

SDK

Software Development Kit for NKT Photonics Instruments

Instruction Manual



Table of Contents

1	Introduction.....	3
2	Interbus Protocol Description.....	4
2.1	Physical	4
2.2	Telegram	5
2.3	Escape Sequence Conversion	8
2.4	Transmission Example	9
2.5	Reception Example.....	10
3	C++ Source Code.....	12
4	C# Source Code.....	13
5	Labview Driver.....	14
5.1	Inputs and Outputs	14
5.1.1	PortParameters.....	16
5.1.2	InParameters	17
5.1.3	OutParameters.....	18
5.2	Programming Examples	20
5.2.1	Reading Example.....	20
5.2.2	Writing Example.....	21
6	Register Files.....	22
6.1	General Registers.....	24
6.2	Koheras AdjustiK/BoostiK System (K81-1 to K83-1)	25
6.3	Koheras BasiK Module (K80-1)	26
6.4	SuperK EXTREME System (S4x2)	27
6.5	RF Driver (A901) and SuperK SELECT (A203).....	28
6.6	SuperK VARIA (A301).....	29
7	USB Driver.....	30
8	Generic User Interface	33
8.1	Installing the software	33
8.2	Register File Path.....	36
8.3	Using the Generic User Interface Software	36
8.3.1	Front Panel	36
8.3.2	Starting Up.....	38
8.3.3	Controls	39
8.3.4	Readings.....	40
8.3.5	Error and Status.....	41
8.3.6	Graph.....	42
8.3.7	Functions	43
8.3.8	Loops	44

1 Introduction

Introduction

Please take the necessary time to read this software manual, which contains important information about how to get started with the Software Development Kit.

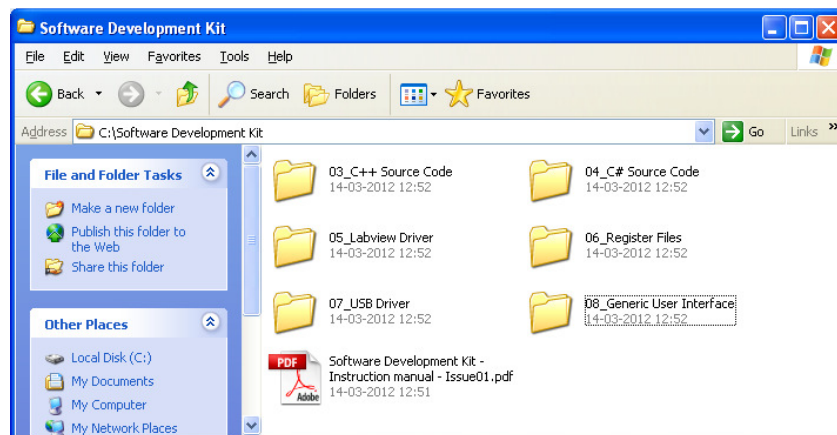
Warning

Laser products are as such potentially dangerous. Read the instruction manual for the laser, amplifier and/or accessory before continuing.



Contents

The Software Development Kit features this manual, C++ source code, C# source code, a Labview driver, register lists, USB driver and a generic user interface.



Manual

This manual covers a description of the NKT Photonics Interbus protocol, description of how to use the C++ and C# source code examples and Labview driver, how to read the register lists, how to install USB driver and how to install and use the Generic User Interface.

Protocol Description

The [Interbus Protocol Description](#) section is a low level description for those who wants to write their own code e.g. for a micro controller in their own system that should be able to communicate with NKT Photonics modules/systems.

C++ Source Code

The [C++ Source Code](#) section describes how to get started with the C++ source code from the 03_C++ Source Code folder.

C# Source Code

The [C# Source Code](#) section describes how to use the C# source code from the 04_C# Source Code folder.

Labview Driver

The [Labview Driver](#) section explains how to use the Labview driver from the 05_Labview Driver folder.

Register Files

The [Register Files](#) section explains how to read use the information from the register files from the 06_Register Files folder. There are separate register files for the different types of modules and systems.

USB Driver	The USB driver for NKT Photonics products with USB interface (e.g. SuperK Extreme and Koheras AdjustiK), the USB driver can be found in the 07_USB Driver folder. The USB Driver section describes how to install the USB driver.
Generic User Interface	The Software Development Kit includes a Generic User Interface. This user interface uses the information from the register files in the 06_Register File folder, so it can be of help to e.g. understand what to write to which registers and to read out what has been written to the different registers. The Generic User Interface section describes how to install and use the Generic User Interface from the 08_Generic User Interface folder.

2 Interbus Protocol Description

This section describes the NKT Photonics Interbus protocol and module hardware integration. It is not to be confused with INTERBUS developed by Phoenix Contact.

2.1 Physical

Physical	<p>The NKT Photonics Interbus standard is based on a 2-wire RS485 interface. Baudrate is 115.2 kbit/s, 8 data bits, 1 stop bit, no parity. In some systems, communication goes through a USB or Ethernet port, but the telegrams are built in the exact same way.</p> <p>When communicating directly with a single module (or a number of modules) on an RS485 bus, there are no hardware handshake signals in use. However, when communicating through a USB, Ethernet or RS-232 port, handshake signals are available. Handshake mode must be set to either Hardware (CTS/RTS) or None. If None is used, please check that the host's RTS signal is set low (logic '0'). Otherwise, equipment may not be able to respond.</p>
----------	---

2.2 Telegram

Framing

A telegram is a complete message framed by a start and a stop character.

[SOT][MESSAGE][EOT]

Start Of Telegram (SOT): 13 0D hex

End Of Telegram (EOT): 10 0A hex

Message

A message consists of a header, a payload and cyclic redundancy check (CRC).

[HEADER][PAYLOAD][CRC16]

To distinguish between SOT/EOT bytes, and similar bytes appearing in the message, an escape sequence conversion is performed before transfer. This is explained in [Escape Sequence Conversion](#).

Header

The header consists of 3 bytes: A destination address, a source address, and a message type.

[DESTINATION][SOURCE][TYPE]

Destination: 1 byte, 0..255

Source: 1 byte, 1..255

Type: 1 byte, 0..8

The destination address describes which module the message is intended for. Each module on a bus must have its own unique address, which is a number between 1 and 48.

Address 0 is reserved for special purposes. Addresses above 64 are considered host addresses (from PC or other equipment).

The source address tells which module has sent the message, thus where the response should be directed. For host equipment (such as a computer), the source address must be greater than 64 (40h).

The message type tells something about the purpose of the message.

Code	Type	Description
0	Nack	Response. Message not understood, not applicable, or not allowed.
1	CRC error	Response. CRC error in received message.
2	Busy	Response. Cannot respond at the moment. Module too busy.
3	Ack	Response. Received query message understood.
4	Read	Query. Read the contents of a register.
5	Write	Transmission. Write something to a register.
6	(Not used)	
7	(Not used)	
8	Datagram	Response. Register content returned; caused by a previous "Read".

Address Cycling

In multi-tasking systems, it may sometimes be difficult to be certain which response fits to which transmitted telegram. Even though the NKT Photonics modules/systems handle requests in the correct order, it is not always certain that the operating system or the software in the host computer does the same.

To overcome this, a simple solution can be to change the source address every time a new telegram is transmitted. When a module responds, the target address in the response telegram is identical to the source address in the request telegram, so the host computer can easily pair the response with the request. Addresses from 65 (41h) to 255 (FFh) can be used for this.

Payload

The payload consists of one or more register bytes, possibly followed by a number of data bytes. The number of data bytes in a single telegram cannot exceed 240.

Multi-byte numeric values (more than one byte in size) are transmitted little-endian, i.e. LSB first.

A few registers are standardized, meaning that they have the same purpose in all modules.

Reg #	Description
61h	Module type number. Is used to determine what type of module the host is communicating with.
64h	Firmware version code.

For other registers, please refer to the register list for the specific module you are communicating with.

CRC16

The CRC16 value consists of two bytes calculated on the complete message (header and payload).

Note: The CRC16 calculation is performed before escape sequence conversion on the outgoing telegram, and is always transmitted big-endian, i.e. MSB first.

Prior to calculation, the CRC16 value must be 0. Below is an example on how to implement a CRC16 algorithm in C. Char is an 8 bit value, and int is a 16 bit value.

```
// Update the CRC for transmitted and received data using
// the CCITT 16bit algorithm (X^16 + X^12 + X^5 + 1).
```

```
unsigned char ser_data;
static unsigned int crc;

crc = (unsigned char)(crc >> 8) | (crc << 8);
crc ^= ser_data;
crc ^= (unsigned char)(crc & 0xff) >> 4;
crc ^= (crc << 8) << 4;
crc ^= ((crc & 0xff) << 4) << 1;
```

When transmitting, all bytes in the message must be passed through the CRC calculating function, and the resulting two CRC bytes placed in the end of the message, MSB first.

When receiving, all bytes including the CRC bytes should be passed through the same CRC function. If there are no transmission errors, the result is 0.

For software development purposes, this web page can be helpful for calculating and checking CRC values: <http://www.lammetbics.nl/comm/info/crc-calculation.html>

Remember to set the input type to Hex (not ASCII). The correct CRC result for Interbus is CRC-CCITT (XModem).

On-line CRC calculation and free library

- [Introduction on CRC calculations](#)
- [Free CRC calculation routines for download](#)
- [CRC calculation support forum](#) **New**

"0F420470" (hex)	
1 byte checksum	197
CRC-16	0x24A0
CRC-16 (Modbus)	0x00A0
CRC-16 (Sick)	0x08BD
CRC-CCITT (XModem)	0x1570
CRC-CCITT (0xFFFF)	0x91B0
CRC-CCITT (0x1D0F)	0x1B60
CRC-CCITT (Kermit)	0xD015
CRC-DNP	0xF59C
CRC-32	0x3E5022DC

0F 42 04 70

Input type: ☐ ASCII ☒ Hex

Please note that the Lammertries web site is not related to NKT Photonics in any way.

2.3 Escape Sequence Conversion

Before a message is transmitted, it must be escape sequence converted. If a data byte is equal to one of the three numbers 10, 13 or 94 (0Ah, 0Dh or 5Eh), special action must take place. A start of escape sequence character is inserted in the message, and the escape sequence character (ECC) is added to the data byte.

The reason for this conversion is that the SOT and EOT characters can never ever be used for anything else than Start of Telegram and End of Telegram, in order to make telegram framing completely clear for all Interbus modules.

Below is a C example on how to make the escape sequence conversion work in transmission mode. The function "com1_putc" handles the transmission by the use of the UART.

```
void TXnocrc (unsigned char data)
{
    if(data == SOT || data == EOT || data == SOE)
    {
        com1_putc(SOE);
        data += ECC;
    }
    com1_putc(data);
}
```

Special Characters

Name	Code (Dec)	Code (Hex)	Meaning	Transmitted as (Hex)
EOT	10	0A	End of telegram	5E 4A
SOT	13	0D	Start of telegram	5E 4D
ECC	64	40	Escape sequence character	40 (not converted)
SOE	94	5E	Start of escape sequence	5E 9E

Example (all bytes in Hex):

When the following sequence of data (without framing) is transmitted:

[0A][3B][04][10][83][0A],

the data bytes 0Ah and 0Ah must be converted, in order not to be interpreted as EOT bytes.

After conversion:

[5E][4A][3B][04][10][83][5E][4A]

And the final telegram, including framing, will be:

[0D][5E][4A][3B][04][10][83][5E][4A][0A]

2.4 Transmission Example

Below is a C-code example with all functions to transmit a complete telegram.

```
#define SOT 0x0D
#define EOT 0x0A
#define SOE 0x5E
#define ECC 0x40

unsigned int CRC_add1(char data, unsigned int crc)
{
    crc = (unsigned char)(crc >> 8) | (crc << 8);
    crc ^= data;
    crc ^= (unsigned char)(crc & 0xff) >> 4;
    crc ^= (crc << 8) << 4;
    crc ^= ((crc & 0xff) << 4) << 1;
    return crc;
}

void TXnocrc(char data)
{
    if(data == SOT || data == EOT || data == SOE)
    {
        com1_putc(SOE);
        data += ECC;
    }
    com1_putc(data);    // Sends data to UART
}

unsigned int tx1(char data, unsigned int crc)
{
    crc = CRC_add1(data,crc);
    if(data == SOT || data == EOT || data == SOE)
    {
        com1_putc(SOE);
        data += ECC;
    }
    com1_putc(data);
    return crc;
}

// Dest is the destination address, size is number of data bytes to be
// transmitted, type is the
// type byte and *data is a pointer to the data. MY_NODE is the address of
// the transmitter.
// DIR1 is a hardware pin to control direction of RS485 interface.

void TxTlg(char dest, char size, char type, char *data)
{
    unsigned int crc_value;
    char tt;

    DIR1 = OUTGOING;                // Change direction of
    RS485                            // RS485
    com1_putc(SOT);                  // Start a new telegram
    crc_value = tx1(dest, 0);        // Transmitting
    destination address
```

```

        crc_value = tx1(MY_NODE, crc_value);        // Transmitting source
address
        crc_value = tx1(type, crc_value);          // Transmitting type
        for(tt=0;tt<size;tt++)
        {
            crc_value = tx1(*data,crc_value);      // Transmitting data
            data++;                                // Increment datapointer
        }
        TXnocrc((crc_value>>8) & 0xFF);           // high byte of CRC
        TXnocrc(crc_value & 0xFF);                // low byte of CRC
        com1_putc(EOT);                           // End telegram
        DIR1 = INGOING;                           // Change the direction
of RS485
    }

```

2.5 Reception Example

Analysis of a received telegram is done in the reverse order of packing for transmission.

When a SOT is received a new telegram is initiated. Set CRC value to 0.

When a SOE is received, subtract ECC (40h) from the next data byte and then calculate the new CRC.

When anything but SOT, EOT or SOE is received, calculate the new CRC.

When an EOT is received - CRC must be 0, otherwise a CRC-error has occurred.

For historical reasons, a few module types do not respond to "Write" commands. All future modules, however, will be able to answer a successful "Write" with "Acknowledge".

Below is a C-code example on how to implement the reception, reverse escape sequence conversion and CRC check. Some additional checks may be implemented to catch overrun errors etc.

```

short TelegramReady, Inframe, Escape_seq1; // boolean
char rcv_counter, RcBuffer[BUFLNGTH], *RcInPtr1;
unsigned int crc1;

void COM1_isc(void) //Called by interrupt
{
    char ch;
    while(UARTnotEmpty)
    {
        if(!FERR1) // no Framing Error
        {
            ch = RcREG1; // Get byte from UART
            switch(ch)
            {
                case SOT: // New telegram
                    rcv_counter1 = 0;
                    inframe1=1;
                    crc1 = 0;
                    RcInPtr1 = &RcBuffer[0];
                    TelegramReady = 0;
                    break;
                case EOT: // End of telegram
                    inframe1=0;
                    if(CRC1 == 0) TelegramReady = 1; // CRC ok and telegram
                    is ready
                    break;
                case SOE: // start of escape sequence
                    Escape_seq1 = 1;
                    break;
                default: // Data byte
                    if(inframe1==1)
                    {
                        if(Escape_seq1 == 1)
                        {
                            Escape_seq1 = 0;
                            ch -= ECC;
                        }
                        crc1 = CRC_add1(ch, crc1);
                        *RcInPtr1 = ch;
                        if(rcv_counter1 < BUFLNGTH)
                        {
                            RcInPtr1++;
                            rcv_counter1++;
                        }
                    }
                    break;
            }
        }
    }
}

```

3 C++ Source Code

Purpose

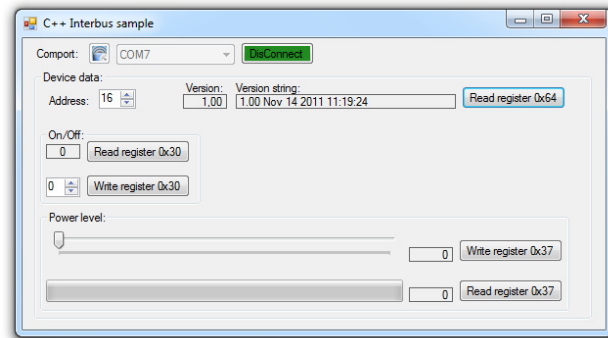
This section describes how to use the C++ example application.

Requirements

The module(s)/system(s) to be controlled must feature a digital interface which is connected directly or via an adaptor to the computer. To compile and run the example code, you will need a Visual Studio 2010 installation. The Visual C++ 2010 Express edition could be downloaded from Microsoft's website.

Description

The IBSamplecpp example is a small windows application written in Visual Studio 2010 C++ source. The example code illustrates how to communicate and control the connected module(s)/system(s) via the binary NKT Photonics protocol.



Please note! The sample code does not implement retransmission on CRC error in the communication, 3-5 retries should be handled in a real world application. And ideally the communication should run in its own thread to allow the GUI to be responsive while the communication is in progress.

Files

The zip archive IBSamplecpp.zip contains a simple Visual Studio 2010 C++ project.

The most interesting files might be: *InterbusFunc.h* and *InterbusFunc.cpp* which contains the protocol implementation and some helper routines:

InterbusFunc.h: (brief)

```
// unsigned char functions
static unsigned char readInterbus_Byte(SerialPort^ wrkPort, unsigned char deviceId, unsigned char registerId);
static bool writeInterbus_Byte(SerialPort^ wrkPort, unsigned char deviceId, unsigned char registerId, unsigned char data);

// unsigned short functions
static unsigned short readInterbus_UInt16(SerialPort^ wrkPort, unsigned char deviceId, unsigned char registerId);
static bool writeInterbus_UInt16(SerialPort^ wrkPort, unsigned char deviceId, unsigned char registerId, unsigned short data);

// unsigned long functions
static unsigned long readInterbus_UInt32(SerialPort^ wrkPort, unsigned char deviceId, unsigned char registerId);
static bool writeInterbus_UInt32(SerialPort^ wrkPort, unsigned char deviceId, unsigned char registerId, unsigned long data);

// float functions
static float readInterbus_Float32(SerialPort^ wrkPort, unsigned char deviceId, unsigned char registerId);
static bool writeInterbus_Float32(SerialPort^ wrkPort, unsigned char deviceId, unsigned char registerId, float data);

// Generic/Stream functions
static array<unsigned char>^ readInterbus_Stream(SerialPort^ wrkPort, unsigned char deviceId, unsigned char registerId);
static bool writeInterbus_Stream(SerialPort^ wrkPort, unsigned char deviceId, unsigned char registerId, array<unsigned char>^ data);

// Current MasterId
static unsigned char masterId = 66;

// Current Message timeout in mS
static int timeout_mS = 50;
```

4 C# Source Code

Purpose

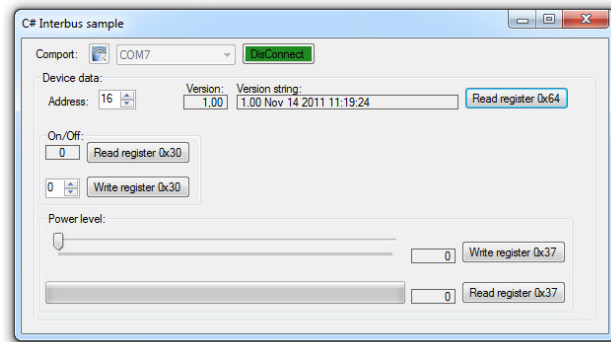
This section describes how to use the C# example application.

Requirements

The module(s)/system(s) to be controlled must feature a digital interface which is connected directly or via an adaptor to the computer. To compile and run the example code, you will need a Visual Studio 2010 installation. The Visual C# 2010 Express edition could be downloaded from Microsoft's website.

Description

The IBSample example is a small windows application written in Visual Studio 2010 C# source. The example code illustrates how to communicate and control the connected module(s)/system(s) via the binary NKT Photonics protocol.



Please note! The sample code does not implement retransmission on CRC error in the communication, 3-5 retries should be handled in a real world application. And ideally the communication should run in its own thread to allow the GUI to be responsive while the communication is in progress.

Files

The zip archive IBSample.zip contains a simple Visual Studio 2010 c# project.

The most interesting file might be: *InterbusFunc.cs* which contains the protocol implementation and some helper routines:

InterbusFunc.cs: (brief)

```
// Current MasterId
public static byte masterId = 66;

// Current Message timeout in mS
public static int timeout_mS = 35;

// unsigned Byte functions
public static byte readInterbus_Byte(SerialPort wrkPort, byte deviceId, byte registerId)
public static bool writeInterbus_Byte(SerialPort wrkPort, byte deviceId, byte registerId, byte data)

// unsigned Int 16 functions
public static UInt16 readInterbus_UInt16(SerialPort wrkPort, byte deviceId, byte registerId)
public static bool writeInterbus_UInt16(SerialPort wrkPort, byte deviceId, byte registerId, UInt16 data)

// unsigned Int 32 functions
public static UInt32 readInterbus_UInt32(SerialPort wrkPort, byte deviceId, byte registerId)
public static bool writeInterbus_UInt32(SerialPort wrkPort, byte deviceId, byte registerId, UInt32 data)

// float 32 functions
public static Single readInterbus_Float32(SerialPort wrkPort, byte deviceId, byte registerId)
public static bool writeInterbus_Float32(SerialPort wrkPort, byte deviceId, byte registerId, Single data)

// Generic/Stream functions
public static byte[] readInterbus_Stream(SerialPort wrkPort, byte deviceId, byte registerId)
public static bool writeInterbus_Stream(SerialPort wrkPort, byte deviceId, byte registerId, byte[] data)
```

5 Labview Driver

Purpose This section describes how to use and how to establish communication from LabView with NKT Photonics modules/systems with digital interface, based on the NGSerialPort VI from NKT Photonics.

Requirements The module(s)/system(s) to be controlled must feature a digital interface which is connected directly or via an adaptor to the computer. The NGSerialPort VI can be utilized in LabView 8.5 or any newer revision of LabView.

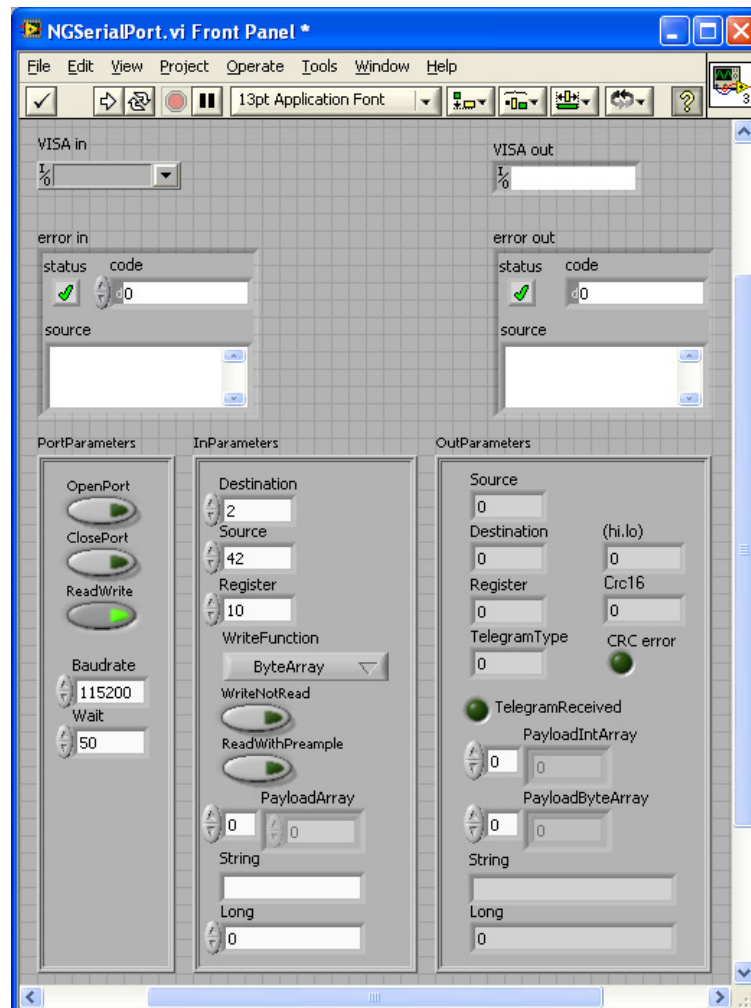
Description The NGSerialPort VI contains all functions needed for establishing communication from LabView on the computer the NKT Photonics product(s) is connected to.

The VI is used to setup, open and close the communication port and it can be used to both read from and write to registers of different types.

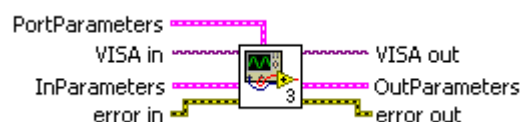
5.1 Inputs and Outputs

This section describes the inputs and outputs on the NGSerialPort VI, which is available both on the front panel as well as on the icon in the block diagram.

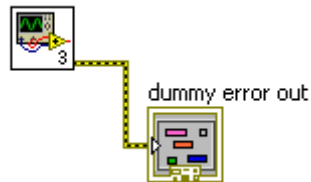
Front Panel



Icon



VISA in	This is an input, which is used to configure which COM-port to be used for communication. Please notice that even if the module(s)/system(s) may be connected to the computer with an USB connection, this connection will establish a virtual COM-port, which should be used for the communication.
VISA out	This terminal is an output, which will provide COM-port number used for the communication. This can be connected to the following NGSerialPort in the block diagram if desired.
error in	Typical error input cluster used in LabView. It can be connected to the previous function with an error out. Please be aware that there will not be any communication if the error in informs about a previous error.
error out	Typical error output cluster. It can be connected to the following function with an error in. However please notice that if write commands are sent to input registers on a module/system that does not reply back with an acknowledge (ACK) or not acknowledge (NACK) the NGSerialPort VI will time out and will generate an error even though nothing is actually wrong. So in this case the error out should not be connected to the following function with an error in, but instead be terminated with a dummy terminal.

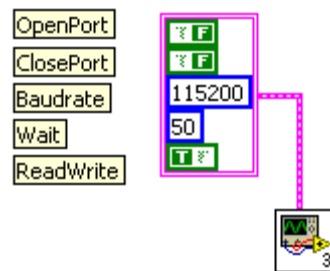


Modules and systems that does not generate ACK and NACK are:

- Koheras BasiK Module
- Koheras AdjustiK/BoostiK System

5.1.1 PortParameters

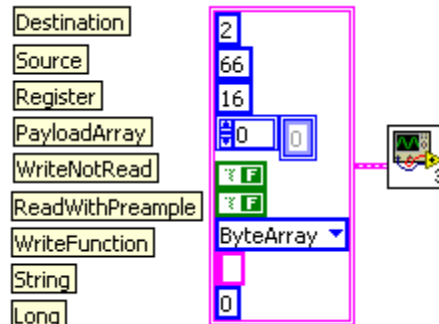
Input cluster to setup, open and close communication port and to select whether the VI should be used for communication. This input cluster can either be configured with the bundle function in LabView or a constant as shown below.



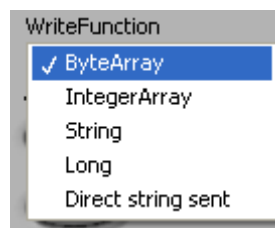
- OpenPort** The communication port does not have to be opened every time LabView should be communicating with the module/system. The communication port must be opened the first time (or prior) and again after the communication port has been closed if further communication is desired.
- ClosePort** When no further communication is desired, the communication port should be closed with the ClosePort input. If the communication is not ended with the ClosePort command, the port will be hanging and communication cannot be established from other programs.
- Baudrate** This input sets the baudrate on the computer. All NKT Photonics products communicating with the binary NKT Photonics protocol are communication with a baudrate at 115200 bits per second. The computer should have the same baudrate as the module(s)/system(s), so set this input to 115200.
- Wait** This input configures how long time in milliseconds the NGSerialPort should be waiting from a response from the interfacing module/system. Typical values are 50 or 100 milliseconds.
- ReadWrite** With the ReadWrite input it is selected if the NGSerialPort VI should be used for actual communication and not just for opening or closing the communication port. No matter if the VI should be used for writing or reading this input should be set high.

5.1.2 InParameters

Input cluster with information about addresses, whether the VI should be used for reading or writing, in case it should be used for writing what type of data should be written and the actual data to be written. This input cluster can either be configured with the bundle function in LabView or a constant as shown below.



- Destination** This input is used to set the address of the module/system to communicate with. Module addresses are typically in the range from 1 to 48.
- Source** This input is used to set the address of the computer. It is recommended that the address of the host computer is 65 (41h) or higher. In the example above, 66 is used.
- Register** This input configures the register address to read or write.
- WriteFunction** In case data should be transmitted to a module/system, the WriteFunction selector is used to select whether it is a ByteArray, IntegerArray, String, Long or a Direct string that should be transmitted.

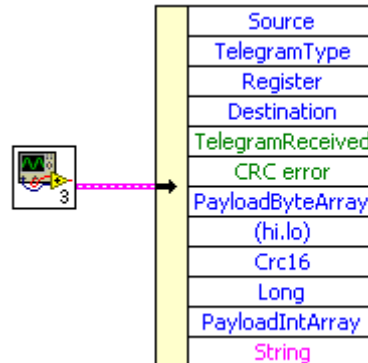


If just an integer should be transmitted the Long function can be used.

- WriteNotRead** This input is used to configure whether the VI should be used for writing or reading. The input should be set low for reading and high for writing.
- ReadWithPreamble**
- PayloadArray** If a ByteArray or an IntegerArray should be transmitted, the array should be inserted into the PayloadArray. If a single byte or integer should be transmitted, it should go into the PayloadArray by using the Build Array function leading to an array with just a single element.
- String** When a String should be transmitted it should be inserted into the String input.
- Long** Input for Long values.

5.1.3 OutParameters

Output cluster to with information about addresses, whether the VI should be used for reading or writing, in case it should be used for writing what type of data should be written and the actual data to be written. The output can be splitted up with the Unbundle By Name function as shown below.



Source This output provides the sender address of the received telegram.

TelegramType The TelegramType output provides a general information about the received telegram.

Code	Type	Description
0	NACK	Response. Message not understood, not applicable, or not allowed.
1	CRC error	Response. CRC error in received message.
2	Busy	Response. Cannot respond at the moment. Module too busy.
3	ACK	Response. Received message understood.
4	Read	Query. Read the contents of a register.
5	Write	Transmission. Write something in a register.
8	Datagram	Response. Register content returned; caused by a previous "Read".

Register This output provides the register address for the received telegram.

Destination The Destination output provides the receiver address of the received telegram. If the Source address is set to 42h in the InParameters the module/system will reply back to 42h in the Destination address with its response.

TelegramReceived Boolean output, which is low if no telegram is received and high if a telegram is received.

CRC error The binary protocol utilizes a cyclic redundancy check (CRC) to ensure data are not corrupted during the transmission. When the NGSerialPort VI receives a telegram it will verify the CRC value, and in case the telegram got corrupted during the transmission it will report a CRC error (high if error).

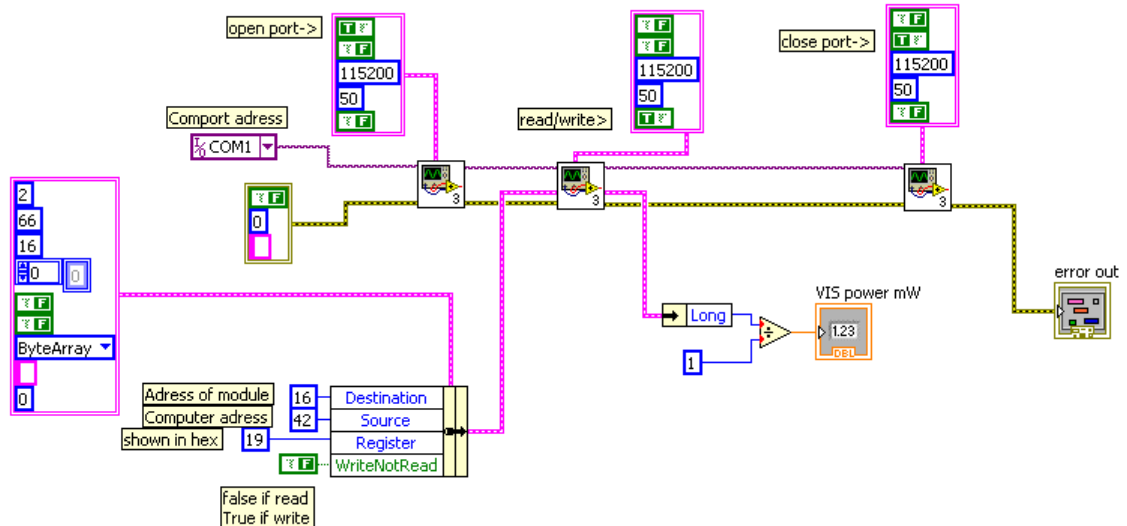
PayloadByteArray	This output array provides the received data as bytes in array form.
(hi.lo)	Not used. For development use.
Crc16	The Crc16 output provides the CRC value for the received telegram.
Long	This output provides the received data as a Long value.
PayloadIntArray	This output array provides the received data as integers in array form.
String	The String output provides the received data as a String.

5.2 Programming Examples

This section provides two programming examples, i.e. an example for reading a value from a module and another example for writing a value to a module.

5.2.1 Reading Example

Following an example is shown for how to read out a value from a module.



Opening Port

First time the NGSerialPort VI is called, it is only for opening the communication port. In this example COM1 is used for the communication port.

Building Up Telegram

Before the NGSerialPort VI is called for the second time, the InParameters cluster is configured with a constant and Bundle By Name function. In this example the module to communicate with has address 16h (22 dec), the computer is given address 42h (66 dec) and the register to receive has address 19h. Please notice that from the block diagram it cannot directly be seen whether the numbers are shown as decimal or hex numbers, so this can often explain if the communication is not working properly.

Requesting and Receiving Telegram

When the NGSerialPort VI is called for the second time the PortParameters are set to read/write.

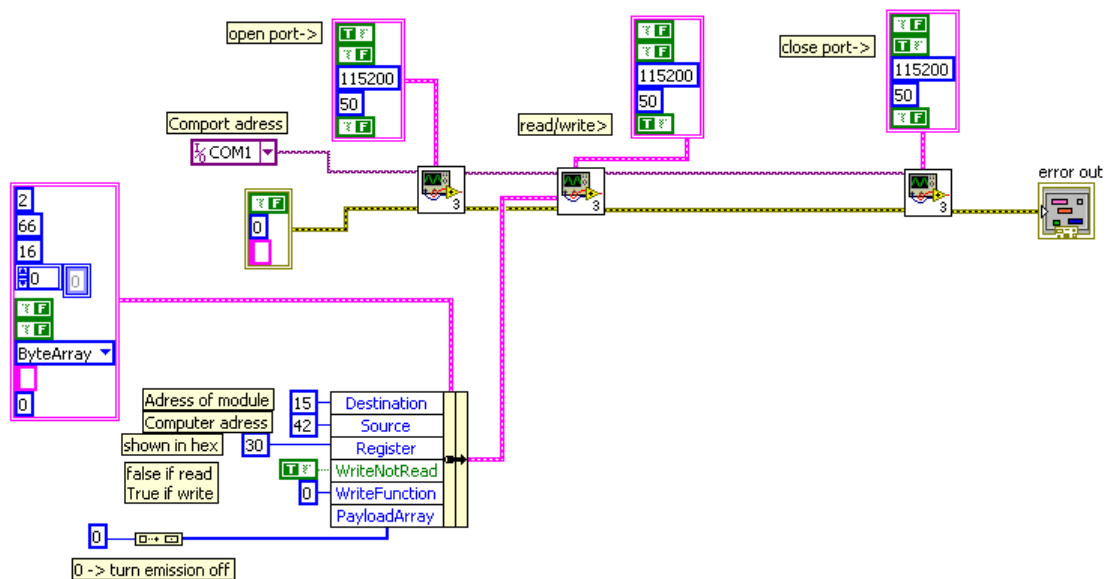
After the NGSerialPort VI has been called the second time, the type of received data is picked out with the Unbundle By Name function, and in this case scaled with a factor of 1 before it is shown in the VIS power mW indicator.

Closing Port

Last before ending the program the port is closed by setting the second boolean value high in the PortParameter cluster for the third call of the NGSerialPort VI.

5.2.2 Writing Example

Following an example is shown for how to write a value to a module.



Building Up Telegram

Before the NGSerialPort VI is called for the second time, the InParameters cluster is configured with a constant and Bundle By Name function. In this example the module to communicate with has address 15h (21 dec), the computer is given address 42h (66 dec) and the register to write to has address 30h. The WriteNotRead input is set high to enable writing. The WriteFunction is set to 0, which means a ByteArray will be transmitted. In this example register 30h is an emission on/off register, so by sending a 0 into the Build Array function and into the PayloadArray input the emission will be shut off when the NGSerialPort transmits this telegram.

Writing

When the NGSerialPort VI is called for the second time the PortParameters are set to read/write.

Error Looping

Like in the reading example the error out from one NGSerialPort VI is connected to the following function. This will only work for writing to registers when the module respond back with an ACK (Acknowledge), otherwise the function will time out and generate an error. For modules/systems that do not generate ACK a dummy terminal must be used as shown earlier in this document.

6 Register Files

For information about what registers to read from and write to, please refer to the Register Files.

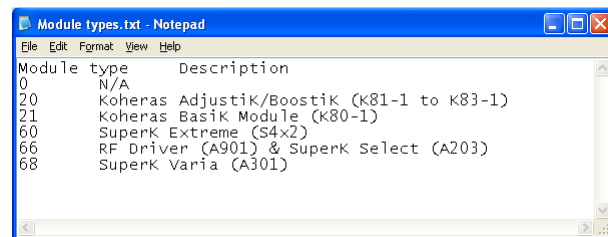
Each Register File provides information about read and read/write registers in one module or mainboard.

Notice

In systems like e.g. SuperK Extreme or Koheras BoostiK, it is recommended to communicate with the mainboard and not the different sub-modules, as it otherwise can happen that one or multiple sub-modules are operating in different states than the system mainboard is capable of.

Module Types

Open the Module types.txt file with e.g. Microsoft Notepad to get an overview of the different types of modules.



So for communication with e.g. a SuperK Extreme system it is seen that it has the module type number 60h. So for information about registers to communicate with in a SuperK Extreme system, please refer to the register file "60.txt".

How to Read Register File

Open the desired register file in e.g. Microsoft Excel. The register files are tab delimited ASCII files, which Microsoft Excel can open and present.

Module type	Description	Value	Unit	Address	Value
60	SuperK Extreme (S4x2)				
#					
Readings					
11	NTC1 temperature	°C	116		0.1
#					
Controls					
30	Emission	0=Off;1=On	U8		1
31	Setup bits	0=Current mode;1=Power mode	U16		1
32	Interlock	(>0=reset interlock)	U16		1
34	Pulse-Picker ratio	Times	U16		1
35	Pulse-Picker delay	ns	U8		0.25
36	Watchdog interval	Seconds	U8		1
37	Power level	%	U16		0.1
38	Current level	%	U16		0.1
65	Module serial number		string		
6C	User text		string		
#					
Status bits					
0	Emission LED on				
1	Interlock off				
2	Interlock power failure				
3	Interlock loop off				
4	External disable				
5	Supply voltage low				
6	Module temp range				
7	-				
8	-				
9	-				
10	-				
11	-				
12	-				
13	-				
14	USB log error code present				
15	Error code present				
#					
Error code					
0	No error				
#					
#					

First the file tells that in register 61h the module type can be read out from the module/system. In this case it will respond back with value 60h, which we from the Module types list know is a SuperK Extreme mainboard.

Readings and Controls

The section with Readings includes all read only registers. The Controls section includes all registers that can be written to (unless they are write-protected). In most cases it is also possible to read the content of a Control register.

Register Address

For Readings and Controls registers, their addresses are found in the first column.

Register Description

In the second column there is a small description of the different registers.

Unit

The third column provides units for the different registers like: V, °C, Seconds, etc.

Type

The fourth column provides information about the type of content of the different registers,

Type	Variable
U8	Unsigned 8-bit Integer
U16	Unsigned 16-bit Integer
U32	Unsigned 32-bit Integer
I8	Signed 8-bit Integer
I16	Signed 16-bit Integer
I32	Signed 32-bit Integer
H8	Hexadecimal 8-bit Integer
H16	Hexadecimal 16-bit Integer
H32	Hexadecimal 32-bit Integer
string	Text string

Scaling

The last column tells what factor that should be multiplied with the contents of the actual register to get to the unit in the third column.

An example: A value 43264 is read from a register with the Unit °C, Type U16 and Scaling 0.001. The Type U16 tells it is an unsigned 16-bit integer, so it is a positive number. The Scaling factor 0.001 tells that the number has 3 digits after the decimal point, so 43264 becomes 43.264 °C.

6.1 General Registers

This section describes the common registers for NKT Photonics modules/systems.

Module Type	Register address 61h tells which module the host computer is communication with. This is used to establish or verify which type of module the host computer is connected to.
Serial Number	On register address 65h the serial number of the module/system can be read. The serial number is an 8 character string. In case of multiple modules/system on the same bus, the serial number can be used for identification of a given module/system.
Status Bits	On register address 66h the status of a module/system can be monitored. The status bits are combined in either one or more bytes. For explanation of the different status bits, please refer to the respective Register file.
Error Code	On register address 67h the module/system will provide an error code in case the actual module/system has shut down. The error code is read out as a byte value. Please refer to the respective Register file for more information about the various error codes.

Please notice that the Koheras AdjustiK/BoostiK Systems (K81-1 to K83-1) and Koheras BasiK Module (K80-1) do not generate error codes.

6.2 Koheras AdjustiK/BoostiK System (K81-1 to K83-1)

This section describes specific registers for Koheras AdjustiK/BoostiK Systems with product numbers starting with K81-1, K82-1 and K83-1.

Module Address	For communication with the Koheras AdjustiK/BoostiK System communication should go through the mainboard. The address for the mainboard is 0Fh (15 dec).
General	Output power and wavelength can be controlled for the system. The actual output power level and estimated wavelength cannot be monitored through the Interbus interface.
Setpoint	Dependant on whether the system is operating in current or power mode the output power can be controlled by writing either a new pump current level or output power level to the Setpoint register (23h). The Setpoint value is an unsigned 16-bit integer value. Pump current is set in mA and output power is set in hundredths of milliwatt.
Wavelength Offset	The Wavelength Offset is an unsigned 16-bit integer value of the wavelength of the laser in nanometers, 1-3 nm below the actual wavelength. The Wavelength is a constant value for a specific system as the wavelength adjustment is made with the FL Setpoint register.
FL Setpoint	<p>Dependant on whether the fiber laser is operating in temperature or wavelength mode, the wavelength can be controlled by writing either a temperature or wavelength to the FL Setpoint register (25h). The FL Setpoint value is an unsigned 16-bit integer value. Temperature is set in milli-degrees C. The wavelength setting is made in picometers and is relative to the Wavelength offset value.</p> $\text{Total wavelength [nm]} = \text{Wavelength Offset [nm]} + \text{FL Setpoint pm} / 1000$
Emission	The Emission register (30h) is an unsigned 8-bit integer register. Writing 0 to this register will turn off laser emission from the Koheras AdjustiK/BoostiK System. Writing 1 to this register will turn on laser emission if the Interlock circuit has been reset.

6.3 Koheras BasiK Module (K80-1)

This section describes specific registers for Koheras BasiK Module with product numbers starting with K80-1.

Setpoint, FL Setpoint, Wavelength Offset and Emission are similar as for Koheras AdjustiK/BoostiK System (K81-1 to K83-1).

Module Type and Module Address	The module type number is 21h. The standard module address is 10 (0Ah), but the actual address may be different, as there can be several modules on one bus.
Current/Power Mode	If the laser module supports both Current and Power mode, it is possible to shift between the two modes with the Current/Power Mode register (31h). The value 0 will put the laser module into Current Mode. The value 1 will put the laser module into Power Mode.
Piezo Modulation	If the laser module features Piezo Modulation, this can be enabled and disabled with the Piezo Modulation register (32h). Writing the value 0 to the register will disable Piezo Modulation, whereas the value 1 will enable this.
RIN Suppression	On E15 BasiK modules the RIN Suppression circuit can be enabled or disabled by writing either 1 or 0 respectively to the RIN Suppression register (33h).
Temperature/ Wavelength Mode	On Koheras BasiK Modules supporting both Temperature and Wavelength Mode, it is possible to shift between these two modes by writing either 0 (Temperature) or 1 (Wavelength) to the Temperature/Wavelength Mode register (34h).
	The four registers mentioned above all operate with 8-bit unsigned integers.
Readings	The following readings: Fiber Laser Temperature (11h), Pump Current (15h), Output Power (18h), Module Temperature (19h) and Module Input Voltage (1Bh) can all be monitored through the Interbus interface. Scaling can be seen in the Register file.

6.4 SuperK EXTREME System (S4x2)

This section describes specific registers for SuperK EXTREME with product numbers starting with S4x2.

Module Type and Module Address	The module type number is 60h. The standard module address is 15 (0Fh).
Inlet Temperature	The Inlet temperature can be monitored by reading register 11h.
Emission	Emission can be controlled with the Emission register (30h). The Emission register is based on an 8-bit unsigned integer value, where 0 turns off laser emission and 3 turns on laser emission (if the Interlock circuit has been reset).
Setup Bits	With the Setup Bits register (31h) the operation mode of the SuperK EXTREME System can be controlled. This register is based on a 16-bit unsigned integer value. Writing 0 to this register will put the system into Current mode, whereas 1 will put it into Power mode.
Interlock	If the door interlock is in place, the key switch on the front plate is in On position and the External bus is terminated with e.g. a bus defeater then the Interlock circuit can be reset via the Interbus interface by sending a value greater than 0 to the Interlock register (32h).
Pulse-Picker Ratio	For SuperK EXTREME Systems featuring the Pulse-Picker option, the ratio for the Pulse-Picker can be controlled with the Pulse-Picker Ratio register (34h).
NIM Delay	On systems with NIM trigger output the delay of this trigger signal can be adjusted with the NIM Delay register (35h). The input for this register should be an unsigned 8-bit value in steps of 0.25 nanoseconds.
Watchdog Interval	The system can be set to make an automatic shut-off (laser emission only – not electrical power) in case of lost communication. The value in the watchdog interval register determines how many seconds with no communication the system will tolerate. If the value is 0, the feature is disabled.
Power Level	When the system is operated in Power Mode the setpoint in the Power Level register (37h) is used. The setpoint is in tenths of percentage (per mille).
Current Level	When the system is operated in Current Mode the setpoint in the Current Level register (38h) is used. The setpoint is in tenths of percentage (per mille).
User Text	The SuperK EXTREME System has a User Text register (6Ch). This is a 40 character string register, which can be used by the user for e.g. identification.

6.5 RF Driver (A901) and SuperK SELECT (A203)

This section describes specific registers for an Internal (as part of a SuperK EXTREME system) or External RF Driver (A901) together with a SuperK SELECT (A203) accessory. All communication is with RF Driver, not with SuperK SELECT

Module Type and Module Address	<p>The RF Driver module type is 66h. An Internal RF Driver, i.e. when mounted inside the SuperK Extreme chassis, the standard module address is 6. On an External RF Driver the address depends on the tiny rotary switch (marked 0..9 and A..F) on the rear. The module address is determined by adding 16 (10h) to the hexadecimal setting of the switch.</p> <p>The SuperK SELECT module type is 67h. The address depends on the tiny rotary switch (marked 0..9 and A..F) located close to the connectors. The module address is determined by adding 16 (10h) to the hexadecimal setting of the switch.</p>
RF Power	<p>RF Power out of the RF Driver to the SuperK SELECT is controlled with the RF Power register (30h). Writing 0 to this register will turn off RF Power, whereas the value 1 will turn on RF Power if the RF Driver has its RF output connected to a RF input on a SuperK SELECT.</p>
Setup Bits	<p>Register 31h contains three setup bits which control the operation of the RF Driver.</p> <p>Bit 0: Use temperature compensation. When this feature is on, the RF Driver output is compensated for filter temperature.</p> <p>Bit 1: Use optimal power table. When this feature is on, RF output power settings are scaled according to wavelength settings.</p> <p>Bit 2: Blanking Level. Can be used to set the Blanking Level input high or low, without having an actual blanking level input signal.</p>
Minimum Wavelength	<p>From the Minimum Wavelength register (34h) the lowest wavelength the SuperK SELECT supports can be read.</p>
Maximum Wavelength	<p>From the Maximum Wavelength register (35h) the highest wavelength the SuperK SELECT support can be read.</p>
Crystal Temperature	<p>The temperature of the crystal in the SuperK SELECT can be monitored by the Crystal Temperature register (38h).</p>
FSK Mode	<p>A special non-free option. Please contact NKT Photonics for further information.</p>
Wavelengths	<p>The RF Driver features 8 channels, i.e. up to 8 wavelengths can be controlled at the same time. These 8 setpoints should be written to the registers 90h-97h. Each register holds a 4-element array of 32-bit unsigned integers. The 32-bit unsigned integers are for wavelengths in picometers. The 4-elements are for the 4 states of Frequency Shift Keying (FSK) operation (non-free option). If FSK mode is not used, the element on index 0 is used.</p>
Amplitudes	<p>The amplitudes of the 8 channels are controlled similar to the wavelengths with 8 4-elements arrays with unsigned 16-bit integers on registers B0h-B7h. The 16-bit integers are for the amplitude settings in tenths of percentage. The 4-elements are for the 4 states of FSK operation (non-free option). If FSK mode is not used, the values from index 0 are used.</p>
Modulation Gain Settings	<p>For use with the non-free FSK option. Adjusts the gain of the analog modulation inputs. Each of the 8 channels has its own gain setting, thus using the registers C0h-C7h. Contact NKT Photonics for further information.</p>

6.6 SuperK VARIA (A301)

This section describes specific registers for SuperK VARIA (A301) accessory.

The SuperK VARIA module type is 68h. The address depends on the tiny rotary switch (marked 0..9 and A..F) located close to the connectors. The module address is determined by adding 16 (10h) to the hexadecimal setting of the switch.

Monitor Input

Reading register (13h) will show current output power in percent. Requires the optional monitor to be attached for this register content being valid.

ND Setpoint

The output level of the SuperK VARIA is controlled with an unsigned 16-bit integer value sent to the ND Setpoint register (32h). The value in this register is in tenths of percentage.

SWP Setpoint

Short Wave setpoint register (33h) unsigned 16-bit integer. The value in this register is in tenths of an nm.
ex. write 3500 to register (33h) will set the SWP position at 350.0 nm.

LWP Setpoint

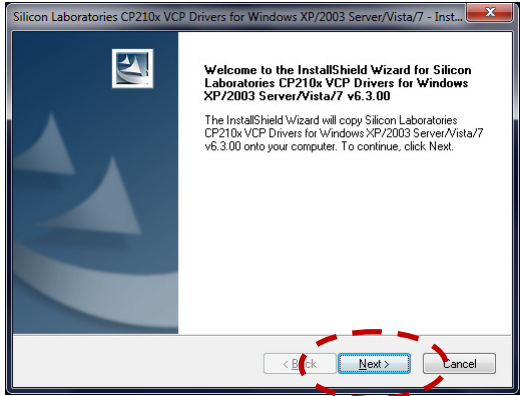
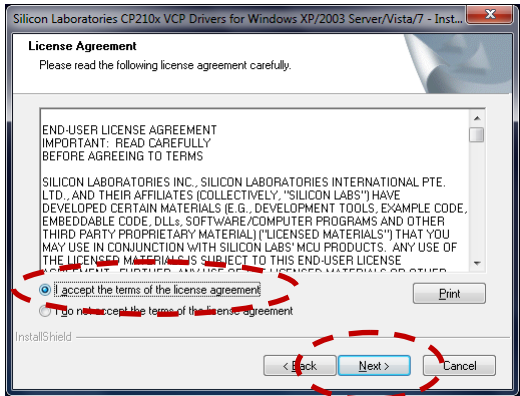
Long Wave setpoint register (34h) unsigned 16-bit integer. The value in this register is in tenths of an nm.
ex. write 5512 to register (34h) will set the LWP position to 551.2 nm.

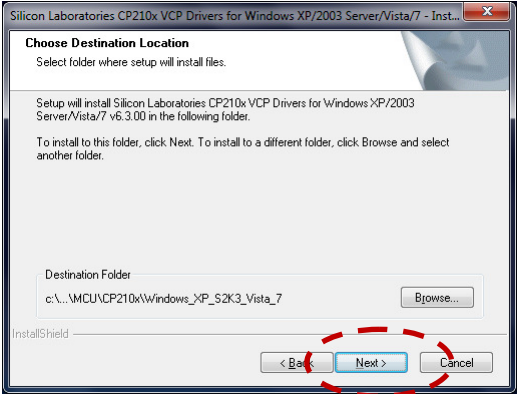
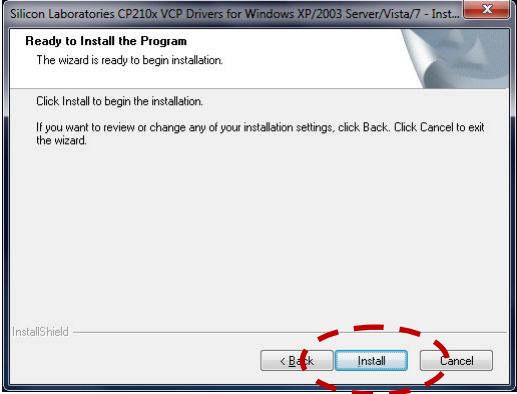
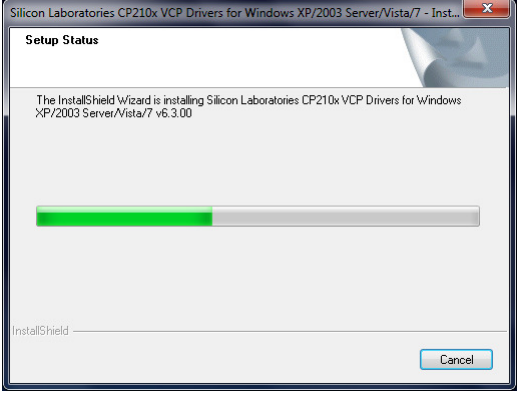
7 USB Driver

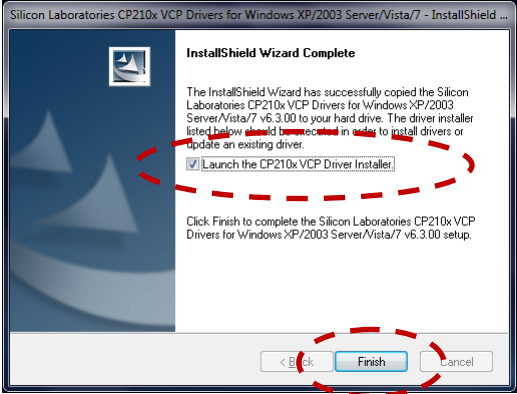
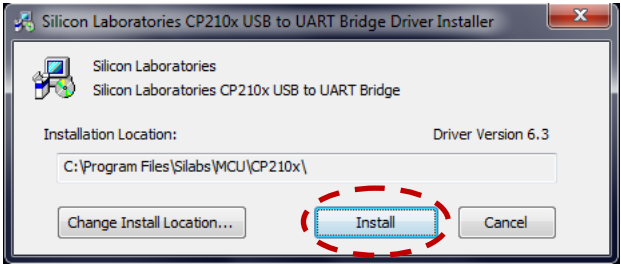
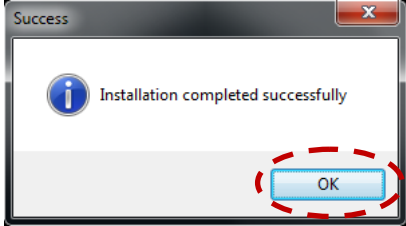
Install the Silicon Laboratories CP210x USB to UART Bridge software before connecting the system with the USB port on the computer.

USB Driver

The following procedure describes how to install the driver software.

Step	Description
1	The USB driver is found in the 07_USB Driver folder in the Software Development Kit. Alternatively it can be downloaded from www.nktphotonics.com/software .
2	Run the 'CP210x_VCP_Win_XP_S2K3_Vista_7.exe' file
3	Click Next on the window that appears. 
4	Read the License Agreement, choose I accept the terms of the license agreement, and click Next. 

5	<p>Click on Next to proceed the installation.</p> 
6	<p>Click on Install to begin the installation.</p> 
7	<p>Files are being copied to the computer.</p> 

8	<p>Select Launch the CP210x VCP Driver Installer and click Finish to begin the actual driver installation.</p> 
9	<p>In the dialog box that appears, click on the Install button to install the USB driver.</p> 
10	<p>When the installation has completed, click OK to end the installer.</p> 

8 Generic User Interface

Description

The Generic User Interface software is a development tool, i.e. a help for programmers to verify and understand how to communicate with NKT Photonics products. The Generic User Interface is capable of controlling all NKT Photonics products communicating via the NKT Photonics Interbus, e.g. Koheras Basik module and SuperK EXTREME products on a PC running Microsoft Windows.

Labview Runtime Engine

The Generic User Interface is developed in Labview 2011, which means that the Labview 2011 runtime engine will be installed with the Generic User Interface.

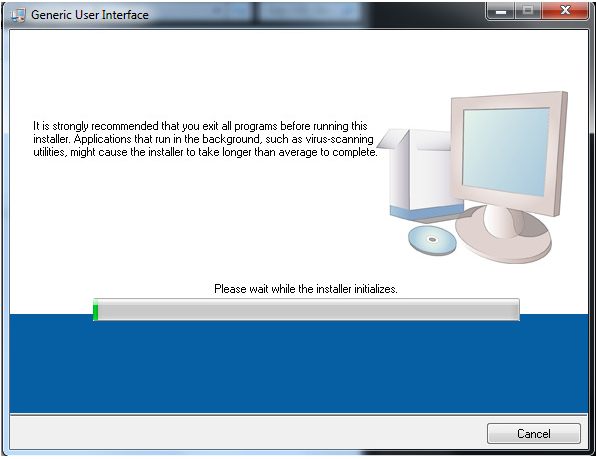
8.1 Installing the software

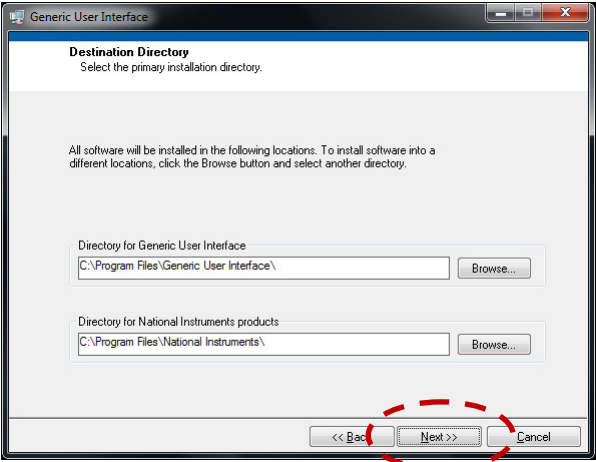
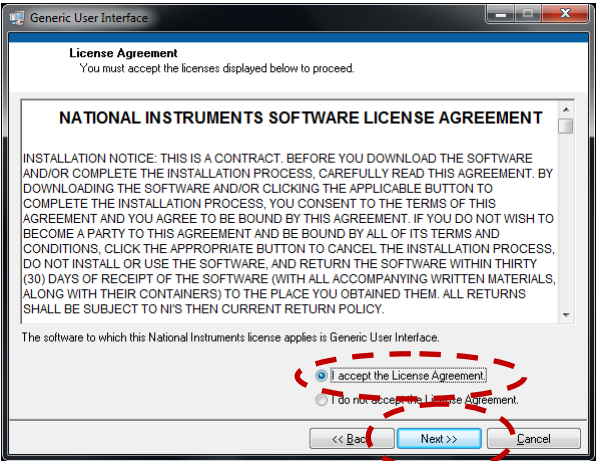
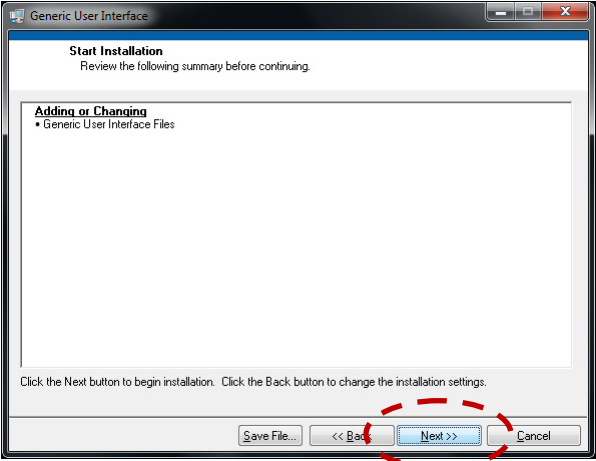
USB and Ethernet

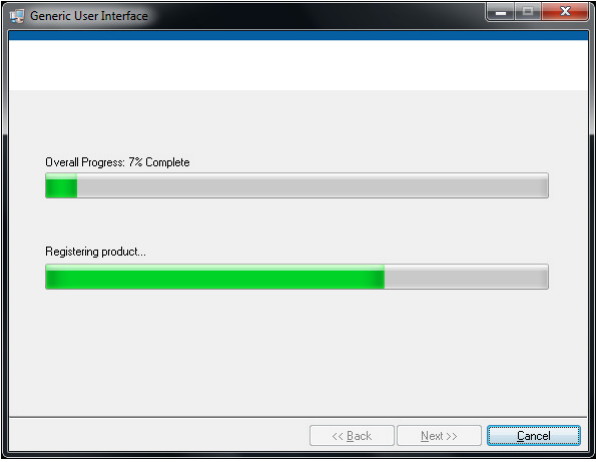
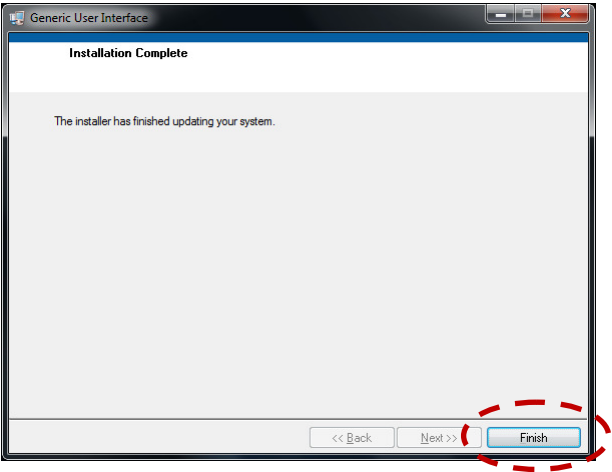
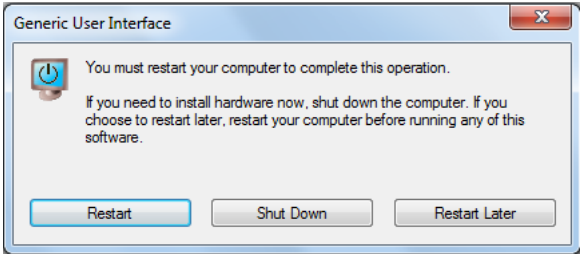

For information about how to install USB driver please refer to [USB Interface](#). For information about how an Ethernet interface should be configured for communication, please refer to the hardware instruction manual.

Generic User Interface Software

Use the following procedure to install the Generic User Interface software.

Step	Description
1	The installation package is located in the 08_Generic User Interface folder in the Software Development Kit.
2	Run setup.exe.
3	<p>An installer window will show up on the screen.</p> 

4	<p>The installer asks for destination directories. Choose the suggested directories by clicking Next.</p> 
5	<p>Read the License Agreement, and choose I accept the License Agreement(s) and Next to proceed.</p> 
6	<p>Click Next to start copying files to the computer.</p> 

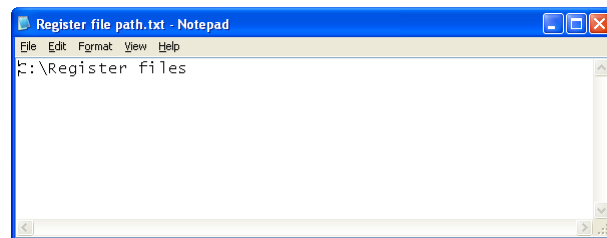
7	<p>A new window shows the progress of the installation.</p> 
8	<p>When the installation has completed, click Next to end the installer.</p> 
9	<p>After installation of the software, the computer may have to be restarted. In this case click on the Restart button to restart the computer.</p> 
10	<p>In the Start, Programs folder you will find a shortcuts to Generic User Interface. Click on this shortcut to run Generic User Interface.</p> 

8.2 Register File Path

The Generic User Interface uses the Register Files described in [Register Files](#). When the Generic User Interface has found a module on the chosen communication port and module address it will determine its module type number. The responded module type number will then be used to locate the corresponding register file, and from this it will show control and reading registers.

In the file folder where the Generic User Interface.exe file is installed, there is a file called "Register file path.txt". Open this file and make sure the path is correct for the folder holding the Register files described in [Register Files](#).

The default register file path is C:\Register files, but change this in the text file if the files are located another place.



8.3 Using the Generic User Interface Software

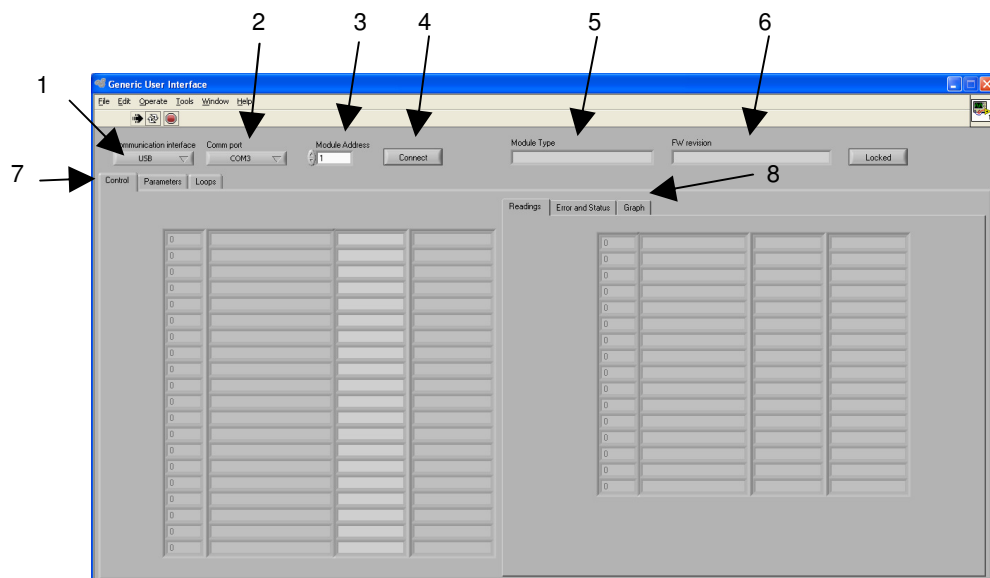
Introduction

This section includes 8 parts, representing the overall control of the user interface:

- **Front Panel : General introduction to the user interface.**
- **Starting Up: How to start up and use the user interface.**
- **Controls: Read/Write registers.**
- **Readings : Read registers.**
- **Error and Status: Error Code and Status Bits.**
- **Graph: Graphical view of register values.**
- **Functions: Upload firmware and download log.**
- **Loops: Loop sequence.**

8.3.1 Front Panel

Below the primary buttons and functions are described for the front panel of the Generic User Interface.



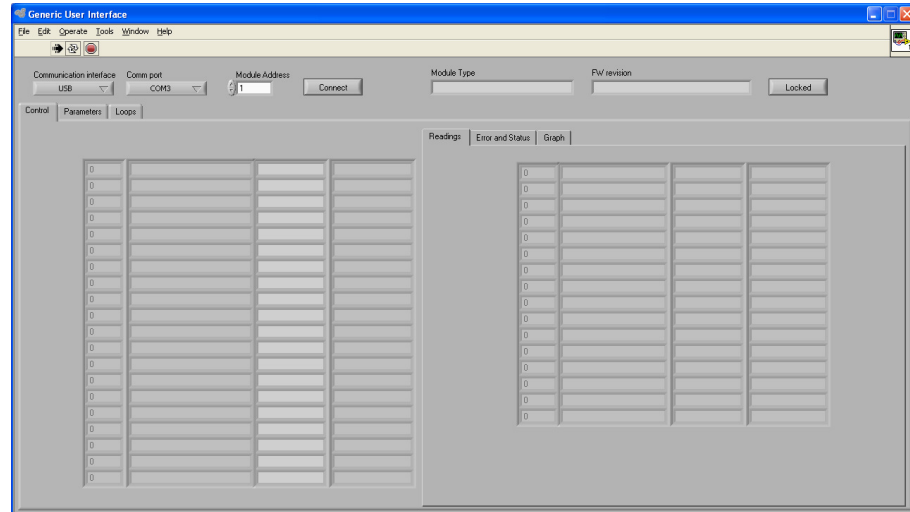
1 – Communication Interface	Choose between USB or Ethernet.
2 – Comm Port or IP Address	<p>For USB, choose the COM-port to communicate on. COM-port marked with *, indicates a known type of communication port, so this is likely the COM-port to choose.</p> <p>For Ethernet, type in the IP address to communicate with.</p>
3 – Module Address	Type in (or use the up/down buttons to select) the module address to communicate with.
4 - Connect	<p>Click on the Connect button to try establish communication between the Generic User Interface and a module/system on the chosen Module Address on the chosen Communication Port.</p> <p>When the Connect button is active, it says Connected. Click again to end and close the communication port.</p>
5 – Module Type	In this text field the Generic User Interface will inform about what module type it is communicating with. This text string comes from the Register File for the actual type of module.
6 – Firmware Revision	In this text field the user interface will inform about the current firmware revision in the module it is communicating with.
7 - Primary Panes	<p>In the upper left corner it is possible to shift between the primary panes of the Generic User Interface.</p> <p>Controls:</p> <ul style="list-style-type: none"> This pane includes all input registers including readouts of output registers. See Controls and Readings for details. <p>Functions:</p> <ul style="list-style-type: none"> On this pane it may be possible to upload new firmware to the module and download log data. See Functions for details. <p>Loops:</p> <ul style="list-style-type: none"> On this pane it is possible to automatically step through a sequence of register values. See Loops for details.
8 - Secondary Panes	With the Primary Pane set to Controls it is possible to chose between different other views of register values. See Error and Status and Graph for details.

8.3.2 Starting Up

Starting Up

Start Generic User Interface via Start - Programs – Generic User Interface or with the Generic User Interface short-cut on the desktop.

Choose between USB or Ethernet as Communication interface.

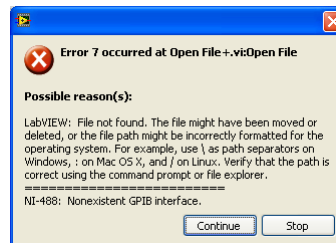


If USB is used, then select the COM port number for the connection (an asterisk next to the COM port in the drop down menu indicates that it is a known Communication interface, so this is likely the COM port to choose). If Ethernet is used, then type in the IP address for the system to communicate with.

Choose the Module Address to communicate with. The Module Address can be changed on-the-fly even if the Generic User Interface is already connected.

Register File Path

Make sure the correct path to the [Register Files](#) are defined in the [Register File Path](#) file. If the Generic User Interface cannot find the register files it will not be able to establish communication and the dialog below will appear.



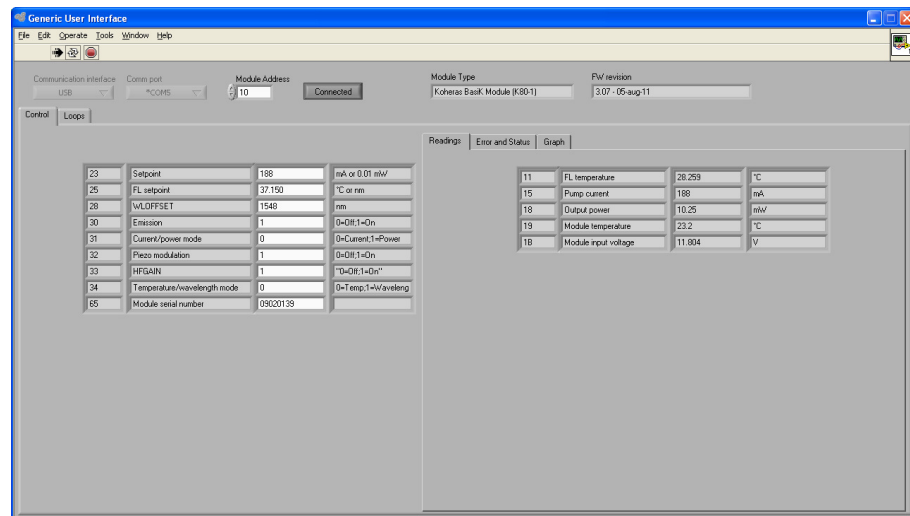
Click Stop and define the correct path in the Register file path.txt file as described in the [Register File Path](#) section.

Click on the Connect button to try to establish communication between computer and a NKT Photonics module/system.

If communication cannot be established, then verify the correct Communication Port and Module Address is used and that the Path field points at the directory with the Register Files.

8.3.3 Controls

In the left side of the Controls window it is possible to read out and write new values to input registers.



The first column of the array informs about the hexadecimal number for the register address. The second column provides a small description of the register and the fourth column provides the unit for the different registers. The content of these three columns is taken directly from the actual Register File.

Register Values

The third column provides the read out from the different registers. The Generic User Interface handles the scaling of the different registers.

Notice

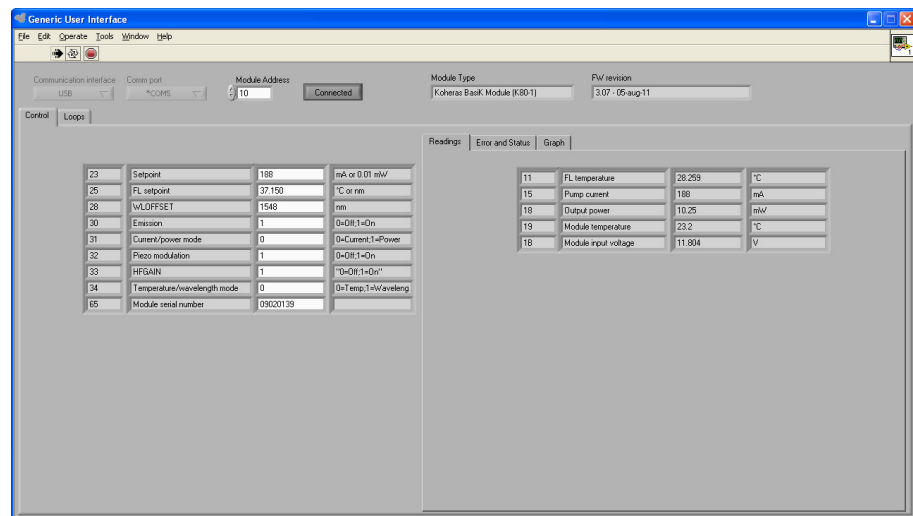
The third column is a string array, even if values are numbers. So the Enter button on the key board is not always the right thing to use when a new value has been typed into the array. Instead click with the mouse on somewhere on the grey back ground for the value to be written to the module.

Warning

If the Module Address is changed on-the-fly, be sure that the content of the array has been updated with the actual content from the new module before a new value is send to the module. Otherwise it may happen that other settings from the previous module are send to the new module. It usually takes a few seconds from the Module Address is changed to the array has been updated when the Minimum Sample Time is set to 1 second (see [Graph](#) for details).

8.3.4 Readings

On the Controls pane, the Readings sub-pane can be selected. On this pane all read-only registers on the chosen module/system is read out.



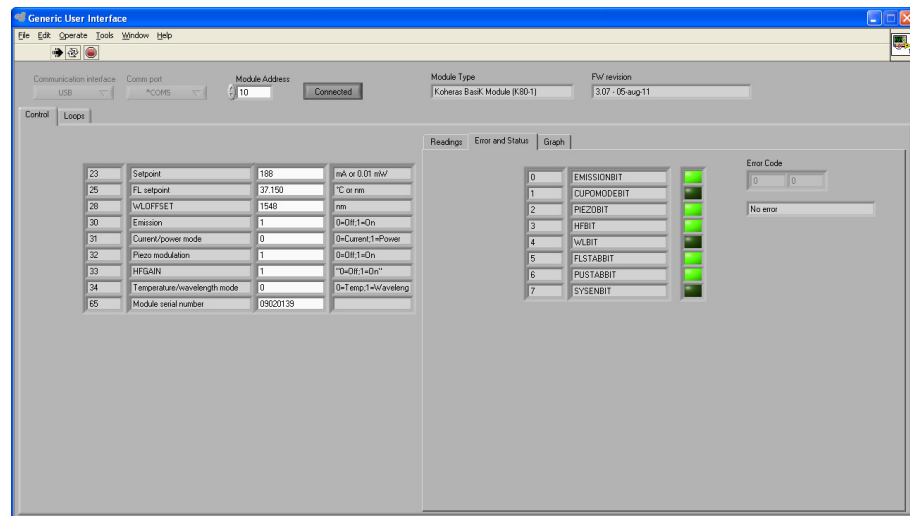
Like for the Control registers, the Reading array contains four columns. The first column shows the hexadecimal number for the register addresses, the second column a short description of the register, the third column the reading from the module/system and the fourth and last column the unit for the values.

Column 1, 2 and 4 is taken directly from the Register File and column 3 are real values read from the module/system.

Like for the Control registers, the Generic User Interface is handling scaling of the registers.

8.3.5 Error and Status

On the Controls pane, the Error and Status sub-pane can be selected. On this pane the status bits and error code from the module can be read.



Status Bits

The Status Bits is shown in an array with three columns. The first column provides the index number for the different Status Bits, the second column a short description of the Status Bits and the last column an indicator on whether the bit is high or low. A dark green color indicates a low bit whereas a light-green color indicates a logical high bit.

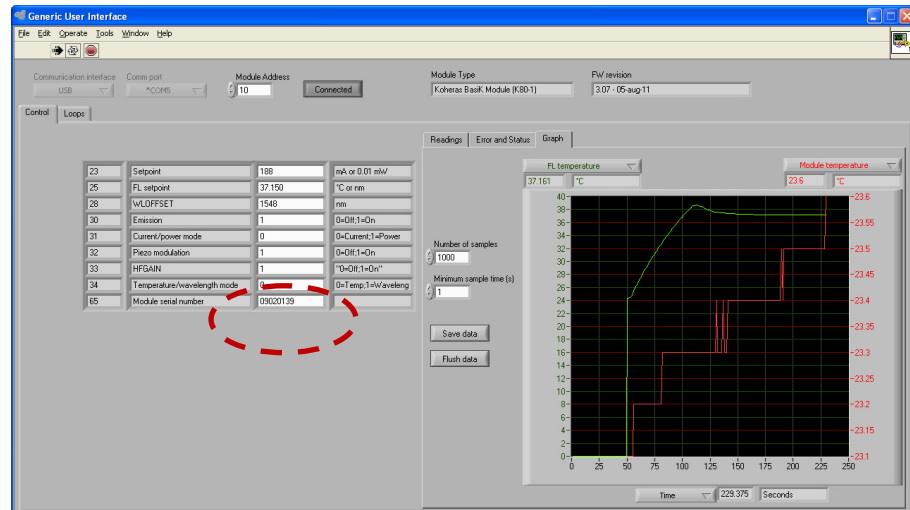
Error Code

If a module has shut down with an Error Code, this can be read out in the Error Code field. The Error Code is an 8-bit number, which is translated into a short description in the text field right underneath the Error Code field.

Description of Status Bits and Error Codes are taken from the Register File.

8.3.6 Graph

On the Controls pane, the Graph sub-pane can be selected. On this pane the content from both the Readings array and the Control array can be viewed.



X-axis

Underneath the graph the parameter that should be used for the X-axis is chosen. Possible parameters are Time, Loop Time, All Readings and All Controls parameters.

If Time is chosen the graph will show a like an oscilloscope with the cursor starting on the left side moving towards the right.

Loop Time is the time counter used for the loop sequences. See [Loops](#) for details.

Readings and Controls parameters depend of course on what type of module/system the Generic User Interface is communicating with.

Next to the X-axis selector the last value is read together with the unit.

Left-hand Y-axis

On top of the graph to the left, the parameter for the left-hand Y-axis is chosen. Exactly the same parameters can be chosen as for the X-axis.

The left-hand Y-axis is written with green text as the gridline for this is green as well as the corresponding curve in the graph.

Right-hand Y-axis

On top of the graph to the right, the parameter for the right-hand Y-axis is chosen. The same parameters as for the X-axis and the left-hand Y-axis can be chosen for the right-hand Y-axis.

The right-hand Y-axis is written with red text, as the corresponding curve and gridline is red.

Views

By using the 3 axis-selectors, it is possible to get not only oscilloscope type of views with the X-axis as a time axis, but also e.g. I-P curves (dependant on modules/systems).

Number of Samples

In this field it is chosen the maximum data points that should be shown in the graph. Right after the Generic User Interface has connected to a module/system there will be no data points to show. As data points are gathered from a module and if the number of data points are about to exceed the Number of Samples the oldest data points will be erased from the computers memory.

Minimum Sample Time

The Minimum Sample Time determines the minimum time between register values are to be updated. It is as it says a minimum value, so the actual time will be slightly larger. Setting the Minimum Sample Time to 0, will make the Generic User Interface

communicate as fast as possible.

Save Data

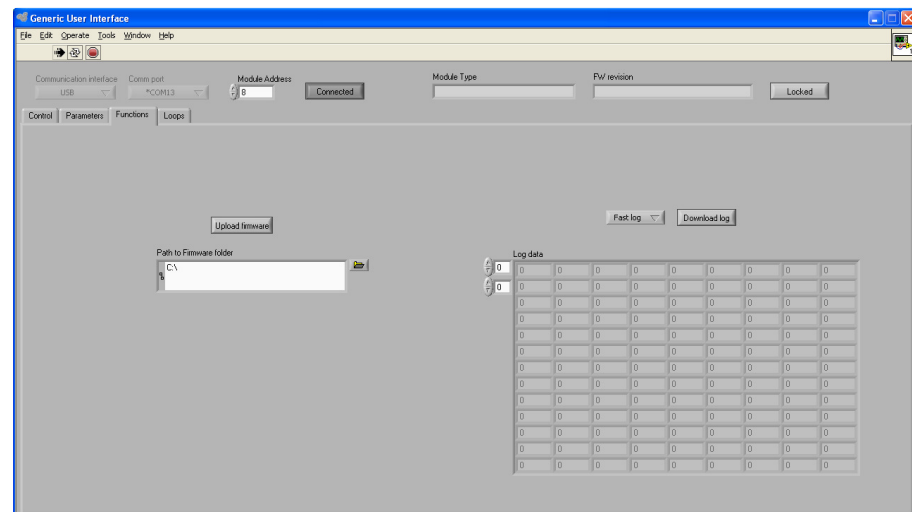
The Save Data button makes it possible to save all recorded data to a tab-delimited ASCII-file. All recorded data means not only what is shown on the graph, but also all other parameters that is not chosen.

Flush Data

Clicking on the Flush Data button will erase all recorded data, i.e. the graph will start all over again.

8.3.7 Functions

For some types of modules/systems it is possible to upload new firmware and/or download log data from the module. If these functions are possible for the module/system the Generic User Interface is communicating with, these will be available on the Functions pane.



Upload Firmware

To upload new firmware to the module the Generic User Interface is connected to, browse for the directory where the desired hex-file to be uploaded is located. Here after click on the Upload Firmware button for the upload to begin. As long as the upload is in progress the button will say Uploading Firmware.

Download Log

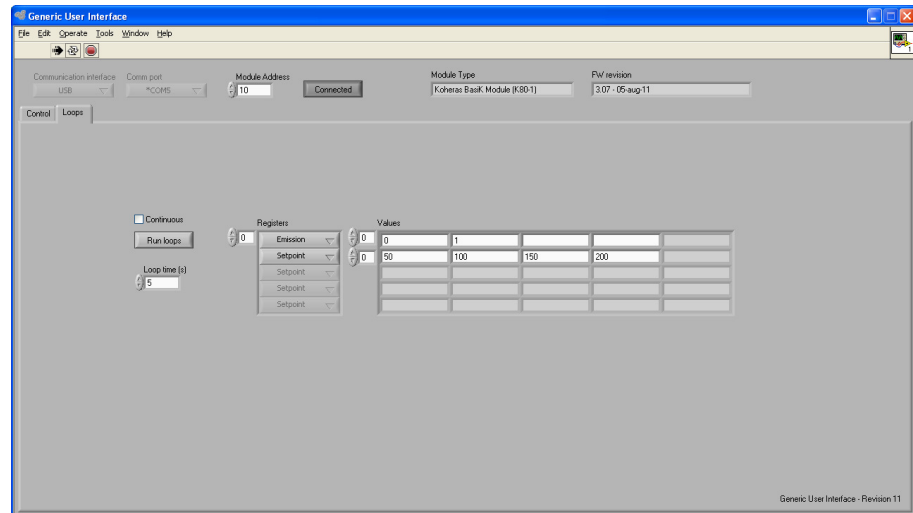
If the module the Generic User Interface is connected to features a module log-function, the log file(s) can be downloaded with the Generic User Interface. If more than one log-file is available the log-file to be downloaded is selected with the selector to the left. Click on the Download Log button to begin the actual download. The log-files can easily become very large, so it can be very time consuming to download these log-files.

8.3.8 Loops

Loops

On the Loops pane it is possible to let the Generic User Interface automatically run through a defined sequence.

The Loop sequence will step through all possible combinations of the chosen parameters.



Registers

In the Registers array the parameters for the sequence to step through is selected. The first defined parameter will be the one parameter that is changed everytime, and the last defined parameter will be the one that is changed the least times.

Values

In the Values array the different setpoints the chosen parameters to run through is defined. All values in the Values array are scaled according to the scaling factors in the Register lists.

It is not required that there should be an equal number of values for all parameters. Cells to the right of the last value can be left blank and they will be ignored.

Loop Time

Loop Time determines the minimum time the Generic User Interface should maintain each state in the sequence.

Continuous

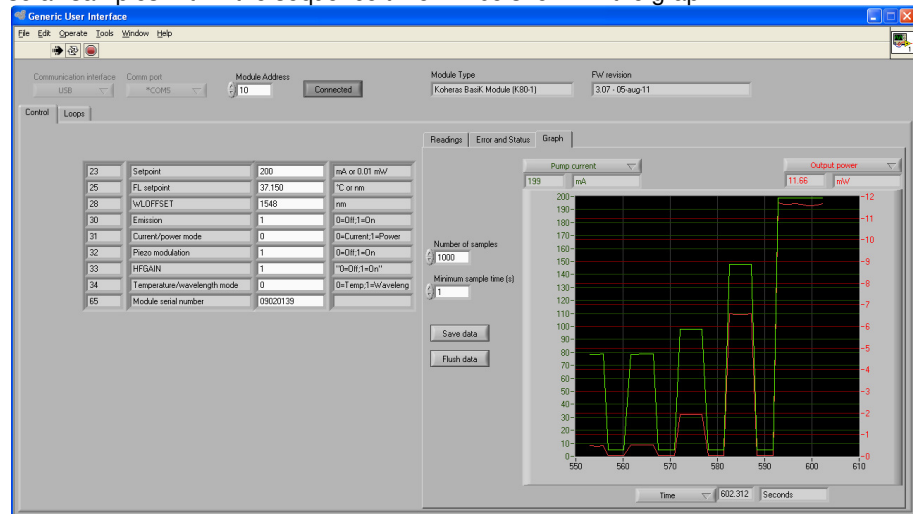
Is the Continuous check mark set, this would make the Generic User Interface run the sequence over and over again.

Run Loops

Click on the Run Loops button to start the sequence. Click on the button again to about the loop sequence if desired without disconnecting the communication between the computer and the module/system.

Graph View

On the Graph window readings from the module/system at the different states of the sequence can be viewed. Just remember to set the Minimum Sample Time to less than the Loop Time and probably also set the Number of Samples to a large number so all samples within the sequence time will be shown in the graph.



NKT Photonics
Blokken 84
3460 Birkerød
Denmark
Phone: +4543483900
Fax: +4543483901
www.nktphotonics.com