AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Faculty of Engineering

Lab Report

Experiment # 05

Experiment Title: Familiarization of assembly language program and Interrupts in a microcontroller.

Date of Perform:	10 April 2025	**Date of Submission:**	27 April 2025
Course Title:	Microprocessor and Embedded Systems Lab		
Course Code:	EE4103	**Section:**	P
Semester:	Spring 2024-25	**Degree Program:**	BSc in CSE
Course Teacher:	**Prof. Dr. Engr. Muhibul Haque Bhuyan**		

Declaration and Statement of Authorship:

1. I/we hold a copy of this Assignment/Case Study, which can be produced if the original is lost/damaged.
2. This Assignment/Case Study is my/our original work; no part has been copied from any other student's work or any other source except where due acknowledgment is made.
3. No part of this Assignment/Case Study has been written for me/us by any other person except where such collaboration has been authorized. by the concerned teacher and is acknowledged in the assignment.
4. I/we have not previously submitted or am submitting this work for any other course/unit.
5. This work may be reproduced, communicated, compared, and archived to detect plagiarism.
6. I/we permit a copy of my/our marked work to be retained by the Faculty Member for review by any internal/external examiners.
7. I/we understand that Plagiarism is the presentation of the work, idea, or creation of another person as though it is your own. It is a form of cheating and is a very serious academic offense that may lead to expulsion from the University. Plagiarized material can be drawn from, and presented in, written, graphic, and visual forms, including electronic data, and oral presentations. Plagiarism occurs when the origin of the source is not appropriately cited.
8. I/we also understand that enabling plagiarism is the act of assisting or allowing another person to plagiarize or copy my/our work.

** Student(s) must complete all details except the faculty use part.*
** Please submit all assignments to your course teacher or the office of the concerned teacher.

Group # 01

Sl No	Name	ID	PROGRAM	SIGNATURE
1	Md.Saikat Hossain	23-51242-1	BSc in CSE	
2	Md.Mosharof Hossain Khan	23-51259-1	BSc in CSE	
3	Rimal Banik	23-51260-1	BSc in CSE	
4	Md.Rahidul Islam	23-51269-1	BSc in CSE	
5	Rahat Ahmed	22-44911-2	BSc in CSE	

Faculty use only

FACULTY COMMENTS	**Marks Obtained**	
	Total Marks	

Contents

Objectives	3
Apparatus	3
Circuit Diagram	3-4
Code Explanation	4-9
Experimental Output Results	9-14
Simulation Output Results	14-19
Answer to Question	19-21
Discussion	22
Conclusion	23
References	23

Objectives:

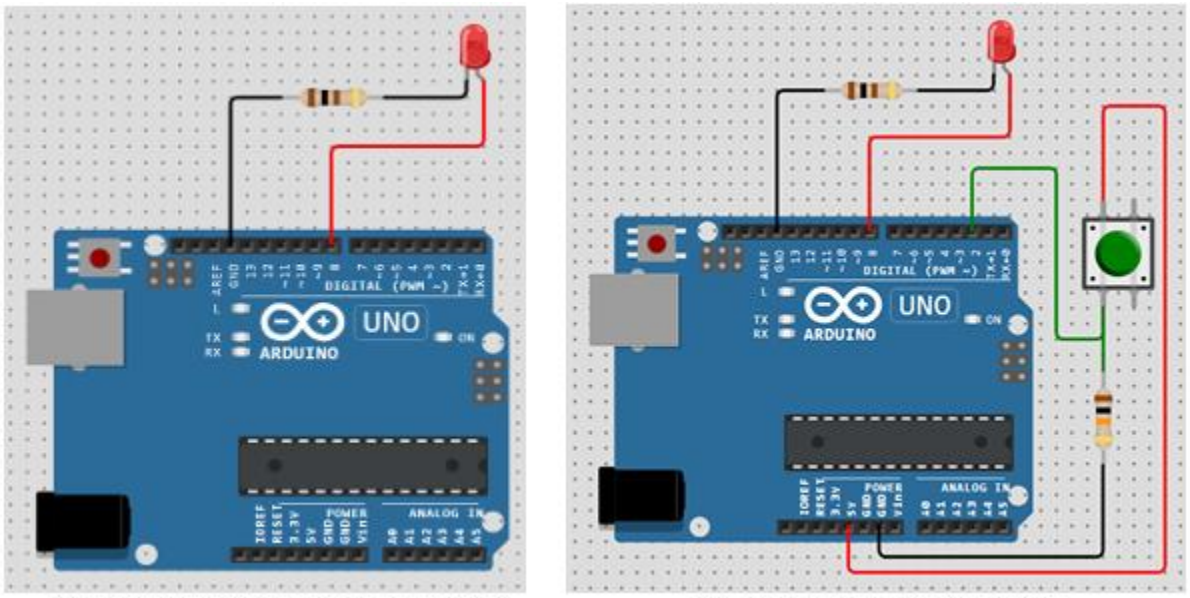
The objectives of this experiment are to

1.Study the assembly language program of an Arduino.
2.Write assembly language programming code for an Arduino.
3.Build a circuit to turn on and off an LED on an Arduino Microcontroller Board connected to an I/O port of the microcontroller.
4.Study of the external interrupts of an Arduino using its digital I/O port.
5.Build a circuit to turn on and off an LED on an Arduino Microcontroller Board connected to an I/O port of the microcontroller due to the external interrupt.

Equipment List:

1) Arduino IDE 2.3.5
2) Arduino Microcontroller board
3) PC having an Intel processor
4) LED lights
5) One 100Ω resistor
6) One push switch
7) Jumper wires

Circuit Diagram:



(a) LED state change without any switch; (b) LED state change with switch:

Figure-01:Experimental setup using an Arduino Microcontroller

Code Explanation:

1. **Code of an LED light blink:**

The .ino file:

```
//------------------------
// C Code for Blinking LED
//------------------------
extern “C”
{
void start();
void led(byte);
}
//-------------------------------------------------
void setup()
{
start();
}
//-------------------------------------------------
void loop()
{
led(1);
led(0);
}
```

The .S file:

```
;---------------
; Assembly Code
;---------------
#define __SFR_OFFSET 0x00
#include “avr/io.h”
;------------------------
.global start
.global led
;------------------------
start:
SBI DDRB, 5; set PB5 (D13) as o/p
RET; return to setup() function
;-------------------------------------------------------------------------
led:
CPI R24, 0x00; value in R24 passed by caller compared with 0
BREQ ledOFF; jump (branch) if equal to subroutine ledOFF
SBI PORTB, 5; set D13 to HIGH, i.e., the LED will turn ON
RCALL myDelay; Calling a delay function to determine the ON duration of LED
RET; return to loop() function
;-------------------------------------------------------------------------
ledOFF:
CBI PORTB, 5; set D13 to LOW, i.e., the LED will turn OFF
RCALL myDelay; Calling a delay function to determine
```

```
. equ delayVal, 10000; initial count value for the inner loop (0010011100010000)
;-------------------------------------------------------------------------
myDelay:
LDI R20, 100; initial count value for the outer loop
outerLoop:
LDI R30, lo8(delayVal); low byte of delayVal in R30 (00010000)
LDI R31, hi8(delayVal); high byte of delayVal in R31 (00100111)
innerLoop:
SBIW R30, 1; subtract 1 from 16-bit value in R31, R30
BRNE innerLoop; jump if countVal not equal to 0
;--------------
SUBI R20, 1; subtract 1 from R20
BRNE outerLoop; jump if R20 is not equal to 0
RET
```

Explanation: The code integrates C++ and Assembly to blink an LED connected to pin D13 of an Arduino board. In the .ino file, the extern "C" block is used to declare two functions (start() and led(byte)) that are defined in the accompanying assembly file. This allows the Arduino C++ code to interact with low-level assembly instructions. In the setup() function, the start() function is called once. This function is written in assembly and is responsible for configuring pin D13 as an output. This setup is essential for controlling the LED. The loop() function runs continuously. Inside it, led(1) is called to turn the LED on, and led(0) is called to turn the LED off. These function calls go to the assembly-defined led function, which handles the actual control of the hardware pin and includes a delay to keep the LED on or off for a visible duration.In the .S file, the start function sets bit 5 of the DDRB register using the SBI instruction, which configures pin D13 (PB5) as an output. After this configuration, the function returns control back to the C++ setup().The led function compares the value passed in register R24. If the value is 0, it jumps to the ledOFF routine, which clears bit 5 of PORTB, turning the LED off. If the value is not 0, it sets bit 5 of PORTB, turning the LED on. In both cases, a custom delay function named myDelay is called to keep the LED in its current state for a short time.The delay is implemented entirely in assembly. A constant delayVal (10000) is defined to determine how long the delay should last. The myDelay function uses a nested loop approach: the outer loop runs 100 times, and the inner loop counts down from delayVal. The combination of these loops creates a visible delay that controls how long the LED remains on or off.

2. Push button LED control

The btnLED.ino file:

```
//-----------------------------------
// C Code: RGB LED ON/OFF via Buttons
//-----------------------------------
extern "C"
{
void start();
void btnLED();
}
//----------------------
void setup()
{
start();
}
//----------------------
void loop()
{
btnLED();
}
```

The btnLED.S file:

```
;-----------------------------------------
; Assembly Code: RGB LED ON/OFF via Buttons
;-----------------------------------------
#define __SFR_OFFSET 0x00
#include "avr/io.h"
;-----------------------
.global start
.global btnLED
;=================================================================
SBI DDRB, 4; set PB4 (pin D12 as o/p - red LED)
SBI DDRB, 3; set PB3 (pin D11 as o/p - green LED)
SBI DDRB, 2; set PB2 (pin D10 as o/p - blue LED)
CBI DDRD, 2; clear PD2 (pin D02 as i/p - red button)
CBI DDRD, 3; clear PD3 (pin D03 as i/p - green button)
CBI DDRD, 4; clear PD4 (pin D04 as i/p - blue button)
RET
btnLED:
L2: SBIS PIND, 4 ; Skips below statement if the push button of D04 is not pressed
RJMP L1
SBI PORTB, 2 ; Turn ON LED, PB2(D10), if SW of D04 is not pressed
CBI PORTB, 3 ; Turn OFF LED, PB3(D11), if SW of D04 is not pressed
SBIC PIND, 4 ; Skips below statement if the push button of D04 is pressed
RJMP L2
L1: CBI PORTB, 2 ; Turn OFF LED, PB2(D10), if SW of D04 is pressed
SBI PORTB, 3 ; Turn OFF LED, PB3(D11), if SW of D04 is pressed RET
```

Explanation: In this code, a program was developed using both C and Assembly language to control an RGB LED through three push buttons connected to an Arduino microcontroller. The C code (btnLED.ino) serves as the main structure, where the setup() function calls the start() routine to initialize hardware configurations, and the loop() continuously calls the btnLED() function to monitor button inputs and control the LEDs accordingly. In the Assembly code (btnLED.S), the start routine configures pins PB2, PB3, and PB4 (Arduino pins D10, D11, D12) as outputs for the RGB LEDs, and PD2, PD3, PD4 (D2, D3, D4) as inputs for the push buttons. The btnLED routine constantly checks the state of the buttons. Specifically, it monitors pin D4: if the button connected to D4 is not pressed, it turns ON the blue LED (connected to PB2/D10) and turns OFF the green LED (PB3/D11). If the button is pressed, it switches the state — turning OFF the blue LED and ensuring the green LED is activated. The use of SBIS and SBIC instructions allows the program to perform non-blocking checks on input pins without halting the microcontroller. This approach demonstrates efficient input-output handling through Assembly, providing hands-on experience in direct hardware manipulation, interrupt-free polling, and responsive LED control based on real-time button presses.

3.Push button LED control

The pbintLED.ino file:

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;
void setup() {
pinMode(ledPin, OUTPUT);
pinMode(interruptPin, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}
void loop() {
digitalWrite(ledPin, state);
}
void blink() {
state = !state;
}
```

Explanation: In this code, a simple interrupt-based LED control system was implemented using Arduino. The program defines two pins: ledPin (pin 13) connected to an LED and interruptPin (pin 2) connected to a push button. In the setup() function, the LED pin is configured as an output, while the button pin is set as an input with an internal pull-up resistor enabled. The attachInterrupt() function is used to link an external interrupt to the button pin; it triggers the blink() function every time a CHANGE (either press or release) is detected on the button. Inside the blink() interrupt service routine (ISR), the state variable toggles between HIGH and LOW. The loop() function continuously writes this state value to the LED pin, thereby turning the LED ON or OFF based on the last button event. This approach allows the microcontroller to respond immediately to button presses without constantly polling the button in the main loop, making the system more efficient and responsive. The use of interrupts demonstrates a real-world embedded system technique where real-time event detection is critical, such as emergency stop buttons, door sensors, or alarm triggers.OFF.

4. Blinking an LED using a Timer1 ISR :

The blinkLED.ino file:

```
bool LED_State = 'True';
void setup() {
pinMode(13, OUTPUT);
cli(); // stop interrupts till we make the settings
TCCR1A = 0; // Reset the entire A and B registers of Timer1 to make sure that
TCCR1B = 0; // we start with everything disabled.
TCCR1B = 0b00000100; // Set CS12 bit of TCCR1B to 1 to get a prescalar value of 256.
TIMSK1 = 0b00000010; // Set OCIE1A bit to 1 to enable compare match mode of A reg.
OCR1A = 31250; // We set the required timer count value in the compare register, A
sei(); // Enable back the interrupts
}
void loop() {
// put your main code here, to run repeatedly.
}
// With the settings above, this ISR will trigger each 500 ms.
ISR(TIMER1_COMPA_vect) {
TCNT1 = 0; // First, set the timer back to 0 so that it resets for the next interrupt
LED_State = !LED_State; // Invert the LED State
digitalWrite(13, LED_State); // Write this new state to the LED connected to pin D5
}
```

Explanation: In this code, an LED was controlled using the Timer1 Compare Match Interrupt of the Arduino microcontroller. At the start, a boolean variable LED_State was initialized to track the ON/OFF state of the LED. Inside the setup() function, pin 13 was configured as an output pin, and interrupts were temporarily disabled using cli() to safely configure the timer settings. Both TCCR1A and TCCR1B registers were cleared to reset Timer1. The prescaler was then set to 256 by configuring TCCR1B, effectively slowing down the timer clock. The Compare Match A interrupt was enabled by setting the OCIE1A bit in TIMSK1, and a comparison value of 31250 was loaded into the OCR1A register, which causes the interrupt to trigger approximately every 500 milliseconds. After setting up, interrupts were re-enabled using sei(). In the loop() function, no code was needed because the timer interrupt service routine (ISR) handled the blinking. The ISR TIMER1_COMPA_vect toggled the LED_State variable and updated the LED output accordingly. This setup demonstrated a highly efficient non-blocking way to blink an LED with precise timing, leaving the CPU free for other tasks, a method commonly used in real-time embedded systems where accurate timing without blocking the main execution flow is critical.

5. Controlling an LED using a Pin Change Interrupt ISR

The btnLEDpci.ino file:

```
#include "PinChangeInterrupt.h"
#define BTN_PIN 8
#define LED_PIN 7
void BTN_ISR(void)
{
digitalWrite(LED_PIN, !digitalRead(LED_PIN));
```

```
}
void setup() {
pinMode(LED_PIN, OUTPUT);
pinMode(BTN_PIN, INPUT);
attachPCINT(digitalPinToPCINT(BTN_PIN), BTN_ISR, RISING);
}
void loop() {
// Do Nothing
}
```

Explanation: In this code, an LED was controlled using a Pin Change Interrupt (PCI) on the Arduino. The code begins by including the PinChangeInterrupt.h library, which allows handling interrupts on almost any digital pin, not just pins 2 and 3. Two pins were defined: BTN_PIN for the push button (connected to pin 8) and LED_PIN for the LED (connected to pin 7). In the setup() function, the LED pin was set as an output and the button pin as an input. The attachPCINT() function was used to attach the BTN_ISR interrupt service routine (ISR) to the button pin, triggering the ISR whenever a RISING edge (when the button is pressed) was detected. The BTN_ISR function simply toggled the current state of the LED by reading its state and writing the opposite. This technique allowed the LED to turn ON and OFF with each button press, without needing to constantly check the button in the loop(). The loop() function was left empty, demonstrating how interrupts can be used to react to external events efficiently without occupying the main program execution. This approach is ideal for real-time applications where immediate response to user inputs is crucial without wasting processor time in polling.

Experimental Implementation and Output Results:

1. Blink an LED:

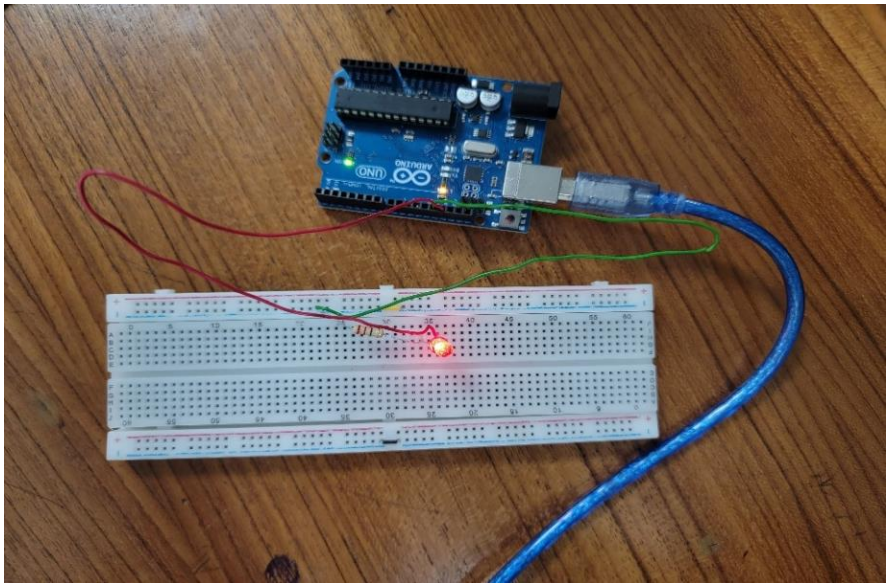


Figure 02: Blink an LED (LED on)

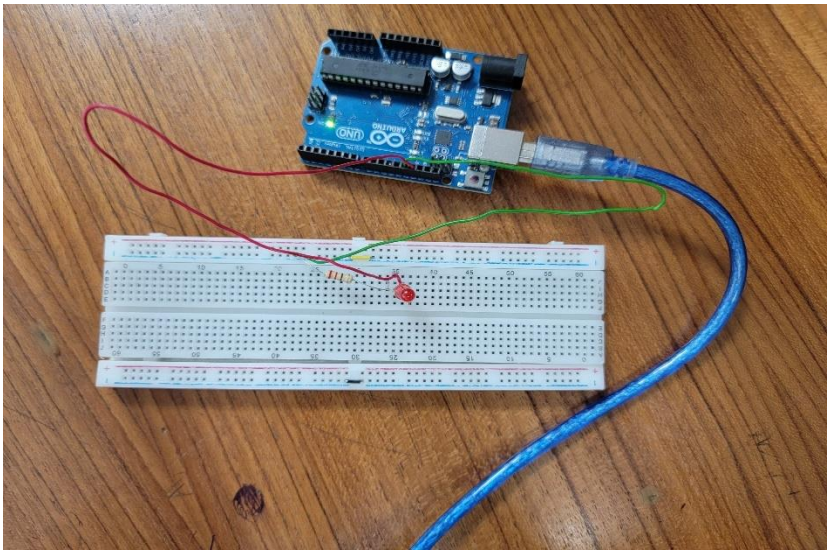


Figure 03: Blink an LED (LED off)

Explanation: The hardware setup involved an Arduino board with an LED connected to pin D13. The assembly code was used to set pin D13 as an output by modifying the DDRB register, and then the LED was controlled by toggling the PB5 bit in the PORTB register. The C code in the .ino file called the start() function to initialize the pin and the loop() function to repeatedly turn the LED ON and OFF. The LED blinked at regular intervals, as the led(1) function turned the LED ON and led(0) turned it OFF, with each state change followed by a delay implemented in Assembly using nested loops. This delay allowed for visible ON/OFF periods. The LED blinked continuously as the code executed in a loop, demonstrating successful pin initialization, control of output devices, and manual delay creation in Assembly. The setup showed how combining Assembly and C code could create an efficient and functional microcontroller program.

2.Push button LED control:

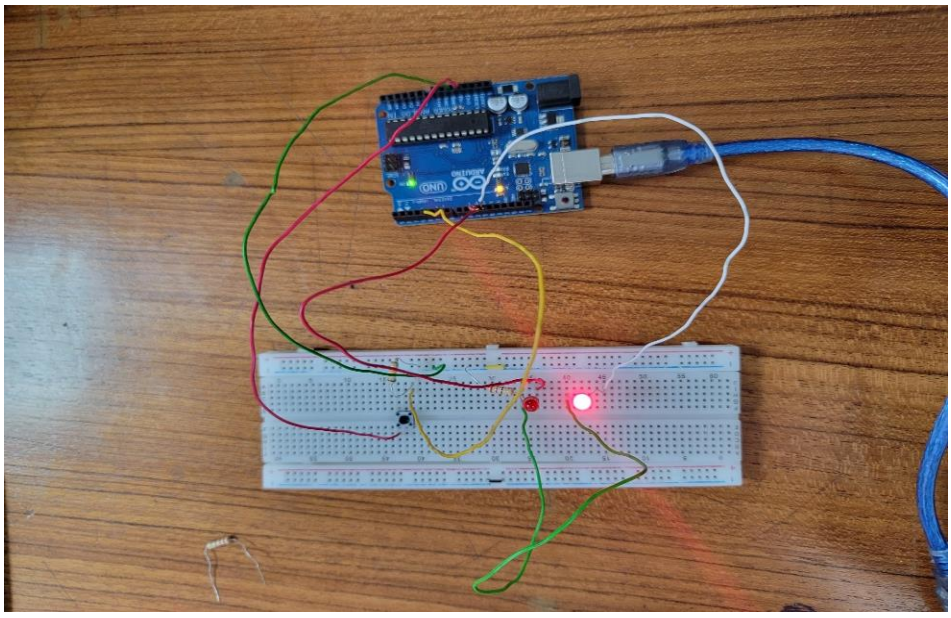


Figure 04: Push button LED control (Button not pressed first LED ON)

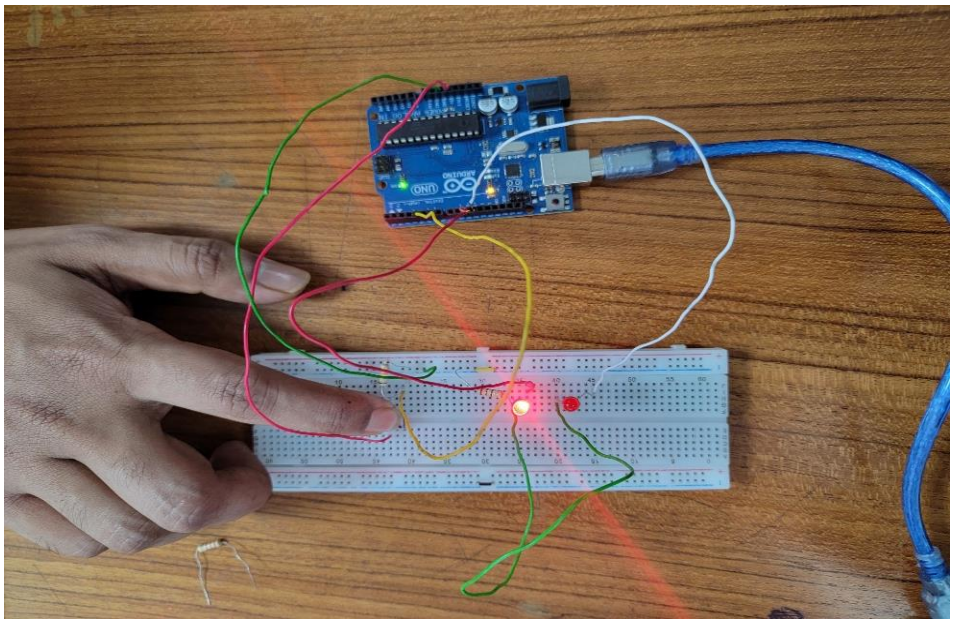


Figure 05: Push button LED control (Button pressed second LED ON)

Explanation: When the hardware setup runs, the microcontroller configures pins for two LEDs as outputs and three buttons as inputs. Initially, the LEDs are off. When a button is pressed, it triggers a corresponding action through the code. The assembly code checks the button states using the SBIS (skip if set) and SBIC (skip if cleared) instructions, and when a button press is detected, it turns on the corresponding LED by setting the corresponding pin high and turning off the others by setting those pins low. The system ensures that only one LED is on at a time by controlling the LEDs in response to the button states. When the button is released, the corresponding LED turns off. The behavior continues in a loop, with the LEDs switching on and off based on button presses.

3.Push button LED control:

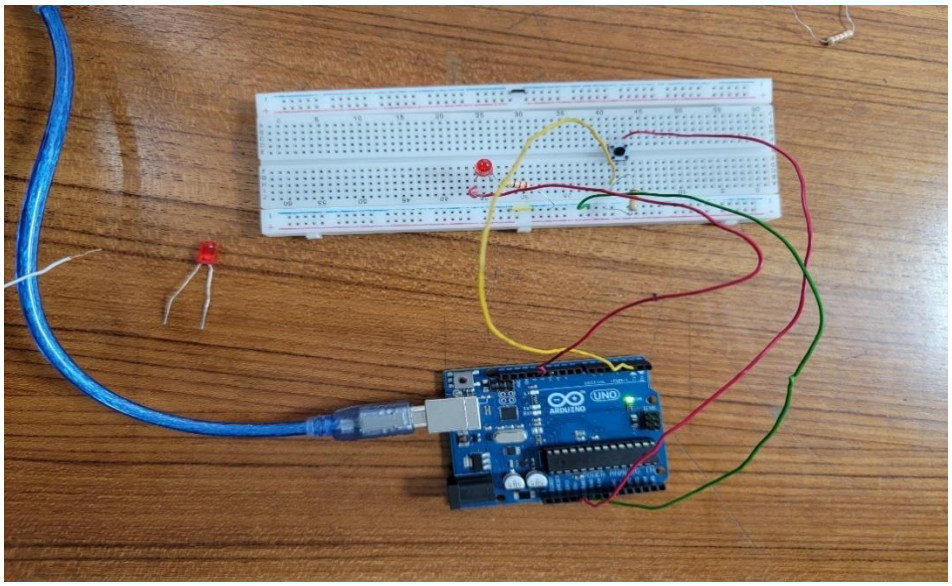


Figure 06: Push button LED control (Button not pressed LED OFF)

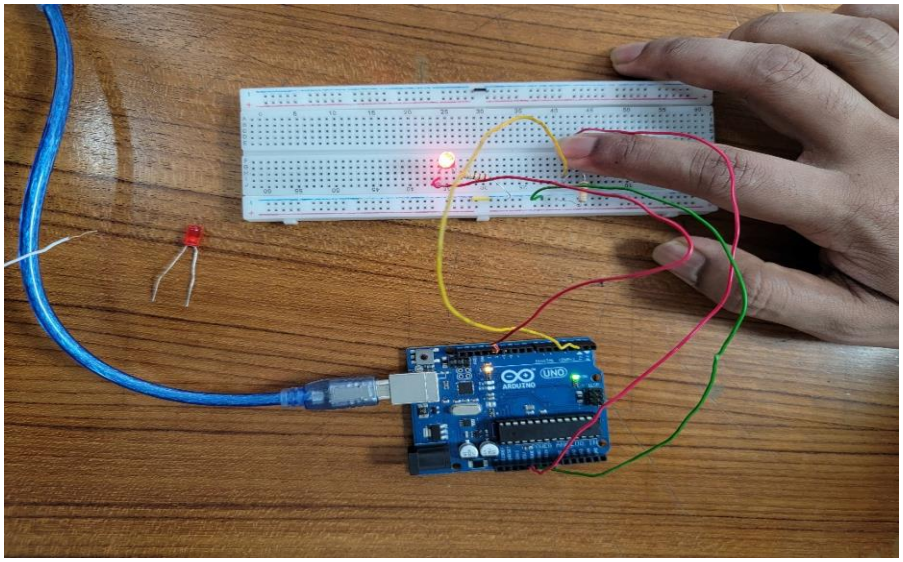


Figure 07: Push button LED control (Button pressed LED ON)

Explanation: The code creates a system where an LED (connected to pin 13) toggles its state each time a button (connected to pin 2) is pressed. When the button's state changes, an interrupt is triggered, calling the blink() function. This function toggles the state of the LED (on or off). The loop() function continuously updates the LED based on the current state, ensuring that the LED responds to button presses by turning on or off.

4.Blinking an LED using a Timer1 ISR:

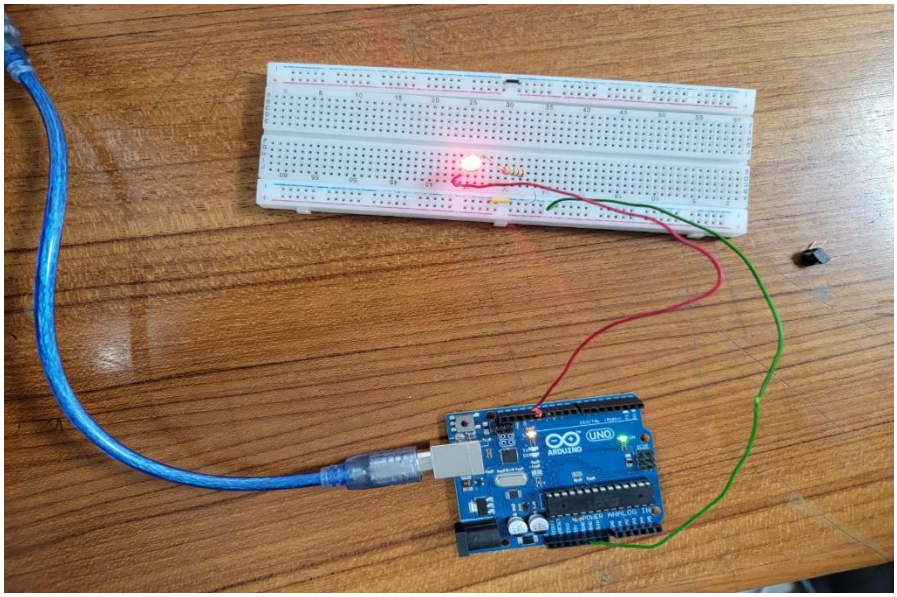


Figure 8**:** Blinking an LED using a Timer1 ISR (LED ON)

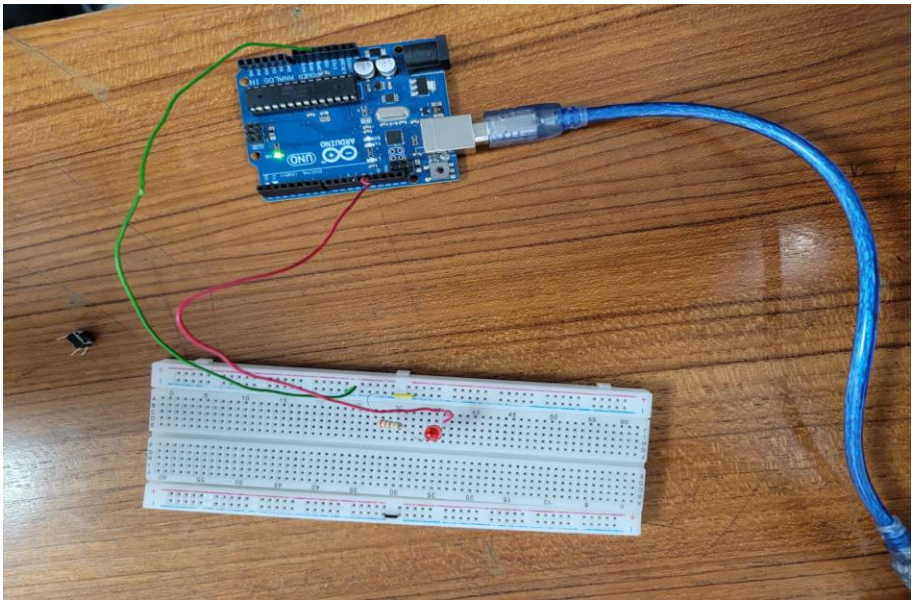


Figure 9**:** Blinking an LED using a Timer1 ISR(LED OFF)

Explanation: When the hardware setup runs, the system initializes by configuring pin 13 as an output for controlling the LED. The interrupt system is initially disabled to ensure no interruptions during the setup. Timer1 is then configured, with a prescaler set to 256 and the compare match register (OCR1A) set to 31250, which triggers an interrupt every 500 milliseconds. After the setup, interrupts are enabled, and the timer begins counting.Every 500 milliseconds, Timer1 triggers the interrupt service routine (ISR), which resets the timer and toggles the LED state. The LED turns on or off depending on the toggled state stored in the LED_State variable. This toggle is handled by the ISR, which calls digitalWrite(13, LED_State) to update the LED's status on pin 13.The process repeats continuously, causing the LED to blink every 500 milliseconds, switching between on and off. This cycle continues as long as the microcontroller is running, creating a blinking effect without blocking other operations.

5.Controlling an LED using a Pin Change Interrupt ISR

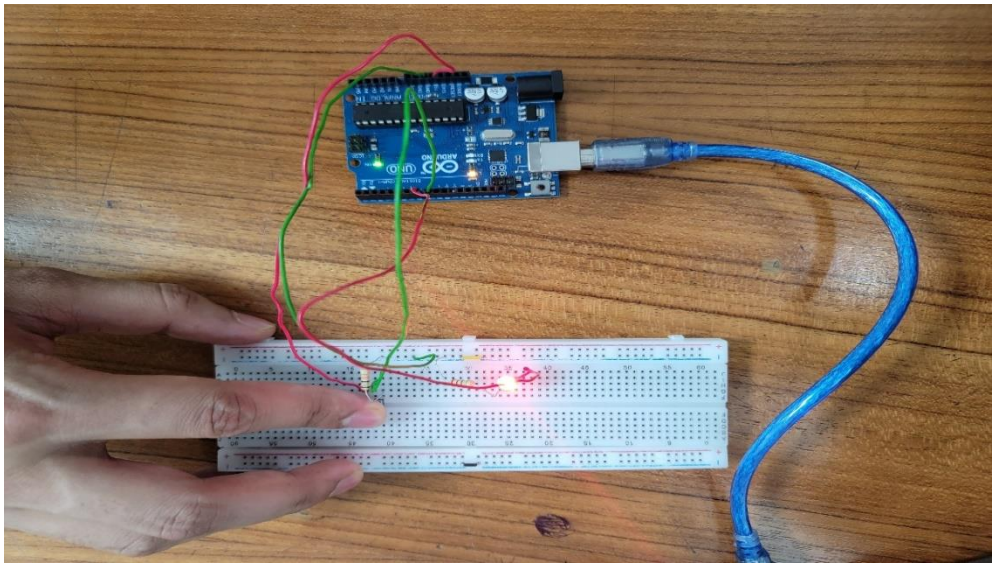


Figure 10: Controlling an LED using a Pin Change Interrupt ISR(button not pressed LED ON)

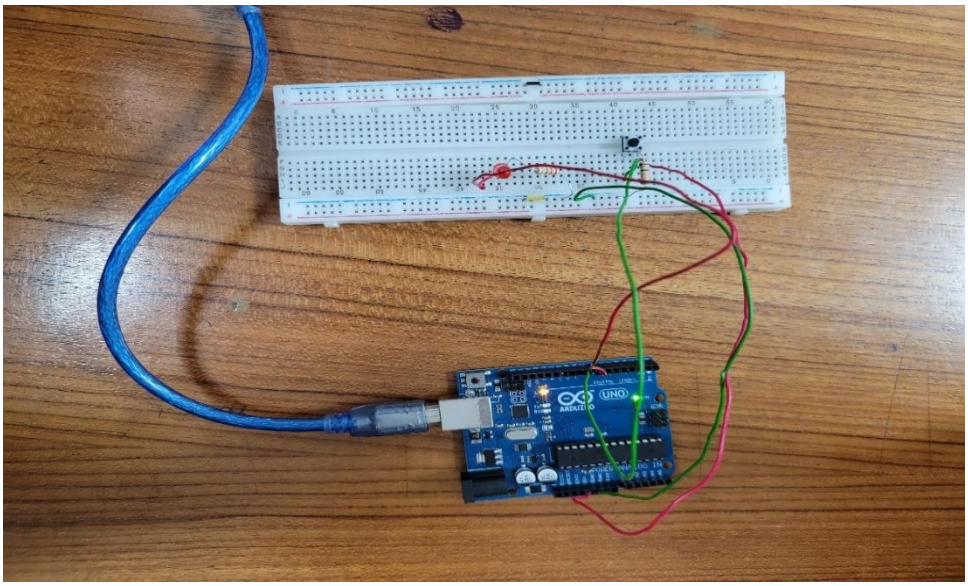


Figure 11: Controlling an LED using a Pin Change Interrupt ISR(button pressed LED OFF)

Explanation: When the hardware setup runs for this code, the microcontroller initializes by configuring pin 7 (LED_PIN) as an output pin and pin 8 (BTN_PIN) as an input pin for the button. The PinChangeInterrupt.h library is used to set up an interrupt that listens for a button press on pin 8.In the setup() function, the pin modes are set for the LED and button, and a pin change interrupt is attached to pin 8 (BTN_PIN). The interrupt is triggered on a rising edge (when the button is pressed, assuming it's connected to ground), which activates the BTN_ISR() function.When the button is pressed, the interrupt service routine (BTN_ISR()) is triggered. In this ISR, the LED state is toggled using digitalWrite(LED_PIN, !digitalRead(LED_PIN)). If the LED is on, it turns off, and if it is off, it turns on. The loop() function doesn't contain any code, as the main action is handled by the interrupt, making the system responsive to button presses without having to continuously poll for the button state.

Simulation Output Results:

1.Blink an LED

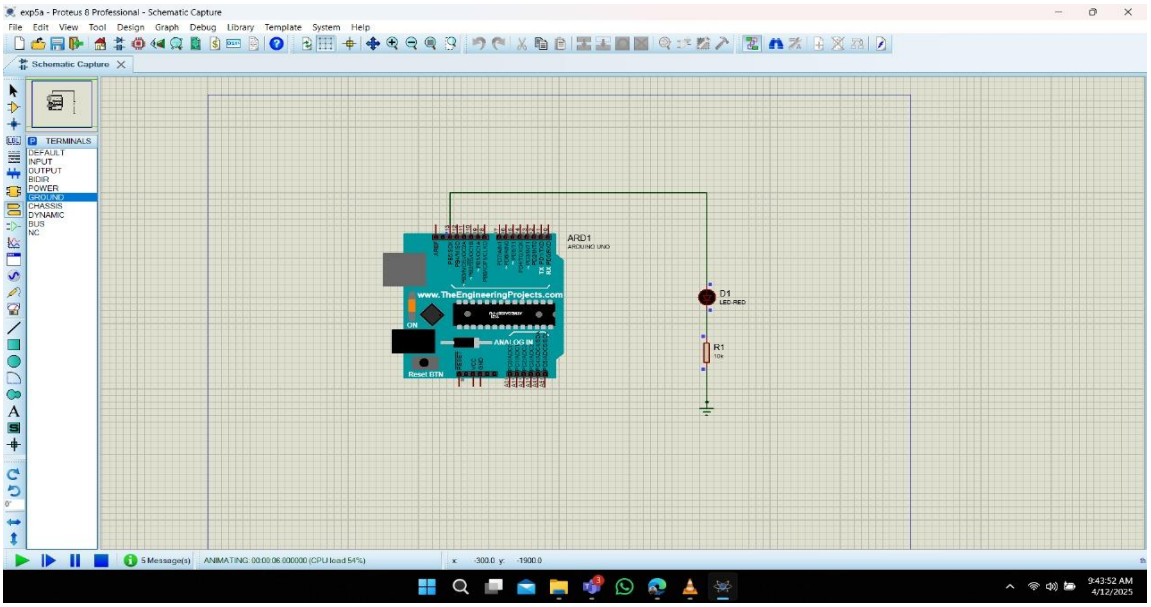


Figure 12: Blink an LED (LED OFF)

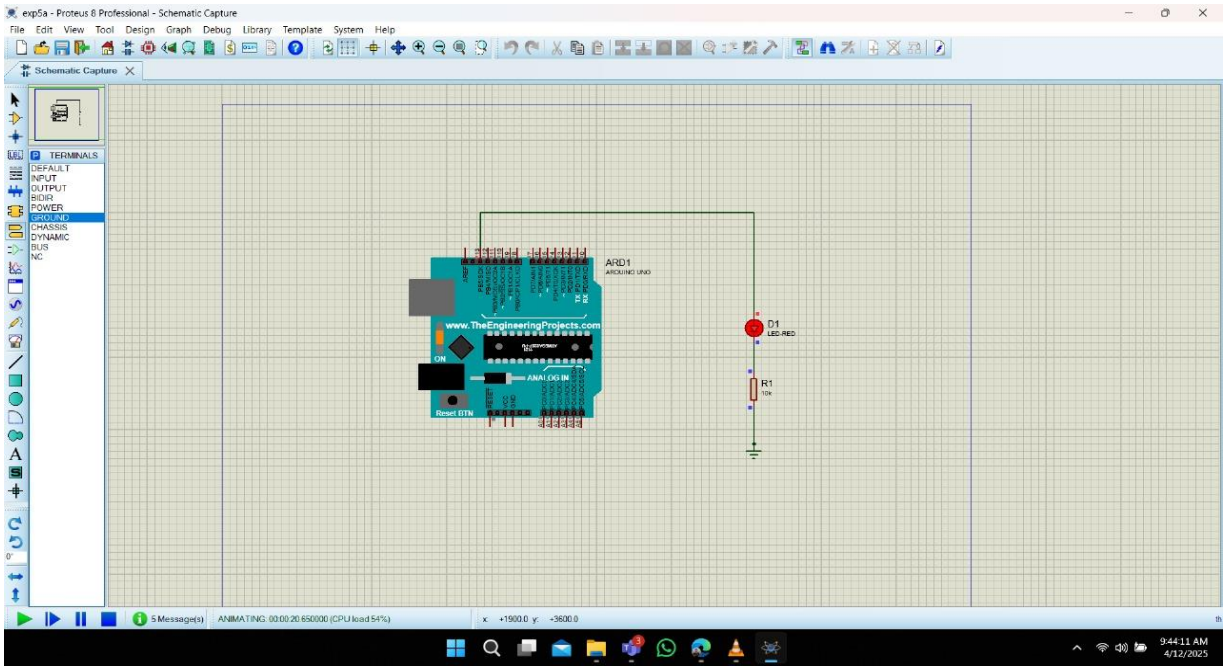


Figure 13: Blink an LED (LED ON)

Explanation: In this simulation, the Arduino Uno board, 100Ω resistor, and LED were set up according to the hardware configuration used in the lab. The program was written and verified in the Arduino IDE 2.3.5, which generated a HEX file. This file was then loaded into the Proteus simulation to simulate the behavior of the circuit. After running the simulation, the results were observed and compared with the actual hardware results to check for consistency and verify the performance of the system.

2.Push button LED control:

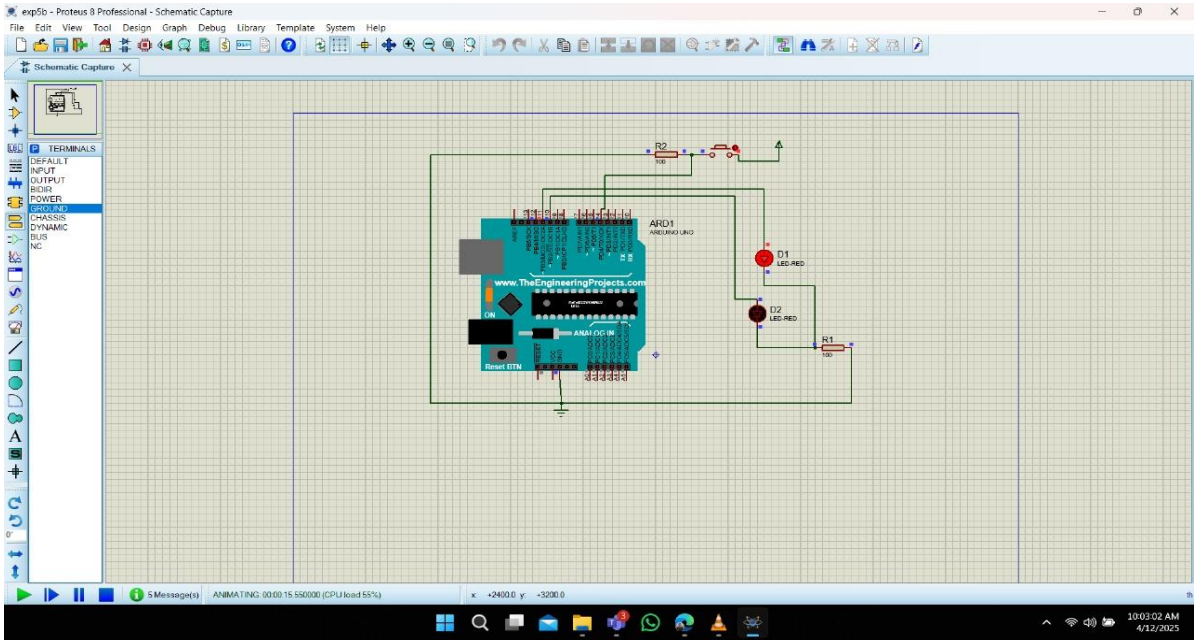


Figure 14: Push button LED control (Button not pressed first LED ON)

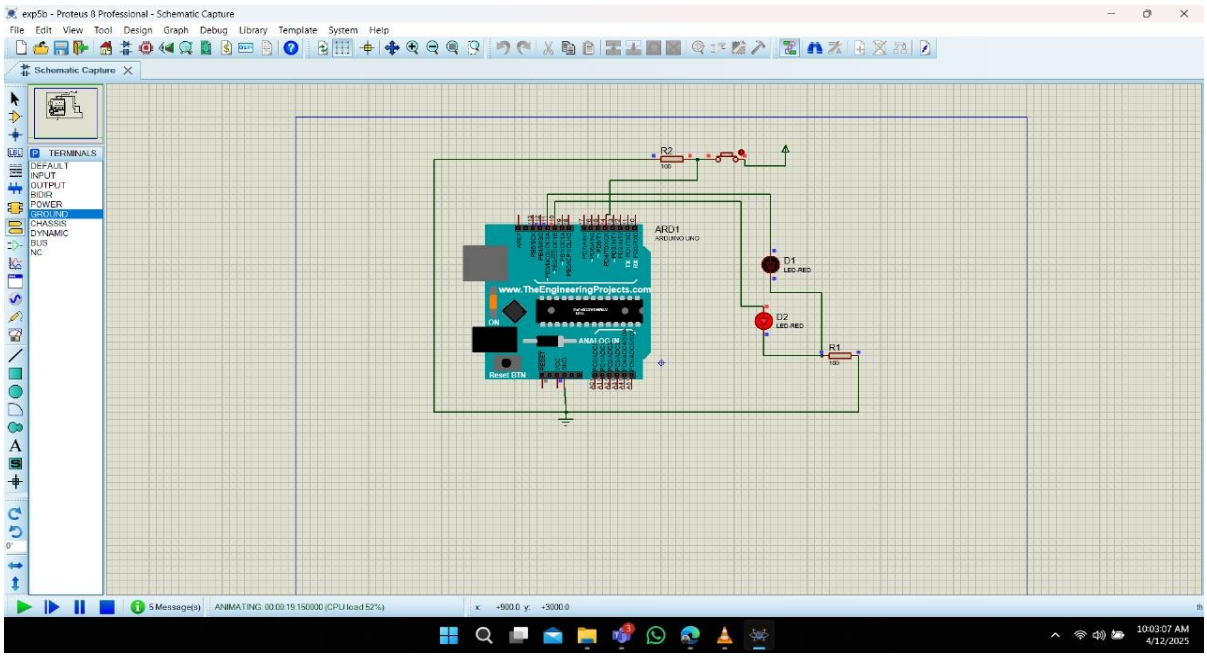


Figure 15: Push button LED control (Button pressed second LED ON)

Explanation: In this simulation, the Arduino Uno, two LEDs, and push buttons were connected as per the hardware setup. The program for controlling two LEDs with push buttons was written and verified in Arduino IDE 2.3.5 to create a HEX file, which was loaded into Proteus. After running the simulation, it was observed that pressing the push buttons correctly turned the LEDs ON and OFF, matching the actual hardware performance and verifying proper system functionality.

3.Push button LED control:

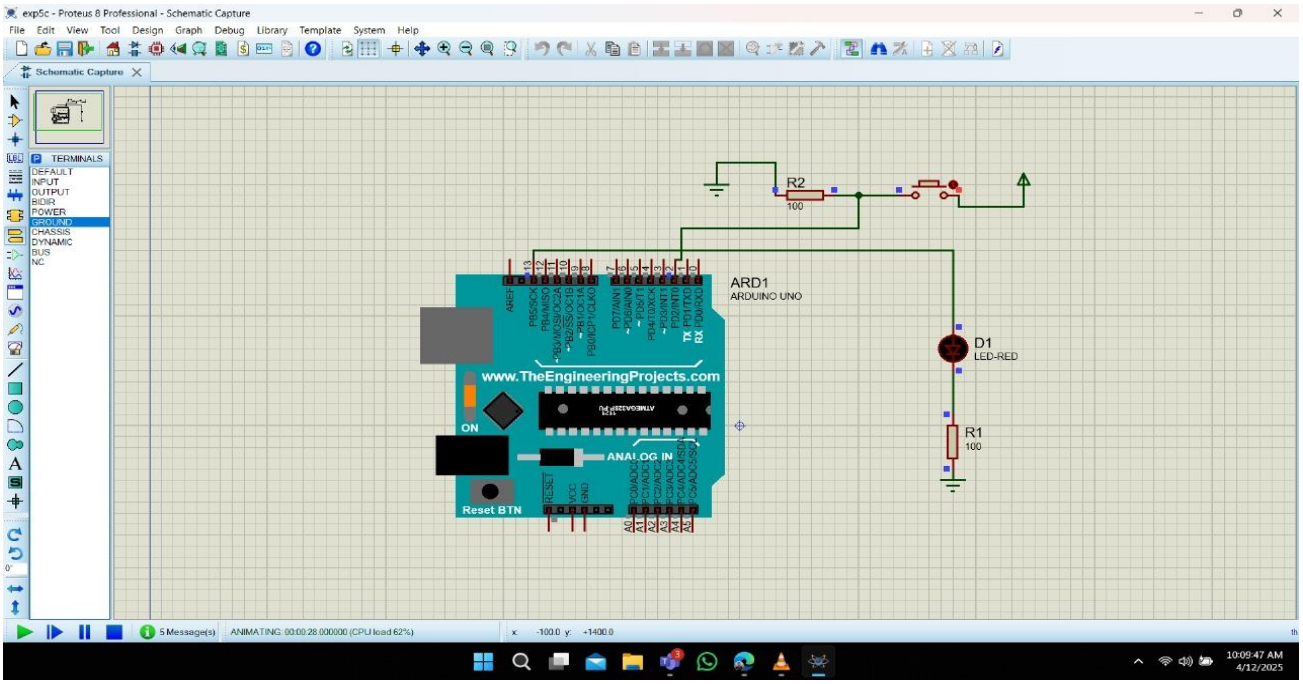


Figure 16: Push button LED control (Button not pressed LED OFF)

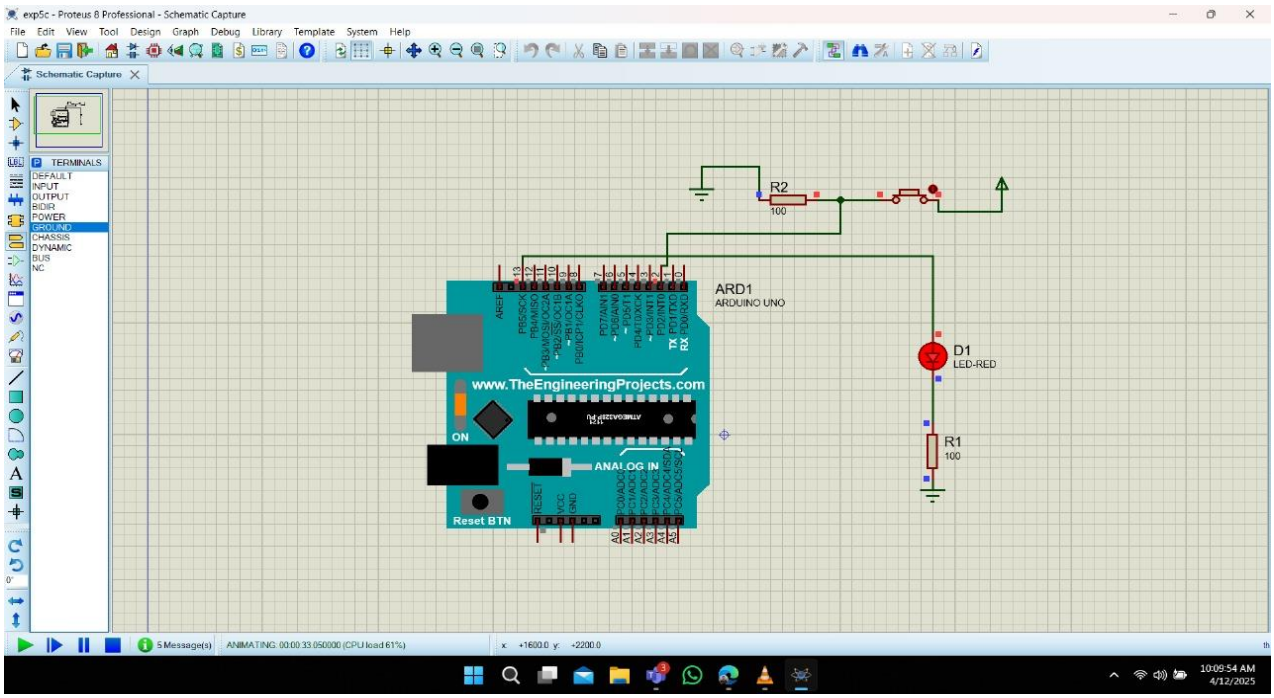


Figure 17: Push button LED control (Button pressed LED ON)

Explanation: In this simulation, the Arduino Uno, a single LED, and a push button were connected based on the hardware setup. The code to control the LED with the push button was written and verified in Arduino IDE 2.3.5, generating a HEX file that was loaded into Proteus. After running the simulation, it was confirmed that pressing the push button toggled the LED ON and OFF as expected, matching the behavior observed on the actual hardware setup.

4.Blinking an LED using a Timer1 ISR:

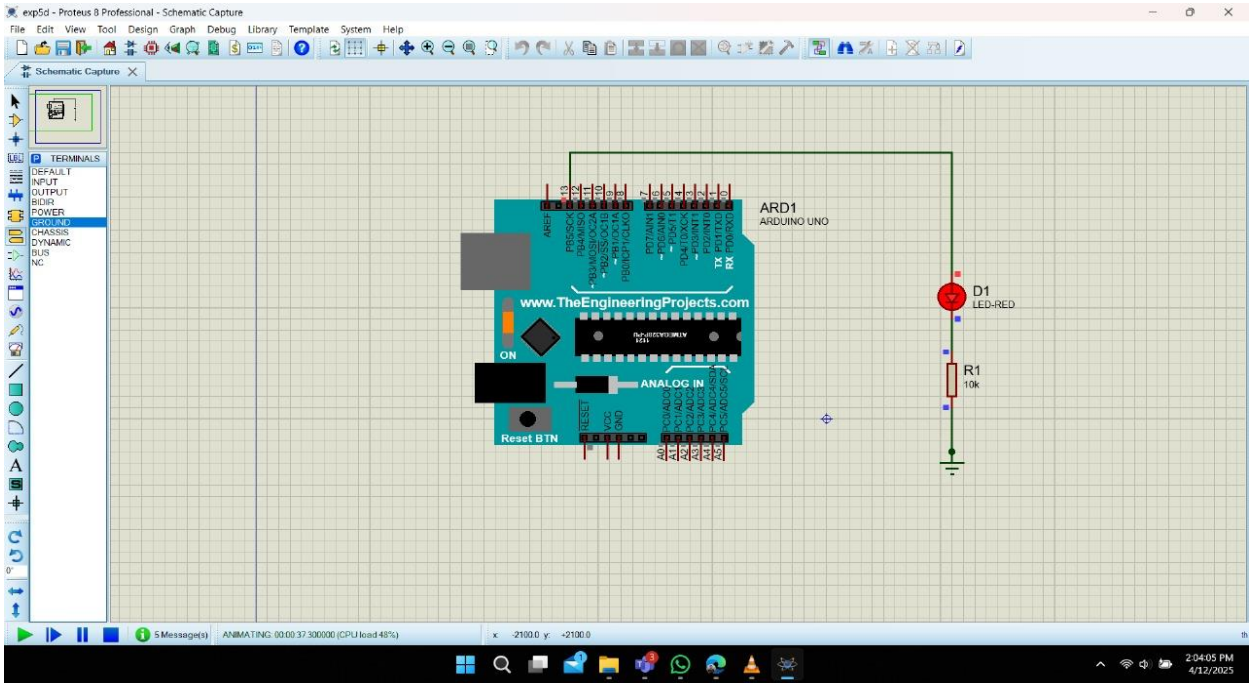


Figure 18**:** Blinking an LED using a Timer1 ISR (LED ON)

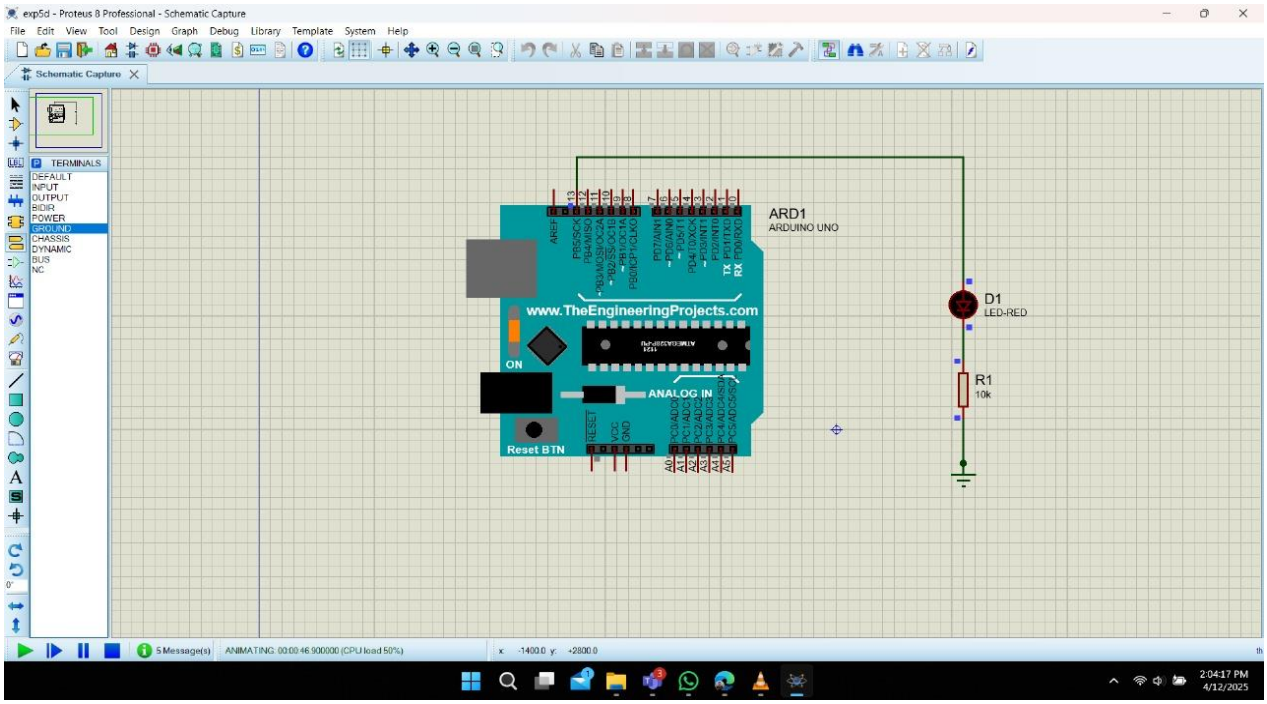


Figure 19**:** Blinking an LED using a Timer1 ISR (LED OFF)

Explanation: In this simulation, the Arduino Uno board, a single LED, and a resistor were connected following the same hardware setup. The program was written using Timer1's Compare Match Interrupt to automatically toggle the LED without needing any code in the loop() function. After uploading the HEX file into Proteus and running the simulation, the LED was observed to blink ON and OFF every 500 milliseconds, confirming that the Timer1 interrupt was working properly and controlling the LED exactly as expected.

PART 5: Controlling an LED using a Pin Change Interrupt ISR:

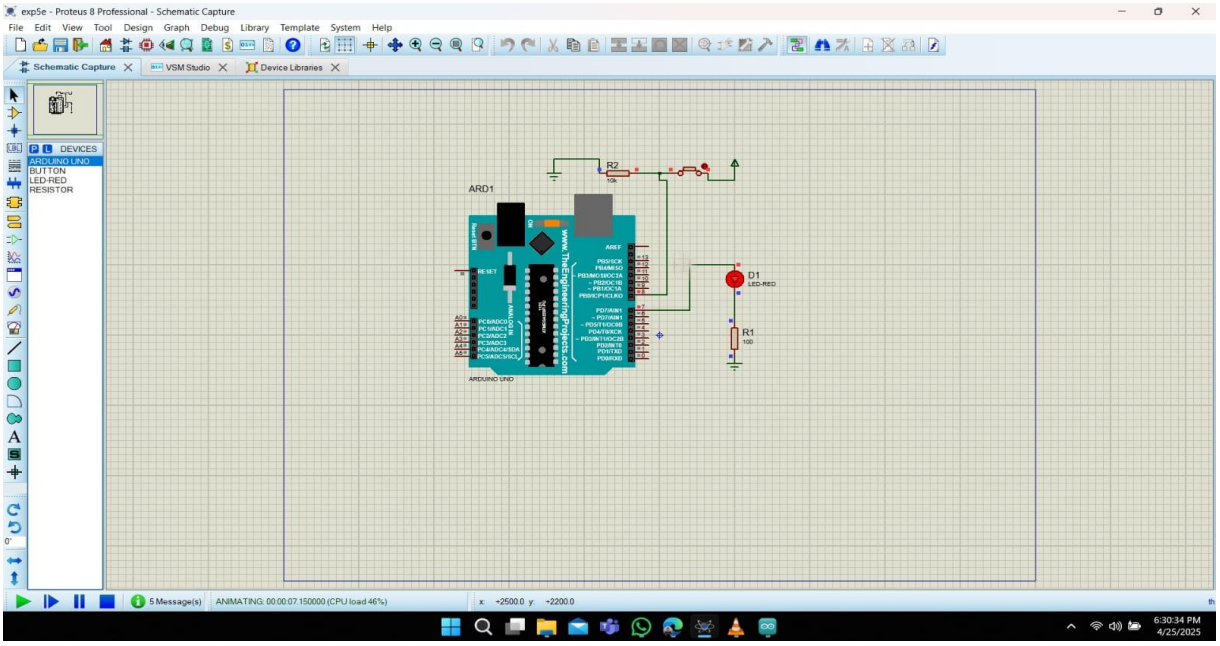


Figure 20**:** Blinking an LED using a Pin Change Interrupt ISR (LED ON)

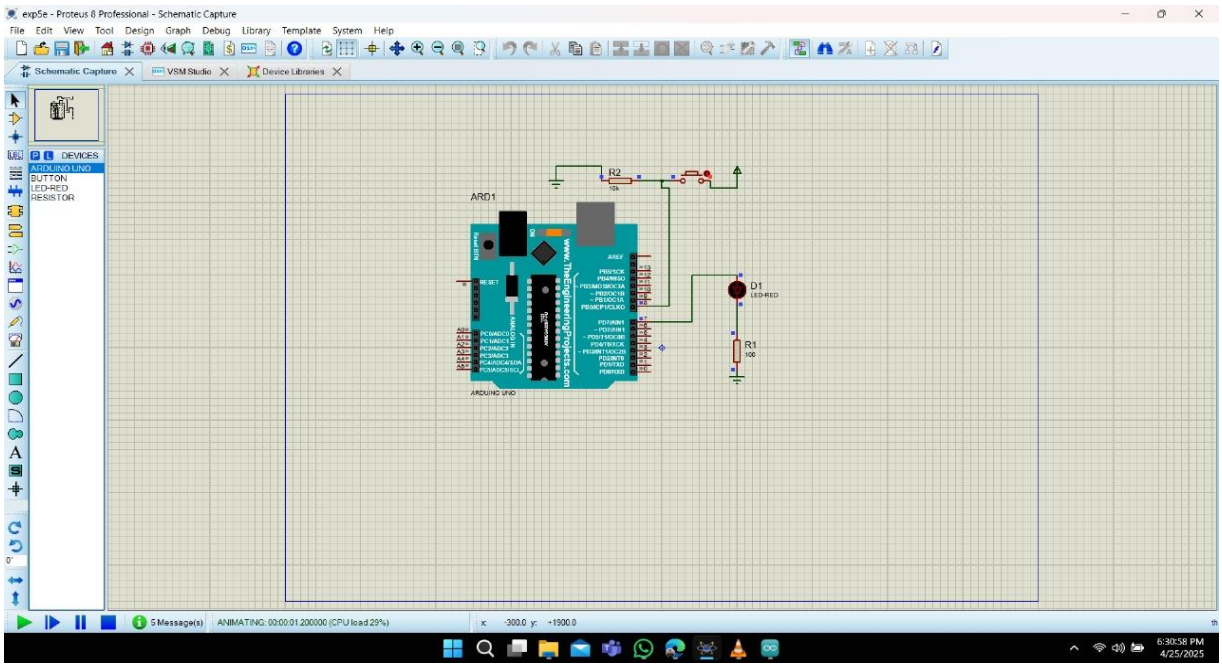


Figure 21**:** Blinking an LED using a Pin Change Interrupt ISR (LED OFF)

Explanation: In this setup, an Arduino Uno board, a resistor, an LED, and a push button were connected based on the hardware configuration. After uploading the code, the LED toggles its state (turns ON if OFF, turns OFF if ON) each time the push button is pressed. The button press triggers a Pin Change Interrupt (PCI) on the rising edge, and inside the interrupt service routine (ISR), the LED's state is inverted. The simulation was run using the HEX file in Proteus, and the LED behavior matched expectations—changing state with every button press just like it would in the real hardware setup.

Answer To Questions:

4. Write a program that will control three LEDs from two switches using the assembly language program.

```
;---------------------------------------------
; Assembly Code: Control 3 LEDs via 2 Buttons
;---------------------------------------------
#define __SFR_OFFSET 0x00
#include "avr/io.h"
;-----------------------
.global start
.global controlLEDs
;=====================================================================
start:
   SBI   DDRB, 0        ; set PB0 (pin D8) as output - LED1
   SBI   DDRB, 1        ; set PB1 (pin D9) as output - LED2
   SBI   DDRB, 2        ; set PB2 (pin D10) as output - LED3
   CBI   DDRD, 2        ; clear PD2 (pin D2) as input - Button1
   CBI   DDRD, 3        ; clear PD3 (pin D3) as input - Button2
   RET
```

```
;============================================================
controlLEDs:
CheckBtn1:
   SBIS  PIND, 2        ; Skip next if Button1 (D2) NOT pressed
   RJMP  CheckBtn2

   ; Button1 pressed: Turn ON LED1 and LED2, OFF LED3
   SBI   PORTB, 0       ; Turn ON LED1
   SBI   PORTB, 1       ; Turn ON LED2
   CBI   PORTB, 2       ; Turn OFF LED3
   RJMP  controlLEDs

CheckBtn2:
   SBIS  PIND, 3        ; Skip next if Button2 (D3) NOT pressed
   RJMP  NoBtn

   ; Button2 pressed: Turn ON LED3, OFF LED1 and LED2
   CBI   PORTB, 0       ; Turn OFF LED1
   CBI   PORTB, 1       ; Turn OFF LED2
   SBI   PORTB, 2       ; Turn ON LED3
   RJMP  controlLEDs

NoBtn:
   ; No button pressed: All LEDs OFF
   CBI   PORTB, 0       ; Turn OFF LED1
   CBI   PORTB, 1       ; Turn OFF LED2
   CBI   PORTB, 2       ; Turn OFF LED3
   RJMP  controlLEDs
```

Explanation: This assembly program effectively demonstrates how to control multiple LEDs based on button inputs using the ATmega328P microcontroller. The program configures three LEDs (red, green, blue) and allows the user to control the LEDs with two buttons. The configuration sets the LED pins as output and button pins as input. Using conditional checks, the program reads the button states and adjusts the corresponding LEDs. When Button 2 (connected to D04) is pressed, it activates the blue LED (PB2) and deactivates the green LED (PB3). Conversely, when the button is released, the blue LED turns off, and the green LED turns on. This logic is repeated continuously, ensuring the LEDs toggle correctly with the user input. Additionally, the assembly code uses efficient instructions for checking the button states and manipulating the PORTB registers to change the LED outputs in real-time. The use of branch instructions allows the program to skip unnecessary steps, making the execution more efficient.

5. Write an LED blink program using Timer2 with a delay of 1 second (the computed value must be loaded onto the OCR2B register) using its Interrupt

```
// blinkLED.ino
bool LED_State = false;

void setup() {
  pinMode(13, OUTPUT);  // Set pin 13 as output for the LED
  cli();  // Disable interrupts while configuring Timer2

  // Reset Timer2 control registers
  TCCR2A = 0;
  TCCR2B = 0;

  // Set prescaler to 1024 (CS22, CS21, CS20 = 1)
  TCCR2B = 0b00000101;

  // Enable interrupt for Timer2 Compare Match B
  TIMSK2 = 0b00000010;

  // Load the computed value to OCR2B (for 1-second delay)
  OCR2B = 15625;

  sei();  // Re-enable interrupts
}

void loop() {
  // The main loop does nothing, as the LED is controlled by Timer2 ISR
}

// Timer2 Compare Match B interrupt service routine
ISR(TIMER2_COMPB_vect) {
  TCNT2 = 0;  // Reset Timer2 to start counting from 0 again
  LED_State = !LED_State;  // Toggle the LED state (on/off)
  digitalWrite(13, LED_State);  // Write the new state to the LED on pin 13
}
```

Explanation: This program uses Timer2 and its Compare Match B Interrupt to blink an LED connected to pin 13 at a 1-second interval. In the setup() function, the LED pin (13) is configured as an output. The program then disables interrupts with cli() to ensure that the timer settings are not interrupted while being configured. The timer is configured by resetting Timer2 control registers (TCCR2A and TCCR2B) and setting the prescaler to 1024, which divides the system clock (16 MHz) by 1024, slowing down the timer's counting speed. This results in a 1-second delay when combined with the compare value in the OCR2B register. The interrupt is enabled by setting the TIMSK2 register, specifically enabling the Compare Match B interrupt, which triggers the interrupt service routine (ISR) when the timer reaches the value in OCR2B. Inside the ISR, the timer is reset (TCNT2 = 0) to start counting from zero again, and the LED's state is toggled, turning it on or off. This process repeats, making the LED blink every second.

7. The answer is already given in the previous section.

Discussion:

This experiment was conducted to understand and gain hands-on experience with assembly language programming and interrupt handling in microcontrollers, particularly using the AVR-based Arduino platform. Through a series of tasks involving LED control, push buttons, and timer-based interrupts, we explored the foundational concepts of low-level programming, hardware-based event handling, and real-time responsiveness in embedded systems. The first part of the experiment involved basic LED blinking using assembly language, allowing us to understand how to directly manipulate microcontroller registers such as PORT, DDRx, and PINx for controlling I/O operations. This gave us a deeper appreciation of how high-level Arduino functions like digitalWrite() and pinMode() actually translate into low-level machine operations.

We then implemented Push Button LED control, where the LED lights up only when the button is pressed. This demonstrated how to monitor input pin states and control output accordingly. We learned about debouncing and logic level detection, which are essential in digital signal handling. By observing how the microcontroller behaves in a polling-based approach, we also realized its limitations—especially the continuous CPU usage and potential. To overcome this limitation, we moved on to Interrupt-driven LED control using a push button, where we used Pin Change Interrupts (PCINT). This method allowed the microcontroller to respond instantly to external events without continuously checking the input pin. The use of Interrupt Service Routines (ISRs) provided us insight into how interrupts pause the current execution to handle high-priority events, making systems more efficient and responsive.

Additionally, we experimented with blinking an LED using Timer1 ISR, which introduced us to timer interrupts. Unlike delay() or software timers, hardware timers operate independently of the main execution flow and provide precise timing with minimal CPU involvement. This is especially useful for tasks that require periodic operations, such as sensor sampling, motor control, or communication protocols.

Finally, we combined Pin Change Interrupts and Timer ISRs to control an LED efficiently based on input conditions and timed events. This helped us learn how to prioritize tasks, manage interrupt flags, and understand the importance of interrupt vector tables in embedded systems.

Here are some real life scenario and application of this experiment:

1. Automatic Door Systems (Shopping Malls, Offices):Interrupts are used to detect motion (via sensors) or button presses to open/close doors. Timers help set how long the door stays open before closing. Assembly-level control ensures fast and reliable operation in commercial embedded controllers.
2. Digital Alarm Clocks: Use hardware timers to keep track of time. Interrupts trigger alarms exactly at preset times. Assembly is often used for efficiency and low power usage in these devices.
3. Home Appliances (Microwave Oven, Washing Machine):Push-button controls use pin change interrupts for user input. Timers manage the operation cycles (e.g., cooking time, spin cycle).Assembly language is used in microcontrollers for optimized memory and execution time.
4. Heart Rate Monitors / Fitness Bands: Sensors generate interrupts for each heartbeat. Timers calculate beats per minute (BPM).Efficient code (often in assembly) ensures low power consumption for longer battery life.
5. Traffic Light Controllers: Timers handle precise ON/OFF durations for LEDs. Interrupts allow emergency override (e.g., from pedestrian buttons or emergency vehicle signals).Code is often written at the register level to optimize real-time performance.
6. Automated Railway Crossing Gates: Interrupts from approaching train sensors trigger the gate operation.Timers control signal light duration and barrier timing. Microcontrollers here often use low-level programming for reliability.
7. Remote-Controlled Toys and Devices: Button presses generate interrupts to control motors or lights. Assembly programs are used to reduce power and enhance performance on small chips.

Conclusion:

This experiment helped us gain practical knowledge of assembly language programming and the use of interrupts in microcontrollers. We explored how hardware interrupts like external, pin change, and timer interrupts allow responsive and efficient control of devices like LEDs and switches. By implementing LED blink, switch-controlled LEDs, and Timer1-based blinking using ISRs, we understood how real-time tasks can be managed without blocking the processor. This hands-on experience strengthened our foundation for developing fast, low-power, and event-driven embedded systems, which are essential for real-life applications such as home automation, traffic control, and medical devices.

References:

[1] **https://www.arduino.cc/.**
[2] **https://www.educba.com**
[3] **https://www.geeksforgeeks.org/led-blinking-using-arduino**