UNIT V
INTERFACING MICROCONTROLLERS

Programming 8051 Timers - Serial Port Programming - Interrupts Programming – LCD &Keyboard Interfacing - ADC, DAC & Sensor Interfacing - External Memory Interface-Stepper Motor and Waveform generation.

5.1 PROGRAMMING 8051 TIMERS

The 8051 has two times / counters. They can be used either as timers to generate a time delay or as counters to count events happening outside the micro-controller.The two timers in 8051 are :

- Timer 0 and
- Timer 1.

They can be used either as timers or as event counters.

5.1.1 Basic registers of the timer

Both Timer 0 and Timer 1 are 16 wide. Since the 8051 has an 8-bit architecture, each 16-bit timer is accessed as two separate registers of low byte and high byte.

Timer 0 registers

The 16-bit register of Timer 0 is accessed as low byte and high byte. The low byte register is called TL0 (Timer 0 low byte) and the high byte register is referred to as TH0 (Timer 0 high byte). These registers can be accessed like any other register, such as A, B, R0, R1, R2, etc. For example, the instruction “MOV TL0, #4FH” moves the value 4FH into TL0, the low byte of Timer 0.

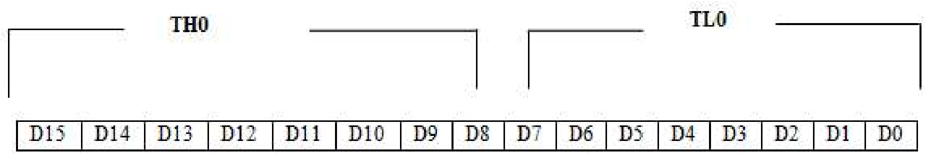


Fig. 5.1Timer 0 registers

Timer 1 registers

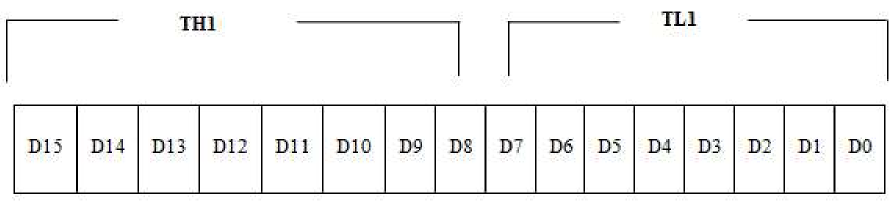


Fig. 5.2 Timer 1 registers

Timer 1 is also 16 bits, and its 16 bit register is split into two bytes, referred to as TL1 (Timer 1 low byte) and TH1 (Timer 1 high byte). These registers are accessible in the same way as the registers of Timer 0.

5.1.2 TMOD (timer mode) register

Both timers 0 and 1 use the same register, called TMOD, to set the various timer operation modes, TMOD is an 8 bit register in which the lower 4 bits are set aside for Timer 0 and the upper 4 bits for Timer 1. In each case, the lower 2 bits are used to set the timer a mode and the upper 2 bits to specify the operation.

(MSB) (LSB)

GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
Timer1				Timer 0			

Fig 5.3: TMOD Register

GATE:

When GATE = 0, the start and stop of timer are controlled by software by means of TR (timer start) bits TR0 and TR1. This is achieved by the instructions “SETB TR1” and “CLR TR1” for Timer 1, and “SETB TR0” and CLR instruction.

When GATE = 1, the start and stop of timer are controlled by hardware.

C/$\overline{T}$:

This bit in the TMOD register is used to decide whether the timer is used as a delay generator or an event counter. If C/$\overline{T}$ = 0, it is used as a timer for time the 8051. If C/$\overline{T}$ = 1, it is used as a counter.

M1,M0 :

M1- Mode bit 0, M2- Mode bit 0

M1	M0	Mode	Operating Mode
0	0	1	13 bit timer mode
0	1	1	16 bit timer mode
1	0	2	8 bit timer with auto reload
1	1	3	Split timer mode

5.1.3 Timer operating modes:

Mode 0:

In this mode, TL0 and Th0 for timer 0 are used as 13 bit register, i.e, all the 8 bits of TL0 and 5 bits in TH0. The 13-bit counter can hold values between 0000 to 1FFFH in TH - TL. Therefore, when the timer reaches its maximum of 1FFH, it rolls over to 0000, and TF is raised.

If C/$\overline{T}$=0, then the register is incremented after every machine cycle, i.e, at the rate of 1/12th of the oscillator frequency. For the register to get incremented the control switch should be closed. This switch is logic controlled and there are many ways by which this logic can be implemented. Some of them are:

- Case 1
  TR0 = 1 (high)
  Gate = 0 (low) and $\overline{INTO}$ = 0 (low)

- Case 2
  TR0 = 1 (high)
  Gate = 0 (high) and $\overline{INT0}$ = 1 (high)

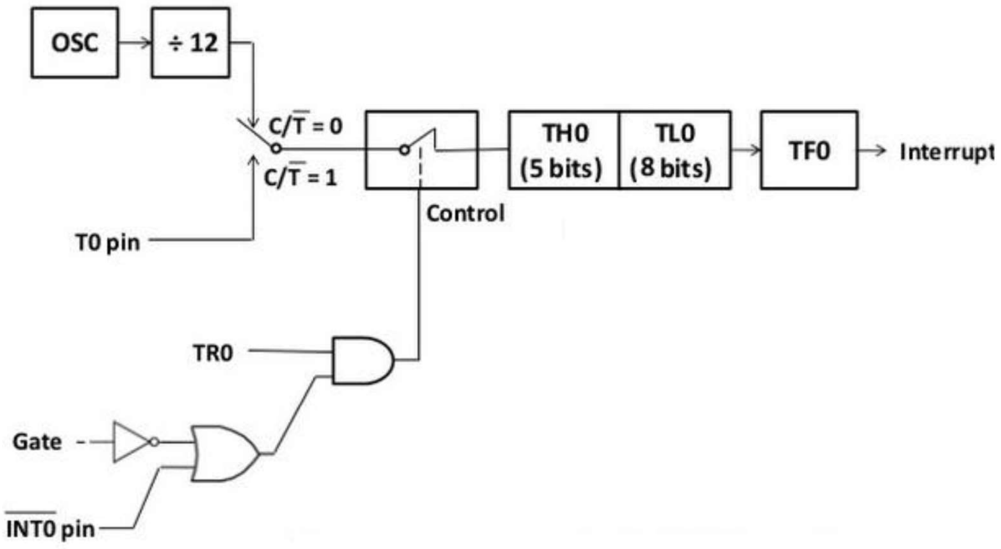


Fig 5.4: Mode 0

Mode 0 programming:

The following are the characteristics and operations of mode 0 :

1. It is a 13-bit timer; therefore, it allows values of 0000 to 1FFFH to be loaded into the timer's registers TL and TH.
2. After TH and TL are loaded with a 13-bit initial value, the timer must be started. This is done by "SETB TR0" for Timer 0 and :SETB TR1" for Timer 1.
3. After the timer is started, it starts to count up. It counts up until it reaches its limit of 1FFFH. When it rolls over from 1FFFH to 0000, it sets high a flag bit called TF (timer flag). This timer flag can be monitored. When this timer flag is raised, one option would be to stop the timer with the instructions "CLR TR0" or "CLR TR1", for Timer 0 and Timer 1, respectively. Again, it must be noted that each timer has its own timer flag: TF0 for Timer 0, and TF1 for Timer1.
4. After the timer reaches its limit and rolls over, in order to repeat the process the registers. TH and TL must be reloaded with the original value, and TF must be reset to 0.

Mode 1:

In mode 1 16 bit timer is used. The 16-bit counter can hold values between 0000 to FFFFH in TH - TL. Therefore, when the timer reaches its maximum of FFFH, it rolls over to 0000, and TF is raised.

Mode 1 programming

The following are the characteristics and operations of mode 1 :

1. It is a 16-bit timer; therefore, it allows values of 0000 to FFFFH to be loaded into the timer's registers TL and TH.
2. After TH and TL are loaded with a 16-bit initial value, the timer must be started. This is done by "SETB TR0" for Timer 0 and :SETB TR1" for Timer 1.
3. After the timer is started, it starts to count up. It counts up until it reaches its limit of FFFFH. When it rolls over from FFFFH to 0000, it sets high a flag bit called TF (timer flag). This timer flag can be monitored. When this timer flag is raised, one option would be to stop the timer with the instructions "CLR TR0" or "CLR TR1", for Timer 0 and Timer 1, respectively. Again, it must be noted that each timer has its own timer flag: TF0 for Timer 0, and TF1 for Timer1.
4. After the timer reaches its limit and rolls over, in order to repeat the process the registers. TH and TL must be reloaded with the original value, and TF must be reset to 0.

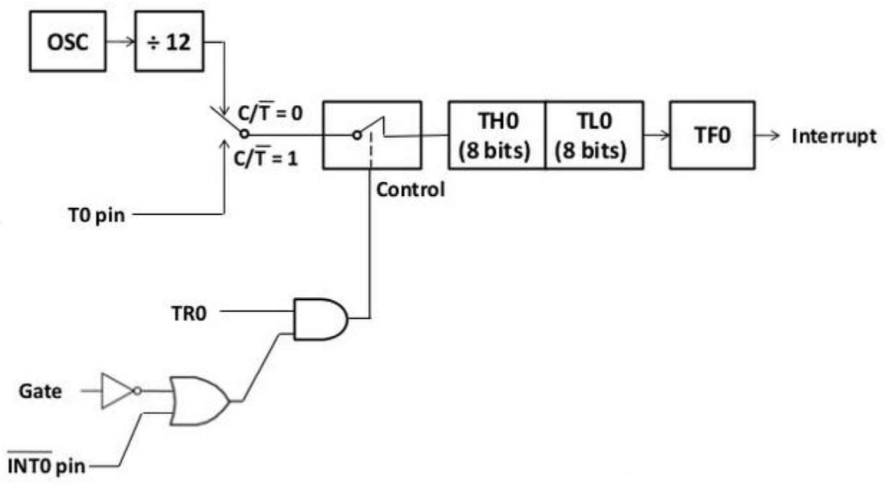


Fig 5.5: Mode 1

Mode 2:

In this mode, the timer register is 8 bits wide. TL0 for Timer 0 (or TL1 for timer 1) is used for this purpose. This mode is also called the auto-reload mode as the timer generates an interrupt on overflow and after generating the interrupt, will also reload the preset value from TH0 into TL0. This preset value can be put in the TH0 through software.

For example, let TL0 = 80H, TH0 = 80H. The interrupt flag is set when TL0 goes from all 1s to all 0s. After generating an interrupt it also reloads the TL0 with the value from TH0 and the starts counting again.

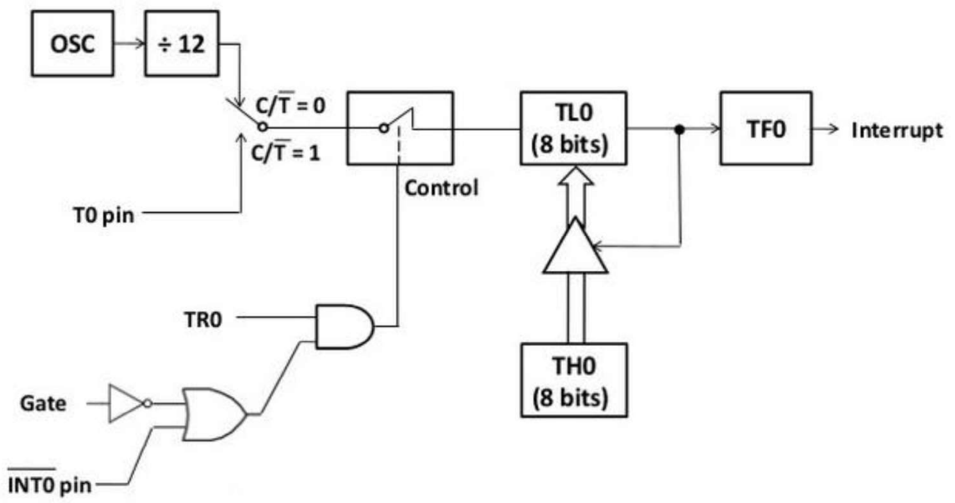


Fig 5.5: Mode 2

Mode 2 programming

The following are the characteristics and operations of mode 2.

1. It is an 8-bit timer; therefore, it allows only values of 00 to FFH to be loadedinto the timer's register TH.
2. After TH is loaded with the 8-bit value, the 8051 gives a copy of it to TL. Thenthe timer must be started. This is done by the instruction "SETB TR0" for Timer 0 and "SETB TR1, for Timer 1. This is just like mode 1.
3. After the timer is started, it starts to count up by incrementing the TL register.It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00, it sets high the TF (timer flag). If we are using Timer 0, TFO goes high; ifwe are using Timer 1, TF1 is raised.
4. When the TL register rolls from FFH to 00 and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register. To repeat the process, we must simply clear TF and let it go without any need by the programmer to reload the original value. This makes mode 2 an auto-reload, in contrast with mode 1 in which the programmer has to reload TH and TL.

It must be emphasized that mode 2 is an 8-bit timer. However, it has an auto-reloading capability. In auto-reload, TH is loaded with the initial count and a copy of it is given to TL. This reloading leaves TH unchanged, still holding a copy of the original value. This mode has many applications, including setting the baud rate in serial communication.

Mode 3 programming:

If the Timer 0 is put into mode 3, then it acts as two 8-bit counters (TL0 and TH0 become two separate counters). All the timer 0 control bits (C/$\overline{T}$, Gate, TR0, TF0 and $\overline{INT0}$) are used by TL0 itself and TH0 register is locked into a timer function. TH0 is counting machine cycle s and has taken over the use of TR1 and TF1 from Timer 1.

Therfore TH0 will now control Timer 1 interrupt. If the Timer 1 is put into mode 3, it just holds the count. The effect is same as setting TR1 = 0, hence opening the switch.

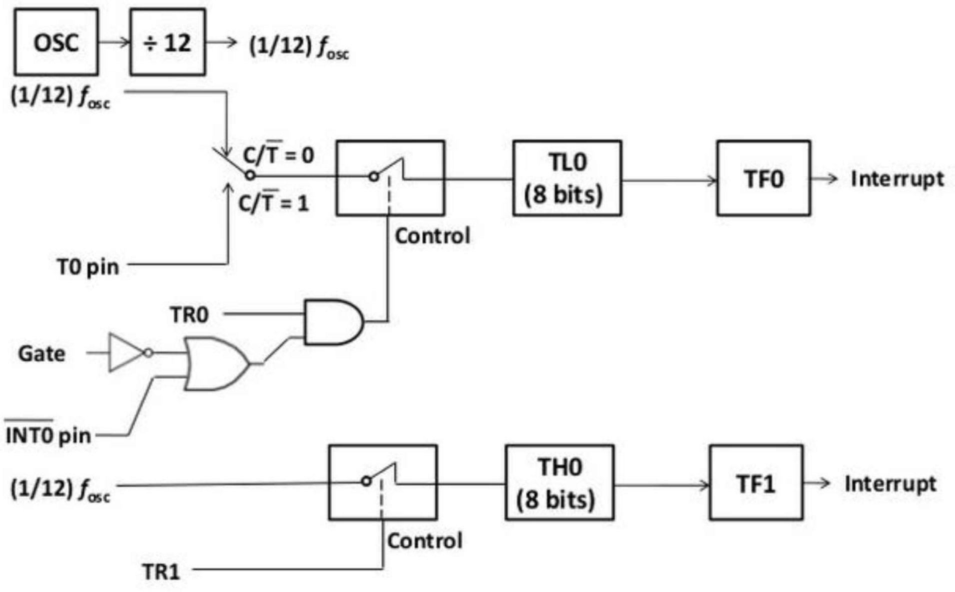


Fig 5.7: Mode 3

5.1.4Counter programming

The timer/counter of the 8051 to generate time delays. These timers can also be used as counters counting events happening outside the 8051.

When thetimer/counter is used as a timer, the 8051's crystal is used as the source of the frequency. When it is used as a counter, however, it is a pulse outside the 8051 that increments the TH, TL register.

C/$\overline{T}$ bit in TMOD register

If C/$\overline{T}$=0, the timer gets pulses from the crystal. In contrast, when C/$\overline{T}$=1, the timer is used as a counter and gets its pulses from outside the 8051.

We use Timer 1 as an event counter where it counts up as clock pulses.These clock pulses could represent the number of people passing through an entrance, or the number of wheel rotations, or any other event that can be converted to pulses.

As application of the timer with C/$\overline{T}$=1, we can feed an external square wave of 60 Hz frequency into the timer. The program will generate the second, the minute, and the hour out of this input frequency and display the result on an LCD. This will be a nice digital clock, but not a very accurate one.

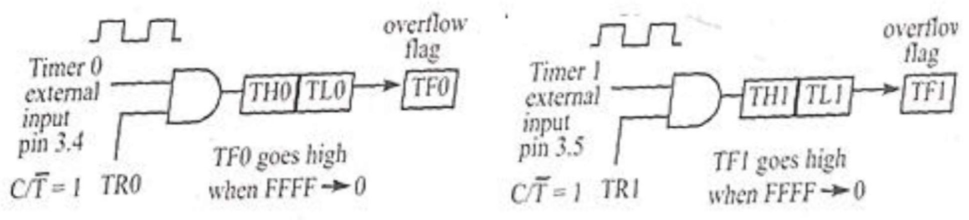

Timer 0 external input pin 3.4
C/$\overline{T}$ = 1
TR0
TH0
TL0
overflow flag
TF0
TF0 goes high when FFFF → 0
Timer 1 external input pin 3.5
C/$\overline{T}$ = 1
TR1
TH1
TL1
overflow flag
TF1
TF1 goes high when FFFF → 0

Fig 5.8: (a) Timer 0 with external input(mode1) (b) Timer 1 with external input (mode 1)

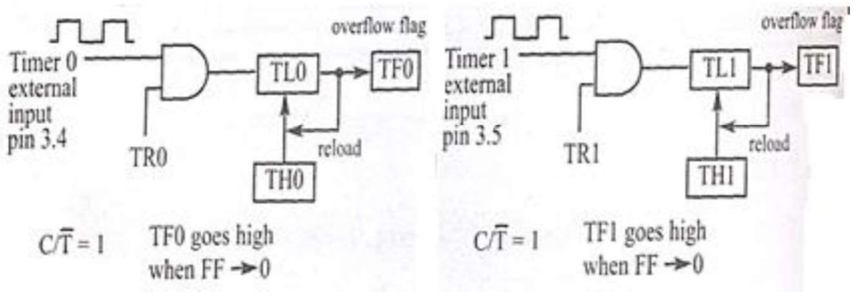

Timer 0 external input pin 3.4
TR0
TL0
TF0
overflow flag
reload
TH0
C/$\overline{T}$ = 1
TF0 goes high when FF → 0
Timer 1 external input pin 3.5
TR1
TL1
TF1
overflow flag
reload
TH1
C/$\overline{T}$ = 1
TF1 goes high when FF → 0

Fig 5.9: (a) Timer 0 with external input(mode 2) (b) Timer 1 with external input (mode 2)

5.1.5 TCON register

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Fig 5.10: TCON Register

TF1 : Timer 1 overflow flag.

Set by hardware when the timer/counter overflows.

Cleared by hardware when the processor vectors to the interrupt routine.

TR1 : Timer 1 run control bit.

Set/cleared by software to turn the timer/counter on/off.

TF0 : Timer 0 overflow flag.

Set by hardware when the timer/counter overflows.

Cleared by hardware when the processor vectors to the interrupt routine.

TR0 : Timer 0 run control bit.

Set/cleared by software to turn the timer/counter on/off.

IE1 : Interrupt 1 edge flag.

Set by hardware when the external interrupt edge is detected.

Cleared when the interrupt is processed.

IT1 : Interrupt 1 type control bit

Set/ Cleared by software to specify the falling edge/low level triggered external interrupts.

- When IT1 = 1 then $\overline{INT1}$ is falling edge triggered
- When IT0 = 0 then $\overline{INT1}$ is low-level triggered

IE0 : Interrupt 0 edge flag.

Set by hardware when the external interrupt edge is detected.

Cleared when the interrupt is processed.

IT0 : Interrupt 0 type control bit

Set/ Cleared by software to specify the falling edge/low level triggered external interrupts.

- When IT0 = 1 then $\overline{INT0}$ is falling edge triggered
- When IT0 = 0 then $\overline{INT0}$ is low-level triggered

Equivalent Instructions for the Timer control register(TCON)

<u>FOR TIMER 0</u>
SETB TRO = SETB TCON.4
CLR TRO = CLR TCON.4
SETB TFO= SETB TCON.5
CLR TFO = CLR TCON.5
<u>FOR TIMER 1</u>
SETB TR1 = SETB TCON6
CLR TR1=CLR TCON 6
SETB TF1=SETB TCON7
CLR TF1=CLR TCON7

5.2 SERIAL PORT PROGRAMMING

5.2.1 Basics of Serial Communication

When a microprocessor communicates with the outside world, it provides the data in byte-sized chunks. In some cases, such as printers, the information is simply grabbed from the 8-bit data bus and presented to the 8-bit data bus of the printer. This can work only if the cable is not too long, since 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. Figure 5.11shows serial versus parallel data transfers.

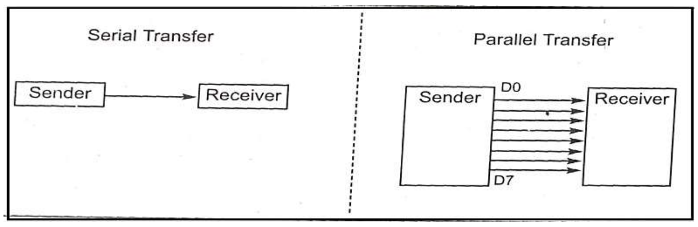


Fig 5.11: Serial versus Parallel Data Transfer

The fact that serial communication uses a single data line instead of the 8-bit data line of parallel communication not only makes it much cheaper but also enables two computers located in two different cities to communicate over the telephone.

For serial data communication to work, the byte of data must be convertedto 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.

5.2.2 Serial data communication

Serial data communication uses two methods,

- Synchronous and
- Asynchronous.

Synchronous:

The synchronous method transfers a block of data (characters) at a time.

Asynchronous:

Asynchronous data communication is widely used for character-oriented transmissions i.e, transfers a single byte at a time

The data coming in at the receiving end of the data line in a serial data transfer is all 0s and 1s; 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

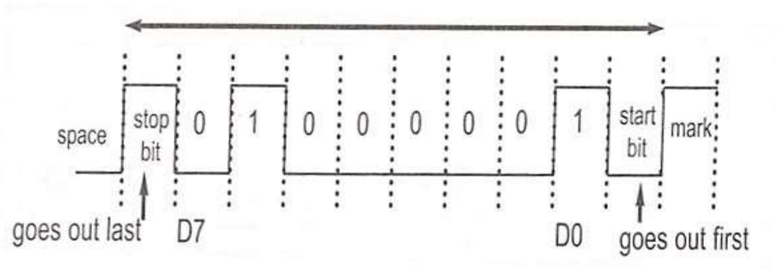


Fig 5. 12: Framing ASCII "A" (41H)

In the asynchronous method, 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 5.12 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. The transmission begins with a start bit followed by D0, which is the LSB, then the rest of the bits until the MSB (D7), and finally, the one stop bit indicating the end of the character "A".

There are special 1C chips made by many manufacturers for serial data communications. These chips are commonly referred to as UART (universal asynchronous receiver-transmitter) and USART (universal synchronous-asynchronous receiver-transmitter). The 8051 chip has a built-in UART.

5.2 3 Data transmission:

- Simplex
- Duplex

Simplex:

Simplex is a communication mode in which only one signal is transmitted, and it always goes in the same direction.

Duplex:

In data transmission if the data can be transmitted and received, it is a duplex transmission. This is in contrast to simplex transmissions. Duplex transmissions can be

- Half or
- Full duplex,

Half duplex:

If the data is transmitted one way at a time, it is referred to as half duplex.

Full duplex :

If the data 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 transmission and one for reception, in order to transfer and receive data simultaneously.

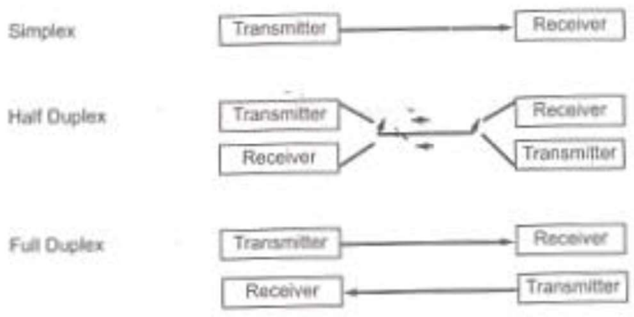


Fig5.13: Simplex, Half-, and Full-Duplex Transfers

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 isbaud rate. However, the baud and bps rates are not necessarily equal. This is due to the fact that baud rate. is the modem terminology and is defined as the number of signal changes per second. In modems a single change of signal, sometimes transfers several bits of data. As far as the conductor wire is concerned, the baud rate and bps are the same, and for this reason in this book we use the terms bps and baud interchangeably.

The data transfer rate of a given computer system depends on communication ports incorporated into that system.

5.2.4 Serial Interface:

RxD pin is used to receive the input serially and TxD is used to transmit the data serially. Since the buffer is only one byte wide , the CPU must read the previous byte before the second byte is fully received, otherwise one of the bytes will be lost. The special function register SBUF is used for both receiving and transmitting the byte.

SBUF register

SBUF is an 8-bit register used solely for serial communication in the 8051. For a byte of data to be transferred via the TxD line, it must be placed in the SBUF register. Similarly, SBUF holds the byte of data when it is received by the 8051 ‘sRxD line. SBUF can be accessed like any other register in the 8051.

Mode 0:

In this mode 8051 transmits and receives through the RxD pin. The TxD outputs the shift clock. While transmitting the LSB of the byte is sent out first. While receiving LSB is received first. Once all the 8 data bits are transmitted, the transmit interrupt (TI) flag is set and an interrupt is generated. After receiving all the 8 bits, the Receive interrupt flag is set and an interrupt is generated.

The baud rate in this mode is constant and it is 1/12th of the oscillator frequency.

Mode 1:

In this mode 8051 transmits 10 bits of information through TxD and receives 10 bits of information through RxD. The 10-bit frame is classified as follows:

1. 1 bit for start
2. 8 bits of data (LSB first)
3. 1 bit for stop.

Once the stop bit is received, it means the reception of one frame is complete, that is a byte of data has come. As in mode 0, here also an interrupt is generated once all the bits in the frame are received or transmitted. Here the baud rate is variable and it depends on timer 1 overflow rate.

Mode 2:

In this mode 8051 transmits 11 bits of information through TxD and receives 10 bits of information through RxD. The 11-bit frame is classified as follows:

1. 1 bit for start
2. 8 bits of data (LSB first)
3. 1 bit can be programmed
4. 1 bit for stop.

Once the stop bit is received, it means the reception of one frame is complete, that is a byte of data has come. As in mode 0, here also an interrupt is generated once all the bits in the frame are received or transmitted. Here the baud rate is 1/32 or 1/64th of the oscillator frequency.

Mode 3:

In this mode 8051 transmits 11 bits of information through TxD and receives 10 bits of information through RxD. The 11-bit frame is classified as follows:

1. 1 bit for start
2. 8 bits of data (LSB first)
3. 1 bit can be programmed
4. 1 bit for stop.

Once the stop bit is received, it means the reception of one frame is complete, that is a byte of data has come. As in mode 0, here also an interrupt is generated once all the bits in the frame are received or transmitted. Here the baud rate is variable and it depends on timer 1 overflow rate.

5.2.5 SCON (serial control) register

The SCON register is an 8-bit register used to program the start bit, stop bit, and data bits of data framing, among other things.

SM0	SM1	SM2	REN	TB8	RB8	T1	RI

Fig 5.14: SCON Serial Port Control Register (Bit-Addressable)

SM0, SM1

SM0 and SM1 are D7 and D6 of the SCON register, respectively. These two bits determine the framing of data by specifying the number of bits per character, and the start and stop bits. They take the following combinations.

SM0	SM1	Mode	Baud Rate
0	0	Serial mode 0 (8 data bits)	1/12th of the oscillator frequency
0	1	Serial mode 1(1 bit for start, 8 bits of data, 1 bit for stop)	Baud rate is variable, depends on timer 1 overflow rate.
1	0	Serial mode 3(1 bit for start, 8 bits of data, 1 bit can be programmed, 1 bit for stop)	1/32 or 1/64th of the oscillator frequency.
1	1	Serial mode 3(1 bit for start, 8 bits of data, 1 bit can be programmed, 1 bit for stop)	Baud rate is variable, depends on timer 1 overflow rate.

SM2

SM2 is the D5 bit of the SCON register. This bit enables the multiprocessing capability of the 8051.

REN

The REN (receive enable), bit is D4 of the SCON register. When the REN bit is high, it allows the 8051 to receive data on the RxD pin of the 8051. As a result if we want the 8051 to both transfer and receive data, REN must be set to 1. By making REN = 0, the receiver is disabled.

TB8

TB8 (transfer bit 8) is bit D3 of SCON. It is used for serial modes 2 and 3. We make TB8 = 0 since it is not used in our applications.

RB8

RB8 (receive bit 8) is bit D2 of the SCON register. In serial mode 1, this bit gets a copy of the stop bit when an 8-bit data is received.

Tl

TI (transmit interrupt) is bit Dl of the SCON register. This is an extremely important flag bit in the SCON register. When the 8051 finishes the transfer of the 8-bit character, it raises the TI flag to indicate that it is ready to transfer another byte. The TI bit is raised at the beginning of the stop bit.

Rl

RI (receive interrupt) is the DO bit of the SCON register. This is another extremely important flag bit in the SCON register. When the 8051 receives data serially via RxD, it gets rid of the start and stop bits and places the byte in the SBUF register. Then it raises the RI flag bit to indicate that a byte has been received and should be picked up before it is lost. RI is raised halfway through the stop bit.

5.2.6 Programming the 8051 to transfer data serially

In programming the 8051 to transfer character bytes serially, the following steps must be taken.

1. The TMOD register is loaded with the value 20H, indicating the use of Timer 1 in mode 2 (8-bit auto-reload) to set the baud rate.
2. The TH1 is loaded with one of the values in Table 10-4 to set the baud rate for serial data transfer (assuming XTAL = 11.0592 MHz).
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits.
4. TR1 is set to 1 to start Timer 1.
5. TI is cleared by the “CLR TI” instruction.
6. The character byte to be transferred serially is written into the SBUF register.
7. The TI flag bit is monitored with the use of the instruction ” JNB TI, xx” to see if the character has been transferred completely.
8. To transfer the next character, go to Step 5.

Doubling the baud rate in the 8051

There are two ways to increase the baud rate of data transfer in the 8051.

1. Use a higher-frequency crystal.
2. Change a bit in the PCON register, shown below.

5.2.7 Power Control register (PCON):

D7							D0
SMOD	-	-	-	GF1	GF0	PD	IDL

Option 1 is not feasible in many situations since the system crystal is fixed. Therefore, we will explore option 2. There is a software way to double the baud rate of the 8051 while the crystal frequency is fixed. This is done with the register called PCON (power control). The PCON register is an 8-bit register. Of the 8 bits, some are unused, and some are used for the power control capability of the 8051. The bit that is used for the serial communication is D7, the SMOD (serial mode) bit. When the 8051 is powered up, D7 (SMOD bit) of the PCON register is zero. We can set it to high by software and thereby double the baud rate.

5.2.8 RS232 standards

To allow compatibility among data communication equipment made by various manufacturers, an interfacing standard called RS232 was set by the Electronics Industries Association (EIA) in 1960.

RS232 is the most widely used serial I/O interfacing standard. This standard is used in PCs and numerous types of equipment.

In RS232, a 1 is represented by -3 to -25 V, while a 0 bit is +3 to +25 V, 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.

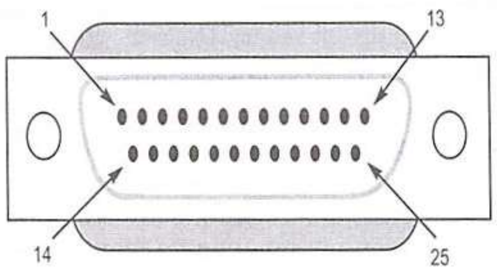


Fig 5.14: RS232 Connector DB-25

RS232 pins

Below table provides the pins and their labels for the RS232 cable, commonly referred to as the DB-25 connector.

PIN	DESCRIPTION
1	PROTECTIVE GROUND
2	TRANSMITTED DATA(TxD)
3	RECEIVED DATA(RxD)
4	REQUEST TO SEND(RTS)
5	CLEAR TO SEND(CTS)
6	DATA SET READY(DSR)
7	SIGNAL GROUND(GND)
8	DATA CARRIER DETECT
9/10	RESERVED FOR DATA TESTING
11	UNASSIGNED
12	SECONDARY DATA CARRIER DETECT
13	SECONDARY CLEAR TO SEND
14	SECONDARY TRANSMITTED DATA
15	TRANSMIT SIGNAL ELEMENT TIMING
16	SECONDARY RECEIVED DATA
17	RECEIVE SIGNAL ELEMENT TIMING
18	UNASSIGNED
19	SECONDARY RQUEST TO SEND
20	DATA TERMINAL READY(DTR)
21	SIGNAL QUALITY DETECTOR
22	RING INDICATOR
23	DATA SIGNAL RATE SELECT
24	TRANSMIT SIGNAL ELEEMENT TIMING
25	UNASSIGNED

5.2.9 Data communicationclassification

Current terminology classifies data communication equipment as DTE (data terminal equipment) or DCE (data communication equipment). DTE refers to terminals and computers that send and receive data, while DCE refers to communication equipment, such as modems, that are responsible for transferring the data.

Examining RS232 handshaking signals

To ensure fast and reliable data transmission between two devices, the data transfer must be coordinated. Many of the pins of the RS-232 connector are used for handshaking signals. Their descriptions are provided below only as a reference and they can be bypassed since they are not supported by the 8051 UARTchip.

1. DTR (data terminal ready). When a terminal (or a PC COM port) is turned on, after going through a self-test, it sends out signal DTR to indicate that it is ready for communication. If there is something wrong with the COM port, this signal will not be activated. This is

an active-low signal and can be used to inform the modem that the computer is alive and kicking. This is an output pin from DTE (PC COM port) and an input to the modem.

2. DSR (data set ready). When DCE (modem) is turned on and has gone through the self-test, it asserts DSR to indicate that it is ready to communicate. Thus, it is an output from the modem (DCE) and input to the PC (DTE). This is an active- low signal. If for any reason the modem cannot make a connection to the telephone, this signal remains inactive, indicating to the PC (or terminal) that it cannot accept or send data.
3. RTS (request to send). When the DTE device (such as a PC) has a byte to transmit, it asserts RTS to signal the modem that it has a byte of data to transmit. RTS is an active-low output from Ihe DTE and an input to the modem.
4. CTS (clear to send). In response toRTS, when the modem has room for storing the data it is to receive, it sends out signal CTS to the DTE (PC) to indicate that it can receive the data now. This input signal to the DTE is used by the DTE to start transmission.
5. DCD (carrier detect, or DCD, data carrier detect). The modem asserts signalDCD to inform the DTE (PC) that a valid carrier has been detected and thatcontact between it and the other modem is established. Therefore, DCD is anoutput from the modern and an input to the PC (DTE).
6. RI (ring indicator). An output from the modem (DCE) and an input to a PC (DTE) indicates that the telephone is ringing. It goes on and off in synchronization with the ringing sound. Of the six handshake signals, this is the leastoften used, due to the fact that modems take care of answering the phone. However, if the PC is in charge of answering the phone, this signal can be used.

5.2.10 8051 Connection To Rs232

The RS232 standard is not TTL compatible; therefore, it requires a line driver such as the MAX232 chip to convert RS232 voltage levels to TTL levels, and vice versa.

RxD and TxD pins in the 8051

The 8051 has two pins that are used specifically for transferring and receiving data serially. These two pins are called TxD and RxD and are part of the port 3 group (P3.0 and P3.1). Pin 11 of the 8051 (P3.1) is assigned to TxD and pin 10 (P3.0) is designated as RxD. These pins are TTL compatible.

MAX232

Since 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 8051 'sTxD and RxD pins. One example of such a converter is MAX232 from Maxim Corp. 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 8051. In other words, with a single +5 V power supply we can power both the 8051 and MAX232, with no need for the dual power supplies that are common in many older systems.

The MAX232 has two sets of line drivers for transferring and receiving data, as shown in Figure. The linedrivers used for TxD are called Tl and T2, while the line drivers for RxD are designated as Rl and R2. In many applications only one of each is used. For example, Tl and Rl are used together for TxD and RxD of the 8051, and the second set is left unused. Notice in MAX232 that the Tl line driver has a designation of Tlin and Tlout on pin numbers 11 and 14, respectively. The Tlin pin is the TTL side and is connected to TxD of the microcontroller, while Tlout is the RS232 side that is connected to the RxD pin of the RS232 DB connector. The Rl 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 TxD pin of the RS232 DB connector, and Rlout (pin 12) is the TTL side that is connected to the RxD pin of the microcontroller. Notice the null modem connection where RxD for one is TxD for the other.

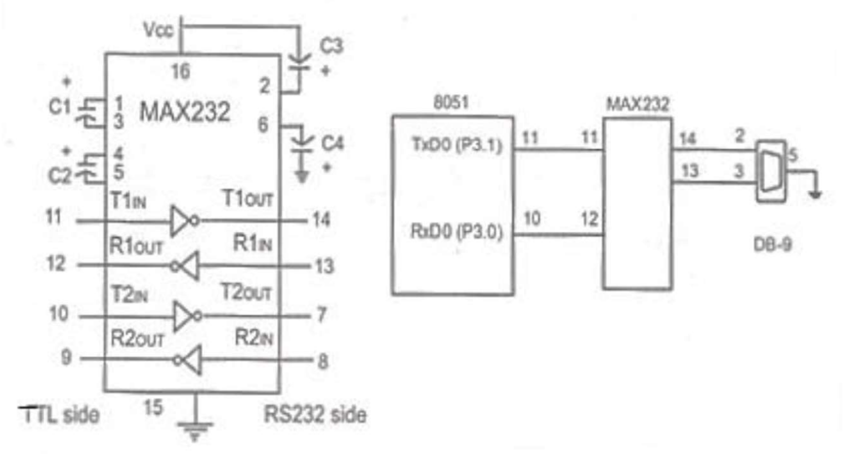


Fig 5.15:Inside MAX 232 and its connection to the 8051

MAX232 requires four capacitors ranging from 1 to 22 μF. The most widely used value for these capacitors is 22 μF.

MAX 233

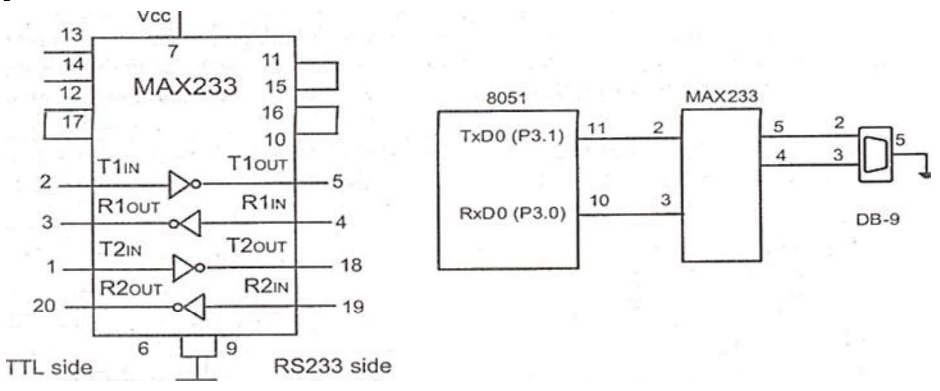


Fig.5.16:Inside MAX233 and its connection to the 8051

To save board space, some designers use the MAX233 chip from Maxim. The MAX233 performs the same job as the MAX232 but eliminates the need for capacitors. However, the MAX233 chip is much more expensive than the MAX232. Notice that MAX233 and MAX232 are not pin compatible. You cannot take a MAX232 out of a board and replace it with a MAX233. For MAX233 with no capacitor used.

5.3 INTERRUPTS PROGRAMMING

5.3.1 Interrupts vs. polling

A single microcontroller can serve several devices. There are two ways to do that:

- Interrupts
- Polling.

Interrupts:

In the interrupt method, whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal. Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device. The program associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler.

Polling:

In polling, the microcontroller continuously monitors the status of a given device; when the status condition is met, it performs the service. After that, it moves on to monitor the next device until each one is serviced. Although polling can monitor the status of several devices and serve each of them as certain conditions are met, it is not an efficient use of the microcontroller.

Advantages of interrupts:

The advantage of interrupts is that the microcontroller can serve many devices (not all at the same time, of course); each device can get the attention of the microcontroller based on the priority assigned to it. The polling method cannot assign priority since it checks all devices in a round-robin fashion. In the interrupt method the microcontroller can also ignore (mask) a device request for service. This is again not possible with the polling method.

Disadvantages of polling:

Polling method wastes much of the microcontroller's time by polling devices that do not need service. So in order to avoid tying down the microcontroller, interrupts are used.

Interrupt service routine:

For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler. When an interrupt is invoked, the microcontroller runs the interrupt service routine. For every interrupt, there is a fixed location in memory that holds the address of its ISR.

Interrupt vector table:

The group of memory locations set aside to hold the addresses of ISRs is called the interrupt vector table

5.3.2 Steps in executing an interrupt

Upon activation of an interrupt, the microcontroller goes through the following steps.

1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack.
2. It also saves the current status of all the interrupts internally (i.e., not on the stack).
3. It jumps to a fixed location in memory called the interrupt vector table that holds the address of the interrupt service routine.
4. The microcontroller gets the address of the ISR from the interrupt vector tableand jumps to it. It starts to execute the interrupt service subroutine until itreaches the last instruction of the subroutine, which is RETI (return from interrupt).
5. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted. First, it gets the program counter (PC) address from the stack by popping the top two bytes of the stack into the PC. Then it starts to execute from that address.

5.3.3 Six interrupts in the 8051

In reality, only five interrupts are available to the user in the 8051, data sheets state that there are six interrupts since they include reset. The six interrupts in the 8051 are allocated as follows.

1. Reset. When the reset pin is activated, the 8051 jumps to address location0000.
2. Two interrupts are set aside for the timers: one for Timer 0 and one for Timer 1. Memory locations 000BH and 001BH in the interrupt vector table belongto Timer 0 and Timer 1, respectively.
3. Two interrupts are set aside for hardware external hardware interrupts. Pinnumbers 12 (P3.2) and 13 (P3.3) in port 3 are for the external hardware interrupts INT0 and INT1, respectively. These external interrupts are also referred to as EX1 and EX2. Memory locations 0003H and 0013H in the interrupt vector table are assigned to INT0 and INT1, respectively.
4. Serial communication has a single interrupt that belongs to both receive andtransmit. The interrupt vector table location 0023H belongs to this interrupt.

Interrupt	ROM Location(Hex)	Pin	Flag Clearing
Reset	0000	9	Auto
External hardware interrupt 0(INT0)	0003	P3.2(12)	Auto
Timer 0 interrupt(TF0)	000B		Auto
External hardware interrupt 1(INT1)	0013	P3.2(13)	Auto
Timer 1 interrupt(TF1)	001B		Auto
Serial COM interrupt (R1 and T1)	0023		Programmer clears it

Enabling and disabling an interrupt

Upon reset, all interrupts are disabled (masked), meaning that none will be responded to by the microcontroller if they are activated. The interrupts must be enabled by software in order for the microcontroller to respond to them. There is a register called IE (interrupt enable) that is responsible for enabling (unmasking) and disabling (masking) the interrupts.

IE (interrupt enable) Register:

D7 D0

EA	X	X	ES	ET1	EX0	ET0	EX0

EA (enable all):

If EA=0, no interrupt will be acknowledged. If EA=1, each interrupt source is individually enabled or disabled by setting or clearing the enable bit.

ES Enables or disables Serial port interrupt:

If ES=0, serial port interrupt is disabled.

ET1 Enables or disables the Timer 1 overflow interrupt:

If ET1 = 0 Timer 1 overflow interrupt is disabled

EX1 Enables or disables the external interrupt 1

If EX1 =0 external interrupt 1 is disabled

ET0 Enables or disables Timer 0 overflow interrupt

If ET0 = 0 Timer 0 overflow interrupt is disabled

EX0 Enables or disables the external interrupt 0

If ET0 = 0 external interrupt 0 is disabled

5.3.4 Programming Timer Interrupts

Roll-over timer flag and interrupt

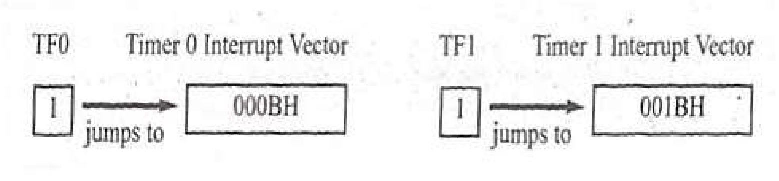


Fig.5.17: TF interrupt

The timer flag (TF) is raised when the timer rolls over. In polling TF, we have to wait until the TF is raised. The problem with this method is that the microcontroller is tied down while waiting for TF to be raised, and cannot do anything else. Using interrupts solves this problem and avoids tying down the controller. If the timer interrupt in the IE register is enabled, whenever the timer rolls over, TF is raised, and the microcontroller is interrupted in whatever it is doing, and jumps to the interrupt vector table to service the ISR. In this way, the microcontroller can do other things until it is notified that the timer has rolled over.

1. The ISR for Timer 0 is located starting at memory location 000BH since it issmall enough to fit the address space allocated to this interrupt.
2. We enabled the Timer 0 interrupt with “MOV IE, #10000010B” in MAIN.
3. While the P0 data is brought in and issued to P1 continuously, whenever Timer0 is rolled over, the TF0 flag is raised, and the microcontroller gets out of the “BACK” loop and goes to 000BH to execute the ISR associated with Timer 0.
4. In the ISR for Timer 0, notice that there is no need for a “CLR TF0” instruction before the RETI instruction. This is because the 8051 clears the TF flaginternally upon jumping to the interrupt vector table.

5.3.5 Programming External Hardware Interrupts

External interrupts INTO and INT1

There are only two external hardware interrupts in the 8051: INT0 and INT1. They are located on pins P3.2 and P3.3 of port 3, respectively. The interrupt vector table locations 0003H and 0013H are set aside for INT0 and INT1, respectively. They are enabled and disabled using the IE register. How are they activated? There are two types of activation for the external hardware interrupts:

1. Level triggered, and
2. Edge triggered.

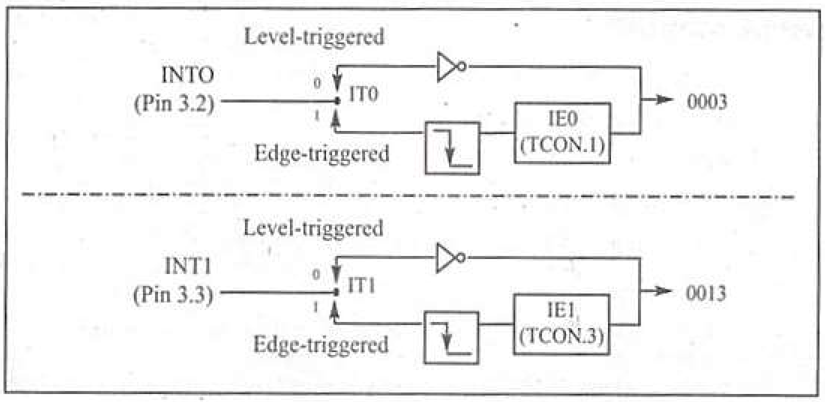


Fig.5.18: Activation of INT0 and INT1

Level-triggered interrupt

In the level-triggered mode, INT0 and INT1 pins are normally high (just like all I/O port pins) and if a low-level signal is applied to them, it triggers the interrupt. Then the microcontroller stops whatever it is doing and jumps to the interrupt vector table to service that interrupt. This is called a level- triggered or level-activated interrupt and is the default mode upon reset of the 8051. The low-level signal at the INT pin must be removed before the execution of the last instruction of the interrupt service routine, RETI; otherwise, another interrupt will be generated.

The TCON register holds, among other bits, the IT0 and IT1 flag bits that determine level- or edge-triggered mode of the hardware interrupts. IT0 and IT1 are bits D0 and D2 of the TCON register, respectively. They are also referred to as TCON.0 and TCON.2 since the TCON register is bit-addressable. Upon reset, TCON.0 (ITO) and TCON.2 (IT1) are both 0s, meaning that the external hardware interrupts of INTO and INT1 pins are low-level triggered.

Edge-triggered interrupts

To make INT0 and INT1 as edge-triggered interrupts, we must program the bits of the TCON register. By making the TCON.0 and TCON.2 bits high with instructions such as “SETB TCON. 0″ and “SETB TCON. 2″, the external hardware interrupts of INT0 and INT1 become edge-triggered. For example, the instruction “SETB CON. 2″ makes INT1 what is called an edge-triggered interrupt, in which, when a high-to-low signal is applied to pin P3.3, in this case, the controller will be interrupted and forced to jump to location 0013H in the vector table to service the ISR (assuming that the interrupt bit is enabled in the IE register).

5.3.6 Programming the Serial Communication Interrupt

Rl and Tl flags and interrupts

TI (transfer interrupt) is raised when the last bit of the framed data, the stop bit, is transferred, indicating that the SBUF register is ready to transfer the next byte. RI (received interrupt), is raised when the entire frame of data, including the stop bit, is received. In other words, when the SBUF register has a byte, RI is raised to indicate that the received byte needs to be picked up before it is lost (overrun) by new incoming serial data. As far as serial communication is concerned, all the above concepts apply equally when using either polling or an interrupt. The only difference is in how the serial communication needs are served. In the polling method, we wait for the flag (TI or RI) to be raised; while we wait we cannot do anything else. In the interrupt method, we are notified when the 8051 has received a byte, or is ready to send the next byte.

In the 8051 only one interrupt is set aside for serial communication. This interrupt is used to both send and receive data. If the interrupt bit in the IE register (IE.4) is enabled, when RI or TI is raised the 8051 gets interrupted and jumps to memory address location 0023H to execute the ISR. In that ISR we must examine the TI and RI flags to see which one caused the interrupt and respond accordingly.

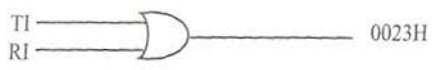


Fig 5.19 Serial Interrupt is invoked by TI and RI flags

5.3.7 Interrupt Priority in 8051/52

Interrupt priority upon reset

When the 8051 is powered up, the priorities are assigned. For example, that if external hardware interrupts 0 and 1 are activated at the same time, external interrupt 0 (INT0) is responded to first. Only after INT0 has been serviced is INT1 serviced, since INT1 has the lower priority. Highest to lowest priority

External interrupt 0(INT0)
Timer interrupt 0(TF0)
External interrupt 1(INT1)
Timer interrupt 1(TF1)
Serial communication (RI+TI)
Timer 2(8052 only) TF2

Interrupt Priority (IP) Register

D7							D0
-	-	-	PS	PT1	PX1	PT0	PX0

PS Serial Port Interrupt Priority level
PT1 Timer 1 Interrupt Priority level
PX1 External Interrupt 1 Priority level
PT0 Timer 0 Interrupt Priority level
PX0 External Interrupt 0 Priority level
Priority bit=1 assigns high priority: priority bit =0 assigns low priority

Interrupt inside an interrupt

If the 8051 is executing an ISR belonging to an interrupt and if another interrupt is activated, a high-priority interrupt can interrupt a low-priority interrupt. This is an interrupt inside an interrupt.

Triggering the interrupt by software

There are times when we need to test an ISR by way of simulation. This can be done with simple instructions to set the interrupts high and thereby cause the 8051 to jump to the interrupt vector table. For example, if the IE bit for Timer 1 is set, an instruction such as "SETS TF1" will interrupt the 8051 in whatever it is doing and force it to jump to the interrupt vector table.

5.4 LCD& KEYBOARD INTERFACING

5.4.1 LCD operation

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

1. 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 theCPU of the task of refreshing the LCD. In contrast, the LED must be refreshedby the CPU (or in some other way) to keep displaying the data.
4. Ease of programming for characters and graphics.

LCD pin descriptions

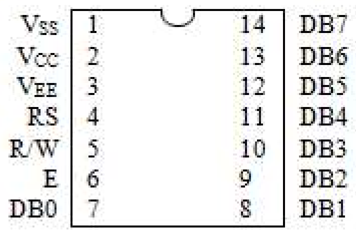


Fig 5.20: Pin Diagram

The LCD discussed in this section has 14 pins. The function of each pin is given in Table. Figure shows the pin positions for various LCDs.

Pin	Symbol	I/O	Description
1	Vss	-	Ground
2	Vcc	-	+5v power supply
3	Vee	-	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

^{V}cc,^{V}ss, $^{\text{and }V}$ee

While V_{cc} and V_{ss} provide +5V and ground, respectively, V_{EE} 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 a command such as clear display, cursor at home, etc.
- 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.

D0-D7

The 8-bit data pins, D0- 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 = 1.

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.

LCD command codes

HEX	Register
1	CLEAR DISPLAY SCREEN
2	RETURN HOME
3	DECREMENT CURSOR(SHIFT CURSOR TO LEFT)
4	INCREMENT CURSOR (SHIFT CURSOR TO RIGHT)
5	SHIFTDISPLAY 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
38	2 LINES AND 5X7 MATRIX

We also use RS = 0 to check the busy flag bit to see if the LCD is ready to receive information. The busy flag is D7 and can be read when R/W = 1 and RS = 0.

- When D7 = 1 (busy flag = 1), the LCD is busy taking care of internal operations and will not accept any new information.
- When D7 = 0, the LCD is ready to receive new information.

Sending commands and data to LCDs with a time delay

For data, make RS = 1. Then send a high-to-low pulse to the E pin to enable the internal latch of the LCD.

Sending code or data to the LCD with checking busy flag

There is a long delay between issuing data or commands to the LCD. However, a much better way is to monitor the busy flag before issuing a command or data to the LCD.

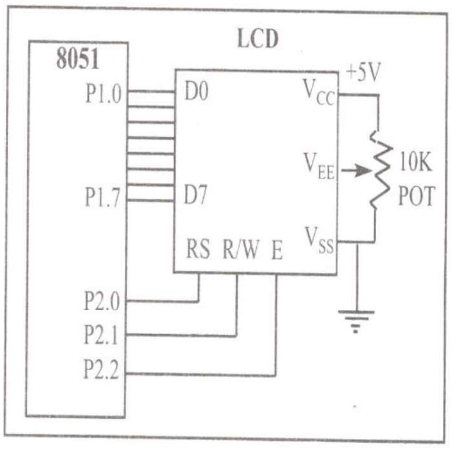


Fig.5.21: LCD connections

.

LCD addressing:

In the LCD, one can put data at any location. The following shows address locations and how they are accessed.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A	A	A	A	A	A	A

where AAAAAAA = 0000000 to 0100111 for line 1 and AAAAAAA - 1000000 to 1100111 for line 2.

	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Line 1(min)	1	0	0	0	0	0	0	0
Line 1(max)	1	0	1	0	0	1	1	1
Line 2 (min)	1	1	0	0	0	0	0	0
Line2 (max)	1	1	1	0	0	1	1	1

The upper address range can go as high as 0100111 for the 40-character-wide LCD, while for the 20-character-wide LCD it goes up to 010011 (19 decimal = 10011 binary). The upper range 0100111 (binary) = 39 decimal, which corresponds to locations 0 to 39 for the LCDs of 40×2 size.

5.4.2 Keyboard Interfacing

Keyboards and LCDs are the most widely used input/output devices of the 8051, and a basic understanding of them is essential.

Interfacing the keyboard to the 8051

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 microprocessor. When a key is pressed, a row and a column make a contact; otherwise, there is no connection between rows and columns. In IBM PC keyboards, a single microcontroller (consisting of a microprocessor, RAM and EPROM, and several ports all on a single chip) takes care of hardware and software interfacing of the keyboard. In such systems, it is the function of programs stored in the EPROM 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

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 1 s for all columns since they are all connected to high (V_{cc}). 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.

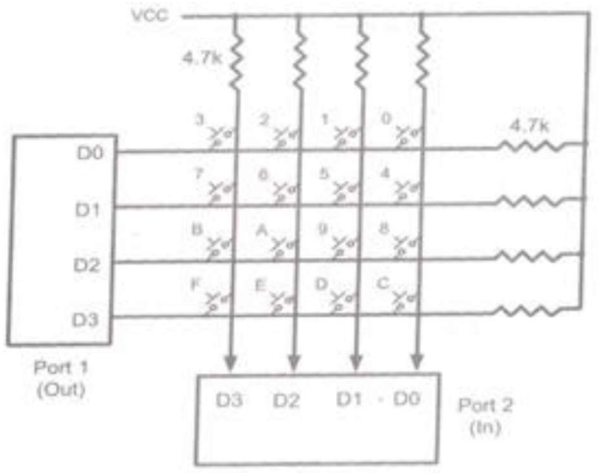


Fig 5.22: Matrix keyboard connection to Rows

Grounding rows and reading the columns

To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch, then it reads the columns. If the data read from the columns is D3 - D0 = 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 - D0 = 1101, this means that a key in the Dl 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 D0 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.

5.5 ADC, DAC & SENSOR INTERFACING

5.5.1 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.

An ADC has n-bit resolution where n can be 8, 10, 12, 16 or even 24 bits. The higher-resolution ADC provides a smaller step size, where step size is the smallest change that can be discerned by an ADC. In this chapter we examine several 8-bit ADC chips. In addition to resolution, 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 ADC chips are either parallel or serial. In parallel ADC, we have 8 or more pins dedicated to bringing out the binary data, but in serial ADC we have only one pin for data out.

ADC0804 chip

The ADC0804 1C is an 8-bit parallel ADC in the family of the ADC0800 series from National Semiconductor. It is also available from many other manufacturers. It works with +5 volts and has a resolution of 8 bits. In the ADC0804, the conversion time varies depending on the clocking signals applied to the CLK IN pin, but it cannot be faster than 110µ.s. The following is the ADC0804 pin description.

CS

Chip select is an active low input used to activate the ADC0804 chip. To access the ADC0804, this pin must be low.

RD (read)

This is an input signal and is active low. The ADC converts the analog input to its binary equivalent and holds it in an internal register. RD is used to get the converted data out of the ADC0804 chip. When CS = 0, if a high-to-low pulse is applied to the RD pin, the 8-bit digital output shows up at the DO - D7 data pins. The RD pin is also referred to as output enable (OE).

WR (write; a better name might be "start conversion")

This is an active low input used to inform the ADC0804 to start the conversion process. If CS = 0 when WR makes a low-to-hightransition,the ADC0804 starts converting the analog input value of V_{in} to an 8-bit digital number. The amount of time it takes to convert varies depending on the CLK IN and CLK R values explained below. When the data conversion is complete, the INTR pin is forced low by the ADC0804.

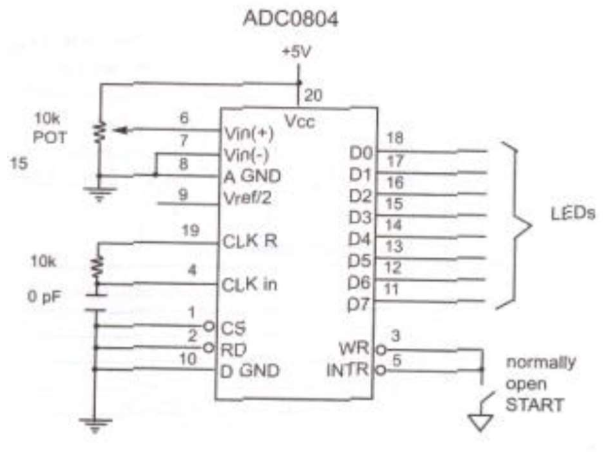


Fig 5.23: ADC0804 Chip (testing ADC0804 in free running mode)

CLK IN and CLK R

CLK IN is an input pin connected to an external clock source when an external clock is used for timing. However, the 804 has an internal clock generator. To use the internal clock generator (also called self-clocking) of the ADC0804, the CLK IN and CLK R pins are connected to a capacitor and a resistor. In that case the clock frequency is determined by the equation:

$$f = \frac{1}{1.1\ Rc}$$

Typical values are R = 10K ohms and C = 150 pF. Substituting in the above equation, we get f= 606 kHz. In that case, the conversion time is 110 μs.

INTR (interrupt; a better name might be "end of conversion")

This is an output pin and is active low. It is a normally high pin and when the conversion is finished, it goes low to signal the CPU that the converted data is ready to be picked up. After INTR goes low, we make CS = 0 and send a high-to-low pulse to the RD pin to get the data out of the ADC0804 chip.

V_{in} (+) and V_{in} (-)

These are the differential analog inputs where $V_{in} = V_{in}(+) - V_{in}(-)$. Often the V_{in} (-) pin is connected to ground and the V_{in} (+) pin is used as the analog input to be converted to digital.

V_{cc}

This is the +5 volt power supply. It is also used as a reference voltage when the $V_{ref}/2$ input (pin 9) is open (not connected).

$V_{ref}/2$

Pin 9 is an input voltage used for the reference voltage. If this pin is open (not connected), the analog input voltage for the ADC0804 is in the range of 0 to 5 volts (the same as the V_{cc} pin). However, there are many applications where the analog input applied to V_{in} needs to be other than the 0 to +5 V range. V_{ref}12is used to implement analog input voltages other than 0 to 5 V. For example, if the analog input range needs to be 0 to 4 volts, $V_{ref}/2$ is connected to 2 volts.

D0-D7

D0- D7 (D7 is the MSB) are the digital data output pins since ADC0804 is a parallel ADC chip. These are tri-state buffered and the converted data is accessed only when CS = 0 and RD is forced low. To calculate the output voltage, use the following formula.

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

whereD_{out} = digital data output (in decimal), V_{in} = analog input voltage, and step size (resolution) is the smallest change, which is (2 x $V_{ref}/2$)/256 for ADC0804.

Analog ground and digital ground

These are the input pins providing the ground for both the analog signal and the digital signal. Analog ground is connected to the ground of the analog V_{in} while digital ground is connected to the ground of the V_{cc} pin. The reason that we have two ground pins is to isolate the analog V_{in} signal from transient voltagescaused by digital switching of the output D0- D7. Such isolation contributes to the accuracy of the digital data output. In our discussion, both are connected to the same ground; however, in the real world of data acquisition the analog and digital grounds are handled separately.

From this discussion we conclude that the following steps must be followed for data conversion by the ADC0804 chip.

1. Make CS = 0 and send a low-to-high pulse to pin WR to start the conversion.
2. Keep monitoring the INTR pin. If INTR is low, the conversion is finished andwe can go to the next step. If INTR is high, keep polling until it goes low.
3. After the INTR has become low, we make CS = 0 and send a high-to-low pulseto the RD pin to get the data out of the ADC0804 1C chip.

Clock source for ADC0804

The speed at which an analog input is converted to the digital output depends on the speed of the CLK input. According to the ADC0804 datasheets, the typical operating frequency is approximately 640 kHz at 5 volts. Since this frequency is too high, we use D flip-flops (74LS74) to divide the frequency. A single D flip-flop divides the frequency by 2 if we connect its Q to the D input. For a higher-frequency crystal, you can use 4 flip-flops.

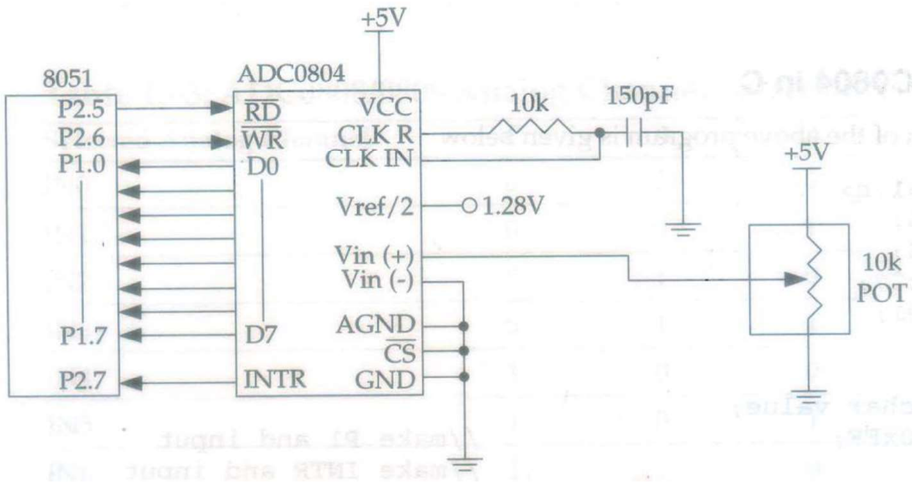


Fig 5.24: 8051 connection to ADC0804 with self-clocking

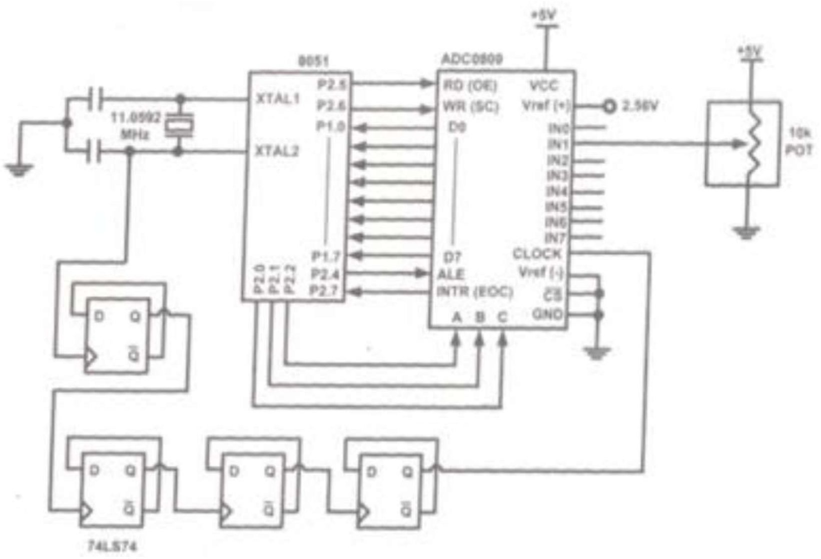


Fig 5.25: 8051 connection to ADC0804 with clock from XTAL2 of 8051

5.5.2 DAC INTERFACING

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 8051.The two methods of creating a DAC are:

- Binary weighted and
- R/2R ladder.

The vast majority of integrated circuit DACsuse the R/2R method since it can achieve a much higher degree of precision. The first criterion for judging a DAC is its resolution, which is a function of the number of binary inputs. The common ones are 8, 10, and 12 bits. The number of data bit inputs decides the resolution of the DAC since the number of analog output levels is equal to 2″, where n is the number of data bit inputs. Therefore, an 8-input DAC such as the DAC0808 provides 256 discrete voltage (or current) levels of output. Similarly, the 12-bit DAC provides 4096 discrete voltage levels. There are also16-bit DACs, but they are more expensive.

MC1408 DAC (or DAC0808)

In the MC1408 (DAC0808), the digital inputs are converted to current (I_{out}), and by connecting a resistor to the I_{out} pin, we convert the result to voltage.The total current provided by the I_{out} pin is a function of the binary numbers at the DO – D7 inputs of the DAC0808 and the reference current (I_{ref}), 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 D0 is the LSB, D7 is the MSB for the inputs, and I_{ref} is the input current that must be applied to pin 14. The I_{ref} current is generally set to 2.0 mA.

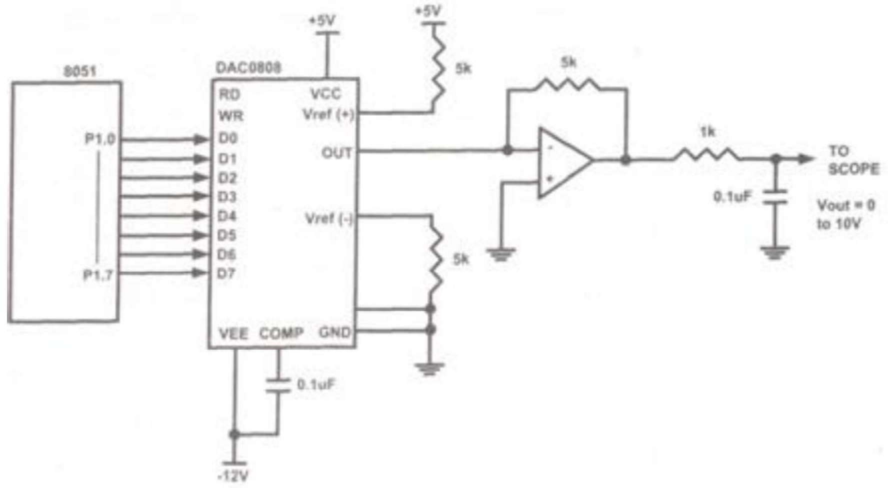


Fig 5.26: 8051 connection to DAC808

Figure 5.26 shows the generation of current reference (setting I_{ref} = 2 mA) by using the standard 5-V power supply and IK and 1.5K-ohm standard resistors. Some DACs also use the zener diode (LM336), which overcomes any fluctuation associated with the power supply voltage. Now assuming that $I_{re}f$ = 2 mA, if all the inputs to the DAC are high, the maximum output current is 1.99 mA

Converting I_{out} to voltage in DAC0808

Ideally we connect the output pin I_{out} to a resistor, convert this current to voltage, and monitor the output on the scope. In real life, however, this can cause inaccuracy since the input resistance of the load where it is connected will also affect the output voltage. For this reason, the I_{ref} current output is isolated by connecting it to an op-amp such as the 741 with R_f = 5K ohms for the feedback resistor. Assuming that R = 5K ohms, by changing the binary input, the output voltage changes.

5.5.3 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, but 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 LM34 requires no external calibration since it is internally calibrated. It outputs 10 mV for each degree of Fahrenheit temperature.

The LM35 series sensors are precision integrated-circuit temperature sensors whose output voltage is linearly proportional to the Celsius (centigrade) temperature. The LM35 requires no external calibration since it is internally calibrated. It outputs 10 mV for each degree of centigrade temperature.

LM35 Temperature sensor series selection guide:

Part	Temperature Range	Accuracy	Output Scale
LM35A	-55C to + 150 C	+1.0 C	10 mV/C
LM35	-55C to + 150 C	+1.5 C	10 mV/C
LM35 CA	-40C to + 110 C	+1.0 C	10 mV/C
LM35 C	-40C to + 110 C	+1.5 C	10 mV/C
LM35 D	0C to + 100 C	+2.0 C	10 mV/C

Signal conditioning and interfacing the LM35 to the 8051:

Signal conditioning is widely used in the world of 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 is commonly called signal conditioning. Signal conditioning can be a current-to-voltage conversion or a signal amplification. For example, the thermistor changes resistance with temperature. The change of resistance must be translated into voltages in order to be of any use to an ADC.

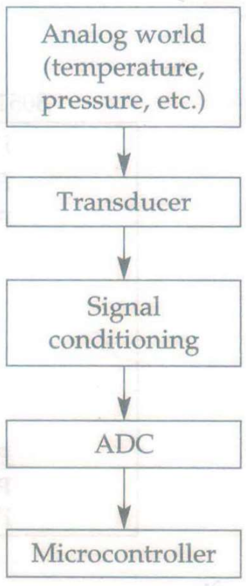


Fig 5.27: Getting data from analog world

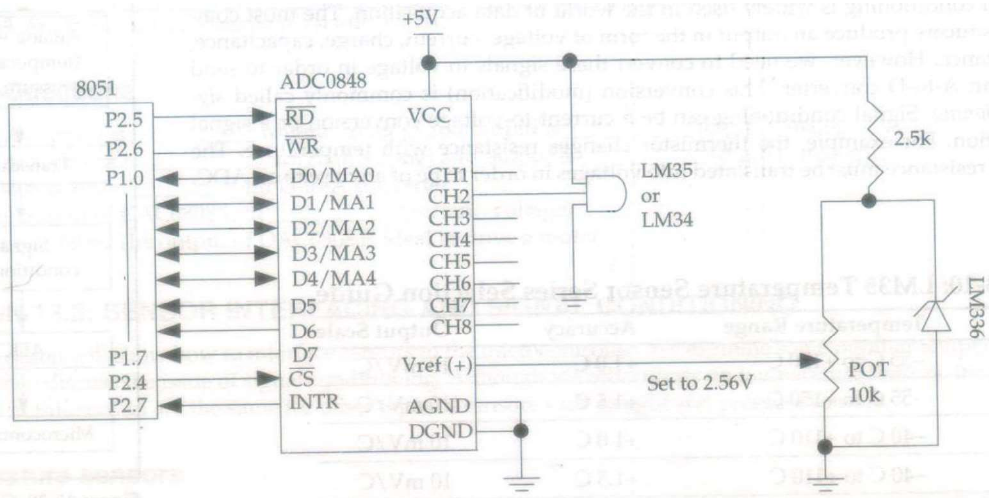


Fig 5.28: 8051 connection toADC0848 and temperature sensor

5.6 EXTERNAL MEMORY INTERFACE

5.6.1 Memory Address Decoding

The CPU provides the address of the data desired, but it is the job of the decoding circuitry to locate the selected memory block.

Memory chips have one or more pins called CS (chip select), which must be activated for the memory's contents to be accessed. Sometimes the chip select is also referred to as chip enable (CE). In connecting a memory chip to the CPU, note the following points.

1. The data bus of the CPU is connected directly to the data pins of the memory chip.
2. Control signals RD (read) and WR (memory write) from the CPU are connected tothe OE (output enable) and WE (write enable) pins of the memorychip, respectively.
3. In the case of the address buses, while the lower bits of the addresses from theCPU go directly to the memory chip address pins, the upper ones are used to activate the CS pin of the memory chip. It is the CS pin that along with RD/WR allows the flow of data in or out of the memory chip. No data can be written into or read from the memory chip unless CS is activated.

There are three ways to generate a memory block selector:

(a) using simple logic gates

(b) using the 74LS138

(c) using programmable logics.

Simple logic gate address decoder

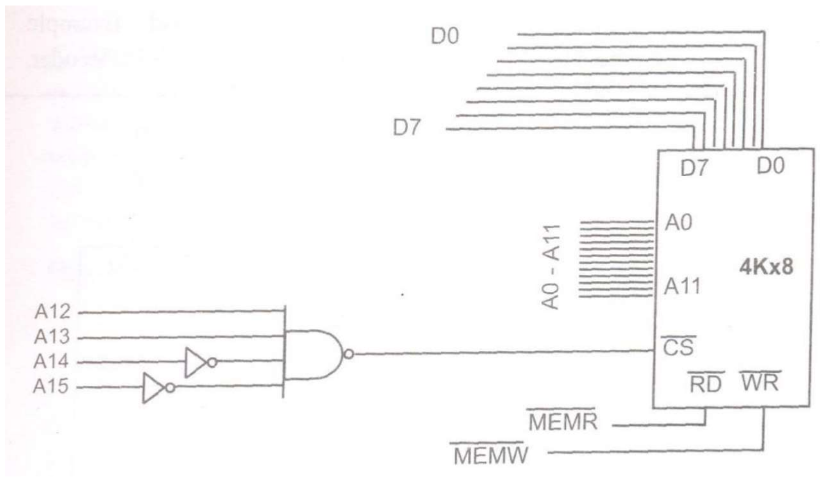


Fig.5.29: Logic gate as decoder

The simplest method of constructing decoding circuitry is the use of a NAND gate. The output of a NAND gate is active low, and the CS pin is also active low, which makes them a perfect match. In cases where the CS input is active high, an AND gate must be used. Using a combination of NAND gates and inverters, one can decode any address range.

Using the 74LS138 3-8 decoder

This is one of the most widely used address decoders. The 3 inputs A, B,and C generate 8 active-low outputs Y0 – Y7. Each Y output is connected to CS of a memory chip, allowing control of 8 memory blocks by a single 74LS138. In the 74LS138, where A, B, and C select which output is activated, there are three additional inputs, G2A, G2B, and Gl. G2A and G2B are both active low, and Gl is activehigh. If any one of the inputs Gl, G2A, or G2B is not connected to an address signal (sometimes they are connected to a control signal), they must be activated permanently either by V_{cc} or ground, depending on the activation level.

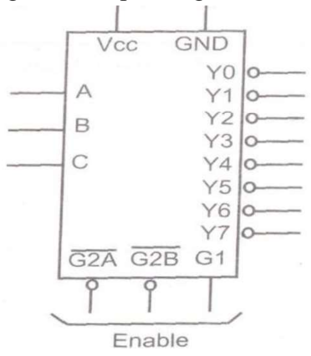
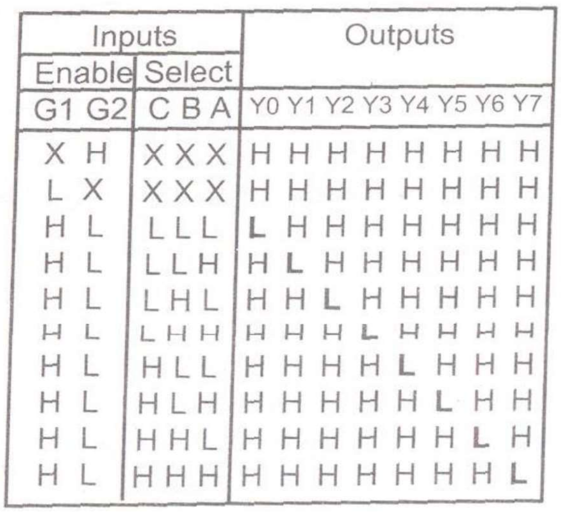


Fig.5.30: 74LS138 Block diagram



Inputs					Outputs							
Enable		Select										
G1	G2	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

Fig.5.31: 74LS138 Function Table

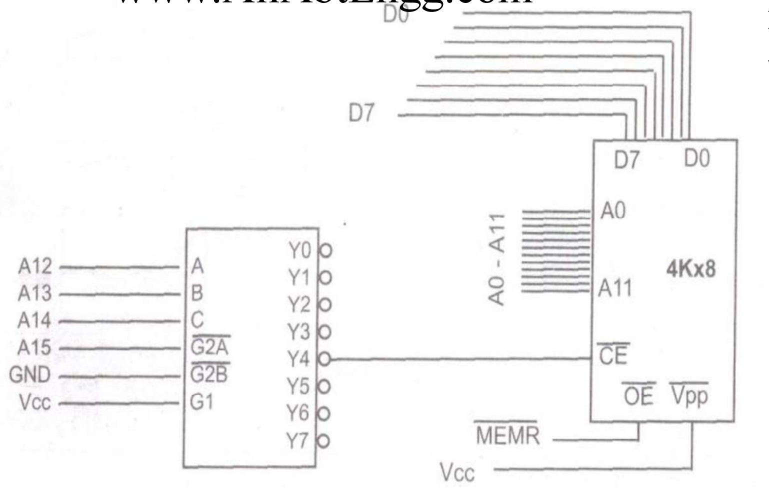


Fig.5.32: 74LS138 as decoder

Using programmable logic as an address decoder:

Some decoders use programmable logic chip such as PAL and GAL chips. One disadvantage of these chips is that they require PAL/GAL software and a programmer. The advantage of these chips is that they can be programmed for any combinations of address range.

5.6.2 8031/51 Interfacing with External Rom

In many systems where the on-chip ROM of the 8051 is not sufficient, the use of an 8031 is ideal since it allows the program size to be as large as 64K bytes. Although the 8031 chip itself is much cheaper than other family members, an 8031-based system is much more expensive since the ROM containing the program code is connected externally and requires more supporting circuitry.

EA pin

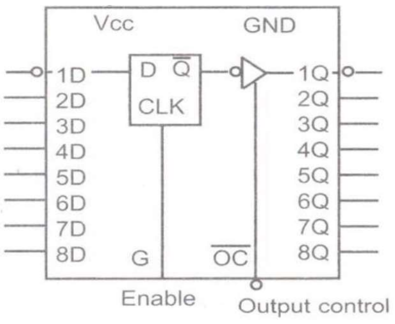
For 8751/89C51/DS5000-based systems, we connect the EA pin to V_{cc} to indicate that the program code is stored in the microcontroller's on-chip ROM. To indicate that the program code is stored in external ROM, this pin must be connected to GND.

P0 and P2 role in providing addresses

Since the PC (program counter) of the 8031/51 is 16-bit, it is capable of accessing up to 64K bytes of program code. In the 8031/51, port 0 and port 2 provide the 16-bit address to access external memory. Of these two ports, P0 provides the lower 8 bit addresses A0 – A7, and P2 provides the upper 8 bit addresses A8 – A15. More importantly, P0 is also used to provide the 8-bit data bus D0 – D7. In other words, pins P0.0 – P0.7 are used for both the address and data paths. This is called address/data multiplexing in chip design. Of course the reason Intel used address/data multiplexing in the 8031/51 is to save pins.

This is the job of the ALE (address latch enable) pin to check when P0 is used for the data path and when it is used for the address path. ALE is an output pin for the 8031/51 microcontroller. Therefore, when ALE = 0 the 8031 uses P0 for the data path, and when ALE = 1, it uses it for the address path. As a result, to extract the addresses from the P0 pins we connect P0 to a 74LS373 latch (see Figure 5.33) and use the ALE pin to latch the address as shown in Figure. This extracting of addresses from P0 is called address/data demultiplexing.

From Figure, it is important to note that normally ALE = 0, and P0 is used as a data bus, sending data out or bringing data in. Whenever the 8031/51 wants to use P0 as an address bus, it puts the addresses A0 – A7 on the P0 pins and activates ALE = 1 to indicate that P0 has the addresses.



Funtion Table

Output control	Enable G	D	Output
L	H	H	H
L	H	L	L
L	L	X	Q0
H	X	X	Z

Fig.5.33 74LS138 D Latch

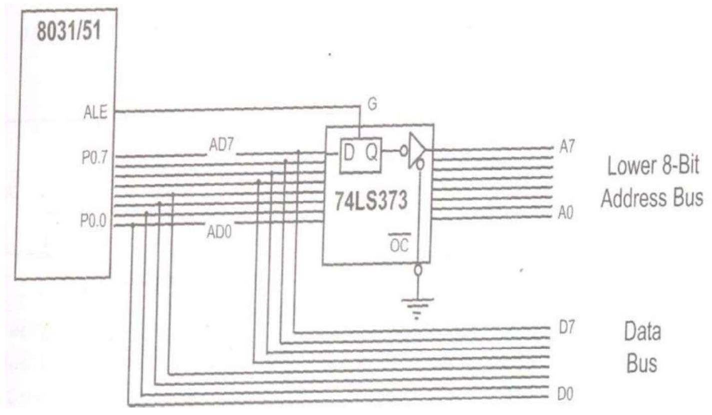


Fig.5..34: Address/ Data multiplexing

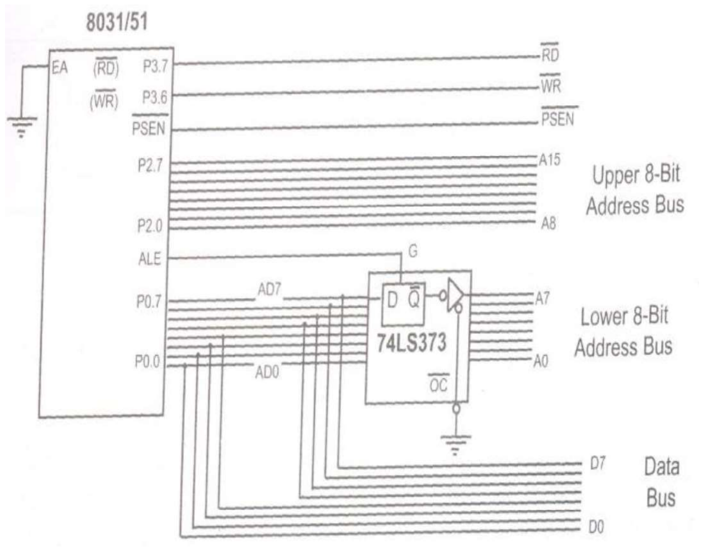


Fig.5.35: Data, address, control buses for 8051

PSEN

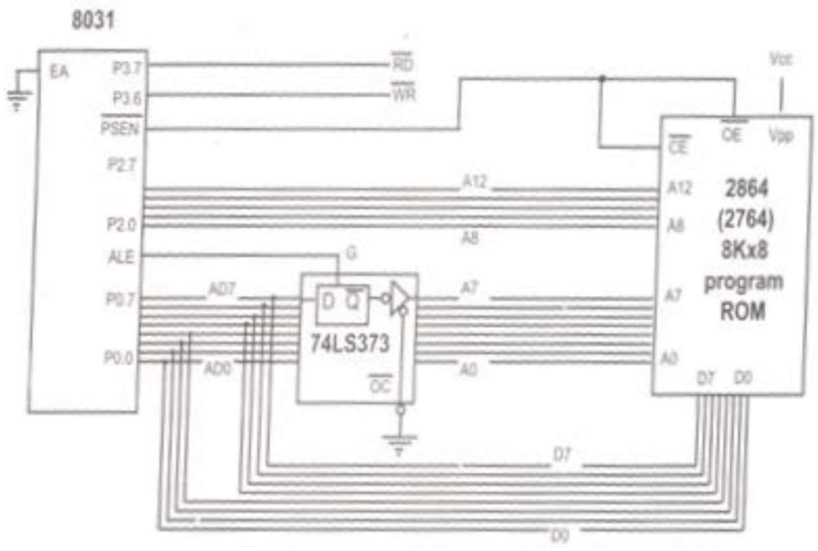


Fig.5.36 8031 connection to external program ROM

To access external ROM containing program code, the 8031/51 uses the PSEN signal. It is important to emphasize the role of EA and PSEN when connecting the 8031/51 to external ROM. When the EA pin is connected to GND, the 8031/51 fetches opcode from external ROM by using PSEN.

In systems where the external ROM contains the program code, burning the program into ROM leaves the microcontroller chip untouched. This is preferable in some applications due to flexibility. In such applications the software is updated via the serial or parallel ports of the IBM PC. This is especially the case during software development and this method is widely used in many 8051-based trainers and emulators.

5.6.3 8051 data memory space

We have stated that the program counter in the 8051 is 16-bit and therefore can access up to 64K bytes of program code. The instruction "MOVC A, @A+DPTR" to get the data. The MOVC instruction, where C stands for code, indicates that data is located in the code space of the 8051. In the 8051 family there is also a separate data memory space.

In addition to its code space, the 8051 family also has 64K. bytes of data memory space. In other words, the 8051 has 128K bytes of address space of which 64K bytes are set aside for program code and the other 64K bytes are set aside for data. Program space is accessed using the program counter (PC) to locate and fetch instructions, but the data memory space is accessed using the DPTR register and an instruction called MOVX, where X stands for external (meaning that the data memory space must be implemented externally).

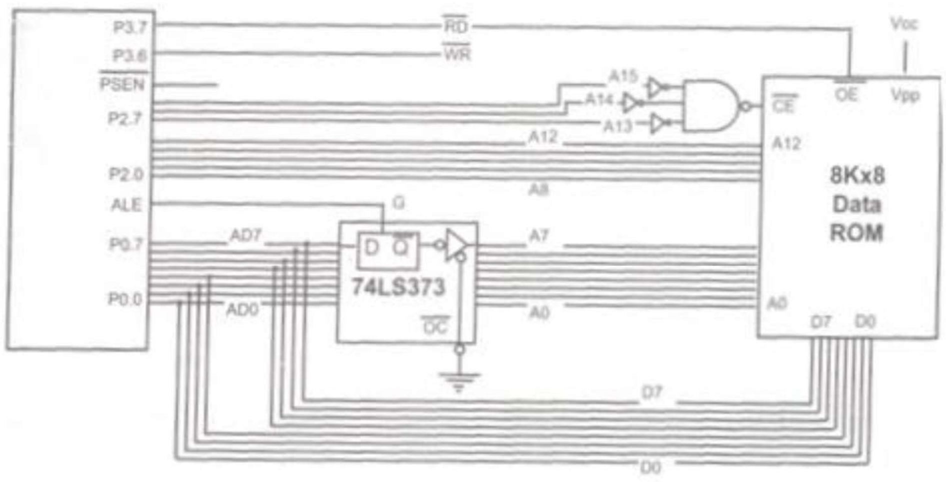


Fig.5.37: 8051 connection to external data ROM

External ROM for data

To connect the 8031/51 to external ROM containing data, we use RD (pin P3.7). For the ROM containing data, the RD signal is used to fetch the data. To access the external data memory space we must use the instruction MOVX .

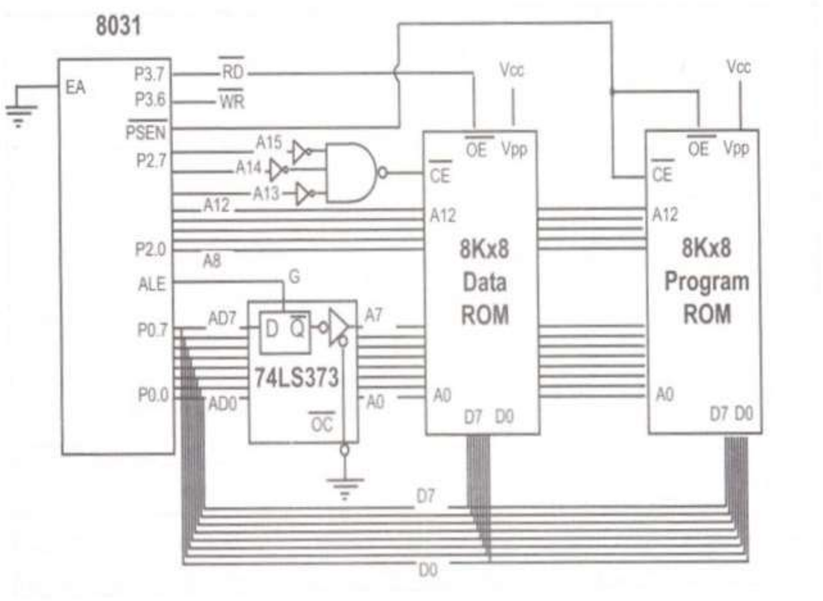


Fig.5.38: 8051 connection to external data ROM and external program ROM

External data RAM

To connect the 8051 to an external SRAM, we must use both RD (P3.7) and WR (P3.6).

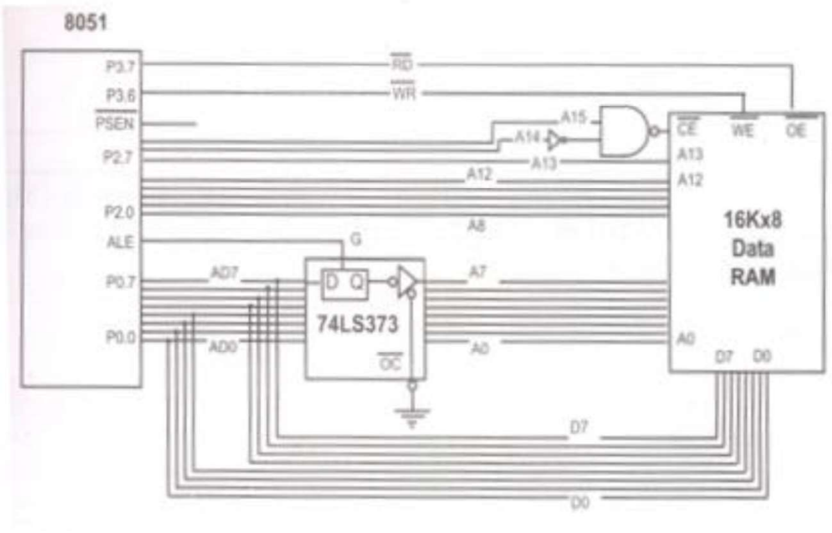


Fig.5.39: 8051 Connection to External Data RAM

MOVX instruction for external RAM data

In writing data to external data RAM, we use the instruction "MOVX @DPTR, A" where the contents of register A are written to external RAM whose address is pointed to by the DPTR register. This has many applications, especially where we are collecting a large number of bytes of data. In such applications, as we collect data we must store them in NV-RAM so that when power is lost we do not lose the data.

5.6.4 A single external ROM for code and data

Assume that we have an 8031 -based system connected to a single 64Kx8 (27512) external ROM chip. This single external ROM chip is used for both program code and data storage. For example, the space 0000 – 7FFFH is allocated to program code, and address space D000H – FFFFH is set aside for data. In accessing the data, we use the MOVX instruction. How do we connect the PSEN and RD signals to such a ROM? Notice in the previous discussion that PSEN is used to access the external code space and the RD signal is used to access the external data space. To allow a single ROM chip to provide both program code space and data space, we use an AND gate to signal the OE pin of the ROM chip as shown in Figure.

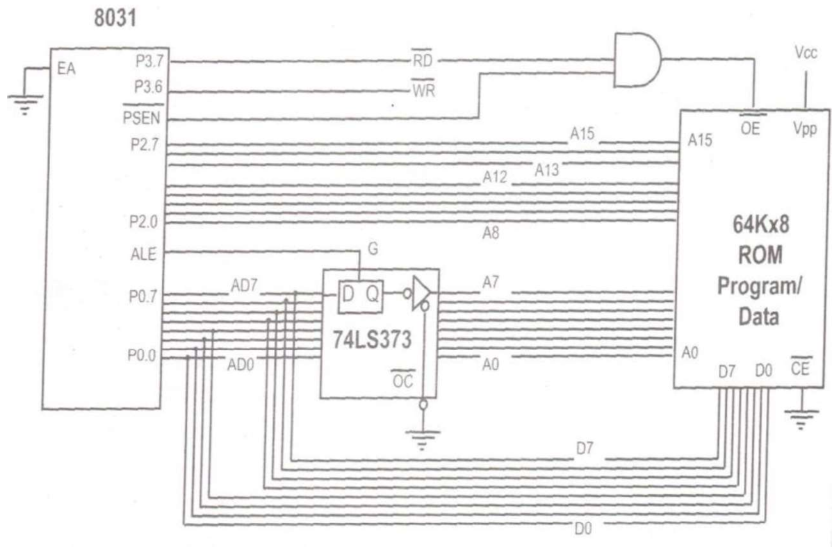


Fig.5.40: A single ROM for both program and data

8031 system with ROM and RAM

There are times that we need program ROM, data ROM, and data RAM in a system.

5.6.5 Interfacing to large external memory

In some applications we need a large amount (256K. bytes, for example) of memory to store data. However, the 8051 can support only 64K bytes of external data memory since DPTR is 16-bit. To solve this problem, we connect A0 – A15 of the 8051 directly to the external memory's A0 – A15 pins, and use some of the P1 pins to access the 64K-byte blocks inside the single 256Kx8 memory chip.

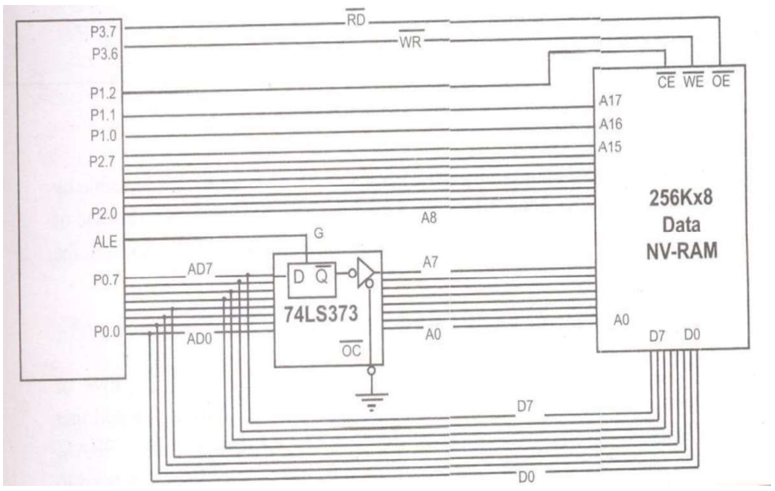


Fig 5.41 8051 Accessing 256x8 External NV_RAM

5.7STEPPER MOTOR

A stepper motor is a widely used device that translates electrical pulses into mechanical movement. In applications such as disk drives, dot matrix printers, and robotics, the stepper motor is used for position control. Stepper motors commonly have a permanent magnetrotor (also called the shaft) surrounded by a stator.

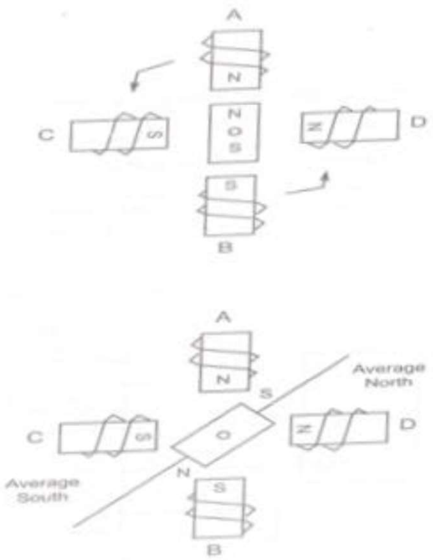


Fig.5.42: Rotor Alignment

There are also steppers called variable reluctance stepper motorsthat do not have a PM rotor. The most common stepper motors have four stator windings that are paired with a center-tapped common.

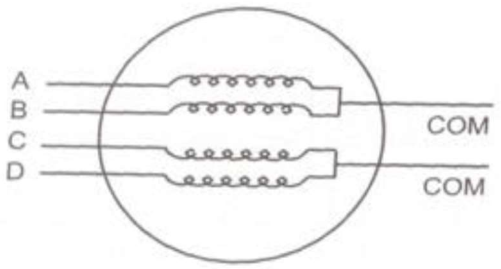


Fig.5.43: Stator windings configuration

This type of stepper motor is commonly referred to as a. four-phase or unipolar stepper motor. The center tap allows a change of current direction in each oftwo coils when a winding is grounded, thereby resulting in a polarity change of the stator.

Notice that while a conventional motor shaft runs freely, the stepper motor shaft moves in a fixed repeat-able increment, which allows one to move it to a precise position. This repeatable fixed movement is possible as a result of basic magnetic theorywhere poles of the same polarity repeland opposite poles attract.

The direction of the rotation is dictated by the statorpoles. The stator poles are determined by the current sent through the wire coils.As the direction of the current is changed, the polarity is also changed causing thereverse motion of the rotor.

The stepper motor discussed here has a total of 6leads: 4 leads representing the four stator windings and 2 commons for thecenter-tapped leads. As the sequence of power is applied to each stator winding,the rotor will rotate. There are several widely used sequences where each has adifferent degree of precision.

It must be noted that although we can start with any of the sequences in Table, once we start we must continue in the proper order. For example, if we start with step 3 (0110), we must continue in the sequence of steps 4, 1,2, etc.

5.7.1 Normal 4-step sequence

CLOCK WISE	STEP #	WINDING A	WINDING B	WINDING C	WINDING D	COUNTER CLOCKWISE
↓	1	1	0	0	1	↑
	2	1	1	0	0	
	3	0	1	1	0	
	4	0	0	1	1	

5.7.2 Half step 8 step sequence

CLOCK WISE	STEP #	WINDING A	WINDING B	WINDING C	WINDING D	COUNTER CLOCKWISE
↓	1	1	0	0	1	↑
	2	1	0	0	0	
	3	1	1	0	0	
	4	0	1	0	0	
	5	0	1	1	0	
	6	0	0	1	0	
	7	0	0	1	1	
	8	0	0	0	1	

Step angle

The step angle is the minimum degree of rotation associated with a single step. Various motors have different step angles.

How much movement is associated with a single step? This depends on the internal construction of the motor, in particular the number of teeth on the stator and the rotor.

Steps per second and rpm relation

The relation between rpm (revolutions per minute), steps per revolution, and steps per second is as follows.

$$\text{Steps per second} = \frac{\text{rpm x Steps per revolution}}{60}$$

The four-step sequence and number of teeth on rotor

The switching sequence shown earlier in Table is called the 4-step switching sequence since after four steps the same two windings will be "ON" How much movement is associated with these four steps? After completing every four steps, the rotor moves only one tooth pitch. Therefore, in a stepper motor with 200 steps per revolution, the rotor has 50 teeth since $4\times50 = 200$ steps are needed to complete one revolution. This leads to the conclusion that the minimum step angle is always a function of the number of teeth on the rotor. In other words, the smaller the step angle, the more teeth the rotor passes.

Motor speed

The motor speed, measured in steps per second (steps/s), is a function of the switching rate.

Holding torque

"With the motor shaft at standstill or zero rpm condition, the amount of torque, from an external source, required to break away the shaft from its holding position. This is measured with rated voltage and current applied to the motor." The unit of torque is ounce-inch (or kg-cm).

5.7.3 Unipolar versus bipolar stepper motor interface

There are three common types of stepper motor interfacing: universal, unipolar, and bipolar. They can be identified by the number of connections to the motor. A universal stepper motor has eight, while the unipolar has six and the bipolar has four. The universal stepper motor can be configured for all three modes, while the unipolar can be either unipolar or bipolar. Obviously the bipolar cannot be configured for universal nor unipolar mode. Table shows selected stepper motor characteristics. Figure shows the basic internal connections of all three type of configurations.

Unipolar stepper motors can be controlled using the basic interfacing shown in Figure , whereas the bipolar stepper requires H-Bridge circuitry. Bipolar stepper motors require a higher operational current than the unipolar; the advantage of this is a higher holding torque.

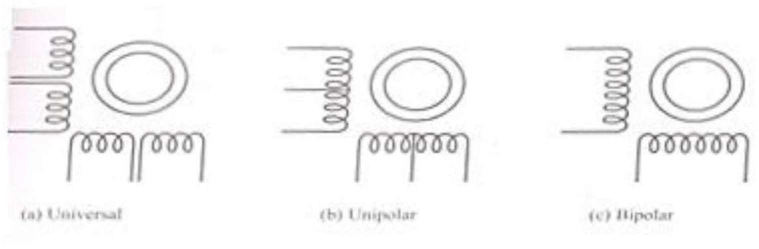


Fig.5.44: Common stepper motor size

5.7.4 Using transistors as drivers

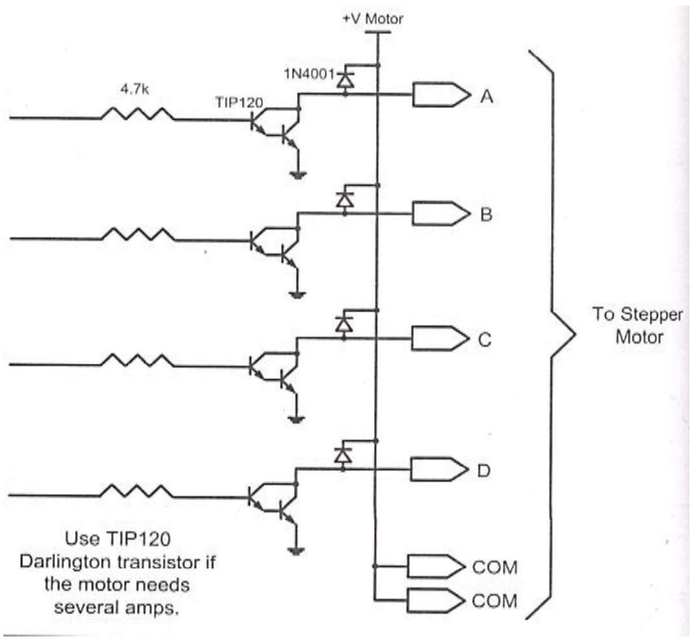

+V Motor
1N4001
4.7k
TIP120
A
B
C
D
COM
COM
To Stepper Motor
Use TIP120 Darlington transistor if the motor needs several amps.

Fig.5.47: Using transistors for stepper motor driver

Figure shows an interface to a unipolar stepper motor using transistors. Diodes are used to reduce the back EMF spike created when the coils are energized and de-energized. TIP transistors can be used to supply higher current to the motor.

5.7.5 Controlling stepper motor via optoisolator

Optoisolators are widely used to isolate the stepper motor's EMF voltage and keep it from damaging the digital/microcontroller system.

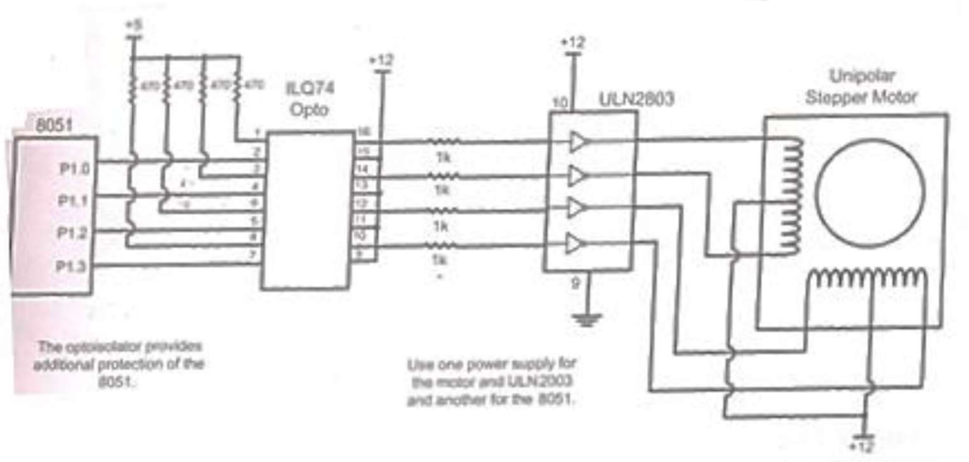

+5
470
ILQ74
Opto
+12
8051
P1.0
P1.1
P1.2
P1.3
1k
ULN2803
Unipolar
Stepper Motor
The optoisolator provides additional protection of the 8051.
Use one power supply for the motor and ULN2003 and another for the 8051.

Fig 5.48: Controlling stepper motor via optoisolator

Program:

```
         MOV R0, #COUNT        ;initialize rotation count
AGAIN:   MOV DPTR, #ETC        ;initialize pointer to excitation code table
         MOV R1,#04            ;initialize counter to excitation code sequence
BACK:    MOVX A, @DPTR         ;get the excitation code
         MOV P1,A              ;send the excitation code
         LCALL DELAY           ;wait for some time
         INC DPTR              ;increment pointer
         DJNZ R1,BACK          ;decrement R1, if not zero go to BACK
         DJNZ R0, AGAIN        ; decrement R0, if not zero go to AGAIN
         RET
         ORG 3000H
ETC:     DB 03H, 06H, 09H, 0CH ;code sequence for clockwise rotation
```

5.8 WAVEFORM GENERATION

Generating a sine wave

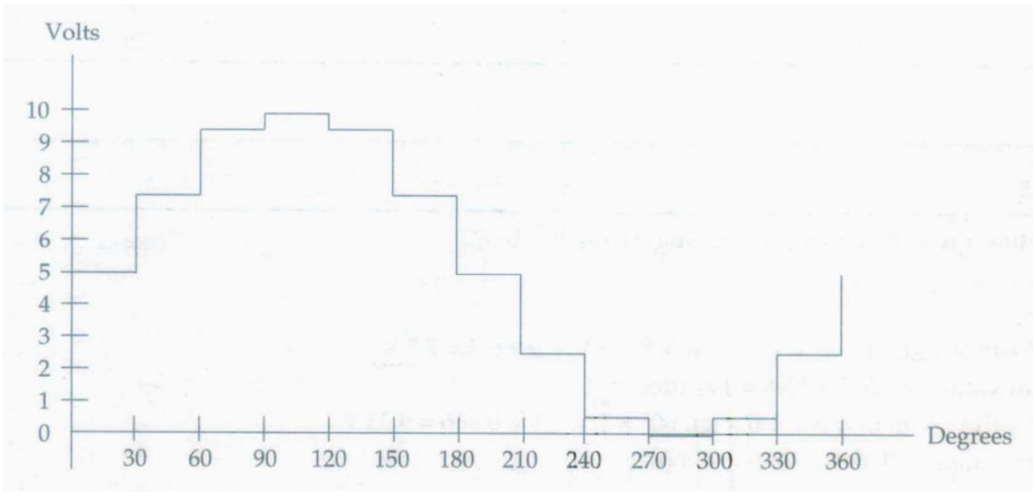
To generate a sine wave, we first need a table whose values represent the magnitude of the sine of angles between 0 and 360 degrees. The values for the sine function vary from -1.0 to +1.0 for 0- to 360-degree angles. Therefore, the table values are integer numbers representing the voltage magnitude for the sine of theta. This method ensures that only integer numbers are output to the DAC by the 8051 microcontroller. Table shows the angles, the sine values, the voltage magnitudes, and the integer values representing the voltage magnitude for each angle (with 30-degree increments). To generate Table , we assumed the full-scale voltage of 10 V for DAC output (as designed in Figure . Full-scale output of the DAC is achieved when all the data inputs of the DAC are high. Therefore, to achieve the full-scale 10 V output, we use the following equation.

$$V_{out} = 5V + (5x \sin\theta)$$

V_{out} of DAC for various angles is calculated and shown in Table .

Angle θ (degrees)	**Sin θ**	**V_{out} (Voltage Magnitude) 5 V + (5 V X sin θ)**	**Values Sent to DAC (decimal) (Voltage Mag. X 25.6)**
0	0	5	128
30	0.5	7.5	192
60	0.866	9.33	238
90	1.0	10	255
120	0.866	9.33	238
150	0.5	7.5	192
180	0	5	128
210	-0.5	2.5	64
240	-0.866	0.669	17
270	-1.0	0	0
300	-0.866	0.669	17
330	-0.5	2.5	64
360	0	5	128

To find the value sent to the DAC for various angles, we simply multiply the V_{out} voltage by 25.60 because there are 256 steps and full-scale V_{out} is 10 volts. Therefore, 256 steps /10 V = 25.6 steps per volt.



LIST OF QUESTIONS

PART A

1. List the types of serial communication.

Serial data communication uses two methods,

- Synchronous and
- Asynchronous.

Synchronous:

The synchronous method transfers a block of data (characters) at a time.

Asynchronous:

Asynchronous data communication is widely used for character-oriented transmissions i.e, transfers a single byte at a time

The data coming in at the receiving end of the data line in a serial data transfer is all 0s and 1s; 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.

2. Write down the different operating modes for serial communication of 8051.

SM0	SM1	Mode	Baud Rate
0	0	Serial mode 0 (8 data bits)	1/12th of the oscillator frequency
0	1	Serial mode 1(1 bit for start, 8 bits of data, 1 bit for stop)	Baud rate is variable, depends on timer 1 overflow rate.