An Overview of
Data Communication in LabVIEW

Elijah Kerry – LabVIEW Product Manager
Certified LabVIEW Architect (CLA)
Data Communication Options in LabVIEW

1. TCP and UDP
2. Network Streams
3. Shared Variables
4. DMAs
5. Web Services
6. Peer-to-Peer Streaming
7. Queues
8. Dynamic Events
9. Functional Global Variables
10. RT FIFOs
11. Datasocket
12. Local Variables
13. Programmatic Front Panel Interface
14. Target-scoped FIFOs
15. Notifier
16. Simple TCP/IP Messaging (STM)
17. AMC
18. HTTP
19. FTP
20. Global variables

... just to name a few ...
Communication is Important

Windows  Real-Time  FPGA
Agenda

- Introduction of Data Communication
- Define Communication Types
- Identify Scope of Communication
  - Inter-process
  - Inter-target
- Next Steps

ni.com/largeapps
Demonstration

The pitfalls of local variables
Common Pitfalls of Data Communication

**Race conditions** - two requests made to the same shared resource

**Deadlock** - two or more depended processes are waiting for each other to release the same resource

**Data loss** - gaps or discontinuities when transferring data

**Performance degradation** - poor processing speed due to dependencies on shared resources

**Buffer overflows** - writing to a buffer faster than it is read from the buffer

**Stale data** - reading the same data point more than once
The Dining Philosophers
Communication Types

• Message/Command  “Get me a soda!”
• Update/Monitor  “The current time is…”
• Stream/Buffer  “…the day the music died…”
• Variable/Tag  “Set Point = 72F”
Message/Command

- Commander (Host) and Worker (Target) Systems
- Must be lossless* (can be buffered)
- Minimal latency
- Typically processed one at a time
- Reads are destructive
- Example: stop button, alarm, error

*some commands may need to pre-empt other commands based on priority
Update/Monitor

- Periodic transfer of latest value
- Often used for HMIs or GUls
- N Targets: 1 Host
- Can be lossy
- Non-buffered

Example: monitoring current engine temperature
Stream/Buffer

• Continuous transfer, but not deterministic
• High throughput
• No data loss, buffered
• 1 Target: 1 Host; Unidirectional

Example: High speed acquisition on target, sent to host PC for data logging
Variable/Tag

- Set Points and PID Constants
- Initial configuration data
- Can be updated during run-time
- Only latest value is of interest
- 1 Host: N Targets

Example: reading/writing the set-point of a thermostat, .ini configuration files
# Choosing Transfer Types

<table>
<thead>
<tr>
<th></th>
<th>Message</th>
<th>Update</th>
<th>Stream</th>
<th>Variable (Tag)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Examples</strong></td>
<td>• Exec Action</td>
<td>• Heartbeat</td>
<td>• Waveform</td>
<td>• Setpoint</td>
</tr>
<tr>
<td></td>
<td>• Error</td>
<td>• Movie</td>
<td>• Image</td>
<td></td>
</tr>
<tr>
<td><strong>Fundamental Features</strong></td>
<td>• Buffering</td>
<td>• Nonhistorical</td>
<td>• Buffering</td>
<td>• Nonhistorical</td>
</tr>
<tr>
<td></td>
<td>• Blocking</td>
<td></td>
<td>• Blocking</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• (Timeout)</td>
<td></td>
<td>• (Timeout)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Single-Read</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Optional Features</strong></td>
<td>• Ack</td>
<td>• Broadcast</td>
<td>• Multi-layer</td>
<td>• Dynamic Lookup</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Buffering</td>
<td>• Group Mgmt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Latching</td>
<td>• Latching</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>• Low-Latency</td>
<td>• Low-Latency</td>
<td>• High-</td>
<td>• Low-Latency</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Throughput</td>
<td>• High-Count</td>
</tr>
<tr>
<td><strong>Configuration</strong></td>
<td>• N Targets: 1 Host</td>
<td>• N Targets: 1 Host</td>
<td>• 1 Target: 1 Host</td>
<td>• N Targets: 1 Host</td>
</tr>
</tbody>
</table>
Scope of Communication

**Inter-process**: the exchange of data takes place within a single application context

**Inter-target**: communication between multiple physical targets, often over a network layer
Defining Inter-process Communication

- Communication on same PC or Target
- Communicate between parallel processes or loops
- Offload data logging or processing to another CPU/Core/Thread within same VI/executable
- Loops can vary in processing priority
- Used to communicate synchronously and asynchronously
Inter-process Communication Options

**Shared Variables**
Update GUI loop with latest value

**Queues**
Stream continuous data between loops on a non-deterministic target

**Dynamic Events**
Register Dynamic Events to execute sections of code

**Functional Global Variables (FGV)**
Use a non-reentrant subVI to protect critical data

**RT FIFOs**
Stream continuous data between time critical loops on a single RT target
Basic Actions

- Set the value of the shift register
Basic Actions

- Get the value currently stored in the shift register
Action Engine

- Perform an operation upon stored value and save result
- You can also output the new value
How It Works

1. Functional Global Variable is a **Non-Reentrant** SubVI
2. Actions can be performed upon data
3. Enumerator selects action
4. Stores result in uninitialized shift register
5. Loop only executes once
Demonstration
Introduction to Functional Global Variables
Benefits: Comparison

Functional Global Variables
- Prevent race conditions
- No copies of data
- Can behave like action engines
- Can handle error wires
- Take time to make

Global and Local Variables
- Can cause race conditions
- Create copies of data in memory
- Cannot perform actions on data
- Cannot handle error wires
- Drag and drop
Understanding Dataflow in LabVIEW

Clump 0

Clump 1

Clump 2

Clump 0
Doing Everything in One Loop Can Cause Problems

- One cycle takes at least 330 ms
- If the acquisition is reading from a buffer, it may fill up
- User interface can only be updated every 330 ms
While Loop

- **Acquire**: 10ms
- **Analyze**: 50ms
- **Present**: 20ms
- **Log**: 250ms

Doing Everything in One Loop Can Cause Problems

- One cycle still takes at least 310 ms
- If the acquisition is reading from a buffer, it may fill up
- User interface can only be updated every 310 ms
**Inter-Process Communication:** ensures tasks run asynchronously and efficiently

**How?**
- Loops are running independently
- User interface can be updated every 20 ms
- Acquisition runs every 10 ms, helping to not overflow the buffer
- All while loops run entirely parallel of each other
Producer Consumer

Best Practices
1. One consumer per queue
2. Keep at least one reference to a named queue available at any time
3. Consumers can be their own producers
4. Do not use variables

Considerations
1. How do you stop all loops?
2. What data should the queue send?
LabVIEW FIFOs

- Queues
- RT FIFOs
- Network Streams
- DMAs
- User Events

In general, FIFOs are good if you need lossless communication that preserves historical information.
Queues

Adding Elements to the Queue

Select the data type the queue will hold

Reference to existing queue in memory

Dequeuing Elements

Dequeue will wait for data or time-out (defaults to -1)
Demonstration
Introduction to LabVIEW Queues
The Anatomy of Dynamic Events

- VI Gets Run on Event
- Defines Data Type
- Data Sent
- Dynamic Events Terminal
- Multiple Loops Can Register for Same Event
Using User Events

LabVIEW API for Managing User Events

Register User Events with Listeners
## Choosing Transfer Types for Inter-process

<table>
<thead>
<tr>
<th></th>
<th>Message</th>
<th>Update</th>
<th>Stream</th>
<th>Variable (Tag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>• Queue&lt;br&gt;• Shared Variable (Blocking, Buffered)</td>
<td>• SE Queue&lt;br&gt;• Notifier&lt;br&gt;• Shared Variable (Blocking)</td>
<td>• Queue&lt;br&gt;• Shared Variable (Blocking, Buffered)</td>
<td>• Local/Global Variable&lt;br&gt;• SE Queue&lt;br&gt;• FGV&lt;br&gt;• Shared Variable&lt;br&gt;• DVR</td>
</tr>
<tr>
<td>RT</td>
<td>• Same as Windows&lt;br&gt;• RT FIFO</td>
<td>• Same as Windows&lt;br&gt;• SE RT FIFO</td>
<td>• Same as Windows&lt;br&gt;• RT FIFO</td>
<td>• Same as Windows</td>
</tr>
<tr>
<td>FPGA</td>
<td>• FIFO (2009)</td>
<td>• SE FIFO (2009)</td>
<td>• FIFO</td>
<td>• Local/Global Variable&lt;br&gt;• FGV</td>
</tr>
</tbody>
</table>
**RT FIFOs vs. Queues**

- Queues can handle string, variant, and other variable size data types, while RT FIFOs can not
- RT FIFOs are pre-determined in size, queues can grow as elements are added to them
- Queues use blocking calls when reading/writing to a shared resource, RT FIFOs do not
- RT FIFOs do not handle errors, but can produce and propagate them

**Key Takeaway:**
RT FIFOs are more deterministic for the above reasons
What is Determinism?

**Determinism:** An application (or critical piece of an application) that runs on a hard real-time operating system is referred to as deterministic if its timing can be guaranteed within a certain margin of error.
LabVIEW Real-Time Hardware Targets

- CompactRIO
- PXI
- Desktop or Industrial PC
- Vision Systems
- Single-Board RIO
RT FIFOs

Write Data to the RT FIFO

- Main FIFO
- RT FIFO Create
- RT FIFO Write
- Select the data type the RT FIFO will hold
- Reference to existing RT FIFO in memory

Read Data from the RT FIFO

- Main FIFO
- RT FIFO Create
- RT FIFO Read
- element out
- Read/Write wait for data or time-out (defaults to 0)
- Write can overwrite data on a timeout condition

Read/Write wait for data or time-out (defaults to 0)
Write can overwrite data on a timeout condition
Demonstration

Inter-process Communication Using RT FIFOs
Defining Inter-target Communication

- PC, RT, FPGA, Mobile Device
- Offload data logging and data processing to another target
- Multi-target/device application
- Network based
Common Network Transfer Policies

“Latest Value” or “Network Publishing”

• Making the current value of a data item available on the network to one or many clients

• Examples
  – I/O variables publishing to an HMI for monitoring
  – Logging temperature values on a remote PC

• Values persist until over written by a new value

• Lossy – client only cares about the latest value
# Latest Value Communication

<table>
<thead>
<tr>
<th>API</th>
<th>Type</th>
<th>Performance</th>
<th>Ease of Use</th>
<th>Supported Configurations</th>
<th>3rd Party APIs?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared Variable*</td>
<td>LabVIEW Feature</td>
<td>1/2</td>
<td>1/2</td>
<td>1:1, 1:N, N:1</td>
<td>• Measurement Studio</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• CVI</td>
</tr>
</tbody>
</table>

*Network buffering should be disabled*
Using Shared Variables Effectively

Programming Best Practices:
• Initialize shared variables
• Serialize shared variable execution
• Avoid reading stale shared variable data
Common Network Transfer Policies

“Streaming”

• Sending a lossless stream of information
• Examples
  – Offloading waveform data from cRIO to remote PC for intensive processing
  – Sending waveform data over the network for remote storage
• Values don’t persist (reads are destructive)
• Lossless – client must receive all of the data
• High-throughput required (latency not important)
# Streaming Lossless Data

<table>
<thead>
<tr>
<th>API</th>
<th>Type</th>
<th>Performance</th>
<th>Ease of Use</th>
<th>Supported Configurations</th>
<th>3rd Party APIs?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Streams NEW!</td>
<td>LabVIEW Feature</td>
<td><img src="image1" alt="Circle" /></td>
<td><img src="image2" alt="Circle" /></td>
<td>1:1</td>
<td>Not this year</td>
</tr>
</tbody>
</table>

What about the shared variable with buffering enabled?

**NO!**
Pitfalls of Streaming with Variables

- Lack of flow control can result in data loss
- Data may be lost if the TCP/IP connection is dropped
- Data loss does not result in an error, only a warning
Network Streams **NEW!**

**Machine A**
- **timeout (ms)**: -1
- **writer buffer size**: 10
- **writer name**: write_end
- **reader url**: //Machine-B/read_end
- **data type**: 0

**Machine B**
- **timeout (ms)**: -1
- **reader buffer size**: 100
- **reader name**: read_end
- **data type**: 0
Network Streams in Action

Use Streams!
Demonstration
Inter-target Communication Using Network Streams
Common Network Transfer Policies

“Command” or “Message”

- Requesting an action from a worker
- Examples
  - Requesting an autonomous vehicle to move to a given position
  - Telling a process controller to begin its recipe
- Values don’t persist (reads are destructive)
- Lossless – client must receive every command
- Low latency – deliver the command as fast as possible
## Network Command Mechanisms

<table>
<thead>
<tr>
<th>API</th>
<th>Type</th>
<th>Performance</th>
<th>Ease of Use</th>
<th>Supported Configurations</th>
<th>3rd Party APIs?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Streams</td>
<td>LabVIEW Feature</td>
<td><img src="#" alt="Circle" /></td>
<td><img src="#" alt="Circle" /></td>
<td>1:1</td>
<td>No</td>
</tr>
</tbody>
</table>
Network Streams

Writing Elements to the Stream

Create Network Stream Writer Endpoint → Write Multiple Elements to Stream → Destroy Stream Endpoint

Reference to reader URL

Select the data type the queue will hold

Reading Elements from Stream

Create Network Stream Reader Endpoint → Read Multiple Elements from Stream (Read will wait for data or time-out (defaults to -1)) → Destroy Stream Endpoint

 nacional instruments

Network Streams
Network Streams

• Lossless transfer, even in connection loss*
• Can be tuned for high-throughput (streaming) or low-latency (messaging)
• Unidirectional, P2P, LabVIEW only
• Not deterministic

Acquire/Control  

Log Data/Process

Writer Endpoint  

FIFO  

Network Streams Engine  

FIFO  

Reader Endpoint
DMA (Direct Memory Access)

- Use for Host to Target Communication (i.e., RT to FPGA)
- Available for newer FPGAs
- Useful for transferring chunks of data
- High latency
Demonstration
Introduction to Direct Memory Access
Target to Host Transfer – Continuous

Total Samples to Read = ???
Read Size = 4
RT Buffer Size = ~5x Read Size

Data element

FPGA FIFO → DMA Engine → RT Buffer

RT Buffer Size = ~5x Read Size

Mod1/AI0

1D Wfm SGL

Wfm Data

RT Buffer
Continuous Transfer - Buffer Overflow

Total Samples to Read = ???
Read Size = 4
RT Buffer Size = ~5x Read Size

Data element

FPGA FIFO

DMA Engine

RT Data Buffer
LabVIEW Web Services

Application Architecture:

LabVIEW Application \rightarrow LabVIEW Web Service \rightarrow Client

Sending Requests via URL:

http://localhost/Calculator/Sum/3/6

Physical Location of Server
Name of Web Service
Mapping to a VI
Terminal Inputs (Optional)
Web Services in LabVIEW

Web Server

Any Client

LabVIEW

Thin Client

Windows and Real-Time
Custom web clients
No runtime engine needed
Standard http protocol
Firewall friendly

National Instruments
Demonstration

Basic Web Services
NI LabVIEW Web UI Builder Product Information

What Is LabVIEW Web UI Builder?

LabVIEW Web UI Builder gives you the ability to develop lightweight, web-based thin client applications through graphical programming. These applications serve as graphical user interfaces (GUIs) that enable users to remotely monitor and control LabVIEW-based measurement and automation systems through a web browser.

See Applications You Can Create

- View Tank Simulator
- View Plastics Plant Control
- View Pipeline Monitor
- View Windmill Monitor

Product Information

- LabVIEW Web UI Builder Overview
- Frequently Asked Questions
Demonstration
Thin-Client Web Interfaces
Early Access Release Details

• Anyone can evaluate for free
  ▪ Fully functional except for ‘Build and Deploy’
  ▪ License for ‘Build and Deploy’ is $1,499 per user
  ▪ License is sold as one-year software lease

• Not part of Developer Suite or Partner Lease
Inter-Target Communication Options

TCP/IP and UDP
Define low-level communication protocols to optimize throughput and latency

Shared Variables
Access latest value for a network published variable

Network Streams
Point to Point streaming in LabVIEW with high throughput and minimal coding

Web UI Builder
Create a thin client to communicate with a LabVIEW Web Service

DMAs
Direct memory access between different components of a system
"Certification is an absolute must for anyone serious about calling himself a LabVIEW expert... At our organization, we require that every LabVIEW developer be on a professional path to become a Certified LabVIEW Architect."

- President, JKI Software, Inc.
Download Examples and Slides

ni.com/largeapps

- Software Engineering Tools
- Development Practices
- LargeApp Community