Parallel Threads Tutorial

Due to the nature of executing programs, we need to take into account the existence and potential hazards of parallel threads. For most simple program executions, most threads are executed virtually instantaneously, however, there may be some instances where the behavior of parallel threads may introduce unforeseen timing errors.

Parallel threads are concurrent processes that execute at the same time within the same frame (such as a For Loop frame for instance, each iteration can be considered a new frame). Each parallel thread may also contain several parallel sub-threads in a sub-frame of execution, which also need to be taken into account during execution. Because LabView requires every parallel thread within a frame to complete execution before moving on to subsequent executions, parallel threads are essential to the dynamics of timing.

Parallel threads can both be dependent or independent of other parallel threads. For instance, we may need to wait for a result from another parallel thread before we can compute a value in the current thread.

For example, let's consider two independent threads A and B which take 1000 milliseconds and 3000 milliseconds to execute respectively as shown in Figure 1. Since these two threads are run in parallel, we only have to wait for the slowest thread to finish, which is thread B.

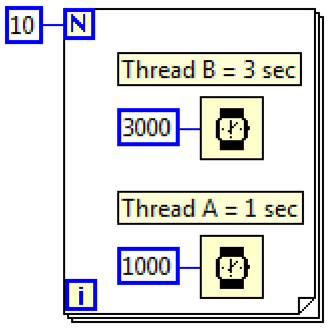

Figure 1

Notice that the total time in this case is 3000 milliseconds not 4000 milliseconds.

Now let's consider dependent threads in the example shown in Figure 2. The run time of this program is on the order of milliseconds; let's consider how these threads interact.

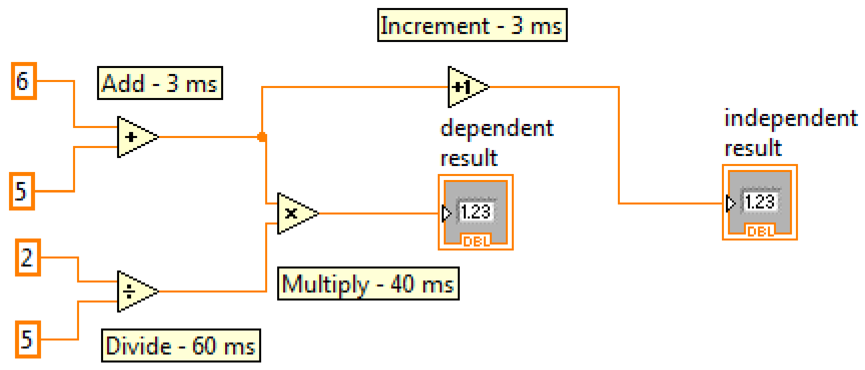

Figure 2

*Note: These operations do not actually take this long to execute in LabView

We will notice that if we highlight execution, that the data from the Add function propagates to both the Increment function and the Multiply function.

If we consider the execution times of each of the functions given in Figure 2 and the way the data flows, we notice that the Multiply function receives a value from the Add function after 3 milliseconds, but it must wait for the result from the Divide function, which executes on a parallel thread before it can proceed with the Multiply operation. Therefore, we say that one thread exhibits as dependency on the other thread and the execution time for the dependent result is 60 ms + 40 ms = 100 ms.

We also notice in Figure 2 that the Increment function which execute in parallel with the Multiply function only has one input and executes immediately after the input is available from the add function, which effectively makes the execution time for the independent result 3 ms + 3 ms = 6 ms.

Parallel threads are often an important factor when considering Timing functions and may cause hazards during execution. For example, consider the following block diagram which is intended to time 10 seconds using the Wait function but also contains another parallel thread that iterates from 1 to 10000 in a While loop during every execution of the For loop. We will notice that this implementation will fail if the time it takes for the parallel thread to complete is more than 1000 milliseconds (the time it takes for the Wait function to complete execution).

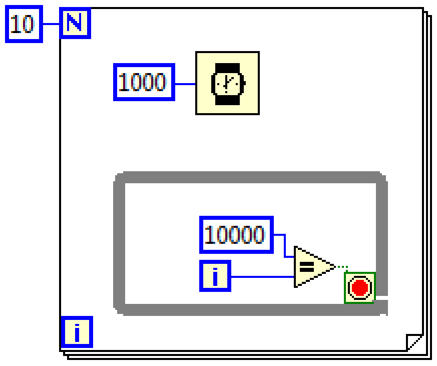

Figure 3

In order to avoid this pitfall, we may consider using other timing functions such as the Metronome function that will allow a more reliable mechanism of timing or clocking program execution.