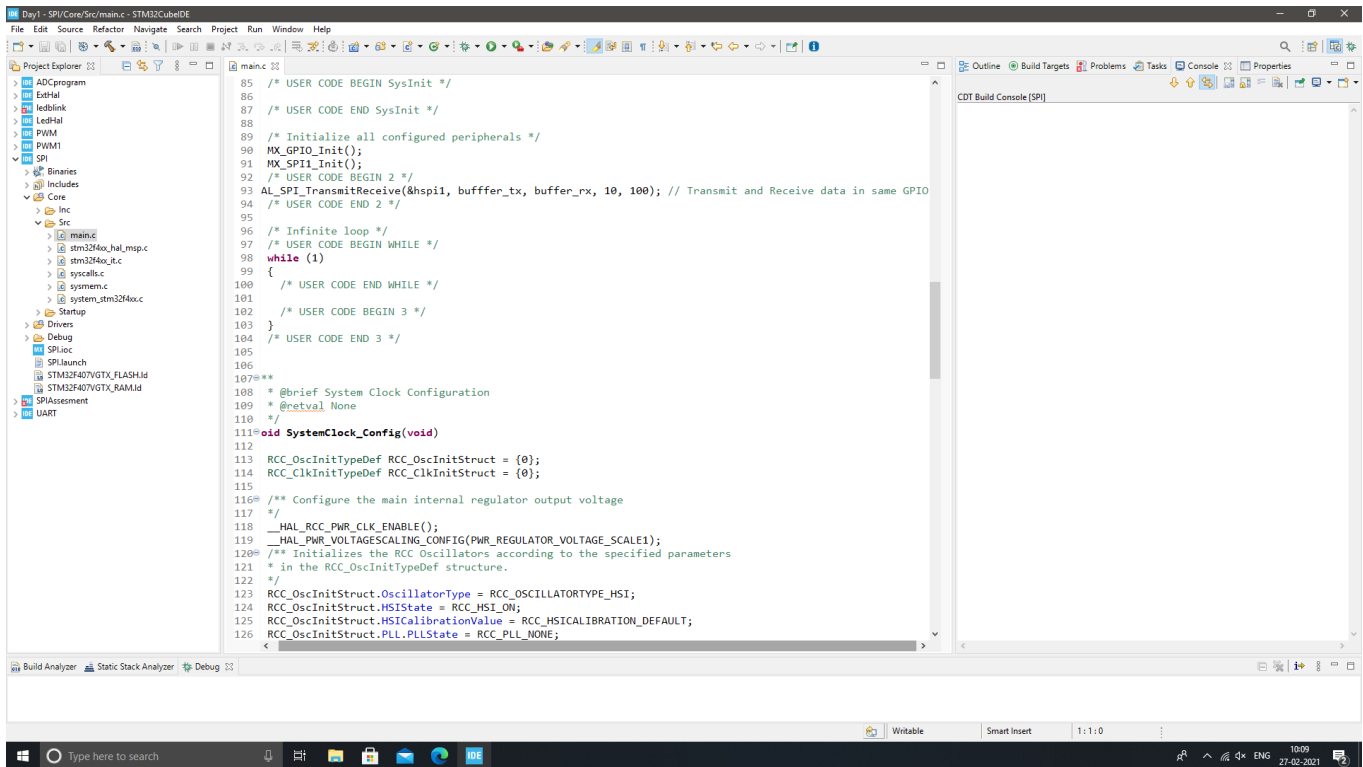
The image displays two screenshots of the STM32CubeIDE environment. The top screenshot shows the 'main.c' file with the following code:

```
1  /* USER CODE BEGIN Header */
2  /**
3   * @brief
4   * @file      : main.c
5   * @brief     : Main program body
6   *
7   * @attention
8   *
9   * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
10  * All rights reserved.</center></h2>
11  *
12  * This software component is licensed by ST under BSD 3-Clause license,
13  * the "License"; You may not use this file except in compliance with the
14  * License. You may obtain a copy of the License at:
15  *      opensource.org/licenses/BSD-3-Clause
16  *
17  */
18  /* USER CODE END Header */
19
20  /* Includes */
21  #include "main.h"
22
23  /* Private includes */
24  /* USER CODE BEGIN Includes */
25
26  /* USER CODE END Includes */
27
28  /* Private typedef */
29  /* USER CODE BEGIN PTD */
30
31  /* USER CODE END PTD */
32
33  /* Private define */
34  /* USER CODE BEGIN PD */
35  /* USER CODE END PD */
36
37  /* Private macro */
38  /* USER CODE BEGIN PM */
39
40  /* USER CODE END PM */
41
42  /* Private variables */
43  /* USER CODE BEGIN Private variables */
44  /* USER CODE END Private variables */
```

The bottom screenshot shows the 'main.c' file with the following code:

```
43 SPI_HandleTypeDef hspi1;
44
45 /* USER CODE BEGIN PV */
46 uint8_t _buffer_tx[10] = {1,2,3,4,5,6,7,8,9,10}; // Transfer Buffer
47 uint8_t _buffer_rx[10]; // Receiver Buffer
48 /* USER CODE END PV */
49
50 /* Private function prototypes */
51 void SystemClock_Config(void);
52 static void MX_GPIO_Init(void);
53 static void MX_SPI1_Init(void);
54 /* USER CODE BEGIN PFP */
55
56 /* USER CODE END PFP */
57
58 /* Private user code */
59 /* USER CODE BEGIN 0 */
60
61 /* USER CODE END 0 */
62
63 /**
64  * @brief The application entry point.
65  * @retval int
66  */
67 int main(void)
68 {
69  /* USER CODE BEGIN 1 */
70
71  /* USER CODE END 1 */
72
73  /* MCU Configuration */
74
75  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
76  HAL_Init();
77
78  /* USER CODE BEGIN Init */
79
80  /* USER CODE END Init */
81
82  /* Configure the system clock */
83  SystemClock_Config();
84
```

The CDT Build Console (SPI) is visible on the right side of both screenshots, showing the build process output.



The screenshot displays the STM32CubeIDE environment with the 'main.c' file open. The code is for an SPI configuration and includes a while loop for data transmission and reception. The Project Explorer on the left shows the project structure, including 'Core' and 'Src' folders. The CDT Build Console on the right is empty. The status bar at the bottom indicates the file is writable and the cursor is at line 1, column 10.

```
85 /* USER CODE BEGIN SysInit */
86
87 /* USER CODE END SysInit */
88
89 /* Initialize all configured peripherals */
90 MX_GPIO_Init();
91 MX_SPI1_Init();
92 /* USER CODE BEGIN 2 */
93 AL_SPI_TransmitReceive(&hspi1, buffer_tx, buffer_rx, 10, 100); // Transmit and Receive data in same GPIO
94 /* USER CODE END 2 */
95
96 /* Infinite loop */
97 /* USER CODE BEGIN WHILE */
98 while (1)
99 {
100 /* USER CODE END WHILE */
101
102 /* USER CODE BEGIN 3 */
103 }
104 /* USER CODE END 3 */
105
106
107 /**
108  * @brief System Clock Configuration
109  * @retval None
110  */
111 void SystemClock_Config(void)
112 {
113   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
114   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
115
116   /** Configure the main internal regulator output voltage
117   */
118   __HAL_RCC_PWR_CLK_ENABLE();
119   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
120   /** Initializes the RCC Oscillators according to the specified parameters
121   * in the RCC_OscInitTypeDef structure.
122   */
123   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
124   RCC_OscInitStruct.HSISource = RCC_HSI_ON;
125   RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
126   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
```

Fig: 2.4.2 SPI configuration code

## 2.5 UART

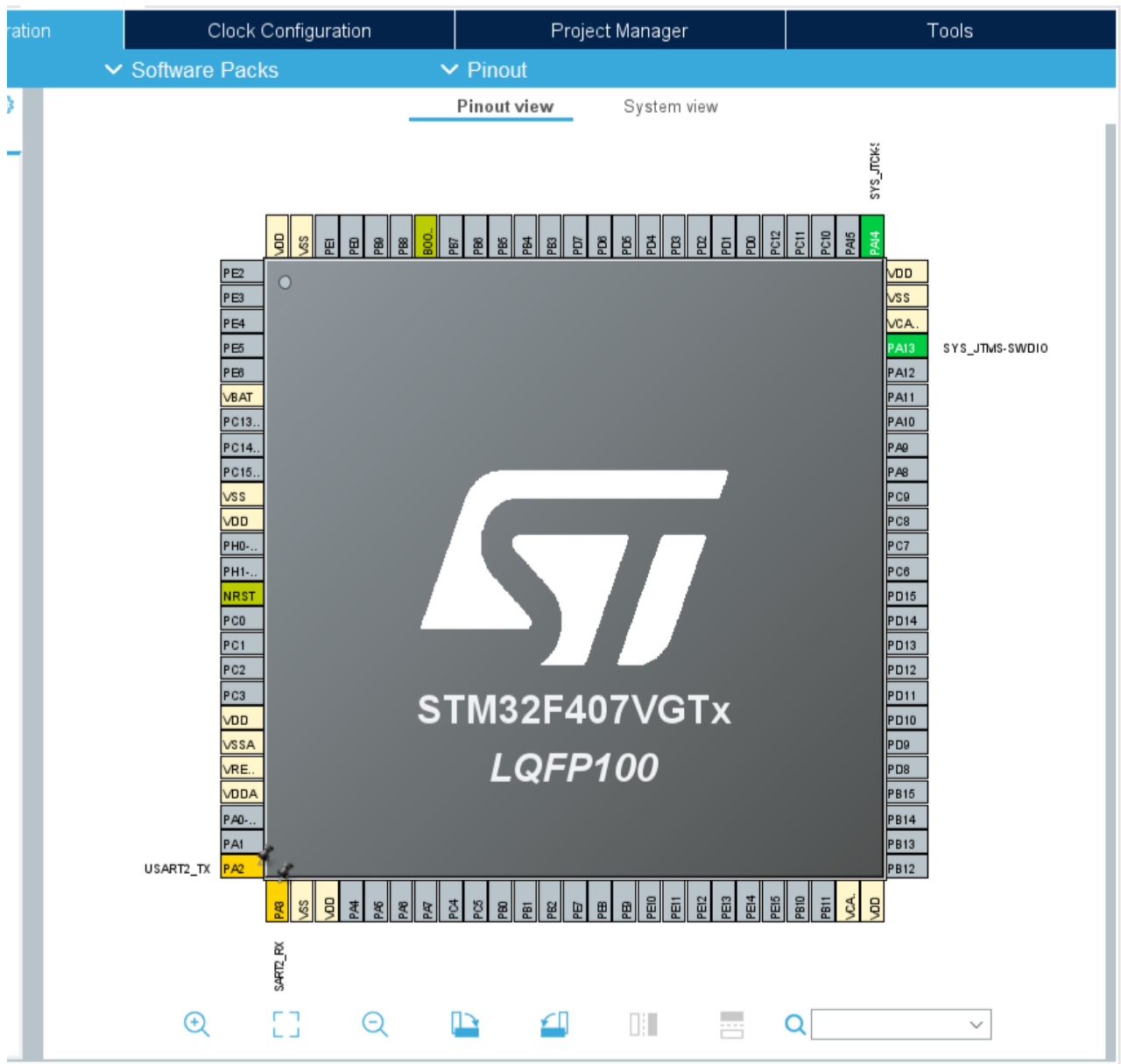
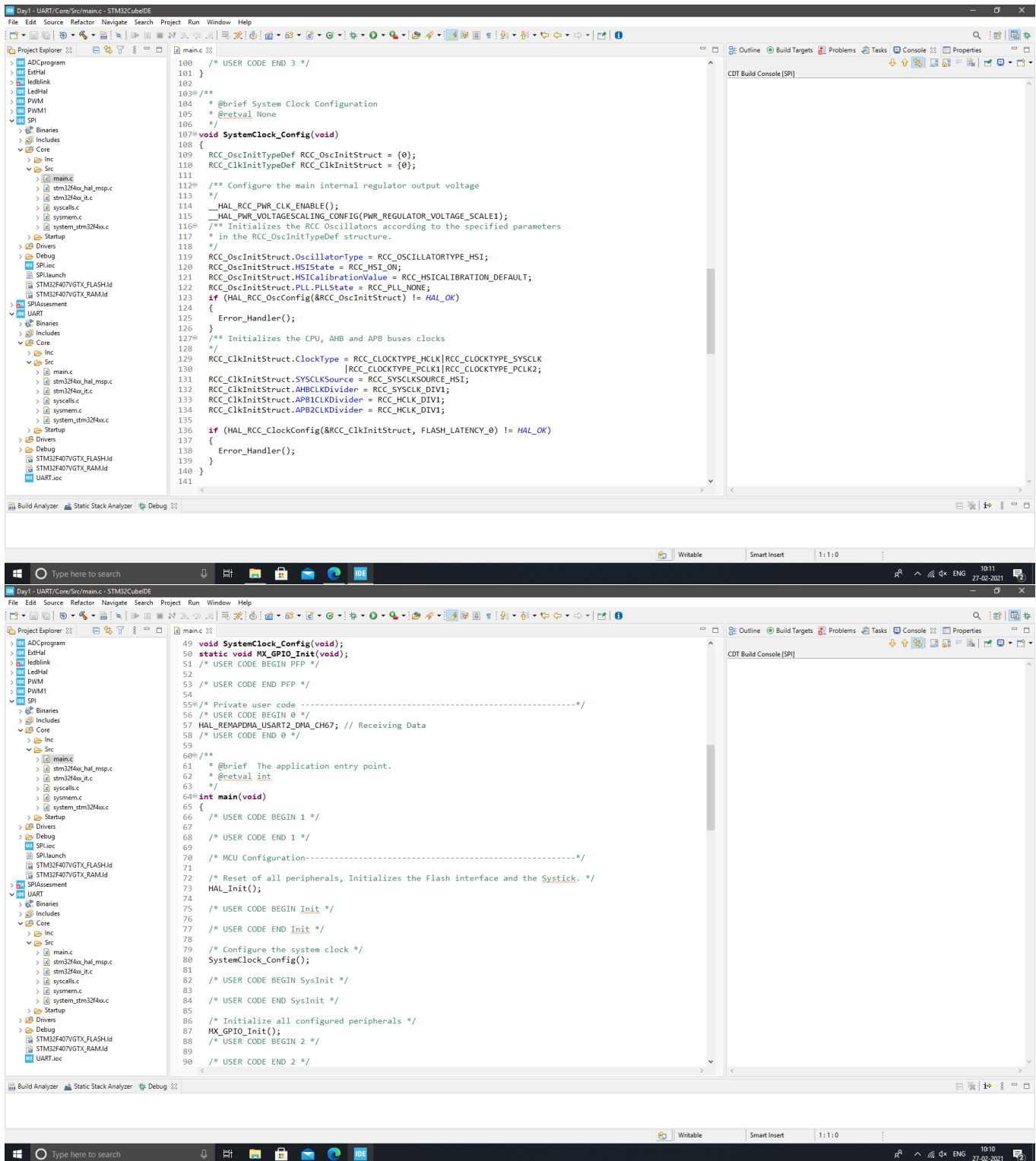


Fig: 2.5.1 UART Pin configuration



The image displays two screenshots of the STM32CubeIDE environment. The top screenshot shows the 'main.c' file with the 'void SystemClock\_Config(void)' function, which configures the RCC (Reset and Clock Control) for the STM32F407VGTx. It sets the oscillator to HSI, configures the PLL, and initializes the system clock. The bottom screenshot shows the 'main.c' file with the 'int main(void)' function, which calls 'HAL\_Init()', 'SystemClock\_Config()', and 'MX\_GPIO\_Init()' before entering an infinite loop.

```
100 /* USER CODE END 3 */
101 }
102
103 /**
104  * @brief System Clock Configuration
105  * @retval None
106  */
107 void SystemClock_Config(void)
108 {
109     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
110     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
111
112     /** Configure the main internal regulator output voltage
113     */
114     __HAL_RCC_PWR_CLK_ENABLE();
115     __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
116     /** Initializes the RCC Oscillators according to the specified parameters
117     * in the RCC_OscInitTypeDef structure.
118     */
119     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
120     RCC_OscInitStruct.HSICalibrationValue = RCC_HSI_CALIBRATION_DEFAULT;
121     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
122     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
123     {
124         Error_Handler();
125     }
126
127     /** Initializes the CPU, AHB and APB buses clocks
128     */
129     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
130         |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
131     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
132     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
133     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
134     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
135
136     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
137     {
138         Error_Handler();
139     }
140 }
141
```

```
49 void SystemClock_Config(void);
50 static void MX_GPIO_Init(void);
51 /* USER CODE BEGIN PFP */
52
53 /* USER CODE END PFP */
54
55 /* Private user code -----*/
56 /* USER CODE BEGIN 0 */
57 HAL_REMAPDMA_USART2_DMA_CH67; // Receiving Data
58 /* USER CODE END 0 */
59
60 /**
61  * @brief The application entry point.
62  * @retval int
63  */
64 int main(void)
65 {
66     /* USER CODE BEGIN 1 */
67
68     /* USER CODE END 1 */
69
70     /* MCU Configuration-----*/
71
72     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
73     HAL_Init();
74
75     /* USER CODE BEGIN Init */
76
77     /* USER CODE END Init */
78
79     /* Configure the system clock */
80     SystemClock_Config();
81
82     /* USER CODE BEGIN SysInit */
83
84     /* USER CODE END SysInit */
85
86     /* Initialize all configured peripherals */
87     MX_GPIO_Init();
88     /* USER CODE BEGIN 2 */
89
90     /* USER CODE END 2 */

```

Fig: 2.5.2 UART configuration code

### Activity 3 – PROJECT ON BCM MODULE

BCM module was implemented using STM32f407VG microcontroller featuring 32 bit ARM-cortex - M4 with FPU core.

This BCM module have following features:

1. Alarm system
2. Seat control
3. Power mirror
4. Automatic wiper system

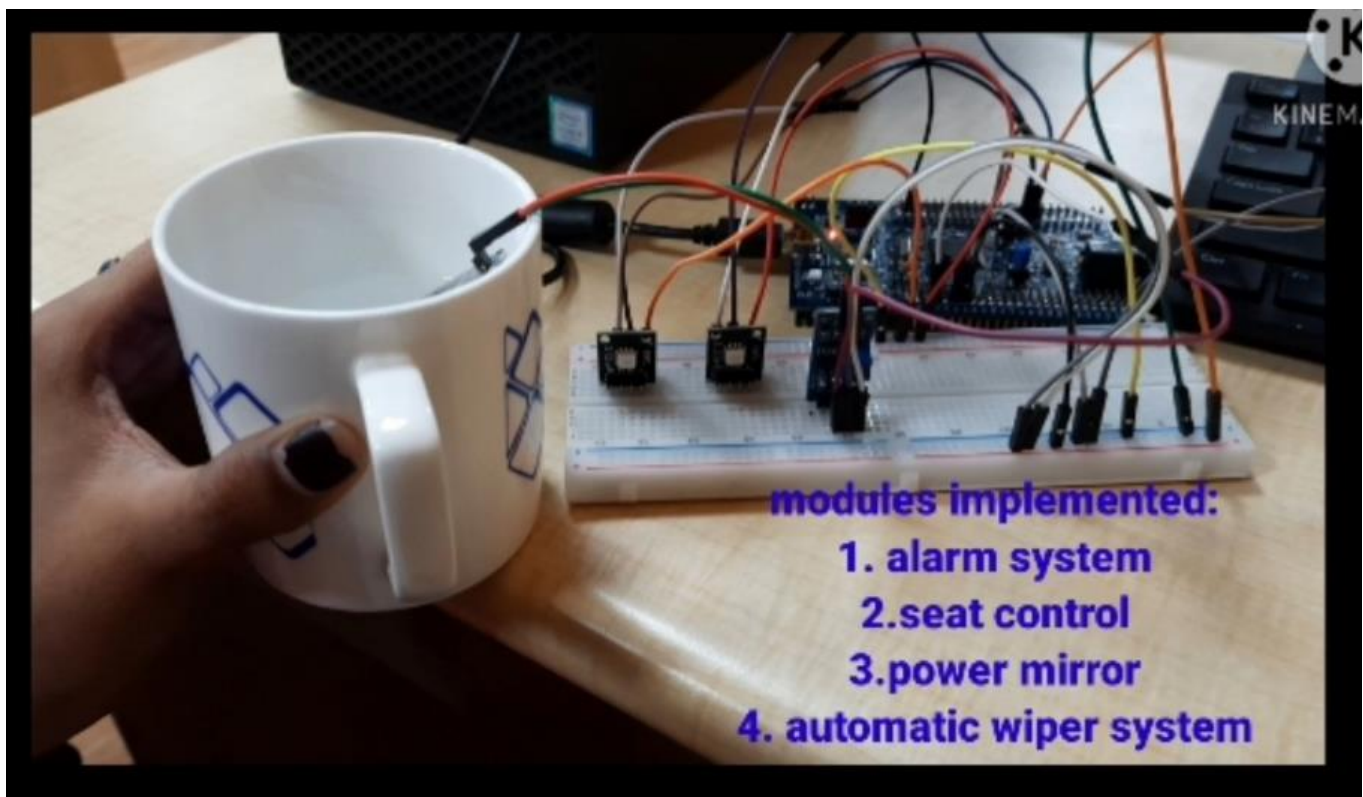


Fig 3.1 Features of BCM module

Below is the pinout and configuration of STM microcontroller used:

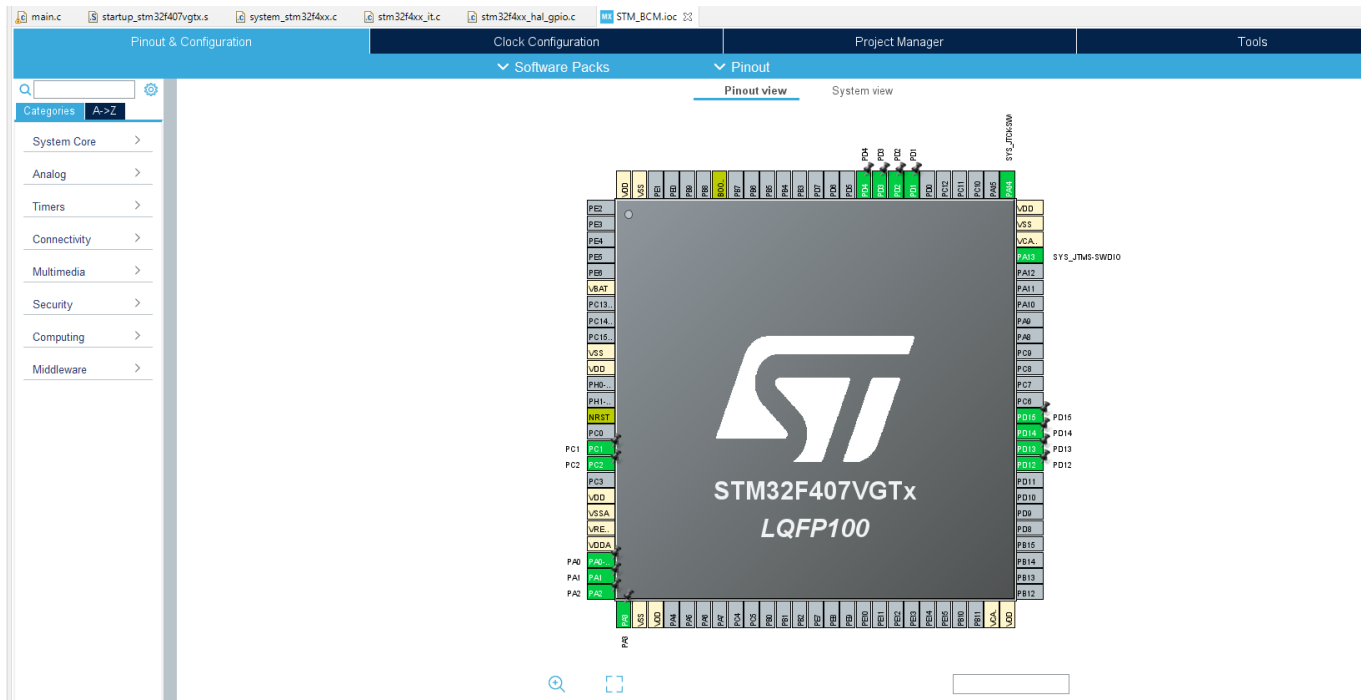


Fig 3.2 Pin configuration

### 1. Alarm system:

PA1: Denotes power status of car (0: OFF, 1: ON)

PA2: Denotes the door status (0: OFF, 1: ON)

PA3: Denotes the door status (0: door closed, 1: door open)

Alarm system uses two variables PA1 and PA2 and Dashboard light as output.

Door safety system checks whether door is locked or unlocked.

```

/-----
if(HAL_GPIO_ReadPin(PA1_GPIO_Port, PA1_Pin) == 1)
{

    if(HAL_GPIO_ReadPin(PA2_GPIO_Port, PA2_Pin)==0 && (HAL_GPIO_ReadPin(PA3_GPIO_Port, PA3_Pin)==0)){

        HAL_GPIO_TogglePin(PD1_GPIO_Port, PD1_Pin);
        HAL_Delay(900);

    }else if((HAL_GPIO_ReadPin(PA2_GPIO_Port, PA2_Pin)==1) && (HAL_GPIO_ReadPin(PA3_GPIO_Port, PA3_Pin)==0)){

        HAL_GPIO_TogglePin(PD2_GPIO_Port, PD2_Pin);
        HAL_GPIO_TogglePin(PD1_GPIO_Port, PD1_Pin);
        HAL_Delay(100);

    }else if(HAL_GPIO_ReadPin(PA2_GPIO_Port, PA2_Pin)==1 && HAL_GPIO_ReadPin(PA3_GPIO_Port, PA3_Pin)==1) {

        HAL_GPIO_TogglePin(PD2_GPIO_Port, PD2_Pin);
        HAL_Delay(100);

    }

}else{
    //pass comment
}

```

Fig 3.3 Alarm system-code

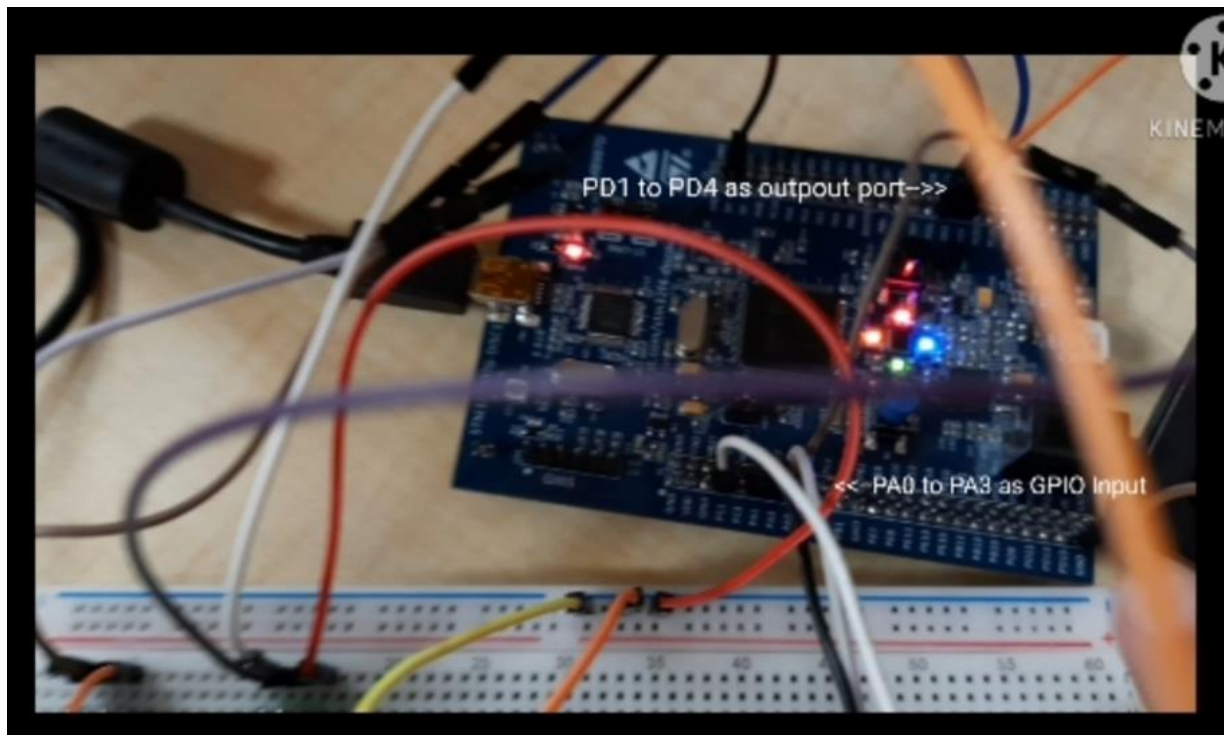
**Implementation:**

Fig 3.4 Alarm system - Implementation



## 2. Seat control:

Adjustment of seat with the help of buttons in two direction

Forward direction: when the input is 1

Reverse direction: when the input is 0

When the input is 1 LED glows in clockwise direction

When the input is 0 LED glows in anti-clockwise direction

```
/*  
*****  
  
if(HAL_GPIO_ReadPin(PA0_GPIO_Port, PA0_Pin)==1){  
  
    HAL_GPIO_TogglePin(PD12_GPIO_Port, PD12_Pin);  
    HAL_Delay(100);  
    HAL_GPIO_TogglePin(PD13_GPIO_Port, PD13_Pin);  
    HAL_Delay(100);  
    HAL_GPIO_TogglePin(PD14_GPIO_Port, PD14_Pin);  
    HAL_Delay(100);  
    HAL_GPIO_TogglePin(PD15_GPIO_Port, PD15_Pin);  
    HAL_Delay(100);  
  
}else{  
    //pass comment  
}
```

Fig 3.5 Seat control system- code

### Implementation:

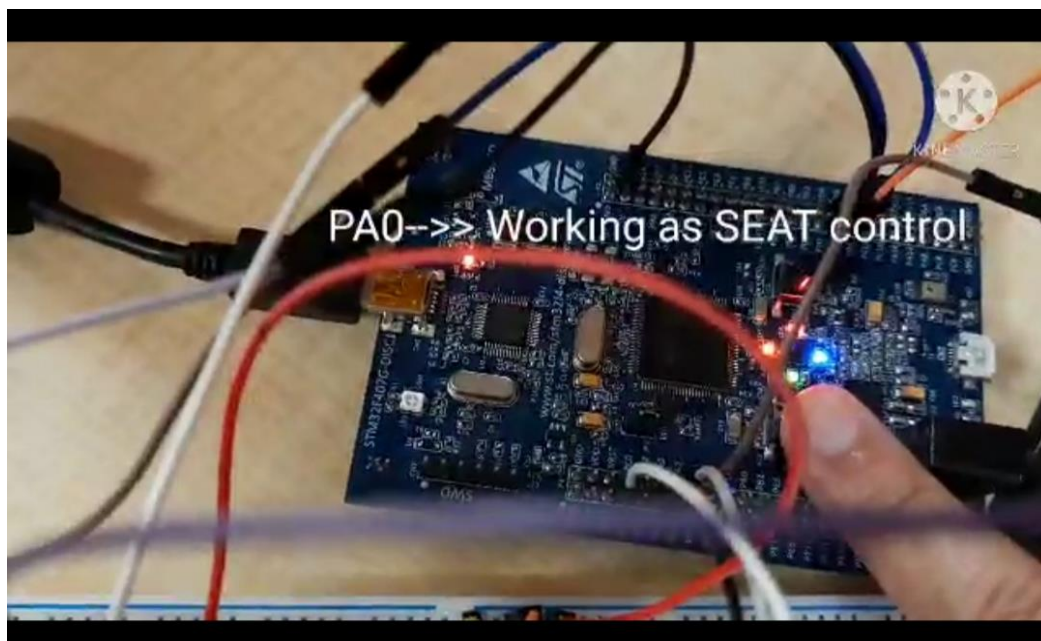




Fig 3.6 Seat control system- Implementation

### 3. Power mirror:

Two of the LEDs glow as a sign of mirrors unfolding when provided with power supply  
PD12 and PD13 are the output pins  
PA0 and PA3 are the input pins.

```
/*  
*****  
*/  
  
if(HAL_GPIO_ReadPin(PA0_GPIO_Port, PA0_Pin)==1 && HAL_GPIO_ReadPin(PA3_GPIO_Port, PA3_Pin)==1){  
    HAL_GPIO_TogglePin(PD12_GPIO_Port, PD12_Pin);  
    HAL_Delay(100);  
    HAL_GPIO_TogglePin(PD13_GPIO_Port, PD13_Pin);  
    HAL_Delay(100);  
}  
  
//while '}'  
}
```

Fig 4.6 Power mirror- code

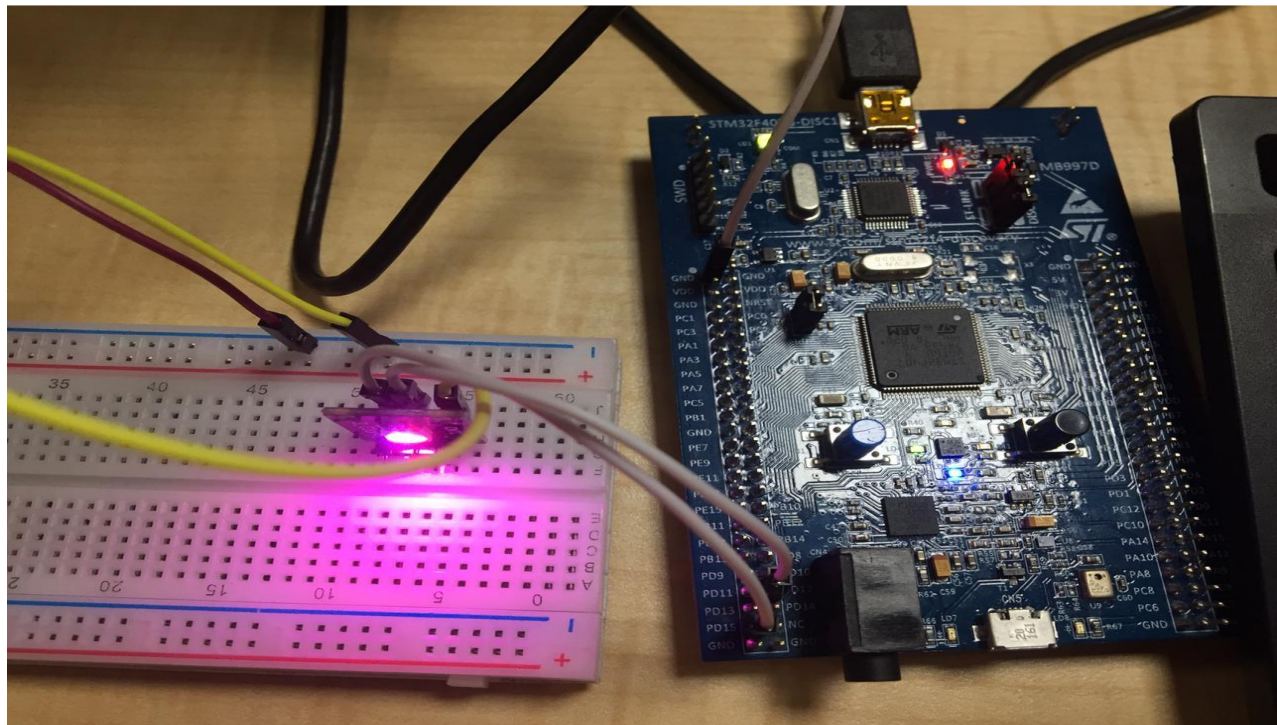


Fig 3.7 Power mirror- Implementation

#### 4. Automatic wiper system:

Whenever the humidity sensor senses the presence of moisture over the windshield it sends the signal to turn ON the wipers.

The input was taken from humidity sensor at PC1

The output was given to the external led at pin PD3

The input was taken from humidity sensor at PC1 pin and output is given to the external led at pin PD3.

\* Whenever the humidity sensor senses presence of moisture over the wind shield it sends the signal to turn ON the wipers.

```

/*****
if(HAL_GPIO_ReadPin(PC1_GPIO_Port, PC1_Pin)==1){

    HAL_GPIO_TogglePin(PD3_GPIO_Port, PD3_Pin);
    HAL_Delay(200);

}else{
//pass comment
}

```

Fig 3.8 Automatic wiper system- code

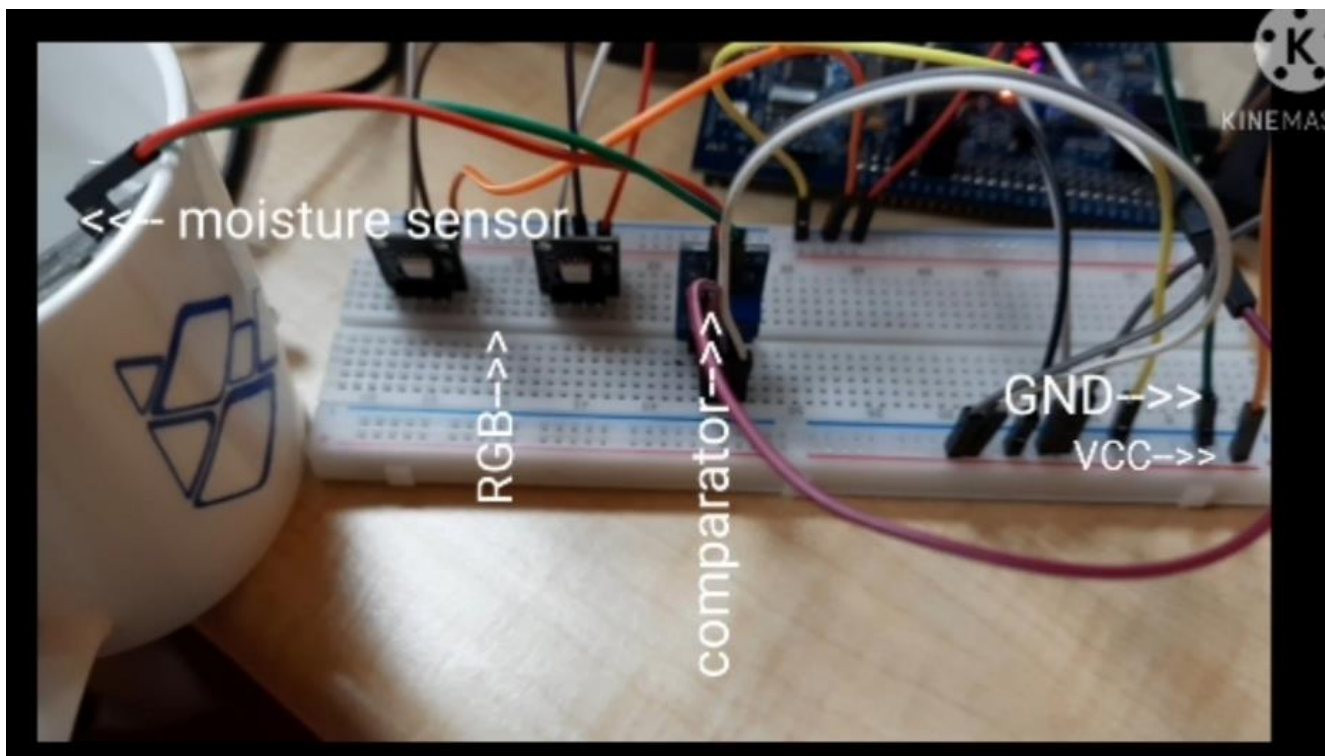


Fig 3.9 Automatic wiper system- Implementation

**Final integrated project:**

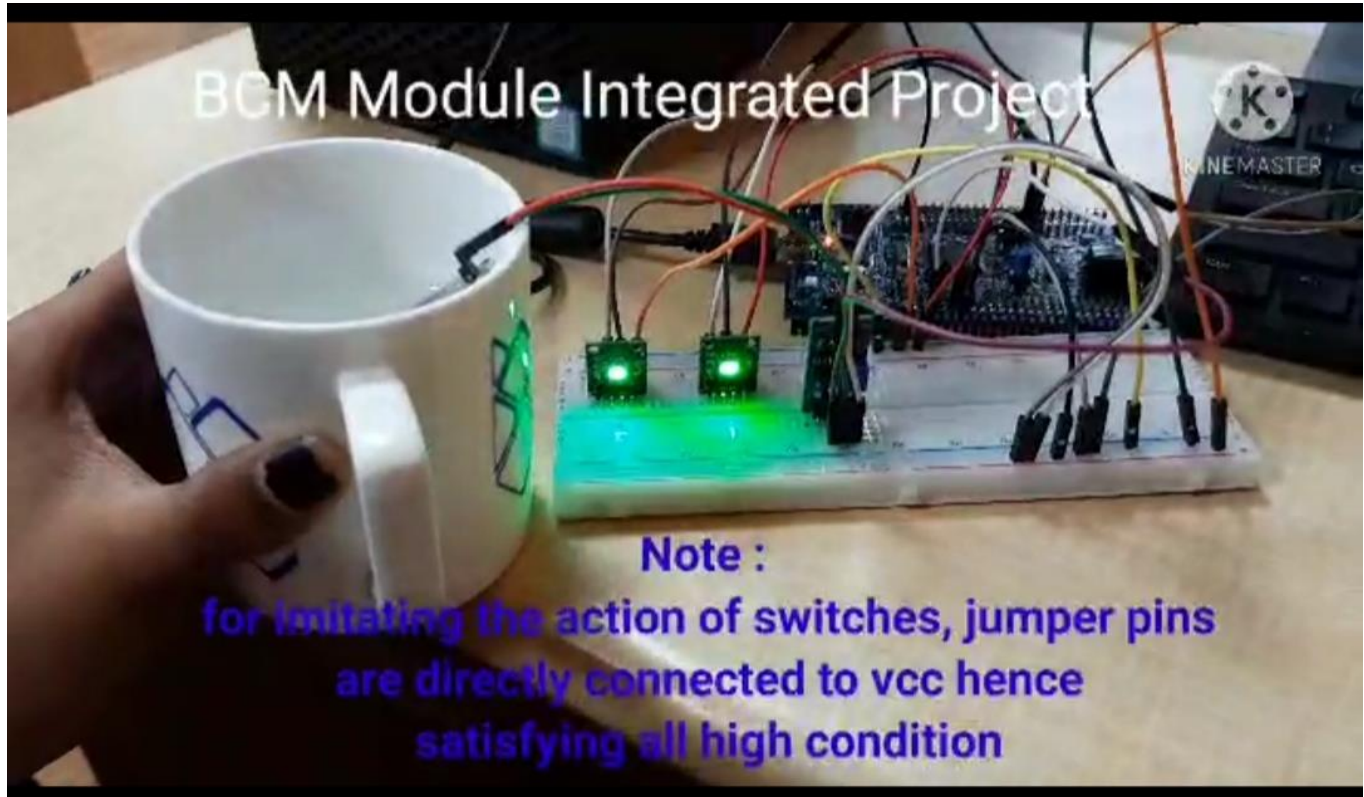


Fig 3.10: Final Integrated BCM