

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.....	3
Low Pass Filter.....	4
MATLAB Code and Results.....	5
Intel Quartus Code and Results	7
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.

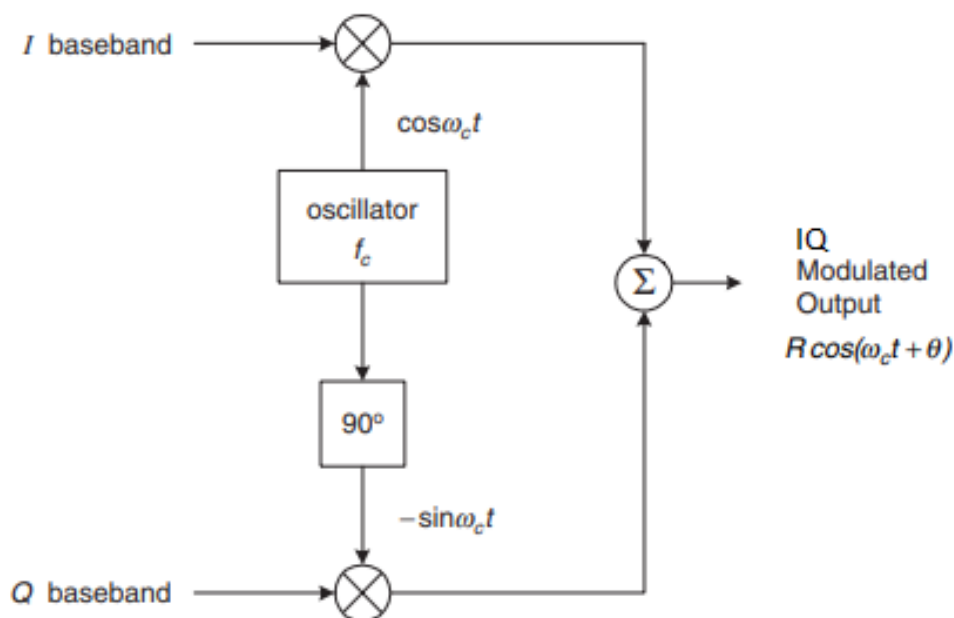


Figure 1: Block diagram of Modulation of IQ Signals

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 demodulate 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.

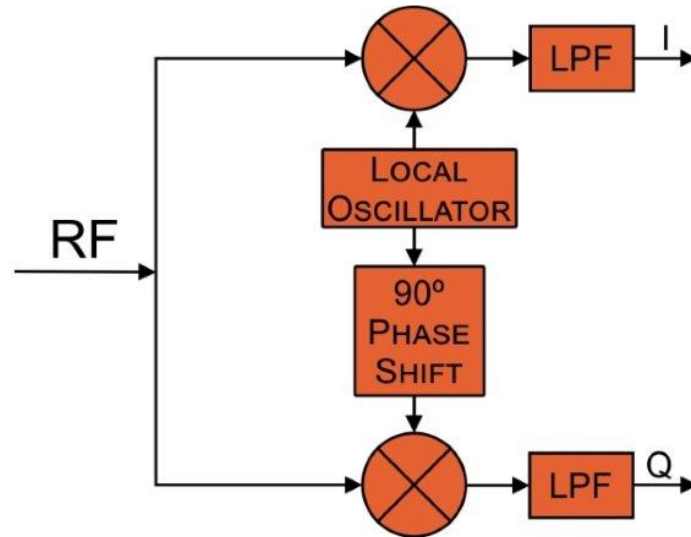


Figure 2: Block Diagram of De-Modulation of IQ Signals

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.

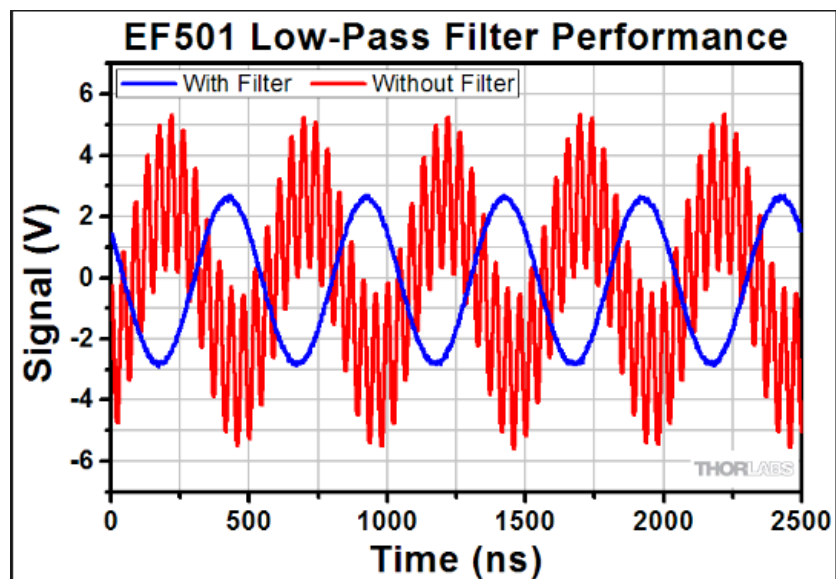


Figure 3: Signals with and without Low Pass Filter

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 (F_m) and the frequency for the carrier signal (F_c).
[Note: $F_c = 10 * F_m$]
3. Generate the Input cosine and sine wave with the frequency F_m 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

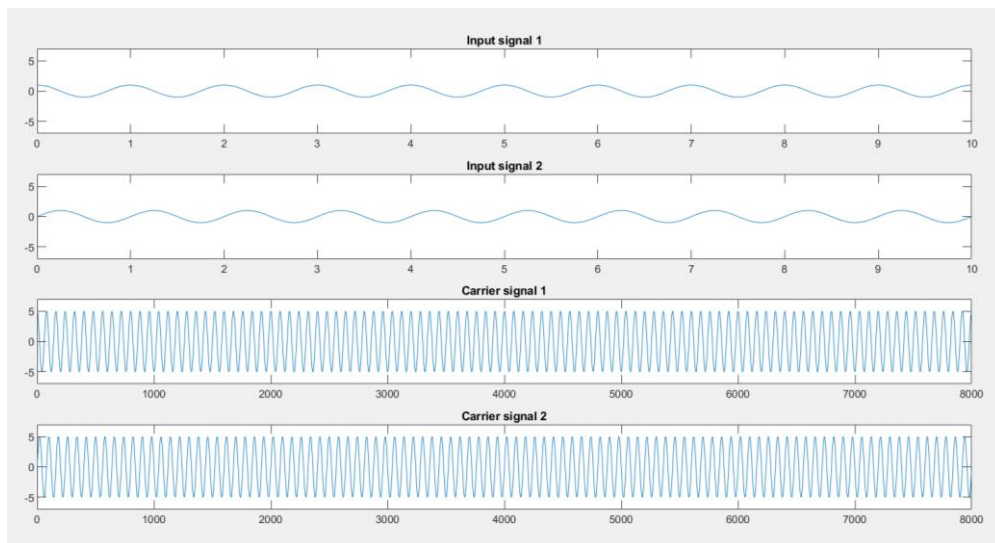


Figure 4: Input Signal and Carrier Signal

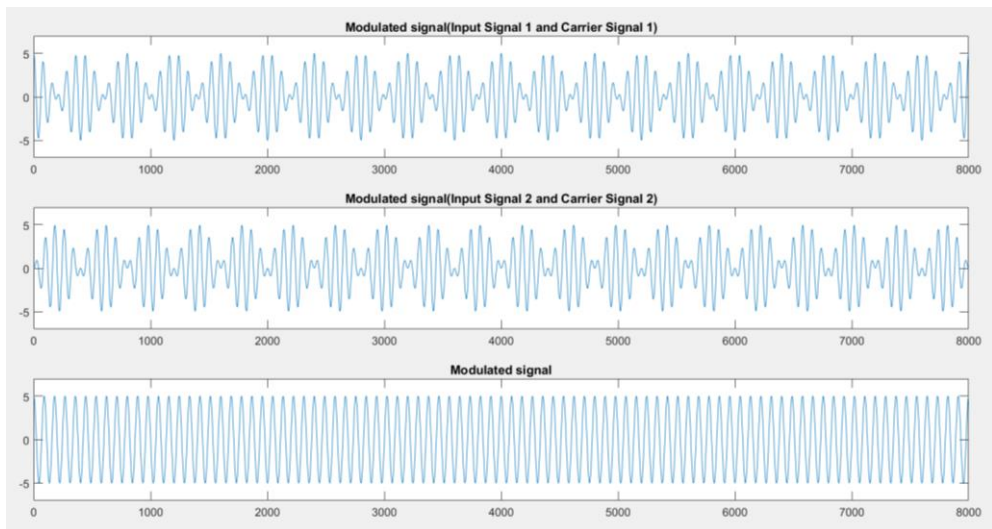


Figure 5: The Modulated signals

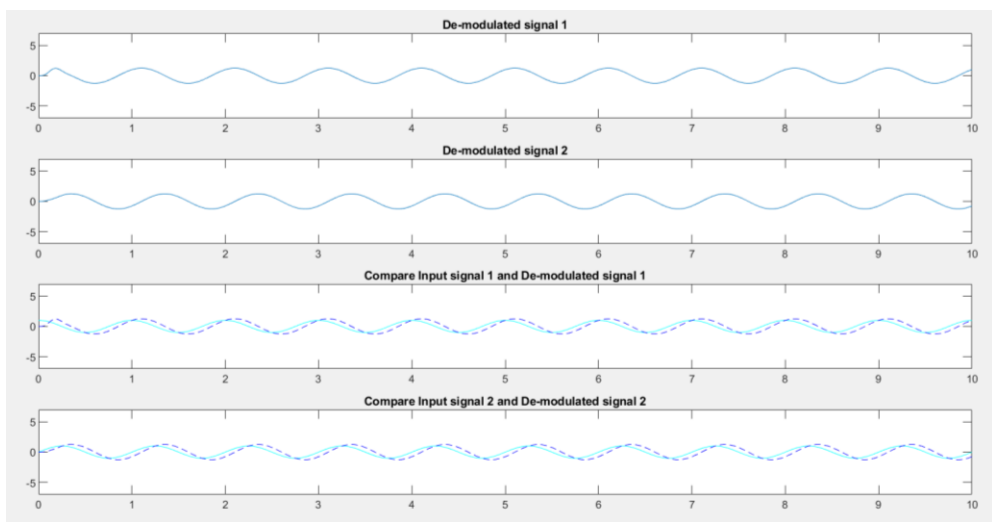


Figure 6: Demodulated Signals

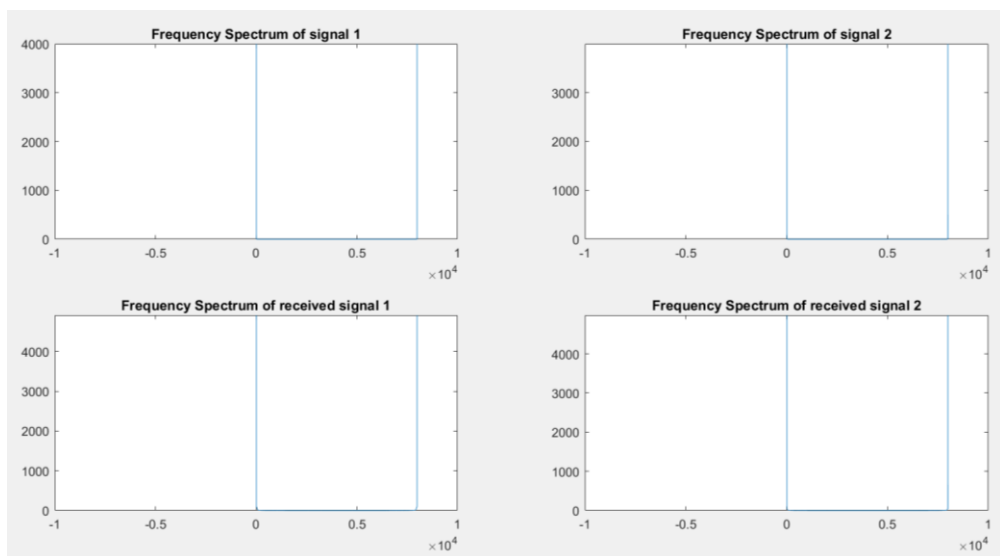


Figure 7: Frequency response of the demodulated signals

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;

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);
```

4. As performed in MATLAB, we shall now create the signals for the input wave and the carrier wave.
5. 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
);
```

```

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;

```



```

Q_mod = q_l_carrier * q_gen_input2;
Y_mod = (q_Q_carrier * q_gen_input1) + (q_l_carrier * q_gen_input2);
end

assign q_l_mod = l_mod[47:24];
assign q_Q_mod = Q_mod[47:24];
assign q_y_mod = Y_mod[48:25];

endmodule

```

6. Perform de-modulation

Code for de-modulation

```

module IQ_demod(
input logic clk,input logic reset_n,
input logic signed [23:0] q_y_mod,
output logic signed [23:0] q_l_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;

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

```

7. 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));  
        end  
    end
```

```
end
```

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

```
endmodule
```

Screenshots of simulated results

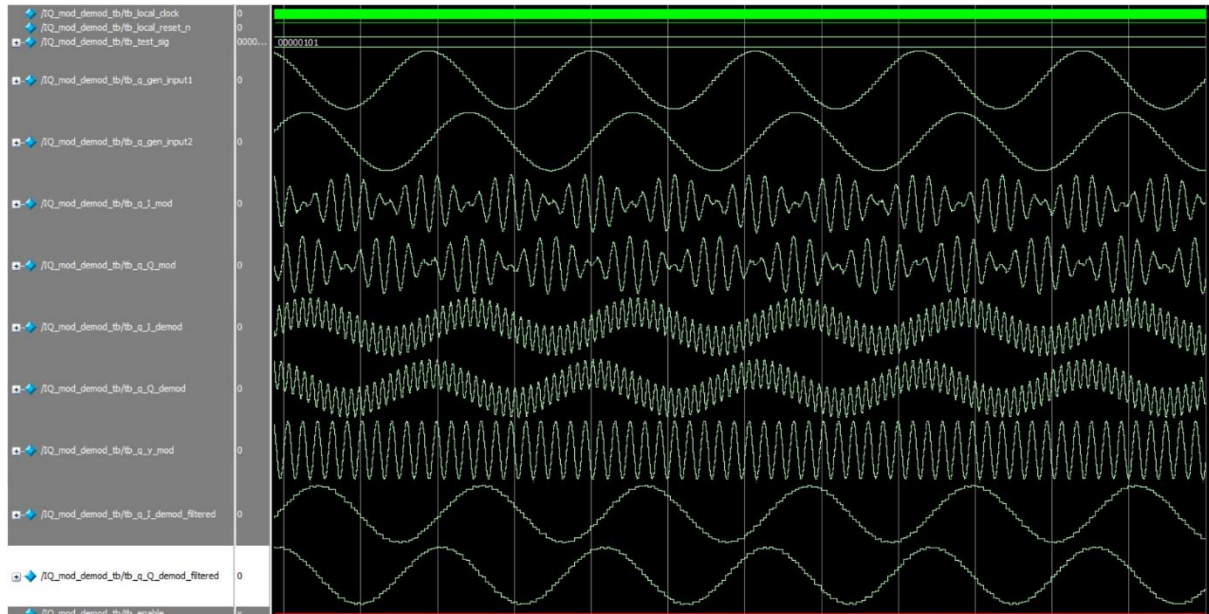


Figure 8: IQ Modulation and Demodulation simulation results from Modelsim

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.


Flow Summary	
 <<Filter>>	
Flow Status	Successful - Thu Feb 07 22:14:32 2019
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	IQ_mod_demod
Top-level Entity Name	IQ_mod
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	60 / 41,910 (< 1 %)
Total registers	91
Total pins	122 / 499 (24 %)
Total virtual pins	0
Total block memory bits	12,288 / 5,662,720 (< 1 %)
Total DSP Blocks	2 / 112 (2 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 9: Flow Summary