Ho Chi Minh City University of Technology
Faculty of Electrical & Electronics Engineering

A Member of Vietnam National University

Design of Schematic and PCB Layout for an LED Dice Game using ATmega328P

A Course Project in Embedded Systems

Members:
Nguyen Huu Dai Nhan - 2251039
Tran Duc Phong - 2251042
Nguyen Vo Thien An - 2151172

Supervisor: Bui Quoc Bao

Date: May 05, 2025

Course Code: [EE3427]

Mục lục

Abstract

This project presents the design and implementation of an LED dice game using the ATmega328P microcontroller. The system displays random numbers (1–6) through four LED groups, triggered by a push-button input with interrupt-driven control, and provides audio feedback via a buzzer. The schematic and PCB layout were designed using Altium, with firmware programmed via a USBasp programmer in the Arduino IDE. Key features include a 600ms rolling animation, random number generation seeded by analog noise, a 50ms button debounce, and a funny beep pattern (30/50/70ms). The prototype was tested for functionality, achieving reliable operation with clear visual and audio feedback. Power consumption is approximately 50–100mA at 5V. The project demonstrates skills in embedded systems, AVR programming, and PCB design, serving as an educational tool for microcontroller-based applications.

1 Introduction

1.1 Background

Electronic dice are interactive embedded systems that combine microcontroller programming with circuit design, offering a practical platform for learning. This project develops an LED dice game using the ATmega328P, integrating hardware and software to simulate a traditional dice with modern electronics.

1.2 Motivation

The project serves as an educational tool for embedded systems and engages students and hobbyists with an interactive electronic device.

2 Project Overview

2.1 Project Goals

- Implement a functional LED dice game with ATmega328P.
- Ensure reliable random number generation and user interaction.
- Design a compact PCB with USBasp programming support.

2.2 Tools Used

- Altium for schematic and PCB design.
- Proteus for simulation and verification
- Arduino IDE for firmware development.
- USBasp programmer for ATmega328P programming.

2.3 List of Components

- ATmega328P microcontroller.
- Four LED groups (GROUP_1, GROUP_2, GROUP_3, LED_4).
- Reset button (PC6, INT1).
- Active buzzer (PB0).
- 6-pin ISP header for USBasp.
- 1KΩ resistors, 100μF capacitors, 5V power supply.

3 System Design

3.1 Functional Requirements

The LED dice game must:

- Display random numbers (1–6) using four LED groups.
- Trigger rolls via a reset button with interrupt on PC6.
- Provide audio feedback (200ms beep, funny beep pattern).
- Implement a 600ms rolling animation with 200ms blinks.

Performance goals include <1s response time, reliable random numbers, and 50ms button debounce.

3.2 Block Diagram

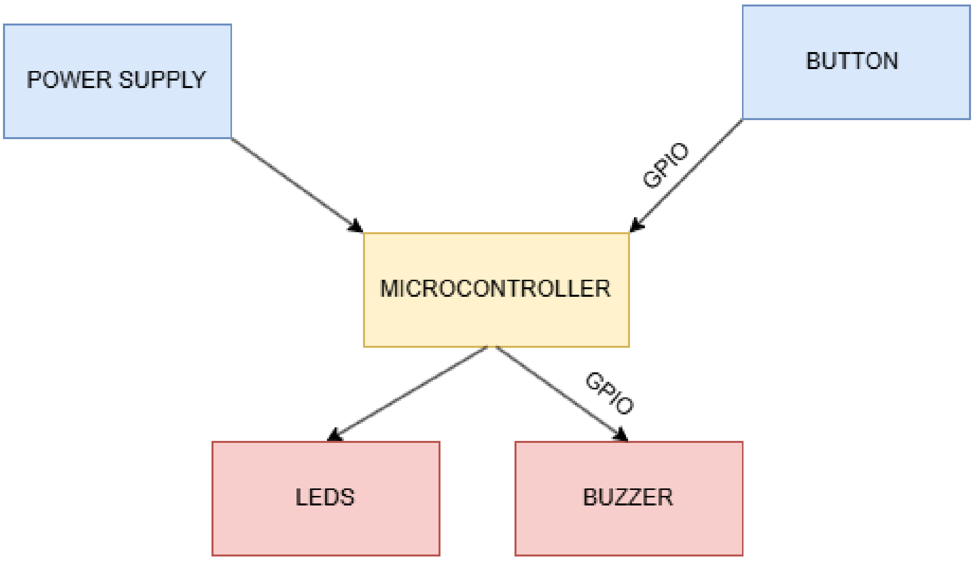

Figure 1: Block Diagram of LED Dice Game.

The system includes the ATmega328P, four LED groups (PC0-PC3), a reset button (PC6), a buzzer (PB0), a 6-pin ISP header, and a 5V power supply.

3.3 Component Selection

- ATmega328P: 8-bit AVR, 32 KB flash, 23 I/O pins, Arduino-compatible.
- LEDs: 3mm/5mm, 20mA, red/green.
- Reset Button: Momentary, PC6, external pull-up.
- Buzzer: Active, PB0.
- USBasp ISP Header: 6-pin for MOSI, MISO, SCK, RESET, VCC, GND.
- Others: 1kΩ resistors, 0.1μF capacitors, 5V power.

Component	Quantity	Part Number	Cost (vnd)
ATmega328P	1	ATMEGA328P-PU	75.000
LED (Blue)	7	WP7113SRD/D	5.000
Button	1	TS-119	20.000
Buzzer	1	PKM17EPPH4002-B0	7.000
6-pin ISP Header	1	HS0787	69.000
1kΩ Resistor	4		2.000
0.1μF Capacitor	1		2.000

Table 1: Bill of Materials (BOM).

4 Schematic Design

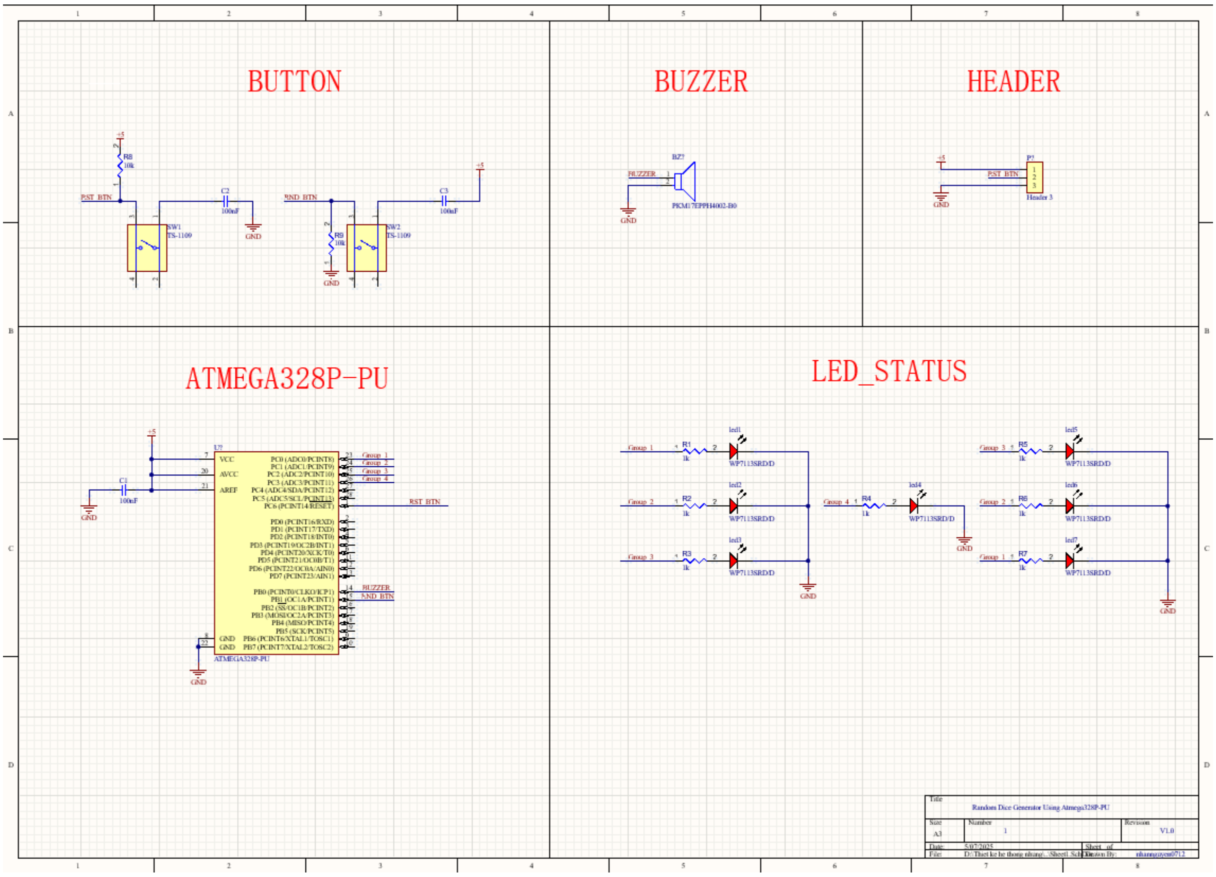

Figure 2: Schematic Diagram of LED Dice Game.

The schematic includes:

- ATmega328P: PC0-PC3 (pins 23–26) for LEDs, PC6 (pin 1) for button, PB0 (pin 14) for buzzer, SPI pins for ISP.
- LEDs: Four groups with 1kΩ resistors, active-high.
- Push Button: PC6, external pull-up, FALLING interrupt.

- Buzzer: PB0, active buzzer.
- ISP Header: 6-pin for USBasp (MOSI, MISO, SCK, RESET, VCC, GND).
- Power: 5V with 0.1μF decoupling capacitor.

4.1 Design Considerations

- LED current limited to <20mA per pin.
- Interrupt-driven button for responsiveness.
- Accessible ISP header for USBasp.
- Decoupling capacitor for power stability.

5 PCB Layout Design

The PCB was designed by importing the schematic into ALtium, defining a 10cm × 7cm board, placing components, routing traces, and generating Gerber files.

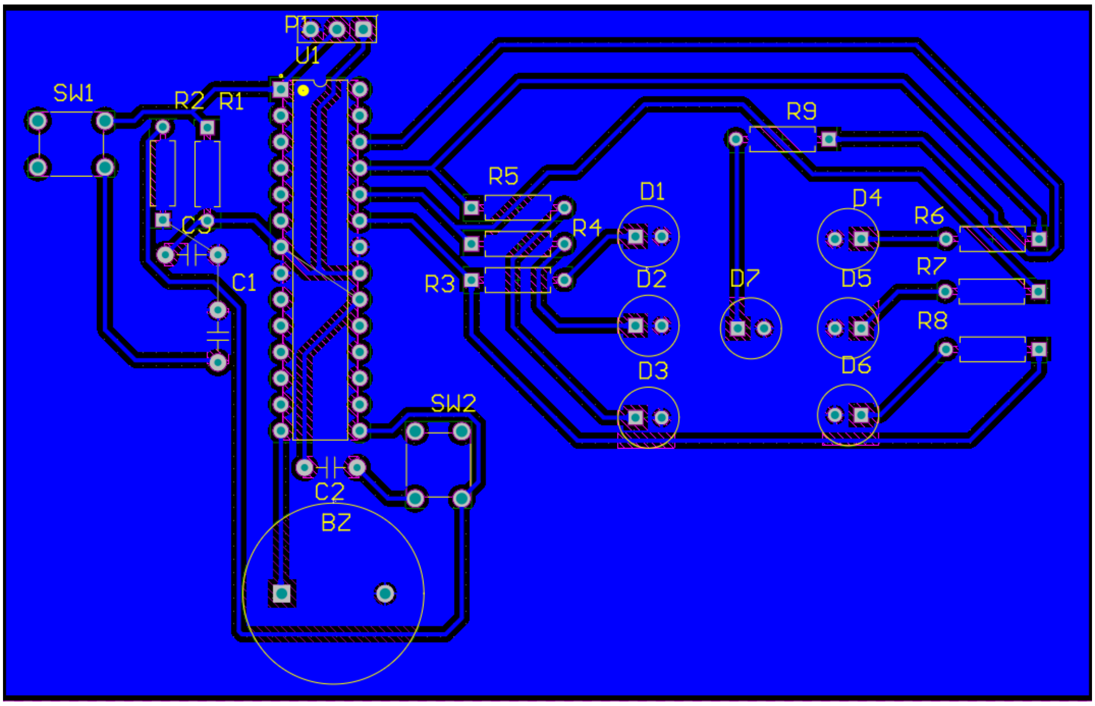

Figure 3: PCB Layout (Top and Bottom Layers).

5.1 Design Considerations

- Component Placement: ATmega328P central, LEDs in dice pattern, button accessible, buzzer near edge, ISP header near board edge.

- Routing: 10–12 mil signal traces, 20–30 mil power traces, 20 mil clearance, short LED traces, dedicated SPI for ISP.
- Power and Ground: Bottom-layer ground plane, 5V rail with decoupling.
- Manufacturability: 6 mil minimum trace width, aligned ISP header.

Tools: Altium. Challenges like ISP routing congestion were resolved by adjusting header placement.

6 Software Implementation

The firmware controls the ATmega328P for LED patterns, button interrupts, and buzzer, programmed via USBasp in Arduino IDE.

6.1 Key Algorithms

- Random Number: randomSeed(analogRead(A0)) seeds with noise, random(1, 7) for 1–6.
- LED Control: DICE_FACES array defines patterns, set_face() maps to PB0–PB3.
- Button: Interrupt on PD3, 50ms debounce in buttonISR.
- Buzzer: beep(200) for 200ms, funny_beep_pattern() for 30/50/70ms beeps.
- Rolling Effect: 600ms, 200ms blinks.

6.2 Programming Environment

Developed in Arduino IDE, with USBasp configured as the programmer. The USBasp connects to the 6-pin ISP header (MOSI, MISO, SCK, RESET, VCC, GND). The ATmega328P uses a 16 MHz clock.

7 Results and Analysis

7.1 Functional Accuracy

The LED dice game operates reliably, displaying correct LED patterns, responding to button inputs, and producing audio feedback.

7.2 Performance

- Accurate LED patterns for all faces.
- Responsive button with 50ms debounce.
- Even random number distribution.
- Clear buzzer audio at 1m.

7.3 Goals vs. Achievements

Achieved functional dice, USBasp integration, and audio feedback. ISR debounce is non-ideal for production.

7.4 Limitations

Single button, active buzzer tone restrictions, no power-saving mode.

7.5 Improvements

- Move debounce to loop().
- Support passive buzzer.
- Add sleep mode.

8 Conclusion

8.1 Summary

Successfully designed an LED dice game with ATmega328P, USBasp programming, and a compact PCB.

8.2 Future Work

- Add OLED for score tracking.
- Optimize PCB size.
- Use ATtiny for cost reduction.

9 References

- ATmega328P Datasheet, Microchip Technology.
- IPC-2221: Generic Standard on Printed Board Design.
- Arduino IDE Reference, https://www.arduino.cc/reference.
- USBasp Documentation, https://www.fischl.de/usbasp.
- Monk, S., “Programming Arduino: Getting Started with Sketches,” McGraw-Hill.

Source Code

Complete AVR code available in project repository.

```
#include <io.h>
#include <delay.h>

// Define CPU frequency (8 MHz, as per build configuration)
#define F_CPU 8000000UL

// Pin definitions for LEDs, button, and buzzer (ATmega328P registers)
#define GROUP_1_PIN 0   // PC0 (Arduino pin A0)
#define GROUP_2_PIN 1   // PC1 (Arduino pin A1)
#define GROUP_3_PIN 2   // PC2 (Arduino pin A2)
#define LED_4_PIN   3   // PC3 (Arduino pin A3)
#define RND_BTN     1   // PB1 (with external pull-up)
// Alternative: Use PC6 (requires RSTDISBL fuse)
// #define RND_BTN     6   // PC6 (with external pull-up, requires RSTDISBL fuse)
#define BUZZER      0   // PB0 (Arduino pin 8)

// Timing constants
#define ROLL_STEPS     10     // Number of steps in the rolling phase
#define BLINK_DELAY    500    // ms (time per step during rolling)
#define DISPLAY_TIME   1500   // ms (time to display final face)
#define DEBOUNCE_TIME  50     // ms

// Dice faces (bit patterns for LEDs, adjusted for correct pin mapping)
const unsigned char DICE_FACES[6] = {
    (1 << LED_4_PIN),                           // Face 1: LED_4 (PC3)
    (1 << GROUP_1_PIN),                         // Face 2: GROUP_1 (PC0)
    (1 << GROUP_1_PIN) | (1 << LED_4_PIN), // Face 3: GROUP_1 (PC0) + LED_4 (PC3)
    (1 << GROUP_1_PIN) | (1 << GROUP_3_PIN), // Face 4: GROUP_1 (PC0) + GROUP_3 (PC2)
    (1 << GROUP_1_PIN) | (1 << GROUP_3_PIN) | (1 << LED_4_PIN), // Face 5: GROUP_1 (PC0) + GROUP_3 (PC2) + LED_4 (PC3)
    (1 << GROUP_1_PIN) | (1 << GROUP_2_PIN) | (1 << GROUP_3_PIN) // Face 6: GROUP_1 (PC0) + GROUP_2 (PC1) + GROUP_3 (PC2)
};

unsigned int counter = 0; // Free-running counter for random seed
unsigned char last_face = 0; // Track the last displayed face

// Simple random number generator (using a seed)
unsigned char simple_rand(unsigned int seed) {
    // Linear congruential generator with modified parameters for better variation
    unsigned int next = (seed * 251 + 179) % 256; // Adjusted parameters for better randomness
    return (next % 6); // Return 0-5 for 6 faces
}

// Generate a random face, ensuring it's different from the last face
unsigned char get_random_face(unsigned int seed) {
    unsigned char face;
    unsigned char attempts = 0;
    do {
        face = simple_rand(seed + attempts) + 1; // Face 1-6
        attempts++;
    } while (face == last_face && attempts < 10); // Avoid repeating the last face
    return face;
}

void init_io(void) {
    // Set PC0, PC1, PC2, PC3 as outputs for LEDs
    DDRC.0 = 1;
    DDRC.1 = 1;
    DDRC.2 = 1;
    DDRC.3 = 1;

    // Set PB1 as input for button (external pull-up, so no internal pull-up needed)
    DDRB.1 = 0;
    PORTB.1 = 0; // No internal pull-up (relying on external pull-up)

    // Alternative: Use PC6 (requires RSTDISBL fuse)
    // DDRC.6 = 0;
    // PORTC.6 = 0;

    // Set PB0 as output for buzzer
    DDRB.0 = 1;

    // Initialize outputs to OFF
    PORTC.0 = 0;
    PORTC.1 = 0;
    PORTC.2 = 0;
    PORTC.3 = 0;
    PORTB.0 = 0;
}

void clear_leds(void) {
    PORTC.0 = 0;
    PORTC.1 = 0;
    PORTC.2 = 0;
    PORTC.3 = 0;
}

void set_face(unsigned char face_idx) {
    unsigned char pattern = DICE_FACES[face_idx];
    PORTC.0 = (pattern & (1 << GROUP_1_PIN)) ? 1 : 0;
    PORTC.1 = (pattern & (1 << GROUP_2_PIN)) ? 1 : 0;
    PORTC.2 = (pattern & (1 << GROUP_3_PIN)) ? 1 : 0;
    PORTC.3 = (pattern & (1 << LED_4_PIN)) ? 1 : 0;
}

void funny_beep_pattern(void) {
    unsigned char i;
    for (i = 0; i < 3; i++) {
        PORTB.0 = 1; // Buzzer ON
        delay_ms(10 * (3 + i * 2)); // 30, 50, 70 ms
        PORTB.0 = 0; // Buzzer OFF
```

```c
        delay_ms(50); // 50ms pause
    }
}

void short_beep(void) {
    PORTB.0 = 1; // Buzzer ON
    delay_ms(100); // Short 100ms beep
    PORTB.0 = 0; // Buzzer OFF
}

void rolling_effect(unsigned int seed) {
    unsigned int steps = ROLL_STEPS;
    unsigned int i;
    unsigned int local_seed = seed;
    for (i = 0; i < steps; i++) {
        unsigned char temp_face = simple_rand(local_seed);
        set_face(temp_face);
        delay_ms(BLINK_DELAY);
        clear_leds();
        local_seed = (local_seed * 251 + 179) % 256; // Update seed for next iteration
    }
}

void show_face(unsigned char face) {
    if (face < 1 || face > 6) return;
    set_face(face - 1);
    funny_beep_pattern(); // Play buzzer sound when final result is displayed
    last_face = face; // Store the last displayed face
    // Keep displaying until the next button press (handled in main loop)
}

void main(void) {
    init_io();

    while (1) {
        // Declare variables at the start of the block
        unsigned int roll_seed;
        unsigned char final_face;
        unsigned int seed_modifier = 0;

        // Increment counter continuously to provide a varying seed
        counter++;
        if (counter > 65535) counter = 0;

        // Accumulate seed_modifier while waiting for button press
        seed_modifier = (seed_modifier + counter) ^ (counter << 3); // Non-linear combination

        // Wait for button press (PB1 goes LOW due to pull-up)
        // Alternative: Use PINC.6 if using PC6 with RSTDISBL fuse
        if (!PINB.1) {
            // Debounce: Wait and confirm button state
            delay_ms(DEBOUNCE_TIME);
            if (!PINB.1) {
                // Short beep to confirm button press
                short_beep();

                // Accumulate counter increments during button press to vary the seed
                while (!PINB.1) {
                    counter++;
                    if (counter > 65535) counter = 0;
                    seed_modifier = (seed_modifier + counter) ^ (counter >> 2); // Further vary the seed
                    delay_ms(10);
                }
                roll_seed = (counter ^ seed_modifier) + (seed_modifier >> 1); // Combine for better variation
                rolling_effect(roll_seed);
                final_face = get_random_face(roll_seed + (seed_modifier << 2));
                show_face(final_face);
            }
        }
    }
}
```

Listing 1: AVR Code for LED Dice Game