Fast Remote Instrument Control with HiSLIP

Application Note

Products:

- R&S®CMW500
- R&S®FSW
- R&S®FPS
- R&S®FSV
- R&S®FSVR
- R&S®ZNB
- R&S®ZNBT
- R&S®ZNC
- R&S®RTO
- R&S®RTE
- R&S®RSC
- R&S®SMW200A
- R&S®SMBV100A
- R&S®SMA100A
- R&S®SMB100A
- R&S®SGS100A
- R&S®SGT100A
- R&S®SGU100A
- R&S®SMC100A
- R&S®SMF100A
- R&S®CMA180

This application note introduces the IVI High Speed LAN Instrument Protocol (HiSLIP) and outlines its main features. HiSLIP is the successor to the VXI-11 LAN remote control protocol. This document also describes guidelines for using this protocol.

J. Engelbrecht, K. Lienhart, G. Kiener
12-Nov-14-1MA208_2e

Application Note

ROHDE&SCHWARZ

Table of Contents

1 Overview

This application note introduces the High-Speed LAN Instrument Protocol (IVI HiSLIP). This protocol was developed by the IVI Foundation[1] and is recommended as LXI HiSLIP Extended Function for LXI based instrumentation[2].

Like GPIB[3] and the LAN-based VXI-11[4] protocol, HiSLIP includes implementation of the IEEE 488.2 communications protocol. HiSLIP includes more IEEE 488.2 protocol features than VXI-11, which provides better GIPB functionality over the LAN network.

The features of the HiSLIP implementation in the latest Rohde & Schwarz instruments is presented. This application note refers to the latest HiSLIP-compatible Virtual Instrument Software Architecture[5] (VISA).

The first part of this document introduces the key features of HiSLIP and a technical comparison with other IEEE 488.2-based communications buses. The second part explains how to set up systems to incorporate HiSLIP into remote control applications. The final part answers frequently asked questions concerning HiSLIP synchronization.

[1] IVI-6.1: High-Speed LAN Instrument Protocol (HiSLIP), http://www.ivifoundation.org/downloads/Class%20Specifications/IVI-6.1_HiSLIP-1.1-2011-02-24.pdf, Retrieved 2012-10-04
[2] LXI stands for LAN eXtensions for Instrumentation, http://www.lxistandard.org/Documents/Specifications/LXI_HiSLIP_Extended_Function_Test_Procedures_v1_01.pdf, Retrieved 2012-10-04
[3] General Purpose Interface Bus (GPIB) according to IEC/IEEE Standard for Higher Performance Protocol for the Standard Digital Interface for Programmable Instrumentation - Part 1: General (Adoption of IEEE Std 488.1-2003), IEEE, Retrieved 2012-10-04
[4] TCP/IP Instrument Protocol Specification VXI-11, http://www.vxibus.org/files/VXI_Specs/VXI-11.zip, Retrieved 2012-10-04
[5] VPP-4.3: The VISA Library, http://www.ivifoundation.org/docs/vpp43.pdf, Retrieved 2012-10-04

2 Introduction to HiSLIP

HiSLIP is the future protocol for TCP/IP-based control of IEEE 488.2 message-based instruments. It includes the conventional features used to control instruments as well as advanced functions. It also offers better performance. HiSLIP can replace GPIB interfaces as well as the VXI-11 and USBTMC[6] protocols. HiSLIP was developed for one-to-one compatibility with VXI-11, which can be replaced easily with a simple modification of the instrument resource string.

2.1 Overview

HiSLIP is a non-proprietary standard approved by the IVI Foundation. The standard is supported by various versions of the VISA I/O library. Consequently, the LXI Consortium adopted HiSLIP as the recommended LAN instrument control protocol for LXI devices. HiSLIP includes the IEEE 488.2 communications features that were missing in the VXI-11 protocol, making it well-suited to replace VXI-11 for the LAN-based control of instruments. In contrast to the VXI-11 protocol, HiSLIP has been developed to work in conventional IPv4 Ethernet networks[7] and IPv6 Ethernet networks. This makes it transparent to the Internet Protocol layer, which is responsible for the virtual addressing of the network components' interfaces.

A selection of IEEE 488.2 protocol features included in HiSLIP:

- Device clear transaction
- Serial poll & status query
- Trigger message
- Service request
- Interrupted error detection via message exchange protocol (IEEE 488.2-compatible)
- Lock management with exclusive and shared locks for controlling multiple virtual instruments

[6] USBTMC is the USB interface based IEC 60488-2:2004
[7] IEEE 802.3-2008 Standard, http://standards.ieee.org/about/get/802/802.3.html, Retrieved 2012-10-04

2.1.2 Setup and Protocol Stack

The SCPI commands for remote control are issued by a remote control application.

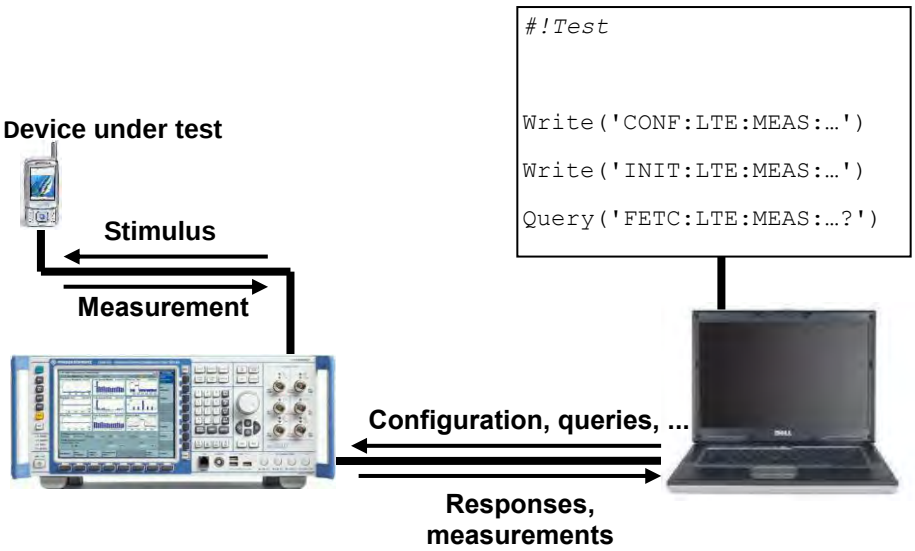

Fig. 1: Basic setup for instrument (server) controlled via PC-based remote controller (PC)

VISA is the standard I/O interface for communicating with the LAN-connected instrument (server) from the application layer of the controller (PC):

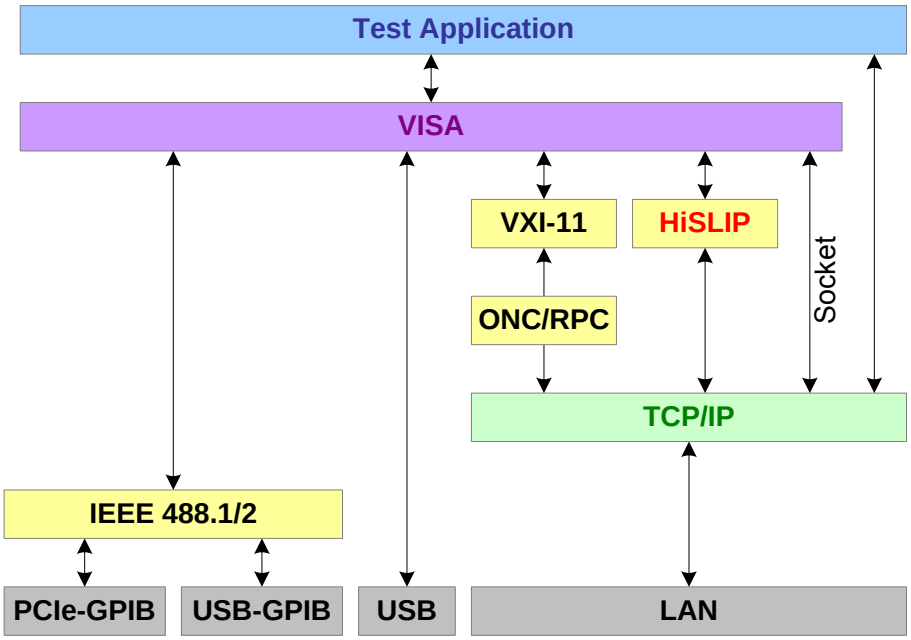

Fig. 2: Extract of the protocol stack and hardware interfaces at the controller for remote instrument control

2.2 Comparison with Other VISA INSTR Protocols

The following table compares several HiSLIP features with the VXI-11 protocol and raw sockets[8] connections via LAN. HiSLIP supports all major protocol functions required to remotely control the instrument (server) and offers greater flexibility:

Comparison of major protocol features			
Feature	**Raw sockets**	**VXI-11**	**HiSLIP**
GPIB emulation	–	✓	✓
Instrument locking (shared & exclusive)	–	(✓)[9]	✓
Support of message exchange protocol	–	✓	✓
IPv6 support	✓	–	✓
High performance	✓	–	✓

Table 1: Comparison of major protocol features

Major improvements of HiSLIP over VXI-11:

- HiSLIP does not use the ONC/RPC[10] protocol in the same way as VXI-11. HiSLIP eliminate superfluous messages and improve performance.
- HiSLIP is more “IT-friendly” than VXI-11, allowing easier data transfer through firewalls and routers. Since only a single port number (IANA[11] TCP port 4880) is registered, firewall configuration is much simpler.
- HiSLIP supports exclusive and shared locks, whereas VXI-11 only supports exclusive locks. A smooth lock mechanism allows locking programs to coexist with lock-unaware programs.

The table below shows the maximum theoretical bandwidth supported by the most common remote control interfaces.

[8] Raw sockets connections allowing direct programming of server via its message channel using the telnet protocol (a socket API). The lack of a control channel creates major limitations in binary transmission, synchronization, service requests, triggering and error handling, e.g. the missing device clear transaction can cause problems.
[9] Supports exclusive locking only.
[10] https://tools.ietf.org/html/rfc1831, Retrieved 2012-10-04
[11] Internet Assigned Numbers Authority, http:://www.iana.org

Bandwidth comparison of interfaces and protocols for remote control			
Interface	**Maximum interface throughput**	**Protocol**	**Typical throughput**
1 Gbit/s LAN	125 Mbyte/s	**HiSLIP**, Raw Sockets	up to 60 Mbyte/s
100 Mbit/s LAN	12.5 Mbyte/s	**HiSLIP**, Raw Sockets	11 Mbyte/s
1 Gbit/s LAN	125 Mbyte/s	VXI-11	34 Mbyte/s
100 Mbit/s LAN	12.5 Mbyte/s	VXI-11	11 Mbyte/s
USB 2.0	60 Mbyte/s	USBTMC	18 Mbyte/s
GPIB-PCI	1.8[12] Mbyte/s	IEEE 488.2	1 Mbyte/s

Table 2: Bandwidth comparison of interfaces and protocols for remote control

Speed comparison with other VISA INSTR protocols

Aside from the previously mentioned HiSLIP features, one major advantage of the protocol is its improved performance. The following table provides a general performance comparison. The test was carried out using a modern desktop PC with a 1 Gbit/s LAN network connection to the R&S®FSW[13] Signal and Spectrum Analyzer.

Speed comparison of interfaces and protocols for remote control[14]						
Commands	**Raw sockets[15]**	**NI GPIB-PCI**	**USBTMC**	**VXI-11**	**HiSLIP**	**VXI-11 vs. HiSLIP**
viReadSTB(…)	n.a.	~190us per call	~305us per call	~300us per call	~140us per call	**~53 % faster**
*OPC?	~180 us per call	~530us per call	~250us per call	~610us per call	~210us per call	**~65 % faster**
Writing 3Mbyte to instrument	No binary transmission	~2800ms per transfer	~590ms per transfer	~250ms per transfer	~29ms per transfer	**~88 % faster**
Reading 3Mbyte from instrument	No binary transmission	~3220ms per transfer	~232ms per transfer	~160ms per transfer	~115ms per transfer	**~28 % faster**

Table 3: Speed comparison of interfaces and protocols for remote control

The throughput between the controller and server (instrument) is limited by various factors and the interactions between these factors, e.g. hard disc speed, network controller, controller (desktop PC) capacity, network topology, router and even the quality of the actual cables.

[12] IEEE 488 interlocked handshake.
[13] www.rohde-schwarz.de/product/FSW.html, firmware version 1.51.
[14] The times per call are averaged over 1000 single time measurements for viReadSTB() and “*OPC?” query commands. The file transfer was repeated five times executed and the time are averaged as well.
[15] The telnet protocol does not support the VISA API function viReadSTB(). Furthermore no binary data transmission is possible.

2.3 HiSLIP Channels

HiSLIP uses a simplified connection model with two TCP connections (port 4880) to a single instrument (server) that is controlled remotely from a controller (PC).

Both the client and the server can store messages in the synchronous buffers and execute them in the given order:
- Data
- DataEND
- Trigger

The asynchronous channel carries GPIB-like meta messages that have to be processed in parallel to the data path:
- Device clear transaction
- Status query & serial polling
- Service request (program interrupt)
- Remote/local instrument control
- Locking and lock information of an instrument
- and others for asynchronous (out-of-band) control commands
- Improved remote/local status control

Both the controller and the instrument (server) handle asynchronous messages with higher priority, processing them before messages received via the synchronous channel.

2.4 HiSLIP Messages and Sequences

2.4.1 Communications Model

HiSLIP data is sent in a “fire and forget” manner with immediate return – unlike VXI-11, whereby each VISA write or VISA read operation is blocked until a VXI-11 device handshake is returned. As a result, it is not possible to guarantee that the instrument has finished (or even started) processing the requested operation before the next request is received. This causes the new request to be stored in the TCP/IP buffers of the instrument (server).

The following figure shows the error-free mechanism for message exchange. No acknowledgement messages are transferred between the controller and server (no handshake).

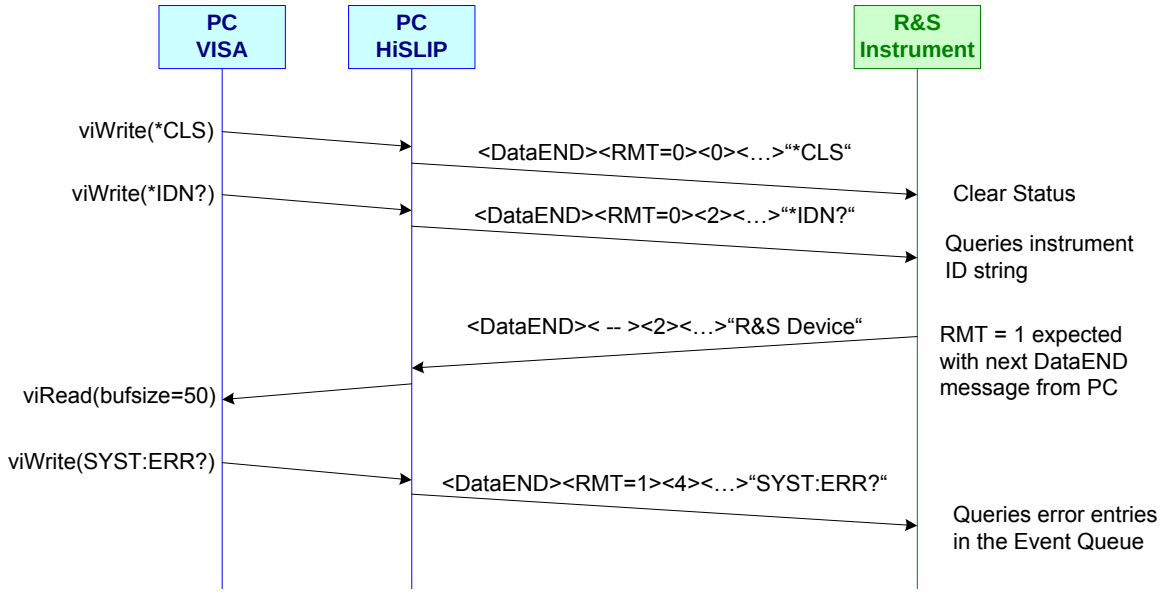

Fig. 3: Simple message sequence between controller (PC) and instrument (server)

As a compensation for the acknowledgement messages of the VXI-11, the RMT bit (remote message terminator) is introduced. The RMT bit in the messages indicates if this is the first Data, DataEND or Trigger message since the controller (PC) has finished reading completely a previous data message from the instrument (server). Thereby the instrument can detect if the controller has received the instrument's DataEND message before the controller has sent its next message. The instrument expects RMT = 1 after having sent its DataEND message.
This mechanism saves time compared to usual handshake message sequences as used in the VXI-11 protocol.
The controller increments the Message ID used as sequence number with each Data, DataEND or Trigger message sent.

2.4.2 Operating Modes

HiSLIP supports two different operating modes.

2.4.2.1 Overlapped Mode

This mode is similar to raw sockets mode. Input and output data and trigger messages are arbitrarily buffered between the controller and the controlled instrument. Their responses are returned in the same order as the queries were sent.

2.4.2.2 Synchronized Mode

This mode closely resembles the requirements of the IEEE 488.2 message exchange protocol for detecting interrupted errors. In general, the synchronized mode and message exchange protocol ensure the instrument (server) responds only to the last query. An incoming response is discarded if it is not associated with the last query that was sent. The synchronized mode ensures compatibility with GPIB, VXI-11 and USBTMC instruments and is the mode supported by the Rohde & Schwarz instruments.
Reading the VISA attribute VI_ATTR_TCPIP_HISLIP_OVERLAP_EN returns the protocol mode of the current session (VI_FALSE for Rohde & Schwarz instruments).

Interrupted error detection in synchronized mode
Interrupted errors are detected and processed on the controller side or on the instrument side. If the controller sends two queries to the instrument and receives the DataEND response to the first query after the second query has been sent, any data responses already buffered in the controller are cleared, which generates an interrupted protocol error. If the instrument receives a second query from the controller before it has finished sending the response to the first query, or if the second query is sent before the controller has received the complete response to the first query, the instrument stops responding to the first query, generates an interrupted error and sends an interrupted message to the controller. Then the instrument sends the controller the response to the second query.
The following figure shows an example where the second query of the controller and the first response of the instrument cross “in-flight”.

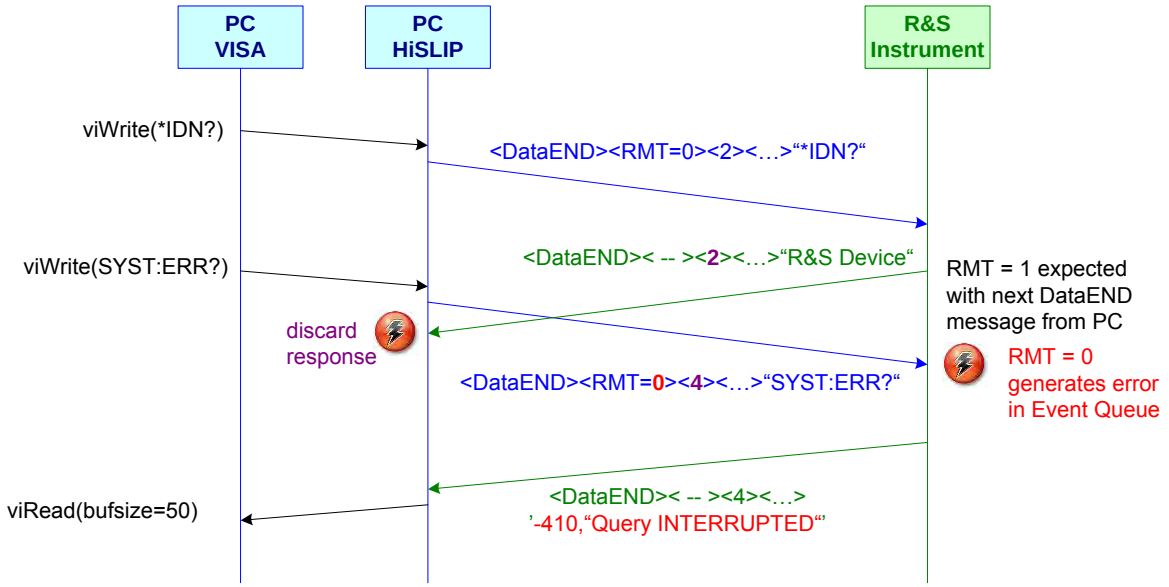

Fig. 4: Message sequence with interrupted error

The controller detects an interrupted error by comparing the sequence number of its last query with the sequence number of the instrument’s response. Both sequence numbers should be equal. After detection of the interrupted error, the controller discards the response.

The instrument detects the interrupted error due to RMT = 0 in the controller's DataEND message: The instrument sets its RMT expectation value to 1 when sending its DataEND message. RMT = 1 would indicate that the controller had received the instrument's DataEND message before sending the second query; RMT = 0 indicates that the controller had not yet read the first response from the TCP/IP buffers. After detection of the interrupted error, the instrument reports the error. Since the instrument has already finished the first response it takes no additional action. The instrument responds to the last query from the controller in the normal way even with the correct response "-410,"Query Interrrupted"".

One possible variation occurs when the instrument detects the wrong RMT value while responding to a previous query. As a result, the instrument would stop responding and would send an interrupted message to the controller, which would then prompt the controller to discard the incomplete response.

2.5 Remote/Local Instrument Control

The remote/local instrument control prevents manual operation of the instrument from its front panel when the instrument is being controlled remotely with SCPI commands. The local key[16] allows to regain access to the instrument to manual operation, while an instrument is still connected to a valid remote control session.

HiSLIP can control the remote/local state of an instrument in a GPIB mnemonic similar to the function call viGpibControlREN(…)[17]. The remote-local commands are sent on the asynchronous channel for GPIB as meta-messages.

For practical reasons, three VISA modes from viGpibControlREN(…) are also specified for HiSLIP:

- Disable front panel operation, controlled by VI_GPIB_REN_ASSERT_ADDRESS mode. Local lockout key is functional.
- Disable front panel operation and perform a local lockout, controlled by VI_GPIB_REN_ASSERT_ADDRESS_LLO mode. Local lockout key is blocked.
- Enable front panel operation and undo a possible local lockout, controlled by VI_GPIB_REN_DEASSERT mode.

It is important to remember that without instrument locking, SCPI commands will always be accepted by the instrument.

[16] IEEE 488 defines the local key in the instrument to regain front panel access, when the front panel access has been disabled via a remote interface.

[17] http://www.ivifoundation.org/docs/vpp43.pdf, p.273f. IVI Foundation, Retrieved 2012-10-04

2.6 Instrument Locking

HiSLIP supports exclusive and shared locking as specified according to the VXIPNP VISA standard[18]. This locking mechanism affects all synchronous and asynchronous commands sent to the instrument after the VISA session has been initialized. Similar to the remote/local state control, this feature allows access for controlling the remote control interfaces of the instrument. When a client closes its session, all locks belonging to this session are released.

Possible configurations:

- **Server is not locked**
  - This allows any client to communicate with the server (instrument).
  - Any client can request a shared or an exclusive lock.
- **Client holds an exclusive lock**
  - This allows one-to-one communications between the server and the client holding the exclusive lock.
  - No other clients are allowed to send messages to the locked instrument.
- **Multiple clients are holding a shared lock[19]**
  - All clients holding this single shared lock are allowed to communicate with the instrument.
  - One client holding this single shared lock is allowed to request an exclusive lock. The server grants the exclusive lock one client at a time. While the exclusive lock is active, no other client holding a shared lock is allowed to send commands to the instrument.
  - No other clients are allowed to send messages to the locked instrument.

If a client holds a shared or exclusive lock, the following applies for clients that do not hold a lock:

- Any lock request will either be granted access if the requested lock becomes available or will timeout after the specified waiting period.
- A device clear transaction is executed immediately. Only the parameters applying to this session are affected; server parameters which also apply to other sessions are not affected.
- Messages sent over the synchronous channel to the server remain in the input buffer. This buffer is handled by the TCP behavior to avoid buffer overflows.
- Any synchronous or asynchronous transactions in progress are completed as usual, causing all data traffic to be transferred from the server to the client.
- Asynchronous service requests are sent to the client in the normal way.

[18] http://ivifoundation.org/docs/vpp43.pdf, p37ff, Retrieved 2012-10-04.
[19] A shared lock is identified by a string.

2.7 HiSLIP Protocol Support in the VISA Library

VISA is the standard I/O interface for accessing the instrument from the application layer on the controller. The following VISA versions support the HiSLIP protocol:

- NI-VISA: supports HiSLIP since version 5.1.0 (IPv4).
  NI-VISA versions 5.2.0 and later is recommended to achieve maximum performance.
- Agilent IO Libraries Suite: supports HiSLIP (IPv4 and IPv6) versions 16.2 and later.

3 Getting Started

A main objective of HiSLIP was to minimize integration testing complexity for existing applications. HiSLIP can coexist or even replace VXI-11 in current and future remote control applications.

3.1 Using HiSLIP in Remote Instrument Control Applications

One key requirement during the development of HiSLIP was to maximize the simplicity of migrating remote control software from one of major remote control interface (GPIB or VXI-11) to the LAN-based HiSLIP for end users. The result of these efforts was a simple, four-step setup procedure for HiSLIP:

1. **Check availability of HiSLIP for remote control channels of the server (instrument)**
   Check if the instrument supports HiSLIP. The following Rohde & Schwarz instruments support HiSLIP:
   - R&S®CMW500 (since firmware version 3.x.y)
   - R&S®ZNB (since firmware version 1.63)
   - R&S®ZNC (since firmware version 1.63)
   - R&S®FSW (since firmware version 1.60)
   - R&S®SMBV100A (since firmware version 2.20.360.114)
   - R&S®SGS100A (since firmware version 2.20.417.20)
   - R&S®SMA100A (since firmware version 2.20.470.18)
   - R&S®SMB100A (since firmware version 2.20.382.35)
   - R&S®RSC (since firmware version 1.34)

2. **Enable HiSLIP on your VISA client (remote control PC)**
   Your controller (remote control PC) must also support HiSLIP. There are currently two possible configurations for Windows®-based clients:
   a) National Instruments VISA
      NI-VISA versions 5.2.0 and later is recommended to achieve maximum performance.
   b) Agilent IO Libraries Suite
      HiSLIP for IPv6 and IPv4 are supported by Ag-VISA versions 16.2 and later.

   Now your client and server are ready to communicate with your instrument via HiSLIP.
   For information regarding the Rohde & Schwarz instrument driver HiSLIP support, please refer to the latest version of the history release notes.

3. **Change the VISA resource string**
   Find out the IP address or hostname (alias for the IP address of the instrument) of the instrument. The address structure for HiSLIP[20] is as follows:

   TCPIP[Interface number]::<IP address>|<Hostname>::hislip0[,<port>][::INSTR]

   The shortest possible HiSLIP VISA resource string would resemble the following:

   TCPIP::CMW500-100052**::hislip0**

   If constant instrument VISA aliases are defined in the Measurement & Automation Explorer software or in the Agilent Connection Export software please refer to the corresponding software documentation for further information.
4. **Send SCPI commands to your instrument**
   The easiest way to communicate with an instrument is to use the interactive tools provided by National Instruments Measurement & Automation Explorer or using the Agilent Connection Expert.
   At this point, your remote control application or your test system infrastructure is ready to be configured for use with the instrument.

   After successfully establishing the connection to the instrument, single commands can be sent and responses can be easily read from the instrument.

   This screenshot shows the scenario with the National Instruments software tools:

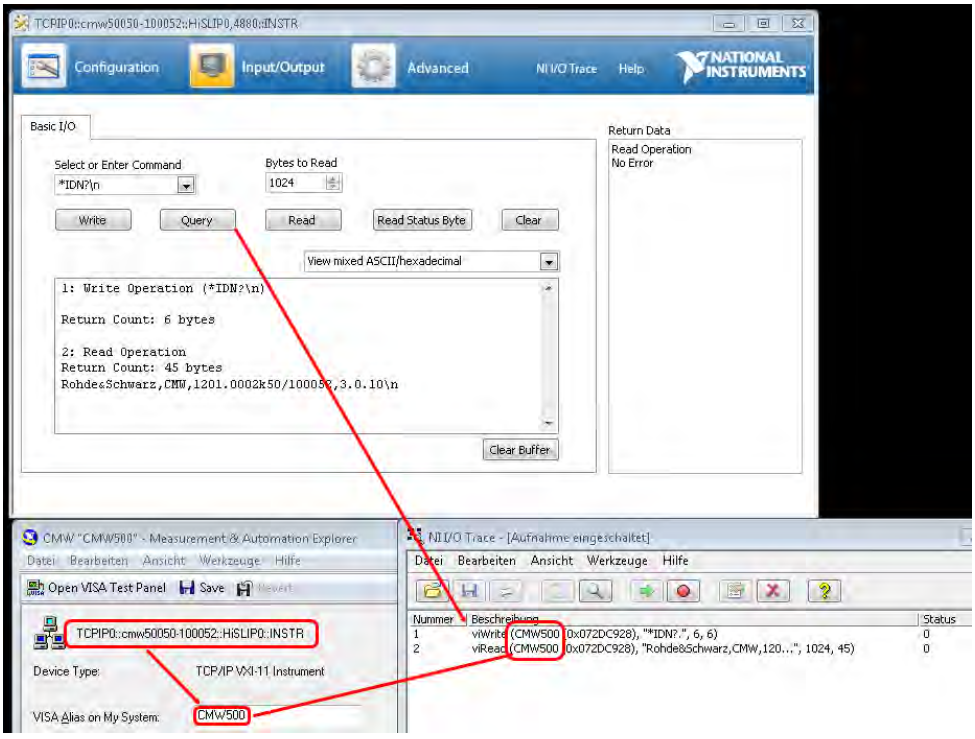

Fig. 5: Example scenario to send single SCPI commands to an instrument via HiSLIP using a VISA alias name

[20] In comparison, the shortest possible VXI-11 VISA resource string looks very similar: “TCPIP::CMW500-100052”

3.2 Example with C Programming Language

Appendix 6.1 contains a small sample of source code. This demonstrates a speed comparison of HiSLIP, GPIB, VXI-11 and raw socket connections.

4 FAQ

The following section provides an overview on how to successfully begin development with the HiSLIP protocol and how to migrate from VXI-11 to HiSLIP.

For information regarding the SCPI programming, please refer to the application notes “1EF62: Hints and Tricks for Remote Control of Spectrum and Network Analyzers”, “1GP79: Top Ten SCPI Programming Tips for Signal Generators” or “1CM100: Optimizing the Performance of Multi Eval. List Mode Tests” (References).

4.1 Synchronizing Remote Control Applications

Remote control applications have to be synchronized with the remote instrument. This is very important. Otherwise, erroneous measurement data and/or undesirable, unstable program behavior may occur.
Some T&M instruments involve time-consuming operations, e.g. because measurements have to be calculated from large data sets or switching between different configurations is required. Special SCPI commands are provided for this reason. They are referred to as “overlapping SCPI commands”. This means that the execution and processing of a command takes a considerable amount of time. The instrument accepts and processes queries in addition to set commands while the overlapping command is being processed[21]. This means that the “operation complete” state of these SCPI commands has to be sent to the controller (e.g. PC remote control application).

For information regarding the remote control instrument synchronization, please refer to the latest version of the instrument manual. For further information please refer to the VISA programming guide.

4.2 Possible Race Conditions in a Remote Control Application

Race conditions do not occur when the overlapping SCPI command synchronization is realized by the “*OPC?” query command or the “*WAI” control command.

[21] If the set command is affecting the currently running operation, the operation is restarted. For example, perform a trace measurement with 100 sweeps on the R&S®FSW Signal and Spectrum Analyzer. The command that starts this measurement is overlapping. During these 100 sweeps, the center frequency, for example, can be changed via remote control. In this case, the sweep count restarts with one again using the modified parameter set.

HiSLIP supports two paths for the data and the control channel. This is similar to the GPIB (IEEE 488.2) interface. In HiSLIP the control channel has a higher execution and processing priority as the data channel. The VXI-11 protocol also uses two channels for data and control messages. The commands are synchronized to a certain extent using RPC[22] acknowledgement messages (refer to chapter 2.4.1).

How does the parallel channels of HiSLIP result in a race condition?

At first glance, these simple measurement steps for a spectrum analyzer should not cause any significant problems:

```
/*
Reset the spectrum analyzer and configure the
status subsystem to allow operation complete
synchronization (*OPC) and service requests (SRQ)
indication in the STB register
*/
#1 LOOP_BEGIN_MEASUREMENTS
#2     *CLS;INIT:IMM;*OPC\n
#3     WHILE MEASUREMENT in PROGRESS
#4            viReadSTB();            //repeated STB byte reads
                                       //to check if measurement is
                                       //in progress
#5     FETCH:MEAS:DATA?\n
#6 LOOP_END
```

This is the case with VXI-11 connections, but this code might result in erroneous measurement data when using HiSLIP[23].
The problem occurs if this example loop is executed more than once (step 1 to 6). The instrument state before the second iteration (step #2) is as the following: one valid measurement was performed by step #2 in the first iteration of the measurement loop. Therefore valid measurement data is still available in the output buffer of the instrument along with pending information about the operation complete status in the status subsystem.

[22] RPC stands for remote procedure calls, refer to http://en.wikipedia.org/w/index.php?title=Remote_procedure_call&oldid=509236106, Retrieved 2012-10-04.
[23] This race condition does not occur with GPIB (IEEE 488.2) connections in the field.

On a fast controller line #2 and line #4 are sent to the instrument almost at the same time[24]. In the second iteration of the measurement loop this lead to the situation that the command line #1 (cleaning the status subsystem[25] and starting the measurement with operation complete synchronization) will be processed by the instrument after the processing the viReadSTB() query. This query returns the “operation complete” status information from the first run of the measurement loop. This allows the program to immediately continue reading measurement values. The measurement data can be a result of the first measurement, because in some cases the instrument is not even finished with the processing of the command sequence sent over the data channel (line #2).

One possible method is to perform synchronization by ensuring there is no remaining data in the status subsystem after the first run. This would result in the following:

```
/*
Reset the spectrum analyzer and configure the
status subsystem to allow operation complete
synchronization (*OPC) and service requests (SRQ)
indication in the STB register
*/
#1 LOOP_BEGIN_MEASUREMENTS
#2     *CLS;INIT:IMM;*OPC;*STB?\n
#2a    viRead();                       //first STB byte read to check if
                                       //measurement is in progress
#3     WHILE MEASUREMENT in PROGRESS
#4            viReadSTB();             //repeated STB byte reads
                                       //to check if measurement is
                                       //in progress
#5     FETCH:MEAS:DATA?\n
#6 LOOP_END
```

This allows to synchronize the clearing of the status subsystem. After reading the “*STB?” query the instrument processed the line #2 completely . The query allows to synchronize the “*CLS” (clean status subsystem), to the start of the measurement and all following read queries.

[24] The commands in line #2 and #5 are transmitted over the HiSLIP synchronous data channel, whereas the command in line #4 is transmitted over the HiSLIP control channel. The commands in the control channel are executed with a higher priority (refer to section 2.3)

[25] For more information on the status subsystem, refer to the instrument manual available at:
http://www.rohde-schwarz.com/en/service_and_support/Downloads/Manuals/

5 References

- IVI High-Speed LAN Instrument Protocol, IVI Foundation, http://www.ivifoundation.org/downloads/Class%20Specifications/IVI-6.1_HiSLIP-1.1-2011-02-24.pdf, Retrieved 2012-10-04
- TCP/IP Instrument Protocol Specification VXI-11, VXIbus Consortium, http://www.vxibus.org/files/VXI_Specs/VXI-11.zip, Retrieved 2012-10-04
- Automated Measurement Control, John M. Pieper, Rohde & Schwarz
- 1EF62: Hints and Tricks for Remote Control of Spectrum and Network Analyzers, Johannes Ganzert, Application Note, http://www.rohde-schwarz.com/appnote/1ef62, Retrieved 2012-10-04
- 1GP79: Top Ten SCPI Programming Tips for Signal Generators, Caroline Tröster, Application Note, http://www.rohde-schwarz.com/appnote/1gp79, Retrieved 2012-10-04
- 1CM100: Optimizing the Performance of Multi Eval. List Mode Tests, Klaus Lienhart, Application Note, http://www.rohde-schwarz.com/appnote/1cm100, Retrieved 2012-10-04

6 Appendix

6.1 Example

Example source code to measure the remote control performance for short commands:

```
/**
  Speed test
  32-bit only (-> due to VISA framework limitation on Mac OS X!)
  Note: Please do not forget compiler optimization
  @JE, R&S
 **/

#include <stdio.h>
#include <stdint.h>
#include <sys/types.h>

/*****************************************************************************/
//Utility routines

#if defined(_MSC_VER)
//Windows platform!
#include <visa.h>
#include <windows.h>
#include <time.h>
#if defined(_MSC_VER) || defined(_MSC_EXTENSIONS)
  #define DELTA_EPOCH_IN_MICROSECS  11644473600000000Ui64
#else
  #define DELTA_EPOCH_IN_MICROSECS  11644473600000000ULL
#endif

struct timezone
{
  int  tz_minuteswest; /* minutes W of Greenwich */
  int  tz_dsttime;     /* type of dst correction */
};

int _gettimeofday(struct timeval *tv, struct timezone *tz)
{
  FILETIME ft;
  unsigned __int64 tmpres = 0;
  static int tzflag;

  if (NULL != tv)
  {
    GetSystemTimeAsFileTime(&ft);

    tmpres |= ft.dwHighDateTime;
    tmpres <<= 32;
    tmpres |= ft.dwLowDateTime;

    /*converting file time to unix epoch*/
    tmpres -= DELTA_EPOCH_IN_MICROSECS;
    tmpres /= 10;  /*convert into microseconds*/
    tv->tv_sec = (long)(tmpres / 1000000UL);
    tv->tv_usec = (long)(tmpres % 1000000UL);
  }

  if (NULL != tz)
  {
    if (!tzflag)
    {
```

```
      _tzset();
      tzflag++;
    }
    tz->tz_minuteswest = _timezone / 60;
    tz->tz_dsttime = _daylight;
  }

  return 0;
}

#else
//Mac OS X platform!
#include <VISA/VISA.h>
#include <sys/time.h>
#endif


double getwalltime()
{
    struct timeval tp;
    double sec, usec;
    // Time stamp before the computations
    _gettimeofday( &tp, NULL );
    sec   = (double)tp.tv_sec;
    usec  = (double)tp.tv_usec/1E6;

    return (double) sec + usec;
}

/*****************************************************************************/
//Measurement routines
#define BUFFER_SIZE 4096
#define MEASUREMENT_CYCLES 1000
#define MUTATION 2
#define SCPI_MARKER_SETUP_INIT "*RST;*WAI;:INIT:CONT OFF;:INIT:IMM;:CALC:MARK1
ON;*OPC?\n"
#define SCPI_MARKER_QUERY ":CALC:MARK1:X %.f;*WAI;:CALC:MARK1:Y?\n"
#define SCPI_OPC_QUERY "*OPC?\n"

int open_resource(ViSession rm, char* resource, ViPSession io)
{
    ViInt32 ret = VI_ERROR_SYSTEM_ERROR;
    char buf[BUFFER_SIZE];
    buf[0] = '\0';

    if(resource == NULL)
        return ret;

    ret=viOpen(rm, resource, VI_NULL, VI_NULL, io);
    if (ret!=VI_SUCCESS)
    {
        printf("Open VISA resource failed!\n");
        return ret;
    }

    ret=viLock(*io, VI_EXCLUSIVE_LOCK, 0, "", VI_NULL);
    if (ret!=VI_SUCCESS)
    {
        printf("Exclusiv locking of VISA resource failed!\n");
        return ret;
    }

    //socket test
    viSetAttribute(*io, VI_ATTR_TERMCHAR_EN, VI_TRUE);
```

```
    viQueryf(*io, "*IDN?\n", "%s", buf);
    printf("\n*********\n");
    printf("Resource open and locked: %s\n", buf);

    return ret;
}

void close_resource(ViSession io)
{
    ViAccessMode val = VI_NO_LOCK;
    while(TRUE)
    {
    viGetAttribute(io, VI_ATTR_RSRC_LOCK_STATE, &val);
    if( val != VI_NO_LOCK )
            viUnlock(io);
        else
            break;
    }

    viClose(io);
}

void print_report(char *resource, char* command, double total_time)
{
    printf("*********\n");
    printf("VISA Resource: %s\n", resource);
    printf("Query: %s\n", command);
    printf("Total time: %f\n", total_time);
    printf("Measurement cycles: %d\n", MEASUREMENT_CYCLES);
    printf("Average query time: %f\n\n", total_time/MEASUREMENT_CYCLES);
}

int performance_test_query(ViSession rm, char* resource, char* command)
{
    ViSession io = VI_NULL;
    int i;
    int ret = VI_ERROR_SYSTEM_ERROR;
    double start;
    double end;
    double total_time = .0;
    double temp;
    char rd_buffer[BUFFER_SIZE];
    ViUInt16 unused_val_stb  = 0;

    rd_buffer[0] = '\n';


    if(resource == NULL)
    {
        printf("ERROR: Resource must not be empty!\n");
        return ret;
    }

    if(command == NULL)
    {
        printf("ERROR: Command must not be empty!\n");
        return ret;
    }

    //open resource for query speed test
    ret = open_resource(rm, resource, &io);
    if(ret != VI_SUCCESS)
        return ret;

    //do the measurement n-times
    for (i=0; i<MEASUREMENT_CYCLES; i++)
    {
        //start time
```

```
            start=getwalltime();

            //for speed test viReadSTB() query speed
            //test insert your code here
            //viReadSTB(io, &unused_val_stb);

            //speed test query command
            viQueryf(io, command, "%s\n", rd_buffer);

            //stop time
            end=getwalltime();

            //sum up milliseconds
            temp = end-start;
            if(temp>=0.){
                total_time += temp;
            }
            else{
                printf("ERROR: Time measuerment failed, start time bigger than stop time");
                total_time = 0;
                return ret;
            }
    }

    //close the resoucre
    close_resource(io);

    //calculate average time
    print_report(resource, command, total_time);

    return VI_SUCCESS;
}

int performance_test_sa_measurement(ViSession rm, char* resource)
{
    ViSession io=VI_NULL;
    int ret = VI_ERROR_SYSTEM_ERROR;
    int i;
    char rd_buffer[BUFFER_SIZE];
    double centerfreq[MUTATION];
    char wrt_buffer[MUTATION][BUFFER_SIZE];
    double start;
    double end;
    double total_time = .0;
    double temp;

    rd_buffer[0] = '\n';
    centerfreq[0] = 0.8e9;
    centerfreq[1] = 1.2e9;

    sprintf(wrt_buffer[0], SCPI_MARKER_QUERY, centerfreq[0]);
    sprintf(wrt_buffer[1], SCPI_MARKER_QUERY, centerfreq[1]);

//open resource for query speed test
    if(resource == NULL)
    {
        printf("ERROR: Resource must not be empty!\n");
        return ret;
    }

    ret = open_resource(rm, resource, &io);
    if(ret != VI_SUCCESS)
    {
        return ret;
    }

    //setup instrument and initate measurement
    viQueryf(io, SCPI_MARKER_SETUP_INIT, "%s\n", rd_buffer);
```

```
    //do the measurement n-times
    for (i=0; i<MEASUREMENT_CYCLES; i++)
    {
    //start time
    start=getwalltime();

        if(i%2)
            //set marker to f1 and query
            viQueryf(io, wrt_buffer[0], "%s\n" ,rd_buffer);
        else
            //set marker to f2 and query
            viQueryf(io, wrt_buffer[1], "%s\n", rd_buffer);

        //stop time
        end=getwalltime();

        //sum up milliseconds
        temp = end-start;
        if(temp>=0.){
            total_time += temp;
        }
        else{
            printf("ERROR: Time measuerment failed, start time bigger than stop time");
            total_time = 0;
            return ret;
        }
    }

    //close the resoucre
    close_resource(io);

    //calculate average time
    print_report(resource, SCPI_MARKER_QUERY, total_time);

    return VI_SUCCESS;
}

int main (int argc, const char * argv[])
{

    ViSession rm;
    int ret=VI_SUCCESS;
    char resource_gpib[]   = "GPIB::20";
    char resource_vxi[]    = "TCPIP::10.110.10.216::instr";
    char resource_hislip[] = "TCPIP::10.110.10.216::hislip0";
    char resource_socket[] = "TCPIP::10.110.10.216::5025::SOCKET";
    char command[] = SCPI_OPC_QUERY;

    if (MEASUREMENT_CYCLES<1)
    {
        printf("\nSet the amount of measurement cycles <0\n");
        return VI_ERROR_SYSTEM_ERROR;
    }

    //open the local VISA resource manager
    if(VI_SUCCESS != viOpenDefaultRM(&rm))
    {
        printf("\nOpen VISA resource manager failed!\n");
        return VI_ERROR_SYSTEM_ERROR;
    }

    //gpib test
    performance_test_query(rm, resource_gpib, command);
    performance_test_sa_measurement(rm, resource_gpib);
    //VXI-11 test
    performance_test_query(rm, resource_vxi, command);
    performance_test_sa_measurement(rm, resource_vxi);
    //HiSLIP test
```

```
    performance_test_query(rm, resource_hislip, command);
    performance_test_sa_measurement(rm, resource_hislip);
    //SOCKET test
    performance_test_query(rm, resource_socket, command);
    performance_test_sa_measurement(rm, resource_socket);

    //close the local VISA resource manager
    viClose(rm);

    return ret;
}
```