Institut Teknologi Bandung

Design of All Digital FM Receiver Circuit

Nursani Rahmatullah
March 2005

Table of Contents

Table of Figures

1. Introduction

The design of the All Digital FM Receiver circuit in this project uses Phase Locked Loop (PLL) as the main core. The task of the PLL is to maintain coherence between the input (modulated) signal frequency, ω_i and the respective output frequency, ω_o via phase comparison. This self-correcting ability of the system also allows the PLL to track the frequency changes of the input signal once it is locked.

Frequency modulated input signal is assumed as a series of numerical values (digital signal) via 8-bit of analog to digital conversion (ADC) circuit. The FM Receiver gets the 8 bit signal every clock cycle and outputs the demodulated signal.

The All Digital FM Receiver circuit is designed using VHDL, then simulated and synthesized using ModelSim SE 6 simulator and Xilinx ISE 6.3i, respectively. FPGA implementation also provided, here we use Virtex2 device. The real measurement is done using ChipScope Pro 6.3i.

2. Architecture Description

The system of All Digital FM Receiver consists of a digital PLL cascaded with digital low pass filter. The block diagram of system is shown in Fig. 1.

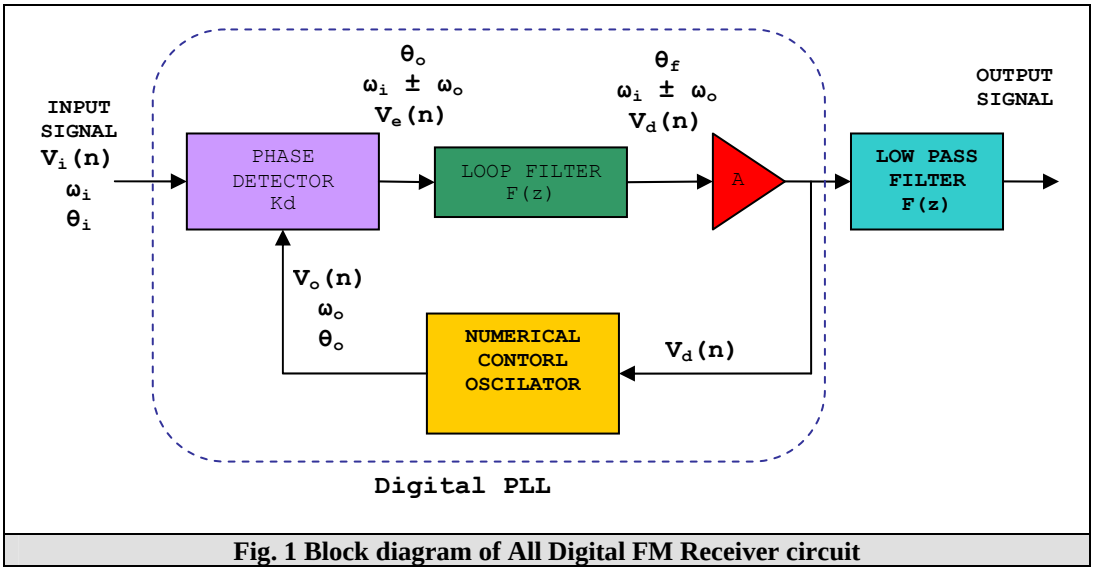

Fig. 1 Block diagram of All Digital FM Receiver circuit

2.1 Phase Detector

Phase Detector (PD) detects phase error between input signal and output signal from NCO. This operation employs a multiplier module. The input signal is frequency modulated, so the input signal $V_i(n)$ can be expressed as follows,

$$V_i(n) = \sin(\omega_i n + \theta_i) \tag{1}$$

Feedback loop mechanism of the PLL will force NCO to generate sinusoidal signal $V_o(n)$ with the same frequency of $V_i(n)$, then

$$V_o(n) = \cos(\omega_i n + \theta_o) \tag{2}$$

Output of phase detector is product of these two signals, using familiar trigonometric identity we obtain

$$\begin{aligned} V_d(n) &= K_d \sin(\omega_i n + \theta_i)\cos(\omega_i n + \theta_o) \\ &= \frac{K_d}{2}\left[\sin(2\omega_i n + \theta_i + \theta_o) + \sin(\theta_i - \theta_o)\right] \end{aligned} \tag{3}$$

K_d is the gain of the phase detector. The first term in (3) corresponds to high frequency component. The second term corresponds to the phase difference between $V_i(n)$ and $V_o(n)$. By removing the first term thru loop filtering, the phase difference can be obtained.

The block diagram of phase detector is a multiplier shown in Fig. 2.

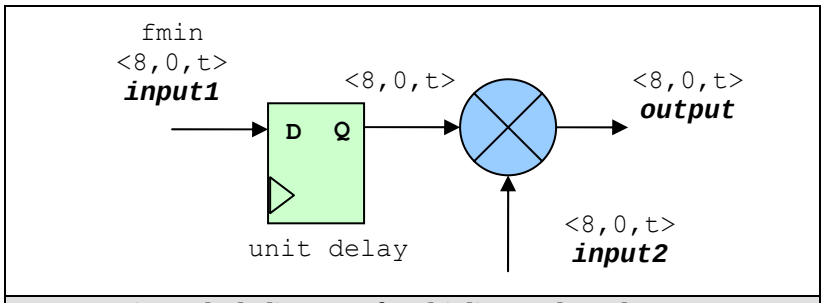

Fig. 2 Block diagram of multiplier as phase detector

Summary of operation:

- *input1* is fmin (modulated data), *input2* is NCO's output. Both input are 2's complement in `<8,0,t>` format, please see [8] for details.
- unit delay is used to synchronize operation,
- then inputs values are multiplied, where *input1* as multiplicand and *input2* as multiplier,
- product will be 16 bit in `<16,0,t>` format, then we scale it by cropping the 8 most bits and feed it to the output in `<8,0,t>` format.

In the VHDL model, we use *Booth's Multiplication algorithm* [2] instead of simple signed arithmetic multiplier operation (denoted by `*`). Arithmetic multiplier will consume large area, while Booth's multiplication algorithm for 8-bit multiplication only needs eight 8-bit adders which is much save in area consumption.

For this algorithm, as shown in Fig. 3, the individual partial products determined from the multiplicand may be: added to, subtracted to, or may not change the final product at all based on the following rules:

- the multiplicand is subtracted from the partial product upon encountering the first 1 in a string of 1's in the multiplier,
- the multiplicand is added to the partial product upon encountering the first 0 provided that there was no previous 1 in a string of 0's in the multiplier,
- the partial product does not change when the bit is identical to the previous multiplier bit.

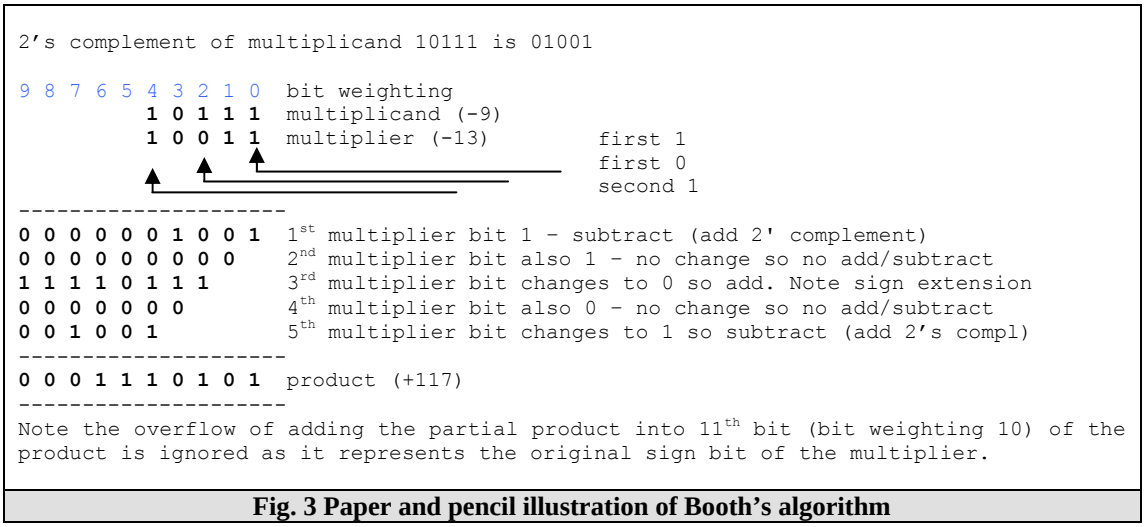

Fig. 3 Paper and pencil illustration of Booth's algorithm

2.2 Loop Filter

Loop filter will remove the high frequency component in (3). Fig. 4 shows the block diagram of a first order loop filter used in the receiver system. In the VHDL model of this block, we need to treat a sign extension from <8,0,t> to <12,4,t> and a multiplication by constant of 15/16.

Summary of operation:

- input *C* is multiplier's output in <8,0,t> format. Output is *D1* <12,4,t>. *D1* will be multiplied by 15/16 and then the product is summed back to *C*
- *dtemp* <12,4,t> is internal signal which is the summing result of *C* and *D1*. *C* must be changed to <12,4,t> before summation, hence,

	<8,0,t>	<12,4,t>
C	C(7 downto 0)	C(7)&C(7)&C(7)&C(7)&C(7 downto 0)

- *dtemp* will be assigned to *D1*. Then *dtemp x 15/16 = dtemp x (1 – 1/16) = dtemp – (dtemp x 1/16) = dtemp - E*

- *E =dtemp x 1/16*, in reality 1/16 multiply can be implemented by just 4 bit right shift operation. Then no multiplier is required.

dtemp <12,4,t>	E <12,4,t> = dtemp x 1/16
dtemp(11 downto 0)	dtemp(11)&dtemp(11)&dtemp(11)&dtemp(11)&dtemp(11 downto 4)

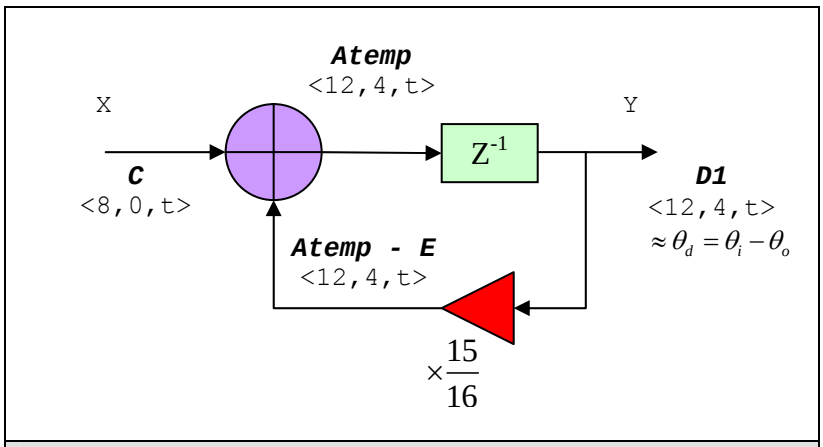

Fig. 4 Block diagram of first order loop filter

First order loop filter as shown in Fig. 4 is a low pass filter with the transfer function

$$H(z) \equiv \frac{Y(z)}{X(z)} = \left(\frac{1}{z - 0.9375} \right) \tag{4}$$

Which has a pole on the real axis at $z = 0.9375$. From stability property of discrete time filter, we know that *H(z)* is stable since its pole is located within the unit circle [1].

2.3 Numerical Controlled Oscillator

Numerical Controlled Oscillator (NCO) will take the corrective error voltage, $V_d(n)$ and then shift its output frequency from its free-running value to the input signal frequency ω_i and thus keep the PLL in lock. The block diagram can be seen in Fig. 5 as follows,

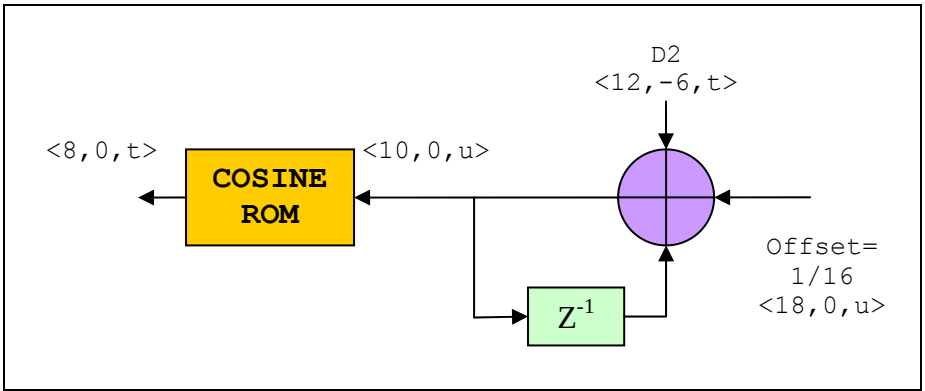

Fig. 5 Block diagram of NCO

Here we assume the NCO free running frequency is 1 MHz and the system clock frequency is 16 MHz; there are 16 sampling points in one cycle of 1 MHz free running frequency. When input is zero, NCO has to generate output equal to free running frequency. Since there are 16 sampling points in one cycle of free running frequency, so the offset must be 1/16. The greater input will produce greater frequency, and vice versa.

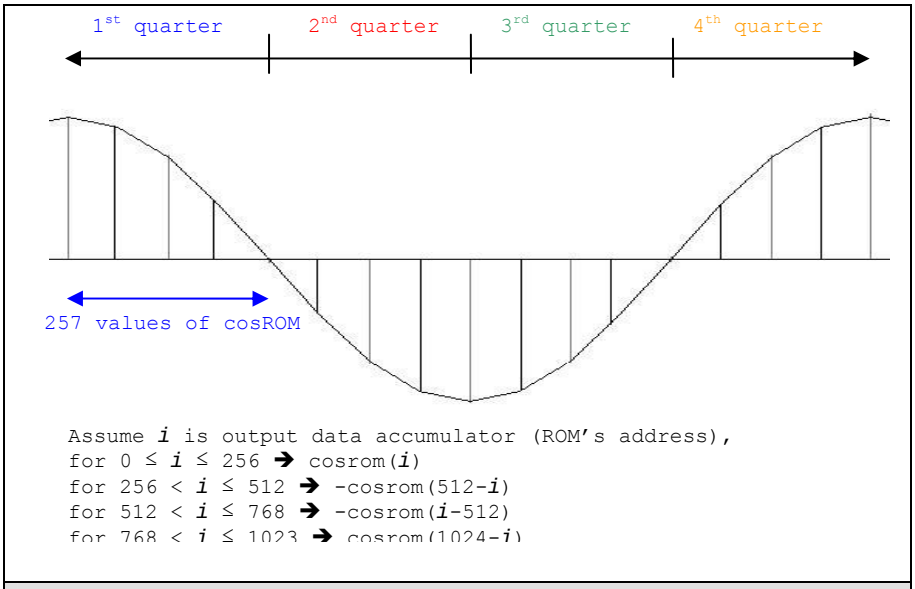

Fig. 6 Data values in one cycle of cosine ROM

The system is a simple integrator which accumulates the input value and maps it into predefined cosine ROM. All 1024 values were given (*file: cos.txt*) to define one cycle of cosine signal, but we actually don't need to use all of these values. Since one cycle can be divided to four quarter, we only need to define the first quarter with 257 values. The remains quarters are duplicated form the first quarter, where the opposite sign is applied to second and third quarter. Illustration is shown in Fig 6.

Summary of operation:

- input *D2* and *offset* are added, note that signed extension form `<12,-6,t>` to `<18,0,u>`.

- the addition result then accumulated by modulo accumulator, then we take 10 most bits as ROM address.

- Address will be mapped to data values in ROM.

2.4 FIR Filter

The last stage of the receiver system is to perform signal shaping. Here we use 16 tap Finite Impulse Response (FIR) filter to perform digital low pass filter. This filter is essentially average filter since its output is equal to the average value of its input over the last *n*-tap samples, where *n* is number of tap used [4]. This configuration needs 16 coefficients, but simplification is taken by assuming all of the coefficients are the same, 1/16. In reality 1/16 multiply can be implemented by just 4 bit right shift operation. Then no multiplier is required.

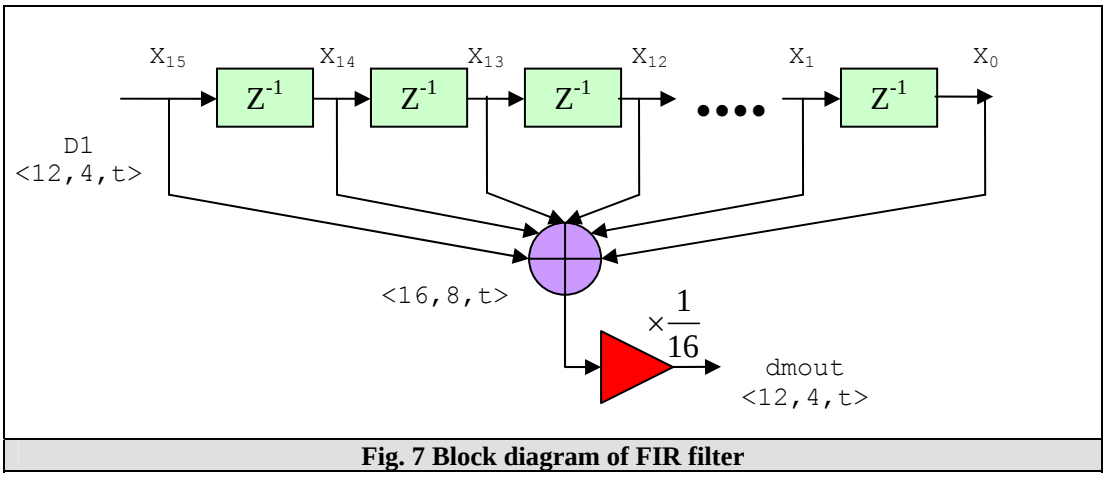

Fig. 7 Block diagram of FIR filter

3. Functional Explanation

Digital PLL system is composed of three basic parts: (1) Phase Detector (PD), (2) Loop filter, (3) Numerical-controlled oscillator (NCO). The complete block diagram of the All Digital FM receiver circuit is shown in Fig. 8.

With no signal input applied to the system. The NCO control voltage $V_d(n)$ is equal to zero. The NCO operates at a set frequency, f_o (or the equivalent radian frequency, ω_o) which is known as the free running frequency. When an input signal is applied to the system, the phase detector compares the phase and the frequency of the input with the NCO frequency and generates an error voltage $V_e(n)$ that is related to the phase and the frequency difference between the two signals.

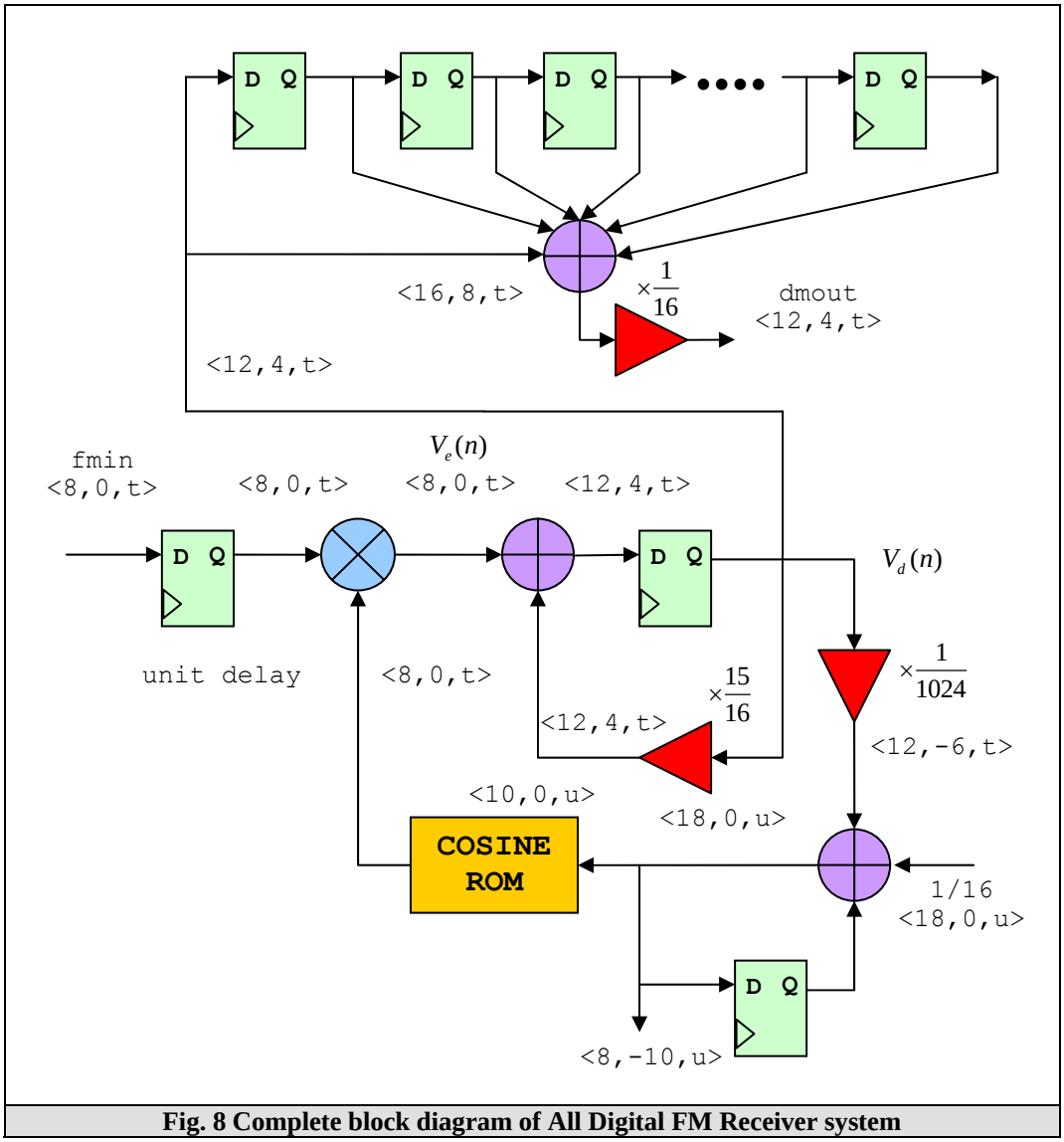

Fig. 8 Complete block diagram of All Digital FM Receiver system

This error voltage is then filtered, amplified by factor of *A = 1/1024*, and applied to the control terminal of the NCO. In this manner, the control voltage $V_d(n)$ forces the NCO frequency to vary in a direction that reduces the frequency difference between ω_o and the input signal. If the input frequency ω_i is sufficiently close to ω_o, the feedback nature of the PLL causes the NCO to synchronize or lock with the incoming signal. Once in lock, the NCO frequency is identical to the input signal except for a finite phase difference.

This net phase difference of θ_e where

$$\theta_e = \theta_i - \theta_o \tag{5}$$

is necessary to generate the corrective error voltage $V_d(n)$ to shift the NCO frequency from its free-running value to the input signal frequency ω_i and thus keep the PLL in lock. This self-correcting ability of the system also allows the PLL to track the

frequency changes of the input signal once it is locked, hence it can be act as FM demodulator in receiver system.

Another means of describing the operation of the PLL is to observe that the phase detector is in actuality a multiplier circuit that mixes the input signal with the NCO signal. This mix produces the sum and difference frequencies $(\omega_i \pm \omega_o)$ shown in (3). When the loop is in lock, the NCO duplicates the input frequency so that the difference frequency component $(\omega_i - \omega_o)$ is zero; hence, the output of the phase comparator contains only a DC component. The loop filter removes the sum frequency component $(\omega_i + \omega_o)$ but passes the DC component which is then amplified and fed back to the NCO.

The single most important point to realize when designing with the PLL is that it is a feedback system and, hence, is characterized mathematically by the same equations that apply to other, more conventional feedback control systems [5]. Mathematical model of the all digital PLL system can be derived to analyze the transient and steady state response. The block diagram of the all digital PLL system in *z* domain (discrete time) and its transformation in *s* domain (continuous time) is shown in Fig. 9.

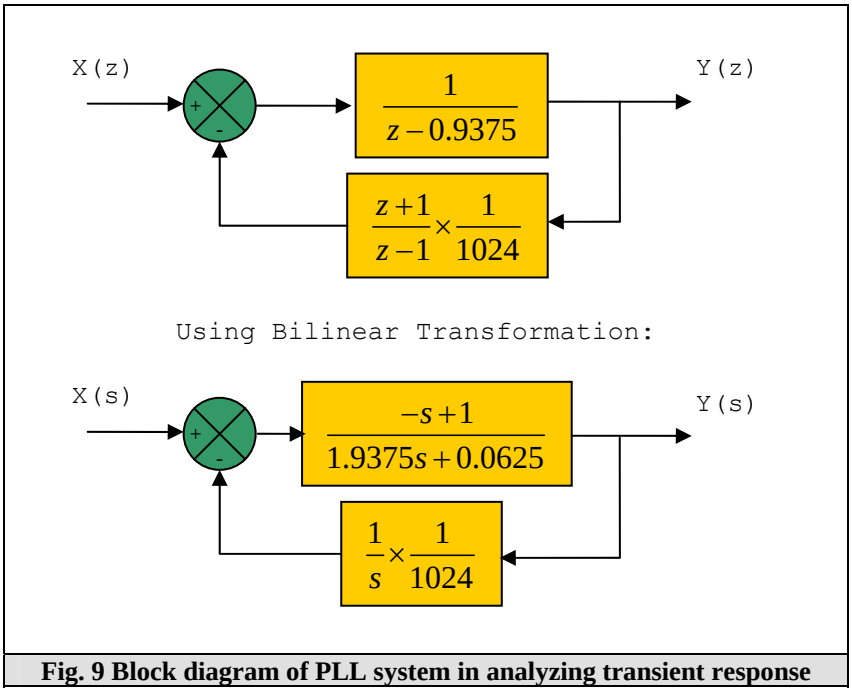

Fig. 9 Block diagram of PLL system in analyzing transient response

Since a physical control system involves energy storage, the output of the system, when subjected to an input, cannot follow the input immediately but exhibits a transient response before a steady state can be reached [3].

The transfer function of the system is

$$\frac{Y(s)}{X(s)} = \frac{-s^2 + s}{1.9375s^2 + 0.06161s + 0.00089} \tag{6}$$

Hence, the PLL system is a second order system. In the test for stability we subjected the system with test signal representing a unit step of frequency at constant phase, this test signal correspond with actual input signal which is a FM modulated signal [5].

Using MATLAB, we can plot unit step response curve for the system as shown in Fig. 10. We see that the system is stable with overshoots at the transient state.

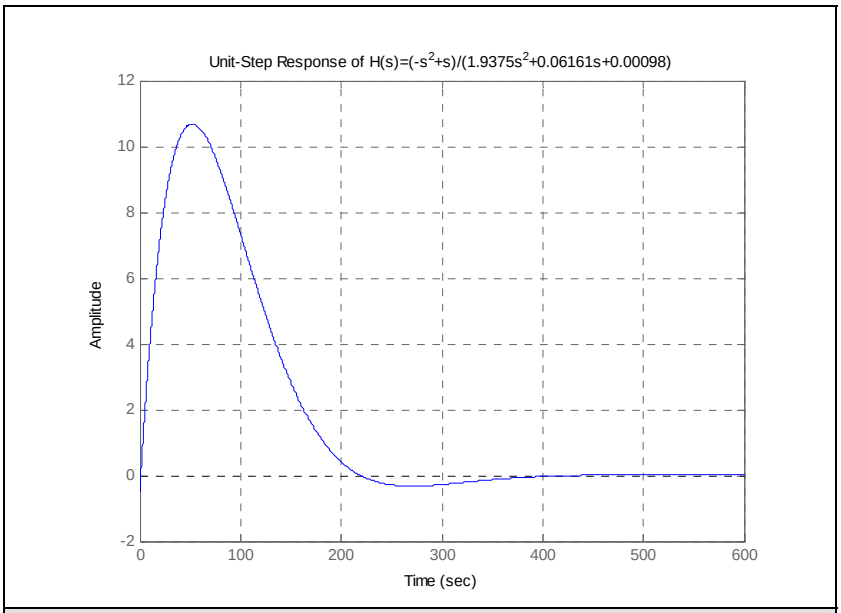

Fig. 10 Unit step response for PLL system used in FM receiver system

4. Critical Path Speed and Circuit Area

Design is synthesized with Xilinx Synthesize Tool (XST), here we use Virtex2 technology with xc2v2000ff896 device and -6 of speed grade.

- Unit delay

  The unit speed from synthesizing of 50 input XOR gate (see Appendix for details) give us the result 6.967 ns total delay and 5 levels, then unit delay is 6.967/5 = 1.393 ns.

- Unit area

  The unit area from synthesizing of 50 input XOR gate (see Appendix for details) give us the result 102 total area gate count and utilize 17 cells, then unit area is 102/17 = 6.

Here is the synthesis result of critical path speed,

```
Timing Summary:
---------------
Speed Grade: -6

   Minimum period: 8.781ns (Maximum Frequency: 113.889MHz)
   Minimum input arrival time before clock: 1.329ns
   Maximum output required time after clock: 4.575ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

--------------------------------------------------------------------------------
Timing constraint: Default period analysis for Clock 'clk'
Delay:               8.781ns (Levels of Logic = 12)
  Source:            I2_dtemp1_4 (FF)
  Destination:       I2_dout_2 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising

  Data Path: I2_dtemp1_4 to I2_dout_2
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDC:C->Q             12   0.449   0.688  I2_dtemp1_4 (I2_dtemp1_4)
     LUT2_D:I0->LO         1   0.347   0.100  I2_Ker240621 (N65860)
     LUT4:I3->O            5   0.347   0.569  I2_Ker249151 (I2_N24917)
     LUT4:I3->O            2   0.347   0.518  I2_Ker2252582 (CHOICE4276)
     LUT3:I0->O            1   0.347   0.000  I2_Ker2252593_G (N65481)
     MUXF5:I1->O           1   0.345   0.383  I2_Ker2252593 (CHOICE4278)
     LUT4:I0->O            1   0.347   0.382  I2_Ker22525105 (CHOICE4279)
     LUT4_L:I3->LO         1   0.347   0.100  I2_Ker22525154 (CHOICE4291)
     LUT2:I1->O            2   0.347   0.519  I2_Ker22525164 (I2_N22527)
     LUT4_L:I3->LO         1   0.347   0.100  I2_Ker22312138 (CHOICE4400)
     LUT4:I2->O            2   0.347   0.518  I2_Ker22312162 (I2_N22314)
     LUT4:I3->O            1   0.347   0.000  I2__n0008<2>323_G (N65471)
     MUXF5:I1->O               1     0.345     0.000    I2__n0008<2>323
(I2__n0008<2>)
     FDC:D                     0.293          I2_dout_2
    ----------------------------------------
    Total                      8.781ns (4.902ns logic, 3.878ns route)
                                       (55.8% logic, 44.2% route)

--------------------------------------------------------------------------------
```

We conclude that the normalized combinational path delay is 8.781/1.393 = 6.304 unit delay

While the synthesized result for circuit area is,

```
Design Summary
--------------
Number of errors:      0
Number of warnings:    8
Logic Utilization:
  Number of Slice Flip Flops:         446 out of  21,504    2%
  Number of 4 input LUTs:           1,226 out of  21,504    5%
Logic Distribution:
  Number of occupied Slices:          834 out of  10,752    7%
  Number of Slices containing only related logic:     834 out of     834  100%
  Number of Slices containing unrelated logic:          0 out of     834    0%
Total Number 4 input LUTs:          1,248 out of  21,504    5%
  Number used as logic:             1,226
  Number used as a route-thru:         22
  Number of bonded IOBs:               22 out of     624    3%
    IOB Flip Flops:                    20
  Number of GCLKs:                      1 out of      16    6%

Total equivalent gate count for design:  13,835
```

We conclude that the normalized circuit area is 13,835/6 = 2,306 gates

5. Appealing Point and Originality

The architecture used in this design has been explained in [7]. This architecture is good. We did something different by optimizing phase detector component to achieve smaller circuit area, and we also modify NCO component.

We optimized the multiplication operation used in the phase detector component. We used Booth's algorithm to replace arithmetic multiplier with some adders. This modification reduces the number of gate for this component from 689 gates decrease to 453 gates.

For NCO component, we only need 257x8-bit ROM rather than 1024x8-bit ROM since one cycle of cosine wave can be divided into four quarter as explained before. Although this modification uses smaller size of ROM, we can't avoid using more registers and several comparators, but it's interesting to work with.

We attempted to find another digital PLL architecture like one which was proposed in [8]. We realize that it is also good and easy to build, but it needs high frequency of clock to drive the counters. Finally we try to implement our design into FPGA, and then we need to do real measurement. The result gives us the correct demodulated output wave as expected.

6. HDL Codes

HDL codes for each component, top level design, and the test bench can be observed as follows,

6.1 Multiplier (Phase Detector)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE IEEE.numeric_std.ALL;

ENTITY multiplier IS
port (CLK      : in std logic;                          -- Declarations
      RESET    : in std logic;
      input1   : in std_logic_vector(7 downto 0);       -- input1 as multiplicand <8,0,t>
      input2   : in signed(7 downto 0);                 -- input2 as multiplier <8,0,t>
      output   : out signed(7 downto 0)                 -- product <8,0,t>
      );
END multiplier ;

ARCHITECTURE behavior OF multiplier IS

signal out temp : signed(15 downto 0);                  -- output buffer
signal input1 buf : signed(15 downto 0);                -- multiplicand buffer
signal part0,part1,part2,part3,part4,                   -- 8 partials product
       part5,part6,part7 : signed(15 downto 0);

begin
process(CLK, RESET)
begin
 if (RESET='1') then
   out_temp <= (others => '0');
```

```
   output <= (others => '0');
   input1 buf <= (others => '0');
   part0 <= (others => '0');
   part1 <= (others => '0');
   part2 <= (others => '0');
   part3 <= (others => '0');
   part4 <= (others => '0');
   part5 <= (others => '0');
   part6 <= (others => '0');
   part7 <= (others => '0');
  elsif rising_edge(CLK) then
    input1 buf <= input1(7)&input1(7)&input1(7)&
                  input1(7)&input1(7)&input1(7)&   -- input buffering with sign
                  input1(7)&input1(7)&             extension
                  signed(input1);                  -- start Booth's algorithm
    if (input2(0)='1') then                        -- check first bit of multiplier
    part0 <= -(input1 buf);                        -- subtract (add 2's complement)
    else
    part0 <= (others => '0');                      -- no change
    end if;

    if (input2(1)='1') then                        -- check second bit of multiplier
    if (input2(0)='1') then                        -- check previous bit
    part1 <= (others => '0');                      -- no change
    else
    part1 <= -(input1 buf);                        -- subtract (add 2's complement)
    end if;
    else
    if (input2(0)='1') then
    part1 <= input1 buf;                           -- add
    else
    part1 <= (others => '0');
    end if;
    end if;

    if (input2(2)='1') then
    if (input2(1)='1') then
    part2 <= (others => '0');
    else
    part2 <= -(input1 buf);
    end if;
    else
    if (input2(1)='1') then
    part2 <= input1_buf;
    else
    part2 <= (others => '0');
    end if;
    end if;

    if (input2(3)='1') then
    if (input2(2)='1') then
    part3 <= (others => '0');
    else
    part3 <= -(input1 buf);
    end if;
    else
    if (input2(2)='1') then
    part3 <= input1 buf;
    else
    part3 <= (others => '0');
    end if;
    end if;

    if (input2(4)='1') then
    if (input2(3)='1') then
    part4 <= (others => '0');
    else
    part4 <= -(input1_buf);
    end if;
    else
    if (input2(3)='1') then
    part4 <= input1 buf;
    else
    part4 <= (others => '0');
    end if;
    end if;
```

```
    if (input2(5)='1') then
    if (input2(4)='1') then
    part5 <= (others => '0');
    else
    part5 <= -(input1 buf);
    end if;
    else
    if (input2(4)='1') then
    part5 <= input1_buf;
    else
    part5 <= (others => '0');
    end if;
    end if;

    if (input2(6)='1') then
    if (input2(5)='1') then
    part6 <= (others => '0');
    else
    part6 <= -(input1 buf);
    end if;
    else
    if (input2(5)='1') then
    part6 <= input1 buf;
    else
    part6 <= (others => '0');
    end if;
    end if;

    if (input2(7)='1') then
    if (input2(6)='1') then
    part7 <= (others => '0');
    else
    part7 <= -(input1_buf);
    end if;
    else
    if (input2(6)='1') then
    part7 <= input1_buf;
    else
    part7 <= (others => '0');
    end if;
    end if;
   out temp <= part0+(part1(14 downto 0)&'0')+      -- summing partials product
               (part2(13 downto 0)&"00")+
               (part3(12 downto 0)&"000")+
               (part4(11 downto 0)&"0000")+
               (part5(10 downto 0)&"00000")+
               (part6(9 downto 0)&"000000")+
               (part7(8 downto 0)&"0000000");
   output <= out_temp(15 downto 8);                 --  crop  8  most  bits  as  final
  end if;                                           product
end process;
END behavior;
```

6.2 Loop Filter

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE IEEE.numeric std.ALL;

ENTITY loop filter IS                               -- Declarations
port ( CLK   : in std_logic;
       RESET : in std_logic;
       C     : in signed(7 downto 0);               -- input <8,0,t> from multiplier
       D1    : out signed(11 downto 0);             -- output <12,4,t> to FIR
       D2    : out signed(11 downto 0)              -- output <12,-6,t> to NCO
       );
END loop_filter  ;
```

```
ARCHITECTURE behavior OF loop filter  IS

signal E     : signed(11 downto 0);                  -- (Atemp x 1/16)
signal dtemp : signed(11 downto 0);                  -- output buffer

begin
process(CLK, RESET)
begin
 if (RESET='1') then
  D1 <= (others => '0');                             -- 15/16 = (1 - 1/16),
  D2 <= (others => '0');                             -- hence, Atemp x 15/16 = Atemp-
  E <= (others => '0');                              (Atemp x 1/16) = Atemp - E
  dtemp <= (others => '0');
 elsif rising edge(CLK) then
  dtemp <=  (C(7)&C(7)&C(7)&C&'0') + dtemp - E;      -- here we scaled input to get
  E <= dtemp(11)&dtemp(11)&dtemp(11)&dtemp(11)&      better result
       dtemp(11 downto 4);                           -- 1/16 multiply is 4 bit right
  D1 <= dtemp;                                       shift  operation
  D2 <= dtemp(11 downto 4)&”0000”;                   -- 1/1024 multiply is 10 bit right
 end if;                                             shift operation
end process;                                         -- D2 = D1 x 1/1024
END behavior;                                        -- note that to get D2, D1 must be
                                                     changed to 18 bit then do the 10
                                                     bit shift right operation and then
                                                     change it to <12,-6,t> format.
```

6.3 Numerical Controlled Oscillator (NCO)

```
LIBRARY ieee;
USE ieee.std logic 1164.all;
USE IEEE.numeric std.ALL;

ENTITY nco IS
port(clk   : in  std_logic;                          -- Declarations
     reset : in  std_logic;
     din   : in  signed(11 downto 0);                --  input  <12,-6,t>  from  loop
     dout  : out signed(7 downto 0)                  filter
     );                                              --  output  data  from  cosine  ROM
END nco ;                                            <8,0,t>

ARCHITECTURE behavior OF nco IS
type vectype is array (0 to 256) of                  -- using first quarter data (257)
     signed(7 downto 0);                             values of file: cos.txt
constant cosrom  : vectype := (                      -- cosine ROM
0 => "01111111",
1 => "01111111",
2 => "01111111",
3 => "01111111",
4 => "01111111",
5 => "01111111",
6 => "01111111",
7 => "01111111",
8 => "01111111",
9 => "01111111",
10 => "01111111",
11 => "01111111",
12 => "01111111",
13 => "01111111",
14 => "01111111",
15 => "01111111",
16 => "01111111",
17 => "01111111",
18 => "01111111",
19 => "01111111",
20 => "01111111",
21 => "01111111",
22 => "01111111",
23 => "01111111",
24 => "01111111",
```

```
25 => "01111110",
26 => "01111110",
27 => "01111110",
28 => "01111110",
29 => "01111110",
30 => "01111110",
31 => "01111110",
32 => "01111110",
33 => "01111101",
34 => "01111101",
35 => "01111101",
36 => "01111101",
37 => "01111101",
38 => "01111101",
39 => "01111100",
40 => "01111100",
41 => "01111100",
42 => "01111100",
43 => "01111100",
44 => "01111011",
45 => "01111011",
46 => "01111011",
47 => "01111011",
48 => "01111010",
49 => "01111010",
50 => "01111010",
51 => "01111010",
52 => "01111010",
53 => "01111001",
54 => "01111001",
55 => "01111001",
56 => "01111001",
57 => "01111000",
58 => "01111000",
59 => "01111000",
60 => "01110111",
61 => "01110111",
62 => "01110111",
63 => "01110111",
64 => "01110110",
65 => "01110110",
66 => "01110110",
67 => "01110101",
68 => "01110101",
69 => "01110101",
70 => "01110100",
71 => "01110100",
72 => "01110100",
73 => "01110011",
74 => "01110011",
75 => "01110011",
76 => "01110010",
77 => "01110010",
78 => "01110010",
79 => "01110001",
80 => "01110001",
81 => "01110001",
82 => "01110000",
83 => "01110000",
84 => "01101111",
85 => "01101111",
86 => "01101111",
87 => "01101110",
88 => "01101110",
89 => "01101101",
90 => "01101101",
91 => "01101101",
92 => "01101100",
93 => "01101100",
94 => "01101011",
95 => "01101011",
96 => "01101010",
97 => "01101010",
98 => "01101010",
99 => "01101001",
100 => "01101001",
```

```
101 => "01101000",
102 => "01101000",
103 => "01100111",
104 => "01100111",
105 => "01100110",
106 => "01100110",
107 => "01100101",
108 => "01100101",
109 => "01100100",
110 => "01100100",
111 => "01100011",
112 => "01100011",
113 => "01100010",
114 => "01100010",
115 => "01100001",
116 => "01100001",
117 => "01100000",
118 => "01100000",
119 => "01011111",
120 => "01011111",
121 => "01011110",
122 => "01011110",
123 => "01011101",
124 => "01011101",
125 => "01011100",
126 => "01011100",
127 => "01011011",
128 => "01011011",
129 => "01011010",
130 => "01011001",
131 => "01011001",
132 => "01011000",
133 => "01011000",
134 => "01010111",
135 => "01010111",
136 => "01010110",
137 => "01010101",
138 => "01010101",
139 => "01010100",
140 => "01010100",
141 => "01010011",
142 => "01010010",
143 => "01010010",
144 => "01010001",
145 => "01010001",
146 => "01010000",
147 => "01001111",
148 => "01001111",
149 => "01001110",
150 => "01001110",
151 => "01001101",
152 => "01001100",
153 => "01001100",
154 => "01001011",
155 => "01001010",
156 => "01001010",
157 => "01001001",
158 => "01001000",
159 => "01001000",
160 => "01000111",
161 => "01000111",
162 => "01000110",
163 => "01000101",
164 => "01000101",
165 => "01000100",
166 => "01000011",
167 => "01000011",
168 => "01000010",
169 => "01000001",
170 => "01000001",
171 => "01000000",
172 => "00111111",
173 => "00111110",
174 => "00111110",
175 => "00111101",
176 => "00111100",
```

```
177 => "00111100",
178 => "00111011",
179 => "00111010",
180 => "00111010",
181 => "00111001",
182 => "00111000",
183 => "00111000",
184 => "00110111",
185 => "00110110",
186 => "00110101",
187 => "00110101",
188 => "00110100",
189 => "00110011",
190 => "00110011",
191 => "00110010",
192 => "00110001",
193 => "00110000",
194 => "00110000",
195 => "00101111",
196 => "00101110",
197 => "00101101",
198 => "00101101",
199 => "00101100",
200 => "00101011",
201 => "00101010",
202 => "00101010",
203 => "00101001",
204 => "00101000",
205 => "00100111",
206 => "00100111",
207 => "00100110",
208 => "00100101",
209 => "00100100",
210 => "00100100",
211 => "00100011",
212 => "00100010",
213 => "00100001",
214 => "00100001",
215 => "00100000",
216 => "00011111",
217 => "00011110",
218 => "00011110",
219 => "00011101",
220 => "00011100",
221 => "00011011",
222 => "00011011",
223 => "00011010",
224 => "00011001",
225 => "00011000",
226 => "00011000",
227 => "00010111",
228 => "00010110",
229 => "00010101",
230 => "00010100",
231 => "00010100",
232 => "00010011",
233 => "00010010",
234 => "00010001",
235 => "00010001",
236 => "00010000",
237 => "00001111",
238 => "00001110",
239 => "00001101",
240 => "00001101",
241 => "00001100",
242 => "00001011",
243 => "00001010",
244 => "00001010",
245 => "00001001",
246 => "00001000",
247 => "00000111",
248 => "00000110",
249 => "00000110",
250 => "00000101",
251 => "00000100",
252 => "00000011",
```

```
253 => "00000010",
254 => "00000010",
255 => "00000001",
256 => "00000000");
signal dtemp : unsigned(17 downto 0);                -- modulo accumulator buffer
signal dtemp1 : integer;
signal din buf : signed(17 downto 0);
constant  offset  :  unsigned(17  downto  0)  :=  -- offset = 1/16 <18,0,u>
"000100000000000000";
begin
process(CLK, RESET)
begin
 if (RESET='1') then
  dout <= (others => '0');
  din_buf <= (others => '0');
  dtemp <= (others => '0');
  dtemp1 <= 0;
 elsif rising edge(CLK) then
  din buf <= din(11)& din(11)& din(11)& din(11)&
din(11)& din(11)&din;                                -- sign extension
  dtemp <= dtemp + unsigned(din_buf) + offset;       -- accumulator
  dtemp1 <= to_integer(dtemp(17 downto 8));          -- mapping input to data values
  if (dtemp1 >= 0) and (dtemp1 < 257) then
   dout <= cosrom(dtemp1);                           Assume  i  is  output  data
  elsif (dtemp1 >= 257) and (dtemp1 < 513) then      accumulator,
   dout <= -cosrom(512-dtemp1);                      for 0 ≤ i ≤ 256 ➔ cosrom(i)
  elsif (dtemp1 >= 513) and (dtemp1 < 769) then      for 256 < i ≤ 512 ➔ -cosrom(512-i)
   dout <= -cosrom(dtemp1-512);                      for 512 < i ≤ 768 ➔ -cosrom(i-512)
  else                                               for 768 < i ≤ 1023 ➔ cosrom(1024-
   dout <= cosrom(1024-dtemp1);                      i)
  end if;
 end if;
end process;
END behavior;
```

6.4 FIR Filter

```
LIBRARY ieee;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.ALL;

entity FIR is
port(clock    : in std logic;                        -- declaration
     reset    : in std logic;
     data_in  : in signed(11 downto 0);              -- input 12 bit
     data_out : out std_logic_vector(11 downto 0)    -- ouput 12 bit
     );
end FIR;

architecture behavior of FIR is
signal d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,
       d11,d12,d13,d14,d15 : signed(15 downto 0);    -- 16 tap FIR
signal sum : signed(15 downto 0);                    -- buffer
begin
process(clock,reset)
begin
   if (reset = '1') then
        d0 <= (others => '0');
        d1 <= (others => '0');
        d2 <= (others => '0');
        d3 <= (others => '0');
        d4 <= (others => '0');
        d5 <= (others => '0');
        d6 <= (others => '0');
        d7 <= (others => '0');
        d8 <= (others => '0');
        d9 <= (others => '0');
        d10 <= (others => '0');
        d11 <= (others => '0');
```

```
        d12 <= (others => '0');
        d13 <= (others => '0');
        d14 <= (others => '0');
        d15 <= (others => '0');
        sum <= (others => '0');
        data out <= (others => '0');
   ELSIF rising edge(clock) THEN
        d0 <= data in(11)&data in(11)&
              data in(11)&data in(11)&data in;
        d1 <= d0;
        d2 <= d1;
        d3 <= d2;
        d4 <= d3;
        d5 <= d4;
        d6 <= d5;
        d7 <= d6;
        d8 <= d7;
        d9 <= d8;
        d10 <= d9;
        d11 <= d10;
        d12 <= d11;
        d13 <= d12;
        d14 <= d13;
        d15 <= d14;
        sum <= (d0+d1+d2+d3+d4+d5+d6+d7+d8+d9+
                d10+d11+d12+d13+d14+d15) srl 4;      -- 1/16 multiply is 4 bit right
 data_out <= std_logic_vector(sum(11 downto 0));    shift operation
 end if;
end process;
end behavior;
```

6.5 Circuit (top level design)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE IEEE.numeric std.ALL;

ENTITY circuit IS
PORT(clk   : IN std_logic;                          -- declaration
     reset : IN std_logic;
     fmin  : IN std logic vector(7 downto 0);       -- modulated data input
     dmout : OUT std logic vector (11 DOWNTO 0)     -- demodulated data output
   );
END circuit ;

ARCHITECTURE behavior OF circuit IS                 -- Architecture declarations
                                                    -- Internal signal declarations
   SIGNAL d1     : signed(11 DOWNTO 0);
   SIGNAL d2     : signed(11 DOWNTO 0);
   SIGNAL dout   : signed(7 DOWNTO 0);
   SIGNAL output : signed(7 DOWNTO 0);

   COMPONENT multiplier                             -- Component Declarations
PORT ( clk    : IN std logic ;
       reset  : IN std logic ;
       input1 : IN std logic vector (7 DOWNTO 0);
       input2 : IN signed (7 DOWNTO 0);
       output : OUT signed (7 DOWNTO 0)
     );
END COMPONENT;
COMPONENT fir
PORT ( clock    : IN std_logic ;
       reset    : IN std_logic ;
       data in  : IN signed (11 DOWNTO 0);
    data out : OUT std logic vector (11 DOWNTO 0)
   );
END COMPONENT;
COMPONENT loop_filter
PORT (clk   : IN     std_logic ;
```

```
      reset : IN     std logic ;
      c     : IN     signed (7 DOWNTO 0);
      d1    : OUT    signed (11 DOWNTO 0);
      d2    : OUT    signed (11 DOWNTO 0)
   );
END COMPONENT;
COMPONENT nco
PORT (clk   : IN     std logic ;
      reset : IN     std logic ;
      din   : IN     signed (11 DOWNTO 0);
      dout  : OUT    signed (7 DOWNTO 0)
   );
END COMPONENT;

BEGIN
      I1 : multiplier                                  -- Instance port mappings.
      PORT MAP (
           clk => clk,
         reset => reset,
         input1 => fmin,
         input2 => dout,
         output => output
      );
   I4 : fir
      PORT MAP (
         clock    => clk,
         reset    => reset,
         data_in  => d1,
         data out => dmout
      );
   I3 : loop filter
      PORT MAP (
         clk   => clk,
         reset => reset,
         c     => output,
         d1    => d1,
         d2    => d2
      );
   I2 : nco
      PORT MAP (
         clk   => clk,
         reset => reset,
         din   => d2,
         dout  => dout
      );
   END behavior;
```

6.6 Test Bench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE std.textio.ALL;

ENTITY circuit_tb IS
END circuit_tb;

ARCHITECTURE behavior OF circuit_tb IS

file vectors: text open read mode is "fm.txt";   -- input read from given file: "fm.txt" for square modulated signal and "fmtri.txt" for triangular modulated signal

COMPONENT circuit                                -- Component Declarations
PORT( clk : IN std_logic;
      reset : IN std logic;
      fmin : IN std logic vector(7 downto 0);
      dmout : OUT std logic vector(11 downto 0)
    );
END COMPONENT;
```

```
SIGNAL clk : std logic := '0' ;                          -- Internal signal declarations
SIGNAL reset : std logic := '1';
SIGNAL fmin : std logic vector(7 downto 0);
SIGNAL dmout : std_logic_vector(11 downto 0);
constant clkperiod : time := 62.5 ns;                    -- system clock frequency = 16 MHz

BEGIN
        uut: circuit PORT MAP(                           -- Instance port mappings.
                clk => clk,
                reset => reset,
                fmin => fmin,
                dmout => dmout
             );

    RESET_GEN: process
    begin                                                -- reset signal generator
        LOOP1: for N in 0 to 3 loop
           wait until falling edge(CLK);
        end loop LOOP1;
        RESET <= '0' ;
    end process RESET_GEN;

clk <= not clk after clkperiod / 2;

                                                         -- clock signal generator
process
variable vectorline : line;                              -- read file vector operation.
variable fmin_var : bit_vector(7 downto 0);
begin
while not endfile(vectors) loop
if (reset = '1') then
fmin <= (others => '0');
else
readline(vectors, vectorline);
read(vectorline, fmin_var);
fmin <= to stdlogicvector(fmin var);
end if;
wait for clkperiod;
end loop;
end process;

END;
```

7. Simulation Waveform

Fig. 11 shows the simulation waveform for all digital FM receiver circuit subjected to square wave modulated data, while Fig. 12 shows the simulation waveform for All Digital FM Receiver circuit subjected to triangular wave modulated data. The first row shows the FM modulated waveform according to the sending data. The second row is NCO output and the third row is phase detector (multiplier) output. The fourth row and the fifth row are the accumulator output and the demodulated output, respectively. At the initial simulation phase, the demodulated output overshoots since the phase synchronization is in convergence phase and then system is stable.

From Fig. 11 and Fig. 12, designed FM receiver circuit successfully demodulates input signal back to the original signal.

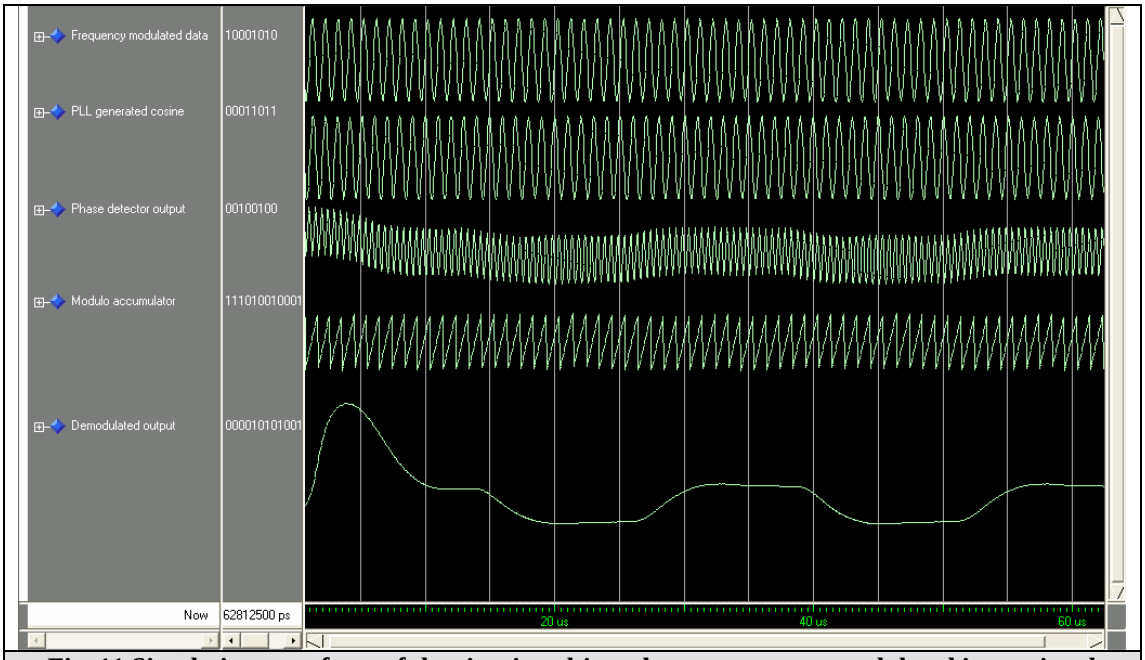

Fig. 11 Simulation waveform of the circuit, subjected to square wave modulated input signal

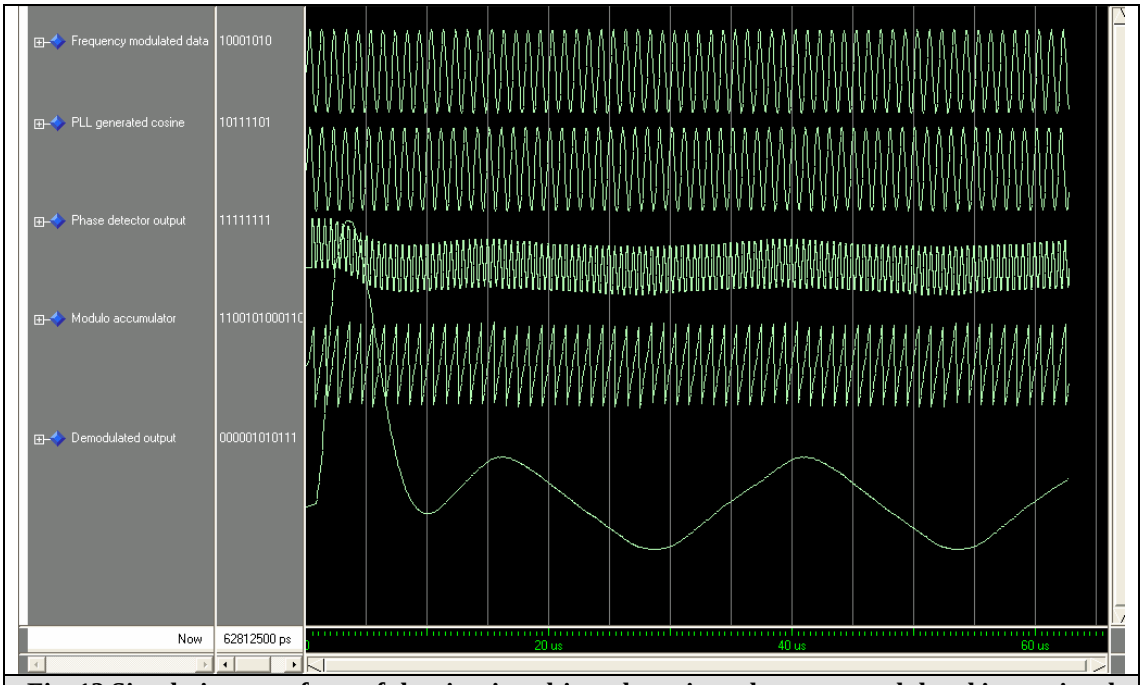

Fig. 12 Simulation waveform of the circuit, subjected to triangular wave modulated input signal

8. FPGA Implementation

We implement the all digital FM receiver circuit designed into FPGA. Here we are using Virtex2 device from Xilinx with XC2V2000 technology and ff896 package. The chip graphic is shown in Fig.13

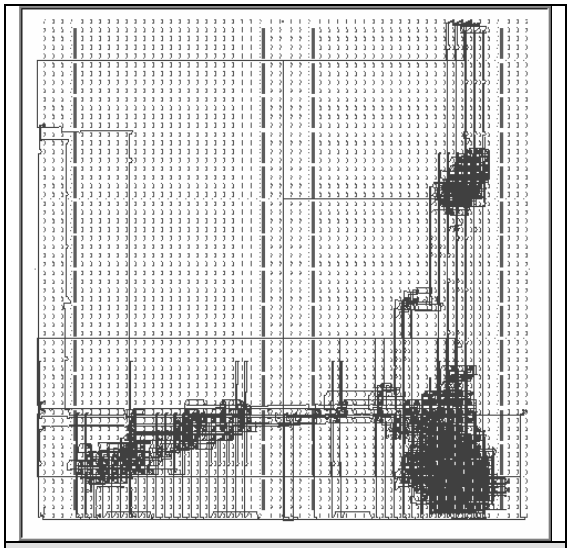

Fig. 13 Chip graphic for the design

ChipScope Pro 6.3i provides an integrated logic analyzer used to capture data in the designed circuit. After design is downloaded to FPGA board, ChipScope Pro will trigger input data and capture the output data via parallel cable in JTAG Boundary Scan mode as shown in Fig. 14. Captured data is in the listing form of 12-bit binary number as shown in Fig. 15. We can adjust how many samples needed to be captured; here we captured 1024 samples output data, then we plot it by ModelSim to obtain the actual demodulated signal view as shown in Fig.16

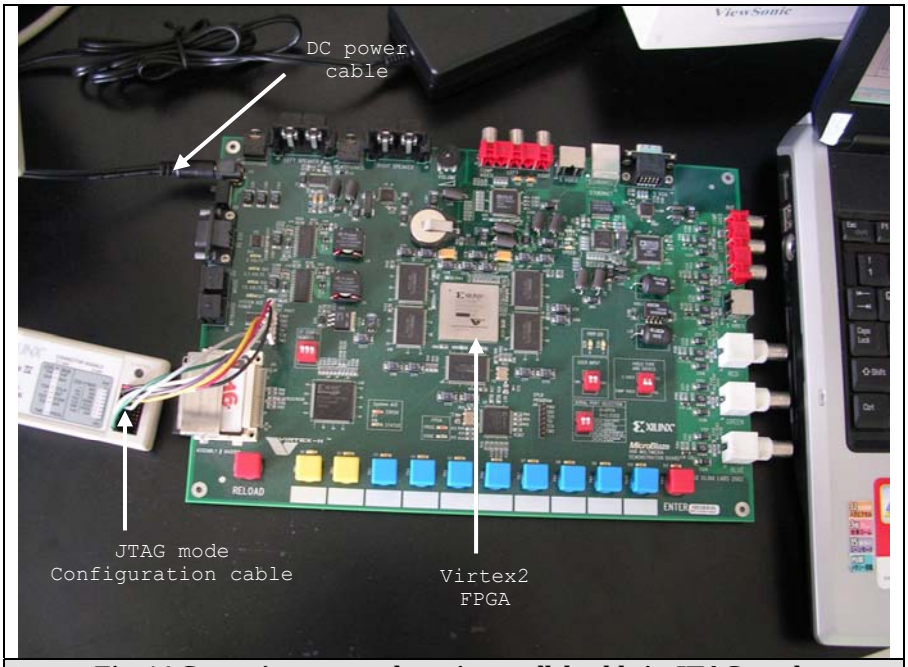

Fig. 14 Capturing output data via parallel cable in JTAG mode

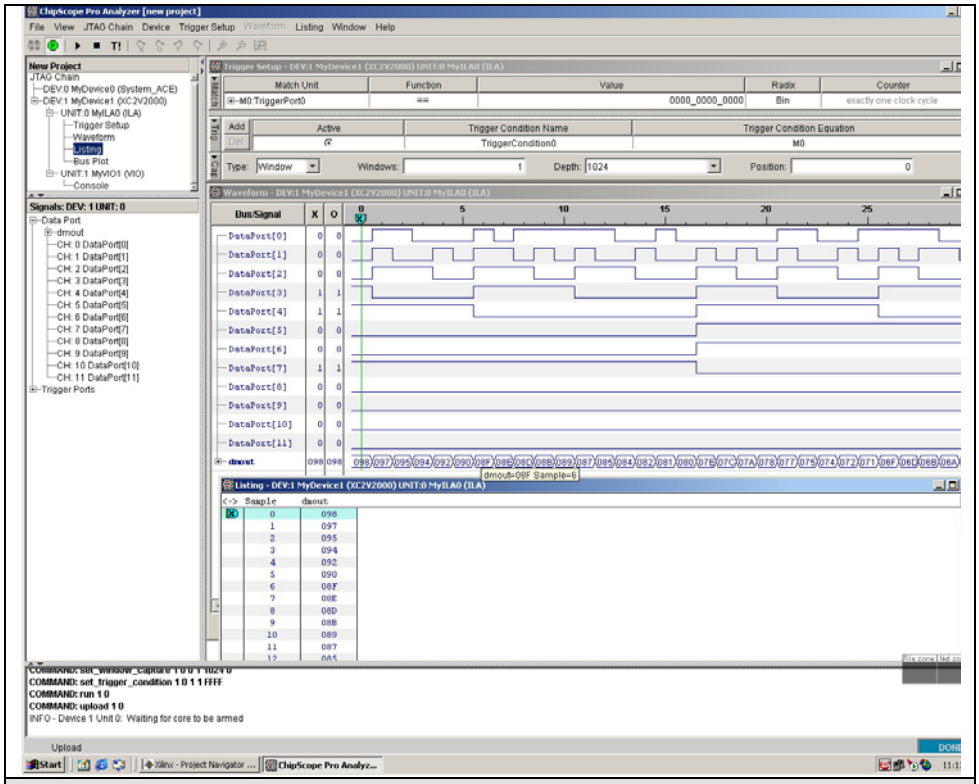

Fig. 15 Captured data and waveform of the output in ChipScope

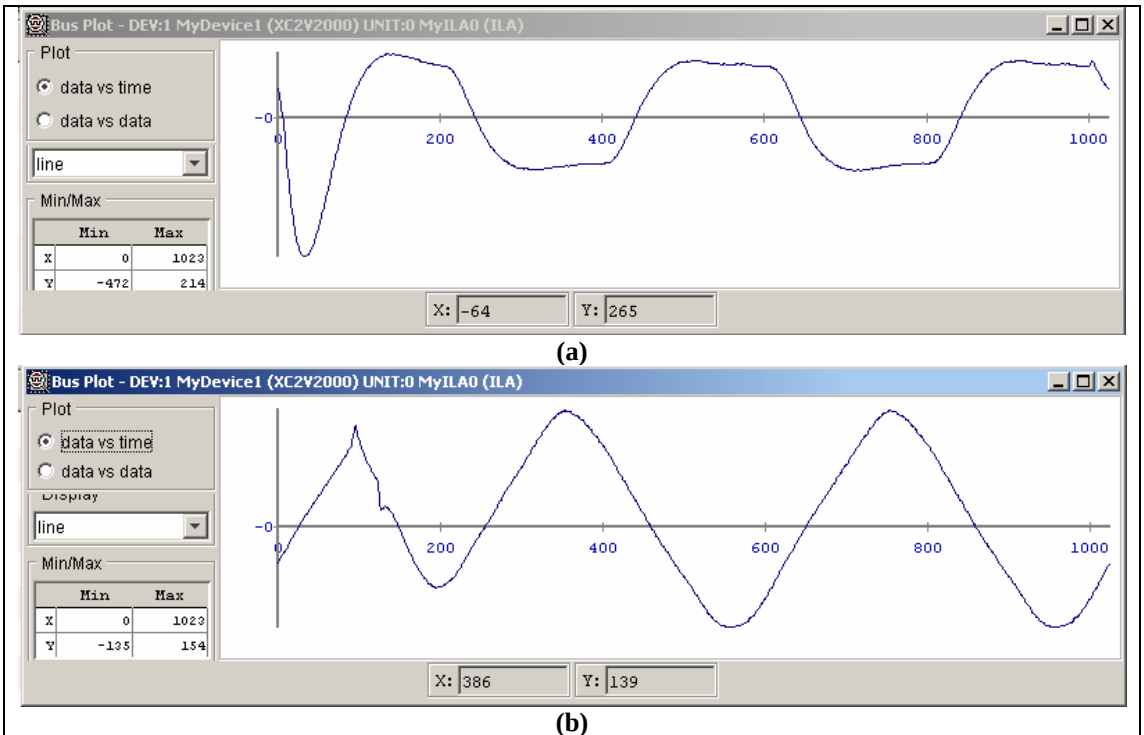

Fig. 16 Actual demodulated data

9. Closing

VHDL and FPGA are always attracting us in our VLSI System Design class. We are enthusiast in joining this program. This subject of study is new for us as beginners, now we can learn basic principle of digital FM receiver and get the opportunity to make our design, we enjoy it. We've found this a great subject to work in because we've gained knowledge about the state of the VLSI Design, its different sectors and the links that exist within it and between other global electronics study.

誰にもまちがいはある、だからエンピツにも消しゴムがついている。

Reference:

[1] "*Modeling PLL*," in Integrated Circuits Application Note AN178 Philips Semiconductors, 1988.
[2] Douglas J. Smith, "*HDL Chip Design*," Doone Publication, 1996
[3] John G. Proakis, Dimitri G. Manolakis, "*Digital Signal Processing*," Prentice Hall, 1996.
[4] Katsuhiko Ogata, "*Modern Control Engineering*," Prentice Hall, 2002.
[5] Naresh K. Sinha, "*Linear Systems*," John Wiley and Sons. Inc, 1991.
[6] P.E. Allen, "*All Digital Phase Locked Loop*," in Lecture Note CMOS Phase Locked Loops, 2003.
[7] Roland E. Best, "*Phase Locked Loop, Theory, Design, and Applications*," McGraw – Hill, 2003.
[8] Website : http://bw-www.ie.u-ryukyu.ac.jp/~wada/design05/spec_e.html

Appendix:

Synthesis result of 50 input XOR gate for computing unit delay

```
Timing Detail:
--------------
All values displayed in nanoseconds (ns)

--------------------------------------------------------------------------------
Timing constraint: Default path analysis
Delay:               6.967ns (Levels of Logic = 5)
  Source:            A<10> (PAD)
  Destination:       Y (PAD)

  Data Path: A<10> to Y
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             1   0.653   0.383  A_10_IBUF (A_10_IBUF)
     LUT4:I0->O            1   0.347   0.383  Mxor_Y_inst_lut4_01 (Mxor_Y__net0)
     LUT4:I0->O            1   0.347   0.383  Mxor_Y_inst_lut4_121 (Mxor_Y__net14)
     LUT4:I0->O            1   0.347   0.383  Mxor_Y_inst_lut4_151 (Y_OBUF)
     OBUF:I->O                 3.743          Y_OBUF (Y)
    ----------------------------------------
    Total                      6.967ns (5.437ns logic, 1.530ns route)
                                       (78.0% logic, 22.0% route)
```

Synthesis result of 50 input XOR gate for computing unit area

```
Design Summary
--------------
Number of errors:      0
Number of warnings:    0
Logic Utilization:
  Number of 4 input LUTs:              17 out of  21,504    1%
Logic Distribution:
  Number of occupied Slices:           13 out of  10,752    1%
  Number of Slices containing only related logic:      13 out of      13  100%
  Number of Slices containing unrelated logic:          0 out of      13    0%
        *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:             17 out of  21,504    1%

  Number of bonded IOBs:               51 out of     624    8%

Total equivalent gate count for design:  102
Additional JTAG gate count for IOBs:  2,448
Peak Memory Usage:  100 MB
```