HD44780 LCD Interfacing

The HD44780 is one of the most ubiquitous LCD controllers in embedded systems, found in countless 16x2 and 20x4 character displays. Understanding its programming model is essential for any embedded developer working with visual output devices.

Understanding the HD44780 Architecture

MPU Interface Modes

The HD44780 supports two primary interface modes, each with distinct advantages:

4-bit Interface Mode

- **Data Lines**: DB4 to DB7 (upper nibble only)
- **Advantage**: Saves 4 I/O pins on your microcontroller
- **Trade-off**: Requires two write cycles per byte (upper nibble first, then lower nibble)
- **Typical Use**: Most embedded applications where I/O pins are precious

8-bit Interface Mode

- **Data Lines**: DB0 to DB7 (full byte)
- **Advantage**: Faster communication (single write cycle per byte)
- **Trade-off**: Uses 8 data pins plus control pins
- **Typical Use**: Applications with abundant I/O resources

Control Signal Functions

Understanding the control signals is crucial for proper LCD operation:

RS (Register Select)

- **Purpose**: Selects target register for current operation

- **RS = 0**: Targets **Instruction Register** (command mode)
- **RS = 1**: Targets **Data Register** (character/data mode)

RW (Read/Write)

- **Purpose**: Determines data direction
- **RW = 0**: Write operation (MCU → LCD)
- **RW = 1**: Read operation (LCD → MCU)
- *Note: Many applications tie RW to ground (write-only) to save an I/O pin*

EN (Enable)

- **Purpose**: Clock signal for data latching
- **Operation**: Data is latched on the **falling edge** of EN
- **Timing**: Must maintain proper setup and hold times

Busy Flag (DB7)

- **Function**: Indicates LCD processing status during read operations
- **DB7 = 1**: LCD is busy executing the last instruction
- **DB7 = 0**: LCD is ready for next command
- **Usage**: Essential for timing-critical applications

Display Data RAM (DDRAM) Organization

The DDRAM stores the characters displayed on screen with a specific memory mapping:

- **Line 1 Address**: 0x00 to 0x0F (addresses 0-15)
- **Line 2 Address**: 0x40 to 0x4F (addresses 64-79)

Note: The non-contiguous addressing (0x00→0x40) is a key characteristic that must be handled in cursor positioning code.

Programming Sequences for 4-bit Mode

Writing Instructions (Command Mode)

Instructions configure LCD behavior and operation modes:

```
1. RS = 0          // Select Instruction Register
2. RW = 0          // Write mode
3. Send upper nibble of instruction on DB4-DB7
4. Generate falling edge on EN
5. Send lower nibble of instruction on DB4-DB7
6. Generate falling edge on EN
7. Wait for busy flag to clear
```

Writing Data/ASCII Characters

Character data is written to the current cursor position:

```
1. RS = 1          // Select Data Register
2. RW = 0          // Write mode
3. Send upper nibble of character on DB4-DB7
4. Generate falling edge on EN
5. Send lower nibble of character on DB4-DB7
6. Generate falling edge on EN
7. Wait for busy flag to clear
```

Busy Flag Checking Routine

Proper busy flag checking ensures reliable operation:

```
1. RS = 0          // Select Instruction Register
2. RW = 1          // Read mode
3. EN = 1          // Enable read
```

```
4. Read data from DB4-DB7
5. Check if DB7 = 1 (busy)
6. If busy, repeat from step 4
7. EN = 0          // Disable when ready
```

Important: Always implement timeout mechanisms to prevent infinite loops if LCD becomes unresponsive.

HD44780 Initialization and Default State

Power-On Reset Behavior

When the HD44780 is reset or powered on, it automatically configures itself with these default settings:

- **Display State**: Clear display (all characters erased)
- **Interface Mode**: 8-bit data interface
- **Display Lines**: Single line display mode
- **Character Font**: 5×8 dot matrix
- **Display Control**: Display OFF, Cursor OFF, Blinking OFF
- **Entry Mode**: Address increment, No display shift

Standard Initialization Sequence

For reliable operation, always perform proper initialization regardless of reset state:

```
1. Wait >15ms after Vcc rises to 4.5V
2. Send Function Set command three times with delays
3. Configure final operating parameters
4. Clear display
5. Set entry mode
6. Turn display on
```

Essential HD44780 Instruction Set

Understanding these core instructions is fundamental for LCD control:

Basic Control Instructions

Instruction	Code	Function	Execution Time
Clear Display	0x01	Clears entire display, cursor to home	~1.52ms
Entry Mode Set	0x06	Address increment, no shift	~37µs
Display Control	0x0C	Display ON, Cursor OFF, Blink OFF	~37µs
Function Set	0x28	4-bit mode, 2-lines, 5×8 font	~37µs

Cursor Positioning Instructions

Instruction	Code	Function	Usage
Line 1 Start	0x80	Set DDRAM address to 0x00	Beginning of first line
Line 2 Start	0xC0	Set DDRAM address to 0x40	Beginning of second line

Detailed Instruction Breakdown

Function Set (0x28)

- **Bit 5**: DL = 0 (4-bit interface)
- **Bit 4**: N = 1 (2-line display)
- **Bit 3**: F = 0 (5×8 character font)
- **Result**: Configures 4-bit, 2-line, 5×8 operation

Display Control (0x0C)

- **Bit 3**: D = 1 (Display ON)
- **Bit 2**: C = 0 (Cursor OFF)
- **Bit 1**: B = 0 (Blink OFF)
- **Result**: Visible display without cursor indicators

Entry Mode Set (0x06)

- **Bit 2**: I/D = 1 (Increment address after each character)
- **Bit 1**: S = 0 (No display shift)
- **Result**: Left-to-right text entry with stationary display

---

Practical Implementation Considerations

Timing Requirements

- **Enable Pulse Width**: Minimum 450ns high time
- **Setup Time**: Data must be stable 140ns before EN falling edge
- **Hold Time**: Data must remain stable 10ns after EN falling edge
- **Command Execution**: Most commands require 37µs, Clear Display needs 1.52ms

Common Programming Pitfalls

1. **Insufficient Delays**: Not waiting for busy flag or using inadequate delays
2. **Incorrect Nibble Order**: Sending lower nibble before upper nibble in 4-bit mode
3. **EN Signal Issues**: Not generating proper falling edges or incorrect timing
4. **Address Confusion**: Forgetting the 0x00→0x40 jump between lines

Best Practices

- **Always Initialize**: Don't rely on power-on defaults
- **Use Busy Flag**: Implement proper busy checking for reliable operation
- **Timeout Protection**: Add timeouts to prevent system lockups

- **Modular Code**: Create reusable functions for common operations
- **Error Handling**: Implement recovery mechanisms for communication failures

---

I2C-Based LCD Interface

I2C based LCD interfacing approach is widely used in embedded systems where GPIO pins are limited but LCD functionality is required.

Understanding I2C LCD Architecture

I2C to Parallel Bridge Concept

I2C-based LCD displays use an **I2C I/O expander** (typically PCF8574) to convert the 2-wire I2C interface into the parallel signals required by the HD44780 controller:

- **I2C Side**: 2 wires (SDA, SCL) + power
- **LCD Side**: 4-bit data + 3 control signals + backlight control
- **Bridge Device**: PCF8574 I/O expander providing 8-bit parallel output

System Architecture

```
MCU (STM32F407VG) ↔ I2C Bus ↔ PCF8574 I/O Expander ↔ HD44780 LCD Controller
    PB6 (SCL)                    8-bit parallel          4-bit data + control
    PB7 (SDA)                    interface               (RS, EN, D4-D7, BL)
```

Pin Mapping on PCF8574

The PCF8574 extender module is connected to LCD module are as follows:

PCF8574 Pin	LCD Signal	Function	Bit Position
P0	**RS**	Register Select	0

PCF8574 Pin	LCD Signal	Function	Bit Position
P1	**RW**	Read/Write	1
P2	**EN**	Enable	2
P3	**BL**	Backlight Control	3
P4	**D4**	Data Bit 4	4
P5	**D5**	Data Bit 5	5
P6	**D6**	Data Bit 6	6
P7	**D7**	Data Bit 7	7

STM32F407VG I2C Hardware Configuration

GPIO Configuration for I2C1

The STM32F407VG uses dedicated I2C1 peripheral on specific GPIO pins:

```
// GPIO Clock Enable
RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;

// Configure PB6 (SCL) and PB7 (SDA) as Alternate Function
GPIOB->MODER &= ~(BV(2*6+1) | BV(2*7+1));     // Clear mode bits
GPIOB->MODER |= (BV(2*6) | BV(2*7));          // Set alternate function mode

// Set Alternate Function to AF4 (I2C1)
GPIOB->AFR[0] |= (4 << (4*6)) | (4 << (4*7)); // AF4 for PB6 and PB7

// Configure as open-drain output (required for I2C)
GPIOB->OTYPER |= (BV(6) | BV(7));
```

```
// No pull-up/pull-down (external pull-ups required)
GPIOB->PUPDR &= ~(BV(2*6+1) | BV(2*7+1) | BV(2*6) | BV(2*7));
```

I2C Peripheral Configuration

```
// Enable I2C1 peripheral clock
RCC->APB1ENR |= RCC_APB1ENR_I2C1EN;

// Software reset I2C1 peripheral
I2C1->CR1 |= I2C_CR1_SWRST;
I2C1->CR1 &= ~I2C_CR1_SWRST;

// Configure peripheral clock frequency (16MHz)
I2C1->CR2 = (16 << I2C_CR2_FREQ_Pos);

// Configure clock control for 100kHz operation
I2C1->CCR = 80;  // Standard mode: CCR = Fpclk / (2 * Fi2c)

// Configure rise time
I2C1->TRISE = 17;  // Maximum rise time for 100kHz

// Enable acknowledgment and I2C peripheral
I2C1->CR1 |= I2C_CR1_ACK | I2C_CR1_PE;
```

Key Configuration Parameters:

- **FREQ**: 16MHz peripheral clock
- **CCR**: 80 for 100kHz I2C clock (16MHz / (2 × 100kHz) = 80)
- **TRISE**: 17 cycles for maximum rise time in standard mode
    - The value in the TRISE register tells the microcontroller how many APB1 clock cycles it should wait for the bus signal to rise after the output drivers are released.
    - 100 kHz Mode: I2C specs defines maximum rise time of 1000 ns.

        - TRISE=(1000/PCLK)+1
        - For 16MHz PCLK, TRISE = (1000 ns / (1/16 mhz)) + 1 = 17
    - 400 kHz Mode: I2C specs defines maximum rise time of 300 ns.
        - TRISE=(300/PCLK)+1

I2C LCD Communication Protocol

Device Addressing

The PCF8574 I/O expander used for LCD interface has specific address mapping:

- **Write Address**: `0x4E` (PCF8574 base address + write bit)
- **Read Address**: `0x4F` (PCF8574 base address + read bit)

Note: PCF8574 address depends on hardware configuration (A0, A1, A2 pins)

Data Frame Structure for LCD Control

Each I2C transaction to the LCD module sends an 8-bit value that controls all LCD interface signals simultaneously:

```
Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0
 D7   |  D6   |  D5   |  D4   |  BL   |  EN   |  RW   |  RS
```

Nibble Transmission Sequence

Since the LCD operates in 4-bit mode, each byte requires two I2C transactions:

Writing Data/Command Nibble

```
void LcdWriteNibble(uint8_t rs, uint8_t data) {
    uint8_t rsFlag = (rs == LCD_DATA) ? BV(LCD_RS_Pos) : 0;
```

```
    // First transaction: Data + control signals + EN high
    uint8_t val = (data & 0xF0) | rsFlag | BV(LCD_BL_Pos) | BV(LCD_EN_Pos);
    I2C_Write(LCD_SLAVE_ADDR_W, val);
    DelayMs(1);

    // Second transaction: Data + control signals + EN low
    val = (data & 0xF0) | rsFlag | BV(LCD_BL_Pos);
    I2C_Write(LCD_SLAVE_ADDR_W, val);
}
```

Writing Complete Byte

```
void LcdWriteByte(uint8_t rs, uint8_t data) {
    // Split data into two nibbles
    uint8_t high = (data & 0xF0), low = (data & 0x0F);

    // Send high nibble first, then low nibble
    LcdWriteNibble(rs, high);
    LcdWriteNibble(rs, low << 4);

    DelayMs(1);  // Wait for LCD processing
}
```

LCD Initialization Sequence for I2C Interface

Complete Initialization Process

The initialization sequence combines I2C setup with HD44780 initialization requirements:

```c
void LcdInit(void) {
    // Step 1: Initialize I2C peripheral
    I2C_Init();

    // Step 2: Wait for LCD power stabilization
    DelayMs(20);

    // Step 3: HD44780 standard initialization sequence
    // Send 0x03 three times (8-bit mode reset sequence)
    LcdWriteNibble(LCD_CMD, 0x03);
    DelayMs(5);
    LcdWriteNibble(LCD_CMD, 0x03);
    DelayMs(1);
    LcdWriteNibble(LCD_CMD, 0x03);
    DelayMs(1);

    // Step 4: Switch to 4-bit mode
    LcdWriteNibble(LCD_CMD, 0x02);
    DelayMs(1);

    // Step 5: Configure LCD parameters
    LcdWriteByte(LCD_CMD, LCD_FN_SET_2LINE);    // 0x28: 4-bit, 2-line, 5x8
    LcdWriteByte(LCD_CMD, LCD_DISP_OFF);        // 0x08: Display off
    LcdWriteByte(LCD_CMD, LCD_CLEAR);           // 0x01: Clear display
    LcdWriteByte(LCD_CMD, LCD_ENTRY_MODE);      // 0x06: Entry mode set
    LcdWriteByte(LCD_CMD, LCD_DISP_ON);         // 0x0C: Display on
}
```

Initialization Timing Requirements

Operation	Delay Required	Purpose
Power-on	15ms	LCD power stabilization

Operation	Delay Required	Purpose
First 0x03	4.1ms	Ensure 8-bit mode recognition
Second/Third 0x03	100us	Mode setting stabilization
4-bit switch	1ms (general)	Interface mode transition
Each command	1ms (general)	Command execution time

Advanced LCD Operations via I2C

String Display Function

```
void LcdPuts(uint8_t line, char *str) {
    // Set cursor to line start address
    LcdWriteByte(LCD_CMD, line);  // line = LCD_LINE1 or LCD_LINE2

    // Write characters one by one
    for(int i = 0; str[i] != '\0'; i++) {
        LcdWriteByte(LCD_DATA, str[i]);
    }
}
```

Backlight Control

The I2C LCD interface provides software-controlled backlight:

```
void LcdBacklight(uint8_t state) {
    uint8_t val = state ? BV(LCD_BL_Pos) : 0;
    I2C_Write(LCD_SLAVE_ADDR_W, val);
}
```

I2C Communication Layer Implementation

Basic I2C Transaction Functions

```
void I2C_Write(uint8_t addr, uint8_t data) {
    I2C_Start();                    // Generate START condition
    I2C_SendSlaveAddr(addr);        // Send slave address + write bit
    I2C_SendData(data);             // Send data byte
    I2C_Stop();                     // Generate STOP condition
}
```

I2C Status Polling

STM32 I2C peripheral requires proper status checking:

```
void I2C_Start(void) {
    I2C1->CR1 |= I2C_CR1_START;    // Send start bit
    while(!(I2C1->SR1 & I2C_SR1_SB));  // Wait for start bit sent
}

void I2C_SendSlaveAddr(uint8_t addr) {
    I2C1->DR = addr;               // Write slave address
    while(!(I2C1->SR1 & I2C_SR1_ADDR)); // Wait for address sent
    // Clear ADDR flag by reading SR1 and SR2
    (void)I2C1->SR1;
    (void)I2C1->SR2;
}

void I2C_SendData(uint8_t data) {
    while(!(I2C1->SR1 & I2C_SR1_TXE)); // Wait until TXE is set
```

```
    I2C1->DR = data;                    // Write data
    while(!(I2C1->SR1 & I2C_SR1_BTF)); // Wait for byte transfer finished
}
```

Advantages of I2C LCD Interface

Hardware Benefits

- **Pin Savings**: Reduces from 6+ pins to just 2 pins (SDA, SCL)
- **Shared Bus**: Multiple I2C devices can share the same bus
- **Long Distance**: Better noise immunity compared to parallel interface
- **Hot Pluggable**: Devices can be added/removed during operation

Software Benefits

- **Standardized Protocol**: Uses standard I2C communication functions
- **Error Detection**: Built-in acknowledgment mechanism
- **Modular Design**: Clean separation between I2C and LCD layers
- **Code Reusability**: Same I2C functions work with other I2C devices

---

Error Handling and Debugging

Common Issues and Solutions

1. **No Display Output**
    - Check I2C connections (SDA, SCL, power, ground)
    - Verify pull-up resistors (typically 4.7kΩ)
    - Confirm correct slave address
2. **Garbled Characters**

- Verify nibble transmission order (high nibble first)
- Check timing delays between operations *
- Ensure proper EN signal transitions

3. **I2C Communication Errors**

- Monitor I2C bus with logic analyzer
- Check for bus conflicts or timing violations
- Verify peripheral clock configuration