From: Bob Sheffield – LSI Corporation
To: T10 Committee
Subject: T10/08-044r1 SBC-3 DIF Granularity
Date: 5 May 2008

Related Documents: SBC-3r12

Revision History:

Revision	Date	Description
r0	1/3/08	Initial version
r1	5/5/08	Clarify justification (4K initiator,…) and add per-command controls

Overview

There has been recent momentum in the storage industry behind the idea of supporting larger logical blocks. This offers the opportunity for improved disk reliability and/or other benefits for systems that use larger logical blocks. Interoperability with existing applications that largely depend on logical blocks having a size of 512 bytes prompts the need for storage virtualization solutions that emulate 512-byte logical blocks for the virtual logical units they expose to applications, but use disk drives formatted with larger logical blocks (e.g., 4096 bytes) as the raw storage for the virtual logical units.

With the move to larger disk blocksizes, applications that access data using matching larger blocksizes (rather than depending upon emulation) may access data more efficiently by avoiding latencies caused by read-modify-write operations in the emulation layer. So, it stands to reason that both traditional applications that access media using 512-byte blocks and new applications that access media using larger blocks (e.g., 4K-byte) will coexist, perhaps in the same platforms (e.g., virtualized servers).

Likewise, there will be a gradual transition from systems that use 512-byte formatted disks to larger blocksize media, and the larger blocksize media may include both media formatted to a larger logical blocksize as well as media formatted to a larger physical blocksize but that emulates 512-byte logical blocks.

This leads to a relatively complex intermix of application expected blocksizes and disk supported blocksizes that, if not appropriately addressed in SCSI standards, leads to a variety of non-interoperability problems, particularly when protection information (PI) is used. The table below lists the relevant combinations, and whether interoperability holds for PI-enabled implementations.

Case	Application client block length	Disk block length		Comment
		Logical	Physical	
1	T[a]	T	T	Traditional usage
2	T	T	L[b]	Emulation of traditional block length
3	T	L	T	Not interoperable (interoperability established by this proposal)
4	T	L	L	
5	L	T	T	
6	L	T	L	
7	L	L	T	Interoperable, but not defined in standards
8	L	L	L	Complete agreement on larger block lengths

[a] **T**: Traditional block length (i.e., 512-bytes)
[b] **L**: Large block length (e.g., 4Kbyte – but could be something else like 2K or 8K)

Today, when protection information is present, interoperability is achieved as long as disks and host applications both support accessing data using 512-byte blocks. But, if either the disk or the application client (but not both) is configured to access data using a larger logical block length, the result is non-interoperability.

The existing definitions for the logical blocks per physical block exponent and lowest aligned logical block address fields in READ CAPACITY (16) parameter data provide interoperability for cases 1 and 2 in the table above, but do not address interoperability for the remaining 6 cases.

Furthermore, in virtualized server and external storage environments, block storage devices are likely to find themselves needing to support an intermix of applications that require both traditional 512-byte access and larger blocksize access, simultaneously.

A solution to this problem should accommodate future increases in blocksize, and should provide the framework to establish interoperability with any application, regardless of the blocksize expected by the application (restricting actual application data blocksize to be 2^nx512 bytes).

Systems such as these should be able to provide protection information for the virtual logical units exposed to applications. Cases 3, 4, 5, and 6 in the table above prompt the need for physical sector formats on the disk drives to accommodate 2^n x (512+8) bytes of information, where 2^n is the disk drive physical block size / virtual device logical block size; in order to provide space for protection information on disk. Furthermore, the disk should be able to check protection information fields even when the granularity of data protected by protection information fields is smaller than the disk blocksize.

This proposal defines a method to specify, on a per-command basis, the *virtual* logical blocksize the device server shall use to transfer data to or from the application client. The CDBs for read and write type commands contain three reserved bits in the same byte as the GROUP NUMBER field. This proposal suggests to use these three bits to encode an VIRTUAL BLOCKSIZE EXPONENT field. An VIRTUAL BLOCKSIZE EXPONENT field set to zero causes operation consistent with existing implementations where the dynamic virtual logical blocksize is the same as the value of the LOGICAL BLOCK LENGTH IN BYTES field reported in READ CAPACITY data. If the VIRTUAL BLOCKSIZE EXPONENT field is non-zero, then the virtual logical blocksize is the value of the reported LOGICAL BLOCK LENGTH IN BYTES x 2^n, where n is the value in the VIRTUAL BLOCKSIZE EXPONENT field.

So, for example, if the LOGICAL BLOCK LENGTH IN BYTES reported in READ CAPACITY data is 512 bytes, and a READ (16) command is received with the VIRTUAL BLOCKSIZE EXPONENT field set to 3, the virtual logical blocksize would be 8 x 512 = 4096. For the processing of that command, the LOGICAL BLOCK ADDRESS field specifies the offset of the requested data in 4096-byte increments from the beginning of media, and the TRANSFER LENGTH field specifies the amount of data to be transferred in 4096-byte increments.

This allows a disk that has a physical blocksize of 4096 bytes and a logical blocksize of 512 bytes to interoperate both with application clients that use a logical blocksize of 512 bytes and with application clients that use a blocksize of 4096 bytes. The disk could support other virtual logical blocksizes as well, depending on the need. What is important is to provide the semantic framework in SBC-3 to implement interoperability where it's needed.

Alternatively, if the logical block length reported in READ CAPACITY data is 1024 bytes, and a READ (16) command is received with the VIRTUAL BLOCKSIZE EXPONENT field set to 1, the virtual logical blocksize would be 2 x 1024 = 2048. For the processing of that command, the LOGICAL BLOCK ADDRESS field specifies the offset of the requested data in 2048-byte increments from the beginning of media, and the TRANSFER LENGTH field specifies the amount of data to be transferred in 2048-byte increments.

This proposal defines the application of protection information to coincide with the virtual logical block boundaries as specified in the CDB.

- For type-1 data protection, the logical block reference tag contains the lower four bytes of the LBAs specified to be transferred in the CDB.
- For type-2 data protection, the CDB specifies the value of the first logical block reference tag transferred and the value increments by one for each subsequent virtual logical block transferred.

This addresses both the need to provide raw block capacity for virtual devices emulating legacy block sizes, and also avoids diluting the effectiveness of the CRC field as logical block sizes increase.

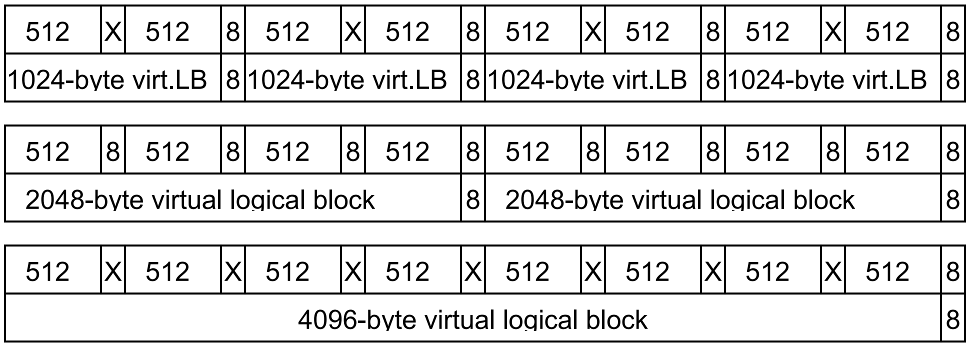
512 X 512 8 512 X 512 8 512 X 512 8 512 X 512 8
1024-byte virt.LB 8 1024-byte virt.LB 8 1024-byte virt.LB 8 1024-byte virt.LB 8
512 8 512 8 512 8 512 8 512 8 512 8 512 8 512 8
2048-byte virtual logical block 8 2048-byte virtual logical block 8
512 X 512 X 512 X 512 X 512 X 512 X 512 X 512 8
4096-byte virtual logical block 8

Figure 1 – Examples mapping logical blocks to virtual logical blocks

Suggested Changes to SBC-3

3.1.40 protection information: Fields appended to each virtual logical block that contain a cyclic redundancy check (CRC), an application tag, and a reference tag. See 4.17.

3.1.xx virtual logical block: A unit of user data (see 3.1.50) composed of 2^n adjacent logical blocks (see 3.1.26) that may be protected by an instance of protection information (see 3.1.40).

3.1.xx virtual logical block address (VLBA): The value used to reference a virtual logical block (see 3.1.xx).

Add the following to the list of symbols and abbreviations:

VLBA	virtual logical block address (see 3.1.xx)

Modify subclause 4.14.1 Error reporting overview to specify reporting the VLBA instead of the LBA in the INFORMATION *field when applicable.*

When a command attempts to access or reference an invalid LBA, the first invalid LBA shall be returned in the INFORMATION field of the sense data (see SPC-4).

When a command attempts to access or reference an invalid VLBA, the first invalid VLBA shall be returned in the INFORMATION field of the sense data.

When a recovered read error is reported and the VLBAE field is set to zero, the INFORMATION field of the sense data shall contain the LBA of the last logical block accessed by the command on which a recovered read error occurred.

When a recovered read error is reported and the VLBAE field is non-zero, the INFORMATION field of the sense data shall contain the VLBA of the last virtual logical block accessed by the command on which a recovered read error occurred.

When an unrecovered error is reported and the VLBAE field is set to zero, the INFORMATION field of the sense data shall contain the LBA of the logical block on which the unrecovered error occurred.

When an unrecovered error is reported and the VLBAE field is non-zero, the INFORMATION field of the sense data shall contain the VLBA of the virtual logical block on which the unrecovered error occurred.

Add subclause 4.6 to describe virtual logical blocks

4.6 Virtual logical blocks

An application client may reference logical blocks using a virtual logical block address (VLBA). A VLBA references a virtual logical block (see 3.1.xx) that is composed of 2^n adjacent logical blocks, where the value of n is specified in the VLBAE field of the command specifying a range of virtual logical blocks to transfer. The LBA of the first logical block in the virtual logical block referenced by a VLBA value of zero shall be the LBA of the lowest aligned logical block as indicated in the LOWEST ALIGNED LOGICAL BLOCK ADDRESS field returned in the READ CAPACITY (16) parameter data. The LBA of the first logical block in the next virtual logical block shall be the LBA of the first logical block in the previous virtual logical block plus 2^n. The pattern continues until the number of remaining LBAs, as determined by the returned logical block address field in READ CAPACITY (16) parameter data, is less than 2^n.

A non-zero value in the VLBAE field specifies that the application client is specifying a virtual logical block address in the LOGICAL BLOCK ADDRESS FIELD of the CDB. If a non-zero value is specified in the VLBAE field, the field in the CDB specifying the number of logical blocks to process (e.g., the TRANSFER LENGTH field or the NUMBER OF LOGICAL BLOCKS field) shall specify the number of virtual logical blocks processed by the command.

If the device server does not support virtual block addressing and the VLBAE field contains a value other than zero, the command shall be terminated with CHECK CONDITION status with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

If the VLBAE field contains a value other than zero that is not supported by the device server, the command shall be terminated with CHECK CONDITION status with the sense key set to ILLEGAL REQUEST and the additional sense code set to INVALID FIELD IN CDB.

4.17.3 Protection information format

Table 7 defines the ~~placement~~ format of protection information in a logical block or virtual logical block.

Table 7 – user data and protection information format

Byte/Bit	7	6	5	4	3	2	1	0
0	USER DATA							
n - 1								
n	LOGICAL BLOCK GUARD							
n + 1								
n + 2	LOGICAL BLOCK APPLICATION TAG							
n + 3								
n + 4	LOGICAL BLOCK REFERENCE TAG							
n + 7								

The USER DATA field shall contain user data. The contents of the USER DATA field shall be used to generate and check the CRC contained in the LOGICAL BLOCK GUARD field.

The LOGICAL BLOCK GUARD field contains the CRC (see 4.17.4) of the contents of the USER DATA field.

The LOGICAL BLOCK APPLICATION TAG field is set by the application client. If the device server detects a:

a) LOGICAL BLOCK APPLICATION TAG field set to FFFFh and type 1 protection (see 4.17.2.3) or type 2 protection (see 4.17.2.4) is enabled; or
b) LOGICAL BLOCK APPLICATION TAG field set to FFFFh, LOGICAL BLOCK REFERENCE TAG field set to FFFF FFFFh, and type 3 protection (see 4.17.2.5) is enabled,

then the device server disables checking of all protection information for the logical block when reading from the medium. Otherwise, the contents of the logical block application tag are not defined by this standard.

The LOGICAL BLOCK APPLICATION TAG field may be modified by a device server if the ATO bit is set to zero in the Control mode page (see SPC-4).

The contents of the LOGICAL BLOCK APPLICATION TAG field shall not be used to generate or check the CRC contained in the LOGICAL BLOCK GUARD field.

The LOGICAL BLOCK REFERENCE TAG field of the first logical block or virtual logical block in the data-in buffer and/or data-out buffer shall contain the value specified in table 8.

Table 8 – Content of the LOGICAL BLOCK REFERENCE TAG field of the first logical block or virtual logical block in the data-in buffer and/or data-out buffer

Protection Type	Content of the LOGICAL BLOCK REFERENCE TAG field of the first logical block or virtual logical block in the data-in buffer and/or data-out buffer
Type-1 protection (see 4.17.2.3)	The least significant four bytes of the LBA contained in the LOGICAL BLOCK ADDRESS field of the command.
Type-2 protection (see 4.17.2.4)	The value in the EXPECTED INITIAL LOGICAL BLOCK REFERENCE TAG field of the command.
Type-3 protection (see 4.17.2.5)	Not defined in this standard.

The LOGICAL BLOCK REFERENCE TAG field subsequent logical blocks in the data-in buffer and/or data-out buffer shall be set as specified in table 9.

Table 9 – Setting of the LOGICAL BLOCK REFERENCE TAG field of the subsequent logical block in the data-in buffer and/or data-out buffer

Protection Type	The content of the LOGICAl block reference tag field of the sebsequent logical block in the data-in buffer and/or data-out buffer
Type-1 protection (see 4.17.2.3) and Type-2 protection (see 4.17.2.4)	The logical block reference tag of the previous logical block plus one.
Type-3 protection (see 4.17.2.5)	Not defined in this standard.

The contents of the LOGICAL BLOCK REFERENCE TAG field shall not be used to generate or check the CRC contained in the LOGICAL BLOCK GUARD field.

Add subclause 4.17.4 to describe application of protection information with virtual logical blocks

4.17.4 Protection information with virtual logical blocks

For commands specifying a non-zero value in the VLBA field, the eight bytes of protection information are appended to each virtual logical block transferred. Media shall be formatted to accommodate 2^n x 8 bytes of protection information (i.e., one instance of protection information for each logical block contained within the virtual logical block), however only one of the protection information fields contained within a virtual logical block shall be used, and it shall contain protection information for the virtual logical block.

The logical block guard field shall contain the CRC calculated over all user data contained within the virtual logical block.

For type-1 protection information, the LOGICAL BLOCK REFERENCE TAG field shall contain the lower four bytes of the VLBA of the virtual logical block.

For type-2 protection information, the LOGICAL BLOCK REFERENCE TAG field shall contain the values as specified in the EXPECTED LOGICAL BLOCK REFERENCE TAG field specified in the CDB, and shall increment by one for each successive virtual logical block transferred.

Modify the descriptions of the applicable block commands (those that contain a GROUP NUMBER field) as follows

Table x – 10-byte CDB format

Byte/Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE							
1	Command dependent							Obsolete
2	(MSB)	LOGICAL BLOCK ADDRESS						
5								(LSB)
6	VLBAE ~~Reserved~~			GROUP NUMBER				
7	(MSB)	TRANSFER LENGTH (or other length field)						
8								(LSB)
9	CONTROL							

A non-zero value in the VIRTUAL LBA EXPONENT (VLBAE) field specifies that the LOGICAL BLOCK ADDRESS field contains the VLBA of the first virtual logical block to be transferred, and that the TRANSFER LENGTH field specifies the number of virtual logical blocks to transfer, where each virtual logical block contains 2^n logical blocks, and n is the value specified in the VLBAE field.

Table x – 12-byte CDB format

Byte/Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE							
1	Command dependent							Obsolete
2	(MSB)	LOGICAL BLOCK ADDRESS						
5								(LSB)
6	(MSB)	TRANSFER LENGTH (or other length field)						
9								(LSB)
10	VLBAE ~~Reserved~~			GROUP NUMBER				
11	CONTROL							

Table x – 16-byte CDB format

Byte/Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE							
1	Command dependent							Obsolete
2	(MSB)	LOGICAL BLOCK ADDRESS						
9								(LSB)
10	(MSB)	TRANSFER LENGTH (or other length field)						
13								(LSB)
14	VLBAE ~~Reserved~~			GROUP NUMBER				
15	CONTROL							

Editor's note: Some of the commands show the high-order bit as "Restricted for MMC-4". This is not required as MMC-4 is not a block-storage device, and so the high-order bit should be "Reserved", and is being used as the high-order bit of the VLBAE field.

Table x – 32-byte CDB format (example)

Byte/Bit	7	6	5	4	3	2	1	0
0	OPERATION CODE							
1	CONTROL							
2	Reserved							
5								
6	VLBAE ~~Reserved~~			GROUP NUMBER				
7	ADDITIONAL CDB LENGTH (18h)							
8	SERVICE ACTION							
9								
10	XXPROTECT			DPO	FUA	Reserved	FUA_NV	Reserved
11	Reserved							
12	(MSB)	LOGICAL BLOCK ADDRESS						
19								(LSB)
20	(MSB)	EXPECTED INITIAL LOGICAL BLOCK REFERENCE TAG						
23								(LSB)
24	(MSB)	EXPECTED LOGICAL BLOCK APPLICATION TAG						
25								(LSB)
26	(MSB)	LOGICAL BLOCK APPLICATION TAG MASK						
27								(LSB)
28	(MSB)	TRANSFER LENGTH (or other length field)						
31								(LSB)