

# EXPERIMENT--03-SIMULATION-OF-PUSHBUTTON-AND-LED INTERFACE WITH ARM CONTROLLER AND PROTEUS

**Aim:** To Interface a Digital output (LED) and Digital input (Pushbutton) to ARM development board , and simulate it in Proteus

**Components required:** STM32 CUBE IDE, Proteus 8 simulator .

## Theory

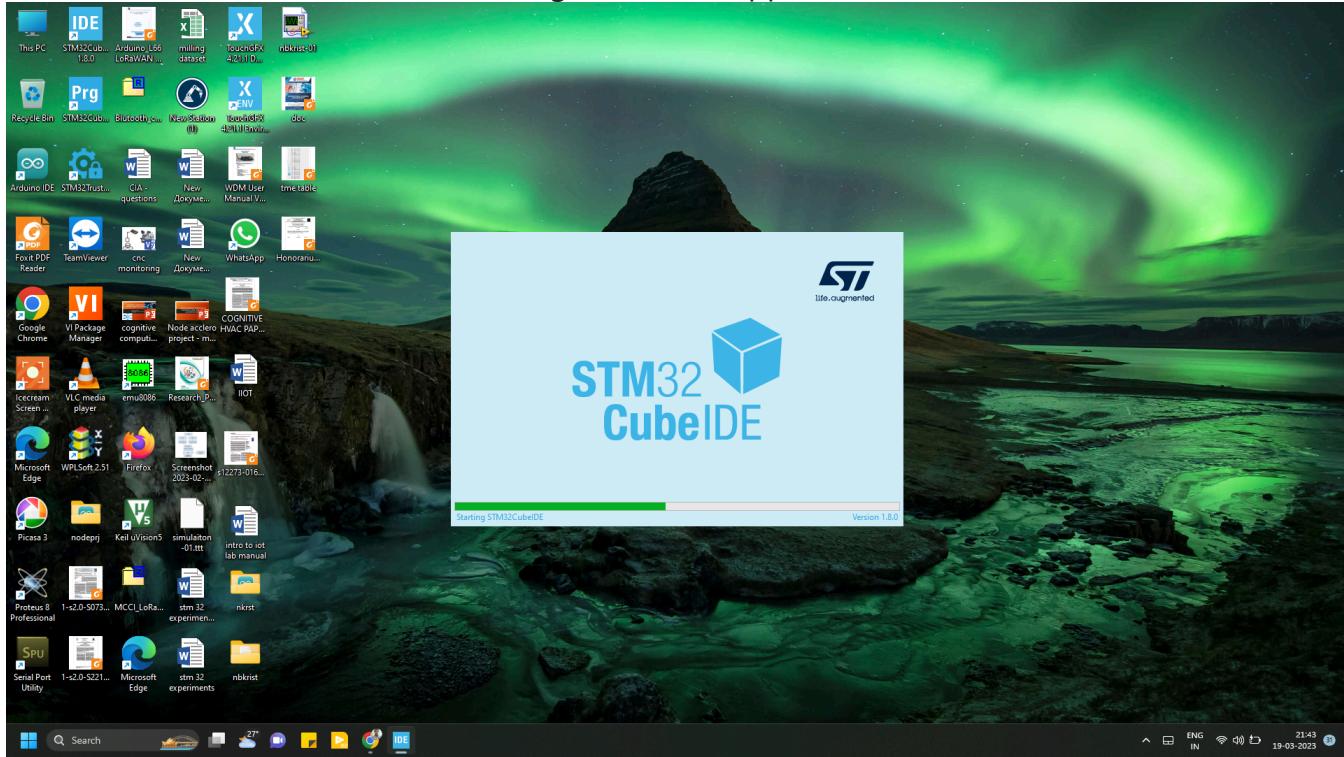
The full form of an ARM is an advanced reduced instruction set computer (RISC) machine, and it is a 32-bit processor architecture expanded by ARM holdings. The applications of an ARM processor include several microcontrollers as well as processors. The architecture of an ARM processor was licensed by many corporations for designing ARM processor-based SoC products and CPUs. This allows the corporations to manufacture their products using ARM architecture. Likewise, all main semiconductor companies will make ARM-based SOCs such as Samsung, Atmel, TI etc.

What is an ARM7 Processor? ARM7 processor is commonly used in embedded system applications. Also, it is a balance among classic as well as new-Cortex sequence. This processor is tremendous in finding the resources existing on the internet with excellence documentation offered by NXP Semiconductors. It suits completely for an apprentice to obtain in detail hardware & software design implementation.

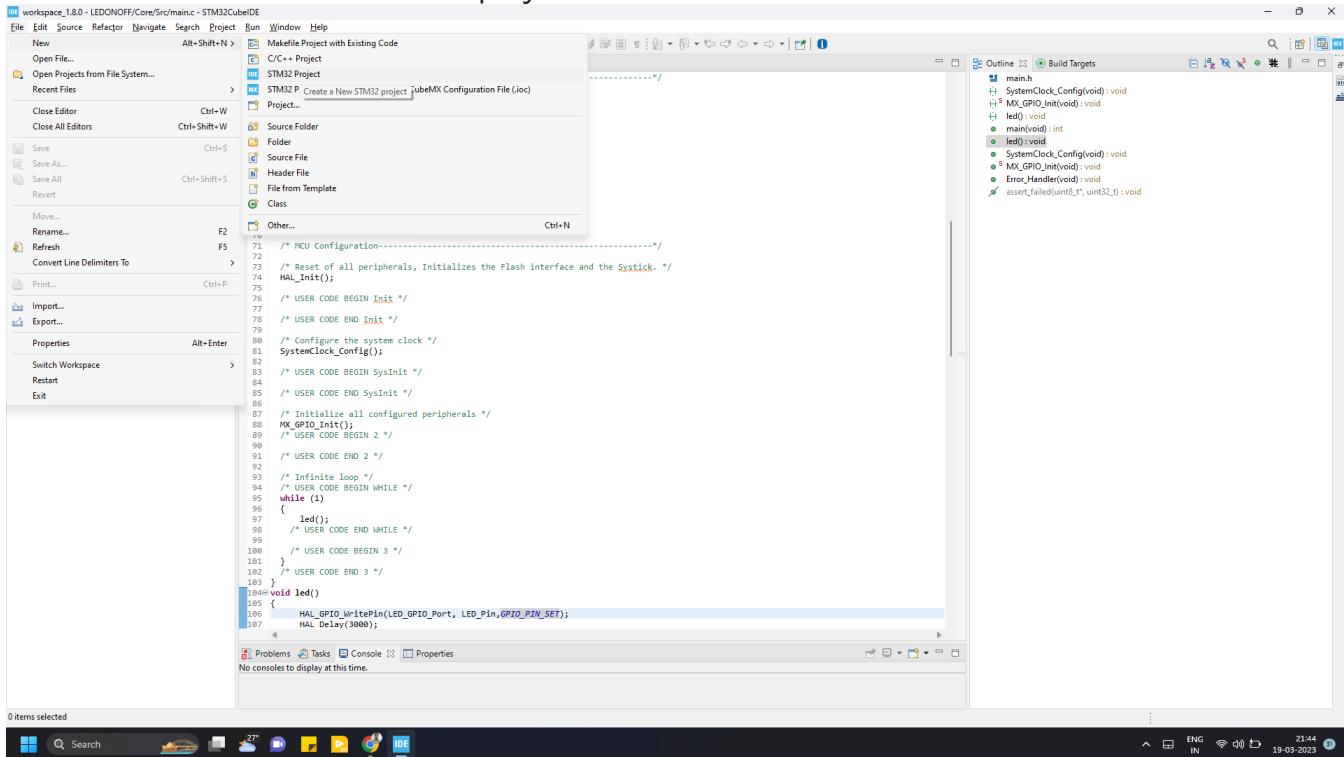
STM32F401xB STM32F401xC ARM® Cortex® -M4 32b MCU+FPU, 105 DMIPS, 256KB Flash/64KB RAM, 11 TIMs, 1 ADC, 11 comm. interfaces Datasheet - production data Features • Core: ARM® 32-bit Cortex® -M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from Flash memory, frequency up to 84 MHz, memory protection unit, 105 DMIPS/ 1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions • Memories – Up to 256 Kbytes of Flash memory – Up to 64 Kbytes of SRAM

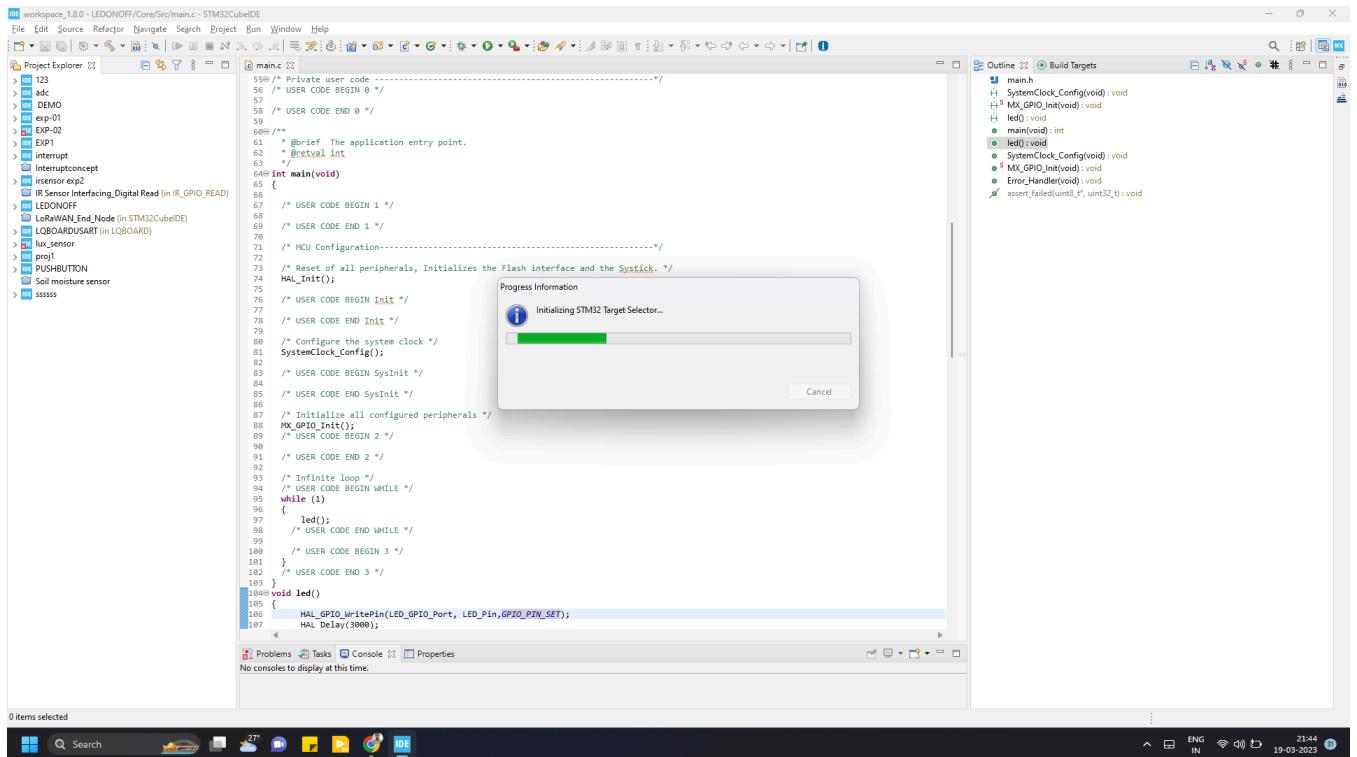
## Procedure:

## 1. click on STM 32 CUBE IDE, the following screen will appear



## 2. click on FILE, click on new stm 32 project





### 3. select the target to be programmed as shown below and click on next

The screenshot shows the STM32CubeMX Target Selection screen. The STM32G071RB is selected as the target. The main panel displays the STM32G0 Series, featuring the STM32G071RB with the following specifications:

- Mainstream Arm Cortex-M0+ MCU with 128 Kbytes of Flash memory memory, 36 Kbytes RAM, 64 MHz CPU, 4x USART, timers, ADC, DAC, comm. I/F, 1.7-3.6V**
- ACTIVE** Active
- Unit Price for 10kU (US\$): 1.803**
- Boards: NUCLEO-G071RB - STM32G071B-DISCO**
- LQFP64**

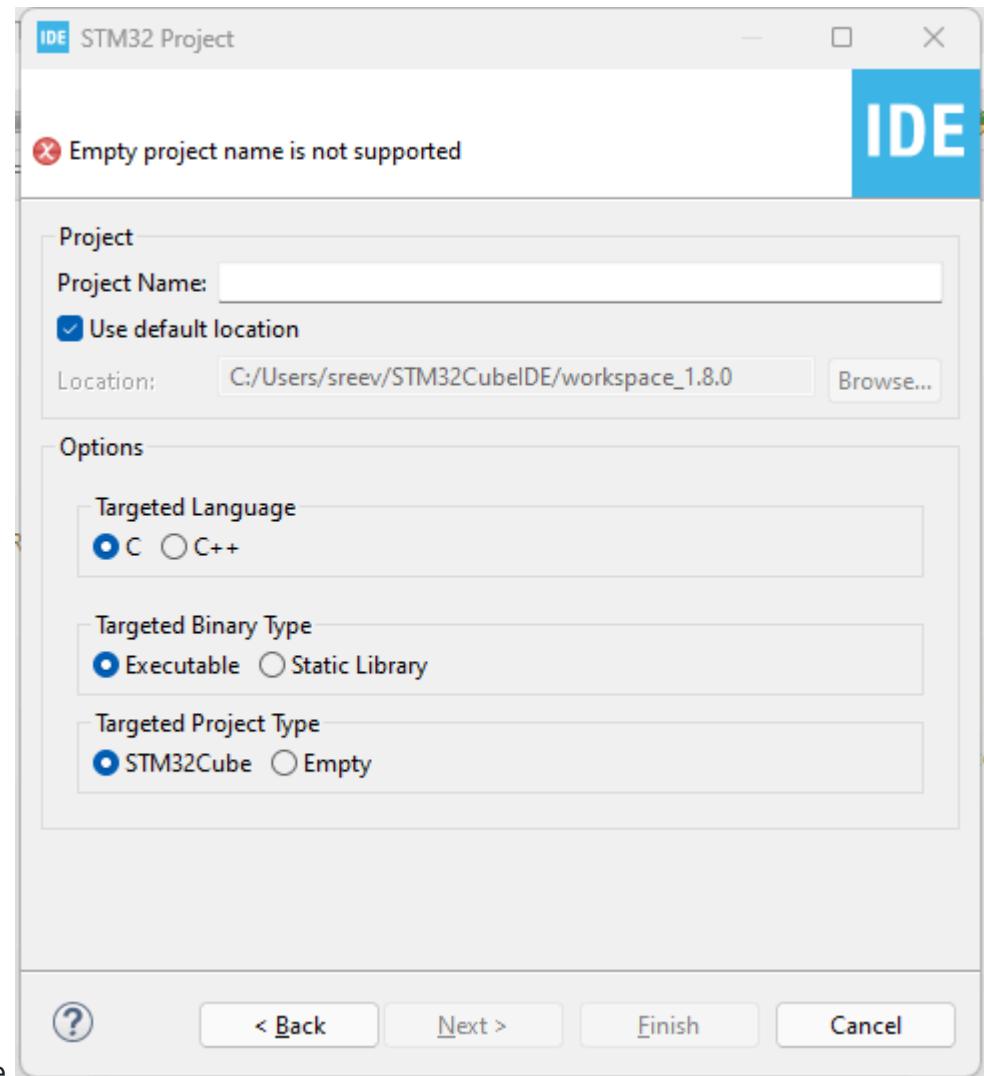
The description notes that the STM32G071RB mainstream microcontrollers are based on high-performance Arm® Cortex®-M0+ 32-bit RISC core operating at up to 64 MHz frequency. They are suitable for a wide range of applications in consumer, industrial and appliance domains and ready for the Internet of Things (IoT) solutions. The devices incorporate a memory protection unit (MPU), high-speed embedded memories (up to 128 Kbytes of Flash program memory with read protection, write protection, proprietary code protection, and securable area, and 36 Kbytes of SRAM), DMA and an extensive range of system functions, enhanced I/Os and peripherals. The devices offer standard communication interfaces (two I²Cs, two SPIs / one I²S, one HDMI CEI, and four USARTs), one 12-bit ADC (2.5 MSPs) with up to 19 channels, one 12-bit DAC with two channels, two fast comparators, an internal voltage reference buffer, a low-power RTC, an advanced control PWM timer running at up to double the CPU frequency, five general-purpose 16-bit timers with one running at up to double the CPU frequency, a 32-bit general-purpose timer, two basic timers, two low-power 16-bit timers, two watchdog timers, and a SysTick timer. The STM32G071RB devices provide a fully integrated USB Type-C Power Delivery controller. The devices operate within ambient temperatures from -40 to 125°C. They can operate with supply voltages from 1.7 V to 3.6 V. Optimized dynamic consumption combined with a comprehensive set of power-saving modes, low-power timers and low-power UART, allows the design of low-power applications.

The MCU/MPU selector sidebar shows the following filters and lists:

- MCU/MPU Filters**: Part Number (STM32G071RB), Core (Arm Cortex-A7 + Arm Cortex-M4, Arm Cortex-M0, Arm Cortex-M0+, Arm Cortex-M3, Arm Cortex-M4, Arm Cortex-M4 + Arm Cortex-M0+, Arm Cortex-M7, Arm Cortex-M7 + Arm Cortex-M4, Arm Cortex-M33).
- Series**: Check/Uncheck All (STM32F0, STM32F1, STM32F2, STM32F3, STM32F4, STM32F7, STM32G0, STM32G4, STM32H7, STM32L0).

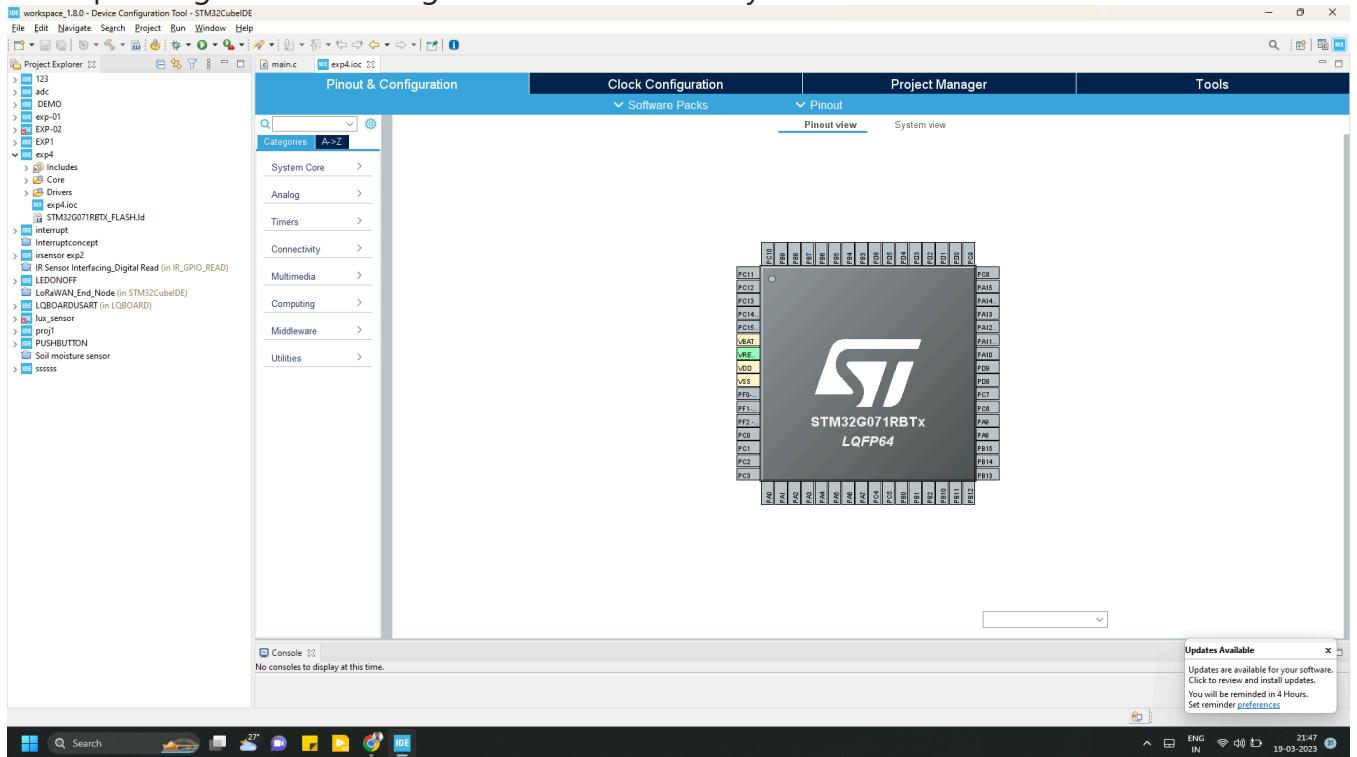
The MCU/MPU List table shows 2 items:

Part No.	Reference	Marketing Status	Unit Price for 10kU (US\$)	Board	Package	Flash	RAM	IO	Freq.
STM32G071RB	STM32G071...	Active	1.803	NUCLEO-09_STM32G071	LQFP64	128 kBBytes	36 kBBytes	60	64 MHz
STM32G071...	STM32G071...	Active	1.803	NUCLEO-09_STM32G071	LQFP64	128 kBBytes	36 kBBytes	60	64 MHz

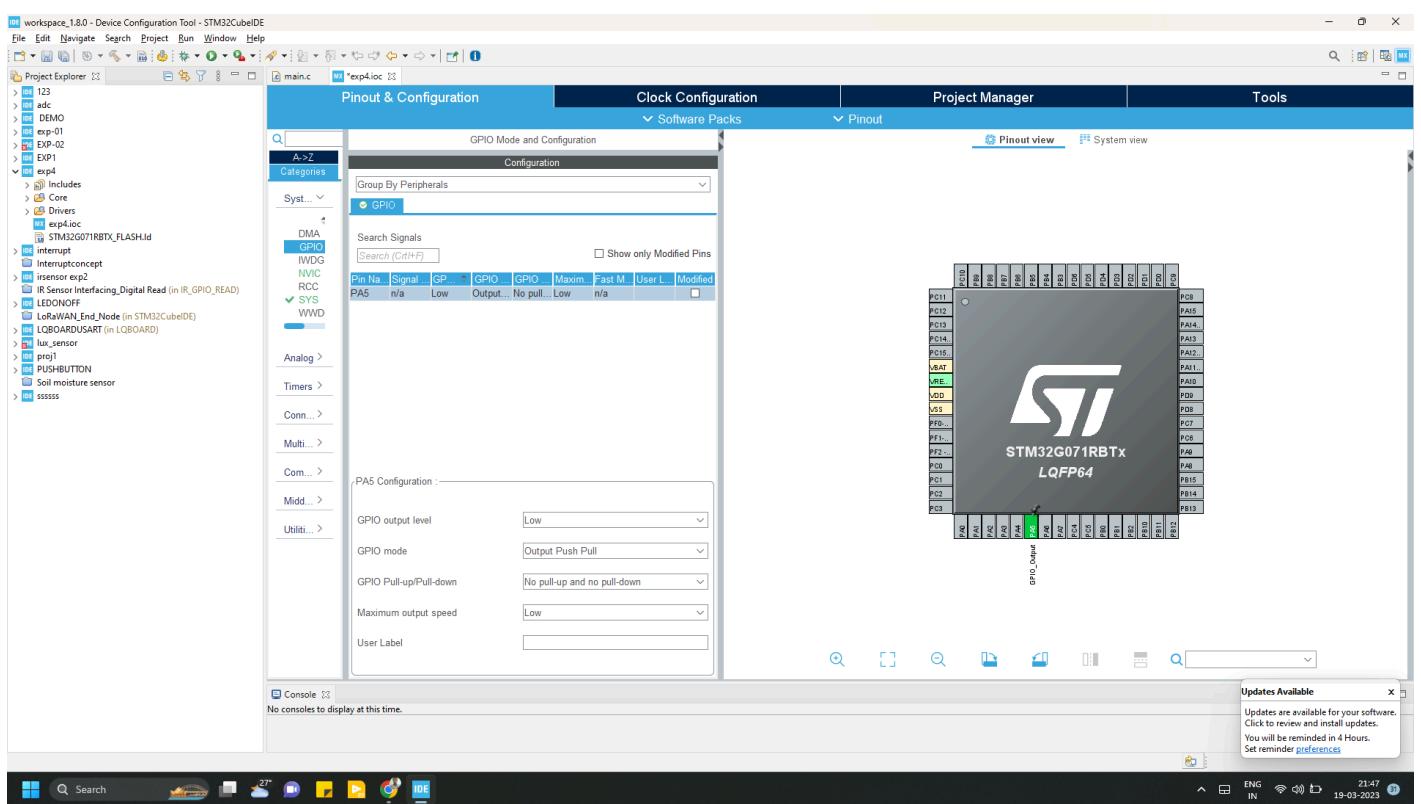
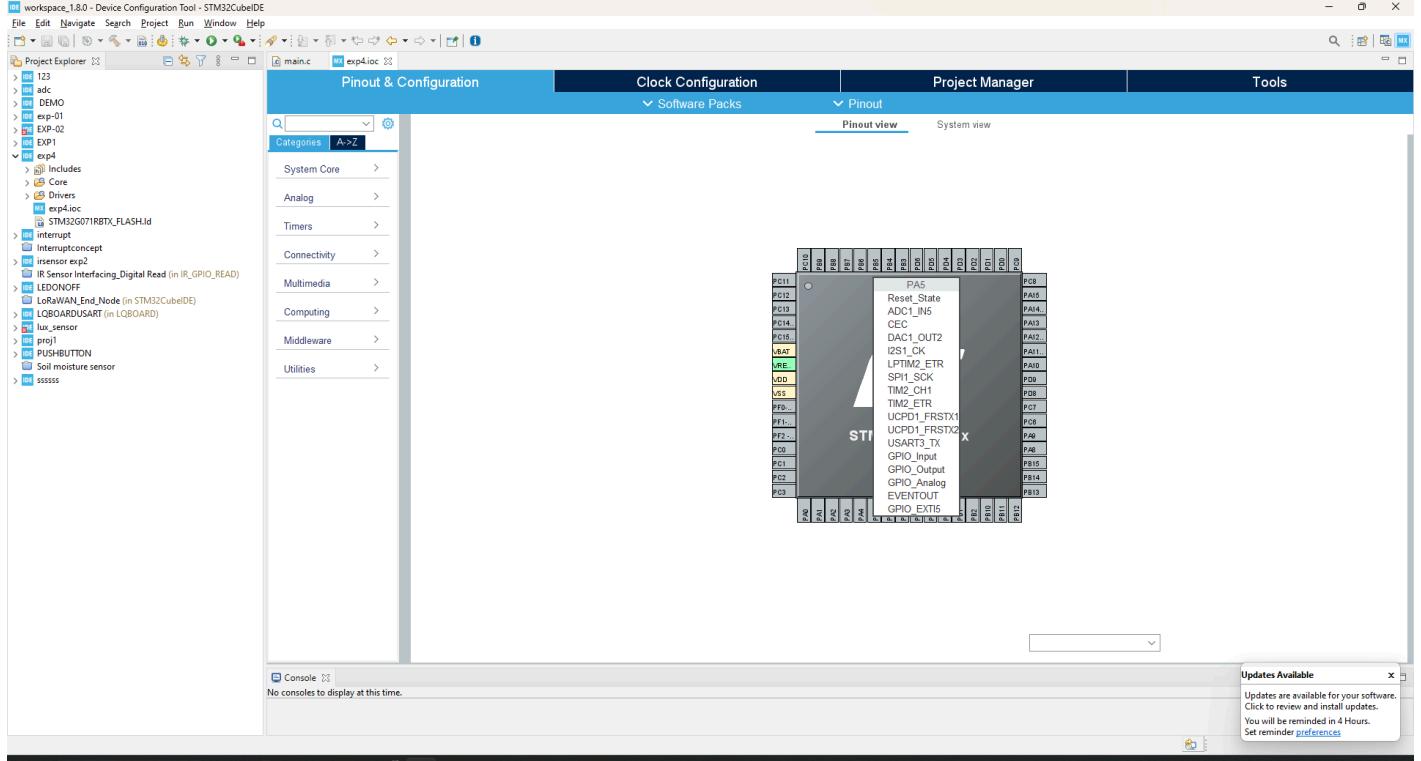


4.select the program name

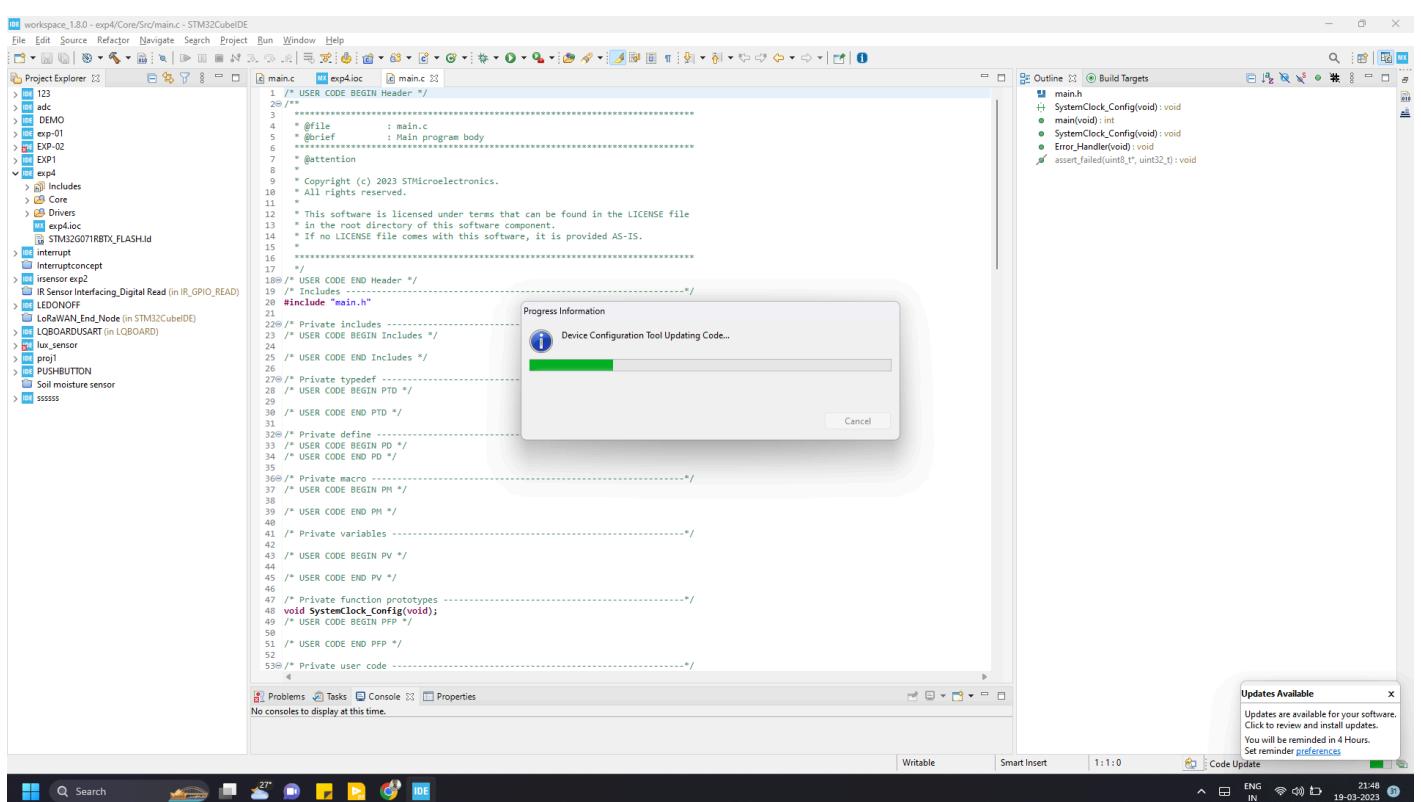
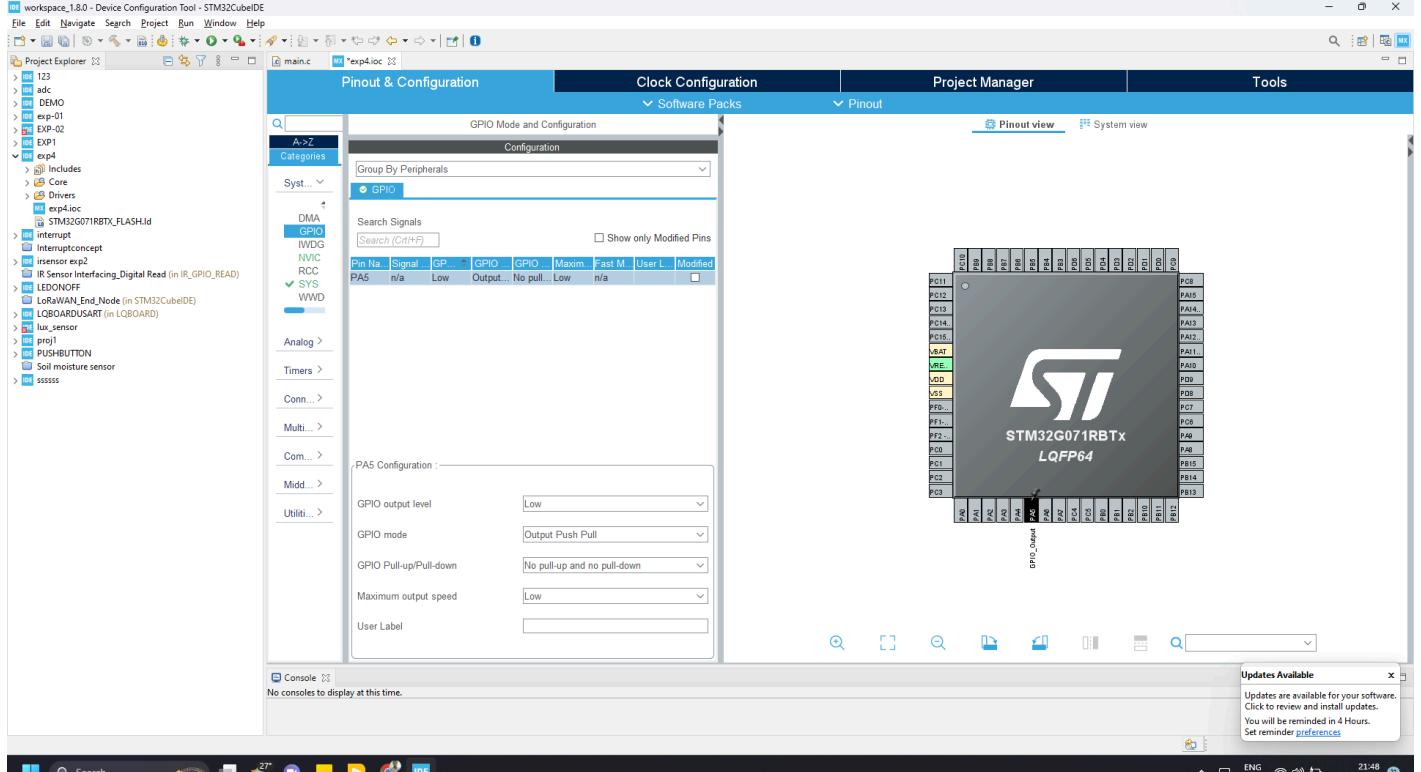
5. corresponding ioc file will be generated automatically



## 6.select the appropriate pins as gipo, in or out, USART or required options and configure



## 7.click on cntrl+S , automaticall C program will be generated



## 8. edit the program and as per required

```

1 /* USER CODE BEGIN Header */
2 /**
3  * @file     : main.c
4  * @brief    : Main program body
5  *
6  * @attention
7  *
8  * Copyright (c) 2023 STMicroelectronics.
9  * All rights reserved.
10 *
11 * This software is licensed under terms that can be found in the LICENSE file
12 * in the root directory of this software component.
13 * If no LICENSE file comes with this software, it is provided AS-IS.
14 */
15
16
17 /**
18 * USER CODE END Header
19 */
20 /* Includes
21 */
22 /* Private includes
23 */
24 /* Private typedef
25 */
26 /* Private variables
27 */
28 /* USER CODE BEGIN PTD */
29
30 /* USER CODE END PTD */
31
32 /* Private define
33 */
34 /* USER CODE BEGIN PD */
35
36 /* Private macro
37 */
38 /* USER CODE BEGIN PM */
39
40 /* USER CODE END PM */
41
42 /* Private variables
43 */
44 /* USER CODE BEGIN PV */
45
46 /* USER CODE END PV */
47
48 /* Private function prototypes
49 */
50 static void MX_GPIO_Init(void);
51
52 /* USER CODE END PPP */
53

```

## 9. use project and build

```

22 /* Private includes
23 */
24 /* USER CODE BEGIN Includes */
25
26 /* Private typedef
27 */
28 /* USER CODE BEGIN PTD */
29
30 /* USER CODE END PTD */
31
32 /* Private define
33 */
34 /* USER CODE BEGIN PD */
35
36 /* Private macro
37 */
38 /* USER CODE BEGIN PM */
39
40 /* USER CODE END PM */
41
42 /* Private variables
43 */
44 /* USER CODE BEGIN PV */
45
46 /* USER CODE END PV */
47
48 /* Private function prototypes
49 */
50 void SystemClock_Config(void);
51 static void MX_GPIO_Init(void);
52

```

CDT Build Console [sssss]

```

21:50:34 **** Build of configuration Debug for project sssss ****
make -j8 all
arm-none-eabi-size sssss.elf
text      858  dec    hex filename
20524   120  1848  22892  596 sssss.elf
Finished building: default.size.stdout

21:50:34 Build Finished. 0 errors, 0 warnings. (took 232ms)

```

## 10. once the project is bulild

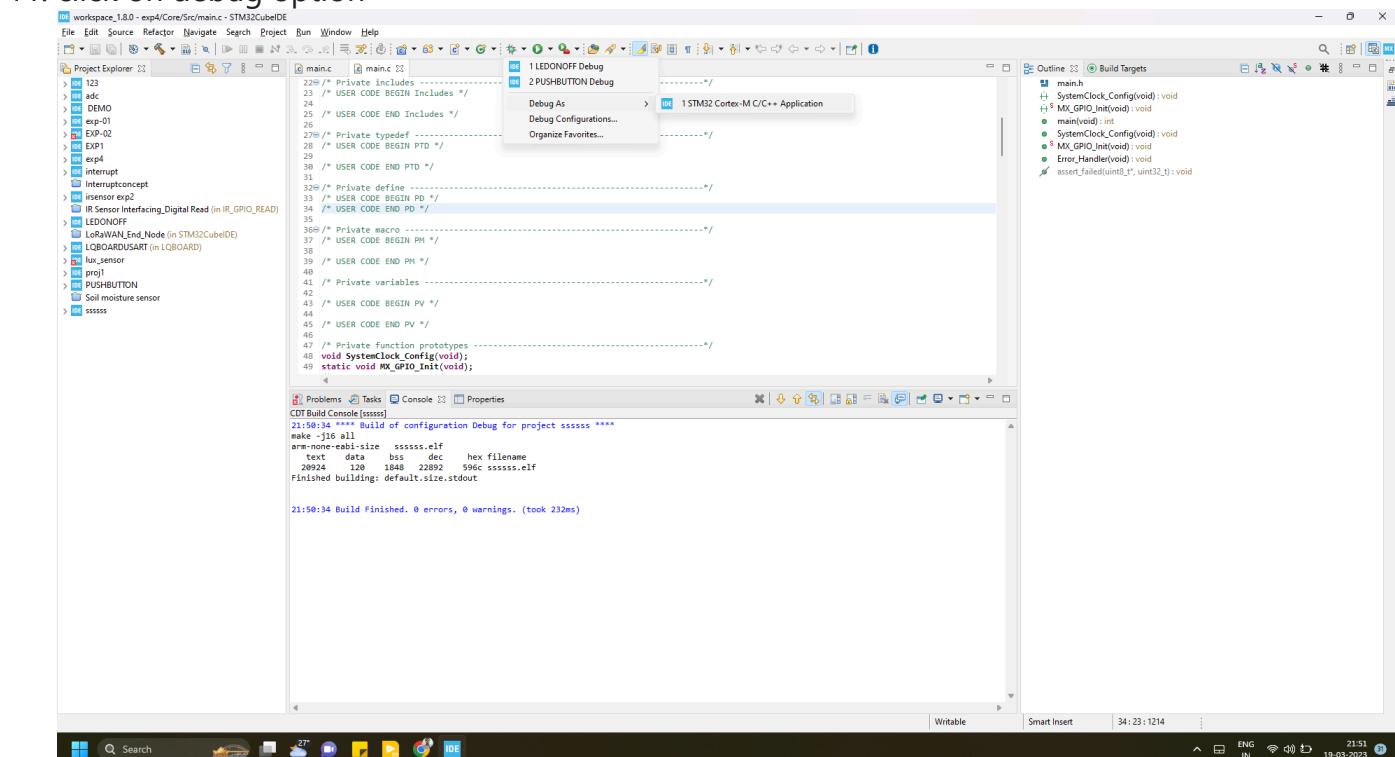
The screenshot shows the STM32CubeIDE interface. In the center, there is a code editor window displaying the main.c file. The file contains C code for an STM32 project, including includes for stdio.h, stdlib.h, and some private headers, as well as definitions for USER\_CODE\_BEGIN and USER\_CODE\_END. Below the code editor is a terminal window showing the build log:

```
21:50:34 **** Build of configuration Debug for project ssssss ****
make -j16 all
arm-none-eabi-size ssssss.elf
    text   data   bss   dec   hex filename
 20924     120   1848  22892   596c ssssss.elf
Finished building: default.size.stdout

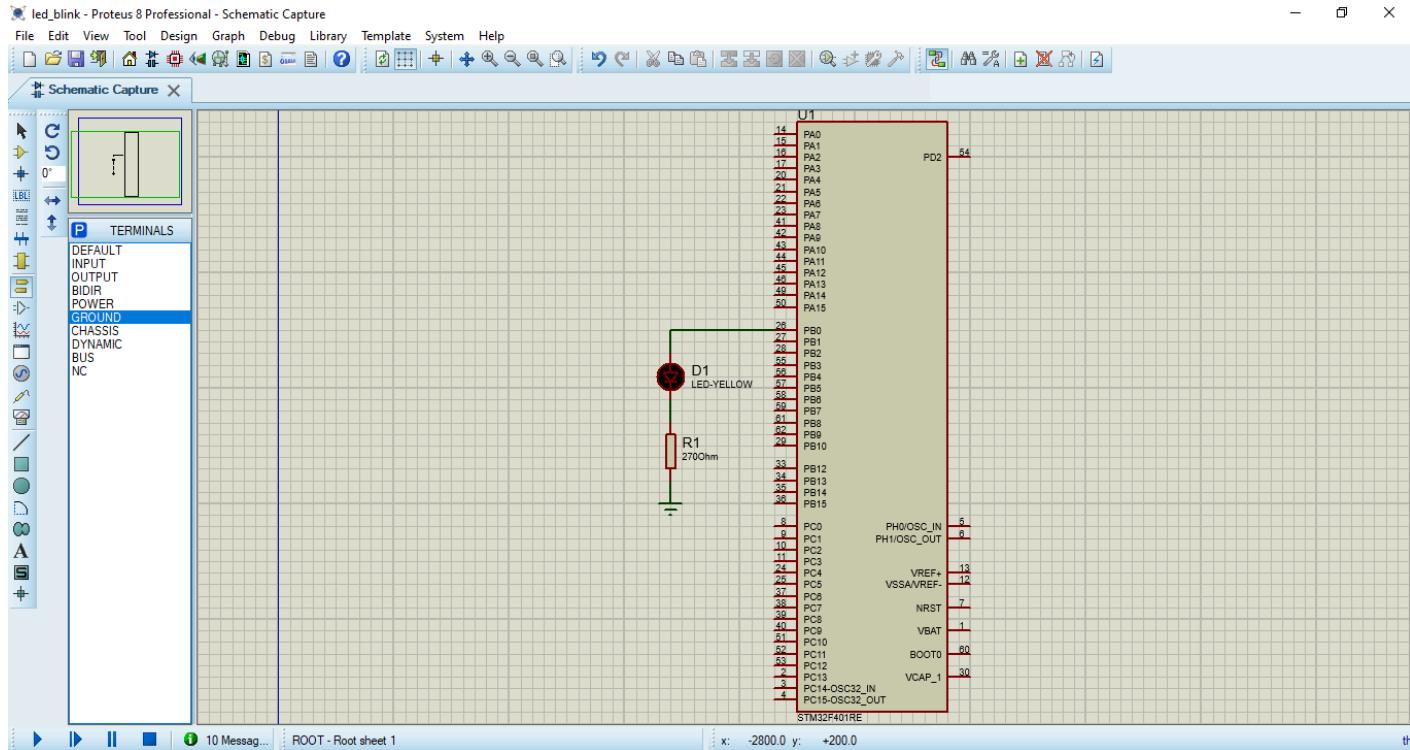
21:50:34 Build Finished. 0 errors, 0 warnings. (took 232ms)
```

At the top of the interface, there is a toolbar with various icons for file operations, and a menu bar with File, Edit, Source, Refactor, Navigate, Project, Run, Window, Help. On the left, there is a Project Explorer window listing several projects and source files. On the right, there are two panes: Outline and Build Targets. The Outline pane shows the structure of the main.c file, and the Build Targets pane shows the selected target as "1 STM32 Cortex-M C/C++ Application". A context menu is open over the Build As section, with options like "1 LEDONOFF Debug", "2 PUSHBUTTON Debug", "Debug As", "Debug Configurations...", and "Organize Favorites...".

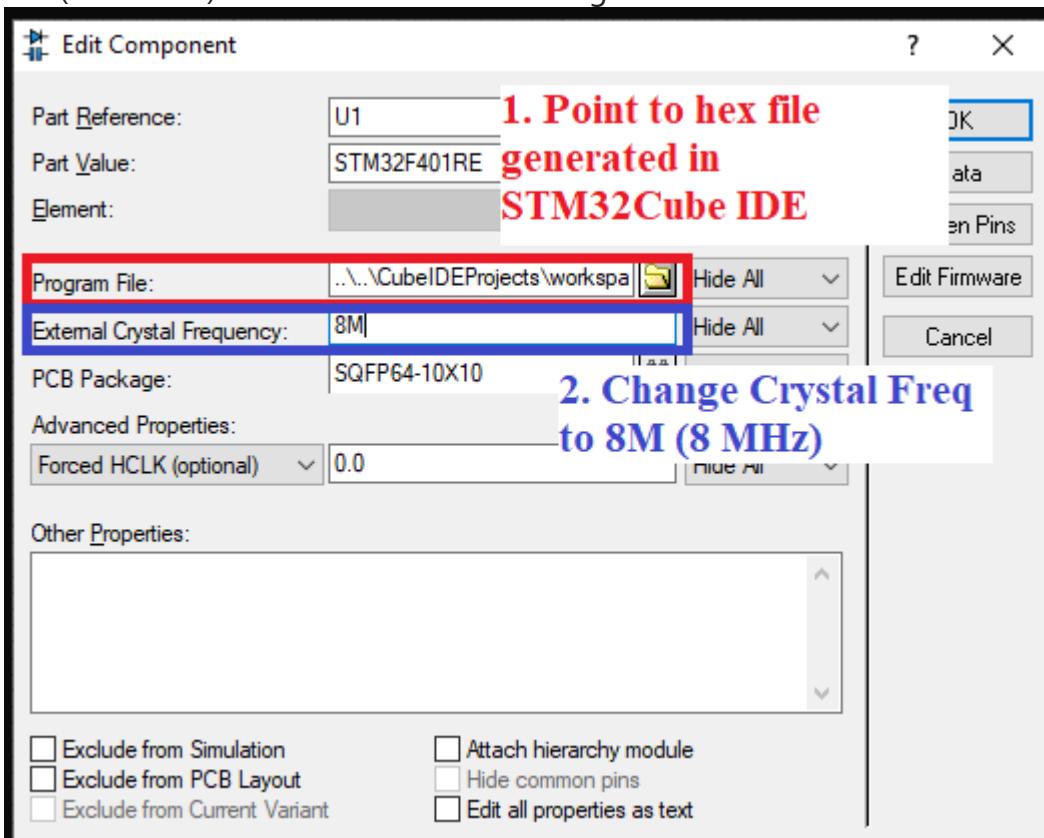
## 11. click on debug option



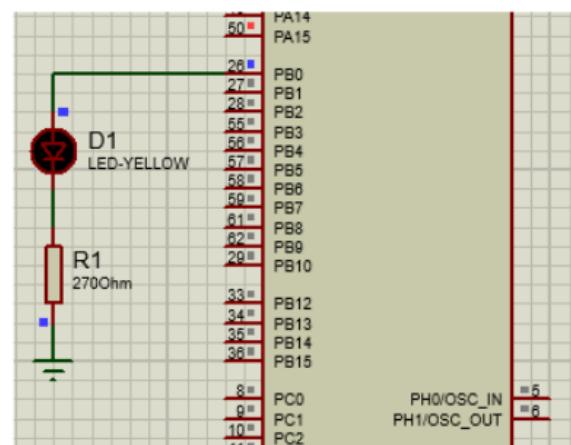
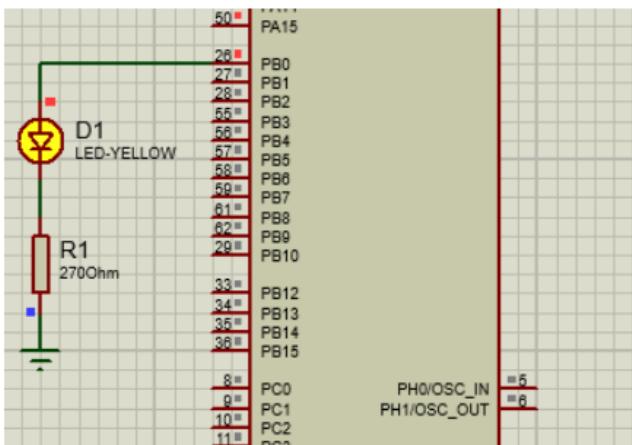
12. Creating Proteus project and running the simulation We are now at the last part of step by step guide on how to simulate STM32 project in Proteus.
13. Create a new Proteus project and place STM32F40xx i.e. the same MCU for which the project was created in STM32Cube IDE.
14. After creation of the circuit as per requirement as shown below



14. Double click on the the MCU part to open settings. Next to the Program File option, give full path to the Hex file generated using STM32Cube IDE. Then set the external crystal frequency to 8M (i.e. 8 MHz). Click OK to save the changes.



15. click on debug and simulate using simulation as shown below



## STM 32 CUBE PROGRAM :

Name : Sam Israel D  
Reg. No: 212222230128

```
#include "main.h"
#include <stdbool.h>
void push_button();
bool button_status;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
```

```
int main(void)
{
    HAL_Init();

    SystemClock_Config();

    MX_GPIO_Init();

    while (1)
    {
        push_button();
    }
}

void push_button()
{
    button_status=HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_13);
    if(button_status==0)
    {
        HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);
    }
    else
    {
        HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_RESET);
    }
}
```

```

void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1);

    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSISState = 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();
    }

    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                                |RCC_CLOCKTYPE_PCLK1;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
    RCC_ClkInitStruct.AHBClockDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1ClockDivider = RCC_HCLK_DIV1;

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

static void MX_GPIO_Init(void)
{
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();

    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

    GPIO_InitStruct.Pin = GPIO_PIN_13;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_5;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

}

void Error_Handler(void)

```

```

{
    __disable_irq();
    while (1)
    {
    }
}

#ifndef USE_FULL_ASSERT

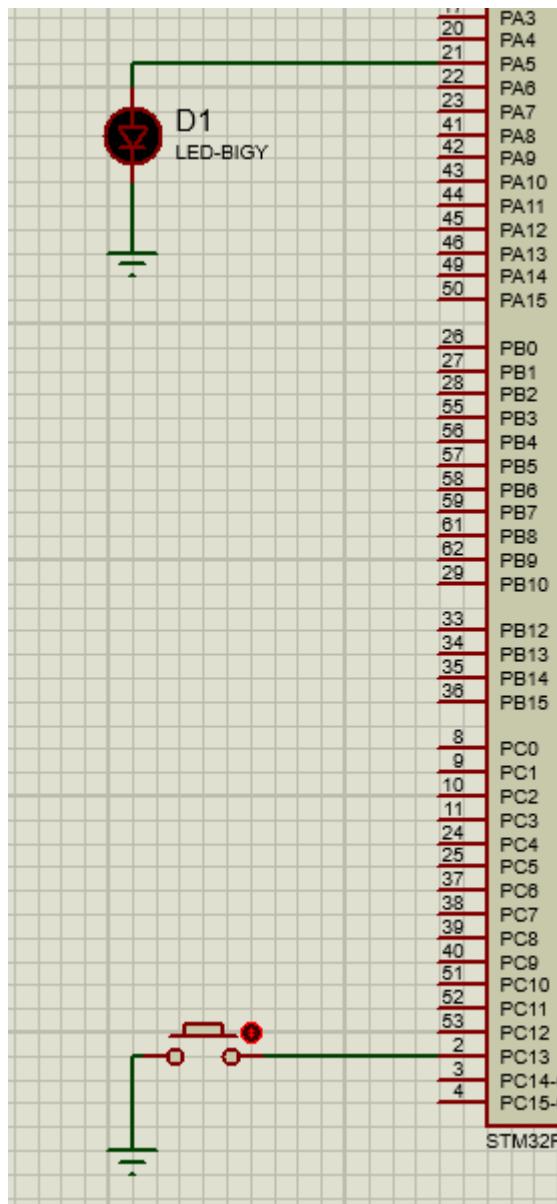
void assert_failed(uint8_t *file, uint32_t line)
{

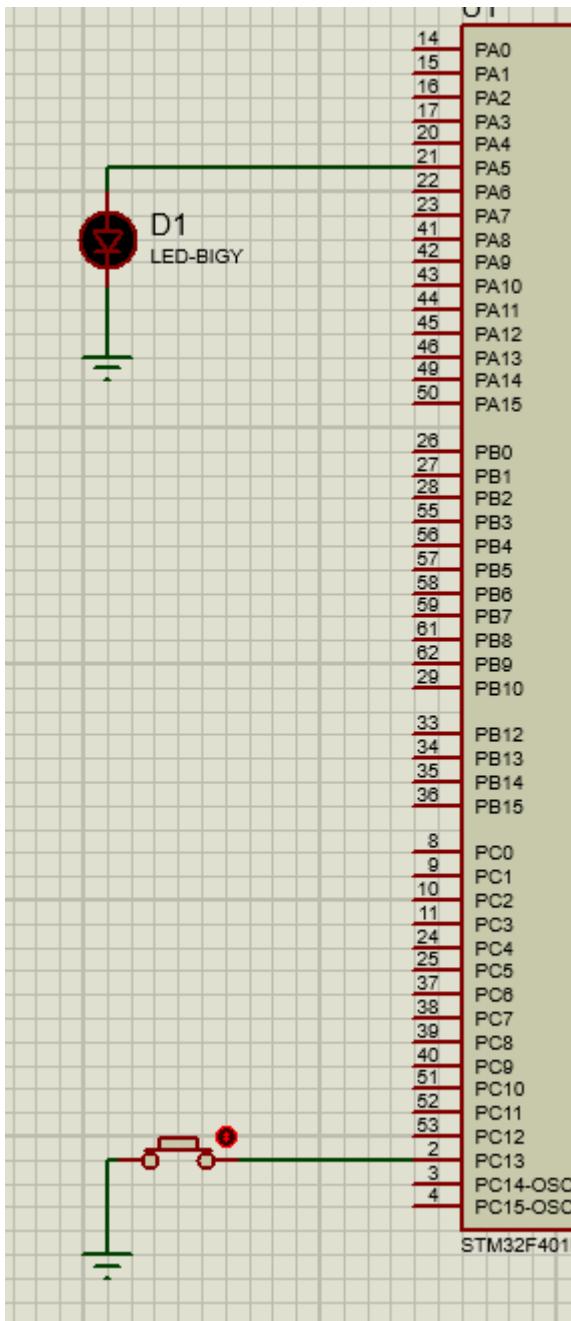
}

#endif

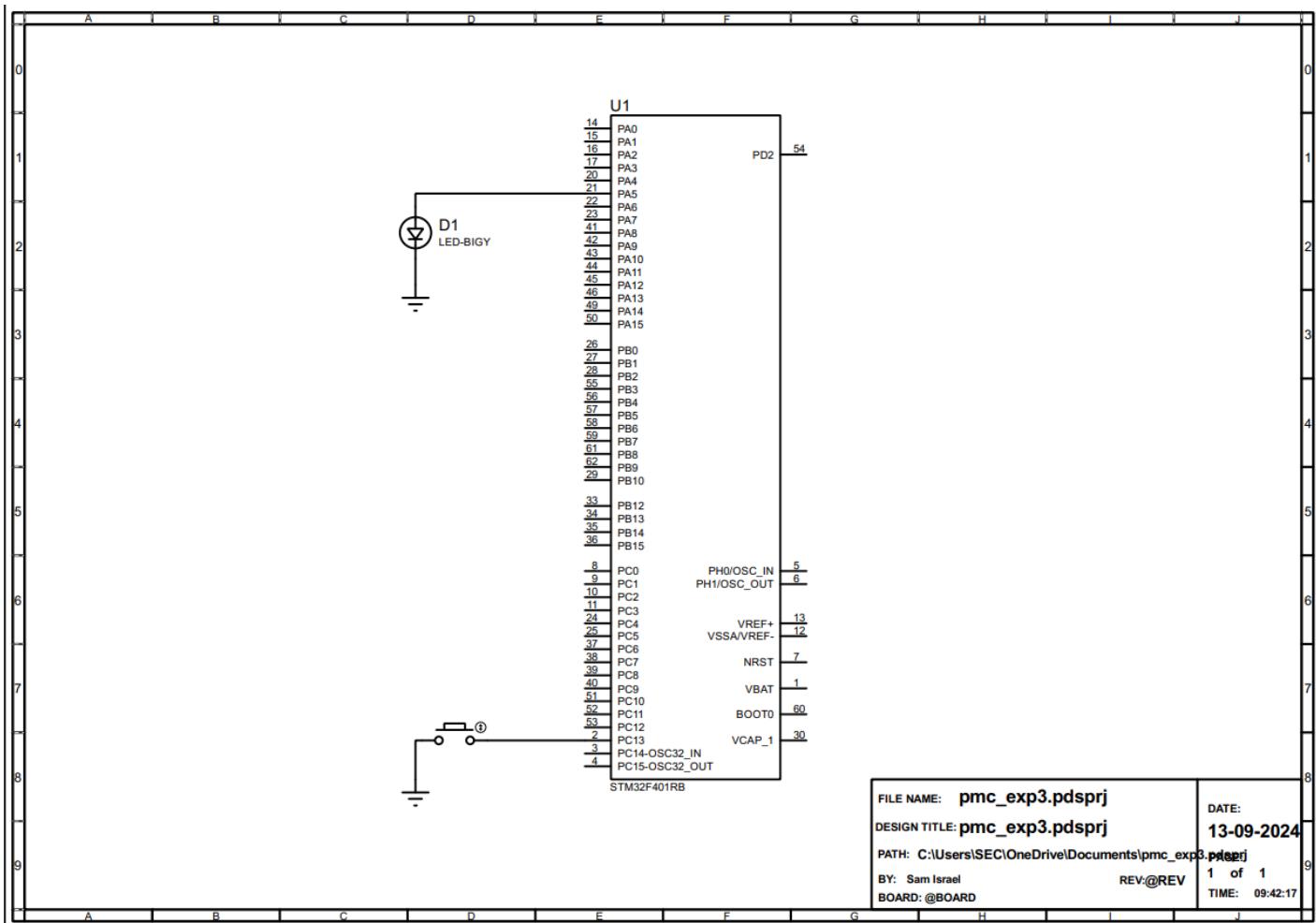
```

## Output screen shots of proteus :





Proteus layout(Add pdf screen shot of circuit here)



## Result :

Interfacing a digital output and digital input with ARM microcontroller are simulated in proteus and the results are verified.