KNOWLEDGE INSTITUTE OF TECHNOLOGY

[Accredited by NAAC, Approved by AICTE New Delhi, Affiliated to Anna University Chennai]

NH – 47, KIOT Campus, Kakapalayam (PO), Salem – 637 504

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

Record Notebook

Course Code & Title	:	**CEC369& IoT Processors**
Year / Sem	:	**III / V**
Academic Year	:	**2024 – 2025 (ODD)**
Regulation	:	**R 2021**

NAME OF THE CANDIDATE : _____

ROLL NUMBER & SECTION : _____

REGISTER NUMBER : _____

KNOWLEDGE INSTITUTE OF TECHNOLOGY, SALEM

Department of Electrical and Electronics Engineering

Vision, Mission, PEOs, PSOs, POs and CO

Vision

- To produce technically competent Electrical and Electronics Engineers having world class skills with ethical and social values

Mission

- To provide state-of-the art facilities in Electrical and Electronics Engineering for improving the learning environment and research activities
- To continuously enrich the knowledge and skill of students towards the employment and creation of innovative products for society
- To develop ethical, social-valued and entrepreneurship skilled Electrical and Electronics Engineers

Programme Educational Objectives (PEOs)

The graduates of Electrical and Electronics Engineering will be able to

PEO – 1 Succeed in the areas of Electrical and Electronics Engineering and other diverse fields by utilizing the fundamental knowledge of engineering, analytical and creative skills

PEO – 2 Design, simulate and develop new innovative product and system in multi-disciplinary fields through applying life-long learning skill and modern tools handling ability

PEO – 3 Demonstrate communication skill, leadership qualities, ethics, team work and social responsibilities

Programme Specific Outcome (PSOs)

The graduates of Electrical and Electronics Engineering will be able to

PSO – 1 Apply current technologies in Embedded System Design for providing solution to real world Problems through smart product development

PSO – 2 Design, develop and implement software based automated system in the field of Electrical Power and Energy to meet out the demands of society and industry

PSO – 3 Analyse and diagnose the faults and defects in electrical devices and systems for Energy Management.

Programme Outcomes (POs)

The graduates of Electrical and Electronics Engineering will be able to:

PO-1 Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO-2	**Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO-3	**Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO-4	**Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO-5	**Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO-6	**The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO-7	**Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO-8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO-9	**Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams and in multidisciplinary settings.
PO-10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO-11	**Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO-12	**Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Outcomes:

At the end of the course, the student should have the

- Explain the architecture and features of ARM.
- List the concepts of exception handling.
- Write a program using ARM CORTEX M3/M4.
- Learn the architecture of STM32L15XXX ARM CORTEX M3/M4.
- Design an SoC for any application

CONTENTS

PROGRAM :

ADDRESS (Hex)	LABEL	MNEMONICS (Hex)	OPCODE	COMMENTS
0x0000		LDR r0, =0x10	E3A00010	Load the immediate value 0x10 into r0 (dividend).
0x0004		LDR r1, =0x20	E3A01020	Load the immediate value 0x20 into r1 (divisor).
0x0008		ADD r2, r0, r1	E0802001	Add r0 and r1, store result in r2 (quotient).
0x000C		STR r2, [RESULT]	E5822000	directly stores the value in r2 to the memory address
0x0010	END	B END	EAFFFFFE	Infinite loop to mark the end of the program.
0x0020	RESULT	DCD 0	00000000	Memory location reserved for storing the result.

Expected Register States

- After LDR r0, =0x10: r0 should contain 0x10.
- After LDR r1, =0x20: r1 should contain 0x20.
- After ADD r2, r0, r1: r2 should contain 0x30 (sum of 0x10 + 0x20).
- After STR r2, [result]: Memory location RESULT should contain 0x30.

Tabulation:

INPUT		OUTPUT	
ADDRESS	**DATA**	**ADDRESS**	**DATA**

EX NO :01 DATE :	ARM ASSEMBLY PROGRAM TO ADD TWO NUMBERS

AIM:

To write a program to add two 32-bit numbers stored in R0 and R1 registers and write the result to R2. The result is stored to a memory location.

a) Run the program with breakpoint and verify the result

b) Run the program with stepping and verify the content of registers at each stage.

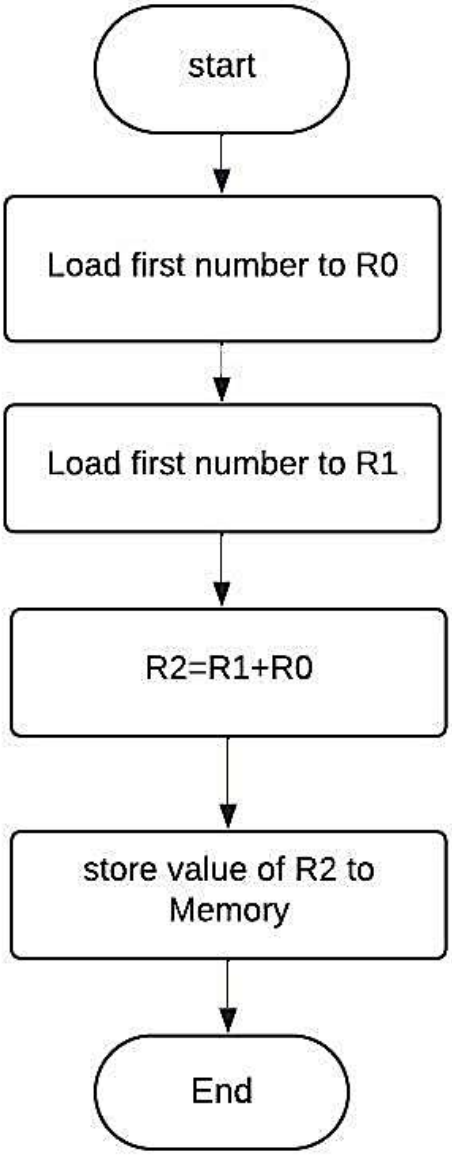
APPARATUS REQUIRED

S.NO	APPRATAUS NAME	RANGE	QUANTITY
1			
2			

PROCEDURE:

Run the Program with Breakpoint and Verify the Result

1. **Set a breakpoint** at the line where the ADD instruction is located. This is where r2 will get the result of r0 + r1.
2. **Run the program** in your debugger.
3. When the program hits the **breakpoint**, verify the values of r0, r1, and r2:
    - r0 should contain the first operand.
    - r1 should contain the second operand.
    - r2 should have the result of the addition.
4. **Continue execution** until the program finishes.
5. Verify that the memory location RESULT contains the sum of the two numbers.


start
Load first number to R0
Load first number to R1
R2=R1+R0
store value of R2 to Memory
End

INFERENCE:

RESULT:

Faculty Signature

PROGRAM : (ARM Assembly with UDIV Instruction)

ADDRESS (Hex)	LABEL	MNEMONICS (Hex)	OPCODE	COMMENTS
0x0000		MOV r3, #0	E3A03000	Initialize r3 (quotient) to 0.
0x0004		CMP r2, #0	E3520000	Compare r2 (divisor) with 0 to avoid division by zero.
0x0008		BEQ END	0A000005	Branch to END if r2 is zero.
0x000C	DIV_LOOP	CMP r1, r2	E1510002	Compare r1 (dividend) with r2 (divisor).
0x0010		BLT END_DIV	BA000003	Branch if r1 < r2 (exit loop).
0x0014		SUB r1, r1, r2	E0411002	Subtract r2 (divisor) from r1 (dividend).
0x0018		ADD r3, r3, #1	E2833001	Increment r3 (quotient).
0x001C		B DIV_LOOP	EAFFFFFA	Branch to DIV_LOOP to continue division.
0x0020	END_DIV	MOV r5, r1	E1A05001	Store the remaining r1 as r5 (remainder).
0x0024	END	B END	EAFFFFFE	Infinite loop to mark the end of the program

ARM Assembly without UDIV (Using Repeated Subtraction)

ADDRESS (Hex)	LABEL	MNEMONICS (Hex)	OPCODE	COMMENTS
0x0000		MOV r3, #0	E3A03000	Initialize r3 (quotient) to 0.
0x0004		CMP r2, #0	E3520000	Compare r2 (divisor) with 0 to avoid division by zero.
0x0008		BEQ END	0A000005	Branch to END if r2 is zero.
0x000C	DIV_LOOP	CMP r1, r2	E1510002	Compare r1 (dividend) with r2 (divisor).
0x0010		BLT END_DIV	BA000003	Branch to END_DIV if r1 < r2.
0x0014		SUB r1, r1, r2	E0411002	Subtract r2 (divisor) from r1 (dividend).
0x0018		ADD r3, r3, #1	E2833001	Increment r3 (quotient).
0x001C		B DIV_LOOP	EAFFFFFA	Repeat the loop until r1 < r2.
0x0020	END_DIV	MOV r5, r1	E1A05001	Store the remaining r1 as r5 (remainder).
0x0024	END	B END	EAFFFFFE	Infinite loop to mark the end of the program.

EX NO : 02 DATE :	ARM ASSEMBLY TO DIVISION TWO NUMBERS

AIM:

Write ARM assembly to perform the function of division. Registers r1 and r2 contain the dividend and divisor, r3 contains the quotient, and r5 contains the remainder.

APPARATUS REQUIRED

S.NO	APPRATAUS NAME	RANGE	QUANTITY
1			
2			

PROCEDURE:

ARM Assembly with UDIV Instruction

UDIV r3, r1, r2:

- UDIV performs an unsigned integer division of r1 by r2 and stores the quotient in r3.
- Example: If r1 = 10 and r2 = 3, then r3 will contain 3.

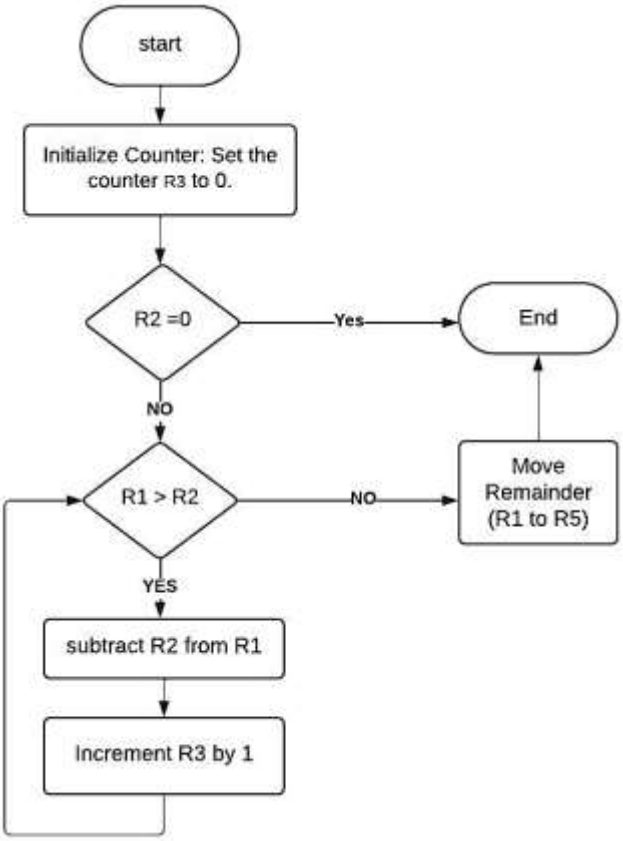
MLS r5, r3, r2, r1:

- MLS (Multiply and Subtract) calculates the remainder.
- It multiplies r3 (the quotient) by r2 (the divisor) and subtracts the result from r1 (the dividend).
- This stores the remainder in r5.
- Example: r5 = 10 - (3 * 3) = 1.

ARM Assembly without UDIV (Using Repeated Subtraction)

Explanation:

1. **MOV r3, #0**: Initialize r3 (quotient) to 0.
2. **CMP r2, #0**: Check if the divisor is zero to avoid a divide-by-zero error.
3. **BLT END_DIV**: Branch if r1 (dividend) is less than r2 (divisor), as the division process should stop when the dividend is smaller than the divisor.
4. **SUB r1, r1, r2**: Subtract r2 (divisor) from r1 (dividend) to reduce the dividend.
5. **ADD r3, r3, #1**: Increment the quotient.
6. **B DIV_LOOP**: Repeat the process until r1 becomes smaller than r2.
7. **MOV r5, r1**: Store the remaining r1 as the remainder in r5.



Flowchart : (ARM Assembly with UDIV Instruction)

Tabulation:

INPUT		OUTPUT	
ADDRESS	**DATA**	**ADDRESS**	**DATA**

Tabulation:

INPUT		OUTPUT	
ADDRESS	**DATA**	**ADDRESS**	**DATA**

INFERENCE:

RESULT:

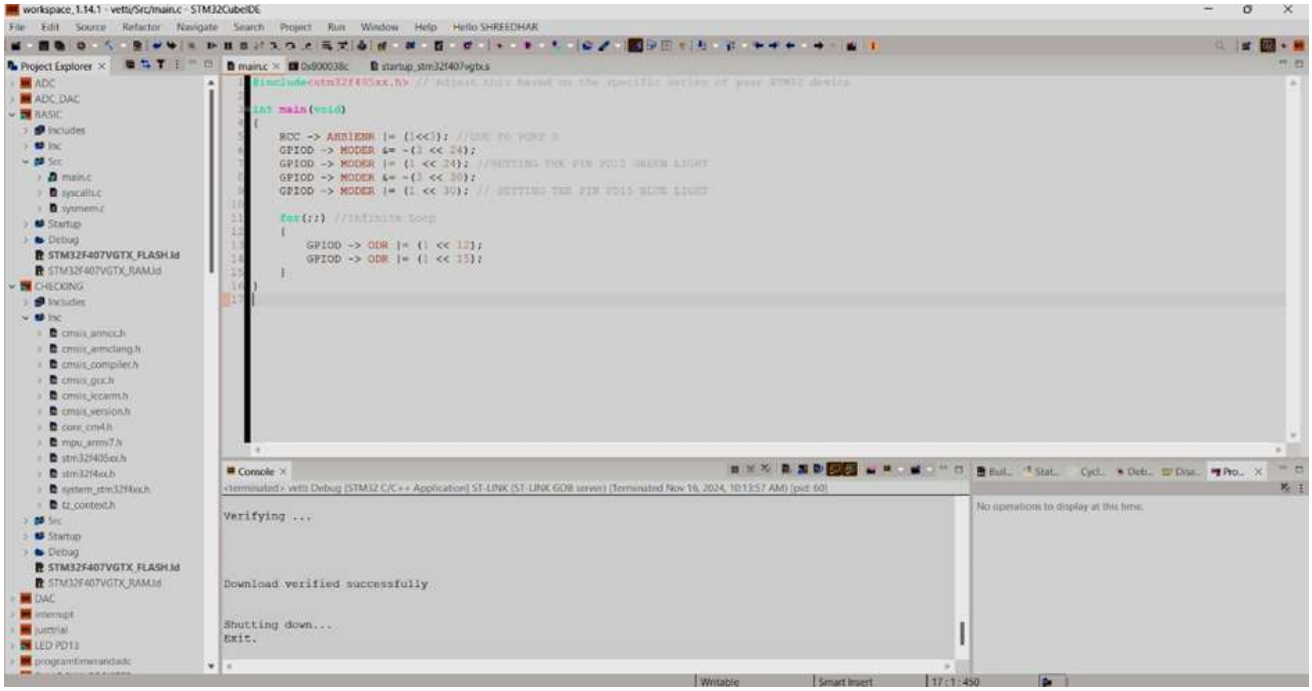
Faculty Signature

Program: Turn On Green and Blue LEDs on STM32LDiscovery

```
#include<stm32f405xx.h> // Adjust this based on the specific series of your STM32 device

int main(void)
{
	RCC -> AHB1ENR |= (1<<3); //DUE TO PORT D
	GPIOD -> MODER &= ~(3 << 24);
	GPIOD -> MODER |= (1 << 24); //SETTING THE PIN PD12 GREEN LIGHT
	GPIOD -> MODER &= ~(3 << 30);
	GPIOD -> MODER |= (1 << 30); // SETTING THE PIN PD15 BLUE LIGHT

	for(;;) //Infinite Loop
	{
		GPIOD -> ODR |= (1 << 12);
		GPIOD -> ODR |= (1 << 15);
	}
}
```



STM IDE

EX NO: 03 DATE :	PROGRAM TO TURN ON LED USING ON STM32

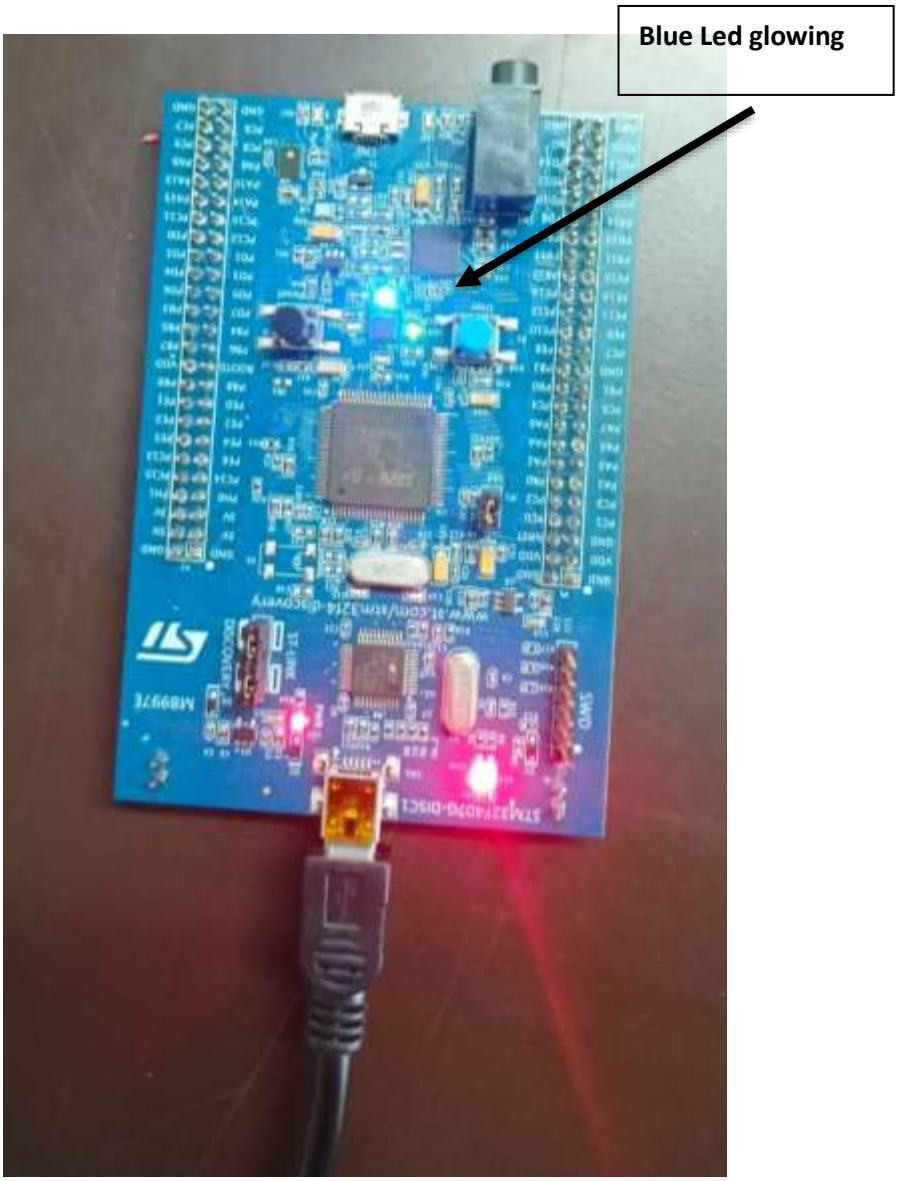
AIM:

Write a program to turn on green LED (Port B.6) and Blue LED (Port B.7) on STM32L Discovery by configuring GPIO.

APPARATUS REQUIRED

S.NO	APPRATAUS NAME	RANGE	QUANTITY
1			
2			

PROCEDURE:


Blue Led glowing

INFERENCE:

RESULT:

Faculty Signature

Program: Transmit a string using on Stm32

```
#include "stm32f4xx.h"  // Include the header file for STM32F407

void usart2_init(void);
void usart2_write(char ch);
void usart2_write_string(const char *str);
void led_init(void);
void led_on(void);
void led_blink(void);

int main(void) {
   // Initialize USART2 and LED
   usart2_init();
   led_init();

   // Transmit the string with LED blinking during transmission
   usart2_write_string("Programming with ARM Cortex");

   // After transmission, turn on the PD12 LED
   led_on();

   while (1) {
      // Infinite loop to keep the PD12 LED on
   }
}

void usart2_init(void) {
   // Enable clock for GPIOA, GPIOD, and USART2
   RCC->AHB1ENR |= (1 << 0);    // Enable GPIOA clock (for USART2 TX)
   RCC->AHB1ENR |= (1 << 3);    // Enable GPIOD clock (for LEDs)
   RCC->APB1ENR |= (1 << 17);   // Enable USART2 clock

   // Configure PA2 (USART2 TX) as Alternate Function
   GPIOA->MODER &= ~(0x3 << 4);  // Clear mode bits for PA2
   GPIOA->MODER |= (0x2 << 4);   // Set PA2 to Alternate Function mode
   GPIOA->AFR[0] |= (0x7 << 8);  // Set AF7 (USART2) for PA2

   // Configure USART2
   USART2->BRR = 0x0683;        // Set baud rate to 9600 (assuming 16 MHz clock)
   USART2->CR1 |= (1 << 3);     // Enable transmitter
   USART2->CR1 |= (1 << 13);    // Enable USART2
}

void usart2_write(char ch) {
   while (!(USART2->SR & 0x80)); // Wait until TXE (Transmit data register empty) is set
   USART2->DR = ch;             // Write character to data register
}

void usart2_write_string(const char *str) {
   while (*str) {
      usart2_write(*str++);     // Transmit each character
```

EX NO: 04 DATE :	PROGRAM TRANSMIT A STRING USING ON STM32

AIM:

Transmit a string “Programming with ARM Cortex” to PC by configuring the registers of USART2. Use polling method.

APPARATUS REQUIRED

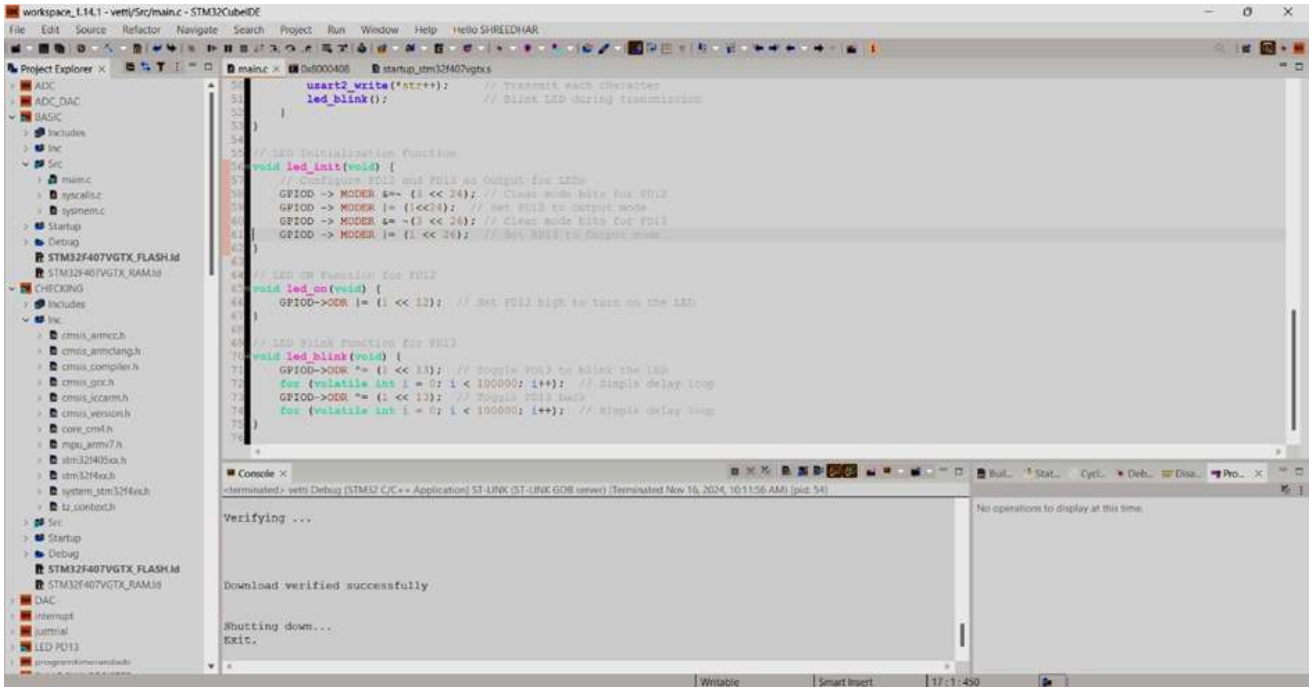
S.NO	APPRATAUS NAME	RANGE	QUANTITY
1			
2			

```
        led_blink();            // Blink LED during transmission
    }
}

// LED Initialization Function
void led_init(void) {
    // Configure PD12 and PD13 as Output for LEDs
            GPIOD -> MODER &=~ (3 << 24); // Clear mode bits for PD12
            GPIOD -> MODER |= (1<<24);  // Set PD12 to Output mode
    GPIOD -> MODER &= ~(3 << 26); // Clear mode bits for PD13
    GPIOD -> MODER |= (1 << 26);  // Set PD13 to Output mode
}

// LED ON Function for PD12
void led_on(void) {
    GPIOD->ODR |= (1 << 12);  // Set PD12 high to turn on the LED
}

// LED Blink Function for PD13
void led_blink(void) {
    GPIOD->ODR ^= (1 << 13);  // Toggle PD13 to blink the LED
    for (volatile int i = 0; i < 100000; i++);  // Simple delay loop
    GPIOD->ODR ^= (1 << 13);  // Toggle PD13 back
    for (volatile int i = 0; i < 100000; i++);  // Simple delay loop
}
```



STM IDE

INFERENCE:

RESULT:

Faculty Signature

PROGRAM: Toggle the Led using on Stm32

```
#include <stm32f405xx.h>
void ms_delay_tim2(uint32_t n)
{
	RCC -> APB1ENR |= (1<<0); // ENABLE THE BUS (174), timer 2 is in 0th position
	TIM2 -> PSC = 16000-1;// Prescaler value , THAT (-1) indicates that the 0 to 15999 (642)
	TIM2 -> ARR = 1000-1;// total speed is 16MHZ , ((16MHZ/16000))=1000.same (-1) indicates from 0 to
999(642)
	TIM2 -> CR1 |= (1<<0);//enable the timer(627), timer 2 in 0th position
	for(uint32_t i=0 ; i<n ; i++)
	{
		while(!(TIM2 -> SR & (1<<0))); //WHEN BOTH 1 IT TERMINATES
		TIM2 -> SR &=~ (1<<0);
		}
}
int main(void)
{
	RCC -> AHB1ENR |= (1<<3);
	GPIOD -> MODER &=~ (3 << 24);
	GPIOD -> MODER |= (1<<24);//GREEN LIGHT PD12
	GPIOD -> MODER &= ~(3 << 30);
	GPIOD -> MODER |= (1 << 30);// SETTING THE PIN PD15 BLUE LIGHT
	GPIOD -> MODER &= ~(3 << 28);
	GPIOD -> MODER |= (1 << 28);// SETTING THE PIN PD14 RED LIGHT
	GPIOD -> MODER &= ~(3 << 26);
	GPIOD -> MODER |= (1 << 26);// SETTING THE PIN PD15 ORANGE LIGHT


while(1)
	{
		GPIOD -> ODR ^=(1<<12);//^ for toggle
		GPIOD -> ODR ^=(1<<13);
		GPIOD -> ODR ^=(1<<14);
		GPIOD -> ODR ^=(1<<15);
		ms_delay_tim2(1);//n=10
	}
}
```

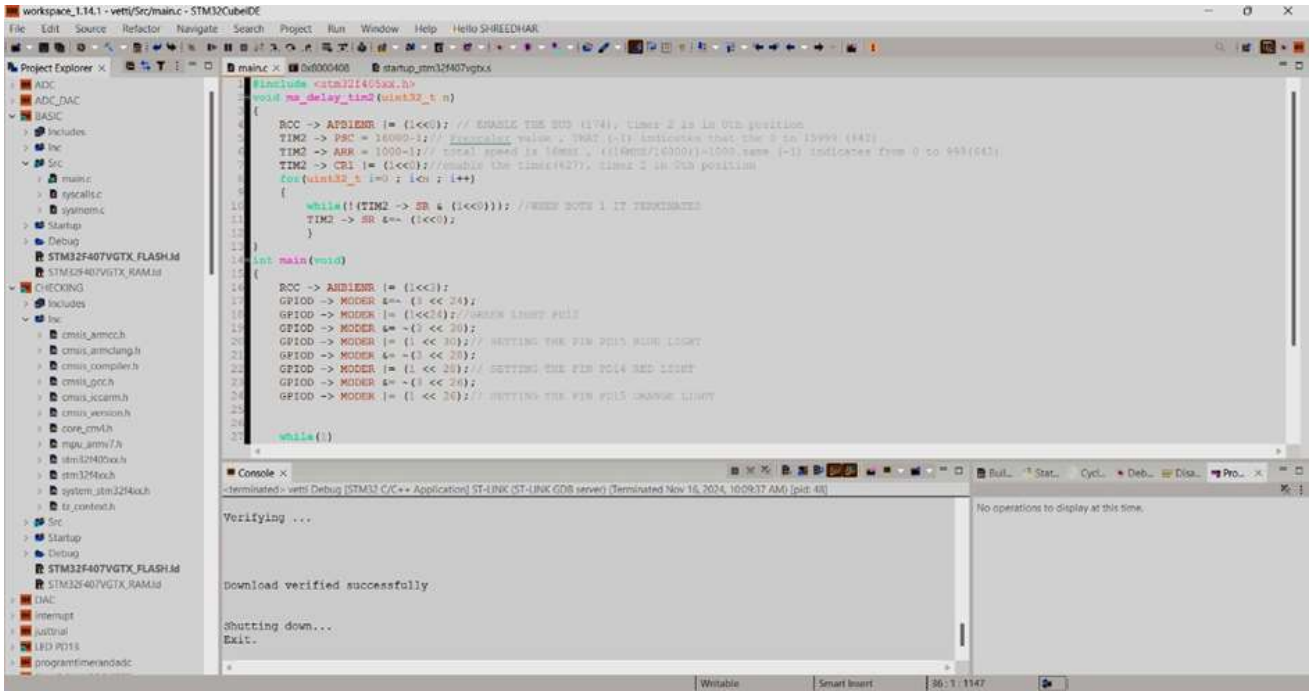
EX NO : 05 DATE :	PROGRAM TO TOGGLE THE LED USING ON STM32

AIM:

Write a program to toggle the LEDs at the rate of 1 sec using standard peripheral library. Use Timer3 for Delay.

APPARATUS REQUIRED

S.NO	APPRATAUS NAME	RANGE	QUANTITY
1			
2			



STM IDE

LED STATUS

 Includes and GPIO Configuration:

- Include stm32l1xx.h for STM32L series (adjust the header file if using a different series).
- GPIO_Config() sets PB6 and PB7 as output pins for the green and blue LEDs.

 Timer3 Configuration:

- Timer3_Config() enables the clock for Timer3 using RCC->APB1ENR.
- Configures the prescaler (TIM3->PSC) to divide the 16 MHz clock by 16,000, resulting in a 1 kHz clock (1 ms tick).
- Sets the auto-reload register (TIM3->ARR) to 1000, giving a 1-second delay.
- Enables the Timer3 update interrupt with TIM3->DIER |= TIM_DIER_UIE.
- Enables Timer3 and configures it to start counting.

 Interrupt Service Routine (ISR):

- TIM3_IRQHandler() is triggered whenever Timer3 reaches the ARR value (1 second).
- Checks the update interrupt flag (TIM3->SR & TIM_SR_UIF) to ensure the interrupt is triggered by the timer update.
- Toggles the LED states using GPIOB->ODR ^= (1 << 6) for the green LED and GPIOB->ODR ^= (1 << 7) for the blue LED.
- Clears the interrupt flag after toggling the LEDs.

 Main Loop:

- The while(1) loop is empty because the LED toggling is managed by the Timer3 interrupt.

PROGRAM: Transmit a string using on Stm32

```
#include <stm32f405xx.h>
#include <string.h>

void USART3_init(void);
void USART3_write(uint8_t ch);
void USART3_write_string(const char* str);
void delayMs(int);
void LED_init(void);
void LED_on_red(void);
void LED_off_red(void);
void LED_on_orange(void);
void LED_off_orange(void);

int main(void)
{
   USART3_init(); /* Initialize USART3 */
   LED_init(); /* Initialize LEDs */

   // Transmit the string over USART3
   LED_on_red(); // Turn on red LED to indicate transmission
   USART3_write_string("Programming with ARM Cortex\n"); // Send string
   LED_off_red(); // Turn off red LED after transmission
   LED_on_orange(); // Turn on orange LED after transmission

   while (1)
   {
      // Main loop can do other tasks
      delayMs(1000); // Delay to avoid busy looping
   }
}

void USART3_init(void)
{
   RCC->AHB1ENR |= (1 << 2); // Enable GPIOC clock
   RCC->APB1ENR |= (1 << 18); // Enable USART3 clock

   // Configure PC10 as TX for USART3
   GPIOC->MODER &= ~(3 << 20); // Clear mode for PC10
   GPIOC->MODER |= (1 << 21); // Set PC10 to alternate function mode
   GPIOC->AFR[1] &= ~(0xF << 8); // Clear alternate function for PC10
   GPIOC->AFR[1] |= (7 << 8); // Set AF7 for PC10

   // Configure PC11 as RX for USART3
   GPIOC->MODER &= ~(3 << 22); // Clear mode for PC11
   GPIOC->MODER |= (1 << 23); // Set PC11 to alternate function mode
   GPIOC->AFR[1] &= ~(0xF << 12); // Clear alternate function for PC11
   GPIOC->AFR[1] |= (7 << 12); // Set AF7 for PC11

   // USART3 settings
   USART3->BRR = 0x0683; // 9600 baud rate at 16MHz
   USART3->CR1 |= (0xC << 0); // Enable RE and TE bits
```

INFERENCE:

RESULT:

Faculty Signature

EX NO : 06 DATE :	PROGRAM TRANSMIT A STRING USING ON STM32

AIM:

Transmit a string “Programming with ARM Cortex” to PC by using standard peripheral library with the help of USART3. Use polling method.

APPARATUS REQUIRED

S.NO	APPRATAUS NAME	RANGE	QUANTITY
1			
2			

```
USART3->CR1 |= (1 << 6); // Enable RX
   USART3->CR2 = 0; // 1 stop bit
   USART3->CR3 = 0; // Default settings
   USART3->CR1 |= (1 << 13); // Enable USART3
}
void USART3_write(uint8_t ch)
{
   // Write the character to USART data register
   USART3->DR = ch;
   while (!(USART3->SR & (1 << 7))); // Wait until TXE is set
}
void USART3_write_string(const char* str)
{
   while (*str) // Transmit until null terminator
   {
      USART3_write(*str++); // Send each character
   }
}
void LED_init(void)
{
   RCC->AHB1ENR |= (1 << 3); // Enable GPIOD clock
   // Configure PD14 (Red LED) as output
   GPIOD->MODER &= ~(3 << 28); // Clear mode for PD14
   GPIOD->MODER |= (1 << 28); // Set PD14 to output mode
   // Configure PD13 (Orange LED) as output
   GPIOD->MODER &= ~(3 << 26); // Clear mode for PD13
   GPIOD->MODER |= (1 << 26); // Set PD13 to output mode
}
void LED_on_red(void)
{
   GPIOD->ODR |= (1 << 14); // Set PD14 high to turn on red LED



void LED_off_red(void)
{
   GPIOD->ODR &= ~(1 << 14); // Set PD14 low to turn off red LED
}
void LED_on_orange(void)
{
   GPIOD->ODR |= (1 << 13); // Set PD13 high to turn on orange LED
}
void LED_off_orange(void)
{
   GPIOD->ODR &= ((1 << 13)); // Set PD13 low to turn off orange LED
}
void delayMs(int n)
{
   int i;
   for (; n > 0; n--)
      for (i = 0; i < 2000; i++); // Simple delay loop
}
}
```

INFERENCE:

RESULT:

Faculty Signature