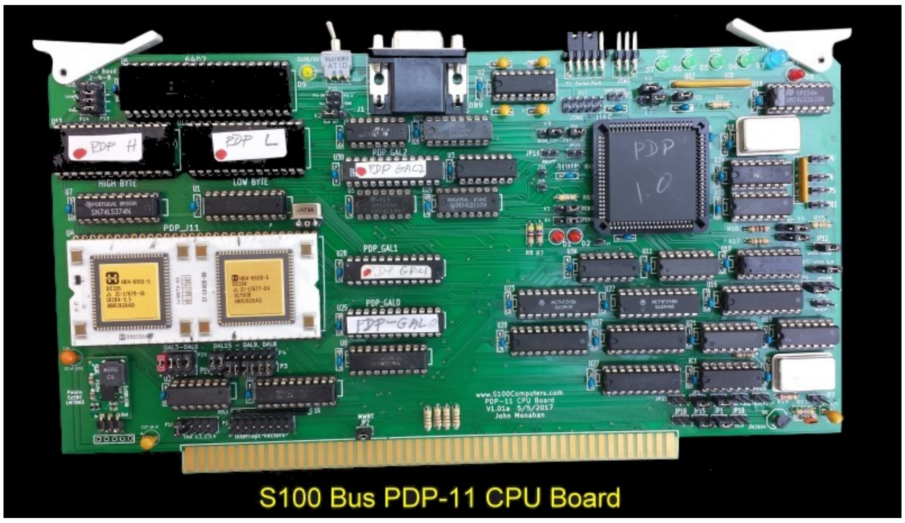
S100 Computers
A Web Site For S-100 Bus Computer Owners

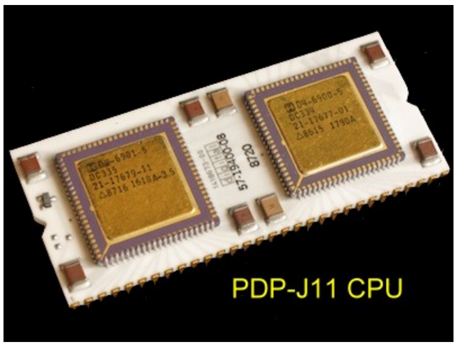
The PDP- J11 S100 Bus CPU Board.



The PDP-11 architecture is an instruction set architecture developed by Digital Equipment Corporation (DEC) during the early 1970's. It was implemented by LSI central processing units (CPUs) and microprocessors used in PDP-11 minicomputers. It was in wide use for most of the 1970s, but was eventually replaced by the more powerful VAX-11 architecture in the 1980s. Sixteen-bit words were stored in a little-endian (with least significant bytes first) format. Thirty-two-bit data, were supported as extensions to the basic architecture, e.g., floating point in the *FPU Instruction Set* and *double-words* in an "Extended Instruction Set". It was a vast improvement over the prior previous PDP-8 and PDP-9 (which had 12- and 18-bit words respectively).

Introduction to the J11 CPU

The single VLSI PDP J-11 CPU chip was DEC's fourth and last PDP-11 microprocessor design, and the first to be done in CMOS. It was an evolution of their earlier PDP F-11 microprocessor. The project was co-developed with Harris Semiconductor. The J-11 was intended to be the definitive microprocessor for the PDP-11 family by providing the full functionality and performance of the PDP-11/70 in a single microprocessor. So the J-11 incorporated most of the elements found in the 11/70 -- such as dual register sets, data space, supervisor mode -- as well as then more modern inventions such as S*ymmetric Multiprocessing* (SMP) support. Microcode-based floating point was standard, with accelerated floating point available as an option. A "*Commercial Instruction Set* " microcode was also intended to be an option.



The J-11 was in fact a multi-chip module consisting of 2 chips mounted on a single 60-pin, over-wide, DIP ceramic carrier , both made by Harris Semiconductor.

The Control Chip implements the micro word access and sequencing functions of the J-11 chip set.
Key features were:-

- ROM/PLA control store (512 x 25 bit PLA terms, 768 ROM terms)
- Chip set micro sequencer
- Next address logic
- Micro subroutine stack
- Interrupt logic
- Abort logic
- Initial decode PLA (Q logic)
- External interface sequencer
- Instruction prefetching logic

The Data Chip implemented the instruction execution and memory management data paths of the J-11 chip set. It shares the responsibility for the external interface and for instruction prefetching with the Control chip. The data chip operates under the control of micro words fetched from the Control chip.
Its key features were:-

- Execution unit
- PDP-11 architectural general registers (16 bit): dual register set, three stack pointers
- Processor status word (PSW)

- Microcode temporary registers (32 bit)
- Full function arithmetic/logic unit (32 bit)
- Single bit shifter
- Byte swapper
- Conditional branch logic
- Memory management unit
- PDP-11 memory management registers: kernel, supervisor, user; instruction and data spaces
- Address translation logic (22 bit)
- Protection logic
- External interface sequencer
- Instruction prefetching logic

A separate third companion high speed Floating Point chip was introduced in 1983.
Its key features were:-

- High performance: accelerated floating point execution by 5X
- f_ and d_floating point format support
- Full PDP-11 floating point instruction set, including MODf
- Arithmetic error checking and reporting
- Single +5V supply

This was an ill-starred chip, with latent bugs requiring its recall from the field on two separate occasions. However by the time the **MicroVAX** arrived (1980's), all issues were resolved for its incorporation in that system.

In many senses the J-11 was a chip ahead of its time. Back then CAD tools were primitive. Having a geographically split team (DEC was in Maynard, Massachusetts, and Harris was in Melbourne, Florida) did not help. The **Data chip** component grew far larger than anticipated. Due to that and other circuit design problems, the first passed CPUs barely ran at 1.25Mhz. It required many passes, and many fixes, to get the chip set to function at 3.75Mhz and then later at 4.5Mhz. However by 1988 the J11 could reach speeds of 18MHz. Indeed the last round of J11 chips in 1990, using modern circuit board components topped out at 20MHz.

The PDP-J11 went on to be perhaps the most successful microprocessor for minicomputers of the 1970's and early 1980's. Members of the PDP-11 line were sold in very high numbers, thanks to the then growing OEM industry. The VAX line of DEC computers began its life as an enhancement to the PDP-11 architecture, the first VAXen were sometimes referred to as **PDP11-7xx**, rather than their official label, **VAX-11/7xx.**

The PDP-J11 CPU Board as an S100 Bus Slave.

As with all our previous CPU boards the normal S100 bus configuration for this board will be where we boot up with the master S100 bus Z80 board and then transfer control over to the PDP-J11 CPU. The board can however also be configured to be a bus master. This I will leave to the user -- It's not too difficult, see the write-up on the 80286 board.

The J11 utilizes 22 Address lines. We will set A22 and A23 to ground when the board is active. This yields a 4MB address space. The J11 utilizes a 16 bit data bus to read 8 or 16 bit data. Ignoring the relevant high/low byte for 8 bit data in. Data writes can be 8 or 16 bits. This requires careful data out buffer redirection on the S100 bus using sXTRQ* etc.

Unlike the Intel 8080/80x86 family of CPUs there are no specific I/O lines. A similar case occurred with our 68000 CPU Board. Any and all addresses in the range (3F)E000H to (3F)FFFFH are understood to access I/O ports by the J11. This board converts addresses in that range to the corresponding S100 bus I/O ports in the range 0 to FFH. Only addresses 0H to 3FDFFFH are available to RAM. Further within the 3FE000-3FFFFH block, 3FFFE0H to 3FFFE8H are "System Registers" (see below). Further complicating matters when the J11 first powers up only the address range 0H to FFFFH is available (16 bits). The CPU internal memory management registers must be programed to access the 4MB (in 64K blocks). So in startup mode accessible RAM is 0H to DFFFH. It is important to remember that any and all addresses in the range E000H to FFFFH are automatically translated to address 3FE000h to 3FFFFFH. In other words, with memory management inactive, if the address lines A15 & A14 are high then address lines A21-A14 are all high - no exceptions.

The J11 Internal Console Monitor

Unlike any other CPU we have seen this CPU has its own internal monitor. This monitor is always there and never changes. Digital calls it the **"ODT Console"**. That's the good news, the bad news is, its commands are very limited. Here is a summary:-

ODT Console Command	Symbol/Example	Function
Slash	n/	Opens a particular memory location defined by the octal (not hex) value n
Carriage Return	CR	Closes an opened location
Line Feed	LF	Closes a location and opens the next location. (Note Ctrl j = LF on a PC keyboard)
Display a register	$n or Rn	Displays the contents of an internal register (n=0-7)
Processor Status Word	S	Displays the PS. (Note this command must follow a '$' or 'R' command)
Go	nG	Jump to and start at RAM location 'n'
Proceed	P	Resumes execution of a program

Most of the commands are fairly obvious. However remember all values are 16 bits and are in octal. Also remember all PDP-J11 addresses must start on an even address line.
The J11 prompt character is always an '@'.
To display the value of RAM at 100H (400 Octal) enter

```
@400/
```
you should see something like:-
```
400/052523
```

To display three RAM locations at 100H - 106H

```
@400/
LF
LF
CR
```
Remember **Ctrl J** is **LF** on a PC keyboard.

Depending on the values in RAM the output could be something like this:-

```
0000400/ 000000
0000402/ 345670
0000404/ 123456
@
```

To display the current value in the R4 register:-

```
@R4/
```

Depending on the values in R4 the output could be something like this:-

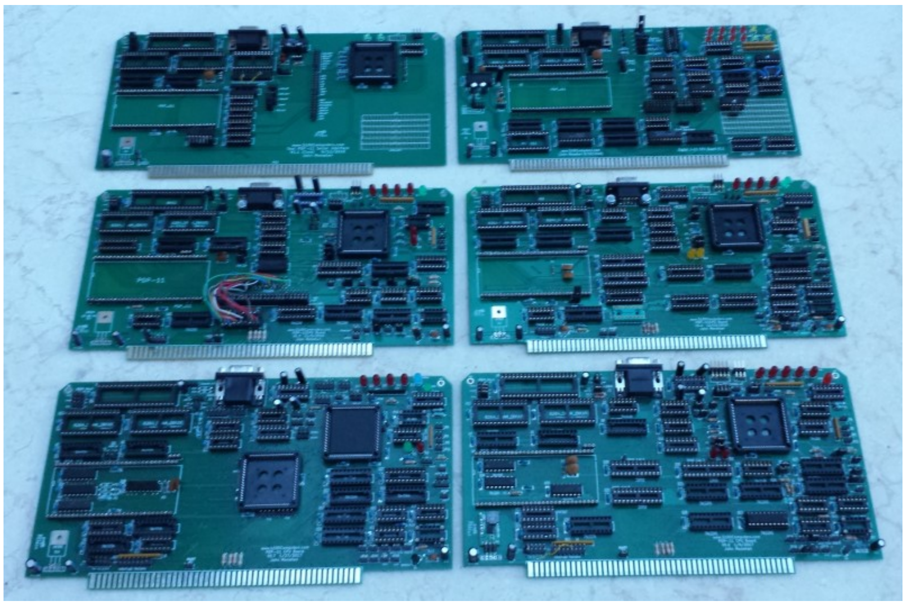
```
@R4/ 000001
@
```

A more complete description of the J11 monitor commands are found in chapter 5 of the DC-11 User Guide. This is an essential document to read and can be downloaded from the bottom of this page.
Within the monitor:-
The Console **Output Data** port is always 3FF76H
The Console **Output Status** port is always 3FFF74H (bit 7)
The Console **Input Data** port is always 3FFF72H
The Console **Input Status** port is always 3FFF70H (bit 7)

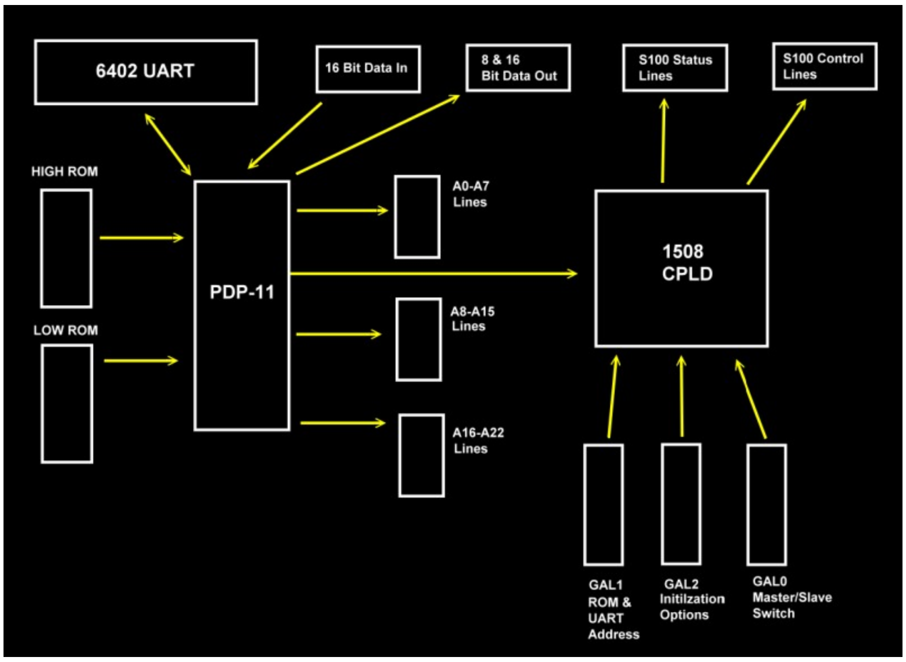
These port/bit designations cannot be changed. The good news here again is, they exactly conform to the use by a 6402 UART. In fact that UART was probably designed with this role in mind.

The Circuit

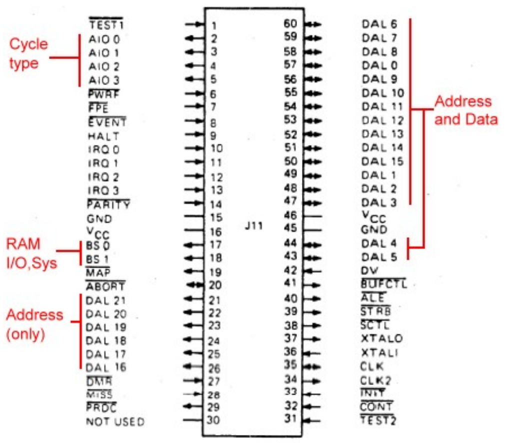
I got the idea for the basic idea for the layout of this board from the pioneering work of Brent Hilpert in Vancouver in an article a while back called a "PDP-11/Hack". The core circuit splices the PDP-J11 chip into a 6402 UART and 16K of static RAM. See here. The circuit was amazing, simple, and worked well. It has recently been further extend by Peter Schranz in Switzerland, see here. Nevertheless it took some 6 prototypes to arrive at this final board.



I ended up with following core circuit for the S100 bus.

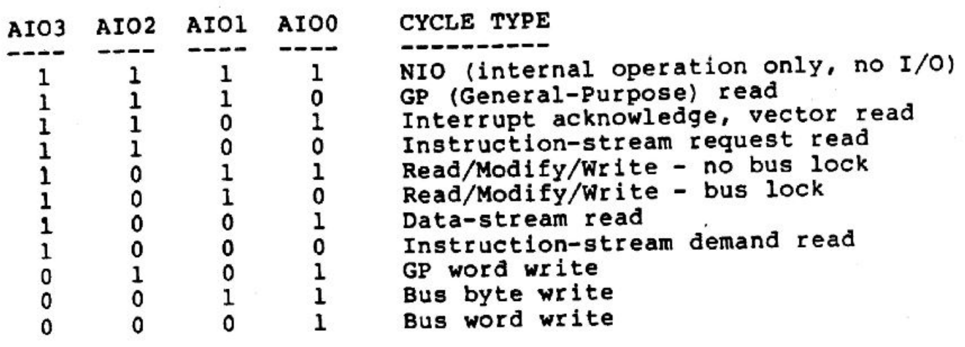


For those that have built some of our other CPU boards this is a fairly familiar layout. The CPLD traps all the PDP status and control signals and translates them into S100 bus signals. The process is simple and straightforward. In fact the whole process could be done by a few more GALs, but the CPLD allows you absolute control of the bus in more detail. What is needed however is a careful analysis of all the PDP timing signals.
Here is the pinout diagram for the chip:-



DCJ11 Pin Assignments

These are well explained in the DC-11 User Guide. As with many other early CPUs the address lines and control lines are multiplexed (DAL0-15). The CPU BS0 and BS1 pins select RAM, the Internal Registers or External I/O ports. The four CPU AIO0-AIO3 pins define the type of bus cycle.



AIO3	AIO2	AIO1	AIO0	CYCLE TYPE
1	1	1	1	NIO (internal operation only, no I/O)
1	1	1	0	GP (General-Purpose) read
1	1	0	1	Interrupt acknowledge, vector read
1	1	0	0	Instruction-stream request read
1	0	1	1	Read/Modify/Write - no bus lock
1	0	1	0	Read/Modify/Write - bus lock
1	0	0	1	Data-stream read
1	0	0	0	Instruction-stream demand read
0	1	0	1	GP word write
0	0	1	1	Bus byte write
0	0	0	1	Bus word write

The CPU BUFCTL* signal determines the data direction. The CONT* input allows you to essentially insert wait states in a cycle. There are 8 pins associated with Interrupts and DMA control. There is not enough room on this board for separate DMA controller chips etc. I will do, as I did for our 8086 CPU series, a special "PDP-11 Support board" later.

Here is the core code in the CPLD for S100 bus control:-

```
Pinnode = BUS_READ;                                         /* BUS CYCLE TYPE */
Pinnode = BUS_BYTE_WRITE;
Pinnode = BUS_WORD_WRITE;

!BUS_READ = ((LAIO3 & LAIO2 & !LAIO1 & !LAIO0)              /* Instruction read - request */
          # (LAIO3 & !LAIO2 & LAIO1 & LAIO0)                /* Read-modify-write no bus lock */
          # (LAIO3 & !LAIO2 & LAIO1 & !LAIO0)               /* Read-modify-write bus lock */
          # (LAIO3 & !LAIO2 & !LAIO1 & LAIO0)               /* Data read */
          # (LAIO3 & !LAIO2 & !LAIO1 & !LAIO0));            /* Instruction read - demand */

!BUS_WORD_WRITE = !LAIO3 & !LAIO2 & !LAIO1 & LAIO0;         /* Bus Word WRITE*/
!BUS_BYTE_WRITE = !LAIO3 & !LAIO2 & LAIO1 & LAIO0;          /* Bus BYTE WRITE */

bpSYNC = STRB & !XFERII;


bMWRT = BUFCTL & !PDP_WR & (!LBS_MEM & ROM_CS) & !XFERII;
!bpWR = BUFCTL & !PDP_WR & ((!LBS_MEM & ROM_CS) # (!LBS_IO & UART_ADDRESS)) & !XFERII;


bsMEMR = !BUFCTL & PDP_WR & (!LBS_MEM & ROM_CS) & !XFERII & !BUS_READ;            /* Onboard ROMS local to board */
bpDBIN = !BUFCTL & PDP_WR & ((!LBS_MEM & ROM_CS) # (!LBS_IO & UART_ADDRESS)) & !BUS_READ & !XFERII;


!OE_D  = !bpWR & !BUS_BYTE_WRITE & LA0 & !XFERII;                                 /* 8 Bit Write Odd address on DAL 8-15 */

!OE_A  = bpDBIN & !LBS_IO & !LA0 & !XFERII;                                       /* 8 Bit I/O Read Even address on DAL 0-7 */


DIRECTION_L = BUFCTL;

!OE_B = (((!bpWR & !BUS_WORD_WRITE & !XFERII) # (bpDBIN & !LBS_MEM & !XFERII))    /* 16 Bit Rd or Wr and 8 bit RAM Rd */
      # (!bpWR & !BUS_BYTE_WRITE & !LA0 & !XFERII));                              /* 8 Bit Write Even address (DAL 0-7) */

DIRECTION_H = BUFCTL;

!OE_C = ((!bpWR & !BUS_WORD_WRITE & !XFERII) # (bpDBIN & !XFERII));               /* 16 Bit Rd or /Wr and 8 bit Rd, DAL 8-15 */


!bsXTRQ = bpDBIN # (!bpWR & !BUS_WORD_WRITE);                /* All reads are 16 bits, For 8 Bits writes bsXREQ must be HIGH */


bsOUT = !bpSYNC & BUFCTL & !PDP_WR & !LBS_IO & !XFERII & (!BUS_WORD_WRITE # !BUS_BYTE_WRITE) & UART_ADDRESS;

bsINP = !bpSYNC & !BUFCTL & PDP_WR & !LBS_IO & !XFERII & !BUS_READ & UART_ADDRESS;
```

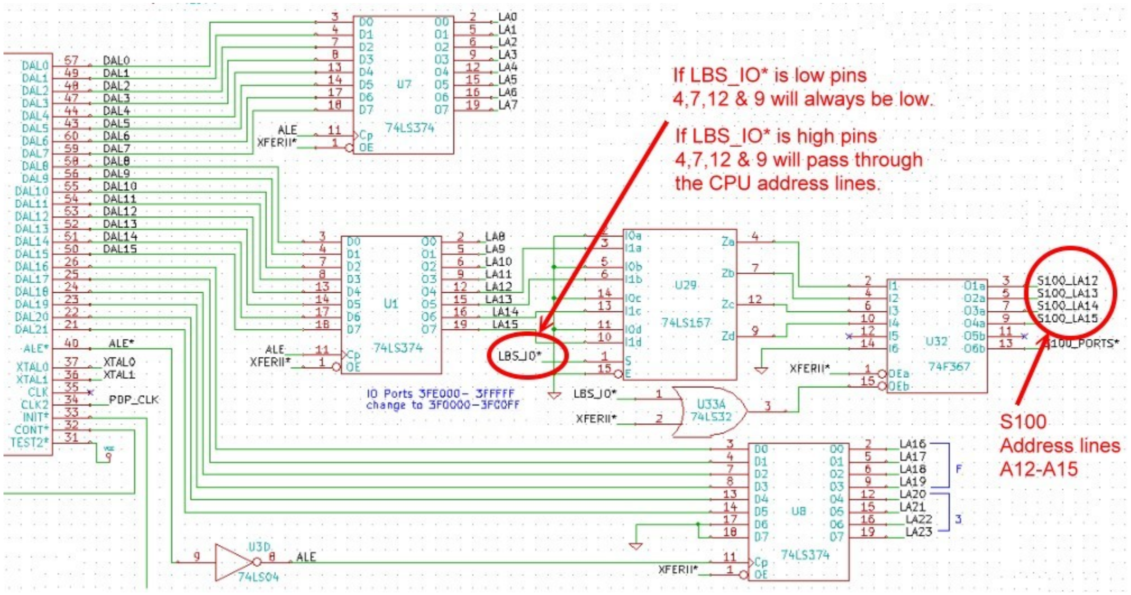
```
bsINTA = 'b'0;                                                          /* Interrupts not done yet */

!bsWO  = bMWRT # bsOUT;

bsM1 = ((ALE & !XFERII & (LAIO3 & LAIO2 & !LAIO1 & !LAIO0))         /* 1100 Instruction Read (request) */
     # (ALE & !XFERII & (LAIO3 & !LAIO2 & !LAIO1 & !LAIO0)));       /* 1000 Instruction Read (demand) */

!ROM_CS = !ROM_ADDRESS & !XFERII;                                    /* Onboard ROMs CS* (Pin 20 of 28C64's) */
!PHANTOM = !ROM_CS;                                                  /* No conflict with onboard RAM/ROM with S100 bus RAM */
                                                                       /* For testing will have RAM/ROMs at C000H (140,000 Octal)
*/

!UART_CS = !UART_ADDRESS & !LBS_IO;                                   /* UART is 3FFF70-3FFF77 (only) (No S100 bus ports allowed)
*/


!CONT = !SCTL & !READY;                                              /* S100 Bus wait states required, stretch PDP-11 Cycle */

!LED1 = CONT;                                                        /* For debugging etc. */
!LED2 = !SIXTN;
```

We need to be especially careful with our address lines. Any an all address in the range 3FFF70H to 3FFF76H need to go directly to the J11 for console/UART IO (and not act as ports on the S100 bus). Any and all addresses in the range C000H to DFFFH need to go to the boards onboard ROM's and not input RAM data from the S100 bus. The "PDP_GAL1" (U28) 22V10 GAL flags these two situations as shown in the schematic (UART_ADDRESS* and ROM_ADDRESS*) to the above CPLD code.

One final CPLD function, the CPLD does provides the 6402 UART with a clock signal. We could have used something like an Intersil HD4702 baud rate generator chip but to save space I just divided down the CPLD 20MHz clock signal.

```
/* Need to divide the 20Hz Oscillator down to obtain the correct BAUD Rate for the UART. */
/* For 9600 Baud (156.24/16 = 9765) is close enough for most Serial ports and is never a problem with the USB/Serial adaptors */

Pinnode = [CD6..0];

CD0.t = 'b'1; /* 10MHz */
CD1.t = CD0; /* 5 MHz */
CD2.t = CD0 & CD1; /* 2.5 MHz */
CD3.t = CD0 & CD1 & CD2; /* 1.25 MHz. */
CD4.t = CD0 & CD1 & CD2 & CD3; /* 625 KHz */
CD5.t = CD0 & CD1 & CD2 & CD3 & CD4; /* 312 KHz */
CD6.t = CD0 & CD1 & CD2 & CD3 & CD4 & CD5; /* 156 KHz */
[CD6..0].ckmux = UART_CLK_IN;

BAUD_CLK = CD6;
```

There is one further address lines complication. The PDP-11 designates addresses 3FE000H - 3FFFFFH as port IO's. We need to "translate them" to 8/16 bit ports 0000H-0FFFFH on the S100 bus. This must be done in hardware and be completely transparent to the J-11 CPU. The following circuit does the trick:-



If LBS_IO* is low pins 4,7,12 & 9 will always be low.
If LBS_IO* is high pins 4,7,12 & 9 will pass through the CPU address lines.
S100 Address lines A12-A15
IO Ports 3FE000- 3FFFFF change to 3F0000-3FC0FF
LBS_IO*
U7 74LS374
U1 74LS374
U29 74LS157
U32 74F367
U33A 74LS32
U8 74LS374
U3D 74LS04
S100_LA12
S100_LA13
S100_LA14
S100_LA15
S100_PORTS*
ALE
XFERII*

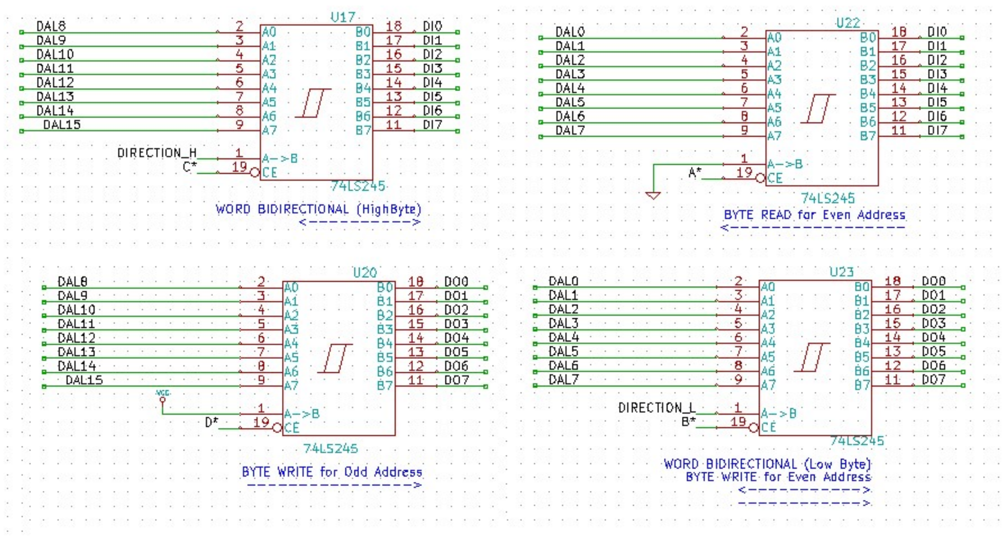
For any port I/O to/from the CPU the LBS_IO* signal generated by U10B will be low. (U10 in fact defines when the CPU is doing a Memory, a "System Memory/Onboard monitor", I/O or Interrupt access to/from the bus).

The GAL U25, is our by now our familiar "GAL0" used to take over S100 bus control with this board from the Z80 bus master. It has been slightly changed here to squeeze in a port to inactivate the onboard ROMs when booting the likes of RT11. Remember if you are using the onboard port EDH (in the GAL) to activate the board be sure you jumper P36 1-2 & 3-4. This is because other boards may be expecting the TMA0* line to be low when this board is active. The onboard port alone will not lower the TMA0* line. Please see here about programming GAL's.

The third GAL, U30, is used in a special circuit to define where in RAM the CPU will start up from on power up. Immediately upon power up the CPU reads the values of the data lines for various initial configurations. It is described below and in detail in the Digital PDP-11 user Guide (page 8-8).

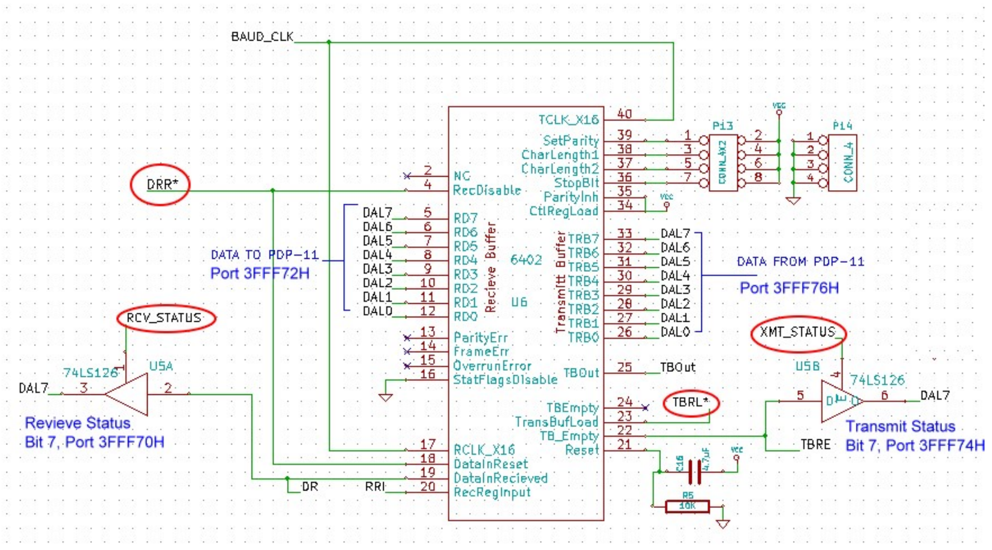
Finally we need to look at the data lines themselves. As I said above the address and data lines are multiplexed on the "DAL0- to DAL15" lines. The ALE* signal is used to latch the address lines in the early part of the CPU cycle. The second half of the bus cycle relates to data. The PDP-J11 processes both 8 and 16 bit data. Both 8 and 16 bit data in is read as 16 bit data. For 8 bits the CPU just reads the upper of lower half of the 16 bit data. However for data writes there are separate signals for 8 and 16 bit data. These are flagged with the AIO0-AIO3 signals from the CPU. We must be especially careful with the S100 bus that the 8 bit data is sent out on the "Data Out" lines on the bus and on the "Data In" lines for 8 bit data in even though in the case of the latter the CPU expects 16 bit data. Unlike say, the 8086, the PDP-J11 will not write 16 bit data to an odd address. If for example you try to send 16 bit data to an odd port number you will trigger a CPU interrupt (see below).

Again our CPLD code takes care of the above requirements using four 74LS373's in the following layout:-

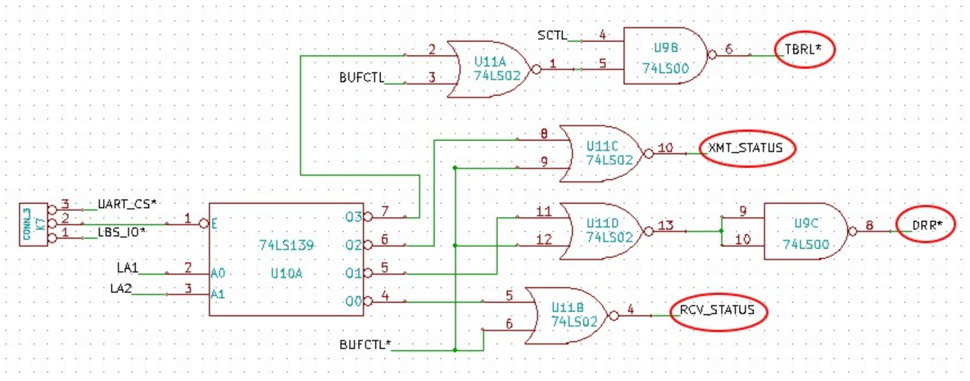

U17
74LS245
DIRECTION_H
C*
WORD BIDIRECTIONAL (HighByte)
U22
74LS245
A*
BYTE READ for Even Address
U20
74LS245
D*
BYTE WRITE for Odd Address
U23
74LS245
DIRECTION_L
B*
WORD BIDIRECTIONAL (Low Byte)
BYTE WRITE for Even Address

The rest of the S100 bus interface should by now be very familiar with those of you that have used our other S100 bus CPU boards. The status and control line buffers are unchanged. The circuits/jumpers to run this CPU as an S100 bus master are all there. The PDP-J11 accepts four interrupt lines (IQ0-IQ3) as well as a special "Event" line. These are brought into the CPU from the S100 bus vector interrupt lines via P15, U24 and U9.

The only other unusual circuit is the onboard UART (U6) . The ODT Console monitor requires a hard wired set of data in/out ports and status port/bits. While these could probably be constructed in 74LSxxx hardware they must act completely transparent to the CPU. In particular the status bits in & out are precisely defined as to when they change. Fortunately the common 6402 UART matches these requirements exactly. In fact that UART may have been built with this application in mind at the time. Here is the core circuit for the UART:-



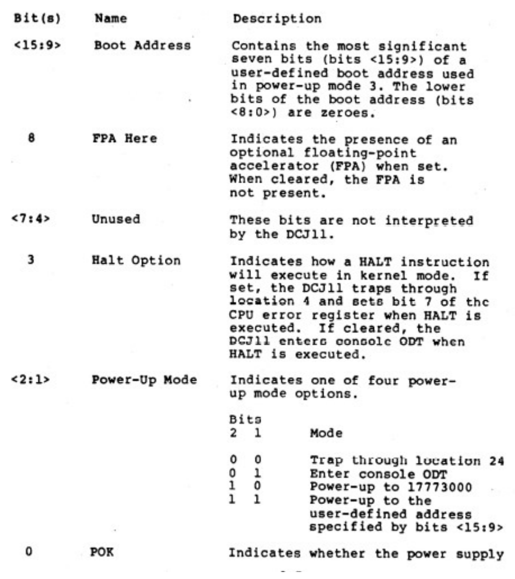
Remember these ports and status bits are hard wired within the CPU chip and cannot be changed. Thanks to Brent Hilpert (see above), this UART can be easily addressed with the following circuit:-



In fact with the above circuit and UART you have a basic CPU circuit the talks to you (without the need for RAM or ROM)! I know of no other CPU chip of this era, that has this capability. It greatly facilitates a step by step construction of a sophisticated S100 bus PDP-J11 CPU board.

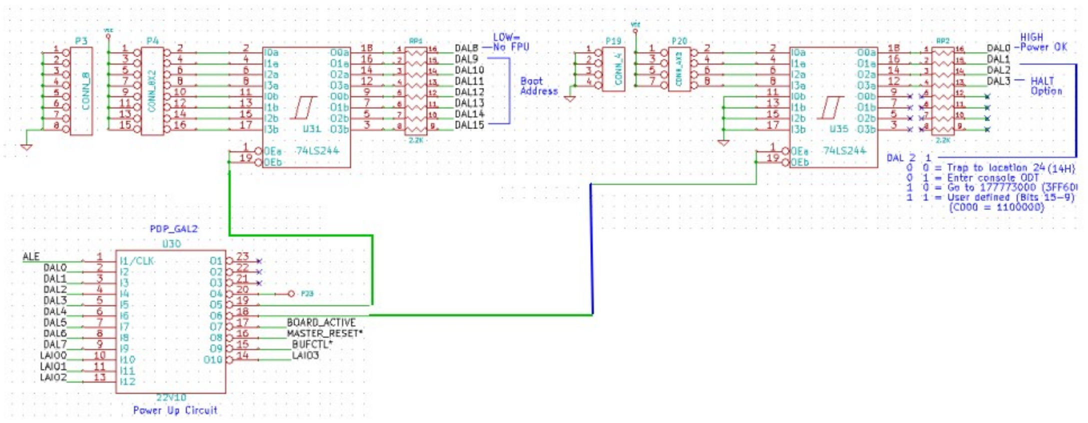
PDP11 CPU POWER UP CONFIGURATION.

There is one remaining circuit on this board we need to discuss. Normally upon power the CPU looks at the very initial values of its DAL0-DAL15 lines for startup information. On pages 8-7 to 8-8 of the PDP User Guide there is a description of a circuit that allows you to define where the CPU starts in its 64K RAM space. Upon power up the CPU inputs the DAL 0-15 lines when a General Purpose Read is placed on the AIO03-AIO0 lines (1110) and the Read code on DAL 0-7 is 000 or 002. This is a single read/pulse. The PDP interprets the 16 bit data as follows:-



Bit(s)	Name	Description
<15:9>	Boot Address	Contains the most significant seven bits (bits <15:9>) of a user-defined boot address used in power-up mode 3. The lower bits of the boot address (bits <8:0>) are zeroes.
8	FPA Here	Indicates the presence of an optional floating-point accelerator (FPA) when set. When cleared, the FPA is not present.
<7:4>	Unused	These bits are not interpreted by the DCJ11.
3	Halt Option	Indicates how a HALT instruction will execute in kernel mode. If set, the DCJ11 traps through location 4 and sets bit 7 of the CPU error register when HALT is executed. If cleared, the DCJ11 enters console ODT when HALT is executed.
<2:1>	Power-Up Mode	Indicates one of four power-up mode options. Bits 2 1 Mode 0 0 Trap through location 24 0 1 Enter console ODT 1 0 Power-up to 17773000 1 1 Power-up to the user-defined address specified by bits <15:9>
0	POK	Indicates whether the power supply

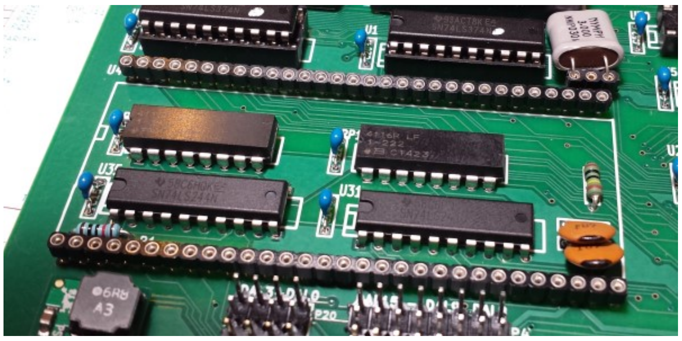
I incorporated the following circuit on the board to trap these situations:-
Here is the circuit:-



The PDP GAL2 Code is:-

```
/P23  := /DAL0 * /DAL1 * /DAL2 * /DAL3 * /DAL4 * /DAL5 * /DAL6 * /DAL7            ;000
       + /DAL0 *  DAL1 * /DAL2 * /DAL3 * /DAL4 * /DAL5 * /DAL6 * /DAL7            ; 002

/U35   = /P23 * LAIO3 * LAIO2 * LAIO1 * /LAIO0 * /BUFCTL@
/U31   = /U35
```

We use the above GAL code to provide the one time only initial pulse to the two 74LS244's. The input values are determined by the jumpers P3,P4 and P19,P20.
Because space is tight on this board the circuit lies below the CPU as shown here.



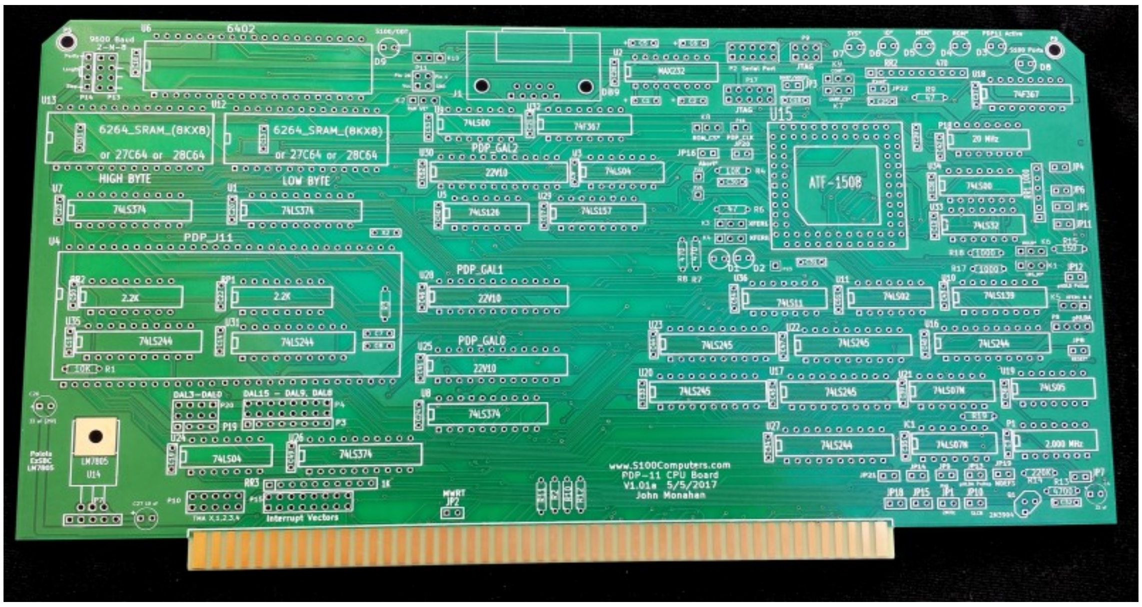
Step By Step Building the PDP-J11 CPU Board.

The build instructions are fairly simple for this board but because it is a complex board building it should not be rushed.
Please note the build instructions below are for the V1.01a version of the board.

As always, first examine the bare board carefully for scratches or damaged traces, use a magnifying glass if need be. A broken trace is almost impossible to detect by eye on a completed board..

Build the board up in functional steps. Avoid the temptation of adding everything at once and popping it into your S-100 box. Step by step is faster in the end -- trust me. We will set the board up as a true S100 bus slave. By programming the CPLD and with the additional circuitry on the board it can in fact run as a bus master. I will leave this arrangement to our more experienced users (hint, pins 6 & 8 of the CPLD and P8,K6)..

Here is a picture of the bare board:-



Solder in all the required IC sockets, resistors, the transistor Q1, resistor arrays, capacitors, jumpers, and the 5V voltage regulator. For the latter you can use a 7805+heatsink approach (e.g... Jameco # 9246333, a L7805CV), or you can use more modern switcher regulators such as thee Pololu # 28500(5V, 2.5A) or EzSBC.com # PSU5aa(5V 2.5A).. Do not add the LED's 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). Insert all jumper arrays. Be sure and have pin 1 of the CPLD socket on the LHS. For prototype boards I typically use common Aluminum Electrolytic caps. For the final board I use the corresponding Tantalum caps. Note K10 can be used with a 3 pin jumper, but it's best to use a small switch (see the picture above). Don't overlook R1 and R19..

Do NOT add sockets for U35, U31 or the resistor packs RP1 & RP2. We will (later) solder these components directly to the board (under the CPU), space is tight on this board..

For prototype boards I generally use "double swipe" IC sockets. For a critical board like this I prefer to use "Machine Tooled" IC sockets. However they are more expensive and you have to be particularly careful not to bend the IC pins. The two clock oscillators can have their own special sockets (e.g. Jameco #133006) but actually I find the "Machine Tooled" IC sockets make a better connection. I in fact solder the 2MHz oscillator (P7) directly to the board since it will never be changed..

Place the board in the bus. Check the voltage to sockets on the board is about 5V by placing the board in your S-100 system using an extender board. With no load you will typically get 5.00V (+/- 0.2V). BTW, your system should boot and run correctly with its Z80 master CPU board. 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 proper solder joints. 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.

Next insert all 9 LED's. Before soldering them in place ground one at a time pins 3,5,7,9 & 11 of U13, pin 13 of U32, on the CPLD side of resistors R7 & R8 and pin 2 of K10 to be sure they are all OK. (I always use a blue LED for D3 "Board Active" in my systems).

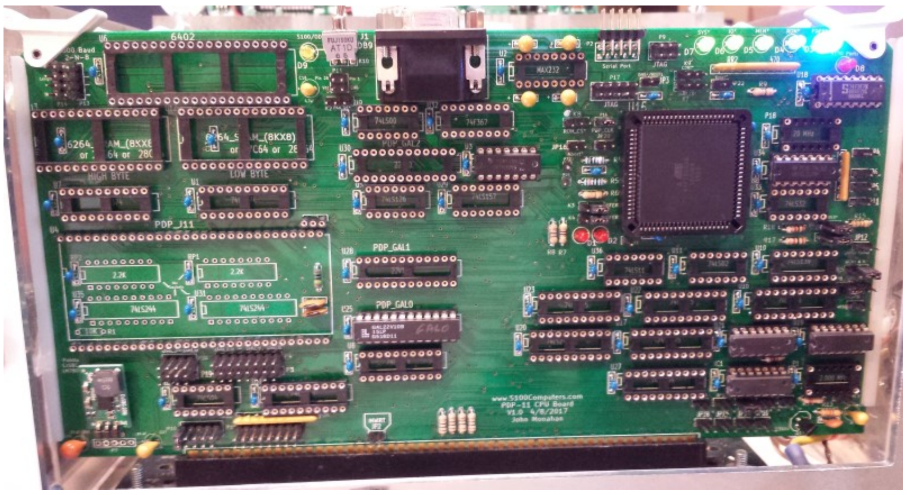
First we will construct the basic circuit required for the master/slave S100 bus flip.
Please remember that for 3 or more pin jumpers, pin 1 is always the square pad. It is not necessary the one on the top, or LHS.

First insert the CPLD in socket U15 taking care that pin 1 is in the center LHS. Program this CPLD wit the code provided at the bottom of this page. If you are using a virgin CPLD you probably don't need to program it first but to be on the safe side you don't want input/output gate conflicts with a previously programed chip. With no chips/jumpers on the board this is never an issue.

The Z80 should boot up correctly with the programmed CPLD.
Next add the programmed 22V10 GAL0 (U25). Again the code is at the bottom of this page. Note this is NOT the same "master/slave" GAL0 we use in all our recent CPU boards.

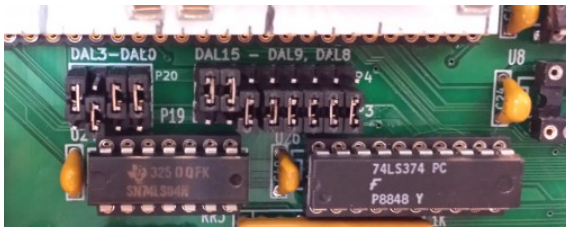
Add U19, (U34) and U3. Jumper P10 1-2 and K5 1-2. (Note pin 1 of K5 is on the RHS).
In your Z80 monitor "O" command (or however in your system you pass control over to an S100 bus slave device), check that that pin 10 of U3 goes from LOW to HIGH. Repeat the "O" command it should flip back to LOW.

Next add U18, IC1 and U21. Add jumpers K3 2-3, K4 2-3, P16, K1 2-3 and JP9. Jumper P8 1-2 and K6 2-3.
Now with the "O" command the LED D3 should light up. (The others are undefined). A reset should turn off this LED.
Here is a picture of the board at this stage.



Next we will add the UART serial interface to the board.
Add U2 (the MAX232N chip). Add U6 (the UART). Add U5, U26, U10, U11, U9 and U7. Add the 20 MHz Oscillator P18.
Assuming you will initially be using the DB9 serial socket J1, jumper P2 1-3, 2-4, 5-7 & 6-8. See the picture below. BTW, the P2 connector is pin for pin compatible with the serial connector on our Serial I/O board so later you can hookup a ribbon cable internally in your system to a serial DB9 connector at the back of your box.
For P13 jumper the bottom 3 pairs of jumpers. The is no connection to the P14 row of pins in our default configuration. Jumper JP22.

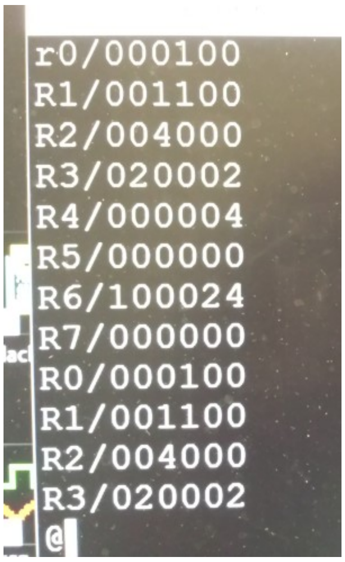
Next Jumper K7 1-2 and K9 2-3. To start, add a 10-12 MHz crystal to pins X2.
Program and add GAL2. Add U31 and U35 and the 2.2K resistor arrays under the CPU as shown above. Be sure they are parallel arrays and not a network. Jumper the power-up configuration jumpers as shown here:



Then carefully add the J11 CPU chip. The pins on this chip are delicate so be very careful inserting the chip into the two rows of sockets. Double check you don't have any chips in backwards.

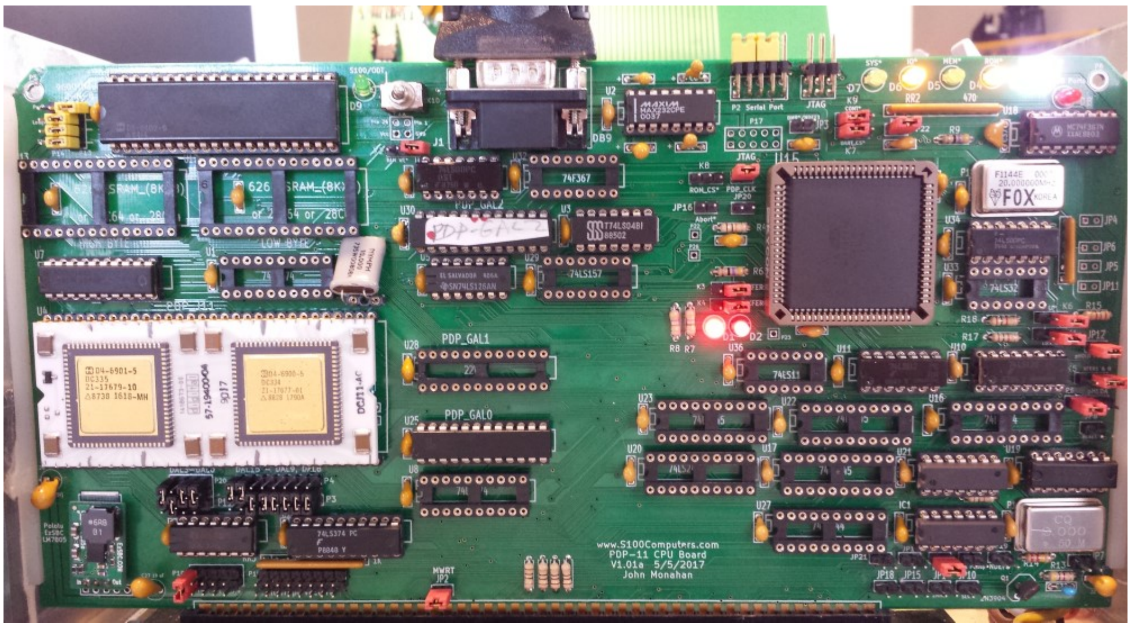
Next connect a serial connection to a TTY terminal on your PC to the top DB9 connector J1. There are many programs. I like "Absolute Telnet" but the free PuTTY one is also excellent. The serial connection to this board must be (in the default mode here), 9600 baud, 8 data bits, 2 stop bits, and no parity.

Power up your system and with the Z80 "O" command transfer control to this board.
Hit a few carriage returns on your TTY terminal. You should see the J11 ODT Console signon prompt '@'.
Since the is no RAM on the board at this time all you can do is display the CPU registers.
Type R0 followed by a '/' . You should see the current (random) value in the CPU's R0 register.
Typing further LF's (Crtrl j's), will cycle through all the CPU registers.
Here is an example output:-



```
r0/000100
R1/001100
R2/004000
R3/020002
R4/000004
R5/000000
R6/100024
R7/000000
R0/000100
R1/001100
R2/004000
R3/020002
@
```

Here is a picture of the board at this stage:-



It is absolutely essential you get the above diagnostic test working before going further. The ability to interact as described above is a fantastic advantage of this CPU chip. Most of our other CPU boards require almost a complete build of the board before getting any life out of them. Debugging such a non-functional board can be problematic.

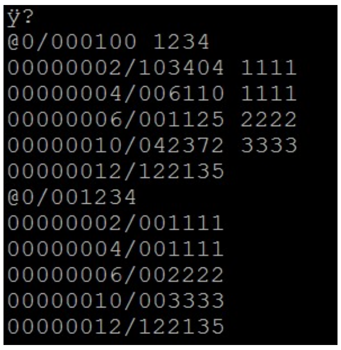
If you do not see the signon prompt, your first suspicion should be the serial connection to your PC. Disconnect the serial connection and try a loop test or a connection to another serial port (ideally at 9600 baud). Be sure you have all the jumpers exactly as described and shown above. Check that pin 25 of the UART pulses low when you enter a CR from your PC. Check you are getting a ALE* pulse on U7 pin 11 add U7 pin 1 is LOW.

Do not go forward until you get the above test to work.

We will next (temporarily) add 16K of local RAM to the board to further test/flush out our circuit. You could in fact skip this step since in the end we will be using only RAM on the S100 bus. However I really recommend you go the extra mile.

Add two HM6264LP-10 (16KX8) static RAM chips (or their equivalent) to U12 & U13. Also add U1.
Jumper P11 1-2 (the jumper should be vertical on the LHS connecting Vcc to pin 26 of the RAM chips).
Jumper K2 2-3 and K8 2-3.
Boot up the CPU with the usual "O" command.
You should be able to see and modify RAM starting at 0H in RAM.

Below we change the 16 bit words at 0-10H in RAM and then verify that we have done so. Remember all values are in octal. All addresses must be on an even byte boundary.



```
ÿ?
@0/000100 1234
00000002/103404 1111
00000004/006110 1111
00000006/001125 2222
00000010/042372 3333
00000012/122135
@0/001234
00000002/001111
00000004/001111
00000006/002222
00000010/003333
00000012/122135
```

If you get this far you are well on your way to having a functional system. The next step is to replace the onboard 16K of RAM with access to the S100 bus RAM. For this we need to add the S100 bus data buffers and for the first time call upon the CPLD to step in and handle RAM I/O requests.

First remove the onboard RAM chips U12 & U13. We will not need them any more.
For the (later) onboard ROMs jumper K2 1-2, and K8 1-2 and remove jumper P11.
Add U29, U32, U33 and U8 (for the S100 bus extended address lines).
Add U17, U20, U22 and U23 (all data buffers to the S100 bus).
Add U16 and U27 (the S100 bus status and control lines).
Add the 2MHz Clock oscillator P1 and jumper JP1. (Note this assumes your Z80/bus master hands over the 2Mhz clock to the current slave).
Add the MWRT jumper JP2. (Note this assumes your Z80/bus master hands over the MWRT signal to the current slave).
Change jumper K7 2-3 and K8 2-3. Add JP1, JP14, JP15, JP12 .
Add U36. Program and insert GAL1 U28.
Repeat the above RAM read/write test.
Go back to your Z80 and fill RAM 0-100H with say 33H. Then check with the PDP-J11 that you see these values (031463 octal).
Within the ODT change a few values to (say) 01 octal.
Go back to your Z80 and confirm the value has changed. Remember within the ODT monitor all reads/writes are words on an even address boundary.

Next we need to add an EEPROM based monitor to the board.
This is where things get a little tricky. There are two ways you can do it. You can just download and burn the simple PDP Monitor I wrote and jump to it from the Digital ODT console via the UART or you can load a binary file into RAM from your PC (via your Z80 monitor with the XModem command) and then from the ODT jump to it.

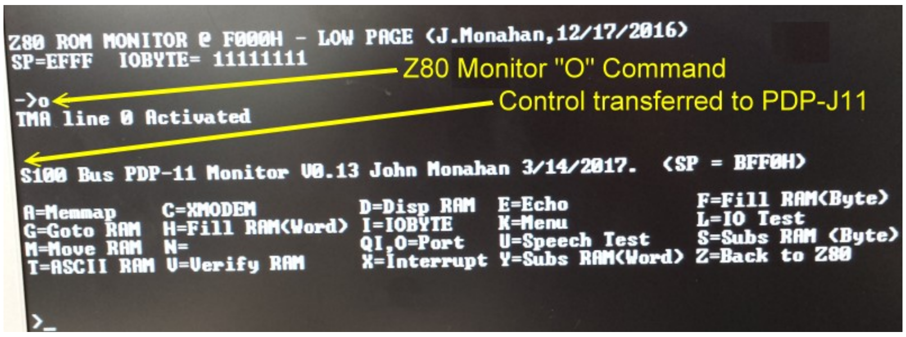
In the case of my EEPROM based code it is currently configured to start at C000H in RAM. So you would type 14000G/ from the ODT console. Note this assumes you are using the default GAL1 code provided below. (Unfortunately the PDP uses up E000H-FFFFH for its own use so we start at C000H for 16K).

In the case of your own code you should place it at 100H in RAM (since the PDP-J11 used the lowest RAM for interrupt vectors). Use the Z80 Master monitor "C" command to do this. From the ODT console you would enter 400G/. The complication in the latter case is that you need two serial connections. One (for example using our Serial IO board) for the file download. The other for this boards UART. Alternatively you could switch them resetting the speeds etc. -- a real pain if you do this often. In fact during my debugging/development of this board I had 3 serial connections from the PC. The third being the serial connection to the CPLD programmer! It's probably easies initially to take your chances using code for the onboard EEPROMS. It is provided at the bottom of this page.

The good news, as we will see below, is that I have written an XModem file transfer routine in the PDP_MON monitor (see below), so once you get the EEROM based monitor working you can use its "C" command to download any other test code directly to RAM from you PC and then jump to it. This makes writing and testing code simple an fast.

OK lets just pop in the two EEPROMS. I use X28C24A's (Jameco #74827). We are using 16 bit access, so split the code into High/Low bytes in your programmer. The high byte goes to U13 the low byte to U12. Be sure and add U24. (It once cost me 4 hours to find that out!). Be sure also K9 is 2-3. Jumper K10 1-2 (or the switch to the RHS) K2 should already from above be jumpered 1-2.
Note K10 determines where the console output will be displayed. If its jumpered 1-2 the output will be displayed on your Propeller Console IO board (and the LED D9 will be lit). If its jumpered 2-3 all output will remain on your PC Serial TTY terminal.
Power up your system and transfer control to the PDP board (Z80 "O" command). After you see the logon "@' prompt type:-
140000G

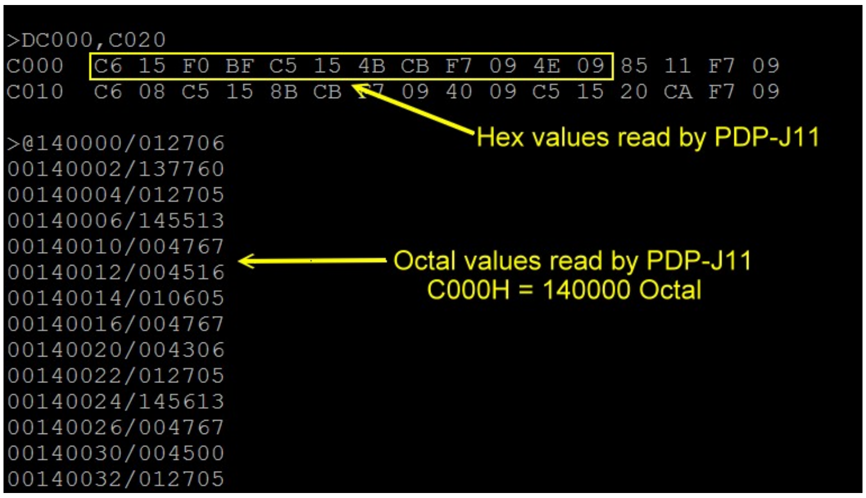
The PDP_MON monitor will signon as shown here:-


```
Z80 ROM MONITOR @ F000H - LOW PAGE (J.Monahan,12/17/2016)
SP=EFFF   IOBYTE= 11111111

->o
TMA line 0 Activated

S100 Bus PDP-11 Monitor V0.13 John Monahan 3/14/2017.  (SP = BFF0H)

A=Memmap    C=XMODEM          D=Disp RAM   E=Echo           F=Fill RAM(Byte)
G=Goto RAM  H=Fill RAM(Word)  I=IOBYTE     K=Menu           L=IO Test
M=Move RAM  N=                QI,O=Port    U=Speech Test    S=Subs RAM (Byte)
T=ASCII RAM V=Verify RAM      X=Interrupt  Y=Subs RAM(Word) Z=Back to Z80

>_
```

What if the PDP_MON does not signon? First be sure you have programmed the EEROMS correctly. Here is the first few bytes of each ROM. Please note this is for an early version of the ROM Monitor (V1.15a). It will probably NOT reflect the current monitor download. You need to look at your version of the PDP_MON.lst file to determine the correct values.

15 BF 15 00 FF 15 <--- High Byte ROM from position 0 (V1.15a)
C6 00 DF E0 FE C5 <--- Low Byte ROM from position 0 (V1.15a)

Check your monitor assembly listing. They should be as shown here:-
You can use the CPU ODT Console to see the octal listing.
Again please note the picture below is for a very early version of the monitor (V0.13). The values will be different for each monitor update, but the process is always the same.


```
>DC000,C020
C000   C6 15 F0 BF C5 15 4B CB F7 09 4E 09 85 11 F7 09
C010   C6 08 C5 15 8B CB F7 09 40 09 C5 15 20 CA F7 09

>@140000/012706
00140002/137760
00140004/012705
00140006/145513
00140010/004767
00140012/004516
00140014/010605
00140016/004767
00140020/004306
00140022/012705
00140024/145613
00140026/004767
00140030/004500
00140032/012705
```

If you do not see the correct values as listed in your PDP_MON.lst file , check your jumper connections. Check your coding of GAL1. Check that it pulses low for ROM access on its pin 22. Lets now discuss the PDP board monitor which I call PDP_MON.

The PDP-11 Onboard Monitor.

Let me say at the onset I don't like octal! I find it bloated and cumbersome to read. All the software below is written with HEX coding in mind. Furthermore to be consistent with our Z80, 8086 family and 68K monitors I have written the menu options (and routines) to be essentially the same. I'm sure I am reinventing the wheel with this simple monitor but I found it a great experience in learning the software aspects of the PDP-11.

I must say I am very impressed with the programming capability of the PDP chip. True it was initially a mini-computer, but relative to the Z80 and even the 8086 or 68K it is a joy to program. Was clearly way ahead of its time! You need to carefully read the PDP-11 handbook to get familiar with its programming capabilities.
Here is a very brief summary.

The PDP has 8 general purpose registers (0-7). Register 7 was normally used as the program counter and register 6 as the stack pointer. So really 6 general purpose registers.
These registers can be used in 8 addressing modes, value, index, index+offset etc.
The CPU has Word (16 bit) and byte (8 bit) address modes
There were processor defined error traps (to RAM 0-100H)
There were up to 4 vectored interrupts.
I/O ports were memory mapped
The CPU can address up the 4MB of RAM (in 64K or smaller segments).

Like the 68K family, PDP assemblers move values left to right, (not right to left as Intel does).
Here for example is our S100 Console In/Out routine:-

```
S100_CONSOLE_OUT:                          ; S100 Bus Console output routine <<<<
         BITB    #BIT2,@#S100_CONOUT_STAT  ; Check bit-2/ready of Propeller board Console Out port (0H)
         BEQ     S100_CONSOLE_OUT          ; busy-loop while bit-2 is 0
         MOVB    R0,@#S100_CONOUT_DATA     ; Send ASCII to Propeller board Console Out port (01H)
         RTS     PC                        ; Note R0 contains ASCII character (as a Byte)

CS100_CONSOLE_IN:                          ; S100 Bus Console input routine <<<<
         BITB    #BIT1,@#S100_CONIN_STAT   ; Check bit-1/ready of Propeller board Console In port (0H)
         BEQ     S100_CONSOLE_IN           ; Nothing there while bit-1 is 0
         MOVB    @#S100_CONIN_DATA,R0      ; Get ASCII from Propeller board Console In port (01H)
         RTS     PC                        ; Note R0 contains ASCII character (as a Byte)
```

The syntax is fairly clear. The '@' means a RAM location. An @R0 means the RAM location pointed to by R0. With most assemblers, you can also use (R0).
The '#' means the next item is a number. You have to decrease or increment the stack yourself and you can control the return location from a subroutine with a register -- though its almost always the PC.

The instruction set is quite extensive see here for a handy summary. You can see it is very extensive particularly for BRANCH instructions.

The next challenge was finding an assembler that would work on a PC (with Windows 10). The are many assemblers and simulators for the PDP currently available on the Web. Many are macro-assemblers and assume you will be actually running the (linked) code on actual PDP hardware. After quite a bit of searching I came across the "AsmPDP" assembler by J.G. Harston. Its quick easy to use and well documented. Best of all it produces a self contained binary file output which you can use XModem (see below) to load on to your S100 system. There is no assumption of linkers, libraries or PDP hardware. Its a remarkable piece of software! It has one quirk however. Because it is written in (of all things) BASIC, it gets confused if you use some predefined Basic words. Things like "PRINT" for a name. The easiest way out is to use lower case. "Print" for example is OK. Other than that I think you will like it.

Before I discuss the PDP_MON itself let me describe the mechanics of how I do things. You may have a different/better way.
The PDP assembly source files always have a ".mac" extension. So our monitor is PDP_MON.mac. I write/edit the program(s) in Microsoft's Visual Studio and save it.
Within the same folder as PDP_MON.mac is a batch file called PDP.bat that contains a line:-

```
AsmPDP  PDP_MON.mac  PDP_MON.bin  PDP_MON.lst
```

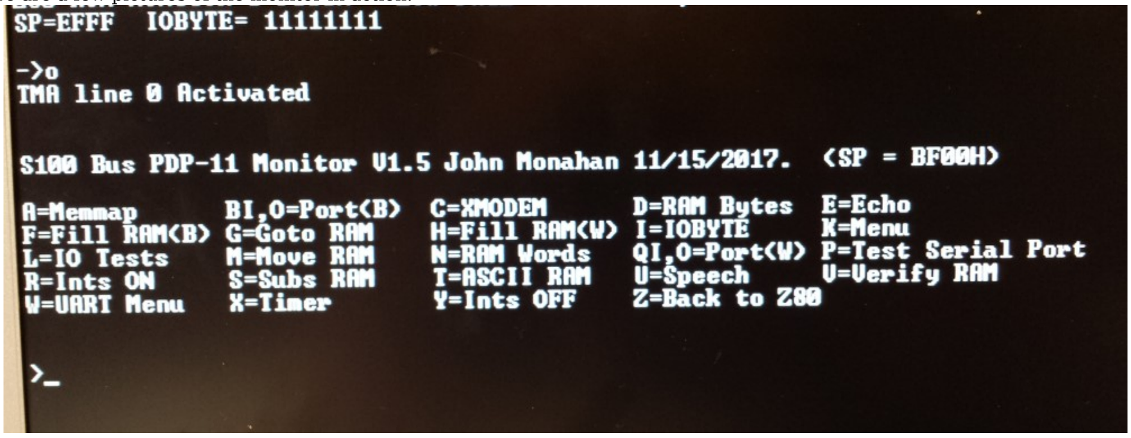
On the Windows 10 desktop I have an Icon which contains only the line
@%comspec%
This brings up a CMD prompt/window where if I type PDP the above assembler will produce

PDP_MON.bin and
PDP_MON.lst

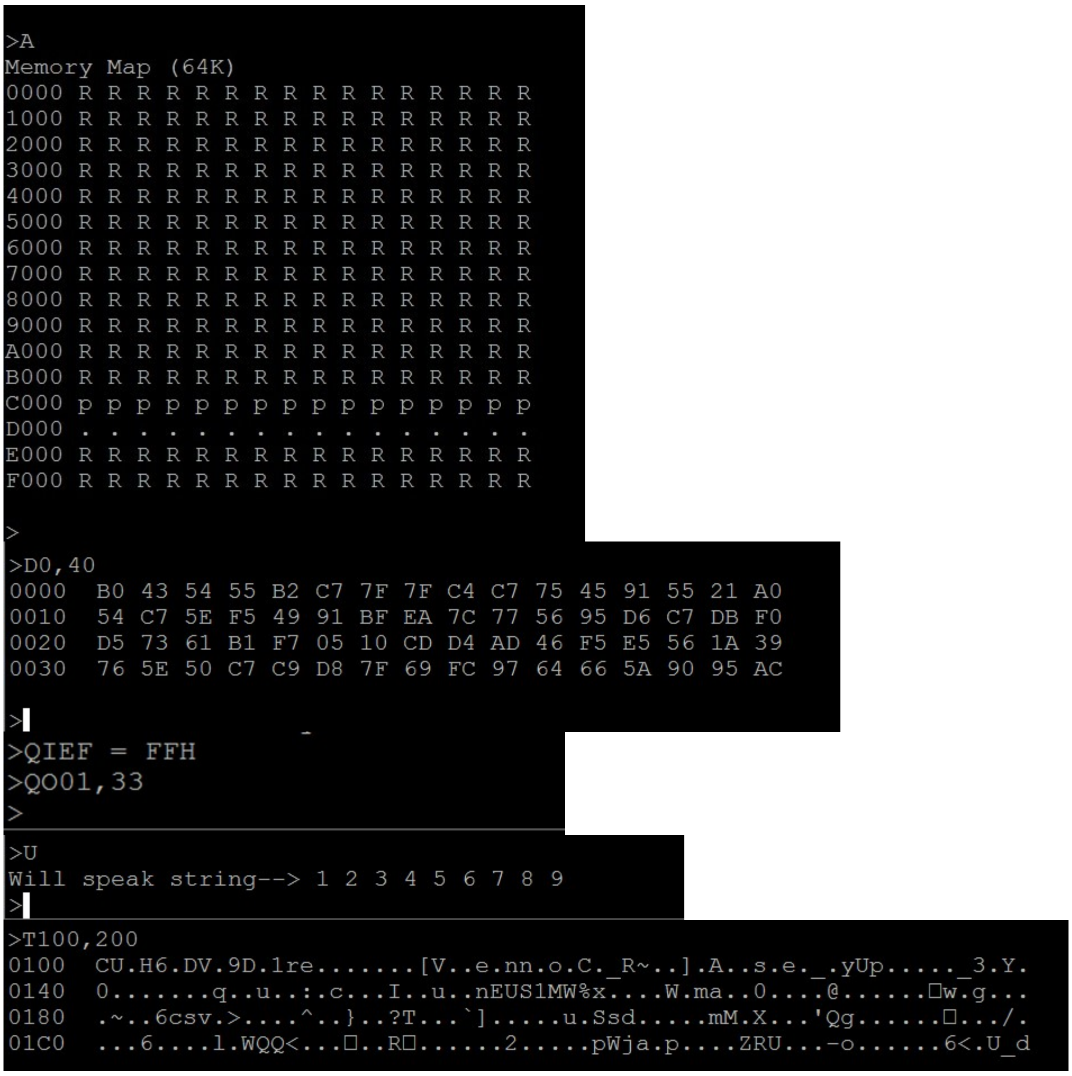
You can then either burn the PDP_MON.bin file into two EEPROMS or, via XModem (see below) load it into low RAM for execution. In all case be sure you have the correct ORG in your source code. Also remember the PDP reserves RAM from 0H to 100H for interrupts and traps. Your code should never reside below 100H. (like CPM !!!!). I use 1000H.

The actual monitor is currently very simple with the basic and familiar commands to read RAM, map RAM, fill RAM, move RAM, Query/Set ports etc. If you have used the Z80 Master or 8086 monitors formats so you will be completely familiar with it. You have two console output options. This board sneaks in an unused status bit in the ODT monitor (bit 0 of port 3FFF70) to decide where to send/read from the console. If jumper K10 is set 2-3 all the PDP_MON I/O will go through the UART serial line to your PC. If K10 is jumpered 1-2 all PDP-MON I/O will go to the Propeller driven Console I/O board. In this latter case the D9 LED will be lit up. It goes without saying you can easily modify the code for a different console I/O hardware setup. Also you may like to put in a small two position switch on the top of the board instead of the K10 jumper.

Here are a few pictures of the monitor in action:-



```
SP=EFFF   IOBYTE= 11111111

->o
TMA line 0 Activated


S100 Bus PDP-11 Monitor V1.5 John Monahan 11/15/2017.  (SP = BF00H)

A=Memmap      BI,O=Port(B)  C=XMODEM      D=RAM Bytes   E=Echo
F=Fill RAM(B) G=Goto RAM    H=Fill RAM(W) I=IOBYTE      K=Menu
L=IO Tests    M=Move RAM    N=RAM Words   QI,O=Port(W)  P=Test Serial Port
R=Ints ON     S=Subs RAM    T=ASCII RAM   U=Speech      V=Verify RAM
W=UART Menu   X=Timer       Y=Ints OFF    Z=Back to Z80


>_
```



```
>A
Memory Map (64K)
0000 R R R R R R R R R R R R R R R R
1000 R R R R R R R R R R R R R R R R
2000 R R R R R R R R R R R R R R R R
3000 R R R R R R R R R R R R R R R R
4000 R R R R R R R R R R R R R R R R
5000 R R R R R R R R R R R R R R R R
6000 R R R R R R R R R R R R R R R R
7000 R R R R R R R R R R R R R R R R
8000 R R R R R R R R R R R R R R R R
9000 R R R R R R R R R R R R R R R R
A000 R R R R R R R R R R R R R R R R
B000 R R R R R R R R R R R R R R R R
C000 p p p p p p p p p p p p p p p p
D000 . . . . . . . . . . . . . . . .
E000 R R R R R R R R R R R R R R R R
F000 R R R R R R R R R R R R R R R R

>
```

```
>D0,40
0000  B0 43 54 55 B2 C7 7F 7F C4 C7 75 45 91 55 21 A0
0010  54 C7 5E F5 49 91 BF EA 7C 77 56 95 D6 C7 DB F0
0020  D5 73 61 B1 F7 05 10 CD D4 AD 46 F5 E5 56 1A 39
0030  76 5E 50 C7 C9 D8 7F 69 FC 97 64 66 5A 90 95 AC

>
```

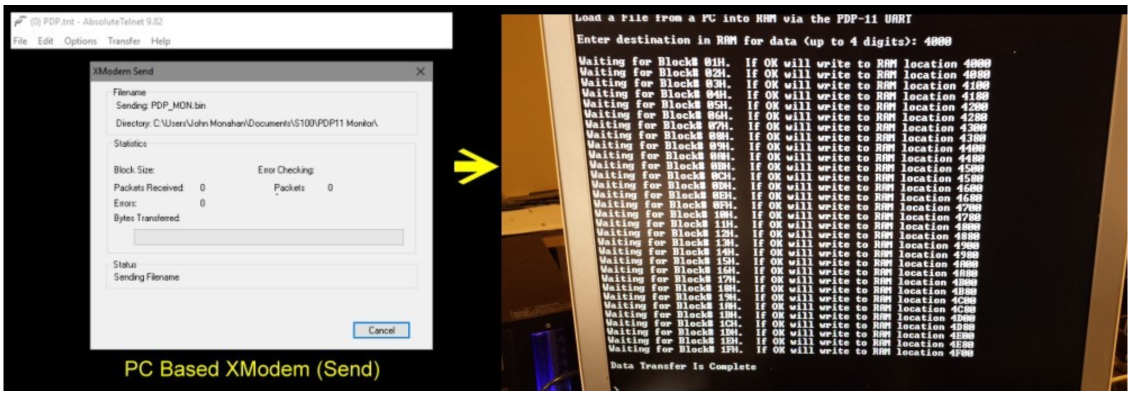
```
>QIEF = FFH
>QO01,33
>
```

```
>U
Will speak string--> 1 2 3 4 5 6 7 8 9
>
```

```
>T100,200
0100  CU.H6.DV.9D.1re.......[V..e.nn.o.C._R~..].A..s.e._.yUp....._3.Y.
0140  0.......q..u..:.c...I..u..nEUS1MW%x....W.ma..0....@......□w.g...
0180  .~..6csv.>....^..}..?T...`].....u.Ssd.....mM.X...'Qg......□.../.
01C0  ...6....l.WQQ<...□..R□......2.....pWja.p....ZRU...-o......6<.U_d
```

Quite a bit of code has been devoted to downloading an XModem compatible file from a PC over the serial line. While you could in theory write the XModem code to share the CPU ODT console I/O for downloads at the same time, the easiest way to do it is to redirect the PDP_MON console I/O to the S100 bus Propeller console IO board and use the onboard UART (U6) to take up the XModem data being downloaded from the PC. The process is exactly the same as the XModem transfer in our Z80 MASTER monitor.

First on the PC side I use the file transfer program Absolute Telnet to send the file from the PC. On the PDP side after entering the "C" command you set the RAM location where you want the binary file placed. Here is a picture of a session:-


(0) PDP.tnt - AbsoluteTelnet 9.82
File Edit Options Transfer Help
XModem Send
Cancel
PC Based XModem (Send)

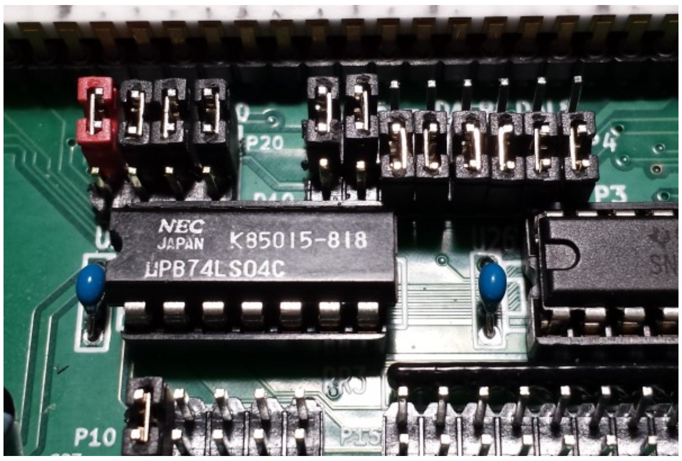
Filename
Sending: PDP_MON.bin
Directory: C:\Users\John Monahan\Documents\S100\PDP11 Monitor\
Statistics
Block Size:
Error Checking:
Packets Received: 0
Packets 0
Errors: 0
Bytes Transferred:
Status
Sending Filename

```
Load a File from a PC into RAM via the PDP-11 UART

Enter destination in RAM for data (up to 4 digits): 4000

Waiting for Block# 01H.  If OK will write to RAM location 4000
Waiting for Block# 02H.  If OK will write to RAM location 4080
Waiting for Block# 03H.  If OK will write to RAM location 4100
Waiting for Block# 04H.  If OK will write to RAM location 4180
Waiting for Block# 05H.  If OK will write to RAM location 4200
Waiting for Block# 06H.  If OK will write to RAM location 4280
Waiting for Block# 07H.  If OK will write to RAM location 4300
Waiting for Block# 08H.  If OK will write to RAM location 4380
Waiting for Block# 09H.  If OK will write to RAM location 4400
Waiting for Block# 0AH.  If OK will write to RAM location 4480
Waiting for Block# 0BH.  If OK will write to RAM location 4500
Waiting for Block# 0CH.  If OK will write to RAM location 4580
Waiting for Block# 0DH.  If OK will write to RAM location 4600
Waiting for Block# 0EH.  If OK will write to RAM location 4680
Waiting for Block# 0FH.  If OK will write to RAM location 4700
Waiting for Block# 10H.  If OK will write to RAM location 4780
Waiting for Block# 11H.  If OK will write to RAM location 4800
Waiting for Block# 12H.  If OK will write to RAM location 4880
Waiting for Block# 13H.  If OK will write to RAM location 4900
Waiting for Block# 14H.  If OK will write to RAM location 4980
Waiting for Block# 15H.  If OK will write to RAM location 4A00
Waiting for Block# 16H.  If OK will write to RAM location 4A80
Waiting for Block# 17H.  If OK will write to RAM location 4B00
Waiting for Block# 18H.  If OK will write to RAM location 4B80
Waiting for Block# 19H.  If OK will write to RAM location 4C00
Waiting for Block# 1AH.  If OK will write to RAM location 4C80
Waiting for Block# 1BH.  If OK will write to RAM location 4D00
Waiting for Block# 1CH.  If OK will write to RAM location 4D80
Waiting for Block# 1DH.  If OK will write to RAM location 4E00
Waiting for Block# 1EH.  If OK will write to RAM location 4E80
Waiting for Block# 1FH.  If OK will write to RAM location 4F00

Data Transfer Is Complete
```

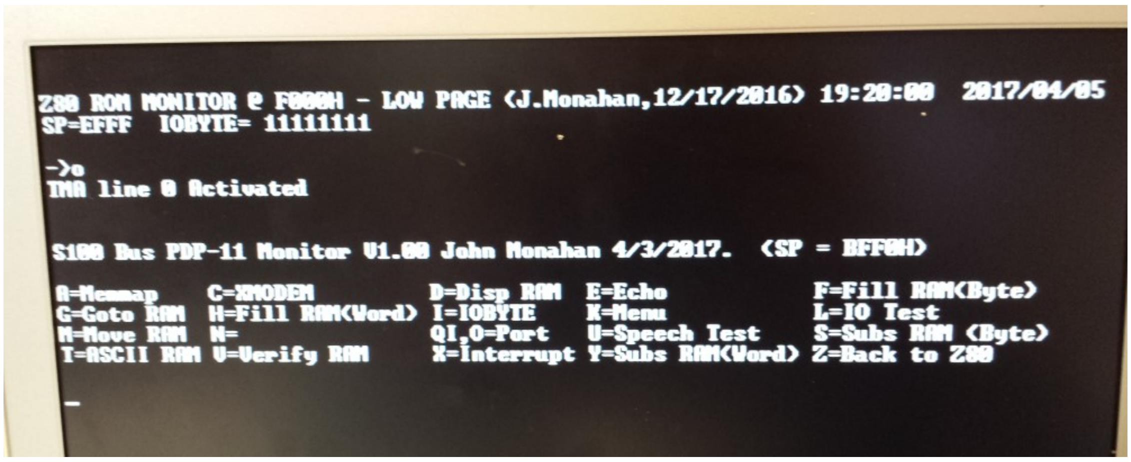
Please note the free Telnet program Tera Term now also works. Remember we use the onboard UART at 9600 Baud (2 stop bits) for file transfers. If you wish to use the S100Computers Serial I/O board (at 38400 Baud, 1 Stop bit) download a file with the Z80 Master monitor and them transfer control to the PDP11 CPU. Use the Z80 monitor XH the C command, then the O command.
The P command allows you to test the Serial I/O board Zilog UART but the is insufficient ROM available for a second XMODEM module.

Lets go back to our power up circuit. Upon power-up you should, (after a short delay), you will see a pulse on U35 & U31 pin 1. As we explained above by jumpering DAL 2,1 to high and DAL15-DAL9 to 1100000 we force the PDP on power-up to C000H in our onboard EEPROM rather than the ODT console. However the CPU after power up will be held in a continuous wait state until it retains bus control (See cont* in the CPLD code).

Within that EPROM code the jumper/switch K10 decides if the console is your S100 bus or the UART. Note you may have to issue a few CR's at the start to clear the status bits. I want to thank Peter Schranz in Switzerland for greatly helping me get the power-up circuit working correctly. Here is a close-up picture of the jumpers for this part of the board.



Here is a picture of a signon directly to the S100 bus using the Z80 "O" command, (no UART connection).



```
Z80 ROM MONITOR @ F000H - LOW PAGE (J.Monahan,12/17/2016) 19:20:00  2017/04/05
SP=EFFF  IOBYTE= 11111111

->o
TMA line 0 Activated


S100 Bus PDP-11 Monitor V1.00 John Monahan 4/3/2017.  (SP = BFF0H)

A=Memmap    C=XMODEM        D=Disp RAM  E=Echo          F=Fill RAM(Byte)
G=Goto RAM  H=Fill RAM(Word) I=IOBYTE   K=Menu          L=IO Test
M=Move RAM  N=              QI,O=Port   U=Speech Test   S=Subs RAM (Byte)
T=ASCII RAM V=Verify RAM    X=Interrupt Y=Subs RAM(Word) Z=Back to Z80

_
```

ROM Inactivation

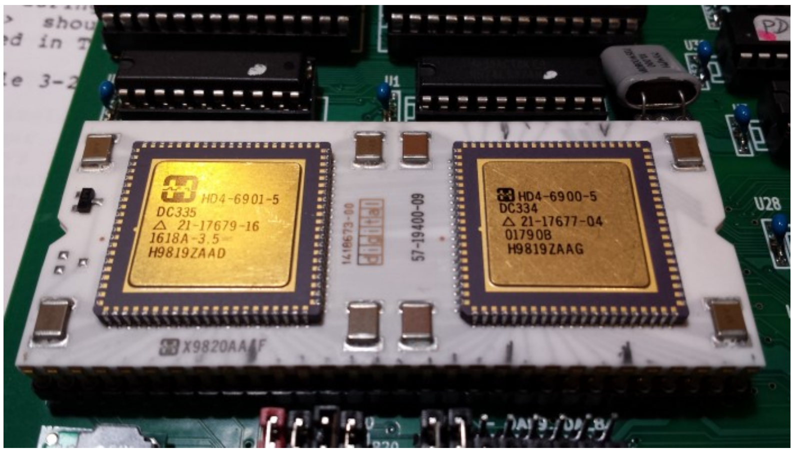
We have one further requirement, OS systems like RT11 assume they have RAM from 0 - E000H upon their initialization and during their memory management configuration process. Before transferring control to the OS in RAM after its loading, we must inactivate the boards onboard ROM's at C000H-DFFFH. We squeeze an extra function into GAL0 such that inputting from port 0ECH (at any time), will send a single negative going pulse out on its pin 18 (SHADOW_ROM*). This signal is picked up by the CPLD and used to inactivate ROM_CS*

```
reg3.d = 'b'1;
reg3.ck = !SHADOW_ROM;                          /* SHADOW_ROM comes from pin 18 of PDP_GAL0 */
reg3.ar = !MASTER_RESET # !TMAx;
ROM_ACTIVE = reg3;

!ROM_CS = !ROM_ADDRESS & !ROM_ACTIVE & !XFERII;       /* Onboard ROMs CS* (Pin 20 of 28C64's) */
```

Obviously you must not input port ECH while running the ROM based monitor.

CPU Speed

Below is a picture of one of the three PDP J11 chips I have. I interpret the 3.5 to mean the clock speed is 3.5 MHz. Perhaps this is wrong. In any event, I find that these PDP-11 CPUs will run fine with a 12MHz crystal. Higher speeds (e.g.. 14.3848 MHz) seem reliable but the CPU has difficulty going to the S100 bus signon on startup. This then is the fastest PHI clock signal we have for an S100 bus CPU board to date. (BTW some reason the power on jump circuit to the ROM requires a faster speed than 8 MHz).



Note the CPU acknowledges the S100 bus Wait request lines XRDY or RDY (U34B) and will issue a stretch signal to the CPU *via* the CPLD pin 73 (cont*). With this wait state capability and our SMB board you can even stop/pause the CPU, single step and set hardware breakpoints etc. (as we done have for our other CPU's), on any S100 bus cycle. The data (in HEX) for the current address lines and data lines will be valid and displayed on the SMB board for each step -- there was no data queue with this early CPU.

Finally, you can also increase the Baud rate for the UART to at least 38400 Baud (see the CD4 output lines in the CPLD code). I have only tested this using a USB serial adaptor interface.

The PDP-11 Board Jumper Table

There are a fair number of jumpers on this CPU board and they need to be set carefully. To run the board as a bus slave (with a Z80 master) here is a typical jumper setup:-

Jumper	Function
JP5	Normally closed. Generate the S100 bus MWRT signal on this board when it is active.
JP1	Normally closed. Generate the S100 bus 2MHz Clock signal on this board when it is active.
P13,P14	These jumpers define the Baud rate, parity and stop bits for the UART. See above for a picture
P11	Jumpers to accommodate RAM or ROM chips. For RAM J11 1-2. For EEPROM's leave open.
JP4,JP5, JP6,JP7	Normally all open unless the board is a bus master.
JP13,JP15	Normally closed to utilize the S100 bus Phantom Line. Normally no need for JP15
K2 & K3	RAM/ROM OE* & WR* signals. For RAM jumper both 2-3. For EEPROM jumper both 1-2.
K1	Normally 1-2
K9	Normally allow S100 bus wait states with 1-2, During build use 2-3.
K7	UART Select, Set 1-2 for core circuit testing. Normally 2-3
K5	For slave mode 1-2
JP9	Jumper after the CPLD is programmed
K6,JP12	Set 1-2 and jumper JP12
P8	Set 1-2 and jumper JP13
P9 or P17	CPLD JTAG programming socket. For Rockfield Research 1508 programmer use P5. Pin 1 is bottom left.
P22	Spare input CPLD pin
K3 & K4	XFERI & XFERII, In slave mode, 2-3
P10	S100 Bus TMA line to activate board. Normally 1-2 (or 3-4) closed.
JP19,JP20,JP18,JP20	Jumpers to patch connections to extra S100 bus lines. Normally all unconnected
P22	Inputs from S100 bus Interrupt vectors. Normally all open.
K10	This can be a mini-switch or jumper to determine where the PDP_MON monitor data I/O is sent
P2	This a serial port connector for the UART with a pinout like our Serial board. If the DB9 socket is used it must be jumpered as described above.
JP8, JP11, JP10	Normally all open. Used only for a bus master

PDP-11 S100 Board Parts

The PDP-11 chip is not available from most vendors. It is found on eBay -- either as the chip alone see here or it can be pulled from Digital boards. It is almost always socketed on those boards. A typical eBay price is $100/chip. Currently (4/10/2017), Paul Birkel is doing a chip group purchase -- see the Forum group below. Most of the other components are common. I usually get them from obtained from Anchor Electronics and Jameco. Note the U17, U20, U22, U23 and U27 and U16 require 74LSxxx chips. 74xxx or 74Fxxx don't seem to work as well.

PDP-11 CPU Board Bugs & Notes.

This board will be the basis of a hardware portfolio of a number of boards that hopefully one day will allow us to run Digital PDP-11 operating system(s) on the S100 bus. This is not going to be easy. The PDP-11 bus systems while well documented, differ in many hardware respects to the S100 bus. If you just want to use the PDP-J11 chip and this board to write your own custom software you have here what you need. The challenge is to design the circuit hardware so the Digital software does not know it is not in a digital bus. I fully anticipate a few versions of our PDP boards. If you detect a hardware anomaly please let me know so I can address it in the next version.

The next version will have...

1. The PDP CPU's treat addresses in the range 3FE000H to 3FFFFFH as I/O ports (as we discussed above). This is a total port range of 0000H to 1FFFH. The current boards range overlaps 0000 to 0FFFH and 1000 to 1FFFH as the same on the S100 bus. The address line LA12 should not go into U29. (The patch is, bend out pin 4 of U029 and on the back of the board jumper pin 12 of U1 to pin 2 of U32). This converts the CPU addresses 3FFE00H to 3FFFFFH to S100 bus 16 bit ports 0H to 1FFFH without modifying any PDP-11 code. It's probably unlightly the address overlaps are an issue but its good to be safe.

2. The PDP treats its "DL11-W" event timer/serial board in an unusual way in terms of how interrupts are handled. The timer "events" are handled differently than "normal" interrupts. The signal comes into the CPU on pin 8 (EVENT*). The CPU requires a dedicated timer event acknowledge signal on the bus going back to the timer/serial board. Currently I will use the NDEF3 spare S100 line (coming out of the CPLD pin 60). However the NDEF3 and NDEF1 spare lines were intended to handle the situation where the PDP-11 is itself a slave, and will pass control to another slave/DMA controller on later boards. This was done for example on our 80286 CPU board. Again an obscure issue, nevertheless the next version of the PDP-11 CPU board will utilize another S100 bus spare line.

This V2 board is currently in its prototype stage.

Bugs detected so far...

For some reason there are traces from pins 10 & 11 of U19 to pins 2 & 4 of U8 that should not be there. Please bend out from the socket pins 10 & 11 of U19.

U18 in the schematic is listed as a 74F367. I find at low clock speeds (~3MHz) this chip leads to random characters on the S100 bus console. It seems fine at the 10MHz speed. I recommend however you replace the U18 chip with a 74LS367 an 74367 or an old 8T97.

The Reset Signal.

Since laying out this board I now have a much better understanding of the PDP J11 CPU. With the early CPLD software, system power up and with the Z80 "O" command the PDP-11 always took control of the bus fine. However if you hit reset and enter the "O" command a second time it did not hand over control back to the PDP-11.

After a careful study of key signals and reading in detail the Digital technical documents I have now solved this issue. First its important to appreciate that the HALT signal (which is active HIGH) is somewhat misnamed. HALT will ALWAYS force the J11 into its ODT onboard monitor mode -- no exceptions. If it is held high you can never get out of that state, so something like 140000G will NOT jump to the onboard ROM/monitor. With HALT high you can examine registers, RAM etc. however. What you normally need to do is pulse HALT high. This drops you into the ODT mode from which (amongst other things) you can jump to your onboard ROM/Monitor.

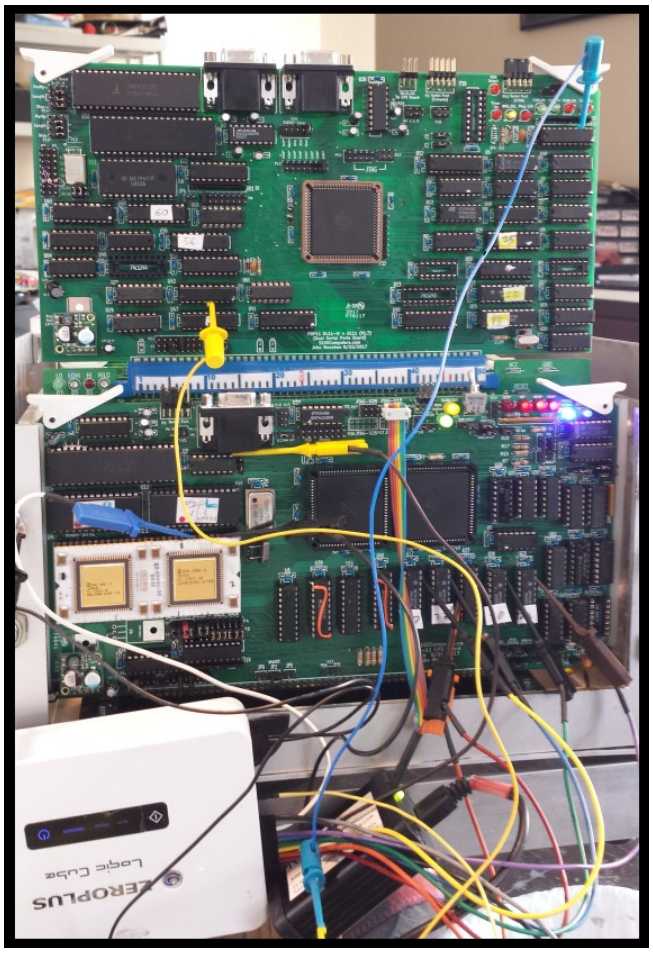
The INIT* line is active LOW. When the J11 gets control of the bus (XFERI & XFERII go low), IF (and only if) INIT* pulses low will the Power on circuit described above become active. The various DAL0-15 lines are read (one time) and the J11 decides what to do. Normally it would be to boot to our onboard ROM. However with the appropriate jumper settings it can be forced to jump into its ODT mode. Since HALT is low from there 140000G will bring you to your onboard monitor. If HALT is high the J11 will never activate the power on circuit -- so in the normal configuration of this board (once it is built/debugged) HALT will always be LOW.

Because of the above, I have updated the CPLD code for this board. Also the GAL2 code needs to be reburnt. See below

On this board INIT* is pulsed low with MASTER_RESET* (JP22 closed). This seems to work fine. Better, and on our future V2 board, INIT* is pulsed low after the J11 has bus control. The updated CPLD code below does this, but to use this extra (nice to have function), you need to remove the JP22 jumper and jumper pin 2 of JP19 to pin 1 of JP22.

Remember there is an S100_ONLY equate in the PDP11 ROM Monitor. Be sure it is set to false if you want the above SW1/ODT switch option to work.

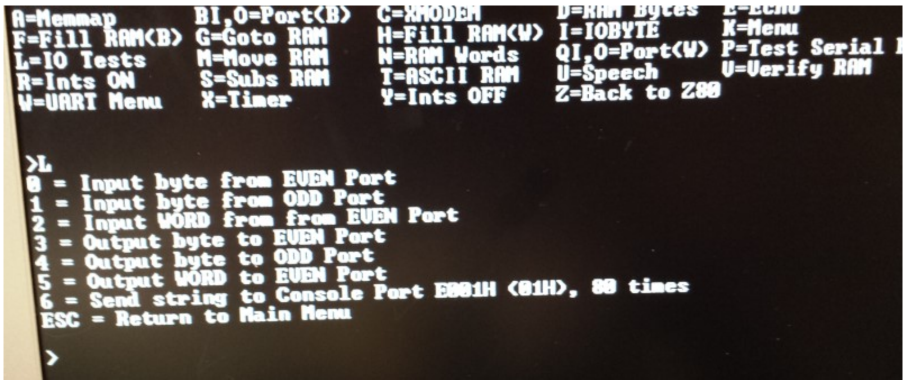
Outputting Data to 8 & 16 bit I/O Ports.
Since laying out this board I have been working on a V2 board as well as a "PDP11 Support Board". A very difficult issue is building a system which exactly emulates how Digital OS's handle interrupts and the exact hardware configuration they expect. This involves not only a careful reading of the (well documented) Digital manuals but a careful analysis of some critical bus and CPU signals. It involved quite a bit of time with signal analyzers. FYI, here is an example:-



More on all this at a later date. During this work I noticed that with the current V1 board CPLD code people are using there was a problem reading (byte) data from odd numbered ports. The PDP-J11 in fact never reads bytes. It reads them as words and internally selects the upper or lower byte. (It does write bytes however). On the IEEE-969 S100 bus, memory cards signal they are able to respond to 16 bit requests (sXTRQ* low) by returning the SIXTN* signal low. All our S100Computers RAM boards do this. However few S100 bus I/O ports are 16 bits wide -- in fact the older 70's ones don't even have a SIXTN* signal (The line will always be high). So when the J11 CPU sends out a (byte) read from an odd numbered port it actually appears on the bus as a Word port read. The old CPLD software did not take into account that the actual port is only 8 bits and in fact the data appears on the DAL0-DAL7 via U22 & U23. Since it is an 8 bit read, only the U22 input DI0-DI7 has valid data. The U23 buffer is also looking at the DO1-DO7 lines and presumably is seeing FF's. Amazingly the above situation works with every S100 bus board I tried. However it is not really correct.

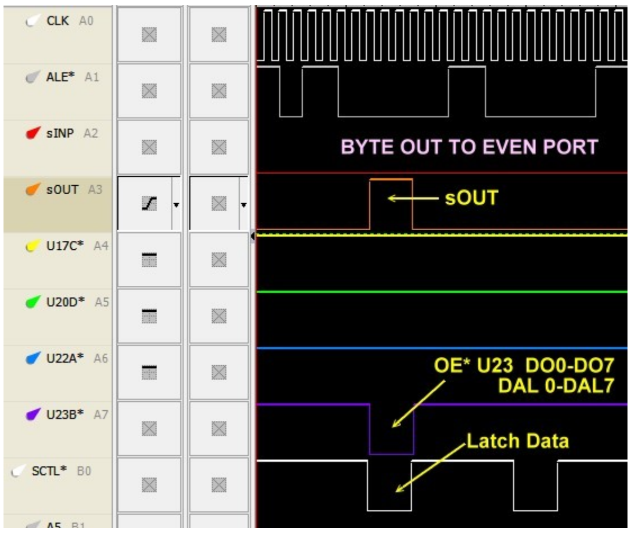
For S100 bus Odd port inputs from 8 bit wide ports the CPLD needs to know it is a NOT a 16 bit port (SIXTN* high) and reroute the input from the DI0-DI7 S100 bus lines to the J11 CPU DAL8 - DAL15 lines. This is done using U17.

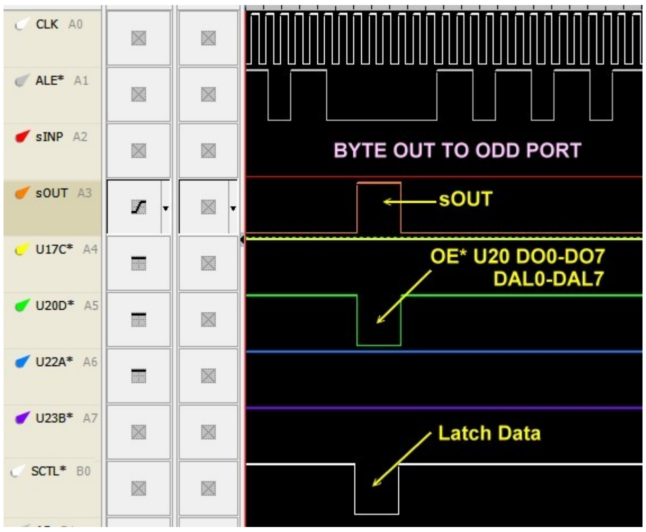
Only when I was doing the PDP11 Support board where there are if fact 16 bit wide ports did the above errors show up. This scare forced me to write some routines into our PDP11 Monitor that have simple loops to analyze the various combinations. I'm also updating the monitor (see below). There are some options in that monitor code you can currently ignore until the Support Board arrives. Here is the signon menu:-

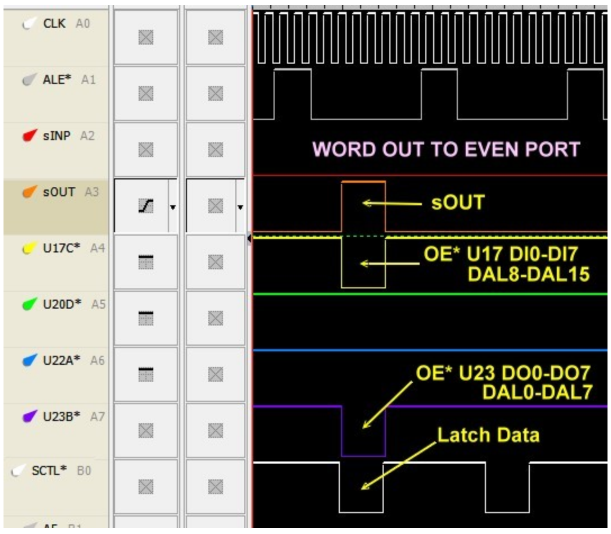


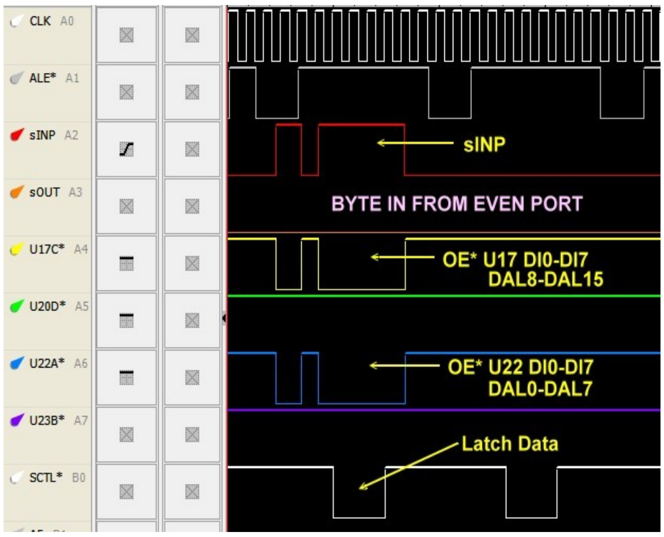
```
A=Memmap      BI,O=Port(B)  C=XMODEM      D=RAM Bytes   E=Echo
F=Fill RAM(B) G=Goto RAM    H=Fill RAM(W) I=IOBYTE      K=Menu
L=IO Tests    M=Move RAM    N=RAM Words   QI,O=Port(W)  P=Test Serial I
R=Ints ON     S=Subs RAM    T=ASCII RAM   U=Speech      V=Verify RAM
W=UART Menu   X=Timer       Y=Ints OFF    Z=Back to Z80

>L
0 = Input byte from EVEN Port
1 = Input byte from ODD Port
2 = Input WORD from from EVEN Port
3 = Output byte to EVEN Port
4 = Output byte to ODD Port
5 = Output WORD to EVEN Port
6 = Send string to Console Port E801H (01H), 80 times
ESC = Return to Main Menu

>
```

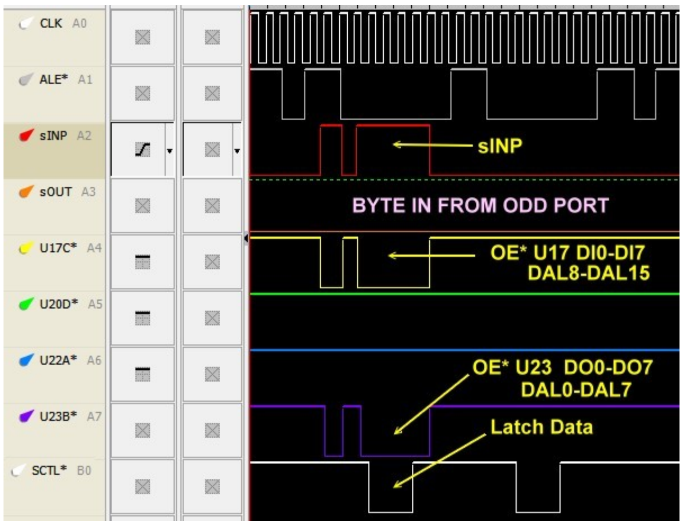
Here is a detailed signal analysis of all the Port I/O combinations using the various monitor options:-

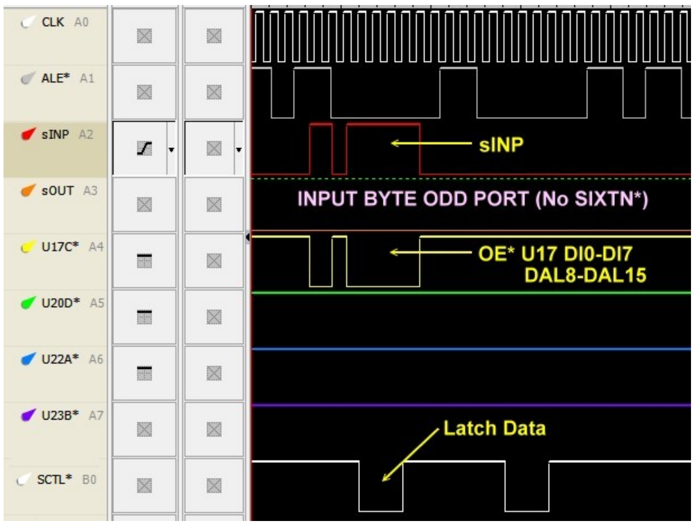


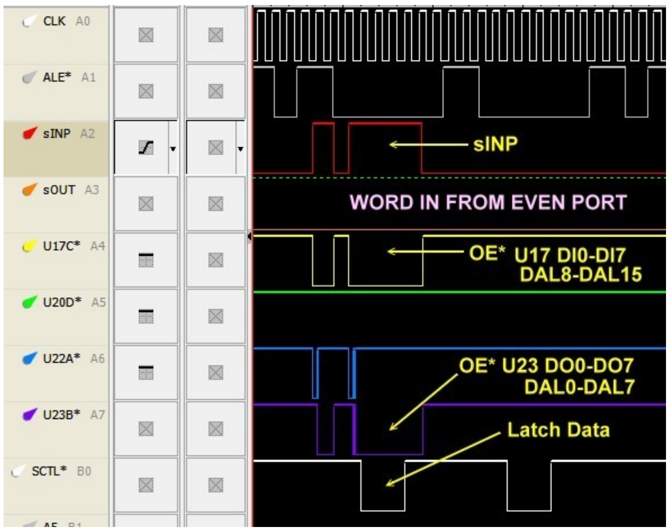












The critical part of the update CPLD code is:-

```
!OE_C = (((!bpWR & !BUS_WORD_WRITE & !XFERII & !bsINTA) # (bpDBIN & !XFERII & !bsINTA))     /* 16 Bit Rd or /Wr and 8 bit Rd,
DAL 8-15 */
        #(bsINTA & !XFERII & !bsINTA));

!OE_D = !bpWR & !BUS_BYTE_WRITE & LA0 & !XFERII & !bsINTA;                                  /* 8 Bit Write Odd address on DAL
8-15 */


!OE_A = ((bpDBIN & !LBS_IO & !LA0 & !XFERII) & !bsINTA & SIXTN);                           /* 8 Bit I/O Read on DAL 0-7 */


!OE_B = (((!bpWR & !BUS_WORD_WRITE & !XFERII & !bsINTA) # (bpDBIN & !XFERII & !bsINTA & !SIXTN)) /* 16 Bit Rd or Wr and 8 bit
(RAM) Rd */
        # (!bpWR & !BUS_BYTE_WRITE & !LA0 & !XFERII & !bsINTA));                             /* 8 Bit Write Even address (DAL
0-7) */
```

There are a few other small tweaks as well. **So please update your CPLD code** -- see below.

Also remember if you are using the onboard port (EDH) to activate the board be sure you jumper P36 1-2 & 3-4. This is because other boards may be expecting the TMA0* line to be low when this board is active. The onboard port alone will not lower the TMA0* line. This article describes the V1.0 version of the board. The board made available to users (see below) is V1.01a. It is essentially the same board with some very minor part rearrangements and silkscreen changes. (In particular Pin 21 of the UART utilizes the inverted S100 Reset* signal rather than a PO Cap discharge circuit).

S100 Bus Address and Data lines displayed on the SMB

The monitor ROM on the PDP11 CPU board does not access the S100 bus address lines while it is running. All the code there resides on the PDP11 board. The only active S100 bus address lines are above 0FFFFH thus you see 3F0000H on the HEX displays of the SMB. To prove this if you reassemble the monitor code to an ORG of 1000H (ROMS EQU FALSE in the code). Upload the file .bin from your PC to 1000H in RAM (Z80 XH command followed by C command) , transfer control to the PDP with the Z80 "O" command and the within the PDP monitor type G1000, The PDP will run the monitor at 1000H in RAM and all the address lines are displayed. Of course any PDP running program on S100 bus RAM will display all the address lines.

Likewise within the PDP monitor, most actual data in/out on the S100 bus with the code there is byte wide so normally only the lower 8 bits of data on the SMB HEX displays light up. Again to prove this, if you fill RAM with a word (Monitor "H" command) you will see the full 16 bit data bus light up i.e. all 4 HEX displays. Since the monitor code is in ROM on the CPU board the data lines will not be activated for monitor code reads.

This board is clearly only the tip of the iceberg to building a Digital PDP-11 system on the S100 bus. Next up will be a "PDP-11 Support Board" along the lines we have for our MSDOS Support board. The long term goal being to have a system to run an early Digital PDP-11 OS. Also I have not addressed programming the chip to use RAM above 64K. Everything is there to do so. I'm hoping others will provide examples.

Realizing that a number of people might want to utilize a board like this a "group purchases" was opened on Google Groups S100Computers Forum. Join the group and add your name to the list if you would like a bare board if a second later batch of boards is done. Boards are $18/board. A group purchase of the PDP-J11 chip was also run. See the forum for more details. Do not e-mail me directly for any requests.

Finally please note, currently (July 2017), the PDP-11 monitor is being actively updated. Please go to the PDP-11 Software page for the most current update.

For the PDP11 Monitor software, please note there are various equates at the start of the monitor code that you need to adjust depending on your need (S100_ONLY, for example).

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.

DC-11 User Guide	(2/13/2017)	
PDP-11 Programmers Card	(4/5/2017)	
AsmPDP Assembler (Zip File)	(4/5/2017)	(From http://mdfs.net/Software/PDP11/Assembler)

PDP11 Monitor

PDP-11 Board KiCAD Schematic Pdf file	(V1.1a 5/9/2017)
PDP-11 CPLD Code (PLD file)	(V 1.11 12/30/2017)
PDP-11 CPLD Code (Zip file)	(V 1.11 12/30/2017)
U25, U8 & U30 GALs WinCUPL Code (Zip file)	(V 1.01 12/30/2017)
PDP-11 KiCAD Files Folder	(V1.1a 5/9/2017)
Gerber Files Folder	(V1.1a 5/9/2017)
PDP-11 BOM (XLS File)	(V1.1a 5/9/2017) (Supplied by Rick Bromagem)
PDP-11 BOM (Pdf File)	(V1.1a 5/9/2017)

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

This page was last modified on 12/31/2017