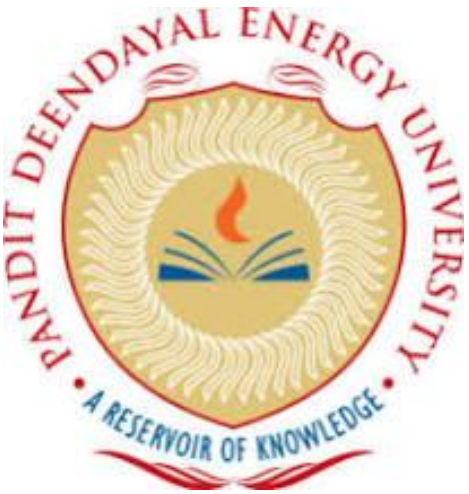
Embedded Systems Laboratory (20IC304P)

Lab Report Submitted to

Pandit Deendayal Energy University, Gandhinagar

for



Bachelor of Technology

in

Information and Communication Technology Department

Submitted by
Jay Patel
22BIT138

Course Faculty
Gunjan Thakur

Department of Information Communication and Technology
School of Technology
PANDIT DEENDAYAL ENERGY UNIVERSITY, GANDHINAGAR - 382007, India

FEB-2024

EXP NO.	EXPERIMENT TITLE	DATE	SIGNATURE

Experiment 1

Aim: Familiarization with IDE and trainer kits/boards

Objective: To familiarize students with the Integrated Development Environment (IDE) and the hardware trainer kits/boards used in embedded systems development, including:

1. Proteus for simulation
2. Arduino IDE for programming (flashing)
3. ATmega2560 development board for hardware validation

Introduction: This experiment aims to familiarize students with the tools and hardware used in embedded systems development. The focus is on understanding how to use an Integrated Development Environment (IDE) such as Arduino IDE for programming and Proteus for simulation, along with the ATmega2560 development board for hardware validation. These tools are essential for designing, testing, and implementing embedded system applications, bridging the gap between theory and practical implementation.

Circuit Components:

- PC with Proteus software installed
- Arduino IDE installed on PC
- ATmega2560 development board
- USB cable for programming
- Power supply for the development board
- Basic electronic components (e.g., LEDs, resistors, and jumper wires)

Key Concepts: An Integrated Development Environment (IDE) is a software suite that consolidates tools needed for software development. In embedded systems, it typically includes a code editor, compiler, debugger, and simulation tools.

- **Proteus:** A simulation and circuit design software that allows users to design and test embedded systems virtually before hardware implementation.

- **Arduino IDE:** An open-source software that provides a user-friendly platform for writing, compiling, and uploading code to microcontrollers.
- **ATmega2560 Development Board:** A microcontroller board based on the ATmega2560, widely used for learning and prototyping embedded systems projects.

Procedure:

1. **Setting up the IDE:**
    - Install and launch Arduino IDE on the PC.
    - Configure the necessary compiler and toolchain.
    - Familiarize yourself with the interface, including the code editor, debugging tools, and output console.
2. **Simulation in Proteus:**
    - Design a basic circuit with the ATmega2560 in Proteus.
    - Write and simulate a blinking LED program.
    - Verify the circuit functionality before proceeding to hardware implementation.
3. **Connecting the ATmega2560 Development Board:**
    - Connect the development board to the PC using a USB cable.
    - Install necessary drivers if required.
    - Open the Arduino IDE and select the correct board and port settings.
4. **Writing and Uploading a Simple Program:**
    - Write a basic program (e.g., blinking an LED) in Arduino IDE.
    - Compile and debug the program within the IDE.
    - Upload the program to the ATmega2560 board.
    - Observe the execution of the program on the board.
5. **Interfacing with Basic Peripherals:**
    - Connect external components like LEDs, switches, or sensors to the development board.
    - Modify the program to read input from a switch or sensor and control an output (e.g., turn on an LED when a button is pressed).
    - Upload and test the updated program.
6. **Understanding Debugging and Serial Communication:**
    - Use debugging tools within the IDE to step through the program.
    - Send and receive data via the serial monitor or terminal.
    - Analyse the output and troubleshoot any issues.

Diagrams:

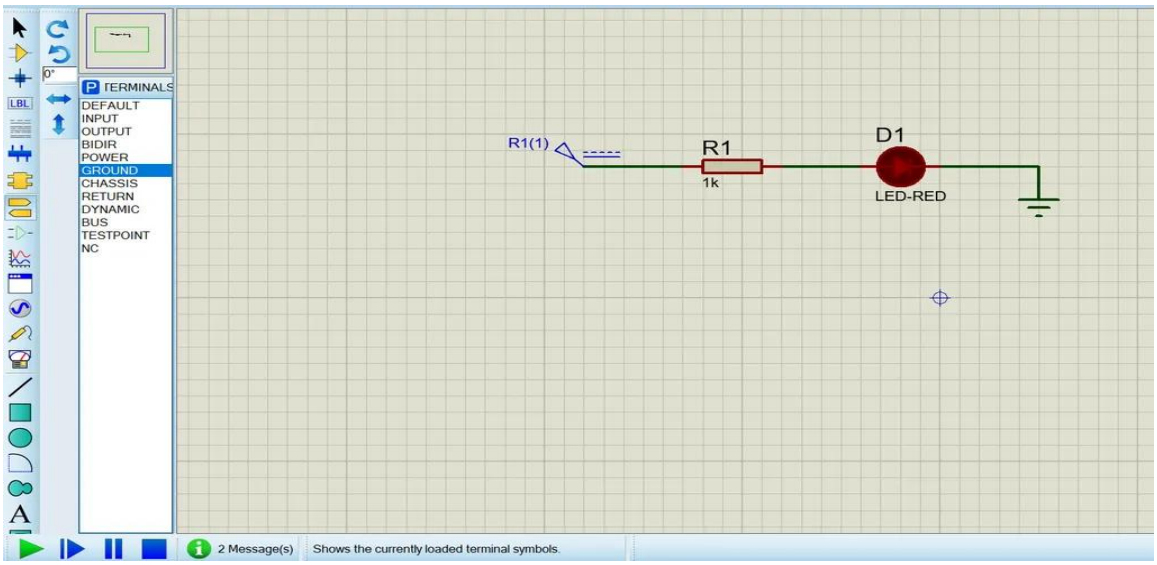


Fig. 1: Flashing LED in Proteus

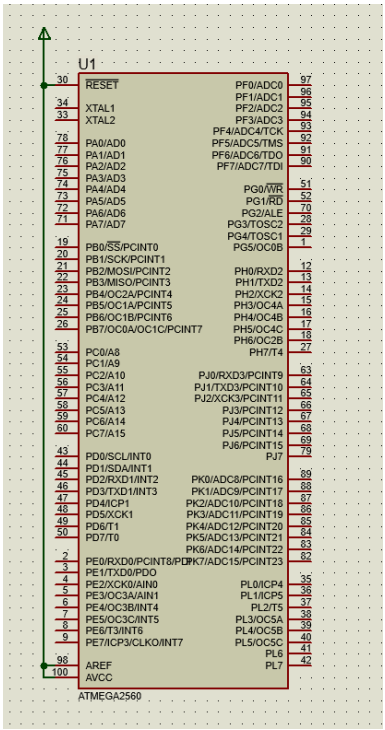


Fig. 2: ATMega2560 circuit in Proteus

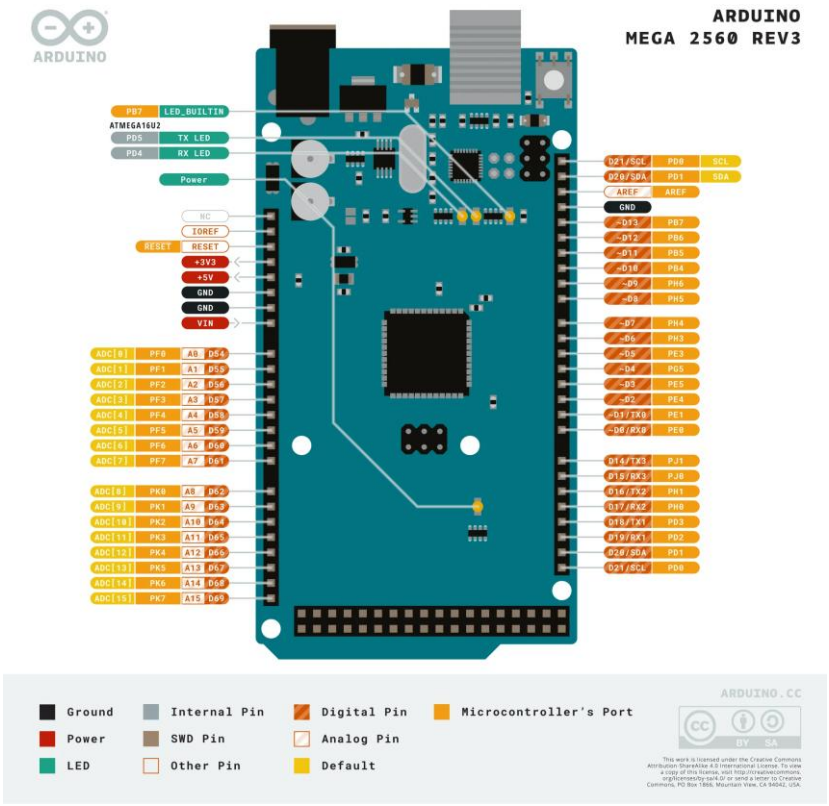


Fig.3: ATMega2560 Pin Diagram

Applications:

- **Embedded Systems Prototyping:** Quick development and testing of projects, including robotics and IoT applications.
- **Simulation and Testing:** Proteus allows virtual testing of circuits and microcontroller code, reducing dependency on physical hardware for initial validation.
- **Real-Time Systems Development:** ATmega2560's support for PWM, UART, and timers makes it suitable for real-time applications like motor control and automation.
- **Sensor Integration and Monitoring:** Widely used in data acquisition systems for environmental monitoring, industrial automation, and more.

Conclusion: This experiment provides practical exposure to using IDEs, simulation tools, and microcontroller development boards. Students learn how to interface components, debug programs, and simulate circuits, which are crucial for embedded system development.

Experiment 2

Aim:

(A) Blink a LED at regular intervals with the help of AVR2560.
(B) Generate a specific LED pattern- LED Blinking 1 to 8.

Components Required:

1. Proteus Software
2. ATmega2560 microcontroller module in Proteus library.
3. LEDs (virtual in Proteus).
4. Resistors (220Ω for current limiting).

Theory:

The **ATmega2560** is an advanced microcontroller based on the **8-bit AVR RISC architecture**. It is designed by Microchip Technology and is widely used in embedded systems and applications requiring high performance, flexibility, and cost-effectiveness. Below is an overview of its architecture, features, and capabilities**:**

1. **High-Performance Architecture:**
   - 8-bit AVR CPU with RISC architecture.
   - Executes most instructions in a single clock cycle.
   - Up to 16 MIPS throughput at 16 MHz.
2. **Memory:**
   - Flash Memory: 256 KB of in-system self-programmable memory for storing programs.
   - SRAM: 8 KB for data storage.
   - EEPROM: 4 KB for storing non-volatile data.
   - Read-While-Write capabilities for Flash memory.
3. **Clock Speed:**
   - Maximum clock frequency of 16 MHz.
   - Internal and external clock source support with programmable clock prescalers.
4. **I/O Ports:**
   - 86 general-purpose input/output (GPIO) lines.
   - Programmable pull-up resistors.
5. **Peripheral Features:**
   - Timers/Counters: 6 timers (4 with PWM capability).

- USART: 4 Universal Synchronous/Asynchronous Receiver Transmitters.
- SPI: Serial Peripheral Interface for high-speed synchronous data transfer.
- I^2C (TWI): Two-Wire Interface for communication with other devices.
- ADC: 16-channel, 10-bit ADC.
- PWM: Pulse Width Modulation support on several pins.
- Analog Comparator: Built-in analog comparator.

6. **Interrupts:**
   - 73 interrupt vectors.
   - Programmable priority levels.
7. **Power Management:**
   - Various power-saving modes: Idle, Power-Down, Power-Save, Standby, and Extended Standby.
   - Brown-Out Detection (BOD) with programmable trigger levels.
8. **Development Support:**
   - JTAG (IEEE 1149.1 compliant) interface for on-chip debugging.
   - Bootloader support for firmware updates without an external programmer.

Pin Configuration:

The ATmega2560 comes in several package types (TQFP, VQFN, etc.) with 100 pins. Below is a summary of its pin functionality:

1. **Power Pins:**
   - VCC: Digital supply voltage.
   - GND: Ground.
   - AVCC: Supply voltage for the ADC.
2. **Port Pins:**
   - PORTA to PORTK for GPIO.
   - Configurable as input or output pins.
3. **Special Function Pins:**
   - XTAL1/XTAL2: External clock input/output.
   - RESET: External reset input.
   - JTAG: Debugging interface.

Development Tools:

- Arduino Mega 2560: A popular development board based on the ATmega2560.
- AVR Studio/Atmel Studio: IDE for programming and debugging.
- Programmers: USBasp, AVRISP, etc., for burning the firmware.

Diagrams:

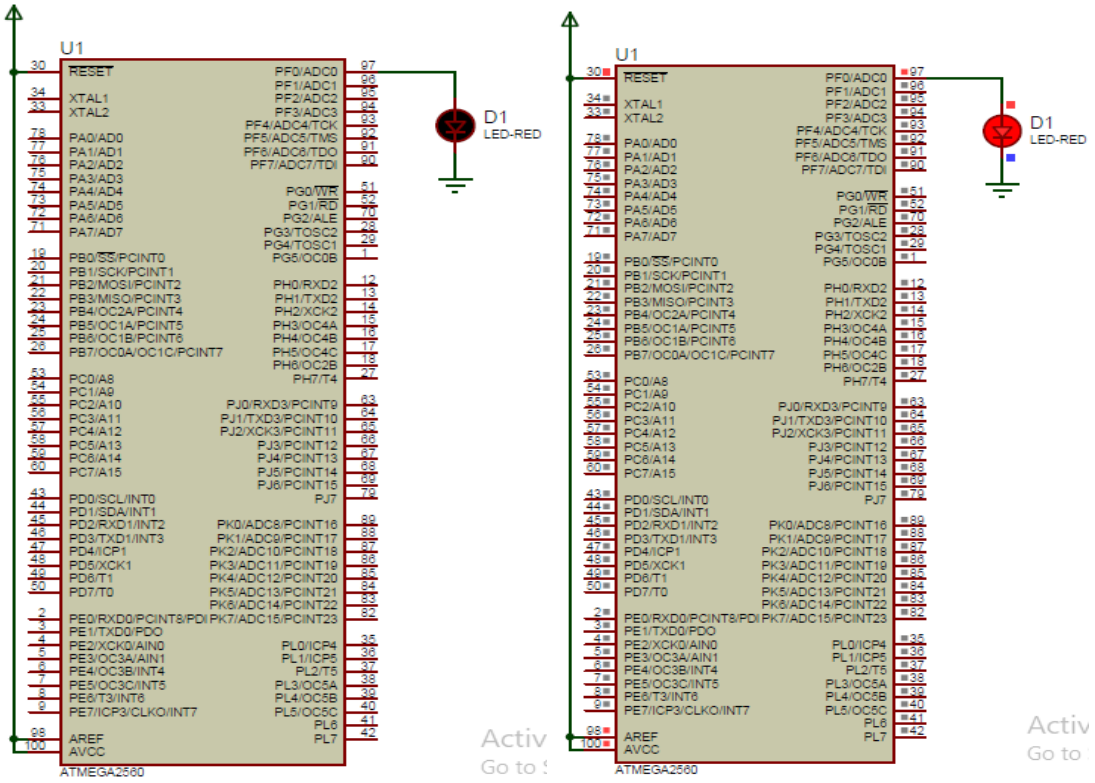
(A) Blink a LED at regular intervals:

Code-I:

main.c

```c
#include <avr/io.h>
#include <util/delay.h>

#define LED_PIN PF0

int main(void) {
    DDRF = 0x01; // Configure PF0 as output

    while (1) {
        // Turn the LED ON
        PORTF = 0x01;      // Set PF0 HIGH
        _delay_ms(1000);   // Wait for 1 second

        // Turn the LED OFF
        PORTF = 0x00;      // Set PF0 LOW
        _delay_ms(1000);   // Wait for 1 second
    }
}
```

Output:

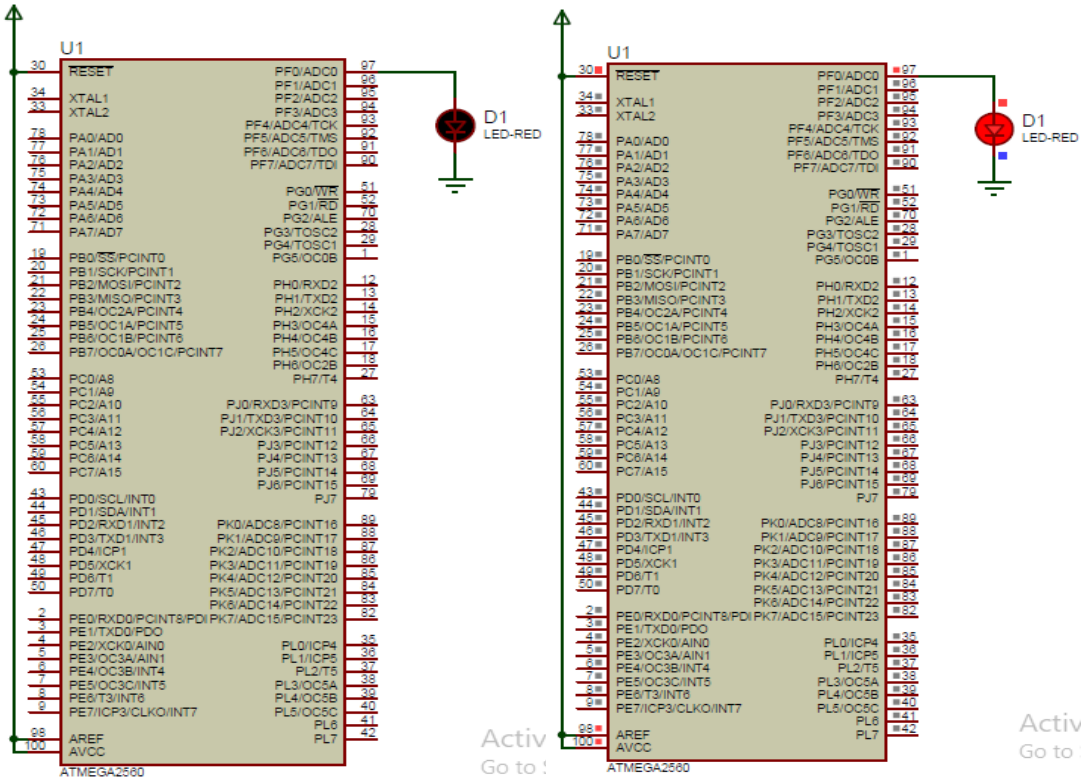


Code-II:

*main.c

```c
#include <avr/io.h>
#include <util/delay.h>

// Define LED pin as PB0
#define LED_PIN PF0

int main(void) {
    // Configure PB0 as output
    DDRF |= (1 << LED_PIN); // Set PF0 as output by setting its bit in DDRB

    while (1) {
        // Turn the LED ON
        PORTF = 0x01;              // Set PF0 HIGH
        _delay_ms(1000);          // Wait for 1 second

        // Turn the LED OFF
        PORTF = 0x00;               // Set PF0 LOW
        _delay_ms(1000);           // Wait for 1 second
    }
}
```

Output:



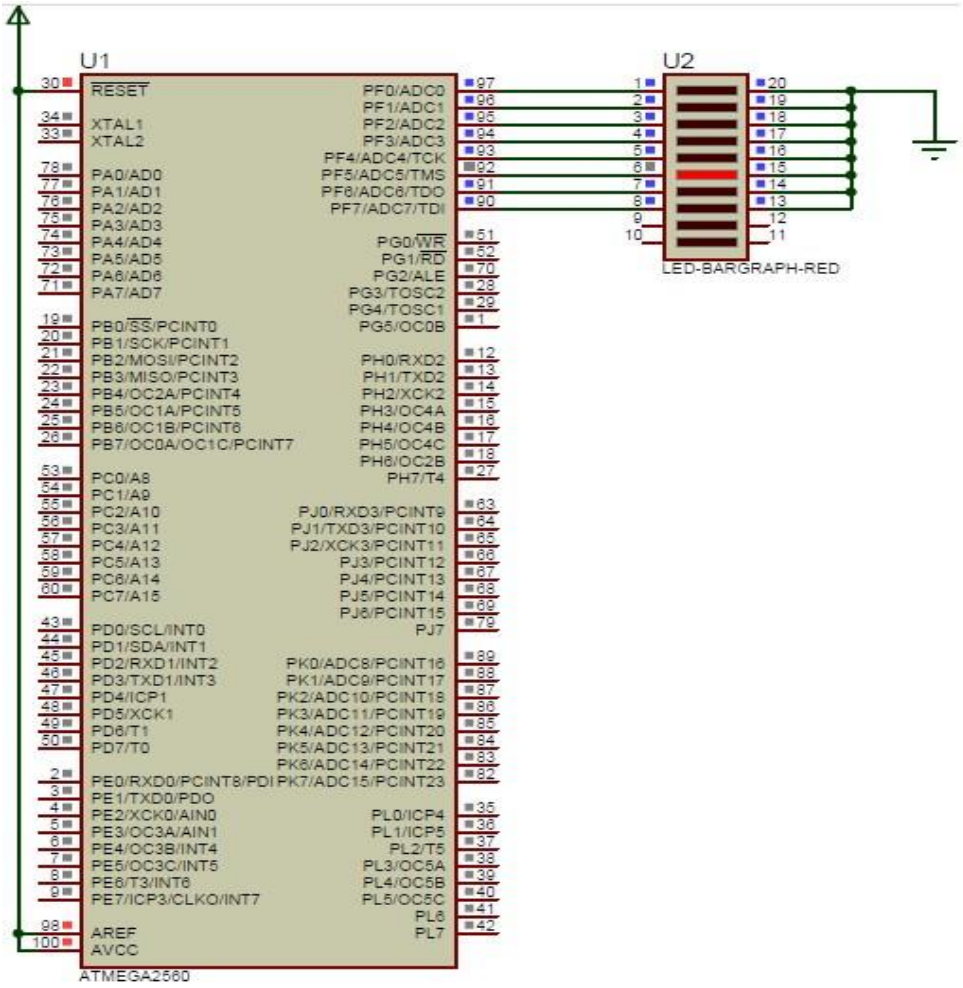
(B) Generate a sequential LED pattern:

Code:

main.c

```c
#ifndef F_CPU
#define F_CPU 8000000UL // Clock speed is 8 MHz
#endif

#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRF = 0xFF; // Configure PORTF as output (set all pins of PORTF to output)
    PORTF = 0x00; // Initialize PORTF with all pins LOW (turn off all LEDs initially)

    int i;
    while (1)
    {
        // Loop through each pin in PORTF (PF0 to PF7)
        for (i = 0; i < 8; i++)
        {
            PORTF = (0x01 << i); // Turn on one LED at a time (shifting bit left)
            _delay_ms(500);      // Delay for 500 milliseconds to make the LED visible
        }
    }
}
```

Output:



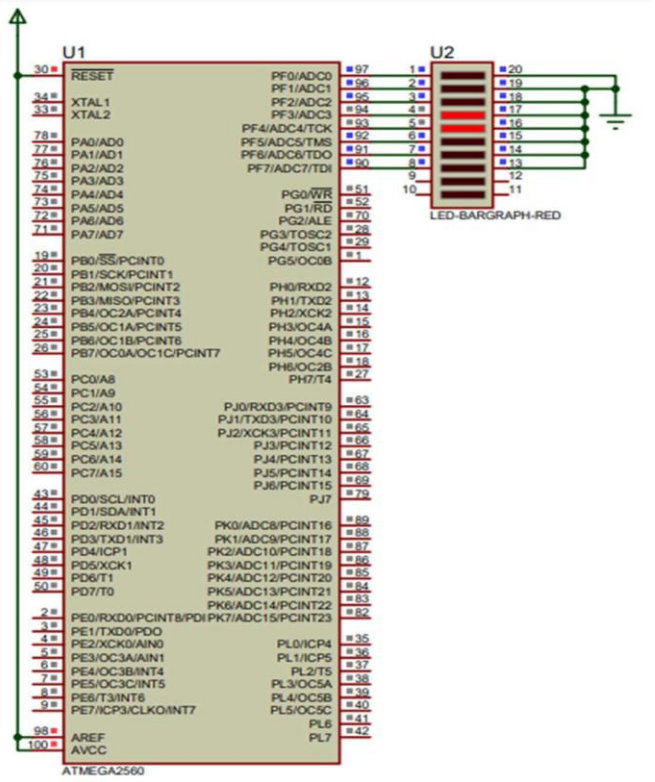
(C) Generate a specific LED pattern:

Code:

main.c

```c
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <util/delay.h>

int main()
 {
    DDRK = 0xFF;
    int arr[]={0,24,60,126,255,126,60,24};
    int i;
    while (1)
    {
          for(i=0;i<8;i++)
          {
          PORTK = arr[i];
          _delay_ms(500);
          }
    }
    return 0;
 }
```

Output:



Atmega 2560 Specifications:

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

Applications:

- Robotics and automation.
- Internet of Things (IoT) devices.
- Industrial control systems.
- Data acquisition and processing systems.
- Educational and research projects.

Conclusion:

This experiment successfully demonstrated LED control using the AVR2560 microcontroller, focusing on blinking and pattern generation.

1. LED Blinking at Regular Intervals – By utilizing delay functions or timer interrupts, we achieved consistent and precise LED blinking, highlighting the importance of timing control.
2. Sequential LED Blinking (1 to 8) – The LEDs were programmed to turn on in a sequence from LED 1 to LED 8, confirming the correct implementation of looping logic and GPIO manipulation.
3. LED Blinking from the Center Outward – The LEDs illuminated outward from the center, showcasing the ability to create custom patterns through efficient port control.

Overall, this experiment reinforced fundamental concepts of embedded systems, including GPIO handling, timing mechanisms, and structured programming, proving the AVR2560's effectiveness in LED-based applications.

Experiment 3

Aim:

To program the ATmega2560 to:

1. Display numbers on a multi-digit 7-segment display.
2. Implement a counter that increments and displays the count on the 7-segment display.

Components Required:

1. Proteus Software (for simulation).
2. ATmega2560 microcontroller module in Proteus library.
3. Multi-digit 7-segment display (common cathode or common anode, depending on the setup).
4. Resistors (220 Ω for current limiting).
5. Arduino IDE for code generation.

Key Concepts:

1. **7-Segment Display**:

   A 7-segment display is made up of 7 LEDs arranged to form the digits 0-9. Each segment can be controlled to form a number or a letter. In a multi-digit display, several 7-segment displays are connected to show multiple digits.

2. **Digit Encoding**:

   Each digit (0-9) is represented by turning on specific segments of the 7-segment display. The segments are typically labeled as A to G, and specific patterns of HIGH (on) and LOW (off) are used to display each digit.

3. **Timing Control**:

   A counter is implemented to cycle through the digits of the display. Delays and multiplexing are used to control the timing of each digit.

Diagrams:

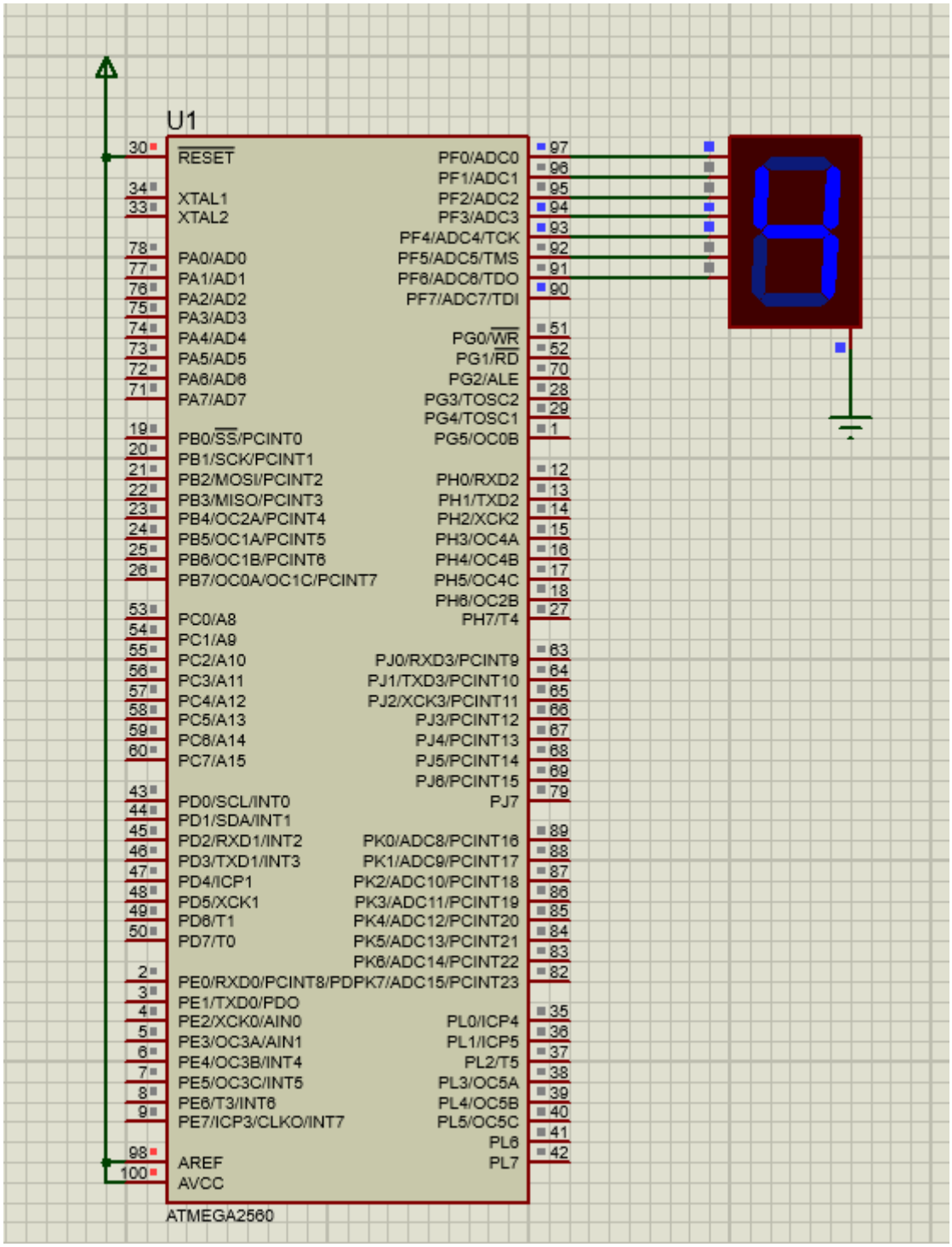
1. Counter - Single Digit - Common Cathode 7-Segment Display:

Code:

main.c

```c
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <util/delay.h>

int main()
{
    DDRF = 0xFF;
    int arr[] = {0x3F, // 0: A, B, C, D, E, F (G = OFF)
                 0x06, // 1: B, C
                 0x5B, // 2: A, B, D, E, G
                 0x4F, // 3: A, B, C, D, G
                 0x66, // 4: B, C, F, G
                 0x6D, // 5: A, C, D, F, G
                 0x7D, // 6: A, C, D, E, F, G
                 0x07, // 7: A, B, C
                 0x7F, // 8: A, B, C, D, E, F, G
                 0x6F};// 9: A, B, C, D, F, G
    int i;
    while (1)
    {
        for (i = 0; i < 10; i++)
        {
            PORTF = arr[i];
            _delay_ms(500);
        }
    }
    return 0;
}
```

Output:


U1
RESET
XTAL1
XTAL2
PA0/AD0
PA1/AD1
PA2/AD2
PA3/AD3
PA4/AD4
PA5/AD5
PA6/AD6
PA7/AD7
PB0/SS/PCINT0
PB1/SCK/PCINT1
PB2/MOSI/PCINT2
PB3/MISO/PCINT3
PB4/OC2A/PCINT4
PB5/OC1A/PCINT5
PB6/OC1B/PCINT6
PB7/OC0A/OC1C/PCINT7
PC0/A8
PC1/A9
PC2/A10
PC3/A11
PC4/A12
PC5/A13
PC6/A14
PC7/A15
PD0/SCL/INT0
PD1/SDA/INT1
PD2/RXD1/INT2
PD3/TXD1/INT3
PD4/ICP1
PD5/XCK1
PD6/T1
PD7/T0
PE0/RXD0/PCINT8/PDPK7/ADC15/PCINT23
PE1/TXD0/PDO
PE2/XCK0/AIN0
PE3/OC3A/AIN1
PE4/OC3B/INT4
PE5/OC3C/INT5
PE6/T3/INT6
PE7/ICP3/CLKO/INT7
AREF
AVCC
PF0/ADC0
PF1/ADC1
PF2/ADC2
PF3/ADC3
PF4/ADC4/TCK
PF5/ADC5/TMS
PF6/ADC6/TDO
PF7/ADC7/TDI
PG0/WR
PG1/RD
PG2/ALE
PG3/TOSC2
PG4/TOSC1
PG5/OC0B
PH0/RXD2
PH1/TXD2
PH2/XCK2
PH3/OC4A
PH4/OC4B
PH5/OC4C
PH6/OC2B
PH7/T4
PJ0/RXD3/PCINT9
PJ1/TXD3/PCINT10
PJ2/XCK3/PCINT11
PJ3/PCINT12
PJ4/PCINT13
PJ5/PCINT14
PJ6/PCINT15
PJ7
PK0/ADC8/PCINT16
PK1/ADC9/PCINT17
PK2/ADC10/PCINT18
PK3/ADC11/PCINT19
PK4/ADC12/PCINT20
PK5/ADC13/PCINT21
PK6/ADC14/PCINT22
PL0/ICP4
PL1/ICP5
PL2/T5
PL3/OC5A
PL4/OC5B
PL5/OC5C
PL6
PL7
ATMEGA2560

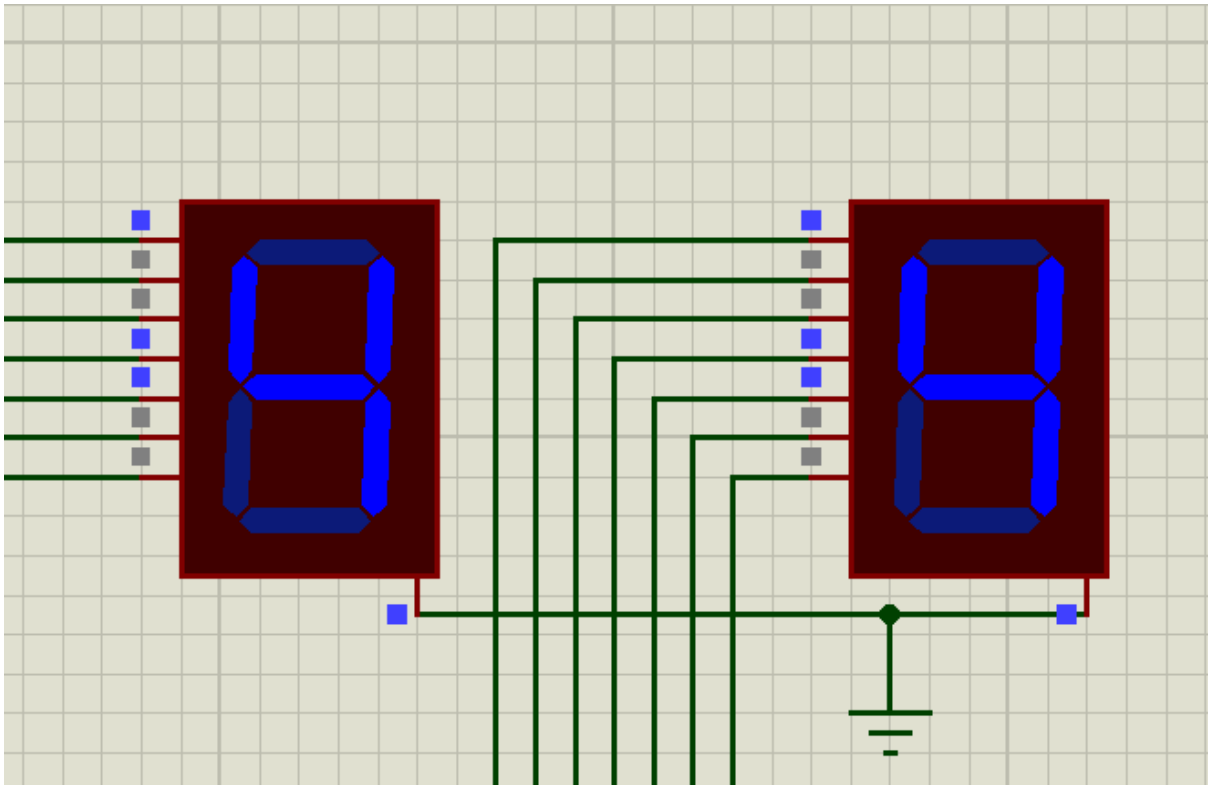
2. Counter – Multi Digit - Common Cathode 7-Segment Display:

Code:

main.c

```c
#include <avr/io.h>
#include <util/delay.h>

#define SEGMENT_PORT_MSD PORTF   // Define the port for the most significant digit
#define SEGMENT_DDR_MSD  DDRF    // Define the data direction register for PORTF
#define SEGMENT_PORT_LSD PORTK   // Define the port for the least significant digit
#define SEGMENT_DDR_LSD  DDRK    // Define the data direction register for PORTK

// Segment codes for digits 0-9 (common cathode)
uint8_t segment_codes[] = {
    0x3F, // 0: A, B, C, D, E, F (G = OFF)
    0x06, // 1: B, C
    0x5B, // 2: A, B, D, E, G
    0x4F, // 3: A, B, C, D, G
    0x66, // 4: B, C, F, G
    0x6D, // 5: A, C, D, F, G
    0x7D, // 6: A, C, D, E, F, G
    0x07, // 7: A, B, C
    0x7F, // 8: A, B, C, D, E, F, G
    0x6F  // 9: A, B, C, D, F, G
};

int main(void) {
    uint8_t tens, units;               // Variables for tens and units digits
    SEGMENT_DDR_MSD = 0xFF;            // Set PORTF as output (MSD)
    SEGMENT_DDR_LSD = 0xFF;            // Set PORTK as output (LSD)
    SEGMENT_PORT_MSD = 0x00;           // Initialize PORTF to 0 (all segments off)
    SEGMENT_PORT_LSD = 0x00;           // Initialize PORTK to 0 (all segments off)

    while (1) {
        for (tens = 0; tens < 10; tens++) { // Loop through tens digit (0-9)
            for (units = 0; units < 10; units++) { // Loop through units digit (0-9)
                SEGMENT_PORT_MSD = segment_codes[tens]; // Display tens digit on PORTF
                SEGMENT_PORT_LSD = segment_codes[units]; // Display units digit on PORTK
                _delay_ms(500); // Delay for 500 milliseconds
            }
        }
    }
}
```

Output:



Conclusion:

This experiment successfully programmed the **ATmega2560** to control a **two-digit 7-segment display** and implement a **00-99 counter**. The display correctly mapped segment codes and incremented every **500ms**, demonstrating proper **counting logic and multiplexing**.

Experiment 4

Aim:

Digital Input Handling Using Push Buttons.

1. Embedded C code to interface the push button switch with the ATmega2560.
2. Modify the program so that pressing the button toggles the LED state (ON/OFF) instead of directly controlling it.

Components Required:

1. ATmega2560 Microcontroller Development Board (e.g., Arduino Mega or standalone ATmega2560 board)
2. Push Button Switch
3. Resistors (1kΩ pull-down resistor)
4. LED (for output indication)
5. Seven Segment Display
6. Breadboard and Connecting Wires
7. Power Supply (or USB connection for programming and power)

Theory:

Push buttons are widely used in embedded systems for user interaction. They are simple mechanical switches that, when pressed, establish an electrical connection, allowing the microcontroller to detect user input. The ATmega2560 microcontroller can read push button inputs and respond accordingly by controlling output devices like LEDs.

In this experiment, we study how to interface a push button with the ATmega2560 and

Implement a function where pressing the button toggles the LED state.

1. **Interfacing a Push Button with ATmega2560**

    - The push button is connected to a GPIO pin configured as an input.
    - A pull-up or pull-down resistor is used to ensure a stable logic level when the button is not pressed.

- When the button is pressed, the circuit completes, and the microcontroller detects a state change.

Two Common Configurations:

1. **Pull-Down Resistor Configuration (Active-High Button):**
   - The input pin is LOW (0) by default.
   - Pressing the button pulls the pin HIGH (1), which is detected by the microcontroller.
2. **Pull-Up Resistor Configuration (Active-Low Button)**
   - The input pin is HIGH (1) by default.
   - Pressing the button pulls the pin LOW (0), which is detected by the microcontroller.
   - Internal pull-up resistors can be enabled in ATmega2560 to avoid external components.

2. Direct Button-Controlled LED

- The microcontroller continuously checks the button state.
- When pressed, it turns the LED ON, and when released, the LED turns OFF.
- This approach is simple but does not allow the LED to stay ON after releasing the button.

Steps:

- Configure the button pin as an input with a pull-up resistor.
- Configure the LED pin as an output.
- Continuously read the button state and control the LED accordingly.

3. Implementing Toggle Functionality

Instead of directly controlling the LED, we modify the program to toggle the LED state on each button press.

Concept:

- When the button is pressed, the LED changes state (ON → OFF or OFF → ON).

- A debounce delay is added to avoid multiple detections due to mechanical vibrations.
- The program waits for the button to be released before detecting the next press.

Steps:

- Detect a button press and implement a debounce delay.
- Wait for the button to be released before proceeding.
- Use bitwise XOR operation to toggle the LED state.

4. **Timing and Debounce Handling**

- Mechanical switches do not change states instantly. When pressed, they bounce between ON and OFF multiple times within a few milliseconds.
- A debounce delay (typically 50-100 ms) ensures the microcontroller registers a single press.
- Delays or interrupt-based detection can be used for stable input reading.

5. **Programming Approach**

- Configure GPIO pins for input (button) and output (LED).
- Implement a function to detect button presses and debounce.
- Use logical operations to toggle the LED on each press.
- Ensure the program efficiently waits for a complete press-release cycle before detecting the next press.

Diagrams:

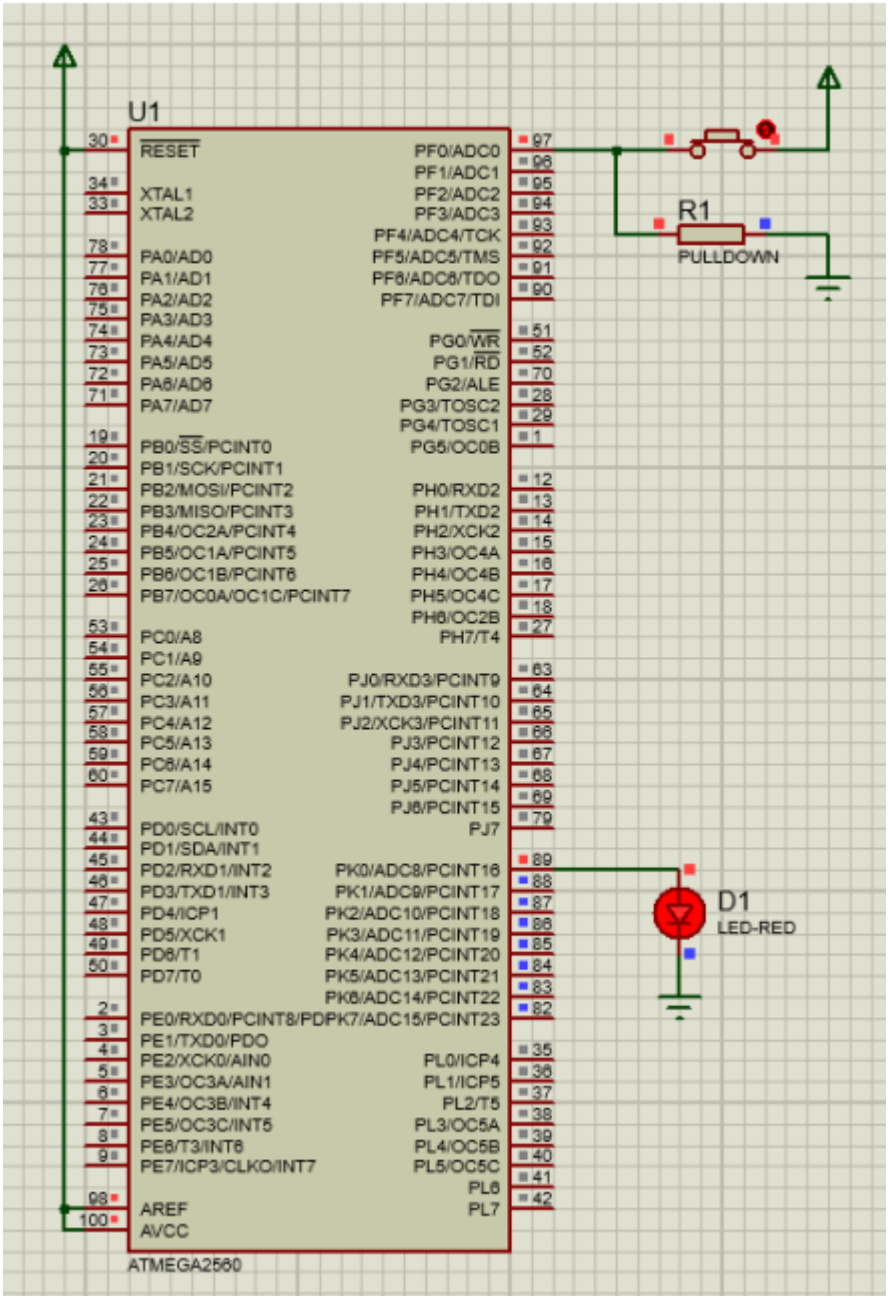
1. **Embedded C code to interface the push button switch with the ATmega2560:**

   Code:

main.c

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <util/delay.h>

int main()
 {
    DDRF=0x00;
    DDRK=0xFF;

    while(1){

        if(PINF == 0x01){
            PORTK = 0x01;
            _delay_ms(500);
        }
        else{
            PORTK = 0x00;
        }
    }
    return 0;
 }
```

Output:


U1
RESET
XTAL1
XTAL2
PA0/AD0
PA1/AD1
PA2/AD2
PA3/AD3
PA4/AD4
PA5/AD5
PA6/AD6
PA7/AD7
PB0/SS/PCINT0
PB1/SCK/PCINT1
PB2/MOSI/PCINT2
PB3/MISO/PCINT3
PB4/OC2A/PCINT4
PB5/OC1A/PCINT5
PB6/OC1B/PCINT6
PB7/OC0A/OC1C/PCINT7
PC0/A8
PC1/A9
PC2/A10
PC3/A11
PC4/A12
PC5/A13
PC6/A14
PC7/A15
PD0/SCL/INT0
PD1/SDA/INT1
PD2/RXD1/INT2
PD3/TXD1/INT3
PD4/ICP1
PD5/XCK1
PD6/T1
PD7/T0
PE0/RXD0/PCINT8/PDI
PE1/TXD0/PDO
PE2/XCK0/AIN0
PE3/OC3A/AIN1
PE4/OC3B/INT4
PE5/OC3C/INT5
PE6/T3/INT6
PE7/ICP3/CLKO/INT7
AREF
AVCC
ATMEGA2560
PF0/ADC0
PF1/ADC1
PF2/ADC2
PF3/ADC3
PF4/ADC4/TCK
PF5/ADC5/TMS
PF6/ADC6/TDO
PF7/ADC7/TDI
PG0/WR
PG1/RD
PG2/ALE
PG3/TOSC2
PG4/TOSC1
PG5/OC0B
PH0/RXD2
PH1/TXD2
PH2/XCK2
PH3/OC4A
PH4/OC4B
PH5/OC4C
PH6/OC2B
PH7/T4
PJ0/RXD3/PCINT9
PJ1/TXD3/PCINT10
PJ2/XCK3/PCINT11
PJ3/PCINT12
PJ4/PCINT13
PJ5/PCINT14
PJ6/PCINT15
PJ7
PK0/ADC8/PCINT16
PK1/ADC9/PCINT17
PK2/ADC10/PCINT18
PK3/ADC11/PCINT19
PK4/ADC12/PCINT20
PK5/ADC13/PCINT21
PK6/ADC14/PCINT22
PK7/ADC15/PCINT23
PL0/ICP4
PL1/ICP5
PL2/T5
PL3/OC5A
PL4/OC5B
PL5/OC5C
PL6
PL7
R1
PULLDOWN
D1
LED-RED

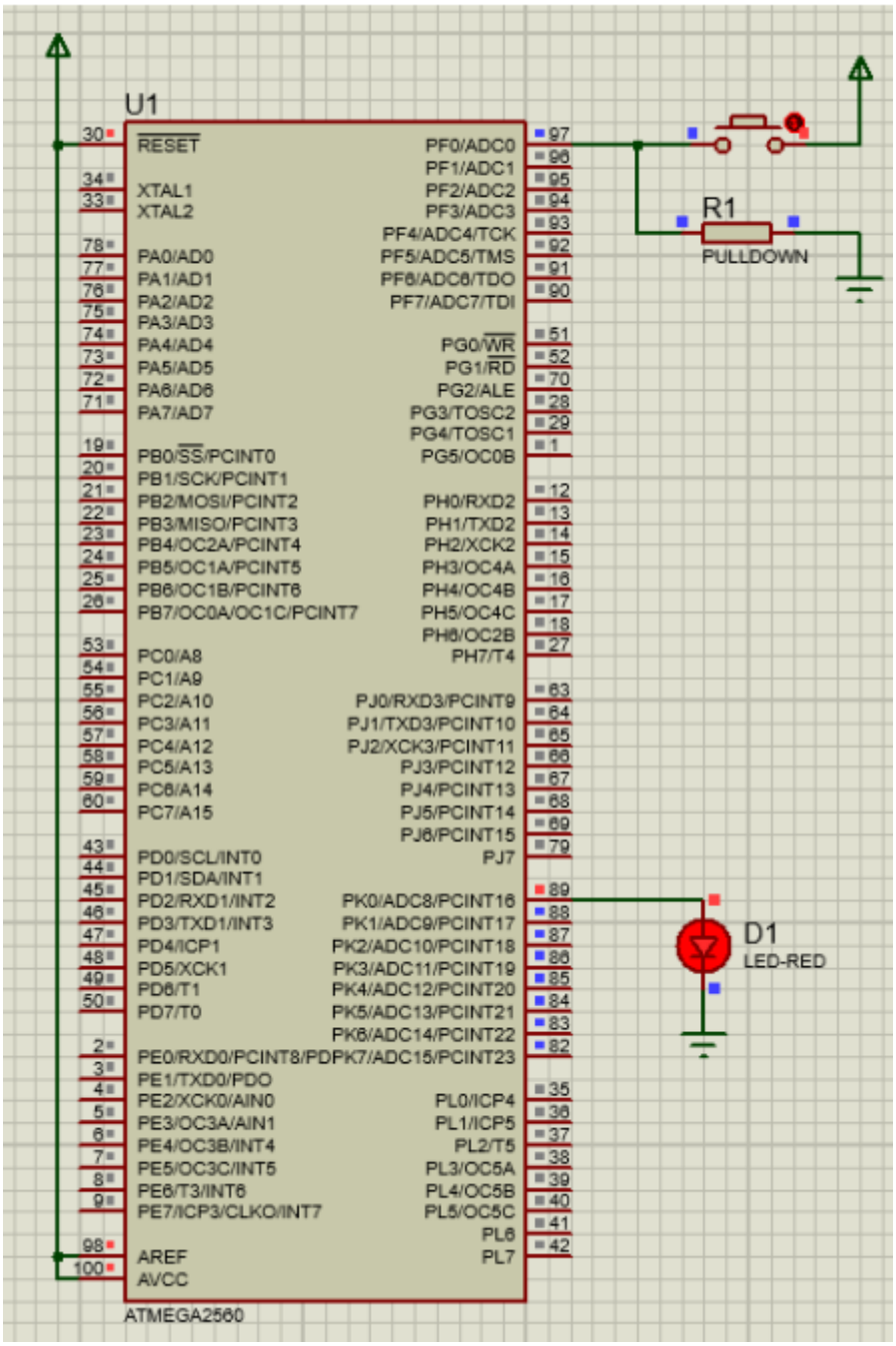
2. Counter – Multi Digit - Common Cathode 7-Segment Display:

Code:

main.c

```c
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <util/delay.h>

int main()
 {
   DDRF=0x00;
   DDRK=0xFF;
   PORTK=0x00;
   while(1){
      if(PINF & 0x01){
        _delay_ms(100);
        while(PINF & 0x01);
        PORTK ^= 0x01;
      }
   }
   return 0;
 }
```

Output:

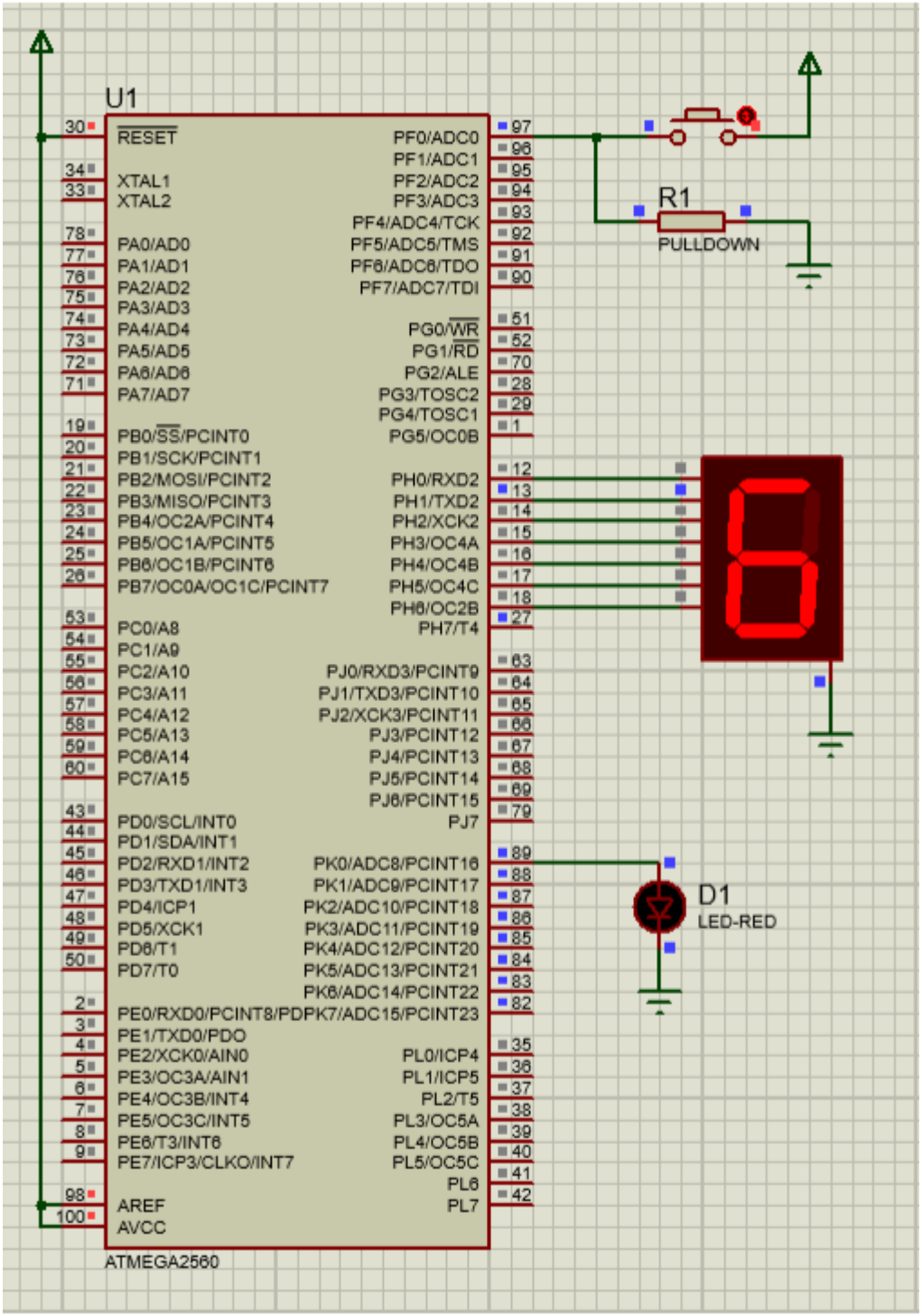

U1
RESET
XTAL1
XTAL2
PA0/AD0
PA1/AD1
PA2/AD2
PA3/AD3
PA4/AD4
PA5/AD5
PA6/AD6
PA7/AD7
PB0/SS/PCINT0
PB1/SCK/PCINT1
PB2/MOSI/PCINT2
PB3/MISO/PCINT3
PB4/OC2A/PCINT4
PB5/OC1A/PCINT5
PB6/OC1B/PCINT6
PB7/OC0A/OC1C/PCINT7
PC0/A8
PC1/A9
PC2/A10
PC3/A11
PC4/A12
PC5/A13
PC6/A14
PC7/A15
PD0/SCL/INT0
PD1/SDA/INT1
PD2/RXD1/INT2
PD3/TXD1/INT3
PD4/ICP1
PD5/XCK1
PD6/T1
PD7/T0
PE0/RXD0/PCINT8/PDO
PE1/TXD0/PDO
PE2/XCK0/AIN0
PE3/OC3A/AIN1
PE4/OC3B/INT4
PE5/OC3C/INT5
PE6/T3/INT6
PE7/ICP3/CLKO/INT7
AREF
AVCC
PF0/ADC0
PF1/ADC1
PF2/ADC2
PF3/ADC3
PF4/ADC4/TCK
PF5/ADC5/TMS
PF6/ADC6/TDO
PF7/ADC7/TDI
PG0/WR
PG1/RD
PG2/ALE
PG3/TOSC2
PG4/TOSC1
PG5/OC0B
PH0/RXD2
PH1/TXD2
PH2/XCK2
PH3/OC4A
PH4/OC4B
PH5/OC4C
PH6/OC2B
PH7/T4
PJ0/RXD3/PCINT9
PJ1/TXD3/PCINT10
PJ2/XCK3/PCINT11
PJ3/PCINT12
PJ4/PCINT13
PJ5/PCINT14
PJ6/PCINT15
PJ7
PK0/ADC8/PCINT16
PK1/ADC9/PCINT17
PK2/ADC10/PCINT18
PK3/ADC11/PCINT19
PK4/ADC12/PCINT20
PK5/ADC13/PCINT21
PK6/ADC14/PCINT22
PK7/ADC15/PCINT23
PL0/ICP4
PL1/ICP5
PL2/T5
PL3/OC5A
PL4/OC5B
PL5/OC5C
PL6
PL7
ATMEGA2560
R1
PULLDOWN
D1
LED-RED

3. **Modify the program so that pressing the button toggles the LED state (ON/OFF) instead of directly controlling it.**

Code:

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <util/delay.h>
// Function to display count on the 7-segment display
void display_count(int num, int *array) {
    PORTH = array[num];  // Send the corresponding 7-segment pattern
}
int main() {
    DDRF = 0x00;  // Set PORTF (PF0) as input for push button
    DDRK = 0xFF;  // Set PORTK as output (LED)
    DDRH = 0xFF;  // Set PORTH as output (7-segment display)
    PORTK = 0x00; // Ensure LED is OFF initiall
    // Lookup table for 7-segment display (Common Cathode)
    int array[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};
    int count = 0;  // Initialize count
    while (1) {
        if (PINF & 0x01) {  // Check if button is pressed
            _delay_ms(100);  // Debounce delay
            // Wait until button is released
            while (PINF & 0x01);
            // Toggle LED and increment count
            PORTK ^= 0x01;
            count = (count + 1) % 10;  // Increment count (reset at 10)
            display_count(count, array);  // Update 7-segment display
        }
    }
    return 0;
}
```

Output:



Conclusion:

This experiment demonstrates how the ATmega2560 can handle digital input using push buttons. Initially, the button was used to directly turn the LED ON and OFF. By modifying the logic, we implemented a toggle function, where each button press switches the LED state. This concept is

fundamental in embedded systems for implementing mode selection, power control, and user interface interactions.