YILDIZ TECHNICAL UNIVERSITY

DEPARTMENT OF BIOMEDICAL ENGINEERING

BME3321 INTRODUCTION TO MICROCONTROLLER PROGRAMMING LABORATORY

EXPERIMENT SHEETS

Experiment 1: C Programming Language Basics: Data Types, Variables, Arrays, Loops, Conditionals, Functions , Pointers , Structures

Objectives

The objectives of Experiment 1 are

- to learn C programming language basics Data types, Variables, Arrays, Loops, Conditionals, Functions , Pointers , Structures

Apparatus Required:

- Dev C++

Preliminary Work:

- Install required Dev C++ programme (https://sourceforge.net/projects/orwelldevcpp/)

Figure 1

- Study the L03 notes.
- Write the following codes in the experimental work in Dev C++.

Experimental Work:

1. Logical Operator

Write the code that will calculate the truth table of the p'Vq proposition using OR operator and display it on the screen. Don't forget to adjust the spaces to make the output look neat.

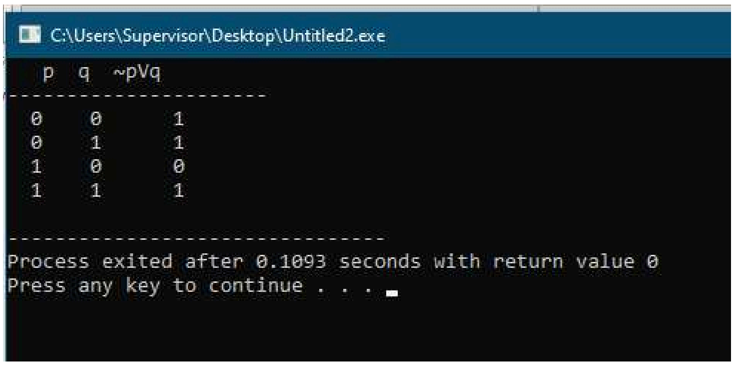


Figure 2: Program Output Window

Answer:

```
#include <stdio.h>

int main (void)

{

printf("  p   q    ~pVq\n");

printf("----------------------\n");

printf("%3d%5d%7d \n", false, false, !false||false);

printf("%3d%5d%7d \n", false, true, !false||true);

printf("%3d%5d%7d \n", true, false, !true||false);

printf("%3d%5d%7d \n", true, true, !true||true);

return (0);

}
```

2. Loops (for-while) & Conditionals

a. Write the code that produces the following output by using nested loops.

```
1
2  1
3 2 1
4 3 2 1
5 4 3 2 1
6 5 4 3 2 1
7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
```

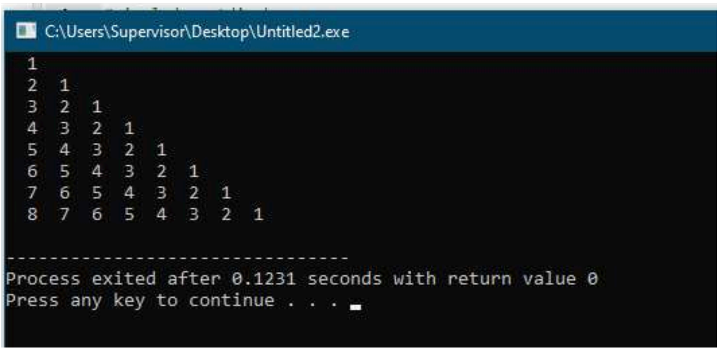


Figure 4: Program Output Window

Answer:

```
# include <stdio.h>
int main (void)

{
        int i,j;
        for (i=1; i<=8; i++)
        {
                for (j=i; j>=1; j--)
                {
                  printf("%3d", j);
                 }
              printf("\n");
        }
        return(0);
}
```

b. Write the C program that receives the n value from the user, which is a positive integer value, as input and find all prime numbers up to n and display them on the screen.

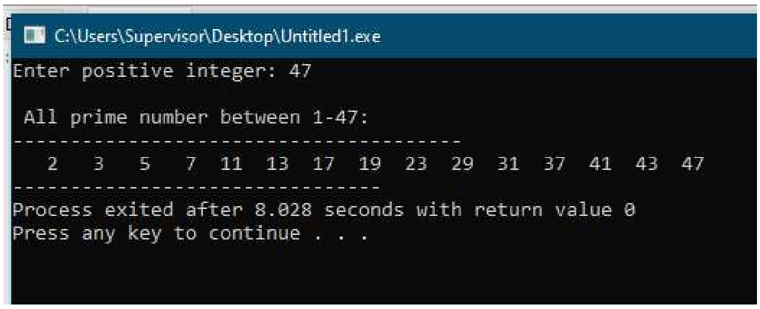


Figure 3: Program Output Window

Answer:

```
#include <stdio.h>

int main (void)

{

    int n;

    int num, p,i;

    printf("Enter positive integer: ");

    scanf("%d", &n); /*Taking n value*/

    printf("\n All prime number between 1-%d: ",n);

    printf("\n---------------------------------------\n");

    for( num = 2; num <= n; num = num+1)

    {/*Let's assume that the number is prime: p=1 means the number is prime, p=0 means it is not prime*/

    p=1; /*Assume the number is prime*/

    i=2; /*The variable i is used to control which numbers the number entered by the user can be divided by i*/

    while ((i<num) && p==1)

    {

            if (num%i == 0)

            p=0; /*The number is not prime because it is divisible.*/

            i=i+1;

    }

    if (p==1)

    printf("%4d" , num);      }

    return (0);

}
```

3. Functions

Write a code that takes a positive integer value from the user and shows the number of digits of the integer on the screen. While writing the code, use the function named ***'basamak_bul ()'***. *basamak_bul ()* function must receive an integer value from where it was called, find the number of digits of the integer and return it to where it was called.

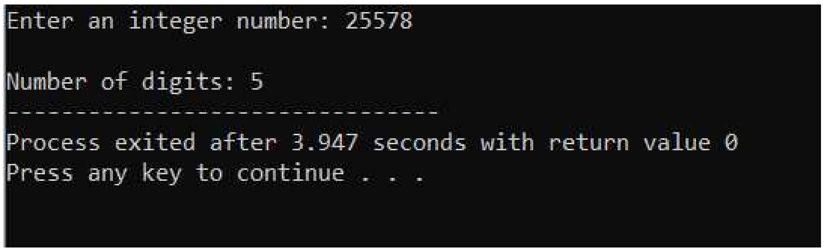


Figure 5: Program Output Window

Answer:

```
#include <stdio.h>

int basamak_bul(int x);

int main (void)

{

        int a,t;

        printf("Enter an integer number: ");

        scanf("%d",&a);

        t=basamak_bul(a);

        printf("\nNumber of digits: %d",t);

        return(0);

}

int basamak_bul(int x)

{

        int digit = 0;

        while (x)

        {

        digit = digit+1;

        x=x/10;

        }

        return(digit); }
```

4. Pointers

a. Write the C program that orders 3 integers received from the user and displays the first input values and sequential values on the screen. Return the sequential numbers as reference parameters. (Use the **pointers** for this.) Use the **'replace ()'** function to sort the 3 numbers.

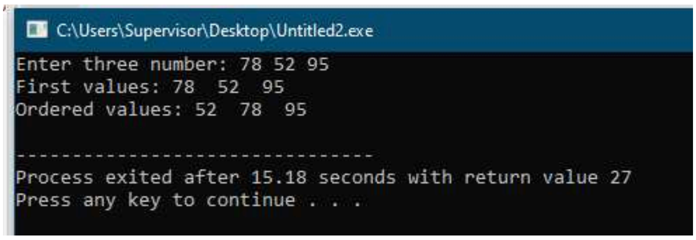


Figure 6: Program Output Window

Answer:

```
#include <stdio.h>
/* Program that sorts the 3 entered numbers*/

void replace(int*,int*);

int main (void)
{
        int x,y,z;
        printf("Enter three numbers: ");
        scanf("%d%d%d", &x,&y,&z);
        printf("First values: %d  %d  %d\n", x,y,z);
        if (x>y) replace(&x,&y);
        if (x>z) replace(&x,&z);
        if (y>z) replace(&y,&z);
        printf("Ordered values: %d  %d  %d\n", x,y,z);
}

/*replace() function that changes the values of two parameters*/

void replace(int *a, int *b)
/*a and b are taken as reference parameters. Therefore, changes in this function will be
reflected in the sent parameter.*/
{
        int temporary;
        temporary = *a;
        *a = *b;
        *b = temporary;
}
```

b. Write a function that finds the perimeter and area of a rectangle. In the function, receive the width and length of the rectangle as the value parameter and return the perimeter and area as the reference parameter.

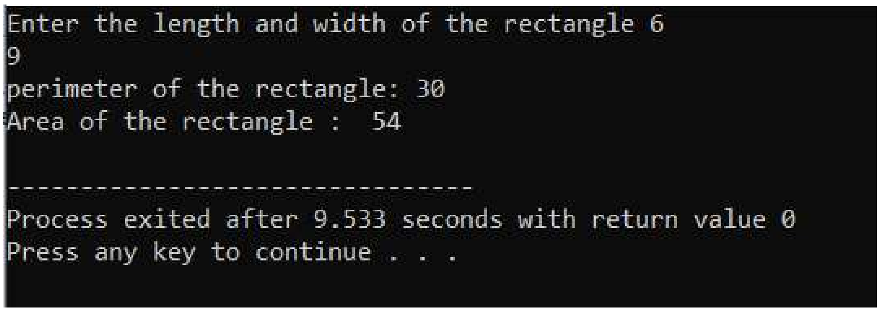


Figure 7: Program Output Window

Answer:

```
/* Program to find the perimeter and area of a rectangle. While the width and height of the rectangle are taken as value parameters, the perimeter and area are returned as reference parameters */

#include<stdio.h>

void perimeter_area(int, int , int*, int*);

int main (void)

{

    int width,length,perimeter,area;

    printf("Enter the length and width of the rectangle ");

    scanf("%d%d",&length,&width);

    if (length<0 || width<0)/*input control*/

    printf("\nYou entered the wrong value");

    else

    {

            perimeter_area(width, length , &perimeter, &area);

            printf("perimeter of the rectangle: %d\n", perimeter);

            printf("Area of the rectangle :  %d\n", area);

    }

}

void perimeter_area(int w, int l, int *e, int *a)

{

    *e = 2* (w+l); /*Calculate perimeter*/

    *a = w*l; /*Calculate area*/ }
```

5. Arrays

Write a C program in which the user enters integers in a 5-element array and after each integer value entered in the array, it shows whether the entered number is odd or even. This program should consist of two functions. The **'bul ()'** function must receive an integer value from where it is called and show it is odd or even. The **'main ()'** function should receive 5 integer values from the user, store them in an array and show that the array elements are odd or even using the **bul()** function.

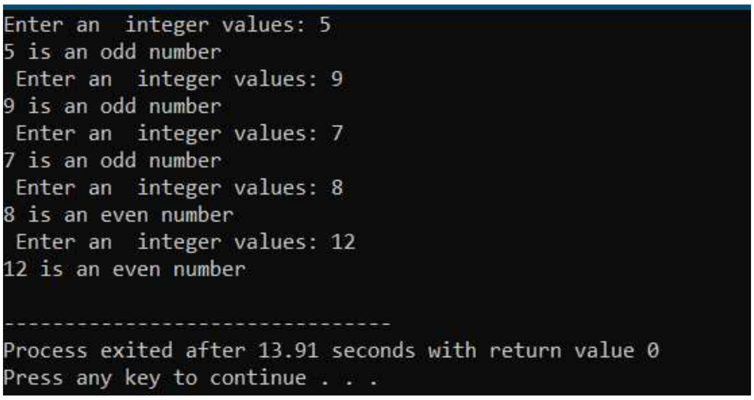


Figure 8: Program Output Window

Answer:

```
#include<stdio.h>

void bul (int);

int main (void)

{

    int k[5],i;

    for (i = 0; i<=4 ; i++)

    {

            printf("Enter an  integer values: ");

             scanf("%d",&k[i]);

             bul(k[i]);

    }

    return(0);

}
```

```
void bul (int a)
{
    if (a%2 == 0)
      printf("%d is an even number\n ", a);
       else
         printf("%d is an odd number\n ", a);
}
```

6. Structures

Write a program that receives the coordinates of two points (x1,y1) and (x2,y2) as input from the user and calculates the distance between them. The x and y coordinates of each point in your program should be kept in a struct. The distance formula is as follows:

distance= $\sqrt{(x1 - x2)^2 + (y1 - y2)}^2$

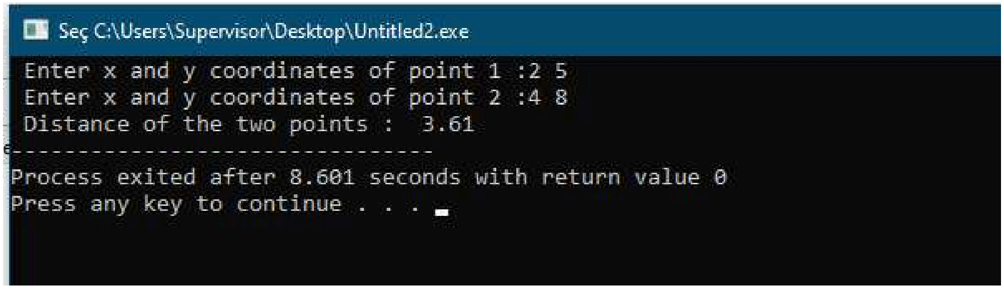


Figure 9: Program Output Window

Answer:

```
#include <stdio.h>

#include <math.h>

int main (void)

{

struct point

{

int x,y;

};

struct point p1, p2;

float distance;

/*Read the coordinates of two points*/

printf(" Enter x and y coordinates of point 1 :");

scanf ("%d %d", &p1.x, &p1.y);

printf(" Enter x and y coordinates of point 2 :");

scanf ("%d %d", &p2.x, &p2.y);

/* Calculate the distance between two points*/

distance = sqrt (((p1.x-p2.x)*(p1.x-p2.x))+((p1.y-p2.y)*(p1.y-p2.y)));

printf(" Distance of the two points : %5.2f", distance);

return (0); }
```

EXPERIMENT 2: GENERAL-PURPOSE INPUT/OUTPUT (GPIO)

Objectives

The objectives of Experiment 2 are

- to learn tools/environment for STM32F4 microcontroller programme and architecture
- to use Driving GPIO functions (HAL_GPIO_WritePin, HAL_GPIO_ReadPin, HAL_GPIO_TogglePin) and GPIO Output Data Register (ODR), GPIO Input Data Register (IDR)

Apparatus Required:

- STM32CubeMx
- Keil µVision (MDK ARM)
- STM32 ST-Link Utility
- STM32F4 Microcontroller
- STM32F4 Reference Manual
- STM32F4 User Manual

Preliminary Work:

1. Install required programmes (STM32CubeMx, Keil µVision (MDK ARM), STM32 ST-Link Utility)
   STM32CubeMx-----> https://www.st.com/en/development-tools/stm32cubemx.html
   ST-Link Utility-----> https://www.st.com/en/development-tools/stsw-link004.html
   Keil µVision-------->https://www.keil.com/download/product/

Figure 1

2. Study the GPIO (Lecture 4) notes. Write the codes of the experimental work in Keil µVision at home.

Experimental Work:

1. First, open the CubeMx program. Select "Manage embedded software packages" from the 'Help' menu (Figure 2).

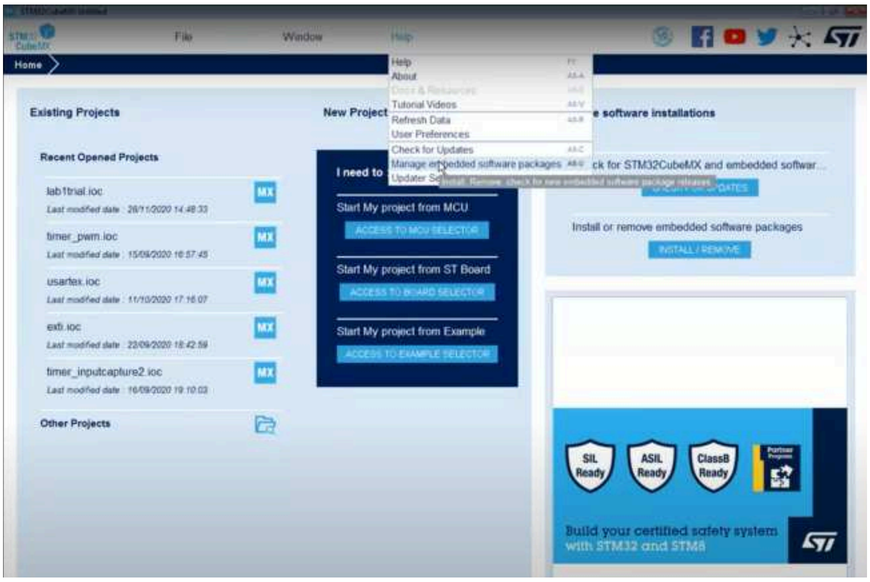


Figure 2

2. According to the microcontroller you have, the relevant package (STM32F4) is selected and installed (Figure 3).

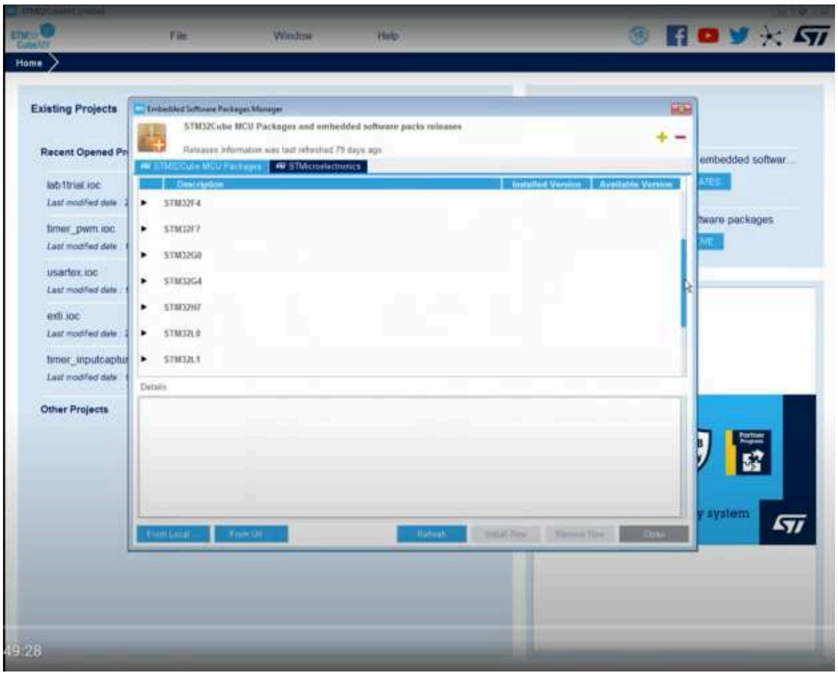


Figure 3

3. To prevent it from updating in ordinary, settings are made as seen in Figure 4 and Figure 5.

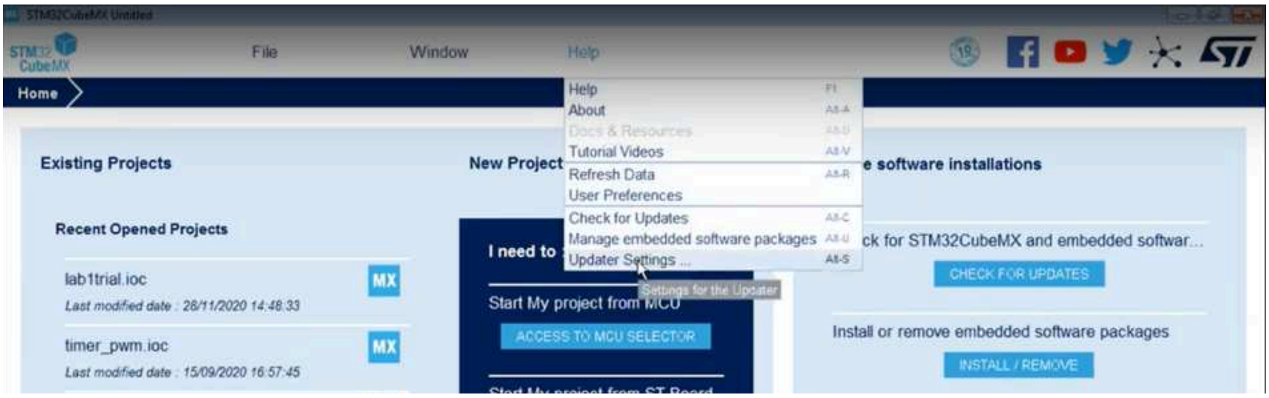


Figure 4

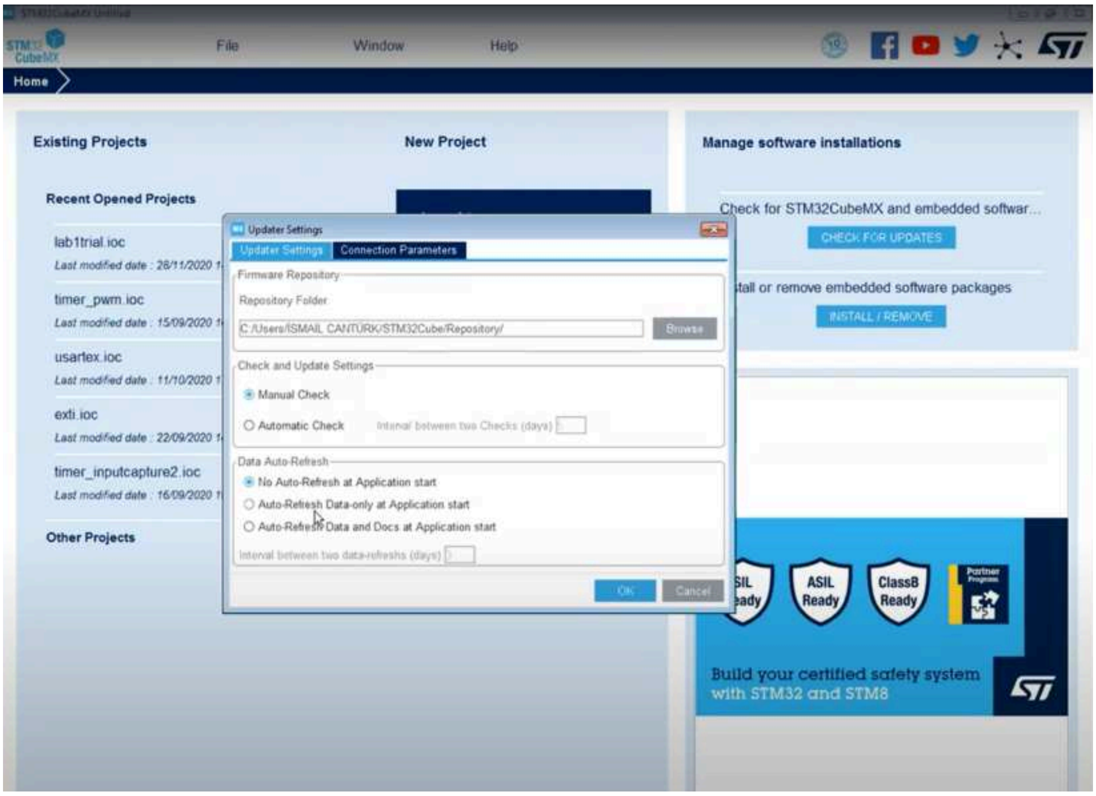


Figure 5

4. Now you can create a new project file to start programming. Select “New Project” from the File menu (Figure 6). The relevant microcontroller is found and selected from the Part Number as in Figure 7. After choosing our STM32F4 discovery card, choose “Start Project” from the top menu (Figure 7).

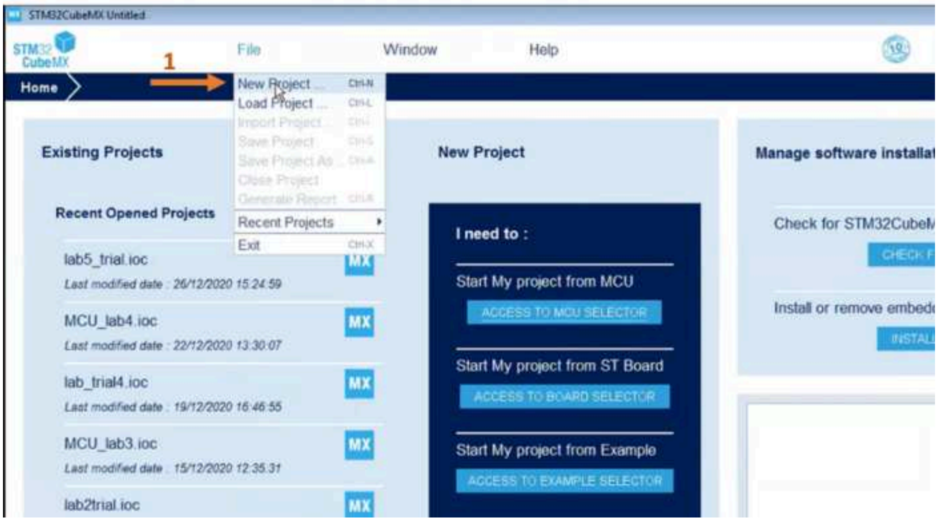


Figure 6

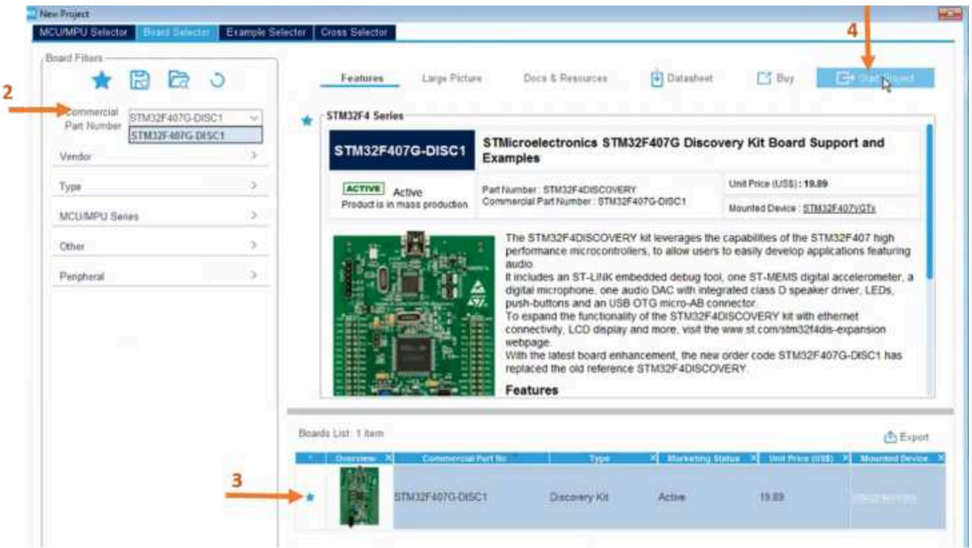


Figure 7

5. You will see a screen like in Figure 8. You can see the schematic of the microcontroller you will use here. You can use it this way. However, in terms of power consumption and more stable operation, it will be better if you turn off the pins that you will not use in the lab. The green ones show the open pins and the gray ones show the reset state ones. In this lab, you will basically use push pull buttons and leds. So you can turn off the others. Which of these pins are can be understood by looking at the user manual. You should not close the pins that provide the connection between the microcontroller and the programmer. You can check which of these pins are in the user manual (Figure 9). You can close and define tasks of a pin by left-clicking on the pin. You completed the Pinout Configuration.

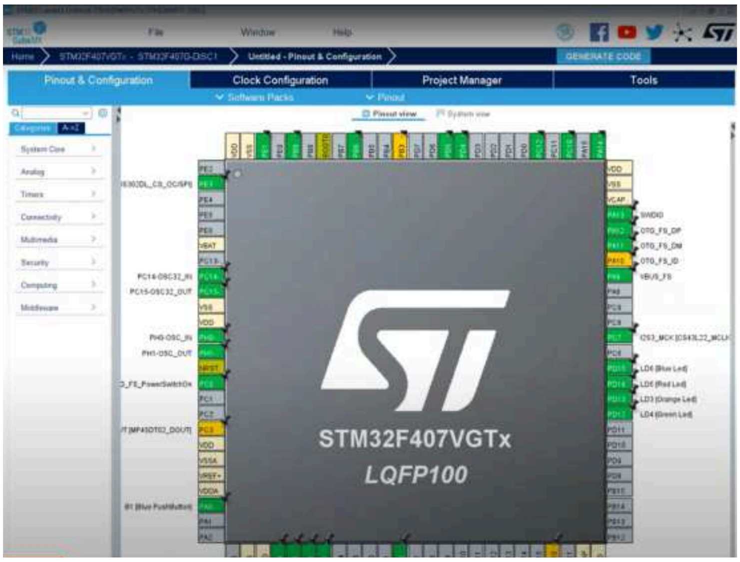
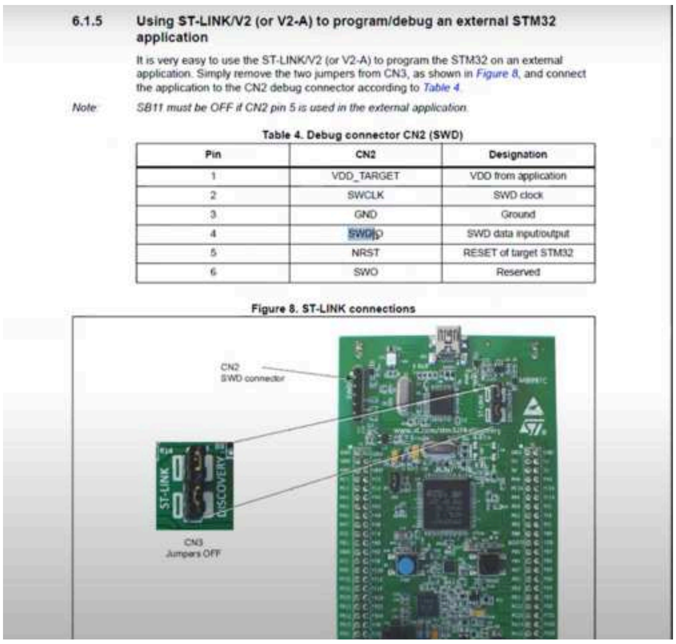


Figure 8



6.1.5 **Using ST-LINK/V2 (or V2-A) to program/debug an external STM32 application**

It is very easy to use the ST-LINK/V2 (or V2-A) to program the STM32 on an external application. Simply remove the two jumpers from CN3, as shown in Figure 8, and connect the application to the CN2 debug connector according to Table 4.

Note: SB11 must be OFF if CN2 pin 5 is used in the external application.

Table 4. Debug connector CN2 (SWD)

Pin	CN2	Designation
1	VDD_TARGET	VDD from application
2	SWCLK	SWD clock
3	GND	Ground
4	SWDIO	SWD data input/output
5	NRST	RESET of target STM32
6	SWO	Reserved

Figure 8. ST-LINK connections

Figure 9

6. There is nothing we can change in the Clock Configuration section for this experiment. Come to the Project Manager menu and set the fields as in Figure 10 (Give the project name using English characters without spaces. You can create the project in a file on the desktop. Select the MDK-ARM option for Toolchain/IDE. Uncheck 'Use latest available version' to prevent it from updating). Then select the Generate Code and so the template file is out. Select “Open Project” from the opened screen (Figure 11). Keil µVision will be opened.

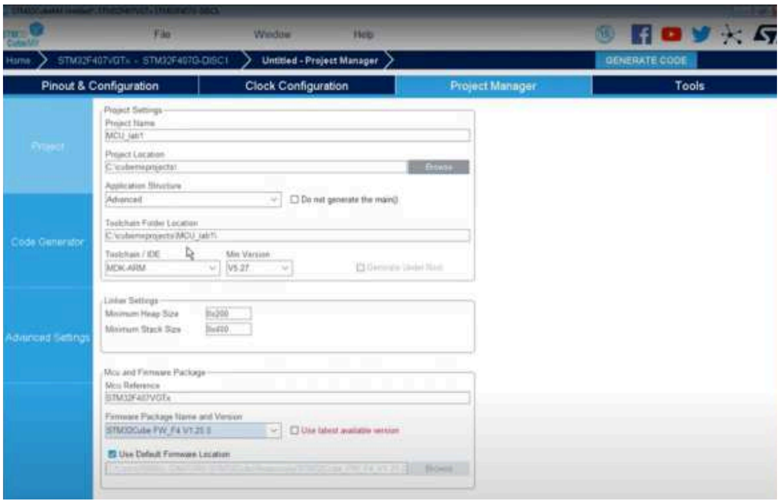


Figure 10

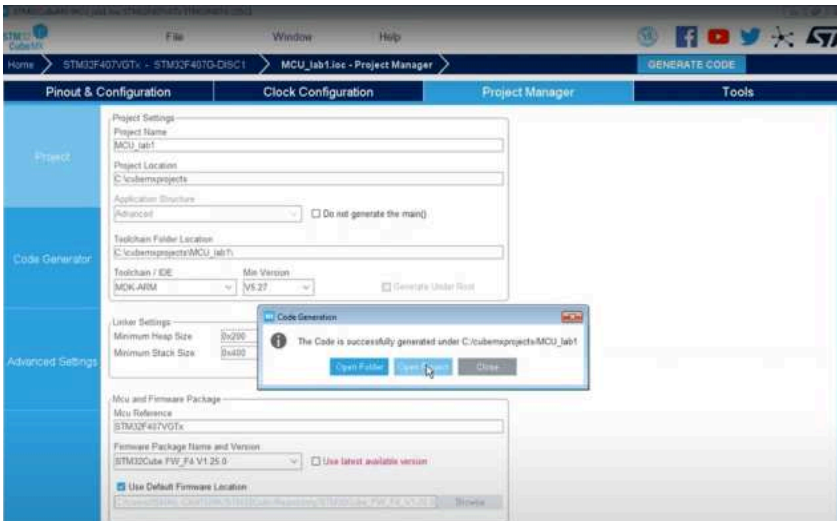


Figure 11

7. First, you need to come to the Pack Installer (Figure 12) and install the package of the relevant microcontroller, as you did in CubeMx. The relevant microcontroller is selected from the 'Search' menu and 'Install' is selected for the required package(.....) (Figure 13).

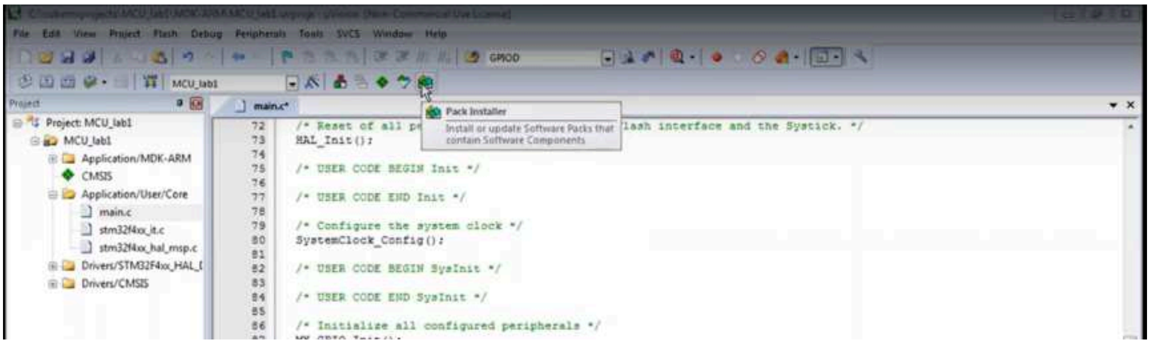


Figure 12

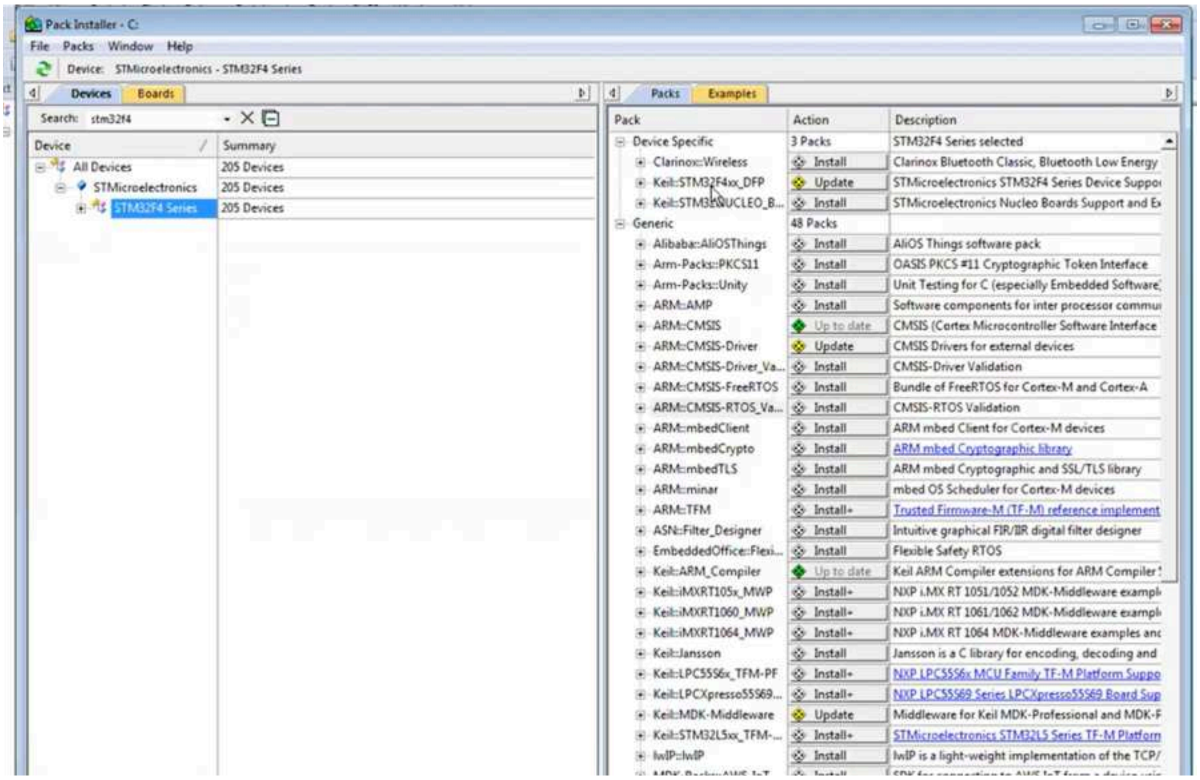


Figure 13

8. Select “Options for Target” menu (Figure 14). Come to the “Debug” menu and select “ST Link Debugger” and ‘Settings’, respectively (Figure 15). Then choose “Reset and Run” from “Flash Download" so that when you run our code, it will reset its previous information. If you do not select it, you have to press the reset button for the code to run (Figure 16). When you go back to the previous menu, you should choose the “C++” menu and select the “Level 0” for Optimization (Figure 17).

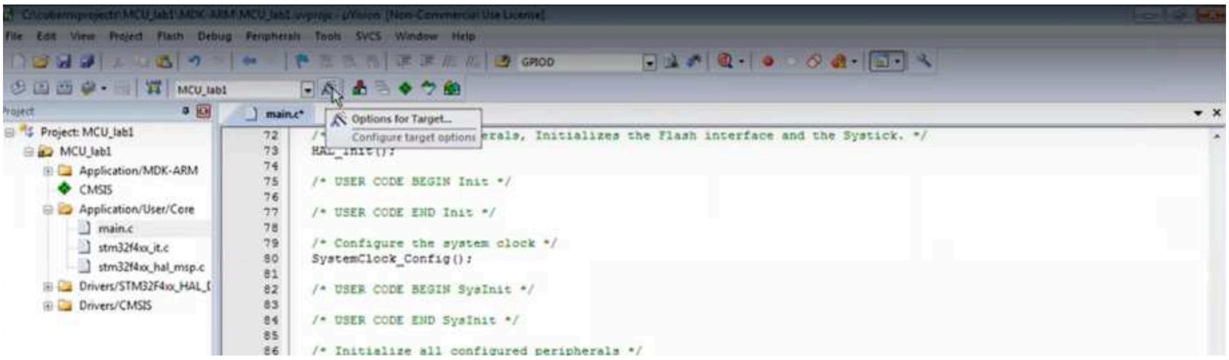


Figure 14

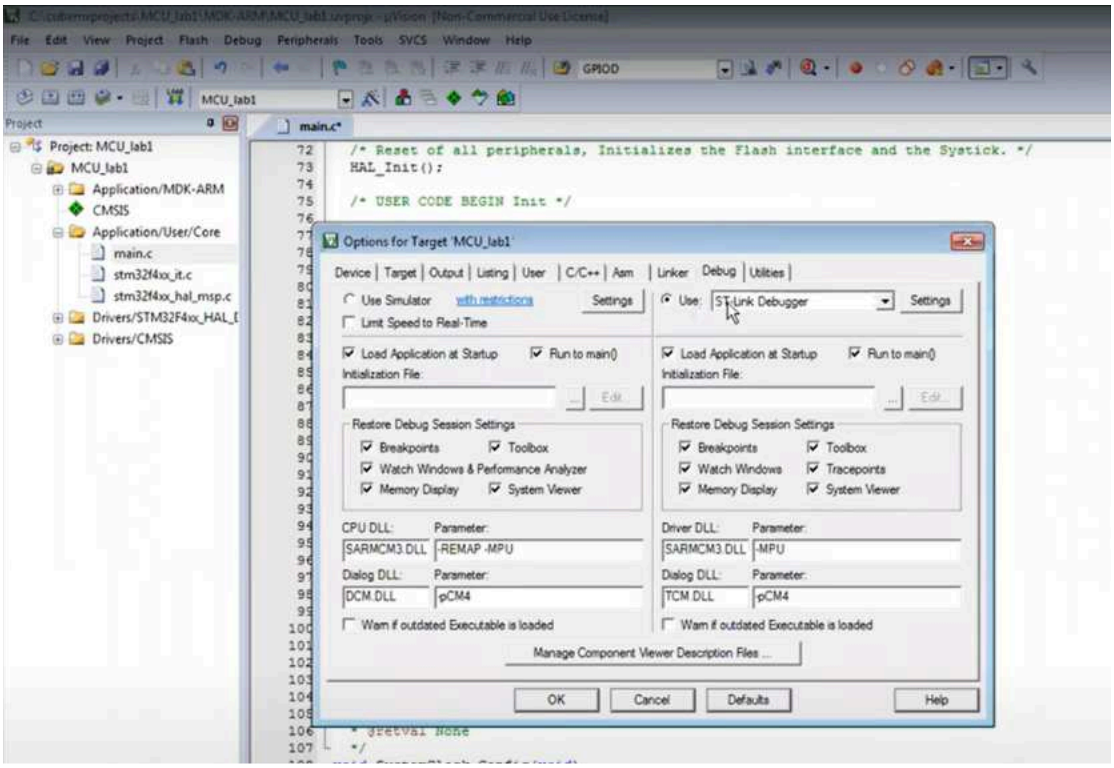


Figure 15

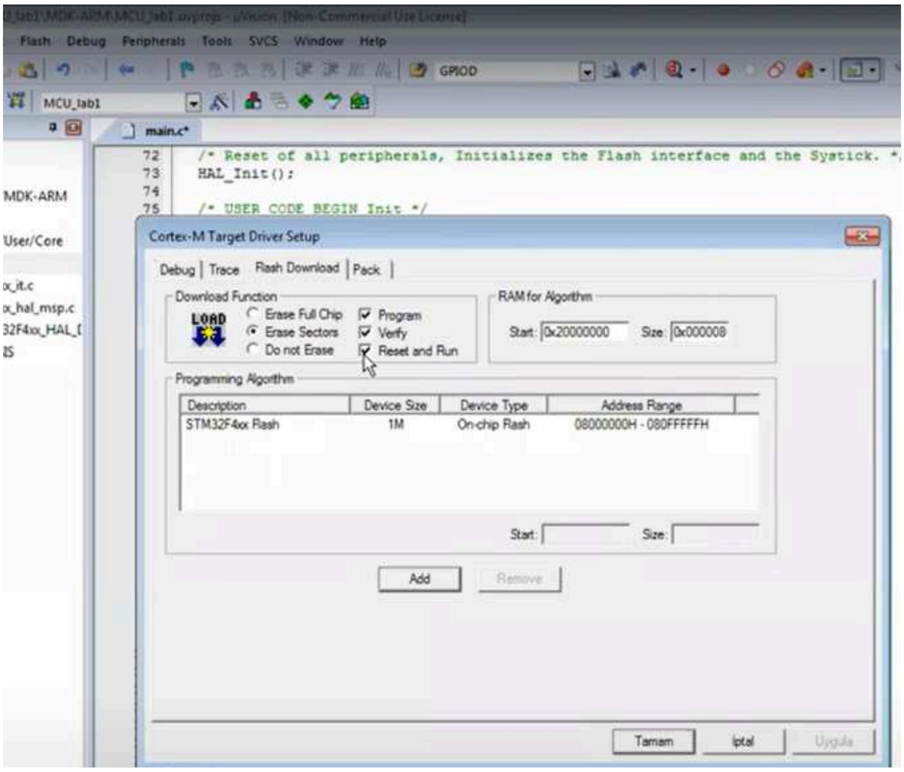


Figure 16

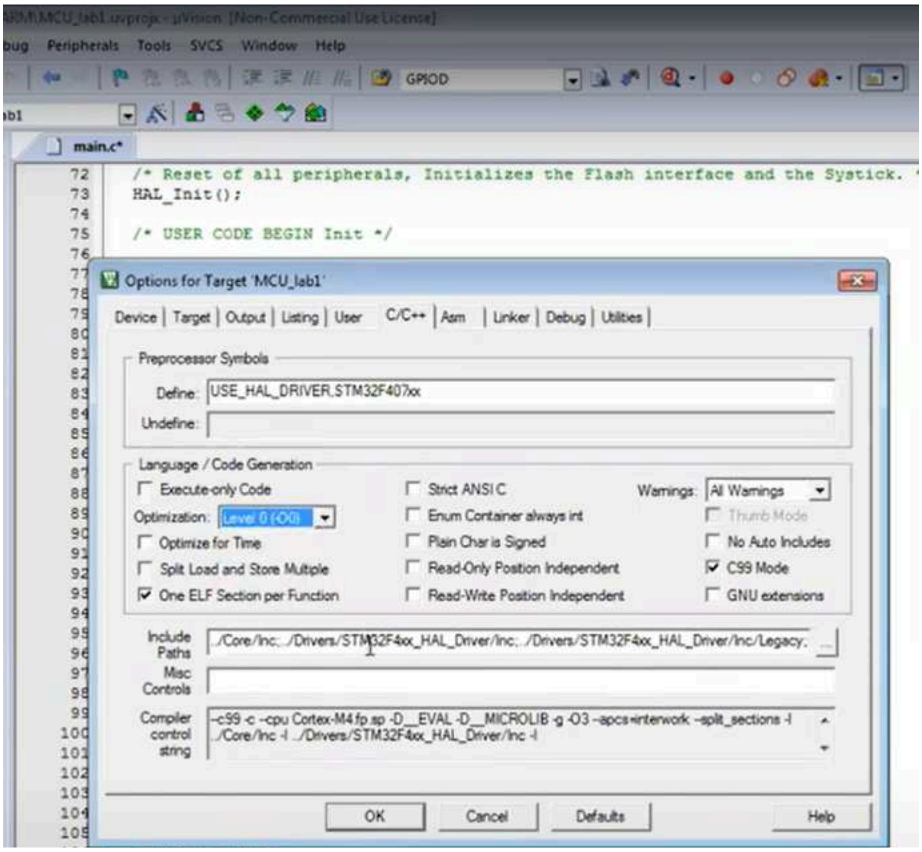


Figure 17

9. Now you can start to write our code. The “main.c” file on the left is the file created for us in Keil µVision. Here, you can write various codes using the C language.

   Follow the steps below for each code you want to run.

- Write the relevant codes in the part reserved for the user in the while loop.
- After writing the codes, click the build button to create the hex file and other files that will be uploaded to the microcontroller.
- Click the ‘Load’ button to load the codes into the microcontroller.

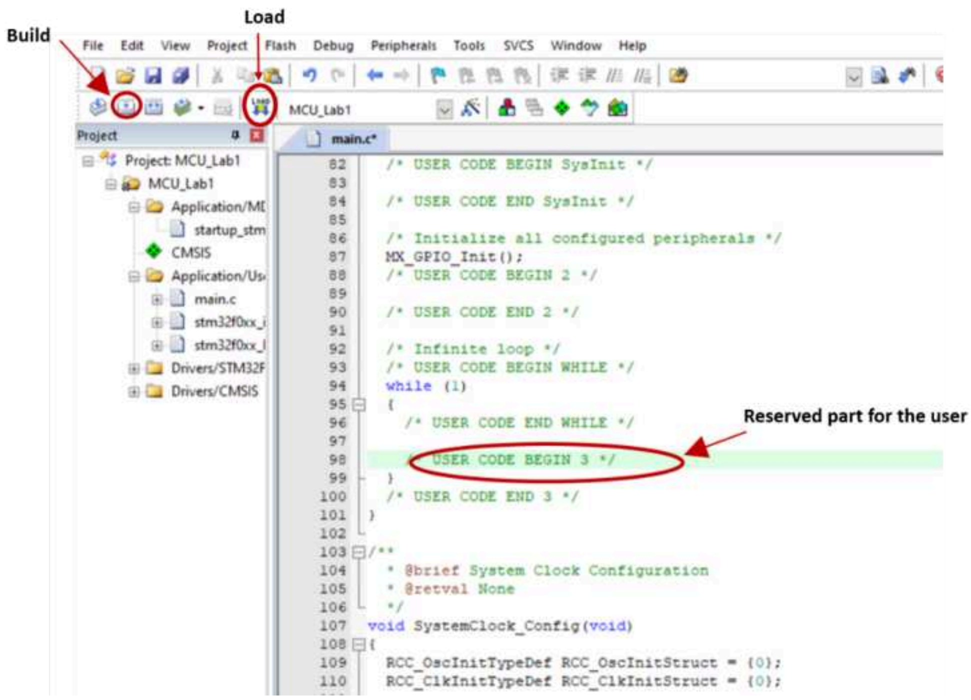


Figure 18

1. Use the HAL_GPIO_TogglePin function. Write the following code. Then write the same code without using the ‘HAL_Delay’ function.

   //Toggle the LED connected to the D12 pin at half-second intervals.
   HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12); // to toggle led which is connected to the D12 pin.
   HAL_Delay(500); //Wait 500 ms

2. Use the ‘HAL_GPIO_WritePin’ function. Write the following code.

   // Toggle the LED at 2 second intervals
   HAL_GPIO_WritePin (GPIOD, GPIO_PIN_14, GPIO_PIN_SET); //Write logic 1 to the output data register of the pin
   HAL_Delay(2000); //Wait 2 s
   HAL_GPIO_WritePin (GPIOD, GPIO_PIN_14, GPIO_PIN_RESET); //Write logic 0 to the output data register of the pin
   HAL_Delay(2000); //Wait 2 s

3. Use the Output Data Register (ODR) directly

```
// Use Output Data register directly
GPIOD->ODR|=0xF000; // Turn on the leds connected to the D12,D13,D14&D15 pins.
```

4. Toggle the LEDs using ‘ODR’.

```
// Use Output Data register directly to do toggle leds
GPIOD->ODR|=0xF000; // Turn on the leds connected to the D12,D13,D14&D15 pins.
HAL_Delay(2000); //Wait 2 s
GPIOD->ODR&=0x0000; // Turn off the leds connected to the D12,D13,D14&D15 pins.
HAL_Delay(2000); //Wait 2 s
```

5. Floating light respectively using ‘ODR’

```
//bitwise shifting
GPIOD->ODR|=0xF000; //All of the leds on
HAL_Delay(500); //Wait 500 ms
//Shift the bits right
for (i=1;i<5;i++)
{
  GPIOD->ODR>>=1; //Shift the bits right
  HAL_Delay(500); //Wait 500 ms

}
```

6. Floating light respectively using ‘ODR’.

```
//bitwise shifting
 GPIOD->ODR=0x0F00; //All of the leds off (assign bit values)
 HAL_Delay(500); //Wait 500 ms
 //Shift the bits left
 for (i=1;i<5;i++)
     {
       GPIOD->ODR<<=1; //Shift the bits left
       HAL_Delay(500); //Wait 500 ms
      }
```

7. The program that toggles the LEDs when we push the button using IDR (Input Data Register).

```
if (GPIOA->IDR&0x0001) //Checking if the button is pushed
{
  HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12); // to toggle led which is connected to the D12 pin.
  HAL_Delay(200); //Wait 200 ms
}
```

EXPERIMENT 3: GENERAL-PURPOSE INPUT/OUTPUT (GPIO)

Objectives

The objectives of Experiment 3 are

- to learn how to use
- ✔ GPIO Output Data Register (ODR),
- ✔ Reading Button Value using Input Data Register (IDR),
- ✔ Debugger,
- ✔ Bit Set Reset Register (BSRR),
- ✔ GPIO_ReadPin function

Apparatus Required:

- STM32CubeMx
- Keil µVision (MDK ARM)
- STM32 ST-Link Utility
- STM32F4 Microcontroller
- STM32F4 Reference Manual
- STM32F4 User Manual

Preliminary Work:

1. Study the GPIO (lecture 4) notes.
2. Write the codes of the experimental work in Keil µVision.

Experimental Work:

1. Reading Button Value (Button debouncing). You can understand whether your code is running and control the changes of the variable (Figure 1->Start/Stop Debug Session) using the debugger. Come to the i variable and right click. We select "Add i to" and "Watch 1" (Figure 2). Then click to "Run" and follow the changes of variable i (Figure 1). Right-click on the i which is in the Watch 1 window to convert the i displayed as hexadecimal to decimal. Here you can reset the i value by pushing the reset button on STM32F4G-DISC card and doing a hardware reset.

```
// Program that increase the value of the variable i by one each time the button is pushed

    if (GPIOA->IDR&0x0001) //Checking if the button is pushed using IDR
      {
       i=i+1; // Increase the value of the variable i by one each time the button is pushed
      }
```

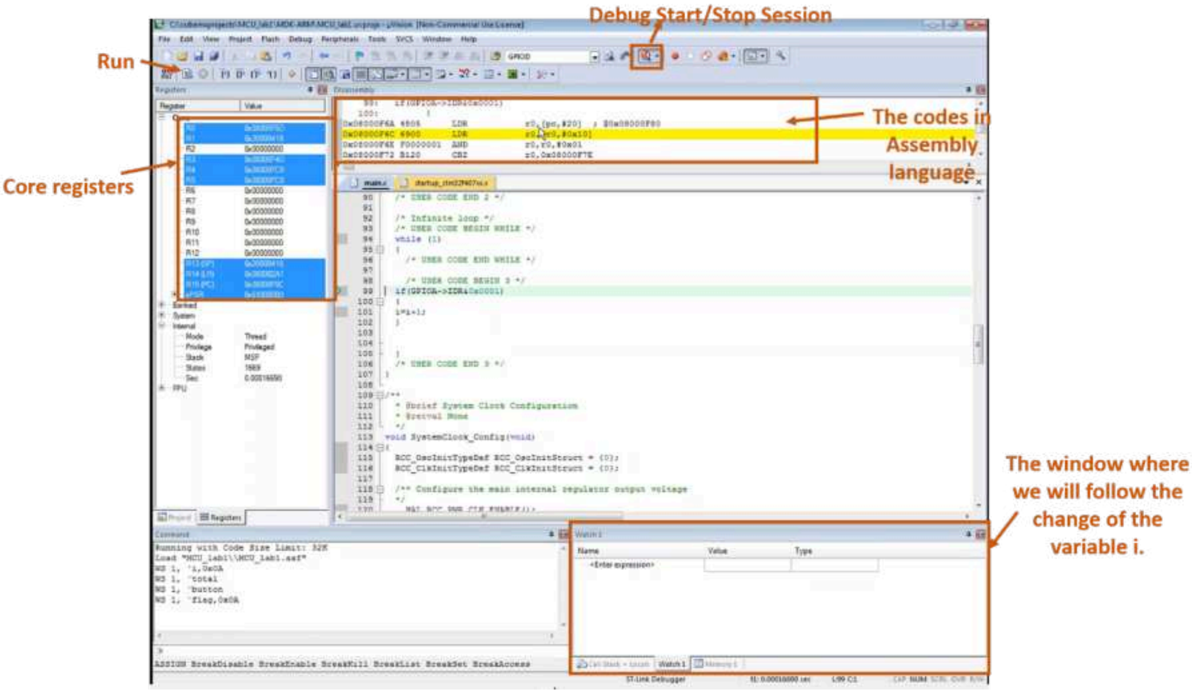


Figure 1

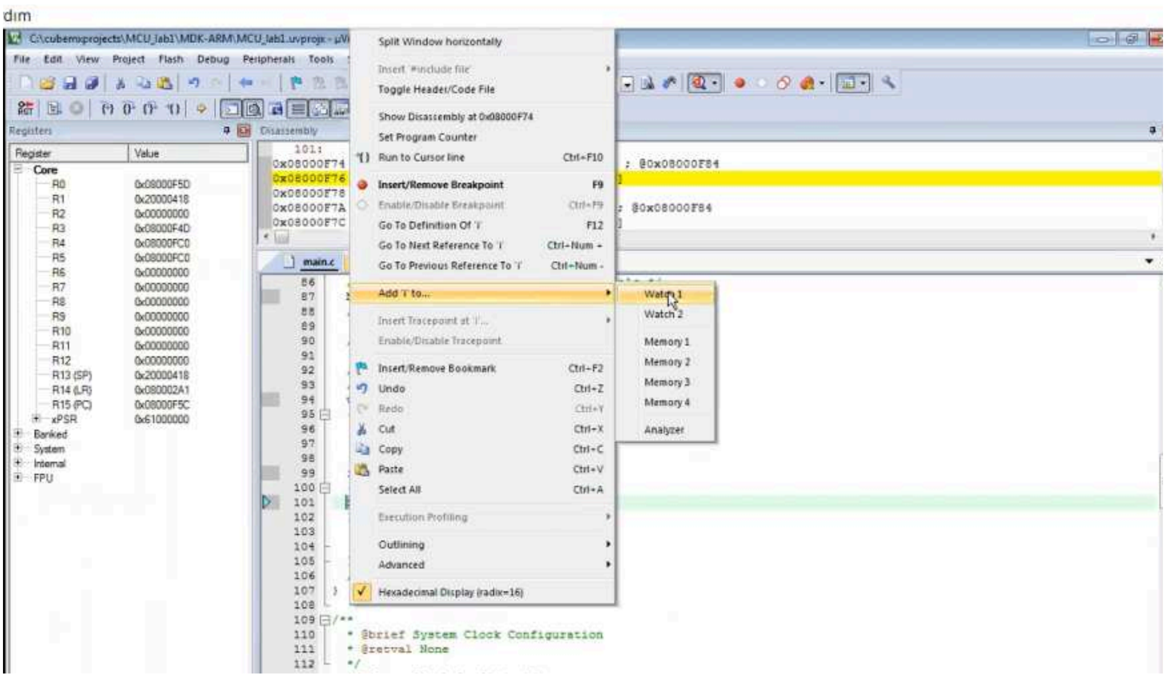


Figure 2

2. Reading Button Value (Prevent button debouncing using HAL_Delay)

// Program that increments the value of the variable i by one each time the button is pushed

```
if (GPIOA->IDR&0x0001) //Checking if the button is pushed using IDR
 {
  i=i+1;
  HAL_Delay(200); //Wait 200 ms
 }
```

3. The program that turns on the LEDs when we push the button.

```
if (GPIOA->IDR&0x0001) //Check if the button is pushed
{
 i=i+1;
 HAL_Delay(200); //Wait 200 ms
 GPIOD->ODR=0xF000; //Assign 1 to the PD12, PD13, PD14 & PD15 pins
}
```

4. The program that turns the LEDs on when we push the button otherwise turns the LEDs off (use BSRR to assign logic 0 to the relevant pins)

```
GPIOD->BSRR=0xFFFF0000; //Reset ODR pins of the D port using BSRR
if (GPIOA->IDR&0x0001) //Check if the button is pushed using IDR
  {
   i=i+1;
   GPIOD->ODR=0xF000; //Assign 1 to the PD12, PD13, PD14 & PD15 pins and
   HAL_Delay(2000); //Wait 2 second
  }
```

5. The program that turns the LEDs on when we push the button otherwise turns the LEDs off (use ODR to assign logic 0 to the relevant pins)

```
GPIOD->ODR=0x0000; //Assign logic 0 to the pins at the D port
if (GPIOA->IDR&0x0001) //Checking if the button is pushed using IDR
  {
   i=i+1;
   HAL_Delay(200); //Wait 0.2 second
   GPIOD->ODR=0xF000; //Assign 1 to the PD12, PD13, PD14 & PD15 pins,
   HAL_Delay(2000); //Wait 2 second
  }
```

6. The program that turns the LEDs on when we push the button otherwise turns the LEDs off (use ODR to assign logic 0 to the relevant pin and use ReadPin function to read the button).

```
GPIOD->ODR=0x0000; //Assign logic 0 to the pins at the D port

 if(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)) //Check if the button is pushed using ReadPin function

{
  i=i+1;
  HAL_Delay(200); //Wait 0.2 second
  GPIOD->ODR=0xF000; //Assign 1 to the PD12, PD13, PD14 & PD15 pins, assign
  HAL_Delay(2000); //Wait 2 second
 }
```

7. A program that increases the value of i by one each time a button is pushed, at the same time, if i is an even number, toggles the related LED(which is connected to the PD12 pin), otherwise it turns off the LED (Control the i value using debugger). Use debug to see how to change the i variable.

```
while (1)
{

if(GPIOA->IDR&0x0001) //Check if the button is pushed using IDR

{

i=i+1; // Increase the value of i by one each time a button is pushed.

HAL_Delay(200);

}

// If i is an even number, let the led toggle otherwise led is off

if(i%2==0) //Check for an even number

{

 HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12); //Toggle 12th pin of the D port.

 HAL_Delay(200); //Wait 0.2 second

}

else

{

 GPIOD->BSRR=0xFFFF0000; //Reset ODR's pins of the D port using BSRR

}

}
```

8. Create a function named button that checks if the value of the variable i is an even number or not. If the value of the variable i is an even number, this function will turn on the LEDs connected to the 12th and 14th pins of the D port, otherwise LEDs which are connected to 13th and 15th pins of the D port. Then, write another code in the while loop that checks if the button is pushed and increments the value of the variable i by one each time the button is pushed. Call the button function here. Use the debugger to monitor what the value of the variable i is each time you push the button, and also observe which led is lit based on that value.

   (Create the button function at the part of the Private function prototypes (PFP) in the main.c file.
   Write a code in while loop to check if button is pushed and call button function here)

   //If it is an even number, the function that turn on the LEDs connected to the 12th. and 14th pins of the D port, otherwise the function that turns on the LEDs connected to the 13th. and 15th pins of the D port.

```
void button (int a) // Creating a function named button
{
if (a%2==0) //Check if a is an even number
{
 GPIOD->ODR=0x5000; //Turn on the LEDs which are connected to 12th. and 14th pins of the D port
}
else
{
  GPIOD->ODR=0xA000; //Turn on the LEDs which are connected to 13th. and 15th pins of the D port
}
}


while (1)
{
//Checking if the button is pushed and calling the button function
if( GPIOA->IDR&0x0001) // Check if the button is pushed
{
 i=i+1;
 HAL_Delay(200);
}
 button(i); //Call the button function
}
```

EXPERIMENT 4: INTERRUPTS

Objectives

The objectives of Experiment 4 are

- to learn how to use interrupt peripherals

Apparatus Required:

- STM32CubeMx
- Keil µVision (MDK ARM)
- STM32 ST-Link Utility
- STM32F4 Microcontroller
- **A Jumper Cable (female-female)**

Preliminary Work:

1. Study the Interrupt (L05) notes
2. Write the codes of the experimental work in Keil µVision.

Experimental Work:

1. Create a new project in CubeMx. Select STMF407VGTx and then STM32F407G-DISC1. First adjust the Pinout&Configuration settings. Close the unnecessary pins. Select the PA0 pin as GPIO_EXTI0 and PA1 pin as GPIO_EXTI1. Select the PD 12-13-14-15 pins as GPIO_Output.

2. Come to the System Core menu. You can change the pin configurations by selecting related pins from here (Figure 1). Select the pull down for PA0&PA1 pins. Select "Output Push Pull" for the GPIO Mode, "Low" for the GPIO output level & Maximum output speed for related pins (PD12- PD13- PD14- PD15).

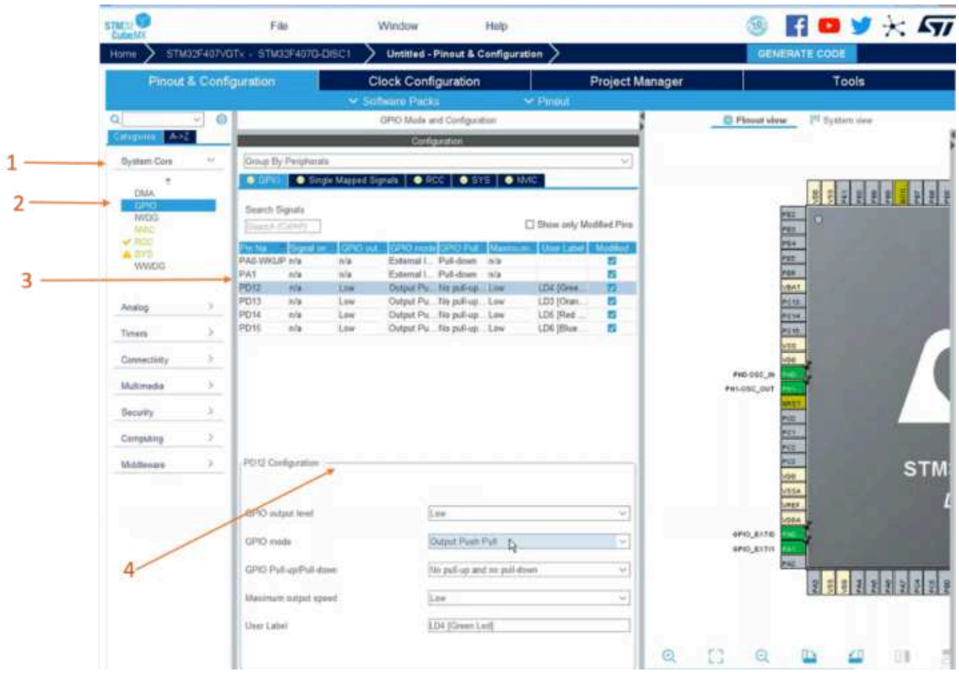


Figure 1

3. Come to the NVIC menu. Firstly set the enable mode for the EXTI line0 & EXTI line1(Figure 2, 1 and 2 steps). Then, identify the priority levels of the interrupts. Select the Priority Group as 2 bits (which indicate how many bits are needed to identify the priority level) (Figure 2, 3. step). Then, select preemption priority as 1 for the EXTI0 and as 2 for the EXTI1 (Figure 2, 4. step).

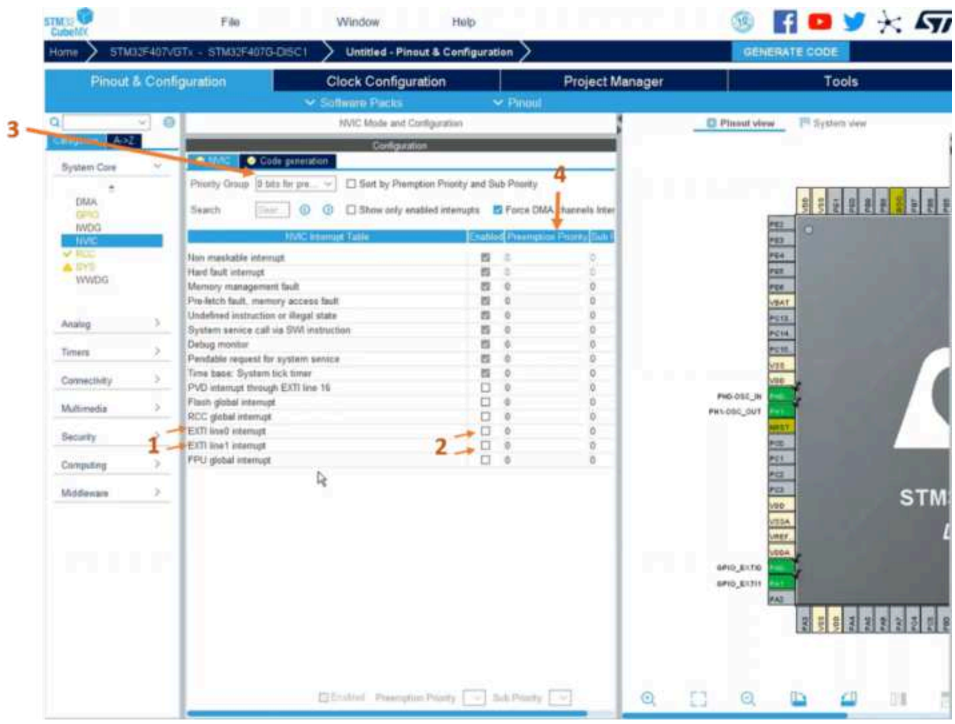


Figure 2

4. Come to the Clock Configuration menu and control the settings as in Figure 3.

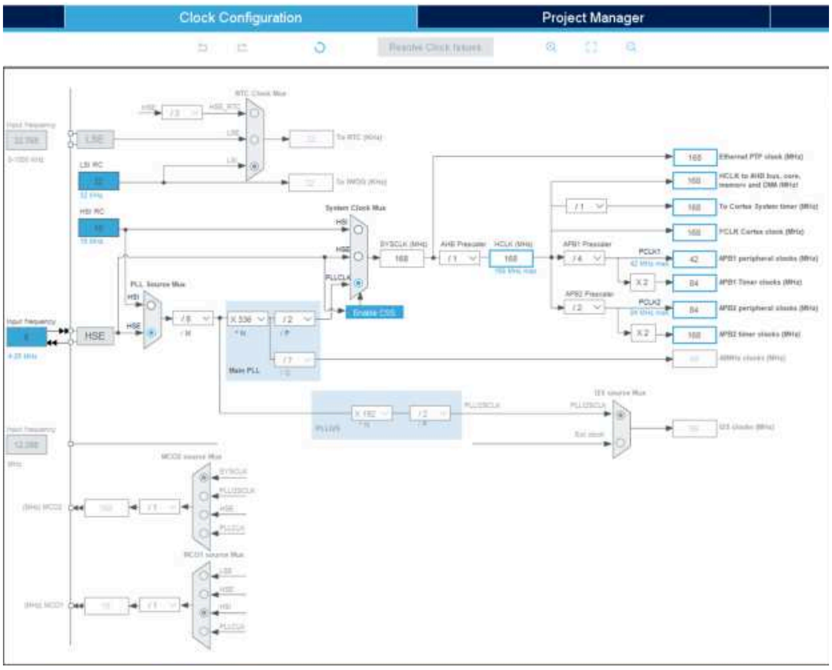


Figure 3

5. Come to the Project Manager and adjust necessary settings as in Figure 4. Then click the Generate Code (Figure 4). Keil µVision programme will be opened. You can see the settings which were already done in CubeMx in the main.c file in Keil µVision (Figure 5). You can change the adjustments from here without going back to the CubeMx. Build the codes in the main.c file. Double click the interrupt file (stm32f4xx.c). You can see the interrupts functions here (Figure 6). You can write codes in the functions. If you want to understand what the function does, you can right click on the function and select the 'Go to Definition…' shown as in Figure 7.

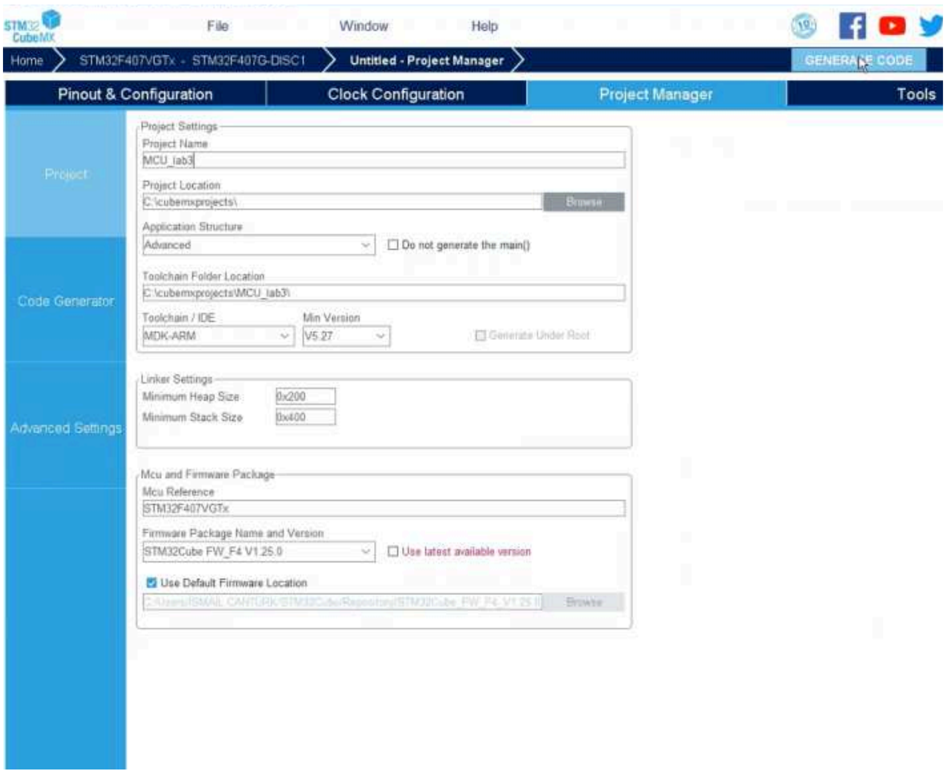


Figure 4

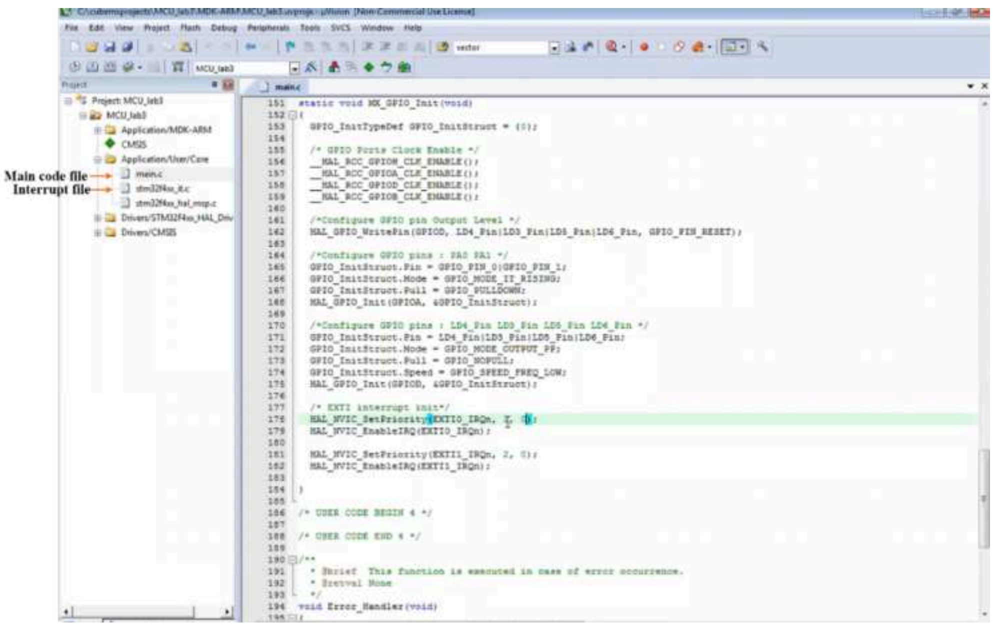


Figure 5

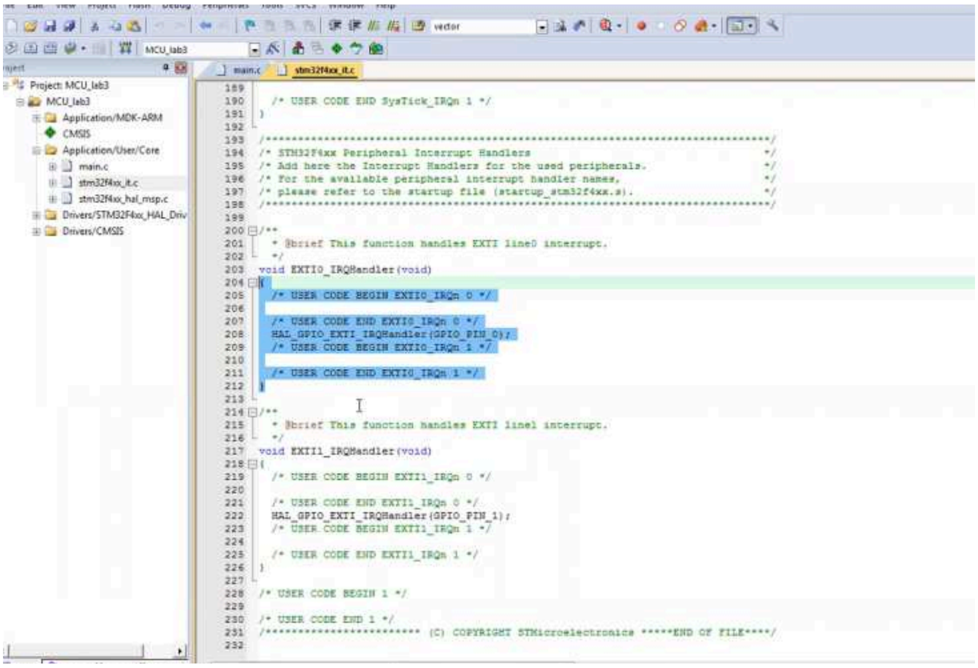


Figure 6

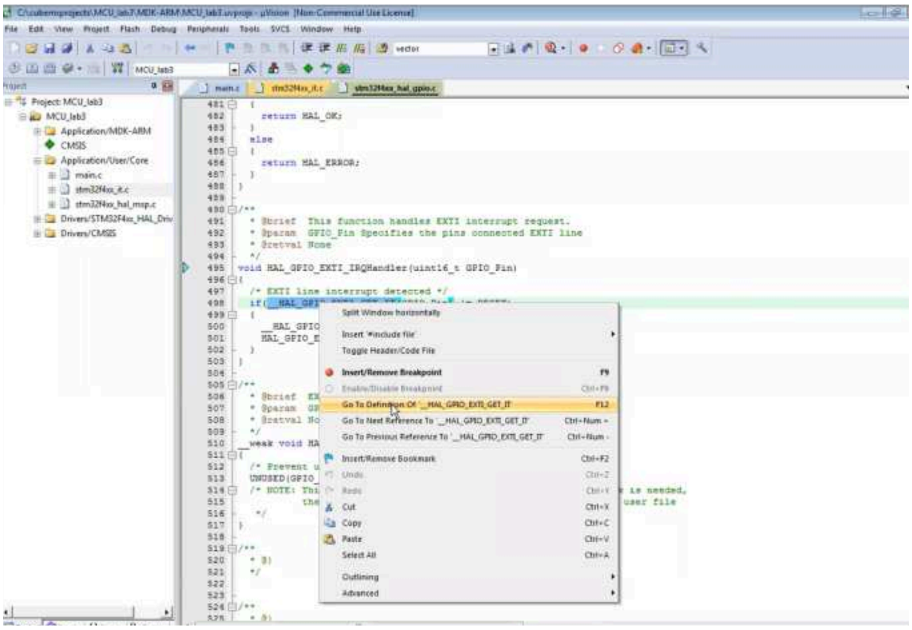


Figure 7

6. **a.** Write an interrupt handler function that increases by 1 the value of the variable i if an interrupt is generated from the PA0 pin. Write the codes in EXTI0_IRQHandler function in stm32f4xx_it.c file.
   b. Write an interrupt handler function that increases by 1 the value of the variable a if an interrupt is generated from the PA1 pin. Write the codes in EXTI1_IRQHandler function in stm32f4xx_it.c file.
   Don't forget to identify variable a and i variables in the 'private variables' part of the stm32f4xx_it.c file.
   Observe the change of the i variable when you push the button using debugger.

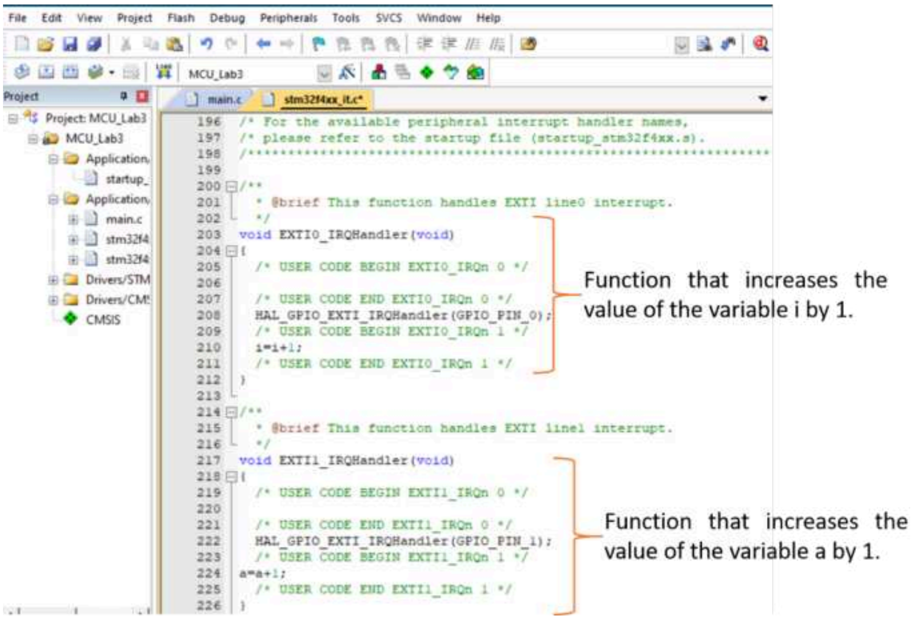


Figure 8

7. Follow the instructions given in a, b, c in order. The relevant codes are given below.

a. When there is no interrupt, the LED connected to the 12th pin lights up continuously. (Write the relevant code inside the while loop in main.c).

```
while (1)
{
  /* USER CODE BEGIN 3 */
 //Light the 12th pin when the interrupt handler is not working
 HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12); // Toggle the PD12 LED
 HAL_Delay(100); //Wait 100 ms
}
/* USER CODE END 3 */
```

b. When the interrupt is received from the PA0 pin, the value of the i variable increases by 1. Reset all pins connected to port D using BSRR. After the LED is connected to the PD13 pin lights for 5 seconds, all the pins connected to the D port are reset again. Write the relevant code inside the EXTI0_IRQHandler function.

```
void EXTI0_IRQHandler(void)
{
 HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
 /* USER CODE BEGIN EXTI0_IRQn 1 */
i=i+1; //increase i value by 1
GPIOD->BSRR=0xFFFF0000; //Reset the PD pins
GPIOD->BSRR=0xFFFF2000; // Set 1 PD13
HAL_Delay(5000); //Wait 5 s
GPIOD->BSRR=0xFFFF0000; // Reset the PD pins
 /* USER CODE END EXTI0_IRQn 1 */
}
```

c. When the interrupt is received from the PA1 pin, the value of the ‘a’ variable increases by 1. Reset all pins connected to port D using BSRR. After the LED is connected to the PD15 pin lights for 5 seconds, all the pins connected to the D port are reset again. Write the relevant code inside the EXTI1_IRQHandler function.

```
void EXTI1_IRQHandler(void)
{
 HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
 /* USER CODE BEGIN EXTI1_IRQn 1 */
a=a+1; //increase a value by 1
GPIOD->BSRR=0xFFFF0000; //Reset the PD pins
GPIOD->BSRR=0xFFFF8000; // Set 1 PD15
HAL_Delay(5000); //Wait 5 s
GPIOD->BSRR=0xFFFF0000; // Reset the PD pins
 /* USER CODE END EXTI1_IRQn 1 */
 }
```

d. Compile the codes and upload them to the microcontroller. Observe the change of i and a variable using debugger. Use the button on the microcontroller to send an interrupt from the PA0 pin. Use the 5V on the microcontroller discovery card to send the interrupt from the PA1 pin (You can connect 5V to the PA1 pin with the help of a jumper).

8. Use priorities of the interrupts (Go back to the 3 to remember the priorities of the interrupts). Use the same codes as in 7.
   a. After giving an interrupt from PA0 pin, give another interrupt from PA1 pin before the interrupt handler is completed. Observe the changes of i, variables and LEDs. Observe the ‘**Tail Chaining Scenario**’.
   b. After giving an interrupt from PA1 pin, give another interrupt from PA0 pin before the interrupt handler is completed (**Late Arrival Scenario**). Observe the changes of i, variables and LEDs.

9. Learn how to use the interrupt mask register (Examine the properties of the register from Reference Manual). Write a code inside the EXTI1_IRQHandler function. When a value is greater than 5, mask pin 1 using the Interrupt Mask Register. Use EXTI->IMR statement to reach the interrupt mask register and assign a hexadecimal number to this register that will set the corresponding pin value to zero. Build and Load the code. Use a debugger to observe the chaining of a value. Write down your observations about what the result was.

```
void EXTI1_IRQHandler(void)
{
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
  /* USER CODE BEGIN EXTI1_IRQn 1 */
a=a+1;
GPIOD->BSRR=0xFFFF0000;//Reset the PD pins
GPIOD->BSRR=0xFFFF8000;// Set 1 PD14
HAL_Delay(5000); //Wait 5 s
GPIOD->BSRR=0xFFFF0000;// Reset the PD  pins
//Since a>5, interrupts from line 1 are not detected
if (a>5)
{
EXTI->IMR=0x7FFFFD; //Masked the 1. pin
}
  /* USER CODE END EXTI1_IRQn 1 */
}
```

EXPERIMENT 5: TIMERS

Objectives

The objectives of Experiment 5 is

- to learn how to use Timer peripherals

Apparatus Required:

- STM32CubeMx
- Keil µVision (MDK ARM)
- STM32 ST-Link Utility
- STM32F4 Microcontroller
- A Female-Female Jumper Cable

Preliminary Work:

1. Study the Timer (lecture 6,7) notes
2. Write the codes of the experimental work in Keil µVision.

Experimental Work:

1. Create a new project in CubeMx (Figure 1). Select STMF407VGTx, then STMF407G-DISC1 and finally Start Project (Figure 2). First adjust the Pinout&Configuration settings. Close the unnecessary pins (Figure 3).

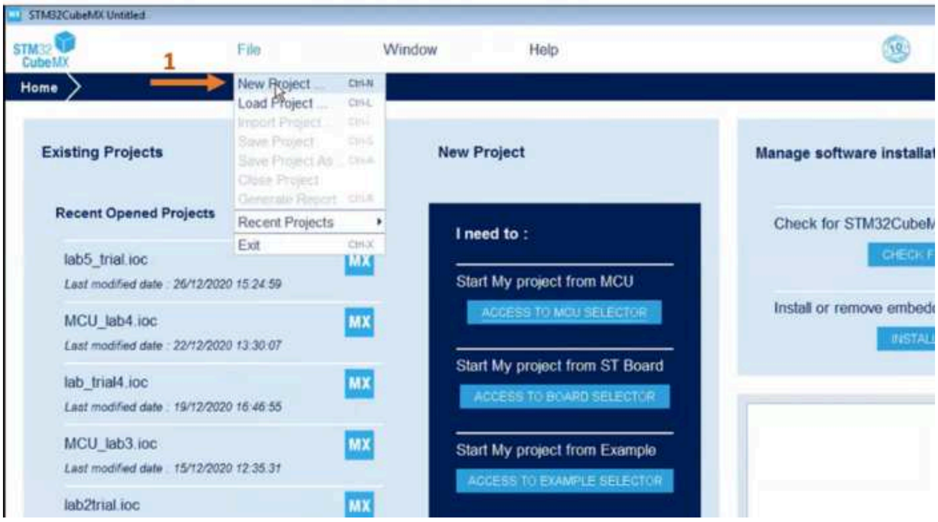


Figure 1

Figure 2

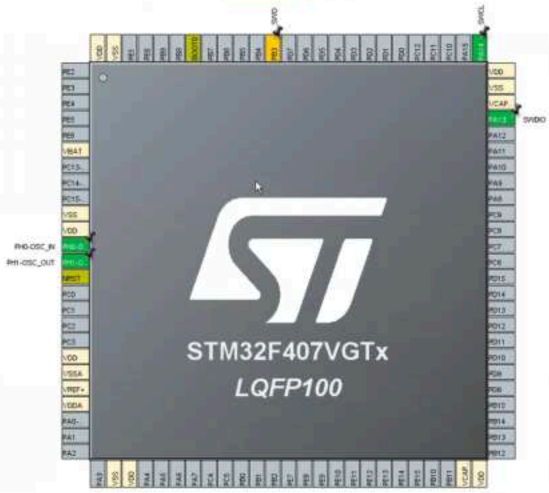


Figure 3

2. We use General Purpose Timers (TIM2 to TIM5) mostly, so look at the reference manual to get information about these timers. Find out which bus is connected to TIM2 from the reference manual. Then come back to CubeMx and check the speed of this bus in the Clock Configuration tab (Figure 4). Then come back to Pinout&Configuration tab and choose which timer you will use (we will use TIM2) and make the necessary settings for this timer (select clock source as internal clock) (Figure 5). Adjust these settings as the prescaler value is 41999, counter mode is up, counter period is 1999. Think about what the meaning of these adjustments are. Go to the NVIC tab and enable the interrupt (Figure 6). Also, go to NVIC in the System Core tab and adjust Preemption Priority value as 1 for the TIM2 (Figure 7).

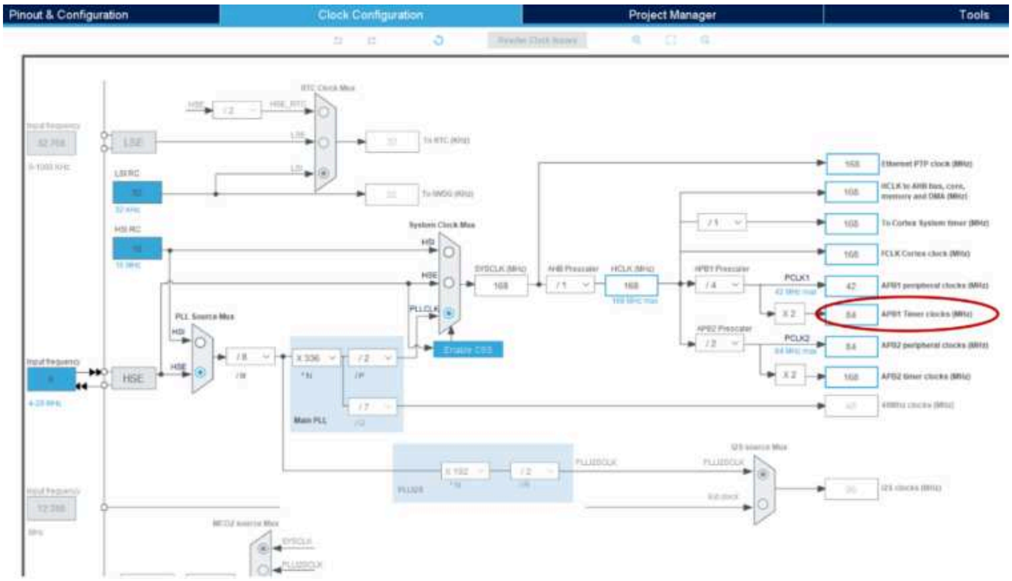


Figure 4

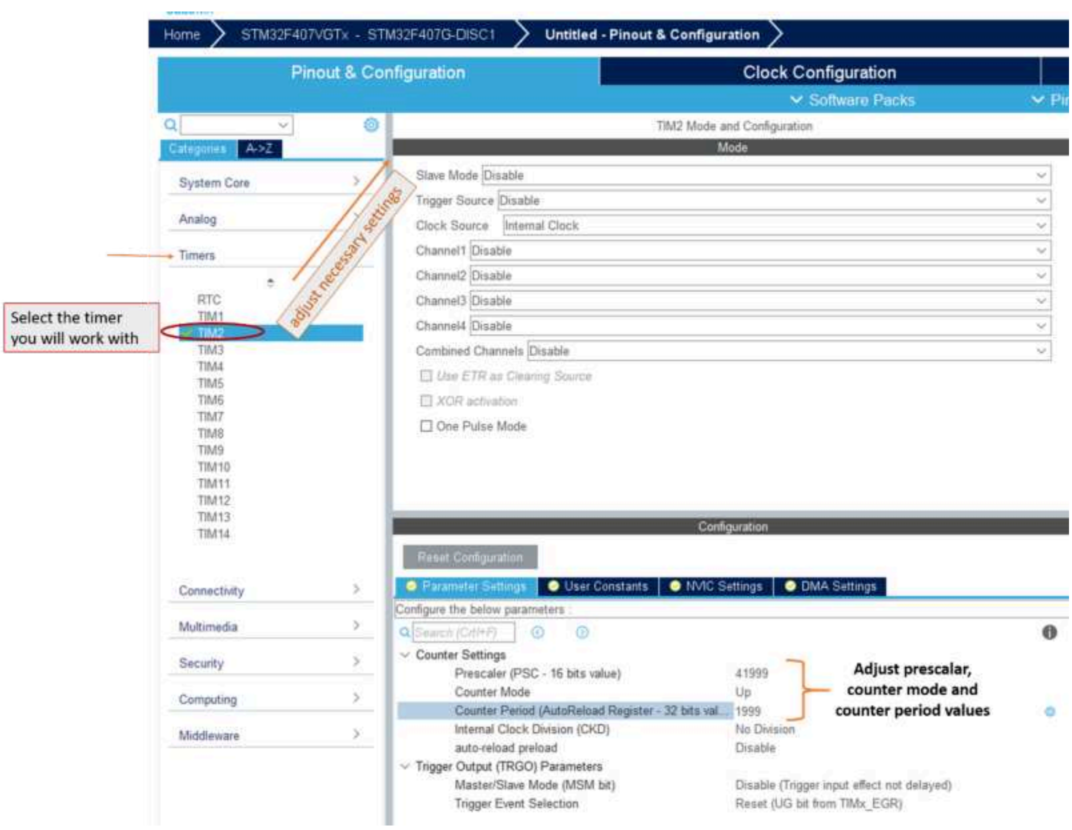


Figure 5

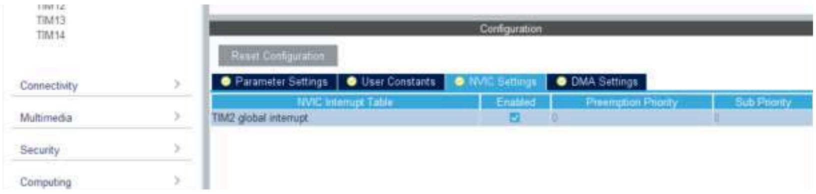


Figure 6

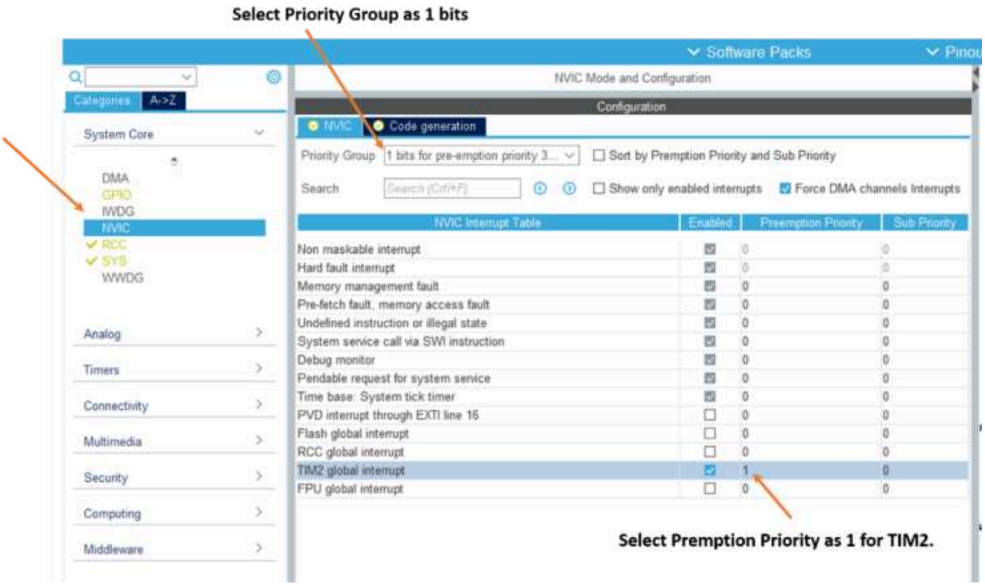


Figure 7

3. Come to the Project Manager tab and set necessary configurations here. Then you can continue with the Keil µVision. Don't close the CubeMx, because you will change something from CubeMx.

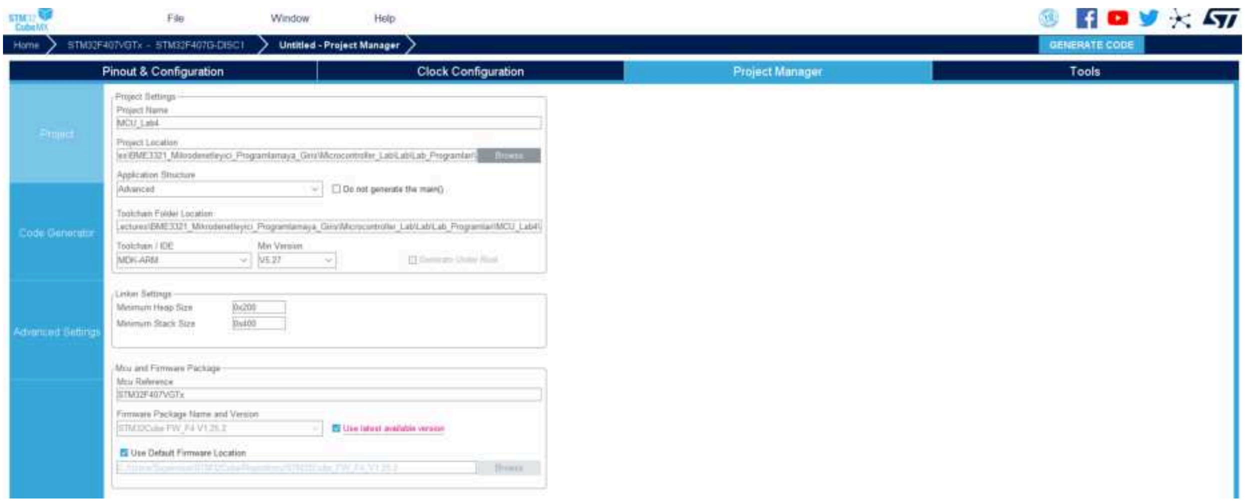


Figure 8

4. Build the main.c file.

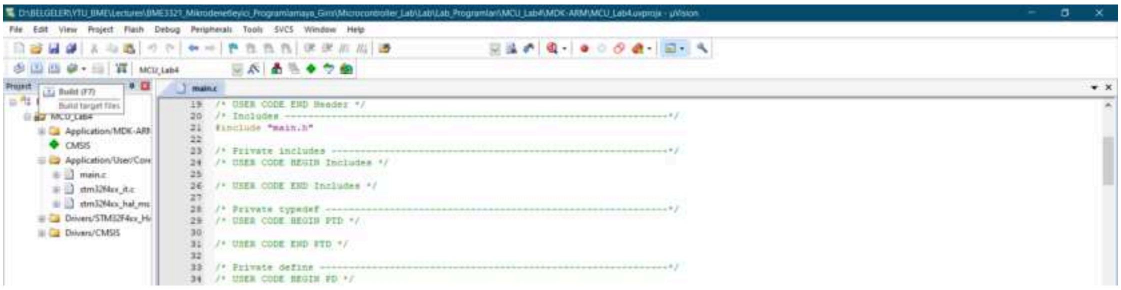


Figure 9

5. Start the TIM2 in interrupt mode using the HAL_TIM_Base_Start_IT function under /*USER CODE BEGIN 2*/ comment line in main.c file. Then go to the interrupt file (stm32f4xx_it.c) to write the interrupt.

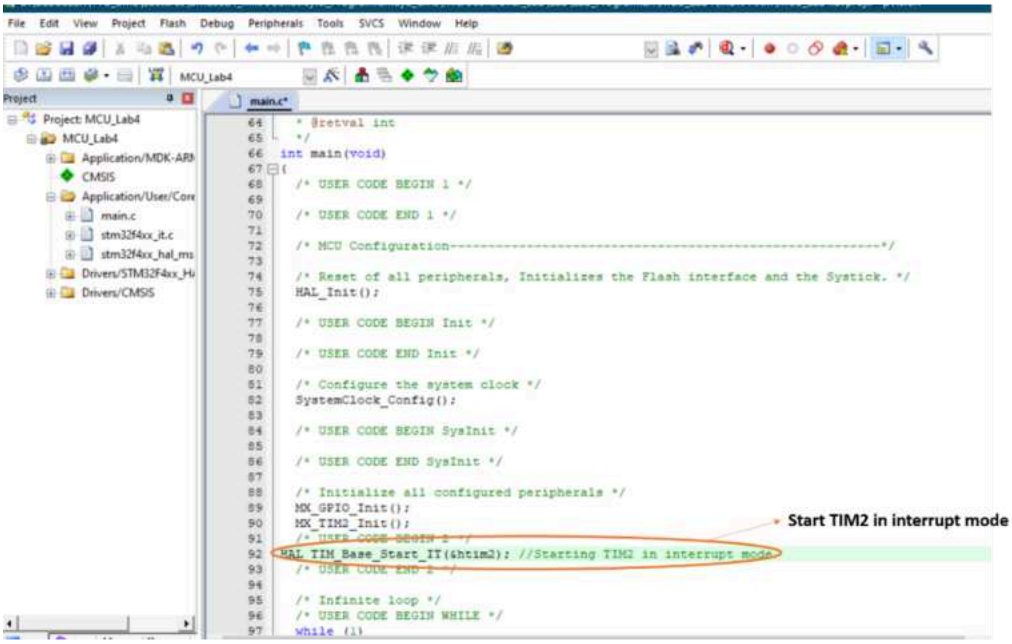


Figure 10

6. Come to the ‘TIM2_IRQHandler’ function in the stm32f4xx_it.c file. When the timer completes each period value, write the code that comes into the interrupt request and increases the i variable by 1. Don’t forget to define ‘int i’. Build the file and load the codes to the microcontroller. Observe the change of i variable using debugger. Think about what's going on inside the TIM2_Handler function and how.

```
 */
void TIM2_IRQHandler(void)
{
  /* USER CODE BEGIN TIM2_IRQn 0 */

  /* USER CODE END TIM2_IRQn 0 */
  HAL_TIM_IRQHandler(&htim2);
  /* USER CODE BEGIN TIM2_IRQn 1 */
  i=i+1;
  /* USER CODE END TIM2_IRQn 1 */
}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Figure 11

7. Make a clock application using the TIM2 timer. Define three variables: second, minute, hour. Increment the value of the second variable each time an interrupt request is generated. When the value of the variable i is equal to 60, reset it again and assign a value to the minute variable. When the value of the minute variable is equal to 60, the value of the minute is reset and the value of the hour variable is increased by one. When the value of the hour variable is equal to 12, the value of the hour variable is also reset. Let the cycle continue like this. Observe the change of variables using debug.(Figure 12.1)
   Change the htim2.Init.Period value as 19 in main.c file (Figure 12.2). Observe the change of variables using debug. Explain how there has been a change in the operation of the code.

```
void TIM2_IRQHandler(void)
{
  /* USER CODE BEGIN TIM2_IRQn 0 */

  /* USER CODE END TIM2_IRQn 0 */
  HAL_TIM_IRQHandler(&htim2);
  /* USER CODE BEGIN TIM2_IRQn 1 */
  second=second+1;
  if (second==60)
  {
    second=0;
    minute=minute+1;
  }

  if (minute==60)
  {
    minute=0;
    hour=hour+1;
  }

  if(hour==12)
  {
    hour=0;
  }
  /* USER CODE END TIM2_IRQn 1 */
}
```

Figure 12.1

```
TIM_MasterConfigTypeDef sMasterConfig = {0};

/* USER CODE BEGIN TIM2_Init 1 */

/* USER CODE END TIM2_Init 1 */
htim2.Instance = TIM2;
htim2.Init.Prescaler = 41999;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 19;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
{
  Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
{
  Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
```

Figure 12

8. Close the Keil µVision and go back to CubeMx to generate a PWM signal. Use another TIM (TIM4) so disable the clock source for TIM2 (Figure13). Find out which bus TIM4 is connected from the reference manual. Then come back to CubeMx and check the speed of this bus in the Clock Configuration tab. Then come back to Pinout&Configuration tab and choose which timer you will use (we will use TIM4) and make the necessary settings for this timer (select clock source as internal clock and PWM Generation CH4)(Figure 14). Adjust these settings as prescaler value is 41999, counter mode is up, counter period is 1999. Explain what the meaning of these adjustments are. Come to the Project Manager tab and set necessary configurations here. Then you can continue with the same Keil µVision file (Figure 15).

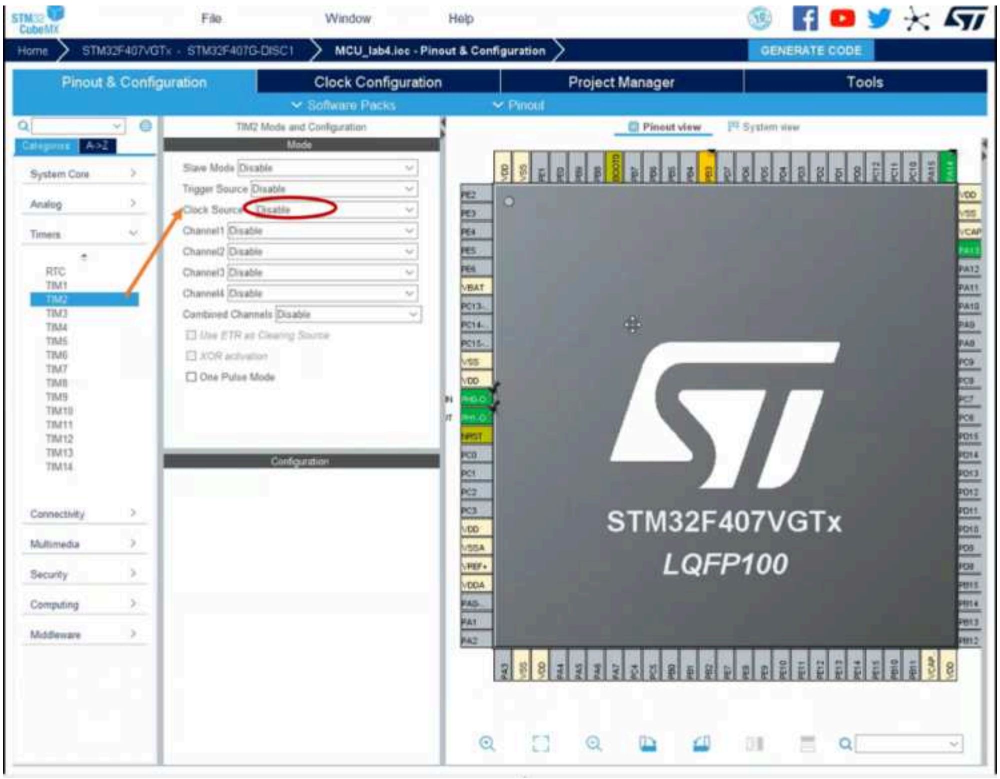


Figure 13

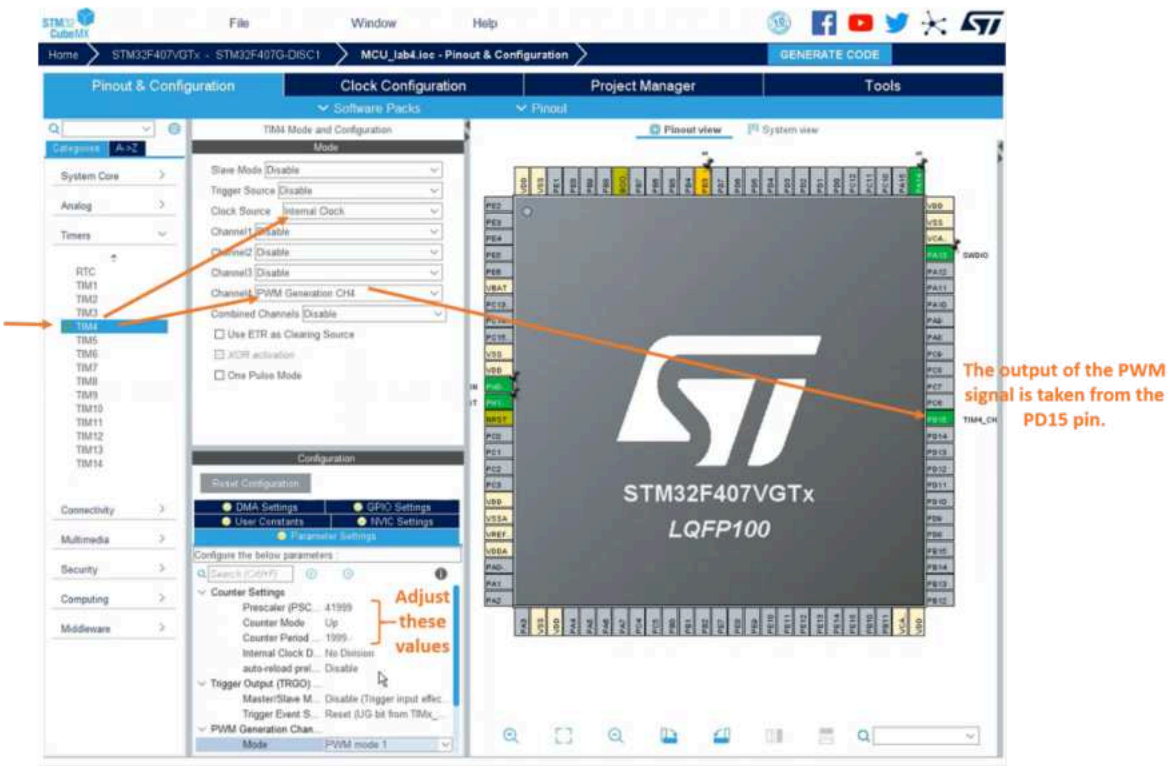


Figure 14

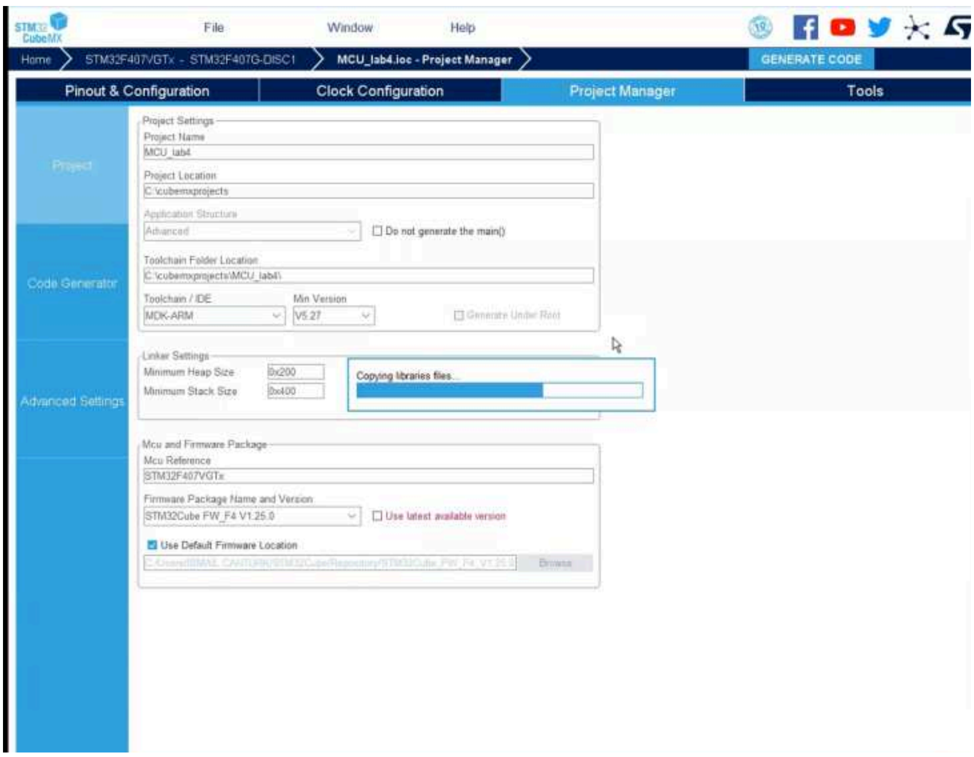


Figure 15

9. You can see the functions for PWM in the HAL Library as shown in Figure 16. We use the HAL_TIM_PWM_Start function. This PWM signal is on in 500 of the Counter Period and off in the remaining 1500. Use __HAL_TIM_SetCompare macron (Figure 17). Observe the condition of the LED connected to the PD15 pin.

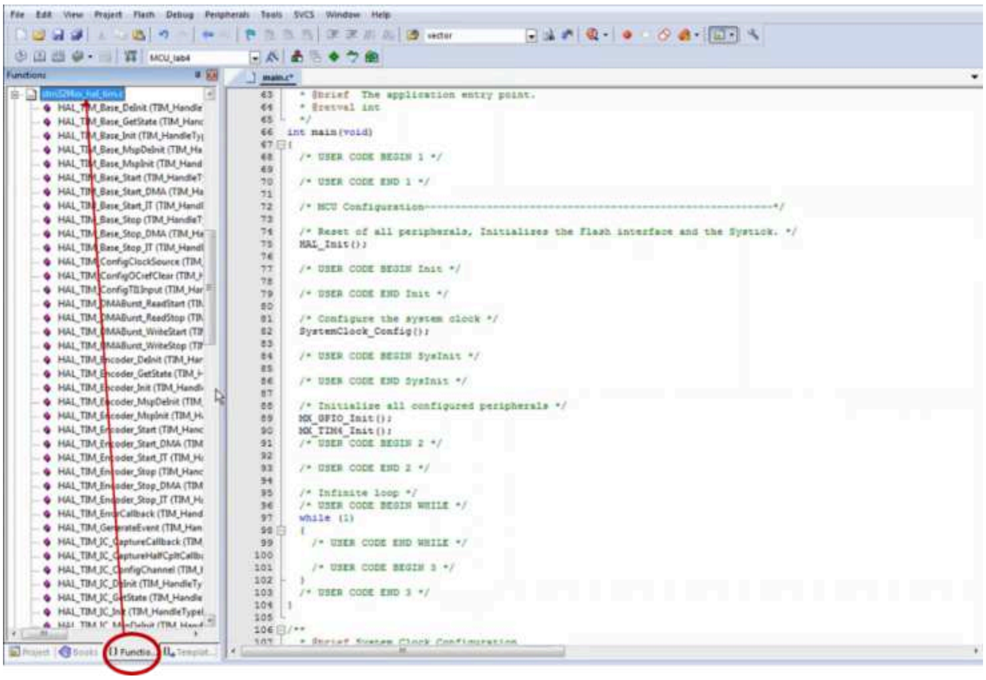


Figure 16

```
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM4_Init();
  /* USER CODE BEGIN 2 */
HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_4); //Starts the PWM signal generation.
__HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_4,500);//In 500 of the Counter Period it is on mode, while the remaining 1500 it is off.
  /* USER CODE END 2 */

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

Figure 17

10. Use CCR (Capture Compare Register) directly to generate PWM signals. This PWM signal is ON mode in 100 of the Counter Period and OFF mode in the remaining 1900. Change the value of ON and OFF mode as 1500, 1000 respectively (Changing duty cycle). Observe the conditions of the LED connected to the PD15 pin.

```

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM4_Init();
  /* USER CODE BEGIN 2 */
HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_4); //Starts the PWM signal generation.
TIM4->CCR4=100; //In 100 of the Counter Period it is ON mode, while the remaining 1500 it is OFF.
  /* USER CODE END 2 */

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

Figure 18

11. Now, we continue with input capture mode. First, you close the Keil µVision and go back to the CubeMx and change some configurations. Use TIM2 in input capture mode and Channel 1 (which is connected to PA0 pin) of the TIM2 is used in input capture direct mode. So, use a cable to connect PA0 with PD15 pins. Set prescaler value is 41999, counter mode is up, counter period is 1999 for TIM2 (Figure 19). Go to NVIC settings and enable the interrupt for TIM2 (Figure 20). Go to System Core and set the Preemption Priority as 1 for TIM2 (Figure 21). Then click Generate Code and Open Project in Keil µVision.

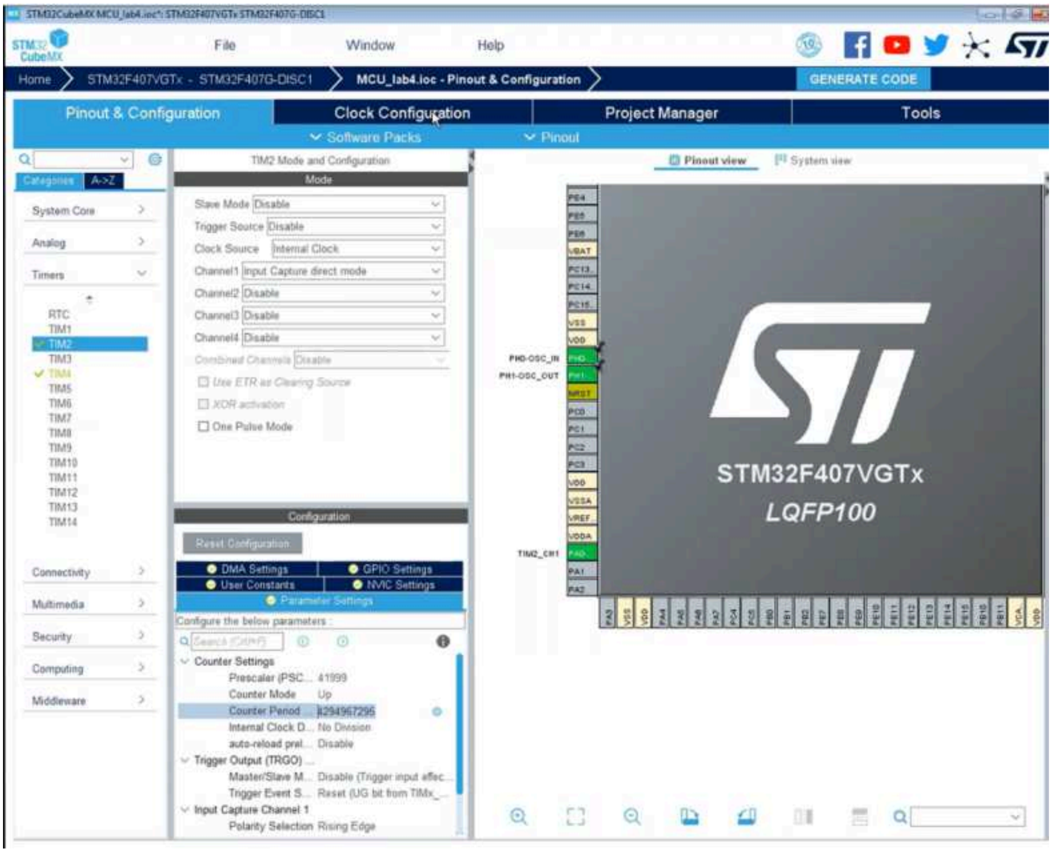


Figure 19

Figure 20

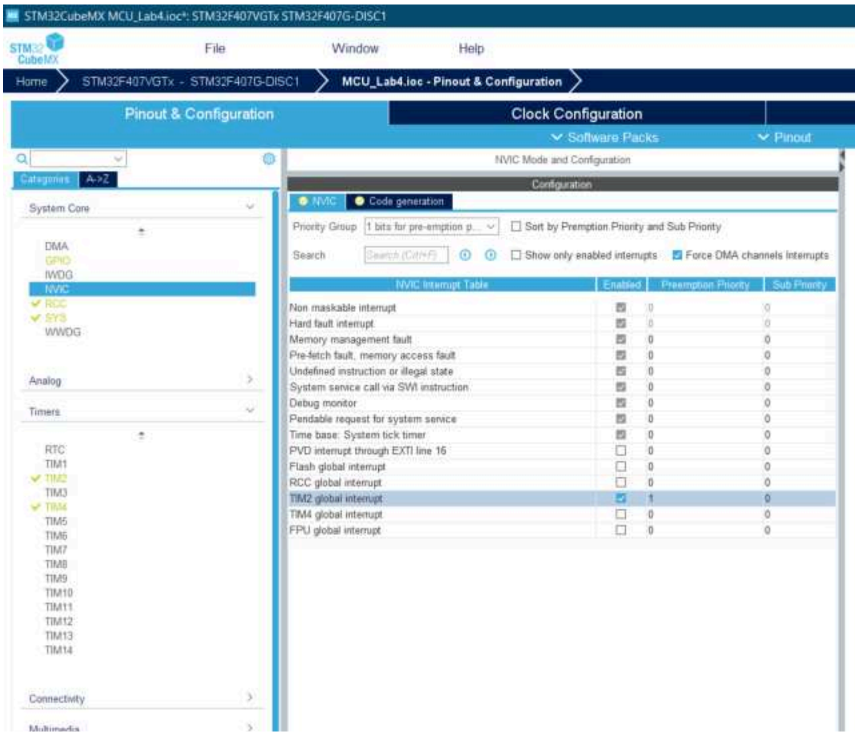


Figure 21

12. Start the input capture mode for TIM2 using HAL_TIM_IC_Start_IT function with PWM signal like in Figure 22. So, you can capture the generated PWM signal frequency. You will use the TIM2 interrupt mode. Go to the stm32fxx.c file and write the codes like in Figure23 in this file. Build and load the code. Observe a and b value using debug. Find 'Capture value' (CNT1-CNT2) and calculate applied signal period according to this value multiplying period value for each value (Figure 24).

```
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM4_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */
HAL_TIM_IC_Start_IT(&htim2,TIM_CHANNEL_1);//Start the input capture mode for TIM2
HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_4); //Starts the PWM signal generation.
//__HAL_TIM_SetCompare(&htim4,TIM_CHANNEL_4,100);//In 100 of the Counter Period it is on mode, while the remaining 15
TIM4->CCR4=100; //In 100 of the Counter Period it is ON mode, while the remaining 1500 it is OFF.
  /* USER CODE END 2 */

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

Figure 22

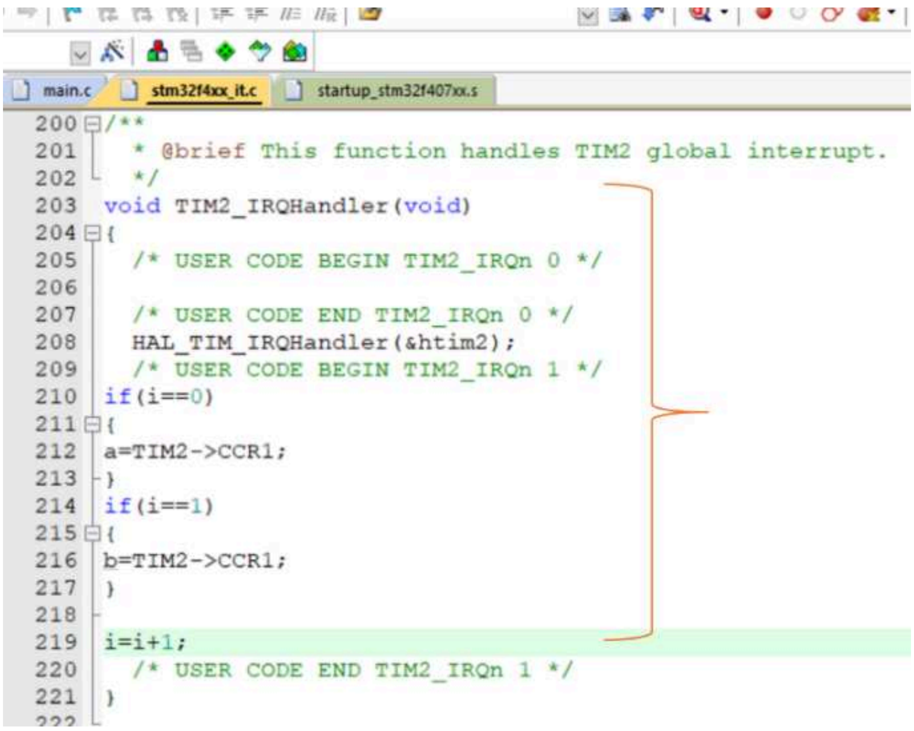


Figure 23

Input capture mode

$$Period = Capture.\left(\frac{TIMx\_CLK}{(Prescaler+1)(CH_{Prescaler})(Polarity_{Index})}\right)^{-1}$$

$$Capture = CNT_1 - CNT_0 \ if \ CNT_1 > CNT_0$$

$$Capture = TIMx_{Period} - CNT_0 + CNT_1 \ if \ CNT_1 < CNT_0$$

Figure 24

EXPERIMENT 6: USART PERIPHERALS

Objectives

The objectives of Experiment 6 is

- to learn how to use Universal Synchrous / Asynchrous Serial Communnications peripherals

Apparatus Required:

- STM32CubeMx
- Keil µVision (MDK ARM)
- STM32 ST-Link Utility
- STM32F4 Microcontroller
- **2 Jumper Cables (female-female)**

Preliminary Work:

1. Study the USART (L08) notes.
2. Write the codes of the experimental work in Keil µVision.

Experimental Work:

1. Create a new project in CubeMx (Figure 1). Select STMF407VGTx, then STMF407G-DISC1 and finally Start Project (Figure 2). First adjust the Pinout&Configuration settings. Close the unnecessary pins (Figure 3).

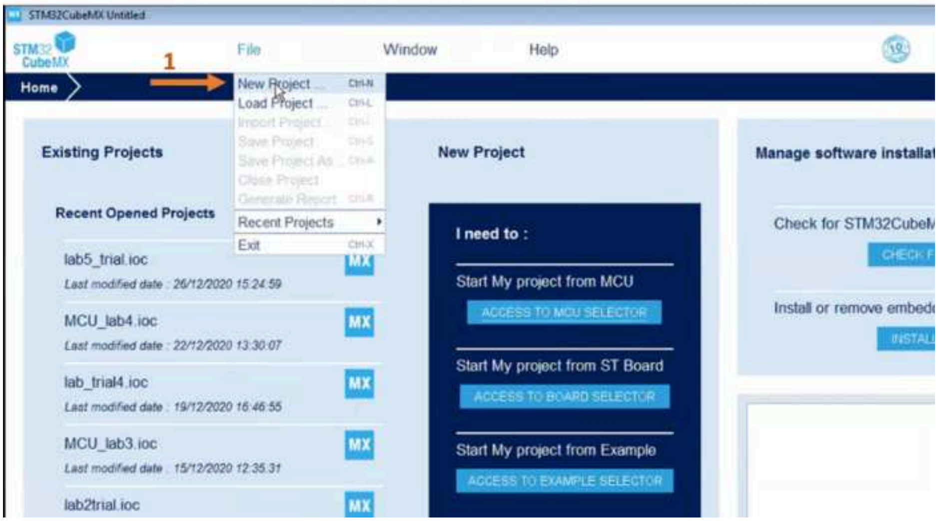


Figure 1

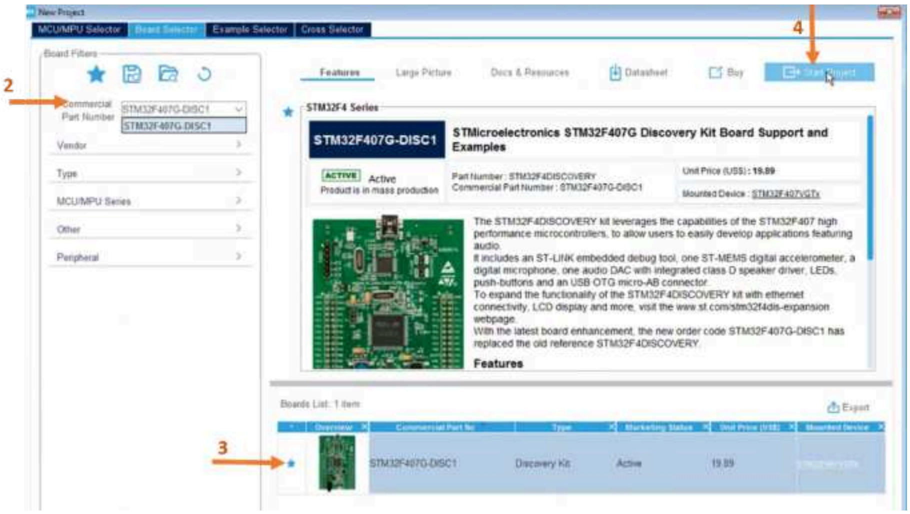


Figure 2

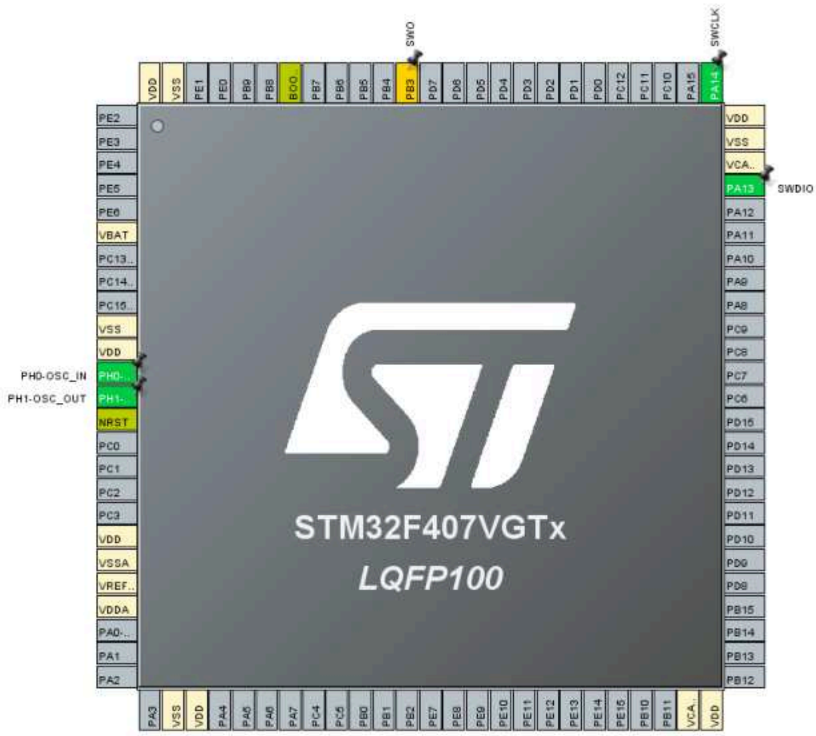


Figure 3

2. We use USART1 and USART2 peripherals in the lecture. Set the configuration for USART1 (Figure 4) and USART 2 (Figure 5). Use USARTs in Asynchronous Mode. BoudRate is 9600 Bits/s, Word Length is 8 Bits, Parity is None, Stop Bit is 1, Data Direction is Receive and Transmit, Over Sampling is 16 Samples for both of the USARTs. Select the PA0 pin as GPIO_EXTI0. Go to GPIO settings and set the GPIO configurations for the PA0 pin as in Figure 6. Set the NVIC configurations as in Figure 7.

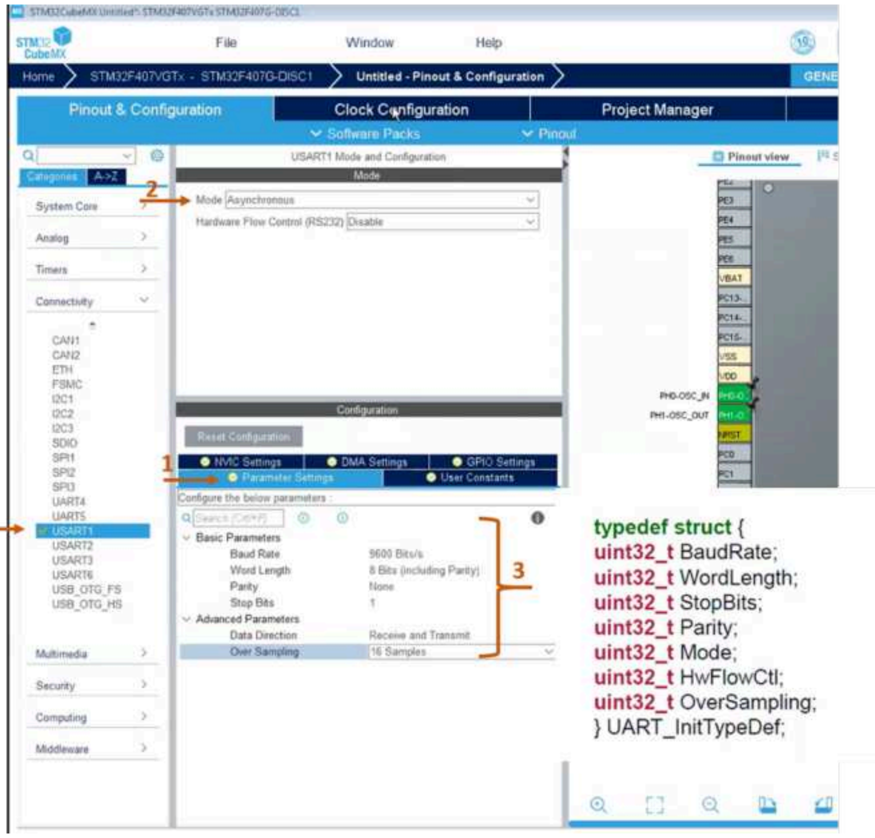


Figure 4

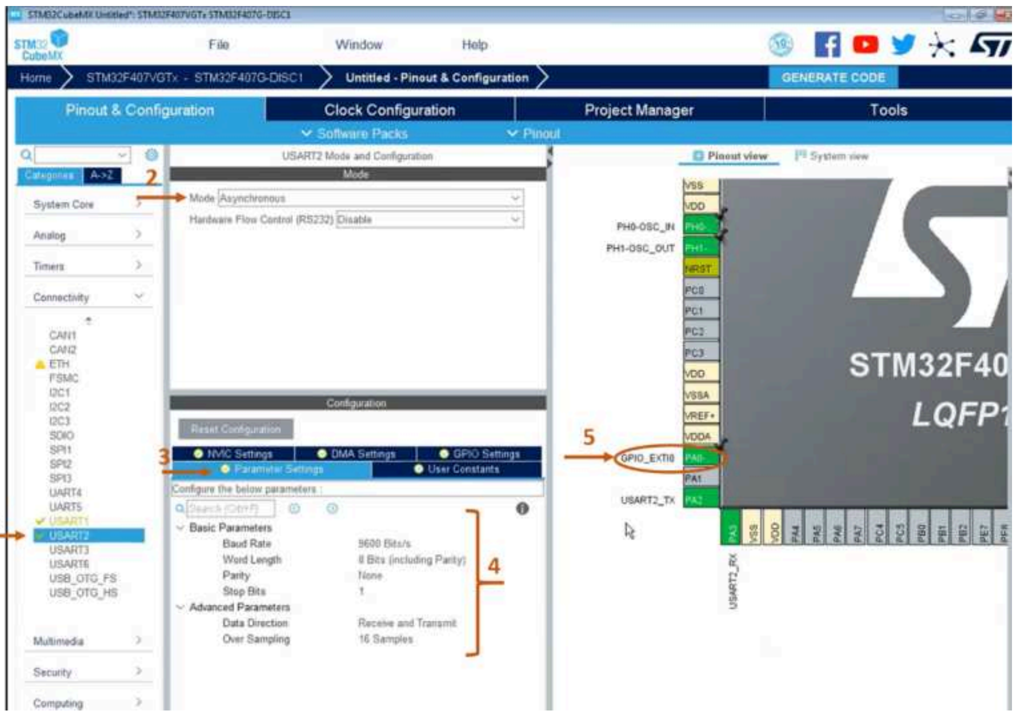


Figure 5

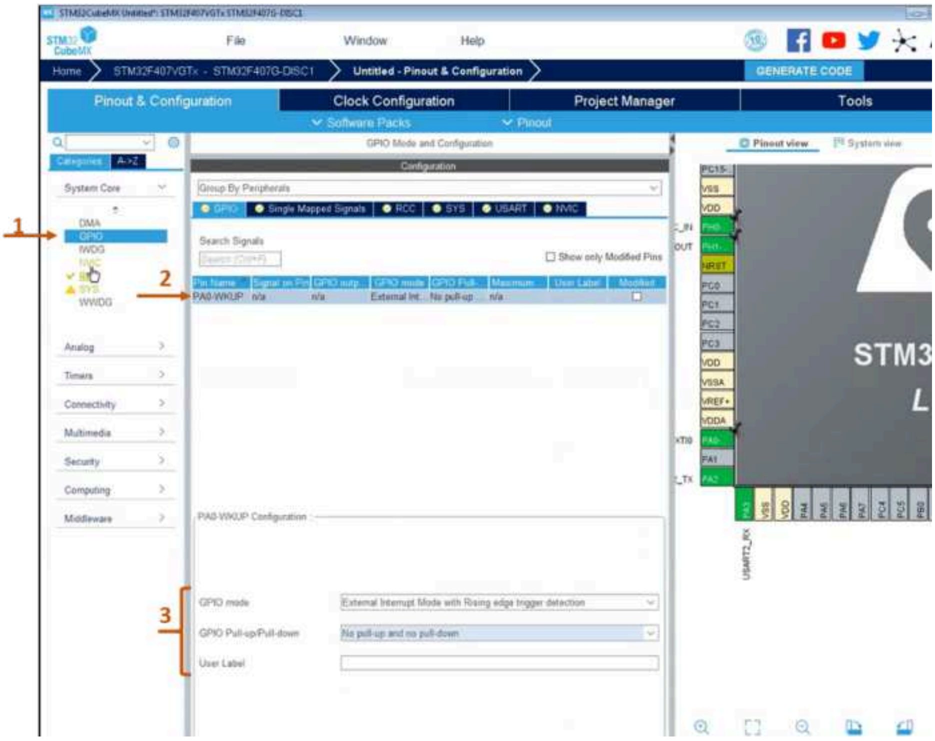


Figure 6

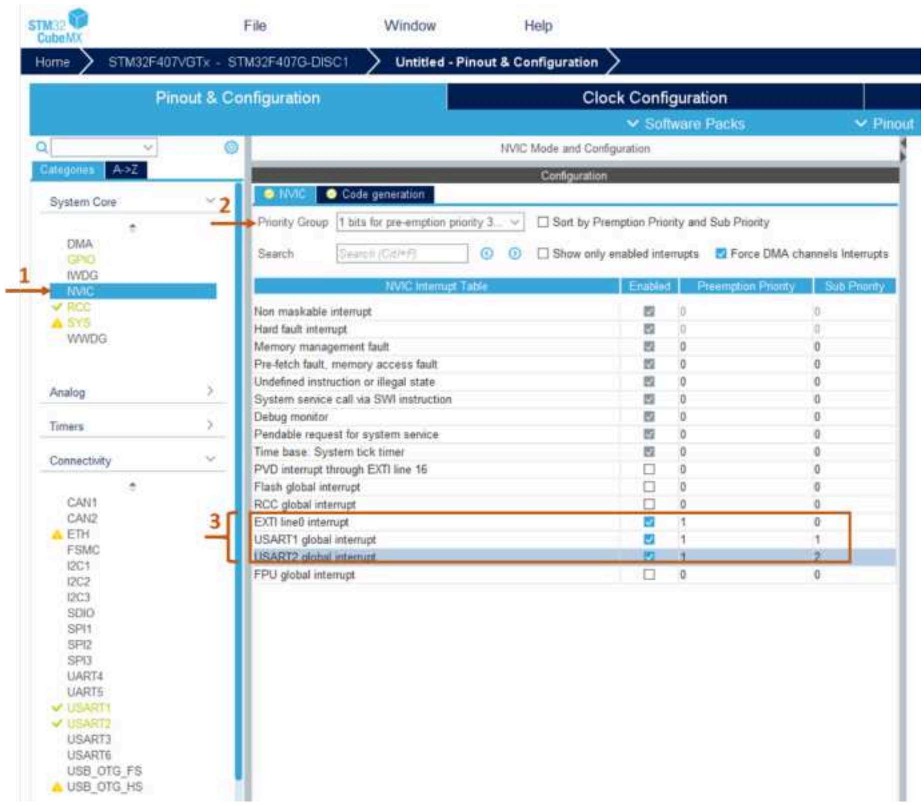


Figure 7

3. There is no hardware connection between the USART. Use the cables to connect the USART peripherals to each other. (Connect PA2 with PA10; Connect PA3 with PA9 using cables). Now, you can set the project manager configurations as in Figure 8 and go to the Keil µVision by selecting Generate Code.

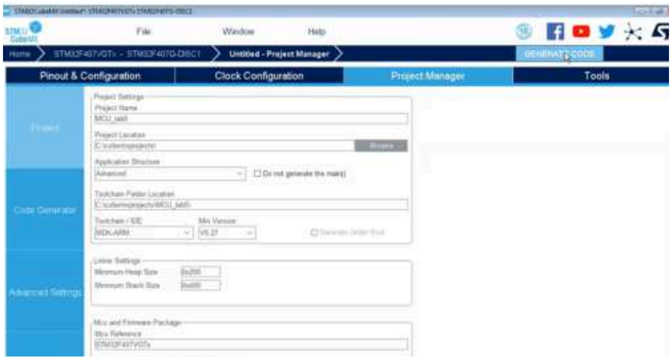


Figure 8

4. Go to main.c file and build the codes. Examine the main.c file to observe the configurations which are already done in CubeMx. You should write the codes in interrupt mode. Open the stm32fxx_it.c interrupt file.

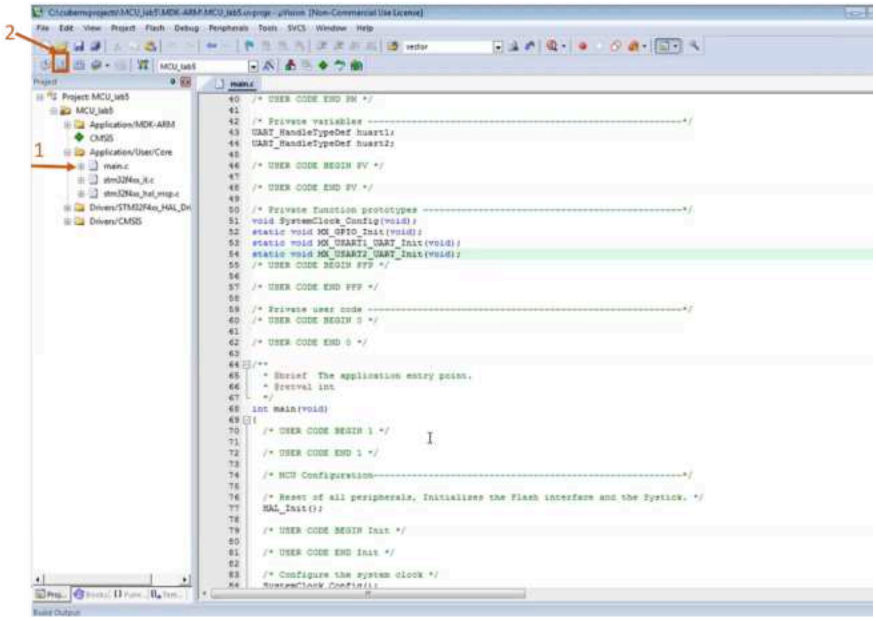


Figure 9

5. When the button is pushed, transmit data from UART2 and write it into the transmit buffer. Then go to UART2 and receive data from UART1. Write the data into the receive buffer.
   You should write the codes in interrupt mode. Open the stm32fxx_it.c interrupt file and write the codes in that file (Figure 12). First, define the transmit and receive buffer which will keep the transmitted and received data (Figure 10) . Use HAL_UART_Transmit_IT function in EXTI0_IRQHandler function and HAL_UART_Receive_IT function in USART2_IRQHandler function which is given

in Figure 10. Finally, build and Load the codes. Follow the change of variables using the Debug menu (Figure 13).

```c
/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
char transmit[]="hello";
char receive [6]; // 6:sizeof(hello)+1 null bit
/* USER CODE END PV */
```

Figure 10

UART Communication in Interrupt Mode

To transmit a sequence of bytes in interrupt mode, the HAL defines the function:
HAL_UART_Transmit_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);

To receive sequence of bytes in interrupt mode, the HAL defines the function:
HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);

Figure 11

main.c | stm32f4xx_it.c*

```c
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */

  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0); //reset the IRQ
  HAL_UART_Transmit_IT(&huart2, (uint8_t*)transmit, sizeof(transmit)); //Transmit the transmit variable from USART2 when we push the button.
  /* USER CODE BEGIN EXTI0_IRQn 1 */

  /* USER CODE END EXTI0_IRQn 1 */
}
/**
  * @brief This function handles USART1 global interrupt.
  */
void USART1_IRQHandler(void)
{
  /* USER CODE BEGIN USART1_IRQn 0 */

  /* USER CODE END USART1_IRQn 0 */
  HAL_UART_IRQHandler(&huart1);
  /* USER CODE BEGIN USART1_IRQn 1 */

  /* USER CODE END USART1_IRQn 1 */
}

/**
  * @brief This function handles USART2 global interrupt.
  */
void USART2_IRQHandler(void)
{
  /* USER CODE BEGIN USART2_IRQn 0 */

  /* USER CODE END USART2_IRQn 0 */
  HAL_UART_IRQHandler(&huart2);
  /* USER CODE BEGIN USART2_IRQn 1 */
HAL_UART_Receive_IT(&huart1, (uint8_t*)receive,sizeof(receive));//Receive the data from UART1
  /* USER CODE END USART2_IRQn 1 */
}
```

Figure 12

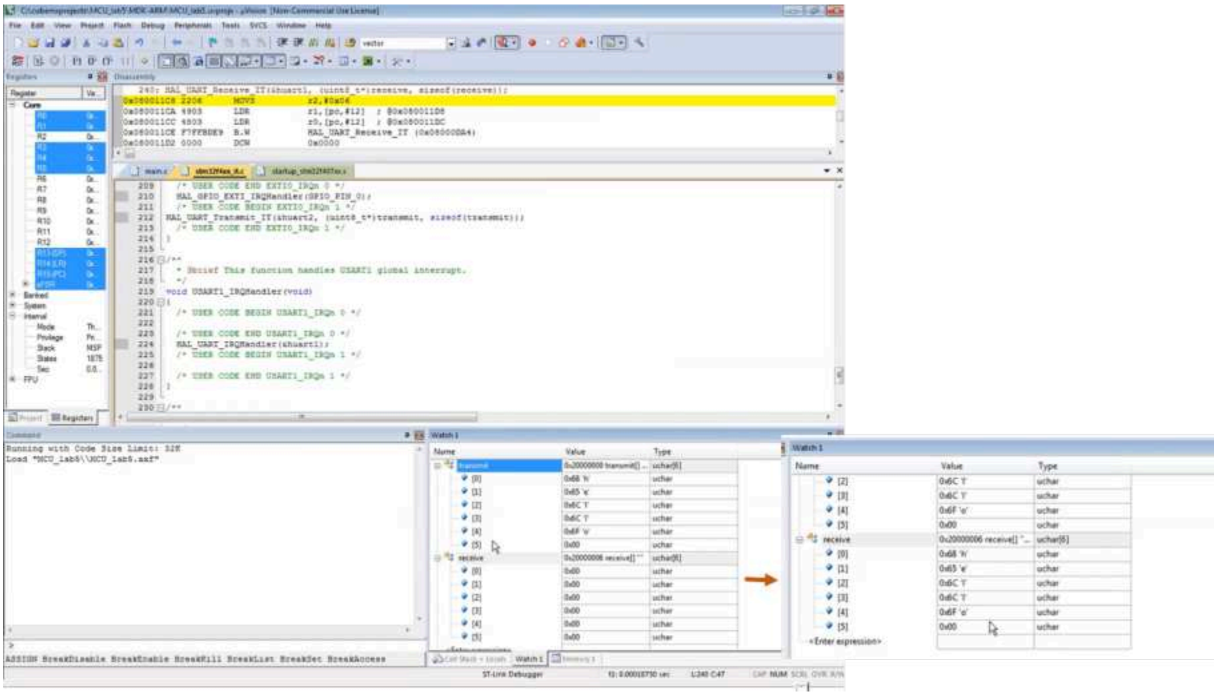


Figure 13

6. You will make the changes within the EXTI0_IRQHandler function. Define tansmit1, transmit2, receive and i variables. Assign “hello” to transmit1 and “world” to transmit2 in char type (Figure 14). Each time the button is pushed, the value of the i variable increases by 1. If it is an even number, transmit1 variable, if odd, transmit 2 variable from USART2 (Figure 15). Let USART2 also receive this data from USART1. Follow the change of variables using the Debug menu (Figure 16).

```
/* Private variables ----------
/* USER CODE BEGIN PV */
int i;
char transmit1[]="hello";
char transmit2[]="world";
char receive [6];
/* USER CODE END PV */
```

Figure 14

```
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */

  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0); //reset the IRQ
  HAL_Delay(100);
  i=i+1;
  if (i%2==0)
  {
  HAL_UART_Transmit_IT(&huart2, (uint8_t*)transmit1, sizeof(transmit1)); //Transmit the transmit1 variable from USART2 when we push the button.
  }
  else
  {
  HAL_UART_Transmit_IT(&huart2, (uint8_t*)transmit2, sizeof(transmit2)); //Transmit the transmit2 variable from USART2 when we push the button.
  }
  /* USER CODE BEGIN EXTI0_IRQn 1 */
```

Figure 15

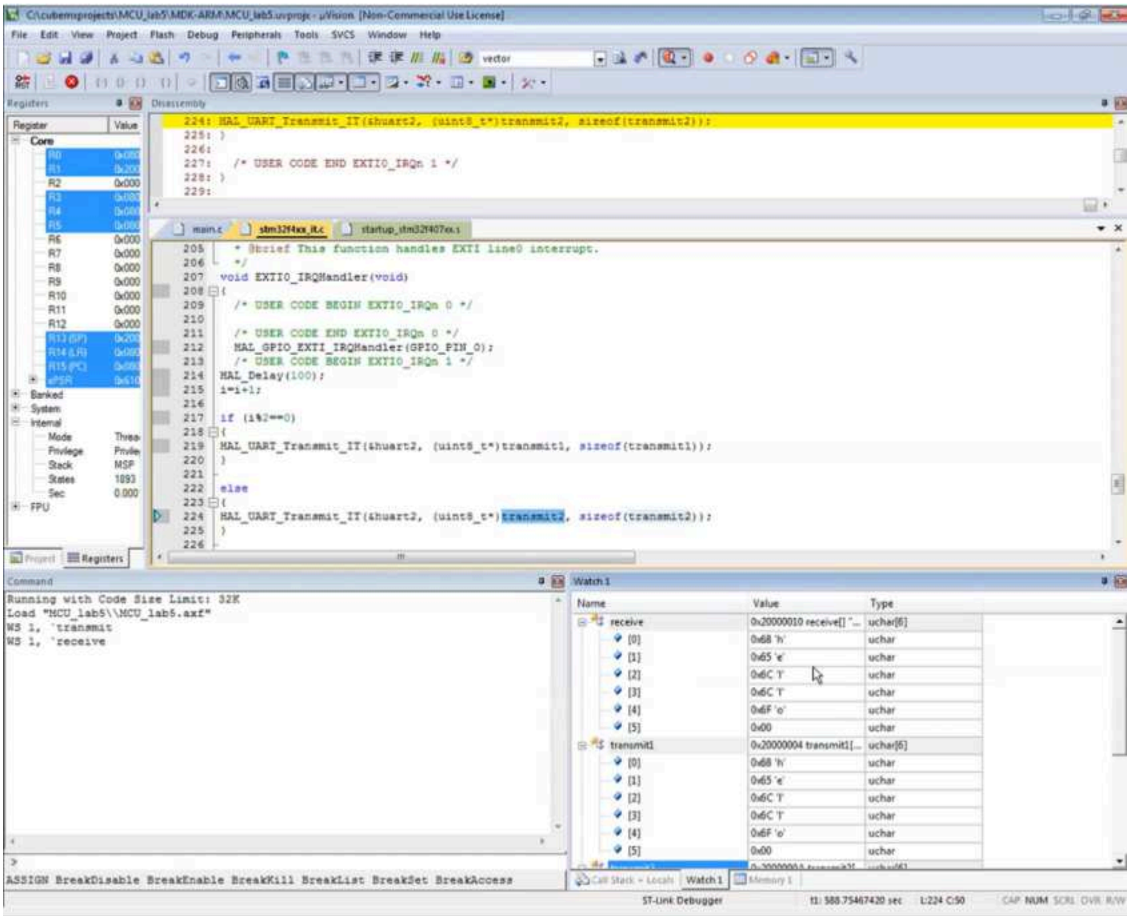


Figure 16