

# CASE STRUCTURE AND SEQUENCE STRUCTURE



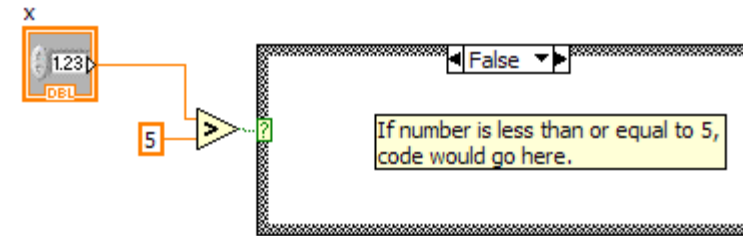
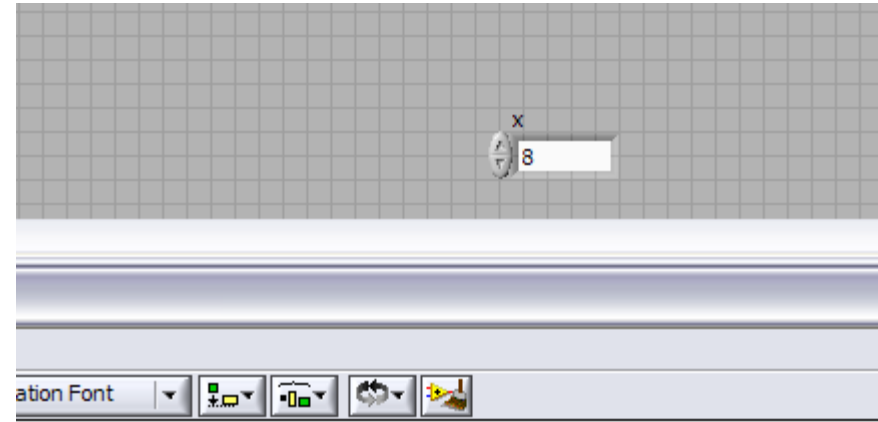
# CASE STRUCTURE

- Case structures are used when you want a particular case to implement a piece of code. This is similar to an IF-ELSE statement.

# CASE STRUCTURE EXAMPLE #1

Here is a simple example of a case structure.

If a user selected number is  $>5$ , then the case statement would implement the “True” case. Else, implements the “False Case”

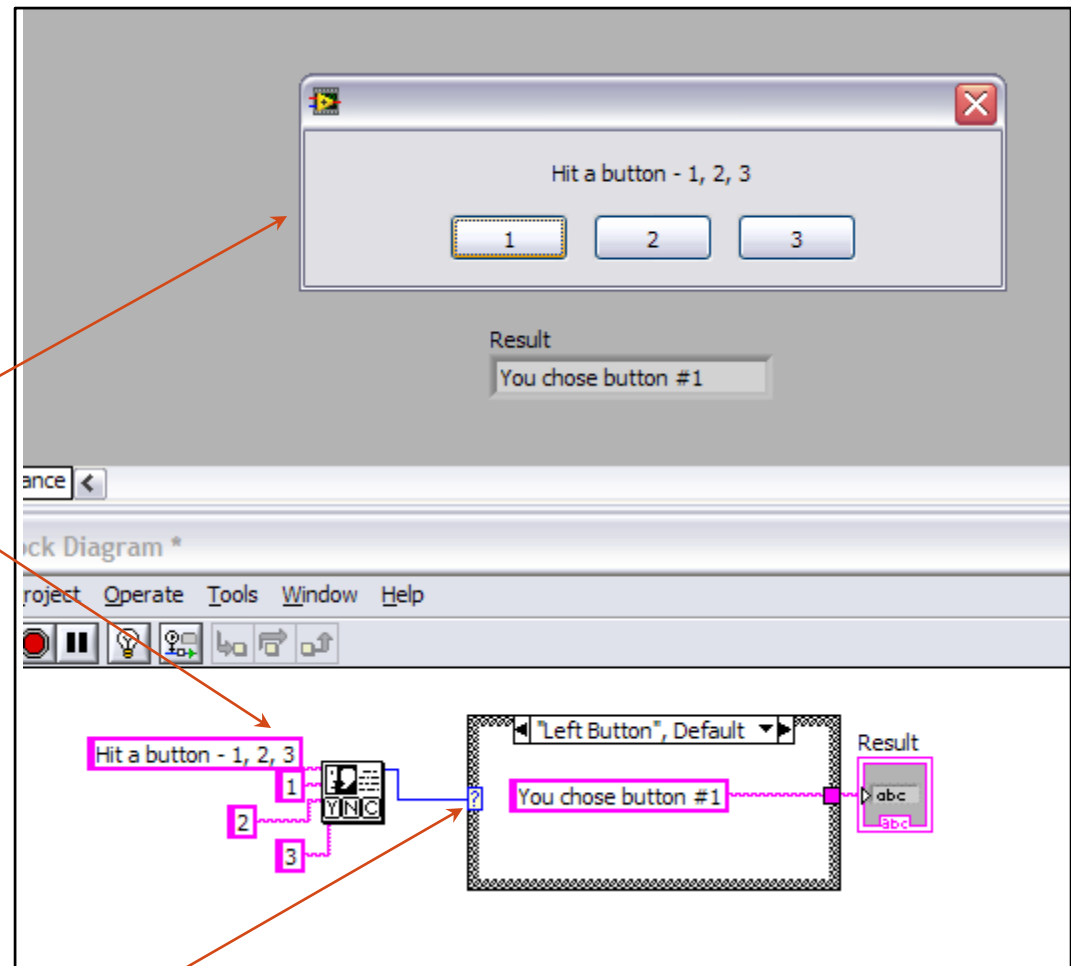


# CASE STRUCTURE EXAMPLE #2

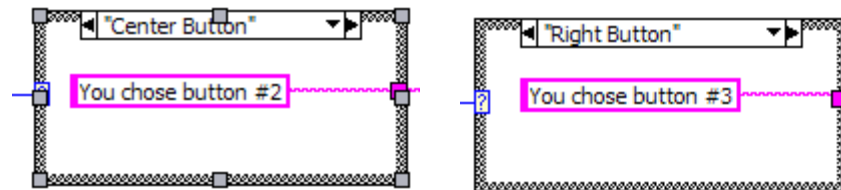
In this example, I used the “3 Button Dialog” vi and named the 3 buttons “1”, “2”, and “3”. If the user chooses the left button, then it will display in Result – “You chose button #1.”

The other two cases say “You chose button #2” and “You chose button #3.”

By the way, when you wire up the “?” on the case structure, it will automatically make all the cases for you. If it doesn’t, then you can right click on the title and choose “Show case structure for every case.”



Here are the other cases.



# CASE STRUCTURE

CLAD note: Strings inputted into Case Structures are case-sensitive.

# CLAD QUESTION

Which data type is not accepted by the case selector terminal on a case structure?

- a. Arrays
- b. Enumerated type values
- c. Strings
- d. Integers

# CLAD QUESTION

Which data type is not accepted by the case selector terminal on a case structure?

- ☒ a. Arrays
- b. Enumerated type values
- c. Strings
- d. Integers

# CLAD QUESTION

You can use all of the following data types as inputs to the case selector terminal except:

- a. Doubles
- b. Enumerated type values
- c. Strings
- d. Integers



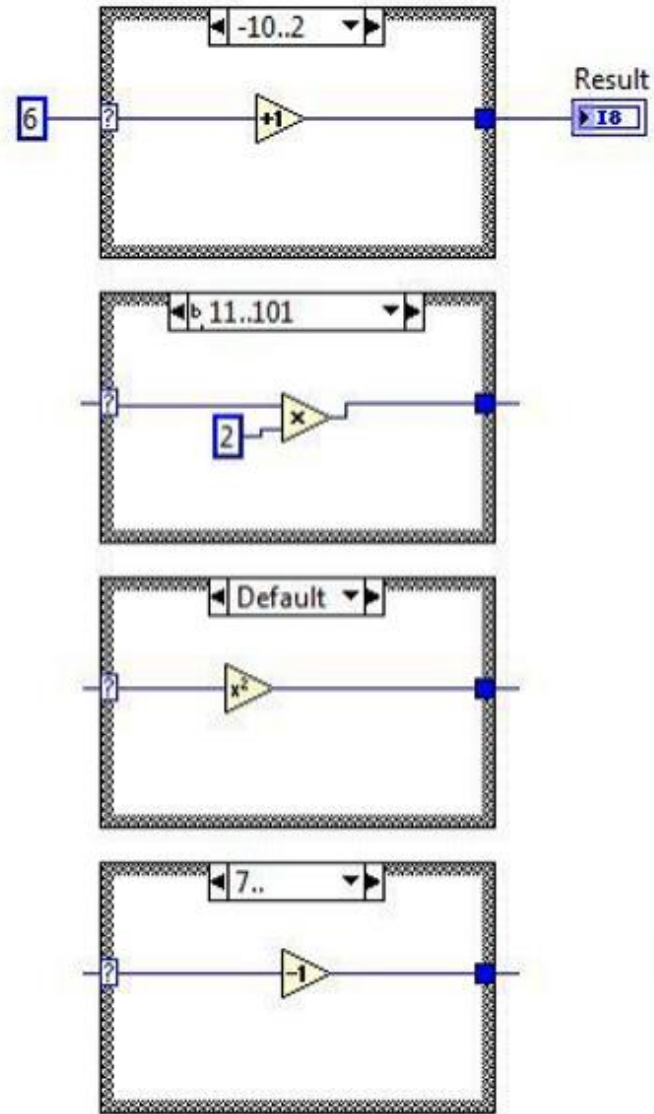
# CLAD QUESTION

You can use all of the following data types as inputs to the case selector terminal except:

- a. Doubles
- b. Enumerated type values
- c. Strings
- d. Integers

# CLAD QUESTION

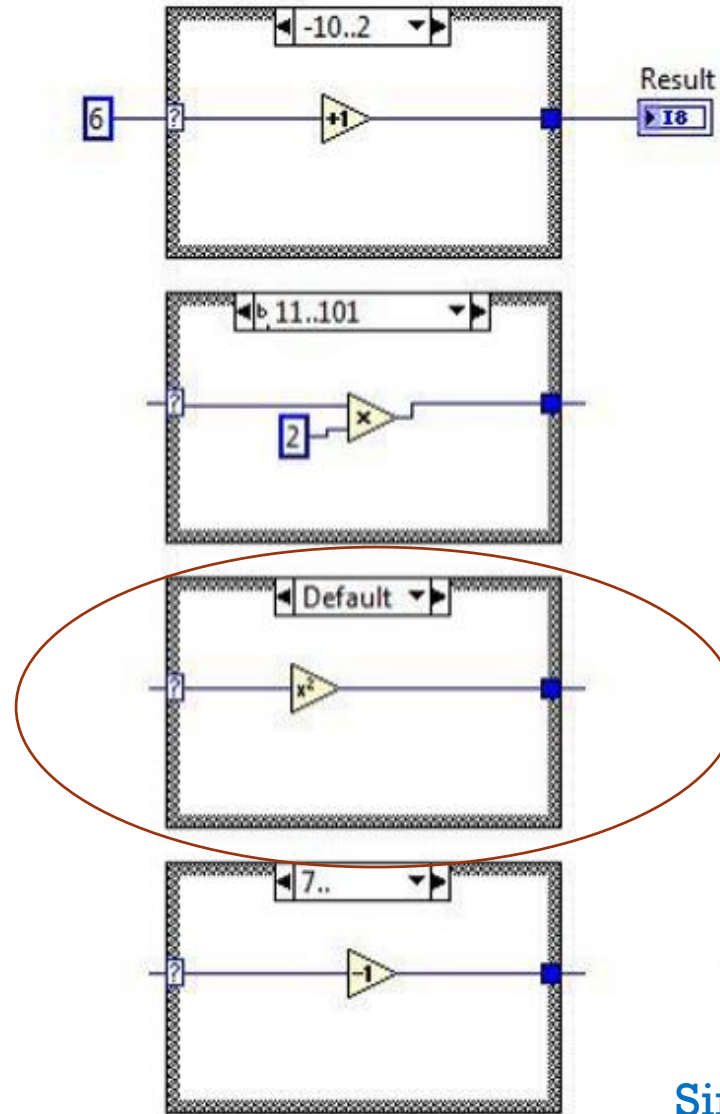
28. This graphic displays all the cases of a single case statement. What value does the **Result** indicator display after the VI executes?



- a. 5
- b. 7
- c. 12
- d. 36

# CLAD QUESTION

28. This graphic displays all the cases of a single case statement. What value does the **Result** indicator display after the VI executes?



- a. 5
- b. 7
- c. 12
- d. 36

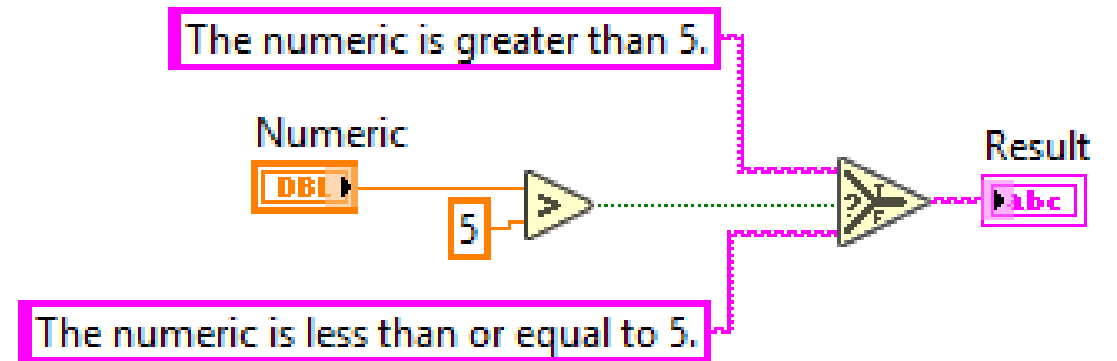
Since there's not a specific case for 6, it uses the default case.

# SELECT VI

If you have a scenario where you have a T/F (Boolean) feeding into a Case structure, then sometimes it's better to just use the "Select" VI since it takes up less real estate on your block diagram.

The Select VI is found in the Comparison palette.

It looks like ....

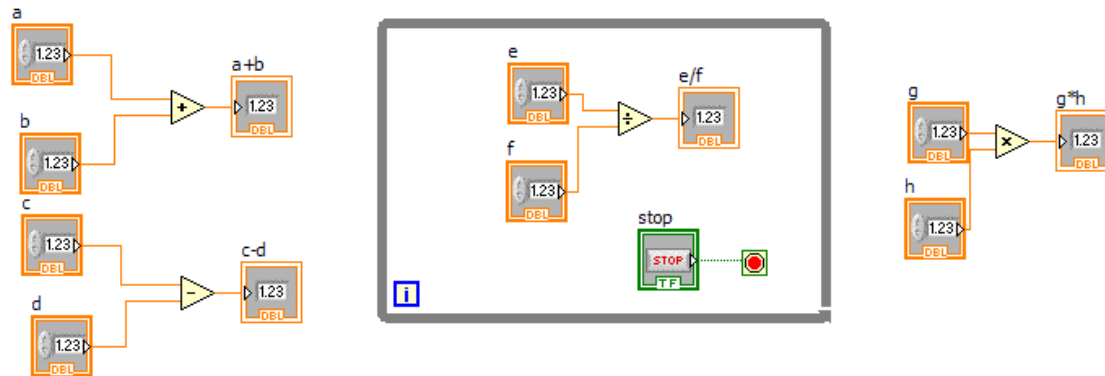
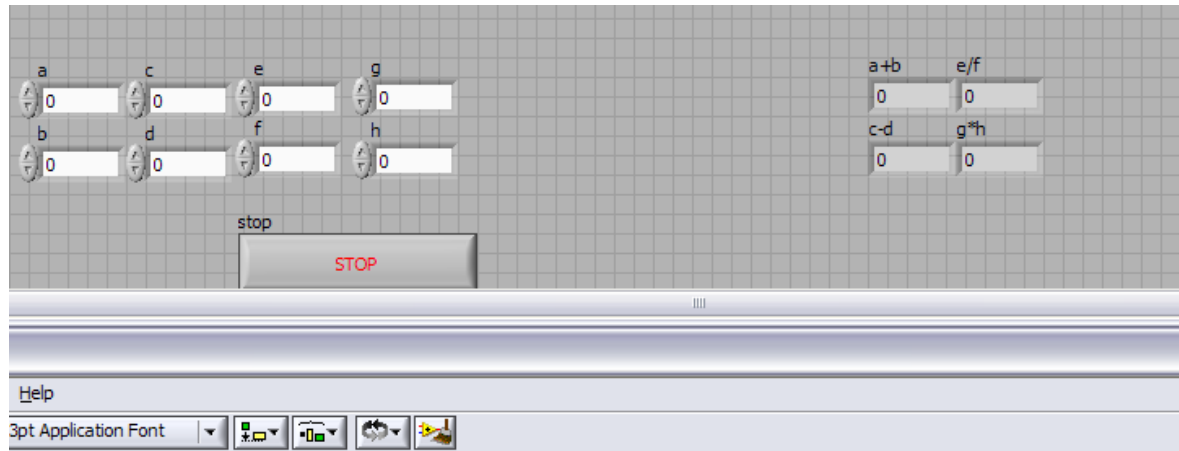


The Select VI must take a Boolean for the control, but the values fed into it can be lots of things including strings, integers, doubles, etc.

# CONCURRENCY

- Before I go into the Flat and Stacked Sequence structures, I want to talk about concurrency.

# CONCURRENCY



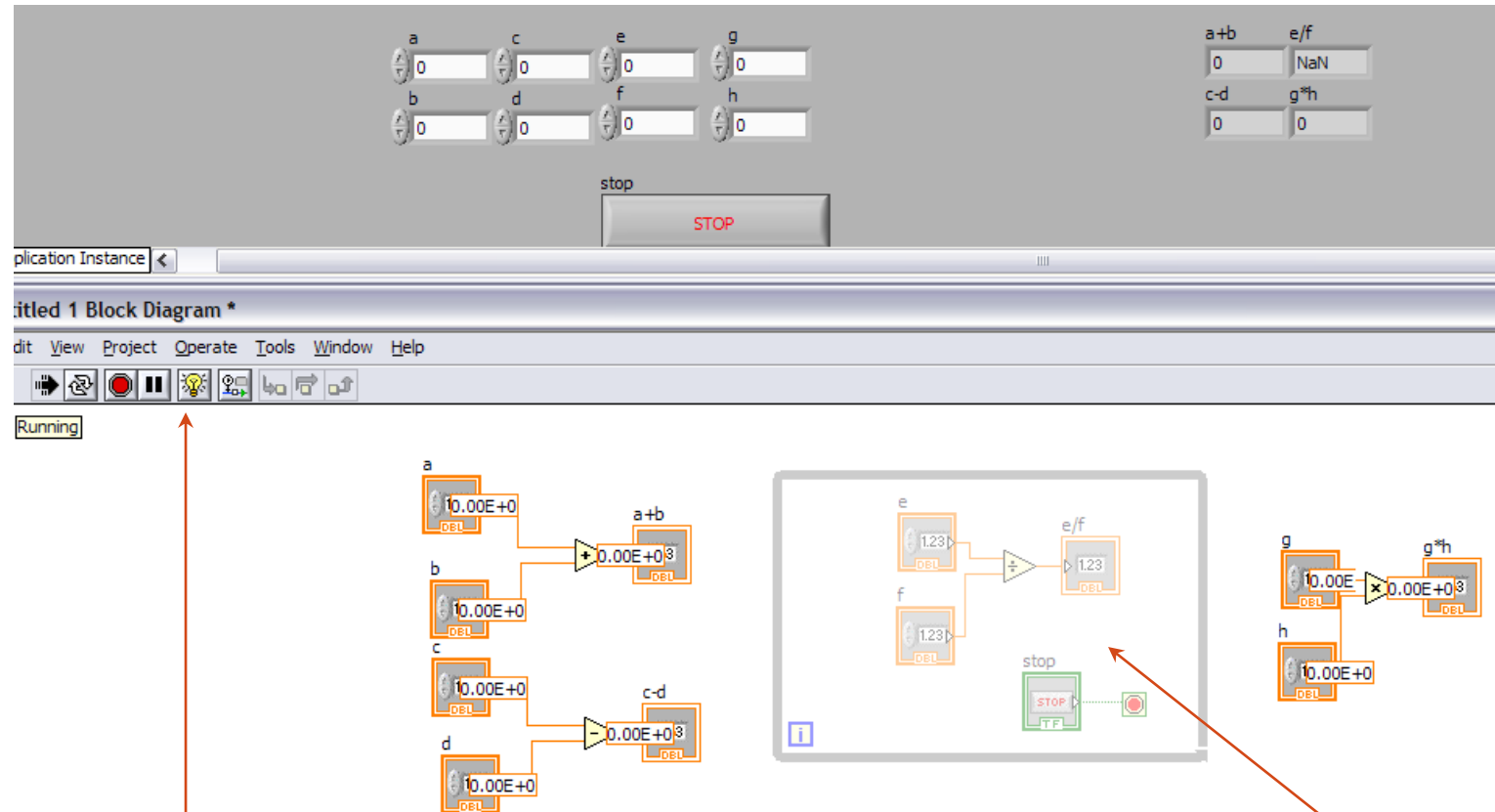
In this example, the addition, subtraction, and multiplication happen at the same time! It doesn't matter that the multiplication is after the while loop!

Then the division happens.

So code inside loops happen after everything outside of it is implemented!

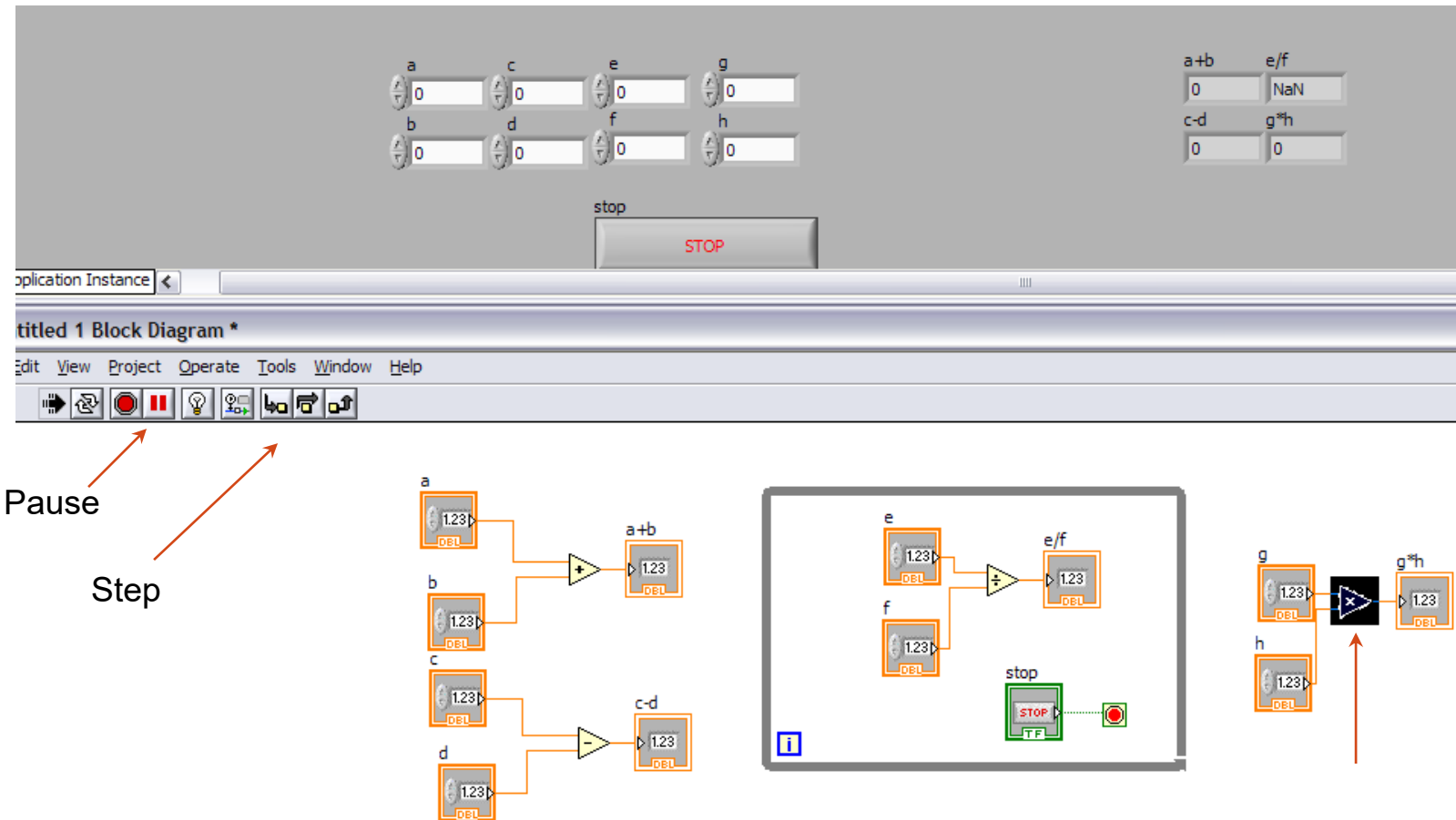
The +, -, \* happen only once, while the division happens repeatedly until the user hits the stop button.

# CONCURRENCY AND USING THE LIGHT BULB



You can see the order of how things are implemented by clicking on the light bulb. This is a great tool for troubleshooting! Items that are grayed out have not yet been implemented. Notice that the while loop has not been implemented yet. The light bulb is a great tool to find where your code might be stuck in an infinite loop.

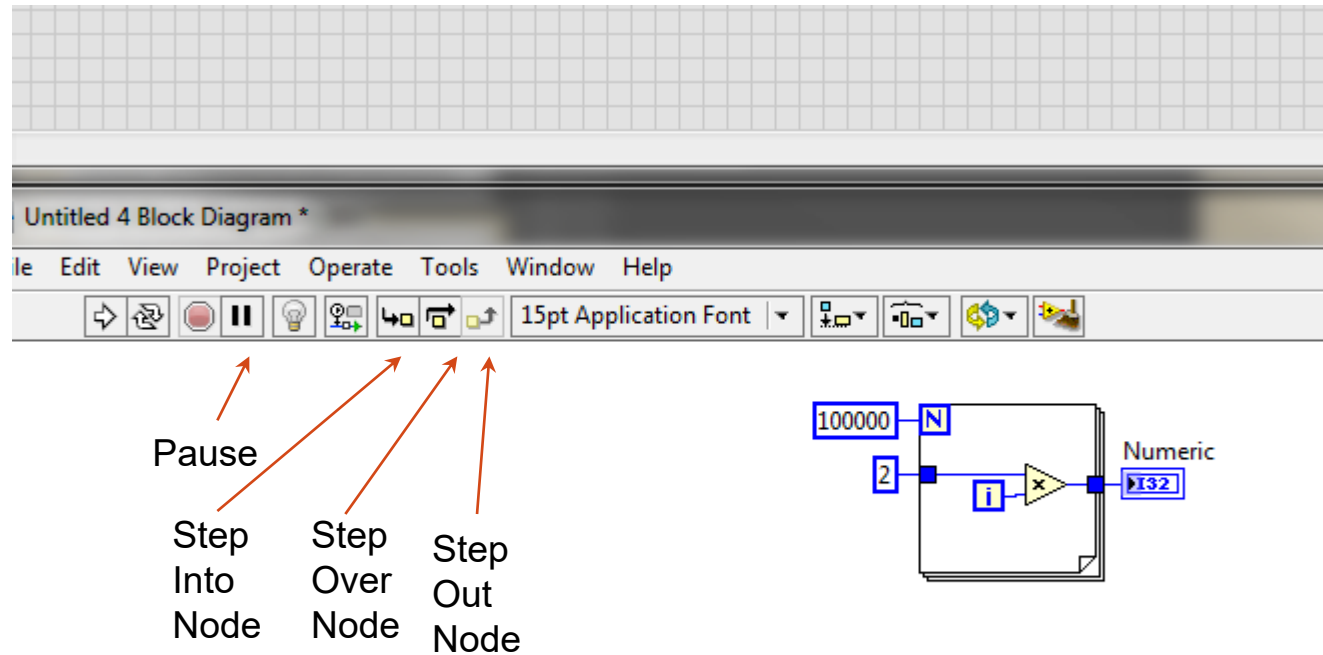
# CONCURRENCY AND USING THE PAUSE BUTTON



If you want to step through your code, you can use the pause button and the step buttons. You will see that the multiplication, addition, and subtraction actually do not happen concurrently. They actually happen sequentially, but this happens so fast that it will seem like it happens concurrently (for all intensive purposes this happens concurrently). If you notice, the first thing that happens is the multiply (it's highlighted). Stacked sequences and flat sequences will allow control over this.



# DEBUGGING: USING THE PAUSE BUTTON



Step Into Node allows single stepping through all 100,000 iterations of the For loop.

Step Over allows the user to implement all the code in a structure (For loop in this case) without having to go into that structure. Once you're in that structure, then Step Over just single steps like the Step Into. Have to go through all 100,000 iterations. Also works to step over SubVIs (*we'll discuss subVIs at a later date*).

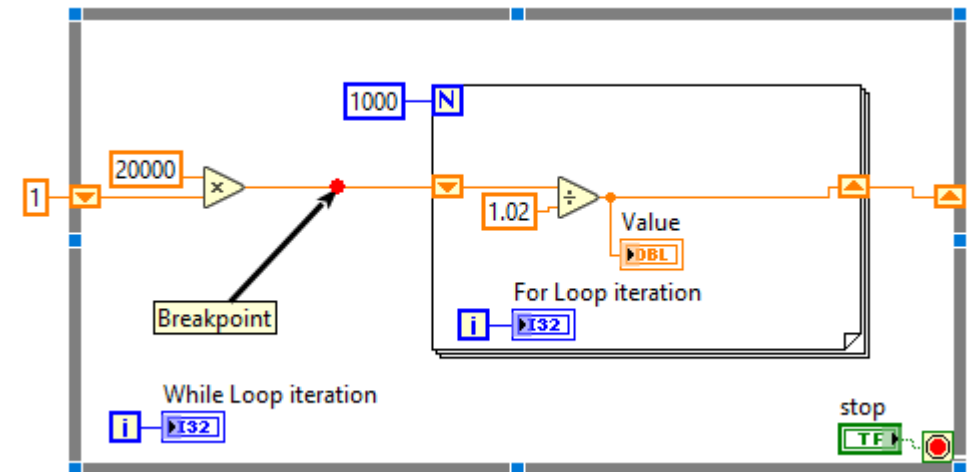
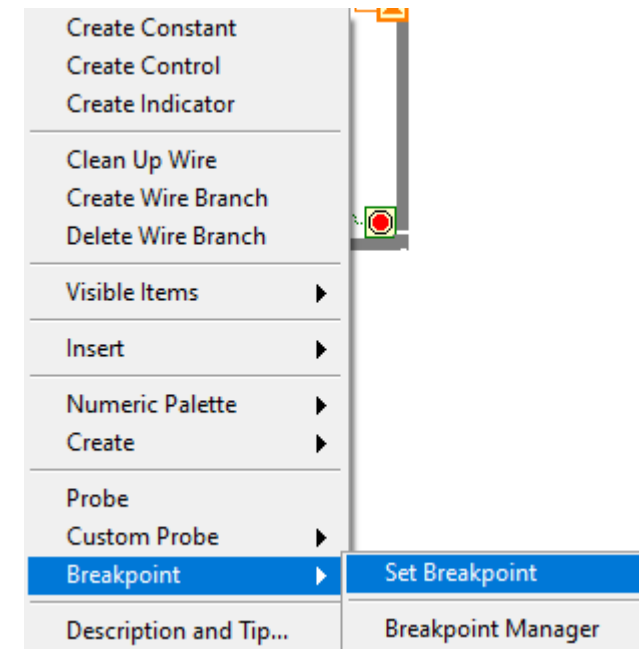
Step Out Node allows you to get out of the structure (For loop) without going through 100,000 iterations.

# DEBUGGING: USING BREAKPOINTS

Breakpoints (similar in other languages) causes the program to pause. Breakpoints can be created by right-clicking on a wire and selecting Breakpoint > Set Breakpoint.

In the program to the right, if we wanted to see what the indicator “Value” is on the 20th For loop iteration of the 3rd While loop iteration, then the fastest way to tell this is to add a breakpoint as shown. The program will pause at the breakpoint and we would select “Step Over” so we don’t have to go through 1000 iterations of the For loop. Then we would continue to hit “Step Over” until the While loop iteration is at i=2. At this point, we would choose “Step Into” until the For loop iteration is at i=19.

In around 30 seconds, we can figure out what this value is.



# CLAD QUESTION

Clicking on the \_\_\_\_\_ button allows you to bypass a node in the Block Diagram without single-stepping through the node.

- a. Step Into
- b. Step Over
- c. Step Out
- d. Step Through

# CLAD QUESTION

Clicking on the \_\_\_\_\_ button allows you to bypass a node in the Block Diagram without single-stepping through the node.

- a. Step Into
- b. Step Over
- c. Step Out
- d. Step Through

# CLAD QUESTION

**Q24:** What is the purpose of these block diagram toolbar buttons?



- A** To display the data flowing through wires as the VI runs
- B** To step through the block diagram when execution is paused
- C** To start or skip SubVIs
- D** To run the VI in continuous or single run mode.

# CLAD QUESTION





**Q24:** What is the purpose of these block diagram toolbar buttons?



- A To display the data flowing through wires as the VI runs
- B To step through the block diagram when execution is paused**
- C To start or skip SubVIs
- D To run the VI in continuous or single run mode.





# CLAD QUESTION

**Q25:** When using single step debugging with a SubVI which of the following is not possible?

- A** Step Into () while the execution flow is paused on the SubVI icon
- B** Step Out () while the execution flow is paused on a node inside the block diagram of the SubVI.
- C** Finish VI () while the execution flow is paused on the block diagram of the SubVI
- D** Finish Block Diagram () while the execution flow is paused on a node inside the SubVI

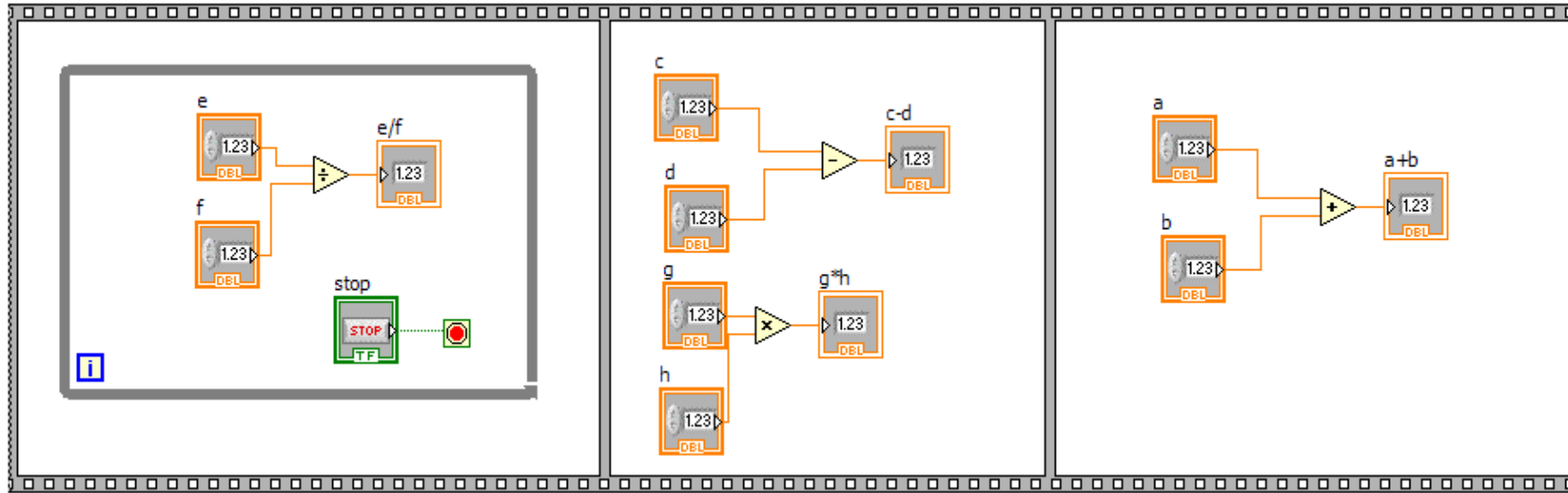
# CLAD QUESTION

**Q25:** When using single step debugging with a SubVI which of the following is not possible?

- A** Step Into () while the execution flow is paused on the SubVI icon
- B** Step Out () while the execution flow is paused on a node inside the block diagram of the SubVI.
- C** Finish VI () while the execution flow is paused on the block diagram of the SubVI
- D** Finish Block Diagram () while the execution flow is paused on a node inside the SubVI



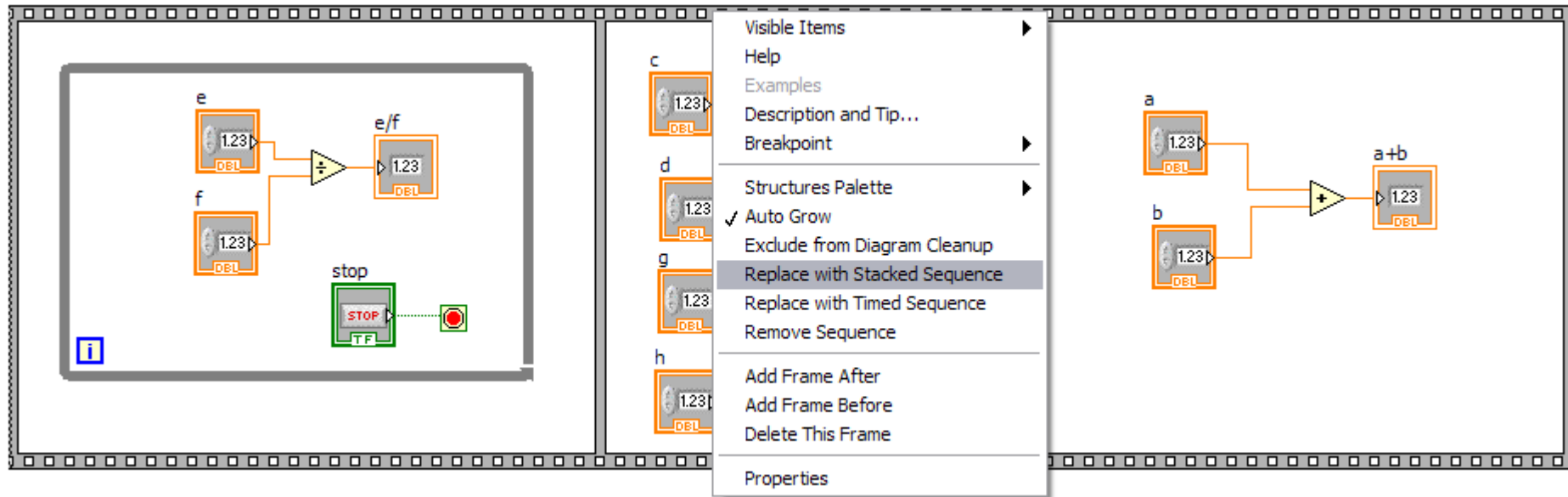
# FLAT SEQUENCE (CH.6, P. 216)



You can force code to happen in a particular sequence by using a flat sequence. A flat sequence is supposed to mimic a film strip.

In this example, we forced the while loop to happen first, then subtraction and multiplication, and finally addition. The second frame will not be implemented until the while loop finishes (it will finish as soon as the user hits the stop button).

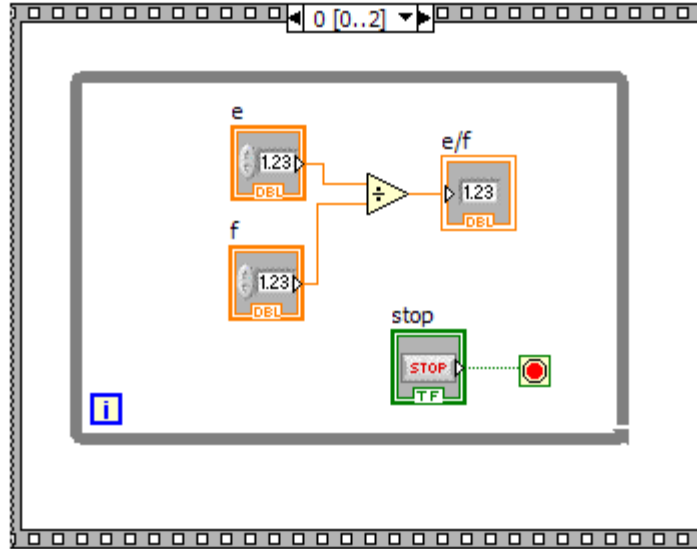
# CONVERTING FLAT SEQUENCE TO A STACKED SEQUENCE



If you right-click on the flat sequence, an option occurs for you to replace a stacked sequence from a flat sequence.

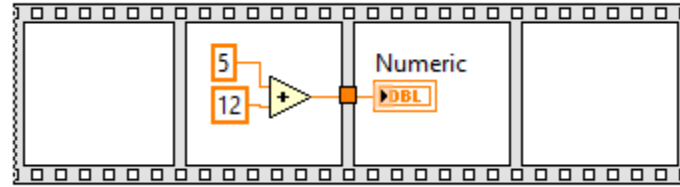
You can also just start with a stacked sequence.

# STACKED SEQUENCE

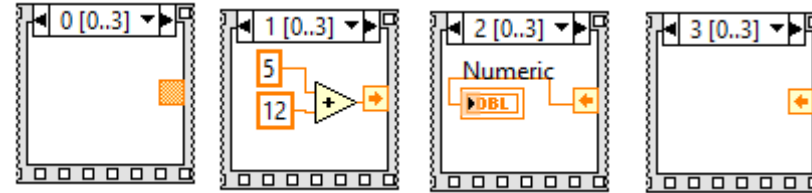


Here is the same flat sequence converted to a stacked sequence. Notice at the top it says 0 [0..2]. The first 0 means we are in frame #0 (which will happen first) and the [0..2] means that there exists frames 0 through 2 (3 frames total).

# PASSING VALUES BETWEEN FRAMES OF A STACKED SEQUENCE



This flat sequence converts to this stacked sequence.

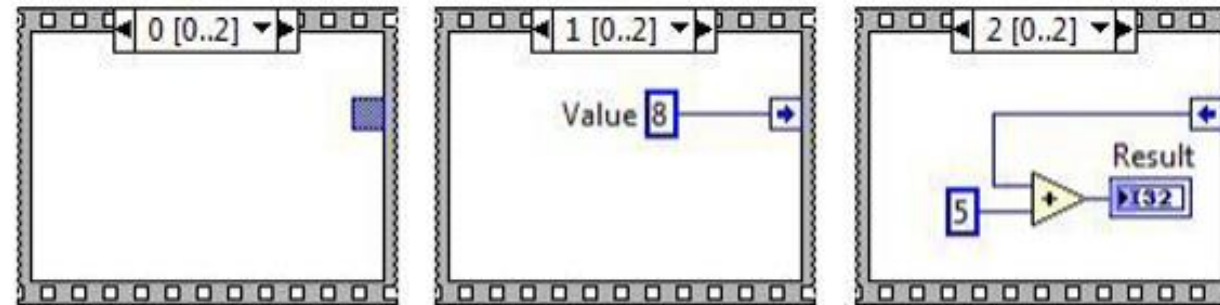


Notice that frame 0 has an orange square. This means that there exists a node, but frame 0 doesn't have access to it (because it doesn't get written to until frame 1). Notice that frame 1 writes into the node and frame 2 reads from this node. Another interesting thing is that frame 3 (or any frame afterwards) has access to this node.

So the stacked sequence is a little bit different than the flat sequence.

# CLAD QUESTION

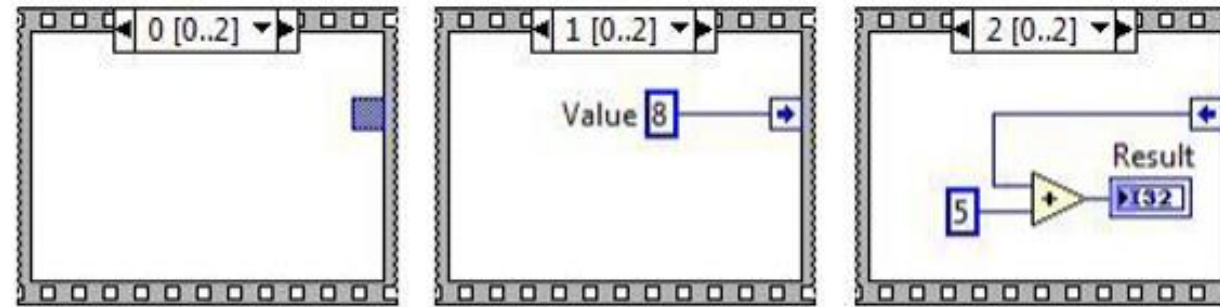
Why is the sequence local terminal displayed as unassigned in Frame 0 of the stacked sequence structure?



- a. The developer chose not to wire the value to any terminal in this frame
- b. The value is available only to frames after frame 1
- c. The data type of the terminal is incompatible with the data type of Value
- d. The developer disabled the terminal

# CLAD QUESTION

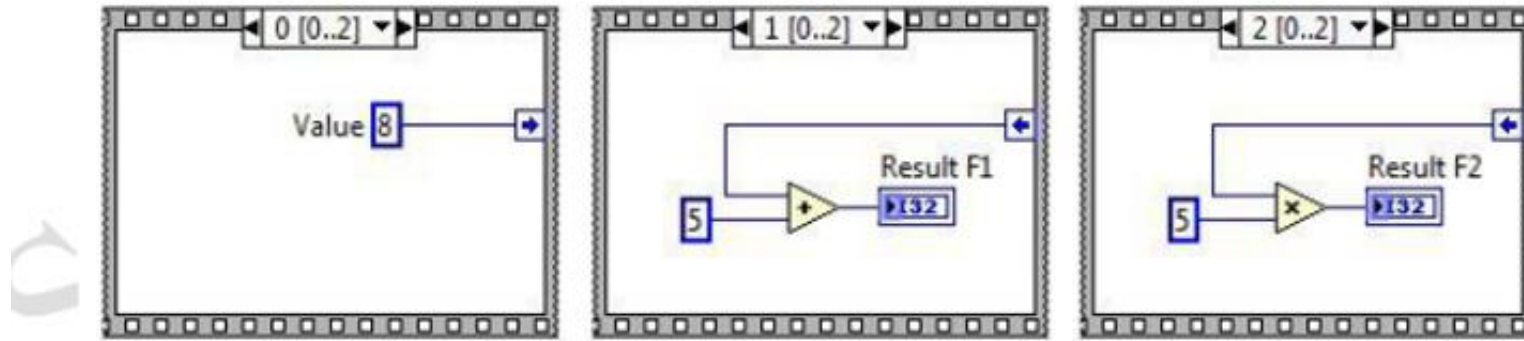
Why is the sequence local terminal displayed as unassigned in Frame 0 of the stacked sequence structure?



- a. The developer chose not to wire the value to any terminal in this frame
- ☒ b. The value is available only to frames after frame 1
- c. The data type of the terminal is incompatible with the data type of Value
- d. The developer disabled the terminal

# CLAD QUESTION

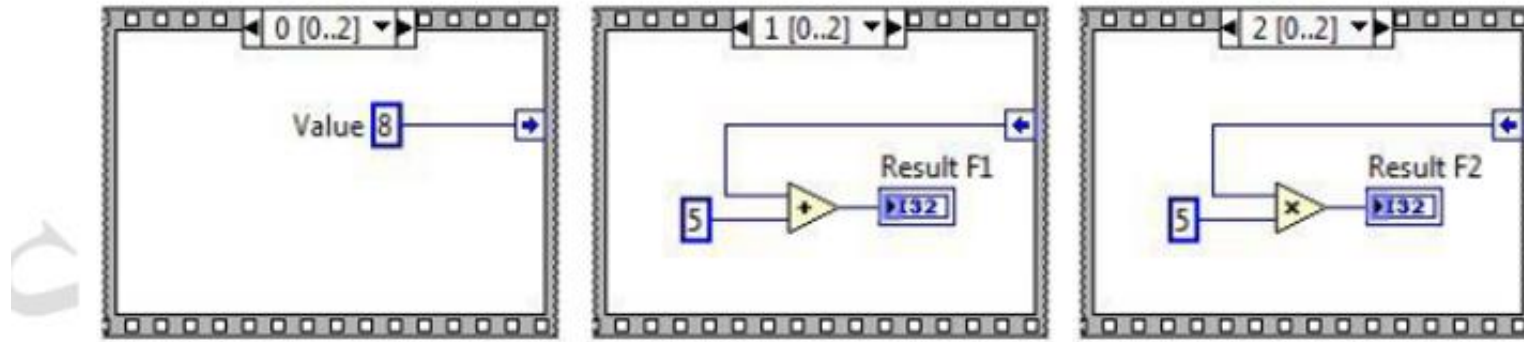
What value does the **Result F2** indicator display after the VI containing this Stacked Sequence structure executes?



- a. 0
- b. 25
- c. 40
- d. 65

# CLAD QUESTION

What value does the **Result F2** indicator display after the VI containing this Stacked Sequence structure executes?



- a. 0
- b. 25
- c. 40
- d. 65

8 is fed into the node in frame 0. Frame 1 doesn't change this value so the node is still equal to 8 in frame 2.  
 $8 \times 5 = 40$ .



# CLAD QUESTION

**Q25:** Which statement about sequence structures is FALSE?

- A**      Sequence structures support parallel operations within frames.
- B**      Terminating the execution of a sequence structure before the entire sequence is completed is not possible without aborting.
- C**      Sequence structures execute frames in a sequential order.
- D**      Sequence structures stop when an error is detected.

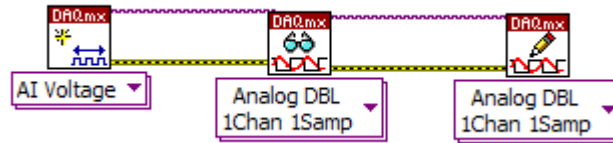
# CLAD QUESTION

**Q25:** Which statement about sequence structures is FALSE?

- A**      Sequence structures support parallel operations within frames.
- B**      Terminating the execution of a sequence structure before the entire sequence is completed is not possible without aborting.
- C**      Sequence structures execute frames in a sequential order.
- D**      Sequence structures stop when an error is detected.

Sequence Structures do not have conditionals, so there's no way to stop a sequence structure once it starts.

# ANOTHER WAY TO PERFORM THINGS SEQUENTIALLY

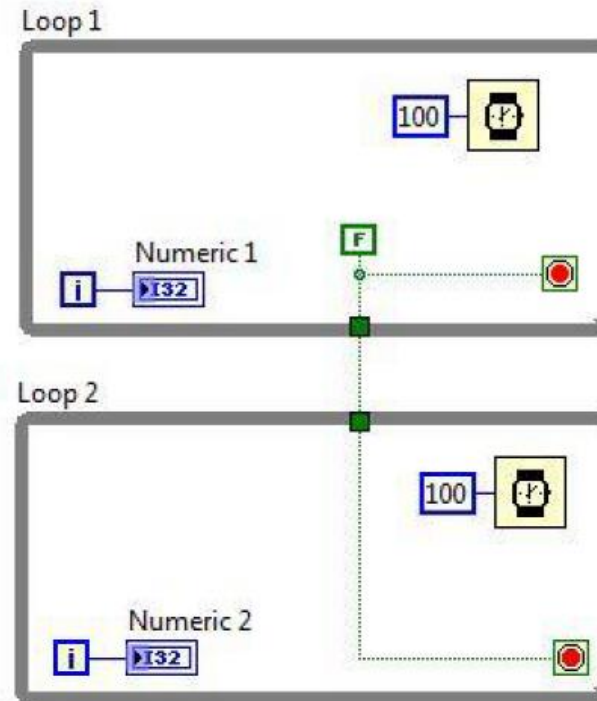


Another way to implement items sequentially is by connecting the wires from one icon to the next icon. So items on this example happen from left to right. If these items weren't wired together, then they would happen at the same time.

Don't worry about the function of these items for now, we'll get to them later. 😊

# CLAD QUESTION

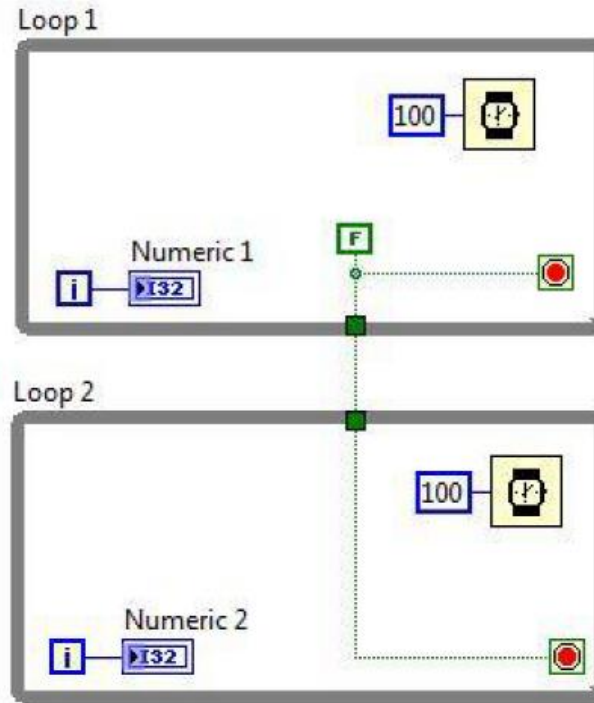
**Q22:** What is the behavior when the code executes?



- A** Loop 1 and Loop 2 run simultaneously
- B** Both loops run one time and stop
- C** Loop 2 runs after Loop 1 stops
- D** Loop 1 runs forever and Loop 2 never runs

# CLAD QUESTION

Q22: What is the behavior when the code executes?



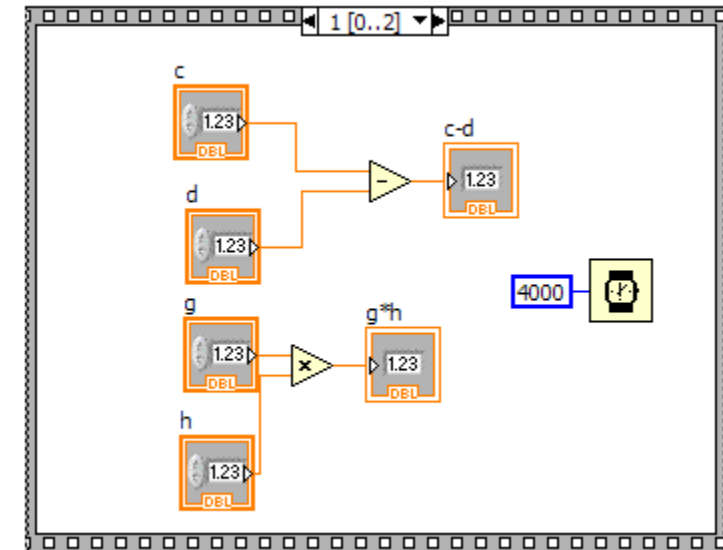
Loop 2 never runs because it's waiting on all of the inputs to be ready before it begins. Since Loop 1 runs forever, Loop 2 never receives the input from Loop 1 since Loop 1 runs indefinitely.

- A Loop 1 and Loop 2 run simultaneously
- B Both loops run one time and stop
- C Loop 2 runs after Loop 1 stops
- D Loop 1 runs forever and Loop 2 never runs

# TIMING (CH.6, P. 220)

You can also have a particular frame (or iteration of a while loop or for loop) implement for a particular amount of time. I added a 4000 ms = 4 sec wait to frame #1. Normally this frame would take about 1 ms to implement and now I forced it to take 4 sec to implement.

The subtraction & multiplication **happen at the beginning of the frame** as well as the Wait (ms) function (they run in parallel). All functions must finish before moving to the next frame (since the Wait (ms) takes longer, the program will wait for this to finish before moving on).



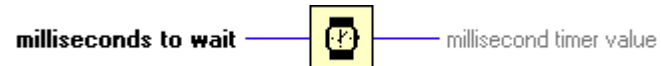
Wait (ms) can be helpful when communicating with some instruments that can't process data very fast and need a little bit of a delay before communicating with the LV program.

You should also add timing to most programs with loops so that the computer doesn't have to run the loops as fast as possible, which will slow down other computer processes.

# TIMING (P. 220)

The timing functions that I use the most include Wait (ms), Get Date/Time in Seconds, and High Resolution Relative Seconds.

As we mentioned in the While loop slides, the Wait (ms) can add a delay or be added to a while loop to slow it down so that CPU can work on other tasks.



Tick Count (ms): Displays the number of ms that have elapsed. There is not a base reference for this function so you cannot read the tick count (ms) and link it to a real-world date or time. It has a number range from 0 to  $2^{32}-1$ . Therefore, there may be a slight chance that if you are subtracting two tick counts, a rollover could have happened which would mess up your value.

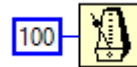


The Get Date/Time in Seconds function returns the time and date. This is really useful to put in code whenever you are collecting data so that you know exactly when your data was taken.



# TIMING (P. 220)

The Wait Until Next ms Multiple function acts very similarly to the Wait (ms) function, except it waits until the system clock is at a multiple of some defined number. Outdated.



For example, if the system clock is at 10537, then the Wait Until Next ms Multiple function would wait until the system clock was 10600 before moving on. If this were in a While loop, then the first iteration would likely run for less than 100 ms and subsequent iterations would run for exactly 100 ms.

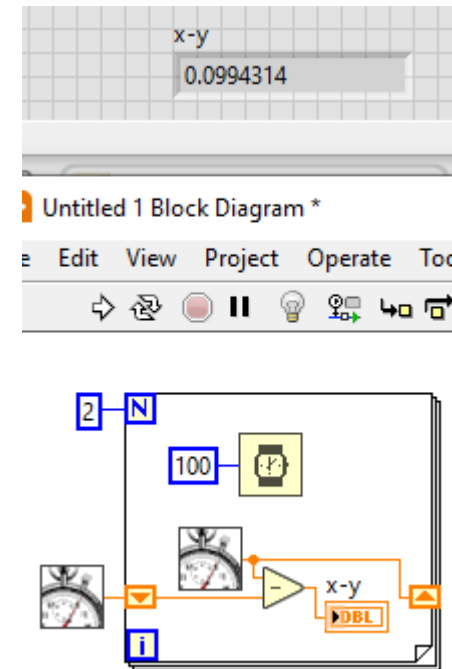
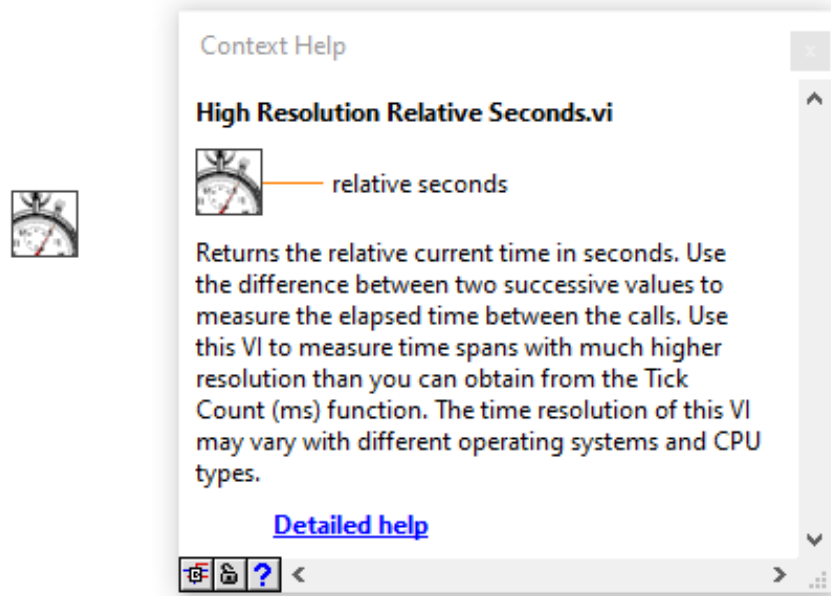
For example, if the system clock were 10537, then the first While loop iteration would wait 63 ms for the system clock to become 10600 before moving on to the next iteration. The next iteration would wait for the clock to become 10700, so it would last for 100 ms. And all iterations afterwards would run for 100 ms.

I prefer the Wait (ms) function because it ensures that every iteration runs for the defined time, so I never use the Wait Until Next ms Multiple.



# TIMING (P. 220)

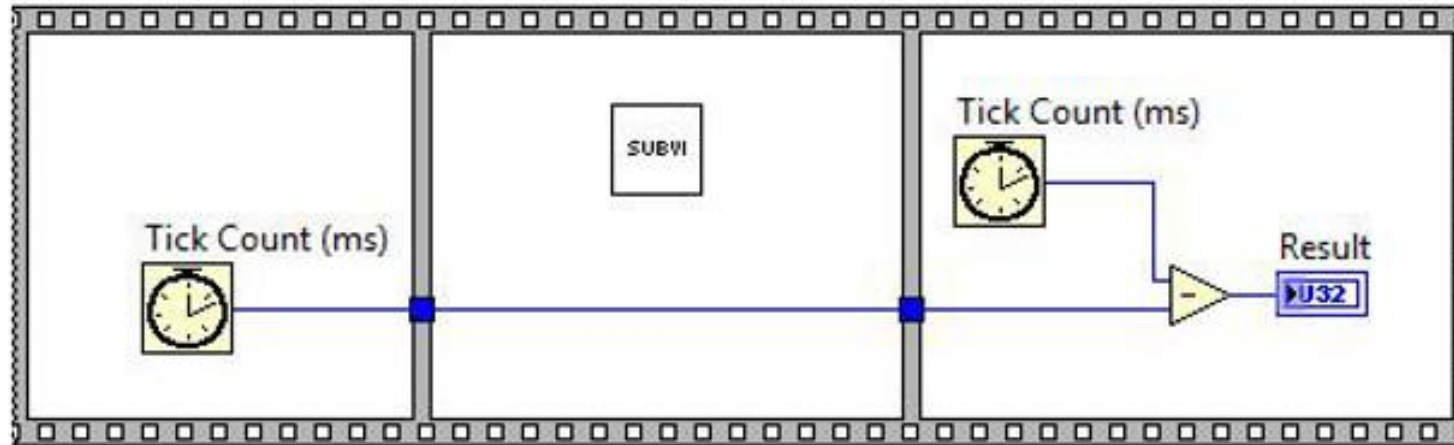
A timing VI that came out recently is the High Resolution Relative Seconds. I especially like this when I want to know the precise time between 2 consecutive iterations.



As you can see, the wait(ms) is set to 100 ms, but a more precise value of the time between iterations is 99.4314 ms.

# CLAD QUESTION

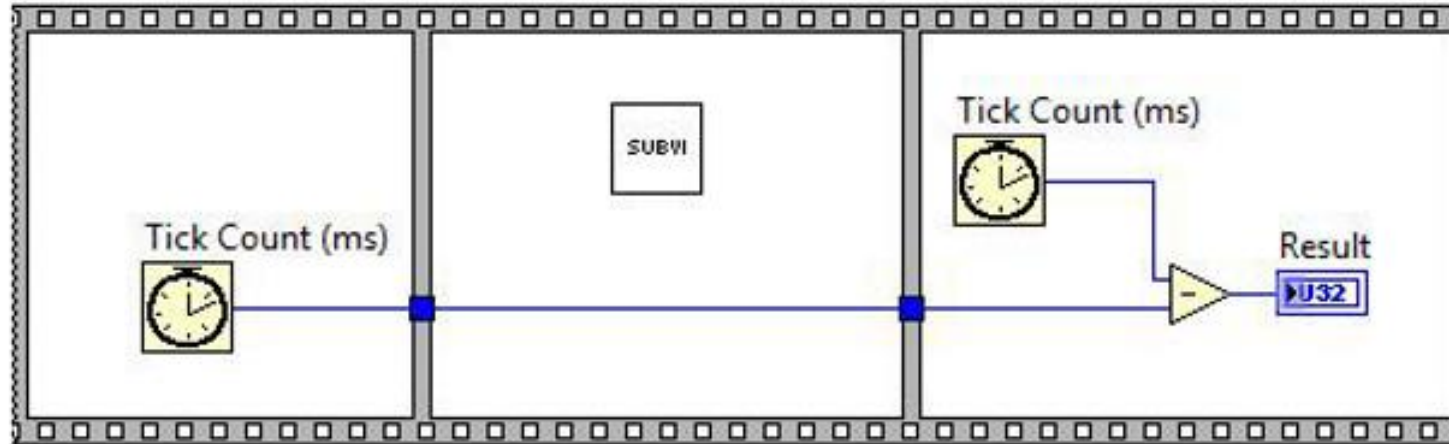
What value does the **Result** indicator display after the code snippet executes?



- a. The time elapsed in milliseconds during the execution of SubVI
- b. Zero
- c. Number of seconds elapsed since January 1, 1970
- d. The time elapsed in milliseconds during the execution of the sequence structure

# CLAD QUESTION

What value does the **Result** indicator display after the code snippet executes?



- a. The time elapsed in milliseconds during the execution of SubVI
- b. Zero
- c. Number of seconds elapsed since January 1, 1970
- d. The time elapsed in milliseconds during the execution of the sequence structure

# CLAD QUESTION

**Q15:** Which timing function can result in logic errors when it rolls over to zero?

**A**

Wait Until Next ms Multiple



**B**

Tick Count (ms)



**C**

Get Date/Time In Seconds



**D**

Wait (ms)



# CLAD QUESTION

**Q15:** Which timing function can result in logic errors when it rolls over to zero?

**A**

Wait Until Next ms Multiple



**B**

Tick Count (ms)



**C**

Get Date/Time In Seconds







**D**

Wait (ms)







# CLAD QUESTION

Which timing function (VI) is the best choice for timing control logic in applications that run for extended periods of time?

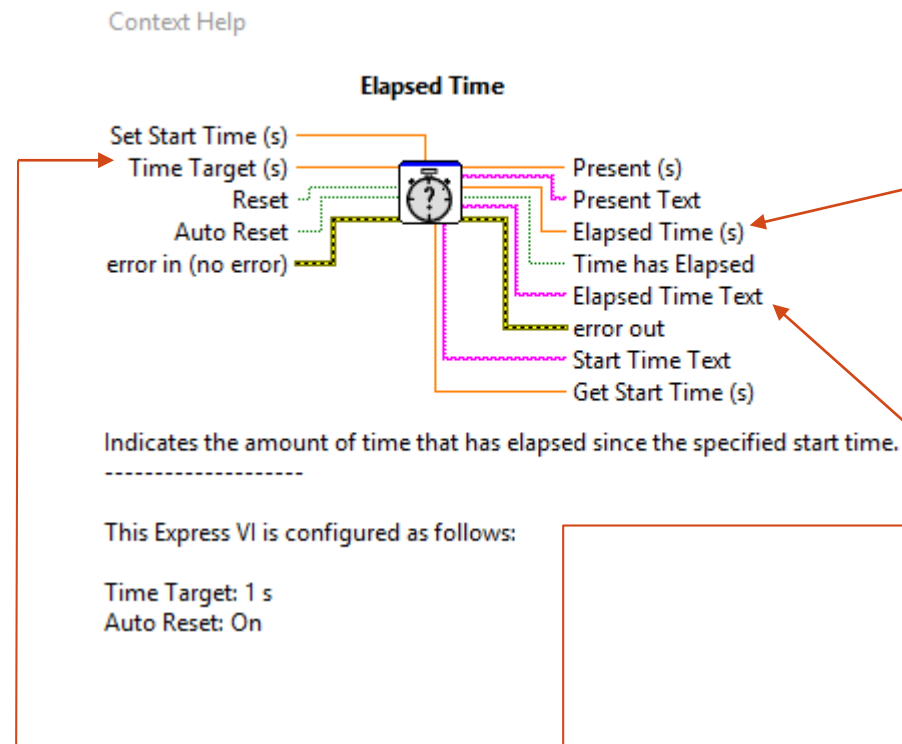
- a.  Tick Count (ms)
- b.  Wait (ms)
- c.  Get Date/Time In Seconds
- d.  Format Date/Time String

# CLAD QUESTION

Which timing function (VI) is the best choice for timing control logic in applications that run for extended periods of time?

- a.  Tick Count (ms)
- b.  Wait (ms)
- c.  Get Date/Time In Seconds
- d.  Format Date/Time String

# TIMING – ELAPSED TIME FUNCTION



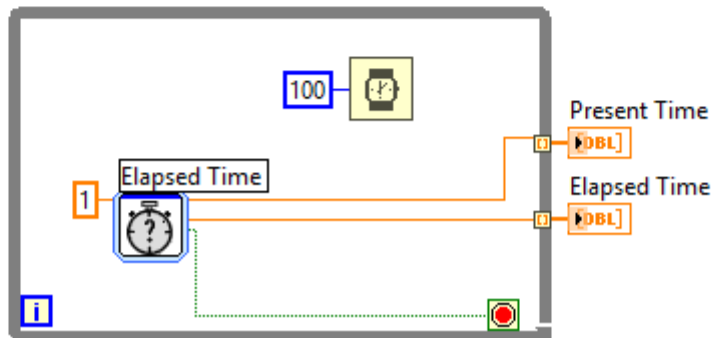
Tells the time that has elapsed since the function is first called. If you put this in a while loop, then it will tell the elapsed time after the first iteration.

Sends out a True if the elapsed time is greater than the Time Target (s). This is great for stopping a while loop after a certain amount of time.



# TIMING – ELAPSED TIME FUNCTION

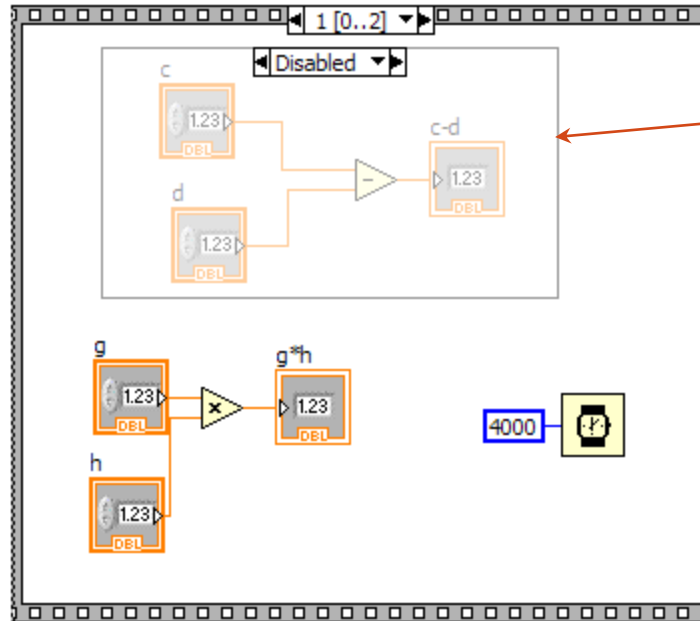
Elapsed Time	Present Time
0	3606298161.9597
0.100109	3606298162.0598
0.199211	3606298162.1589
0.299441	3606298162.2591
0.399653	3606298162.3593
0.49986	3606298162.4595
0.600073	3606298162.5597
0.699175	3606298162.6588
0.79987	3606298162.7595
0.899899	3606298162.8596
1.00007	3606298162.9597
0	0



Stops the While loop after 1 second.

*Note: Present time is not that useful since it's the time after 12AM (Universal Time) on Jan. 1st, 1904.*

# DIAGRAM DISABLE



Doesn't  
implement

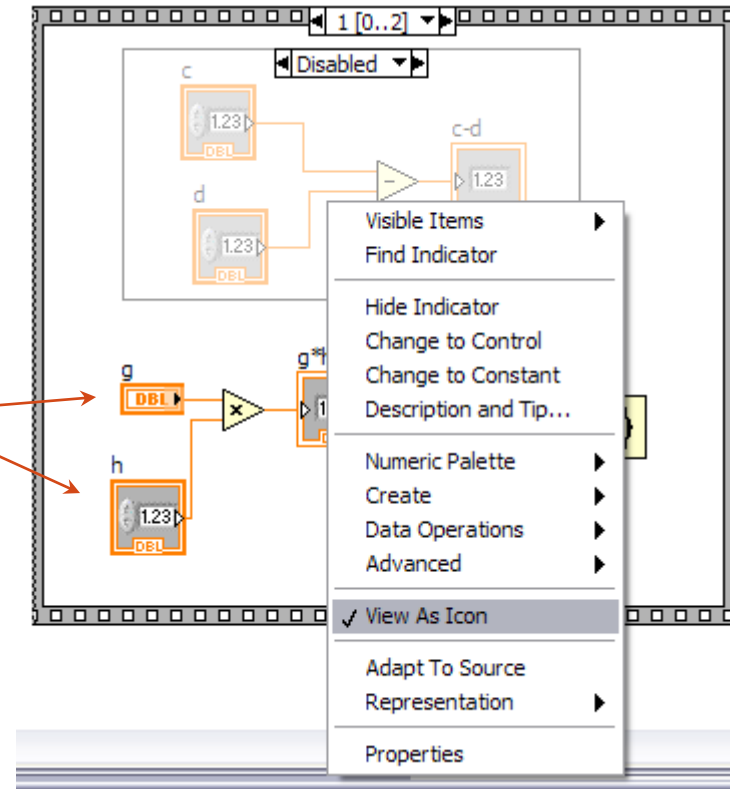
Diagram Disable is a great way to comment out code! This was a nice feature that they added to LabVIEW version 6.

You simply just draw a box around code that you don't want to implement and it will comment the code out. This way can save the code for a later time.

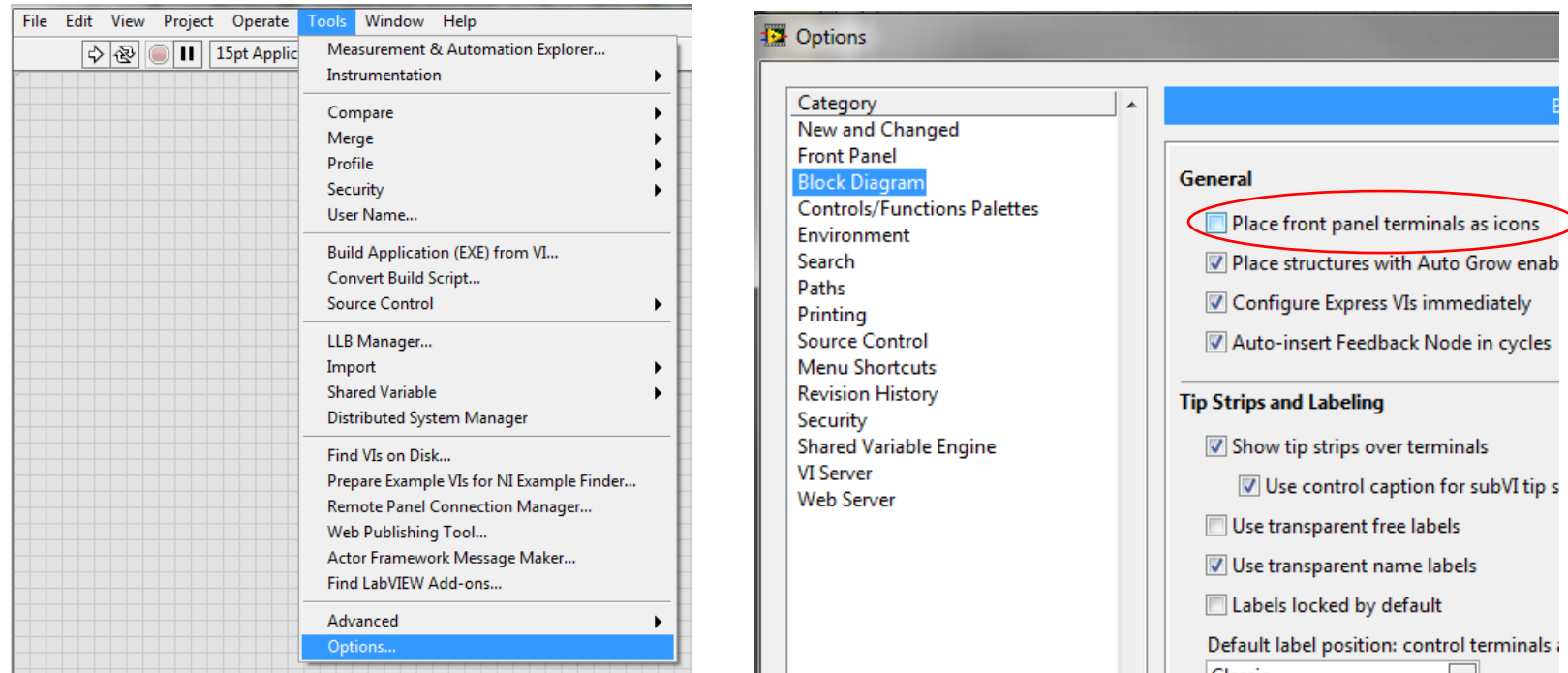
# VIEW AS ICON

Usually the icons on the block diagram take up too much space so I almost always unselect “View As Icon.”

Notice that g and h are the same type of numerical control, but g is much smaller in size than h because I unselected “View as Icon” for g.



# VIEW AS ICON



You can have LV automatically default to using the smaller icons by going to Tools > Options > Block Diagram > Uncheck “Place front panel terminals as icons”.