IOP ADB Driver ERS

G D

Rev. 1.0 A1 1/3/90

Introduction

The Modern Victorian architecture, and Four Square and F19 implementations, contains two Input Output Processors (IOPs), formerly called Peripheral Interface Controllers (PICs) which are programmable input / output processors that have a shared memory interface with the main CPU (68030). By off loading some of the input / output tasks to the IOPs, the main CPU will have more free cycles and better performance in a multitasking environment. On the current IOP based CPU projects, one IOP will be connected to a SWIM disk interface chip. When the floppy disk is not in use, this IOP is fairly idle, and can be used to directly control the Apple Desktop Bus (ADB) which supports all input devices like mice and keyboards. This produces both a cost savings and a performance improvement by eliminating the need for, and cost of, the ADB transceiver processor used on the Macintosh II, and by reducing the amount of processing and number of interrupts that the 68030 performs. The basic message passing protocol between the Main CPU and the IOP is described in the *IOP Manager ERS* document. This document will describe the format of the messages used to communicate between the Macintosh ADB Manager and the IOP based ADB driver. Since the existence of IOP Manager is model dependent, and given that no user written code should ever execute on this IOP, or need to call the IOP Manager, the information in this document should not be documented outside of Apple Computer, specifically, I feel that **this information should NOT appear in *Inside Macintosh*.**

IOP Based ADB Functionality

The ADB Manager for IOP ADB based machines will provide the same interfaces and functionality as the Macintosh II ADB Manager, but will be modified to pass messages to an IOP based ADB Driver, instead of accessing the ADB transceiver processor through VIA1. Although the code will be modified, the functionality of the ADB Manager will be unchanged. Additionally, the code for the mouse and keyboard drivers will not need to change, since they use the ADB Manager to communicate with their devices.

In addition to processing the low level ADB transactions, and Auto Polling, the IOP based ADB driver also handles Service Request (SRQ) Polling without intervention from the 68030. SRQ polling usually occurs when alternating between multiple input devices (eg. switching between typing and moving the mouse), and requires polling all of the connected devices, searching for the device that needs service. The IOP code is written to optimize this process, by searching in Most Recently Used (MRU) order, so that devices that are rarely used will be searched last. This of course is not a real advantage unless you have three or more ADB devices (most systems have exactly two), but it will look great in some marketing spec sheet ("Advanced MRU based SRQ search algorithm.").

Compatibility Impact

Since the modified ADB Manager and IOP based ADB Driver will implement exactly the same functionality as the Macintosh II ADB Manager, any application or driver that runs on the Macintosh

II, and accesses ADB devices through the ADB Manager, or through the Apple Mouse and Keyboard drivers, should not have any compatibility problems with the IOP based implementation.

The ADB Manager has several low memory globals that point to internal routines and data structures, which are not documented in *Inside Macintosh*. Although their usage and structure is mostly unchanged, there are some changes, and any code the was directly accessing any of them may no longer work. This is also the case with the Macintosh Portable Power Manager based implementation of ADB.

There is one area where I will make every attempt to avoid compatibility, and that is performance. As every Macintosh user is aware, when the Floppy Disk is active (especially when formatting a disk), the mouse isn't. This is due to the amount of uninterrupted processor time needed by the SONY driver. This situation can still occur on this IOP, since it is also servicing the floppy disks, but I am confident that with some fine tuning of the code, taking advantage of the DMA features of the IOP, and co-operation between these two IOP based drivers, a significant improvement over the "look and feel" of the Macintosh II can be achieved.

Message Passing Overview

The 680XX based ADB Manager will communicate with the IOP based ADB Driver using the message passing interfaces provided by the IOP Manager. The format and contents of these messages is described below. Developers should access the ADB devices by using the ADB Manager or higher level keyboard or mouse drivers, and should NOT communicate directly with the IOP based ADB driver, just as they shouldn't directly access the ADB transceiver chip on other system. The information below is to be used internal to Apple for the Macintosh ADB Manager development, and possibly for the A/UX keyboard and mouse driver development.

An IOP based driver can receive messages from the main CPU, and will notify the main CPU when processing of the message request is completed. It can also return information in the completed message. This method will be used for the main CPU to request ADB operations to be performed. The main CPU to IOP message number 3 will be used for this purpose.

It is also possible for the IOP based driver to send messages to the main CPU, which will be used to notify the main CPU that an ADB device has some input data available. The IOP to main CPU message number 3 will be used for this purpose.

IOP ADB Driver Message Format

Offset	Length	Direction	Description
$00	1	In/out	Flags
$01	1	In/Out	Data Count
$02	1	In/Out	ADB command
$03	2...8	In/Out	ADB data / Poll Enable Bit Mask
$0B	21	In/Out	unused

The Data Count byte indicates the number of valid bytes in the ADB data field, which may be zero, or in the range 2 to 8 bytes. Its size does not include the ADB command or any of the other bytes in this message. The ADB command is either the command byte to be sent over the ADB bus, or the command that was sent that corresponds to the data that was received from an ADB device. The bits

of the Flags byte are used as follows.

Bit 7 Explicit Command = 1, Implicit Command = 0
Bit 6 Auto/SRQ Polling Enabled = 1, Disabled = 0
Bit 5 Update Device Polling Bit Mask = 1, No Update = 0
Bit 4 unused
Bit 3 unused
Bit 2 Service Request Detected = 1, no SRQ = 0
Bit 1 Receive Timeout = 1, Received Valid data = 0
Bit 0 unused

If a message sent to the IOP has an Implicit Command indicated (Bit 7 = 0), and the Update Device Polling Bit Mask flag is set (Bit 5 = 1), then the ADB Data field should contain a 16 bit mask indicating which device addresses should be Auto/SRQ polled. The most significant bit corresponds to device address 15, and the least significant bit corresponds to device address 0. If the bit is set, it indicates that this device should be polled, and if it is cleared, it indicates that polling should be avoided for this device address.

How the Messages are Used

In order to process _ADBop traps, the ADB Manager can pass an Explicit ADB transaction request to the IOP ADB driver in two ways. The first, and most common way is for the ADB Manager to initiate an Main CPU to IOP message request, setting the flag bits in the message to indicate that it is an Explicit Command, and also pass the ADB Command byte, and any associated data (for an ADB listen command), setting the data count to reflect the number of bytes of data to be sent. When the IOP receives this message, it will save the information about the requested transaction (but will not immediatly proeess it), and acknowledge the message immediatly. Until this transaction is actually processed, the ADB Manager must queue any new _ADBop requests.

When the IOP actually finishes processing the request, it will initiate an IOP to Main CPU message request, passing back the results of that transaction (the Explicit Command bit will be set in the flags byte, and the Service Request and Timeout bits will be set appropriately). When the ADB manager receives this message, it can check its queue, and if any new requests were queued, it can pass them to the IOP in the reply to the message completion (this is the second way to make a request), otherwise, it must clear the Explicit Command bit when it sends the reply, and may want to enable Auto Polling.

When the IOP ADB driver is Auto Polling, it will poll the most recently used device (which has it's polling enable bit set) until it receives data, or until another device asserts Service Request. To handle a service request, it will start polling all of the other devices (which have their polling enable bit set) in most recently used order until it hits one that returns data. If after polling all of the enabled devices, SRQ is active, and no data was received from any of the devices, SRQ polling will continue, polling ALL device addresses, ignoring the enable mask. Once the IOP actually receives data from a device, it will notify the Main CPU, passing a message which will have the Explicit Command bit cleared, and the Auto Poll bit set. The Device address in the ADB Command byte of the message will indicate the device that sent the data, and the byte count will indicate how may bytes of the receive data buffer are valid.

This method will reduce the interrupt traffic on the Main CPU when auto polling, to one interrupt each time a device has data, and requires no interrupts to direct the service request polling. This will hopefully reduce some system overhead.