Best practices for Inter-Processes & Inter-Target Data Communication using LabVIEW
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-scopeed 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
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
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>• Non-historical</td>
<td>• Buffering</td>
<td>• Non-historical</td>
</tr>
<tr>
<td></td>
<td>• Blocking (Timeout)</td>
<td>• Blocking (Timeout)</td>
<td>• Blocking (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 Buffering</td>
<td>• Dynamic Lookup</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Group Mgmt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Latching</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>• Low-Latency</td>
<td>• Low-Latency</td>
<td>• High-Throughput</td>
<td>• Low-Latency</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></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>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Unidirectional</td>
<td></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
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
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
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

<table>
<thead>
<tr>
<th>While Loop</th>
<th>Acquire</th>
<th>10ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>While Loop</td>
<td>Analyze</td>
<td>50ms</td>
</tr>
<tr>
<td>While Loop</td>
<td>Log</td>
<td>250ms</td>
</tr>
<tr>
<td>While Loop</td>
<td>Present</td>
<td>20ms</td>
</tr>
</tbody>
</table>
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

ni.com
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</td>
<td>• SE Queue</td>
<td>• Queue</td>
<td>• Local/Global Variable</td>
</tr>
<tr>
<td></td>
<td>• Shared Variable (Blocking, Buffered)</td>
<td>• Notifier</td>
<td>• Shared Variable (Blocking, Buffered)</td>
<td>• SE Queue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Shared Variable</td>
<td>• Local/Global Variable</td>
<td>• FGV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Blocking)</td>
<td>• DVR</td>
<td></td>
</tr>
<tr>
<td>RT</td>
<td>• Same as Windows</td>
<td>• Same as Windows</td>
<td>• Same as Windows</td>
<td>• Same as Windows</td>
</tr>
<tr>
<td></td>
<td>• RT FIFO</td>
<td>• RT FIFO</td>
<td>• SE RT FIFO</td>
<td></td>
</tr>
<tr>
<td>FPGA</td>
<td>• FIFO (2009)</td>
<td>• SE FIFO</td>
<td>• FIFO</td>
<td>• Local/Global Variable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2009)</td>
<td></td>
<td>• 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
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

ni.com
RT FIFOs

Write Data to the RT FIFO

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

Read Data from the RT FIFO

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

ni.com
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>3\textsuperscript{rd} Party APIs?</th>
</tr>
</thead>
</table>
| Shared Variable* | LabVIEW Feature       | ![Performance](chart) | ![Ease of Use](chart) | 1:1, 1:N, N:1             | • Measurement Studio  
• CVI                                  |

*Network buffering should be disabled*
Using Shared Variables Effectively

Programming Best Practices:
• Initialise shared variables
• Serialise 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>![ ]</td>
<td>![ ]</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

Machine B

ni.com
Network Streams in Action

Machine 1
Writer Endpoint

Flow Control (2)

Machine 2
Reader Endpoint

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="./circle.png" alt="Circle" /></td>
<td><img src="./circle.png" alt="Circle" /></td>
<td>1:1</td>
<td>No</td>
</tr>
</tbody>
</table>
DMA (Direct Memory Access)

- Use for Host to Target Communication (ie: RT to FPGA)
- Available for newer FPGAs
- Useful for transferring chunks of data
- High latency
Target to Host Transfer – Continuous

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

FPGA FIFO → DMA Engine → RT Buffer
Continuous Transfer - Buffer Overflow

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

FPGA FIFO

Data element

Mod1/AI0

DMA Engine

RT Data Buffer

ni.com
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.
How do I go further?

• Training
  – LabVIEW Core2
  – LabVIEW Connectivity

• Reading
  – Simple Machine Control Reference Application
    – [http://zone.ni.com/devzone/cda/epd/p/id/6250](http://zone.ni.com/devzone/cda/epd/p/id/6250)

• Talk to your
  – Local Engineer