University of Politécnica de Madrid

"Escuela Técnica Superior de Ingenieros Industriales"

Automatic, Electronic Engineering and Industrial Computing Department

Master on Industrial Electronic

Implementation of an online signal processing system for a multidimensional ultrasound Doppler flow velocimeter by means of FPGAs.

Author: Enrique Pérez de las Heras

Supervisor: Eduardo de la Torre

March 2013

Master Thesis

Independence declaration

During the 6 months that I have to stay here in Dresden I meet a lot of people and I spent very good moments. However I have had bad moments too and I would like to be grateful to the people who really want me for their support received during all this time. Thank you.

Abstract.

In the field of the magnetohydrodynamics (*MHD)* the measurement of opaque liquid metal flows influenced by magnetic fields are necessary. The purpose is to manipulate flows of liquid metals, semiconductors or electrolytes in fields of metallurgy, crystal pulling and electrochemistry by means of magnetic forces, in order to optimize production processes and to improve product qualities.

The efforts are focused to design a system for flow field measurements of opaque fluids based on the Ultrasound Doppler Velocimetry (*UDV)* that it will be able to provide two dimensional flow field measuring of two velocity profile (*2d-2c*) using a "pulse wave Doppler" to get a multidimensional flow mapping with a high spatial and temporal resolution.

In a first stage, two *4-channel* data acquisition cards for Personal Computers (*PCs)* acquire and store the data in their internal memories. Then, the data is transferred to the hard disk and it is processed off-line. But due to the limitation of the memories, the measurement time is significantly restricted. So an online signal processing must be implemented to reduce the amount of data and to allow a continuous storage with an unlimited measurement time.

A system based in a Field Program Gate Array (*FPGA)* device manufactured by National Instruments (*NI*) and controlled by the *LabVIEW* software is used to solve the problem mentioned above. Part of the signal processing made currently in a *PC* using *MATLAB* software scripts is going to be carried out by the *NI* system. The digital filtering and the down-sampling algorithm are going to be implemented in the *FPGA* card to improve the signal processing of the data and to reduce the amount of data to transfer to the PC.

Furthermore the properly configuration of the Analog to Digital Converter (*ADC)* tarjet connected to the FPGA card is going to be carried out too, because it is necessary to reduce the sampling frequency of the *ADC* to *25 MHz*. Therefore, the *ADC* is going to acquire samples at *25 Ms/s* and they are going to be stored in several First-Input-First-Output (*FIFO)* memories mapped in the *FPGA* device. A *FPGA* device is able to execute parallel tasks at the same time. So the *FIFO* memories are going to be read and filtered by

two digital filters at the same time. The Matched filter is one of the two filters that it is going to be implemented in the FPGA. It is a kind of band-pass filter but much more restrictive and it is used to filter the noise of the incoming signals. And the Hilbert filter is the other filter to implement. It is based on the Hilbert transform and it shifts *90º* the output of the Matched filter to find out the direction of the flow field measured.

Once the samples are filtered, a down-sampling algorithm must be applied before transferring the data to the *PC*. This down-sampling algorithm is going to be programmed in the *FGPA* too, and it is going to reduce the amount of data transferred according to the value configured on a variable programmed on the visual interface (*HOST VI*) programmed using *LabVIEW* in the PC. This program is also going to store the data received from the *FGPA* in a *TDMS* file which it is going to be read by a *MATLAB* script presenting the results of the measurements.

Index.

1. Introduction. 11
2. The ultrasound Doppler measuring systems. 14
2.1 Ultrasound basic principles. 14
2.2 The Ultrasound Doppler Velocimetry (*UDV*). 20
2.3 The Ultrasound Doppler Array Velocimetry (*UDAV*) 24
3. The Signal Processing. 34
3.1 Convolution and Fourier Transform. 34
3.2 Filter Design. 37
3.2.1 The Matched Filter 37
3.2.2 The Hilbert Filter. 41
3.3 Finite Impulse Response Filters. 44
4. The National Instrument System 48
4.1 NI PXIe-7965R 49
4.2 NI 5761 51
4.3 NI PXIe-1073 55
5. FPGA Signal Processing 56
5.1 Filter Design VI 57
5.2 FPGA VI 58
5.3 HOST VI 70
6. Test and results 78
6.1 FPGA VI. 78
6.2 HOST VI. 85
6.2.1 Testing the ADC sampling frequency. 85
6.2.2 Testing the Filters and the Down-Sampling algortihm. 87
7. Summary and Outlook 94
8 Appendix 97
8.1 Appendix A: Resolution theory 97
8.2 Appendix B: The Schwarz inequality 97
8.3 Appendix C: MATLAB script Matched filter. 98
8.4 Appendix D: MATLAB script Hilbert Filter. 98
8.5 Appendix E: SPI Protocol 99
8.6 Appendix F: Phase Jitter, Phase Noise and Time Jitter 100
9. References 101

List of figures.

2.1a: Basic ultrasound principles.
2.1b: Scatterers.
2.1.c: Sound Intensity Profile.
2.1d: Lateral resolution.
2.1e: Depth of penetration or bandwidth.
2.1f: Angle θ_1 formed with respect to the direction of propagation of the ultrasonic wave.
2.2a: Generic burst signal of *8* cycles.
2.2b: This figure shows how it is possible to measure the depth of one particle knowing the time T_d.
2.2c: Aliasing problem.
2.3a: Sensor setup for vector field measurements.
2.3b: Design of the linear ultrasound array.
2.3c: Electronic traversing principle.
2.3d: Simulated sound field of a square ultrasound transducer (*5x5* mm^2) in diagonal and parallel cross-section in normalized sound pressure.
2.3e: Transducer control scheme of *4*-channel parallel operation mode.
2.3f: Excitation pattern of segmental arrays in *4*-channel parallel operation mode.
2.3h: Block diagram of the measuring system based in a *FPGA* with the *DSP* in the *PC*.
2.3i: Block diagram of the measuring system based in a *FPGA* with the Filtering and the Down-sampling in the *FPGA* and the rest of the signal processing in the *PC*.
3: Processing of a signal.
3.1a: Examples of the Fourier series.
3.2.1a: Output signal with noise added.
3.2.1b: Burst signal with F_0=*8Mhz* and $N = 24$.
3.2.1c: Magnitude Response of the Matched Filter.
3.2.2a: Magnitude Response of the Hilbert Filter.
3.3a: Direct-Form Structure.
3.3b: Direct-Form transposed structure for a second order filter.
4: *NI* System.
4.1a: *NI PXIe-7965R* and *VIRTEX V FPGA*.
4.1b: *CLIP* Relationship.
4.2a: *NI 5761*.

4.2b: *NI 5761* Low Speed *CLIP* Signal Block Diagram.
4.3: *NI PXIe-1073*.
5.1:Block diagram which design the filter in *FXP* and get the *.fds file compatible with *LabVIEW*.
5.2a: Translating the *FPGA VI* to a *FPGA* circuit.
5.2b: *ADC* initialization and main program.
5.2c: Table provides by the manufacturer of the *NI5761* to configure the *CLIP* with the desired clock source.
5.2d: Moore machine to acquire samples from the *ADC*.
5.2e: Connections between the loops for one channel.
5.2f: *.fds *VI* to Start *IP* Generator.
5.2g: Magnitude response of the Matched (on the top) and Hilbert (on the bottom) filters Floating Point Vs Fixed Point.
5.2h: Interconnection between the *FIFO* memories in the different loops.
5.2i: Moore *FSM* of the Down-sampling algorithm.
5.2j: Group delay of the Hilbert Filter.
5.2k: Taking into account the group delay.
5.2i: Moore *FSM* of the Down-sampling algorithm in the Hilbert Filter.
5.3a: Configuration window *HOST VI*.
5.3b: *HOST VI* Flowchart.
5.3c: Echo signals received by the *Channel 0* of the *ADC*. *NEEP = 50, Samples/Echo = 1600 and Gate = 50*.
5.3d: Doppler signals extracted from the 300 echoes received.
5.3e: Description of quadrature signals to find the velocity direction.
5.3f: Description of quadrature signals to find the velocity direction with aliasing.
6.1a: Placing the simulated sinus signal in the acquire loop (Top) and Configuration parameters of the simulated sinus signal (Bottom).
6.1b: Placing two data arrays in the transferring loop to test the results of the filters.
6.1c: *FPGA VI* Testing Matched Filter.
6.1d: *FPGA VI* Testing Hilbert Filter.
6.1e: *FPGA VI* Testing Hilbert Filter and the down-sampling algorithm.
6.2.1a: Configuration hardware used to test the sampling frequency.
6.2.1b: Signals to configure in the AFG-3101 to do the sampling test.
6.2.1c: Sampling frequency results.
6.2.2a: Connexions between the function generators and the *ADC*.

6.2.2b: Tektronix configuration table to test the filters and the down-sampling algorithm.
6.2.2c: *HOST VI* Testing Matched Filter.
6.2.2d: *HOST VI* Testing Hilbert filter: Envelope at the output of the filters.
6.2.2e: *HOST VI* Testing Hilbert filter: Zoom at the output of the filters.
6.2.2f: *HOST VI* Testing Hilbert filter and the down-sampling algorithm: Envelope at the output of the filters.
6.2.2g: *HOST VI* Testing Hilbert filter and the down-sampling algorithm: Zoom at the output of the filters.
6.2.2h: *HOST VI* Testing an On-line Signal processing: Envelope at the output of the Matched Filter.
8.3: *MATLAB* script of the Matched Filter.
8.4: *MATLAB* script of the Hilbert Filter.
8.5a: *SPI* signals.
8.5b: *SPI* Transfer Data.

List of symbols.

Symbol	Unit	Explanation
UDV		Ultrasound Doppler Velocimetry.
UDAV		Ultrasound Doppler Array Velocimetry.
MHD		Magnetohydrodynamics.
PIV		Particle Image Velocimetry.
LDA		Laser Doppler Anemometry.
2d-1c		2 dimensional flow field measuring of 1 velocity profile.
2d-2c		2 dimensional flow field measuring of 2 velocity profile.
GaInS		Gallium Indium Stannide.
PRT		Pulse Repetition Period.
PRF		Pulse Repetition Frequency.
NEEP		Number of Emissions per Profile.
FIR		Finite Impulse Response.
NI		National Instruments.
ADC		Analog Digital Converter.
FPGA		Field Programmable Gate Array.
FIFO		First Input First Ouput.
MSPS		Mega Sampled Per Second.
DA		Distributed Arithmetic.
MAC		Multiply-Accumulate.
FSM		Finite State Machine.
CLIP		Component Level IP.
VHDL		Very High Speed Hardware Description Language.
SPI		Serial Peripheral Interface.
λ	[m]	Wavelength (see equation 2.1a).
c	[m/s]	Velocity of sound(see equation 2.1a).
f	[Hz]	Frequency (see equation 2.1a).
d	[m]	Distance (see equation 2.1b).
t	[s]	Time (see equation 2.1b).
f_t	[Hz]	Frequency of the echoes (see equation 2.1c).
f_e	[Hz]	Frequency of the burst signals (see equation 2.1c).

Symbol	Unit	Explanation
v	[m/s]	Velocity of the particle (see equation 2.1c).
θ_1	[º]	Angle betw transducer and particle (see equation 2.1c).
f_r	[Hz]	Echo frequency from the transducer (see equation 2.1d).
f_g	[Hz]	Frequency of the echo (see equation 2.1d).
θ_2	[º]	Angle betw transducer and particle (see equation 2.d).
f_d	[Hz]	Doppler frequency (see equation 2.1e).
D	[m]	Distance betw particle and converter (see equation 2.2a).
T_d	[Hz]	Time delay betw bursts and echoes (see equation 2.2a).
T_{prf}	[Hz]	Period of the pulse repetition freq. (see equation 2.2b).
F_{prf}	[Hz]	Frequency of the pulse repetition freq. (see equation 2.2d).
V_{max}	[m/s]	Maximum velocity to measure (see equation 2.2d).
D_{max}	[m]	Maximum distance to measure (see equation 2.2e).
$y(t)$		Output signal (see equation 3.1a).
$x(t)$		Input (see equation 3.1a).
$F\{x(t)\}$		Fourier Transform(see equation 3.1c).
$F^{-1}\{x(t)\}$		Inverse Fourier Transform (see equation 3.1d).
$h(t)$		Matched filter Impulse Response (see equation 3.2.1g).
k		Gain (see equation 3.2.1g).
$b(td-t)$		Time reversal of the burst signal (see equation 3.2.1g).
F_n	[Hz]	Normalized Frequency (see equation 3.2.2f).
F_s	[Hz]	Sample Frequency (see equation 3.2.2f).
F	[Hz]	Original Frequency (see equation 3.2.2f).
N_T		Number of periods to show (see equation 6.1a).
T	[s]	Period of the Original signal (see equation 6.1a).

1. Introduction.

Magnetohydrodynamics (*MHD*) is the academic discipline which studies the dynamics of electrically conducting fluids. Its idea is that magnetic fields can induce currents in a moving conductive fluid, creating forces on the fluid and moreover changing the magnetic field itself.

Melt processes are present in some industrial processes and *MHD* provides manifold possibilities for electromagnetic flow control. The optimization of them can result very interesting to achieve more efficient productions and higher quality in the items manufactured. So, the objective is the manipulation of liquid metal flows, semiconductors or electrolytes in fields of metallurgy, crystal pulling and electrochemistry by means of magnetic forces.

In the area of *MHD* still exists the exigency of fundamental researches in order to understand the interaction of liquid metal flows with the Lorentz forces. The applied magnetic fields used for the generation of electromagnetic forces do not include only homogeneous continuous fields but particularly the combination of continuous and temporally as well as spatially varying fields.

Besides ongoing numerical simulations, a comprehensive understanding of the interaction between liquid metal flows and different kinds of applied magnetic fields requires detailed experimental investigations, too. Reliable and precise data about the velocity structure are necessary for a validation of the theoretical models implemented in the numerical computer codes. At the moment, it is not possible to build a measurement system which it supports high melting point liquid metals. Then, only liquid metals with low melting point like the eutectic Gallium Indium Stannide alloy (*GaInSn*), are used in the experiments. They are considered as an important tool to investigate the flow structure and related transport processes in liquid metal flows.

A non-invasive, multidimensional, multi-component measurement system for flow fields is desired to do the experiments. However, the instrumentation of respective experiments appears to be very difficult due to the opaqueness, the high temperatures or the chemical aggressiveness of the fluid. Some powerful optical methods like Particle Image Velocimetry (*PIV*) and Laser Doppler

Anemometry (*LDA*) are used for measurements in transparent liquids. But obviously they fail in opaque melts.

Therefore, it is necessary to find a solution for the opaque liquids. This solution is based on the ultrasounds and the Doppler signals obtained from the echo signals. Currently, a few commercial ultrasound Doppler devices, like the Ultrasound Doppler Velocimetry (*UDV*), are available on the market, but their characteristics are limited. For instance the *UDV* can only get measurements of a single velocity profile, providing two-dimensional flow field measurements of one velocity component (*2d-1c*). And it also can only control a low number of ultrasound transducers.

The Ultrasound Doppler Array Velocimetry (*UDAV*) system developed by Laboratory for Measurement and Testing Technologies of the TU Dresden, offers an attractive non-invasive possibility to measure flow velocities in opaque fluids, controlling a high number of ultrasound transducers (*2* arrays of *25* piezo-elements) and getting two-dimensional flow field measurements of two velocity components (*2d-2c*) profiles.
The method used is based on the pulsed echo technique (”Pulsed Wave Doppler") produced by the repetitive emission of short ultrasound bursts. These echo signals have important information for the reconstruction of a velocity profile. It provides a multidimensional flow mapping with a high spatial and temporal resolution, desired for example in the investigations of turbulent flows occurring during the electromagnetic stirring of metal melts.

At the moment the *UDAV* system is able to acquire a *2d-2c* velocity profiles using a personal computer (*PC*) to process the echo signal. The signal processing is based on filtering the echo signals with a couple of filters (Matched and Hilbert filters), down-sampling their outputs and applying a correlation algorithm which gives the velocity profile. The amount of data to process causes that the signal processing is very slow and a lot of time is spent doing an “off-line” signal processing of the data.

One of the reasons because the current system spent a lot of time to process the echoes is the filtering. So the implementation of the filters and the down-sampling algorithm are going to be carried out in the Master thesis. A Field Programmable Gate Array (*FGPA)* device manufactured by National Instruments (*NI*) is going to be used to implement part of this signal

processing. Furthermore, a down-sampling algorithm is going to be implemented in the FPGA to reduce the amount of data getting a faster signal processing and achieving an on-line acquisition of the velocity profiles.

2. The ultrasound Doppler measuring systems.

In this section, firstly the basic principles of the ultrasound are going to be introduced. Then, the theory of the UDV is going to be explained later. And finally the UDAV is going to be treated in detail.

2.1 Ultrasound basic principles.

Ultrasound is acoustic energy spread in wave forms with a frequency range of *1 - 12 MHz* well above to the human hearing range (*15KHz – 20KHz*). With higher frequencies the sound move like electromagnetic beams and it is reflected in a form of light beams.

The wavelength λ is function of the frequency f and the sound velocity c. The sound velocity is a constant and it changes according to the properties of the material used.

$$\lambda = \frac{c}{f} \qquad [2.1a]$$

If the velocity of the sound (c) and the time (t) spent by the ultrasound are known, the distance (D) between the ultrasound transducer and the object is given by the equation 2.1b. Note that this equation gives the total distance measured when the ultrasound pulse goes to the object, it is hit and then it goes back to the transducer. So this distance has to be divided by *2* to find out "the one-way" distance.

$$D = c \cdot t \qquad [2.1b]$$

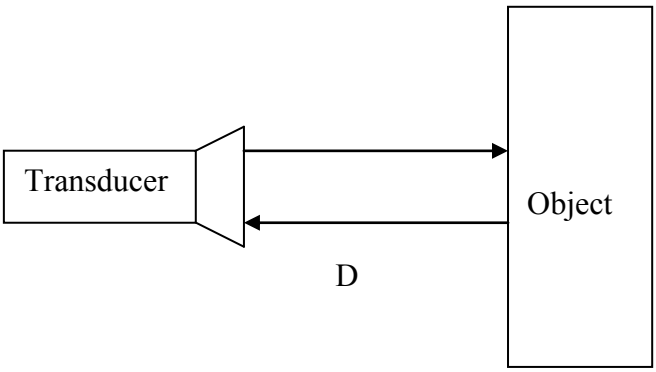


Figure 2.1a: Basic ultrasound principles. [1]

Echoes and Scattered.

An echo is produced when a sound wave travels through the air or liquid hits against a solid surface.

The type of echo processed by the *UDV* is known as scattered and the reflecting structures (particles) are usually named scatterers (figure 2.1b).

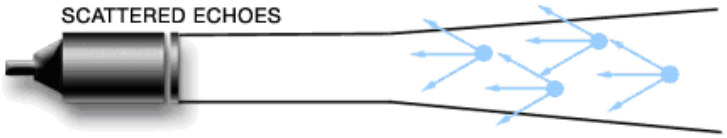


Figure 2.1b: Scatterers. [2]

Sound intensity profile.

The ultrasound beam is generated by a kind of piezoelectric crystal which its form changes when electric current is applied. This piezoelectric crystal can be the receiver of the ultrasound too. Figure 2.1c shows the sound intensity profile of an ultrasound beam. The beam generated changes with the distance. So it is possible to find three different zones.

- Near: it is called the Fresnel zone. It is the zone where the diameter of the beam decreases when the distance from the transducer increases.
- Focal: it is the zone where the concentration of the beam diameter is the highest, giving the greatest degree of focus.
- Far: it is called the Fraunhoffer zone. It is the zone where the beam diameter increases when the distance from the transducer increases, too.

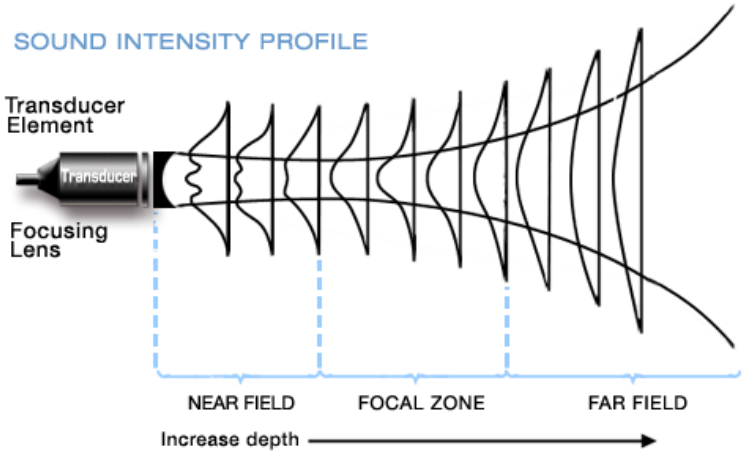


Figure 2.1.c: Sound Intensity Profile. [2]

The resolution.

A very important parameter which defines the quality of an ultrasound device is the resolution. There are three types:

- Lateral resolution: it resolves objects which are placed side by side. Parameters like the focal depth (F), the wavelength (λ) and the probe diameter or aperture of the ultrasound probe (D) are depending on it (Figure 2.1d). The frequency of the ultrasound beam changes the lateral resolution. If the frequency is higher, the lateral resolution is better.

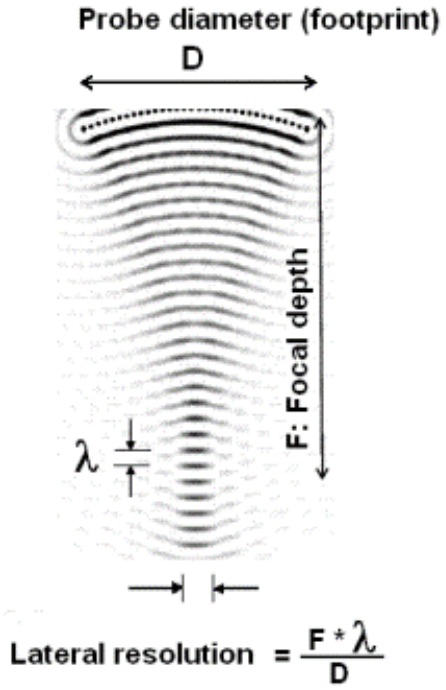


Figure 2.1d: Lateral resolution. [3]

- Axial resolution: it resolves objects that they are placed above and below each other. It depends on the frequency of the ultrasound beam. If the frequency is higher, the axial resolution is lower.
- Depth of penetration or bandwidth: it is the resolution along the beam and it depends on the length of it. The length of the beam is not proportional to the frequency spread (figure 2.1e). There is more attenuation in the higher frequencies than in the lower frequencies. So, when the frequency is higher, the resolution is better, but nevertheless the bandwidth is lower.

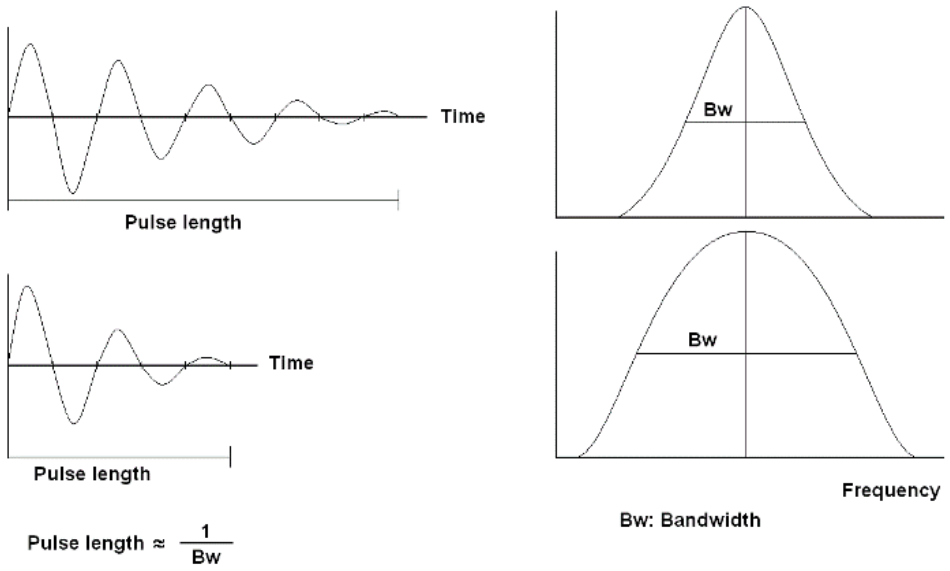


Figure 2.1e: Depth of penetration or bandwidth. [3]

The Doppler effect.

The Doppler effect is produced when the frequency of the ultrasound beam is modified when it is reflected by a moving surface.

To proof in a mathematical way this phenomenon, it is necessary to imagine that one ultrasound transducer is generating beams with f_e of frequency, and they are propagated in a medium where the speed of the sound c is known. Furthermore a particle (or scatterer) is moving with a velocity v which it is considered negative when the movement of the particle is toward the ultrasound transducer. If the angle θ_1 formed between the ultrasound and the direction of propagation of the beam is considered, the frequency of the echoes received f_t is determined by the equation 2.1c.

$$f_t = f_e - \frac{f_e \cdot v \cdot \cos\theta_1}{c} \quad [2.1c]$$

Figure 2.1f shows a picture which describes the equation stated before.

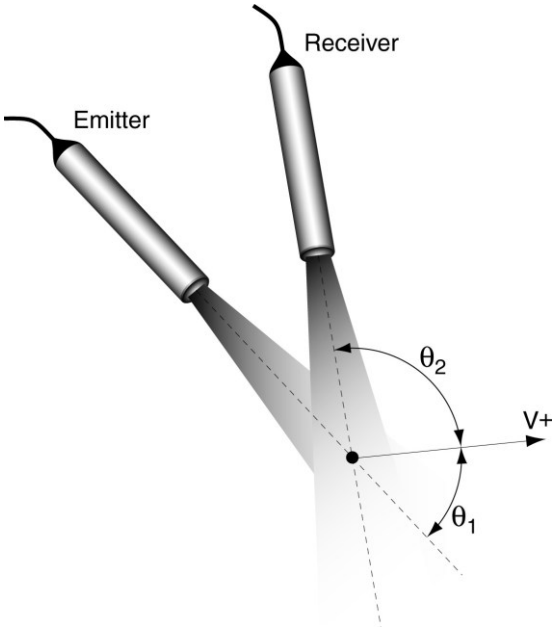


Figure 2.1f: angle θ_1 formed with respect to the direction of propagation of the ultrasonic wave. [4]

The ultrasound beam will not be totally reflected when the acoustic impedance of the particle is different respecting to the surrounding medium. The frequency of the echo measured by the transducer acting like a receiver is expressed by the equation 2.1d.

$$f_r = f_g + \frac{f_g \cdot v \cdot \cos\theta_2}{c} \qquad [2.1d]$$

The difference between the emitted and received beams is known as the Doppler frequency shift. And it is expressed by the equation 2.1e. In this equation it is necessary to neglect the second order terms because the velocity of the particle is much smaller than the speed of sound ($v << c$). But moreover if the ultrasound transducer is used to receive the echoes, the equation 2.1e is reduced to the equation 2.1f.

$$f_d = \frac{f_e \cdot v \cdot (\cos\theta_1 - \cos\theta_2)}{c} \qquad [2.1e]$$

$$f_d = \frac{2 \cdot f_e \cdot v \cdot \cos\theta_1}{c} \qquad [2.1f]$$

2.2 The Ultrasound Doppler Velocimetry (*UDV*).

A burst signal is a sinus signal composed by a determinate number of cycles (figure 2.2a).

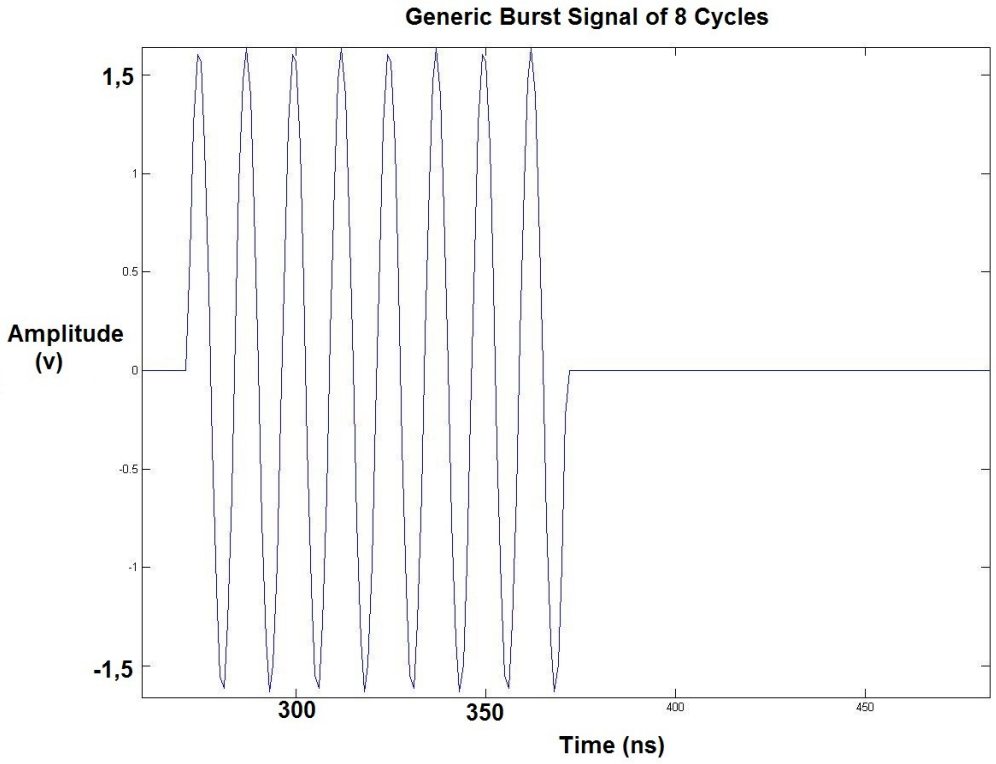


Figure 2.2a: Generic burst signal of 8 cycles.

A pulse Doppler ultrasound emits burst signals with a short number of cycles periodically. And it receives continuously the echoes produced by the

particles. The echoes are sampled at the same time. And this time is relative to the emission of the bursts. Therefore, the shift positions of particles can be measured.

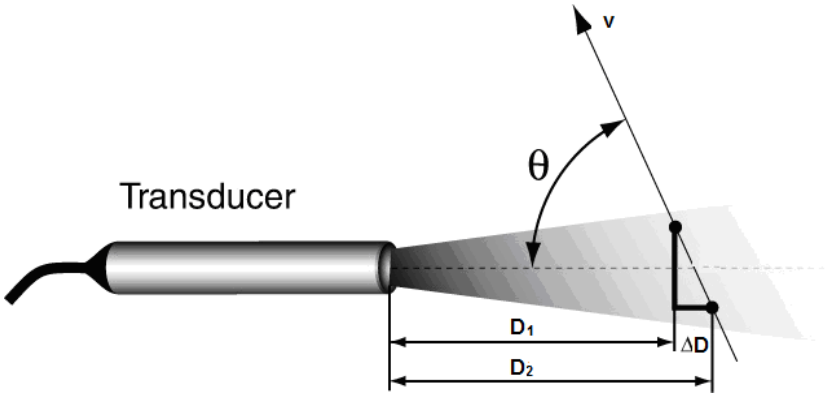


Figure 2.2b: This figure shows how it is possible to measure the depth of one particle knowing the time T_d. [4]

Figure 2.2b shows one particle inside the ultrasonic beam. The time delay T_d between the burst and the echo is known because previously, the frequency of the echo was calculated, and the frequency of the burst is known. Then the depth D (which is the distance from the particle to the converter) is computed by the equation 2.2a:

$$D = \frac{c \cdot T_d}{2} \qquad [2.2a]$$

The sound velocity in the liquid is c. The depth is divided by 2 because it is only interesting to find out the distance in only one way.

When the particle is moving with an angle θ in relation to the axis of the beam, its velocity can be measured inquiring the depth difference between two consecutive burst signals generated in a Period of the pulse repetition frequency (T_{prf}). The difference between the depths is given by the equation 2.2b.

$$(D_2 - D_1) = v \cdot T_{prf} \cdot \cos\theta = \frac{c}{2} \cdot (T_2 - T_1) \qquad [2.2b]$$

Then the velocity of the particle is expressed by the equation 2.2c, resulting as the same as the Doppler equation. But it is important to keep in mind what happens when several particles are involved. In this situation several echoes are produced in a random way. So it is necessary to apply a correlation algorithm to the echoes, to measure the velocity profile.

$$v = \frac{c \cdot f_d}{2 \cdot f_e \cdot \cos\theta} \qquad [2.2c]$$

Limitations and maximum values

A Doppler ultrasound gives a relation between the spatial information and the velocity values. But this information is limited by the Nyquist theorem. Therefore the maximum velocity (without the aliasing effect) is measured according to the frequency of the pulse repetition frequency[1] ($F_{prf}=1/T_{prf}$) of the burst signals. And it is expressed by the equation 2.2d.

$$v_{\max} = \frac{c \cdot F_{prf}/2}{2 \cdot f_e \cdot \cos\theta \cdot} = \frac{c}{4 \cdot f_e \cdot \cos\theta \cdot T_{prf}} \qquad [2.2d]$$

The aliasing effect is produced when the measured velocity is higher than v_{max}. Then all the frequencies above the half of the sampling frequency ($>F_{prf}/2$) are duplicated or aliased in the low frequency region. Figure 2.2c shows the relationship between the real frequency and the measured frequency.

Having a look at the figure 2.2c, it is possible to see a brief description about the aliasing problem. At point 1 the frequency of the signal is lower than $F_{prf}/2$ (Nyquist limit), so the measured frequency and real frequency are the same. At point 2, the real Doppler frequency of the signal is higher than the

[1] F_{prf} and *PRF* are going to be used to express the same.

Nyquist limit. Therefore, the point 2 is duplicated in the point 3, giving an erroneous velocity value (negative frequency).
When the aliasing is produced in an analog signal, it is easy to solve it, if the signal is filtered with a low pass filter which removes all the frequencies above the Nyquist limit. However in a pulsed Doppler ultrasound the signal has already been sampled and it is not possible to apply an analog filter to remove the aliasing. The solution would be to do an adaptation between the sampling rate and the F_{prf} to measure properly the Doppler signal.

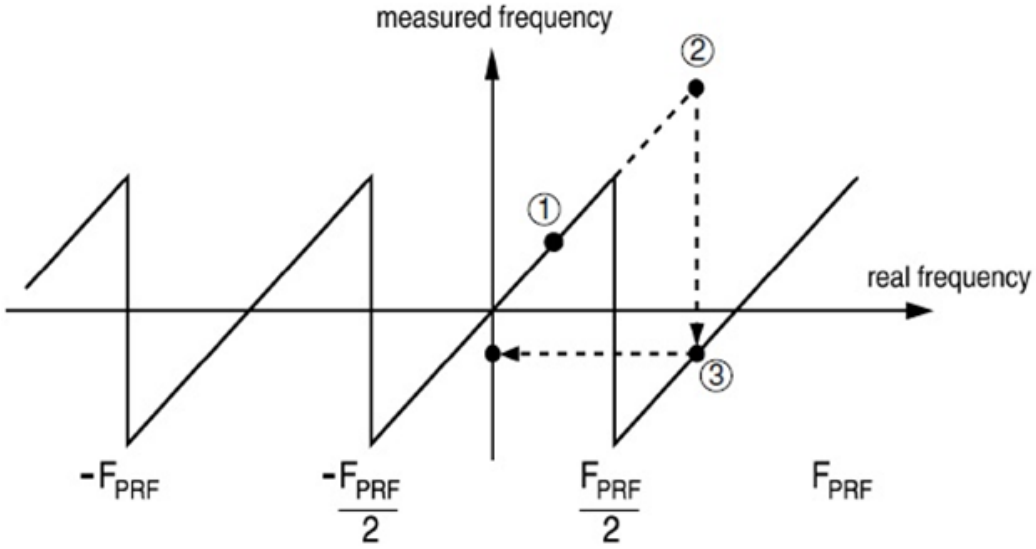


Figure 2.2c: Aliasing problem. [4]

Moreover the maximum depth which is possible to measure given a velocity, is determined by the time needed for the ultrasound beam to go toward the particle, hit it and go back to the transducer. The F_{prf} gives the maximum depth computed by the equation 2.2e.

$$D_{\max} = \frac{T_{prf} \cdot c}{2} \quad [2.2e]$$

If the F_{prf} is reduced, the maximum measurable depth is increased. However the maximum Doppler frequency measured is also reduced. So the maximum depth and velocity is given by the equation 2.2f.

$$D_{\max} \cdot V_{\max} = \frac{c^2}{8 \cdot f_e} \quad [2.2f]$$

2.3 The Ultrasound Doppler Array Velocimetry (*UDAV*)

Two linear ultrasound transducer arrays equipped with *25* transducer elements each one are controlled by a special customized hardware to determinate two-dimensional flow fields of two velocity components profiles (*2d-2c*).

Ultrasound sensor.

The ultrasound sensor system is composed by two identical linear ultrasound transducer arrays placed in an orthogonal arrangement spanning a square measuring plane of *70x70 mm*^{2}. (Figure 2.3a)

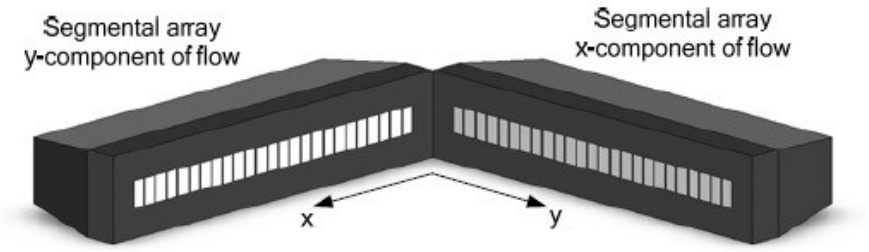


Figure 2.3a: Sensor setup for vector field measurements. [6]

Each ultrasound array has *25* elements (size *2.5 mm* × *5 mm*) with pitch between elements of *2.8 mm*. So the length of the ultrasound array is *70 mm* with gaps of *0.3 mm* (approximately) between adjacent piezo elements (Figure 2.3b).

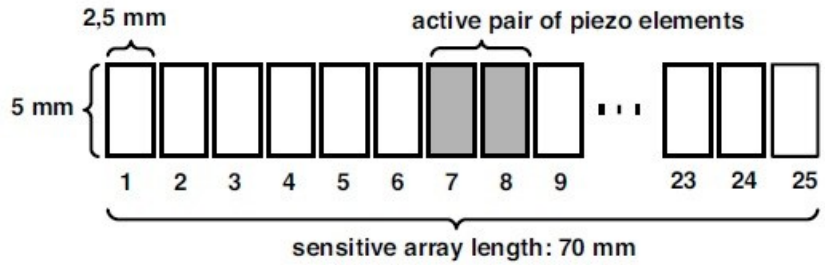


Figure 2.3b: Design of the linear ultrasound array. [5]

The dimensions of the piezo transducer elements were chosen according to the required demands. A high number of array elements are necessary to get a dense scanning of flow fields. And all the array elements have to be placed in a small size over the given length (*70 mm*).

The ultrasound arrays are designed to operate as segmental arrays. Segmental arrays are actuated in groups of elements to achieve a low beam divergence (resulting in a high spatial resolution) on the one hand and a small beam line pitch (resulting in a high measuring line density) on the other hand. Two adjacent rectangular piezo elements are combined in pairs resulting in an active square transducer (size *5 x 5 mm*^{2}) which can be shifted by half of its edge length to maintain the operation of the low beam divergence (Figure 2.3c). As a consequence, this segmental operation using *25* piezo elements provides *24* measuring lines per array.

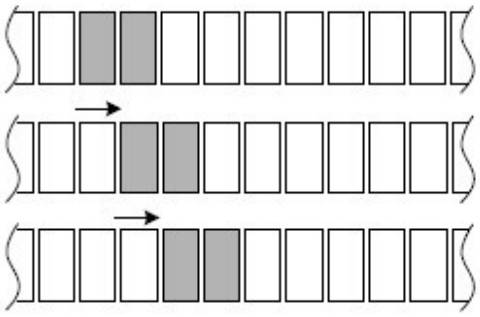


Figure 2.3c: Electronic traversing principle. [5]

Figure 2.3d shows the results of the sound field simulation of the diagonal and parallel cross-section for a *5x5 mm*^{2} square transducer. The transducer surface is segmented into small elements of area, transmitting a spherical wave each one. The position of the focus (identical to the near field) depth and the *-6 dB*-focus width (identical to the lateral resolution) is obtained.

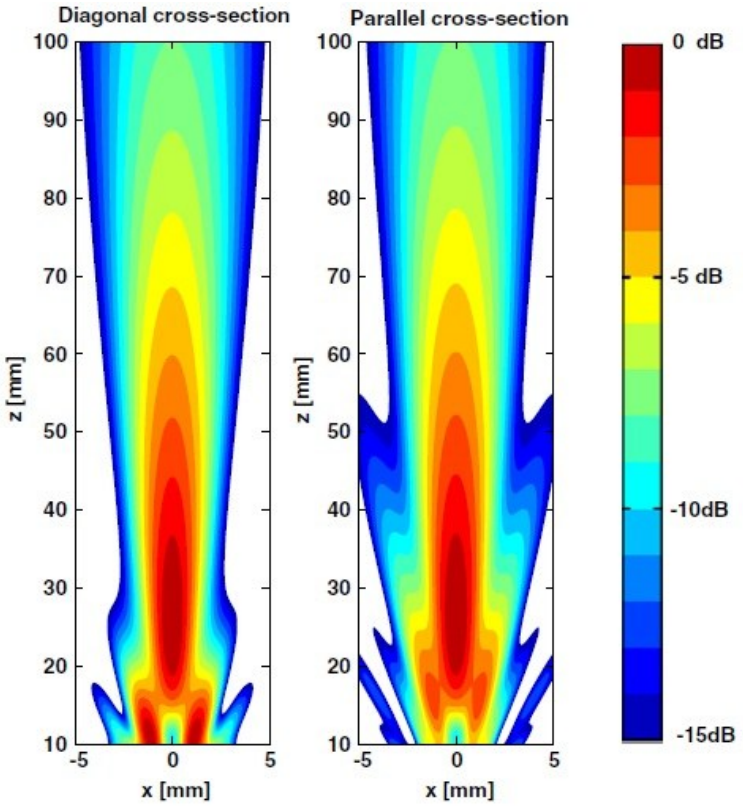


Figure 2.3d: Simulated sound field of a square ultrasound transducer (*5x5 mm*^{2}) in diagonal and parallel cross-section in normalized sound pressure. [5]

The simulations are performed for Gallium Indium Stannide (*GaInSn)*, with a sound velocity $c = 2730\ m\ s^{-1}$. The gap of *0.3 mm* between the piezo elements is insignificant small and was despised to get a compromise between a high penetration depth of the ultrasound beam in the fluid and a high spatial resolution. The emission frequency of the burst signals chosen is: $f_e = 8\ MHz$

corresponding to a wavelength of $\lambda=0:342\ mm$. The near field depth is computed with $z = 25.4\ mm$. There, the lateral resolution for both cross sections can be evaluated with $\Delta x = 3\ mm$ ($2,7\ mm + 0,3\ mm$). It complies with the small traversing step width of about $2.8\ mm$.
With a $f_e = 8\ MHz$ and 8 harmonic wave cycles of ultrasound pulses (*Number of cycles*), the axial resolution[2] obtained is $\Delta z = 1.4\ mm$ (approximately).

Operation principle.

Generally, ultrasound arrays work in sequential time-division multiplexing. So, the array elements are operated one after the other giving a high measurement rate. If the number of piezo elements is very high, it increases the time in the duration of the measurement and it worsens the temporal resolution. So, a sequential time-division multiplexing is not enough for operating the transducer elements of the arrays.

There are two possible solutions to solve this problem. The first one is to achieve a high temporal resolution despite the high number of array elements parallelizing the measurement as much as possible (Figure 2.3e). Thereby, the spacing between the active piezo pairs of transducer elements is chosen in such an extent that crosstalk[3] can be neglected. The optimum spacing for the present measurement system was determined by sound field simulations and crosstalk investigations. For the current array design experimental investigations revealed that a configuration with a distance of 4 inactive array elements between each active element pair induces a tolerable crosstalk of less than 1%. This spacing permits, in the current array design, the parallel operation of four transducer pairs, scanning the entire flow field in $n = 6$ switching steps.

[2] Appendix A: Resolution theory.
[3] Crosstalk: when one transducer element receives the effect of the echo signal from other transducer element.

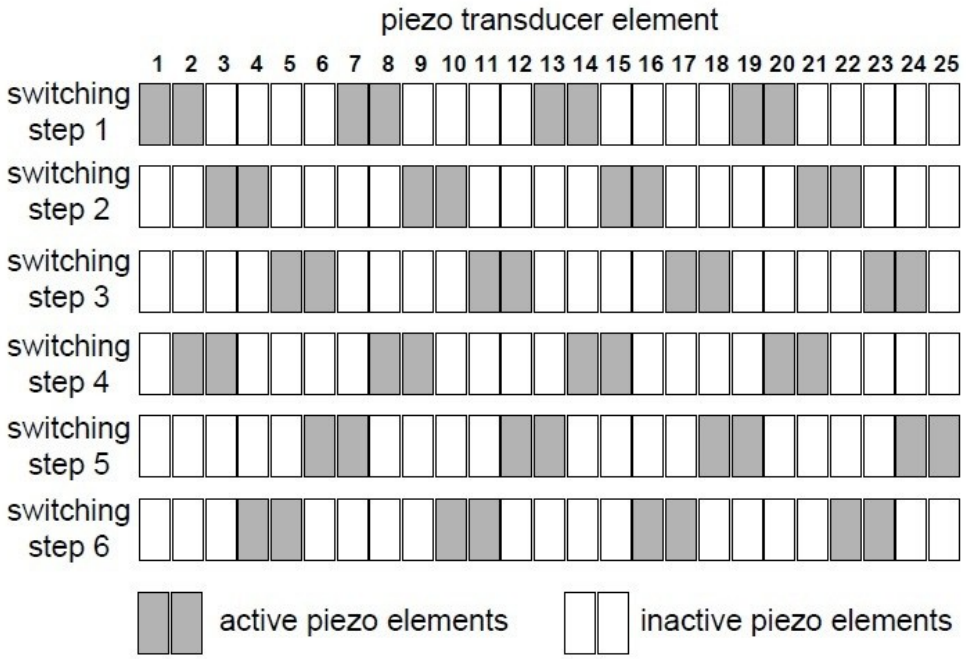


Figure 2.3e: Transducer control scheme of 4-channel parallel operation mode. [6]

Many consecutive echo signals are necessary for the reconstruction of the velocity profiles. An Optimum selection of the Pulse Repetition Time[4] (*PRT*) has to be selected to measure the amplitude of the velocity profile. The *PRT* is the inverse of the Pulse Repetition Frequency (*PRF*), and it has to be chosen small to avoid the aliasing when high flow velocities are given. However small *PRT* values for slow flows, decreases the velocity resolution of the measurement. Then, it is necessary to check that with moderate flow velocities, the recording time of the echo signal[5] is much shorter than the *PRT*. So the gap time between the end of the recording time and the next ultrasound pulse transmitted is not used, and a second solution is proposed. It uses this gap time to excite and record further measuring lines (Figure 2.3f). Therefore it will be possible to sample all remaining measuring lines within

[4] PRT: Time between consecutive ultrasound pulses.
[5] ,The recording time of the echo signal: is the measurement depth and it is related with the depth of the vessel.

this gap time, and the resulting temporal resolution is mainly determined by the velocity measurement range and the size of the fluid vessel.

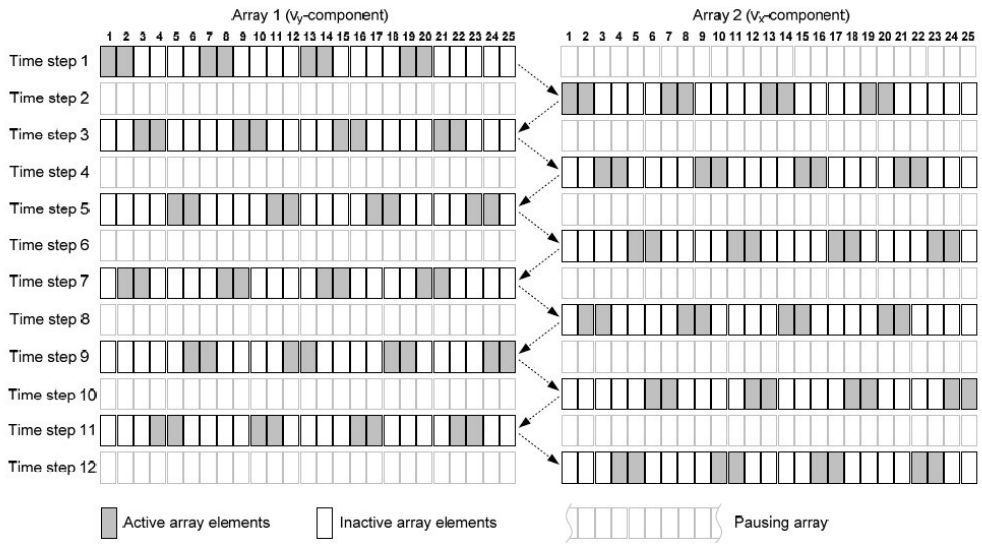


Figure 2.3f: Excitation pattern of segmental arrays in 4-channel parallel operation mode. [6]

The figure 2.3f shows the pulse excitation pattern of the UDAV. The left side represents the excitation pattern of a single array. In that case 6 time steps (switching steps) are needed to sample all 24 measuring lines once. The pattern is repeated a Number of Emissions per Profile (*NEPP*) times according to the number of pulse emissions required to get the velocity profile. During several *PRTs* the entire flow field is obtained. It increases the temporal resolution distinctly and results a quasi-simultaneous determination of all velocity profiles.
The excitation pattern is not enabled linearly to avoid the multiple operation of the same element in successive time steps.
The entire pattern in the figure 2.3f is applied to measuring *2C* vector fields. Both linear arrays have to be driven mutually. Then one is working while the other is reconfigured.

Working principle.

Figure 2.3g shows a block diagram of the current measuring system. Its aim is the multiplex circuit which implements the excitation pattern. It has two switching matrices, one for each array, and a control unit which generates the trigger signals, configures the switching matrices and communicates with the personal computer.

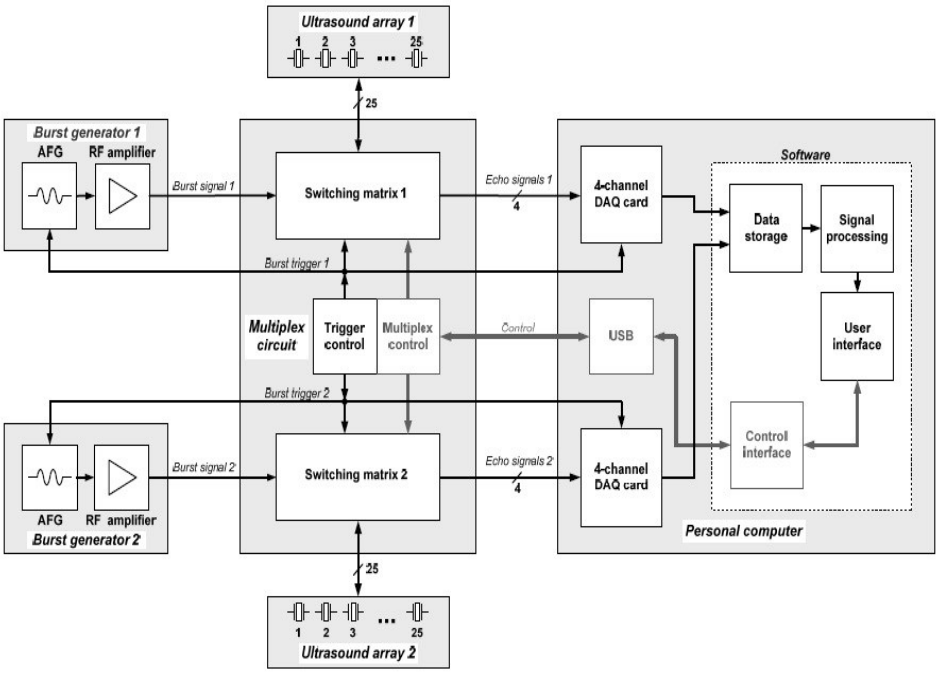


Figure 2.3g: Ultrasound measurement system for two-dimensional two-velocity- component flow mapping. [6]

The trigger control initiates the arbitrary function generators (*AFG 1* and *AFG 2*) to generate the burst signals. These signals are amplified by high voltage RF amplifiers and transmitted to the corresponding switching matrices. Each switching matrix compromises two systems of electonic multiplexers (the transmitting and the receiving multiplexer) and a set of fast high-voltage switches which it distributes the high-voltage burst signal to the transducer elements.

The Transmitting/Receiving switches separate the received pulse echoes from the exciting burst signals. Then, the echo signals received are amplified and driven to the 4-channel matrix output and filtered by a band-pass filter. The switching matrices are controlled by the multiplex control circuit and synchronized with the burst triggers.
The multiplexed echo signals of both switching matrices are digitized by two 4-channel data acquisition (*DAQ*) cards installed in a personal computer (*PC*) and the sampling rates are set at *25MS/s*.

Every time a burst signal is triggered, the segment of the echo signal which corresponds to the measurement depth is recorded. After capturing and recording of multiple segments of echo signals, the *2d-2c* velocity vectors of the flow field are calculated.

Problems and limitations.

Due to the limited data memory of the *DAQ* cards the measurement duration is restricted to separate periods lasting for some seconds to a few minutes (depending on the pulse repetition frequency applied). The interruption of the measurement is caused by the necessity to read out the memory of the *DAQ* cards before the next data acquisition cycle can be started.

Therefore, it is necessary to perform an online signal processing and reduce the data volume. It would allow a continuous storage and an unlimited measurement time. A system based on a Field Programmable Gate Array (*FPGA)* and manufactured by National Instruments (*NI*) is going to be used to solve this problem.

The solution.

The NI system is going to replace the two *DAQ* cards and it is basically composed by an Analog to Digital Converter (*ADC)* composed by *4* channels and one *FPGA* device. The *ADC* device will sample the echo signals at *25 MHz* using the *4* channels at the same time. Then, the samples will be stored in two First-Input-First-Output (*FIFO*) memories and transferred to a Personal Computer (*PC)* (Figure 2.3h). Only the echo signals received by one of the two ultrasound arrays can be sampled, because the *ADC* target has only 4 channels. So the measurement is restricted to *2d-1c* velocity profiles.

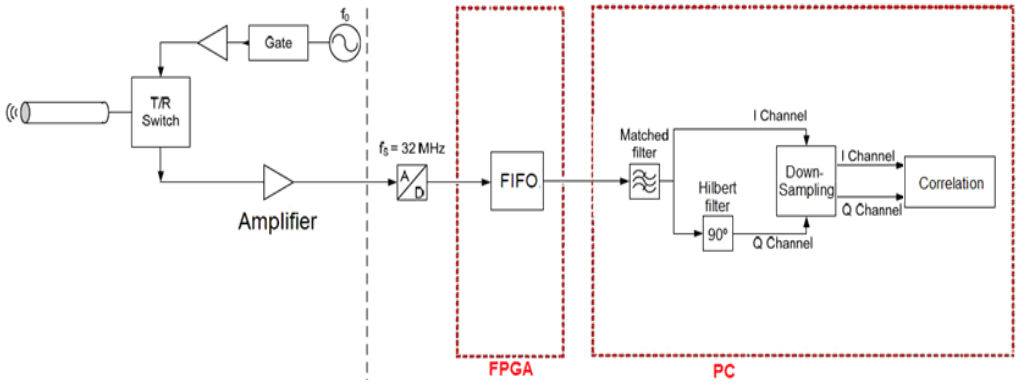


Figure 2.3h: Block diagram of the measuring system based in a *FPGA* with the *DSP* in the *PC*.

The signal processing of the samples is based on a filtering and down-sampling. A Matched and Hilbert Filters are going to filter the incoming echoes and a down-sampling algorithm will reduce the amount of data to transfer to the PC. However a problem has been found. It is not possible to use high *PRFs* to measure high velocity profiles because the time which delays the signal processing in the *PC* is very long. And the main cause of this delay is the filtering of the digital echo signals.

Two digital filters and a down-sampling algorithm must be implemented in the FPGA to reduce the time of the signal processing and to measure high velocity profiles (Figure 2.3i). The Matched filter is used to filter the noise from the amplifier in the Multiplex circuit. And the Hilbert filter is used to shift 90º the output of the Matched filter to find out the direction of the velocity profile.

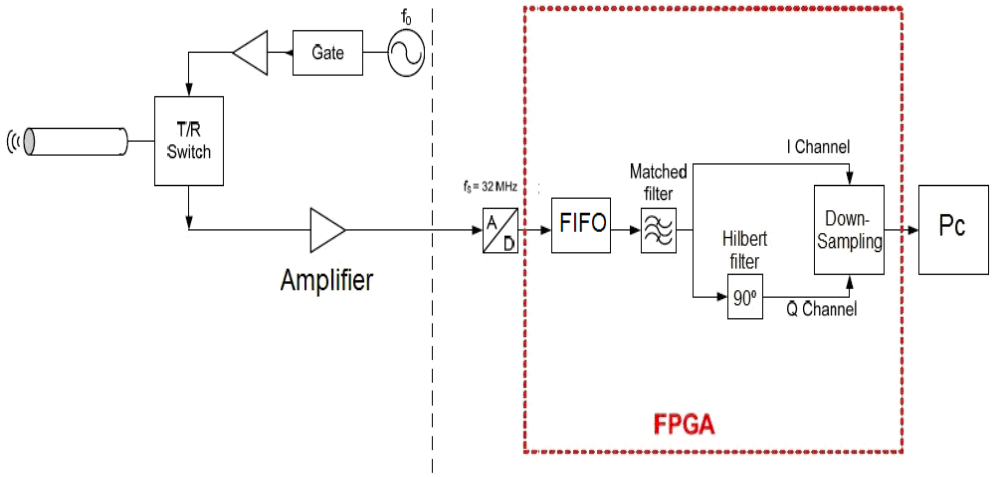


Figure 2.3i: Block diagram of the measuring system based in a FPGA with the Filtering and the Down-sampling in the FPGA and the rest of the signal processing in the PC

3. The Signal Processing.

The signal processing is the area which studies and analyzes the behaviour of the signals in a mathematic way. Figure 3 shows how an analog signal can be processed. In a first step an analog-to-digital converter (*ADC*) is necessary to convert the analog signal in a digital signal. Then the signal processing is applied over the digital signal. And finally in some cases it is necessary to get the analog signal again. So a digital-to-analog converter is placed in the last stage.

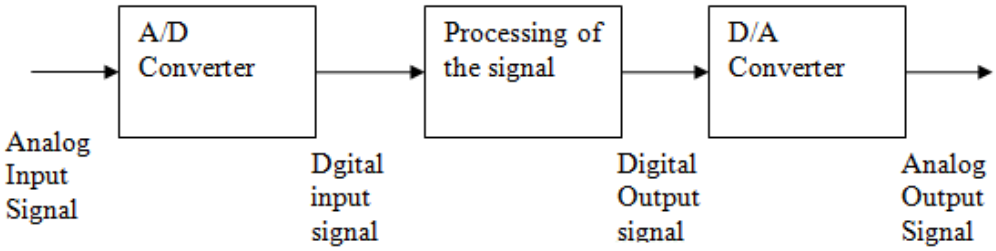


Figure 3: Processing of a signal

Focusing in the processing of the signal, one application is the filtering. This chapter is divided in two parts. The first one explains basic mathematical concepts about the signal processing like the convolution, the Fourier transform and other topics. And the second one is about the filtering. Two digital filters have been designed to process the signal acquired by the *ADC*.

3.1 Convolution and Fourier Transform.

Convolution and Fourier Transform are one of the most important operations in the signal processing. It is very important that these concepts are very clear. So they are going to be explained in detail.

The convolution

The convolution in the time domain is denoted by the * symbol. The equation 3.1a shows the convolution between a signal *x(t)* with a system *h(t)* viewed from the input side.

$$y(t) = x(t) * h(t) \qquad [3.1a]$$

And the equation 3.1b shows the meaning of *x(t) * h(t)* given the *y(t)* viewed from the output side. It is basically an integral which takes the limit from minus infinity to infinity and multiply the instantaneous value of *x* in a time τ by the system shifted a time τ, too.

$$y(t) = \int_{-\infty}^{+\infty} x(\tau) \cdot h(t-\tau) \cdot d\tau \qquad [3.1b]$$

The Fourier Transform.

The Fourier transform is used to link the signal expressed in the time domain with the frequency domain: *F{x(t)}*. Equation 3.1c shows this process. And the equation 3.1d shows the inverse Fourier transform that it is used to get the signal in the time domain when it is given in the frequency domain: $F^{-1}\{x(t)\}$.

$$F\{x(t)\} = \int_{-\infty}^{+\infty} x(t) \cdot e^{-jwt} \cdot dt \qquad [3.1c]$$

$$F^{-1}\{x(w)\} = \int_{-\infty}^{+\infty} x(w) \cdot e^{jwt} \cdot dw \qquad [3.1d]$$

Where:

$$e^{jwt} = \cos(wt) + j \cdot \sin(wt)$$
$$e^{-jwt} = \cos(wt) - j \cdot \sin(wt)$$
$$w = 2 \cdot \pi \cdot f \cdot t$$

Note that the convolution process in the time domain, applying the Fourier transform is a simple multiplication in the frequency domain (Equation 3.1e).

$$y(t) = x(t) * h(t) \rightarrow y(f) = x(f) \cdot h(f) \qquad [3.1e]$$

The Fourier transform is applied to any signal: periodic or no-periodic. However Fourier series can be only applied to periodic signals and it is defined throwing the equation 3.1f.

$$x(t) = a_0 + \sum_{n=1}^{\infty} a_n \cdot \cos(2 \cdot \pi \cdot f \cdot t \cdot n) - \sum_{n=1}^{\infty} b_n \cdot \sin(2 \cdot \pi \cdot f \cdot t \cdot n) \qquad [3.1f]$$

Where:

$$a_0 = \frac{1}{T} \cdot \int_{-T/2}^{+T/2} x(t) \cdot dt$$

$$a_n = \frac{2}{T} \cdot \int_{-T/2}^{+T/2} x(t) \cdot \cos\left(\frac{2 \cdot \pi \cdot t \cdot n}{T}\right) \cdot dt$$

$$b_n = -\frac{2}{T} \cdot \int_{-T/2}^{+T/2} x(t) \cdot \sin\left(\frac{2 \cdot \pi \cdot t \cdot n}{T}\right) \cdot dt$$

Some examples of Fourier series are shown in the figure 3.1a. On the left side of the picture is possible to see periodic signals[6] in the time domain and on the right side the Fourier series of these signals in the frequency domain.

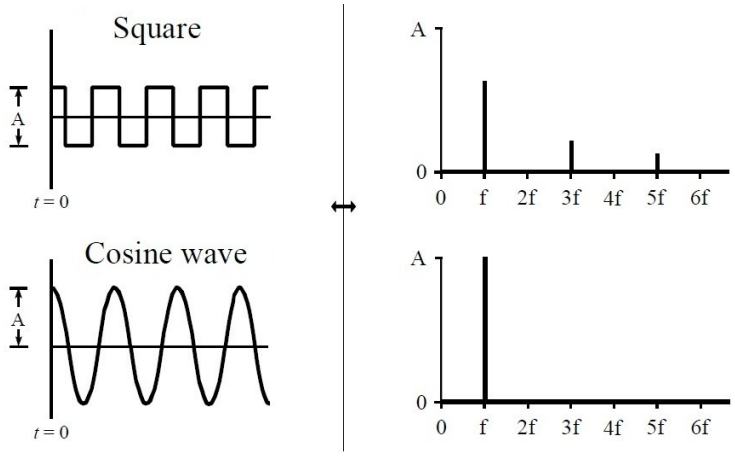


Figure 3.1a: Examples of the Fourier series. [9]

[6] A periodic signal is when it is repeated along the time. And a no-periodic signals is when it does not change along the time.

3.2 Filter Design.

A Matched filter and a Hilbert filter have been used to do the signal processing. Both filters were previously designed and simulated using the *MATLAB* program. And now they must be implemented into the *FPGA* device.

To accomplish this task, it was necessary studying the theory based on these filters. So in this section, the theory principles, characteristics, design and magnitude response of the filters are going to be explained in detail.

Once all the theoretical concepts are understood, both filters will be implemented using a (Finite Impulse Response) *FIR* structure. So the process to design a *FIR* filter is treated, and the design method and the structure of the filters are described, too.

3.2.1 The Matched Filter

The Matched filter is one of the filters used to process the echo signals. Some mathematical concepts are going to be explained to understand its working. And finally, the script in *MATLAB* which it designs the Matched filter is going to be explained, too.

Definition and mathematical analysis

The matched filter is a special kind of band pass filter but much more restrictive which maximizes the Signal Noise Ratio (*SNR)* of the filtered signal and it has an impulse response that it is the reverse time-shifted version of the input signal.

For instance, a signal $x_i(t)$ affected by a noise $n_i(t)$ is transmitted and filtered by a system $h(t)$. Then the expression at the output of the system is showed in the figure 3.2.1a.

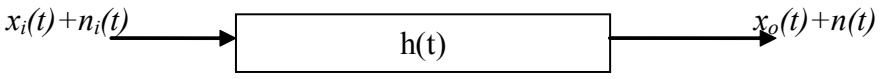


Figure 3.2.1a: Output signal with noise added

The main objective is to find out an expression for *h(t)* which maximizes *SNR* as large as possible in a time $t=t_d$ (equation 3.2.1a).

$$SNR_{\max} = \frac{|x_o(td)|^2}{|n_o(t)|^2} \qquad [3.2.1a]$$

Where $|xo(t_d)|^2$ is a simple convolution between the input $x_i(t)$ and the system *h(t)* (Equation 3.2.1b). And the noise ($|n_0(t)|^2$) is the term which it is necessary to minimize. It is assumed for instance, a Gaussian noise with a spectral density of $N_0/2$ (Equation 3.2.1c).

$$|x_o(td)|^2 = |x_i(td) * h(t)|^2 = |F\{h(f)\cdot x_i(f)\}|^2 = \left|\int_{-\infty}^{+\infty} h(f)\cdot x_i(f)\cdot e^{2\pi ftd} df\right|^2$$

[3.2.1b]

$$|n_o(t)|^2 = \frac{N_o}{2}\int_{-\infty}^{+\infty} |h(f)^2|\cdot df \qquad [3.2.1c]$$

The Schwarz inequality[7] is applied to maximize the *SNR*. And it is given by the equation 3.2.1d.

$$\frac{|x_0(t_d)|^2}{n_0{}^2(t)} \le \frac{2}{N_0}\int_{-\infty}^{+\infty} |x_i(f)|^2\cdot df \;\rightarrow\; \left.\frac{|x_0(t_d)|^2}{n_0{}^2(t)}\right|_{\max} = \frac{2}{N_0}\int_{-\infty}^{+\infty} |x_i(f)|^2\cdot df \qquad [3.2.1d]$$

But according to the Schwarz inequality, this is only true if the condition of the equation 3.2.1e is true, too. (Look at the equation 3.2.1f for the time domain).

$$h(f) = k\cdot x^*{}_i(f) \qquad [3.2.1e]$$

$$h(t) = k\cdot x^*{}_i(t_d - t) \qquad [3.2.1f]$$

[7] Appendix B: The Schwarz inequality.

Where k is an arbitrary constant and it can be used to modify the gain of the filter and x_i* is the complex conjugate of x_i, i.e. the time reversal image of the input signal.

Design and response

According to the equation 3.2.1f, the impulse response of a Matched filter is the time reversal image of the input signal. If the burst signal is *b(t)*, the matched filter impulse response is given by the equation 3.2.1g.

$$h(t) = k\, b^{*}(td-t) \quad [3.2.1g]$$

The coefficients of this filter are obtained using a script in *MATLAB*[8] that it finds out the impulse response of the filter.

So in a first step, the input burst signal has to be designed to get its time reversal image. It is composed by $N = 8$ periods with a frequency $F_0 = 8MHz$. But after various tests, the response of this filter is very good if $N = 24$. So finally the burst signal designed has *24* periods (figure 3.2.1b).

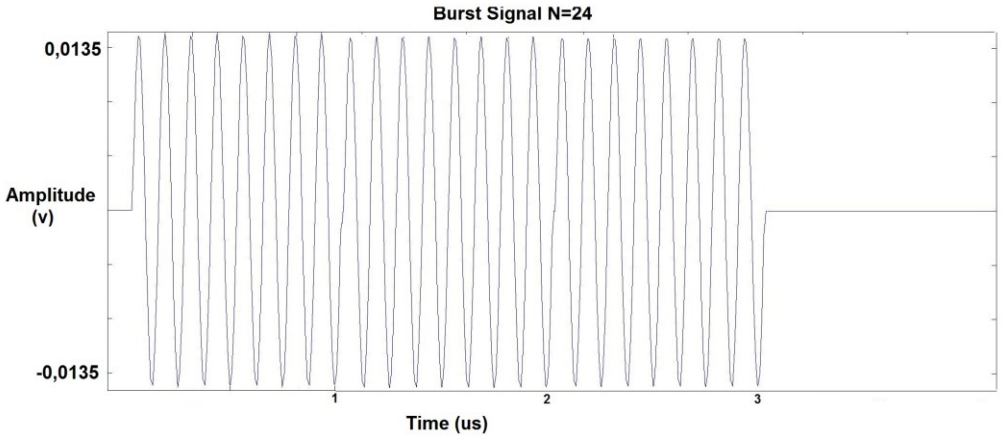


Figure 3.2.1b: Burst signal with $F_0 = 8\ MHz$ and $N = 24$.

[8] Appendix C: Matched Filter Script.

The order of the filter is directly related with the number of coefficients. If the sampling frequency $F_s = 25MHz$ and $N = 24$, the order of the filter is *74*. It is important to keep in mind that if the order of the filter is very high, a lot of hardware resources will be used. So it is necessary to find a good compromise between the filter order and the hardware resources to get an optimal implementation in the FPGA. In this case, it has been possible to implement the filter with *74* coefficients. But if we want to reduce the order of filter, two parameters can be modified: the sample frequency (F_s) and the number of periods (N). If the sample frequency is lower or the number of periods is decreased, the order of the filter will be reduced. The lowest sample frequency to get a successful response is $F_s = 20\ MHz$. And the lowest number of periods to get an acceptable response is $\boldsymbol{N} = 18$.

Figure 3.2.1c shows the magnitude response in *dB* of the filter. The main lobe is centred at the frequency of *8 MHz*. It means that signals with *8MHz* of frequency (aprox.) can only pass and for the rest one, their amplitudes will be attenuated until *40 db* when $F_{InputSignal} < F_0$ and until 30 db when $F_{InputSignal} > F_0$.

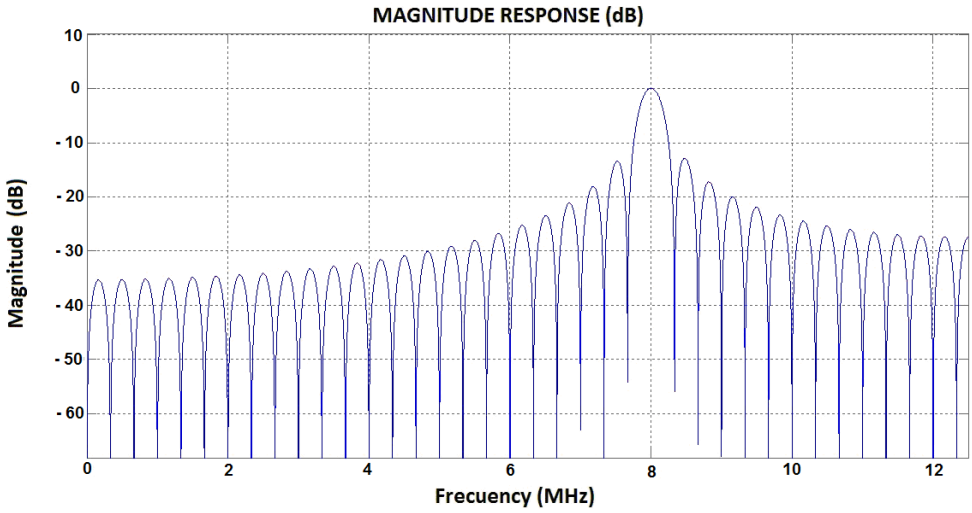


Figure 3.2.1c: Magnitude Response of the Matched Filter

3.2.2 The Hilbert Filter.

The Hilbert filter is the other filter used to process the echo signals. Firstly the mathematical concept is going to be explained. And finally, its design in *MATLAB* is going to be described.

Definition and mathematical analysis

The Hilbert filter is basically used to shift the incoming signals by 90º. Comparing its input and its output, it is possible to get the direction of the velocity profile.

The design of the Hilbert filter is based on the Hilbert transform. The Hilbert transform is able to convert any real sinusoid $A \cdot \cos(\omega \cdot t)$ to a positive frequency complex sinusoid signal $e^{j(\omega t+\theta)}$ generating a phase-quadrature component $A \cdot \sin(\omega \cdot t)$ as the "imaginary part" (Equation 3.2.2a).

$$A \cdot e^{j(wt+\phi)} = A \cdot \cos(wt+\phi) + jA \cdot \sin(wt+\phi) \qquad [3.2.2a]$$

When the signals are more complicated, they can be expressed as a sum of several sinusoids, and a Hilbert filter ($h_t()$) which shifts each sinusoid a quarter of cycle can be designed. The magnitude of this filter takes the value *1* for all the frequencies. And this filter introduces only a phase shift in the frequencies according to the following criterion:

- For the positive frequencies ➔ phase shift of $-\pi/2$
- For the negative frequencies ➔ phase shift of $+\pi/2$

$$H_t(f) = \begin{cases} -j = e^{-j\frac{\pi}{2}} \rightarrow f > 0Hz \\ 0 \rightarrow f = 0Hz \\ +j = e^{+j\frac{\pi}{2}} \rightarrow f < 0Hz \end{cases} \qquad [3.2.2b]$$

It could be interesting to find out what happens when *x(t)* is added to the Hilbert transform $h_t(x)$. A new complex sinusoid signal *z(t)* (called analytic signal) which has not negative frequency is formed [equation 3.2.2c].

$$z(t) = x(t) + jh_t(x) \qquad [3.2.2c]$$

Equation 3.2.2d shows a real signal *x(t)* which phase is going to be shifted $\pm j$ to get a complex sinusoid

$$\begin{aligned} x(t) &= x_+(t) + x_-(t) \\ x_+(t) &= e^{jw_0 t} \\ x_-(t) &= e^{-jw_0 t} \end{aligned} \qquad [3.2.2d]$$

The phase shifts of $-\pi/2$ (-90º) is applied to the positive frequency and $+\pi/2$ (+90º) to the negative frequency. Then, it is possible to get the equation 3.2.2e.

$$\begin{aligned} y(t) &= y_+(t) + y_-(t) \\ y_+(t) &= e^{-j\frac{\pi}{2}} e^{jw_0 t} = -je^{jw_0 t} \\ y_-(t) &= e^{+j\frac{\pi}{2}} e^{-jw_0 t} = +je^{-jw_0 t} \end{aligned} \qquad [3.2.2e]$$

Therefore, the analytic signal *z(t)* can be rewritten as the equation 3.2.2f because $y(t) = h_t(x)$.

$$\begin{aligned} z(t) &= z_+(t) + z_-(t) \\ z_+(t) &= x_+(t) + jy_+(t) = e^{jw_0 t} - j^2 e^{jw_0 t} = 2e^{jw_0 t} \\ z_-(t) &= x_-(t) + jy_-(t) = e^{-jw_0 t} + j^2 e^{-jw_0 t} = 0 \end{aligned} \qquad [3.2.2f]$$

In the equation 3.2.2f is possible to see that for negative frequencies of *x(t)*, their components are filtered out and for positive frequencies, their components are not filtered out and they have also a gain of 2.

Design and response.

The coefficients of this filter are obtained using a script in *MATLAB*[9]. A low order ($N = 6$) and a small pass-band (given by the frequency vector) has been chosen in order to save hardware resources.

The frequency vector is normalized to the Nyquist frequency, that it is the half of the sample frequency. So the frequency normalized is expressed by the equation 3.2.2g:

$$F_n = F \cdot \left(\frac{2}{F_s} \right) \quad [3.2.2g]$$

Where F is the frequency original, and F_s is sample frequency (*25MHz*).

The normalized frequency of the burst signal comes up to F_n (*8MHz*) = *0.64*. The frequency vector is built from *3.75MHz* to *8.75MHz*. It is enough because previously, the burst signals have already been filtered out at higher and lower frequencies using the Matched filter.

Figure 3.2.2a shows the magnitude response of the Hilbert filter. The range of frequencies is approximately *4-8 MHz* and the gain of this filter is *0 db*.

[9] Appendix D: Hilbert Filter script.

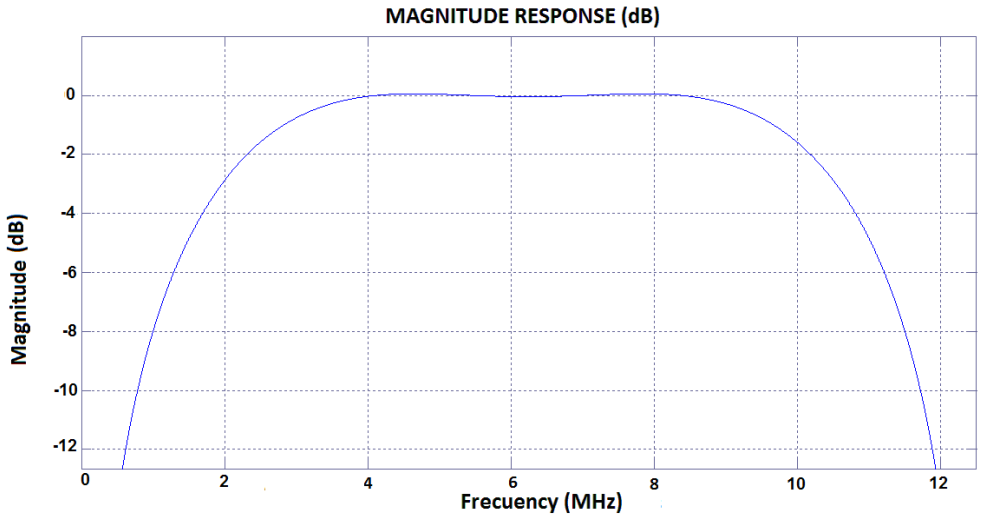


Figure 3.2.2a: Magnitude Response of the Hilbert Filter

3.3 Finite Impulse Response Filters.

A Finite Impulse Response filter (*FIR* filter) is a type of filter with a finite duration impulse response.

The behaviour and performance of a *FIR* filter can be defined by:

- Filter Coefficients: they are the impulse response of the filter. The *FIR* coefficients are a set of constants which are multiplied by delayed samples values.
- Impulse Response: it is the output of the filter in the time domain when the input is an impulse. The impulse response of a *FIR* filter is the set of the filter coefficients.
- Tap: The taps of a *FIR* filter describe the amount of memory needed, the number of calculations required, and the amount of "filtering" that the *FIR* filter can do.
- Multiply-Accumulate (*MAC*): it is the multiplication operation of a coefficient by the corresponding data delayed, accumulating its result. There is usually one *MAC* per tap.

The output of a discrete-time *FIR* filter is given by the sum of the current and previous input values. Equation 3.3a is the mathematical equation which describes the output of a *FIR* filter.

$$y[n] = b_0 x[n] + b_1 x[n-1] + ... + b_N x[n-N] = \sum_{i=0}^{N} b_i x[n-i] \qquad [3.3a]$$

Where:

x[n]: input signal,
y[*n*]: output signal,
b_i: filter coefficients.
N: filter order.

Structures

There are multiples structures to implement a *FIR* filter and the most used are the direct-form (figure 3.3a) and the direct-form transposed (figure 3.3b). The first one is the most common, but it spends a lot of hardware resources because it uses a lot of adders. However the second one has small additions separated by delay elements and it works better in a *FPGA* saving area and optimizing resources. Furthermore, the direct-form structure needs extra pipeline registers between the adders to reduce the delay of the adder tree and hence, to achieve a high throughput. However, the direct-form transposed structure achieves high throughput without adding any extra pipeline registers.

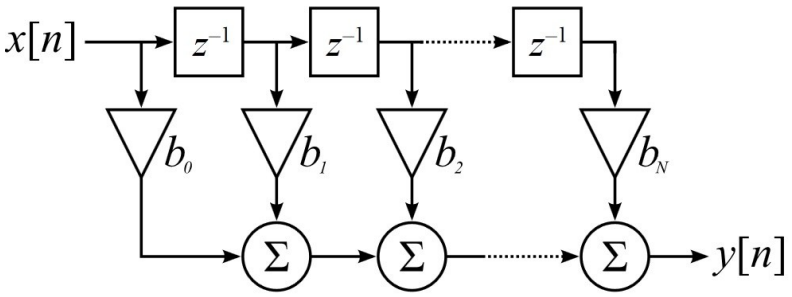


Figure 3.3a: Direct-Form Structure. [10]

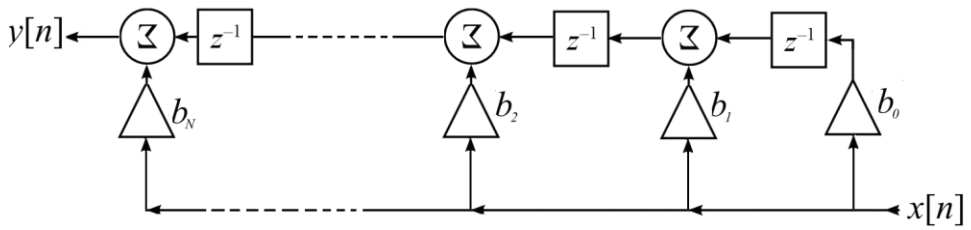


Figure 3.3b: Direct-Form transposed structure for a second order filter. [12]

For this task, eight filters have to be implemented in the National Instrument (NI) system. And the structures were tested on it. The direct-form *FIR* structure was tested with only 2 filters. And it spent a *50%* of the *FPGA* resources. Then, it would be impossible to implement the eight filters in the *FPGA*. However the direct-form transposed *FIR* structure was tested with the eight filters. And it spends the *70%* of the *FPGA* resources. So finally the last one structure is going to be used for the implementation of the digital filters in the *FPGA* device.

Quantizing: Fixed Point Vs Floating Point

FIR filters are often implemented using multipliers and adders placing according to the structure selected (Direct-Form transposed in this case). If these components have to be programmed in a *FPGA*, the complexity and the area will be higher or lower according to the data type used to make these operations.

Fixed point and floating point are the two data type available to process a signal. The numbers in fixed point (FXP) are integers (positive and negative) with a minimum size of 16 bits, fielding up to *65,536* possible bit patterns (2^{16}). Fixed point term means how numbers are represented with a fixed number of digits after or before the decimal point (*123.45, 1234.56, 12345.67*, etc). However the floating point numbers have a size of *32* bits and they are represented with a mantissa and an exponent. The placement of the decimal point in the floating point numbers, can 'float' between the significant digits of the number (*1.234567, 123456.7, 0.00001234567, 1234567000000000*, etc).

In conclusion, the way to represent the largest and smallest number (dynamic range) and precision is better in the floating point number.

When a *FIR* filter is designed, their coefficients are usually expressed in a high precision data type (floating point). The *FPGA* system does not work with numbers in floating point. So they have to be converted to fixed point using a rounding process. This rounding process introduces errors to magnitude response of the filter and the quantized filter may not meet with the filter requirements. It is fundamental to check the magnitude response of the filter after the data type conversion before doing anything. And if the difference between both designs is not very high, you can go on with the implementation. In the section 5.2.1 is possible to see the difference between the coefficients in floating point and *FXP*.

The *ADC* used has *14 bits* of resolution. Then, the data received has to be stored in a variable which data type is an integer of *16 bits*. It means that *14 bits* are going to be the real part and the remaining *2 bits* are going to be the decimal part of the data received.

4. The National Instrument System

The National Instrument system used is composed by a *NI 5761* target (Analog Digital Converter, *ADC*), a *NI FlexRIO PXIe-7965R* (*FPGA* module) and a *NI PXIe-1073* (chassis-communication card):

Figure 4: NI System

With the *NI 5761* and the *NI PXIe-7965R* is possible to get powerful solutions for applications which requirements are: a high-speed digitizer and custom real-time processing.

4.1 NI PXIe-7965R

The *NI FlexRIO NI PXIe-7965R* device uses one of the most powerful *FPGA* of the market: *Xilinx Virtex-5 SXT*. It is totally recommended for high-speed digital signal processing and continuous filtering.

Figure 4.1a*: NI PXIe-7965R* and *VIRTEX V FPGA* [15]

Programming a *FPGA* is not an easy task. Description hardware languages like *VHDL* or *Verilog* are the most popular to do it. However the software of *NI* (*LabVIEW*) provides an easy way to program it using a graphical language ('*G*' language). When *LabVIEW* is installed, several additional packages have to be installed according to the project to develop. For this purpose, the *FPGA* module package has to be installed. And this module includes a feature for *HDL IP* integration called Component Level *IP* (*CLIP*). The *NI PXIe-7965R* device supports two types of *CLIP* (Figure 4.1b):

- User-defined *CLIP* allows inserting *VHDL* code into a *FPGA* target, enabling *VHDL* code to communicate directly with the *FPGA* LabVIEW program called usually *FPGA VI*. [16]
- Socketed *CLIP* provides the same *IP* integration functionality of the user-defined *CLIP*, but also allows the *CLIP* to communicate directly with circuitry external to the *FPGA*. Adapter module socketed *CLIP* allows communicating the *IP* directly with the *FPGA VI* and the external adapter module connector interface. [16]

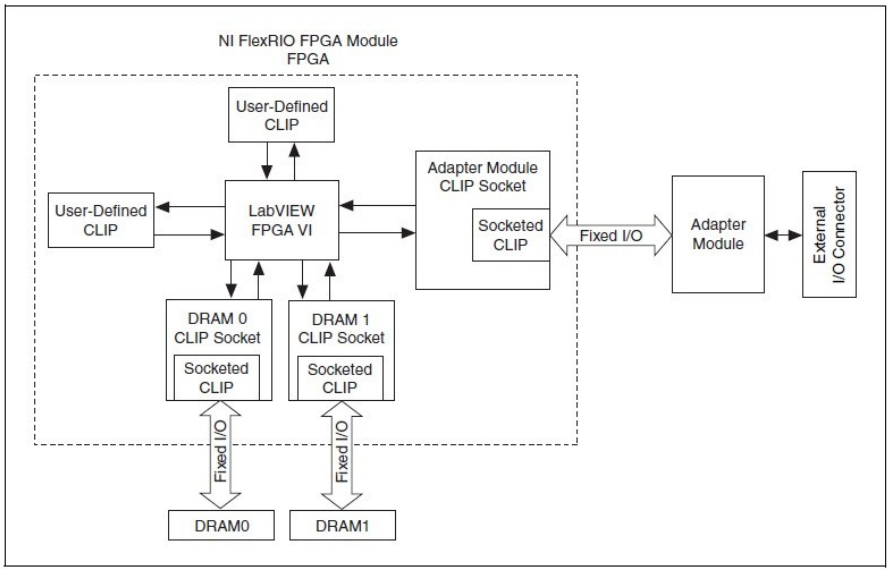


Figure 4.1b: CLIP Relationship. [16]

4.2 NI 5761

This *ADC* module has *4* channels with *14* bits of resolution and it is able to acquire samples at *250Ms/s* simultaneously by the *4* channels.

Figure 4.2a: *NI 5761*. [17]

The *NI 5761* is controlled by the *FPGA*. So the configuration lines and the data lines of the *ADC* have to be mapped in the *FPGA* using an adapter module and a *LabVIEW FPGA CLIP* interface to connect the hardware *NI5761* (*ADC* card) to the *NI PXIe-7965R* (*FPGA* module).

***NI 5761* Adapter Module**

The *NI 5761* digitizer adapter module has two important chips: The Texas Instruments *ADS62P49* Analog to Digital converter (*ADC*) chip that it is ideal for Intermediate Frequency (*IF)* and Radio Frequency (*RF)* communications applications. And the Analog Devices *AD9512* clock generator chip that it is recommended for high speed *ADCs*, Digital to Analog Converters (*DACs)* and other devices.

The *ADS62P49* is a family of dual channel *14-bit A/D* converters with sampling rates up to *250 MSPS*. At every falling edge of the input clock, the analog input signal of each channel is sampled simultaneously and the digital code available at the output can be presented in binary (*0x0000 - 0x3FFF*) or *2s* complement format (*0x1FFF - 0x2000*). The working modes of the *ADC* can be configured using a serial interface programming (*SPI*[10]). A special interest it is the Multiplexed Output Mode or Low Speed Mode which it is only recommended for low sampling frequencies < *65MSPS*. [18].

The *AD9512* provides a multi-output clock distribution that emphasizes low jitter and low phase noise to maximize data converter performance. It is ideally suited for data converter clocking applications where maximum converter performance is achieved by encode signals with subpicosecond jitter[11]. [19]

The sample frequency desired for the *ADC* chips (*NI 5761*) is *25 MHz*. The *AD9512* chip has two independent inputs (*CLK1* and *CLK2*). And it is possible to introduce an external clock signal using *CLK1* or *CLK2*. This external clock signal can be directed by one of the five independent clock outputs of the *AD9512* and it can be connected to the external clock input of the *ADC* (*ADS62P49*).

LabVIEW FPGA CLIP

The *NI 5761* target (*ADC*) ships with several types of socketed *CLIP* items to add to the *LabVIEW* project: Multi Sample, Single Sample and Low Speed *CLIP*. And the last one is the socketed *CLIP* used. This socketed *CLIP* acquires one sample per clock cycle at the maximum sample rate of *150 MHz*. And it's possible to set a lower sample rate using an external clock connector. The Low Speed characteristic of this socketed *CLIP* item allows working the *ADC* with a very low sample clock, configuring in the right way the internal registers of the *ADC* using the *SPI* protocol. Then it is possible that the *ADC* works at *25Ms/s* using the *CLK IN* connector of the card.

[10] Appendix E for more information about the *SPI* protocol.
[11] Appendix F The phase jitter, phase noise and time jitter.

Figure 4.2b describes the signals of this *CLIP*. Not all of them are used. So the most important are only explained:

- *SPI* idle output (Boolean): The *NI 5761* has two *ADS62p49 ADC* chips with two channels. The *ADS62p49 ADC* chip needs a CLK chip (*AD9512*) to work properly. The *SPI* idle output indicates when the *SPI* interface to either the *CLK* chip or *ADCs* is being accessed. It's necessary wait for this bit to be *TRUE* to use the *SPI* or Apply Settings signals.
- Sample Clock Select input (Unsigned *8* bits): it's used to select the clock source (external clock *CLK IN* connector). So the *U8* value to configure in this input is *2*.
- Sample Clock Commit input: When changed from *FALSE* to *TRUE*, updates the sample rate.
- Synthesizer Locked output: Indicates when the onboard clock synthesizer is locked. When the clock synthesizer block synthesizes the different clock sources, the *AD9512* generates the clock for the *ADS62P49* chips.
- Configuration Error output: it indicates with a *TRUE* Boolean value that an error occurred while has been enable the *ADC* module.
- Synthesizer Locked output: it indicates that the onboard clock synthesizer is locked.
- Enable Low Speed Mode: it configures the *ADC* in the Low Speed mode *(< 65MSPS*).

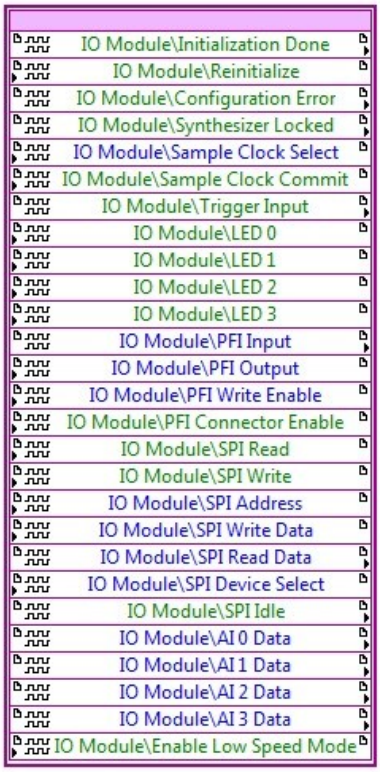


Figure 4.2b: *NI 5761* Low Speed *CLIP* Signal Block Diagram. [20]

4.3 NI PXIe-1073

It is the chassis where the cards are placed (Figure 4.3). It has five slots to insert NI cards, but only one is used.

A *MXI-Express* controller is integrated to link with the host computer at least at *208 MB/s*.

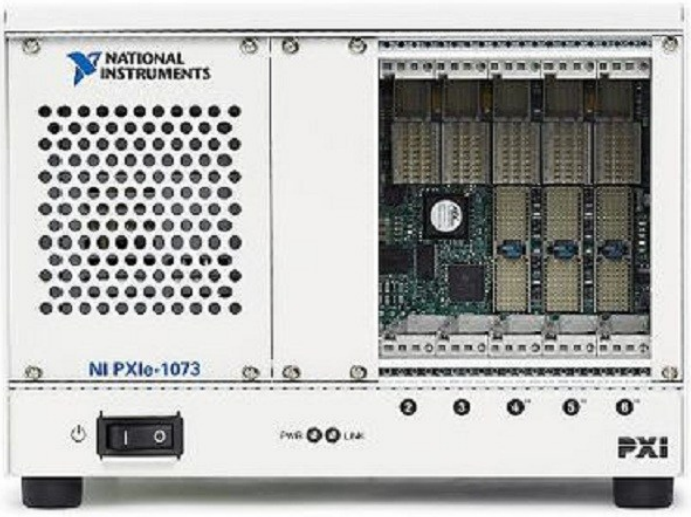


Figure 4.3: NI PXIe-1073. [15]

5. FPGA Signal Processing

One solution was presented in the section 2.2 to do the signal processing faster. This solution proposes implement a part of the signal processing in the *FPGA*.

MATLAB and *LabVIEW* are the software tools used to design/implement the filters in the *NI* System (*FPGA*). *MATLAB* designs the Matched and Hilbert filters giving their coefficients in floating point data type. And *LabVIEW* implements those filters in the FPGA and it also implements a down-sampling algorithm to reduce the amount of data to transfer to the personal computer (PC). This information will be read by *MATLAB* and it will finally give the velocity profile.

The way to work in *LabVIEW* is using visual interfaces files (**.vi*). Each "*VI*" file has a block diagram which implements the different processes to get the final results. Basically there are two important *VIs* in this *LabVIEW* project: The *FPGA VI* and the *HOST VI*.

In the *FPGA VI* is designed the block diagram that it is going to be programmed in the *FPGA*. This block diagram has the next important functions:

1) Configuring and Programming the *NI 5761* module (*ADC*) in the low-speed mode.
2) Acquiring the samples.
3) Processing the Samples: Filtering and Down-sampling.
4) Transferring the samples to the *HOST VI.*

The *HOST VI* will receive the data from the *FPGA* and write them into files. But firstly, it is necessary to finish the design of the filters. We need to convert the coefficients from floating point to fixed point, because the *FPGA* only works with this type of data.

5.1 Filter Design VI

An extra *VI* is designed to convert the coefficients from floating point to fixed point data type and therefore, it creates the *.fds file with code compatible with *LabVIEW* to will program the filters in the *FPGA*.

LabVIEW has the possibility to import and execute *MATLAB* scripts using a tool called "*Mathscript*". Attending to the figure 5.1 several steps have to be followed to get the final design of the filters. Firstly, the coefficients of the filter in floating point data type are read from the script (1) and then the function transfer of the filter is obtained (2).Later, the filter is built according to the structure selected (3), and afterwards the coefficients are quantified and converted in fixed point data type (4). Finally the file **.fds* is gotten (5).

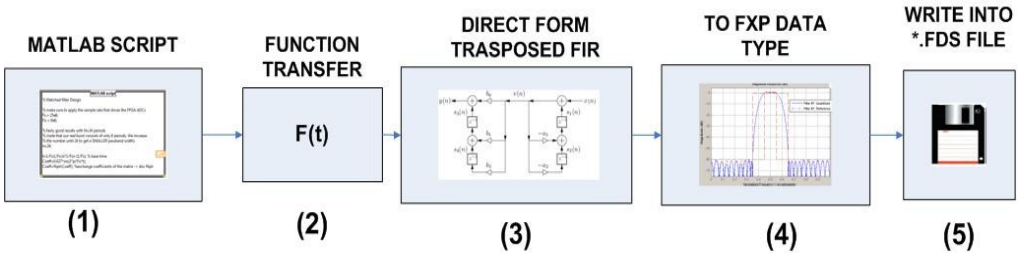


Figure 5.1: Block diagram which design the filter in *FXP* and get the **.fds* file compatible with *LabVIEW*.

The Fixed Point resolution

It is important to define the properly resolution of the fixed point data type to avoid quantification errors. The word length and the integer word length fixed point parameters have to be configured for the input and the output of the filters according to the resolution of the *ADC*.

The *ADC* resolution is *14* bits. So the filters must be built with a standard input/output of *16* bits (word length). And moreover, at least *14* bits should be the integer part and the rest should be the decimal part. So the integer word length configuration should be:

For the input:

- Word length: *16* bits
- Input Word Length: *14* bit

For the output:

- Word length: *16* bits
- Output Word length: User Defined (*14* bits)

5.2 FPGA VI

Every *FPGA* is composed of a finite number of predefined resources with programmable interconnects. These interconnects implement the digital circuit to design with *LabVIEW*. When the *FPGA VI* is created, a circuit schematic describes how logic blocks are wired together on the *FPGA*. The compilation tools of *LabVIEW* translate the *FPGA VI* into the *FPGA* circuit (Figure 5.2a).

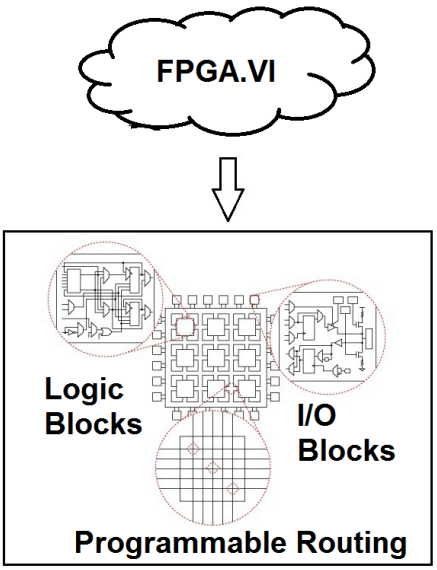


Figure 5.2a: Translating the *FPGA VI* to a *FPGA* circuit. [20]

Several rules have to be followed when a *NI FlexRIO* device (*NI PXIe-7965R*) is programmed, because it's important to generate a synthesize code to optimize the speed and the space in the *FPGA*. For instance, a *FPGA* chip

does not work with numbers in floating point, so we have to convert them into fixed point data type.

The block diagram of the *FPGA VI* program is split in two parts: the *ADC* initialization and the Main Program (figure 5.2b).

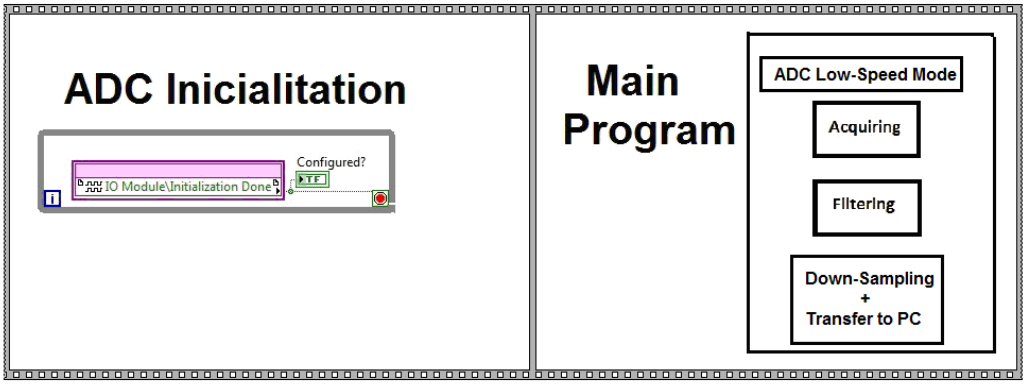


Figure 5.2b: *ADC* initialization and main program

When the bit file is downloaded into the *FPGA*, the block diagram before is executed sequentially from the left to the right side. The *FPGA* will wait in a while loop until the *ADC* is configured successfully. Then the Main program will be executed.

The main program contains four important loops:
1) The low-speed configuration *ADC* loop.
2) The acquiring loop.
3) The processing loop.
4) The transferring loop.

The low speed configuration loop is configured on real time using a while loop where several parameters are configured to put the *ADC* in the right working mode. An external source clock of *25 MHz* (generated by a function generator, for instance) is necessary to configure the *ADC* at *25 MSPS*. So in the configuration loop, the "Sample Clock Select" input has also to be configured as *2* (figure 5.2c) and the Boolean "Enable Low Speed Mode" input has to also be *TRUE* to enable the low speed mode of the *ADC*

Sample Clock Select	U8	Selects your clock source from one of the following options: 0 = Internal Sample clock 1 = Internal Sample clock locked to an external Reference clock through the CLK IN connecto 2 = External Sample clock through the CLK IN connector 3 = Internal Sample clock locked to an external Reference clock through IoModSyncClock 4 = External Sample clock through IoModSyncClock These changes do not take effect until you assert the Sample Clock Commit signal.

Figure 5.2c: Table provides by the manufacturer of the *NI 5761* to configure the *CLIP* with the desired clock source. [16]

In the acquiring loop the samples from the *ADC* are acquired at *25 MHz*. Figure 5.2d shows the Moore machine implemented to get this purpose. In this loop, the acquisition algorithm is implemented. It is waiting until a software trigger or hardware trigger is armed. At least it does not know what type of trigger it is. So in a second state, the type of trigger is evaluated and then the acquisition starts. A sample counter starts to run and when it counts until the limit fixed by one variable configured in the *HOST VI* the acquisition stops and it waits for another trigger and so on.

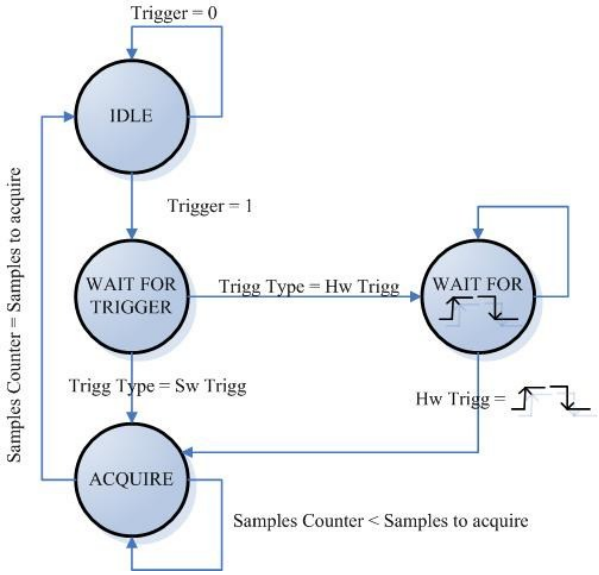


Figure 5.2d: Moore machine to acquire samples from the *ADC*.

The acquiring loop is a timed loop (single-cycle) where the timing corresponds exactly to the clock rate of the *FPGA* clock. Code inside a single-cycle Timed Loop is more optimized and takes up less space on the *FPGA* compared to the same code inside other type of loops because it removes registers and other components which they can increase the final size of the program. When this loop is used, all operations inside must fit within one cycle of the *FPGA* clock, because otherwise a timing error is reported.

When the acquisition starts, the samples are filtered at the same time using a processing loop. And finally in the transfer loop the samples are down-sampled and transferred to the host computer.

The capability to execute all the code in a concurrent way means that all the loops of the *FPGA VI* are executed in "parallel" or "concurrent" working mode achieving a very high throughput.

Set of *FIFO* memories are used to transfer the samples from one loop to another loop. Basically there are two types: the Target Scoped *FIFO* and the Target to *HOST – DMA FIFO*. The difference between them is that the Target Scoped *FIFO* can only be used to transfer data inside the *FPGA VI* and the Target Scoped *FIFO* can only be used to transfer data from the *FPGA VI* to the *HOST VI*.

The input of the matched filter is a *FIFO* memory which collects the samples in the acquire loop. And the output of this filter is connected to two *FIFO* memories. One of them will be the input to the Hilbert filter and the other one will be one of the outputs of the system (because the other output of the system will be the output of the Hilbert filter). Figure 5.2e shows a block diagram with the *FIFO* memory connections between all the loops for only one channel of the *ADC*.

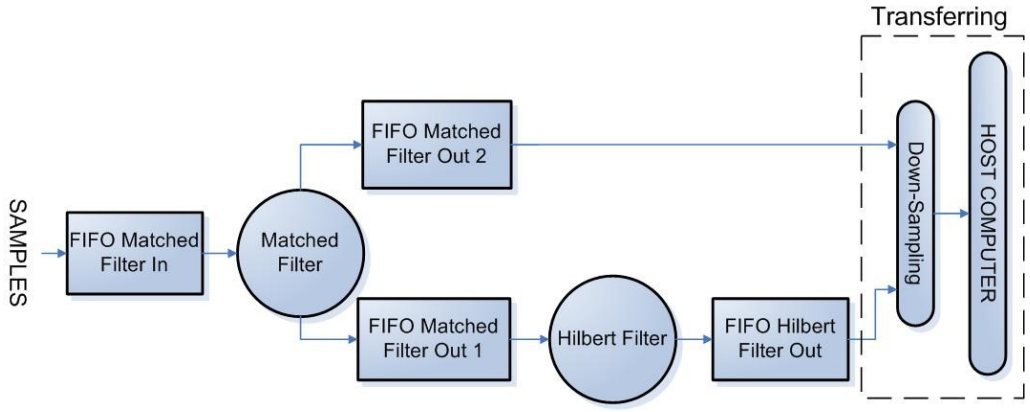


Figure 5.2e: Connections between the loops for one channel.

FPGA filter implementation.

In the section 5.1 was explained how it was possible to get a **.fds* file using the Filter Design *VI*. Now with the *IP* Generator tool of *LabVIEW* this **fds* file is going to be loaded. (Figure 5.2f)

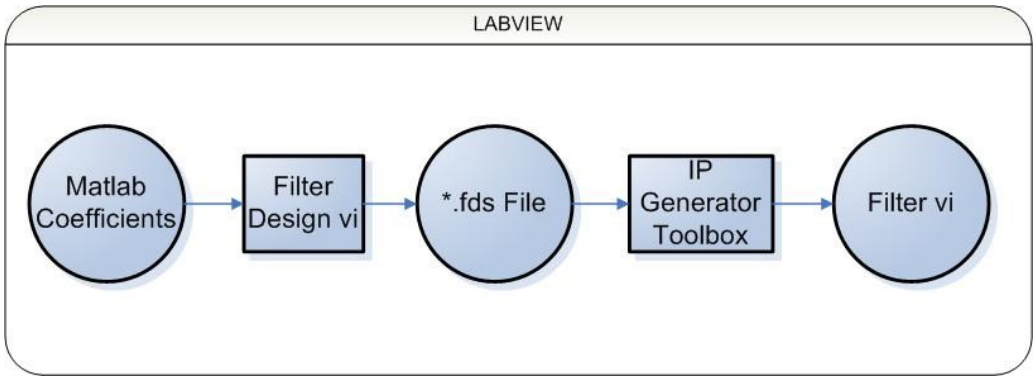


Figure 5.2f: **.fds* VI to Start *IP* Generator.

The *IP* Generator tool of *LabVIEW* is the easiest way to implement a *FIR* filter in a *FPGA*. It allows to create a synthesizable block diagram from **.fds* files. This tool gives the magnitude response of the *FIR* filter and then, it is important to check the quantification errors produced by the conversion of the coefficients.

Figure 5.2g shows the magnitude response difference between both data type for the Matched and Hilbert filters. And it is possible to appreciate that the differences are minimal. Thus the implementation is possible.

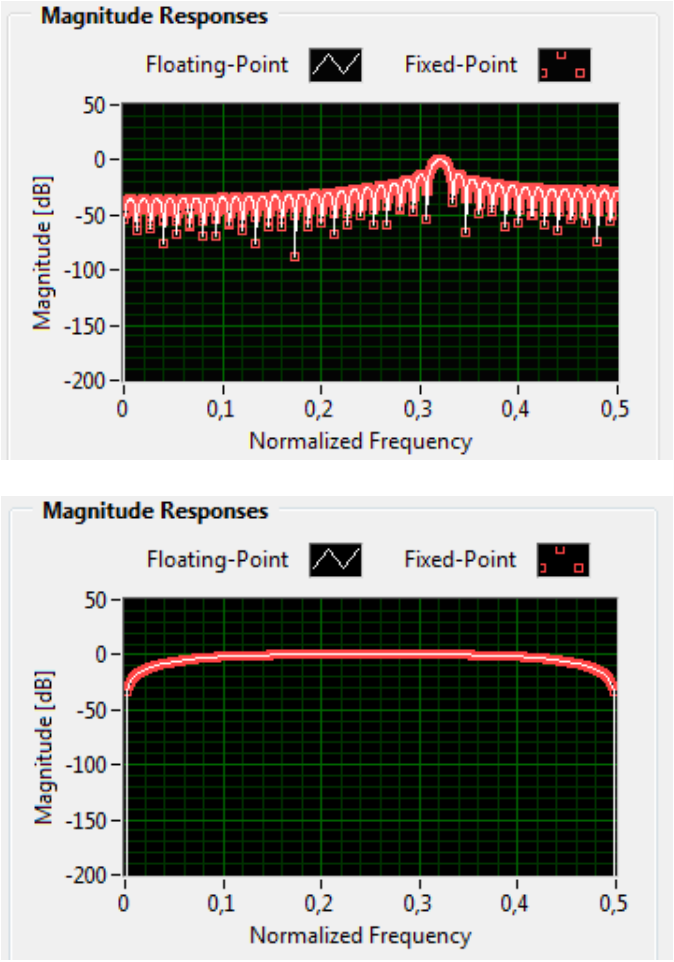


Figure 5.2g: Magnitude response of the Matched (on the top) and Hilbert (on the bottom) filters Floating Point Vs Fixed Point .

There are two methods to do the implementation of the *FIR* filters: Distributed Arithmetic (*DA*) and Multiply-Accumulate (*MAC*). According to the method selected the throughput of the filter could be higher or lower. Depending on the structure selected to build the filters (direct-form transposed

FIR), *DA* sometimes offers a higher throughput than *MAC*. But the first one spent more space and more resources in the *FGPA* than the second one. So it is important to find a compromise between the throughput and the resources.

The throughput of a filter is expressed in *cycles/sample*. It indicates the minimum input intervals between two adjacent inputs. So the smaller *cycles/sample* will be for the higher throughput and therefore the sample frequency supported by the filter at the input will be higher too. Given the throughput in *cycles/second*, the equation 5.2a calculates this throughput in *Mega Samples/second* (*MS/s*):

$$Throughput(MS/s) = \frac{FPGAClockRate(s/cycles)}{Throughput_0(cycles/sample)} \qquad [5.2a]$$

For instance, if the *FPGA* clock rate for a *FIR* filter is *40 MHz* and the $Throughput_0$ selection is *4 cycles/sample*, the *Throughput* of the generated *FIR* filter is *10 MS/s*. However, in the *Virtex 5 FPGA* chip, a *FIR* filter can run at 1*00 MHz*. So the filter throughput is increased *2.5* times. Attending to the maximum sample rate (*25 MHz*) it is important to choose the right maximum $Throughput_0$ of the filters. For the Matched filter and the Hilbert filter the maximum $Throughput_0$ using a *MAC* structure is *4 cycles/samples*. If the arithmetic selected is *DA,* the maximum $Throughput_0$ does not change because the direct-form transposed *FIR* structure does not offers a lot of possibilities of optimization.

The main task is to implement four Matched filters and four Hilbert in the *FPGA* (four ADC channels). The resources spent out are directly proportional to the number of coefficients of the filters. So the Hilbert filter with only *6* coefficients would not be a problem. However the *74* coefficients of the Matched filter are a problem, because this implementation spends out a lot of resources. After making several tests, the resources spent in the *FPGA* with a *DA* implementation were 50% only for one channel. So when the four channels were implemented, all the resources were spent out. However, using a *MAC* implementation, the resources used were *74%* for the *4* channels. So finally the *MAC* implementation was used to program the filters in the *FPGA*.

Once all the parameters are configured, the *IP* generator builds a *VI* with the filter inside. The *MAC* arithmetic of the filters is executed inside a timed loop

(single cycle) running at *100 MHz*, four times more than the acquisition loop (*25 MHz*). So the filter is going to process the acquired sample *x* before arriving the next acquired sample *x-1*.

Figure 5.2h shows a block diagram of the Acquisition loop, Matched filter loop, Hilbert filter loop and transferring loop are implemented in the *FPGA*. All the loops are executed continuously in a parallel working mode.

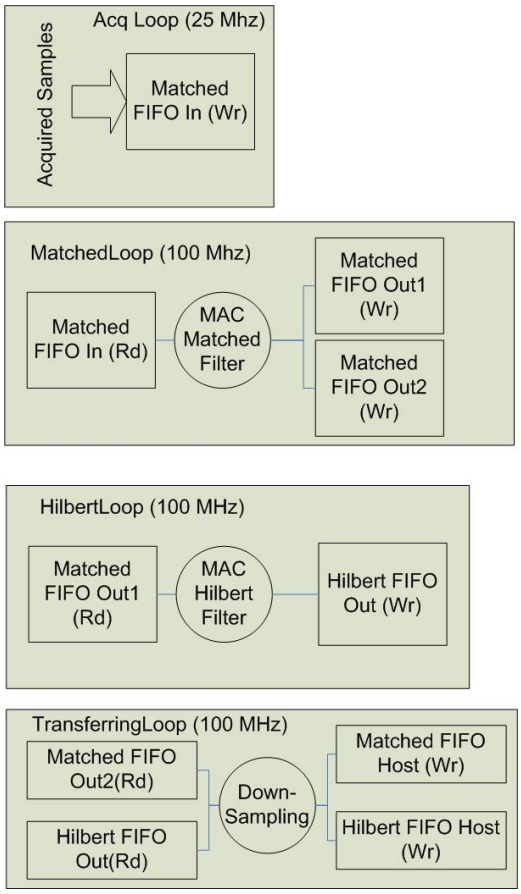


Figure 5.2h: Interconnection between the *FIFO* memories in the different loops.

In the Matched filter loop, the samples are read from the *FIFOs* with the samples acquired in the acquisition loop (Matched *FIFO* in). Then they are

filtered out, and the resulting data is written in two *FIFO* memories (Matched *FIFO* Out1 and Matched *FIFO* out2). The Matched *FIFO* Out1 is going to be read in the Hilbert filter loop, and this filter is going to shift the incoming signal *90º* storing the resulting data in another *FIFO* memory (Hilbert Filter Out).

In the transferring loop, the samples stored in the *FIFO* memories of the filters (Matched *FIFO* Out2 and Hilbert *FIFO* Out) are read. Then, if it was necessary to reduce the amount of data to transfer to the PC, a down-sampling algorithm would be applied. And in the last step, all the samples are stored in the Target Scoped *FIFO* memories (Matched *FIFO* Host and Hilbert *FIFO* Host) to transfer them to the host computer (*HOST VI*).

The Down-sampling algorithm

The down-sampling algorithm is basically a counter which throws away the samples that you don't want to transfer to the *HOST VI*. The Moore Finite State Machine (Moore *FSM*) of the figure 5.2i shows how it works.

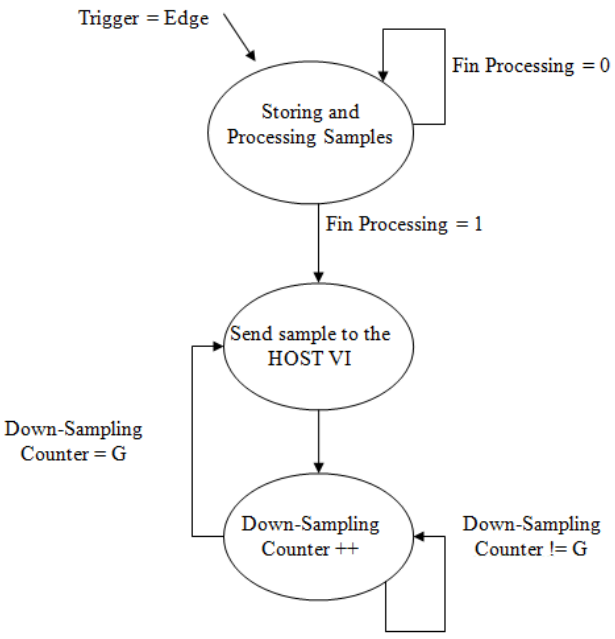


Figure 5.2i: Moore *FSM* of the Down-sampling algorithm

The algorithm is waiting until all the samples are processed by both filters and stored in the *FIFO* memories (Matched *FIFO* Out2 and Hilbert *FIFO* Out). Then the number of sample *G* will be only sent to the *HOST VI*. *G* is the number of the samples that you want to throw away. For instance, one trigger acquires *1600* samples. If the down-sampling variable takes the value *50*, it means that the samples number *0, 50, 100, 150, 200, ..., 1550*[12] are going to be sent to the *HOST VI*.

The Group delay of the Hilbert Filter

The Hilbert Filter is used to shift *90º* the output of the Matched filter. Afterwards the outputs of both filters are down-sampled and transferred to the host computer. But the response of the Hilbert filter delays a little quantity of time (group delay). Figure 5.2j shows the group delay of the Hilbert filter. It is expressed in samples and it means that *3* samples are necessary to throw away to get the right value at the output of the filter.

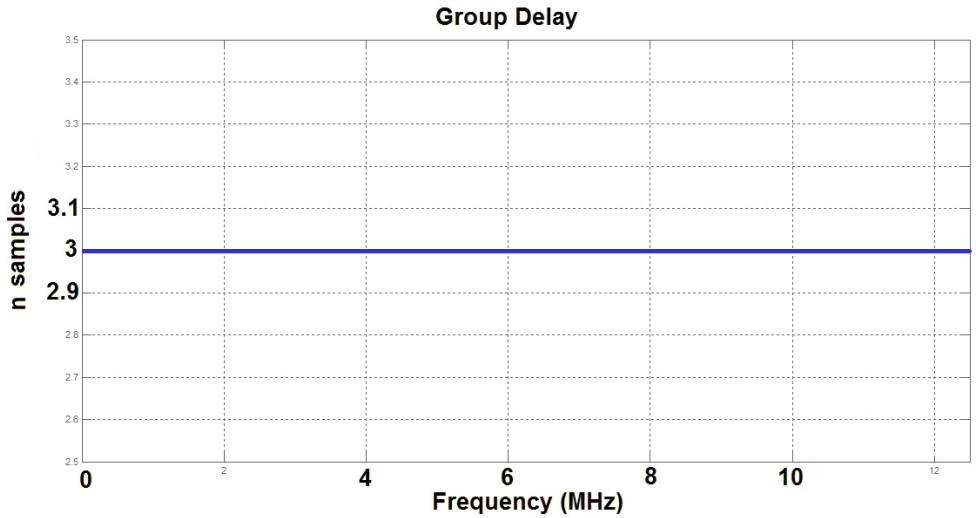


Figure 5.2j: Group delay of the Hilbert Filter.

[12] Note: The total number of samples to send to the HOST VI are 1600/50 = 32 samples. It is important to take into account that the first sample sent is the sample placed in the position 0 of the FIFO and the last sample sent is the 1599. So the sample placed in the position 1600 is not sent.

The group delay has to be taken into account when the direction of the velocity profile is gotten. It means, if at the output of the Matched filter a sample *n* is given, then at the output of the Hilbert filter three samples have to be thrown away to get the sample *n* "shifted" *90º*. So in the transferring loop is necessary to do a modification before writing the samples in the *FIFO* Hilbert Host (Figure 5.2k).

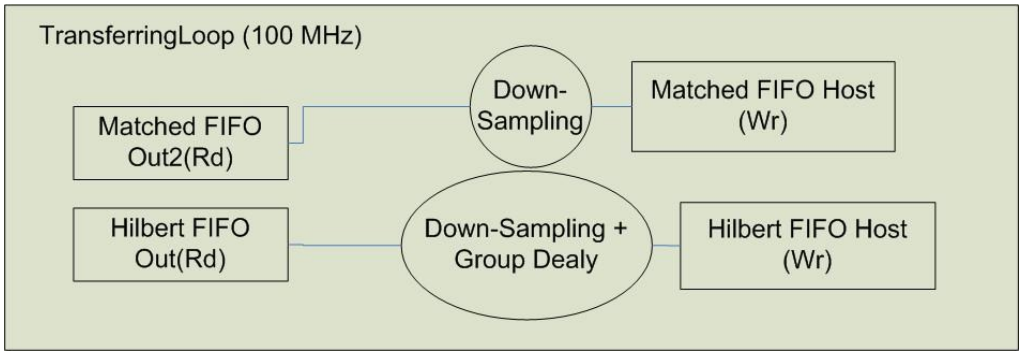


Figure 5.2k: Taking into account the group delay.

The More *FSM* of the figure 5.2i has to be modified due to the Group delay, too. But only for the Hilbert filter (Figure 5.2l). There are several options to implement the group delay, but the best of them is doing the count of the down-sampling counter, *3* samples longer. For instance, if the value configured in the down-sampling variable is *50*. Then, the samples number *0+3, 50+3, 100+3, 150+3, 200+3, ..., 1550+3* are going to be sent to the *HOST VI*.

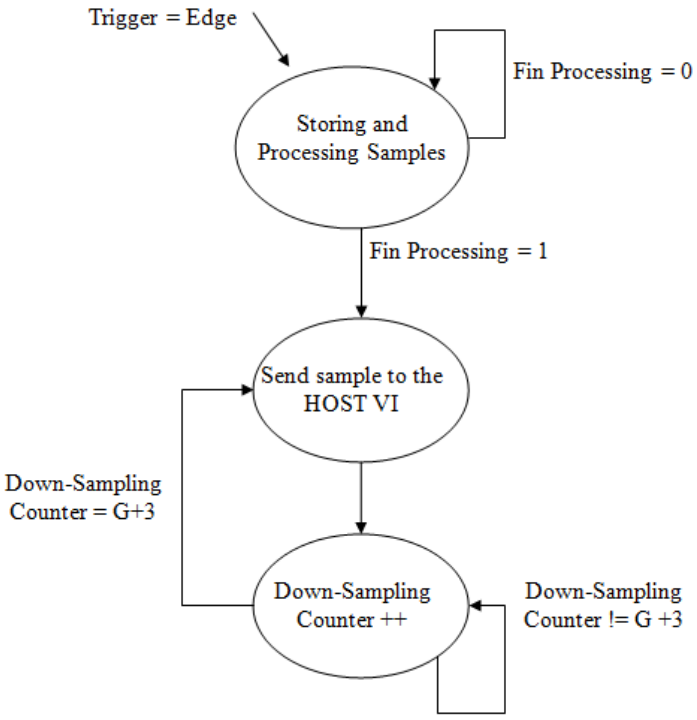


Figure 5.2i: Moore *FSM* of the Down-sampling algorithm in the Hilbert Filter.

5.3 HOST VI

The *HOST VI* is the program which it is running in the *PC*. A visual interface with several parameters must be configured when it is executed. Then the program starts to run. The samples are filtered out and down-sampled in the *FPGA*, and they are stored in a *TDMS* file to be read by *MATLAB* later.

Samples/Echo, Number of Echoes and Gate.

It is necessary to know which parameters must be given to the *HOST VI* when one or various velocity profiles want to be estimated.

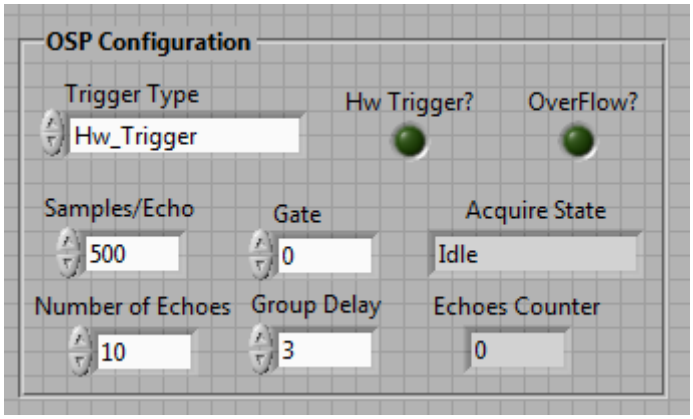


Figure 5.3a: Configuration window *HOST VI*.

Figure 5.3a shows the configuration window of the *HOST VI*, where the parameters to configure are:

- *Trigger Type*: It is possible to choose between a Software Trigger (*Sw_Trigger*) simulated by the *HOST VI* program or a Hardware Trigger (*Hw_Trigger*) given by an external source. *Sw_Trigger* is used to test the program and to do the simulations. However when the system is working normally the *Hw_Trigger* has to be configured.
- *Trigger Edge*: it is the edge of the external trigger which it is going to be used to acquire the samples when the *Hw_Trigger* is selected.

Rising/Falling edge are the two possible options that it is possible to configure.

- *Samples/Echo*: number of samples to acquire per echo signal received.
- *Number of Echoes*: According to the number of velocity profiles desired to measure, it is necessary to count a determined number of echoes.
- *Gate*: it is the down-sampling variable of the *FPGA VI*. It is relationship with number of samples of the Doppler signals.
- *Group delay*: It's the number of samples that you are going to throw away due to the time delay of the Hilbert Filter. For the current design of the Hilbert filter, this parameter always must take the value *3*.

Host VI Flowchart.

Figure 5.3b shows a flowchart of the *HOST VI* program. When the program is executed, the *FPGA* card (*PXIe-7965R*) is configured firstly and the *ADC* card is configured secondly (*NI5761*). The type of trigger is evaluated. When a software trigger (*Sw_Trigger*) is programmed, the program waits until the *Start* button is pressed, and then a test mode is going to work. One trigger is only programmed. Therefore, one echo is only processed. The signal processing starts to work in the *FPGA* and it stops when counter of samples (*Samples Counter*) reaches the limit configured in *Samples/Echo*. The test mode must only be used in the simulations.
However when the *Hw_Trigger* is selected, the normal working mode is going to work. An external trigger synchronized with the *PRF* (Pulse Repetition Frequency) informs when the acquisition starts. When a rising/falling edge is detected, the *Samples Counter* starts to work. It stops when it reaches the limit fixed by *Samples/Echo*. The acquisition will finish when *Echoes counter* reaches the limit configured in *Number of echoes*. Echoes counter is a counter which count basically the number of triggers produced. One echo signal is sampled when one *Hw_Trigger* is produced. For instance, if *300* echo signals are necessary to measure one velocity profile, *300* triggers are necessary to count. So the *Number of echoes* is configured as *300.* Of course, when the *Start* button is pressed, the acquisition starts again and the counters are reset.

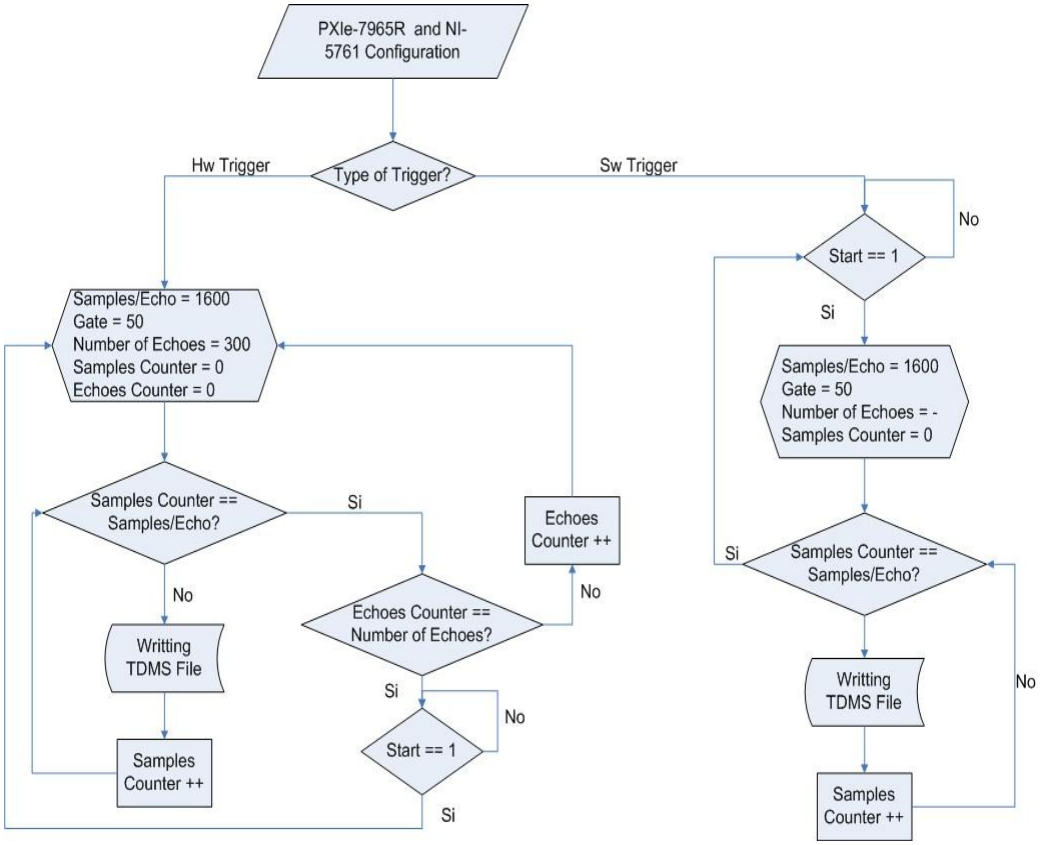


Figure 5.3b: Host VI Flowchart.

Velocity magnitude and velocity direction.

In the section 2.3 (UDAV), the *NEPP* parameter (Number of Emissions per profile) is another important parameter to take into account. If *NEEP=50,* It means that one pair of ultrasound transducer sends *50* burst signals and receives *50* echo signals. So, one echo signal is going to be received for one pair of ultrasound transducer in the first *PRF* (*6* switching steps). And the next echo signals is going to be received in the second *PRF* and so on until *50 PRF*. We have to loop *50* times the *6* switching steps to receive in total *50x6=300* echo signals for all the ultrasound transducers[13].

[13] Four ultrasound transducers are going to be enabled each switching step. Then, 50 echoes are going to be received in parallel by the 4 ultrasound transducers in one switching step.

Figure 5.3c shows the evolution of the echoes for the *channel 0* of the ADC. For instance, with *NEPP = 50*, *Samples/Echo = 1600* and *Gate = 50*, the number of echo signals received in *50* switching steps are *300* (*Number of echoes = 50 echoes x 6 switching steps*) to get one velocity profile. The number of samples acquired per echo are *1600*, but only one sample is going to be transferred to the host computer and we are going to throw away *50* (*Gate = 50*).
Note that, if we want to get *1000* velocity profiles, *Number of Echoes = 50x6x1000 = 300000 echoes*.

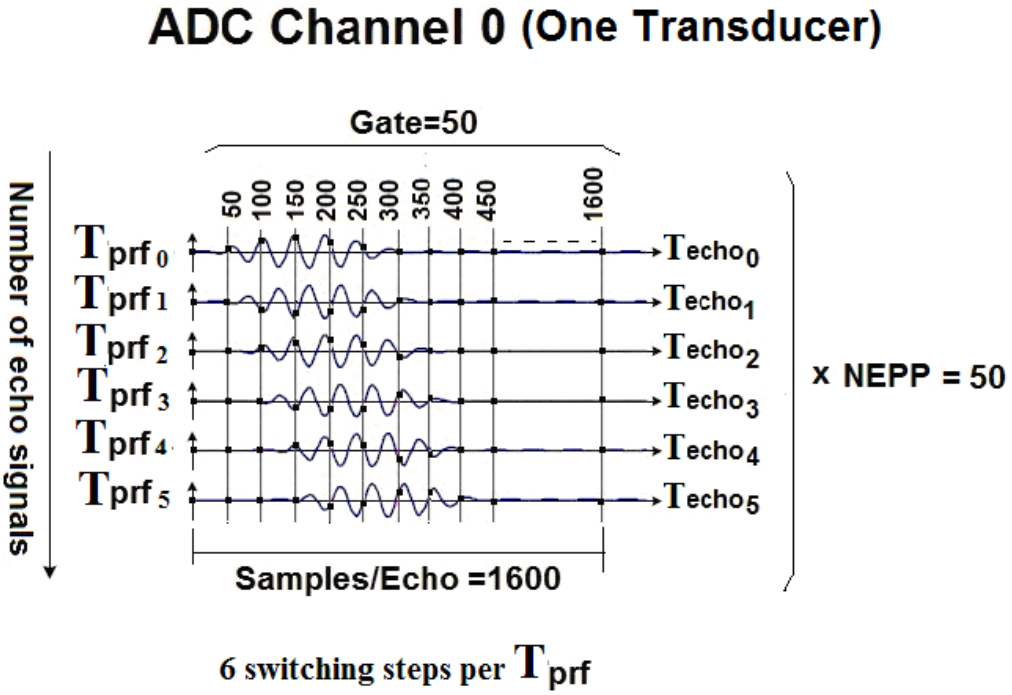


Figure 5.3c: Echo signals received by the *Channel 0* of the *ADC*. *NEEP=50, Samples/Echo=1600 and Gate=50*.

Figure 5.3d shows the 32 Doppler signals extracted from the echo signals down-sampled with a *Gate = 50*. In this example, the samples of *64* Doppler signals are stored in a *TDMS* file. The half of the Doppler signals will be the down-sampled output of the Matched filter and the rest, the down-sampled output of the Hilbert filter.

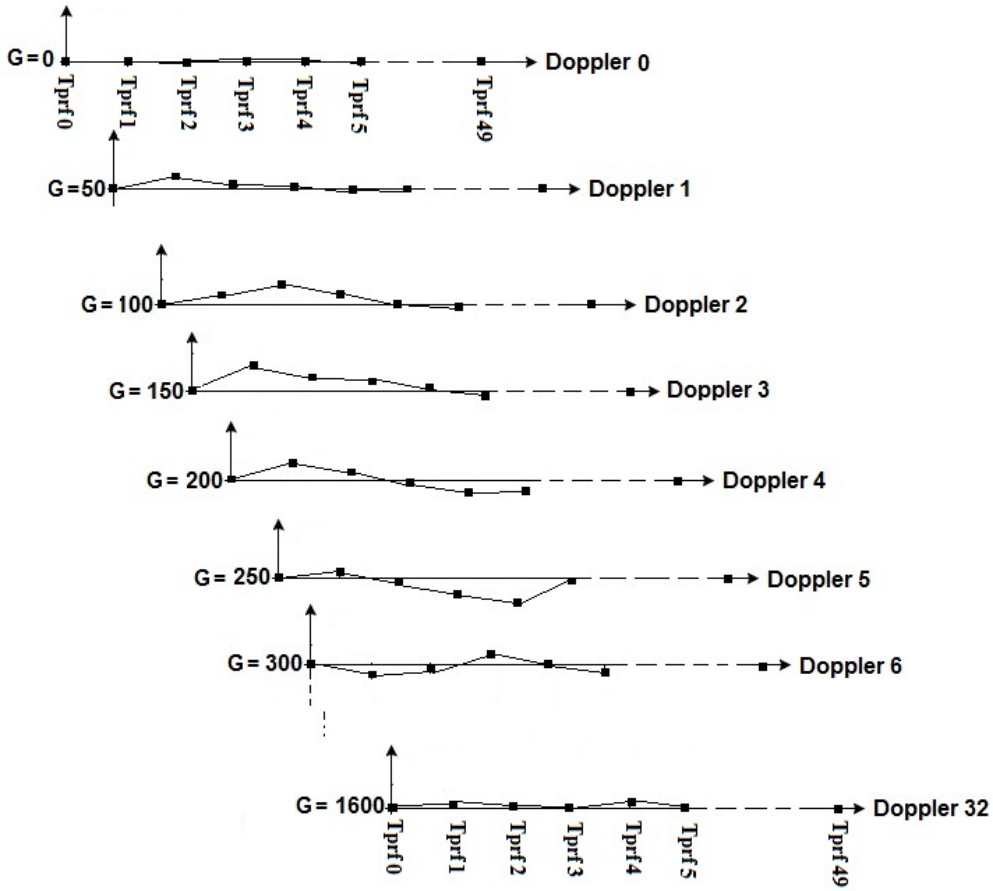


Figure 5.3d: Doppler signals extracted from the *300* echoes received.

The *TDMS* file is going to be read by a script in *MATLAB* with a correlation algorithm to measure the velocity and direction of the particle inside the opaque fluid. Remembering the equations of the section 2.1 (Ultrasound: Basic principles), it is possible to find out the distance (d_1) from the particle to the transducer called depth too (equation 5.3a) and the velocity (v) of the particle (equation 5.3b). In the equation 5.3b an approximation it is necessary to do. The angle θ must take the value *0*, because in one hand it is not possible to measure it and in other hand its value does not matter, giving the resulting equation 5.3c.

$$d_1 = \frac{c \cdot t_1}{2} \qquad [5.3a]$$

$$v = \frac{c \cdot \Delta f_1}{2 \cdot f_e \cdot \cos\theta} \qquad [5.3b] \quad \rightarrow \quad v = \frac{c \cdot \Delta f_1}{2 \cdot f_e} \qquad [5.3c]$$

Having a look to the picture of the figure 5.3e it is possible to identify all the parameters of the before equations.

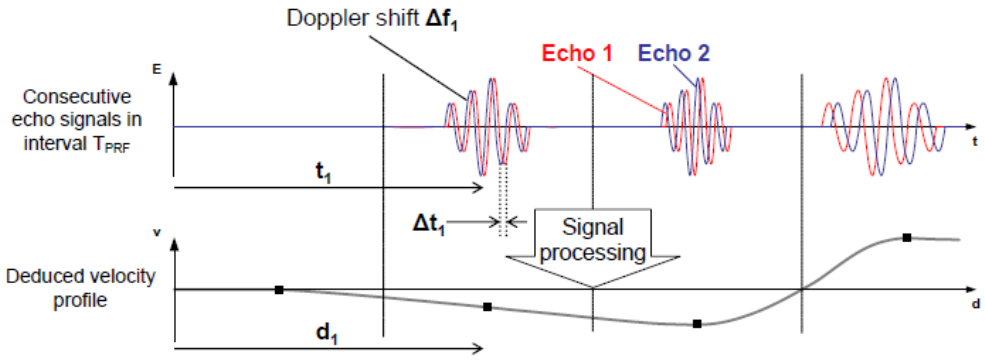


Figure 5.3e: Velocity measurement using two consecutive echo signals.

For the equation 5.3a the time delay between the burst signal 1 emitted and the echo signal 1 received (t_1) is related with the sampling period (T_s), the *Samples/Echo* and the *Gate* according to the equation 5.3d. Note that if we want to know the time delay between the burst signal 2 and the echo signal 2

(t_2), we have to apply the same equation 5.3d but we must also add the t_1. Because all the time delays are measured from $t = 0$.

$$t_1 = \frac{Samples/Echo}{Gate} \cdot T_s \qquad [5.3d]$$

For the equation 5.3c, the frequency of the burst signal (f_e) and the velocity of the sound (c) are known. The Doppler frequency (Δf_1) is measured applying the Fourier transform theory in the correlation algorithm. So all the parameters are known and the velocity can be estimated.

The direction of the velocity is obtained from the correlation algorithm too. It is essential to use the quadrature modulation to obtain the real and the imaginary part of the echo signals received. The sampled quadrature signals are obtained using a Hilbert filter sampling the echo signals received $\pi/2$ later. Figure 5.3e shows one peak of the echo signal received. Two samples are necessary to obtain the velocity direction. The sample 1 (*S1*) is obtained at the output of the Matched filter and the sample 2 (*S2*) is obtained at the output of the Hilbert Filter. In the first acquisition *S1* takes the maximum value and *S2* takes the value *0*. If the echo is moving toward the transducer ($-v$), *S1* takes a negative value and the *S2* takes a positive value. If the echo was moving against the transducer ($+v$) *S2* takes a negative value and *S1* takes a positive value.

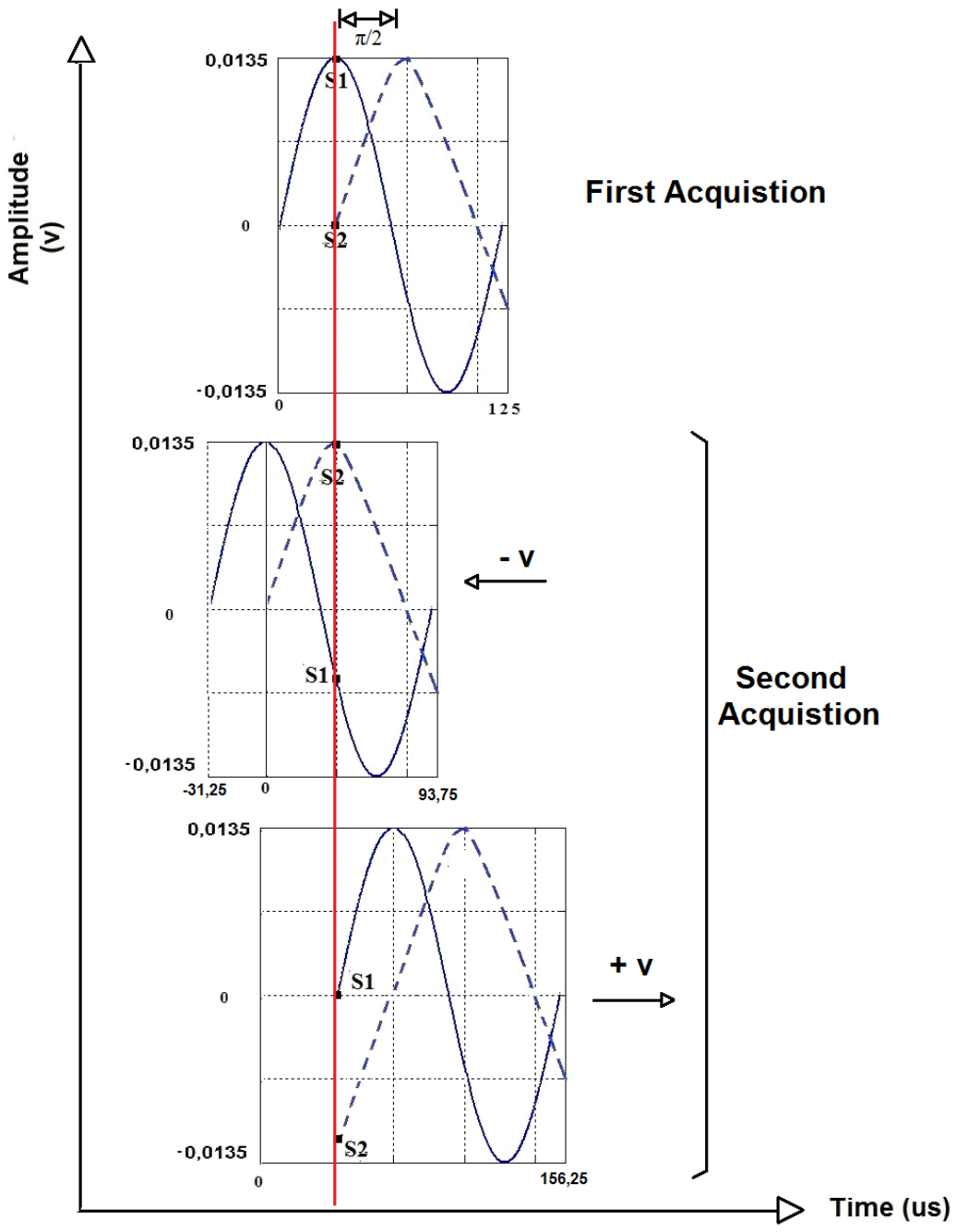


Figure 5.3e: Description of quadrature signals to find the velocity direction.

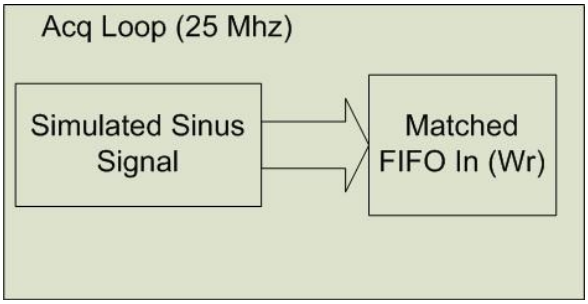
6. Test and results

In the *FPGA* system eight filters (two filters per channel of the *ADC*) have been implemented. The echo signals are acquired at the sample frequency of *25 Ms/s*. They are filtered out, down-sampled, transferred and stored into the host computer in a parallel working mode.

The properly working of the filters, the down-sampling algorithm and the data storing into a *TDMS* file are going to be tested and their results are going to be showed only for the channel *0*. Because for the rest of the channels, the results obtained are the same.

6.1 FPGA VI.

LabVIEW has the possibility to test the filters and the down-sampling algorithm before they are programmed into the *FPGA* device. But previously, it is necessary to make a *FPGA* test bench *VI* which provides the simulated input signals and it also shows the results of the tests in several graphs.
In the FPGA test bench VI the inputs of the Matched *FIFO* memories are going to be simulated by sinus signals generated by *LabVIEW* itself (Figure 6.1a).



Type of Signal	Frequency	Amplitude
Sinus	8 MHz	1200 mVpp

Figure 6.1a: Placing the simulated sinus signal in the acquire loop (Top) and Configuration parameters of the simulated sinus signal (Bottom).

Furthermore, two graphs must be placed in the transferring loop before sending the data to the HOST (Figure 6.1b).

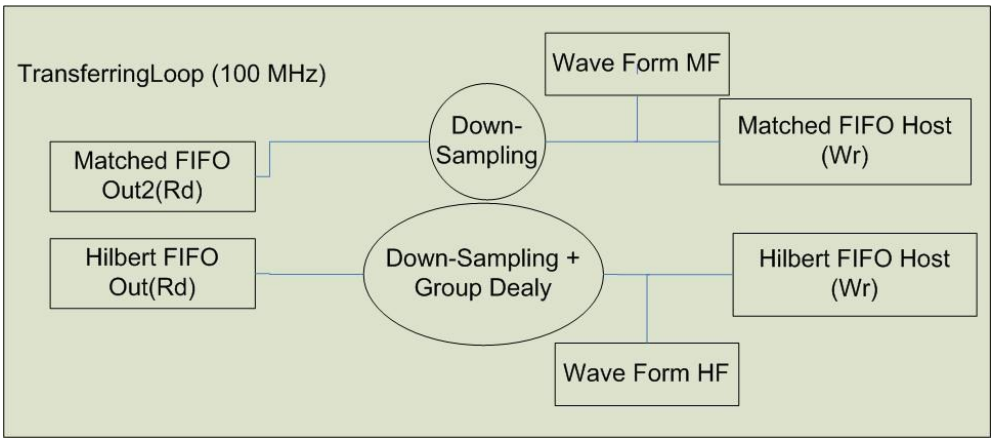


Figure 6.1b: Placing two data arrays in the transferring loop to test the results of the filters.

Testing the Matched filter.

The Matched filter is checked changing the frequency of the simulated sinus signal (for instance, *F = 1 MHz*). And it is also necessary to configure the parameters of the *FGPA* visual interface before starting the simulations:

- *Trigger Type = Sw_ Trigger*[14].
- *Trigger Edge = -.*
- *Samples/Echo = 200.*
- *Gate = 0.*
- *Group Delay = 3.*

[14] Note that the Sw_Trigger is the only option valid to test the signal processing in the FPGA VI.

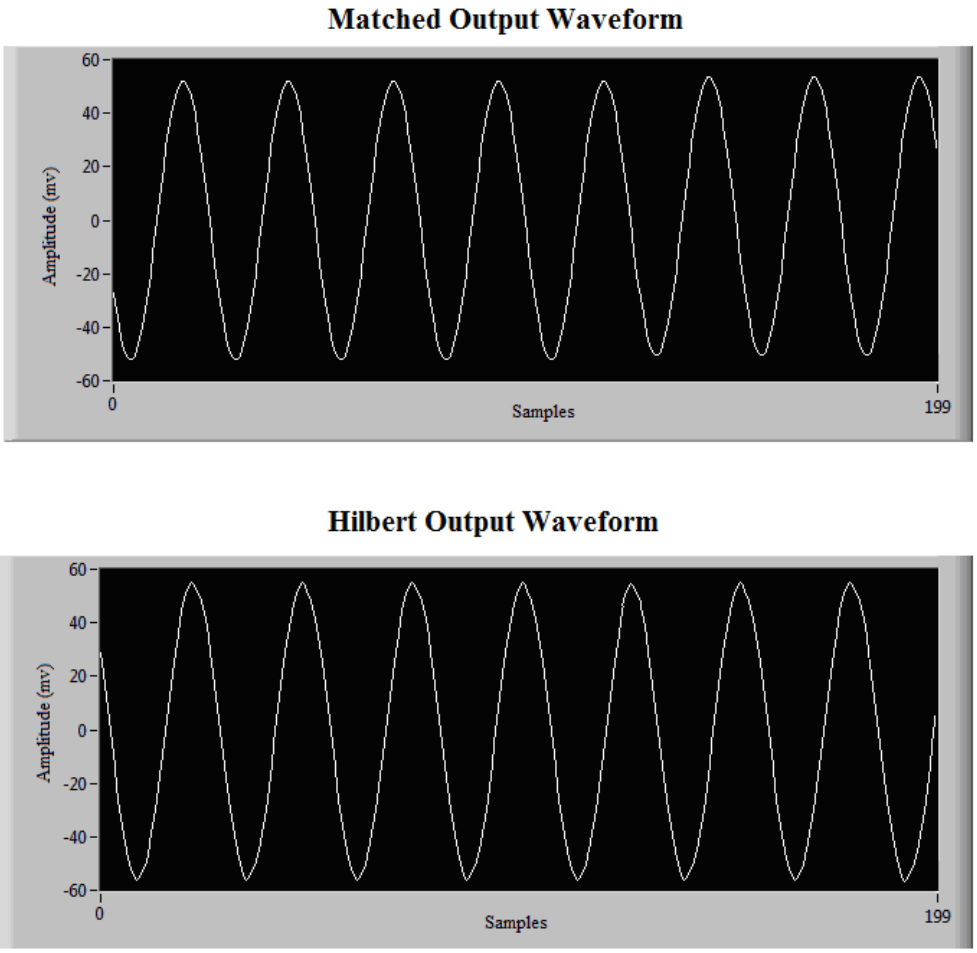


Figure 6.1c: *FPGA VI* Testing Matched Filter.

If the sampling frequency is $F_s = 25\ MHz$ (*25Ms/s*), Then, the number of samples acquired in $(1/F) = T = 1\ \mu s$ is $F_s \cdot T = 25$ samples. If *samples/echo = 200*, it means that *200* samples are going to be acquired. So the number of periods showed in the figure 6.1c is 8 (equation 6.1a):

$$N_T = \frac{Samples\ /\ Echo}{F_s \cdot T} = 200/25 = 8 \quad [6.1a]$$

Furthermore, attending to the graphs extracted from the *LabVIEW* Waveforms (figure 6.1c), it is possible to see that the amplitudes at the output of both filters are in the range of *+/- 60 mV* (almost *0 mV*). So the Matched filter works properly and it filters out the incoming signals when their frequencies are not *8 MHz*.

Testing the Hilbert filter and down-sampling algorithm.

Two tests are going to be presented. The test 1 is going to check the properly working of the Hilbert filter and the test 2 is going to check the properly working of both filters applying a down-sampling algorithm.

Test1: The simulated sinus signal is configured with *F = 8 MHz* and *1200 mVpp* of amplitude (Figure 6.1a). And the parameters to configure in the *FGPA* visual interface before starting the simulations are:

- *Trigger Type = Sw_ Trigger.*
- *Trigger Edge = -.*
- *Samples/Echo = 20.*
- *Gate = 0.*
- *Group Delay = 3.*

Figure 6.1d shows the wave forms at the output of the Matched and Hilbert filter when *Gate = 0* and only *20* samples are acquired. Now *T = 125 ns*. It means that 6 periods (or cycles) are shown in the figure 6.1d.

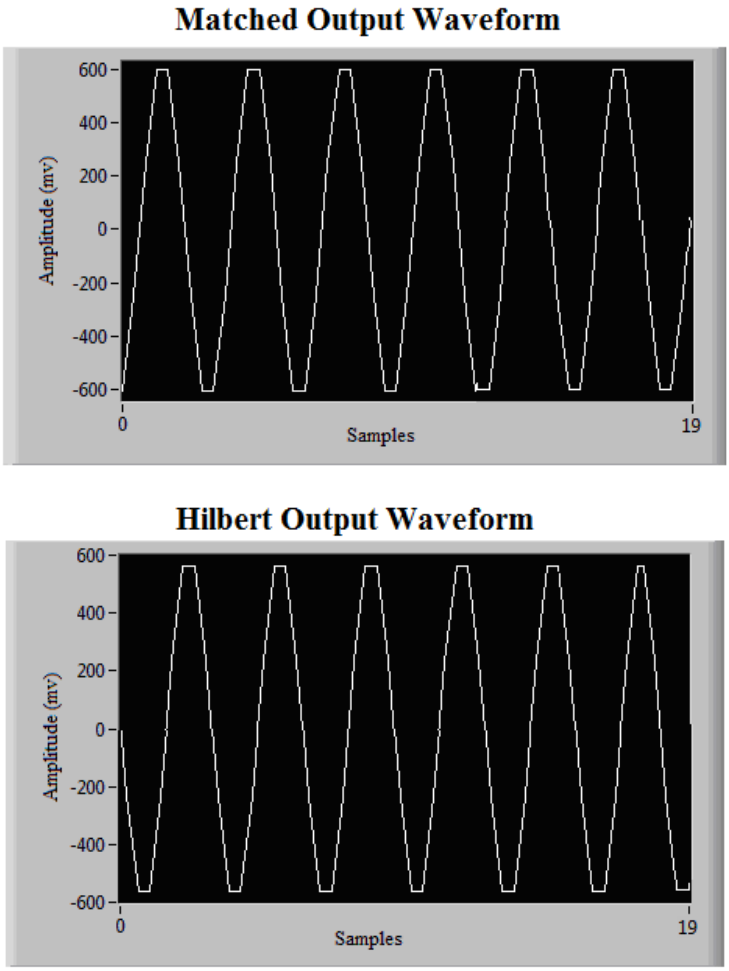


Figure 6.1d: *FPGA VI* Testing Hilbert Filter.

If the *Group Delay = 3,* it implies that *3* samples are thrown away at the beginning. And then, the output of the Hilbert filter is given. It is possible to check the *90º* of phase shift of the Hilbert filter comparing with the output of the Matched filter. For instance, for the sample *0*, the output of the matched filter takes the minimum value *(-600 mV*) and the output of the Hilbert filter takes the value *0 mV*.

Test 2: The simulated sinus signal is configured with *F = 8 MHz* and *1200 mVpp* of amplitude (Figure 6.1a). And the parameters to configure in the *FGPA* visual interface before starting the simulations are:

- *Trigger Type = Sw_ Trigger.*
- *Trigger Edge = -.*
- *Samples/Echo = 100.*
- *Gate = 10.*
- *Group Delay = 3.*

Attending to the configuration of the variables *Samples/Echo* and *Gate* is possible to know the total number of samples to send to the *HOST VI* o *PC* by the equation 6.1b.

$$TotalSamples = \frac{Echo / Samples}{Gate} = \frac{100}{10} = 10 samples \qquad [6.1b]$$

Then the number of cycles to show in the figure 6.e should be almost 3. Moreover, it is important to keep in mind that for the Matched filter the samples stored in the positions *0, 10, 20, 30, 40, 50, 60, 70, 80* and *90* of the FIFO are should be transferred and for the Hilbert filter the samples 3*, 13, 23, 33, 43, 53, 63, 73, 83* and *93* should be transferred to the PC (or HOST VI). Comparing the two graphics is possible to see that the phase shift of the Hilbert filter is working too, because for the sample number *0* the Matched filter takes the value *0 mV* and the Hilbert filter minimum value (*-600 mV*).

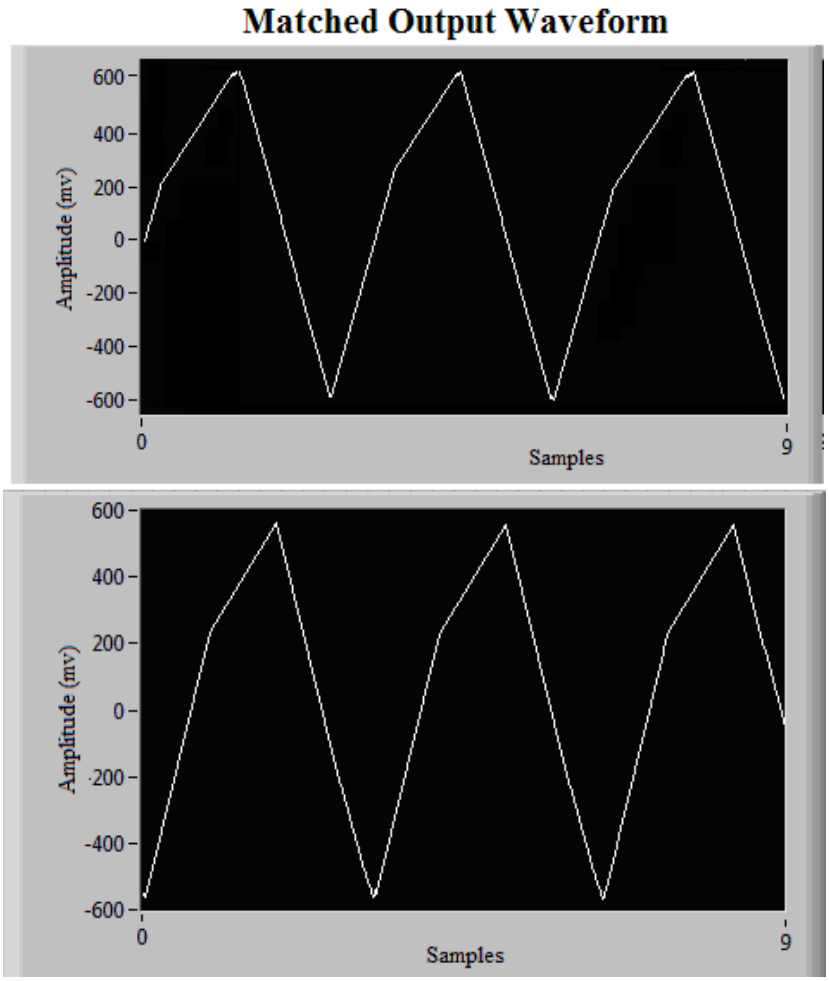


Figure 6.1e: *FPGA VI* Testing Hilbert Filter and the down-sampling algorithm.

6.2 HOST VI.

This section is going to test the ADC sampling frequency, the filters and the down-sampling algorithm programmed in the FPGA device using a real system with real signals. The results are going to be stored in a TDMS which is going to be read by MATLAB later.

6.2.1 Testing the ADC sampling frequency.

The hardware configuration used to test the ADC sampling frequency is shown in the figure 6.2.1a.

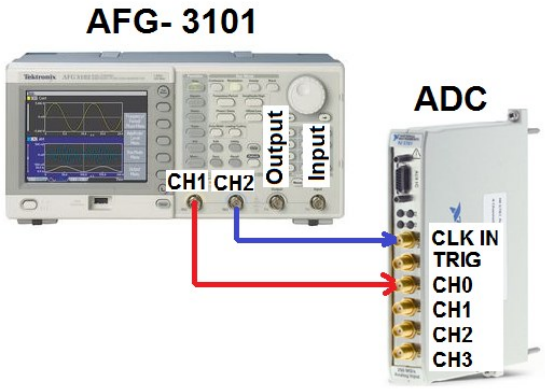


Figure 6.2.1a: Configuration hardware used to test the sampling frequency.

And the table of the figure 6.2.1b shows the simulation signals that they must be configured in the *AFG-3101* to do this test.

Function Generators	Quantity	Characteristics	Configuration
AFG3101	1	• Bandwidth: 100MHz • Max Sample Rate: 1GS/s	• CH1 Configuration: - Amplitude: 800 mVpp. - Offset: 0 V. - Frequency: 8MHz. - Function: Continuous sinus signal. • CH2 Configuration: - Amplitude: 2 Vpp. - Offset: 0 V. - F: 25 MHz. - Function: Continuous Square Signal.

Figure 6.2.1b: Signals to configure in the AFG-3101 to do the sampling test.

And it is also necessary to configure the parameters of the *HOST* visual interface (*HOST VI*) before starting the test:
- *Trigger Type = Sw_ Trigger.*
- *Trigger Edge = Rising Edge.*
- *Samples/Echo = 20.*
- *Gate = 0.*
- *Number of Echoes = 1.*
- *Group Delay = 0*

Note that the sampling frequency is going to be tested. If *Samples/Echo = 20*, it means that 20 samples are going to be acquired when one trigger is produced. So applying the equation 6.2.1a it is possible to know how many periods (N_T) of the signal acquired are going to re represented.

$$N_T = \frac{Samples\,/\,Echo(samples)}{Fs(MegaSamples\,/\,\sec)\cdot T(\sec)} = \frac{20}{25}\cdot 8 = 6.4T \qquad [6.2.1a]$$

The resulting acquired signal is shown in the figure 6.2.1c and it is possible to check that the sampling frequency is right because almost 6.4 periods are shown.

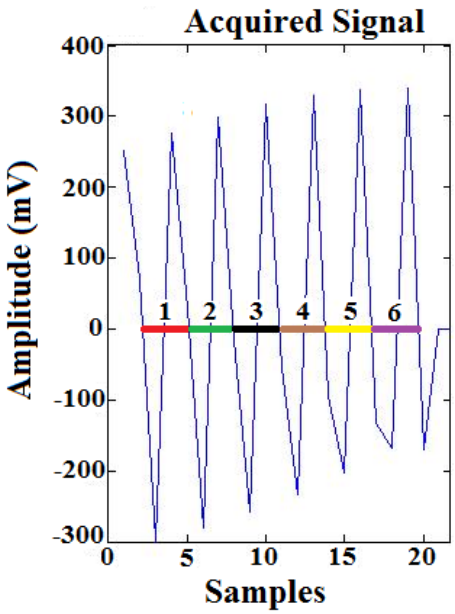


Figure 6.2.1c: Sampling frequency results.

6.2.2 Testing the Filters and the Down-Sampling algortihm.

The hardware used is presented below. Three *AFG 3000* Tektronix series function generators are going to provide the signals to test the system. Figure 6.2.2a shows the hardware connexions to do the tests. The *AFG3101* is used to generate the sampling signal (*25 MHz*) for the *ADC* (*CH1*) and the external trigger of the *ADC* (*CH2*) (which it is going to be the *PRF,* too). And the *AFG3021B* are used to simulate the *4* echo signals to drive to the *4* channels of the *ADC* card. All the channels are connected to the NI5761 ADC.

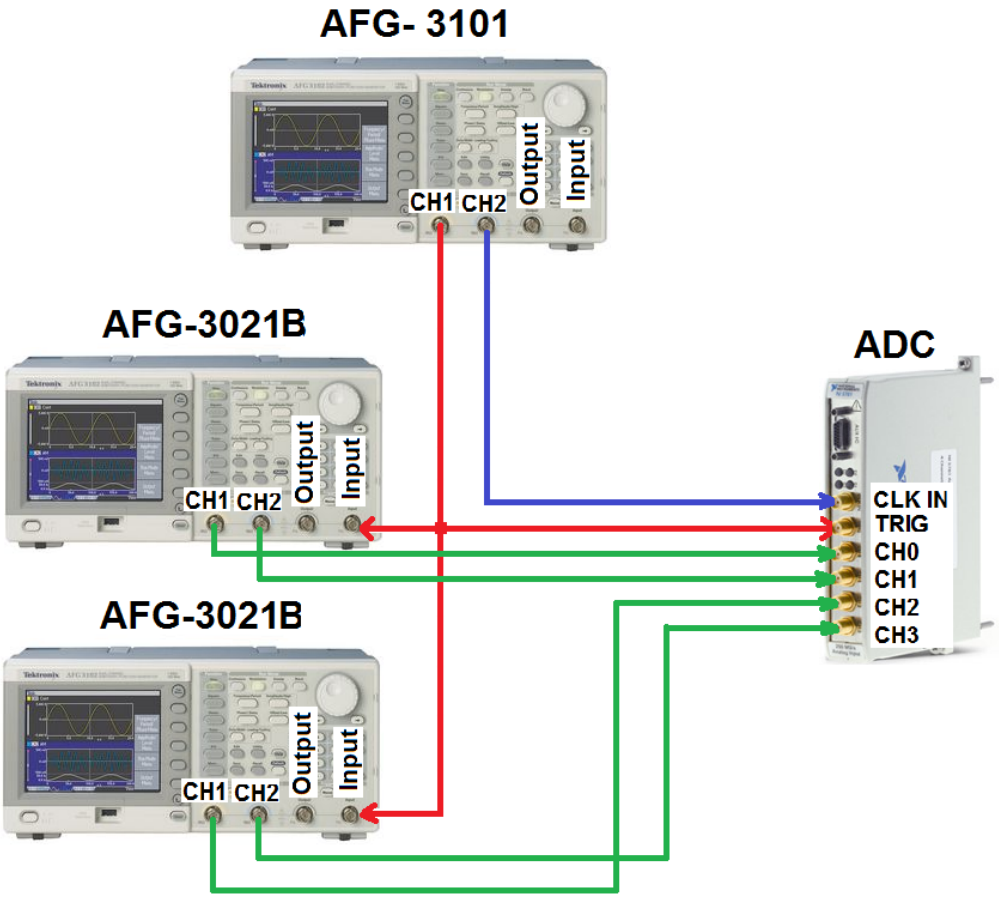


Figure 6.2.2a: Connexions between the function generators and the *ADC*.

The table of the figure 6.2.2b shows the configuration parameters of the function generators.

Function Generators	Quantity	Characteristics	Configuration
AFG3101	1	• Bandwidth: 100MHz • Max Sample Rate: 1GS/s	• CH1 Configuration: - Amplitude: 4 Vpp. - Offset: 0 V. - F: 18 KHz. - Function: Continuous Square Signal. • CH2 Configuration: - Amplitude: 2 Vpp. - Offset: 0 V. - Frequency: 25MHz. - Function: Continuous square signal.
AFG3021B	2	• Bandwidth: 25MHz • Max Sample Rate: 250MS/s	• CH1 and CH2 Configuration: - Amplitude: 800 mVpp. - Offset: 0 V. - Frequency: 8 MHz. - Time delay: 0 us. - Cycles: 8. - Source: External. - Slope = Positive - Function: Sinus Burst Signal..

Figure 6.2.2b: Tektronix configuration table to test the filters and the down-sampling algorithm.

After this previously configuration, the test starts and the results are stored in a *TDMS* file. This *TDMS* file is read by *MATLAB* and the results are plot using *MATLAB*, too.

Testing the Matched filter.

The Matched filter is checked changing the frequency of the simulated sinus signal (for instance, *F = 1 MHz*). And it is also necessary to configure the parameters of the HOST visual interface (HOST VI) before starting the simulations:

- *Trigger Type = Hw_ Trigger.*[15]
- *Trigger Edge = Rising Edge.*
- *Samples/Echo = 500.*
- *Gate = 0.*
- *Number of Echoes = 10.*
- *Group Delay = 3*

The *Number of Echoes* is *10* and *Samples/Echo = 500*. It means that *10* echoes composed by *500* samples each one are going to be processed (figure 6.2.2c). The amplitude at the output of the Matched filter is almost *0 mV*. (And therefore the amplitude at the output of the Hilbert filter is almost *0 mV* too).

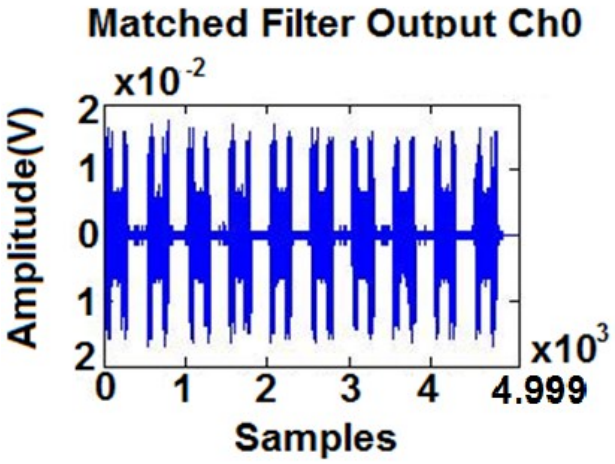


Figure 6.2.2c: *HOST VI* Testing Matched Filter.

[15] The Hw_Trigger is the option valid to test the signal processing in the HOST VI when then *Number of echoes > 1*.

Testing the Hilbert filter and down-sampling algorithm.

Two tests are going to be presented. The test 1 is going to check the properly working of the Hilbert filter and the test 2 is going to check the properly working of both filters applying a down-sampling algorithm.

Test1: The simulated echo signals are configured with *F = 8 MHz* and 8*00 mVpp* of amplitude (Figure 6.2b). And the parameters to configure in the *HOST* visual interface before starting the simulations are:

- *Trigger Type = Hw_ Trigger.*
- *Trigger Edge = Rising Edge.*
- *Samples/Echo = 500.*
- *Gate = 0.*
- *Number of Echoes = 10.*
- *Group Delay = 3.*

Figure 6.2.2d shows the waveforms of the ten echoes acquired and filtered out. On the top of the figure is possible to see the output of the Matched filter and on the bottom the output of the Hilbert filter. According to the configuration of the parameters, 10 echoes composed by 500 samples are represented.

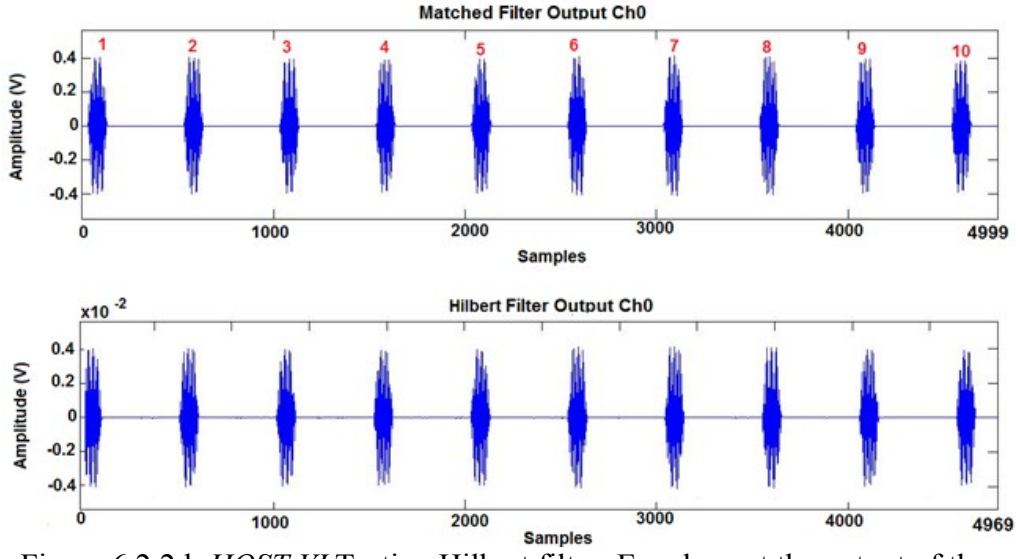


Figure 6.2.2d: *HOST VI* Testing Hilbert filter: Envelope at the output of the filters.

Doing a zoom in the figure 6.2.2d is possible to check that the Hilbert filter is working properly because its output is the Matched output shifted 90º (figure 6.2.2e). The red points show the phase shift between the filter outputs. When the sample number *0* of the Matched output takes the maximum value, the sample at the Hilbert output takes almost the value *0*. The envelope of the signals are not the same because the quantification errors produced by the data type conversion.

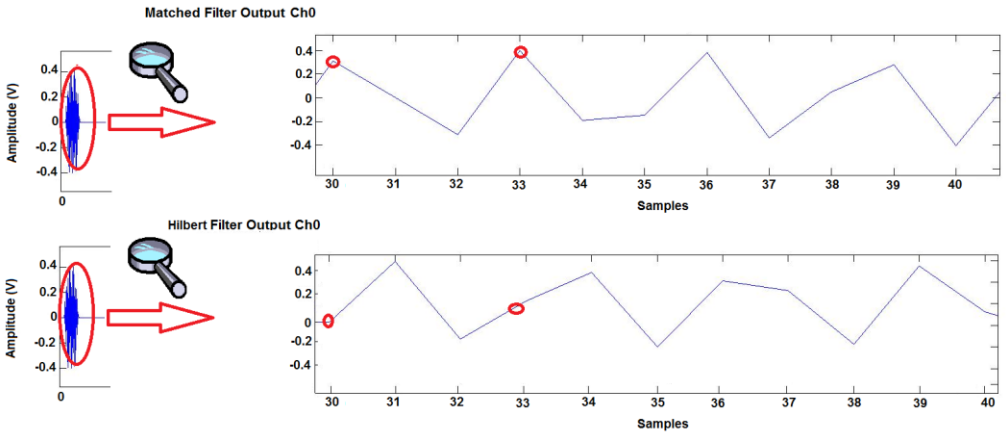


Figure 6.2.2e: *HOST VI* Testing Hilbert filter: Zoom at the output of the filters.

Test2: The simulated echo signals are configured with *F = 8 MHz* and 8*00 mVpp* of amplitude (Figure 6.2.2b). And the parameters to configure in the *HOST* visual interface before starting the simulations are:

- *Trigger Type = Hw_ Trigger.*
- *Trigger Edge = Rising Edge.*
- *Samples/Echo = 500.*
- *Gate = 100.*
- *Number of Echoes = 10.*
- *Group Delay = 3.*

In the figure 6.2.2f is possible to see that *10* echoes acquired and filtered out when a *Gate = 100* is applied. The original acquired signal is composed by *500* samples. But according to the *Gate* configured, only *5* samples are transferred. For the Matched filter the samples *0, 100, 200, 300 and 400* are

transferred and for the Hilbert filter are transferred the samples *3, 103, 203, 303 and 403*.

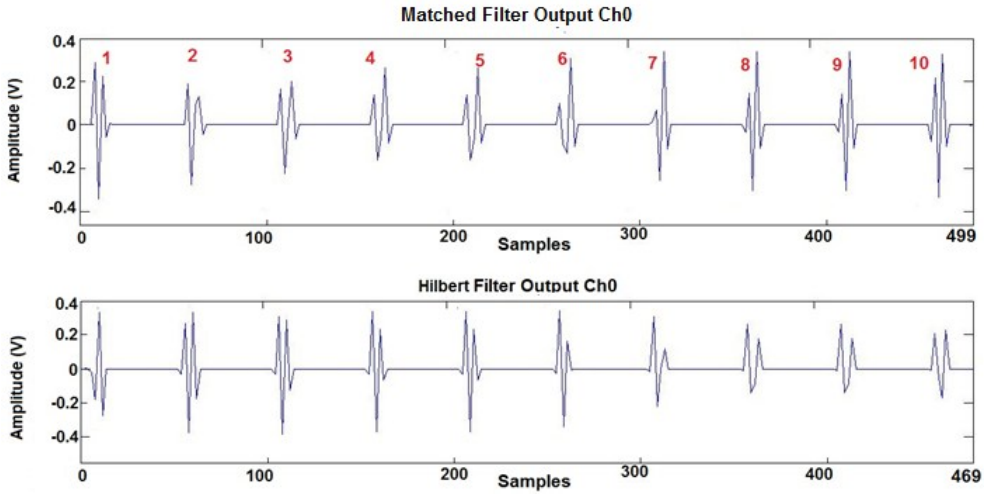


Figure 6.2.2f: *HOST VI* Testing Hilbert filter and the down-sampling algorithm: Envelope at the output of the filters.

Doing a zoom in the figure 6.2.2f, it is possible to check how the Hilbert filter is working properly when the down-sampling algorithm is applied. When the output of the Matched filter takes the value *0*, the output of the Hilbert filter takes almost the minimum value (figure 6.2.2g).

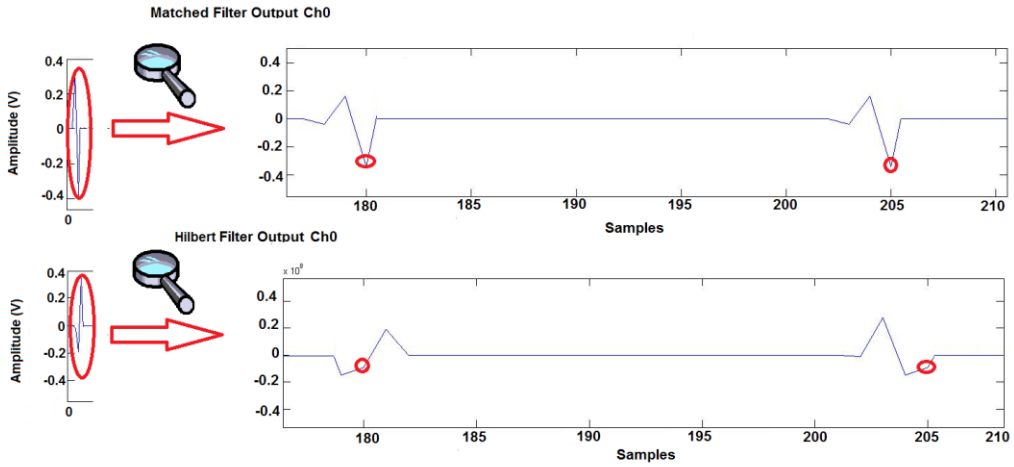


Figure 6.2.2g: *HOST VI* Testing Hilbert filter and the down-sampling algorithm: Zoom at the output of the filters.

7. Summary and Outlook

In this Master thesis, part of the signal processing has been implemented in a *FPGA* system manufactured by National Instruments.

The first task was to reduce the sampling frequency to *25 MHz* of the *ADC* using the experimental Low Speed *CLIP* provided by *LabVIEW* for the *NI 5761* card. After several tests this *CLIP* worked properly. Note that before this, the only way to reduce the sampling frequency of the *NI 5761* card was programming a firmware algorithm in the *FPGA* which throw away part of the samples acquired by the *ADC* to achieve the desired sampling frequency. And furthermore the minimum sampling frequency achieved with this method was *33.3 MHz*. Now it is possible to configure the *ADC* at the precise sampling frequency desired via hardware.

The next task was to implement the Matched and Hilbert digital filters designed by *MATLAB* in the *FPGA* system. But foremost, a *LabVIEW VI* had to be programmed to convert the coefficients to fixed point data type and to define the direct-form transposed *FIR* structure of these filters. Then the Matched filter and Hilbert filter was implemented in the *FPGA* with the down-sampling algorithm for the *channel 0* firstly. Then, it was tested with a *LabViEW* test bench *VI* and when it was checked the properly working, everything was extended for the four channels.

The final task was to program a visual interface (*VI*) to configure the parameters of the signal processing and read the data stored into the *PC* previously. *LabVIEW* was the software tool used to achieve this task. A *HOST VI* was programmed to receive the data results from the *FPGA* system and stored it into a *TDMS* file. An algorithm to synchronize the external trigger of the *ADC* with the T_{prf} of the echo signals had to be implemented in the *HOST VI*, too. And finally a script in *MATLAB* was programmed to read the data from the *TDMS* file and present the results.

All the main tasks of the Master thesis have been carried out successfully. Due to it, it is possible to reduce the data volume, allowing a faster signal processing of the echo signals received. However it has not been possible to

test in a real experiment with the *UDAV*. So in the future, it would be great to check this implementation in the *UDAV* with real measurements.

My Master thesis has been the second investigation project completed where a system based on a *FPGA* has been used to improve the signal processing of the *UDAV*. It solves some deficiencies of the other project but it is open to future works, too.

For instance, an important limitation in the *ADC* card (*NI 5761*) does not allow to measure *2d-2c* velocity profiles because it is necessary *8* channels and it has only 4 channels providing *2d-1c* velocity profiles. Therefore, it would be very recommendable to add another *ADC* card with more channels (at least 8 channels) if it would be available on the market.

The most laborious task has been the implementation of the filters in the *FPGA* system. Several troubles were found while this task was developed. And one of them was the amount of resources spent by the filters. At the moment the *74%* of the resources of the *FPGA* are used. If it was necessary to implement more part of the signal processing (like the correlation algorithm or more filters) it would not be possible. So in the future, if one *FGPA* system is going to be used, it would be very nice to think how it is possible to reduce the area spent especially by the Matched filter. Because the Matched filter has *74* coefficients and the Hilbert filter has only *6*. And the resources spent are higher if the number of coefficients is higher. So reducing the number of coefficients is the clue to get a good solution. For instance if the Matched filter is exchanged by a simple band-pass filter, the resources spent out are reduced considerably. But a band-pass filter is not so restrictive like the Matched Filter and several tests must be done to check if this filter could be valid.

Another interesting future work could be use only one program to do everything. Actually *MATLAB* is used to get the coefficients of the filters and to apply the correlation algorithm to measure the velocity profile. And *LabVIEW* is used to filter the samples and to apply the down-sampling algorithm. *LabVIEW* should be the only program to use in the future, because in this way, a lot of time would be saved and the process to measure the velocity profile would be more comfortable.

Furthermore another reason to use only *LabVIEW* is because the *MATLAB* script which reads the *TDMS* file can only run in *Windows XP*. And it is not possible to adapt it for the future operating system like *Windows 7* because the libraries, which they are provided by National Instruments, are only valid for old operating systems.

8 Appendix

8.1 Appendix A: Resolution theory

In pulsed wave Doppler measurement systems the axial resolution is expressed by:

$$\Delta z = \frac{Ncycles \cdot c}{2 \cdot fe} \quad [8.1a]$$

Where

c = the sound velocity,
fe = the emission frequency
Ncycles = the number of wave cycles of the ultrasound pulse

8.2 Appendix B: The Schwarz inequality

The inequality states this relationship:

$$\left| \int_{-\infty}^{+\infty} f1(x) f2(x) dx \right|^2 \leq \int_{-\infty}^{+\infty} |f1(x)|^2 df \int_{-\infty}^{+\infty} |f2(x)|^2 dx \quad [8.2a]$$

But this hold true only if and only if:

$$f1(x) = k \cdot f2^*(x) \quad [8.2b]$$

Where $f_2^*(x)$ is complex conjugate of the function $f_1(x)$.

8.3 Appendix C: MATLAB script Matched filter.

```
% Matched filter Design

% make sure to apply the sample rate that drives the FPGA ADCs
Fs = 25e6;
Fo = 8e6;

% fairly good results with N=24 periods
% (note that our real burst consists of only 8 periods. We increase
% the number until 24 to get a SMALLER passband width)
n=24;

t=1/Fs:1/Fs:(n*1/Fo)-(1/Fs); % base time
Coeff=0.027*sin(2*pi*Fo*t);
Coeff=fliplr(Coeff); %exchange coefficients of the matrix -> doc fliplr
```

Figure 8.3: *MATLAB* script of the Matched Filter.

8.4 Appendix D: MATLAB script Hilbert Filter.

```
% FIR least-squares Hilbert transformer filter designed using the FIRLS
function.

% All frequency values are normalized to 1.

N = 6; % Order
F = [0.3 0.7]; % Frequency Vector
A = [1 1]; % Amplitude Vector
W = 1;  % Weight Vector

coeff  = firls(N, F, A, W, 'hilbert'); % Linear-Phase FIR filter
```

Figure 8.4: *MATLAB* script of the Hilbert Filter.

8.5 Appendix E: SPI Protocol

SPI (Serial Peripheral Interface) operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines.
The *SPI* bus specifies four logic signals:

- *SCLK*: Serial Clock (output from master);
- *MOSI*; *SIMO*: Master Output, Slave Input (output from master);
- *MISO*; *SOMI*: Master Input, Slave Output (output from slave);
- *SS*: Slave Select (active low, output from master)

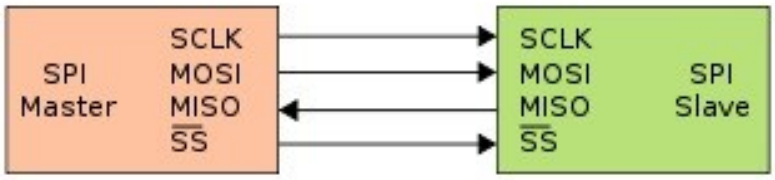


Figure 8.5a: *SPI* signals.

To begin a communication, the master first configures the clock. Then it pulls the chip select low for the desired chip and it must wait for at least that period of time before starting to issue clock cycles.

During each *SPI* clock cycle, a full duplex data transmission occurs: the master sends a bit on the *MOSI* line; the slave reads it from that same line. The slave sends a bit on the *MISO* line; the master reads it from that same line.

Transmissions normally involve two shift registers connected in a ring where the master and slave have exchanged register values (Figure 8.5b)

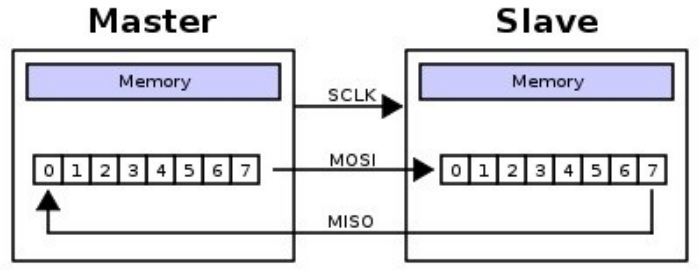


Figure 8.5b: SPI Transfer Data.

8.6 Appendix F: Phase Jitter, Phase Noise and Time Jitter

- Phase Jitter: An ideal sine wave can be thought of as having a continuous and even progression of phase with time from 0 degrees to *360* degrees for each cycle. Actual signals, however, display a certain amount of variation from ideal phase progression over time. This phenomenon is called phase jitter. Although many causes can contribute to phase jitter, one major cause is random noise, which is characterized statistically as being Gaussian (normal) in distribution.

- Phase noise: A phase jitter leads to a spreading out of the energy of the sine wave in the frequency domain, producing a continuous power spectrum. This power spectrum is usually reported as a series of (in *dBc/Hz*) at a given offset in frequency from the sine wave (carrier). The value is a ratio (expressed in *dB*) of the power contained within a *1 Hz* bandwidth with respect to the power at the carrier frequency. Then It is meaningful integrate the total power contained within some interval of offset frequencies (for example, *10 kHz* to *10 MHz*). This is called the integrated phase noise over that frequency offset interval.

- Time Jitter: Phase noise is a frequency domain phenomenon. In the time domain, the same effect is exhibited as time jitter. For example, in a square wave, the time jitter is seen as a displacement of the edges from their ideal (regular) times of occurrence.

9. References

[1] C.N.Chua. The Principles of Medical Ultrasound. January 14, 2009

[2] Echo-Web: The Online Learning Center for Echocardiographers. Sample Course Echo 202.

[3] Asbjørn Støylen. Basic ultrasound, echocardiography and Doppler for clinicians. November 2010

[4] http://www.signal-processing.com. DOP2000 Model 2125/2032 User's manual. Software: 4.02 Rev: 1. Signal Processing S.A. Pages: 162.

[5] Sven Franke, Lars Büttner, Jürgen Czarske, Dirk Räbiger, Sven Eckert. Ultrasound Doppler system for two-dimensional flow mapping in liquid metals. 2 May 2010. Pages: 8.

[6] S. Franke; H. Lieske; A. Fischer; L. Büttner; J. Czarske; D. Räbiger; S. Eckert. Two-dimensional ultrasound Doppler velocimeter for flow mapping of unsteady liquid metal flows. Pages: 17.

[7] John G. Proakis and Dimitris G. Manolakis. Digital Signal Processing. Principles, Algorithms and Applications. Edition 1996. Pages: 1033.

[8] Jorgen Arendt Jensen. Estimation of blood velocities using ultrasound. Cambridge University Press, 1996. Pages: 336.

[9] Steven W. Smith. The Scientist & Engineer's Guide to Digital Signal Processing. Digital Signal Processing Chapter. Ph.D. 1997

[10] http://en.wikipedia.org/wiki/Finite_impulse_response. Finite Impulse Response. Wikipedia. May 2010.

[11] Adrian Venezuela. FIR Filters. Version: 1.5. August 21, 2005. Pages: 6.

[12] Julious O. Smith III. https://ccrma.stanford.edu/~jos/filters/. Introduction to the Digital Filter. August 19, 2011.

[13] Nguyen Huu Phuong. FIR Filter Design: The Window design method. Jul 8, 2009. Pages: 52.

[14] Analog Devices. Fixed Point vs Floating Point Digital Signal Processing. September 5, 2009. Pages: 2.

[15] www.ni.com. National Instruments FlexRIO FPGA Modules. NI PXI-795xR, NI PXIe-796xR. Ni.com. May 2010.

[16] www.ni.com. National Instruments 5761R User Guide and Specifications. Ni.com. May 2010.

[17] www.ni.com. National Instruments 5761. Adapter Module of 14 Bits, 250 MS/s for NI FlexRIO. Ni.com. May 2010.

[18] ADS62P49 Dual Channel 14-/12-Bit, 250-/210-MSPS ADC With DDR LVDS and Parallel CMOS Outputs. Texas Instruments. January 2011. Pages: 84.

[19] AD9512 1.2 GHz Clock Distribution IC, 1.6 GHz Inputs, Dividers, Delay Adjust, Five Outputs. Analog Devices. Rev A. February 2005. Pages: 48.

[20] www.ni.com. National Instruments LabVIEW 2010 SP 1. Ni.com. January 2010

[21] www.ni.com National Instrument Technical Support. Ni.com. May 2011.

[22] http:\\zone.ni.com. FPGA Design, Development and Programming Tutorial. National Instruments Developer Zone. January 12, 2011.

[23] http:\\zone.ni.com.Developing High Speed Continuous Buffered Data Acquisition Applications with CompactRIO. National Instruments Developer Zone. December 20, 2006.

[24] http:\\zone.ni.com. Designing Digital Filters with NI LabVIEW and the Digital Filter Design Toolkit. National Instruments Developer Zone. March 17, 2008.

[25] http:\\zone.ni.com. Importing External IP Into LabVIEW FPGA. National Instruments Developer Zone. July 29, 2011.

[26] http://www.ni.com/swf/presentation/us/labview/lvfpga/default.htm. LabVIEW On-line Tutorial. Ni.com. December 2009.

[27]] http:\\zone.ni.com. Using Fixed-Point Data Types with Integer-Based IP in LabVIEW FPGA 8.5.x. National Instruments Knowledge Base. March 17, 2008.

[28]] http:\\zone.ni.com. Creating Connector Terminals for SubVIs in LabVIEW. National Instruments Knowledge Base. October 6, 2005.

[29]] http:\\zone.ni.com. Designing Ultra-High Throughput FIR Filters with no Multiplier on NI FPGA Platforms. National Instruments Developer Zone. August 4, 2009.

[30] www.ni.com. Aliasing. LabVIEW 8.2 Help. August 2006.

[31] www.ni.com Error, “Compilation failed due to a Xilinx error,” When Using NI FlexRIO. Knowledge Base. December 16, 2010.