|  |
| --- |
| Signal Processing Hardware WS18/19 |
|  |
| IQ Modulation and De-Modulation for Audio Frequency Signals |
| Report submitted by :  **Bharath Ramachandraiah**  Matrikelnr : 762596  **Bharadwaj Krishnan**  Matrikelnr : 762559  **Aishwarya Prakash**  Matrikelnr : 762591  **Vibha Chandre Gowda**  Matrikelnr : 762525 |
|  |
| **Feb/08/2019** |

Contents

[Introduction to IQ Modulation and IQ Demodulation 4](#_Toc536638053)

[Low Pass Filter 5](#_Toc536638054)

[MATLAB Code and Results 6](#_Toc536638055)

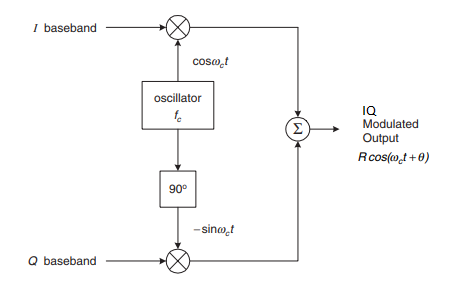
[Intel Quartus Code and Results 7](#_Toc536638056)

Conclusion……………………………………………………………………………………………………………………………………….13

# **Introduction to IQ Modulation and IQ Demodulation**

One modulation technique that lends itself well to digital processes is called "IQ Modulation", where "I" is the "in-phase" component of the waveform, and "Q" represents the quadrature component. In its various forms, IQ modulation is an efficient way to transfer information, and it also works well with digital formats. An IQ modulator can actually create AM, FM and PM. It works something like this:

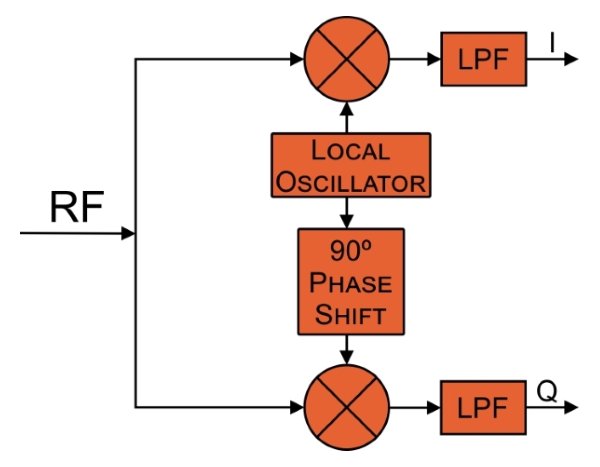
The local oscillator generates the carrier sinusoid. The local oscillator signal itself becomes the I carrier and a 90° phase shift is applied to create the Q carrier. The I and Q carriers are multiplied by the I and Q data streams and the two signals resulting from these multiplications are summed to produce the IQ-modulated signal.



**Figure 1: Block diagram of Modulation of IQ Signals**

Demodulation is the process of extracting the information signal from the carrier wave. In IQ demodulation, we are extracting the In-phase component and the Quadrature component from the modulated signal. This is done by multiplying the modulated input with the In-phase carrier wave and the Quadrature carrier wave.

An IQ Demodulator can modulate signals of AM, FM and PM. The diagram in the next page shows the technique of demodulation. RF serves as the modulated input to the demodulator. The modulated signal after multiplying with the carrier signals is passed through the low pass filter to extract the high frequency components and we get back the original In-phase and Quadrature component.

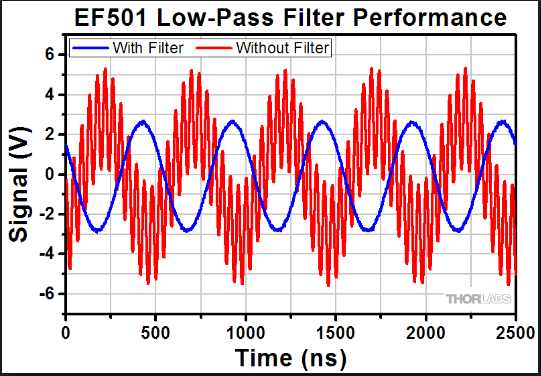


**Figure 2: Block Diagram of De-Modulation of IQ Signals**

# **Low Pass Filter**

A low-pass filter (LPF) is a filter that passes signals with a frequency lower than a selected cut-off frequency and attenuates signals with frequencies higher than the cut-off frequency.

The Low pass filter is used to extract the original I baseband and Q baseband signals from the demodulated signal.



**Figure 3: Signals with and without Low Pass Filter**

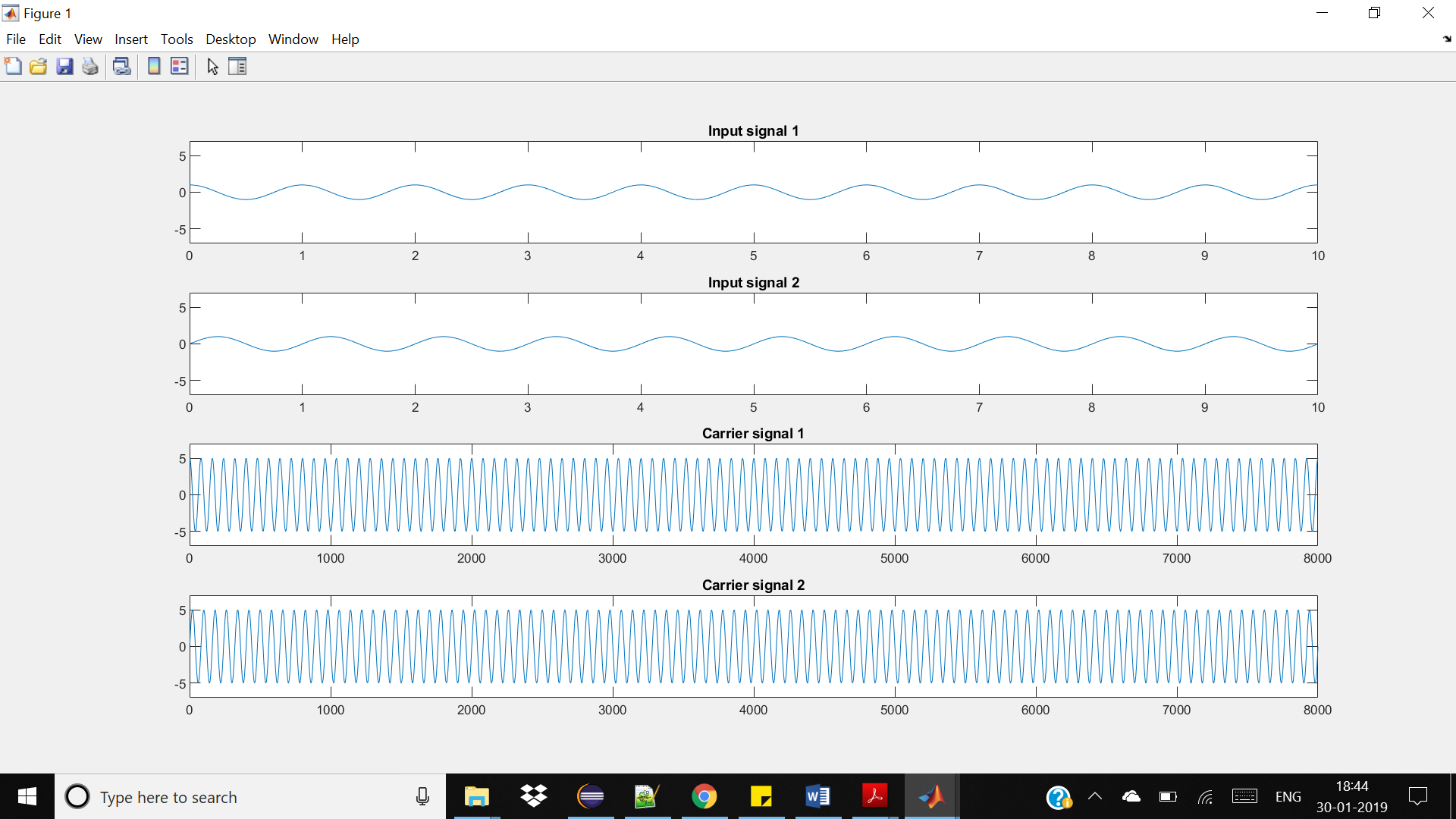
# **MATLAB Code and Results**

As a first step, the Modulation and Demodulation is simulated using MATLAB.

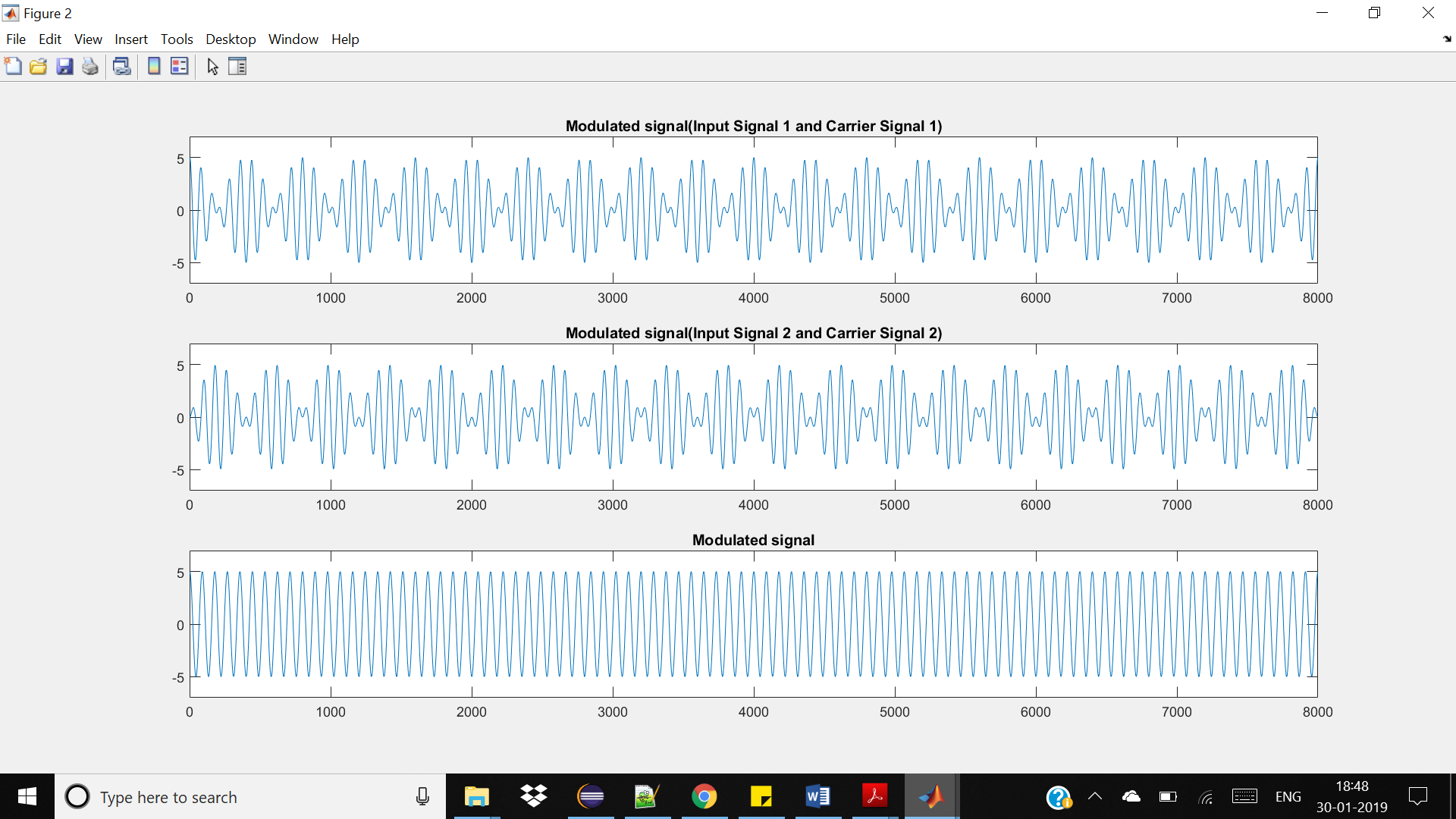
**Procedure**

1. Open MATLAB and create a new file and save with .m extension.
2. Define the frequency for the input signal (Fm) and the frequency for the carrier signal (Fc). [Note: Fc = 10 \* Fm]
3. Generate the Input cosine and sine wave with the frequency Fm for the input wave and do the same for the carrier wave.
4. Perform modulation.
5. Plot the Input wave, carrier wave and the modulated wave.
6. Perform de-modulation and plot the wave.
7. Apply the low pass filter to the de-modulated wave and extract the original input wave and plot them.
8. For validation, perform Frequency spectrum analysis of the input signal and the filtered de-modulated signal.

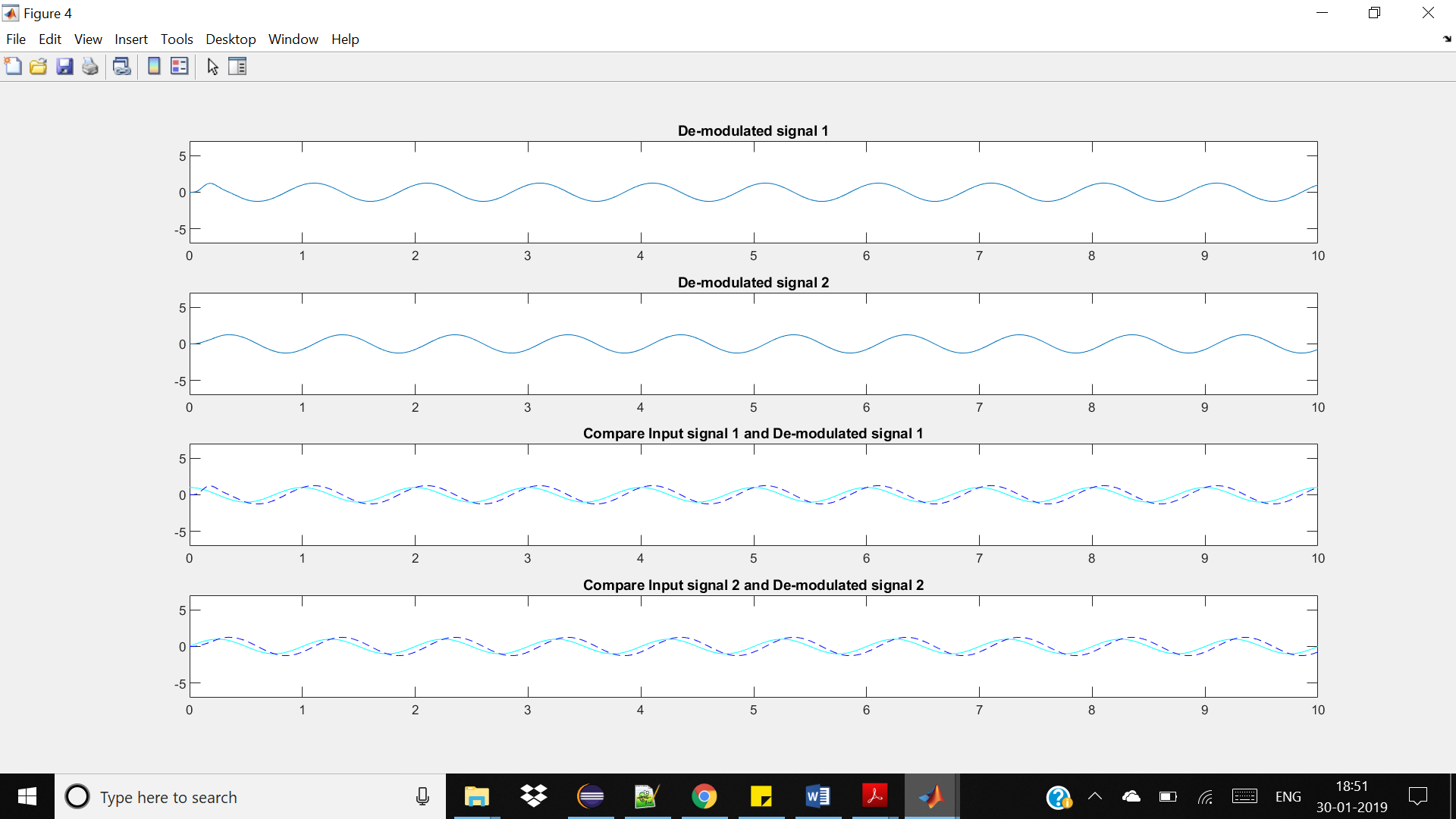
Below are the simulated results



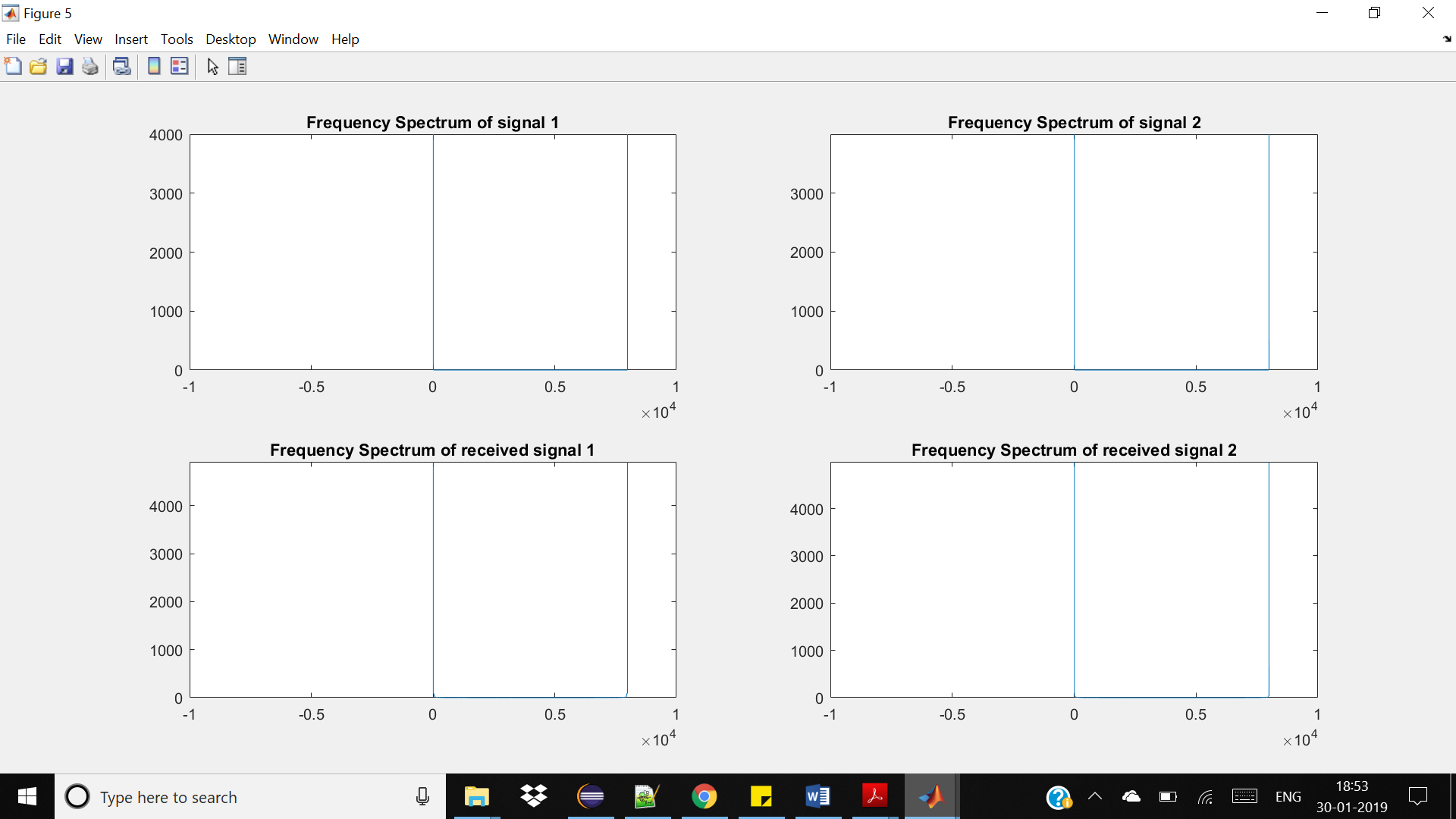
**Figure 4: Input Signal and Carrier Signal**



**Figure 5: The Modulated signals**



**Figure 6: Demodulated Signals**



**Figure 7: Frequency response of the demodulated signals**

# **Intel Quartus Code and Results**

**Procedure:**

1. Open Quartus Prime and start a new project.
2. Select 5CSXFC6D6F31C6N as the FPGA. Create a system Verilog file having and keep the module name and file name same.
3. We shall now generate a look-up table [LUT] which will be used to generate the sine and cosine waves.

**Code to generate the Look-Up table:**

clear all; close all; clc;

*clear all; close all; clc;*

*n = linspace(0,1,257);*

*y = fi(( 0.99\*sin(2\*pi\*n)),1,24,23);*

*set(gcf,'color','w');*

*plot(t,f);*

*box off; axis tight;*

*fileID = fopen('C:\YOUR DIRECTORY','w'\sine\_0\_360\_24bit\_256.txt','w');*

*for i=1:length(f)-1*

*fprintf(fileID,'%s\n',hex(f(i)));*

*end*

*fclose(fileID);*

n = linspace(0,1,257);

y = fi(( 0.99\*sin(2\*pi\*n)),1,24,23);

set(gcf,'color','w');

plot(t,f);

box off; axis tight;

fileID = fopen('C:\YOUR DIRECTORY','w'\sine\_0\_360\_24bit\_256.txt','w');

for i=1:length(f)-1

fprintf(fileID,'%s\n',hex(f(i)));

end

fclose(fileID);

1. As performed in MATLAB, we shall now create the signals for the input wave and the carrier wave.
2. Now, perform modulation.

**Code for modulation**

*module IQ\_mod(*

*input logic clk,input logic reset\_n,*

*input logic signed [23:0] q\_gen\_input1,*

*input logic signed [23:0] q\_gen\_input2,*

*output logic signed [23:0] q\_I\_mod,*

*output logic signed [23:0] q\_Q\_mod,*

*output logic signed [23:0] q\_y\_mod*

*);*

module IQ\_mod(

input logic clk,input logic reset\_n,

input logic signed [23:0] q\_gen\_input1,

input logic signed [23:0] q\_gen\_input2,

output logic signed [23:0] q\_I\_mod,

output logic signed [23:0] q\_Q\_mod,

output logic signed [23:0] q\_y\_mod

);

logic [10:0] time\_base\_carrier = 0;

*logic [10:0] time\_base\_carrier = 0;*

*always\_ff@(posedge clk)*

*if(reset\_n == 1'b0)*

*time\_base\_carrier <= 11'd0;*

*else*

*time\_base\_carrier <= (time\_base\_carrier + 1'b1) % 1041;*

*logic enable\_carrier; assign enable\_carrier = (time\_base\_carrier == 11'd1040) ? 1'b1 : 1'b0;*

*logic [7:0] phase\_accumulator\_carrier = 0;*

*logic [7:0] freq\_tuning\_word = 8'b00000101;*

*always\_ff@(posedge clk)*

*if(reset\_n == 1'b0)*

*phase\_accumulator\_carrier <= 8'd0;*

*else*

*begin*

*if(enable\_carrier)*

*phase\_accumulator\_carrier <= phase\_accumulator\_carrier + freq\_tuning\_word;*

*end*

*logic signed [23:0] lut\_rom\_Sin\_signal [0:2\*\*8-1];*

*logic signed [23:0] lut\_rom\_Cos\_signal [0:2\*\*8-1];*

*initial*

*begin : generate\_signal*

*$readmemh("cosine\_0\_360\_24bit\_256.txt",lut\_rom\_Cos\_signal);*

*$readmemh("sine\_0\_360\_24bit\_256.txt",lut\_rom\_Sin\_signal);*

*end*

*logic signed [23:0] q\_I\_carrier, q\_Q\_carrier;*

*always\_ff@(posedge clk)*

*begin*

*if (enable\_carrier)*

*begin*

*q\_I\_carrier = lut\_rom\_Cos\_signal[phase\_accumulator\_carrier];*

*q\_Q\_carrier = lut\_rom\_Sin\_signal[phase\_accumulator\_carrier];*

*end*

*end*

*logic signed [47:0] I\_mod = 0, Q\_mod = 0;*

*logic signed [48:0] Y\_mod = 0;*

*always\_ff@(posedge clk)*

*begin*

*if (enable\_carrier)*

*I\_mod = q\_Q\_carrier \* q\_gen\_input1;*

always\_ff@(posedge clk)

if(reset\_n == 1'b0)

time\_base\_carrier <= 11'd0;

else

time\_base\_carrier <= (time\_base\_carrier + 1'b1) % 1041;

logic enable\_carrier; assign enable\_carrier = (time\_base\_carrier == ? :

logic [7:0] phase\_accumulator\_carrier = 0;

logic [7:0] freq\_tuning\_word = 8'b00000101;

always\_ff@(posedge clk)

if(reset\_n == 1'b0)

phase\_accumulator\_carrier <= 8'd0;

else

begin

if(enable\_carrier)

phase\_accumulator\_carrier <= phase\_accumulator\_carrier + rd;

end

logic signed [23:0] lut\_rom\_Sin\_signal [0:2\*\*8-1];

logic signed [23:0] lut\_rom\_Cos\_signal [0:2\*\*8-1];

initial

begin : generate\_signal

$readmemh("cosine\_0\_360\_24bit\_256.txt",lut\_rom\_Cos\_signal);

$readmemh("sine\_0\_360\_24bit\_256.txt",lut\_rom\_Sin\_signal);

end

logic signed [23:0] q\_I\_carrier, q\_Q\_carrier;

always\_ff@(posedge clk)

begin

if (enable\_carrier)

begin

q\_I\_carrier = lut\_rom\_Cos\_signal[phase\_accumulator\_carrier];

q\_Q\_carrier = lut\_rom\_Sin\_signal[phase\_accumulator\_carrier];

end

end

logic signed [47:0] I\_mod = 0, Q\_mod = 0;

logic signed [48:0] Y\_mod = 0;

always\_ff@(posedge clk)

begin

if (enable\_carrier)

I\_mod = q\_Q\_carrier \* q\_gen\_input1;

Q\_mod = q\_I\_carrier \* q\_gen\_input2;

*Q\_mod = q\_I\_carrier \* q\_gen\_input2;*

*Y\_mod = (q\_Q\_carrier \* q\_gen\_input1) + (q\_I\_carrier \* q\_gen\_input2);*

*end*

*assign q\_I\_mod = I\_mod[47:24];*

*assign q\_Q\_mod = Q\_mod[47:24];*

*assign q\_y\_mod = Y\_mod[48:25];*

*endmodule*

Y\_mod = (q\_Q\_carrier \* q\_gen\_input1) + (q\_I\_carrier \* q\_gen\_input2);

end

assign q\_I\_mod = I\_mod[47:24];

assign q\_Q\_mod = Q\_mod[47:24];

assign q\_y\_mod = Y\_mod[48:25];

endmodule

1. Perform de-modulation

**Code for de-modulation**

module IQ\_demod(

*module IQ\_demod(*

*input logic clk,input logic reset\_n,*

*input logic signed [23:0] q\_y\_mod,*

*output logic signed [23:0] q\_I\_demod,*

*output logic signed [23:0] q\_Q\_demod*

*);*

*logic [10:0] time\_base\_carrier = 0;*

*always\_ff@(posedge clk)*

*if(reset\_n == 1'b0)*

*time\_base\_carrier <= 11'd0;*

*else*

*time\_base\_carrier <= (time\_base\_carrier + 1'b1) % 1041;*

*logic enable\_carrier; assign enable\_carrier = (time\_base\_carrier == 11'd1040) ? 1'b1 : 1'b0;*

*logic [7:0] phase\_accumulator\_carrier = 0;*

*logic [7:0] freq\_tuning\_word = 8'b00000101;*

*always\_ff@(posedge clk)*

*if(reset\_n == 1'b0)*

*phase\_accumulator\_carrier <= 8'd0;*

*else*

*begin*

*if(enable\_carrier)*

input logic clk,input logic reset\_n,

input logic signed [23:0] q\_y\_mod,

output logic signed [23:0] q\_I\_demod,

output logic signed [23:0] q\_Q\_demod

);

logic [10:0] time\_base\_carrier = 0;

always\_ff@(posedge clk)

if(reset\_n == 1'b0)

time\_base\_carrier <= 11'd0;

else

time\_base\_carrier <= (time\_base\_carrier + 1'b1) % 1041;

logic enable\_carrier; assign enable\_carrier = (time\_base\_carrier == 11'd1040) ? 1'b1 : 1'b0;

logic [7:0] phase\_accumulator\_carrier = 0;

logic [7:0] freq\_tuning\_word = 8'b00000101;

always\_ff@(posedge clk)

if(reset\_n == 1'b0)

phase\_accumulator\_carrier <= 8'd0;

else

begin

if(enable\_carrier)

phase\_accumulator\_carrier <= phase\_accumulator\_carrier + freq\_tuning\_word;

*phase\_accumulator\_carrier <= phase\_accumulator\_carrier + freq\_tuning\_word;*

*end*

*logic signed [23:0] lut\_rom\_Sin\_signal [0:2\*\*8-1];*

*logic signed [23:0] lut\_rom\_Cos\_signal [0:2\*\*8-1];*

*initial*

*begin : generate\_signal*

*$readmemh("cosine\_0\_360\_24bit\_256.txt",lut\_rom\_Cos\_signal);*

*$readmemh("sine\_0\_360\_24bit\_256.txt",lut\_rom\_Sin\_signal);*

*end*

*logic signed [23:0] q\_I\_carrier, q\_Q\_carrier;*

*always\_ff@(posedge clk)*

*begin*

*if (enable\_carrier)*

*begin*

*q\_I\_carrier = lut\_rom\_Cos\_signal[phase\_accumulator\_carrier];*

*q\_Q\_carrier = lut\_rom\_Sin\_signal[phase\_accumulator\_carrier];*

*end*

*end*

*logic signed [47:0] I\_demod = 0, Q\_demod = 0;*

*always\_ff@(posedge clk)*

*begin*

*if (enable\_carrier)*

*begin*

*I\_demod = q\_I\_carrier \* q\_y\_mod;*

*Q\_demod = q\_Q\_carrier \* q\_y\_mod;*

*end*

*end*

*assign q\_I\_demod = I\_demod[47:24];*

*assign q\_Q\_demod = Q\_demod[47:24];*

*endmodule*

end

logic signed [23:0] lut\_rom\_Sin\_signal [0:2\*\*8-1];

logic signed [23:0] lut\_rom\_Cos\_signal [0:2\*\*8-1];

initial

begin : generate\_signal

$readmemh("cosine\_0\_360\_24bit\_256.txt",lut\_rom\_Cos\_signal);

$readmemh("sine\_0\_360\_24bit\_256.txt",lut\_rom\_Sin\_signal);

end

logic signed [23:0] q\_I\_carrier, q\_Q\_carrier;

always\_ff@(posedge clk)

begin

if (enable\_carrier)

begin

q\_I\_carrier = lut\_rom\_Cos\_signal[phase\_accumulator\_carrier];

q\_Q\_carrier = lut\_rom\_Sin\_signal[phase\_accumulator\_carrier];

end

end

logic signed [47:0] I\_demod = 0, Q\_demod = 0;

always\_ff@(posedge clk)

begin

if (enable\_carrier)

begin

I\_demod = q\_I\_carrier \* q\_y\_mod;

Q\_demod = q\_Q\_carrier \* q\_y\_mod;

end

end

assign q\_I\_demod = I\_demod[47:24];

assign q\_Q\_demod = Q\_demod[47:24];

endmodule

1. Apply Low pass filter to the de-modulated wave

**Code for Low – pass filter**

*module lp\_filter(*

*input logic clk, input logic reset\_n, input logic enable,*

*input logic signed [23:0] d,*

*output logic signed [23:0] q*

*);*

*parameter N = 4;*

*logic signed [23:0] delay [N-1:0];*

*logic signed [24:0] sum = 0;*

*integer i;*

*always\_ff@(posedge enable)*

*if(reset\_n == 0)*

*begin*

*for (i=0; i < N; ++i)*

*begin : initialize*

*delay[i] <= 0;*

*end*

*end*

*else*

*begin*

*sum = 0;*

*for (i = 0; i < N; ++i) begin : shift\_fir*

*sum = (sum + delay[i]);*

*if(i == 0)*

*delay[i] <= d;*

*else*

*delay[i] <= delay[i-1];*

*end*

*sum <= ((d + sum) / (N + 1));*

module lp\_filter(

input logic clk, input logic reset\_n, input logic enable,

input logic signed [23:0] d,

output logic signed [23:0] q

);

parameter N = 4;

logic signed [23:0] delay [N-1:0];

logic signed [24:0] sum = 0;

integer i;

always\_ff@(posedge enable)

if(reset\_n == 0)

begin

for (i=0; i < N; ++i)

begin : initialize

delay[i] <= 0;

end

end

else

begin

sum = 0;

for (i = 0; i < N; ++i) begin : shift\_fir

sum = (sum + delay[i]);

if(i == 0)

delay[i] <= d;

else

delay[i] <= delay[i-1];

end

sum <= ((d + sum) / (N + 1));

end

*end*

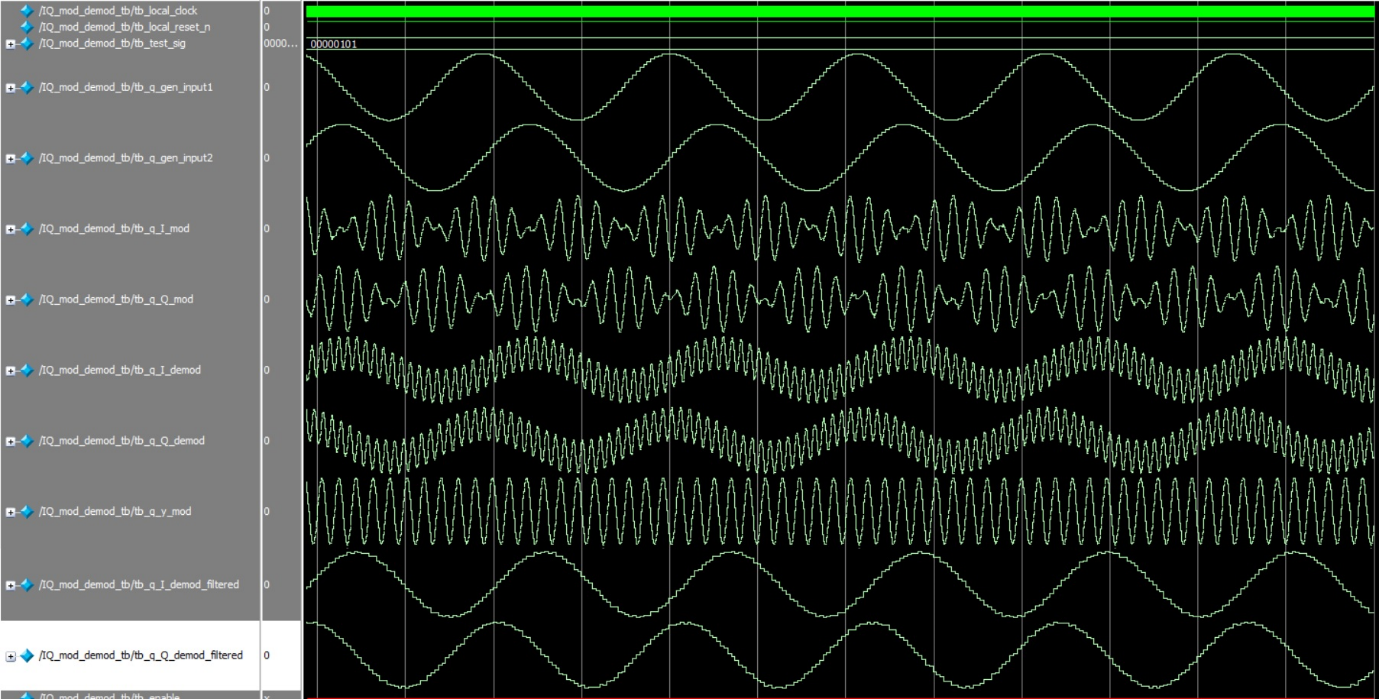
*assign q = sum[24:1];*

*endmodule*

assign q = sum[24:1];

endmodule

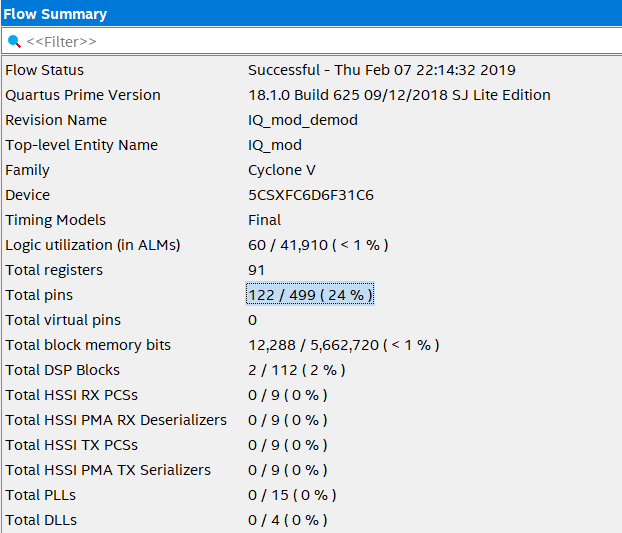
**Screenshots of simulated results**



**Figure 8: IQ Modulation and Demodulation simulation results from Modelsim**

**Conclusion**:

The audio frequency signal is finally modulated and the demodulated signal is passed through the Lowpass filters to get back the original signal which is simulated successfully in Intel Quartus Prime having the following Flow Summary.



**Figure 9: Flow Summary**