ICS ELECTRONICS
division of Systems West Inc.

IEEE 488 APPLICATION BULLETIN

SRQ HANDLING WITH A VXI-11 INTERRUPT CHANNEL ON ICS's MODEL 8065 ETHERNET-TO-GPIB CONTROLLER

INTRODUCTION

In GPIB systems, the SRQ (ServiceReQuest) signal is used by a GPIB device to notify the GPIB Controller that the device requires servicing. SRQs are traditionally used to inform the user of a completed task, data ready, an error condition or some other condition that the user has enabled. SRQ handling depends upon the programming language and operating system. Some languages and systems have an On-SRQ capability to jump to an interrupt routine. Others require that the program wait for the SRQ line to be asserted or periodically test the state of the SRQ line. In others, the user can create an 'ibnotify' routine to set a flag or handle the interrupt.

This need to use SRQ notification still exists within VXI-11 systems, but SRQs and interrupt handling must be handled in an entirely different way. This is due to VXI-11 communication being via Ethernet rather than a GPIB bus. The user can still implement a polling method that periodically samples the state of the SRQ line. When it becomes active, the program then has to Serial Poll all of the devices that are enabled to generate SRQs to find the device that is requesting service. This method works okay with one or two devices but becomes slow and loads the network with extra traffic when applied to a large number of devices.

THE VXI-11 METHOD

The VXI-11 Specification provides a Reverse Interrupt Channel that notifies the user when an instrument is requesting service and can be used to identify the requesting instrument. The problem is that the VXI-11 specification discusses the Reverse Interrupt Channel concept at numerous points throughout the Specification and does not give the reader a good comprehensive description of the Reverse Channel operation. This Application Note is intended to give the user the necessary guidelines to implement a Reverse Interrupt Channel in his operating system. An example is not provided since each operating system and programming language has different constraints that limit an example's usefulness and portability.

VXI-11 REVERSE INTERRUPT CHANNEL

The VXI-11 Specification defines the Interrupt Channel in the Connection Model section, Section B2, of the specification.

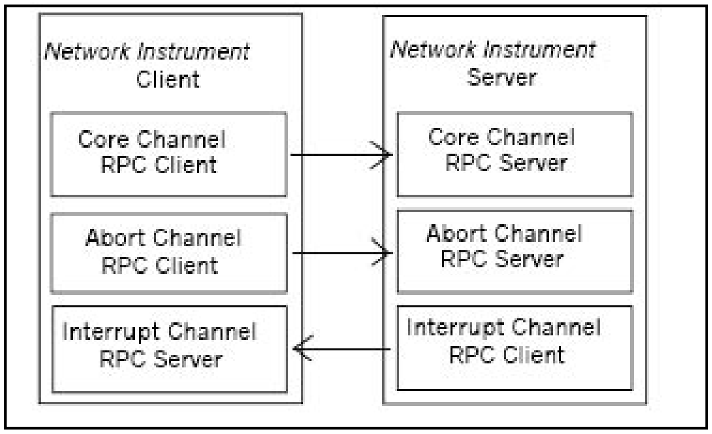
Network Instrument Client
Core Channel RPC Client
Abort Channel RPC Client
Interrupt Channel RPC Server
Network Instrument Server
Core Channel RPC Server
Abort Channel RPC Server
Interrupt Channel RPC Client

Figure B.4 Network Instrument Channels

Figure B.4 from the VXI-11 Specification illustrates the Network Instrument Client (user) and the Network Instrument Server (8065) channel connections. The Network Instrument Server (8065) provides two RPC services (Core Channel RPC Server and Abort Channel RPC Server). Note that the Network Instrument Client also has an RPC Server (Interrupt Channel RPC Server). The Core Channel is required for normal operation. The Abort and Interrupt Channels are optional.

The concept of RPC communication is that the originator of the message is considered the RPC Client, with the receiver of the message being defined as the RPC Server. Since the SRQ notification must originate from the Network Instrument Server (8065), this defines the Network Instrument Server (8065) as being the RPC Client when sending the SRQ notification message.

REVERSE CHANNEL METHODS

The Network Instrument Client can use three different methods to receive SRQ notification messages. The first method assumes that the client is using a support library such as a TCP/IP (VXI-11) capable VISA. The client would register a function to be executed upon receipt of an SRQ notification message and the support library would call the function when the SRQ notification message is received.

The next two methods are more commonly used by the RPC client. First, a client might register an RPC service, to be invoked upon receipt of an RPC SRQ notification message from the Interrupt Channel RPC Client. Next, the client might establish a TCP listen socket. The Interrupt Channel RPC Client would then connect to this socket and send the SRQ notification message through it.

Registering an RPC service is relatively simple, with numerous examples existing. However, the developer must remember that the client computer must have an RPC PortMapper service installed and enabled. This requirement may be beyond the abilities of the customer environment due to network and/or system installation requirements. Consequently, it is more suited for a controlled application with a dedicated controller system where the client application runs on a system under control by the developer of the application client. In other words, it is best run on an in-house system.

Since the Interrupt Channel RPC Client does not require an RPC PortMapper service to establish a connection to the Interrupt Channel RPC Server, it might be easier to create a TCP socket level service to handling incoming SRQ notification messages. The VXI-11 create_intr_chan function specifies the RPC service information (IP address, port, program number, program version, and TCP/UDP protocol) rather than requiring the RPC client to query an RPC Port-Mapper service for the information. The RPC client then attempts to establish a socket connection using the RPC service information specified in the VXI-11 create_intr_chan function.

ENABLING AND IDENTIFYING GPIB INSTRUMENTS

The application client uses the VXI-11 device_enable_srq function for each device that is to generate a SRQ. As part of the VXI-11 device_enable_srq function, the client specifies a unique identifier for each GPIB instrument being enabled. A handle might consist of the ASCII string “device 14”. Then when the VXI-11 device_intr_srq message is sent to the Interrupt Channel RPC Server, the handle is included in the message as the sole VXI-11 data field. The Interrupt Channel RPC Server is then able to easily determine which instrument is requesting service (SRQ generator) by examining the handle portion of the VXI-11 device_intr_srq message.

If or when a SRQ event occurs for a GPIB instrument that has been specified through a device_enable_srq VXI-11 function, a VXI-11 device_intr_srq RPC message is then sent through the channel established by the earlier VXI-11 create_intr_chan function. This message will be an RPC message with the attendant RPC header information and a VXI-11 payload section. The RPC header information can be safely ignored and the VXI-11 payload section will contain only the VXI-11 device_intr_srq handle field.

INTERRUPTING INSTRUMENT IDENTIFICATION

The VXI-11 device_intr_srq handle field is defined as being of an opaque data type (Note 1). This means that the first four bytes are the length field, in network order. The length field should be put into host order through the usage of a network-to-host conversion macro. The length field specifies the actual length of the following string that contains the handle that was defined by the VXI-11 device_enable_srq function. The handle data should be used to determine which GPIB instrument is generating the SRQ message.

PROGRAM OUTLINE

The following steps outline a simple test program created to test the RPC methods using the RPC service in the Client Application computer.

1. Create a Reverse Channel.
2. Create a link to an instrument.
3. Issue a readstb to ensure no pending SRQ status.
4. Write a string to clear the device of error conditions.
5. Write a string to enable SRQ on error. With an IEEE-488.2 device the error can be created by an unrecognized command.
6. Enable SRQ for the instrument using link ID from step-2.
7. Cause an error condition (which generates the SRQ).
8. Wait for a background signal saying reverse message received.
9. The background sets a flag for the foreground thread that identifies the instrument that generated the message.
10. Perform a readstb on the instrument and confirm value of 0x60.

RECOMMENDATIONS

1. While socket coding is relatively simple, it is not an intuitively easy task. It is therefore recommended that such socket coding as a socket level Interrupt Channel RPC Server be done by an engineer experienced in socket coding.

2. If creating a socket level Interrupt Channel RPC Server, it is important to handle network disconnects properly. If a network error happens, it is possible that the socket connection will break.

3. If the Interrupt Channel connection (socket level or RPC level) should become disconnected, it must be regenerated through the usage of the VXI-11 create_intr_chan and device_enable_srq functions.

4. If the Core Channel used to create an Interrupt Channel is disconnected, the Network Instrument Server will disconnect the Interrupt Channel RPC Client. Thus if it is wished to keep the Interrupt Channel active, the initial Core Channel itself must be kept active.

5. It is possible (but not advisable) to create multiple Interrupt Channels for multiple Network Instrument Client support. However, it is important to note that if multiple Interrupt Channels are attached to the same GPIB instrument, only the first Interrupt Channel RPC Server will be notified.

6. The user must be consistent with his use of instrument links. The same link used to originally link to the GPIB device in the Core Channel should be used with all other functions sent to the same GPIB device for the related Interrupt Channel.

7. The Interrupt Channel is a one-way channel. The background should not attempt to send the instrument any messages or perform a readstb on the instrument. That should be left to the foreground operation.

8. If the 8065 fails to connect to an open SRQ Socket it is probably due to a TCP stack error. When a socket connect request is passed to the TCP/IP stack layer, it nearly always executes a successful connection. This is of course assuming that the target box is there, listening, and ready for a connection. However sometimes odd things happen. A failure to successfully connect is in this area of “normally works”, but doesn’t always.

   Normally error handling would retry a connection and/or handle the error in some other way. The problem in this specific case is that the client may have given an incorrect IP, port address, or may not have the port in a listen state. So error handling inside the 8065 becomes problematic. The 8065 does not assume that the connection error was due to a TCP layer error and should be retried. Instead the 8065 handles the error by aborting the connection attempt.

   Note that if and only if the create_intr_chan function believes it has successfully established a specified IP/port, will it return a 0 (no) error status. Thus if it returns a 0 error status and your application believes that no connection is established, the application must issue a destroy_intr_chan to close out the channel prior to creating a new Reverse Channel.

   If a Reverse Channel connection is not successful (as determined by the application) within a pre-defined amount, the client should re-issue the create channel VXI-11 command. Thus the error handling is done by the client (application). 8065 commands are actually functions being called by the RPC application. If a function fails to properly perform, it is up to the application to do error checking and error handling. Remember that if the create_intr_chan function returns a 0 status, the 8065 believes that a connection is established (no error on connection attempt).

9. The 8065's Auto Polling takes place in the background and can alter the address state of the enabled devices. The application client should disable the device from generating an SRQ when maintaining the device's Listening or Talking address state is critical.

SUMMARY

This Application Note outlines several ways to use a VXI-11 Reverse Interrupt Channel to notify the Client Application about a GPIB device that has generated a SRQ. This note also includes the outline of a test program the user can create to verify that the Reverse Channel is operating correctly and some recommendations about writing RPC programs. While VXI-11 systems are harder to program, the notification process identifies the device generating the SRQ which can be a considerable time savings in a large system with multiple devices.

Note 1 - Opaque Data Type - a variable length opaque data defined as a leading Big-Endian 4-byte length field and a series of bytes (followed by NULL padding to create a length, multiple of 4).

Example:
0x00, 0x00, 0x00, 0x05, ‘*’, ‘I’, ‘D’, ‘N’, ‘?’, 0x00, 0x00, 0x00

The last 3 bytes are to expand the data to a length which is a multiple of 4. The first 4 bytes is the length count which defines how many bytes are valid data.

Note that the data length is 5, but the total data field length is 8.
- 4 bytes of length
- 5 bytes of data
- 3 bytes of padding