Measurement application for injector solenoid

Measurement application for an injector solenoid in the Wärtsilä 46CR engine

Tony Ohls

Degree Thesis for Information Technology (Bachelor of Information Technology)

Bachelor's thesis in Information Technology

Vaasa 2012

BACHELOR’S THESIS

Author:	Tony Ohls
Degree Programme:	Information Technology
Supervisor:	Mats Braskén

Title: *Measurement application for injector solenoid*

Date 12.04.2012 Number of pages 31

Summary

This thesis was commissioned by Wärtsilä Finland – Services and concerns creating an application with LabVIEW that will measure key values from an injector solenoid intended for a Wärtsilä 46CR engine.

The application created will analyse two signals that are sampled with a frequency of 100 000 Hz and based on these signals, the key values will be calculated. The key values are: the time it takes for a valve to start opening from the moment power is fed into the solenoid, the time it takes for the valve to start closing from the moment power is cut to the solenoid, the time it takes for the valve to close from the moment power is cut from the solenoid, and the final key value is the amplitude of the bounce that happens after the valve closes in percentage compared to the amplitude value of the fully open valve.

After the key values are found they will be presented in a user interface. After a test run of the component is complete, it will also be possible to print the minimum, average and maximum key values to an HTML file.

Language: English

Keywords: LabVIEW, signal analysis, measurement system, injector solenoid

OPINNÄYTETYÖ

Tekijä:	Tony Ohls
Koulutusohjelma:	Informaatioteknologia
Ohjaaja:	Mats Braskén

Nimike: Mittaussovellus injektori solenoidiin

22.04.2012 31 sivua

Tiivistelmä

Wärtsilä Finland – Services on tilannut tämän insinöörityön, joka koskee sovelluksen luomista, joka luodaan LabVIEWillä. Tämä sovellus tulee mittaamaan injektori solenoidin avain-arvoja, jota tullaan käyttämään Wärtsilä 46CR moottorissa. Sovellus, jota luodaan tulee analysoimaan kahta signaalia, jota samplataan taajuudella 100 000 Hz ja näiden kahden signaalin avain-arvot tullaan laskemaan tämän perustelulla.

Avain-arvot ovat: Aika, kuinka kauan kestää ennen kuin venttiili alkaa avautua sen jälkeen kun virta syötetään injektori solenoidiin. Aika, kuinka kauan kestää ennen kuin venttiili alkaa sulkeutua sen jälkeen kun virta on sammutettu injektori solenoidilta. Aika, kuinka kauan kestää venttiilin sulkeutumiseen, sen jälkeen kun virta injektori solenoidilta on sammutettu, ja lopullinen avain-arvo on kuinka korkealle venttiili pomppii, kun se sulkeutuu.

Kun avain-arvot on löydetty esitetään arvot käyttöliittymässä. Sen jälkeen kun komponentin koeajo on tehty valmiiksi on myös mahdollista tallentaa pienin arvo, keskiarvo ja korkein arvo kaikista avain-arvoista yhteen tiedostoon, joka myöhemmin tarvittaessa voidaan tulostaa paperille tai lukea tietokoneelta.

Kieli: englanti

Avainsanat: LabVIEW, signaalin analysointi, mittausjärjestelmä, injektori solenoidi

EXAMENSARBETE

Författare:	Tony Ohls
Utbildningsprogram och ort:	Informationsteknologi
Handledare:	Mats Braskén

Titel: Mätprogram för solenoidinjektor

Datum 30.04.2012 Sidantal 31

Sammanfattning

Detta ingenjörsarbete görs på beställning av Wärtsilä Finland – Services, och berör skapandet av en applikation i LabVIEW. Denna applikation kommer att mäta några nyckelvärden för en solenoidinjektor som skall användas i en Wärtsilä 46CR motor.

Applikationen som skapas skall analysera två signaler som samplas med en frekvens på 100 000 Hz och baserat på dessa två signaler kommer nyckelvärdena att beräknas. Nyckelvärdena är tiden det tar för en ventil att börja öppnas efter att ström matas till solenoidinjektorn; tiden det tar för ventilen att börja stängas efter att strömmen till solenoidinjektorn stängs av; tiden det tar för ventilen att stängas efter att strömmen till solenoidinjektorn stängs av och det slutliga nyckelvärdet är hur högt ventilen studsar när den stängs.

Efter att nyckelvärdena hittats skall de presenteras för användaren i ett användargränssnitt. Efter att en testkörning av komponenten slutförs skall det också vara möjligt att spara de minsta, de högsta samt medelvärdena av alla nyckelvärden i en fil, som senare vid behov kan skrivas ut på papper eller läsas av på en dator.

Språk: engelska

Nyckelord: LabVIEW, signal analys, mätsystem, solenoidinjektor

Table of Contents

1 Introduction

This thesis concerns a project commissioned by Wärtsilä Finland – Services and is about measuring key values of an injector solenoid[1] in their Wärtsilä 46CR engines. The valve in an injector solenoid is magnetically controlled and opens when the solenoid is fed a current and closes when the current is cut. The reason for the testing is that the old component is to be replaced by a new type of injector solenoid and it will need to be correctly calibrated based on new criteria.

To accomplish this task Wärtsilä Finland – Services has started a project to create a test-bench in which the component will be tested. In addition to the physical test bench, a measurement application will also be needed to monitor the signals that can be acquired from the component.

The task of the measurement application will be to measure time intervals concerning opening and closing the valve and, additionally, how much the valve bounces after it has been closed.

2 Assignment

The assignment for this thesis concerns the creation of the program needed for the test-bench. The program will receive two different signals: a solenoid drive current signal and a valve lift signal. These two signals will both have a sampling frequency of 100 000 samples per second (Hz).

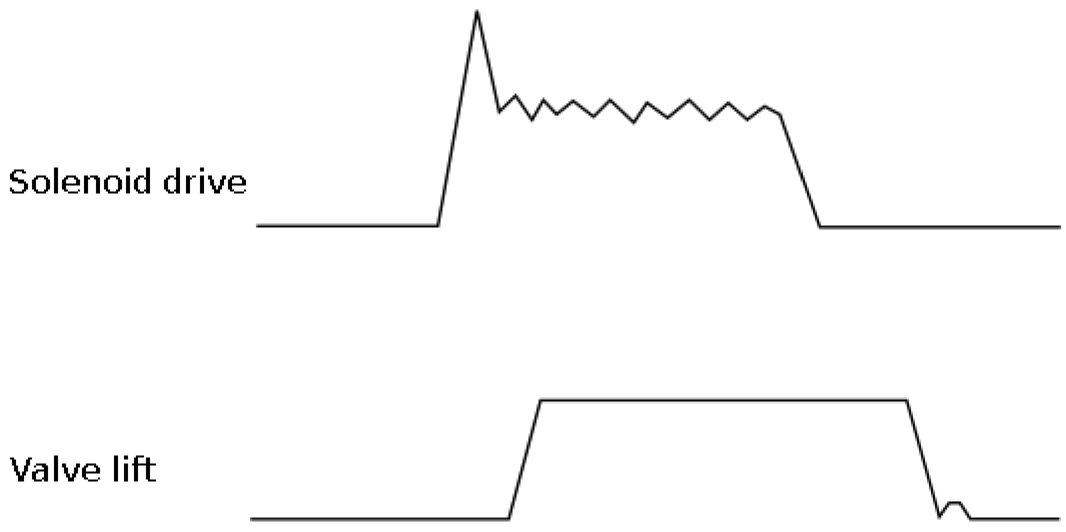


Figure 1. A model of the signals.

Based on these two signals, illustrated in figure 1 above, the program should be able to determine the following values:

- **T3:** The time it takes between current being fed to the solenoid and the valve to open up 20% of its fully opened state.
- **Time to start to close:** The time it takes from when the current to the solenoid stops until the valve closes to 20% from fully closed state.
- **T4:** The time it takes from when the current to the solenoid stops to the moment when the valve is completely closed (before the bounce).
- **Bounce:** The bounce of the valve after it has closed, defined by a percentage of the total amplitude.

The values of interest are: the minimum, the maximum and the average of all these measured values. The values will be shown on a user interface and they will be updated with a frequency of about once every one or two seconds in order to make sure the user is not overwhelmed by flashing numbers. The signal will, however, be analysed in the background as fast as the computer can handle them.

The time values that are to be measured need to have a precision of 10 microsecond (µs). All the values are required to be in the following ranges based on the parameters defined by Wärtsilä:

- **T3:** Between 750 µs and 870 µs.
- **Time to start to close:** Between 700 µs and 860 µs.
- **T4:** Between 1165 µs and 1315 µs.
- **Bounce:** The bounce is required to be less than 10% of the maximum valve lift.

The program will have these limits indicated on the user interface. These limits will only be editable by Wärtsilä Finland and every time they change, a new version of the program will be deployed to the installations with the test bench at the request of Wärtsilä Finland. This is to safeguard that they are always in control of what the limits are.

The reason why Wärtsilä is interested in these values is that they need them when calibrating the injector. In the injector there is a spring that can be tightened or loosened depending on if the key values are out of range. The spring is used when the valve in the injector is opened and closed and it thus affects the time it takes to perform these actions. It also has an impact on how high the bounce is after the valve closes. There are also some other actions that can be taken; amongst them is to clean the injector or increasing the air gap in the injector.

The program should be able to print the results of the measurements into a file that can later be viewed or printed as needed. There should be two different versions of the printout; one for external use that can be shown to customers and one version only for internal use containing additional information.

At Wärtsilä's request the program will be created using LabVIEW. The main reason for this is that LabVIEW is used quite frequently for similar tasks and it is possible to create a runtime version from LabVIEW that can be run at all the installations without needing to pay for any additional licenses beyond the license needed for the development environment. The program will be used by a handful of Wärtsilä network companies around the world where the test bench will be installed.

The project will be separated into different parts and this thesis will only concentrate on the application that is to be created. The hardware will be taken care of by another party and there have been assurances that the two previously mentioned signals will be available and the measurement application will be developed based on this assumption.

The thesis will first give a short introduction to LabVIEW. This introduction will be followed up by an in-depth look at different parts of the assignment and how to calculate them. After this the more practical parts will be discussed and the implementations will be presented. At the end of the thesis a short conclusion with a bit of discussion will be found.

3 LabVIEW

LabVIEW[2] is a system design platform and development environment created by National Instruments. It is widely used in industry for analysing signals and testing different products, for example engines and electrical components. LabVIEW is a graphical programming language that makes use of the G programming language that was released for the Apple Macintosh in 1986.

The G programming language is a so-called data-flow language. When coding in LabVIEW, nodes are placed graphically in a block diagram and then connected with wires. Nodes can have both input and output variables that can and, in some cases, have to be linked to other nodes. When a node has all the input variables it needs, it will automatically run and send the output variables down the wires, if they are connected to any other node. Since many nodes can have the input variables needed at the same time, G is inherently able to execute multiple nodes in parallel.

In LabVIEW there are two main windows used when coding, the *Front Panel* and the *Block Diagram*. The two different windows have the following functionality:

- **Front panel.** On the front panel it is possible to add elements, for example buttons, graphs, indicators and controls, which will make up the user interface. It is often possible to change the appearance of the elements through various properties menus directly in the front panel window. All the elements created on the front panel will also show up on the block diagram. Figure 2 is an example of a front panel in LabVIEW and it shows an interface for calculating an area of a triangle. In the figure, Base (cm) and Height (cm) are controls that the user can change while Area (cm^2) is an indicator that shows the calculated area of the triangle.

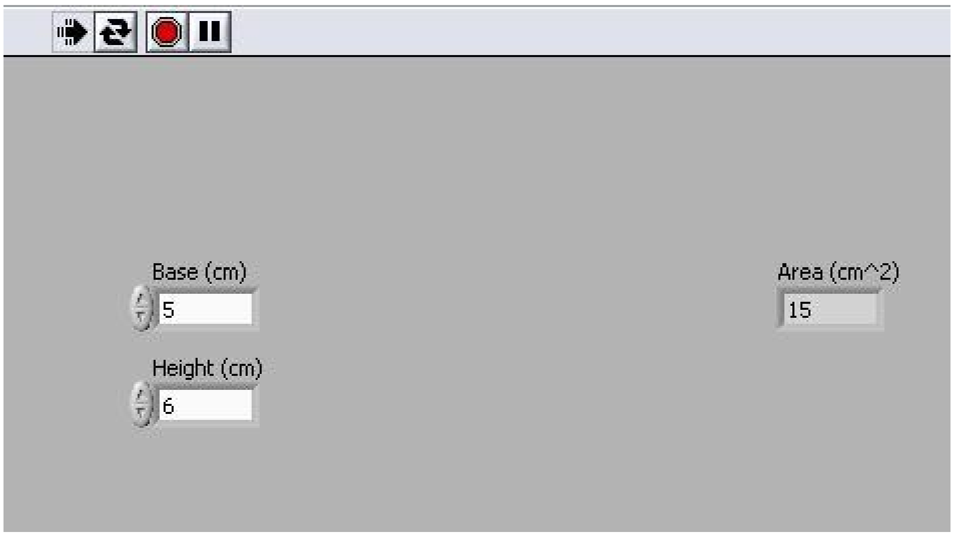


Figure 2. A screenshot of the front panel, taken from the LabVIEW Getting Started Block Diagram tutorial[3].

- **Block diagram.** On the block diagram all the functionality of a LabVIEW program is defined. Here the nodes and wires are created that then define the data flow of the program. In figure 3 below a block diagram is shown that belongs to the front panel example in figure 2. In the block diagram example, Base (cm) and Height (cm) are controls that both output a value each. These values are then multiplied with each other. The result of this multiplication is then multiplied by 0.5 and the result of this multiplication is then shown in the Area (cm^2) control.

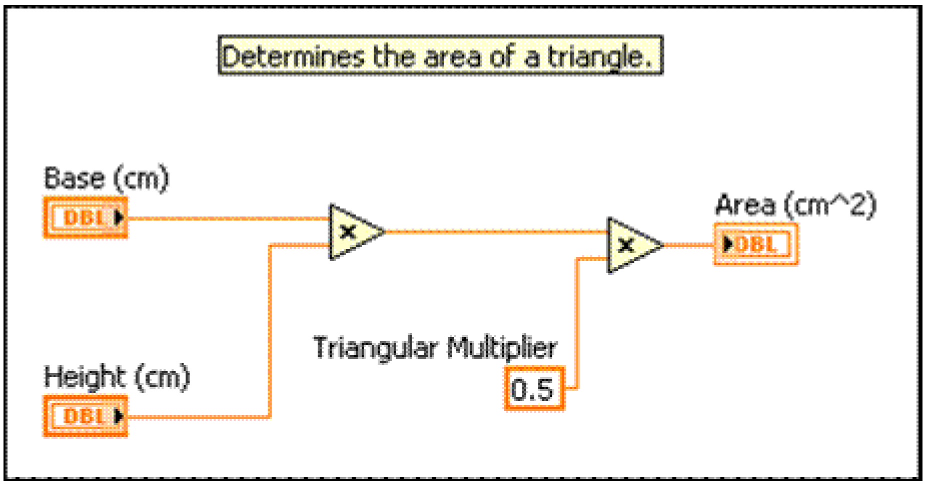


Figure 3. A screenshot of a block diagram, taken from the LabVIEW Getting Started Block Diagram tutorial[3].

LabVIEW is able to create runtime executables for all the programs created in the development environment. These runtime executables can be run on any computer as long as the LabVIEW run-time engine is installed on the target machine. The run-time engine is available for most common operating systems and thus most LabVIEW programs should be cross-platform compatible without any major changes.

Programs created with LabVIEW are called *Virtual Instruments* or VI for short. If a program is created for the purpose of being used by another VI, in text programming terms this would be quite similar to a function, it is called a subVI. In a subVI in addition to the normal functionality of a program inputs and outputs will also need to be defined. The inputs and outputs are defined on the front panel with a control that is called a *Connector Pane*. The inputs need to be an element of the type *control* while the outputs need to be of the element type *indicator*. On the connector pane, so-called *terminals* are defined which will then be linked to controls or indicators. It is also on the connector pane that the importance of the inputs and outputs is set. The importance can have 3 different levels: required, recommended and optional. If the importance of an input is set to required, then the subVI will not run unless that input is supplied. Recommended and optional inputs and outputs are, however, not necessary for the execution of the subVI. Below there is a short example of a subVI that calculates the area of a triangle by requiring two input values: height and width. When the size of the area has been calculated, the subVI will output the result.

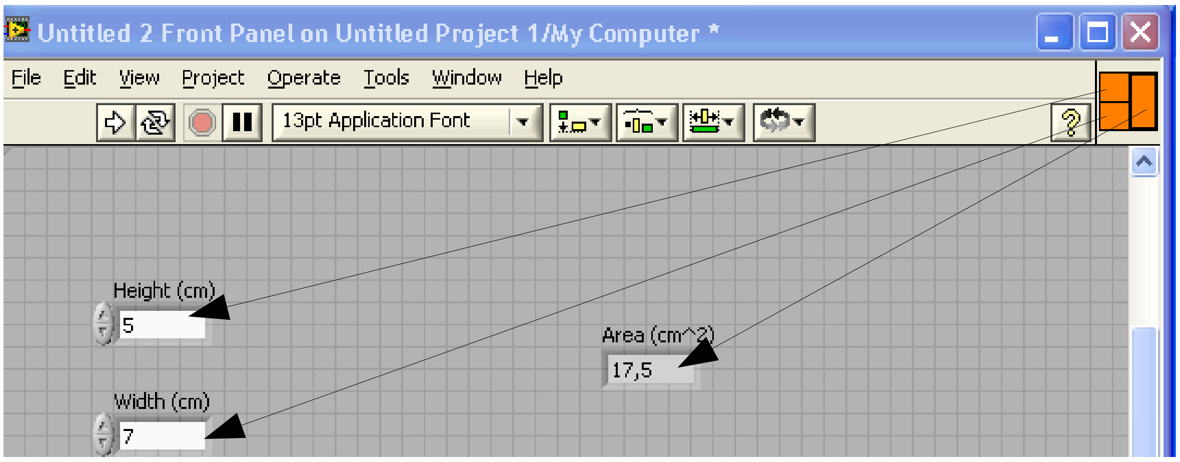

Untitled 2 Front Panel on Untitled Project 1/My Computer *
File Edit View Project Operate Tools Window Help
13pt Application Font
Height (cm)
5
Width (cm)
7
Area (cm^2)
17,5

Figure 4. A front panel of the subVI. The arrows show to which terminals the nodes are connected.

In figure 4 above of the subVI front panel, it is shown to which terminals the elements are connected. The height and width controls are connected to the two terminals on the left and the area indicator is connected to the terminal on the right. From the colour in the terminal it is possible to see what type the data is. Orange in this case represents a numeric value of type double.

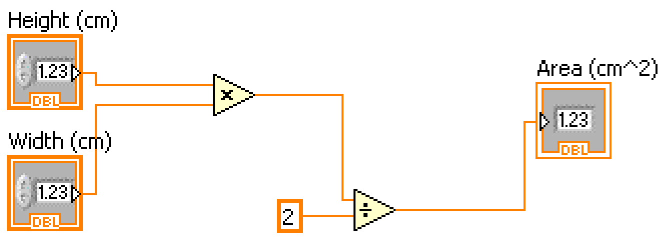


Figure 5. A block diagram of the subVI.

The block diagram in figure 5 above shows what the subVI does. When it is called the supplied height and width will be multiplied and the product is then divided by two and the result is output from the subVI.

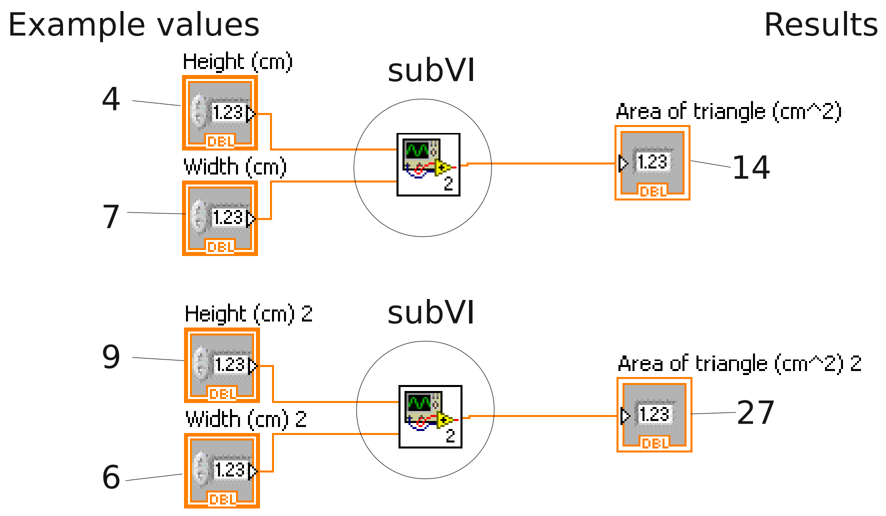


Figure 6. A block diagram of where the subVI is used.

The block diagram in figure 6 shows how to use the subVI. As can be seen from the figure, it works in principle in the same way as a function in text-based programming.

4 In-depth look at the assignment

4.1 Acquiring the signals

Getting the signals from the hardware to the software will be taken care of by another part of the project at Wärtsilä and will therefore be mostly outside the scope of this thesis. What is known about these signals is that there will be two signals that are received by a hardware card of some kind. As of the writing of this thesis there is no information of what the card in question will be. However, the signals received are: one voltage or current signal from the solenoid and one signal from the valve lift eddy-current sensor[4].

An eddy-current sensor operates with magnetic fields and is a non-contact device capable of getting measurements with a very high resolution of the position or change of position of any conductive target. One of the advantages of using Eddy-current sensors is that there can be non-conductive material between the sensor and the target without any effect on the measurement.

As mentioned there are two different kinds of signals that can be received from the solenoid and, in the case of this project, both of them are viable. There is, however, an advantage with the voltage signal as it will have sharp edges, while the current signal will have rounder edges due to the inductors and capacitors in the circuit.

After the signals have been acquired by the hardware on the computer, LabVIEW has drivers for common hardware[5] and thanks to this it is possible to get access directly to the signals when creating an application.

4.2 Measuring the times

To measure the times, the amplitude of both signals will need to be repeatedly sampled and the samples monitored. When the amplitude of the samples starts rising or falling or it reaches a specified value, it will trigger a counter that will count how many samples are between the triggers. The periods are defined by the following:

- T1: When the amplitude of the current signal starts to rise, the counter starts. The counting of the samples ends when the valve lift signal reaches 20% of the maximum value. The maximum value of the valve lift is the amplitude of the valve when it is fully opened.
- Time to start to close: When the amplitude of the current signal drops from the maximum amplitude, the counter starts. When the amplitude of the valve signal drops from the maximum amplitude, the counter stops.
- T2: When the amplitude of the current signal drops from the maximum value, the counter starts. When the amplitude of the valve signal lowers to 20% of the maximum amplitude, the counter stops.

Based on these definitions a period should look like figure 7 below.

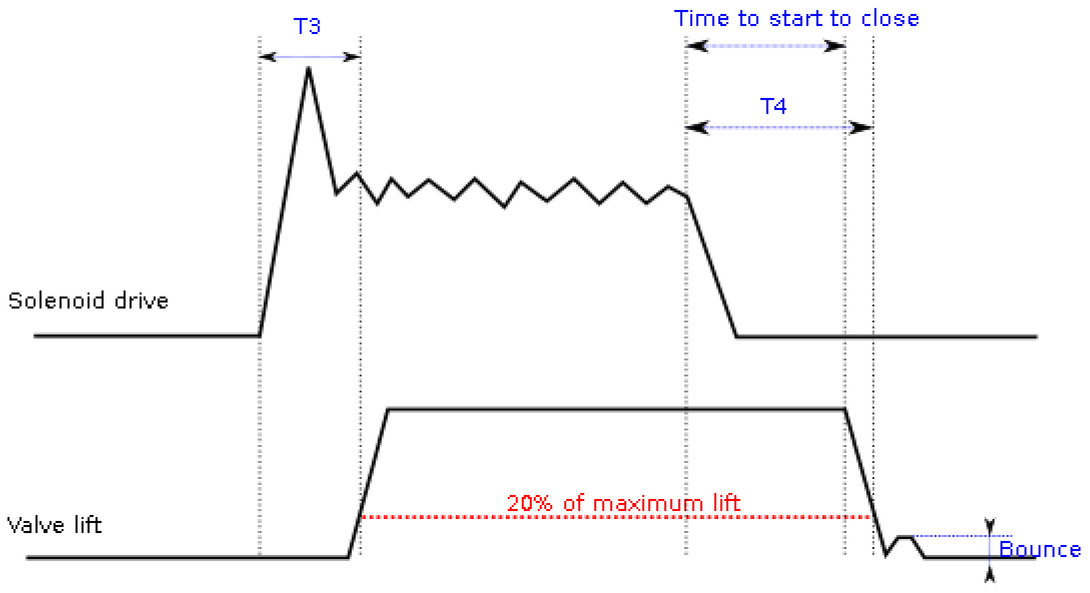


Figure 7. A model of a period with the times to be measured.

After the number of samples have been determined, the length of the interval will be calculated based on the sample frequency in Hz and the number of samples according to the following formula:

$$Time = \frac{Number\ of\ samples}{Sample\ frequency}$$

The formula to get the values directly in microseconds is then as follows:

$$Time(microseconds) = \frac{Number\ of\ samples}{Sample\ frequency} * 10^6$$

An alternative way to calculate the length of the intervals would be to use a timer. For this assignment a timer that can measure differences of at least 10 microseconds would be needed and this high precision presents a problem. In normal operating systems timers are only capable to provide, at best, a precision of 1 millisecond[6][7]. Thus to get the required precision, a real-time operating system would be needed. The problem with real-time operating systems is that they usually require special hardware that is usually more expensive than normal hardware as is the software developed for the system.

Since the measurement application will not need to take any real-time actions based on the data gathered, it was instead decided to make use of buffers in the signal acquisition hardware and calculate the times based on the number of samples, since a fixed sample frequency is used.

4.3 Measuring the bounce

To measure the bounce after the valve closes the peaks of the valve lift signal will be calculated with the help of a function that is included in LabVIEW called *Peak Detector VI*. The peaks will be recalculated for each period and then the amplitude of the second highest of these peaks should be the size of the bounce. In figure 8 below the bounce to measure is shown on the signal.

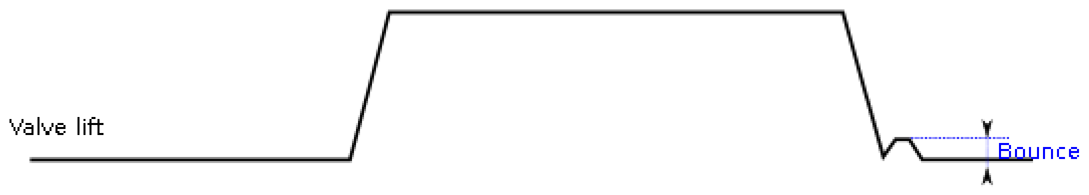


Figure 8. A model of the valve lift signal with the bounce.

5 Planning, research and practical implementation

5.1 Planning

A meeting was held at Wärtsilä where the requirements for the program were discussed. During this meeting a diagram was drawn up to explain the functionality desired from the program. This diagram was then translated into the UML use case diagram that can be seen in figure 9 below.

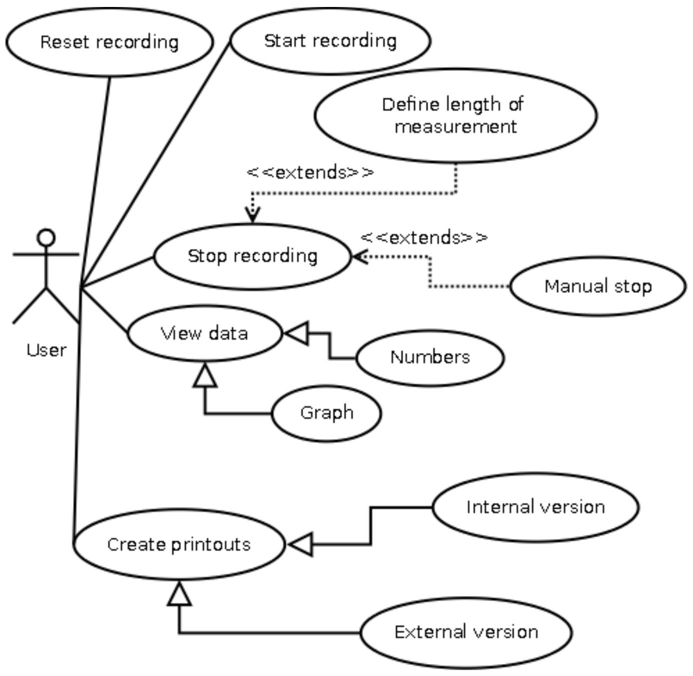


Figure 9. A simplified version of the use case diagram.

To explain the use case diagram shortly, a user should be able to:

- Reset the measurement.
- Start the measurement.
- Stop the measurement either manually or by defining an end condition.
- View the data as numbers and as a graph.
- Create a printout in two different versions, an internal version with all the information and an external version with less information.

In addition to these use cases, the plan would also be to program the application in such a way that it is as easy as possible to make changes to it at a later time. Because of this, the application makes use of subVIs and is separated into different parts that can be changed independent of each other.

5.2 The interface

5.2.1 Research and planning

After all the requirements had been taken into consideration, work on the user interface started. The first thing that was considered was which fields would be needed to show the data in a good and clear way. Because of the large amount of periods that would be measured, the suggestion was to create three different fields to show the data: a numeric field displaying the lowest value (amplitude or time), a numeric field displaying the average value of all gathered periods and finally a numeric field displaying the highest value. To ease the display of these fields, the expected boundaries should also be shown next to the measured values.

In addition to the numeric fields, a graph over the two available signals should also be shown. This graph would not be of great importance but would only be there in order to visualise the signals to the users. Because of this the graphs would only show the most recently collected period.

On the subject of updating the values in the user interface, a decision was made to update it with an interval of one second. The reason for this decision is that changes that would take place several times a second would not be possible to follow for the user. The length of the interval will still be tweaked as the testing of the application progresses.

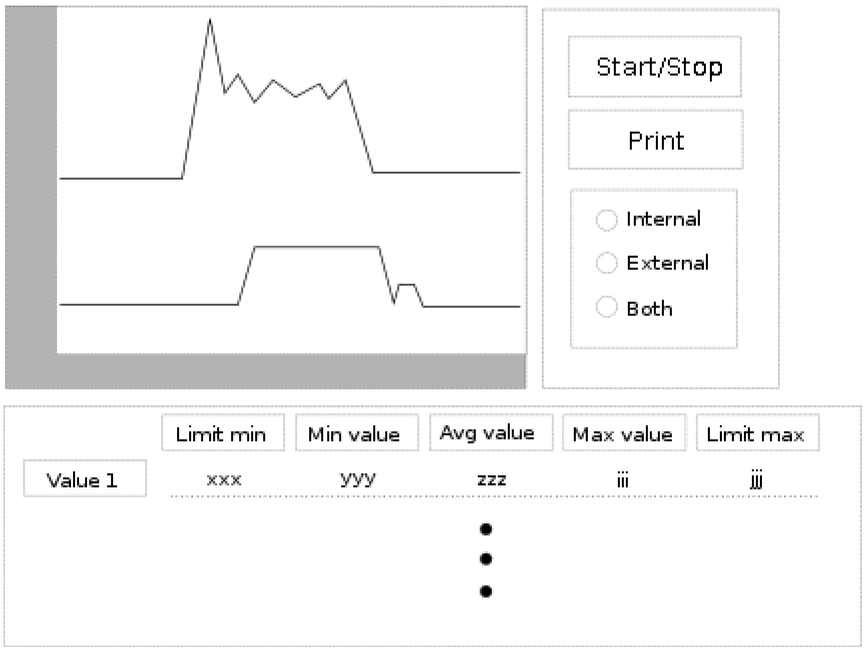
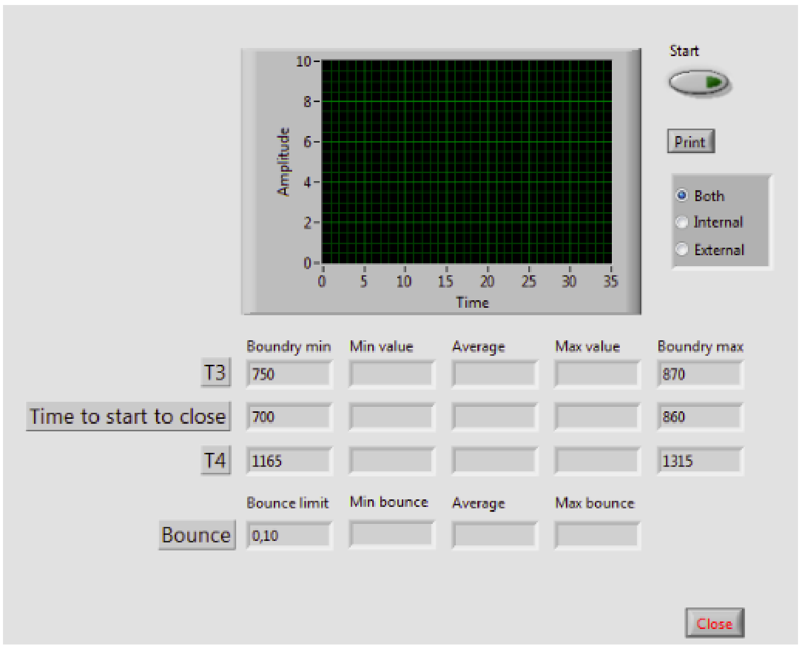


Figure 10. Sketch of the interface.

Figure 10 above is the first sketch of the user interface and is the model used when the interface was created in LabVIEW. The idea is to have the numeric fields beneath the graph in a tabular format for easy reading. The buttons next to the graph would be the start button, which is used to start the capture of the data, a print button with an option box where the print style would be defined and finally a stop button. The stop button could be replaced by making the start button into a toggle button.

5.2.2 Implementation

When implementing the user interface a decision was made to separate the interface from all other functionality. The signal acquisition and analysis is done in other files and the interface only fetches the data from the them. By implementing the interface separately it makes it far simpler to later change from the simulated signals to the real signals, since the user interface file will not need to be updated as long as the output variables stay the same.



[✓ Both] Internal External

	Boundry min	Min value	Average	Max value	Boundry max
T3	750				870
Time to start to close	700				860
T4	1165				1315

	Bounce limit	Min bounce	Average	Max bounce
Bounce	0,10			

Figure 11. A first prototype of the interface in LabVIEW.

Based on the earlier sketch the interface was created trying to follow it as closely as possible. See the result in figure 11 above. A close button was added to make it possible to abort the execution of the program without closing the window. This might be useful if the user wants to restart the application without restarting the LabVIEW run-time environment. Except for this button the prototype interface is the same as the sketch.

The limit values are defined as constant values in the *Block Diagram* of the interface VI. This is done to make it possible for personnel at Wärtsilä Finland – Services to make changes if necessary without giving the same rights to normal users. The downside of this is that when the limits are changed a new version, of the program has to be compiled and distributed to all the installations that use the application and will need the changes.

5.2.3 Progress

The user interface itself is finished unless some further feedback is received with suggestions on how to change it. All the fields are connected to the data flow and the start and stop button does what it is supposed to do. The one thing that will still need to be implemented in the user interface is the print functionality but this will be discussed in a later chapter.

In addition to the user interface, the engine is integrated into the same VI. This engine is what drives the application and decides when a new signal will be fetched from the signal acquisition subVI and then sends it on to the signal analysis subVI. The engine also takes care of when the user interface should be updated and it also keeps tabs on all the previous measurements and calculates the minimum, average and maximum values based on the list of values.

5.3 Signal acquisition and analysis

5.3.1 Planning and research

While getting the signal from the hardware to the application will mostly be taken care of by drivers available to LabVIEW, analysing the signal might require a bit more than the raw signal.

When measuring the intervals *T3*, *Time to start to close* and *T4*, it is important to know when signals rise or fall. One of the problems with the solenoid voltage graph is that it contains noise that will make it difficult to detect when the signal either rises or falls. With a smooth graph it would be very easy to detect peaks and valleys of the signal and, based on these, get the starting samples when the signals rise and fall. However, when trying to detect peaks and valleys on an unfiltered and non-smoothed signal, a problem occurs that can be illustrated by the following picture:

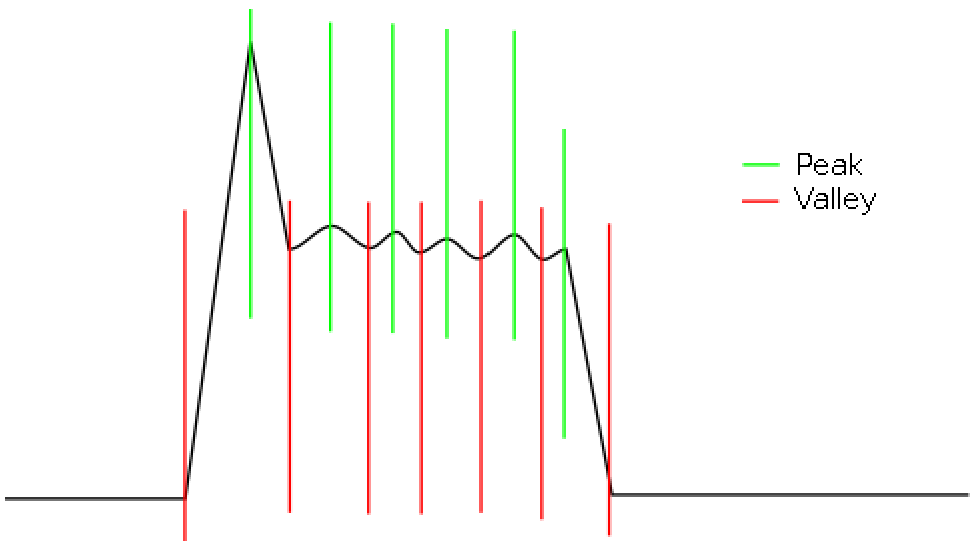


Figure 12. Peaks and valleys in an unfiltered and non-smoothed solenoid drive signal.

As can be seen in figure 12, multiple valleys and peaks could be found in an unfiltered and non-smoothed signal and it would be non-trivial to decide which peaks and valleys are the ones that are of interest to the application. Now, if the signal could be smoothed or filtered the result would instead be similar to the signal in figure 13 below. The signal could be filtered with a low-pass filter[8] which should take care of most of the noise and then, if necessary, it could be smoothed out a bit with a fitting smoothing algorithm depending on what the signal looks like after it has been filtered.

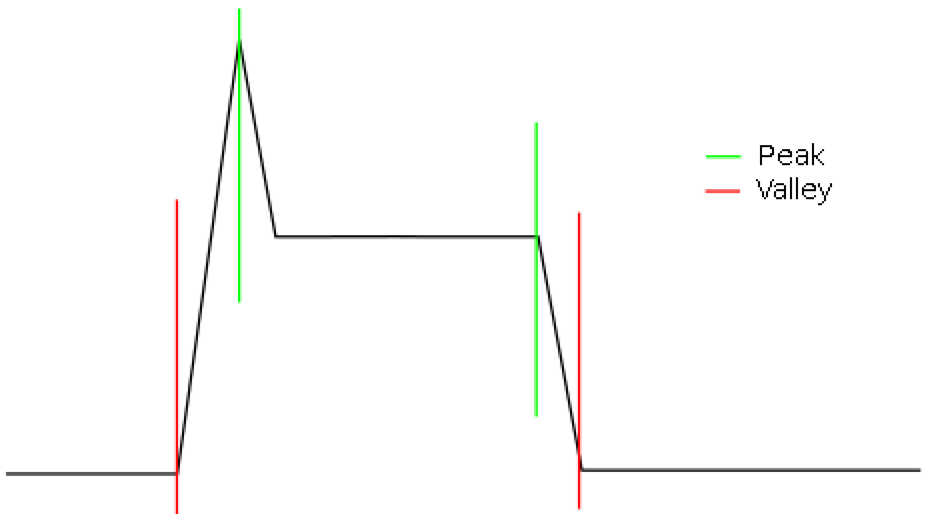


Figure 13. Ideal case of peaks and valleys on a filtered and smoothed signal.

As can be seen from figure 13, the number of peaks and valleys would be drastically reduced and it would be easy to get the correct sample for when the signal rises or falls. It should, however, be noted that it will probably not be possible to get as ideal a signal as that in figure 13, but as long as the number of peaks and valleys is reduced and the differences in the amplitude between samples lowered, it should make finding out where the signal rises and falls far easier.

The result of using a low-pass filter and a simple smoothing can be seen in figure 14 below. The first graph shows the generated signal with added noise and the second graph shows the signal after it has been filtered and smoothed.

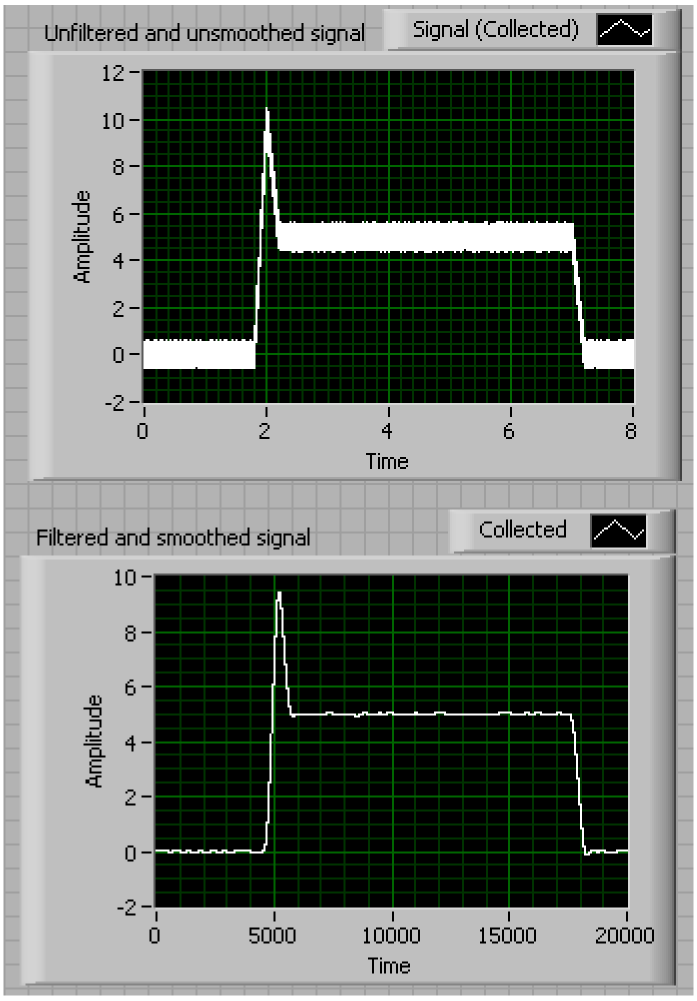


Figure 14. An example of the generated signals in LabVIEW.

The smoothing algorithm used in the above example is:

$$S_j = \frac{Y_{j-1} + Y_j + Y_{j+1}}{3}$$

where Y_j is the current sample, Y_{j-1} is the previous sample and Y_{j+1} is the next sample.

To get the required values from the valve lift signal, the peak and valley detection tool in

LabVIEW is used to detect the amplitude of the bounce as per figure 15 below.

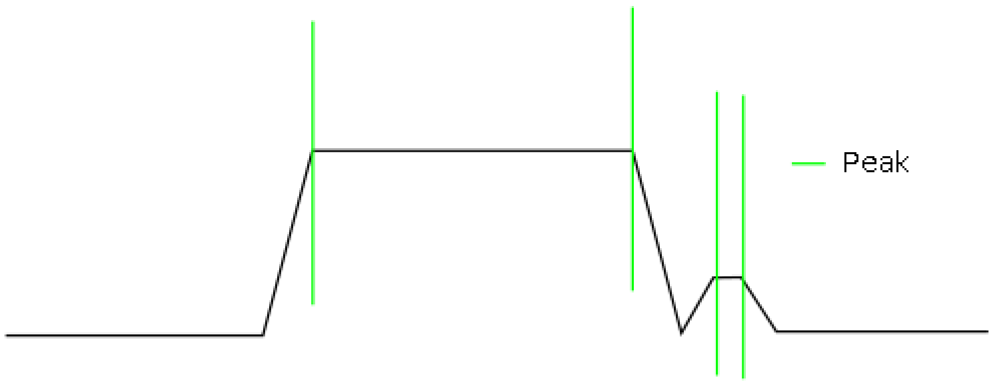


Figure 15. Peaks in the valve lift signal.

Now that the procedure for one period has been defined the next problem would be how to define a period. Right now the period is defined in the application by the following rules:

- When starting a new measurement or a period ends, a new period begins.
- A period ends when:
  - a) The solenoid voltage signal has risen from the off state and dropped back down to the off state once (see figure 16 for a reference of what is meant by the states).
  - b) The valve lift has risen and dropped twice.

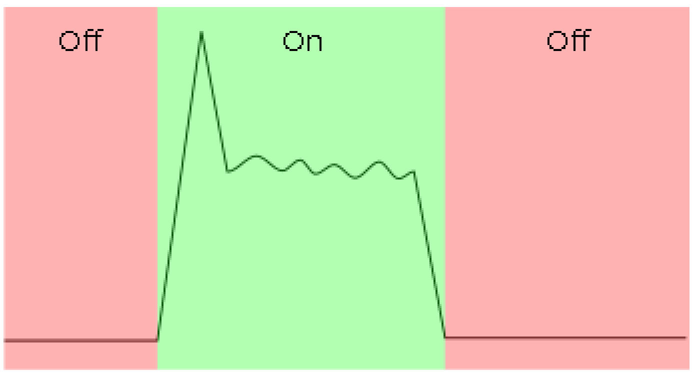
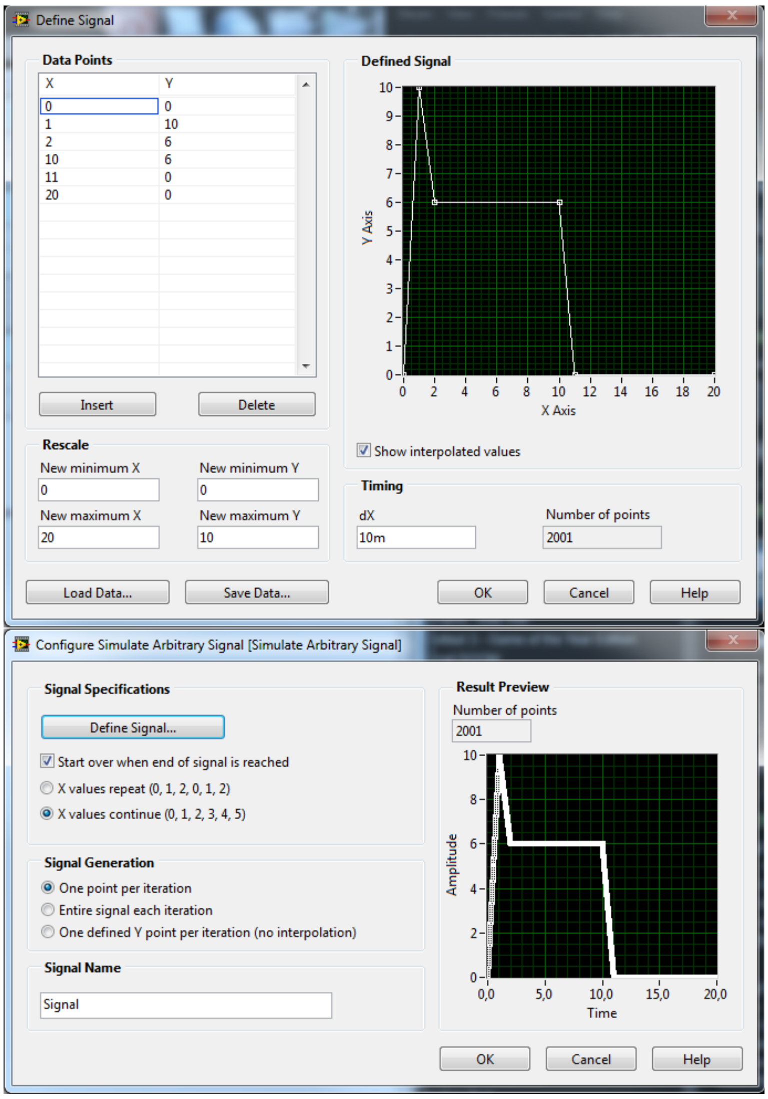


Figure 16. Solenoid drive signals' on and off states.

To be on the safe side, at the end of the period a number of samples could be included to make sure that the whole period will be included.

5.3.2 Simulated signal

When the project began there was no real implementation of the test bench yet, and thus the signals had to be simulated for the start of the project. To simulate the signals some built-in functions in LabVIEW were used to simulate arbitrary signals. What is meant by this is that to generate the signals some points are specified in a table in a VI and then a sample frequency is specified in the same VI. This then results in a signal that is repeatedly generated and output one sample at a time. It is also possible to output all samples at the same time but one sample at a time is closer to reality so that is what the VI is configured to generate. These arbitrary signals could be defined by drag-and-drop or by entering key values into a table, after that the timing could be defined and based on that the signal would be created. See figure 17 below for the interface of creating an arbitrary signal in LabVIEW.


Define Signal

Data Points

X	Y
0	0
1	10
2	6
10	6
11	0
20	0

Rescale

New minimum X: 0	New minimum Y: 0
New maximum X: 20	New maximum Y: 10

[✓] Show interpolated values

Timing

dX: 10m	Number of points: 2001

Configure Simulate Arbitrary Signal [Simulate Arbitrary Signal]

Signal Specifications

[✓] Start over when end of signal is reached
○ X values repeat (0, 1, 2, 0, 1, 2)
● X values continue (0, 1, 2, 3, 4, 5)

Signal Generation

● One point per iteration
○ Entire signal each iteration
○ One defined Y point per iteration (no interpolation)

Signal Name: Signal

Result Preview

Number of points: 2001

Figure 17. The interface for defining an arbitrary signal in LabVIEW.

The signals used during the first stages of testing were using a period of 3000 samples because generating two signals and analysing them was very heavy on the performance of the test computer. Some noise was added to the solenoid voltage signal but otherwise no signal manipulation was used beyond the standard LabVIEW generators.

5.3.3 Implementation

When developing the signal acquisition and analysis parts of the program, a decision was made to split them up into different subVIs to make different parts of the application more independent. One subVI was created to take care of the signal generation and another one to analyse the signals. This will be helpful later when the software will be adapted to the test bench by replacing the signal acquisition subVI with the real signal and the analysis part should at worst only need some small adjustments.

The signal acquisition subVI will, beyond capturing and filtering/smoothing the signal, gather enough samples for a full period and when this period is gathered it will be output by the subVI. By outputting the period, while it will generate more work to replace the subVI, it will at the same time lessen the amount of work needed in the analysis subVI.

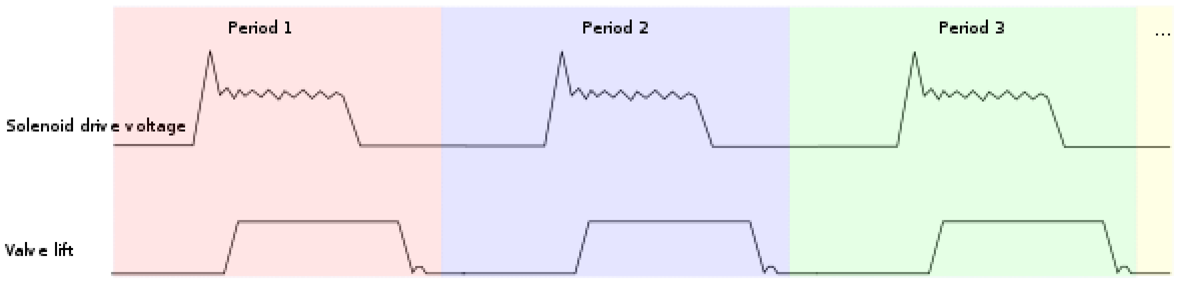


Figure 18. Illustration of the output from the signal acquisition subVI, every period is a different output value

After a period of the signal has been gathered and the value has been output from the subVI, it will be sent to the signal analysis subVI. Figure 18 above shows an approximation of what the output should look like. Every period would be a separate output value that will be sent to the analysis subVI.

In the signal analysis subVI, one value for each of the sought after variables will be calculated. By finding the peaks and valleys from the solenoid drive voltage signal, the program should be able to figure out:

a) When the signal starts to rise.

b) When the signal starts to fall.

When using the built-in function to find peaks and valleys in LabVIEW, two values will be returned for every peak/valley, namely the position on the time axis and the amplitude of the peak/valley. Depending on how well smoothed the signal is, it might be necessary to go through all the peaks and valleys and figure out which are the ones that we want, in other words which ones do not exist due to noise or other disturbances. To do this, an assumption will be made that in each period there will only be one time when the signal rises and one time when the signal falls. Of course the signal will fall twice because of the spike at the beginning of the signal, but that spike will be ignored. The next step to finding the correct points where the signal starts to rise and fall would then be to compare the amplitudes of the different peaks and valleys. The difference in amplitude between two subsequent points should be noticeably higher than the rest and would thus indicate that the signal has either risen and fallen between two points.

In addition to the peaks and valleys from the voltage signal, the valve lift signal would be searched through to find all the indexes where the amplitude of the signal crosses a specified limit. According to this thesis this limit would be 20% of the maximum amplitude of the valve lift signal. When all the points have been gathered, getting the number of samples the different intervals consist of is a simple subtraction.

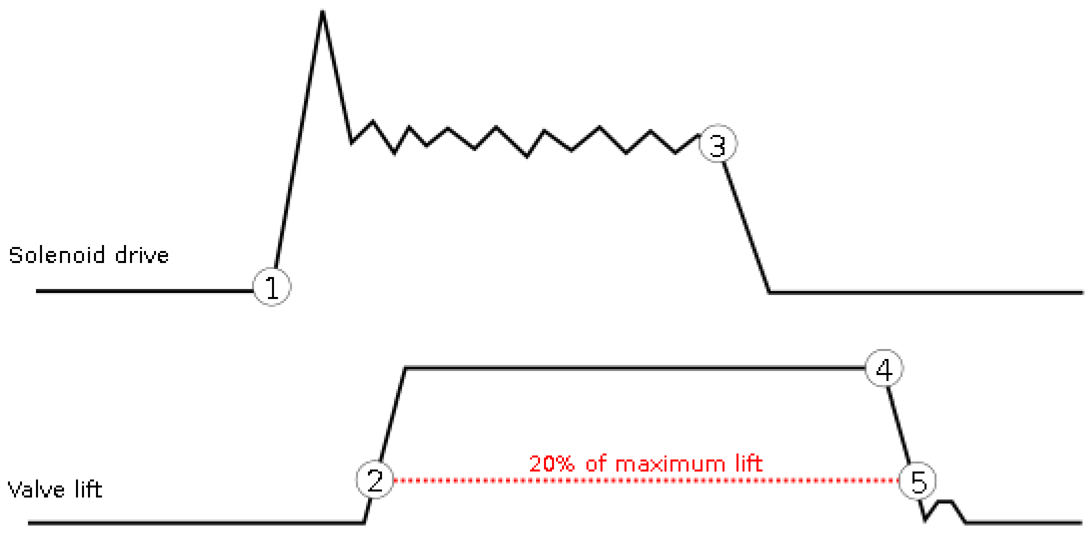


Figure 19. Signals with the points of interest numbered.

In figure 19 above the numbers represent the indexes at which the boundaries of the intervals are. Based on these indexes then we get the length of the intervals in samples by the following:

- Index of 2 - Index of 1 = length of interval *T3*
- Index of 4 - Index of 3 = the length of interval *Time to start to close*
- Index of 5 - Index of 3 = the length of interval *T4*

Once the lengths of the intervals are known in terms of number of samples, they are converted to a time interval in microseconds.

Getting the amplitude of the bounce is easier than getting the length of the intervals, since only the valve lift signal needs to be analysed and to get the bounce it is only necessary to get the peaks of the signal. After all the peaks have been found, a simple limit is applied to the results of about 90% of the maximum amplitude of the signal. The lower peak that remains is the bounce. The reason for this limit is that the valve should open the same amount every time. However, just to be on the safe side, a margin is added.

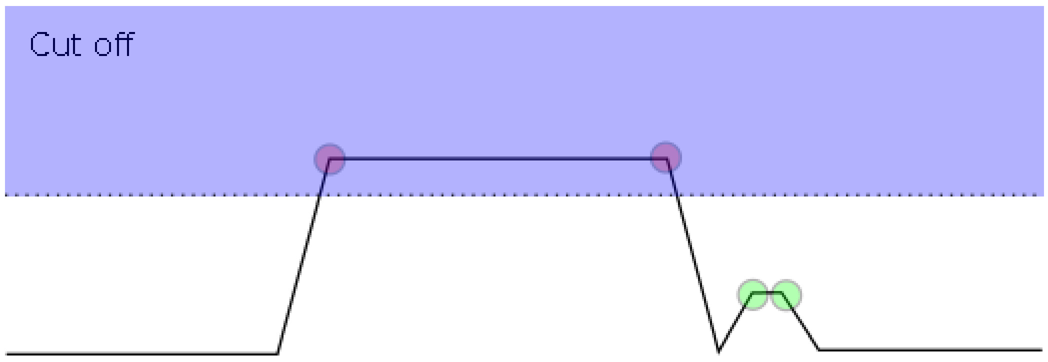


Figure 20. Valve lift peaks with limit

As can be seen from the above figure the two peaks with higher amplitudes would be ignored and the bounce would have two peaks. In the end it should not matter which one of these peaks is used since they should both have the same amplitude. If, however, the real data proves this to be incorrect, it is a small matter to simply compare them and set the higher of the amplitudes to be the value of the bounce.

5.3.4 Problems

The biggest problem with the program is finding out where the solenoid drive voltage starts to rise and when it starts to fall. It will most likely be necessary to adjust the filtering and smoothing of the signal very carefully since even an error of one sample off would lead to an inaccuracy of 10 microseconds, which can be quite significant because of the precision required. It might very well be that the method of getting the index of when the signal starts to rise and fall will need to be changed at a later stage of development.

5.4 Printing the data

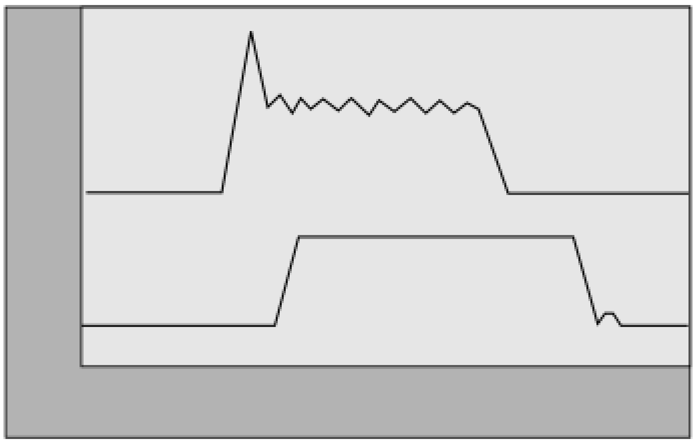
5.4.1 Research

At the start the intention was to create PDF files for printing from LabVIEW, but after some research it was discovered that printing reports to PDF format was quite non-trivial and the only solution available was to rely on third-party libraries that had no direct connection to National Instruments. The problem with this would be that every installation where the application was run would have to install these extra libraries and there might be problems with licensing depending on which library would be used. Instead of the PDF solution another solution was searched for and was found in the functionality to print out select elements from the *Front Panel* to an HTML file.

As mentioned above when creating the HTML reports it is possible to define which elements that should be visible in the final report. This is very useful since two different kinds of reports need to be created. To begin with the plan was to have two different *Front Panels* or maybe to hide and show different elements on the *Front Panel* in the code when printing, but thanks to this functionality of being able to create reports independently from the *Front Panel* these ideas are not needed. Finally when an element is added to a report it is also converted into an image of either JPG or PNG format so that it should look the same on all devices without having to worry about the available fonts on the device it is viewed on, or any possible themes that could change the appearance of the elements.

5.4.2 Planning

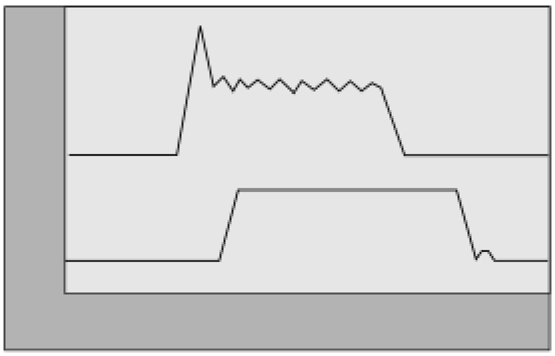
The assignment asks for two different kinds of reports, one format intended for internal use in Wärtsilä and one format intended to be shown to customers so that they can see the results of the measurements. The internal report should contain all the values and limits while the external report would only need to show if the tests have passed or failed.



T3 test:	PASSED
Time to start to close test:	PASSED
T4 test:	FAILED
Bounce test:	PASSED

Figure 21. Sketch of the external report.

Figure 21 above shows an example of what the external report could look like. The labels would be static text fields and the PASSED/FAILED text would be based on a strict check of all the values so if (even) one measurement exceeds the specified limit, the text would show up as FAILED. The graph would only serve as a visual aid to represent the signals but would not really serve any practical purpose beyond that.



	Min	Average	Max	Limit min	Limit max
T3 test:	780 µs	790 µs	800 µs	750 µs	870 µs
Time to start to close test:	790 µs	800 µs	810 µs	700 µs	860 µs
T4 test:	1300 µs	1320 µs	1330 µs	1165 µs	1315 µs
Bounce test:	4 %	6 %	7 %		10 %

Figure 22. Sketch of the internal report.

In figure 22 above an example is shown of what an internal report could look like. Instead of only having a PASSED or FAILED text, it should show all the values from the *Front*

Panel in the application and also the limits so that it is easy to see which values are either too high or low. The limits are defined as constants in the *Block Diagram* of the interface VI and are fetched at the creation of the report.

6 Results

6.1 Data flow between the virtual instruments

The data flow between the different parts of the program is a very important part and defines how the application will work in the final product. The data flow has been designed to minimise the amount of work the separate parts will have to do to work, meaning that the part that gets the signal will only get the signal and output a period, the signal analysis part will only take two periods of the signals as input data and output the resulting key values.

The signal acquisition subVI will capture the raw signal and then convert it into one period that is then output to the interface VI. In the case of this program the interface VI also contains the so-called engine code. The engine code is the piece of the application that connects all subVIs together with the interface and pretty much controls the data flow on a higher level.

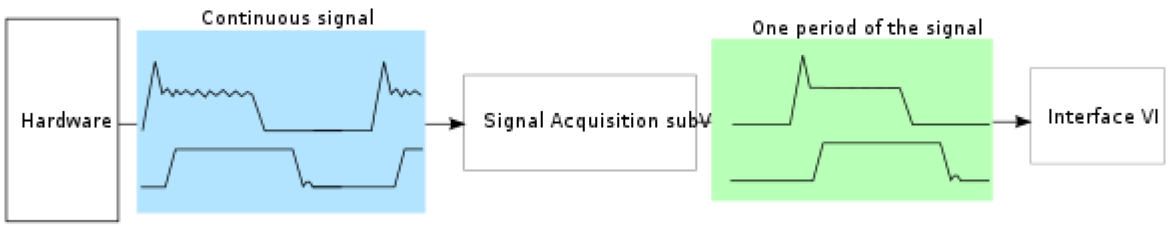


Figure 23. Data flow from hardware to the interface.

To start with the first data will come from the hardware into the signal acquisition subVI. This data consists of two raw signals that are subsequently collected into a period and smoothed out as necessary. After the signals have been prepared, one period from the signal is sent to the interface VI and the signal acquisition subVI goes in standby and waits for the interface VI to request a new value. The flow of the data in this process has been visualised in figure 23 above.

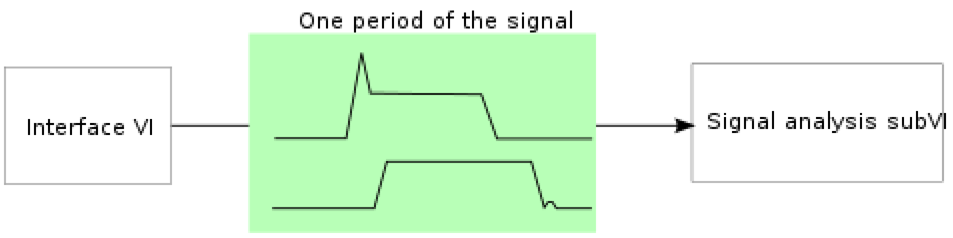


Figure 24. Data flow from the interface to the signal analysis instrument.

After the interface VI has received a period of the signal, the period will be sent to the signal analysis subVI. When the signal has been received by the analysis subVI, it will get all the measurements that are of interest. The data flow of this process is visualised in figure 24 above.

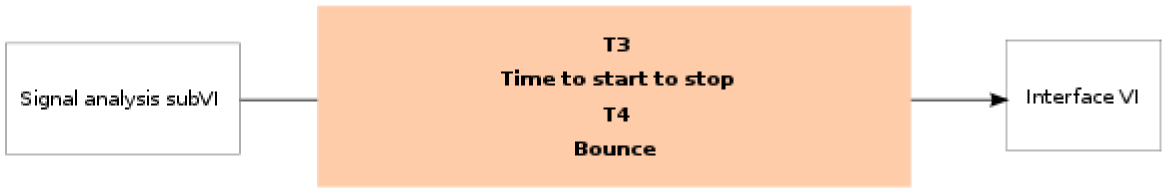


Figure 25. Data flow from the signal analysis instrument back to the interface.

When the analysis of the signal is done the values will be sent back to the interface and the signal analysis subVI will wait for the next period. When the interface receives the new values, they will be added to a list of all the previous values and the minimum and maximum value are fetched from the array. The data flow of this process is visualised in figure 25 above.

There is unfortunately not a built-in function that can get the average value so to get this the list will be iterated through and all the values added together and finally divided by the number of values in the list.

In figure 26 below, the block diagram, the core of the program, has been defined. In the block diagram, functions are defined for how the key values are split up into minimum, maximum and the average values. One important difference compared to what the final program will be like is that in the block diagram an array of all the previously collected key values is sent to the analysis function, while in the final program I will handle the array in the program that is defined in the block diagram below.

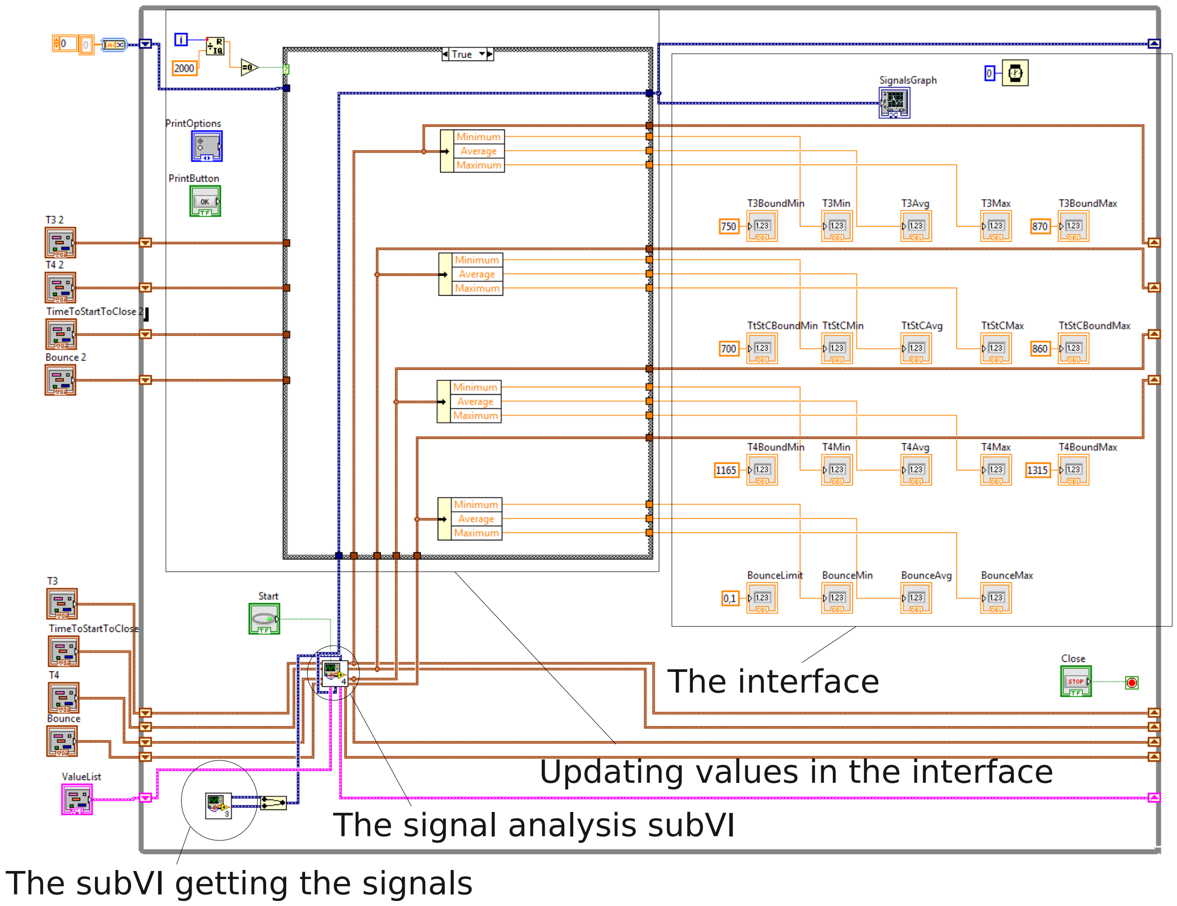


Figure 26. The block diagram of the core functionality in the measurement application.

7 Conclusions and discussion

During this project I have learned a lot about LabVIEW and its capabilities. The differences between a graphical programming language and a text-based programming language are quite large and dataflow programming was a new concept to me when I started. When I discussed the differences between the two different approaches to the programming languages with an engineer from National Instruments, he told me that programmers usually have a harder time to get accustomed to the process of creating programs in LabVIEW than non-programmers. One of the things that caused me quite a bit of trouble was learning to abandon the use of variables in favour of the shift registers that exist in LabVIEW. A shift register makes it possible to save values inside loops between iterations, pretty much like a variable.

A difficult part in the project was to generate signals that would be close to what I would expect to get in a real situation. Since the signals are so important and, if the signals are even a bit off, the results from the analysis can be completely different than to what I would like. All this led to me separating the signal acquisition functionality from the rest of the program as much as possible so that it would be easy to implement the real function for getting the signals later.

One of the things I did not consider in the beginning was the difficulty of getting accurate times when measuring times less than 1 ms. At one point I considered using a real-time operating system to calculate the time but, because of the cost of the hardware and the additional software that would be necessary to do this, I later decided against it.

8 References

[1] Petrol Injectors
http://www.crypton.co.za/Tto%20know/Actuators/injectors.html (Fetched: 17.04.2012)

[2] What is LabVIEW?
http://www.ni.com/labview/whatis/ (Fetched: 17.04.2012)

[3] Tutorial: Block Diagram
http://zone.ni.com/devzone/cda/tut/p/id/7565 (Fetched 26.04.2012)

[4] Eddy-current sensor overview http://www.lionprecision.com/inductive-sensors/index.html (Fetched: 12.04.2012)

[5] Hardware Integration with NI LabVIEW
http://www.ni.com/labview/whatis/hardware-integration/ (Fetched: 16.04.2012)

[6] Microsoft, Timers, Timer Resolution, and Development of Efficient Code
http://download.microsoft.com/download/3/0/2/3027D574-C433-412A-A8B6-5E0A75D5B237/Timer-Resolution.docx (Fetched: 12.04.2012)

[7] Bovet, D.P. and Cesati, M. (2005). *Understanding the Linux Kernel.* 3rd Edition pp. 287 – 290.

[8] All About Circuits, Vol II AC – FILTERS Low-pass filters
http://www.allaboutcircuits.com/vol_2/chpt_8/2.html (Fetched: 12.04.2012)