



UNIVERSITÀ DEGLI STUDI DI PAVIA

Dept. of Electrical, Computer and
Biomedical Engineering



Data Acquisition Systems and the NI LabVIEW environment

Prof. Lodovico Ratti



Data Acquisition (DAQ)

- Use of some data acquisition technique can be convenient, when not mandatory, in the following situations
 - when remote control of instruments located in dangerous or hardly or non accessible areas is required
 - when repetitive measurements have to be carried in an automatic fashion
 - when data have to be acquired to undergo further processing or to be presented (e.g., presentation of the results from some measurement in some form)
 - when you need to control a process in a closed loop system

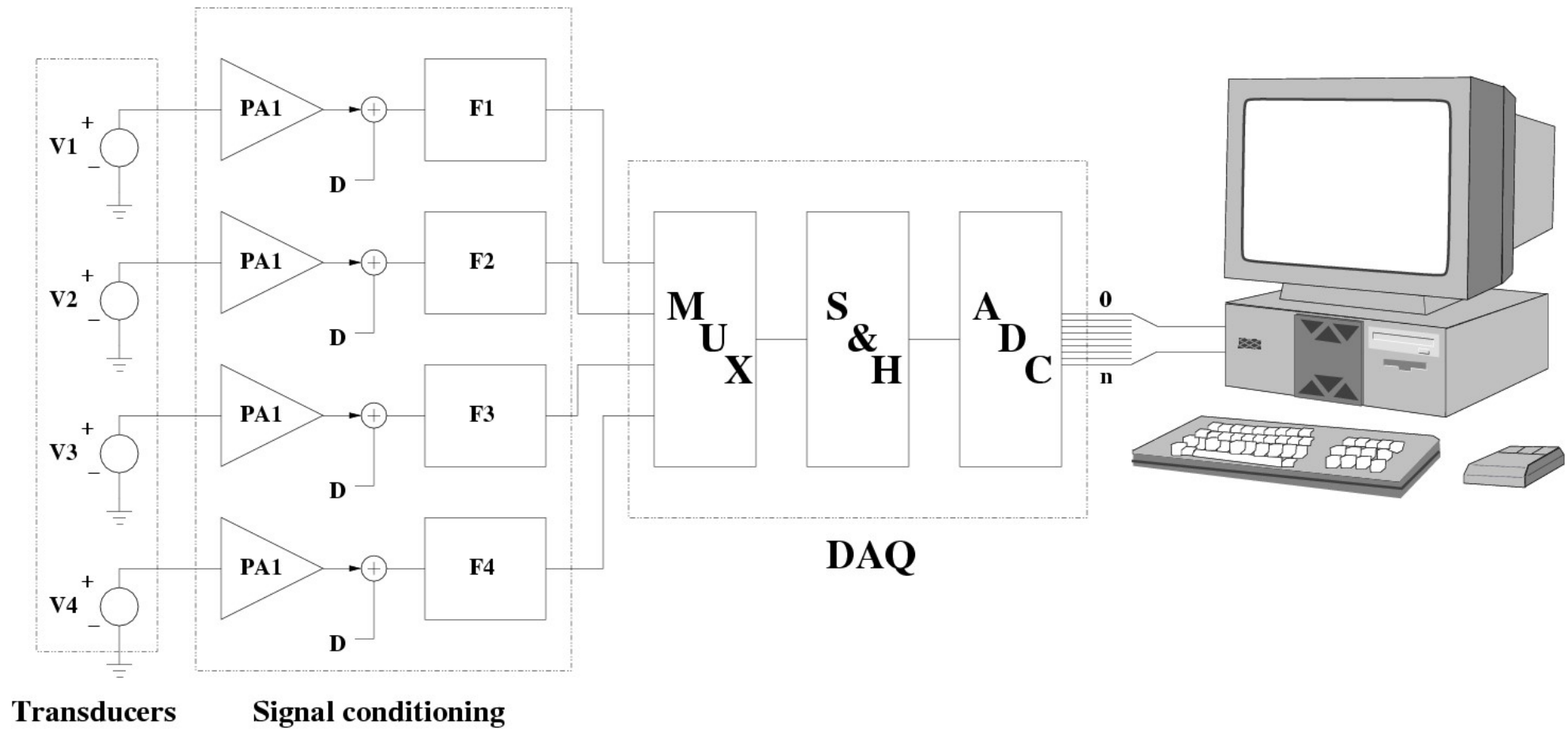


Hardware and Software for DAQ

- Generally, a DAQ system consists of a few interconnected parts, including
 - a transducer
 - some signal conditioning circuits
 - a data acquisition board (DAQ board), equipped at least with an analog to digital converter (ADC) but generally performing other functions (e.g., amplification, multiplexing)
 - a microcontroller, a microprocessor, a PC, which someone has instructed on what to do with the collected data
 - some software providing an interface between the acquisition chain and the user, for data processing and/or representation



Data Acquisition System





Transducers

- Transducers are used to convert a physical quantity (e.g., speed, position, mass, strength, pressure) in an electrical signal (a change in a voltage, current or charge, for example); they may provide
 - an analog signal (e.g., thermocouples, thermistors, photodiodes)
 - a digital signal (like in the case of an encoder for position or speed measurements)





Signal Conditioning

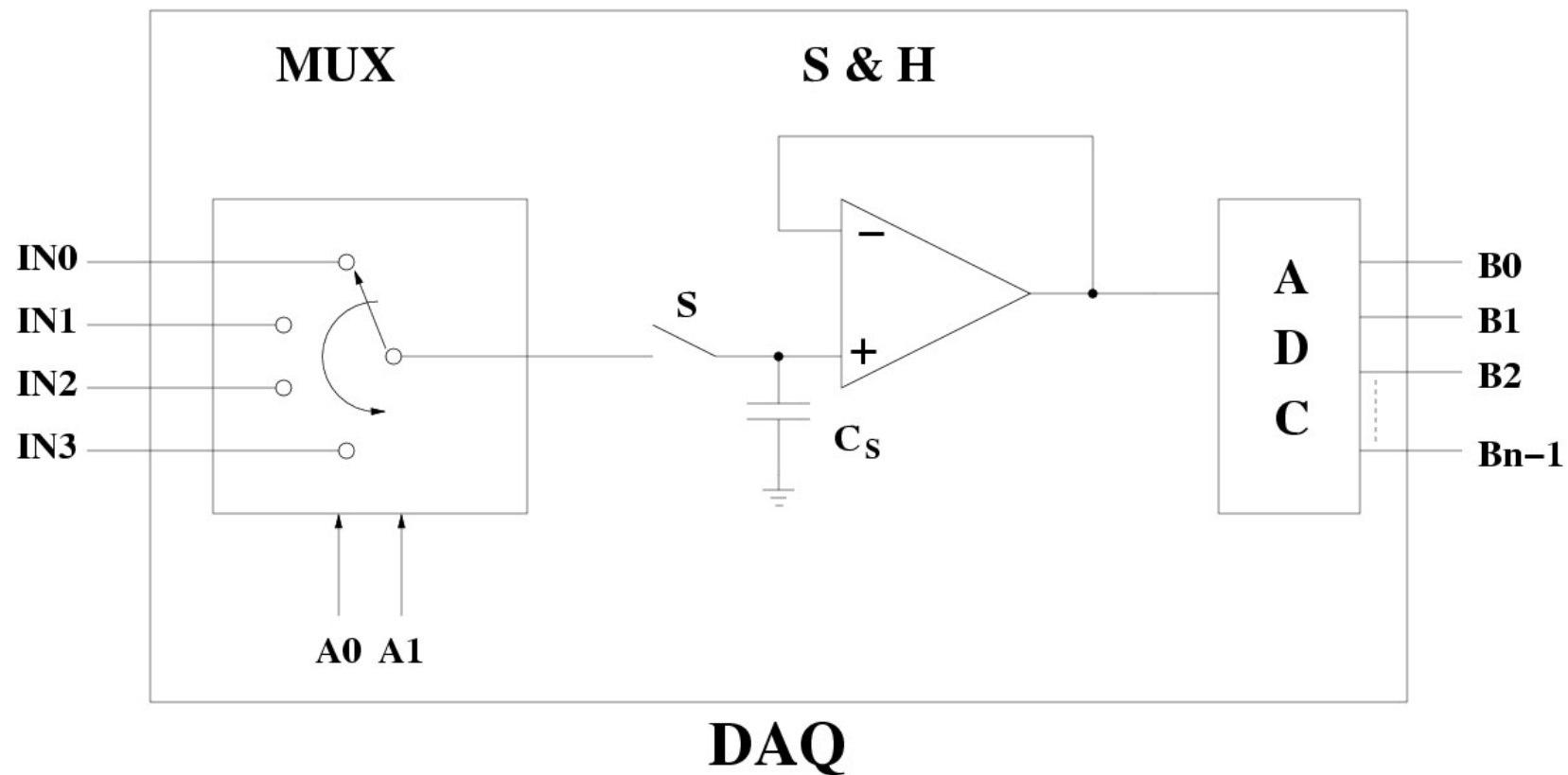
- Depending on the characteristic of the transducer and on the signal delivered by the transducer itself, we may need to perform tasks of some kind, such as
 - amplification
 - we may need to electrically isolate the transducer from the rest of the chain to avoid, for instance, high voltage transients (and to avoid damaging some components) or ground loops (which may alter the measurement results)
 - filtering (typically against noise, or to avoid aliasing, i.e., data corruption in the case of sampled signals)
 - some transducers may need to be biased to work correctly (e.g., photodiodes)



DAQ Board



Provides an interface between the conditioning circuits and the microcontroller/microprocessor/PC





Computer and Acquisition Software

- They are used to perform a few tasks, such as
 - long term data storage (on hard disks or solid state disks)
 - data processing and analysis
 - representation of data after processing
- they also provide a friendly interface between the user and the acquisition/measurement system



Data Acquisition from Measuring Instrumentation

- A particular case of a data acquisition system - the measurement result is generally provided by the instrument already in the form of a binary stream of data. The acquisition chain will include
 - a communication interface at the instrument end with the relevant firmware (the measurement operation performed by the instrument include signal conditioning and A-to-D conversion)
 - a communication interface at the computer end
 - a communication bus (ethernet, RS232, GPIB)
 - some program controlling the acquisition process on the computer



LabVIEW (National Instruments)

- LabVIEW is an environment for software development, not so different from other development environments based on more popular programming languages
- LabVIEW leverages a graphical programming language which allows the user to develop programs in the form of **block diagrams**



LabView (National Instruments)

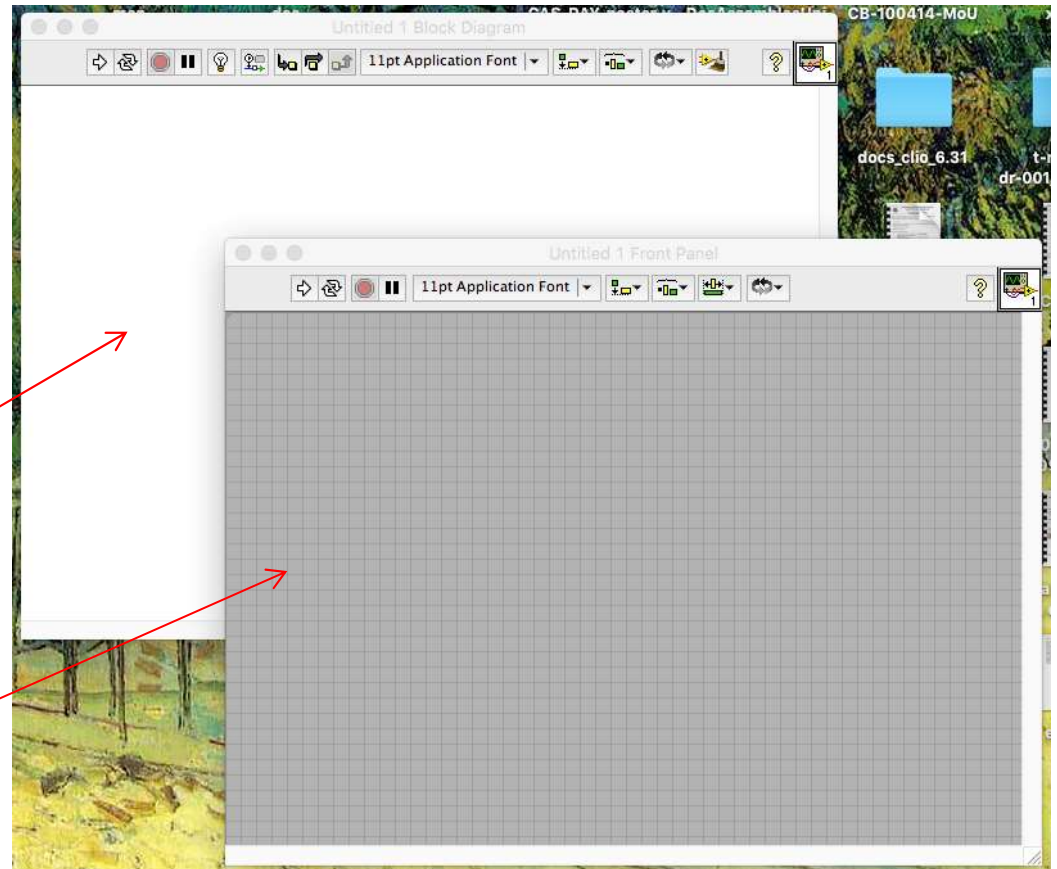
- In the LabVIEW environment we can find
 - libraries of functions for all possible programming needs
 - specific libraries for the various operating systems and platforms
 - libraries for remote control of instrumentation
 - libraries for data storage, analysis and representation
 - tools for program development and debugging
- Runs on Windows, OSX, Linux

Virtual Instrument (VI)

- Programs developed in the LabVIEW environment are called **Virtual Instruments**, or VIs. A **VI** consists of
 - an interactive user interface, or **front panel**
 - a **block diagram** which serves the same purpose as the source code in the textual programming languages

block
diagram

front
panel

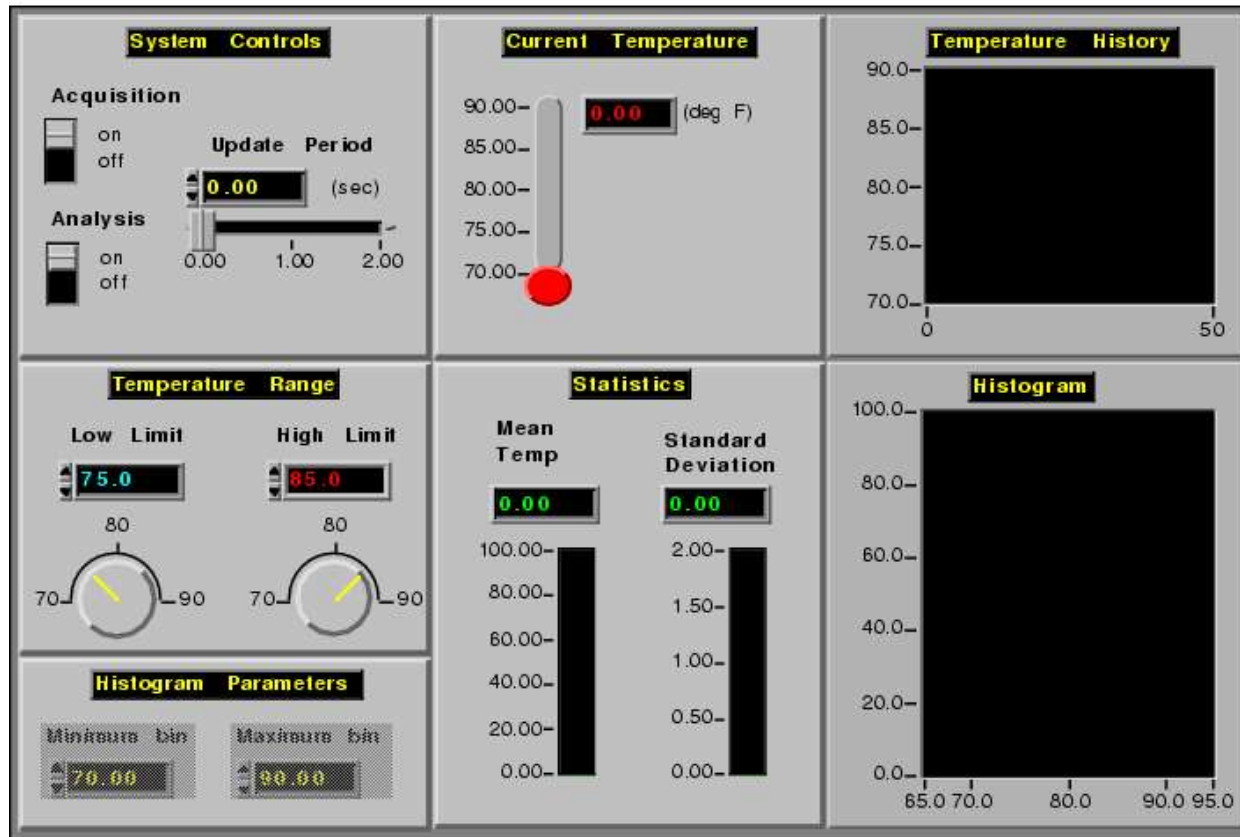




Front Panel

Temperature System Demo.vi
Last modified on 11-04-2000 at 18:39

Page 1



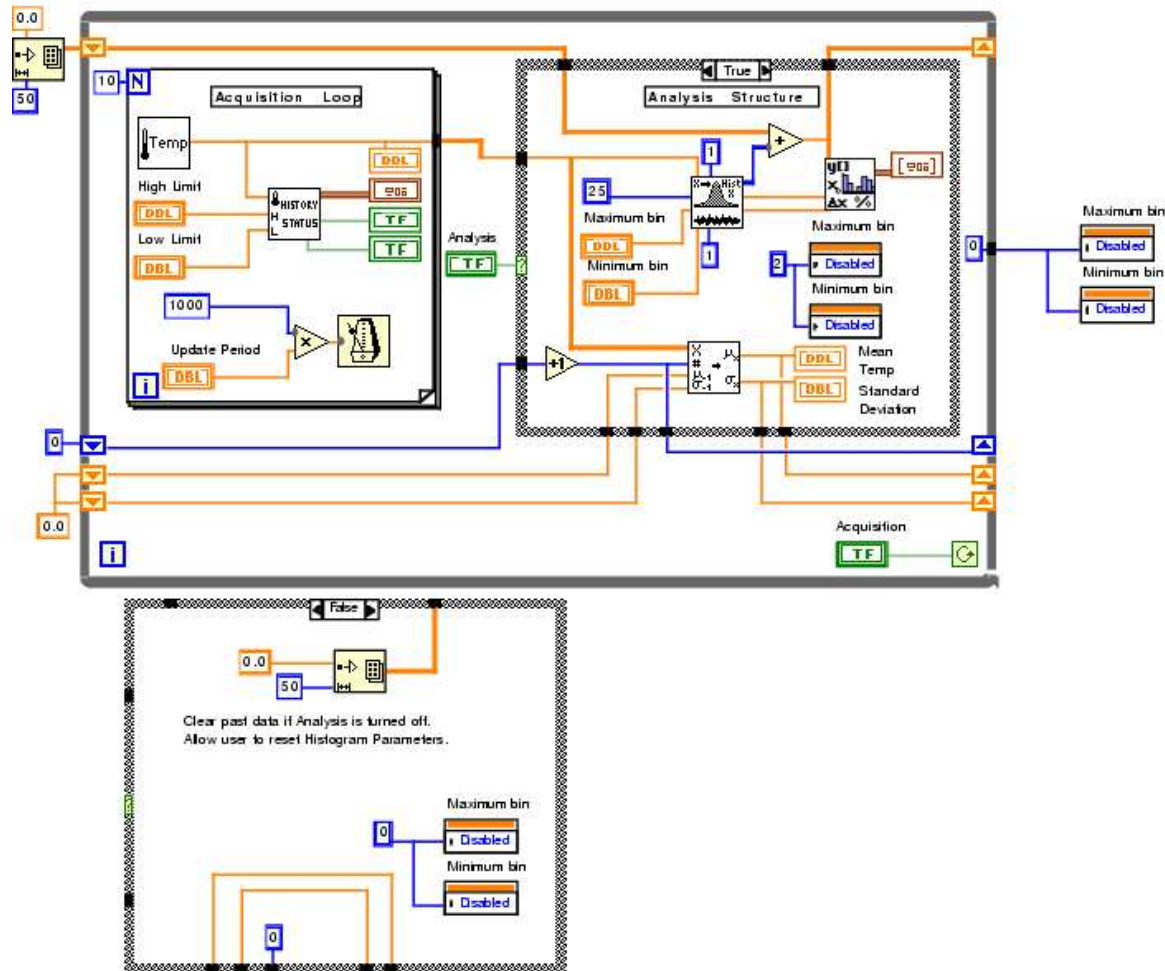
- Interactive user interface
- Resembles (or may resemble) the front panel of a real instrument, where you can find dials, keys, indicators of many kinds



Block Diagram

Temperature System Demo.vi
Last modified on 11-04-2000 at 18:15

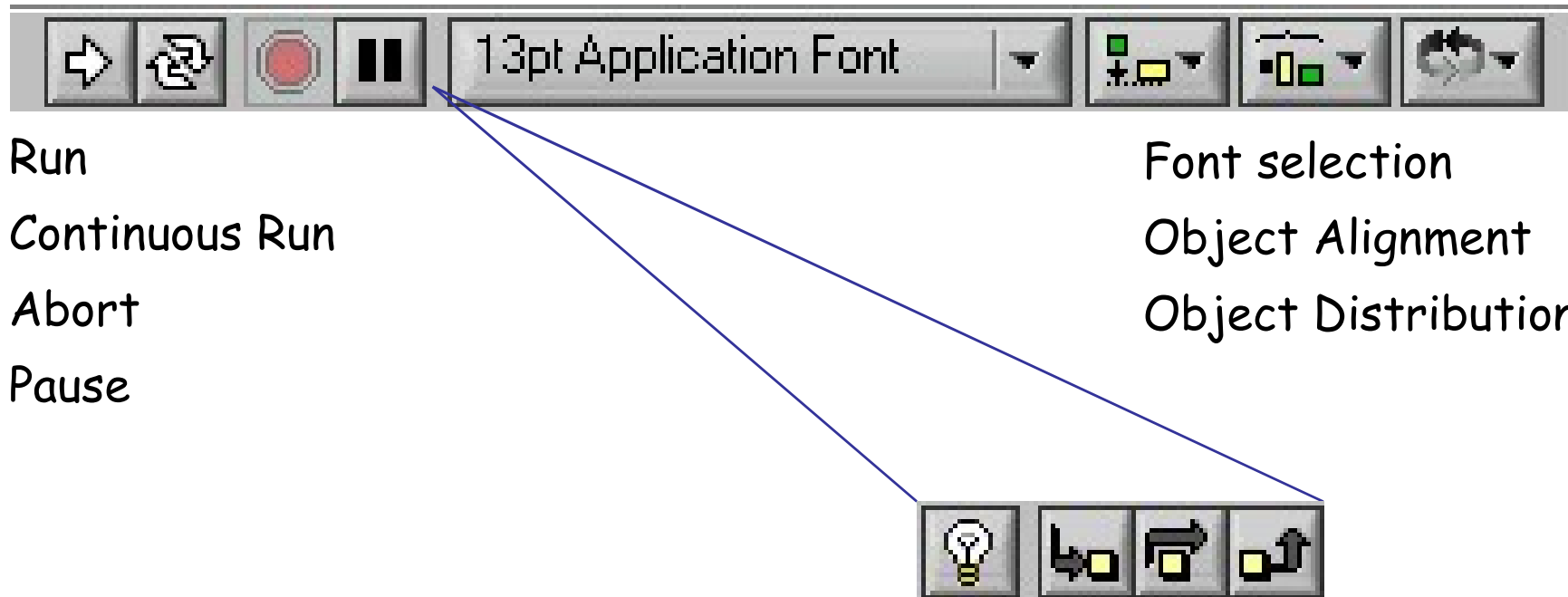
Page 1



Is the source code of the virtual instrument

Consists of blocks connected to each other, with the addition of structures governing the flow of data

Status toolbar



Other functions available in the block diagram window toolbar (debug functions)

- Highlight Execution
- Step Into (follows the execution into a subVI)
- Step Over
- Step Out



Tools Palette

- Floating palette containing editing and debugging tools



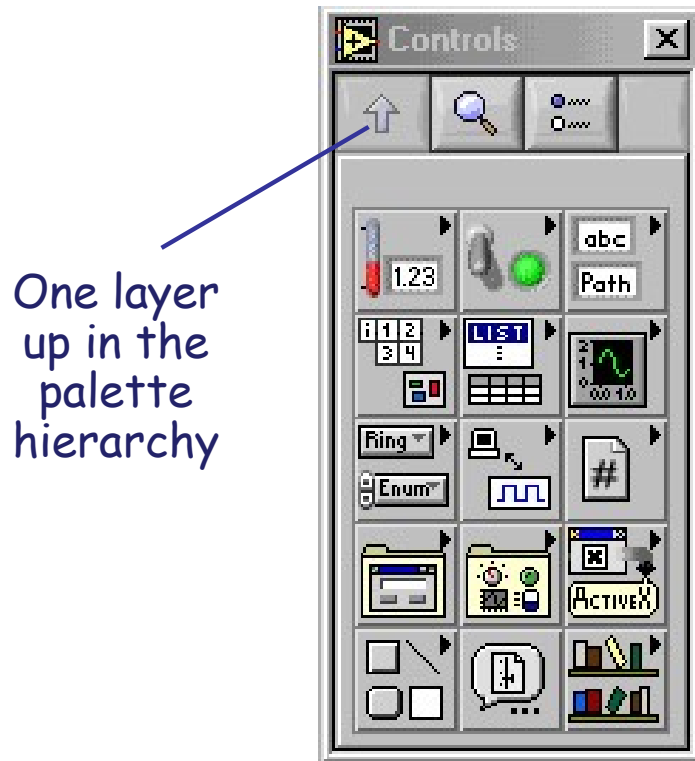
- Operate value (interaction with VI)
- Position/Size/Select
- Text Editing
- Wiring Tool
- Object Shortcut Menu
- Scroll Window
- Set/Clear Breakpoint
- Probe Data
- Get color



Control and Function Palettes

Controls Palette
(can be opened in the front panel window)

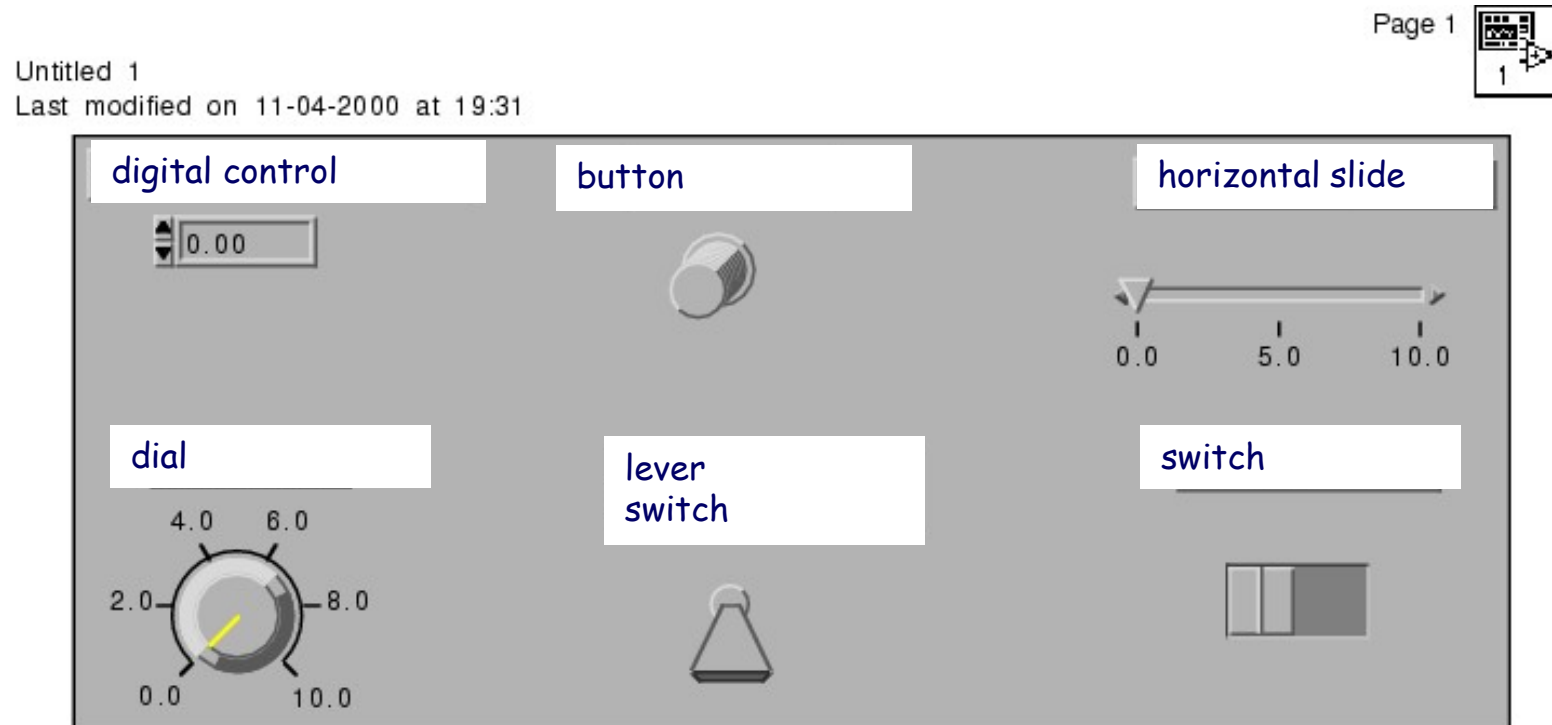
Functions Palette
(can be opened in the block diagram window)





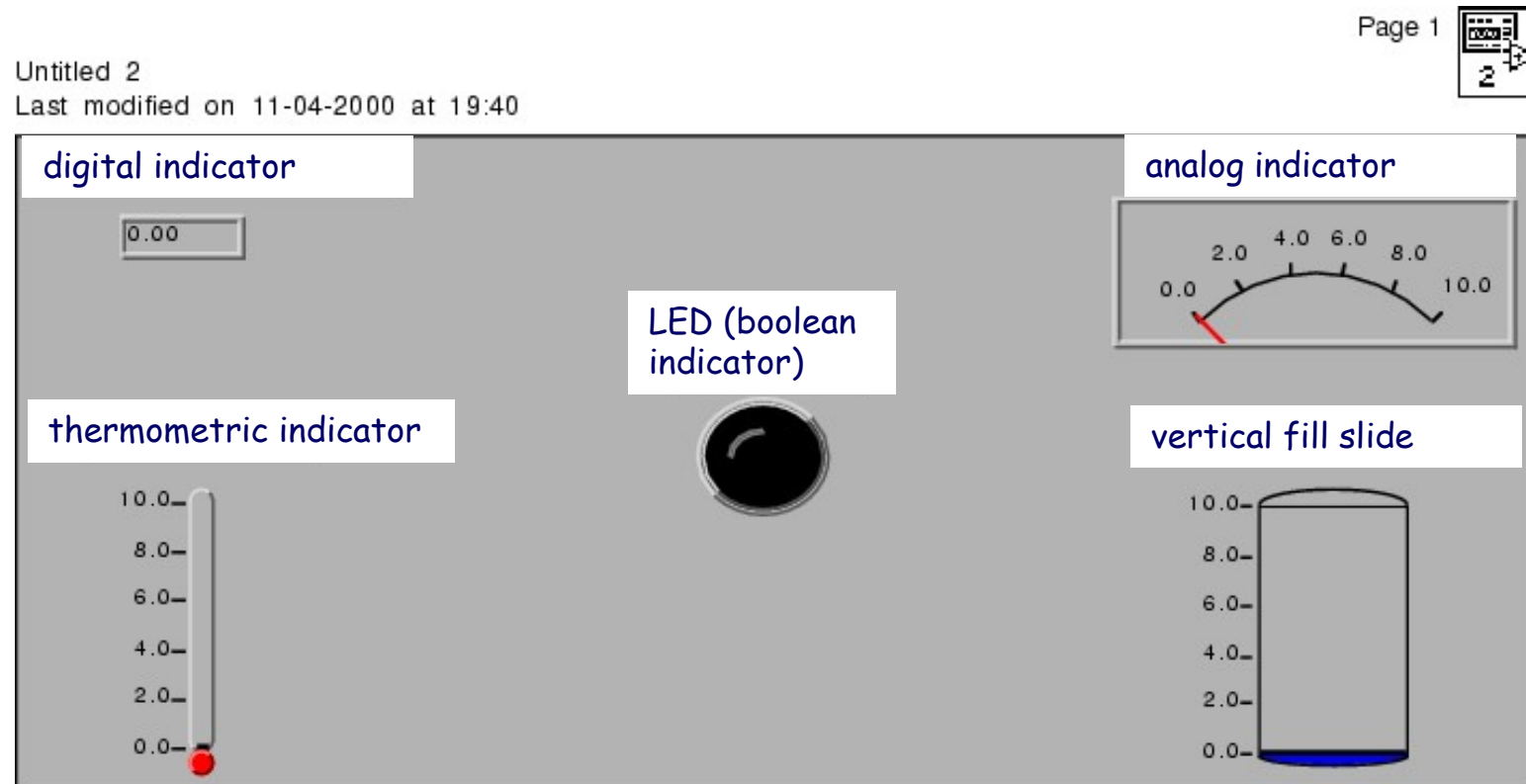
Controls

- A **Control** is an object in the front panel window which is used to feed (**input**) data into a VI



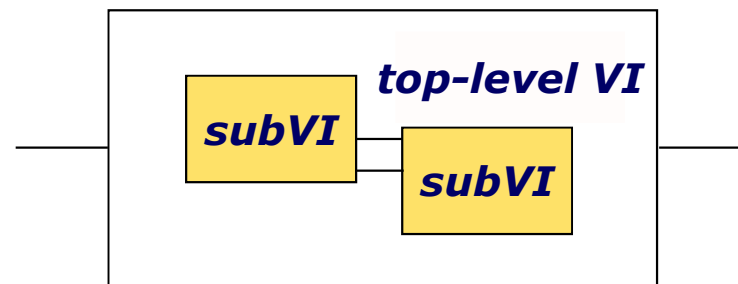
Indicator

- An **Indicator** is an object in the front panel window which is used to represent data produced (**output**) by a VI

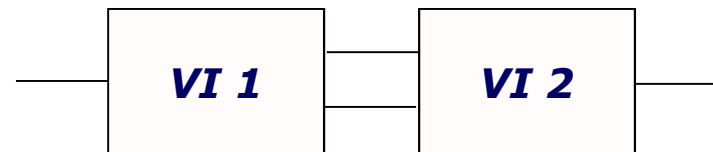


Program Structure

- A VI can be structured according to two different approaches, which can be mixed together
- **Hierarchical Structuring:** a VI can be used as a top level program (top level VI) or as a sub-program (sub-VI)



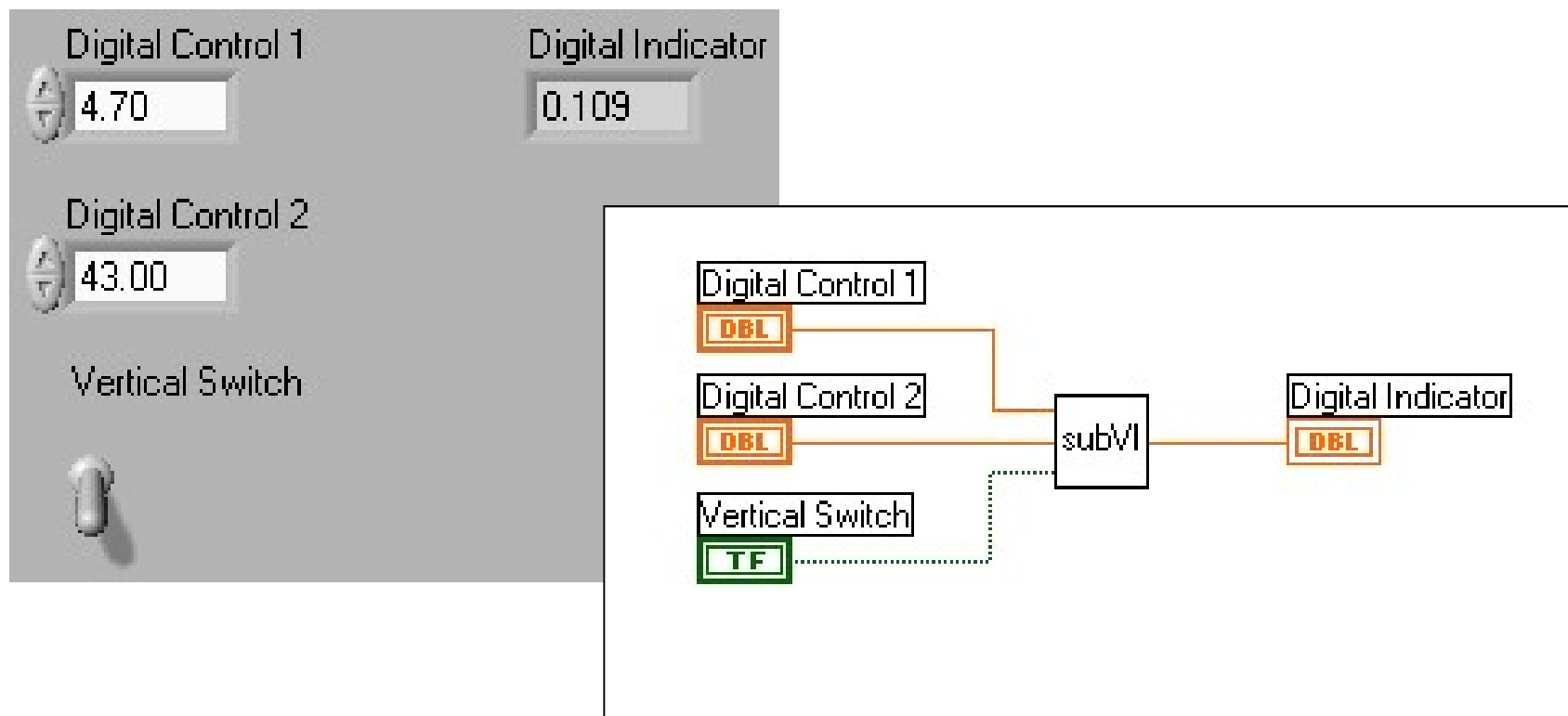
- **Sequential Structuring:** data produced by a VI can be fed into another VI cascaded to the first one





Sub-VI

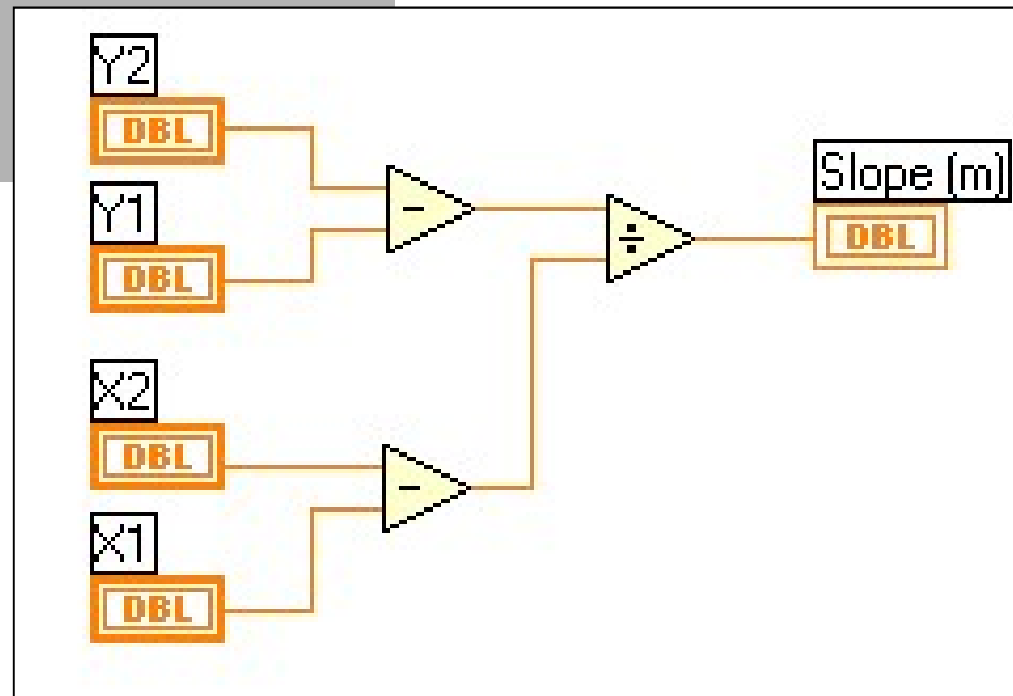
- A VI can be selected and included inside another VI (hierarchical structuring) - from the functions palette, "select a VI"
- The VI must have an icon and (input and/or output) terminals



An example: calculation of the angular coefficient of a straight line

- $Y = mX + q$
- m : slope of the straight line

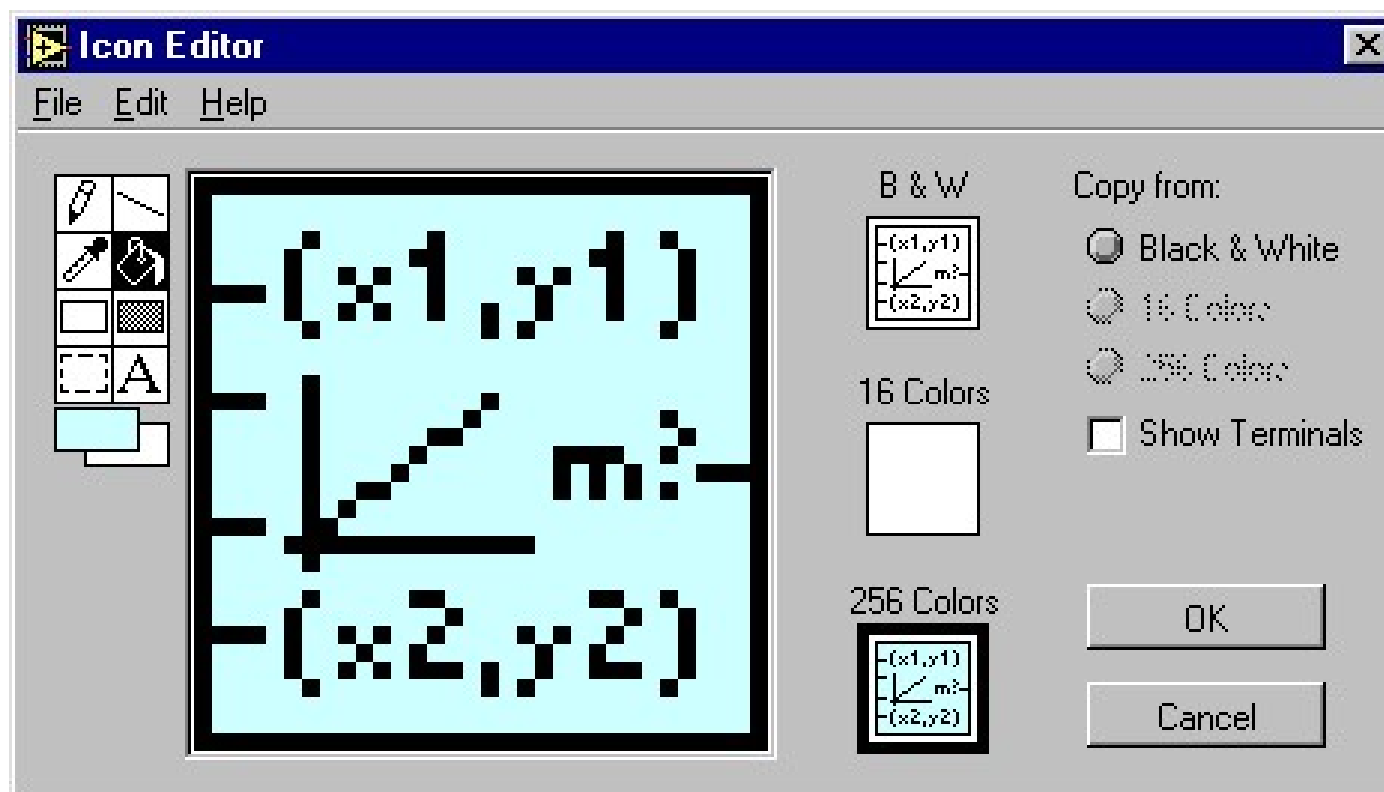
X1	Y1	Slope (m)
4.00	1.45	
X2	Y2	1.32
8.50	7.40	



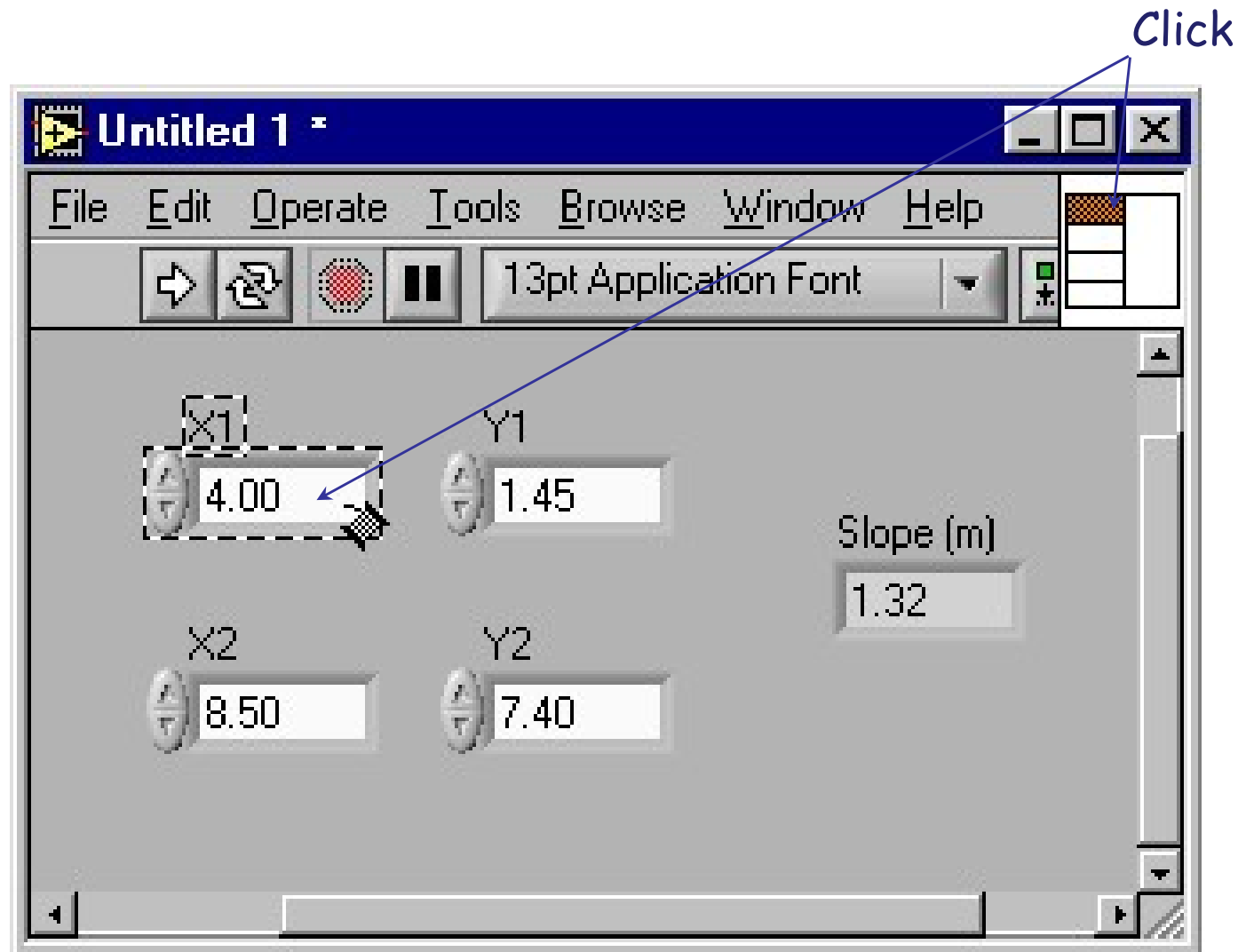


Icon Creation

- Right-click on the icon pane (either in the front panel or in the block diagram)
- Different ways to create the icon: B&W for printing, 16 or 256 colors

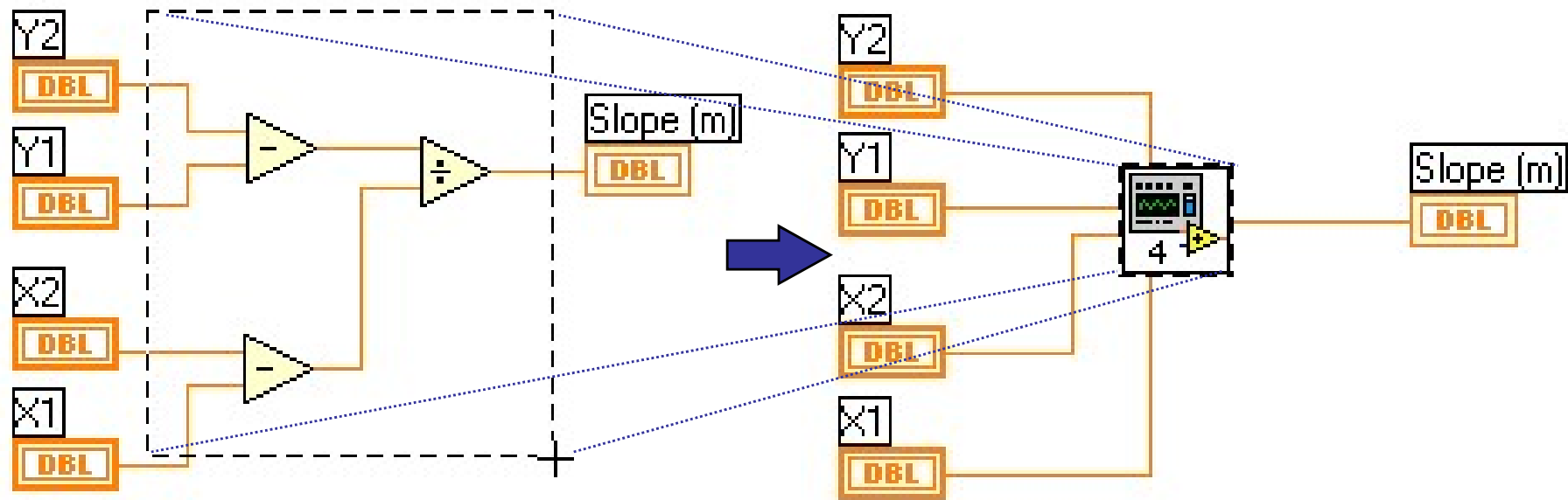


Terminal (connector) Creation



An alternative way to create a sub-VI

- Select the block diagram region including the objects you want to include in the sub-VI
- Select "Create SubVI" from the Edit menu





Context Help

Context Help (CTRL+H)

Simple or Detailed Version

Locked (Fixed)

Show optional terminals

Detailed Information



Debugging Techniques

- Look for errors



By pressing a broken run button, a list of the errors in the relevant block diagram will appear

- Highlight the data flow



By pressing the Execution Highlighting button, the path followed by the data is represented through a bubble (together with the relevant values) sliding along the wires

- Use probing tools



By right-clicking on the wire, you can select a custom probe - as an alternative, the probing tool can be selected from the tools palette and connected to a wire

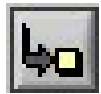
Debugging Techniques

- Breakpoints



With the breakpoint tool, in the tools palette, one can select the wire or node where the program should stop during the execution

- Step Into, Step Over and Step Out for step-by-step execution



The Step Into button enables the step-by-step execution; pressing the button after the first time makes the execution advance to the subsequent node



The Step Over button also enables the step-by-step execution and makes it possible to skip the subsequent node



The Step Out button makes it possible to leave the execution of a node



Functions

- Functions are elementary blocks of the graphical programming language
 - control structures (for and while loops, case structures, etc.)
 - numerical functions
 - logical functions
 - functions working with strings
 - functions working with vectors
 - grouping functions
 - comparison functions



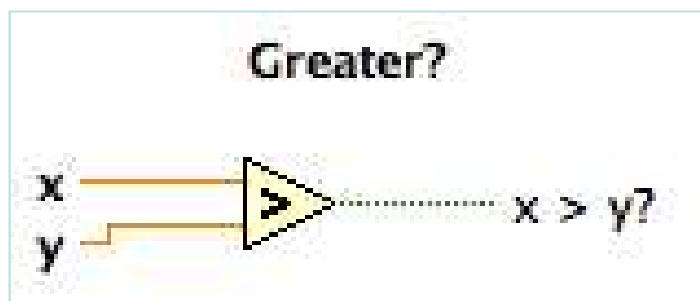
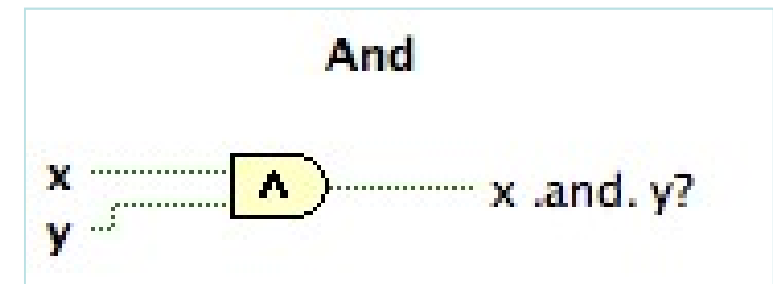
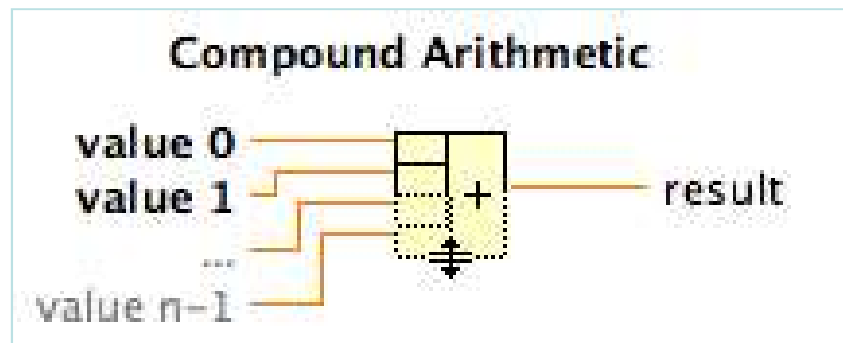
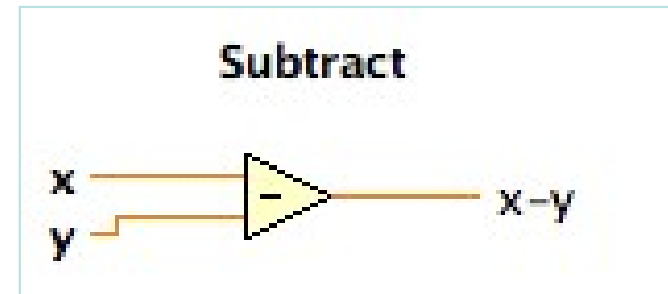
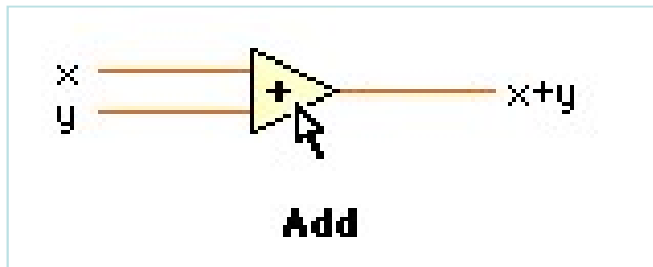


Functions

- timing, error and dialog window management functions
- file management functions
- data acquisition functions
- functions for remote communication with measuring instruments
- functions for remote communication with computers
- data analysis functions















Some examples of functions



































Connectors

- Path followed by data between two nodes - the connector color depends on the data type
 - blue for integer numbers
 - orange for floating point numbers
 - green for boolean data
 - pink for alphanumeric strings
 - the connector thickness provides information about the type of data (scalar, 1D or 2D vector)

	Scalar	1D Array	2D Array	
Numeric	 	 	 	Orange(floating point) Blue (integer)
Boolean				Green
String				Pink

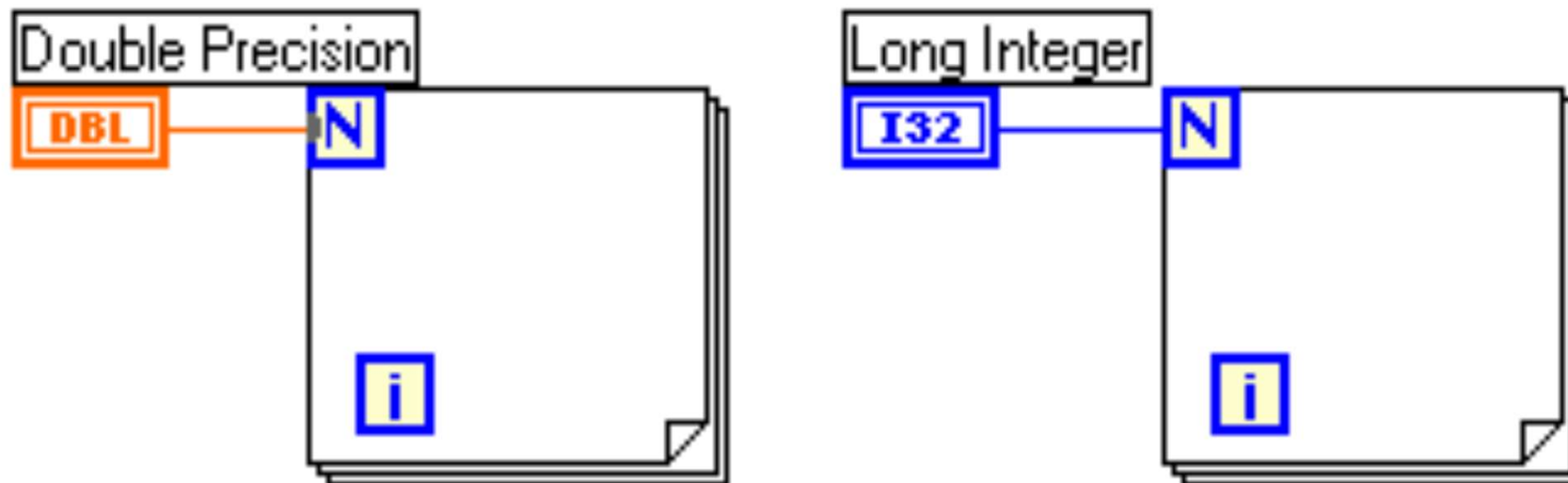


Data Type

Control	Indicator	Data Type	Color
		Single-precision floating-point numeric	Orange
		Double-precision floating-point numeric	Orange
		Extended-precision floating-point numeric	Orange
		Complex single-precision floating-point numeric	Orange
		Complex double-precision floating-point numeric	Orange
		Complex extended-precision floating-point numeric	Orange
		Signed 8-bit integer numeric	Blue
		Signed 16-bit integer numeric	Blue
		Signed 32-bit integer numeric	Blue
		Unsigned 8-bit integer numeric	Blue
		Unsigned 16-bit integer numeric	Blue
		Unsigned 32-bit integer numeric	Blue
		Enumerated type	Blue
		Boolean	Green
		String	Pink

Numerical Data Conversion

- Numerical data are, by default, double precision floating point numbers (8 bytes) or long integer numbers (4 bytes)
- LabVIEW automatically converts one data type into the other (CAST operation)
- A grey dot indicates an implicit CAST operation



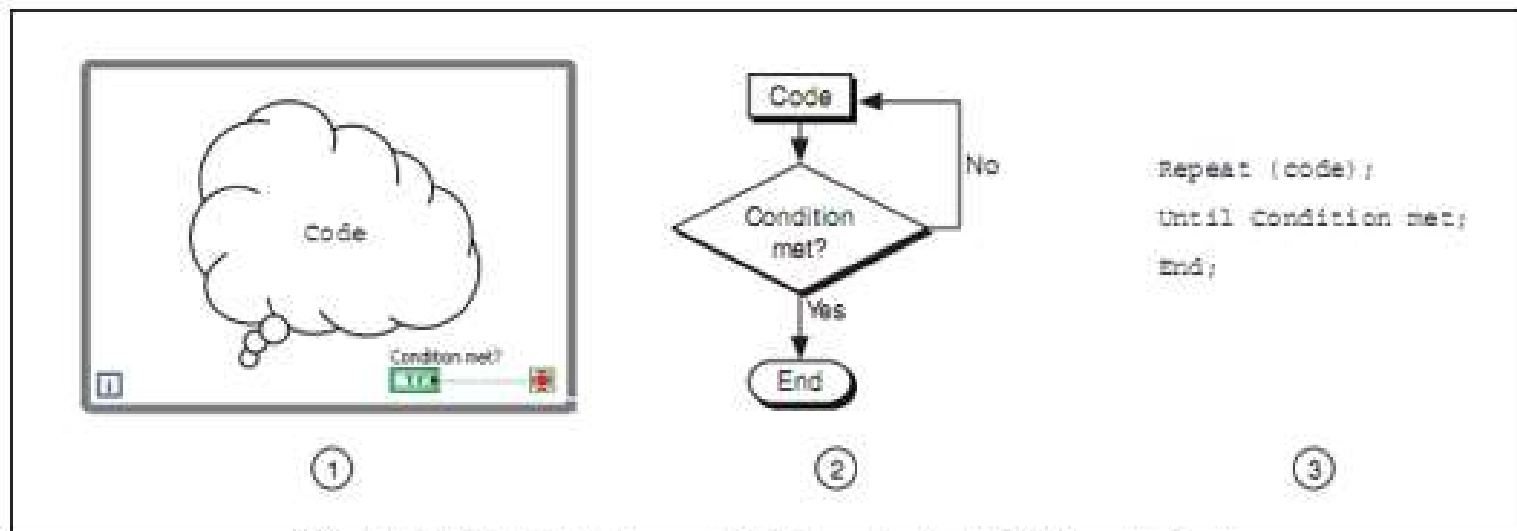


Control Structures

- Control structures let the programmer control the execution of the source code and the data flow
- The most common control structures are the **While Loop**, the **For Loop** (used to execute many times the same part of the source code, according to some conditional expression) and the **Case Structure** (used to execute one among different code parts again depending on some conditional expression)
- A more complete list of control structures includes
 - While Loop
 - For Loop
 - Case Structure
 - Sequence Structure
 - Formula Node

While Loop

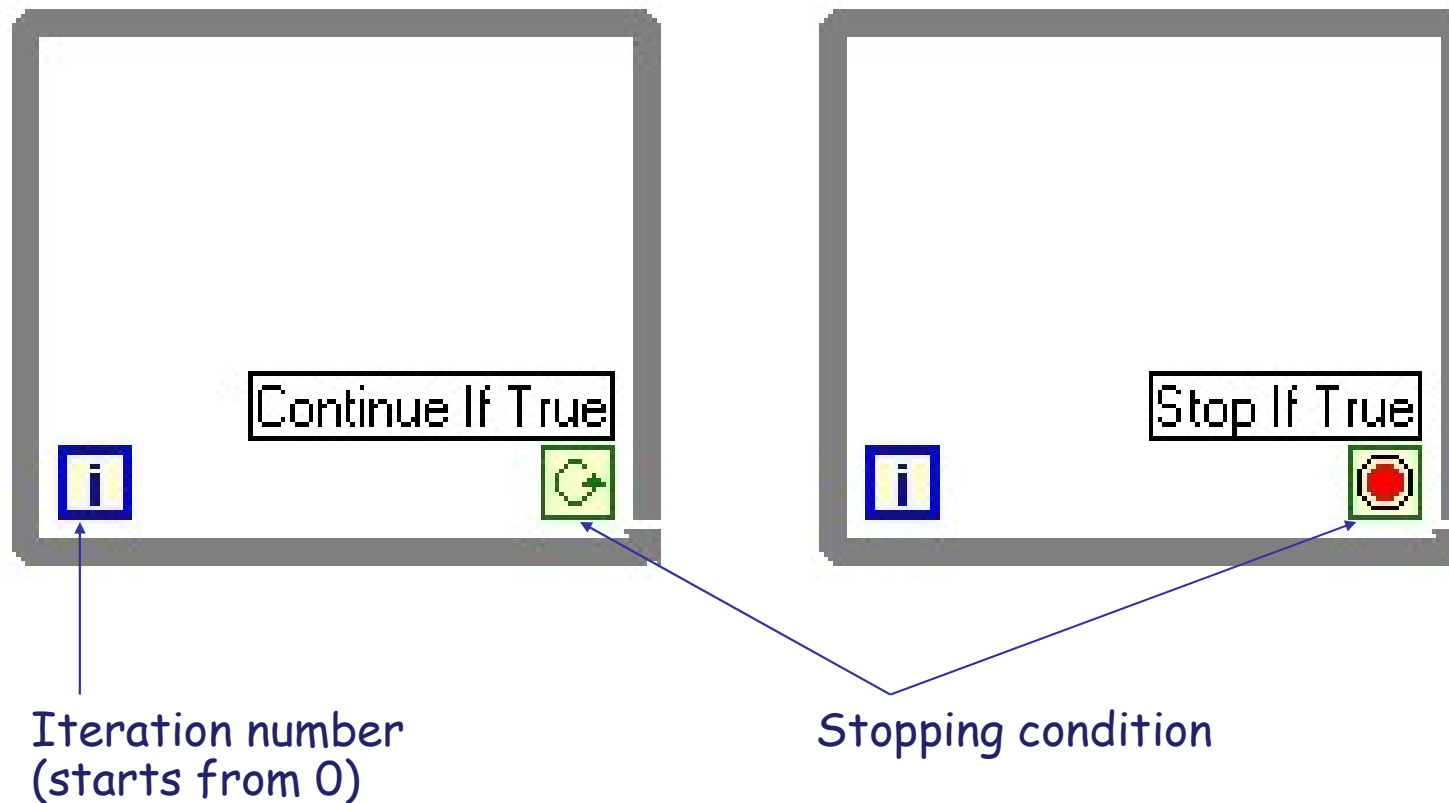
- Like a do loop or a repeat-until loop in a textual programming language, a **While Loop** executes part of a block diagram until some condition is verified (or ceases to be verified)



(1) LabVIEW While Loop | (2) Flowchart | (3) Pseudo Code

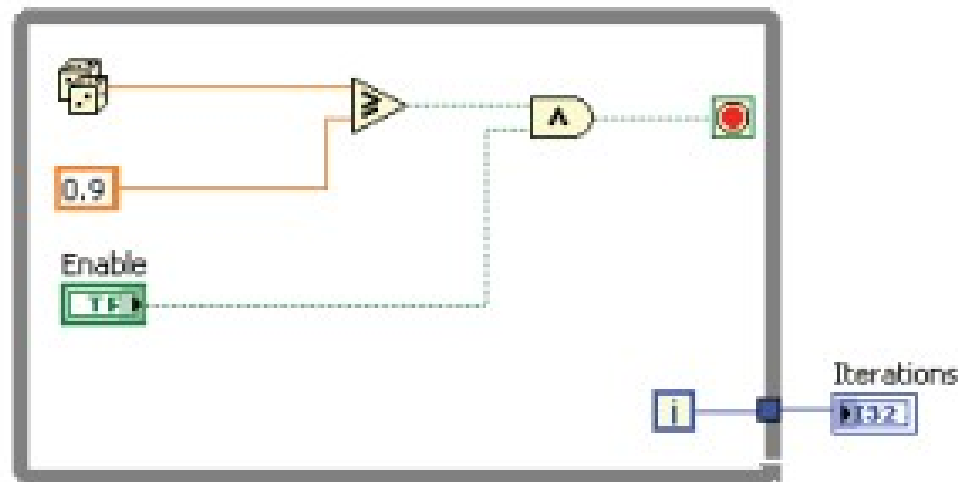
Stopping Condition for the While Loop

By right-clicking on the conditional terminal one can choose when the loop execution should stop (stop if the condition is true or if it is false)



Structure Tunneling

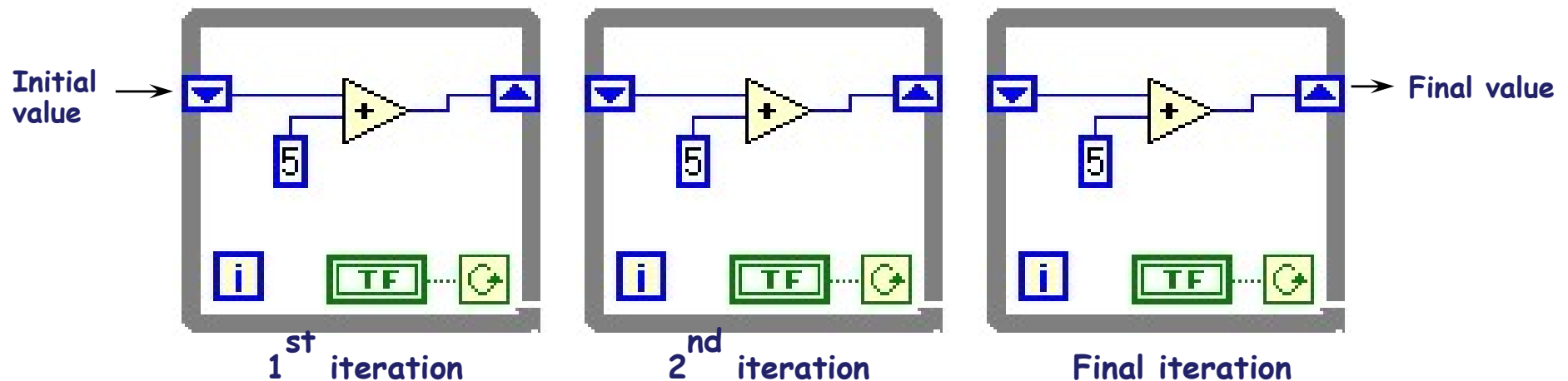
- Data go in and out of the structures through **tunnels**
 - A tunnel appears as a full (disabled indexing, scalar data) or empty (enabled indexing, vectorial data) square on the edge of the While Loop (or of other structures)
 - The square has the same color as the type of data connected to the tunnel
 - Data flows out of the loop when the loop stops
 - When data are provided to a loop through a tunnel, the loop starts only after the data have arrived to the tunnel





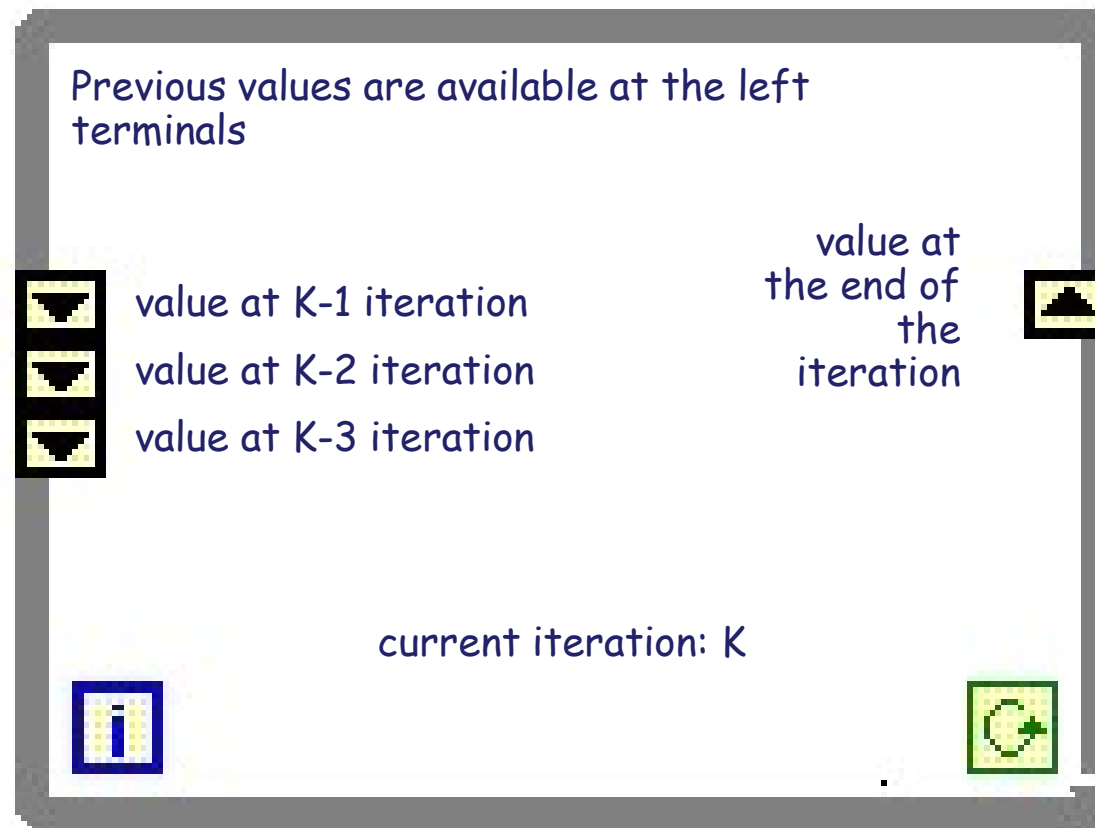
Shift Registers

- Are available at the left and right sides of loop structures
- To add a pair of shift registers, right-click on the structure edge and select "Add Shift Register"
- The right terminal of the shift register pair preserves the value at the end of the current iteration
- The left terminal returns the same value to the subsequent iteration



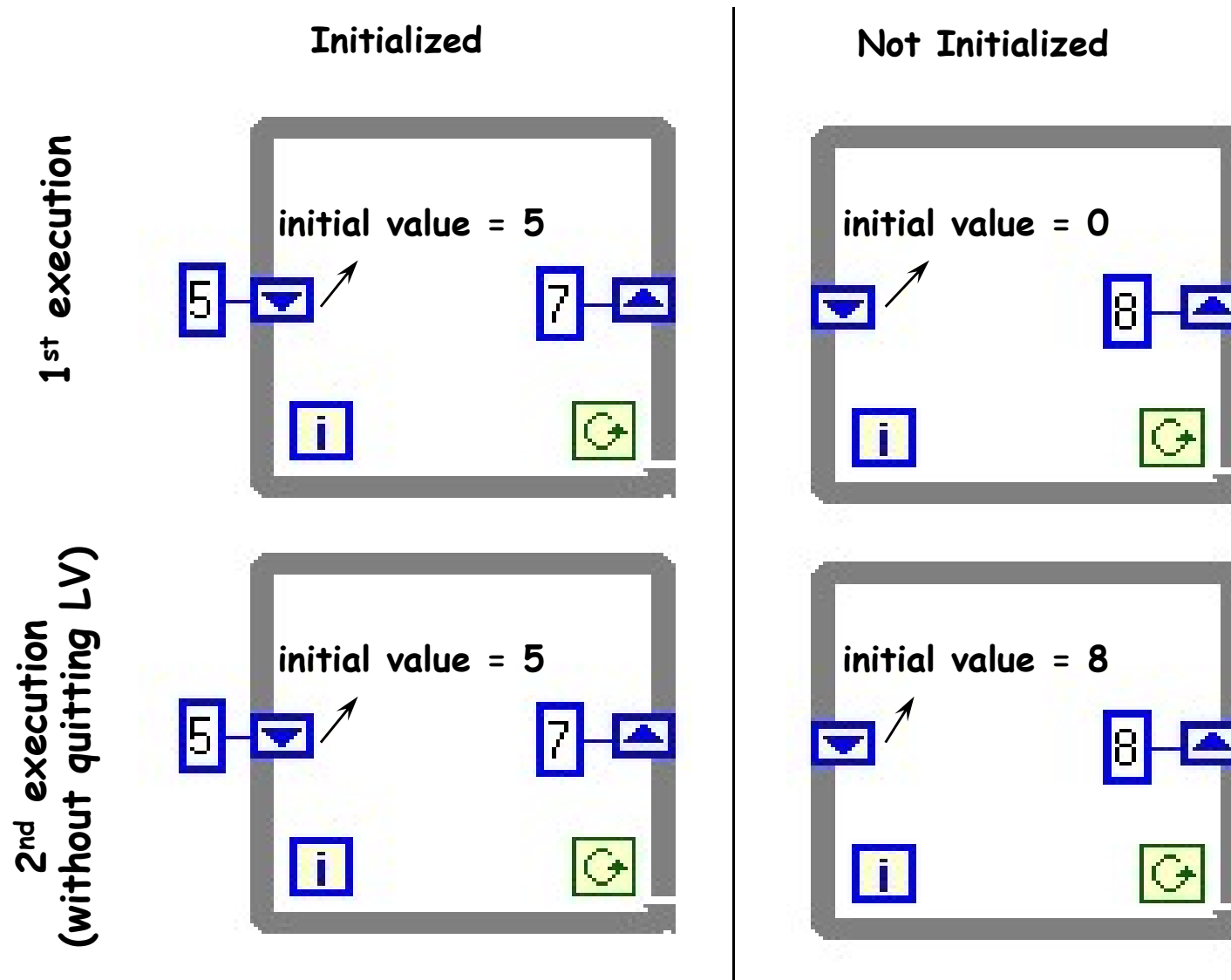
Shift Registers

right-click on
the left
terminal to add
new elements



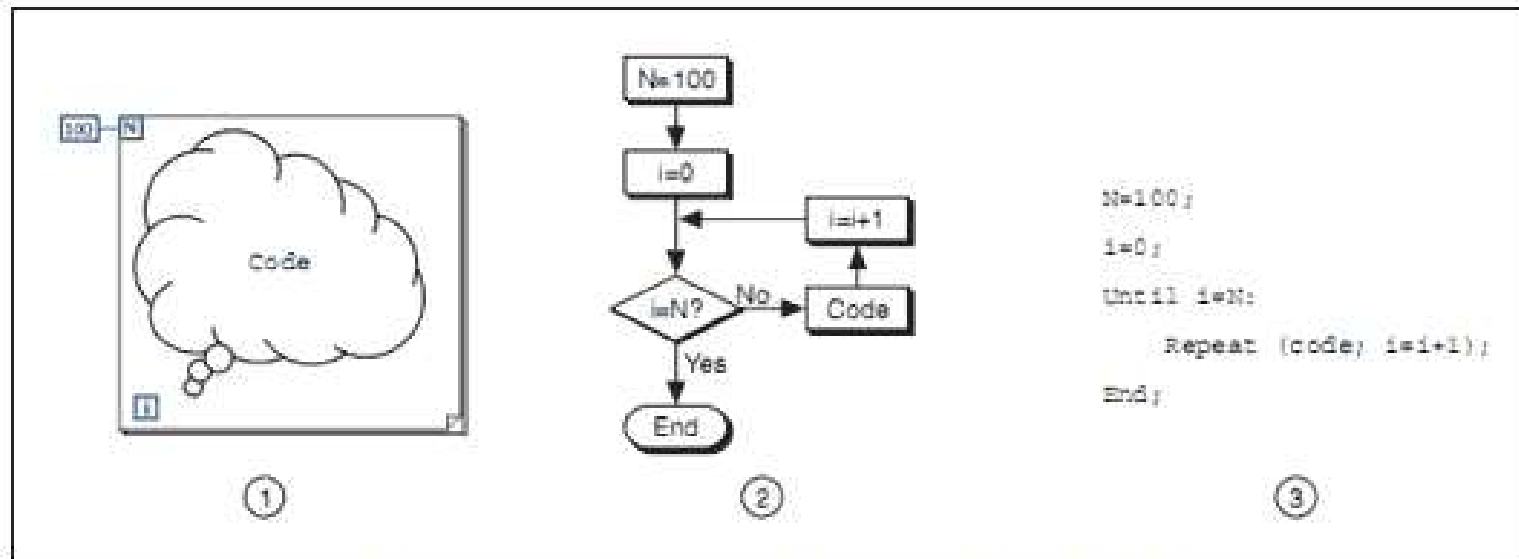
right-click on
the edge to add
a new shift
register

Shift Register Initialization



For Loop

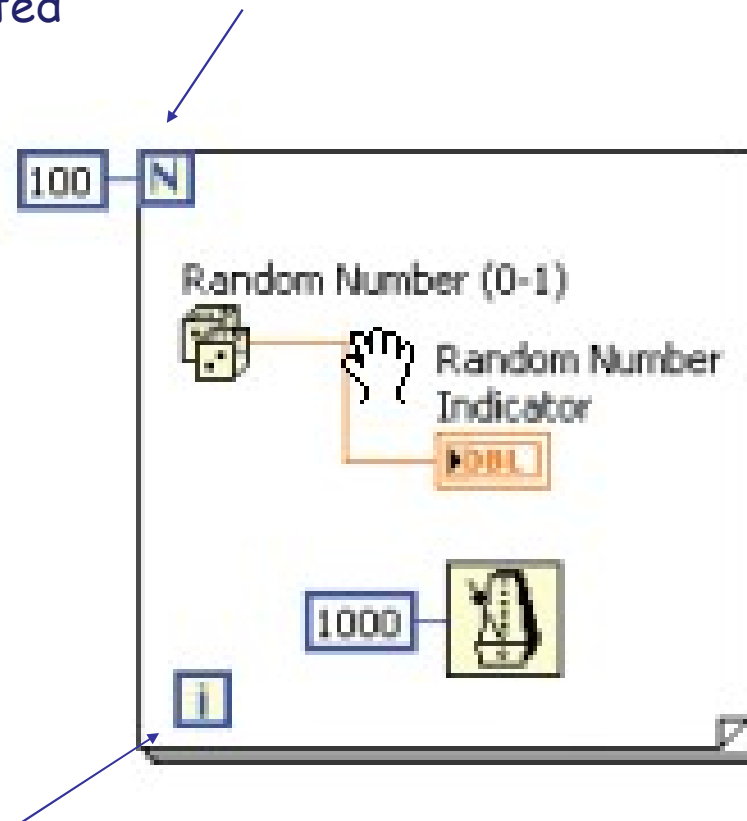
- A **For Loop** executes part of a block diagram a predetermined number of times



(1) LabVIEW For Loop | (2) Flowchart | (3) Pseudo Code

For Loop

the **count terminal** is an input terminal of a For Loop indicating how many times the sub-diagram has to be sequentially executed



the iteration terminal is an output terminal providing the number of completed iterations



Loop Timing

- When a loop completes an iteration, it immediately starts executing the subsequent one, unless a stopping condition is found
- Frequently one needs to control the iteration frequency or its timing
 - For example, suppose you are acquiring some data and you want to acquire them every 10 seconds → some iteration timing method is needed such that acquisitions take place every 10 seconds
 - Even if you do not need to control the acquisition frequency, you may need to provide the processor with the time required to perform some other tasks, such as user interface management

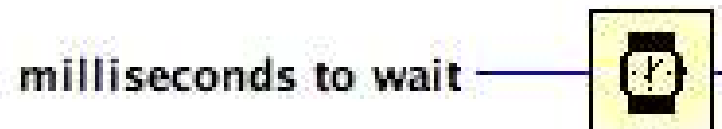
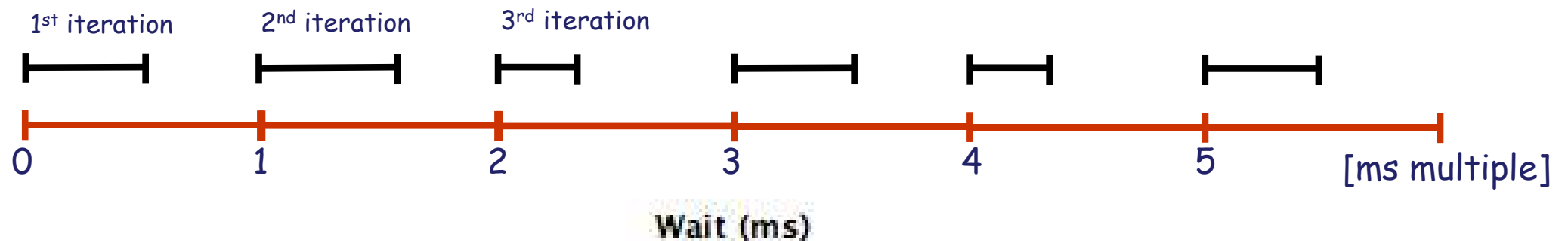




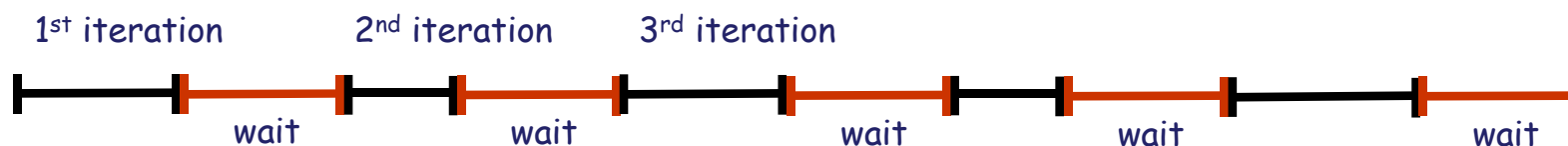
Loop Timing



Waits until the timer content is a multiple of "millisecond multiple" before starting an iteration - generally used to synchronize the loop execution with the system clock



Waits for the specified number of milliseconds before starting an iteration

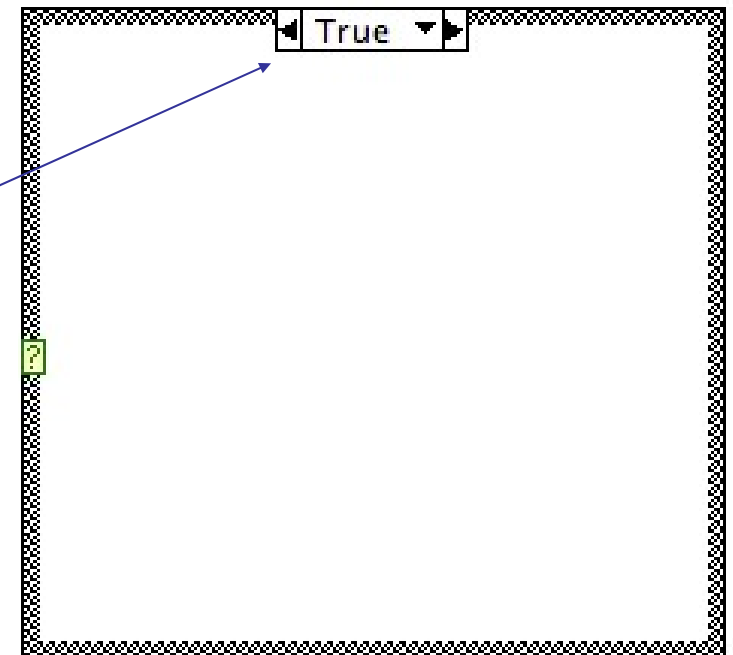


Case Structure

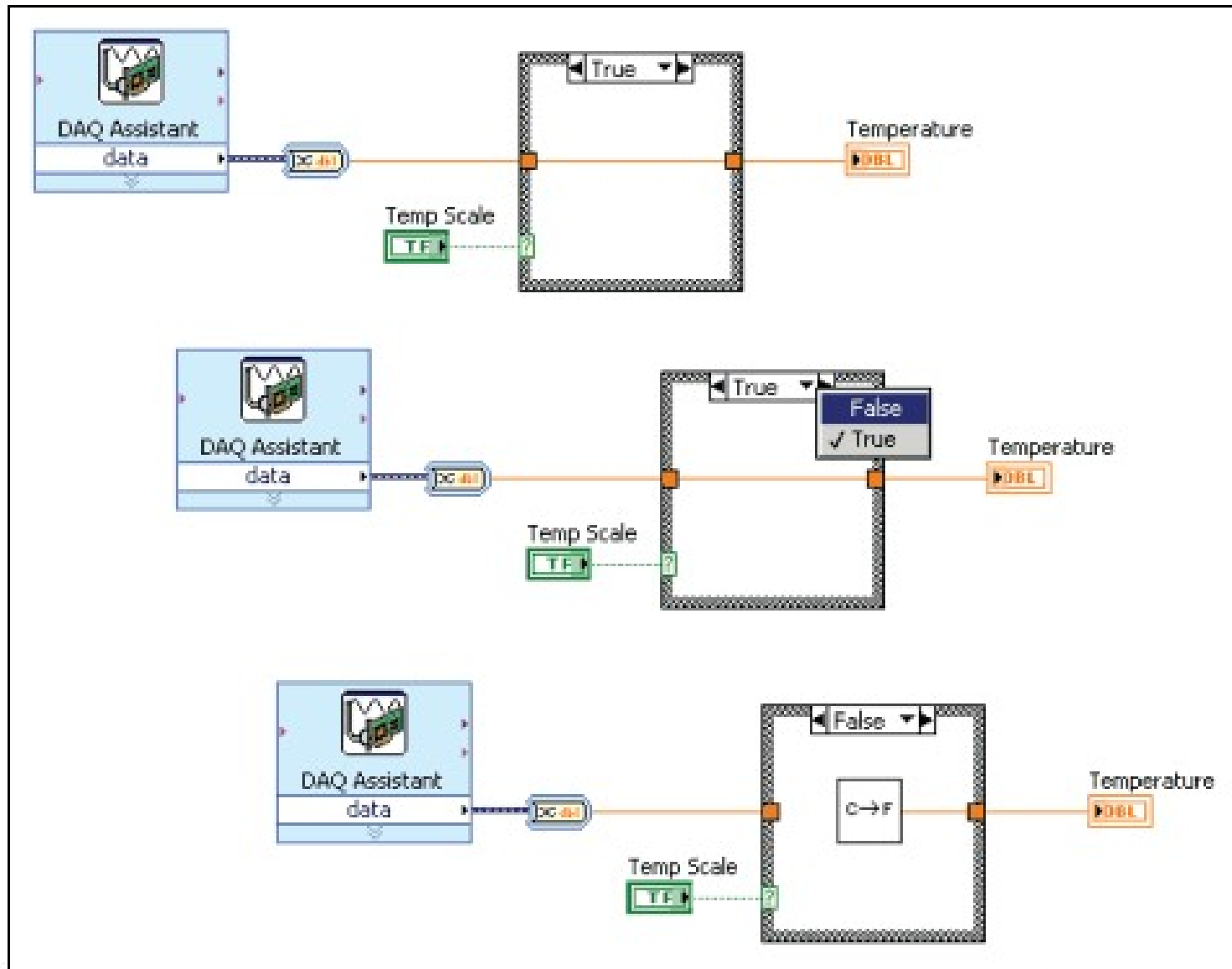
- A **Case Structure** has two or more sub-diagrams, or cases (conditions) - similar to switch instructions or if-then-else structures in textual programming languages
- one sub-diagram at a time is visible and one single case at a time is executed
- the input value is used to select the sub-diagram that is to be executed

label of the case selector - contains the name of the selector value corresponding to the case and two arrows to go from one case to the others

an input value, or selector, has to be connected to the selector terminal to select the case to be executed



Case Structure: an example

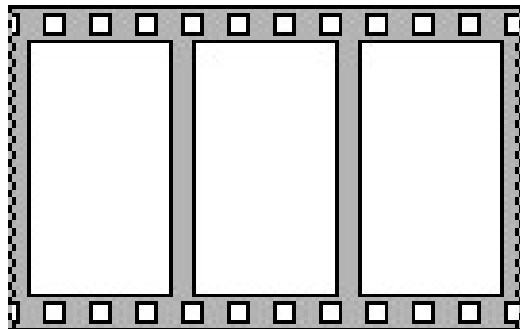




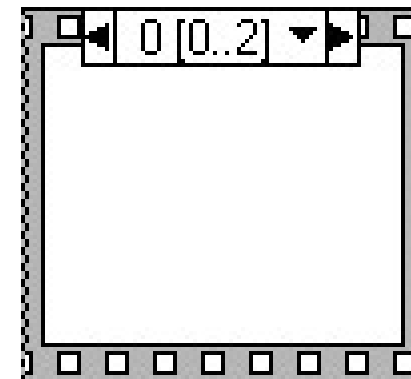
Sequence Structure

- A **Sequence Structure** executes, following a predetermined sequence, the sub-diagrams included in a series of frames, either from left to right (flat sequence) or from frame 0 to frame N-1 (sequence of N stacked frames)

Flat sequence structure

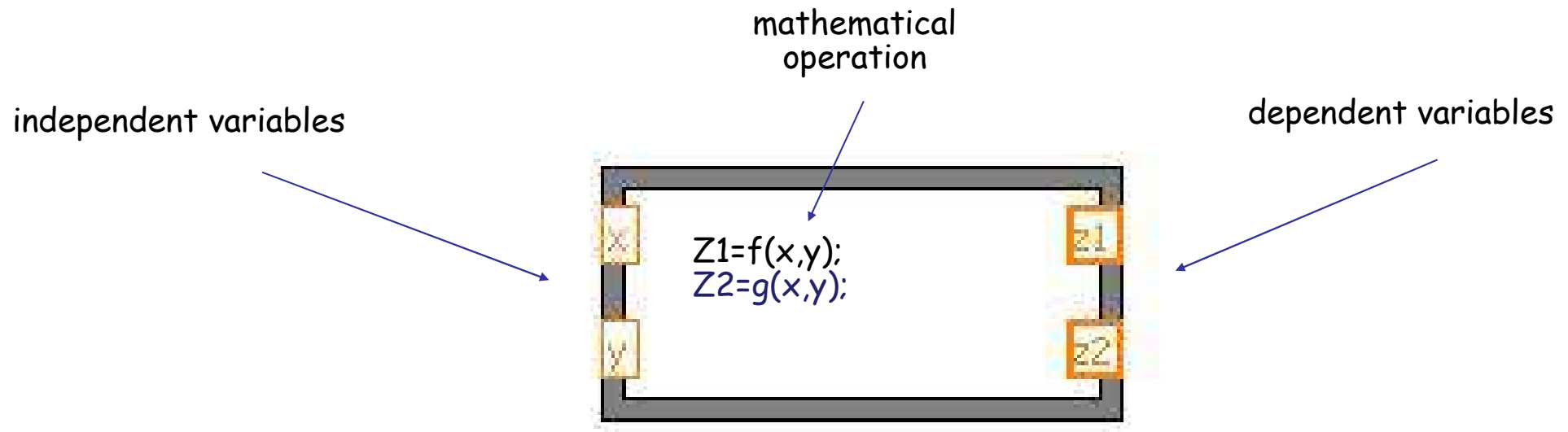


Stacked sequence structure



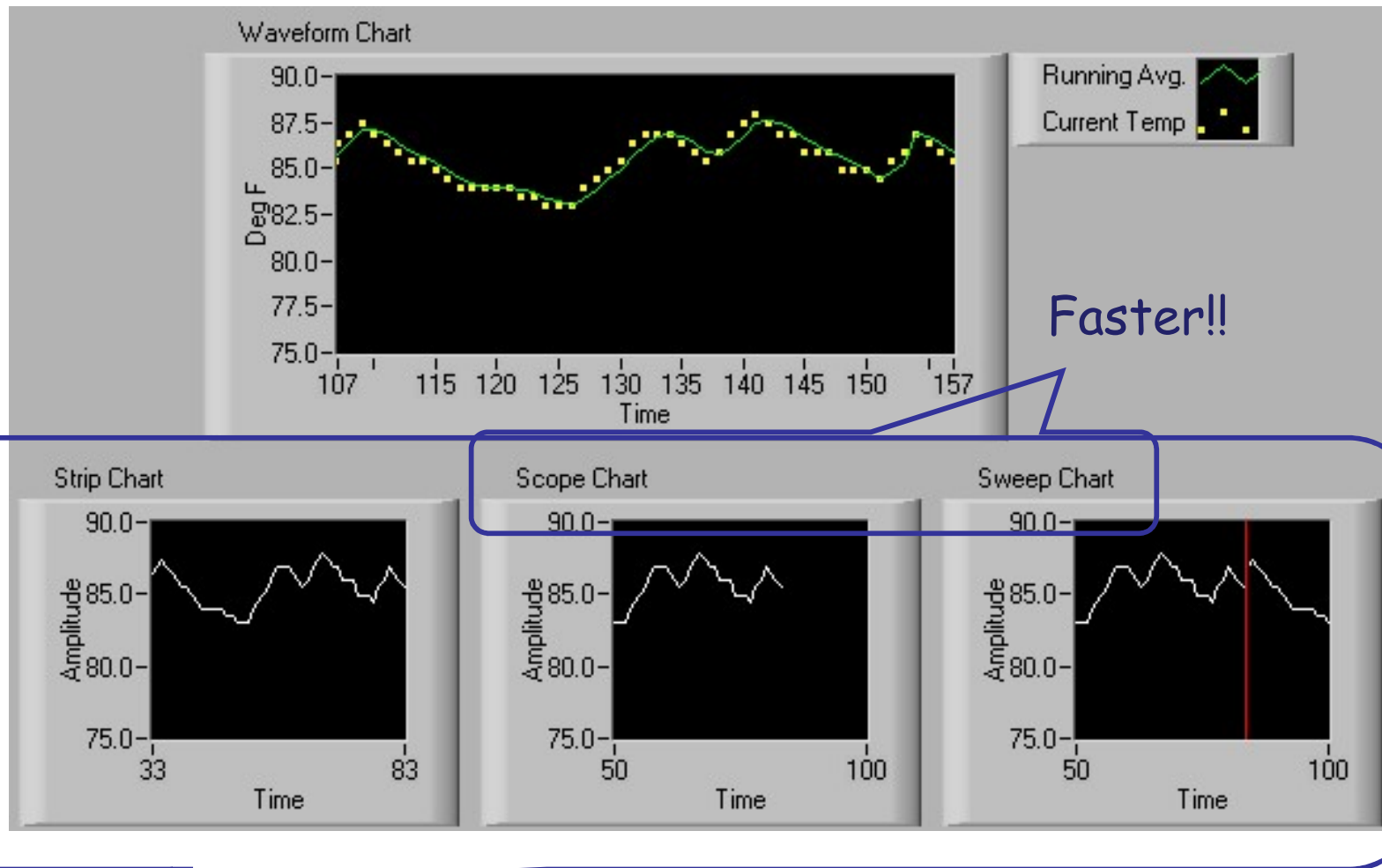
Formula Node

- The Formula Node can be used to execute complex mathematical operations by using the syntactic structure of the C language (do not forget the final ";")



Waveform Chart

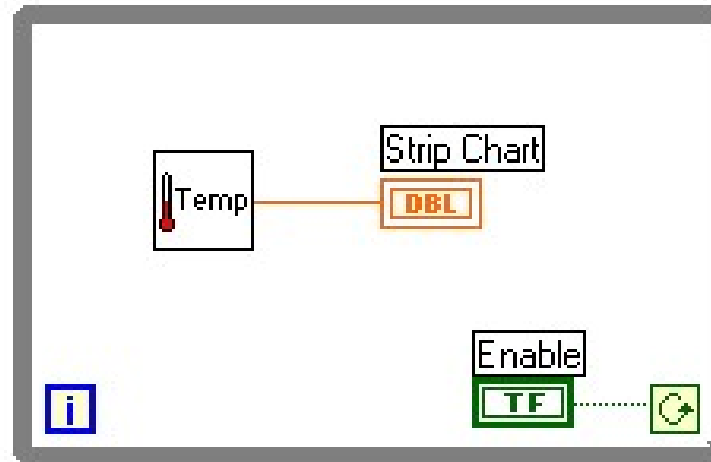
Can be found in the Controls >> Graph sub-palette



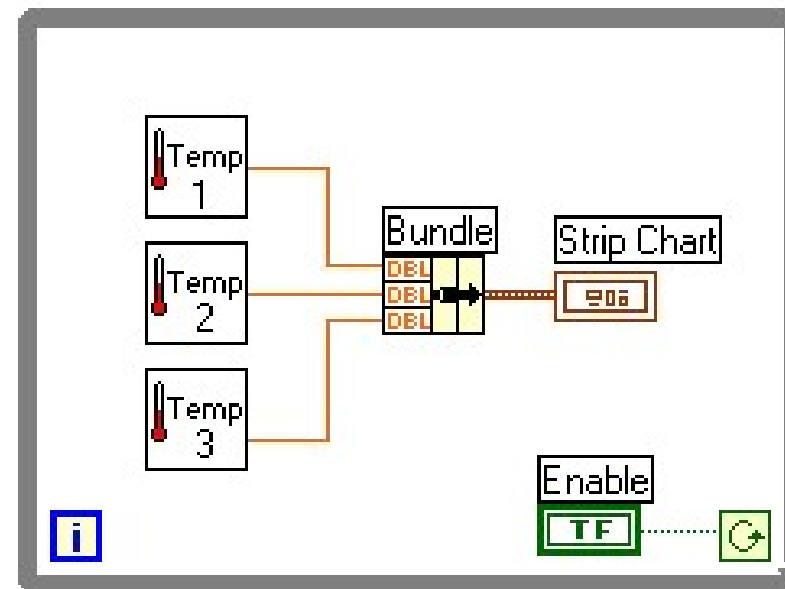
Three scrolling modes: right-click >> Advanced >> Update Mode

Input Data for a Waveform Chart

- Single-Plot Chart



- Multiple-Plot Chart
(Function palette
>> Bundle)





Exercises

- Using a **case structure**, create a LabVIEW VI which, starting from an input temperature (introduced through a control and expressed in degree Celsius) let you choose the way to represent it (through one single indicator) either in degree Celsius or Fahrenheit ($C = (F - 32) * 5/9$).
- Create a LabVIEW VI to solve second degree equations (use the **formula node structure**). The virtual instrument should also indicate the case of a negative discriminant ($b^2 - 4ac$) by switching on a LED (boolean indicator).
- Create a LabVIEW VI to sum the first N integer numbers ($1 + 2 + 3 + 4 + \dots + N - 1 + N$, use a **FOR loop** with **shift registers**). Compute also $N(N+1)/2$ in the same VI and verify that the final result is the same as the one you get with the FOR loop.