MSA University

Faculty Of Engineering

Electronics and Electrical

Digital System Interface

Course Project:
Memory Game Based On Keypad & LCD

By:

Abdullah Nasser Sarhan	236985
Mohamed Salah	238547
Ahmed Mahmoud	231773

Abstract:

This project aims to implement a simple interactive memory game using the ATmega32 microcontroller, a 4x4 matrix keypad, and a 16x2 LCD. The goal is to help users improve their memory skills by repeating a sequence of numbers shown on the LCD. The game starts with an easy level and increases in difficulty as the player progresses. Key features include random number generation, score tracking, retry attempts, and game win/loss states. The system reads keypad inputs, displays sequences on the LCD, and compares user responses to evaluate correctness. The program was developed in embedded C using AVR-GCC and tested on real hardware. Results showed stable performance and accurate sequence checking, providing an engaging user experience.

1. Introduction

Memory games are widely used in educational and entertainment settings to enhance cognitive skills such as concentration and pattern recognition. This project focuses on building a standalone memory game using the ATmega32 microcontroller, making use of its GPIO capabilities to interface with both a keypad and an LCD. The game provides a hands-on demonstration of digital input/output handling, user interface design, and microcontroller programming.

2. Project Objectives

- Design and implement a memory game that challenges users to repeat sequences.
- Interface ATmega32 with a 4x4 keypad and a 16x2 LCD.
- Use embedded C for programming the logic and game mechanics.
- Include functionality for scoring, levels, and limited retries.
- Provide feedback on success or failure through the LCD.

3. System Design and Implementation

3.1 Hardware Components

- Microcontroller: ATmega32
- Display: 16x2 LCD
- Input Device: 4x4 Matrix Keypad
- Power Supply: 12V DC Battery connected to MOSFIT Regulator out 5V

3.2 Software Tools

- IDE: Atmel Studio (Microchip Studio)
- Simulator : Proteus 8
- Burner : eXtreme Burner - AVR
- AVR Programmer: USBasp

3.3 Game Flow

1. Wait for the user to press any key to start the game.
2. Display a number sequence on the LCD.
3. Prompt the user to repeat the sequence using the keypad.
4. Check if the user input matches the generated sequence.
5. Increase level and score on success; reduce lives on failure.
6. End the game if lives run out or all levels are completed.

3.4 Random Sequence Generation

Random numbers are generated using the Timer0 counter value with simple manipulation to ensure variety. Each number is converted to a character and displayed one by one.

3.5 LCD & Keypad Interface

- The LCD operates in 4-bit mode and receives commands/data using two nibbles.
- The keypad scanning method sets one column low at a time and reads the rows to detect pressed keys.

4. Results and Discussion

The game was tested for functionality and responsiveness. The LCD correctly displayed sequences, and the keypad reliably detected user inputs. The delay between levels provided a smooth transition, and the scoring mechanism was accurate. Users found the difficulty progression intuitive. A retry mechanism was added using the 'A' key to delete mistakes. The game ends with a congratulatory or game-over message based on performance.

5. Conclusion

This memory game successfully integrates hardware components with embedded software to deliver an interactive experience. It demonstrates practical skills in digital systems interfacing and C programming for microcontrollers. Future enhancements could include EEPROM-based score saving, sound output, or dynamic difficulty adjustment.

6. References

ATmega32 Datasheet. Microchip Technology Inc.

7. Appendices

7.1 Full source code

```c
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

// === LCD Definitions ===
#define LCD_C_P PORTB
#define LCD_D_P PORTA
#define LCD_C_DDR DDRB
#define LCD_D_DDR DDRA
#define EN 0
#define RW 1
#define RS 2

// === Keypad Definitions ===
#define KEYPAD_PORT PORTD
#define KEYPAD_PIN  PIND
#define KEYPAD_DDR  DDRD

// === Global Variables ===
unsigned char sequence[10];
unsigned char user_input[10];
unsigned char level, lives, score;

unsigned char KEYS[4][4] = {
     {'1','4','7','A'},
     {'2','5','8','0'},
     {'3','6','9','='},
     {'/','*','-','+'}
};

// === LCD Enable Pulse ===
void enable() {
     _delay_us(1);
     LCD_C_P |= (1 << EN);
     _delay_us(1);
     LCD_C_P &= ~(1 << EN);
     _delay_ms(2);
}

// === Send Command or Data to LCD ===
void SEND_C_D(unsigned char a, unsigned char b) {
     if (b == 0)
               LCD_C_P &= ~(1 << RS); // Command mode
     else if (b == 1)
               LCD_C_P |= (1 << RS); // Data mode

```

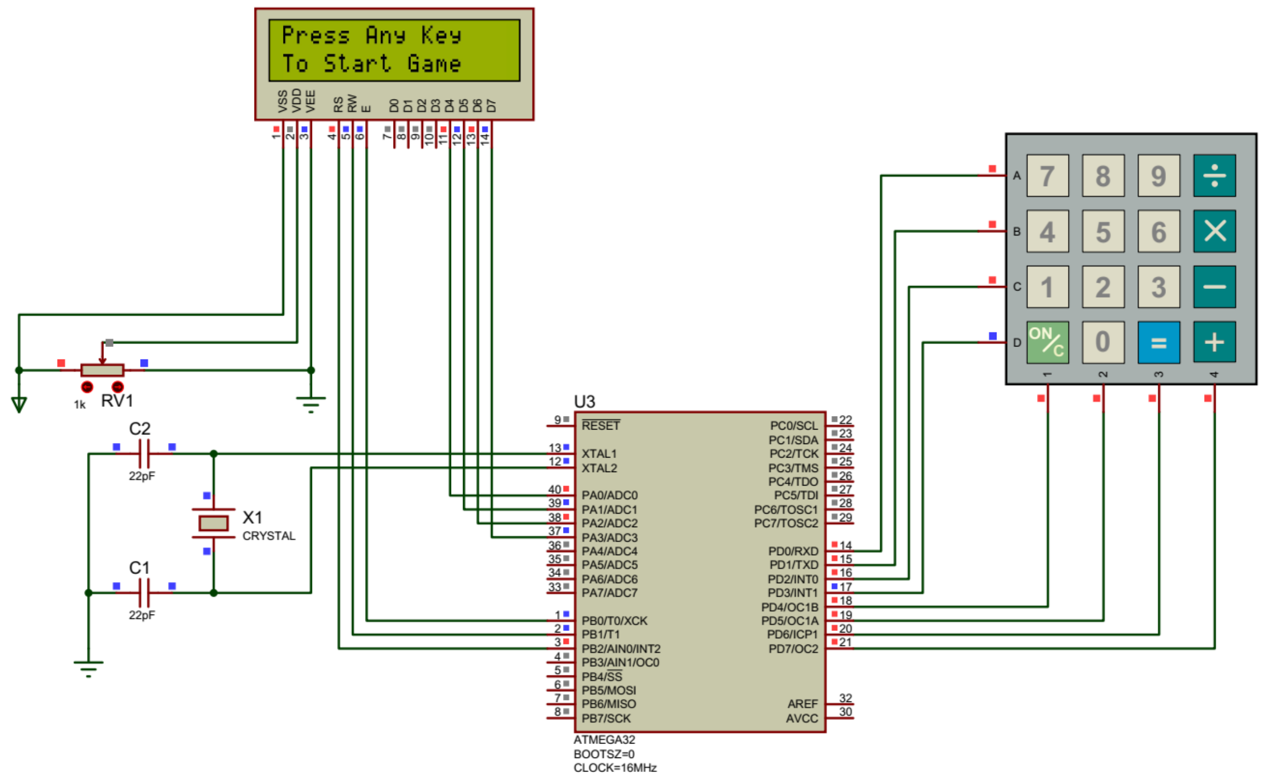
```
    LCD_C_P &= ~(1 << RW); // Write mode
    LCD_D_P &= 0xF0; // Clear upper nibble
    LCD_D_P |= (a >> 4); // Send upper nibble
    enable();

    LCD_D_P &= 0xF0; // Clear upper nibble again
    LCD_D_P |= (a & 0x0F); // Send lower nibble
    enable();
}

// === Set Cursor Position ===
void LCD_X_Y(unsigned char x, unsigned char y) {
    if (y == 0)
            SEND_C_D((0x80 + x), 0);
    if (y == 1)
            SEND_C_D((0xC0 + x), 0);
}

// === Send String to LCD ===
void SEND_D_ST(const char* s) {
    while (*s != '\0') {
            SEND_C_D(*s, 1);
            _delay_ms(5);
            s++;
    }
}

// === Initialize LCD ===
void LCD_INIT() {
    unsigned char com[] = {0x33, 0x32, 0x01, 0x28, 0x0C, 0x06, 0x80};
    for (unsigned char i = 0; i <= 6; i++) {
            SEND_C_D(com[i], 0);
            _delay_ms(20);
    }
}

// === Get Pressed Key from Keypad ===
unsigned char GET_KEY() {
    unsigned char key = 0;
    unsigned char row_data;

    // Set columns as outputs and rows as inputs with pull-ups
    KEYPAD_DDR = 0x0F;  // PD0-PD3 as outputs, PD4-PD7 as inputs
    KEYPAD_PORT = 0xFF; // Enable pull-ups on all pins

    // Scan each column
    for (unsigned char col = 0; col < 4; col++) {
            // Set current column LOW
            KEYPAD_PORT &= ~(1 << col);
            _delay_us(1);  // Keep short settling time

            // Read row data
            row_data = KEYPAD_PIN >> 4;  // Shift right to get row bits

            // Check each row
            for (unsigned char row = 0; row < 4; row++) {
                    if (!(row_data & (1 << row))) {  // If key is pressed
                            key = KEYS[row][col];
                            _delay_ms(5);  // Reduced from 15ms to 5ms

                            // Check if key is still pressed
```

```c
                if (!(KEYPAD_PIN & (1 << (row + 4)))) {
                    // Wait for key release
                    while (!(KEYPAD_PIN & (1 << (row + 4))));
                    _delay_ms(5);  // Reduced from 15ms to 5ms
                    return key;
                }
            }
        }

        // Set column back to HIGH
        KEYPAD_PORT |= (1 << col);
    }

    return 0;
}

// === Display Generated Sequence on LCD ===
void DISPLAY_SEQUENCE(unsigned char* sequence, unsigned char len) {
    for (unsigned char i = 0; i < len; i++) {
        SEND_C_D(0x01, 0);
        _delay_ms(2);
        LCD_X_Y(0, 0);
        SEND_C_D(sequence[i], 1);
        _delay_ms(300);
    }
    SEND_C_D(0x01, 0);
    _delay_ms(2);
}

// === Get User Input from Keypad ===
void GET_USER_INPUT(unsigned char* input, unsigned char len) {
    unsigned char i = 0;
    unsigned char key;

    LCD_X_Y(0, 0);
    SEND_D_ST("Repeat:");
    LCD_X_Y(0, 1);

    while (i < len) {
        key = GET_KEY();
        if (key) {
            if (key == 'A' && i > 0) {
                i--;
                LCD_X_Y(i, 1);
                SEND_C_D(' ', 1);
                LCD_X_Y(i, 1);
            } else if (key != 'A') {
                input[i] = key;
                SEND_C_D(key, 1);
                i++;
            }
        }
    }
}

// === Compare User Input with Sequence ===
unsigned char CHECK_MATCH(unsigned char* seq1, unsigned char* seq2, unsigned char len) {
    for (unsigned char i = 0; i < len; i++) {
        if (seq1[i] != seq2[i]) return 0;
    }
    return 1;
```

```c
}

// === Generate Random Sequence Using TCNT0 ===
void GENERATE_SEQUENCE(unsigned char* seq, unsigned char len) {
     for (unsigned char i = 0; i < len; i++) {
               seq[i] = '0' + (TCNT0 % 10);
               TCNT0 += 17;
     }
}

// === Game Over Display ===
void GAME_OVER() {
     SEND_C_D(0x01, 0);
     _delay_ms(2);
     LCD_X_Y(0, 0);
     SEND_D_ST("Game Over!");
     LCD_X_Y(0, 1);
     SEND_D_ST("Score:");
     SEND_C_D(score + '0', 1);
     _delay_ms(1500);
}

// === Game Win Display ===
void GAME_WIN() {
     SEND_C_D(0x01, 0);
     _delay_ms(2);
     LCD_X_Y(0, 0);
     SEND_D_ST("You Win!");
     LCD_X_Y(0, 1);
     SEND_D_ST("Score:");
     SEND_C_D(score + '0', 1);
     _delay_ms(1500);
}

// === Reset Game Variables ===
void RESET_GAME() {
     level = 1;
     lives = 3;
     score = 0;
}

// === Wait for Start Signal ===
void WAIT_FOR_START() {
     SEND_C_D(0x01, 0);
     _delay_ms(2);
     LCD_X_Y(0, 0);
     SEND_D_ST("Press Any Key");
     LCD_X_Y(0, 1);
     SEND_D_ST("To Start Game");

     while (!GET_KEY());

     SEND_C_D(0x01, 0);
     _delay_ms(2);
}

// === Main Function ===
int main(void) {
     LCD_D_DDR |= 0x0F;
     LCD_C_DDR |= 0x07;
     KEYPAD_DDR = 0x0F;
```

```c
    KEYPAD_PORT = 0xFF;

    LCD_INIT();
    TCNT0 = 47;

    while (1) {
        RESET_GAME();
        WAIT_FOR_START();

        while (lives > 0 && level <= 9) {
            SEND_C_D(0x01, 0);
            _delay_ms(2);
            LCD_X_Y(0, 0);
            SEND_D_ST("Level: ");
            SEND_C_D(level + '0', 1);
            _delay_ms(300);

            GENERATE_SEQUENCE(sequence, level);
            DISPLAY_SEQUENCE(sequence, level);
            GET_USER_INPUT(user_input, level);

            if (CHECK_MATCH(sequence, user_input, level)) {
                score++;
                level++;
            } else {
                lives--;
                if (lives > 0) {
                    LCD_X_Y(0, 0);
                    SEND_D_ST("Try Again");
                    _delay_ms(800);
                }
            }
        }

        if (lives == 0)
            GAME_OVER();
        else
            GAME_WIN();
    }
}

```

7.2 Circuit diagram


Press Any Key
To Start Game
VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
RV1
1k
C2
22pF
X1
CRYSTAL
C1
22pF
U3
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
ATMEGA32
BOOTSZ=0
CLOCK=16MHz
A
B
C
D
7
8
9
÷
4
5
6
×
1
2
3
−
ON/C
0
=
+