IOP SWIM Driver ERS

C 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. This IOP will contain the code to support the SONY Driver, which supports all disk devices that can be connected to the SWIM chip. 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 with the IOP based disk 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*.**

SONY Driver Functionality

The SONY Driver for SWIM IOP based machines will provide the same functionality as the Macintosh IIx SONY Driver (including FDHD disk drive support), but will be modified to pass messages to an IOP based SWIM Driver, instead of directly accessing the IWM or SWIM hardware. All of the driver OPEN, CLOSE, CONTROL, STATUS and PRIME calls will take the same parameters and return the same results on both the Macintosh IIx and SWIM IOP based machines.

The Macintosh IIx SONY Driver allocates its track cache (9K Bytes) in the system heap. On SWIM IOP machines, the track cache will reside in IOP RAM, freeing up some system heap space, and may be larger (18K Bytes), so that it can cache two tracks (one from each side of the disk). This has both a positive and negative performance impact. On the positive side, it frees up 9KB of system heap RAM, which can be put to better uses, and it has a larger cache so the chances of a hit in the cache are greater. On the negative side, since the cache is managed by, and resides in the IOP, you must pay for all of the overhead associated with cross processor message passing and data transfer, even when you hit in the cache, but overall, I think that the positives will out weigh the negatives.

The Macintosh IIx SONY Driver that supports the SWIM chip, uses the IWM half of the SWIM chip for all access to GCR disks (400K/800K), and switches the SWIM chip into ISM mode when accessing MFM disks (720K/1440K). The IOP SWIM Driver will use ISM mode exclusively, and not use IWM mode at all. The reasons for using just the ISM are as follows. Only ISM mode provides the hardware handshake signals used by the IOP DMA hardware, the IWM has no handshake signal. The only way to access the HeadSelect control line from the IOP is through a register in the ISM register set, it cannot be accessed in IWM mode. The ISM provides a much nicer interface to software than the IWM, which will make the driver code, faster, smaller, and easier to understand, than if the IWM mode were used. Mode switching is not required if only one mode is used, and ISM mode is the only mode that will support all of formats and drives that we need to support, IWM mode only supports 400K/800K GCR and HD-20.

There is some risk associated with this decision. No other project has used the GCR capabilities of the ISM mode of the SWIM chip within Apple, since prior machines use IWM mode exclusively for GCR mode, and just use ISM mode to access MFM encoded disks. However, I understand that the consulting firm that designed the ISM has tested, and does use the GCR capabilities of the ISM successfully. Additionally, the implementation of the IOP SWIM driver has progressed to the point where GCR reads and writes are implemented, and they appear to be working without any problems.

There is one extension that is being considered if development resources exist, and if it appears to be a desirable extension. That extension is the addition of HD-20 (the old slow non-SCSI ones) support. Support for the HD-20 was not included in the Mac II ROMs, but was later implemented in a RAM based version of the SONY Driver, which along with a special cable (the Mac II does not have an external disk connector), or using the built in external floppy connector on the Macintosh SE-30 or IIcx, allowed access to the HD-20 (the intention was to allow users who upgraded to Mac IIs to transfer their data off of old HD-20s onto newer SCSI hard disks). If this type of upgrade path is still popular, the only way that it could be supported would be through the IOP based driver, it would be best to have it implemented in the standard driver, instead of having two drivers. If development resources do not exist, then this feature may not be implemented at all.

Compatibility Impact

Since the new SONY Driver and IOP SWIM Driver will implement exactly the same functionality as the Macintosh IIx SONY Driver, any application that runs on the Macintosh IIx, and accesses the SONY Driver through the Device Manager, should not have any compatibility problems with the IOP based implementation.

The SONY Driver has several low memory globals that point to internal routines and data structures used by the driver. This was so that the ROM code could jump indirect through these RAM locations, which could be patched to fix potential bugs in the ROM code. If there are any applications that use these patch vectors to directly call these internal SONY Driver routines (or worse yet, to change the way these routines work), or access an of the internal data structures of the SONY Driver, they will probably not run correctly on SWIM IOP based machines. *Inside Macintosh* does not document these data structures, vectors, or the routines that they point to, so anyone who is doing this is probably also doing other sleazy, model dependent things and is asking for trouble.

Many Copy Protection schemes access the disk in ways other than by accessing the SONY Driver through the Device Manager, and may do thing such as patching portions of the SONY Driver, or directly accessing the IWM hardware. Since the 680XX CPU, or SONY Driver on SWIM IOP based machines will not be accessing the disk interface hardware directly, **it is to be expected that many copy protected applications WILL NOT RUN on SWIM IOP based machines!**

It may be possible to support some copy protected software, if they are just making minor patches to the SONY Driver, such as changing the values of Address Marks, or the GCR encoding tables. This type of support , if implementable, will not be added until after the driver is fully implemented, and debugged, and we are able to determine which applications might benefit from this type of kludge. I feel that this should only be done if there are some very strong business reasons to support it. Another approach might be to work with the copy protection developers (and to be fair, the developers of copy protection copying/defeating software), to establish a driver interface to accomplish what they need to do, and implementing it in this and all future SONY drivers.

Pages 2 and 3 of Macintosh Technical Note #2 (May 1986) and pages 21, 24 and 25 of Macintosh Technical Note #117 (March 1987) address using undocumented low memory globals, directly

accessing the hardware, and copy protection. It's not like we haven't been warning developers that something like the IOPs might come along someday.

Message Passing Overview

The 680XX based SONY Driver will communicate with the IOP based SWIM 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 floppy drives by using the File System or the SONY Driver, and should NOT communicate directly with the IOP based SWIM driver, just as they shouldn't directly access the IWM/SWIM chips on other system. The information below is to be used internal to Apple for the Macintosh SONY Driver development, and possibly for the A/UX floppy 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 disk operations to be performed. The main CPU to IOP message number 2 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 a disk has just been inserted, or manually ejected. The IOP to main CPU message number 2 will be used for this purpose.

Additionally, the IOP based SWIM driver will request data movement between the IOP and main CPU memories, using IOP to main CPU message number 1, as described in the *IOP Manager ERS* document.

Main CPU to IOP SWIM Driver Request Kinds

By convention, the first byte of any message associated with the IOP based SWIM driver will be a request kind. The request kinds for the main CPU to IOP SWIM driver messages are as follows. The error codes returned by these calls are the same error codes that the current Macintosh SONY driver returns.

- $01 – Initialize
- $02 – ShutDown
- $03 – StartPolling
- $04 – StopPolling
- $05 – SetHFSTagAddr
- $06 – DriveStatus
- $07 – Eject
- $08 – Format

- $09 – FormatVerify
- $0A – Write
- $0B – Read
- $0C – ReadVerify
- $0D – CacheControl
- $0E – TagBufferControl
- $0F – GetIcon
- $10 – DiskDupInfo
- $11 – GetRawData

Initialize

Offset	Length	Direction	Description
$00	1	In	Initialize request byte ($01)
$01	1	In/Out	unused
$02	2	Out	Error Code
$04	28	Out	List of Drive Kinds

This needs to be the first call made to the IOP SWIM driver, and is normally made by the SONY driver when it is first opened at system startup time. It causes the IOP SWIM driver to initialize its hardware and data structures, and return a list of drives that are connected to the system, what kind of drive they are, and an indication of the drive number to use to refer to them in future calls.

The driver can theoretically support 28 drives, numbered 0 through 27. The corresponding byte in the List of Drive Kinds returned by the IOP indicates the drive kind for each of the 28 possible drives. The encoding of the Drive Kind byte is the same encoding used by the SONY driver control call 23, as follows.

0 No such drive
1 Unspecified drive
2 400K only drive
3 400K/800K drive
4 400K/800K/720K/1440K drive (FDHD)
5 Reserved
6 Reserved
7 Hard Disk 20

ShutDown

Offset	Length	Direction	Description
$00	1	In	ShutDown request byte ($02)
$01	1	In/Out	unused
$02	2	Out	Error Code
$04	28	In/Out	unused

This call used to shutdown the IOP based SWIM driver. It is not currently implemented, or needed, and may be removed in the future.

StartPolling

Offset	Length	Direction	Description
$00	1	In	StartPolling request byte ($03)
$01	1	In/Out	unused
$02	2	Out	Error Code
$04	28	In/Out	unused

This call used to enable the IOP based SWIM driver polling for disk insertion / eject requests. The Macintosh OS SONY driver always wants polling to be enabled, and starts polling at driver open time.

StopPolling

Offset	Length	Direction	Description
$00	1	In	StopPolling request byte ($04)
$01	1	In/Out	unused
$02	2	Out	Error Code
$04	28	In/Out	unused

This call used to disable the IOP based SWIM driver polling for disk insertion / eject requests.

SetHFSTagAddr

Offset	Length	Direction	Description
$00	1	In	SetHFSTagAddr request byte ($05)
$01	1	In/Out	unused
$02	2	Out	Error Code
$04	4	In	HFS Tag buffer address

$08	24	In/Out	unused

This call used to support the extended 20 byte HFS file system tags that are used on the HD-20. The first 12 bytes are also used by the 400K/800K floppy formats, and are passed in the message buffer, but there was not enough room for the 8 bytes of extended info, so instead the IOP is notified of the address of those 8 bytes, and when the IOP needs to access them it will send a move request to the main CPU requesting them. If this call is never made, or if a buffer address of zero is passed to it, then the IOP will not make requests for HFS tag data.

DriveStatus

Offset	Length	Direction	Description
$00	1	In	DriveStatus request byte ($06)
$01	1	In	Drive number
$02	2	Out	Error Code
$04	2	Out	Track
$06	1	Out	Write Protect
$07	1	Out	Disk In Place
$08	1	Out	Installed
$09	1	Out	Sides
$0A	1	Out	Two Sided Format
$0B	1	Out	New Interface
$0C	2	Out	Disk Errors
$0E	4	Out	Drive Info
$12	1	Out	MFM Drive
$13	1	Out	MFM Disk
$14	1	Out	MFM Format
$15	1	Out	Disk Controller
$16	2	Out	Current Format (bit mask)
$18	2	Out	Formats Allowed (bit mask)
$1A	4	Out	Disk Size (blocks)
$1E	1	Out	IconFlags
$1F	1	Out	unused

Returns Drive Status information for the disk and the drive specified. The meanings of most of these bytes are the same as those used by many of the control and status calls to the SONY driver. Bytes $02 through $0D are used for status call 8. Bytes $0E through $11 are used for control call 23. Bytes $12 through $15 are used for status call 10. Bytes $16 through $18 are two 16 bit masks which are used to indicate the current format and formats allowed, for status call 6. The bits have the following meanings.

Bit 0	HD-20 disk format
Bit 1	400K GCR format
Bit 2	800K GCR format
Bit 3	720K MFM format
Bit 4	1440K MFM format

The Disk Size in Bytes $1A through $1D also used for status call 6. Byte $1E is used for control calls 21 and 22, to determine if the IOP based driver needs to supply the icons, or if the default icons used by the SONY driver is correct. This is used for the HD-20 drive which supplies its own icons. The bits have the following meanings.

Bit 0 0 - use default Media icon, 1 - call IOP for Media icon
Bit 1 0 - use default Drive icon, 1 - call IOP for Drive icon

Eject

Offset	Length	Direction	Description
$00	1	In	Eject request byte ($07)
$01	1	In	Drive number
$02	2	Out	Error Code
$04	28	Out	Same as DriveStatus request

Ejects the disk from the drive specified, and returns updated drive status reflecting the state of the drive after the disk has been ejected. See the DriveStatus message for the meanings of the returned status bytes.

Format

Offset	Length	Direction	Description
$00	1	In	Format request byte ($08)
$01	1	In	Drive number
$02	2	Out	Error Code
$04	28	Out	Same as DriveStatus request
$04	2	In	Format Kind
$06	1	In	Format Byte for Sector Header (0 = use default)
$07	1	In	Interleave (0 = use default)
$08	4	In	Sector Data Main CPU RAM address (0 = use default data)
$0C	4	In	Tags Main CPU RAM address (0 = use default tags)

Formats the disk in the drive specified, using the specified format kind, which the bit number of the format kind to use, as described in the DriveStatus request. Tag and Data Buffer pointers may be supplied to allow formatting and writing to the entire disk in just one pass. The sector interleave factor may also be specified, as well as the FormatByte field of the sector headers. These options are provided to support the Disk Duplicator application. After the format is complete, it returns updated drive status reflecting the state of the drive after the disk has been formatted. See the DriveStatus message for the meanings of the returned status bytes.

FormatVerify

Offset	Length	Direction	Description

$00	1	In	FormatVerify request byte ($09)
$01	1	In	Drive number
$02	2	Out	Error Code
$04	28	Out	Same as DriveStatus request

Verifies that the disk in the drive specified is correctly formatted, and that each block of the disk can be successfully read. Note that this call will not perform any error retrys, all errors are fatal. This is because it is expected to be used immediatly after formatting a disk, to determine that the disk can be used reliably, and soft errors would indicate unreliable media.

Write

Offset	Length	Direction	Description
$00	1	In	Write request byte ($0A)
$01	1	In	Drive number
$02	2	Out	Error Code
$04	4	In	Main CPU RAM address
$08	4	In	Disk Block Number
$0C	4	In	Block Count
$10	12	In/Out	Tag Data
$1C	4	In / Out	unused

Writes *Block Count* disk blocks, using data starting at *Main CPU RAM address*, to the disk in the drive specified by *Drive number*, starting at *Disk Block Number*, using *Tag Data* for the first block written, and updating it for each successive disk block. When the write is complete, error status is returned in *Error Code*, and *Tag Data* is updated to reflect the tags of the last block transferred.

Read

Offset	Length	Direction	Description
$00	1	In	Read request byte ($0B)
$01	1	In	Drive number
$02	2	Out	Error Code
$04	4	In	Main CPU RAM address
$08	4	In	Disk Block Number
$0C	4	In	Block Count
$10	12	Out	Tag Data
$1C	4	In / Out	unused

Reads *Block Count* disk blocks, into the data area starting at *Main CPU RAM address*, from the disk in the drive specified by *Drive number*, starting at *Disk Block Number*. When the read is complete, error status is returned in *Error Code*, and *Tag Data* is updated to reflect the tags read from the last block transferred.

ReadVerify

Offset	Length	Direction	Description
$00	1	In	ReadVerify request byte ($0C)
$01	1	In	Drive number
$02	2	Out	Error Code
$04	4	In	Main CPU RAM address
$08	4	In	Disk Block Number
$0C	4	In	Block Count
$10	12	Out	Tag Data
$1C	4	In / Out	unused

Reads *Block Count* disk blocks from the disk in the drive specified by *Drive number*, starting at *Disk Block Number* and compares the data from disk to the data in the data area starting at *Main CPU TAG address*. When the read is complete, error status is returned in *Error Code*, and *Tag Data* is updated to reflect the tags read from the last block transferred.

CacheControl

Offset	Length	Direction	Description
$00	1	In	CacheControl request byte ($0D)
$01	1	In/Out	unused
$02	2	Out	Error Code
$04	1	In	Cache Enable Flag
$05	1	In	Cache Install Flag
$06	26	In / Out	unused

Controls the track caching feature. The meaning of two parameter bytes are the same as the *csParam* bytes that are passed to control call 9 in the SONY driver.

TagBufferControl

Offset	Length	Direction	Description
$00	1	In	TagBufferControl request byte ($0E)
$01	1	In/Out	unused
$02	2	Out	Error Code
$04	4	In	Main CPU RAM address
$08	24	In / Out	unused

Specifies the *Main CPU RAM address* of an alternate tag buffer. An address of zero is used to disable

the alternate tag buffer. The meaning of parameter is the same as the *csParam* bytes that are passed to control call 8 in the SONY driver.

GetIcon

Offset	Length	Direction	Description
$00	1	In	GetIcon request byte ($0F)
$01	1	In	Drive number
$02	2	Out	Error Code
$04	4	In	Main CPU RAM address
$08	2	In	Icon Kind (0=Media, 1=Drive)
$0A	2	In / Out	unused
$0C	2	In	Max Byte Count
$0E	18	In / Out	unused

Specifies the *Main CPU RAM address* of a buffer to receive the icon data, the *Icon Kind* and the *Max Byte Count* of the receiving buffer. This call should only be used when the Drive Status indicated that the IOP should be called for icon data.

DiskDupInfo

Offset	Length	Direction	Description
$00	1	In	DiskDupInfo request byte ($10)
$01	1	In/Out	unused
$02	2	Out	Error Code
$04	2	Out	Version Number ($0410)
$06	1	Out	Sector Header Format Byte
$07	25	In / Out	unused

Returns the information needed by the Disk Duplicator Program. The version number indicates that the driver support this level of functionality. The Sector Header Format Byte is the format byte field from the last sector that was accessed on any drive.

GetRawData

Offset	Length	Direction	Description
$00	1	In	GetRawData request byte ($11)
$01	1	In	Drive number
$02	2	Out	Error Code
$04	4	In	Clock Bits Buffer Main CPU RAM address
$08	4	In	Data Bytes Buffer Main CPU RAM address

$0C	4	In / Out	Byte Count Requested / Bytes Transferred
$10	2	In	Search Mode
$12	2	In	Cylinder Number
$14	1	In	Head Number
$15	1	In	Sector Number

Allows reading of the RAW data from the disk, at the specified cylinder and head. The parameters correspond to those used by Control Call 18244 in the Macintosh Sony Driver.

IOP SWIM Driver to Main CPU Request Kinds

By convention, the first byte of any message associated with the IOP based SWIM driver will be a request kind. The request kinds for the IOP SWIM driver to main CPU messages are as follows.

- $01 – DiskInserted
- $02 – DiskEjectButton
- $03 – DiskStatusChanged

DiskInserted

Offset	Length	Direction	Description
$00	1	Out	DiskInserted request byte ($01)
$01	1	Out	Drive Number
$02	2	Out	Error Code
$04	28	Out	Drive status information

Informs the main CPU that a disk has just been inserted in the drive specified by *DriveNumber*, and the new status for that drive is returned in *Drive status information*, which has the same format as the DriveStatus request.

DiskEjectButton

Offset	Length	Direction	Description
$00	1	Out	DiskEjectButton request byte ($02)
$01	1	Out	Drive Number
$02	2	Out	Error Code
$04	28	Out	Drive status information

Informs the main CPU that the eject button has been pressed on the drive specified by *DriveNumber*, requesting the system to eject that disk. The current status for that drive is returned in *Drive status information*, which has the same format as the DriveStatus request.

DiskStatusChanged

Offset	Length	Direction	Description
$00	1	Out	DiskStatusChanged request byte ($03)
$01	1	Out	Drive Number
$02	2	Out	Error Code
$04	28	Out	Drive status information

Informs the main CPU that the status of the drive specified by *DriveNumber*, may have changed, and the main CPU copy of the status may now be incorrect. The new status for that drive is returned in *Drive status information*, which has the same format as the DriveStatus request.