# MODULE 4
# INTERFACING SUB SYSTEMS WITH AVR

**SYLLABUS**

Basics of Serial Communication – ATMEGA32 connection to RS232 - AVR serial port programming in C- LCD interfacing - Keyboard interfacing - ADC interfacing -- DAC interfacing - Sensor interfacing

## BASICS OF SERIAL COMMUNICATION

Computers transfer data in two ways: parallel and serial. In parallel data transfers, often eight or more lines (wire conductors) are used to transfer data to a device that is only a few feet away. Devices that use parallel transfers include printers and IDE hard disks; each uses cables with many wires. Although a lot of data can be transferred in a short amount of time by using many wires in parallel,the distance cannot be great. To transfer to a device located many meters away, the serial method is used. In serial communication, the data is sent one bit at a time,in contrast to parallel communication, in which the data is sent a byte or more at a time.

- When a microprocessor communicates with the outside world, it provides the data in byte-sized chunks.
- For some devices, such as printers, the information is simply grabbed from the 8-bit data bus and presented to the 8-bit data bus of the device. This can work only if the cable is not too long, because long cables diminish and even distort signals. Furthermore, an 8-bit data path is expensive.
- For these reasons, serial communication is used for transferring data between two systems located at distances of hundreds of feet to millions of miles apart.
- The fact that a single data line is used in serial communication instead of the 8-bit data line of parallel communication.

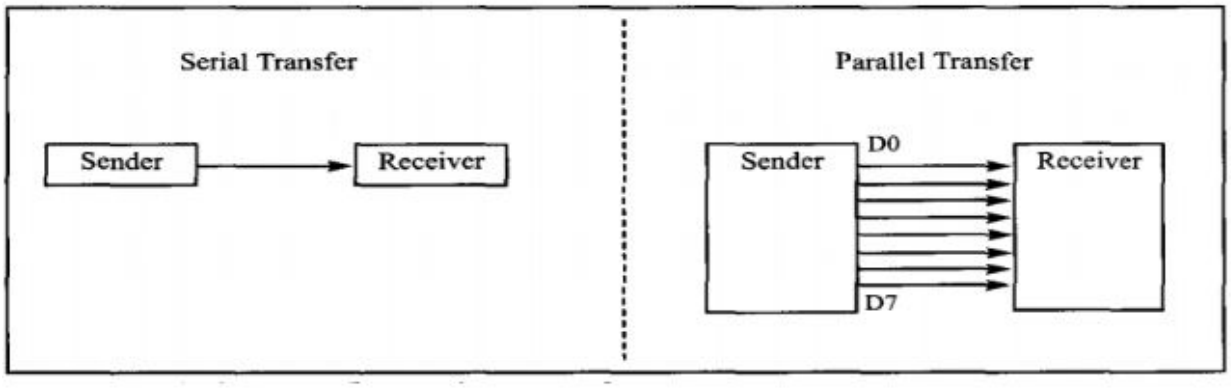



**Figure : Serial versus Parallel Data Transfer**

- For serial data communication to work, the byte of data must be converted to serial bits using a parallel-in-serial-out shift register; then it can be transmitted over a single data line.
- This also means that at the receiving end there must be a serial-in-parallel-out shift register to receive the serial data and pack them into a byte.
- Of course, if data is to be transmitted on the telephone line, it must be converted from Os and Is to audio tones, which are sinusoidal signals.
- This conversion is performed by a peripheral device called a **modem,** which stands for "**modulator/demodulator."**

Asynchronous and Synchronous

- ❖ Serial data communication uses two methods, asynchronous and synchronous.
- ❖ The synchronous method transfers a block of data (characters) at a time, whereas the asynchronous method transfers a single byte at a time.
- ❖ Special IC chips are made by many manufacturers for serial data communications.
- ❖ A serial data communications. These chips are commonly referred to as UART (universal asynchronous receiver-transmitter) and USART (universal synchronous asynchronous receiver-transmitter).

**Half- and full-duplex transmission**

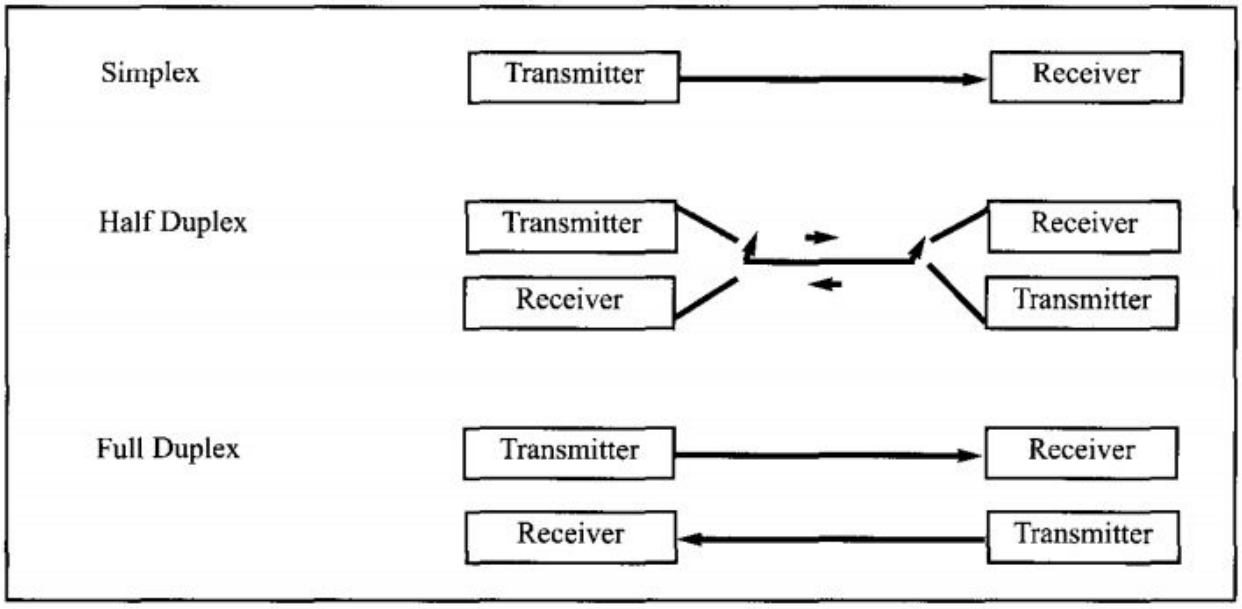



**Figure : Simplex, Half-, and Full-Duplex Transfers**

- In data transmission, if the data can be both transmitted and received, it is a duplex transmission.
- This is in contrast to simplex transmissions such as with printers, in which the computer only sends data.
- Duplex transmissions can be half or full duplex, depending on whether or not the data transfer can be simultaneous.
- If data is transmitted one way at a time, it is referred to as half duplex.
  can go both ways at the same time, it is full duplex. Of course, full duplex requires two wire conductors for the data lines (in addition to the signal ground),
- one for one for transmission and one for reception, in order to transfer and receive data simultaneously.

**Asynchronous serial communication and data framing**

The data coming in at the receiving end or the data line in a serial data transfer is all Os and Is; it is difficult to make sense of the data unless the sender and receiver agree on a set of rules, a *protocol*, on how the data is packed, how many bits constitute a character, and when the data begins and ends.

**Start and stop bits**

- **Asynchronous** serial data communication is widely used for ***character oriented transmissions.***
- while ***block-oriented*** data transfers use the **synchronous** method.
- In the asynchronous metho,each character is placed between **start and stop bits**. This is called **framing**.

- In data framing for asynchronous communications, the data, such as ASCII characters, are packed between a start bit and a stop bit. The start bit is always one bit, but the stop bit can be one or two bits.

- The start bit is always a 0 (low), and the stop bit(s) is 1 (high).

- For example, look at Figure 11-3 in which the ASCII character "A" (8-bit binary 0100 0001) is framed between the start bit and a single stop bit. Notice that the LSB is sent out first.

- when there is no transfer, the signal is 1 (high),which is referred to as mark. The 0 (low) is referred to as space. Notice that the transmission begins with a start bit (space) followed by DO, the LSB, then the rest of the bits until the MSB (D7), and finally, the one stop bit indicating the end of character “A”.

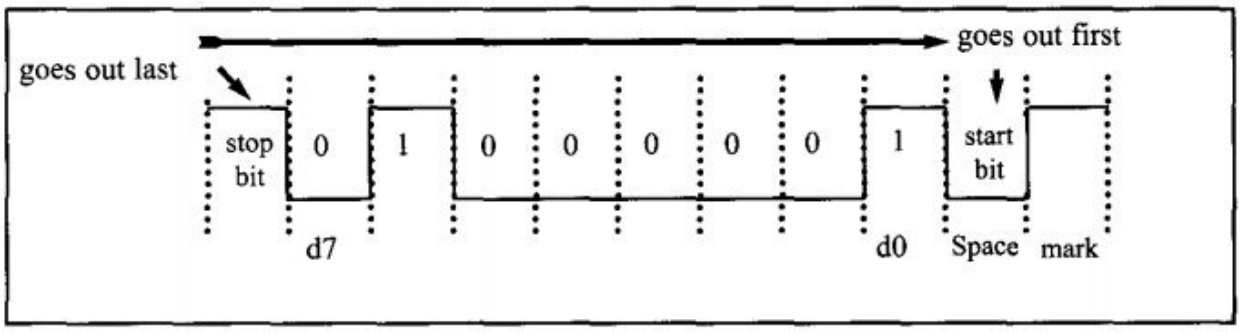



**Figure:Framing ASCII 'A' ( 4 lH)**

**Data transfer rate**

- The rate of data transfer in serial data communication is stated in bps (bits per second).

- Another widely used terminology for bps is baud rate. However, the baud and bps rates are not necessarily equal.
- This is because baud rate is the modem terminology and is defined as the number of signal changes per second.
- In modems, sometimes a single change of signal transfers several bits of data. As far as the conductor wire is concerned, the baud rate and bps are the same.
- The data transfer rate of a given computer system depends on communication ports incorporated into that system. For example, the early IBM PC/XT could transfer data at the rate of 100 to 9600 bps.

**RS232 standards**

- RS232 is one of the most widely used ***serial I/O interfacing standards.*** This standard is used in PCs and numerous types of equipment.
- Because the standard was set long before the advent of the TTL logic family, however, its input and output voltage levels are not TTL compatible.
- In RS232, a 1 is represented by -3 to -25 V, while a 0 bit is + 3 to + 25 volts, making -3 to +3 undefined.
- For this reason, to connect any RS232 to a microcontroller system we must use **voltage converters** such as MAX232 to convert the TTL logic levels to the RS232 voltage levels, and vice versa.
- MAX232 IC chips are commonly referred to as **line drivers.**

**ATMEGA32 CONNECTION TO RS232**

In this section, the details of the physical connections of the ATmega32 to
RS232 connectors are given. The RS232 standard is not TTL compatible; therefore, a line driver such as the MAX232 chip is required to convert RS232 voltage levels to TTL levels, and vice versa. The interfacing of ATmega32 with RS232 connectors via the MAX232 chip is shown below.

**RX and TX pins in the ATmega32**

- The ATmega32 has two pins that are used specifically for transferring and receiving data serially.
- These two pins are called TX and RX and are part of the Port D group (PDO and PDl) of the 40-pin package.

- Pin 15 of the ATmega32 is assigned to TX and pin 14 is designated as RX. These pins are TTL compatible; therefore, they require a line driver to make them RS232 compatible. One such line driver is the MAX232 chip.

**MAX232**

- Because the RS232 is not compatible with today's microprocessors and microcontrollers, we need a line driver (voltage converter) to convert the RS232's signals to TTL voltage levels that will be acceptable to the AVR's TX and RX pins.
- The MAX232 converts from RS232 voltage levels to TTL voltage levels, and vice versa. One advantage of the MAX232 chip is that it uses a +5 V power source, which is the same as the source voltage for the AVR.

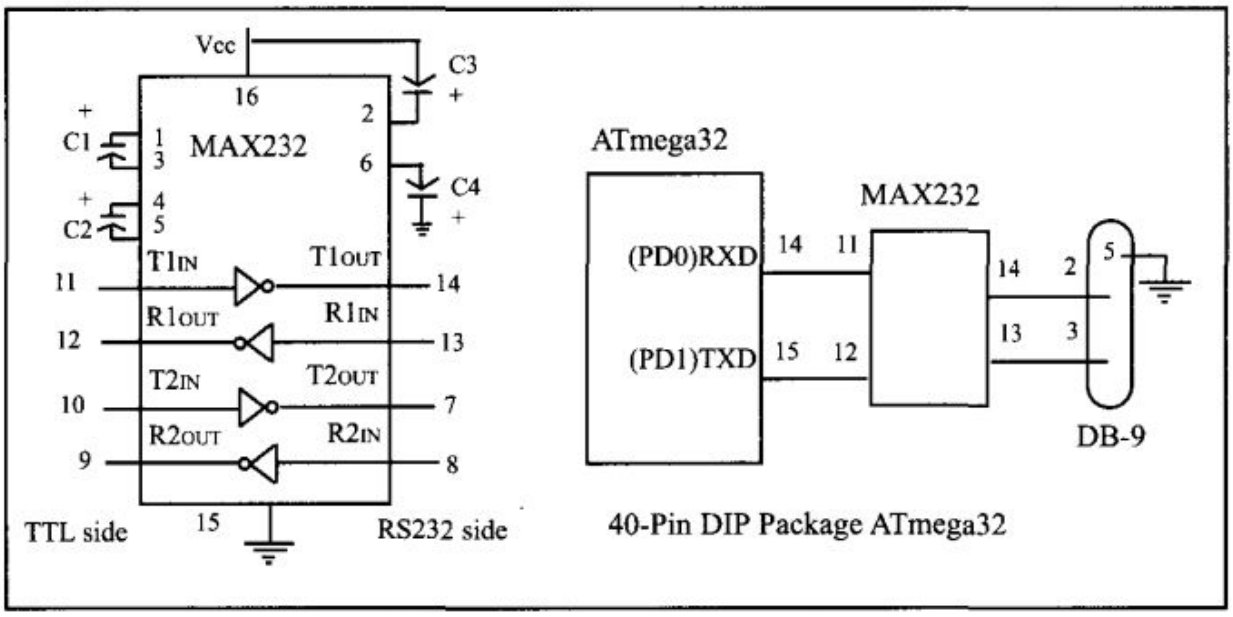



**Figure (a) Inside MAX232 and Figure (b) Its Connection to the ATmega32 (Null Modem).**

- The MAX232 has two sets of line drivers for transferring and receiving data.
- The line drivers used for TX are called Tl and T2,while the line drivers for RX are designated as Rl and R2.
- In many applications only one of each is used. For example, Tl and Rl are used together for TX and RX of the AVR, and the second set is left unused.

- Notice in MAX232 that the Tl line driver has a designation of T1 in and TI out on pin numbers 11 and 14, respectively. The T1 in pin is the TTL side and is connected to TX of the microcontroller,
- while Tl out is the RS232 side that is connected to the RX pin of the RS232 DB connector.
- The RI line driver has a designation of Rlin and Rlout on pin numbers 13 and 12, respectively. The Rlin (pin 13) is the RS232 side that is connected to the TX pin of the RS232 DB connector, and Rlout (pin 12) is the TTL side that is connected to the RX pin of the microcontroller.

- MAX232 requires four capacitors ranging from 0.1 to 22 μF. The most widely used value for these capacitors is 22 μF.

## AVR SERIAL PORT PROGRAMMING IN C

All the special function registers of the AVR are accessible directly in C compilers by using the appropriate header file.

**Example 11-9**

Write a C function to initialize the USART to work at 9600 baud, 8-bit data, and 1 stop bit. Assume XTAL = 8 MHz.

**Solution:**

```
void usart_init (void)
{
   UCSRB = (1<<TXEN);
   UCSRC = (1<< UCSZ1)|(1<<UCSZ0)|(1<<URSEL);
   UBRRL = 0x33;
}
```

**Example 11-10** (C Version of Example 11-4)

Write a C program for the AVR to transfer the letter 'G' serially at 9600 baud, continuously. Use 8-bit data and 1 stop bit. Assume XTAL = 8 MHz.

**Solution:**

```
#include <avr/io.h>                          //standard AVR header
void usart_init (void)
{
   UCSRB = (1<<TXEN);
   UCSRC = (1<< UCSZ1)|(1<<UCSZ0)|(1<<URSEL);
   UBRRL = 0x33;
}
void usart_send (unsigned char ch)
{                                            //wait until UDR
   while (! (UCSRA & (1<<UDRE)));            //is empty
   UDR = ch;                                 //transmit 'G'
}

int main (void)
{
   usart_init();                             //initialize the USART
   while(1)                                  //do forever
     usart_send ('G');                       //transmit 'G' letter
   return 0;
}
```

## LCD INTERFACING

In recent years the LCD is finding widespread use replacing LEDs (seven-segment LEDs or other multi segment LEDs). This is due to the following reasons:

I. The declining prices of LCDs.
2. The ability to display numbers, characters, and graphics. This is in contrast to LEDs, which are limited to numbers and a few characters.
3. Incorporation of a refreshing controller into the LCD, thereby relieving the CPU of the task of refreshing the LCD. In contrast, the LED must be refreshed by the CPU (or in some other way) to keep displaying the data.
4. Ease of programming for characters and graphics.

**LCD operation**

LCD pin descriptions:

The function of each pin is given in Table

| Pin | Symbol | I/O | Description |
|---|---|---|---|
| 1 | $V_{SS}$ | -- | Ground |
| 2 | $V_{CC}$ | -- | +5 V power supply |
| 3 | $V_{EE}$ | -- | Power supply to control contrast |
| 4 | RS | I | RS = 0 to select command register, RS = 1 to select data register |
| 5 | R/W | I | R/W = 0 for write, R/W = 1 for read |
| 6 | E | I/O | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

**figure:Pin Descriptions for LCD**

**Vcc. Vss, and Vee**

While V cc and V ss provide +5 V and ground, respectively, VEE is used for controlling LCD contrast.

**RS, register select**

- There are two very important registers inside the LCD.
- The RS pin is used for their selection as follows.

- If RS = 0, the instruction ***command code register*** is selected,allowing the user to send commands such as *clear display, cursor at home*, and so on.
- If RS = 1 the ***data register*** is selected, allowing the user to send data to be displayed on the LCD**.**

**R/w, read/write**

- R/W input allows the user to write information to the LCD or read information from it.
- R/W = 1 when reading; R/W = 0 when writing.

**E, enable**

- The enable pin is used by the LCD to latch information presented to its data pins.
- When data is supplied to data pins, a high-to-low pulse must be applied to this pin in order for the LCD to latch in the data present at the data pins.
- This pulse must be a minimum of 450 ns wide.

**DO-D7**

- The 8-bit data pins,DO-D7, are used to send information to the LCD or read the contents of the LCD's internal registers.
- To display letters and numbers, we send ASCII codes for the letters A-Z, a-z, and numbers 0-9 to these pins while making RS = I.
- There are also instruction command codes that can be sent to the LCD to clear the display or force the cursor to the home position or blink the cursor.
- In this section you will see how to interface an LCD to the AYR in two different ways. We can use 8-bit data or 4-bit data options. The 8-bit data interfacing is easier to program but uses 4 more pins.

**LCD Command Codes**

| Code (Hex) | Command to LCD Instruction Register |
|---|---|
| 1 | Clear display screen |
| 2 | Return home |
| 4 | Decrement cursor (shift cursor to left) |
| 6 | Increment cursor (shift cursor to right) |
| 5 | Shift display right |
| 7 | Shift display left |
| 8 | Display off, cursor off |
| A | Display off, cursor on |
| C | Display on, cursor off |
| E | Display on, cursor blinking |
| F | Display on, cursor blinking |
| 10 | Shift cursor position to left |
| 14 | Shift cursor position to right |
| 18 | Shift the entire display to the left |
| 1C | Shift the entire display to the right |
| 80 | Force cursor to beginning of 1st line |
| C0 | Force cursor to beginning of 2nd line |
| 28 | 2 lines and 5 × 7 matrix (D4–D7, 4-bit) |
| 38 | 2 lines and 5 × 7 matrix (D0–D7, 8-bit) |

**Sending commands and data to LCDs**

To send data and commands to LCDs you should do the following steps.
Notice that steps 2 and 3 can be repeated many times:

1. Initialize the LCD.
2. Send any of the commands from Table 12-2 to the LCD.
3. Send the character to be shown on the LCD.

**Initializing the LCD**

- To initialize the LCD for 5 x 7 matrix and 8-bit operation, the following sequence of commands should be sent to the LCD: 0x38, 0x0E, and 0x01.

- Next we will show how to send a command to the LCD. After power-up you should wait about 15 ms before sending initializing commands to the LCD. If the LCD initializer function is not the first function in your code you can omit this delay.

**Sending commands to the LCD**

- To send any of the commands to the LCD, make pins RS and R/W = 0 and put the command number on the data pins (DO--D7).
- Then send a high-to-low pulse to the E pin to enable the internal latch of the LCD. Notice that after each command you should wait about 100 μs to let the LCD module run the command.
- Clear LCD and Return Home commands are exceptions to this rule.

**Sending data to the LCD**

- To send data to the LCD, make pins RS= 1 and R/W = 0. Then put the data on the data pins (DO-D7) and send a high-to-low pulse to the E pin to enable the internal latch of the LCD.
- Notice that after sending data you should wait about 100 μs to let the LCD module write the data on the screen.

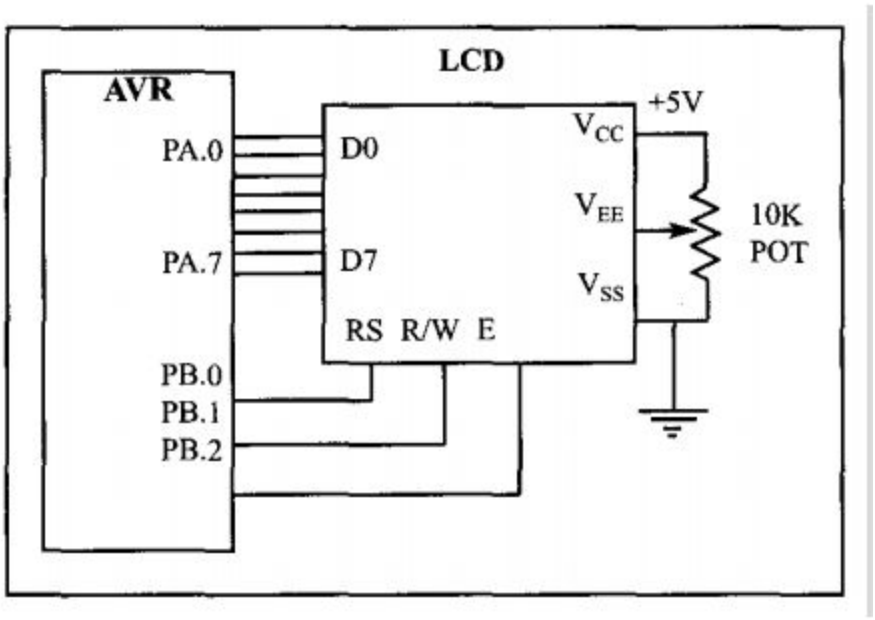



Figure : LCD Connections for 8-bit Data

*(Please refer to the lab Record for LCD interfacing programs.)*

## KEYBOARD INTERFACING

Keyboards and LCDs are the most widely used input/output devices in microcontrollers such as the AVR and a basic understanding of them is essential.

### Interfacing the keyboard to the AVR

- At the lowest level, keyboards are organized in a matrix of rows and Columns.
- The CPU accesses both rows and columns through ports; therefore, with two 8-bit ports, an 8 x 8 matrix of keys can be connected to a microcontroller.
- When a key is pressed, a row and a column make a contact; otherwise, there is no connection between rows and columns.
- In x86 PC keyboards, a single microcontroller takes care of hardware and software interfacing of the keyboard.
- In such systems, it is the function of programs stored in the Flash of the microcontroller to scan the keys continuously, identify which one has been activated, and present it to the motherboard.

### Scanning and identifying the key

- Figure below shows a 4 x 4 matrix connected to two ports.
- The rows are connected to an output port and the columns are connected to an input port.
- If no key has been pressed, reading the input port will yield 1s for all columns since they are all connected to high (VCC).
- If all the rows are grounded and a key is pressed,one of the columns will have 0 since the key pressed provides the path to ground.
- It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed.

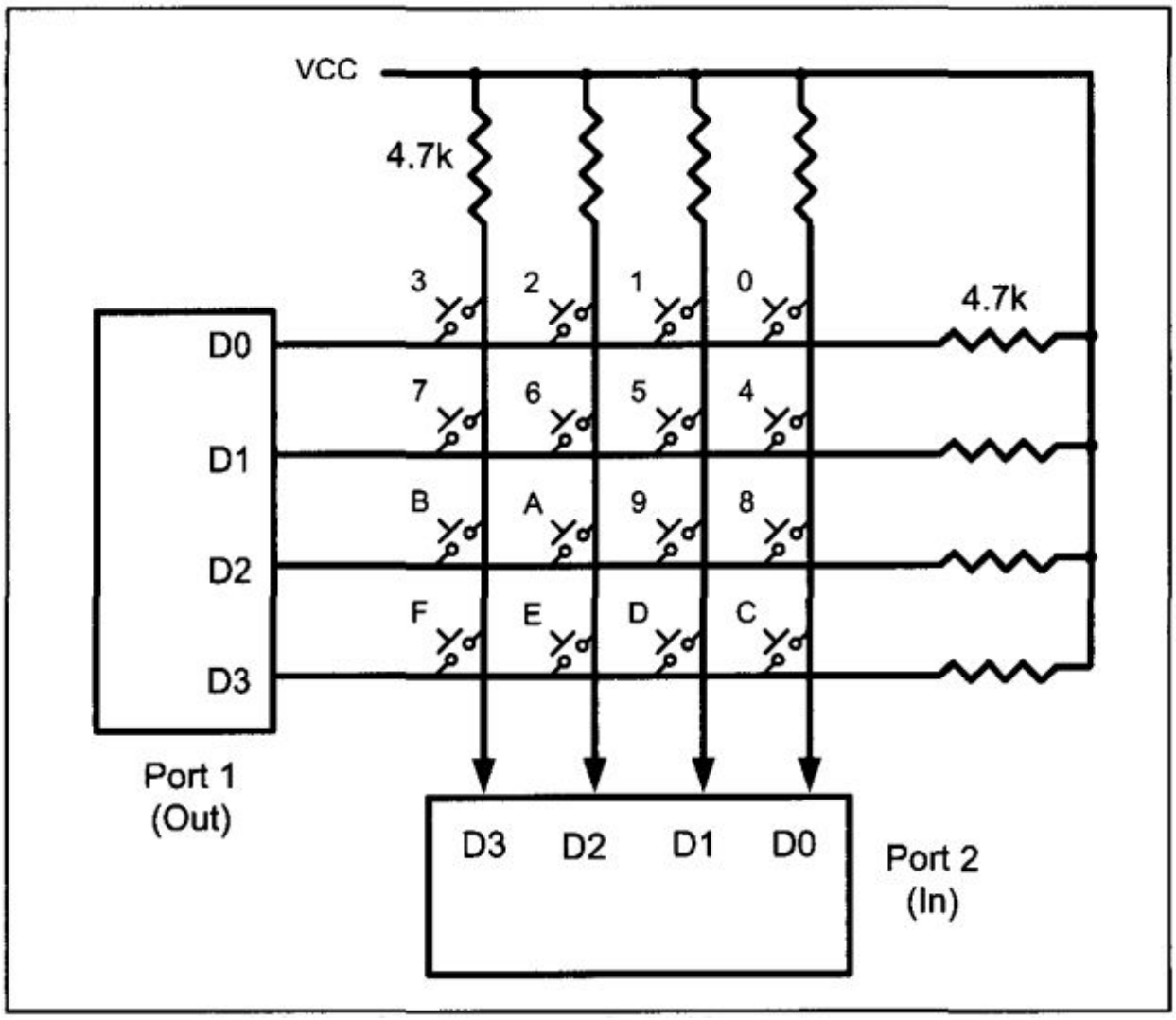


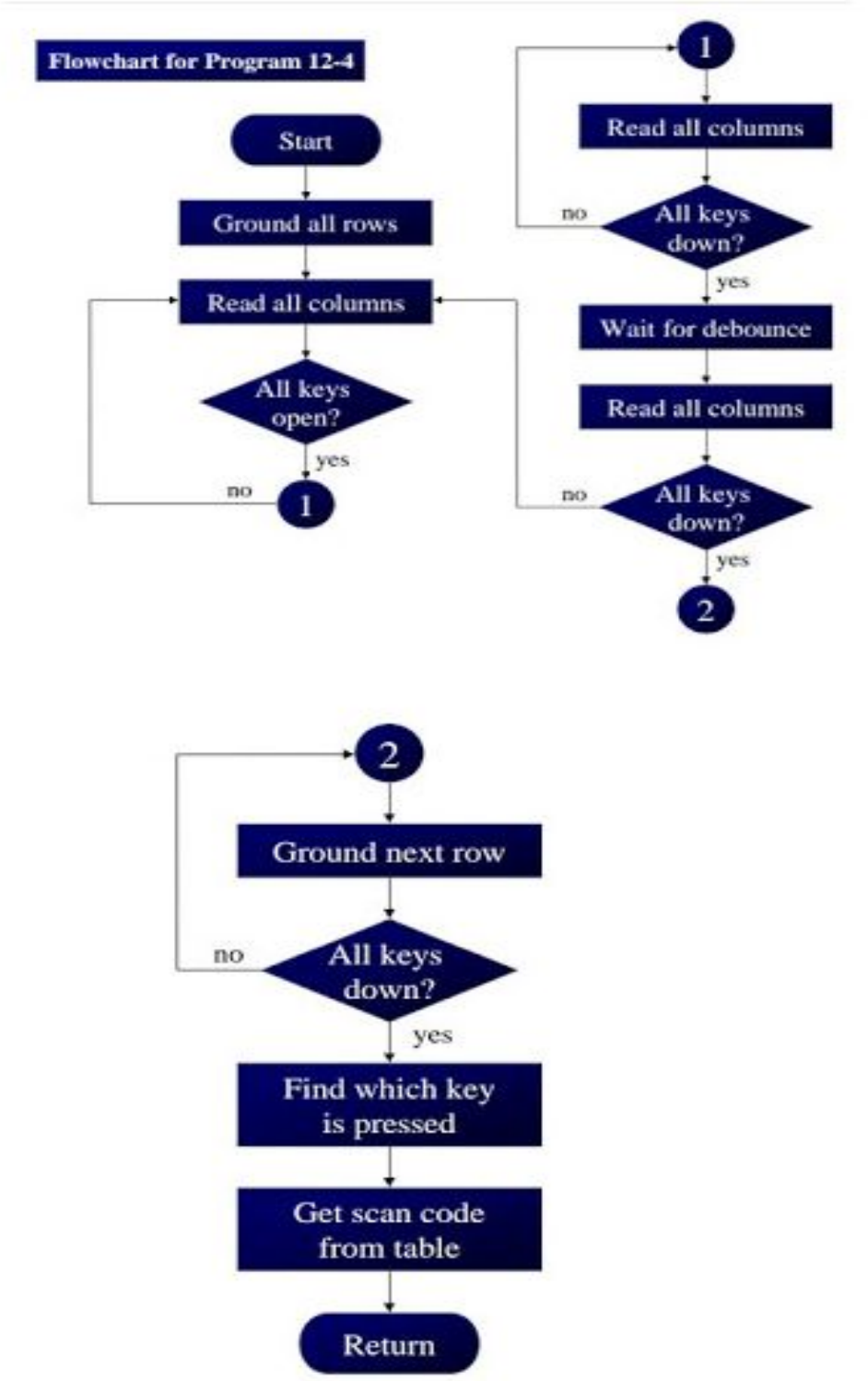

***Figure : Matrix Keyboard Connection to Ports***

**Grounding rows and reading the columns**

- To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch, and then it reads the columns.
- If the data read from the columns is D3-DO = 1111, no key has been pressed and the process continues until a key press is detected.
- However, if one of the column bits has a zero, this means that a key press has occurred.
- For example, if D3-DO = 1101, this means that a key in the DI column has been pressed.
- After a key press is detected, the microcontroller will go through the process of identifying the key.

- Starting with the top row,the microcontroller grounds it by providing a low to row DO only; then it reads the columns.
- If the data read is all Is, no key in that row is activated and the process is moved to the next row. It grounds the next row, reads the columns, and checks for any zero.
- This process continues until the row is identified.
- After identification of the row in which the key has been pressed, the next task is to find out which column the pressed key belongs to. This should be easy since the microcontroller knows at any time which row and column are being accessed.

**Flowchart for Program 12-4**




(*please refer to the lab record for keyboard interfacing program)*

## ADC INTERFACING

### ADC devices

- Analog-to-digital converters are among the most widely used devices for data acquisition.
- Digital computers use binary (discrete) values, but in the physical world everything is analog (continuous). Temperature, pressure (wind or liquid), humidity, and velocity are a few examples of physical quantities that we deal with every day.
- *A physical quantity is converted to electrical (voltage, current) signals using a device called a* ***transducer****.*
- Transducers are also referred to as sensors.Sensors for temperature, velocity, pressure, light, and many other natural quantities produce an output that is voltage (or current).
- Therefore, we need an analog-to-digital converter to translate the analog signals to digital numbers so that the microcontroller can read and process them.

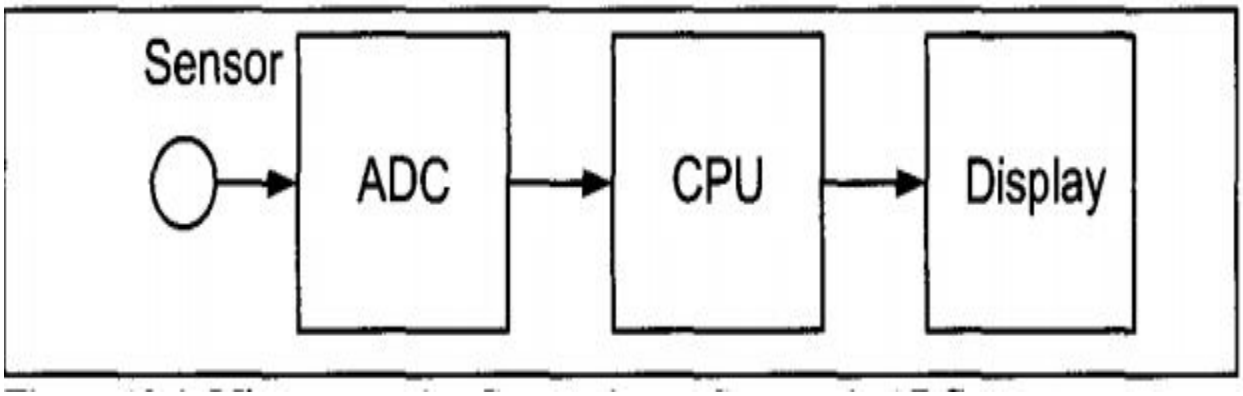



***Figure : Microcontroller Connection to Sensor via ADC***

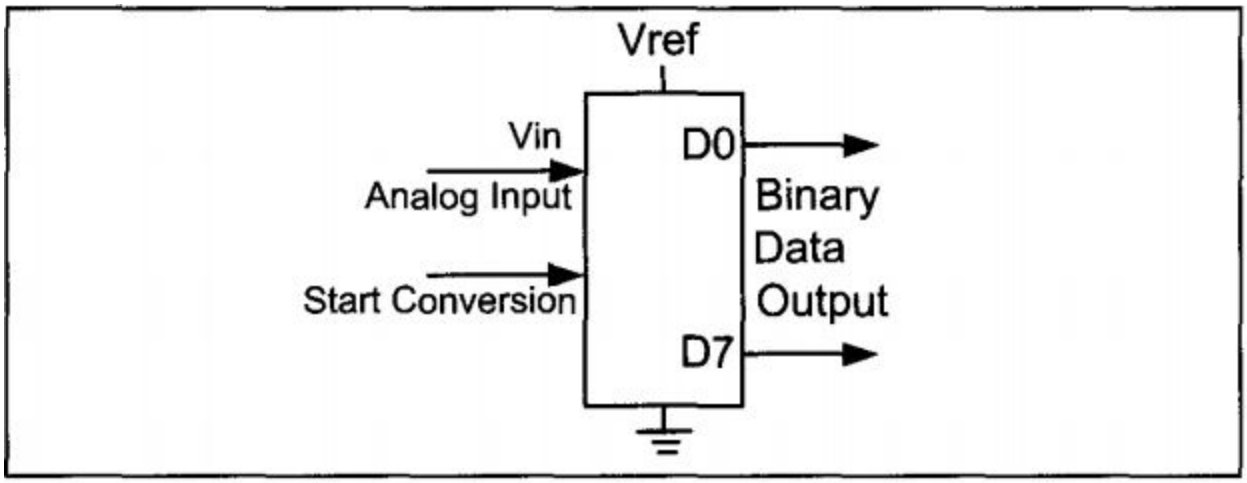



***Figure : An 8-bit ADC Block Diagram***

**Some of the major characteristics of the ADC**

**Resolution**

- The ADC has n-bit resolution, where n can be 8, 10, 12, 16, or even 24 bits.
- Higher-resolution ADCs provide a smaller step size, where step size is the smallest change that can be discerned by an ADC.
- Although the resolution of an ADC chip is decided at the time of its design and cannot be changed, we can control the step size with the help of what is called Yref.

**Conversion time**

- conversion time is another major factor in judging an ADC.
- Conversion time is defined as the time it takes the ADC to convert the analog input to a digital (binary) number.
- The conversion time is dictated by the clock source connected to the ADC.

**Vref**

- Vref is an input voltage used for the reference voltage.
- The voltage connected to this pin, along with the resolution of the ADC chip, dictate the step size.
- For an 8-bitADC, the step size is Vref 256 because it is an 8-bitADC, and 2 to the power of 8 gives us 256 steps.

**Digital data output**

- In an 8-bit ADC we have an 8-bit digital data output of DO-D7,
- while in the 10-bit ADC the data output is DO-D9.
- To calculate the output voltage, we use the following formula:

$$D_{out} = \frac{V_{in}}{step\ size}$$

- where Dout = digital data output (in decimal), Vin = analog input voltage, and step size (resolution) is the smallest change, which is Vref'256 for an 8-bitADC.

**ADC PROGRAMMING IN THE AVR**

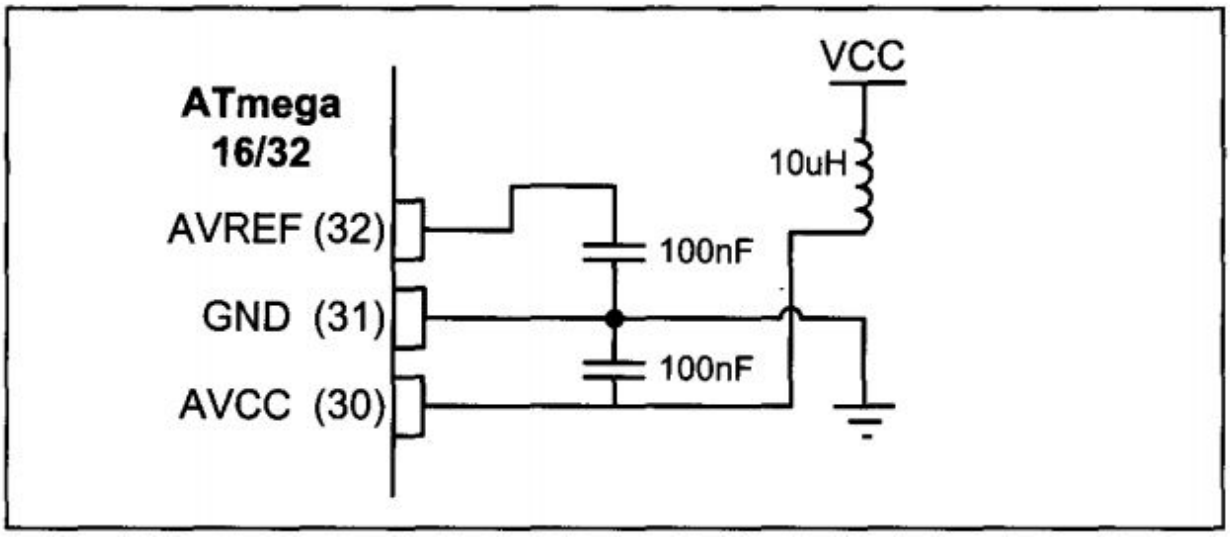
Because the ADC is widely used in data acquisition, in recent years an increasing number of microcontrollers have had an on-chip ADC peripheral, just like timers and USART. An on-chip ADC eliminates the need for an external ADC

connection, which leaves more pins for other I/O activities. The vast majority of AVR chips come with ADC.

**ATmega32 ADC features**

The ADC peripheral of the ATmega32 has the following characteristics:
(a) It is a 10-bitADC.
(b) It has 8 analog input channels, 7 differential input channels, and 2 differential input channels with optional gain of I Ox and 200x.
( c) The converted output binary data is held by two special function registers called ADCL (ND Result Low) and ADCH (ND Result High).
(d) Because the ADCH:ADCL registers give us 16 bits and the ADC data out is only 10 bits wide, 6 bits of the 16 are unused. We have the option of making either the upper 6 bits or the lower 6 bits unused.
(e) We have three options for Vref- Vref can be connected to AVCC (Analog V cc), internal 2.56 V reference, or external AREF pin.
( f) The conversion time is dictated by the crystal frequency connected to the XTAL

**AVR ADC hardware considerations**




**Decoupling AVCC from VCC**

The AVCC pin provides the supply for analog ADC circuitry. To get a better accuracy of AVR ADC we must provide a stable voltage source to the AVCC pin. Figure above shows how to use an inductor and a capacitor to achieve this.

**Connecting a capacitor between Vref and GND**

By connecting a capacitor between the AVREF pin and GND you can make the Vref voltage more stable and increase the precision of ADC.

**SENSOR INTERFACING**

**Temperature sensors**

- Transducers convert physical data such as temperature, light intensity,flow, and speed to electrical signals.
- Depending on the transducer, the output produced is in the form of voltage, current, resistance, or capacitance. For example,temperature is converted to electrical signals using a transducer called a thermistor.
- A thermistor responds to temperature change by changing resistance, its response is not linear.
- The complexity associated with writing software for such nonlinear devices has led many manufacturers to market a linear temperature sensor.
- Simple and widely used linear temperature sensors include the LM34 and LM35 series from National Semiconductor Corp.

**LM34 and LM35 temperature sensors**

The sensors of the LM34 series are precision integrated-circuit temperature sensors whose output voltage is linearly proportional to the Fahrenheit temperature.
The LM35 series sensors are precision integrated-circuit temperature sensors whose output voltage is linearly proportional to the Celsius (centigrade) temperature.

**Signal conditioning**

- Signal conditioning is widely used in the world of data acquisition.
- The most common transducers produce an output in the form of voltage, current, charge, capacitance, and resistance. We need to convert these signals to voltage.
- In order to send input to an A-to-D converter. This conversion (modification) is commonly called signal conditioning.

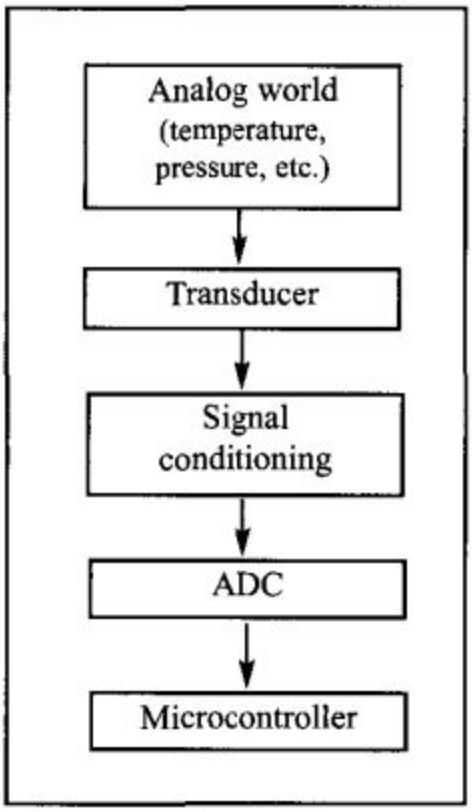



*Figure:Getting Data from the Analog World*

- Signal conditioning can be current-to-voltage conversion or signal amplification. For example, the thermistor changes resistance with temperature. The change of resistance must be translated into voltages to be of any use to an ADC.

**Interfacing the LM34 to the AVR**

- The AID has I 0-bit resolution with a maximum of I 024 steps.
- LM34 (or LM35) produces 10 mV for every degree of temperature change.
- Now,if we use the step size of 10 mV, the Vout will be 10,240 mV (10.24 V) for full scale output. This is not acceptable even though the maximum temperature sensed by the LM34 is 300 degrees F, and the highest output we will get for the AID is 3000 mV (3.00 V).

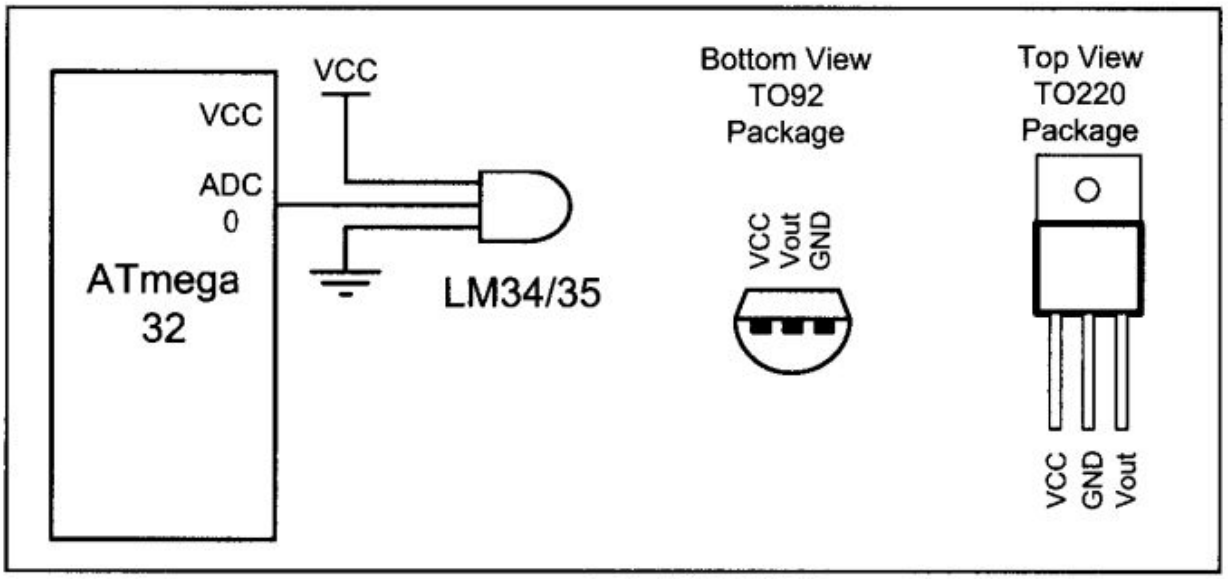



***Figure : Connection to AVR and Its Pin Configuration***

## DAC INTERFACING

Digital-to-analog converter (DAC)

- The digital-to-analog converter (DAC) is a device widely used to convert digital pulses to analog signals.
- In this section we discuss the basics of interfacing a DAC to the AVR.
- The two methods of creating a DAC: binary weighted and R/2R ladder. The vast majority of integrated circuit DACs, including the MC1408 (DAC0808) used in this section,
- use the R/2R method because it can achieve a much higher degree of precision.

## MC1408 DAC (or DAC0808)

- In the MC 1408 (DAC0808), the digital inputs are converted to current Iout), and by connecting a resistor to the lout pin.we convert the result voltage.
- The total current provided by the lout pin is a function of the binary numbers at the DO-D7 inputs of the DAC0808 and the reference current (I,0r), and is as follows:

$$I_{out} = I_{ref}\left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256}\right)$$

- where DO is the LSB, D7 is the MSB for the inputs,

- Iref is the input current that must be applied to pin 14.
- The Iref current is generally set to 2.0 mA.

**Converting Iout to voltage in DAC0808.**

- Ideally we connect the output pin Iout to a resistor, convert this current to voltage, and monitor the output on the scope.
- In real life, however, this can cause inaccuracy because the input resistance of the load where it is connected will also affect the output voltage. For this reason, the Iref current output is isolated by connecting it to an op-amp such as the 7 41 with Rf= 5 kilohms for the feedback resistor. Assuming that R = 5 kilohms, by changing the binary input, the output voltage

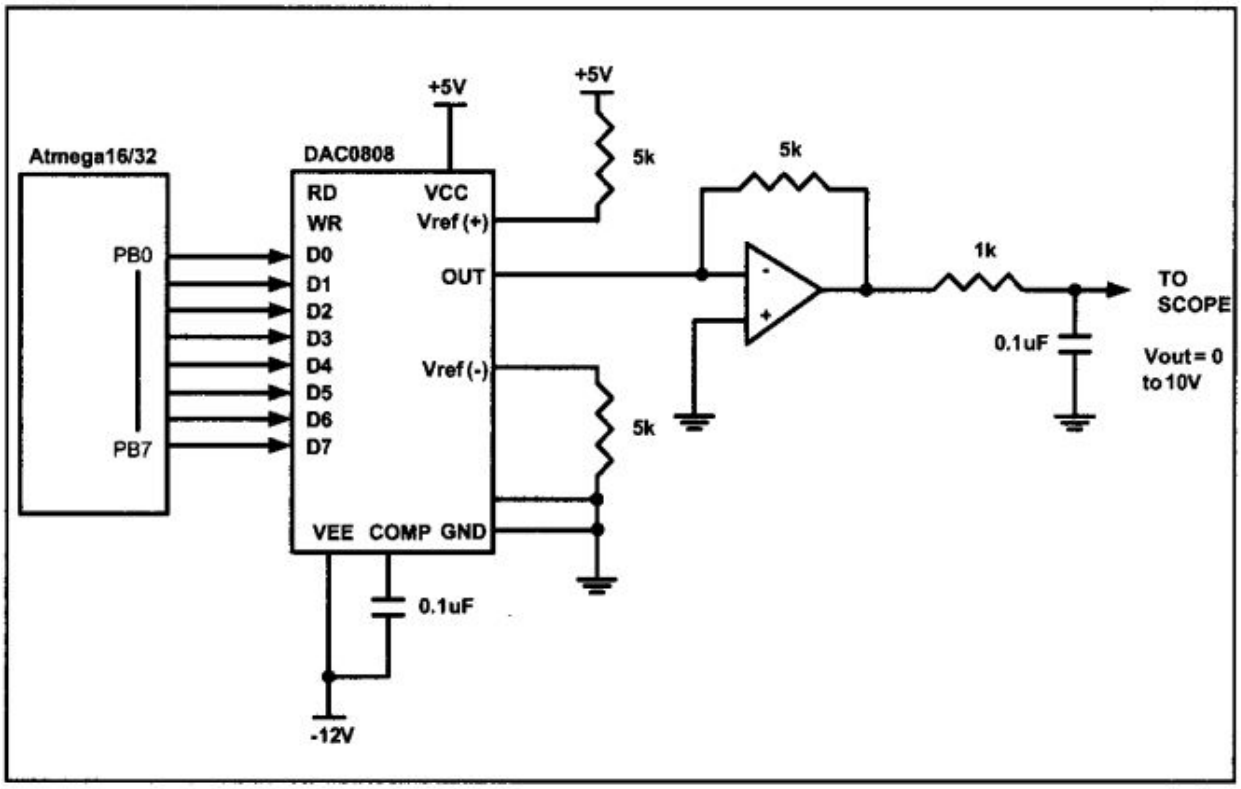



*Figure : AVR Connection to DAC0808*