SpaceWire-to-GigabitEther
User Guide

Contents of the document

Comments on the SpaceWire-to-GigabitEther device and this user guide are welcome. Please commit your idea to improve the device and the documentation. Feel free to contact to yuasa@juno.phys.s.u-tokyo.ac.jp.

Overview

SpaceWire-to-GigabitEther Converter

This device provides a easy-to-use connection to the SpaceWire network via GigabitEthernet. A user can send/receive SpaceWire packets to/from a user program running on an ordinary PC. The class library written in C++ is also available for user programs which run on the PC. Using the library, users can perform Remote Memory Access Protocol (RMAP) to RMAP Target nodes connected to the converter through the SpaceWire network. This device is not flight qualified, but originally intended for SpaceWire/RMAP-based data acquisition system of scientific experiments and ground tests of flight modules which use SpaceWire interfaces.

The SpaceWire-to-GigabitEther device utilizes ZestET1 board. SpaceWire Open IP and Host I/F are implemented on the UserFPGA of the board. Host I/F controls the onboard TCP/IP stack (GigEx from Orange Tree; http://www.orangetreetech.com/), and operates as a TCP server which listens/accepts a connection from a client PC on which user program runs. When disconnected, the converter goes back to the listening state and waits for a new connection.

SpaceWire I/F operates at 10.4 MHz by default. The link speed is variable, and the maximum Tx clock is 125 MHz which is equivalent to the original clock of the FPGA used in the device. The converter can send/receive any size of packet with EEP/EOP. It is also able to emit/receive TimeCode.

SpaceWire
(MDM-9 or Dsub-9 for low cost applications)

Power (+5V)

GigabitEther

35mm

75mm

50mm
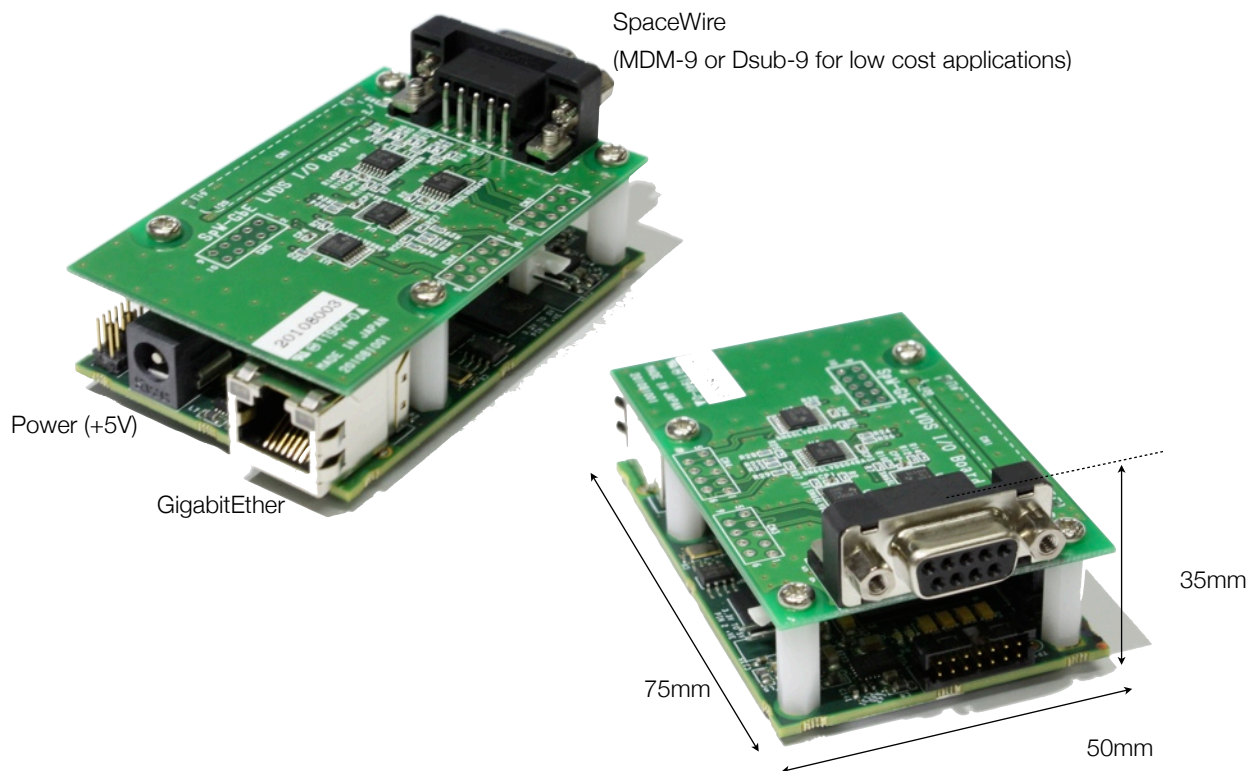
FPGA Block Diagram

This device implements SpaceWire IP core and SpaceWire to TCP/IP converter on Xilinx FPGA (XC3S1400A). The figure below shows the block diagram of the FPGA logic. The Open SpaceWire IP core which is distributed by Japan SpaceWire Users Group is used.
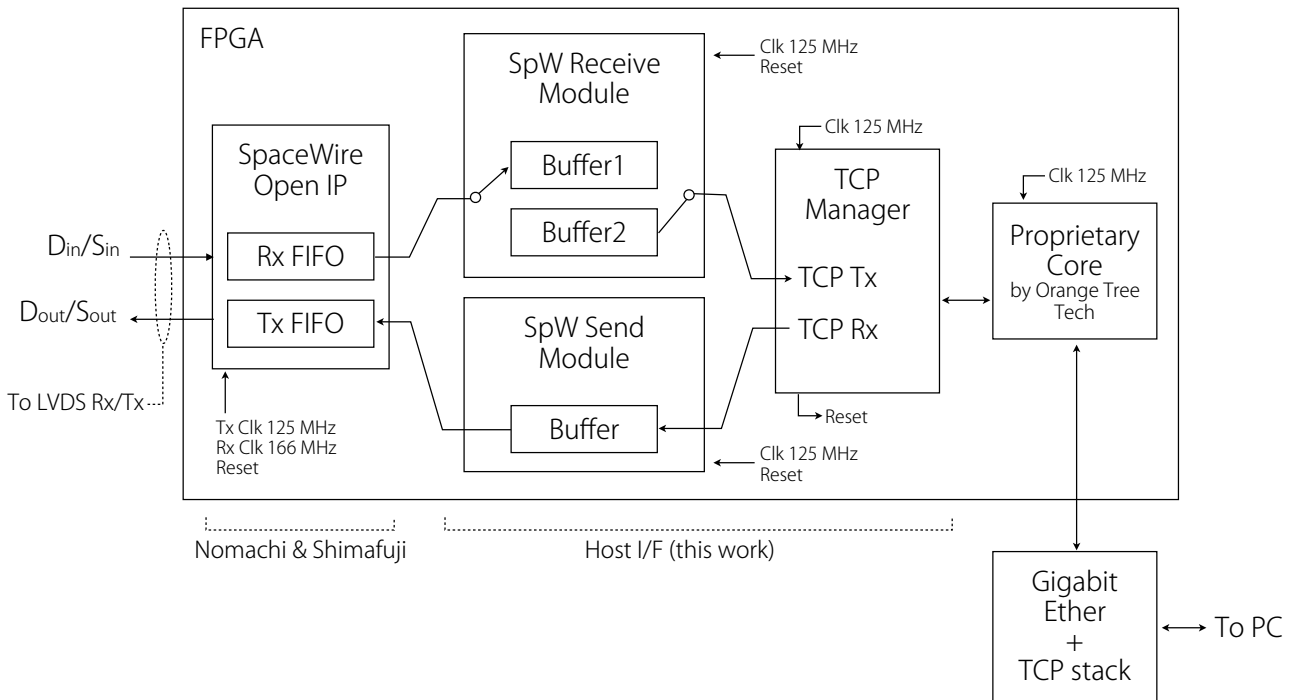
Encapsulation of SpaceWire packets

To transfer SpaceWire packets over a TCP/IP connection, the device encapsulates the packets with simple wrapper protocol. Details are described in "Implementation" chapter.

Requirements

The SpaceWire-to-GigabitEther device can be used from Linux/Mac/Windows(Cygwin) if they have Ethernet interface. To achieve fast data transfer speed, >1GHz CPU and GigabitEther are recommended. SpaceWire/RMAP Library is written in C++, therefore a C++ compiler for example GNU GCC greater than 4.0 is necessary to build user programs using the library.

The device accepts DC+5V. A/C power supply is shipped with the device.

Update information and Example programs

See http://www-utheal.phys.s.u-tokyo.ac.jp/~yuasa/spc/SpaceWire-to-GigabitEther/ for updates of FPGA IP Core, User Guide pdf, and SpaceWire/RMAP Library. The web site also provides example programs for SpaceWire-to-GigabitEther.

Usage

After some initial settings, the SpaceWire-to-GigabitEther device can be used as an external SpaceWire interface from an ordinary PC without any driver software. Communication between the device and the PC (or a user program on it) is performed over TCP/IP on GigabitEther. SpaceWire packets are encapsulated using a simple wrapper protocol, and transferred over the TCP/IP link. User program can send/receive SpaceWire packets via the virtual SpaceWire interface class provided in the SpaceWire/RMAP Library. Also, RMAP accesses can be done very easily using the software RMAP stack in the library. In the following sections, practical usage of the device is presented.

To obtain SpaceWire/RMAP Library, visit http://www-utheal.phys.s.u-tokyo.ac.jp/~yuasa/spc/SpaceWire-to-GigabitEther/ .

Change IP address

Before using the SpaceWire-to-GigabitEther device, users should change the network configuration of the device such as the IP address appropriately. Because the device uses Orange Tree Technology's ZestET1 board as its main board, users can follow Users Guide provided by Orange Tree to change the configuration. The operation can be performed via the web interface implemented on ZestET1 using the web browser.

Since the default IP address of the device is 192.168.1.100, a user may be need to change the IP address his/her PC. Typical configuration for PC is:

      IP address : 192.168.1.101   Subnet : 255.255.255.0

      Gateway or Router : 192.168.1.1 (not important at this moment).

Connect Ethernet cable between your PC and the SpaceWire-to-GigabitEther device, then access 192.168.1.100 from the web browser. Note that Safari (MacOS's default browser) is not supported when changing the network configuration, and therefore users need to use other browsers such as Camino (http://caminobrowser.org/. Then, fill the form with IP address, subnet mask, and gateway. By clicking update button, the filled information is used and the network device of the board is reset. Hereafter, the PC cannot communicate with the SpaceWire-to-GigabitEther device anymore except for the case that the same subnet information is valid even after the change. The user should change the PC's network configuration again so that it can reach the SpaceWire-to-GigabitEther device that has the new IP configuration.

Send/Receive SpaceWire packets

After power on, SpaceWire interface of the SpaceWire-to-GigabitEther device automatically tries to open SpaceWire link. When a user program connects to the device, it can send and receive packets using the encapsulation protocol shown in the following chapter. Although users can implement the protocol by themselves, it may be easier to use the implementation provided in SpaceWire/RMAP Library (SpaceWireIFOverIPClient.cc). Typical usage is like below.

```cpp
vector<unsigned char> sdata;//send data
vector<unsigned char> rdata;//received data

SpaceWireIF* spacewireif=new SpaceWireIFOverIPClient("133.11.165.60",10030);
        //IP address of the SpaceWire-to-GigabitEther device and the IP port number should be provided.
spacewireif->initialize();
spacewireif->open();

//prepare send data
...push the packet content into the 'sdata' vector...

//Example : send packet
spacewireif->send(&sdata);

//Example : receive packet
spacewireif->receive(&rdata);

//Example : send TimeCode
unsigned char timecode=0x07;
spacewireif->sendTimeCode(timecode);
```

By using the software RMAP protocol stack in the library, users can perform RMAP read/write accesses easily. Detailed usage of the SpaceWire/RMAP Library is described in its documentation ("SpaceWire/RMAP Library" document which is found in SpaceWire Wiki or Shimafuji's web site).

Example programs provided with the device show practical usages of SpaceWireIFOverIPClient class and underlying implementation class (SpaceWireSSDTPModule.cc), and raw communication with the device.

RMAP Read/Write

Users can perform RMAP access to RMAP targets by invoking the RMAP initiator's read/write methods. In SpaceWire/ RMAP Library, there are two main modules which realize RMAP transaction; RMAPEngine and RMAPSocket. The RMAPEngine class handles RMAP transactions and controls SpaceWire interface by acting as multiplexer for multiple RMAP requests. The RMAPSocket class is the RMAP Initiator interface for user threads, and has methods for RMAP read/write. To create a new RMAPSocket instance, users need to provide the destination information, RMAPDestination class, to the RMAPEngine. This class holds various information related to the RMAP target to which the RMAPSocket is opened, for example Logical Address, Path Address, Destination Key, and so on. Data are stored in std::vector so that users do not have to worry about the memory management and buffer over flow.

The standard way of using these classes are as follows.
(1) Open SpaceWire interface (SpaceWireIFOverIPClient).
(2) Instantiate and start RMAPEngine.
(3) Prepare RMAPDestination.
(4) Create RMAPSocket instance providing the RMAPDestination information to RMAPEngine.
(5) Invoke read() or write().
The code below presents sample code of these procedures.

```cpp
vector<unsigned char> sdata;//send data
vector<unsigned char> rdata;//received data

//(1) open SpaceWire interface
SpaceWireIF* spacewireif=new SpaceWireIFOverIPClient("133.11.165.60",10030);
spacewireif->initialize();
spacewireif->open();

//Because SpaceWire DIO sometimes does not accept 125 MHz Tx clock,
//here Tx clock is set at 125/(1+1) = 62.5 MHz.
((SpaceWireIFOverIPClient*)spacewireif)->setTxDivCount(1);

//(2) Instantiate and start RMAPEngine
RMAPEngine* rmapengine=new RMAPEngine(spacewireif);
rmapengine->start();

//(3) Prepare RMAPDestination
//in this example, Shimafuji's SpaceWire DIO is used.
RMAPDestination rmapdestination_to_dio;

rmapdestination_to_dio.setDraftFCRC();
rmapdestination_to_dio.setDestinationLogicalAddress(0x30);

rmapdestination_to_dio.setSourceLogicalAddress(0xfe);
rmapdestination_to_dio.setDestinationKey(0x02);

rmapdestination_to_dio.dump();

//(4) Creates RMAPSocket instance
RMAPSocket* rmapsocket_to_dio=rmapengine->openRMAPSocketTo(rmapdestination_to_dio);

//address and length
unsigned int address=0x00000000;
unsigned int length=0x10;

//(5-1) RMAP Read
rdata=*(rmapsocket_to_dio->read(address,length));


//(5-2) RMAP Write
for(unsigned int i=0;i<length;i++){
        sdata.push_back((unsigned char)i%0x100);
}
rmapsocket_to_dio->write(address,&sdata);
```

RMAP to multiple targets and Path Addressing

In the concept of SpaceWire/RMAP Library, an RMAP access is performed via RMAP Initiator (RMAP Socket). One RMAPSocket is associated with an RMAP target node. Therefore, when users want to access other nodes, RMAPSocket instances for the nodes should be created. The code below shows how to create multiple RMAPSocket together with an example of Path Addressing to/from an RMAP target.

```cpp
//RMAPSocket to Shimafuji's SpaceWire DIO
RMAPDestination rmapdestination_to_dio;
rmapdestination_to_dio.setDraftFCRC();
rmapdestination_to_dio.setDestinationLogicalAddress(0x30);
rmapdestination_to_dio.setSourceLogicalAddress(0xfe);
rmapdestination_to_dio.setDestinationKey(0x02);
RMAPSocket* rmapsocket_to_dio=rmapengine->openRMAPSocketTo(rmapdestination_to_dio);

//RMAPSocket to Shimafuji's SpaceWire FlashADC
//Assume that this board is accessed using Path Addressing
RMAPDestination rmapdestination_to_fadc;
vector<unsigned char> path_to_fadc;
vector<unsigned char> path_from_fadc;
path_to_fadc.push_back(0x03);
path_to_fadc.push_back(0x01);
path_from_fadc.push_back(0x02);
path_from_fadc.push_back(0x04);
rmapdestination_to_fadc.setDraftFCRC();
rmapdestination_to_fadc.setDestinationLogicalAddress(0x40);
rmapdestination_to_fadc.setDestinationPathAddress(path_to_fadc); //set path address TO the target FROM SpaceWire-to-GigabitEther
rmapdestination_to_fadc.setSourceLogicalAddress(0xfe);
rmapdestination_to_fadc.setSourcePathAddress(path_from_fadc); //set path address FROM the target TO SpaceWire-to-GigabitEther
rmapdestination_to_fadc.setDestinationKey(0x02);
RMAPSocket* rmapsocket_to_fadc=rmapengine->openRMAPSocketTo(rmapdestination_to_fadc);

//RMAP Accesses
rdata=*(rmapsocket_to_dio->read(address,length));
rdata=*(rmapsocket_to_fadc->read(address,length));

rmapsocket_to_fadc->write(address,&sdata);
rmapsocket_to_dio->write(address,&sdata);
```

RMAP from multiple threads

SpaceWire/RMAP Library includes Thread Library which provides Java-like easy-to-use thread class. Users can inherit the Thread class to create thread class, and can start the thread just by calling the `start()` method of the child class. User - defined thread routine can be written inside `void run(){}`.

RMAPEngine can handle multiple RMAP requests from different threads, and therefore simultaneous concurrent RMAP transactions can be performed. As described later, this increases the RMAP data transfer dramatically, and therefore it is recommended to implement RMAP data transfer function as thread, and execute them in parallel.

See example program main_RMAPspeedtest.cc for how to use multiple threads to perform concurrent RMAP transactions.

Change Tx speed

The default SpaceWire Tx clock is 125/12=10.4 MHz because it is defined that the initial operating speed should be (10+/-1) MHz in the SpaceWire standard.

Users can change the Tx clock by invoking `SpaceWireIFOverIPClient::setTxDivCount(unsigned char txdivcount)`. The Tx clock is generated by dividing the original clock of 125 MHz with `(txdivcount+1)`. The lowest operating clock defined in the SpaceWire standard is 2 MHz, and therefore the maximum `txdivcount` should be 63. By setting `txdivcount` at 0, users can operate the device at the maximum link speed of the present implementation. If the node connected to the SpaceWire-to-GigabitEther device does not support for example 125 MHz link, they just fail to open connection, and no communication can be done from user program.

How to update IP core of the FPGA on the device

When updating the FPGA logic of the SpaceWire-to-GigabitEther device, use `CPanel.exe` contained in a CD-ROM provided from Orange Tree or distributor of SpaceWire-to-GigabitEther (probably Shimafuji Electric). The program runs on Windows or Wine on MacOSX (see "How to use CPanel.exe on Mac" below). It provides users ways of programing FPGA or EEPROM connected to it. A user should prepare a .bit file for XC3S1400A, and select it from the program, and then click

"FPGA" or "Flash" for direct configuration of FPGA or EEPROM, respectively. The procedure for updating the FPGA logic is shown below.

1. Open CPanel.exe (see below if Mac is being used).
2. Connect the SpaceWire-to-GigabitEther device to the LAN to which the PC is also connected (IP address and subnet mask should be set properly to find the device on the same local network).
3. Click "Refresh" button in the CPanel window to find the device.
4. Select the device in the list by clicking it.
5. Select the new .bit file, which should be programed to the UserFPGA of the device, by clicking "Browse" button.
6. Click "Flash" to program the bit file to the EEPROM on the device. It will take a few minutes to complete the process.
7. Reboot the device by plug out an in the power supply after "Successfully completed" message has appeared.

Also see "FPGA IP core revision history" chapter.
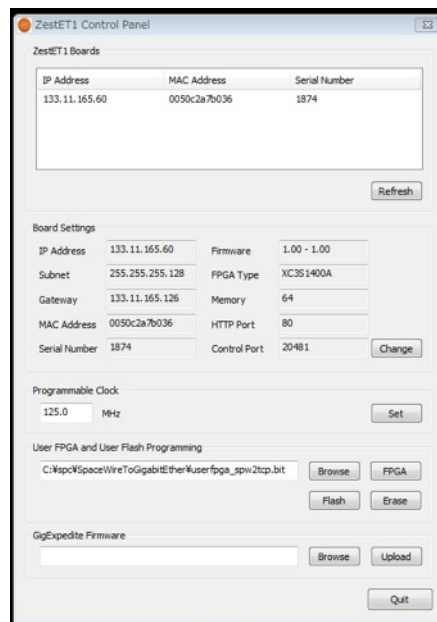
How to use CPanel.exe on Mac

Wine implements Win32API-compatible APIs. Windows programs can run on Wine. Using Wine, it is possible to execute CPanel.exe on MacOS X.

Download the source of Wine from http://www.winehq.org/download/, and compile and install it. After the installation, execute the line below in Terminal.app. CPanel.exe should have been copied from the CD-ROM of ZestET1. The screenshot below shows the CPanel.exe window running on Mac.

```
(on Terminal.app, execute below)
> wine CPanel.exe
```

Examples

Example programs of SpaceWire-to-GigabitEther can be obtained from http://www-utheal.phys.s.u-tokyo.ac.jp/~yuasa/spc/ SpaceWire-to-GigabitEther/. The programs are intended to show how to use this device in your application.

Install SpaceWire/RMAP Library

The example programs use SpaceWire/RMAP Library. Visit SpaceWire Wiki (http://www.astro.isas.ac.jp/SpaceWire/wiki/) or Shimafuji Electric's web site, and download the source. The release of 2010-07-06 or later should be used.

Before compiling the library, please check that SSDTP2 protocol is selected in the source file "SpaceWireSSDTPModule.cc" by "`#define SSDTP2`". If not, undefine SSDTP1 and define SSDTP2. Correct lines should be like below.

```
(in SpaceWireSSDTPModule.cc)

#define SSDTP2
#undef SSDTP1
```

Move the extracted folder "SpaceWireRMAPLibrary" to the default installation folder /usr/local/. Then, type `make` in build/. If a user wants to place the library elsewhere, he/she should change the path defined in build/Makefile so as to specify the appropriate path to the library folder before compiling.

```
mv SpaceWireRMAPLibrary /usr/local
cd /usr/local/SpaceWireRMAPLibrary/build
make
```

When make completes, binaries and libraries are located in /usr/lcoal/SpaceWireRMAPLibrary/posix/. Move to the build of the example programs.

Build example programs

Extract the archive of the example programs. Move to build/ folder, and then execute `make`. If you have installed SpaceWire/ RMAP Library in different folder other than /usr/local/SpaceWireRMAPLibrary, please set the path in build/Makefile before `make`. Compiled binaries are located in examples/build folder.

```
cd SpaceWireToGigabitEther/examples/build
make
```

Execute example program

Before executing a program, connect the SpaceWire-to-GigabitEther device to a LAN to which your PC is also connected., and set IP address, subnet mask, and gateway address of the SpaceWire-to-GigabitEther device properly.

(1) main_SpaceWireIFOverIPClient

This program demonstrates the basic SpaceWire packet transfer functionality. First it sends a packet, and then receives. If a loop-back connector (see below) is attached to the device's SpaceWire port, the sent packet itself will be received immediately. After the receive, it emits TimeCode.

(2) main_loopbackspeedtest

This program measures the data transfer speed in SpaceWire layer with the loop-back configuration. Please prepare a loop-back connector before executing this program. After inputing the packet size and the number of iterations, this program sends and receives packets concurrently. When completing, the effective data transfer speed is dumped.

(3) main_RMAP

This program shows how to perform RMAP read and write using the SpaceWire-to-GigabitEther device. Note that the RMAPDestination setting used in the program, by default, assumes Shimafuji's SpaceWire Digital I/O as an RMAP target. If you use other kind of RMAP target, please modify the destination information setting in the code (e.g. logical address, path address, destination key and so on).

(4) main_RMAPspeedtest

    This program measures RMAP data transfer speed by reading data from mass memory on an RMAP target. Similar to main_RMAP, the program also assumes Shimafuji's SpaceWire Digital I/O as a target by default.

Implementation details

Encapsulating protocol (SSDTP2)

SpaceWire has no limitation of the length of the packet, and each SpaceWire packet is terminated using the end of packet character (EOP) or the error end of packet character (EEP). On the other hand, TCP/IP provides us a simple socket which transfers bytes as stream, and there is no delimiter to handle the end (or dividing point) of the data being transferred. Therefore an encapsulating protocol should be used to transfer SpaceWire packets over a TCP/IP socket. In SpaceWire-to-GigabitEther, a simple header-followed-by-size-and-data protocol is defined and used. The name of the protocol is SSDTP2[1]. In the following sections we describe how we transfer packets and control data using this protocol.

Basic structure of SSDTP2 packets

In SSDTP2, encapsulated data have the following structure.

<Flag 1byte> <Reserved 1byte> <Size 10bytes> <Cargo Size bytes>

Flag specifies type of the packet. Packet types are Data and Control. Data encapsulates SpaceWire packet, and Control contains information needed to control the connection of the SpaceWire-to-TCP/IP converters on both ends of the TCP/IP link (i.e. SpaceWire-to-GigabitEther and a user program on PC). Size contains the size of Cargo part. Cargo part can be data or codes which contains control information. Below, the encapsulation structures are described individually.

Data

SpaceWire packets terminated with EOP or EEP are encapsulated shown below.

Flag (Data)	Reserved (0x00)	Size[9]	Size[8]
Size[7]	Size[6]	Size[5]	Size[4]
Size[3]	Size[2]	Size[1]	Size[0]
Data[0]	Data[1]	Data[2]	Data[3]
...	Data[Size-1]		

Flag (Data) byte is set at 0x00 for EOP-terminated packet and 0x01 for EEP-terminated packet. Size part is fixed 10-byte length, and holds the size of Data part in radix 256. The size of Data part can be restored and set by using the code below. Data part contains <destination address> and <cargo> of SpaceWire packet, and not <end_of_packet> (see the standard ECSS-E-ST-50-12C).

```
/* Code to restore the size from the byte array.  */
/* buffer[] should contain the bytes shown above. */
unsigned int size=0;
for(unsigned int i=2;i<12;i++){
    size=size*0x100+buffer[i];
}

/* Code to set the size to the byte array. */
/* buffer[] should contain the bytes shown above. */
unsigned int size=PacketSize;
for(unsigned int i=11;i>1;i--){
    buffer[i]=size%0x100;
    size=size/0x100;
}
```

---

[1] SSDTP1 was used in SpaceWire-to-TCP/IP converter software running on SpaceCube1 computer. The protocol was updated to implement more functionalities when SpaceWire-to-GigabitEther device was developed.

Large data (segmentation)

When the size of a packet is too long to handle as a single packet, software or hardware logic may divide the packet into multiple segments. In such a case, although the encapsulated packet structure is the same as above, Flag is set at 0x02 to show that the data is segmented and has no end of packet character. Size part should contain the size of the segmented data. After a certain number of un-terminated segments, a terminated segment which has Flag of 0x00 (EEP) or 0x01 (EEP) will complete the whole packet data.

Note that this segmentation has nothing to do with the SpaceWire standard, and arises simply from the difficulty of handling unlimited length of packets in the encapsulating protocol and its implementation. For example when only a small amount of data buffer is available on a SpaceWire-to-TCP/IP converter logic, a SpaceWire packet whose size is larger than the buffer size can not be received without any segmentation. The simple segmentation shown here allows the logic to send a part of the large packet to the TCP/IP side when the buffer becomes full (for example) specifying that the encapsulated data is continued (i.e. not terminated by EOP/EEP). When EOP/EEP is received, the logic can complete sending the segmented data.

Controls

Control informations are TimeCode and RegisterAccess. SpaceWire TimeCode information is encapsulated as Control packet in SSDTP2. A user program can access registers in the SpaceWire-to-GigabitEther device. By RegisterAccess, it can change for example frequency of Tx, and get the number of packets transferred. In the following sections, individual encapsulations of Control informations are described.

Encapsulated TimeCode

SSDTP2 encapsulates SpaceWire TimeCode so as to allow user programs to emit or receive TimeCode using the SpaceWire-to-GigabitEther device. The encapsulated structure shown below is used to encapsulate TimeCode information. Flag is 0x30 when sending TimeCode from a user program to the SpaceWire network via the device, and 0x31 when TimeCode is received at the device from the SpaceWire network. Usually, a user program sends Flag=0x30 and receives Flag=0x31 to/from the device. When Flag=0x31 is received, the user program may perform TimeCode-related operation. Size[0] should be 0x02, and the remaining Size part (Size[1]-Size[9]) should be filled with 0x00. In TimeCode byte, LSB 6bits are used to store 6-bit TimeCode value (time counter value). MSB 2bits are reserved in the standard, and should be "00".

Flag (Send/Got TimeCode)	Reserved (0x00)	Size[9] (0x00)	Size[8] (0x00)
Size[7] (0x00)	Size[6] (0x00)	Size[5] (0x00)	Size[4] (0x00)
Size[3] (0x00)	Size[2] (0x00)	Size[1] (0x00)	Size[0] (0x02)
TimeCode	Reserved (0x00)		

When the SpaceWire-to-GigabitEther device receives this encapsulated TimeCode information from TCP/IP side (i.e. user program), it asserts tick_in of SpaceWire IP core providing 6-bit time_in value (TimeCode value) as soon as possible. When the device receives a TimeCode from SpaceWire side, it transfers the above encapsulated TimeCode information to TCP/IP size (i.e. user program). To know the arrival of TimeCode with as short latency as possible, a user program should always wait in receive method.

Changing SpaceWire link speed

The Tx link speed of the SpaceWire-to-GigabitEther device can be changed by sending a control packet like below. The "TxDiv count" specified in the packet is used to divide the original clock of 125 MHz to generate the Tx clock which is fed to SpaceWire IP Transmitter. Users can change the Tx speed more easily by invoking `setTxDivCount(unsigned int)` method of the SpaceWireIFOverIPClient class (see "Practical Usage" chapter).

Flag (Change Tx Speed)	Reserved (0x00)	Size[9] (0x00)	Size[8] (0x00)
Size[7] (0x00)	Size[6] (0x00)	Size[5] (0x00)	Size[4] (0x00)
Size[3] (0x00)	Size[2] (0x00)	Size[1] (0x00)	Size[0] (0x02)
TxDiv count	Reserved (0x00)		

IP port number and the number of SpaceWire port

SSDTP2 uses IP port number 10030 for SpaceWire port 1, 10031 for SpaceWire port 2, 10032 for SpaceWire port 3 and so on. Although at this moment (2010-07-05), only SpaceWire port 1is implemented on the SpaceWire-to-GigabitEther device, there are plenty amount of available logic in the FPGA (XC3S1400A), additional SpaceWire IPs can be implemented in future updates. The table below shows the FPGA logic utilization with the current SpaceWire-to-GigabitEther converter logic.

Device Utilization Summary				[-]
Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	1,765	22,528	7%	
Number of 4 input LUTs	2,202	22,528	9%	
Number of occupied Slices	1,937	11,264	17%	
Number of Slices containing only related logic	1,937	1,937	100%	
Number of Slices containing unrelated logic	0	1,937	0%	
Total Number of 4 input LUTs	2,486	22,528	11%	
Number used as logic	2,183			
Number used as a route-thru	284			
Number used as Shift registers	19			
Number of bonded IOBs	90	161	55%	
Number of ODDR2s used	81			
Number of BUFGMUXs	3	24	12%	
Number of DCMs	2	8	25%	
Number of RAMB16BWEs	15	32	46%	
Average Fanout of Non-Clock Nets	3.27			

Performance

Several test programs are provided with the SpaceWire-to-GigabitEther device to show the details of implementation and to allow users to measure the data transfer speed. This chapter presents measured data especially data transfer speed because fast data transfer is generally important in many applications of SpaceWire.

Measurement setup

In the measurements, MacBook Pro computer is used to execute test programs. The table below shows the spec of the computer.

CPU	Core 2 Duo 2.4GHz
OS	Mac OS X 10.6
RAM	4GB
Ethernet	Gigabit Ethernet

SpaceWire layer

The data transfer speed was measured in the loop-back configuration using the loop back cable program `main_loopbackspeedtest.cc` and a cable which connects Dout and Sout to Din and Sin respectively. When a packet is sent from the test program, it goes through the SpaceWire-to-GigabitEther device and immediately reaches the device again. The program has two concurrently running threads for sending and receiving packets. Users can change the size of packet and the number of iterations (the number of packets to be transferred). To make the measurement practical, like many user programs this program also uses the implementation of the SSDTP2 encapsulating protocol provided in SpaceWire/RMAP Library to send/receive packets (i.e. without any specific implementation intended for higher performance).
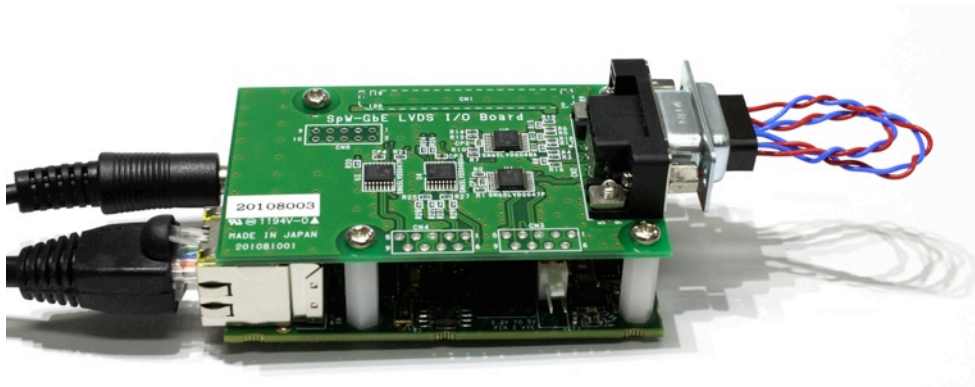
The figure below shows the result of the data transfer speed measured in different packet sizes and 125-MHz SpaceWire link. When the packet size is enough large, for example >100 bytes, the SpaceWire-to-GigabitEther and the SpaceWire/RMAP Library provides ~95 Mbps transfer speed to user programs. In smaller packet sizes (<50 bytes), the speed reduces significantly. This drop is expected because in such small packet sizes the overhead of TCP/IP layer cannot be neglected. Although this might be a problem in applications where the packet size is small and the timing of packet transfer is highly important, the maximum speed of ~95 Mbps (or efficiency of 95/125=75%) would be appealing for many users especially in scientific instrument developments.

Loop-back test at 125MHz SpaceWire link
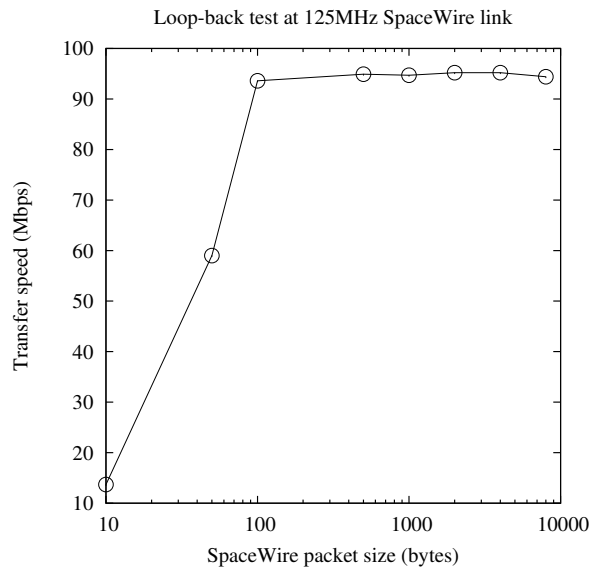
RMAP layer

Practically, a user program sends and receives data using RMAP stack of SpaceWire/RMAP Library. Basically, because RMAP involves transactions (i.e. Command and Reply) between the RMAP initiator and the RMAP target, physical round trip time between them and transaction control in the RMAP stack reduces the data transfer speed from the value achievable in the SpaceWire layer.

The test program main_RMAPspeedtest included in the example was used to read data from SDRAM of SpaceWire Digital I/O board. The link speeds were 62.5 MHz (SpaceWire-to-GigabitEther Tx) and 100 MHz (SpaceWire Digital I/O Tx). The read length and the number of concurrent RMAP read threads were surveyed in realistic range so that users understand how these parameters affect the result. To reduce the effect of the fluctuation of the performance of the TCP/IP layer, three measurements were done in one set of parameters, and then calculated averaged transfer speeds followed by the standard deviations.

The measurement were done in two different configurations. (1) First we directly connected the SpaceWire-to-GigabitEther device to the PC, (2) The PC and the SpaceWire-to-GigabitEther device was connected via IEEE802.11n (300Mbps) and the local area network (three GbE hubs). The figure below shows the measurement configurations.
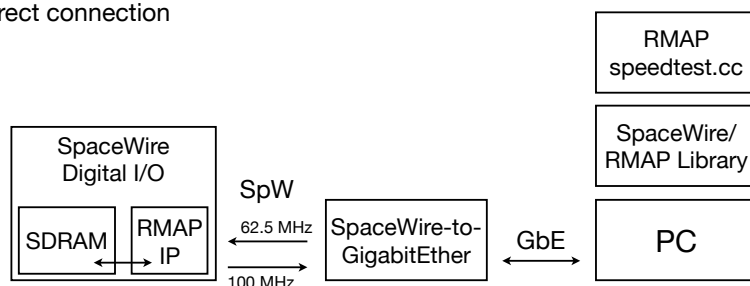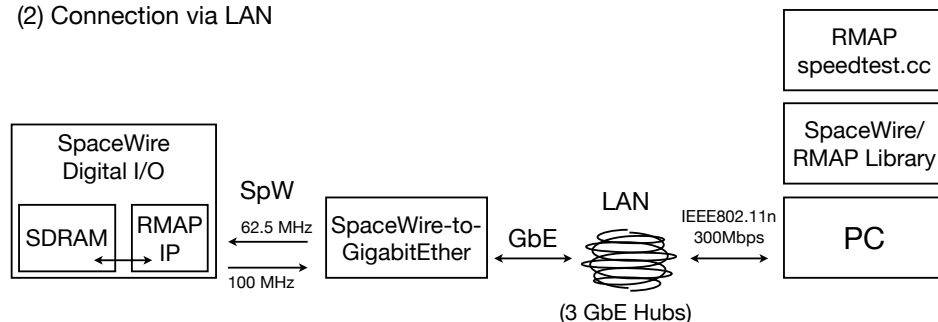
Result of the measurement is shown below. For the configuration (1),the transfer speeds are 38.2 Mbps and 16.2 Mbps in 8000- and 1000-byte cases in a single thread mode. In the dual thread mode, the values increase to 70.1 Mbps (184%) and 27.8 Mbps (172%) respectively. In (2) configuration, due to larger round trip time in GbE and TCP/IP layers, the transfer speed decreases from the direct connection case. From this result, it is recommended, if possible, to use the SpaceWire-to-GigabitEther device by directly connecting to a PC on which a user program runs.

As expected, larger read length provides faster transfer speed. In single thread RMAP Read, there is some idle time in the SpaceWire link, especially SpaceWire Digital I/O Tx side, and in the TCP/IP connection while the read-out program sends an RMAP Command. Similarly, When RMAP Reply is transferred from the target to the PC, the other side of SpaceWire link (Tx from the SpaceWire-to-GigabitEther device) is idle. By increasing the number of concurrent threads, multiple RMAP transactions are performed simultaneously, and the idle time of the links are reduced dramatically. In sufficiently long read length (>4k bytes) and sufficient concurrency (>3 threads), the transfer speed saturates at near the maximum limited by the SpaceWire exchange protocol which converts 8-bit character into 10 bits.
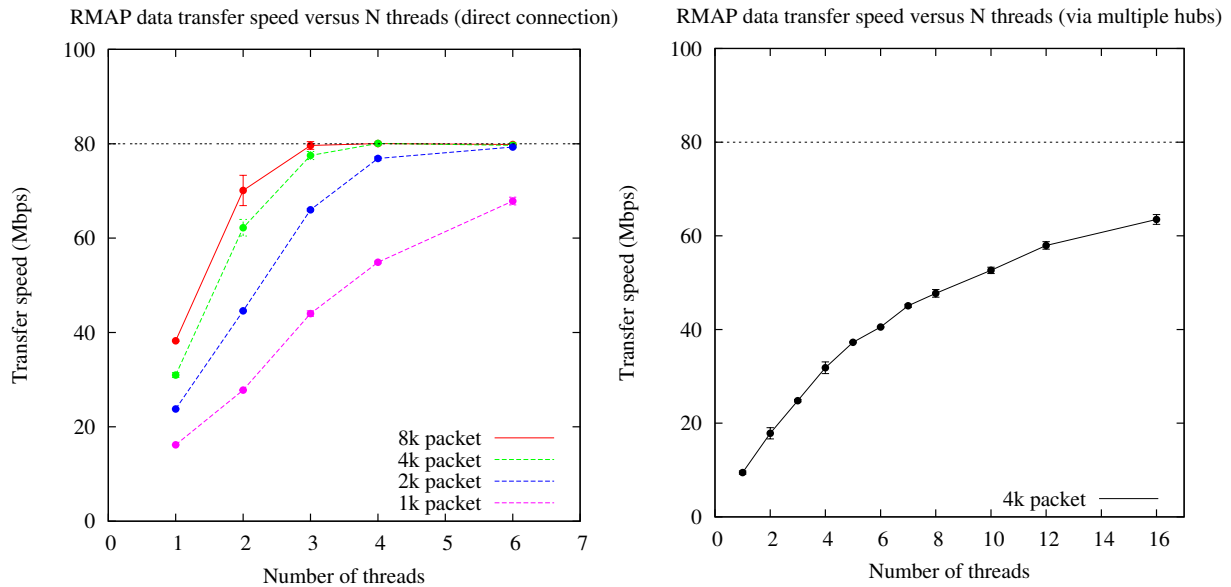
RMAP Read data transfer speed measured using the SpaceWire-to-GigabitEther device and SDRAM on Shimafuji's SpaceWire Digital I/O. Data points and errors show the average and the standard deviations of the data transfer speed measured three times. Abcissa is the number of threads which concurrently performs RMAP Read. *Left* : Results of the configuration (1). Red, green, blue, and purple data points show result with read lengths of 8000, 4000, 2000, and 1000 bytes respectively. The horizontal line is the theoretical limit of the data rate over a 100-MHz SpaceWire link. *Right* : RMAP Read data transfer speed measured interleaving three hubs between the PC and the SpaceWire-to-GigabitEther device. The read length of 4000 bytes was utilized in the measurement.

TimeCode jitter

The SpaceWire-to-GigabitEther device can emit TimeCode following Send TimeCode command from the user program on the PC. Even if the user program tries to emit TimeCode periodically, the actual timing should have some jitter, or statistical delay, due to uncertainties related to the software timer used in the user program and the TCP/IP stack.

To obtain practical estimation of the TimeCode jitter, a user program which emits TimeCode periodically at 64 Hz (one per 15.625ms) was developed. The TimeCode emitted from the SpaceWire-to-GigabitEther device is received by the SpaceWire Digital I/O board which was connected to the device as in the case of (1) of the above RMAP test. UserFPGA on the board measured the arrival time of individual TimeCodes, and the measured data was stored on the SDRAM of the board. The user program read the data via RMAP at a certain frequency, and calculated intervals of each pair of two TimeCodes.

The histogram below shows a result of the measurement. The mean of the time interval between two TimeCodes is ~15.7ms, and the bottom-to-bottom width of the distribution is about 0.3ms. The difference between the expected value (15.625ms) and the measured one (~15.7ms) is though to arise from a slight delay of the `usleep()` function which was used to implement 15.625-ms sleep in the user program.

To reduce the absolute delay from the expected frequency, one may be able to calibrate the parameter of the `usleep()` function; for example `usleep(15500)` might result better mean time duration than `usleep(15625)` in a certain environment. The optimum value depends on the performance of the CPU and the operating system of a computer, and therefore the calibration should be done system-by-system.

TimeCode interval measured on SpaceWire DIO

Mean=15.75ms
$1\sigma_{stddev}$=0.04ms
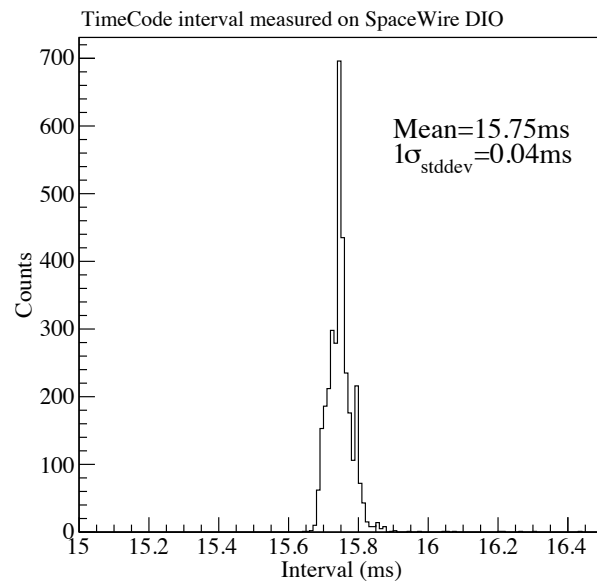
FPGA IP Core revision history

2010-07-18

The SpaceWire IP core used in the FPGA was updated to version 1.02. The bug in the link initialization is fixed in this version.

2010-07-06

The first release. This version of IP Core consists SpaceWire IP core ver 1.00, and therefore sometimes fails to connect link with a connected node. If users are using this version, please update to the latest release.