# Embedded_session_3

# Embedded System Session 3 - Keypad & LCD Interface

---

Created: 2025-07-15
Author: Fares Hesham Mahmoud
Tags: AVR, Embedded system, Keypad, LCD
Status: #Formatted

---

*اللهمَّ علِّمنا ما ينفعنا، وانفعنا بما علمتنا، وزدنا علمًا. وافتح علينا فتحًا عظيمًا.*

## Code Design Principles

**Clean Code Guidelines** >

Make your code:

- Clean: Easy to understand and maintain
- Configurable: Easy to modify parameters
- Readable: Self-documenting with clear names
- Editable: Modular and well-structured

## Software Architecture Layers

### Processing Call Tree

The embedded system follows a hierarchical structure:

```
Application (Main)
        ↓
     HAL (Hardware Abstraction Layer)
        ↓
     MCAL (Microcontroller Abstraction Layer)
        ↓
   Microcontroller Hardware
```

### Layer Descriptions

| Layer | Distance from Hardware | Responsibility | Examples |
|---|---|---|---|
| MCAL | Closest | Direct microcontroller control | DIO, ADC, Timer drivers |
| HAL | Middle | Hardware component abstraction | LED, Motor, Sensor drivers |
| Application | Furthest | Business logic and main program | User interface, control algorithms |
| Lib | Cross-cutting | Common utilities for all layers | Math functions, data types |

**Layer Interaction Example**

```
// Application Layer
int main(void) {
    LED_Init();         // HAL function
    LED_TurnOn();       // HAL function
}

// HAL Layer (LED Driver)
void LED_TurnOn(void) {
    DIO_SetPin(LED_PORT, LED_PIN, HIGH);  // MCAL function
}

// MCAL Layer (DIO Driver)
void DIO_SetPin(uint8_t port, uint8_t pin, uint8_t value) {
    if (value == HIGH) {
        *port_registers[port] |= (1 << pin);  // Direct hardware
access
    }
}
```

## Pull-up and Pull-down Resistors

### Internal Pull-up Configuration

ATmega32 provides internal pull-up resistors that can be enabled via software:

```
// Enable internal pull-up on input pin
DDR_REG &= ~(1 << PIN_NUM);  // Set as input (0)
PORT_REG |= (1 << PIN_NUM);   // Enable pull-up (1)
```

### External Pull-up Resistor


**Operation:**

- **Switch Open:** Pin reads **HIGH (1)**
- **Switch Pressed:** Pin reads **LOW (0)**

### External Pull-down Resistor


**Operation:**

- **Switch Open:** Pin reads **LOW (0)**
- **Switch Pressed:** Pin reads **HIGH (1)**

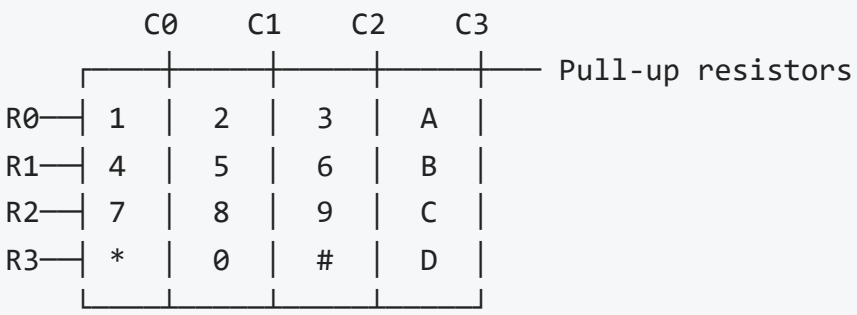
## Keypad Interface

## 4×4 Matrix Keypad Design

A 16-button keypad requires only 8 pins instead of 16 individual pins:

- 4 Row pins (outputs from microcontroller)
- 4 Column pins (inputs to microcontroller)
- All pins connected to pull-up resistors

## Keypad Connection Diagram




## Keypad Scanning Algorithm

The scanning process involves systematically testing each row:

### Keypad Scanning Sequence

#### Step 1: Test Row 0

```
// Set Row 0 to GND, others to VCC
ROW0 = 0; ROW1 = 1; ROW2 = 1; ROW3 = 1;
// Test all columns
if (COL0 == 0) button_pressed = '1';
if (COL1 == 0) button_pressed = '2';
if (COL2 == 0) button_pressed = '3';
if (COL3 == 0) button_pressed = 'A';
```

#### Step 2: Test Row 1

```
// Set Row 1 to GND, others to VCC
ROW0 = 1; ROW1 = 0; ROW2 = 1; ROW3 = 1;
// Test all columns
if (COL0 == 0) button_pressed = '4';
if (COL1 == 0) button_pressed = '5';
```

```
    if (COL2 == 0) button_pressed = '6';
    if (COL3 == 0) button_pressed = 'B';
```

Steps 3-4: Continue for remaining rows...

### Keypad Implementation

```
#define KEYPAD_ROWS 4
#define KEYPAD_COLS 4

// Keypad layout definition
const char keypad_layout[KEYPAD_ROWS][KEYPAD_COLS] = {
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}
};

char Keypad_GetKey(void) {
    for (uint8_t row = 0; row < KEYPAD_ROWS; row++) {
        // Set current row LOW, others HIGH
        Keypad_SetRowPattern(row);

        // Small delay for signal stabilization
        _delay_us(10);

        // Check all columns in current row
        for (uint8_t col = 0; col < KEYPAD_COLS; col++) {
            if (Keypad_IsColumnPressed(col)) {
                // Wait for key release (debouncing)
                while (Keypad_IsColumnPressed(col));
                return keypad_layout[row][col];
            }
        }
    }
    return 0; // No key pressed
}
```

## LCD (Liquid Crystal Display)

### LCD Specifications

Common LCD configurations:

- 2×16: 2 rows, 16 columns per row
- 4×20: 4 rows, 20 columns per row

## LCD Internal Architecture

| Component | Function | Description |
|---|---|---|
| CGROM | Character Generator ROM | Stores standard character patterns |
| CGRAM | Character Generator RAM | Stores 8 custom characters |
| DDRAM | Display Data RAM | Stores characters to be displayed |

## LCD Pin Configuration

### Power Pins

| Pin | Name | Function | Connection |
|---|---|---|---|
| 1 | VSS/GND | Ground | 0V |
| 2 | VDD/VCC | Power Supply | +5V |
| 3 | V0 | Contrast Control | Potentiometer center |
| 15 | A (Anode) | Backlight + | +5V through resistor |
| 16 | K (Cathode) | Backlight - | GND |

### Data Pins

| Pin | Name | Function |
|---|---|---|
| 7-14 | D0-D7 | 8-bit data bus |

**4-bit Mode Operation**

Most applications use 4-bit mode (D4-D7 only) to save microcontroller pins:

- 8-bit data sent as two 4-bit nibbles
- Upper nibble sent first, then lower nibble
- Requires 2 write operations per byte

### Control Pins

| Pin | Name | Function | Logic Levels |
|---|---|---|---|
| 4 | RS | Register Select | 0 = Command, 1 = Data |
| 5 | RW | Read/Write | 0 = Write, 1 = Read |
| 6 | E | Enable | Rising edge triggers operation |

## LCD Command/Data Types

### Command Mode (RS = 0)

Commands configure LCD behavior:

| Command | Hex | Function |
|---|---|---|
| Clear Display | 0x01 | Clear screen, cursor to home |
| Return Home | 0x02 | Cursor to position (0,0) |
| Entry Mode Set | 0x06 | Increment cursor after write |
| Display Control | 0x0C | Display on, cursor off |
| Function Set | 0x38 | 8-bit, 2-line, 5×8 font |

### Data Mode (RS = 1)

Actual characters to display (ASCII values).

## LCD Write Sequence

**LCD Write Timing**

```
void LCD_WriteData(uint8_t data) {
    // 1. Set data on pins
    LCD_DATA_PORT = data;

    // 2. Configure control pins
    LCD_RW = 0;     // Write operation
    LCD_RS = 1;     // Data mode

    // 3. Generate enable pulse
    LCD_E = 1;      // Enable high
    _delay_us(1);   // Hold time
    LCD_E = 0;      // Enable low (falling edge executes)
```

```
    // 4. Wait for LCD processing
    _delay_ms(2);  // Command execution time
}
```

## LCD Initialization Sequence

```
void LCD_Init(void) {
    // Wait for LCD power-up
    _delay_ms(20);

    // Function Set: 8-bit, 2-line, 5x8 font
    LCD_WriteCommand(0x38);
    _delay_ms(5);

    // Display Control: Display ON, Cursor OFF
    LCD_WriteCommand(0x0C);
    _delay_ms(1);

    // Clear Display
    LCD_WriteCommand(0x01);
    _delay_ms(2);

    // Entry Mode: Increment cursor, no shift
    LCD_WriteCommand(0x06);
    _delay_ms(1);
}
```

## Advanced LCD Functions

```
// Set cursor position
void LCD_SetCursor(uint8_t row, uint8_t col) {
    uint8_t address;
    if (row == 0)
        address = 0x80 + col;        // First row
    else
        address = 0xC0 + col;        // Second row
    LCD_WriteCommand(address);
}

// Display string
void LCD_WriteString(const char* str) {
    while (*str) {
```

```
        LCD_WriteData(*str++);
    }
}

// Display number
void LCD_WriteNumber(int32_t number) {
    char buffer[12];
    sprintf(buffer, "%ld", number);
    LCD_WriteString(buffer);
}

// Create custom character
void LCD_CreateCustomChar(uint8_t location, uint8_t pattern[]) {
    LCD_WriteCommand(0x40 + (location * 8));  // Set CGRAM address
    for (uint8_t i = 0; i < 8; i++) {
        LCD_WriteData(pattern[i]);  // Write pattern
    }
}
```

## Design Best Practices

### Avoiding Magic Numbers

**⚠ Magic Numbers Problem** >

Magic numbers are unrecognized values with unclear meaning:

Bad Example:

```
LCD_WriteCommand(0x38);  // What does 0x38 mean?
_delay_ms(47);           // Why 47 milliseconds?
```

Good Example:

```
#define LCD_FUNCTION_SET_8BIT_2LINE  0x38
#define LCD_INIT_DELAY_MS           47

LCD_WriteCommand(LCD_FUNCTION_SET_8BIT_2LINE);
_delay_ms(LCD_INIT_DELAY_MS);
```

### 2D Array Usage

For keypad layout definition:

```
// Keypad character mapping
uint8_t keypad_chars[ROWS][COLUMNS] = {
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}
};

// Access specific key
char pressed_key = keypad_chars[row_index][col_index];
```

## Practical Application Examples

### Simple Calculator Interface

```
int main(void) {
    uint32_t number1 = 0, number2 = 0, result = 0;
    char operator = 0;
    char key;

    LCD_Init();
    Keypad_Init();

    LCD_WriteString("Calculator Ready");
    LCD_SetCursor(1, 0);

    while (1) {
        key = Keypad_GetKey();
        if (key != 0) {
            if (key >= '0' && key <= '9') {
                // Number input
                LCD_WriteData(key);
                if (operator == 0) {
                    number1 = number1 * 10 + (key - '0');
                } else {
                    number2 = number2 * 10 + (key - '0');
                }
            } else if (key == '+' || key == '-' || key == '*' || key ==
'/') {
                // Operator input
                operator = key;
```

```
                LCD_WriteData(' ');
                LCD_WriteData(key);
                LCD_WriteData(' ');
            } else if (key == '#') {
                // Calculate result
                switch (operator) {
                    case '+': result = number1 + number2; break;
                    case '-': result = number1 - number2; break;
                    case '*': result = number1 * number2; break;
                    case '/': result = number1 / number2; break;
                }
                LCD_WriteString(" = ");
                LCD_WriteNumber(result);

                // Reset for next calculation
                number1 = number2 = result = 0;
                operator = 0;
            }
        }
    }
}
```

### LCD Character Set Display

```
// Display all printable ASCII characters
void LCD_ShowAllChars(void) {
    LCD_Clear();
    for (uint16_t i = 32; i <= 126; i++) {  // Printable ASCII range
        LCD_WriteData((uint8_t)i);
        _delay_ms(200);

        // Move to next line after 16 characters
        if ((i - 32 + 1) % 16 == 0) {
            LCD_SetCursor(1, 0);
        }
    }
}
```

## References


- HD44780 LCD Controller Datasheet
- Matrix Keypad Interface Techniques
- ATmega32 I/O Port Documentation

- Embedded C Programming Best Practices