DWT compression.
C and VHDL implementation.

Adalberto Claudio Quirós.

August 2013.

Compresion en DWT . Implementacion en C y VHDL.

Autor : Adalberto Claudio Quiros
Tutor: Phd. Jafar Sannie
Illinois Institute of Technology, Chicago
12 of August 2013.

Index.

Abstract.

In this report we are going to explain how it was implemented the design of an accurate design to filter and compress an ultrasonic signal. We are going to analyze the Discrete Wavelet Transform (DWT). Because the Continuous Wavelet Transform (CWT) has information that is highly redundant and when we decide to reconstruct the signal, all that information is useless. Therefore, sampling the CWT, we are able to take into account only the kind of information that is necessary for an accurate reconstruction of the signal, but with much less information. In addition, DWT gives us enough information to study and analyze the original signal, reducing the computation time needed.

In the implementation it has been followed several steps that will be explain along this report together with the tools used. The implementation has been design to be implemented finally in a FPGA, but this step wasn't achieved because we ran out of time.

Introduction.

The discrete wavelet transform (DWT) it has been essential in this project because its properties and its utility in compression that has been effectively demonstrated. Therefore, knowing that we are going to work with DWT, we are going to implement filters with concrete parameters to compress the signal as good as possible.

Along the project we have used different tools to obtain an accurate result. The language programing that has been used are C, in parallel to MatLab, and VHDL to generate the implementation to FPGA.

First of all, we have to learn the basis from our project: DWT and its properties.

Discrete wavelet transform:

It is very important to understand firstly what is a wavelet and which are the advantages that we can obtain from this kind of representation. A wave is an oscillating function of time or space that is periodic. On the other hand, wavelets are localized waves. They have energy concentrated in time or space. A wavelet is an orthogonal function that can be applied to a finite group of data. Figure 1 is an example of a wave and a wavelet.

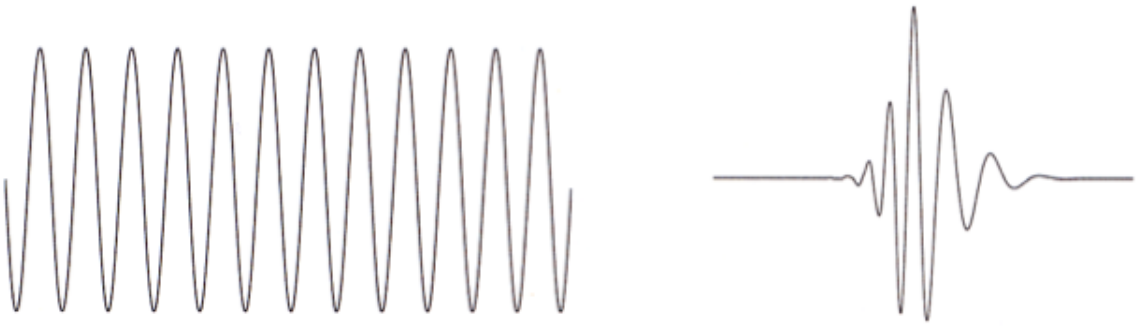

Figure 1. Demonstration of (a) a wave and (b) a wavelet

The wavelet basis is a set of functions that are defined by a recursive difference equation:

$$\phi(x) = \sum_{k=0}^{M-1} c_k \phi(2x - k)$$

The range of the summation is determined by a specified number of nonzero coefficients M, which is arbitrary and will be referred to as the *order* of the wavelet. This class of wavelet functions is, by definition, constrained to be zero

outside a small interval. This makes the wavelet transform able to operate on a finite set of data. Here we have the first property, called "compact support". Most wavelet functions appear to be extremely irregular because the fact that the recursion equation assures that the wavelet function is not differentiable everywhere. Therefore, the functions that are selected to perform transforms are the ones that have a discernible shape. Figure 2 shows us two examples of functions used. In this figure we observe the Haar function and the Daubechies 4 function.

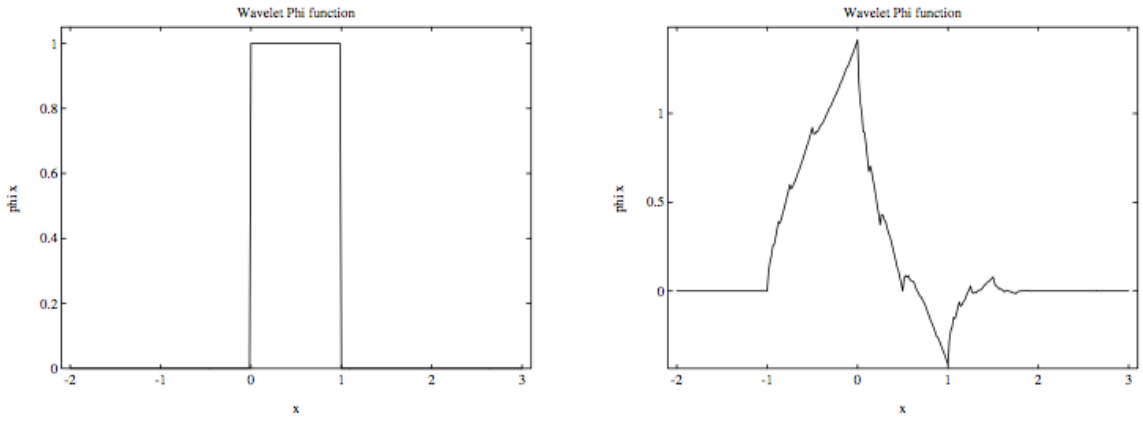

Figure 2. Haar and DB4 wavelet basis functions

In general the signals are time-domain signals in their raw format. Therefore, it will be very useful to be able to have representation of the signal to study the signal from this representation. This is where Wavelet Transform is very useful because it provides us a time-frequent representation of the signal so we can study its properties from this representation.

The Discrete Wavelet Transform is based on the Short Time Fourier Transform (STFT). The STFT is a way of changing the Fourier Transform for those signals that are not stationary. If this region where the signal is going to be assumed to be stationary is too small, we look at that signal from narrow windows. They should be narrow enough that the portion of the signal seen from these windows is indeed stationary. Therefore, it is going to used a version of the Fourier Transform: the STFT.

The difference between STFT and Fourier Transform is that in the STFT the signal is divided into small segments that are going to be assumed stationary. Thus, we have to choose and appropriate value for the window "*w*", where the width of the window must be equal to the segment of the signal where it is assumed to be stationary.

$$STFT_X^{(w)}(t,f) = \int_t [x(t) \times w^*(t-t')] \times e^{-j2\pi ft} dt$$

where $x(t)$ is the signal, $w(t)$ is the window function, and the operator * is the complex conjugate.

The wavelet series is just a sampled version of the Continuous Wavelet Transform (CWT) and its computation may consume significant amount of time and resources depending on the resolution required. On the other hand, the DWT is based on sub-band coding is found to yield fast computation of wavelet transform.

The idea of the DWT is very similar that the one that is used in the CWT. In the continuous case the signals are analyzed using a set of basis functions, which relate to each other by simple scaling and translation. Meanwhile, in the discrete case, a time-scale representation of the digital signal is obtained using digital filtering techniques.

Therefore, in the discrete case it is going to be used different kinds of filters, of high and low pass, to analyze the signal in different scales. The resolution obtained is changed by upsampling and downsampling operation. Downsampling, or subsampling, a signal corresponds to reducing the sampling rate or removing some of the samples of the signal. Subsampling by a factor '*n*' reduces the number of samples in the signal '*n*' times. Upsampling a signal corresponds to increase the sampling rate of a signal by adding new samples to the signal. Upsampling by a factor '*n*' increases the number of samples in the signal '*n*' times. The following figure (Figure 3) shows the idea of the DWT's funds: driving the signal through the high pass and low pass filter we obtain the approximation and the details of the signal.

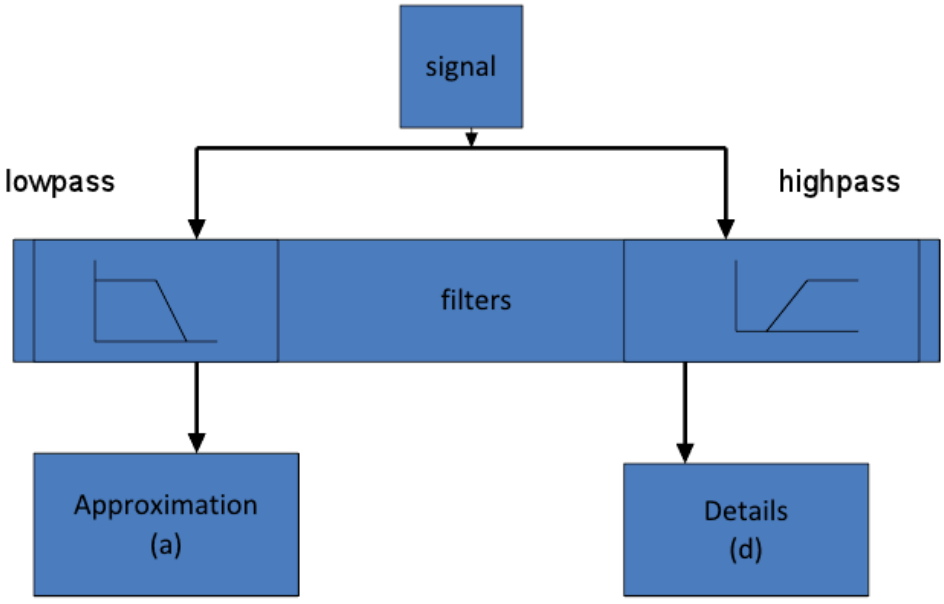

Figure 3. DWT filters.

Multiresolutional analysis (MRA) analyzes the signal at different frequencies with different resolutions. Every spectral component is not resolved equally as was in the case of the STFT. MRA has been designed to give good time resolution and poor frequency resolution at high frequencies and good frequency resolution and poor time resolution for low frequencies. This is very useful in cases where we want to study a signal with high frequencies components for short durations and low frequency components for long duration.

As we said before, to study and perform transform are used several functions that are have a discernable shape. We are going to study in the following paragraphs two examples. One that is very simple and it is the **Haar Wavelets** and a second example that it is the one that it is going to be applied on this project: **Daubechies 10.**

Haar Wavelets:

It is a compact, dyadic and orthonormal wavelet transform. It is the oldest and simplest orthonormal wavelet with compact support. It consists in an odd rectangular pulse pair. Its main disadvantage is that it is not continuous and therefore, it is not differentiable.

The Haar Wavelet is defined as it follows:

$$\psi(t)\begin{cases}1 & 0 \le t < 1/2\\ -1 & 1/2 \le t < 1\\ 0 & otherwise\end{cases}$$

and its scaling function:

$$\phi(t)\begin{cases}1 & 0 \le t < 1\\ 0 & otherwise\end{cases}$$

Although the discontinuity, the Haar Wavelet has several properties that are very useful when we are working with wavelet analysis.

Properties:

- Any continuous real function with real support can be approximated uniformly by linear combinations of $\phi(t)$, $\phi(2t)$, $\phi(4t)$, …,$\phi(2^n t)$, and their shifted functions. This extends to those function spaces where any function wherein can be approximated by continuous functions.
- Any continuous real function on [0,1] can be approximated uniformly on [0,1] by linear combinations of the following functions: 1, $\psi(t)$, $\psi(2t)$, $\psi(4t)$, …, $\psi(2^n t)$, and their shifted functions.
- Haar function is orthogonal:

$$\int_{-\infty}^{\infty} 2^{\frac{n+n1}{2}} \psi(2^n t - k)\, \psi(2^{n1} t - k_1) dt = \delta_{n,n1}\delta_{k,k1}$$

where $\delta_{n,n1}$ represents the Kronecker delta.

- When we apply the transformation, the length of the output and the input is the same.

In Figure 4 we can observe different representations of the Haar wavelet:

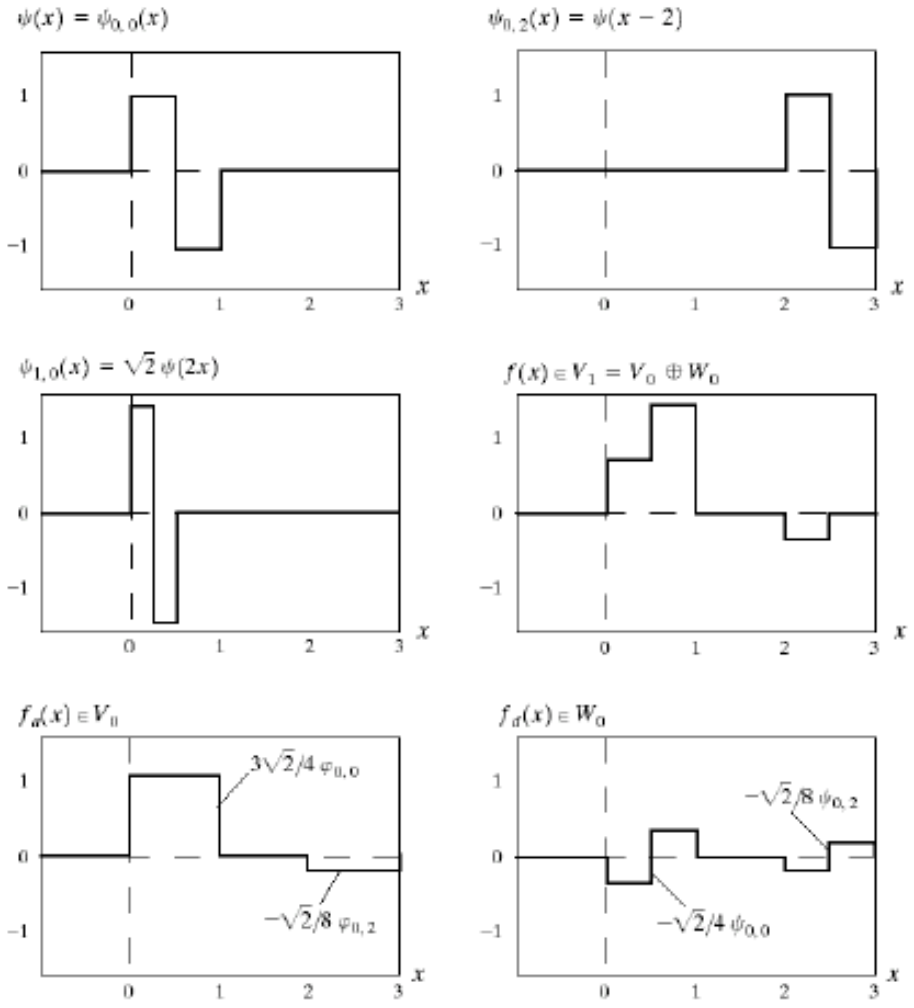

Figure 4. Haar wavelet examples.

Daubechies 10:

Daubechies wavelets are a family of orthogonal wavelets defining a discrete wavelets transform and characterized by a maximal number of vanishing moments for some given support. Depending on each wavelet, it will be used one scaling function, called the father wavelet, which generates an orthogonal multiresolution analysis. The number that follows the function is the number of vanishing moments that this function has. In this case, we will be working with Daubechies 10; thus, our wavelet has 10 vanishing moments.

The following images (Figure 5) show the scaling and wavelet functions and the amplitudes of the frequency spectra of those functions for different daubechies wavelets.

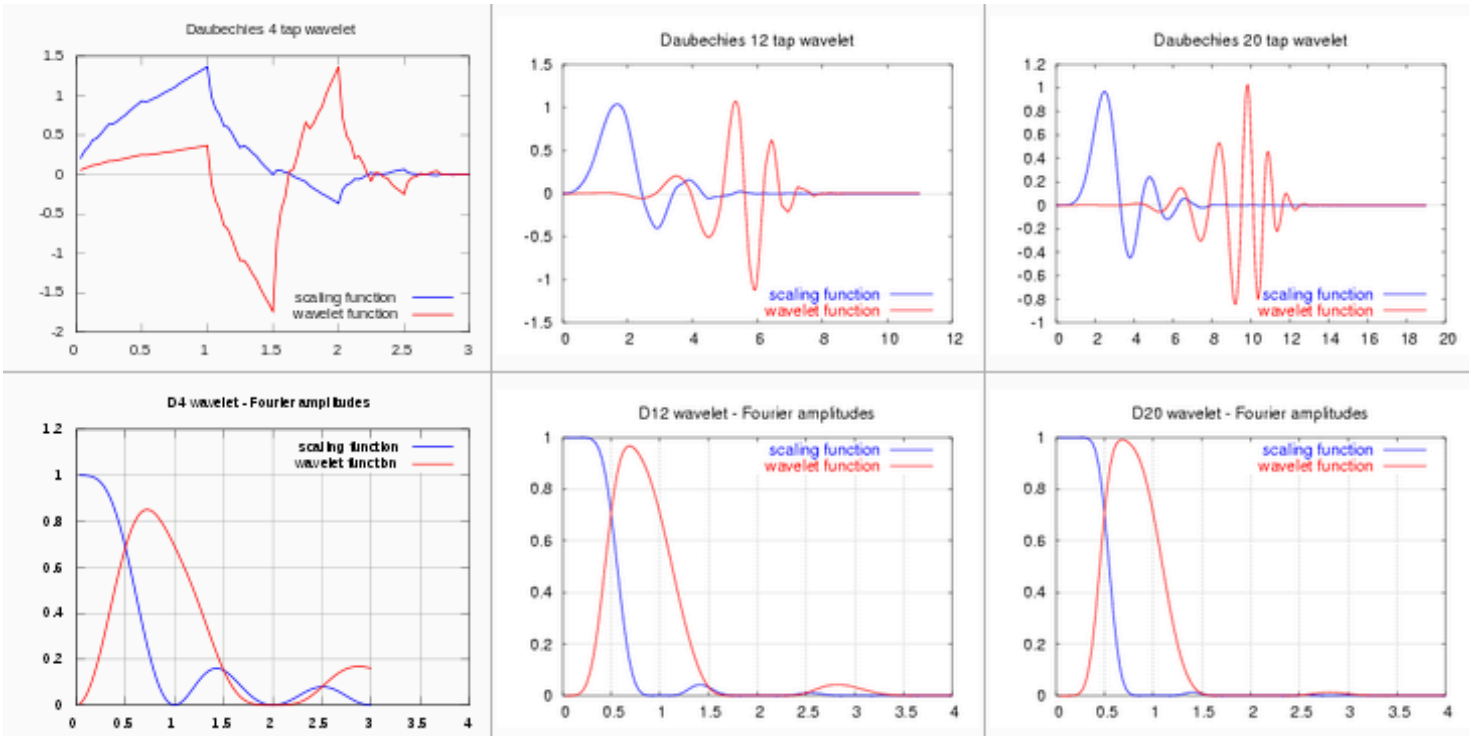

Figure 5. Daubechies scaling and wavelet functions.

As we said before, in our case we are going to work with the Daubechies 10 wavelets because it has been found that it has a better behavior to study ultrasonic signals, the element that is going to be study in this project. Therefore, our wavelet is described in Figure 6:

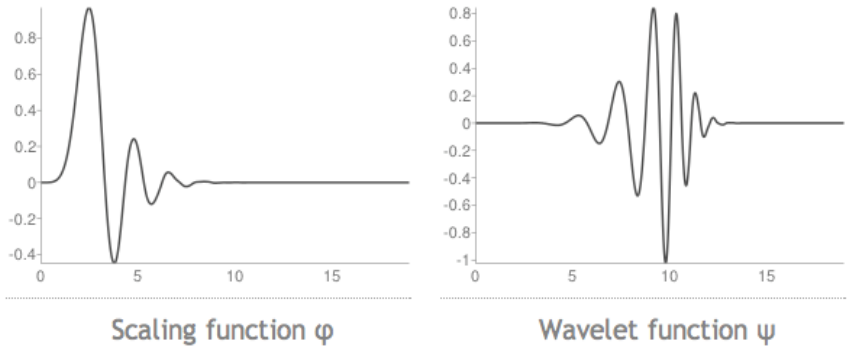

Figure 6. Daubechies 10 scaling and wavelet function

And the coefficients that we are going to work with are shown in Figure 7:

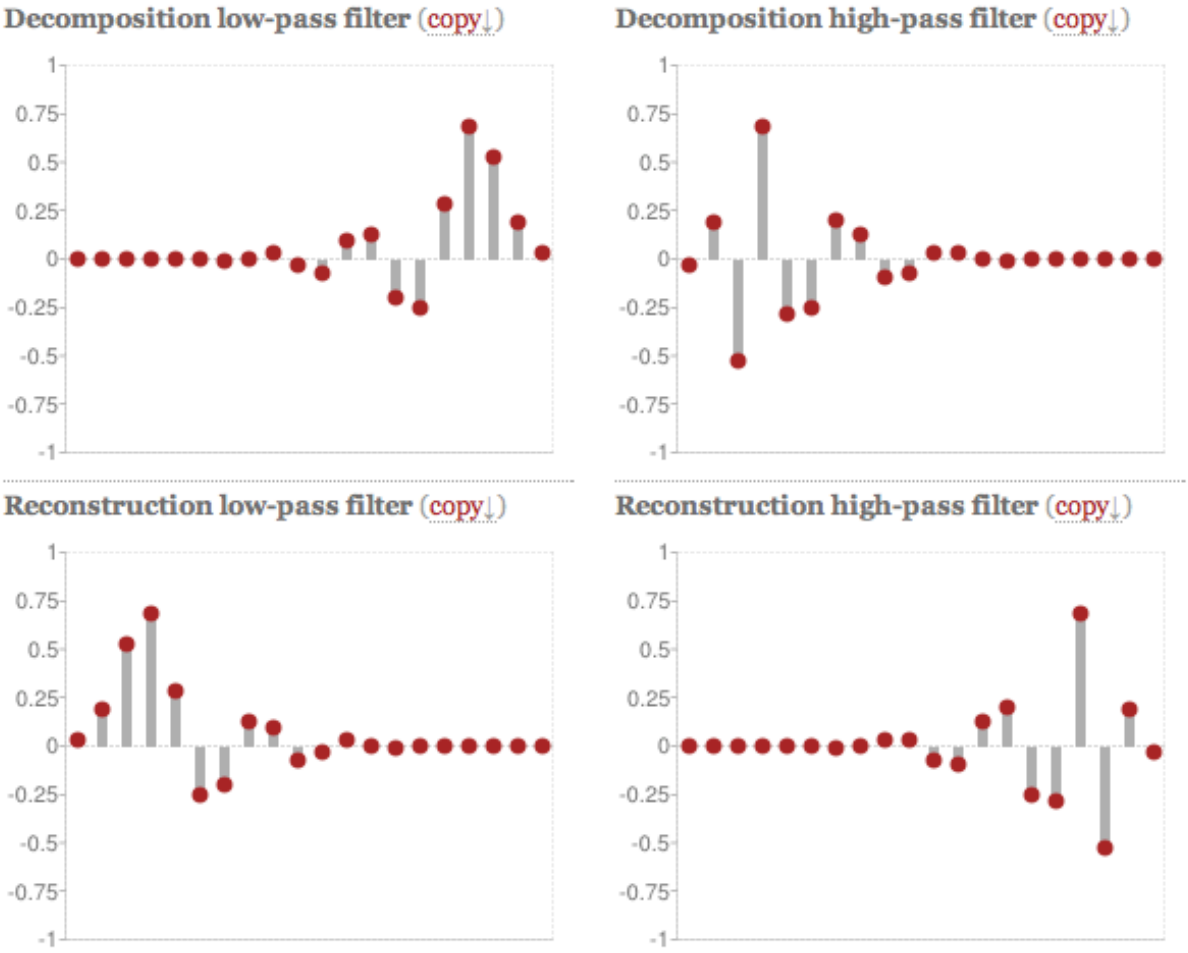

Figure 7. Daubechies 10 coefficients.

Those elements are going to be implemented in the filters to compress the signal and later we are going to implement the reconstructions filters to study the correctness of our design.

Final design:

Once we have understood the previous points (DWT, Haar transform, Daubechies wavelets) we have enough knowledge to start developing our design. The next step that we have to accomplish is to work out the schematic of our design. Our challenge is to filter as much as we can our signal and with all the outputs of the filter, decide which can be discarded because it doesn't give us relevant information or the information provided by that amount of output data is redundant and it is irrelevant. Thus, if we filter the signal several times and we discard few of those outputs, we will be reducing the size of our initial signal.

The following figure (Figure 8) shows the schematic that it is going to be performed to filter the signal:

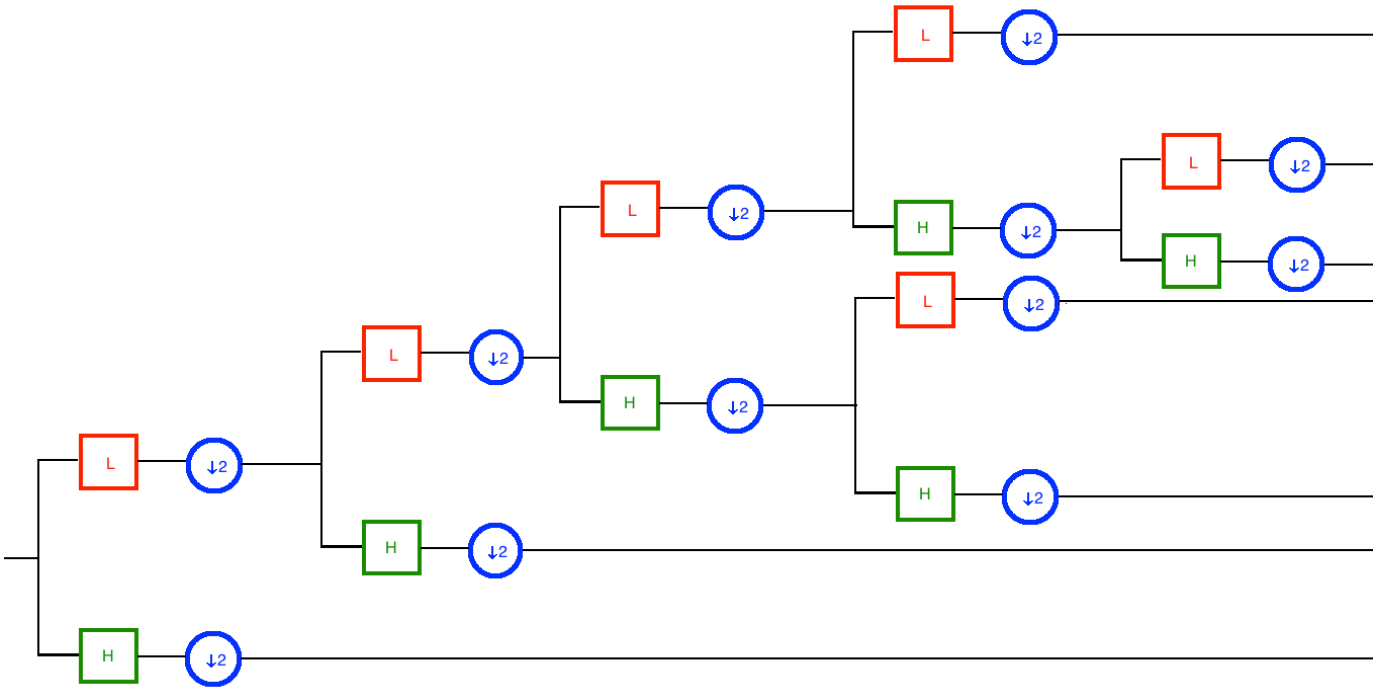

Figure 8. Schematic.

In Figure 8 we can see the different types of blocks and filters that are going to be implemented in our design. The red ones are the low pass filters (L) and the green ones are the high pass filters (H). In addition we have included the blue blocks that are going to be employed to compress the signal obtained from the filters, taking the half of the elements. As part of the implementation, and as addition to the project to test it and study the accuracy of our implementation, we will implement the reconstruction model for our design (Figure 9).

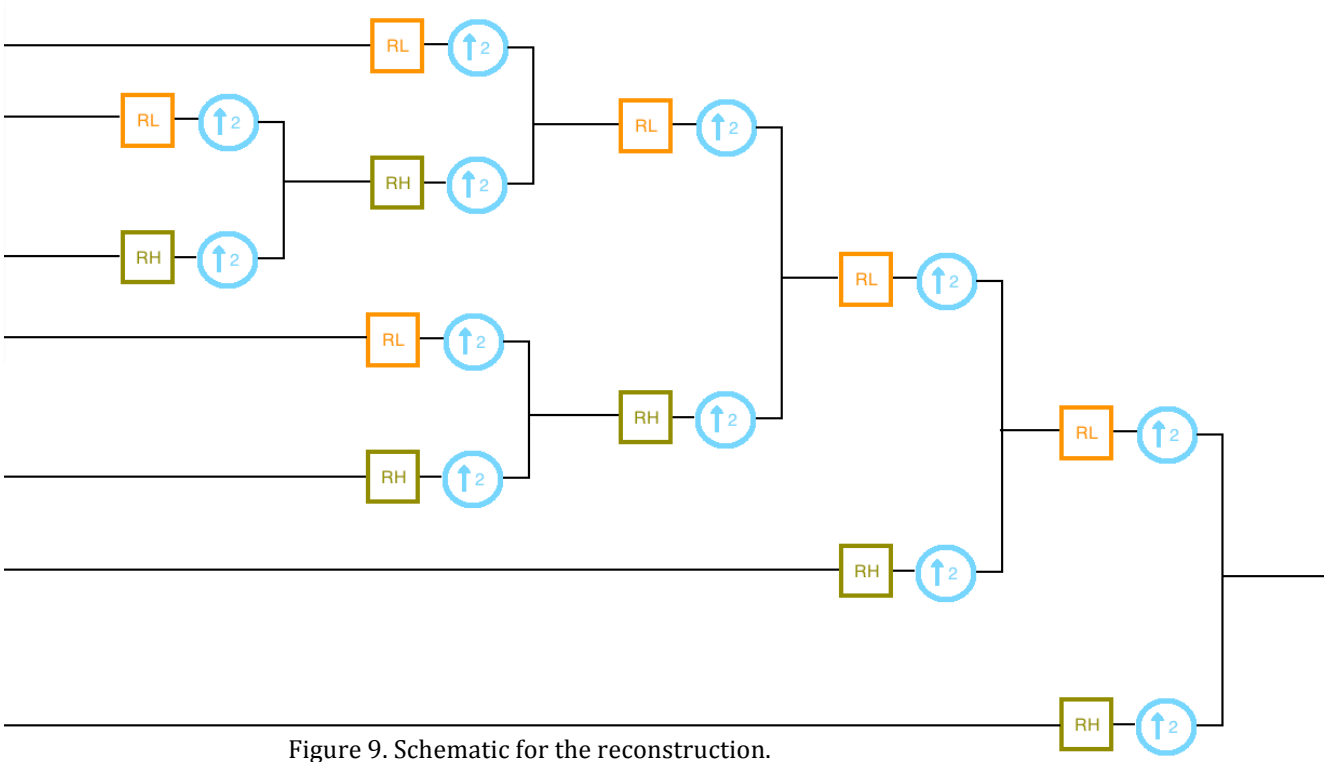

Figure 9. Schematic for the reconstruction.

In this case we appreciate two different types of filter: low pass reconstruction (RL) and high pass reconstruction (RH). Those new filters have an important characteristic: the elements from the reconstruct filters are the inverse from the direct filters. Therefore, the first element of the low pass filter is the last element of the reconstruction low pass filter and so on. The other circular block is to expand the output of the filters. We introduce zeros between one element and the following, so we increase the number of element multiplying the size of the vector by 2. The input of each filter is the addition of the two signals that has been obtained previously.

The reconstruction module will be implemented only in C code to study de correctness of our design. In following steps this module won't be used, instead it will be compared the values obtained from each filter with the C implementation.

Therefore, now we know how do we have to face the project and which are the following steps that we have to follow and the tools that are going to be used. First we will implement the whole design (compression and reconstruction) in C code. Later, once we have understood how to design the filter and the data flow, we could continue developing the VHDL code that will let us implement our design in the FPGA. Finally, our design will be tested in the FPGA, studying the correctness of our design and obtaining several conclusions.

C code implementation.

In this section of the project the main tool that is going to be used is C programing and Matlab. This will be a very complex part of the project where we were looking for the maximum accuracy possible in the implementation. In the following lines we will explain our goals, what kinds of problems we had to deal with, how we resolved them and the results that we obtained in different implementations that where done.

Goals of the C implementation:

The main objective of this part was a complete implementation of the whole design (the schematic showed in Figure 8 combined with the schematic from Figure 9), where we first compress the signal and afterwards that compression is reconstructed to study it. If the reconstruction is very similar to the initial signal we have done a good implementation. On the other hand, if we loose too many values or the signal reconstructed differs too much from the initial values we should redo the implementation because we have made several mistakes.

In order to make it easier and avoid errors we decided to create little modules where we defined a function separately form the rest, so later, in the main corps of the program we will call to all of those functions and execute them to obtain the expected result. Some of these functions are similar. This is the case of the functions responsible of filter the signal, where the main operation are the same but we have to change the values of the filter coefficients.

Functions used:

- `printSignal`: this function has been created to print in console the signal that we want. Its behavior is very simple. It has three inputs: the name that we want to give to the signal, the vector that we want to print in console and the length of this vector.

- `convolve`: in this functions it has been implemented the convolution. This is one of the most important elements of this project because it is the one that let us to filter the signal. It has several inputs: the signal that is going to be filter, its length, the filter coefficients and the filter's coefficients length. The output is called *result* and it will be the input for following filters. The idea of this block is to multiply each value of the signal by the correspondent value of the filter and add that value to the cumulative sum. Then we will obtain a vector which length will be the length of the signal plus the length of the filter coefficients (this one is constant, it will be always 20) and we have to subtract 1 because the convolution.

- `DWTHigh`: this block has several declaration inside it. The main idea is that it takes the signal (it is an input) that we want to filter through the high pass filter and generate a new vector that has 38 components more than the input signal vector. These 38 more components are 19 at the beginning and 19 at the end, and their value is not random, they are mirrors of the input signal. The value 19th will be equal to the 1st value of the signal, the 18th equal to the 2nd, so on. After these first 19th values of this new vector we will have the input signal, and the last 19 values we will do the same. The 1st of these last 19 values will be equal to the last value of the input signal and so on. After this first step we will call to the function `convolve` and then realize the convolution. It is remarkable to see that the filter coefficients are defined in this this block. Of course, the coefficients for a high pass filter. Once we have filter the signal we are going to cut the first 20 elements and the last 20. It is necessary to cut the output of the filter because we have to remember that the signal has been extended to perform the convolution, so once we have convoluted the signal and filter it, we have to discard the elements that has been included for a proper implementation. Once we have the signal without the elements discarded we have to compress it as it shows Figure 8. We pick one value and the next value is discarded, the following again is taken and the next discarded and so on. Actually this is done in one loop (Figure 10):

```
int i;
for (i = 0; i < (SignalLen + 19)/2; i++ )
{
    resultHcomp[i] = resultHigh[20+2*i];
}
```

Figure 10. Cut and compression of the signal filtered.

- `DWTHighRecon` : in this block we are going to implement the reconstruction for the signal that has been driven through high pass filters, in Figure 9 we can observe that we are going to use this block several times, in particular the same number of times that we have used the high pass filter. First of all we are going to initialize a new vector where we are going to introduce the input signal, but as in the previous block (`DWTHigh`), in a specific way: one value of this vector is from the input signal and the following is a '0'. Thereby, we are expanding our signal, increasing its length multiplying it by 2. Afterwards we convolve it with the coefficients for the high pass reconstruction filter. Then we have a vector which size doesn't match with the supposed length that it should have the output of this block. Therefore, as in the rest of the filters, we have to cut the signal obtained after the convolution. We are going to use two variables: '`first`' and '`last`'. To calculate '`first`' we calculated the integer part of the result between the length of the signal expanded and included the 19 values of the convolution after been divided by 2. The value of `last`' has been calculated after the length of the signal that has been used for convolution is subtracted by the round up of the previous value, with this two signals we are able to print in console the output signal after been cut (remember that we had to expand it

to generate an accurate performance of the convolution). After all the previous steps we have explained, we have the signal reconstructed.

- `DWTLow`: this module is similar to `DWTHigh` but the coefficients are different because now we are in the low pass filter. All the steps and implementation used in this module are the same that has been implemented in `DWTHigh`.

- `DWTLowRecon`: as it happens with `DWTLow`, this module is exactly the same as `DWTHighRecon` except for the coefficients that has been used, that are different from filter to filter.

- `DWT`: in this module we call to `DWTHigh` and `DWTLow` functions and the we obtained the whole filtered (low pass and high pass) of the signal. This will help us in the main function because we just have to call to one function instead of two (`DWTHigh` and `DWTLow`).

- `IDWT`: as the previous function, in this module we are going to call the two functions that reconstruct the signal, `DWTHighRecon` and `DWTLowRecon`. As in `DWT`, in this module we call this two function obtaining the final result reconstructed.

- `main`: this is the main part of the program. In this block we are going to call the previous functions and control the data flow of the signals. The first thing that we are going to do is to declare the initial signal, the one that has been obtained after several experiments and that we want to filter. Afterwards we are going to determine the signals lengths for the several steps that we have to implement and we are going to give different names to the signals obtained after each stage (L, H, LL, LH, LLL, ...). Finally, we are going to call the function in sequence to obtain the multistage that we want to perform. In this module it has been implemented a last function to create a .txt file so we can take the output and compare it with the original values in MatLab so we can study the correctness of our implementation.

Problems in C implementation:

During the C implementation we have to go step by step in order to avoid future problems. However, we had some problems and we had to deal with them to obtain an accurate implementation. The final result shows that all the little troubles that we had along the project were solved in the best possible way.

The first problem we had was the convolution. We had a problem with the lengths of the signals, in particular with the output. This problem was observed when we decided to implement stage by stage, first only one stage, then two stages and so on. We show that because the convolution, when we divided by two to compress the signal after been filtered, depending on if it was even or odd we

could lose a value. Of course, in the multistage implementation we lost more values, specifically 13 values. This first approximation was good as approximation, but we need a more exact method. Therefore we learn how does MatLab build its convolution function. At the beginning we didn't use the 38 values that we include 19 at the beginning and 19 at the end of the signal. This new implementation let us to perform a quite exact implementation of the convolution so we don't lose now any value.

In addition at the very beginning, when our implementation was in its first steps, we had several elements which value was very big, we are talking about values of 10^7 when we are working with values less than 100. With the introduction of the 38 values we don't have to worry about those errors because they will appear in the values that will be cut later.

The third problem was concern about how extract the results from the program. We found that the best way to test our results was to compare the reconstruction with the original signal on MatLab. Therefore we had to find a way to convert it to .txt and then extract those values from the .txt in MatLab. Thus, we decided to create a few more code lines where we generated a .txt (Figure 11):

```
FILE *filePtr;
filePtr = fopen("./User/Desktop/reconSignal.txt","w");

int t;
for (t = 0; t < ELEMENT_COUNT(signal); t++){
    fprintf(filePtr,"%.3g\t,", reconSignal[t]);
}

fclose(filePtr);

return 0;
```

Figure 11. Function to convert result to .txt

Results from C implementation:

We have done several implementations. We started with only one stage, then we designed two stages (only in the low pass filter path), later we implemented the three stages design (again only in the low pass filter path) and finally the multistage was implemented. In each of the cases we obtained the results from the reconstruction that was compared with the original in MatLab, so we could study the results obtained. In addition, in the last implementation (multistage) we obtained the values of the compression, in other words, the outputs from Figure 8.

In the following figures we are going to show the results obtained in each case and explain how good was the implementation.

- **One stage implementation:** in this first case we are going to implement only one stage. Figure 12 shows the schematic of this first case.

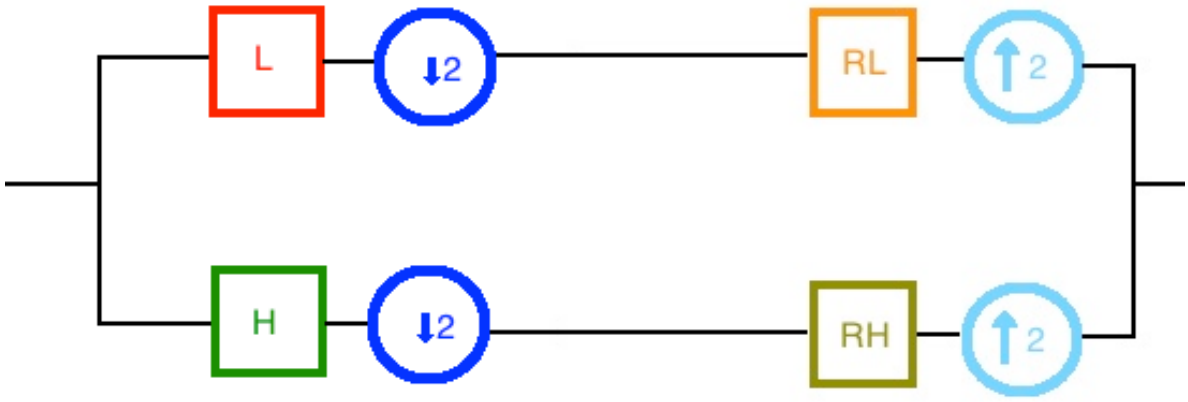

Figure 12. One stage schematic.

The results obtained in this case are shown in the following figures. Figure 13 is shows the original signal and the reconstruction. In Figure 14 we can see the error obtained between both signals.

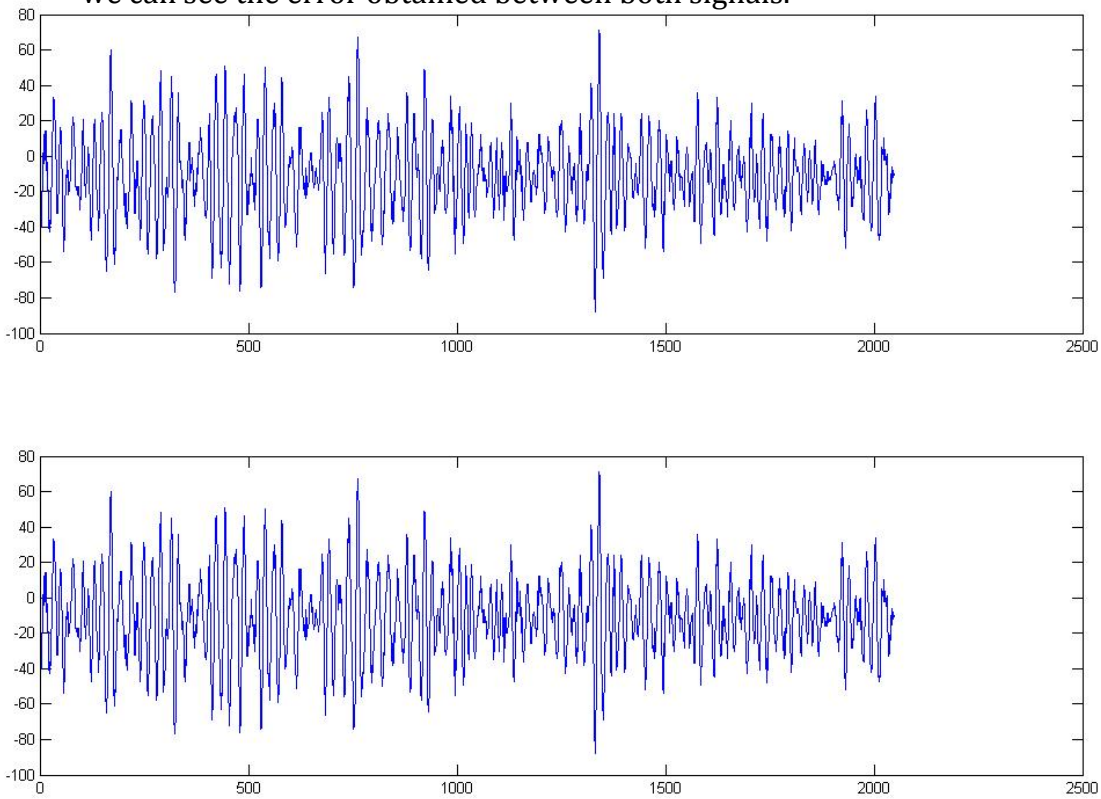

Figure 13. Original signal and the reconstructed. One stage.

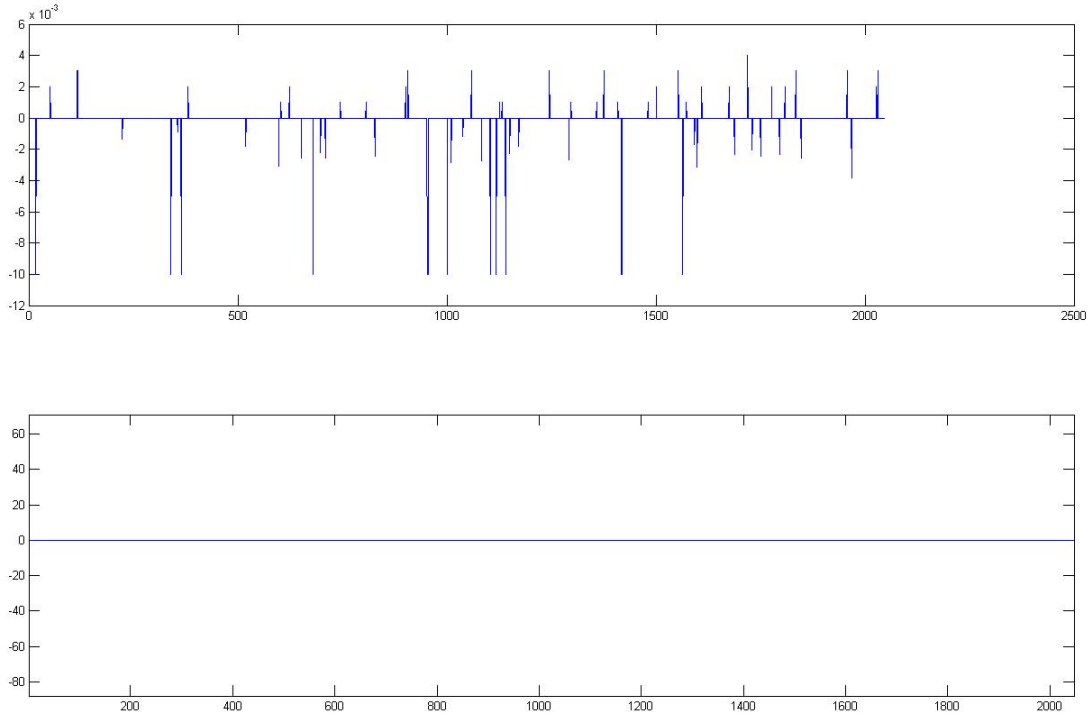

Figure 14. Error obtained between reconstructed and original signal.

We can observe that in this first implementation that the difference between the original and the reconstructed signal is almost imperceptible (Figure 13). In addition, if attend to Figure 14 we can see that the highest error is 0.01, 1% from the initial signal.

- **Two stages implementation:** the schematic for this case is represented in Figure 15.

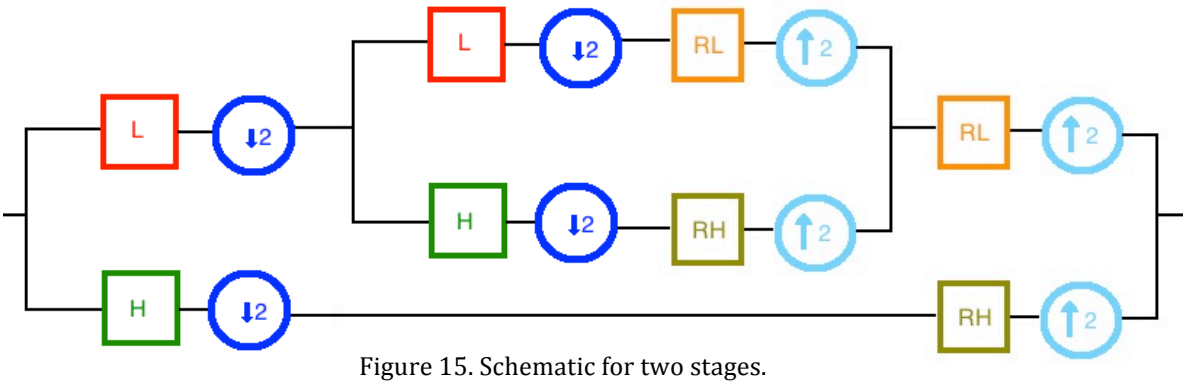

Figure 15. Schematic for two stages.

The results obtained in this second case are shown below. As in the previous case, Figure 16 will show us both signals plotted. In Figure 17 has been plotted the error obtained between both signals.

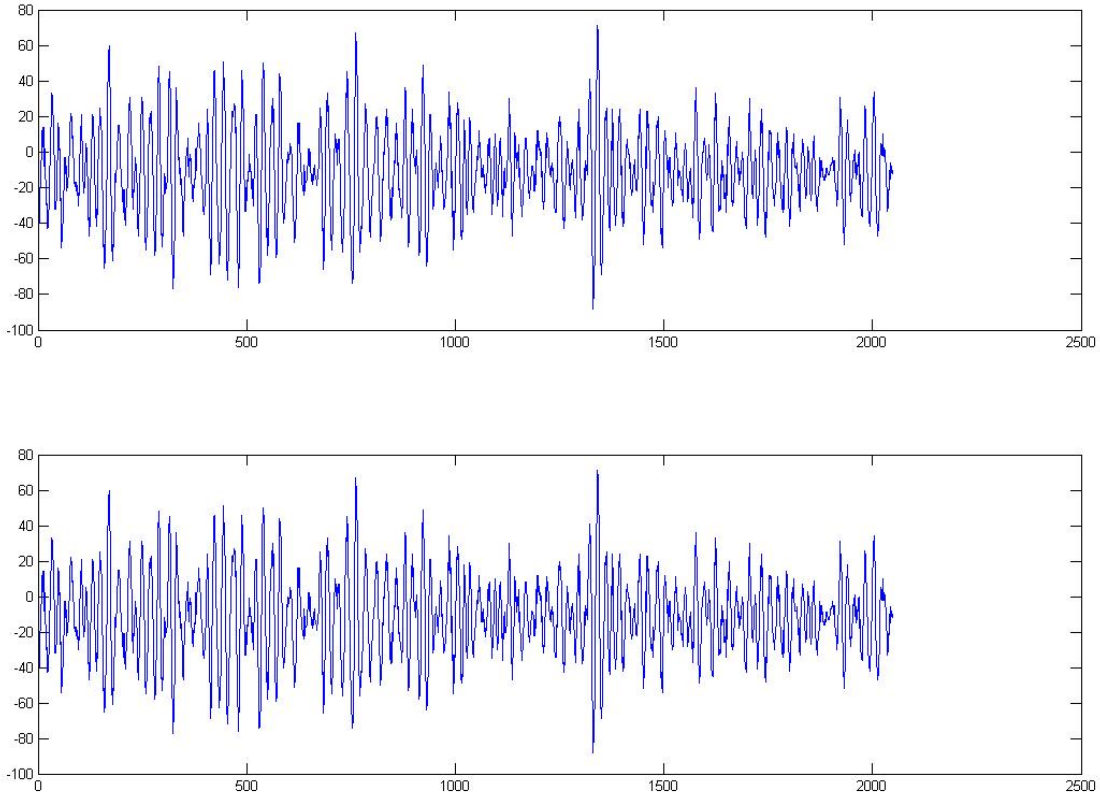

Figure 16. Original signal and the reconstructed. Two stages.

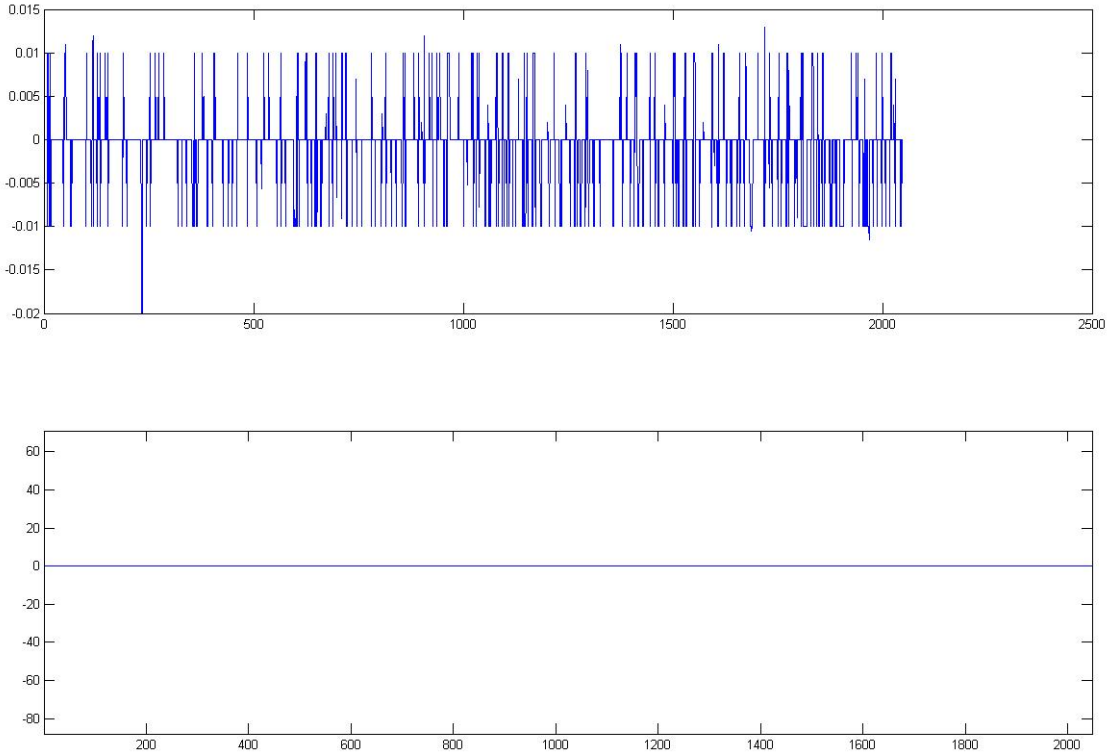

Figure 17. Error obtained between reconstructed and original signal.

As in it happened in the previous case, the difference between the original and the reconstructed couldn't be appreciate. Although, when we study the error figure (Figure 17), we see that in this case the error is bigger. This is because the convolution that generates several errors that when they are convoluted again (second stage), they become bigger.

- **Multistage implementation:** the schematic of this case is the one showed in Figure 8 combined with Figure 9.the results for this very last implementation are going to be shown in the following images. Figure 18 is a representation of both, original and reconstructed signal. Meanwhile, Figure 19 shows the error obtained between original signal and the one that has been reconstructed.

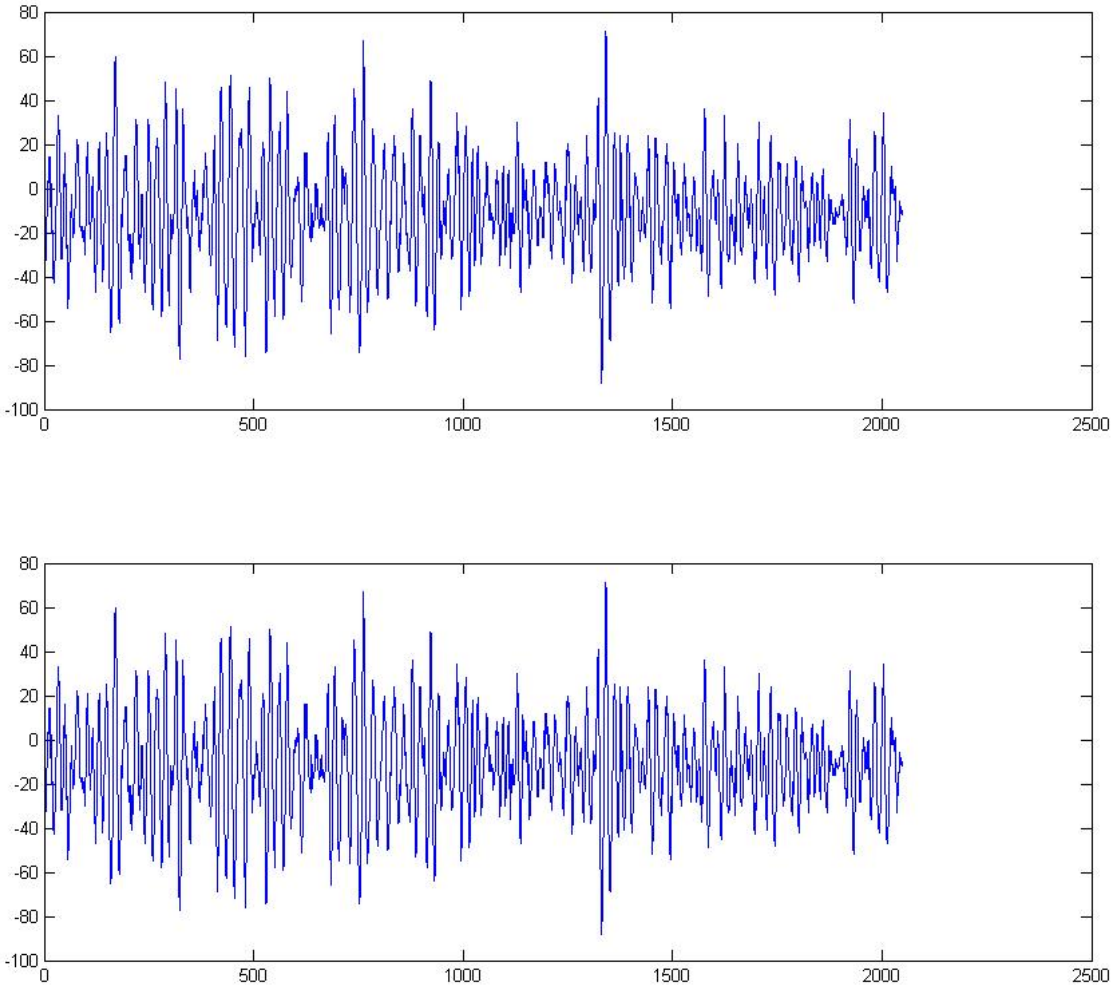

Figure 18. Original signal and the reconstructed. Multistage.

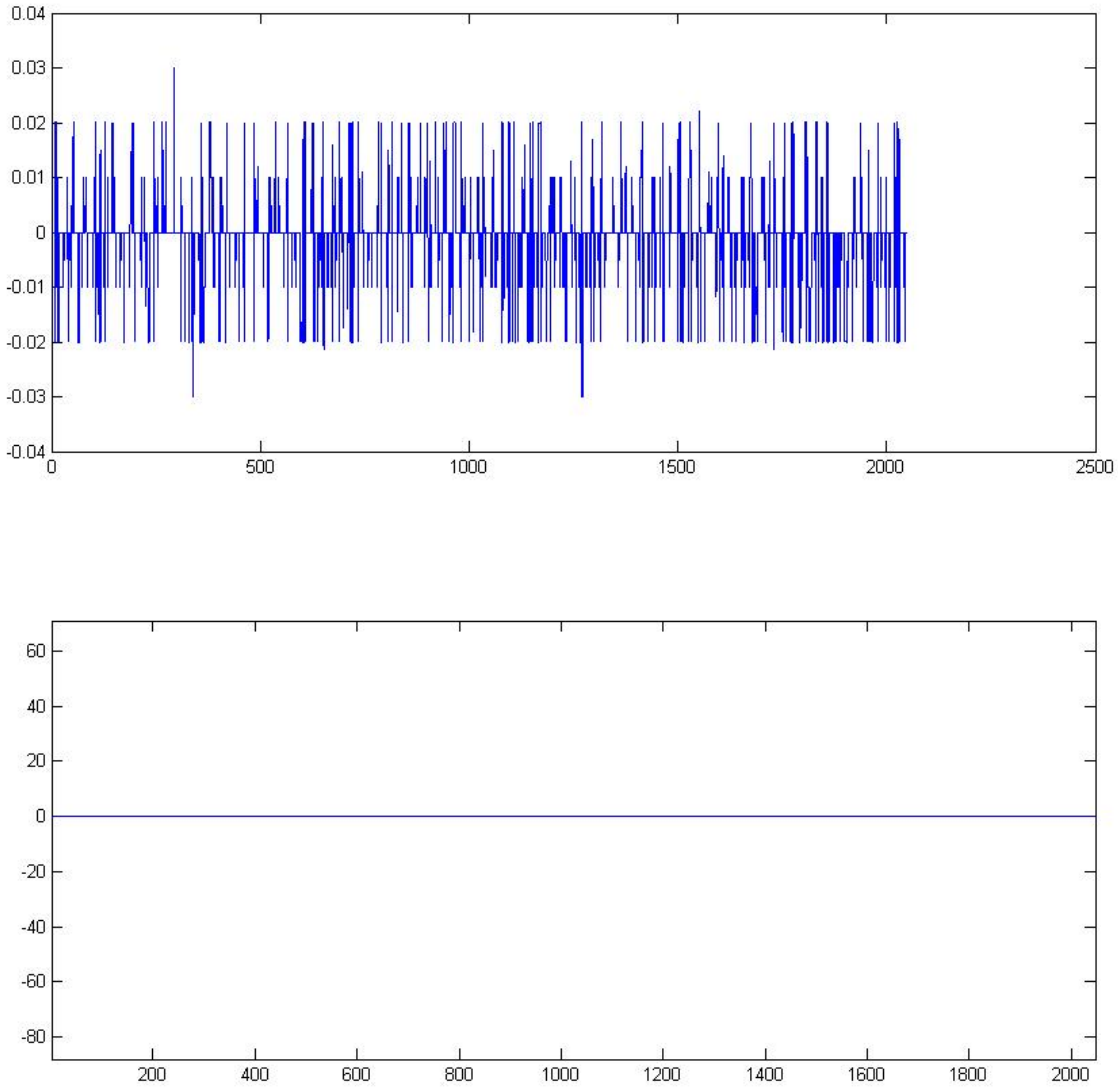

Figure 19. Error obtained between reconstructed and original signal.

As in previous cases, the error has become bigger because the number of convolutions that we have to implemented to design the multistage. However, when we compare the error obtained with the values used (the second graph from Figure 19), we see that again the error obtained is negligible.

In conclusion, attending to the results obtained, we can assert that the design that has been implemented performs a very accurate decomposition of the signal in its different parts (remember that filtering we divided the signal in details and approximations). Therefore, our implementation is as good as the one that MatLab implements in its design.

To calculate the error in MatLab we implemented the following commands, so we were able to calculate the value of the error for several cases as our project requested.

```
C=fopen('reconSignal2Stage.txt');
b=textscan(C,'%f', 'delimiter', ',' )
B = b{1};
subplot(2,1,1),plot(grain)
subplot(2,1,2),plot(B)
>> for i = 1:2047,
p(i,1) =   abs(grain(i,1))-abs(C(i,1));
end
subplot(2,1,1),plot(p),subplot(2,1,2),plot(p)
axis([1 2047 -88 71])
```

The first segment of the code let us to plot the original signal (grain) and the one obtained after the C code filtering. The second segment of the code is how we have calculated the error of the signal and afterwards it has been plotted the error value in two graphs, the first one showing the error and the second one showing the relative error between the value of the error and the highest absolute values.

VHDL implementation.

Once we had the C implementation we wanted to continue developing our project to finally implement it on a FPGA, so the next step was VHDL implementation. This step was very big because the language was completely different and we had to design the number of memories used and how activate each action depending on a bus register that will be active from the microprocessor. Therefore, change from a programming language to the other was a big step that included several changes in our code. It is important to notice that in VHDL design we didn't implement the reconstruction of the signal.

In this implementation we have defined a new type of vector to operate with it. The subtype's name is 'float'. With this subtype we can store in a standard logic vector a real number.

For VHDL we decided to create different modules, each one with a specific task. There were implemented 3 different modules: one for high pass filter, one for low pass filter and the third one was a final state machine that allows us to activate signals when there is any particular condition.

Goals of VHDL implementation:

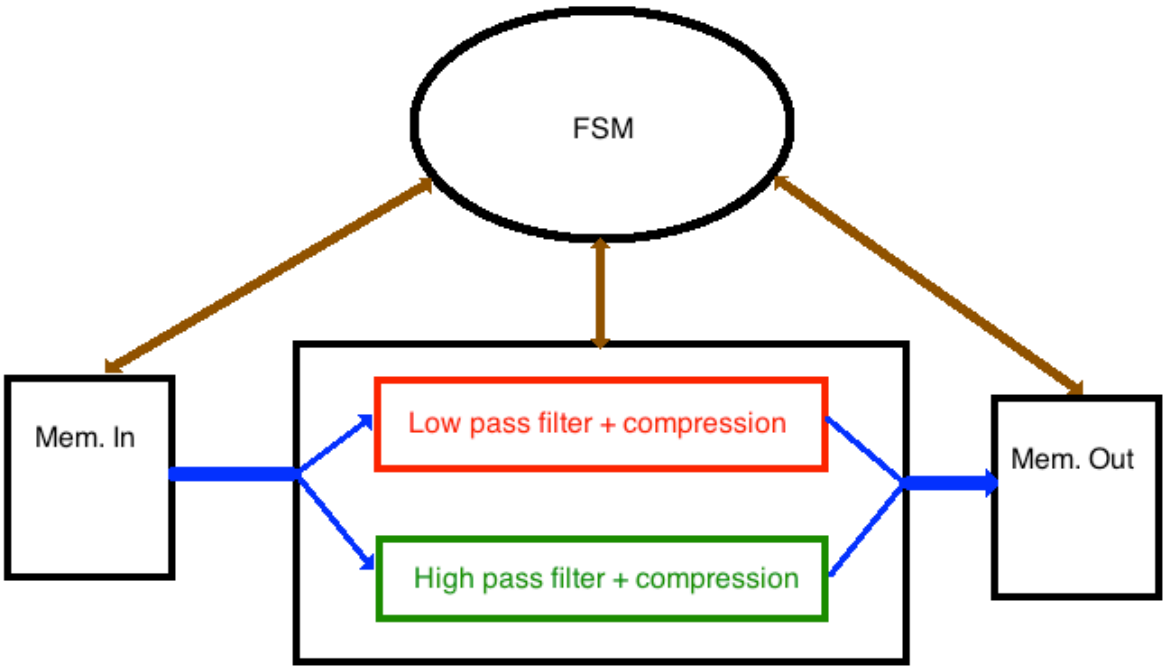

Figure 20. VHDL schematic.

The VHDL implementation was a challenge for us and it supposed a great experience in hardware programming and design. We focus ourselves in one stage implementation. For this implementation we needed several modules with

particular attributes. In Figure 20 we can see a schematic of our VHDL implementation.

In Figure 20 we can see all the elements that has been created for a correct implementation of one stage filtering. We can see the Mem. In, where all elements are stored at the beginning, Mem. Out, where the values obtained after filter the signal are stored, the FSM that will control the data flow depending on the inputs that it receives and a central corpse where are implemented both filters. It is important to notice that the blue threats work in only one direction (Mem. In to filters, filters to Mem. Out), meanwhile the brown threats work in both ways. This is because we need to send data form each element to FSM so different signals will be activated to continue with the logical process of the implementation.

Modules used:

- FIRLow, FIRHigh: These two modules perform the convolution of the signal with the coefficients of the DWT, either high or low depending of the module. The structure of the module is based on the definition of the FIR filter, using a bench of arrays as delays z^{-1}. These modules are synchronous, so the output is one point per cycle, and with an enable input signal.

- Compress: The aim of this entity is to compress the signal giving an output for every two inputs, due to the circular convolution there is a need to discard the first 19 input points and the last 19 points, so this module ensure that this operation is performed. Also synchronous, and with an input signal that controls if the module is working. There is an output signal nxtelement that goes up when the module has taken the 20 input, this is needed because the final state machine FSM will know when to star writing the output memory.

- DWTHigh, DWTLow: This entity encapsulates the modules FIRHigh, FIRLow and Compress into one final model.

- FSM: The final state machine that controls the sequence of the behavior of the different modules, this entity has this different states:

    1. S1: Starts reading the input memory. Initialize the values. Goes to S2.
    2. S2: Enables the filter, so the convolution starts to be performed, also the address of the input memory is refreshed (+1). Goes to S3.
    3. S3: Enables the compressor. Refresh the address of the input memory. Goes to S4 if nextelement ='1', if not goes to S3a.
    4. S3a: Refresh the address of the input memory. Goes to S4 if nextelement ='1', if not goes to S3.
    5. S4: Starts writing in the output memory, refresh the address if the input memory also the output memory. Goes to S5 if the address of the input memory reaches 2047, if not goes to S4a.

6. S4a: Refresh the input memory address, but not the output because the compressor will only give a point every two cycles. Goes to S5 if the address of the input memory reaches 2047, if not goes to S4.
7. S5: Stop reading the input memory, and refresh the output memory address. Goes to S6 if the address of the output memory reaches 1032, if not goes to S5a.
8. S5a: State introduced just to control when the output address will be refreshed.
9. S6: Activates the register control signal to write in it, so the program knows that the DWT has finished.

- Userlogic: This is the final implementation of the whole system that will perform the DWT in the FPGA, with all the connections between the different modules as the Figure 20 shows. This module will contains two memories:

    1. Input Memory: In this memory the processor will write the data to be processed, with one input data, and enables and another output data.
    2. Output Memory: This memory will contains the results of the DWT, with two input data, the DWT high and DWT Low will be done at the same time, also has an output data signal where the results will be read.

- For every module a test bench has been done, in order to ensure that every component of the DWT works properly:

    1. DWTtb
    2. FSMtb
    3. Firhtben

Results of VHDL implementation:

In this section we are going to show the testbenches that were written to test the correctness of each module and of each design. It is important to remark that in this case it wasn't implemented the reconstruction part of the design. In this case, for VHDL, it was implemented only one stage, but the modules, except for the main, would be the same in case of we had decided to implement several stages.

- **FSM:** the final state machine was explained before, so in this section we will only show the results obtained after had designed the module and the testbench. Figure 21 shows the first transition between states. The most important is the change between state 3 and state 4, because the value of nextelement.

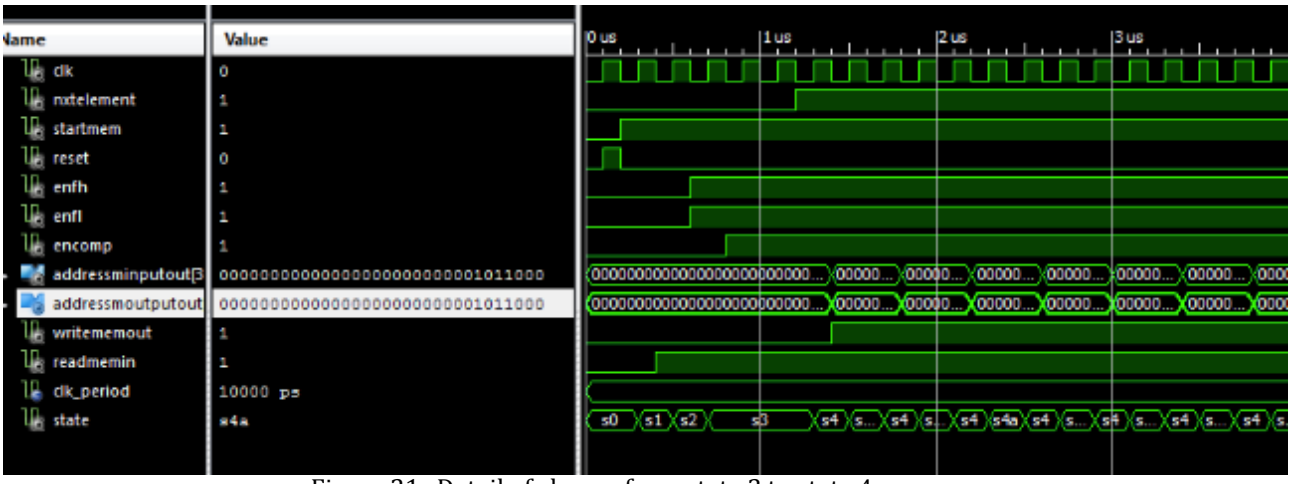

Figure 21. Detail of change from state 3 to state 4.

In the previous image we can see that the FSM does every transition when it has to be done, including the transition from state 3 to state 4 when nextelement is equal to '1'.

In the following figure (Figure 22) we will see the transition from state 4 to state 5 when the addressminputout has reached the value of 2048, the number of elements that it has to be read for the first stage of the filtering.

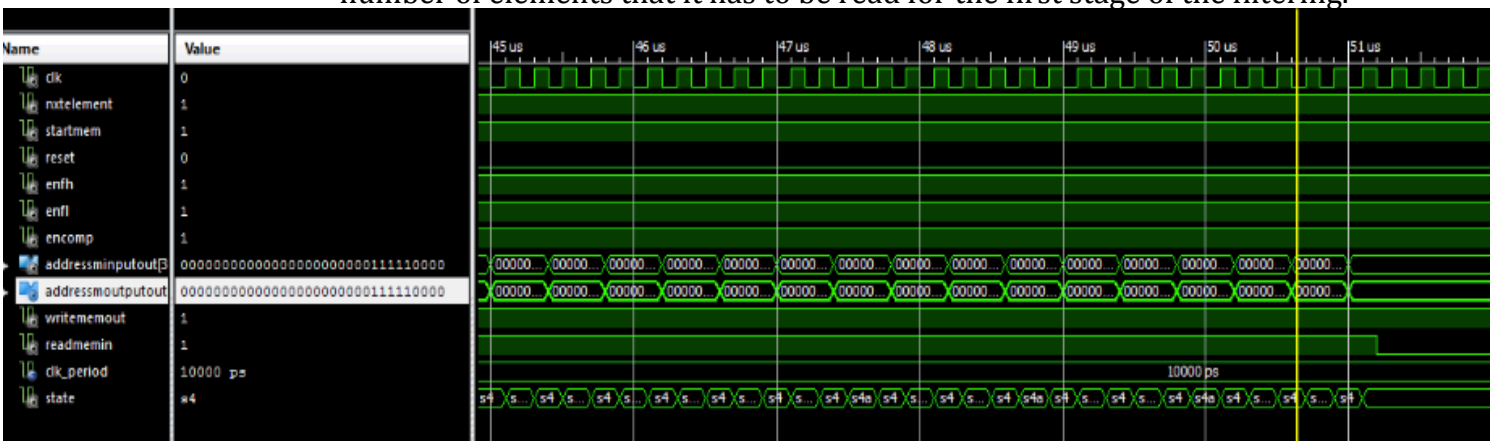

Figure 22. Detail of change from state 4 to state 5.

- **Compressor:** in Figure 23 we can see that the first 20 elements are not compressed because, as we explained below, we have to include 19 elements at the beginning and 19 at the end to obtain an accurate convolution. Then, once the 19th element has been discarded, we start taking elements to begin with the compression, taking only the even elements and discarding the odd ones.

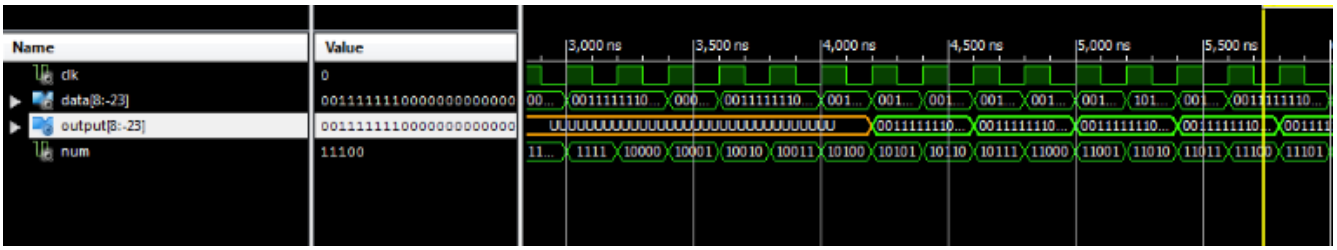

Figure 23. Detail of compression modules.

- **Filters:** in this case we are including both, high pass filter and low pass filter. In Figure 24 we can see how the filter works, in both case the behavior of the filter is the same and the only difference between both of them is that the filter values depend from filter to filter.

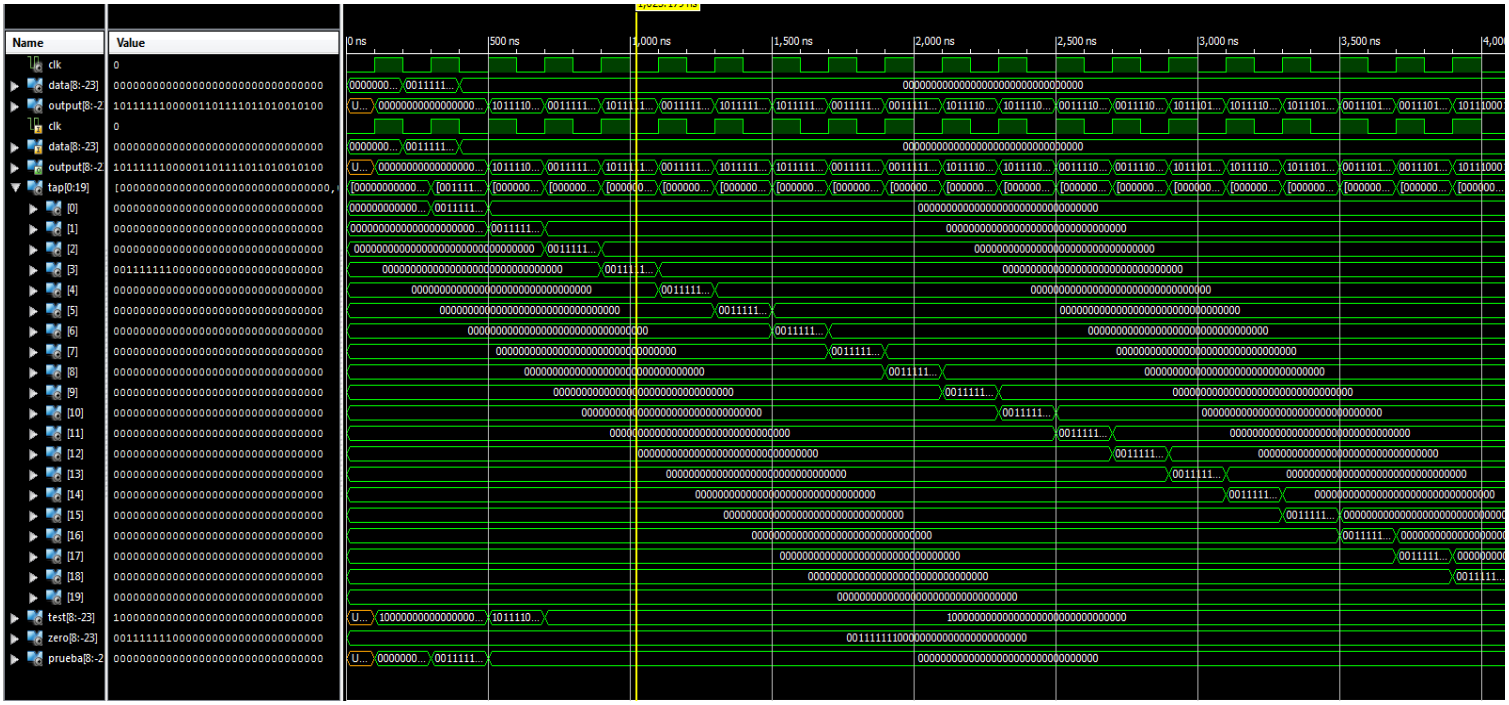

Figure 24. Detail of filters modules.

In the previous figure we can see how the data goes from element to element each time we have a new edge clock equal to '1'. And the output, because we decided to put a '1' at the input, is the values of the filter. Therefore, the implementation of the filter has been tested and it works perfectly, with no errors.

Conclusion.

In all the previous pages we have covered several topics: we first learned what a DWT is; then we decided to learn some about the most used filters and the one that would be implemented in this project; later we started with one way to implement the design, C implementation, and finally we learned how to implement it into VHDL.

Both ways of implementation where tested and we saw that the results obtained where very close to the ones that should be obtained in a real implementation. It was implemented in different ways, starting at the very bottom of the project, beginning with only one stage, and then going step by step making it more difficult. The results obtained in the case of the C implementation were very close to the initial values so they were expected to be as good as they finally were. In addition it was shown that it is possible to implement a design in C in the same way that it is done in MatLab.

With the VHDL implementation we had several problems. We had to face a very new language for us and it was a great challenge for us. In this case we decided to implement only one stage, and only the compression, avoiding the reconstruction that will have taken us a lot of more work on the project and it was useless because to test the correctness of the result we could just see the values of the one stage implementation in C.

Future work.

Once we have finished this project we noticed that it could be done more work continuing the line of work developing better functions and implementing more cases. Of course, all implementations are able to enhancements, and there is no implementation that would be the best. Depending on what we want, faster, more precision, etc, we have a bunch of implementation that will match the implementation required.

One first step for future work, and this would be very worthy; it could be implement the VHDL design on a FPGA. This step was tried but because several factors it wasn't able to perform a correct implementation on the FPGA. This step is very important because will let us to test our implementation in a real element. Because all the test that were done in VHDL (testbenches) we expect that in FPGA it shouldn't be a big deal and there wont be so many errors, so the implementation should be accurate enough. But, because we ran out of time, we couldn't implement it on the FPGA.

Other way to keep on with this project would be the implementation of the multistage design. At the beginning we started the design for only one stage. However, we were thinking about how it could be possible to implement it in a multistage. Of course, it would be more difficult to implement it because the usage of memory. Therefore we become to an idea: in the output memory we will only write the output of the filters that are not going to filter again (this signal are the ones that are output of the high pass filter in general, see Figure 8 for more details), and rewrite in the input memory the values that are going to be filtered again. With this implementation we should store the memory of the last position written in output memory and keep on writing since that position, and store the number of elements that has been rewrite in input memory (remember, each time we filtered we reduced the number of elements) so we will know how many elements are going to be filtered in following steps. This idea could be a possible solution for this step.

References.

1. Wikipedia.
2. www.wavelets.pybytes.com/wavelet/db10
3. The wavelet tutorial. By Robi Polikar.
4. Discrete Wavelet Transform. By Pramod Govindan.
5. FPGA implementation of Daubechies Polyphase-Decimator filter.
6. Design and FPGA Implementation of four orthogonal DWT filter banks using lattice structures.

Appendix.

1. C Code for one stage:

```
#include <stdio.h>
#include <stdlib.h>
//#include <stddef.h>
#include <math.h>
#include <time.h>

#define ELEMENT_COUNT(X) (sizeof(X) / sizeof((X)[0]))

#define ITERATIONS 100

void printSignal(const char * Name, float * Signal, size_t
SignalLen){
	int i;

	for (i = 0; i < SignalLen; i++)
	{
		printf("%s[%d] = %f\n", Name, i, Signal[i]);
	}
	printf("\n");
}

void convolve(const float Signal[], int SignalLen, const float
filterDB10[], int filterDB10Len, float Result[])
{
	int n;

	for (n = 0; n < SignalLen + filterDB10Len - 1; n++)
	{
		int kmin, kmax, k;

		Result[n] = 0.0;

		kmin = (n >= filterDB10Len - 1) ? n - (filterDB10Len -
1) : 0;
		kmax = (n < SignalLen - 1) ? n : SignalLen - 1;

		for (k = kmin; k <= kmax; k++)
		{
			Result[n] += Signal[k] * filterDB10[n - k];
		}
```

```
	}
}

void DWTHigh( const float signal[], const int SignalLen, float
resultHcomp[]){
	float filterDB10High[] =  { -0.0267, 0.1882, -0.5272, 0.6885,
-0.2812, -0.2498, 0.1959, 0.1274, -0.0931, -0.0714, 0.0295,
0.0332,-0.0036, -0.0107, -0.0014, 0.0020, 0.0007, -0.0001, -
0.0001, 0.0000};
	int p;


	const int m = SignalLen+1;
	const int n =  SignalLen +38;//even
	const int q = m + 38;//odd
	const int o = q + 20 - 1;//odd
	const int r = n + 20 - 1;//even
	float x[n];
	float resultHigh[r];

	//Lenght test.
	//printf("Tamano resultante de m = %d\n",m);
	//printf("Tamano resultante de q = %d\n",q);
	//printf("Tamano resultante de n = %d\n",n);
	//printf("Tamano resultante de o = %d\n",o);
	//printf("Tamano resultante de r = %d\n",r);
	//printf("Tamano resultante de Elements Filters  =
%ld\n",ELEMENT_COUNT(filterDB10High));

	for (p = 0; p < (SignalLen + ELEMENT_COUNT(filterDB10High) -
1)/2; p++)
	{
		resultHcomp[p] = 0.0;
	}

	for (p = 0; p < 19; p++)
	{
		x[p] = signal[19-p-1];
		x[p+19+SignalLen] = signal[SignalLen-1-p];

	}

	for(p=0; p < SignalLen; p++){

		x[p+19] = signal[p];
```

```
	}


	//printSignal("x", x, n);
	//printf("\n");



	convolve(x, ELEMENT_COUNT(x),
		    filterDB10High, ELEMENT_COUNT(filterDB10High),
		    resultHigh);
	p = ELEMENT_COUNT(resultHigh);
	//printSignal("resultHigh", resultHigh, p);
	//printf("\n");

	int i;
	for (i = 0; i < (SignalLen + 19)/2; i++ )
	{
		resultHcomp[i] = resultHigh[20+2*i];
	}

	//printSignal("resultHcomp", resultHcomp, i);
	//printf("\n");

}

void DWTLow( const float signal[], const int SignalLen, float
resultLcomp[])
{
	float filterDB10Low[] =  { 0.0000, 0.0001, -0.0001, -0.0007,
0.0020, 0.0014, -0.0107, 0.0036, 0.0332, -0.0295, -0.0714, 0.0931,
0.1274, -0.1959, -0.2498, 0.2812, 0.6885, 0.5272, 0.1882, 0.0267};

	int p;

	const int n =  SignalLen +38;
	const int r = n + 20 - 1;
	float x[n];
	float resultLow[r];

	//Lenght test.
	//printf("Tamano resultante de n = %d\n",n);
	//printf("Tamano resultante de r = %d\n",r);
	//printf("Tamano resultante de Elements Filters  =
%ld\n",ELEMENT_COUNT(filterDB10Low));
```

```
	for (p = 0; p < (SignalLen + ELEMENT_COUNT(filterDB10Low) -
1)/2; p++)
	{
		resultLcomp[p] = 0.0;
	}

	for (p = 0; p < 19; p++)
	{
		x[p] = signal[19-p-1];
		x[p+19+SignalLen] = signal[SignalLen-1-p];

	}

	for(p=0; p < SignalLen; p++){

		x[p+19] = signal[p];

	}


	//printSignal("x", x, n);
	//printf("\n");



	convolve(x, ELEMENT_COUNT(x),
			filterDB10Low, ELEMENT_COUNT(filterDB10Low),
			resultLow);
	p = ELEMENT_COUNT(resultLow);
	//printSignal("resultLow", resultLow, p);
	//printf("\n");

	int i;
	for (i = 0; i < (SignalLen + 19)/2; i++ )
	{
		resultLcomp[i] = resultLow[20+2*i];
	}

	//printSignal("resultLcomp", resultLcomp, i);
	//printf("\n");

}

void DWTHighRecon( const float resultHigh[], const int
resultHighLength, float signalHigh[]){
```

```
	float reconDB10High[] = { 0.0000, -0.0001, -0.0001, 0.0007,
0.0020, -0.0014,  -0.0107, -0.0036, 0.0332, 0.0295, -0.0714, -
0.0931,
		0.1274, 0.1959, -0.2498, -0.2812, 0.6885, -0.5272,
0.1882,-0.0267};
	int m = 2*resultHighLength;
	int f = 20;
	int s = m-f+2;
	int p;
	int l = m-1;
	int h = l+19;
	float y[l];
	float yalt[h];
	float d = (h-s)/2;
	int first = floor(d);
	int last = h-ceil(d);

	//Printfs
	//printf("Tamano resultante de m = %d\n",m);
	//printf("Tamano resultante de f = %d\n",f);
	//printf("Tamano resultante de s = %d\n",s);
	//printf("Tamano resultante de p = %d\n",p);
	//printf("Tamano resultante de l = %d\n",l);
	//printf("Tamano resultante de h = %d\n",h);
	//printf("Tamano resultante de d = %f\n",d);
	//printf("Tamano resultante de first = %d\n",first);
	//printf("Tamano resultante de last = %d\n",last);
	//printf("\n");

	for(p=0;p<l;p++){
		y[p]=0;
	}

	for(p=0; p<resultHighLength; p++){
		y[2*p]=resultHigh[p];
	}

	convolve(y, ELEMENT_COUNT(y), reconDB10High,
ELEMENT_COUNT(reconDB10High), yalt);
	for(p=first; p<last;p++){
		signalHigh[p-first]=yalt[p];
	}

	//printSignal("signalHigh", signalHigh, p-first);
```

```
}

void DWTLowRecon( const float resultLow[], const int
resultLowLength, float signalLow[]){

      float reconDB10Low[] = { 0.0267, 0.1882, 0.5272, 0.6885,
0.2812, -0.2498, -0.1959, 0.1274, 0.0931, -0.0714, -0.0295,
0.0332,
            0.0036, -0.0107, 0.0014, 0.0020, -0.0007, -0.0001,
0.0001, 0.0000};
      int m = 2*resultLowLength;
      int f = 20;
      int s = m-f+2;
      int p;
      int l = m-1;
      int h = l+19;
      float y[l];
      float yalt[h];
      float d = (h-s)/2;
      int first = floor(d);
      int last = h-ceil(d);

      //Printfs
      //printf("Tamano resultante de m = %d\n",m);
      //printf("Tamano resultante de f = %d\n",f);
      //printf("Tamano resultante de s = %d\n",s);
      //printf("Tamano resultante de p = %d\n",p);
      //printf("Tamano resultante de l = %d\n",l);
      //printf("Tamano resultante de h = %d\n",h);
      //printf("Tamano resultante de d = %f\n",d);
      //printf("Tamano resultante de first = %d\n",first);
      //printf("Tamano resultante de last = %d\n",last);
      //printf("\n");

      for(p=0;p<l;p++){
            y[p]=0;
      }

      for(p=0; p<resultLowLength; p++){
            y[2*p]=resultLow[p];
      }

      convolve(y, ELEMENT_COUNT(y), reconDB10Low,
ELEMENT_COUNT(reconDB10Low), yalt);
      for(p=first; p<last;p++){
            signalLow[p-first]=yalt[p];
```

```
	}

	//printSignal("signalLow", signalLow, p-first);


}

void DWT( const float signal[], const int SignalLen, float
resultHcomp[], float resultLcomp[]){
	DWTHigh(signal, SignalLen, resultHcomp);
	DWTLow(signal, SignalLen, resultLcomp);

}
void IDWT( const float resultHigh[], const int resultHighLen,
const float resultLow[], const int resultLowLen, float
signalHigh[], float signalLow[], float reconSign[], int u){
	int g;
	DWTHighRecon(resultHigh, resultHighLen, signalHigh);
	DWTLowRecon(resultLow, resultLowLen, signalLow);
	for(g=0; g<u;g++){
		reconSign[g]=signalHigh[g]+signalLow[g];
	}
	//printSignal("reconSignal", reconSign, u);

}

int main(void){
	float signal[] = {Signal points};
	int signalLen = ELEMENT_COUNT(signal);
	struct timespec begin, end;
	long diff_sec, diff_nsec, m_sec, m_nsec;
	int i;

	//printSignal("signal", signal, signalLen );

	int u =(signalLen + 20 - 1)/2;
	int d =(u + 20 - 1)/2;
	int e =(d + 20 - 1)/2;
	int f =(e + 20 - 1)/2;
	int g =(f + 20 - 1)/2;

	float resultH[u];
	float resultL[u];
	float resultLH[d];
	float resultLL[d];
	float resultLLH[e];
```

```
	float resultLLL[e];
	float resultLLLL[f];
	float resultLLLH[f];
	float resultLLHL[f];
	float resultLLHH[f];
	float resultLLLHL[g];
	float resultLLLHH[g];

	float reconSignalLLLH[f];
	float signalLLLHL[f];
	float signalLLLHH[f];
	float reconSignalLLL[e];
	float signalLLLL[e];
	float signalLLLH[e];
	float reconSignalLLH[e];
	float signalLLHH[e];
	float signalLLHL[e];
	float reconSignalLL[d];
	float signalLLH[d];
	float signalLLL[d];
	float reconSignalL[u];
	float signalLH[u];
	float signalLL[u];
	float reconSignal[signalLen];
	float signalH[signalLen];
	float signalL[signalLen];

	//printf("Tamano resultante de u = %d\n", u);
	//printf("\n");
	//printf("Tamano resultante de signal = %d\n", signalLen);

	//-------------------COMPRESSION-----------------------

	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &begin);

	for(i = 0; i < ITERATIONS; i++)
	{
		DWT(signal, signalLen, resultH, resultL);
	}

	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);
	diff_sec = difftime(end.tv_sec, begin.tv_sec);
	diff_nsec = end.tv_nsec > begin.tv_nsec ? end.tv_nsec -
begin.tv_nsec : 1000000000 - begin.tv_nsec + end.tv_nsec;

	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &begin);
```

```
	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);
	m_sec = difftime(end.tv_sec, begin.tv_sec);
	m_nsec = end.tv_nsec > begin.tv_nsec ? end.tv_nsec -
begin.tv_nsec : 1000000000 - begin.tv_nsec + end.tv_nsec;

	fprintf(stderr, "Compression took %ld.%09ld s\n", diff_sec -
m_sec, diff_nsec > m_sec ? diff_nsec - m_sec : 1000000000 - m_sec
+ diff_nsec);

	//-----------------------------------------------------

	//-----------------DECOMPRESSION-----------------------

	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &begin);

	for(i = 0; i < ITERATIONS; i++)
	{
		IDWT(resultH, u, resultL, u, signalH, signalL,
reconSignal,signalLen );
	}

	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);
	diff_sec = difftime(end.tv_sec, begin.tv_sec);
	diff_nsec = end.tv_nsec > begin.tv_nsec ? end.tv_nsec -
begin.tv_nsec : 1000000000 - begin.tv_nsec + end.tv_nsec;

	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &begin);
	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);
	m_sec = difftime(end.tv_sec, begin.tv_sec);
	m_nsec = end.tv_nsec > begin.tv_nsec ? end.tv_nsec -
begin.tv_nsec : 1000000000 - begin.tv_nsec + end.tv_nsec;

	fprintf(stderr, "Decompression took %ld.%09ld s\n", diff_sec
- m_sec, diff_nsec > m_sec ? diff_nsec - m_sec : 1000000000 -
m_sec + diff_nsec);

	//-----------------------------------------------------

	int y;
	float r;
	for(y =0; y< signalLen; y++){
		r =abs(abs(signal[y])-abs(reconSignal[y]));
		if( r > 4*e-2){
			printf("Error recontructing the signal, excceded
error");
		}
```

```
	}

	FILE *filePtr;
	filePtr = fopen("reconSignal.txt","w");

	int t;
	for (t = 0; t < signalLen; t++){
		fprintf(filePtr,"%.3g\t,", reconSignal[t]);
	}

	fclose(filePtr);


	return 0;


}
```

2. C Code for two Stages:

```
#include <stdio.h>
#include <stdlib.h>
//#include <stddef.h>
#include <math.h>
#include <time.h>

#define ELEMENT_COUNT(X) (sizeof(X) / sizeof((X)[0]))

#define ITERATIONS 100

void printSignal(const char * Name, float * Signal, size_t
SignalLen){
	int i;

	for (i = 0; i < SignalLen; i++)
	{
		printf("%s[%d] = %f\n", Name, i, Signal[i]);
	}
	printf("\n");
}

void convolve(const float Signal[], int SignalLen, const float
filterDB10[], int filterDB10Len, float Result[])
{
```

```
      int n;

      for (n = 0; n < SignalLen + filterDB10Len - 1; n++)
      {
            int kmin, kmax, k;

            Result[n] = 0.0;

            kmin = (n >= filterDB10Len - 1) ? n - (filterDB10Len -
1) : 0;
            kmax = (n < SignalLen - 1) ? n : SignalLen - 1;

            for (k = kmin; k <= kmax; k++)
            {
                  Result[n] += Signal[k] * filterDB10[n - k];
            }
      }
}

void DWTHigh( const float signal[], const int SignalLen, float
resultHcomp[]){
      float filterDB10High[] =  { -0.0267, 0.1882, -0.5272, 0.6885,
-0.2812, -0.2498, 0.1959, 0.1274, -0.0931, -0.0714, 0.0295,
0.0332,-0.0036, -0.0107, -0.0014, 0.0020, 0.0007, -0.0001, -
0.0001, 0.0000};
      int p;


      const int m = SignalLen+1;
      const int n =  SignalLen +38;//even
      const int q = m + 38;//odd
      const int o = q + 20 - 1;//odd
      const int r = n + 20 - 1;//even
      float x[n];
      float resultHigh[r];

      //Lenght test.
      //printf("Tamano resultante de m = %d\n",m);
      //printf("Tamano resultante de q = %d\n",q);
      //printf("Tamano resultante de n = %d\n",n);
      //printf("Tamano resultante de o = %d\n",o);
      //printf("Tamano resultante de r = %d\n",r);
      //printf("Tamano resultante de Elements Filters  =
%ld\n",ELEMENT_COUNT(filterDB10High));
```

```
      for (p = 0; p < (SignalLen + ELEMENT_COUNT(filterDB10High) -
1)/2; p++)
      {
            resultHcomp[p] = 0.0;
      }

      for (p = 0; p < 19; p++)
      {
            x[p] = signal[19-p-1];
            x[p+19+SignalLen] = signal[SignalLen-1-p];

      }

      for(p=0; p < SignalLen; p++){

            x[p+19] = signal[p];

      }


      //printSignal("x", x, n);
      //printf("\n");



      convolve(x, ELEMENT_COUNT(x),
                  filterDB10High, ELEMENT_COUNT(filterDB10High),
                  resultHigh);
      p = ELEMENT_COUNT(resultHigh);
      //printSignal("resultHigh", resultHigh, p);
      //printf("\n");

      int i;
      for (i = 0; i < (SignalLen + 19)/2; i++ )
      {
            resultHcomp[i] = resultHigh[20+2*i];
      }

      //printSignal("resultHcomp", resultHcomp, i);
      //printf("\n");

}

void DWTLow( const float signal[], const int SignalLen, float
resultLcomp[])
{
```

```
     float filterDB10Low[] =  { 0.0000, 0.0001, -0.0001, -0.0007,
0.0020, 0.0014, -0.0107, 0.0036, 0.0332, -0.0295, -0.0714, 0.0931,
0.1274, -0.1959, -0.2498, 0.2812, 0.6885, 0.5272, 0.1882, 0.0267};

     int p;

     const int n =  SignalLen +38;
     const int r = n + 20 - 1;
     float x[n];
     float resultLow[r];

     //Lenght test.
     //printf("Tamano resultante de n = %d\n",n);
     //printf("Tamano resultante de r = %d\n",r);
     //printf("Tamano resultante de Elements Filters  =
%ld\n",ELEMENT_COUNT(filterDB10Low));

     for (p = 0; p < (SignalLen + ELEMENT_COUNT(filterDB10Low) -
1)/2; p++)
     {
          resultLcomp[p] = 0.0;
     }

     for (p = 0; p < 19; p++)
     {
          x[p] = signal[19-p-1];
          x[p+19+SignalLen] = signal[SignalLen-1-p];

     }

     for(p=0; p < SignalLen; p++){

          x[p+19] = signal[p];

     }


     //printSignal("x", x, n);
     //printf("\n");



     convolve(x, ELEMENT_COUNT(x),
                 filterDB10Low, ELEMENT_COUNT(filterDB10Low),
                 resultLow);
     p = ELEMENT_COUNT(resultLow);
```

```
	//printSignal("resultLow", resultLow, p);
	//printf("\n");

	int i;
	for (i = 0; i < (SignalLen + 19)/2; i++ )
	{
		resultLcomp[i] = resultLow[20+2*i];
	}

	//printSignal("resultLcomp", resultLcomp, i);
	//printf("\n");

}

void DWTHighRecon( const float resultHigh[], const int
resultHighLength, float signalHigh[]){

	float reconDB10High[] = { 0.0000, -0.0001, -0.0001, 0.0007,
0.0020, -0.0014,  -0.0107, -0.0036, 0.0332, 0.0295, -0.0714, -
0.0931,
		0.1274, 0.1959, -0.2498, -0.2812, 0.6885, -0.5272,
0.1882,-0.0267};
	int m = 2*resultHighLength;
	int f = 20;
	int s = m-f+2;
	int p;
	int l = m-1;
	int h = l+19;
	float y[l];
	float yalt[h];
	float d = (h-s)/2;
	int first = floor(d);
	int last = h-ceil(d);

	//Printfs
	//printf("Tamano resultante de m = %d\n",m);
	//printf("Tamano resultante de f = %d\n",f);
	//printf("Tamano resultante de s = %d\n",s);
	//printf("Tamano resultante de p = %d\n",p);
	//printf("Tamano resultante de l = %d\n",l);
	//printf("Tamano resultante de h = %d\n",h);
	//printf("Tamano resultante de d = %f\n",d);
	//printf("Tamano resultante de first = %d\n",first);
	//printf("Tamano resultante de last = %d\n",last);
	//printf("\n");
```

```
	for(p=0;p<l;p++){
		y[p]=0;
	}

	for(p=0; p<resultHighLength; p++){
		y[2*p]=resultHigh[p];
	}

	convolve(y, ELEMENT_COUNT(y), reconDB10High,
ELEMENT_COUNT(reconDB10High), yalt);
	for(p=first; p<last;p++){
		signalHigh[p-first]=yalt[p];
	}

	//printSignal("signalHigh", signalHigh, p-first);


}

void DWTLowRecon( const float resultLow[], const int
resultLowLength, float signalLow[]){

	float reconDB10Low[] = { 0.0267, 0.1882, 0.5272, 0.6885,
0.2812, -0.2498, -0.1959, 0.1274, 0.0931, -0.0714, -0.0295,
0.0332,
		0.0036, -0.0107, 0.0014, 0.0020, -0.0007, -0.0001,
0.0001, 0.0000};
	int m = 2*resultLowLength;
	int f = 20;
	int s = m-f+2;
	int p;
	int l = m-1;
	int h = l+19;
	float y[l];
	float yalt[h];
	float d = (h-s)/2;
	int first = floor(d);
	int last = h-ceil(d);

	//Printfs
	//printf("Tamano resultante de m = %d\n",m);
	//printf("Tamano resultante de f = %d\n",f);
	//printf("Tamano resultante de s = %d\n",s);
	//printf("Tamano resultante de p = %d\n",p);
	//printf("Tamano resultante de l = %d\n",l);
	//printf("Tamano resultante de h = %d\n",h);
```

```
	//printf("Tamano resultante de d = %f\n",d);
	//printf("Tamano resultante de first = %d\n",first);
	//printf("Tamano resultante de last = %d\n",last);
	//printf("\n");

	for(p=0;p<l;p++){
		y[p]=0;
	}

	for(p=0; p<resultLowLength; p++){
		y[2*p]=resultLow[p];
	}

	convolve(y, ELEMENT_COUNT(y), reconDB10Low,
ELEMENT_COUNT(reconDB10Low), yalt);
	for(p=first; p<last;p++){
		signalLow[p-first]=yalt[p];
	}

	//printSignal("signalLow", signalLow, p-first);


}

void DWT( const float signal[], const int SignalLen, float
resultHcomp[], float resultLcomp[]){
	DWTHigh(signal, SignalLen, resultHcomp);
	DWTLow(signal, SignalLen, resultLcomp);

}
void IDWT( const float resultHigh[], const int resultHighLen,
const float resultLow[], const int resultLowLen, float
signalHigh[], float signalLow[], float reconSign[], int u){
	int g;
	DWTHighRecon(resultHigh, resultHighLen, signalHigh);
	DWTLowRecon(resultLow, resultLowLen, signalLow);
	for(g=0; g<u;g++){
		reconSign[g]=signalHigh[g]+signalLow[g];
	}
	//printSignal("reconSignal", reconSign, u);

}

int main(void){
	float signal[] = {Signal Points};
	int signalLen = ELEMENT_COUNT(signal);
```

```
struct timespec begin, end;
long diff_sec, diff_nsec, m_sec, m_nsec;
int i;

//printSignal("signal", signal, signalLen );

int u =(signalLen + 20 - 1)/2;
int d =(u + 20 - 1)/2;
int e =(d + 20 - 1)/2;
int f =(e + 20 - 1)/2;
int g =(f + 20 - 1)/2;

float resultH[u];
float resultL[u];
float resultLH[d];
float resultLL[d];
float resultLLH[e];
float resultLLL[e];
float resultLLLL[f];
float resultLLLH[f];
float resultLLHL[f];
float resultLLHH[f];
float resultLLLHL[g];
float resultLLLHH[g];

float reconSignalLLLH[f];
float signalLLLHL[f];
float signalLLLHH[f];
float reconSignalLLL[e];
float signalLLLL[e];
float signalLLLH[e];
float reconSignalLLH[e];
float signalLLHH[e];
float signalLLHL[e];
float reconSignalLL[d];
float signalLLH[d];
float signalLLL[d];
float reconSignalL[u];
float signalLH[u];
float signalLL[u];
float reconSignal[signalLen];
float signalH[signalLen];
float signalL[signalLen];

//printf("Tamano resultante de u = %d\n", u);
//printf("\n");
```

```
	//printf("Tamano resultante de signal = %d\n", signalLen);

	//-------------------COMPRESSION----------------------

	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &begin);

	for(i = 0; i < ITERATIONS; i++)
	{
		DWT(signal, signalLen, resultH, resultL);
		DWT(resultL, u, resultLH, resultLL);
	}

	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);
	diff_sec = difftime(end.tv_sec, begin.tv_sec);
	diff_nsec = end.tv_nsec > begin.tv_nsec ? end.tv_nsec -
begin.tv_nsec : 1000000000 - begin.tv_nsec + end.tv_nsec;

	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &begin);
	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);
	m_sec = difftime(end.tv_sec, begin.tv_sec);
	m_nsec = end.tv_nsec > begin.tv_nsec ? end.tv_nsec -
begin.tv_nsec : 1000000000 - begin.tv_nsec + end.tv_nsec;

	fprintf(stderr, "Compression took %ld.%09ld s\n", diff_sec -
m_sec, diff_nsec > m_sec ? diff_nsec - m_sec : 1000000000 - m_sec
+ diff_nsec);

	//-----------------------------------------------------

	//-----------------DECOMPRESSION----------------------

	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &begin);

	for(i = 0; i < ITERATIONS; i++)
	{
		IDWT(resultLH, d, resultLL, d, signalLH, signalLL,
reconSignalL, u);
		IDWT(resultH, u, reconSignalL, u, signalH, signalL,
reconSignal,signalLen );
	}

	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);
	diff_sec = difftime(end.tv_sec, begin.tv_sec);
	diff_nsec = end.tv_nsec > begin.tv_nsec ? end.tv_nsec -
begin.tv_nsec : 1000000000 - begin.tv_nsec + end.tv_nsec;
```

```
	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &begin);
	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);
	m_sec = difftime(end.tv_sec, begin.tv_sec);
	m_nsec = end.tv_nsec > begin.tv_nsec ? end.tv_nsec -
begin.tv_nsec : 1000000000 - begin.tv_nsec + end.tv_nsec;

	fprintf(stderr, "Decompression took %ld.%09ld s\n", diff_sec
- m_sec, diff_nsec > m_sec ? diff_nsec - m_sec : 1000000000 -
m_sec + diff_nsec);

	//-----------------------------------------------------

	int y;
	float r;
	for(y =0; y< signalLen; y++){
		r =abs(abs(signal[y])-abs(reconSignal[y]));
		if( r > 4*e-2){
			printf("Error recontructing the signal, excceded
error");
		}
	}

	FILE *filePtr;
	filePtr = fopen("reconSignal.txt","w");

	int t;
	for (t = 0; t < signalLen; t++){
		fprintf(filePtr,"%.3g\t,", reconSignal[t]);
	}

	fclose(filePtr);


	return 0;


}
```

3. C Code for Multi-Stage:

```
#include <stdio.h>
#include <stdlib.h>
//#include <stddef.h>
#include <math.h>
#include <time.h>
```

```
#define ELEMENT_COUNT(X) (sizeof(X) / sizeof((X)[0]))

#define ITERATIONS 100

void printSignal(const char * Name, float * Signal, size_t
SignalLen){
      int i;

      for (i = 0; i < SignalLen; i++)
      {
            printf("%s[%d] = %f\n", Name, i, Signal[i]);
      }
      printf("\n");
}

void convolve(const float Signal[], int SignalLen, const float
filterDB10[], int filterDB10Len, float Result[])
{
      int n;

      for (n = 0; n < SignalLen + filterDB10Len - 1; n++)
      {
            int kmin, kmax, k;

            Result[n] = 0.0;

            kmin = (n >= filterDB10Len - 1) ? n - (filterDB10Len -
1) : 0;
            kmax = (n < SignalLen - 1) ? n : SignalLen - 1;

            for (k = kmin; k <= kmax; k++)
            {
                  Result[n] += Signal[k] * filterDB10[n - k];
            }
      }
}

void DWTHigh( const float signal[], const int SignalLen, float
resultHcomp[]){
      float filterDB10High[] =  { -0.0267, 0.1882, -0.5272, 0.6885,
-0.2812, -0.2498, 0.1959, 0.1274, -0.0931, -0.0714, 0.0295,
0.0332,-0.0036, -0.0107, -0.0014, 0.0020, 0.0007, -0.0001, -
0.0001, 0.0000};
      int p;
```

```
      const int m = SignalLen+1;
      const int n =  SignalLen +38;//even
      const int q = m + 38;//odd
      const int o = q + 20 - 1;//odd
      const int r = n + 20 - 1;//even
      float x[n];
      float resultHigh[r];

      //Lenght test.
      //printf("Tamano resultante de m = %d\n",m);
      //printf("Tamano resultante de q = %d\n",q);
      //printf("Tamano resultante de n = %d\n",n);
      //printf("Tamano resultante de o = %d\n",o);
      //printf("Tamano resultante de r = %d\n",r);
      //printf("Tamano resultante de Elements Filters  =
%ld\n",ELEMENT_COUNT(filterDB10High));

      for (p = 0; p < (SignalLen + ELEMENT_COUNT(filterDB10High) -
1)/2; p++)
      {
            resultHcomp[p] = 0.0;
      }

      for (p = 0; p < 19; p++)
      {
            x[p] = signal[19-p-1];
            x[p+19+SignalLen] = signal[SignalLen-1-p];

      }

      for(p=0; p < SignalLen; p++){

            x[p+19] = signal[p];

      }


      //printSignal("x", x, n);
      //printf("\n");



      convolve(x, ELEMENT_COUNT(x),
                  filterDB10High, ELEMENT_COUNT(filterDB10High),
                  resultHigh);
```

```
	p = ELEMENT_COUNT(resultHigh);
	//printSignal("resultHigh", resultHigh, p);
	//printf("\n");

	int i;
	for (i = 0; i < (SignalLen + 19)/2; i++ )
	{
		resultHcomp[i] = resultHigh[20+2*i];
	}

	//printSignal("resultHcomp", resultHcomp, i);
	//printf("\n");

}

void DWTLow( const float signal[], const int SignalLen, float
resultLcomp[])
{
	float filterDB10Low[] =  { 0.0000, 0.0001, -0.0001, -0.0007,
0.0020, 0.0014, -0.0107, 0.0036, 0.0332, -0.0295, -0.0714, 0.0931,
0.1274, -0.1959, -0.2498, 0.2812, 0.6885, 0.5272, 0.1882, 0.0267};

	int p;

	const int n =  SignalLen +38;
	const int r = n + 20 - 1;
	float x[n];
	float resultLow[r];

	//Lenght test.
	//printf("Tamano resultante de n = %d\n",n);
	//printf("Tamano resultante de r = %d\n",r);
	//printf("Tamano resultante de Elements Filters  =
%ld\n",ELEMENT_COUNT(filterDB10Low));

	for (p = 0; p < (SignalLen + ELEMENT_COUNT(filterDB10Low) -
1)/2; p++)
	{
		resultLcomp[p] = 0.0;
	}

	for (p = 0; p < 19; p++)
	{
		x[p] = signal[19-p-1];
		x[p+19+SignalLen] = signal[SignalLen-1-p];
```

```
	}

	for(p=0; p < SignalLen; p++){

		x[p+19] = signal[p];

	}


	//printSignal("x", x, n);
	//printf("\n");



	convolve(x, ELEMENT_COUNT(x),
			filterDB10Low, ELEMENT_COUNT(filterDB10Low),
			resultLow);
	p = ELEMENT_COUNT(resultLow);
	//printSignal("resultLow", resultLow, p);
	//printf("\n");

	int i;
	for (i = 0; i < (SignalLen + 19)/2; i++ )
	{
		resultLcomp[i] = resultLow[20+2*i];
	}

	//printSignal("resultLcomp", resultLcomp, i);
	//printf("\n");

}

void DWTHighRecon( const float resultHigh[], const int
resultHighLength, float signalHigh[]){

	float reconDB10High[] = { 0.0000, -0.0001, -0.0001, 0.0007,
0.0020, -0.0014,  -0.0107, -0.0036, 0.0332, 0.0295, -0.0714, -
0.0931,
		0.1274, 0.1959, -0.2498, -0.2812, 0.6885, -0.5272,
0.1882,-0.0267};
	int m = 2*resultHighLength;
	int f = 20;
	int s = m-f+2;
	int p;
	int l = m-1;
	int h = l+19;
```

```
	float y[l];
	float yalt[h];
	float d = (h-s)/2;
	int first = floor(d);
	int last = h-ceil(d);

	//Printfs
	//printf("Tamano resultante de m = %d\n",m);
	//printf("Tamano resultante de f = %d\n",f);
	//printf("Tamano resultante de s = %d\n",s);
	//printf("Tamano resultante de p = %d\n",p);
	//printf("Tamano resultante de l = %d\n",l);
	//printf("Tamano resultante de h = %d\n",h);
	//printf("Tamano resultante de d = %f\n",d);
	//printf("Tamano resultante de first = %d\n",first);
	//printf("Tamano resultante de last = %d\n",last);
	//printf("\n");

	for(p=0;p<l;p++){
		y[p]=0;
	}

	for(p=0; p<resultHighLength; p++){
		y[2*p]=resultHigh[p];
	}

	convolve(y, ELEMENT_COUNT(y), reconDB10High,
ELEMENT_COUNT(reconDB10High), yalt);
	for(p=first; p<last;p++){
		signalHigh[p-first]=yalt[p];
	}

	//printSignal("signalHigh", signalHigh, p-first);


}

void DWTLowRecon( const float resultLow[], const int
resultLowLength, float signalLow[]){

	float reconDB10Low[] = { 0.0267, 0.1882, 0.5272, 0.6885,
0.2812, -0.2498, -0.1959, 0.1274, 0.0931, -0.0714, -0.0295,
0.0332,
		0.0036, -0.0107, 0.0014, 0.0020, -0.0007, -0.0001,
0.0001, 0.0000};
	int m = 2*resultLowLength;
```

```
	int f = 20;
	int s = m-f+2;
	int p;
	int l = m-1;
	int h = l+19;
	float y[l];
	float yalt[h];
	float d = (h-s)/2;
	int first = floor(d);
	int last = h-ceil(d);

	//Printfs
	//printf("Tamano resultante de m = %d\n",m);
	//printf("Tamano resultante de f = %d\n",f);
	//printf("Tamano resultante de s = %d\n",s);
	//printf("Tamano resultante de p = %d\n",p);
	//printf("Tamano resultante de l = %d\n",l);
	//printf("Tamano resultante de h = %d\n",h);
	//printf("Tamano resultante de d = %f\n",d);
	//printf("Tamano resultante de first = %d\n",first);
	//printf("Tamano resultante de last = %d\n",last);
	//printf("\n");

	for(p=0;p<l;p++){
		y[p]=0;
	}

	for(p=0; p<resultLowLength; p++){
		y[2*p]=resultLow[p];
	}

	convolve(y, ELEMENT_COUNT(y), reconDB10Low,
ELEMENT_COUNT(reconDB10Low), yalt);
	for(p=first; p<last;p++){
		signalLow[p-first]=yalt[p];
	}

	//printSignal("signalLow", signalLow, p-first);


}

void DWT( const float signal[], const int SignalLen, float
resultHcomp[], float resultLcomp[]){
	DWTHigh(signal, SignalLen, resultHcomp);
	DWTLow(signal, SignalLen, resultLcomp);
```

```
}
void IDWT( const float resultHigh[], const int resultHighLen,
const float resultLow[], const int resultLowLen, float
signalHigh[], float signalLow[], float reconSign[], int u){
      int g;
      DWTHighRecon(resultHigh, resultHighLen, signalHigh);
      DWTLowRecon(resultLow, resultLowLen, signalLow);
      for(g=0; g<u;g++){
            reconSign[g]=signalHigh[g]+signalLow[g];
      }
      //printSignal("reconSignal", reconSign, u);

}

int main(void){
      float signal[] = {Signal Points};
      int signalLen = ELEMENT_COUNT(signal);
      struct timespec begin, end;
      long diff_sec, diff_nsec, m_sec, m_nsec;
      int i;

      //printSignal("signal", signal, signalLen );

      int u =(signalLen + 20 - 1)/2;
      int d =(u + 20 - 1)/2;
      int e =(d + 20 - 1)/2;
      int f =(e + 20 - 1)/2;
      int g =(f + 20 - 1)/2;

      float resultH[u];
      float resultL[u];
      float resultLH[d];
      float resultLL[d];
      float resultLLH[e];
      float resultLLL[e];
      float resultLLLL[f];
      float resultLLLH[f];
      float resultLLHL[f];
      float resultLLHH[f];
      float resultLLLHL[g];
      float resultLLLHH[g];

      float reconSignalLLLH[f];
      float signalLLLHL[f];
      float signalLLLHH[f];
```

```
	float reconSignalLLL[e];
	float signalLLLL[e];
	float signalLLLH[e];
	float reconSignalLLH[e];
	float signalLLHH[e];
	float signalLLHL[e];
	float reconSignalLL[d];
	float signalLLH[d];
	float signalLLL[d];
	float reconSignalL[u];
	float signalLH[u];
	float signalLL[u];
	float reconSignal[signalLen];
	float signalH[signalLen];
	float signalL[signalLen];

	//printf("Tamano resultante de u = %d\n", u);
	//printf("\n");
	//printf("Tamano resultante de signal = %d\n", signalLen);

	//-------------------COMPRESSION-----------------------

	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &begin);

	for(i = 0; i < ITERATIONS; i++)
	{
		DWT(signal, signalLen, resultH, resultL);
		DWT(resultL, u, resultLH, resultLL);
		DWT(resultLL, d, resultLLH, resultLLL);
		DWT(resultLLL, e, resultLLLH, resultLLLL);
		DWT(resultLLH, e, resultLLHH, resultLLHL);
		DWT(resultLLLH, f, resultLLLHH, resultLLLHL);
	}

	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);
	diff_sec = difftime(end.tv_sec, begin.tv_sec);
	diff_nsec = end.tv_nsec > begin.tv_nsec ? end.tv_nsec -
begin.tv_nsec : 1000000000 - begin.tv_nsec + end.tv_nsec;

	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &begin);
	clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);
	m_sec = difftime(end.tv_sec, begin.tv_sec);
	m_nsec = end.tv_nsec > begin.tv_nsec ? end.tv_nsec -
begin.tv_nsec : 1000000000 - begin.tv_nsec + end.tv_nsec;
```

```
     fprintf(stderr, "Compression took %ld.%09ld s\n", diff_sec -
m_sec, diff_nsec > m_sec ? diff_nsec - m_sec : 1000000000 - m_sec
+ diff_nsec);

     //-------------------------------------------------

     //-----------------DECOMPRESSION-----------------------

     clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &begin);

     for(i = 0; i < ITERATIONS; i++)
     {
          IDWT(resultLLLHH, g, resultLLLHL, g, signalLLLHH,
signalLLLHL, reconSignalLLLH, f);
          IDWT(reconSignalLLLH, f, resultLLLL, f, signalLLLH,
signalLLLL, reconSignalLLL, e);
          IDWT(resultLLHH, f, resultLLHL, f, signalLLHH,
signalLLHL, reconSignalLLH, e);
          IDWT(reconSignalLLH, e, reconSignalLLL, e, signalLLH,
signalLLL, reconSignalLL, d);
          IDWT(resultLH, d, reconSignalLL, d, signalLH, signalLL,
reconSignalL, u);
          IDWT(resultH, u, reconSignalL, u, signalH, signalL,
reconSignal,signalLen );
     }

     clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);
     diff_sec = difftime(end.tv_sec, begin.tv_sec);
     diff_nsec = end.tv_nsec > begin.tv_nsec ? end.tv_nsec -
begin.tv_nsec : 1000000000 - begin.tv_nsec + end.tv_nsec;

     clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &begin);
     clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);
     m_sec = difftime(end.tv_sec, begin.tv_sec);
     m_nsec = end.tv_nsec > begin.tv_nsec ? end.tv_nsec -
begin.tv_nsec : 1000000000 - begin.tv_nsec + end.tv_nsec;

     fprintf(stderr, "Decompression took %ld.%09ld s\n", diff_sec
- m_sec, diff_nsec > m_sec ? diff_nsec - m_sec : 1000000000 -
m_sec + diff_nsec);

     //-------------------------------------------------

     int y;
     float r;
     for(y =0; y< signalLen; y++){
```

```
            r =abs(abs(signal[y])-abs(reconSignal[y]));
            if( r > 4*e-2){
                  printf("Error recontructing the signal, excceded
error");
            }
      }

      FILE *filePtr;
      filePtr = fopen("reconSignal.txt","w");

      int t;
      for (t = 0; t < signalLen; t++){
            fprintf(filePtr,"%.3g\t,", reconSignal[t]);
      }

      fclose(filePtr);


      return 0;


}
```

4. Code for VHDL implementation:

Compress:

```
library ieee;
library ieee_proposed;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee_proposed.float_pkg.all;

use ieee_proposed.fixed_float_types.all;

entity Compress is
        port(
          clk      : in   std_logic;
          encomp        : in   std_logic;
          data      : in   float (8 downto -23);
          output    : out  float (8 downto -23);
          nxtelement : out  std_logic
        );
end entity Compress;
```

```
architecture behavior of Compress is
	signal num	: integer :=0;
	begin
		CompressSelection : process(clk) is
		begin
			if (rising_edge(clk)) then
				if (encomp = '1') then
					num <= num  + 1;
					if (num = 19) then
						nxtelement <= '1';
					end if;
					if (((num)>= 20) and (num<=2066)) then
						if (((num) mod 2) = 0) then
							output <= data;
						end if;
					end if;
						else output
<="00000000000000000000000000000000";
						end if;
			end if;
end process;
end architecture;
```

FIR:

High pass filter

```
library ieee;
library ieee_proposed;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee_proposed.float_pkg.all;

use ieee_proposed.fixed_float_types.all;


entity FIR is
	port(

		clk		: in  std_logic;

		enF		: in  std_logic;
		data	: in  float (8 downto -23);
		output	: out float (8 downto -23)
		);
```

```
end entity FIR;


architecture behavior of FIR is

        subtype float32 is float (8 downto -23);
        type float_array is array(0 to 19) of float32;
        signal tap: float_array :=
("00000000000000000000000000000000","00000000000000000000000000000000
0","00000000000000000000000000000000","00000000000000000000000000000000
000","00000000000000000000000000000000","00000000000000000000000000000000
00000","00000000000000000000000000000000","00000000000000000000000000000000
0000000","00000000000000000000000000000000","00000000000000000000000000000000
000000000","00000000000000000000000000000000","00000000000000000000000000000000
00000000000","00000000000000000000000000000000","00000000000000000000000000000000
0000000000000","00000000000000000000000000000000","00000000000000000000000000000000
000000000000000","00000000000000000000000000000000","00000000000000000000000000000000
00000000000000000","00000000000000000000000000000000","00000000000000000000000000000000
000000000000000000");

        begin
                process(clk) is
                  variable
coef0,coef1,coef2,coef3,coef4,coef5,coef6,coef7,coef8,coef9,coef10,coef11,coef12,coef1
3,
                                        coef14,coef15,coef16,coef17,coef18,coef19 :
float32;
                        begin
                                if (rising_edge(clk)) then
                                        if (enF = '1') then
                                                coef0 := to_float  (-0.0267 , coef0);
                                                coef1 := to_float  ( 0.1882 , coef1);
                                                coef2 := to_float  (-0.5272 , coef2);
                                                coef3 := to_float  ( 0.6885 , coef3);
                                                coef4 := to_float  (-0.2812 , coef4);
                                                coef5 := to_float  (-0.2498 , coef5);
                                                coef6 := to_float  ( 0.1959 , coef6);
                                                coef7 := to_float  ( 0.1274 , coef7);
                                                coef8 := to_float  (-0.0931 , coef8);
                                                coef9 := to_float  (-0.0714 , coef9);
                                                coef10 := to_float ( 0.0295 , coef10);
                                                coef11 := to_float ( 0.0332 , coef11);
                                                coef12 := to_float (-0.0036 , coef12);
                                                coef13 := to_float (-0.0107 , coef13);
                                                coef14 := to_float (-0.0014 , coef14);
                                                coef15 := to_float ( 0.0020 , coef15);
```

```
                                        coef16 := to_float ( 0.0007 , coef16);
                                        coef17 := to_float (-0.0001 , coef17);
                                        coef18 := to_float (-0.0001 , coef18);
                                        coef19 := to_float ( 0.0000 , coef19);

                                        for i in 19 downto 1 loop
                                                tap(i) <= tap(i-1);
                                        end loop;
                                        tap(0) <= data;
                                        output <=
tap(0)*(coef0)+tap(1)*(coef1)+tap(2)*(coef2)+tap(3)*(coef3)+tap(4)*(coef4)+tap(5)
*(coef5)+tap(6)*(coef6)+tap(7)*(coef7)+tap(8)*(coef8)+tap(9)*(coef9)+tap(10)*(coe
f10)+tap(11)*(coef11)+tap(12)*(coef12)+tap(13)*(coef13)+tap(14)*(coef14)+tap(15
)*(coef15)+tap(16)*(coef16)+tap(17)*(coef17)+tap(18)*(coef18)+tap(19)*(coef19);

                                        else output <=
"00000000000000000000000000000000";

                                        end if;
                                end if;
                end process;
end architecture;
```

FIRLow:

```
library ieee;
library ieee_proposed;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee_proposed.float_pkg.all;
use ieee_proposed.fixed_float_types.all;

entity FIRLow is
        port(
                clk             : in  std_logic;
                enF             : in  std_logic;
                data    : in  float (8 downto -23);
                output  : out float (8 downto -23)
                );
end entity FIRLow;

architecture behavior of FIRLow is

        subtype float32 is float (8 downto -23);
        type float_array is array(0 to 19) of float32;
```

```
	signal tap: float_array :=
("00000000000000000000000000000000","00000000000000000000000000000000
0","00000000000000000000000000000000","00000000000000000000000000000000
000","00000000000000000000000000000000","00000000000000000000000000000000
00000","00000000000000000000000000000000","00000000000000000000000000000000
0000000","00000000000000000000000000000000","00000000000000000000000000000000
000000000","00000000000000000000000000000000","00000000000000000000000000000000
00000000000","00000000000000000000000000000000","00000000000000000000000000000000
0000000000000","00000000000000000000000000000000","00000000000000000000000000000000
000000000000000","00000000000000000000000000000000","00000000000000000000000000000000
0000000000000000","00000000000000000000000000000000","00000000000000000000000000000000
00000000000000000");

	begin
		process(clk) is

		 variable
coef0,coef1,coef2,coef3,coef4,coef5,coef6,coef7,coef8,coef9,coef10,coef11,coef12,coef1
3, coef14,coef15,coef16,coef17,coef18,coef19 : float32;
			begin
				if (rising_edge(clk)) then
					if (enF = '1') then
						coef19 := to_float ( 0.0267 , coef0);
						coef18 := to_float ( 0.1882 , coef1);
						coef17 := to_float ( 0.5272 , coef2);
						coef16 := to_float ( 0.6885 , coef3);
						coef15 := to_float ( 0.2812 , coef4);
						coef14 := to_float (-0.2498 , coef5);
						coef13 := to_float (-0.1959 , coef6);
						coef12 := to_float ( 0.1274 , coef7);
						coef11 := to_float ( 0.0931 , coef8);
						coef10 := to_float (-0.0714 , coef9);
						coef9  := to_float (-0.0295 , coef10);
						coef8  := to_float ( 0.0332 , coef11);
						coef7  := to_float ( 0.0036 , coef12);
						coef6  := to_float (-0.0107 , coef13);
						coef5  := to_float ( 0.0014 , coef14);
						coef4  := to_float ( 0.0020 , coef15);
						coef3  := to_float (-0.0007 , coef16);
						coef2  := to_float (-0.0001 , coef17);
						coef1  := to_float ( 0.0001 , coef18);
						coef0  := to_float ( 0.0000 , coef19);
						for i in 19 downto 1 loop
							tap(i) <= tap(i-1);
						end loop;
						tap(0) <= data;
```

```
                                        output <=
tap(0)*(coef0)+tap(1)*(coef1)+tap(2)*(coef2)+tap(3)*(coef3)+tap(4)*(coef4)+tap(5)
*(coef5)+tap(6)*(coef6)+tap(7)*(coef7)+tap(8)*(coef8)+tap(9)*(coef9)+tap(10)*(coe
f10)+tap(11)*(coef11)+tap(12)*(coef12)+tap(13)*(coef13)+tap(14)*(coef14)+tap(15
)*(coef15)+tap(16)*(coef16)+tap(17)*(coef17)+tap(18)*(coef18)+tap(19)*(coef19);
                                        else
output<="00000000000000000000000000000000";
                                        end if;
                                end if;
        end process;
end architecture;
```

DWTH:

```
library ieee; library ieee_proposed;
 use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
 use ieee.std_logic_unsigned.all;
 use ieee_proposed.float_pkg.all;
use ieee_proposed.fixed_float_types.all;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity DWTH is
        port(
                clk             : in  std_logic;
                enF             : in  std_logic;
                encomp          : in  std_logic;
                data      : in  float (8 downto -23);
                output    : out float (8 downto -23);
                nxtelement : out std_logic
                );
end DWTH;

architecture Behavioral of DWTH is
                --signals:
                --components
                component FIR port(
                                        clk             : in  std_logic;
                                        enF             : in  std_logic;
```

```
                                    data    : in  float (8 downto -23);
                                    output  : out float (8 downto -23)
                                                        );
            end component;
            component compress port(
                                    clk                 : in  std_logic;
                                    encomp      : in  std_logic;
                                    data        : in  float (8 downto -23);
                                    output      : out float (8 downto -23);
                                    nxtelement : out std_logic
                                                        );
            end component;

            SIGNAL firoutput : float (8 downto -23);

begin

            filterH : entity WORK.FIR(behavior)
                                            port map (clk,enF,data,firoutput);

            comp    : entity WORK.compress(behavior)
                                            port
map(clk,encomp,firoutput,output,nxtelement);

end Behavioral;
```

DWTL:

```
library ieee; library ieee_proposed;
 use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
 use ieee.std_logic_unsigned.all;
 use ieee_proposed.float_pkg.all;
use ieee_proposed.fixed_float_types.all;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity DWTL is
        port(
            clk                 : in  std_logic;
```

```
                enF             : in  std_logic;
                encomp          : in  std_logic;
                data            : in  float (8 downto -23);
                output          : out float (8 downto -23);
                nxtelement      : out std_logic
                );
end DWTH;

architecture Behavioral of DWTL is
                --signals:
                --components
                component FIRL port(
                                clk             : in  std_logic;
                                enF             : in  std_logic;
                                data            : in  float (8 downto -23);
                                output          : out float (8 downto -23)
                                                );
                end component;
                component compress port(
                                clk               : in  std_logic;
                                encomp            : in  std_logic;
                                data              : in  float (8 downto -23);
                                output            : out float (8 downto -23);
                                nxtelement        : out std_logic
                                                );
                end component;

                SIGNAL firoutput : float (8 downto -23);

begin

                filterH : entity WORK.FIRL(behavior)
                                                port map (clk,enF,data,firoutput);

                comp    : entity WORK.compress(behavior)
                                                port
map(clk,encomp,firoutput,output,nxtelement);

end Behavioral;
```

FSM:

```
library ieee;
library ieee_proposed;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
```

```
 use ieee.std_logic_unsigned.all;
 use ieee_proposed.float_pkg.all;
use ieee_proposed.fixed_float_types.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity FSM is
                port(
                        clk             : in    std_logic;
                        enF             : out   std_logic;
                        encomp          : out   std_logic;
                        nxtelement      : in    std_logic;
                        startmem        : in    std_logic;   --it works as a reset.
                        addressMinputout   : inout  std_logic_vector(31 downto 0);
                        addressMoutputoutH : inout  std_logic_vector(31 downto 0);
                        addressMoutputoutL : inout  std_logic_vector(31 downto 0);
                        writeMemout             : out   std_logic;
                        readMemin               : out   std_logic;
                        datacontrol             : out   std_logic_vector(31 downto 0);
                        regcontrol              : out   std_logic
                        );
end FSM;

architecture Behavioral of FSM is
        type state_type is (s1, s2, s3, s3a, s4, s4a, s5, s5a, s6);
        SIGNAL state : state_type;
        begin
        process (state)
                variable auxin, auxoutH, auxoutL     : std_logic_vector(31 downto 0) :=
(others => '0');
                variable auxwriteMemout, auxreadMemin : std_logic;

                begin
                        case state is
                         when s1  => readMemin <= '1';
                                                enF     <= '0';
                                                encomp  <= '0';
                                                addressMinputout <=
"00000000000000000000000000000000";
```

```
                                        addressMoutputoutH  <=
"00000000000000000000000000000000";
                                        addressMoutputoutL  <=
"00000000000000000000010000001001";
                                        writeMemout       <= '0';
                                        auxin             :=
"00000000000000000000000000000000";
                                        auxoutH           :=
"00000000000000000000000000000000";
                                        auxoutL           :=
"00000000000000000000010000001001";
                                        datacontrol                 <=
"00000000000000000000000000000000";
                                        regcontrol                <= '0';

                    when s2  => enF <= '1';
                                        auxin := auxin +
"00000000000000000000000000000001";
                                        addressMinputout   <= auxin;
                    when s3  => encomp <= '1';
                                        auxin := auxin +
"00000000000000000000000000000001";
                                        addressMinputout   <= auxin;

                    when s3a => auxin := auxin +
"00000000000000000000000000000001";
                                        addressMinputout   <= auxin;
                    when s4  => auxin := auxin +
"00000000000000000000000000000001";
                                        addressMinputout   <= auxin;
                                         writeMemout       <= '1';
                                        auxoutH           := auxoutH +
"00000000000000000000000000000001";
                                        addressMoutputoutH <= auxoutH;
                                        auxoutL           := auxoutL +
"00000000000000000000000000000001";
                                        addressMoutputoutL <= auxoutL;
                    when s4a => auxin := auxin +
"00000000000000000000000000000001";
                                        addressMinputout   <= auxin;
                                        writeMemout       <= '0';
                    when s5 =>  auxreadMemin      := '0';
                                        readMemin         <= auxreadMemin;
                                        auxoutH           := auxoutH +
"00000000000000000000000000000001";
                                        addressMoutputoutH <= auxoutH;
```

```
                                        auxoutL          := auxoutL +
"00000000000000000000000000000001";
                                        addressMoutputoutL <= auxoutL;
                    when s5a => addressMoutputoutH <= auxoutH;
                                        addressMoutputoutL <= auxoutL;
                                        writeMemout      <= '0';
                    when s6  => writeMemout     <= '0';
                                        encomp                          <= '0';
                                        enF                     <= '0';
                                        datacontrol             <=
"00000000000000000000000000001111";
                                        regcontrol              <= '1';
                    end case;
            end process;
        process (clk, nxtelement, startmem)
            Begin
            if( startmem='1') then
                    state <= s1;
            end if;
                    if (rising_edge(clk)) then

                        case state is
                        when s1 => state <= s2;
                        when s2 => state <= s3;
                        when s3 => if (nxtelement = '1') then state <= s4;
                                    else state <= s3a;
                                end if;
                        when s3a=> if (nxtelement = '1') then state <= s4;
                                     else state <= s3;
                                        end if;
                        when s4 => if (CONV_INTEGER(addressMinputout)  >=
2047) then
                                        state <= s5;
                                    else state <= s4a;
                                    end if;
                        when s4a=> if (CONV_INTEGER(addressMinputout)  >=
2047) then
                                        state <= s5;
                                    else state <= s4;
                                     end if;
                        when s5 => if (CONV_INTEGER(addressMoutputoutH) >=
1032) then
                                        state <= s6;

                                    else state <= s5a;
                                    end if;
```

```
                        when s5a=> if (CONV_INTEGER(addressMoutputoutH) >=
1032) then
                                                state <= s6;

                                                else state <= s5;
                                                end if;
                        when s6 => state <= s1;
                end case;

                end if;

        end process;

end Behavioral;
```

User Logic:

```
------------------------------------------------------------------------------
-- user_logic.vhd - entity/architecture pair
------------------------------------------------------------------------------
--
-- ************************************************************************
-- ** Copyright (c) 1995-2012 Xilinx, Inc.  All rights reserved.          **
-- **                                                                     **
-- ** Xilinx, Inc.                                                        **
-- ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"        **
-- ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND
**
-- ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,       **
-- ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,
**
-- ** APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION          **
-- ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,
**
-- ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE
**
-- ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY            **
-- ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE
**
-- ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR
**
-- ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF
**
-- ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
**
-- ** FOR A PARTICULAR PURPOSE.                                          **
```

```
-- **                                                **
-- ************************************************************************
--
------------------------------------------------------------------------------
-- Filename:         user_logic.vhd
-- Version:          1.00.a
-- Description:      User logic.
-- Date:             Wed Jul 24 13:05:47 2013 (by Create and Import Peripheral Wizard)
-- VHDL Standard:    VHDL'93
------------------------------------------------------------------------------
-- Naming Conventions:
--   active low signals:                  "*_n"
--   clock signals:                       "clk", "clk_div#", "clk_#x"
--   reset signals:                       "rst", "rst_n"
--   generics:                            "C_*"
--   user defined types:                  "*_TYPE"
--   state machine next state:            "*_ns"
--   state machine current state:         "*_cs"
--   combinatorial signals:               "*_com"
--   pipelined or register delay signals:   "*_d#"
--   counter signals:                     "*cnt*"
--   clock enable signals:                "*_ce"
--   internal version of output port:     "*_i"
--   device pins:                         "*_pin"
--   ports:                               "- Names begin with Uppercase"
--   processes:                           "*_PROCESS"
--   component instantiations:            "<ENTITY_>I_<#|FUNC>"
------------------------------------------------------------------------------

-- DO NOT EDIT BELOW THIS LINE --------------------


library proc_common_v3_00_a;
use proc_common_v3_00_a.proc_common_pkg.all;

library ieee;

library work;
library ieee_proposed;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee_proposed.float_pkg.all;

use ieee_proposed.fixed_float_types.all;
```

```
use work.all;

-- DO NOT EDIT ABOVE THIS LINE --------------------

--USER libraries added here

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_SLV_AWIDTH              -- Slave interface address bus width
--   C_SLV_DWIDTH              -- Slave interface data bus width
--   C_NUM_MEM                 -- Number of memory spaces
--
-- Definition of Ports:
--   Bus2IP_Clk                -- Bus to IP clock
--   Bus2IP_Resetn             -- Bus to IP reset
--   Bus2IP_Addr               -- Bus to IP address bus
--   Bus2IP_CS                 -- Bus to IP chip select for user logic memory selection
--   Bus2IP_RNW                -- Bus to IP read/not write
--   Bus2IP_Data               -- Bus to IP data bus
--   Bus2IP_BE                 -- Bus to IP byte enables
--   Bus2IP_RdCE               -- Bus to IP read chip enable
--   Bus2IP_WrCE               -- Bus to IP write chip enable
--   Bus2IP_Burst              -- Bus to IP burst-mode qualifier
--   Bus2IP_BurstLength        -- Bus to IP burst length
--   Bus2IP_RdReq              -- Bus to IP read request
--   Bus2IP_WrReq              -- Bus to IP write request
--   IP2Bus_AddrAck            -- IP to Bus address acknowledgement
--   IP2Bus_Data               -- IP to Bus data bus
--   IP2Bus_RdAck              -- IP to Bus read transfer acknowledgement
--   IP2Bus_WrAck              -- IP to Bus write transfer acknowledgement
--   IP2Bus_Error              -- IP to Bus error response
--   Type_of_xfer              -- Transfer Type
------------------------------------------------------------------------------

entity user_logic is
 generic
 (
   -- ADD USER GENERICS BELOW THIS LINE ---------------
   --USER generics added here
   -- ADD USER GENERICS ABOVE THIS LINE ---------------

   -- DO NOT EDIT BELOW THIS LINE ----------------------
   -- Bus protocol parameters, do not add to or delete
   C_SLV_AWIDTH                : integer           := 32;
```

```
  C_SLV_DWIDTH                : integer           := 32;
  C_NUM_MEM                   : integer           := 3
  -- DO NOT EDIT ABOVE THIS LINE ----------------------
 );
 port
 (
  -- ADD USER PORTS BELOW THIS LINE ------------------
  --USER ports added here
  -- ADD USER PORTS ABOVE THIS LINE ------------------

  -- DO NOT EDIT BELOW THIS LINE ----------------------
  -- Bus protocol ports, do not add to or delete
  Bus2IP_Clk                  : in  std_logic;
  Bus2IP_Resetn               : in  std_logic;
  Bus2IP_Addr                 : in  std_logic_vector(C_SLV_AWIDTH-1 downto 0);
  Bus2IP_CS                   : in  std_logic_vector(C_NUM_MEM-1 downto 0);
  Bus2IP_RNW                  : in  std_logic;
  Bus2IP_Data                 : in  std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  Bus2IP_BE                   : in  std_logic_vector(C_SLV_DWIDTH/8-1 downto 0);
  Bus2IP_RdCE                 : in  std_logic_vector(C_NUM_MEM-1 downto 0);
  Bus2IP_WrCE                 : in  std_logic_vector(C_NUM_MEM-1 downto 0);
  Bus2IP_Burst                : in  std_logic;
  Bus2IP_BurstLength          : in  std_logic_vector(7 downto 0);
  Bus2IP_RdReq                : in  std_logic;
  Bus2IP_WrReq                : in  std_logic;
  IP2Bus_AddrAck              : out std_logic;
  IP2Bus_Data                 : out std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  IP2Bus_RdAck                : out std_logic;
  IP2Bus_WrAck                : out std_logic;
  IP2Bus_Error                : out std_logic;
  Type_of_xfer                : out std_logic
  -- DO NOT EDIT ABOVE THIS LINE ----------------------
 );

 attribute MAX_FANOUT : string;
 attribute SIGIS : string;

 attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
 attribute SIGIS of Bus2IP_Resetn : signal is "RST";

end entity user_logic;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------
```

```
architecture IMP of user_logic is

  --USER signal declarations added here, as needed for user logic

  ------------------------------------------
  -- Signals for user logic memory space example
  ------------------------------------------
  type RAM is array (0 to 2047) of std_logic_vector (31 downto 0);
  --type DO_TYPE is array (0 to C_NUM_MEM-1) of std_logic_vector(C_SLV_DWIDTH-1
downto 0);
  signal mem_data_out         : std_logic_vector (31 downto 0);
  signal mem_address          : std_logic_vector(7 downto 0);
  signal mem_select           : std_logic_vector(0 to 2);
  signal mem_read_enable      : std_logic;
  signal mem_ip2bus_data      : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal mem_read_ack_dly1    : std_logic;
  signal mem_read_ack_dly2    : std_logic;
  signal mem_read_ack         : std_logic;
  signal mem_write_ack        : std_logic;
  signal status_reg                       : std_logic_vector(31 downto 0);
  signal ram_input_mem        : RAM;
  signal ram_output_mem       : RAM;
  signal access_input_element   : integer;
  signal access_output_element  : integer;
  signal writeMemout          : std_logic;
  signal Routaddres_out1      : std_logic_vector(31 downto 0);
  signal Routaddres_out2      : std_logic_vector(31 downto 0);
  signal Routdata_in1         : float (8 downto -23);
  signal Routdata_in2         : float (8 downto -23);
  signal Rinaddres_out        : std_logic_vector(31 downto 0);
  signal Rindata_out          : float (8 downto -23);
  signal RinCRead             : std_logic;
  signal enF                              : std_logic;
  signal encomp                                 : std_logic;
  signal nxtelementH,nxtelementL: std_logic;
  signal regcontrol                       : std_logic;
  signal startmem                         : std_logic;
  signal datacontrol                  : std_logic_vector(31 downto 0);
  signal access_input_data    : std_logic_vector(31 downto 0);
  signal access_output_data   : std_logic_vector(31 downto 0);

  ---components declaration

                component DWTL port(
                                clk             : in  std_logic;
                                enF             : in  std_logic;
```

```
                                encomp          : in  std_logic;
                                data     : in  float (8 downto -23);
                                output   : out float (8 downto -23);
                                nxtelement: out std_logic
                                        );
        end component;

        component DWTH port(
                                clk             : in  std_logic;
                                enF             : in  std_logic;
                                encomp          : in  std_logic;
                                data     : in  float (8 downto -23);
                                output   : out float (8 downto -23);
                                nxtelement: out std_logic
                                        );
        end component;

        component FSM port(
                                clk          : in    std_logic;
                                enF             : out   std_logic;
                                encomp                  : out   std_logic;
                                nxtelement      : in    std_logic;
                                startmem        : in    std_logic;   --it works as a
reset.
                                addressMinputout   : inout  std_logic_vector(31
downto 0);
                                addressMoutputoutH : inout  std_logic_vector(31
downto 0);
                                addressMoutputoutL : inout  std_logic_vector(31
downto 0);
                                writeMemout             : out   std_logic;
                                readMemin               : out   std_logic;
                                datacontrol             : out   std_logic_vector(31
downto 0);
                                regcontrol              : out   std_logic
                                        );
        end component;

begin

 --USER logic implementation added here

        DWTHigh : DWTH PORT MAP(
                                        clk => Bus2IP_Clk ,
                                        enF => enF,
                                        encomp => encomp,
```

```
                                        data => Rindata_out,
                                        output => Routdata_in1,
                                        nxtelement => nxtelementH
                                         );

        DWTLow  : DWTL PORT MAP(
                                        clk       => Bus2IP_Clk,
                                        enF       => enF,
                                        encomp    => encomp,
                                        data      => Rindata_out,
                                        output    => Routdata_in2,
                                        nxtelement  => nxtelementL
                                         );

        FSMfinal  : FSM PORT MAP (
                                        clk       => Bus2IP_Clk,
                                        enF       => enF,
                                        encomp    => encomp,
                                        nxtelement  => nxtelementH,
                                        startmem    => startmem,
                                        addressMinputout   => Rinaddres_out,
                                        addressMoutputoutH => Routaddres_out1,
                                        addressMoutputoutL => Routaddres_out2,
                                        writeMemout       => writeMemout,
                                        readMemin                 => RinCread,
                                        datacontrol       => datacontrol,
                                        regcontrol        => regcontrol
                                                );
-----------------------------------------
-- Example code to access user logic memory region
--
-- Note:
-- The example code presented here is to show you one way of using
-- the user logic memory space features. The Bus2IP_Addr, Bus2IP_CS,
-- and Bus2IP_RNW IPIC signals are dedicated to these user logic
-- memory spaces. Each user logic memory space has its own address
-- range and is allocated one bit on the Bus2IP_CS signal to indicated
-- selection of that memory space. Typically these user logic memory
-- spaces are used to implement memory controller type cores, but it
-- can also be used in cores that need to access additional address space
-- (non C_BASEADDR based), s.t. bridges. This code snippet infers
-- 3 256x32-bit (byte accessible) single-port Block RAM by XST.
-----------------------------------------
mem_select     <= Bus2IP_CS;
mem_read_enable <= ( Bus2IP_RdCE(0) or Bus2IP_RdCE(2) );
mem_read_ack    <= mem_read_ack_dly1 and (not mem_read_ack_dly2);
```

```
mem_write_ack   <= ( Bus2IP_WrCE(0) or Bus2IP_WrCE(1) );

-- this process generates the read acknowledge 1 clock after read enable
-- is presented to the BRAM block. The BRAM block has a 1 clock delay
-- from read enable to data out.

BRAM_RD_ACK_PROC : process( Bus2IP_Clk ) is
begin

 if ( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
  if ( Bus2IP_Resetn = '0' ) then
   mem_read_ack_dly1 <= '0';
   mem_read_ack_dly2 <= '0';
  else
   mem_read_ack_dly1 <= mem_read_enable;
   mem_read_ack_dly2 <= mem_read_ack_dly1;
  end if;
 end if;

end process BRAM_RD_ACK_PROC;
-- REGISTER

REG : process(Bus2IP_Clk)
begin
 if(Bus2IP_Clk'event and Bus2IP_Clk = '1') then
  if Bus2IP_Resetn = '0' then
   status_reg <= (others => '0');
  else
   if Bus2IP_WrCE(0) = '1' then
                    if Bus2IP_Data(0) = '1' then
                                    startmem <='1';
                            end if;
     status_reg <= Bus2IP_Data;
   end if;
                  if regcontrol ='1' then
                          status_reg <= datacontrol;
                  end if;
  end if;
 end if;
end process;

-- Input Block

RAM_IN : process(Bus2IP_Clk)
begin
 if(Bus2IP_Clk'event and Bus2IP_Clk = '1') then
```

```
    if Bus2IP_WrCE(1) = '1' then
      ram_input_mem(CONV_INTEGER(Bus2IP_Addr(12 downto 2))) <= Bus2IP_Data;
    end if;
    if RinCRead = '1' then
       Rindata_out <= to_float(ram_input_mem(CONV_INTEGER(Rinaddres_out)));
    end if;
         end if;
  end process;

  -- Output Block

  RAM_OUT : process(Bus2IP_Clk)
  begin
   if(Bus2IP_Clk'event and Bus2IP_Clk = '1') then
    if writeMemout = '1' then
     ram_output_mem(CONV_INTEGER(Routaddres_out1)) <= to_slv(Routdata_in1);
                  ram_output_mem(CONV_INTEGER(Routaddres_out2)) <=
to_slv(Routdata_in2);
    end if;
   end if;
  end process;

  MEM_RD_BLOC2_PROC : process(Bus2IP_Clk) is
  begin
   if(Bus2IP_Clk'event and Bus2IP_Clk = '1') then
    mem_data_out <=ram_output_mem(CONV_INTEGER(Bus2IP_Addr(12 downto 2)));
   end if;
  end process MEM_RD_BLOC2_PROC;

  -- Bus 2 IP Mux

  MEM_IP2BUS_DATA_PROC : process( mem_data_out, mem_select ) is
  begin
   case mem_select is
    when "001" =>  mem_ip2bus_data <= status_reg;
    when "100" =>  mem_ip2bus_data <= mem_data_out;
    when others => mem_ip2bus_data <= (others => '0');
   end case;
  end process MEM_IP2BUS_DATA_PROC;
end IMP;
```