SANDIA REPORT
SAND2011-8284
Unlimited Release
Printed September 2011

SpaceWire Model Development Technology for Satellite Architecture

Brian Van Leeuwen, John M. Eldridge, and Jacob Leemaster

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2011-8284
Unlimited Release
Printed September 2011

SpaceWire Model Development Technology for Satellite Architecture

Brian Van Leeuwen
Critical Infrastructure Systems, 5628

John M. Eldridge
Embedded System Engineering, 5632

Jacob Leemaster
High Integrity SW Systems, 2622
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-1027

Abstract

Packet switched data communications networks that use distributed processing architectures have the potential to simplify the design and development of new, increasingly more sophisticated satellite payloads. In addition, the use of reconfigurable logic may reduce the amount of redundant hardware required in space-based applications without sacrificing reliability. These concepts were studied using software modeling and simulation, and the results are presented in this report. Models of the commercially available, packet switched data interconnect SpaceWire protocol were developed and used to create network simulations of data networks containing reconfigurable logic with traffic flows for timing system distribution.

CONTENTS

FIGURES

ACRONYMS

EEP	end of error packet
EOP	end of packet
ESC	escape
FCT	flow control token
LChar	link-character
NChar	normal-character
PDF	probability density function
SNL	Sandia National Laboratories
SOH	state of health
SpW	SpaceWire
SpWBS	SpW Broadcast Server

EXECUTIVE SUMMARY

Sandia National Laboratories (SNL) is pursuing the development and use of packet switched data communications networks that use distributed processing architectures. These architectures have the potential to simplify the design and development of new, increasingly more sophisticated satellite payloads. In addition, the use of reconfigurable logic may reduce the amount of redundant hardware required in space-based applications without sacrificing reliability. These concepts were studied using software modeling and simulation, and the results are presented in this report. Models of the commercially available, packet switched data interconnect protocol – SpaceWire (SpW) – were developed and used to create network simulations of data networks containing reconfigurable logic with traffic flows for timing system distribution. This report covers two aspects of the modeling activity conducted. One aspect is the development and construction of the general purpose, high-fidelity SpaceWire models. The second aspect is the application of these models to a specific use case.

The modeling and simulations utilize the OPNET Modeler simulation environment. The OPNET tool is widely used for performing network and communication simulations. The system performs discrete event simulations where each data byte or packet within the network is accurately generated, transferred, and processed as it would be in an actual network. While OPNET Modeler provides a large library of standard network models for computers and typical computer networks, it does not have specific features for the SpW protocol. It does, however, provide the features necessary to add new or custom protocols to the model environment. The design team used these extension features within OPNET Modeler to create a set of general purpose modules to represent many of the network elements or basic building blocks to represent most SpaceWire networks. The modules include models for SpW nodes, routers, broadcast servers, and links. These modules can be arranged to represent networks during the design stage that can then be analyzed for desired behavior.

The second aspect of the report is an in-depth analysis of the accurate distribution of system time across a SpW network. The feature set of the SpW protocol has not fully matured to inherently and completely include the distribution of system time. To accomplish this task, the team developed a packet broadcast mechanism that would layer upon the standard SpW protocol. A representative SpW network was constructed within the OPNET Modeler simulation environment. Based on this network representation, several simulations were executed to study the behavior of the network with respect to packet transmission time, jitter, and the accuracy of distributed system time.

The SpW models provide a generalized tool for examining network behavior and will represent most network designs. The final section of the report discusses future work and directions. SNL's extension of the SpW protocol is in fact a collection of protocols that include the base protocol for link startup, transmission and routing, and extension protocols to provide remote memory access and reliable data delivery. Future extensions to the model environment would include modules for remote memory access (RMAP) and the remote data delivery protocol (RDDP) for reliable data delivery.

INTRODUCTION

The European Space Agency, in collaboration with other international space agencies, specifies and supports a serial data link standard to enable the transfer of large amounts of data on board satellites. The standard, named SpaceWire (SpW) and defined in Reference 1, is a satellite communication network based in part on the IEEE 1355 standard of communications. A SpW network is typically comprised of a number of links, nodes, and routers. SpW routers are necessary to coordinate traffic between indirectly connected processing nodes and to expand the node's link connectivity since nodes can only be directly connected to a limited number of other nodes. Routers also reduce the number of point-to-point links and enable redundant paths in case of link failures. The current SpW standard describes a mechanism that can enable modern satellite systems to transfer large amounts of data on board the satellite with link data rates that can range from 2 Mbps up to 400 Mbps. The total bandwidth of a SpW network expands with the increasing number of nodes and links within the network.

Sandia National Laboratories (SNL) has investigated the SpW protocol for satellite payload communications. The protocol provides a functional starting point for developing and implementing a payload communication capability. However, there are gaps in the protocol's desired functionality. A robust set of communication features are desired to distribute time information, reliably (without error) distribute data files, and to reconfigure network routing in the face of node failures. To explore these communication issues, SNL has developed high-quality network simulation models of the SpW protocol, nodes, links, and routers. The models are developed using the OPNET network modeling and simulation software package [2]. This software allows the creation of discrete event simulations and custom model development. This capability can then be used to precisely model SpW protocols and network architectures.

A number of limitations in the SpW protocol required the development of extensions to SpW. First, the SpW standard does not include quality of service with the exception of time codes. Most payloads require the ability to prioritize messages, and one mechanism to do that is through a broadcast capability. The second limitation of SpW is the lack of a standard time distribution mechanism. The proposed time distribution mechanism can then use the broadcast extension. While the SpW protocol natively offers a mechanism to broadcast a time trigger, it does not provide a complete method to distribute system time. The approach that SNL took to resolve this issue is to broadcast a time value from a master node within the network to all other nodes. Each node receiving the broadcasted time value would store that value and apply it on receipt of the next SpW time trigger.

A requirement for any approach to a general broadcast method was that it layer upon the existing SpW standard. That is, it would be compatible with the existing protocol and not necessitate the recreation of existing intellectual property or the revision of the existing SpW standard. Rather, the goal was to extend the standard to include the new capability.

With the objective of enabling the development of a SpW broadcast method and the development and analysis of a time distribution function, a high-fidelity SpW modeling and simulation capability was developed. The SpW models provide a method to validate our candidate approaches and to evaluate quickly the effect of different network topologies and operating

parameters on the accuracy of the timing distribution. Additionally, models of SpW nodes and SpW wormhole routers provide a means to create and evaluate system architectures under various network traffic conditions. Network device failures and fault-recovery approaches can be analyzed using the models and simulations. Analysis capability with the model includes impacts of device failure and network reconstitution. Our model-based approach enables the design team to verify protocol effectiveness and identify design sensitivities and margins.

MODEL DEVELOPMENT

The SpW protocol standard is comprised of various mechanisms to meet the objective of high-performance onboard data handling. To obtain detailed analysis of the SpW operation, the protocol is modeled in high detail. SpW analysis should account for how SpW networks combine application data and control plane data on the same links. Application data is passed as one or more data packets that are composed of characters. At the link exchange level, data and control characters are separated into two types: link-characters (LChars) and normal-characters (NChars). LChars are those that are used in the exchange level and are not passed on to the packet level. Examples of LChars are the flow control token (FCT) character, the time-code character sequence, and the escape (ESC) character. NChars are the characters that are passed on to the packet level as application data. SpW encodes eight-application data bits into 10-bit NChars. Further details on the SpW standard can be obtained from Reference 1.

Our SpW model development is done in the OPNET Modeler (Version 15.0) network simulation environment [2]. OPNET Modeler includes an extensive model library of network devices; however, OPNET Modeler does not include SpW models in its standard model library. Fortunately, OPNET Modeler does include the capability for users to develop node, link, and router models based on custom protocols. Since we expect SpW to be utilized in new satellite designs, having high-fidelity models available to perform analysis is important and the investment in time is justified.

OPNET Modeler includes rich mechanisms to create network traffic. In our time synchronization analysis, we are able to clearly identify when messages are created and when they arrive at their intended target. The messages can be general application layer traffic, traffic supporting system time distribution, state-of-health (SOH) messages, or broadcast messages. Application layer traffic can be generated to represent actual data files being transported through the network. OPNET Modeler also includes extensive probing of the network capability. Collecting data on packet arrival times, end-to-end delays, and packet jitter are possible. Probes measuring queue sizes and network link utilization can easily be configured.

A modeling objective was to have representative models of the various SpW modules and protocol requisites to support system design activities in all phases of a project. These project phases range from custom protocol extension analysis to assessing SpW architectures and their operation under stressful scenario conditions resulting from possible link and node failures. In pursuit of this objective, we developed the SpW models to be modular. The modular approach enables the combination of endpoints and routers in various architectures. Figure 1 illustrates an example SpW network. Also in Figure 1 is an illustration of one of the nodes in the example

network. In this example, each node is comprised of three specific modules: a SpW application node (i.e., Node_12), a SpW wormhole router (i.e., Node_1012), and a SpW broadcast server (i.e., SpWBS24). The SpW router is the connection point that combines or interconnects the various applications nodes and broadcast servers.

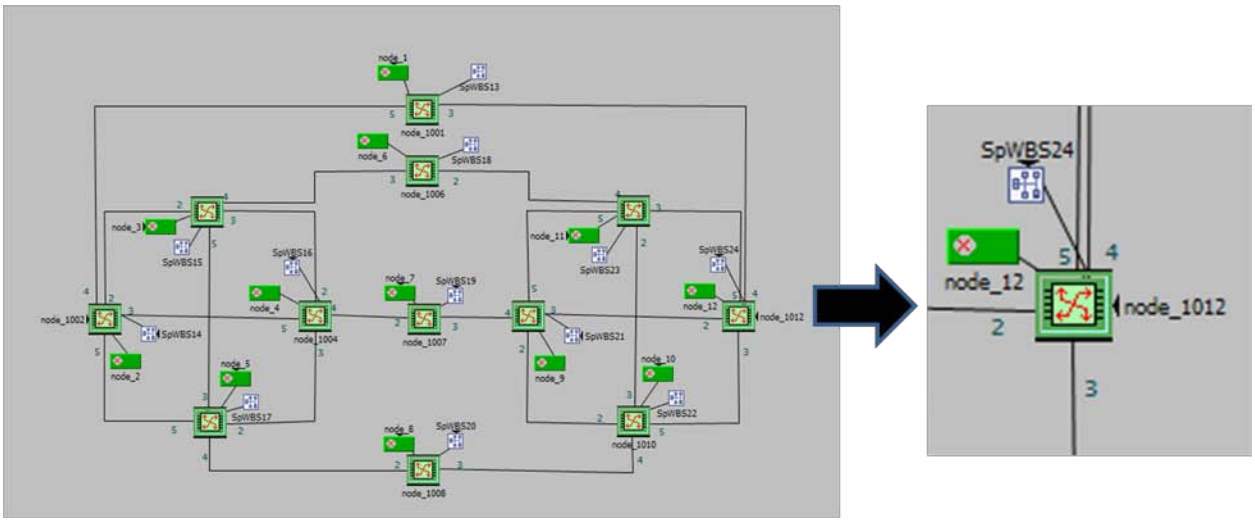

Figure 1. Example SpW network topology.

The image on the right side of Figure 1 illustrates how a single node in SpW network is created. A single processing node is comprised of a router node model (i.e., Node 1012 in the figure), a host node (i.e., Node 12 in the figure), and a broadcast server (i.e., Node SpWBS24 in the figure). Each of these node models is connected by a SpW link model.

The following sections describe details on what is implemented in these four fundamental models and how the implementation is done.

SpW Host Node Model

Our model development begins with development of a SpW host. The host supports various applications with a single communication protocol stack. The protocol stack implements the various SpW protocols that implement many features of the SpW standard including the functionality at the various communication-stack levels. It faithfully implements the disassembly of application layer data (i.e., noted as “cargo” at the packet level) and the reassembly of the resulting NChars at the destination node. Additionally, processes such as the startup sequence, flow control, Time Code process, and realistic representation of various buffering and queuing functions are included.

The SpW Host Node model is shown in Figure 2.

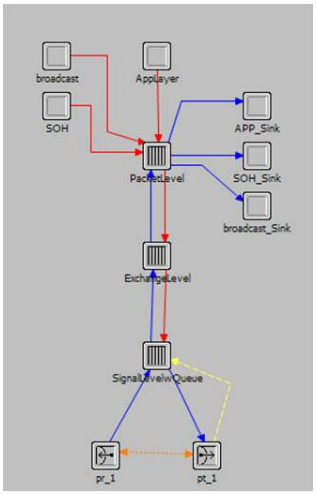

Figure 2. SpW Host Node model.

As shown in Figure 2, the SpW Host Node model can be decomposed into several process models with each residing in one of the square icons in the figure. Following is a description of each of the process models.

AppLayer, Broadcast, and SOH Process Models

Each of these models generates data packets of a specified size and a specified packet generation rate. The packet generation can also be defined by a specification file. The name of the process indicates the type of packet being generated and signals the SpW protocol to act on the packet as appropriate. SOH packets can be defined as a periodic packet that originates from each node. Each process can be configured independently and thus will create data packets independently.

App_Sink, Broadcast_Sink, and SOH_Sink Process Models

The sink process models receive the packets that were destined for the specific node. When a packet is received, the time it was created is noted along with reception time. These performance metrics are recorded for analysis.

PacketLevel Model

This process implements the segmentation and reassembly of packets. This process receives data packets from one of the packet generation processes and segments into NChars (8 bits data, 10 bits with overhead). When NChars are received by this process, the NChars are reassembled into data packets. This process creates and receives the end of packet (EOP) character, indicating the complete packet has been transmitted. The process also manages the reset of the receive sequence if an end of error packet (EEP) is received. When FCTs are received by the Exchange Layer, the Exchange Layer creates a control message indicating additional NChars can be

transmitted. On a receive process, this layer notifies the Exchange Layer of available buffer space. Thus the Exchange Layer can then request more NChars by sending more FCTs.

ExchangeLevel Model

This process manages the flow control in coordination with the opposing SpW interface. The process receives notification from the Packet Level process on available buffer space and creates and transmits the necessary FCTs to the opposing interface. The process also receives FCTs created and transmitted by the opposing interface. The received FCTs are used to manage the transmission of NChars from the host node. Error characters are received and processed by resetting the interface flow.

Null characters are processed by this layer. Null characters are used to initiate connections and are transmitted when no other character is being transmitted. In order to minimize processing of Null characters in the model an abstraction is introduced. In the model the Null characters that are part of the connection setup process are faithfully modeled; however, the constant stream of Null characters that real SpW interfaces transmit are not modeled. This is a necessary abstraction since the process of transmitting and receiving Null characters for each link not involved in transmitting NChars or Time Codes becomes a large consumer of simulation resources. The impacts to network Time Code transmissions such as jitter are included in the model as a random delay associated with the transmission of a Null character.

Additionally, the ExchangeLevel process creates and receives the SpW Time Codes. The Time Codes are processed in this layer and made available to other layers.

SignalLevelQueue Model

The SignalLevelQueue process model creates a queue and accepts SpW characters from the ExchangeLevel process. Characters received from the upper layer are either immediately transferred to the transmitter or, if the transmitter is busy, placed in a queue for later transmission. Immediate transfer to the transmitter means the character is sent to the transmitter, but if the transmitter is sending an NChar, for example, that NChar transmission must be completed before transmitting the character. In the case of an NChar, a time delay equivalent to transmitting between one and ten bits will be incurred. Otherwise, if the queue of this layer model has characters, the arriving character is added to the queue in a first-in, first-out basis.

This layer manages the immediate transmission of SpW Time Codes. According to the SpW standard NChars, EOP, and EEP characters are transmitted on the SpW link in the order they are received. However, Time Codes are immediately transferred to the transmitter regardless of whether there are characters in the process queue. Thus a Time Code can be delayed only by the SpW character being transmitted, whether it is a Null, NChar, EOP, or EEP. Thus if a SpW host is transmitting an application layer packet, the application layer packet will be segmented into NChars and the NChars will be modulated onto the SpW link with the possibility of only Time Codes to be inserted into the NChar flow.

Node Receiver and Transmitter (pr_1 and pt_1) Models

The node transmitter and receiver are connected through a full-duplex link that is configured to a specific data rate. Since SpW supports heterogeneous link data rates in a single SpW network, the transmitter and receiver must be capable of adjusting its data rate to the link capacity. Additionally, the transmitter signals the SignalLevelQueue process of its transmission state. This is necessary since SpW implements a form of transmission priority with its near-immediate action on SpW Time Codes. Thus in this implementation the transmitter queues are not used since priorities cannot be managed within the queue, and thus they are performed by the SignalLevelQueue. Note that the SpW standard describes a start sequence that begins with each link's data rate set to 10 Mbps. According to the SpW standard, the links then can adjust to another data rate. Our SpW model does not implement this mechanism and each link should be configured to its expected data rate for the experiment duration.

SpW Wormhole Router Node Model

A critical component to obtaining realistic simulation analysis results in SpW is the representation of the SpW router. SpW routing is based on wormhole routing [1]. In wormhole routing the first byte of the packet, or NChar, contains a destination address that the wormhole router uses to determine the output port to which it will direct the received NChars. If the identified output port is not being used, then the NChar is immediately routed to that port. The port is marked as busy and all remaining NChars for that flow are immediately forwarded to the port. During the transfer of data through the receiving and transmitting ports, the ports are considered busy until the last character of the packet has passed through the router. If the router determines the requested output port is busy, then the flow-control stops the incoming NChar stream at the input port until the output port completes its current flow and is released. Each SpW port manages its data flow with its opposing port and will stop the flow of incoming NChars by ceasing to send flow control tokens to the source node.

SpW protocols associated with both endpoints and routers manage the flow of the various SpW standard data and control characters. The ports manage the exchange of flow control tokens that control the rate of data flows since a SpW network may have links with asymmetric data rate capacity. Additionally, the protocols manage the buffering of NChars and the near-immediate transmission of Time Codes. For analysis, these protocol effects are faithfully modeled.

The node model of the SpW wormhole router is shown in Figure 3. This router model supports up to six interfaces; however, it can be expanded to any number of interfaces. Figure 3 illustrates six groups of process models that include the SignalLevelQueue process, the transmit process, and the receive process. These process models are identical to the processes in the host node described in the above section.

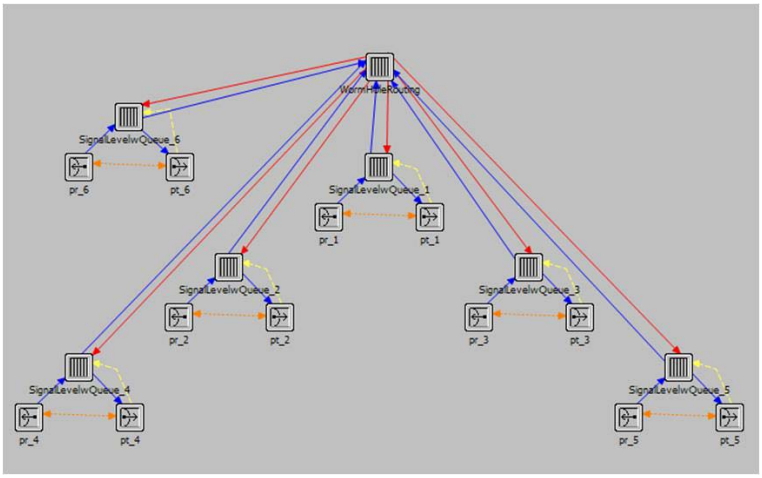

Figure 3. SpW Wormhole Router node model.

WormholeRouting Process

This WormholeRouting process implements the root process of the wormhole router. Its primary role is to create a child process for each of the interfaces identified in the node model. Each child process is an identical state machine process model that can independently be in a different state. For example, if Interface Number One is receiving a data flow and transferring the data to Interface Number Two, each of the interfaces must manage their flow with the opposing interface independent of each other. Additional data flows occurring on other interfaces must function independent of the initial data flow. The master-to-child process approach is a good fit to model the wormhole router, and it is illustrated in Figure 4.

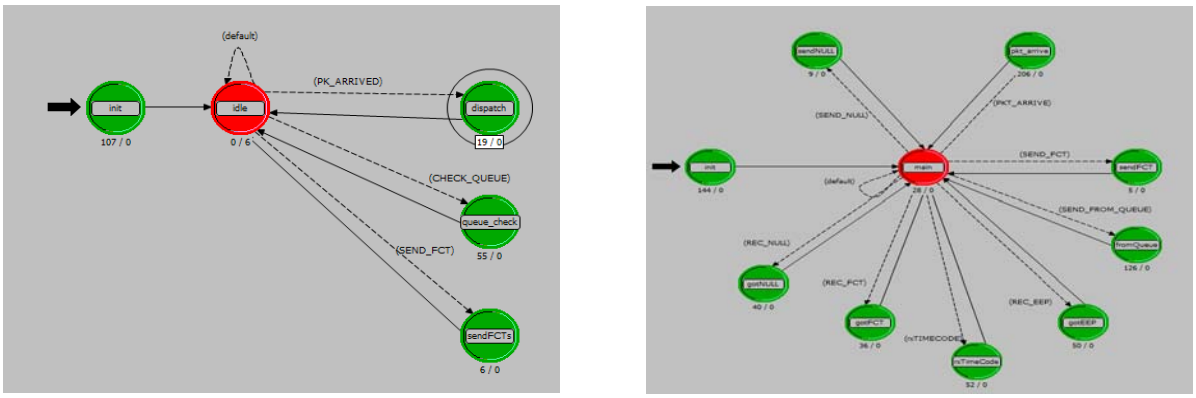

Figure 4. SpW Wormhole Router master process (left) and slave process (right).

The master process manages the mapping of the two child processes needed to support a specific data flow. The master process receives SpW characters from the various router interfaces and directs them to the appropriate child process. Each child process manages the flow control and SpW character buffering with each opposing interface as provided by a host node, a broadcast server, or another router interface. Additionally, the SpW router distributes a Time Code received on one interface to all other connected interfaces. The child process that receives a Time Code duplicates and then sends a copy Time Code to every other interface.

The SpW router's configuration can either use a specified Routing Table or Discover its route selection. In the case of the Discover configuration, the router model discovers the nodes connected to each of its interfaces. The discovered nodes are listed in a routing table and are mapped to the appropriate interface. When the router receives a character with a target destination, it looks up the destination in the routing table and directs the character to the appropriate transmit interface. This discovery approach works for routes that are limited to a single hop.

The other approach is to read a preconfigured routing table at initialization. In this case, no route discovery is performed, and the Excel formatted routing table is read and stored in memory by the master process model. Each slave process model accesses the appropriate row of routing configuration that maps the intended transmit interface for a destination host.

As with the SpW host node model, the SpW router model does not continuously stream Null characters as a real SpW device would. An abstraction is necessary in the models to avoid the excessive computation resources to source and sink a constant Null character stream on all interface pairs. The models faithfully produce Null characters in the setup phase of the SpW link. Once the link is set up, no Nulls are sourced; however, the Time Code character jitter that results from the minor delay variation if Null characters were being streamed is modeled as a random delay.

SpW Broadcast Server Model

Several approaches to broadcast were considered in this development. Fully serial broadcast occurs when a single node sends out the broadcast message as a unicast message to each target node. This approach lacks network efficiency and is too slow to completely distribute the time broadcast. A fully parallel broadcast is where a router would receive a message on a single interface and simultaneously transmit it to all of its other interfaces. The SpW standard does not support this, and implementation would require substantial modification to the standard. Neither a fully serial broadcast nor fully parallel broadcast is used. The broadcast servers use several techniques at the protocol level to guarantee that no loops, infinite broadcast storms, or spurious re-broadcasts occur. This broadcast solution has been fully developed in VHDL and tested in actual custom hardware.

The broadcast mechanism used for the SpW Broadcast Server (SpWBS) includes several stages:

Local-to-Server Stage - A SpWBS receives a local-to-server type packet containing the broadcast message.

Server-to-Server Stage - The initiating SpWBS sends a server-to-server type packet containing the broadcast message to every other enabled SpWBS in the network.

Server-to-Local Stage - Once a SpWBS receives a Server-to-Server stage packet, or the initiating SpWBS finishes the Server-to-Server stage transmission, it sends Local-to-Server type packets with the broadcast message to every enabled and connected local port.

This broadcast approach has efficiencies in that it partially distributes bandwidth utilization across the network and obtains parallelization of the Server-to-Local stage of broadcast. The approach requires that every router with nodes receiving broadcast messages have an attached SpWBS and it requires an additional header byte to distinguish between Local-to-Server, Server-to-Local, and Server-to-Server type messages. The Server-to-Local header byte can be removed if the system can guarantee that the replacement byte will never have the same value as a Local-to-Server header byte. As in Reference 3, all Server-to-Local packets use physical addresses to avoid the latency of requiring the router to check its routing table.

The SpWBS model uses several of the process models used for the SpW host model and is shown in Figure 5. The SpWBS employs a hybrid broadcast approach derived from work described in Reference 3. The implementation of this approach has two main configurable aspects: which local ports will receive broadcasts, and a list of the logical addresses of all *other* broadcast servers in the network, of which there is one per router. This broadcast approach was modified with the primary goal of distributing system time across a SpW network. This broadcast approach is fully compatible with existing SpW hardware. The approach creates a "broadcast server" that appears as an additional SpW endpoint on every router in the network. A packet intended for broadcast is transmitted to the local "broadcast server," which then forwards the packet to all other broadcast servers in the network. Once this is completed, every broadcast server in the network will forward the packet to the appropriate local ports on their respective routers.

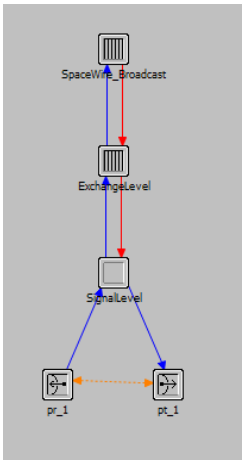

Figure 5. SpW Broadcast Server node model.

SpaceWire_Broadcast Process Model

This process model serves the primary broadcast function within the SpW model. The basic functions are the reception of a broadcast packet and the duplication and transmission of the packet. When this process receives the various NChars representing the broadcast packet, the NChars are reassembled back into a complete packet before duplication of the packet. When the packet has been successfully reassembled, the SpWBS process duplicates the packet for each additional SpWBS that should receive the broadcast packet. The packets are queued in the order

of the SpWBS list and transmitted in a unicast fashion to each SpWBS. Upon completion, the process duplicates the broadcast packet for each local host node and transmits the packet to each local host in a series of unicast transmissions.

The configuration of the SpWBS node model requires that each SpWBS be configured with a list of other SpWBS that should receive the broadcast and a list of local hosts that should receive the broadcast.

SpW Link Model

A custom link model was created to represent a full duplex SpW link. The link model can be configured to represent any data rate. Additional attributes include link delay and link error models. In the case of our SpW model, the standard link delay model used is distance-based. For the use case presented in this report, the link error model is configured to not introduce errors in the link data flow; however, the error model can be configured to cause errors (e.g., random or burst). An image of the SpW link model attribute list is shown in Figure 6.

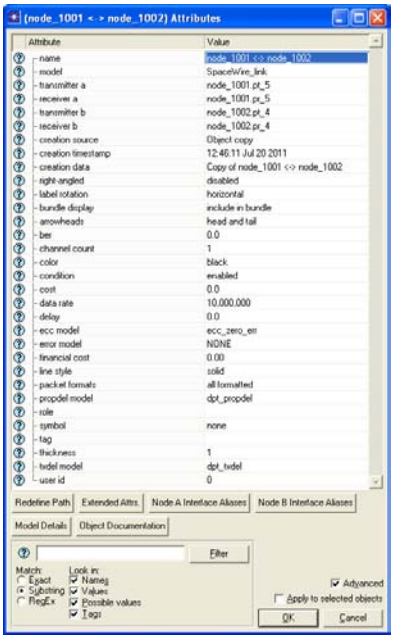

Figure 6. Attribute list of SpW link model.

SYSTEM TIME DISTRIBUTION PRECISION ANALYSIS WITH HIGH-FIDELITY SPACEWIRE MODEL

The standard SpW Time Code comprises the SpW ESC character followed by an eight-bit data character. The data character contains 6 bits of system time and 2 bits for control flags. A time-master node asserts a periodic "tick." With each "tick," the time-master node immediately sends out a Time Code with the 6-bit time field incremented before transmission [4]. This Time Code mechanism is limited to a six-bit resolution and increments each network device's internal time counter from the current Time Code value to the next. The counter rolls from its maximum value of 63 to zero because of its six-bit field size limit. The purpose of the counter is to prevent endless retransmission of the time around a looped network and not necessarily to carry a time value.

This time synchronization approach accomplishes the distribution of system time. This is done by sending a system-wide broadcast, containing what the system time will be at the next Time Code "tick." This broadcast is sent out in an efficient, semi-parallel manner using broadcast servers. The SpW network node that created the system time broadcast message waits a predetermined time that is sufficient for the time message broadcast to propagate throughout the network. After waiting, the node transmits a Time Code tick indicating to the network that the time described in the previous time message is now current. Thus, the various network applications have access to an unambiguous system time.

Unambiguous system time is derived from a central time server that employs broadcast to distribute time updates. System time is a 32-bit integer representation, and it is broadcast to all nodes in the network. When the timekeeper endpoints receive a SpW Time Code, the previously mentioned received system time message is output as the current time after having been validated by combinational logic. The time endpoint evaluates whether it is synchronized with the rest of the SpW network with every received SpW Time Code "tick." If the time endpoint believes itself to be synchronized with the rest of the time endpoints in the SpW network, it considers itself to be "locked" and asserts a corresponding signal.

The time endpoint determines if it is "locked" in the following way: After every "tick," the expected value of the next system time message is calculated. The calculated value is considered to be the value of the current system time message plus one. If the next received system time message matches the expected value, the endpoint concludes that it is synchronized with the rest of the network. If the next received time message does not match the expected value, or no system time message is received by the next "tick," then the endpoint assumes that a synchronization error has occurred, indicates that it is no longer "locked," and will simply increment its system time as a "best guess."

To demonstrate the time synchronization analysis capability, a SpW network comprised of 12 nodes, routers, and broadcast servers was created, as shown in Figure 1. The architecture, constructed in OPNET Modeler, uses the various custom-built SpW nodes and process modules that were added to go beyond OPNET's standard libraries. In this demonstration case, Node 10 is considered the time master and thus originates both the Time Codes and the system-time broadcast messages. Following the time synchronization mechanism, Node 10 will create a

broadcast message immediately following a Time Code transmission. The broadcast message will be transmitted to the broadcast server associated with Node 10 (i.e., Node 1010). This broadcast message contains the time that the next transmitted Time Code will clock into the various network slave nodes. Since Time Codes are not delayed by full application layer file transfers, the broadcast will not arrive at a slave node before the previously sent Time Code. However, there is no guarantee that the broadcast message will arrive at the slave nodes before the arrival of the following Time Code transmission. In these cases, where the following Time Code arrives at the slave node before the broadcast time message, the system is said to have lost synchronization "lock." The following section describes the analysis results of the network in Figure 1 for time synchronization precision.

RESULTS AND DISCUSSION

The network in Figure 1 with Node 10 producing both the Time Codes and the broadcast messages is assessed for time synchronization delay variation. In this analysis, we record the receipt of a broadcast message and the time at which the broadcast message's time value is clocked into the slave node's clock. The node's time is then compared with a global absolute time. The difference of the absolute time and node clock time is recorded and plotted in Figure 7 as a distribution of time differences and in Figure 8 as a probability density function (PDF). In Figure 7 the timing mechanism results begin at 10 seconds on the y-axis because the model performs initialization actions before 10 seconds. This time value was selected by the analyst and is not based on any SpW protocol specification.

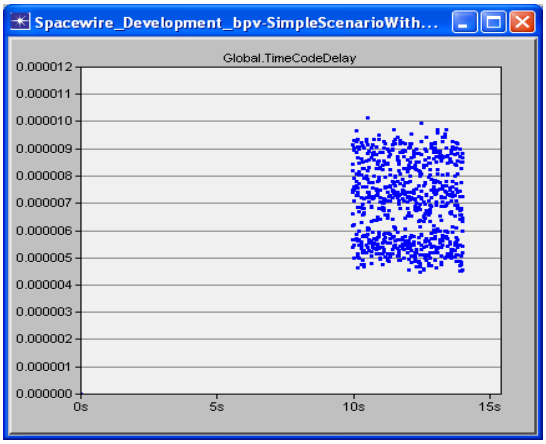

Figure 7. Distribution of time differences calculated as the difference between the absolute time and the node's clock as set by the SpW time distribution approach. The x-axis represents the simulation run time in seconds and the y-axis is the time difference in seconds.

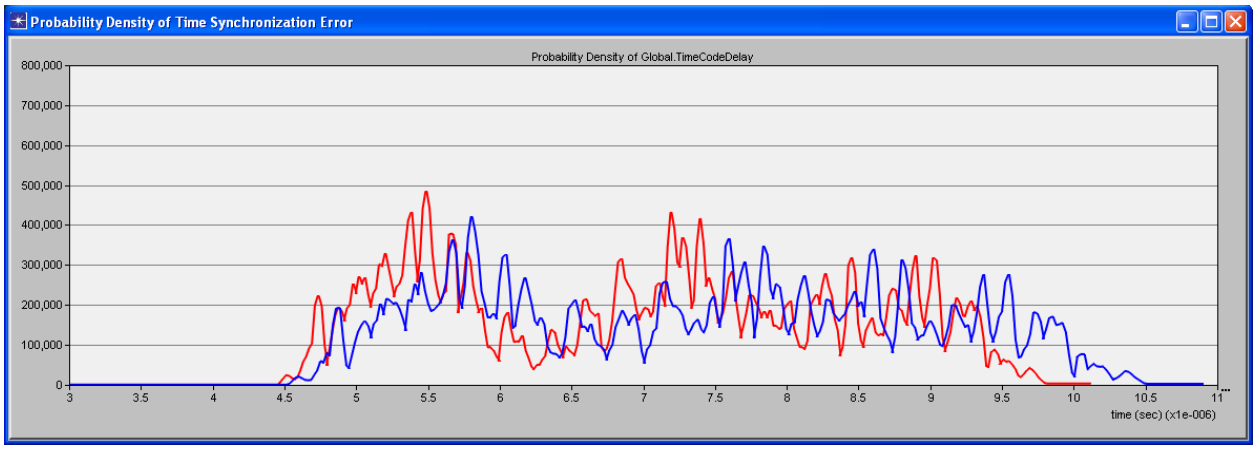

Figure 8. Resulting PDF of the time synchronization error when the network is lightly loaded (red trace) and heavily loaded (blue trace). The y-axis describes the relative likelihood for the time-precision value. The y-axis should be normalized by dividing by 50E6.

Figure 7 describes a time synchronization error averaging approximately 7.0 µsec. The plot graphs the time variation between each node's local clock and the system's actual reference time. Each point in the y-axis is the time variation for each node in the network versus the progressing simulation time. The plot in Figure 8 is the resulting PDF of the data shown in Figure 7 and has three regions centered at approximately 5.5 µsec., 7.3 µsec., and 9.0 µsec. Each region describes the variation in time synchronization based on the number of network hops necessary to forward the Time Code to a particular node. Each additional hop adds more variation and thus leads to more spreading of the plot as you move from left to right on the time axis in Figure 8. The variation between the lightly loaded network (red trace) and the heavily loaded network (blue trace) results from more NChars on the network. A network transmitting more NChars increases the probability that a Time Code will be delayed by the time to complete an NChar transmission. The variation is not significant since a NULL can cause a delay of up to an 8-bit transmission time whereas an NChar can cause a delay of up to a 10-bit transmission time. In the demonstration network, the SpW links operate at 10 Mbps. Also note that the Time Code period was 6 msec and the maximum application-layer file size was less than 60 Kbits and thus were easily within range so the network would not lose time synchronization lock. Selection of 6 msec and a file size of less than 60 Kbps is arbitrary; however, selecting a file size larger than what can be transmitted within the Time Code period will result in the network losing time synchronization lock. This will result if multiple Time Codes are received before the time broadcast being received. Time Codes are not delayed by the file being transmitted but time broadcasts are delayed by file transmissions. We elaborate on synchronization lock in the following section.

FUTURE WORK AND CONCLUSIONS

The time synchronization resolution of this approach is limited by the frequency of Time Code transmissions. The frequency of Time Code transmissions is limited by the requirement of sufficient time for the broadcast system time message to propagate throughout the network. It is believed possible to decouple the need for a one-to-one correlation of system time messages and Time Code transmissions to obtain an improved synchronization error. However, the theoretical upper limit of system time synchronization precision is limited by the latency and jitter inherent in SpW Time Code function. Time Code enhancement techniques [5,6] could be incorporated into our time distribution approach to improve time synchronization.

The Time Code enhancement techniques described by Reference 5 can be incorporated in the SpW models and evaluated to determine the level of improved time synchronization performance. If timing improvements are identified, the proposed improvements described in Reference 5 can be implemented to provide high-accuracy, low-jitter delivery of Time Codes across the network. An improved system would maintain the current time distribution approach as is, thus still guaranteeing time to be provided by a central server. In this case local clocks/calculations are only used for verification of the received time. The additional introduced clock element would reside in the timekeeper and increment from a locally running clock, but be synchronized against a field in the system time message at the arrival of the higher-accuracy Time Code. Given the highly deterministic latency and ultra-low jitter of the high-accuracy time code ticks proposed in References 5 and 6, we could potentially pre-calculate what the time value should be when the tick arrives at every timekeeper, provide that information in the system time message, and synchronize against it in the timekeeper using high-accuracy Time Code delivery methods. Therefore, fine-time will be provided locally, using the system-time broadcast for verification and synchronization. This approach can be modeled to determine expected performance.

Additional features will be incorporated into the OPNET Models to expand the representation of the SpW protocol and the nodes. Specifically, a model of Remote Memory Access Protocol (RMAP) for SpW will be developed. RMAP provides a standard method of reading and writing to registers and memory across a SpW network. This will further the analysis capability of application performance.

Additionally, SNL has developed a Live/Virtual/Constructive capability [7] that combines real devices, emulated devices, and simulated devices in a single hybrid experiment. Use cases in the SpW development activities have been identified that will benefit from merging the SpW models into hybrid experiments to assess satellite network ideas at various stages of the development. This approach is expected to support assessing the behavior of actual hardware before the availability of complete system hardware.

A viable system distribution approach has been demonstrated that can be employed without modification to the SpW standard. The time distribution approach has been modeled in a high-fidelity simulator and analysis has identified the range of time synchronization for various SpW network architectures. The broadcast solution has been fully developed in VHDL and tested in actual hardware. Further integration and testing in actual hardware continues in this development activity.

REFERENCES

[1] European Space Agency, ECSS-E-ST-50-12C, SpaceWire - Links, nodes, routers, and networks, *ESA-ESTEC Requirements & Standards Division*, July 31, 2008.

[2] OPNET Technologies, Inc., www.opnet.com.

[3] A. Roberts, S. G. Dykes, R. Klar, and C. C. Mangels, A Link-Layer Broadcast Service for SpaceWire Networks, *Aerospace Conference, 2007 IEEE*, March 2007, pp. 1-10.

[4] S. Parkes, The Operation and Uses of the SpaceWire Time Code, *ISWS International SpaceWire Seminar 2003*, November 2003.

[5] B. Cook and P. Walker, Time Code Enhancements for SpaceWire, *2006 MAPLD International Conference*, Washington, D.C., September 25, 2006.

[6] B. M. Cook, Reducing SpaceWire Time Code Jitter, Internet: www.4links.co.uk/bibliography/Reducing-Time Code-Jitter-on-SpaceWire.pdf, October 27, 2003 [accessed July 4, 2011].

[7] B. Van Leeuwen, V. Urias, J. Eldridge, C. Villamarin, and R. Olsberg, Performing cyber security analysis using a live, virtual, and constructive (LVC) testbed, *Military Communications Conference, 2010 - MILCOM 2010*, pp. 1806-1811, October 31, 2010–November 3, 2010.

APPENDIX

This appendix contains notes intended to assist the user of the SpaceWire (SpW) models.

- Each state machine process model should be examined to be certain constant values are set appropriately. The constant values are set in the INIT stage. Examples include:
    - Null and normal-character (NChar) character sizes,
    - Link data rate, and
    - Maximum flow control tokens (FCTs) allowed

- For the SpaceWire_Signal_level_Proc_wQueue process, check the “recv” state to be certain the Time Codes are delayed by the correct distribution.
    - // TimeCode Pkts are sent to transmitter immediately
    - // Since no regular pkts being sent expect NULL pkts being sent
    - op_pk_send_delayed(pkptr, TXLayer_sendSTRM, op_dist_outcome (null_dist));

- In the SpW Broadcast server (SpWBS) the size of the SpWBS and localBS list/table is limited to 20 nodes per two integer tables with 20 places. See state variables (i.e., SpWBS_Table[20]). Also see the initialize code in INIT state.

- For the SpW router, routing tables are created in an EXCEL spreadsheet as follows:
    - Save as a .csv file
    - Row 1 represents Router-1 routing table
        - Column A indicates interface number to send data to Node 1
        - Column B indicates interface number to send data to Node 2

- For the SpW router, note configuration how routing table is accessed:
    - A small network uses neighbor discovery – no routers chained together.
    - Networks with multiple router layers use imported routing table

- For host nodes that originate state of health (SOH) messages, be certain not to send SOH packets (or any packets) to self. For example, if Node 1 is a Telemetry Node that receives all SOH packets, do not auto config to send to "Destination =1”.

- If an end of error character (EEP) is generated and part of the simulation, check results for expected operation. Testing of this function is limited.

DISTRIBUTION

1	MS0503	James W. Daniels	5337
1	MS0503	Dominic A. Perea	5337
1	MS0503	Mythi M. To	5337
1	MS0503	Christopher K. Wojahn	5337
1	MS0503	Donald E. Tolsch	5339
1	MS0513	Richard D. Hunt	5336
1	MS0621	Dallas Wiener	5632
1	MS0661	Daniel E. Gallegos	2623
1	MS0661	Mark W. Learn	2623
1	MS0661	Aaron D. Niese	2623
1	MS0671	Jennifer M. Depoy	5628
1	MS0672	Brian P. Van Leeuwen	5628
1	MS0860	Matthew W. K. Brown	2622
1	MS0860	Jacob E. Leemaster	2622
1	MS0971	Ethan L. Blansett	5735
1	MS0980	Jay F. Jakubczak	5710
1	MS0980	Matt P. Napier	5571
1	MS0982	Jaime Gomez	5732
1	MS0982	Dan J. Kral	5732
1	MS0986	David M. Bullington	2664
1	MS0986	Jonathon W. Donaldson	2664
1	MS0986	Justin W. Enderle	2664
1	MS0986	David Heine	2664
1	MS0986	Jeffrey L. Kalb	2664
1	MS0986	David S. Lee	2664
1	MS0986	J. (Heidi) Ruffner	2664
1	MS1027	John M. Eldridge	5632
1	MS0359	D. Chavez, LDRD Office	1911
1	MS0899	RIM-Reports Management	9532 *(electronic copy)*


Sandia National Laboratories