INTRODUCTION TO LABVIEW

<u>2nd Year Microprocessors Laboratory</u>

2012-2013

<u>INTRODUCTION</u>

For the first afternoon in the lab you will learn to program using LabVIEW. This handout is designed to give you an introduction to this programming language, and to using it for computer control of experiments. It is designed to teach you the basics which you'll need for the 2nd year Microprocessors Laboratory.

Some useful instructions;

Keep ***brief*** notes in your lab book; the places where we expect notes or an answer to a question in this handout should be clear.

To print your LabVIEW programs;

You can either print screen, or (better method) choose *File » Print » Choose file » Icon, description, panel & diagram » RTF file → Print,* which will give you a word-readble file with figures.

Learning outcomes;
For successful completion of this introduction to LabVIEW (1 afternoon) you will be able to

- Demonstrate knowledge of LabVIEW by
    - performing simple mathematical calculations
    - plotting, analysing and saving data
    - controlling simple experiments using a data acquisition board.

<u>IMPORTANT SAFETY INFORMATION</u>

1. The box labelled Servo Controller Interface is the interface between the computer and the experiments. **DO NOT** connect or disconnect the cables between this box and the computer.
2. When using a computer, it is important to take regular breaks to rest the eyes and stretch.

1. INTRODUCTION TO LABVIEW

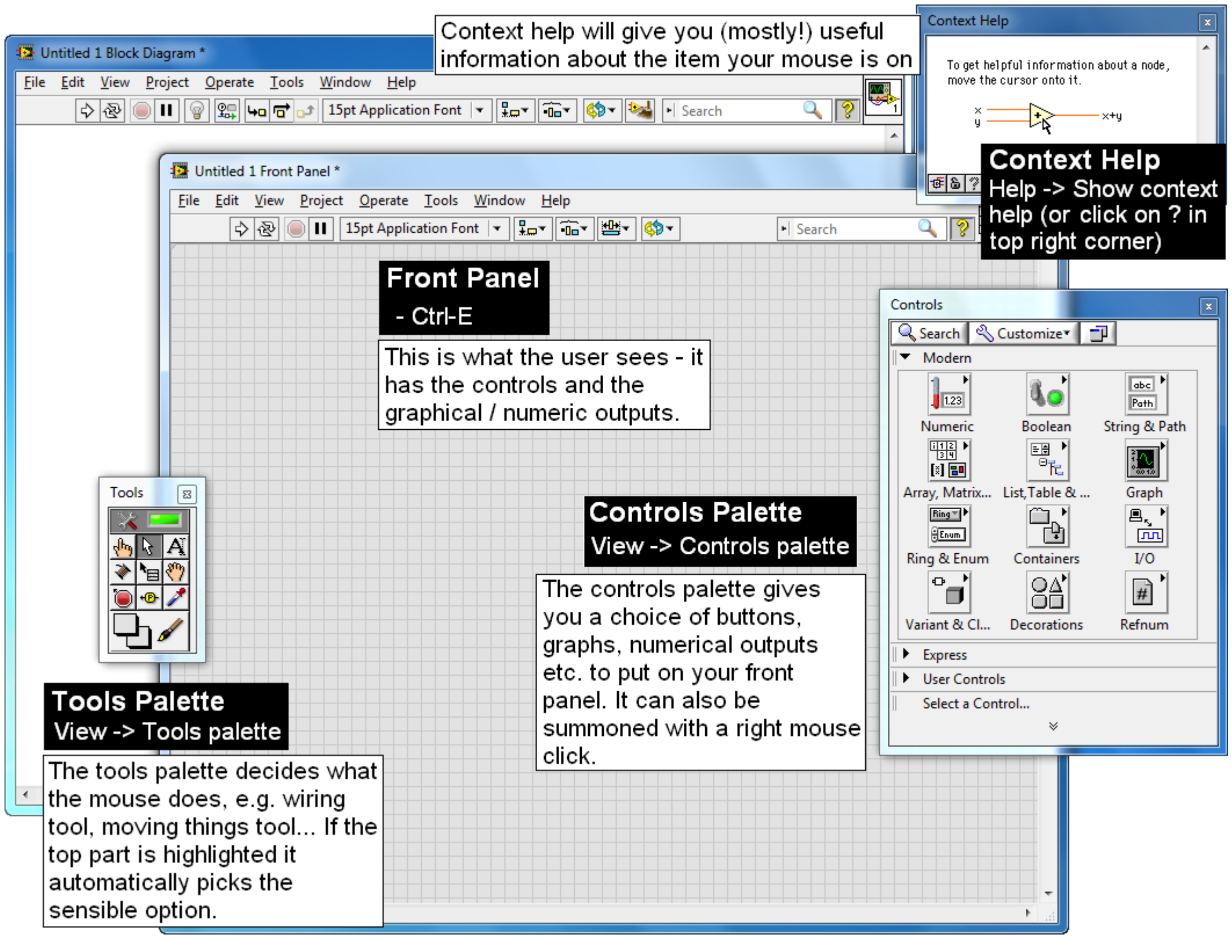
LabVIEW is a graphical programming language, and is a nice way of building virtual instruments (VIs) such as PC-based oscilloscopes. Attempt all the exercises even if you know the language already; these will be useful for later parts of the laboratory, and may be marked for credit. LabVIEW has many further built-in training programs, and more examples are available on the National Instruments website and in the books in the lab (see References).

1.1 Starting LabVIEW

To run LabVIEW, from the *Start* menu choose *All Programs → Nat. Inst. LabVIEW 2011 SP1* (this may also be at the top of the *Start* menu list). A window will open and offer you the option of opening an existing routine or starting a new one.

1.2 "My first routine" – a simple calculator

Open a blank VI; figures 1.1 & 1.2 show the program windows, with keyboard shortcuts and menu options to open or switch between them. ***Note**: LabVIEW has a built-in automatic tool selection feature on the Tools Palette (top section) that changes the behaviour of the mouse pointer, depending on what type of object you are pointing at*. You should have open a blank grey Front Panel – to make the user interface – and a Block Diagram, which is where the programming is done. It can be useful to open Context Help.


Context help will give you (mostly!) useful information about the item your mouse is on

Context Help
Help -> Show context help (or click on ? in top right corner)

To get helpful information about a node, move the cursor onto it.

Front Panel
- Ctrl-E

This is what the user sees - it has the controls and the graphical / numeric outputs.

Controls Palette
View -> Controls palette

The controls palette gives you a choice of buttons, graphs, numerical outputs etc. to put on your front panel. It can also be summoned with a right mouse click.

Tools Palette
View -> Tools palette

The tools palette decides what the mouse does, e.g. wiring tool, moving things tool... If the top part is highlighted it automatically picks the sensible option.

Figure 1.1; Front Panel. Control of the windows is described in the black boxes.

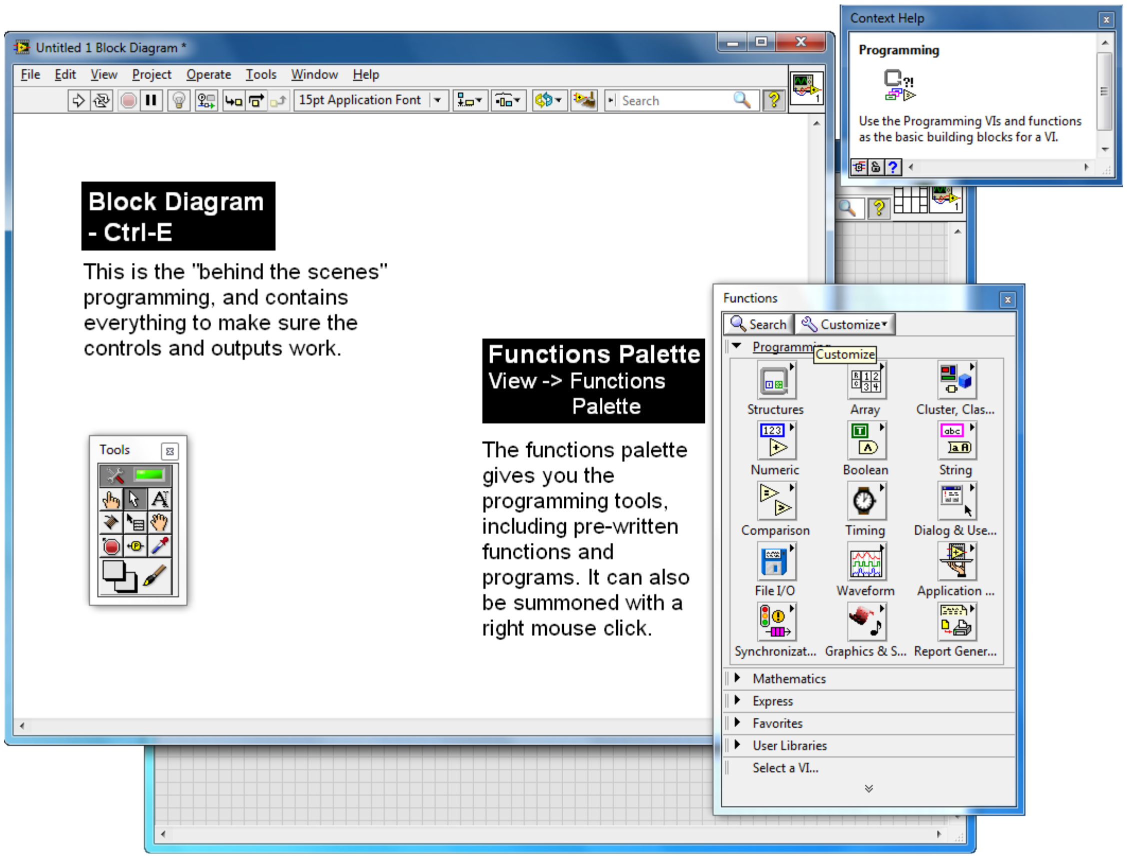

Untitled 1 Block Diagram *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Block Diagram - Ctrl-E
This is the "behind the scenes" programming, and contains everything to make sure the controls and outputs work.
Functions Palette
View -> Functions Palette
The functions palette gives you the programming tools, including pre-written functions and programs. It can also be summoned with a right mouse click.
Tools
Context Help
Programming
Use the Programming VIs and functions as the basic building blocks for a VI.
Functions
Search
Customize
Programming
Structures
Array
Cluster, Clas...
Numeric
Boolean
String
Comparison
Timing
Dialog & Use...
File I/O
Waveform
Application ...
Synchronizat...
Graphics & S...
Report Gener...
Mathematics
Express
Favorites
User Libraries
Select a VI...

Figure 1.2; Block diagram.

1.2 A. Your first challenge will be to build a simple calculator to add two numbers – this is only intended to show you how to start programming! Start by building your user interface – this should have two numeric inputs for the numbers, and an output to display the result (fig 1.3).

- Open the controls palette and choose *Modern*. To see icons and words as in figure 1.1, click the *Customize* button on the *Controls Palette* and choose *View this palette as » Category (icons & text).* You can also choose *Change visible palettes* to see *Modern*, if it's not visible.
- Add some numerical controls for inputting the numbers. In figure 1.3 I chose *Modern » Numeric » Numeric Control,* but there are other options! Click on your chosen control, then click on the Front Panel where you want to place it. Add a second control for the second number.
- Choose an output for displaying the results, such as *Modern » Numeric » Numeric Indicator*, place on the Front Panel, and arrange the icons. To add writing, either double-click on the Front Panel or choose the "A" tool from the Tools Palette.

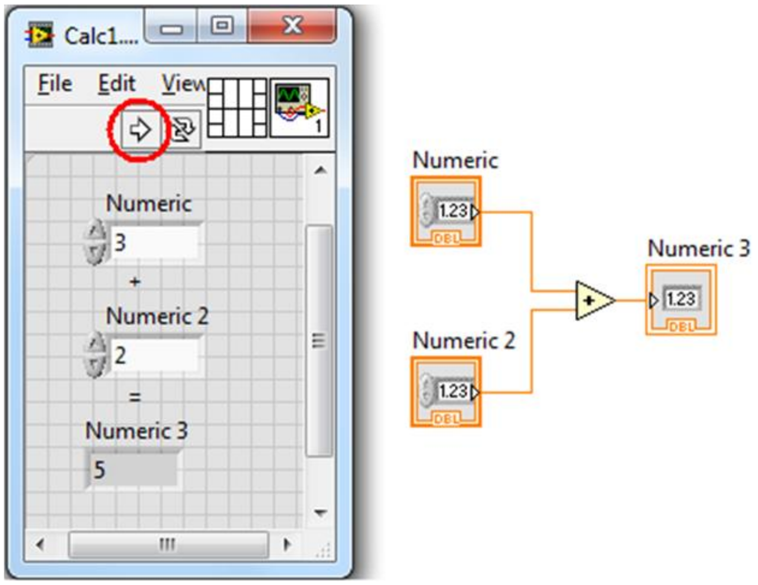
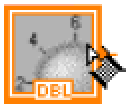


Figure 1.3; Calculator A; front panel (left) and block diagram (right)

Now go to the block diagram. The two numerical inputs and one output will need rearranging. ***Note: LabVIEW does not execute from left to right or from top to bottom. The functions execute when the data they need reaches them, so arrange your program like a flow diagram.***

- Open the functions palette and choose *Programming » Numeric » Add* (the triangle shown in figure 1.3) and place on the block diagram.
- In LabVIEW, data flows along wires, so you need to wire everything up – hover the mouse over the little output arrow on the right-hand side of a numerical input, and the pointer should change to a spool of wire. Left-click once on the arrow, and again on one of the inputs for the *Add* (this will become clear when you try it).



- Wire the second numerical input to the *Add*. Wire the output of the *Add* to the numerical indicator, as shown in figure 1.3.
- Don't forget to save your program.

Now go back to the front panel and input some numbers. To run your program, press the arrow underneath "Edit" – if the arrow is broken, there are errors in your program. *Check the answer.*

Things to try;
- Display your answer pictorially, using e.g. a tank or a meter, as well as a number. You may need to alter its properties to give the right range of numbers – right click on the display and choose *Properties, Scale*.
- Right-click a numerical input control, and choose *Properties*. What does changing the data range, the format and precision do? (You can change its label in the "Appearance" tab).
- On the block diagram, right click on the *Add* icon. One of the choices is "Replace". Try replacing it with some other mathematical operation.
- Open Context Help and hover the mouse over the orange wire. What does it tell you about the data type? (see also Appendix B.)

1.2 B. A calculator that only does one thing is not very useful! Calculator B (shown in fig 1.4) uses a case structure to give more options. The *Case Structure* works as "if… then do… else…" does in text-based languages, and allows you to choose the action.

- Open the functions palette. Choose *Structures » Case Structure*. To place this on the block diagram, click towards the top left of the *Add* button and drag the mouse to the bottom right of it. This may break the wires – ***press Ctrl-B to remove all broken wires***.
- If needed, rewire the inputs to *Add*, and the result to the output (Case 1, figure 1.4).
- A standard case structure has two possible actions for an input of *True* or *False*. Click the right pointing arrow at the top of the *Case Structure* to open the case for an input of *True*. Into this window place your second mathematical function, and wire it up.
- This structure needs a true/false, i.e. Boolean, control. Right-click on the green ? on the left hand side of the case structure and choose *Create Control*. This will give you a Boolean control on the Front Panel. Move this to a sensible place and play with its properties (right click the control for its menu); in the *Operation* tab it should switch when pressed. In figure 1.4 a Toggle switch is used.
- Save your program with a different name ("Substitute copy for original").

Now test your program, changing between the functions before running.

The Case Structure has one or more sub-diagrams, or cases, exactly one of which executes when the program runs. The value wired to the green ? determines which case to execute and can be Boolean, string, integer, ring or enumerated type. It is possible to use this to choose between +, -, x and / for a more complicated calculator *(**try this if you want a challenge – a Ring menu control may be useful, and you can add cases by right-clicking on the structure border; include your program in your lab book if you do this**)*.

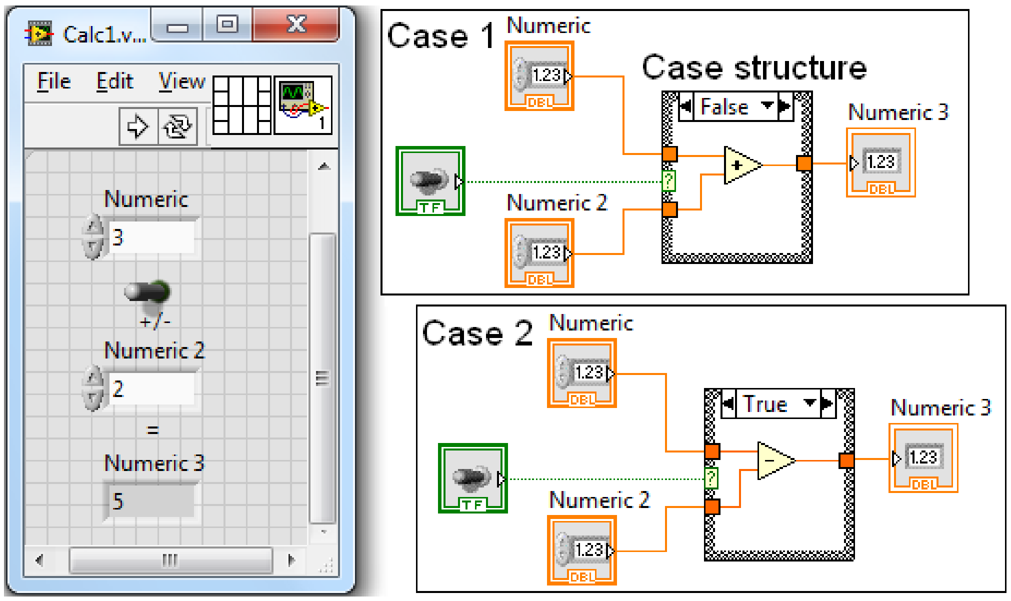
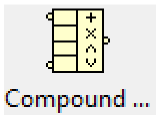


Figure 1.4; Calculator B; Front panel (left) and block diagram (right), showing two different cases.



1.2 C. LabVIEW has many built in mathematical operations. Go back to your first version of the calculator. Right-click on the *Add* button on the block diagram and choose *Replace*. If you want to add more than two numbers use *Compound Arithmetic* (shown right). To add more inputs make this icon larger.

For a more complicated calculation you can create a function using the built in Formula VI; Functions palette, *Express » Arith. & Comp… » Formula*. This will take as many inputs as you need and you can program in your chosen formula (you can use the Formula structure instead if you like using written code).

Choose a formula or expression and test the express VI for several inputs. Note the formula and comment on the process and the results in your lab book.

1.3 While Loops, For Loops and Shift Registers

When a program is running it will continue to run until either it has finished, or it is stopped. In the previous examples there was a definite stopping point. If this isn't reached by a sensible time you can use the red *Abort Execution* button to the right of the run arrow to stop a routine; **this is not recommended** as it can cause problems (e.g. when saving a file, LabVIEW does "open file"; "write"; "close file", and aborting before the file is closed can corrupt it). Instead, you can use different types of loop to define a stop point, including the following;

- *While loop* – this will keep running until a stop button is pressed, doing exactly the same thing each time it runs (each iteration)
- *For loop* – this is related to *"for N=1 to 10, do…"* in text-based languages. It will repeat the operation N times.

It is sometimes required to carry data from one iteration to the next. This simplest way to do this is using shift registers.

By now you should be getting to know your way around LabVIEW.

- Open a new blank VI and program Figure 1.5a (essentially a counter). Start by placing the *Add* function on the block diagram, along with a constant (right-click an input to the *Add*, and choose *Create » Constant*; 1 in this figure, choose your own number).
- On the block diagram, put a *While Loop* (*Structures » While Loop*) around the number and

the *Add*, in the same way as adding a *Case Structure*. A Stop button will appear in the bottom right hand corner of the loop – right click on this and choose *Create » Control* to get a Stop button. Wire these together if needed.

- Add some timing control (*Timing » Wait until next ms multiple*, and create a numeric constant); a small delay on each iteration of the program will free the processor to do any other essential tasks.
- Wire from the output of the *Add* to the edge of the While Loop. An orange box will appear on the Loop – right-click on this and choose *Shift Register*, and the up-arrow box will appear. A corresponding box should appear on the left side of the loop (if not, the mouse pointer will change and you will need to click on that side). This will feed the output back into the While Loop so it acts as the start point for the next iteration.
- Now wire the rest! The constant to the left of the loop (2 in this case) is required to give an initial input. Run the program and check what happens; press the Stop button to finish.

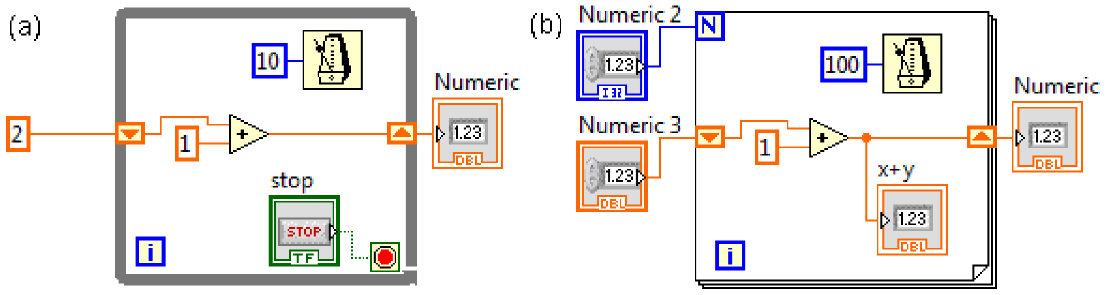


Figure 1.5; Simple addition routine with shift registers, using (a) While or (b) For Loops

As in text-based programming languages, you can run a program for a chosen number of iterations using a For Loop. On your program right-click on the While Loop and choose *Replace with For Loop* (as in figure 1.5b). Now you don't need the stop button – but you do need to tell it how many iterations to do by wiring a control or a constant to the *N* (right-click, *Create Constant* or *Control*). Choose the number of iterations and run the program. To give yourself time to see what the program does, increase the delay.

Make brief notes on the behaviour of the two loops, noting the differences between each "Numeric" output, and "x+y" on figure 1.5.

Up until now you have worked with single or double accuracy real data, and Boolean data. The other types of data, including arrays and dynamic data, are shown in Appendix B.

Congratulations – now you know some LabVIEW!

2. INTRODUCTION TO COMPUTER CONTROL

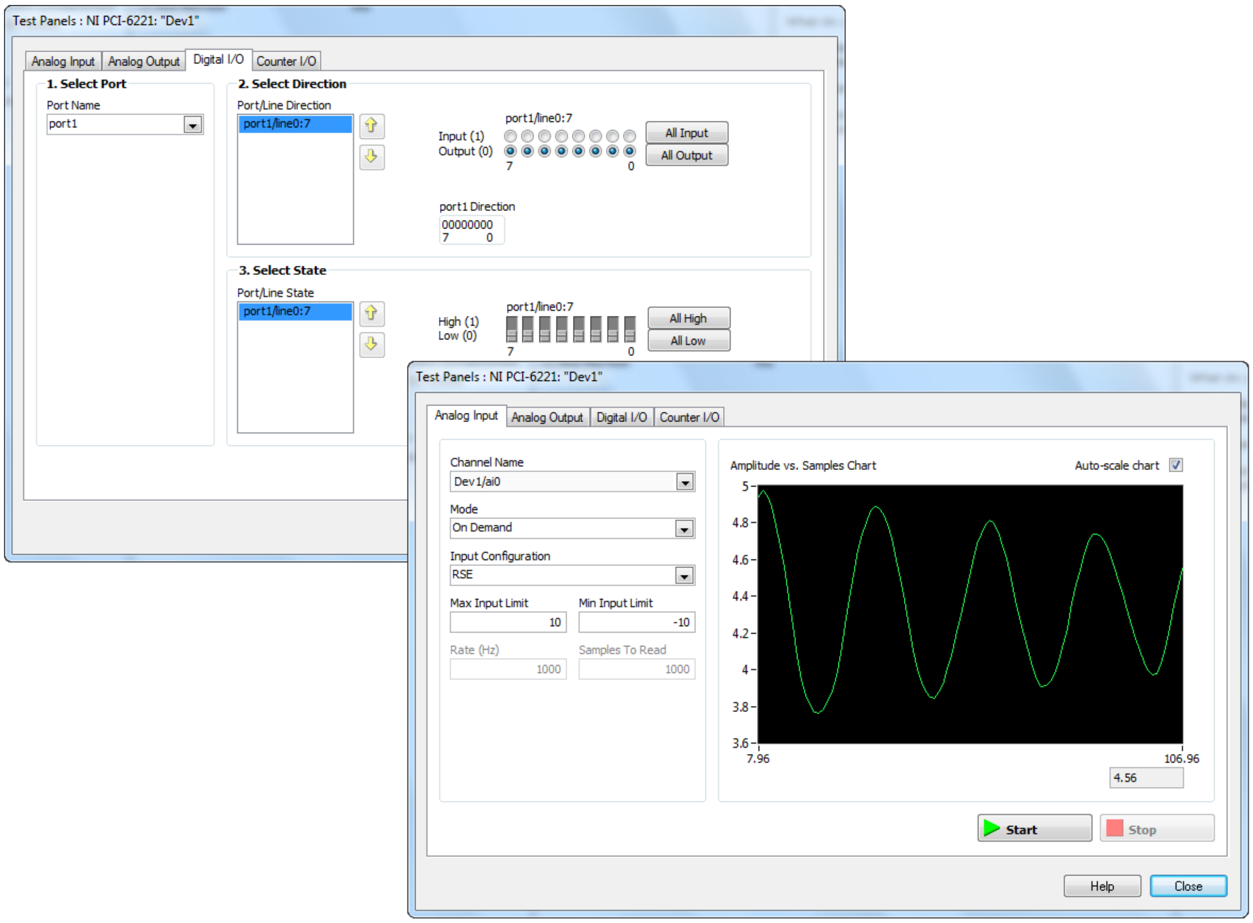
When you have completed the Introduction to LabVIEW exercises you should be ready to learn control of experiments using LabVIEW and the data acquisition (DAQ) board (the analogue to digital, A-D system – see appendix A). The box labelled Servo Controller Interface (the interface between the DAQ board and the experiments) should now be switched on. The following exercises will familiarise you with passing digital and analogue information to and from the equipment on the bench.

2.1 Measurement & Automation; checking the system

Plugged into the Interface box should be an LED board, a pair of crocodile clips, and the

“Nodding Donkey”, a moving arm with a sliding mass for exploring control and feedback mechanisms. The sliding mass is controlled by a motor using a digital output from the computer, and a potentiometer gives an analogue input related to the arm position. You should also have a resistor ladder for use with the crocodile clips.

It is important to check equipment before doing experiments. Open **Measurement & Automation** from the desktop. On the left hand panel choose *My System » Devices & Interfaces » NI PCI-6221: “Dev 1”*; this is the National Instruments DAQ board. Start by performing a self test – click on *Self-Test*, this should open a window reading “Okay”. If not, see a demonstrator.


Test Panels : NI PCI-6221: "Dev1"

1. Select Port
Port Name: port1

2. Select Direction
Port/Line Direction: port1/line0:7
port1 Direction: 00000000

3. Select State
Port/Line State: port1/line0:7

Test Panels : NI PCI-6221: "Dev1"

Channel Name: Dev1/ai0
Mode: On Demand
Input Configuration: RSE
Max Input Limit: 10
Min Input Limit: -10
Rate (Hz): 1000
Samples To Read: 1000

Figure 2.1; Test Panels; Digital IO and Analogue Input in Measurement & Automation program

2.1.1 Test digital control

- Open *Test Panels*. First check the LED board. Open the *Digital I/O* tab. The LED board is on port1, so set this in section 1 of the Digital IO tab (figure 2.1, left)
- Set to “All Output” in section 2.
- Press Start – this starts the computer talking to the port. In part 3 of this window, try changing some of the switches (next to the all-high and all-low buttons, see figure 2.1).
- If changing the switches turns on and off the LEDs on the board, this is set up correctly. Choose “All Low” and press Stop.
- Now check the digital control of the moving arm. This is on port 0, again as “All Output”.
- Press start, and turn on the switches, starting on the side marked 0. The sliding mass, and hence the arm, should move – this movement will become very noticeable by the time the output is at 00011111. Reselect “All Low” and press Stop.

2.1.2 Test analogue input

- Now open the Analogue Input tab.
- Firstly, check the crocodile clips. These are on Dev1/ai1, with an Input Configuration of RSE and input limits of ± 10 (figure 2.1, right).
- Press Start and check that the graph changes when you connect the clips to different pins on the resistor ladder. Remember to connect its battery! Press Stop.
- The arm position reading is on Dev1/ai0 (RSE, ± 10). Check the settings, press Start, and gently nudge the arm. The graph should update as in figure 2.1, right.
- Press Stop and close the Measurement & Automation window.

This has confirmed that all components are working correctly. You have achieved computer control! However, it is much better to do this in a program which can control things in exactly the way you want, and then analyse and save the data. This is when LabVIEW is useful…

2.2 Digital output; control of LEDs

The first exercise here is to control the LED board using LabVIEW. Initially, you will control just one LED.

- Open a new blank VI in LabVIEW. Place a button / switch on the Front Panel – this will be the on-off switch for the LED. To make it show "on" or "off" you can right click the switch and in *Properties* choose *Show Boolean Text*, and add text in *Multiple Strings*. In the *Operation* tab, choose *Switch when pressed*.
- Place an LED on the Front Panel. If you don't know where in the Functions Palette these are you can Search, look through the options, or as a last resort ask a demonstrator.
- On the Block Diagram place a While Loop around the button and the LED and add a Stop button. Wire between the button and LED.
- Run the program – pressing the button should turn the LED on screen on or off. Press the Stop button.

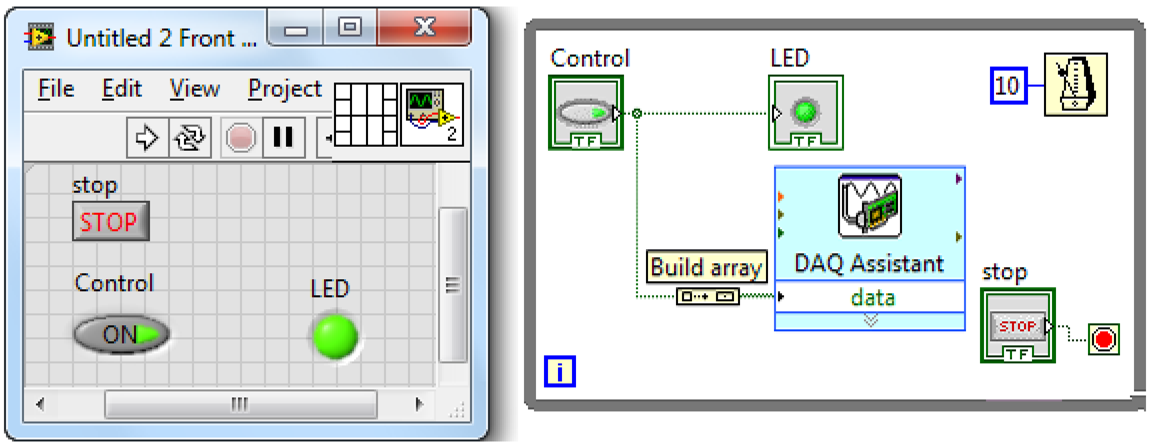


Figure 2.2 Control of a single LED on board and on screen

Now to talk to the LED board. LabVIEW has a series of Express VIs, sub-programs written by National Instruments, which do common tasks. To talk to equipment you need *Express » Output » DAQ Assist*, so place this on the Block Diagram. A configuration window will open. Choose *Generate Signals*, with a *Digital Output*, and a *Line Output*. The LED board is, as in section 2.1, port1, and we will talk to one LED on line0. Press Finish.

- In the next window choose *1 Sample, On Demand*, and press Okay. This will send a signal to the LED board whenever LabVIEW executes an iteration.
- If you wire the button directly to the DAQ Assistant you will get an error, as the DAQ

expects an Array input. Instead, on the Block Diagram place a *Build Array* (*Programming » Array » Build Array*). The output will be a one-dimensional array containing the required data. Wire as shown in figure 2.2, including a small delay.

Run the program. Pressing the button should control both the screen and the board LED. Don't forget to save your program.

The second exercise is to control all 8 LEDs on the LED board and to use this to translate a number from a decimal input on the screen to a binary representation on the board. Save your program as a new version.

- On the front panel replace the button and LED with a numerical control. Right-click this, choose Representation, and pick U8 (8-bit, unsigned integer). This will send the expected number of bits to the DAQ assistant. In Properties, under Data Entry, set the steps to be integer steps.

What is the range of numbers you can represent with 8 bit binary? Explain your calculation in your lab book and set the limits for the numerical control.

- On the block diagram, convert from a number to a Boolean array to send the correct output to the LED board. Fortunately, LabVIEW has a tool for this in *Programming » Numeric » Conversion » Number to B…*
- Wire the number to the conversion, and the conversion to the DAQ Assistant (figure 2.3).

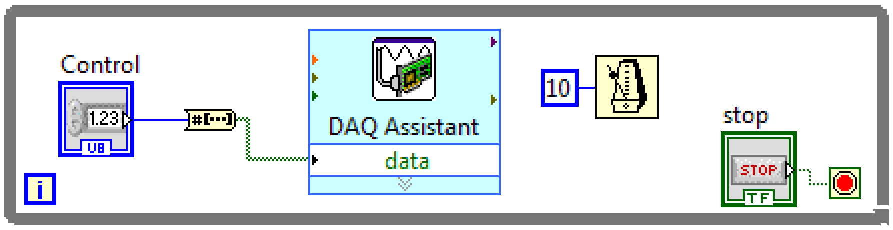


Figure 2.3 Numeric control of all LEDs

- The DAQ expects only one line input, but is now going to get 8 bits of information. Double-click the DAQ Assistant. Click *Add Channels* from the top menu, and add Dev1 / port1 / line1 (you should already have line0). Repeat this until all eight lines on port1 are included in the right order.

Press Okay. It will think for a while, then your program should be ready to run.

Test that your program works and correctly converts from a number to binary. Give the conversion of several numbers in your lab book.

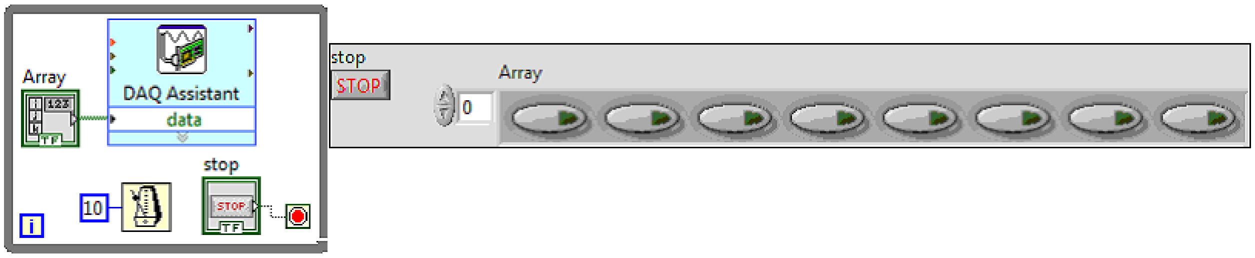


Figure 2.4 Boolean control of all LEDs

You can also control the LEDs using an array of Boolean switches. In a new version of the program delete the Numeric Control and Conversion.

- On the Front Panel, open the *Controls Palette*, choose *Modern » Array, Matrix & Cluster » Array*, and place on the Front Panel.

- Again from Controls, choose *Modern » Boolean » Push Button* and place this ***inside*** the array. Stretch the array towards the right until you can see eight buttons (figure 2.4).
- On the Block Diagram, wire the array to the DAQ Assistant.

Go back to the Front Panel. All the buttons will be grey, and the program won't run until you press the far right button to **initialise** the array – LabVIEW always needs this with new arrays! Run the program, have fun turning LEDs on and off… then ***turn off all LEDs*** and press Stop.

By using LabVIEW skills you have already learnt it is possible to switch between numeric and switch control in a single program. *If you have time*, attempt this (& put your program in your lab book).

2.3 Analogue input; voltmeter

This section is concerned with reading in analogue signals, which can then be processed and/or plotted in LabVIEW and saved to a file for future analysis. For this you will use the resistor ladder, consisting of a 9 V battery and a chain of 10 nominally equal resistors. Using the crocodile clips you can measure the voltage across different numbers of resistors on the chain.

In section 2.1 you used Measurement & Automation to plot the analogue input from the crocodile clips. To recreate this you need a graphical display. LabVIEW offers several options, the most useful being Waveform Graph (this updates with a set number of points each time) and Waveform Chart (updates one point at a time). Open a new Blank VI and place a waveform chart (*Express » Graphs » Waveform Chart*) on the Front Panel. On the Block Diagram,

- Choose *Express » Output » DAQ Assist* and place on the Block Diagram.
- Set up the DAQ; *Acquire signals » Analogue Input » Voltage » Dev1, ai1* and press Finish. Set-up with a signal input range of -10 to +10 V, RSE terminal configuration and 1 sample (on demand). Press Okay.
- Wire between the DAQ Assist and the chart, add a While Loop and Stop button so you can stop the program acquiring data, and a short delay.

Run the program and test by checking the voltage across one or more resistors. You can change the properties of the chart, for example turning off auto-scaling.

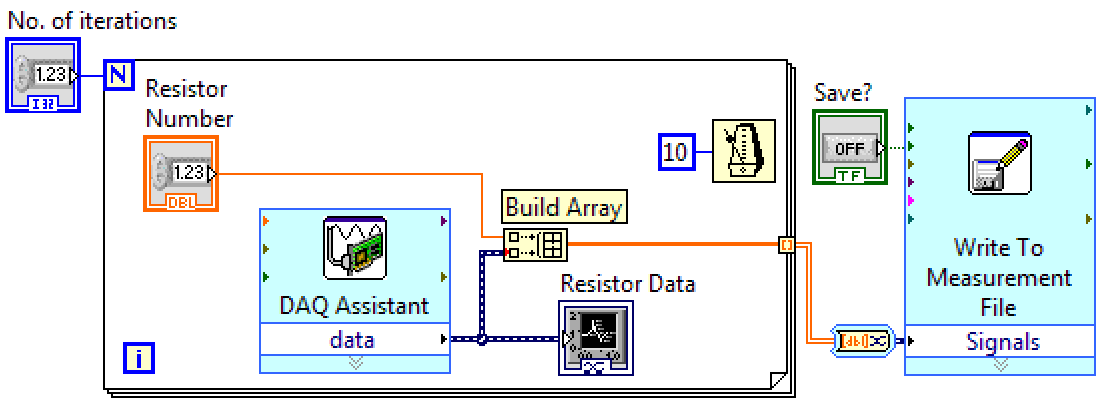
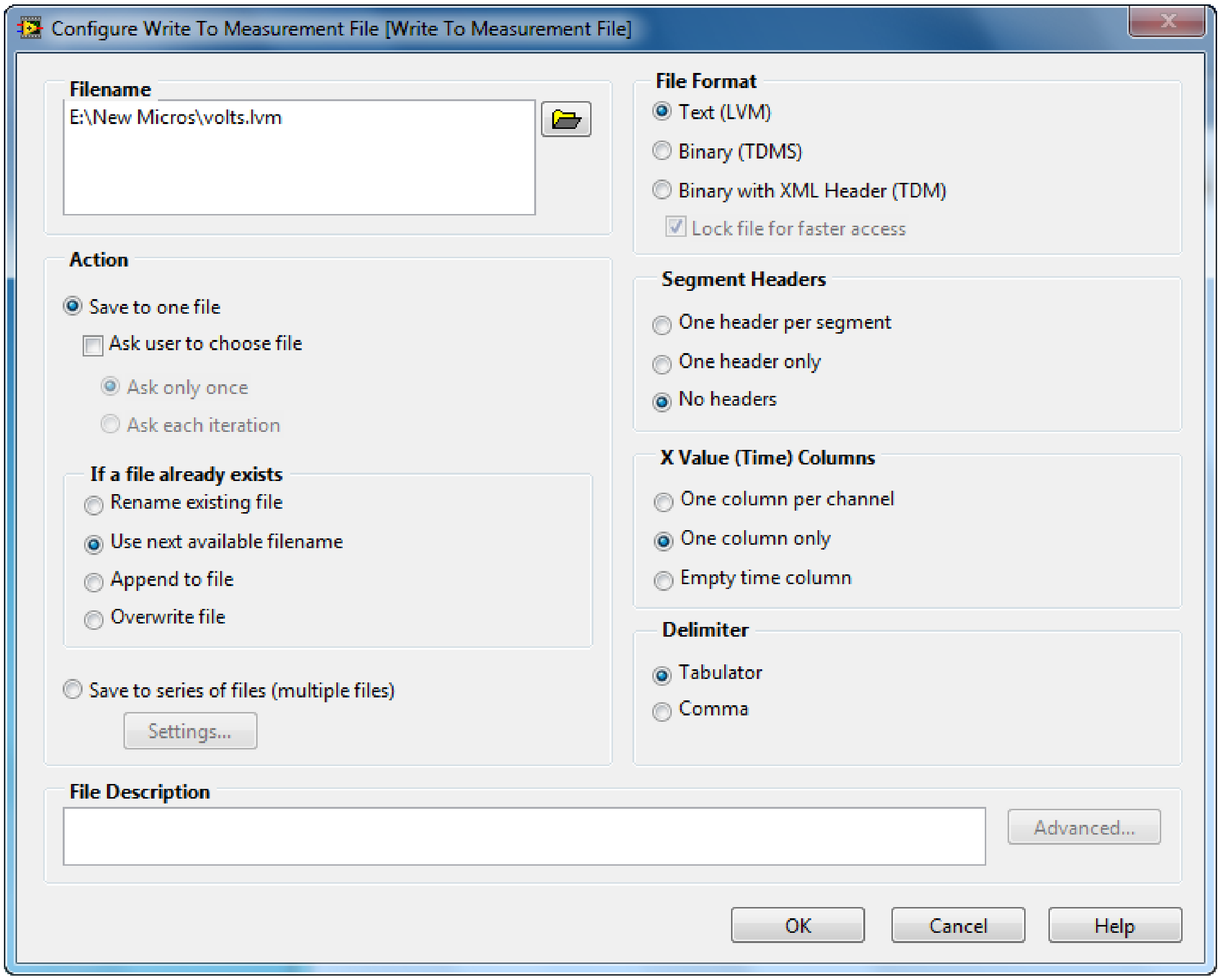


Figure 2.5 Analogue input and data saving routine for resistor ladder measurements

This program gives a graphical output, but in order to get a reliable measure of the voltage at different points on the chain you will need to take an average. This can be done manually, by saving a set number of points and using a data analysis program, or by using feedback loops (e.g. with shift registers). Figure 2.5 shows a suggested program to save the voltage drop over a resistor a chosen number of times. This is based on your previous program, replacing the While Loop with a For Loop, with a control for the number of iterations (amount of data for each resistor). This allows you to input the resistor number; use a *Build Array*, extending so it has two inputs, giving an array containing resistor number and voltage. The data is output to a file using an Express VI, found in *Programming » File I/O » Write Meas File*. When you place this on the block diagram a window will open for you to change the settings.


Configure Write To Measurement File [Write To Measurement File]

Filename
E:\New Micros\volts.lvm

Action
[✓ Save to one file]
☐ Ask user to choose file
[✓ Ask only once] Ask each iteration

If a file already exists
Rename existing file
[✓ Use next available filename]
Append to file
Overwrite file

Save to series of files (multiple files)
Settings...

File Format
[✓ Text (LVM)]
Binary (TDMS)
Binary with XML Header (TDM)
✓ Lock file for faster access

Segment Headers
One header per segment
One header only
[✓ No headers]

X Value (Time) Columns
One column per channel
[✓ One column only]
Empty time column

Delimiter
[✓ Tabulator]
Comma

File Description

Advanced...

OK Cancel Help

Figure 2.6; Write Meas File

Set this up as in Figure 2.6, choosing a sensible name for the file on your H: drive or USB stick. Using headers will give information about the measurement but can make the file harder to read. To save multiple data to one file choose “Append to file”.

Context Help will show you the wiring options for this VI; one is called *Enable* and allows you to turn on and off saving the data using a switch. When you wire from *Build Array* to the signal input an *Array to Dynamic Data* converter appears. Double-click this and check it is converting “2D array of scalars, columns are channels”. Run the program and check what the output file looks like – you should be able to read it using notepad.

Perform an experiment to test the properties of the circuit (e.g. what values can you find?), plot the results in Origin (or Excel, if you must), and comment on the results.

You should now be ready to proceed to the main control experiment.

REFERENCES

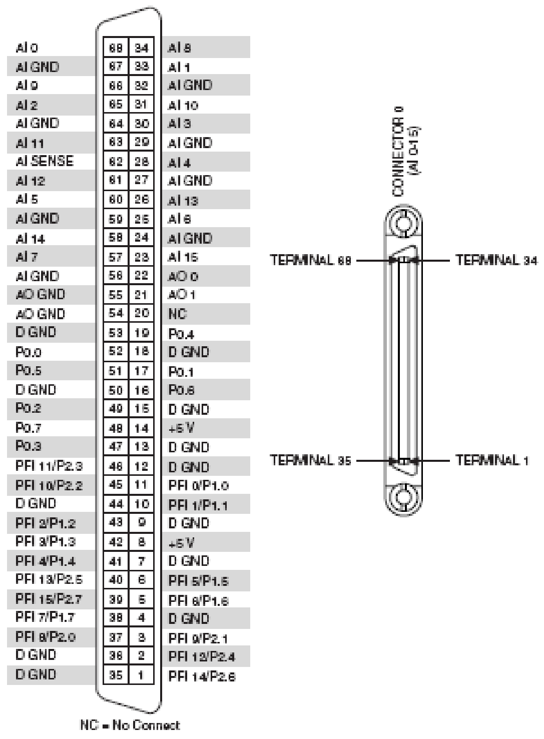
There are several LabVIEW 8 textbooks in the cupboard in the lab, and the library has several books based on earlier versions, for example

- LabVIEW 8 Student Edition, Robert Bishop, Prentice Hall, London (2006)
- LabVIEW for Everyone, Jeffrey Travis and Jim Kring, Prentice Hall, London (2006).

There may be newer versions available, and there are also several e-books linked from the library catalogue. It can also be useful to get information from websites, but be aware that not everything is official or even right!

http://www.ni.com is the official National Instruments site.

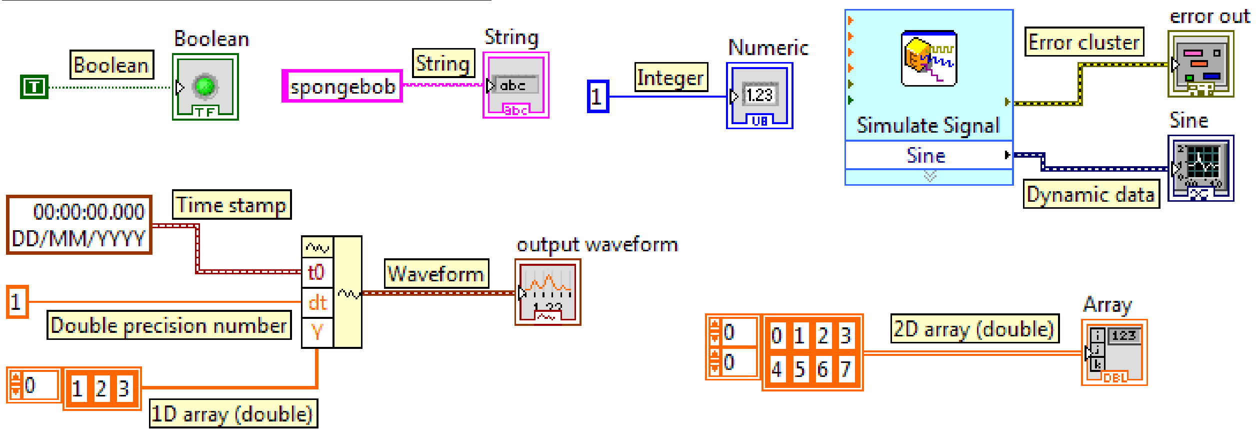
APPENDIX A: NI PCI-6221 DAQ data acquisition board



Data acquisition (DAQ) techniques sample information about the real world and put it into a form that can be understood and manipulated by a computer. A typical system has an experiment connected through cables to a DAQ board (in this case via an interface box) which then passes data to and from the computer. The DAQ hardware digitizes signals, performs digital to analogue conversions to generate analogue output signals, and measures and controls digital I/O signals.

Each computer in this lab has an interface board fitted in a PCI slot. The data acquisition board, with connections through the Servo Controller Interface, is a National Instruments M Series DAQ, PCI-6221 (68 pin version). It has 16 analogue inputs, 24 digital I/O and 2 analogue outputs, and can be controlled via the NI-DAQmx software. The manual is available on the National Instruments website; pin inputs and outputs are shown in the figure. This board provides the connections (ports) with the real world.

APPENDIX B: LabVIEW Data Types



T
Boolean
Boolean
spongebob
String
String
1
Integer
Numeric
Simulate Signal
Sine
Error cluster
error out
Sine
Dynamic data
00:00:00.000 DD/MM/YYYY
Time stamp
t0
dt
Y
Waveform
output waveform
1
Double precision number
0
1 2 3
1D array (double)
0
0
0 1 2 3
4 5 6 7
2D array (double)
Array