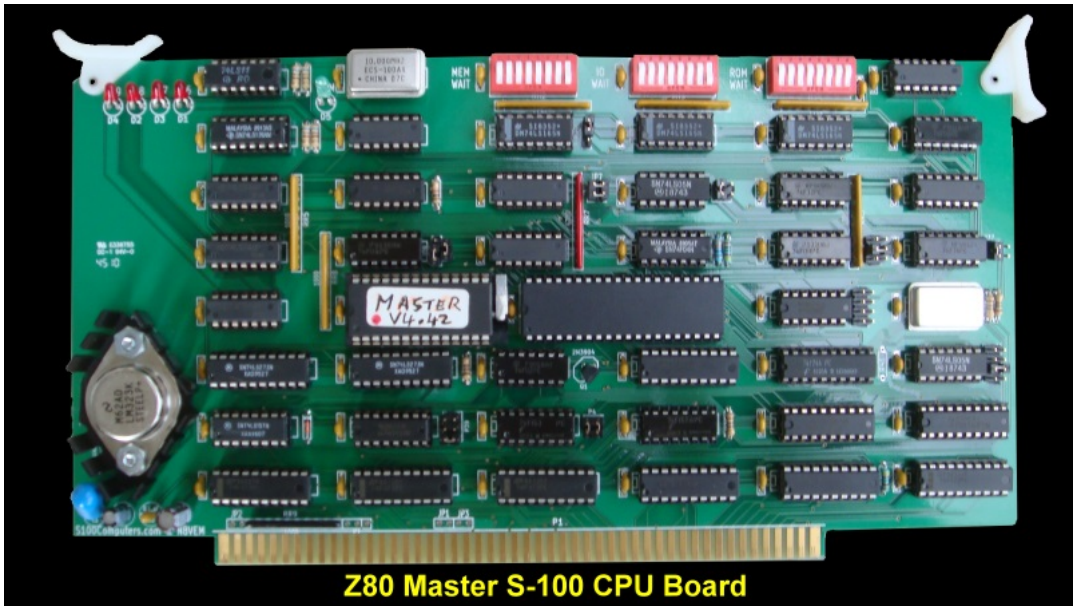
S100 Computers

A Web Site For S-100 Bus Computer Owners

An S-100 Z80 CPU Board.



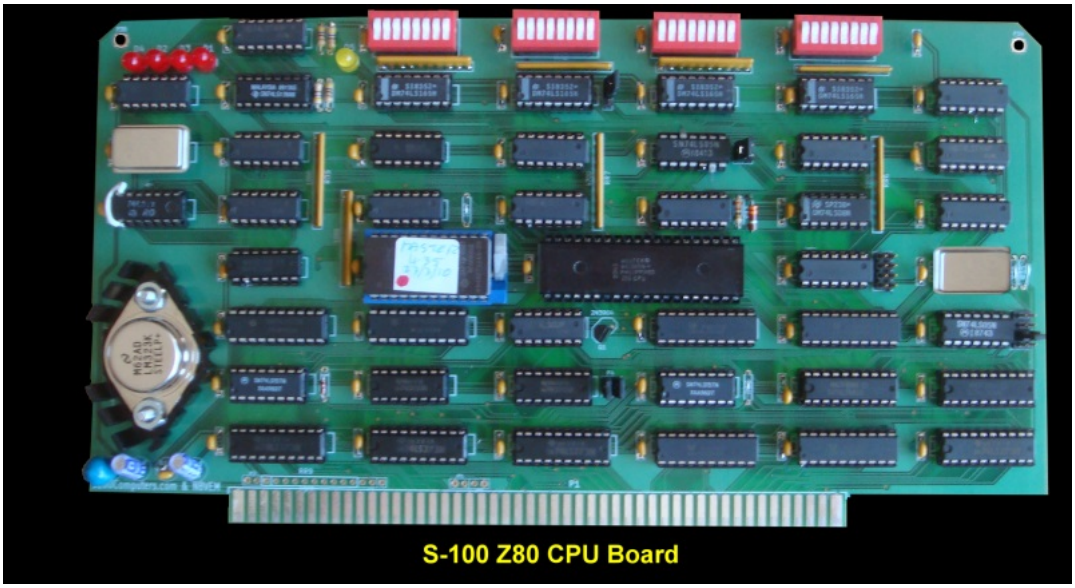
Introduction

Every serious S-100 board manufacture had their own CPU card. Many had a number of them. While almost everybody started off with Altair or IMSAI cards. These cards rapidly got replaced by faster better cards with more functionality on the card. The early TDL Z80 card was a great early improvement. In my early days I always liked the SD Systems Z80 Card. It was as solid as a rock. Easily going past the 4 MHz specs. Another rock solid CPU card was the Cromemco ZPU card. Soon people really appreciated having an onboard ROM containing basic code to input/output to ports and memory etc. Using a so called "Monitor" program.

As I have said and described elsewhere, I have over the years found the Intersystem's Z80-II to be the most reliable and flexible S-100 CPU card for the half dozen or so I have used over the years. This board should not be confused with the earlier Intersystem's Z80 (I) card. Which while fine, was nothing special. Apart from all the then common features found on a Z80 board (and being completely S-100 IEEE-696 compliant), it had an extremely clever and powerful ability to allow the Z80 to address up to 1 MG of RAM in 4K "windows" within the Z80's address space. The board could be modified to allow these windows to be enlarged into 16K "windows". This is described elsewhere and will be discussed in more detail below. But its primary importance is that it could be used to address greater than 64K of RAM for CPM3 and that it can be used to load/examine 8086 code at the top of the 1MG address space.

The reason for designing an improved version of this S-100 Z80 board was that hacking of the Intersystem's board to convert it to 16K "windows" is messy, complex and prone to errors. For others the board is difficult to find on places like eBay. I did not want/need the onboard AMD 9519 interrupt controller since I now have a more flexible fully dedicated S-100 Interrupt board utilizing the more common 8259A PIC. See here. Besides the AMD chip runs so hot you could fry eggs on it! Nor did I need all those low capacity EPROM options. A single 2732 EEPROM (flexible, but normally at address 0F000H-0FFFFH) would be a typical Z80 monitor requirement. I have streamlined and simplified the chip layout taking advantage of more recent/common TTL IC's. Lastly an extremely flexible set of options to add wait states for I/O, Memory reads, EPROM and INTA's was added to allow the use of extremely fast Z80 CPU's (see below).

The board is by far the most complicated board I have laid out to date. So a prototype was in order. I started with an almost exact image of the Intersystem's Z80-II board and then started to remove and add components. Using a step by step process and using a logic probe and data analyzer between the two boards I get a fully functional, fast and efficient board going with less than a weeks work. Here is a picture of that prototype board:-



The Z80 CPU Board Circuitry

The complete schematic of the prototype board can be seen here. At first it does look a bit scary, but once you split it into its components it gets a lot easier. First the board circuit can be split into its address/data line components and its S-100 status and control signals. Let us first look at the address lines.

Address lines.

Pushing the sixteen Z80 address lines out on to the S-100 bus on Z80 boards is usually not complicated. Typically it is done with a two buffer/driver IC's (that can be tri-stated by the S-100 ADSB* signal if the board is no longer the bus master). The reason this board is more complicated is that we have added the capability to intercept the address the Z80 "thinks" it is putting out with another actual address on the bus. The circuitry itself is a bit complicated, but for most it can be regarded as a "black box" that has the following properties:-

There are two address space windows 0-3FFFH and 4000H to 7FFFH that intercept the Z80 address lines and modify them depending on 8 bit "offset values' placed in ports at D2H and D3H on the board (U17 & U16).

The window "offset value" placed in port D2H (or D3H) is used as the top six bits of a 20-bit address space (allowing up to 1MG to be addressed).

Upon board reset, these "offsets" are 0H, so the address put out by the Z80 is the same as that which appears on the bus.

This address modification does not affect Z80 addresses above 7FFFH.

The two 16K windows are actually independent of each other and while often contiguous, can be different and used to move data from one memory location to another anywhere in the 1MG address space.

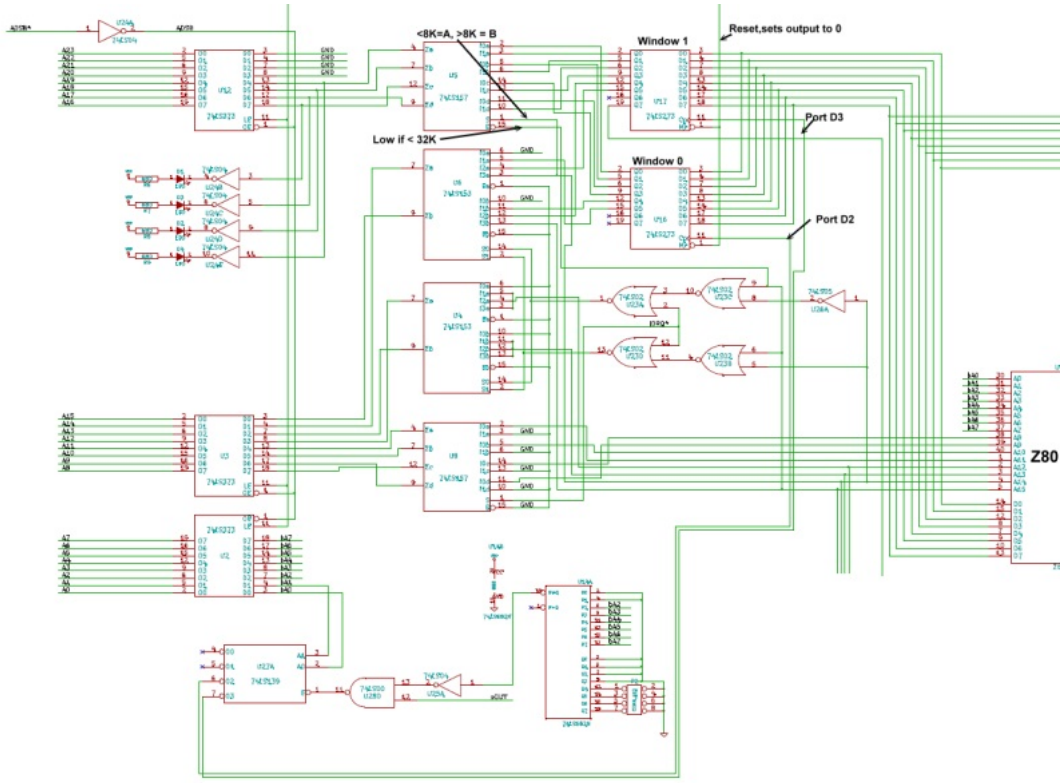
I know the above sounds confusing. Lets take an example. Suppose you want to see what is in RAM at FC000H to FFFFFH (a region addressed by say a 8086 CPU reset/monitor). If you output to port D2H the "offset value" 0FCH and then examine with your Z80 monitor program memory from 0H to 3FFFH what the Z80 will actually get is what is in RAM at FC000H to FFFFFH. If you have no RAM at that location in your system you will just see FFH's. Until you issue a new value to port D2H any Z80 code addressing memory from 0 to 3FFFH will actually be looking in hardware at RAM at FC000H to FFFFFH. If you placed 80H in port D2H you would be looking at a 16K window starting at 80000H in RAM. If you issue 0H to port D2H (or do a hardware reset) the Z80 address lines match with the hardware address lines (i.e. a 16K window starting at 0H in RAM -- or no actual translation).

The second port (D3H) works exactly the same except its window is 4000H to 7FFFH. So again if you issued 0FCH to port D3H When the Z80 accesses memory between 4000H to 7FFFH it will actually get in hardware memory at FC000H to FFFFFH.

Note the port address (D2H & D3H) are configured using jumper P2 with only 5-6 connected.

You can assign another port number if you like with jumpers on P2 (FxH -0xh).

Here is a picture of the address translation circuitry.

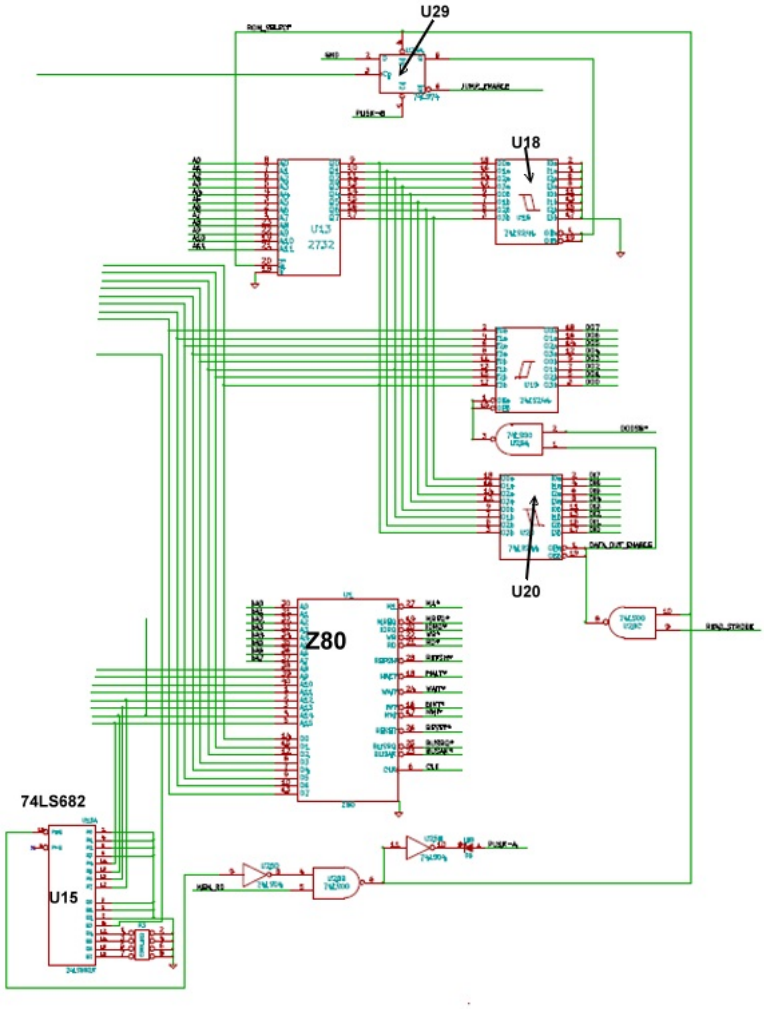


The relevance for all this becomes very apparent if you want to use CPM3 (or CPM+). See here for a more detailed description of that operating system.
By utilizing the above 16K windows you can very quickly and easily implement a banked CPM system.
With the CPM3 system I describe there, you will notice for example that the A16 address line D1 LED on the board will now light up -- indicating the Z80 is addressing RAM above 64K.

However we are getting ahead of ourselves. Remember for now, upon startup/reset, the Z80 sees only 64K of RAM just like any other S-100 CPU board.

The EPROM/Power On Jump Circuit.

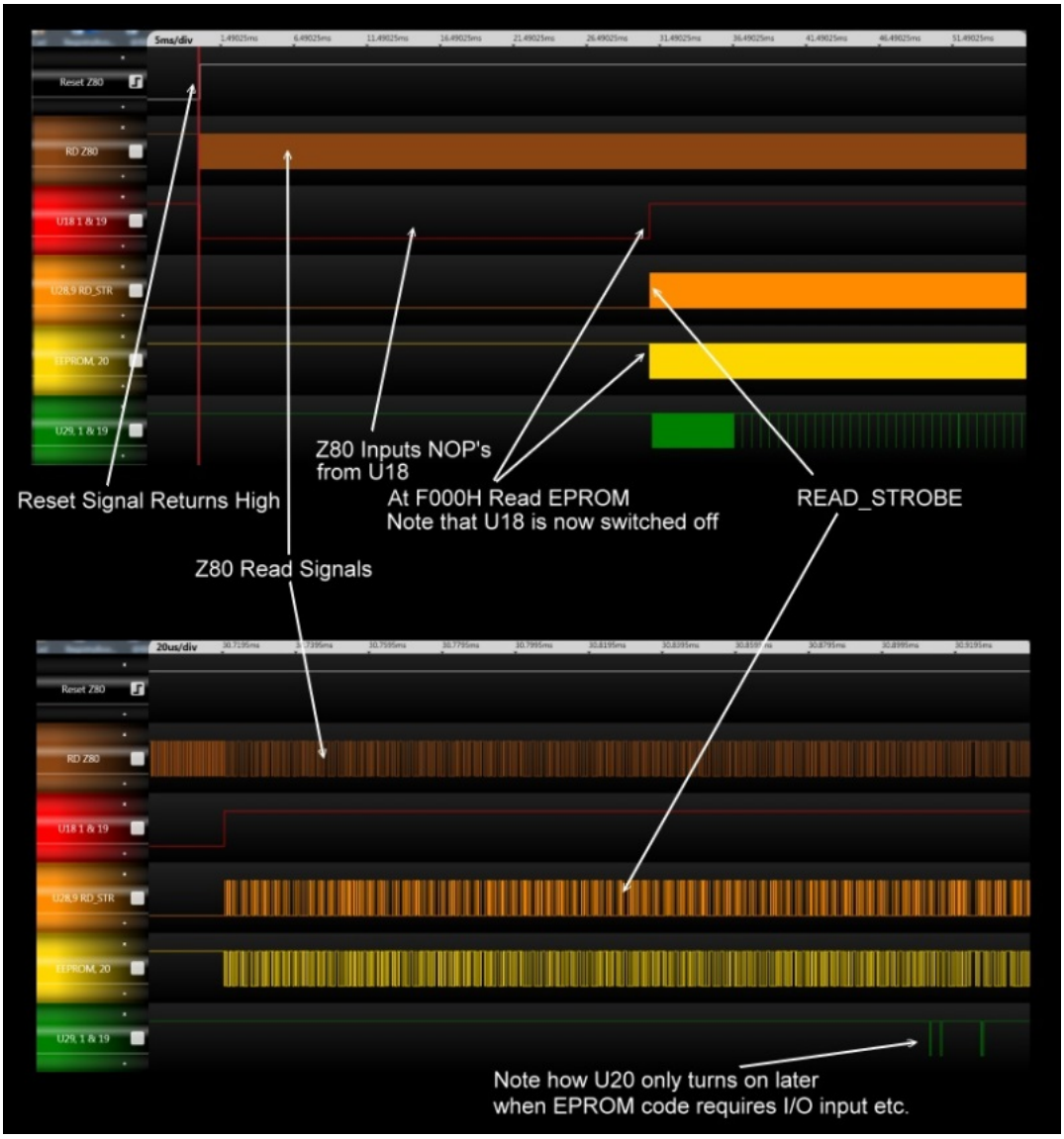
The Z80 boots up on reset at 0H in RAM. Typically we want it to jump to high RAM (e.g. E000H or F000H) where we have a Monitor program in EPROM. For this we need to force to start at that location upon power on or reset. The circuit below accomplishes this.



The trick is to force 00H (NOP's) on to the data bus until the desired Address is reached. The Boot (upper 4) address lines are set with a jumper (P3) of U15. In my case I use F000H. So upon reset, all address lines are 0H. The output from U29 will turn on the inputs from U18. These will set the data lines to 0H. The Z80 will this see only the NOP opcode on its data lines. This will increase the address lines one byte. The process will repeat until the U15 matches the address lines (A12-A15) with that on Jumper P3. At that point pin 19 of U15 will go low. ROM_SELECT* will go low. This causes U29 to switch off U18 and turn on the EPROM outputs.

Note, Pin 1 & 19 of U20 the "Normal" buffer to input data from the S-100 bus to the Z80 will be high all this time because the READ_STROBE input never goes high. It is inhibited by the "Jump Enable" output of U29 going (via pin #3 of U41) to the Boards main READ_STROBE/pDBIN signal. The latter is not shown in the above diagram but can be seen in the main schematic.

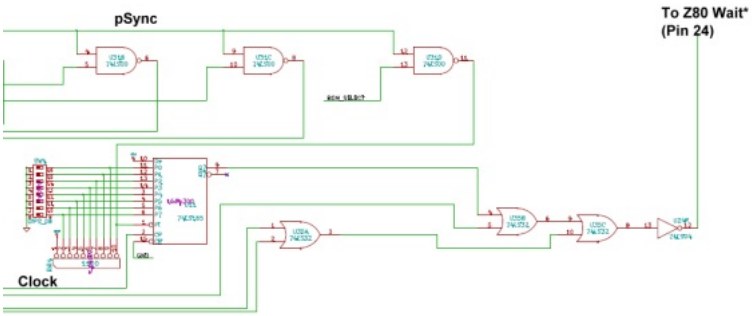
Here is a Logic Analyzer Display of some of the signals (using a USBee SX).



You can place the EPROM on any 4K boundary in the Z80's 64K address space. I use 0F000H to 0FFFFH for the following Monitor program. Note this prototype board has space for only a 2732A EPROM. The next version of the board will have options for EEPROMs as well.

Wait State Generator Circuits

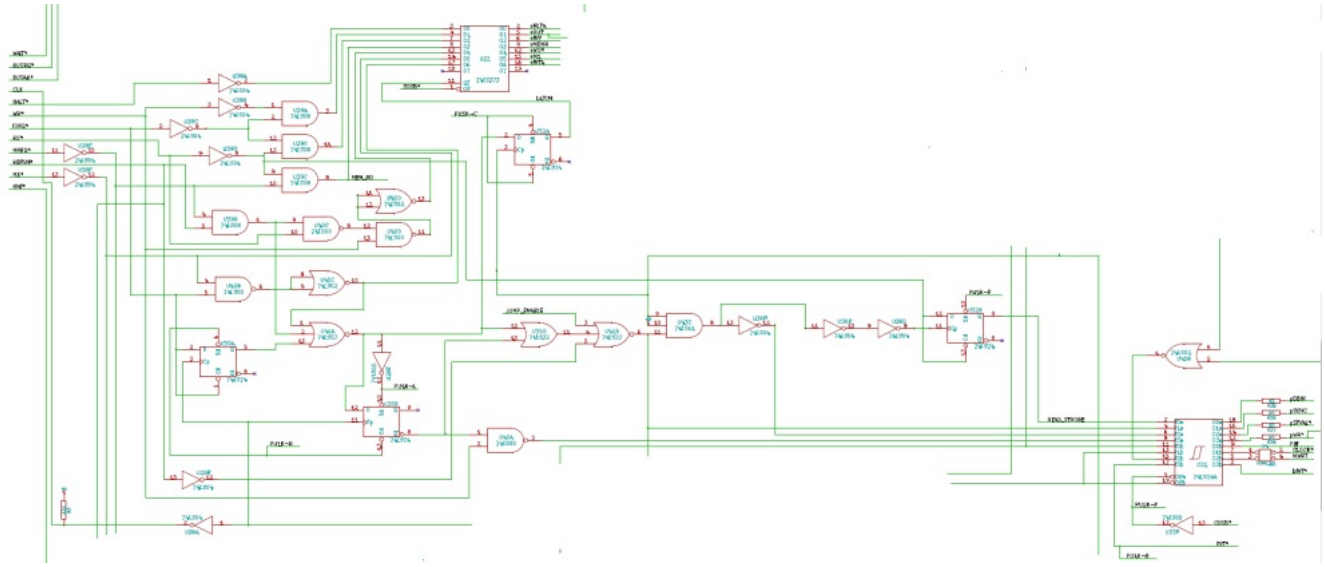
Since we will be pushing this board to speeds past 8MHz we will need to have the ability to add bus wait states for use with older slower S-100 boards. The usual tried and true 74LS165 circuit is used. Here is an example:-



By setting the switch anything from 0 to 8 wait states can be added to bus signals. On this prototype board I have 4 wait state generators to allow one to individually tune wait states for the onboard ROM, M1 type Memory accesses, any memory access and I/O port access. On the second addition of the board I have combined the M1 memory access and any memory access into one unit. For the Z80 an M1 type memory access has the shortest access time, so with a slow RAM board you can first try setting a M1 access wait state addition, if that does not work you can fall back to (slower) all memory accesses.

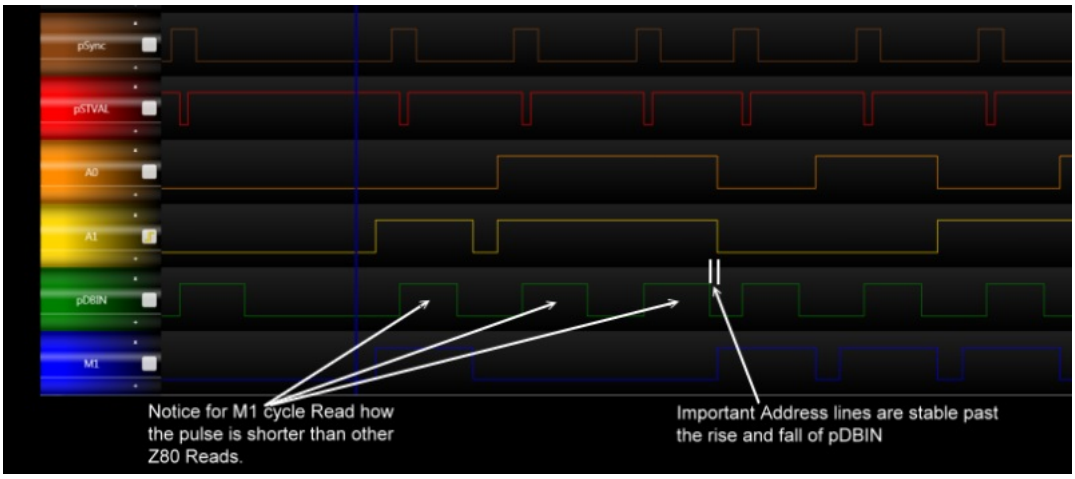
Status Signal Decoding.

The control signals from the Z80 cannot be directly used on the S-100 bus. They must be first converted into what is essentially Intel 8080 type signals. The circuitry for this is a bit complex and is shown here just for completeness. For detail, see the above complete schematic .pdf file.

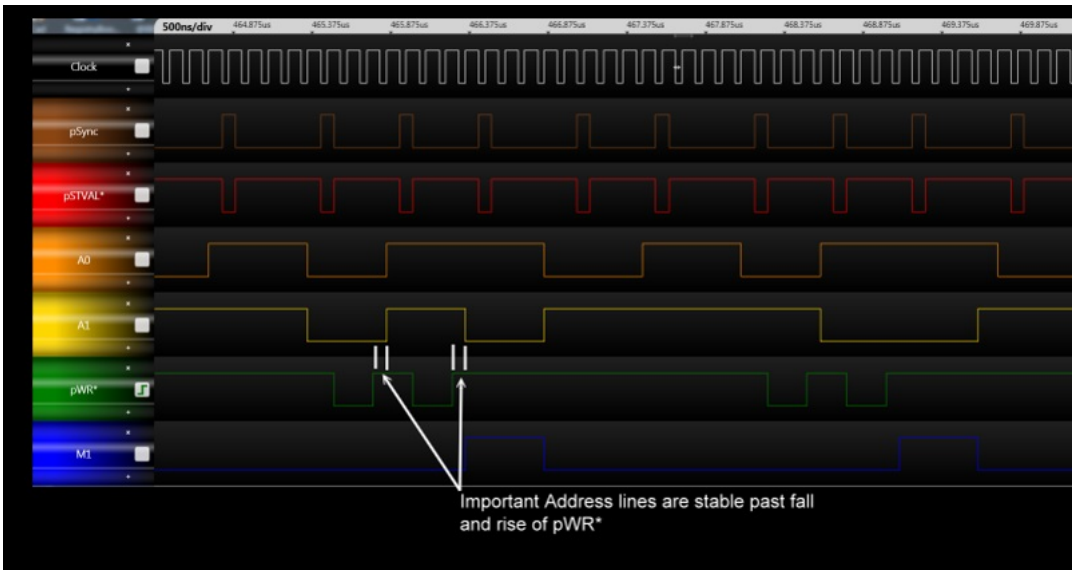


For those that may be interested here are the results of my Logic Analyzer S-100 signals when running the board in a 22 slot system (12slots occupied), active terminated, running the Z80 at 8MHz with one I/O wait state.

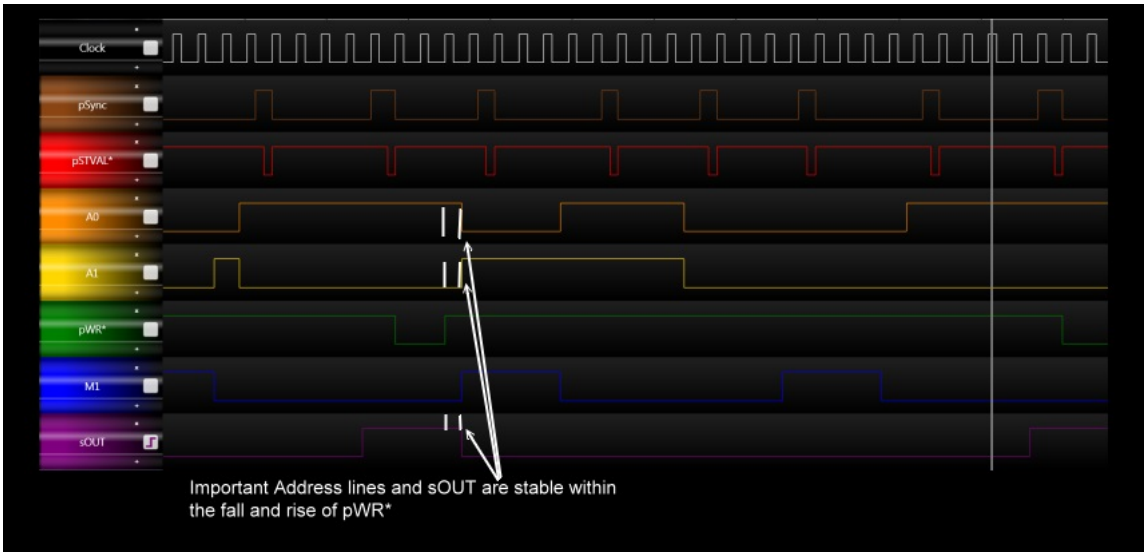
Here are the relevant signals for a memory read operation:-



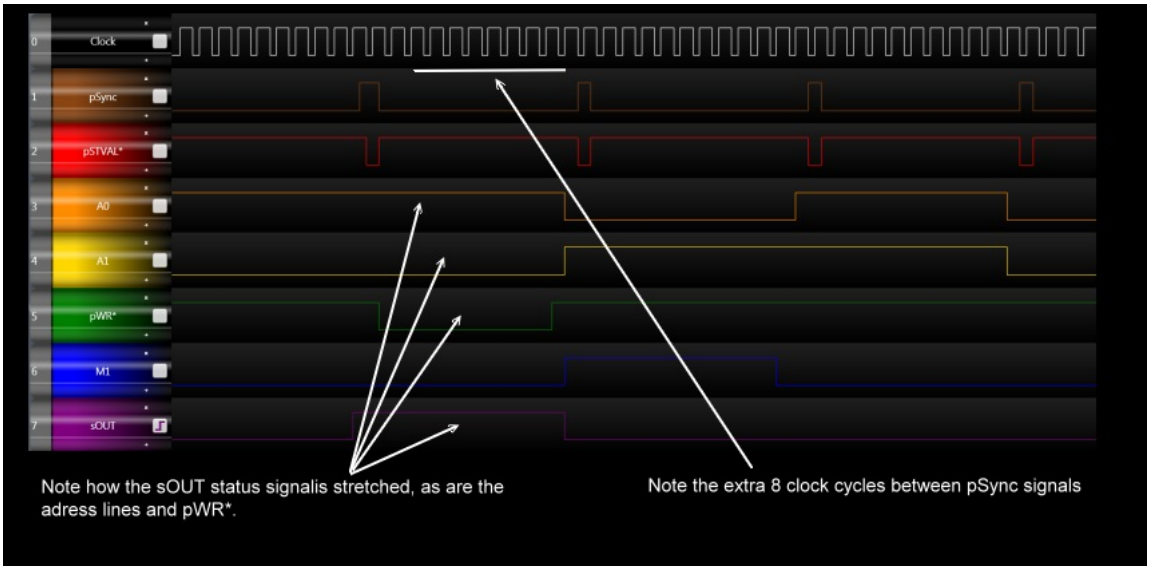
Here are the relevant signals for a memory write operation:-



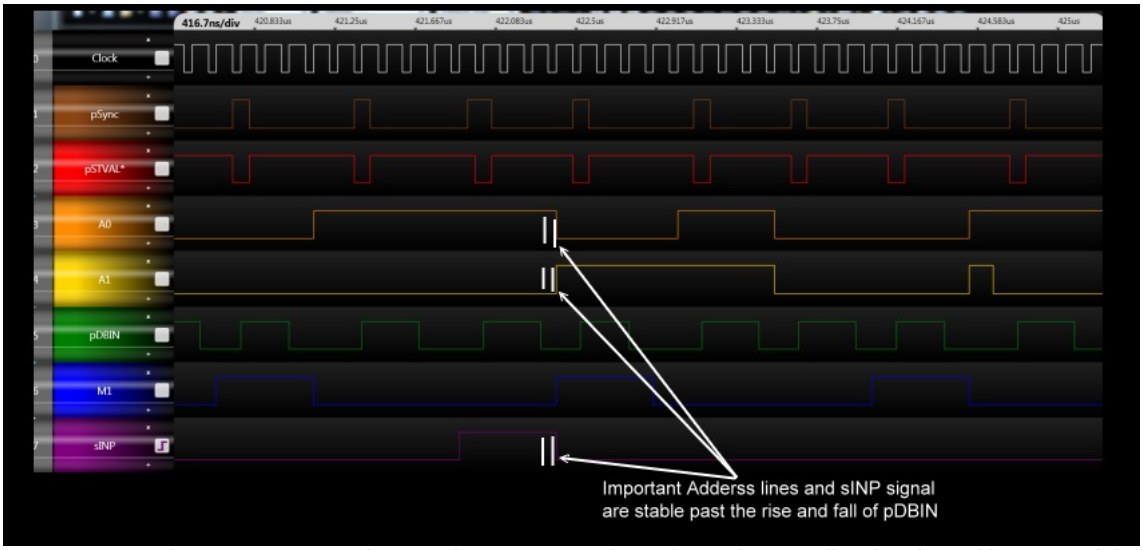
Here are the relevant signals for an I/O port write operation:-



Here is the same I/O port write operation with 8 wait states (SW3) added for all Z80 I/O operations:-



Here are the signals for an I/O port read operation:-



As you can see above in every case the critical Z80 pDBIN and pWR* signals are well within the stable region of the Address lines and the Bus status lines. This is very important. If the address lines or status lines are not at their correct levels at the termination of the pDBIN or pWR* signals, glitching will occur and the while system will be unstable. When, if, or should you play around with this board it is very important this requirement is met. The IEEE-696 standard defined the Address/Status line "overhang" to be 50ns for pDBIN and 0.2 of a clock cycle for pWR*. However this was for 4MHZ systems and old slow RAM chips. You can certainly push things with new static RAM boards like the 4MG Static board described here.

This Z80 board has a number of operating options which give the user some control over the processor-to-bus interface, while still corresponding to the IEEE bus specification. These options include processor speed selection by changing the oscillator. Values from 2 to 8MHz can be used.

In addition, a special feature has been included to allow the implementation of extremely high reliability systems. In order to meet the IEEE-696 standard for S-100 bus devices using a Z-80 processor, all bus address and status signals should be latched. This is because the Z-80 chip can

change its effective address at the end of an instruction fetch cycle while the bus read strobe is still valid. The IEEE-696 specification actually prohibits such timing ambiguities. To comply with this specification the board includes hardware latches on all address and status lines to the S-100 bus. There are two modes of operation of these latches: *Partial Latching* and *Full Latching*.

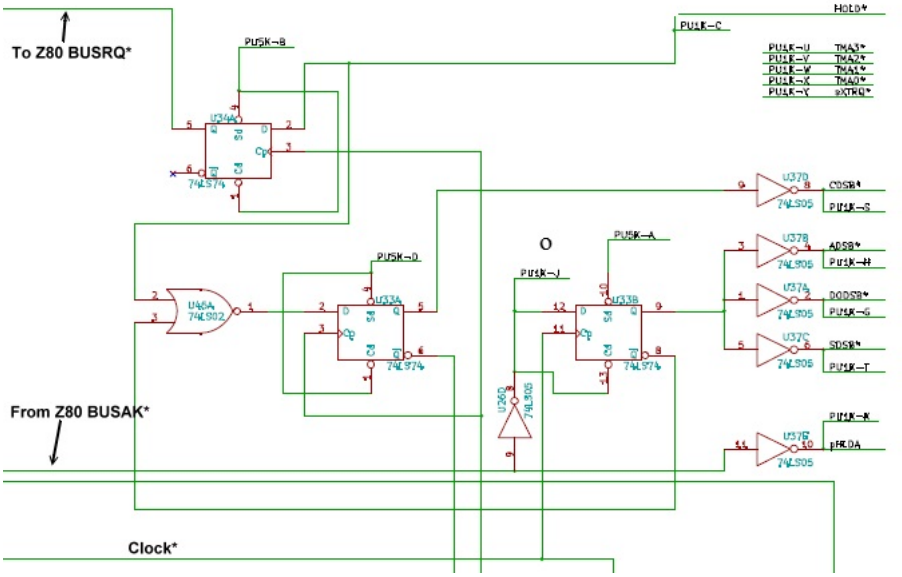
The Full Latch mode is designed for extremely high reliability systems, or operation in electrically noisy environments. The Full Latch mode halves the number of signal transitions on the bus and restricts all changes in address and status signals to specific, non-critical parts of the bus cycle, thus drastically reducing bus noise and signal crosstalk.

However, the *Full Latch* mode decreases the effective memory access time for the processor, requiring that either fast memory boards be used (For a 4MHZ system, 160 nsec. RAM worst case board access for M1 cycles would require a chip speed of approx. 125 nsec), or that a single wait state be added to all M1 cycles, slowing the processor approximately 10%. Other system parameters will usually reduce the wait state requirement to insignificance, but the occasional necessity for the faster M1 cycle can be satisfied with a *Partial Latch* mode. This mode does not control signal transitions as does the *Full Latch* mode, but provides for operation at 4 MHz without wait states if proper cycle triggering is employed. Clearly for our 4MG Static RAM board a partial latch mode is not an issue.

DMA Circuitry and Master/Slave Control Transfer Circuitry

The last major feature of this Z-80 board concerns the relationship of the processor to Direct Memory Access devices on the bus. This can be a classical DMA controller chip utilizing the bus or a situation where the Z80 relinquishes control of the bus to another CPU. The IEEE-696 goes into great detail as to how this is accomplished. It defines a special protocol for the transfer of bus control from a permanent bus master, in this case our Z-80, to a" temporary bus master such as a DMA device or another CPU. This protocol involves a specially timed and overlapped transfer of the various signal groups on the bus such that the DMA device and the CPU are both driving the most critical bus lines in given inactive states during the transfer operation.

BTW, the circuit which controls this timing could be included on either the "Master" CPU board or on the DMA/Slave CPU board. I have included on the this Z-80 board for two reasons: First, if it is included on the master CPU board, it need not be duplicated on every DMA/Slave board in the system, and second, since few older S-100 DMA boards included such a circuit, it is much easier to modify those old boards to meet the S-100 standard if this circuit is centralized on the master CPU board. The relevant part of the boards circuit to accomplish this is shown here:-

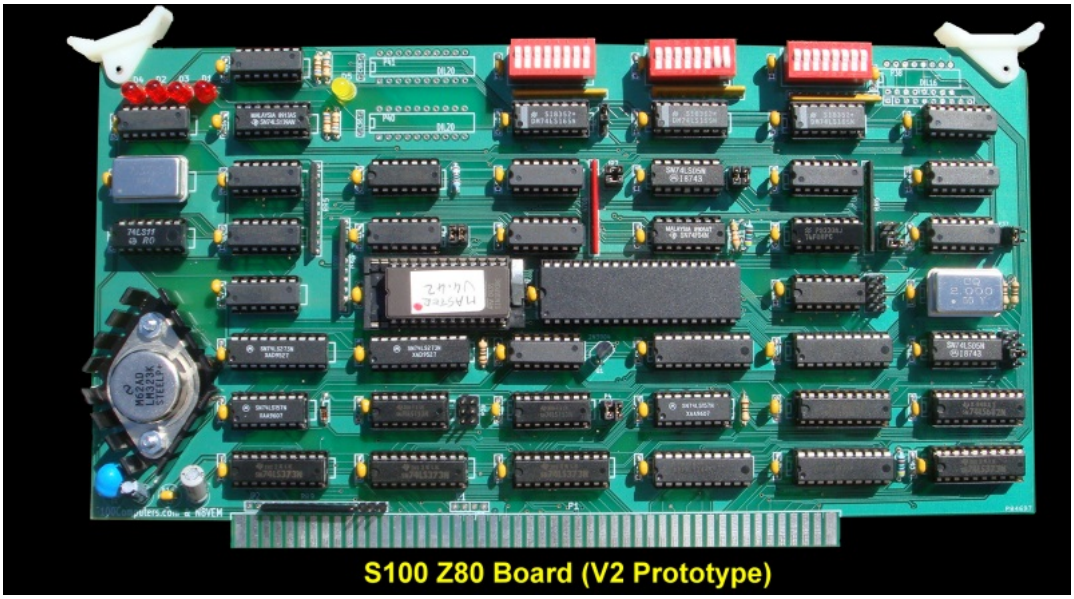


This Z-80 board, then, contains the above circuit which will conduct all the timing of the bus transfer, and tri-state its own bus drivers according to the IEEE specification (sections 2.8.2.1 through 2.8.2.4). The board will continuously sense the S-100 HOLD* line to see if a DMA controller or another CPU wants to control the bus. It sees HOLD* go low, it will acknowledge (when it is ready), by raising pHLDA and carry out an order transfer of bus transfer signals, (XFER I and XFER II *per* IEEE-696 protocol), on the ADSB, DOSB, SDSB, and CDSB lines respectively. Note it is up to the DMA controller/Slave CPU to determine if it has the highest bus priority if there are multiple (up to 16) units on the bus. This is done by reviewing the status of the 4 S-100 lines (DMA0-3).

The board also includes pull-up resistors on the four DMA arbitration lines, so that they need not appear elsewhere in the system. Also note the 10 ohm resistors in series with the control output lines, to prevent driver fatigue and glitches during the overlapped bus transfer.

A Second Prototype Board

Because this board will be used a lot I decided to do a second prototype to fine tune some of the boards features. This is essentially the same as the first prototype just some fine tuning of the circuitry was done.



One of the additions to the board is the ability to utilize EEPROMS such as the 28C64 EEPROM (as well as the old 2723A EPROMS).

I got the Z80 board to work reliably in both my systems at **10MHz**. For this I have 1 wait state on M1 memory reads, 1 wait state for all I/O cycles and 2 wait states for the on board 2732 EPROPM (or 1 wait state with a 28C64 EEPROM).
Note for these speeds one must use the more recent 10MHZ CMOS Z80 chips by Zilog.
These can be obtained from certain sources, for example Mouser (Part # 692-Z84C0010PEG).
Also remember for these speeds you will need an active terminated S-100 bus.

For this speed, many of the 74LSxx chips had to be upgraded to 74Fxx type chips. However it turns out some must not be upgraded. The delay coming into pin 11 of U32 is very critical, (see the schematic) . This is the central read strobe (pDBIN) and must not rise until everything else settles. Also U34 must be 74LS74

All 74xx chips are 74LS, except the following which are 74Fxx:-
U2,U3, U12, U8, U4, U23, U25, U38, U39, U40, U45, U30, U35, U22 and U21.

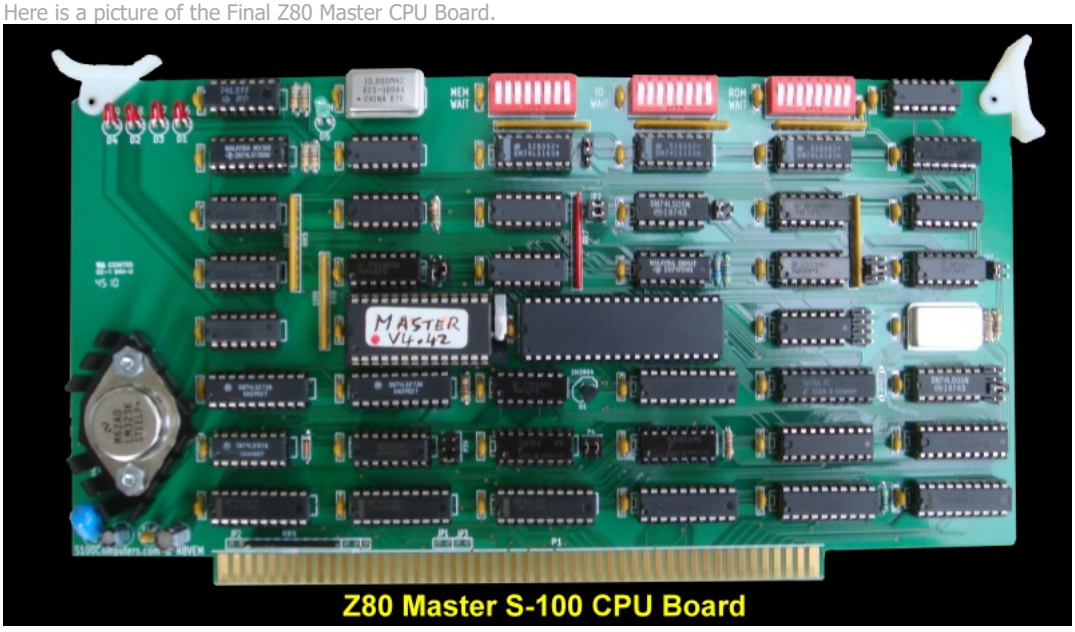
With "normal" 74LSxx chips the board appears to work fine at 8MHZ.

Final 10MHz Master Z80 CPU Board.

The final "production" version of this board is now completed and people have received boards. Some minor tweaks of the above prototype board were made. In particular we removed the IMSAI front panel connector (top right hand corner). This allowed considerable trace optimization. The IMSAI front panel would probably not work with this board at these speeds anyway, besides the SMB fulfills many of these functions. Here is a schematic of the third Z80 Prototype board. The board layout can be seen here.

The board in my bus actually *"works most of the time"* at 11 MHz However it's not completely reliable at these speeds. Some day I may try tweaking the above chips some more. At 10MHz it's completely reliable. It works fine too handing control over to an 8086 as a master/slave switch. It also appears to work fine with the interrupts from the PIC/RTC board.

Here is a picture of the Final Z80 Master CPU Board.



Note BTW the large 3A voltage regulator is close to the right hand side of the board. I found it necessary to clip off two of the "fins" so the card would easily slide into my card cage. I think the square type heat sinks would be OK.

Unlike many of the other boards on this site you cannot build and check out the board in stages. To start with however use a 2 or 4 MHz clock oscillator. If your board does not come up at these speeds, change your Monitor code in EPROM to one with an EPROM with just 76H's (HALT) and

see if you can get the Z80 to always halt after a reset. If it does not, check the POC circuitry and see above how the board steps through memory addresses to arrive at the boot EPROM address. If you are using our SMB none of the jumpers JP4-7 need be used. Of course you can disable them on the SMB and use them here if you like. Lastly don't expect this board to run at 10MHz speeds in a non-terminated S-100 bus or with old/slow S-100 memory boards.

Note if you decide to use a EEPROM such as a Samsung MK28C64A 8KX8, you have to place it at an 8 K boundary. So in high RAM this would be at E000H. If you want to use E000H-FFFFH for your monitor then that's fine. More typically however you will need only 4K for a monitor starting at say F000H to FFFFH. In this situation you can either program the second 4K of the EEPROM's 8K space and jumper JP8 1-2 and P39 5-6 (i.e. utilize A1 and A12) and set the POJ jumper P3, to F000H, or alternatively just program the lower 4K of the EEPROM's 8K space and force the EEPROMS A12 line to low, so jumper JP8 10-2 as before, but jumper P39 3-4. BTW, you can have a second completely different second monitor in the EEPROM switchable to the upper or lower half of the EEPROM with jumpers P39 3-4 or 1-2.

You must also be sure to "burn" your monitor into the upper or lower half of the EEPROM. (For the upper half, using a Wellon VP-280 Programmer, the "Load Buffer Address" would be 1000H, the "File Address" is F000H).

Note you can totally inactivate the ROM also. That is a very important function for CPM3. You do this by outputting xxxxxxx1 to port D3H. You can shadow it back in by xxxxxxx0 to port D3H. i.e. bit 0 of port D3H.

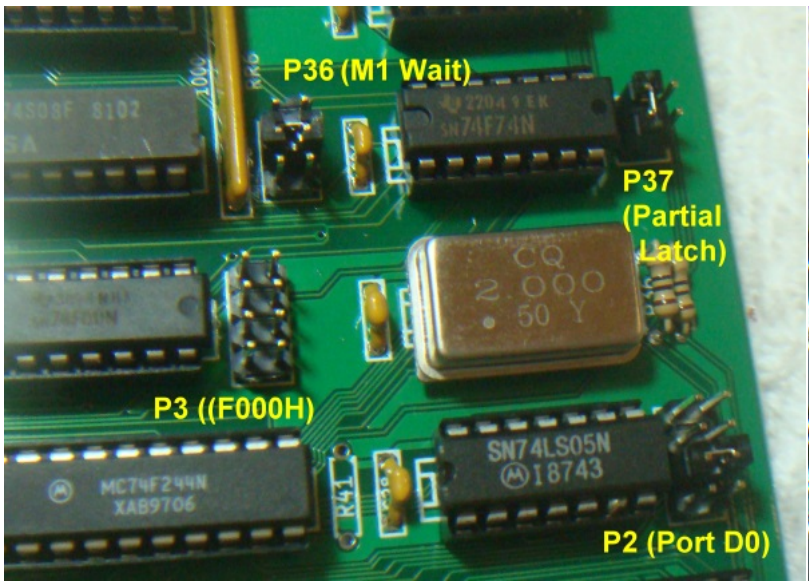
Be sure for testing not to blow yourself out of the water. Assemble the monitor at say 9000H, load the hex file there with ZSID, G9000H, Display ROM at say F000-F100. Then QOD3,01. Then display/change RAM at F000H.

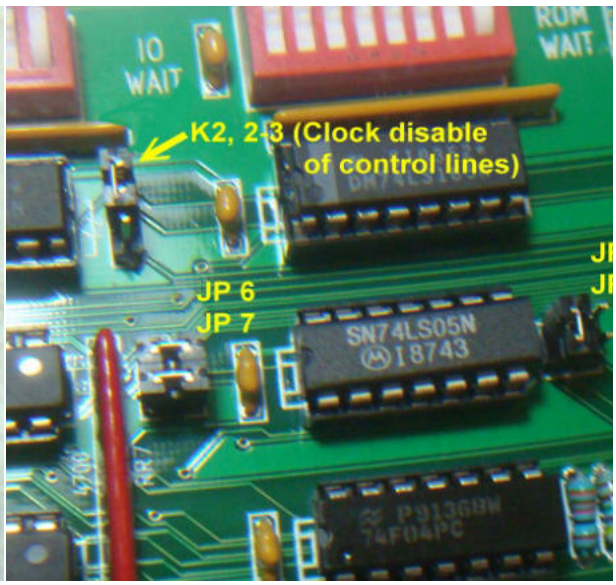
This is documented in the software section for bringing up CPM3. See for example the file ZLDRBIOS.ASM here.
Open the "MOST CURRENT FLOPPY ZSYSGEN SOFTWARE" .ZIP file

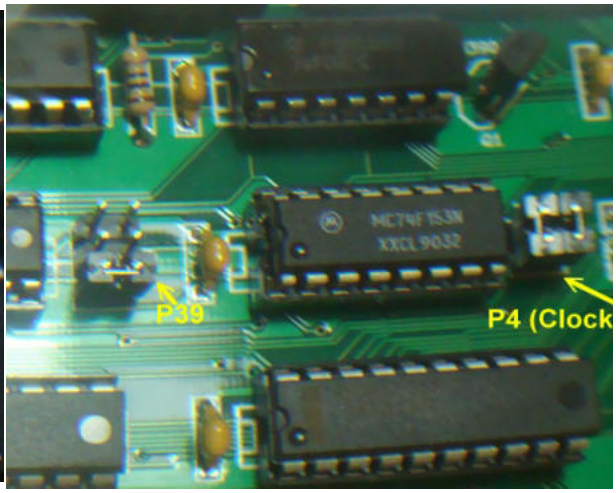
The ROM switching takes advantage of the fact that the above 16K window mapping frees up bits 0 and 1 of ports D2H and D3H (or whatever address you have yours jumpered for). In the schematic see pin 9 of u15 is connected to pin 19 of U17. This allows you to "inactivate" pin 19 of U14.

Hopefully all this is not too confusing. See here if you need more help. If you still are having problems, start with an old 2732 EPROM and burn your initial monitor in that chip.

For a 10 MHz system with the 74Fxx or 74Sxx chips I mentioned above, here are pictures of the jumpers to get you going.







A Description of the Board Jumpers.

The board contains a number of important jumpers that determine how it functions. Most will not need to be changed once the system is running but it is very important they are configured correctly. In no particular order:-

Jumper	Function
JP5,JP6,JP4,JP7	Used only if the board is to act as a bus master or are not generated by a front panel board. Generates Power On, Reset etc.
JP4, 1-2	Use only if no other board generates the S-100 2MHz clock signal when the Z80 is active
JP4, 3-4	Use only if no other board generates the S-100 MWRT signal when the Z80 is active
SW4 (ROM WAIT)	Sets number of wait states for onboard EEPROM (0-8). I use 1 wait state, so switch 8 (right most switch) is closed, the rest are open
P36	Allows wait states. 1-2, every sINTA, 3-4, M1 memory bus cycles, 5-6 All Memory cycles. I use 3-4.
SW2 (ME WAIT)	Sets number of wait states for P36 options. I use 1 wait state for M1 cycles only, so switch 8 (right most switch) is closed, the rest are open
SW3 (I/O WAIT)	Sets number of wait states for port I/O cycles. I use 2 wait states, so switch 7 & 8 (right most switches) closed, the rest are open
K2	Normally set 2-3, however older pre-IEEE 696 boards (for example the Cromemco dazzler board) often require 1-2.
P37	Normally 1-2 (Partial latch mode).
K1, 2-3	If NMI software is not implemented do not connect
P39, J88, JP9	These jumpers are to configure different EPROMS and EEPROMS. (For a 28C64:- P39 5-6, JP8 1-2, JP9 closed).
P2, 5-6	Memory window configuration port. I use I/O port D0H
P3.	No jumpers. This sets the PROM boot address to F000H.
JP1, JP2, JP3	This provides extra ground lines on board IF ALL boards meet IEEE-696 specs. Normally unconnected.

An Updated V2 Version of the Z80 CPU S-100 Board.

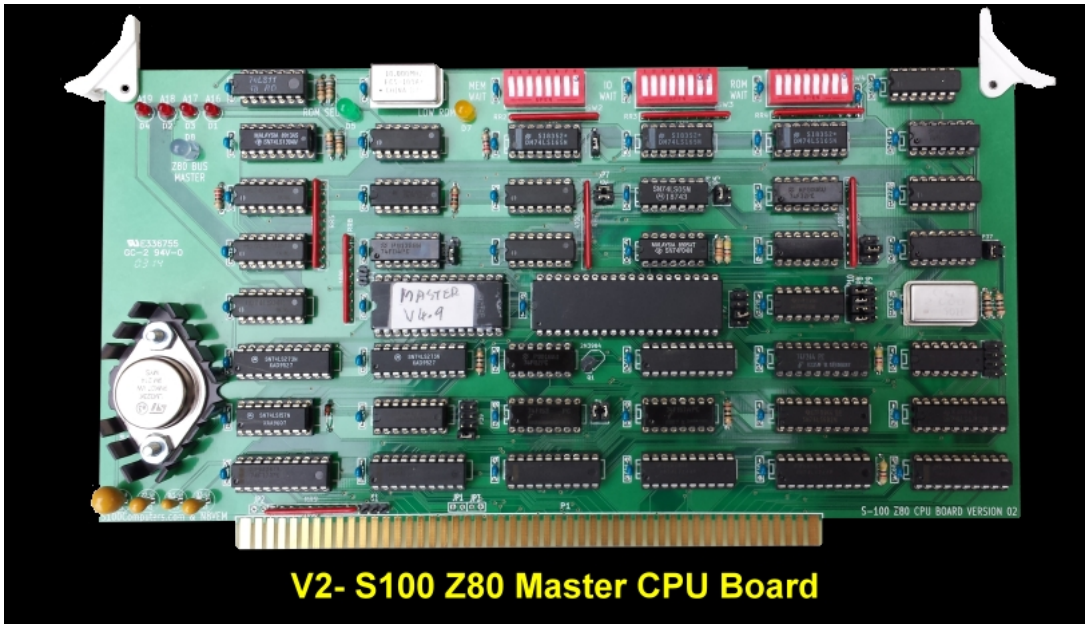
The above board has proved to be extremely popular and reliable. Two batches of 20 were already made and distributed to users. I decided to add a few small changes to the board to improve it's functionality. We are calling this board the **V2-Z80 CPU Board**.

The changes/additions are:-

1. Allow the user to partition an onboard 28C64 EEPROM into an upper and lower 4K window so that in effect an 8K monitor can be utilized, yet taking up only 4K of the CPU's address space.
2. Add an LED to indicate whether the Z80 is addressing the upper or lower 4K portion of the EEPROM
3. Add jumpers so older 2732, 27C64 or 28C64 etc. PROMS can still be used
4. Add an LED (blue) to indicate when the Z80 has control of the S-100 Bus in a master/slave arrangement with other CPU's.

In order to utilize the two 4K pages of the 28C64 EEPROM the Z80 boot monitor software needs to be modified (see below). However in all cases the default board configuration allows old EPROMS and their software to be used unchanged.

Here is a picture of the assembled V2 Z80 CPU board:-



For people that currently have our previous boards you will find that there are essentially no changes to this boards chip components. It should be possible to quickly do a "board components transplant" with no problems. The tried and true circuitry (described above) is unchanged.

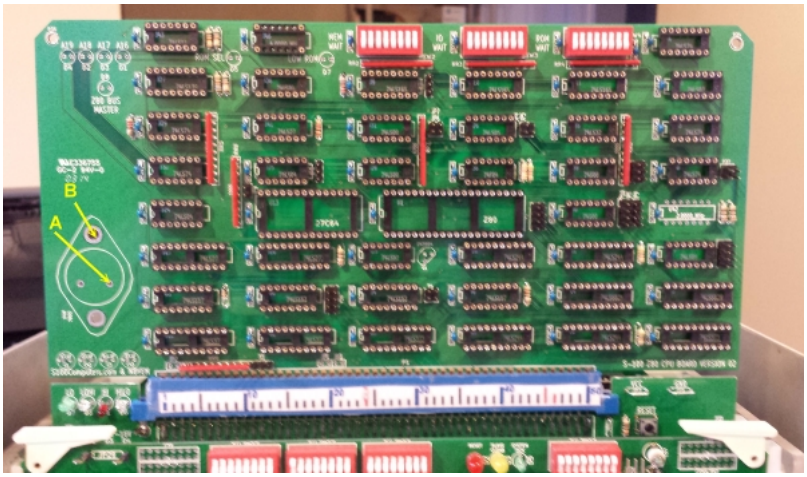
Step By Step Building the V2-Z80 CPU Board.

The first step is to examine the bare board carefully for scratches or damaged traces. Use a magnifying glass if need be. The quality of the boards we get is excellent. I must have done 30 by now, never had a problem, but there is always a first time. A broken trace is almost impossible to detect by eye on a completed board.

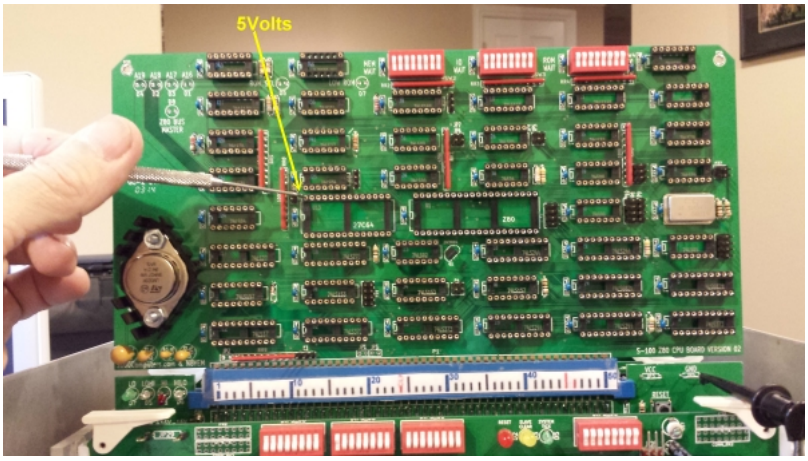
Next solder in all the required IC sockets, resistors, resistor arrays, capacitors, jumpers, and switched SW2-4. Do not add the LED's or LED bar yet. Be sure you put the resistor arrays in with the correct orientation of pin 1. Check their values before soldering (they are difficult to remove).

You can use cheaper "double swipe" IC sockets. However for a critical board like this I prefer to use "Machine Tooled" IC sockets (e.g. Jameco # 38623). Unfortunately they are more expensive and you have to be particularly careful not to bend the IC pins. If you think you will be doing a lot of EEPROM burning you should use the Low Profile ZIF sockets (e.g. Jameco #102745) for the EEPROM socket. However be aware that long term these do not make as good a connection as the above Machine Tooled IC sockets. The two clock oscillators should have their own special sockets (e.g. Jameco #133006).

Its good to check there are no board shorts at this stage. With the switches SW2-4 in the open position you should have an almost infinite resistance between points A & B on the board at this stage.



Next add the Voltage regulator and Diode D6. Check the voltage to sockets on the board is above 5V by placing the board in your S-100 system using an extender board. With no load you will typically get 5.00V (+/- 0.25V). BTW, your system should boot and run correctly with its own Z80 CPU if you have one at this stage. If not, you have a serious solder bridge somewhere on the board. Before you do anything else with a magnifying glass go over every socket on the board and examine for a proper solder joint. I like to "reheat" each joint just to be on the safe side. The silk screen/varnish on these boards us quite thick. It's easy not to have a good solder joint for the ground pins. Double check. Extra time here will save you hours later!



Note the Z80 chip itself has pin 11 as its +5 input.

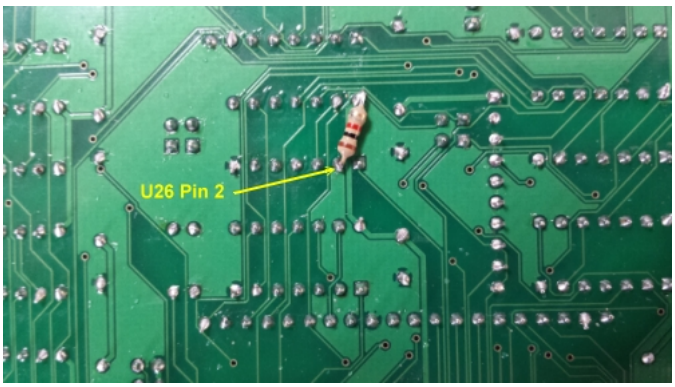
Next we will add the 7 LED's. Insert each LED into the board (usually the longer lead into the square pad) but do not solder yet. Insert the board into your S-100 system and with a probe tied to +5V or ground touch appropriate signaling pin socket that controls that LED. For example pin 9 of U38 high should light up LED D8. BTW, the color of the LED's is up to you. D8 is the Z80 Bus master active LED. I like to have all boards that are "active" light with a blue LED. The four address lines are red, ROM active LED green, Low ROM page active, Yellow.

Unlike many of the other boards on this web site it's not really practical to add chips one by one to this board with functional circuit testing. So you kind of have to take the plunge and add everything. Start with a 4MHz Oscillator in U46. Indeed at these speeds you can use 74LSxx chips throughout. If you have an earlier version of the board you are in great shape. Setup the jumpers as they were on that board. For U13 (EEPROM) set P39 (A12) high or low as you had it before. The only other new area is the P8,P9,P10 jumpers. We will normally want the monitor jump location to be F000H, so Jumper P8 1,2,3 & 4 to P10 1,2,3 & 4. No jumpers on P3. At 4 MHz wait state switches are normally not important. Before you pop the into your system check the orientation and number of each IC on the board. Please do this, I don't know haw many times I have ignored this and created un-necessary problems for myself. Remove your current CPU board (if present) and pop it into your system.

If you are lucky the board should come right up. The (blue) D8, LED should light up as well as the ROM, D5, LED. If not, don't worry often there is a trivial error. First check all jumpers one more time. If you reach a road block replace the onboard monitor with one containing only 76's (HLT) and see if the CPU goes into the halt state (S-100 bus pin 48 high). If OK, you can step things along with EPROMS of increasing complexity to identify the problem. See here. If you have our SMB things are easier. If after an exhaustive study you cannot get it to work let me know I may be able to work on your board but I honestly don't want to get into the repair business! With care and prober soldering etc. You really should have no problems, indeed the issue may be another board in your system.

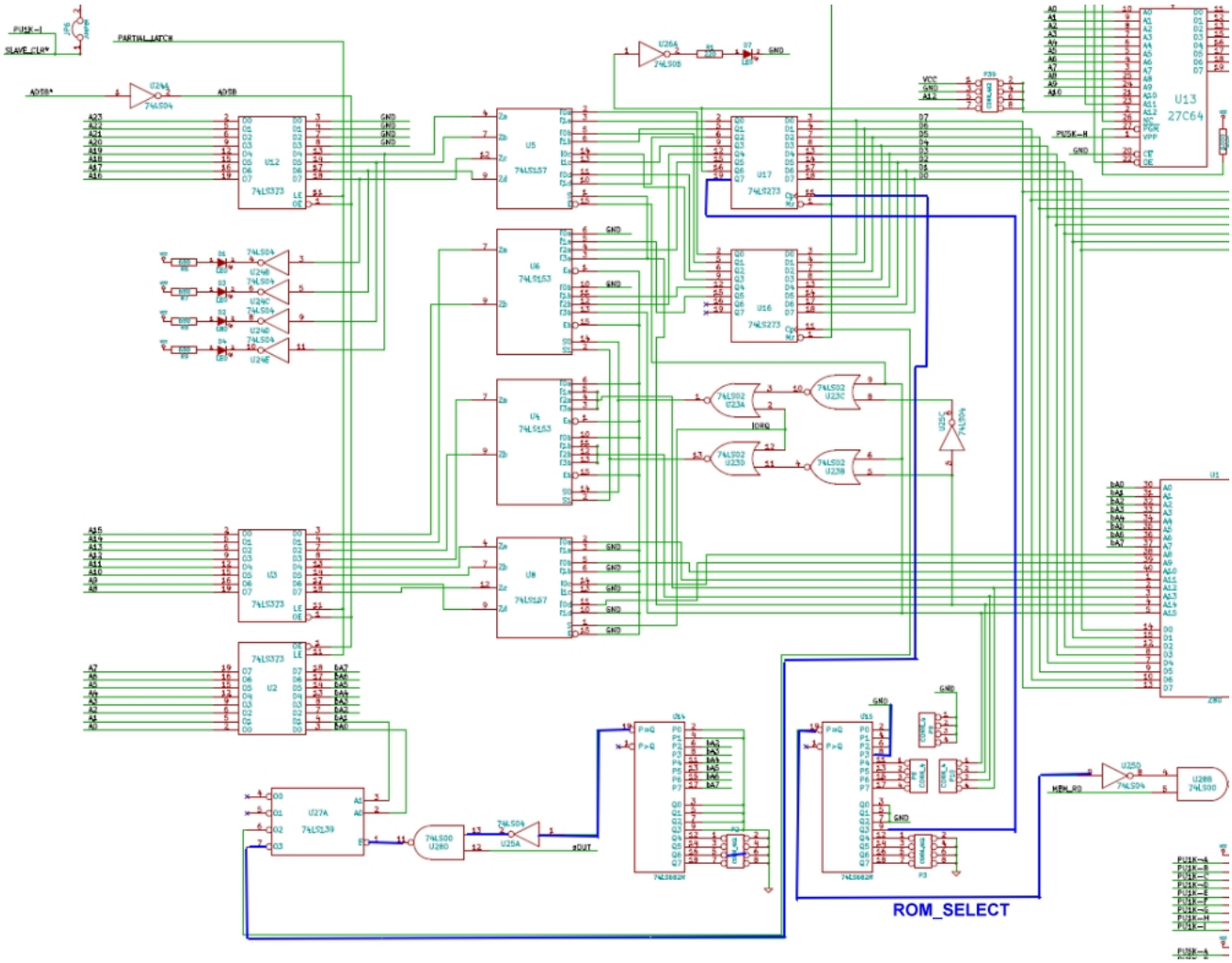
Bugs.

I have found only one minor bug with this board so far. The low ROM select gate U26 pin 2 for LED D7 needs to have a 1K pull-up resistor tied to +5V.
This can be done on the back of the board as shown here:-



Inactivating The Onboard ROM Circuit.

If you remember from above, the board has an output port consisting of U14 and U27A (which in my configuration P2 jumpered 5-6) corresponds to ports D2H and D3H.
If you output to D3H bit 0 high (i.e. QO D3,1) you will inactivate the onboard EEPROM. However you cannot do it within the monitor code itself (since it will blow itself out of the water). If you for example assemble the monitor at say 100H, boot it with CPM and QO D3,1 You will see the is no longer the ROM at F000-FFFFH.
This ability to remove the ROM is mainly used for CPM3 where one can use multiple banks of 64K each. Lets look at the circuit in a little more detail:-



U27 pin 7 clocks data into U17. U17 pin 19's output goes to the U15 comparator. If its pin 9 is high it will not match with pin 8 and so pin 19 will be high leading to inactivation of ROM_SELECT. This totally inactivates the onboard ROM circuit. The underlying RAM on the bus now appears. (Actually it was always there, it's just the Z80 never got to see it with Memory Reads. But Memory Writes are active.

If for example after reset you type:-

MF000,FFFF,F000
QOD3,1

You will actually have a functional monitor in RAM.
This is because the ROM and RAM at that location are now the same.
This can be seen with the memory map "A" command and "D" commands.
As an aside BTW, this allows you to single step through the ROM monitor code with our SMB.

Running the Board at 10MHz.

As with the above V1 board, this V2 Z80 board to works reliably in both my systems at **10MHz**.
For this I have 1 wait state on M1 memory reads, 1 wait state for all I/O cycles and 2 wait states for the on board 2732 EPROPM (or 1 wait state with a 28C64 EEPROM).
Note for these speeds one must use the more recent 10MHz CMOS Z80 chips by Zilog.
These can be obtained from certain sources, for example Mouser (Part # 692-Z84C0010PEG).
Also remember for these speeds you will need an active terminated S-100 bus.

For this speed, many of the 74LSxx chips have to be upgraded to 74Fxx type chips. However it turns out some must not be upgraded. The delay coming into pin 11 of U32 is very critical, (see the schematic) . This is the central read strobe (pDBIN) and must not rise until everything else settles. Also U34 must be a 74LS74.

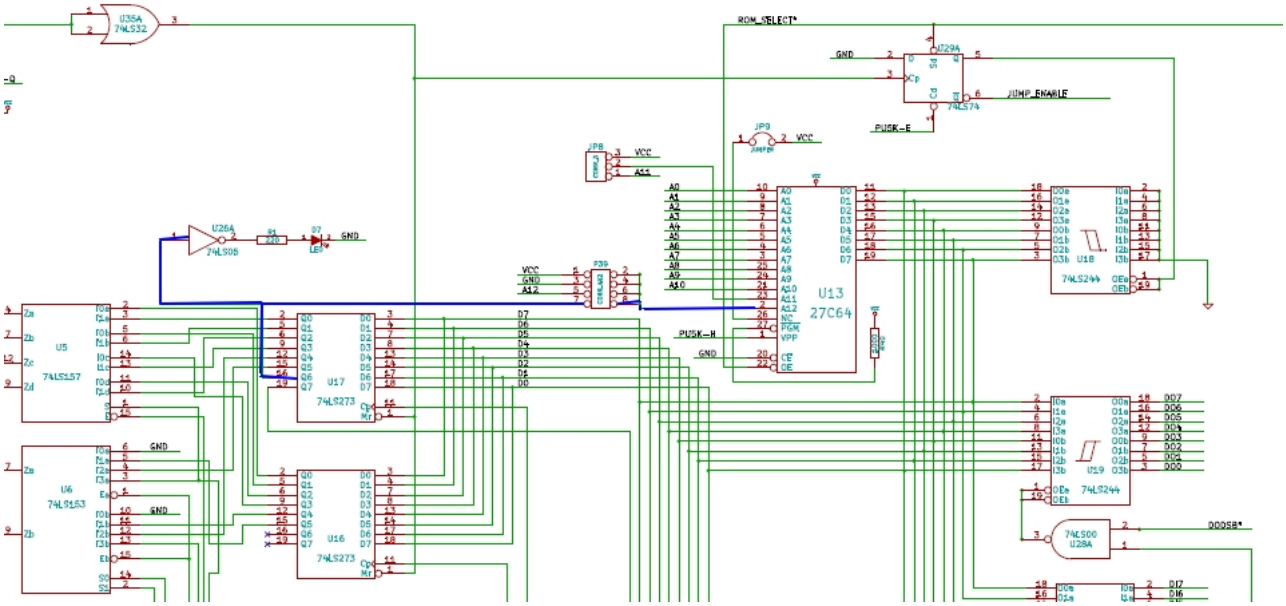
All 74xx chips are 74LS, except the following which are 74Fxx:-
U2,U3, U12, U8, U4, U23, U25, U38, U39, U40, U45, U30, U35, U22 and U21.

With "normal" 74LSxx chips the board appears to work fine at 8MHZ.

Activating The Onboard ROM Low/High 4K Page Code.

Now we get to the new addition of the V2 board. Everything described above for Port D3H works for the original board and this new V2 board. On the original board you can use jumper P39 to address the full 8K of ROM 27C64/28C68 on the board (jumpering 5-6) with a starting address at (say) E000H. Alternatively you can have two different sets on 4K monitor code in the 8K ROM and manually use P39 jumpered 1-2 or 3-4 to address them with a starting address at (say) F000H.

On this new V2 board we use the spare bit 1 of port D3H to set the ROM address line high or low. See here:-



If we jumper P39 7-8, we can in software directly control the high/low status of the address line A12 going to the PROM. Outputting 0H to port D3H will select the lower 4K section of the ROM. Outputting 2H to port D3H will select the upper 4K section of the ROM. Remember outputting 01H to port D3H will totally remove the ROM from the Z80's address space (see above).

Z80 Monitor Software.

This ability to flip 4K "pages" within the ROM effectively allows one to almost double the effective size of the code in your Z80 monitor. The only thing to watch out for is that the actual code used for the switch reside in EXACTLY the same place in both 4K pages. I have modified my well tested Master.Z80 monitor to have Floppy disk/CPM booting code in the lower 4K and the ability to download a ZModem formatted file directly to RAM from a PC (via a serial port) in the upper 4K. You can actually do a lot with 8K of ROM with a Z80 -- just be careful about the common code used during the switch.

Here is a Video of the V2-Z80 Board Monitor with the file transfer command running:-

0:00 / 4:31

Please see here for more information on software to run with this board.

BUGS

Other than the need for a 1K pull-up resistor for pin 2 of U26 (described above), there are no reported bugs with this board so far.
(This bug has been corrected on the V2.1 and latter boards)

The jumper settings for the V2.1 board are different than for the earlier boards. Henry Broekhuyse has written up a nice sumary here.

S100 Bus Master/Slaves.

Please note this board is set to act as an IEEE-696 bus master. It should work with any of our other CPU boards described on this site. It is important to remember however that when this CPU board relinquishes control of the bus to a slave device, it inactivates all of its address, data, status and control lines while the slave has control of the bus. The S100 bus signals ADSB*, DDSB*, SDSB* and CDSB* will all go low as specified by the IEEE-696 protocol. Some older S100 bus boards DMA driven boards may not expect this.

A Production S-100 Board

Realizing that a number of people might want to utilize a board like this together with a group of people on the Google Groups S100Computers Forum, "group purchases" are made from time to time. Please see here for more information.

The links below will contain the most recent schematic of this board.
Note, it may change over time and some IC part or pin numbers may not correlate *exactly* with the text in the article above.

PDF FILE OF THE MASTER Z80 MONITOR V4.7 (Using DS12887 Clock Chip) (V4.7 8/14/2018)
"MASTER.Z80" MONITOR SOFTWARE V4.7 (V4.7, FINAL 8/14/2018)

MOST CURRENT VERSION TEXT FILE OF THE MASTER Z80 MONITOR (MASTER0.Z80 V5.4) (Using two 4K Pages) (V5.4 6/18/2019)
MOST CURRENT VERSION TEXT FILE OF THE MASTER Z80 MONITOR (MASTER1.Z80 V5.4) (Using two 4K Pages) (V5.4 6/18/2019)
MOST CURRENT "MASTER.Z80" MONITOR SOFTWARE V5.4 (.ZIP file) (Using two 4K pages) (V5.4 6/18/2019)

MOST CURRENT Z80 CPU V1 BOARD SCHEMATIC (FINAL, 02/28/2010)
MOST CURRENT Z80 CPU V1 BOARD LAYOUT (FINAL, 02/28/2010)

MOST CURRENT Z80 CPU V2 BOARD SCHEMATIC (V2.1, FINAL, 4/22/2010)
MOST CURRENT Z80 CPU V2 BOARD LAYOUT (V2.1, FINAL, 4/22/2010)
KiCAD BOM for this board V2.0a (.pdf file) (V2.0 6/7/2019, Supplied by Rick Bromagen)
KiCAD BOM for this board V2.0a (.xls file) (V2.0 6/7.2019, Supplied by Rick Bromagen)
Z80 CPU V2 BOARD JUMPER BLOCK DIAGRAM (V2, 03/24/2014)
KiCAD files for V2.01 board (S100 Z80 V2.01.zip 12/16/2016)

Other pages describing my S-100 hardware and software.
Please click here to continue...

This page was last modified on 12/06/2021