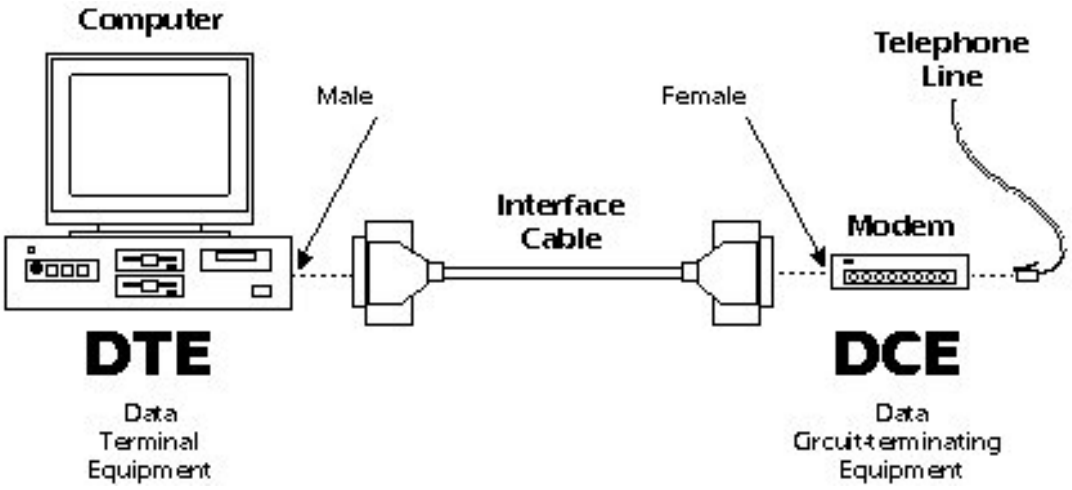
TCSS 372 Laboratory Project 2
RS 232 Serial I/O Interface

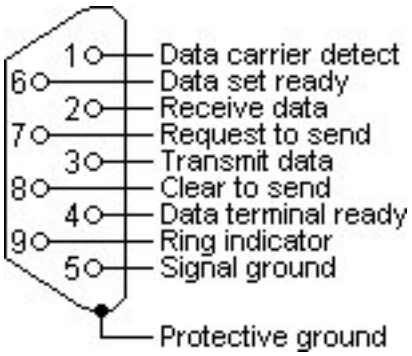
BACKGROUND

In the early 1960s, a standards committee, known as the Electronic Industries Association (EIA), developed a common serial bus interface standard for asynchronous data communications equipment, named RS232. It is still used today for connecting peripheral equipment, like printers, to computers, as well as connecting computers together. It is now being replaced by a faster asynchronous serial bus, USB or Universal Serial Bus. Originally, data communications was thought to mean digital data exchange between a centrally located mainframe computer and a remote computer terminal, or possibly between two terminals without a computer involved. These devices were linked by telephone voice lines, and consequently required a modem at each end for signal translation. While simple in concept, the many opportunities for data error that occur when transmitting data through an analog channel require a relatively complex design. It was thought that a standard was needed first to ensure reliable communication, and second to enable the interconnection of equipment produced by different manufacturers, thereby fostering the benefits of mass production and competition. It specified signal voltages, signal timing, signal function, a protocol for information exchange, and mechanical connectors. You will see both 25 pin and 9 pin connectors used for RS232 today.

If the full EIA232 standard is implemented as defined, the equipment at the far end (left end below) of the connection is named the DTE device (Data Terminal Equipment, usually a computer or terminal), and has a male connector. Equipment at the near end (right end below) of the connection (the telephone line interface) is named the DCE device (Data Circuit-terminating Equipment or Data Communications Equipment, usually a modem), has a female connector, and utilizes the same available pins for signals and ground. The cable linking DTE and DCE devices is a parallel straight-through cable with no cross-overs or self-connects in the connector hoods. If all devices exactly followed this standard, all cables would be identical, and there would be no chance that an incorrectly wired cable could be used. This drawing shows the orientation and connector types for DTE and DCE devices:

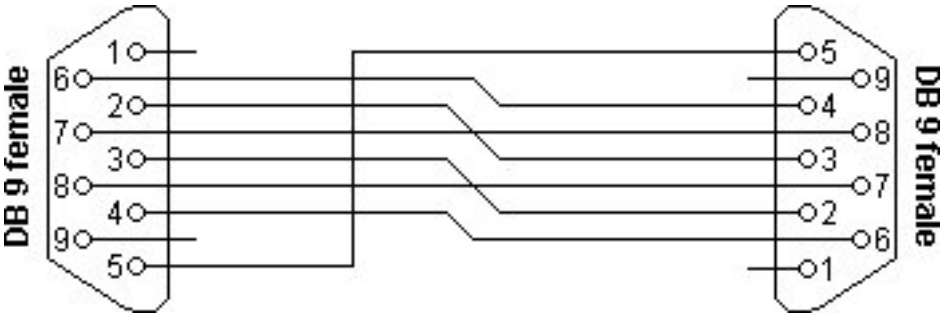


The signals on an RS232 9 pin connector are:



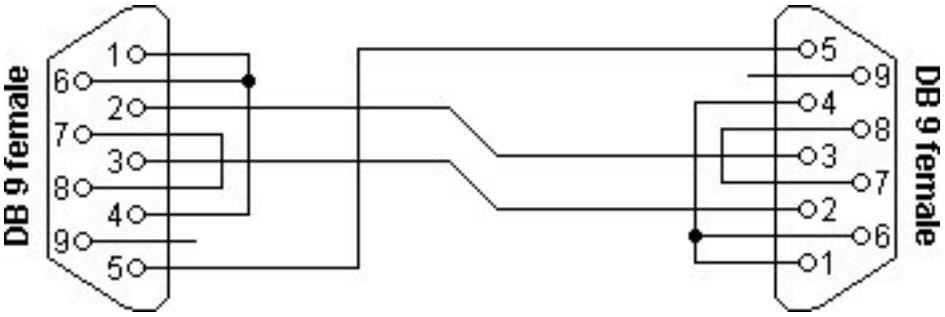
If one wants to connect two computers together, a "null modem" connection can be used in place of the pair of modems. Since both computers are DTE's, the straight through connection from a DTE to a DCE will not work. A null modem crosses the send and receive lines and connects the other wires, the flow control or "handshaking signals", correctly for a DTE to communicate with another DTE.

The configuration for a null modem is:



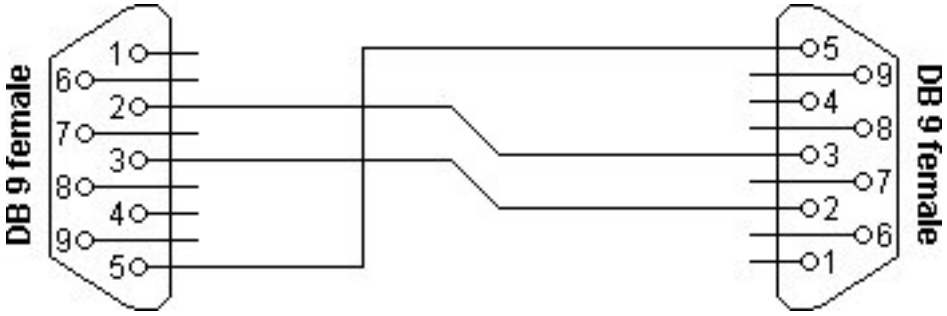
"Full Handshaking Null Modem"

Sometimes null modems are connected with "loop back" connections to "fake" the handshaking as follows:



"Loop back Handshaking Null Modem"

Sometimes designers actually don't bother with the handshaking signals at all. They don't program the equipment to either look at or to generate the signals:



"No Handshaking Null Modem"

This lack of standard for a connection often gives one fits when configuring a system! Maybe you have been there? For our project we will ignore handshaking. That will make it simpler for us, but still not at all trivial.

RS232 transmits a sequence of bytes, each of which is transmitted as a sequence of pulses. These pulses are sent at a selected Baud (or bit) rate:

```
Baud rate: Max speed of transmission of bits:
    Typically 110, 300, 1200, 2400, 4800, 9600, 19200 bits/sec
```

The protocol for the sequence of pulses representing the bytes of ASCII Characters is:

```
Start bit:
    A first bit always of the same polarity for equipment to sync on

Data Bits:
    The useful data follows the start bit:
    Typically 5, 6, 7, or 8 bits

Parity Bit:
    A bit used for error checking:
    Even Parity, Odd Parity, Mark Parity, Space Parity, No Parity

Stop Bits:
    The trailing bits after the data and parity to ensure time to "catch"
    data between bytes: Typically 1, 1.5, or 2 bits
```

Understand that both communicating devices must be using the same protocol parameters for the communication to be successful.

Here is an example Java program that sends characters or strings of characters over the cable. You may want to use a version of it to generate your characters on the host computer:

```
import java.io.*;
import javax.comm.*;
public class SimpleWrite
{
    public static void main(String[] args) throws Exception
    {
        OutputStream outputStream;
        outputStream = get_the_serial_port();
        byte[] data = {'a'};      // string sent
        for (int i = 0; i < 1000; i++)
        {
            outputStream.write(data);
            System.out.println ("i = " + i);
            Thread.sleep(1000);   // milliseconds suspended
        }
    }
    public static OutputStream get_the_serial_port() throws Exception
```

```
    {
        CommPortIdentifier portId;
        portId = CommPortIdentifier.getPortIdentifier("COM1");  // COM Port #
        SerialPort serialPort;
        serialPort = (SerialPort) portId.open("SimpleWriteApp", 2000);
        serialPort.setSerialPortParams(9600,   // baud rate
                SerialPort.DATABITS_8,
                SerialPort.STOPBITS_1,
                SerialPort.PARITY_NONE);
        return serialPort.getOutputStream();
    }
}
```

This program uses the javax.comm package. It is not included in the java JDK(jre). It can be obtained at:
http://java.sun.com/products/javacomm/index.jsp

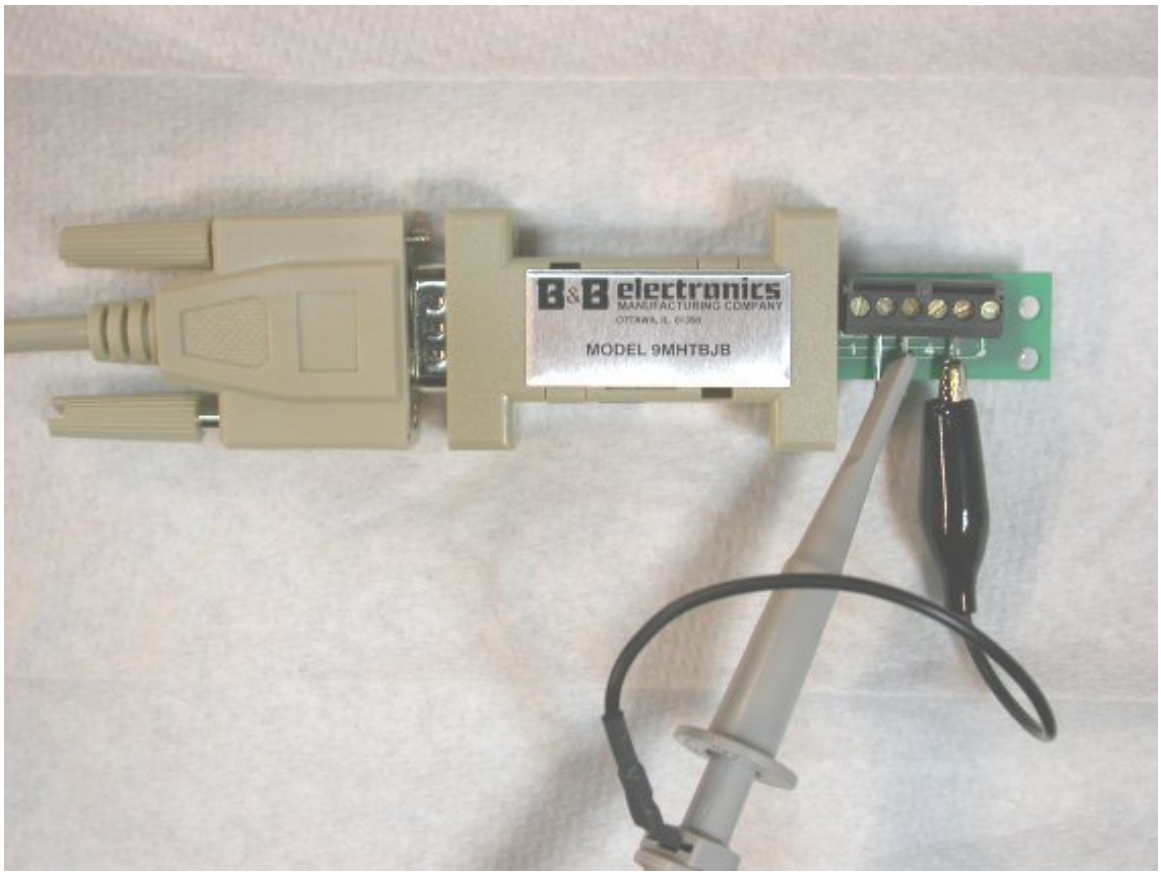
OBJECTIVE

We want to design and build an interface that will capture characters sent by the host computer via the RS 232 serial protocol, and display the data bits on LED's.

ASSIGNMENT (Work in pairs.)

1) Do the following procedure to understand the RS-232 protocol:

   a) Hook channel 1 of the analog oscilloscope to the output of a serial port (Ground is pin 5). The transmitted signal is on pin 3. Here is a picture of our oscilloscope connected to the serial cable from our computer.



Hook channel 2 to the pulse generator output and set the pulse rate to approximately 9600 Hz.

b) Transmit a character using Java Comm Library (javax.comm is installed on the laboratory computer JDK's). Describe what you see. Draw a picture of your wave form identifying the frequency, voltage between the transmitted character, the value of a "1" and a "0" level, the start bit, the data bits and their order and orientation, the partiy bit, and the stop bits. Explain how you know those identifications are correct. How did you use the pulse generator display to confirm your identification of the bit rate?

c) Transmit a zero (the number 0, not the character '0'), and record the result.

d) Choose other characters to transmit, and record the results.

e) Choose alternate values for EACH parameter: Baud rates, data bits, parity, and stop bits.

Record the results and observations.

2) Design a logic circuit that will capture a character sent. Each time a new character is sent, it should replace the last value. Lets assume the protocol is 1 start bit, 8 data bits, no parity, and two stop bits.

   Use a 1489 Line Receivers to buffer the signal from the RS-232 line. The voltage levels on the RS-232 cable are approximately 15 volts. The 1489 Line Receiver will handle these and give you a ‘1’ level consistent with 5 volt logic.

   I would use a 74HC163 4-bit synchronous binary counter for the state machine to control your circuit. The 74HC163 has a synchronous reset. That means that the reset occurs on the next clock pulse after the reset signal appears on the reset pin.

   I would use a 74HC164 8-bit synchronous shift register to capture the character bits. The outputs can be connected to leds in series with 1k current limiting resistors (we don’t want to burn out the leds).

3) After you are comfortable that your design should work, build and test your circuit.

REPORT

Each individual should submit a formal report. Your report should include documentation of your lab results.

Notes:

1) In general, don’t leave control inputs open. They should be connected to a logical 1 or a logical 0.

2) The 74HC163 Synchronous Binary Counter has a synchronous reset, i.e., it resets when the reset input is a logical 0 on the rising edge of the clock.

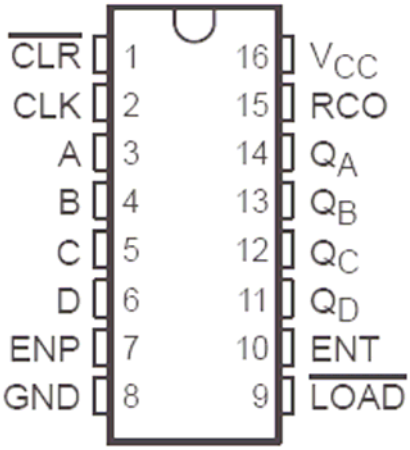
3) The 1489 is an inverting buffer. It accepts an input significantly higher than a logical 1, like the15 volts that the RS-232 line provides. The control inputs on it SHOULD be left open!

4) Our chip inventory includes:

74xx00	2 Input Nand Gates (4)
74xx02	2 Input Nor Gates (4)
74xx04	Invertors (6)
74xx08	2 Input And Gates (4)
74xx32	2 Input Or Gates (4)
74xx74	D Flip Flops (2)
74xx109	J/K Flip Flops (2)
74xx112	J/K Inverted Clock Flip Flops (2)
74xx125	Tri-state Buffers (4)
74xx163	4 Bit Sync Binary Counter with Sync Reset (1)
74xx164	8 Bit Synchronous Shift Register (1)
MC1489	Inverting Line Receivers (4)

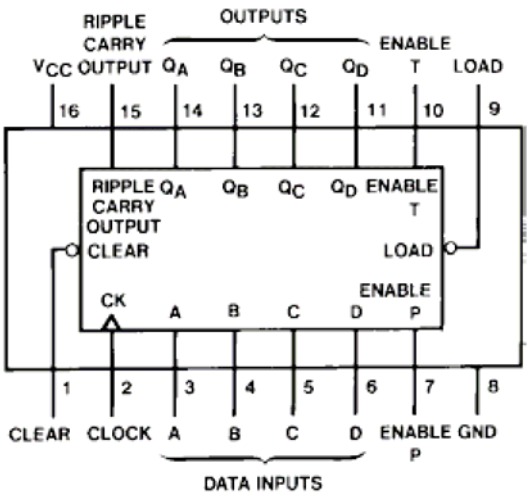
74HC163 4 Bit Synchronous Binary Counter:

http://rocky.digikey.com/WebLib/Texas%20Instruments/Web%20data/SN74HC163.pdf



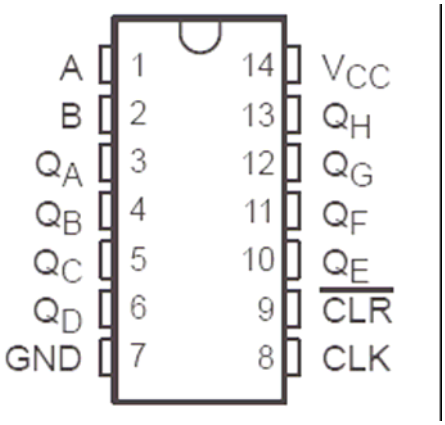
CLK	CLR	ENP	ENT	Load	Function
↑	L	X	X	X	Clear
X	H	H	L	H	Count & RC disabled
X	H	L	H	H	Count disabled
X	H	L	L	H	Count & RC disabled
↑	H	X	X	L	Load
↑	H	H	H	H	Increment Counter

H = HIGH Level
L = LOW Level
X = Don't Care
↑ = LOW-to-HIGH Transition



74HC164 8 Bit Synchronous Shift Register:

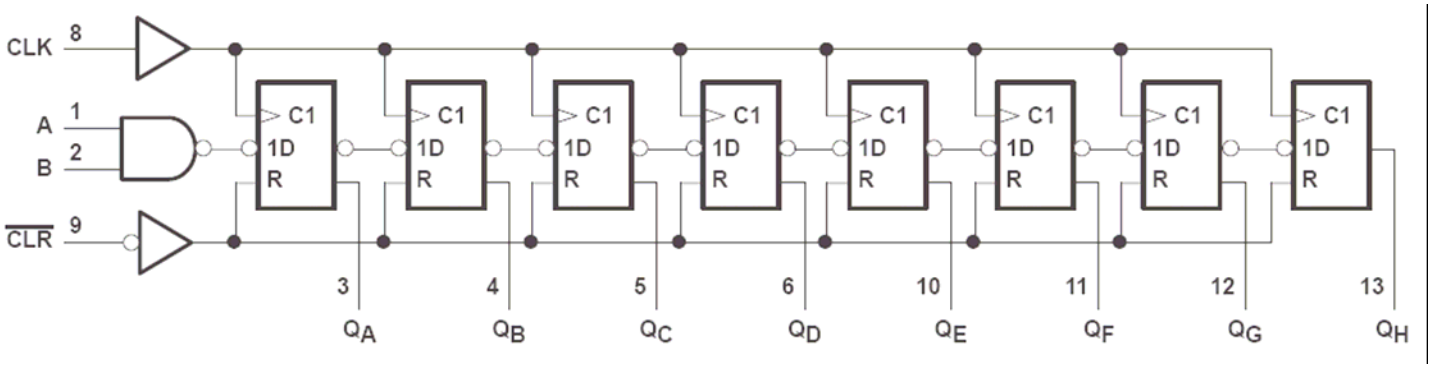
http://rocky.digikey.com/WebLib/Texas%20Instruments/Web%20data/SN74HC164.pdf



FUNCTION TABLE

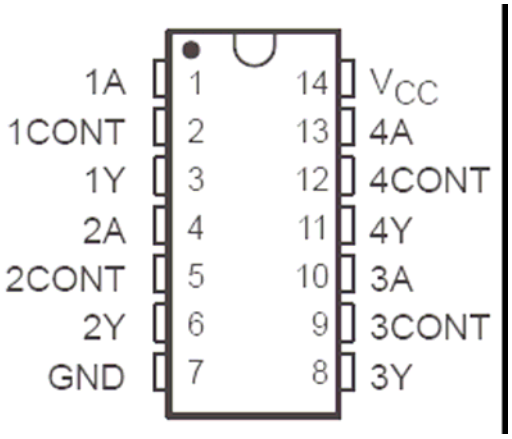
INPUTS				OUTPUTS		
CLR	CLK	A	B	Q_A	$Q_B \ldots Q_H$	
L	X	X	X	L	L	L
H	L	X	X	Q_{A0}	Q_{B0}	Q_{H0}
H	↑	H	H	H	Q_{An}	Q_{Gn}
H	↑	L	X	L	Q_{An}	Q_{Gn}
H	↑	X	L	L	Q_{An}	Q_{Gn}

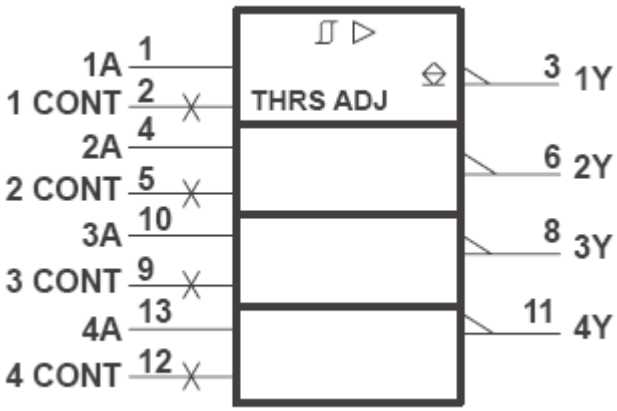
Q_{A0}, Q_{B0}, Q_{H0} = the level of Q_A, Q_B, or Q_H, respectively, before the indicated steady-state input conditions were established
Q_{An}, Q_{Gn} = the level of Q_A or Q_G before the most recent ↑ transition of CLK: indicates a 1-bit shift



MC1489N Quad Line Receiver:

http://focus.ti.com/lit/ds/symlink/mc1489.pdf

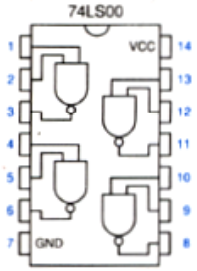




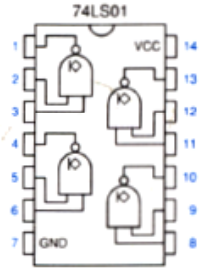
Note: The control inputs on it SHOULD be left open!

TTL Combinational SSI Devices

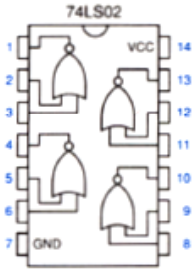
74LS00



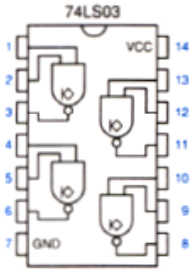
74LS01



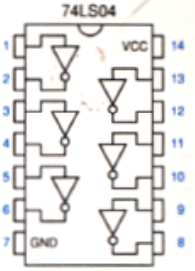
74LS02



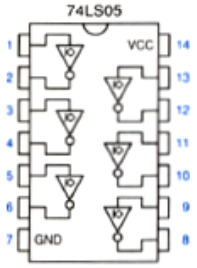
74LS03



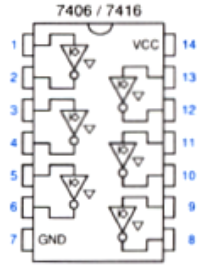
74LS04



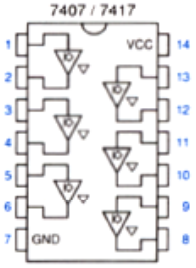
74LS05



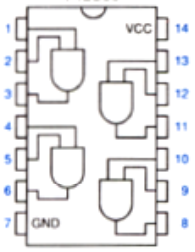
7406 / 7416



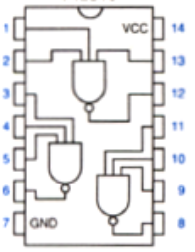
7407 / 7417



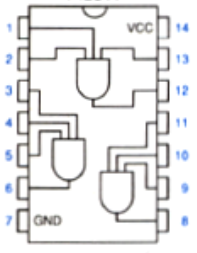
74LS08



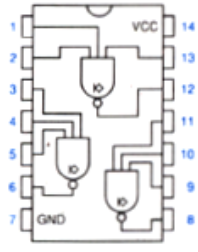
74LS10



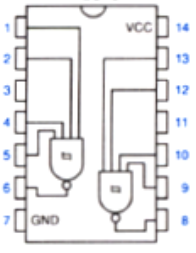
74LS11



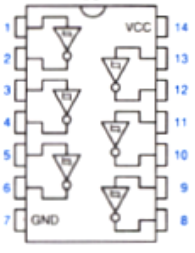
74LS12



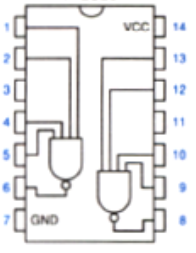
74LS13



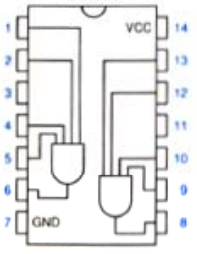
74LS14



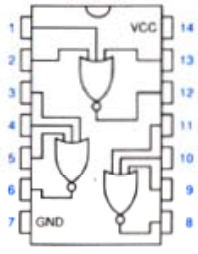
74LS20



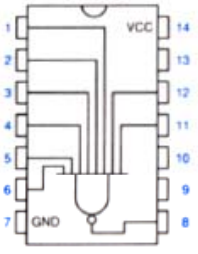
74LS21



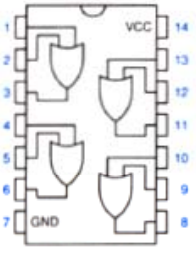
74LS27



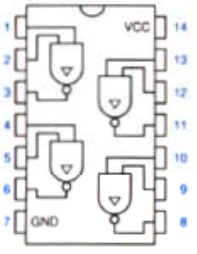
74LS30



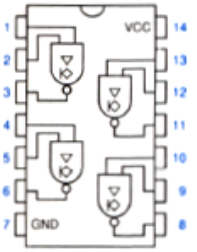
74LS32



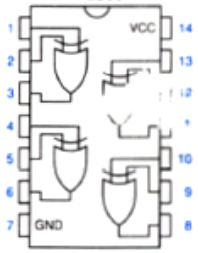
74LS37



74LS38

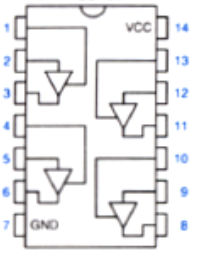


74LS86

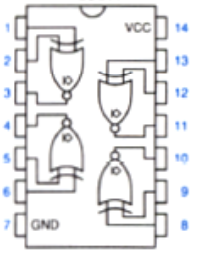


74LS125

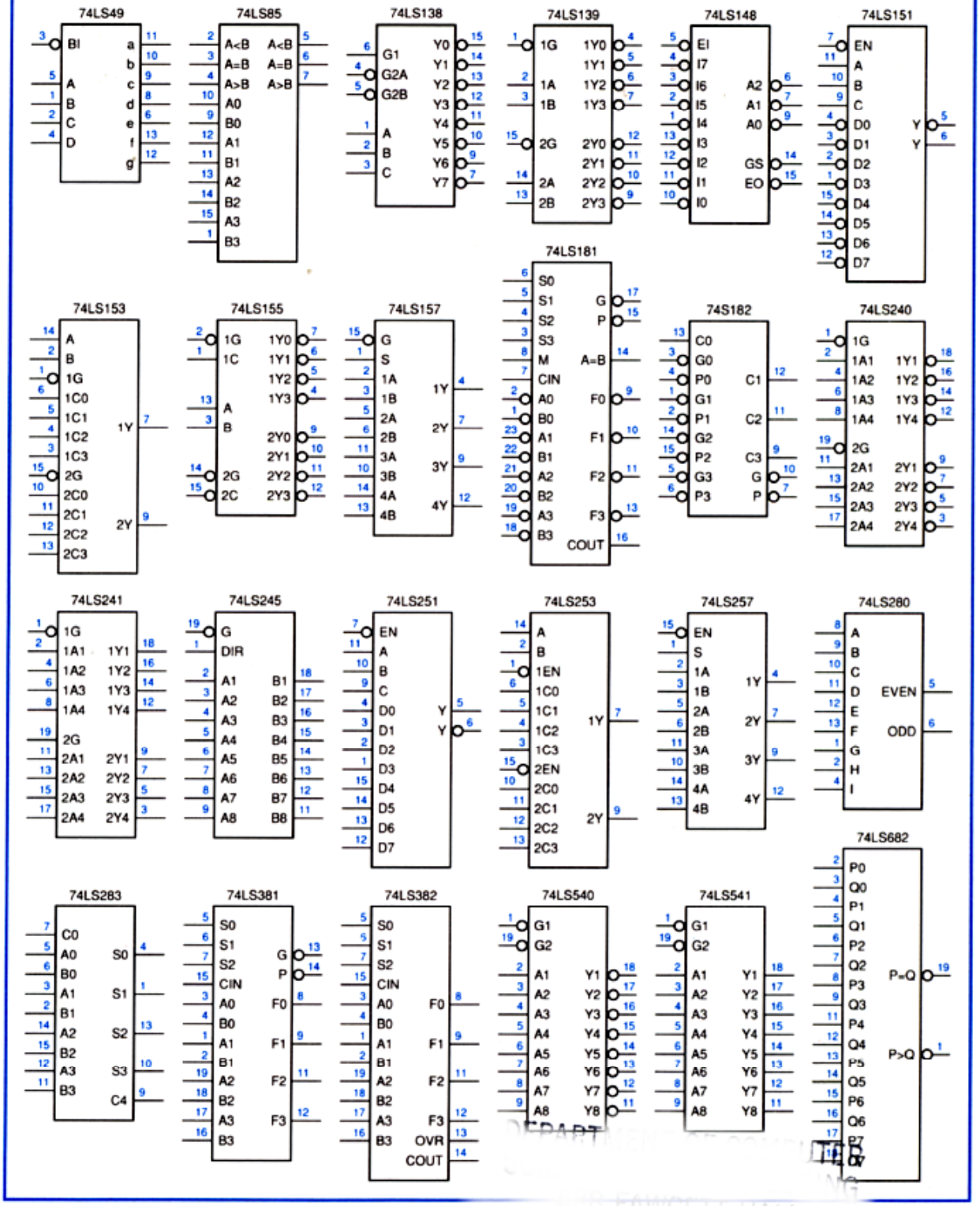
74LS126



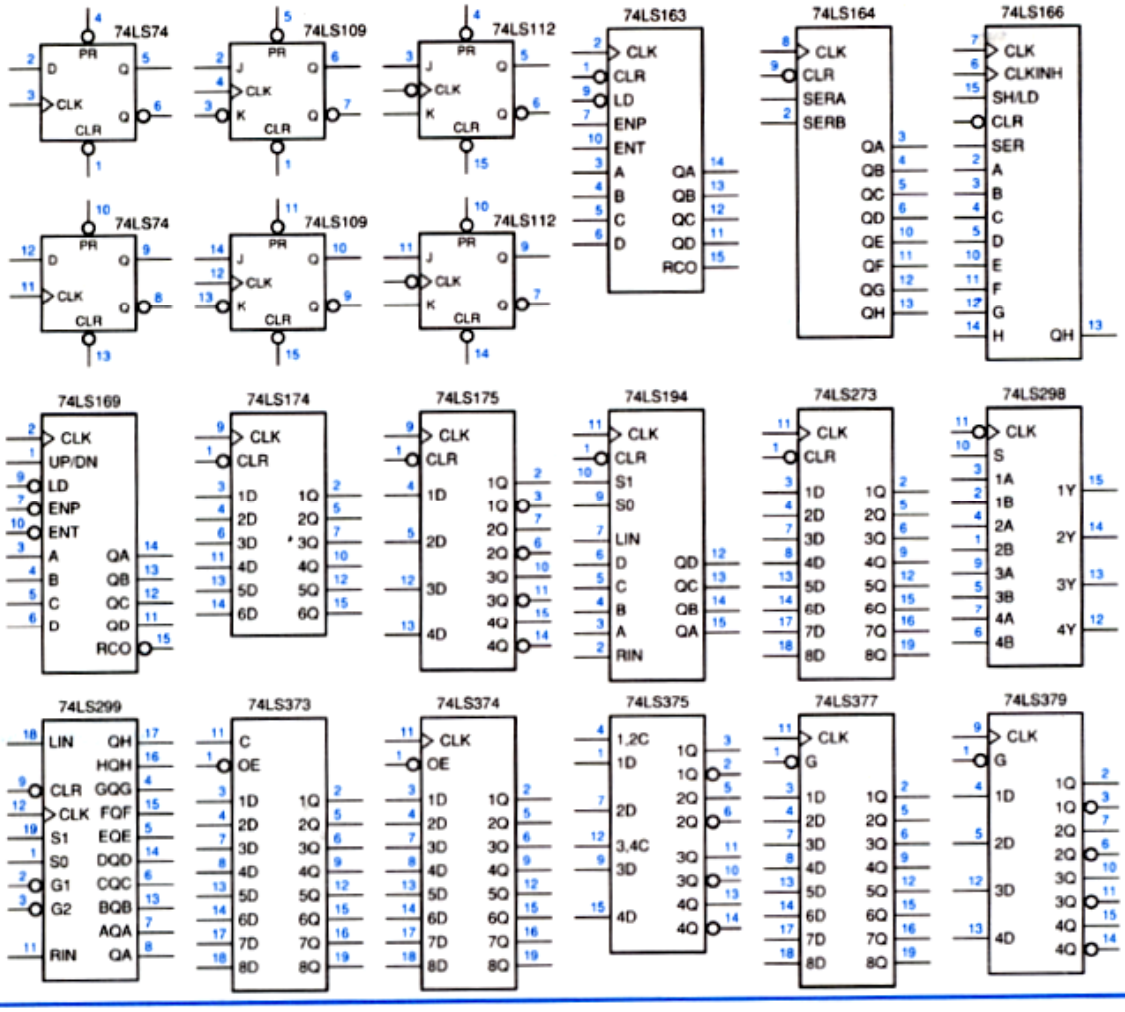
74LS266



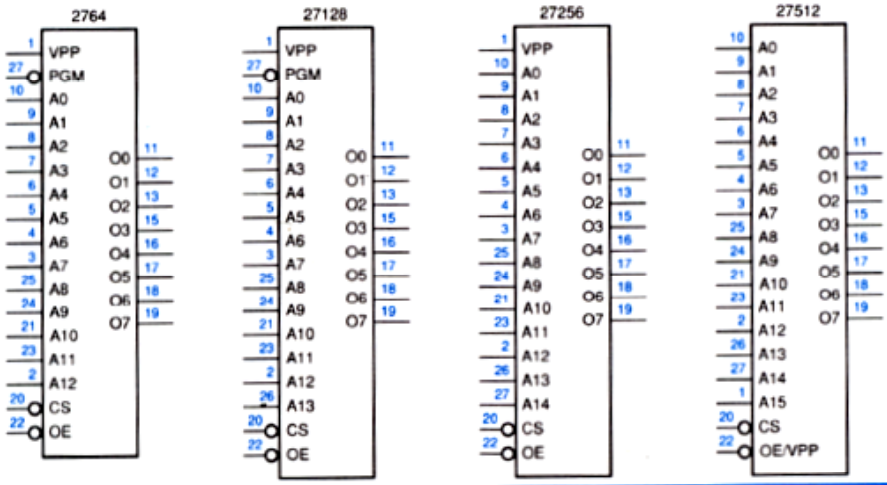
TTL Combinational MSI Devices



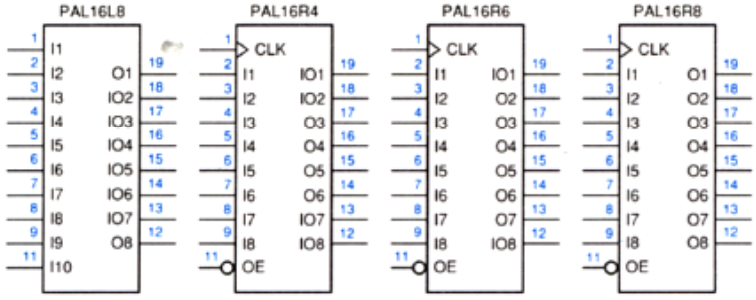
TTL Sequential MSI Devices



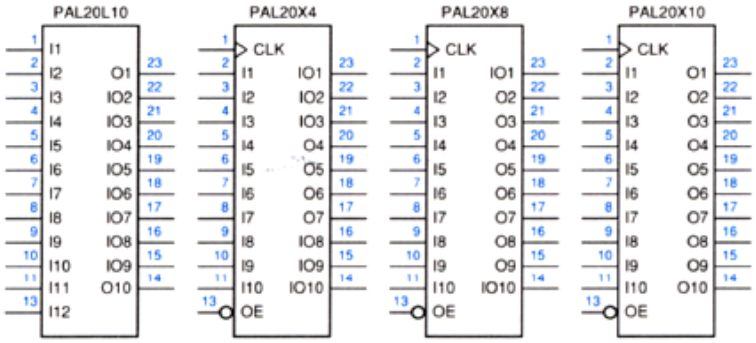
EPROMs



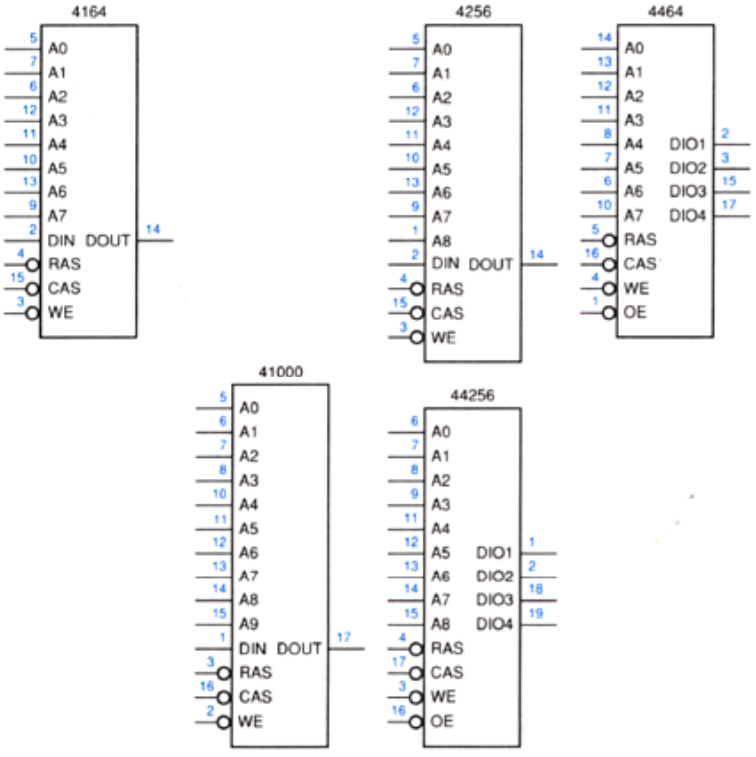
16R-Series 20-pin PLDs



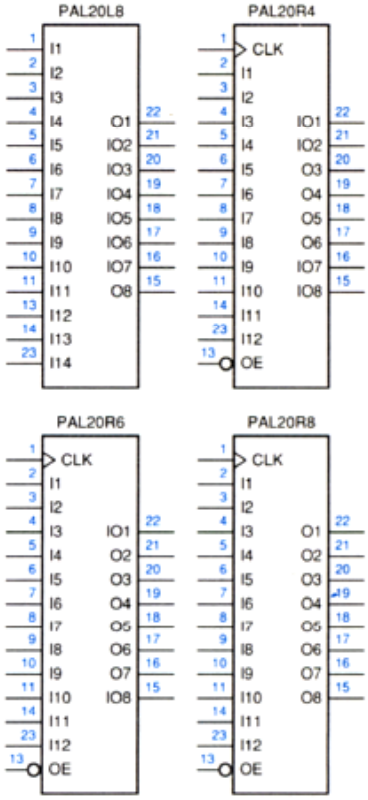
20X-Series 24-pin PLDs



Dynamic RAMs



20R-Series 24-pin PLDs



Static RAMs