Parallel PORTS: Input-Output Unit

Content:
- Parallel ports (simple parallel I/O port and multi-functional parallel I/O ports)
- Pull-up resistor.
- I/O Ports of PIC16F887
- Interfacing Switch, LED, 7-segment display, Matrix display

Prepared By-
Mohammed Abdul kader
Lecturer, EEE, IIUC
Parallel ports

- Parallel ports provide the ability to input or output binary data with a single bit allocated to each pin within the port.
- They are called parallel ports because the initial chips that provided this support grouped several pins together to create a controllable data port similar to that used for data and address buses. It transfers multiple bits of information simultaneously, hence the name parallel port. Although the name implies that the pins are grouped together, the individual bits and pins within the port can usually be used independently of each other.

These ports are used to provide parallel interfaces such as the keyboard interface, switch interface, printer interface, output signals to LEDs and alphanumeric displays, LCD displays and so on. As inputs, they can be used with switches and keyboards to support control panels.
Parallel ports (Cont.)

A simple parallel I/O port

✓ The basic operation is shown in the diagram which depicts an 8 pin port.
✓ The port is controlled by two registers: a data direction register which defines whether each pin is an output or an input and a data register which is used to set an output value by writing to it and to obtain an input value by reading from it.
✓ The actual implementation typically uses a couple of buffers which are enabled depending on the setting of the corresponding bit in the data direction register.
Multi-function I/O ports

✓ With many parallel I/O devices that are available today, either as part of the on-chip peripheral set or as an external device, the pins are described as general-purpose and can be shared with other peripherals.
✓ For example, a pin may be used as part of a serial port as a control signal.
✓ It may be used as a chip select for the memory design or simply as an I/O pin.
✓ The function that the pin performs is set up internally through the use of a function register which internally configures how the external pin is connected internally.
✓ If this is not set up correctly, then despite the correct programming of the other registers, the pin will not function as expected.
Pull-up resistors

It is important to check if a parallel I/O port or pin expects an external pull-up resistor. Some devices incorporate it internally and therefore do not need it. If it is needed and not supplied, it can cause incorrect data on reading the port and prevent the port from turning off an external device.
**INPUT/OUTPUT PORTS - PIC16F887**

**35 input/output pins**

to synchronize the operation of I/O ports with the internal 8-bit organization of the microcontroller, they are, grouped into five ports denoted by:

- **PORT A**: 8 bits (8 pins)
- **PORT B**: 8 bits (8 pins)
- **PORT C**: 8 bits (8 pins)
- **PORTD**: 8 bits (8 pins)
- **PORTE**: 4 bits (4 pins)

**The exception is the RE3 pin which is always configured as an input.**

### Controlling I/O Ports

- Every Port has a corresponding TRIS register to control whether it is input or output port.
- By clearing any bit of the TRIS register (bit=0), the corresponding port pin is configured as an output. Similarly, by setting any bit of the TRIS register (bit=1), the corresponding port pin is configured as an input. This rule is easy to remember: **0 = Output, 1 = Input.**
**PORT A**

- 8-bit wide, bidirectional port.
- Bits of the TRISA and ANSEL registers control the Port A pins.
- All Port A pins act as digital inputs/outputs. Five of them can also be analog inputs (denoted by AN):

<table>
<thead>
<tr>
<th>Bit name</th>
<th>ANSEL</th>
<th>TRISA</th>
<th>PORTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 7</td>
<td>Bit 7</td>
<td>Bit 7</td>
<td>RA7</td>
</tr>
<tr>
<td>Bit 6</td>
<td>Bit 6</td>
<td>Bit 6</td>
<td>RA6</td>
</tr>
<tr>
<td>Bit 5</td>
<td>Bit 5</td>
<td>Bit 5</td>
<td>RA5</td>
</tr>
<tr>
<td>Bit 4</td>
<td>Bit 4</td>
<td>Bit 4</td>
<td>RA4</td>
</tr>
<tr>
<td>Bit 3</td>
<td>Bit 3</td>
<td>Bit 3</td>
<td>RA3</td>
</tr>
<tr>
<td>Bit 2</td>
<td>Bit 2</td>
<td>Bit 2</td>
<td>RA2</td>
</tr>
<tr>
<td>Bit 1</td>
<td>Bit 1</td>
<td>Bit 1</td>
<td>RA1</td>
</tr>
<tr>
<td>Bit 0</td>
<td>Bit 0</td>
<td>Bit 0</td>
<td>RA0</td>
</tr>
</tbody>
</table>

RA0 = AN0 (determined by the ANS0 bit of the ANSEL register)
RA1 = AN1 (determined by the ANS1 bit of the ANSEL register)
RA2 = AN2 (determined by the ANS2 bit of the ANSEL register)
RA3 = AN3 (determined by the ANS3 bit of the ANSEL register)
RA5 = AN4 (determined by the ANS4 bit of the ANSEL register)

To configure a pin as an analog input, the appropriate bit of the ANSEL or ANSELH registers must be set (1). To configure a pin as a digital input/output, the appropriate bit must be cleared (0).
PORTB and TRISB register

- Similar to port A, a logic one (1) in the TRISB register configures the appropriate portB pin as an input and vice versa.
- Six pins of this port can act as analog inputs (AN). The bits of the ANSELH register determine whether these pins are to be configured as analog inputs or digital inputs/outputs:

<table>
<thead>
<tr>
<th>PORTB</th>
<th>RB7</th>
<th>RB6</th>
<th>RB5</th>
<th>RB4</th>
<th>RB3</th>
<th>RB2</th>
<th>RB1</th>
<th>RB0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R/W (x)</td>
<td>R/W (x)</td>
<td>R/W (x)</td>
<td>R/W (x)</td>
<td>R/W (x)</td>
<td>R/W (x)</td>
<td>R/W (x)</td>
<td>Features</td>
</tr>
<tr>
<td></td>
<td>Bit 7</td>
<td>Bit 6</td>
<td>Bit 5</td>
<td>Bit 4</td>
<td>Bit 3</td>
<td>Bit 2</td>
<td>Bit 1</td>
<td>Bit 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TRISB</th>
<th>TRISB7</th>
<th>TRISB6</th>
<th>TRISB5</th>
<th>TRISB4</th>
<th>TRISB3</th>
<th>TRISB2</th>
<th>TRISB1</th>
<th>TRISB0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R/W (1)</td>
<td>R/W (1)</td>
<td>R/W (1)</td>
<td>R/W (1)</td>
<td>R/W (1)</td>
<td>R/W (1)</td>
<td>R/W (1)</td>
<td>Features</td>
</tr>
<tr>
<td></td>
<td>Bit 7</td>
<td>Bit 6</td>
<td>Bit 5</td>
<td>Bit 4</td>
<td>Bit 3</td>
<td>Bit 2</td>
<td>Bit 1</td>
<td>Bit 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ANSELH</th>
<th>ANS13</th>
<th>ANS12</th>
<th>ANS11</th>
<th>ANS10</th>
<th>ANS9</th>
<th>ANS8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R/W (1)</td>
<td>R/W (1)</td>
<td>R/W (1)</td>
<td>R/W (1)</td>
<td>R/W (1)</td>
<td>R/W (1)</td>
</tr>
<tr>
<td></td>
<td>Bit 7</td>
<td>Bit 6</td>
<td>Bit 5</td>
<td>Bit 4</td>
<td>Bit 3</td>
<td>Bit 2</td>
</tr>
</tbody>
</table>

RB0 = AN12 (determined by the ANS12 bit of the ANSELH register)
RB1 = AN10 (determined by the ANS10 bit of the ANSELH register)
RB2 = AN8 (determined by the ANS8 bit of the ANSELH register)
RB3 = AN9 (determined by the ANS9 bit of the ANSELH register)
RB4 = AN11 (determined by the ANS11 bit of the ANSELH register)
RB5 = AN13 (determined by the ANS13 bit of the ANSELH register)
PORTB and TRISB register (Continued)

All the port B pins have built in pull-up resistors, which make them ideal for connection to push buttons (keyboard), switches and optocouplers. In order to connect these resistors to the microcontroller ports, the appropriate bit of the WPUB register should be set.*

* Apart from the bits of the WPUB register, there is another bit affecting the installation of all pull-up resistors. It is the RBPU bit of the OPTION_REG.
**PORTC and TRISC register**

- Port C is an 8-bit wide, bidirectional port.
- Bits of the TRISC register determine the function of its pins. Similar to other ports, a logic one (1) in the TRISC register configures the appropriate portC pin as an input.

**PORTD and TRISD register**

- Port D is an 8-bit wide, bidirectional port.
- Bits of the TRISD register determine the function of its pins. A logic one (1) in the TRISD register configures the appropriate portD pin as an input.
**PORTE and TRISE register**

- Port E is a 4-bit wide, bidirectional port.
- The TRISE register’s bits determine the function of its pins. Similar to other ports, a logic one (1) in the TRISE register configures the appropriate portE pin as an input.
- The exception is the RE3 pin which is always configured as an input.

Similar to ports A and B, three pins can be configured as analog inputs in this case. The ANSELH register bits determine whether a pin will act as an analog input (AN) or digital input/output:

- RE0 = AN5 (determined by the ANS5 bit of the ANSEL register);
- RE1 = AN6 (determined by the ANS6 bit of the ANSEL register); and
- RE2 = AN7 (determined by the ANS7 bit of the ANSEL register).
Summary of Controlling I/O Pins
Interfacing Switch and LED (Simple read and write operation)

Interfacing Switch: Switch Debouncing

Switch bouncing is another real-world problem that happens too quickly for human perception but which can doom an electronics project. When a switch is toggled, contacts have to physically move from one position to another. As the components of the switch settle into their new position, they mechanically bounce, causing the underlying circuit to be opened and closed several times. For embedded systems designers, the most common exposure to the problem is with user interface switches, in which proper care must be taken to correctly count the number of times a user presses and releases a switch.

The left-hand image shows a simple push switch with a pull-up resistor. The right hand image shows the trace at the output terminal, Vout, when the switch is pressed. As can be seen, pressing the switch does not provide a clean edge. If this signal was used as an input to a digital counter, for example, you'd get multiple counts rather than the expected single count. Note that the same can also occur on the release of a switch.

*In the switch waveform the bouncing lasts for about 150us.*
Eliminating debouncing

Hardware solution (adding extra components with pin)

There are many different approaches to cleaning up switch bounce. Below is a debouncing circuit. The basic idea is to use a capacitor to filter out any quick changes in the switch signal.

Starting with the switch open.
- The capacitor C1 will charge via R1 and D1.
- In time, C1 will charge and Vb will reach within 0.7V of Vcc.
- Therefore the output of the inverting Schmitt trigger will be a logic 0.

Now close the switch
- The capacitor will discharge via R2.
- In time, C1 will discharge and Vb will reach 0V.
- Therefore the output of the inverting Schmitt trigger will be a logic 1.
Interfacing Switch and LED (Simple read and write operation)-Continued

Eliminating debouncing (Cont.)

✔ But what about bounce conditions? If bounce occurs and there are short periods of switch closure or opening, the capacitor will stop the voltage at Vb immediately reaching Vcc or GND. Although, bouncing will cause slight charging and discharging of the capacitor, the hysteresis of the Schmitt trigger input will stop the output from switching.

✔ What about the diode? Well the resistor R2 is required as a discharge path for the capacitor, without it, C1 will be shorted when the switch is closed. Without the diode, D1, both R1 and R2 would form the capacitor charge path when the switch is open. The combination of R1 and R2 would increase the capacitor charge time, slowing down the circuit. So, can't you just make R1 smaller? Ideally no, when the switch is closed, R1 is connected across the supply rails, so too small a resistor value would lead to unwanted wasted current.
By software

Debouncing a switch in software is very simple. The basic idea is to sample the switch signal at a regular interval and filter out any glitches. There are a couple of approaches to achieving this. One of the approach is discussed below-

_This approach uses a counter to time how long the switch signal has been low. If the signal has been low continuously for a set amount of time, then it is considered pressed and stable._

1. Setup a counter variable, initialize to zero.
2. Setup a regular sampling event, perhaps using a timer. Use a period of about 1ms.
3. On a sample event:
   4. if switch signal is high then
   5. Reset the counter variable to zero
   6. Set internal switch state to released
   7. else
   8. Increment the counter variable to a maximum of 10
   9. end if
10. if counter=10 then
11. Set internal switch state to pressed
12. end if
Debouncing solution in MikroC Pro

**Button Library**
The Button Library provides routines for detecting button presses and debouncing (eliminating the influence of contact flickering upon pressing a button).

<table>
<thead>
<tr>
<th>Prototype</th>
<th><code>unsigned short Button(unsigned short *port, unsigned short pin, unsigned short time, unsigned short active_state);</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Returns</strong></td>
<td>255 if the pin was in the active state for given period. 0 otherwise</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Function eliminates the influence of contact flickering upon pressing a button (debouncing). Parameter <code>port</code> specifies the location of the button; parameter <code>pin</code> is the pin number on designated port and goes from 0..7; parameter <code>time</code> is a debounce period in milliseconds; parameter <code>active_state</code> can be either 0 or 1, and it determines if the button is active upon logical zero or logical one.</td>
</tr>
<tr>
<td><strong>Requires</strong></td>
<td>Button pin must be configured as input.</td>
</tr>
</tbody>
</table>
Interfacing Switch and LED (Simple read and write operation)-Continued

Following example reads RB0, to which the button is connected; on transition from 1 to 0 (release of button), PORTC is inverted:

```c
bit oldstate;            // Old state flag

void main() {
  ANSEL  = 0;           // Configure AN pins as digital I/O
  ANSELH = 0;
  C1ON_bit = 0;         // Disable comparators
  C2ON_bit = 0;
  TRISB0_bit = 1;       // set RB0 pin as input
  TRISC = 0x00;         // Configure PORTC as output
  PORTC = 0xAA;         // Initial PORTC value
  oldstate = 0;
  do {
    if (Button(&PORTB, 0, 1, 1)) {               // Detect logical one
      oldstate = 1;                                // Update flag
    }
    if (oldstate && Button(&PORTB, 0, 1, 0)) {   // Detect one-to-zero transition
      PORTC = ~PORTC;                              // Invert PORTC
      oldstate = 0;                                // Update flag
    }
  } while(1);                                       // Endless loop
}
```
LED Interface

The resistor is important in interface to limit the flowing current and avoid damaging the LED.
Examples related to switch and LED interface

Example 1: Suppose, 8 LEDs are connected with PORTD of PIC16F887. Write a program in C to blink these every 1s interval.

```c
void main() {
    ANSEL=0;
    ANSELH=0; // All pins are digital I/O
    C1ON_bit=0;
    C2ON_bit=0; // Analog comparators connected to PORTA are disabled
    TRISD=0x00; // All pins of PORTD are output pin
    PORTD=0x00; // initial value of PORTD
    while(1) // infinite loop
    {
        delay_ms(1000); // Wait for 1 sec
        PORTD=0xFF; // All LED blinks
        delay_ms(1000); // Wait for 1 sec
        PORTD=0x00; // All LED stop blinking
    }
}
```
Example 2: Suppose, 8 LEDs are connected with PORTB of PIC16F887 and initial value of PORTB is (11)\textsubscript{16}. Write a program in C to invert the status of LEDs every 1s interval.

```c
void main() {
    ANSEL=0;
    ANSELH=0; // All pins are digital I/O
    C1ON_bit=0;
    C2ON_bit=0; // Analog comparators connected to PORTA are disabled
    TRISB=0x00; // All pins of PORTB are output pin
    PORTB=0x11; // initial value of PORTD
    while(1) // infinite loop
    {
        delay_ms(1000); // Wait for 1 sec
        PORTB=~PORTB; // All LED blinks
    }
}
```
Example 3: Suppose, 8 LEDs are connected with PORTB of PIC16F887. Write a program in C to show binary numbers 0 to 255 on LEDs. Use 200ms time delay between two numbers.

```c
unsigned char i;
void main() {
    ANSEL=0x00;
    ANSELH=0x00;
    TRISB=0x00;
    while(1)
    {
        for(i=0;i<=255;i++)
        {
            PORTB=i;
            delay_ms(200);
        }
    }
}
```
Example 4: Suppose, 8 LEDs are connected with PORTB of PIC16F887. Write a program in C to show the even binary numbers from 120 to 200 on LEDs. Use 500ms time delay between two numbers.

```
unsigned char i;
void main() {
    ANSEL=0x00;
    ANSELH=0x00;
    TRISB=0x00;
    while(1)
    {
        for(i=120;i<=200;i++)
        {
            PORTB=i;
            delay_ms(500);
        }
    }
}
```
Examples related to switch and LED interface (Continued)

Example 5: Suppose, Initial value of PORTB is F1H. Write a program in C to shift the values in left direction until all bits of PORTB becomes clear. Hold the value of PORTB for 1sec after a shift. Repeat the sequence.

```c
void main() {
    ANSEL=0x00;
    ANSELH=0x00;
    TRISB=0x00;
    PORTB=0xF1;
    while(1)
    {
        delay_ms(1000);
        PORTB<<=1;
        if(PORTB==0)
            PORTB=0xF1;
    }
}
```

Example 6: Suppose, Initial value of PORTB is F1H. Write a program in C to shift the values in right direction until all bits of PORTB becomes clear. Hold the value of PORTB for 1sec after a shift. Repeat the sequence. (Similar to Example 5, use >> operator for right shift)
Example 7: Suppose, Initial value of PORTB is F1H. Write a program in C to rotate the value in right direction. Use 200ms time delay between two shifts.

```c
bit a;
void main() {
    ANSEL=0x00;
    ANSELH=0x00;
    TRISB=0x00;
    PORTB=0xF1;
    while(1)
    {
        a=PORTB.F0;
        delay_ms(200);
        PORTB>>=1;
        PORTB.F7=a;
    }
}
```
Examples related to switch and LED interface (Continued)

Example 8: Suppose, a PUSH switch is connected to PB0 pin and a LED is connected to PC5 pin of PIC16F887. Write a program in C, so that a press in push switch blinks the LED 5 times with an time interval of 200 ms and turn off after blinking.

```c
unsigned char i;
bit a;
void main() {
  ANSEL=0x00;
  ANSELH=0x00;
  TRISB.F0=1;
  TRISC.F5=0;
  OPTION_REG.F7=0;
  WPUB=0x01;
  while(1)
  {
    if (Button(&PORTB, 0, 1, 0))
    {
      a = 1;
    }
    if (a && Button(&PORTB, 0, 1, 1)) {
      for (i=1;i<=5; i++) {
        PORTC.F5=1;
        delay_ms(200);
        PORTC.F5=0;
        delay_ms(200);
      }
      a = 0;
    }
  }
}
```
Example 9: Suppose, a PUSH switch is connected to PB0 pin and a LED is connected to PC5 pin of PIC16F887. Write a program in C so that a press in push switch blinks the LED 5 times with an time interval of 200 ms and turn off after blinking.

```c
bit oldstate0;
bit oldstate1;
bit rotateleft;
bit rotateright;
void main() {
    ANSEL=0x00;
    ANSELH=0x00;
    TRISC=0x00;
    TRISB.F0=1;
    TRISB.F1=1;
    OPTION_REG.F7=0;
    //internal pullup register on
    WPUB=0x03;
    PORTC=0b11110000;
    while(1) {
        if (Button(&PORTB, 0, 1, 0)) {    //detect click on button
            oldstate0 = 1;
        }
        if (oldstate0 && Button(&PORTB, 0, 1, 1))
            oldstate0 = 0;
        if (oldstate0 & & Button(&PORTB, 0, 1, 1))
            //detect the release on clicked button, active on 0 to 1 transition (release of button)
            { rotateleft=PORTC.F7;
              PORTC<<=1;
              PORTC.F0=rotateleft;
              oldstate0 = 0;
            }
    }
```
Examples related to switch and LED interface (Continued)

```c
if (Button(&PORTB,1,1,0)) {
    oldstate1=1;
}
if(oldstate1 && Button(&PORTB,1,1,1)) {
    rotateright=PORTC.F0;
    PORTC>>=1;
    PORTC.F7=rotateright;
    oldstate1=0;
}
```

**Problem:**  
Design a 8-bit binary counter using microcontroller which increment its value when user press a push switch SW1 and decrement its value when user press a push switch SW2.
Interfacing LED displays

✓ Basically, an LED display is nothing more than several LEDs molded in the same plastic case. There are many types of displays composed of several dozens of built in diodes which can display different symbols.

✓ Most commonly used is a so called 7-segment display. It is composed of 8 LEDs, 7 segments are arranged as a rectangle for symbol displaying and there is an additional segment for decimal point displaying.

✓ In order to simplify connecting, anodes and cathodes of all diodes are connected to the common pin so that there are common anode displays and common cathode displays, respectively.

✓ On connecting, each diode is treated separately, which means that each must have its own current limiting resistor.

![Diagram of LED display and connection to microcontroller]
**Problem:** A common cathode seven segment display is connected with PORTC. The segment a, b, c, d, e, f and g are connected to PC0, PC1, PC2, PC3, PC4, PC5, PC6 and PC7 respectively. Write a program to show the numbers 2, 4, 6 and 8 in the display repeatedly at 1 second interval.

Main()
{
    ANSEL=0;
    ANSELH=0;
    TRISC= 0;
    While(1)
    {
        PORTC=0b01011011;
        delay_ms(1000);
        PORTC=0b01100110;
        Delay_ms(1000);
        PORTC=0b01111101;
        Delay_ms(1000);
        PORTC=0b01111111;
        Delay_ms(1000);
        PORTC=0b01111111;
        Delay_ms(1000);
    }
}
Interfacing Multiple LED displays

Displays connected to the microcontroller usually occupy a large number of valuable I/O pins, which can be a big problem especially if it is needed to display multi digit numbers. The problem is more than obvious if, for example, it is needed to display two 6-digit numbers (a simple calculation shows that 96 output pins are needed in this case). The solution to this problem is called MULTIPLEXING. Only one digit is active at a time, but they change their state so quickly making impression that all digits of a number are simultaneously active.

First a byte representing units is applied on a microcontroller port and a transistor T1 is activated at the same time. After a while, the transistor T1 is turned off, a byte representing tens is applied on a port and a transistor T2 is activated. This process is being cyclically repeated at high speed for all digits and corresponding transistors.
Decimal UP Counter with Four 7-Segment Display Multiplexing

This counter shows the decimal number 0 to 9999 in 7-segment display repeatedly. Programming code is given in next page.
Decimal UP Counter with Four 7-Segment Display Multiplexing (Cont.)

```c
unsigned short i, DD0, DD1, DD2, DD3;
unsigned int Count;
// ------ Function to Return mask for common anode 7-seg. display
unsigned short mask(unsigned short num)
{
    switch (num)
    {
        case 0 : return 0xC0;
        case 1 : return 0xF9;
        case 2 : return 0xA4;
        case 3 : return 0xB0;
        case 4 : return 0x99;
        case 5 : return 0x92;
        case 6 : return 0x82;
        case 7 : return 0xF8;
        case 8 : return 0x80;
        case 9 : return 0x90;
    } //case end
}
void main()
{
    ANSEL=0;
    ANSELH=0;  // All I/O pins are digital
    C1ON_bit=0;
    C2ON_bit=0;  // Disable Comparators
    TRISB = 0x00;  // Set PORTB direction to be output
    PORTB = 0xff;  // Turn OFF LEDs on PORTB
    TRISA = 0x00;  // PORTA is output
    Count = 0;  // Initial Value of Counter
}
```
Decimal UP Counter with Four 7-Segment Display Multiplexing (Cont.)

While(1)
{
    DD0 = Count%10; // Extract Ones Digit
    DD0 = mask(DD0);
    DD1 = (Count/10)%10; // Extract Tens Digit
    DD1 = mask(DD1);
    DD2 = (Count/100)%10; // Extract Hundreds Digit
    DD2 = mask(DD2);
    DD3 = (Count/1000); // Extract Thousands Digit
    DD3 = mask(DD3);
    for (i = 0; i<=50; i++)
    {
        PORTB = DD0;
        RA0_bit = 0; // Select Ones Digit
        RA1_bit = 1;
        RA2_bit = 1;
        RA3_bit = 1;
        delay_ms(5);
        PORTB = DD1;
        RA0_bit = 1;
Decimal UP Counter with Four 7-Segment Display Multiplexing (Cont.)

RA1_bit = 0; // Select Tens Digit
RA2_bit = 1;
RA3_bit = 1;
delay_ms(5);
PORTB = DD2;
RA0_bit = 1;
    RA1_bit = 1;
RA2_bit = 0; // Select Hundreds Digit
RA3_bit = 1;
delay_ms(5);
PORTB = DD3;
RA0_bit = 1;
RA1_bit = 1;
RA2_bit = 1;
RA3_bit = 0; // Select Thousands Digit
delay_ms(5);
}

Count = Count + 1;
if (Count > 9999)  Count = 0;
}
**Interfacing LED Matrix Display**

**Theory of LED dot matrix display**

- In a dot matrix display, multiple LEDs are wired together in rows and columns. This is done to minimize the number of pins required to drive them.

- For example, a 8×8 matrix of LEDs (shown below) would need 64 I/O pins, one for each LED pixel. By wiring all the anodes together in rows (R1 through R8), and cathodes in columns (C1 through C8), the required number of I/O pins is reduced to 16.

- Each LED is addressed by its row and column number. In the figure below, if R4 is pulled high and C3 is pulled low, the LED in fourth row and third column will be turned on. Characters can be displayed by fast scanning of either rows or columns.
Now, we will learn how to display still characters by column scanning in a standard 5×7 pixel format.

Suppose, we want to display the alphabet A. We will first select the column C1 (which means C1 is pulled low in this case), and deselect other columns by blocking their ground paths (one way of doing that is by pulling C2 through C5 pins to logic high). Now, the first column is active, and you need to turn on the LEDs in the rows R2 through R7 of this column, which can be done by applying forward bias voltages to these rows. Next, select the column C2 (and deselect all other columns), and apply forward bias to R1 and R5, and so on. Therefore, by scanning across the column quickly (> 100 times per second), and turning on the respective LEDs in each row of that column, the persistence of vision comes into play, and we perceive the display image as still.
Theory of LED dot matrix display (Cont.)

Necessity of external transistor

You should have noted that across each row, one pin is sourcing the current for only one LED at a time, but a column pin may have to sink the currents from more than one LED. For example, the column C1 should be able to sink the currents from 6 LEDs while displaying the alphabet ‘A’. A microcontroller’s I/O pin cannot sink this much of current, so external transistor arrays are required. You can use ULN2003A which is an IC that has transistor array inside.
Problem: The rows of a 7X5 LED matrix display are connected with PORTB and columns with PORTA. Write a program to show letters A, B, C and D to the display.

unsigned short Alphabets[20] = { 0x7e, 0x09, 0x09, 0x09, 0x7E, // A
                                   0x7f, 0x49, 0x49, 0x49, 0x36, // B
                                   0x3e, 0x41, 0x41, 0x41, 0x22, // C
                                   0x7f, 0x41, 0x41, 0x22, 0x1c, // D
                              };

unsigned short count, column, num, repeat;
void main() {
    ANSEL = 0x00;
    ANSELH = 0x00;
    C1ON_bit = 0;
    C2ON_bit = 0;
    TRISB = 0x00; // Set PORTB direction to be output
    TRISA = 0x00; // Set PORTA direction to be output
    PORTB = 0x00; // Turn OFF LEDs on PORTB
While(1)
{
    for (num=0; num<4; num++) {
        for (repeat=0; repeat<100; repeat++) {
            column = 1;
            for(count = num*5; count < (num*5+6); count++) {
                PORTB = Alphabets[count];
                PORTA = column;
                Delay_ms(1);
                column = column<<1;
            }
        }
    }
}
Interfacing Several Matrix Display

For interfacing two 8X8 dot matrix display we need 32 I/O pins. But this number can be reduced to only 5 by using shift register and a Johnson counter.

Here, matrix display is driving by row scanning
Role of shift registers (74HC595)

The use of shift registers minimizes the number of I/O pins required to drive the columns of the LED matrix. For driving 16 columns separately, we need 16 I/O pins of microcontroller, however, with the use of two 74HC595 ICs, this number is reduced to 3. 74HC595 is an 8-stage serial-in, serial or parallel-out shift register, with a storage register. The shift register and storage register have separate clocks: SH_CP (pin 11) and ST_CP (pin 12). Data is fed serially into the register through DS pin (14) and is shifted on the positive-going transitions of the SH_CP input. However, the data in each register does not appear at the output pin of 74HC595 unless it is transferred to the storage register. This happens on a positive-going transition of the ST_CP input. 74HC595 also provides a serial standard output, Q7’ (pin 9) for cascading. The serial output of the first shift register is connected to the serial input (DS pin) of the second shift register, so that the 16-bit column data can be transferred serially through the DS pin of the first shift register. This requires 16 clock pulses on SH_CP followed by a clock pulse on ST_CP. The asynchronous reset pin (MR) is always pulled high (deactivated) whereas the output enable (OE) pin is permanently grounded (always enabled).
Interfacing Several Matrix Display (Cont.)

*Role of counter (CD4017)*

CD4017 is a 5-stage divide-by-10 Johnson counter with 10 decoded outputs and a carry out bit. The counter is cleared to zero count by a logical “1” on its reset line (15). The counter is advanced on the positive edge of the clock signal (pin 14), when the clock inhibit pin (13) is grounded. The 10 decoded outputs are normally in the logical “0” state and go to the logical “1” state only at their respective time slot. Each decoded output remains high for 1 full clock cycle. The carry-out signal completes a full cycle for every 10 clock input cycles and is used as a ripple carry signal to any succeeding stages. The 8 rows of LED matrix are sequentially connected to the decoded outputs, Q0- Q7, of CD4017 through ULN2803 IC that has eight Darlington pairs, each of which provides a ground path to sink the combined current of all LEDs in a row. At the end of every 8th clock cycle, the microcontroller will reset the counter by issuing a logical “1″ to its Reset pin (15).