BENEFITS OF LRU-CENTRIC FIBRE CHANNEL TESTING

Yönet A. Eracar, Ph.D.
Teradyne, Inc.
700 Riverpark Drive
N. Reading, Mass. 01864
978-370-1608
yonet.eracar@teradyne.com

Abstract – Fibre Channel is a highly reliable, gigabit, serial interconnect technology. Commercial applications of the Fibre Channel technology allow concurrent communications among storage devices using upper level protocols such as Small Computer System Interface (SCSI) and Internet Protocol (IP). Fibre Channel technology is scalable and flexible due to its support of various topologies, such as dedicated point-to-point, arbitrated loops, and scaled switched networks [1].

Advanced avionics programs and applications have an increasing need for bandwidth while maintaining low latency, determinism, and reliability. Therefore Fibre Channel is being selected as an avionics communication solution for a variety of new military aircraft and upgrades to existing aircraft [2]. Testing at all stages (factories and depots) is necessary to guarantee the reliable and deterministic communication of avionics.

Fibre Channel avionics networks present new challenges to those responsible for maintenance testing at the depots:

- **The requirement of testing an avionics application communicating via an upper level protocol, while the test environment presents a low level, physical interface.**
- **Emulating the system environment of a line replaceable unit (LRU) to obtain complete functional test coverage.**
- **Storing and monitoring large amounts of data passing through the Fibre Channel network.**

Existing test instruments and emulators in the market operate at the Fibre Channel protocol level and they lack necessary support for LRUs. Testing at the Fibre Channel protocol level detects bad receivers, transmitters, media, and Fibre Channel processors, but fails to find problems in the inner workings of an LRU.

In this paper, we will address the testing challenges listed above with an LRU-centric approach. We will specify an LRU-centric Fibre Channel testing framework and provide implementation guidelines using example applications.

INTRODUCTION

Fibre Channel Technology

Fibre Channel is an accepted international standard that is administered by the T11 Technical Committee [9] of the International Committee for Information Technology Standards (INCITS). Fibre Channel technology is widely used by the commercial industry for Storage Area Network (SAN) applications. Fibre Channel is also used as a high speed data bus for avionics systems. Currently, Fibre Channel operates at data rates of 1.0625 Gbps, 2.125 Gbps, and 4.25 Gbps. Future data rates are projected to be 8.5 Gbps and 10 Gbps.

Fibre Channel has an architecture that contains 6 levels to isolate the complexities associated with each level (See Figure 1 for a comparison of Fibre Channel levels and Open Systems Interconnection (OSI) layers) [10].

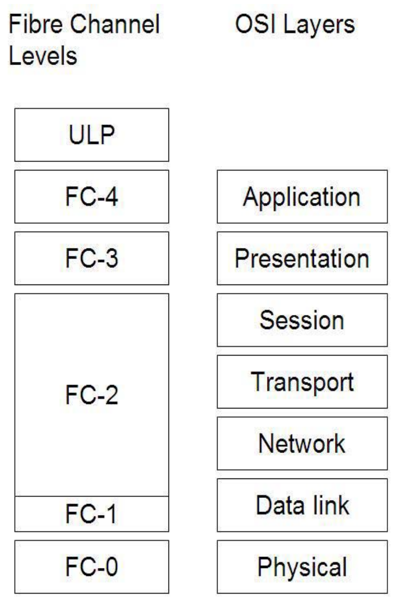

Figure 1 Fibre Channel Architectural Levels compared to OSI Layers

The FC-0 layer [5,6,7] defines the physical interface characteristics of Fibre Channel standard, such as data rates, optical and electrical variants used at each data rate, connectors, maximum distance capabilities, and other characteristics such as the wavelengths for optical and the signal levels for the electrical media.

The FC-1 layer defines the transmission protocol for Fibre Channel communication. The transmission protocol includes the serial encoding (8b/10b encoding is used), decoding, error control, and link control. The clock information is embedded in the serial data stream.

The FC-2 layer defines the signaling and framing protocol of Fibre Channel. The protocol involves the framing structure, the flow of control, and upper level protocol-independent class of services.

The FC-3 layer is reserved, but not fully specified, to provide common, protocol-independent services to support multiple protocols over multiple ports.

The FC-4 layer defines the mapping of the application to the network control structures. An application for Fibre Channel may be an upper-level protocol (ULP), such as Small Computer System Interface (SCSI), Anonymous Subscriber Messaging (ASM), Internet Protocol (IP), MIL-STD-1553, IPI, or High Performance Parallel Interface (HIPPI).

A Fibre Channel network consists of two or more devices connected by an interconnection scheme called a topology [8]. Sources and destinations of information in Fibre Channel network are called nodes. Each node has one or more node ports (N_Port, NL_Port, or generically just Nx_Port). A node port is a hardware function that allows the node to transmit and receive information using the Fibre Channel interface. Figure 2 illustrates a node, its ports, and communication links.

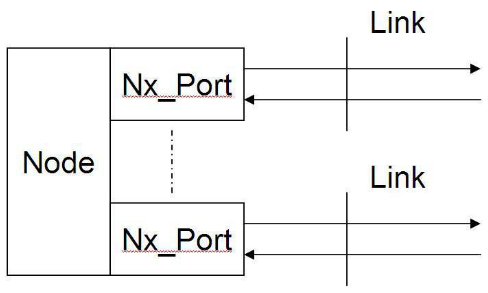

Figure 2 Terminology used in Fibre Channel Nodes

The Fibre Channel standard supports three topologies:

- Point-to-point topology: There are exactly two N_Ports connected together. No switch (routing function) is present.
- Arbitrated loop topology: There are a practical minimum of two and a maximum of 126 NL_Ports connected in a loop. Switch (routing function) is distributed into each NL_Port.
- Switched fabric topology: Up to 14 million ports can be connected together. Switch (routing function) is centralized via a generic environment.

The smallest information unit for Fibre Channel communication between two nodes is called a frame. A frame is made up of transmission words, which contain 4 bytes each. A frame contains a start-of-frame delimiter (4 bytes), a header of 6 transmission words, an optional payload up to 528 transmission words, a 4-byte long Cyclic Redundancy Check (CRC), and an end-of-frame delimiter (See Figure 3). Frame header contains information for identification (destination and source address), placement within a related collection of frames, flow of control, and basic service parameters. The payload is optional. The payload, when exists, can contain raw data, as well as, frames for an upper-level protocol.

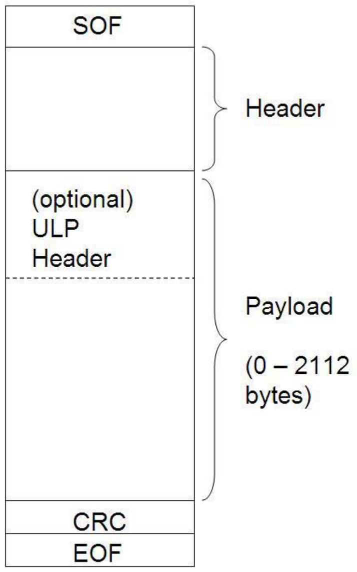

Figure 3 Frame Structure

One or more frames make a sequence. A sequence is a unidirectional transfer of a set of frames that move between the same source and destination nodes. One or more sequences make an exchange. An exchange is bidirectional (See Figure 4). A port can execute multiple exchanges simultaneously.

Certain processes are used to manage an exchange between nodes. Exchange management involves login procedures (i.e. Fabric Login, Point-to-Point N_Port Login) to establish communication session between two nodes, maintain and control the communication until the intended one or more exchanges are complete, and finally end the session with a logout procedure.

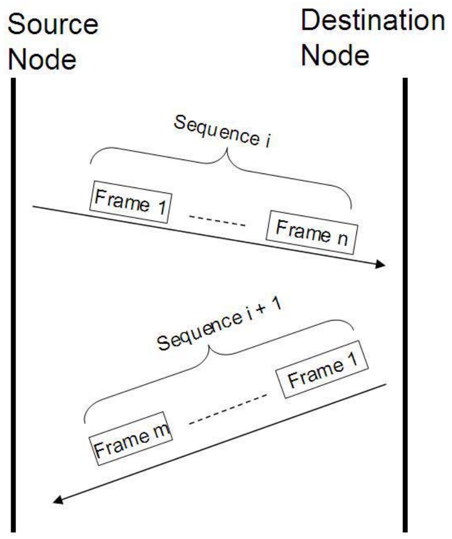

Figure 4 Exchange, sequences, and frames

The Fibre Channel standard defines multiple delivery options referred to as Classes of Service to support the needs of a wide variety of applications and data types. Fibre Channel implements Quality of Service (QoS) features, such as guaranteed bandwidth, guaranteed latency, acknowledged delivery, notification of non-delivery, end-to-end flow control, and guaranteed in-order delivery of frames within a sequence, using different classes of service. Generally, there are five user classes of services defined in Fibre Channel. However, only two of them, Class2 and Class-3, are widely deployed in the industry. Class-3 is the dominant class of service in the commercial and mil-aero markets. It is a best-effort, packet-switched class of service that resembles a datagram service with no significant QoS features [2].

Fibre Channel Usage and Test Requirements in Military/Avionics Applications

Advanced avionics technologies generate data in the form of voice and video data at higher rates than the current communication bus architectures (i.e. MIL-STD-1553). As an answer to this limitation, Fibre Channel has replaced the aging MIL-STD-1553 standard as the main control and data bus for military avionics. However, Fibre Channel is not the only high speed communication bus technology used by the avionics. A typical avionics Line Replaceable Unit (LRU) uses more than one data and/or control bus interfaces simultaneously (See Figure 5). While a Fibre Channel is used as the primary data and control bus, secondary data buses (i.e. source synchronous and/or slow parallel – Peripheral Component Interconnect (PCI), Virtual Machine Environment (VME)) are used as wide data pipes for short distances. For example, an LRU may receive a command from the Fibre Channel port and start transmitting data through its source-synchronous data port as a response.

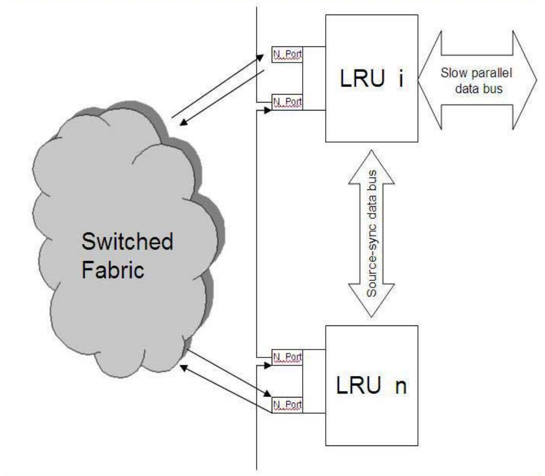

Figure 5 An LRU with multiple ports supporting different interfaces

Within the T11 Fibre Channel Standards group, a technical committee is responsible with the definition of upper-level protocols for military/avionics applications. To this date, three upper-level protocols are published; a mapping of MIL-STD-1553 over Fibre channel, ASM, and Remote Direct Memory Access (RDMA), which is a SCSI light protocol.

One advantage of using MIL-STD-1553 over Fibre Channel is that a user, who is familiar with the legacy MIL-STD-1553 standard, can operate using MIL-STD-1553 commands without being concerned with the transport layer. Figure 6 illustrates, the mapping of a typical MIL-STD-1553 operation to Fibre Channel, sending 1553 Bus Controller (BC) to - Remote Terminal (RT) Transfer Command.

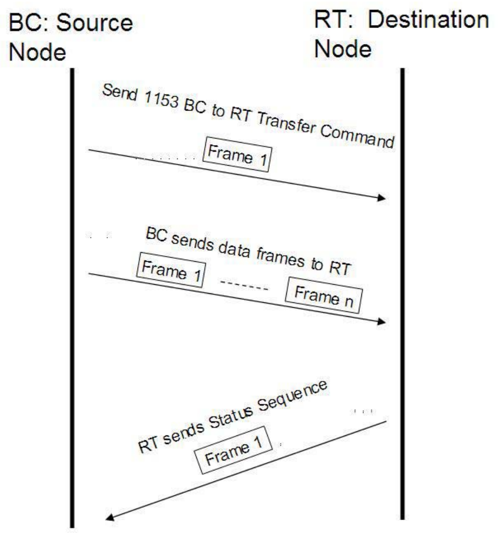

Figure 6 1553 BC-to-RT operation over Fibre Channel

ASM protocol used in military is illustrated in Figure 7. ASM is a basic Producer-Consumer paradigm. The avionics applications are designed to generate and consume data at periodic rates. Therefore, once the Producers and Consumers are synchronized in the same time domain, they do not need an external controller to maintain the communication.

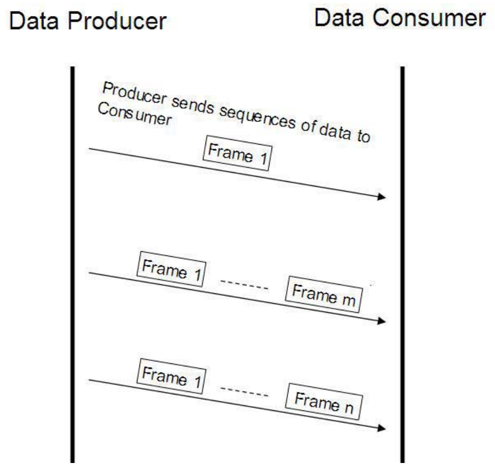

Figure 7 Example ASM exchange

Recent studies [2, 3, 4] listed the following minimum intermediate-level (the LRU is tested until a fault is identified to the LRU subsystem System Replaceable Unit) capabilities for a Fibre Channel test system:

1. Adherence to FC-0, FC-1, and FC-2 standards for lower-level protocols and FC-4 standard for upper-level protocols.
2. Capability of optical transmitter and receiver to operate within specified ranges.
3. Capability to accommodate multiple channels of connectivity for copper and fiber links concurrently.
4. Capability to generate "bad" traffic by programmatically injecting errors (i.e. Cyclic Redundancy Check – CRC error injection).
5. Capability to differentiate "bad" traffic from "good" traffic.
6. Capability to generate and read traffic concurrently.
7. Capability to generate traffic in response to a trigger condition.
8. Capability to generate traffic from a file in binary format (streaming).
9. Capability to send and receive a sequence of frames programmatically multiple times with a certain period.

One major Fibre Channel testing requirement common to all levels of maintenance is the support for upper level protocols over Fibre Channel. A desired test system (instrument and development environment) should allow the test engineer to program and test at an abstraction level that matches the upper level protocol, not the underlying low-level Fibre Channel protocol.

Current Fibre Channel Solutions

Currently, many Fibre Channel test solutions are targeting Storage Area Networks (SAN). They are either Fibre Channel device emulators (distance, delay, frame dropping, buffer-to-buffer credit estimation, frame configuration, and traffic generation) or parametric test instruments (i.e. bit error test, fabric login testing, signal loss).

A traditional Fibre Channel test environment includes active test tools that generate traffic conditions needed to test all of the fabric and equipment capabilities, together with passive protocol analyzers to transparently monitor traffic information within the network. Significant challenges are related to the integration of heterogeneous test tools and various application programming interfaces (API) in a common test environment.

However, these are not true automated test equipment (ATE) solutions. Traffic generation and monitoring is supported, but the testing is left as an exercise for the user. They do not have built-in support for upper-level protocols, i.e. nothing over FC-2 level. Creation of sequences and management of exchanges are handled in the users' test programs.

LRU-Centric Fibre Channel Testing

In [3], the authors point out two areas of interest in testing the Fibre Channel network implementation in the Joint Strike Fighter (JSF) program: a) the parametric testing of the laser transceivers, and b) the analysis of the JSF ASM protocol implementation. Although these are specifically listed as the testing requirements for the JSF program, we believe they reflect the common concerns about various military/avionics Fibre Channel implementations.

Our goal is to address the test requirements listed above with a LRU-centric solution that provides:

- Built-in test functionality for upper-level protocols as well as the Fibre Channel protocol.
- Test development environment (i.e. tools, software drivers) that supports FC-4 level and upper-level protocols in addition to FC-2 level.
- Software and hardware features (i.e. triggering, protocol encoding and decoding, deterministic and accurate scheduling of transmissions) to emulate the system environment of the LRU.

The following sections give a detailed list of features, with some examples, that are part of our solution. The features covered in these sections are:

- Creation of test data
- Controlling test data generation

- Expressing test flow
- Emulating the system environment

Our solution is a set of design guidelines for a test development and execution environment and indirectly the test instrument that will support this software abstraction. It should be noted that the features explained below are generic enough to support other high speed serial bus protocols with minimum changes.

Creation of Test Data

Static Test Data Sources

Test (Source) data, in its most generic form, is a collection of arbitrary transmission words. These transmission words are organized into structures called static data sources. There are two types of static data sources: transmit and expect.

Transmit data sources hold a sequence of transmission words that a port is to broadcast. The sequence is arbitrary, although most test applications use structured data in the form of properly formatted frames and sequences. The generic nature of data sources provides support for higher level abstractions like frames and sequences, yet still offers arbitrary error injection capabilities for those test applications that require it.

The transmission word set in a transmit data source can be presented in a grid with 3 columns (See Table 1):

- The first column contains the 0-based offset of the transmission word in the set.
- The second column contains a depiction of the transmission word.
- The third column contains either a count that indicates the total number of consecutive instances of that transmission word or a description of the use of that transmission word in a frame.

Offset	Word			#/Type
0	IDLE			4
4	SOFf			SOF
5	0x06	0x000011		HDR 0
6	0x00	0x000022		HDR 1
7	0x48	0x290000		HDR 2
8	0x46	0x00	0x0000	HDR 3
9	0x03B1	0xFFFF		HDR 4
10	0x00000000			HDR 5
11	0x00000011			DATA 0
12	0x22222222			DATA 1
13	AUTO			CRC
14	EOFn			EOF
15	IDLE			6
*				

Table 1 Presentation of Transmit Data

However, this form of creating test data is still a lower level of abstraction than desired, if the goal is only streaming data from node A to node B with no error injection. In this case, the specification of test data should be as simple as:

```
TRANSMIT SEQUENCE (
        SourceNode: A,
        DestNode: B,
        SourceData:<file_spec>,
        <class of service>,
        <Other service parameters:
                Max payload size,
                frame control, sequence
                control, ...>
);
```

The underlying implementation should handle breaking up the binary test data file into frames and sequences.

Expect data sources hold a specification of what sequence of transmission words is expected at the port's receiver. Expect data has two components: the data to expect and a mask to indicate which bits of that data are relevant in determining a pass/fail result. Expect data should be presented in a similar fashion as transmit data. The principal difference lies in the display of the data in indicating relevant and masked (non-relevant) data. If the cell contains all relevant data (it's not masked), then it is rendered as it would be for transmit data. Masked data is displayed according to the following rules:

- Data is always in hex.

- Hex digits that are completely masked are displayed as “-”.
- Hex digital that are partially masked are displayed as “?”.

Table 2 shows examples of combining logic state and mask data to produce expect data:

Logic State Data	Mask	Displayed Expect Data
0x12345678	0xFFFFC000	0x----?678
0x12345678	0xFFFFFFFF	0x--------
0x12345678	0x00000000	0x12345678
0x12345678	0xF0F0F0F0	0x-2-4-6-8

Table 2 Presentation of Expect Data

There are times when the test engineer might not care about the payload of a frame, but still would like to verify the frame header information, including the EOF. However, not all FC frames have the same size of payload. For help in these applications, the “#/Type” column of the Source Data presentation might use a count value, rather than a description, for a frame payload expects word. To specify any number of occurrences of a data word, one would specify the value 0 or the text “Any”. Table 3 shows how to expect condition an ASM frame with any payload, enforcing a minimum 4 word payload size to hold the ASM header. For example, if the received data does not have 0x04 hex value at the first byte of HDR 0 word (offset 1), it will constitute a failure for the test.

In addition to perform comparison between the expected data and the received data, a test application may need to store a part of received data (i.e. type of frame, service parameters) for future use, such as a conditional test for branching test execution or constructing a response frame. Therefore, we introduce the concept of test data variables that retain their values until the test ends or another received data store overwrites it. In Table 4, `<R_CTL>` is defined as a variable that will hold the contents of the first byte of HDR 0 word (offset 1). A variable’s scope (visibility) is limited to the expect data store that defines the variable and the execution threads (see Section: Expressing Sequences of Test Data) that use that particular expect data store.

Offset	Word			#/Type
0	Any SOF			SOF
1	0x04	n/a		HDR 0
2	0x-	0x-----		HDR 1
3	0x49	0x-----		HDR 2
4	0x-	0x00	0x---	HDR 3
5	0x---		0x---	HDR 4
6	0x-----			HDR 5
7	0x-------			4
8	0x-------			Any
9	AUTO			CRC
10	EOFn			EOF
11	IDLE			2
*				

Table 3 Presentation of Expect Data for an ASM Frame

Offset	Word			#/Type
0	Any SOF			SOF
1	**<R_CTL>**	0x------		HDR 0
2	0x--	0x------		HDR 1
3	0x--	0x------		HDR 2
4	0x--	0x00	0x----	HDR 3
5	0x----		0x----	HDR 4
6	0x--------			HDR 5
7	0x--------			4
8	0x--------			Any
9	AUTO			CRC
10	EOFn			EOF
11	IDLE			2

Table 4 Variable definition in Expect Data

Received Data Stores

Received data should be presented in a similar fashion as source data. The captured data words in a received data store are presented in a grid with 3 columns that serve the same purposes as with source data. The received data should be able to display collections of words as frames, if valid frames have been collected.

Received data presentation should contain test failure information for those execution thread actions that also do data comparisons. Mismatches between the actual and expected data are indicated with failure icons in the row header cell, and appropriate coloring changes (text color or cell background color) in the “Word” column cell (See Table 5).

Expect Data:

Offset	Word	#/Type
0	IDLE	6

Received Data:

Offset	Word	#/Type
0	IDLE	2
2	**ARB (FF)**	1
3	IDLE	3

Table 5 Presenting test failure information

Test Data Generation Control

Execution threads are sequences of actions that control what data a port transmits and the timing used to generate that data. Data transmissions can be programmed to occur once or repeatedly either with no added delay or at some fixed rate. Transmit actions in an execution thread can happen immediately one after the other, or only after the occurrence of a suitable trigger event. Possible trigger events include things such as:

- Frame header/payload content matches
- Triggers received from an external port
- Various error conditions (i.e. incorrect CRC, invalid frame data)
- A timer expired
- Complex triggers made from combinations of these trigger events

Receive actions are often related to transmit actions. For example, in a test we may tell the UUT to send us (the test instrument) some data. The UUT does this when it receives an appropriate command sent by our test instrument’s transmitter. Transmitter actions do not affect the test instrument’s receiver in any way. However, it is natural for the user to think of the receiver and transmitter operating in tandem, because such a logical grouping simplifies TPS development. Toward this end, certain actions may have associated receive actions. These actions are:

- Transmit Data
- Event Wait

Each receive instruction includes a timing window that determines when to start looking for a response (the data collection window delay) and how long to wait for it (the data collection window duration). There are two main types of receive instructions: collect data and compare.

Table 6 lists all actions that can take place in an Execution Thread.

Action	**Description**
Tx <data source>	Transmit Data: Send out a set of transmission words contained in a particular data source. The data can contain one or more frames or an entire sequence.
Rx <data store>	Collect Data: Store the received data in the given data store (i.e. memory, file).
?= <expect data store>	Compare Data: Compare the received data against the expected data.
EVENT <name of trigger event>	Event Wait: Wait for a particular trigger event to occur before continuing on to the next transmitter action.
PAUSE <time>	Pause: Wait for a certain period of time before continuing on to the next transmitter action.
LOOP <loop parameters>	Identifies the beginning of a repeated set of actions. The user can specify the number of times to repeat the action set as well as the period of each loop iteration.

Table 6 List of actions of an Execution Thread

Table 7 illustrates an execution thread for a given Nx_Port emulated by the test instrument. Line 0 indicates the beginning of a loop with a set of loop parameters. In this case the transmit action (Tx) at Line 1 will be executed 5 times with a period of 1 ms. Then the execution will halt at Line 2 until the test instrument receives the “Trigger A” event. After the test instrument receives the “Trigger A” event, the execution will resume and immediately “Frame A” will be transmitted (Line 3). Response from the UUT to “Frame A” will be stored in “Resp1” data store and will be compared against “Exp1” data source. If the test specified in “Exp1” fails, the execution thread will be marked as Failed.

ID	Action	
0	- LOOP 5 times @ 1 ms	
1	Tx “Sequence1”	
2	EVENT “Trigger A”	
3	- Tx “Frame A”	
4	Rx “Resp1”	?= “Exp1”

Table 7 An example Execution Thread

In addition to Pass/Fail conditions, a user may define conditions based on the values of the variables set by the received data.

Expressing Test Flow

In our solution, a test scripting language is used to specify the operation of a port during a Fibre Channel test. The script consists of commands to run execution threads and control constructs that determine the order in which those execution threads run. This section describes the test script capabilities in general terms. The most basic instruction is one that runs an execution thread. Complementing this instruction is a set of control constructs including (See Table 8 for detailed explanations on these instructions):

- Simple counted loop mechanism
- A mechanism to exit a loop prematurely
- If / ElseIf / Else constructs
- Something to indicate the end of a test
- Various signal generation and testing methods to provide software-based cross-port synchronization

Conditional control constructs test for two different types of conditions: thread conditions and signal states. A thread condition is one that is set as a result of executing an execution thread. Execution threads have three built-in thread conditions and an additional set of user-defined conditions as described earlier: Passed, Failed, or Executed.

A simple test that executes one execution thread and conditionally selects the next execution thread based on the pass/fail result of the first might look something like this (in pseudo-code):

```
Run thread “T1”
If “T1” Passed,
    Run thread “T2”
Else
    Run thread “T3”
```

Many test instrument hardware do not have any built-in test capability, so the task of determining the pass/fail status of thread T1 is up to the PC. The elapsed time between T1 finishing and T2/T3 starting is determined by the processing power of the PC, the amount of data to collect and test, and what the loading on the PC is. As a result, test scripts are better suited for emulating upper-level protocols and application-level functions, rather than providing real-time responses to particular frames.

Signals are a means for cross-port synchronization. The state of those signals can be tested and test script execution can be affected by these signal states. The test scripts provide ways to set and clear signals in one port, and test them in another port. Below is a typical application where activity on one port initiates activity on a second port:

```
On Port “A”:
    Run thread “T1”
    If “T1” Passed,
        Signal “S1”

On Port “B”:
    Wait for signal “S1”
    Run thread “T2”
```

The synchronization done here is based on the pass/fail status of thread “T1”.

Table 8 provides a complete list of Test Flow instructions:

Test Flow Instruction	**Description**
RUN threadName;	Runs the execution thread with the specified name.
DONE;	Indicates that the port is done. The test is done when all of the ports taking part in the test are done.
REPEAT n { <statement-list> }	Repeats the specified list of statements n times.
IF (<cond1>) { <statement-list-1> } **ELSEIF** (<cond2A>) { <statement-list-2A> } **ELSEIF** (<cond2B>) { <statement-list-2B> } **ELSE** { <statementlist-C> }	Conditional execution. The ELSE and ELSEIF clauses are optional. It is possible to have multiple ELSEIF clauses.
LEAVE IF (<cond>);	Exits an enclosing repeat loop if the specified condition is true. The IF portion is optional. If not present, the loop is exited unconditionally.
ASSERT signalName;	Asserts the signal with the specified name.
CLEAR signalName;	De-asserts the signal with the specified name.
WAITFOR (<signalCond>);	Pauses execution on the port until the specified signal condition is true.

Table 8 Test Flow Instructions

CONCLUSIONS

The following are the benefits of LRU-centric Fibre Channel testing demonstrated by our framework:

- The support for military avionics protocols (i.e. ASM, 1553, RDMA) allows programming at system level and enables system-level testing.
- LRU-level messaging emulates LRUs in their final environment. Real-time scheduling and stimulus-response pairing finds bad LRUs, not just bad Fibre Channel interfaces.
- Message-level pattern matching and error detection simplifies TPS programming compared to Fibre Channel protocol level testing. Providing an abstraction level closer to the LRU application will hide the details of protocol (physical) level programming.
- LRU-level programming and debug tools help the creation of more efficient and higher quality tests quickly.

ACKNOWLEDGEMENTS

I would like to acknowledge Mike McGoldrick, Dongxian Li, and Bill O'Neill for their contributions.

REFERENCES

[1] FCIA (Fibre Channel Industry Association) http://www.fibrechannel.org/
[2] Warden, G. and Fleissner, B., "Fibre Channel Testing for Avionics Applications", IEEE AUTOTESTCON'2004, San Antonio, TX, USA.
[3] McKinzie, F.C., "Achieving Greater Automation in Fibre Channel Test Equipment for Parametric and ASM Protocol Analysis and Testing", AUTOTESTCON 2003
[4] Barker, C.R., "Intermediate Level Maintenance Fibre Channel Testing", AUTOTESTCON 2002
[5] ANSI, 1994, FC-PH, "Fibre Channel Physical and Signaling Interface", ANSI X3.230:1994
[6] ANSI, 1997, FC-PH-2, "Fibre Channel 2nd Generation Physical Interface", ANSI X3.297:1997
[7] ANSI, 1998, FC-PH-3, "Fibre ChannelGeneration Physical Interface", ANSI X3.303:1998
[8] Kembel, R.W., "Fibre Channel: A Comprehensive Introduction", 2003, Northwest Learning Associates.
[9] T11 web site., Address: http://www.t11.org
[10] Precision Fibre Channel, "Fibre Channel Class", Training Manual, 2006.