

An Implementation of a GSN to Fibre Channel Bridge Using Scheduled Transfer Encapsulation of the SCSI Protocol

Abstract

Modern disk storage subsystems are expanding in capacity and increasing in speed at a rate that is slower than the increases in computational speed. This results in overall system performance degradation as the CPU becomes starved for data from slower I/O subsystems.

This document outlines a means of substantially increasing the speed and capacity of storage subsystems by outlining a hardware implementation that features an 800 megabyte per second Gigabyte System Network (GSN, a.k.a. HIPPI-6400) bridge to eight 100 megabyte per second Fibre Channel RAID storage subsystems. SCSI commands, data, and responses are mapped between the GSN port and the Fibre Channel ports by encapsulating SCSI commands into the Scheduled Transfer protocol over GSN. A thin SCSI to ST translation layer implemented as a device driver on a host computer system can easily access petabytes of Fibre Channel storage at data rates not achievable by other technologies.

**Dr. Stephen W. Bailey
Chief Scientist
GENROCO, Inc.**

**Donald D. Woelz
V. P. of Marketing
GENROCO, Inc.**

Introduction

Fibre Channel, at 100 megabytes per second, is fast becoming the high performance interconnect of choice for storage subsystems. The advent of Gigabyte Systems Network (GSN, a.k.a. HIPPI-6400) networking at 800 megabytes per second makes it an ideal choice as a network backbone. Concentrating multiple Fibre Channel data streams onto a GSN network provides a means of creating very large storage subsystems with an extremely fast interconnect.

This document describes such a system and presents some of the hardware and software issues addressed to implement the design. The issues discussed include:

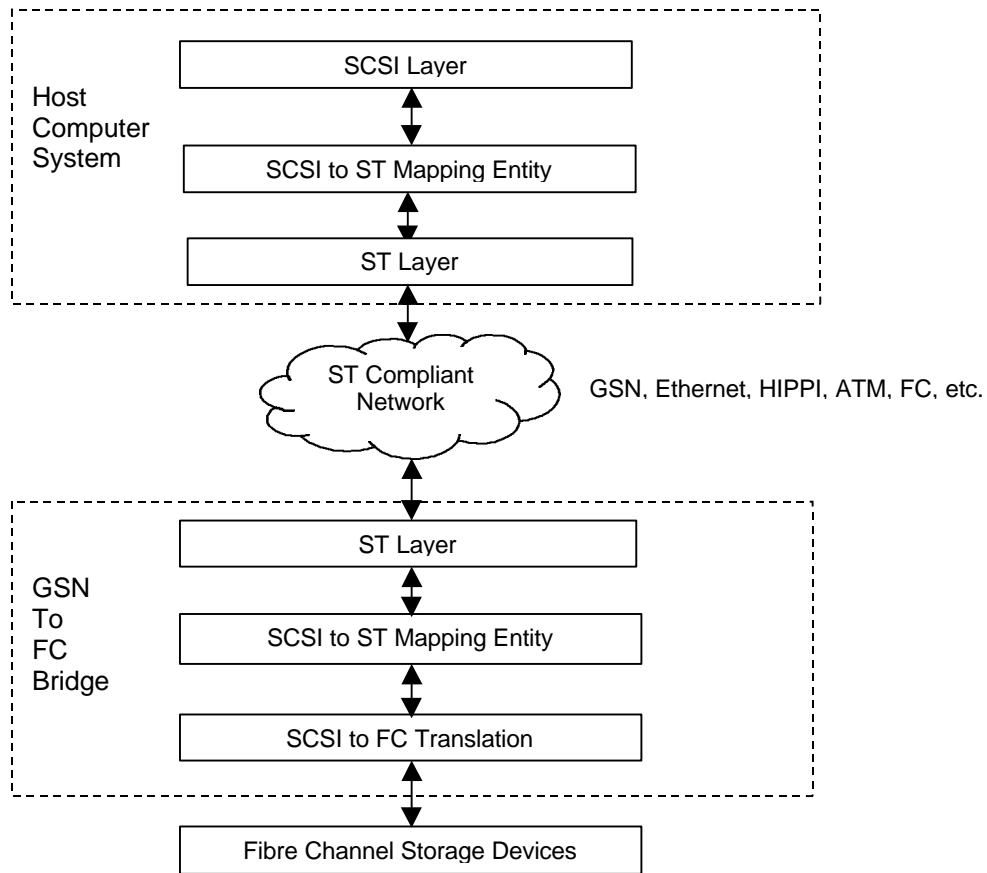


Figure 1 – Data flow block diagram

- a means of encapsulating standard SCSI commands and responses into an efficient network protocol capable of being carried by many disparate network types
- a means of extracting the SCSI commands in the bridge and translating them to the FC subsystem

Note: This document assumes that the reader has a working knowledge of the Scheduled Transfer protocol, HIPPI-6400, the SCSI protocol, and the FCP Fiber Channel SCSI protocol. Terminology and definitions from those disciplines are used throughout this paper.

A block diagram of the data flow is shown in Figure 1. A mapping entity in the host operating system encapsulates SCSI storage commands generated by the host application into the Scheduled Transfer (ST) protocol. The resulting ST messages are transmitted through the network fabric to the GSN to FC Bridge. The bridge extracts the SCSI storage commands and data from the ST messages, translates them to the corresponding FC sequences, and retransmits them through the FC ports to the FC storage devices. SCSI responses and data from the FC storage devices follow a similar reverse route back to the host operating system.

Scheduled Transfer encapsulation of SCSI

There are several features of the Scheduled Transfer protocol that make it an effective transport for accessing disk storage on a network. These include:

- Flow-controlled Block Read and Write sequences
- Port selection, sequence identification, and operation validation
- Efficient mapping between the issuer's and receiver's natural buffer sizes;
- Block striping support
- Mappings onto HIPPI-6400-PH, HIPPI-FP (for HIPPI-800 traffic), Ethernet, ATM, and Fibre Channel lower-layer protocols.

Most modern commercial operating systems include support for SCSI storage, and a significant infrastructure of storage management and control functionality is layered above that SCSI architecture. The ability to leverage that infrastructure, and the fact that FC also incorporates the SCSI protocol (FCP), are strong influencing factors in selecting SCSI as the storage protocol.

A mapping or encapsulation of SCSI into the ST protocol provides a very efficient, high bandwidth methodology for implementing networked storage. The following paragraphs outline the issues associated with mapping SCSI onto ST.

In this discussion, the ST Initiator corresponds to the SCSI-to-ST mapping entity (device driver) in the host system and the ST Responder corresponds to the ST-to-FC mapping entity in bridge. Initiator and Responder correspond loosely to SCSI Initiator and SCSI Target respectively in this implementation as the ST-to-FC mapping entity in the bridge acts as a proxy to the FCP (SCSI) targets on the FC ports. In order delivery through the ST fabric is assumed.

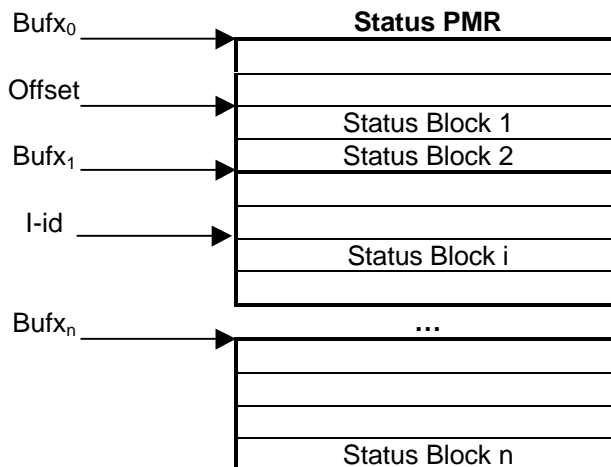
Connection Setup

The setup of an ST Virtual Connection (VC) is done between the ST layer of the Initiator system and the ST Responder process in the GSN-to-FC Bridge when a SCSI operation is required to an FC port on the bridge. The optional payload of the Request_Connection operation contains the size and number of status blocks used to receive the SCSI operation completion status.

In a typical implementation, the Initiator will send a port number that is used to address its global table of active Virtual Connections. In other words, the port number will be used for selection (rather than validation) of the Virtual Connection context. The Responder will do the same with the port number that it returns.

After the Virtual Connection is established, the Responder must request that the Initiator expose a status Persistent Memory Region (PMR) using a Request_Memory_Region message with the Size equal to the status block size times the number of status blocks. The initial Offset of the status PMR returned by the Initiator must be a multiple of the individual status block size. The I-id of a SCSI encapsulated data transfer operation on ST is used as an index to select the status block in the status PMR for placing completion status.

It may be reasonable for an Initiator to use the same status PMR across all storage Virtual Connections since the operation limit of storage operations on an interface (to all targets) is almost certainly less than sum of the operation limits (queue depths) of all active targets.



Status PMR Layout

Write Operations

When a SCSI Write operation is requested, the ST Initiator sends a Request_To_Send to the Responder. The 32 byte optional payload (see Annex B of the ST standard) of the Request_To_Send operation would contain the following:

- Byte 0: ST optional payload opcode for FCP COMMAND (0x??)
- Byte 1: length = 32
- Bytes 2-3: target number
- Bytes 4-31: FCP COMMAND payload for the Write operation, bytes 0-27 (everything except data length, which is in the T_len field of the ST header)

The target number parameter placed in bytes 2 and 3 of the 32 byte optional payload is used to address multiple storage targets which can be reached through the same ST Virtual Connection. The target number is independent of the FCP Logical Unit Number (LUN) embedded in the FCP COMMAND payload. For example, if the target endpoint of a SCSI storage operation on a Virtual Connection is a bridge to some other storage interface (e.g. parallel SCSI or Fibre Channel), the target number is intended to address targets on the opposite side of the bridge.

The Responder then sends one or more Clear_To_Send messages to control the flow of data from the Initiator to its buffers. The Initiator sends the data with Flags T = 1 and I = 0 (data delivered silently) except for the last STU. Flags T = 0 and I = 1 are sent with the last STU to inform the Responder that all of the data has been delivered.

After all of the data has been received by the Responder, it initiates an ST Put sequence with Flags T = 0 and I = 1 (interrupt) to send an FCP RESPONSE payload to the Initiator. The Flags will cause the Initiator ULP (SCSI driver) to be notified that the operation has completed. The Put sequence uses the Mx from the PMR setup operation, and the Bufx and Offset computed from the initial Bufx and Offset from the PMR setup operation and the I-id, to place the FCP RESPONSE payload into the correct status block. The FCP RESPONSE payload has a variable length that includes:

- SCSI status
- data residual (with flags signaling residual direction)
- SCSI sense length
- FCP response information length
- FCP response information (optional)
- SCSI sense data (optional)

There is a potential for an operation underrun residual (i.e., the SCSI command specifies or implies a smaller data transfer than indicated by the Length parameter in the Request_To_Send operation). In response to this situation, the Initiator must be able to end the transfer before all of the data specified in the Request_To_Send operation has been transferred. Under some circumstances no data may be transferred at all. In normal instances this should occur rarely so it is not necessary that this tear down operation be a high performance path. Furthermore, this will only happen after a complete ST Block has been transferred. The target will not request more data than the SCSI command specifies.

Read Operations

When a SCSI Read operation is requested, the mapping to the ST protocol follows a pattern similar to that for a SCSI Write operation. The ST Initiator sends a Request_To_Receive to the Responder. The 32 byte optional payload (same as in Write operations) contains:

- Byte 0: ST optional payload opcode for FCP COMMAND (0x??)
- Byte 1: length = 32
- Bytes 2-3: target number
- Bytes 4-31: FCP COMMAND payload for the Read operation, bytes 0-27 (everything except data length, which is in the T_len field of the ST header)

The Responder sends a Request_To_Send to the Initiator echoing the Request_To_Receive authentication parameters. The Initiator sends a single Clear_To_Send for the entire data movement operation (see discussion below). The Responder sends the data with Flags T = 1 and I = 0 (silent delivery) except for the last STU. Setting Flags T = 0 and I = 1 on the last STU permits the ULP (SCSI layer) to know that all of the data was delivered.

After all data is sent to the Initiator, the Responder issues an ST Put sequence with Flags T = 0 and I = 1 (interrupt) to the status block in the PMR as described for Write operations. This Put sequence transfers an FCP RESPONSE payload.

A single Clear_To_Send is indicated for SCSI Read operations to permit a bridge implementation with modest buffering requirements when using existing storage devices (parallel SCSI and Fibre Channel). If multiple Clear_To_Send messages were permitted for a single SCSI Read operation, the Responder (Bridge) device would have to either buffer data for the entire operation or have logic to split the SCSI command and coalesce status returns from each of the split commands.

As with SCSI Write operations, the potential for operation underrun residual implies that the Initiator must be able to end the transfer before all the data specified in the Request_To_Send operation has been transferred. In normal operation this should occur rarely, so it is not necessary that this tear down process be a high performance path.

Abort Operations

As with any ST data movement sequence, an ST encapsulated SCSI operation may be aborted by either the Initiator or the Responder by performing an ST End sequence.

Timeout Handling

Timeout handling as described in the ST specification applies directly to encapsulated SCSI ST connections.

In addition, the Initiator must maintain timeouts on each outstanding data operation. In the event of a timeout, either because the requested data did not arrive or because the status Put operation was not performed successfully, the Initiator must determine additional information about the cause of the timeout before taking further steps. It should be determined if the Responder can still be reached. If not, the Virtual Connection must be torn down along with all outstanding operations on that VC. If the Virtual Connection is still viable, the Initiator should try to abort the operation with an ST End sequence up to two times. If the End succeeds, the operation should be retried a reasonable number of times.

GSN to FC Bridge Hardware Overview

A hardware implementation of a GSN to FC bridge is shown in block diagram form in Figure 2.

The GSN port and each FC port of the bridge employ a 160 megahertz Java engine to process and control data flow between and through each port. The processors associated with the FC ports, in addition to controlling the data channel and its communications with the GSN port, encapsulate and de-encapsulate SCSI commands and responses on the Schedule Transfer (ST) protocol and translate them to the appropriate SCSI commands and responses on the FC ports. Buffer memory at each FC port on the bridge is used as data buffers for the ST data transfers.

Bridging ST Storage to FCP

The process of communication between the ST mapping entities at both ends of the storage transfer over ST was discussed above. It is also informative to outline how the translation, or bridging, between the ST encapsulated SCSI and the FCP protocol takes place in the bridge.

Due to the somewhat limited addressing offered by GSN, SCSI connections are between a SCSI Initiator in the SCSI to ST mapping entity and an FCP proxy in the GSN to FC bridge, rather than between the SCSI Initiator and the SCSI target connected on the FC port. Targets behind the proxy are addressed with the 16-bit target number field in the optional payload of the ST Request_To_Send and Request_To_Receive operations.

Connection Setup

When an ST Virtual Connection is required for an FCP Target connected to an FC port on the bridge, the bridge will validate the VC request. This includes verifying:

- Ethertype
- Compatible endian (in Flags)
- PMR support (in Flags)
- Well known port
- Free VC state table entry in the Bridge
- Max_STU is greater than or equal to 2K
- Status block size (Power of 2)
- Number of status blocks.

The Bridge then allocates a VC state table entry with the Initiator I-Port, I-Key, R-Key, (equal to R-Key plus 1 with state maintained in the VC state table), Source ULA, and status block size.

The Bridge will then return the ST Connection_Answer response with:

- Success status
- Dynamic port number
- R-Key
- R-Bufsize (same as R-Max_STU = 2K)
- R-Slots

Finally, the Bridge sends a Request_Memory_Region message with a size equal to the PMR status block size times the number of status blocks. The initiator will expose the PMR and return a Memory_Region_Available response. The bridge will record the parameters of the status PMR, including Mx, Initial Bufx, and Initial offset.

Read Operations

The central data structure used by the bridge to handle SCSI Read operations is the Read state table. The bridge uses elements of the Read state table to generate data STUs, and to validate and bridge control operations.

The Read state table will include:

- M_len (from GSN header)
- Next Bufx, Offset
- STU_num (also used as next expected FC data frame SEQ_CNT)
- Initiator's buffer size

A Read operation is begun when the Initiator issues a Request_To_Receive message. When the bridge receives this message it will validate the Key values for the ST VC (i.e., I-Port, R-Port, R-Key), ascertain that the bridge has a free read state table entry, and extract the FCP command payload from the optional payload of the Request_To_Receive message.

The information that the bridge puts into the Read state table includes:

- FCP command payload
- I-id, the transfer length
- Receiver buffer size
- Expected FC frame SEQ_CNT (0)
- R-id (state table index, preinitialized)
- D_ULA from the S_ULA of the Request_To_Receive message
- D_ID from the target number field in the Request_To_Receive optional payload.

All other fields are preinitialized.

Upon successful completion of the above, the Bridge sends a Request_To_Send to the Initiator with the usual validation keys and the R-id which is an index to the Read state table.

When the bridge receives the Clear_To_Send message, it will validate it appropriately. The Blocksize must be the length of the entire transfer. I-Offset must be an integer multiple of 2K.

The Bridge now performs a login with the addressed FC target number if it has not already done so. If the login fails, it must perform a Put to the status PMR of an FCP RESPONSE frame with the response information indicating a nonexistent target.

The Bridge records the I-Mx, I-Bufx, and I-Offset values in the Read state table and sends the FCP command to the Target storage device on the FC Port.

When the Bridge receives an FCP data frame from the Target storage device, it will select a Read state table entry based upon the FC OXID and validate the SEQ_CNT parameter against the Expected SEQ_CNT. If the SEQ_CNT is not correct, the frame is discarded. If a valid frame is received, it will store the SEQ_CNT into the Param field (STU_num), and increment the Expected SEQ_CNT, storing it in the Read state table.

From the received frame length, the bridge computes a GSN M_len and write it into the GSN header. M_len is the FC frame payload length, plus 8 (SNAP header), plus 40 (ST header). If F_CTL<19> = 1, indicating the last frame of the sequence, set the Flags L = 1, T = 0, and I = 1 in the ST Header. The bridge must then send the header and the FC data on the GSN interface. The Bridge updates the Offset and Bufx in the Read state table to reflect that the Offset is incremented by the FC frame payload length. If the offset crosses the Initiator buffer boundary, Bufx must be incremented and Offset reset to 0.

Upon receipt of an FCP response frame from the Target storage device on an FC Port, the Bridge must free the Read state table entry and perform an ST Put operation to the Initiator containing the payload of the FCP response frame. The Put operation sends the FCP RESPONSE frame to the status PMR region.

Write Operations

Every outstanding write operation uses an entry in a Write state table.

An ST Request_To_Send message received by the Bridge indicates the start of a Write operation. When the Bridge receives this message, it must validate the Keys (I-Port, R-Port, R-Key) and ascertain that there is a free Write state table entry. It then extracts the FCP command from the optional payload of the Request_To_Send message.

At this point, the Bridge must login with the FC target if it has not already done so. If the login fails, the Bridge must perform an ST Put operation to the status PMR with an FCP RESPONSE frame indicating a nonexistent target.

The Bridge then allocates a Write state table entry and stores the I-id. The R-id and OXID are assigned a value that is an index into the Write state table. The D_id is initialized from the target number in the Request_to_Send optional payload. All other fields of the Write state table are preinitialized.

The bridge now sends the FCP command frame to the Target storage device using the FCP command payload and the transfer length.

When the bridge receives an FCP Transfer Ready frame from the Target storage device, it must validate the S_ID. It clears the Expected STU_num parameter in the Write state table and sends an ST Clear_To_Send message to the Initiator with Mx, Offset, and Bufx equal to zero, and R-id is an index into the Write state table.

When the bridge receives a data STU from the Initiator in response to the Clear_To_Send message, it must select a Write state table entry using the R-id value and validate the STU_num with the expected STU_num in the Write state table. If the STU_num is invalid, the bridge will discard the STU. If it is valid, the bridge will create an FC header with F_CTL<19> = L (last), and RO = Bufx shifted left 12 bits and OR'd with Offset. The bridge will send the resultant FC frame on the FC port and increment the expected STU_num in the Write state table.

When the bridge receives an FCP response frame, it must follow the same process as it does for the Read operation.

FC Link Event Handling

Any time an FC link event occurs, the Bridge will send an End for all outstanding data transfer operations on all VCs for this FC port and remove any FC login associations.

A problem that is not addressed in this document is what to do with messages that may arrive while the link is down. One option would be to discard all messages and let ULP timeout mechanisms handle the resulting condition.

Notes:

I-Offset as a multiple of 2K: This limitation could be relaxed if the bridge were capable of sending the first portion of an FC frame as one STU and the second portion as another STU. Also, permitting arbitrary starting offsets would require the Bufx and Offset generating logic to be substantially more complicated.

Max_STU >= 2K: As with the $I\text{-Offset} \% 2K == 0$ limitation, $\text{Max_STU} < 2K$ would be possible if the bridge could send a single FCP data frame as several STUs.