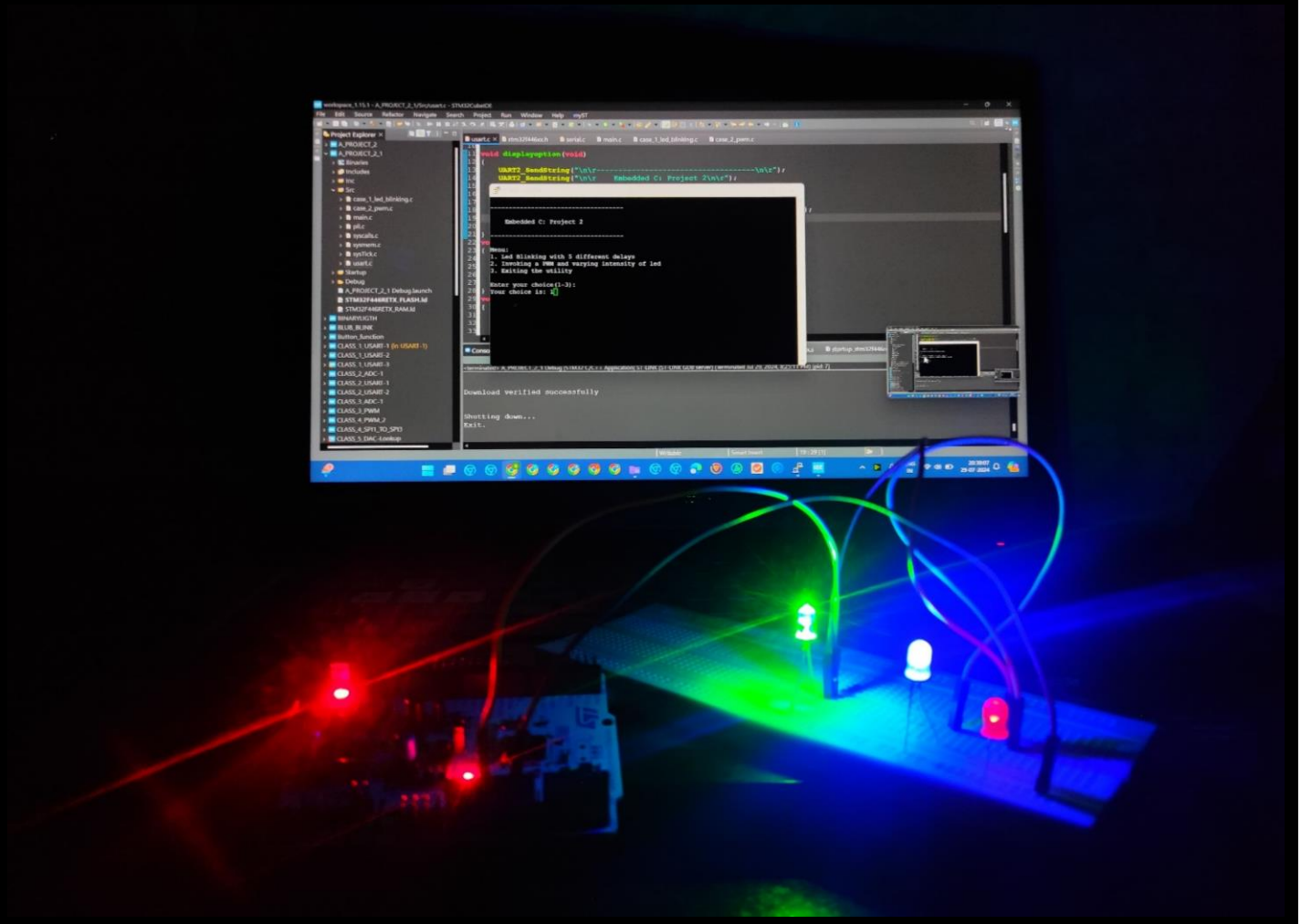
C. V. RAMAN GLOBAL UNIVERSITY

Bhubaneswar, Mahura-752052

PROJECT 2



Made By:

Group – 21

Team Members:

Sofia Akhtar (2201020130)

Sugam Shaw (2201020139)

B. Tech (CSE)

5th Semester

Under the guidance of Md. Salahuddin in CDAC LAB

Problem Statement

Project 2

Implement a menu driven utility using uart Rx, where based on what is entered it should control various operations as listed below.

1. Should invoke led blinking but should keep changing the delay of the LED blinking by press on Button (PC13).

2. Should invoke a PWM and varying intensity of light based on button pressed.

3. Exit the utility.

A. Function Descriptions and Algorithms of Usart.c:

1. **Function: `displayoption`**

   Description: The `displayoption` function is responsible for displaying a menu with options over UART. The menu allows the user to select from different functionalities such as LED blinking with various delays, invoking PWM to vary LED intensity, and exiting the utility. This function sends a pre-defined set of strings to the UART for displaying the menu options.

   Algorithm:

   - Call `UART2_SendString` to send the header and menu options.
   - Continue sending strings until all menu options are displayed.

2. **Function: `UART2_SendChar (char ch)`**

   Description: The `UART2_SendChar` function transmits a single character over USART2. It loads the character into the USART2 Data Register (DR) and waits until the transmission is complete.

   Algorithm:

   - Load the character into the `USART2->DR` register.
   - Wait until the transmit data register is empty by checking `USART_SR_TXE` flag.

3. **Function: `UART2_SendString (char *string)`**

   Description: The `UART2_SendString` function sends a string of characters over USART2. It repeatedly calls `UART2_SendChar` for each character in the string until the null terminator is reached.

   Algorithm:

   - Iterate through each character in the string.
   - For each character, call `UART2_SendChar`.
   - Continue until the end of the string (`\0`).

4. **Function:** `Usart2_config`

   Description: The `Usart2_config` function configures the USART2 peripheral for communication. It enables the necessary clocks, sets the alternate function modes for PA2 and PA3, configures the speed and alternate functions, and sets the USART2 control registers for operation.

   Algorithm:

   - Enable the clocks for GPIOA and USART2.
   - Set PA2 and PA3 to alternate function mode.
   - Configure high-speed mode for PA2 and PA3.
   - Set the alternate functions for PA2 (USART2 TX) and PA3 (USART2 RX).
   - Clear and configure `USART2->CR1` for USART enable, 8-bit word length, receiver, and transmitter enable.
   - Set the baud rate in `USART2->BRR`.

5. **Function:** `receiverdata`

   Description: The `receiverdata` function receives a single byte of data over USART2. It waits until the receive data register is not empty and then reads the received byte from the data register.

   Algorithm:

   - Wait until data is received by checking `USART_SR_RXNE` flag.
   - Read the received data from `USART2->DR`.
   - Return the received data.

B. Function Descriptions and Algorithms Case1_led_blinking.c:

1. **Function:** `led_blinking_init`

   Description: The `led_blinking_init` function initializes the GPIO pins for the LED and button. It enables the clocks for GPIOA and GPIOC, sets PA1 as an output pin for the LED, and sets PC13 as an input pin for the button.

   Algorithm:

   - Enable the clock for GPIOA and GPIOC using `RCC->AHB1ENR`.
   - Configure PA1 as an output pin:
     - Clear the MODER bits for PA1.
     - Set PA1 to output mode by setting the appropriate bits in the MODER register.
   - Configure PC13 as an input pin by clearing the MODER bits for PC13.

2. **Function:** `led_blinking_start`

Description: The `led_blinking_start` function starts the LED blinking with a variable delay. The delay can be controlled by pressing a button connected to PC13. The delay increases each time the button is pressed until it reaches a maximum value of 5, after which it resets to 1.

Algorithm:

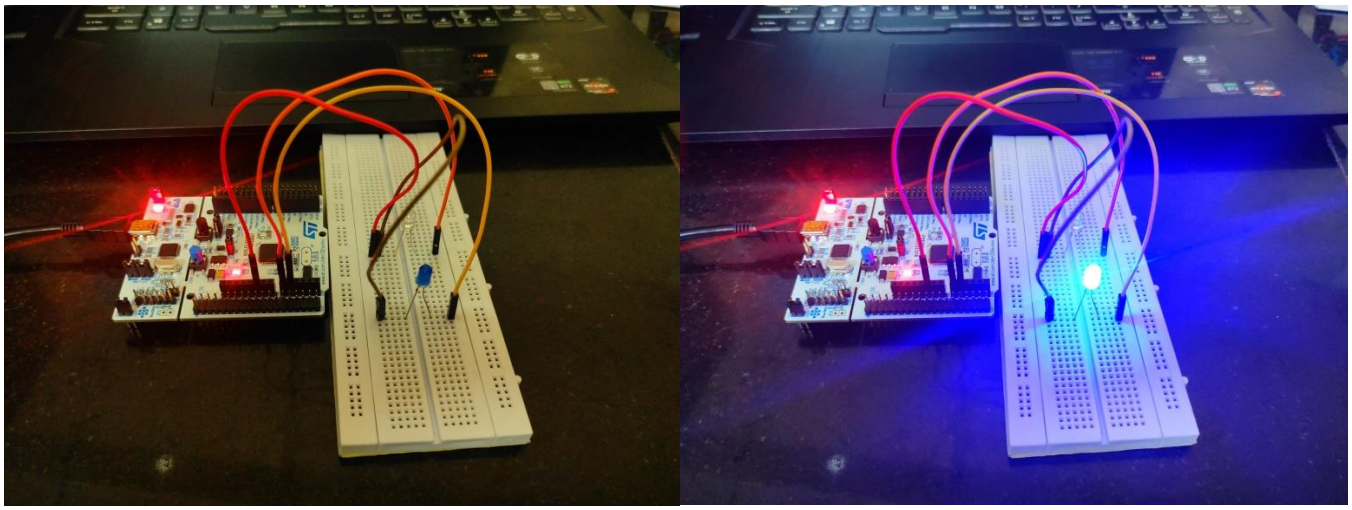
- Enter a loop that continues while `led_blinking_delay` is less than or equal to 5.
- Toggle the LED connected to PA1 using `GPIOA->ODR`.
- Introduce a delay by looping for `1000000 * led_blinking_delay` iterations.
- Check if the button connected to PC13 is pressed:
  - If the button is pressed, increment `led_blinking_delay`.
  - If `led_blinking_delay` exceeds 5, break the loop.
  - Wait until the button is released.
- Reset `led_blinking_delay` to 1 after the loop exits.

3. **Function: `led_blinking_stop`**

Description: The `led_blinking_stop` function stops the LED blinking by turning off the LED connected to PA1.

Algorithm:

- Clear the bit corresponding to PA1 in the `GPIOA->ODR` register to turn off the LED.



C. Function Descriptions and Algorithms Case2_pwm.c:

1. **Function: `PWM_Intensity_init`**

Description: The `PWM_Intensity_init` function initializes the PWM setup for controlling LED intensity. It configures the system clock, initializes the SysTick timer, sets up GPIO pins for PWM output and button input, and configures the TIM2 timer for PWM operation.

Algorithm:

- Configure the system clock using the PLL settings by calling `clockSpeed_PLL`.
- Initialize the SysTick timer by calling `SysTick_Init`.
- Enable the clock for GPIOA and GPIOC using `RCC->AHB1ENR`.
- Configure PA0 for PWM output:
  - Reset the mode for PA0.
  - Set PA0 to alternate function mode.
  - Set the alternate function for PA0.
- Configure PC13 as input for the button by clearing the mode for PC13.
- Enable the clock for TIM2 using `RCC->APB1ENR`.
- Configure TIM2 for PWM:
  - Set the auto-reload register (`TIM2->ARR`) to 100.
  - Set PWM mode 1 on channel 1 by setting bits in `TIM2->CCMR1`.
  - Enable capture/compare on channel 1 using `TIM2->CCER`.
  - Enable the TIM2 counter by setting the appropriate bit in `TIM2->CR1`.

2. **Function: `PWM_Intensity_start`**

Description: The `PWM_Intensity_start` function starts controlling the PWM intensity of the LED. The brightness increases in steps of 10 each time a button connected to PC13 is pressed, up to a maximum value of 100. The function continuously checks the button state and adjusts the PWM duty cycle accordingly.

Algorithm:

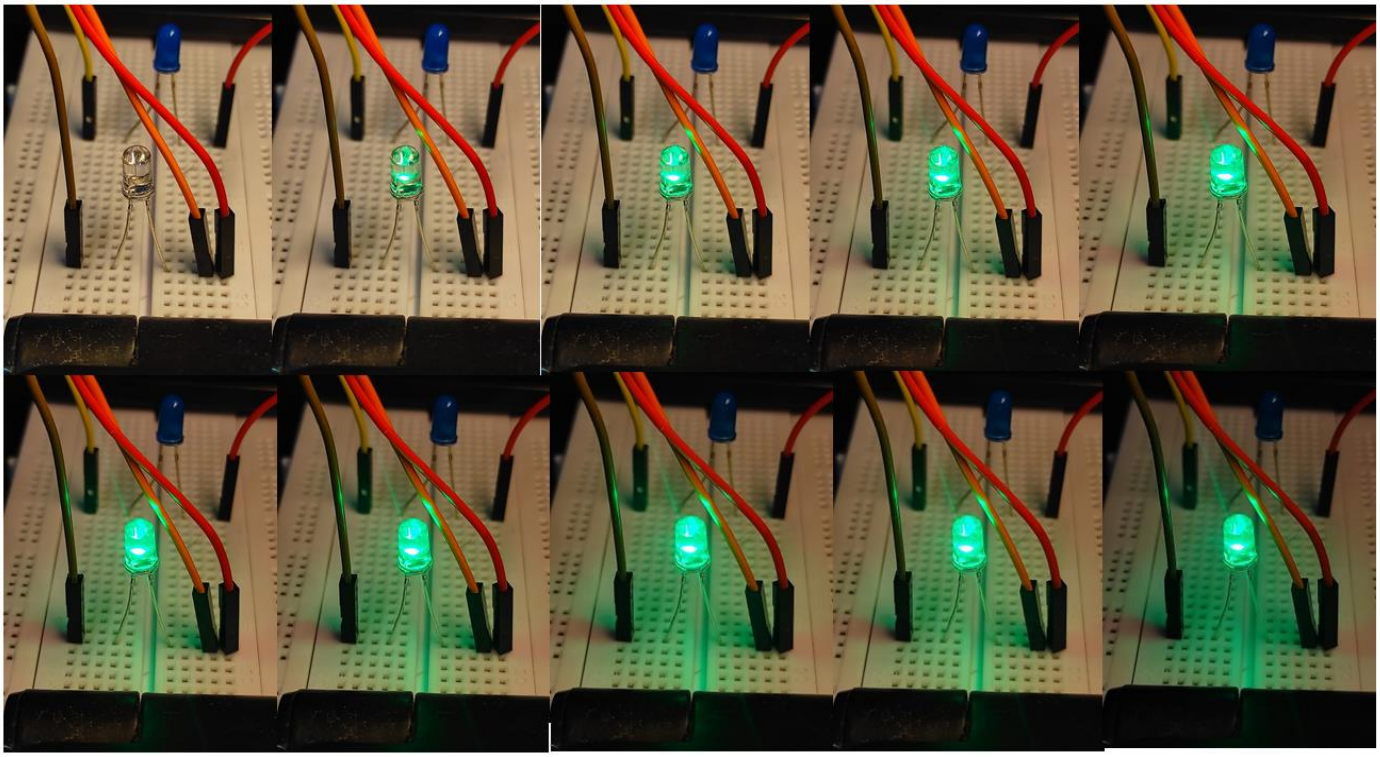
- Loop while `bright_pwm` is less than or equal to 100.
- Check if the button connected to PC13 is pressed:
  - If the button is pressed, set the PWM duty cycle (`TIM2->CCR1`) to `bright_pwm`.
  - Increment `bright_pwm` by 10.
  - If `bright_pwm` exceeds 100, exit the loop.
  - Wait until the button is released.
- Reset `bright_pwm` to 0 after exiting the loop.

3. **Function: `PWM_Intensity_close`**

Description: The `PWM_Intensity_close` function stops the PWM intensity control by setting the PWM duty cycle to 0, effectively turning off the PWM signal.

Algorithm:

- Set the PWM duty cycle (`TIM2->CCR1`) to 0 to turn off the PWM.



D. Function Descriptions and Algorithms Main.c:

1. **Function:** `main`

   Description: The `main` function initializes the USART2 for communication, sets up the LED blinking and PWM intensity control functionalities, and runs an infinite loop to handle user menu options received via UART. Based on the user's choice, it performs the corresponding operations such as starting/stopping LED blinking or adjusting PWM intensity, and provides feedback through UART.

   Algorithm:

   - Configure USART2 by calling `Usart2_config`.
   - Initialize LED blinking functionality by calling `led_blinking_init`.
   - Initialize PWM intensity control by calling `PWM_Intensity_init`.
   - Enter an infinite loop to handle user menu options:
     - Display available options to the user by calling `displayoption`.
     - Receive user choice via UART using `receiverdata`.
     - Echo the user's choice back to them using `UART2_SendString` and `UART2_SendChar`.
     - Use a `switch` statement to handle the user's choice:

        - Case '1': Start and stop LED blinking by calling `led_blinking_start` and `led_blinking_stop`, then send a message indicating the end of case 1.
        - Case '2': Start and stop PWM intensity control by calling `PWM_Intensity_start` and `PWM_Intensity_close`, then send a message indicating the end of case 2.
        - Case '3': Stop LED blinking and PWM intensity control by calling `led_blinking_stop` and `PWM_Intensity_close`, then send a message indicating exit and break the loop.
        - Default: Send an error message indicating an invalid choice.
    - Break the loop if the user chooses option '3'.
- Return 0 to indicate successful execution.

E. Connection details:

Sl No	Peripheral	Port & Pin No	Type	Description	Mode
1	LED1(Green)	PA0	OUTPUT	Shows LED blinking with adjustable delay.	Alternate Function
2	LED2(Blue)	PA1	OUTPUT	Shows varying light intensity with PWM.	Output
3	BUTTON	PC13	INPUT	Control LED operations, including PWM for varying light intensity and LED blinking.	Input
4	USART2_Tx	PA2	DATA TRANSMISSION	Used in the transmission of data	Alternate Function
5	USART2_Rx	PA3	DATA RECEIVING	Used in the recieving of data	Alternate Function

F. Conclusion

The developed menu-driven utility on the STM32F446xx microcontroller provides a robust and user-friendly interface for controlling LED blinking and PWM-based LED intensity. By leveraging USART communication, the system allows users to interact with the microcontroller efficiently, selecting different operations and receiving real-time feedback. This project demonstrates the effective integration of various hardware and software components, highlighting the versatility and capabilities of the STM32F446xx microcontroller for embedded applications.

G. Future Scope

1. **Enhanced User Interface:**
    - Implement a more advanced user interface with an LCD display or a graphical screen to provide a more intuitive and visually appealing way to interact with the system.
2. **Additional Functionality:**
    - Expand the menu options to include additional functionalities, such as temperature sensing, motor control, or communication with other peripherals (e.g., SPI, I2C).
3. **Remote Control:**
    - Integrate wireless communication modules (e.g., Bluetooth, Wi-Fi) to enable remote control and monitoring of the system from a smartphone or a computer.
4. **Data Logging:**
    - Implement data logging capabilities to record user interactions and system responses, which can be useful for debugging and performance analysis.
5. **Energy Efficiency:**
    - Optimize the power consumption of the system by implementing low-power modes and more efficient power management strategies, making it suitable for battery-operated applications.
6. **Real-time Operating System (RTOS):**
    - Integrate an RTOS to manage the different tasks more efficiently, providing better timing accuracy and multitasking capabilities.
7. **Advanced PWM Control:**
    - Implement more sophisticated PWM control algorithms, such as PID control, to provide smoother and more precise control over LED intensity and other PWM-based applications.
8. **Firmware Over-the-Air (FOTA) Updates:**
    - Enable FOTA updates to allow for remote firmware updates, improving the system's maintainability and scalability.