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

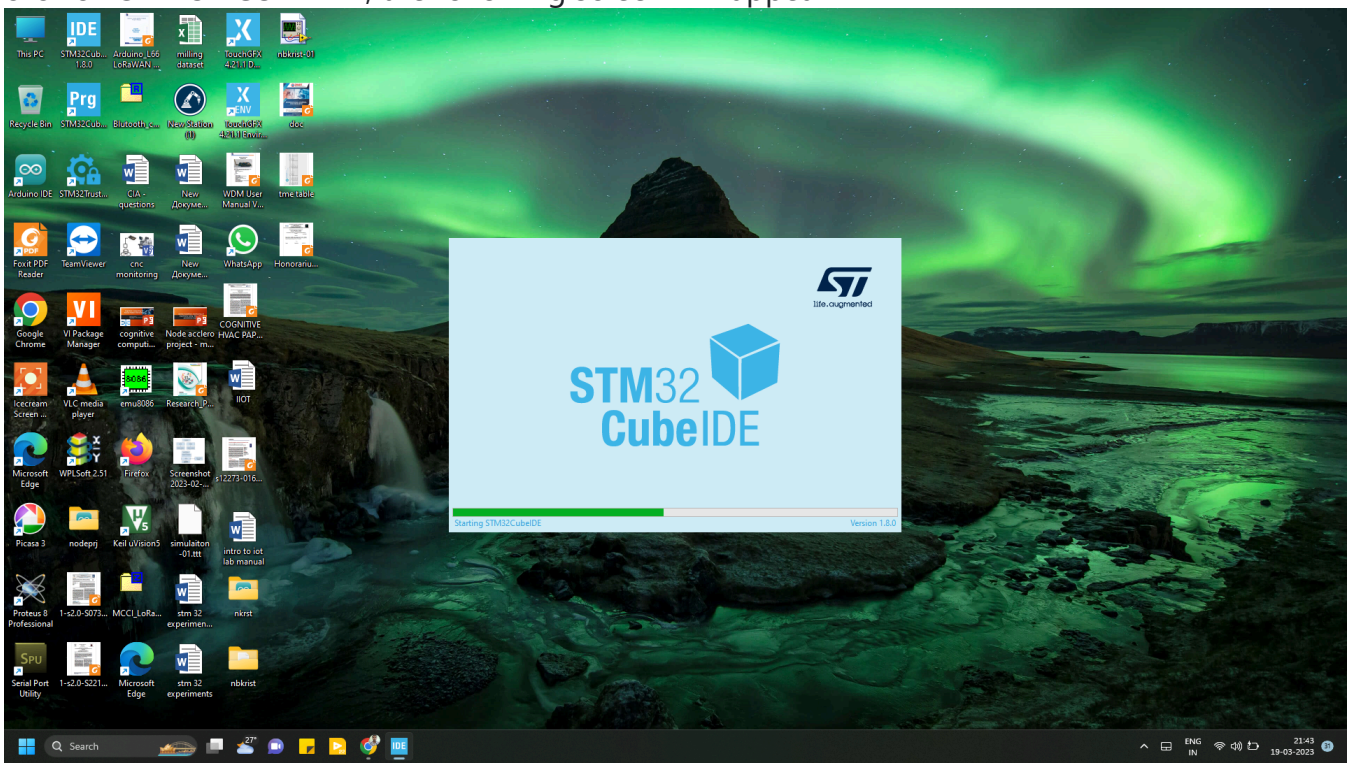
Components required: STM32 CUBE IDE, Proteus 8 simulator .

Theory

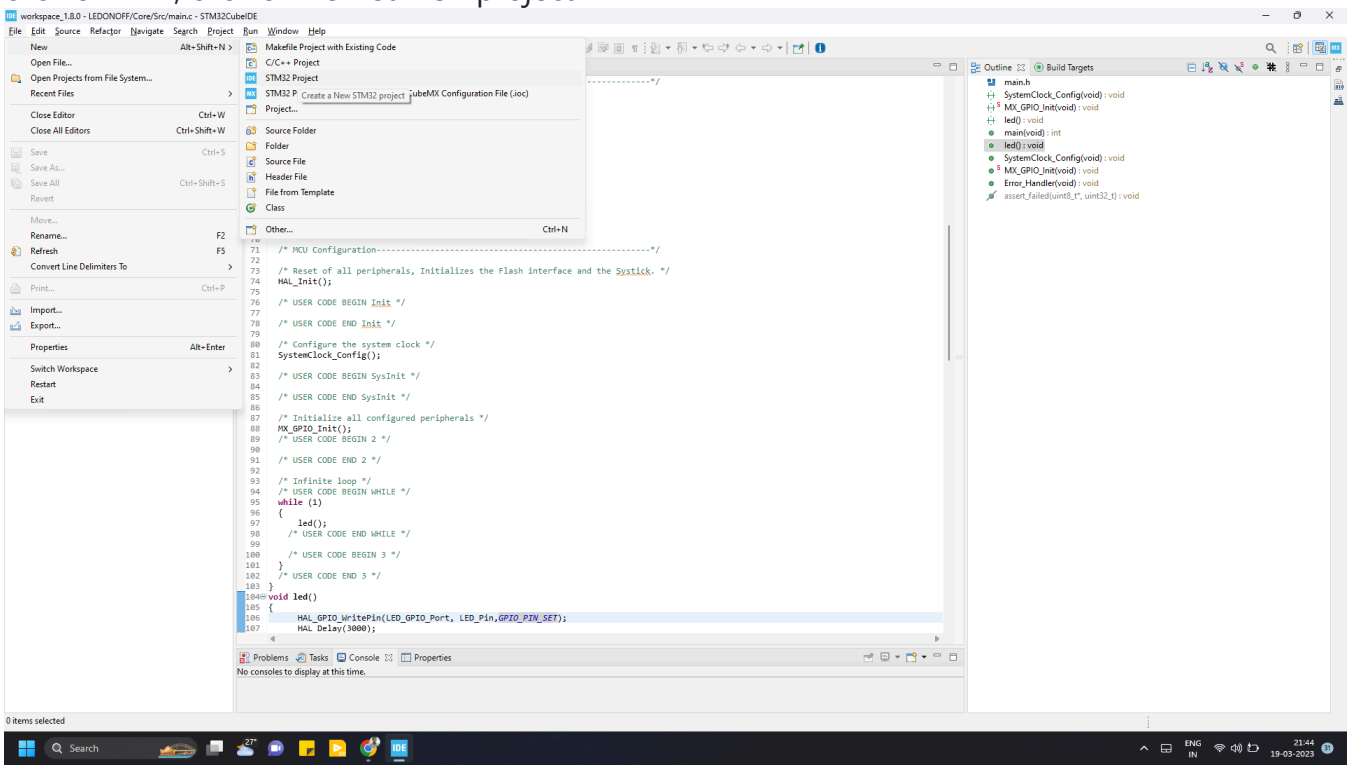
The full form of an ARM is an advanced reduced instruction set computer (RISC) machine, and it is a 32-bit processor architecture expanded by ARM holdings. The applications of an ARM processor include several microcontrollers as well as processors. The architecture of an ARM processor was licensed by 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

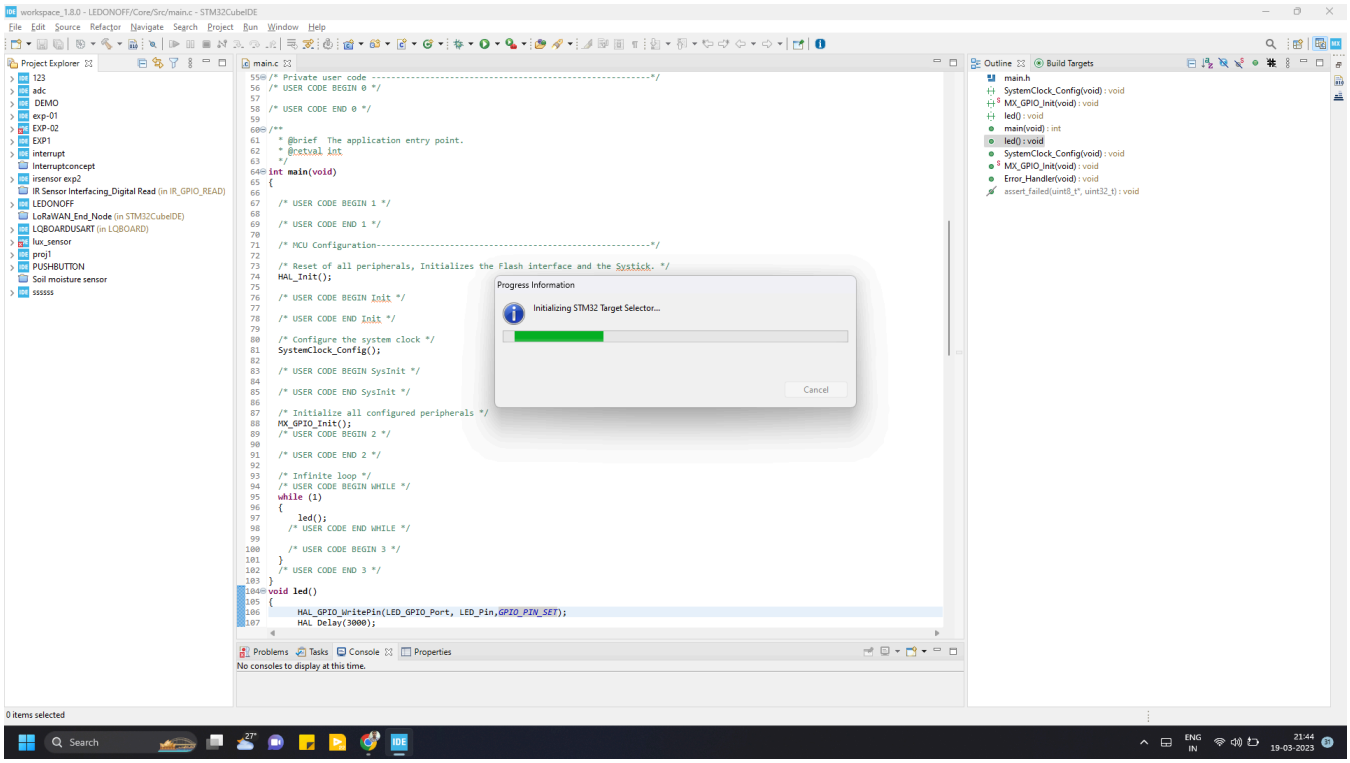
Procedure:

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

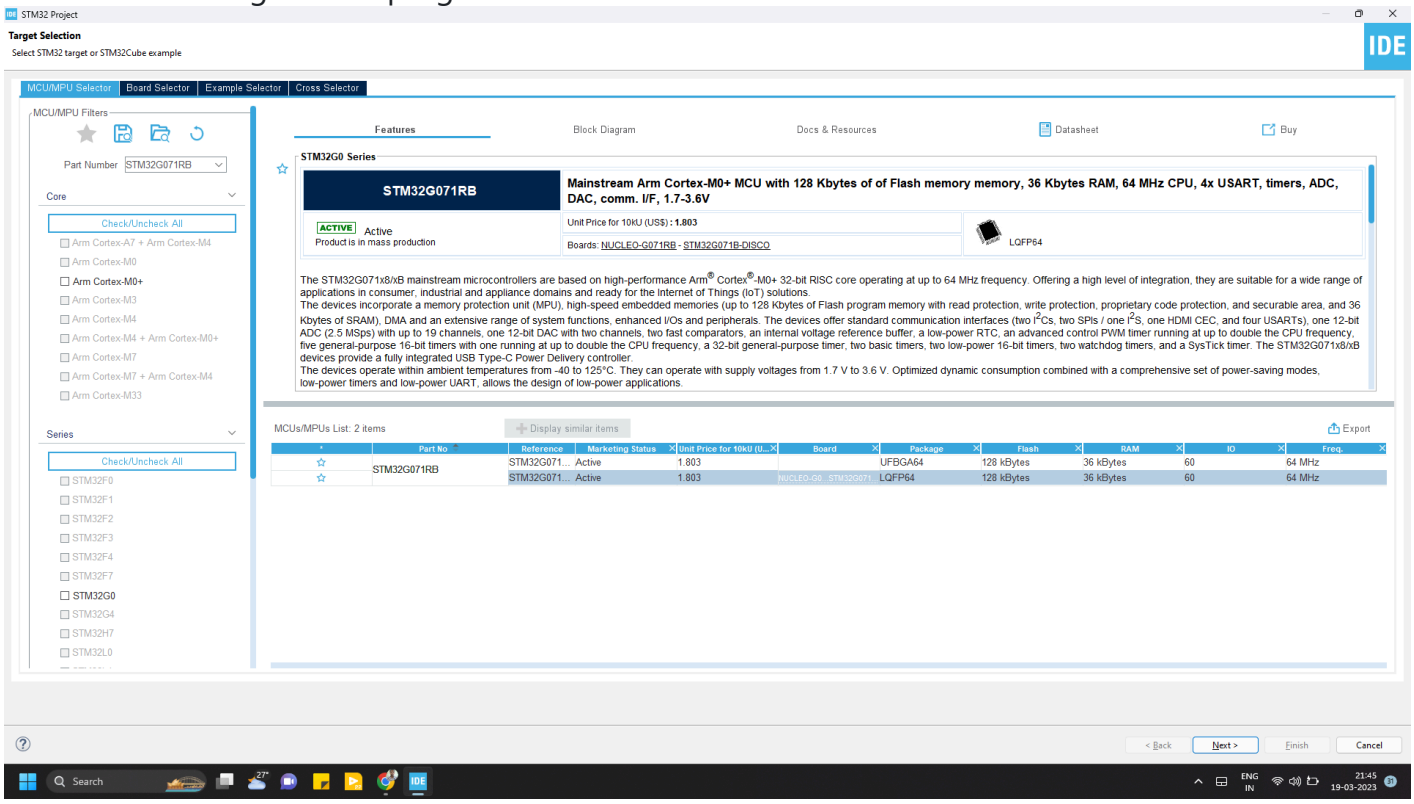


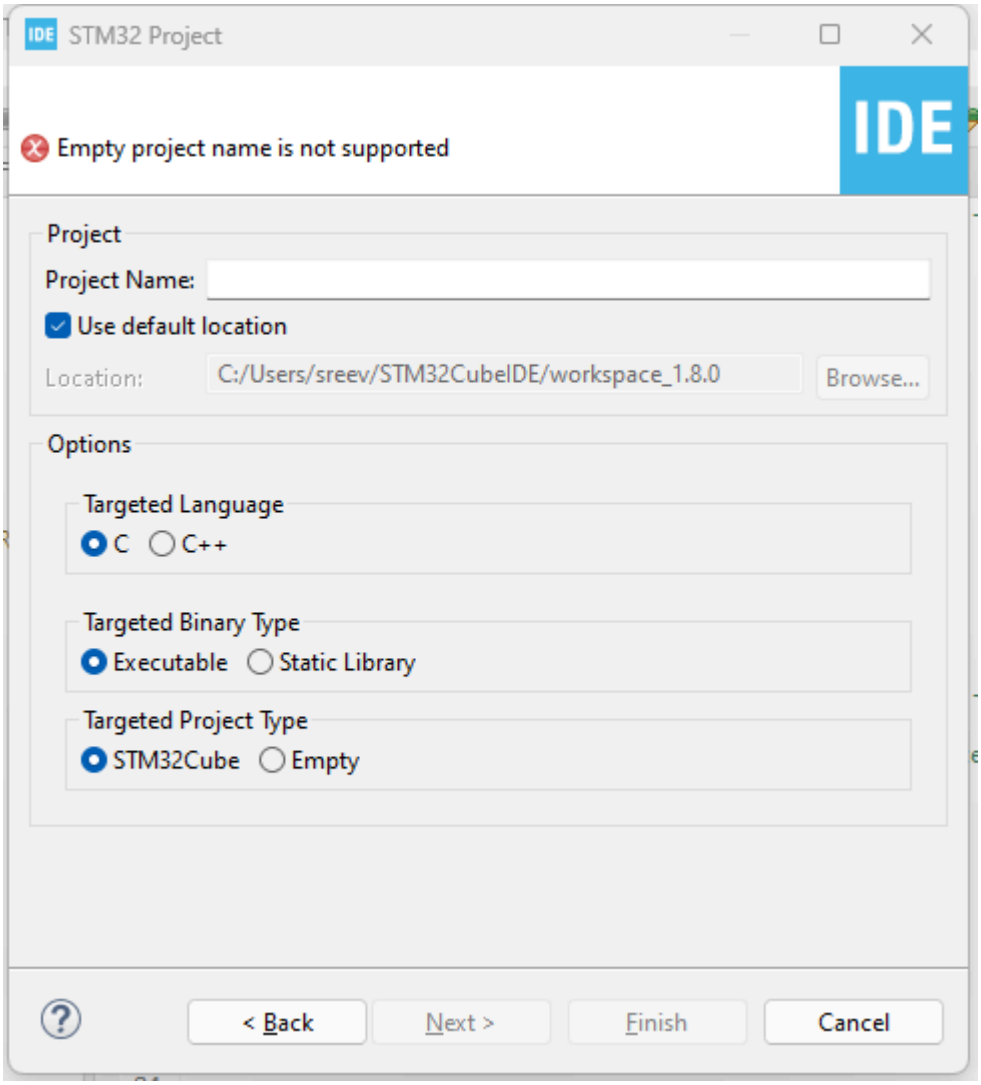
2. click on FILE, click on new stm 32 project





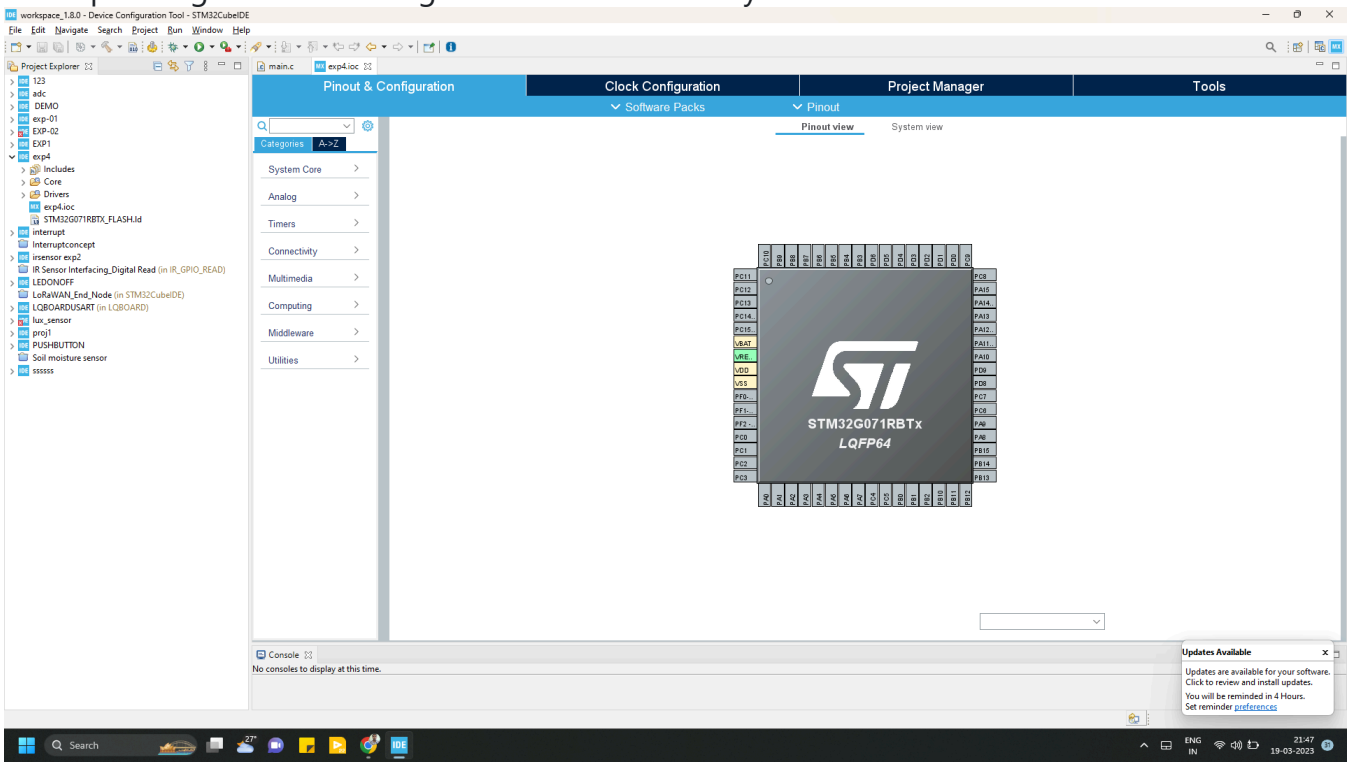
3. select the target to be programmed as shown below and click on next



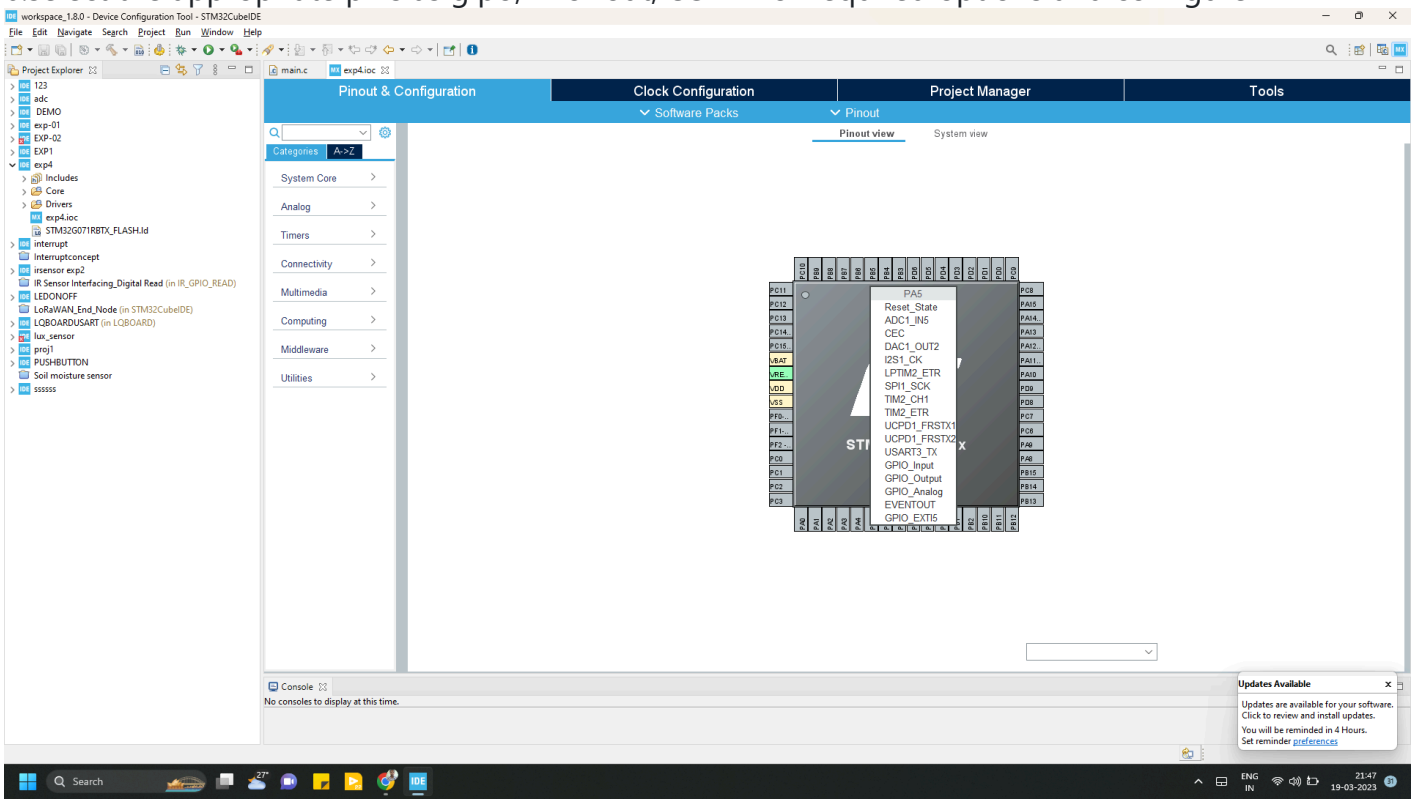


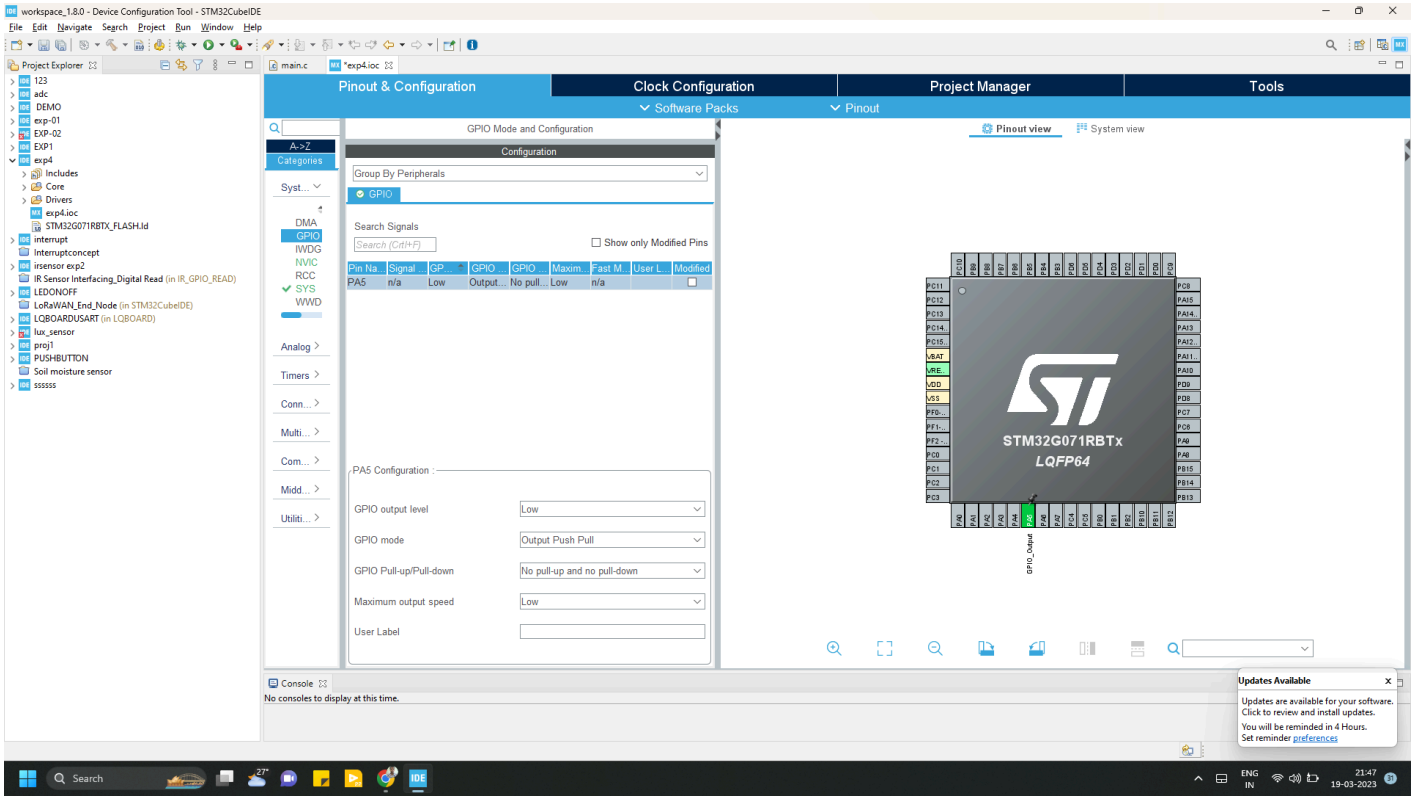
4.select the program name

5. corresponding ioc file will be generated automatically

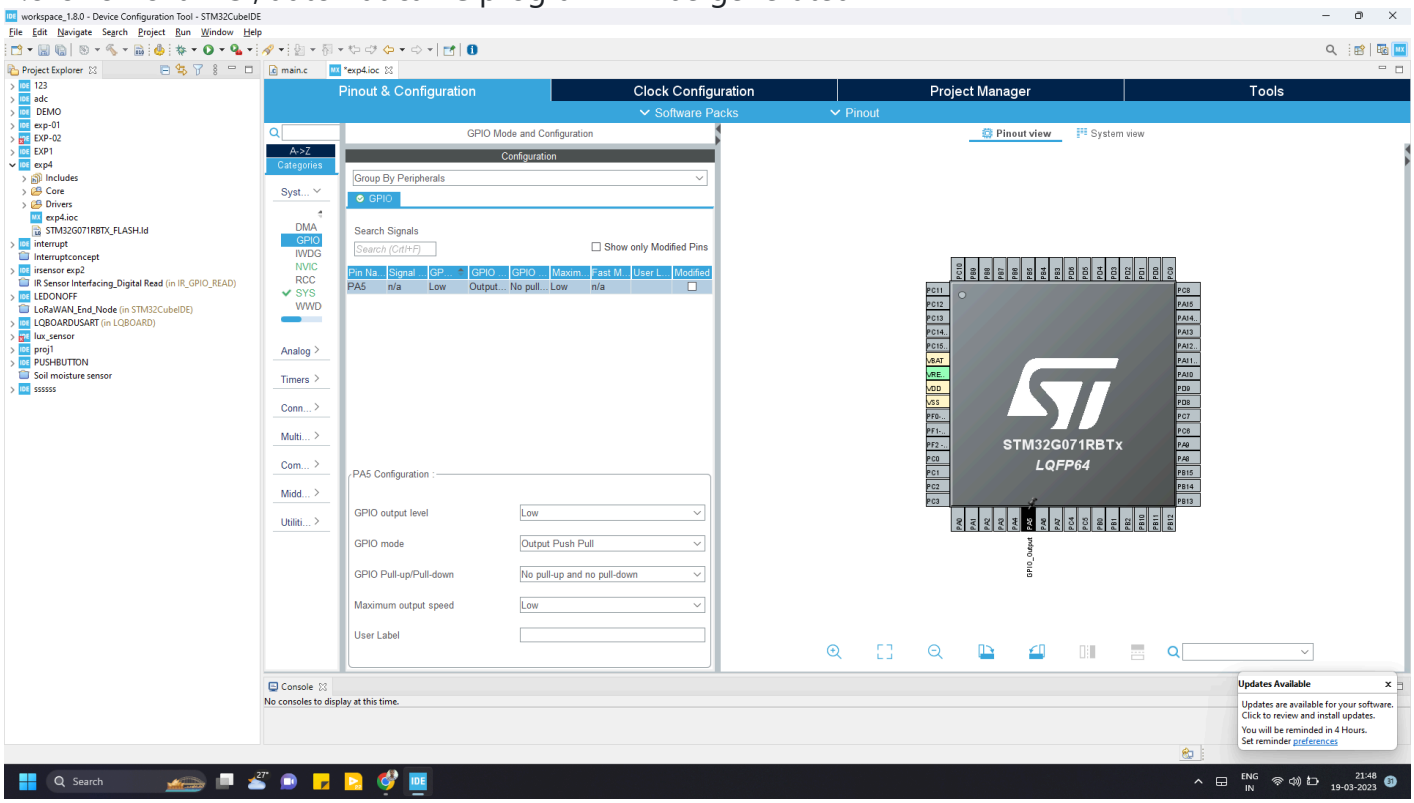


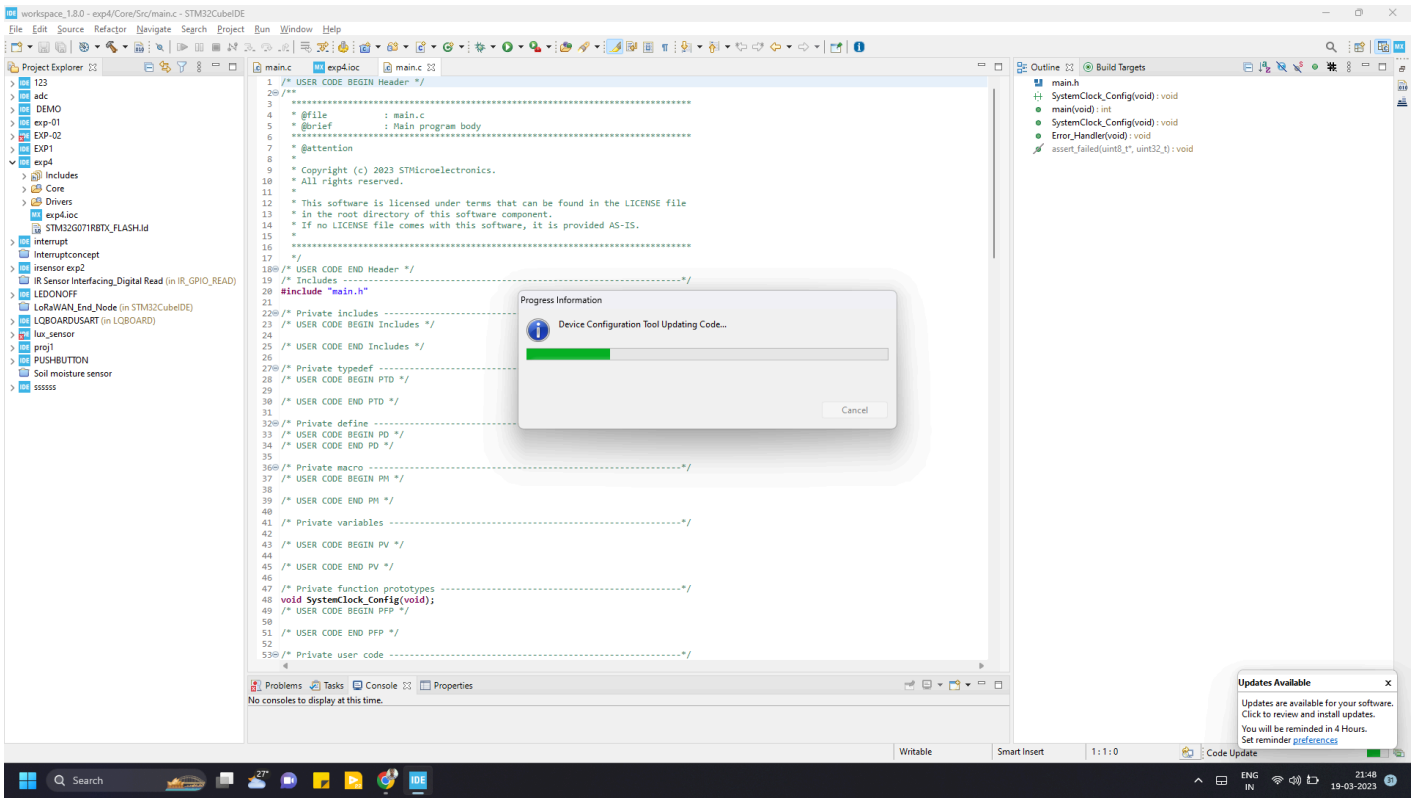
6.select the appropriate pins as gipo, in or out, USART or required options and configure



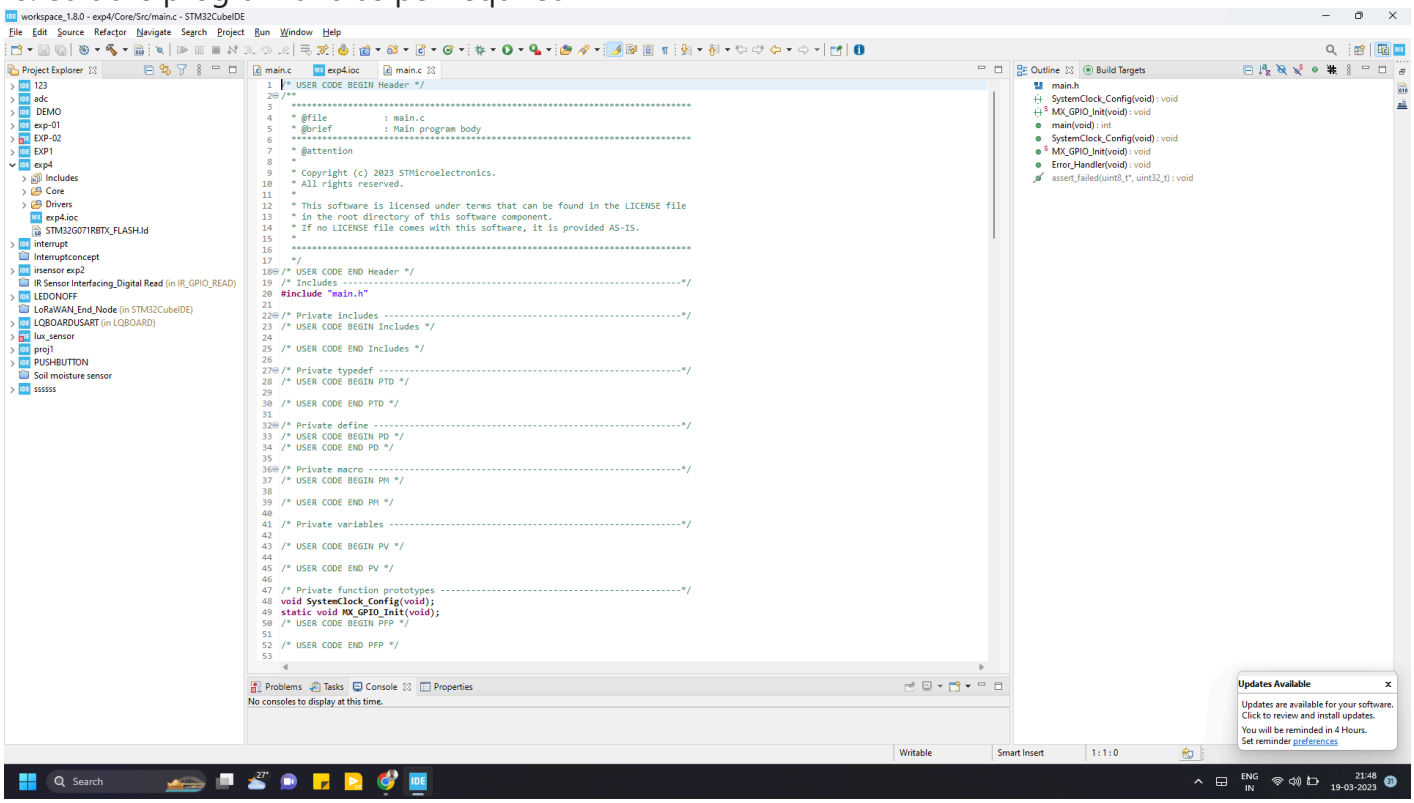


7.click on cntrl+S , automaticall C program will be generated

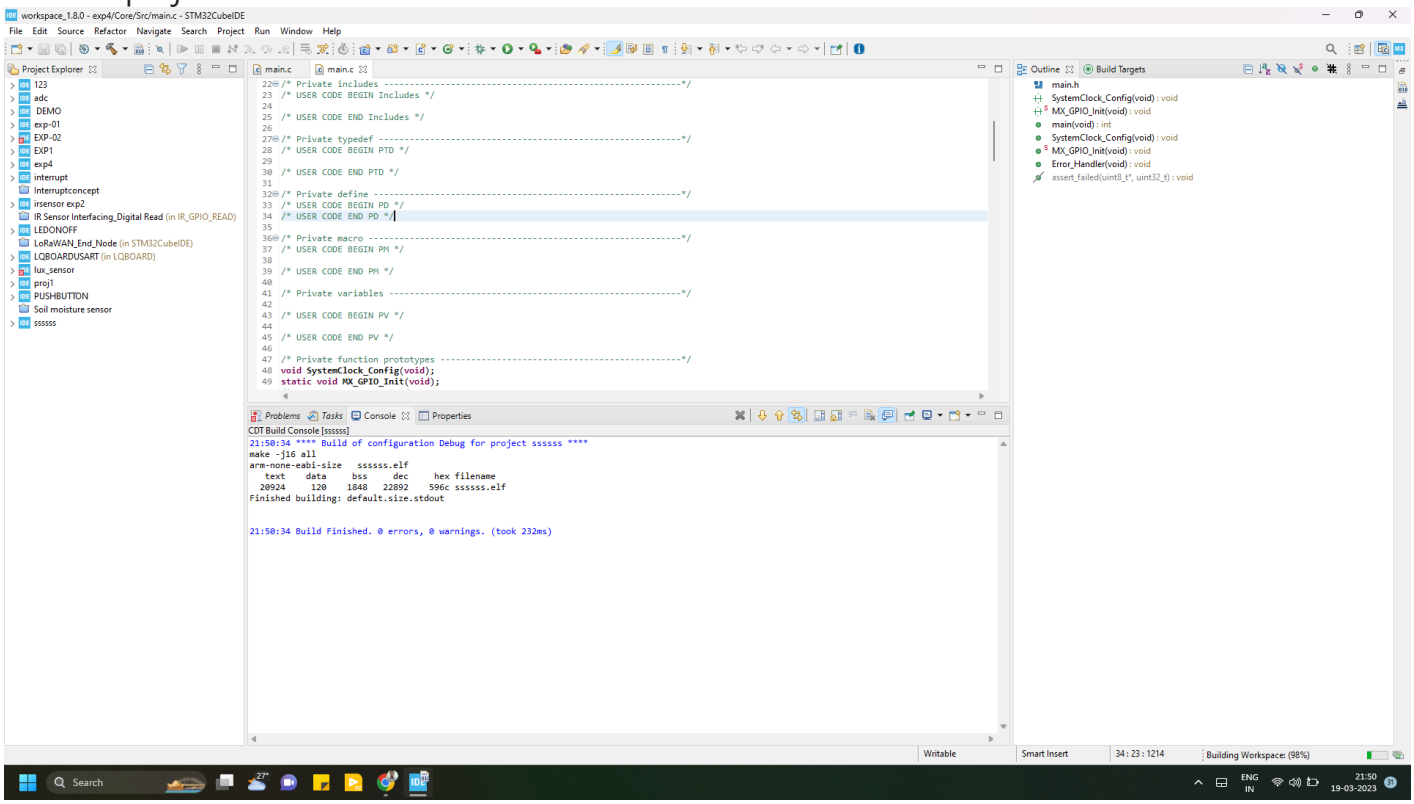




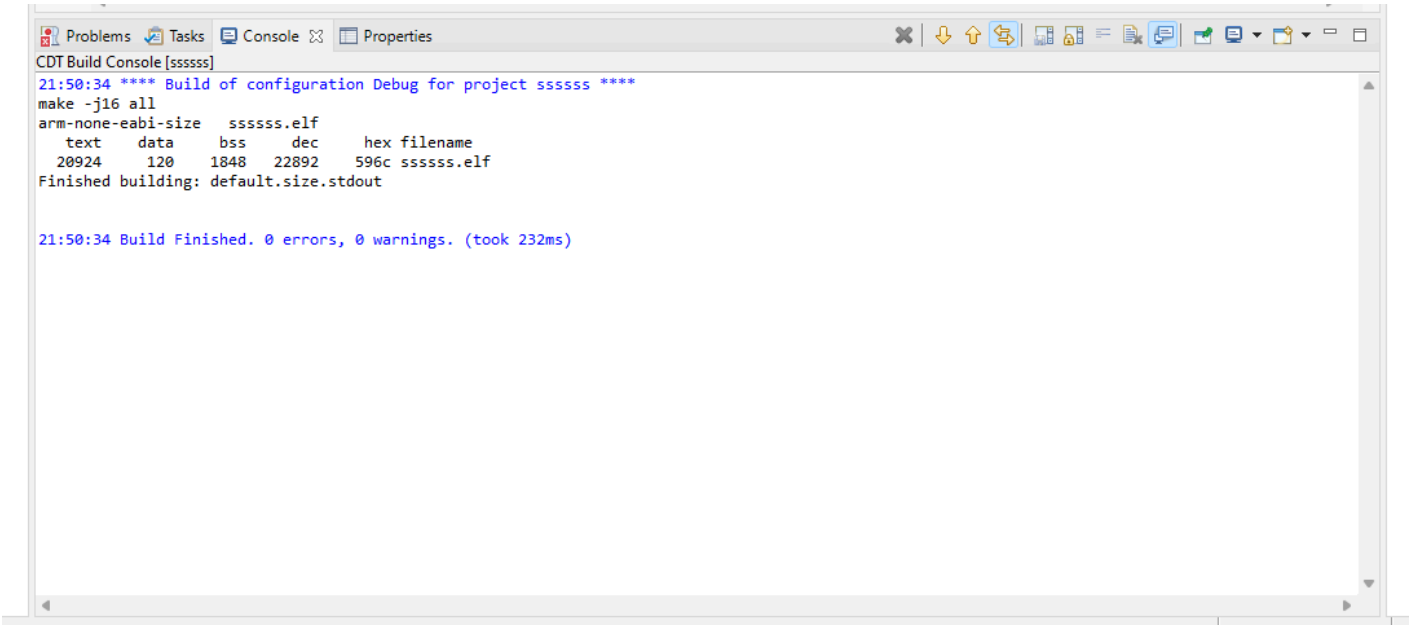
8. edit the program and as per required



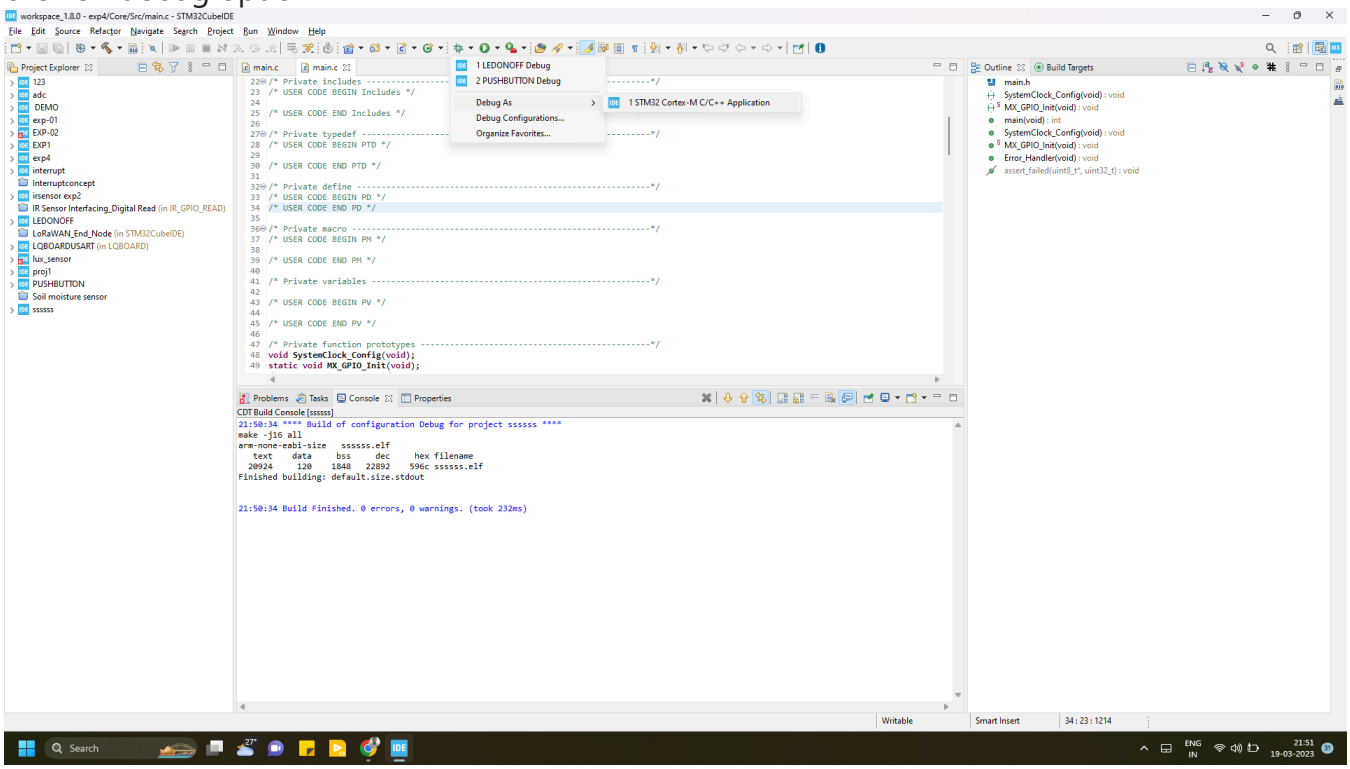
9. use project and build



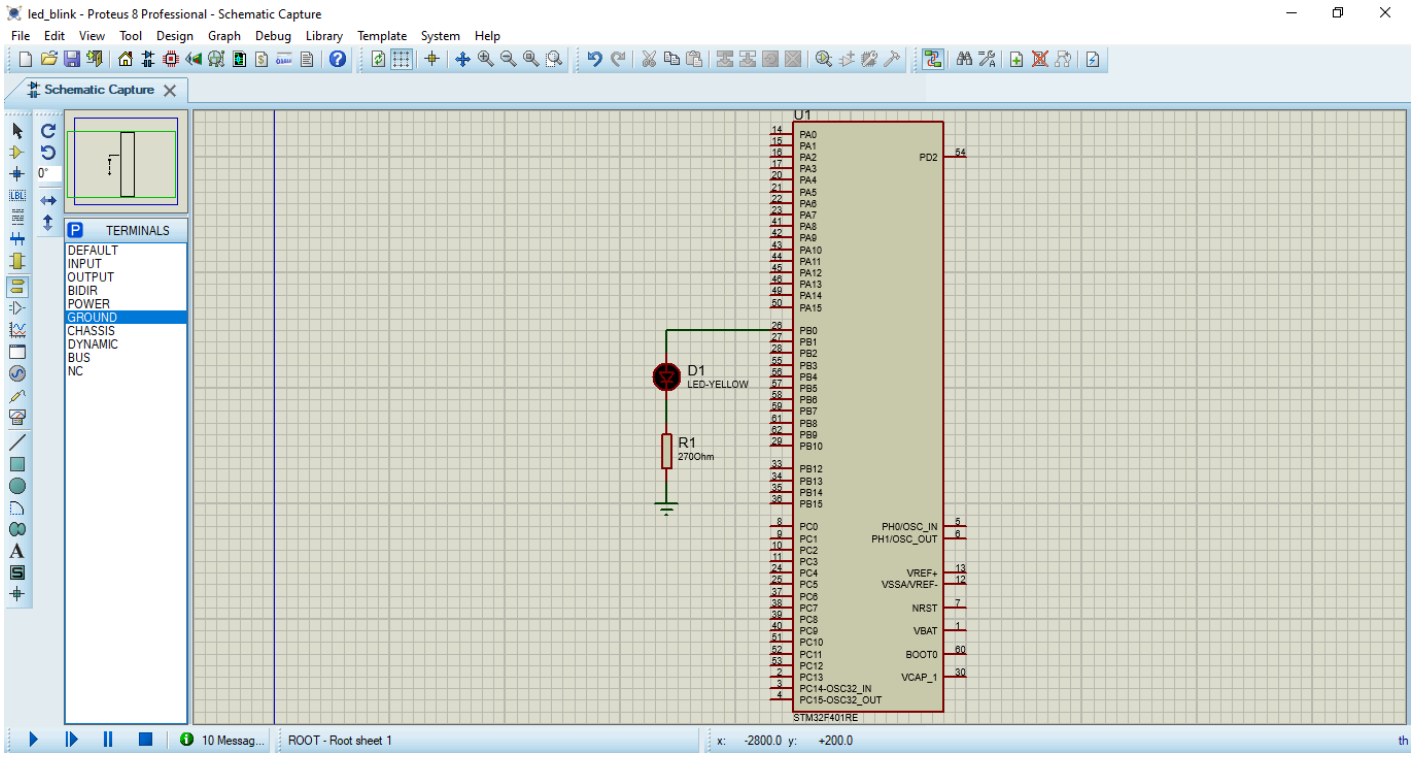
10. once the project is bulild



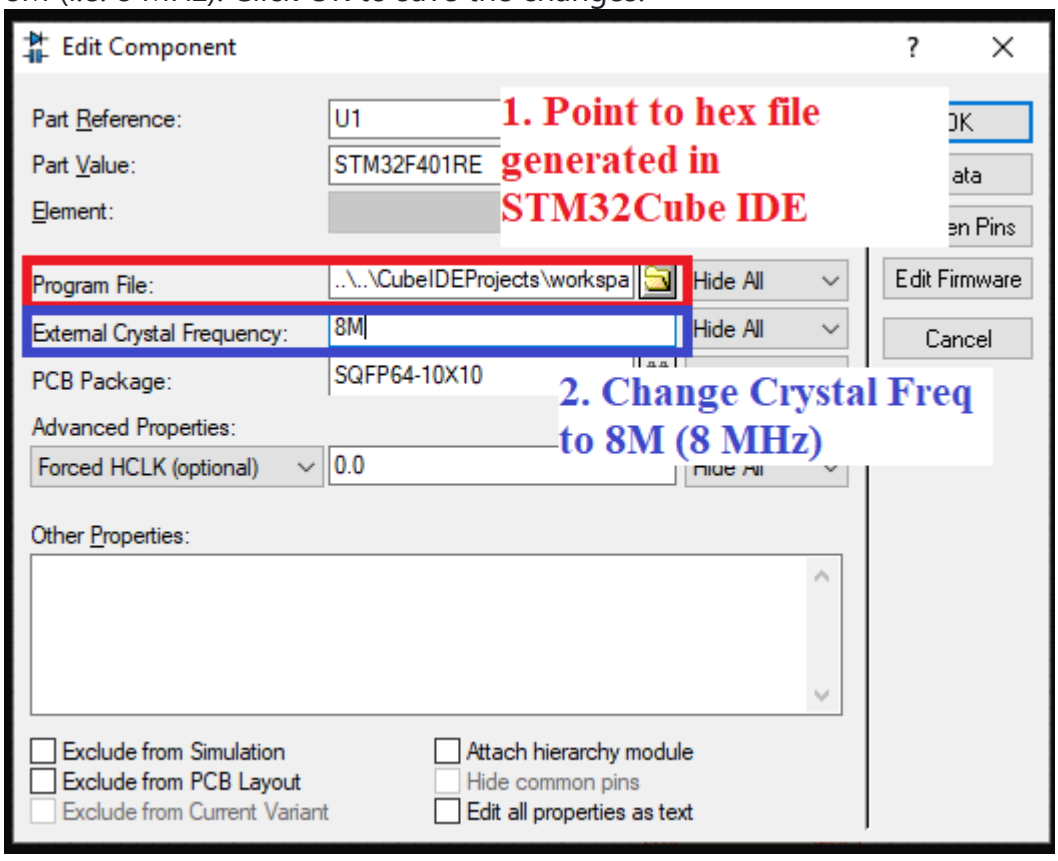
11. click on debug option



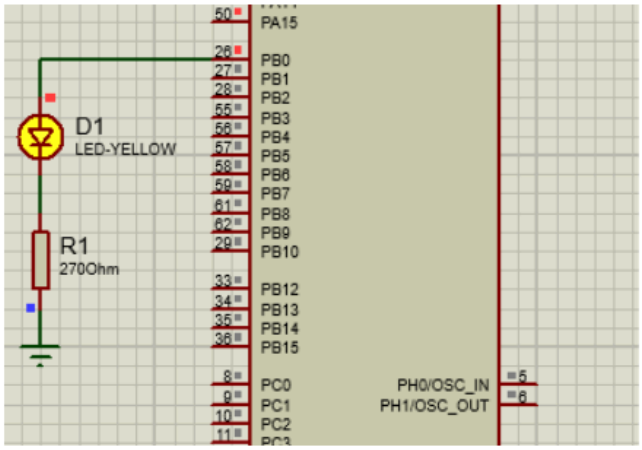
12. Creating Proteus project and running the simulation We are now at the last part of step by step guide on how to simulate STM32 project in Proteus.
13. Create a new Proteus project and place STM32F40xx i.e. the same MCU for which the project was created in STM32Cube IDE.
14. After creation of the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

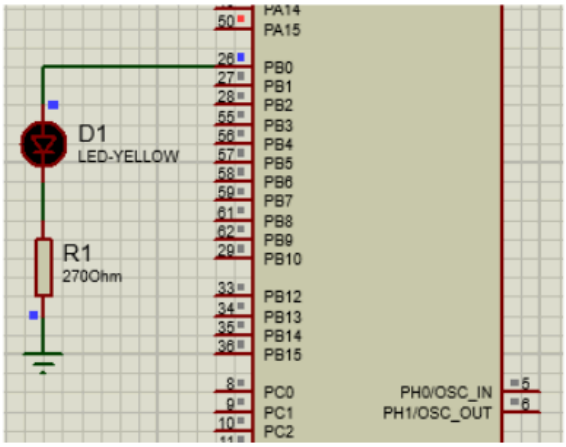


14. Double click on the the MCU part to open settings. Next to the Program File option, give full path to the Hex file generated using STM32Cube IDE. Then set the external crystal frequency to 8M (i.e. 8 MHz). Click OK to save the changes.



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





STM 32 CUBE PROGRAM :

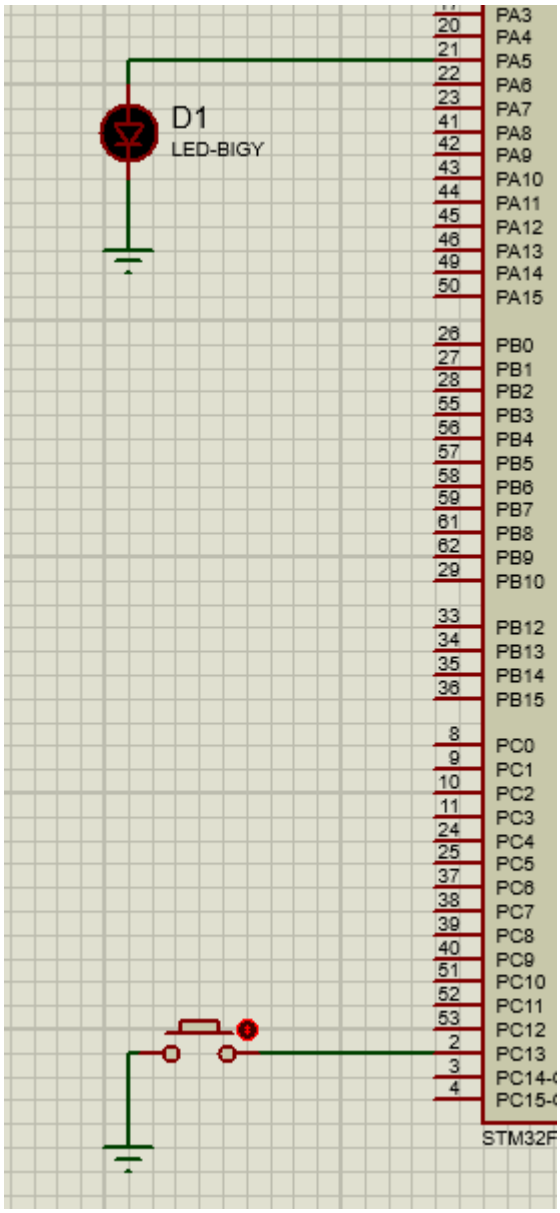
```
Name : Sam Israel D
Reg. No: 212222230128
```

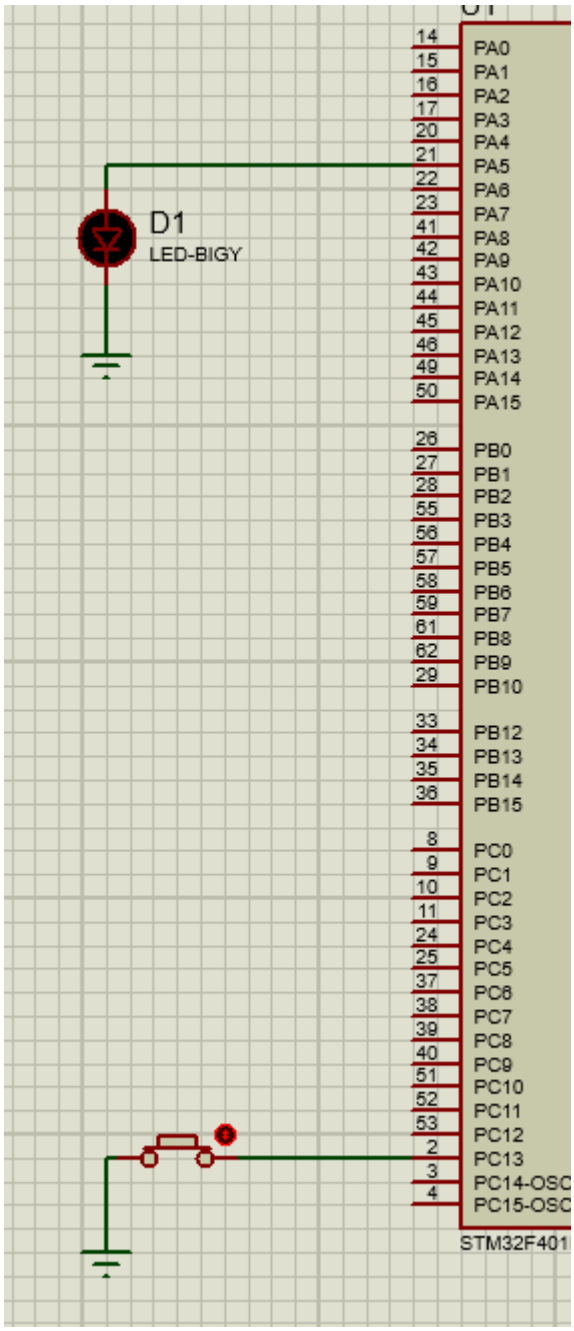
```
#include "main.h"
#include <stdbool.h>
void push_button();
bool button_status;

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.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = 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)
```

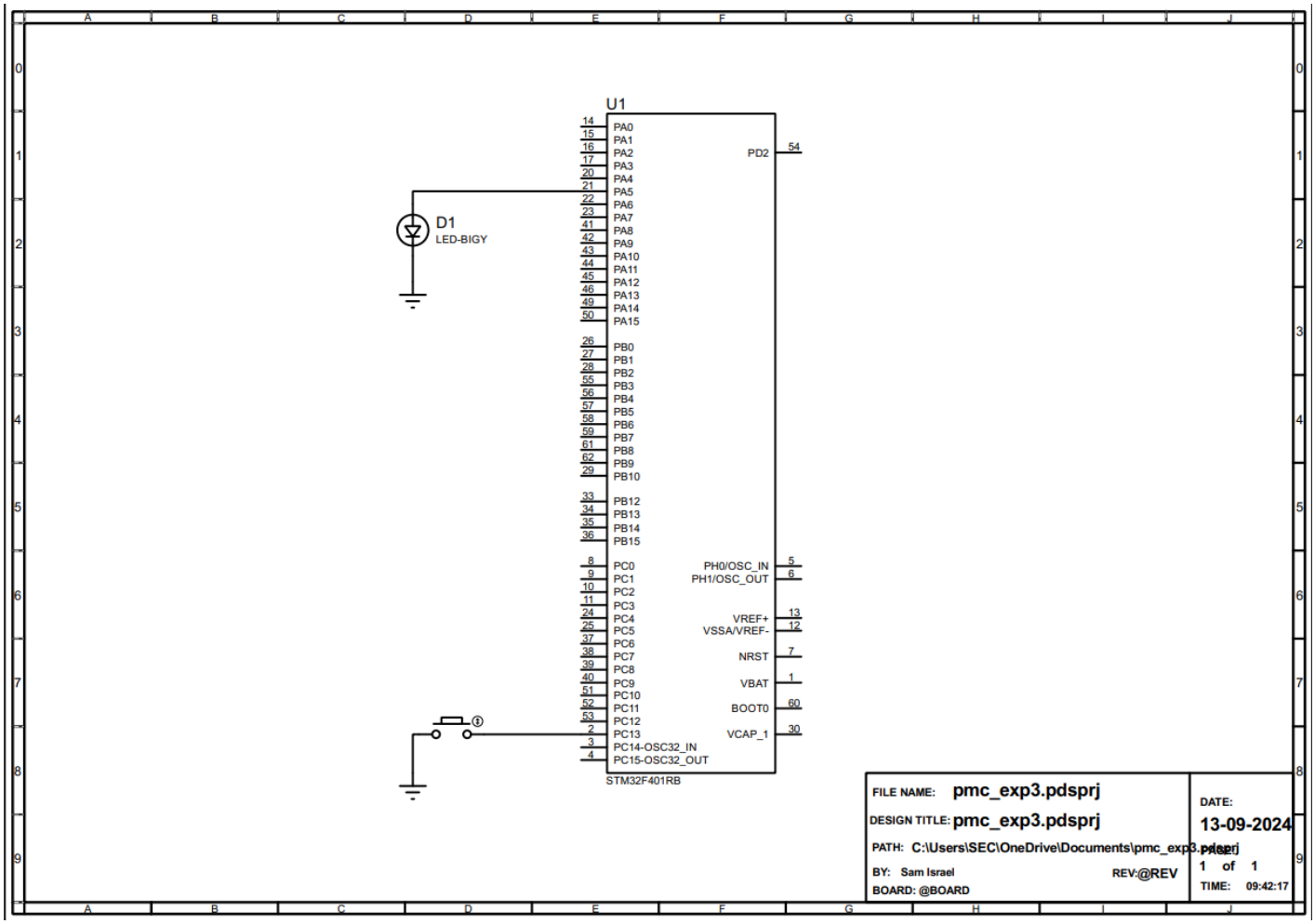
```
{
  __disable_irq();
  while (1)
  {
  }
}

#ifdef  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.