Software 35.42

Documentation

We now ship the End User License Agreement (EULA) in doc/EULA.pdf.

API

Windows 11

We now fully support Windows 11, Windows 11 native is found in the **Windows/Win11** directory

Python

Support for python 3.9 on Windows platforms, if you are running a later version of python on Windows platforms, please contact support@4links.co.uk and we will be able to turn around an update relatively quickly.

C

EtherSpaceLink_close_all this closes all connections to 4links devices. This may be useful when you want to reset an application and re-run it. Specifically, this may be useful for resetting a **LabView** application. Please note once called any C++/python contexts created before this call will fail afterwards.

C++

Virtual function **data_written** callback for monitoring data being sent. This is useful for tracking data sent by higher level functions (RMAP for example)

RMAP API (C++/Python)

The RMAP API has been extended to allow for RMAP responses being read in the next packet rather than waiting for packets arriving asynchronously. This is achieved by using the **set_sync_rx** call on the reply context.

Mixing of routed and non-routed links on spacewire connections, such that some links can be used for routed spacewire connections and other links can be used for proprietary formats.

The **esl::spacewire::device_t** class has been modified to enable callbacks whenever an RMAP call is made with the parameters to the RMAP call. This allows for applications to see the RMAP calls they are making via the **diagnostic_text** virtual function on the esl::spacewire::connection_t object

Detailed RMAP debugging is available by setting the following environment variables **ESL_SPW_DEBUG_FILE** to the debug file and **ESL_SPW_DEBUG_LEVEL** to a value indicating the level of debug required.

Hardware Support

C++/Python

API calls are now available for the following hardware platforms

- RTR4
  - Adding / Querying routes
- SRS4/SRS8
  - Adding / Querying routes
  - Setting SpaceWire link speeds
- Loki (Router)
  - Adding / Querying routes
  - Setting SpaceWire link speeds
  - Querying build information
  - Configuring System interface
  - Configuring MRAM interface
  - Setting / Reading MRAM values

LabView

Separate header file for use by the LabView import wizard present in the **include/labview** directory present in the install package. This header file is standalone and does not require other support header files and is optimized for LabView.

The following functions have been added for use RMAP applications within a LabView environment

void * EtherSpaceLink_RMAP_EndPoints (void * con_ , int link_, uint32_t flags_, uint32_t timeout_, const char * str_)

This function creates an endpoint for communications with an RMAP Target

con_ is the connection to the DSI as returned by **EtherSpaceLink_open**
link_ is the link number on the DSI
flags_ is a bitmask of flags controlling communication with the target
bit 0 (1) if set allows multiple RMAP commands to be used in different threads
timeout_ is the timeout on the RMAP response
str_ is a string defining the path address and end point.

- the packet path
- logical address
- return path
- return address

For example: **[1 2 3 4 0] 73 [4 3 2 1] 65** creates an RMAP endpoint with path address 1 2 3 4 0 and logical address 73 with a return path of 4 3 2 1 and return logical address of 65.

The return value is the end point for use in the read and write functions described below.

int EtherSpaceLink_RMAP_SYNC_Write (void * endpoint_ , uint8_t flags_, uint8_t key_, uint8_t ewa_, uint32_t addr_, const unsigned char * data_, size_t size_)

This function performs an RMAP Write

endpoint_ value used return by **EtherSpaceLink_RMAP_EndPoints**
flags_ RMAP write flags (4 for auto increment 0 otherwise)
key_ key to use in the RMAP write verb
ewa_ Extended Write Address
addr_ Address to write to
data_ Data to write
size_ Size of the data to write

The function returns < 0 if a network error has occurred otherwise it's the return code from the **endpoint**

int EtherSpaceLink_RMAP_SYNC_Read (void * endpoint_ , uint8_t flags_, uint8_t key_, uint8_t ewa_, uint32_t addr_, unsigned char * data_, size_t size_, size_t *rsz_)

This function performs an RMAP Write

endpoint_ value used return by **EtherSpaceLink_RMAP_EndPoints**
flags_ RMAP write flags (4 for auto increment 0 otherwise)
key_ key to use in the RMAP write verb
ewa_ Extended Write Address
addr_ Address to write to
data_ where the response is written to
size_ size of the data we wish to read
rsz_ size of data actually returned by the endpoint (if rsz_ > size_ then the data is truncated at size_ bytes)

The function returns < 0 if a network error has occurred otherwise it's the return code from the **endpoint**

Utilities

spw_ioio

This utility is a native version of the java spw_io program (and if java isn't installed becomes the default program used as spw_io)

The following commands have been added

__help ()

Provides help on commands supported by the utility

__status

Displays which the link status of all links on the currently active DSI.

__detail

detail on|off Enables / Disables detailed packet logging for packets produced by plugins or in built commands such that if you want to see the packet data being sent by a plugin or by the use of one of the RMAP commands.

__spacewire_endpoint

Creates a logical endpoint for routed SpaceWire operations, where you specify

- endpoint name
- the dsi address
- port number on the DSI
- the packet path
- logical address
- return path
- return address

For example, **__spacewire_endpoint endpoint dsi 4 ([1 2 3] 63 [4 5 6] 78)** would create a routed spacewire endpoint on with path addressing bytes 1 2 3 to logical address 63 and return path addressing of bytes 4 5 6 and return logical address of 78 on port of 4 of the dsi. Preceding the value with a # indicates a hexadecimal value.

There must be an active spacewire connection on the given port of the dsi.

__rmap

Allows the user to send RMAP command and control RMAP diagnostics.

detail <device> on|off switches on or off detailed recording of RMAP data on the given spacewire endpoint

read <device> key flags ewa address size perform an RMAP read on the given spacewire endpoint where

key is the key value as per RMAP protocol

flags should be 4 if auto increment is required 0 otherwise

ewa (**Extended Write Address)** a value between 0 and 255 to provide a 40 bit addressing range when combined with address (most significant byte)

address lower 32 bits of address being accessed

size how much data you want to retrieve

The program displays the status code on receipt of the reply. If the reply is less than 128 bytes in size the response will be displayed otherwise the data will be written to a file and the filename will be provided.

write <device> key flags ewa address data perform an RMAP write on the given spacewire endpoint where

key is the key value as per RMAP protocol

flags should be 4 if auto increment is required 0 otherwise

ewa (**Extended Write Address)** a value between 0 and 255 to provide a 40 bit addressing range when combined with address (most significant byte)

address lower 32 bits of address being accessed

data The data you wish to write if you have a small amount of data then you can specify it in [ ] e.g. [1 2 3 4 5]. If you have a large amount of data to write then use @**filename** and the contents of the file will be sent

The program displays the status code on receipt of the RMAP reply

rmw <device> key flags ewa address data perform an RMAP read on the given spacewire endpoint where

key is the key value as per RMAP protocol

flags should be 4 if auto increment is required 0 otherwise

ewa (**Extended Write Address)** a value between 0 and 255 to provide a 40 bit addressing range when combined with address (most significant byte)

address lower 32 bits of address being accessed

data you can specify it in [ ] e.g. [1 2 3 4 5]. If you want to use a file then use @**filename** and the contents of the file will be sent

mask you can specify it in [ ] e.g. [1 2 3 4 5]. If you want to use a file then use @**filename** and the contents of the file will be sent

The program displays the status code on receipt of the reply as well as the replied data.

__rtr4

Allows the user to control the routing table of the RTR 4 router. Unlike other devices the RTR4 has a fixed TX speed of 50Mbits and a maximum RX speed of 50Mbits.

route <device> key address port [delete]

Sets up a route on the RTR4 addressed by the spacewire endpoint **device** (note that the you need not specify the RMAP 0 control address in the path), **key** is the key configured on the unit. **address** is the logical address which is being configured. **port** is the physical of the router to which traffic for the given logical address is routed. **delete** is an optional parameter which specifies whether the first byte of the message is deleted.

get_route <device> key address

Gets the route for the given logical address on the RTR4 addressed by the spacewire endpoint **device** (note that you need not specify the RMAP 0 control address in the path), **key** is the key configured on the unit. **address** is the logical address which is being requested

routes <device> key

Displays all the routes for the given logical address on the RTR4 addressed by the spacewire endpoint **device** (note that you need not specify the RMAP 0 control address in the path), **key** is the key configured on the unit.

__srs

Allows the user to control the routing table of the SRS router.

route <device> key address port [delete]

Sets up a route on the SRS addressed by the spacewire endpoint **device** (note that the you need not specify the RMAP 0 control address in the path), **key** is the key configured on the unit. **address** is the logical address which is being configured. **port** is the physical of the router to which traffic for the given logical address is routed. **delete** is an optional parameter which specifies whether the first byte of the message is deleted.

speed <device> key port devicespeed

Sets the link TX speed on the specified port on the SRS addressed by the spacewire endpoint **device** (note that the you need not specify the RMAP 0 control address in the path), **key** is the key configured on the unit. **port** is the physical of the router being set, **devicespeed** is the TX speed in MBs of the port

get_route <device> key address

Gets the route for the given logical address on the RTR4 addressed by the spacewire endpoint **device** (note that you need not specify the RMAP 0 control address in the path), **key** is the key configured on the unit. **address** is the logical address which is being requested

routes <device> key

Displays all the routes for the given logical address on the RTR4 addressed by the spacewire endpoint **device** (note that you need not specify the RMAP 0 control address in the path), **key** is the key configured on the unit.

__loki

Allows the user to control one of the 4Links **loki** boards. This loki command line is also documented in the loki folder in the manuals directory.

create name endpoint key

Returns basic information (version number etc) about the loki board at the given spacewire endpoint with the given key. For various reasons the spacewire address must include the port 0 for accessing the RMAP 0 control plane.

scan name endpoint key

Scans the given endpoint for peripherals and creates them as appropriate.

Creates the following devices as appropriate

name.control.latt.0 used for controlling the routing of spacewire traffic on the inbuilt spacewire router

name.control.spacewire.N used for controlling the physical attributes of spacewire port N

name.control.mram.N used for controlling the MRAM bulk storage device present in the loki board

name.control.router.0 used for controlling the core configuration of the router board (key, port etc)

name.control.latt.0

The command __loki (**name.control.latt.0 <arguments>)** is used for controlling the routing table discovered above. It is like the routing commands present in the srs and rtr4

route address port [delete]

Sets up a route on the router. **address** is the logical address which is being configured. **port** is the physical of the router to which traffic for the given logical address is routed. **delete** is an optional parameter which specifies whether the first byte of the message is deleted.

routes

Displays the routing table

name.control.spacewire.N

The command __loki (**name.control.spacewire.N <arguments>)** is used for controlling the physical attributes of spacewire port N on the loki board

status

Displays the current status of the given spacewire link

disable

Disables the given link

enable

Enables the given link

speed

Sets the speed of the link in Mbs. The speed must be supported by the hardware

name.control.mram.0

The command __loki (**name.control.mram.0 <arguments>)** is used for controlling the bulk storage MRAM device

status

Displays the current status of the MRAM controller

nola <value>

Sets the MRAM logical address to the given value

mkey <value>

Sets the MRAM logical address key to the given value

echp <value>

Sets the echo port to the given value

csr0 <value>

Sets the control register csr0 to the given value

auth lacheck keycheck echocheck

Controls the logical address, key and echo port checking, values are 0 or 1 as appropriate

name.mram.0

The command __loki (**name.mram.0 <arguments>)** is used for reading/writing to/from the bulk storage MRAM device

write offset data

Writes **data** to the MRAM device at **offset**, **data** may be an values specified by **an array [v1 v2 v3 ..]** or if you want to write a file use **@filename**

read offset size

Reads **size** bytes starting at offset **offset** from the mram storage

name.control.router.0

The command __loki (**name.control.router.0 <arguments>)** is used for controlling the core router functionality and addressing

status

Displays the current status of the board

nola <value>

Sets the router logical address to the given value

mkey <value>

Sets the router key to the given value

echp <value>

Sets the echo port to the given value

RMAP Plugins

RMAP is a transport protocol sitting on top of SpaceWire on which developers build their own application protocols. Users may wish to develop their own application plugins on top of SpaceWire without recourse to writing their own RMAP stacks.

The spw_ioio command offers plugins for encoding and decoding packets received, this has been extended such that developers can write applications within the lua environment for handling the transmission and reception of RMAP frames allowing complex behaviours to be built beyond encoding and decoding of application data. RMAP reply data will **not** be presented to any protocol decoders.

The following commands are added

esl_rmap_read (name, flags, key, ewa, sz)
performs an RMAP read command it takes the the following arguments

the spacewire endpoint name
rmap flags (4 if you require auto increment)
the RMAP key
EWA Extended Write Address
Address of the data to read
Number of bytes to read

This function returns < 0 if a network error occurred, otherwise it is the status code returned by the RMAP target.

esl_rbyte (index)

Returns the byte at the given index

esl_ruint32 (index, littleendian)

Returns the 32 bit word at the given index
If littleendian is non zero then the 32 bit value will be treated as little endian
If littleendian is zero then the 32 bit value will be treated as big endian

An example of an RMAP plugin using the read calls shown above is held in the file rmap_read_application.lua

esl_rmap_write (name, flags, key, ewa, sz)

performs an RMAP read command it takes the the following arguments

the spacewire endpoint name
rmap flags (4 if you require auto increment)
the RMAP key
EWA Extended Write Address
Address of the data to read

This function returns < 0 if a network error occurred, otherwise it is the status code returned by the RMAP target.

esl_wbyte (index, value)

Writes the byte at the given index

esl_wuint32 (index,value. littleendian)

Returns the 32 bit word at the given index
If littleendian is non zero then the 32 bit value will be treated as little endian
If littleendian is zero then the 32 bit value will be treated as big endian

An example of an RMAP plugin using the read calls shown above is held in the file rmap_write_application.lua

Utilities

esl_rtr4

Utility for configuring RTR4 from JSON configuration file (not available for CentOS 5/6)

esl_srs48

Utility for configuring SRS4/8 from JSON configuration file (not available for CentOS 5/6)

esl_loki

Utility for configuring LOKI from JSON configuration file (not available for CentOS 5/6)

Software 35.34

API

Time Synchronization: Ability to set the time synchronization state of a 1U unit using command line tools or API verbs. See **esl_tt_mode**

Waveform call-back (C++)
The waveform call-back function has been modified to accept the timestamp that the waveform has been generated on and the raw waveform data has been placed in an access **class.**

virtual bool waveform_data (int erridx, int unit, int port, int ew_port, int event, char * cause, esl::timestamp * ts, esl::utils::special_data * data_) {return true;}

Waveform on Connect and ErrorReset states
With suitable versions of the 1U and 3U firmware it is now possible to gather waveforms on transition to the ErrorReset and Connected states.

Python

Documentation

Updated **python.pdf** documentation

Raspberry PI

We now support python on the Raspberry PI

Waveform Call-back

It is now possible to record waveforms in python see python example **connect_waveform**

Java

Documentation

Updated **java.pdf** documentation

Java 17

We now provide support for Java 17 which has LTS to 2029 see
https://www.oracle.com/java/technologies/java-se-support-roadmap.html

Waveforms

We now provide a call-back for the receipt and saving of waveforms

platforms/Java

Various Linux platforms use different versions of Java based upon OpenJDK and the packages for those platforms are based on the version of java available to the release. We provide support for versions of Java the libraries and executables being found in this directory. Note the two subdirectories are for 32-bit and 64-bit versions of Java

C++

We ship a C++ API to use with our units, unfortunately there is no standard for decorating names in C++ and use of slightly different compilers may result in linkage errors. If you suffer from a unresolved symbols which should be resolvable please contact support@4links.co.uk and we will rebuild for your particular compiler

Debug Library

We now ship the instrumented debug library in a separate debug directory enabling detailed debugging to be performed without over writing the live non debug library. This debug library can be selected by setting LD_LIBRARY_PATH and PATH variables. If you are having trouble with your application, please use **ESL_TX_DEBUG and ESL_RX_DEBUG** (discussed in the concept guide)

Command Line

esl_tt_mode
Utility for controlling time synchronization mode for 1U devices. Usage is

esl_tt_mode <ipaddr> to find current state

esl_tt_mode <ipaddr> isol to set isolated

esl_tt_mode <ipaddr> sync to set synchronized

esl_tt_mode <ipaddr> ref to set as a reference for other units

CCSDS Plugin

We now have a Spacewire CCSDS plugin available for the nativene read program

Software 35.32

API

Support for timecodes automatically generated on 1U (4.7+) and 3U (2.3+)
Allow automatic timecodes to be generated (see timecodes.pdf)

Debug Environment Variables
Ability to track what your application is doing on spacewire using ESL_TX_DEBUG and ESL_RX_DEBUG (discussed in the concept guide)

Support for timecodes clocked on 1U SMA (4.8+)
Allow timecodes to be generated under external hardware control (see timecodes.pdf)

Support for reading 1U Sensor values (4.8+)
Introduced verb for reading 1U temperatures, fan speed and battery voltage synchronously and asynchronously using **EtherSpaceLink_device_sensor**. The following sensors are supported (starting at sensor 0)

- Temperature Zone 1
- Temperature Zone 2
- Fan Speed (*32)
- VBatt

See **esl_version.c**

Support for setting SMA voltage thresholds on 1U (4.8+)
EtherSpaceLink_set_SMA_name_voltage / **EtherSpaceLink_set_SMA_voltage** verbs introduced.

Valid names are

SMA_78_RXV
SMA_56_RXV
SMA_12_RXV
SMA_34_RXV
SMA_78_TXV
SMA_56_TXV

API Enumerations

EtherSpaceLink_SMA_78_RXV
EtherSpaceLink_SMA_56_RXV
EtherSpaceLink_SMA_12_RXV
EtherSpaceLink_SMA_34_RXV
EtherSpaceLink_SMA_78_TXV
EtherSpaceLink_SMA_56_TXV

Python Examples
The examples had the key and the flags field the wrong away around.

Applications

spw and **spw_ioio** support automated generation of timecodes (see timecodes.pdf)

msr now supports reset of timetags via **/**resetTT option on 1U (4.7+) and 3U (2.3+)

spw,**msr** and spw_ioio now support setting threshold voltages on the SMA

Documentation

timecodes.pdf describing automated timecodes

concept_guide.pdf introduction to spacewire and the 4Links products

Software 35.28

Platform

We now support **Debian 11 (Bullseye)**

API

Windows PIP/Wheel Package for Python 3.7
This package would not run standalone (it required 4 Links API to be installed as well)

Windows Python 3.9
We now support Windows 3.9 as part of the API and a PIP/Wheel package

Documentation

Updated **python.pdf** with information about Windows Python Support. If the version of python being used is not on our supported list, end users are requested to contact support@4links.co.uk in order for us to build support for the requested version of python

Software 35.26

Documentation

Device Update

updates.pdf how to update the firmware for units

Error Injection

ErrorInjection.pdf error injection documentation and examples
transport_credit example program controlling transport credit

API

C

EtherSpaceLink_set_speed_delay**,** the function **EtherSpaceLink_set_speed** can take a while to transition from the one speed to another (for example 10Mbs to 200Mbs) as the PLL is transitioned to the correct speed. It is possible for your application to run at incorrect speeds. In order to alleviate this issue, we now introduce a delay. You can use **EtherSpaceLink_set_speed_delay** to extend or remove the delay.

C++/Python

speed_delay binding to **EtherSpaceLink_set_speed_delay**

Applications

spw_msrr

spw_msrr is a native version of the **java read** program. It provides the same functionality as the java program without having to install a Java Runtime.

We have added two options to this program for managing the timestamp display of the unit (currently the time displayed is the time in nanoseconds since unit powered on)

-epochtime displays the time as a nanosecond / epoch combination

-humantime displays the time in a conventional date format

We now ship the dissectors for RMAP on Windows platforms.

Software 35.24

Documentation

End User License Agreement

We now provide a formal End User License Agreement

Programs

spw_echo

This program is used to echo data between ports. It has been updated to

1. Support proxying of data between more than two ports
2. Allow filtering of data using a LUA plugin (see plugins\lua\spwecho/frame_filter.lua)

spw_ioio spw_msrr

These replacements for the java READ and SPWIO java programs now correctly run on windows

spw_read spw_io

Windows wrappers for the java READ and SPWIO java and spw_ioio, spw_msrr programs. If installed the java programs are called. If java is not installed the compatible spw_msrr and spw_ioio programs are called.

Software 35.22

Installation

Debian Stretch

We support Debian Stretch. Packages are supplied in deb format.

Debian Buster

We support Debian Buster. Packages are supplied in deb format.

Ubuntu

Packages are now supplied in deb format, and the use of alien is no longer required.

Alma Linux 8

With the change of emphasis of future CentOS builds, various new distributions are coming to the market in order to fulfill the same role. The first of these is Alma Linux.

Packages are in the standard RPM format.

Utilities

spw_msrr

A new program which is identical in command syntax to the Java Read program. This provides read functionality without having to install a Java runtime. Custom dissectors can be written without recourse to java or C++ using the **Lua** programming language. See the document **dissectors.pdf.**

Note that this program is not available on CentOS 5 or CentOS 6.

spw_ioio

A new program which is identical in command syntax to the Java SpwIO program. This provides spwio functionality without having to install a Java runtime. Custom dissectors and encoders can be written without recourse to java or C++ using the **Lua** programming language.

See the document **dissectors.pdf** for information on writing dissector and **encoders.pdf** for writing an encoder.

Note that this program is not available on CentOS 5 or CentOS 6.

READ.jar

This program used to give extreme (10^9 megabytes per second) estimated link speeds when decoding packets. This has been resolved.

The RMAP dissector has been updated to include better information to handle read modify writes better.

We now show auto increment as **AI** rather than **"..."**

3ucodes

This program displays diagnostic information for error codes on the 3U units. If you don't supply a code, it displays information about all codes, if given a specific code to displays information about that code.

3usetup

This program allows users to quickly configure their 3U devices.

spw_io

This is a wrapper to call the Java Program spwio.jar , if the java app is not installed it will call the **spw_ioio** program.

spw_msr

This is a wrapper to call the Java Program read.jar , if the java app is not installed it will call the **spw_msr** program.

API

C++/Python Virtual Callbacks

All virtual callbacks are now private virtual functions allowing callbacks to be **only** called when the event occurs. This primarily is meant to resolve issues associated with calling virtual functions from python resulting in application locking up due to **GIL** issues.

Fuzzing Support

pre_flush application callback added to aid fuzz testing, see the document **fuzz.pdf** in the documents section.

C++/C/Python Protocol Decoders

We provide API access to the decoder library for C/C++ and python programs. This allows users to embed 4links and custom dissectors in their applications. The C binding may be used to access the decoder library from LabView as well.

Python examples are **msr_decode_file** and **msr_decode_buffer**

C/C++ examples are **proto_decode_buffer.c** and **proto_decode_file.c**

C++/Python

set_close_on_exec (Linux/MacOS) if you wish to use fork in your application you may need to set this flag in order to ensure correct network operation.

Software 35.20

API

C/C++/Python

The address of the 4Links unit can have a bind device / address specified, such that the connection is bound to that device / ip address prior to the connection being made. This is achieved by appending @<devicename> or @ipaddress to the connection string, for example dsi@eth0 or dsi@10.10.10.1. Port number overrides must be present before this string , for example dsi:4949@eth0.

Binding to a device is not present on windows, and may require privilege on certain platforms.

RMAP

When performing an RMAP write of zero bytes an additional 0x00 EOP sequence was sent this has been resolved

The RMAP API documentation has been updated to include references to the examples

C

It is now possible to initiate MSR captures from C, since these functions are exported in our DLL it is now very easy to capture from such tools as LabView. See **eslcapture.h**

Utilities

rmap_server

In order facilitate testing of RMAP initiators who wish to have a memory read/write interface we have shipped rmap_server. It's usage is **rmap_server <dsi> port** where dsi is the ip address of your DSI and port is the port number we attach to. There will then be an RMAP memory target running on the DSI.

Simply connect your initiator to a spacewire port on your DSI and you should be able to test it.

Software 35.18

API

RMAP

Server performance improvement, previously RMAP API only read from 8 byte chunks, now uses large network buffer.

0 byte transfers, rmap reads,writes, and readmodifywrites with 0 sized buffers would fail.

Improved memory usage, rmap does not allocate buffers for handling rmap responses instead uses the rmap transaction parameter as a context.

Transaction Identifier, introduced API verbs for controlling the scope of the transaction idenifier (previously on a per dsi basis), now can be on a per port basis, per dsi basis, and global basis.

Allowed user callback class to assign transaction identifiers

Software 35.16

API

C

Introduced **EtherSpaceLink_hold_data** as a means of holding data until either J1/J2 transition to high.

C++ / Python/ Java

hold_data as a means of holding data until either J1/J2 transition

C++ / Python RMAP

clientside/serverside API available see **rmap_api.pdf**
A set of examples for C++ and python
C++ application programming simplified
C++ callback now passed target logical address rather than return address logical address
Sometimes RMAP Read calls returned with previous data due to locking issue

MSR API

Improved example for msr python binding
Now accepts JSON capture configuration

Java API

Java API now reports errors by exception rather than return code

Documentation

User Guides

Some of the user guides had an incorrect pinout with respect to the spacewire D type connector

Python

Updated documentation on the python capture interface classes **msr_capture** and **pcap_capture**

.

Utilites

Virtual DSI

On Windows platforms the return value from the accept call was not checked correctly.

Web Interface

We provide a **msrw** an application which provide a web based JSON service allowing JSON requests to control the acquisition of data from an msr. This Is documented in **web_msr.pdf.**

Java Wrappers (Linux and MAC)

These wrapper classes did not handle quoted parameters correctly. This has been resolved.

Software 35.14

API

Non Blocking Calls

When non-blocking calls are made without erroring out it is not possible to differentiate between timeout calls and calls interrupted by meta data callbacks as both cases have PART_EOP_EEP set as the packet type. That is a return value of 0 and metadata type set to PART_EOP_EEP.

This has been changed such that in the event of a timeout the metadata is now set to TIMEOUT.

Software 35.12

API

C

We now provide **libetherspacelink_static.a** on Linux/Mac and **etherspacelink_static.lib** on Windows, for those developers who wish to have a runtime embedded into their applications without recourse to an external runtime.

Python

We now support python 3 on **CentOS 7** and **CentOS 8**

We now supply a PIP (whl) package which can be used to install the python module without recourse to install the complete package. We currently do not support source whl distributions.

Utilities

vdsi Introduced a virtual **DSI** utility allowing application developers to code against our API without having physical access to our hardware. This may be useful for software test and integration. Please see the documentation in **docs/V-DSI**

Wireshark

Wireshark.pdf has been updated with the complete list of packet codes

Software 35.08

API

C++

nolinks function now returns unsigned as can't have a device with negative links and errors are thrown.

active_port now accepts an unsigned port number as port numbers can't be negative.

Java

The supplied jar file etherlinks.jar was built using an unnamed package which may be problematic for developers. The jar file now packages in **com._4links.spacewire.**

In order to change your java accordingly you need to add **import com._4links.spacewire.*;** to your java.

Utilities

spw

The program breaks down packets into chunks for transmission. When only transmitting on a single link it no longer does this.

The spw program now prints out the chunk size it is using, and if no chunking is taking place indicates this too.

Documentation

graphical_msr.pdf

Updated documentation with a section on how to use our software in conjunction with gtkwave.

Software 35.06

API

C++

We have reduced the size of the esl::connection class such that it is easier to instanciate said objects on the stack. Previously objects of this type held on the stack (especially in Windows) would lead to program aborts.

Software 35.04

API

python

We have introduced the **msr_capture** and **pcap_capture** classes for application developers who wish to control an msr during their runs. The msr_capture captures in the 4links native format and pcap_capture captures in a pcap form.

The constructor takes 4 parameters

- The hostname / ip address of the msr
- The target file / directory
- The capture configuration file (as generated by the GUI program
- File Size, if you want the capture to split the data into multiple files please specify the file size that you want the capture to be split into, this size is in MB. If you don't want file splitting then specify 0

We have supplied the python examples msr and msr_cap.

Python API Performance

We have updated the python API guide discussing implications of pythons threading model.

The python interpreter has some known issues with respect to threading, whilst python uses system threads to implement threading, the actual interpreter can only run in one thread at a time, if other threads are not waiting on I/O they will be waiting for the python interpreter to be assigned to that thread.

Essentially python code in python threads is run co-operatively.

A more elegant way of putting this, is that currently you will never get more than a single cores worth of python code running at any instant.

If you need to have more than a single cores worth of code running. then your application must be process based as opposed to thread based.

Java Read Application

This program displayed false EOP's at the end of packet captures. This has now been resolved.

Software 35.00

API

C

EtherSpaceLink_open does not check the result code of the asynchronous connect call correctly and will try and negotiate on a connection which is not connected resulting in a somewhat incorrect error code being returned. You will still get an error, but with this correction or more applicable error.

C

Introduction of **EtherSpaceLink_request_link_status_port** which requests port status without switching active link on DSI/MSR and 3U devices. On ESL devices it will switch the active port.

C++/ python

request_link_status_port method added which performs the same as the function above for the connection class

The timetag callbacks on Windows would report incorrect wall times thus prdocuing incorrect pcap packet timestamps.

msr

Corrected usage of the /f parameter so as not to error.

The msr program has the ability to break large captures into smaller configurable chunks. Currently they are incrementally numbered, the file name is now the start and end times of the capture. If you want the chunks to be incrementally file numbered set the environment varable **ESL_FILE_INDEX**

PCAP capture

The offset from UTC in the PCAP header is now populated ensuring consistent timing when reading pcap files in different timezones..

Software 34.98

New Linux Platform

We now support **Ubuntu 20.04** with this release comes for support for both python 2.7 and python 3.

The default is for python 2.7 (as this is the install default) but if you wish to use python 3 then set the environment variable **ESL_PYTHON_VER** to **3**

Software 34.96

Release Notes

msr capture program

The msr capture program allows users to capture traffic on specific ports, currently it captures traffic on all ports irrespective of this parameter. This has now been rectified.

Software 34.94

spwio Documentation

The PDF for spwio was incorrectly printed in landscape as opposed to portrait.

esl_srscfg

We have introduced the **esl_srscfg** utility in order to make configuring SRS routers simpler.

esl_spw

We have introduced **esl_rspw** a utility which allows one to send and receive RMAP/CCSDS and raw messages without resort to complex programming.

spwio RMAP plugin

This plugin displayed "..." when either more than 8 bytes of data were received or if the increment address flag was set, this caused a degree of confusion. Instead, if the increment address flag is set the plugin displays the token "AI"

Software 34.87

MSR Capture

When requested to capture NULLS (ESC-FCT) the msr capture program generates the appropriate records whilst recording to PCAP formatted files.

Java Read Program

The Java read program now displays NULLS (ESC FCT) when encountered.

Wireshark Plugin

The wireshark plugins now display ESC FCT (NULL) and FCT frame data.

Spw program

We now allow a spin up period in the average throughput calculations

SRS Configuration

We have improved error reporting when configuring SRS routers.

Software 34.86

CentOS-8

We now provide support for CentOS-8

Apple Mac 19.0.0

We now provide support for MacOS Catalina (19.0.0)

API Change

EtherSpaceLink_dump_status

This function provides a means of dumping given context to stderr for help in debugging applications.

Software 34.83

Wireshark / PCAP Integration

When using packet aggregation with the final EOP being preserved (ESL_PCAP_RECORD_EOP_TS) we didn't preserve the final EOP if short timestamps were sent by the msr. This is now rectified.

API Change

The EthersSpaceLink_close, EthersSpaceLink_shutdown, EthersSpaceLink_abort calls now shutdown the spacewire ports automatically, user applications no longer have to.

Documentation

The documentation for the 3U was missing, this is now included in the doc package.

Software 34.82

Release Notes

Graphical Capture program

When running capture recording to multiple files, the file size limit is now respected.

Software 34.81

Version Number change

We have now added patch release numbers to the version identification.

The version numbering is now of the format

<ReleaseNumber>.<VersionNumber><PatchLevel>

The first patch to this version will be known as 34.82.

C++ RMAP

The C++ API for RMAP has been corrected with respect to the endianness of the command byte, previously the bit positions of command flags were reversed causing corrupt RMAP messages.

READ program

The java read program did not load the RMAP plugin correctly.

RMAP plugin

Support for Read Modify Write verb has been added

MSR capture program

The msr capture program places waveforms in the current directory if the environment variable ESL_WF_DIR is not set. ESL_WF_DIR is an environment variable that controls where waveform files are saved.

Waveform files

When a waveform file is saved by the msr program, the name of the file is printed to stdout and the waveform file is prefixed with the epoch time. Allowing easy identification of the capture file in any logs.

SVG Waveform files

These are now scaled correctly, so there are no longer have large blocks of unreadable color.

Python 3.7 Support on Windows

Support for Python 3.7 on windows. If python 3.7 is required, set ESL_PYTHON_VER to 3.7 prior to running 4linksvars.

Python timecode example

examples/python/timecode is an example of how to send and listen for Spacewire timecodes.

Graphical Capture program on Windows

The graphical capture program uses GTK, however the GTK runtime ships with a DLL used by other packages and this can stop the program starting. The environment program 4linksvars.bat puts the GTK libraries in the path before other libraries.

Change of License for example programs

All examples are now shipped with the BSD license making copying and reuse of the examples easier.

Graphical Capture program / WIreshark Integration

It is now possible to capture directly into Wireshark from within the graphical capture program

Improved PCAP buffering

The PCAP file buffer has been made an optimal size.

API Changes

Return Values

Throughout a return value of –1 was used to indicate an error, this has been changed. If a function returns a value < 0 an error has occurred, the returned value being the error code as described in EtherSpaceLink_Constants.h. This change does not affect the C++ or Python bindings as error handling is now by exception.

EtherSpaceLink_read_packet_full

When called with **EtherSpaceLink_REPORT_SPECIAL_DATA** and **EtherSpaceLink_REPORT_EXTENSION_DATA** this function used to return –2 – <number of bytes>. This conflicts with the error handling as defined above. Instead the returned flags indicate whether a size is being returned. If the returned flags are **EtherSpaceLink_SPECIAL_SIZE** or **EtherSpaceLink_EXTENSION_SIZE** then the returned value is the number of special or extension bytes.

Example:

```
unsigned char rxbuf[4096];
int bytes_received, extension, ii;
int flags;
int active_port;

EtherSpaceLink esldev = EtherSpaceLink_open("1.1.1.1:1234");
if (!esldev)
{
    printf( "Unable to connect\n");
    return 1;
}

bytes_received = EtherSpaceLink_read_packet_full ( esldev,
                                    rxbuf,
                                    sizeof(rxbuf),
                                    &flags,
                                    EtherSpaceLink_REPORT_EXTENSION_DATA
                                    | EtherSpaceLink_REPORT_SPECIAL_DATA
                                   );
If (bytes_received < 0)
{
   // Error condition
   fprintf (stderr, “Error %d \n”, bytes_received);
   return 1;
}
switch (flags)
```

```
{
    case EtherSpaceLink_SPECIAL_SIZE:
    case EtherSpaceLink_EXTENSION_SIZE:
            fprintf (stdout, "%d bytes of special data\n", bytes_received);
}
```

receivePacket (java)

This function no longer reports the size of extension or special data as a negative number. Instead one must check the context of the return value by calling **get_terminator.** If this function returns **SPECIAL_SIZE** or **EXTENSION_SIZE** then the size returned is the extension size. Note that if **enable_callbacks** is called the application developer does not have to perform analysis of special and extension_data, and instead you get high level callbacks (e.g. link changed, error, or waveform data);

EtherspaceLink_fastclose

Introduced new API verb which shuts down the connection as soon as possible using SO_LINGER with a timeout of 0.

Language Implementations

C++ method fastclose

Java method fastclose

Python method fastclose

EtherspaceLink_sendtimecode

Explicit function to send a Spacewire timecode.

Language Implementations

C++ method send_timecode(timecode)

Java method send_timecode(timecode)

Python method send_timecode(timecode)

EtherspaceLink_get_error

This returns the last error to occur in the **current thread** in the EtherSpaceLink library, mostly it should not be required to call this as functions return the error code directly. This change ensures that error codes are thread safe between receiver and transmit threads.

EtherspaceLink_get_error_text

This returns a string description of the error last error to occur in the **current thread.**

This change ensures that error codes are thread safe between receiver and transmit threads.