The Software Implementation of Parallel Device Handlers and Drivers, 1983-03-18 Version, Unknown Revision

The document itself begins on the next page.

Document source:

Original backup tapes owned by Dutchman2000, obtained by Atarimania.
Documentary research and PDF layout by Laurent Delsarte.
Note that these backup tapes contain A LOT of information spread out in many folders, meaning it will take time to process the important bits.

Document identification:

Original file name:	NORDIN.00025.DOC extracted from CEO.01JUN84
Title of document:	The Software Implementation of Parallel Device Handlers and Drivers, 1983-03-18 Version, Unknown Revision
Author(s):	Rick K. Nordin (Richard K. (Hud) Nordin)
Original file date:	1983-03-18
Type of document:	Memo
Target audience:	Internal
Status:	Draft
Reference (Atari):	(unknown)
Reference (Laurent Delsarte):	For any discussion, this PDF has been given the reference **BKUP-1983-03-18-MEMO-0006A-E** which should be quoted in any communication.
Tags:	#Atari #8bit #6502 #600XL #800XL #1201XL #1250XLD #1400XL #1450XLD #Parallel #PBI #Handler #Driver

Comments:

This version, the oldest, dating from March 1983, contains the entire structure but only a few paragraphs have been written.
Note the mention of the 1201XL and 1250XLD, which do not yet have their definitive commercial name (although they were never marketed in the end).

In the “original version” (the “raw text version”), the author chose to express certain hexadecimal numbers with the postfix “H”, uppercase. Some, not all. For the sake of clarity, I've removed the “H” and added the prefix “$” in front of all hexadecimal numbers.

This page intentionally left blank

The Software Implementation of Parallel Device Handlers and Drivers, 1983-03-18 Version, Unknown Revision

R. K. Nordin
March 18, 1983

Table of Contents

1. Overview

Computers in the Atari 600XL / 800XL / 1201XL[1] / 1250XLD[2] 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 8 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 device numbers 0, 1, 6, and 7 are reserved for Atari.)

The ROM in every parallel device is two kilobytes, addressed $D800 through $DFFF, in the same address space as the Operating System's BCD Floating-Point Package.

When a parallel device is selected, addresses $D800 through $DFFF refer to locations within the device ROM. Selection of a device and its ROM is accomplished by setting the corresponding bit in hardware register $D1FF. For instance, storing $04 at $D1FF selects device 2. (If more than one bit is set, the device with the lowest corresponding number is selected.) Storing $00 at $D1FF selects the Floating-Point Package and deselects all parallel devices.

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.

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-instating the Floating-Point Package.

A parallel device may be selected for the following four reasons:
1. Initialization
2. Processing of a Device Handler request
3. Processing of a low-level Serial I/O (SIO) type request
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 within the ROM.

In support of parallel device handlers and drivers, the Operating System apportions 512 bytes of RAM, addressed $D600 through $D7FF, to the 8 possible devices.

1 [[The 1201XL was later known as the Atari 1400XL, never released, quickly discarded]]
2 [[The 1250XLD was later known as the Atari 1450XLD, never released, several prototypes' iterations]]

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 OS 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.

Parallel Device ROM	Mandatory?	Data Table
$D800 - $D801	Optional	ROM Checksum (low byte, high byte)
$D802	Optional	Revision Number
$D803	**Mandatory**	ID Number 1 Value = $80
$D804	Optional	Name or Type
$D805 - $D807	**Mandatory**	Low-level I/O Vector Value = JMP address
$D808 - $D80A	**Mandatory**	IRQ Handler Vector Value = JMP address
$D80B	**Mandatory**	ID Number 2 Value = $91
$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 address
$D81C	Optional	Not Used Value = 0

It should be noted that Device Handler Vector Table ($D80D through $D81B) has the same format as other Operating System Resident Handler (e.g., Printer Handler) Vector Tables.

3. General device selection process

Four registers are used in the parallel device selection process:
1. PDVS ($D1FF), Parallel device select hardware register
2. SHPDVS ($0248), Parallel device select shadow
3. PDVMSK ($0247), Parallel device mask
4. PDIMSK ($0249), Parallel device IRQ mask

In general, the device selection process is similar for all cases. The Operating System selects each device and transfers control to the appropriate routine within the device. If during this process the Operating System discovers that the remaining devices need not be selected – for instance, if the latest device selected performed the desired function – then the selection process is terminated.

Throughout the selection process – including the restoration of the Floating-Point Package when the selection process is complete – 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 value is not available by reading PDVS. For instance, if code within a device ROM needs to know the device's number, it can inspect SHPDVS to determine which device is currently selected.

The parallel device mask, PDVMSK, is used to control the selection process in all but the case of initialization. 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 mask, PDIMSK, is used only in the case of IRQ handling. Parallel device *n* will be selected to handle its IRQ only if bit *n* is set in PDIMSK – as well as in PDVMSK. Again, the Operating System does not set PDIMSK. It is the responsibility of each device's initialization routine to set the PDIMSK bit corresponding to the device's number if the device driver is to handle IRQ's. PDIMSK is provided for software control of device selection in the case of IRQ handling. Since IRQ handling is an extremely time-sensitive process, it may be desirable to disable the IRQ handling for a parallel device.

4. Device initialization interface

(this section is blank)

5. Device handler interface

(this section is blank)

6. Low level device I/O interface

(this section is blank)

7. Device IRQ handler interface

(this section is blank)

8. RAM availability

(this section is blank)