Objective:

A microcontroller is a small integrated circuit that controls a single operation in an embedded system. On a single chip, a typical microcontroller contains a CPU, memory, and input/output (I/O) peripherals. Microcontrollers are utilized in products and equipment that are automatically operated, such as automobile engine control systems, implantable medical devices, remote controls. Office machinery, appliances, power tools, toys and other embedded systems. To complete the work with a microcontroller, C/C++ and assembly language are unquestionably required. For low power consumption, some microcontrollers may use four-bit words and run at frequencies as low as 4KHz. A microcontroller is a compact microcomputer designed to control the functions of embedded systems in office equipment, Robotics, home appliances, motor vehicles and a variety of other devices. A microcontroller has components like - memory, peripherals and most importantly a processor.

Although a microcontroller is capable of doing a wide range of tasks the following issues have

have arisen while dealing with an implementing the lab:

① To get familiarized with Microcontroller.
② To make the LED blink using STM32 and the delay functions.
③ To get introduce with the Implementation of a traffic control system using STM32.

The apparatus and software name:

1| STM32 Board;
2| Bread board;
3| Jumper wires;
4| STM32 Cube IDE (1.0.1 or any Recent version)

Theory and Programs:

STM32CubeIDE is a powerful C/C++ development environment with tools for debugging, code generation, code opti Compilation and Peripheral setup of STM32 microcontrollers and microprocessor With the help of the build and stack analyze in STM32CubeIDE, users may learn vital details about the state of their projects and their memory needs. Along with live Variable monitoring, a Serial Wire Viewer interface, and fault analyzers, STM32CubeI also offers basic and sophisticated debugging faci lities, including views of CPU core registers memory and peripheral registers.

Procedure:

1| First we familiarized ourselves with the different type Arduino family components. STM32 Microcontroller is one of them.

2| After that, we know the basic working principal of thinker cad software.

3| At first we need to open software thinker cad inonline to implement the given circuit. Then we select different type of components to implement it.

4| Next, we write the code to run traffic light system.

5| After that we implements the circuit in the breadboard and connect this experiment in the hardware.

# Source code:

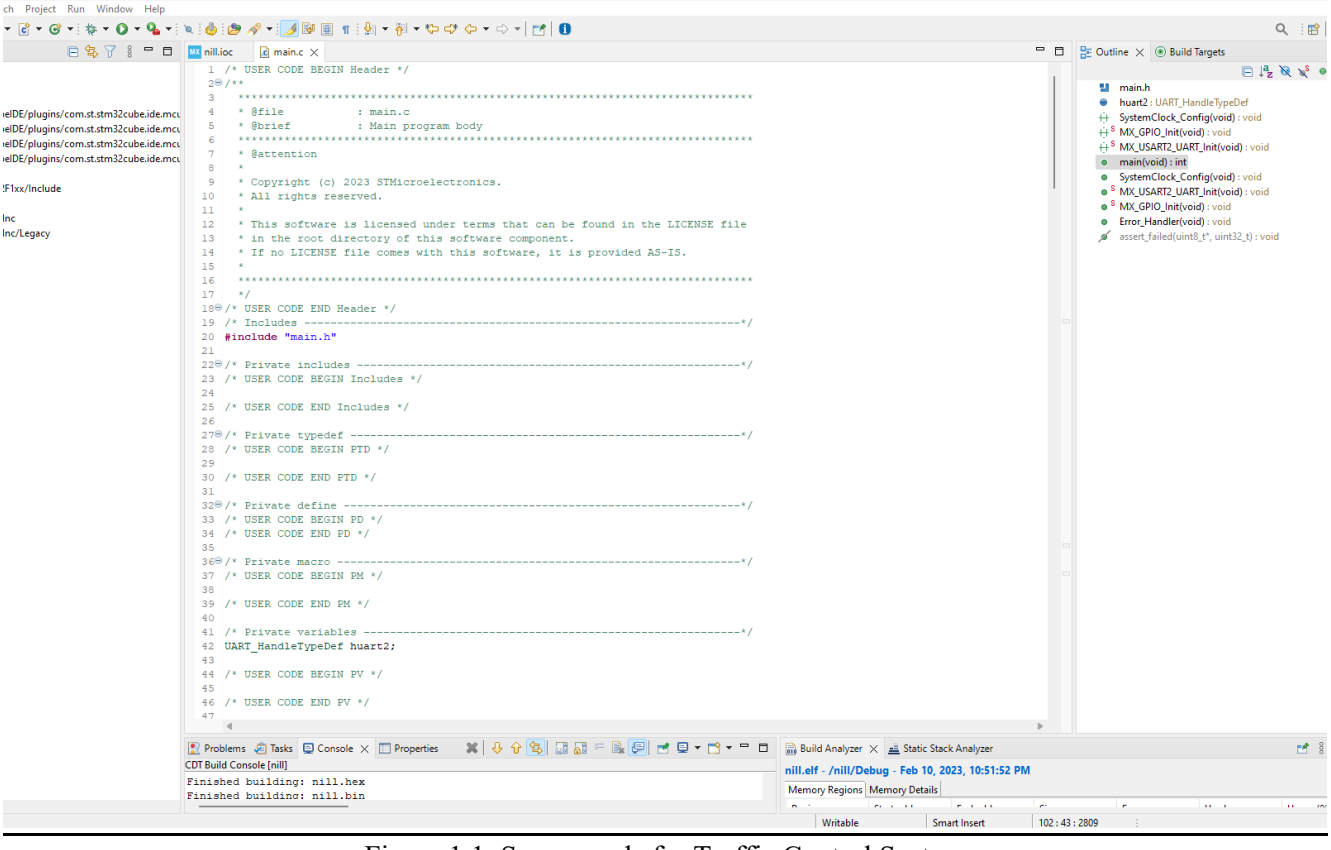



Figure 1.1: Source code for Traffic Control System

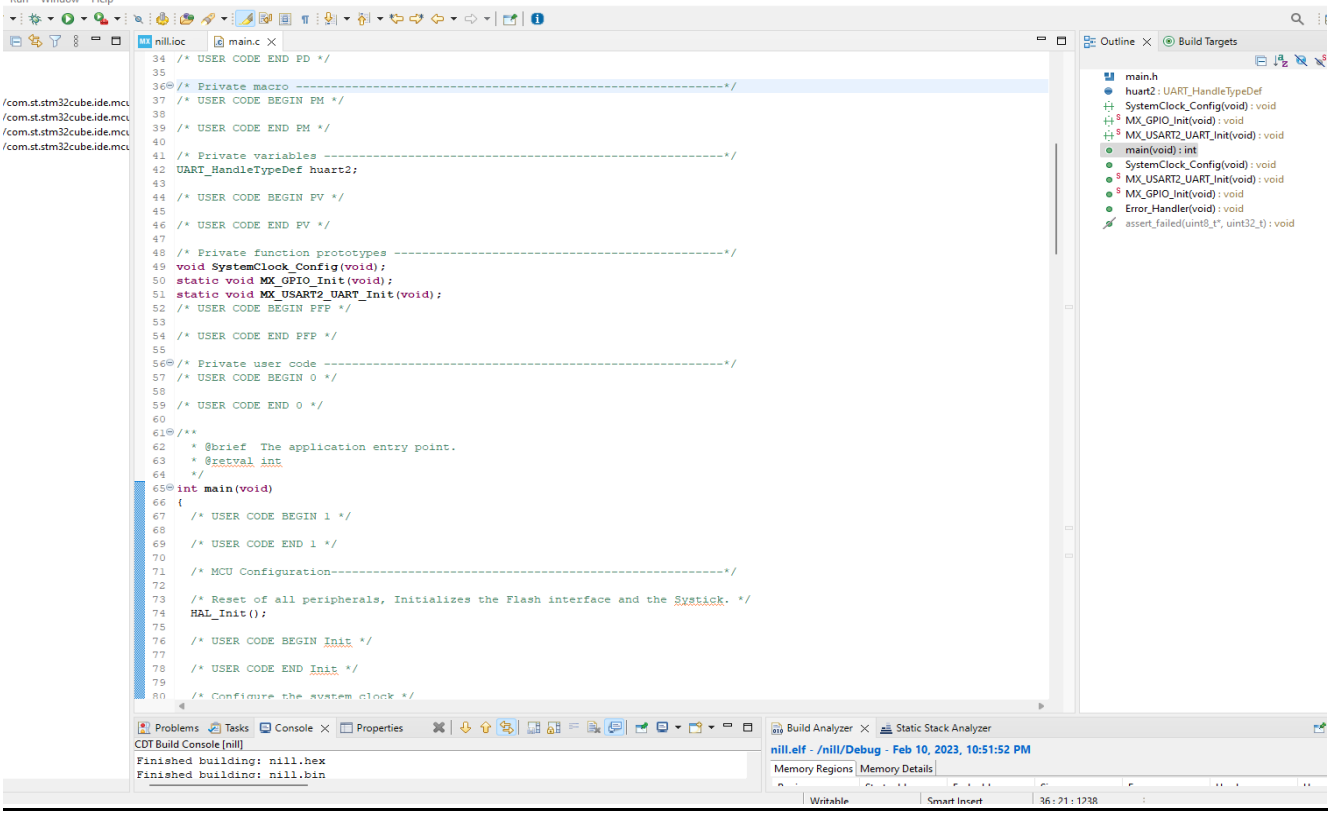
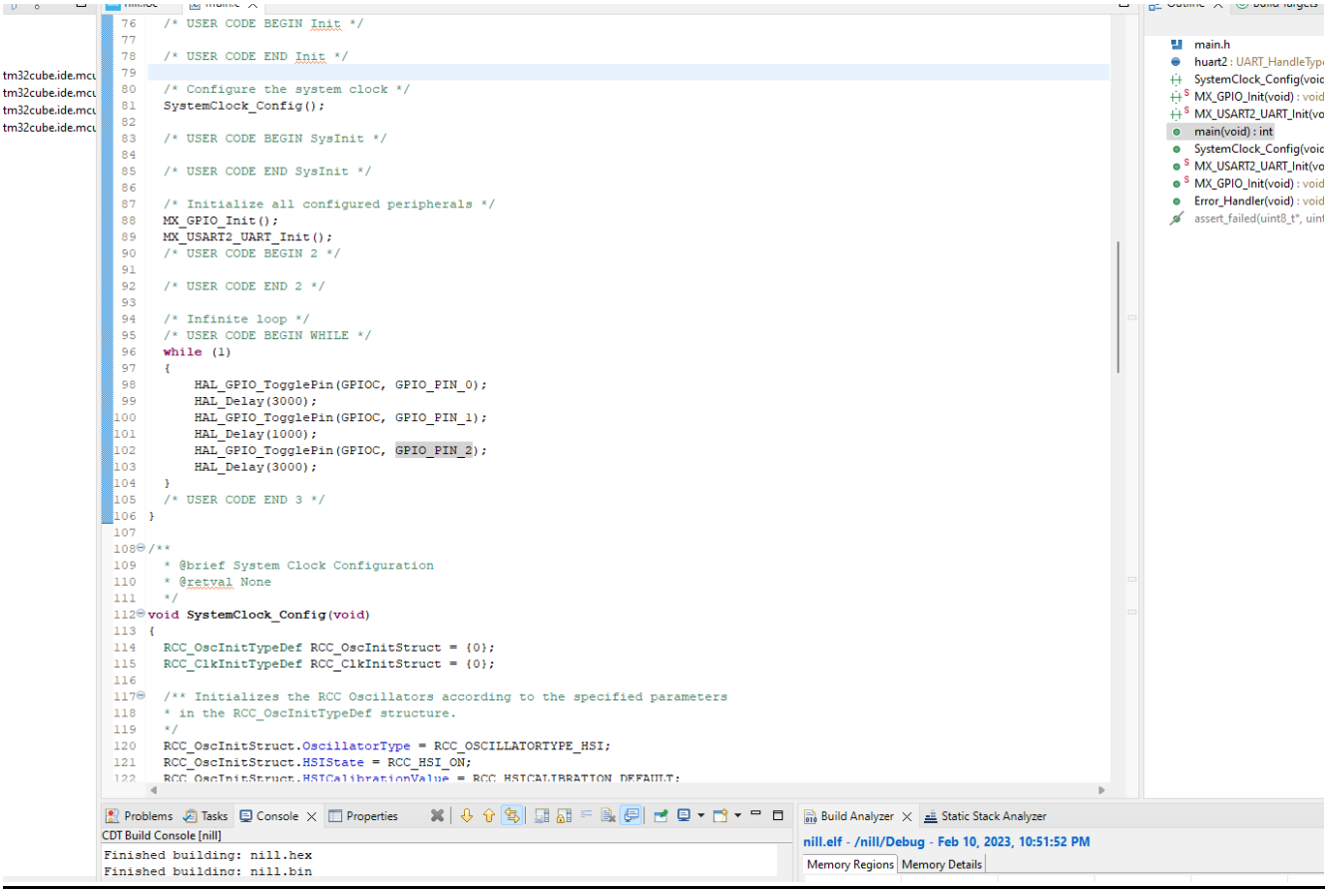


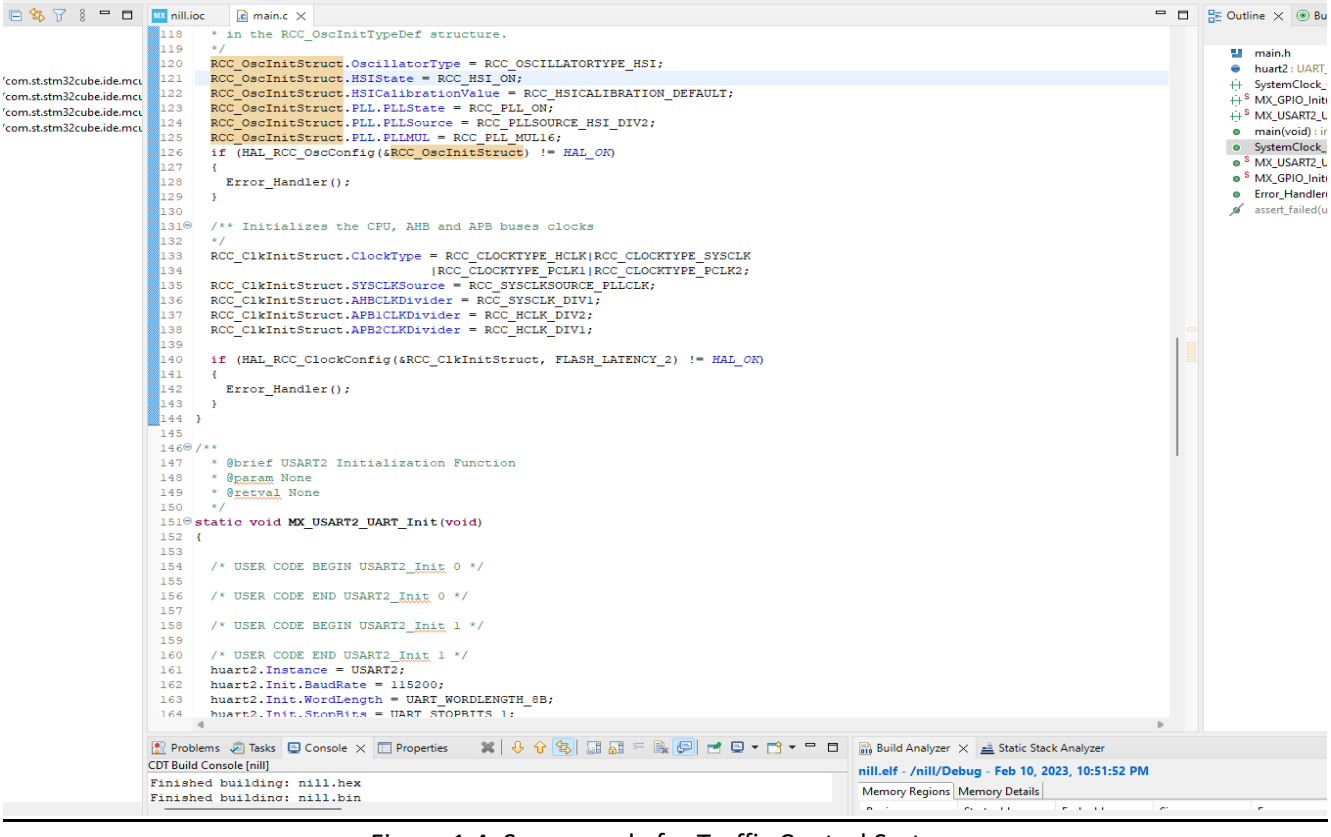

Figure 1.2: Source code for Traffic Control System



```
  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

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

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
      HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_0);
      HAL_Delay(3000);
      HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_1);
      HAL_Delay(1000);
      HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_2);
      HAL_Delay(3000);
  }
  /* USER CODE END 3 */
}

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

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
```

```
Finished building: nill.hex
Finished building: nill.bin
```

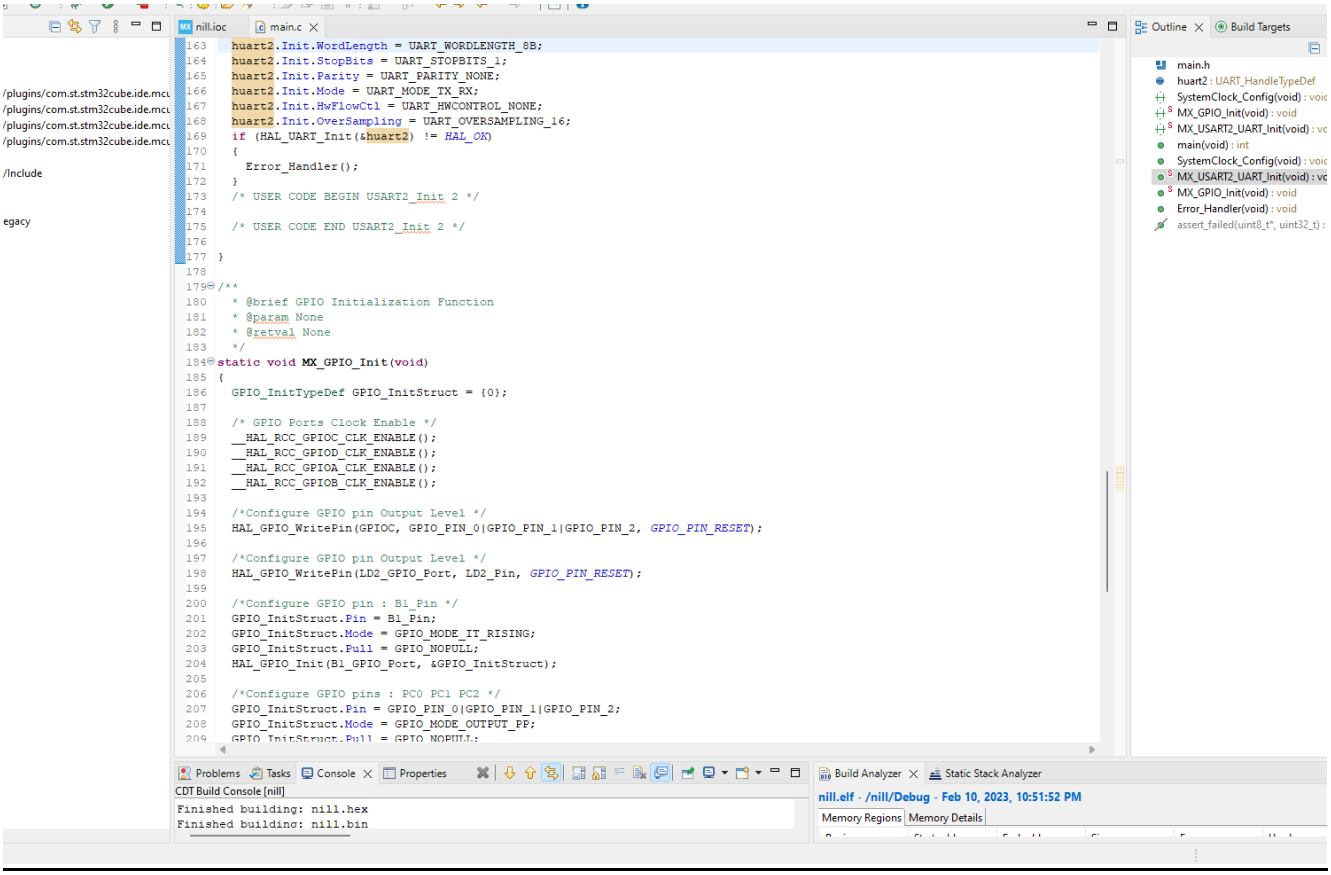
Figure 1.3: Source code for Traffic Control System



```
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI_DIV2;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL16;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
```

```
Finished building: nill.hex
Finished building: nill.bin
```

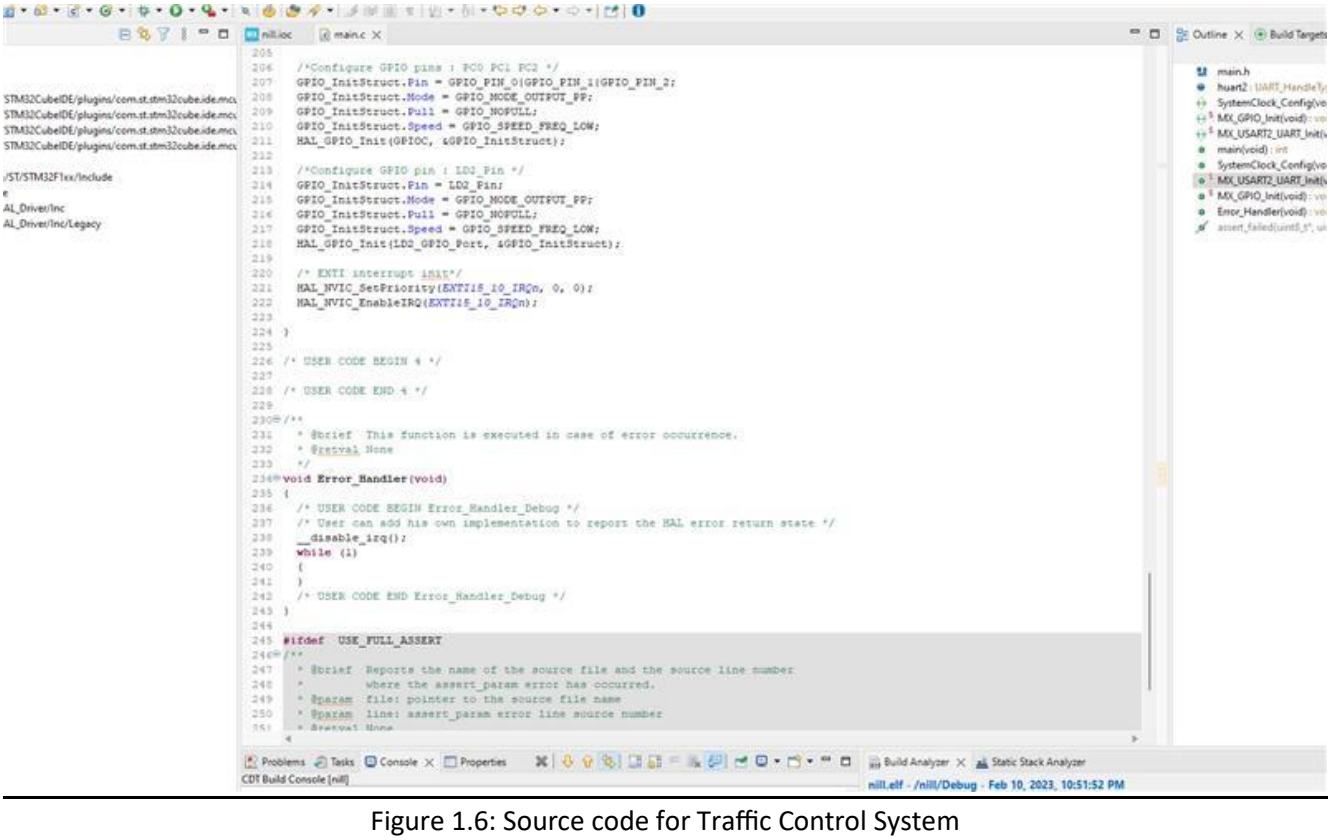
Figure 1.4: Source code for Traffic Control System



```
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

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

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : PC0 PC1 PC2 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
```

```
Finished building: nill.hex
Finished building: nill.bin
```

Figure 1.5: Source code for Traffic Control System



```
  /*Configure GPIO pins : PC0 PC1 PC2 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
```

```
CDT Build Console [nill]
nill.elf - /nill/Debug - Feb 10, 2023, 10:51:52 PM
```

Figure 1.6: Source code for Traffic Control System

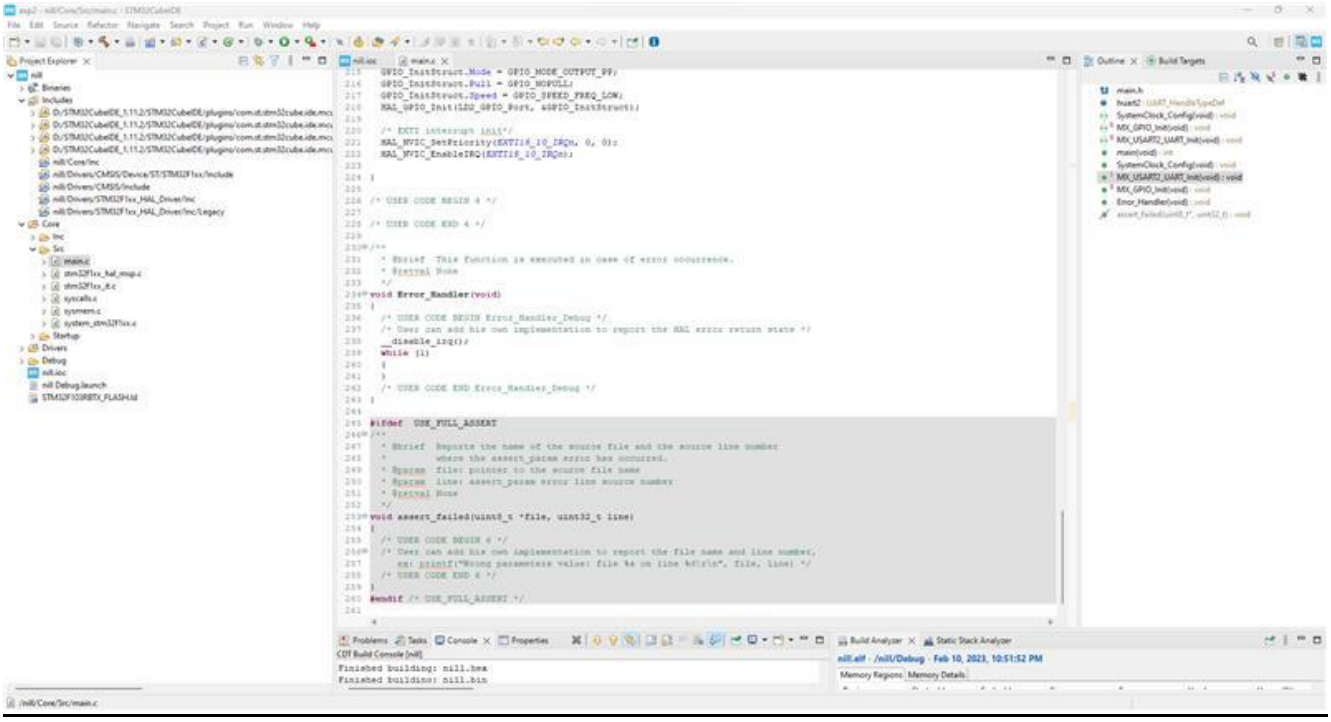


Figure 1.7: Source code for Traffic Control System

## Hardware Implementation:

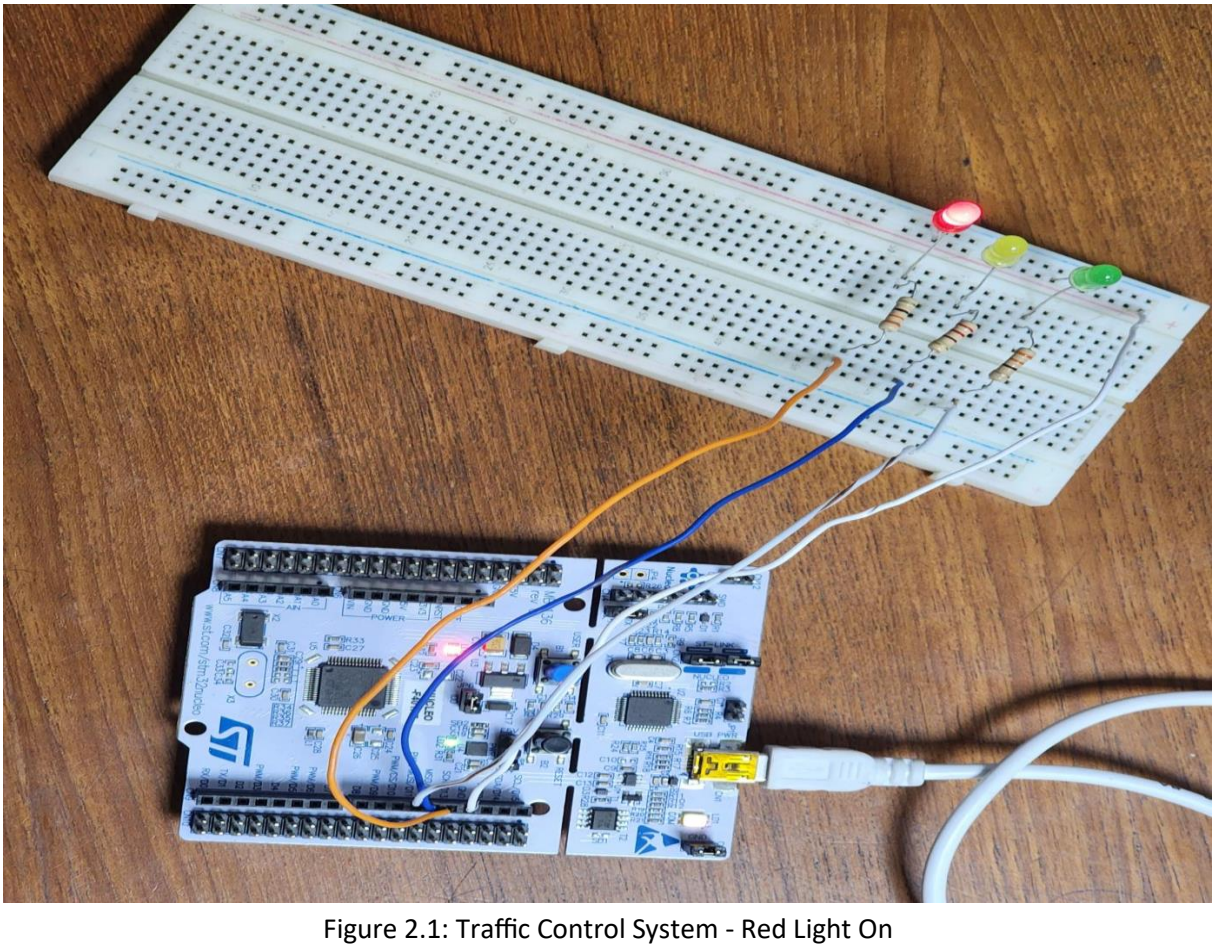


Figure 2.1: Traffic Control System - Red Light On

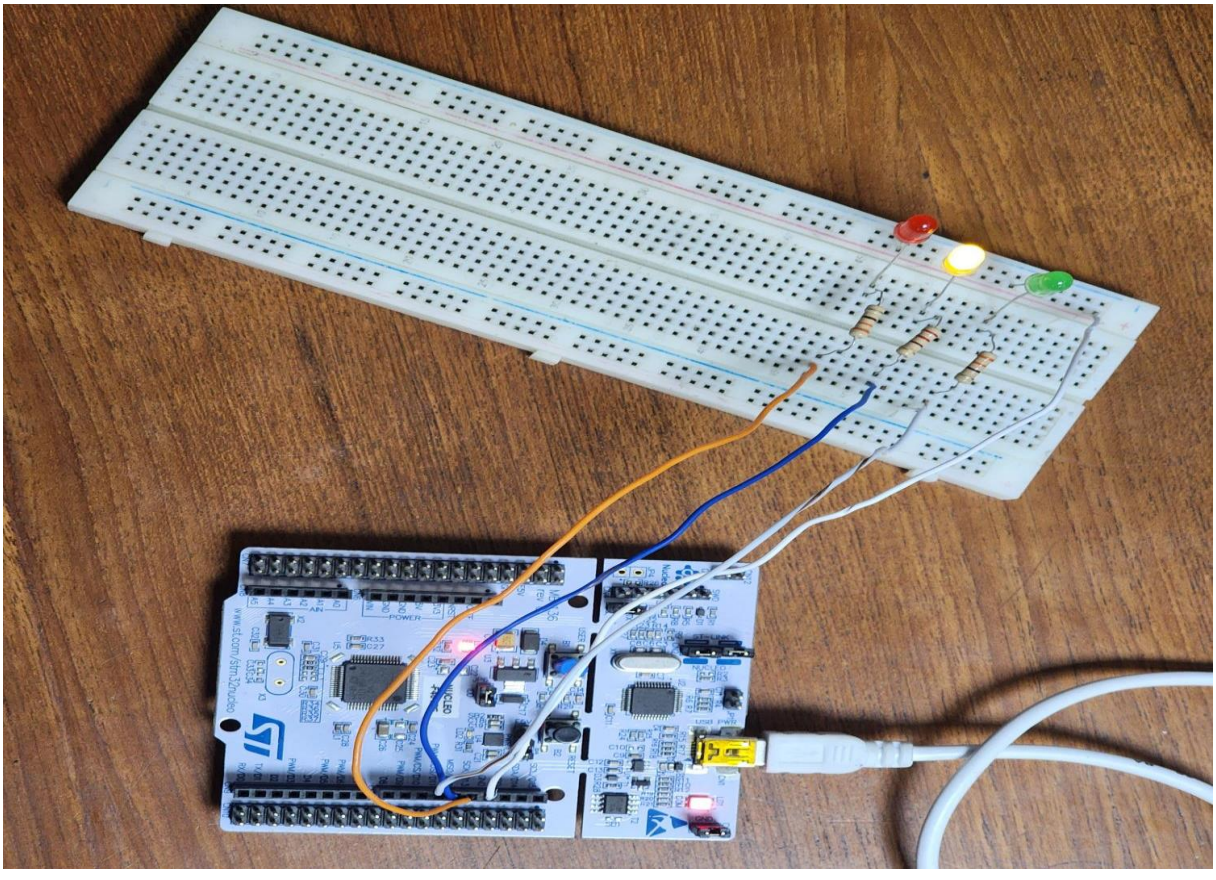


Figure 2.2: Traffic Control System - Yellow Light On

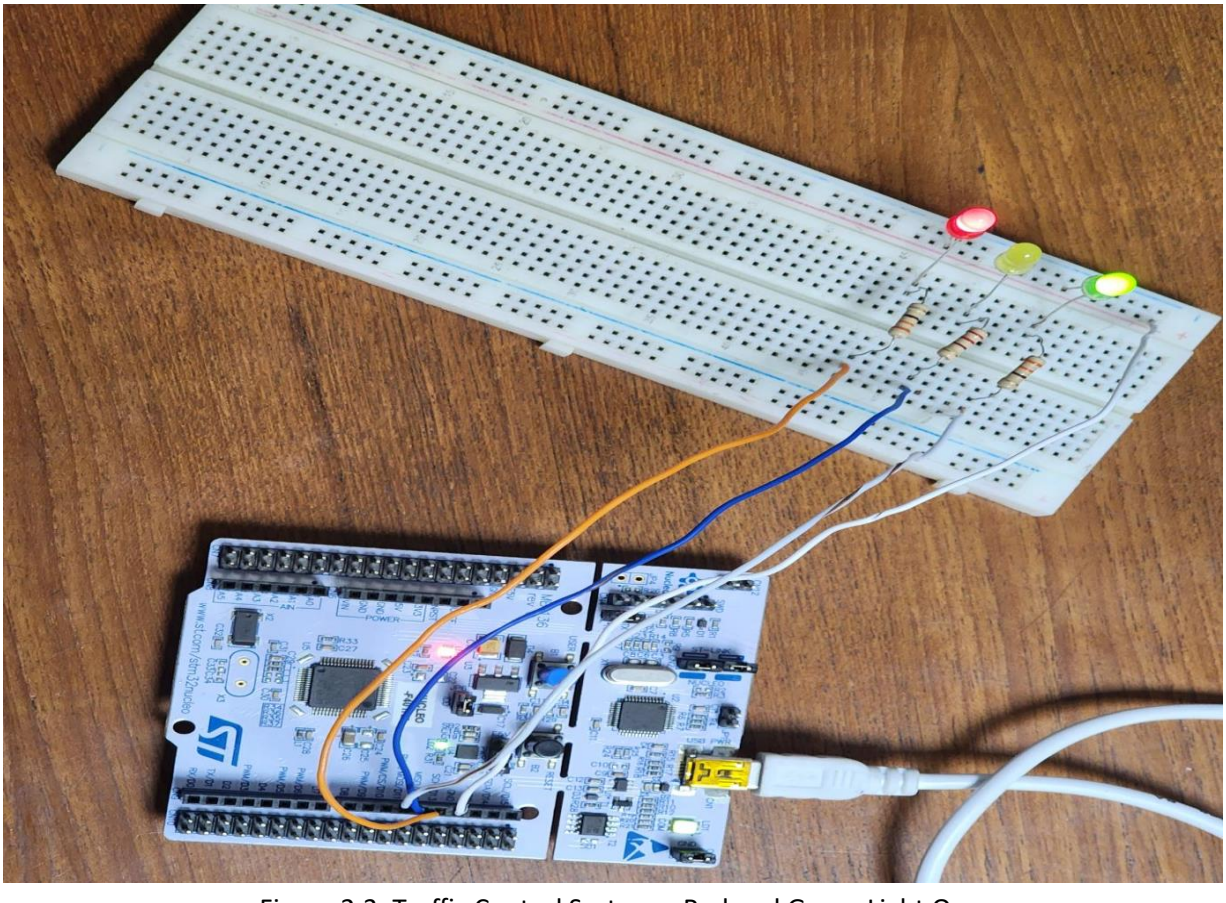
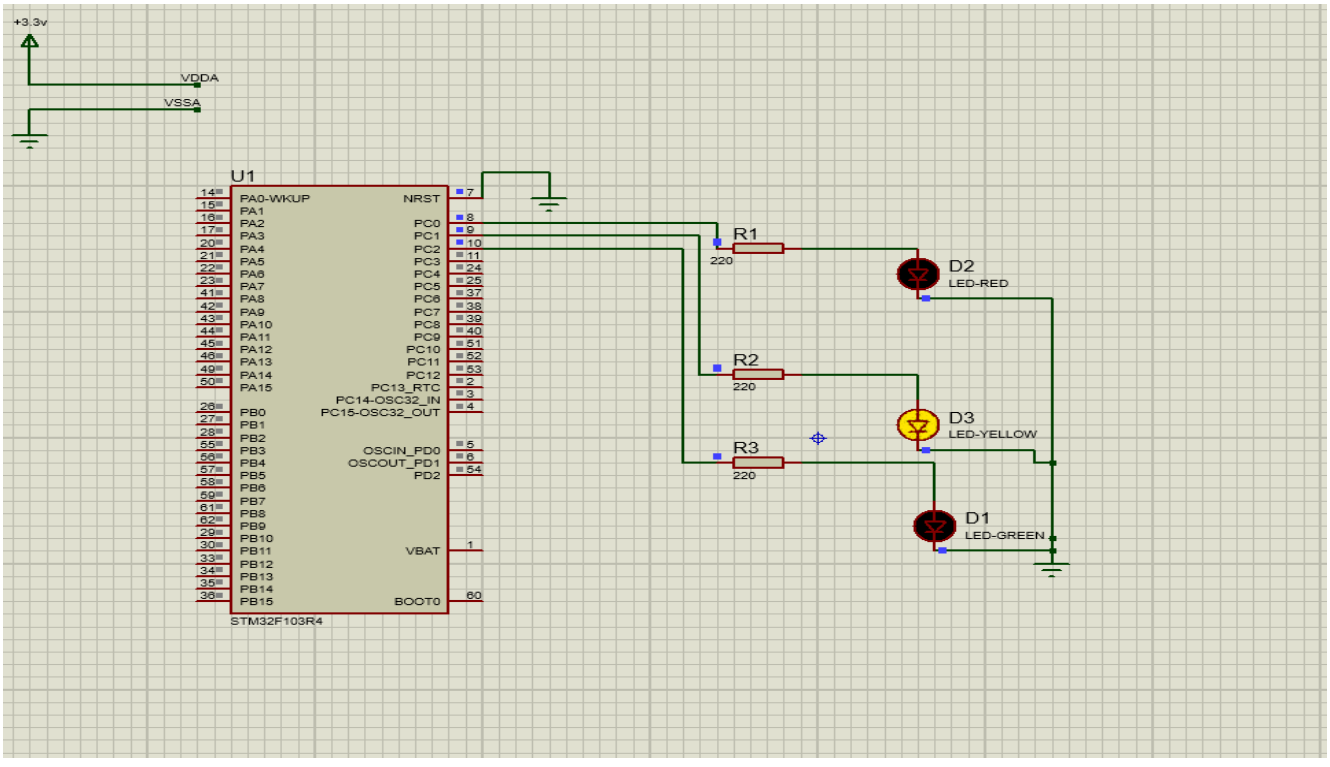
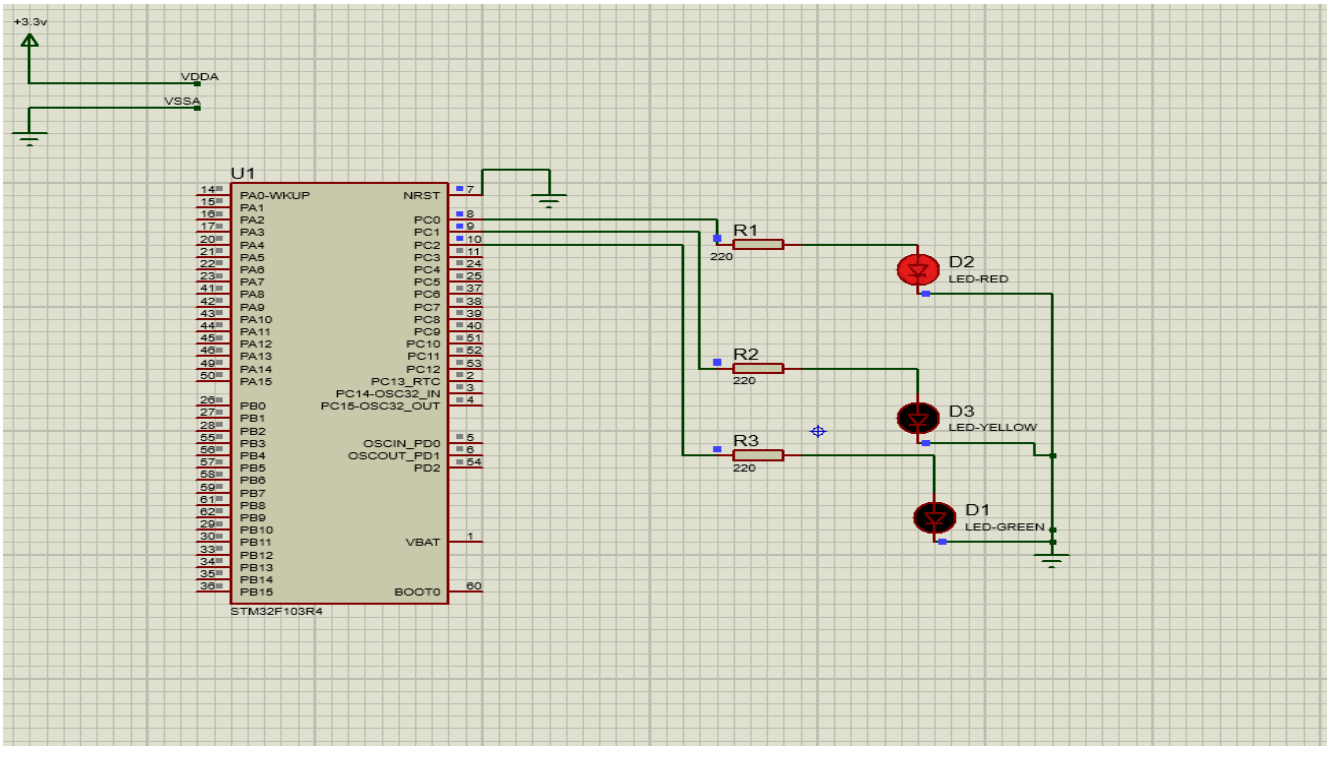
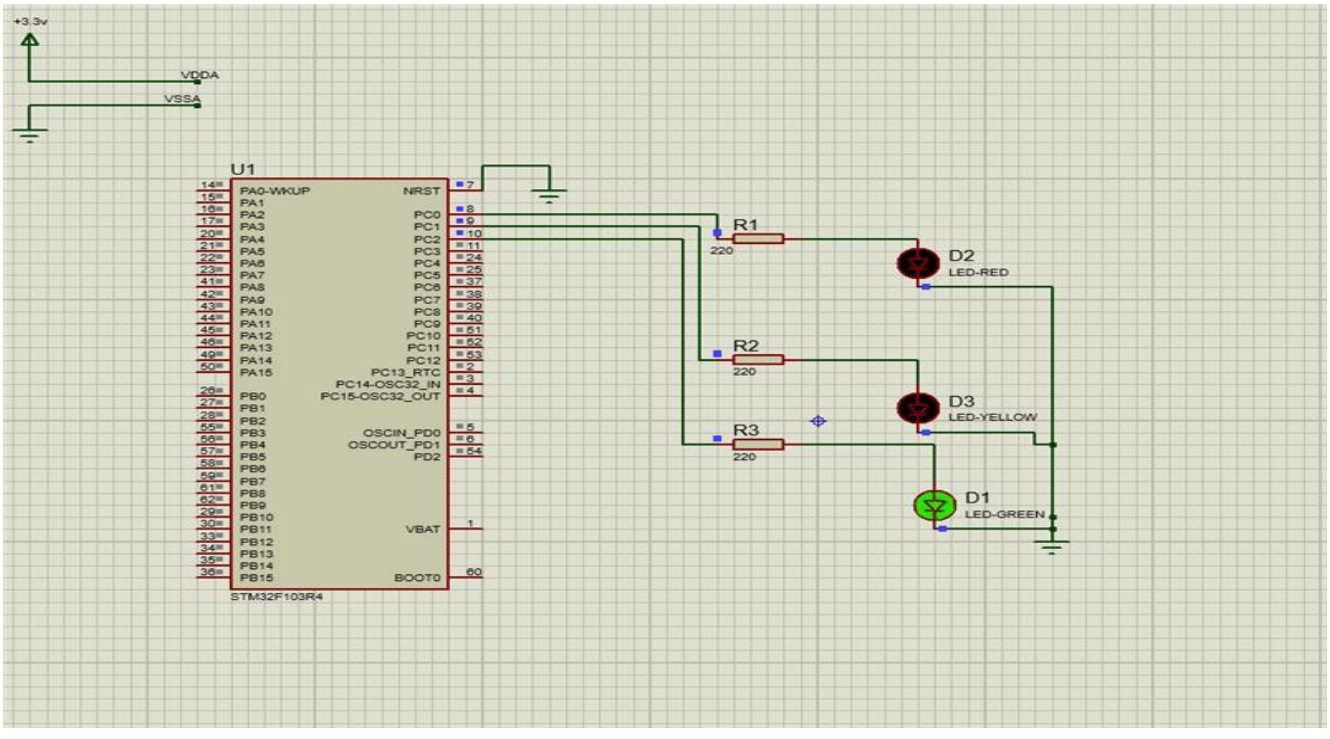


Figure 2.3: Traffic Control System – Red and Green Light On

## Simulation:


+3.3v
VDDA
VSSA
U1
PA0-WKUP
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PA8
PA9
PA10
PA11
PA12
PA13
PA14
PA15
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PB8
PB9
PB10
PB11
PB12
PB13
PB14
PB15
NRST
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
PC8
PC9
PC10
PC11
PC12
PC13_RTC
PC14-OSC32_IN
PC15-OSC32_OUT
OSCIN_PD0
OSCOUT_PD1
PD2
VBAT
BOOT0
STM32F103R4
R1
220
R2
220
R3
220
D2
LED-RED
D3
LED-YELLOW
D1
LED-GREEN


+3.3v
VDDA
VSSA
U1
PA0-WKUP
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PA8
PA9
PA10
PA11
PA12
PA13
PA14
PA15
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PB8
PB9
PB10
PB11
PB12
PB13
PB14
PB15
NRST
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
PC8
PC9
PC10
PC11
PC12
PC13_RTC
PC14-OSC32_IN
PC15-OSC32_OUT
OSCIN_PD0
OSCOUT_PD1
PD2
VBAT
BOOT0
STM32F103R4
R1
220
R2
220
R3
220
D2
LED-RED
D3
LED-YELLOW
D1
LED-GREEN




## Report:

All codes, scripts and proteus simulation of the blink program and traffic light system is attached above.

## Discussions:

In the following experiment, the primary target was to acquire knowledge on a microcontroller STM32 and assemble a demo light control system. The experiment with three LED lights (RED, GREEN and YELLOW) and STM32 board, and some wires to prepare a light system. After connecting the LEDs and wires to the board as shown in the image, it was time to connect them with the STM32 microcontroller. STM32 had 28 Pins. Wires are gained in PA5, PA6 and PA7 at the end. Once the circuit construction has be done, the USB should be inserted into the computer to develop program commands on the STM32 board. The required code was written in "while loop" which was 4 to 5 lines as shown in the screenshot of the source code section. After building and running the program from STM32 cube IDE, the board showed results to blink LED lights.

## Conclusions:

The experiment's main goal was to familiarize participants with microcontrollers. We learned about microcontrollers while working on this experiment and built a traffic control system using an STM32 microcontroller. Our course instructor walked us through the entire procedure so that we could successfully run this code. Working on STM32 was difficult at first, but it progressively became easier to manage. However, with the assistance of our teacher, the code was eventually run successfully. As a result, we concluded our report.