Concept: Comparing Queues With Local Variables

Goal

In this exercise, you run and examine a prebuilt producer/consumer design pattern VI that transfers data generated by the producer loop to each of the consumer loops using local variables and queues.

Description

The files that you need to complete this exercise are here:
`<NI eLearning>\LV Core 2\Queues\Exercise`.

1. Open `Queues vs Local Variables.vi` from the `<Exercise>` directory. The front panel of this VI is shown in Figure 1.

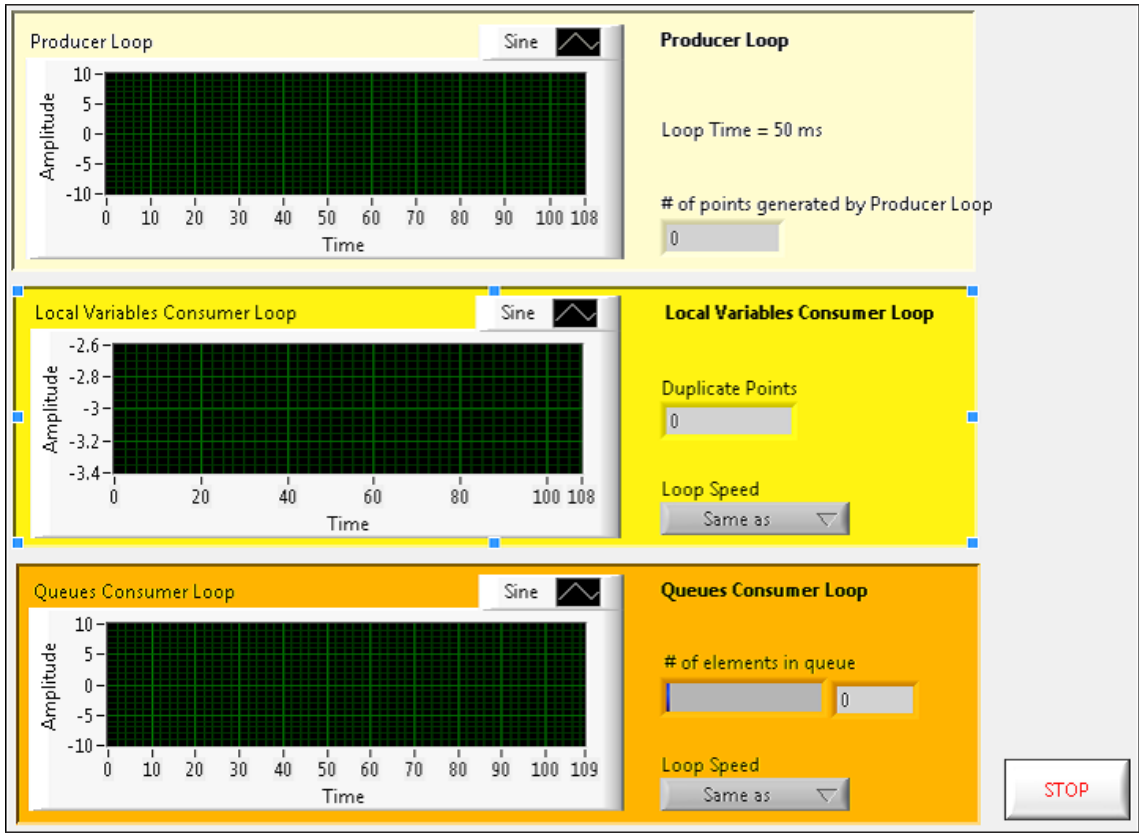

Figure 1. Front Panel of the Queues vs Local Variables VI

2. Run the VI. The producer loop generates data and transfers it to each consumer loop using a local variable and a queue.

3. Display and examine the block diagram for this VI.

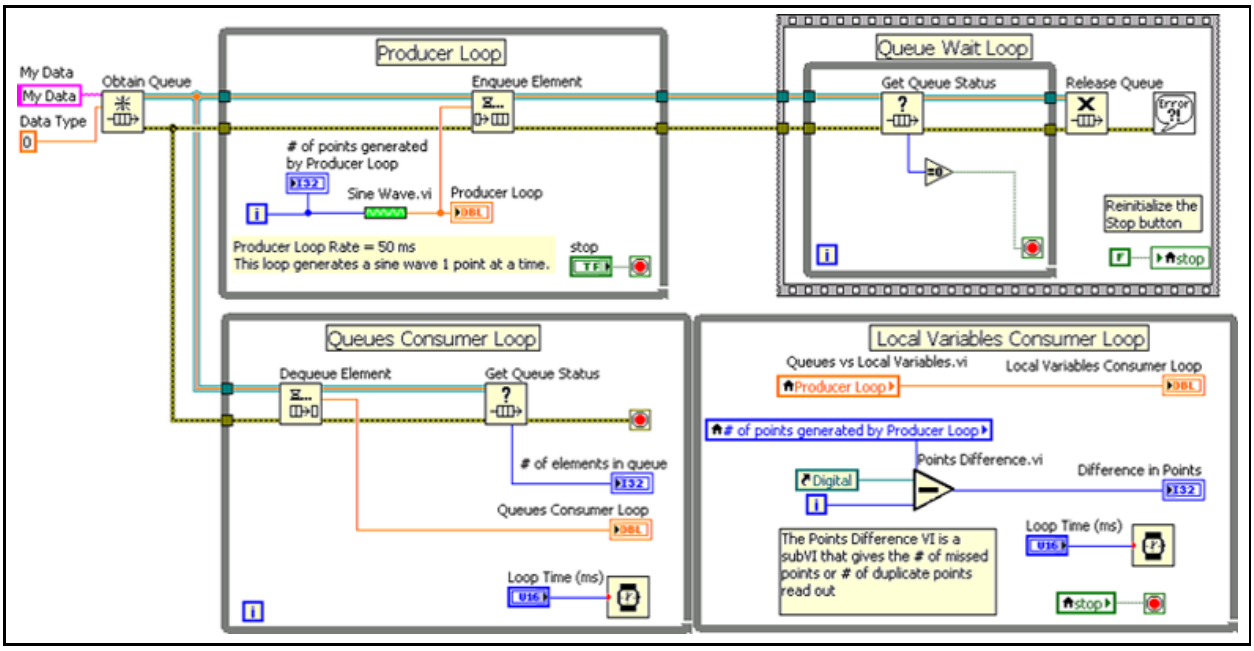

Figure 2. Block Diagram of the Queues vs Local Variables VI

Creating a Queue

The Obtain Queue function, placed to the left of the Producer Loop, creates the queue.

The My Data string constant, wired to the name (unnamed) input of the Obtain Queue function, assigns a name to the queue you want to obtain or create.

The Data Type numeric constant, wired to the element data type input of the Obtain Queue function, specifies the type of data that you want the queue to contain.

Queuing Data Generated by the Producer Loop

The Enqueue Element function inside the Producer Loop adds each data element generated by the Sine Wave subVI to the back of the queue.

Dequeuing Data from the Producer Loop inside the Queue's Consumer Loop

The Dequeue Element function inside the Queue's Consumer Loop removes an element from the front of the queue and outputs the data element to the Queue's Consumer Loop waveform graph.

The Get Queue Status function inside the Queue's Consumer Loop indicates how many elements remain in the queue. In order to process these data elements, you must execute the Queue's Consumer Loop faster than the Producer Loop, or continue to process after the Producer Loop has stopped.

Waiting for the Queue to Empty

The While Loop inside the Flat Sequence structure waits for the queue to empty before stopping the VI. Refer to this While Loop as the Queue Wait Loop.

The Get Queue Status function inside the Queue Wait Loop returns information about the current state of the queue, such as the number of data elements currently in the queue.

The Equal To 0? function wired to the stop condition of the Queue Wait Loop checks if the queue is empty.

The Release Queue function to the right of the Queue Wait Loop releases and clears reference to the queue.

The Simple Error Handler to the right of the Release Queue function reports any error at the end of execution.

Local Variable's Consumer Loop

The Producer Loop generates sine wave data and writes it to a local variable while the Local Variable's Consumer Loop periodically reads out the sine wave data from the same local variable. The Points Difference VI inside the Local Variable's Consumer Loop outputs the number of missed points or number of duplicate points read out.

Switch to the front panel of this VI.

1. Select the loop time speed of the Local Variable's Consumer Loop and observe the Local Variable's Consumer Loop waveform graph and the results generated on the Duplicate Points indicator.

   - ❑ Ensure that the Loop Speed selected is **Same as Producer Loop** and observe the waveform graphs for both the Producer Loop and the Local Variable's Consumer Loop. A race condition may occur resulting in missed points or duplicated data.

   - ❑ Select **Maximum Speed** from the pull-down menu of the **Loop Speed** control and observe the waveform graph of the Local Variable's Consumer Loop. A race condition occurs because data is consumed faster than it is produced, allowing the local variable to read the same value multiple times.

- ❑ Select **1/2 as Producer** from the pull-down menu of the **Loop Speed** control and observe the waveform graph of the Local Variable's Consumer Loop. A race condition occurs because data is produced faster than it is consumed. The data changes before the local variable has a chance to read it.

- ❑ Select the remaining options available from the pull-down menu of the **Loop Speed** control and observe the data retrieval.

2. Stop the VI.

   Data transfer between two non-synchronized parallel loops using local variables causes a race condition. This occurs when the Producer Loop is writing a value to a local variable while the Local Variable's Consumer Loop is periodically reading out the value from the same local variable. Because the parallel loops are not synchronized, the value can be written before it has actually been read or vice-versa resulting in data starvation or data overflow.

Queue's Consumer Loop

1. Run the VI. Select the loop time speed of the Queue's Consumer Loop and observe the Queue's Consumer Loop waveform graph and the results generated on the # of elements in queue indicator.

   - ❑ Ensure that the **Loop Speed** selected is **Same as Producer** and observe the value of the # of elements in queue indicator. The value should remain zero. Hence with queues, you will not lose data when the producer and consumer loops are executing at the same rate.

   - ❑ Select **Maximum Speed** from the pull-down menu of the **Loop Speed** control and observe the value of # of elements in queue. The value should remain zero. Hence with queues, you will not lose data if the consumer loop is executing much faster than the producer loop.

   - ❑ Select **1/2 as Producer** from the pull-down menu of the **Loop Speed** control and observe the value of # of elements in queue. The data points will accumulate in the queue. You will need to process the accumulated elements in the queue before reaching the maximum size of the queue to avoid data loss.

   - ❑ Select the remaining options available from the pull-down menu of the **Loop Speed** control and observe the synchronization of data transfer between the producer loop and the consumer loop using queues.

2. Stop the VI.

   When the Producer Loop and Queue's Consumer Loop run at the same speed, the number of elements in the queue remains unchanged. When the Queue's Consumer Loop runs slower, the queue quickly backs up and the Producer Loop must wait for the Queue Consumer Loop to remove the elements. When the Queue's Consumer Loop runs faster, the queue is quickly emptied and the consumer loop must wait for the Producer loop to insert elements. Hence queues synchronize the data transfer between the two independent parallel loops and thus avoid loss or duplication of data.

3. Close the VI. Do not save changes.

End of Exercise

Notes