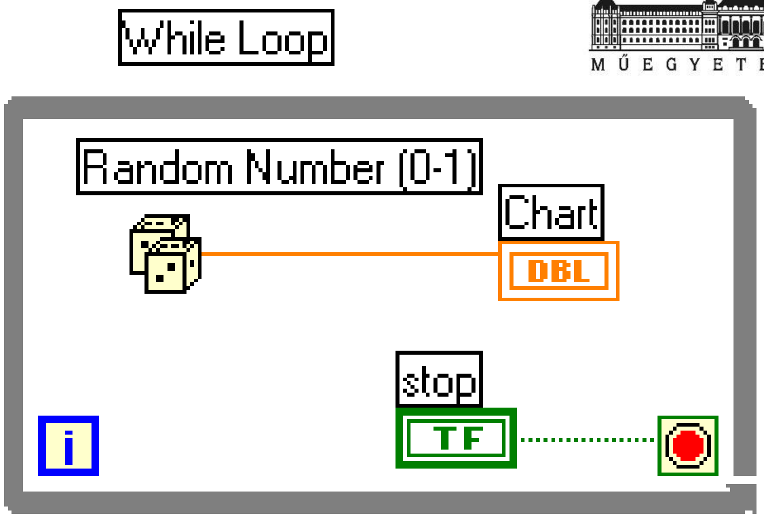
The LabVIEW graphical programming environment
Part 2

Taik Salma, Guo Jian

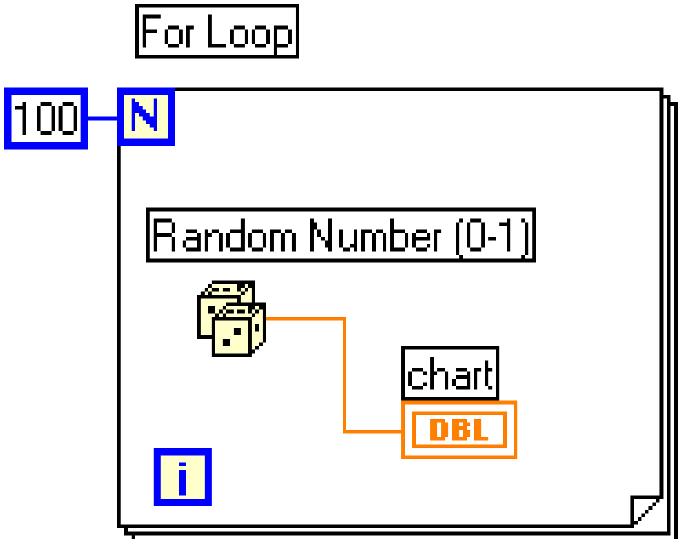
Training Project Laboratory 2020/2021

❖ Loops Structure

- **While Loops**
  - Run according to a condition
  - Run at least once
  - Have an output iteration terminal

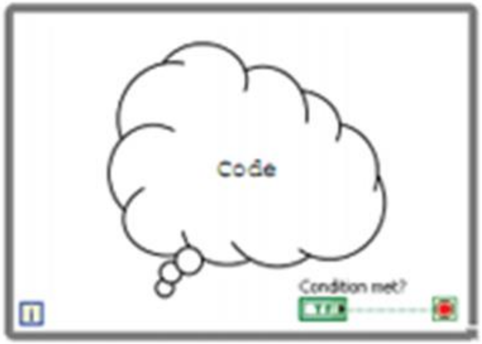


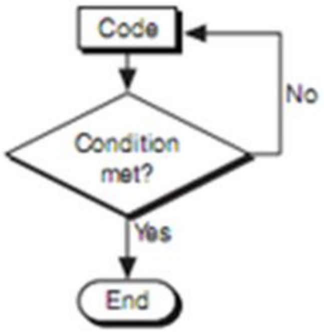
- **For Loops**
  - Run According to input N
  - Have an input iteration Terminal
  - Have an output iteration terminal



❖ While Loops

- Similar to a Do Loop or a Repeat-Until Loop in text-based programming languages,
- It executes the code it contains until a condition is met,

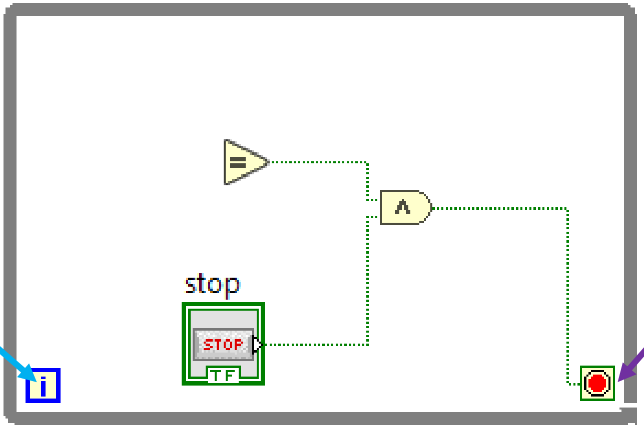




- **Reads the initial values once before the loop starts,**
- Functions Palette

>> Structures

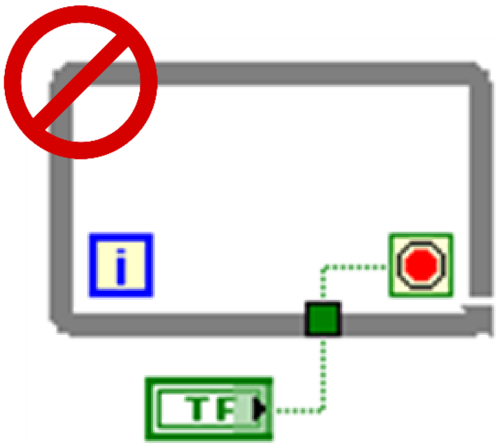
Output iteration terminal: Indicates the number of completed iterations

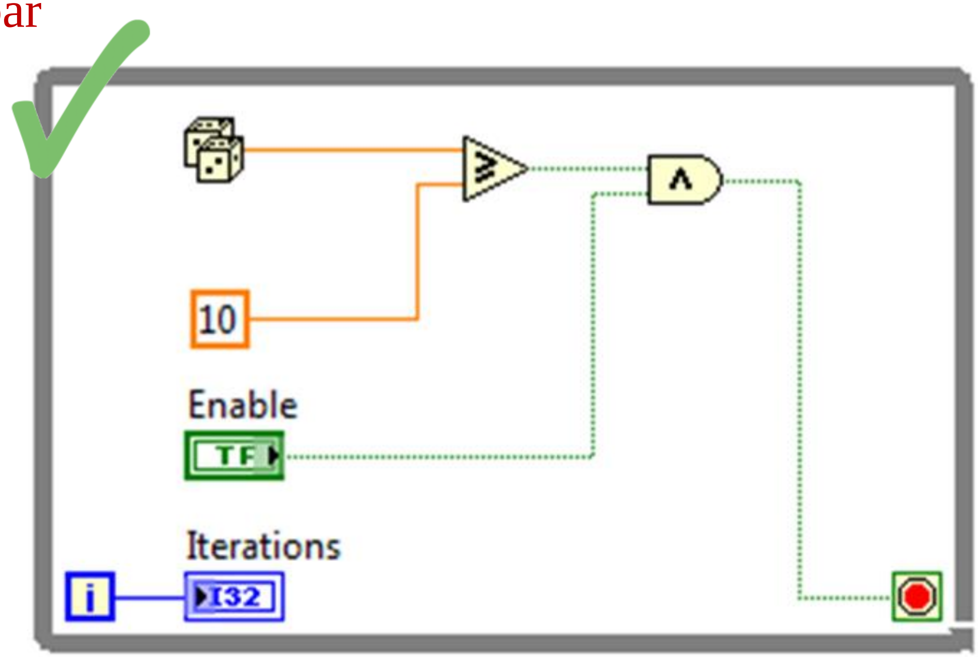


Conditional terminal: Receives a specific Boolean to stop or to continue

❖ Infinite Loops

- The condition of the termination of the While loop should be always inside of the loop to avoid generating an Infinite Loop
- To stop an infinite loop, you must abort the VI by clicking the **Abort Execution** button on the toolbar

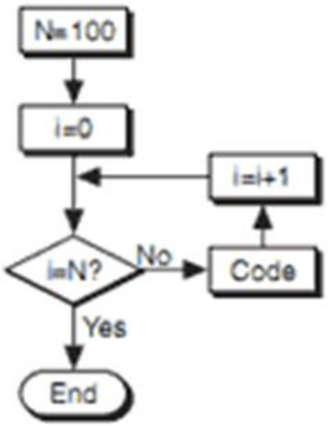




❖ **For Loops**

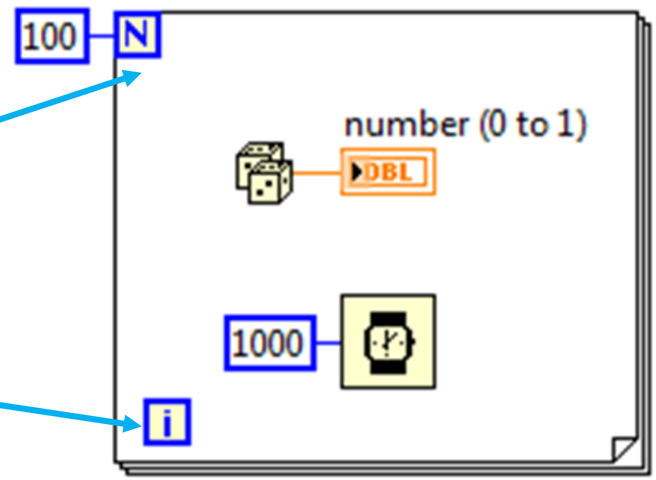
- Executes the code it contains N number of times.
- The iteration count starts from 0.
- Functions Palette

>> Structures



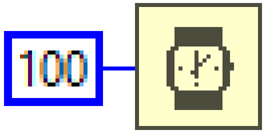
Loop count: input terminal to specify the iterations number

Output iteration terminal: Indicates the number of completed iterations



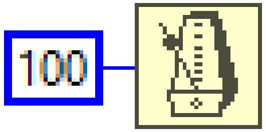
❖ Time in Loops

1) Wait function



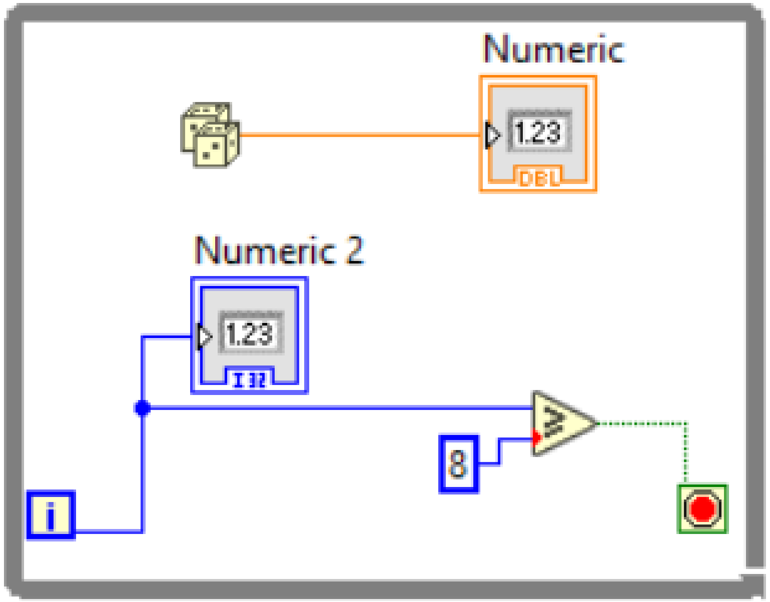
- Allows to control the iteration frequency or timing
- The input of the function is in millisecond
- If the function is placed inside the loop, the second execution of the loop is after the amount of time specified in the Wait function

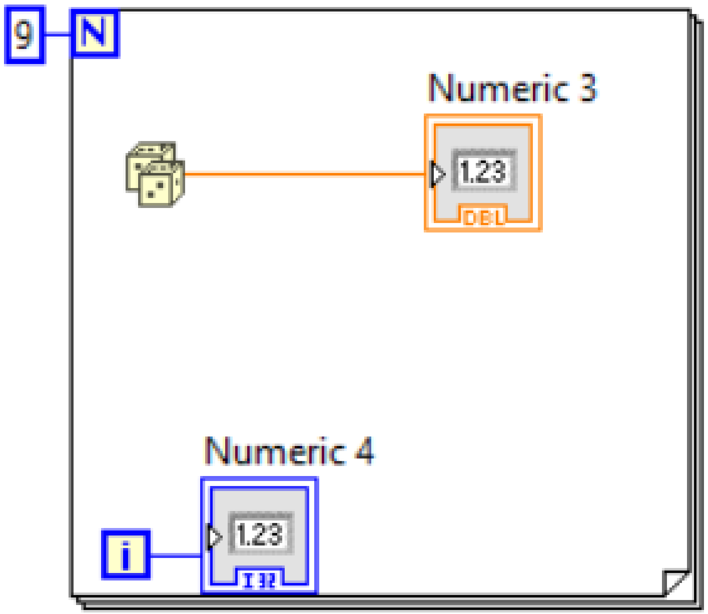
2) Wait until next ms multiple



- Waits until the value of the system millisecond clock becomes a multiple of the specified **millisecond multiple**
- Can be used this to synchronize activities.

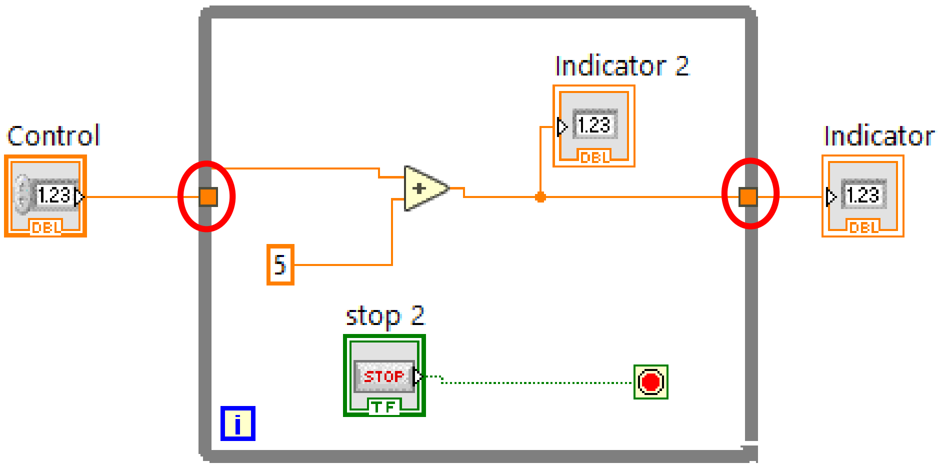
❖ Example : While and For loops

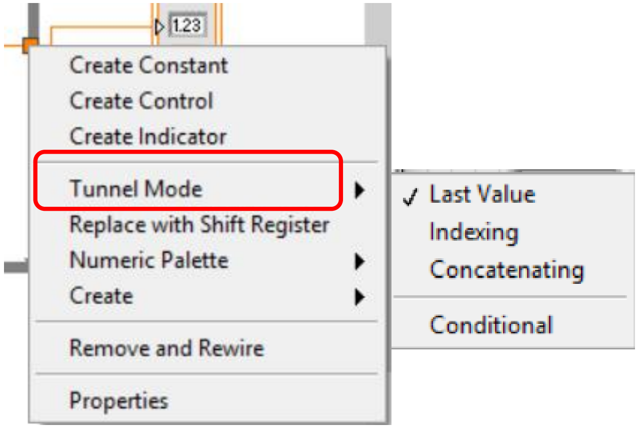




❖ Tunnel modes

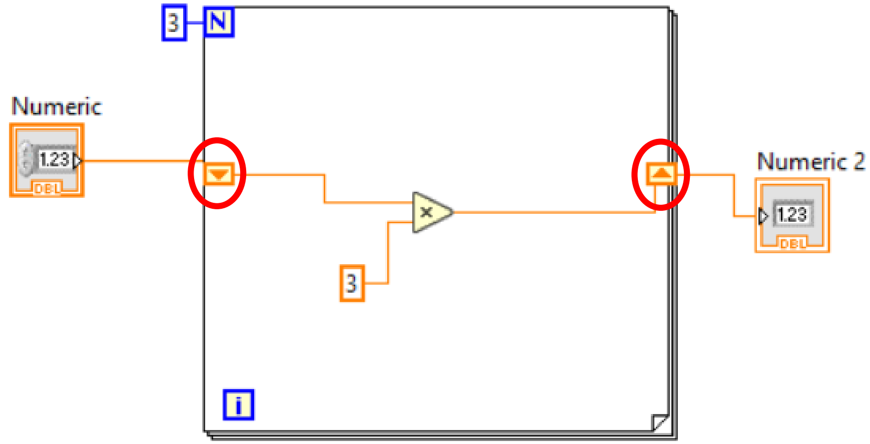
- The tunnels feed data into and out of a structure,
- The Loop starts only when the data arrives to the Tunnel,
- Data passes in and out of the tunnel before the loop start and after the loop terminates; respectively,
- Multiple modes are possible for the tunnels:
  - Last Value: display the last value from the last iteration
  - Indexing: builds an array of all the values
  - Concatenating: by leaving the loop, the array in the input will be concatenated
- Right click on the tunnel to change the mode.

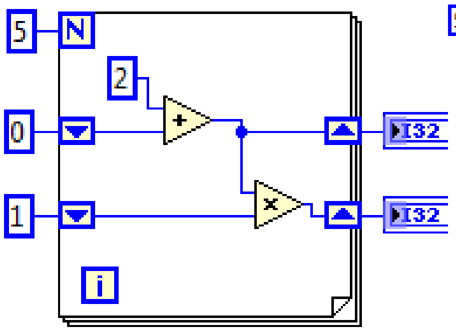




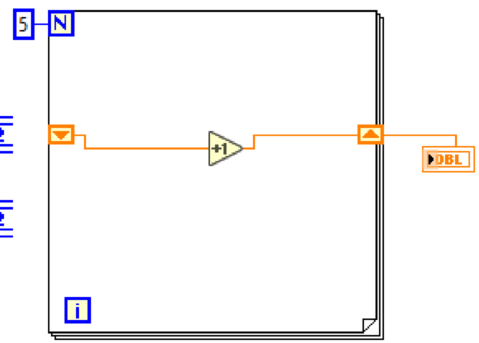
❖ Shift Registers

- A pair of terminals that could be seen as a tunnel,
- Allows to pass the data from previous iteration of the loop to the next one,
- The upper arrow stores data at the end of the iteration,
- If the shift register is not initialized, The value of the last loop execution will be used by the loop





Initialized



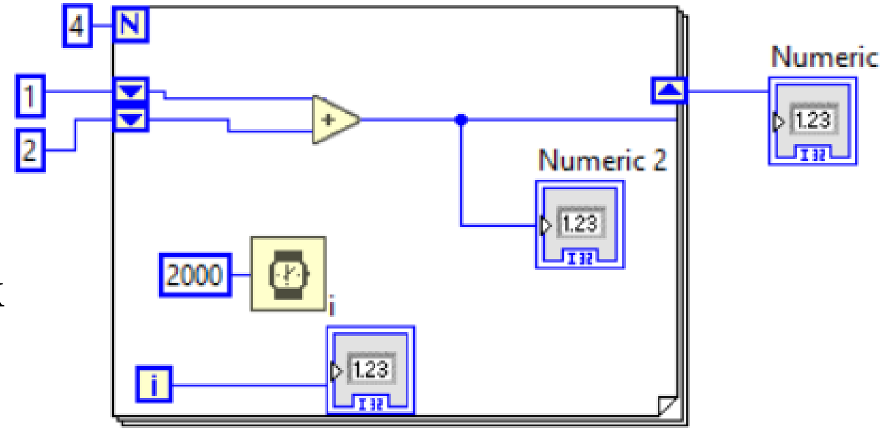
Uninitialized

❖ Stacked Shift Registers

- Allow to access data from previous iterations,

- To create a stacked shift register right-click the left shift register terminal and select Add Element,

- The value after each iteration is stored and passed. The output of he next iteration is passed but the previous value is shifted.



i=0;	i=1;	i=2;	i=3;
1;	3;	4;	7
+	+	+	+
2	1	3	4

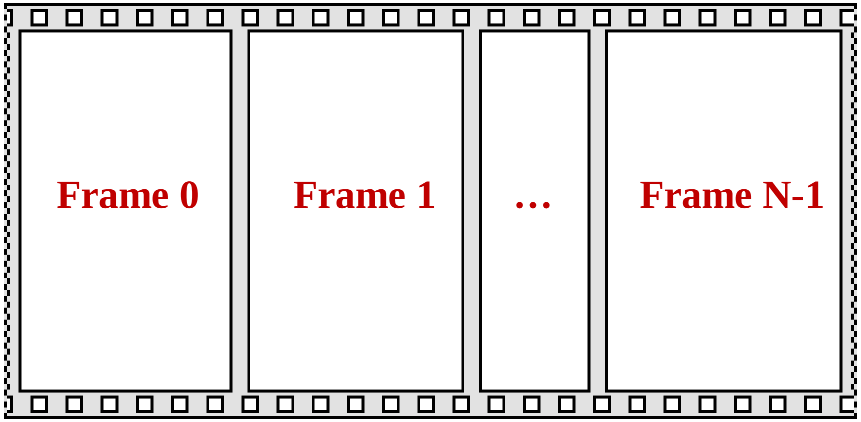
❖ **Sequence Structure**

- LabVIEW uses the Sequence Structure to obtain control flow within a dataflow framework,

- It is an ordered set of frames that execute sequentially,

- It executes frame 0, followed by frame 1, then frame 2, until the last frame executes,

- Data can be transferred through the frames of the sequence.



❖ Sequence Structure

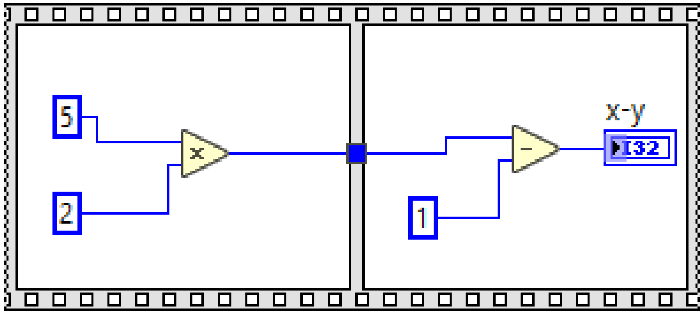
Execution direction

- **Flat sequence:**

➔ The frames are organized sequentially side-by-side and executed from left to right,

➔ Data is passed through tunnels,

➔ All the frames of the program are visible.

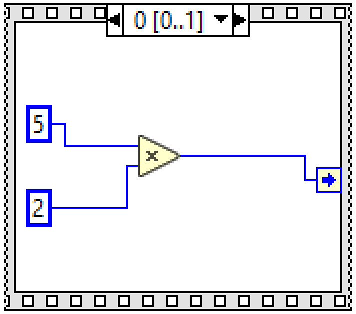


- **Stacked sequence:**

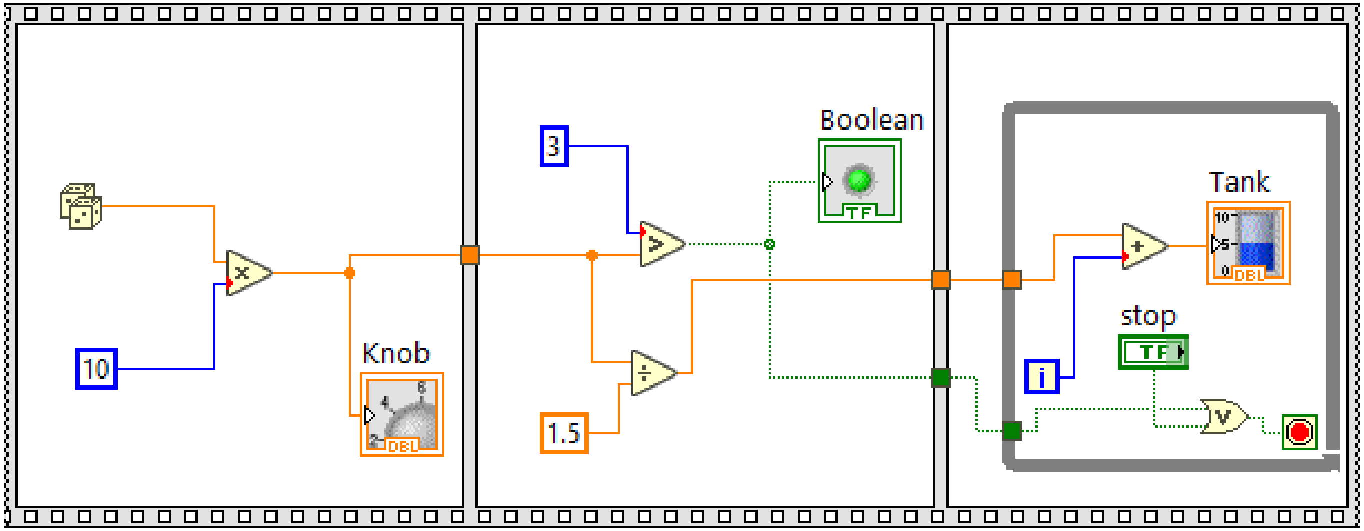
➔ the frames are stacked and numerated from 0 to N-1

➔only one frame is visible at a time,

➔ sequence local (data source/sink) should be initialized to pass the data. It can complicate the program



❖ Example: Flat and Stacked structures

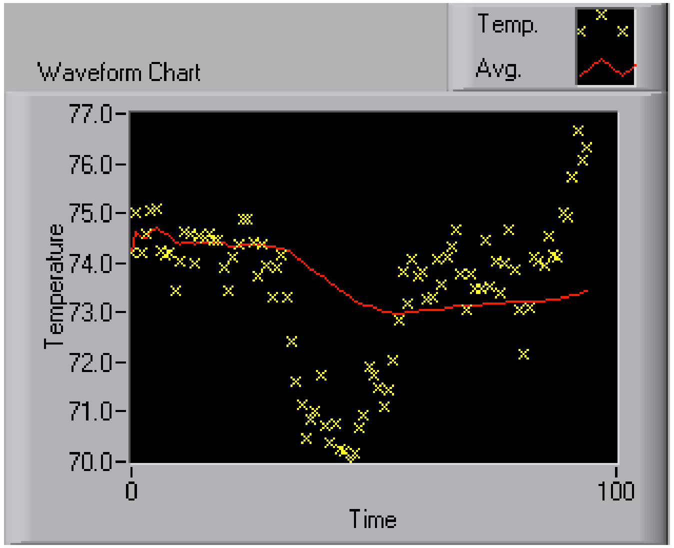


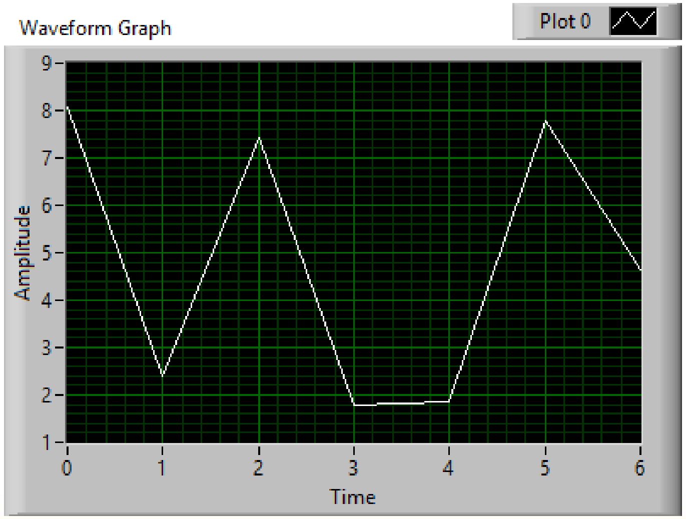
❖ Exercise 6: Loops, arrays and sequences

Build 1D (10*1 elements) and 2D (2*10) arrays with random numbers between 0 and 50. Find the maximum and the minimum of both arrays, and replace them with 50 and 0; respectively.

❖ Charts and Graphs

- Special numerical indicator used to display data in a graphical form,
- They are accessible from the Controls Palette,
- Charts accumulate the data at each execution and display past and new data, each point is displayed separately
- Graphs discard the previously plotted data and display only the new data as an array.

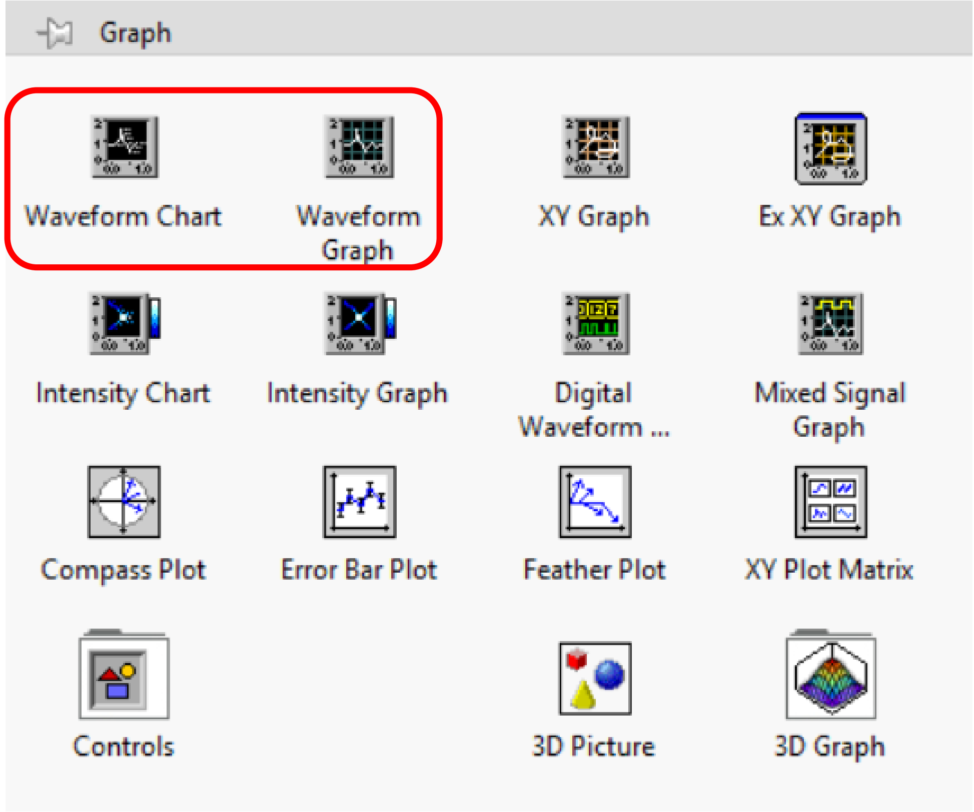




❖ Types of Charts and Graphs

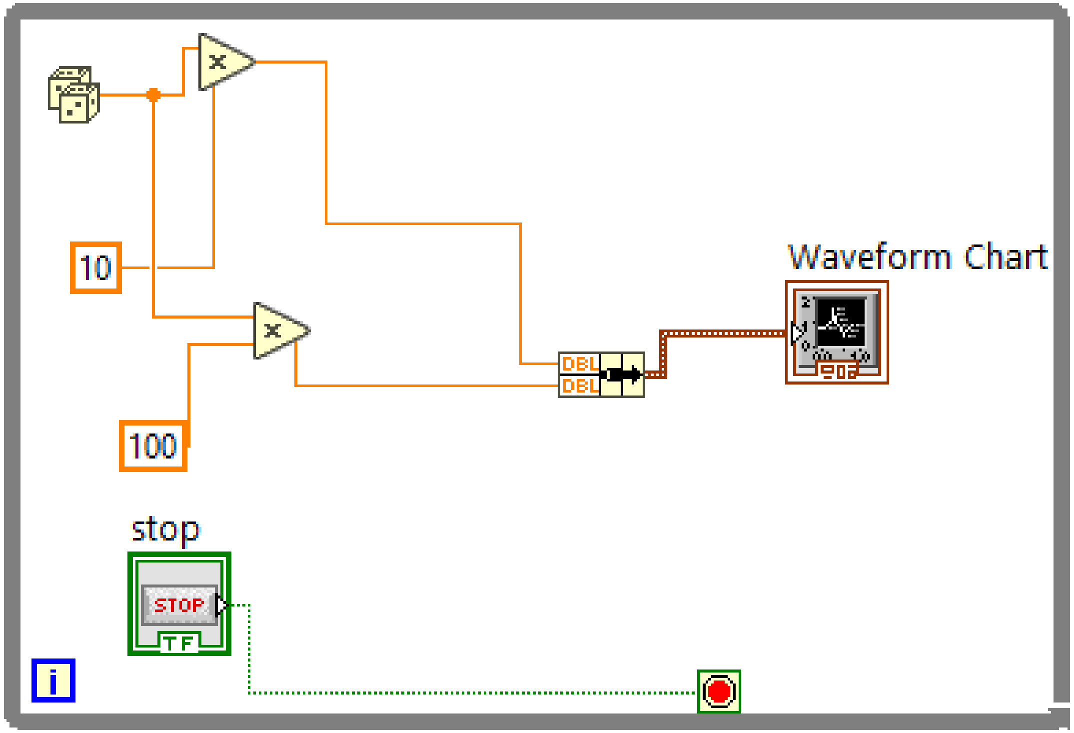
- **Waveform**: Display data acquired at a constant rate

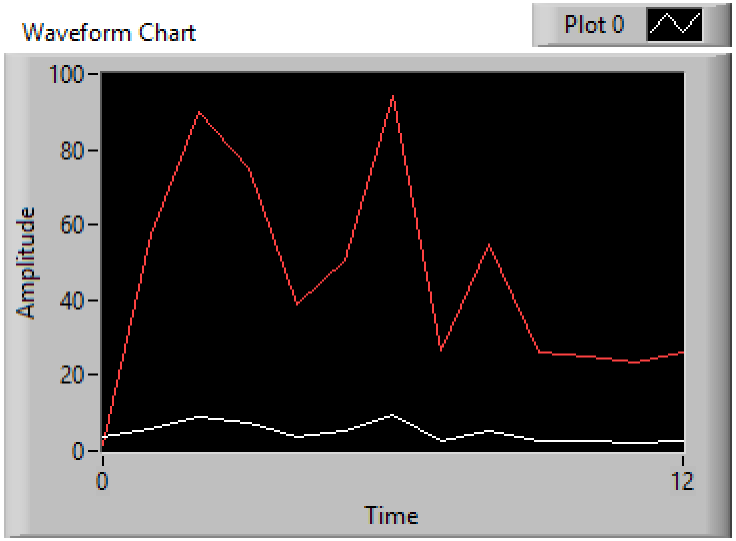
- **XY**: Display data acquired at X and Y axis,



❖ Displaying Multiple Plots on a Chart

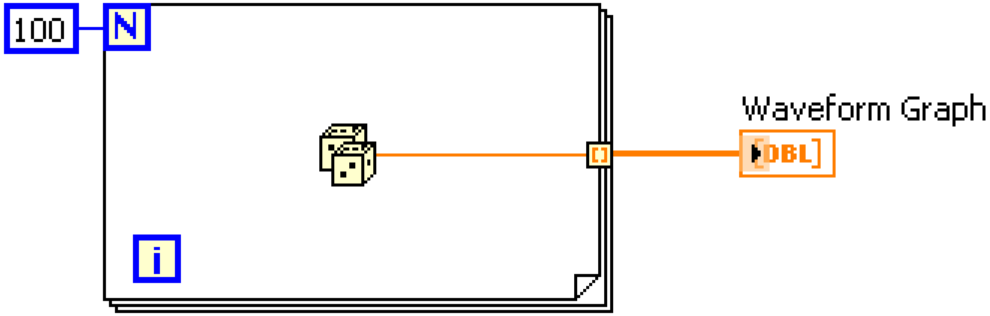
- To display more than one plot on a waveform chart, bundle the data together using the Bundle function.



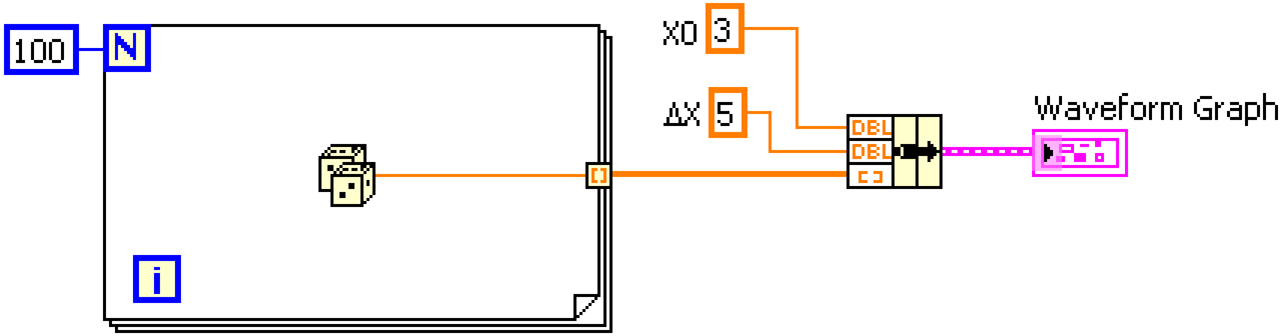


❖ Single-Plot Waveform Graphs

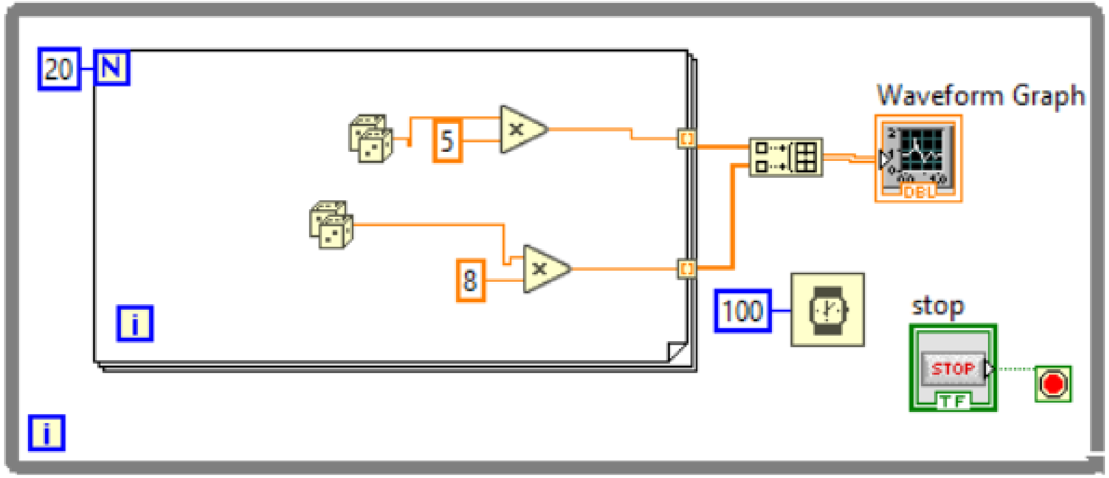
- **Uniform X axis by default:**
  $X_0 = 0$ and $\Delta x = 1$



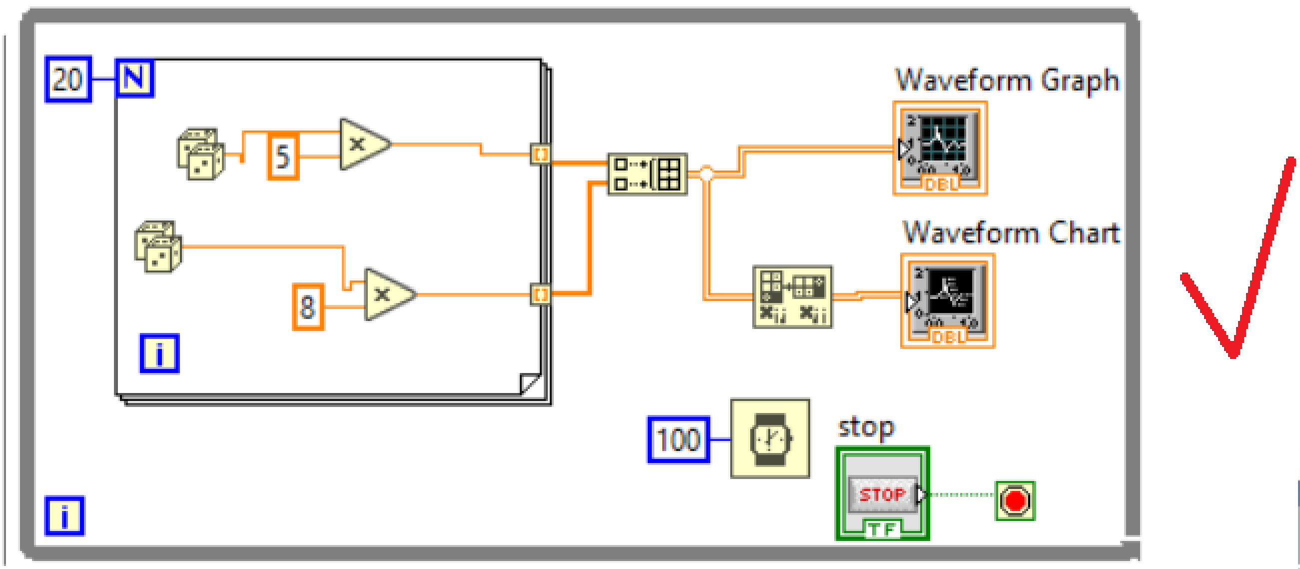
- **Uniform X axis:**
  X_0 and Δx can be chosen using **Bundle**



❖ Displaying Multiple Plots on a Graphs



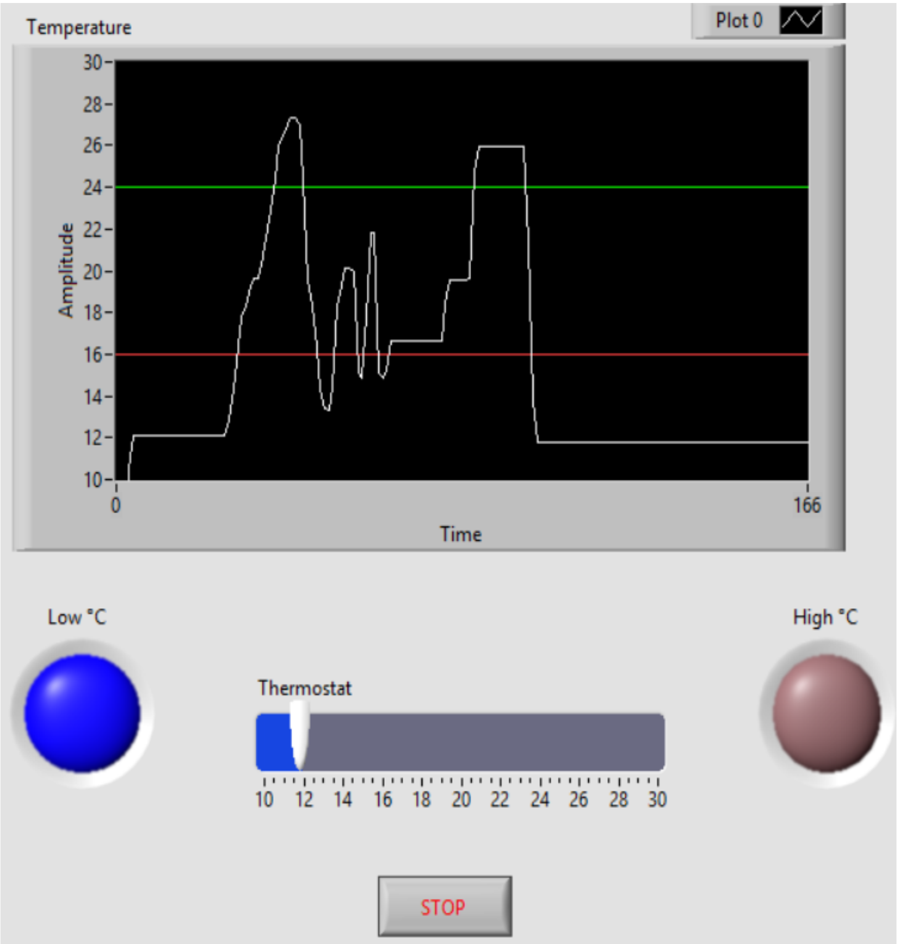
Can we do the same for the 2D waveform charts?



❖ Exercise 7: Monitoring Temperature

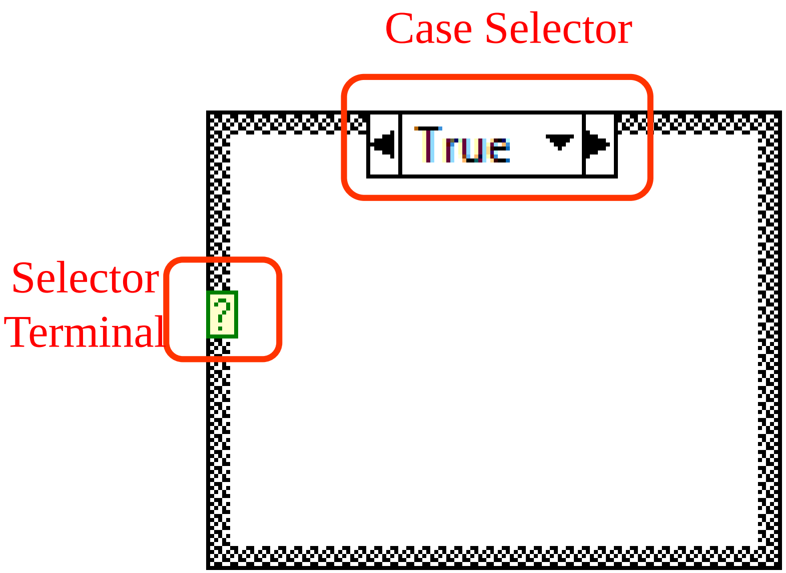
Control the temperature with a thermostat, and visualize the temperature changes with a predefined range of temperature (min and max). If the temperature is higher than the max turn on a LED to indicate it. If the temperature is lower than the min turn on another Led. (different Led colors are possible in the properties of the LEDs)

➔ Use a waveform chart



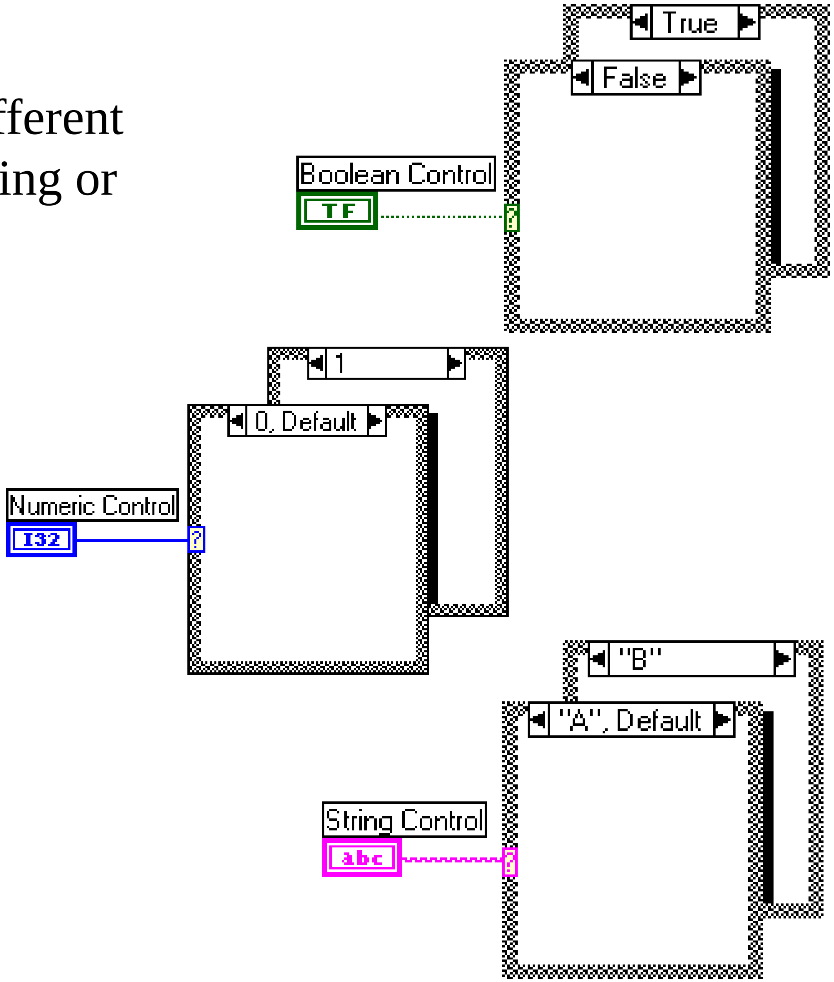
❖ Case Structures

- Conditional structure that has many cases similar to If… Then…Else or to Switch,
- Can have multiple cases,
- The cases are stacked and can be accessed with the Case Selector,
- Only one case is visible and executed at a time,
- The cases are executed based on the input condition (wired value to the Selector Terminal) and the corresponding label,
- Available in Functions palette>> Structures.



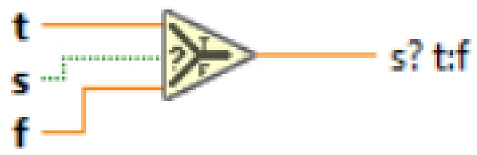
❖ Case Structures

- The Selector Terminal can be wired to different datatypes which are Boolean, Integer, String or Enumerated values,
- The Labels in the Case Selector are adapted to the datatype of the Selector Terminal,
- The cases should be created to cover all the possible input values except for the Boolean input,
- A Default case can be created to handle the other input values not specified in the cases.



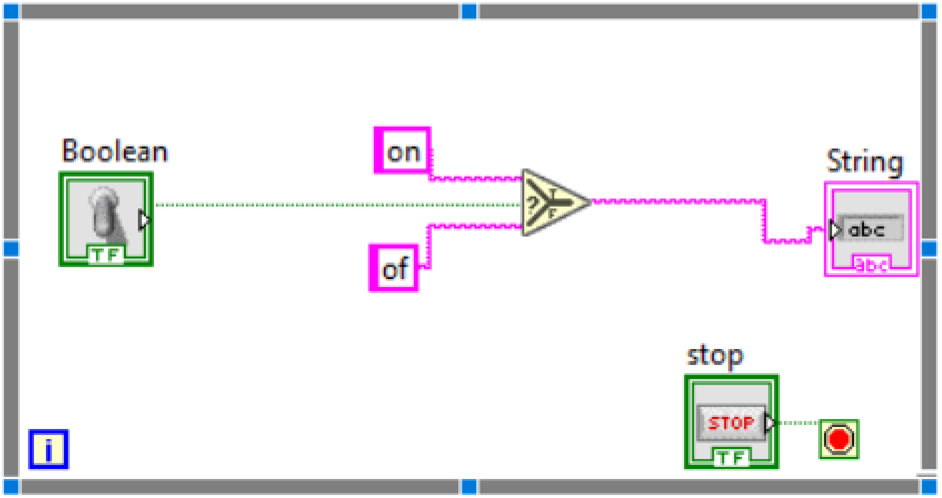
❖ Case Structures

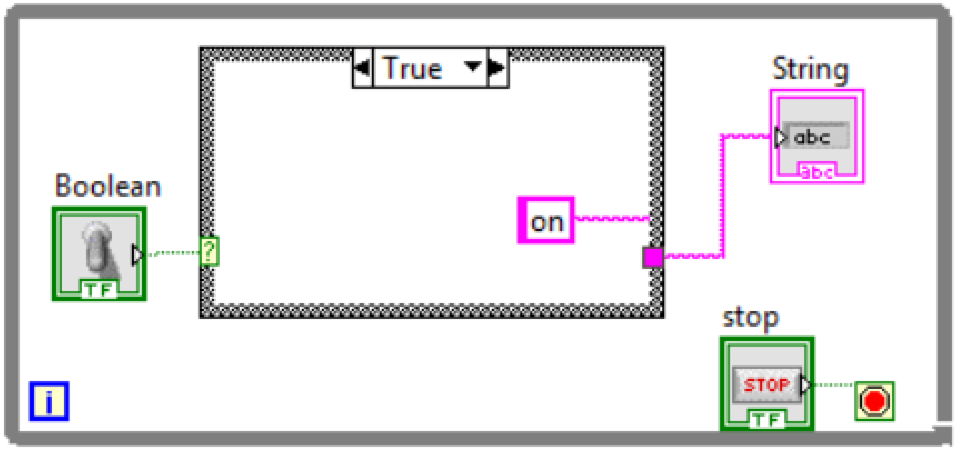
- Select function is similar to the case structures,



- Select function has two possible cases only,

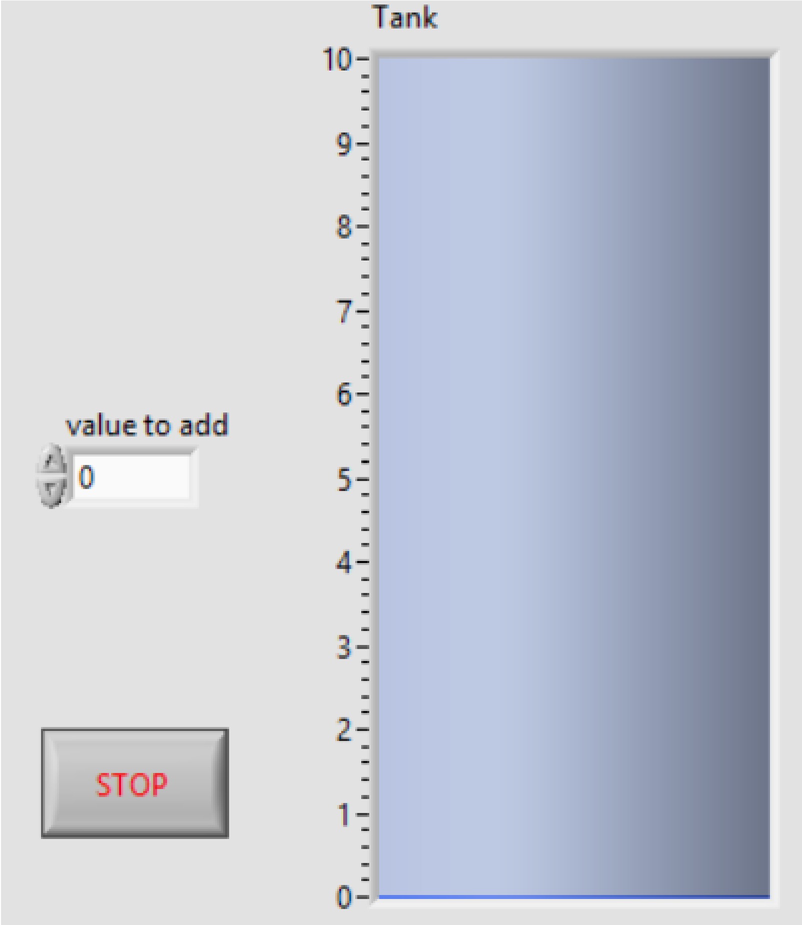
- The case structures can handle multiple cases.



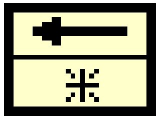


❖ Exercise8: Case Structures

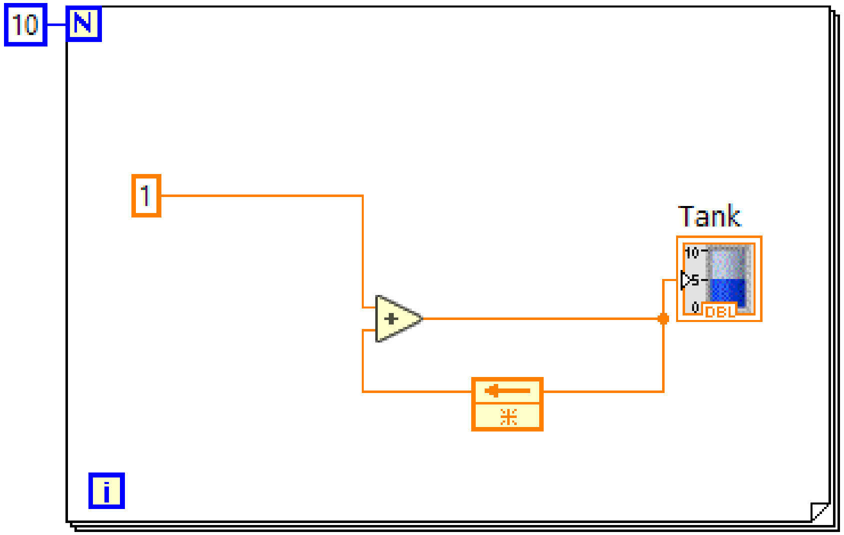
Fill in the tank with a giving value until a certain threshold, then increase or decrease the value added to the tank



❖ Feedback Node

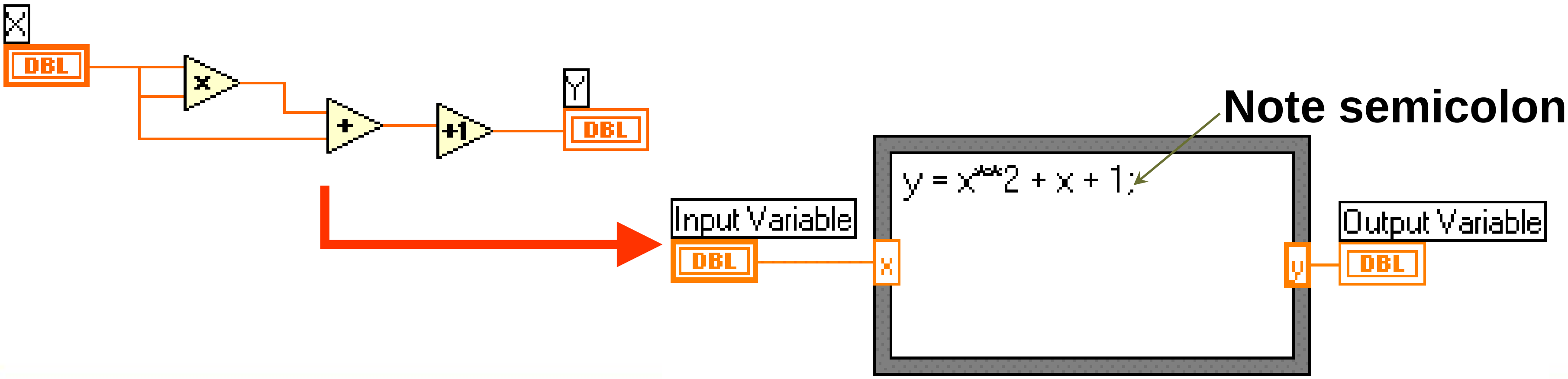


- It appears in Loops when the output of a node is wired to its output
- It stores data from previous block diagram executions or loop iterations
- Does not perform any action on the data it receives, it only passes it to the next input terminal



❖ **Formula Nodes Structure**

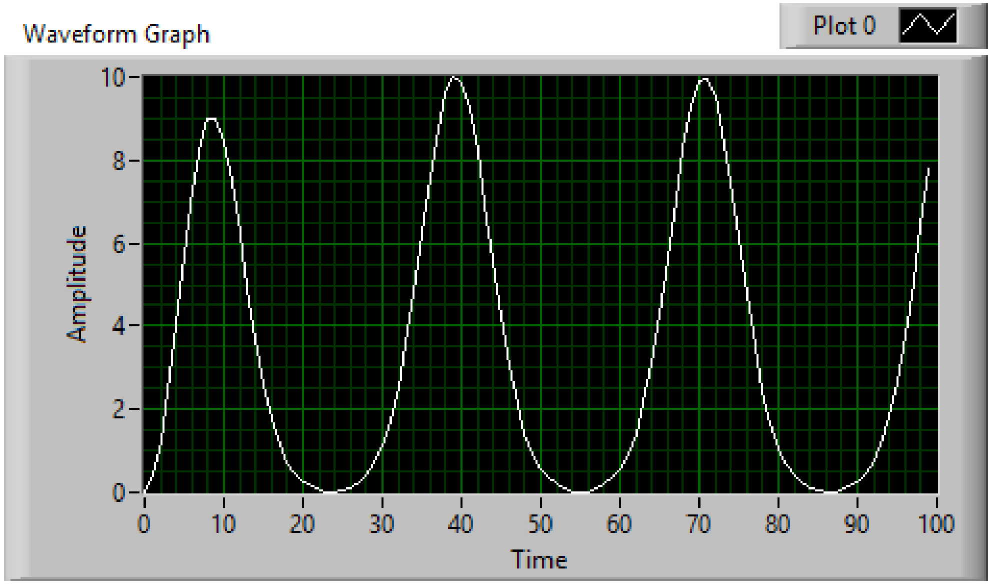
- In the Structures subpalette,
- A convenient, text-based node used for complicated mathematical operations on a block diagram using the C++ syntax structure,
- It receives Inputs to use in the mathematical operation and generates the resulting outputs variables,
- Each statement must terminate with a semicolon (;)



❖ Exercise 9– Formula Nodes

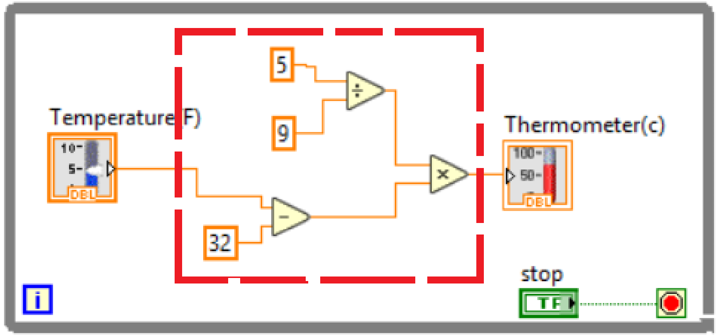
Create an application that solves this equation and display the output signal y.

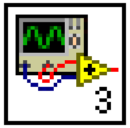
$$a = \tanh(x) + \cos(x)$$
$$y = a^3 + a$$

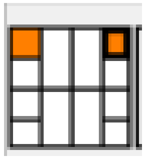


❖ SubVIs

- A VI already created that can be used in another VI as a function/control icon,
- Compress a part of the code into one icon with its own inputs and outputs,
- Can be used as much as needed, no need to recreate the code,
- 3 major parts: the actual code (Front Panel, and Block Diagram), a customizable icon, and the connector pane,

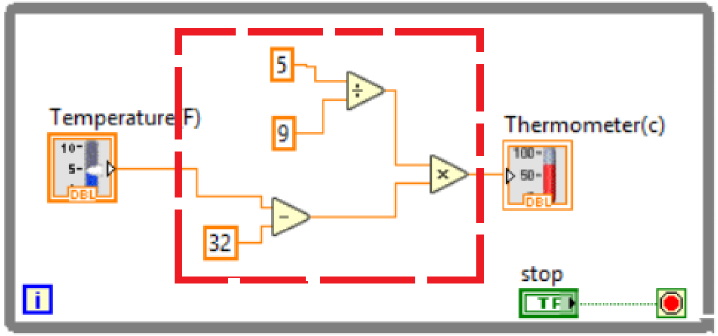


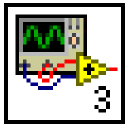


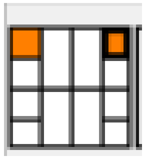


❖ <u>SubVIs</u>

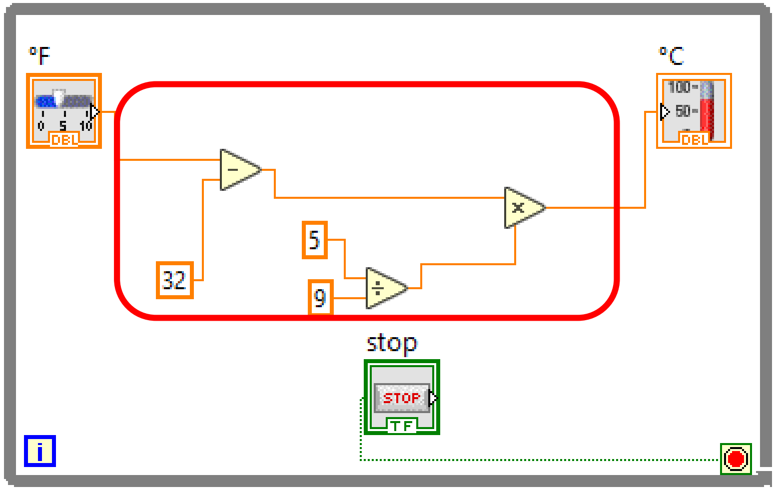
- The actual code can be any wired code of terminals, controls and indicators,
- After creating the subVI, an icon appears that represents the subVI,
- The icon can contain images and text,
- The connector pane shows the set of terminals of the subVI relate to controls and indicators on the front panel
- Terminals can be added by right click on the connector pane





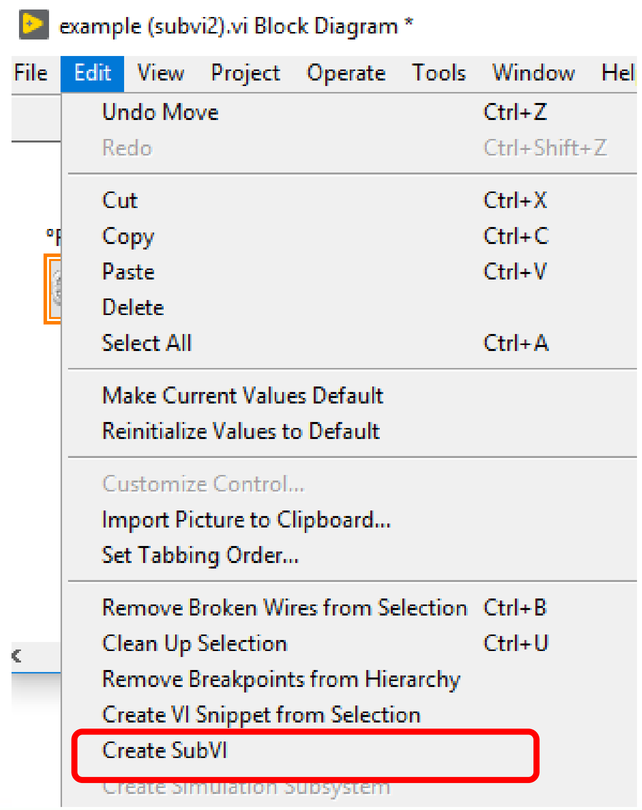


❖ Create a SubVI



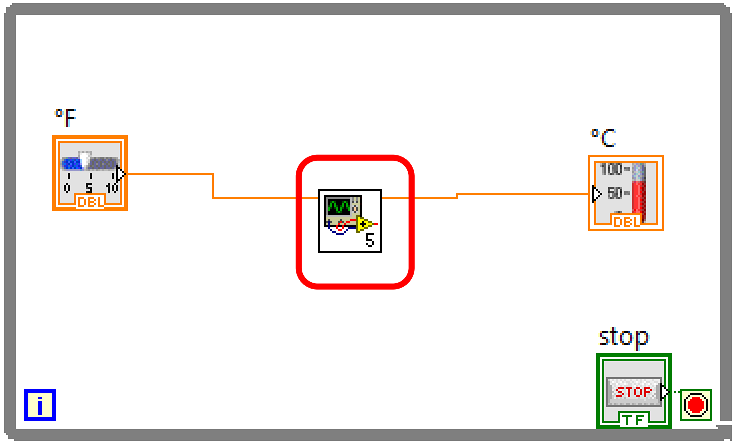
1) Select the part of the code you want

2) Go to Edit>> Create SubVI

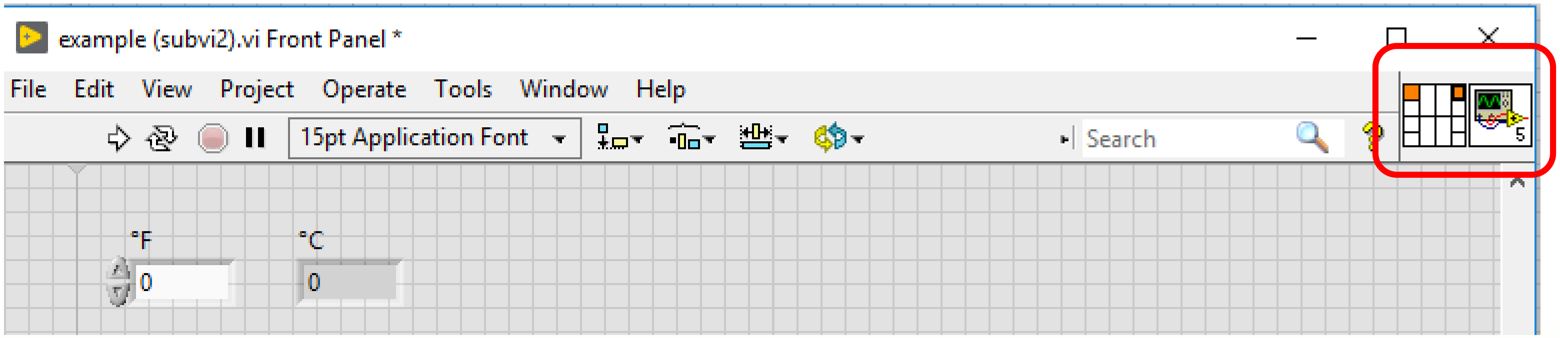


❖ Create a SubVI

3) An Icon will automatically generated appear and the connector pane as well

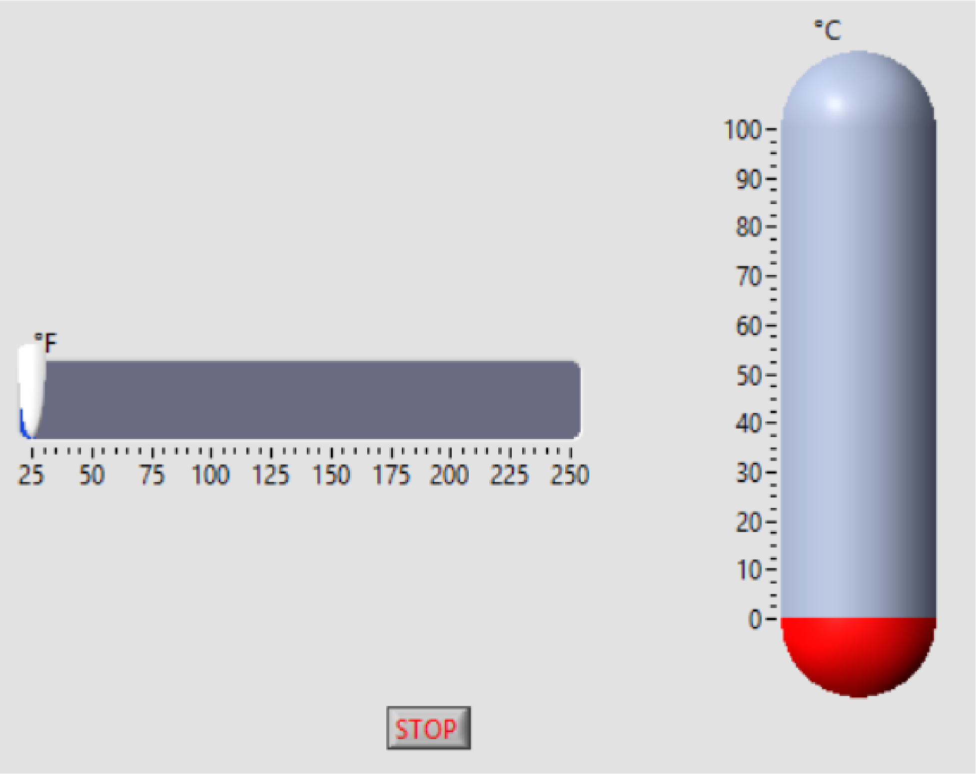


4) Double click on the icon to see the subVI code and to modify the icon and connector pane



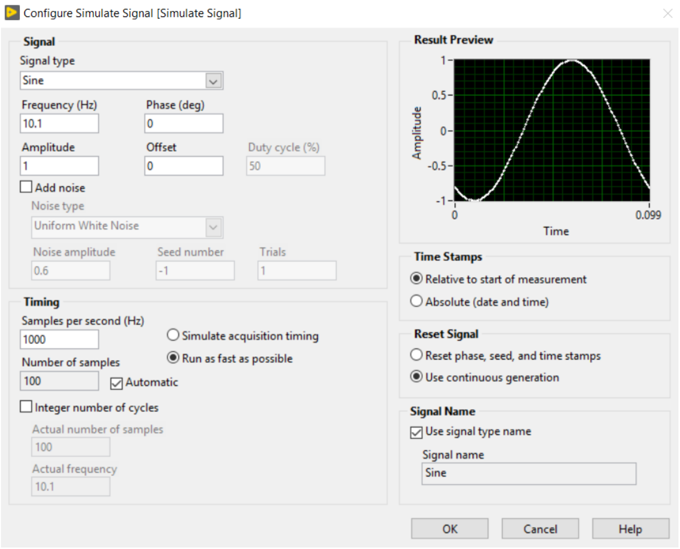
❖ Example: Create a SubVI

➔ Convert Fahrenheit to Celsius: $°C = {}^{5}/_{9}\,(°F - 32)$

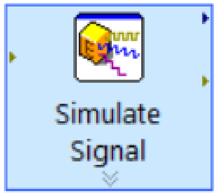


❖ Express VIs

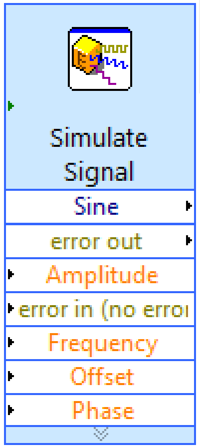
- Nodes that require minimal wiring because they can be configured interactively through a dialog box



- Used for common measurement tasks



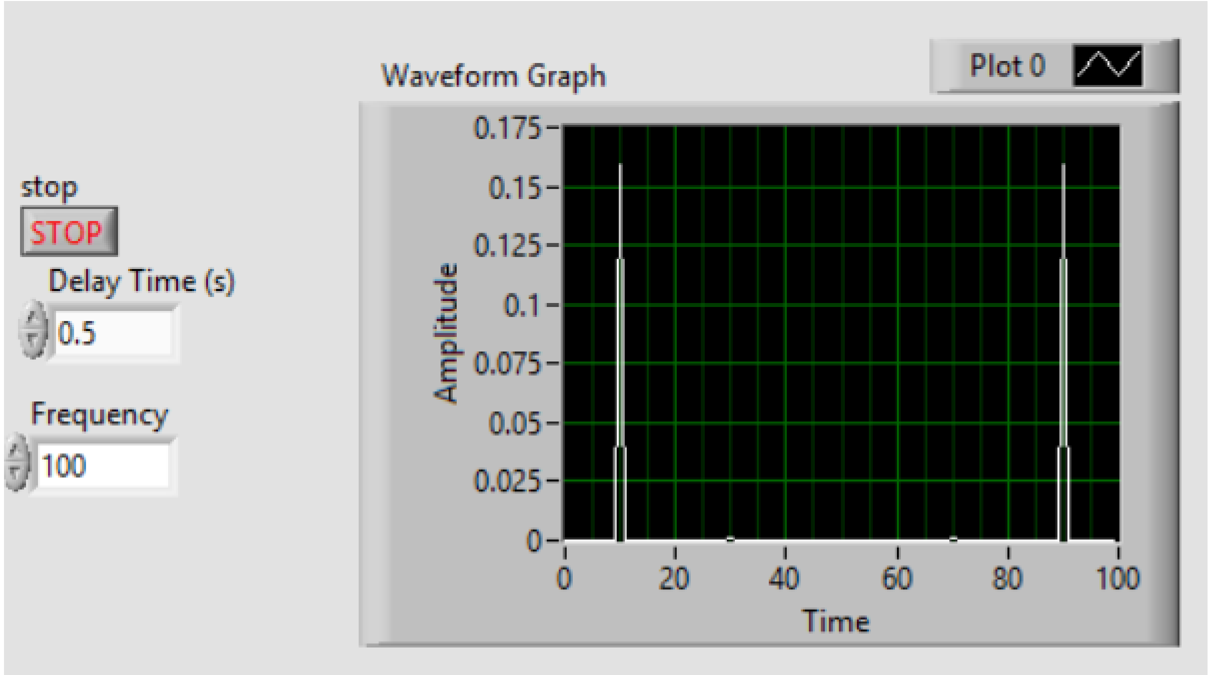
- They appear on the block diagram as expandable nodes with icons surrounded by a blue field



- Can be found in Functions palette >> Express

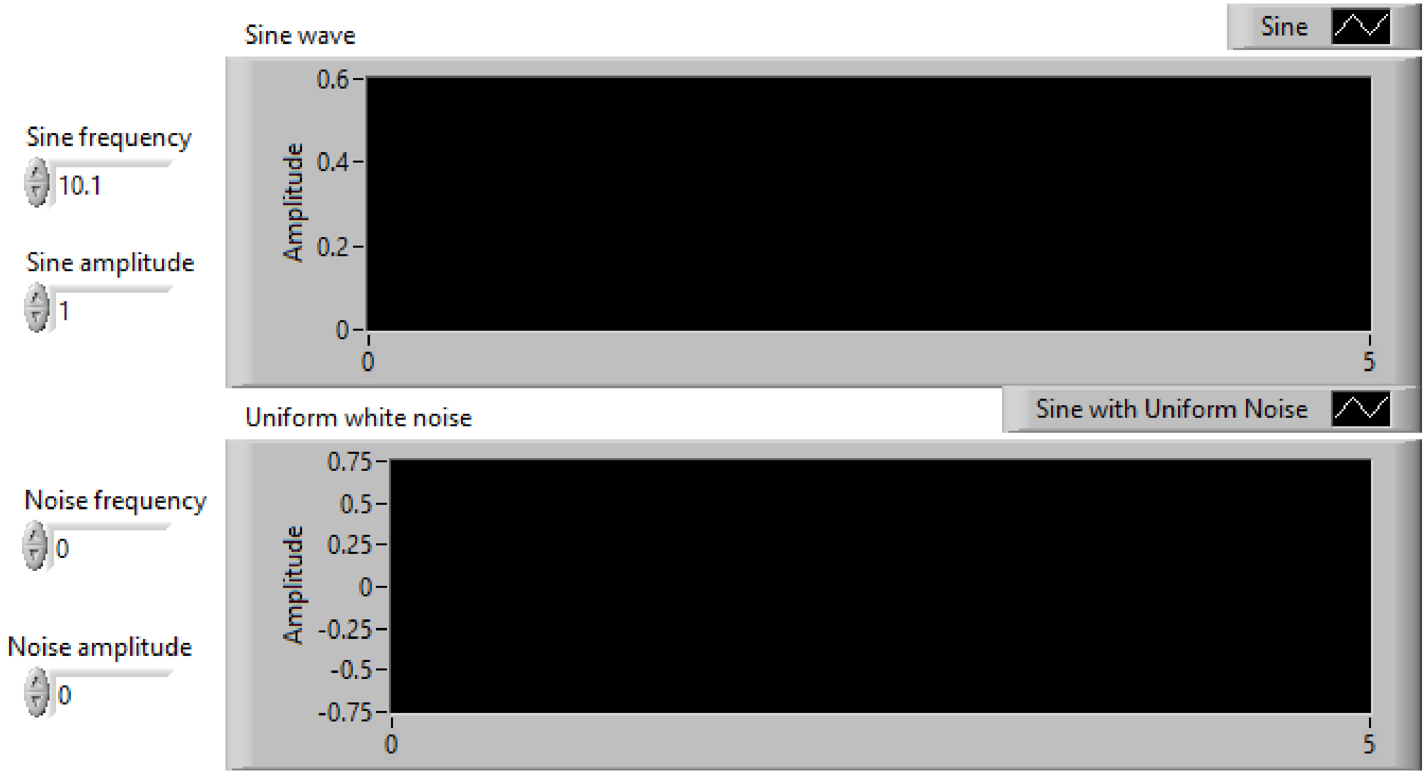
❖ Example: Express VIs

- Power spectrum of a signal



❖ Exercise 10: Express VIs (Part 1)

- Generate a signal (sine wave, triangle, square) and a noise (white, random, etc.).



❖ Exercise 10: Express VIs (Part 2)

- Merge the two signals into one chart, and add the two signals together