Figure 18-12. LCD Connection to Parallel Port

Figure 18-13. Stepper Motor Connection to Parallel Port

ULN2003 Connection for Stepper Motor
Pin 8 = GND
Pin 9 = +5V

(use a separate power supply for motor)
A circuit called a charge pump is used to generate this negative voltage. A charge pump consists of two capacitors and a switch. One of the capacitors is charged to a positive voltage by the main power supply. Then the terminals of this capacitor are switched to the terminals of the second capacitor. The first capacitor discharges rapidly into the second, charging it negatively with respect to system ground. This process is switched rapidly, and a steady negative voltage supply is produced in the second capacitor.

The MAX232 uses two charge pumps. The first is used to double the system voltage of +5 volts to obtain a +10 volt supply that more closely matches the RS-232 standard. The second charge pump inverts this +10 volts to obtain a −10 volt supply. As shown in Figure A.30, the schematic for the Serial Interface, capacitors C10, C11, C12, and C13 are used by the Maxim chip in its charge-pump circuit.

Software

The 68HC11 has several registers that are used to control its built-in UART. In the 68HC11 documentation, this subsystem is referred to as the Serial Communications Interface (SCI).

The SCI control registers are:

- **BAUD** The baud rate control register, at address $1028. Sets the serial communications speed.
- **SCCR1** The SCI control register 1, at address $102C. Determines how many data bits are transmitted per byte.
- **SCCR2** The SCI control register 2, at address $102D. Contains enable bits for the receiver and transmitters, and interrupts generated on receive and transmit.
- **SCDR** The SCI data register, at address $102F. This is the register that contains the actual communications data.
- **SCSR** The SCI status register, at address $102E. Contains various status and error code bits.

In typical use, one would initialize the SCI system at the beginning of the program and then use primarily the SCDR register for transmitting and receiving data and the SCSR register for determining when outgoing data are finished or incoming data are available.

Figure A.31 is an example program that demonstrates how to use these registers to send and receive characters through the 68HC11’s serial port.

Let’s examine the program in detail. Here is where the SCI system is enabled and initialized for 9600–N–8–1 communication:

```assembly
lda #30 ; value for 9600 baud with an 8 MHz xtal
staa BAUD
lda #80c ; turn on xmit and rcv enable bits
staa SCCR2
```
In the program's main loop, it calls a subroutine to receive a character, adds one to the value that's received, and then transmits the new value back:

```c
loop    bsr    rcv_char
       inca
       bsr    xmit_char
       bra    loop
```

The routine to transmit a character waits for any previous characters in progress to finish by testing Bit 6 of the SCSR register. When this bit is a one, the previous transmit operation is complete. The new transmission is initiated simply by writing the desired value to the SCDR register:

```asm
xmit_char:
    ldab    SCSR   ; first check if ready
    andb    #$40   ; bit 6 is transmit complete flag
    beq     xmit_char ; wait until TC flag is 1
    staa    SCDR   ; transmit A register
    rts
```

The routine to receive a character waits for a character to appear by testing Bit 5 of the SCSR register. When this bit is a one, the character is retrieved from the SCDR register. The act of retrieving the character automatically clears the flag in the SCDR:
The transmission of a byte begins with a "start bit," which is sent at the logic zero voltage and lasts for one bit-time.

Then the data for the given byte are transmitted, least significant bit first.

Finally, a stop bit is produced, which is always a logic one.

Sometimes an additional error-checking bit, known as the parity bit, may be included (after the data bits but before the stop bit). The parity bit may be "even parity" or "odd parity." If it's even parity, then all of the data bits plus the parity bit (taken one at a time) should add to an even number.

In the example transmission in the diagram, there is a start bit, followed by eight data bits, followed by a stop bit. This format is known as "N–8–1": no parity, eight data bits, and a stop bit that's one bit long in duration. Sometimes the stop bit may be 1.5 or 2 bits long.

So, for example, "E–7–1" would mean even parity, seven data bits, and one bit-time stop bit. The N–8–1 format is by far the most common presently in use, although occasionally one finds devices that communicate differently.

**Hardware**

The 68HC11 chip includes a hardware module to generate waveforms compatible with the RS-232 systems. (This module is sometimes referred to as a UART for "universal asynchronous receiver/transmitter.") Because the 68HC11 operates at TTL, or 0 to 5v levels, it requires external circuitry to convert its own signals to the RS-232 voltages.

The Handy Board uses the Maxim MAX232 serial transceiver to solve the voltage problem. This chip is located on the Handy Board's separate Serial Interface and Battery Charger board. Figure A.29 illustrates how the Serial Interface Board sits between the desktop computer and the Handy Board performing the voltage conversion.

One of the difficulties in generating RS-232 signals is obtaining the negative voltage required to transmit a logic one. However, it turns out that the specified −15 volts is not required: −10 volts will do for most applications, and some modern computers use −5 volt levels.\(^3\)

---

\(^3\) Desktop computers differ by platform in how tightly they adhere to the official serial line specification. For instance, nearly all Intel-compatible PCs accept zero volts as a valid "logic one" (remember logic one should be a negative voltage). Macintosh computers, on the other hand, require at least −2v for the logic one level. This causes problems if circuit designers reduce the serial spec to simply a 0 to 5v range, which is then accepted by the PC but rejected by the Mac.
When the distance is short, the digital signal can be transferred as it is on a simple wire and requires no modulation. This is how IBM PC keyboards transfer data between the keyboard and the motherboard. However, for long-distance data transfers using communication lines such as a telephone, serial data communication requires a modem to modulate (convert from 0s and 1s to audio tones) and demodulate (converting from audio tones to 0s and 1s).

Serial data communication uses two methods, asynchronous and synchronous. The synchronous method transfers a block of data (characters) at a time while the asynchronous transfers a single byte at a time.

It is possible to write software to use either of these methods, but the programs can be tedious and long. For this reason, there are special IC chips made by many manufacturers for serial data communications. These chips are commonly referred to as UART (universal asynchronous receiver-transmitter) and USART (universal synchronous-asynchronous receiver-transmitter). The COM port in the IBM PC uses the 8250 UART, which is discussed in Section 17.3. The synchronous method and the Intel USART 8251 are discussed in Section 17.4.

**Half- and full-duplex transmission**

In data transmission if the data can be transmitted and received, it is a duplex transmission. This is in contrast to simplex transmissions such as printers, in which the computer only sends data. Duplex transmissions can be half or full duplex, depending on whether or not the data transfer can be simultaneous. If data is transmitted one way at a time, it is referred to as half duplex. If the data can go both ways at the same time, it is full duplex. Of course, full duplex requires two wire conductors for the data lines (in addition to ground), one for transmission and one for reception, in order to transfer and receive data simultaneously. See Figure 17-2.

---

![Figure 17-2. Simplex, Half- and Full-Duplex Transfers](image)

510  
CHAPTER 17: SERIAL COMMUNICATION AND 16450/8250/51
Asynchronous serial communication and data framing

The data coming in at the receiving end of the data line in a serial data transfer is all 0s and 1s; it is difficult to make sense of the data unless the sender and receiver agree on a set of rules, a protocol, on how the data is packed, how many bits constitute a character, and when the data begins and ends.

Start and stop bits

Asynchronous serial data communication is widely used for character-oriented transmissions, and block-oriented data transfers use the synchronous method. In the asynchronous method, each character is put in between start and stop bits. This is called framing. In data framing for asynchronous communications, the data, such as ASCII characters, are packed in between a start bit and a stop bit. The start bit is always one bit but the stop bit can be one or two bits. The start bit is always a 0 (low) and the stop bit(s) is 1 (high). For example, look at Figure 17-3 where the ASCII character "A", binary 0100 0001, is framed in between the start bit and 2 stop bits. Notice that the LSB is sent out first.

![Diagram of a frame with start and stop bits](image)

**Figure 17-3. Framing ASCII "A" (41H)**

In Figure 17-3, when there is no transfer the signal is 1 (high), which is referred to as mark. The 0 (low) is referred to as space. Notice that the transmission begins with a start bit followed by D0, the LSB, then the rest of the bits until the MSB (D7), and finally, the 2 stop bits indicating the end of the character "A".

In asynchronous serial communications, peripheral chips and modems can be programmed for data that is 5, 6, 7, or 8 bits wide. This in addition to the number of stop bits, 1 or 2. While in older systems ASCII characters were 7-bit, due to extended ASCII characters, 8 bits are required for each character. Small non-ASCII keyboards use 5- and 6-bit characters. In some older systems, due to the slowness of the receiving mechanical device, 2 stop bits were used to give the device sufficient time to organize itself before transmission of the next byte. However, in modern PCs the use of 1 stop bit is common. Assuming that we are transferring a text file of ASCII characters using 2 stop bits, we have a total of 11 bits for each character since 8 bits are for the ASCII code, and 1 and 2 bits are for start and stop bits, respectively. Therefore, for each 8-bit character there are an extra 3 bits, or more than 30% overhead.

**Example 17-1**

Calculate the total number of bits used in transferring 5 pages, each with 80x25 characters. Assume 8 bits per character and 1 stop bit.

**Solution:**

For each character, a total of 10 bits is used, 8 bits for each character, 1 stop bit, and 1 start bit. Therefore, the total number of bits is $80 \times 25 \times 10 = 20,000$ bits per page. For 5 pages, 100,000 bits will be transferred.
In some systems in order to maintain data integrity, the parity bit of the character byte is included in the data frame. This means that for each character (7- or 8-bit, depending on the system) we have a single parity bit in addition to start and stop bits. The parity bit is odd or even. In the case of an odd-parity bit the number of data bits, including the parity bit, has an odd number of 1s. Similarly, in an even-parity bit the total number of bits, including the parity bit, is even. For example, the ASCII character "A", binary 0100 0001, has 0 for the even-parity bit. UART chips allow programming of the parity bit for odd-, even-, and no-parity options, as we will see in the next section. If a system requires the parity, the parity bit is transmitted after the MSB, and is followed by the stop bit.

**Data transfer rate**

The rate of data transfer in serial data communication is stated in **bps** (bits per second). Another widely used terminology for bps is **baud rate**. However, the baud and bps rates are not necessarily equal. This is due to the fact that baud rate is the modern terminology and is defined as number of signal changes per second. In modems, there are occasions when a single change of signal transfers several bits of data. As far as the conductor wire is concerned, the baud rate and bps are the same, and for this reason in this book we use the terms **bps** and **baud** interchangeably.

The data transfer rate of a given computer system depends on communication ports incorporated into that system. For example, the early IBM PC/XT could transfer data at the rate of 100 to 9600 bps. However in recent years, PCs, PS, and 80x86 compatibles transfer data at rates as high as 19,200 bps. It must be noted that in asynchronous serial data communication, the baud rate is generally limited to 100,000 bps.

**Example 17-2**

Calculate the time it takes to transfer the entire 5 pages of data in Example 17-1 using:
(a) 2400 bps  
(b) 9600 bps

**Solution:**

(a) 100,000/2400 = 41.67 seconds  
(b) 100,000/9600 = 10.4 seconds

---

![Diagram](image)

**Figure 17-4. UART-to-RS232 Connections using MC1488 and MC1489 Chips**