Serial Polling and SRQ Servicing with NI-488.2 Software and LabVIEW

Vanessa Trujillo

Introduction

One function of a GPIB Controller is to detect and respond to service requests from devices on the bus. The Service Request Line (SRQ) on the GPIB is designed to signal the Controller when a service request is pending. The Controller must then determine which device asserted the SRQ line and respond accordingly. The most common method for SRQ detection and servicing is the serial poll. This application note describes how the NI-488.2 software detects and responds to service requests from IEEE 488 devices. In addition, this application note also includes code examples that demonstrate how to implement the NI-488.2 software routines and functions in a LabVIEW program.

The Need for Serial Polling – A Classroom Analogy

The communication between the GPIB controller and the instruments could be compared to the one between an instructor and the students in a classroom. In the classroom, an instructor is in charge of the class and controls activity. The GPIB works in a similar fashion, where the Controller determines when tasks are performed. In the classroom, a student must have permission to speak out loud, and on the GPIB no devices can communicate unless they are addressed to talk on the bus. But, how can a device let the controller know that it has something to communicate if it cannot start talking to the Controller without the Controller telling it to do so?

The question can be interpreted in the classroom analogy as, how can a student let the instructor know that he or she has something to say, considering that the student cannot talk to the instructor unless the instructor says so? A good method is for the student to raise his or her hand. Similarly, the instrument can assert a hardware line called SRQ. This is a completely separate line in the GPIB bus dedicated to letting the Controller know that a device needs attention.

Why should a device ever need to request service? In some cases, a Controller can request data from a device (writing to the device) and read the data back as soon as the write operation is completed. But this is not possible in every case. Sometimes a Controller might try to read data back from an instrument before data is available (an instrument could take a long time to generate the data), and get a time-out error. This situation is similar to a professor asking the student for the answer to a problem before the student is able to finish the problem. In this case, the professor will prefer to wait for the student to raise his or her hand whenever he or she has completed the problem. Once the student raises his or her hand, the professor will ask the student for the answer. However, in the GPIB case, because all of the devices share the SRQ line, the Controller does not know *which* device requested service. Serial polling is needed so the Controller can find out which device requested attention.

341560A-01 *August 1998*

The Theory of Serial Polling

Serial polling is a method of obtaining specific information from GPIB devices when they request service. When you conduct a serial poll, the Controller queries each device, searching for the one that asserted the SRQ line. Each device responds to the poll by returning the value of the Status Byte contained in its Status Byte Register (see Figure 1). Device-dependent conditions, such as the presence of available data or an error condition, determine this value. ANSI/IEEE Standard 488.1-1987 specifies that one bit in the Status Byte, the RQS bit (bit 6), be TRUE if the device requested service. The other bits in the Status Byte are left to the instrument manufacturer to define. IEEE 488.1-compatible instruments may have bits that determine if an instrument error has occurred or if the device is conducting a self-test. Because these bit definitions are not consistent among instrument vendors, the method for determining the cause of a service request varies with each device.

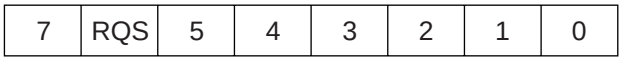

Figure 1. Status Byte Register

ANSI/IEEE Standard 488.2-1987 solves this problem by defining certain service request conditions so that one model describes the Status Byte for all 488.2-compliant devices. The IEEE 488.2 standard builds on and extends the IEEE 488.1 Status Byte discussed in the last section (see Figure 2). IEEE 488.2 defines the RQS bit like the IEEE 488.1 standard. IEEE 488.2 adds the Message Available (MAV) bit and the Event Status Bit (ESB). The MAV bit is set if the device has been previously queried for data and has a pending data message to send. The ESB bit indicates that one of the standard events defined in the Standard Event Status Register has occurred. By setting the corresponding bits in the Standard Event Status Enable Register, you define which standard events will set the ESB. These events include Power-On, User Request, Command Error, Execution Error, Device Dependent Error, Query Error, Request Control, and Operation Completed. By setting the corresponding bits in the Service Request Enable Register, you can configure an instrument to assert the SRQ line when ESB or MAV are set, or when a manufacturer-defined condition occurs.

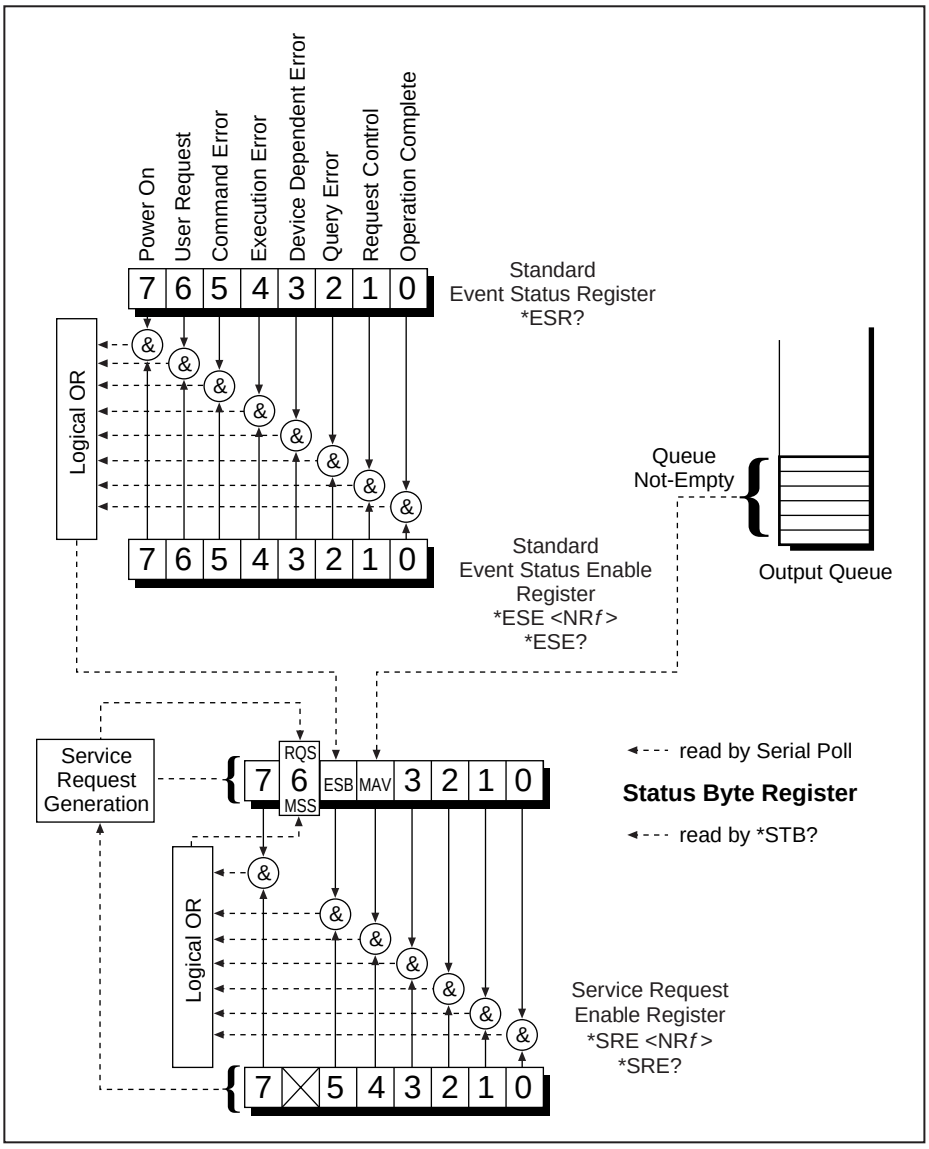

Figure 2. IEEE 488.2 Status Reporting Model

LabVIEW Functions for Serial Polling

The following functions are included in the LabVIEW GPIB 488 subpalette and are essential for performing serial polling in LabVIEW:

GPIB Serial Poll (`ibrsp` in NI-488) performs a serial poll and returns the value of the Status Byte from a single device. The program must manually check the RQS bit in the Status Byte value to determine whether that device requested service.

GPIB Wait (`ibwait` in NI-488) waits for the state indicated by a specified Status Word mask in a particular GPIB bus. The Status Word is a global variable (16 bit) that contains, among other GPIB information, the state of the SRQ line (bit 12, SRQI), and whether a device has requested service (bit 11, RQS). If the vector that is passed in equals 0, it does not wait for any specified case and will update `ibsta` or the Status Word. However, if the mask is set to check

for bit 12 (SRQI), the function will wait until that particular line gets asserted for a particular GPIB board, the address of which is specified in GPIB Wait.

Wait for GPIB RQS (`ibwait` in NI-488, with mask for bit 11) is similar to using the GPIB Wait function with a mask for bit 11, but using the GPIB address of the particular device address instead of the address of the GPIB board (when waiting for bit 12). This function waits for a particular device to request service, and not for a board (to which several devices could be connected) to receive a service request. This function will wait until either the RQS bit is asserted or the time-out is exceeded.

The following examples show how to service SRQs and serial poll the devices using LabVIEW GPIB functions.

In LabVIEW Example 1 (Figure 3), a command is written to an NI Instrument Simulator. The command includes instructions for the instrument to generate a Service Request (SRQ) if an Operation Completed event is registered in the Event Status Register (ESR). The first two commands write these instructions to the Service Request Enable Register and the Event Status Enable Register (IEEE-488.2 specification). The third command asks the instrument simulator to generate a sine wave; the last command generates an Operation Completed event. The following section of the code waits for the SRQ to be generated by checking bit 12 of the Status Word. Once it has been generated, a serial poll is performed on the device simulator to read the Status Byte and learn the reason it generated an SRQ. If bit 5 is set (from the Event Status Register having the Operation Complete bit set) and bit 6 is set (RQS, the device has requested service), then the Controller goes ahead and reads the data back from the device simulator. Notice that the ESR register must be cleared. This prevents the instrument from generating unwanted service requests.

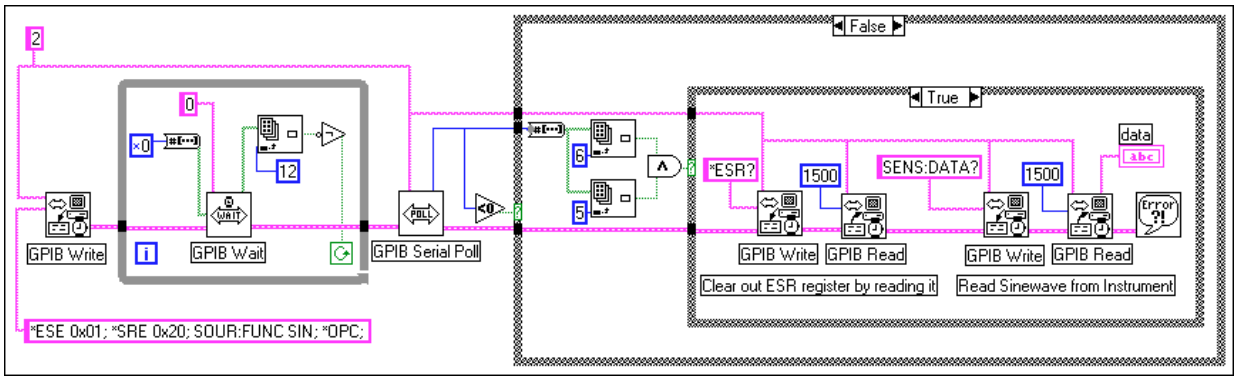

Figure 3. LabVIEW Example 1

LabVIEW Example 2 (Figure 4) shows another way of waiting for the device to request service. However, in this last example, the processor stops and waits without performing any other operations. In the first example, other operations could have been executed in parallel while waiting for the device to request service.

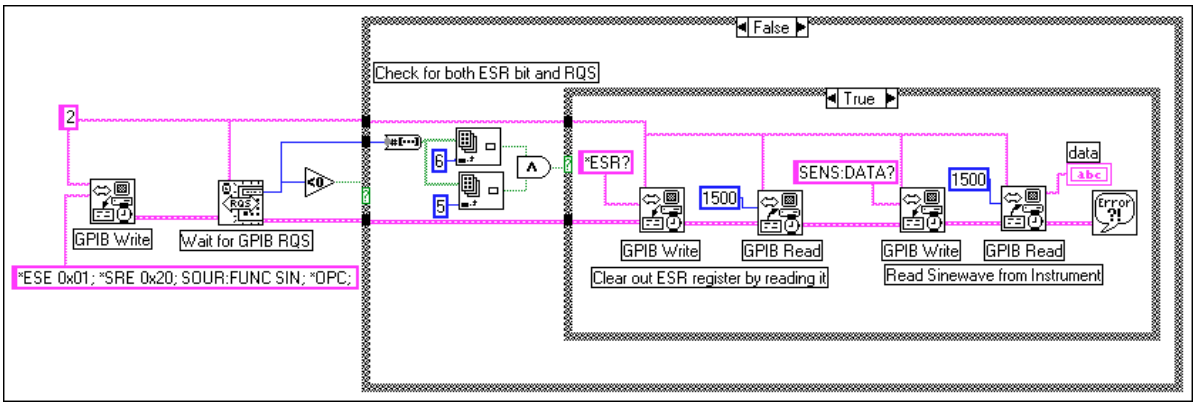

Figure 4. LabVIEW Example 2

Serial Polling LabVIEW GPIB 488.2 Functions

GPIB 488.2 functions add new features for serial polling so that you can poll several devices with one GPIB command. LabVIEW has included these functions in the GPIB 488.2 functions subpalette – `AllSpoll` and `FindRQS`.

`AllSpoll` can serial poll multiple devices with a single routine call. `AllSpoll` places the status byte from each instrument polled into a predefined array. You must manually check the RQS bit in the Status Byte of each device to determine whether that device requested service.

`FindRQS` can serial poll several devices. If one of the devices requests service by asserting SRQ, the routine returns the index and Status Byte value of the first device requesting service.

If you know that only one instrument asserted the SRQ and would like to know which one, and its serial poll response byte, then use the `FindRQS` function. If you suspect that more than one instrument may have asserted the SRQ, then use `AllSpoll` to receive the serial poll response bytes from all the devices. If the status byte has bit 6 set, it means that the device has requested service. In other words, the value of the status byte is at least hex 40 when a device requests service.

LabVIEW Example 3 (Figure 5) illustrates how to perform serial polling using LabVIEW NI-488.2 functions. Notice that this GPIB 488.2 example is very similar to the GPIB examples previously shown. Most LabVIEW GPIB functions have been replaced with their equivalent LabVIEW GPIB 488.2 functions, such as “Send” and “Receive” instead of “GPIB Write” and “GPIB Read,” “ReadStatus” instead of “GPIB Serial Poll,” and “Test SRQ” instead of using “GPIB Wait” and setting the mask to 0. In addition, this example shows a better way of controlling the possible errors in the program by using a shift register to pass the error cluster in between cycles of the while loop.

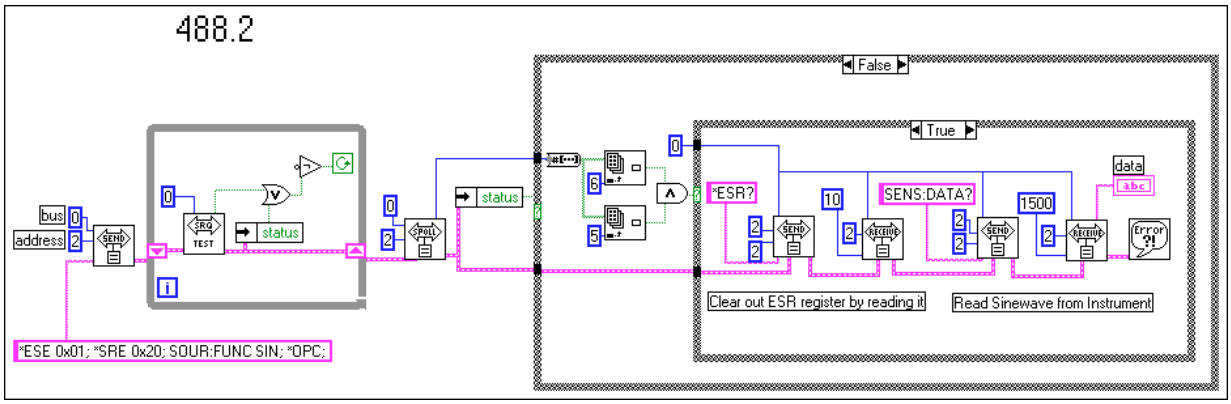

Figure 5. LabVIEW Example 3

Summary

This application note describes how the GPIB Controller uses the NI-488 software to detect and respond to service requests from IEEE 488 devices on the bus. It also includes LabVIEW code examples that demonstrate how you can use the LabVIEW GPIB functions to serial poll devices.

341560A-01 Aug98