UNIT - 2

Embedded Hardware Design

1. Analog and Digital electronic components :-

Analog Electronic components :-

* Resistors, capacitors, diodes, inductors, operational amplifiers (OP Amps), transistors, etc. are the commonly used analog electronic components in embedded hardware design.
* A Resistor limits the current flowing through a circuit. Interfacing of LEDs, buzzer etc.
* Capacitors and inductors are used in signal filtering and restorating ckts Reset circuit implementation, matching circuits of RF designs, power supply decapling etc.
* P-N junction diode, Schottky diode, Zener diode etc. are the commonly used diodes in embedded hardware circuits.
  - ↳ These are used for voltage clamping applications, Reverse polarity protection, voltage rectification, freewheeling of current produced in inductive ckts, etc
* Transistors in embedded applications are used for either switching or amplification purpose.
  - ↳ In switching application, the transistor is in either ON or OFF state.
  - ↳ In Amplification operation, the transistor is always in the ON state

Digital electronic components :-

* Digital electronics deal with digital or discrete signals.
* Microprocessors, microcontrollers and systems on chips (SOCs) work on digital principles.
* Embedded systems employ various digital electronic circuits for 'glue logic' implementation.
* 'glue logic' is the custom digital electronic circuitry required to achieve compitable interface b/w two different integrated circuit chips.
  - Address decoders, latches, encoders/ decoders etc. are examples of glue logic
* Transistor Transistor Logic (TTL), complementary metal oxide semiconductor (CMOS) logic etc are some of the standards describing the electrical characteristics of digital signals in a digital system.
* The following sections give an overview of the various digital I/O interface standards and the digital circuits/ components used in embedded systems development

(i) open collector and Tri-state output

(ii) Logic Gates — Basic building blocks of digital ckts
Ex:- AND, OR, XOR, NOT, NAND, NOR, XNOR ..

(iii) Buffer — Logic circuit for amplifying the current or power.

(iv) Latch — used for storing binary data
- +ve edge, -ve edge.

(v) Encoder — The encoder encodes the corresponding input state to a particular output format. Ex:- 4 to 2, 8 to 3, 16 to 4 encoders.

(vi) Decoder — A decoder is a logic circuit which generates all possible combinations of the input signals.
Ex: 2 to 4, 3 to 8, 4 to 16 decoders.

(vii) Multiplexer – A digital switch connects one input line from a set of input lines, to an output line at a given point of time

Ex:- 74S151 – 8 to 1 MUX

(viii) De-Multiplexer – De-Multiplexer switches the input signal to the selected output line among a number of output lines.

Ex:- 1 to 2 DE-MUX.

(ix) Combinational circuits :– A combinational circuit is a combination of the logic gates. The output of the combinational circuit, at a given point of time, is dependent only on the state of the inputs at the given point of time.

Ex:- Encoders, decoders, multiplexers, de-multiplexers, adder circuits, comparators, multiple i/p gates etc.

(x) Sequential circuits :- Digital logic circuit, whose output at any given point of time depends on both the present and past inputs, is known as sequential circuits.

Ex:- SR-FF, JK-FF, D, T, Flip-flops.

3.1 IO TYPES AND EXAMPLES

A serial port is a port for serial communication. Serial communication means that over a given line or channel one bit can communicate and the bits transmit at periodic intervals generated by a clock. A serial port communication is over short or long distances.

A parallel port is a port for parallel communication. Parallel communication means that multiple bits can communicate over a set of parallel lines at any given instance. A parallel port communicates within the same board, between ICs or wires over very short distances of at most less than a meter.

A serial or parallel port can provide certain special features and sophistication (Section 3.4) by using a processing element.

Ports can interconnect by wireless. Wireless or mobile communication is serial communication but without wires, can be over a short-range personal area network as well as long-range wireless network, and transmission takes place by using carrier frequencies. The carrier modulates the serial bits before transmission in air [Sections 3.5 and 3.13]. A receiver demodulates and retrieve the serial bits back.

Serial and parallel ports of IO devices can be classified into following IO types: (i) Synchronous serial input (ii) Synchronous serial output (iii) Asynchronous serial UART input (iv) Asynchronous serial UART output (v) Parallel port one bit input (vi) Parallel one bit output (vii) Parallel port input (viii) Parallel port output. Some devices function both as input and as output; for example, a modem.

3.1.1 Synchronous Serial Input

The part 1 in Figure 3.1(a) shows a synchronous input serial port. Each bit in each byte and each received byte is in synchronization. Synchronization means separation by a constant interval or phase difference [part 2 in Figure 3.1(a)]. If clock period equals T, then each byte at the port is received at input in period 8 T. The bytes are received at constant rates. Each byte at the input port separates by 8 T and data transfer rate for the serial line bits is 1/T bps [1 bps = 1-bit per second]. The sender, along with the serial bits, also sends the clock pulses SCLK (serial clock) to the receiver port pin. The port synchronizes the serial data-input bits with clock bits.

The serial data input and clock pulse-input are on same input line when the clock pulses either encode or modulate serial data input bits suitably. The receiver detects clock pulses and receives data bits after decoding or demodulating.

When a separate SCLK input is sent, the receiver detects at the middle, positive or negative edge of the clock pulses that indicate whether data-input is 1 or 0 and saves the bits in 8-bit shift register. The processing element at the port (peripheral) saves the byte at a port register from where the microprocessor reads the byte.

Synchronous serial input is also called master output slave input (MOSI) when the SCLK is sent from the sender to the receiver and slave is forced to synchronize sent inputs from the master as per the master clock inputs. Synchronous serial input is also called master input slave output (MISO) when the SCLK is sent to the sender (slave) from the receiver (master) and the slave is forced to synchronize sending the inputs to master as per the master clock's outputs.

Synchronous serial input is used for interprocessor transfers, audio inputs and streaming data inputs.

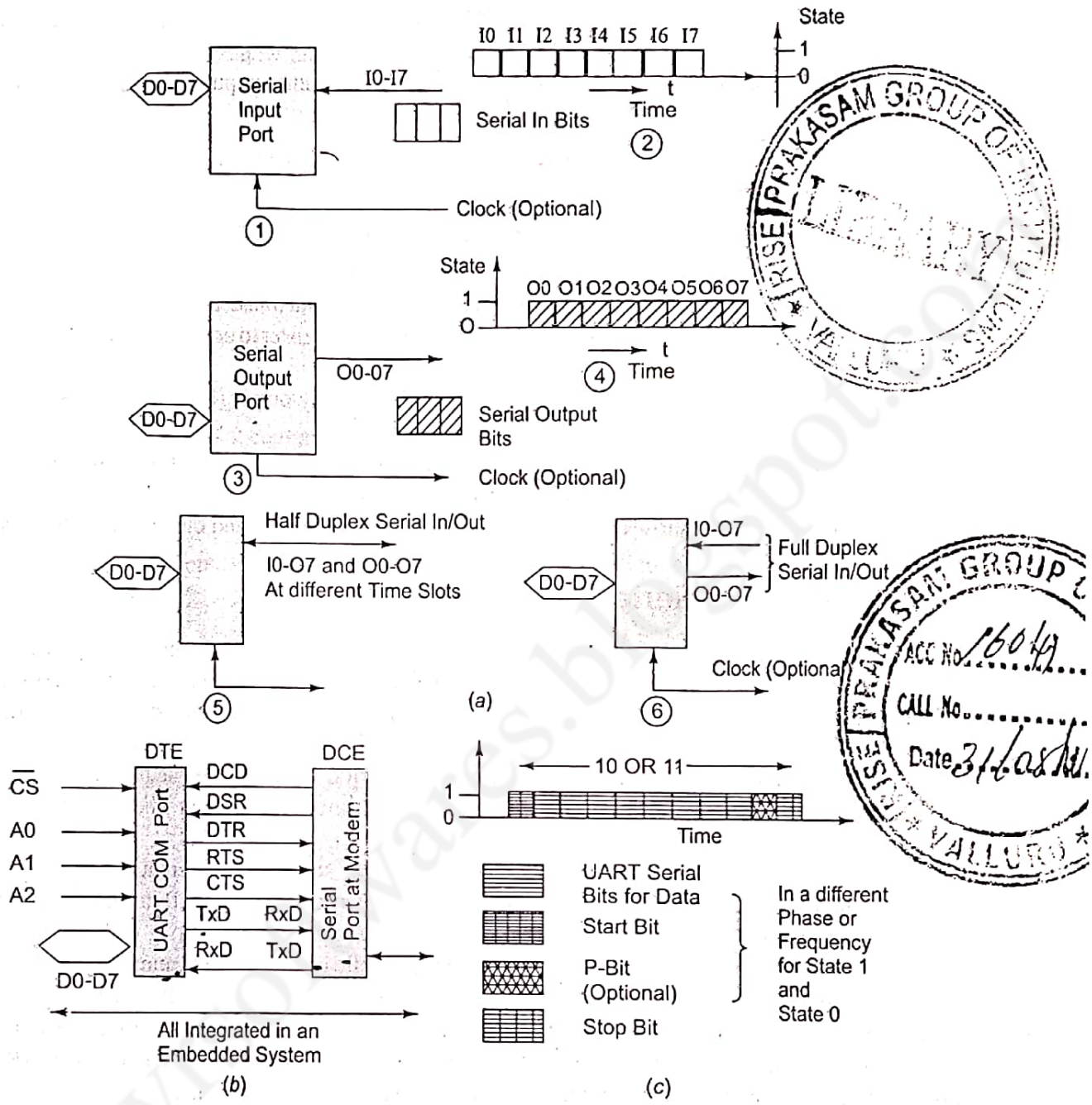


Fig. 3.1 (*a*) Input serial port, Output Serial port, Bi-directional half-duplex serial port, and Bi-directional full-duplex serial port (*b*) Handshaking signals at COM port in computer and (*c*) a UART serial port bits

3.1.2 Synchronous Serial Output

The part 3 in Figure 3.1(a) shows a synchronous output serial port. Each bit in each byte is in synchronization with a clock. The bytes are sent at constant rates [part 4 in Figure 3.1(a)]. If the clock period equals T, then the data transfer rate is 1/T bps. The sender sends either the clock pulses at SCLK pin or the serial data output and clock pulse-input through same output line when the clock pulses either suitably modulate or encode the serial output bits.

The processing element at the port (peripheral) sends the byte through a shift register at the port to which the microprocessor writes the byte.

Synchronous serial output is used for interprocessor transfers, audio outputs and streaming data outputs.

3.1.3 Synchronous Serial Input–Output

The part 5 in Figure 3.1(a) shows a synchronous serial input–output port. Each bit in each byte synchronizes with the clock input and output. The bytes are sent or received at constant rates as shown in parts (2) and (4) in Figure 3.1(a)]. The IOs are on same IO line when the clock pulses suitably modulate or encode the serial input and output, respectively. If the clock period equals T, then the data transfer rate is 1/ T bps. The processing element at the port (peripheral) sends and receives the byte at a port register to or from which the microprocessor writes or reads the byte.

Synchronous serial input/outputs are also called master input slave output (MISO) and master output slave input (MOSI), respectively.

They are used for interprocessor transfers and streaming data. The bits are read from or written on magnetic media such as a hard disk or on optical media such as a CD by using devices with serial synchronous IO ports.

The part 6 in Figure 3.1(a) shows the IO synchronous port when input and output lines are separate.

3.1.4 Asynchronous Serial input

Figure 3.1(b) shows the asynchronous input serial port line, denoted by RxD (receive data). Each RxD bit is received in each byte at fixed intervals but each received byte is not in synchronization. The bytes can separate by variable intervals or phase differences. Figure 3.1(c), on the right side, shows the starting point of receiving the bits for each byte, indicated by a line transition from 1 to 0 for a period T. When a sender shifts after every clock period T, then a byte at the port is received at input in period 10 T or 11 T. The time of 2 T is due to use of additional bits at the start and end of each byte. An addition time of 1T is taken when a P-bit is sent before the stop bit.

The bit transfer rate (for the serial line bits) is (1/ T) baud per second but different bytes may be received at varying intervals. The word 'Baud' is taken from a German word for raindrop. Bytes pour from the sender like raindrops at irregular intervals. The sender does not send the clock pulses along with the bits.

The receiver detects *n* bits at the intervals of T from the middle of the first indicating bit. n = 0, 1, ..., 10 or 11, finds out whether the data-input is 1 or 0 and saves the bits in an 8-bit shift register. The processing element at the port (peripheral) saves the byte at a port register, from where the microprocessor reads the byte.

Asynchronous serial input is also called UART input if the serial input is according to the UART protocol (Section 3.2.3). Asynchronous serial input is used for keypad and modem inputs.

3.1.5 Asynchronous Serial Output

Figure 3.1(b) shows the asynchronous output serial port line, denoted by TxD (transmit data). Each bit in each byte is sent at fixed intervals but each output byte is not in synchronization (it is separated by a variable interval or phase difference). The Figure 3.1(c) shows the starting point of sending the bits for each byte, which is indicated by a line transition from 1 to 0 for a period T. The sender port of TxD does not send clock pulses along with the bits.

The sender transmits bytes at the minimum intervals of n T. Bits start from the middle of the start indicating bit, where n = 0, 1, ..., 10 or 11 and sends the bits through a 10- or 11-bit shift register [Figure 3.1(c)]. The processing element at the port (peripheral) sends the byte at a port register to where the microprocessor writes the byte.

Asynchronous serial output is also called UART output if the serial output is according to a UART protocol (Section 3.2.3). Asynchronous serial output is used for modem and printer inputs.

3.1.6 Parallel Port

A parallel port can have one or multibit input or output and can be bi-directional IO.

(i) One bit input, output and IO
(ii) Eight or more bit input, output and IO

Section 3.3 will describe parallel device ports in detail.

3.1.7 Half Duplex and Full Duplex

The part 5 in Figure 3.1(a) on the left side shows the IO serial port (bi-directional half-duplex serial port). Half duplex means that at any point communication can only be one way (input or output) on a bi-directional line. An example of half-duplex mode is telephone communication. On one telephone line, we can talk only in the half-duplex mode. The part 6 in Figure 3.1(a) shows the separate input and output serial port lines. Full duplex means that the communication can be both ways simultaneously. An example of the full duplex asynchronous mode of communication is communication between the modem and computer through the TxD and RxD lines [Figure 3.1(b)].

There are two types of communication ports for IOs: serial and parallel. Serial line port communication is synchronous when a clock of the master device controls the synchronization of the bits on the line. Serial line port communication is asynchronous when clocks of the sender and receiver are independent and bytes are received, not necessarily at constant phase differences. Serial communication can be full duplex, which means simultaneously communication both ways, or half duplex, which means one way communication.

3.1.8 Examples of Serial and Parallel Port IOs

Table 3.1 gives a classification of IO devices into various types. It also gives examples of each type.

Table 3.1 Examples of various types of IO devices

IO Device Type	*Examples*
Serial synchronous input	Inter-processor data transfer, reading from CD or hard disk, audio input, video input, dial tone, network input, transceiver input, scanner input, remote controller input, serial IO bus input, reading from flash memory using SDIO (Secure Data Association IO) card
Serial synchronous output	Inter-processor data transfer, multiprocessor communication, writing to CD or hard disk, audio output, video output, dialer output, network device output, remote TV Control, transceiver output, and serial IO bus output, writing to flash memory using SDIO card
Serial asynchronous input	Keypad controller serial data-in, mice, keyboard controller data in, modem input, character inputs on serial line [also called UART (universal receiver and transmitter) input when according to UART mode]

(Contd)

IO Device Type	*Examples*
Serial asynchronous output	Output from modem, output for printer, the output on a serial line [also called UART output when according to UART mode]
Parallel port single bit input	(i) Completion of a revolution of a wheel, (ii) achieving preset pressure in a boiler, (iii) exceeding the upper limit of the permitted weight over the pan of an electronic balance, (iv) presence of a magnetic piece in the vicinity of or within reach of a robot arm to its end point and (v) filling a liquid up to a fixed level
Parallel port single bit output	(i) PWM output for a DAC, which controls liquid level, temperature, pressure, speed or angular position of a rotating shaft or a linear displacement of an object or a d.c. motor control (ii) pulses to an external circuit
Parallel port input	(i) ADC input from liquid level measuring sensor or temperature sensor or pressure sensor or speed sensor or d.c. motor rpm sensor (ii) Encoder inputs for bits for angular position of a rotating shaft or a linear displacement of an object
Parallel port output	(i) LCD controller for multiline LCD display matrix unit in a cellular phone to display on screen the phone number, time, messages, character outputs or pictogram bit-images or e-mail or web page (ii) print controller (iii) stepper motor coil driving output bits

3.2 SERIAL COMMUNICATION DEVICES

3.2.1 Synchronous, Iso-synchronous and Asynchronous Communications from Serial Devices

Synchronous Communication When a byte (character) or frame (a collection of bytes) of data is received or transmitted at constant time intervals with uniform phase differences, the communication is called *synchronous*. Bits of a data frame are sent in a fixed maximum time interval. ***Iso-synchronous*** is a special case when the maximum time interval can be varied.

An example of synchronous serial communication is frames sent over a LAN. Frames of data communicate, with the time interval between each frame remaining constant. Another example is the inter-processor communication in a multiprocessor system. Table 3.2 gives a synchronous device port bits.

Figure 3.1(a) part 2 showed the serial IO bit format and serial line states as a function of time. Two characteristics of synchronous communication are as follows:

1. Bytes (or frames) maintain a constant phase difference. It means they are synchronous, that is, in synchronization. There is no permission for sending either the bytes or the frames at random time intervals; this mode therefore does not provide for handshaking *during* the communication interval. [Handshaking means that the source and destination first exchange the signals between them before they communicate the data bits.] The master is the one whose clock pulses guide the transmission and slave is the one which synchronizes the bits as per the master clock.
2. A clock ticking at a certain rate must always be there to serially transmit the bits of all the bytes (or frames). The clock is not always implicit to the synchronous data receiver. The transmitter generally transmits the clock rate information in the synchronous communication of the data.

Table 3.2 Synchronous device port bits

S.No.	*Bits at Port*	*Compulsory or Optional*	*Explanation*
1.	Sync code bits or bi-sync code bits or frame start and end signaling bits	Optional	A few bits (each separated by interval ΔT) as Sync code for frame synchronization or signaling precedes the data bits[1]. There may be inversion of code bits after each frame. Flag bits at start and end are also used in certain protocols
2.	Data bits	Compulsory	*m* frame bits or 8 bits transmit such that each bit is at the line for time ΔT or each frame is at the line for time m.ΔT[2]
3.	Clock bits	Mostly not optional	Either on a separate clock line or on a single line such that the clock information is also embedded with the data bits by an appropriate encoding or modulation

[1]Reciprocal of ΔT is the transfer rate in bit per second (bps).
[2]m may be a large number. It depends on the protocol.

Figure 3.2 gives ten methods by which synchronous signals, with the clocking information, are sent. (i) There are two separate lines for the data bits and clock. The parallel-in serial-out (PISO) and serial-in parallel-out (SIPO) are used for transmitting and receiving the signals for data, respectively. (ii) There is a common line and the clock information is encoded by modulating the clock with the stream of bits. (iii) There are preceding and succeeding additional synchronizing and signaling bits. There are five common methods of encoding the clock information into a serial stream of the bits: (a) Frequency Modulation (FM (b) Mid Frequency Modulation (MFM) (c) Manchester coding (d) Quadrature amplitude modulation (QAM) (e) Bi-phase coding. The synchronous receiver separates serial bits of the message as well as synchronizing clock.

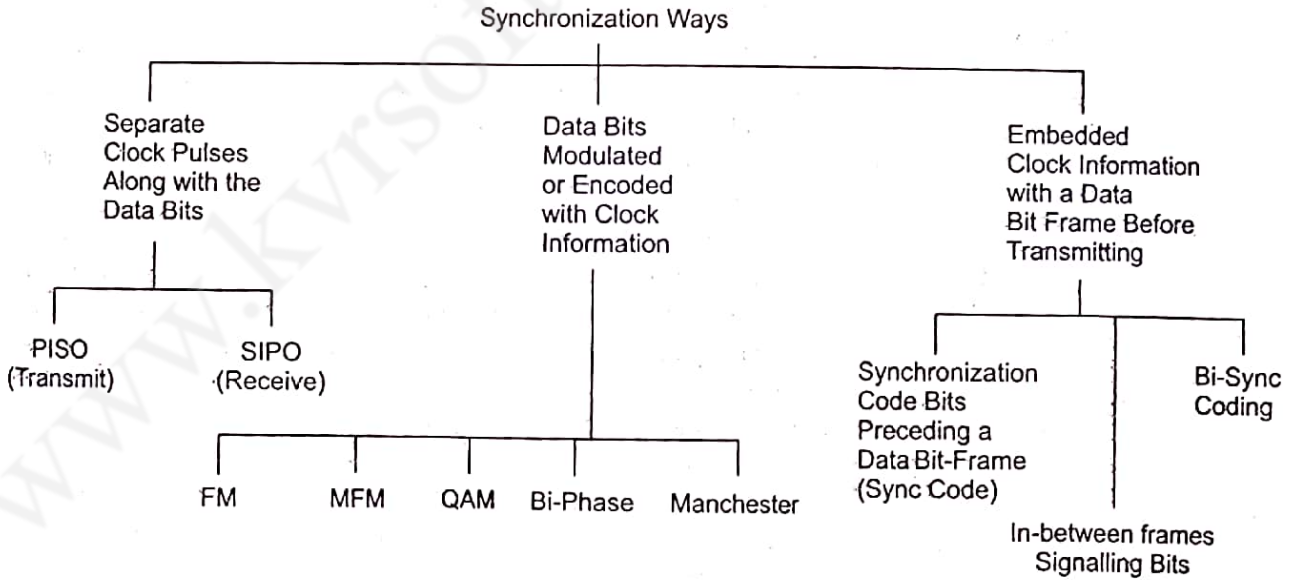


Fig. 3.2 Ten ways by which the synchronous signals with the clocking information transmit from a master device to slave device

Asynchronous Communication When a byte (characters) or frame (a collection of bytes) of data is received or transmitted at variable time intervals, communication is called *asynchronous.* Voice data on the line is sent in asynchronous mode. Over a telephone line the communication is asynchronous. Another example is keypad communication.

An example of mode of asynchronous communication is RS232C communication between the UART devices (Section 3.2.2).

UART communication (Section 3.2.3) for asynchronous data is used for the transfer of information between the keypad or keyboard and computer.

Two characteristics of asynchronous communication are as follows:

1. Bytes (or frames) need not maintain a constant phase difference and are asynchronous, that is, not in synchronization. Bytes or frames can be sent at variable time intervals. This mode therefore facilitates in-between handshaking between the serial transmitter port and serial receiver port.
2. Though the clock must tick at a certain rate to transmit bits of a single byte (or frame) serially, it is *always implicit* to the asynchronous data receiver. The transmitter *does not* transmit (neither separately nor by encoding using modulation) along with serial stream of bits any clock rate information in asynchronous communication. The receiver clock is thus not able to maintain identical frequency and constant phase difference with the transmitter clock.

When a device sends data using a serial communication frame, it may not be as simple as shown in Figures 3.1(a) and (b) or as given in Table 3.2. It can be complex and has to be as per the protocol, which is followed by transmitting and receiving devices during communication between them.

Example 3.1

An IBM personal computer has two COM ports (communication ports), COM1 and COM2. These have 8 bytes at IO addresses 0x3F8 and 0x2F8.

Figure 3.1(b) showed COM port handshaking signals besides TxD and RxD. When a modem connects, it detects a carrier signal on the telephone line. A modem sends *data carrier detect* DCD signal at time t_0. A modem then communicates *data set ready* (DSR) signal at time t_1 when it receives the bytes on the line. The receiving end responds at time t_2 by data terminal ready (DTR) signal. After DTR, *request to send* (RTS) signal is sent at time t_3 and the receiving end responds by *clear to send* (CTS) signal at time t_4. After the response CTS, the data bits are transmitted by modem from t_5 to the receiver terminal at successive intervals [Figure 3.1(c)]. Between two sets of bytes sent in asynchronous mode, the handshaking signals RTS and CTS can again be exchanged. This explains why the bytes do not remain synchronized during asynchronous transmission.

A communication system may use the following protocols for synchronous or asynchronous transmission from a device port: RS232C, UART, HDLC, X.25, Frame Relay, ATM, DSL and ADSL. These are protocols for networking the physical devices in telecommunication and computer networks. Ethernet and token ring are protocols used in LAN networks. There are a number for protocols for serial communication. RS232C, UART and HDLC are described in Sections 3.2.2 to 3.2.4.

The protocols in embedded network devices such as bridges, routers, embedded Internet appliances use bridging, routing, application and web protocols. Internet enabled embedded systems use *application* protocols — HTTP (hyper text transfer protocol), HTTPS (hyper text transfer protocol Secure Socket Layer), SMTP (Simple Mail Transfer Protocol), POP3 (Post office Protocol version 3), ESMTP (Extended SMTP), TELNET (Tele network), FTP (file transfer protocol), DNS (domain network server), IMAP 4 (Internet Message Exchange Application Protocol) and Bootp (Bootstrap protocol) and others (Section 3.11).

Embedded wireless appliances use wireless protocols— IrDA, Bluetooth, 802.11 and others (Section 3.13).

Synchronous, iso-synchronous and asynchronous are three ways of communication. Clock information is transmitted explicitly or implicitly in synchronous communication. The receiver clock continuously maintains constant phase difference with the transmitter clock. HDLC is a data link protocol for computer networks and telecommunication devices. RS232C and UART are asynchronous mode communication standard.

3.2.2 RS232C/RS485 Communication

(i) RS232C RS232C communication is between DTE (computer) COM (communication) port and DCE (modem) port. DTE stands for 'Data Terminal Equipment'. DCE stands for 'Data Communication Equipment'. RS232C is an interfacing signal standard between DCE and DTE.

Figure 3.1(b) showed the interfacing (handshaking) signals on a RS232C port. The receive data and transmit data signals from RS232C port are RxD and TxD, respectively. RS232C port serial RxD and TxD bits are asynchronous and follow UART protocol (Section 3.2.3). Receiver end voltage level from – 3 to – 25 V denotes logic 1 and voltage level from +3 to + 25 V denotes logic 0. Transmitter end voltage level from – 5 to – 15 V denotes logic 1 and voltage level from + 5 to + 15 V denotes logic 0.

Example 3.2

The IBM personal computer two COM ports (communication ports) called COM2 and COM1, have IO addresses 0x2F8-0x2FF and 0x3F8-0x3FF, respectively. The COM port is RS232C port. It has TxD and RxD serial output and input. It has handshaking signals $\overline{\text{DCD}}$, $\overline{\text{DSR}}$, $\overline{\text{DTR}}$, $\overline{\text{RTS}}$ and $\overline{\text{CTS}}$.

RS232C port in a computer is used upto 9600 baud/s asynchronous serial transmission rate with UART mode communication. Generally baud rates are set at 300, 600, 1200, 4800 and 9600. When transmitting upto 0.25 m or 1 m on cable (untwisted) the maximum baud rate can be 115.2 k or 38.4 k baud/s, respectively.

RS232C port is used for keyboard serial communication at 1200 baud/s asynchronous serial transmission rate with UART mode communication at IBM PC COM port. The signals used are $\overline{\text{RTS}}$, $\overline{\text{CTS}}$, TxD and RxD for keypad communication. A mice port also is RS232C COM port in the computer. (New mice nowadays use USB in place of COM port).

Example 3.3

A mobile smart phone has a Bluetooth device for personal area wireless network. A Bluetooth device is capable of emulating DCE serial port, which can now communicate in UART mode.
A computer on the other hand has a serial port called COM port (Example 3.1). The mobile device is placed on a cradle. The mobile device port data-pins connect the cradle pins. The cradle connects to the computer or laptop COM port. The mobile and computer serial ports then communicate. The data (for example, pictures or address book data) between them synchronize between COM and emulated serial at Bluetooth device.

RS232C standard provides for UART serial port asynchronous mode communication. A different set of voltage levels are prescribed for the 0s and 1s in RS232C standard.

(ii) RS485 RS485, now called EIA-485 is a protocol for physical layer in case of two wire full or half duplex serial connection between multiple points. Transmission is at 35 Mbps upto 10 meter and 100 Kbps up to 1.2 km. Electrial signals are between + 12 V and –7 V. Logic 1 is +ve and 0 is reverse polarity. Difference in potential defines logic 1 and 0. A converter is used to convert RS232C bits to RS485 and another for vice versa.

3.2.3 UART

Figure 3.1(b) showed handshaking signals of RS232C port and UART serial bits in the output to a serial line device. The UART mode is as follows:

1. A line is in non-return to zero (NRZ) state. It means that in the idle state the logic state is 1 at serial line.
2. The start of serial bits is signaled by $1 \rightarrow 0$ transition (negative edge) on the line for a period equal to reciprocal of baud rate. The baud rate is preset at both receiver and transmitter. The receiver detects the start bit at middle of the interval, logic 0 state of the transmitter start bit.
3. UART bits, when sending a byte, consist of start bit, 8 data bits (for example, for an ASCII character or for a command word), option programmable bit (P-bit) and stop bit, each during the interval δT. When sending or receiving a byte, the logic states during interval 10 δT or 11 δT are as shown in Figure 3.1(c) as a function of time t. A bit period, δT is equal to the reciprocal of baud rate, the rate at which the bits from UART transmitter are sent. One extra bit before the stop bit is programmable bit P and is called TB8 at the transmitter and RB8 at the receiver.
4. The data bits in certain specific cases can be 5 or 6 or 7 instead of 8.
5. The stop bit can be for a minimum interval of 1.5 δT or 2 δT instead of δT in certain specific cases.
6. Optional programmable bit (P-bit) can be used for parity detection or can be used to specify the purpose of the serial data bits that are before the P-bit. For example, P can specify bits as the bits of a control or command word when P = 1 and data bits when P = 0. Bit P can specify the address of receiver when P = 1 and data when P = 0 so that only the addressed receiver wakes up and receives the data in the subsequent data transfers. When P is used as address/data specification, it provides a means to interface a number of UART devices through a common set of TxD and RxD lines and form a UART bus.

UART 16550 includes a 16-byte FIFO buffer and is nowadays used more commonly as compared to the original IBM PC COM port, which had an 8-bit register at UART port and was based on 8250 and did not include the FIFO buffer.

UART serial port communication is usually either in 10 bits or in 11 bits format: one start bit, 8 data bits, one optional bit and one stop bit. UART communication can be full duplex, which is simultaneously both ways, or half duplex, which is one way. It is an important communication mode.

3.2.4 HDLC Protocol

When data are communicated using the physical devices on a network, synchronous serial communication may be used. **HDLC** (High Level Data Link Control) is an International Standard protocol for a data link network. It is used for linking data from point to point and between multiple points. It is used in telecommunication and computer networks. It is a bit-oriented protocol. The total number of bits is not necessarily an integer multiple of a byte or a 32-bit integer. Communication is full duplex.

Table 3.3 gives the synchronous network device port bits in an HDLC protocol. The reader may refer to a standard textbook, for example, "*Data Communications, Computer Networks and Open Systems*" by Fred Halsall from Pearson Education (1996) for details of HDLC and its field bits.

Table 3.3 Format of bits in synchronous HDLC protocol-based network device

S.No.	*Bits[1] at Port*	*Present Compulsorily or Optionally*	*Explanation*
1	Frame start and end signaling flag bits	Compulsory	Flag bits at start as well as at end are (01111110)
2	Address bits for destination	Compulsory	8-bits in standard format and 16-bits in extended format
3a	Control bits Case 1: information frame	Compulsory as per case 1 or 2 or 3	First bit 0, next 3 bits N(S), next bit P/F[2] and last 3 bits N(R) in standard format N(R)[3] and N(S) = 7 bits each in extended format
3b	Control bits case 2: supervisory frame	—	First two bits (10), next 2 bits RR[3] or RNR or REJ or SREJ, next bit P/F and last 3 bits N(R) in standard format. N(R)[4] and N(S)[4] = 7 bits each in extendd format
3c	Control bits Case 3: unnumbered frame	—	First two bits (11), next 2 bits M[5], next bit P/F and last 3-bit remaining bits for M. [8 bits are immaterial after M bits in extended format]
4	Data bits	Compulsory	m frame bits transmit such that each bit is at the line for time ΔT or, each frame is at the line for time m.ΔT and also there is bit stuffing.[6]
5	FCS (Frame check sequence) bits	Compulsory	16-bits in standard format and 32 in extended format
6	Frame end flag bits	Compulsory	Flag bits at end are also (01111110)

[1]Bits are given in order of their transmission or reception.

[2]P/F = 1 and P means when a primary station (Command device) is polling the secondary station (receiving device). P/F = 1 and F means when receiving device has no data to transmit. Usually it is done in last frame.

[3]RR, RNR, REJ and SREJ are messages to convey 'Receiver ready,' 'Receiver not ready,' 'Reject,' and 'Selective reject'. REJ or SREJ is a negative acknowledgement (NACK). NACK is sent only when the frame is rejected.[*A child cries only when milk is not given on need, else it remains silent!*] 'Reject' means that the receiver received a frame out-of-sequence; it is rejected and a repeat transmission of all the frames from the point of frame rejection is requested using REJ. 'Selective reject' means that a frame is received out-of-sequence; it is to be rejected and a selective repeat transmission is requested for this frame using SREJ.

[4]N(R) and N(S) means received (earlier) and sending (now) frame sequence numbers. These are modulo 8 or 128 in standard or extended format frame, respectively.

[5]M five bits are for a command (or response) from a transmitter. Examples of a command are *reset*, *disconnect* or *set a defined mode type*; examples of a response are a message from the receiver for a disconnect mode accepted, frame rejected, command rejected, and for an unnumbered acknowledgement.

[6]When five 1s transmit for the data, one 0 is stuffed additionally. This prevents misinterpretation by receiver the data bits as flag bits (01111110).

3.2.5 Serial Data Communication using the SPI, SCI and SI Ports

Microcontrollers have internal devices for SPI or SCI or SI as explained below. Each device has separate registers for control, status, serially received data bits and transmitting serial bits. Each device is programmable as described below. The device can be used in programmed IO modes or in interrupt driven reception and transmission.

Synchronous Peripheral Interface (SPI) Port Figure 3.3(a) shows an SPI port signals. Figure 3.3(b) shows SPI port in 68HC11 and 68HC12 microcontroller. It has full-duplex feature for synchronous communication. There are signals SCLK for serial clock, MOSI and MISO output from and input to master.

[Section 3.1]. Figure 3.3(b) shows programmable features and DDR feature of Port D. An SPI feature is programmable rates for clock bits, and therefore for the serial out of the data bits down to the interval of 0.5 μs for an 8 MHz crystal at 68HC11.

SPI is also programmable for defining the occurrence of negative and positive edges within an interval of bits at serial data *out* or *in*. It is also programmable in the open-drain or totem pole output from a master to a slave and for device selection as master or slave. This can be done by a signal to hardware input SS (slave select when 0) pin. In the hardware the slave select pin connects to '1' at the *master* SPI device and to '0' at the *slave*. Defining SPI as slave or master can also be done by software. Programming a bit at the device control register does this.

68HC12 provides SPI communication device operations at 4 Mbps. SPI device operates up to 2 Mbps in 68HC11.

Serial Connect Interface (SCI) Port Figure 3.3(c) shows an SCI port programmable features and DDR port bits in 68HC11/12. SCI is a UART asynchronous mode port. Communication is in full-duplex mode for the SCI transmission and receiver. SCI baud rates are fixed as prescaling bits. Rate not programmable separately for individual serial *in* and *out* lines. A baud rate can be selected among 32 possible ones by the three-rate bits and two prescaling bits. The SCI receiver has a *wake up* feature and is programmable by RWU (Receiver wakeup Unavailable) bit. It is enabled if RWU (1st bit of SCC2, Serial Communication Control Register 2) is set, and is disabled if RWU is reset. If RWU if set, then the receiver of a slave is not interrupted by the succeeding bytes. SCI has two control register bits, TB8 and RB8. RWU feature helps in inter-processor communication, and SCI is defined for transmission and for reception using the SCC2 bits. UART communication, when programmed by control bits, is in 11-bit format. A number of processors can communicate on the bus in UART mode by RWU, when RB8 and TB8 bits are set.

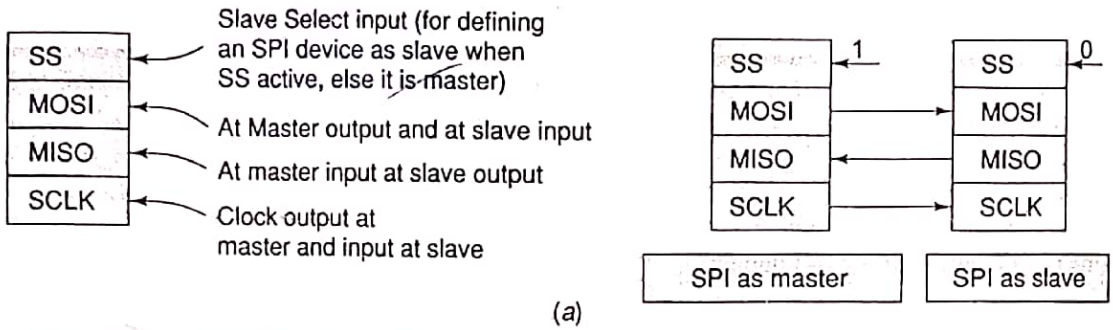
There are separate hardware devices at 68HC11 for synchronous and asynchronous communications. These are SPI and SCI, respectively. 68HC12 provides two SCI communication devices that can operate at two different clock rates. Standard baud rates can be set up to 38.4 kbps. There is only one SCI and standard baud rates in 68HC11 can be set up to 9.6 kbps only.

Serial Interface (SI) Port Figure 3.3(d) shows an SI port. SI is a UART mode asynchronous port interface. It also functions as USRT (universal synchronous receiver and transmitter). SI is therefore a synchronous–asynchronous serial communication port called USART (universal synchronous–asynchronous receiver and transmitter) port. It is an internal serial IO device in 8051. There is an on-chip common hardware device called SI in Intel 80196. Its features are as follows: programmable-rates register after loading the 14 bits at BAUD_RATE register twice. SI operates in one of the following ways:

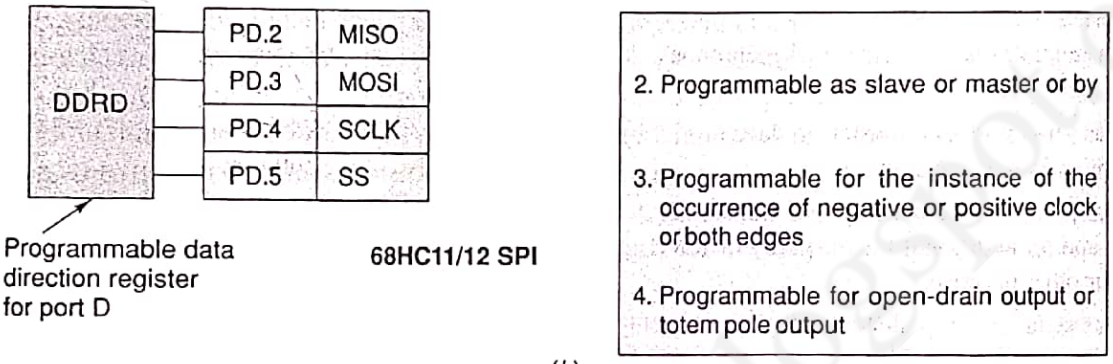
(i) Half-duplex synchronous mode of operation, called mode 0. When a 12 MHz crystal is at 8051 and is attached to the processor, the clock bits are at the intervals of 1 μs.

(ii) Full-duplex asynchronous serial communication, called mode 1 or 2 or 3. Using a timer, the baud rate varies according to the programmed timer bits in modes 1 and 3. Using SMOD bit at SFR called PCON, when mode 2 is used, the baud rate is programmable at two rates only. It is 1/64 or 1/32 of oscillator frequency at 8051. TB8 and RB8, when using 11-bit format, provide the 10th bit for error-detection or for indicating whether the sent data byte is a command or data for the receiving SI device.

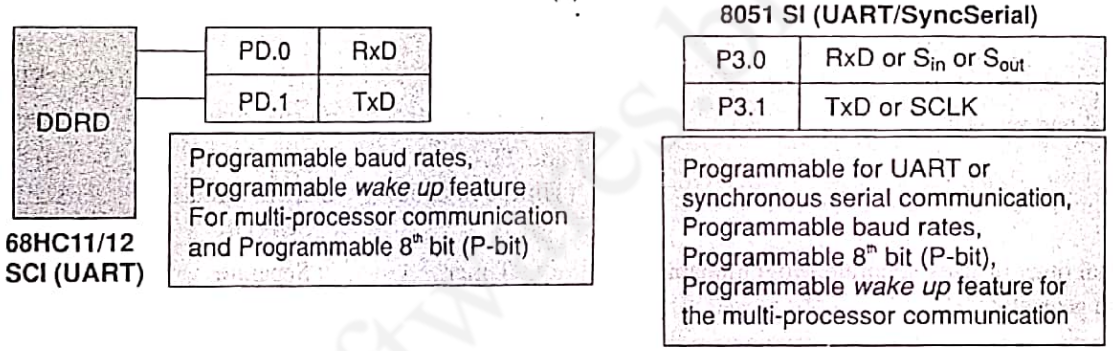
Most microcontrollers have internal serial communication SPI and SCI or SI-like devices for serial communication. The IBM personal computer has two UART chips for the two COM ports. Table 3.4 gives the features of internal serial ports in select microcontrollers.



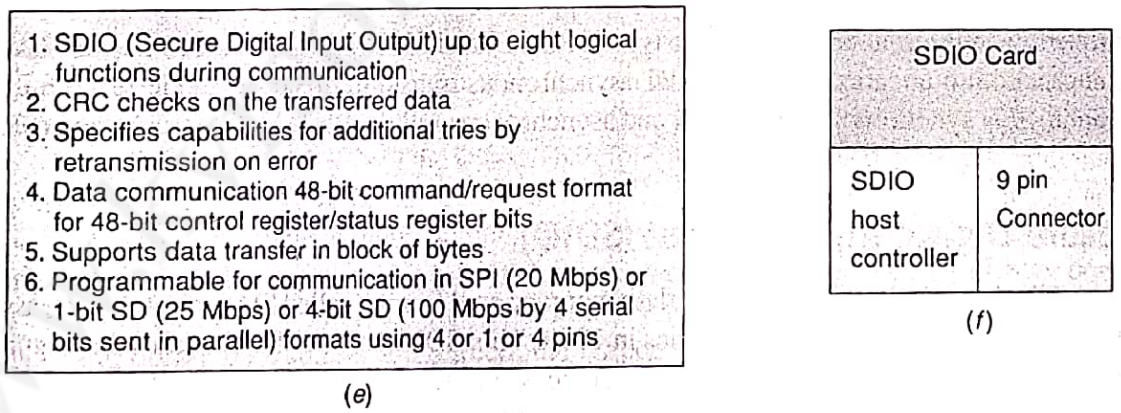
(a)



(b)



(c) (d)



(e) (f)

Fig. 3.3 (*a*) SPI port signals (*b*) SPI port programmable features and DDR at port D in 68HC11/12 (*c*) SCI port in 68HC 11/12 (*d*) SI port in 8051 (*e*) SDIO communication features (*f*) SDIO card structure

Table 3.4 Processor with internal serial ports in microcontrollers

Features	*Intel 8051 and Intel 8751*	*Motorola M68MC11E2*	*Intel 80196*
Synchronous serial port (half or full duplex)	Half	Full	Half
Asynchronous UART port (half or full duplex)	Full	Full	Full
Programmability for 10 as well as 11 bits per byte from UART	Yes	Yes	Yes
Separate un-multiplexed port pins for synchronous and UART serial ports	No	Yes (Separate 4 Pins)	No
Synchronous serial port as a master or slave definition by software or hardware	Software	Hardware and software	Software
UART serial port programmability as a transmitter or receiver and for additional bit for parity or RWU or control or other purpose.	Yes	Yes	Yes
Synchronous serial port registers	SCON, SBUF and TL-TH 0-1	SPCR, SPSR and SPDR	SPCON SPSTAT BAUD_RATE and SBUF
UART serial port registers	SCON, SBUF and TL-TH 0-1 (Time 2 in 8052)	BAUD, SCC1 SCC2, SCSR SCIRDR and SCITDR	SPCON SPSTAT BAUD_RATE and SBUF
Uses internal timer or uses separate programmable BAUD rate generator.	Timer	Separate	Separate as well as the Timer

Microcontrollers have internal devices of three types SPI, SCI and SI. SPI is synchronous master slave mode serial full duplex communication. SCI is UART asynchronous transmitter-receiver mode serial full duplex communication. SI is synchronous half duplex and asynchronous full duplex UART serial communication.

3.2.6 Secure Digital Input Output (SDIO)

Secure Digital (SD) Association created a new flash memory card format, called SD format. It is an association of over 700 companies started from 3 companies in 1999. This SDIO card for SD format IOs [Figure 3.3(e)] has become a popular feature in handheld mobile devices, PDAs, digital cameras and handheld embedded systems. SD card size is just 0.14 × 2.4 × 3.2 cm. SD card {Figure 3.3(f)] is also allowed to stick out of the handheld device open slot, which can be at the top in order to facilitate insertion of the SD card.

SDIO is an SD card with programmable IO functionalities such that it (a) can be used upto eight logical functions, (b) can provide additional memory storage in SD format, and (c) can provide IOs using protocols in systems such as IrDA adapter, UART 16550, Ethernet adapter, GPS, WiFi, Bluetooth, WLAN, digital camera, barcode or RFID code reader.

Figure 3.3(e) shows an SDIO communication device port features. It supports SPI (Section 3.2.5), 4-bit and 1-bit SD formats. Both SPI and SD formats specify that there should be interrupt handling of the IOs and also the CRC checks on transferred data, and specifies capabilities for more tries on error. SDIO card [Figure 3.3(f)] has 9 pins. Six pins are for communication using SPI or SD. A processing element function is used as SDIO host controller to process the IOs. The controller may include SPI controller to support SPI mode for the IOs and supports the needed protocol functionality internally. Maximum clock rate supported for SPI is 20 MHz for a maximum of 20 Mbps data transfers. There is an optional 4-bit SD mode, which uses 4 data lines. Maximum clock rate supported is 25 MHz for maximum 100 Mbps SD bit data transfer in 4-bit SD mode. Four serial bits simultaneously transmits at four times clock rate on 4 SD lines in this mode. Four-bit SD mode is compromise between serial and parallel bits communication to enhance serial transfer rate four times. In 1-bit SD mode, with 25 MHz clock the maximum data transfer rate is 25 Mbps and one serial bit transmits at 1 line only.

SDIO card has a control section called function 0. It necessarily uses function 1 and optionally uses functions between 2 and 7 (depending on the application in devices used such as Bluetooth, PHS, GPS or digital camera). Each function has PCMIA (Personal Computer Manufacturer Interface Adapter) defined card information structure and registers, for example, ID number, function enable bit, supported bus width (1 or 4), voltage, power needs, clock rates and interrupt enable bit. Each function's specifications for the register bits and protocols have been defined in SD standard. A standard device driver can therefore be written. A new function can also be defined.

Data communication is in 48-bit command/request format for 48-bit control register/ status register bits and supports data transfer of blocks of bytes. For single byte transactions, SDIO card may also include a UART 16550 mode communication over the SD bus.

SDIO is an SPI based 9 pin connector card, which supports SPI as well as 1-bit SD or 4-bit SD communication. SDIO supports 8 logic functions. SDIO functions include IOs with several protocols, for example, IrDA adapter, UART 16550, Ethernet adapter, GPS, WiFi, Bluetooth, WLAN, digital camera, barcode or RFID code reader.

3.3 PARALLEL DEVICE PORTS

The parallel port of devices transfers number of bits over the wires in parallel. Parallel wires capacitive effect reduces the length up to which parallel communication can be done. High capacitance results in delay for the bits at the other end undergoing transition from 0 to 1 or from 1 to 0. High capacitance can also result in noise and cross talk (induced signals) between the wires. Therefore, parallel port carries the bits upto short distances, generally within a circuit board or IC.

Figure 3.4(a) shows the parallel input, output, and bi-directional device ports. Figure also shows a device-interfacing circuit with the processor and system buses. Parallel port inputs I0 to I7 may be to a keypad controller. Parallel port outputs O0 to O7 may be output bits to LCD display output controller. BR_i and BR_0 are the input and output data buffers at bi-directional IO port.

A device port connects to the address bus signals, A_i and A_j through a port address decoder. $\overline{IORD}$ and $\overline{IOWR}$ are additional control signals for a port device read and write, respectively, in case of an 80x86 processor, which has IO mapped IOs. The memory read and write signals, $\overline{RD}$ and $\overline{WR}$ are used in the processor with memory mapped IOs [Section 2.2.2].

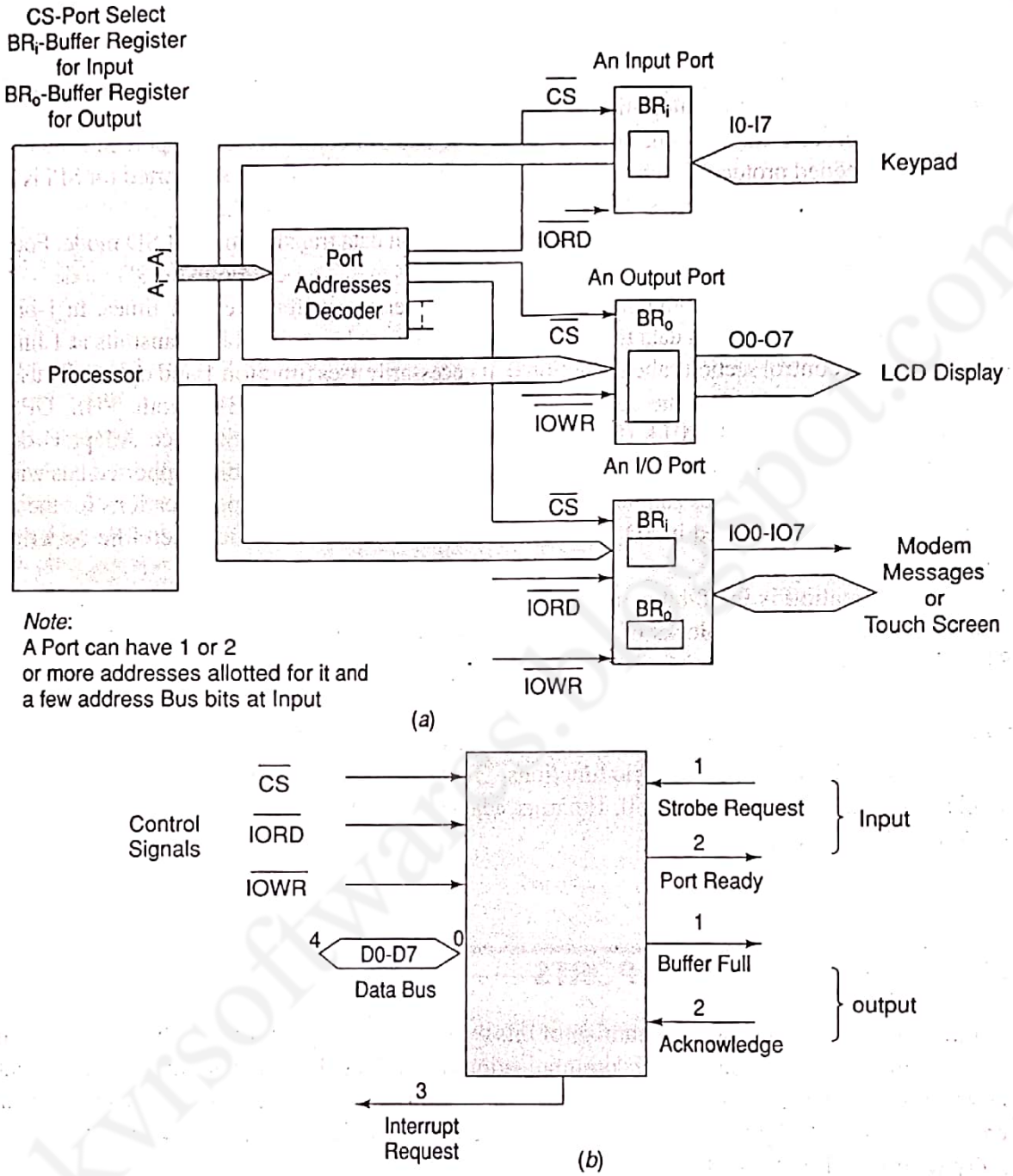


Fig. 3.4 (*a*) Parallel input port, output port, and a bi-directional port for connecting the device (*b*) The handshaking signals when used by the IO ports

Example 3.4

IBM personal computer has a parallel port with a 25 pin connector. There are 8 IO pins, 5 input pins for status signals (four active high S3 to S6, one active low $\overline{S7}$) from external device port (for example, printing device port) and 4 output pins for control signals (one active high C2 and three active low $\overline{C0}$, $\overline{C1}$ and $\overline{C3}$, The 8 pins are ground pins (Pins at 0 V). The status pins and control pins are provided for handshaking between peripheral and computer.

Figure 3.4(b) shows the handshaking signals. An external input device to the device port makes a strobe request, *STROBE,* after it is ready to send the byte and the system IO device sends the acknowledgement, *PORT READY* when BR_i (receiving buffer) is empty.

An external output receiving device sends the message *ACKNOWLEDGE* when the IO device port ends the *BUFFER FULL* signal. The processor is sent the *INTERRUPT REQUEST* message when BR_o transmitting buffer is empty not full (available for next write) or when the receiving buffer is full (available for next read). This enables the processor to interrupt and retransmit next byte(s) in next cycle or receive the byte(s) from input using the appropriate service routines for output or input from the port, respectively.

Example 3.5

Intel 8255 is a programmable peripheral interface (PPI) chip. A PPI device has four addresses, three for the ports and one for the control word. There are three 8-bit ports: port A, B and C. Port C can also be programmed to function in bit set-reset mode. Each bit of this port can be set to 1 or reset to 0 by an appropriate control word. Alternatively, the ports can be grouped as Group A (Port A and Port C upper four bits) or Group B (Port B and Port C lower four bits).

1. In mode 0 programming for a group, each port group does not use handshaking signals.
2. Mode 2 programming is used for port A as input as well as output. In mode 2 programming for the group A, port A uses handshaking signals, STROBE, PORT READY, BUFFER FULL, ACK and INTERRUPT and port A functions as a bi-directional IO port.
3. Mode 1 programming is either for port as input or as output. In mode 1 programming for the group A or B, port A or B uses only one of the two handshaking signal pairs, either (STROBE, PORT READY) or (BUFFER FULL, ACK) plus one INTERRUPT signal.

The following characteristics are taken into consideration when interfacing a device port.

1. A device port may have multi-byte data input buffers and data output buffers. Suppose there is an eight-byte buffer. Assuming that a device (as in the 80196 microcontroller) can generate three interrupts, one on receiving a byte, one on receiving the fourth byte and one when the buffer is full, then the deadline for servicing these interrupts increases up to eight times compared to the case when there is a single byte register instead of buffer.
2. A port may have a DDR (Data Direction Register) (as in the 68HC11 microcontroller). This is an advantage since each bit of the port is now programmable. It can be set as input or output. DDR programs the port bits.
3. Port LSTTL-driving capability and port-loading capability are important characteristics. A port may be an OD (open drain) port. It has zero driving capability unless the drain connects the positive supply voltage. If the given port has OD gates, an appropriate pull-up resistance or transistor is connected to each port pin to provide the driving capability. The drain or collector connects to the supply voltage to provide the pull-up.
4. If a given port is *quasi bi-directional* (as in 80196), then the port pins have limited driving capability, which suffices for a period of one or a few clock cycles and drives a LSTTL gate for that period. When this device port connects to more than one LSTTL, then an appropriate pull-up circuit will be required for the port pins.
5. There may be multiple or alternate functionality in the port pins; for example, 80196 input port pins. Each pin of P2 has an alternative use as multi-channel analog input facility for 8 analog inputs. Another example is 8051 two ports P0 and P2. These port bits also have an alternate function in that they bring out when needed the internal multiplexed buses for the external program and memories whenever the

internal memory is insufficient. Each pin of P3 in 8051 has multiple uses. These are used during serial communication, timer/counter signals, interrupt-signals, and $\overline{RD}$ and $\overline{WR}$ control signals for external memories. 68HC11 ports B and C are of 8 bits each and have alternative uses for the port pins in it. One of the alternate functions is to bring out the internal address and data buses, respectively.

6. A port may have provision for multiplexed output to connect to multiple systems or units.
7. A port may have provision for demultiplexed inputs from multiple systems or units.

A parallel device port can have parallel inputs, parallel outputs, bi-directional and quasi-bi-directional IOs. A parallel device port can have handshaking pins. A parallel device port can also have control pins for control-signal outputs to external circuit and status pins for inputs of status signals to external circuits.

3.3.1 Parallel Port Interfacing with Switches and Keypad

A 16 keys keypad has many applications. A mobile smart phone device has 16 keys and four menu: select up, down, left, right keys. Assume that an IO device has two ports, A and C. The device has a processing element which functions as a keypad-controlling device (controller).

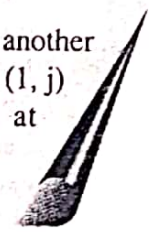
Figure 3.5(a) shows how a set of switches or a keypad of 16 keys and four menu-select keys can interface to the device. Four bits of an 8-bit input port A (A_4-A_7) can be used for the four menu select keys. Assume that the idle state logic state equals 1. The 16 keys can be considered as arranged in four rows and four columns. The other four bits of A (A_0-A_3) are inputs from sense lines from four rows. Assume that the idle state logic state is equal to 1. The four bits of output port C (C_0-C_3) are output to sense lines in four columns.

The processing element in device activates for polling the output port C ten times each second and sends C_0-C_3 = 0000; after a wait it reads D_0-D_7 and A_4-A_7. The processing element computes the code of the pressed key and generates a status signal when a key is found pressed. From the bit pattern found at A_0-A_3, the processing element computes 7-bit ASCII code of the pressed key at that instance and can output that code at D_0-D_6. It also outputs D_7 = 1 when a specific key is found pressed, else D_7 = 0. The processing element also processes the bounces when a key is pressed. This takes care of bouncing effects. The processing element is thus functioning as a keypad controller, as it is keypad specific.

Example 3.6

A mobile phone keypad is smart and is called T9 keypad. Nine keys are used to enter not only the numbers but also text of messages. The processing element is programmed as a state machine to compute the ASCII code to be sent. A state machine generates the states. For example, a key marked as number 5 is in state (0, 5) in reset state, which is also its idle state. The key-state undergoes transition to state (1, 5) when it is pressed first time. When it is pressed second time within 1 s, the key state becomes (1, j). This state corresponds to character j. If it is pressed third time within 1 s, the key-state becomes (1, k). The state of the key changes in a cyclic fashion. (1, 5) → (1, j) → (1, k) → (1, l) → (l, 5) → (1, j), ... The transition of a key state occurs only if it is found pressed within 1 s of the previous transition, and the appropriate action takes place as per the state. The processing element computes the ASCII code from the read value of A_0-A_3 and key state at an instance. After processing is over or after 1s, the key-state resets to (0, 5).

Two key states simultaneously or separately undergoing transitions can define a transition to another state. For example, when there is transition to (1, j) state after another key state is (1, #), then (1, j) undergoes another transition to (1, j), and when that key state is (0, #) it remains at (1, j).



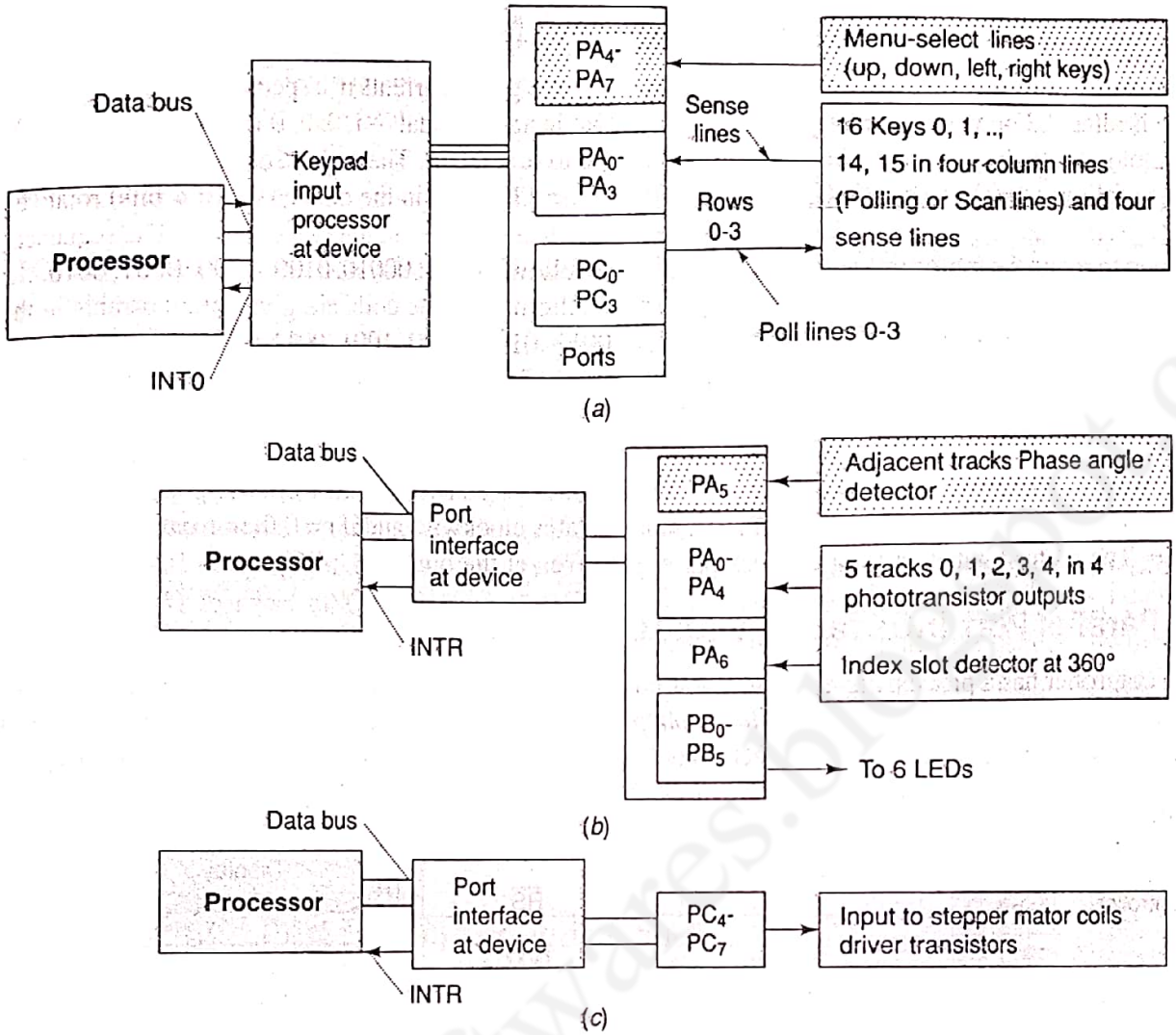


Fig. 3.5 (*a*) Parallel input port A and a four bit output port C used for interfacing a set of 16 keys in keypad and four menu select keys (*b*) Parallel input port A connected to an encoder circuit which senses the rotated or linear position of a moving shaft and port B connected to 6 LEDS (*c*) Four bit *parallel output port* C connected to a stepper motor

A parallel device having a number of input and output bits can be used to find the code of the pressed key in a matrix of keys. A keypad controller has a processing element to compute the code of the pressed key and to generate a status signal when any key is found pressed. A mobile phone keypad controller processes the states of the keys to enable application of same keypad for dialing as well as editing SMS messages.

3.3.2 Parallel Port Interfacing with Encoders

Encoder is a device that measures angular or linear position of a rotating or moving shaft. It has application in robots and industrial plants. A rotatory-angle encoder has multiple tracks on a rotating disk. Each track has half of the segments transparent and half opaque. A linear encoder has a multi-slotted plate. A set of n infrared (IR) LED and phototransistor pairs generate n-bit inputs for a port. The encoder connects to parallel port, as shown in Figure 3.5(b).

3.3.3 Parallel Port Interfacing with Stepper Motor

A stepper-motor rotates by one step angle when its four coils are given currents in a specific sequence and that sequence is altered. For example, assume that currents at an instance equal + i, 0, 0, 0 in four coils X, X', Y, Y'. The motor rotates by one step when the currents change to 0, + i, 0, 0. The sequences at intervals of T are changed as follows: 1000, 0100, 0010, 0001, 1000, 0100, ... [The bits in the nibble (set of 4 bits) rotate by right shift.] Here 1 corresponds to + i. The motor thus rotates n step angles in interval of (n.T). The sequences are changed to rotate the motor in the opposite direction, as follows: 0001, 00010, 0100, 1000, 0001, 0010, ... [The bits in the nibble (set of 4 bits) rotate by left shift.] Alternately, the coils are given the currents in the sequence of 1100, 0110, 0011, 1001, 1100, 0110, ..., or 0011, 0110, 1100, 1001, 0011, 0110, ... The motor rotates (n/2) steps in interval equals to (n.T/2). T is the period of clock pulses that drives the motor by change of coil currents to the next sequence.

The coils connect to parallel port 4 output pins, as shown in Figure 3.5(c). Alternatively, a processing element called stepper-motor deriver can be used. The driver is given two outputs from the port: clock pulses and a rotating direction bit r. For example, if r = 1, motor rotates clockwise and if r = 0 then motor rotates anti-clockwise. The motor rotates as long as clock pulses are given at the output PC_4-PC_7.

3.3.4 Parallel Port Interfacing with LCD Controller

An LCD controller has a processing element that needs three control signals as inputs and 8 input/output bits for parallel set of 8 IO bits. Eight-bit *parallel output port B* pins PB_0-PB_7 connect LCD controller, as shown in Figure 3.6(*a*). LCD controller also connects to one bit PC_0 at an output port for RS (register select) signal.

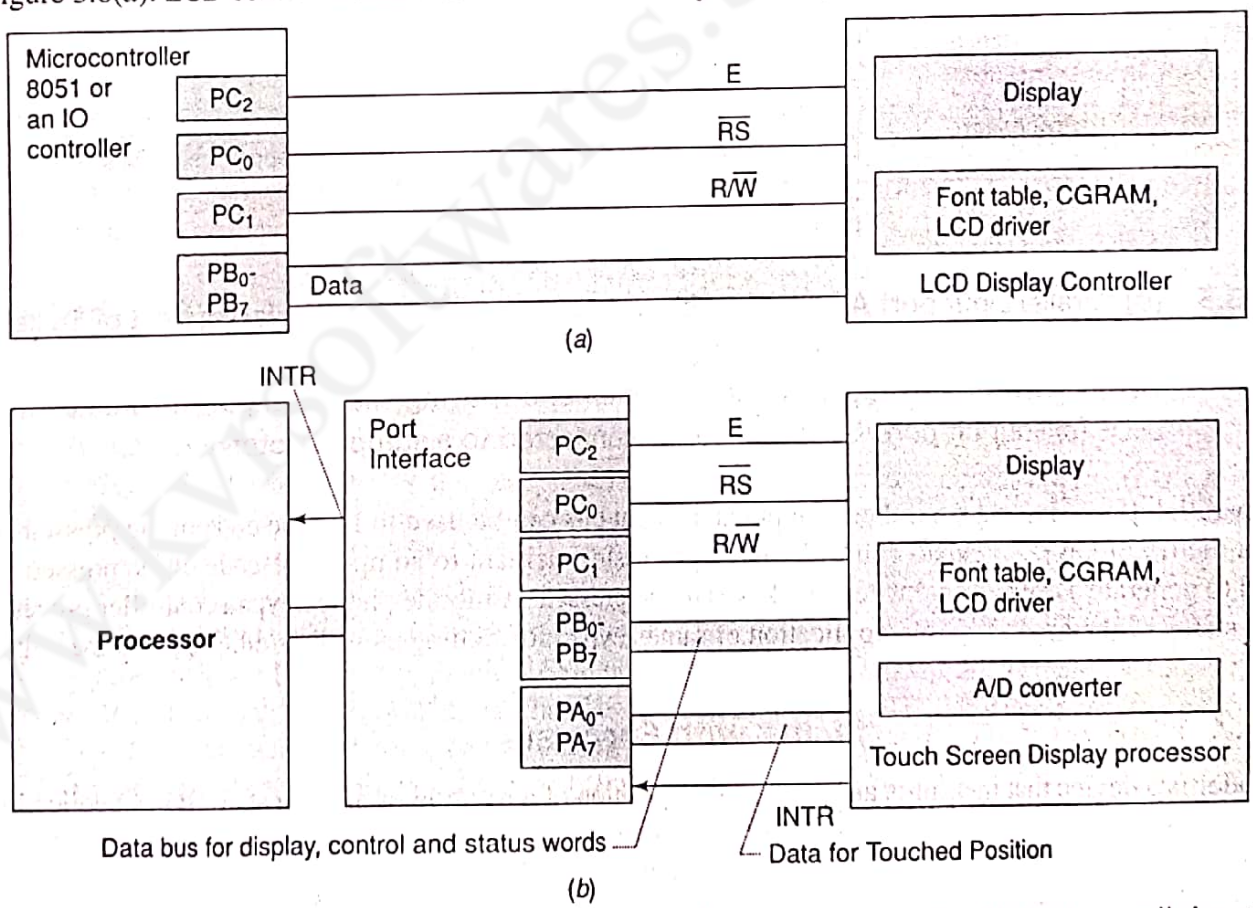


Fig. 3.6 (*a*) Eight bit parallel output port B connected to an LCD controller (*b*) 8-bit parallel output port B and 8-bit parallel input port A connected to a touch screen control circuit

When RS is reset as 0, PB_0-PB_7 communicates a control word to control register of the LCD controller. When RS is set as 1, PB_0-PB_7 communicates data to the LCD controller.

The LCD controller also connects to a one bit PC_1 at output port for R/$\overline{W}$ (read/write). PC_1 is set to 1 when status register of LCD controller is read from PB_0-PB_7. PC_1 is reset to 0 when writing into LCD controller the PB_0-PB_7 bits. The processing element generates all signals required for LCD displays.

The LCD controller is sent control words and data words for initialization and programming PB_0-PB_7 bits, PC_0 and PC_1 outputs for each word to LCD controller. The controller then has to be enabled by sending 1 at E pin. It connects to one bit PC_2 at output port for E (enable). There is an interval in which the controller may be in disabled state. During this interval, it cannot accept instructions or data through the output of control word or data port pins. For example, a control instruction is to clear display. The internal processing element has to clear the bytes at all the N addresses in N characters LCD display. Assume that in a typical LCD, it is 150 µs. When the first 1 is written at E, then 0 is written and a 150 µs delay program is called in-between; the E output creates a negative going pulse at LCD controller. It disables sending of any control word or data for a period of 150 µs.

LCD controller has M displayed character ROM addresses. M = 128 for 128 ASCII codes. For each distinct ASCII character, there is a 64-bit graphic. The LCD controller has an internal CGRAM (character graphic RAM). For each ASCII character, 8 bytes are sent from font table ROM to CGRAM address. CGRAM has N addresses. N = 64 when 64 characters are displayed at the LCD. An address changes by incrementing or decrementing the cursor position to the previous or next address on screen. By sending appropriate control words followed by data, the LCD controller is programmed to display up to 64 characters on the screen.

A parallel device having 8 output data and 3 bits for E, $\overline{RS}$ and R/$\overline{W}$ can be used to connect to an LCD controller.

3.3.5 Parallel Port Interfacing with Touchscreen

Touchscreen is an input device cum LCD display device. It is also interfaced through IO port B functioning as data bus far display, control and status words to an LCD display device controller. The interface uses an additional input port A for a byte, which corresponds to the address of the position touched on the screen.

A touchscreen is either resistive or capacitive. On touching at a position on the screen, there is change in resistance or capacitance, which depends on the touched position. A touch can be finger or stylus. The stylus is about one-fifth thinner than a pencil and about half of the length of the pencil. The resistance or capacitance is a part of a bridge circuit that generates an analog voltage. An 8-bit ADC is given an input from a bridge circuit and the 8-bit ADC output connects to 8-bit input port A.

Eight-bit *parallel IO port B* pins PB_0-PB_7, E, $\overline{RS}$ and R/$\overline{W}$ and eight-bit *parallel input port* A connect to touch screen circuit ports as shown in Figure 3.6(b). An interrupt signal INTR is issued whenever the screen is touched.

Example 3.7

A PocketPC has a touchscreen. The touchscreen device facilitates GUIs. It can display menus as well as a virtual keypad. Using the keypad on screen and stylus, a set of characters can be entered for creating or editing SMS messages, e-mails, or other files. The stylus is held like a pencil and is used to touch the virtual keypad and then the device selects the menu and commands on the screen.

A parallel device having 8 input data bits from an ADC and 8 IO data bus and 4 bits for INTR, E, $\overline{RS}$ and R/$\overline{W}$ can be used to connect a port interface with a processing element to a touchscreen. The ADC generates input bits for the port from the analog signal, which is as per the touched position on the screen.

3.4 SOPHISTICATED INTERFACING FEATURES IN DEVICE PORTS

A device port may not be as simple as the one for a stepper motor port or for a serial line UART. Nowadays, a complex embedded system has highly sophisticated IO devices, for example, SDIO card (Section 3.2.6), IO devices with fast serialization and de-serialization of data, fast transceiver, and real time video processing system. The following are the few sophisticated interfacing device and port features.

1. Let the operation voltage level expected for logic state 1 = 5 V (TTL or CMOS). The Schmitt trigger circuit has a property that when a transition from 0 to 1 occurs, only if the voltage level exceeds 2/3 of the 5 V level is there a transition to 1. Similarly, when a transition from 1 to 0 occurs, only if the voltage level lowers below 1/3 of the 5 V level is there a transition to 0. Hence, the Schmitt trigger circuit eliminates noise as large as 2/3 of 5 V, or 3.3 V, when it is superimposed at an input line to the device. One great advantage of the in-built Schmitt trigger circuit at the port is conditioning of the signal by noise elimination. Otherwise, a device port input will need an external chip for Schmitt trigger-based noise elimination. Such a device is used in transceivers for repeating systems, which are used in long distance communication.
2. When a device port is waiting for instructions, power management can be done at the gates of the device. Lately, a new technology called DataGate (from Xilinx) has been developed for use at ports. DataGate is a programmable ON/OFF switch for power management; DataGate makes it possible to reduce power consumption by reducing unnecessary toggling of inputs when these are not in use. The great advantage of an inbuilt DataGate-like circuit at a device port is reduced power dissipation when the device port is operated at fast speeds. Such a device is extremely useful in systems connected to a common bus and there is a need to control unnecessary input toggling. For example, in a bus interface unit, the input signals should activate only when the input has to be passed to the circuit. As the number of bus interfaces in the system grows, the demand to prevent needless switching of input signals increases.
3. Earlier, port interfaces used to be either open drain CMOSs or TTLs or RS232Cs. (*i*) Nowadays, a system may be required to operate at a voltage lower than 5 V. [Recall Section 1.3.1.] Low Voltage TTL (LVTTL) and Low Voltage CMOS (LVCMOS) gates may be used at the device ports for 1.5 V IO. (*ii*) Nowadays, a system may be required to operate using advanced IO standard interfaces. Examples are High Speed Transreceiver Logic (HSTL) and Stub-series Terminated Logic (SSTL) standards. HSTL is used for high-speed operations; SSTL is used when the buses are to be isolated from relatively large stubs.
4. A device connects to a system bus and also to IO bus when it is networked with other devices. Device and bus-impedances during an IO should match. Else, line reflections occur. Recent developments make it feasible to match these dynamically. For example, a new technology, called XCITE (Xilinx Controlled Impedance Technology) can be used. The great advantage of an inbuilt device for dynamically matched impedances is that when resistors are replaced with digitally, dynamically controlled and matched impedances in the devices, there are no line reflections and therefore no missing bits or bus faults.

5. An IO device may consist of multiple gigabit (622 Mbps to 3.125 Gbps) transceivers (MGTs). Special support circuitry is needed for this rate. Rocker IO ™ serial transreceivers are examples of circuits that provide support circuitry at this rate.
6. A device for an IO may integrate a SerDes (serialization and de-serialization) subunit. SerDes is a standard subunit in a device where the bytes placed at 'transmit holding buffer' serialize on transmission, and once the bits are received these de-serialize and are placed at the 'receiver buffer'. Once the device SerDes subunit is configured, serialization and de-serialization is done automatically without the use of the processor instructions. The great advantage of the SerDes unit is that these operations are fast when compared to operations without SerDes. [A device for IO may integrate a DAA (direct access arrangement using analog IOs along with one master and seven slave CODECs) or McBSP (multi channel buffered serial port with high speed communication) subunit when serializing.
7. Recently, multiple IO standards have been developed for IO devices. A support to the multiple IO standards may be needed in certain embedded systems. A technology, Flexible Select IO ™ -Ultra technology, supports over 20 single-ended and differential IO signaling standards. Advantages of multiple standard device ports are obvious.
8. An IO device may integrate a digital Physical Coding Sublayer (PCS). Analog audio and video signals can then be pulse code modulated (PCM) at the sublayer. The PCS sublayer directly provides codes from analog inputs within the device itself. The codes are then saved in the device data buffers. The advantage of an inbuilt PCS at device port is that there is then no need of external PCM coding. Besides, these operations are performed in the background as well as fast. It improves the system's performance when there are multimedia inputs at the device.
9. A device for IO may integrate an analog unit Physical Media Attachment (PMA) for connecting direct inputs and outputs of voice, music, video and images. The great advantage of inbuilt PMA is that the device directly connects to physical media. PMA is needed for real-time processing of video and audio inputs at the device.

Nowadays, IO devices have sophisticated features. Schmitt trigger inputs are used for noise elimination. Devices with low voltage gates and devices using power management by preventing unnecessary toggling at the inputs are used for sophisticated applications. Dynamically controlled impedance matching is a new technology and it eliminates line reflections when interfacing the devices. The SerDes subunit serializes and deserializes outputs and inputs in the devices. A port may have DAA, McBSP, PCS and PMA subunits for analog IOs for video and audio devices.

3.5 WIRELESS DEVICES

Wireless devices have become very common in recent years for serial transmission of bits.

Wireless devices use infrared (IR) or radio frequencies after suitable modulation of data bits. IrDA (Section 3.13.1), Bluetooth (Section 3.13.2), WiFi, 802.11 WLAN (Section 3.13.3) and ZigBee (Section 3.13.4) have become popular protocols for wireless communication of data bits from a source to the receiver.

An IR source communicates over a line of sight and the receiver phototransistor is used for detecting infrared rays. Example of applications of IR communication includes handheld TV remote controllers and robotic systems. IR devices use IrDA protocol.

Radio frequencies communicate over short and long distances. The transmitter and receiver use antennae to transmit and receive signals and modulator and demodulators to carry the data bits using RF frequencies. Mobile GSM wireless devices use 890–915 MHz, 1710–1785 MHz, or 1850–1910 MHz bands. Mobile CDMA wireless devices use 2 GHz carrier frequencies. Bluetooth and ZigBee wireless devices (Sections 3.13.2 and 3.13.4) use 2.4 GHz or 900 MHz frequencies.

The number of frequency bands is limited, while a large number of devices may need to communicate. Therefore, time and frequency division multiplexing are used. An innovative method is radio frequency hopping over a wider spectrum, as in Bluetooth devices. The transmitted carrier frequencies hop among different channels at a given hopping rate. The transmitter modulates the data bits as per protocol specifications. The receiver tunes to these hopped carrier frequencies at a given hopping rate and in the same hopping sequence as the ones used by the transmitter. The receiver demodulates and detects the data bits as per physical-layer protocol used for transmitting.

Several wireless devices network use FHSS or DSSS transmitters and receivers. Popular protocols are IrDA, Bluetooth, 802.11 and ZigBee.

3.6 TIMER AND COUNTING DEVICES

Most embedded systems need a timing device.

3.6.1 Timing Device

A timer device is a device that counts the regular interval (δT) clock pulses at its input. The counts are stored and incremented on each pulse. It has output bits (in a count register or at the output pins) for the period of counts. The counts multiplied by interval δT gives the time. The (counts–initial counts) $\times \delta T$ interval gives the time interval between two instances when the present count bits are read and the initial counts are read. It has an input pin (or a control bit in a control register) for resetting to make all count bits = 0. It has an output pin (or a status bit in status register) for output when all count bts equal0 after reaching the maximum value, which also means timeout on the overflow.

3.6.2 Counting Device

A counting device is a device that counts the input for events that may occur at irregular or regular intervals. The counts gives the number of input events or pulses since it was last read.

Blind Counting Synchronization A counting device may be a free running (blind counting) device with a prescaler for the clock input pulses and for comparing the counts with the ones preloaded in a compare register. The prescalar can be programmed as p = 1, 2, 4, 8, 16, 32, ..., by programming a prescaler register. It divides the input pulses as per the programmed value of p. It has an output pin (or a status bit in the status register) for output when all count bits equal 0 after reaching the maximum value, which also means after timeout or on overflow. The counter overflows after $p \times 2^n \times \delta T$ interval. It can have an input pin (or a control bit in control register) for enabling an output when all count bits equal count preloaded in the compare register. At that instance, a status bit or output pin also sets in and an interrupt can occur for event of comparison equality. This device is useful for the alarm or processor interrupts at preset instances or after preset intervals with respect to another event from another source.

The counting device may be the free running (blind counting) device with a prescalar for the clock input pulses, for comparing the counts with the ones preloaded in a compare register as well as for capturing counts on an input event. This device functions are similar to the above, but there is an addition input pin for sensing an event and for saving the counts at the instance of that event. At this instance, a status bit can also set in and a processor interrupt can occur for the capture event.

The above device is useful for alarm generation and processor interrupts at the preset times as well as for noting the instances of occurrences of the events and processor interrupts for requesting the processor to use the captured counts on the events. Alarm generation can be synchronized with the input capture events. Writing counts into the compare register does this. Counts in the register are set equal to capture register counts plus additional counts, which define the interval after which an alarm is to be generated.

A blind counting free running counter with prescaling, compare and capture registers has a number of applications. It is useful for action or initiating a chain of actions, and processor interrupts at the preset instances as well as for noting the instances of occurrences of the events and processor interrupts for requesting the processor to use the captured counts on the events for future actions.

3.6.3 Timer cum Counting Device

A timer cum counting device is a counting device that has two functions. (1) It counts the input due to the events at irregular instances and (2) It counts the clock input pulses at regular intervals. An input or a status bit in the timing device register controls the mode as timer or counter. The counts gives the number of input events or pulses since it was last read. It has an output pin (or a status bit in status register) for output when all count bits equal 0 after reaching the maximum value, which also means timeout or overflow interrupts to the processor.

Table 3.5 lists twelve uses of a timer device. It also explains the meaning of each use.

Table 3.5 Uses of Timer Device

S.No.	*Applications and Explanation*
1.	Real Time Clock Ticks (functioning as system heart beats). [Real time clock is a clock that once the system starts it, does not stop and can't be reset. Its *count value* can't be reloaded. *Real time endlessly flows and never returns*!] Real Time Clock is set for ticks using prescaling bits and rate-set bits in appropriate control registers. Section 3.8 gives the details.
2.	Initiating an event after a preset delay time. Delay is as per *count-value* loaded.
3.	Initiating an event (or a pair of events or a chain of events) after a comparison between the preset time with counted value. Preset time is loaded in a Compare Register. [It is similar to presetting an alarm.]
4.	Capturing the *count-value* at the timer on an event. The information of *time* (instance of the event) is thus stored at the *capture register.*
5.	Finding the time interval between two events. *Counts* are captured at each event in the capture register and read. The intervals are thus found out. A service routine does the counts read on interrupt.
6.	Wait for a message from a queue or mailbox or semaphore for a preset time when using an RTOS. There is a predefined waiting period before RTOS lets a task run without waiting for the message. (Section 7.4)

(Contd)

S.No.	Applications and Explanation
7.	Watchdog timer. It resets the system after a defined time. Section 3.7 gives details.
8.	Baud or Bit Rate Control for serial communication on a line or network. Timer timeout interrupts define the time of each baud.
9.	Input pulse counting when using a timer, which is ticked by giving non-periodic inputs instead of the clock inputs. The timer acts as a counter if, in place of clock inputs, the inputs are given to the timer for each instance to be counted.
10.	Scheduling of various tasks. A chain of software-timer interrupts and RTOS uses these interrupts to schedule the tasks.
11.	Time slicing of various tasks. A multitasking or multiprogrammed operating system presents the illusion that multiple tasks or programs are running simultaneously by switching between programs very rapidly, for example, after every 16.6 ms. This process is known as *context switch.* RTOS switches after preset time-slice from one running task to the next. Each task can therefore run in predefined slots of time.
12.	Time division multiplexing (TDM). Timer device is used for multiplexing the input from a number of channels. Each channel input is allotted a distinct and fixed-time slot to get a TDM output. [For example, multiple telephone calls are the inputs and TDM device generates the TDM output for launching it into the optical fibre.]

A timing device has number of states and Table 3.6 gives the states.

Table 3.6 States in a timer

S.No.	States
1.	Reset State (initial count equals 0)
2.	Initial Load State (initial count loaded)
3.	Present State (counting or idle or before start or after overflow or overrun)
4.	Overflow State (count received to make count equal 0 after reaching the maximum count)
5.	Overrun State (several counts received after reaching the overflow state)
6.	Running (Active) or Stop (Blocked) state
7.	Finished (Done) state (stopped after a preset time interval or timeout)
8.	Reset enabled/disabled State (enabled resetting of count equal 0 by an input)
9.	Load enabled/disabled State (reset count equals initial count after the timeout)
10.	Auto Re-Load enabled/disabled State (enabled count equals initial count after the timeout)
11.	Service Routine Execution enable/disable State (enabled after timeout or overflow)

At least one hardware timer device is a must in a system. It is used as a system clock. Let number of system clock ticks needed before a system interrupt occurs equals *numTicks*. The hardware timer gets the input from a clock-out signal from the processor and activates the system clock tick as per the *numTicks* preset at the hardware timer. On each system clock tick, the user-mode task interrupts and the system takes control. The system enables the privileged mode actions and the CPU context switches as per the preset state of the system. The system control actions are performed by operating system (software).

Figure 3.7 shows hardware timer control bits (and signals) and status flags. *Control bits* are as per the hardware signals and corresponding bits at control register. Control bits (or signals) can be of nine types. These are: (i) Timer enable (to activate a timer). (ii) Timer start (to start counting at each clock input). (iii) Timer stop (to stop counting) from the next clock input. (iv) Prescaling bits (to divide the clock-out frequency signal from the processor). (v) Up count Enable (to enable counting up by incrementing the count value on each clock input) (vi) Down count Enable (to decrement on a clock input). (vii) Load enable (to enable loading of a value at a register into the timer). (viii) Timer-interrupt enable (to enable interrupt servicing when the timer outs (overflows) and reaches *count value* equals 0) (ix) Time out enable [to enable a signal when the timer overflows (reaches count equals 0)] to another device.

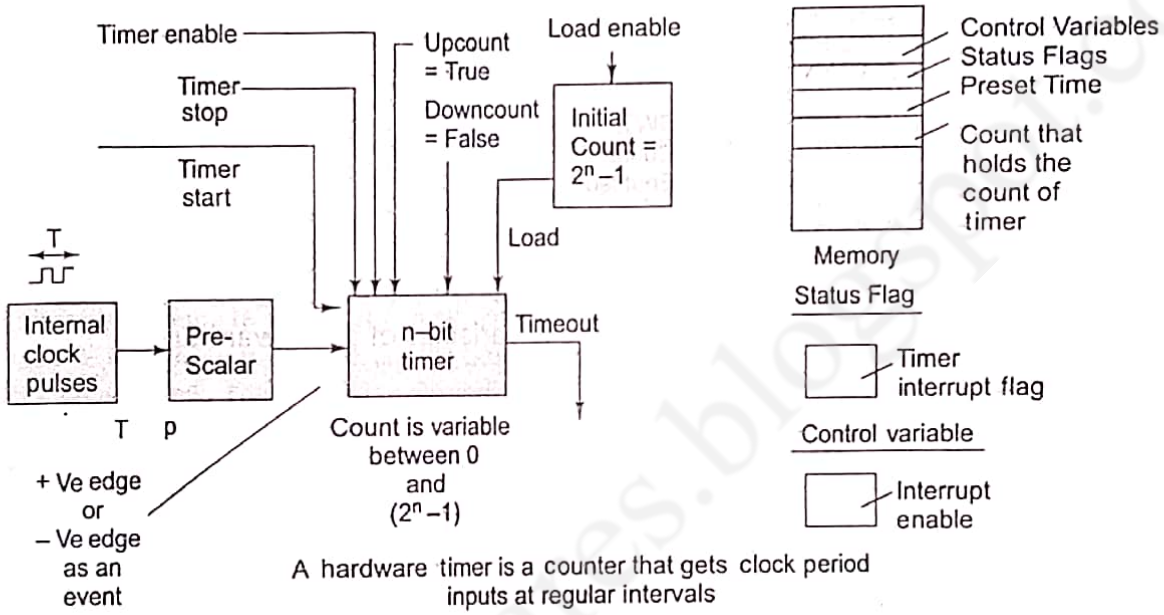


Fig. 3.7 Signals, clock-inputs, control bits and status flags at registers or memory in a hardware timer device

Status flag is as per the corresponding hardware signal time-out from the hardware timer. This flag and signal set when the timer all bits (*count value*) reach to 0.

Table 3.7 lists ten forms of the timers for the uses listed in Table 3.5. Software timer (SWT) is an innovative concept.

The system clock or any other hardware-timing device ticks and generates one interrupt or a chain of interrupts at periodic intervals. This interval is as per the *count-value* set. Now, the interrupt becomes a clock input to an SWT. This input is common to all the SWTs that are in the list of activated SWTs. Any number of SWTs can be made active put in a list of active SWTs. Each SWT will set a status flag on its timeout (*count-value* reaching 0). Figure 3.7 shows the control bits and status bits in an SWT. SWT *control bits* are set as per the *application*. There is no hardware input or output in an SWT. A flag sets when the SWT count-value reaches 0 after reading the maximum. Table 3.8 lists all the variables of SWT. It includes the control-bits and status flags. SWT thus has similar control variables and flags as in the hardware timer or counter.

SWT actions are analogous to that of a hardware timer. While there is physical limit (1, 2 or 3 or 4) for the number of hardware timers in a system, SWTs can be limited by the number of interrupt vectors provided by the user. Processors (microcontrollers) also define the interrupt vector addresses of two or four SWTs.

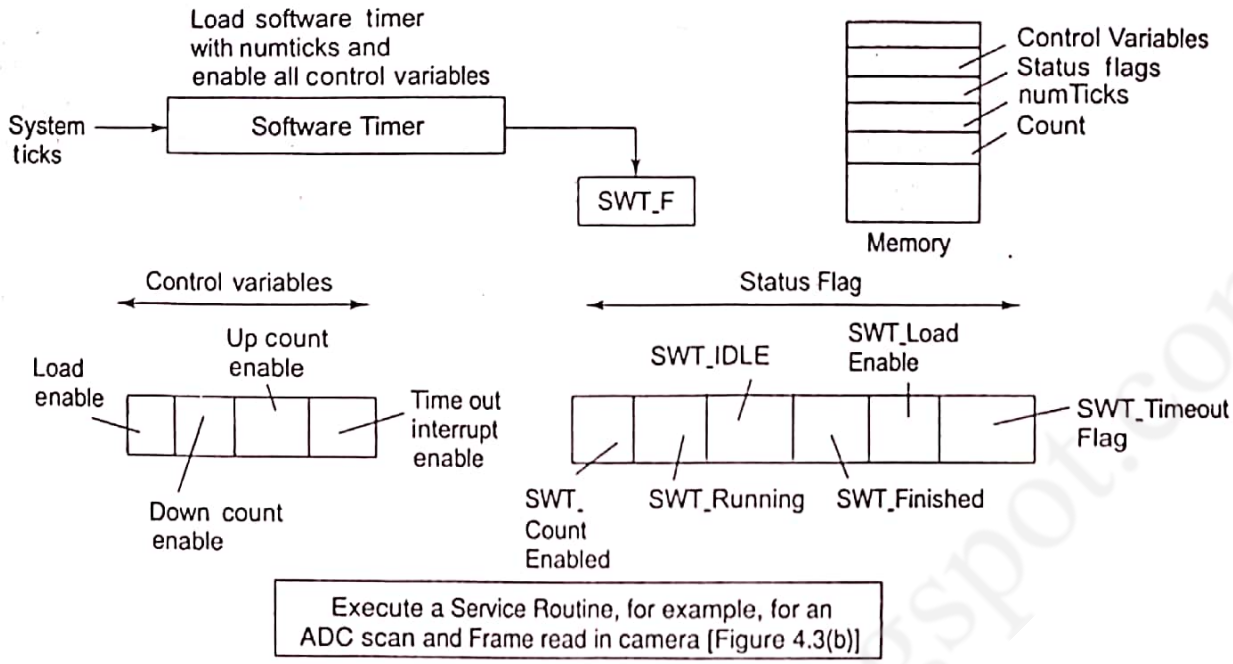


Fig. 3.8 Control bits, status flags and variables of a software timer

Table 3.7 Ten forms of a timer

S.No.	*Types*
1.	Hardware internal timer
2.	Software timer (SWT)
3.	User software-controlled hardware timer.
4.	RTOS-controlled hardware timer. An RTOS can define the clock ticks per second of a hardware timer at a system. [Refer to function OS_Ticks (N) in Section 9.2.1.]
5.	Timer with periodic time-out events (auto-reloading after overflow state). A timer may be programmable for auto-reload after each time-out.
6.	One Shot Timer. (No reload after the overflow and finished state.) It triggers on event-input for activating it to running state from its idle state. It is also used for enforcing time delays between two states or events. On the event or reaching a state one shot timer starts and after the time out another state or event occurs.
7.	Up count action Timer. It is a timer that increments on each count-input from a clock.
8.	Down count action timer. It is a timer that decrements on each count-input.
9.	Timer with its overflow status bit (flag), which auto-resets as soon as interrupt service routine starts running.
10.	Timer with overflow-flag, which does not auto reset.

Timing devices are needed for a number of uses in a system. (i) There can only be a limited number of hardware timers present in the system. A system has at least one hardware timer. The system clock is configured from this. A microcontroller may have 2, 3 or 4 hardware timers. One of the hardware timer ticks forms the inputs from the internal clock of the processor and generates the system clock. Using the systems clock or internal clock, the number of software timers can be drven. These timers are programmable by the device driver programs. (ii) A software timer is software that executes and increases or decreases a count-variable (count value) on an interrupt on a timer output or on a real-time clock interrupt. The software timer also generates interrupt on overflow of count-value or on finishing value. Software timers are used as virtual timing devices. There are a number of control bits and a time-out status flag in each timer device.

Table 3.8 Variables for control bits and status in a software timer

S.No.	32 or 16 or 8 or 1-bit variables
1.	Reset Value 32/16/8
2.	Initial Load Value (numTicks) 32/16/8
3.	Count-value (Preset value) 32/16/8
4.	Maximum Value 32/16/8
5.	Minimum Value 32/16/8
6.	Timer run enable bit
7.	Timer interrupt enable bit
8.	Timer reset enable bit
9.	Timer load enable bit
10.	Timer reload (after finished state) enable bit
11.	Overflow-flag

3.7 WATCHDOG TIMER

Watchdog timer is a timing device that can be set for a preset time interval, and an event must occur during that interval else the device will generate the timeout signal. For example, we anticipate that a set of tasks must finish within 100 ms. The watchdog timer disables and stops in case the tasks finish within 100 ms. The watchdog timer generates interrupts after 100 ms and executes a routine that runs because the tasks failed to finish in the anticipated interval. A software task can also be programmed as a watchdog timer (Section 9.3.3). A microcontroller may also provide for the watchdog timer.

The watchdog timer has a number of applications. One application in a mobile phone is that the display is turned off in case no GUI interaction takes place within a specified time. The interval is usually set at 15, 20, 25, or 30 s in a mobile phone. This saves power.

Another application in a mobile phone is that if a given menu is not selected by a click within a preset time interval, another menu can be presented or a beep can be generated to invite user's attention.

An application in a temperature controller is that if a controller takes no action to switch off the current within the preset time, the current is switched off and a warning signal raised, indicating controller failure. Failure to switch off current may cause a boiler in which water is heated to burst.

Example 3.8

68HC11 microcontroller has a watchdog timer in the hardware. There are two registers, CONFIG (system configuration control register) and COPRST (computer operating properly and processor reset on failure). They are for programming the interrupts of watchdog timer. CONFIG has a bit, NOCOP. It configures when the processor writes the configuration word at address 0x003F. NOCOP is the 2nd bit of CONFIG. If this bit is reset to 0, the COP facility is enabled. [COP means computer (68HC11) operating properly watchdog timer. The COP watchdog timer provides for keeping a watch on execution time of the user program.]

When user program takes a longer time in a routine than planned or expected the user provides for storing at desired intervals; first, the 0x55 and then the 0xAA at the computer-reset control register COPRST. By keeping a watch means that as soon as the watchdog timer overflows (time outs), the program counter is reset according to 16 bits at the lower and higher bytes that are preloaded at addresses 0xFFFA and

0xFFFB, respectively. If these 16 bits are same as the bits in 0xFFFE and 0xFFFF, then the microcontroller executes instructions, which are same as when it resets on power up or else it executes the routine at the 16-bit address fetched from 0xFFFE and 0xFFFF whenever there is failure within the watched time interval.

The 0th and 1st bit of the option register, OPTION, at the address 0x0039 are the CR_1 and CR_0 bits. If NOCOP resets (0) and CR_1-CR_0 = 0–0, the watchdog timer time out occurs after every 2^{16} pulses. As T = 0.5 µs for the processor when the E clock output is 2 MHz, the WDT time-out occurs at every 16.384 ms ($2^{16} \times 0.5$ µs) unless the user software stores at desired intervals before a time out, first the 0x55 and then the 0xAA at the computer reset control register COPRST. This means user program resets the watchlog timer by itself after finishing the watched section of the program. [After 2^{15} pulses if CR_1-CR_0 = 0–1, 2^{14} pulses for 1–0, 2^{13} pulses for 1–1].

A watchdog timer has a number of applications and is a timing device such that it is set for a preset time interval and an event must occur during that interval else the device will generate a timeout signal and interrupt for the failure to get that event in the watched time interval.

3.8 REAL TIME CLOCK

Real time clock (RTC) is a clock that causes occurances of regular interval interrupts on its each tick (timeout). An interrupt service routine executes on each timeout (overflow) of this clock. This timing device once started never resets or is never reloaded with another value. Once it is set, it is not modified later. The RTC is used in a system to save the current time and date. The RTC is also used in a system to initiate return of control to the system (OS) after the preset system clock periods.

Example 3.9

(*i*) Assume that a hardware timer of an RTC for calendar is programmed to interrupt after every 5.15 ms. Assume that at each tick (interrupt) a service routine runs and updates at a memory location. Within one day (86400 s) there will be 2^{24} ticks, the memory location will reach 0x000000 after reaching the maximum value 0xFFFFFF. Within 256 days there will be 2^{32} ticks, the memory location will reach 0x00000000 after reaching the maximum value 0xFFFFFFFF. Note that battery must be used to protect the memory for that long period.

(*ii*) Assume that an RTC has to implement using a software timer. Assume that a hardware 16-bit timer ticks from processor clock after 0.5 µs. It will overflow and execute an overflow interrupt service routine after 2^{15} µs = 32.768 ms. The interrupt service routine can generate a port bit output after every time it runs and can also call a software routine or sends a message for a task. If n = 30, the RTC initiated software will run every 30 × 32.768 ms, which is close to 1 s.

(*iii*) A real time clock timer for interrupts at regular intervals is present in a microcontroller. 68HC11 has a register called the Pulse Accumulator Control Register, PACTL and two lowest significance bits, RT_1-RT_0 (1^{st} and 0^{th}). PACTL is *write* only. If the RT_1-RT_0 pair is 00, an interrupt can occur after 2^{13} pulses of the E clock. If the E clock pulses are of 2 MHz and thus T is 0.5 µs, the interrupts from a real time clock occur after each 4.096 ms. If the RT_1-RT_0 pair is 01, an interrupt can occur after 2^{14} pulses of the E clock, that is, after each 8.192 ms. If the RT_1-RT_0 pair is 10, the interrupt can occur after 2^{15} pulses of the

E clock, that is after each 16.384 ms. If the RT_1-RT_0 pair is 11, an interrupt can occur after each 2^{16} pulses of the E clock, that is, after each 32.768 ms. The real time clock is based on a free running counter in 68HC11. RT_1-RT_0 bits control its rate of ticking.

The interrupts from a real time clock are disabled or enabled by I bit in clock control (CC) register. The interrupts from real time clocks are also locally masked by the 6th bit, RTI in timer interrupt mask register2, TMASK2. This bit is set to unmask and reset to mask the real time clock interrupts. If RTI and I bits permit the interrupt request for real time clock timeout then the microcontroller fetches the lower and higher bytes of the interrupt servicing routine address from the addresses 0xFFF0 for higher byte) and 0xFFF1 (for lower byte). This is the vector address for real time clock interrupts in 68HC11. The interrupt service routine must clear (0) the RTIF, which is interrupt flag for the real time clock interrupts. The RTIF is a bit in timer interrupt flag register2, TFLG2. The TFLAG2 is at address 0x0025. It is set by each interrupt from the real time clock interrupt and therefore it must be cleared in order to enable next interrupt before returning from the corresponding service routine and before the next real-time clock-interrupt occurs.

A real time clock (RTC) provides system clock and it has a number of applications. It is a clock that generates system interrupts at preset intervals. An interrupt service routine executes on each tick (timeout or overflow) of this clock. This timing device once started is generally never reset or never reloaded to another value.

3.9 NETWORKED EMBEDDED SYSTEMS

Each specific IO device may be connected to others using specific interfaces; for example, an IO device connects and is interfaced to an LCD controller, keyboard controller or print controller using specific interface. Bus communication simplifies the number of connections and provides a common protocol for interconnecting different or same type of IO devices.

Any device that is compatible with a system's IO bus can be added to the system (assuming an appropriate device driver program is available), and a device that is compatible with a particular IO bus can be integrated into any system that uses that type of bus. This makes systems that use IO buses very flexible, as opposed to direct interconnections between the processor and each IO device, and it allows system support to many different IO devices (depending on the needs of its users), and it also allows users to change the IO devices that are attached to system as their needs change.

The main disadvantage of an IO bus (and buses in general) is that each bus has a fixed bandwidth that must be shared by all the devices, which connect to the bus. Even worse, electrical constraints (wire length and transmission line effects) cause buses to have less bandwidth than using the same number of wires to connect just two devices. Essentially, there is a trade-off between interface simplicity and bandwidth sharing. Assume that bandwidth of a bus is 200 Mbps. If the bus communicates two devices simultaneously then it does so by 100 Mbps communication by each.

IO devices communicate with the processor through an IO bus, which is separate from the memory bus that the processor uses to communicate with the memory system. Embedded systems connected internally on the same IC or systems at very short, short and long distances, and can be networked using the followings types of IO buses, each functioning according to specific protocols.

1. Using a serial IO bus allows a computer or controller or embedded system to interface network with a wide range of IO devices without having to implement a specific interface for each IO device. When