LED Sequence Controller with Hardware Interrupt and Debouncing

Internship Task Report

August 13, 2025

1 Introduction

This project implements an LED sequence controller using register-level programming on an Arduino microcontroller. The design is based on code by Meet Jain, with additional hardware debouncing using a capacitor and software debouncing logic.

The system uses:

- Three LEDs connected to **PD4**, **PD5**, and **PD6** (Arduino pins 4, 5, 6).
- A push button connected to **PD2** (Arduino pin 2, INT0) to trigger LED pattern changes via hardware interrupt.
- A capacitor connected between pin 2 and GND for hardware debouncing.

2 Hardware Implementation

- **LED Connections:**
  - LED1 → PD4 (pin 4)
  - LED2 → PD5 (pin 5)
  - LED3 → PD6 (pin 6)
- **Button:** Connected to PD2 (pin 2, INT0) with internal pull-up enabled.
- **Debouncing Capacitor:** A capacitor is connected between PD2 and GND to reduce mechanical switch noise.

3 Debouncing

Debouncing is handled in two ways:

1. **Hardware:** The capacitor filters the rapid voltage changes caused by mechanical bouncing.
2. **Software:** A debounce delay of 50 ms is implemented in the interrupt service routine to ignore repeated triggers within that interval.

When the button is pressed for a very short duration, the debounce mechanism prevents any state change. This behavior is demonstrated in the accompanying video.

4 Special Features of the Solution

This implementation has several characteristics that make it stand out:

- **Register-Level Pin Control:** Instead of using high-level Arduino functions such as `digitalWrite()`, the program directly manipulates `DDRD` and `PORTD` registers. This enables faster execution (single CPU cycle) and allows setting/clearing multiple pins in one operation.
- **Hybrid Debouncing Approach:** The combination of a physical capacitor and software timing makes the system highly resistant to switch bounce noise.
- **Interrupt-Driven Input:** The push button triggers an external interrupt (INT0), ensuring instant response without continuous polling in the `loop()` function.
- **Simple State Machine Design:** The LED pattern control is implemented via a `currentState` variable, making the design scalable for more complex sequences in the future.

5 Reference

The register-level LED control uses `DDRD` and `PORTD` manipulation based on the official Arduino port manipulation documentation:

`https://docs.arduino.cc/retired/hacking/software/PortManipulation/`

6 Source Code

The complete Arduino sketch is shown below:

```
/*
 * LED Sequence Controller with Hardware Interrupt
 * Based on working code by Meet Jain
 * 3 LEDs connected to pins 4, 5, 6 (Port D)
 * Push button connected to pin 2 (INT0) with hardware interrupt
 * Register level programming for LED control
 * Hardware debouncing for button
 */

#define LED1_PIN 4   // PD4 (Port D, bit 4)
#define LED2_PIN 5   // PD5 (Port D, bit 5)
#define LED3_PIN 6   // PD6 (Port D, bit 6)

// Button pin
#define BUTTON_PIN 2  // PD2 (INT0)

// Variables for button debouncing and state
volatile unsigned long lastInterruptTime = 0;
volatile bool buttonPressed = false;
const unsigned long debounceDelay = 50; // 50ms debounce delay

// LED sequence state
volatile int currentState = 0; // 0=all off, 1=LED1, 2=LED2, 3=LED3, 4=all on

void setup() {
  Serial.begin(9600);
  // Configure LED pins as outputs and set bits 4, 5, 6 of DDRD as outputs
  DDRD |= (1 << PD4) | (1 << PD5) | (1 << PD6);
  // Initialize all LEDs to OFF
  PORTD &= ~((1 << PD4) | (1 << PD5) | (1 << PD6));
  // Configure button pin as input with internal pull-up
  // Set PD2 as input
  DDRD &= ~(1 << PD2);
  // internal pull-up for PD2
  PORTD |= (1 << PD2);
  // Configure external interrupt INT0 (pin 2)
  // Set interrupt to trigger on button press
  EICRA |= (1 << ISC01);     // Set ISC01 bit
  EICRA &= ~(1 << ISC00);    // Clear ISC00 bit (falling edge)
  // Enable INT0 interrupt
  EIMSK |= (1 << INT0);
  // Enable global interrupts
  sei();
  Serial.println("LED Sequence Controller Started");
  Serial.println("Press button to cycle through LED patterns");
  // Debug: Print initial register values
```

```
  Serial.print("DDRD: 0b");
  Serial.println(DDRD, BIN);
  Serial.print("PORTD: 0b");
  Serial.println(PORTD, BIN);
}

void loop() {
  if (buttonPressed) {
    buttonPressed = false;
    currentState++;
    if (currentState > 4) currentState = 1;
    updateLEDs();
  }
  delay(10);
}


ISR(INT0_vect) {
  unsigned long interruptTime = millis();
  if (interruptTime - lastInterruptTime > debounceDelay) {
    buttonPressed = true;
    lastInterruptTime = interruptTime;
  }
}

void updateLEDs() {
  // print state before changes
  Serial.print("Before - PORTD: 0b");
  Serial.println(PORTD, BIN);
  switch (currentState) {
    case 0: // All LEDs off
      PORTD &= ~((1 << PD4) | (1 << PD5) | (1 << PD6))
      Serial.println("All LEDs OFF");
      break;
    case 1: // Only LED1 On (Pin4)
      PORTD = (PORTD & ~((1 << PD5) | (1 << PD6))) | (1 << PD4);
      Serial.println("LED1 ON (Pin 4)");
      break;
    case 2: // Only LED2 On (Pin5)
      PORTD = (PORTD & ~((1 << PD4) | (1 << PD6))) | (1 << PD5);
      Serial.println("LED2 ON (Pin 5)");
      break;
    case 3: // Only LED3 On (Pin6)
      PORTD = (PORTD & ~((1 << PD4) | (1 << PD5))) | (1 << PD6);
      Serial.println("LED3 ON (Pin 6)");
      break;

    case 4: // All LEDs ON
      PORTD |= (1 << PD4) | (1 << PD5) | (1 << PD6);
      Serial.println("All LEDs ON");
      break;
```

```
  }
  // Print state after changes
  Serial.print("After - PORTD: 0b");
  Serial.println(PORTD, BIN);
  Serial.println("---");
}
```

7 Conclusion

The LED sequence controller works reliably with both hardware and software debouncing. Short button presses that are shorter than the debounce interval do not trigger any LED change, as demonstrated in the recorded video. The use of register-level programming and interrupt-based control makes the solution both efficient and responsive.