NASA/TM—2011-217201

SpaceWire Tiger Team Findings and Suggestions

Joseph A. Ishac
Glenn Research Center, Cleveland, Ohio

November 2011

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include creating custom thesauri, building customized databases, organizing and publishing research results.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at *http://www.sti.nasa.gov*

- E-mail your question via the Internet to *help@sti.nasa.gov*

- Fax your question to the NASA STI Help Desk at 443–757–5803

- Telephone the NASA STI Help Desk at 443–757–5802

- Write to:
  NASA Center for AeroSpace Information (CASI)
  7115 Standard Drive
  Hanover, MD 21076–1320

NASA/TM—2011-217201

SpaceWire Tiger Team Findings and Suggestions

Joseph A. Ishac
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

November 2011

Acknowledgments

The Team would like to thank the contributions of many members of the CoNNeCT staff, including Linda Moore, Doug Reese, Larry Vincent, and Dale Walters, for attending and hosting meetings regarding the SpaceWire system and its status.

This report is a formal draft or working paper, intended to solicit comments and ideas from a technical peer group.

Level of Review: This material has been technically reviewed by technical management.

Available from

NASA Center for Aerospace Information
7115 Standard Drive
Hanover, MD 21076–1320

National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Available electronically at http://www.sti.nasa.gov

SpaceWire Tiger Team Findings and Suggestions

Joseph A. Ishac
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Abstract

This technical report intends to highlight the key findings and recommendations of the SpaceWire Tiger Team for the CoNNeCT project. It covers findings which are technical in nature, covering design concepts and approaches.

1.0 Introduction

The CoNNeCT SpaceWire Tiger Team (hereafter referred to as the "Team") was directed to observe the current operational system and provide suggestions on means to improve and correct deficiencies which would prevent CoNNeCT from adhering to its SpaceWire based requirements. These suggestions will be given to the CoNNeCT project to review and implement as deemed appropriate. It is also CoNNeCT's responsibility to properly test such modifications as the Tiger Team was not provided direct access to hardware or software.

The intended audience of this document is someone familiar with both the project and the technical structure of the software. Detailed background information on the project and subjects such as Direct Memory Access (DMA) will not be covered.

The Team held meetings with CoNNeCT staff members and also had a single meeting with the developer of the SpaceWire driver code. Outside of these meetings, the Team also took part and witnessed several testing sessions. The tests consisted of attempts to transmit data to and from the different software defined radios over SpaceWire and were performed on the engineering model located in the main CoNNeCT laboratory. The Team was not able to directly witness tests on the flight system. Test results deemed appropriate by the CoNNeCT staff were occasionally passed along to Team members and other results were shared upon request.

The remainder of this document is structured as follows. Section 2.0 outlines the major issues identified by both CoNNeCT and the Team. Section 3.0 outlines the major suggestions and solutions presented by the Team.

2.0 Major Issues

Upon formulation, the Team was introduced to several problems presented by the CoNNeCT avionics team. These issues provided a good starting point for the Team and through observation, analysis, and discussion the Team discovered additional issues that should be addressed. This section briefly outlines those issues. They are rationalized in more depth later in the document.

2.1 CoNNeCT Identified Issues

The avionics group identified two major issues of immediate importance, both of which involved DMA transfers. The first was the inability to correctly transfer bidirectional data over a single link while using DMA. The second was the inability to support more than one link using DMA at any one time. The former is necessary in order to facilitate efficient communication between components. The latter is necessary as there are four SpaceWire links and requirements to support more than one at a time. Doing so without the use of DMA would result in the CPU spending a majority of time servicing the SpaceWire links.

2.2 Team Identified Issues

The Team quickly identified an issue regarding the number of interrupts being generated by the SpaceWire drivers. The sheer volume of interrupts was causing the CPU to become overwhelmed and unable to process data correctly, especially at the higher data rates.

Through discussions with the various radio teams, it was discovered that the avionics software could not properly pass error free data. Thus, there was an issue with corrupted and incomplete data. Analysis of the results would show missing data, often at the beginning and end of a run (Ref. 1). However, there were also cases where there were multiple data "holes" present, or data missing in the middle of a run with valid data before and after the occurrence (Ref. 2). Finally, there were cases where no valid data would be received at all. Rather, "gibberish" would be recorded by the avionics software and appear to have a random pattern.

While observing testing, the Team also discovered an issue regarding proper systematic troubleshooting of software changes. As observed, changes to the software would require the presence of three or more CoNNeCT members to test and often led to unhelpful results, or in the worst case, crash the avionics hardware. To compound the issue, impromptu changes to the software would be made and rerun without due analysis backing such changes. Often the only result would be further crashing of the system, resulting in very inefficient use of engineering manpower and resources. Even the success criteria were poorly defined and varied—often to meet the expectations of the moment rather than the end needs of the system.

Studying the software structure and behavior, it was noted that the software does not fully conform to the SpaceWire specification. The discovered deviations would cause data to be missing or introduce unwanted data into the serial stream. The deviations are outlined in more detail in the following section.

Interviews with both the avionics team and the experimental users revealed a clear disparity regarding the type and level of service expected of the avionics software (Ref. 3). Specifically, disagreement within the project was found regarding the form and fidelity of the data being transmitted. Furthermore, the hardware architecture utilizes signaling rates which currently prevent the system from reaching the highest required data rates and the software driver structure seems poorly engineered to handle the uses described in the use case scenarios.

3.0 Findings and Suggested Changes

The following sections cover several key changes or observations that should provide a basis for improving the CoNNeCT SpaceWire drivers. We also discuss in some detail why each change is recommended and how it will help improve particular aspects of the code or architecture.

3.1 Reduce the Number of Interrupts

First and foremost, the system currently generates too many interrupts. Specifically, it generates an interrupt for every 250 bytes of data that is received by the SpaceWire card. Thus, a data arrival rate of 12.5 Mbit/s would generate an interrupt every 160 μs. In order to service each interrupt the CPU must essentially "shelf" what it is currently doing and invoke the routine to handle the incoming request. The time needed to perform that switch is often referred to as the interrupt latency, and a 2002 study of VxWorks showed this latency to be roughly 98 μs (Ref. 4). In addition the current software generates additional, internal interrupts for handling and passing data between driver functions when servicing the SpaceWire link.

Since CoNNeCT must strive to achieve data rates up to 100 Mbit/s it becomes questionable whether the current system will work. The time between interrupts would decrease even further. There would not be enough time for the CPU to handle the incoming requests, nor the other numerous tasks that the CPU must attend to. Reducing the number of interrupts would also allow the system to better handle other hardware devices as each generates its own set of interrupts.

Thus, the goal should be to minimize the number of times an interrupt must be generated. The most accepted way to perform such a task is to utilize DMA and device buffers to queue data in system memory prior to interrupting the CPU. The buffers need not be excessively large. For example, a 32 KB buffer in the 12.5 Mbit/s case above would improve the arrival rate to roughly every 21 ms from 0.16 ms. The SpaceWire hardware has a board level buffer which should be leveraged for both DMA and preventing overflows. In order to prevent data from becoming stale in the buffer a timeout is used to force an interrupt in the absence of incoming data. There are several efficient ways of implementing such a timer at the device driver level by leveraging clock ticks.

3.2 Leverage VxWorks Driver Classes

The current CoNNeCT SpaceWire software architecture utilizes complex structures, design patterns, and custom methods to handle the task of transferring data between the interface board and the CPU. The current system as designed will not accommodate the necessary requirements for servicing multiple SpaceWire connections as it generates a large amount of resource contention, which makes it impossible to operate practically. Leveraging the concepts in this section will allow for a much more manageable and robust system and reduce the overall risk of operating over SpaceWire.

The use of SpaceWire in CoNNeCT is purely for point to point data transfers and is analogous to many other serial devices, such as USB or RS232. The transfer of serial streams is a well supported concept and VxWorks contains native support for efficiently constructing and managing such character drivers (Ref. 5). Leveraging these VxWorks "built-ins" for character drivers allows the user to make standard system calls to interact with the SpaceWire ports. For example, a user could now call open() on "/spw0" and then issue standard write() and read() calls to transmit and receive data.

In addition, the interface would be initialized during system startup and be readily available. The system would automatically handle discarding data when the port is not open, and so forth. In contrast, the current SpaceWire manager task must handle the task of port management and currently reinitializes the interface each time the port is to be used, resulting in discarded data at the beginning of a file. Also, the current software must "guess" as to the completion of the user transfer, essentially closing a port after some set duration has passed. This has shown to truncate data files (missing data at the end of a file) during most tests.

All of these issues could be mitigated by constructing a system using the standard system calls, resulting in consistently complete data sets and more efficient use of the CPU. That efficiency is further increased when multiple SpaceWire interfaces are used simultaneously, as a properly written character driver would allow for the use of the select() system call to quickly service ports that had pending actions.

More information regarding these "built-in" functions are provided in the Appendix A.

3.3 Deliver Data According to Specifications

As with any independent implementation of a specification, there are occasionally discrepancies that cause the behavior of the protocol to vary from what is described in the specification. During the analysis of the current system, the Team dis-

covered two behavioral issues which impacted the data stream delivered to the application.

The first issue is regarding the handling of EEP terminated packets. An EEP marker is used to terminate a SpaceWire packet as a means of indicating that the payload may be incomplete. The technique is often used by SpaceWire switches which may need to truncate a packet while performing wormhole routing. Currently, the avionics software treats such markers as hard errors and resets the link when they are encountered. However, as defined in section 10.6.3.3 and 11.3.3 of the specification, EEP terminated packets are to be treated as normal, with the data passed to the application. Section 11.4 specifies that the link is reset only on "exchange level" errors, such as a parity error. Thus, treating EEP markers as errors causes undue data loss in two ways. First, by not passing the valid data contained in the packet ending with an EEP, and second, being unable to receive data as the link is being unnecessarily reset (Ref. 6).

The presence of EEP in such a simple link topology may be a result of including SpaceWire switches in the radio firmware since radios are constantly generating data and may be overwhelming the internal switch. Implementing the changes suggested in Section 3.2 would help mitigate this issue by keeping the SpaceWire link established after initialization.

The second issue is with regards to handling SpaceWire headers, which may be present at the beginning of a packet. In particular, the CoNNeCT team was noticing a 0x92 character header and passing it to the application user. SpaceWire headers in the range of 0x20 to 0xFE are considered "logical addresses."' Their removal by intermediate devices is optional depending on the network configuration. At the end node, the "header is stripped off and the packet put in a buffer which can be accessed by the destination task (Ref. 6)." Thus, the CoNNeCT team must remove the 0x92 characters before passing the data to the application.

3.4 Implement a Systematic Testing Structure

The Team was able to sit in on several testing sessions and received reports from numerous others. Throughout these tests, it became clear that a proper testing methodology was needed. Success criteria were never clearly defined. For example, a "successful" code update was shown to perform correctly in a short test case, but failed when tested over a longer duration.

While it was clear that the experimental leads were following set test plans, there were no provisions in place to supplement those test plans with the additional information needed to properly diagnose problems. The test plans were structured well to indicate success if the process went smoothly. However, should the avionics software crash or when seemingly "random" data was captured, there was insufficient supplemental information to help diagnose the potential issues. Additional information that was available came in the form of logging statements generated within the device driver software. Such statements at a device driver level are very costly in resources and are often discouraged as they can negatively impact the performance and timing of the system.

Setting up a test was also a costly endeavor. It required the assistance of several engineers to properly configure and execute commands on companion systems, such as the radios and data generator. This team would have to wait in standby as only several tests were attempted over hours of time. Communication of what changes were being made to the avionics software between runs and why the changes may help were unclear as their basis was never tied to any clear metric or observation. In some cases, all of the tests would result in the avionics crashing. Running tests to troubleshoot such crashes should not require the commitment of multiple engineers.

Also, there were issues referencing older code (in this particular case, July snapshots) that had a different set of issues, but was otherwise more stable than the current code according to previous test results. Although code revision software is in place, it seems difficult to reference changes based on date, perhaps either because the changes were left uncommitted, poorly tracked, or became "throw away" code that was kept outside of the revision control system (perhaps simply because it does not conform to the software architecture). The ability to reference which code is being run with each test is very important, even if that code falls into the categories listed earlier. Such code can still be useful as a means of troubleshooting the current build, perhaps simply by looking at the logic structure of both pieces and should be kept for reference.

The Team suggests implementing a proper testing methodology to solve these issues. The test procedures alone are insufficient. For example, supplement the data captured to flash with wire captures using a bus analyzer or other tool to capture data "on the wire". Run the avionics software via VxWork's debugger if possible. A test run should consist of a set of several procedures, run in sequence once. The data is then analyzed to observe why and where potential issues lie.

A correction could then be proposed, implemented, tested locally, and then another run issued. Each run should require the full CoNNeCT support for no more than 1 hr instead of 5. Following such a scheme also plays well into the revision control system and allows for more discrete change sets that are accounted for correctly and with proper rationalization.

To even further facilitate testing, it may be desirable to generate a few fixed test inputs and software or scripts to quickly validate the expected output from those fixed inputs. This would help alleviate the need of specialized equipment (such as the TSIM) to perform a basic run and may prove invaluable to other future tests such as during final system integration.

3.5 Recognize Signaling Rate Limitations

The current system utilizes a 100 MHz signaling rate to drive the SpaceWire interfaces, which is the maximum rate recommended by the manufacturer of the interface boards. Since SpaceWire utilizes a 10 bit encoding to transmit 8 bits of

data, the effective maximum throughput at 100 MHz would be 80 Mbit/s. However, recall that CoNNeCT is required to support a framed data rate of up to 100 Mbit/s. Accounting for the header overhead due to the framing and focusing on only the raw data that will be transmitted over the SpaceWire link reduces that to 97.7 Mbit/s (see Appendix B for a full listing of applicable data rates). However, that value exceeds the theoretical maximum that the link can currently support. To overcome this issue, it may be possible to overclock the SpaceWire boards, driving them at a higher signaling rate. Potential risks arise when overclocking boards. The boards may no longer operate in a stable fashion, and if proper attention is not given to voltages, may damage the board or host system. The CoNNeCT hardware teams will need to properly evaluate such risks. For reference, the current SpaceWire specification currently allows for signaling rates up to 400 MHz.

3.6 Switch to "Streaming Mode"

Upon interviews with the lead software developer for the low level drivers, it was discovered that a special operating mode was developed for use with CoNNeCT. This mode was called "Packet Mode" and constructed with the goal of passing extra line level details of SpaceWire transactions to the device users. Thus, special markers and flags were passed to the application instead of suppressed. Another operating mode that came with the device was coined "Streaming Mode" by the developers and simply passed the data bits from the hardware to the software user, leaving the extra information to be obtained by requesting specific metrics.

While the "Packet Mode" has been in use for over a year, the behavior that is needed is actually described best by the "Streaming Mode." In this mode only the serial data is sent over the PCI bus to be processed by the CPU. Sending additional information with the data introduces unnecessary complications and forces the CPU to spend additional resources to complete an action that the hardware has already performed. It forces you to begin to complicate the driver instead of keeping it simple and concise.

"Streaming Mode" presents the driver developer and user with exactly what is needed—the actual data being sent over the SpaceWire link. The driver does not need to be complicated by deciphering the stream and users can still have access to key metrics when needed by specifically asking the hardware, which keeps its own registers and counters to maintain such metrics.

3.7 Adopt Proper Layering Concepts

The specialized "Packet Mode" discussed earlier was likely developed mainly due to a misuse of proper layering concepts and not adhering to proper layer separation. A large part of this misunderstanding was driven by the concept of SpaceWire "packets." While SpaceWire streams data between two peers, it places this data into segmented payloads, which the specification calls packets. The size and properties of these packets are not correlated to the data that needs to be sent, but are instead a means of allowing multiple users to transmit streams simultaneously over a single link. CoNNeCT operates all SpaceWire links such that only a single stream is ever present as all modems have a dedicated connection for transmitting data. The one modem that also uses SpaceWire for commanding, has a second SpaceWire interface and does not send both data and control over a single link.

Unfortunately, the term packet is also popular at many other layers in the communication stack, such as IP packets. In addition, the links utilized a SpaceWire packet size that was sufficient to fit an entire radio frame. While the size of the SpaceWire packets is acceptable and actually desirable, it led to misrepresenting a SpaceWire packet as a higher level encapsulation rather than being treated as a simple means to deliver data across the link. As such, the current drivers attempted to perform several validity checks on the contents of each SpaceWire packet.

Unfortunately, attempting to validate SpaceWire packets is an impossibly difficult task as there is not enough information to properly ascertain validity. At the SpaceWire level, data is in actuality a stream. Within that stream are other constructs to aid in extracting the proper information. Parsing those constructs is not the job of the SpaceWire driver, but rather of a "higher layer." In the CoNNeCT payload, these constructs differ for each of the three radios in use. Thus, it is up to the application or user of the stream to correctly process the information contained within it. It is the driver's job to simply deliver the data it receives.

To reiterate, the driver code should never attempt to validate the actual data contents of a SpaceWire stream, nor should it make any judgment based on artificial heuristics that are not part of the SpaceWire specification. It should instead focus on processing the data stream and leave validation to the users of the data which can properly evaluate the data.

4.0 Conclusion and Suggested Approach

The Team (see Table I) suggests that the software team take their existing knowledge on interfacing with the SpaceWire hardware and apply it towards a complete rewrite of the driver interface. Rather than attempting to utilize complex design structures, the team should focus on developing routines which adhere to the driver design structure detailed by WindRiver. This would allow users to leverage the built-in functions and allow for proper capturing of data without the need for guessing at durations or startup times. It would also allow for the user to efficiently manage multiple SpaceWire interfaces, and provide simple interfaces such as select() to quickly service only the necessary interfaces.

The Team believes that a rewrite is necessary as the solutions outlined in this document can not be simply inserted into the current code. A rewrite would allow CoNNeCT to correct

the overall approach of the driver. Likewise, the Team does not feel that the current code can be made to support the current requirements due to these flaws.

TABLE I.—TIGER TEAM MEMBERS

Name	Org. code	Role	E-mail
Michael Lichter	DPA	Team Lead	michael.j.lichter@nasa.gov
Monty Andro	DPC	Electronics Engineer	monty.andro@nasa.gov
Joseph Ishac	RHN	Computer Engineer	jishac@nasa.gov
Michael Mackin	DPS	Computer Engineer	mackin@nasa.gov
Linda Moore	DPS	Computer Engineer	Linda.moore@nasa.gov
Mary Jo Shalkhauser	DPC	Electronics Engineer	maryjo.w.shalkhauser@nasa.gov
Glenn Williams	DPA	Electronics Engineer	glenn.l.williams@nasa.gov

References

1. J. Downey, D. Mortensen, et al., “Raw Data, Harris Test Results,” GRC–CONN–TEST–0669, NASA Glenn Research Center, Dec. 2010.
2. J. Nappier, W. Eddy, et al., “Raw Data, GD Test Results,” GRC–CONN–TEST–0469, NASA Glenn Research Center, Dec. 2010.
3. Personal Interviews, CoNNeCT Validation Test Team, Nov.–Dec. 2010.
4. B. Ip, “Performance Analysis of VxWorks and RTLinux,” Technical report, Department of Computer Science, Columbia University, 2002.
5. P. Ayyalasomayajula and A. Tully, “VxWorks - Device Drivers in a Nut Shell,” Technical Report, Department of Computer Architecture, BarcelonaTech, 2001.
6. European Cooperation for Space Standardization, “SpaceWire - Links, nodes, routers and networks,” Space Engineering Standards Document, ECSS–E–ST–50–12C, Jul. 2008.

Appendix A.—VxWorks Driver Support

As noted in WindRiver's driver development guides, VxWorks driver architecture provides support for “Serial Drivers” allowing for easy integration of any components which behave in a serial fashion as does SpaceWire. The devices will become connected to the I/O system and users can gain access to the devices by making use of standard calls such as open(), read(), write(), ioctl(), and so forth. The system would initialize the devices when the system starts and there would be no need to constantly bring the interface up and down.

For example, say that the driver code creates a SpaceWire devices called “/spw0”, “/spw1”, etc. It would be possible for a user to perform the following techniques to interact with a device of this type. In the simplest case the user could open the device, transfer data, and then close the descriptor.

```
if (( fd = open( “/spw0”, O_RDWR, 0)) != ERROR)
// Process data from the radio with read()
// Send data to the radio with write()
// When we are finished close the descriptor
close(fd)
```

Furthermore it would be possible to perform other operations such as setting adjustable parameters within the driver or interacting with multiple open device descriptors as shown in the following code example. The system call ioctl() can also be used for reading metrics such as the number of times the hardware reset the link.

```
// Set a parameter SW_MODE on the spw0 device
ioctl(fd, SW_MODE, 1)
// Use select to read from a descriptor set
select(FD_SETSIZE, &fdset, NULL, NULL, NULL)
```

Detailed descriptions of how to construct a driver which conforms to this model and how to leverage the calls shown above are provided by the WindRiver documentation entitled “Device Driver Fundamentals.”

Appendix B.—Data Rates and Formats

The CoNNeCT payload supports three different radios. Each radio is required to support different data rates for sending and receiving data in what are considered "forward" and "return" paths. Specifically, CoNNeCT's requirement documents call for a specific framed data rate to be supported. Table II shows a breakdown of an uncoded data link message (the data that is transmitted over the RF link). The "frame" is highlighted and consists of the frame header (TFPH) and the frame data (User Data) and totals 256 bytes. The ASM is a marker used to aid the receiver, but is not considered part of the frame. The size and definition of a frame is consistent for all three radios.

TABLE II.—DATA FRAME

Data link format	ASM	TFPH	User data
Size in bytes	4	6	250

However, even though the format is the same, each radio transmits a different portion of the data link over the SpaceWire interface for processing by the avionics software. This slightly alters the data rates necessary over SpaceWire to support the framed data rates required in the specification documents. Furthermore, the signaling rate to each radio link to the avionics is different. The maximum theoretical data rate possible over a link is capped to 80 percent of the signaling rate since SpaceWire utilizes 10 bits to encode 8 bits of data.

Table III shows a breakdown of the three different radios when both sending and receiving data. Thus, each highlighted pair represents a single radio. The first two columns show the signaling rate and the corresponding maximum theoretical data rate. The third column shows the maximum required rate as specified in the system requirement documents for sending and receiving framed data, which may differ per direction. The fourth column shows the type of data that is transmitted over the SpaceWire link. For example, "F-TFPH" indicates that the data being sent is a frame without ("-") the TFPH. If we once again reference Table II, we can find this to mean that only the "User Data" is sent. The final column shows the modified rate needed to support the format in column four. This is the rate that must be supported by the SpaceWire link to achieve the necessary system requirements. As mentioned earlier in the document. The last line shows how it will be impossible for the third radio to achieve the 100 Mbit/s rate due to the signaling rate limitations.

TABLE III.—SPACEWIRE DATA RATES IN MBIT/S

SpaceWire signaling rate (MHz)	Max data rate	Framed data rate (required)	Actual stream over SW	Necessary data rate over SW
24	19.2	0.072	FRAME	0.072
24	19.2	1.000	FRAME	1.000
33	26.4	0.769	F+ASM	0.781
33	26.4	0.769	F+ASM	0.781
100	80	12.50	F-TFPH	12.207
100	80	100.0	F-TFPH	97.656

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-11-2011	Technical Memorandum	

4. TITLE AND SUBTITLE
SpaceWire Tiger Team Findings and Suggestions

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Ishac, Joseph, A.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
WBS 289972.10.03.03

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
John H. Glenn Research Center at Lewis Field
Cleveland, Ohio 44135-3191

8. PERFORMING ORGANIZATION REPORT NUMBER
E-17886

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITOR'S ACRONYM(S)
NASA

11. SPONSORING/MONITORING REPORT NUMBER
NASA/TM-2011-217201

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-Unlimited
Subject Categories: 60 and 61
Available electronically at http://www.sti.nasa.gov
This publication is available from the NASA Center for AeroSpace Information, 443-757-5802

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This technical report intends to highlight the key findings and recommendations of the SpaceWire Tiger Team for the CoNNeCT project. It covers findings which are technical in nature, covering design concepts and approaches.

15. SUBJECT TERMS
Computers; Computer programs; Radio equipment; Operating systems; Modems

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email:help@sti.nasa.gov)
U	U	U	UU	16	19b. TELEPHONE NUMBER *(include area code)* 443-757-5802