Chapter 9

Input/Output (I/O) Ports and Interfacing

Updated: 3/13/12
I/O devices (or peripherals) such as LEDs and keyboards are essential components of the microprocessor-based or microcontroller-based systems.

- Classified into two groups
  - input devices
  - output devices
Block Diagram of I/O Interfacing

Access one port at a time
8-bit registers
I/O ports are associated with a SFR
Each port is associated with 3 registers:
PORT / LAT / TRIS
I/O Ports: Interfacing and Addressing

- To read (receive) binary data from an input peripheral
  - MPU places the address of an input port on the address bus, enables the input port by asserting the RD signal, and reads data using the data bus.

- To write (send) binary data to an output peripheral
  - MPU places the address of an output port on the address bus, places data on data bus, and asserts the WR signal to enable the output port.

- Remember:
  - Writing to the port
    - When the MPU sends out or transfers data to an output port
  - Reading from the port
    - When the MPU receives data from an input port
PIC18F452/4520 I/O Ports (1 of 5)

- MCU includes five I/O ports
  - PORTA, PORTB, PORTC, PORTD, and PORTE
- Ports are multiplexed meaning they can be set up by writing instructions to perform various functions

<table>
<thead>
<tr>
<th>RA6</th>
<th>AN4/SS/LVDIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA5</td>
<td>TOCK1</td>
</tr>
<tr>
<td>RA4</td>
<td>AN3/REF+</td>
</tr>
<tr>
<td>RA3</td>
<td>AN2/REF-</td>
</tr>
<tr>
<td>RA2</td>
<td>AN1</td>
</tr>
<tr>
<td>RA1</td>
<td>AN0</td>
</tr>
</tbody>
</table>

PORTA: Example of Multiple Functions
- Digital I/O: RA6-RA0
- Analog Input: AN0-AN4
- \( V_{REF+} \): A/D Reference Plus Voltage
- \( V_{REF-} \): A/D Reference Minus Voltage
- TOCK1: Timer0 Ext. Clock
- SS: SPI Slave Select Input
- LVDIN: Low voltage Detect Input
Each I/O port is associated with the special functions registers (SFRs) to setup various functions.

- Can be set up as entire ports or each pin can be set up.
  - **PORT**: This register functions as a latch or a buffer determined by the logic levels written into the associated TRIS register.
  - **TRIS**: This is a data direction register. Writing logic 0 to a pin sets up the pin as an output pin, and logic 1 sets up the pin as an input pin.
  - **LAT**: This determines if port is bidirectional.
Internal block diagram of PORTB includes:
- Three internal D flip-flops (latches)
  - Data latch to output data
  - TRIS latch to setup data direction
  - Input latch for input data
PORTB Internal Block Diagram

Three internal D flip-flops (latches):

- **Data latch** to output data
- **TRIS latch** to setup data direction
- **Input latch** for input data

Notation:
- Q- TRIS: 0 → A is enabled
- Q- TRIS: 1 → A is disabled
Interfacing Output Peripherals (1 of 2)

- Commonly used output peripherals in embedded systems are:
  - LEDs, seven-segment LEDs, and LCDs; the simplest is LED
- Two ways of connecting LEDs to I/O ports:
  - LED cathodes are grounded and logic 1 from the I/O port turns on the LEDs - The current is supplied by the I/O port called **current sourcing**.
  - LED anodes are connected to the power supply and logic 0 from the I/O port turns on the LEDs - The current is received by the chip called **current sinking**.

![Common Cathode](image)

![Common Anode](image)

**Active high**

**Active low**
Interfacing Seven-Segment LEDs as an Output (1 of 4)

- Seven-segment LEDs
  - Often used to display BCD numbers (1 through 9) and a few alphabets
  - A group of eight LEDs physically mounted in the shape of the number eight plus a decimal point as shown in Figure 9-5 (a)
  - Each LED is called a segment and labeled as ‘a’ through ‘g’.
Interfacing Seven-Segment LEDs as an Output (2 of 4)

- Two types of seven-segment LEDs
  - Common anode
  - Common cathode

![Diagram of seven-segment LEDs with segments labeled a, b, c, d, e, f, g, and decimal point (dp)].

decimal point
In a common anode seven-segment LED:
- All anodes are connected together to a power supply and cathodes are connected to data lines.
- Logic 0 turns on a segment.
- Example: To display digit 1, all segments except b and c should be off.
- Byte 11111001 = F9H will display digit 1.
Interfacing Seven-Segment LEDs as an Output (4 of 4)

- In a common cathode seven-segment LED:
  - All cathodes are connected together to ground and the anodes are connected to data lines.

- Logic 1 turns on a segment.

- Example: To display digit 1, all segments except b and c should be off.

- Byte 00000110 = 06H will display digit 1.
Segment LEDs to PORTB and PORTC

![Diagram of segment LEDs connected to PORTB and PORTC.]
Seven-Segment Chips

ALPHA/NUMERIC C/A DISPLAY
Sample Program

Title "Ex9-4 BCD Digit Display"
List p=18F452; f=inhx32
#include <p18F452.inc>

REG0 EQU 0x00
NUMBER EQU 0x9
ORL 0
GOTO 0x20
ORG 0x20

START: CLRF PORTB ;Initial reading of display
CLRF TRISB ;Set up PORTB
MOVWF NUMBER ;BCD
MOVWF REG0, 0
MOVWF UPPER CODEADDR ;Copy upper
MOVWF TBLPTRU ;21-bit
MOVWF TBLPTRH
MOVWF LOW CODEADDR ;Copy low
MOVWF TBLPTRL
MOVF REG0, 0, 0
ADDWF TBLPTRL,1,0
TBLRD*
MOVFF TABLAT, PORTB ;Turn on
ORG 0x40 ;Store LED code starting at location U040H

CODEADDR:
DB 0xc0, 0xf9, 0xa4, 0xb0, 0x99 ;LED code for digits 0 to 4
DB 0x92, 0x82, 0xf8, 0x80, 0x98 ;LED code for digits 5 to 9
Interfacing to Multiple 7-Segments
Using the Simulator
Commonly used input peripherals in embedded systems are:

- DIP switches, push-button keys, keyboards, and A/D converters.

DIP switch: One side of the switch is tied high (to a power supply through a resistor called a pull-up resistor), and the other side is grounded. The logic level changes when the position is switched.

Push-button key: The connection is the same as in the DIP switch except that contact is momentary.
Interfacing Dip Switches and Interfacing LEDs
Driving an LED

- Transistor inverter acting as the driver
- Assume LED requires 3.5 Volts and 20 mA
- 2N2222 is Current Amplifier $\rightarrow$ generates collector current (multiplies base-current)
- In this case the current gain is 100!
- Note Vbe=0.7
Driving a RELAY & SOLENOID

- Controlling appliances
- Driving solenoid

![Circuit Diagram]

- LS1 RELAY
- DS1
- 25 Watt Lamp
- RB0
- R1 5.6K
- Q1 2N2222A
- D1
- +5V
- +12V
- Q1 MPSA13
- D1
Example: Reading from an I/O Port

- The instruction: MOVF PORTB, W reads from PORTB.
- To execute the instruction, the MPU does the following:
  - Reads the instruction from memory
  - Places the address of PORTB (F81H) on the address bus of data memory
  - Selects PORTB
  - Asserts the RD signal and enables PORTB
  - Reads logic levels (1/0) of the switches and places on the data bus
  - Saves the reading in the WREG
The pull-up resistors are connected externally. However, PORTB can provide equivalent resistors internally through initialization.

Turning off the internal FET is equivalent to providing a pull-up resistor.
Internal Pull-Up Resistor (2 of 2)

- Bit7 (RBPU) in the INTCON2 register enables or disables the pull-up resistor
  - Instruction to Enable Pull Up Resistors:
    BCF INTCON2 7, 0

C Code: INTCON2bit.RBPU = 0 // pull-ups on

<table>
<thead>
<tr>
<th>B7</th>
<th>B6</th>
<th>B5</th>
<th>B4</th>
<th>B4</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBPU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RBPU = PORTB pull-up resistor enable bit
0 = Pull-up resistors are enabled
1 = Pull-up resistors are disabled
Electrical connection of a push-button key is same as that of a DIP switch except that the connection is temporary when the key is pressed.

- When a key is pressed (or released), mechanical metal contact bounces and can be read as multiple inputs.
- The reading of one contact as multiple inputs can be eliminated by a key-debounce technique, using either hardware or software.
Interfacing Push-Button Keys (2 of 2)

(a)

(b)

Logic 1

:Key Push

20ms

Logic 0

:Key Release

Logic 1
Various Switches
Key Debounce Techniques

- **Hardware technique**
  - Two circuits, based on the principles of generating a delay and switching the logic level at a certain threshold level.
  - Two NAND gates connected back to back, equivalent of a S-R latch. The output of the S-R latch is a pulse without a bounce.
  - An integrated circuit (MAX 6816) that bounces the key internally and provides a steady output.
Problem statement

- A bank of push-button keys are connected as inputs to PORTB.
- The pull-up resistors are internal to PORTB.
- Write a program to recognize a key pressed, debounce the key, and identify its location in the key bank with numbers from 0 to 7.
Interfacing Push-Button Keys (3 of 6)

- **Hardware**
  - PORTB should be set up as input port
  - Internal pull-up resistors should be enabled

- **Software**
  - Checking a key closure Debouncing the key
  - Encoding the key

Alternatively
Interfacing Push-Button Keys - Software DeBounding

- Checking a key closure
  - When a key is open, the logic level is one (assuming pull-ups are enabled) and when it is closed, the logic level is zero.
  - When all keys are open, the reading will be 0xFF, and when a key is closed, the reading will be less than 0xFF.
    - Therefore, any reading less than FFH indicates a key closure.
    - This will be the first read!

- Debouncing the key
  - Software technique
    - Wait for 20 ms.
    - Read the port again.
    - If the reading is still less than FFH, it indicates that a key is pressed.

- Encoding the key
  - Key closure can be identified by rotating the reading right and looking for ‘No Carry’ and counting the rotations
Software Debouncing –
Used for Active LOW!

// >>> Don’t forget the #include <delays.h> statement <<<
// ************************ Switch ******************************
// to use this function, make sure that it is invoked as follows

Switch( 0x04 ) ← switch on bit 2
or
Switch( 0x40 ) ← switch on bit 6
or
Switch( 0x03 ) ← switches on bits 0 and 1

// ********************** CONSTATNS ****************************
#define KEYPORT PORTA // change to match the actual port
#define DELAY 15 // change as needed for time delay – 15 msec.

void Switch( char bit )
{
    do // wait for release
    {
        while ( ( KEYPORT & bit ) != bit );
        Delay1KTCYx(DELAY);
    }
    while( ( KEYPORT & bit ) != bit );

    do // wait for press
    {
        while ( ( KEYPORT & bit ) == bit );
        Delay1KTCYx(DELAY);
    }
    while( ( KEYPORT & bit ) == bit );
}
Software Debouncing – Used for Active LOW!

Another Example

```c
void main (void) {
    unsigned char Switch_Count = 0;
    LED_Display = 1; // initialize
    TRISD = 0b00000000; // PORTD bits 7:0 are all outputs (0)
    INTCON2bits.RBPU = 0; // enable PORTB internal pullups
    WPUHbits.WPUB0 = 1; // enable pull up on RB0
    ANSELH = 0x00; // AN0-12 are digital inputs (AN12 on RB0)
    TRISBbits.TRISB0 = 1; // PORTB bit 0 (connected to switch) is input (1)
    
    while (1) {
        LATD = LED_Display; // output LED_Display value to PORTD LEDs
        LED_Display <<= 1; // rotate display by 1
        if (LED_Display == 0)
            LED_Display = 1; // rotated bit out, so set bit 0
        
        while (Switch_Pin != 1); // wait for switch to be released
        Switch_Count = 5;
        do // monitor switch input for 5 lows in a row to debounce
            if (Switch_Pin == 0)
                Switch_Count++;
            else
                Switch_Count = 0;
        while (Switch_Count < DetectsInARow);
    }
}
```

#define Switch_Pin PORTBbits.RB0

The demo board switch is connected to I/O pin RB0, which is normally pulled up to VDD internally. **When the switch is pressed, it pulls RB0 to ground** (low state).
PIC18F46K20 Pin Diagram
Digital Input Port

Pins are configured as analog or digital in the SFRs ANSEL and ANSELH.
Class Exercise

- Find the following bits in the Data Sheet:

```c
INTCON2bits.RBPU = 0;  // enable PORTB internal pullups
WPUBbits.WPUB0 = 1;   // enable pull up on RB0
ANSELH = 0x00;         // AN8-12 are digital inputs (AN12 on RB0)
TRISBbits.TRISB0 = 1;  // PORTB bit 0| (connected to switch) is input (1)
```
Interfacing a Matrix Keyboard

Actual Keyboard

Keyboard as seen by software before the lookup table

Telephone-style 4x3 Keyboard
Interfacing a Matrix Keyboard

Software

To recognize and encode the key pressed, the program should:

- Ground all the columns by sending zeros.
- Check each key in a row for logic zero.
- Ground one column at a time and check all the rows in that column.
- Once a key is identified, it is encoded based on its position in the column.
Matrix Keyboard Software

```c
// key codes for a telephone style keypad
// stored as static constants in the program memory

rom near char lookupKey[] =
{
    1, 4, 7, 10, // left column
    2, 5, 8, 0,  // middle column
    3, 6, 9, 11 // right column
};

// uses function Switch
// unsigned char Key(void)
{
#define MASK 0x0f
#define ROWS 4

char a;
char keyCode;
PORTB = keyCode = 0; // clear Port B & keyCode
Switch( MASK ); // de-bounce and wait for any key
PORTB = 0xFE; // select a leftmost column

while ( ( PORTA & MASK ) == MASK ) // while no key is found
{
    PORTB = (PORTB << 1) | 1; // get next column
    keyCode += ROWS; // add rows to keycode
}

for ( a = 1; a != 0; a <<= 1)
{
    if ( ( PORTA & a ) == 0 )
    {
        break;
    }
    keyCode++; // find row
}

return lookupKey[keyCode]; // lookup correct key code
```
7-Segment Control Software with De-bounce ➔ Each time the input is Pressed the number Shown by the 7-segment Increments!

```c
// ******************* program memory data *******************************
rom near char look7[] = // 7-segment lookup table
{
    0x40, // 0  active low signals
    0x79, // 1  x g f e d c b a
    0x24, // 2
    0x30, // 3
    0x19, // 4
    0x12, // 5
    0x02, // 6
    0x78, // 7
    0x00, // 8
    0x10  // 9
};

// ******************* data memory data ****************************
int count;

#pragma code

// ******************* de-bounce functions ****************************
void Switch( char bit )
{
    do  // wait for release
        
        while ( ( PORTA & bit ) != bit );
        Delay1KTCYx(30); // 15 ms delay
    }

    while( ( PORTA & bit ) != bit );
    do  // wait for press
        
        while ( ( PORTA & bit ) == bit );
        Delay1KTCYx(30);
    }

    while( ( PORTA & bit ) == bit );

// ******************* main program ******************************
void main (void)
{
    ADCON1 = 0x7F;   // Ports A and B are digital
    TRISA = 1;       // Port A, bit 0 is input
    TRISB = 0;       // Port B is output
    count = 0;       // start count at zero
    while ( 1 )      // main loop
        
        {
            PORTB = look7[count];  // display number
            Switch( 1 );            // wait for pushbutton
            count++;                
            if ( count >= 10 )     // count = 0;
                count = 0;
        }
}
```
Interfacing LCD (Liquid Crystal Display)

- **Problem statement**
  - Interface a 2-line x 20 character LCD module with the built-in [HD44780](https://en.wikipedia.org/wiki/HD44780) controller to I/O ports of the PIC18 microcontroller

- Multi-LCDs refer to LCDs with different interfaces
Converting to ASCII

- The LCD can represent characters in ASCII
- For example number 0x08 → must be converted to 0x38
- To perform this:
  - If \( W=0x08 \) then ASCII=\( \text{XORLW } 0x30 \) \( \rightarrow W=38 \)

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>0</td>
<td>NULL (null)</td>
<td>32</td>
<td>20</td>
<td>0x40</td>
<td>0x40</td>
<td>0x5e</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>1</td>
<td>SOH (start of heading)</td>
<td>35</td>
<td>21</td>
<td>0x41</td>
<td>0x41</td>
<td>0x5f</td>
</tr>
<tr>
<td>2</td>
<td>002</td>
<td>2</td>
<td>STX (start of text)</td>
<td>34</td>
<td>22</td>
<td>0x42</td>
<td>0x42</td>
<td>0x5f</td>
</tr>
<tr>
<td>3</td>
<td>003</td>
<td>3</td>
<td>ETX (end of text)</td>
<td>35</td>
<td>23</td>
<td>0x43</td>
<td>0x43</td>
<td>0x5f</td>
</tr>
<tr>
<td>4</td>
<td>004</td>
<td>4</td>
<td>EOT (end of transmission)</td>
<td>36</td>
<td>24</td>
<td>0x44</td>
<td>0x44</td>
<td>0x5f</td>
</tr>
<tr>
<td>5</td>
<td>005</td>
<td>5</td>
<td>ENQ (enquiry)</td>
<td>37</td>
<td>25</td>
<td>0x45</td>
<td>0x45</td>
<td>0x5f</td>
</tr>
<tr>
<td>6</td>
<td>006</td>
<td>A</td>
<td>ACK (acknowledge)</td>
<td>38</td>
<td>26</td>
<td>0x46</td>
<td>0x46</td>
<td>0x5f</td>
</tr>
<tr>
<td>7</td>
<td>007</td>
<td>B</td>
<td>BEL (bell)</td>
<td>39</td>
<td>27</td>
<td>0x47</td>
<td>0x47</td>
<td>0x5f</td>
</tr>
<tr>
<td>8</td>
<td>010</td>
<td>C</td>
<td>BS (backspace)</td>
<td>40</td>
<td>28</td>
<td>0x48</td>
<td>0x48</td>
<td>0x5f</td>
</tr>
<tr>
<td>9</td>
<td>011</td>
<td>D</td>
<td>HT (horizontal tab)</td>
<td>41</td>
<td>29</td>
<td>0x49</td>
<td>0x49</td>
<td>0x5f</td>
</tr>
<tr>
<td>10</td>
<td>A12</td>
<td>LF</td>
<td>NL line feed, new line</td>
<td>42</td>
<td>2A</td>
<td>0x4A</td>
<td>0x4A</td>
<td>0x5f</td>
</tr>
<tr>
<td>11</td>
<td>B13</td>
<td>VT</td>
<td>(vertical tab)</td>
<td>43</td>
<td>2B</td>
<td>0x4B</td>
<td>0x4B</td>
<td>0x5f</td>
</tr>
<tr>
<td>12</td>
<td>C14</td>
<td>FF</td>
<td>(form feed, new page)</td>
<td>44</td>
<td>2C</td>
<td>0x4C</td>
<td>0x4C</td>
<td>0x5f</td>
</tr>
<tr>
<td>13</td>
<td>D15</td>
<td>CR</td>
<td>(carriage return)</td>
<td>45</td>
<td>2D</td>
<td>0x4D</td>
<td>0x4D</td>
<td>0x5f</td>
</tr>
<tr>
<td>14</td>
<td>E16</td>
<td>SO</td>
<td>(shift out)</td>
<td>46</td>
<td>2E</td>
<td>0x4E</td>
<td>0x4E</td>
<td>0x5f</td>
</tr>
<tr>
<td>15</td>
<td>F17</td>
<td>ST</td>
<td>(shift in)</td>
<td>47</td>
<td>2F</td>
<td>0x4F</td>
<td>0x4F</td>
<td>0x5f</td>
</tr>
<tr>
<td>16</td>
<td>020</td>
<td>DLE</td>
<td>(data link escape)</td>
<td>48</td>
<td>30</td>
<td>0x60</td>
<td>0x60</td>
<td>0x5f</td>
</tr>
<tr>
<td>17</td>
<td>021</td>
<td>DC1</td>
<td>(device control 1)</td>
<td>49</td>
<td>31</td>
<td>0x61</td>
<td>0x61</td>
<td>0x5f</td>
</tr>
<tr>
<td>18</td>
<td>022</td>
<td>DC2</td>
<td>(device control 2)</td>
<td>50</td>
<td>32</td>
<td>0x62</td>
<td>0x62</td>
<td>0x5f</td>
</tr>
<tr>
<td>19</td>
<td>023</td>
<td>DC3</td>
<td>(device control 3)</td>
<td>51</td>
<td>33</td>
<td>0x63</td>
<td>0x63</td>
<td>0x5f</td>
</tr>
<tr>
<td>20</td>
<td>024</td>
<td>DC4</td>
<td>(device control 4)</td>
<td>52</td>
<td>34</td>
<td>0x64</td>
<td>0x64</td>
<td>0x5f</td>
</tr>
<tr>
<td>21</td>
<td>025</td>
<td>NAK</td>
<td>(negative acknowledge)</td>
<td>53</td>
<td>35</td>
<td>0x65</td>
<td>0x65</td>
<td>0x5f</td>
</tr>
<tr>
<td>22</td>
<td>026</td>
<td>SYN</td>
<td>(synchronous idle)</td>
<td>54</td>
<td>36</td>
<td>0x66</td>
<td>0x66</td>
<td>0x5f</td>
</tr>
<tr>
<td>23</td>
<td>027</td>
<td>ETB</td>
<td>(end of transmission block)</td>
<td>55</td>
<td>37</td>
<td>0x67</td>
<td>0x67</td>
<td>0x5f</td>
</tr>
<tr>
<td>24</td>
<td>028</td>
<td>CAN</td>
<td>(cancel)</td>
<td>56</td>
<td>38</td>
<td>0x68</td>
<td>0x68</td>
<td>0x5f</td>
</tr>
<tr>
<td>25</td>
<td>031</td>
<td>EM</td>
<td>(end of medium)</td>
<td>57</td>
<td>39</td>
<td>0x69</td>
<td>0x69</td>
<td>0x5f</td>
</tr>
<tr>
<td>26</td>
<td>032</td>
<td>SUB</td>
<td>(substitute)</td>
<td>58</td>
<td>3A</td>
<td>0x6A</td>
<td>0x6A</td>
<td>0x5f</td>
</tr>
<tr>
<td>27</td>
<td>033</td>
<td>ESC</td>
<td>(escape)</td>
<td>59</td>
<td>3B</td>
<td>0x6B</td>
<td>0x6B</td>
<td>0x5f</td>
</tr>
<tr>
<td>28</td>
<td>034</td>
<td>FS</td>
<td>(file separator)</td>
<td>60</td>
<td>3C</td>
<td>0x6C</td>
<td>0x6C</td>
<td>0x5f</td>
</tr>
<tr>
<td>29</td>
<td>035</td>
<td>GS</td>
<td>(group separator)</td>
<td>61</td>
<td>3D</td>
<td>0x6D</td>
<td>0x6D</td>
<td>0x5f</td>
</tr>
<tr>
<td>30</td>
<td>036</td>
<td>RS</td>
<td>(record separator)</td>
<td>62</td>
<td>3E</td>
<td>0x6E</td>
<td>0x6E</td>
<td>0x5f</td>
</tr>
<tr>
<td>31</td>
<td>037</td>
<td>US</td>
<td>(unit separator)</td>
<td>63</td>
<td>3F</td>
<td>0x6F</td>
<td>0x6F</td>
<td>0x5f</td>
</tr>
</tbody>
</table>
Interfacing LCD

- **Hardware**
  - 20 x 2-line LCD displays (two lines with 20 characters per line)
  - LCD has a display Data RAM (registers) that stores data in 8-bit character code.
  - Each register in Data RAM has its own address that corresponds to its position on the line.
    - The **address range** for Line 1 is 00 to 13H and Line 2 is 40H to 53H.
Interfacing LCD

**Driver HD77480**

- Three control signals:
  - RS – Register Select (RA3)
  - R/W – Read/Write (RA2)
  - E – Enable (RA1)

- Three power connections
  - Power, ground, and the variable register to control the brightness
Interfacing LCD

- Can be interfaced either in the 8-bit mode or the 4-bit mode
  - In the 8-bit mode, all eight data lines are connected for data transfer
  - In the 4-bit mode, only four data lines (DB7-DB4 or DB3-DB0) are connected and two transfers per character (or instruction) are needed
- Driver (HD77480) has two 8-bit internal registers
  - Instruction Register (IR) to write instructions to set up LCD
  - Data Register (DR) to write data (ASCII characters)
# Command and Instruction Set for LCD Type HD44780

<table>
<thead>
<tr>
<th>Command</th>
<th>Code</th>
<th>Description</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Display</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1</td>
<td>Clears the display and returns the cursor to the home position (address 0).</td>
<td>52μs-1.64ms</td>
</tr>
<tr>
<td>Return Home</td>
<td>0 0 0 0 0 0 0 0 0 0 0 1 0 1 0</td>
<td>Returns the cursor to the home position (address 0). Also returns a shifted display to the home position. DD RAM contents remain unchanged.</td>
<td>40μs-1.64ms</td>
</tr>
<tr>
<td>Entry Mode Set</td>
<td>0 0 0 0 0 0 0 0 0 0 0 1 0 1</td>
<td>Sets the display direction and enables/disables the display.</td>
<td>40μs</td>
</tr>
<tr>
<td>Display ON/OFF</td>
<td>0 0 0 0 0 0 0 0 0 0 1 1 0 0 0</td>
<td>Turns the display ON/OFF (D), or the cursor ON/OFF (C), and blinks the character at the cursor position ($) .</td>
<td>40μs</td>
</tr>
<tr>
<td>Cursor &amp; Display Shift</td>
<td>0 0 0 0 0 0 0 1 1 0 0 0 1</td>
<td>Moves the cursor and shifts the display without changing the DD RAM contents.</td>
<td>40μs</td>
</tr>
<tr>
<td>Function Set</td>
<td>0 0 0 0 1 0 0 0 0 0 0 0 0 0 0</td>
<td>Sets the display width (DL), the number of lines in the display (L), and the character font (F).</td>
<td>40μs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Code</th>
<th>Description</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set CG RAM Address</td>
<td>0 0 0 0 1 0 0 0 0 0 0 0 0 0 0</td>
<td>Sets the CG RAM address. CG RAM data can be read or altered after making this setting.</td>
<td>40μs</td>
</tr>
<tr>
<td>Set DP RAM Address</td>
<td>0 0 0 0 1 0 0 0 0 0 0 0 0 0 0</td>
<td>Sets the DP RAM address. Data may be written or read after making this setting.</td>
<td>40μs</td>
</tr>
<tr>
<td>Read Busy Flag &amp; Address</td>
<td>0 0 0 0 1 0 0 0 0 0 0 0 0 0 0</td>
<td>Reads the BUSY flag (BF) indicating that an internal operation is being performed and reads the address counter contents.</td>
<td>1μs</td>
</tr>
<tr>
<td>Write Data to CG or DP RAM</td>
<td>1 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>Writes data into DD RAM or CG RAM.</td>
<td>46μs</td>
</tr>
<tr>
<td>Read Data from CG or DP RAM</td>
<td>1 1 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>Reads data from DD RAM or CG RAM.</td>
<td>46μs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DD RAM: Display data RAM</th>
<th>CG RAM: Character generator RAM</th>
<th>AC: Address counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address: Used for both RD and CG RAM address.</td>
<td>Corresponds to cursor address.</td>
<td>Used for both RD and CG RAM address.</td>
</tr>
</tbody>
</table>

Notes:
- ID = 1: Increment ID = 0: Decrement
- S = 1: Accompanies display shift.
- S/C = 1: Display shift
- S/G = 0: Cursor move
- R/L = 1: Shift to the right, R/L = 0: Shift to the left.
- DL = 1: 8 bits DL = 0: 4 bits
- N = 1: 2 lines N = 0: 1 line
- F = 1: 5x10 dots F = 0: 5x7 dots
- BF = 1: Busy BF = 0: Can accept data
- # Set to 1 on 2x4 modules
- $ With KS0072 is Address Mode.
Interfacing LCD

- **LCD Operation**
  - When the MPU writes an instruction to IR or data to DR, the controller:
    - Sets the data line DB7 high as a flag indicating that the controller is busy completing the operation
    - Sets the data line DB7 low after the completion of the operation
  - The MPU should always check whether DB7 is low before sending an instruction or a data byte
  - After the power up, DB7 cannot be checked for the first two initialization instructions.
Interfacing LCD

- Writing to or reading from LCD

- The MPU:
  - Asserts RS low to select IR
  - Reads from LCD by asserting the R/W signal high
  - Asserts the E signal high and then low (toggles) to latch a data byte or an instruction

- Asserts RS high to select DR
- Writes into LCD by asserting the R/W signal low
- Asserts the E signal high and then low (toggles) to latch a data byte or an instruction
HD44780 Bus Timing

Read timing diagram

- Asserts RS high to select DR
- Writes into LCD by asserting the R/W signal low
- Asserts the E signal high and then low (toggles) to latch a data byte or an instruction

Write timing diagram

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Min</th>
<th>Typ</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_{CIRCLE}</td>
<td>Enable cycle time</td>
<td>1000</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>PW_{EH}</td>
<td>Enable pulse width (high level)</td>
<td>450</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_{ES, t_{EF}}</td>
<td>Enable rise and decay time</td>
<td>-</td>
<td>25</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_{AS}</td>
<td>Address setup time, RS, R/W, E</td>
<td>60</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_{DDR}</td>
<td>Data delay time</td>
<td>-</td>
<td>360</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_{DSW}</td>
<td>Data setup time</td>
<td>195</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_{W}</td>
<td>Data hold time (write)</td>
<td>10</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_{DSW}</td>
<td>Data hold time (read)</td>
<td>5</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_{AH}</td>
<td>Address hold time</td>
<td>20</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>
Interfacing LCD (Write)

Software

To write into the LCD, the program should:

- Send the initial instructions (commands) before it can check DB7 to set up the LCD in the 4-bit or the 8-bit mode.
- Check DB7 and continue to check until it goes low.
- Write instructions to IR to set up the LCD parameters such as the number of display lines and cursor status.
- Write data to display a message.
Resetting LCD

- In 4-bit mode the data is sent in nibbles
  - First we send the higher nibble and then the lower nibble.
- To enable the 4-bit mode of LCD, we need to follow special sequence of initialization that tells the LCD controller that user has selected 4-bit mode of operation:
  - Wait for about 20mS
  - Send the first init value (0x30)
  - Wait for about 10mS
  - Send second init value (0x30)
  - Wait for about 1mS
  - Send third init value (0x30)
  - Wait for 1mS
  - Select \texttt{bus width} (0x30 - for 8-bit and 0x20 for 4-bit)
  - Wait for 1mS

http://www.8051projects.net/lcd-interfacing(commands).php
Organic LED

- Organic light-emitting diodes - OLEDs - emit light when a current flows through them
- Unlike conventional LEDs, OLEDs are made from layers of plastic and other organic (carbon-based) materials
  - Very flexible!
- Applications: displays in MP3 players and phones
- Advantages:
  - cheaper than the techniques required to make conventional LEDs.
  - inherently thin
  - can be made on flexible plastic substrates
  - all colors, and multi-colors, are possible
- Disadvantages
  - incredibly sensitive to moisture which leads to short life - glass blocks all moisture, so displays made on a glass substrate and covered by a second glass sheet can have a long life, particularly if the edges are hermetically sealed

Organic LED

- OLEDs are generally made of several layers
- A typical stack (variations are possible):
  - Anode
  - Electron donor
  - Electron transport
  - Emitter
  - Hole transport
  - Hole donor.
  - Cathode
References

• [http://home.iae.nl/users/pouweha/lcd/lcd0.shtml](http://home.iae.nl/users/pouweha/lcd/lcd0.shtml)
• Huang