UiO : University of Oslo

FYS3240
PC-based instrumentation and microcontrollers

LabVIEW programming II

Spring 2011 – Lecture #3

Bekkeng 30.1.2011

Control flow vs. dataflow programming

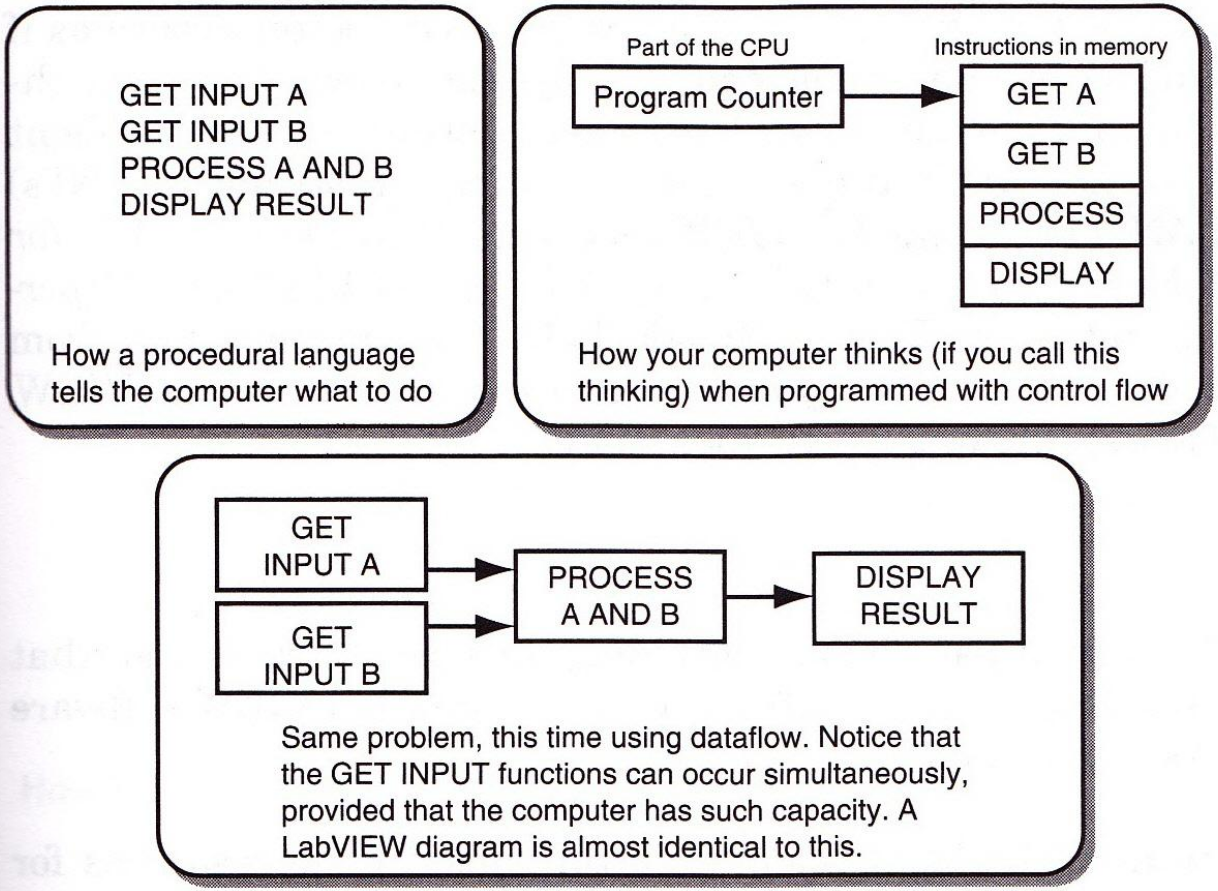
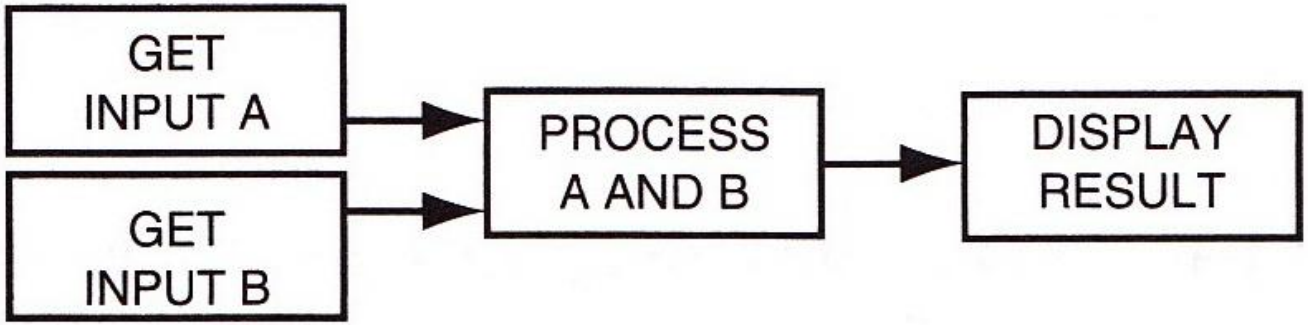


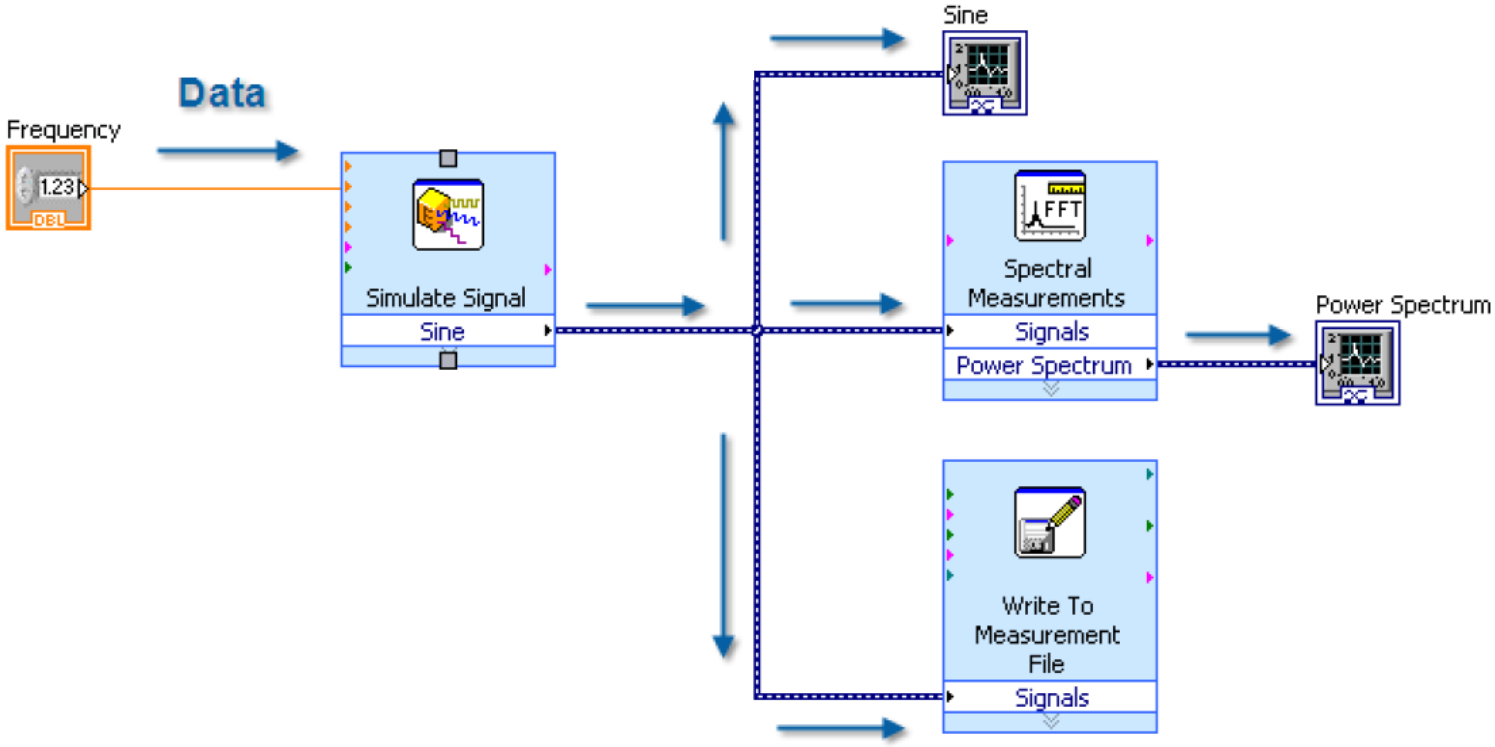
Figure 2.2 Compare and contrast control flow programming in a procedural language with dataflow programming in LabVIEW.

Dataflow Programming Overview

- With a dataflow model, nodes on a block diagram are connected to one another to express the logical execution flow
- When a block diagram node **receives <u>all required inputs</u>, it produces output data and passes that data to the next node** in the dataflow path. The movement of data through the nodes <u>determines the execution order of the functions on the block diagram</u>



Graphical Programming – Dataflow



Dataflow programming with LabVIEW

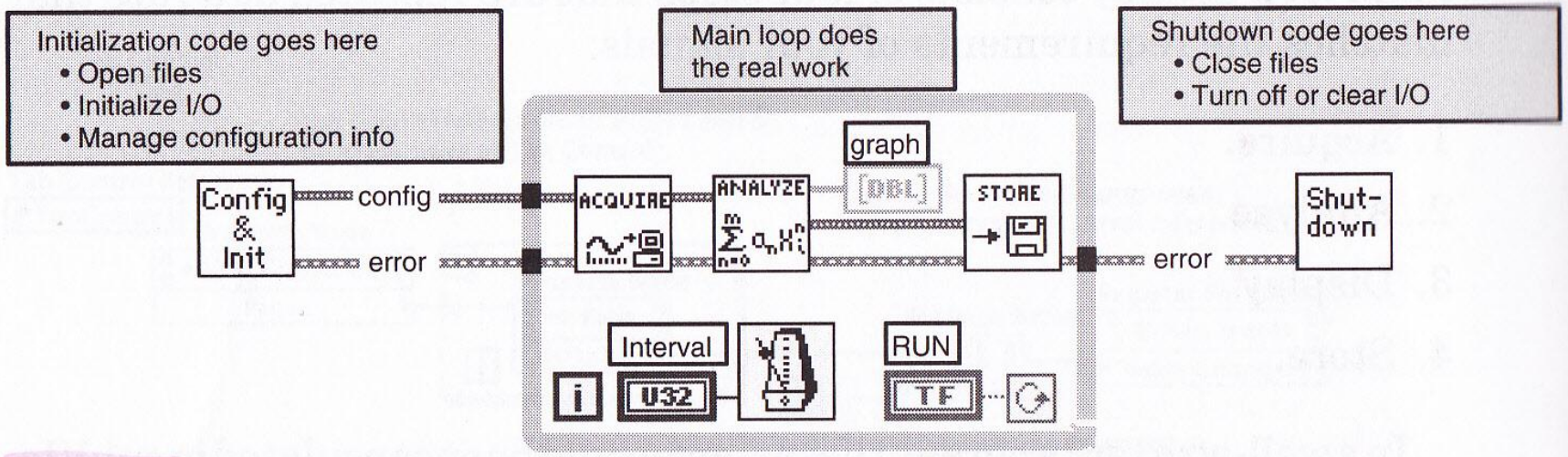
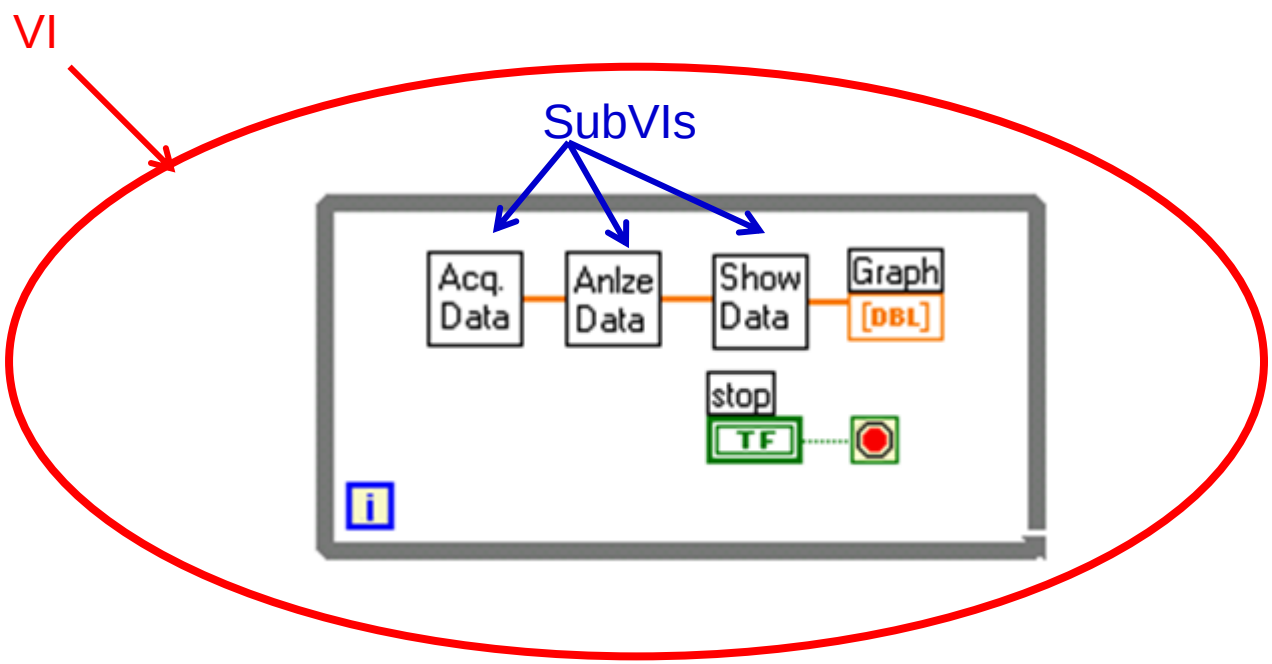


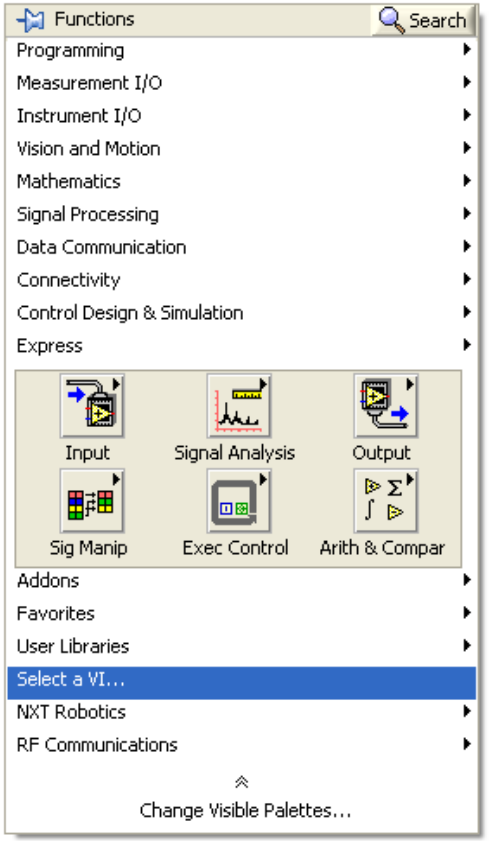
Figure 3.25 Dataflow instead of Sequence structure enhances readability. It has the same functionality as Figure 3.24. Note that the connections between tasks (subVIs) are not optional: they force the order of execution.

SubVIs

- A LabVIEW program is called a Virtual Instrument (VI)
- A subVI is a VI used in a block diagram of another VI
- SubVIs makes the code more readable, scalable, and maintainable
- How to create sub VIs:
  - Create an Icon with I/O connections for the VI

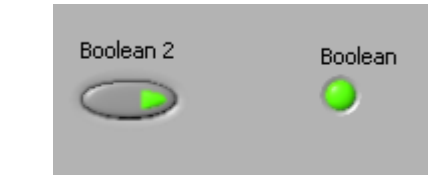


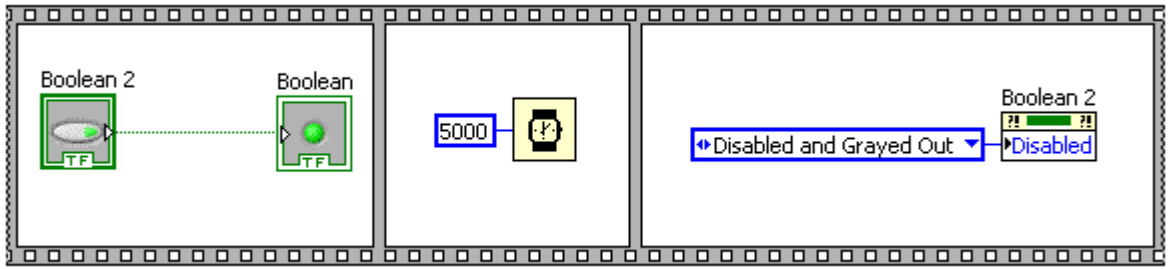
Add SubVI:

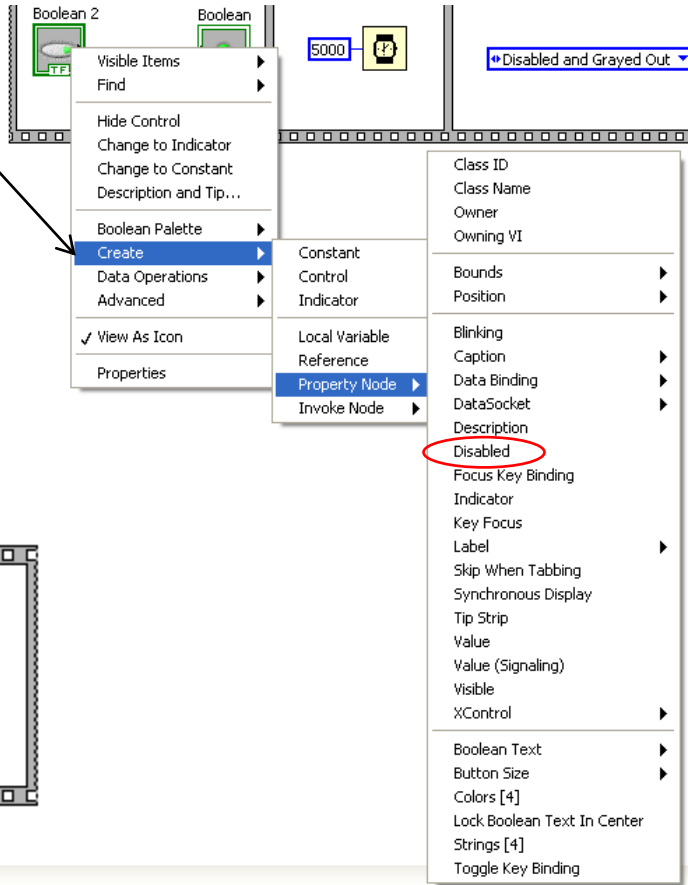


Property Nodes

- Used to manipulate the appearance and behavior of the user interface (Front Panel controls and indicators)
- Limit the number of property nodes due to performance degradation
- Right click the control/indicator to create

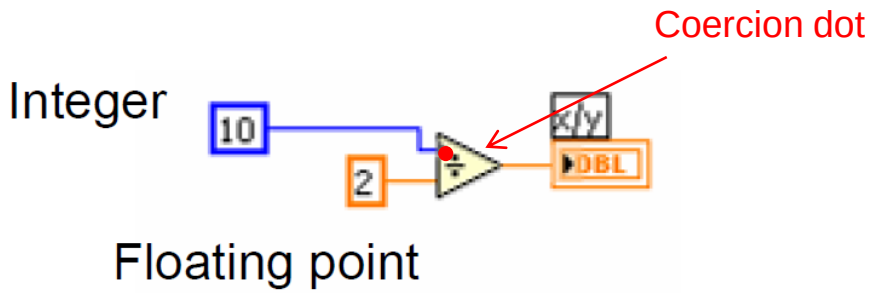




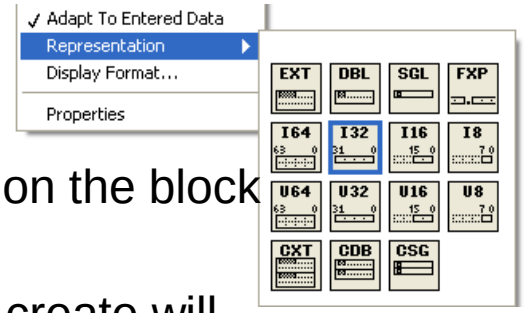


Coercion & type conversion

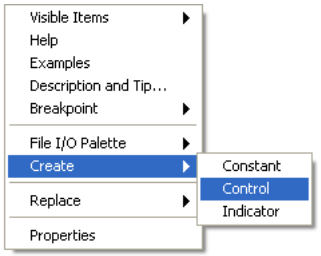
- LabVIEW will convert data types as it sees appropriate

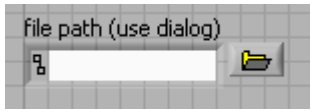


- Avoiding coercions (represented by a red dot) can speed up the application (e.g. using a a <u>conversion</u>)
- To select/change representation, right click the numeric on the block diagram and select **Representation**

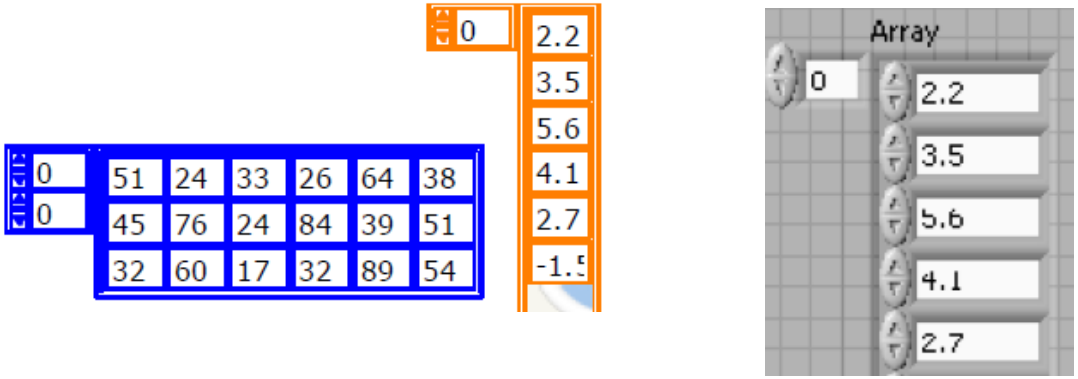


- Right clicking the I/O of a block diagram icon and select <u>create</u> will create the proper data type

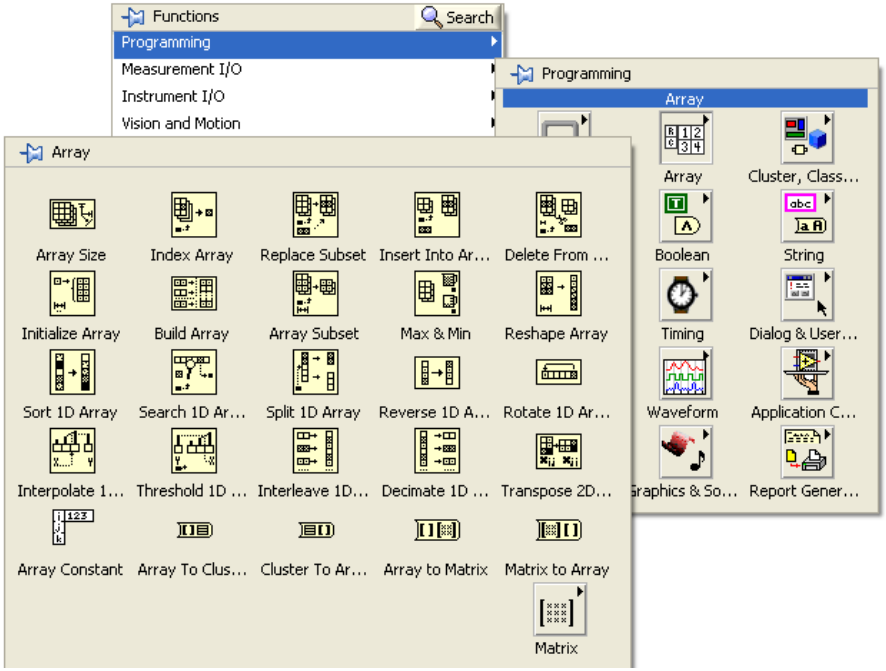


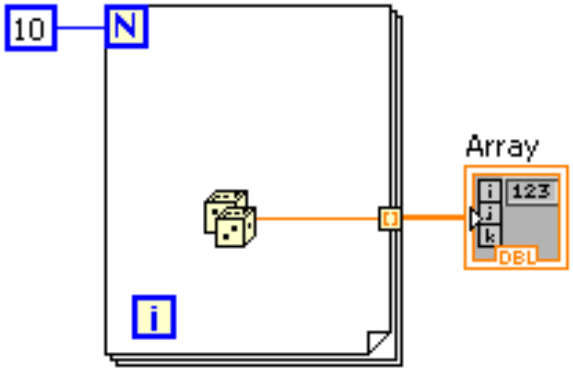


Arrays



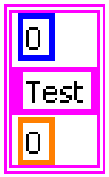
- Can be multidimensional, but cannot be arrays of arrays
- Must have the <u>same data type for each element</u>

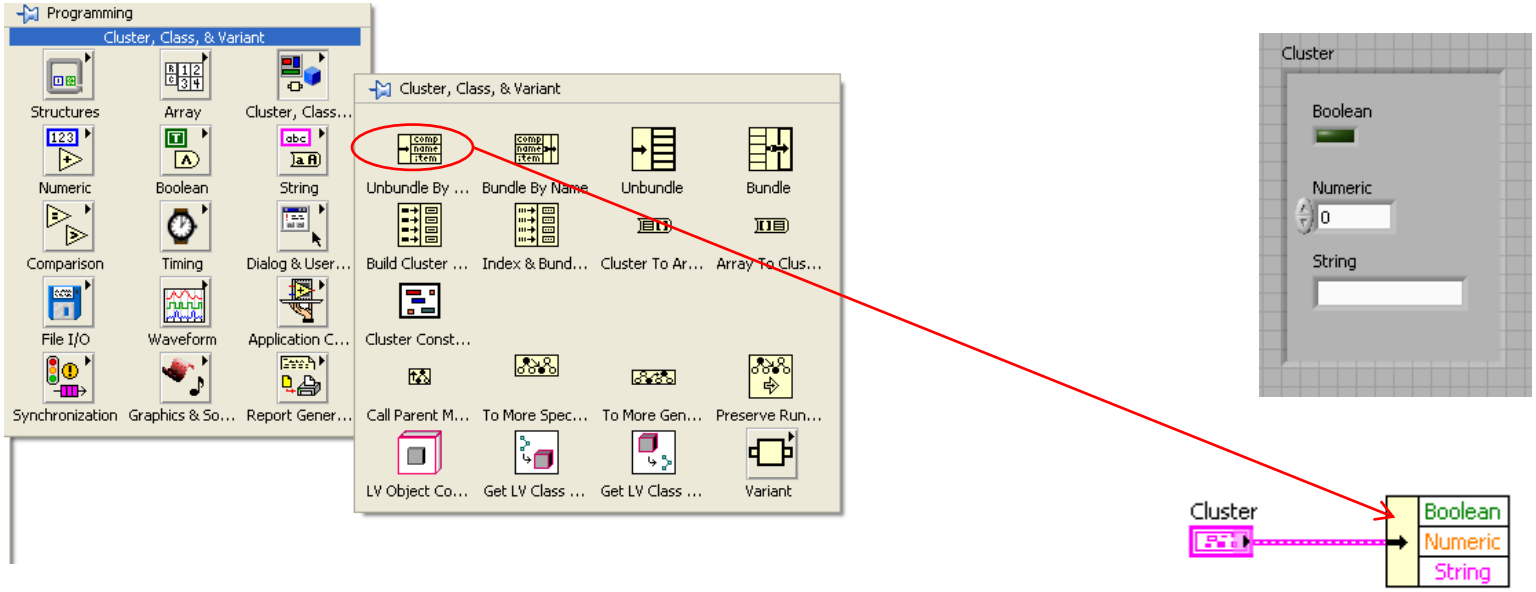




Clusters

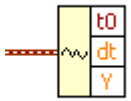
- Used to group related data
  - Reduce the number of terminals (I/O) required on a SubVI
  - Minimize the number of wires on the diagram
- The elements can be of different data types
- Can not contain a mixture of controls and indicators

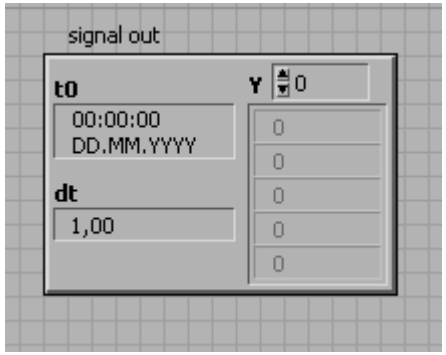


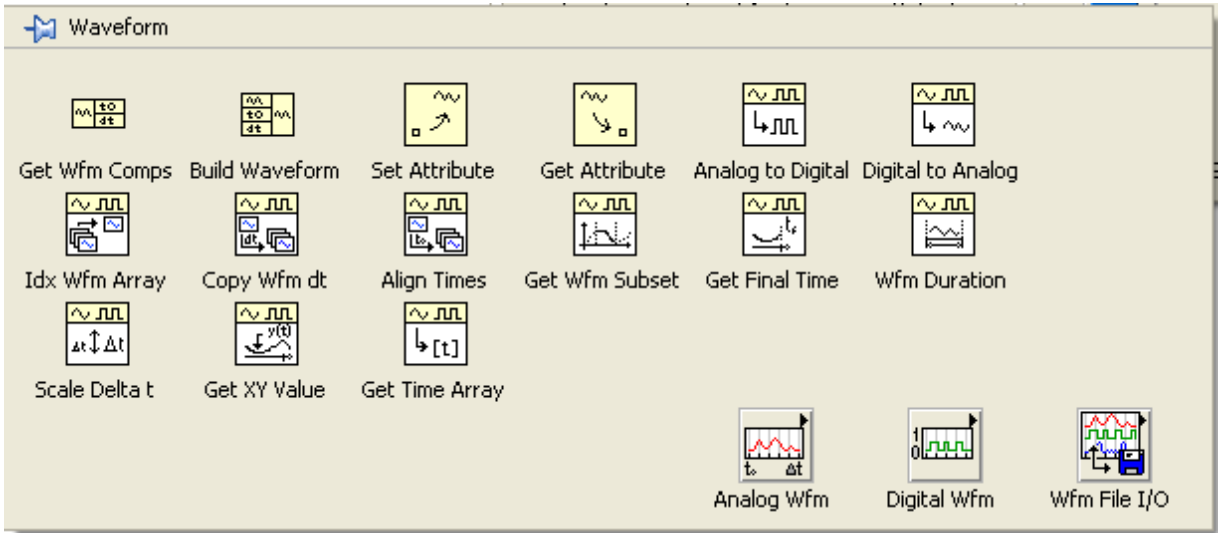


Waveform data type

- The waveform data type carries the **data**, **start time** (t0), and **delta *t*** of a waveform
- Data type used by many signal processing VIs
- Programming – Waveform:







Example: Error Cluster

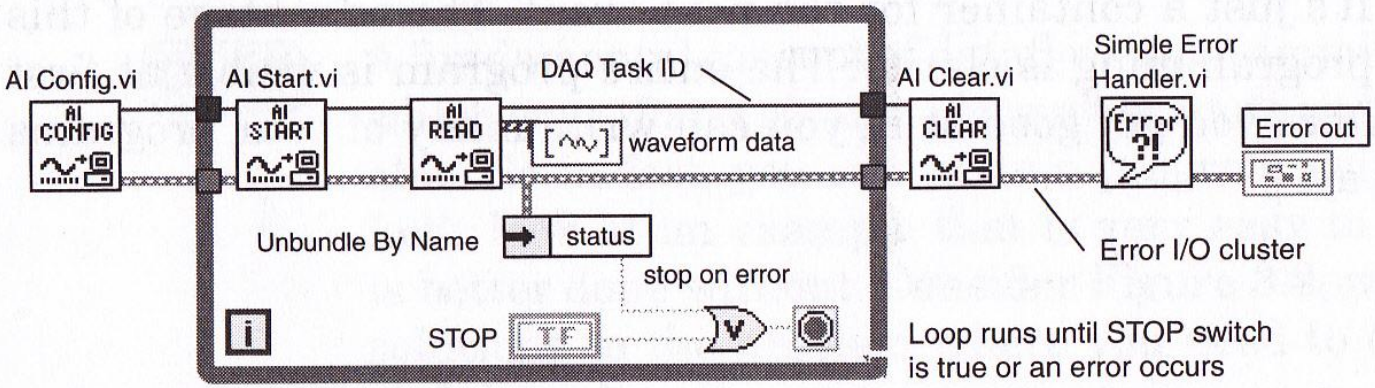
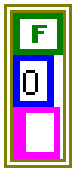
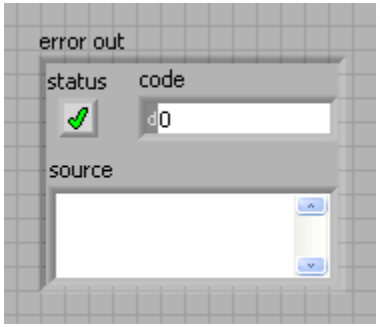


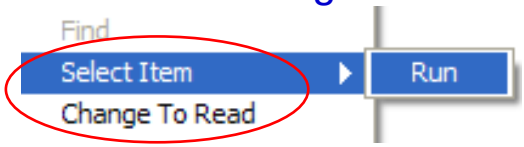
Figure 3.3 A cluster containing error information is passed through all the important subVIs in this program that use the DAQ library. The While Loop stops if an error is detected, and the user ultimately sees the source of the error displayed by the Simple Error Handler VI. Note the clean appearance of this style of programming.

Error constant:



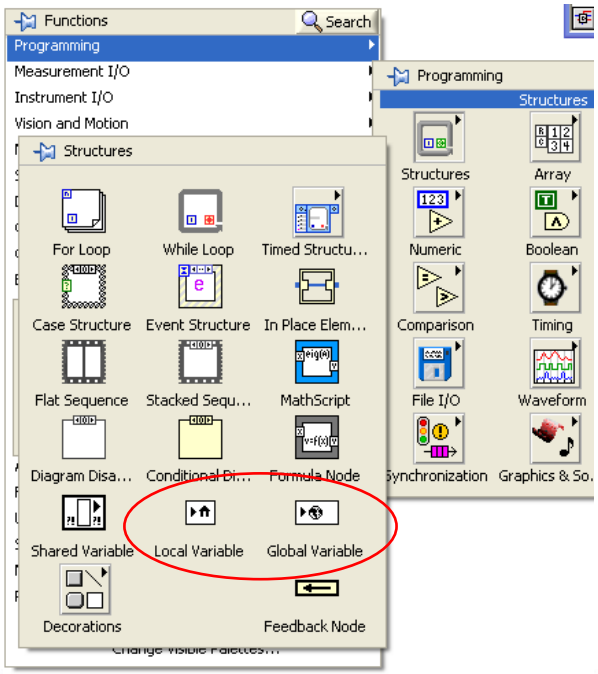


Local variables: Right-click:



Local & Global variables

- **Minimize the use** (especially global variables)
  - **Use wires when possible**
  - Each local variable creates a copy of the data!
- Global variables can create <u>race conditions</u>!
  - When two or more events can occur in any order, but they need to occur in a particular order
  - The data dependency (dataflow) in LabVIEW generally prevents race conditions, but <u>global variables provides a way to violate the strict dataflow</u>
  - Use global variables only when no other good options!



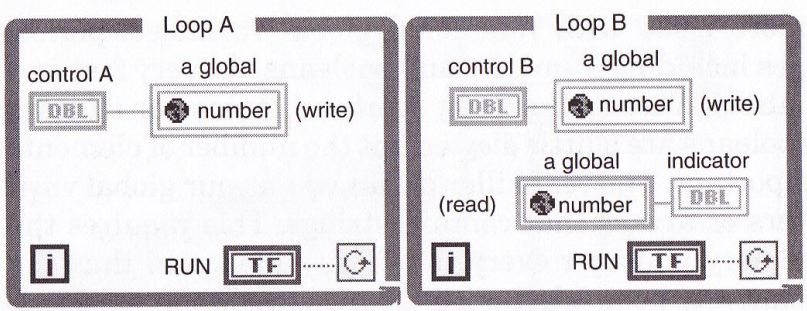
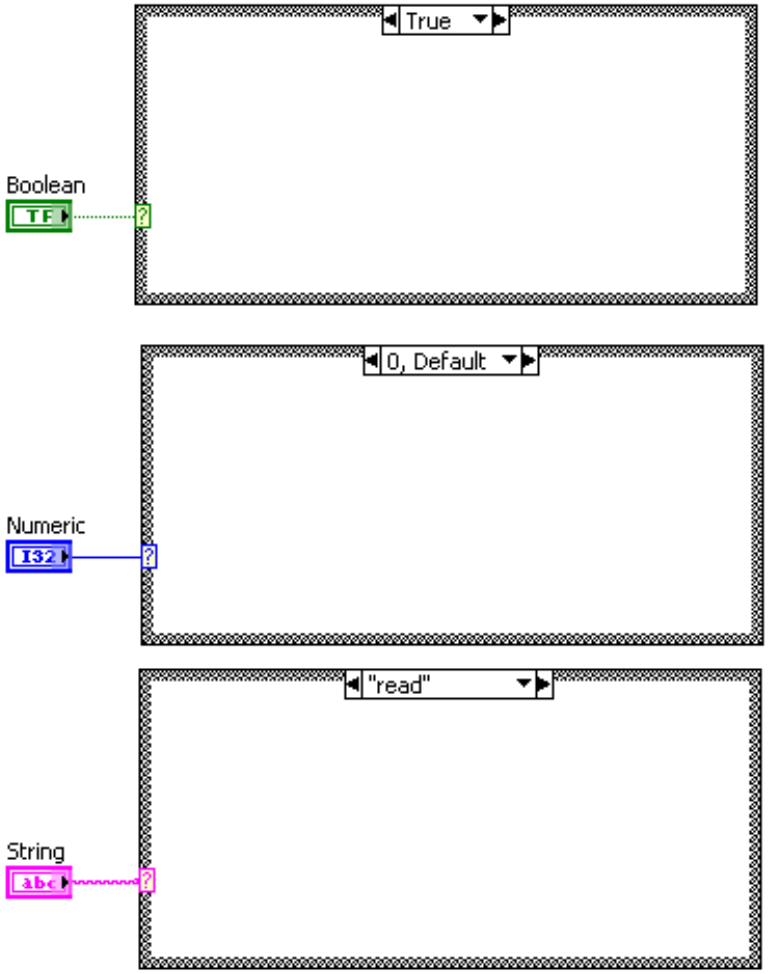
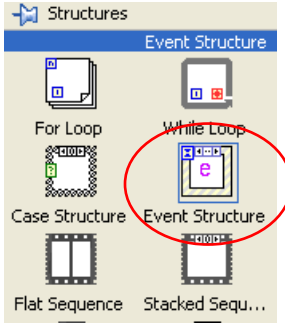


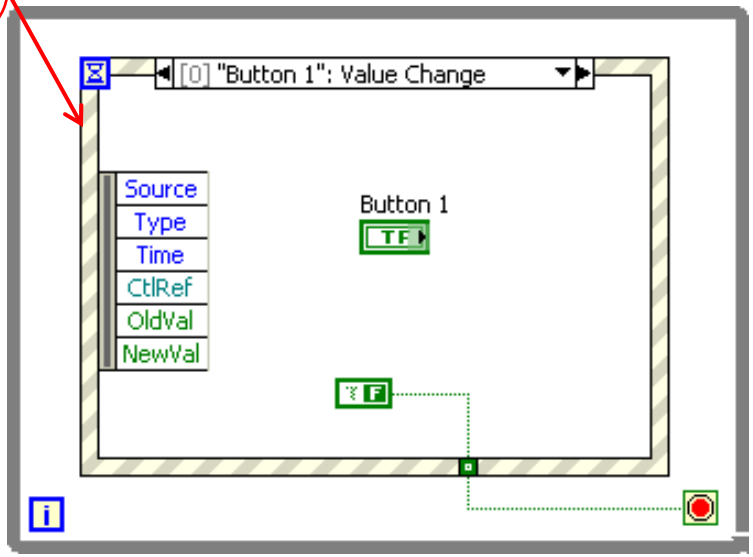
Figure 3.13 Race conditions are a hazard associated with all global variables. The global gets written in two places. Which value will it contain when it's read?

Case structure

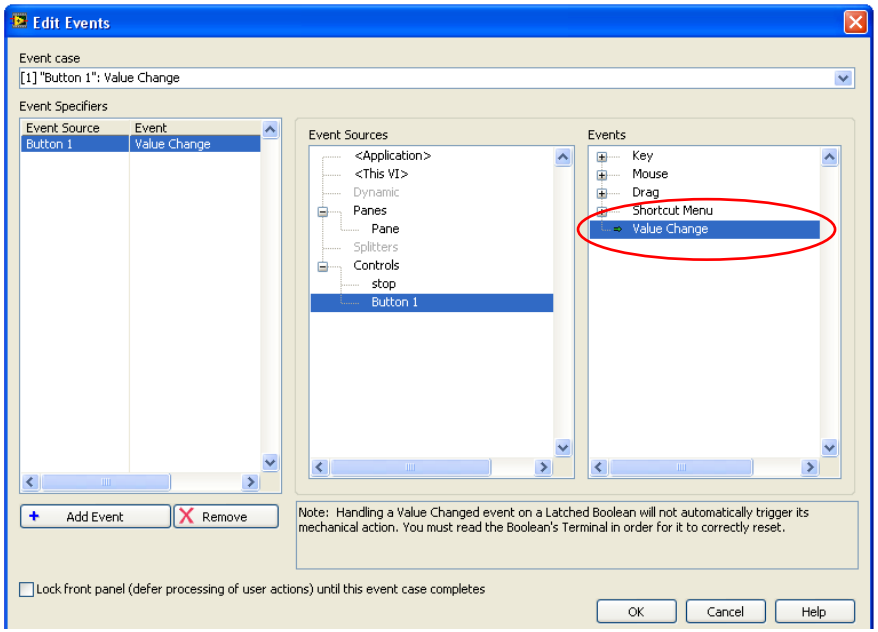


Event Structure

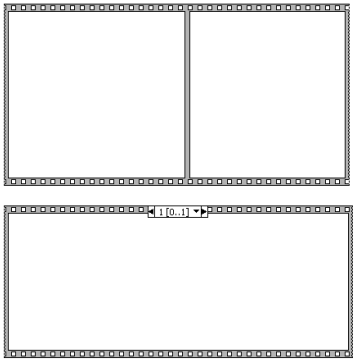




- To limit the CPU usage while waiting for **user interface events** (mouse clicks, key pressed etc.)
  - **Avoids polling!**
- Detects all events!
- Do minimal processing inside event structures!
- How it works:
  - Operating system broadcasts system events (mouse click, keyboard, etc.) to applications
  - Registered events are captured by event structure and executes appropriate case
  - Event structure enqueues events that occur while it's busy



Sequence structure



- Can be used to enforce the order of execution
- Use dataflow programming (data input dependence) to control the dataflow!

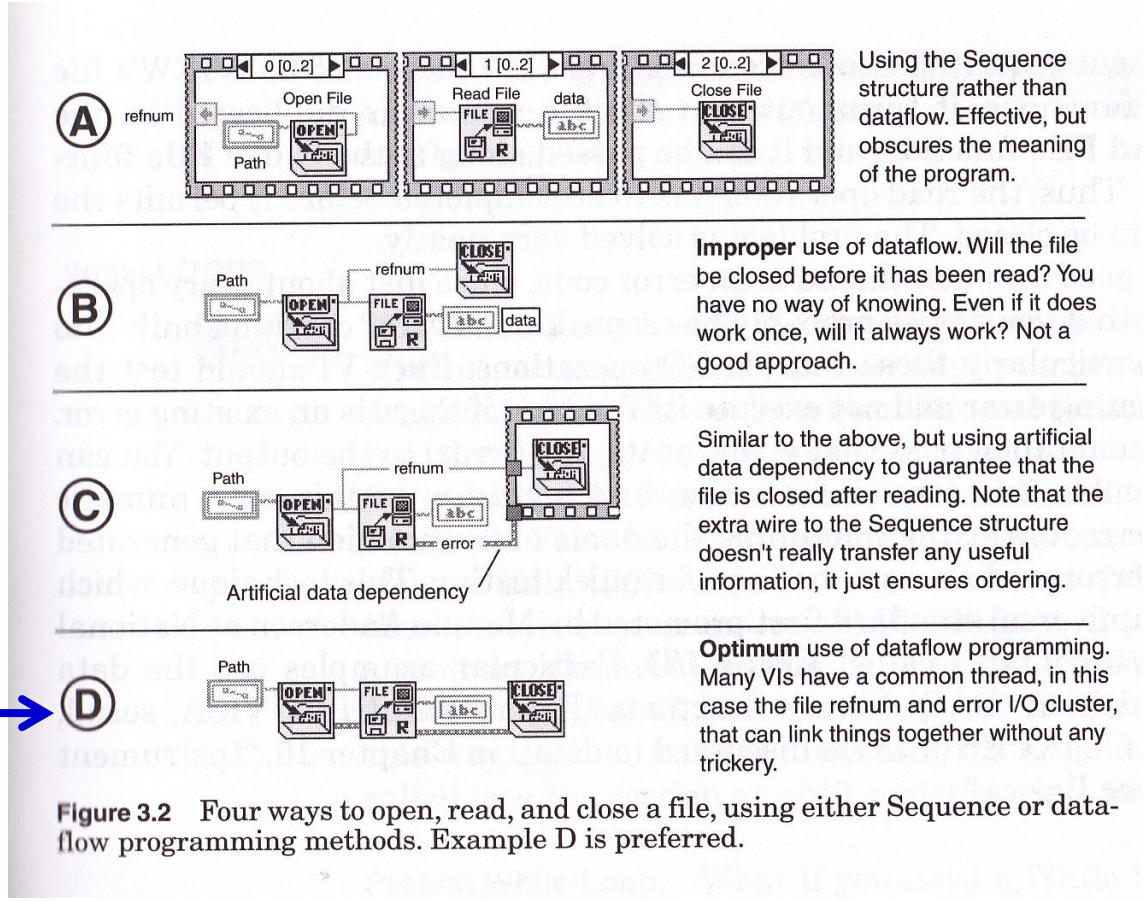


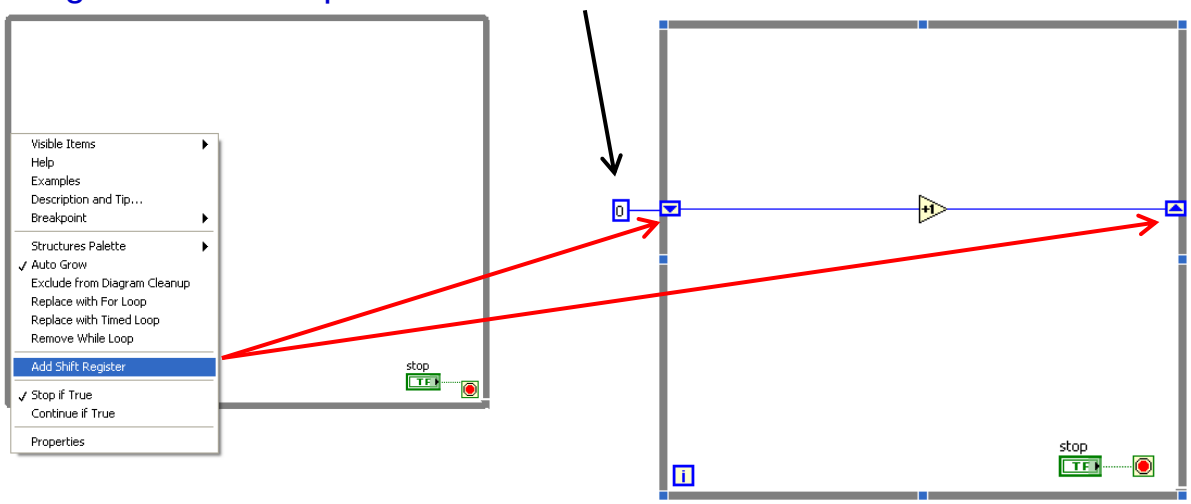
Figure 3.2 Four ways to open, read, and close a file, using either Sequence or dataflow programming methods. Example D is preferred.

Shift registers

- Memory elements available in For Loops and While Loops
- Transfer values from completion of one loop iteration to the beginning of the next
- Initialize the shift registers (unless you want to create a Functional Global)

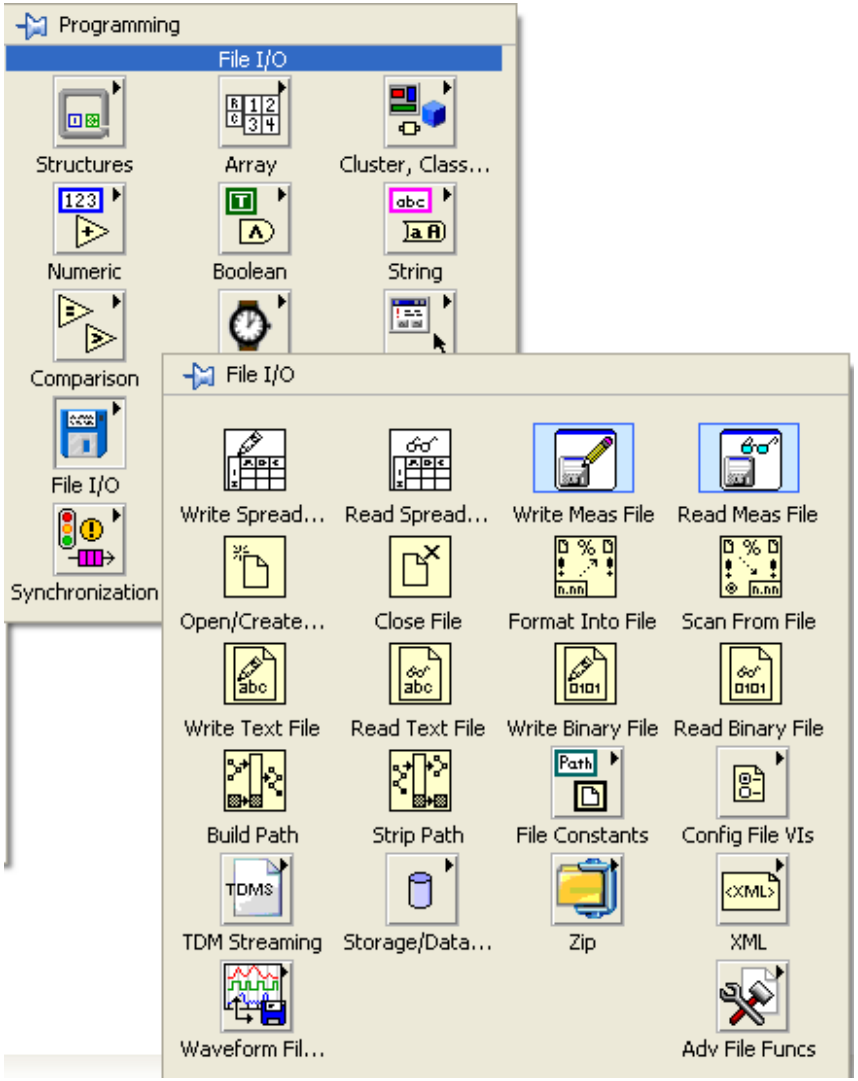
Right-click the loop:

Initialization!

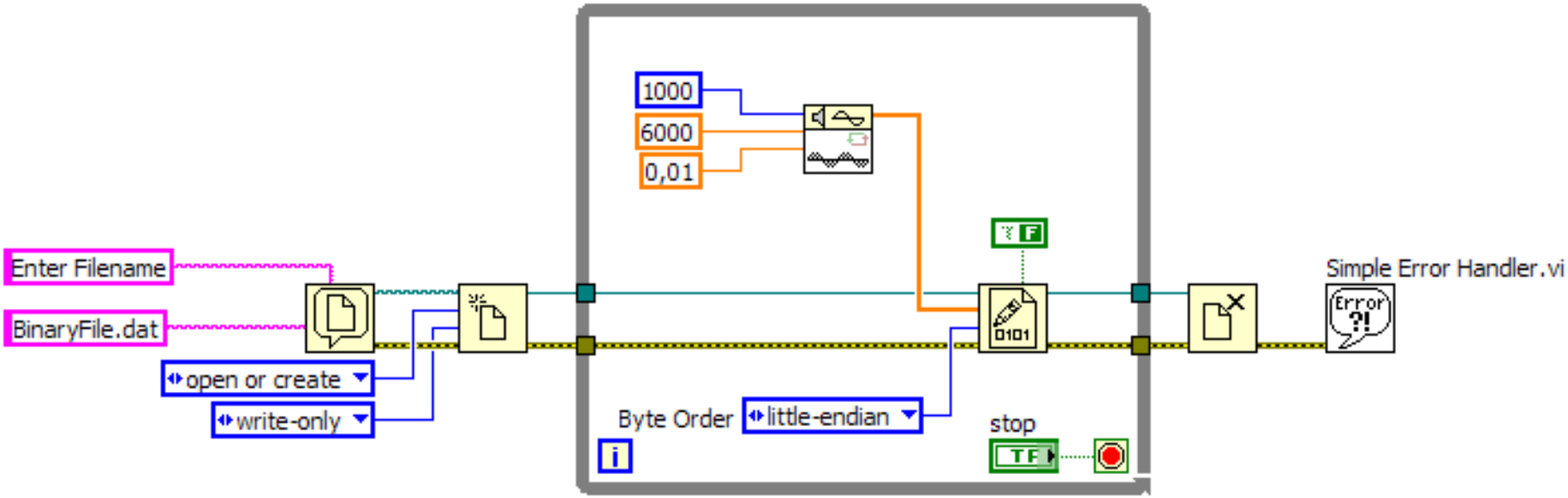


File I/O

- File Types supported in LabVIEW
  - ASCII
  - Binary
  - TDMS
  - Config File
  - Spreadsheet
  - AVI
  - XML



File I/O – Write binary file example



Open File Dialog to select location and name of file to save.

Open new file and set it for write. Also, generate 1000 sample data array that represents a sine wave

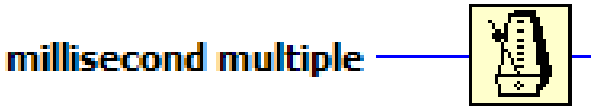
Write data to file. By default, data is written in a 8-byte Little Endian Format. To change formats, convert the array of data to another representation and change the Byte Order enum.

Close file and check for errors.

Software Timing I

- The functions **Wait Until Next ms Multiple**, **Wait (ms)** and **Tick Count (ms)** attempts to resolve <u>milliseconds</u> on a PC within the limitations of the operating system (such as Windows)
- If you need better resolution or accuracy (determinism) you have to use a hardware solution ("software in the loop" degrades precision)
- The system clock/calendar is available using the "Date/time" VIs
  - can give absolute timing (e.g. UTC) and time stamping

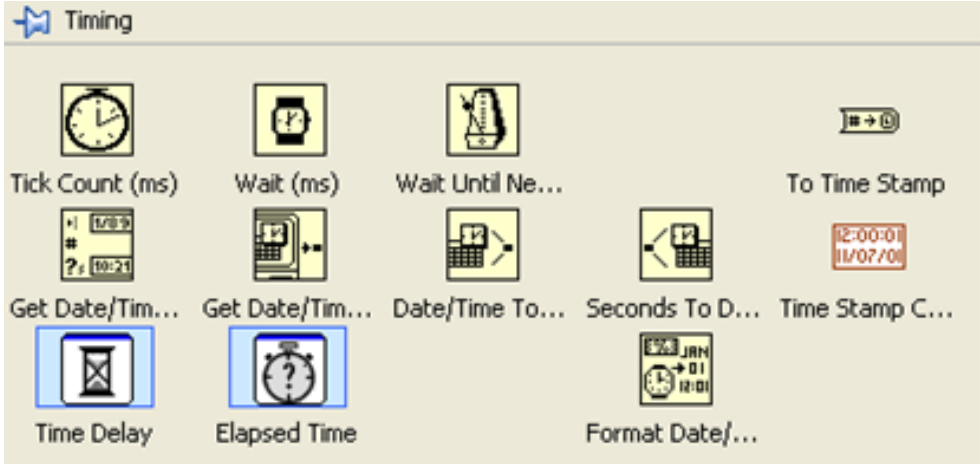
Wait Until Next ms Multiple



Waits until the value of the millisecond timer becomes a multiple of the specified **millisecond multiple**. Use this function to synchronize activities. You can call this function in a loop to control the loop execution rate. However, it is possible that the first loop period might be short. Wiring a value of 0 to the **milliseconds multiple** input forces the current thread to yield control of the CPU.

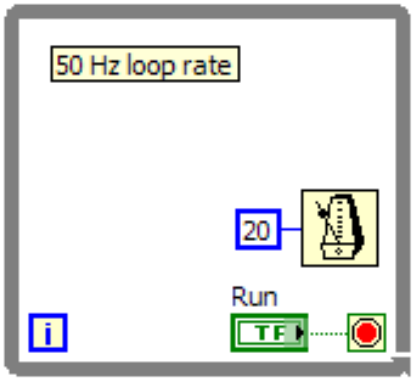
Tick Count (ms)

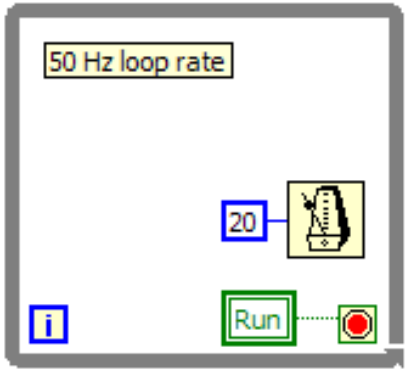
Returns the value of the millisecond timer.



Software Timing II

- To make a while loop run at nice regular intervals add the **Wait Until Next ms Multiple**
  - always use the **Wait Until Next ms Multiple** (or another timer) in a loop to avoid <u>using unnecessary CPU power</u>
  - without any “wait” a while loop will run as fast as possible ...
  - to run the loop as fast as possible but not use all the CPU power place a **Wait (ms)** in the loop with 0-ms to wait
- Two loops can be software synchronized using the **Wait Until Next ms Multiple** in both loops
- To prioritize execution of different parallel loops use <u>Wait functions</u> to slow down lower priority loops in the application



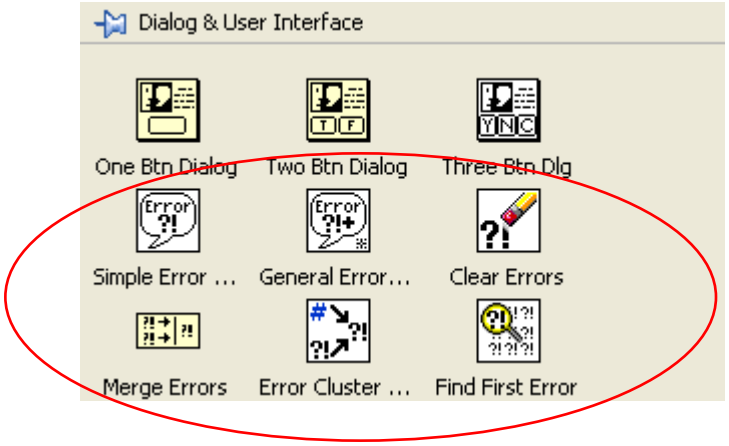


Software Timing III

- If you use software timer functions to control a loop, then you can expect differences in the time interval between each iteration of the loop, depending on what other processes are running on the computer at that instant.
  - If you have several windows open at the same time and you are switching between different windows during your data acquisition, then you can expect a lot of overhead on the Central Processing Unit (CPU), which might slow down the loop that is performing the data acquisition.
  - For this type of timed application it is always better to use hardware timing instead of software timing.

Error handling

- **Propagate the error cluster through every SubVI**
- Merge error lines
- **Possible to catch all errors by using shift registers on the error line**



Report errors using dialog/and or log files

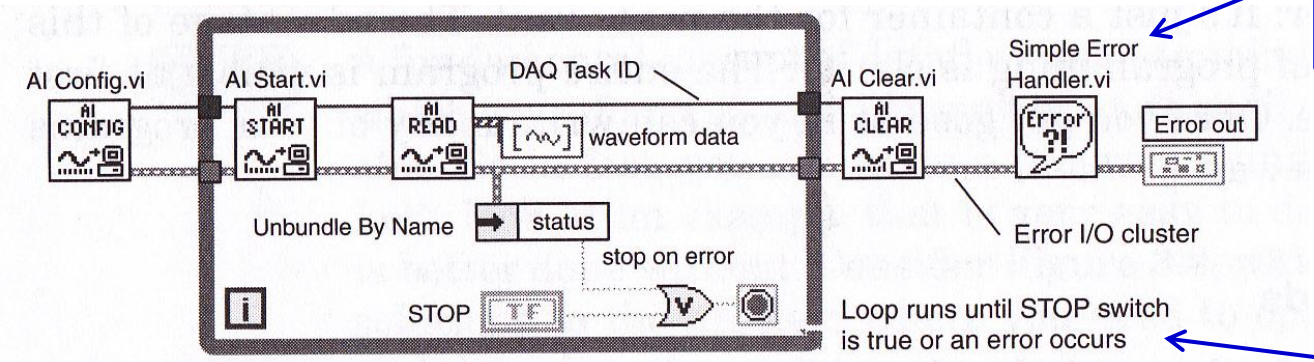


Figure 3.3 A cluster containing error information is passed through all the important subVIs in this program that use the DAQ library. The While Loop stops if an error is detected, and the user ultimately sees the source of the error displayed by the Simple Error Handler VI. Note the clean appearance of this style of programming.

Loop terminates if an error occur

Loop initialization

- Important to preset the controls to a correct initial value at startup of the program
- A sequence structure can be used, see illustration below

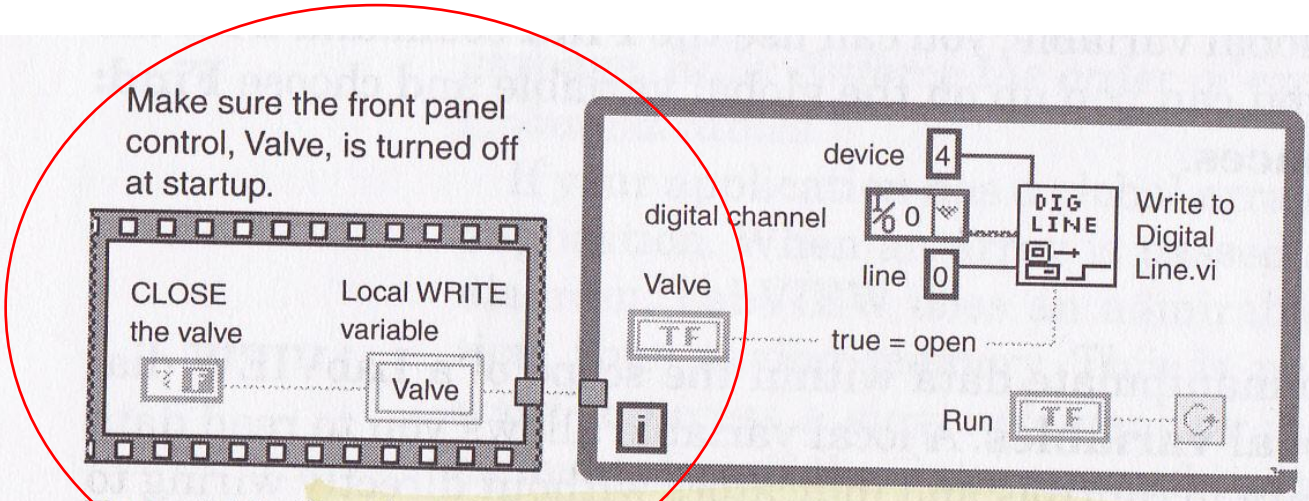


Figure 3.14 Local variables are a convenient way to initialize front panel controls. The Sequence structure guarantees that the initialization is completed before the While Loop starts.

Parallel loops – simplest case

- Sometimes no data need to be exchanged between loops (independent loops)
  - e.g. different sensor signals to be logged at two different rates

- Parallel loops can be stopped using local variables

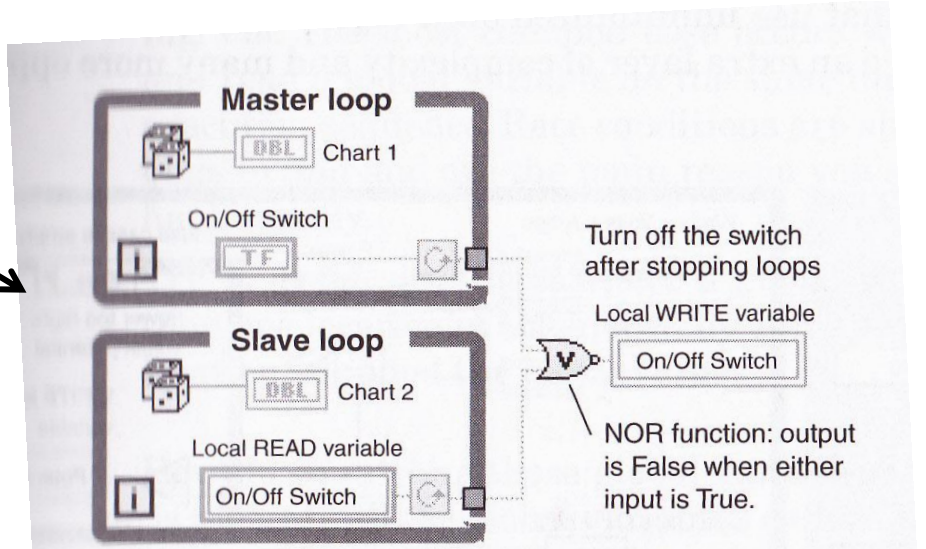
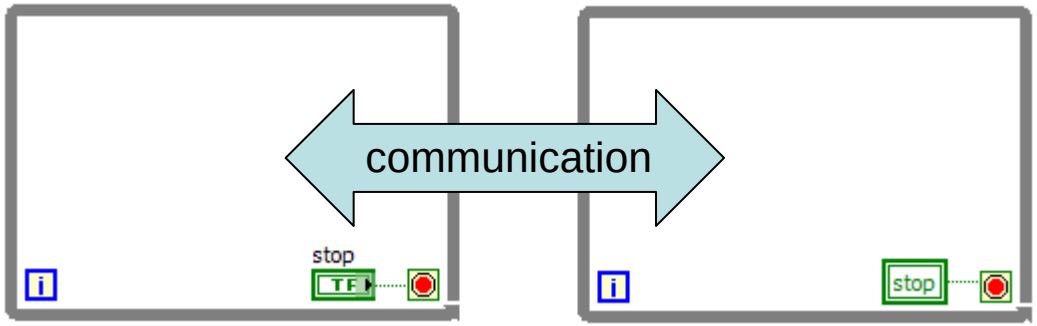


Figure 3.15 This example shows how to use local variables to stop a parallel While Loop. The switch is programmatically reset because latching modes are not permitted for boolean controls that are also accessed by local variables.

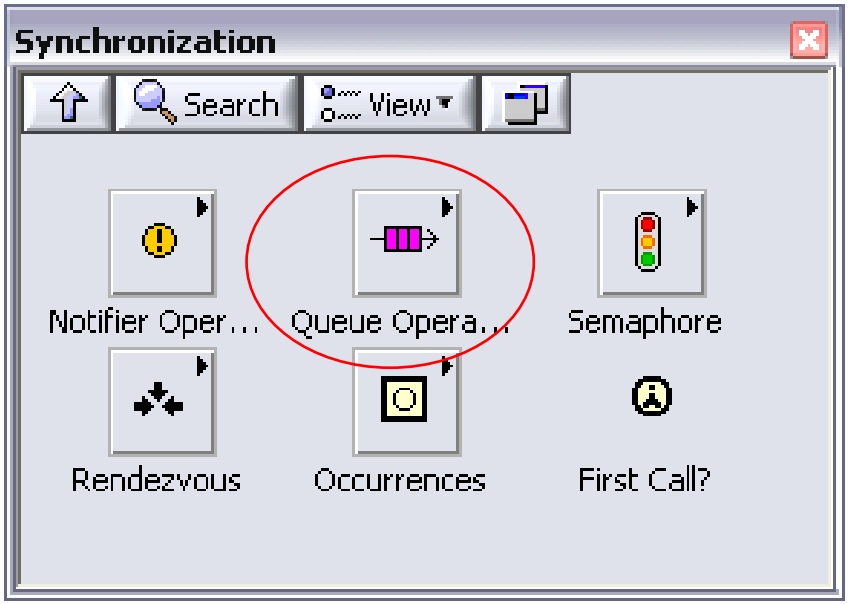
Design Patterns for loop communication

- Master-Slave pattern
- Client – Server pattern
- Producer / Consumer pattern



Loop Communication mechanisms

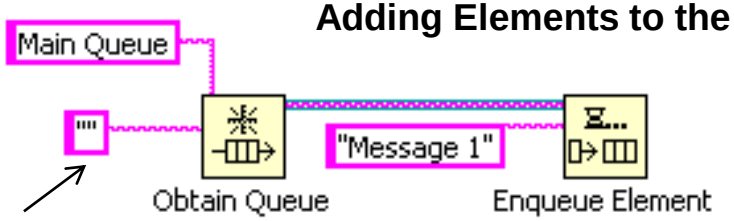
- Variables
- Occurrences
- Notifier
- **Queues**
- Semaphores
- Rendezvous



Queues

- Used for synchronization and data transfer between loops
- Data are stored in a FIFO buffer, and the useful queue depth is limited (only) by the computer's RAM
  - **No data are lost**
- A read (dequeue) from the queue is destructive
  - Data can only be read by one consumer loop (without a new enqueue)
- Different queues must have unique names!

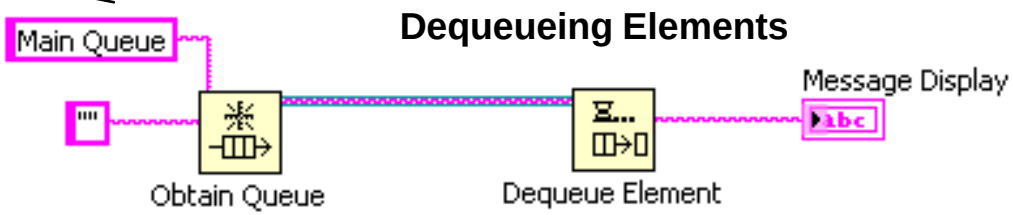
Adding Elements to the Queue



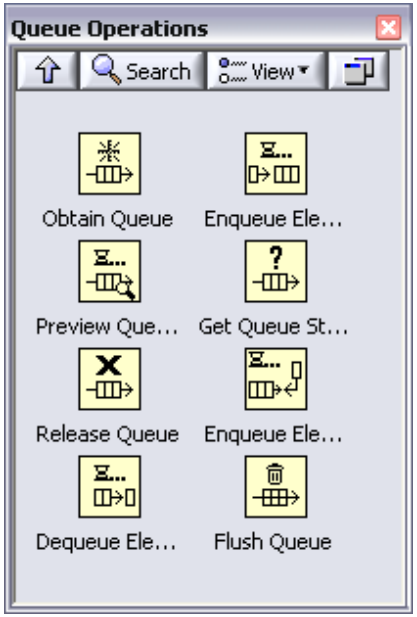
Select the data type the queue will hold

Reference to existing queue in memory

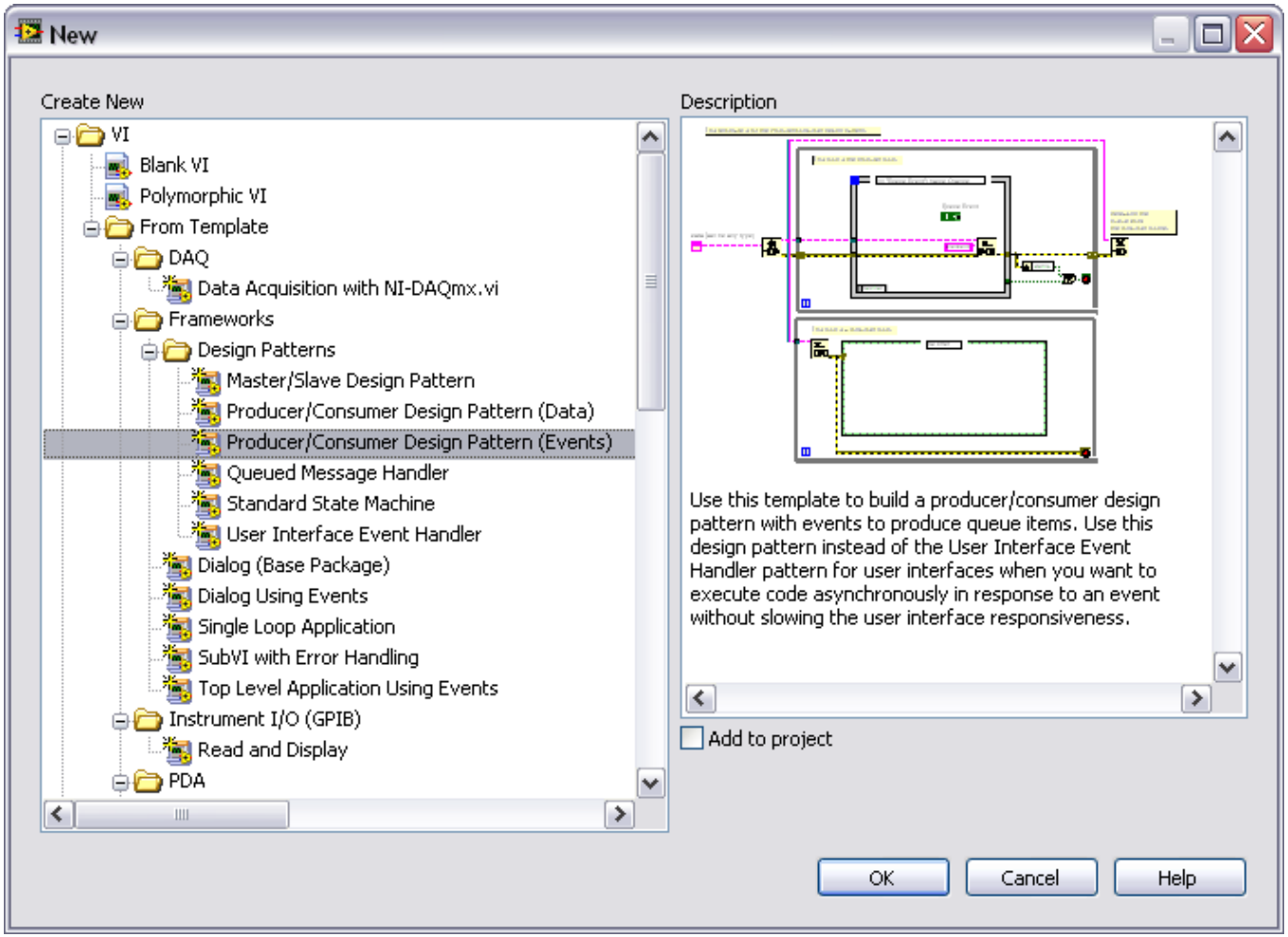
Dequeueing Elements



Dequeue will wait for data or time-out



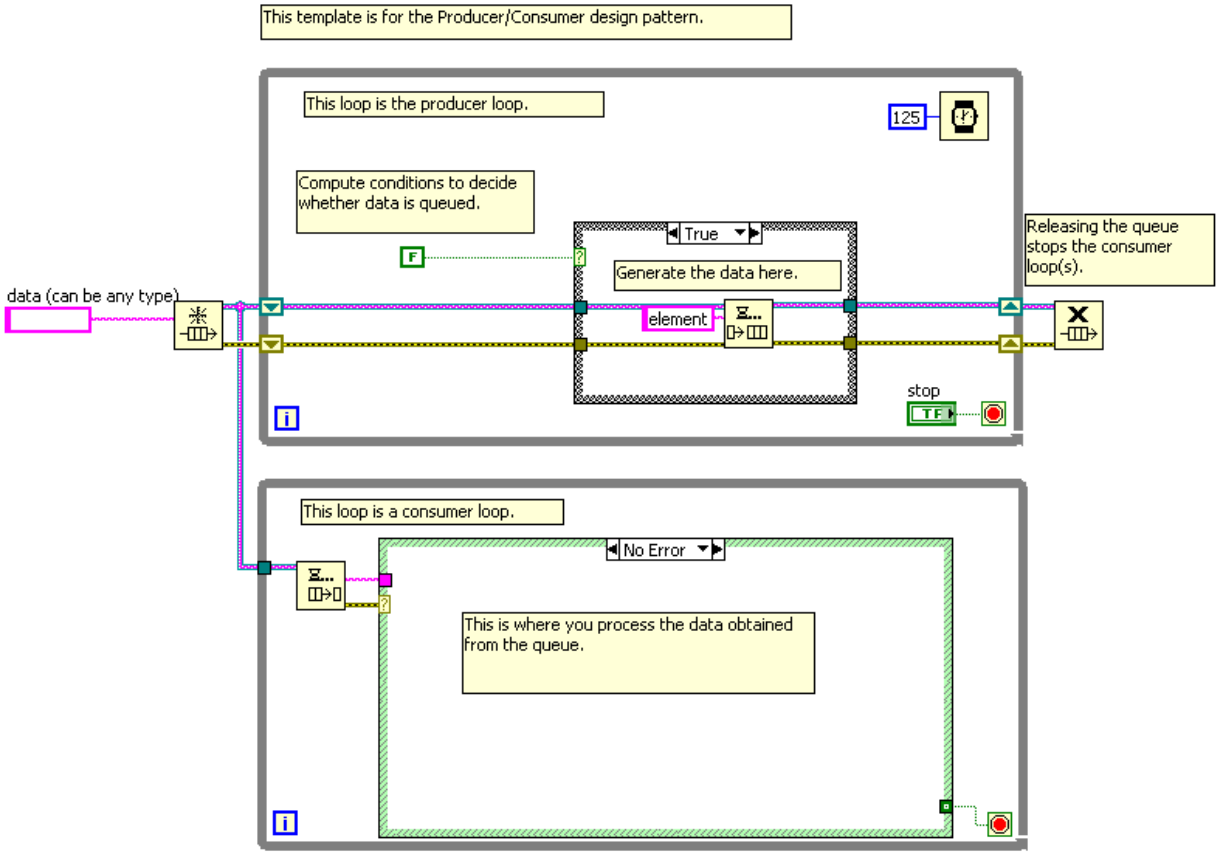
Design patterns (templates)



http://zone.ni.com/devzone/cda/tut/p/id/7605

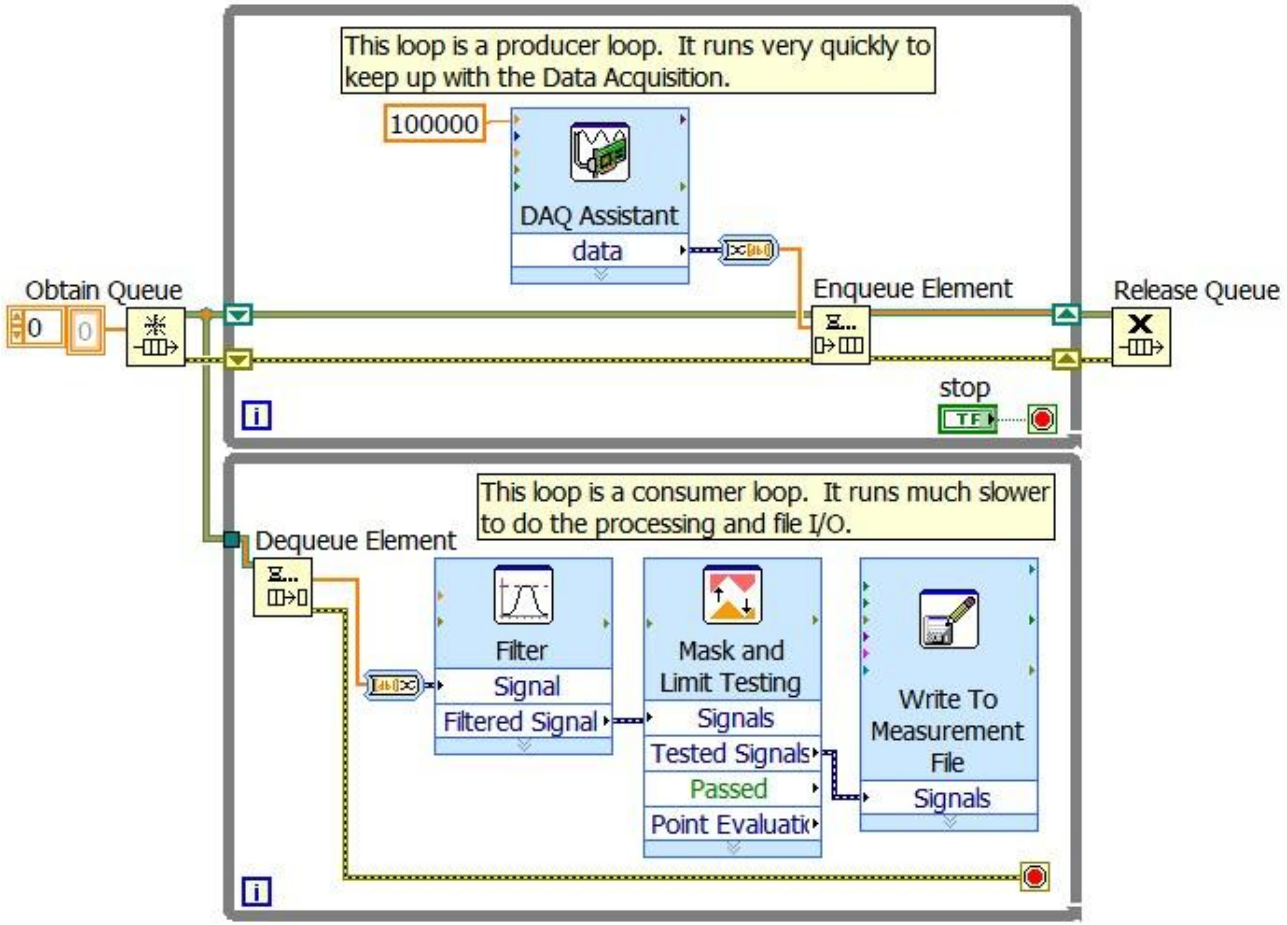
Producer – consumer

- **Queues** are used for loop communications in multi-loop programs, to execute code in parallel and at different rates
- The queues buffer data

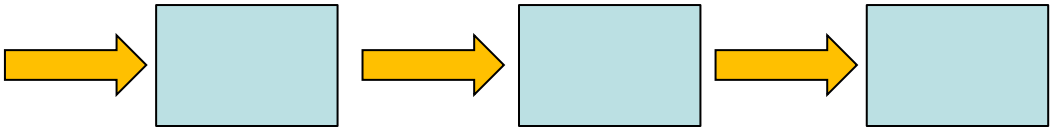


Producer – consumer Example

When we have multiple tasks that run at different speeds and cannot afford to be slowed down.

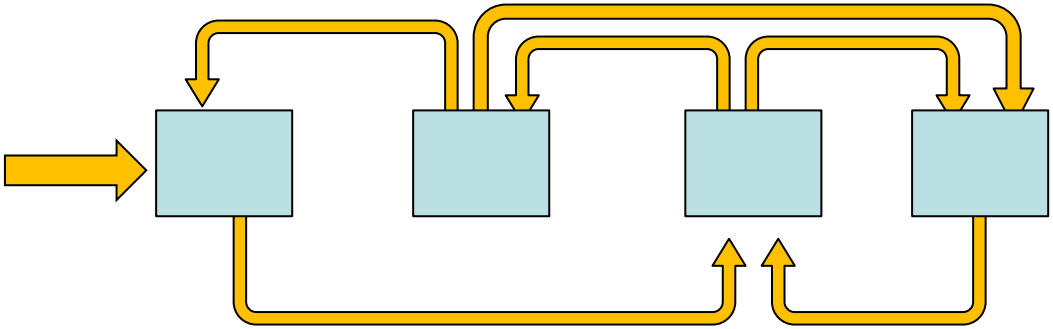


State machines - background



Static Sequence

Known order of execution

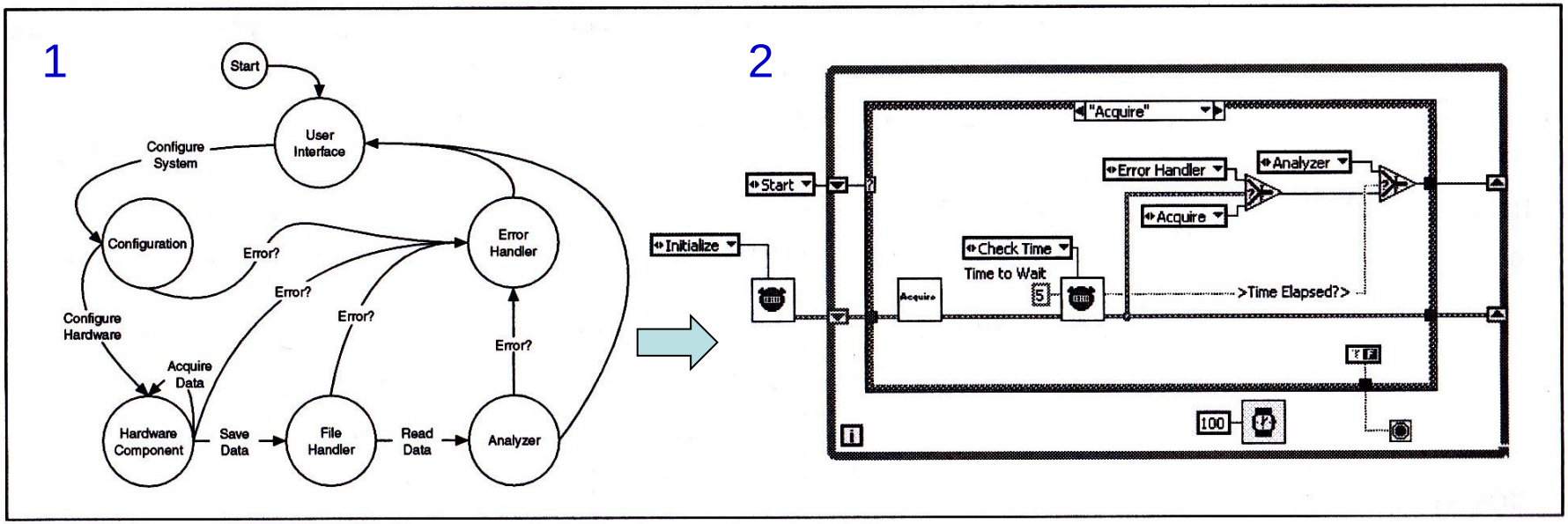


Dynamic Sequence

Distinct states can operate in a programmatically determined sequence

State machine design

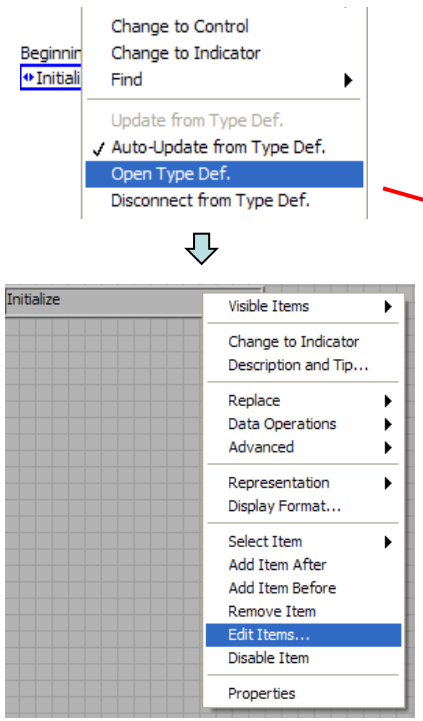
1. Draw the **state diagram**
2. Translate the state diagram into **LabVIEW code**

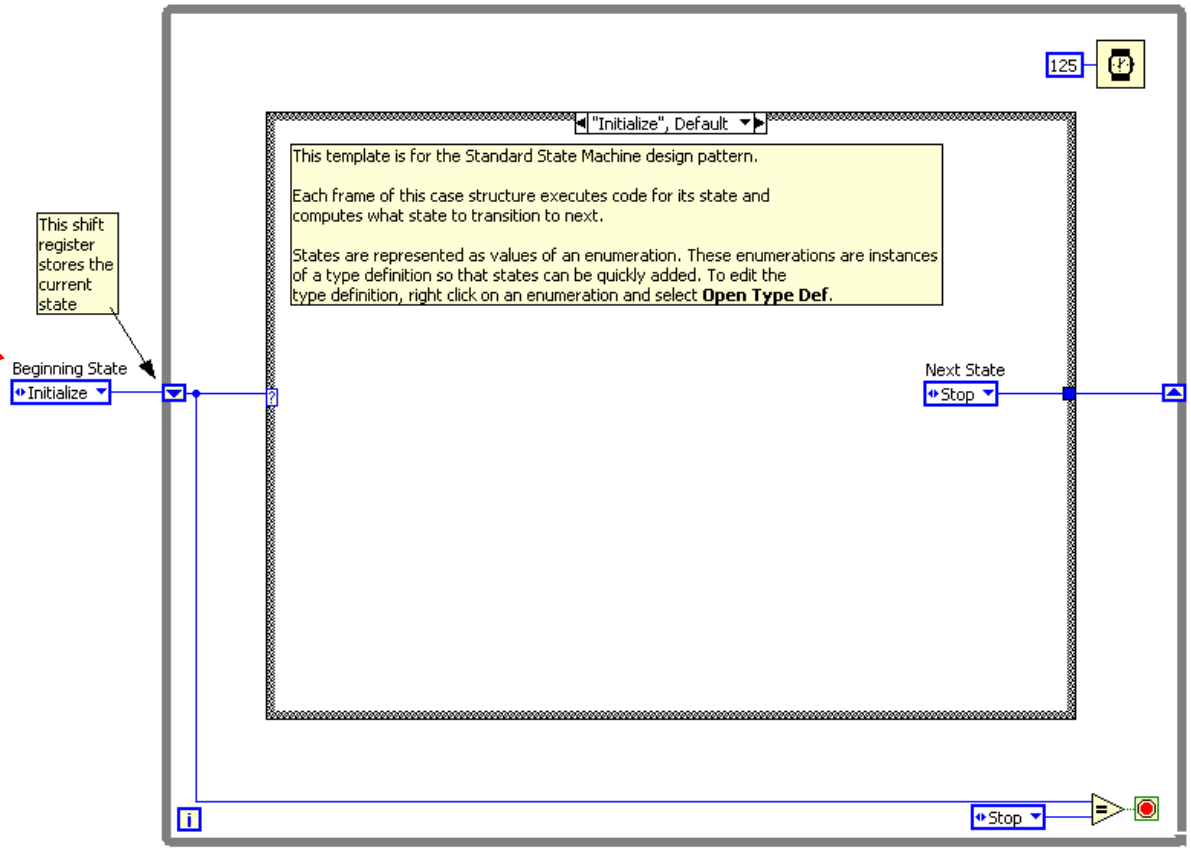


Standard State machines in LabVIEW

- Case structure inside of a While loop
- Each case is a state
- Current state has decision-making code that determines next state
- Use enumerated constants (**typedefs)** to pass value of next state to shift registers

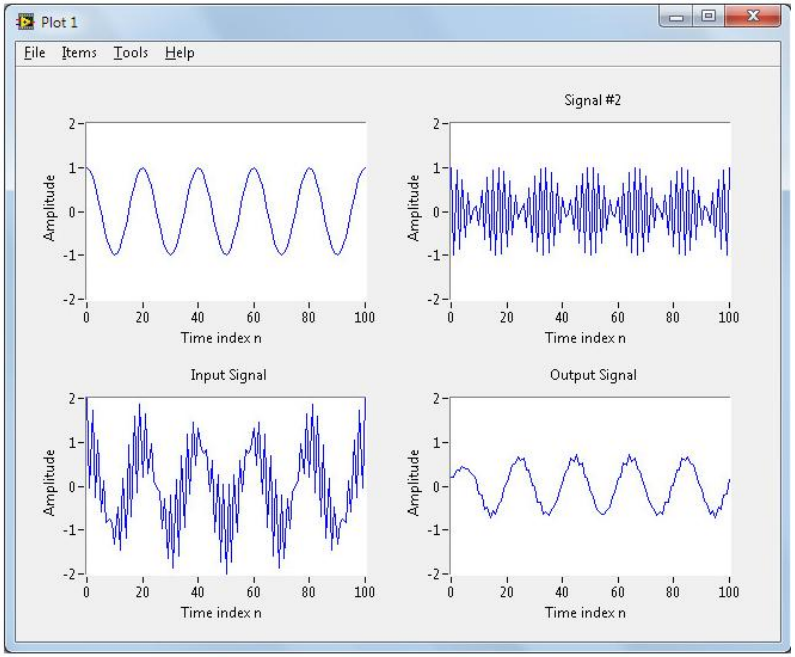
Edit/add/Remove states:





MathScript

- Adds math-oriented, textual programming to the LabVIEW graphical development environment
- General compatibility with widely used **.m file script** syntax (not all Matlab functions are supported)
- Reuse your your Matlab .m files
- Very useful for algorithm development
  - compact code for matrix operations etc
- Available also for RT-targets



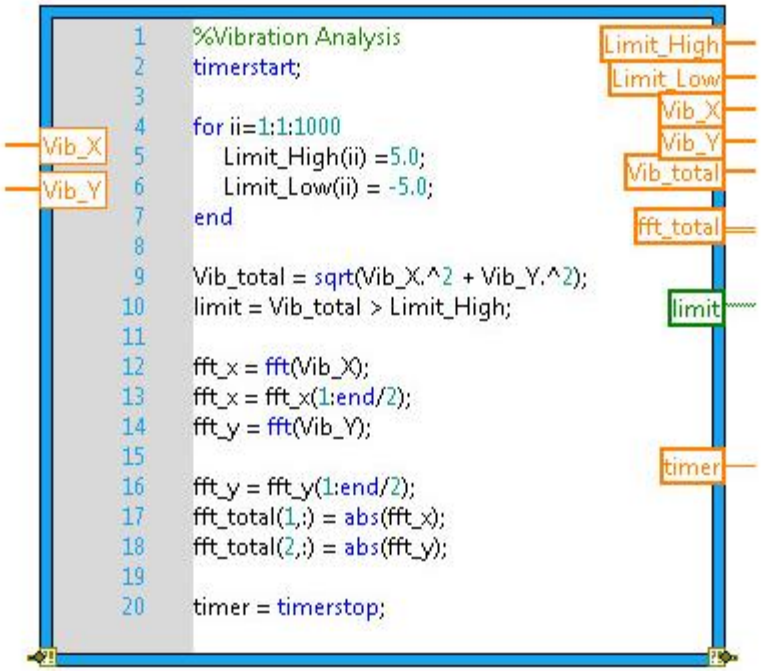
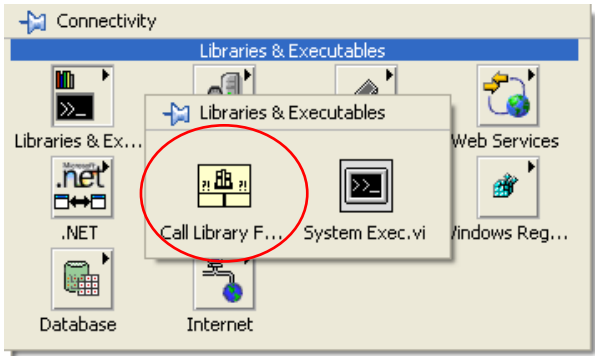
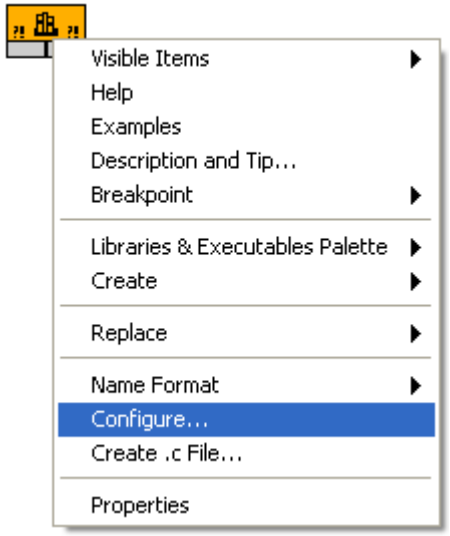


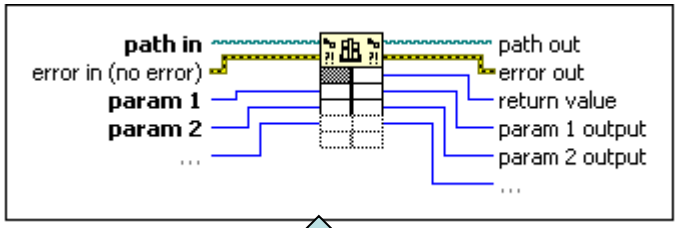
Figure 11. MathScript Node Places Your Custom .m File Code Inline with Graphical G Code

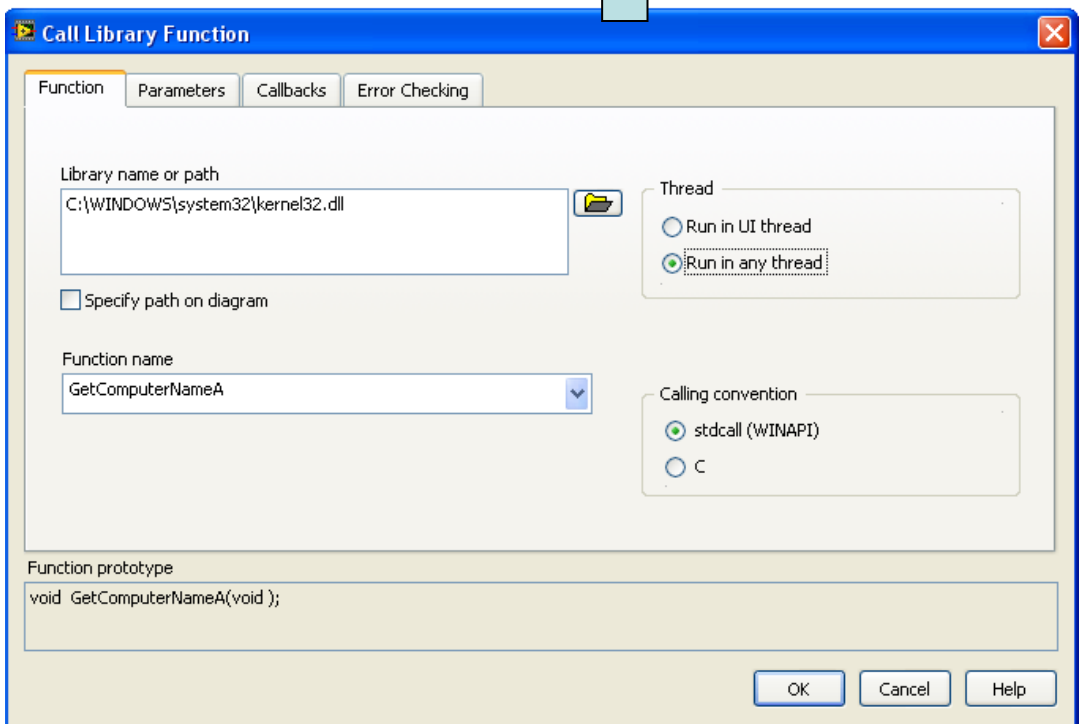
Connectivity – Using DLLs in LabVIEW

- Call Library Function
  - To use DLLs in LabVIEW
  - Almost any compiler can generate a DLL



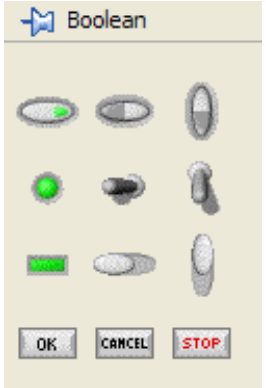


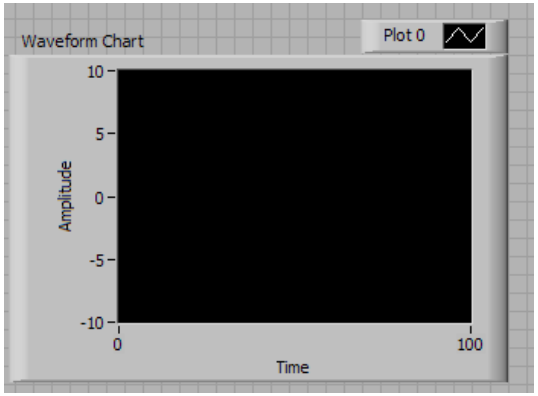


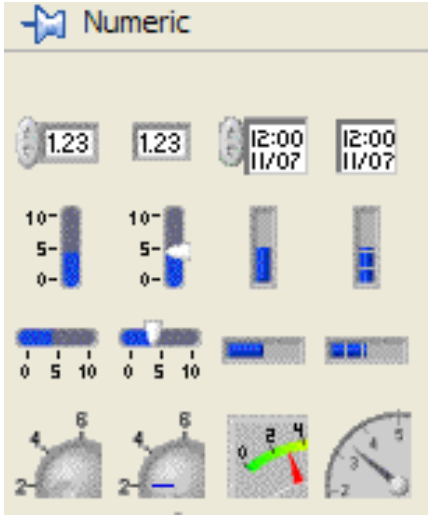


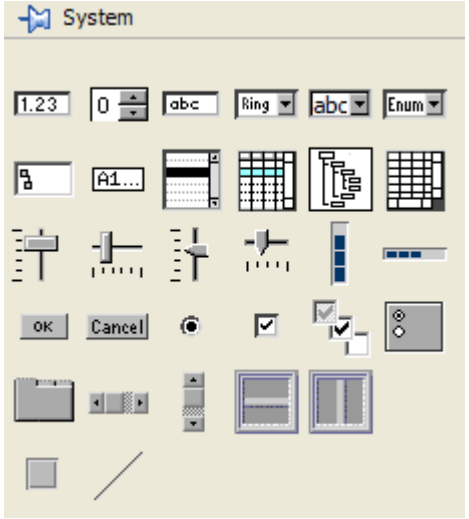
Visualization

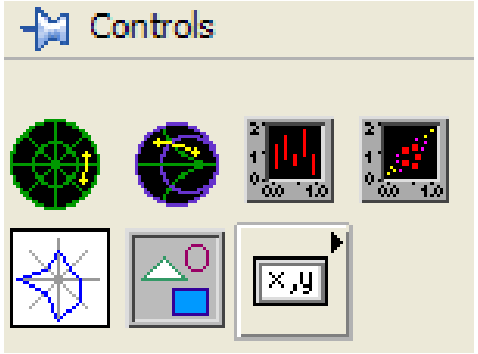
- Displaying data can require considerable computer resources
- Improving display performance:
  - using smaller graphs and images
  - display fewer data points (down sampling)
  - less frequent display updates











Building an application I

- Chose a design architecture (design pattern)
- Start with a paper design ….
  - draw block diagrams
  - draw flow charts/state diagrams
- Prototype the user interface
  - helps defining necessary controls and indicators
- Divide-and-conqure
  - break the problem(s) into manageable blocks
  - make <u>SubVIs</u> for each function
  - put the entire design together

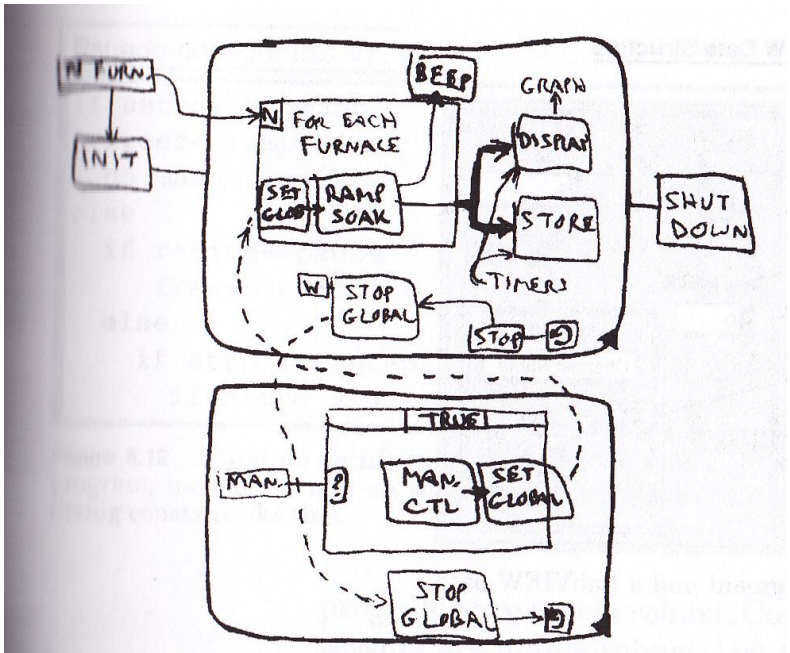


Figure 8.9 Sketch for the main VI used in VBL. Not very detailed at this point, but it sure gets the point across.

Building an application II

- The **spiral model** is a software development process
- Identify risks, and analyze the most important risks
- Data acquisition example:
  - the highest risk is whether the system can properly interface to all hardware devices and acquire, analyze, store and display the data quickly enough
  - therefore; create prototypes to test device communication, acquisition rates etc.
  - then evaluate the results, and continue the process

Spiral model