A Report on

Time Frequency Analysis of FMCW Radar Data

Submitted by

Ayush Mall	2016A3PS0163G
Kushagra Shah	2016A3PS0167G

Under the guidance of

Dr. A. Amalin Prince

Associate Professor and Head, EEE Department

In partial fulfilment of the

Laboratory Project (EEE F366)

At

BITS Pilani, K.K. Birla Goa Campus

(Second Semester, 2018 – 2019)

ACKNOWLEDGEMENTS

We would like to use this opportunity to express our gratitude to everyone who supported us throughout the course of this project. We are thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. We are sincerely grateful to them for sharing their truthful and illuminating views on the number of issues related to the project.

We would like to thank *Dr. A. Amalin Prince* for giving us this great opportunity of working under him on such an interesting topic. We would like to extend our gratitude do *Mr. Gibin Chacko George* for his constant support, words of wisdom and timely suggestions.

We would also like to thank everyone else who was directly or indirectly involved to provide us with this opportunity.

Ayush Mall

Kushagra Shah

ABSTRACT

This report gives an outline of the aim of the project and its applications. The aim is to analyse the previously studied methods for FMCW Radar Signal Processing and implement an efficient algorithm on FPGA to derive an accurate spectrogram for reflectometry data. The different approaches used involve various signal processing concepts – baseline wandering removal, signal normalisation, signal conditioning (using bandpass filter/ wavelet decomposition / empirical mode decomposition). The spectrogram of the signal obtained after processing the signal is compared with that of the original signal.

The algorithms have already been simulated and tested on MATLAB using the available libraries. The design implemented in the form of a python coded GUI is used to verify the obtained output. The algorithms have been modified to make it implementable on FPGA. Vivado HLS is used as the environment to implement the algorithms on Zynq Zedboard.

TABLE OF CONTENTS

LIST OF FIGURES

BACKGROUND

Reflectometry is used to detect or characterise object using the reflection of waves at the surfaces and interfaces of the object. This technique is used to study the properties of plasma using FMCW (Frequency Modulated Continuous Wave) signals.

In this method, the emitted radar signal gets reflected by the plasma surface when the frequency of the signal matches the plasma surface's frequency. The reflected signal is received after some time "t". The FMCW-Radar transmits a high frequency signal whose frequency increases linearly during the measurement phase (up-chirp, see Figure 1).

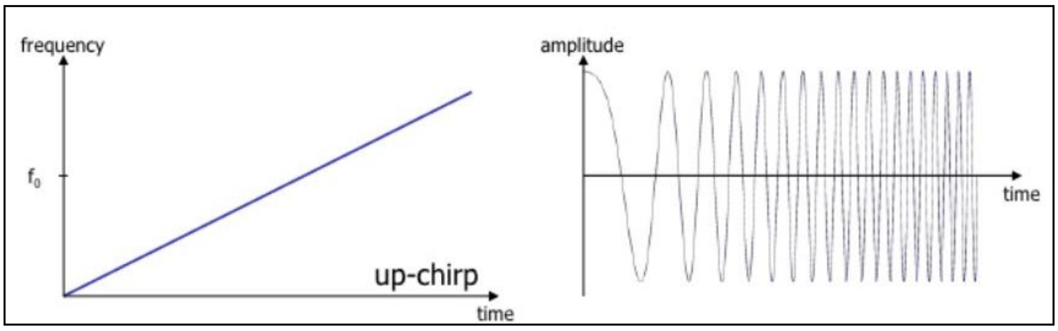


Figure 1 – FMCW Radar up-chirp

Due to the small magnitude of distance, it is difficult to accurately calculate the delay time. Hence, the frequency difference Δf is calculated from the actual transmit frequency and the receive frequency. This difference is directly proportional to the distance. It is transformed via a Fourier transformation (STFT) into a frequency spectrum and then the distance is calculated from the spectrum.

For this project, the sampled data of the received signal is already available. The aim of the project is to process the reflectometry data to obtain an accurate spectrogram. For achieving this, a number of methods are applied. The signal is first filtered to remove baseline wandering. It is then normalised (by using some threshold parameters), because an FM signal has a constant magnitude ideally. Finally, the signal is conditioned using one of the three techniques – bandpass filter, wavelet decomposition, empirical mode decomposition. Through this, we can obtain a better spectrogram because only the essential components of the signal are kept. For plotting the spectrogram, we are using the Short Term Fourier Transform (STFT).

The software implementation of the above algorithms has been already done – simulations were done on MATLAB and Python. For the hardware implementation, few changes were made. Also, Vivado HLS was chosen as the environment for FPGA implementation.

BASELINE WANDERING REMOVAL

The baseline of the signal should deviate as little as possible from a horizontal line. But due to electric signal fluctuations, temperature fluctuations and other factors, there is a short time variation of the baseline from a straight line. This is called baseline wandering. Figure 2 shows an example of baseline wandering removal.

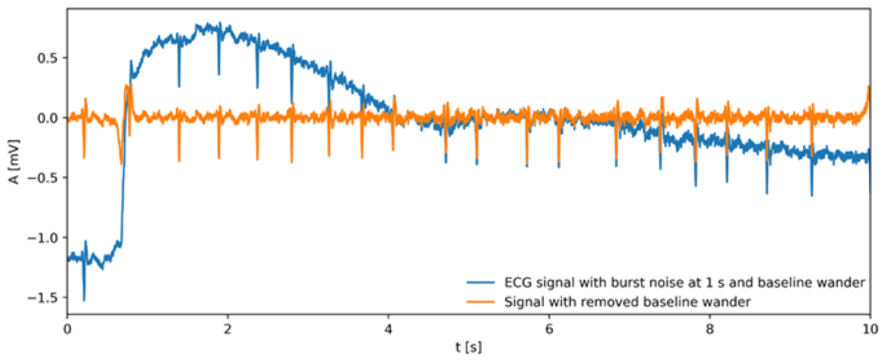
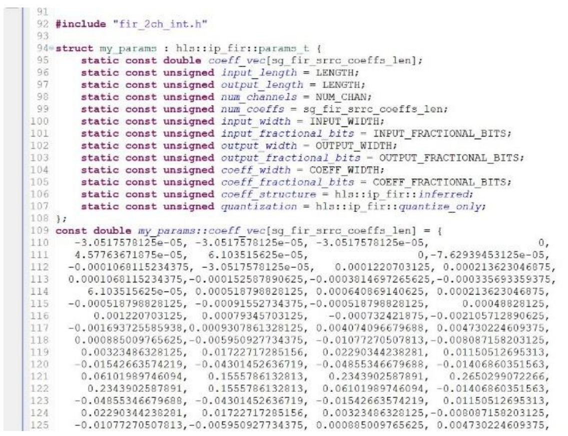


Figure 2 – Example of baseline wandering removal in ECG signals

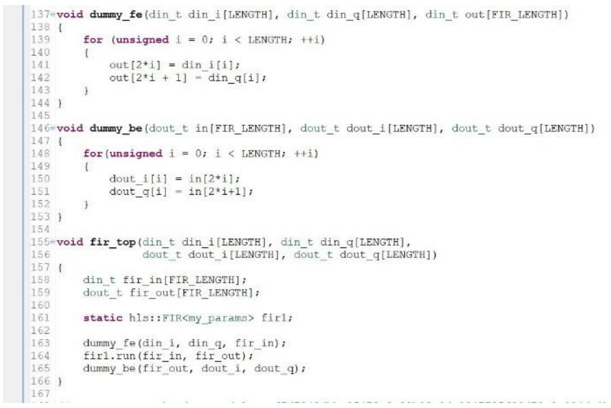
The baseline drift is usually of very low frequency. If the frequency is sufficiently low such that the signal frequency does not lie in that range, then a high pass filter can easily be used. Fortunately, this is our case, and we can simply use a high pass filter for baseline wandering removal. The same result can also be achieved by using a low pass filter and subtracting the filtered signal from the original signal. Both of these methods were used for analysis purposes.

Hardware Implementation:

No changes in the original algorithm were required. An example Vivado HLS project was used for lowpass / highpass filter implementation. Figure 3 shows the snapshot of the code:



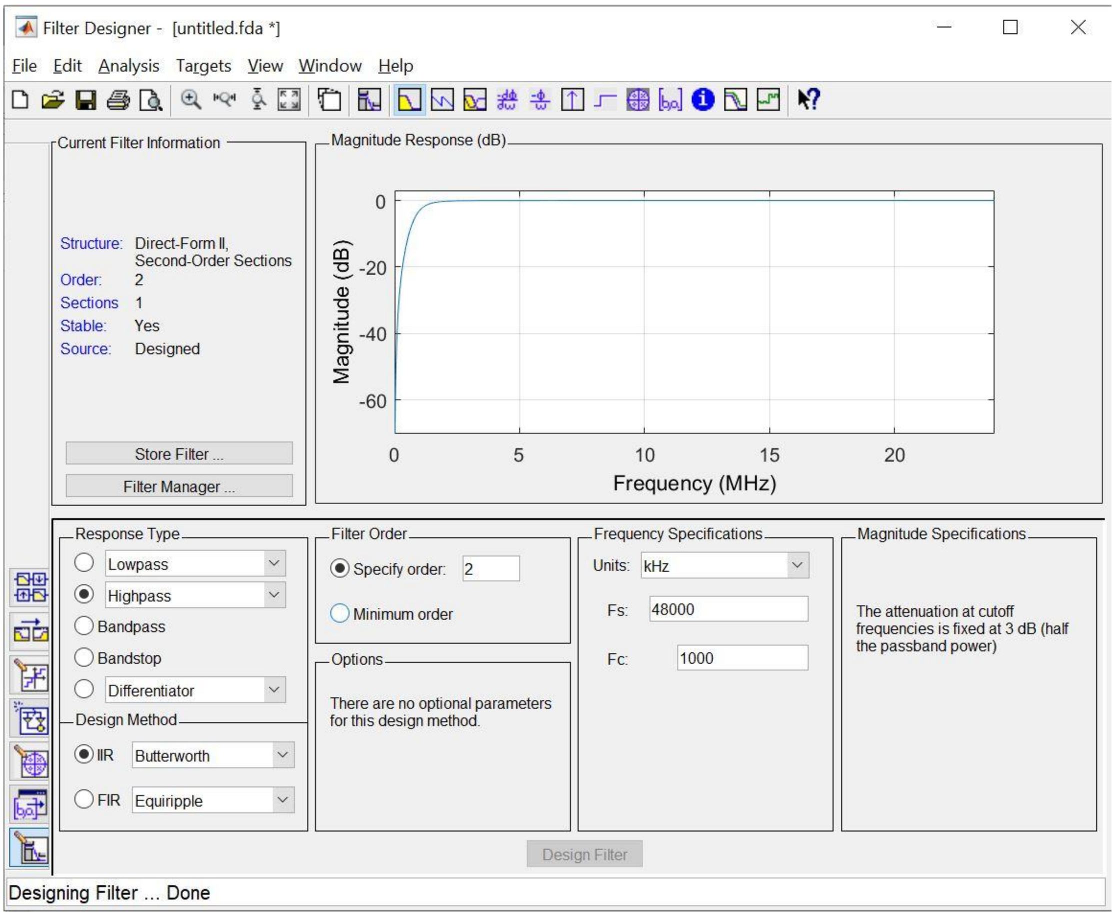
```

#include "fir_2ch_int.h"

struct my_params : hls::ip_fir::params_t {
    static const double coeff_vec[sg_fir_srrc_coeffs_len];
    static const unsigned input_length = LENGTH;
    static const unsigned output_length = LENGTH;
    static const unsigned num_channels = NUM_CHAN;
    static const unsigned num_coeffs = sg_fir_srrc_coeffs_len;
    static const unsigned input_width = INPUT_WIDTH;
    static const unsigned input_fractional_bits = INPUT_FRACTIONAL_BITS;
    static const unsigned output_width = OUTPUT_WIDTH;
    static const unsigned output_fractional_bits = OUTPUT_FRACTIONAL_BITS;
    static const unsigned coeff_width = COEFF_WIDTH;
    static const unsigned coeff_fractional_bits = COEFF_FRACTIONAL_BITS;
    static const unsigned coeff_structure = hls::ip_fir::inferred;
    static const unsigned quantization = hls::ip_fir::quantize_only;
};
const double my_params::coeff_vec[sg_fir_srrc_coeffs_len] = {
   -3.0517578125e-05, -3.0517578125e-05, -3.0517578125e-05,                 0,
   4.57763671875e-05,   6.103515625e-05,                 0,-7.62939453125e-05,
  -0.0001068115234375, -3.0517578125e-05,   0.0001220703125, 0.000213623046875,
  0.0001068115234375,-0.000152587890625,-0.0003814697265625,-0.000335693359375,
     6.103515625e-05, 0.000518798828125, 0.000640869140625, 0.000213623046875,
  -0.000518798828125, -0.00091552734375,-0.000518798828125,     0.00048828125,
      0.001220703125,  0.00079345703125,   -0.000732421875,-0.002105712890625,
  -0.001693725585938,0.0009307861328125, 0.004074096679688, 0.004730224609375,
   0.000885009765625,-0.005950927734375, -0.01077270507813,-0.008087158203125,
    0.00323486328125,  0.01722717285156,  0.02290344238281,  0.01150512695313,
   -0.01542663574219, -0.04301452636719, -0.04855346679688, -0.01406860351563,
    0.06101989746094,   0.1555786132813,   0.2343902587891,   0.2650299072266,
     0.2343902587891,   0.1555786132813,  0.06101989746094, -0.01406860351563,
   -0.04855346679688, -0.04301452636719, -0.01542663574219,  0.01150512695313,
    0.02290344238281,  0.01722717285156,  0.00323486328125,-0.008087158203125,
   -0.01077270507813,-0.005950927734375, 0.000885009765625, 0.004730224609375,
```



```
void dummy_fe(din_t din_i[LENGTH], din_t din_q[LENGTH], din_t out[FIR_LENGTH])
{
    for (unsigned i = 0; i < LENGTH; ++i)
    {
        out[2*i] = din_i[i];
        out[2*i + 1] = din_q[i];
    }
}

void dummy_be(dout_t in[FIR_LENGTH], dout_t dout_i[LENGTH], dout_t dout_q[LENGTH])
{
    for(unsigned i = 0; i < LENGTH; ++i)
    {
        dout_i[i] = in[2*i];
        dout_q[i] = in[2*i+1];
    }
}

void fir_top(din_t din_i[LENGTH], din_t din_q[LENGTH],
             dout_t dout_i[LENGTH], dout_t dout_q[LENGTH])
{
    din_t fir_in[FIR_LENGTH];
    dout_t fir_out[FIR_LENGTH];

    static hls::FIR<my_params> fir1;

    dummy_fe(din_i, din_q, fir_in);
    fir1.run(fir_in, fir_out);
    dummy_be(fir_out, dout_i, dout_q);
}

```

Figure 3 – Example code for filter implementation

The filter coefficients have to be found according to the filter type, order and cutoff frequencies. This is not done inside the C code, rather MATLAB filter designer tool is used to find the filter coefficients beforehand (see Figure 4). These coefficient values are fed manually into the C code.



Structure: Direct-Form II, Second-Order Sections
Order: 2
Sections 1
Stable: Yes
Source: Designed

Response Type: Highpass
Design Method: IIR Butterworth
Filter Order: Specify order: 2
Options: There are no optional parameters for this design method.
Units: kHz
Fs: 48000
Fc: 1000
Magnitude Specifications: The attenuation at cutoff frequencies is fixed at 3 dB (half the passband power)

Designing Filter ... Done

Figure 4 – Using MATLAB filter designer to find HPF coefficients

SIGNAL NORMALISATION

To normalise a signal, a constant amount of gain is generally applied to bring the amplitude to a target level. But if the signal amplitude varies dynamically, application of constant gain does not serve the purpose. Hence, we use automatic gain control (AGC) where the peak output signal level is used to dynamically adjust the gain of the amplifiers. Effectively, AGC lowers the “too high” parts and raises the “too low” parts of the signal. AGC is used in our signal processing unit to overcome the unwanted clutter echoes. This method relies on the fact that clutter returns far outnumber echoes from targets of interest.

We have used two different techniques for signal normalisation:

1. The maximum and minimum envelopes of the signal are detected. For finding the envelope, the local maxima and minima are found. If any maximum/minimum lies below a certain threshold amplitude, then it is neglected. The envelope is then found out by using the Cubic Spline interpolation method. This envelope is used as the peak for deciding the variable gain, which is multiplied with each sample of the signal.
2. The modulus of the sampled signal is taken, and the peaks are found out. The envelope is then found out using the Cubic Spline interpolation method. This envelope is used to find the variable gain. If any value is smaller than a threshold, then it is amplified by the maximum gain to bring it closer to the threshold.

Hardware Implementation:

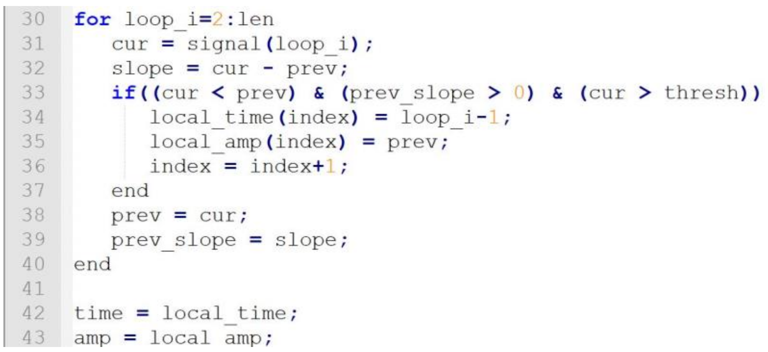
No changes in the original algorithm were required. The challenging task here was to implement the Cubic Spline interpolation. We have directly used the following directory found online:

```
#include <codecogs/maths/approximation/interpolation/cubic.h>
```

Which used a simple function call:

```
Maths::Interpolation::Cubic A(N, x, y);
```

The remaining part of the algorithm involved simple arithmetic operations like finding maxima and minima (as shown in Figure 5).



```
for loop_i=2:len
    cur = signal(loop_i);
    slope = cur - prev;
    if((cur < prev) & (prev_slope > 0) & (cur > thresh))
        local_time(index) = loop_i-1;
        local_amp(index) = prev;
        index = index+1;
    end
    prev = cur;
    prev_slope = slope;
end

time = local_time;
amp = local_amp;
```

Figure 5 – example code for finding local maximas

SIGNAL CONDITIONING

Signal conditioning means manipulating the signal in such a way that it can be further processed. The signal can be amplified, filtered, isolated etc. because not all components of the signal contain valid data. Filtering is the most common method, where the unwanted components can be removed from the frequency domain. Filtering is essentially decomposition and reconstruction of the signal in the frequency domain. Certain methods allow us to do so in the time domain, which include Wavelet Decomposition and Empirical Mode Decomposition. Essentially, we have used three techniques:

1. A simple band pass filter is used to pass only the relevant frequencies of the signal.
2. Wavelet Decomposition is used to find out the wavelet coefficients for different levels. The signal is then reconstructed using only the significant wavelets. Unlike Fourier Transform, instead of decomposing the signal into time unlimited sine waves, it is decomposed into time shifted and scaled, time-limited wavelets.
3. Empirical Mode Decomposition is used to find out the Intrinsic Mode Functions (IMFs) of the signal. The signal is then reconstructed using only the significant IMFs. The first IMF usually carries the high-frequency components; hence, it can be rejected to remove random noise.

BAND PASS FILTER

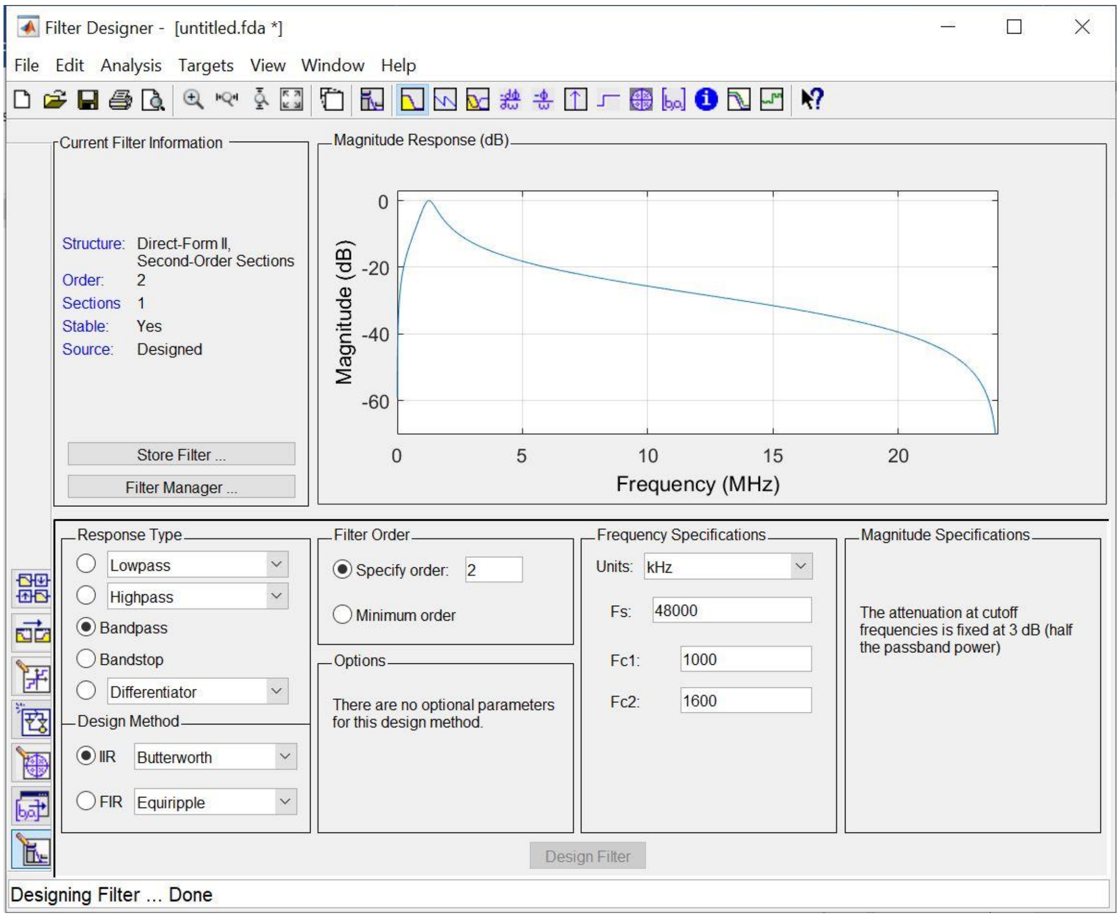
Often bandpass filtering is used to further attenuate the noise relative to the signal to improve the signal-to-noise ratio. We have used butterworth filter of variable order and pass band frequencies for analysis purposes.

For some faults causing a very small reflection, the detection of the fault is very difficult because the amplitude of reflected wave is comparable to the level of noise. The traditional filtering methods may not be able to filter out the noise retaining the small reflected wave corresponding to the fault. Hence, other methods such as Wavelet Decomposition and Empirical Mode Decomposition can be beneficial.

Hardware Implementation:

No changes in the original algorithm were required. An example Vivado HLS project was used for bandpass filter implementation, like before.

As done earlier, the filter coefficients were found out using the MATLAB filter designer tool (see Figure 6) and then fed manually into the C code.



Current Filter Information

Structure: Direct-Form II, Second-Order Sections
Order: 2
Sections: 1
Stable: Yes
Source: Designed

Response Type: ○ Lowpass ○ Highpass [✓ Bandpass] ○ Bandstop ○ Differentiator

Design Method: [✓ IIR] Butterworth ○ FIR Equiripple

Filter Order: [✓ Specify order:] 2 ○ Minimum order

Options: There are no optional parameters for this design method.

Frequency Specifications
Units: kHz
Fs: 48000
Fc1: 1000
Fc2: 1600

Magnitude Specifications: The attenuation at cutoff frequencies is fixed at 3 dB (half the passband power)

Designing Filter ... Done

Figure 6 – Using MATLAB filter designer to find BPF coefficients

WAVELET DECOMPOSITION

The Fourier Transform gives the frequency information of the signal, which means that it tells us how much of each frequency exists in the signal, but it does not tell us when in time these frequency components exist. One way to overcome this short coming is Short Time Fourier Transform (STFT), in which the FT is taken after dividing the signal into small windows. By choosing the size of the window, one may get perfect time resolution or perfect frequency resolution, but not both. The Wavelet Transform (WT) solves the dilemma of resolution to a certain extent. WT implements something called a Multi Resolution Analysis (MRA), which analyses the signal at different frequencies with different resolutions. Every spectral component is not resolved equally as was the case in the STFT.

For computing the Discrete Wavelet Transform (DWT), filter banks are used for construction of the multi-resolution analysis. In DWT, low pass filters (LPF) and high pass filters (HPF) are used. The original signal is passed through the two filters and down-sampled, and the result is called the first level of wavelet transform. This is done iteratively to give detail and approximation coefficients, as shown in Figure 7.

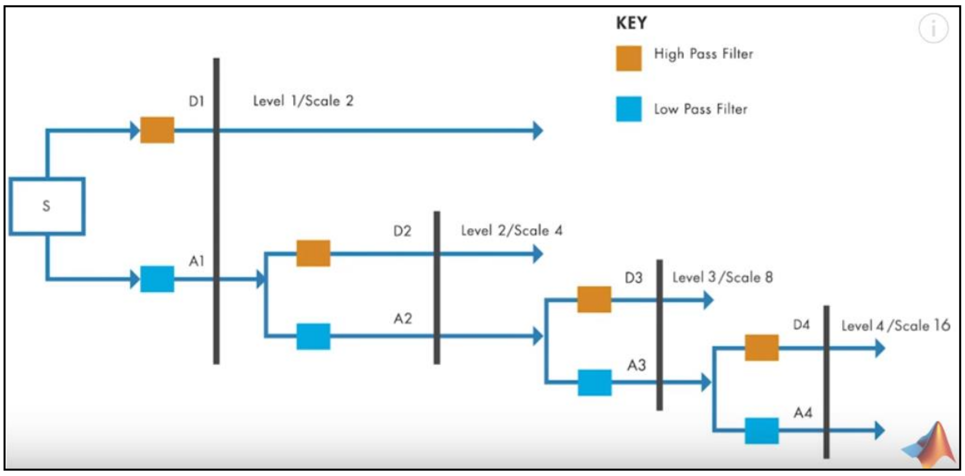


Figure 7 – Computing the Discrete Wavelet Transform (DWT)

Wavelet de-noising technique has been applied in which wavelet transform is used to decompose a signal into N levels of lower resolution components using a mother wavelet. This decomposition results in N detail coefficients and one approximation coefficient. Only some of these coefficients are used and others are discarded for reconstructing the signal.

Hardware Implementation:

The simple wavelet transform can be implemented by set of filters that can divide signals for a particular time into sets of frequency bands. The coefficients of these filters can be derived from the mother wavelet for different translation and scaling factors, as discussed above.

Earlier, the Dual-Tree Complex Wavelet Transform (DTCWT) was proposed which calculates the complex transform of a signal using two separate DWT decompositions (called trees). But some changes in the algorithm are required for an efficient hardware implementation. Hence, the Mallat algorithm (aka the pyramid algorithm), as discussed in the Wavelet Tutorial by Chun-Lin, is used.

It operates on a finite number of data sets. A certain number of input points (collectively called an input block size) are passed through two convolution functions and are then down-sampled to get the output half of the size of the input. These two functions are basically low and high filters.

One half of the output is produced by the low-pass filter function:
$y_{lowpass}[k]=\sum x[n]h0[2k-n]$

And the other half is produced by the high-pass filter function:
$y_{highpass}[k]=\sum x[n]h1[2k-n]$

Here N is the input block size and h0 and H1 are low-pass and high-pass filters respectively. In general, higher-order wavelets (those with more non-zero coefficients) have more relevant information in the low-pass output and then the high-pass output. If the average amplitude of the high-pass filter is below a certain level, it can be neglected while reconstructing the signal without much loss of any useful information in the signal. The high output is called the detail output and the low output is known as approximation output (see Figure 8). Each step of retransforming the low-pass output is called dilation.

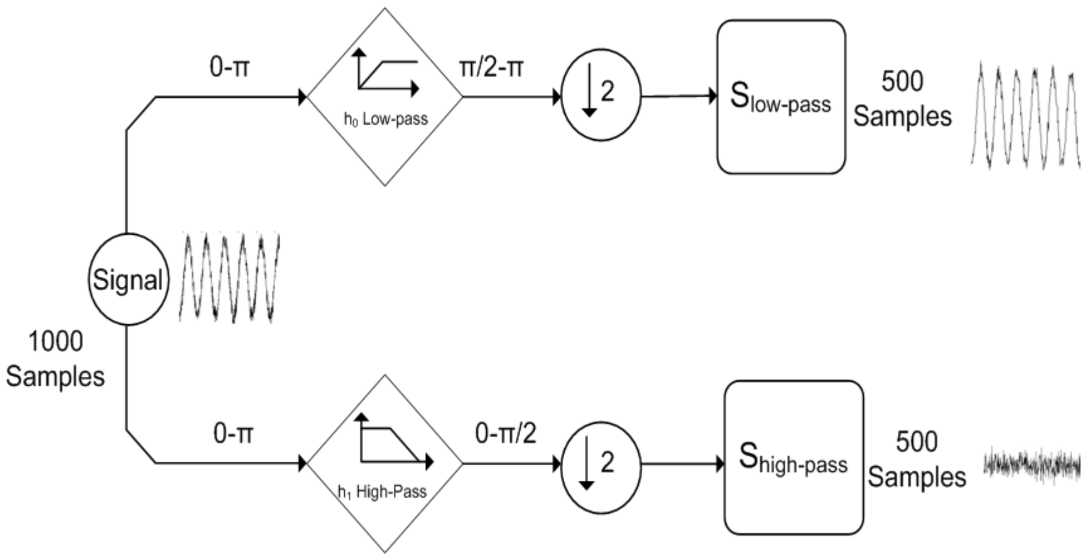


Figure 8 – Decomposition Scheme

As seen in the formula given in the previous section, the coefficients of the filter are convoluted with every alternate sample of data, resulting in a filter output down-sampled by two. If the coefficients of the low-pass filter are used, we will have a half input signal expressed by low-pass output, we need to do the same with high-pass coefficients to get the high-pass output. As we have described before, we can keep just the low-pass signal which has the most representative information of the total signal. For higher order decomposition, we can use the low-pass filter output as the input for the next level. This can be carried till the we have only one sample left. See Figure 9 for a graphical presentation.

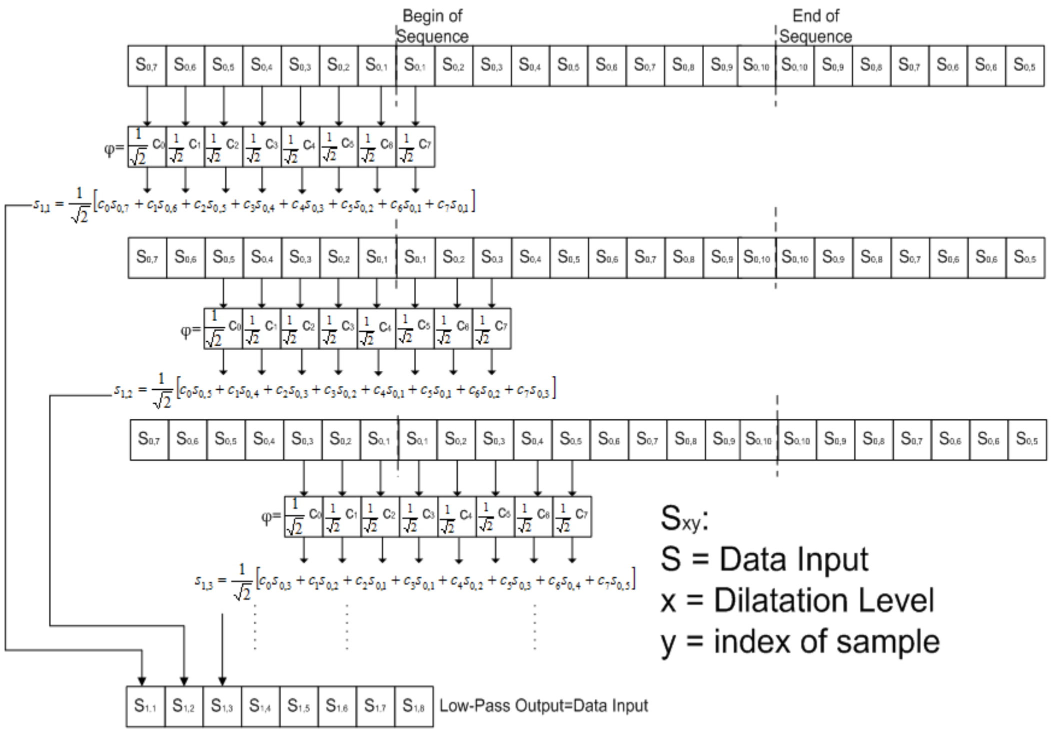


Figure 9 – Steps of the algorithm

The boundary effect (shown in Figure 10) is a common phenomenon seen in DWT and is generally observed when the wavelet function is positioned at the start or the end of the input block. When the convolution function is operating at the start of the sample, and no samples have been provided to avoid boundary effects, the convolution might take place with data that are not in memory space and are essentially garbage value which would either lead to memory overload or drastic misrepresentation of the signal. In this sense, if there is more than one decomposition level, the reconstruction will be inaccurate. To avoid boundary effect, the reflection of the input is generally added to the start and the end of the input. Since it's a part of the signal itself, its properties don't differ much from the signal and don't compromise the signal much.

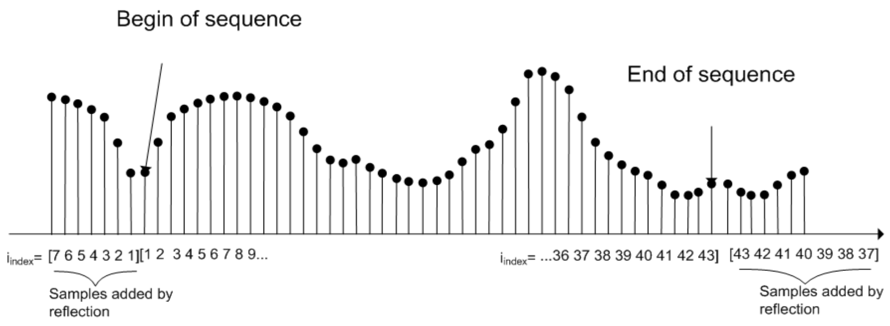


Figure 10 – Boundary effect in DWT

For implementing the DWT in a digital system, following steps are followed:

1) Chose the appropriate mother wavelet according to the need of the application
2) Compute the filter coefficients, using the filter designer in MATLAB
3) Chose the language programming to write the code of DWT that can be later implemented as an HLS project. One of the best options is C code which can work with the IEEE754 format and it is supported by several dedicated devices.
4) Define fixed memory for:
   a) Constants values of wavelet coefficients,
   b) Level of decomposition
   c) Reserve data memory for two arrays
      i) the first one has the same size of input data +2*length filter coefficients
      ii) second one has half size of input data + 2*length filter coefficients (temporal array).

The algorithm is presented as a flow chart in Figure 11.

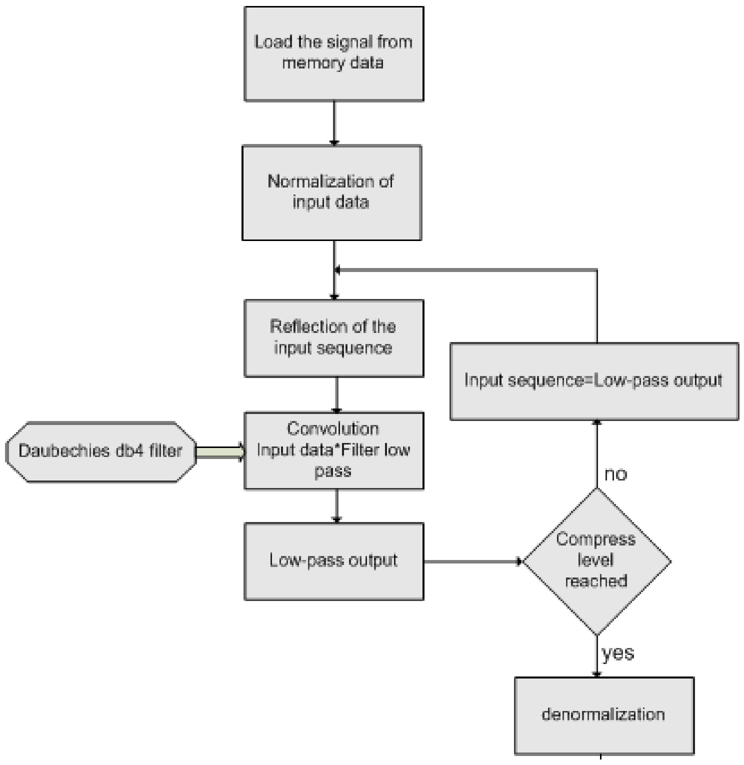
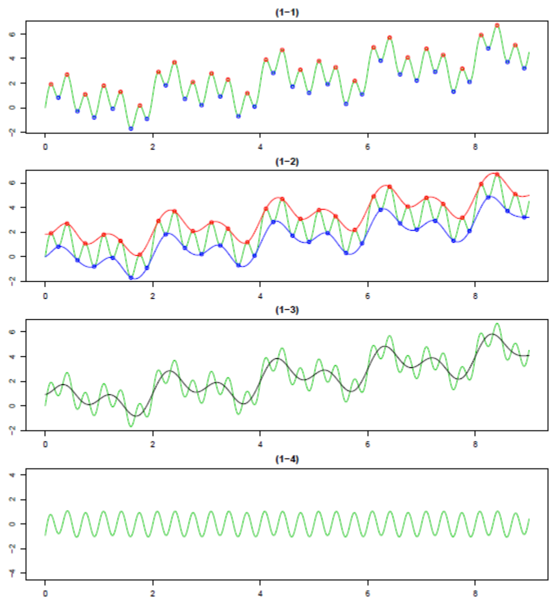

Load the signal from memory data
Normalization of input data
Reflection of the input sequence
Input sequence=Low-pass output
Daubechies db4 filter
Convolution Input data*Filter low pass
no
Low-pass output
Compress level reached
yes
denormalization

Figure 11 – Algorithm Flowchart

EMPIRICAL MODE DECOMPOSITION

Empirical Mode Decomposition (EMD) is a method of breaking down a signal without leaving the time domain. The process is useful for analysing natural signals, which are most often non-linear and non-stationary.

EMD filters out functions which form a complete and nearly orthogonal basis for the original signal. The functions, known as Intrinsic Mode Functions (IMFs), are therefore sufficient to describe the signal, even though they are not necessarily orthogonal. The modes may provide insight into various signals contained within the data. The effective algorithm of EMD is the iteration of the procedure shown in Figure 12.



1-1. First identify the local extrema.

1-2. Consider the two functions interpolated by local maximum and local minimum (upper and lower envelop).

1-3. Their average, envelop mean will yields the lower frequency component than the original signal.

1-4. By subtracting envelop mean, from the original signal $x(t)$, the highly oscillated pattern h is separated.

Figure 12 – One iteration of the Sifting process

EMD is limited in distinguishing different components in narrow-band signals. The narrow band may contain either (a) components that have adjacent frequencies or (b) components that are not adjacent in frequency but for which one of the components has a much higher energy intensity than the other components (see Figure 13).

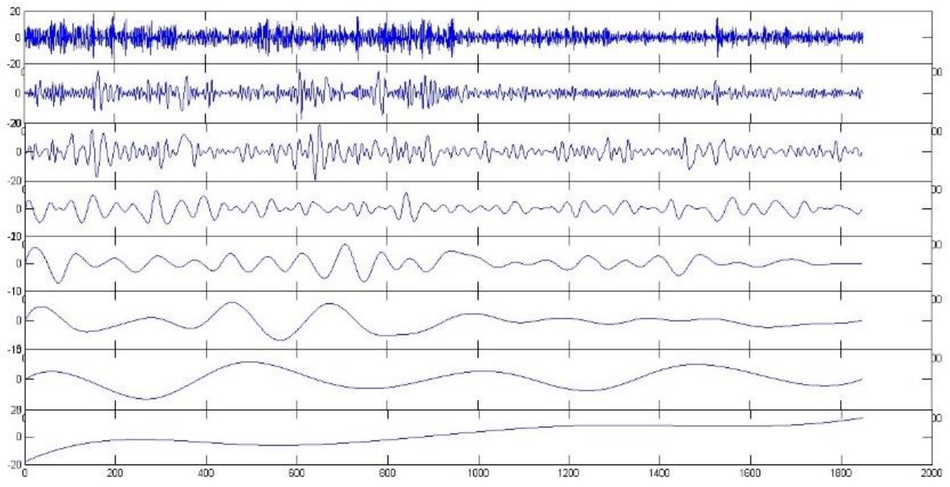


Figure 13 – Several IMFs of a signal

Hardware Implementation:

Some changes are necessary for implementing the complex EMD algorithm on FPGA. Modifications have to be incorporated according to the signal type:

1. **Real signal** – The original algorithm, real EMD can be used directly.
2. **Analytic signal** – The real and imaginary parts have to be treated separately. Hilbert Transform is used to combine the individual outputs.
3. **Complex signal** – The positive and negative frequency components are treated separately. Finally, individual outputs are combined using a different approach.

We need to implement the algorithm for the *analytic signal*. The approach that we were following till now is shown in Figure 14, where we are applying the real EMD algorithm on the real part of the input and its Hilbert Transform separately. This is compared to a standardised approach where real EMD is applied on the real part of the signal, and the Hilbert Transform of the output is taken. Observe the differences in Figure 14.

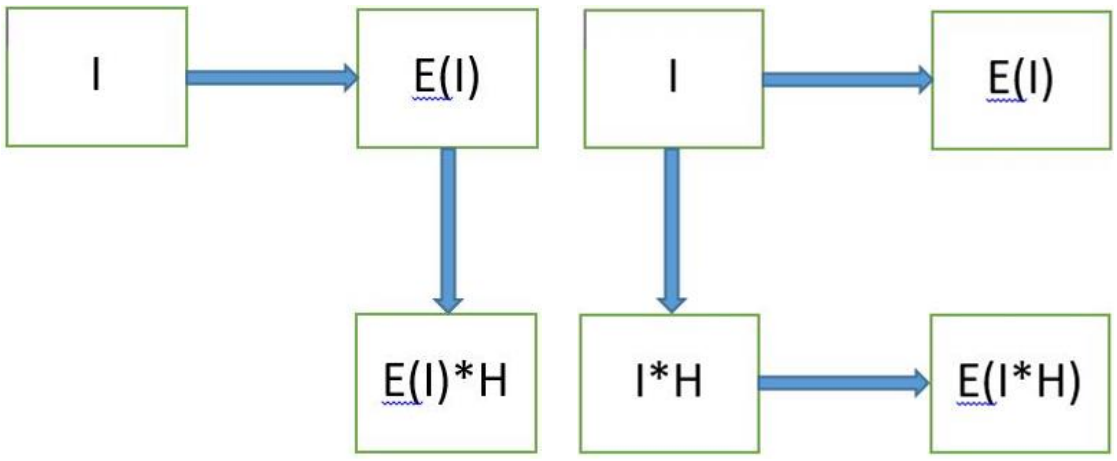


Figure 14 – 'Old Approach' versus 'New Approach'

There is a flaw in the old approach: for both the above methods to be identical, E(I)*H has to be equal to E(I*H), which is not a general rule. The argument is further strengthened by the flowchart shown in Figure 15.

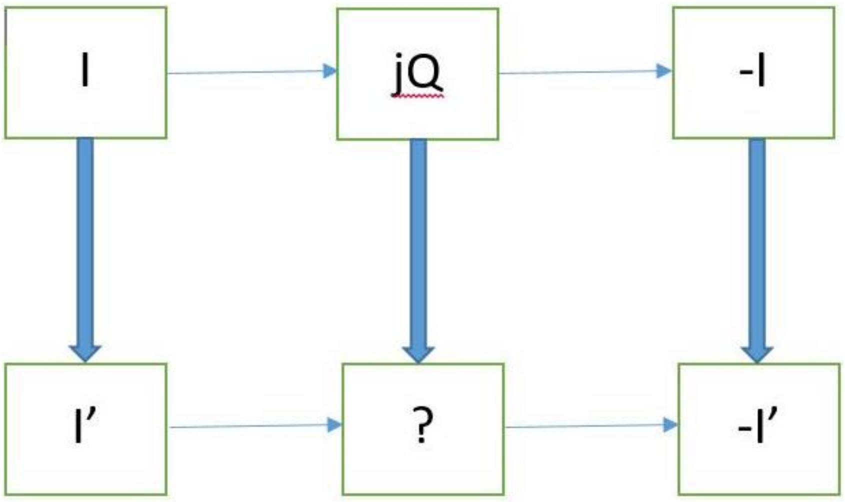


Figure 15 – Flowchart to justify the modification

NOTE for EMD:

Hence, Hilbert Transform and real EMD blocks have to be implemented separately. For the Hilbert Transform block, simple convolution with h[n] is done:

```
for(j=0;j<=i;j++)
    y[i]=y[i]+(x[j]*h[i-j]);
```

For the real EMD implementation, simple arithmetic operations like finding maxima, minima, and mean are done. This is very similar to the code presented in Figure 5.

SPECTROGRAM

A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. The spectrogram is basically the output of the STFT. The Short-Time Fourier Transform (STFT), is a Fourier-related transform used to determine the frequency spectrum in local sections of time called windows.

Fourier transform is performed on a short number of FFT points in a window. The larger the size of the window, the resolution in frequency domain increases but time domain becomes hazy. The windows can be of multiple shapes; the windows used by us are Hanning, Hamming, Blackman, and Bartlett.

Hardware Implementation:

This has not been completed yet.

REFERENCES

Wavelet Decomposition:

The Wavelet Tutorial by Robi Polikar

Mallat Algorithm Tutorial by Chun-Lin, Liu

Empirical Mode Decomposition:

Introduction to EMD with application to a scientific data by Donghoh Kim

Miscellaneous:

codecogs website for Cubic Spline Interpolation
another website for finding Maxima and Minima
Wikipedia for technical definitions