Chapter 7: Parallel Ports

The PIC18 Microcontroller

Han-Way Huang

Minnesota State University, Mankato
Introduction

- Embedded products are designed to allow the user to provide input and receive results.
- The speed and characteristics of I/O devices are quite different from the CPU.
- Peripheral chips are used to resolve the difference between the I/O devices and the CPU.
- The major function of the interface chip is to synchronize the data transfer between the CPU and I/O devices.
- Resistors may be needed to limit the current flow, whereas buffer chips may be needed to increase the current flow required by the I/O devices.
- An interface chip consists of control registers, data registers, status registers, data direction register, and control circuitry.
- Control registers allow the user to set up operation parameters.
- Data registers hold the data to be output or received.
- Status register records the status of the I/O operation.
- Data direction register allows the user to set the direction of data transfer.
- To input, the processor reads data from the data register.
- To output, the processor writes data into the data register.
- The interconnection between the microprocessor, interface chips, and I/O devices are shown in Figure 7.1.
- The address decoder in Figure 7.1 allows only one interface chip to exchange data with the microprocessor at a time.
- Data transfer can be proceeded in parallel or bit by bit (serial method).
- Serial method is meant to be used with slower I/O devices.
- Parallel method is meant to be used with high-speed devices.
- This chapter focuses on parallel I/O.
I/O Addressing

- The processor needs to access the registers of the interface chip to perform the I/O operation.
- An address is needed to select the register inside the interface chip and instructions need to be executed to carry out the appropriate operations.
- Two issues arise here:

  1. Address space sharing
  2. Instruction set and addressing modes sharing
I/O Synchronization

- The role of the interface chip is shown in Figure 7.2.
- For input, the CPU needs to make sure it reads when the data is valid and reads only once.
- For output, the CPU needs to make sure that the output device is ready to accept new data.
- There are two aspects in the I/O synchronization: the synchronization between the CPU and interface chip and the synchronization between the interface chip and the I/O device.

Figure 7.2 The role of an interface chip
Synchronization between the CPU and the Interface Chip

1. **Polling** method. Use a flag bit to indicate the readiness of I/O operation.

2. **Interrupt-driven** method. Use the interrupt signal to inform the CPU that new data is available or output device is ready for more data.

None of the PIC18 I/O ports support either method directly.
Synchronizing the Interface Chip with I/O Devices

1. **No method**
   - Input: The interface returns the current value of the input pins.
   - Output: The interface drives the data written by the CPU directly on output pins.

2. **Strobe method**
   - A strobe signal is used to synchronize data transfer.
   - Input: The input device places data on input pins, wait until data is stable, and then asserts the strobe signal to inform the interface chip to latch the data.
   - Output: The interface chip drives data to be output on the output pins, wait until data is stable, and then asserts the strobe signal to inform the output device to latch the data.
   - PIC18 does not support this method.

3. **Handshake method**
   - Two handshake signals are used to synchronize the data transfer between the interface chip and I/O device. H1 is driven by interface chip and H2 is driven by I/O device.
   - There are two versions of handshaking: pulse mode and interlock mode.
   - Handshaking signals transactions for input and output are shown in Figure 7.3 and 7.4, respectively.
Figure 7.3 Input Handshakes

(a) Interlocked

(b) Pulse mode
Figure 7.4 Output Handshaking

(a) Interlocked

(b) Pulse Mode

Copyright @ 2005 Thomson Delmar Learning
Overview of the PIC18 Parallel I/O Ports

- I/O pins are often grouped into ports.
- A port consists of up to 8 pins, a data direction register (TRISx), a latch register (LATx), and a data register (PORTx); where, x = A…H, J, K.
- Data to be output is written into the latch, which in turn drives the output pins.
- A PIC18 may have as many as 10 I/O ports.
- An I/O port is often multiplexed with one or more peripheral functions.
- When a peripheral function is enabled, the I/O pins cannot be used for general-purpose I/O.
- Devices with 18 pins and 20 pins have two I/O ports: A and B.
- Devices with 28 pins have three I/O ports: A, B, and C.
- Devices with 40 pins (DIP package) and 44 pins (PLCC package) have five parallel ports: A, B, C, D, and E.
- Devices with 64 pins (TQFP package) and 68 pins (PLCC package) have seven ports: A, B, C, D, E, F, and G.
- Most 80-pin (TQFP package) and 84-pin devices (PLCC package) have nine ports but some 84-pin devices (e.g., PIC18F858 in PLCC package) have 10 ports.
- The number of pins available in each port is shown in Table 7.1.
### Table 7.1 Number of pins available in each parallel port

<table>
<thead>
<tr>
<th>Port Name</th>
<th>No. of Pins</th>
<th>Pin Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
<td>RA6..RA0</td>
</tr>
<tr>
<td>B</td>
<td>8</td>
<td>RB7..RB0</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
<td>RC7..RC0</td>
</tr>
<tr>
<td>D</td>
<td>8</td>
<td>RD7..RD0</td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>RE7..RE0</td>
</tr>
<tr>
<td>F</td>
<td>8</td>
<td>RF7..RF0</td>
</tr>
<tr>
<td>G</td>
<td>5</td>
<td>RG4..RG0</td>
</tr>
<tr>
<td>H</td>
<td>8</td>
<td>RH7..RH0</td>
</tr>
<tr>
<td>J*</td>
<td>8</td>
<td>RJ7..RJ0</td>
</tr>
<tr>
<td>K</td>
<td>4</td>
<td>RK3..RK0</td>
</tr>
</tbody>
</table>

Note. The Port J of the PIC18C858 has only four pins. Port K is available in PIC18F858 only.
Reading and Writing the I/O Ports

- Data direction needs to be set before the I/O operation.
- To configure an I/O pin for input, set the associated bit in the TRIS register to 1.
- To configure an I/O pin for output, set the associated bit in the TRIS register to 0.

Example 7.1 Write an instruction sequence to output the hex value 0x33 to port D.
Solution:
The port D should be configured for output before data is written to it.

clrfsb TRISD,A ; configure port D for output
movlw 0x33
movwf PORTD,A

Example 7.2 Write an instruction sequence to read the current value of port D into WREG.
Solution:
setf TRISD,A ; configure port D for input
movf PORTD,W,A ; read port D pin value into WREG
- Part of an I/O port pins can be configured for input whereas others are configured for output.

**Example 7.3** Configure the upper four pins of port D for input and the lower four pins for output.

**Solution:**

```
movlw 0xF0
movwf TRISD,A
```
Interfacing with LEDs

- Several interfacing methods are possible.
- Method (a) is used in SSE demo boards.
- Method (c) provides some protection to the microcontroller.

Figure 7.15 PIC18 output port pin driving an LED
Interfacing with Seven-Segment Displays

- Used to display a few decimal digits.
- A buffer chip is often used to provide enough current to drive the displays.
- A one-display circuit is shown in Figure 7.16.
- Multiple-digit displays can be achieved by using time-multiplexing technique shown in Figure 7.17.
- To display a decimal digit, appropriate pattern must be output to port D in Figure 7.16 and 7.17. These patterns are shown in Table 7.2.

Figure 7.16 Driving a single seven-segment display
Figure 7.17 Port B and Port D together drive six seven-segment displays
The display a decimal digit on display $i$, perform the following operations:

1. Configure both port B and port D for output
2. Drive RB$i$ pin to high
3. Drive other port B pins to low
4. Output the hex value corresponding to the specified decimal digit to port D

<table>
<thead>
<tr>
<th>Decimal digit</th>
<th>Segments</th>
<th>Corresponding Hex Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 1 1 1 1 1 1 0</td>
<td>0x7E</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1 1 0 0 0 0</td>
<td>0x30</td>
</tr>
<tr>
<td>2</td>
<td>1 1 0 1 1 0 1</td>
<td>0x6D</td>
</tr>
<tr>
<td>3</td>
<td>1 1 1 1 0 0 1</td>
<td>0x79</td>
</tr>
<tr>
<td>4</td>
<td>0 1 1 0 0 1 1</td>
<td>0x33</td>
</tr>
<tr>
<td>5</td>
<td>1 0 1 1 0 1 1</td>
<td>0x5B</td>
</tr>
<tr>
<td>6</td>
<td>1 0 1 1 1 1 1</td>
<td>0x5F</td>
</tr>
<tr>
<td>7</td>
<td>1 1 1 0 0 0 0</td>
<td>0x70</td>
</tr>
<tr>
<td>8</td>
<td>1 1 1 1 1 1 1</td>
<td>0x7F</td>
</tr>
<tr>
<td>9</td>
<td>1 1 1 1 0 1 1</td>
<td>0x7B</td>
</tr>
</tbody>
</table>
Example 7.4 Write an instruction sequence to display 5 on the seven-segment display #4 in Figure 7.17.

Solution:
In assembly,

```
clrf TRISB,A ; configure port B for output
clrf TRISD,A ; configure port D for output
movlw 0x5B ; output the segment pattern of 5
movwf PORTD ;
movlw 0x10 ; enable display #4 to light
movwf PORTB ;
```

In C language,

```
TRISB = 0x00;
TRISD = 0x00;
PORTD = 0x5B;
PORTB = 0x10;
```
How to display multiple digit in Figure 7.17?

- Light each digit in turn briefly (say, 1 ms a time) and then turned off.
- When one display is lighted, all other displays are turned off.
- Because of the **persistence of vision**, all displays will appear lighted simultaneously.

**Example 7.5** Write a program to display 654321 on the six seven-segment displays in Figure 7.17 assuming that the PIC18 is running with a 32-MHz crystal oscillator. **Solution:** The values to be output to port B and port D are listed in Table 7.5.

<table>
<thead>
<tr>
<th>Seven-segment display</th>
<th>Displayed BCD digit</th>
<th>Port D</th>
<th>Port B</th>
</tr>
</thead>
<tbody>
<tr>
<td>#5</td>
<td>6</td>
<td>0x5F</td>
<td>0x20</td>
</tr>
<tr>
<td>#4</td>
<td>5</td>
<td>0x5B</td>
<td>0x10</td>
</tr>
<tr>
<td>#3</td>
<td>4</td>
<td>0x33</td>
<td>0x08</td>
</tr>
<tr>
<td>#2</td>
<td>3</td>
<td>0x79</td>
<td>0x04</td>
</tr>
<tr>
<td>#1</td>
<td>2</td>
<td>0x6D</td>
<td>0x02</td>
</tr>
<tr>
<td>#0</td>
<td>1</td>
<td>0x30</td>
<td>0x01</td>
</tr>
</tbody>
</table>
#include <p18F8680.inc>
radix dec

dup_nop macro kk ; this macro will duplicate the “nop” instruction
variable i ; kk times
i = 0 ; need to be at a separate line
while i < kk
nop
i += 1 ; need to be at separate line
endw
endm

lp_cnt set 0x00 ; use data memory 0 as loop count
lp_cnt1 set 0x01 ; another loop count
org 0x00
goto start
org 0x08
retfie
org 0x18
retfie

start clrf TRISB,A ; configure Port B for output
clrf TRISD,A ; configure Port D for output
forever movlw 0x06 ; there are six digits to be displayed
movwf lp_cnt ; " 
The PIC18 Microcontroller

```
loop
    movlw upper disp_tab
    movwf TBLPTRU,A ; "
    movlw high disp_tab ; "
    movwf TBLPTRH,A ; "
    movlw low disp_tab ; "
    movwf TBLPTRL,A ; "
    tblrd*+
    movff TABLAT,PORTD ; output the seven-segment pattern
    tblrd*+
    movff TABLAT,PORTB ; turn on one display
    call wait_ms ; wait for half a millisecond
    decfsz lp_cnt,F,A ; decrement the loop count
    goto loop ; read the next pattern
    goto forever ; go to the start of the pattern table
end
```
The subroutine `wait_ms` creates a delay of 1 ms for a demo board running at 32 MHz crystal oscillator.

```
wait_ms
    movlw 200
    movwf lp_cnt1, A
    again
    dup_nop 37 ; 37 instruction cycles
    decfsz lp_cnt1,F,A ; 1 instruction cycle (2 when [lp_cnt1] = 0)
    goto again ; 2 instruction cycles
    return

disp_tab
    db 0x5F,0x20
    db 0x5B,0x10
    db 0x33,0x08
    db 0x33
    db 0x79,0x04
    db 0x6D,0x02
    db 0x30,0x01
end
```
```c
#include <delays.h>
#include <p18F8680.h>
char display[6][2] = {{0x5F,0x20},{0x5BD,0x10},{0x33,0x08},{0x79,0x04},
                     {0x6D,0x02}, {0x30,0x01}};

void main ()
{
    int i;
    TRISB = 0x00;  /* configure Port B for output */
    TRISD = 0x00;  /* configure Port D for output */
    while (1) {
        for (i = 0; i < 6; i++) {
            PORTD = display[i][0];  /* output display pattern */
            PORTB = display[i][1];  /* turn on one display */
            Delay1KTCYx(8);  /* wait for 1 ms */
        }
    }
}
```
Using the LCD kits with the HD44780 LCD Controller

- Can control the display of up to 80 characters in an LCD kit.
- Block diagram of the HD44780 is shown in Figure 7.20.
- A standard connector for the HD44780-based connection method is shown in Table 7.5.

Figure 7.20 Block diagram of the DMC-20434 LCD kit
Table 7.4 Pin assignment for displays with less than 80 characters

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>symbol</th>
<th>I/O</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VSS</td>
<td>-</td>
<td>Power supply (GND)</td>
</tr>
<tr>
<td>2</td>
<td>VCC</td>
<td>-</td>
<td>Power supply (+5V)</td>
</tr>
<tr>
<td>3</td>
<td>VEE</td>
<td>-</td>
<td>Contrast adjust</td>
</tr>
<tr>
<td>4</td>
<td>RS</td>
<td>I</td>
<td>0 = instruction input, 1 = data input</td>
</tr>
<tr>
<td>5</td>
<td>R/W</td>
<td>I</td>
<td>0 = write to LCD, 1 = read from LCD</td>
</tr>
<tr>
<td>6</td>
<td>E</td>
<td>I</td>
<td>enable signal</td>
</tr>
<tr>
<td>7</td>
<td>DB0</td>
<td>I/O</td>
<td>data bus line 0</td>
</tr>
<tr>
<td>8</td>
<td>DB1</td>
<td>I/O</td>
<td>data bus line 1</td>
</tr>
<tr>
<td>9</td>
<td>DB2</td>
<td>I/O</td>
<td>data bus line 2</td>
</tr>
<tr>
<td>10</td>
<td>DB3</td>
<td>I/O</td>
<td>data bus line 3</td>
</tr>
<tr>
<td>11</td>
<td>DB4</td>
<td>I/O</td>
<td>data bus line 4</td>
</tr>
<tr>
<td>12</td>
<td>DB5</td>
<td>I/O</td>
<td>data bus line 5</td>
</tr>
<tr>
<td>13</td>
<td>DB6</td>
<td>I/O</td>
<td>data bus line 6</td>
</tr>
<tr>
<td>14</td>
<td>DB7</td>
<td>I/O</td>
<td>data bus line 7</td>
</tr>
</tbody>
</table>

- The DB7..DB0 pins are used to exchange data with the microcontroller.
- The R/W pin determines the data transfer direction.
- The E pin enables the data exchange with the LCD kit.
- The HD44780 has an **instruction register** and a **data register**.
- The RS signal selects the instruction register when it is low and selects the data register when it is 1.
- The VEE signal is used to adjust the brightness of the LCD and is often connected to a potentiometer.
The HD44780 Controller

- The HD44780 can be configured to control 1-line, 2-line, and 4-line displays.
- The HD44780 controller has character generator ROM that can generate 208 $5 \times 8$ dot characters and 32 $5 \times 10$ dot characters.
- Character generator RAM is available for defining 8 $5 \times 8$ character patterns and 4 $5 \times 10$ character patterns.
- A set of instructions (Table 7.6) is available for users to configure the LCD operations.
- The HD44780 controller has **display data memory** (DDRAM) to store the display data.
- The **address** of a display data memory location determines where the next character will appear on the LCD screen.
- The mapping from display data memory to the LCD screen position is not sequential and is shown in Table 7.8a, 7.8b, and 7.8c.

Table 7.8a DDRAM address usage for a 1-line LCD

<table>
<thead>
<tr>
<th>Display size</th>
<th>Visible character positions</th>
<th>DDRAM addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 * 8</td>
<td>00..07</td>
<td>0x00..0x07</td>
</tr>
<tr>
<td>1 * 16</td>
<td>00..15</td>
<td>0x00..0x0F</td>
</tr>
<tr>
<td>1 * 20</td>
<td>00..19</td>
<td>0x00..0x13</td>
</tr>
<tr>
<td>1 * 24</td>
<td>00..23</td>
<td>0x00..0x17</td>
</tr>
<tr>
<td>1 * 32</td>
<td>00..31</td>
<td>0x00..0x1F</td>
</tr>
<tr>
<td>1 * 40</td>
<td>00..39</td>
<td>0x00..0x27</td>
</tr>
</tbody>
</table>
Table 7.8b DDRAM address usage for a 2-line LCD

<table>
<thead>
<tr>
<th>Display size</th>
<th>Visible character positions</th>
<th>DDRAM addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 * 16</td>
<td>00..15</td>
<td>0x00..0x0F + 0x40..0x4F</td>
</tr>
<tr>
<td>2 * 20</td>
<td>00..19</td>
<td>0x00..0x13 + 0x40..0x53</td>
</tr>
<tr>
<td>2 * 24</td>
<td>00..23</td>
<td>0x00..0x17 + 0x40..0x57</td>
</tr>
<tr>
<td>2 * 32</td>
<td>00..31</td>
<td>0x00..0x1F + 0x40..0x5F</td>
</tr>
<tr>
<td>2 * 40</td>
<td>00..39</td>
<td>0x00..0x27 + 0x40..0x67</td>
</tr>
</tbody>
</table>

Table 7.8c DDRAM address usage for a 4-line LCD

<table>
<thead>
<tr>
<th>Display size</th>
<th>Visible character positions</th>
<th>DDRAM addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 * 16</td>
<td>00..15</td>
<td>0x00..0x0F + 0x40..0x4F + 0x14..0x23 + 0x54..0x63</td>
</tr>
<tr>
<td>4 * 20</td>
<td>00..19</td>
<td>0x00..0x13 + 0x40..0x53 + 0x14..0x27 + 0x54..0x67</td>
</tr>
<tr>
<td>4 * 40</td>
<td>00..39 on 1st controller and 00..39 on 2nd controller</td>
<td>0x00..0x27 + 0x40..0x67 on 1st controller and 0x00..0x27 + 0x40..0x67 on 2nd controller</td>
</tr>
</tbody>
</table>
Table 7.6 HD44780 instruction set

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Description</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear display</td>
<td>RS: 0, R/W: 0, B7: 0, B6: 0, B5: 0, B4: 0, B3: 0, B2: 0, B1: 0, B0: 1</td>
<td>Clears display and returns cursor to the home position (address 0).</td>
<td>1.64 ms</td>
</tr>
<tr>
<td>Cursor home</td>
<td>RS: 0, R/W: 0, B7: 0, B6: 0, B5: 0, B4: 0, B3: 0, B2: 0, B1: 0, B0: 1</td>
<td>Returns cursor to home position (address 0). Also returns display being shifted to the original position. DDRAM contents remain unchanged.</td>
<td>1.64 ms</td>
</tr>
<tr>
<td>Entry modeset</td>
<td>RS: 0, R/W: 0, B7: 0, B6: 0, B5: 0, B4: 0, B3: 0, B2: 0, B1: 1, B0: 1</td>
<td>Set cursor move direction (I/D), specifies to shift the display (S). These operations are performed during data read/write.</td>
<td>40 μs</td>
</tr>
<tr>
<td>Display on/off control</td>
<td>RS: 0, R/W: 0, B7: 0, B6: 0, B5: 0, B4: 0, B3: 0, B2: 0, B1: 1, B0: 0</td>
<td>Sets on/off of all display (D), cursor on/off (C) and blink of cursor position character (B).</td>
<td>40 μs</td>
</tr>
<tr>
<td>Cursor/display shift</td>
<td>RS: 0, R/W: 0, B7: 0, B6: 0, B5: 0, B4: 0, B3: 0, B2: 0, B1: 1, B0: 1</td>
<td>Sets cursor-move or display-(S/C), shift direction (R/L). DDRAM contents remains unchanged.</td>
<td>40 μs</td>
</tr>
<tr>
<td>Function set</td>
<td>RS: 0, R/W: 0, B7: 0, B6: 0, B5: 0, B4: 0, B3: 0, B2: 0, B1: 1, B0: 1</td>
<td>Sets interface data length (DL), number of display line (N) and character font (F).</td>
<td>40 μs</td>
</tr>
<tr>
<td>Set CGRAM address</td>
<td>RS: 0, R/W: 0, B7: 0, B6: 0, B5: 0, B4: 0, B3: 0, B2: 1, B1: 0</td>
<td>Sets the CGRAM address. CGRAM data is sent and received after this setting.</td>
<td>40 μs</td>
</tr>
<tr>
<td>Set DDRAM address</td>
<td>RS: 0, R/W: 0, B7: 0, B6: 0, B5: 0, B4: 0, B3: 0, B2: 1, B1: 1</td>
<td>Sets the DDRAM address. DDRAM data is sent and received after this setting.</td>
<td>40 μs</td>
</tr>
<tr>
<td>Read busy flag and address counter</td>
<td>RS: 0, R/W: 1, B7: 0, B6: 0, B5: 0, B4: 0, B3: 0, B2: 0, B1: 1, B0: 0</td>
<td>Reads busy flag (BF) indicating internal operation is being performed and reads CGRAM or DDRAM address counter contents (depending on previous instruction).</td>
<td>0 μs</td>
</tr>
<tr>
<td>Write to CGRAM or DDRAM</td>
<td>RS: 1, R/W: 0, B7: 0, B6: 0, B5: 0, B4: 0, B3: 0, B2: 0, B1: 0, B0: 1</td>
<td>Writes data to CGRAM or DDRAM.</td>
<td>40 μs</td>
</tr>
<tr>
<td>Read from CGRAM or DDRAM</td>
<td>RS: 1, R/W: 1, B7: 0, B6: 0, B5: 0, B4: 0, B3: 0, B2: 0, B1: 1, B0: 1</td>
<td>Reads data from CGRAM or DDRAM.</td>
<td>40 μs</td>
</tr>
</tbody>
</table>
Table 7.7 LCD instruction bit names

<table>
<thead>
<tr>
<th>Bit name</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/D</td>
<td>0 = decrement cursor position. 1 = increment cursor position</td>
</tr>
<tr>
<td>S</td>
<td>0 = no display shift. 1 = display shift</td>
</tr>
<tr>
<td>D</td>
<td>0 = display off 1 = display on</td>
</tr>
<tr>
<td>C</td>
<td>0 = cursor off 1 = cursor on</td>
</tr>
<tr>
<td>B</td>
<td>0 = cursor blink off 1 = cursor blink on</td>
</tr>
<tr>
<td>S/C</td>
<td>0 = move cursor 1 = shift display</td>
</tr>
<tr>
<td>R/L</td>
<td>0 = shift left 1 = shift right</td>
</tr>
<tr>
<td>DL</td>
<td>0 = 4-bit interface 1 = 8-bit interface</td>
</tr>
<tr>
<td>N</td>
<td>0 = 1/8 or 1/11 duty (1 line) 1 = 1/16 duty (2 lines)</td>
</tr>
<tr>
<td>F</td>
<td>0 = 5x7 dots 1 = 5 x 10 dots</td>
</tr>
<tr>
<td>BF</td>
<td>0 = can accept instruction 1 = internal operation in progress</td>
</tr>
</tbody>
</table>

The IR Register
- The microcontroller writes command (or instruction) into this register to configure the LCD operation parameters.
- The LCD instruction set is listed in Table 7.6.
The DR Register
- The data to be written into the DDRAM or CGRAM is written into this register.
- To read data from DDRAM or CGRAM, the microcontroller reads from this register.

Address Counter (AC)
- This 7-bit address counter keeps track of the address of the next DDRAM or CGRAM location to be accessed.
- The selection of DDRAM or CGRAM is determined by the instruction.
- The content of this register is incremented after the access to DDRAM or CGRAM.
- The register selection is shown in Table 7.9.

<table>
<thead>
<tr>
<th>RS</th>
<th>R/W</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>IR write as an internal operation (display clear, etc)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Read busy flag (DB7) and address counter (DB0 to DB6)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>DR write as an internal operation (DR to DDRAM or CGRAM)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>DR read as an internal operation (DDRAM or CGRAM to DR)</td>
</tr>
</tbody>
</table>
HD44780 Commands

1. **Clear display** (0x01). This command clears the LCD screen, sets the address counter to 0, sets the cursor to the upper left corner of the LCD screen, and also sets the I/D bit to 1 in entry mode.

2. **Return Home** (0x02). Sets DDRAM address 0 into address counter, move cursor to the upper left corner of the display but does not change the contents of the DDRAM.

3. **Entry Mode Set.** The I/D bit of this command controls the incrementing or decrementing of the DDRAM address. The display will shift if the S bit is 1 when the DDRAM is being written into.

4. **Display On/Off Control.** This command can turn on the display (D = 1), turn on the cursor (C = 1), and turn on the cursor blinking (B = 1).

5. **Cursor or Display Shift.** This command determines to shift the cursor (S/C = 0 bit) or display (S/C bit = 1) to the right (R/L bit = 1) or to the left (R/L bit = 0).
6. **Function Set.** This command allows the user to set the interface data length, select the number of lines, and the font size:
   - **DL** bit. When set to 1, data is exchanged in 8-bit length. Otherwise, data is exchanged in 4-bit length.
   - **N** bit. When set to 1, two-line display is selected. Otherwise, 1-line display is selected.
   - **F** bit. When set to 0, the $5 \times 7$ font is selected. Otherwise, $5 \times 10$ font is selected.

7. **Set CGRAM Address.** This command contains the CGRAM address to be set into the address counter.

8. **Set DDRAM Address.** This command allows to set the DDRAM address into the address counter.

9. **Read Busy Flag and Address.** This instruction reads the busy flag (BF) and the address counter. The BF flag indicates whether the LCD controller is still executing the previously received command.
Interfacing the HD44780 with the PIC18

- There are two methods to interface the LCD kit to the PIC18.
- The first method is treating the LCD kit as an I/O device. One of the available I/O port is used to connect to the DB7…DB0 pins. Other port pins are used to connect to the R/W, E, and S inputs of the LCD kit and generate the required timing sequence on these three pins.
- The second method is to treat the LCD kit as a memory device and use the address decoder to generate a select signal to select the LCD kit. This method allows the user to use the table read and table write instructions to access the LCD kit.
- The second method cannot be used by those PIC18 members that do not support the external memory.
- Only the first method will be used to interface the LCD kit with the PIC18 in this text.
- The circuit for the first approach is shown in Figure 7.21.
The PIC18 Microcontroller

Figure 7.21a LCD interface example (8-bit bus)
Figure 7.21b LCD interface example (4-bit bus)
Timing Consideration for the LCD

- Certain timing parameters must be satisfied in order to successfully access the LCD.
- The read and write timing diagrams are shown in Figure 7.22 and 7.23.
- The HD44780 can operate at either 1 MHz or 2 MHz. The values of timing parameters for 2-MHz operation are shown in Table 7.12.

![HD44780 LCD controller read timing diagram](image-url)
Figure 7.23 HD44780 LCD controller write timing diagram

Table 7.12 HD44780 bus timing parameters (2 MHz operation)

<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_{\text{CYCLE}} )</td>
<td>Enable cycle time</td>
<td>500</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>( PW_{\text{EH}} )</td>
<td>Enable pulse width (high level)</td>
<td>230</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{\text{Er}} ), ( t_{\text{Ef}} )</td>
<td>Enable rise and decay time</td>
<td>-</td>
<td>-</td>
<td>20</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{\text{AS}} )</td>
<td>Address setup time, RS, R/W, E</td>
<td>40</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{\text{DDR}} )</td>
<td>Data delay time</td>
<td>-</td>
<td>-</td>
<td>160</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{\text{DSW}} )</td>
<td>Data setup time</td>
<td>80</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{\text{H}} )</td>
<td>Data hold time (write)</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{\text{DHR}} )</td>
<td>Data hold time (read)</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{\text{AH}} )</td>
<td>Address hold time</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>ns</td>
</tr>
</tbody>
</table>
**LCD Programming**

To display messages on the LCD, the following subroutines are needed:

1. **LCD_rdy:** make sure the LCD is not busy with internal operation
2. **init_lcd:** configure the LCD operation parameters
3. **LCD_cmd:** send a command to the LCD
4. **send2LCD:** generate a timing sequence to output the contents of WREG to the LCD
5. **LCD_putchar:** output the character in WREG to the LCD
6. **LCD_putstr:** output the string in program memory pointed to by TBLPTR to the LCD
7. **LCD_putsr:** output the string in data memory pointed to by FSR0 to the LCD
8. **get_DDRAM_addr:** reads the current DDRAM address and returns it in PRODL.
The following signals are defined to simplify the access of signals for the SSE8680 and SSE8720 demo boards:

<table>
<thead>
<tr>
<th>Signal</th>
<th>Equivalent</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD_DATA</td>
<td>equ</td>
<td>PORTE</td>
<td>LCD data bus</td>
</tr>
<tr>
<td>#define</td>
<td>LCD_D_DIR</td>
<td>TRISE,A</td>
<td>LCD data port direction</td>
</tr>
<tr>
<td>#define</td>
<td>LCD_E</td>
<td>PORTH,7,A</td>
<td>LCD E clock</td>
</tr>
<tr>
<td>#define</td>
<td>LCD_E_DIR</td>
<td>TRISH,7,A</td>
<td>LCD E clock direction</td>
</tr>
<tr>
<td>#define</td>
<td>LCD_RW</td>
<td>PORTH,6,A</td>
<td>LCD read/write line</td>
</tr>
<tr>
<td>#define</td>
<td>LCD_RW_DIR</td>
<td>TRISH,6,A</td>
<td>LCD R/W pin direction</td>
</tr>
<tr>
<td>#define</td>
<td>LCD_RS</td>
<td>PORTH,5,A</td>
<td>LCD register select line</td>
</tr>
<tr>
<td>#define</td>
<td>LCD_RS_DIR</td>
<td>TRISH,5,A</td>
<td>LCD RS signal direction</td>
</tr>
<tr>
<td>#define</td>
<td>lcd_cport</td>
<td>PORTH,A</td>
<td></td>
</tr>
</tbody>
</table>
Procedure for **Writing a Byte** into the **IR register**

**Step 1**
Pull the RS and the E signals to low.

**Step 2**
Pull the R/W signal to low.

**Step 3**
Pull the E signal to high.

**Step 4**
Output data to the output port attached to the LCD data bus. (Need to configure port E for output).

**Step 5**
Pull the E signal to low.
### Procedure for **Reading a Byte** from the **IR register**

**Step 1**  
Pull the RS and the E signals to low.  

**Step 2**  
Pull the R/W signal to high.  

**Step 3**  
Pull the E signal to high.  

**Step 4**  
Read the value of the LCD data bus. (need to configure port E for input)  

**Step 5**  
Pull the E signal to low.
Procedure for Writing a Byte into the LCD data register

**Step 1**
Pull the RS signal to high and pull the E signals to low.

**Step 2**
Pull the R/W signal to low.

**Step 3**
Pull the E signal to high.

**Step 4**
Output data to the I/O port attached to the LCD data bus.

**Step 5**
Pull the E signal to low.
; The following subroutines are written to work for f_{osc} up to 32 MHz.

```assembly
LCD_rdy setf LCD_D_DIR ; configure LCD data port for input
     bcf LCD_RS     ; select IR register
     bsf LCD_RW    ; setup to read busy flag
     bsf LCD_E     ; pull LCD E-line to high
     nop           ; small delay
     movf LCD_DATA,W,A ; read busy flag and DDRAM address
     nop           ; small delay to lengthen E pulse
     bcf LCD_E     ; pull LCD E-line to low
     btfsc WREG,7,A ; is busy flag (BF) cleared
     goto LCD_rdy  
     clrf LCD_D_DIR ; configure data pins for output
     return
```
A common LCD configuration for a 2x20 LCD is as follows:

- Set the LCD to 2-line display
- Turn on display, cursor, and blinking
- Shift cursor right
- Clear the display and return the cursor to home position
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LCD_init</code></td>
<td>clrf LCD_ctl_port ; make sure the LCD control port is low</td>
</tr>
<tr>
<td></td>
<td>bcf LCD_E_DIR ; configure control lines</td>
</tr>
<tr>
<td></td>
<td>bcf LCD_RW_DIR ; directions to output</td>
</tr>
<tr>
<td></td>
<td>bcf LCD_RS_DIR ; &quot;</td>
</tr>
<tr>
<td></td>
<td>movlw 0x3C ; configure display to 2 x 20</td>
</tr>
<tr>
<td></td>
<td>call LCD_cmd ; send command to LCD</td>
</tr>
<tr>
<td></td>
<td>movlw 0x0F ; turn on display and cursor</td>
</tr>
<tr>
<td></td>
<td>call LCD_cmd ; &quot;</td>
</tr>
<tr>
<td></td>
<td>movlw 0x14 ; shift cursor right</td>
</tr>
<tr>
<td></td>
<td>call LCD_cmd ; &quot;</td>
</tr>
<tr>
<td></td>
<td>movlw 0x01 ; clear cursor and return to home position</td>
</tr>
<tr>
<td></td>
<td>call LCD_cmd ; &quot;</td>
</tr>
<tr>
<td></td>
<td>return</td>
</tr>
</tbody>
</table>
get_DDRAM_addr

setf LCD_D_DIR ; configure LCD data port for input
bcf LCD_RS ; select IR register
bsf LCD_RW ; setup to read busy flag
bsf LCD_E ; pull LCD E-line to high
nop ; add a small delay
movf LCD_DATA,W ; read busy flag and DDRAM address
nop ; small delay to length E pulse
bcf LCD_E ; pull LCD E-line to low
movwf PRODL,A
return

send2LCD ; this subroutine writes WREG to LCD

bsf LCD_RS ; select DR register
bcf LCD_RW ; Set write mode
bsf LCD_E ; Setup to clock data
nop
movwf LCD_DATA,A ; write into LCD DR
nop ; a short delay
bcf LCD_E
return
Output a Character to a 2x20 or a 4x20 LCD
- One needs to check if the address counter points to the end of a row.
- To find out if the address counter is at the end of a row, one needs to read the DDRAM address.
- The character that is output to the end of a row cannot be displayed and must be resent to the beginning of the next row.

```
LCD.puts
pushr WREG ; save WREG
call LCD_rdy ; wait until LCD is not busy
popr WREG ; restore data in WREG
call send2LCD ; write WREG content to LCD
pushr WREG ; save WREG data
call get_DDRAM_addr ; get the DDRAM address in PRODL
movlw 0x13
cpseq PRODL,A
goto chk_0x53
; reach the end of first line
movlw 0xC0
call LCD_cmd ; set DDRAM address to 0x40
call LCD_rdy
popr WREG
popr WREG
return
```
chk_0x53  movlw      0x53
        cpfseq     PRODL,A
        goto      chk_0x27

; reach the end of the second line
movlw      0x94
call       LCD_cmd
call       LCD_rdy
popr       WREG
call       send2LCD
return

; reach the end of the third line
chk_0x27  movlw      0x27
        cpfseq     PRODL,A
        return
movlw      0xD4
        movlw      0x54
        call       LCD_cmd
        call       LCD_rdy
        popr       WREG
        call       send2LCD
        return
; output a string (pointed to by TBLPTR) in program memory to the LCD
LCD_putstr tblrd*+
    movf TABLAT,W,A
    tstfz WREG,A
    goto send_it
    return
send_it call LCD_putc
    goto LCD_putstr

; output a string (pointed to by FSR0) in data memory to the LCD
LCD_putsr movf POSTINC0,W
    tstfz WREG,A
    goto send_it
    return
send_it call LCD_putc
    goto LCD_putsr
In C language,

```c
#define LCD_DATA PORTE
#define LCD_D_DIR TRISE
#define LCD_RS PORTHbits.RH5
#define LCD_RS_DIR TRISHbits.TRISH5
#define LCD_RW PORTHbits.RH6
#define LCD_RW_DIR TRISHbits.TRISH6
#define LCD_E PORTHbits.RH7
#define LCD_E_DIR TRISHbits.TRISH7

/* These are function prototype definitions */
void LCD_rdy(void);
void LCD_cmd(char cx);
void LCD_init(void);
char get_DDRAM_addr(void);
void LCD_putchar(char dx);
void LCD_putstr(rom char *ptr);
void send2LCD(char xy);
```
/*******************************************/
/* the following function waits until the LCD is not busy. */
/*******************************************/

void LCD_rdy(void)
{
    char test;
    LCD_D_DIR = 0xFF; /* configure LCD data bus for input */
    test = 0x80;
    while (test) {
        while (test)
        {
            LCD_RS = 0; /* select IR register */
            LCD_RW = 1; /* Set read mode */
            LCD_E = 1; /* Setup to clock data */
            test = LCD_DATA;
            Nop();
            LCD_E = 0; /* complete a read cycle */
            test &= 0x80; /* check bit 7 */
        }
    }
    LCD_D_DIR = 0x00; /* configure LCD data bus for output */
}
/******************************************
/* The following function sends a command to the LCD. */
******************************************/

void LCD_cmd(char cx)
{
    LCD_rdy();             /* wait until LCD is ready */
    LCD_RS = 0;           /* select IR register */
    LCD_RW = 0;           /* Set write mode */
    LCD_E = 1;            /* Setup to clock data */
    Nop();
    LCD_DATA = cx;        /* send out the command */
    Nop();
    LCD_E = 0;            /* small delay to lengthen E pulse */
}
/**************************************************************************/
/* The following function initializes the LCD kit properly.               */
**************************************************************************/

void LCD_init(void)
{
    PORTH = 0; /* make sure LCD control port is low */
    LCD_E_DIR = 0; /* configure LCD control port for output */
    LCD_RS_DIR = 0; /* “ */
    LCD_RW_DIR = 0; /* “ */
    LCD_cmd(0x3C); /* configure display to 2 rows */
    LCD_cmd(0x0F); /* turn on display, cursor, and blinking */
    LCD_cmd(0x14); /* shift cursor right */
    LCD_cmd(0x01); /* clear display and move cursor to home */
}
/*******************************************************************/
/* The following function obtains the LCD cursor address. */
*******************************************************************/

char get_DDRAM_addr (void)
{
    char temp;
    LCD_D_DIR = 0xFF; /* configure LCD data port for input */
    LCD_RS = 0; /* select IR register */
    LCD_RW = 1; /* setup to read busy flag */
    LCD_E = 1; /* pull LCD E-line to high */
    Nop(); /* small delay */
    temp = LCD_DATA & 0x7F; /* read DDRAM address */
    Nop(); /* small delay to length E pulse */
    LCD_E = 0; /* pull LCD E-line to low */
    return temp;
}
/******************************************************************************
/* The following function sends a character to the LCD kit. */
*******************************************************************************/

void LCD_putc (char dx)
{
    char addr;
    LCD_rdy();                         /* wait until LCD internal operation is complete */
    send2LCD(dx);
    LCD_rdy();                         /* wait until LCD internal operation is complete */
    addr = get_DDRAM_addr();
    if (addr == 0x13) {
        LCD_cmd(0xC0);                 /* set address to 0x40 (1st column of the 2nd row) */
        LCD_rdy();
        send2LCD(dx);
    }
    else if(addr == 0x53) {
        LCD_cmd(0x94);                 /* set address to 0x14 (1st column of the 3rd row) */
        LCD_rdy();
        send2LCD(dx);
    }
}
else if(addr == 0x27) {
    LCD_cmd(0x0D4);  /* set address to 0x54 (1st column of the 4th row) */
    LCD_rdy();
    send2LCD(dx);
}

/*************************************************************************
/* The following function outputs a string to the LCD.                      */
/*************************************************************************

void LCD_putstr(rom char *ptr)
{
    while (*ptr) {
        LCD_putc (*ptr);
        ptr++;
    }
}
/**************************************************************/
/* The following function outputs a string to the LCD. */
/**************************************************************/
void LCD_putsr(char *ptr)
{
    while (*ptr) {
        LCD_putc (*ptr);
        ptr++;
    }
}
/**************************************************************/
/* The following function writes a character to the LCD. */
/**************************************************************/
void send2LCD(char xy)
{
    LCD_RS = 1;
    LCD_RW = 0;
    LCD_E = 1;
    LCD_DATA = xy;
    Nop();
    Nop();
    LCD_E = 0;
}
Interfacing with DIP Switches

Reading a byte from the DIP switches to WREG

\[
\begin{align*}
\text{movlw} &\quad \text{0xFF} &; \text{configure port F for input} \\
\text{movwf} &\quad \text{TRISF} &; \quad \text{“} \\
\text{movf} &\quad \text{PORTF,W} &; \text{read portF}
\end{align*}
\]
Interfacing with Keypad

Types of Key Switches

1. Membrane: A plastic or rubber membrane presses one conductor onto another. This type of switches can be made very small.
2. Capacitive: Two parallel plates. Pressing the plates changes the distance between the plates and changes the capacitance.
3. Hall effect: The motion of the magnetic flux lines of a permanent magnet perpendicular to a crystal is detected as voltage appearing between the two faces of the crystal.
4. Mechanical: Two metal contacts are brought together to complete an electrical circuit.

Mechanical Keypads and Keyboard

- Low cost and strength of construction
- Most popular
- Pressing the key switch generates a series of pulses instead of a single clean output
- Human being cannot press and release the key switch 20 ms
- A debouncing process is required for correct operation
**Keypad Input Program** Consists of Three Parts

1. Keypad scanning
2. Key switch debouncing
3. Table lookup

**Keypad Scanning**

- Performed to detect which key is being pressed
- Performed row by row and column by column
- The rows and columns of a keypad are simply conductors
- A simple keypad is shown in Figure 7.26b.
- The row to be scanned is pulled to low. Other rows are pulled high.
- When a key is pressed, the corresponding row and column are shorted together and is detected low.
- The diodes in Figure 7.26b provides protection of accidental simultaneous press of multiple keys.
Figure 7.26b Sixteen-key keypad connected to PIC18 (used in all SSE demo boards)
Keypad Debouncing

- When a key is pressed, the voltage of the key switch falls and rises a few times within a period of about 5 ms as a contact bounces.
- A debouncer will recognize that the switch is closed after the voltage is low for about 10 ms and will recognize that the switch is open after the voltage is high for about 10 ms.
- Both hardware and software debouncing solutions are available.
- The simplest and most popular software solution is **wait-and-see**. The program simply waits for 10 ms after detecting a key switch is closed and re-examines the same key.
- Three hardware debouncing techniques are shown in Figure 7.27.

ASCII Code Lookup

- The keypad input subroutine returns the ASCII code to the caller.
- This step can be embedded in the debouncing procedure.
(a) Set-reset latch

(b) CMOS gate debouncer

(c) Integrating RC circuit debouncer

Figure 7.27 Hardware debouncing techniques
Example 7.10 Write a program to perform keypad scanning, debouncing, and returns the ASCII code of the pressed key to the caller.

Solution:

The instruction sequence for the first row is as follows:

```
kepad_dir  equ    TRISJ  ; keypad direction control register
kepad      equ    PORTJ ; keypad port

get_key    movlw  0x0F    ; configure the upper four pins of keypad
             movwf  keypad_dir,A  ; port for output, others for input
             movlw  0xF0    ; set all keypad rows to high
             iorwf  keypad,F  ;
             scan_r0   movlw  0xEF  ; prepare to scan row 0 (driven by RB4)
             andwf  keypad,F,A
             scan_k0   btfss  keypad,0,A  ; check key 0
                         goto   db_key0
             scan_k1   btfss  keypad,1,A  ; check key 1
                         goto   db_key1
             scan_k2   btfss  keypad,2,A  ; check key 2
                         goto   db_key2
             scan_k3   btfss  keypad,3,A  ; check key 3
                         goto   db_key3
...```

The PIC18 Microcontroller

```assembly
    db_key0    call    wait10ms ; wait for 10 ms
    btfsc    keypad,0,A ; is key 0 still closed?
    goto    scan_k1 ; key 0 not pressed, check key 1
    movlw    0x30
    return

    db_key1    call    wait10ms
    btfsc    keypad,1,A ; is key 1 still closed?
    goto    scan_k2
    movlw    0x31
    return

    db_key2    call    wait10ms
    btfsc    keypad,2,A ; is key 2 still closed?
    goto    scan_k3
    movlw    0x32
    return

    db_key3    call    wait10ms
    btfsc    keypad,3,A
    goto    scan_r1
    movlw    0x33
    return
...```
This subroutine creates a 10 ms delay using timer 0 with $f_{\text{OSC}} = 32$ MHz.

```assembly
wait10ms
    bcf INTCON,INT0IE,A ; disable TMR0 interrupt
    bcf INTCON,INT0IF,A ; clear TMR0IF flag
    movlw 0x03 ; configure TMR0 with 1:16 prescaler
    movwf T0CON,A ; configure TMR0
    movlw 0x3C
    movwf TMR0H,A
    movlw 0xBB
    movwf TMR0L,A ; load (50000 - 12) into TMR0
    bsf T0CON,TMR0ON,A ; enable TMR0

dlyloop
    btfss INTCON,INT0IF,A ; is 10 ms over yet?
    goto dlyloop
    return
end
```

The C language version is in the following slides:

Copyright @ 2005 Thomson Delmar Learning
#include <p18F8680.h>
#define keypad_dir TRISJ
#define keypad PORTJ

void wait_10ms(void);

unsigned char get_key(void)
{
    keypad_dir = 0x0F;  /* configure RJ7..RJ4 for output */
    keypad |= 0xF0;    /* RJ3..RJ0 for input */
    while (1) {
        keypad &= 0xEF;  /* set RJ4 to low to scan the first row */
        if (!(keypadbits.RB0)) {
            wait_10ms();
            if (!(keypadbits.RB0)) return 0x30; /* return the ASCII code of 0 */
        }
        if (!(keypadbits.RB1)) {
            wait_10ms();
            if (!(keypadbits.RB1)) return 0x31; /* return the ASCII code of 1 */
        }
    }
    ...
}
The MAX5102 DAC

- Dual-channel, 8-bit DAC made by Maxim
- Block diagram and pin assignment are shown in Figure 7.28 and 7.29.

Figure 7.28 MAX5102 Pin configuration

Figure 7.29 MAX5102 Pin configuration
## Pin Functions

- **OUTA, OUTB:** analog voltage output.
- **D7..D0:** data inputs. Digital code to be converted is sent to MX5102 over these eight pins.
- **REF:** Reference voltage input. This signal controls the magnitude of OUTA and OUTB.
- **WR:** Write. When this signal is low, new digital code can be written into the MAX5102.
- **A0:** DAC address select bit. DAC address select bit. Channel A is selected when A0 = 0.
- **VDD:** Positive supply voltage. The range of VDD is from 5V to 15V.
- **GND:** Ground.
- **SHDN:** Shutdown. When set to high, this signal shuts down the chip to save power.

## Voltage Output

\[ V_{\text{OUT}} = \frac{(NB \times V_{\text{REF}})}{256} \]

where, \(NB\) is the digital input to be converted
\(V_{\text{REF}}\) is the reference voltage
Functioning of the MAX5102 DAC

- To write new data to be converted, the \( \overline{WR} \) input must be low.
- Data input is latched on the rising edge of the \( \overline{WR} \) signal.
- After \( \overline{WR} \) goes high, the data input can change without affecting \( V_{OUT} \).
- Circuit connection with the PIC18 is shown in Figure 7.31.

![Circuit connection diagram between the MAX5102 and PIC18](image-url)

Figure 7.30 Circuit connection between the MAX5102 and PIC18
Example 7.11 Write a program to generate a 1KHz square wave from OUTA pin assuming the instruction cycle time is 200 ns (correspond to 20 MHz crystal oscillator).

Solution: The procedure is as follows:

Step 1
Configure the pins RD7..RD0, RB1..RB0 for output.

Step 2
Clear the RB0 pin to low to select OUTA channel.

Step 3
Pull the RB1 pin to low.

Step 4
Output 255 to Port D.

Step 5
Pull the RB1 pin to high.

Step 6
Wait for 0.5 ms.

Step 7
Pull the RB1 pin to low.

Step 8
Output 0 to Port D.

Step 9
Pull the RB1 pin to high.
Step 10
Wait for 0.5 ms.

Step 11
Go to Step 3.

The C program to generate the waveform is as follows:

```c
#define DAC_data PORTD /* DAC data port */
#define DAC_dat_dir TRISD /* DAC data bus direction register */
#define DAC_ctl_dir TRISB /* DAC control pins direction */
#define DAC_WR PORTBbits.RB1 /* DAC write control pin */
#define DAC_A0 PORTBbits.RB0 /* DAC A0 pin */
#define output 0x00
#define high 1
#define low 0

void main (void)
{
    DAC_dat_dir = output; /* configure DAC data bus direction to output */
    DAC_ctl_dir &= 0xFC; /* configure WR & A0 pins for output */
    DAC_A0 = 0; /* select OUTA channel */
}
```
while (1) {
    DAC_WR = low;
    DAC_data = 255;  /* output a high voltage */
    DAC_WR = high;
    Delay100TCYx(25);  /* wait for 0.5 ms */
    DAC_WR = low;
    DAC_data = 0;  /* output a low voltage */
    DAC_WR = high;
    Delay100TCYx(25);  /* wait for 0.5 ms */
}