Software Manual – Programming and Integration

SEA 9521

BiSS/SSI Interface Module

s·e·a Science & Engineering Applications Datentechnik GmbH

Doc. No.: HB/SEA 9521 SoftwareManual/1.2.b/Mar-2023

Subject to modifications.

S.E.A. Datentechnik GmbH takes no responsibility for damage arising out of or related to this document or the information contained in it.

Product and company names listed are trademarks or trade names of their respective companies.

Address:

S.E.A. Datentechnik GmbH
Muelheimer Strasse 7
53840 Troisdorf
Germany

Phone: +49 2241 12737-0
Fax: +49 2241 12737-14

For support please contact the support email address:

techsupport@sea-gmbh.com

Contents

1 Change Notes

#	Version	Changes
1	1.0.0	Release of the new BiSS/SSI driver architecture based on the NI Module Development Kit 2.0, MDK2.
2	1.1	Minor changes: • A4 page size • Getting Started chapter • Installation chapter • Deployment chapter
3	1.2.a	Changes: • Support for NI CompactRIO 904x targets added. Oldest supported LabVIEW version is now 2017.
4	1.2.b	Changes: • Changes of document formatting • New entries in table 6 and 9 • New section on trigger latencies in chapter 5

2 Getting Started

2.1 General

Before starting to work with the SEA 9521 module please read this document and the complete hardware manual carefully. If there are any questions about operating the module or if any term is not understood, please contact the vendor before using the module. Please check the download area on the S.E.A. website `https://www.sea-gmbh.com` for updates of the manuals.

Refer to the hardware manual for details on operation instructions, safety guidelines and specifications for the SEA 9521 module.

Refer to the appropriate NI™ documentation for details on NI™ hardware.

We believe that all information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event of technical or typographical errors, we reserve the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult the vendor if errors are suspected.

2.2 End User License Agreement (EULA)

Before operating the SEA 9521 module and the provided software you have to agree to the terms and conditions (EULA). This agreement is part of the software installation procedure. If you do NOT agree you can send back the hardware and software package within a period of two weeks after delivery. In this case S.E.A. will refund the product price and shipping costs.

In addition, the terms and conditions are available through the LabVIEW™ `Help` menu after installation.

2.3 Symbols, Notations and Nomenclature

To improve clarity specific structuring elements (symbols) are used which have the following meaning:

Symbol	Meaning
Names	Specific names are printed using an *italic font*
[Text]	Place holders are marked by squared brackets
	Locations (paths, menus, URLs...) are printed using a `courier font`
	The yellow sign highlights important notes and warnings
	The blue mark highlights tips
	Reference to other documents

Tab. 1: Structuring elements/symbols

For a better understanding, a short list will be given:

Name	Meaning
device	A device is a physical object such as e.g. a (rotary) encoder which is connected via a cable to one of the three BiSS/SSI connectors of SEA 9521.

Name	Meaning
channel (ch)	A SEA 9521 module has three physical channels named CHAN 1 (upper connector), CHAN 2 (middle connector) and CHAN 3 (lower connector). To each channel, a single device can be attached.
slave	Each device can have one (standard) or two slave(s) which can be accessed via the respective channel. In most cases, one slave per device and channel is used. In safety applications, however, it may be necessary to have two redundant physical methods within one device to measure e.g. the angle which then would be addressed as Slave 1 and Slave 2, respectively. In this case, there would be two slaves per single device and channel.
BiSS master	Inside the SEA 9521 module, a BiSS master runs on a FPGA which handles communication with all slaves attached. There are two basic communication modes: a) data communication (high speed) and b) register communication (lower speed). The former reads positional data (and encoder error and warning bits) from the slaves. The latter includes device parameters and general status data etc. The master can be accessed via method, property and IO nodes, see chapter 6.2 *Application Programming Interface (API)*.
BiSS/SSI data (position data)	In the BiSS literature, this will also be called *Single Cycle Data* or *SCD*. It contains the actual encoder data such as encoder position and encoder error and warning bits. A complete BiSS/SSI data word (up to 64 bits) is transmitted every cycle.
register communication	BiSS devices may contain electronic data sheets (EDS) or other (non-)volatile register memory (containing e.g. temperature data) which can be read in parallel to positional data at a much lower speed. This kind of communication is called register communication. In BiSS-C mode a single bit is transmitted every cycle.
register data	Data transmitted using register communication such as temperature values, configuration parameters, electronic data sheet parameters etc.

Tab. 2: Nomenclature

3 Installation

SEA 9521 modules require a special driver software (also called module support) for operation. The driver software can be directly downloaded and installed through the *JKI VI Package Manager*. Additionally the driver software can be downloaded either from the *NI Tools Network*™ or from the S.E.A. download site and installed separately.

Due to continuous improvements the required software is not enclosed with the module hardware when shipped. The latest version of the driver software can be downloaded from the S.E.A. web site:

https://www.sea-gmbh.com

Navigation: On the main page select *Support* from the main menu. On the support page select *Downloads* from the local menu and click on the *Download Area* button. On the download site select your module type from the *Hardware Products* category.

The driver software is to be installed on a development device only (refer to the hardware requirements section below). In order to install the software package double-click or open the .vip file inside the VI Package Manager and follow the instructions on the screen. This procedure installs the driver including application programming interface (API), tools, examples and all related documentation. Further resources (if available) like application notes, drawings etc. can be downloaded separately from the location as stated above.

3.1 Hardware Requirements

The SEA 9521 module requires at least a PC to program and compile the user application. In order to run the application an NI CompactRIO Real-Time controller with the SEA module inserted in an arbitrary chassis slot is required.

The driver software requires a PC (*development device*) for programming/compiling and a NI CompactRIO (*run-time device*) with FPGA chassis to run the application.

The cRIO modules do not auto-discover in NI cRIO-904x/905x chassis. This is a known behaviour. The workaround is to manually add the modules to the LabVIEW project. For more details refer to: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000YSbmCAG&l=en-GB

3.2 Software Requirements

The development device and the run-time device are described in the hardware requirements section above.

Software requirements for the development device:

- JKI VI Package Manager (VIPM) – latest available version
- NI LabVIEW Development environment 2017 or higher
- NI LabVIEW Real-Time 2017 or higher
- NI LabVIEW FPGA 2017 or higher + suitable Xilinx tools
- NI CompactRIO drivers

Software requirements for the run-time device (NI CompactRIO):

- NI LabVIEW Run-Time Engine / LabVIEW Run-Time Engine for Real-Time targets (standard installation)

The driver software (module support) only operates in the FPGA interface (programming) mode of a NI CompactRIO system. The *Scan Interface* and *DAQ* modes are not supported at present.

4 Quick Start

This section describes how easy the SEA 9521 module can be integrated into an existing CompactRIO system.

Refer to the hardware manual for proper installation of the hardware.

The present (Q1/2019) the cRIO modules do not auto-discover in cRIO-904x chassis. The workaround is to manually add the modules to the LabVIEW project. NI works currently on a solution. The NI's Corrective Action Request (CAR) is #687418.

The present (Q1/2019) the cRIO modules do not auto-discover in NI-9144 and NI-9145 EtherCAT chassis. This issue only affects chassis running latest chassis firmware 16.0 and 16.1 which comes with NI-Industrial Communications for EtherCAT 16.0 and 16.1 driver. NI works currently on a solution. The NI's Corrective Action Request (CAR) #637095.

Follow the steps below to create a simple application using the SEA 9521 module and run it on a NI CompactRIO system (for this tutorial the NI cRIO-9040 has been used exemplary):

1. Ensure that you meet the hardware and software requirements listed in the previous chapter and the required software is installed on your PC and on the NI CompactRIO system.
2. Insert the module into your NI CompactRIO system in a slot of your choice (for this tutorial slot 1 has been used). Connect the GPS antenna to the module and place the antenna outside buildings or next to a window.
3. Connect your NI CompactRIO system to the development PC via Ethernet cable and ensure a correct IP configuration of participating devices. Power up all devices.
4. Install the SEA 9521 driver software. For this, the VI Package Manager (VIPM) is required. If VIPM is not yet installed on your system please start LabVIEW and select *Tools → Find LabVIEW Add-ons...*. Complete the VIPM installation following the instructions.
5. Optional (required for cRIO-904x/905x): Detect/add the NI CompactRIO system in the NI-MAX (*Measurement & Automation Explorer*) *Remote Systems* (German: *Netzwerkumgebung*). In the *Devices and Interfaces* settings, ensure that the SEA 9521 is configured to the Program Mode *LabVIEW FPGA*, refer Fig. 1. Close NI-MAX afterwards.

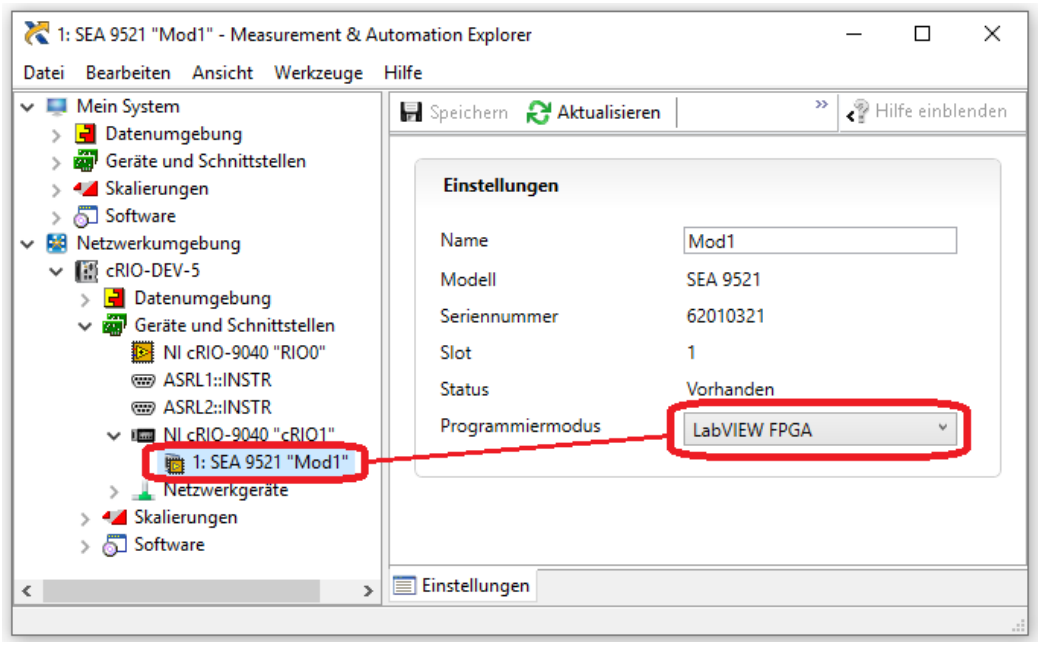


Fig. 1: Quick Start – NI-MAX configuration

6. Start LabVIEW and create a new, blank project.
7. The project explorer window appears. In this window select the uppermost item in the tree (*Project: Untitled Project X.lvproj*) and select *New → Targets and Devices* right-clicking on it, like shown below:

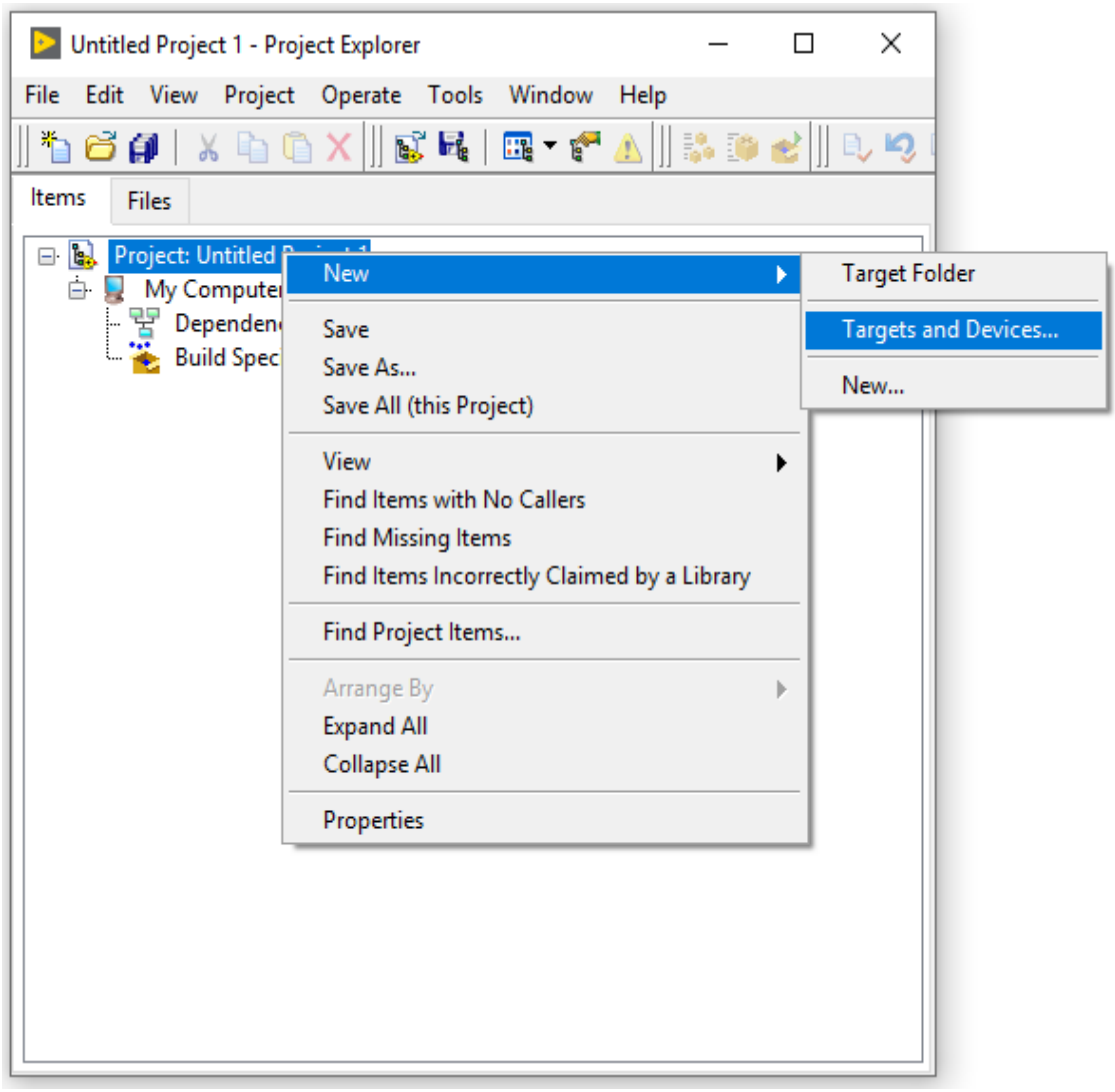


Fig. 2: Quick Start – Add new target

8. Select your NI CompactRIO system from the list of available targets or add it manually. The chassis, FPGA target and cRIO modules are detected automatically. If automatic detection fails please add the system components (chassis, FPGA target, cRIO modules) manually to the LabVIEW project.
9. Ensure that the chassis is configured to *LabVIEW FPGA Interface* RIO programming mode.
10. The configuration is completed. After a successful discovering the project explorer window should look similar to figure 2.

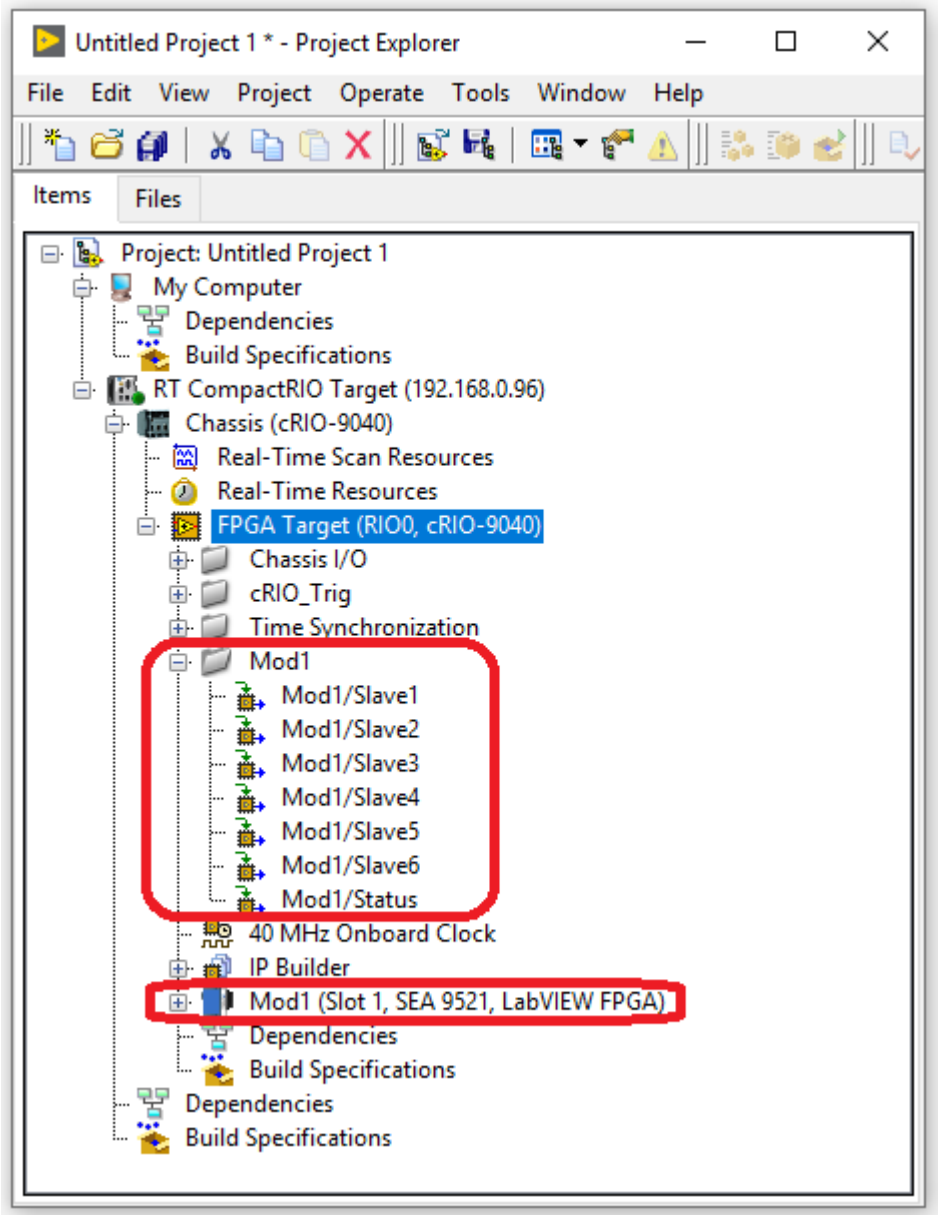


Fig. 3: Quick Start – Project explorer after completing the configuration

The project is now fully configured and the SEA 9521 resources can be used in the user FPGA application.

You can continue from here to learn how to implement a simple FPGA application using the SEA 9521 module.

11. Create a new FPGA VI in the project explorer window. For this right-click on the *FPGA Target (RIO0...)* and select *New –› VI*. Refer to figure 4 below:

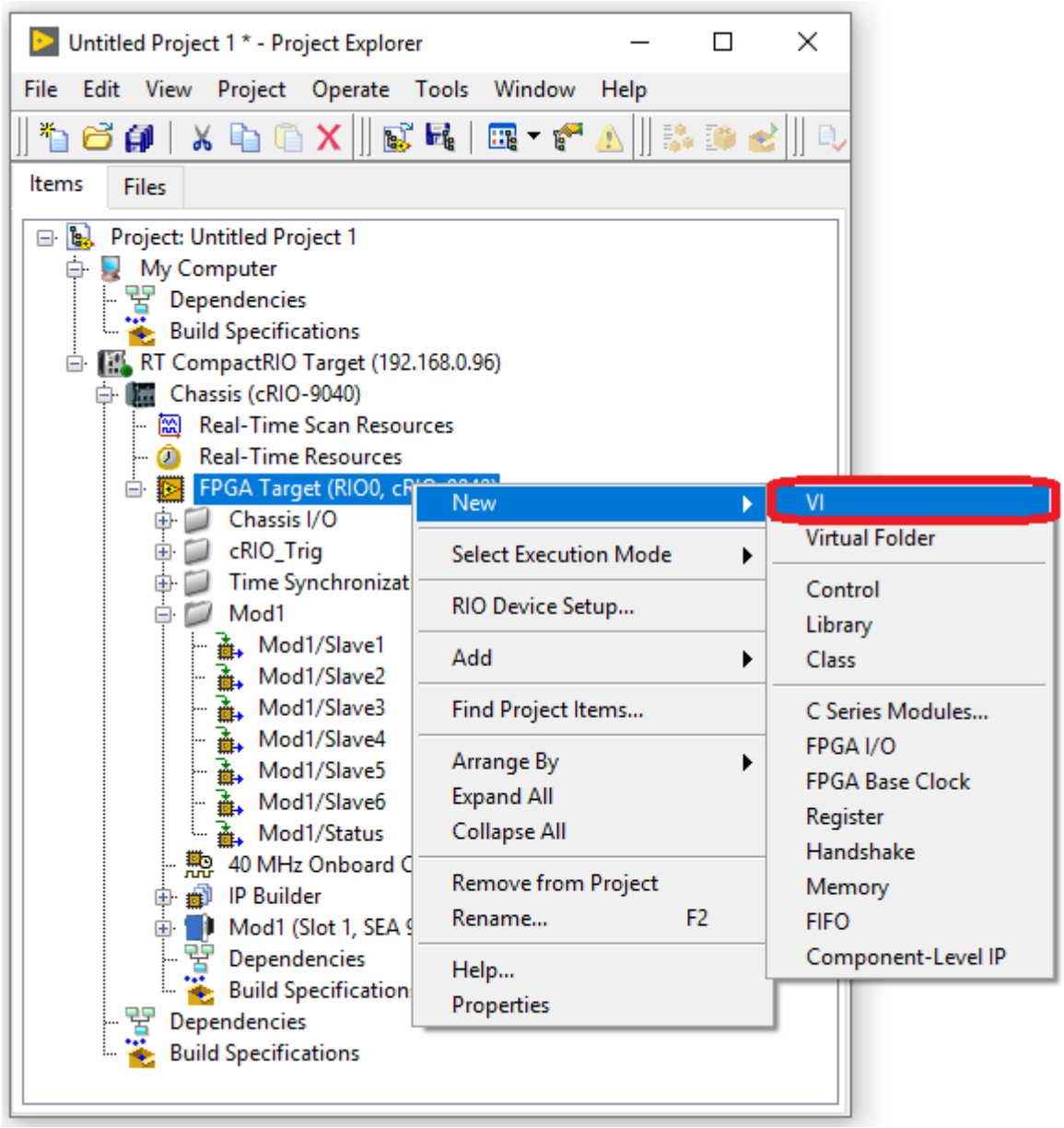


Fig. 4: Quick Start – Add new FPGA VI

12. Open the block diagram of the VI just created and place a method node from the functions palette (*FPGA IO –› IO Method*) on the block diagram. Afterwards select e.g. the item *Mod1* (depending on the cRIO slot used).
13. Finally select the method to be executed from the list of available methods for the selected item. For this tutorial, select the *Init* method as shown in Fig. 5.

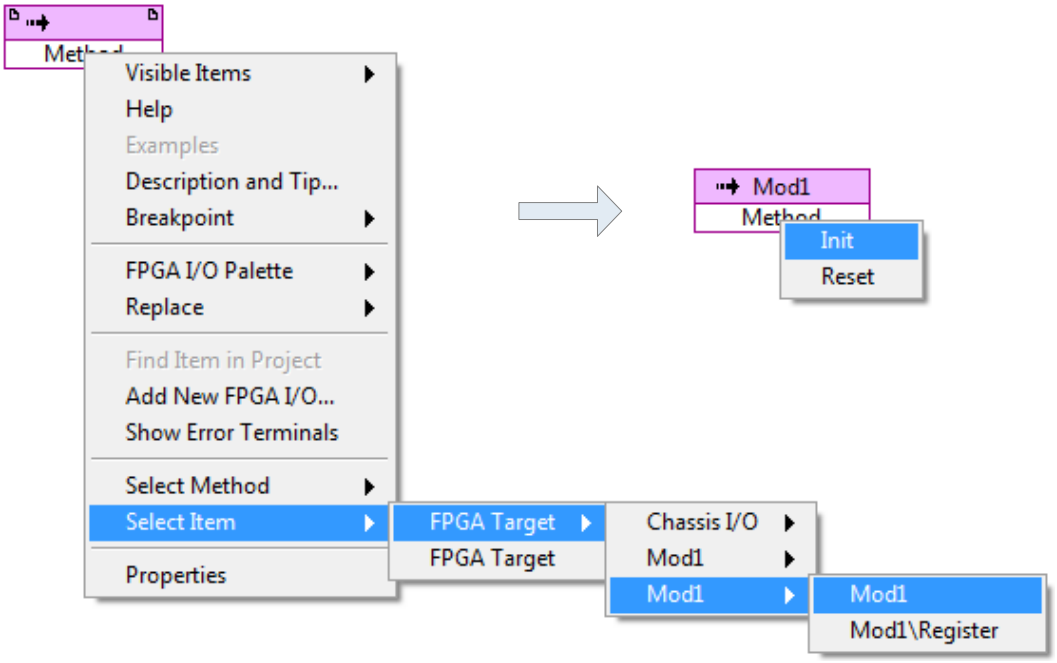


Fig. 5: Quick Start – Create initialization method

14. Choose *Select a VI...* from the block diagram context menu.
Select *SEA-9521_InitParamsGenerate_Standard.vi* from the folder
`../LabVIEW 20XX/examples/SEA/SEA 9521/Shared/API`
and place it on the block diagram. Create a constant (or a control) at the *Standard Module Config* terminal, enter your encoder type and the singleturn/multiturn bit resolution.

    Connect all terminals as shown in Fig. 6. Do not forget to connect *Slave* and *Channel Config*. The standard initialization VI will assume a standard CRC check for BiSS encoders, 1 error and 1 warning bit.

    Create a bus speed constant and set it to 2 MHz or 800 kHz (not too high).

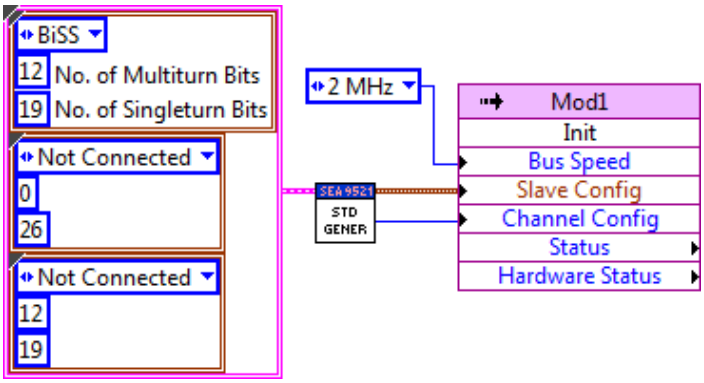


Fig. 6: Quick Start – Initialization parameters

15. Repeat steps 12 and 13 with a FPGA IO node and select the *Slave1* input (*Slave1* represents the first device on channel 1) to read the BiSS/SSI data from the encoder attached to channel 1. Create an indicator for the data input and place a while loop around the IO node. Add a flat sequence to ensure initialization before data acquisition, see Figure 7.

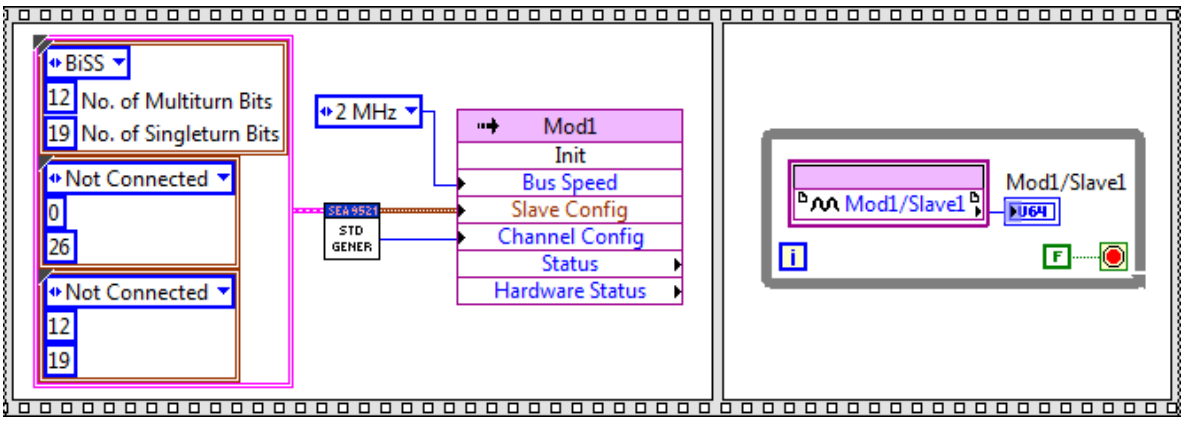


Fig. 7: Quick Start – Final block diagram

16. Save the created FPGA VI, create a build specification and compile it.
17. The *Mod1/Slave1* indicator will display the BiSS-C/SSI data word which was read from the encoder attached to channel 1. It contains the singleturn (ST) bits, the multiturn (MT) bits, and optional error and warning bits.
    In the shown example, the data length will sum up to 12+19+1+1=33 bits or 12+19=31 bits if no error and warning bits exist (SSI). In standard BiSS encoders, MT will be the most significant bits (MSBs), and the warning bit will be the least significant bit (LSB).

All BiSS functions are accessible via either IO, property or method nodes. For a complete list of function nodes, please refer to section 6.2 *Application Programming Interface (API)*.

5 Functional Overview

5.1 cRIO Platform

The SEA 9521 is a cRIO compatible module that can be used in a wide range of carriers from National Instruments. All CompactRIO systems with a programmable (FPGA) backplane are currently supported. SEA 9521 is not supported in systems without a FPGA programmable backplane like CompactDAQ.

5.2 BiSS/SSI Interface

The SEA 9521 module provides unidirectional BiSS and SSI support (actuators are not yet supported).The unidirectional BiSS protocol is a serial interface protocol for the synchronous, fast and safe exporting of sensory data. The SEA 9521 module supports up to three BiSS devices connected directly to the three data channels in point-to-point configuration. Each device may contain multiple slaves. In total, up to six slaves are supported. Queued (or Daisy-Chained) devices in bus configuration are not yet supported by the SEA 9521 module. More details on the BiSS Interface can be found on the Open Source website: https://www.biss-interface.com/

The BiSS interface is hardware compatible with the Synchronous Serial Interface (SSI). It is therefore possible to attach SSI devices and/or BiSS devices to the same SEA 9521 module. As there can only be set a single bus speed for all three channels, the speed of the slowest devices (usually SSI devices if connected) will determine the maximum speed. Hence, if high speed BiSS/SSI data transfer is desired, slower BiSS or SSI devices need to be connected to a second or third SEA 9521 module in different cRIO slots. The higher the bus speed, the higher the maximal possible data acquisition rate.

5.2.1 BiSS: Continuous Mode (C-Mode)

The SEA 9521 module supports the BiSS continuous mode (C-mode) where sensor data can be transmitted continuously while concurrently sending control data at a lower speed. This control data may comprise status data, device parameters, temperature values or configuration parameters. Some devices additionally provide electronic data sheets (BiSS EDS) which can also be read by the SEA module driver.

5.2.2 Electronic Data Sheet (BiSS EDS)

Via register communication BiSS-C compliant electronic data sheets (EDS) can be read from the devices. Details can be found the BiSS website: http://www.biss-interface.com/ in the Downloads section (keyword: BiSS EDS). The parameters read can then be used for encoder configuration. In principle, an automatic configuration can be performed. However, due to the non-standardized character of the BiSS Interface and the many different manufacturers and encoders, an automatic initialization can currently not be recommended and is no part of the SEA 9521 package.

5.3 SEA 9521 Functionality

The SEA 9521 cRIO module allows communication between the CompactRIO backplane and BiSS/SSI encoders.

The SEA 9521 cRIO module is suitable for a wide variety of applications, including position monitoring and position control loops. The BiSS technology offers a serial differential bus protocol with added cyclic redundancy check (CRC) for secure transmission of positional and register data. The module is capable of capturing three independent channels simultaneously. Cable delays are considered and measured continuously to guarantee equal signal propagation delays. Position values are read with FPGA reliability and speed.

Summary of SEA 9521's abilities:

- offers 3 independent channels to drive 3 devices (and up to 6 slaves) in point-to-point (PtP) configuration
- supports unidirectional BiSS-C

- supports SSI Gray Code as well as SSI Binary
- configurable BiSS/SSI bus speed ranging from 200 kHz up to 8 MHz
- integrated line delay compensation
- full access to encoder memory (register communication)
- continuous register access without disturbing positional data acquisition (block reads with sizes of up to 64 bytes are possible)
- CRC-secured data communication

All functions are accessible as nodes within the FPGA programming level.

In the current driver version, it is NOT POSSIBLE to attach BiSS/SSI devices to arbitrary channels!

It is IMPERATIVE to attach the first device to channel 1 (upper connector), the second device to channel 2 (middle connector) and the third device to channel 3 (lower connector). If a channel with a lower number is left unconnected, the other channels WILL NOT WORK.

Likewise, if a device attached to a channel malfunctions or is disconnected, the other channels will stop working as well. Only after proper reconnection and re-initialization, the data transfer will work again.

The maximum data acquisition rate is dependent on many factors such as:

- bus speed
- number of slaves/devices attached
- encoder measurement and processing delays
- line propagation delays

Only approximate values can be given. If the maximum bus speed of 8 MHz is possible (which is limited by the cable length, see below) and only one (three) BiSS slave(s) with a minimal time delay (refer to encoder's data sheet) is connected, data acquisition rate can be up to 33 (20) kHz.

Some BiSS devices will have a maximum acquisition rate much lower than 33 (20) kHz. Be sure to not exceed the device's maximum rating.

SEA 9521 fully supports MXI-Express RIO, R Series Expansion, and EtherCAT Chassis.

Every additional slave will increase processing time by about 8 microseconds. The exact processing time can easily be determined by placing the IO node with all necessary data channels in a while loop and measuring the loop time.

The possible cable length is also dependent on many factors such as:

- bus speed
- electro-magnetic noise (cable shielding)
- encoder/line driver hardware, etc.

In principle, cable length could easily be up to 100 m or more even at maximum bus speed (due to the line delay compensation), but *S.E.A Datentechnik GmbH* cannot warrant any cable length larger than 10 m for any given setup. The safe cable lengths have to determined on an application specific basis.

Latency of BiSS encoder sampling after receiving a trigger

When calling the IO Node (see chapter 6) it takes the SEA 9521 module 95 ns (±10ns) to pull the BiSS Master Clock line low for the first time. This usually is the signal for the encoder to sample its position.

So the maximum delay is a sum of:

- Delay between a trigger and IO Node call (~25 ns, if the delay is 1 clock tick of a 40 MHz FPGA)
- Delay between IO Node call and BiSS Master Clock pulled low (95 ns ±10 ns)
- Cable delay (usually neglectable)
- Encoder delay between receiving the first flank high-low of the master clock and sampling (please refer to the specifications of the encoder manufacturer)

5.4 Power Up Behavior

At power up, a hardware reset of the SEA 9521 module is executed. However, before reading BiSS or SSI data from the attached encoders, an initialization has to be performed using the *Init* method node as described in section 6.2 *Application Programming Interface (API)*.

6 Programming

6.1 Examples

The SEA 9521 driver software is delivered with a set of examples that demonstrate specific aspects of the module. Please refer to the examples to learn how to program the module and how to retrieve position values.

The examples are available via the example finder. Use search keywords like *BiSS, SSI* or *9521* to find the related examples.

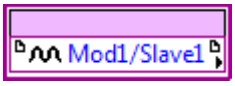
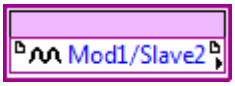
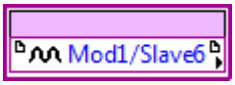
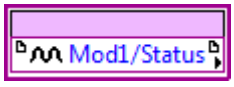
6.2 Application Programming Interface (API)

The Application Programming Interface (API) provides access to the underlying BiSS protocol functionality. This interface exposes nodes that can be used within a LabVIEW FPGA application. The following different node types are available:

- IO nodes
- Property Nodes
- Method Nodes

6.2.1 IO Nodes

For each slave, the IO nodes deliver the acquired BiSS/SSI data (U64) and a general status word providing information about the BiSS/SSI data transfer. The exact length and contents of the BiSS/SSI data is defined by the device or rather the device manufacturer. Rotary encoders e.g. will usually transmit multiturn and singleturn bits as well as 1 error and 1 status bit. The exact meaning of the data words have to be deduced from the device's manual or data sheet. To achieve valid data transfers, only the data length as well as the CRC polynomial used (if any) is important.

Node	Direction	Description
	read only	Retrieves the BiSS/SSI data word of the first slave (slave1) attached to channel 1.
	read only	Retrieves the BiSS/SSI data word of the second slave (slave2) usually attached to channel 2.
...	read only	...
	read only	Retrieves the BiSS/SSI data word of the sixth slave (slave6) which is usually attached to channel 3 – if at all.
	read only	Status information about the BiSS/SSI data transfer for all slaves. For details, refer to chapter 6.3.3 *Status*.

Tab. 3: SEA 9521 IO Nodes

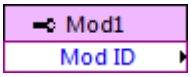
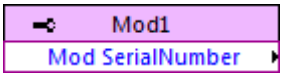
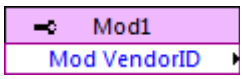
It is highly recommended to read all slave data and the status information using a single IO node in order to ensure proper synchronization between all devices.

Line propagation delay is taken into account by propagation delay compensation featured by the SEA 9521 module. When executing the IO node, a hardware trigger signal is sent synchronously to all slaves. After reception, the slaves will sample their respective positions and will return the data.

The exact sampling time may slightly differ if there are different types of devices attached (with different sampling delays) and/or if the cable lengths of the attached devices largely differs.

6.2.2 Property Nodes

The property nodes deliver information about the module type and identity.

Node	Direction	Description
	read only	Retrieves the module ID from the module. For SEA 9521 module, it retrieves 0069 (0x0045).
	read only	Retrieves the serial number from the module. The serial number is an 8 digit decimal or BCD number.
	read only	Retrieves the Vendor ID. This value identifies the manufacturer of the module. S.E.A. modules always return 0x4711.

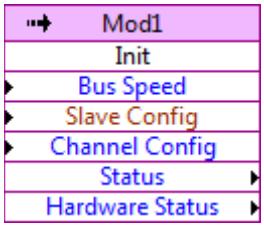
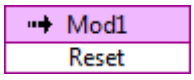
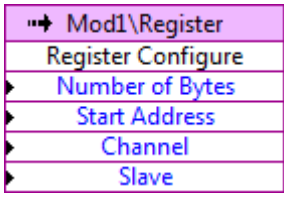
Tab. 4: SEA 9521 Property Nodes

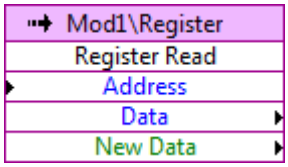
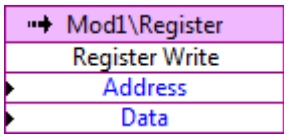
To reduce FPGA slice usage and compile time, the property node functionality can be disabled like shown in the examples, refer also to the section 6.5 *FPGA Usage and Optimization*.

Using property nodes will force a re-initialization if the module's IO nodes were already initialized. Reading IO nodes after reading a property node will throw error #358600.

6.2.3 Method Nodes

Method nodes provide mandatory initialization as well as auxiliary functionality. The register communication is primarily handled using method nodes, for example.

Name	Direction	Description
	read and write	This method node is **mandatory** and needs to be called before reading any data from the attached encoders (this includes register communication data). This node requires some input parameters. They are described in greater detail in section 6.2.4. The returned status information will help to detect any initialization errors.
	—	This method node resets the driver (e.g. if an irretrievable error occurs). After reset, the *Init* method node has to be called again in order to re-establish proper data transfer.
	write only	The *Register Configure* method can only be executed after the mandatory *Init* method call. A register block read starting from *Start Address* to *Start Address + Number of Bytes – 1* on slave *Slave* listening on channel *Channel* can be configured. For more details on the register communication, see section 6.2.6. In the current driver version, the register configuration cannot be run concurrently with IO node calls.

Name	Direction	Description
	read only	After a successful register transfer of the previously configured register block, the registers can be read out using this method. For more details on the register communication, see section 6.2.6. The register read method node is the only method node that can be executed in a concurrent while loop in parallel with the IO nodes without disturbing the BiSS/SSI data transfer.
	write only	This method node executes a register write to a configured register block. For more details on the register communication, see section 6.2.6. In the current driver version, the register write method cannot be run concurrently with IO node calls.

Tab. 5: SEA 9521 Method Nodes

6.2.4 Initialization

The *Init* method node is essential to receive data words from the BiSS or SSI devices. Calling this method will:

1. Set the desired BiSS/SSI bus speed (default: 2 MHz)
2. Set the individual BiSS/SSI data length for every slave
3. Set the individual CRC polynomial (default: 0x43 = x^6+x+1) for every slave
4. Select the CRC mode: inverted (default) or non-inverted
5. Select the channel type: BiSS-C (default) or SSI
6. Select the SSI slave mode: SSI Binary (default) or SSI Gray
7. Select the slave to channel mapping
8. Return the initialization status information (for details, cf. section 6.3.3)

To simplify initialization, the SEA 9521 driver software offers example VIs called *SEA-9521_InitParamsGenerate_Standard.vi* or *SEA-9521_InitParamsGenerate _Advanced.vi* from `LabVIEW 20xx/Examples/SEA/SEA 9521/Shared/API` folder. The appropriate usage can be seen in the SEA 9521 examples, cf. section 6.2.5.

Due to the plethora of different BiSS device manufacturers and BiSS devices, there is no automatic device detection mechanism implemented in the SEA 9521 software package. The current BiSS standard is rather a recommendation than a genuine standard and possesses many preliminary elements. S.E.A. Datentechnik GmbH cannot warrant the correctness of the Electronic Data Sheet (EDS) contents which would be a prerequisite for an automatic configuration. The EDS and its contents solely lies in the device manufacturer's responsibility.

For a proper initialization, the following parameters need to be set:

a) Bus Speed:

Simply right-click on the input and choose *Constant* or *Control* to get the default enumeration. You can then choose between 19 different bus speed setups.

The allowed maximal bus speed of SSI devices is usually much lower than 8 MHz and can be obtained from the manufacturer's data sheet. To obtain reliable results, the maximal bus speed **MUST NOT BE EXCEEDED!** As SSI encoders usually do not have a CRC, resulting errors are hard to detect!

b) Slave Config:

Each U16 value configures one slave starting from slave1 to slave6. The easiest way is to use the example VIs *SEA-9521_InitParamsGenerate_Standard.vi* or *SEA-9521_InitParamsGenerate _Advanced.vi* from the examples to facilitate configuration. Alternatively own parameter generation can be implemented.

Structure (recommended values are bold):

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Lower U8	SSI GRAY	ENABLE	BiSS/SSI DATA LENGTH – 1					
Upper U8	INVERTED	CRC POLYNOMIAL BITS **1**:7						

SSI GRAY (only valid for SSI slaves)	
=0: SSI Binary	=1: SSI Gray Code

ENABLE	
=0: disable data transfer	**=1:** enable BiSS/SSI data transfer

BiSS/SSI DATA LENGTH – 1
The data length is device dependent and can either be extracted from the device's data sheet or by contacting the device manufacturer. For rotary encoders, the data length typically consists of the singleturn and the multiturn resolution and the number of error and warning bits.

INVERTED	
=0: no inversion	**=1:** inversion

If the CRC polynomial has been inverted (as in most cases), this bit has to be set. The device's data sheet should provide information about the CRC inversion.

CRC POLYNOMIAL BITS 1:7
Bits 1–7 of the CRC polynomial have to be entered here. Bit 0 (LSB) is simply omitted.

Example:

A single BiSS-C rotary encoder with

- 12 bits multiturn resolution,
- 19 bits singleturn resolution, and
- 1 error and 1 warning bit

needs to be connected.

Consequently, the data length sums up to 19 + 12 + 1 + 1 = 33 bits.

The CRC polynomial is given in the device's data sheet as 0x43, inverted which is equal to:

$x^6 + x + 1 = \mathbf{1}*x^6 + \mathbf{0}*x^5 + \mathbf{0}*x^4 + \mathbf{0}*x^3 + \mathbf{0}*x^2 + \mathbf{1}*x^1 + \mathbf{1}*x^0 =$

100 0011 bin = 0x43 hex

Hence, Bits 1:7 (LSB is omitted) of the polynomial are:

10 0001 bin = 0x21 hex = 33

Parameters:

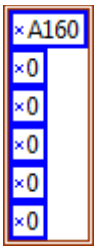
SSI GRAY			= 0
ENABLE			= 1
BiSS/SSI DATA LENGTH – 1	= 33 – 1	= 32	= 10 0000 bin
INVERTED			= 1
CRC POLYNOMIAL		= 33	= 010 0001 bin

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Lower U8	0	1	1	0	0	0	0	0
Upper U8	1	0	1	0	0	0	0	1

As a result, the *Slave Config* parameter of slave1 is

1010 0001 0110 0000 bin = 0xA160 hex

and the *Slave Config* cluster constant should look like this (only slave1 is used):



In the current driver version, it is NOT POSSIBLE to configure and use slaves in arbitrary order. Slave1 has to be used first (and be attached to channel 1), only then slave2 can be used and so on. Refer to the SLAVELOC parameter on pages 22ff. for more details.

c) Channel Config:

The Channel Config describes the properties of the three data channels. It determines which slave is connected what kind of slaves (BiSS-C or SSI) are attached. The easiest way is to use the example VIs *SEA-9521_InitParamsGenerate_Standard.vi* or *SEA-9521_InitParamsGenerate _Advanced.vi* from the examples to facilitate configuration. Alternatively own parameter generation can be implemented.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte1	SLAVELOC							
Byte2	x[1]	x	SELSSI3	BISSMOD3	SELSSI2	BISSMOD2	SELSSI1	BISSMOD1
Byte3	x	x	x	x	x	x	x	x
Byte4	x	x	x	x	x	x	x	x

[1] *("x" designates don't care bits)*

SELSSI1, SELSSI2, SELSSI3	
=0: ch1, ch2, ch3 are BiSS channels	=1: ch1, ch2, ch3 are SSI channels

BISSMOD1, BISSMOD2, BISSMOD3	
=0: BiSS-B devices on ch1, ch2, ch3	**=1**: BiSS-C devices on ch1, ch2, ch3

SLAVELOC		
Bit	=0	=1
0	not allowed	Slave1 on ch1
1	Slave2 on ch1 or *NC*[1]	Slave2 on ch2
2	Slave3 on the same channel as Slave2 or *NC*	Slave3 on the next channel
3	Slave4 on the same channel as Slave3 or *NC*	Slave4 on the next channel
4	Slave5 on the same channel as Slave4 or *NC*	Slave5 on the next channel
5	Slave6 on the same channel as Slave5 or *NC*	Slave6 on the next channel

[1] *NC = not connected*

Example:

- An SSI device with a single slave (Slave1) is connected to channel 1 (ch1).
- A BiSS-C device with two slaves (Slave2 and Slave3) is connected to channel 2 (ch2).
- A BiSS-B device with one slave (Slave4) is connected to channel 3 (ch3).

Parameters:

SELSSI1	= 1	(channel 1)
SELSSI2	= 0	(channel 2)
SELSSI3	= 0	(channel 3)
BISSMOD1	= 0 or 1 (don't care)	(channel 1)
BISSMOD2	= 1	(channel 2)
BISSMOD3	= 0	(channel 3)

SLAVELOC		
Bit	=0	=1
0	---	Slave1 on ch1
1	---	Slave2 on ch2
2	Slave3 on ch2 as well	---
3	---	Slave4 on ch3
4–7	not connected	---

SLAVELOC = 0000 1011 bin

Channel Config should look like this:

	Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Byte1	0	0	0	0	1	0	1	1
Byte2	x[1]	x	0	0	0	1	1	0 (x)
Byte3	x	x	x	x	x	x	x	x
Byte4	x	x	x	x	x	x	x	x

[1] *("x" designates "don't care" bits)*

Assuming that all don't care bits are 0, Channel Config must be set to:

0000 0000 0000 0000 0000 0110 0000 1011 bin = 0x60B hex = 1547

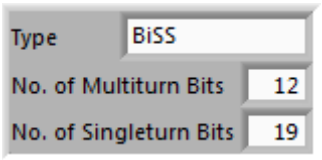
There exist two examples that get position (with and without cRIO RT host integration) which can be found via the example finder looking for the keyword *9521*. In this examples, the proper initialization of one or more encoders can be studied.

6.2.5 Simplified Initialization

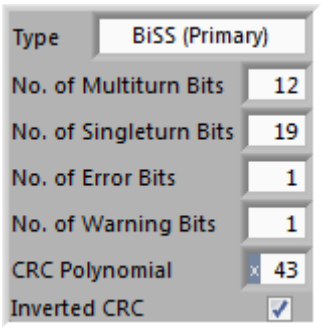
To avoid setting *Slave Config* and *Channel Config* (see pages 20ff) manually, the SEA 9521 package offers exemplary generation VIs called *SEA-9521_InitParamsGenerate _Standard.vi* and *SEA-9521_InitParamsGenerate _Advanced.vi* which are also showcased in the GetPosition examples (see section 6.1).

	Useful for standard BiSS/SSI encoders. In a FPGA target, this VI will slightly increase FPGA usage.
	Useful for customized BiSS/SSI encoders. In a FPGA target, this VI will moderately increase FPGA usage.

After instantiating the VI, create default constants or controls for the respective *Standard* or *Advanced Module Config* input using the context menu.



- The *Standard Module Config* allows choosing the encoder type such as BiSS, SSI Binary, or SSI Gray Code and the number of multiturn and singleturn bits. For BiSS encoders, it assumes a single error and a single warning bit and the standard CRC polynomial of 0x43, inverted. For SSI encoders, it assumes no error and warning bits and no CRC (CRC polynomial = 0x0)



- The *Advanced Module Config* additionally allows choosing the number of error and warning bits as well as the CRC polynomial and the CRC inversion mode. Furthermore, it supports devices containing more than one slave. This can be necessary in safety applications where a secondary slave is used for verification of the primary slave's data input. In such a case, use the *BiSS (Primary)* and *BiSS (Secondary)* selection for the first and the second slave.

The *Slave Config* and *Channel Config* outputs have to be directly connected to the appropriate *Init* method node inputs like shown in the examples.

6.2.6 Register Communication

Register communication is only possible when using BiSS-C compliant devices with accessible registers. After successfully calling the *Init* method node, a register communication can be configured before the BiSS (positional) data transmission can be started. Therefore, the *Register Configure* method node needs to be called with the appropriate input parameters. The method node needs the starting register address and the number of register bytes to be transmitted. Additionally, a single channel and a single slave number have to be selected.

In the current driver version, it is not possible to read register data from different channels and slaves at the same time. A single channel and slave have to be selected prior to transmitting the requested data.

In the current driver version, calling the *Configure Register* method node will temporarily prevent the IO node from executing. Hence,

- IT IS NOT possible to change the register read configuration while synchronously receiving BiSS (positional) data. If registers from different slaves have to be read, BiSS (positional) data acquisition has to be temporarily interrupted while switching the register communication channel.
- IT IS POSSIBLE, however, to read register data synchronously to BiSS (positional) data AFTER the configuration has finished.

After configuration, the registers can be read using the *Register Read* method node. It will always read Byte (U8) values from a given address (which has to be in range of the previously configured register address range). The boolean output *New Data* of the node will indicate (value equals true) the arrival of a new register data value. It will be set to *false* after a successful read.

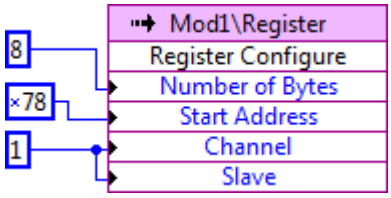
New register data will only become available if IO node calls (requesting BiSS/SSI data) are executed in parallel. The BiSS-C protocol transmits register data bit-wise alongside the BiSS/SSI data words. Hence, many IO node calls have to be performed before the register data words become available. As a result, the register data rate is much lower than the BiSS/SSI data rate.

Example:

From Slave1 on channel 1, the BiSS Device ID and Manufacturer ID which according to the BiSS-C recommendations can usually be found in registers 0x78 – 0x7F (8 bytes) needs to be read.

After initialization, the *Register Configuration* method node is called with the following parameters:

Number of Bytes	= 8
Start Address	= 0x78 (= 120)
Channel Select	= 1
Slave ID	= 1



Afterwards, IO node method calls need to be initiated in a high speed while loop whereas a concurrent (slower) while loop calls the *Register Read* method node with addresses ranging from 0x78 to 0x7F (single U8 values will be read).

Every time *New Data* returns true, a new set of register values has arrived.

There exists a *RegisterCommunication* example which can be found via the example finder looking for the keywords *9521* or *BiSS*. It elaborates the concepts and schemes outlined in this section. The Electronic Data Sheet example showcases the register write.

6.3 IO Nodes: Data Formats

The IO nodes will retrieve BiSS/SSI data of all the different slaves as well as status information about the BiSS/SSI data transfer. The register data values will be transmitted bit-wise alongside the BiSS/SSI data if a register communication was previously configured. Hence, many IO node calls are necessary to perform a single register value transfer, cf. section 6.2.6.

In the current driver version, the three data channels are not completely independent from each other. If any slave on any channel malfunctions and stops communicating, all other slaves and channels will stop communicating as well. In this case, the damaged device has to be replaced or repaired and the initialization phase has to be repeated.

It is NOT POSSIBLE to run a device on channel 2 if channel 1 is not used and also impossible to run a device on channel 3 if channel 2 is not used.

6.3.1 BiSS/SSI Data (Single Cycle Data)

The BiSS/SSI data will usually contain positional information such as multiturn (MT) and singleturn (ST) as well as error (sometimes called alarm) and warning bit(s).

The actual contents of the transmitted BiSS/SSI data SOLELY LIES in the DEVICE MANUFACTURER's hands and is, of course, device-dependent.

S.E.A. Datentechnik GmbH can, thus, only give some guidelines on how to interpret the received data.

If open questions regarding the BiSS/SSI data contents remain, please contact the device manufacturer or study the device's data sheet.

The BiSS/SSI data words received will always have a size of 8 bytes (U64) and will in standard cases look like this:

(MSB)	MT bits (if applicable)	ST bits	1 error bit (BiSS) no error bit (SSI)	1 warning bit (BiSS) no warning bit (SSI)	(LSB)

For example, if MT = 12 and ST = 25, the data length containing relevant information will add up to

12 + 25 + 1 + 1 = 39 bits (BiSS) or
12 + 25 + 0 + 0 = 37 bits (SSI).

BiSS: Bit 0 will be the warning bit, bit 1 the error bit, bits 2 – 26 the singleturn bits and bits 27 – 38 will be the multiturn bits.

SSI: Bits 0 – 24 will be the singleturn bits and bits 25 – 36 will be the multiturn bits.

In this case, bits 39 – 63 (BiSS) or bits 37 – 63 (SSI) will not contain any useful information and should be masked out to avoid any disturbances.

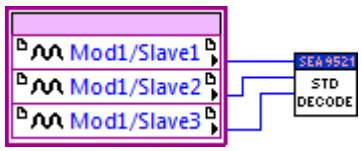
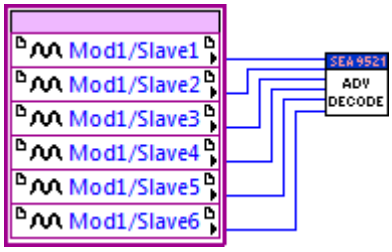
In many cases, SSI devices will not have a CRC for error detection.

6.3.2 Simplified BiSS/SSI Data Decoding

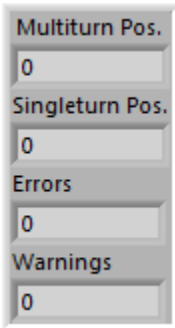
Separating the BiSS/SSI data words into multi- and singleturn as well as error and warning information is application-specific and can be tedious. Depending on the encoder used, there might be different numbers of error, warning, multi- and singleturn bits. To simplify decoding, the SEA 9521 software package, offers two example VIs called

- *SEA-9521_SlaveDataDecode_Standard* and
- *SEA-9521_SlaveDataDecode_Advanced*

in the `LabVIEW 20xx/Examples/SEA/SEA 9521/Shared/API` folder.

Example VI	Description
	Useful for standard BiSS/SSI encoders. • This VI will only work, if the *SEA-9521_InitParamsGenerate _Standard.vi* was called beforehand, refer to section 6.2.5. • In a FPGA target, this VI will slightly increase FPGA usage.
	Useful for customized BiSS/SSI encoders. • This VI will only work, if the *SEA-9521_InitParamsGenerate _Advanced.vi* was called beforehand, refer to section 6.2.5. • In a FPGA target, this VI will slightly increase FPGA usage.

The VIs will return output clusters for every slave which contain U64 values for multi- and singleturn and U8 values for errors and warnings. The standard BiSS protocol assumes error and warning bits being active low. The error and warning bits received are negated (inverted) during the decoding process before being converted to U8 values. Hence, *Errors* and *Warnings* will be equal to zero if no error or warning occurred.



The decoding VIs will make the following assumption:

- BiSS/SSI data words have standard format without any alignment bits (see section 6.3.1)

Only the standard decoder will make the following additional assumption:

- Exactly one error and one warning bit for BiSS encoders
- No error and no warning bit for SSI encoders

If the encoders do not meet these requirements, decoding has to be customized.

To decrease FPGA usage, customized decoding with constants and fixed array sizes will be favorable. Refer to the register communication example for suggestions.

Depending on the device manufacturer's choice, error and warning bits may be active high which will lead to non-zero values if no error or warning occurred.

6.3.3 Status

The Status is a 32 bits double word (U32) that provides operational information about the BiSS/SSI data and register transfer or about the initialization phase.

The format is:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte1 (LSB)	nERR	nWDERR	1	nCRCERR	nREGERR	REGEND	1	EOT
Byte2	SVALID4	0	SVALID3	0	SVALID2	0	SVALID1	0
Byte3	0	0	0	0	SVALID6	0	SVALID5	0
Byte4 (MSB)	0	0	0	0	0	0	0	0

Name	Value	Description
EOT End-Of-Transmission	0	At least one of the slaves did not answer (timeout). This **severe** error will occur if a slave malfunctions or if the hardware connection is damaged. Error code 358603 will be thrown which will enforce a re-initialization of the data connection (using the *Init* method node).
	1	All data values arrived in time
nERR (not) error	0	Accumulated error flag: either a CRC error, a watchdog error or a register error occurred
	1	No error
REGEND	—	The REGEND status is related to register communication and indicates whether a register data transmission has been completed or not. This status is only relevant for module-internal processes.
nCRCERR (not) CRC error	0	At least one of the BiSS/SSI data values received did not pass the cyclic redundancy check (CRC). The SVALID bits will specify the error location (slave ID). The respective data value received cannot be trusted.
	1	No CRC error
nWDERR (not) Watchdog error	0	a) The data acquisition rate is too high. At least one sensor could not answer fast enough. Reduce data acquisition rate, or b) another error leading to a too slow slave response occurred. A re-initialization is necessary to delete this error flag.
	1	No watchdog error
nREGERR (not) register error	0	An error during register communication occurred. Check if the slave connected is capable of BiSS-C compliant register communication
	1	No register error
SVALIDx (x = 1...6 = slave ID)	0	Slave 'x' DID NOT send a valid BiSS datum.
	1	Slave 'x' did send a valid BiSS datum.

Tab. 6: Status Information

Examples:

1. Correct data transmission (without register communication or register communication is still unfinished) while three BiSS slaves (with CRC) are attached:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte1 (LSB)	1	1	1	1	1	0	1	1
Byte2	0	0	1	0	1	0	1	0
Byte3	0	0	0	0	0	0	0	0
Byte4 (MSB)	0	0	0	0	0	0	0	0

0010 1010 1111 0011 bin = 0x2AFB hex

2. Same case, but CRC of second slave fails and a register communication with e.g. slave1 has finished and was successful:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte1 (LSB)	0	1	1	0	1	1	1	1
Byte2	0	0	1	0	0	0	1	0
Byte3	0	0	0	0	0	0	0	0
Byte4 (MSB)	0	0	0	0	0	0	0	0

0010 0010 0110 1111 bin = 0x226F hex

6.3.4 Hardware Status

The Hardware Status after a proper initialization should always return:

0x5555 = 21845

If a different value is displayed, please see the corresponding trouble shooting item in chapter 9.

6.4 Error Codes

In case of unexpected behavior, the driver software returns an error cluster. The option S*how Error Terminals* of a node (right-clicking on the node) must be enabled to obtain error information.

Adding error terminals will increase FPGA usage and compilation time.

The possible error and warning codes are listed in table 7.

Error Code	Reported by	Description
Module Errors		
65536	any node	No module or invalid module type found. Please insert SEA 9521 module into the correct slot. If a SEA 9521 module is recognized as invalid, please contact SEA technical support.
65537	any node	Incorrect module type found. Only SEA 9521 modules will work with this driver version. Please check if all modules are inserted into the right slot.
Encoder Errors		
358600	IO Nodes	BiSS/SSI data transmission is not (yet) properly initialized. Please execute the *Init* method node before reading any data from IO nodes.
358601	*Init*	Initialization of the BiSS/SSI data transmission failed. Check *Status* and *Hardware Status* bytes to retrieve information about possible reasons. If the nCRCERR status bit is low, check the encoder initialization settings (number of multi- and singleturn bits and error and warnings bits as well as CRC information). If the EOT status bit is low, check the hardware connections and the number of devices configured. It is not possible to use channel 2 if channel 1 is not connected. It is not possible to use channel 3 if channels 1 and 2 are not connected.
358602	*Register Configure*	The *Register Configure* method node cannot be called without previously successfully initializing the BiSS/SSI data transfer by calling the *Init* method node.
358603	IO Nodes & *Register Write*	A timeout during data acquisition occurred (EOT status bit is low, see section 6.3.3). At least one device malfunctions or a connection error exists. This error will enforce a re-initialization of the BiSS/SSI data transfer (by calling the *Init* method node). This error will occur only ONCE: every subsequent IO node call will lead to error code 358600 due to the required re-initialization.
358604	*Register Read* & *Register Write*	The register read failed. Possible reasons: 1. Register Communication has not yet been configured: call the *Register Configure* method node 2. Register address out of range: the register read must be in the same address range as has been previously configured The register write failed. Reason: Register Communication has not yet been configured: call the *Register Configure* method node

Error Code	Reported by	Description
358605	all Register nodes	A register communication method node has been called even though the *Conditional Disable Symbol* "REGCOMM" was set to "OFF", cf. section 6.5.
358606	all Property nodes	A module property was accessed even though the *Conditional Disable Symbol* "EEPROM" was set to "OFF" (default), cf. section 6.5.
Encoder Warnings		
358615	*Register Read* & *Register Write*	Register Communication failed. Possible reasons: 1. The attached device is not BiSS-C compliant regarding register access or doesn't feature register communication 2. A CRC error occurred during register communication. Try to read/write again
VI Errors		
358610	Standard Decoder	*SEA-9521_SlaveDataDecode_Standard.vi* was called without previously calling the *SEA-9521_InitParamsGenerate_Standard.vi*
358611	Advanced Decoder	*SEA-9521_SlaveDataDecode_Advanced.vi* was called without previously calling the *SEA-9521_InitParamsGenerate_Advanced.vi*

Tab. 7: Error and Warning Codes

6.5 FPGA Usage and Optimization

The SEA 9521 driver software consumes approx. 25–30% of available FPGA space, when using the NI 9104 backplane. Note that the FPGA space consumption for the driver code is not linear in any way. It varies with the chassis types and number of total modules installed.

However, the current driver software architecture offers a way that can help to reduce the FPGA space further if FPGA space is precious. Within the SEA 9521 driver core certain functional blocks can be disabled using *Conditional Disable Symbols*. A disabled functional block is not compiled into the bitfile and therefore does not consume FPGA space. When using *Conditional Disable Symbols* some API functions become non-executable and must cannot be used. Please refer to the table below for details on using this optimization feature:

Conditional Disable Symbol	Value Range	Description
EEPROM	OFF ON (default)	OFF disables the following nodes: • Mod ID • Mod SerialNumber • Mod VendorID
REGCOMM	ON (default) OFF	OFF disables the following nodes: • Register Configure • Register Read • Register Write

Tab. 8: Conditional Disable Symbols

The *Conditional Disable Symbols* can be defined in the project property dialog box like shown in Fig. 8.

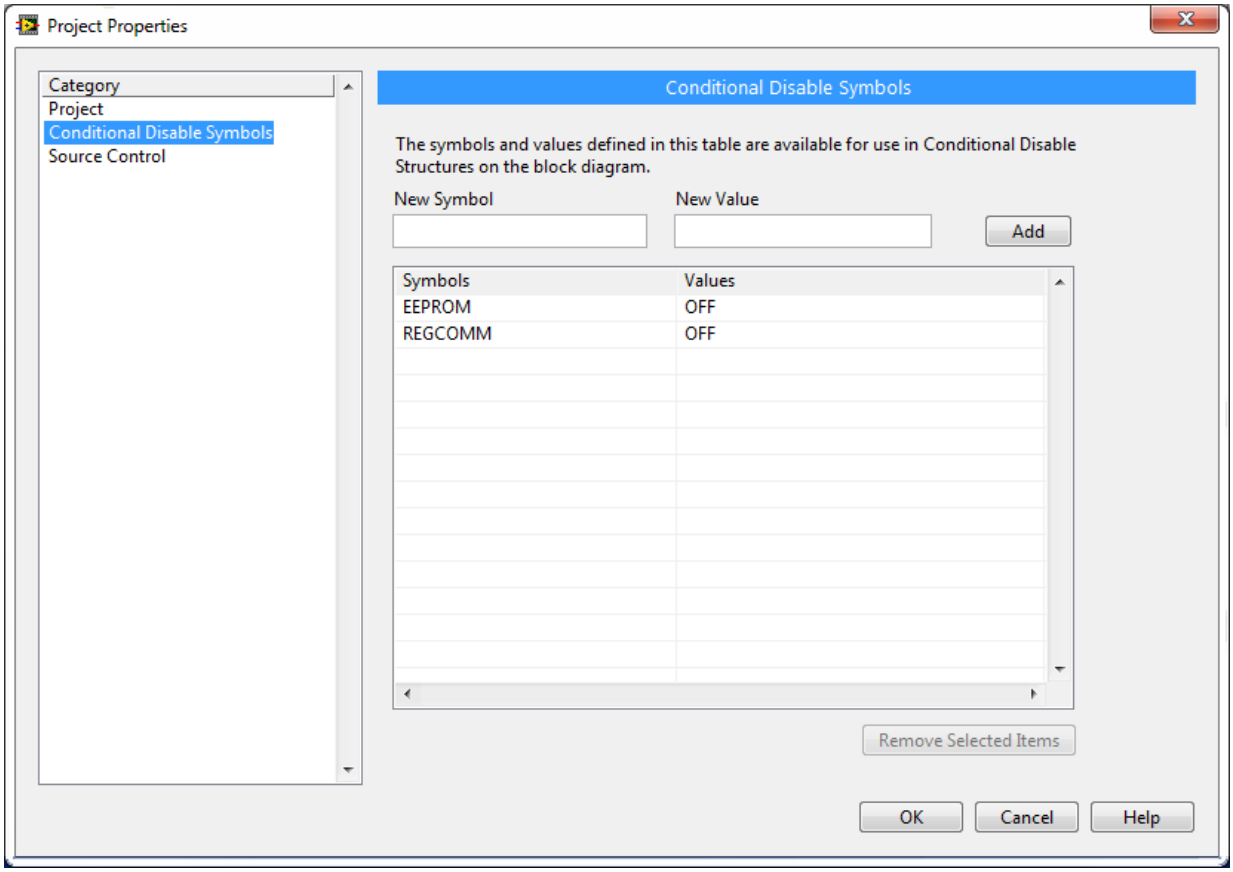


Fig. 8: Conditional Disable Symbols

This dialog box is accessible by right-clicking on the project root item “Project: *‹project name›*” in the LabVIEW project explorer. If no conditional disable symbols are defined the default values are used (EEPROM=ON; REGCOMM=ON).

The init parameter generation and decoding VIs described in sections 6.2.5 and 6.3.2 will increase FPGA usage due to their flexible structure. Using constants and fixed-sized arrays and array manipulation (index array, array subset) will decrease FPGA usage. Refer to the *RegisterCommunication* example for possible suggestions.

7 BiSS Electronic Data Sheet (BiSS EDS)

BiSS-C describes an electronic data sheet (EDS) which can be retrieved by the SEA 9521 software package. There exists a *SEA 9521 ReadElectronicDataSheet* example which can be found via the example finder looking for the keyword *9521*. In this example, the first EDS bank of a BiSS-C compliant device on channel 1 is read.

Only the device manufacturer is responsible for the contents and its accuracy. In principle, it can be used to automatically detect and configure BiSS-C devices attached to the SEA 9521 module.

If another bank of the device register memory has to be read (e.g., the BiSS Profile banks), the example has to be adapted accordingly.

ATTENTION: If the device is not fully BiSS-C compliant, the EDS example may potentially lead to device malfunction if device configuration memory is overwritten unintentionally during the bank switch. If you are unsure about the device, do not run the example.

The BiSS EDS is described on the BiSS Interface website: https://www.biss-interface.com/
In the *Downloads* section, look for the keyword *EDS* to download the current version.

The current SEA 9521 driver version refers to the BiSS EDS description available in January, 2013. Due to the preliminary character of the BiSS EDS description, later versions may potentially differ.

Additionally, device manufacturers may integrate own add-ons or application specific changes which cannot be handled by the SEA 9521 software.

8 Deployment

If a user application containing the SEA 9521 is deployed to a run-time device (i.e. NI CompactRIO) the following parts are necessary to be deployed along with the compiled application (i.e. `rtexe`):

- LabVIEW Run-Time Engine 2017 or higher

9 Trouble Shooting

The following hints might help you to determine if the module and/or software behave correctly and how to identify module failures.

#	Problem	How to solve
1	The module is not detected within LabVIEW	1. Make sure you have the CompactRIO system powered and connected to your PC. 2. Make sure you have installed the current BiSS/SEA 9521 driver software (≥ 1.0.0). 3. If previous driver versions exist, uninstall them properly before installing the current driver.
2	The *Init* method node returns an error	1. Check if you are using fully functional BiSS or SSI encoders 2. Check if your encoder connections are in the correct order. Channel 2 cannot be used without a functional encoder attached to channel 1. Channel 3 cannot be used without functional encoders attached to channels 1 and 2. 3. If you are using self confectioned cables, check the pinout and contact.
3	The *Init* method node only works with low *Bus Speed* setups	Check your cables and the overall cable length. A high speed connection is only possible with high-quality encoders (line drivers) and cables. High electro-magnetic noise might also influence your possible bus speed. High bus speeds are only necessary if a high data acquisition rate is mandatory. Determine the highest possible data acquisition time by placing the IO node with all necessary data channels in a while loop and measuring the loop time. Compare the result with your data rate specifications. The lower the bus speed, the lower the error proneness due to parasitic electro-magnetic coupling.
4	The Hardware Status does not return value 0x5555 (= 21845)	• Check the configuration settings again. It is not possible to use channel 2 if channel 1 is not connected to an encoder. It is not possible to use channel 3 if channels 1 and 2 are not connected. For unused channels select encoder Type *Not Connected*. • Carefully check the wiring and the pin assignments. • Check whether the external voltage supply is sufficient. • Check if the Status LED lights up at the latest when the software is executed.
5	The IO nodes suddenly stop delivering the position data	Check all connectors and encoders. If at least one encoder stops responding properly, the *Init* method node has to be called again in order to recover. If the error occurs again, enable the error terminals in the IO node and indicate the error status. Additionally, check the IO node status information.
6	The IO nodes suddenly throw the error code 358600 (new initialization necessary) even though the previous initialization has been successful	At least one of the devices malfunctions or a problem with the hardware connection exists. The problem described can only happen if a data acquisition timeout occurred (see also problem #5). The appropriate timeout error code 358603 is only thrown ONCE. All subsequent IO node calls throw error code 358600 until a proper re-initialization is performed. Make sure to react to error code 358603 immediately before calling the IO node again. In this case, you can also check the status information for details.

#	Problem	How to solve
7	The retrieved position data seems not to be correct when using a BiSS device	It is highly recommended to use the CRC functionality when driving BiSS devices by entering the correct CRC polynomial during the initialization process. You can then check the nREGERR status bit in the IO node status. If it permanently stays low, either the CRC polynomial used is wrong or – more likely – the data length (which was configured during initialization) is wrong. Check the device's data sheet or contact the device manufacturer to determine the correct data length. If no CRC error exists (nREGERR = 1) you can be sure that the BiSS/SSI data was transmitted correctly. In this case, your handling of the position data might be wrong. Make sure that you 1. properly handle error and warning bits, 2. distinguish properly between singleturn and multiturn, and 3. properly masked out the data bits that are not carrying useful information (they might be different from '0' without apparent reason) Without using the CRC functionality, it is not possible to check the correct data length, cf. problem #8
8	The retrieved position data seems not to be correct when using a SSI device	Make sure to keep bus speed below maximum rating! SSI devices usually don't have a CRC which makes it much more difficult to detect errors. Make sure that the configured data length is correct (if unsure contact the device manufacturer). Be aware that SSI devices will not have error and warning bits in most cases. Make sure that you are using the SSI device in the correct mode. Data can be transmitted in a) Binary or b) Gray code format (see section 6.2.4)
9	A watchdog error occurs in the status word during IO node reads	Data acquisition rate might be too high. Reduce IO node loop execution rate by e.g. introducing FPGA Loop Timers. This might also occur if solely the status information is read without reading any slave data. In this case, simply add a dummy slave input.

Tab. 9: Trouble shooting

A Figures

B Tables