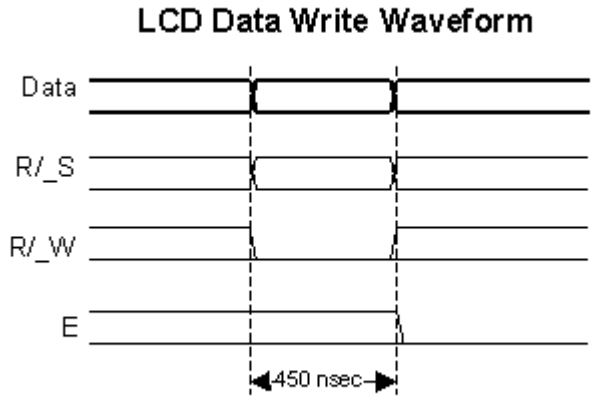
LCD Interfacing using HD44780 Hitachi chipset compatible LCD's

Most alphanumeric LCD displays have HD44780 compatible driver chipsets that follow the original Hitachi commands to control the LCD.

The most common connectors for alphanumeric LCD's are either 14-pin single row or 2X7 pins dual row connectors.

Pin	Description
1	**GND** (Ground)
2	**Vcc** (Supply Voltage)
3	**Vee** (Contrast Voltage)
4	**R/S** (Instruction/Register Select)
5	**R/W** (Read/Write)
6	**E** (Clock)
7	**D0** (Data0)
8	**D1** (Data1)
9	**D2** (Data2)
10	**D3** (Data3)
11	**D4** (Data4)
12	**D5** (Data5)
13	**D6** (Data6)
14	**D7** (Data7)

A typical LCD write operation takes place as shown in the following timing waveform:



The interface is either a 4-bit or 8-bit parallel bus that allows fast reading/writing of data to and from the LCD.

This waveform will write an ASCII Byte out to the LCD's screen. The ASCII code to be displayed is eight bits long and is sent to the LCD either four or eight bits at a time. If 4-bit mode is used, two nibbles of data (First high four bits and then low four bits with an E Clock pulse with each nibble) are sent to complete a full eight-bit transfer. The E Clock is used to initiate the data transfer within the LCD.

8-bit mode is best used when speed is required in an application and at least ten I/O pins are available. 4-bit mode requires a minimum of six bits. In 4-bit mode, only the top 4 data bits (DB4-7) are used.

The R/S pin is used to select whether data or an instruction is being transferred between the microcontroller and the LCD. If the pin is high, then the byte at the current LCD Cursor Position can be read or written. If the pin is low, either an instruction is being sent to the LCD or the execution status of the last instruction is read back (whether or not it has completed).

The HD44780 instruction set is shown below:

R/S	**R/W**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	**Instruction/Description**
4	5	14	13	12	11	10	9	8	7	Pins
0	0	0	0	0	0	0	0	0	1	Clear Display
0	0	0	0	0	0	0	0	1	*	Return Cursor and LCD to Home Position
0	0	0	0	0	0	0	1	ID	S	Set Cursor Move Direction
0	0	0	0	0	0	1	D	C	B	Enable Display/Cursor
0	0	0	0	0	1	SC	RL	*	*	Move Cursor/Shift Display
0	0	0	0	1	DL	N	F	*	*	Set Interface Length
0	0	0	1	A	A	A	A	A	A	Move Cursor into CGRAM
0	0	1	A	A	A	A	A	A	A	Move Cursor to Display
0	1	BF	*	*	*	*	*	*	*	Poll the "Busy Flag"
1	0	D	D	D	D	D	D	D	D	Write a Character to the Display at the Current Cursor Position
1	1	D	D	D	D	D	D	D	D	Read the Character on the Display at the Current Cursor Position

The bit descriptions for the different commands are:

"*" - Not Used/Ignored. This bit can be either "1" or "0"

Set Cursor Move Direction:
ID - Increment the Cursor after Each Byte Written to Display if Set
S - Shift Display when Byte Written to Display

Enable Display/Cursor
D - Turn Display On(1)/Off(0)
C - Turn Cursor On(1)/Off(0)
B - Cursor Blink On(1)/Off(0)

Move Cursor/Shift Display
SC - Display Shift On(1)/Off(0)
RL - Direction of Shift Right(1)/Left(0)

Set Interface Length
DL - Set Data Interface Length 8(1)/4(0)
N - Number of Display Lines 1(0)/2(1)
F - Character Font 5x10(1)/5x7(0)

Poll the "Busy Flag"
BF - This bit is set while the LCD is processing

Move Cursor to CGRAM/Display
A - Address

Read/Write ASCII to the Display
D - Data

Reading Data back is best used in applications which required data to be moved back and forth on the LCD (such as in applications which scroll data between lines). The "Busy Flag" can be polled to determine when the last instruction that has been sent has completed processing.

For most applications, there really is no reason to read from the LCD. The application can write to the LCD and wait the maximum amount of time for each instruction (4.1 milliseconds for clearing the display or moving the cursor/display to the "home position", 160 microseconds for all other commands). Different LCD's execute instructions at different rates and to avoid problems later on (such as if the LCD is changed to a slower unit), the recommended maximum delays can be used.

Before you can send commands or data to the LCD, the LCD must be initialized. For 8-bit mode, this is done as follows:

1. Wait more than 15 msecs after power is applied.
2. Write 0x030 to LCD and wait 5 msecs for the instruction to complete
3. Write 0x030 to LCD and wait 160 usecs for instruction to complete
4. Write 0x030 AGAIN to LCD and wait 160 usecs or Poll the Busy Flag
5. Set the Operating Characteristics of the LCD
    - Write "Set Interface Length"
    - Write 0x010 to turn off the Display
    - Write 0x001 to Clear the Display
    - Write "Set Cursor Move Direction" Setting Cursor Behavior Bits
    - Write "Enable Display/Cursor" & enable Display and Optional Cursor

In 4-bit mode, the high nibble is sent before the low nibble and the E pin is toggled each time four bits is sent to the LCD. To initialize in 4-bit mode:

1. Wait more than 15 msecs after power is applied.
2. Write 0x03 to LCD and wait 5 msecs for the instruction to complete
3. Write 0x03 to LCD and wait 160 usecs for instruction to complete
4. Write 0x03 AGAIN to LCD and wait 160 usecs (or poll the Busy Flag)
5. Set the Operating Characteristics of the LCD
    - Write 0x02 to the LCD to Enable 4-Bit Mode

      All following Instruction/Data Writes require two nibble writes.
    - Write "Set Interface Length"
    - Write 0x01/0x00 to turn off the Display
    - Write 0x00/0x01 to Clear the Display
    - Write "Set Cursor Move Direction" Setting Cursor Behavior Bits
    - Write "Enable Display/Cursor" & enable Display and Optional Cursor

Code Example for 4-bit mode LCD initialization and displaying a message:

```
main()
{
	unsigned char i;

	/* Initialize the LCD as the very first thing */
	InitLCD();

	/* Write a simple message to the LCD */
	WriteLCD( "Hello" );
}


void InitLCD()
{
	/* Wait a bit after power-up */
	delay(200);

	/* Initialize the LCD to 4-bit mode */
	WriteCtrl(3);
	delay(50);

	WriteCtrl(3);
	delay(10);

	WriteCtrl(3);
	delay(10);

	WriteCtrl(2);
	delay(10);

	/* Function Set */
	WriteCtrl(2);
	delay(10);

	WriteCtrl(8);
	delay(10);

	/* Display OFF */
	WriteCtrl(0);
	delay(10);

	WriteCtrl(8);
	delay(10);

	/* Display ON */
	WriteCtrl(0);
	delay(10);

	WriteCtrl(0x0F);
	delay(10);

	/* Entry mode */
	WriteCtrl(0);
	delay(10);

	WriteCtrl(6);
	delay(10);

	/* Clear Screen */
	WriteCtrl(0);
	delay(10);
```

```
	WriteCtrl(1);
	delay(100);

	/* Cursor home */
	WriteCtrl(0);
	delay(10);

	WriteCtrl(2);
	delay(100);
}


void WriteLCD( unsigned char* message )
{
	unsigned char i;

	for( i=0; i<20; i++ )
	{
		if( !message[i] )
			break;

		WriteData(message[i]);
	}
}

void WriteCtrl( unsigned char value )
{
	clrbit(CTRL);

	WriteCommon( value );
}

void WriteData( unsigned char value )
{
	setbit(CTRL);

	WriteCommon( value >> 4 );
	WriteCommon( value );
}

void WriteCommon( unsigned char value )
{
	clrbit(READ);

	value = value & 0x0F;
	value = value << 4;

	P0 = P0 & 0x0F;
	P0 = P0 | value;

	setbit(STROBE);
	delay(1);
	clrbit(STROBE);
	setbit(READ);

	delay(1);
}
```