

Chapter 1

Methods for measuring

The measurements performed at Moholt and in Bergen required accelerometers to be set up at relevant locations. No previous hardware or software set-up was available, so a lot of time was spent making the equipment ready for use. At the Department of Structural Engineering, there was an interest in learning how this type of equipment can be utilized for future projects. Preparing for measurements included installing software and learning LabVIEW programming, as well as configuring hardware.

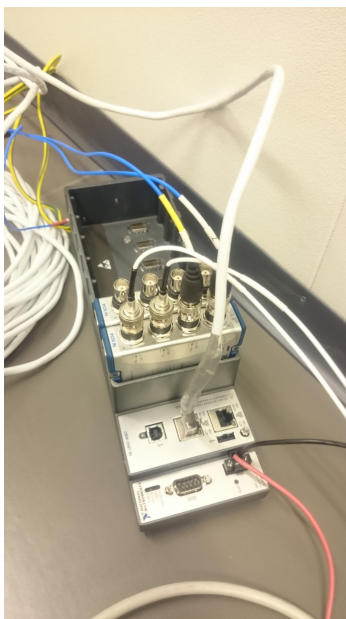


Figure 1.1: CompactRIO

1.1 Hardware

The following hardware was used to measure accelerations:

- NI 9067 CompactRIO Chassis (cRIO)
 - Kistler K-Beam Triaxial MEMS Capacitive AC Accelerometer, model 8395A2DOATTA00 (Kistler)
 - PCB Piezotronics accelerometer, model 356A16 (PCB)
 - NI 9234 dynamic signal acquisition module
 - Data and power cables
 - Coaxial extension cables
 - NI 9144 EtherCAT slave chassis
 - Ethernet cable
 - USB A to B cable
- Power supply set to 24V
- Computer with LabVIEW installed
- Special screws needed to mount the accelerometers to the structure.
- In addition some practical tools were needed; such as a drill, screwdrivers and tape.

The accelerometers were connected to the acquisition modules through coaxial cables specially made for minimizing noise. On both days at Moholt, coaxial extension cables were used as well [7], since only the cRIO chassis was available at the time. The acquisition modules were mounted directly to the chassis. The chassis and the Kistler accelerometers were both powered separately with a 24V power source[3, 5] while the modules and PCB accelerometers [6, 4] were powered through the chassis. The modules have four channels. Each channel is able to acquire data from one accelerometer in one direction. In order to communicate with the chassis, a computer connected by a USB cable was used. It is also possible to communicate either with an Ethernet cable or over the internet. Both communications were considered at some point, but they were found unnecessary for this thesis. In order to properly acquire the torsional mode shapes each accelerometer was placed as far away from the expected torsional centre as possible, on opposing sides.



Figure 1.2: Kistler accelerometer

The two types of accelerometers work in two different ways; the Kistlers oscillate around zero [3], while the PCBs starts to oscillate around a non-zero value [6]. The PCBs are after a couple of seconds automatically de-trended until they oscillate around zero. In post-processing of the data, the mean value was subtracted from each channel as a simple form of de-trending. The two types have different preferred frequencies where they exhibit optimal measurement accuracy; the Kistlers can sense acceleration successfully down to 0Hz, while the PCBs can only sense accurately down to 0.5Hz. This minimum value for the PCBs is a bit close to the 1Hz component expected from some of the measurements, and this may affect the amount of noise in the results. When the time series were watched as they unfolded, only noise could be seen with the naked eye, unless some local effect, like a door shutting nearby, caused a large spike in the amplitude.

The output from the accelerometers is in volts and therefore has to be converted into meters per second squared. The PCBs and the Kistlers have different constant scaling factors; the PCBs have a scaling factor of $98,1m/s^2/V$ [6] while the Kistlers have a scaling factor of $4,905m/s^2/V$ [3].

1.2 Software and hardware communication

How the different parts of the equipment communicate with each other is a highly important part of choosing the most accurate and efficient programming. Acceleration data is obtained when voltages are induced inside the accelerometers. This is an analog signal that is sent via a coaxial cable into the modules converting the analog signal into digital samples of data. This data is further processed by VIs on the Real-Time Target (the cRIO). A Real-Time Target is defined as having sufficient speed to run programs and acquire data in a way that is perceived as immediate. The processed data is then written to files on a local or external storage unit.

1.3 Connecting the cRIO

After installing the software and drivers for LabVIEW and cRIO, the cRIO target was imported into a LabVIEW project. At this point either *Scan Engine* or *FPGA* was chosen depending on the requirements for the VI in question. The Scan Engine and FPGA are two different ways of gathering and processing data.

1.3.1 Scan Engine

Scan Engine is the mode in which LabVIEW automatically uses a built-in function for side-stepping FPGA and it is possible to access the input and output modules directly in the VI. It can be set to an acquisition rate of up to 1kHz, and the modules are automatically timed and synchronized. Since the maximum acquisition rate is sufficiently higher than the relevant frequencies of structural dynamics, the scan engine is a suitable choice as long as aliasing effects and sampling rates are under control. However, it is not certain what rate the modules convert the analogue signal into digital samples when using Scan Engine. This reduces controllability in regards to aliasing. One advantage by using Scan Engine, is that it utilizes the Real-Time processor on the Real-Time Target. Compiling VIs and controlling every data point becomes remarkably simple. For instance, it is easy to timestamp every single data point in order to keep track of the sampling rate.



Figure 1.3: Channel inputs in Scan Mode

During the two trips to Bergen, scan mode VIs were used to acquire data from 8 channels simultaneously. However, the VIs on the two different trips were very different. The first two VIs used a for loop to acquire an array of data. This for loop was supposed to run every millisecond and save all the data every n th iteration. On the second trip a VI using two separate loops was used; one timed loop synchronized to scan engine and one while loop. The timed loop was used to acquire data and send it through a FIFO (explained in Subsection 1.4.6) to the while loop which saved the data to disk. This method was chosen as it separates the two processes into two loops that can run at different speeds. This is highly recommended to avoid loss of data points [1].

1.3.2 FPGA

Field-Programmable Gate Array (FPGA) is a silicon-chip that is reprogrammable. It provides higher performance and stability than the Scan Engine, and is useful for rapid input/output processing. The maximum sampling rate can be up to 51.2kHz for the NI 9234 modules. Possible sampling rates for FPGA are given in Equation (1.1), and it can be seen that $n=31$ give the lowest possible sampling rate at approximately 1652Hz. FPGA is however more cumbersome and complicated than the Scan Engine and it is not possible to store data directly. In order to store data, the FPGA VI communicates via a FIFO to a VI on the Real-Time target. Although this makes the programming a bit more expansive and the compilation of the code takes typically several minutes, the end result is worth it as the FPGA produces a much more stable sampling rate and therefore also cleaner and more accurate results. In the FPGA VI, no solution on how to timestamp every data point was developed. Therefore, every 280th point was timestamped in the real-time host VI.

$$f_s = \frac{13.1072 * 10^6}{256 * n}, n = 1, 2, \dots, 31 \quad (1.1)$$

On the second trip to Bergen a simple form of FPGA programming was used to acquire data. This VI used a for loop within a while loop in FPGA, where the for loop included the FIFO which read the data channel by channel. The FPGA settings and module start-up was set before the while loop and then the data was acquired within the while loop. The sampling rate was set to 1652Hz. The reason behind setting it as low as possible was to make the output files more manageable. Also, the frequencies of interest is a lot lower than the sampling frequency so it does not affect the results. On the Real-Time target a while loop was used to extract the data from the FIFO and store it. Before the while loop, the FPGA target was specified and start-up commands were sent to the FPGA VI.

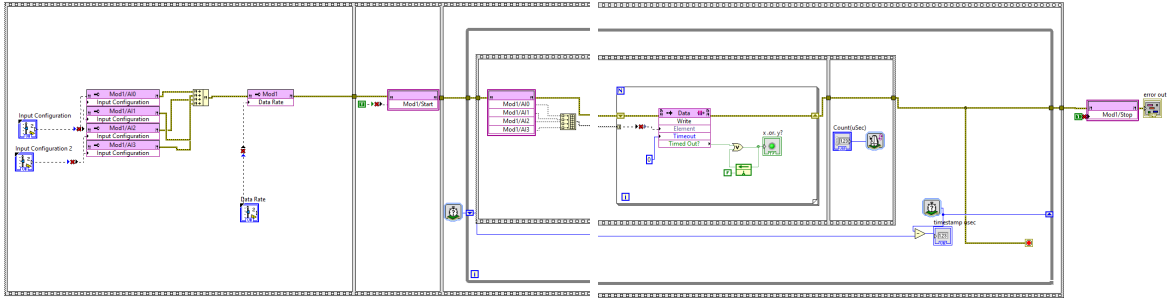


Figure 1.4: VI on FPGA target

1.3.3 Expansion Chassis

When evaluating the dynamic properties of the buildings, a set of accelerometers in different parts of the building was needed in order to acquire mode shapes. These accelerometers were set so far apart from each other that using coaxial extension cables would have been needed. This could possibly produce a lot of noise. It was therefore decided that a slave chassis should be used. The slave chassis was connected to the cRIO with an Ethernet cable. In the limited time available between the first and second trip to Bergen, no FPGA VI utilizing both the cRIO and the slave chassis was made. Therefore, the slave chassis was only used in scan mode [2].

1.4 LabVIEW programming

In this part a proposed solution for many functions in LabVIEW is discussed, all these has at some point been considered as necessary functions, but some of them has later been deemed unnecessary. Nevertheless, their function and solutions will be discussed here, as it may save time for someone in the future. These solutions are only proposed solutions that has been shown to work, but may not be the best solutions. All solutions are based on the ones provided by NI [1].

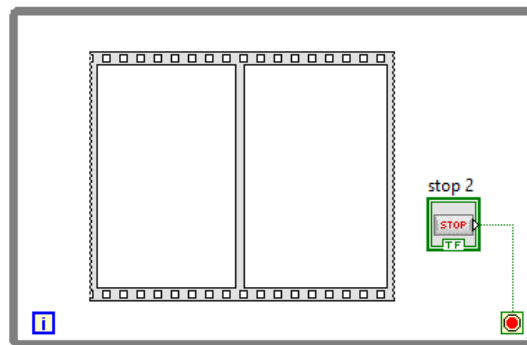


Figure 1.5: Flat Sequence Structure inside a While Loop

Everything that happens in the LabVIEW VI that is supposed to run continuously should be placed inside a While Loop. A While Loop will run until it is stopped, either by aborting or if a True is fed to the Loop Condition. Usually, processes in LabVIEW goes from left to right, but different timing and

placements may complicate this. To achieve more control over what happens first; a Flat Sequence Structure can be used. A Flat Sequence Structure is split into compartments and it is ensured that the whole compartment is finished before it moves on to the next. This movement is also from the left to the right.



Figure 1.6: Waveform Graph

All places where there is an error out there may be placed an indicator. This is very useful when something goes wrong, especially when working with files. It should also be noted that placing Waveform Chart or Waveform Graph as indicators at strategic locations gives an indication as to what is going on. These charts and graphs has to be implemented in the front panel before they are visible in the block diagram. The Waveform Chart needs continuous data while the Waveform Graphs needs a series of data.

1.4.1 Gathering Data

When the cRIO is connected in the right manner, some form of loop is needed to acquire more than one point of data. The loop that should be chosen depends on the form in which data is required to be collected. The most important thing to get right when acquiring data is to synchronize the different channels and have a controlled sampling. In scan mode, the synchronization was done automatically by the scan engine, while in FPGA the synchronization was done manually by starting all channels at the same time. Keeping track of the sampling rate was done by timestamping as many points as possible.

1.4.2 Creating and saving files

There are several ways to store data in files in LabVIEW, but the way chosen here is to use the tdms functions. The tdms-format is the preferred format for instruments made by NI, as it is their own format. This way saves data into .tdms files which can be opened in Excel by using a tdms-importer. The tdms-importer is included when installing LabVIEW. Excel is limited in the number of datapoints it is able to open so a Matlab code was used for obtaining the data from the longer time series. In order to save a .tdms file, TDMS Open, TDMS Write and TDMS Close needs to be used. The TDMS Open function either opens or replaces an existing file or creates a new file, it needs a placement and a name for the file as well as whether it should open, replace or create. The TDMS Write writes the input data into the file that has been opened by TDMS Open so they need to be connected. TDMS Close is used to close the file and therefore finalizing the file creation.

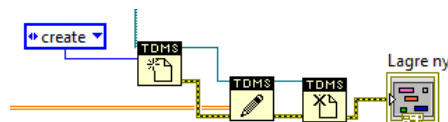


Figure 1.7: Creation of a TDMS file with 2D array data

1.4.3 Conditional saving

Conditional saving is a set of functions that decides whether data should be saved as a new file, appended to the file or not saved at all. This is very useful when processing large amounts of data, specifically when some of it is uninteresting. By doing this, filling hard drives with unnecessary data and time consuming processing without results is avoided.

A way to achieve conditional saving is to use case structures, which are able to do different processes depending on an input value. In this proposed solution what decides whether or not it should save is *trigger mechanisms*. The trigger mechanisms are explained in Subsection 1.4.9. The trigger mechanisms gives a True or a False which decides the case to use. Inside each case there is another case that decides whether or not it should save depending on what has happened before. The point is to save data a certain amount of time after trigger values has been reached. For example a time series of 1 minute with triggered values reached will make it save 30 minutes after. The complete length of the file saved will be the length of one time series multiplied by the amount the second counter is set to.

- True
 - New condition: 1st counter=0 (T/F)
- False
 - New condition: 2st counter above certain value (T/F)
- True - True
 - Creates a new file
 - Exports the filename to a FIFO
 - Sets the first counter to one
- True - False
 - Appends the time series to the file
 - Adds +1 to the first counter
 - Sets the second counter to zero
- False - True
 - Sets both counters to true.
- False - False
 - Appends the time series to the file

When running long time series at Moholt Two, a new file was created every hour so that the resulting files were manageable in size. To do this shift registers can be used to count how many loop iterations have been performed. If the time lapsed per loop is controlled then a certain amount of time per file can be set. If one is added to the numeric which goes trough a Shift Register it will count each loop. This number can then be used in a Quotient and Remainder. It returns zero if a number is divisible by an input number. The Quotient and Remainder can then be connected to an Is Equal which returns true when it is equal to a certain number, in this case zero. When it is true the case structure resolves to create a new file. When it is false however the filename goes straight trough the Case Structure and is kept between each loop in a Shift Register.

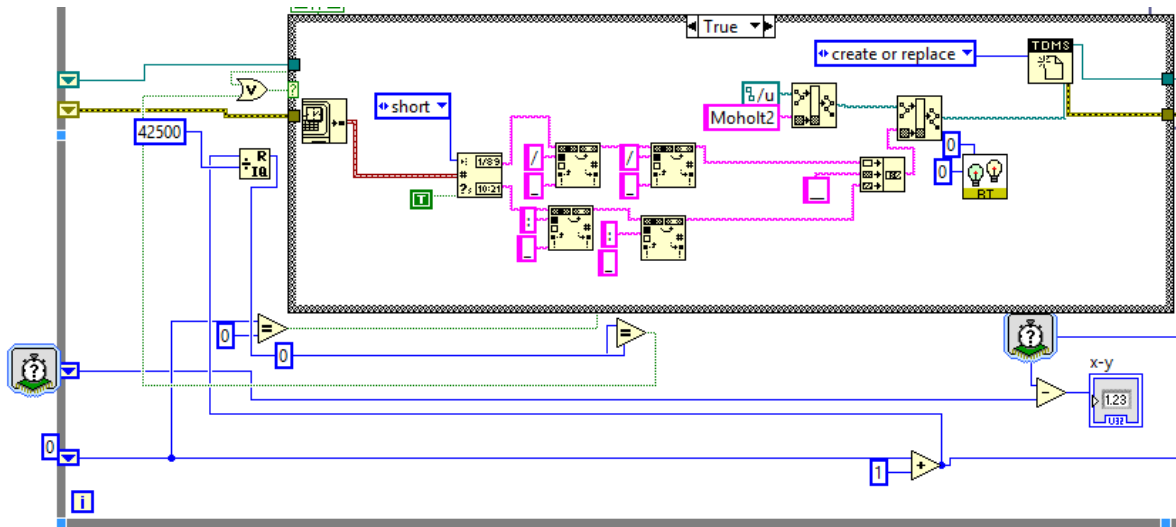


Figure 1.8: Creating new files every two hours

1.4.4 Timestamped filenames

Timestamping filenames is a useful tool for separating different time series from each other. It is not the easiest thing to do in LabVIEW however, as the cRIO does not accept the simplest form of the filenames. As can be seen in the figure it requires multiple functions. The reason for this is that the LabVIEW timestamp format is not compatible with file names on the cRIO and other storage units it is connected to.

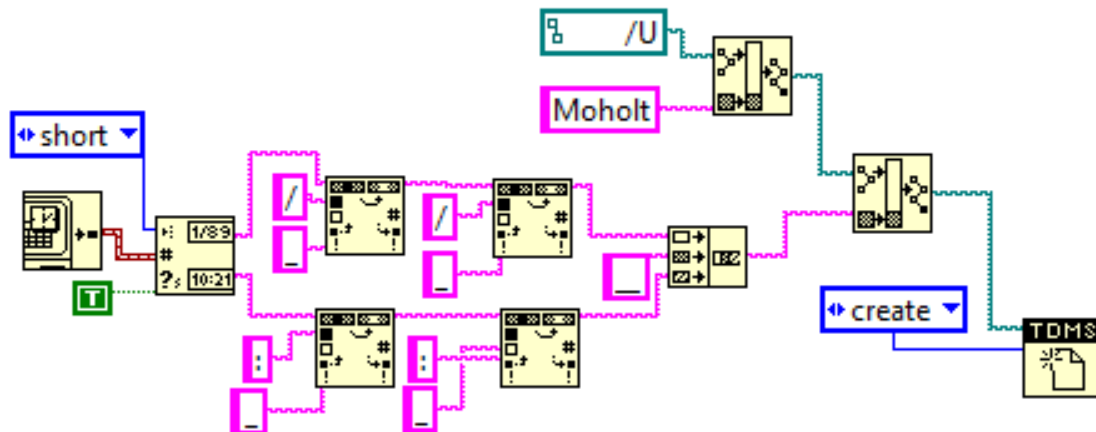


Figure 1.9: Process behind timestamped filenames

1.4.5 Run programs at startup

For a program to run independently it has to start when the cRIO is started. In order to make a VI run at startup a startup executable has to be made. This is done by right-clicking on Build Specification in the project and choosing build, and in the information box choosing the appropriate Vis. Clicking build will then create an executable. It will run at startup if Set as Startup is chosen by Right-Clicking. This function allows the cRIO to take measurements over longer periods of time without constant supervision and it was used at Moholt when time series were recorded over night.

1.4.6 First-in first-out (FIFO) mechanisms

First-In First-Out mechanisms (FIFOs) use rapidly accessible memory to store data temporarily, and require an input and an output. On the input side, the FIFO writes data in a certain order. On the output side, the FIFO reads data in the order it was written. This makes it possible to move data easily within a VI or between VIs. When acquiring data, having control of the sampling rate is essential. When data is written to a file, the time spent is of a more arbitrary nature. Therefore, the FIFO acts as a buffer between the input and output and ensures that data is sampled evenly without data loss.

To create a shared variable FIFO right-click on the cRIO or other device and choose New – Shared Variable, in the information box that appears (or right-click on the FIFO in the project) it is important to set what kind of variable it is (number, array etc.). After that it is just to drag it from the library in the project to the block diagram. To change between Read or Write right-click on the FIFO in the block diagram. If it is wanted to use the FIFO as real-time buffer, Enable RT FIFO has to be checked in the information box.



Figure 1.10: FIFO with Double Precision Float data

To create Real-time FIFOs the functions just needs to be dragged-and-dropped from the toolbox into the VI and connected together. A real-time FIFO has to be created before it can be used and therefore is similar to the tdms file functions.

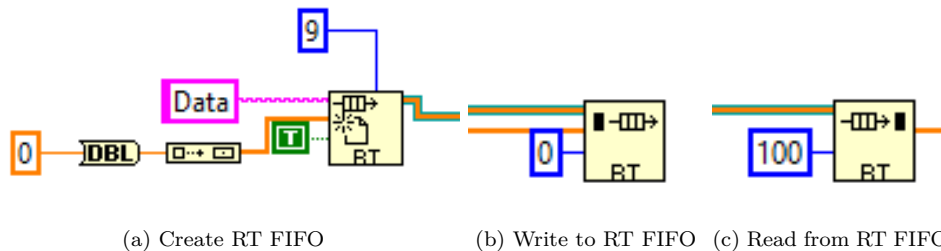


Figure 1.11: RT FIFO structure

1.4.7 Filtering and downsampling

There are several ways to filter the data with respect to the frequency content of the signal. A filter can be applied to an entire channel in a measurement file or applied to a continuous stream of data. In the scan mode VIs, a point-by-point butterworth filter was set to filter out frequency content above 80% of the *Nyquist frequency* in order to avoid *aliasing* [8]. The filter was placed inside the timed loop in scan mode. Otherwise, downsampling could not have been done before the FIFO and that might have slowed down the FIFO.

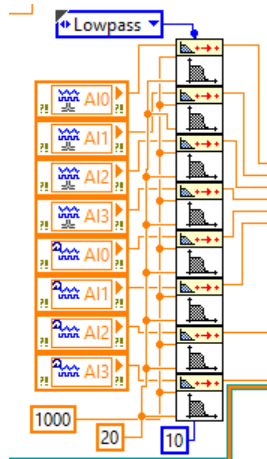


Figure 1.12: Continuous Lowpass Butterworth filter

After the data had been filtered it was downsampled. The reason for doing this was to make the FIFO and data logging more efficient and therefore avoid problems regarding sampling rate. The downsampling was performed by only letting every n 'th point pass through the FIFO, where n is called the *downsampling factor*.

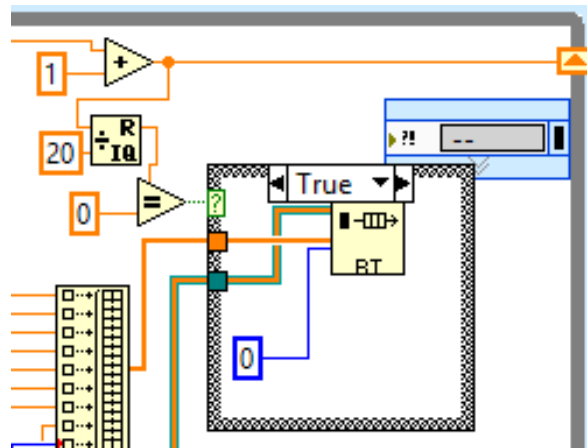


Figure 1.13: Downsampling

To downsample a loop counter and a Case Structure can be used. This method works the same way as when creating two-hour files expect that instead of filenames a FIFO is only in the True part of the Case Structure and therefore data is only logged when it is True.

1.4.8 Aliasing

Aliasing occurs when data is acquired at a sampling frequency lower than twice the highest frequency of the harmonic components in the signal, or when unfiltered data is downsampled. The effect causes folding, which means that frequency content above half the sampling rate will appear in the area underneath half the sampling rate in the APSD of the signal. The same effect in the time domain is shown in Figure 1.14. Therefore, the data has to be filtered before downsampling. This means filtering out the frequencies above of half the desired frequency. This frequency $f_{sampling}/2$ is called the Nyquist frequency. Filters in the time domain are not able to completely remove all the frequency components outside the cut-off frequency. Therefore, the cut-off frequency of a lowpass filter is typically set to 80% of the Nyquist frequency.

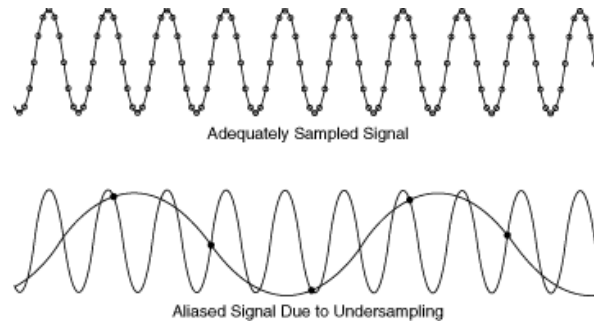


Figure 1.14: Aliasing

Below is a presentation of the effects from aliasing when a 1000Hz signal containing 15,70,80 and 90Hz components is downsampled to 50 Hz. As can be seen the frequency components of 70,80 and 90 Hz are folded down into the area below the Nyquist frequency of 25Hz. This is because when downsampling an unfiltered signal the picked-out data points generate a spurious frequency component equal to $|n * f_{sampling} - f_{signal}|$ [8], where n is the integer that causes $n * f_{sampling}$ to be closest to f_{signal} . The effect was also tested with a butterworth filter with order 10 and a cut-off frequency of 20 Hz and it can be seen that this causes the spurious frequency components to disappear while the component at 15Hz is still present. This shows that the filters used in the data processing has the desired effect.

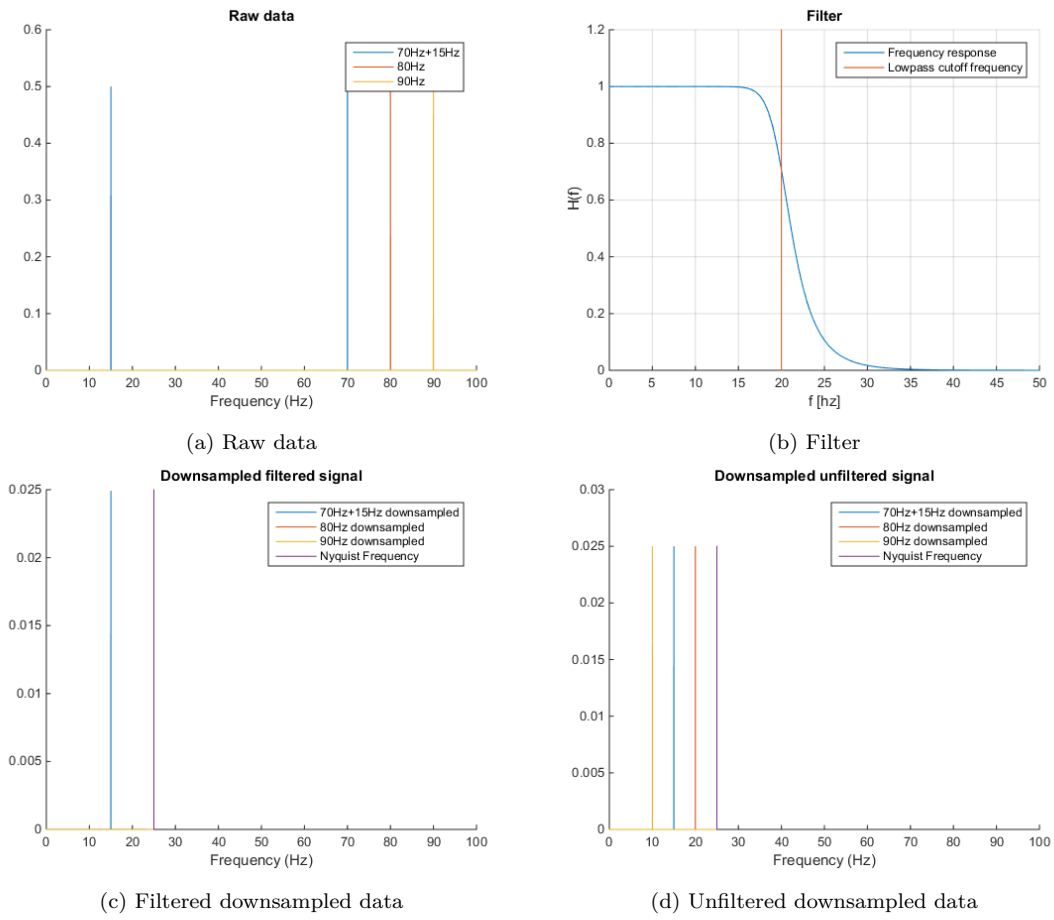


Figure 1.15: Effects of aliasing for filtered and unfiltered signal

1.4.9 Trigger mechanisms

At some point in the process using trigger mechanisms to choose when the data was supposed to be stored was considered. The point of this was to have certain threshold values that indicated that the data was interesting. By doing this it would have been possible for the system to run independently without storing an enormous amount of data. Naturally, these trigger mechanisms could really be anything, such as wind above a certain value or acceleration above a certain value.

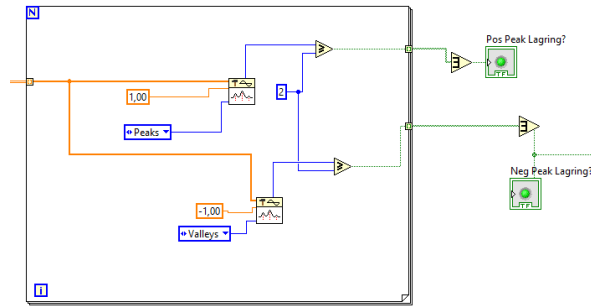


Figure 1.16: A trigger mechanism with Peak Detectors

The method tested here was that the acceleration in a certain timespan needed to be above a certain threshold value a certain number of times. To do this a For Loop was created that was supposed to run through each channel, and if one of them triggered the system would store the data. In the For Loop where two Peak Detector.vi, one set to peaks and one set to valleys. The Peak detector.vi is a built in VI that counts the number of peaks or valleys it detects. A threshold value can be set to ignore peaks under it. Before all of this, the data has to be filtered or the peak detector will detect an unwanted high amount of peaks. After the peak detectors a greater or equal function can be set to decide the amount of peaks that will make it trigger. The greater or equal function returns true or false that can be used to decide if data should be stored. When the true or false values are taken out of the for loop it will be on the form of an array. Then, an Or Array Elements can be used to detect if one of the array elements is true. This will make it return a single true value.

1.4.10 Useful LabVIEW mechanisms

Perhaps the most useful LabVIEW mechanisms is the array functions, especially the Build Array and the Transpose 2D Array functions. These are instrumental when collection and processing the data, as it is very important that the data is of the right type at all times. The for loop is an example of where transposing is important; whether or not the data is correctly transposed decides if it goes through 4 points a 100 times or 100 points 4 times.

LabVIEW also has a large number of functions that can convert data from one form to another. For example To Double Precision Float or To Single Precision Complex. Another useful function is the RT LEDs.vi; it can be used to turn on and off one of the LED lights on the cRIO. This is very useful when running an independent program on the cRIO, as it makes it possible to observe if it is doing what it is supposed to.

Another useful function is a Shift Register which stores values from one loop iteration to the next. It can be created by Right-Clicking on the Wire Tunnel and clicking Replace with Shift Register.

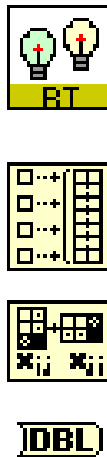


Figure 1.17: RT LED, Build Array, Transform 2D Array and To Double Precision Float

1.5 VI examples

Figures 1.18 and 1.19 shows a scanmode VI and an FPGA VI that has been successfully. They are found in the respective subfolders in the digital appendix.

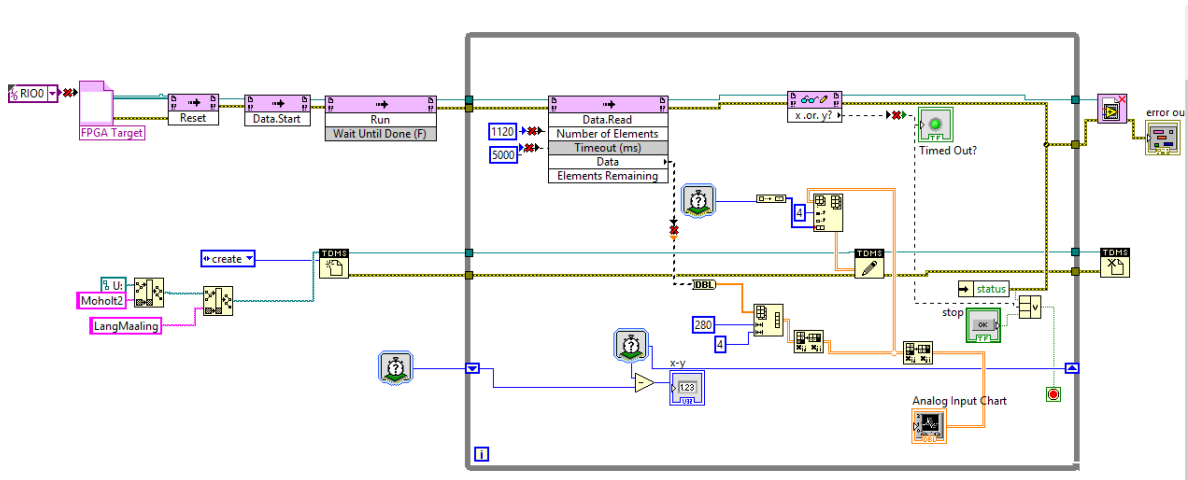


Figure 1.18: FPGA host VI from Moholt Two

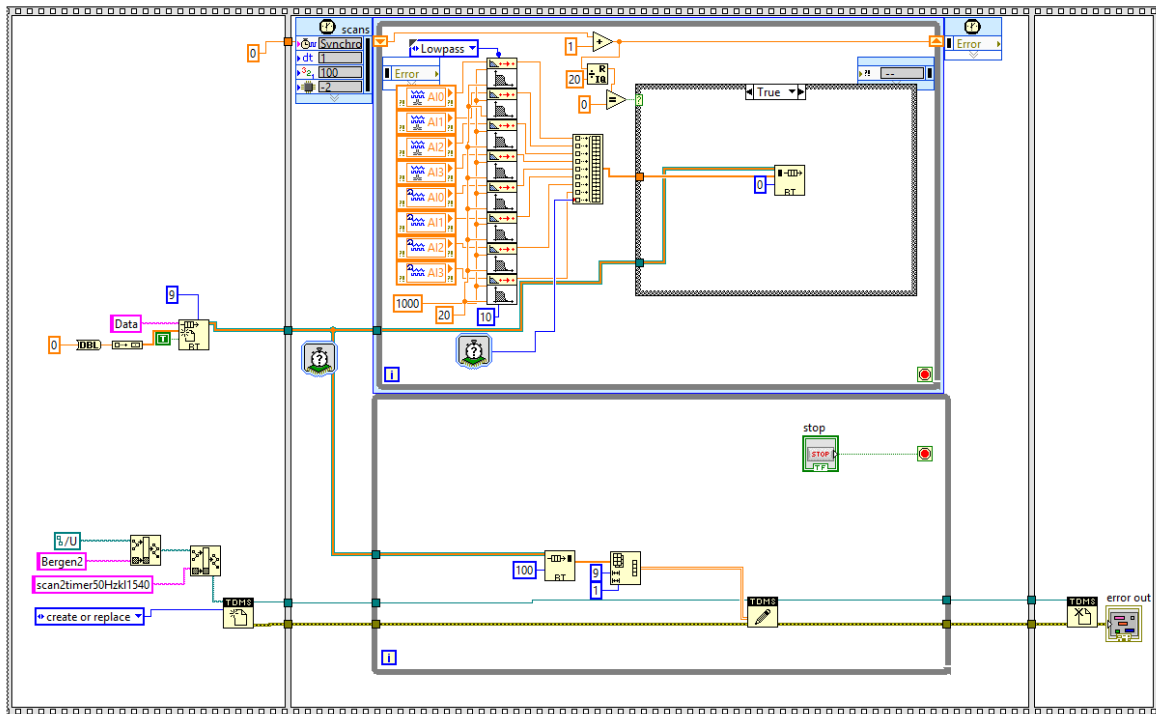


Figure 1.19: Scanmode VI from Bergen Day 2

Bibliography

- [1] National Instruments. Getting started with compactrio - logging data to disk. <http://www.ni.com/tutorial/11198/en/>. Accessed: 2016-05-23.
- [2] National Instruments. Setting up ethercat on ni programmable automation controllers. <http://www.ni.com/white-paper/10555/en/>. Accessed: 2016-05-23.
- [3] Kistler. *K-Beam® Accelerometer Capacitive MEMS, Triaxial Accelerometer Type 8395A...* Accessed: 2016-05-23.
- [4] National Instruments. *NI 9234 Data Sheet*. Accessed: 2016-05-23.
- [5] National Instruments. *SPECIFICATIONS NI cRIO-9067 Embedded Real-Time Controller with Reconfigurable FPGA for C Series Modules*. Accessed: 2016-05-23.
- [6] PCB Piezotronics. *Model 356A16 Platinum Stock Products; Triaxial, high sensitivity, ceramic shear ICP® Installation and Operating Manual*, rev. b edition. Accessed: 2016-05-23.
- [7] PCB Piezotronics. Introduction to piezoelectric accelerometers. http://www.pcb.com/techsupport/tech_accel. Accessed: 2016-05-26.
- [8] Carlo Rainieri and Giovanni Fabbrocino. Operational modal analysis of civil engineering structures. Technical report, Springer, 2014.