SM-HBA:
SAS/FC Common HBA API

T10/07-221r0
3 May 2007
Bob Nixon/Emulex
Introduction to SM-HBA

- A work undertaken in T11
  - Thanks to Krithivas (Intel) for editing and most of the technical work
- Major goals
  - Add an API for SAS HBAs
  - Include the richer port structure of SAS
  - Re-model the FC API to maximize common API for common features

- A word of apology: the following is written in pidgin-UML (no that does not mean it is sprinkled with white splats)
SM-API context (with some lapses)

Figure 1 - Context for SM-HBA
FC-HBA “model”

**HBA**
- HBA handle, model #, firmware version, ..., # of Ports

1 1-n

**Port**
- Port index, Port name, Port address, Fabric name, ...
- speed, topology, physical attributes

1 0-n

**Protocol statistics**
- FC-4 TYPE, I/O counters: Input operations, output operations, control operations

**Port statistics**
- traffic counters: frames, words
- link error counters: LESB or equivalent

Heavy boxes are presumed physical devices.

Grey text denotes implicit attributes, typically used to identify an object within some scope.

Green stuff is added complic ... ah ...

Components

Pink text is just to draw your attention.
SM-HBA “model” grew from FC-HBA “model”

- **HBA**
  - HBA handle, model #, firmware version, ..., # of Ports
  - 1
  - 1-n

- **Port**
  - Port index, Port name, Port address, Fabric name, ...
  - 1
  - 1-n

- **Phy**
  - Phy index, speed, topology, physical attributes
  - 1
  - 1-n

- **Protocol statistics**
  - FC-4 TYPE, I/O counters: Input operations, output operations, control operations
  - 1
  - 0-n

- **Port statistics**
  - Traffic counters: frames, words
  - Link error counters: LESB or equivalent
  - 1
  - 1
SM-HBA structure: A SAS HBA

**HBA**
- HBA handle, model #, firmware version, ..., # of Ports

**SAS Port**
- Port index, Port address, ..., # of Phys

**SAS Phy**
- Phy index, phy identifier, domain
- Port id, speed, physical attributes

**Protocol statistics**
- SAS Protocol type, I/O counters: Input operations, output operations, control operations

**Phy statistics**
- Traffic counters: frames, words
- Link error counters
SM-HBA structure: An FC HBA

- **HBA**
  - HBA handle, model #, firmware version, ..., # of Ports

- **FC Port**
  - Port index, Port name, Port ID, Fabric name, ..., # of Phys=1

- **FC Phy**
  - Phy index, speed, topology, physical attributes

- **Protocol statistics**
  - FC-4 TYPE, I/O counters: Input operations, output operations, control operations

- **Phy statistics**
  - Traffic counters: frames, words link error counters: LESB or equivalent
API for Adapter configuration

- Pretty much identical for SAS, FC, and mixed HBAs
- Hardware identification and inventory information
- A port count
API for Port configuration

- Not quite identical for SAS and FC
- Hardware identification and inventory information
- Port name/address information
  - WWPN for both
  - N_Port_ID for FC
- Discovered Port count
- Port type (FC or SAS)
- Type-specific protocol support (e.g., FCP, SSP)
- Phy count (for FC, it’s restricted to small values of 1)
API for Phy configuration

- Mostly divergent for SAS and FC
- Hardware identification
- Type-specific speeds supported and active
- Max frame size for FC
- Domain ID for SAS
API for statistics

- Structurally the same
  - Protocol statistics per Port
  - Link statistics per Phy
- But different statistics for SAS than FC
API for discovered device management

- Manages HBAs/drivers that map SAN devices into a virtual SPI interface (bus, target, LUN)
- Very similar... For each local port:
  - List of discovered ports, with limited port info for each
    - May include switch/expander ports (e.g., FC Name Server; SAS SMP)
  - List of SCSI devices/logical units currently mapped into the OS (called “target mappings”, though they probably are LU mappings)
  - List of SCSI devices/logical units that are desired to be mapped into the OS (called “persistent bindings”). This is one of the few configurable entities
API for Fabric/Domain management

- Management command passthrough functions
  - Passthrough of application-constructed SMP IU for SAS
  - Passthrough of application-constructed CT IU for FC

- Specific functions to construct certain FC ELSs
  - Parameters for specific ELS fields, not just a raw IU

- All send one command, wait for a response, and the raw response IU is among the function return parameters

- We’re a little nervous how much damage might be done by the passthrough functions
API for SCSI discovery

- If we were nervous about management passthrough, the idea of SCSI passthrough REALLY bothered us
  - Our OS friends made sure we remained appropriately concerned
  - Eventually we set our ground rule: we would pass through no SCSI command that was not essential to discovery
  - These are not raw passthrough. Like the FC ELS functions, they have parameters for each field.
  - This led to…

- Three SCSI command APIs:
  - REPORT LUNS
  - INQUIRY
  - READ CAPACITY

- Identical for FC and SAS
API for asynchronous event notification

- Same architecture, similar event classes, some common events
- API registers callback functions for any of several event classes
  - Registrations remain until explicitly removed (or the application goes away).
  - May register multiple callbacks for a class; they all get called for each occurrence. Caveat emptor.
- Callback function is called when an event in its class occurs
  - Callback function receives event identity and a “token” from the registration call
  - The token allows common handling of several event classes
What’s in SM-HBA-2

- API for FC virtualization features
- Functions to create relationships among ports and phis
  - May apply to reconfiguring port/phy assignment in SAS
- A place to keep host bus parameters (e.g., PCI address)
- API for SCSI Management Protocol operations
- Time and interest permitting, API for management of host security policy (note, this is not aimed at zoning, that’s considered fabric policy)
  - SCSI key management would be within scope
- More flexible API for searching configuration (e.g., traverse relationships both ways, not just top-down)
Questions?
Backup material
API for Adapter configuration

6.3.1 Generic Adapter Attribute

```c
typedef struct SMHBA_AdapterAttributes {
    char     Manufacturer[64];
    char     SerialNumber[64];
    char     Model[256];
    char     ModelDescription[256];
    char     HardwareVersion[256];
    char     DriverVersion[256];
    char     OptionROMVersion[256];
    char     FirmwareVersion[256];
    HBA_UINT32 VendorSpecificID;
    char     DriverName[256];
    char     HBASymbolicName[256];
    char     RedundantOptionROMVersion[256];
    char     RedundantFirmwareVersion[256];
} SMHBA_ADAPTERATTRIBUTES, *PSMHBA_ADAPTERATTRIBUTES;
```
API for Port configuration

typedef struct SMHBA_PortAttributes {
    HBA_PORTTYPE     PortType;
    HBA_PORTSTATE    PortState;
    char             OSDeviceName[256];
    SMHBA_PORT       PortSpecificAttributes;
} SMHBA_PORTATTRIBUTES, *PSMHBA_PORTATTRIBUTES;

typedef struct SMHBA_FC_Port {
    HBA_WWN        NodeWWN;
    HBA_WWN        PortWWN;
    HBA_UINT32     FcId;
    HBA_COS        PortSupportedClassofService;
    HBA_FC4TYPES   PortSupportedFc4Types;
    HBA_FC4TYPES3  PortActiveFc4Types;
    HBA_WWN        FabricName;
    char           PortSymbolicName[256];
    HBA_UINT32     NumberOfDiscoveredPorts;
    HBA_UINT6      NumberOfPhys;
} SMHBA_FC_PORT, *PSMHBA_FC_PORT;

typedef struct SMHBA_SAS_Port {
    HBA_SASPORTPROTOCOL PortProtocol;
    HBA_WWN            LocalSASAddress;
    HBA_WWN            AttachedSASAddress;
    HBA_UINT32         NumberOfDiscoveredPorts;
    HBA_UINT32         NumberOfPhys;
} SMHBA_SAS_PORT, *PSMHBA_SAS_PORT;
API for Phy configuration

SAS Phy

typedef struct SMHBA_SAS_phy {
    HBA_UINT8 PhyIdentifier;
    HBA_SASPHYSPEED NegotiatedLinkRate;
    HBA_SASPHYSPEED ProgrammedMinLinkRate;
    HBA_SASPHYSPEED HardwareMinLinkRate;
    HBA_SASPHYSPEED ProgrammedMaxLinkRate;
    HBA_SASPHYSPEED HardwareMaxLinkRate;
    HBA_WWN domainPortWWN;
} SMHBA_SAS_PHY, *PSMHBA_SAS_PHY;

FC Phy

typedef struct SMHBA_FC_phy {
    HBA_FCPHYSPEED PhySupportedSpeed;  /* PhySupportedSpeed */
    HBA_FCPHYSPEED PhySpeed;           /* PhySpeed */
    HBA_FCPHYTYPE PhyType;
    HBA_UINT32 MaxFrameSize;            /* MaxFrameSize */
} SMHBA_FC_PHY, *PSMHBA_FC_PHY;
API for statistics

Protocol Statistics

/* Statistical counters for FC-4, SSP, STP, SMP protocols */
typedef struct SMHBA_ProtocolStatistics {
    HBA_INT64 SecondsSinceLastReset;
    HBA_INT64 InputRequests;
    HBA_INT64 OutputRequests;
    HBA_INT64 ControlRequests;
    HBA_INT64 InputMegabytes;
    HBA_INT64 OutputMegabytes;
} SMHBA_PROTOCOLSTATISTICS, *PSMHBA_PROTOCOLSTATISTICS;

SAS phy statistics

typedef struct SMHBA_SASPhyStatistics {
    HBA_INT64 SecondsSinceLastReset;
    HBA_INT64 TXFrames;
    HBA_INT64 TWords;
    HBA_INT64 RXFrames;
    HBA_INT64 RXWords;
    HBA_INT64 InvalidWordCount;
    HBA_INT64 RunningDisparityErrorCount;
    HBA_INT64 LossOfWordSyncCount;
    HBA_INT64 PhyResetProblemCount;
} SMHBA_SASPHYSTATISTICS, *PSMHBA_SASPHYSTATISTICS;

FC phy statistics

typedef struct SMHBA_FCPHYStatistics {
    HBA_INT64 SecondsSinceLastReset;
    HBA_INT64 TXFrames;
    HBA_INT64 TWords;
    HBA_INT64 RXFrames;
    HBA_INT64 RXWords;
    HBA_INT64 LIPCount;
    HBA_INT64 NOSCount;
    HBA_INT64 ErrorFrames;
    HBA_INT64 DumpedFrames;
    HBA_INT64 LinkFailureCount;
    HBA_INT64 LossOfSyncCount;
    HBA_INT64 LossOfSignalCount;
    HBA_INT64 PrimitiveSeqProtocolErrCount;
    HBA_INT64 InvalidTXWordCount;
    HBA_INT64 InvalidCRCCount;
} SMHBA_FCPHYSTATISTICS, *PSMHBA_FCPHYSTATISTICS;
API for discovered device management

```c
typedef struct SMHBA_ScsiEntry {
    SMHBA_SCSIID ScsiId;
    SMHBA_PORTLUN PortLun;
    SMHBA_LUID LUID;
} SMHBA_SCSIENTRY, *PSMHBA_SCSIENTRY;

typedef struct SMHBA_TargetMapping {
    HBA_UINT32 NumberOfEntries;
    SMHBA_SCSIENTRY entry[1];    /* Variable length array containing mappings */
} SMHBA_TARGETMAPPING, *PSMHBA_TARGETMAPPING;

typedef struct SMHBA_BindingEntry {
    SMHBA_BIND_TYPE type;
    SMHBA_SCSIID ScsiId;
    SMHBA_PORTLUN PortLun;
    SMHBA_LUID LUID;
    HBA_STATUS Status;
} SMHBA_BINDINGENTRY, *PSMHBA_BINDINGENTRY;

typedef struct SMHBA_Binding {
    HBA_UINT32 NumberOfEntries;
    SMHBA_BINDINGENTRY entry[1];    /* Variable length array */
} SMHBA_BINDING, *PSMHBA_BINDING;
```
API for Fabric/Domain management

<table>
<thead>
<tr>
<th>Fabric and Domain Management Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HBA_SendCTPassThruV2</td>
<td>8.4.1</td>
</tr>
<tr>
<td>HBA_SetRNIDMgmtInfo</td>
<td>8.4.2</td>
</tr>
<tr>
<td>HBA_GetRNIDMgmtInfo</td>
<td>8.4.3</td>
</tr>
<tr>
<td>HBA_SendRNIDV2</td>
<td>8.4.4</td>
</tr>
<tr>
<td>HBA_SendRPL</td>
<td>8.4.5</td>
</tr>
<tr>
<td>HBA_SendRPS</td>
<td>8.4.6</td>
</tr>
<tr>
<td>HBA_SendSRL</td>
<td>8.4.7</td>
</tr>
<tr>
<td>HBA_SendLIRR</td>
<td>8.4.8</td>
</tr>
<tr>
<td>HBA_SendRLS</td>
<td>8.4.9</td>
</tr>
<tr>
<td>SMHBA_SendTEST</td>
<td>8.4.10</td>
</tr>
<tr>
<td>SMHBA_SendECHO</td>
<td>8.4.11</td>
</tr>
<tr>
<td>SMHBA_SendSMPPassThru</td>
<td>8.4.12</td>
</tr>
</tbody>
</table>
API for Fabric/Domain management

Example raw passthrough function

```
HBA_UINT32  SMHBA_SendSMPPassThru(
    HBA_HANDLE    handle,
    HBA_WWN       hbaPortWWN,
    HBA_WWN       destPortWWN,
    HBA_WWN       domainPortWWN,
    void          *pReqBuffer,
    HBA_UINT32    ReqBufferSize,
    void          *pRspBuffer,
    HBA_UINT32    *pRspBufferSize
);
```

Example structured passthrough function

```
HBA_STATUS HBA_SendRPL ( 
    HBA_HANDLE handle, 
    HBA_WWN hbaPortWWN, 
    HBA_WWN agent_wwn, 
    HBA_UINT32 agent_domain, 
    HBA_UINT32 portIndex, 
    void *pRspBuffer,
    HBA_UINT32 *pRspBufferSize
);
```
API for SCSI discovery

Example SCSI passthru function

```c
HBA_STATUS SMHBA_ScsiInquiry ( 
    HBA_HANDLE handle, 
    HBA_WWN hbaPortWWN, 
    HBA_WWN discoveredPortWWN, 
    HBA_WWN domainPortWWN; 
    SMHBA_SCSILUN smhbaLUN, 
    HBA_UINT8 CDB_Byte1, 
    HBA_UINT8 CDB_Byte2, 
    void *pRspBuffer, 
    HBA_UINT32 *pRspBufferSize, 
    HBA_UINT8 *pScsiStatus, 
    void *pSenseBuffer, 
    HBA_UINT32 *pSenseBufferSize
);`
```
API for asynchronous event notification

<table>
<thead>
<tr>
<th>Event Handling Functions</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMHBA_RegisterForAdapterAddEvents</td>
<td>8.7.2</td>
</tr>
<tr>
<td>SMHBA_RegisterForAdapterEvents</td>
<td>8.7.3</td>
</tr>
<tr>
<td>SMHBA_RegisterForAdapterPortEvents</td>
<td>8.7.4</td>
</tr>
<tr>
<td>SMHBA_RegisterForAdapterPortStatEvents</td>
<td>8.7.5</td>
</tr>
<tr>
<td>SMHBA_RegisterForAdapter PhyStatEvents</td>
<td>8.7.6</td>
</tr>
<tr>
<td>SMHBA_RegisterForTargetEvents</td>
<td>8.7.7</td>
</tr>
<tr>
<td>HBA_RegisterForLinkEvents</td>
<td>8.7.8</td>
</tr>
<tr>
<td>HBA_RemoveCallback</td>
<td>8.7.9</td>
</tr>
</tbody>
</table>

8.7.4 SMHBA_RegisterForAdapterPortEvents

8.7.4.1 Format

```c
HBA_STATUS SMHBA_RegisterForAdapterPortEvents(
    void (*pCallback)(
        void *pData,
        HBA_WWN portWWN,
        HBA_UINT32 eventType,
        HBA_UINT32 fabricPortID
    ),
    void *pUserData,
    HBA_HANDLE handle,
    HBA_WWN portWWN,
    HBA_UINT32 specificEventType,
    HBA_CALLBACKHANDLE *pCallbackHandle
);
```

6.8.1.4 Port Category Event Types

- `#define HBA_EVENT_PORT_UNKNOWN` 0x200
- `#define HBA_EVENT_PORT_OFFLINE` 0x201
- `#define HBA_EVENT_PORT_ONLINE` 0x202
- `#define HBA_EVENT_PORT_NEW_TARGETS` 0x203
- `#define HBA_EVENT_PORT_FABRIC` 0x204
- `#define HBA_EVENT_PORT_BROADCAST_CHANGE` 0x205
- `#define HBA_EVENT_PORT_BROADCAST_SES` 0x208
- `#define HBA_EVENT_PORT_BROADCAST_D24_0` 0x206
- `#define HBA_EVENT_PORT_BROADCAST_D27_4` 0x207
- `#define HBA_EVENT_PORT_BROADCAST_D01_4` 0x209
- `#define HBA_EVENT_PORT_BROADCAST_D04_7` 0x20A
- `#define HBA_EVENT_PORT_BROADCAST_D16_7` 0x20B
- `#define HBA_EVENT_PORT_BROADCAST_D29_7` 0x20C
- `#define HBA_EVENT_PORT_ALL` 0x2FF
Here’s the trick for virtualizing FC Ports...

If the number of HBAs is zero, the number of FC Phys shall also be zero.
...a tweak more for the PCI IOV guys...

- **HBA**: HBA handle, manufacturer, model #, serial #, firmware version, ... # of Ports
- **Bus Address**: variants for different standard buses
- **FC Phy**: Phy index, speed, topology, physical attributes
- **HBA**: HBA handle, manufacturer, model #, serial #, firmware version, ... # of Ports
- **FC Port**: Port index, Port name, Port address, Fabric name, ... # of Phys = 1
- **Physical HBA**: PHBA index, manufacturer, model #, serial #, # Ports supported, # Phys, # HBAs
- **Protocol statistics**: FC-4 TYPE, I/O counters: Input operations, output operations, control operations
- **Port statistics**: traffic counters: frames, words
- **Phy statistics**: traffic counters: frames, words, link error counters
...and finally, VSANs

- **Bus Address**: variants for different standard buses
  - 0-1

- **HBA**: HBA handle, manufacturer, model #, serial #, firmware version, ...
  - 0-1

- **Physical HBA**: PHBA index, manufacturer, model #, serial #, # Ports supported, # Phys, # HBAs
  - 0-1

- **Fabric**: VF_ID, Fabric name, in-order delivery, ...
  - 0-n

- **FC Port**: Port index, Port name, Port address, Fabric name, ...
  - 0-n

  - # of Phys=1

- **Protocol statistics**: FC-4 TYPE, I/O counters: Input operations, output operations, control operations

- **Phy statistics**: traffic counters: frames, words

- **FC Phy**: Phy index, speed, topology, physical attributes
  - 0-1

  - 1-n

  - Link error counters

- **Port statistics**: traffic counters: frames, words

- **Number of Ports**: 0-n