SCSI Connection FPI

Darryl Chan

1 Revision History

12/1/95	0.1	Initial draft
12/20/95	0.2	Modified with feedback from IO team.
2/22/96	1.0	Additions and changes made during development and debug.

2 Related Document

1. Inside Macintosh - Devices
2. SCSI Plug-in Design Document by Martin Minow
3. Driver Family Matching Design Document by Pradeep Kathail
4. Inside Macintosh - Designing PCI cards and Drivers for Power Macintosh Computers
5. ANSI Standard for SCSI-II (ANSI X3.232-199x)
6. Next Generation MacOS I/O Architecture, 3rd Drafts. by Holly Knight et. al.

3 Vision For SCSI Connection FPI

Design Goals

- Change from a state-less to a state-full interface with access control.
- Eliminate the usage of the PB in the family FPI.
- Reduce the complexity of having to build a PB.
- Simplify the process of identifying an invalid parameter.
- Reduce memory management responsibility from the client and centralize it to the family.
- Improve performance for request handling.
- In order to shorten the development window, the SCSI Family and SIM will continue to utilize the PB structure to pass information. In other words, the SCSI Family will translate the input parameter and build a PB to communicate with the appropriate SIM.

Assumptions

- The SCSI Family does not support the original SCSI Manager commands (an interface where the client expects to see all SCSI phase changes).
- The SCSI Family does support the SCSI Manager 4.3 interface; however, it is supported through the compatibility layer only.
- The SCSI Family will handle memory management for the PB that is used to pass request information to the SIM.

4 FPI Interface

The following is a list of the new connection protocol for the SCSI Family only. The compatibility interface is identical to the SCSI Manager 4.3 as defined in the *Inside Macintosh - Devices*.

Device Iterator and Bus Information

1. SCSIGetInfo

OSStatus SCSIGetInfo (&BusCount, &DeviceCount)

This command inquires the number of buses and device attached that are currently instantiated and discovered by the SCSI Family.

BusCount - number of buses (HBAs) found.

DeviceCount - number of devices attached to the buses.

2. SCSIGetDeviceList

OSStatus SCSIGetDeviceList (BusID, &SCSIDevList, &DeviceCount)

This command returns the number of devices (**DeviceCount**) attached to this **busID**. In addition, it returns a device description list (**SCSIDevList**).

BusID - the bus ID number

DeviceCount - number of devices attached to this bus

SCSIDevList - device description structure for each attached device defined as follow:

```
struct  SCSIDevList
{
        RegEntryRef;
        DeviceIdent;
};
```

Note: this iteration model will mostly be change in the next revision. This only provide a sample of what is available to discovery what is device is attached to what bus.

3. SCSIBusInquiry

OSStatus SCSIBusInquiry (ConnectionID, &SCSIBusInfo)

The client uses this command to inquiry about the SIM and hardware characteristics. This is a synchronous command.

SCSIBusInfo is nearly identical to the **SCSIBusInquiryPB** in **SCSI Mgr 4.3** with a few exceptions of several fields that related to the PB specification.

```
struct  SCSIBusInfo
{
        scsiEngineCount;        /* <- Number of engines on HBA */
        scsiMaxTransferType;    /* <- Number of transfer types for this HBA*/
        scsiDataTypes;          /* <- which data types are supported by this SIM
*/
        scsiBIReserved4;        /* <- */
        scsiFeatureFlags;       /* <- Supported features flags field */
        scsiVersionNumber;      /* <- Version number for the SIM/HBA */
        scsiHBAInquiry;         /* <- Mimic of INQ byte 7 for the HBA */
        scsiTargetModeFlags;    /* <- Flags for target mode support */
        scsiScanFlags;          /* <- Scan related feature flags */
        scsiSIMPrivatesPtr;     /* <- Ptr to SIM private data area */
        scsiSIMPrivatesSize;    /* <- Size of SIM private data area */
        scsiAsyncFlags;         /* <- Event cap. for Async Callback */
        scsiHiBusID;            /* <- Highest path ID in the subsystem */
        scsiInitiatorID;        /* <- ID of the HBA on the SCSI bus */
        scsiBIReserved0;        /**/
```

Copyright © 1995-96 Apple Computer, Inc. ***Confidential***

```
        scsiBIReserved1;        /* <- */
        scsiFlagsSupported;     /* <- which scsiFlags are supported */

        scsiIOFlagsSupported;   /* <- which scsiIOFlags are supported */
        scsiWeirdStuff;         /* <- */
        scsiMaxTarget;          /* <- maximum Target number supported */
        scsiMaxLUN;             /* <- maximum Logical Unit number supported*/
        scsiSIMVendor[vendorIDLength];/* <- Vendor ID of SIM (or XPT if
bus<FF) */
        scsiHBAVendor[vendorIDLength];/* <- Vendor ID of the HBA */
        scsiControllerFamily[vendorIDLength];/* <- Family of SCSI Controller
*/
        scsiControllerType[vendorIDLength];/* <- Specific Model of SCSI Con-
troller used */
        scsiXPTversion[4];      /* <- version number of XPT */
        scsiSIMversion[4];      /* <- version number of SIM */
        scsiHBAversion[4];      /* <- version number of HBA */
        scsiHBAslotType;        /* <- type of "slot" that this HBA is in*/
        scsiHBAslotNumber;      /* <- slot number of this HBA */
        scsiSIMsRsrcID;         /* <- resource ID of this SIM */
        scsiBIReserved3;        /* <- */
};
```

Connection

4. SCSIOpenConnection

OSStatus SCSIOpenConnection (RegEntryRef, ConnectionType, &ConnectionID)

This opens a new connection to a device or bus in order to inquire or make requests. When the **RegEntryRef** is referenced to a device node in the NameRegistry, the connection be opened as a device connection. Otherwise, the connection is made as a bus connection. The significance of this lies in the I/O operations. If it is a device connection, the client must use SCSIExecIOCmd to send a request to the SCSI Family. Otherwise, the client must use SCSIExecIOControlCmd to make the same request. The difference between the two commands is the latter requires a DeviceIdent input. See the **I/O Operation** section for further details.

RegEntryRef - should be pointing to the target device of interest.

ConnectionType - is the type of connection, i.e. read_only, read/write, reserved (lock).

ConnectionID - is the ID that used to identify the state and type of connection made by the clients.

5. SCSICloseConnection

OSStatus SCSICloseConnection (ConnectionID)

This terminates the connection for the client.

ConnectionID - is the ID that used to identify the state and type of connection made by the clients.

Note: these functions will be changed when **Arbitration Services** is implemented.

I/O Operations

6. SCSIExecIOCmd

OSStatus SCSIExecIOCmd (ConnectionID, SCSIDataObject, SCSICDBObject, SCSIFlagsObject, &SCSIExecIOResult, &SCSIExecIOTag)

OSStatus SCSIExecIOAsyncCmd (ConnectionID, &KernelNotification, SCSIDataObject, SCSICDBObject, SCSIFlagsObject, &SCSIExecIOResult, &SCSIExecIOTag)

OSStatus SCSIExecIOControlSyncCmd (ConnectionID, SCSIDataObject, SCSICDBObject, SCSIFlagsObject, &SCSIExecIOResult, &SCSIExecIOTag)

OSStatus SCSIExecIOControlAsyncCmd (ConnectionID, &KernelNotification, SCSIDataObject, DeviceIDent, SCSICDBObject, SCSIFlagsObject, &SCSIExecIOResult, &SCSIExecIOTag)

These commands are used to make request to the SCSI device. For bus connections, the client must use the control commands to send requests. For device connection, the client must use the standard ExecIO command set.

KernelNotification is the data structure indicates the IO request model.

SCSIDataObject is the data structure that encapsulates the data field items.

```
Struct  SCSIDataObject
{
        scsiDataPtr,
        scsiDataLength,
        scsiDataType,
        scsiSGListCount
};
```

SCSICDBObject is the data structure that encapsulates the SCSI Command Data Block (CDB) and its associated fields.

```
Struct  SCSICDBObject
{
        scsiCDBLength,
        scsiCDB
};
```

SCSIFlagsObject are option bits that are interpreted by the SIM.

Copyright © 1995-96 Apple Computer, Inc. ***Confidential***

```
struct  SCSIFlagsObject
{
        scsiFlags;
        scsiIOFlags;
        scsiTransferType;
};
```

SCSIExecIOResult is the data structure that encapsulates the numerous return status to the client. If the call is asynchronous, the caller's notification routine will have this pointer as the parameter.

```
struct  SCSIExecIOResult
{
        scsiResult;
        scsiResultFlags;
        scsiSenseLength;                // the actual sense length returned
        scsiDataResidual;               // residual data length
        scsiSense[kMaxAutoSenseByteCount];// maximum sense buffer
        SCSIExecIOTag;
        scsiSCSIstatus;
};
```

Note that this means that even flag notification must not be the only notification type within the kernel notification record.

SCSIExecIOTag is a cookie that identifies this operation. It can be used to Terminate or Abort an IO operation or acquire sense data for a particular operation.

I/O Control

7. SCSIAbortIO / SCSITerminateIO

OSStatus SCSIAbortIO (SCSIExecIOTag)

OSStatus SCSITerminateIO (SCSIExecIOTag)

This command aborts / terminates a queued IO. With these commands, the Family code uses the **SCSIExecIOTag** to match up with the corresponding PB pointer and produces a new PB to send to the SIM. These are synchronous commands.

The **SCSIAbortIO** / **SCSITerminateIO** function cancels the **SCSIExecIO** Request identified by the **IOTag.** If the request has not yet been delivered to the device, it is removed from the queue and the task is completed with a result code of **scsiRequestAborted** / **scsiTerminated**. If the request has already been started, the SIM attempts to send a **ABORT** / **TERMINATE IO PROCESS** message to the device. The function returns the **scsiUnableToAbort** / **scsiUnableToTerminate** result code if the specified request has already been complete.

The SCSITerminateIO function differs from the SCSIAbortIO function only in the message it sends over the bus. **TERMINATE IO PROCESS** is an optional SCSI-2 message

that instructs the device to complete a request normally although prematurely, while attempting to maintain media integrity.

SCSIExecIOTag is the cookie that was acquired at the start of an **SCSIExecIOCmd** operation.

8. SCSIReleaseQCmd

OSStatus SCSIReleaseQ (ConnectionID)

This command releases the SIM device queued which is locked due to an error. This is a synchronous command.

ConnectionID this ID must acquired from a device connection because the device queue independent from the bus (initiator) itself.

9. SCSIClearQueue

OSStatus SCSIClearQueue (ConnectionID)

This command clears the SIM device queued. This is a synchronous command.

ConnectionID this ID must acquired from a device connection because the device queue independent from the bus (initiator) itself.

Note that this command is not yet implemented.

10. SCSIBusReset / SCSIDeviceReset

OSStatus SCSIBusResetSync (ConnectionID)

OSStatus SCSIBusResetAsync (ConnectionID, &KernelNotification, &OSStatus)

OSStatus SCSIDeviceResetSync (ConnectionID)

OSStatus SCSIDeviceResetAsync (ConnectionID, &KernelNotification, &OSStatus)

These commands perform bus and device reset respectively.

ConnectionID this ID must match with the type of operation, otherwise a failure occurs.

KernelNotification is the data structure indicates the IO request model. If this pointer is NULL, the family will operate this as a synchronous operation.

OSStatus this operating result status is useful only for the async commands.

Copyright © 1995-96 Apple Computer, Inc. ***Confidential***

Set Options

11. SCSISetHandshake

OSStatus SCSISetHandshake (ConnectionID, HandshakeObject)

This command sets up the handshaking instruction to perform data transfer. This is a synchronous command.

Open Issue: this may not be supported for Copland because this is a feature provided by the original SCSI Manager.

SCSIHandshakeObject - the handshaking instruction content.

```
struct  SCSIHandshakeObject
{
        scsiHandshake [8];
};
```

12. SCSISetTimeout

OSStatus SCSISetTimeout (ConnectionID, scsiTimeout, scsiSelectTimeout)

This command sets up the time-out parameter of executing an IO. This is a synchronous command.

ConnectionID regardless of the type of connection, the time-out parameter will be set for this SIM.

scsiTimeout this parameter set up the duration for command time-out.

scsiSelectTimeout this parameter set up the duration for selection time-out.

Open Issue: These time-out parameters are set on a per SIM bases only. Is this an acceptable solution?

13. SCSISetIOOptions

OSStatus SCSISetIOOptions (ConnectionID, SCSIIOOptionsObject)

This command sets up the IO option flags in **scsiFlags** and **scsiIOFlags** of the **SCSI_IO** PB. This is a synchronous command.

ConnectionID regardless of the type of connection, the time-out parameter will be set for this SIM.

SCSIIOOptionsObject this parameter is the combined data of **scsiFlags** and **scsiIOFlags** of the **SCSI_IO** PB.

```
struct  SCSIIOOptionsObject
{
        scsiFlags;
        scsiIOFlags;
};
```

Open Issue: need to identify which flags can be globally set and which ones should be set on a per IO bases.

Open Issue: the global flags will be set as a per SIM instance. Is this acceptable?

5 SCSI Plugin Interface

This section describes the SCSI plugin interface. For the most part, there is no changes from the original design. There are a few enhancements that's noteworthy. The instantiation of the plugin has changed with the implementation of DriverFamily-Matching Service. A dispatch table mechanism has been put in place to simplify the process of acquiring all the plugin entry points and it also allows for plugin version comparison for compatibility purposes.

Plugin Initialization

1. SCSIPluginInit

OSStatus SCSIPluginInit (&PluginControlBlock)

This plugin entry point is called by the SCSI Family to initialize the plugin. In addition, the plugin should perform a hardware compatibility test to ensure this plugin is fully function with the hardware (HBA).

PluginControlBlock this structure is used by both the plugin and family to exchange information about the plugin.

```
struct PluginControlBlock {
        ioPBSize;           /* <- size of SCSI_IO_PBs required for this SIM*/
        oldCallCapable;     /* <- true if this SIM can handle old-API calls*/
        busID;              /* -> bus number for the registered bus*/
        simSlotNumber;      /* <- cookie to place in scsiHBASlotNumber (PCI)*/
        simSRsrcID;         /* <- cookie to place in scsiSIMsRsrcID (PCI)*/
        simRegEntry;        /* -> The SIM's RegEntryIDPtr(PCI)*/
        maxTargetID;        // <- max Target ID of this bus
        initiatorID;        /* <- comes from the NVRAM */
        scsiTimeout;        // <- bus time out period
        scsiFlagsSupported;// <- scsiFlags supported by this SIM
        scsiSelectTimeout;  // <- selection time out period
        scsiIOFlagsSupported;// <- scsiIOFlags supported by this SIM
        scsiDataTypes;      // <- scsiDataType supported by this SIM
};
```

The arrows indicate how the information are passed: from family -> to plugin and to family <- from plugin.

Copyright © 1995-96 Apple Computer, Inc. *Confidential*

Start IO

2. SCSIPluginAction

void SCSIPluginAction (*SCSI_PB)

This is the entry point to the plugin to start all I/O operations.

SCSI_PB is identical to the SCSI_PB in SCSI Mgr 4.3

IO Completion

3. SCSIFamBusEventForSIM

OSStatus SCSIFamBusEventForSIM (busID, *busEvent)

This routine is called by the plugin's ISR. It's purpose is to queue up this busEvent in the hardware interrupt level and activate the family glue code in the plugin task level.

busID this indicates which bus has an interrupt and the proper plugin task will be activated.

busEvent this structure is defined by the plugin code and the family has no knowledge of the content.

4. SCSIPluginHandleBusEvent

void SCSIPluginHandleBusEvent (*busEvent)

This routine is called by the family glue code to handle a bus event. The plugin does clean up work here and calls the SCSIFamMakeCallback family routine.

busEvent this structure is defined by the plugin code and the family has no knowledge of the content.

5. SCSIFamMakeCallback

void SCSIFamMakeCallback (*SCSI_PB)

This routine is called by the plugin code (SCSIPluginHandleBusEvent) to indicate that the request has been completed. All IO completions must be handled in this routine. Note that this routine will ALWAYS operate in the context of the plugin task.

This family routine does

- checks for and perform PMFIO or Memlist clean up if necessary
- intervene with Bus Inquiry request with some result changes
- clear PB and return them to the look-aside-list
- reply to client with a result buffer

SCSI_PB is identical to the SCSI_PB in SCSI Mgr 4.3

6 SCSI Plugin Dispatch Table

The IO Team has implemented a new mechanism to instantiate a plugin with DFM. The family expert code calls **DFMLoadPlugin** to load and acquire the **SCSIPlug-inDispatchTable** structure**.** The corresponding plugin must have defined, assigned, and exported this structure. For further details, see the DFM document.

```
enum
{
        kSCSIPluginVersion = 0x02019600// date and version ••• temporary
};

struct  SCSIPluginInfo
{
        UInt32          version;
        UInt32          reserved1;
        UInt32          reserved2;
        UInt32          reserved3;
};

// plugin needs to export this structure
struct  SCSIPluginDispatchTable
{
        SCSIPluginInfo    header;
        SCSIPluginActionEntry scsiPluginAction;// address of SIM action routine
        SCSIPluginHandleBusEventEntryscsiPluginHandleBusEvent;
        SCSIPluginInitEntryscsiPluginInit;
};
```

Copyright © 1995-96 Apple Computer, Inc. ***Confidential***