The Comprehensive Guide To The Atari Parallel Bus

WORK IN PROGRESS
v 0.91

By Brian E. Reifsnyder

Table of Contents

Forward
Atari 600XL/800XL Parallel Bus Interface
Atari 130XE/800XE Parallel Bus Interface
Atari 1090 XL Expansion System
Timing
Reserved PBI Addresses
Third Party PBI Address Use
Parallel Device Handlers and Drivers
Example Parallel Device Source Code
Mechanical Specifications
Publication Errors
Sources

Forward

The Atari Parallel Bus Interface or PBI is a powerful connection to the address and data busses of the Atari computers. While much can be accomplished with this bus, the information that is available is rather involved, scattered over multiple sources and, is sometimes contradictory. This document is an attempt to bring all of that information into one place in order to ease the development of parallel bus hardware.

The Atari Parallel Bus was only available, in some form, on the 600XL, 800XL, 130XE, and 800XE.

I am also including information on the 1090 XL Expansion System as it was designed to use the parallel bus to connect up to 5 cards to the Atari Home Computer.

Atari 600XL/800XL Parallel Bus Interface

PBI Edge Connector Pinout As Looking From Back of Computer

Top

01 03 05 07 09 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49

02 04 06 08 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50

Bottom

1	GND	Ground	Power
2	/EXTSEL	External Select	Input
	Held at +5VDC by computer. Setting this pin low disables the internal DRAM by disabling CAS. Note that setting this pin low will disable the CAS output at pin 41.		
3	A0		Output
	A0 – A15 are address lines set by the CPU and ANTIC only.		
4	A1		Output
5	A2		Output
6	A3		Output
7	A4		Output
8	A5		Output
9	A6		Output
10	GND	Ground	Power
11	A7		Output
12	A8		Output
13	A9		Output
14	A10		Output

15	A11		Output
16	A12		Output
17	A13		Output
18	A14		Output
19	GND	Ground	Power
20	A15		Output
21	D0		Bidirectional
22	D1		Bidirectional
23	D2		Bidirectional
24	D3		Bidirectional
25	D4		Bidirectional
26	D5		Bidirectional
27	D6		Bidirectional
28	D7		Bidirectional
29	GND	Ground	Power
30	GND	Ground	Power
31	Phi2		Output

This is the main system clock. Note that on the PBI Phi2 will be delayed as it is buffered by a 74LS08 AND gate on the system board. A 74LS08 can have a propagation delay time of anywhere from 3 to 18 nanoseconds.

32	GND	Ground	Power
33	Reserved		

Probably the best use for this pin would be to connect it to the /HALT signal so as to allow for external memory upgrades that support ANTIC banking. This is supported starting with the 2022 Reproduction of the 1090 XL Expansion System, R1.

34 Reset

Active low signal which occurs either on power-up or by depressing the reset key on the computer.

35 IRQ Interrupt Request Input

Held at +5VDC by the computer. Setting this pin low will initiate an interrupt request. See the chapter on “Parallel Device Handlers and Drivers”, Section 7.

36 RDY Ready Input

Held at +5VDC by the computer. This signal allows the PBI device to halt the microprocessor only during read cycles. Holding this signal low will extend the read cycle.

37 Reserved

Note: On an XL computer, this pin is not connected. However, on a 1090 XL Expansion system this pin is connected to card select and held low.

38 EXTENB External Enable Output

Set by the Memory Management Unit (MMU) on the system board. When this pin is set low, internal DRAM and CAS are disabled. When this pin is set high, the MMU is enabling the DRAM. Internal to the Atari, this pin supplies the MMU CAS Inhibit signal.

This pin is set low, disabling the internal DRAM, under the following conditions:

Addresses from $D000 to $D7FF are called.
A cartridge is inserted in the left cartridge slot.
(RD5 is high and addresses from $A000 to $BFFF are called.)
A cartridge is inserted in the right cartridge slot. On the XL this can apply to it’s only cartridge slot and is dependent upon the cartridge.
(RD4 is high and addresses from $8000 to $9FFF are called.)
If BASIC is enabled, when addresses are called from $A000 to $BFFF.
If the OS is enabled, an internal OS address is called with the address ranges as follows:

$5000-$5800 (If self test is enabled.)
$C000-CFFF (If OS ROM is enabled.)
$D800-$DFFF (If OS ROM is enabled and Math pack is enabled as per pin 43.)
$E000-$FFFF (If OS ROM is enabled.)

Note that if a PBI address (i.e. D1FF) is called that External Enable will be set low as many PBI addresses are in the range from $D000 through $D7FF. It is therefore important to note, and contrary to some documentation, that External Enable is not necessarily set high when an address used by a PBI device is set on the system bus.

39	Reserved		
40	REF	Refresh	Output

Controlled by the ANTIC chip, this signal can be used for refresh timing on dynamic memories.

This signal is sometimes held low by PBI expansions to prevent the MMU from acting on an address.

Note: Atari did not use this signal for any PBI devices, including the 1064 and 1066 memory expansions.

41	CAS	Column Address Strobe	Output

Column Address Strobe for DRAM Refresh. Disabled when External Select (Pin 2) is set low.

42	GND	Ground	Power
43	/MPD	Math Pack Disable	Input

Held at +5VDC by the computer. Setting this pin low will instruct the Memory Management Unit (MMU) to disable the OS ROM from $D800 to $DFFF. This allows for the mapping of the RAM or ROM of a parallel port device in this location.

Note that Atari specifies that $D800 to $DFFF will contain the parallel port device ROM.

When the Math Pack is disabled, External Enable (pin 38) will be set high when addresses from $D800 to $DFFF are called. Therefore, when mapping parallel port device memory into this location, /EXTSEL (pin 2) will need to be set low.

44	RAS	Row Address Strobe	Output

Row Address Strobe for DRAM Refresh.

45	GND	Ground	Power
46	R/W	Read / Write	Output

Set by the CPU or ANTIC. Signal is high for read and low for write.

47	+ 5 VDC / Reserved	Power	

+5 VDC on Atari 600XL only. Unused on Atari 800XL.

Note that some Atari 800XL's have been modified to provide +5 VDC on this pin.

48	+ 5 VDC / Reserved	Power	

+5 VDC on Atari 600XL only. Unused on Atari 800XL. See pin 47, above.

49	AUDIO	Audio In	Input

This pin is used for a mono audio signal. Any signal on this pin will be amplified by the computer's audio amplifier and output through the TV and/or monitor output on pin 3 of the monitor jack. Atari specifies that this signal be 100mV peak to peak with a 4.7k ohm load impedance.

50	Ground	Power

Atari 130XE/800XE Parallel Bus Interface

The Atari 130XE and 800XE used a combination of the cartridge port and an “enhanced cartridge interface” or ECI to provide an interface that is similar to the XL Parallel Bus Interface. Some significant changes exist, however and are listed below.

Changes between the XL Parallel Bus Interface and the XE Cartridge Port/ECI combination:

Signals added to the XE Cartridge Port/ECI combination:

/D1xx
/HALT

Signals not present in the XE Cartridge Port/ECI combination:

CAS
RAS
RDY
EXTENB

Cartridge Port and Enhanced Cartridge Interface As Looking From Back of Computer

Top

Cartridge Port															Enhanced Cartridge Interface						
A	B	C	D	E	F	H	J	K	L	M	N	P	R	S	A	B	C	D	E	F	H
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	1	2	3	4	5	6	7

Bottom

Cartridge Port:

A	RD4	RAM Disable 4	Input
	When set high, RD4 instructs the MMU to disable the RAM from $8000-$9FFF.		
B	GND	Ground	Power
C	A4		Output
	A0 – A15 are address lines set by the CPU and ANTIC only.		
D	A5		Output

E	A6		Output
F	A7		Output
H	A8		Output
J	A9		Output
K	A12		Output
L	D3		Bidirectional
M	D7		Bidirectional
N	A11		Output
P	A10		Output
R	R/W		Output
	See PBI pin 46.		
S	B02		Output
	See PBI pin 31.		
1	/S4	Select 4	Output
	When set low, /S4 is the chip select for the 8k cartridge ROM located at $8000-$9FFF.		
2	A3		Output
3	A2		Output
4	A1		Output
5	A0		Output
6	D4		Bidirectional
7	D5		Bidirectional
8	D2		Bidirectional
9	D1		Bidirectional
10	D0		Bidirectional

11	D6		Bidirectional
12	/S5	Select 5	Output

When set low, /S5 is the chip select for the 8k cartridge ROM located at $A000-$BFFF.

13	+5 VDC		Power
14	RD5	RAM Disable 5	Input

When set high, RD5 instructs the MMU to disable the RAM from $A000-$BFFF.

15	/CCTL	Cartridge Control	Output

Normally held at +5vdc. If an address of $D500-$D5FF is called, this signal is set low. If a cartridge has the appropriate logic the CPU can read from and/or write to registers on the cartridge.

Enhanced Cartridge Interface:

A Reserved

Probably the best use for this pin would be to connect it to pin 16 of the MMU (U3 / CO61608) so as to provide the missing EXTENB signal required by some memory upgrades. Note that making this connection would eliminate the requirement for a 1090 XE Compatibility Board, for memory cards, starting with the 2022 Reproduction of the 1090 XL Expansion System, R8.

B	IRQ		Input

Held at +5VDC by the computer. Setting this pin low will initiate an interrupt request. See the chapter on “Parallel Device Handlers and Drivers”, Section 7.

C	/HALT	HALT	Output

Controlled by ANTIC. When set low, ANTIC pauses the 6502 CPU and takes control of the address and data bus.

D	A13		Output
E	A14		Output
F	A15		Output
H	GND	Ground	Power

1	/EXTSEL	External Select	Input

Held at +5VDC by computer. Setting this pin low disables the internal DRAM by disabling CAS.

2	RST	Reset

See PBI pin 36.

3	/D1xx	Output

Set by a 74LS138 (U2) on the 130XE system board, this signal is normally high and set low if an address from $D100 through $D1FF is called.

This signal is not to be confused with External Enable (EXTENB).

4	/MPD	Math Pack Disable	Input

Held at +5VDC by the computer. Setting this pin low will instruct the Memory Management Unit (MMU) to disable the OS ROM from $D800 to $DFFF. This allows for the mapping of the RAM or ROM of a parallel port device in this location.

5	AUDIO	Audio In	Input

See PBI pin 49.

6	REF

See PBI pin 40.

7	+5 VDC	Power

Atari 1090 Expansion System

Internal Edge Connector Pinout As Looking From Top of Expansion System

Front Back

50 48 46 44 42 40 48 36 34 32 30 28 26 24 22 20 18 16 14 12 10 08 06 04 02

49 47 45 43 41 39 37 35 33 31 29 27 25 23 21 19 17 15 13 11 09 07 05 03 01

1	+10 VDC		Power

Unregulated +10 VDC. As per Atari's specifications, this voltage may vary by as much as +40% or -15%. Each card should draw no more than 500 milliamps.

2	+10 VDC		Power

See pin 1.

3	/EXTSEL	External Select	Input

See PBI pin 2.

An open collector gate should be used to drive this signal low.

4	AUDIO	Audio In	Input

See PBI pin 49.

5	EXTENB	Buffered External Enable	Output

See PBI pin 38.

6	GND	Ground	Power
7	REF	Buffered Refresh	Output

See PBI pin 40.

Note that this refresh signal is buffered and cannot be used as in "input" to shut down the MMU as described under PBI pin 40.

8	/MPD	Math Pack Disable	Input

See PBI Pin 43.

An open collector gate should be used to drive this signal low.

9	RDY	Ready	

See PBI pin 36

Starting on the 2022 Reproduction of the 1090 XL Expansion System, R8, this pin is held high by the 1090 circuitry. Holding this pin low will disable the EXTENB signal at the transceiver (U7). The purpose of this feature is to permit an XE Compatibility Board to re-create the EXTENB signal, without interference from a floating transceiver pin, in the event the 1090 is connected to an Atari 130XE or 800XE. Holding this pin low will also enable U7's outputs to pins 28 and 30.

This pin was re-purposed because RDY is not available on the 130XE/800XE ECI and there is no evidence Atari used the RDY pin on any of it's 1090 cards.

10	IRQ	Interrupt Request	

See PBI pin 35.

An open collector gate should be used to drive this signal low.

11	AC		

60Hz reference only.

12	RST	Buffered Reset	Output

See PBI pin 34.

13	Card Select		

Held low by the 1090 XL Expansion System. Atari planned this line to be an address select line for processors that could address more than 64k of memory.

Connected to pin 37 of the PBI.

No use of this line is known to exist.

14		Address Buffer Select	1090 Input

Enabled by default and is normally held low by the 1090 circuitry. A 1090 card can disconnect the 1090 address bus from the computer by holding this high. This does not disconnect the address bus between 1090 cards.

15	R/W	Buffered Read/Write	Output

See PBI pin 46.

16	-12 VDC		Power

Expansion cards should use no more than 30 milliamps as per Atari's specifications.

Note: The only known use for this pin is shown on the schematic for the Parallel/Serial Interface Card.

17	Buffered A15		Output

See PBI Pin 3 for more information on the address lines. All buffered address lines can be turned off as per the Address Buffer Select on pin 14.

18	GND	Ground	Power
19	A14	Buffered A14	Output
20	Phi2	Buffered Phi2	Output

See PBI Pin 31. Note that Phi2 is already buffered by the 74LS08 in the computer. The buffering of Phi2, on the 1090, adds an additional delay.

21	A13	Buffered A13	Output
22	GND	Ground	Power
23	A12	Buffered A12	Output
24	+5V Reference	Reference	

Reference only, not to be used for power. Expansion cards should draw no more than 10 mA.

25	A11	Buffered A11	Output
26	Reserved		

No connection.

On the original 1090 this pin was not connected. Starting on the 2022 Reproduction of the 1090 XL Expansion System, R1, this pin is a buffered /HALT signal from either an Atari 130XE or an Atari XL. If it is desired that the /HALT signal be available on an

Atari XL, an Atari 600XL or 800XL will need to be modified so as to have an internal jumper wire connecting the /HALT signal to pin 33 the PBI.

27	A10	Buffered A10	Output
28	Reserved		

Connected to pin 39 of the PBI on a standard 1090 Board.

Starting on the 2022 Reproduction of the 1090 XL Expansion System, R8, this pin is used for the RD4 signal from an Atari 130XE/800XE only if pin 9 is held low. If pin 9 is not held low, this pin is disconnected.

29	A9	Buffered A9	Output
30	Reserved		

Connected to Pin 47 of the PBI.

Starting on the 2022 Reproduction of the 1090 XL Expansion System, R8, this pin is used for the RD5 signal from an Atari 130XE/800XE only if pin 9 is held low. If pin 9 is not held low, this pin is disconnected.

31	A8	Buffered A8	Output
32		Data Buffer Select	1090 Input

Disabled by default and is normally held high by the 1090 circuitry. A 1090 card can connect the data bus on the 1090 to the computer by setting this low. This does not connect or disconnect the data bus between 1090 cards. An open collector gate should be used to drive this signal low.

In many cases, the card will drive this signal low either on a write or after the address has been decoded and the data bus is required by the card.

33	A7	Buffered A7	Output
34	D7	Buffered D7	Bidirectional
35	A6	Buffered A6	Output
36	D6	Buffered D6	Bidirectional
37	A5	Buffered A5	Output
38	D5	Buffered D5	Bidirectional

39	A4	Buffered A4	Output
40	D4	Buffered D4	Bidirectional
41	A3	Buffered A3	Output
42	D3	Buffered D3	Bidirectional
43	A2	Buffered A2	Output
44	D2	Buffered D2	Bidirectional
45	A1	Buffered A1	Output
46	D1	Buffered D1	Bidirectional
47	A0	Buffered A0	Output
48	D0	Buffered D0	Bidirectional
49		Computer Ground Return	Power
	Isolated ground connection through the flat cable to the computer.		
50	+12 VDC		Power
	Expansion cards should use no more than 30 milliamps as per Atari's specifications.		
	Note: The only known use for this pin is shown on the schematic for the Parallel/Serial Interface Card.		

Timing

Most timing information on data sheets is accompanied by a timing diagram with variables that have to be cross-referenced in a table to determine the actual timing of each event. As the Atari runs at 1.77 or 1.79 MHz, I decided to put this information in tabular format in chronological order. (PAL systems run at 1.77 MHz and NTSC systems run at 1.79 MHz.) Times are in nanoseconds, the rise/fall time of Phi2 is not shown, and worst-case numbers are used.

Event:	PAL/1.77 MHz	NTSC/1.79 MHz
Phi2 is low (Start of clock cycle)	0	0
Address is set	159	157
R/W is set	159	157
Phi2 is high	283	279
Data is set (CPU Write)	396	391
Data setup time for read (CPU Read)	509	503
Phi2 is low	565	558
Read data hold time	576	569
Write data hold time	632	624

Note: Timing data is computed using the SY6500 datasheet from Synertek.

You will need to keep in mind propagation delay times of every component in your PBI device to make sure addresses are handled and data is processed and/or ready at the correct time. In addition, Phi2 is buffered and delayed by the 74LS08 on the system board. If a 1090 Expansion is used, an additional delay is created by the transceiver propagation time. Propagation times can also vary due to temperature and other factors. Substituting faster chips will also reduce propagation times.

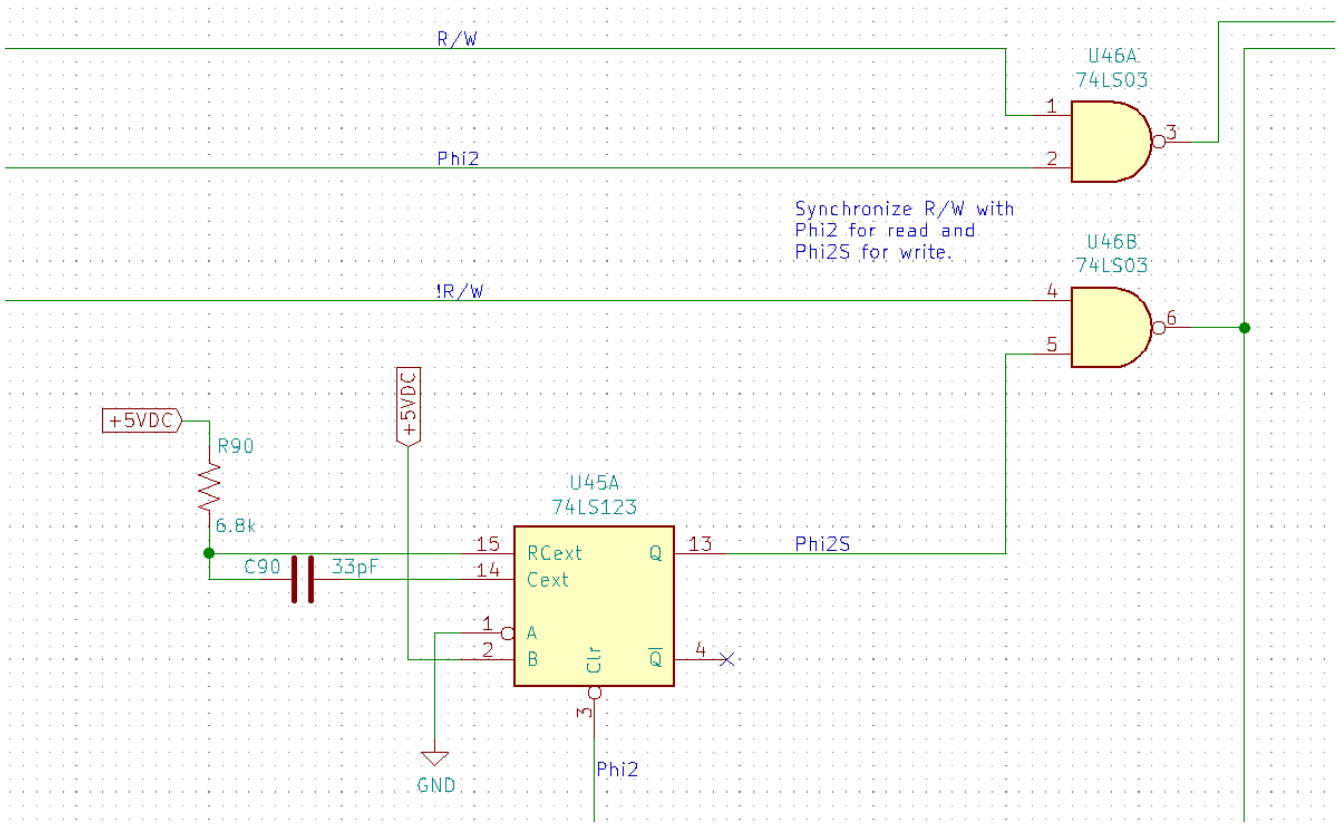
The Atari XL Expansion System Technical Specification, Revision A, from May 11, 1984 has a table of timing information for the 1090 XL Expansion System. This table appears to be for NTSC systems only and differs slightly from what has been computed above. This information is shown below in tabular format and chronological order.

Event:	Time in nanoseconds:
Phi2 is low (Start of clock cycle)	0
Address is set	177
R/W is set	186
EXTENB setup time	195
EXTSEL setup time	250
Phi2 is high	279
Data is set (CPU Write)	422
Data setup time for read (CPU Read)	486
Phi2 is low	558

Read data hold time	565
Write data hold time	591

As Phi2 is buffered twice with a 1090, the Atari XL Expansion System Technical Specification specifies that the delay between Phi2 and Phi2 on the 1090 bus can be anywhere from 5 to 31 nanoseconds. This delayed Phi2 signal can be referred to as Buffered Phi2.

As you can see from the above tables, the timing information does vary slightly. The biggest problem, however, is when writing information to a register or memory through the parallel bus. If there is too much of a delay between the fall of Phi2 on the CPU and when the data being stored, the data bus will contain corrupted information and this corrupted information will be stored. This is because once Phi2 falls, the CPU is working on the next instruction and holds the data on the bus for a very limited time. This is all compounded by the delay between Phi2 on the processor and Phi2 after it has been buffered. One solution is to create a shortened Phi2 signal by the use of a 74LS123 chip. By using this shortened Phi2 such that a chip or register is signaled to write the information from the data bus after the data has been provided by the CPU but before Phi2 actually falls the data will not be corrupted. The following use of a 74LS123 has been shown to work:



As per the datasheet for the 74LS123 chip, the pulse will be approximately 150 nanoseconds. The 74LS123 datasheet also specifies that the typical propagation delay from low to high is 30

nanoseconds and from high to low is 18 nanoseconds. Once the offset created by the buffering of Phi2 is factored in, Phi2S is falling after the data is on the bus and before Phi2 actually falls. This will result in Phi2S falling at approximately 508 nanoseconds.

If the worst timing information of the previous 2 tables is combined into as single table, with the delayed Buffered Phi2 and Phi2S factored in, the chronological order of events is as follows:

Event:	Time in nanoseconds:
Phi2 is low (Start of clock cycle)	0
Buffered Phi2 is low	31
Address is set	177
R/W is set	186
EXTENB setup time	195
EXTSEL setup time	250
Phi2 is high	279
Buffered Phi2 is high	310
Phi2S is high (Phi2S timing is based on Buffered Phi2)	340
Data is set (CPU Write)	422
Data setup time for read (CPU Read)	503
Phi2S is low (Phi2S timing is based on Buffered Phi2)	508
Phi2 is low	558
Read data hold time	565
Buffered Phi2 is low	589
Write data hold time	591

Note: Transceivers on the 1090 board will add additional delays to the address bus, data bus, and other signals such as R/W.

Reserved PBI Addresses

The following PBI addresses are reserved per Atari's PBI standard. However, the PBI was rarely used and the few vendors that did use the PBI either did not use or only "loosely" followed this standard. Only $D1FF is used by the XL/XE Operating System.

$D100 - $D1AF		Available
$D1B0 - $D1C7		Speech/Modem/Disc Registers
$D1C8- $D1CE		Atari Reserved
$D1CF	Read	Alternate Interrupt Register (Atari 1450 only)
$D1D0 - $D1DF		Audio Registers
$D1E0 - $D1E7		Atari Reserved
$D1E8 - $D1EF		Parallel/Serial Registers
$D1F0 - $D1F7		Alternate CPU Registers
$D1F8 - $D1FD		80 Column Video Registers (80CVC)

$D1FA	Write	CRTC register address
$D1FB	Read/Write	CRTC register data
$D1FC	Write	VRAM aperture base
$D1FD	Write	VRAM aperture base

$D1FE	Write	RAM Bank Select Register

Note: No software is known to use $D1FE to select a RAM bank.

Memory Bank Address:	16384-32767	($4000-$7FFF)
Select Bank:	53758, BN*	($D1FE, BN)
Deselect Bank:	53758, 128	($D1FE, 80)

* BN is the Bank Number as follows:

First RAM Card:	0 – 3	$0-$3
Second RAM Card:	16-19	$10-$13

$D1FF	*	Read	Hardware IRQ Register

If the IRQ line is pulled low the Operating System can read this register to determine which device set the IRQ. Bits are set as per $D1FF (Write).

$D1FF	*	Write	Hardware Select Register

With the exception of memory only cards, each PBI device has an “id” that is selected by setting the appropriate bit as follows:

Bit:	7	6	5	4	3	2	1	0
Device:	7	6	5	4	3	2	1	0

Up to 8 devices can be accessed, but only one device at a time.

i.e. Setting $D1FF with $20 selects device 5.

$D600 - $D61F	Device D0
$D620 - $D63F	Reserved for modems
$D640 - $D67F	Device D1
$D680 - $D6BF	Device D2
$D6C0 - $D6FF	Device D3
$D700 - $D73F	Device D4
$D740 - $D77F	Device D5
$D780 - $D73F	Device D6
$D7C0 - $D7FF	Device D7

Note: There is no evidence that Atari used the $D600-$D7FF region in any of their 1090 cards.

$D800 - $DFFF	Atari OS ROM Floating Point Mathematics Package Parallel Device ROM when device is selected by $D1FF and MPD is set low.

Used by: 80CVC

ROM Vector Table:

$D800	*	ROM Check Sum Low Byte
$D801	*	ROM Check Sum High Byte

$D802	*	ROM Revision Number
$D804	*	Device Type
$D805		JMP ($4C)
$D806		I/O Vector Low Byte
$D807		I/O Vector High Byte
$D808		JMP ($4C)
$D809		Interrupt Vector Low Byte
$D80A		Interrupt Vector High Byte
$D80B		ID Number ($91)
$D80C	*	Device Name (ASCII)
$D80D		Open Vector Low Byte -1
$D80E		Open Vector High Byte
$D80F		Close Vector Low Byte -1
$D810		Close Vector High Byte
$D811		Get Byte Vector Low Byte -1
$D812		Get Byte Vector High Byte
$D813		Put Byte Vector Low Byte -1
$D814		Put Byte Vector High Byte
$D815		Status Vector Low Byte -1
$D816		Status Vector High Byte
$D817		Special Vector Low Byte -1
$D818		Special Vector High Byte
$D819		JMP ($4C)
$D81A		Init Vector Low Byte
$D81B		Init Vector High Byte
$D81C	*	Unused

* = Optional

Third Party PBI Address Use

The following PBI addresses are used by third parties. These addresses were taken from The Altirra Hardware Reference Manual and are included as a reference so as to avoid address conflicts when designing PBI devices. Please see the Altirra Hardware Reference Manual for more information.

CSS BLACK BOX

$D120 - $D13F		6551 ACIA
$D150 - $D17F		6522 VIA
$D180 - $D1BF		6821 PIA
$D1C0	Write	Control register
$D1FF	Read	Status Register
$D600 - $D6FF		Black Box RAM

ICD MULTI I/O (MIO)

$D1C0 - $D1DF		6551 ACIA
$D1E0 - $D1FB		Mirrors of $D1FC - $D1FF
$D1FC	Read	Reset SCSI bus strobe
$D1FC	Write	RAM bank A8-A15
$D1FE	Read	Status 1
$D1FE	Write	Control 1
$D1FF	Read	Status 2
$D1FF	Write	Control 2
$D600 - $D6FF		MIO RAM

ULTIMATE 1 MB

$D100 - $D1BE	I/O RAM (When enabled.)
$D1BF	Bank switch register
$D1FF	Emulated PBI device configuration

$D600 - $D7FF	I/O RAM (When enabled.)
$D800 - $DFFF	RAM or OS ROM (When enabled.)

VideoBoard XE

$D640 - $D65F or $D740 - $D75F	Register Map

Parallel Device Handlers and Drivers

What follows is the text of the Atari document "The Software Implementation of Parallel Device Handlers and Drivers", Revision B, May 22, 1984. This is included within this document in an effort to put all parallel device information in one place so that it is easier to develop new hardware. Where deemed necessary, additional notes have been added to the text and are noted with an asterisk.

1. OVERVIEW

Computers in the Atari 600XL/800XL/1450XLD line provide a parallel bus. Parallel devices on this bus physically include a ROM which contains code for handling I/O requests and driving the device. This document describes the interface between the Operating System and a Parallel Device Handler and/or driver.

As many as eight parallel devices, numbered 0 through 7, may be on the parallel bus. A device's number is determined by hardware on the device, e. g., a configuration switch. The ROM space in every parallel device is two kilobytes, addressed D800H through DFFFH, in the same address space as the Operating System's Floating Point Package. When a parallel device is selected, addresses D800H through DFFFH refer to locations within the device ROM. Selection of a device and it's ROM is accomplished by setting the corresponding bit in hardware register D1FFH. For instance, storing 04H at D1FFH selects device 2. (It is illegal to set more than one bit in D1FFH.) Storing zero at D1FFH deselects any parallel device and selects the Floating Point Package. Since the Floating Point Package and each parallel device ROM occupy the same address space, the parallel device handlers and drivers may neither use the Floating Point Package nor invoke any other code which uses the Floating Point Package.

> * NOTE: Only the parallel device is able to disable the "Operating System's Floating Point Package". This is accomplished when the parallel device sets the Math Pack Disable pin, on the parallel port, low.

The Operating System is responsible for selecting the parallel devices at appropriate times, transferring control to the devices via fixed vectors within the device ROM's, receiving control back from the devices, and re-installing the Floating Point Package. A parallel device may be selected for the following four reasons:

1. Initialization

2. Processing of a Device Handler requests

3. Processing of a low-level Serial I/O (SIO) type requests

4. Processing of a parallel device interrupt request (IRQ)

A parallel device ROM, then, would provide code to process each of these four occurrences and vectors to the appropriate portions of code with the ROM.

In support of parallel device handlers and drivers, the Operating System apportions 512 bytes of RAM, addressed D600H through D7FFH, to the eight possible devices.

> * NOTE: At this time there is no evidence that D600H through D7FFH was ever used by Atari in a parallel port device. Third party vendors, however, made use of this region for their own RAM. Additionally, given that a stock Atari 600XL does not have RAM at this location there is no way for the Operating System to provide memory that is not available. The Alitirra Hardware Reference Manual, Chapter 11, states: "The address range $D600-D7FF is reserved for PBI device addressing and can be used for RAM, ROM, or I/O of an actively selected PBI device."

2. PARALLEL DEVICE ROM REQUIREMENTS

The Operating System requires a data table which starts at the low address of a parallel device ROM. This data table affirms the existence of a ROM for the device selected and provides vectors to routines within the ROM. Device selection and transferring of control from the Operating System will not be performed correctly unless this data is correct.

The data table consists of mandatory and optional entries. Only the mandatory entries are required for correct operation. The optional entries describe the ROM and device and only suggestions as to their use are given here.

It should be noted that the Device Handler Vector Table (D80DH through D81CH) has the same format as other Operating System Resident Handler (e g., Printer Handler) vector tables.

Parallel Device ROM Data Table

D800 – D801	(Optional) ROM Checksum (low byte, high byte)
D802	(Optional) Revision Number
D803	(Mandatory) ID Number 1 Value = 80H
D804	(Optional) Name or Type
D805 – D807	(Mandatory) Low-level I/O Vector Value = JMP address (low byte, high byte)
D808 – D80A	(Mandatory) IRQ Handler Vector Value = JMP ($4C) address (low byte, high byte)
D80B	(Mandatory) ID Number 2 Value = 91H
D80C	(Optional) Device Name Value = Device Name in ASCII

D80D – D80E	(Mandatory) Device Handler Open Vector Value = address-1 (low byte, high byte)
D80F – D810	(Mandatory) Device Handler Close Vector Value = address-1 (low byte, high byte)
D811 – D812	(Mandatory) Device Handler Get-Byte Vector Value = address-1 (low byte, high byte)
D813 – D814	(Mandatory) Device Handler Put-Byte Vector Value = address-1 (low byte, high byte)
D815 – D816	(Mandatory) Device Handler Status Vector Value = address-1 (low byte, high byte)
D817 – D818	(Mandatory) Device Handler Special Vector Value = address-1 (low byte, high byte)
D819 – D81B	(Mandatory) Initialization Vector Value = JMP ($4C) address (low byte, high byte)
D81C	(Optional) Not Used Value = 0

3. GENERAL DEVICE SELECTION PROCESS

Seven registers are used in the parallel device selection process:

1. PDVS (D1FFH), Parallel device select hardware registers

2. SHPDVS (0248H), Parallel device select shadow

3. PDVMSK (0247H), Parallel device mask

4. PDVI (D1FFH), External parallel device IRQ hardware registers

5. PDIMSK (0249H), External parallel device IRQ mask

6. IPDVI (D1CFH), Internal parallel device IRQ hardware registers

7. IPDIMK (0254H), Internal parallel device IRQ mask

Writing a value to the parallel device selection register, PDVS, causes a device or the Floating Point Package to be selected. Setting bit n of PDVS causes parallel device n to be selected. Storing 0 into PDVS causes the Floating Point Package to be selected.

* NOTE: The computer does not have the PDVS register at D1FFH. This register must be on each parallel device. Also, see the previous note regarding Floating Point Package selection. Parallel memory devices do not have a PDVS register.

RAM location SHPDVS is a shadow of the value written into the device selection register, PDVS. Whenever the Operating System writes a value into PDVS, the same value is stored in SHPDVS. This is necessary because the device selection is not available by reading PDVS. For instance, if code within the device ROM needs to know the device's number it can inspect SHPDVS to determine which device is currently selected. (N.B. When storing values into PDVS and SHPDVS, you MUST store first into SHPDVS and then into PDVS.)

The parallel device mask, PDVMSK, is used to control the selection process in all cases except for initialization and IRQ processing. By convention, if bit n of PDVMSK is set then device number n exists on the parallel bus. If bit n of PDVMSK is clear, then device n will not be selected. The Operating System, itself, does not set PDVMSK. It is the responsibility of each device's initialization routine to set the PDVMSK bit corresponding to the device's number.

The parallel device IRQ hardware registers, PDVI and IPDVI, indicate which of the parallel devices initiated an IRQ. These are read-only locations. Bit n of PDVI is set if external parallel device n initiated an IRQ. And bit n of IPDVI is set if internal parallel device n initiated an IRQ. (External parallel devices are devices that are physically located outside the computer and are connected via the Parallel I/O Port. Internal parallel devices are devices that are built into the computer, e. g., the disk drive on the 1450XLD.

* NOTE: As with PDVS (D1FFH write), PDVI (D1FFH read) would be located on a parallel device if the parallel device can initiate an IRQ. Not all parallel devices require PDVI.

The parallel device IRQ masks, PDIMSK, and IPDIMK, are used to determine which parallel devices are allowed to receive IRQ requests. Parallel device n will be selected to handle its IRQ only if bit n in PDIMSK or IPDIMK (whichever is appropriate) is set. The Operating System does not set PDIMSK or IPDIMK. It is the responsibility of each device's initialization routine to set the PDIMSK or IPDIMK bit corresponding to the device's number if the device driver is to handle IRQ's.

N.B.: Clearing a devices bit in PDIMSK or IPDIMK merely prevents the OS from recognizing IRQ's initiated by that device; it does not prevent the IRQ's from occuring. Consequently, IF A PARALLEL DEVICE INITIATES AN IRQ WHEN ITS BIT IN PDIMSK OR IPDIMK IS CLEAR, THE OPERATING SYSTEM WILL GO INTO AN INFINITE LOOP. In order to prevent an infinite loop, the parallel device handler must set its bit in PDIMSK or IPDIMK before it enables its hardware, and it must disable its hardware before it clears its bit in PDIMSK or IPDIMK.

4. DEVICE INITIALIZATION INTERFACE

4.1 OPERATING SYSTEM INITIALIZATION

During cold-start and warm-start initialization, after initializing resident device handlers, and before attempting to initialize the cartridge, the Operating System initializes each of the parallel devices. In order of device number (number 0 first), the Operating System selects each device and, if the two ID bytes are correct (D303H contains 80H and D808BH contains 91H), transfers control to the parallel device initialization routine via a JSR to the jump vector at D819H. Thus, only those devices which physically exist and have the correct ID byte values are initialized.

4.2 PARALLEL DEVICE INITIALIZATION

After receiving control from the operating system, the parallel device initialization routine performs device dependent initialization.

In addition, certain device independent initialization must be performed. In order to receive control to process device handler requests and low-level device I/O, the device initialization routine must set the bit in PDVMSK corresponding to the number of the device. (That bit is the lowest order bit set in SHPDVS, the device select shadow.) In order to receive control to handle device IRQ's, the device initialization routine must set the bit in PDIMSK or IPDIMK corresponding to the number of the device.

Also, if the device provides its own device handler (some parallel devices may instead rely on resident device handlers to process medium-level I/O requests), it must ensure that the name of the device is in the Operating System Device Table, HATABS, and that the corresponding handler address is that of the resident Generic Parallel Device Handler, GPDVV (E48FH). The address of the Generic Parallel Device Handler, and not the address of the parallel device handler at D80DH, must be used because the device does not remain selected at all times. It is the responsibility of the Generic Parallel Device Handler to select the device and transfer control to the appropriate handler routine within the device ROM.

Upon completion, the parallel device initialization routine returns control to the operating system via an RTS.

5. DEVICE HANDLER INTERFACE

5.1 GENERIC PARALLEL DEVICE HANDLER

The Generic Parallel Device Handler is a resident handler which is responsible for invoking the handler routines within the parallel device ROM's. Like other device handlers, it processes medium-level I/O requests to open a device, close a device, get a byte, put a byte, return status, and perform special functions. Typically, the Generic Parallel Device Handler is called by CIO, but it may be called directly by an application, as well. Like other resident device Handlers, the Generic Parallel Device Handler is reached via an entry in HATABS which provides the address (GPDVV, E48FH) of a table of vector entries into the handler. The Operating System, itself, does not enter the Generic Parallel Device Handler into HATABS. It is not invoked unless some parallel device which supports a device hanlder enters it (and the device's name) into HATABS.

The Generic Parallel Device Handler routines, not knowing which of the parallel device handlers will process the request, select each of the parallel devices, in order of device number (number

0 first), and transfer control to the corresponding routine within the handler. The entry point of the parallel device handler's routine is determined from the address vector table starting at D80DH. The C Status Flag is used to determine if the selected parallel device handler actually performed the request. Upon return to the Generic Parallel Device Handler, if the C Flag is clear, the currently selected parallel device did not perform the request and the next parallel device is selected. (Thus, even those parallel devices which do not provide a device handler must have a simple handler routine which ignores all handler requests and returns the C Flag clear.) If the C Flag is set, the request was performed and the Generic Parallel Device Handler terminates the selection process and returns to the routine which called it. If after calling all of the parallel device handlers the handler request has not been performed, the Generic Parallel Device Handler returns a Non-existent Device (82H) status to the calling routine.

The entry conditions into a parallel device handler are the same as for the resident device handlers. Parameter passing is accomplished using the A, X and Y registers and the page zero IOCB. On entry, the A register contains a data byte, if necessary; the X register contains the index to the originating IOCB; and the Y register contains a Function Not Supported (92H) status.

5.2 PARALLEL DEVICE HANDLER

The function of a parallel device handler is very similar to the resident device handlers. One difference is that a parallel device handler may be called even though the request is for a different device type or a different device unity number. The parallel device handler must check the request (in the page zero IOCB or, in the case of a put-byte request, in the originating IOCB) and return the C Flag clear if it cannot handle the request. The other difference is that the handler would not call SIO to perform the physical I/O, but instead would call a low-level I/O routine (driver) within the parallel device ROM.

After handling the request, the parallel device handler returns to the Generic Parallel Device Handler via an RTS. On exit, the A register contains a data byte, if necessary; the Y register contains the status for the request; and the C Flag should be set to indicate that the request was handled.

N.B.: Prior to beginning the device selection process, the Generic Parallel Device Handler sets the critical I/O flag (CRITIC) to disable deferred vertical blank processing. This is a bug in the O.S., and usually has undesirable consequences. In order to compensate for this bug, any parallel device handler routine which is more than a few instructions long should store 0 into CRITIC.

6. LOW-LEVEL DEVICE I/O INTERFACE

Parallel device ROM's may contain a routine similar to SIO for performing the low-level physical I/O for the parallel device.

6.1 OPERATING SYSTEM LOW-LEVEL I/O

All low-level I/O requests (including those for Serial I/O) will go to the Operating System low-level I/O routine which attempts to perform the request via the low-level I/O routines within the parallel device ROM's. Serial I/O requests are included so that the resident device handler or application need not know whether the device is serial device or parallel device. For example, the parallel disk handler on the 1450XLD relies on the DOS to handle medium-level (CIO-type) I/O

requests. The DOS makes low-level (SIO-type) I/O requests which are sometimes for a serial drive and other times for a parallel drive.

The Operating System low-level I/O routine (PIO) is reached via the jump vector at SIOV (E459H).

As in the Generic Parallel Device Handler, PIO selects each of the parallel devices, in order of device number (number 0 first), and transfers control to the low-level I/O routine within the parallel device ROM. Control is transferred via a JSR to the jump vector at D805H. Again, the C Status Flag is used to determine if the selected parallel device low-level I/O routine actually performed the request. If a parallel device I/O routine returns the C Flag clear, then the next parallel device is selected. If a device I/O routine returns the C Flag set, the selection process is terminated and control is returned to the routine which requested the I/O. If after calling all of the parallel device low-level I/O routines the I/O request has not been performed, then the Operating System low-level I/O routine calls the resident Serial I/O routines to process the request.

To disable deferred vertical blank processing, the Operating System sets the Critical Section Flag, CRITIC, during the selection process.

The entry conditions into a parallel device low-level I/O routine are the same as for the entry into the resident Serial I/O routine. All parameters passed are contained in the Device Control Block (DCB, 0300H).

6.2 PARALLEL DEVICE LOW-LEVEL I/O

The parallel device low-level I/O routine (driver) performs requests for physical I/O for a parallel device. Because the request may be for a different device type or a different device unit number, the parallel device low-level I/O routine must check the request (bus ID and unit number in the DCB) and return the C Flag clear if the I/O request is for some other device.

If the parallel device ROM provides both a CIO-level handler and an SIO-level driver, it is recommended that the ROM incorporate a single routine for actually performing the physical I/O. This routine would not check the request for the correct device type and physical device unit number. Both the parallel device handler (if present) and the parallel device low-level I/O routine, which do check the request for validity, could then call this routine to perform the validated I/O request.

After performing the request, the parallel device low-level I/O routine returns to the Operating System low-level I/O routine via an RTS. On exit, the I/O has been initiated (or completed, if the device is not interrupt-driven), the Y register contains the status for the request, and the C Flag should be set to indicate that the request was processed.

N.B.: The parallel device low-level I/O routine is not allowed to modify the DCB, except for DUNIT (0301H). The original value of DUNIT is saved by PIO before the parallel device low-level I/O routine is called, and it will be restored after the parallel device low-level I/O routine returns. This restoration takes place only at the very end of PIO, so that if a parallel device low-level I/O routine changes DUNIT and then returns with Carry clear, subsequent parallel device low-level I/O routines and SIO will see the modified value of DUNIT.

7. DEVICE IRQ HANDLER INTERFACE

7.1 OPERATING SYSTEM IRQ PROCESSING

When an IRQ occurs, after checking for a serial input IRQ, the Operating System checks PDVI and IPDVI to see if parallel device initiated the IRQ. If a parallel device initiated the IRQ and the bit corresponding to that device is set in PDIMSK or IPDIMK (whichever is appropriate), the Operating System transfers control to the IRQ handler of the parallel device which initiated the IRQ. If more than one device initiated the IRQ, only the lowest-numbered device will get control (IRQ's from higher-numbered devices remain pending). Transfer of control is accomplished via a JSR to the jump vector at D808H.

Because a parallel device may be executing when the interrupt occurs, before selecting a device to handle the IRQ, the Operating System places the value of the device select shadow, SHPDVS, on the stack. When all parallel device IRQ's have been handled, the value of the device select from the stack is restored to the PDVS (and SHPDVS).

After the parallel device IRQ handler returns control, the Operating System returns to the interrupted routine via an RTI.

7.2 PARALLEL DEVICE IRQ HANDLING

The parallel device IRQ handling routine processes the interrupt. After the interrupt has been processed, the parallel device IRQ handler returns to the Operating System via an RTS instruction.

The Operating System support of the parallel bus is designed so that parallel I/O and serial I/O are able to be done concurrently. Therefore, in order to avoid the loss of serial port data, interrupts from parallel devices must be cleared and interrupts enabled within about 150 microseconds.

N.B.: Earlier revisions of this document stated that a parallel device IRQ routine could re-enable interrupts by executing a CLI instruction. Those earlier revisions are wrong. A PARALLEL DEVICE IRQ ROUTINE MUST NOT ENABLE INTERRUPTS.

8.0 RAM AVAILABILITY

8.1 PAGE D6xxH AND D7xxH RAM

> * NOTE: At this time there is no evidence that D600H through D7FFH was ever used by Atari in a parallel port device. Third party vendors, however, made use of this region for their own RAM. Additionally, given that a stock Atari 600XL does not have RAM at this location there is no way for the Operating System to provide memory that is not available. The Alitirra Hardware Reference Manual, Chapter 11, states: "The address range $D600-D7FF is reserved for PBI device addressing and can be used for RAM, ROM, or I/O of an actively selected PBI device."

512 bytes of RAM, addressed D600H through D7FFH, are available for use by parallel device handlers and drivers. This RAM is not used at all by the O.S.; in particular, it is not zeroed during coldstart or warmstart.

Each card slot has a portion of this RAM allocated to it, according to the following scheme:

D600H – D61FH	Slot 0 RAM
D620H – D63FH	Reserved for use by modem devices
D640H – D67FH	Slot 1 RAM
D680H – D6BFH	Slot 2 RAM
D6C0H – D6FFH	Slot 3 RAM
D700H – D73FH	Slot 4 RAM
D740H – D77FH	Slot 5 RAM
D780H – D7BFH	Slot 6 RAM
D7C0H – D7FFH	Slot 7 RAM

As the table indicates, slots 1-7 each own 64 bytes of RAM, while slot 0 owns 32 byes of RAM. There are 32 bytes reserved for use by modem devices because 64 bytes is not enough to provide the buffering that a modem requires.

Obviously, a parallel device handler which uses more than 32 bytes of page D6xxH and D7xxH RAM cannot be placed in slot 0. With this one exception, parallel device handlers should be designed so that they will function properly in any slot. In particular, before accessing page D6xxH and D7xxH RAM, the handler must determine which slot it is in (by examining SHPDVS) and do an address calculation.

8.2 ZEROPAGE RAM

There are 11 bytes on page zero which can be used by parallel device handlers and drivers (except during IRQ processing):

STATUS	[30H,1]
CHKSUM	[31H,1]
BUFRLO	[32H,1]
BUFRHI	[33H,1]
BFENLO	[34H,1]
BFENHI	[35H,1]
BUFRFL	[38H,1]
RECVDN	[39H,1]
XMTDON	[3AH,1]
CHKSNT	[3BH,1]
NOCKSM	[3CH,1]

Those 11 bytes are normally reserved for use by SIO. Since it is not possible for SIO to be active at the time that a parallel device handler or driver is called (except during IRQ processing), parallel device handlers and drivers may use these locations freely; their original values do not have to be saved and restored.

In addition, there are 4 bytes on page zero that are reserved for use by parallel device IRQ routines:

ABUFPT	[1CH,4]

The only part of the OS which uses these 4 bytes is the immediate IRQ handler. Parallel device IRQ routines need not save and restore the original values of these memory locations.

Parallel device IRQ routines which need more than 4 bytes of zeropage RAM should use the 11 SIO bytes listed above; naturally, the IRQ routine must save and restore the original values of these bytes. Parallel device IRQ routines needing more than 15 bytes of zeropage RAM, and parallel device non-IRQ routines needing more than 11 bytes of zeropage RAM, can get more by saving and restoring other locations on page zero; the safest spot to use for this purpose is probably the zeropage IOCB ([20H,0CH]).

8.3 OTHER RAM

The following memory locations may be used as scratch storage by non-IRQ parallel device routines. They are normally reserved for use by SIO.

CDEVIC	[23AH,1]
CCOMND	[23BH,1]
CAUX1	[23CH,1]
CAUX2	[23DH,1]
TEMP	[23EH,1]
ERRFLG	[23FH,1]
CRETRY	[29CH,1]
DRETRY	[2BDH,1]
TIMFLG	[317H,1]
STACKP	[318H,1]
TSTAT	[319H,1]

8.4 VECTORS

Parallel device handlers should provide, in page D6xxH and D7xxH RAM, whatever vectors are necessary to replace the parallel device handler with a RAM-resident handler. Prime candidates for vectoring are the IRQ entry point (through D808H) and the low-level entry point (through D805H).

There is no uniform scheme for vectoring due to the fact that any program which intercepts one of these vectors is going to be extremely device-specific anyway.

8.5 FALSE-ADDRESS WARNING

Page D5xxH is used for hardware locations within cartridges. Therefore, when parallel device handlers access their RAM on pages D6xxH and D7xxH, they must be careful not to generate any references to page D5xxH.

Due to a quirk in the 6502 microprocessor, this is not a simple as it sounds. Whenever indexed addressing is used to cross a page boundary, the 6502 will generate an extra memory cycle during which it references the memory location one page below the desired address.

For example, consider:

```
LDX   #$FE
LDA   $D510,X
```

During execution of the LDA instruction, the 6502 will generate a reference to memory location D50EH, as well as to the desired address D60EH.

This applies not only to LDA instructions, but to all instructions (including stores). It also applies to indirect indexed addressing, as in LDA (ZPAGE), Y.

Example Parallel Device Source Code

This section provides source code examples for operation of a parallel bus device. While this source code's function is not verified by this author, it is provided as a starting point for developing software for a parallel bus device.

From Antic, April 1985, Page 78, Parallel Bus Revealed:

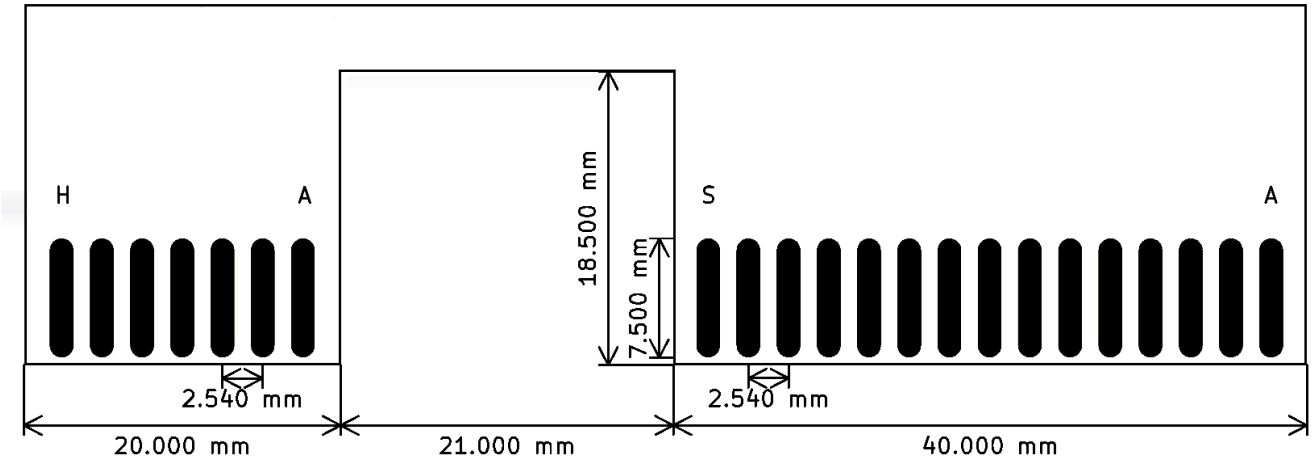
```
10 ; Parallel Device Handler Example
20 ; By Earl Rice
30 ; ANTIC Magazeine
40 ;
50 ; (ASM,,#D:MYFILE.OBJ) because the obj code is put
60 ; where there is no RAM available.
70          .OPT OBJ
80 ;  EQUATES
90 PDVMSK = $0247                   ; Parallel device mask (indicates which are
0100 PDIMSK = $0249                 ; Parallel interrupt mask (not used in this
0110 GPDVV = $E48F                  ; Generic Parallel Device Vector
0120 ;
0130 HATABS = $031A                 ; Device handler table
0140 CRITIC = $42                   ; Critical code section flag
0150 ;
0160 DEVNAM = 'T                    ; Device name, E.G. T for "Telephone".
0170 HWGET = $D100                  ; Hardware GET register
0180 HWPUT = $D100                  ; Hardware PUT register.
0190 HWRSET = $D101                 ; Hardware reset (clears get register).
0200 HWSTAT = $D101                 ; Hardware STATUS register
0210 ;
0220        *=      $D800
0230 ; ROM vector table
0240        .WORD 0                 ; Optional ROM checksum
0250        .BYTE 0                 ; Optional Revision number
0260        .BYTE $80               ; Mandatory ID number
0270        .BYTE 0                 ; Optional Name or Type
0280        JMP NONEED              ; Lo-level IO vector, which we don't need
0290        JMP NONEED              ; IRQ handler vector, which we don't need.
0300        .BYTE $91               ; Mandatory ID number
0310        .BYTE DEVNAM            ; Device name
0320        .WORD NONEED-1          ; Open vector, which we don't need.
0330        .WORD NONEED-1          ; Close vector, which we don't need.
0340        .WORD GETBYT-1          ; GET BYTE vector.
0350        .WORD PUTBYT-1          ; PUT BYTE vector.
0360        .WORD GETSTA-1          ; GET STATUS vector.
0370        .WORD NONEED-1          ; SPECIAL vector, which we don't need.
0380        JMP INIT                ; INIT vector at Power up or reset.
```

```
0390            .BYTE 0                     ; NOT USED
0400 ;
0410 ; CODE STARTS HERE
0420 ;
0430 ; Initialize device and device handler
0440 INIT
0450            LDA PDVMSK                  ; Get enabled device flags
0460            ORA #1                      ; Set bit 0
0470            STA PDVMSK                  ; & replace
0480 ; Note: if device used interrupts we would set bit 0 of
0490 ;
0500 ; Put device name in Handler table HATABS
0510            LDX #0
0520 ;                  Top of loop
0530 SEARCH
0540            LDA HATABS,X                ; Get a byte from table
0550            BEQ FNDIT                   ; 0? Then we found space.
0560            INX
0570            INX
0580            INX
0590            CPX #36                     ; Length of HATABS
0600            BCC SEARCH                  ; Still looking
0610            RTS                         ; No room in HATABS, devicenot initialized
0620 ;
0630 ;                  We found a spot.
0640 FNDIT
0650            LDA #DEVNAM                 ; Get device name.
0660            STA HATABS, X               ; Put it in blank spot.
0670            INX
0680            LDA #GPDVV&$FF              ; Get lo byte of vector.
0690            LDA #GPDVV/$0100            ; Get hi byte of vector.
0700            STA HATABS+2,X
0710            RTS
0720 ;
0730 ; GET BYTE routine.
0740 GETBYT
0750            LDA #0
0760            STA CRITIC                  ; Enabled deferred vertical blank.
0770            LDA HWGET                   ; Get a byte from hardware.
0780            STA HWRSET                  ; Reset hardware.
0790            SEC                         ; Indicate we handled it.
0800            RTS
0810 ;
0820 ; PUT BYTE routine.
0830 PUTBYT
0840            LDX #0
0850            STX CRITIC                  ; Enable deferred vertical blank.
```

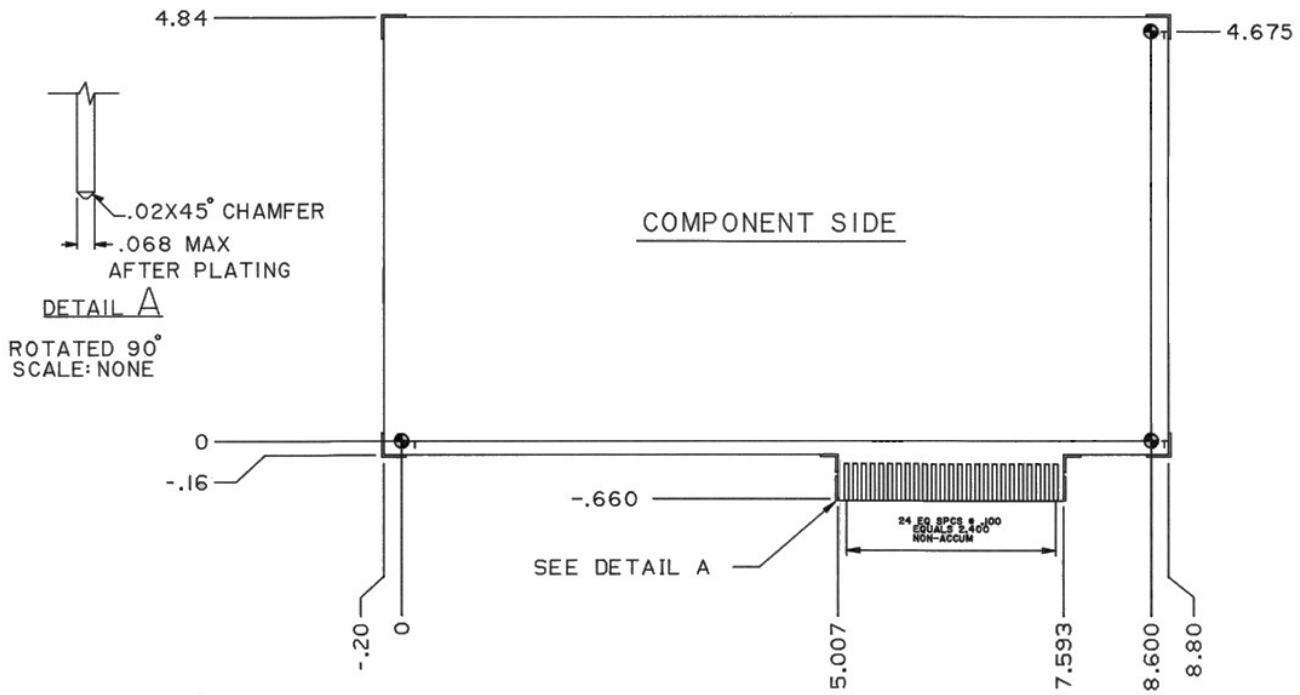
```
0860            STA HWPUT               ; Put byte to hardware.
0870            SEC                     ; Indicate we handled it.
0880            RTS
0890 ;
0900 ; GET STATUS routine.
0910 GETSTA
0920            LDA #0
0930            STA CRITIC              ; Enabled deferred vertical blank.
0940            LDA HWSTAT              ; Get HW status.
0950            SEC                     ; Indicate we handled it.
0960            RTS
0970 ;
0980 ; Do nothing routine.
0990 NONEED
1000            SEC                     ; Indicate we handled it.
1010            RTS
1020 ;
1030 ;
1040            .END
```

Mechanical Specifications

Atari XE Cartridge Port with ECI:



Card For 1090 XL Expansion System:



HOLE CHART								
	CODE	HOLE DIA.	QTY.	PLATED THRU	⌖DIA	X	Y	TOL.
TOOLING HOLE	T	.156 ± .002 / .001	3	NO	⌖DIA	X	Y	.006 (M)

UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN INCHES
TOLERANCES ON:
ANGLES - ± 3°
.X = ± .1
.XX = ± .01
.XXX = ± .005

Publication Errors

This section details errors discovered in various publications.

Mapping the Atari

Appendix Twelve
Errors in the description of addresses $D100-$D1FF

Appendix Fourteen
Errors in the names and description of the pins for the XL parallel bus.
Errors in the pin assignments for the XE ECI.

Antic Vol. 3 No. 10 - February 1985 - Money Mastery – Toolbox
Parallel Bus Revealed: Part 2, Earl Rice

EXTENB does not enable the external hardware as described. If EXTENB is high, the external hardware is signaled that the computer is accessing the RAM. For example, EXTENB is not set high if an address in D1xx is called.

Sources

800XL MMU source code by Bob Woolley
Altirra Hardware Reference Manual by Avery Lee
Antic Vol. 3 No. 9 - January 1985 - Super Utilities - Toolbox
Parallel Bus Revealed, Transfer 100,000 Bytes Per Second, Earl Rice
Antic Vol. 3 No. 10 - February 1985 - Money Mastery – Toolbox
Parallel Bus Revealed: Part 2, Earl Rice
Antic Vol. 3 No. 11 - March 1985 - Ultimate Printer Guide – Toolbox
Parallel Bus Revealed: Part 3, Earl Rice
Antic Vol. 3 No. 12 - April 1985 - Computer Frontiers – Toolbox
Parallel Bus Revealed: Part 4, by Earl Rice
Atari 1090/1 XL 64k RAM Card Owner's Guide
Atari XL Expansion System, Technical Specification, Revision A, 5/11/84
Mapping the Atari (Revised Edition) by Ian Chadwick
Mechanical Specifications Blueprint for 1090 XL 80 Column Card
Mechanical Specifications Blueprint for 1090 XL Z80 Card
Schematic diagram of the Atari 600XL
Schematic diagram of the Atari 800XL
Schematic diagram of the Atari 130XE
Schematic diagram of the Atari 1090 XL Expansion System
Synertek, SY6500 datasheet
Texas Instruments datasheet for the SN74LS123 Retriggerable Monostable Multivibrators.
The Software Implementation of Parallel Device Handlers and Drivers
Revision B, May 22, 1984