**SERCON Description, Theory of Operation and Build Procedure**

2018.09.05 by Bob Bell

Board Version 1

**Background**

SERCON was originally short for SERial CONsole. While building the 6502 S-100 Computers CPU board, I came to the realization that I could not run the 6502 board in the same bus as my Z80 CPU, since my Z80 board is not capable of handing off control of the bus to other bus masters like some others. So the 6502 can only be a bus master and never a temporary master in my system. This means I must physically remove my Z80 board to begin testing the 6502 CPU board. Doing so, however, also removes my console serial port as it resides on my Z80 CPU board. The solution was to replace the Z80 CPU board console port with another console port. In order to not have to make any alterations to the console port drivers in my ROM monitor and CP/M BIOS, I determined that the replacement console hardware would have to mimic exactly the function of my existing hardware. After some searching, I could not find anything in existence that had the same characteristics. Since I didn’t want to re-write any software, all programmable serial consoles, like the S-100 Computers Serial I/O board were out. I ran across the Compupro Interfacer I - it had the hardware functions I wanted, but, of course, is not readily available. So I decided to create a board based roughly on this design. This included two hardware-configured serial ports with an interesting feature that permits the hardware configuration to be modified by software if desired. The old design had the capability to generate interrupts if needed, and used very inexpensive UART chips. These features were duplicated. To this design, I added several features that should make the board appealing to a larger number of users looking for a simple serial interface solution. This includes the ability to address the serial ports at any 8 bit I/O address, a 16-bit board address decoder option, the ability to put any control signal on any bit and at any polarity, and the ability to put the status bits on any bit and at any polarity. I also made provisions for either a 148x style RS-232 interface, or the more modern MAX232 RS-232 interface. In the theme of trying to be as versatile as possible, I also included jumpers to permit any RS-232 signal to be routed to any of the off-board serial interface pins.

After the serial ports were tested and functioning, I decided to add a parallel printer port. The design of this was based on a parallel printer port I had built many years ago and is still running in my first S-100 machine. While doing research for this part of the board, I discovered that my original design was not 100% compliant with the current IEEE 1284 (2006) standard for parallel printer ports. So I altered my original design to be compliant, but included jumpers to make the new design backward compatible to my old hardware, so when I use the board, I will not have to make changes to my CP/M BIOS until I am ready. As with the serial ports, the I/O port address of this hardware is selectable to any 8-bit port address, and has the same 16-bit I/O port address option. To accommodate as many variables as possible, the control signals are configurable to any bit and any polarity. One of the inadvertent design decisions of many years ago was to use the write pulse to the data output latch as the Strobe signal to the printer. I never had any problem with it, and discovered that IEEE 1284 specifies a particular timing be followed for the Strobe signal. My old hardware (and this design) did not meet those specifications, and, in fact, the generation of Strobe from the latch write line is not mentioned. Rather, the standard shows the use of a control line connected to Strobe that is programmatically set low then high to generate the Strobe signal. I decided to implement both methods and let the builder choose which one to use. The old method, now called Auto-Strobe, has been updated to bring the timing closer to the specification. Lastly, if needed, a flexible interrupt circuit has been included that permits generating a “printer ready” interrupt on any Ack or only on Ack after Strobe. It also permits the user to select Ack polarity – rising edge or falling edge to generate the interrupt.

A random number generator was added last. After much research and testing, and learning more about statistics than I ever wanted to know, I came up with a design that lets the builder of the board choose the configuration of the random number generator. It can be built in one of three ways, with several options within each configuration. None of the configurations pass the chi-squared test for randomness, but they come very close. None of the configurations exclude the others, and if built to option 3 (8 independent generators), any of the other options may be configured by changing jumpers (and adding or removing one IC.) All three options generate an 8-bit number. The first option uses a single white noise source to clock an 8-bit counter. This is the least expensive method and works fine for low volume reads. The second option runs two noise generators clocking two 4-bit counters. The third option runs one noise generator for each bit. Divide by 2 circuits can be enabled on this option to help even the distribution of 0’s and 1’s. For all options, Schmitt Triggers can be jumpered in to improve the noise signal before clocking flip-flops or counters.

**Board Description Details and Theory of Operation**

Board General:

The board is an IEEE-696 compliant 8-bit I/O slave with an option to add 16-bit I/O addressing. If the 16-bit option is installed, it can be enabled or disabled by jumper.

All configuration is done with jumpers on various types of male headers. This method of configuration was chosen for several reasons:

1. Less expensive than dip switches.
2. Provides a positive indication of logic. The jumper in place guarantees a particular bit is low and there is no guessing at the meaning of the legend on a dip switch.
3. Takes less board real-estate.

All bus logic input connections are no more than a single LSTTL load.

All bus logic output connections can drive up to 20 proper LSTTL loads.

In the two serial ports, there are a number of jumpers referred to as “flex jumpers”. This was devised as a way to impart the versatility the board needed without taking up too much real estate. These are similar to the common shorting shunt jumpers (hereafter referred to as “jumpers”) used for configuration of many S-100 boards, including other areas of this board. The difference is that these jumpers are flexible, each made from short pieces of small-gauge wire (stranded works best) and a pair of wire-applied female connectors. The length of the wire can be adjusted as they are made, and the flexibility of the jumper means it can be used to span several pins of a header to connect non-adjacent pins.

Details for making a flex jumper:

1. Cut a piece of 22 or 24 gauge stranded hook-up wire to 1” long (or as long as it is needed).  
2. Strip the insulation from both ends about 5/32”.  
3. Crimp one AMP 104480-3 (or similar) terminal onto each end.  
4. (optional) Cut a ½” long piece of 1/16” heat shrink tubing.  
5. (optional) Slide the heat shrink tubing over the terminal and heat until it shrinks over the terminal.

The tool for crimping the terminals is a Hanlong HT-202A, Jameco 99443. It is a little tricky until one gains some experience.

Shown above as optional, the heat shrink tubing helps prevent inadvertent shorts between the terminals of adjacent flex jumpers.

It would also be possible to use wire-wrap for these jumper connections as long as the wire-wrap tool and wire are appropriate for 0.025” square pins on a 0.1” pitch.

Serial Ports:

The two serial ports are identical. The description here is for serial port 1, but applies equally to serial port 2.

A transmit/receive baud rate clock is generated from a 4.9152 MHz oscillator (U49) and divided down by U50 and U51 to provide baud rates of 38.4K, 19.2K, 9600, 4800, 2400, 1200, 600 and 300 baud. The baud rate is selected by jumper set JS16, a 2x8 duel row header.

The UART, U22, chosen for this design is the very common and inexpensive TR1602 or IM6402 or one of several identical (or nearly identical) cousins. Early versions of this chip needed -12 volt power on pin 2, and a -12 volt power source is provided as a build option (described later). If the -12 volt power supply is required at first, but the UART is changed later to one that does not need it, jumper J15 is available to remove this supply. As of the writing of this document, the following UARTs have been tested successfully:

* Western Digital TR1602-PL (requires -12V power)
* Intersil IM6402IPL
* RCA CDP6402CE
* General Instruments AY-5-1013 (requires -12V power)

As mentioned briefly above, the UART can be located anywhere in the I/O address range of the typical 8-bit S-100 processor. Each serial system requires two consecutive I/O ports in this range. Port selection is done with the first seven in a set of eight jumpers at JS13. There is one read/write port (the odd-numbered port) for status (read) and control (write) and the next consecutive port (even) for data. The eighth jumper in JS13 allows these ports to be swapped. U43 is the address comparator, and U30 decodes to the four I/O port control signals:

\*WRCB1 – Write Byte to Control Register  
\*WRDT1 – Write Data to UART  
\*RDST1 – Read Status Byte  
\*RDDT1 – Read Data from UART

This UART is hardware programmable, making its configuration entirely possible by jumpers, without any software. The control port is implemented as a pair of 4-bit registers (U20 and U21) that have both true and inverted outputs. These outputs appear on a 16-pin duel row header (J17) with true and inverted next to each other. These signals are jumpered to J16, the desired control inputs to the UART, via flex jumpers. These jumpers are provided to permit the placement of any control signal on any bit and at any polarity. The registers are reset by the S-100 \*RESET and \*SLVCLR signals so at power on or system reset, each control signal can assume either high or low. Thus if the control port is never written, the configuration would be entirely set by hardware. Subsequently, writing to the control port can override the hardware settings for software control of the configuration if desired.

Table of Control Signals:

U1RIE – UART 1 Receiver Interrupt Enable. Enables logic that can interrupt the CPU when the UART receiver has an incoming byte to process.  
U1TIE – UART 1 Transmitter Interrupt Enable. Enables logic that can interrupt the CPU when the UART transmitter buffer register is empty and can accept a new byte to send.  
U1SCA – UART 1 Serial Control A. Can be used to send a RS-232 control signal. Depending on user configuration of jumpers, this could be, for example, RTS.  
U1SCB – UART 1 Serial Control B. Can be used to send a RS-232 control signal. Depending on user configuration of jumpers, this could be, for example, DTR.  
U1SB – UART 1 number of Stop Bits. Programs the number of stop bits for the UART. A 0 sets it for 1 stop bit; a 1 sets it for 2 stop bits.  
U1PI – UART 1 Parity Inhibit. A 0 enables parity; a 1 disables parity.  
U1EP – UART 1 Even Parity. A 0 sets odd parity; a 1 set even parity.  
U1NB – UART 1 Number of data Bits. A 0 sets 7 data bits; a 1 sets 8 data bits.

For status reads, a pair of bus buffers, one inverting (U32) and one non-inverting (U33), provides the flexibility of either true or inverted status bits. Both polarities of each bit appear on a 16-pin duel row header at J24. Flex jumpers are used to connect the desired polarity of each bit to the desired data bit on J23.

Table of Status Signals:

SSA1 – Serial Status A UART 1. Can be used to receive a RS-232 status signal. Depending on user configuration jumpers, this could be used, for example, as CTS.  
SSB1 - Serial Status B UART 1. Can be used to receive a RS-232 status signal. Depending on user configuration jumpers, this could be used, for example, as DSR.  
PE1 – Parity Error UART 1. When true, indicates the UART receiver detected a parity error in the current receive operation. This bit is never asserted by the UART if Parity is disabled.  
FE1 – Framing Error UART 1. When true, indicates the UART receiver detected a framing error in the current receive operation.  
OE1 – Overrun Error UART 1. When true, indicates the UART receiver detected an overrun error in the current receive operation.  
DAV1 – Data Available UART 1. When true, indicates the UART has received a complete byte from the RS-232 interface, and the byte is available to be read from the data port.  
TBE1 – Transmitter Buffer Empty UART 1. When true, indicates that the UART transmitter buffer is empty and can accept a new byte to transmit.  
EOC1 – End of Communications UART 1. Transmitter has completed sending all data.

The RS-232 interface and level converters can be built in two mutually exclusive ways. The first way uses the legacy 1488 and 1489 type chips at U12 and U11. This chipset requires +12 and -12 volt power supplies, and these are provided as build options (see below). The second way uses the newer MAX232 chip at U13 and U14 that contain on-board power converters, and thus do not require the separate power supplies. However, they require four capacitors each to operate their power converters. All RS-232 signals appear on J5, a 9-pin single-row header. Adjacent to this header is a 10-pin single-row header, J6 that corresponds to the pins of the 10-pin duel-row header, P2, for connection to the off-board cable. Flex jumpers are used to connect the desired RS-232 signals of the 9-pin header to the desired pins on the 10-pin header. A set of two 3-pin jumpers, J7 and J8 are provided to steer the +12 and-12 volt potentials from the power supplies or the Max chip if these signals are needed at J5 to force any pins to a particular condition. The +12 and -12 pins are current-limited to about 10mA.

Table of RS-232 signals on the 9-pin header:

ZDO1 – Data Out. RS-232 Serial Data Out (from board to external device)  
ZDI1 – Data In. RS-232 Serial Data In (from external device to board)  
ZCO1A – Control Out A. RS-232 Control Signal A Out (from board to external device). This could be, for example, RTS.  
ZCI1A – Status In A. RS-232 Status Signal A In (from external device to board). This could be, for example, CTS.  
ZCO1B – Control OUT B. RS-232 Control Signal B Out (from board to external device). This could be, for example, DTR.  
ZCI1B – Status In B. RS-232 Status Signal B In (from external device to board). This could be, for example, DSR.

P2 is pinned to be compatible with the same cables used by the IBM PC/AT type computers for connecting the board to a DB-9 male or properly prepared DB-25 male connector that are typically installed on the rear panel ready to make connections to peripheral equipment.

Note that if the MAX232 build option is chosen, U14 is shared between serial port 1 and Serial port 2. This is accommodated properly in the Bill of Materials and Parts List.

Parallel Port:

The parallel port is IEEE 1284 compliant for an “SPP” style parallel printer interface. It requires four consecutive I/O ports and can be located anywhere in the I/O address range of the typical 8-bit S-100 processor. This is done with a set of six jumpers, JS12. (There are eight jumpers in the set with two unused.) An address comparator, U42 and port decoder, U27, create four port read and four port write signals that terminate on two 4x3 jumper sets, JS9 and JS10, permitting any of the four port read or write signals to be used for controlling the parallel port.

The parallel port control signals are:

RDPS – ReaD Parallel Status. Activates U10 to place the parallel port status bits on the data bus.  
WRPC – Write Parallel Control. Writes the contents of the data bus to U19 and U18A.  
WRPD – Write Parallel Data. Writes the contents of the data bus to U9, which sends the data off-board to the printer via P1.

A fourth I/O port read control signal from the port decoder was utilized to implement the Random Number Generator. This is described below in the Random Number Generator section.

The parallel status port has the bits arranged as shown in the following table:

D0 – unused  
D1 – unused  
D2 – Interrupt Control Status  
D3 – \*Fault signal from printer  
D4 – Select signal from printer  
D5 – PERROR signal from printer  
D6 – \*ACK signal from printer  
D7 – BUSY signal from printer

The parallel control signals from U19 are both true and inverted polarity, and appear on the pins of JS11. The user may select either polarity for driving the four parallel port control signals off the board:

D0 - \*STROBE  
D1 - \*AUTOFEED  
D2 - \*INIT  
D3 - \*SELECTIN

U18A gets D4 from the data bus. This is used as the Interrupt Enable control. When D4 is high, logic is enabled that can interrupt the CPU when the printer becomes ready for more data. U18A is reset (Interrupts disabled) on system reset or power-on-clear.

U18B is the Interrupt Control Flip-Flop. It can be used to ensure that an interrupt only occurs on ACK after a byte has been written to the parallel port. J14 lets the user select this feature or Interrupt on ACK at any time. J13 permits the selection of either rising edge or falling edge of ACK to trigger the control flip-flop.

The IEEE 1284 standard describes a three-step process for clocking data to the printer. First the data is written to the output latch, U9, then D0 of the control port is written high (or low depending on the position of JS11 pin 2) to set the \*STROBE signal low to send the data in U9 to the printer, then D0 of the control port is reversed to de-assert \*STROBE. However, many years ago, prior to the standard, my original printer port design simply connected the \*STROBE line to the clock of the output register, so data was written to the register on the rising edge of the pulse, and the printer would accept the data on the falling edge. This has always worked very reliably with several different printers. So, this method was included in this design as a feature called Auto-strobe. To use it, JS11 pin 2 (strobe polarity select) is left disconnected, and J3 is jumpered to send the output register clock pulse to the \*STROBE pin. A simple R/C pulse-stretcher brings the Strobe pulse timing into closer conformity with the IEEE standard.

Another “mistake” discovered in my original parallel port interface design was the location of the busy bit in the status word. I had it on bit 5, but the standard now puts it on bit 7. So for this implementation, bit 7 is the normal setting, but J5 can be used to move it to bit 5 to work with a printer driver expecting to see the Busy signal on bit 5.

The off-board connector, P1, is a duel-row 26 pin header pinned identically to that used with the IBM PC style computers. This makes it possible to use IBM PC standard cable assemblies terminating in DB-25 female connectors for making the board to rear-panel connection.

Random Number Generator:

As described briefly in the Background section above, the random number generator can be built three different ways. All possibilities result in an 8-bit number at U7. When \*RDRNDM is active, the 8-bit number is gated onto the data bus to be read by the CPU. \*RDRNDM piggy-backs off the printer interface port decoders and is selected by JS9.

Up to eight noise generators are implemented on the board, typified by the components C32, Q13, R10, Q14 and one resistor of RN1 (pin 5). The transistors used are the very common and inexpensive 2N3904, but nearly any NPN transistor should work. The design is quite common for a white noise generator; tweaked on the breadboard for the highest frequency white noise. Output of the generator is coupled through C41 to an amplifier composed of Q16 and one resistor in RN3 (pin 5) and one resistor in RN2 (pin 5). The amplifier output is TTL level; it can be used to drive any one of several signal conditioning devices, depending on configuration.

Configuration 1 – Single Noise Source

This configuration uses the components listed above with the exception that the resistor networks are replaced by discreet resistors of the same value from pin 5 to pin 1, a minor cost savings. The BOM, parts list and board silk screen reflect this alteration for this particular configuration. In order to generate 8 bits from one noise source, the output of amplifier Q16 clocks the 8-bit counter U6. JS7 must be jumpered pins 3 to 4, and JS8 must be jumpered 4 to5. No other jumpers are needed. One variation is possible. JS7 may be jumpered pins 2 to 5 instead of 3 to 4. This puts a Schmitt Trigger between the noise source and the counter. The Schmitt Trigger ensures that the input always transitions between the high and low thresholds. It may or may not improve the randomness of the generator; tests have been inconclusive.

Configuration 2 – Two Noise Sources

To the components listed in Configuration 1, these parts are added: C31, Q9, R9, Q10, C39 and Q12. Three more discreet resistors of the same value as used in Configuration 1 above are added to the resistor network locations pins 6 to 10. JS7 remains jumpered 3 to 4, or 2 to 5 if the Schmitt Trigger is used, to clock the first 4-bit counter. JS8 is jumpered pin 3 to pin 6 to clock the second 4-bit counter, or 2 to 7 if the Schmitt Trigger is desired. (JS8 pin 4 to 5 is removed.)

Configuration 3 – Eight Noise Sources

In this configuration, all components are installed except U6, the 8-bit counter. J1 and J2 must be jumpered to bypass the counter. Each bit has the same options controlled by eight identical jumper sets. Each jumper set, typical of JS1 provides these possibilities:

Pins 3 to 4 – the bit comes directly from the noise generator.  
Pins 2 to 5 – the bit comes from the Schmitt Trigger.  
Pins 1 to 6 – the bit comes from a flip-flop, clocked by the noise source conditioned by the Schmitt Trigger, which guarantees a 50/50 duty cycle for each bit.

Because JS8 has eight pins, the numbering is different, but the jumper configuration is the same as the others if pins 4 and 5 are ignored.

Ancillary circuits:

All interrupt logic signals arrive at U25, an open-collector driver. There is a driver for each serial port interrupt and one for the parallel port. The outputs appear on J21, where flex jumpers are used to connect each desired interrupt signal to the appropriate S-100 Interrupt pin on J23: \*VI0 though \*VI7, \*INT or \*NMI.

The 5 volt power supply has three implementation options. If the board will be used in a machine with a 5 volt buss, then J27 and the filter capacitors are the only components needed. On a +8 volt bus, either U38 or U39 are required and J27 is not used. The linear regulator U38 requires a heat sink. The switching regulator, U39, does not. On a fully populated prototype board, except option A (16-bit I/O) and U6 removed for 8 source noise, and options D1 and D2 (148x RS-232 chips), current draw on the 5 volt supply was measured at 0.55 amps

If any of the options are built that require -12V (either Serial option with old-style UARTs or the 148x RS-232 options), then U40 and associated filter caps are needed. No measurement of current draw on this supply has been made.

If +12 is needed for any options (148x RS-232 or Random Number Generator), then U41 and associated filter caps are installed. No measurement of current draw on this supply has been made.

For CPUs with 16-bit I/O addressing capability, a 16-bit board base address decoder is provided using components JS15, RN7 and U46. If enabled by jumping J28 pins 1 to 2, when the eight high order address lines match the jumpers configured on JS15, IO16 will be true to enable the low address comparators already described above. When not enabled, or if this feature is not installed, then J28 must be jumpered 2 to 3.

**Board Construction Procedure and Checkout Notes**

While following these notes, please reference the SERCON BOM for component values and reference designators. The options on this board can be built in any order, with exceptions noted.

It is possible to construct this board using any one of several different methods:

1. Solder everything, insert all IC’s, insert board in bus and power on. With some luck, everything will work. If not, start troubleshooting using provided schematic and this document.
2. Solder the board a feature at a time, insert IC’s, insert board in bus and power on. With some luck, the option will work. If not, start troubleshooting using provided schematic and this document.
3. Solder all passive components, including IC sockets if used, insert just the active components needed for one option and test generally following the steps below for method D.
4. Follow the steps below, testing each option as it is built using a procedure that will guarantee that if any problems crop up, it will be much easier locating the issue.

Depending on the system being used to test this board, it may be necessary to use a bus extender so various locations on the board can be probed with test equipment and clip leads attached for some tests.

During the course of testing, the following test equipment and software resources may be needed:

1. DMM or similar to measure continuity and voltage.
2. Oscilloscope, logic analyzer and/or logic probe.
3. A program on the system this board is being tested in that can read and write I/O ports. (The designer’s SCP200, or SCP201 program can do this.)
4. A program on the system this board is being tested in that can send and receive ASCII to/from a serial port. (The designer’s SCP200, or SCP201 program can do this.)
5. A program on the system this board is being test in that can accept simple machine language programs, can execute them with at least one breakpoint, and can examine CPU registers and memory locations. A good choice for this would be SID/ZSID.

**Steps for the recommended construction method D:**

If any abnormalities are detected, do not go on to the next step until it has been resolved.

1. 🞏 Visually inspect the board carefully using a magnifying glass and strong light to find any potential problems with the board like bad runs (there should not be any), solder shorts between adjacent pads (there should not be any) and any other issues that would affect the operation of the board.
2. 🞏 With an ohmmeter, test that all power supplies are open:
   1. +8 to ground
   2. +5 to ground
   3. +16 to ground
   4. +12 to ground
   5. -16 to ground
   6. -12 to ground
3. 🞏 Determine which options will be installed and make sure all parts are on-hand. Note that the Parts list will automatically create a total count of parts needed based on the options chosen. The BOM will guide the builder as to what components are to be installed and where.
4. 🞏 If using IC sockets, these may be installed at any time prior to the chips being inserted. The BOM does not call out IC sockets, but the parts list does include them for purchasing.

**Board Base and 5 volt power Checkout. This must be completed before any other options:**

1. 🞏 Leaving all chips out for now, install components for board base and one of the +5 volt power options, as per the BOM. If using IC sockets, install these as well. The +5 power options, J1, J2 and J3, are listed later in this document. Return here when one of the +5 volt power options has been completed.
2. 🞏 Plug the board into an S-100 computer backplane and turn it on. Test for +5 volts on pin 14 of U26, U28 and U29, and on pin 20 of U45, U47 and U48.
3. 🞏 Power off. Insert U26 (74LS86), U28 (74LS08), U29 (74LS10) and U45 (74LS244).
4. 🞏 Insert board in bus and power on. Assuming a monitor program or OS is running, with oscilloscope, logic analyzer or logic probe…
   1. Examine U28 pin 8 and pin 11 for activity.
   2. Examine U29 pin 8 – should be high; U29 pin 12 should be low.
   3. Examine U28 pin 6 – should be high; U26 pin 3 should be low.
   4. Actuate the computer’s reset switch while observing U28 pin 6 – should go low.
   5. Actuate the computer’s reset switch while observing U26 pin 3 – should go high.
   6. Examine U45 pins 3, 5, 7, 9, 12, 14, 16 and 18. There should be activity on all 8 pins. In the very unlikely event that one or more is missing, it is possible that the running program may not actuate a particular address line. In this case, confirm the non-moving line by observing the corresponding input (see the schematic.)
5. 🞏 Power off. Insert U47 (74LS244) and U48 (74LS244).
6. 🞏 Insert board in bus and power on. Examine U47 pins 3, 5, 7, 9, 12, 14, 16 and 18 for activity on all 8 pins.
7. 🞏 This completes the checkout of the Board Base.

**Option A – 16 bit Address Decoder**

1. 🞏 Install components for Address Decoder as per the BOM, including ICs.
2. 🞏 Using jumpers on JS15, set the high-order I/O address. A jumper on is a low, off is a high.
3. 🞏 Testing of this option must be deferred until there is at least one function installed and checked (one serial port, the parallel port or the random number generator.) Come back to this location when this has been accomplished.
4. 🞏 Set jumper J28 ON.
5. 🞏 Insert board in bus and power on.
6. 🞏 Re-test one or more of the completed functions, now using a 16-bit address.
7. 🞏 The function should respond exactly as it did with an 8-bit address.
8. 🞏 This completes the checkout of option A, 16-bit Address Decoder.

**Option B - Serial Port 1 Base**

1. 🞏 Ensure JP3 (16-bit I/O) is jumpered 2 to 3 or OFF.
2. 🞏 Leaving all chips out, install components for Serial Port 1 as per the BOM.
3. 🞏 Insert ICs U32 (74LS240) and U33 (74LS244).
4. 🞏 Insert board in bus and power on.
5. 🞏 Prepare a simple logic generator: A short piece of wire or a clip lead with one end connected to ground (0V), like one of the Aux ground pins, and the other end connected to a 1K 1/4w resistor. This will be used to force the input on U32 and U33 to a known state. (Generally, unconnected inputs on TTL chips will float to a high level, but this is not guaranteed.)
6. 🞏 With a clip lead, temporarily connect U32 pin 1 to ground (0v).
7. 🞏 While observing the logic state on J24, touch the unconnected end of the 1K resistor of the logic generator to pin 8 of U11. (Yes, there is no chip here yet.) Observe that J24 pin 16 is high and J24 pin 1 is low. Check each pair of status signals in this manner. Here is a table to show all that needs to be checked:  
      
    Force logic on J24 pin observations  
    ----------------------------------------------------  
    U11 pin 8 1 – low 16 – high  
    U11 pin 6 2 – low 15 – high  
    U22 pin 13 3 – low 14 – high  
    U22 pin 14 4 – low 13 – high  
    U22 pin 15 5 – low 12 – high  
    U22 pin 19 6 – low 11 – high  
    U22 pin 22 7 – low 10 – high  
    U22 pin 24 8 – low 9 - high
8. 🞏 Power off. Remove U32 ground. Populate ICs U43 (74LS688) and U30 (74LS138).
9. 🞏 With up to seven 2-position jumpers on JS13, select an 8-bit I/O port base address that is currently unused in the computer this is being tested in. A jumper on the bit sets it low. Since the serial port occupies two consecutive ports, A0 does not factor in the base addressing. A0X can selectively invert A0. This provides the ability to swap the two ports. For now, put a jumper on these pins so A0 is passed on to the decoder unchanged.
10. 🞏 Insert board in bus and power on. With a logic probe or anything capable of observing momentary logic pulses, prepare to examine U30 pin 13. Idling, this should be high.
11. 🞏 With a monitor program, or anything that can output data to a port, write a byte (anything) to the I/O port addressed with JS13. A short low pulse should be detected.
12. 🞏 Observing U30 pin 12, output anything to base address + 1. A short low pulse should be detected.
13. 🞏 Observing U30 pin 11, input from base address. A short low pulse should be detected.
14. 🞏 Observing U30 pin 10, input from base address + 1. A short low pulse should be detected.
15. 🞏 Power off. Insert U20 (74LS175) and U21 (74LS175).
16. 🞏 Insert board in bus and power on. Looking at J17, observe all ‘+’ pins should be low and all the ‘-‘ pins should be high. This is the reset state of the UART control register.
17. 🞏 With a monitor or other appropriate program, output a non-zero byte to the I/O port base address.
18. 🞏 Observe all the ‘+’ pins – they should be a positive logic binary representation of the byte that was output.
19. 🞏 Try outputting several different combinations of bits to be sure they all change, the ‘-‘ pins are always the complement of the ‘+’ pins, and the binary representation on the pins is always as expected.
20. 🞏 Power off. Set initial hardware programming of UART using flex jumpers on J17 to J16. This will be the default mode of operation of the UART upon power-up or reset. Refer to the Table of Control Signals earlier in this document and determine which control signals need to be driven by which bit, and what the initial polarity will be. All bits should be connected, especially the four UART programming bits and the two Interrupt enable bits. For example, the following table shows a possible initial programming setup:  
      
    Signal Name Bit Default J17 pin J16 pin  
    -------------------------------------------------------------------------------------------------------  
     Receiver interrupt enable 0 Off CB0+ RIE  
     Transmitter interrupt enable 1 Off CB1+ TIE  
     Serial Control A 2 Off CB2+ SCA  
     Serial Control B 3 Off CB3+ SCB  
     Number of stop bits 4 1 CB4+ #SB  
     Parity (Inhibit) 5 No parity CB5- PI  
     Parity (Even) 6 Even parity CB6- EVP  
     Number of bytes in word 7 8 CB7- #B
21. 🞏 Populate U49 (4.9152 MHz Oscillator), U50 (74LS74) and U51 (74LS393).
22. 🞏 Insert board in bus and power on.
23. 🞏 With a ‘scope (preferred) or logic analyzer (or a logic probe if nothing else), probe the signals on JS15, pins 1 through 8. Observe continuously running square waves of descending frequency. Pin 1 should be about 614 KHz. Each one down the connector should be half the frequency of the one before.
24. 🞏 Power Off. If a UART requiring -12V is being used, go to and complete the construction of the -12V power supply, option L, before continuing with this option.
25. 🞏 Choose a baud rate (300 is recommended for now) by placing a jumper on one of the horizontal sets of pins on JS16.
26. 🞏 Populate U22 (IM6402 / TR1602 / AY-5-1013). If a UART requiring -12V is inserted, be sure to put a jumper on J15.
27. 🞏 Temporarily connect U13 pin 11 to pin 12. This connects the UART serial input to the serial output forming a TTL-level loop-back. Anything that is transmitted by the UART will be received immediately by the UART.
28. 🞏 Plug the board into the bus and power on.
29. 🞏 This step assumes a common UART hardware program, such as the one suggested in step 22, has been completed on J17 / J16. Using a monitor or other program capable of reading and writing to I/O ports, read port base +1 and note the value. Write a different value to port base +1. Read port base +1 again and observe it has changed to the value written. Write another byte to base +1 and read base +1 to observe the new value. Try as many different bytes as needed to be sure the UART is transmitting and receiving as expected.
30. 🞏 Using additional flex jumpers, connect status bits to data bus bits. The bits to be connected are highly dependent on serial port drivers or the settings of a communications program. Most drivers and programs will need to have access to at least the UART Transmit Buffer Empty status signal to know when it can write a byte to the UART, and the UART Receive Data Available status signal to know when a byte has arrived from the outside world. For example, a CP/M serial port driver that expects Receive Data Available on bit six would have J23 D6 jumpered to J24 DAV. Either polarity may be selected (+ for active high – for active low), depending on what the driver is expecting. Not all status signals need to be connected, only the ones that are needed.
31. 🞏 Insert board in bus and power on.
32. 🞏 Using a ‘scope or logic probe, examine U22 pin 19 (Data Available). Following a reset, this should be low. Examine U22 pin 22 (Transmitter Buffer Empty). The expected idle state of this pin is high.
33. 🞏 Using a program capable of entering and running simple machine code with breakpoints, such as SID/ZSID or some monitors, enter this simple program into memory at 4000H (ignoring comments, and substituting the board’s UART status address for base+0, and data address for base+1):  
     4000: IN A,<base+0> ;Read idle state of UART status  
     4002: LD B,A ;Save in B  
     4003: LD A,42H ;Anything in A will do  
     4005: OUT <base+1>,A ;Send byte to UART; momentarily sets Transmit Buffer Empty false  
     4007: IN A,<base+0> ;Read status from UART  
     4009: LD C,A ;Save in C  
     ;Set a breakpoint here. UART will finish sending the byte in a split second  
     400A: IN A,<base+0> ;Read status again now that byte was sent and received via loopback  
     400C: LD D,A ;Save in D  
     400D: IN A,<base+1> ;Read data byte from UART – this resets Data Available  
     400F: LD H,A ;Save in H to compare with transmitted byte  
     4010: IN A,<base+0> ;Read status  
     4012: LD E,A ;Save in E  
     4013: NOP  
     ;Set a breakpoint here to halt execution  
    Run this program at 4000 with a breakpoint set at 400A. (Do not attempt to step this program.)  
    At 4002, the status idle state is stored in B. TBE should be true (ready to transmit) and DAV should be false (no data received.) At 4005, a byte is written to the transmitter, and then the status is sampled immediately following and stored in C. This should show the TBE bit false, as it is not ready for a new byte when it is still transmitting. Also at this point, the receiver has not received the full byte yet, so DAV should still be false.  
    The breakpoint before 4009 is needed to ensure there is enough time for the UART to transmit through the loopback to itself without writing a time delay loop. After the break, start execution again with a breakpoint set at 4013. Reading the status at 400A after the breakpoint is hit should read TBE true again and now DAV true with a byte ready to read. At 400D, reading the data byte automatically resets the DAV bit in the UART, so reading status again at 4010, after reading the data, should result in the idle state again: TBE true, and DAV false. The data read from the receiver is stored in the H register and can be compared to what was sent. The status bits are saved in B, C, D and E for comparison after the program is executed. The program must be executed full-speed to capture the status and cannot be stepped. Success at this step proves that minimal operation status reads are good.
34. 🞏 Power Off and remove the board. This completes the checkout of option B, Serial Port 1 Base.

**Option C - Serial Port 2 Base**

Although it is possible to build Serial Port 2 before Serial Port 1, it is highly recommended that Serial Port 1 be assembled and tested first, since it contains the components and construction steps for the common baud rate generator.

1. 🞏 Ensure JP3 (16-bit I/O) is jumpered 2 to 3 or OFF. (If the 16-bit Address Decoder option A was installed and tested after one of the main functions was installed, then this jumper could remain set 1 to 2 or ON.)
2. 🞏 Leaving all chips out, install components for Serial Port 1 as per the BOM.
3. 🞏 Insert ICs U37 (74LS240) and U36 (74LS244).
4. 🞏 Insert board in bus and power on.
5. 🞏 Prepare a simple logic generator: A short piece of wire or a clip lead with one end connected to ground (0V), like one of the Aux ground pins, and the other end connected to a 1K 1/4w resistor. This will be used to force the input on U37 and U36 to a known state. (Generally, unconnected inputs on TTL chips will float to a high level, but this is not guaranteed.)
6. 🞏 With a clip lead, temporarily connect U36 pin 1 to ground (0v).
7. 🞏 While observing the logic state on J26, touch the unconnected end of the 1K resistor of the logic generator to pin 6 of U17. (Yes, there is no chip here yet.) Observe that J26 pin 16 is high and J26 pin 1 is low. Check each pair of status signals in this manner. Here is a table to show all that needs to be checked:  
      
    Force logic on J26 pin observations  
    ----------------------------------------------------  
    U17 pin 6 1 – low 16 – high  
    U17 pin 11 2 – low 15 – high  
    U24 pin 13 3 – low 14 – high  
    U24 pin 14 4 – low 13 – high  
    U24 pin 15 5 – low 12 – high  
    U24 pin 19 6 – low 11 – high  
    U24 pin 22 7 – low 10 – high  
    U24 pin 24 8 – low 9 - high
8. 🞏 Power off. Remove U36 ground. Populate ICs U44 (74LS688) and U31 (74LS138).
9. 🞏 With up to seven 2-position jumpers on JS14, select an 8-bit I/O port base address that is currently unused in the computer this is being tested in. A jumper on the bit sets it low. Since the serial port occupies two consecutive ports, A0 does not factor in the base addressing. A0X can selectively invert A0. This provides the ability to swap the two ports. For now, put a jumper on these pins so A0 is passed on to the decoder unchanged.
10. 🞏 Insert board in bus and power on. With a logic probe or anything capable of observing momentary logic pulses, prepare to examine U31 pin 13. Idling, this should be high.
11. 🞏 With a monitor program, or anything that can output data to a port, write a byte (anything) to the I/O port addressed with JS14. A short low pulse should be detected.
12. 🞏 Observing U31 pin 12, output anything to base address + 1. A short low pulse should be detected.
13. 🞏 Observing U31 pin 11, input from base address. A short low pulse should be detected.
14. 🞏 Observing U31 pin 10, input from base address + 1. A short low pulse should be detected.
15. 🞏 Power off. Insert U23 (74LS175) and U34 (74LS175).
16. 🞏 Insert board in bus and power on. Looking at J19, observe all ‘+’ pins should be low and all the ‘-‘ pins should be high. This is the reset state of the UART control register.
17. 🞏 With a monitor or other appropriate program, output a non-zero byte to the I/O port base address.
18. 🞏 Observe all the ‘+’ pins – they should be a positive logic binary representation of the byte that was output.
19. 🞏 Try outputting several different combinations of bits to be sure they all change, the ‘-‘ pins are always the complement of the ‘+’ pins, and the binary representation on the pins is always as expected.
20. 🞏 Power off. Set initial hardware programming of UART using flex jumpers on J19 to J18. This is the default mode of operation of the UART upon power-up or reset. Refer to the Table of Control Signals earlier in this document and determine which control signals need to be driven by which bit, and what the initial polarity will be. All bits should be connected, especially the four UART programming bits and the two Interrupt enable bits. For example, the following table shows a possible initial programming setup:  
      
    Signal Name Bit Default J19 pin J18 pin  
    -------------------------------------------------------------------------------------------------------  
     Receiver interrupt enable 0 Off CB0+ RIE  
     Transmitter interrupt enable 1 Off CB1+ TIE  
     Serial Control A 2 Off CB2+ SCA  
     Serial Control B 3 Off CB3+ SCB  
     Number of stop bits 4 1 CB4+ #SB  
     Parity (Inhibit) 5 No parity CB5- PI  
     Parity (Even) 6 Even parity CB6- EVP  
     Number of bytes in word 7 8 CB7- #B
21. 🞏 If a UART requiring -12V is being used, and it was not assembled for Option B, Serial Port A Base, go to and complete the construction of the -12V power supply, option L, before continuing with this option.
22. 🞏 Choose a baud rate (300 is recommended for now) by placing a jumper on one of the horizontal sets of pins on JS17.
23. 🞏 Populate U24 (IM6402 / TR1602 / AY-5-1013). If a UART requiring -12V is inserted, be sure to put a jumper on J20.
24. 🞏 Temporarily connect U16 pin 11 to pin 12. This connects the UART serial input to the serial output forming a TTL-level loop-back. Anything that is transmitted by the UART will be received immediately by the UART.
25. 🞏 Plug the board into the bus and power on.
26. 🞏 This step assumes a common UART hardware program, such as the one suggested in step 20, has been completed on J19 / J18. Using a monitor or other program capable of reading and writing to I/O ports, read port base +1 and note the value. Write a different value to port base +1. Read port base +1 again and observe it has changed to the value written. Write another byte to base +1 and read base +1 to observe the new value. Try as many different bytes as needed to be sure the UART is transmitting and receiving as expected.
27. 🞏 Using additional flex jumpers, connect status bits to data bus bits. The bits to be connected are highly dependent on serial port drivers or the settings of a communications program. Most drivers and programs will need to have access to at least the UART Transmit Buffer Empty status signal to know when it can write a byte to the UART, and the UART Receive Data Available status signal to know when a byte has arrived from the outside world. For example, a CP/M serial port driver that expects Receive Data Available on bit six would have J25 D6 jumpered to J26 DAV. Either polarity may be selected (+ for active high – for active low), depending on what the driver is expecting. Not all status signals need to be connected, only the ones that are needed.
28. 🞏 Insert board in bus and power on.
29. 🞏 Using a ‘scope or logic probe, examine U24 pin 19 (Data Available). Following a reset, this should be low. Examine U22 pin 22 (Transmitter Buffer Empty). The expected idle state of this pin is high.
30. 🞏 Using a program capable of entering and running simple machine code with breakpoints, such as SID/ZSID or some monitors, enter this simple (Z80) program into memory at 4000H (ignoring comments, and substituting the board’s UART status address for base+0, and data address for base+1):  
     4000: IN A,<base+0> ;Read idle state of UART status  
     4002: LD B,A ;Save in B  
     4003: LD A,42H ;Anything in A will do  
     4005: OUT <base+1>,A ;Send byte to UART; momentarily sets Transmit Buffer Empty false  
     4007: IN A,<base+0> ;Read status from UART  
     4009: LD C,A ;Save in C  
     ;Set a breakpoint here. UART will finish sending the byte in a split second  
     400A: IN A,<base+0> ;Read status again now that byte was sent and received via loopback  
     400C: LD D,A ;Save in D  
     400D: IN A,<base+1> ;Read data byte from UART – this resets Data Available  
     400F: LD H,A ;Save in H to compare with transmitted byte  
     4010: IN A,<base+0> ;Read status  
     4012: LD E,A ;Save in E  
     4013: NOP  
     ;Set a breakpoint here to halt execution  
    Run this program at 4000 with a breakpoint set at 400A. (Do not attempt to step this program.)  
    At 4002, the status idle state is stored in B. TBE should be true (ready to transmit) and DAV should be false (no data received.) At 4005, a byte is written to the transmitter, and then the status is sampled immediately following and stored in C. This should show the TBE bit false, as it is not ready for a new byte when it is still transmitting. Also at this point, the receiver has not received the full byte yet, so DAV should still be false.  
    The breakpoint before 4009 is needed to ensure there is enough time for the UART to transmit through the loopback to itself without writing a time delay loop. After the break, start execution again with a breakpoint set at 4013. Reading the status at 400A after the breakpoint is hit should read TBE true again and now DAV true with a byte ready to read. At 400D, reading the data byte automatically resets the DAV bit in the UART, so reading status again at 4010, after reading the data, should result in the idle state again: TBE true, and DAV false. The data read from the receiver is stored in the H register and can be compared to what was sent. The status bits are saved in B, C, D and E for comparison after the program is executed. The program must be executed full-speed to capture the status and cannot be stepped. Success at this step proves that minimal operation status reads are good.
31. 🞏 Power Off and remove the board. This completes the checkout of option C, Serial Port 2 Base.

**Option D1 or D3 - RS-232 Interface on Serial Port 1**This option requires option B to be built and tested first.

1. 🞏 To begin, one of the RS-232 Interface options must be selected, D1 (148x chips) or D3 (Max232). If D1 is selected, then options K and L are required. If options K and L are needed, complete those options first, described later in this document. Then return here to complete D1.
2. 🞏 Populate D1 or D3 components as desired, as per the BOM, including ICs.
3. 🞏 If D1, put jumpers on J7 pins 2 to 3 and J8 pins 2 to 3.  
   Else for D3, jumper J7 pins 1 to 2 and J8 pins 1 to 2.
4. 🞏 Temporarily short J5 pins 1 and 2. This creates a RS-232 level loopback. Anything transmitted by the UART will be received by the UART through the RS-232 interface.
5. 🞏 Insert board in bus and power on.
6. 🞏 Using a monitor or other program capable of reading and writing to I/O ports, read port base +1 and note the value. Write a different value to port base +1. Read port base +1 again and observe it has changed to the value written. Write another byte to base +1 and read base +1 to observe the new value. Try as many different bytes as needed to be sure the UART is transmitting and receiving as expected.
7. 🞏 At this point, all control, status and data logic is checked and functioning as individual components. Now all will be tested together. This will require some form of communications program on the computer. The program and the hardware must be matched: Data, status and control ports must be the same, status must be using the same bits and polarity. With everything setup, the program should be able to send and receive characters through the local loopback. If so, this test is successful.
8. 🞏 Power Off. Remove the temporary shorting jumper from J5.
9. 🞏 The next step will configure the serial port I/O pins on P2. Using flex jumpers, connect the required functions on J5 to the desired pins on J6. The J6 pins correspond exactly to the pins on P2. For example, if a standard IBM-style 10 pin header to DB-9 (or DB-25) cable assembly is plugged into P2, then for minimal communications, the functions needed will be DO (data going off the board) to pin 3, DI (data coming onto the board) to pin 2, and 0V (ground) going to pin 5.
10. 🞏 Plug in a known working serial device to the DB-9 (or DB-25) connector, such as a terminal. It must not require any particular handshaking signals (CTS, DTR, etc.) at this point. Set the baud rate on the device to the same as selected on JS16. Set the serial parameters to the same as programmed on J16 / J17. Most work with 8 data bits, 1 stop bit and no parity (parity even/odd is irrelevant.)
11. 🞏 Insert board in bus and power on.
12. 🞏 Using a monitor or other program capable of reading and writing to I/O ports, write an ASCII byte (e.g. 55H for ‘U’) to port base +1. This should transmit off the board and be received and displayed by the serial device. Send an ASCII character/byte from the serial device. Read port base +1. The value read should correspond to what was sent from the serial device. For example, suppose the character ‘A’ was sent. The hex equivalent, 41H, should be read from the port.
13. 🞏 Next, run the communications program used in step 7 above. Typing on the terminal should display in the communications program; typing in the communications program should display on the terminal. Note that typing on either device and having it appear on both is not typical and usually means the communications program or the terminal/serial device is configured for half-duplex operation. This is usually not desirable.
14. 🞏 This completes the checkout of Option D1 or D3, RS-232 Interface for Serial Port 1.

**Option D2 or D4 - RS-232 Interface on Serial Port 2**This requires option C to be built and tested first.  
If option D1 (148x chips) is selected for Serial Port 1, then option D2 must be selected for this option.  
If option D3 (Max232 chips) is selected for Serial Port 1, then option D4 must be selected for this option.  
Intermixing of 148x chips and Max232 chips is not possible.

1. 🞏 To begin, one of the RS-232 Interface options must be selected, D2 (148x chips) or D4 (Max232). If D2 is selected, then options K and L are required and were most likely already installed for option D1.
2. 🞏 Populate D2 or D4 components as desired, as per the BOM, including ICs.
3. 🞏 If D2, put jumpers on J9 pins 2 to 3 and J10 pins 2 to 3.  
   Else for D3, jumper J9 pins 1 to 2 and J10 pins 1 to 2.
4. 🞏 Temporarily short J12 pins 1 and 2. This creates a RS-232 level loopback. Anything transmitted by the UART will be received by the UART through the RS-232 interface.
5. 🞏 Insert board in bus and power on.
6. 🞏 Using a monitor or other program capable of reading and writing to I/O ports, read port base +1 and note the value. Write a different value to port base +1. Read port base +1 again and observe it has changed to the value written. Write another byte to base +1 and read base +1 to observe the new value. Try as many different bytes as needed to be sure the UART is transmitting and receiving as expected.
7. 🞏 At this point, all control, status and data logic is checked and functioning as individual components. Now all will be tested together. This will require some form of communications program on the computer. The program and the hardware must be matched: Data, status and control ports must be the same, status must be using the same bits and polarity. With everything setup, the program should be able to send and receive characters through the local loopback. If so, this test is successful.
8. 🞏 Power Off. Remove the temporary shorting jumper from J12.
9. 🞏 The next step will configure the serial port I/O pins on P3. Using flex jumpers, connect the required functions on J12 to the desired pins on J11. The J11 pins correspond exactly to the pins on P3. For example, if a standard IBM-style 10 pin header to DB-9 (or DB-25) cable assembly is plugged into P3, then for minimal communications, the functions needed will be DO (data going off the board) to pin 3, DI (data coming onto the board) to pin 2, and 0V (ground) going to pin 5.
10. 🞏 Plug in a known working serial device to the DB-9 (or DB-25) connector, such as a terminal. It must not require any particular handshaking signals (CTS, DTR, etc.) at this point. Set the baud rate on the device to the same as selected on JS17. Set the serial parameters to the same as programmed on J18 / J19. Most work with 8 data bits, 1 stop bit and no parity (parity even/odd is irrelevant.)
11. 🞏 Insert board in bus and power on.
12. 🞏 Using a monitor or other program capable of reading and writing to I/O ports, write an ASCII byte (e.g. 55H for ‘U’) to port base +1. This should transmit off the board and be received and displayed by the serial device. Send an ASCII character/byte from the serial device. Read port base +1. The value read should correspond to what was sent from the serial device. For example, suppose the character ‘A’ was sent. The hex equivalent, 41H, should be read from the port.
13. 🞏 Next, run the communications program used in step 7 above. Typing on the terminal should display in the communications program; typing in the communications program should display on the terminal. Note that typing on either device and having it appear on both is not typical and usually means the communications program or the terminal/serial device is configured for half-duplex operation. This is usually not desirable.
14. 🞏 This completes the checkout of Option D2 or D4, RS-232 Interface for Serial Port 2.

**Option E – Parallel Port and Random Number Generator Base**

1. 🞏 Install components for this option as per the BOM, including the ICs.
2. 🞏 Using jumpers, configure the base address on JS12 to be an unused address block in the computer. Note that this option uses four consecutive I/O ports. A jumper in sets that bit low.
3. 🞏 Insert board in bus and power on.
4. 🞏 Run a monitor or other program capable of reading and writing I/O ports.
5. 🞏 Observe the logic state of JS9, pin2 (Read Base+0). It should be high. Perform a port read from base+0 and observe a low pulse on JS9, Pin 2. Repeat this check for the other three active signals on JS9:
   1. Pin 5 – Read base+1
   2. Pin 8 – Read base+2
   3. Pin 11 – Read base+3
6. 🞏 Observe the logic state of JS10, pin 2 (Write Base+0). It should be high. Perform a port write to base+0 and observe a low pulse on JS10, pin 2. Repeat this check for the other three active signals on JS10:
   1. Pin 5 – Write base+1
   2. Pin 8 – Write base+2
   3. Pin 11 – Write base+3
7. 🞏 This completes the checkout of Option E.

**Option F – Parallel Port Interface**

This option requires option E to be built and tested first.

1. 🞏 The base address for the parallel port was established in option E. Now the data, control and status registers must be setup using JS9 and JS10. The settings will be determined by the driver in BIOS. An example should illuminate the process.
   1. Data write on port base+0, so connect JS10 pin 1 to pin 2.
   2. Control write on port base base+1, so connect JS10 pin 5 to 6.
   3. Status read on port base+1, so connect JS9 pin 4 to 5.

Your particular settings may differ from this example, and may alter the details of subsequent tests.

1. 🞏 Insert board in bus and power on.
2. 🞏 Run a monitor or other program capable of reading and writing I/O ports.
3. 🞏 With a logic analyzer, logic probe or oscilloscope, observe the parallel data outputs on P1, pins 3 (D0), 5 (D1), through 17 (D7) while writing bytes to port base+0. Observed data should match the data being written.
4. 🞏 Jumper J3, the Auto-strobe jumper. While observing P1 pin 1, output anything to port base+0. A low-level pulse should occur.
5. 🞏 With a logic analyzer, logic probe or oscilloscope, observe the levels on JS11 pins 1, 4, 7 and 10 while writing bytes to port base+1. The lowest 4 bits should appear on these pins. Also observe U18 pin 5. It should follow bit 4 of the byte written to port base+1.
6. 🞏 Temporarily connect J4 pin 2 to ground. Read port base+1 and observe bit 5 low. Then connect J4 pin 2 to +5. Read port base+1 and observe bit 5 high. If this one bit works as expected, it can be assumed the remaining will be okay. Remove the temporary connection.
7. 🞏 Configure the jumpers for your printer. Here is an example:
   1. Auto-strobe – Yes
   2. JS11, 5 to 6 (/AutoFeed = high or off)
   3. JS11, 8 to 9 (/Init = high or off)
   4. JS11, 11 to 10 (/SelectIn = low or on)
   5. J4, 1 to 2 (Busy on bit 5 – required by some BIOS drivers)
8. 🞏 Connect a “standard” IBM PC-style parallel port cable assembly to P1. Connect a parallel printer to the DB-25F on the other end.
9. 🞏 Turn on the printer.
10. 🞏 Locate a text file to be printed. Send it to the printer using PIP, like this…  
    A>PIP LPT:=TEST.TXT
11. 🞏 The file should print. This completes the checkout of Option F.

**Option G1 – Random Number Generator – One Noise Source**

This option requires option E and option K to be built and tested first.  
  
If this option is being built to satisfy the pre-requisite for option G2 (8 independent noise sources), then *do not install the discreet resistors as called out for RN1, RN2 and RN3*. Instead, install the resistor networks from the G3 option.

1. 🞏 Install components for this option as per the BOM, including the ICs.
2. 🞏 Place a jumper on JS7, pin 2 to 5. This connects the output of the noise source, buffered and conditioned by a Schmitt Trigger, to the input of the counter U6.
3. 🞏 Place a jumper on JS8, pin 4 to 5. This configures the counter as an 8-bit counter.
4. 🞏 Place a jumper on JS9 from one of the middle pins to one of the pins on the right labeled RD RNG. For this example, use pins 11 to 12 corresponding to base+3.
5. 🞏 Insert board in bus and power on.
6. 🞏 Run a monitor or other program capable of reading I/O ports.
7. 🞏 Read input port base+3 and observe a random number.
8. 🞏 Continue reading input port base+3 for as many random numbers as required to be sure it is working.
9. 🞏 This completes the checkout of option G1.

**Option G2 – Random Number Generator – Two Noise Sources**

This option requires option G1 to be built and tested first.

If this option is being built to satisfy the pre-requisite for option G2 (8 independent noise sources), then *do not substitute the discreet resistors as called out for RN1, RN2 and RN3*. Rather, install the resistor networks from the G3 option.

1. 🞏 Install all components for this option as per the BOM, including the IC’s.
2. 🞏 Remove JS8 pin 4 to 5 jumper if it is installed. Place a jumper on JS8, pin 2 to 7 (same pins as JS7) to connect the second noise source to the counter. The counter is now configured as two 4-bit counters clocked by two different noise sources.
3. 🞏 Insert board in bus and power on.
4. 🞏 Run a monitor or other program capable of reading I/O ports.
5. 🞏 Read input port base+3 and observe a random number.
6. 🞏 Continue reading input port base+3 for as many random numbers as required to be sure it is working.
7. 🞏 This completes the checkout of option G2.

**Option G3 – Random Number Generator – Eight Noise Sources**This option requires option G2 to be built and tested first.

1. 🞏 Install all components for this option as per the BOM, including the IC’s.
2. 🞏 REMOVE U6. The counter is not used when all 8 bits are from independent noise sources.
3. 🞏 Put jumpers on J1 and J2. These bypass the two bits that were clocking the counter to the random number generator input port.
4. 🞏 Remove any jumpers from JS7 and JS8.
5. 🞏 Put jumpers on the top two pins (1 and 6) of each of the eight RNG OPT mode select jumpers, JS1, JS2, JS3, JS4, JS5, JS6, JS7 and JS8. This selects a Schmitt Trigger buffered flip-flop for each random bit. After testing is complete, feel free to use any of the options: Flip-Flop, Schmitt Trigger only or Direct Noise Source. The bottom two pins of JS8, labeled “8bit Cntr” are not used for this option.
6. 🞏 Insert board in bus and power on.
7. 🞏 With a ‘scope or logic probe, confirm that there is random bit activity on pin 1 of each of the eight option selection jumpers, JS1 through JS8.
8. 🞏 Run a monitor or other program capable of reading I/O ports.
9. 🞏 Read input port base+3 and observe a random number.
10. 🞏 Continue reading input port base+3 for as many random numbers as required to be sure it is working.
11. 🞏 This completes the checkout of option G3.

**Option H – Interrupt Logic for Parallel Port and/or Either or both Serial Ports**This option requires option B, C or F to be built and tested first.

1. 🞏 Install all components for this option as per the BOM, including the IC’s.
2. 🞏 If only parallel port interrupts are being used, proceed with step 5.
3. 🞏 If serial interrupts are needed, complete option I (interrupts from either serial port), then return here to complete this option.
   1. 🞏 This test can use the same serial device connected to P2 as was used in the checkout of option D1/D3 above. Alternately, if no serial device is connected, serial output can be looped back to serial input by jumping J5 pin 1 to pin 2.
   2. 🞏 With a long flex jumper or clip lead, connect J21 pin2 to JS12 pin 9 to provide a pullup resistor for the open-collector output of U25.
   3. 🞏 Insert board in bus and power on.
   4. 🞏 With a ‘scope, logic analyzer or logic probe, examine J21 pin 2. At idle, it should be high.
   5. 🞏 Write a byte to the serial 1 port control register that enables serial receive interrupts. This will vary depending on how the control register was configured in step 22 of option B above. No change in J21 pin 2 should be observed.
   6. 🞏 From the connected serial device, send a single character to SERCON. Alternately, if loopback was selected, write a byte to the serial port 1 transmitter that will then be received.
   7. 🞏 At this point, J21 pin 2 should go low. If this pin had been connected to one of the bus interrupt pins on J22 via a flex jumper, and the CPU was running with interrupts enabled, an interrupt cycle would be started.
4. 🞏 If the parallel port option F was installed, the following steps can check the operation of the parallel port interrupt logic:
   1. 🞏 Insert jumper at J13 pin 2 to 3 to select positive polarity ACK.
   2. 🞏 Insert jumper at J14 pin 1 to 2 to select interrupt on any ACK.
   3. 🞏 With a long flex jumper or clip lead, connect J21 pin 5 to JS12 pin 9 to provide a pullup resistor for the open-collector output of U25.
   4. 🞏 With a long flex jumper or a clip lead, connect P1 pin 1 to P1 pin 19. This loops the strobe line back to the Acknowledge line to simulate a printer.
   5. 🞏 With a ‘scope, logic analyzer or logic probe, examine J21 pin 5. At idle, it should be high.
   6. 🞏 Write a 10H to the parallel port control register to enable parallel interrupts. No change in J21 pin 5 should be observed.
   7. 🞏 Write a byte of any value to the parallel port data register.
   8. 🞏 At this point, J21 pin 5 should pulse low. This will be a rather narrow pulse, so a logic probe with a pulse catcher/stretcher would be ideal to see it.
5. 🞏 This completes the checkout of option H, Interrupt Logic for Parallel port and/or either or both serial ports.

**Option I – Interrupt Logic, Either or Both Serial Ports**This option requires option B or C and option H to be built and tested first.

1. 🞏 Install all components for this option as per the BOM, including IC’s.
2. 🞏 Testing for this option is in the Option H procedure.
3. 🞏 This completes the assembly of option I.

**Option J1 – Positive 5 volt Power Supply for Boards running on a +5 volt power bus  
CAUTION! DO NOT USE THIS OPTION FOR SYSTEMS WITH A STANDARD 8 VOLT BUS!**

1. 🞏 Populate all components for this option as per the BOM.
2. 🞏 Put a jumper on J27.
3. 🞏 Insert board into bus and power on.
4. 🞏 With a DMM, measure about +5 volts on U25 pin 14.
5. 🞏 This completes the checkout of option J1.

**Option J2 – Positive 5 volt Linear Regulator Power Supply**

**Note: Be sure Option J1 is not populated and J27 is not jumpered if using this option**

1. 🞏 If option H is planned to be installed, insert C53 before the remaining option J2 components, as the part may be under the heatsink and difficult to install if the heat sink is installed first.
2. 🞏 Populate all components for this option as per the BOM.
3. 🞏 Insert board into bus and power on.
4. 🞏 With a DMM, measure about +5 volts on U25 pin 14.
5. 🞏 This completes the checkout of option J2.

**Option J3 – Positive 5 volt Switching Regulator Power Supply**

**Note: Be sure Option J1 is not populated and J27 is not jumpered if using this option**

1. 🞏 Populate all components for this option as per the BOM.
2. 🞏 Generally, the regulator should be physically stable without the hardware to hold it on the board. But if it is used, the screw should be inserted from the back of the board, with the components stacked as follows:
   1. Screw
   2. #2 washer
   3. SERCON board
   4. #2 x 0.1” spacer
   5. Pololu regulator
   6. #2 x 0.1” spacer (to prevent the nut from contacting or damaging components on the Pololu)
   7. #2 hex nut
3. 🞏 Insert board into bus and power on.
4. 🞏 With a DMM, measure about +5 volts on U25 pin 14.
5. 🞏 This completes the checkout of option J3.

**Option K – Positive 12 Volt Power Supply**

1. 🞏 Populate all components for this option as per the BOM.
2. 🞏 Insert board into bus and power on.
3. 🞏 With a DMM, measure about +12 volts on U12 pin 14.
4. 🞏 This completes the checkout of options K.

**Option L – Negative 12 Volt Power Supply**

1. 🞏 Populate all components for this option as per the BOM.
2. 🞏 Insert into bus and power on.
3. 🞏 With a DMM, measure about -12 volts on U12 pin 1.
4. 🞏 This completes the checkout of options L.

End of SERCON Description, Theory of Operation and Build Procedure.