FYS3240
PC-based instrumentation and microcontrollers

LabVIEW programming II

Spring 2015 – Lecture #3
Dataflow programming

- 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 all required inputs, it produces output data and passes that data to the next node in the dataflow path. The movement of data through the nodes determines the execution order of the functions on the block diagram.
LabVIEW Graphical Programming – Dataflow
How LabVIEW Implements Multithreading

• Parallel code paths on a block diagram can execute in unique threads
• LabVIEW automatically divides each application into multiple execution threads (originally introduced in 1998 with LabVIEW 5.0)
LabVIEW Example – Two separate threads

Two separate tasks that are not dependent on one another for data will run in parallel (two threads) without the need for any additional program code.
How LabVIEW Implements Multithreading

1. LabVIEW compiler analyzes diagram and assigns code pieces to “clumps”

2. Information about which pieces of code can run together are stored in a run queue

3. If block diagram contains enough parallelism, it will simultaneously execute in all system threads

... # of threads scales based on # of CPUs
Basic LabVIEW code architecture

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
Type conversion

- LabVIEW will convert data types as it sees appropriate.
  - Coercion dot (means that a type conversion is done)

- Avoiding coercions (represented by a red dot) can speed up the application.
- 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 **create** will create the proper data type.
Arrays

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

Simple method to create an array:
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 \((t_0)\), and delta \(t\) of a waveform

- Data type used by many signal processing VIs
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 race conditions!**
  - 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 global variables provides a way to violate the strict dataflow
  - Use global variables only when no other good options!
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!

Program like this!

---

**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)
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 milliseconds 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.
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 using unnecessary CPU power
  – without any “wait” a while loop will run as fast as possible ...
  – wiring a value of 0 to the **Wait functions** forces the current thread to yield control of the CPU.

• Two loops can be software synchronized using the **Wait Until Next ms Multiple** in both loops

• To prioritize execution of different parallel loops use **Wait functions** 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 (jitter) 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.
  – In DAQ applications you should use hardware timing instead of software timing, if possible.
LabVIEW Timed Loop

- 1 kHz internal clock (Windows)
- 1 MHz internal clock (for RT-targets)

A Timed Loop gives you:
- possibility to **start the loop at a precise time** (using a time stamp)
- **phase (offset) control**
- possibility to **specifies the processor** you want to handle execution
- execution **priority**
- precise **determinism in a real-time operating system**

When Timed Loops as used on Windows (no RTOS) the OS can preempt your structure at any time to let a low-priority task run (based on “fairness”)! 
Execution priority

• In LabVIEW **while loops** run at normal priority, and **timed loops** run between time-critical priority and above **high priority**.

• Therefore if you would like to have control of the priority of each aspect of your application you can use **timed loops**, and set the priority between them using the priority input.

---

File - VI Properties » Execution:
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

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 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
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
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 in a FIFO structure in PC RAM.

**Software timing** - the producer loop will run every 125 ms (8 Hz).
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:**

- Change to Control
- Change to Indicator
- Description and Tips...
- Replace
- Data Operations
- Advanced
- Representation
- Display Format...
- Select Item
- Add Item After
- Add Item Before
- Remove Item
- Edit Item
- Disable Item
- Properties
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 Matlab .m files
- Very useful for algorithm development
  - compact code for matrix operations etc
- Available also for RT-targets
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 SubVIs for each function
  - put the entire design together
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