UPPSALA
UNIVERSITET

TVE 13 045
Examensarbete 15 hp
Juni 2013

Simulation model of an ultrasonic sensor used in non-destructive testing

Joachim Björsell
Viktor Ferm Lithén
Joel Törmä

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Simulation model of an ultrasonic sensor used in non-destructive testing

Joachim Björsell, Viktor Ferm Lithén & Joel Törmä

In the steel industry the welding and structure of steel pipes are tested to make sure that the quality of the pipes are good enough. Today this is often done by taking some samples, cutting them and examining them manually. By testing the steel pipes with ultrasonic testing instruments all the samples can be examined without any destructing being done which can decrease the cost and time spent examining samples.

This project was about creating a simulation model in MATLAB/Simulink and Xilinx ISE for the signal processing part in a larger non-destructive ultrasonic testing project. The model would be able to generate pulses and make relevant processing of the transmitted and received signal.

The completed model can be configured as the user wishes for different simulations. It can handle the frequencies in the interval that are being used in non-destructive ultrasonic testing and do relevant processing of the signal. The Simulink model can be improved, for example by adding some noise to the different MATLAB function blocks. The simulation is done by saving and loading files between MATLAB/Simulink and Xilinx ISE but with some licenses and software it would be possible to do a co-simulation.

Handledare: Lars Johansson
Ämnesgranskare: Daniel Carlsson
Examinator: Martin Sjödin
ISSN: 1401-5757, TVE 13 045

Populärvetenskaplig sammanfattning

I stålindustrin undersöks svetsfogar och struktur på stålrör som produceras för att säkerställa att de har nog bra kvalitet för dess ändamål. Idag gör man det ofta manuellt genom att kapa och mäta på stickprov. Med ultraljud kan man undersöka alla stålrör utan att de förstörs.

Målet med det här projektet var att skapa en simuleringsmodell i MATLAB/Simulink och Xilinx ISE för signalbehandlingsdelen till ett instrument för ultraljudstester på stålrör. Kraven på modellen var att den ska kunna hantera de frekvenser som används i ultraljudstester och göra relevant behandling av de signaler som skickas in i modellen.

Den färdiga modellen kan bli justerad för att passa de simuleringar som ska köras. Den klarar av de frekvenser som används i ultraljudstester och gör relevant behandling av de signaler som skickas in i modellen.

Contents

1. INTRODUCTION

In the steel industry the welding and structure of steel pipes are tested to make sure that the quality of the pipes are good enough. Today this is often done by taking some samples, cutting them and examining them manually. This way, only a few samples can be checked and those samples are destroyed. Another way of examining the quality of the welding and the structure is by using a non-destructive ultrasonic testing method. The test is done by analysing the echo from ultrasound that is reflected from the examined sample. From the ultrasonic testing data is collected and with high speed processing it is possible to create real-time 3D images of the sample. The 3D-imaging is a desired testing method in the steel industry since all the steel pipes can be examined with high speed and precision without destroying any samples.

1.1 Background

Ultrasonic testing is done by sending acoustic pulses towards the examined material through a medium which will prevent total reflection for the sound waves when they encounter the material. Some part of the pulse will transmit through the material while the rest will be reflected. By measuring the time of arrival (TOA) for each returned pulse (also referred to as TOA) and knowing the propagation speed of the pulse the thickness of the material can be calculated. The way each TOA travels and the resulting graph of these is shown in Figure 1.1[2].

The time measurements for each TOA is done with digital circuits, often with a field-programmable gate array (FPGA). The controlling and analysing of the system is done digitally and the signal processing is done by analogue circuits. There are great requirements from the analogue circuits for a system which is able to generate a 3D-image in real time and with great accuracy. A model of a system like this can be seen in Figure 1.2. During development of ultrasonic systems, simulation models of the circuits are vital since the time consumption and cost for testing ideas with real components would be unsustainable. In this thesis a model like this was built. It contains all essential analogue parts, digital pulse generation and digital signal analysing.

All the Matlab and VHDL code and functions created during the project are gathered in Appendix B.

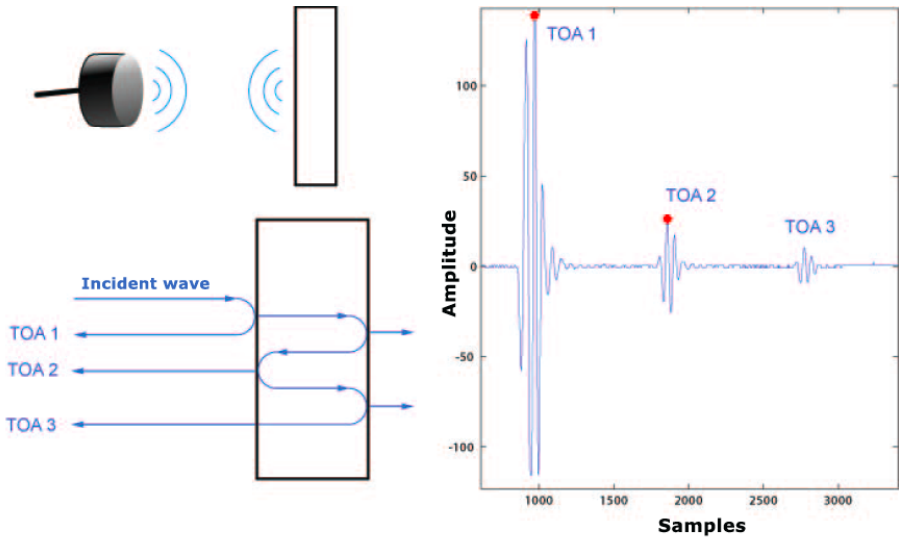

Figure 1.1: A visual model of how the pulses echo from the material and a graph of the pulse response from the echo when examining sample with ultrasound[2]

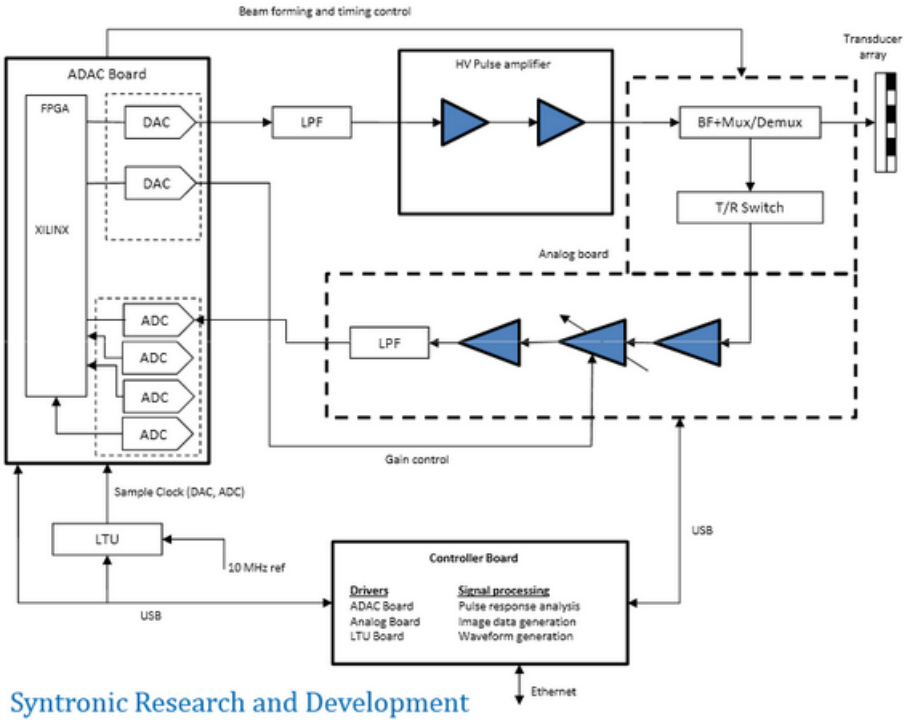

Figure 1.2: An overview model of the complete ultrasonic system planed to be able to generate a 3D-model in real time

1.2 Thesis

The aim of this thesis is to generate a digital pulse train that can be used as input in an analogue simulation model and then analyse the output from the model digitally. The analogue model is going to be created from scratch and function as a behaviour model and not include, for example, all noise additions and nonlinearities. Doing so will limit the use of the model but make the model much simpler to create and possible to complete in the short period of time that is available for working on the project. It will also leave more of the design and structure for us to decide.

The system is going to be configured to simulate an examination of a steel pipe with water as transport medium and be adjustable so it can be used for other similar purposes. Because of the demands on the instrument the model should be constructed so it can handle high frequencies, up to at least 50MHz.

The project will consist of two main parts. One is the simulation model of the analogue part of ultrasonic testing done in Simulink. The other is generation of digital components that generates pulses, receives the pulse response and analyses it which is done in Xilinx ISE.

The goal is that the simulation model can be used to try different models when constructing an instrument which will be able to, in real-time, generate a 3d-image of the examined object. The requirements of the instrument is to have a precision of 5 μm and then make PASS/FAIL-decisions based on the measurements.

1.3 Development Tools

The main simulation is done in Simulink, a simulation tool in MATLAB. All the code is generated through MATLAB and the possibilities to make your own function using MATLAB code is almost infinite. During this project MATLAB version R2013a student edition has been used.

The pulse generation and signal processing in the FPGA is done in Xilinx ISE. The language used in Xilinx ISE is very high speed integrated circuits hardware description Language (VHDL) which is an industry-standard language used for modelling and to synthesise digital hardware. VHDL is used for Xilinx programmable FPGAs to design signal processing FPGA circuit. With Xilinx ISE it is possible to integrate many functions in the circuit as digital parts and simulate the pulse generation and signal processing with behaviour similar to that of the physical system.

1.4 Theory

1.4.1 Ultrasound

Ultrasound is an oscillating sound pressure wave with higher frequency than what humans are able to hear. Ultrasonic devices are constructed to use ultrasound for many different types of measurements. Some examples are detecting objects, measuring distances, detecting flaws in materials, called non-destructive testing, and medical imaging. In non-destructive testing by using an ultrasonic system ultrasound waves are produced by piezoelectric transducers to examine the structure of samples and detecting flaws in them without any destruction being made. One type of ultrasonic testing is pulse-echo mode. In these systems the piezoelectric transducers convert electricity to sound waves and the reversed conversion to both transmit and receive the sound waves. The other type will instead of receiving the echo analyse the transmission of the signal.

When a sound wave travels from one material to another the wave will be totally reflected or partially reflected and partially transmitted depending on the attributes of the materials. If the sample is put in a medium that leads to partial reflection of the sound waves it is possible to determine the thickness and possible imperfections within the material from the bounces. That is why doctors use a gel between the skin and the apparatus when performing an ultrasonography on pregnant women. The frequencies used in the non-destructive testing are most commonly in the range from 0.1MHz to 20MHz, but even higher frequencies are being tested. Higher frequencies leads to better resolution but a lower image depth since the waves with shorter wavelength reflects and scatters from small anomalies and also decreases faster in amplitude when travelling through materials.[1]

1.4.2 Reflection

To calculate the energy loss of an ultrasonic wave when travelling through a material the relation

$$A_{out} = Ae^{-\alpha d}\ [\mathrm{V}] \tag{1.1}$$

is used, where α is the materials absorption coefficient, A is the amplitude of the signal and d is the thickness of the material[8]. The absorption coefficient for water[9],

$$\alpha_{water} = 0.2 * (10^{-6} f)^2\ [\mathrm{dB/m}], \tag{1.2}$$

and the reflection constant, the amount of the signal that is reflected when travelling between mediums and the rest is transmitted, [7]

$$R = (\frac{Z_1 - Z_2}{Z_1 + Z_2})^2 \, [-], \tag{1.3}$$

where Z_1 and Z_2 is the acoustic impedance of the materials, are used to calculate the the loss of energy in the signal while travelling through the material and medium. The acoustic impedance is calculated by using the relation [7],

$$Z = \rho c \, [\text{kg/ms}], \tag{1.4}$$

where ρ is the density of the material and c is the propagation speed of the wave in the material. When the wave is reflected or transmitted some noise will be created.

1.4.3 Two's complement

Two's complement is a mathematical operation to create signed numbers from binary code. The first number in two's complement represents the highest absolute value of the negative numbers. For an example the number four represented in ordinary binary code is 100, the same code represented with two's complement is -4. The following bits after the first bit represent the number added to the highest absolute value of the negative numbers, e.g 110 represented with two's complement is $-4+2=-2$. To represent the positive value four with two's complement an extra bit is needed and the number four is written 0100. Therefore two's complement can only represent numbers between $-(2^{N-1})$ to $(2^{N-1}-1)$ where N is the number of bits in the code string.

1.4.4 Digital-to-analogue-Converter

The first component of the analogue part in a ultrasonic system is the digital to analogue converter (DAC). The purpose of the DAC is to work as a link between the digital and the analogue world. When the digital signal is generated in the FPGA, the DAC will receive a signal composed of discrete sampled values in binary code divided into bits. The DAC is converting the signal to a continuous, analogue signal which can be processed and used as input in the transducer.

One way to simulate a DAC is to build a so called R-2R ladder network[3]. A R-2R ladder consists of a series of bits arranged from the most significant bit (MSB) to the least significant bit (LSB) where the value of the bit is either one or zero depending on the input signal. The bits are afterwards amplified with a gain of $1/(2^n)$ where n is the bit number, n=1 for the MSB and n=total

number of bits for the LSB, summed together and multiplied with a reference signal. An example of how the bits are gained when using 4 bits can be seen in the Table 1.1.

Table 1.1: Amplification of bits

Bits	Amplification
1 (MSB)	1/2
2	1/4
3	1/8
4 (LSB)	1/16

The DAC will have limitations on how high or low the amplitude of the signal can be. The DAC will have a preset voltage interval that determines the maximum and minimum value of the signal that the DAC can handle.

The DAC is sampled using an internal digital clock. The digital clock will not be exact which results small distortion in the output signal, the distortion is called jitter.

To prevent the signal from becoming completely cut of at certain frequencies when the signal is filtered after the DAC some white noise is added to the signal after the DAC.

1.4.5 Low Pass Filter

After the DAC the low pass filter (LPF) is located. The main purpose of the LPF is to filter out the frequencies higher than the desired signal to prevent aliasing in the reflection. To achieve this the LPF is placed after the white noise is added and before the reflection. The desired effect can be achieved with a low order filter. This makes the filter less complicated and cheaper. It also creates a smooth signal as close to the original as possible. To achieve this the passband edge is set close to the centre frequency of the original signal.

1.4.6 Pre-Amplifier

Because of the low efficiency of the transducer and the attenuation of the sound waves a high-voltage pulse is needed. Therefore the pre-amplifier is placed before the transducer that transforms the signal to sound waves. The amplitude restrictions from the DAC demands extra amplification to achieve the high-voltage pulse. The amplifier is one of the primary sources of noise in the received signal[4].

1.4.7 Transducer

A transducer is a component which transforms one energy form to another. The piezoelectric transducer used in ultrasonic systems that uses pulse-echo mode transforms the electrical pulse to ultrasonic sound waves, receives the echo and transforms it back to an electrical signal[5]. This results in that both the transmitting and the receiving signal passes through this component. The efficiency in the transformation is low and therefor needs a high-voltage input.

1.4.8 Transmitter/Receiver Switch

The transmitter/receiver switch (TR switch) is located just before the transducer. Since both the transmitting and the receiving signal will pass through the transducer a component is needed to direct these two signals so the transmitting signal arrives at the transducer and that the received signal can be analysed without any significant distortion from the transmitting signal. The switch closes the connection to the analyse part when there is a transmitting pulse coming through. The rest of the time it is open so the received pulse response can go through the system and be analysed[10]. Because of the high voltage of the transmitted compared to the received pulse response the components handling the received signal would be blinded if the transmitting signal would reach these components, meaning the pulse response can not be read.

1.4.9 Low Noise Amplifier

The low noise amplifier (LNA) is the first component to process the received signal after it have been sent back through the TR switch. The LNA is designed to increase the signal/noise ratio (SNR) as much as possible. The primary sources of noise, the pre-amplifier, transducer and environmental noise[4], would effect the result much more without a LNA. The LNA is located close to the receiver to reduce the loss of amplitude from the cable.

1.4.10 Variable-Gain Amplifier

After the LNA the variable-gain amplifier (VGA) is located. The VGA amplifies the signal with a varying size of the amplification. This is needed because of the behaviour of ultrasonic waves the amplitude of the signal from the transducer will decrease with time, see (1.1) and (1.3). The pulse response amplitude for the first and second TOA are amplified differently so both can be read. This is done by the VGA. In this case a time-gain control (TGC) is used, that is controlled by the FPGA by time. A TGC is a time controlled VGA that changes the

amplification depending on the time. If the time it takes for each TOA to return and the size of the attenuation for each TOA is known, it is possible to control the TGC with time to increase the amplification for every TOA of one pulse at the time and then reset before the next pulse response returns.

Each TOA is amplified to about the same amplitude and can be read by the analogue to digital converter (ADC) with great accuracy.

1.4.11 Matched Filter

After the signal has been processed through the LNA and the VGA it is filtered through a matched filter. The filter is designed to locate the shape of the pulse that is sent through the transducer. The matched filter will increase the amplitude significantly when it recognises the pulse shape. This results in that the SNR does not have to be as high for the ADC to be able to read the time when the signal arrives[2].

1.4.12 analogue-To-Digital-Converter

The ADC is the last component in the analogue circuit before the process signal will be converted to a digital signal. The ADC works similar to a reversed DAC. The ADC requires an even V_{ppt} for every TOA and a sufficiently high SNR. This makes the biggest requirements for especially the LNA and the VGA. This component allows the information to be read by, for example, a computer and enables many different kinds of usage of the data.

2. RESULTS

In this project a simulation system has been created in two different development environments, Xilinx ISE and MATLAB/Simulink. Xilinx ISE handles the digital circuit simulation and MATLAB/Simulink handles the analogue simulation. The software saves and loads data from files so it is possible to run each simulation with input generated by the other software.

The pulse is made from a MATLAB function and saved to a file. The Xilinx ISE code loads that file, converts it to binary bits and saves it to another file. The Simulink model reads the binary values from that file and uses it as digital input. The result from Simulink is exported back to Xilinx ISE where it is analysed.

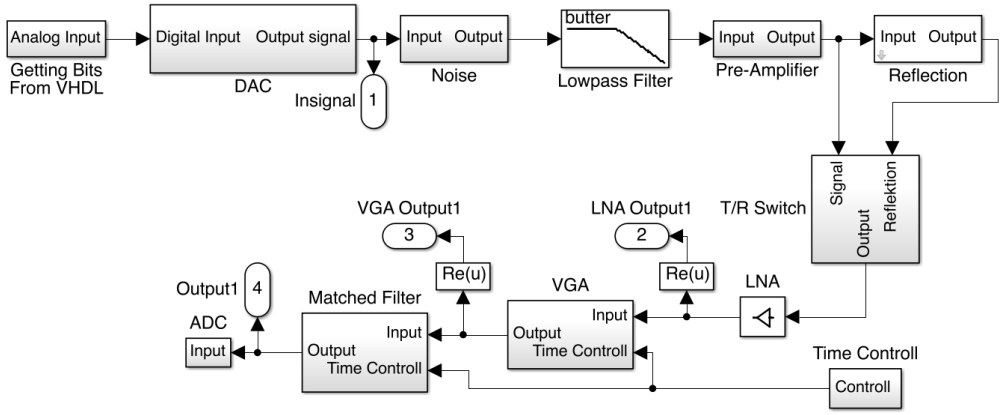

Figure 2.1: Analogue model containing block for each component when importing data from VHDL

The system works well and many variables are easy to configure depending on how the user wishes to run the simulation. The user can also choose to simulate an A-scan, where the result shows the energy for the pulse response, or a normal simulation, where the result shows a normal time diagram for the signal, in the system. An example of a normal scan is shown in Appendix A.

The presentation of results from the analogue model is done by graphs from time scopes. An example of the input and the output of a simulation is shown in Figure 2.2 and 2.3. In this simulation the signal consist of three pulses, with a repetition frequency of 1 MHz where each pulse has a centre frequency of

40 MHz, an initial amplitude of 5 V and a sample frequency of 800 MHz

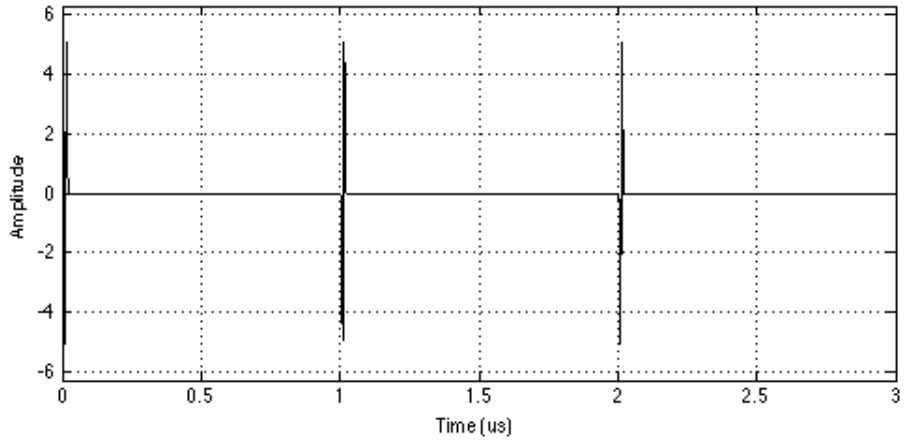

Figure 2.2: The input signal in the analogue simulation in Simulink after passing through the DAC

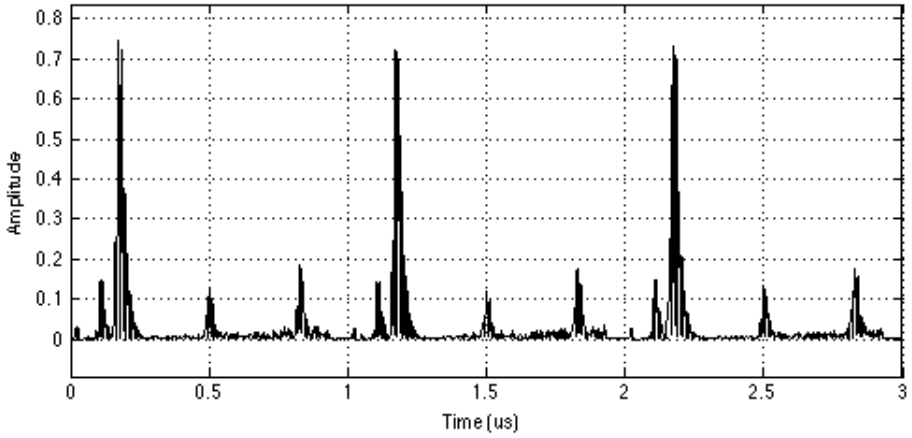

Figure 2.3: Result of the analogue simulation in Simulink before passing through the ADC presented as A-scan

2.1 Components

2.1.1 Input Signal

The input signal consists of bits generated with VHDL-code. This code is stored in a file and loaded into the simulation through a MATLAB function. To handle

the matrix from the file as a signal we use an unbuffer, the block diagram of this in Simulink is shown in Figure 2.4

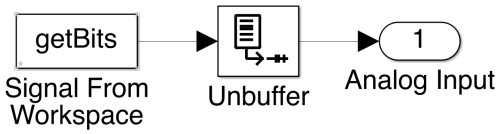

Figure 2.4: The Simulink block to handle the output from VHDL as a digital signal

2.1.2 Digital-to-analogue-Converter

The DAC used in the simulations has a resolution of 17 bits and creates an image of the signal clear enough to use in the rest of the simulation. To create the DAC in Simulink the R-2R ladder network model was used[3]. The simulation block consists of a subsystem with 17 inputs where the first input is the lowest value of the signal and the rest are arranged from MSB to LSB and gained with a decreasing factor of two for each bit, the MSB is gained with $1/2$ and the LSB is gained with $1/2^{16}$. The subsystem for how the MSB to the eighth bit is gained is shown in Figure 2.5

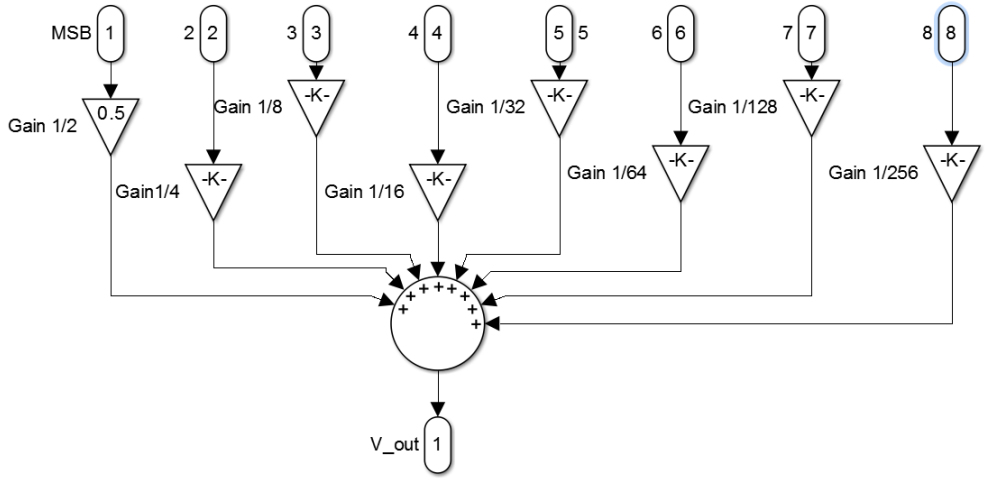

Figure 2.5: Simulink block of the first subsystem of the R-2R ladder

After the bits have been amplified, two's complement is used to calculate the analogue value of the signal at the present time and the signal is gained with the DAC output interval. The signal is then sampled using a "sample and hold" block controlled by the rising edge of a digital clock. To simulate how the DAC

would behave in a real circuit, jitter is added to the sample clock using a random number block and a delay block. The jitter effects the output of the DAC by holding the sample values for a random time before they are sent out. This random delay is shown in Figure 2.6. How the jitter delay is implemented to the sample clock is shown in Figure 2.7. An overview of how the entire DAC is built in Simulink is shown in Figure 2.8.

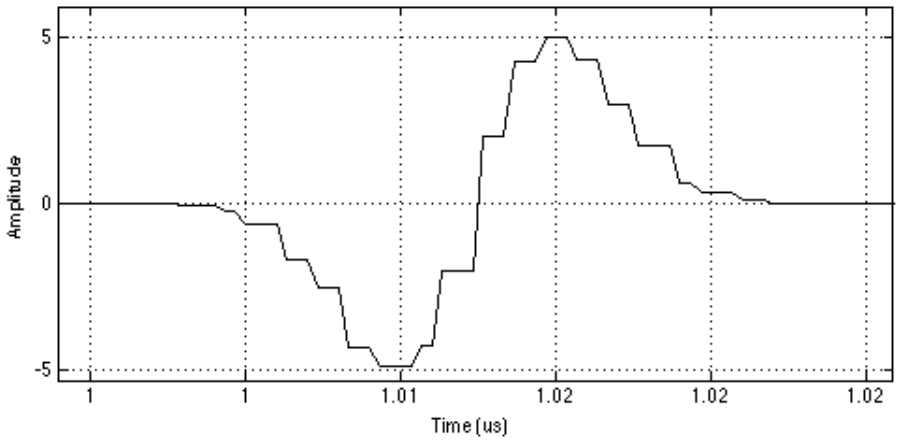

Figure 2.6: The pulse after the DAC with added jitter

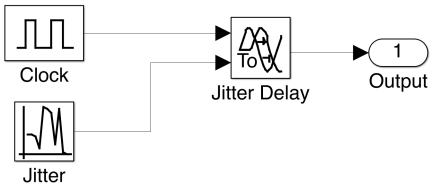

Figure 2.7: Simulink blocks of the sample clock with jitter

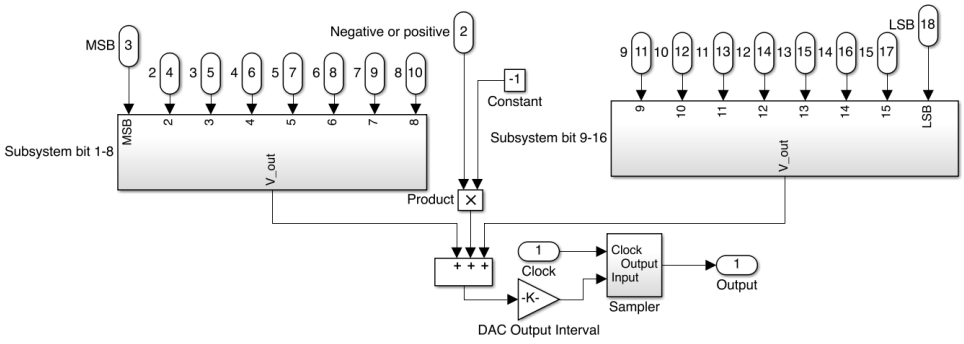

Figure 2.8: Simulink blocks of the DAC, subsystem 1-8 is shown in Figure 2.5

2.1.3 White noise and LP-filter

To implement the white Gaussian noise (WGN) in the simulation model an add white Gaussian noise (AWGN) block is added to the model after the DAC. After the noise is added the signal travels through a Butterworth low pass filter with a passband edge frequency set close to eight times the centre frequency of the original pulses and with an order of seven to filter out high frequency noise and prevent aliasing.

2.1.4 Reflection

The reflection model consists of three time delays, one phase changer, for the first reflection, and a number of gains to simulate the loss of energy the signal gets when it is reflected, transmitted and travels through the materials. White noise and two gain blocks are used to simulate how the transducer behaves in a real life circuit. How the reflection part is built in Simulink is shown in Figure 2.9.

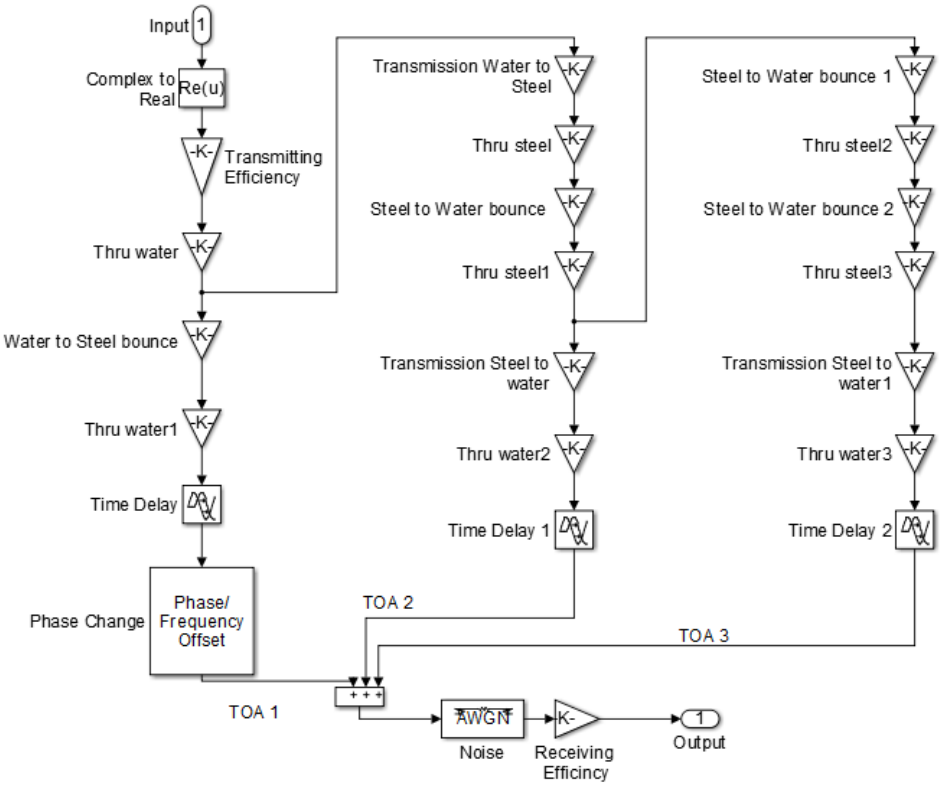

Figure 2.9: Simulink blocks of the reflection model

To calculate the energy lost when the signal travels through the materials (1.1) is used. The absorption coefficients for water is calculated using (1.2). For steel the constant lies between 1-2.5, the value used in the simulation is 2. The energy lost in the reflection and in the transmissions is calculated using (1.3) and (1.4). The time delays are calculated using the thickness and the propagation speed of the pulse in the material and the transfer medium. All the variables are easily changed by the user to simulate different scenarios.

2.1.5 Transmitter/Receiver Switch

By comparing the transmitting signal to a constant the switch will only let signals through when the amplitude of the transmitting signal is lower than a specified constant. This should avoid the LNA from getting blinded. Ground is connected to the other input to simulate the switch when it is disconnected.

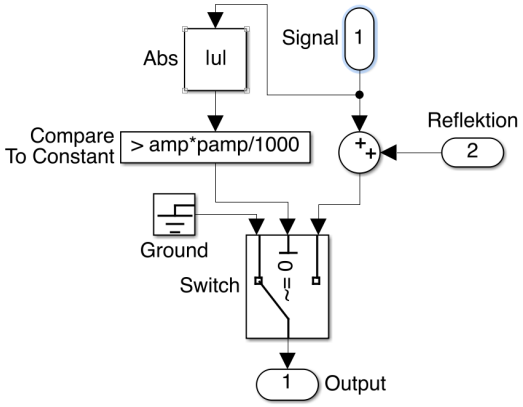

Figure 2.10: Simulink block of the TR switch created for the model

2.1.6 Pre-Amplifier and LNA

To simulate the LNA and the pre-amplifier the block Amplifier from SimRF blockset[6], a blockset contained in the student version of Simulinks library 2013a, is used. This block allows noise and non-linearity to be added if wished for. Approximate values for LNA and the pre-amplifier are used in the model. The amplifiers are implemented in the model as shown in Figure 2.1.

2.1.7 Variable-Gain Amplifier

The VGA, in this case a TGC, is represented by a MATLAB function that is configured for the specific simulation to increase the amplification linearly for the

period of the pulse response and reset before the first TOA of the next pulse arrives. The linear amplification of the VGA can be seen when looking at the noise in Figure 2.11.

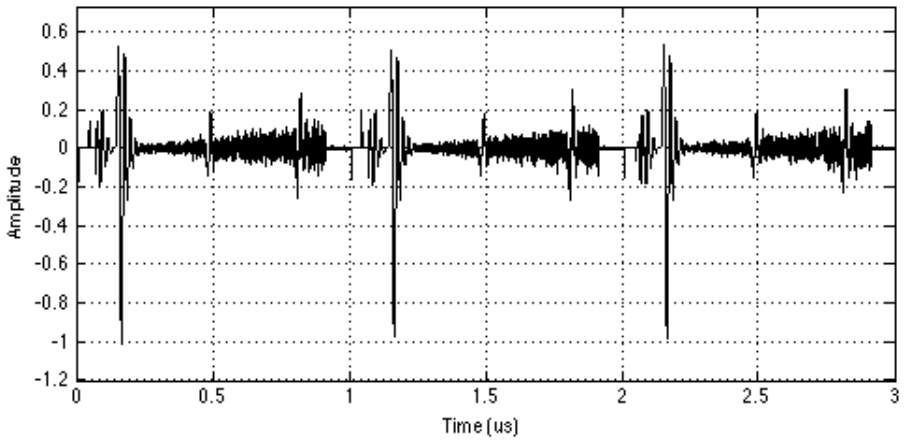

Figure 2.11: Time scope from Simulink after the VGA

2.1.8 Matched Filter

To simulate the matched filter without any delay a buffer is used to store as many samples as the pulse consists of. A MATLAB function is used to compare the signal to the filter and then only put out the last value to get a fair result. The function calculating the filtered signal also controls what type of scan the user wishes to run, normal or A-Scan. The entire block diagram of the matched filter in Simulink is shown in Figure 2.12.

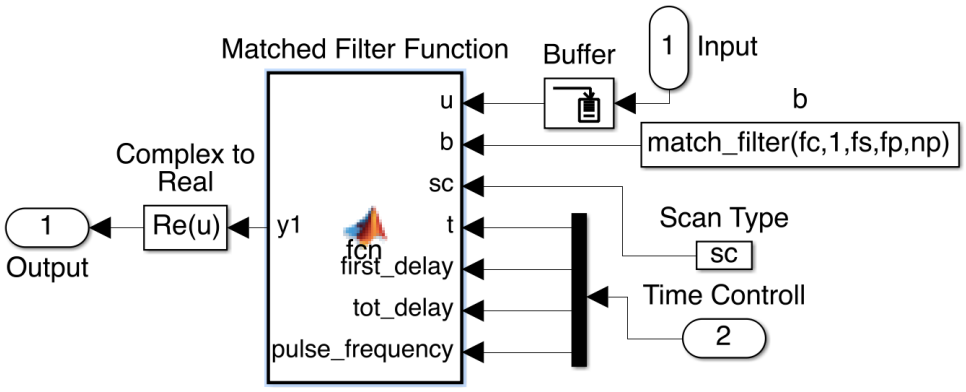

Figure 2.12: Simulink block of the matched filter created for the model

2.1.9 Output Signal

Conversion of the signal to digital bits is done by an ideal ADC and an "integer to bit converter". The bits are saved to workspace and through a function saved to a file that can be read by VHDL. The simulink block can be seen in Figure 2.13.

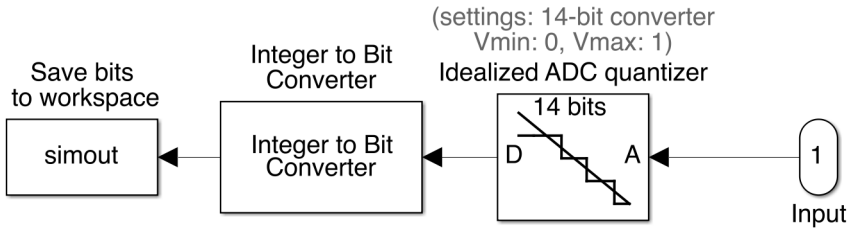

Figure 2.13: The Simulink blocks to convert the analogue output to digital and saves it

2.2 VHDL

2.2.1 Pulse generation

The pulses used in the simulations are first generated in MATLAB from the pulse generation script. The samples from the pulse generation are saved to a text file. The VHDL code for pulse generation loads data from the text file, converts it to binary numbers and uses it as output. The output is saved to another text file and is used as digital input for simulations in Simulink. Figures of a pulse generation in both MATLAB and Xilinx ISE can be seen in Appendix A.

2.2.2 Signal processing

The pulse response from the simulations in Simulink is saved to a text file as binary samples. The VHDL script for signal processing reads the data from the text file and runs a simulation with the data as input. If the value of the input is considered as a peak the time it occurs is saved to a slot in the RAM. A figure of a simulation in Xilinx ISE where the time for values considered as a peak, a TOA, are saved to RAM can be seen in Appendix A.

3. DISCUSSION

The relatively new area required quite some reading up on until the work with the model could begin. The challenge to decide by ourselves how the model would be built by just mimic the behaviour, not construct the exact component, allowed some experimenting with different already existing components in Simulink. The experimenting resulted in a deeper understanding of the system at the same time progress was made with the model. This was an important thing in the completion of the model. Because of the lack of experience with the development environments, except for MATLAB alone, the time estimation of the work required was tough.

The work with MATLAB/Simulink went on parallel to the work with VHDL. Not much co-operation between the two software was tested during the development. The data transferring system between the software was designed when the model was complete. The pulse-generation function was used to generate discrete values and transferred manually to VHDL. From those values the digital pulse generation was simulated in VHDL.

To be able to test different scenarios more efficiently another similar Simulink model was created. The stand-alone model uses the pulse function as an input and converts it into a digital signal using an ADC. Some figures of this model is shown in Appendix A.

3.1 Components

3.1.1 Digital-to-analogue-Converter

Creating the DAC, several already existing DAC blocks in Simulink were tested at first. Non of these achieved the desired result. A R-2R ladder model was therefore created to simulate the behaviour similar to that of a DAC. The model satisfies the requirements even though it doesn't behave completely like an actual DAC.

Jitter was added to the DAC by delaying the sample clock. To achieve a realistic result the delay was a normally distributed random number in the interval $[0, 2 \cdot jitter]$, where the size of the constant $jitter$ is decided by the user. The delay made the jitter easy to adjust and the normal distribution is just an example used to show the effect of the jitter in the result.

3.1.2 White noise and LP-Filter

Without the WGN the signal would be completely cut of at certain times when being filtered. In the Simulink model the SNR of the WGN can be set by the user to simulate different scenarios.

The requirements of the LPF were to keep the order of the filter low and prevent any aliasing. The Simulink block used to achieve this is the "analogue filter design" block designed as a Butterworth LPF with an order of seven and a passband edge frequency of eight times the centre frequency of the pulses. This resulted in a smooth signal with only a small amount of noise.

3.1.3 Reflection

The first design of the reflection block consisted of few gain block in an attempt to make is as compact as possible. This made adjustments more complicated to make. When representing every single source of attenuation by a gain block adjustments can easily be made.

When the signal travels from the less dense medium to the more dense medium in the first reflection a phase shift of 180 degrees occurs. The phase of the first reflection in the simulation must therefor be shifted. To implement this a Phase/Frequency offset change block was added.

The low efficiency of an actual piezoelectric transducers when transmitting and receiving is simulated with gain blocks in the beginning and in the end of the reflection block. The values of these gain blocks are adjustable to simulate different transducers. When using gain block both the signal and the noise will change with the same ratio. In an actual transducer this is not the case. A WGN was therefor added at the end of the block to let the user adjust the noise level. Noise created during transmission and receiving is also simulated by that WGN block.

There were some difficulties to find reliable sources for the absorption coefficients for water and steel. There are different values for steel depending on how the material is made. An approximate constant was used in the simulation made to test the model. It is easy for the user to change the value depending on what type of material that is being examined. The attenuation constant for water was calculated from the centre frequency by using (1.2.)

3.1.4 Transmitter/Receiver Switch

In reality the switch is a much more complicated component than the one made in this model. Some energy will always leak through the TR switch into the analyse part. The model is designed to either let nothing or everything

through. This way the goals of the simulation were fulfilled . By changing the constant the signal is compared to it is possible to adjust when the signal will be cut of. This way it is possible to, if the noise level is calculated, almost completely block out any part of the signal. The noise level specific values for all the previous components needs to be calculated. To make this work in general is very difficult. This would result in a small leakage that could be seen as an imitation of the actual leakage or just a small flaw in the model.

3.1.5 Pre-Amplifier and LNA

This was quite easy to implement because of the amplifier block in the SimRF blockset. The block has settings for both non-linearity and noise. The block requires a complex signal. The complex part $0i$ was added to the signal to get the model to run.

3.1.6 Variable-Gain Amplifier

The VGA is an important component for analysing the second and the third pulse response. The model started with only an amplifier, similar to the LNA with the intention to get a running model. This was not helpful since the first pulse is so much stronger than the rest. Trying to use the same Simulink block as for the pre-amplifier and the LNA made the time controlling of the VGA very hard. A MATLAB function was created for the time controlling. Difficulties occurred when trying to create a system that distinguishes the pulse responses from the noise with good SNR. The function of the VGA is easily configured inside the MATLAB function block for modification.

3.1.7 Matched Filter

Calculating matched filters in MATLAB was done by using the function "filter(B, A, X)". The filter function works as a matched filter when B is a time reversed vector of the pulse. The symmetry of the pulse used in the model makes it possible to do the time reversing the pulse by multiplying it with -1. X needs to be an array with the minimum length of the pulse to make it possible for the function to compare the signal with the complete pulse.

To make the simulation run in real time with present data at the correct time a buffer was added. The buffer holds as many samples as the pulse is long and sends it to the MATLAB function that calculates the filtering. For every new array it sends out, all the values except one will be the same. The output from the function that calculates the filtering is the last value of the array. This is done because of how the filter function presents the result. When the function

returns the first value of the array the last few data samples will be lost and if it returns the last value the first few data samples will be lost, but since the last samples are the ones indicating if the signal is similar to the pulse the function returns the last value of the array. No important data will be lost because of the delay in the reflection.

The function also calculates the settings for the scan type. This is done by setting a constant at the beginning of the simulation corresponding to the scan type chosen. The normal scan shows the filtered signal and the A-Scan calculates the energy level of the filtered signal. Removing the energy added by the filter normalisation is done by dividing the result by a normalisation constant calculated from the equation

$$norm = \frac{1}{\sum_{i=1}^{m} (b_n)^2} \; [-]. \tag{3.1}$$

3.2 VHDL

The original plan was to do a co-simulation between Xilinx ISE and MatLab/Simulink but it required licenses or software which were not available. Instead separate simulations were done in Xilinx ISE and MatLab/Simulink by saving to and loading from files. In Figure A.9 and Figure A.10 in Appendix A, which shows the zoomed in second pulse in a generated pulse train in both MATLAB and Xilinx ISE, different sample clocks where used in the simulations which created a small error. This can be fixed by using more accurate sample clock frequencies. In the VHDL code for pulse generation the sample frequency used were 0.334ns while the actual MATLAB frequency were $\frac{1}{3}$ns. The load/save data to file system requires that the data is printed in the right format. In MATLAB the saving to and loading from files is done with MATLAB functions and in VHDL it is done in the scripts for pulse generating and signal processing.

In the VHDL code read data from simulation, that is used for saving the pulse response to RAM in Xilinx ISE, the specification for what defines a pulse response was made by running a simulation with the output from the Simulink system converted to integer and looking at the pulse response. An example of this can be seen in Appendix A. The specification for a pulse response can be configured by adding more conditions to the code.

4. CONCLUSION

In conclusion, the two systems that was created works well and all the variables are easy to change so the user can adapt the program depending on what type of situation the user wants to simulate. The model is built to simulate an ultrasound system and can handle frequencies in the interval used for non-destructive ultrasonic testing. To run a simulation the user has to define all the signal, material and transport medium attributes that are required. An example of this is shown Appendix A.

Because of the time frame of this project the main focus was to create a working model of a complete system and therefore the optimisation of each component in the analogue model had lower priority than making the model run smooth. For example, the MATLAB function blocks lacks noise and in the reflection part the noise is added to all of the signals in the end instead of after every time the signal goes through a gain block. Since the function blocks works as a normal MATLAB function the noise addition can be implemented by anyone with a little experience in the area, to fix the noise addition in the reflection part multiple AWGN blocks can be added.

Another problem with the model is that no co-simulations between MATLAB/Simulink and Xilinx ISE were done since no licenses or free software needed were available. Instead the simulation is done by saving and loading information from files between the programs.

Bibliography

[1] Jack Blitz and Geoff Simpson. *Ultrasonic methods of non-destructive testing*, volume 2. Springer, 1995.

[2] R.V. Canales and C. Massatoshi Furukawa. Signal processing for corrosion assessment in pipelines with ultrasound pig using matched filter. In *Industry Applications (INDUSCON), 2010 9th IEEE/IAS International Conference on*, pages 1–6, 2010.

[3] Alexandre César Rodrigues da Silva and Ian Andrew Grout. *A Methodology and Tool to Translate MATLAB®/Simulink® Models of Mixed-Signal Circuits to VHDL-AMS*. 2011.

[4] G. Hayward, R.A. Banks, and L. B. Russell. A model for low noise design of ultrasonic transducers. In *Ultrasonics Symposium, 1995. Proceedings., 1995 IEEE*, volume 2, pages 971–974, 1995.

[5] Josef Krautkrämer, Werner Grabendörfer, and Herbert Krautkrämer. *Ultrasonic testing of materials ...* Springer, Berlin, 3. ed. edition, 1983.

[6] Mathworks. Simrf, design and simulate rf systems, May 2013.

[7] Carl Nordlning and Jonny Österman. *Physics Handbook for Science and Enginering*, volume 8:5. Studentliteratur AB, Lund, Sweden, 2010.

[8] Dharmendra Kumar Pandey and Shri Pandey. Ultrasonics: A technique of material characterization. *Acoustic Vaves*, pages 397–430.

[9] Aerospace Eng. & Eng. Mechanics Peter B. Nagay. Ultrasonic nondestructive evaluation. University Lecture, 2003.

[10] Niranjan Talwalkar. *Integrated CMOS transmit-receive switch using on-chip spiral inductors*. PhD thesis, Stanford University, 2003.

A. FIGURES

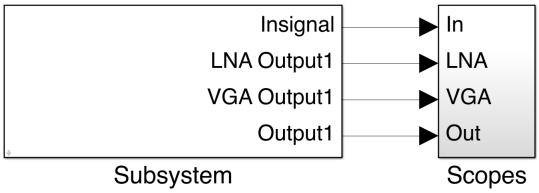

Figure A.1: Simulink model at the highest order when importing data from VHDL

A.1 Stand-Alone Model

These figures are the Simulink blocks that are changed when we adjusted our model to generated the pulse train directly in MATLAB/Simulink instead of in Xilinx ISE.

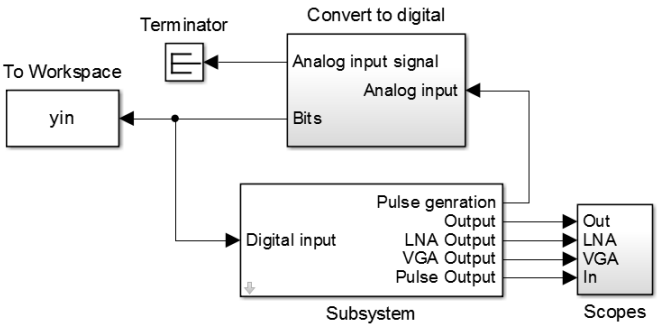

Figure A.2: Simulink model using input from MATLAB instead of Xilinx ISE at the highest order

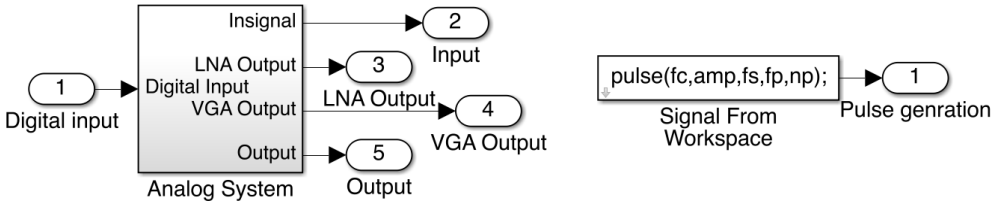

Figure A.3: Subsystem containing pulse generation and the analogue system using input from MATLAB instead of Xilinx ISE

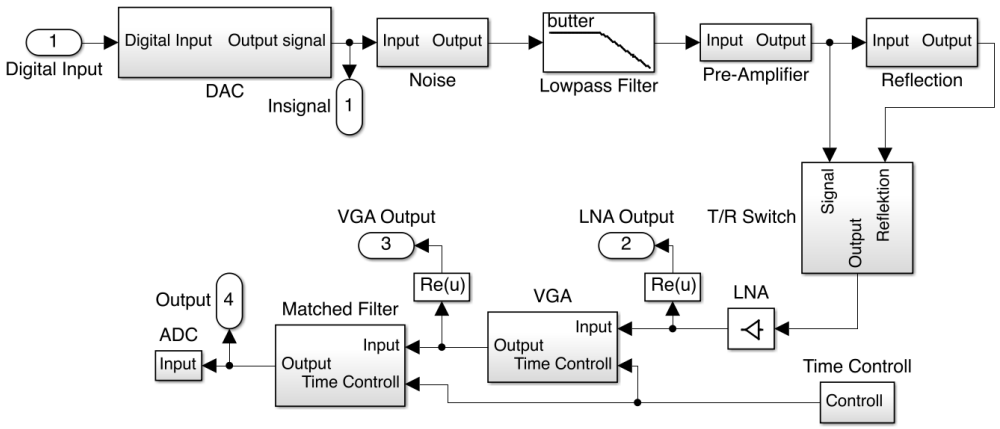

Figure A.4: Analogue model using input from MATLAB instead of Xilinx ISE containing blocks for each component

A.2 Parameter Window

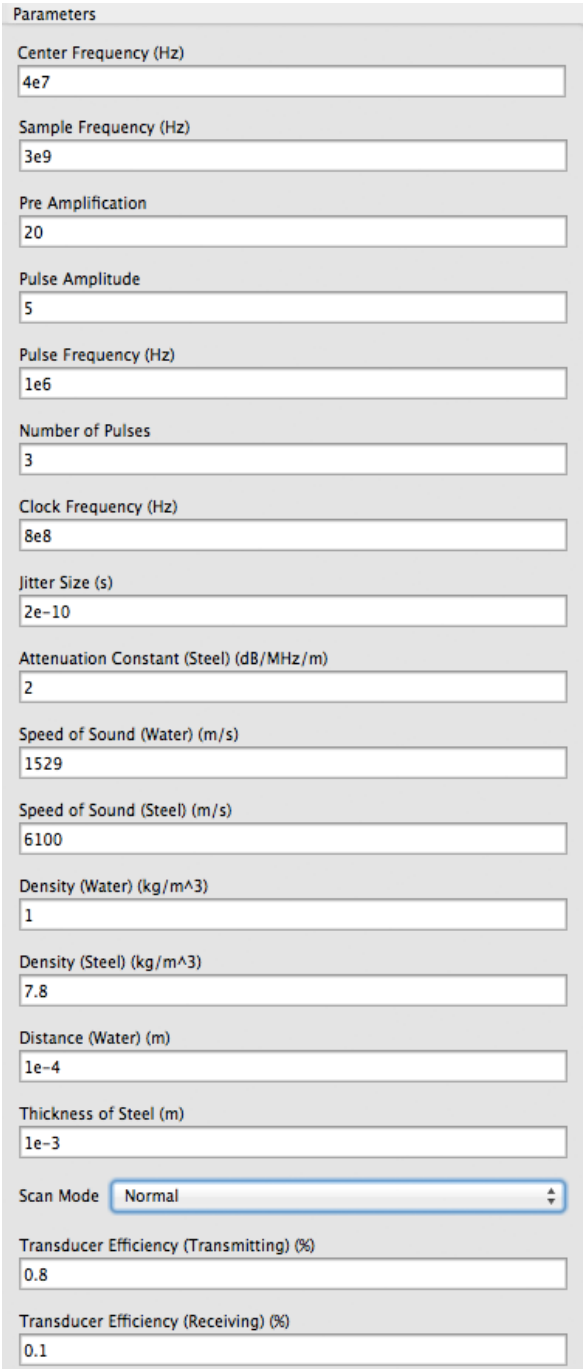

Figure A.5: The list of parameters that the user can change

A.3 Normal Scan

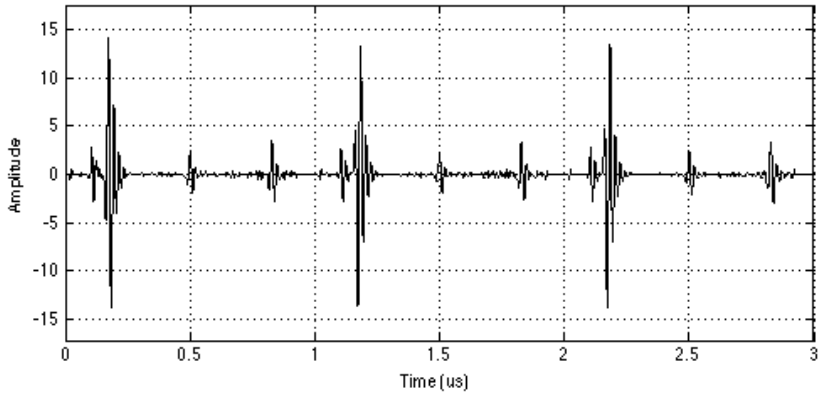

Figure A.6: Plot of the scan result when using normal scan

A.4 Xilinx ISE and MATLAB related pulse generation figures

The figures shows the generated pulse train in both MATLAB and Xilinx ISE. In the figures there is a small error because of the different sample clocks used in the simulations. The Xilinx ISE figures represents the pulse train in 17 bits.

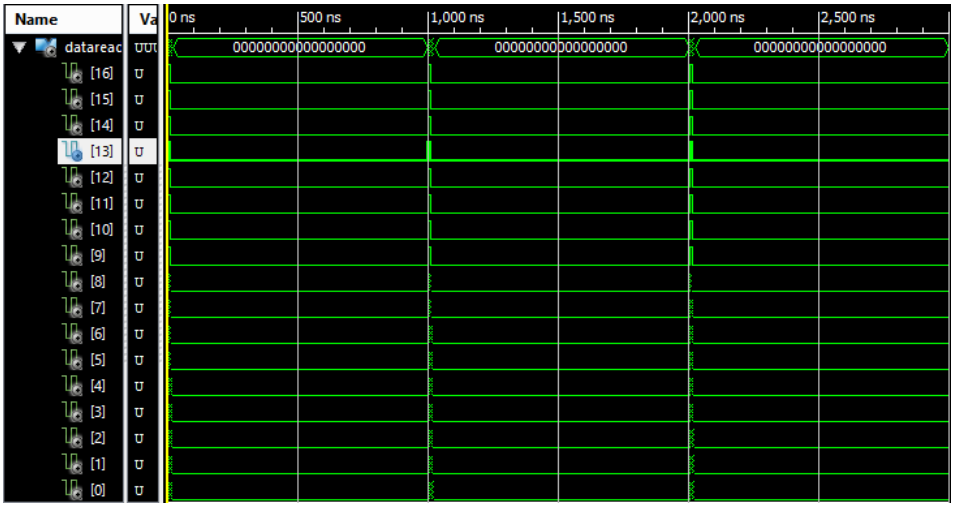

Figure A.7: The generated 17 bits pulse train, Xilinx ISE

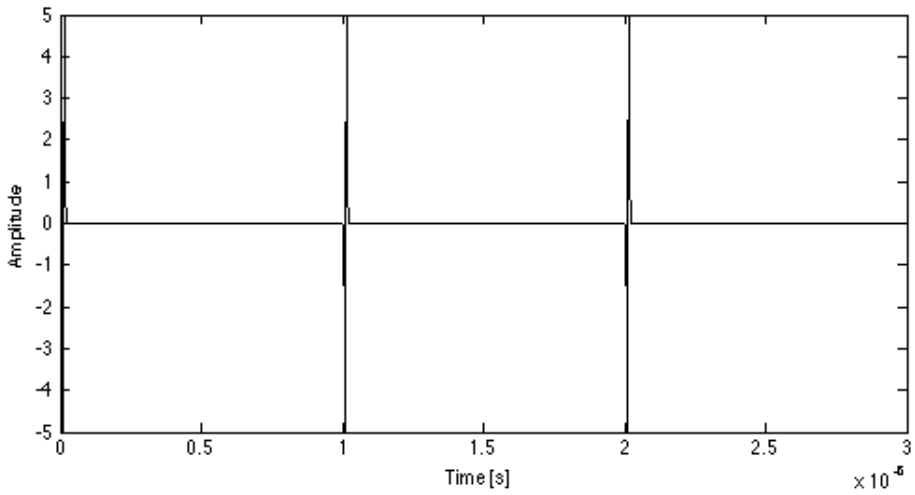

Figure A.8: The generated pulse train, MATLAB

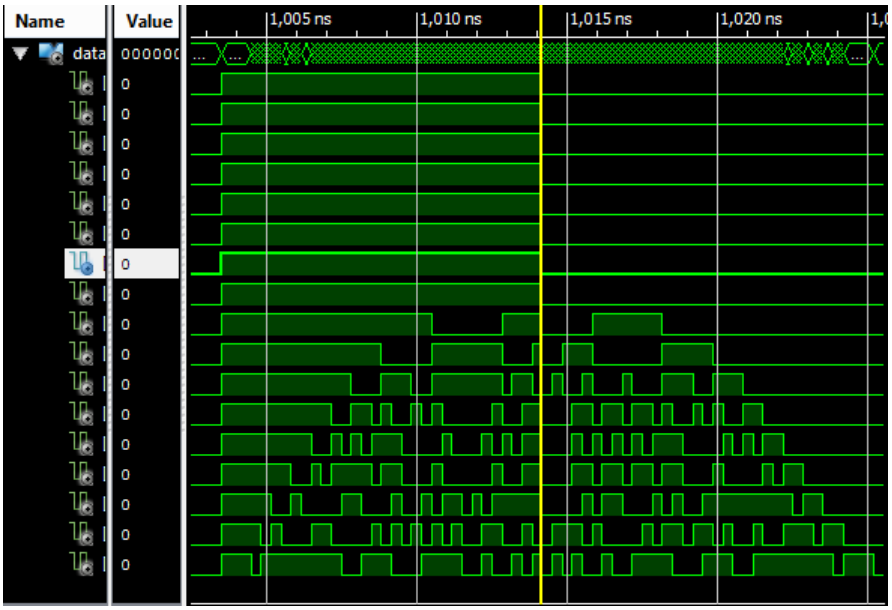

Figure A.9: The zoomed in second pulse in the generated 17 bits pulse train, Xilinx ISE

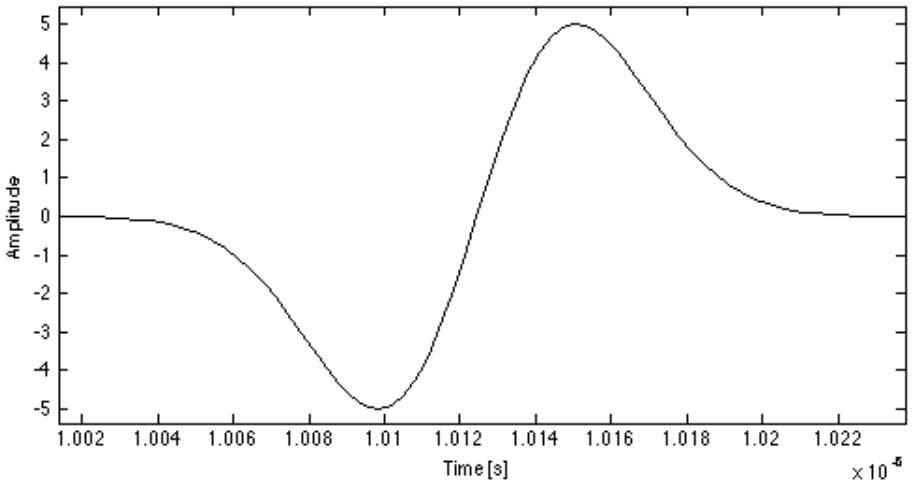

Figure A.10: The zoomed in second pulse in the generated pulse train, MATLAB

A.5 Signal processing Xilinx ISE

The figure shows signal processing for the first pulse response from the Simulink output. When the 14 bits value converted to integer is considered as a peak the time of arrival for the peak is saved to RAM.

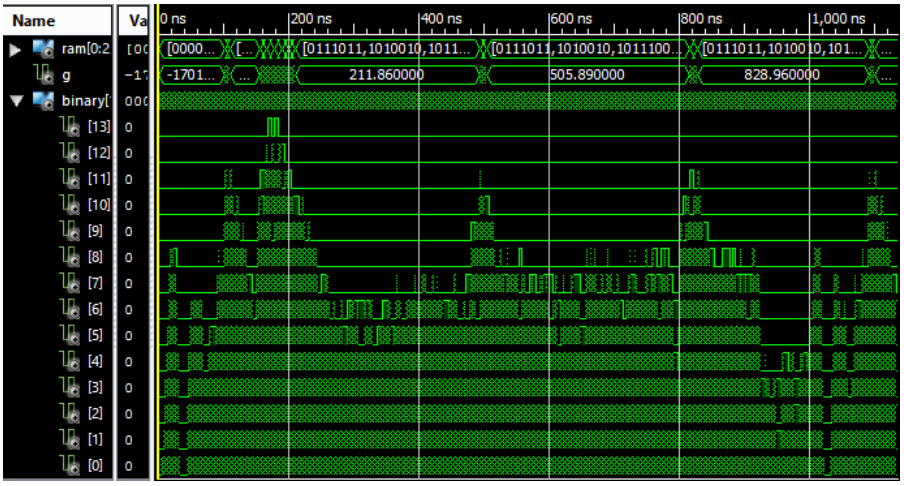

Figure A.11: The signal processing for the 14 bits pulse respons in the interval 0 ns to 1140 ns in Xilinx ISE, saves to RAM when the value converted to integer is higher than 1500

B. CODE

B.1 Matlab

```
function [y] = pulse(c_freq, amp, samp_freq, pulse_freq, ...
    num_pulses)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

if c_freq*2≤samp_freq && pulse_freq≤c_freq
    %Setting sample frequency and center frequency
    fs=samp_freq;
    fc=c_freq;
    %Time vector
    t=linspace(-0.5/fc,0.5/fc,fs/fc);
    %Adjusts the shape of the pulse
    s=0.15;
    a=fc*t/s;
    %Normalizes the pulse to an amplitude of 1
    norm=1/(exp(-1/2)/sqrt(2));
    %Creating the pulse
    yp=amp*norm*a.*exp(-a.^2);
    %Create the space between pulses
    space=zeros(1,(1/pulse_freq-1/fc)*fs-1);
    %Create a time vector corresponding to the pulse ...
       frequency and the
    %number of pulses
    time=linspace(0, ...
       num_pulses*(1/pulse_freq),num_pulses*(length(yp)+length(space)));
    %Create the pulse train
    y=[];
    for i=1:num_pulses
        y=[y yp space];
    end

    %Matched filter
    %b=yp(end:-1:1)/amp;

    %Plot the pulses in time
    %plot(time,y)

    %Plot filtered signal
```

```
    %fil=filter(b,1,y);
    %plot(time,fil)
end
```

```
y = pulse(fc,amp,fs,pf,np); %fc=center frequency, ...
    amp=amplitude, fs=sample frequency, pf=pulse frequency, ...
    np=nr of pulses

vec_lsb = 2^(-6); %
vec_int = round( y / vec_lsb ); %converts the decimal values ...
    to integers where vec_lsb determines the accuracy

fid = fopen('3.txt','wt');  %3.txt is the file, wt for ...
    writing in text mode
fprintf(fid,'%4.0f\n',vec_int);  %Writes vec_int as one ...
    integer per row.
fclose(fid); %closes the file
```

```
function y = vga(u,t,first_delay,tot_delay,pulse_frequency)

time=mod(t-first_delay,1/pulse_frequency); %Calculates where ...
    in the period the signal is

if t≤first_delay %Only amplify if the first pulse resonce ...
    has arrived
    y=u;

elseif time>tot_delay %Does not amplify if the the signal is ...
    after the third pulse responce and before the first ...
    responce of the next pulse
    y=u;

else
    gain=16*time/tot_delay; %Amplifies linear to time
    y=u*gain;

end
```

```
function y1 = ...
    mathed_filter(u,b,sc,t,first_delay,tot_delay,pulse_frequency)
%Reverses the filter if it's the first pulse response
if mod(t-first_delay,pulse_frequency)≤first_delay
     b=-1*b;
end
%Filters the signal, u is as many samples as the pulse itself.
y = filter(b,1,u);
y1=y(end); %Only put out the last element.


%Settings for the scantype. A-Scan will calculated the ...
    energy while Normal
%will display the signal as it is.
switch sc
     case 1
         %Normal
         y1=y(end);
     case 2
         %A-Scan
         norm=1/sum(b.^2);
         y1=abs(y(end)*norm);
end
```

```
function Dataout = getBits()

A = textread('2.txt', '%s'); %Select file to read
ncols = size(A, 1); %Calculates number of columns in the A ...
    matrix
nrows = size(A{1}, 2); %Calculates number of rows in the A ...
    matrix
A = reshape(sscanf([A{:}], '%1d'), nrows, ncols); % Reshapes ...
    A to a readable matrix
Dataout=A';
```

```
function y = saveToFile(hey)

fid = fopen('1.txt','wt');  % Opens file to write to and ...
    note the 'wt' for writing in text mode
% Put outs the data in a format that is readable in VHDL
```

```
fprintf(fid,'%1.0f %1.0f %1.0f %1.0f %1.0f %1.0f %1.0f %1.0f ...
   %1.0f %1.0f %1.0f %1.0f %1.0f %.0f\n', hey');
fclose(fid);
```

B.2 VHDL

```
library std;
use std.textio.all;  --include package textio.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use IEEE.std_logic_textio.all;

--entity declaration
entity filehandle is
end filehandle;

--architecture definition
architecture Behavioral of filehandle is

--period of clock,bit for indicating end of file.
signal clock,endoffile : bit := '0';

--type for the data to write.
signal    dataread : std_logic_vector(16 downto 0);

signal    datatosave : real;

--line number of the file read or written.
signal    linenumber : integer:=1;

begin


clock <= not (clock) after 0.167 ns;    --clock ...
   with time period 0.334 ns(sample time)


--read process
reading :
process
```

```
    file   infile    : text is in  "3.txt";   ...
       --declare input file
    variable  inline    : line; --line number ...
       declaration
    variable  dataread1    : real;
begin
wait until clock = '1' and clock'event;
if (not endfile(infile)) then   --checking the ...
   "END OF FILE" is not reached.
readline(infile, inline);       --reading a line ...
   from the file.

read(inline, dataread1);
dataread <=...
   std_logic_vector(to_signed(integer(dataread1),17)); ...
   --converts the data to binary
else
endoffile <='1';          --set signal to tell end ...
   of file read file is reached.
end if;

end process reading;

--write process
writing :
process
    file      outfile  : text is out "2.txt";  ...
       --declare output file
    variable  outline  : line;   --line number ...
       declaration
begin
wait until clock = '0' and clock'event;
if(endoffile='0') then   --if the file end is not ...
   reached.
--write(linenumber,value(binary ...
   type),justified(side),field(width),digits(natural));
write(outline, dataread, right, 17);
-- write line to external file.
writeline(outfile, outline);
linenumber <= linenumber + 1; --jumps to next line
else
```

```
null;
end if;

end process writing;

end Behavioral;
```

```
library std;
use std.textio.all;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use IEEE.std_logic_textio.all;

entity saveram is

end saveram;

architecture Behavioral of saveram is

type ram_t is array (0 to 255) of ...
   std_logic_vector(6 downto 0); --256 slots ram, ...
   7bits each
signal ram : ram_t := (others => (others => '0'));
signal c : integer range 0 to 255:=0; --index for ...
   slot in RAM.
signal i : integer range 0 to 10000:=0; ...
   --minimum: nr of samples
signal g : real;


--period of clock,bit for indicating end of file.
signal endoffile : bit := '0';

--data read from the file.
signal    dataread : real;
```

```
--type of data from the input file.
signal    binary : std_logic_vector(13 downto ...
   0):="00000000000000"; --size of file to read

signal    datatosave : integer;

signal    dataint : integer range -10000 to ...
   10000:=0;
--line number of the file read or written.
signal    linenumber : integer:=1;


type state_type is (s0,s1);

signal state: state_type :=s0;

begin


--read process
reading :
process

    variable line_content : string(1 to 14); ...
       --size of file to read
    variable line_num : line;
    variable j : integer := 0;

    variable char : character:='0';
    file   infile    : text;

begin

file_open(infile,"1.txt",READ_MODE); --declare ...
   input file
while not endfile(infile) loop --loop as long as ...
   not the end of the file
readline (infile,line_num); --line to read in file
READ (line_num,line_content);
```

```
for j in 1 to 14 loop
            char := line_content(j);
            if(char = '0') then
                binary(14-j) ≤ '0'; --converts ...
                   string to binary
            else
                binary(14-j) ≤ '1'; --converts ...
                   string to binary
            end if;
        end loop;
          i ≤ i+1;
          dataread ≤ ...
             real(to_integer(unsigned(binary))); ...
             --binary to real
          dataint ≤ integer(dataread); --real to ...
             integer

          --State code, changes state and saves ...
             data when specified conditions are ...
             fulfilled
     if (state = s0) then
     if(dataint > 1500) then --conditions for a ...
        state change (input value higher than 1500)
         ram(c) ≤ ...
            std_logic_vector(to_unsigned(i,7)); ...
            --time for event
        c ≤ c+1; --c is the RAM slot to save data in
        g ≤ 0.33*real(i); --time when data is ...
           saved to slot
      state ≤ s1; --changes state as soon as ...
         specified conditions are fulfilled
     end if;
    elsif (state = s1) then
    if(dataint <1500) then --conditions for a ...
       state change (input value lower than 1500)
      state ≤ s0; --changes state as soon as ...
         specified conditions are fulfilled
    end if;
     end if;
```

```
        wait for 0.33 ns; --after reading each ...
           line wait for 0.33ns(sample time).
        end loop;
        endoffile <= '1'; --sets endoffile=1 when ...
           all the lines in the file are read
      file_close(infile);  --after reading all ...
         the lines close the file
        wait;
end process;
end Behavioral;
```