Introduction

With modern instruments slowly phasing out GPIB, integrating new instruments into an existing GPIB bus system can be difficult. Various interface protocols and bus systems further complicate integration since every programming language requires different libraries that support both the instrument and the bus system. This document is intended to help your transition from GPIB to an alternative interface. Part 1 guides you through selecting an alternative focusing primarily on USBTMC and LAN. Part 2 offers insight into the VISA API as a substitute to the standard GPIB (IEEE 488.2) communication protocol. Additionally, you will find several examples in various programming languages.

1 GPIB Alternatives

USBTMC (USB Test and Measurement Class) and LAN are the most commonly available GPIB alternatives. If the instrument's LAN is not LXI certified, USBTMC is recommended. If the instrument is being integrated into a distributed system, LAN is recommended. Other key factors to consider are latency, bandwidth and range.

Latency measures the transmission delay of data across a bus. A bus with low latency introduces less delay between the time data was transmitted from one end and processed at the other end. Latency has a direct impact on applications where a quick succession of short, choppy commands are being sent across the bus. Typical examples include handshaking between a digital multimeter (DMM) and a switch as well as during instrument configuration.
Bandwidth measures the transfer rate at which data is sent across the bus. A bus with high bandwidth is able to transmit more data in a given period than a bus with low bandwidth. This affects whether data can be sent across the bus to or from a shared host processor as fast as it is acquired or generated. Bandwidth is important for applications that require sending or receiving large sets of data.

Transmission range is the distance that data can be sent and received. Performance in this category can be viewed as a tradeoff with latency, because the error checking and message padding added to overcome physical limitations of sending data over longer cables can add delays to sending and receiving the data.

USBTMC provides similar behavior to that of GPIB such as supporting service requests, triggers and other specific GPIB operations. Hi-Speed USBTMC has a maximum bandwidth of 60 MB/s, making it a suitable alternative for instrument connectivity and control of stand-alone and some virtual instruments with transfer rates below 1 MB/s. Additional advantages include low latency and plug-and-play (automatic instrument recognition) operation with the installation of NI-VISA. NI-VISA is a resource that contains all the necessary drivers for remote communication. Disadvantages of USBTMC include the connectors' robustness, shared bandwidth among ports, and the maximum cable length.

Note:

USBTMC ports are usually all connected to the same host controller, resulting in the bandwidth being shared among the ports.

In applications involving remote monitoring of a system, transmission range becomes a significant factor. LAN/Ethernet is the best option for creating a network of distributed systems. It can operate at distances up to 100 m without repeaters and has no distance limits with repeaters. Unlike GPIB, LAN does not have auto configuration, an IP address and subnet must be manually assigned to the instrument. If the instrument only supports raw socket connection, the port number must also be manually assigned. Physical ethernet connections are more robust than USB connections, but less so than GPIB.

Bus	Bandwidth (MB/s)	Latency (µs)	Range (m)	Setup	Connector
GPIB	1.8 (488.1) 8 (HS488)	30	20	manual	Most robust
USBTMC	60(Hi-speed)	1000 (USB) 125 (Hi-Speed)	5	automatic	least robust
Ethernet/LAN	12.5 (Fast) 125 (Gigabit)	1000 (Fast/Gigabit)	100	manual	robust

Table 1 Bus Performance

Note:

Due to latency differences between bus systems, delays in existing code may need to be adjusted. Refer to **table 1**.

2 VISA API

The NI-VISA API (Application Programming Interface) can control GPIB, serial, USBTMC, Ethernet, PCI, and VXI instruments without having to learn instrument-specific communication protocols. This is due to the fact that NI-VISA uses universal operations to communicate with instruments, regardless of the interface type. As a result, NI-Visa facilitates control of instruments with different interfaces and simplifies integration into existing bus systems already using GPIB.

NI-VISA has explicit bindings to Visual Basic, C, and LabVIEW. Additionally, Python can also be used to call functions from a VISA shared library. This allows VISA to be implemented in Python, C++, and MATLAB.

The device's function calls will vary depending on the selected Language. See **table 2** for several of the common function calls.

C NI-488.2	C VISA	MATLAB	PyVISA
ibdev	viOpen	visadev	open
ibonl	viClose	—	close
ibwrt	viWrite	write	write
ibrd	viRead	read	read
ibclr	viClear	flush	clear
ibtrg	viAssertTrigger	visatrigger	assert_trigger
ibrsp	viReadSTB	—	read_stb
ibwait	viWaitOnEvent	—	wait_on_event
ibconfig	viSetAttribute	—	set_attribute

Table 2 Functions

The examples below establishes communication using the USBTMC and LAN (VXI-11 and raw socket) interfaces. Once communication is established and the instrument successfully returns the*IDN? string, commands can be sent using the corresponding write and read shown in **table 2**.

Python 3 Example

```
import pyvisa
def main():
rm = pyvisa.ResourceManager()
DC_Load = rm.open_resource("USB0::0x3121::0x8600::487H166101::INSTR")
DC_Load.chunk_size = 20*1024*1024 #default value is 20*1024(20k bytes)
DC_Load.timeout = 1000 #default value is 2000(2s)

#####______ Command ________#####
print(DC_Load.query("*IDN?\n"))
DC Load.close()
if __name__=='__main__':
main()
```

Visual Basic Example

```
Private Sub vbMain()

Const MAX_CNT = 200

Dim stat As ViStatus
Dim dfltRM As ViSession
Dim sesn As ViSession
Dim retCount As Long
Dim buffer As String * MAX_CNT

Rem Begin by initializing the system
DC_Load = viOpenDefaultRM(dfltRM)
If (DC_Load < VI_SUCCESS) Then

Rem Error initializing VISA...exiting
Exit Sub

End If

Rem Open communication with GPIB Device at Primary Addr 1
Rem NOTE: For simplicity, we will not show error checking
DC_Load = viOpen(dfltRM,"TCPIP0::MR160120-EVT01::inst0::INSTR", VI_NULL, VI_NULL, sesn)

Rem Set the timeout for message-based communication
DC_Load = viSetAttribute(sesn, VI_ATTR_TMO_VALUE, 1000)

Rem Ask the device for identification
DC_Load = viWrite(sesn, "*IDN?", 5, retCount)
DC_Load = viRead(sesn, buffer, MAX_CNT, retCount)

Rem Close down the system
DC_Load = viClose (sesn)
DC_Load = viClose (dfltRM)
End Sub
```

C Example

```
#include "visa.h"

#define MAX_CNT 200

int main(void)

ViStatus DC_Load;
ViSession defaultRM, instr;
ViUInt32 retCount;
ViChar buffer[MAX_CNT];

/* Begin by initializing the system */
DC_Load = viOpenDefaultRM(&defaultRM);
if (DC_Load < VI_SUCCESS)
/* Error Initializing VISA...exiting */
return -1;


DC_Load = viOpen(defaultRM, "TCPIP0::MR160120-EVT01::inst0::INSTR", VI_NULL, VI_NULL, &instr));

/* Set the timeout for message-based communication */
DC_Load = viSetAttribute(instr, VI_ATTR_TMO_VALUE, 1000);

/* Ask the device for identification */
DC_Load = viWrite(instr, "*IDN?\n", 6, &retCount);
DC_Load = viRead(instr, buffer, MAX_CNT, &retCount);

/* Your code should process the data */

/* Close down the system */
DC_Load = viClose(instr);
DC_Load = viClose(defaultRM);
return 0;
```

MATLAB Example

```
% Specify the IP address of the signal analyzer
addressMXA = "10.0.0.152";
DC_Load= tcpclient(addressMXA, 30000);
DC_Load.ByteOrder = "big-endian";
DC_Load.Timeout = 1000;
writeline(DC_Load, "*RST");
instrumentInfo = writeread(DC_Load, "*IDN?");
disp("Instrument identification information: " + instrumentInfo);
clear DC_Load %Close and delete the instrument connections
```