



M. S. Ramaiah University of Applied SCIENCES

University House, Gnanagangothri Campus,
New BEL Road, MSR Nagar, Bangalore - 560 054.
KARNATAKA, INDIA

Tel: +91 80 4536 6666 | Fax: +91 80 4536 6677 | Email: uas@msruas.ac.in
Web : www.msruas.ac.in



**Group Project Title : VHDL IMPLEMENTATION OF EFFICIENT SSB
USING DIFFERENTIAL PHASE SHIFTER**

Project Team :

Sl. No.	Reg. No.	Student Name	Department
1.	17ETEC004097	Samanwaya Das	ECE
2.	17ETEC004080	Prakhar Vishwakarma	ECE
3.	17ETEC004082	Preethi H S	ECE
4.	17ETEC004119	Swapna H	ECE

**B-Tech
3rd year**

Faculty of Engineering and Technology

**M. S. Ramaiah University of Applied SCIENCES
Bengaluru - 560054**

B-Tech
2017

VHDL IMPLEMENTATION OF EFFICIENT SSB USING DIFFERENTIAL PHASE SHIFTER



Project Team

	Reg. No.	Student Name
1	17ETEC004097	Samanwaya Das
2	17ETEC004080	Prakhar Vishwakarma
3	17ETEC004082	Preethi H S
4	17ETEC004119	Swapna H

B.Tech Supervisor: Dr. Punithavathi D

FACULTY OF ENGINEERING AND TECHNOLOGY

M.S.RAMIAH UNIVERSITY OF APPLIED SCIENCES

Bengaluru -560058

Certificate

This is to certify that the Project titled “VHDL implementation of efficient SSB modulation using differential phase shifter” is a bonafide work carried out in the Department of Electronics and Communication Engineering by Samanwaya Das, Prakhar Vishwakarma, Preethi H S and Swapna H, bearing Reg. No.17ETEC004097 , 17ETEC004080, 17ETEC004082 and 17ETEC004119 in partial fulfilment of requirements for the award of B.Tech Degree in Electronics and communication Engineering of M.S. Ramaiah University of Applied Sciences.

Supervisor: Dr.Punithavathi D

Head – Dept.of ECE

Dean-FET

Declaration

VHDL implementation of efficient SSB modulation using differential phase shifter

The project work is submitted in partial fulfilment of academic requirements for the award of **B. Tech.** Degree in the **Department of Electronics and communication Engineering** of the Faculty of **Engineering and Technology** of M.S.Ramaiah University of Applied Sciences. The project report submitted here with is a result of our own work and in conformance to the guidelines on plagiarism as laid out in the University Student Handbook. All sections of the text and results which have been obtained from other sources are fully referenced. We understand that cheating and plagiarism constitute a breach of University regulations, hence this project report has been passed through plagiarism check and the report has been submitted to the supervisor.

	Reg. No.	Student Name	Signature
1	17ETEC004097	Samanwaya Das	
2	17ETEC004080	Prakhar Vishwakarma	
3	17ETEC004082	Preethi H S	
4	17ETEC004119	Swapna H	

Date:

Acknowledgements

We would like to express our most profound thankfulness to each one of the individuals who gave us the likelihood to finish this product. A special gratitude we give to our major project supervisor, **Dr. Punithavathi D** whose commitment in invigorating proposals and consolation, helped us to organize our venture particularly in composing this prototype.

We are thankful to ,Dr.S.Malathi H.O.D of Electronics and communication Engineering, for providing us an opportunity to implement efficient SSB modulation using VHDL and obtain the best results.

Our grateful regards to Professors , Associate Dean and ,Dean, Faculty of Engineering and Technology ,M.S.Ramaiah University of Applied Sciences, Bangalore, for their constant support and motivation.

We take the opportunity to thank all our lectures and staff of Electronics Laboratory, who have directly or indirectly helped our project. We pay our respects and love to our parents and all other family members and friends for their love encouragement throughout our career. Last but not the least we express our thanks to our friends for their cooperation and support.

Table of Contents

1.Introduction.....	11
2.Aim and objections.....	13
3.SSB modulation	15
3.1.Literature survey.....	15
3.2. Conventional SSB architecture	16
3.2.1.Phasing method	16
3.2.2.MATLAB implementation of conventional method	16
3.2.3 Coefficients of Hilbert transform	17
3.3 .Proposed SSB architecture.....	18
3.3.1.Dual filter.....	18
3.3.2.MATLAB implementation of dual filter method.....	18
3.4.Comparsion between Hilbert coefficients and dual filter	21
4.Filter characteristics.....	22
5.Results and conclusion from MATLAB simulation	24
5.1. conventional method	24
5.2.Dual filter method	27
6.VHDL implementation of hardware simulation.....	32
7.Objectivies achieved and conclusion	54
7.1.Simulating the Hartley architecture and proposed methodology in MATLAB	54
7.1.1. conventional.....	54

7.1.2.Dual phase shifter	54
7.2.Obtainig the filter characteristics	54
7.3.Implemenation the architecture using VHDL	54
8.Appendix.....	57
8.1.conventional method	57
8.2.MATLAB code to generate dual fliter coefficients	59
8.3.MATLAB code for dual filter	63
8.4.MATLAB code for finding filter characteristics	65
8.5.VHDL code.....	66
8.6.MATLAB code to read text file and plot the spectrum.....	80

List of Tables

Table	Page no
Table 3.1: Coefficients of Hilbert transform	17-18
Table.3.2: Coefficients of A and B	19-21
Table3.3: Comparisons between Hilbert and dual filter coefficients	21

List of Figures

Figure

Figure.3.1:Block diagram of phasing method SSB.....	16
Figure 3.2:Block diagram of dual filter	18
Figure 4.1: Magnitude and phase of Hilbert filter(a and b).....	22
Figure 4.2: +45 degree filter characteristics	23
Figure 4.3: -45 degree filter characteristics	23
Figure 5.1:Message signal of conventional method	25
Figure 5.2: output of both the signal	25
Figure 5.3: SSB modulation	26
Figure 5.4 : Side band spectrum of conventional method	26
Figure 5.5:Single side amplitude spectrum $X(t)$	27
Figure 5.6: Message signal of dual filter	28
Figure 5.7: Dual output	29
Figure 5.8: SSB modulation	29
Figure 5.9: Spectrum of $X(t)$	30
Figure 5.10: Single sided amplitude spectrum $X(t)$	31
Figure 6.1: Project summary	32
Figure 6.2 : Clock wiz ip	34
Figure 6.3: clock wiz output of MMCM.....	35

Figure 6.4: message signal ip	36
Figure 6.5: Carrier signal ip.....	37
Figure 6.6: multiplier ip	38
Figure 6.7: Dual fliter ip.....	39
Figure 6.8:Hilbert FIR filter ip	40
Figure 6.9: LPF FIR ip	41
Figure 6.10:LPF response.....	41
Figure 6.11 :Final ip hierarchy	42
Figure 6.12:Synthesis completed	43
Figure 6.13: Final RTL schematic.....	44
Figure 6.14: wave window.....	45
Figure 6.15: Reset	46
Figure 6.16: clocking 200MHz_P LVDS	47
Figure 6.17:Clocking 200MHz_n LVDS	48
Figure 6.18: Waveform generated	49
Figure 6.19: SSB modulated output.....	49

Figure 6.20: Text file import to MATLAB	50
Figure 6.21: Text file import to MATLAB	51
Figure 6.22: Single side spectrum of conventional method	52
Figure 6.23:Single side spectrum of dual filter	53

1. Introduction

As Single sideband modulation or SSB is derived from amplitude modulation (AM) and SSB modulation overcomes a number of the disadvantages of AM.

Single sideband modulation is normally used for voice transmission, but technically it can be used for many other applications where two-way radio communication using analogue signals is required.

As a result of its widespread use there are many items of radio communication equipment designed to use single sideband radio including: SSB receiver, SSB transmitter and SSB transceiver equipment.

Many users requiring two-way radio communication will use single sideband and they range from marine applications, generally HF point to point transmissions, military as well as radio amateurs or radio hams.

Currently this modulation technique can be extended to assisting various DSP applications. Some applications include, the body area network communication, short-range communication and underwater communication. Using SSB, as stated earlier can overcome various limitations over conventional AM transmission.

One of the most widely, implemented methods for SSB generation is Hartley modulator. This architecture employs a “Hilbert Transform” filter. The Hilbert Transform, basically gives a 90-degree phase shift. The project explored here, also uses the Hartley architecture. This project explores the limitations of this architecture, and implements a method which employs a dual filter approach to tackle the limitations.

For understanding purpose, the architecture already existing and the one proposed have been simulated using MATLAB. MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.

The simulations after being conducted in MATLAB is implemented using VHDL. VHDL (VHSIC-HDL) (Very High Speed Integrated Circuit Hardware Description Language) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-

programmable gate arrays and integrated circuits. VHDL can also be used as a general-purpose parallel programming language. VHDL implementation allows for actual hardware level simulation, which can then be implemented using FPGA (Field Programmable Gate Array), and later stages to ASIC (Application Specific Integrated Circuit).

2. Aim and Objectives

This chapter should contain the following:

Preamble to the Chapter

- **Title**
- ❖ VHDL Implementation of efficient SSB modulation using Differential phase shifter
- **Aim**
- ❖ To design a differential phase shifter for the efficient SSB modulation and to implement it using VHDL
- **Objectives**

Methods and Methodology/Approach to attain each objective

Objective No.	Statement of the Objective	Method/ Methodology	Resources Utilised
1	To review the literature on existing SSB modulation techniques	<ul style="list-style-type: none">• To collect literature on existing SSB modulation from IEEE indexed journals/reference books.	IEEE conference papers and journal papers.

2	To determine the filter coefficients for the design of the filter using MATLAB	<ul style="list-style-type: none"> To design the filter using MATLAB 	MATLAB
3	To design SSB modulator	<ul style="list-style-type: none"> To design SSB modulation using modulator using MATLAB 	MATLAB
4	To implement the modulator in VHDL and verification using MATLAB	<ul style="list-style-type: none"> To implement the modulator using VHDL To verify using MATLAB 	Xilinx 14.7 and MATLAB
5	To compare the outputs	<ul style="list-style-type: none"> To compare and verify of the outputs and also to test the efficiency 	-----

3. SSB Modulation

3.1 Literature survey :

Single sideband, SSB modulation is basically a derivative of amplitude modulation, AM. By removing some of the components of the ordinary AM signal it is possible to significantly improve its efficiency.

It is possible to see how an AM signal can be improved by looking at the spectrum of the signal. When a steady state carrier is modulated with an audio signal, for example a tone of 1 kHz, then two smaller signals are seen at frequencies 1 kHz above and below the main carrier.

If the steady state tones are replaced with audio like that encountered with speech or music, these comprise many different frequencies and an audio spectrum with frequencies over a band of frequencies is seen. When modulated onto the carrier, these spectra are seen above and below the carrier.

As the carrier is not transmitted, this enables a 50% reduction in transmitter power level for the same level of information carrying signal. [For an AM transmission using 100% modulation, half of the power is used in the carrier and a total of half the power in the two sideband - each sideband has a quarter of the power (Rectangular carrier).]

SSB takes advantage of the fact that the entire original signal is encoded in each of these "sidebands". It is not necessary to transmit both sidebands plus the carrier, as a suitable receiver can extract the entire original signal from either the upper or lower sideband. There are several methods for eliminating the carrier and one sideband from the transmitted signal. Producing this single sideband signal is too complicated to be done in the final amplifier stage as with AM. Conventional amplitude-modulated signals can be considered wasteful of power and bandwidth because they contain a carrier signal and two identical sidebands. Therefore, SSB transmitters are generally designed to minimize the amplitude of the carrier signal. When the carrier is removed from the transmitted signal, it is called suppressed-carrier SSB.

Single side-band (SSB) technologies are especially useful in optical fiber communication systems,

such as higher density wavelength multiplexing and long-haul fiber transmission due to less nonlinear optical effects, because of the reduced optical power.

3.2 Conventional SSB Architecture:

3.2.1 Phasing method (Hartley modulator)

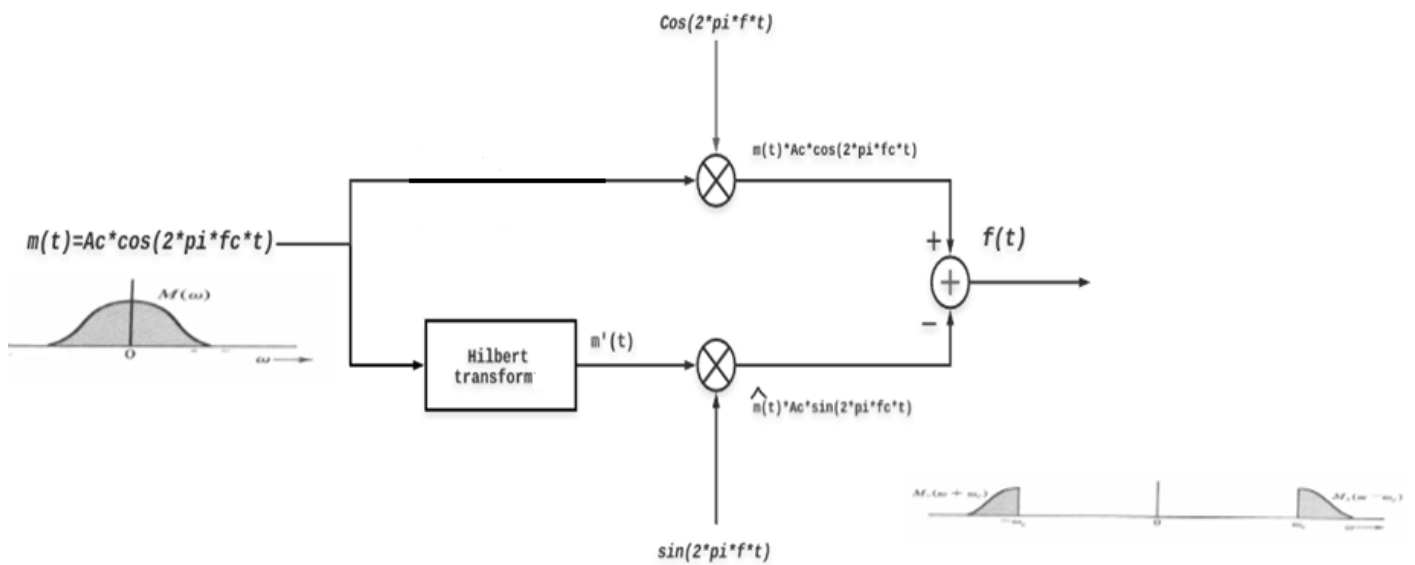


Figure 3.1 block diagram of phasing method SSB

By analysing the architecture (The above architecture is implemented for discrete signals) we see that the message signal $m[n]$ is divided into 2 halves, one part of the message signal is given to the Hilbert transform. The function of Hilbert transform is given as $H(f) = -\delta \text{sgn}(f)$. The phase difference found between the Hilbert transform and the message signal should ideally be 90 degree, but due to hardware limitations it is not exactly 90 and the amplitudes of both the signals are not the same. The output of Hilbert transform and message is multiplied with carrier signal, with a sine and a cosine component respectively. The obtained signals are added to give SSB modulation. Here some error in SSB occur because the Hilbert transform is used to generate 90 degree phase shift, in practice when implemented using this would give us some error at the hardware level.

3.2.2 MATLAB implementation of conventional method

A message signal of 1kHz and carrier signal of 5kHz for ease of plotting and representation. Now to generate the coefficients of the Hilbert transform filter a program is used (Iowa Hills Hilbert Filter Designer). The above filter coefficients have been generated by using a raised cosine window. The design is specific to the clock frequency/sampling frequency which is 49Khz.

The Hartley architecture for the conventional method has 2 paths which end up getting modulated. For the first path there is no phase shift and the signal is directly modulated, and the second path we use the Hilbert transform to get a phase shift of 90 degree and then modulate this with the carrier. A MATLAB code which finds the phase difference between these 2 paths has been used. To find the phase difference between the two-signal, fast Fourier transform (fft) of Hilbert transform output and the message signal (can be referred to in APPENDIX 8.1) has been obtained. The filter outputs are then multiplied with the carrier signal (Sine and Cosine). The obtained signals are added to give SSB modulation. SSB modulation time domain and the frequency domain plots are obtained.

Coefficients	H
-0.003098750403898756	A(0)
-0.020135571740449009	A(1)
-0.020939191991494636	A(2)
-0.000202907720244840	A(3)
0.026952793338152693	A(4)
0.034346295140573725	A(5)
0.007402212025804578	A(6)
-0.038808195575967708	A(7)
-0.063568924468410643	A(8)
-0.027845568148826612	A(9)
0.073581669737100258	A(10)
0.200516197928349321	A(11)
0.288205417056360269	A(12)
0.288205417056360269	A(13)
0.200516197928349321	A(14)
0.073581669737100258	A(15)
-0.027845568148826612	A(16)
-0.063568924468410643	A(17)
-0.038808195575967708	A(18)
0.007402212025804578	A(19)

0.034346295140573725	A(20)
0.026952793338152693	A(21)
-0.000202907720244840	A(22)
-0.020939191991494636	A(23)
-0.020135571740449009	A(24)
-0.003098750403898756	A(25)
-0.003098750403898756	A(26)

Table 3.1 coefficients of heilbert transform

. The above filter coefficients have been generated by using a raised cosine window

3.3 : Proposed SSB architecture:

3.3.1 Dual filter (45 degree quadrature):

The dual differential filter solves SSB generation problem which persists with conventional Hartley architecture. The details of which are explained below;

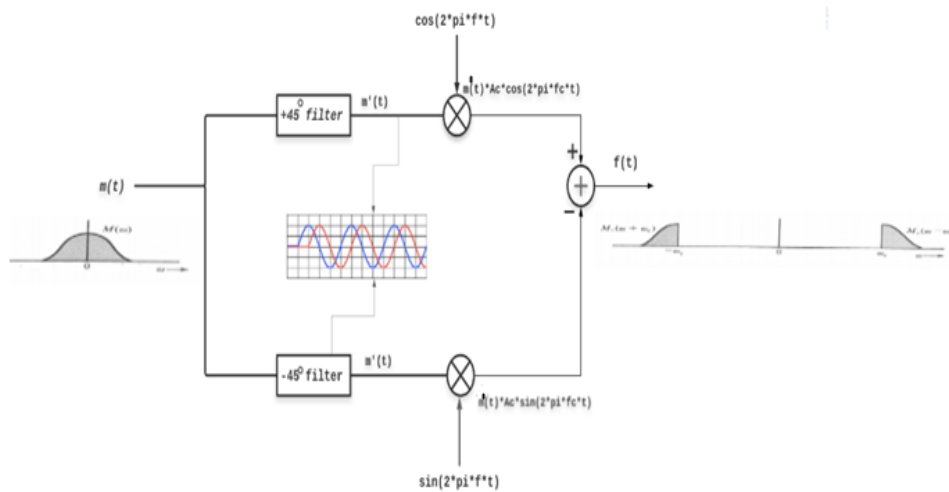


Figure 3.2 block diagram of dual filter

By analysing the architecture (The above architecture is implemented for discrete signals) we see that the message signal $m[n]$ is divided into 2 half , one part of the message signal is given to the $+45$ degree filter and the other part is given to -45 degree filter .The phase and the amplitude of the filter at the output when measured gives us about 90 degree phase shift and the amplitude are

same. So, the output of these filters is multiplied with sin and cos carrier signal. The obtained signals are added to give efficient SSB modulation. It is efficient because since the phase shift is 90 and amplitudes is almost the same there will be less chances of error to happen.

3.3.2.MATLAB implementation of dual filter method :

A message signal of 1kHz and carrier signal of 5kHz are considered for ease of plotting and representation. Now to generate the coefficients of the 2 filter we use a MATLAB code (Refer APPENDIX). Here we observe that filter B coefficients are flipped version of coefficients of filter A. Which essentially suggest similar filter characteristics, which would be shown in the filter characteristics section. The design is specific to the clock frequency/sampling frequency which is 49Khz.

The architecture for the dual filter is slightly modified version of Hartley architecture, which has been described in the block diagram section. This method has 2 paths which end up getting modulated. For the first path there is one filter which provides +45-degree phase shift after which the signal is modulated, and the second path is similar to the first, except for a -45-degree phase shift. A MATLAB code which finds the phase difference between these 2 paths has been used. To find the phase difference between the two-signal, fast Fourier transform (fft) of Hilbert transform output and the message signal (can be referred to in APPENDIX 8.3) has been obtained. The filter outputs are then multiplied with the carrier signal (Sine and Cosine). The obtained signals are added to give SSB modulation. SSB modulation time domain and the frequency domain plots are obtained

Coefficients of filter A and filter B

Filter coefficients	A
0.0000	A(0)
-0.0000	A(1)
-0.0000	A(2)
-0.0103	A(3)
-0.0000	A(4)
-0.0336	A(5)
0	A(6)
-0.0613	A(7)
0.0000	A(8)

-0.0685	A(9)
-0.0000	A(10)
-0.0000	A(11)
0.0000	A(12)
0.7466	A(13)
0	A(14)
-0.3952	A(15)
0	A(16)
-0.1540	A(17)
0.0000	A(18)
-0.0507	A(19)
0.0000	A(20)
0.0000	A(21)
0.0000	A(22)
0.0149	A(23)
0.0000	A(24)
0.0109	A(25)
0	A(26)
0.0032	A(27)

Filter coefficients	B
0.0000	B(27)
-0.0000	B(26)
-0.0000	B(25)
-0.0103	B(24)
-0.0000	B(23)
-0.0336	B(22)
0	B(21)
-0.0613	B(20)
0.0000	B(19)
-0.0685	B(18)
-0.0000	B(17)
-0.0000	B(16)
0.0000	B(15)
0.7466	B(14)
0	B(13)
-0.3952	B(12)
0	B(11)
-0.1540	B(10)
0.0000	B(9)
-0.0507	B(8)
0.0000	B(7)
0.0000	B(6)

0.0000	B(5)
0.0149	B(4)
0.0000	B(3)
0.0109	B(2)
0	B(1)
0.0032	B(0)

Table 3.2. coefficients of A and B

The above are the obtained coefficients, which are generated by the MATLAB code. The code is referenced in the APPENDIX 8.2.

3.4 COMPARSION BETWEEN HILBERT COEFFICIENTS AND DUAL FILTER COEFFICIENTS

Hilbert	DUAL
The coefficients are generated using a program for designing FIR Hilbert filters.	The coefficients are generated by writing a MATLAB code.
Number of coefficients (27).	Number of coefficients (28).
Single filter used.	The coefficients of one filter is flipped coefficients of another.
Using Hilbert transform it is practically impossible to get unity gain in Hardware.	Using dual filter, it is possible to get matched gain for both the parts of the signal.

Table 3.3. comparison between Hilbert and dual filter coefficients

4. Filter characteristics

The filter coefficients and the corresponding FIR filters are characterized using MATLAB.

First for the Hilbert FIR filter, the filter characteristics is found out.

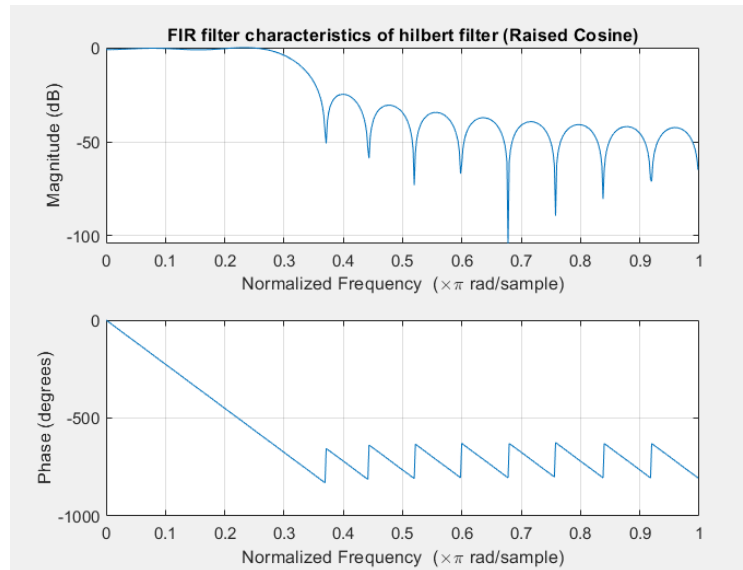


Figure 4.1 (a) magnitude and phase of Hilbert filter

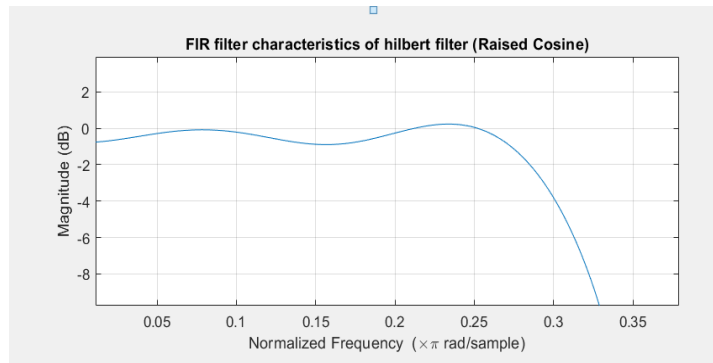


Figure 4.1 (b) Shows the ripples in pass band

The above figures signifies, the magnitude and phase of the filter. The MATLAB command `freqz` is used which converts, the coefficients provided into the Transfer function, and provides us with the response. The above filter is designed for 49Khz sampling frequency using Iowa Hills Hilbert Filter Designer. Upon close examination we find that there are ripples in the passband. (Refer APPENDIX

8.4)

Second for the +45/-45-degree dual filters characteristics are found out..

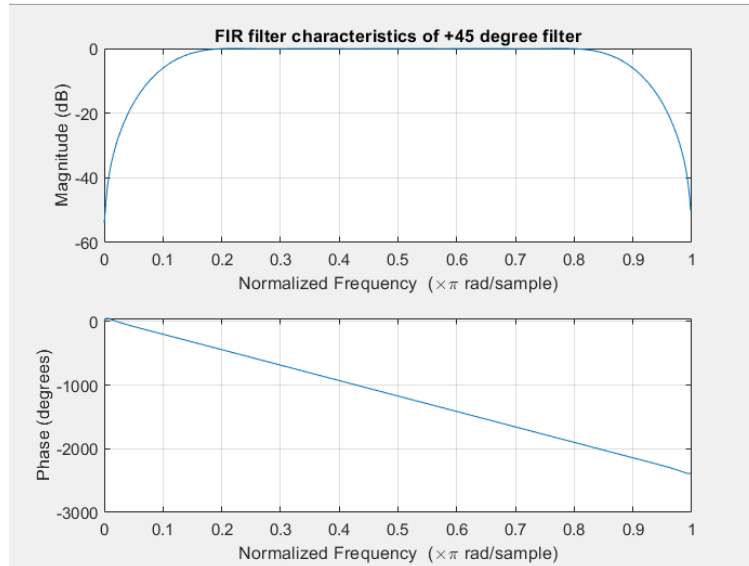


Figure 4.2 +45 degree filter characteristics

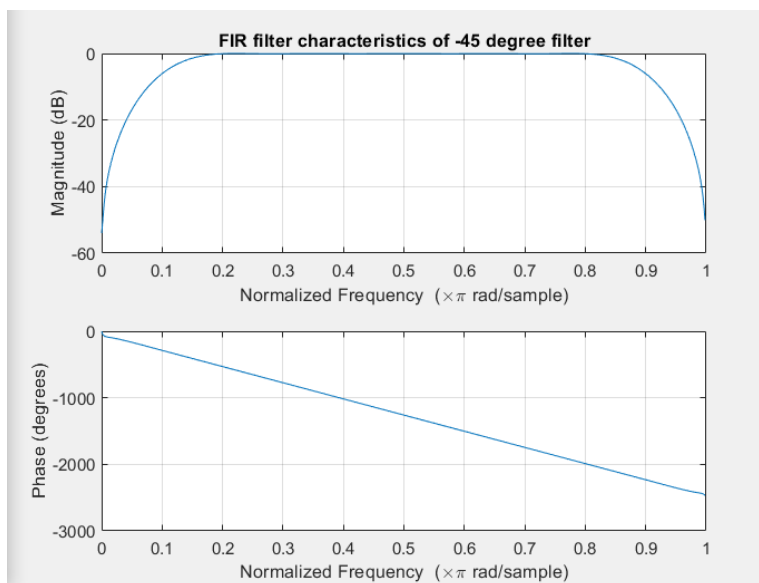


Figure 4.3 -45 degree filter characteristics

The above figure signifies, the characteristics of +45/-45-degree dual filter. Upon, close observation and superimposing the phase response we would observe a net phase difference of 90 degree throughout the band. These filters, have identical magnitude spectrum which essentially gives identical magnitude at the end of the 2 filter stages. The MATLAB command freqz is used which

converts, the coefficients provided into the Transfer function, and provides us with the response.
(Refer APPENDIX 8.4)

5.RESULTS and CONCLUSION from MATLAB simulation

freqz The MATLAB command window output is provided, with the corresponding conclusions. The code is referenced at the (APPENDIX 8.1,8.2,8.3).

5.1 Conventional method:

Results

N1 =27

N =401

Nb =401

mag_X1 = 227.9301

idx_X1 =11

mag_Y1 =245.0811

idx_Y1 =11

phase_lag_diff1 = 91.0486

The results obtained above, are from the MATLAB code, which gives us the magnitudes of the signal tracing the path of Hilbert filter and without it. Also, it provides us with the phase difference of the same.

Conclusion from simulation

From the results obtained, we can observe that phase difference is not exactly 90 degree and the magnitudes are also not the same, i.e 227.9301 and 245.0811 respectively. These differences in the amplitude and phase difference not being equal to 90 degree (91.0486) contribute to erroneous SSB modulation.

Graphs obtained

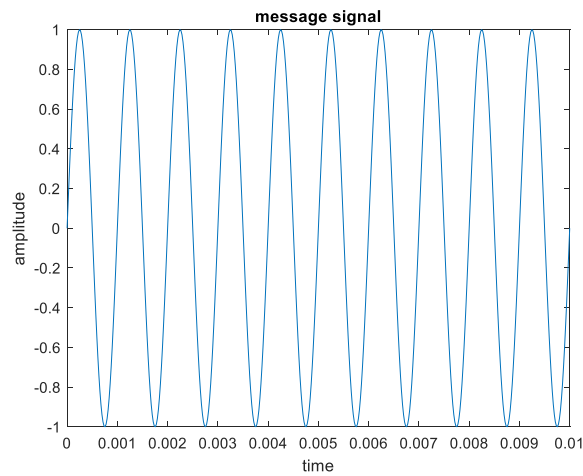


Figure 5.1 message signal of conventional method

The fig3 above show the graph of the message signal having frequency 1KHz and amplitude 1. This would be modulated with the carrier (5Khz).

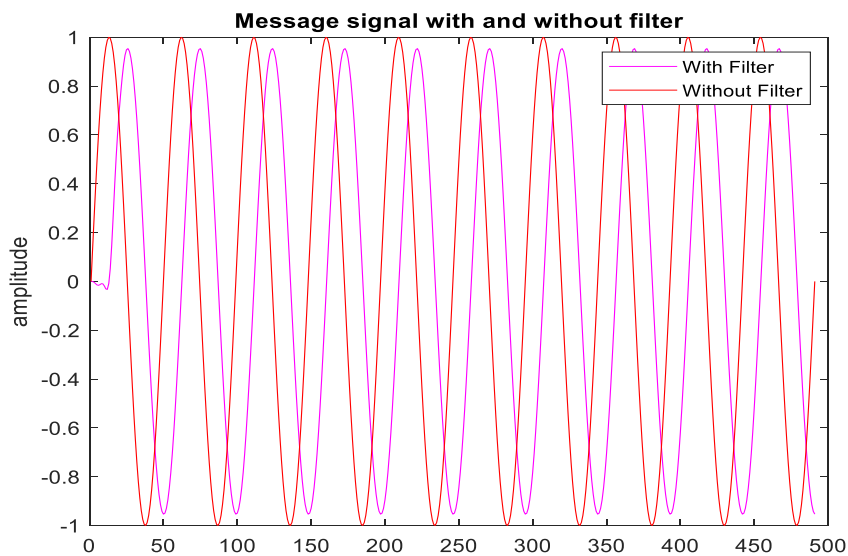


Figure 5.2 output of both the signals

The above graph shows us output of the Hilbert transform and the message signal. Also, it can be observed that the magnitude of the two filters are different and with a phase shift which can be perceived to be of approximately 90 degrees.

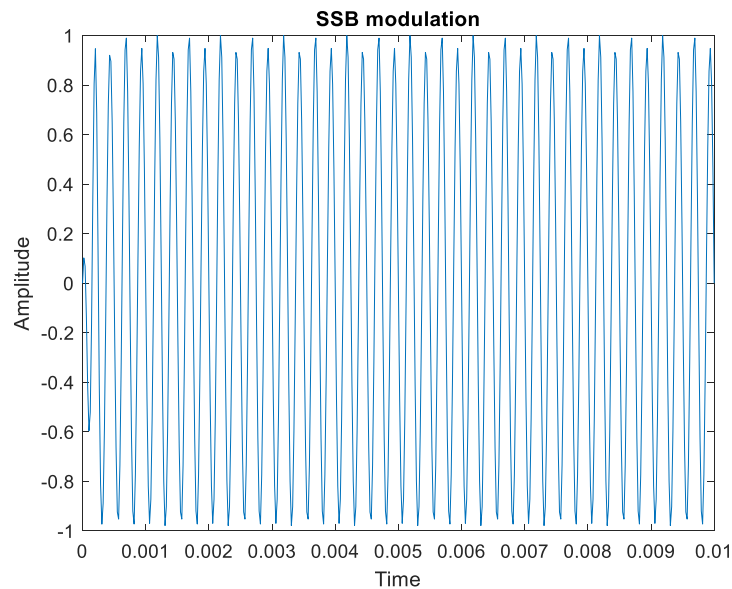


Figure 5.3 SSB modulation

The above graph shows the SSB obtained in time domain. It can be observed that, due to one of the components of SSB side bands not being completely suppressed we do not get a perfect sine wave.

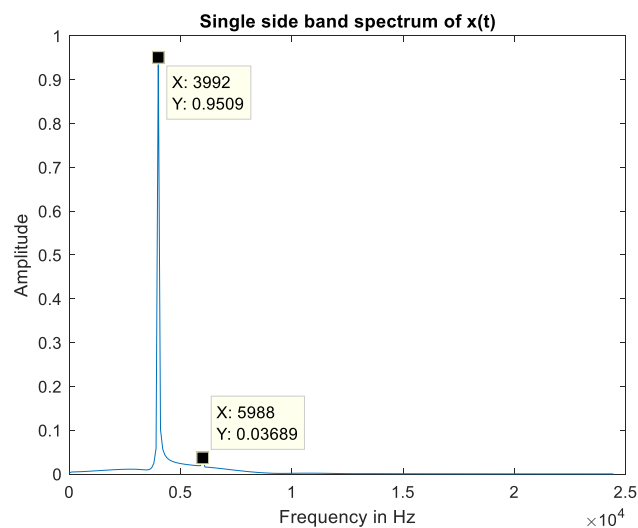


Figure 5.4 side band spectrum of conventional method

As one of the sidebands gets suppressed, this graph shows the lower side band of SSB (fc-fm) i.e the dominant side band. So, for a "lower sideband" mode of SSB transmission, the transmitted signal

would have the spectrum shown above. In this case, the lower frequency edge of the baseband signal has been translated in frequency so that it is located at 4 kHz.

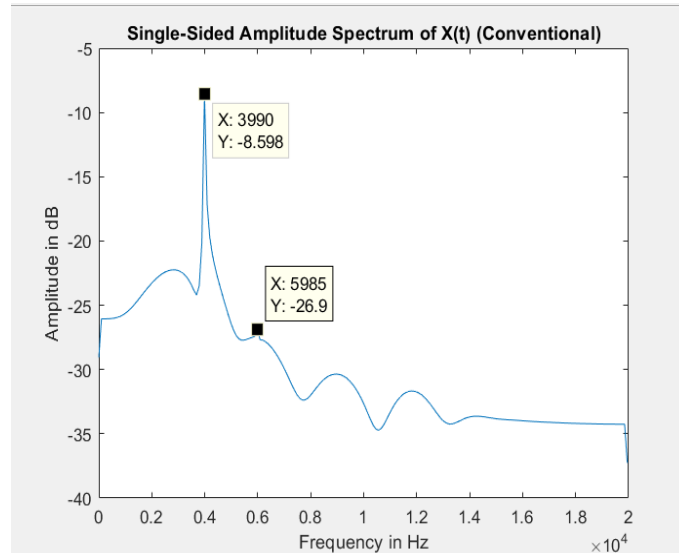


Figure 5.5 single sided amplitude spectrum X(t)

The above graph shows the separation between two side band in dB scale. The suppression is around 19 dB between lower and upper side band.

5.2 Dual filter method:

The MATLAB command window output is provided, with the corresponding conclusions. The code is referenced at the (APPENDIX 8.1,8.2,8.3) .

Results:

```
>> ssb
```

```
N1 =28
```

```
N = 401
```

```
Nb = 401
```

```
mag_X1 = 27.8678
```

```
idx_X1 =11
```

```
mag_Y1 =27.7080
```

idx_Y1 = 11

phase_lag_diff1 = -90.8679

The results obtained above, are from the MATLAB code, which gives us the magnitudes of the signal tracing the path of Hilbert filter and without it. Also, it provides us with the phase difference of the same.

Conclusion from MATLAB simulation:

From the dual +/- 45-degree filter, it can be observed that the phase difference is -90.8679 which is almost 90 degree, better than that of the conventional (91.0486) and the magnitude of two filters are 27.8678 and 27.7080, which are almost equal. Thus, relatively we can conclude of better specifications for SSB modulation through simulation for the dual filter method.

Graphs obtained:

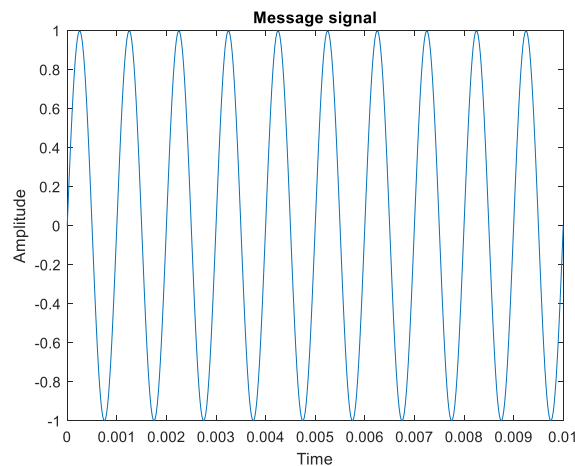


Figure 5.6 message signal of dual filter

This graph shows the continuous sinusoidal waveform of message signal having frequency of 1KHz and amplitude of 1. This would be the signal which would be modulated with the carrier (5KHz).

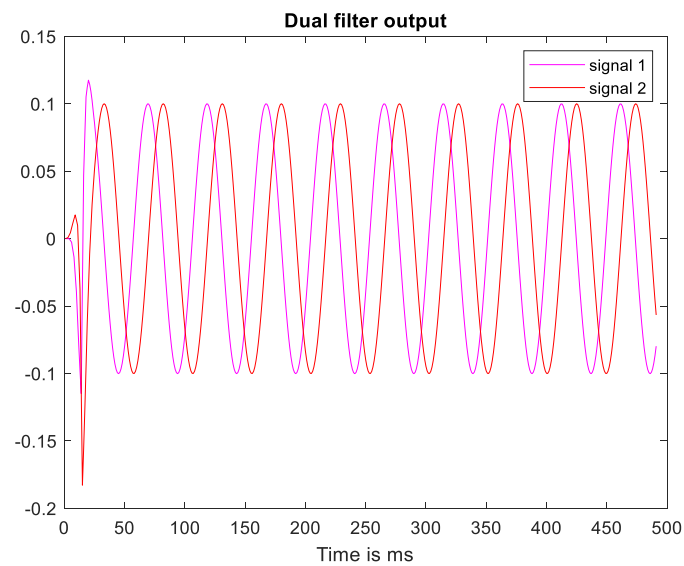


Figure 5.7 dual filter output

The above graph shows us output of the two filters; we can see the same range of oscillations for both the filters. Also, we can observe that the amplitude for the two filters are same and there is almost a 90-degree phase shift.

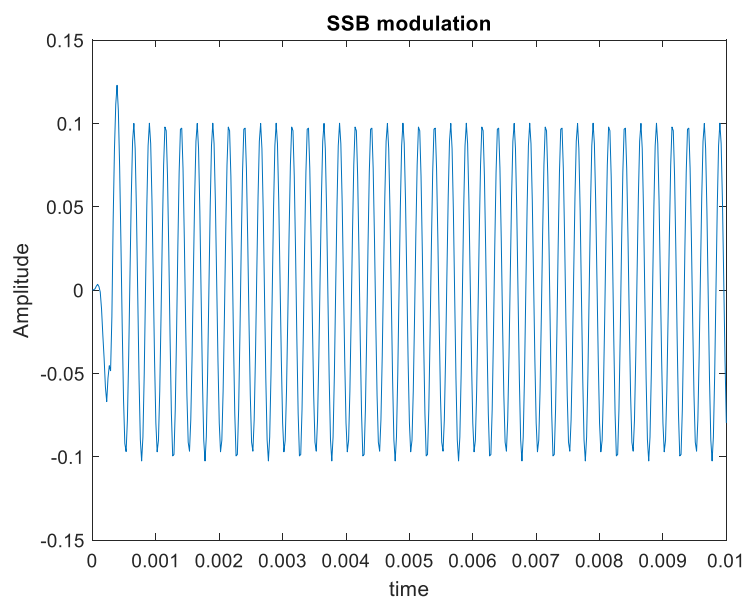


Figure 5.8 SSB modulation

The above figure shows the single sideband modulation as a continuous waveform in time domain where one of the sidebands has been suppressed. Due to some prevalence of the suppressed sideband, there is a distorted wave. Upon observing and comparing it with the conventional method we get a relatively better sinusoid.

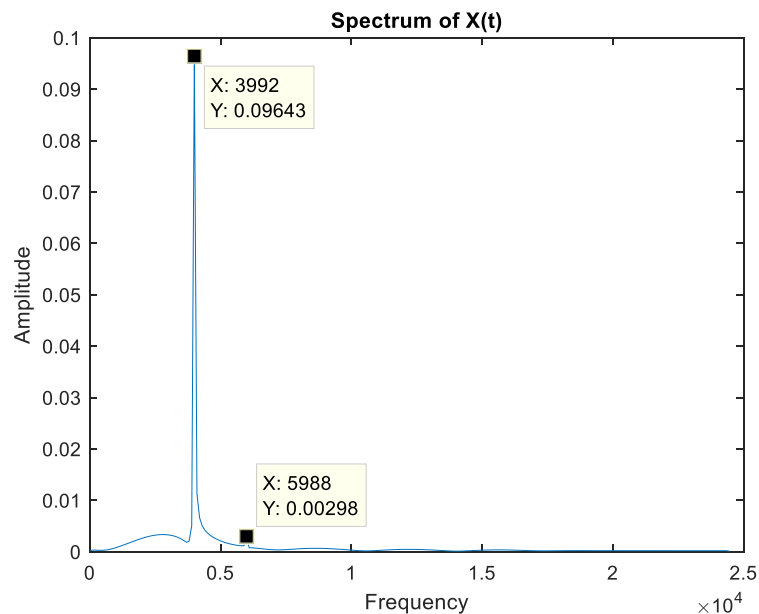


Figure 5.9 spectrum of X(t)

As one of the sidebands has been suppressed, the above graph shows the lower side band of SSB (fc-fm) . So, for a "lower sideband" mode of SSB transmission, the transmitted signal would have the spectrum shown in above. In this case, the (lower) frequency edge of the baseband signal has been translated in frequency so that it's located at 4kHz.

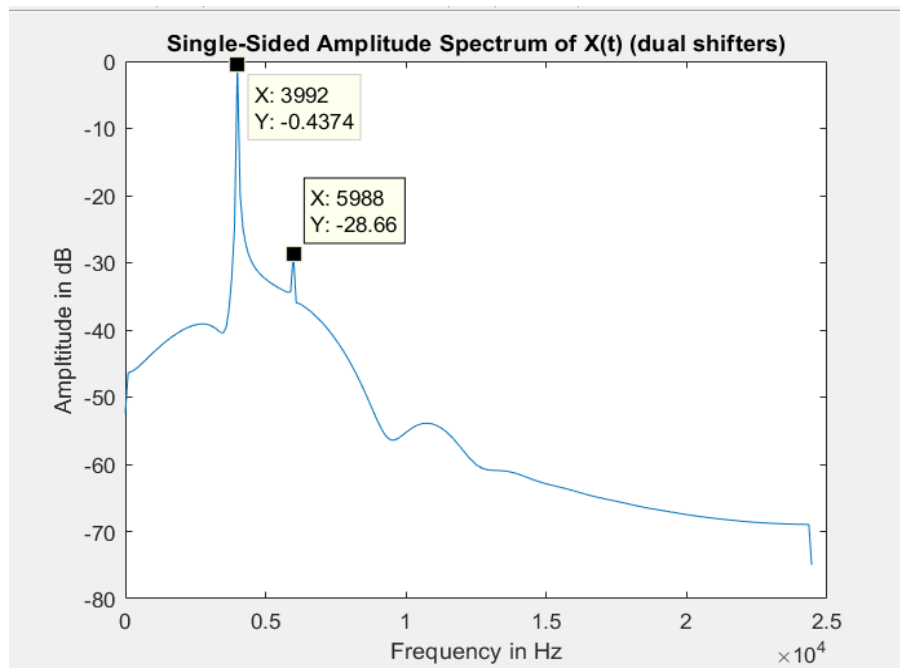


Figure 5.10 single sided amplitude spectrum of X(t)

The above graph shows the separation between two side band. The separation is 27dB between lower and upper side band. As we can observe that one of the bands is suppressed and the separation between the two bands is 27 dB. This improves the signal to noise ratio which reduces noise and interference. In SSB spectrum both the side bands are mirror images of each other, hence carry same information. This leaves only one sideband. The prevalent bands can be observed in the graph, which is the spectrum of the SSB signal.

Basically, spectrum allows us to observe the signal in wide ranges of frequencies. In this spectrum carrier and a sideband are suppressed and a single sideband is allowed for transmission. Also, in this spectrum, transmission of a greater number of signals are allowed. Signal fading is less likely to occur.

- Thus, it can be concluded that we get a better phase accuracy in the proposed methodology that -90.8679 degrees vs 91.0486 degrees.
- It can also be concluded that we get a better USB suppression in proposed methodology that is 27dB vs 19 dB

6.VHDL IMPLEMENTATION of hardware simulation

The implementation of the hardware-based simulation is done using VHDL. For this simulation Xilinx's **Vivado** design suite is chosen. **Vivado**, design Suite is a software suite produced by Xilinx for synthesis and analysis of HDL designs. The edition used to implement is the free licence, *Vivado WebPack* edition 2016. **Vivado**, is basically an EDA (Electronic design automation) tool. The Electronic design automation (EDA), also referred to as electronic computer-aided design (ECAD), is a category of software tools for designing electronic systems such as integrated circuits and printed circuit boards. The tools work together in a design flow that chip designers use to design and analyse entire semiconductor chips. Since a modern semiconductor chip can have billions of components, EDA tools are essential for their design.

For implementing the architecture-based hardware simulation, the kintex – 7 series-based libraries are used. Kintex – 7 is basically a FPGA family designed by Xilinx. All, the related libraries and IP's used are provided in the *Xilinx's Repository*, provided in the **Vivado** design suite.

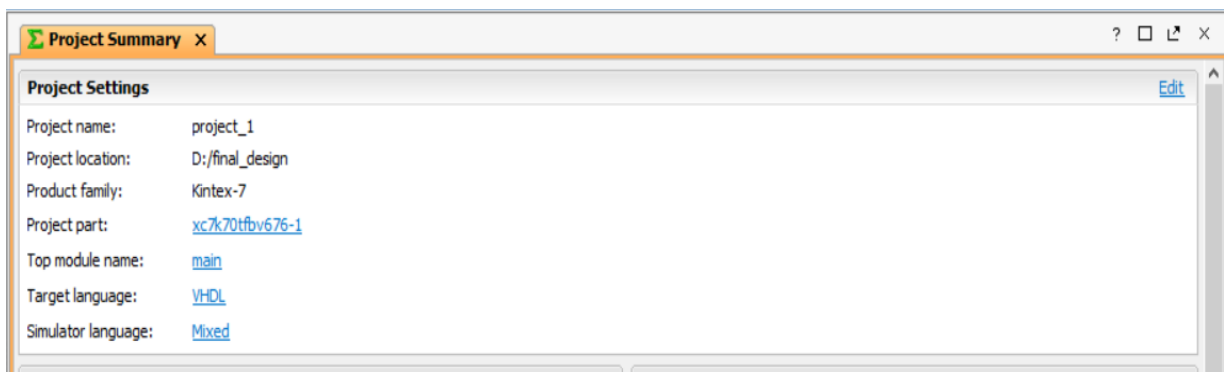


Figure 1.1 Project Summary

The figure above, shows the project summary, specifying the Kintex – 7 family of devices.

To implement the architecture in **Vivado**, IP cores have been used. The detailed summary of the same is provided below;

Before, describing the IP cores, the clocking scheme provided to the architecture is LVDS form of clocking. LVDS, stands for **Low Voltage Differential signalling**. The LVDS operates at low power and can run at high speeds. This basically helps in getting the timing and synchronization of the architecture right. So, for this we use 2 voltage levels this can be specified as *200 Mhz_p* and *200 Mhz_n*. This, essentially signifies the default clock of 200 Mhz, which would be down converted to the required frequency in the project. LVDS reduces the generation of electromagnetic noise. This noise reduction is due to the equal and opposite current flow in the two wires creating equal and opposite electromagnetic fields that tend to cancel each other.

The *single ended* clocks, on the other hand have only one polarity i.e positive. This is the simplest and most commonly used signalling technique. Single-ended signalling is less expensive to implement than differential, but it lacks the ability to reject noise caused by:

- differences in ground voltage level between transmitting and receiving circuits.
- induction picked up on the signal wire.

There is a reset signal made available too. Which is helpful in clearing out the registers while power on. It basically flushes out the junk values, so that erroneous data is not present.

Thus, taking these all into view the clocking scheme chosen in the project is *LVDS*.

i. First, is the **MMCM (Mixed mode Clock manager)**

The system clock default is 200 Mhz, from 200 Mhz it is required to generate 49 Khz. 49 Khz, being very small compared to 200 Mhz, we down convert this in 2 stages. The first step being converting the 200 Mhz to any intermediate frequency greater than 5Mhz. To do this we use the *clocking*

wizard ip, in which MMCM mode is used. 9.8 Mhz is chosen as the frequency to down convert as it satisfies, the condition of being greater than 5Mhz (to prevent any timing errors).

Next stage being using a counter to convert 9.8 Mhz to 49 Khz. The counter is set for 200 rounds, after every 200 rounds it toggles. $(9.8 \text{ Mhz} / 200) = 49\text{Khz}$, this logic is used to create the clock frequency of 49 Khz. 49Khz would be our clocking frequency, thus matching it with the sampling frequency.

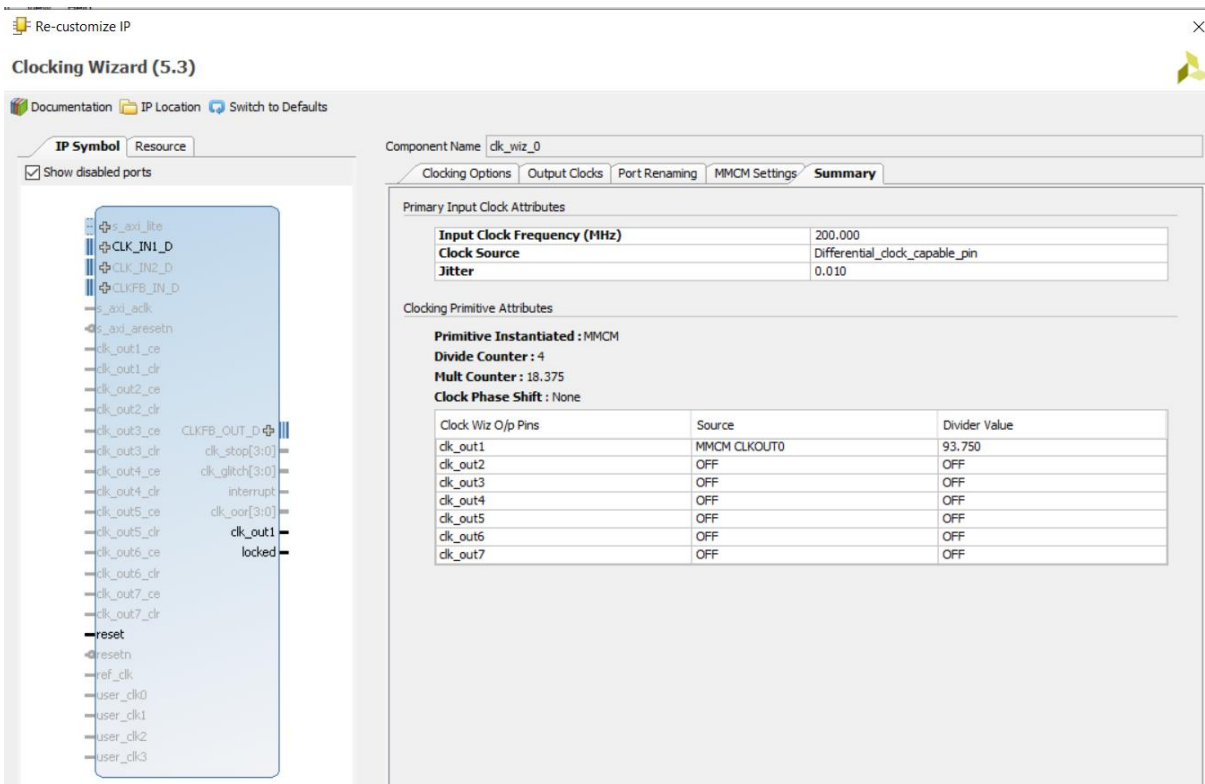


Figure 6.2 clock wiz ip

The above figure, shows the *clock wizard ip*.

Component Name: clk_wiz_0

Clkgen Options **Output Clocks** Port Renaming MMCM Settings Summary

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle Requested
		Requested	Actual	Requested	Actual	
<input checked="" type="checkbox"/> clk_out1	clk_out1	9.800	9.800	0.000	0.000	50.000
<input type="checkbox"/> clk_out2	clk_out2	100.000	N/A	0.000	N/A	50.000

Figure 6.3 clock wiz output of MMCM

The above figure, shows the selected 9.8 Mhz with 50% duty cycle as output.

- ii. The second IP used is the **DDS compiler** to generate sine and cosine for message signal.

Part of the architecture for generating SSB, we need a message signal. Message signal is that signal, which essentially is from end users. These are the signals, which are given as input from the user and received by another user. Here, since this is a demonstration of the architecture; an inbuilt source is chosen. The message signal is a small 1Khz signal, such small values are chosen for ease of simulation and demonstration. The architecture can very well be extended to human perceivable ranges and above.

The generation of this message is done by the DDS compiler ip of *Xilinx's repository*. This ip forms a part of DSP related ips.

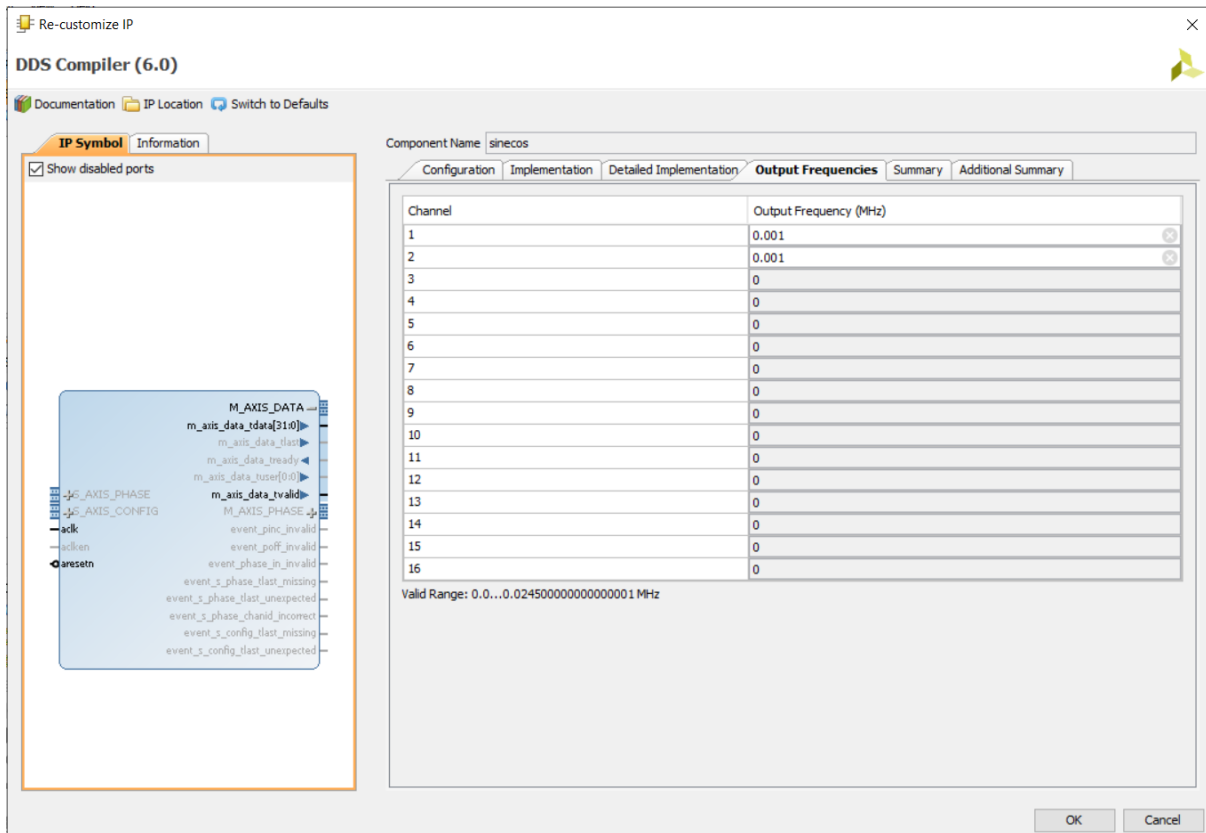


Figure 6.4 Message signal ip

The figure above, shows the IP. Here provision is given to use both cosine and sine as a message signal. However, the one chosen for this project is *sine*. The output is sliced into 16 bits, for each sine and cosine.

- iii. The third IP used is the **DDS compiler** to generate sine and cosine for carrier signal.

For any modulation scheme, carrier frequency forms an essential part. So, for this we use DDS compiler again to generate a carrier frequency of 5Khz. This frequency is again chosen such that, it is easy for representational and demonstration purpose.

The SSB modulation through Hartley architecture and the proposed methodology require the modulation by sine and cosine of the carrier. For this, again the output is sliced 16 bits for sine and cosine.

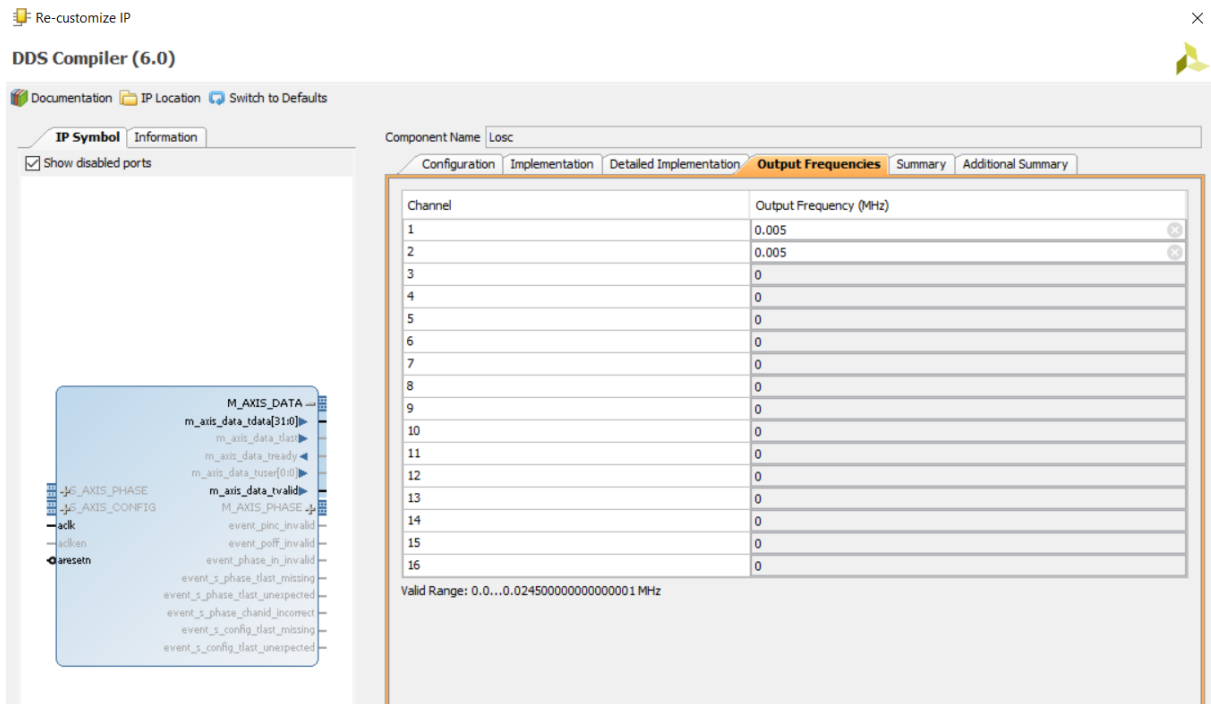


Figure 6.5 Carrier Signal ip

The above figure shows the 2 channels configured for 16 bit length. One channel generates the sine of carrier frequency, the other generates the cosine of carrier frequency.

iv. The fourth IP used is the **multiplier.**

The SSB by conventional Hartley and Hilbert method both have a multiplication element. The signals which are out of the Hilbert filters and message signal, and dual filters (+45/-45 degree), needs to be multiplied with the carrier, in order for modulation to occur.

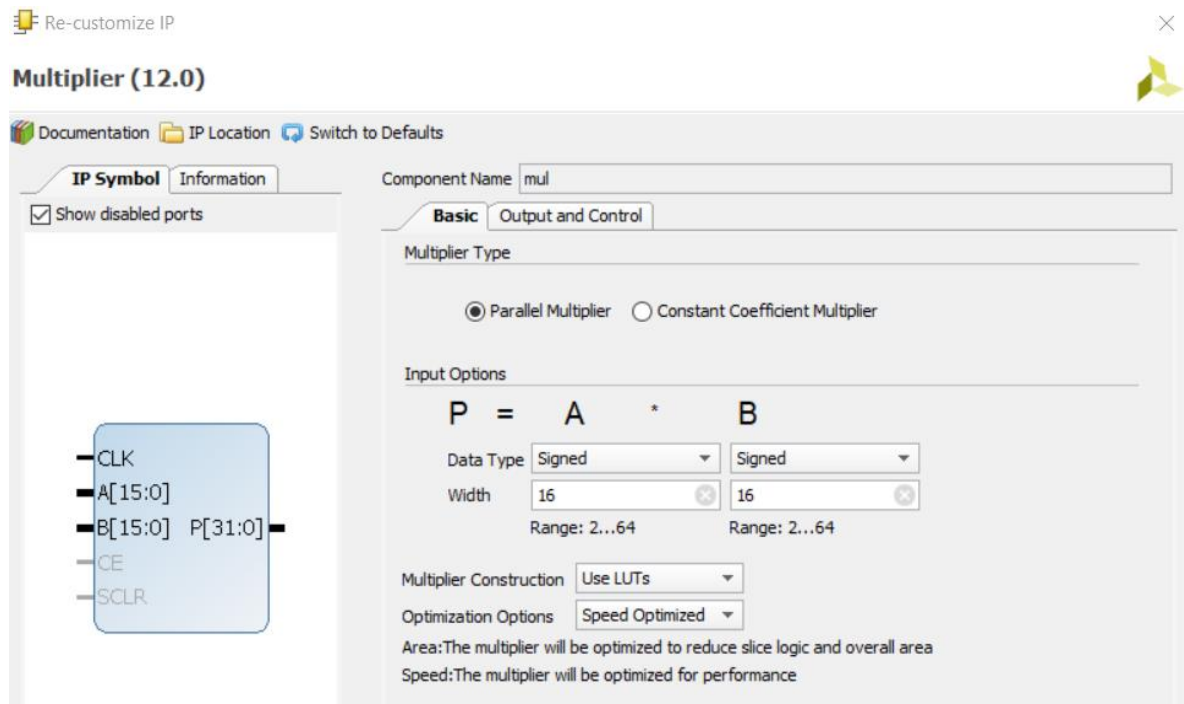


Figure 6.6 Multiplier ip

The figure above shows, the multiplier IP. There are 4 such multipliers used in this project. 2 for conventional and 2 for the proposed methodology. In here, through one of the ports carrier frequency of 5Khz is given and through one of the other ports message of 1 KHz passing through the Hilbert or only message or dual filter (+45/-45 degree) is given. It is to be noted that, both the ports multiply 16 bits of data, to give 32 bits of output.

- v. The fifth IP is, **FIR compiler** for 2 dual phase shifters (+45/-45 degree).

Filters here form the most important and significant part of the project. This is basically the implementation of the 2 dual filters for the proposed improvement over the Hartley architecture. The 45-degree dual filters are accepting the message signal, and shifting the signals by a phase of - 45 and +45 which is provided at the output. Note that it takes 16 bit message data and provides 16 bit output data for further process.

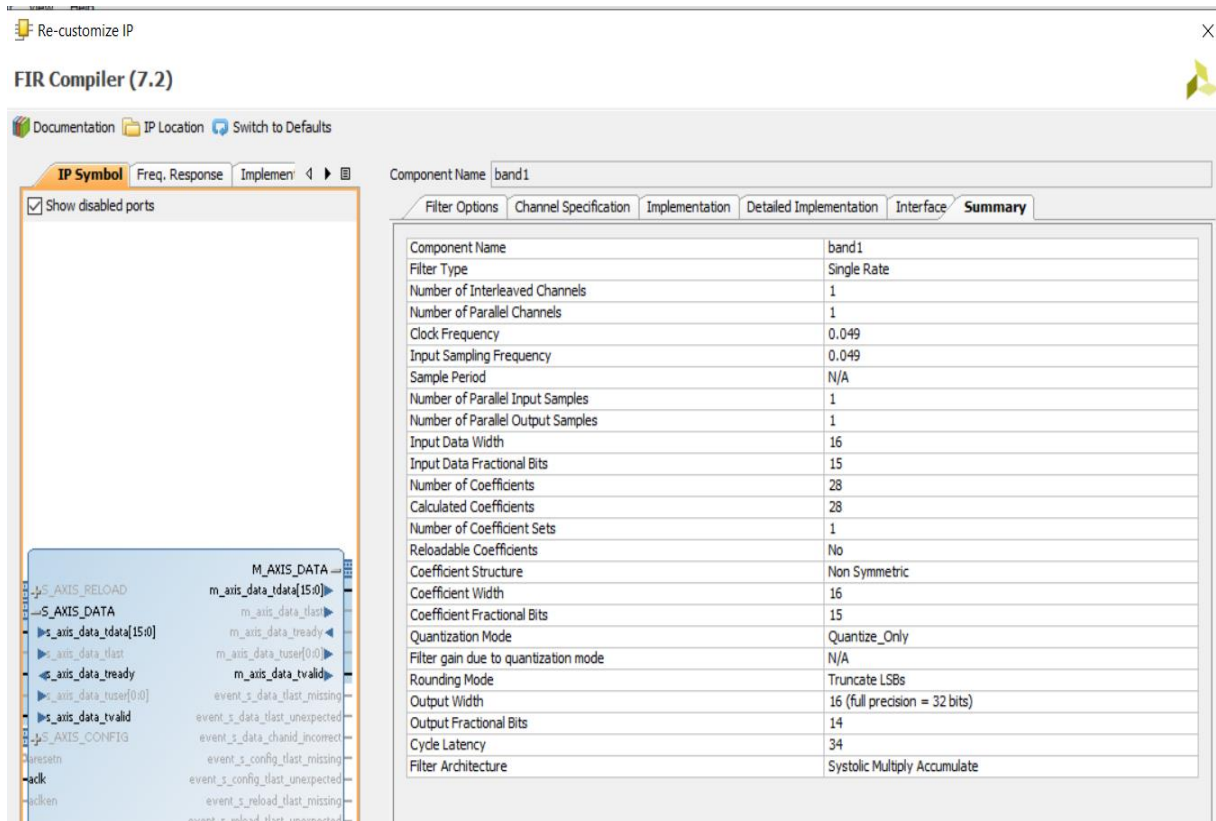


Figure 6.7 dual FIR filter IP

The figure above, shows the brief summary of the configuration. It is to be noted that 2 separate Ip's have been used to implement the 2 dual filters.

vi. The sixth IP, FIR compiler for Hilbert transform filter.

The Hilbert transform is a filter which essentially gives a phase shift 90 degree. This forms the essential part of Hartley architecture to generate the SSB. Here, the input given is the message signal of 16 bits, and an output of 16 bit is provided for further stages.

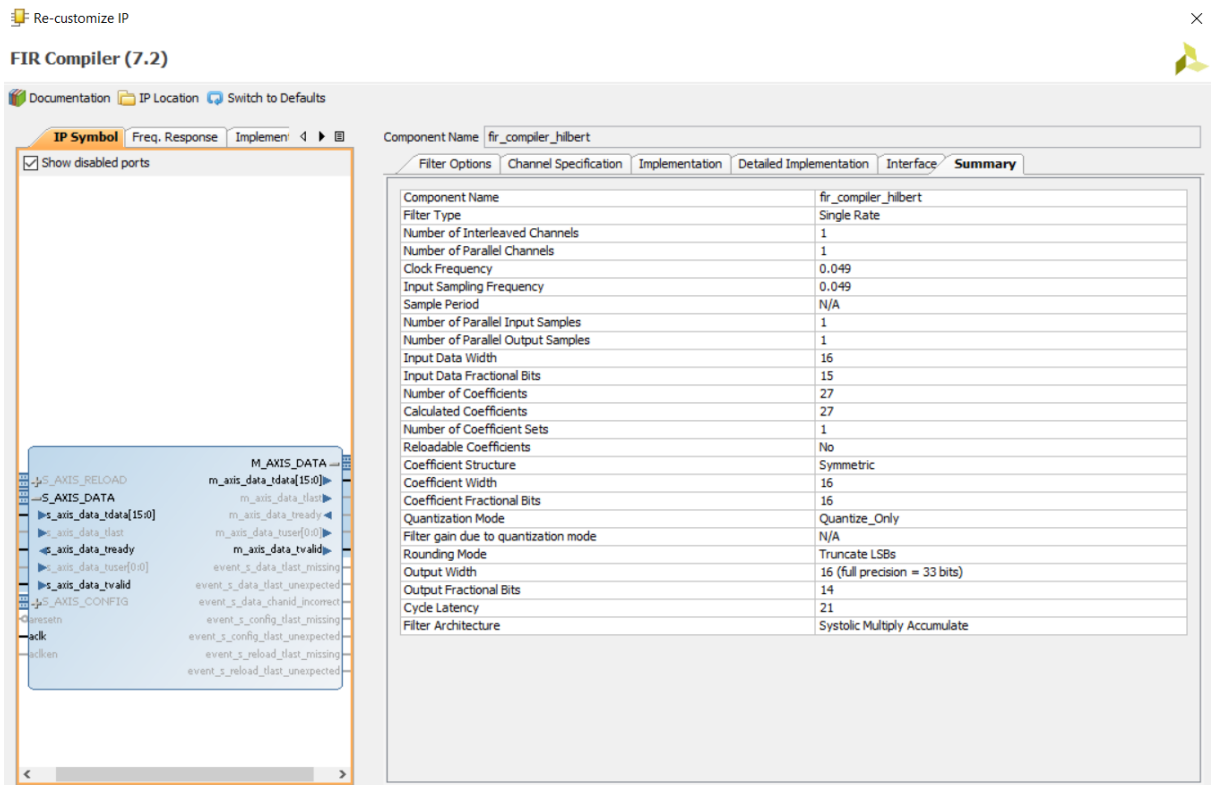


Figure 6.8 Hilbert FIR filter ip

The figure above shows the Hilbert filter implementation. It shows a brief summary of the configuration.

- vii. The seventh IP, **FIR compiler** for removing few harmonics at the output after generating the SSB for dual filter method. It is basically a low pass filter.

The final output produced, has many harmonics of the resultant signal. A few of the high frequency harmonics are attenuated to view SSB spectrum clearly.

The figure above shows, the spectrum of the LPF. It basically suppresses the higher frequency harmonics of the sidebands 4Khz and 6 KHz.

It is to be noted that, there are 2 such LPF used for both conventional and proposed method.

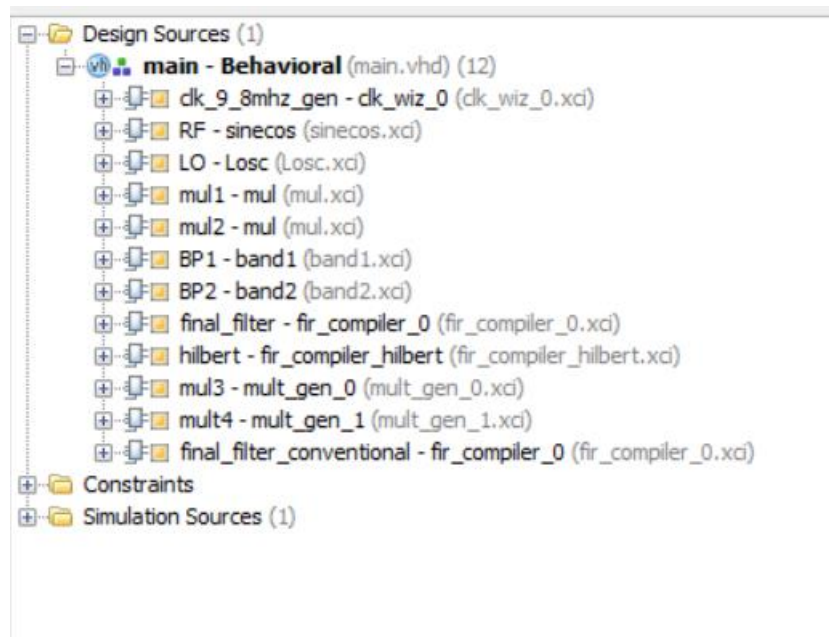


Figure 6.11 Final ip hierarchy

The above figure shows the IP order and the VHDL main file.

The IP integration and behavioural model is written using VHDL. The code can be referred in the **(APPENDIX 8.5)** .

After writing the VHDL code, the project is synthesized.

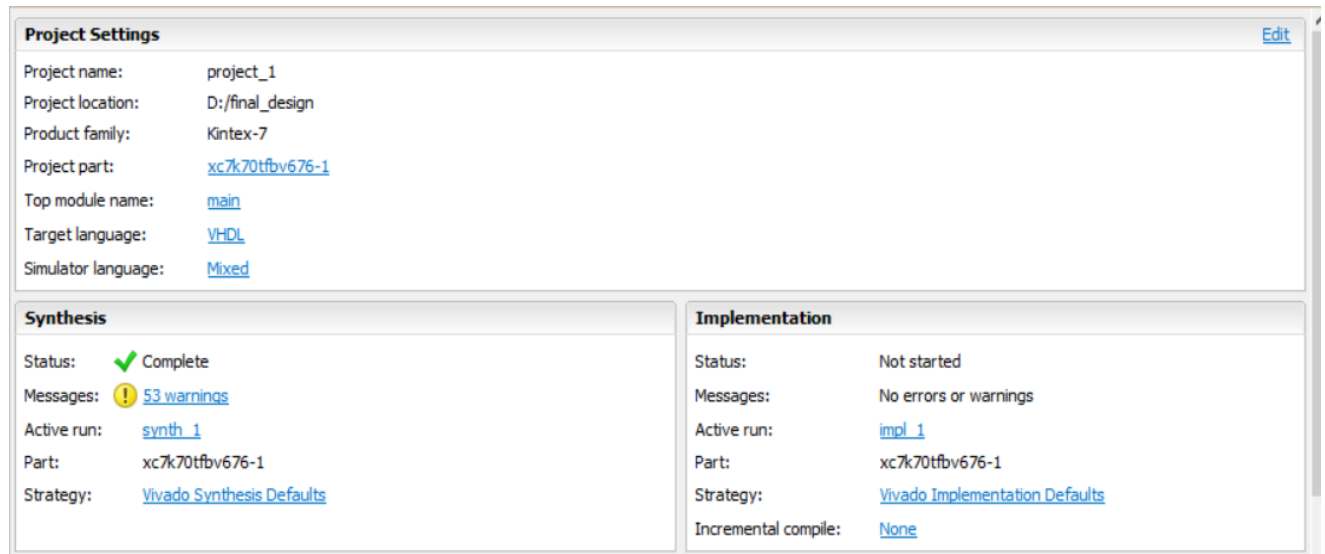


Figure 6.12 Synthesis completed

The above figure shows that, the synthesizer is successful.

The synthesized file is then elaborated to generate RTL (Register Transfer Level) schematics.

SIMULATION

First the waveform window is generated.

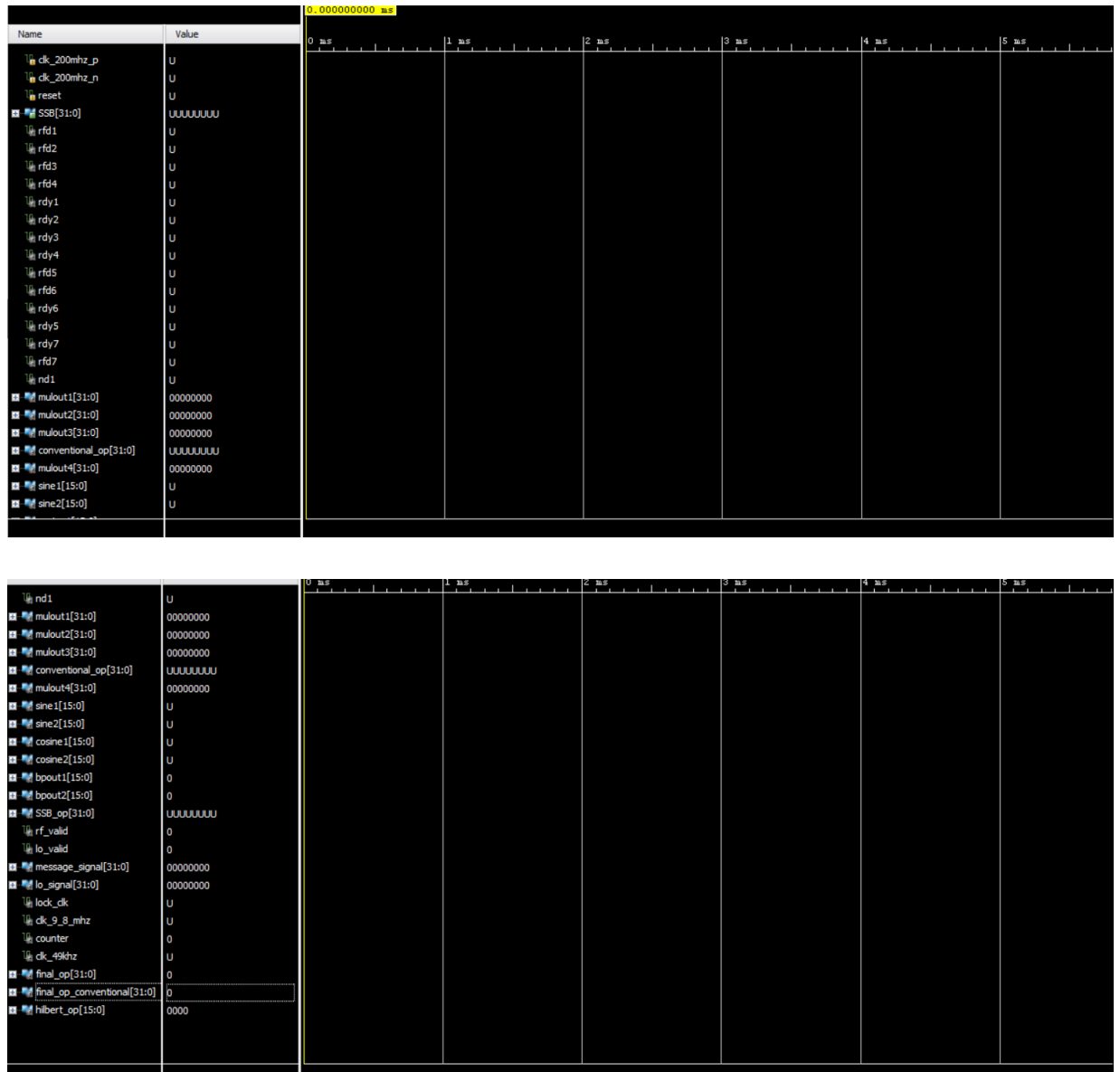


Figure 6.14 Wavewindow

The above figure, shows us the simulation waveform window.

First, we reset and clear out all the registers before running the simulation. The reset is initialized by force constant.

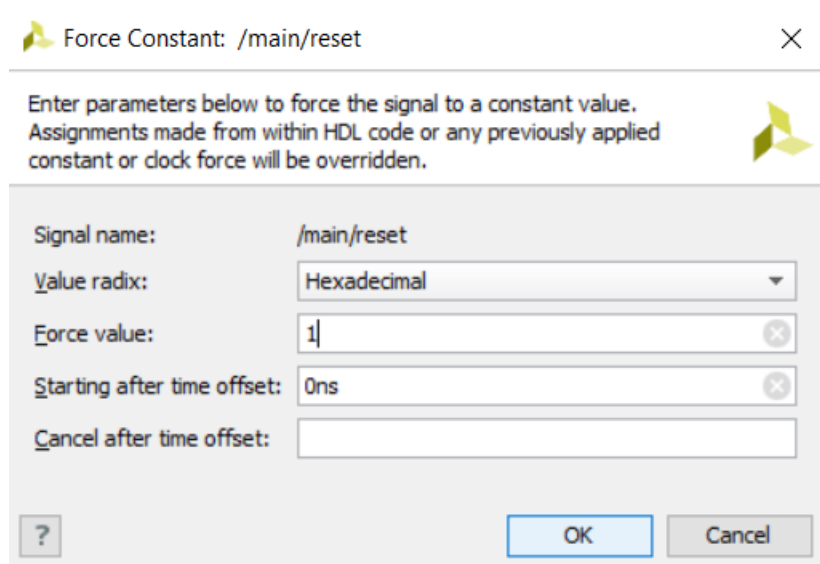
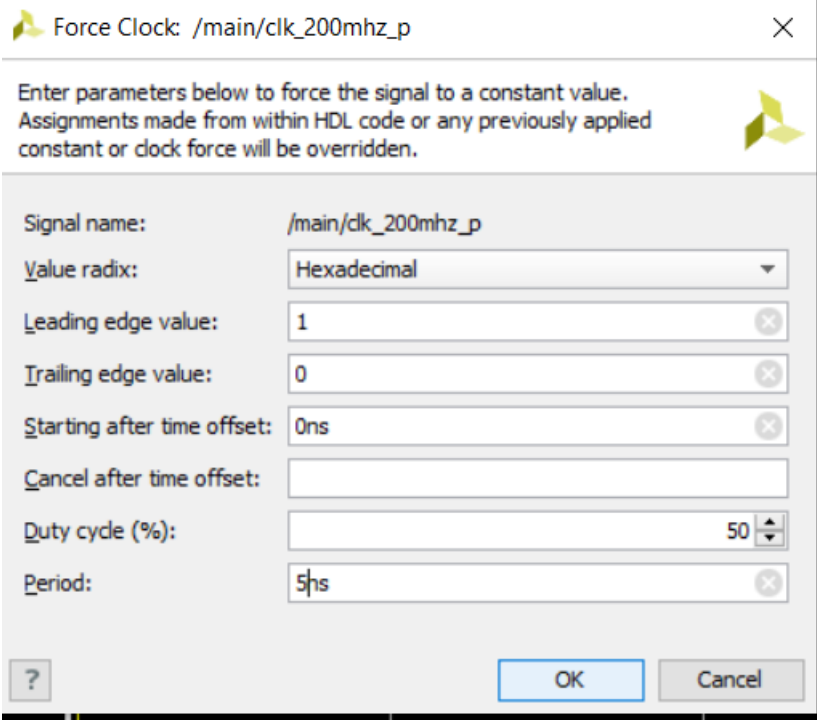


Figure 6.15 RESET

Also, after forcing the constant we would configure the system clock.

For this first we select the 200Mhz_p clock, and select force clock.



Force Clock: /main/clk_200mhz_p

Enter parameters below to force the signal to a constant value. Assignments made from within HDL code or any previously applied constant or clock force will be overridden.

Signal name: /main/clk_200mhz_p

Value radix: Hexadecimal

Leading edge value: 1

Trailing edge value: 0

Starting after time offset: 0ns

Cancel after time offset:

Duty cycle (%): 50

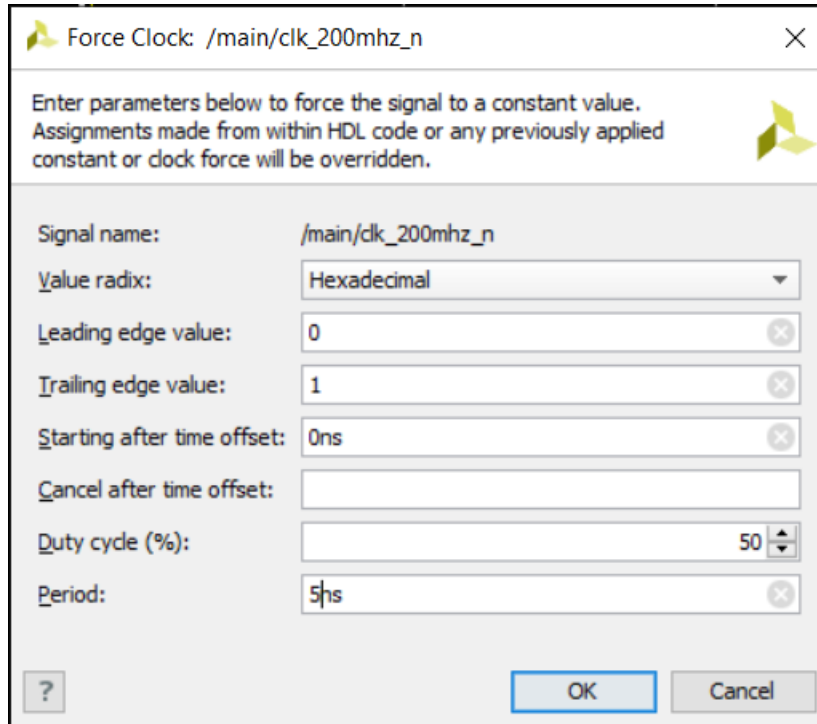
Period: 5ns

? OK Cancel

Figure 6.16 Clocking 200Mhz_P LVDS

The leading-edge value is 1 and trailing edge value is 0 selected. Then, keeping 50% duty cycle we choose the time period as 5ns. $T = 1 / f = 1 / 200\text{Mhz} = 5\text{ns}$.

Similarly, we select the 200Mhz_n clock, and select force clock.



Force Clock: /main/clk_200mhz_n

Enter parameters below to force the signal to a constant value.
Assignments made from within HDL code or any previously applied constant or clock force will be overridden.

Signal name: /main/clk_200mhz_n

Value radix: Hexadecimal

Leading edge value: 0

Trailing edge value: 1

Starting after time offset: 0ns

Cancel after time offset:

Duty cycle (%): 50

Period: 5ns

? OK Cancel

Figure 6.17 Clocking 200Mhz_n LVDS

The leading-edge value is 0 and trailing edge value is 1 selected. Then, keeping 50% duty cycle we choose the time period as 5ns. $T = 1 / f = 1 / 200\text{Mhz} = 5\text{ns}$.

The 200Mhz_p and 200Mhz_n is complimentary to each other.

After running the simulation for some time (200ns) to reset the registers, we then de select the force constant and initializing it to 0. This basically, allows for the simulation to run.

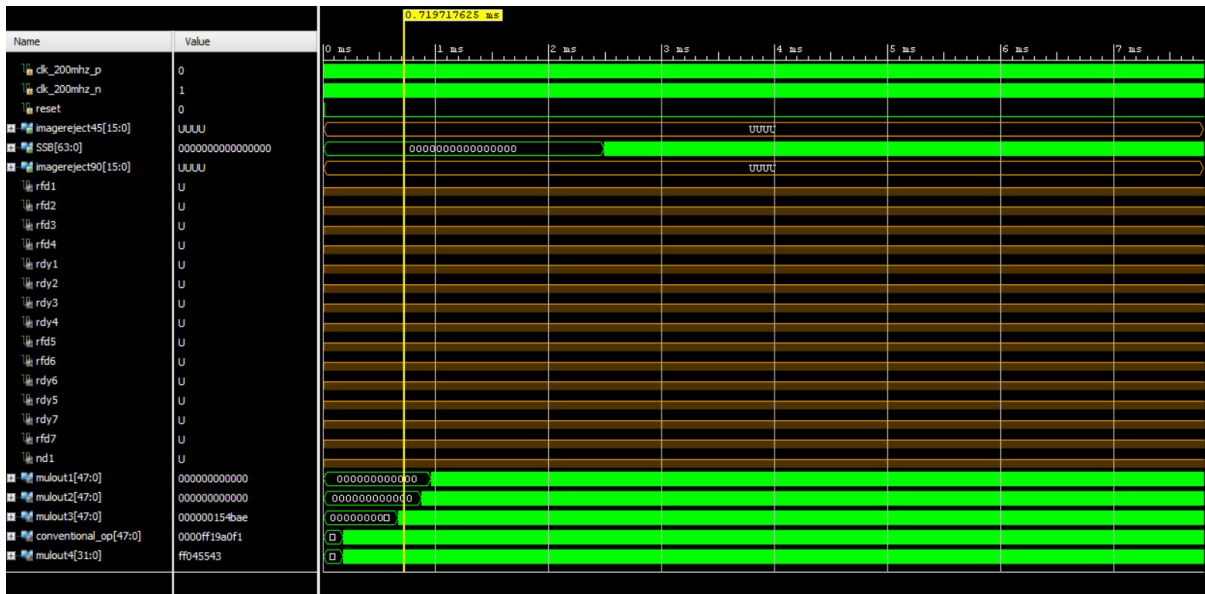


Figure 6.18 Waveform generated

The figure above shows the simulated waveform of the model. Now to view the output we select the end ports. The end ports respectively give us the output by *conventional Hartley architecture* and by the *proposed methodology*.

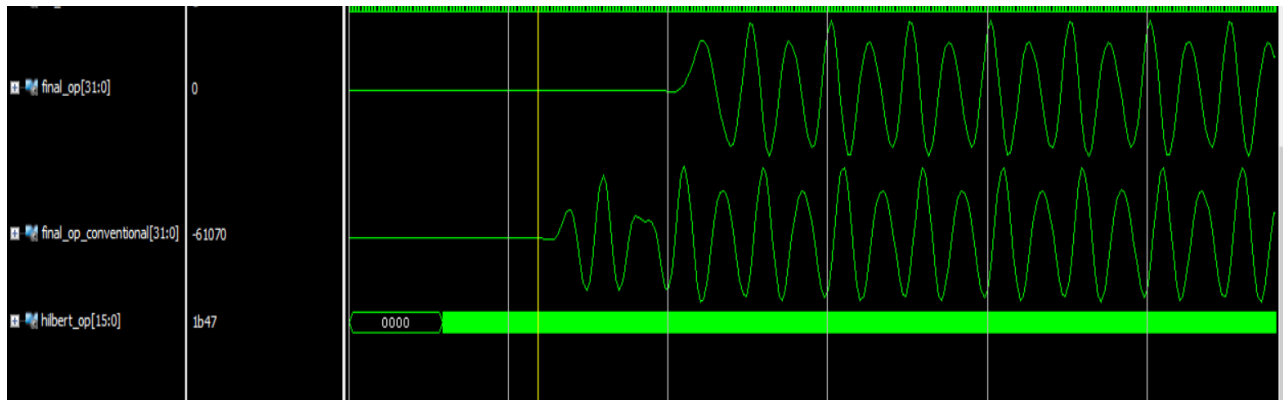


Figure 6.19 SSB modulated output

The figure above is the, output by the proposed methodology and Hartley architecture. The first sinusoid represents the output of the *dual phase shifter/proposed* method and the second sinusoid represents the output of the *conventional Hartley architecture*.

To view the wave in Analog form, from the wave window the radix is selected to be of *signed number type*. The wave type is then selected from *digital to Analog*.

MATLAB analysis of the generated text file.

The outputs are written in text file.

The data in the text files are used to analyse the spectrum of the SSB. To do this, the text files are imported in MATLAB, they are converted to matrix form and then their respective spectrum is found out. The code, which is used to run the text file in MATLAB is referenced in **APPENDIX 8.6**.

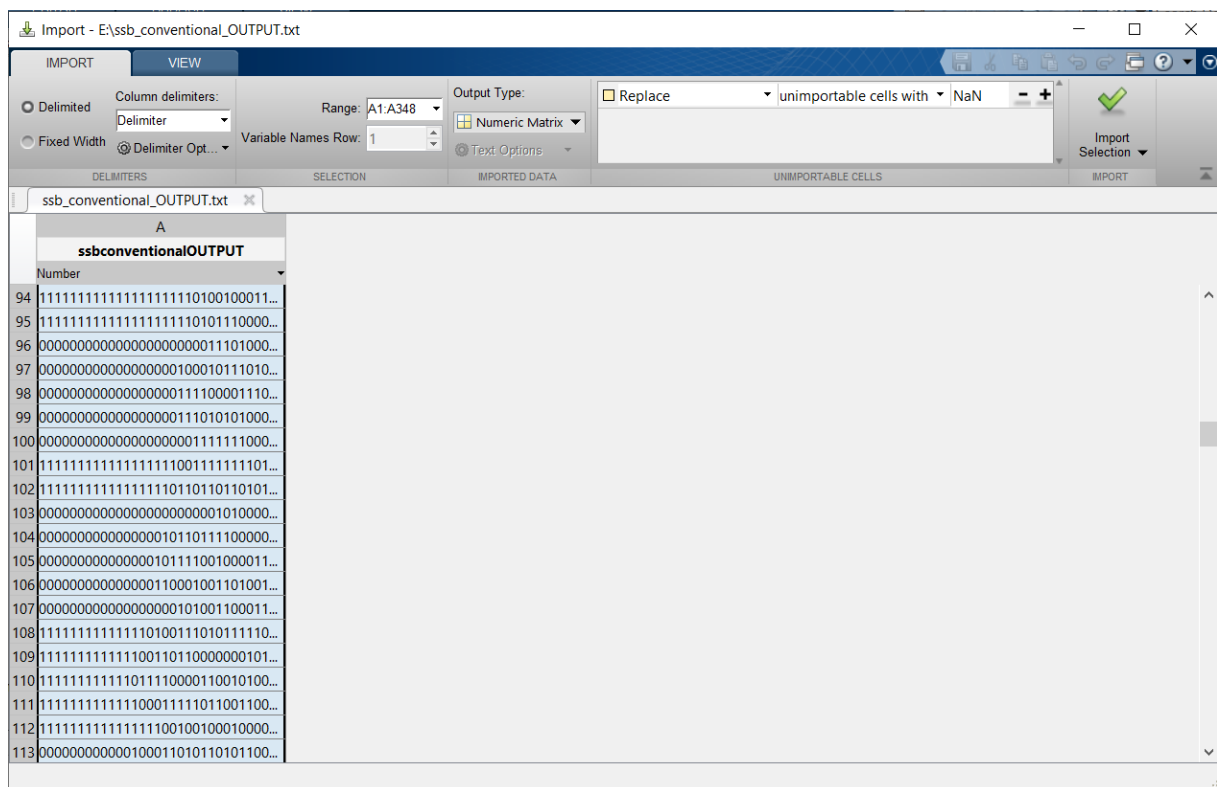


Figure 6.20 Text file import to MATLAB

The figure above, shows the imported text file to MATLAB for conventional method.

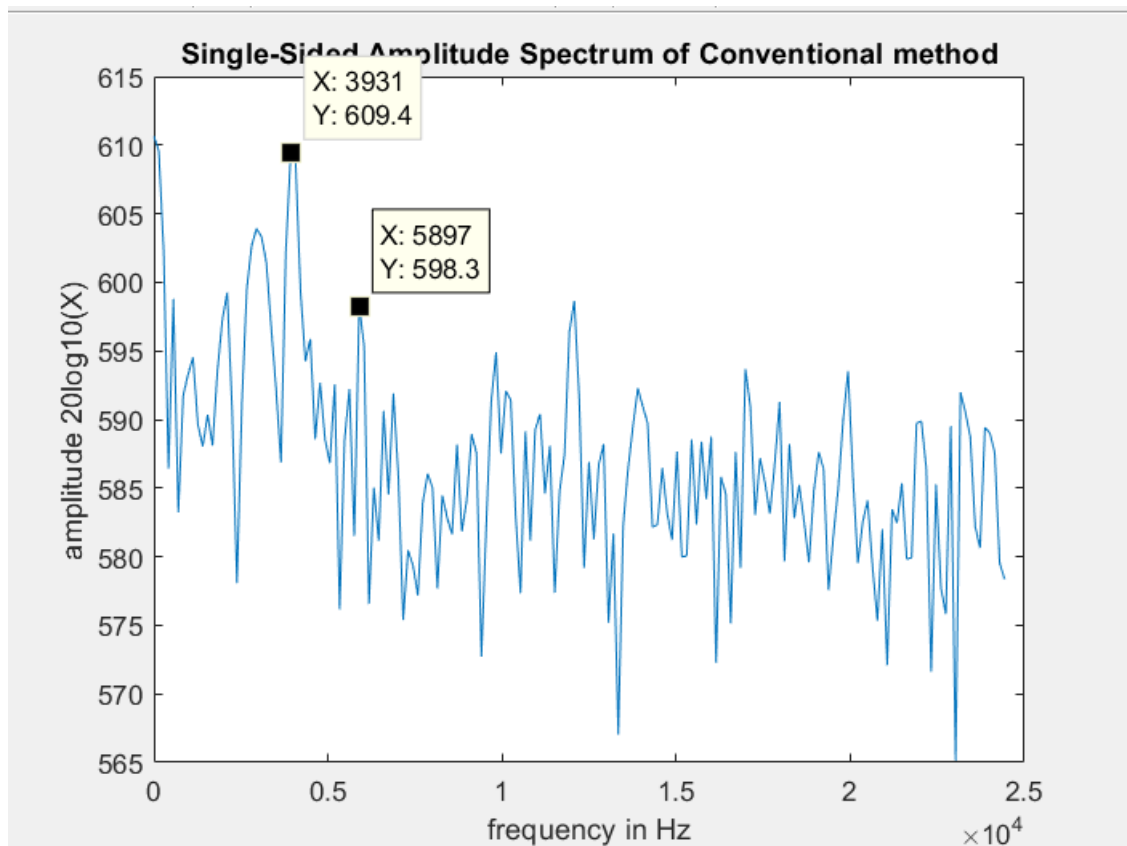


Figure 6.22 single sided spectrum of conventional method

The figure above shows the spectrum obtained, for the conventional SSB output. Here, for approximately 4Khz i.e is the LSB a magnitude of 609 dB has been observed. For, approximately 6Khz i.e is the USB a magnitude of 598 dB has been observed. This essentially means the LSB is prevalent and USB has been suppressed. The suppression of about 11dB, between the 2 can be concluded.

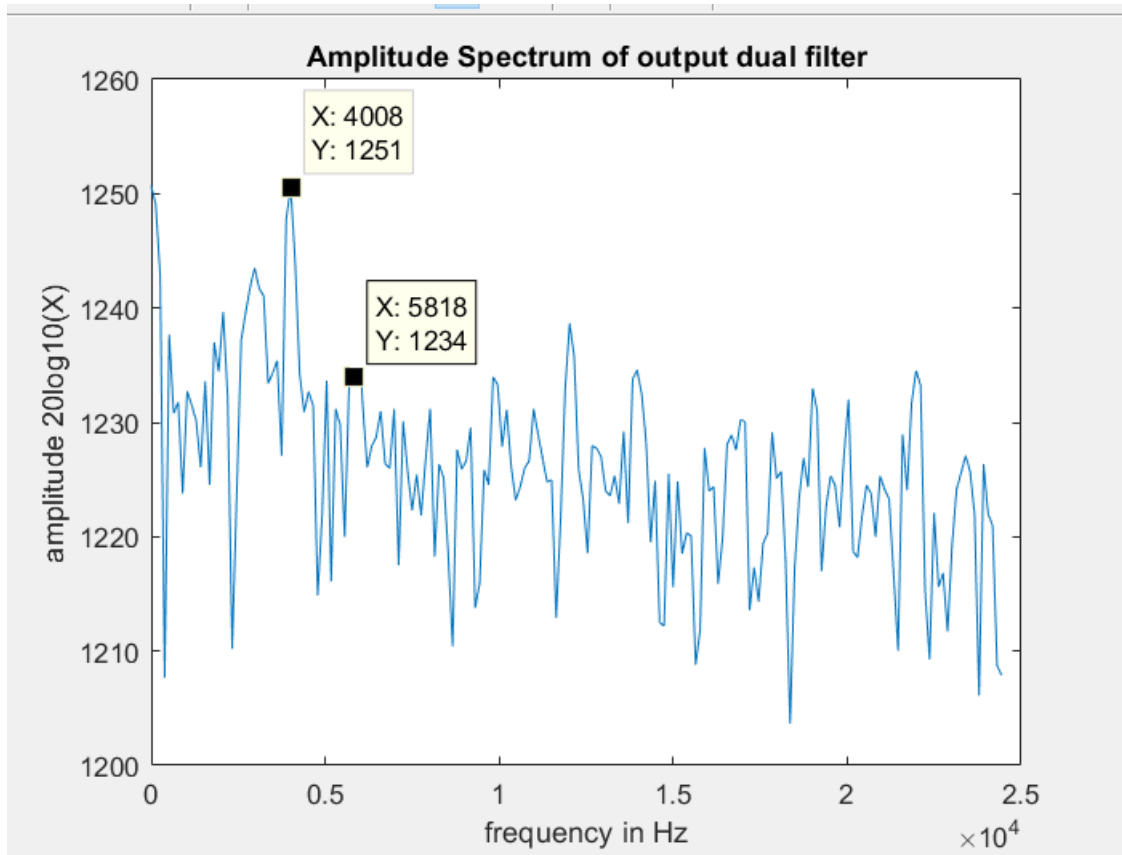


Figure 6.23 single sided spectrum for dual filter

The figure above shows the spectrum obtained, for the dual filter SSB output. Here, for approximately 4Khz i.e is the LSB a magnitude of 1251 dB has been observed. For, approximately 6Khz i.e is the USB a magnitude of 1234 dB has been observed. This essentially means the LSB is prevalent and USB has been suppressed. The suppression of about 17dB, between the 2 can be concluded.

Thus, from the simulation output, it can be concluded that the proposed methodology has given us a better suppression of the sidebands to generate SSB compared to conventional Hartley architecture. The suppression of 11dB vs 17dB. We can extend this, for different ranges of frequency and verify the suppression

7.OBJECTIVES ACHIEVED AND CONCLUSION

Here are the objectives which were achieved by this project.

7.1 Simulating the Hartley architecture and proposed methodology in MATLAB.

7.1.1 Conventional

- Designing the FIR Hilbert filter, generating the filter coefficients.
- Finding the phase difference between signal passing through Hilbert filter and without any filter. This has been found out to be 91.0486 degree.
- Simulating the defined Hartley architecture in MATLAB to generate time domain SSB.
- Obtaining the spectrum of the MATLAB simulated SSB, from which the magnitude suppression of the sidebands relative to each other has been found out. This has been observed to be around 19 dB.

7.1.2 Dual phase shifter.

- Designing the FIR +45/-45, generating the filter coefficients.
- Finding the phase difference between signal passing through +45 degree filter and -45 degree filter. This has been found out to be -90.8679 degree.
- Simulating the defined dual phase shifter architecture in MATLAB to generate time domain SSB.
- Obtaining the spectrum of the MATLAB simulated SSB, from which the magnitude suppression of the sidebands relative to each other has been found out. This has been observed to be around 27 dB.

7.2 Obtaining the filter characteristics.

- The filter characteristics of the conventional method has been obtained using MATLAB.
- The filter characteristics of the +45/-45 differential phase shifter filter has been obtained using MATLAB.

7.3 Implementing the architecture using VHDL

- Identifying the required IP's for constructing the architecture. (Clock Wizard, DDS Compiler Losc, DDS Compiler to sincos, FIR Filter DDS Compiler, Multiplier IP), configuring the same, to add to the project.
- VHDL code, to implement the architecture.
- Synthesizing the project in Vivado to get the elaborated RTL Schematic.
- Generating the waves in the waveform window, while doing behavioral simulation and observing SSB wave.
- Obtaining the text-file for analysis, both for conventional and proposed architecture.
- Importing the text-file in MATLAB, to obtain the spectrum.
- The VHDL implementation results in about 11 dB of suppression between the upper and lower sidebands in conventional method.
- The VHDL implementation results in about 17 dB of suppression between the upper and lower sidebands in dual-filter method.

From, the results obtained and analysing the obtained outputs by both conventional and proposed dual filter method it can be concluded that, the proposed methodology is an improvement over the conventional Hartley architecture. Comparing the results, from both MATLAB and VHDL implementation it has been observed that the suppression between the dominant and suppressed sidebands is consistently high in the proposed methodology. Also, when comparing the phase difference between the dual filters and Hilbert filter, it has been observed for same design parameters a slightly better phase separation has been observed in the proposed methodology (- 90.8679 degree vs 91.0486 degree).

Bibliography

- [1] C.S. Turner, "An efficient analytic signal generator", Signal Processing Magazine, vol. 23, no. 4, pp. 91-94, July 2009.
- [2] A. Reilly, G. Frazer and B. Boashash, "Analytic signal generation-tips and traps," in IEEE Transactions on Signal Processing, vol. 42, no. 11, pp. 3241-3245, Nov 1994
- [3] P. Duraiswamy, J. Bauwelinck, J. Vandewege, "Efficient implementation of 90° phase shifter in FPGA", EURASIP Journal on Advances in Signal Processing, vol. 2011, pp. 32-32, 2011
- [4] D. Shi and Y. J. Yu, Design of linear phase FIR filters with high probability of achieving minimum number of adders, IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 58, no. 1, pp. 126-136, Jan. 2011.
- [5] Y. J. Yu and Y. C. Lim, optimization of linear phase FIR filters in dynamically expanding sub-expressions space, Circuits, Syst., Signal Process., vol. 29. no. 1, pp. 65-80, 2010
- [6] Douglas Perry, VHDL: Programming by Example Hardcover – Import, 16 Jul 2002
- [7] Patel / Mitta, Programming in MATLAB ®: A problem-solving approach – 2014
- [8] B.P. Lathi , Modern Digital and Analog Communication Systems – 2011
- [9] <http://www.iowahills.com/Index.html> for FIR Hilbert and LPF filter design.

MATLAB CODE FOR FILTER:

8.1 Conventional Method

```
%this program is for 90

clear all;

close all;

fm = 1e3;%Message frq 1Khz

fs=49e3; % sampling frequency

t = 0:1/fs:0.01;

x = sin(2*pi*fm*t); % message signal

plot(t,x)

xlabel('time')

ylabel('amplitude')

title('message signal')

>Loading the hilbert coefficients.

A = [-0.003098750403898756 -0.020135571740449009 -
0.020939191991494636 -0.000202907720244840
0.026952793338152693 0.034346295140573725
0.007402212025804578 -0.038808195575967708 -
0.063568924468410643 -0.027845568148826612
0.073581669737100258 0.200516197928349321
0.288205417056360269 0.288205417056360269
0.200516197928349321 0.073581669737100258 -
0.027845568148826612 -0.063568924468410643 -
0.038808195575967708 0.007402212025804578
0.034346295140573725 0.026952793338152693 -
0.000202907720244840 -0.020939191991494636 -
0.020135571740449009 -0.003098750403898756];

N1 = length(A);
```

```

filterout_A = filter(A,1,x);%filter A
N=length(filterout_A);
B= x;
Nb=length(B);
figure (2)
plot(filterout_A,'m');
hold on;
plot(B,'r');
X1=fft(filterout_A);
%Y1=fft(Bs);
Y1=fft(B);
% Determine the max value and max point.
% This is where the sinusoidal
% is located.
[mag_X1 idx_X1] = max(abs(X1));
[mag_Y1 idx_Y1] = max(abs(Y1));
% determine the phase difference
% at the maximum point.
pX1 = angle(X1(idx_X1));
pY1 = angle(Y1(idx_Y1));
phase_lag1 = (pY1- pX1)*180/pi;
% determine the amplitude scaling
amplitude_ratio_diff1 = mag_Y1/mag_X1;
phase_lag_diff1=mod(phase_lag1+180,360)-180

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

fc = 5e3;%carrier frequency 5 KHz
t1 = 0:1/fs:0.01;
I=(cos(2*pi*fc*t1));
Q=(sin(2*pi*fc*t1));
As=(filterout_A).*Q;
Bs=B.*I;
SSB=As+B;

figure(3)
plot(t1,SSB)

Y=fft(SSB);
L=length(Y);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = fs*(0:(L/2))/L;
figure;
plot(f,P1)
figure;
plot(f,20*log10(P1)) % plot in db
title('Single-Sided Amplitude Spectrum of X(t)')

```

8.2 MATLAB CODE TO GENERATE DUAL FILTER COEFFICIENTS

```

function [A_of_k, B_of_k] = QuadFilterCoeffs_Turner(w1,
w2, a, N)

```

```

% [A_of_k, B_of_k] = QuadFilterCoeffs_Turner
%
% Computes the coefficients for the quadrature filters
% described by article: "An Efficient Analytic Signal
Generator"
% by Clay S. Turner.
%
%      w1: beginning frequency of filters' passband (-3dB
point).
%      "w1" must be in the range of 0 -to- 0.5.
%      w2: end frequency of filters' passband (-3dB point).
%      "w2" must be in the range of 0 -to- 0.5, and
greater than w1.
%      a: half the filters' desired transition-region
width.
%      "a" must be in the range of 0 -to- 0.5.
%      N: number of filter taps.
%      A_of_k: the computed in-phase filter's coefficients.
%      B_of_k: the computed quadrature-phase filter's
coefficients.
%
% Code by Richard (Rick) Lyons.  Sept. 2008.
% Use this code however you wish, but please do not
change
% the above comment lines.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Begin the function

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

% A little error checking

if w1<a || w1>0.5-a

    beep, disp(' '), disp(' ')

    error('w1 must be in the range: a <= w1 <= 0.5-a');

end

if w2<=w1

    beep, disp(' '), disp(' ')

    error('w2 must be greater than w1');

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

% Compute A_of_k (in-phase) and B_of_k (quad-phase)

%   coefficients from Eqs. (9) & (10).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

    k = 0:N-1;                                % Index of A_of_k
coefficients

    t = 2*pi*(k-(N-1)/2);                      % Time-domain variable
vector for Eq. (10)

    A_numer = 2*pi*pi*cos(a*t);                % Numerator factor
of A_of_k

    A_denom = t.*(4*a*a*t.*t-pi*pi);           % Denominator

    A_of_k = (sin(w1*t+pi/4) -
sin(w2*t+pi/4)).*(A_numer./A_denom);          % In-phase imp.
resp.

```

```

% Check for, and correct, indeterminate values in
"A_of_k" (eliminate the "NaNs")

    TempIndex = find(isnan(A_of_k)==1);

if length(TempIndex) > 0           % Are there any NaNs
in A_of_k?

    Angle_1 = (pi/4)*((a+2*w2)/a); % There are
Nan's in A_of_k

    Angle_2 = (pi/4)*((a+2*w1)/a);
    Angle_3 = (pi/4)*((a-2*w1)/a);
    Angle_4 = (pi/4)*((a-2*w2)/a);

for K = 1:length(TempIndex) % Correct A_of_k for each NaN
if t(TempIndex(K)) < -pi/(2*a)+0.1 % Is NaN at t=-
pi/(2a)?

    A_of_k(TempIndex(K)) =
a*(sin(Angle_3)-sin(Angle_4));
else, end

if t(TempIndex(K)) > pi/(2*a)-0.1 % Is NaN at t=pi/(2a)?

    A_of_k(TempIndex(K)) =
a*(sin(Angle_1)-sin(Angle_2));
else, end

if abs(t(TempIndex(K))) < 0.0001 % Is NaN at t=0?

    A_of_k(TempIndex(K)) = sqrt(2)*(w2-
w1);
else, end

end

elseend

    A_ofK = A_of_k

```

```
B_of_k = flipplr(A_of_k) % Flip "A_of_k", left to
right
```

8.3 MATLAB CODE FOR DUAL QUADRATURE FILTER :

```
%this program calculates the delay between + and - 45
degree phase shift
```

```
clear all;
```

```
close all;
```

```
fm = 1e3;%Message frq 1Khz
```

```
fs=49e3; % sampling frequency
```

```
t = 0:1/fs:0.01;
```

```
%t = 0:1/fs:0.001;
```

```
x = sin(2*pi*fm*t); % message signal
```

```
plot(t,x)
```

```
A=[0.0000    -0.0000    -0.0000    -0.0103    -0.0000    -
0.0336     0    -0.0613     0.0000    -0.0685 -0.0000    -
0.0000     0.0000     0.7466           0    -0.3952           0
-0.1540     0.0000    -0.0507 0.0000     0.0000     0.0000
0.0149     0.0000     0.0109           0     0.0032];
```

```
N1 = length(A)
```

```
filterout_A = filter(A,1,x);%filter A
```

```
N=length(filterout_A)
```

```
B= flipplr(A);
```

```
filterout_B = filter(B,1,x);%filter B
```

```
Nb=length(filterout_B)
```

```
figure (2)
```

```
plot(filterout_A, 'm');
```



```

hold on;

plot( filterout_B,'r');

X1=fft(filterout_A);

%Y=fft(c1);

Y1=fft(filterout_B);

% Determine the max value and max point.

% This is where the sinusoidal

% is located.

[mag_X1 idx_X1] = max(abs(X1));

[mag_Y1 idx_Y1] = max(abs(Y1));

% determine the phase difference

% at the maximum point.

pX1 = angle(X1(idx_X1));

pY1 = angle(Y1(idx_Y1));

phase_lag1 = (pY1- pX1)*180/pi;

% determine the amplitude scaling

amplitude_ratio_diff1 = mag_Y1/mag_X1;

phase_lag_diff1=mod(phase_lag1+180,360)-180

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fc = 5e3;%carrier frequency 5KHz

t1 = 0:1/fs:.01;

I=(cos(2*pi*fc*t1));

Q=(sin(2*pi*fc*t1));

As=(filterout_A).*I;

Bs=(filterout_B).*Q;

```

```

SSB=As+Bs;

figure(3)

plot(t,SSB)

Y=fft(SSB);

L=length(Y);

P2 = abs(Y/L);

P1 = P2(1:L/2+1);

P1(2:end-1) = 2*P1(2:end-1);

f = fs*(0:(L/2))/L;

figure;

plot(f,P1)

figure;

plot(f,20*log10(P1)) % plot in db

title('Single-Sided Amplitude Spectrum of X(t)')

```

8.4 MATLAB CODE FOR FINDING FILTER CHARACTERISTICS:

```

%Loading the dual filter coefficients

A=[0.0000    -0.0000    -0.0000    -0.0103    -0.0000    -
0.0336     0    -0.0613     0.0000    -0.0685 -0.0000    -
0.0000     0.0000     0.7466           0    -0.3952           0
-0.1540     0.0000    -0.0507 0.0000     0.0000     0.0000
0.0149     0.0000     0.0109           0     0.0032];

B=fliplr(A);

%Loading the Hilbert coefficients

```

```

C = [-0.003098750403898756 -0.020135571740449009 -
0.020939191991494636 -0.000202907720244840
0.026952793338152693 0.034346295140573725
0.007402212025804578 -0.038808195575967708 -
0.063568924468410643 -0.027845568148826612
0.073581669737100258 0.200516197928349321
0.288205417056360269 0.288205417056360269
0.200516197928349321 0.073581669737100258 -
0.027845568148826612 -0.063568924468410643 -
0.038808195575967708 0.007402212025804578
0.034346295140573725 0.026952793338152693 -
0.000202907720244840 -0.020939191991494636 -
0.020135571740449009 -0.003098750403898756];

figure(1)

freqz(A,1)

title('FIR filter characteristics of +45 degree filter')

figure(2)

freqz(B,1)

title('FIR filter characteristics of -45 degree filter')

figure(3)

freqz(C,1)

title('FIR filter characteristics of hilbert filter
(Raised Cosine)')

```

8.5 VHDL CODE:

```

-----

-- Company:

-- Engineer:

--

-- Create Date: 13:46:06 12/03/2020

-- Design Name:

-- Module Name: main - Behavioral

```

```

-- Project Name:

-- Target Devices:

-- Tool versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--
-----

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

use STD.textio.all;

use IEEE.STD_LOGIC_TEXTIO.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 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 main is
port(
--clk: in std_logic;

clk_200mhz_p: in std_logic;

clk_200mhz_n: in std_logic;

reset: in std_logic;

SSB:out std_logic_vector( 31 downto 0)

);
end main;

```

architecture Behavioral of main is

COMPONENT sinecos

```

PORT (

    aclk : IN STD_LOGIC;

    aresetn : IN STD_LOGIC;

    m_axis_data_tvalid : OUT STD_LOGIC;

    m_axis_data_tdata : OUT STD_LOGIC_VECTOR(31 DOWNT0 0)

);

```

END COMPONENT;

COMPONENT Losc

```

PORT (

    aclk : IN STD_LOGIC;

    aresetn : IN STD_LOGIC;

    m_axis_data_tvalid : OUT STD_LOGIC;

```

```

    m_axis_data_tdata : OUT STD_LOGIC_VECTOR(31 DOWNT0 0)
);
END COMPONENT;

```

```

COMPONENT mul

PORT (

    CLK : IN STD_LOGIC;

    A : IN STD_LOGIC_VECTOR(15 DOWNT0 0);

    B : IN STD_LOGIC_VECTOR(15 DOWNT0 0);

    P : OUT STD_LOGIC_VECTOR(31 DOWNT0 0)

);
END COMPONENT;

```

```

COMPONENT band1

PORT (

    aclk : IN STD_LOGIC;

    s_axis_data_tvalid : IN STD_LOGIC;

    s_axis_data_tready : OUT STD_LOGIC;

    s_axis_data_tdata : IN STD_LOGIC_VECTOR(15 DOWNT0 0);

    m_axis_data_tvalid : OUT STD_LOGIC;

    m_axis_data_tdata : OUT STD_LOGIC_VECTOR(15 DOWNT0 0)

);
END COMPONENT;

```

```

COMPONENT band2

PORT (

```

```

    aclk : IN STD_LOGIC;

    s_axis_data_tvalid : IN STD_LOGIC;

    s_axis_data_tready : OUT STD_LOGIC;

    s_axis_data_tdata : IN STD_LOGIC_VECTOR(15 DOWNT0 0);

    m_axis_data_tvalid : OUT STD_LOGIC;

    m_axis_data_tdata : OUT STD_LOGIC_VECTOR(15 DOWNT0 0)

);

END COMPONENT;

```

```

COMPONENT fir_compiler_hilbert

PORT (

    aclk : IN STD_LOGIC;

    s_axis_data_tvalid : IN STD_LOGIC;

    s_axis_data_tready : OUT STD_LOGIC;

    s_axis_data_tdata : IN STD_LOGIC_VECTOR(15 DOWNT0 0);

    m_axis_data_tvalid : OUT STD_LOGIC;

    m_axis_data_tdata : OUT STD_LOGIC_VECTOR(15 DOWNT0 0)

);

END COMPONENT;

```

```

--COMPONENT delay_element

--PORT (

    --D : IN STD_LOGIC_VECTOR(15 DOWNT0 0);

    --CLK : IN STD_LOGIC;

    --Q : OUT STD_LOGIC_VECTOR(15 DOWNT0 0)

```

```

--);

--END COMPONENT;

COMPONENT mult_gen_0

PORT (

    CLK : IN STD_LOGIC;

    A : IN STD_LOGIC_VECTOR(15 DOWNT0 0);

    B : IN STD_LOGIC_VECTOR(15 DOWNT0 0);

    P : OUT STD_LOGIC_VECTOR(31 DOWNT0 0)

);

END COMPONENT;

COMPONENT mult_gen_1

PORT (

    CLK : IN STD_LOGIC;

    A : IN STD_LOGIC_VECTOR(15 DOWNT0 0);

    B : IN STD_LOGIC_VECTOR(15 DOWNT0 0);

    P : OUT STD_LOGIC_VECTOR(31 DOWNT0 0)

);

END COMPONENT;

-----clk_9.8mhz_gen-----

component clk_wiz_0

port

(-- Clock in ports

clk_in1_p      : in  std_logic;

clk_in1_n      : in  std_logic;

-- Clock out ports

clk_out1       : out  std_logic;

-- Status and control signals

```



```

reset      : in  std_logic;

locked     : out  std_logic

);

end component;

COMPONENT fir_compiler_0

PORT (

    aclk : IN STD_LOGIC;

    s_axis_data_tvalid : IN STD_LOGIC;

    s_axis_data_tready : OUT STD_LOGIC;

    s_axis_data_tdata : IN STD_LOGIC_VECTOR(31 DOWNT0 0);

    m_axis_data_tvalid : OUT STD_LOGIC;

    m_axis_data_tdata : OUT STD_LOGIC_VECTOR(31 DOWNT0 0)

);

END COMPONENT;

signal rfd1,rfd2,rfd3,rfd4,rdy1,rdy2,rdy3,rdy4,rfd5,rfd6,rdy6,rdy5,rdy7,rfd7,nd1:std_logic;

--signal mulout1,mulout2:std_logic_vector(15 downto 0);

signal mulout1,mulout2,mulout3:std_logic_vector(31 downto 0);

signal conventional_op:std_logic_vector(31 downto 0);

signal mulout4:std_logic_vector(31 downto 0);

--signal mu1out3,mulout4:integer;

--signal lpout1,lpout2,hilout,d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13: std_logic_vector(15
downto 0);

signal sine1,sine2,cosine1,cosine2:std_logic_vector(15 downto 0);

signal bpout1,bpout2:std_logic_vector(15 downto 0);

signal SSB_op:std_logic_vector(31 downto 0);

--signal sin1,sin2,cos1,cos2: integer;

signal rf_valid,lo_valid: std_logic;

signal message_signal,lo_signal: std_logic_vector(31 downto 0);

```

```

--sine1<=std_logic_vector( to_signed(sin1,sin1'length));

signal lock_clk: std_logic;

signal clk_9_8_mhz:std_logic;

signal counter: integer range 0 to 255;

signal clk_49khz: std_logic;

signal final_op: std_logic_vector(31 downto 0);

signal final_op_conventional: std_logic_vector(31 downto 0);

signal hilbert_op: std_logic_vector(15 downto 0);

--signal shift_op: std_logic_vector(31 downto 0);

```

```

begin

```

```

clk_9_8mhz_gen : clk_wiz_0

```

```

    port map (

```

```

        -- Clock in ports

```

```

        clk_in1_p => clk_200mhz_p,

```

```

        clk_in1_n => clk_200mhz_n,

```

```

        -- Clock out ports

```

```

        clk_out1 => clk_9_8_mhz,

```

```

        -- Status and control signals

```

```

        reset => reset,

```

```

        locked => lock_clk

```

```

    );

```

```

process(clk_9_8_mhz)

```

```

begin

```

```

    if (reset ='1') then

```

```

clk_49khz<='0';
counter<=0;
elsif rising_edge(clk_9_8_mhz) then
    if (counter =199) then
        clk_49khz <= '1';
        counter <= 0;
    else
        counter <= counter + 1;
        clk_49khz <= '0';
    end if;
end if;
end process;

```

RF : sinecos

```

PORT MAP (
    aclk => clk_49khz,
    aresetn => not(reset),
    m_axis_data_tvalid => rf_valid,
    m_axis_data_tdata => message_signal --1khz
);
sine2<=message_signal(15 downto 0);

```

LO : Losc

```
PORT MAP (  
    aclk => clk_49khz,  
    aresetn => not(reset),  
    m_axis_data_tvalid => lo_valid,  
    m_axis_data_tdata => lo_signal  
);
```

```
sine1<=lo_signal(31 downto 16);--5khz
```

```
cosine1<=lo_signal(15 downto 0); ---5khz
```

mul1: mul

```
port map (  
    clk => clk_49khz,  
    a => bpout1,  
    b => cosine1,  
    p => mulout1);
```

mul2: mul

```
port map (  
    clk => clk_49khz,  
    a => bpout2,  
    b => sine1,  
    p => mulout2);
```

BP1 : band1

```
PORT MAP (  
    aclk => clk_49khz,  
    s_axis_data_tvalid => '1',  
    s_axis_data_tready => open,  
    s_axis_data_tdata => sine2,  
    m_axis_data_tvalid => open,  
    m_axis_data_tdata => bpout1  
);
```

BP2 : band2

```
PORT MAP (  
    aclk => clk_49khz,  
    s_axis_data_tvalid => '1',  
    s_axis_data_tready => open,  
    s_axis_data_tdata => sine2,  
    m_axis_data_tvalid => open,  
    m_axis_data_tdata => bpout2  
);
```

final_filter : fir_compiler_0

```
PORT MAP (  
    aclk => clk_49khz,  
    s_axis_data_tvalid => '1',
```

```

s_axis_data_tready => open,
s_axis_data_tdata => SSB_op,
m_axis_data_tvalid => open,
m_axis_data_tdata => final_op
);

```

```

SSB_op<=mulout1+mulout2;
ssb<=final_op;

```

-----conventional method-----

```

hilbert : fir_compiler_hilbert

```

```

PORT MAP (
    aclk => clk_49khz,
    s_axis_data_tvalid => '1',
    s_axis_data_tready => open,
    s_axis_data_tdata => sine2,
    m_axis_data_tvalid => open,
    m_axis_data_tdata => hilbert_op
);

```

```

--shift_reg : delay_element

```

```

--PORT MAP (
    -- D => sine2,
    --CLK => clk_49khz,
    --Q => shift_op

```

```
-- );
```

```
mul3 : mult_gen_0
```

```
PORT MAP (
```

```
CLK => clk_49khz,
```

```
A => hilbert_op,
```

```
B => sine1,
```

```
P => mulout3
```

```
);
```

```
mult4 : mult_gen_1
```

```
PORT MAP (
```

```
CLK => clk_49khz,
```

```
A => sine2,
```

```
B => cosine1,
```

```
P => mulout4
```

```
);
```

```
conventional_op<=mulout3+mulout4;
```

```
final_filter_conventional : fir_compiler_0
```

```
PORT MAP (
```

```
aclk => clk_49khz,
```

```
s_axis_data_tvalid => '1',
```

```
s_axis_data_tready => open,
```

```
s_axis_data_tdata => conventional_op,
```

```
m_axis_data_tvalid => open,
```

```

        m_axis_data_tdata => final_op_conventional
    );

stim_proc: process(clk_49khz)

    file outfile : text is out "ssb_dual_filter_OUTPUT.txt";

    file outfile1 : text is out "ssb_conventional_OUTPUT.txt";

    variable D :std_logic_vector(31 downto 0);

    variable D1 :std_logic_vector(31 downto 0);

    variable buf : line;

    variable buf1 : line;

    BEGIN

        if clk_49khz'event and clk_49khz='1' then

            D:=final_op;

            D1:=final_op_conventional;

            write(buf,D);

            writeline(outfile,buf);

            write(buf1,D1);

            writeline(outfile1,buf1);

            -- end if;

        end if;

    END PROCESS;

--end process;

end Behavioral;

```


8.6 MATLAB CODE TO READ THE TEXT-FILE AND PLOT THE SPECTRUM:

```
Y=fft(ssbconventionalOUTPUT);
L=length(Y);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
fs = 49e3;
f = fs*(0:(L/2))/L;
figure;
plot(f,20*log10(P1))% plot in db
title('Single-Sided Amplitude Spectrum of Conventional
method')
xlabel('frequency in Hz')
ylabel('amplitude 20log10(X)')

Y=fft(ssbdualfilterOUTPUT);
L=length(Y);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
fs = 49e3;
f = fs*(0:(L/2))/L;
figure;
plot(f,20*log10(P1))% plot in db
title('Amplitude Spectrum of output dual filter')
xlabel('frequency in Hz')
```

```
ylabel('amplitude 20log10(X)')
```