LabVIEW State Machine Architectures

Presented By Scott Sirrine Eaton Corporation
State Machines- Personal Background

• Employment
  • Lead Product Engineer (Software development) at Eaton Corporation’s R&D facility in Galesburg MI.

• LabVIEW developer for 12 years (version 3.1)
  • Data Acquisition and Control
  • NVH
  • Vehicle bus applications (Can/J1939)

• Certifications
  • Certified LabVIEW Architect since Aug 2003
  • Certified LabVIEW Developer Oct 2002

• Education
  • Master (MSCIS)
  • Bachelor (BSCIS)
  • Associate electronics
**State Machines- Overview**

- Discuss the concept of “State Machines” as they pertain to LabVIEW based application development. Focus will be on design considerations and merits for selecting various State Machines models.

- **Topics**
  - Single loop
  - Multiple Loops (Asynchronous)
  - Supporting techniques
    - Queue
    - Functional Globals
  - Multi-threading and performance (comments)
State Machines - Definition

- Wikipedia - Model of behavior composed of finite number of states, transitions between those states, and actions. Made up of entry, exit, input, and transition actions. Used quite a bit in UML based modeling (decision trees/flowcharts).

- LabVIEW Based State Machine - Decision based execution framework. Based on a While loop used in conjunction with a Case statement and a shift register. Branch control can be enhanced by the use of Events, queues, and functional globals.

Note: Due to LabVIEW’s inherent parallelism, execution performance can be further enhanced by the use of multiple state machines running in parallel.
State Machines- Single Loop

- This is an example of a basic state machine with a while loop, case statement, and shift register. It has been enhanced slightly to include an event structure for capturing user actions.

**Uses**
- Main Application VI
- Intermediate VI
- VIs with complex execution and decision trees

**Pros**
- Easy to implement
- Very Scalable
  - Error handling
  - Input validation

**Cons**
- Additional overhead vs. pass-through
- No history
- Single exit branch (without adding extra code)
State Machines- Multiple Loops

- Multiple execution loops better leverage LabVIEW’s inherent parallelism.
- Separate loop for User IO allows the main application to continue acquiring data if program focus passes to a popup window.

**Uses**

- Primarily the main application VI

**Pros**

- Parallelism
- Leverages Multi-core CPU
- Asynchronous (potential)

**Cons**

- More complex loop interaction
- More complex data buffering
State Machines - Multiple Loops cont.

- This is an example of data acquisition, analysis, and control application running 3 asynchronous loops. The application offers better determinism, CPU utilization, and scales well with multiple CPU cores.

- Loop 1 - Acquisition, Scaling, and Display updates.
- Loop 2 - Test Control
- Loop 3 - User Interface
State Machines - Multiple Loops cont.

- Loop 1 - Acquisition, Scaling, and data logging.
- Loop 2 - Display updates
- Loop 3 - User Interface

This is an example of vehicle bus acquisition running 3 asynchronous loops. Display updates are performed in a separate loop from the data acquisition. This allows the daq loop to sample data at a faster rate than the real-time display updates.
State Machines- Queues

- Queues are located under the “Synchronization” palette and function as a stack (FIFO). A queue can enhance the function of a state machine’s shift register by queuing up multiple actions.

In this example assume the VI was acquiring data and a hardware error occurred. Depending on if the VI was logging data or simply to update displays will determine the states needed for proper shutdown. Without a queue, special states for each of these events would need to be developed, which adds to the VI's overall size and complexity.

**Pros**
- Easy to implement
- Very Scalable
- Simpler states
- Multiple exit states

**Cons**
- Small amount of extra code
State Machines - Functional Globals

- A functional global is simply a VI with a while loop and a case statement with a un-initialized shift register. Because the shit register is un-initialized it retains state as long as the VI is loaded in memory.

- These globals do not make multiple memory copies like a “global” variable and can make a very tangible impact on overall memory footprint.

- If the functional global’s execution is not set to “Reentrant” there is only one instance in memory and race conditions are avoided.

- In the context of the state machine, it can be used to exchange data between the loops.

Note: The biggest negative is overuse of FGs goes against the concept of dataflow.
State Machines- The Global Queue

• In the earlier Daq/Analysis/Control application each loop uses a queue. To better support inter-loop control, the queues themselves were placed in a functional global. This simplified interaction with the various queues.
State Machines- Conclusion

- This presentation is by no means a comprehensive discussion of the concept of state machine, but instead tries to highlight the opportunities offered by LabVIEW to develop powerful application which are not only multi-threaded, but harness multi-core. LabVIEW has served me well in developing stable and fast applications over the years, and hopefully it is proving just as useful to all of you.