A Top-Down Verilog-A Design on the Digital Phase-Locked Loop

Report of the Project Assignment

Presented for Ph.D Qualifying Exam

By

Ching-Hong Wang

Advisory Committee:

Steven Bibyk, Professor of the ECE Department, Advisor

Bradley D. Clymer, Professor of the ECE Department

Eylem Ekici, Professor of the ECE Department

Index

Appendix A. Verilog-A Behavior Models

Appendix B. Schematics and Block Diagrams

Appendix C. Simulation Results

Appendix D. Netlist Lists

Abstract

Modern high frequency, high performance system-on-chip design is heading to include more and more analog or mixed signal circuits as well as digital blocks. As the complexity of a system grows, it becomes more and more important to implement the system simulation and top-down design methodology as well. Verilog-A, which is studied in this report, is one of the most excellent top-down hardware description language specifically for analog and mixed signal designs. Its compatibility with pure digital hardware description languages (HDLs), such as Verilog and VHDL, is one of the most important advantages. In addition, the top-down characteristics make Verilog-A able to achieve system-level simulation that Matlab usually does. Even yet well developed nowadays, the potential capability for synthesis with digital HDLs is another unbeatable attraction. In this report, a digital phase-locked-loop of the magnetic hard-disk read channel is implemented and simulated by Verilog-A to experience its advantages.

Preview

While many types of signal processing have moved to the digital domain, analog and mixed signal circuits have been proven fundamentally necessary in today's high performance systems.[1] Unfortunately, modern analog and mixed signal processing is characteristically more complicated than that of the purely digital designs. Due to this reason, it turns to be necessary to have system-level simulation in advance for the reasons of performance evaluations and conceptual designs. System-level simulation can also help designers evaluate the entire architecture prior to heading into a detail circuit design.

Based on the three advantages described at the beginning of this report, compatibility with pure HDLs, capability of system level simulation and potentials for synthesis, we select Verilog-A to evaluate a digital phase-locked-loop circuit which is often used in the magnetic hard-disk read channel path.

Table.1 briefly summarizes the pros and cons of three different design methodologies existing in the hardware design world. Comparing with analog SPICE and digital HDLs, there is apparently a gap between digital design and analog design. Digital HDLs can not conduct with differential equations, frequencies, s-domain and z-domain functions that are strengths of analog SPICE. However, it is difficult to have conditional and behavioral descriptions and simulations while executing SPICE to simulate a big system. Verilog-A is like a combination between analog and digital design methods. It has capabilities running conditional simulations, behavioral simulations as well as conducting with differential equations, frequencies, s-domain and z-domain functions.

Seeking anticipation to future synthesizable capabilities and system-level simulations, Verilog-A becomes our main topic for the mixed-signal hardware design method on the digital phase locked loop project.

	Design Methodology Comparison		
	Mixed Signal Design Methodology		**Digital Design Methodology**
	Schematic Capture/SPICE (Button-Up process)	**AHDLs (Verilog-A) (Top-Down process)**	**HDLs (VHDL, Verilog) (Top-Down process)**
Pros	--	Fast	Fast
	--	Easy debug (Automatically)	Easy debug (Automatically)
	--	--	Synthesizable
	--	Support analog and digital models	--
	--	Integration with HDLs (digital portion only)	Integration with AHDLs (digital portion only)
	--	Model library separated from simulator	Model library separated from simulator
	--	Support behavior modeling	Support behavior modeling
	--	Ability for entire system-level simulation	Limit to digital system-level simulation
	Most accurate performance simulation	Conditional and mathematical simulation (s-domain, z-domain, differential equation …,etc)	--
	Model interchange widely supported	--	Model interchange widely supported
Potentials	--	Potentials for synthesizable capability	--
	--	Potentials for exchange of HDLs models with analog functions	--
Cons	Slow	--	--
	Difficult debug (Manually, only designers know the accuracy)	--	--
	None synthesis	--	--
	For analog model	--	For digital model
	Worse integration with HDLs	--	--
	Model library inherited with simulator	--	--
	Structural modeling only	--	--
	Unable system-level simulation	--	--
	Mathematical simulation but in transistor-level only	--	Conditional simulation. (No mathematical simulation)
	--	Yet widely model interchange supported	--

Table.1 Comparison of hardware design methodologies

The digital phase-locked-loop block diagram of a magnetic hard-disk read channel shown on Fig.1 is referred to the paper presented by Toshio Murayama in 1996 [2].

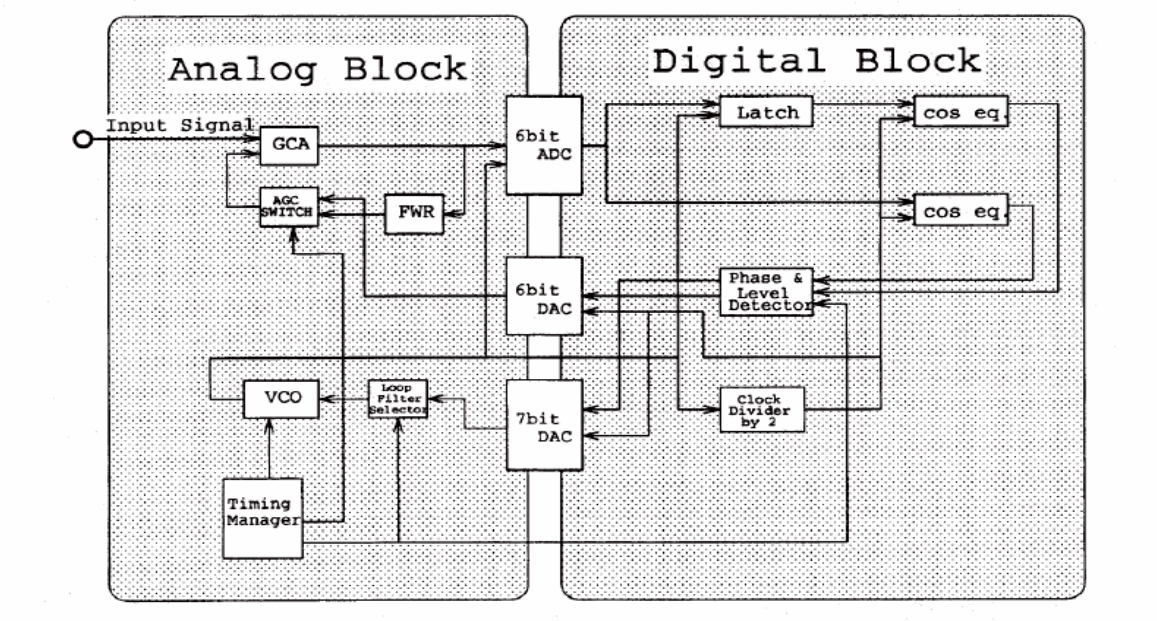


Fig.1 Magnetic hard-disk read channel diagram

By means of the Verilog-A hardware description language, the behavior models of those blocks that construct a digital phase-locked loop are coded. Using these modeled blocks could we successfully compose an entire digital phase-locked-loop in our Cadence environment shown on Fig.2.

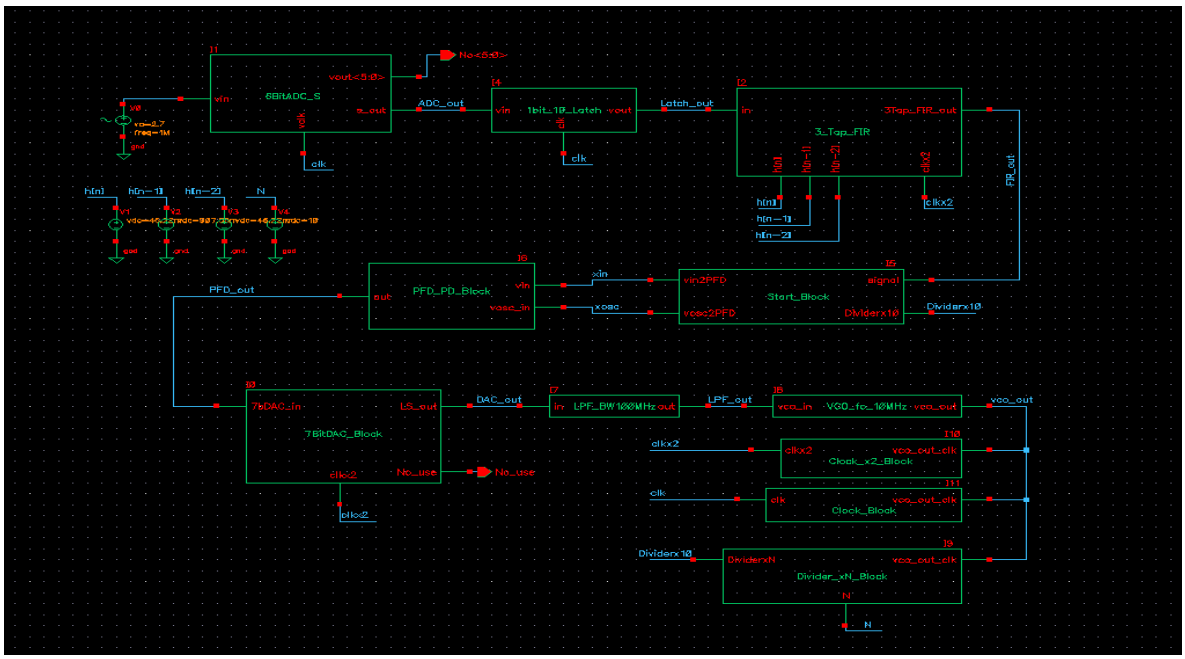


Fig.2 Analog-and-digital PLL design in Cadence environment

Chapter 1

Introduction

1-1. Design Objective

The digital phase-locked-loop circuit that we are going to study and implement is following the wider solid line path shown on Fig.3. The rests are used for the magnitude detection and locked-loop. The magnitude detection and locked-loop consists of the GCA (Gain Controlled Amplifier), FWR (Full Wave Rectifier), AGC (Automatic Gain Control), 6-bit ADC, latch, cosine equalizer, level (and phase) detector and 6-bit DAC. This loop is however neither implemented nor covered in this report.

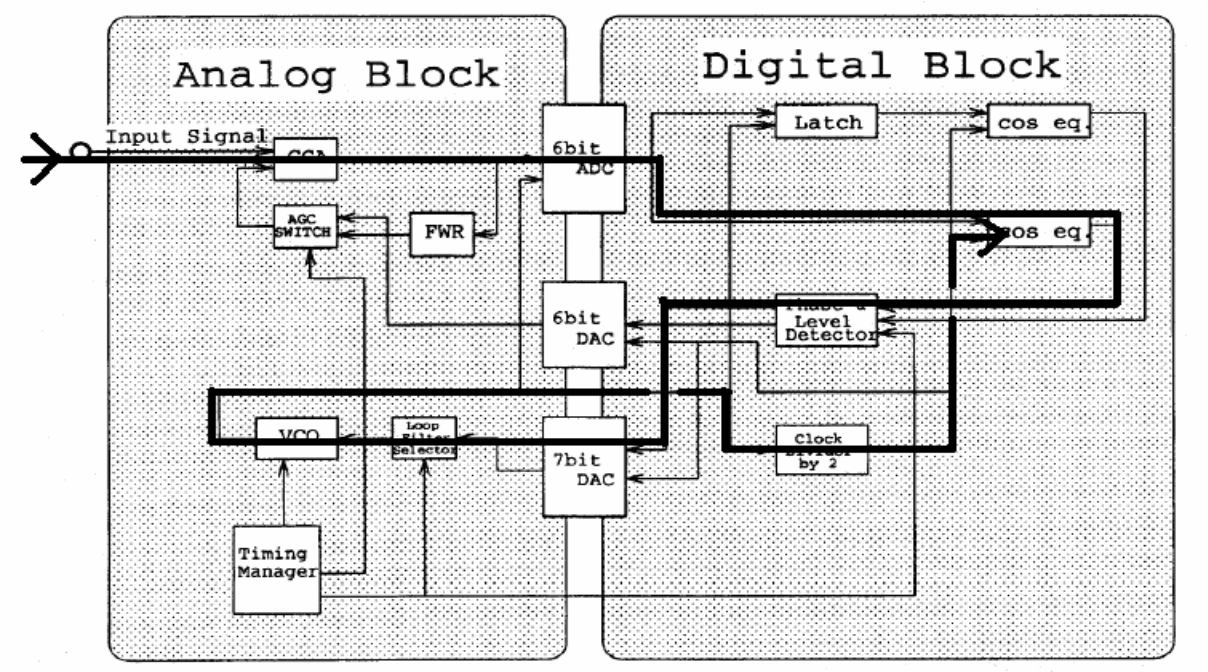


Fig.3 Analog-and-digital PLL signal path

Along the phase-locked signal path, there are several essential components that build up the entire PLL in which we are interested. They are 6-bit ADC,

cosine equalizer, phase (and level) detector, 7-bit DAC, loop filter, VCO and clock divider. It is very similar to a conventional phase-locked loop but replace the charge pump block with a combination of the digital counter and 7-bit DAC. The counter is embedded into the digital phase detector and works as the input level for the 7-bit DAC. As the output of phase detector sends a plus or minus signal to the counter, it will count up or down and represents the digitized level of an input for DAC. Following sections will basically introduce the traditional phase-locked loop circuit blocks.

Within the digital block there is a cosine equalizer that behaves as a digital filter and is also called as a 3-Tap finite impulse response (3-Tap FIR) filter. The cosine equalizer, digital phase detector and 7-bit DAC compose the function same as that of the conventional phase frequency detector plus charge pump. We can refer to the simulation result on Appendix C-1 that shows the 3-Tap FIR filter function on a noisy input signal. The 3-Tap FIR filter consists of one adder, two delay blocks and three multipliers. Appendix A-1 is the Verilog-A code for the adder. The delay block is modeled as the Appendix A-2 and its simulation result is on Appendix C-2. The Verilog-A model for the multiplier is on Appendix A-3.

1-2. Conventional Phase-Locked loop

Depends on the different purposes, the phase-locked loop (PLL) most often deals with signals or clocks to reduce timing jitters, suppresses clock skews, synthesizes higher frequencies, and assists data and clock recoveries. Traditionally there are four kinds of phase-locked loops. They are linear PLL (LPLL), digital PLL (DPLL), all-digital PLL (ADPLL), and software PLL (SPLL) all around the world. However, the phase-locked-loop circuit is conventionally based on three essential components.[3]

1. Voltage-Controlled-Oscillator (VCO)
2. Loop Filter (LF)
3. Phase Detector (PD)

The phase detector first compares the differences between the input signal and the VCO output signal. It then generates a phase error according to the difference. Passing through the loop filter, the feedback loop will present a control signal for the voltage-controlled oscillator to either increase or decrease the oscillating frequency in accordance to the controlled voltage level. Recursively, the entire phase-locked loop will be able to lock the input signal within a lock-in time. Fig.4 shows the conventional phase-locked-loop block diagrams as well as its transfer functions.

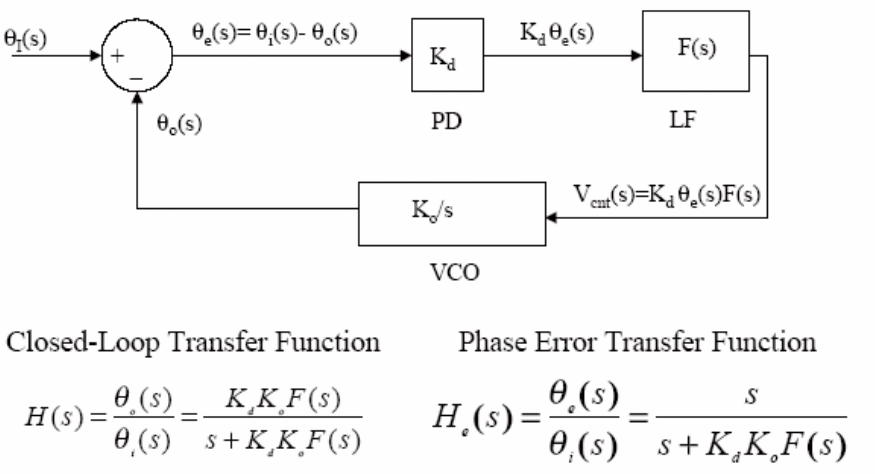


Fig.4 Block diagram and transfer function of a PLL

1-2-1. Voltage-Controlled-Oscillator (VCO)

As its name described, a voltage-controlled oscillator has the feature that its oscillating frequency on the output port is under controlled by the input control voltage, Vcnt. The block diagram and its predicted waveform for ideal simulation are shown on Fig.5 and Fig.6 respectively.

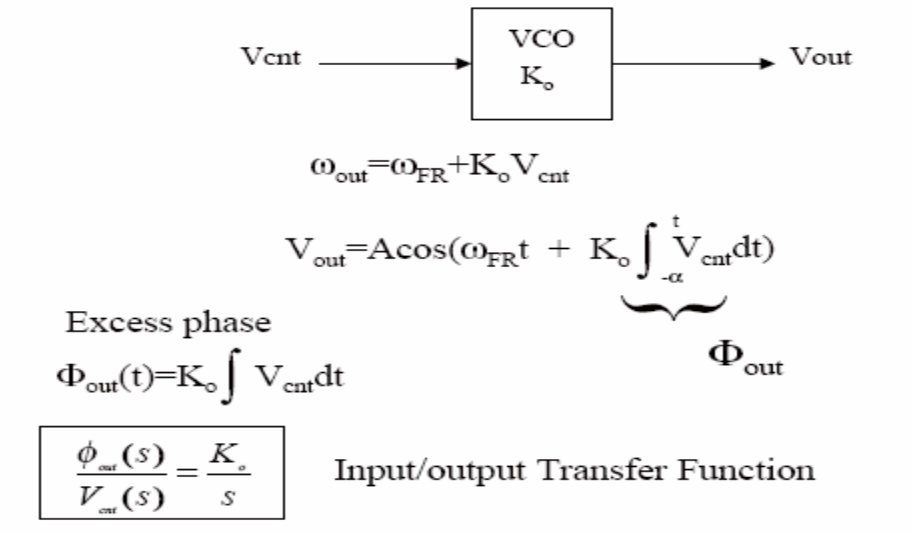


Fig.5 Block and the transfer function of a VCO

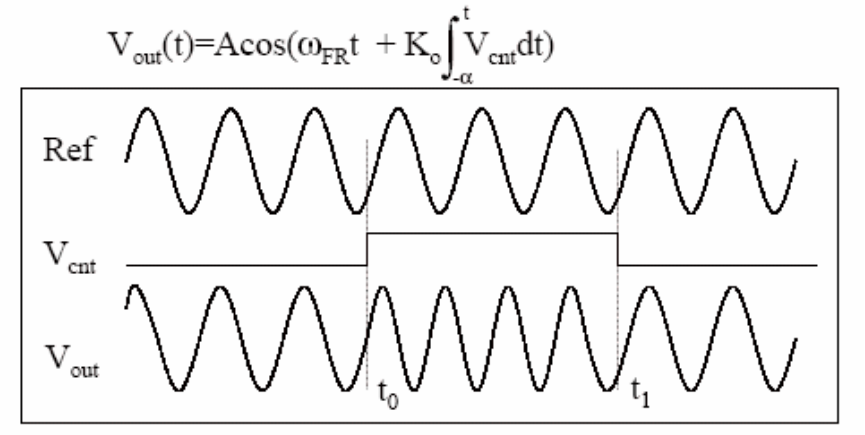


Fig.6 Frequency response as V_{cnt} changes

1-2-2. Charge-Pump Loop Filter

The basic charge-pump loop filter consists of a charge-up as well as a discharge path accompanying with a capacitor to configure the low pass filtering function. It is controlled by the output of the phase detector and either charges or discharges to a voltage level to control the oscillating frequency of VCO. As shown on Fig.7, the switch S1 works for the charging path and the switch S2 is for discharging use.

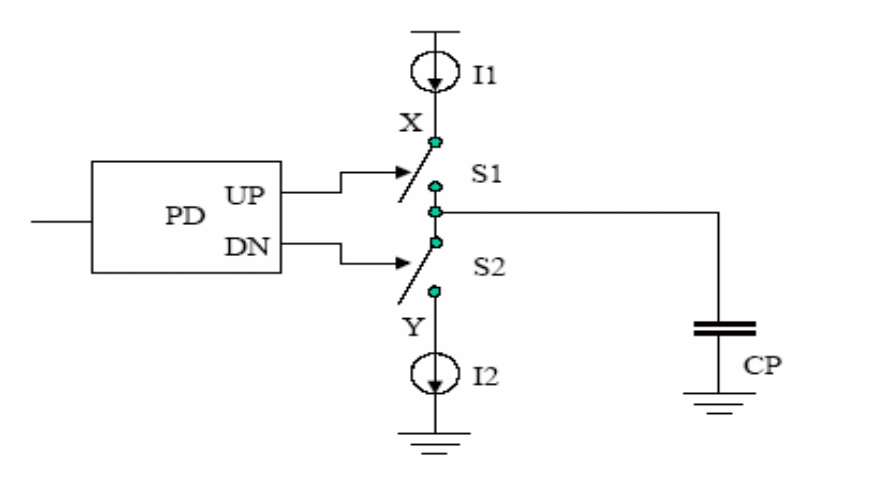


Fig.7 A charge-pump loop filter

1-2-3. Phase Detector

Traditionally, there are three most important phase detectors usually mentioned on the papers. They are listed below:

1. Exclusive XOR Gate
2. J.K. Flip-Flop
3. Phase-Frequency Detector (PFD)

Straightforwardly as their names described, the exclusive XOR gate and JK flip flop are very simple phase detectors and easy to design. However, the phase frequency detector (PFD) is becoming more and more popular because it differs greatly from the other two types of phase detectors. As its name implies, its output signal depends not only on a phase error but also on a frequency error, denoted as $\Delta \omega = \omega_1 - \omega_2$, when the PLL has not yet been locked.[03]

The conventional phase-frequency detector has its block diagram shown as Fig.8.

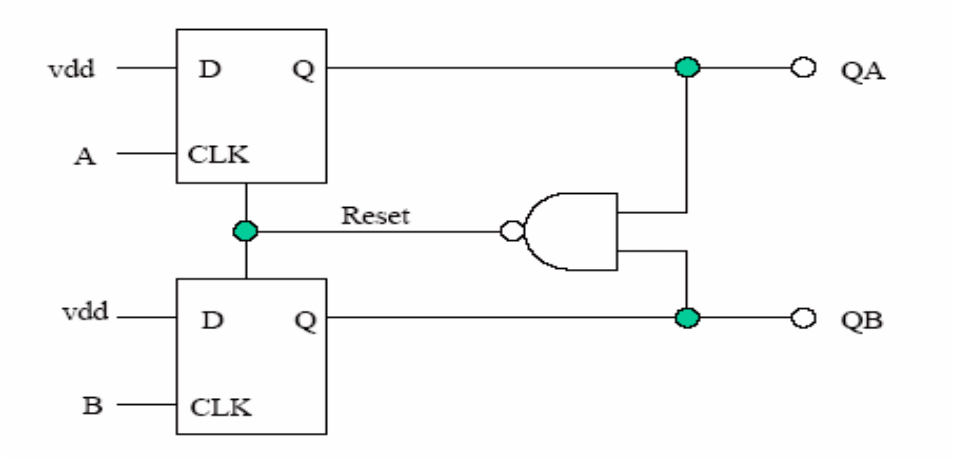


Fig.8 Block diagram of a phase-frequency detector

As the two input signals (A,B) change their states, the corresponding outputs (Q_A,Q_B) will present information telling the next charge-pump block to charge up or discharge down. The state diagram of this phase-frequency detector is shown on Fig.9.

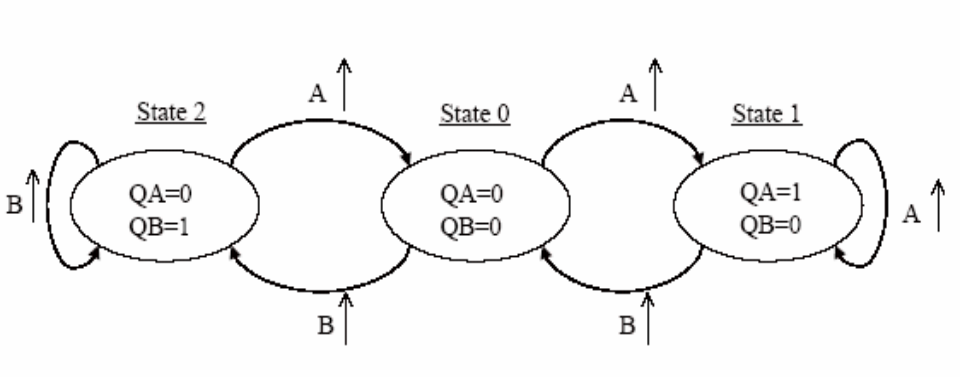


Fig.9 State diagram of the phase-frequency detector

The State0 represents that both Q_A and Q_B are 0 and denotes as a “same” state. State1 means the input signal (A) is leading the VCO output signal (B). State2 is opposite against state1 that means signal A lags to signal B. On the events that either input signal or VCO output signal has rising edge, the PFD will detect their difference in both phase and frequency domain and as well reflects the proper state on the PFD outputs

Chapter 2

Matlab System-Level Design and Simulation

As mentioned previously, the modern complicated analog and mixed signal chip designs require system level design and simulation prior to getting into detail circuit designs. This process can assist designers evaluating their design architectures before really struggling on some unsatisfying specifications.

Matlab, a well known powerful numerical simulation tool, is one of the good choices to do system level design and simulation. There are two approaches to implement the system level tasks. First, designers can use the embedded M-file functions or model their preferred M-files to do the system simulation. The second approach is the use of simulink, an embedded block diagram method of Matlab, to accomplish those system level design and simulation. Although Matlab is a nice simulation tool for our system level design, the emphasis is not on it in this report. The reason that Matlab is not a major design technique instead in this report is its poor compatibility with pure hardware description languages. We will only go through the matlab design methodologies as section 2-1 and section 2-2.

2-1. M-file Block Design and Simulation

In this section, we are going to experience a practical system-level behavior modeling through the Matlab M-file code. The example in section 2-1-1 presents the M-file that we can use to model a voltage controlled oscillator.

2-1-1. Voltage-Controlled-Oscillator

The M-file described on Fig.10 is a simple example to simulate a voltage-controlled oscillator. It comes out the simulation result on Fig.11. Here we use an embedded Matlab function, vco(), to model our required signal. Such vco.m file is located under the signal processing toolbox. Fig.12 presents us in details what the embedded function vco() should be.

```
% Matlab M-file, VCO Design
% Designer: CHING-HONG WANG
Fs = 100;
t = 0:1/Fs:2;
x = sin(2*pi*t);
y = vco(x,10,Fs);
subplot(2,1,1),plot(t,x,':')
title('Control Signal')
xlabel('Time    (sec)'), ylabel('Voltage (V)')
subplot(2,1,2),plot(t,y)
title('VCO Output Response')
xlabel('Time    (sec)'), ylabel('Voltage (V)')
```

Fig.10 M-file for a voltage-controlled oscillator

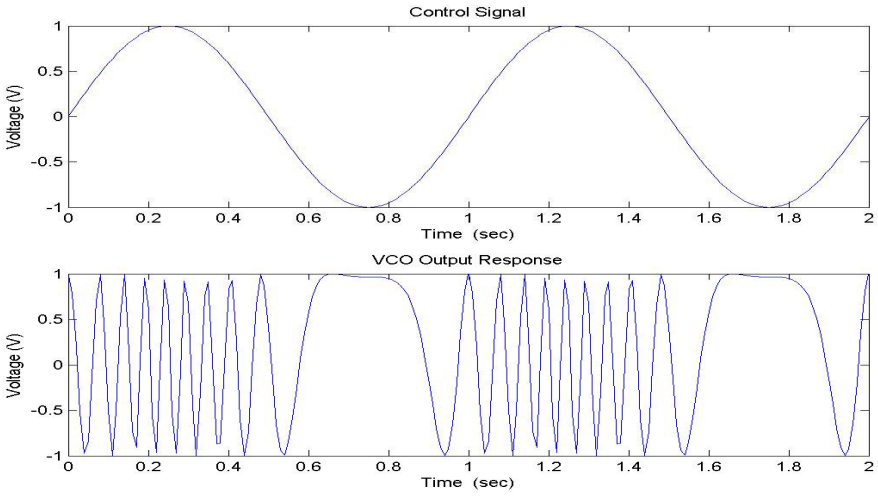


Fig.11 Simulation results of a VCO designed in M-file

```
function y = vco(x,range,Fs)
%VCO    Voltage controlled oscillator

if nargin<3
      Fs = 1;
end
if nargin<2
      Fc = Fs/4;
      range = Fc;
end
x_max = max(max(x));
x_min = min(min(x));
if (x_max>1)|(x_min<-1)
      error('      X outside of range [-1,1]')
end
if length(range)>1
      Fc = mean(range);
      range = (range(2) - Fc)/Fs*2*pi;
else
      Fc = range;
      range = (Fc/Fs)*2*pi;
end

y = modulate(x,Fc,Fs,'fm',range);
```

Fig.12 The embedded vco.m function model

2-2. SimuLink Block diagram and Simulation

A simulink block diagram design method is demonstrated on section 2-2-1. It is designed for a phase-frequency detector.

2-2-1. Phase-Frequency Detector

By means of the embedded block diagrams, we can also have our system level designs in the Matlab simulink. Fig13 is a typical block diagram design and within the solid line is a conventional phase frequency detector. The simulation result is shown on Appendix C-4.

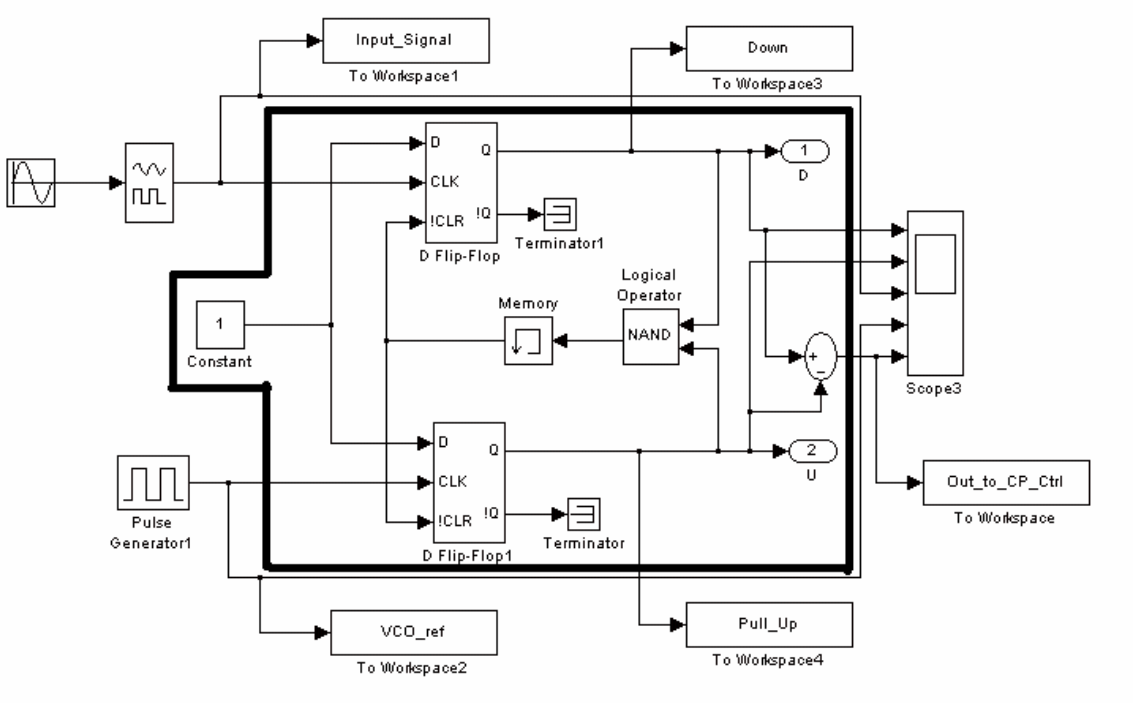


Fig.13 A phase-frequency detector implemented by Simulink

Chapter 3

Verilog-A System-Level Design and Simulation

Verilog-A, an advanced analog and mixed signal hardware description language that we emphasized at the beginning, plays the most important role for our digital phase-locked loop design in this report. The compatibility with pure HDLs as well as its system simulation ability catches our whole attentions. In this chapter, there are two design examples demonstrating its functions. Section 3-1 is an example of designing the voltage controlled oscillator in Verilog-A. The practical design of a phase-frequency detector in the digital block presented on the paper [2] is studied and implemented in Verilog-A as well in section 3-2. All the other portions not included in this chapter are covered inside the appendices.

3-1. VCO Design in Verilog-A

In this section, we demonstrate a voltage controlled oscillator modeled by the Verilog-A HDL. The detail code is shown on Fig.14. A central frequency is predefined as 10MHz when no control voltage applies. The phase of its sinusoidal output signal is varying and decided by the controlled voltage level. As the control voltage level increases, the frequency of its output signal also increases proportionally. Refering to Fig.15, it presents to us that a higher voltage level of LPF_out produces a higher oscillating frequency, VCO_out. This is corresponding to the solid rectangular region selected on Fig.16 and Fig.17. Appendix A-4 shows the Verilog-A code for a low-pass filter (LPF). The simulation result of a LPF is on Appendix C-5.

```
// VerilogA Design, VCO, veriloga
// Designer : CHING-HONG WANG
`include "constants.h"
`include "discipline.h"
`define PI    3.14159265358979328468643383279502884197 1
module VCO(vin,vout);
    input       vin;
    output      vout;
    electrical    vin, vout;
    parameter real vout_center_level = 0;
    parameter real vout_amp=5;
    parameter real center_freq=10000000;    //when vin @DC
    parameter real Hz_volt_gain=8000000;
    real Wc;                                    //actual center frequency in rad/s
    real phi;                                   //phi=Wc*t
    real delt_phi;
    real inst_freq;
    integer CycleCount;
analog begin
        @(initial_step) begin
           Wc=2*`PI*center_freq;
        end
        phi=Wc*$abstime;
        CycleCount=phi/(2*`PI);
        phi=phi-(CycleCount*2*`PI);
        delt_phi=2*`PI*idt(V(vin),0)*Hz_volt_gain;
        V(vout)    <+    (vout_center_level + (vout_amp/2)
                                              *sin(phi+delt_phi));
        inst_freq=center_freq+V(vin)*Hz_volt_gain;    //update    frequency

        $bound_step(0.04/inst_freq);
end
endmodule
```

Fig.14 Verilog-A model of a voltage-controlled-oscillator

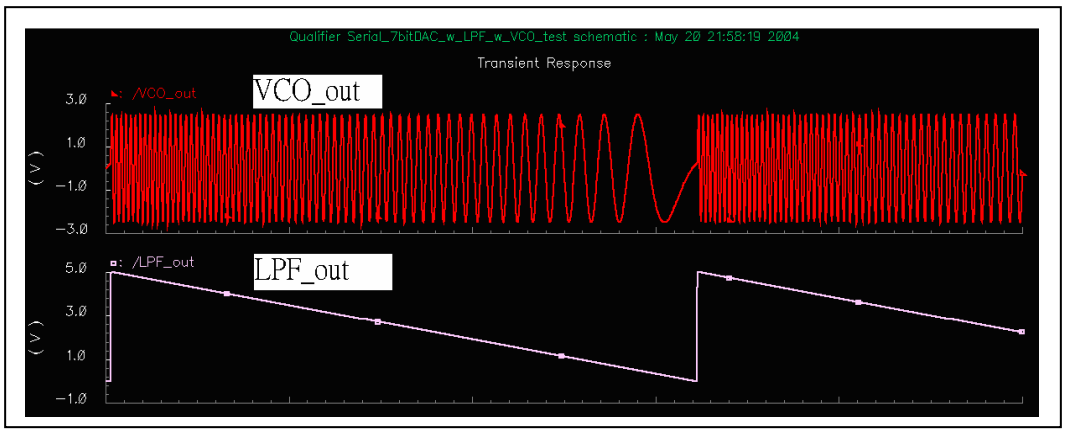


Fig.15 Waveform of the VCO output and the controlling LPF_out

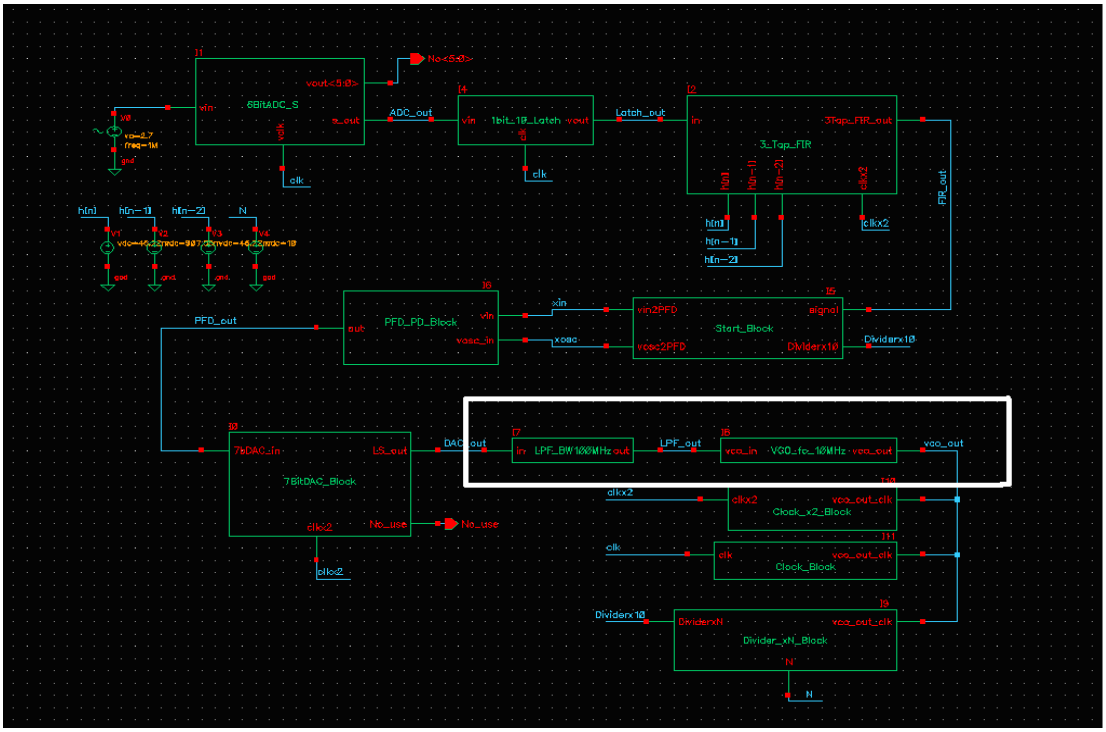


Fig.16 The VCO output and input (control) path

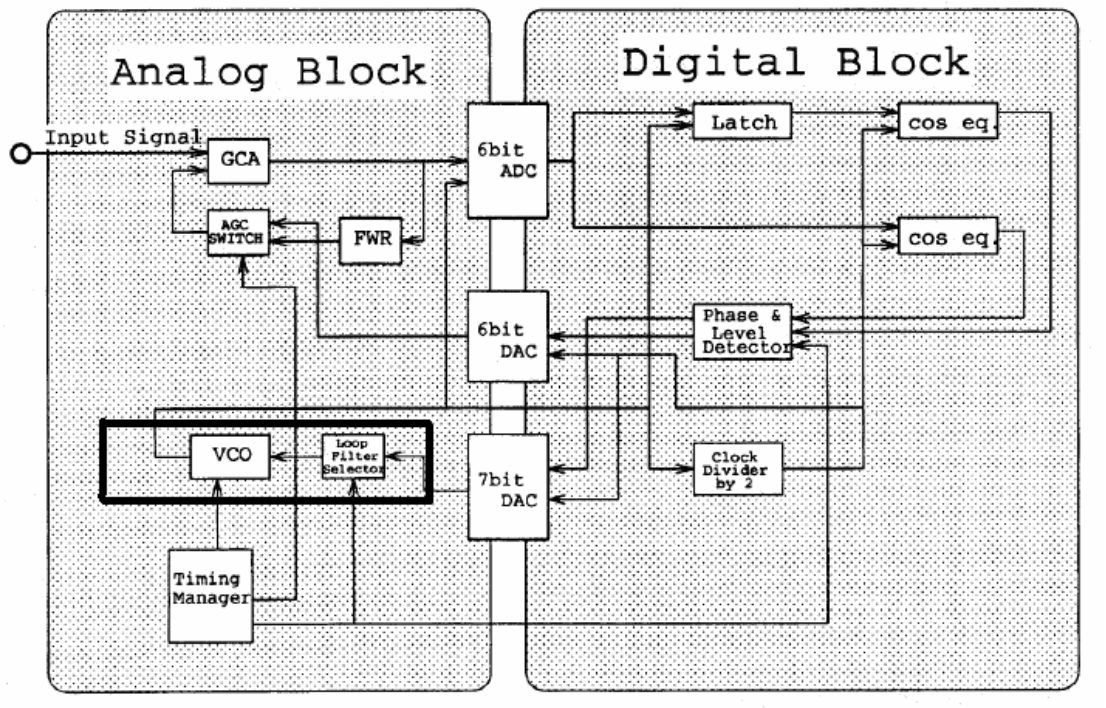


Fig.17 VCO functions in the block diagram of SONY's paper

3-2. Practical Design of PFD in Digital Block

The phase frequency detector presented on SONY's paper [02] is entirely combined by several digital blocks. It includes a cosine equalizer, a phase detector and an embedded 7-bit counter. Since the cosine equalizer is the same as a 3-Tap FIR filter, we can easily recognize its function and simplify the PFD as having approximately the phase detector plus a counter. The phase detector grabs the filtered output signal from the cosine equalizer and compares it with the output of the clock divider, which divides the frequency from VCO by an integer 2. Then the phase detector decides either the following charge-up or discharge process for the 7-bit counter. If the counter receives a charge-up signal, it is going to count up one bit. On the other hand, it will count down one bit once it

receives a discharge command. Fig.18 presents the Verilog-A code for a conventional phase-frequency detector.

```
// VerilogA for Exam, PFDUpTrigger, veriloga
// Designer: CHING-HONG WANG

`include "constants.h"
`include "discipline.h"
`define behind 0
`define same 1
`define ahead 2
//'ahead'    =>    'plus_out' = 1, 'minus_out' = 0; ( vin is leading vosc, so
plus-out=1 to speedup vosc's freq )
//'same'      =>    'plus_out' = 0, 'minus_out' = 0;
//'behind' =>    'plus_out' = 0, 'minus_out' = 1; ( vin is lagging vosc, so
minus-out=1 to slow down vosc's freq)
module PFDUpTrigger(vin, vosc, plus_out, minus_out);
    input vin, vosc;
    output plus_out, minus_out;
    electrical vin, vosc, plus_out, minus_out;
    parameter real vlogic_high = 5;
    parameter real vlogic_low    = 0;
    parameter real vlogic_mid    = 2.5;
    parameter real vtrans = 2.5;
    parameter real tdel = 0 from [0:inf);
    parameter real trise = 1n from (0:inf);
    parameter real tfall = 1n from (0:inf);
    integer vin_up, vosc_up, vin_down, vosc_down;    // flags
    real plusctrl, minusctrl;
    real in_value, osc_value;
    integer state;
```

Fig.18 Verilog-A of a conventional PFD (part A)

```
analog begin
        @(initial_step) begin
            in_value = V(vin);
            osc_value =V(vosc);
        end
        @ ( cross(V(vin) - vtrans, +1) ) begin
            vin_up    = 1;
            in_value = 5;
            if ( osc_value <= 2.5 )
                    osc_value = 0;
            else osc_value =5;
            if ( in_value == osc_value ) begin
                state = `same;
            end
            else begin
                    if ((in_value ==5 )    && (osc_value==0)) begin
                            state = `ahead;
                    end
                    if ((osc_value == 5) && (in_value == 5)) begin
                            state = `same;
                    end
            end
        end
        @ ( cross(V(vosc) - vtrans, +1) ) begin
            vosc_up = 1;
            osc_value =5;
            if ( in_value <= 2.5)
                    in_value = 0;
            else in_value =5;
            if (in_value == osc_value) begin
                state = `same;
            end
```

Fig.18 Verilog-A of a conventional PFD (part B)

```
        else begin
                if ((osc_value ==5) && (in_value ==0)) begin
                        state = `behind;
                end
                if ((in_value ==5) && (osc_value==5)) begin
                        state = `same;
                end
        end
end
@ ( cross(V(vin) - vtrans, -1) ) begin
        vin_up    = 0;
        in_value =0;
        if (osc_value >= 2.5)
            osc_value =5;
        else osc_value =0;
        if ( in_value == osc_value) begin
            state = `same;
        end
        else begin
                if ((in_value ==0) && (osc_value ==5)) begin
                        state = `ahead;
                end
                if ((osc_value =0) && (in_value =0)) begin
                        state = `same;
                end
        end
end
```

Fig.18 Verilog-A of a conventional PFD (part C)

```
@ ( cross(V(vosc) - vtrans, -1) ) begin
            vosc_up = 0;
            osc_value =0;
            if (in_value >= 2.5)
                  in_value =5;
            else in_value=0;
            if (in_value == osc_value) begin
                 state = `same;
            end
            else begin
                  if ( (osc_value==0) && (in_value==5)) begin
                              state = `behind;
                  end
                  if (( osc_value = 0) && (in_value =0)) begin
                              state = `same;
                  end
            end
      end
      if ((vin_up==1) && (state==`ahead)) begin
            plusctrl =vlogic_high;
            minusctrl=vlogic_low;
      end
      else begin
           if ((vin_up ==1) && (state==`same)) begin
                  plusctrl =vlogic_low;
                  minusctrl=vlogic_low;
           end
      end
      if ((vin_up ==0) && (state ==`ahead)) begin
            plusctrl =vlogic_low;
            minusctrl=vlogic_low;
      end
```

Fig.18 Verilog-A of a conventional PFD (part D)

```
        else begin
            if ((vin_up ==0) && (state==`same)) begin
                    plusctrl =vlogic_low;
                    minusctrl=vlogic_low;
            end
        end
        if ((vosc_up==1) && (state ==`behind)) begin
                plusctrl =vlogic_low;
                minusctrl=vlogic_high;
        end
        else begin
            if ((vosc_up==1) && (state==`same)) begin
                    plusctrl =vlogic_low;
                    minusctrl=vlogic_low;
            end
        end
        if ((vosc_up ==0) && (state ==`behind)) begin
                plusctrl =vlogic_low;
                minusctrl=vlogic_low;
        end
        else begin
             if ((vosc_up ==0) && (state==`same)) begin
                    plusctrl =vlogic_low;
                    minusctrl=vlogic_low;
             end
        end
        V(plus_out)    <+ transition(plusctrl, tdel, trise, tfall);
        V(minus_out) <+ transition(minusctrl, tdel, trise, tfall);
    end
endmodule
```

Fig.18 Verilog-A of a conventional PFD (part E)

As previous description, there is an embedded 7-bit counter along with the conventional phase detector to build up the control signal for the 7-bit digital-to-analog converter (7-bit DAC), which behaves as the charge-pump block in conventional phase-locked loop circuit. The block diagram of practical phase detector can be easily understood as shown on Fig.19. This is also referred to the solid selected region on Fig.20 and Fig.21 as well.

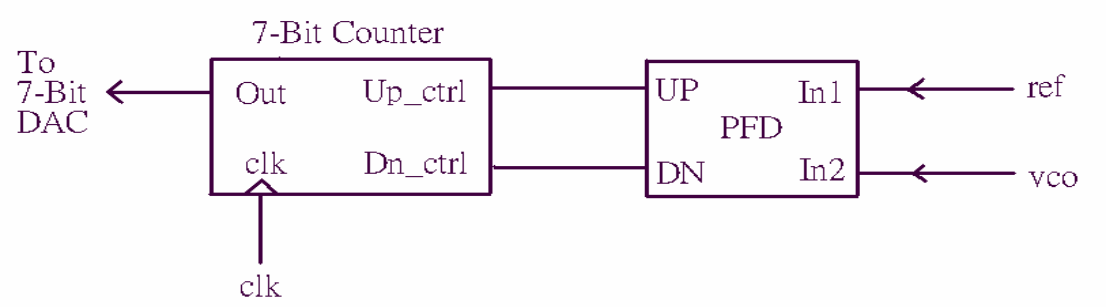


Fig.19 practical digital phase detector design (includes counter)

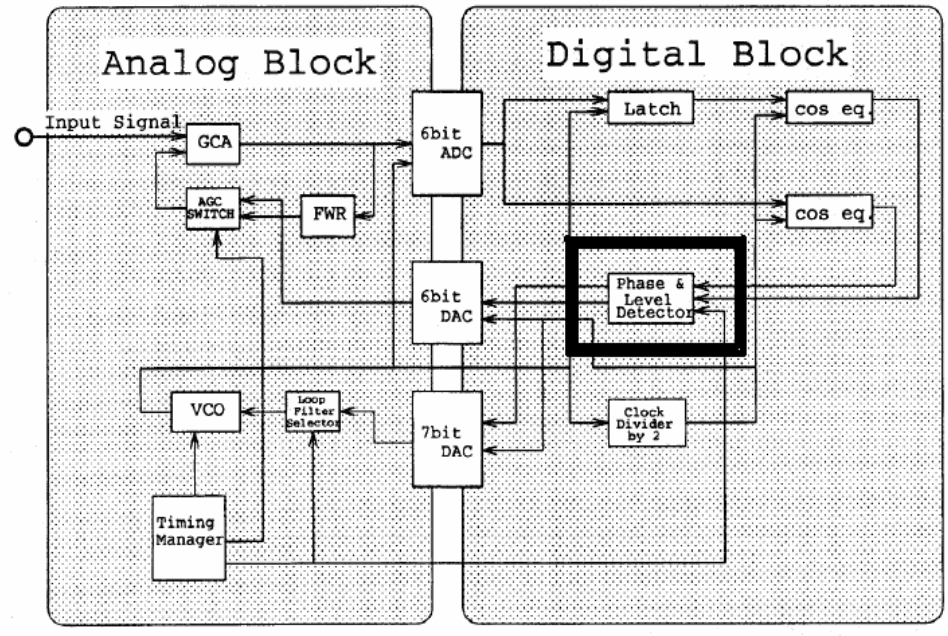


Fig.20 Phase detector block

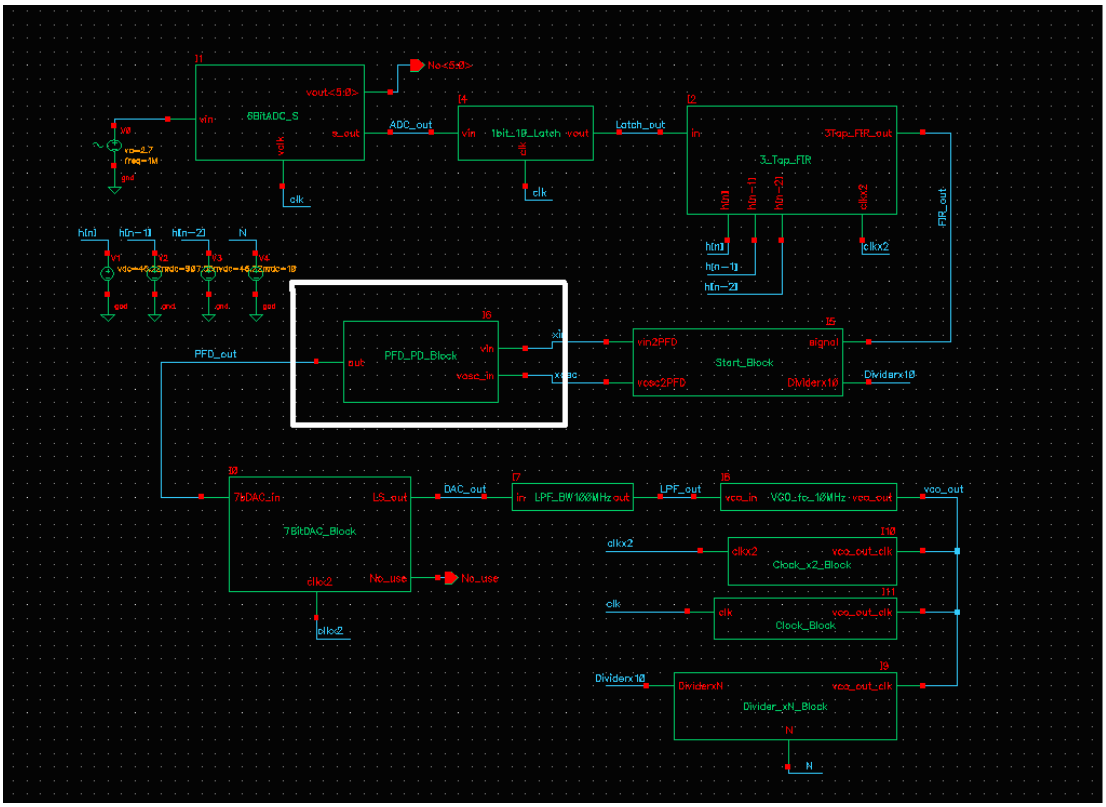


Fig.21 A practical phase-frequency detector in Cadence

The output of 7-bit counter presents the digitized information to the 7-bit digital-to-analog converter. Decoded by the DAC, the analog loop filter will get an analog signal to conduct for the following voltage-controlled oscillator. The frequency output of the VCO will then be controlled by the decoded voltage level.

Chapter 4

Test Bench Studies

In addition to the designed function blocks, it is also important to have test benches that can help verify the performances of our designs. Verilog-A provides several commands to generate random numbers in specified distributions that may be useful for our needs. The test benches studied in this report only provide a starting point to experience verification processes. To generate more sets of test benches in the future can assist and modify our design more perfectly.

Verilog-A provides functions to generate random numbers in the following distribution patterns:

* Pseudo-Random
* Uniform
* Normal (Gaussian)
* Exponential
* Poisson

The main function generators used in this report are focused on the pseudo-random, uniform and normal distributions. They are described in the following sections.

4-1. Random-Bit Noise Generator

To realistically simulate the environments that a real circuit may encounter, it is necessary to include noise generators for our simulations. The uniform-distributed, normal-distributed (Gaussian) and pseudo-random noise

generators are discussed.

4-1-1. Uniform-Distributed Random-Bit Generator

The uniform-distributed random-bit generators in Verilog-A include two functions, $rdist_uniform( ) and $dist_uniform( ). The former function is to generate real numbers and the later one is to generate integer numbers evenly distributed throughout a specified range. Fig.22 shows our Verilog-A code of the uniform-distributed random-bit generator used in this project followed by its simulation waveform as shown on Fig.23.

```
`include "constants.h"
`include "discipline.h"
// To generate random integer numbers that are evenly distributed
module Uniform_Random(in, out);
    input       in;
    output    out;
    electrical    in, out;
    parameter integer start_range = -2.5;
    integer      seed, end_range;
    real          ran_num;
    analog begin
        @(initial_step) begin
                seed          = 2;
                end_range = 2.5;
        end
        ran_num = $dist_uniform(seed, start_range, end_range);
        V(out) <+ V(in)+ran_num;
    end
endmodule
```

Fig.22 Uniform-Distributed Random-Bit Generator

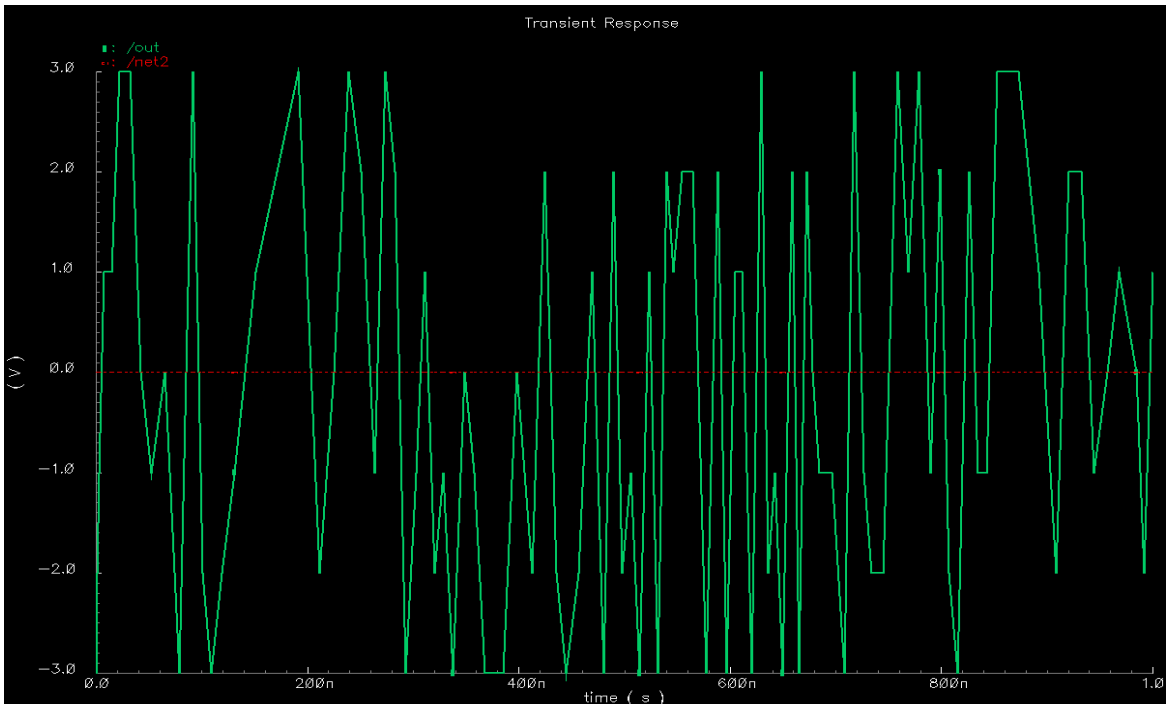


Fig.23 Waveform of the Uniform-Distributed Random-Bit Generator

4-1-2. Normal-Distributed Random-Bit Generator

The normal-distributed (Gaussian) random-bit generator in Verilog-A code is shown on Fig.24. Its simulation waveform can be apparently seen on Fig.25.

```
integer        seed;
parameter    integer mean=0;
parameter    integer standard_deviation=1;
real             ran_num;
analog begin
   @(initial_step) begin
         seed    = 25;
   end
   ran_num = $rdist_normal(seed,mean,standard_deviation);
   V(out) <+ V(in)+ran_num;
end
```

Fig.24 Normal-Distributed Random-Bit Generator

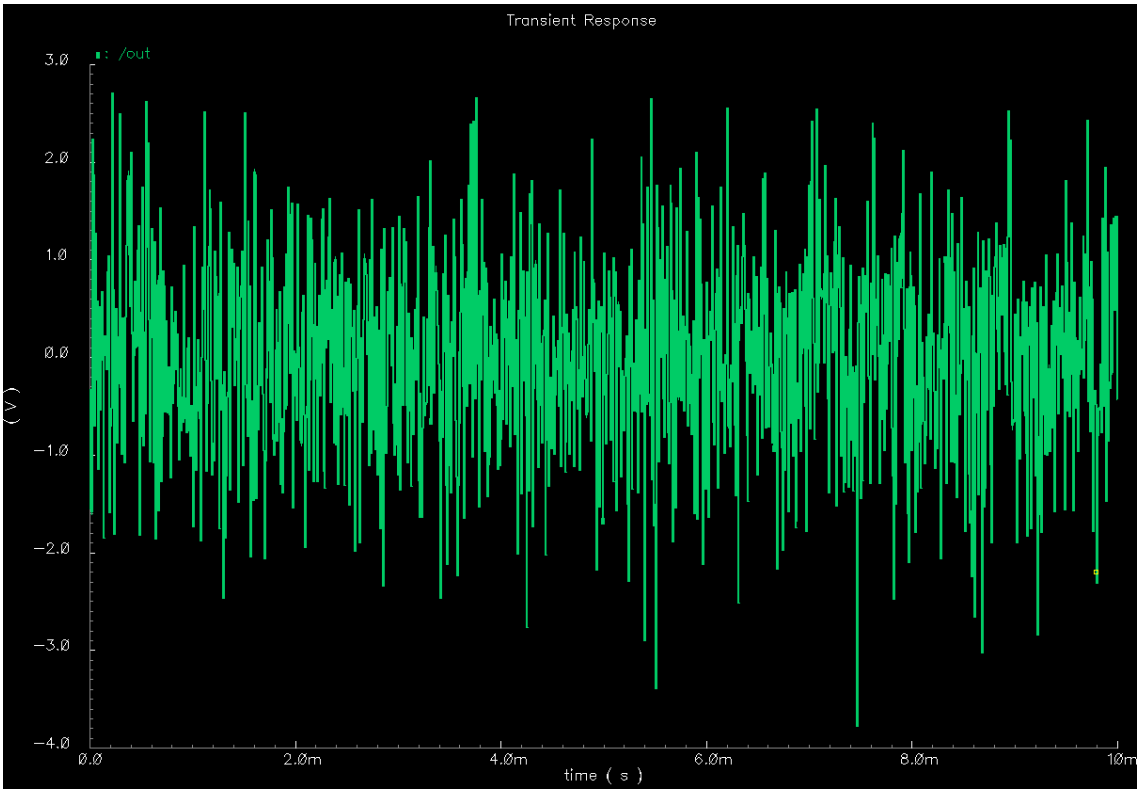


Fig.25 Waveform of the Normal-Distributed Random-Bit Generator

4-1-3. Pseudo-Random Random-Bit Generator

Pseudo-random random-bit generator coded in Verilog-A can accompany with a timer function. The timer function - timer() can help designers extend the visibility of a random-bit. It's defined as a function of timer(start, period). Fig.26 shows the pseudo-random random-bit generator in Verilog-A, followed by its simulation waveform on Fig.27.

```
`include "constants.h"
`include "discipline.h"
module Random_Pseudo_Generator(outbit);
   electrical outbit;
   integer seed, num;
   analog begin
       @(initial_step) begin
           seed = 1;         // initialize point
       end
       num = abs($random(seed) % 2);   //imodulate by 2 to  generates 0 & 1
       if (num < 0.5)
        V(outbit) <+ 0.0;
       else
          V(outbit) <+ 5.0;
   end
endmodule
```

Fig.26 Pseudo-Random Random-Bit Noise Generator

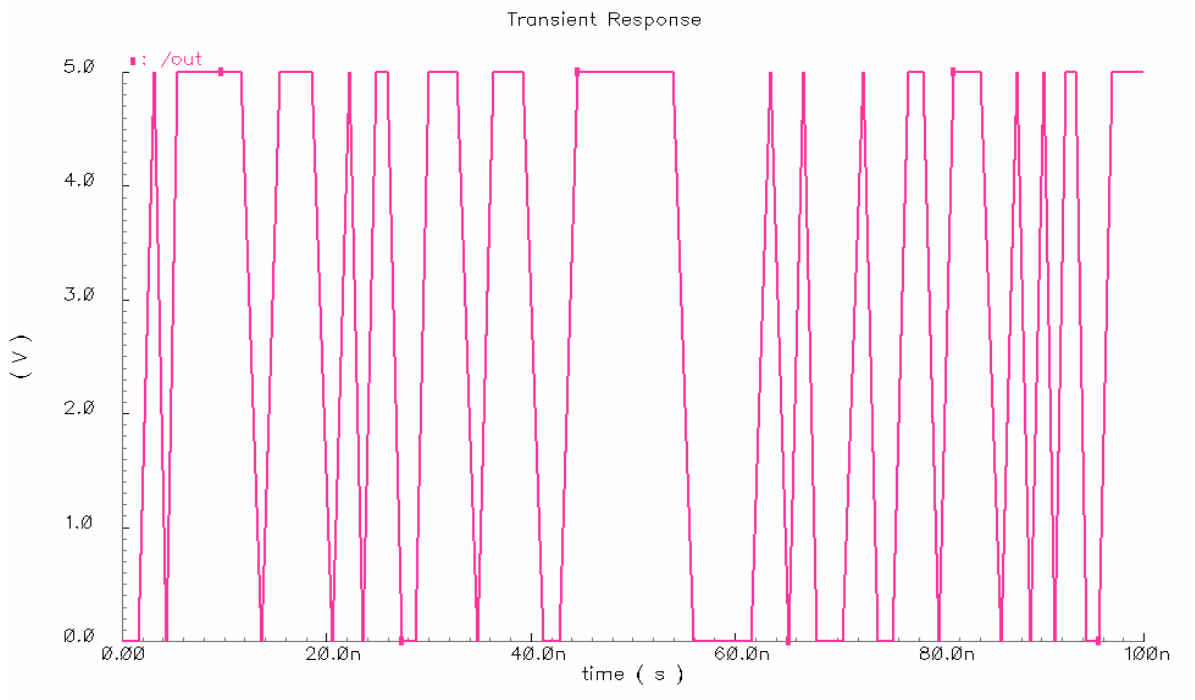


Fig.27 Waveform of the pseudo-random random-bit generator

4-2. Random-Bit-Generator-Included DAC

In this section, the analog path between the digital-to-analog converter and the voltage-controlled-oscillator is simulated while inserting a random-bit noise generator into it. This process can be used to verify the satisfaction of the following analog filter design. The certain path includes a DAC, a noise generator and an analog low pass filter. Its schematic is shown on Fig.28. The netlist of such kind of testbench is attached as appendix D.

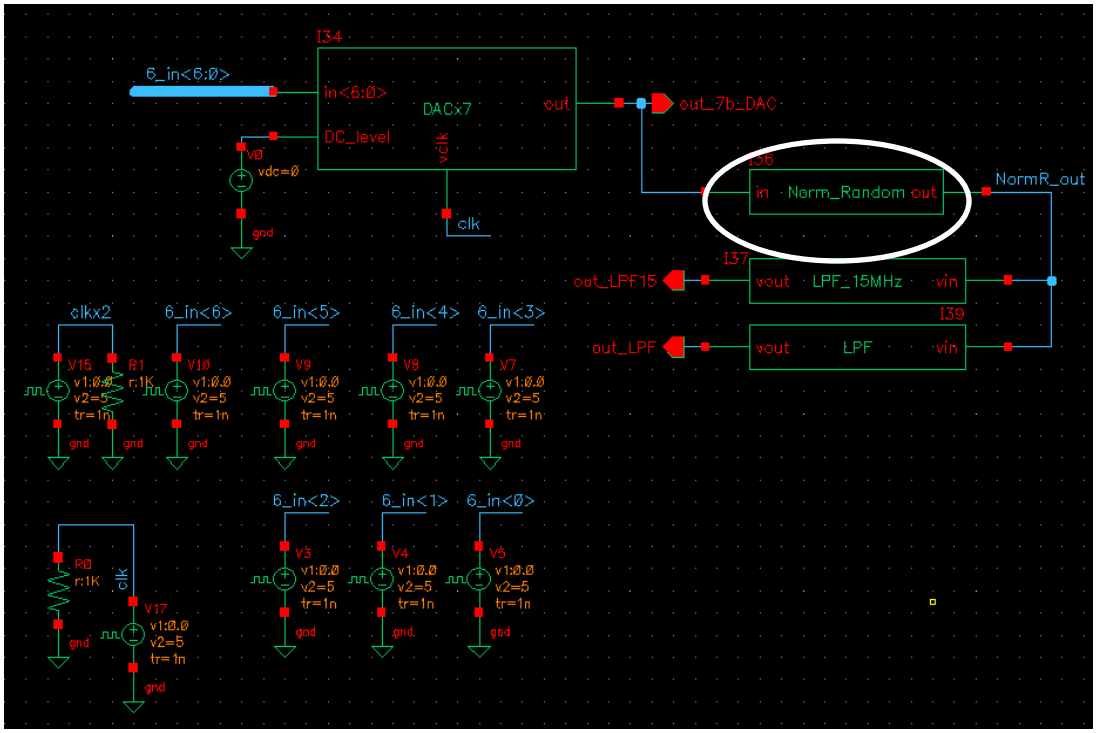


Fig.28 Schematic of the random-bit-generator-included test bench

After inserting the pseudo-random random-bit noise generator into the path discussed above, a simulation was run for satisfactions. There are two filters connected to the end of such analog path to compare performances of their output

waveforms. As we can see, the ideally designed waveform provided by the 7-bit digital-to-analog converter is DAC_out. It is used to drive the voltage controlled oscillator for different frequencies. Before considering noises, the designed ideal low pass filter which has 100MHz as its bandwidth is good enough to generate clear signals to drive the oscillator. However, the output waveform shows us the driving source gets worse if there were noises. Comparing the waveform comes out from the 100MHz filter (labeled 100MHz_out_pseudo on Fig.29) with that from the 15MHz filter (labeled 15MHz_out_pseudo on Fig.29), it is easily found an improvement occurs. Through such testbench experiments, we can begin to modify or improve our designs. In the future, the ideal low pass filter will be replaced with the other realistic components such as analog Butterworth or Chebyshev low pass filters. Building up more testbenches will also be our future works to more guarantee the performance of this project.

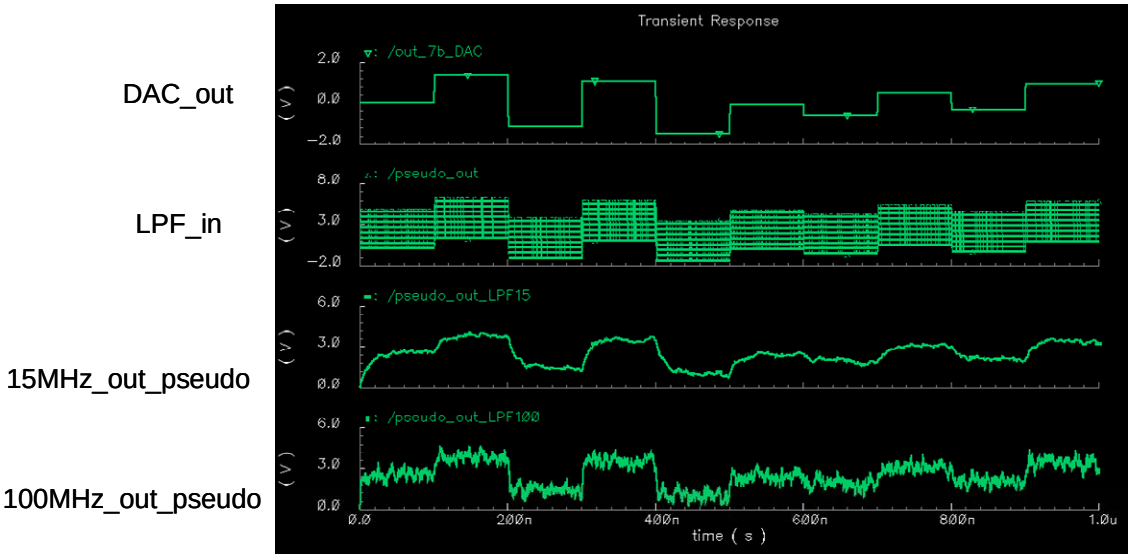


Fig.29 Waveforms of the random-bit-generator-included test bench

Chapter 5

Entire DPLL Design and Simulation

5-1. Schematic of the Entire DPLL Design

After establishing Verilog-A models for all the essential blocks, we build up an entire digital phase-lock loop circuit as shown on Fig.22. Verilog-A models of all the block diagrams are coded in this report. However, not all of them are discussed in details. Block diagrams and Veriolog-A models that are not included in these chapters will be covered in the appendices.

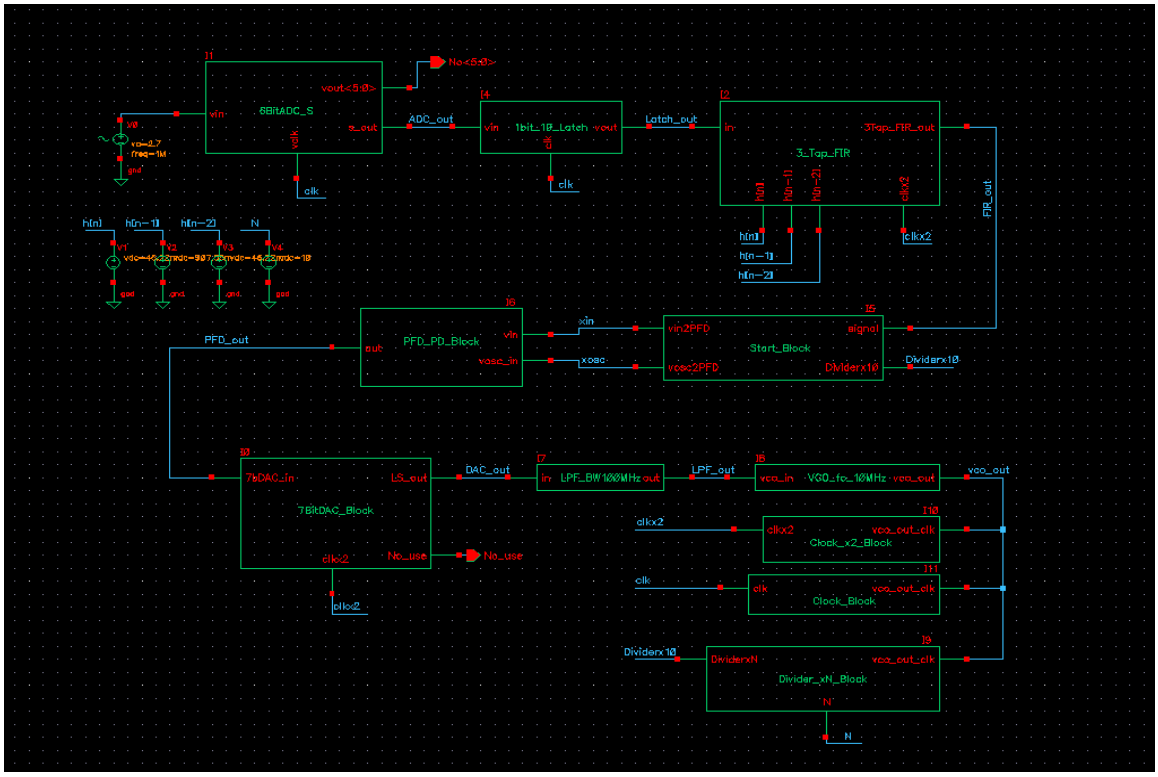


Fig.30 Entire digital phase-locked loop blocks in Cadence

5-2. Simulation Waveform of the Entire DPLL Design

Running simulation by means of the Cadence Analog Artist simulator, we get the simulation results presented on Fig.23 for the system level Verilog-A DPLL circuit. As we can see, the output voltage of LPF_out is reset to zero volts at 8.5us, after that time the correct output will begin to produce out. The oscillating frequency, vco_out, changes its value whenever the plus or minus signal presents. Within a certain lock-in time, we are confident that the phase-locked loop will become stable and synchronize to the frequency of input signal that comes in from the hard-disk.

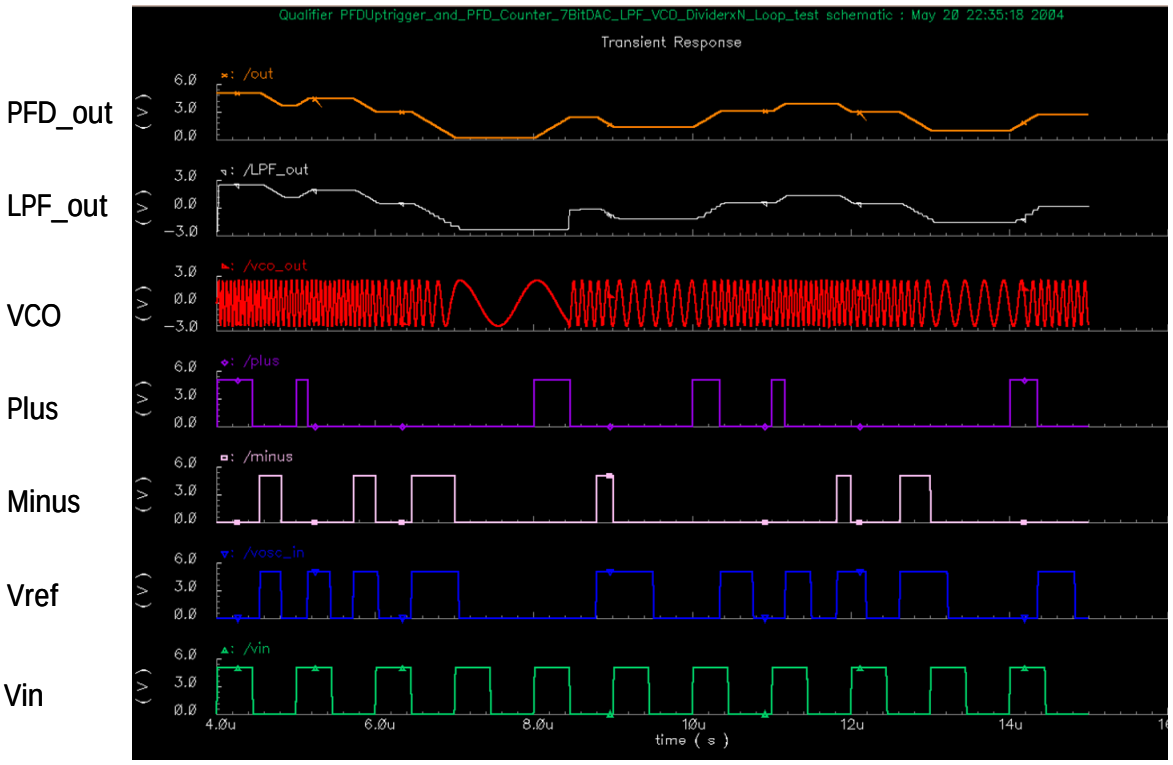


Fig.31 Waveform of the entire DPLL

Chapter 6

Conclusion

According to the system-level Verilog-A models and simulations presented in this report, we can recognize several advantages of Verilog-A. First, due to that implementing system-level designs and simulations are necessary for the modern complicated mixed signal circuits. Verilog-A becomes a very excellent choice to accomplish this task. Second, modeling in Verilog-A the behavior functions of a block that is going to be designed is fully compatible with Verilog, a pure digital hardware design description language. We can take their merits such as fast design, easy to debug in software and reusable ability in the whole design process. The future capability for automatic synthesis is also becoming one of the most important goals of Verilog-A. Many of the other characteristics and advantages are listed in Table.2. Also a practical design of the Verilog-A digital phase-locked-loop circuit is experienced in this report to convince designers about the compatibilities of Verilog-A in Cadence environment as well.

	Design Methodology Comparison		
	Mixed Signal Design Methodology		Digital Design Methodology
	Schematic Capture / SPICE (Button-Up process)	**AHDLs (Verilog-A) (Top-Down process)**	**HDLs (VHDL, Verilog) (Top-Down process)**
Speed (Fast)	--	V	V
Debug (Easy)	--	V	V
Synthesizable	--	* Potential	V
Digital Model Support	--	V	V
s-Domain Model Support	V	V	--
z-Domain Model Support	V	V	--
Differential Equation Support	V	V	--
Analog Model Support	V	V	--
System-Level Simulation Capability	--	V	V (Digital Only)
Simulation Accuracy	V (Best)	--	--
Transistor-Level Model / Simulation	V	--	--
Conditional Model / Simulation	--	V	V
Behavior Model / Simulation	--	V	V
Structural Model / Simulation	--	V	V
Widely Model Interchange Support	V	--	V
Capability for Exchanges of HDLs Models with Analog Functions	--	* Potential	--
Models Separated from Simulator	--	V	V

Table.2 Summarized Comparison of Hardware Design Methodologies

Reference A

[1] Razavi, Design of Analog CMOS ICs.

[2] Toshio Murayama, Yuji Gendai, “A Top-Down Mixed-Signal Design Methodology Using a Mixed-Signal Simulator and Analog HDL”, 1996, IEEE

[3] Roland E. Best, “Phase-Locked Loops, Design, Simulation, and Application”

Reference B

Equalizer and FIR:

1-1. Patrick Pai, A. A. Abidi and Ramon A. Gomez. “A Simple Continuous-Time Equalizer for Use in Magnetic Storage Read Channels”. In Comm, Vol,. 10 No, 1, January 1992, IEEE.

1-2. Hui Wu, Jose A. Tierno, Petar Pepelijugoski, “Integrated Transversal Equalizer in High-Speed Fiber-Optic Systems”. JSSC, Vol. 38, No.12, December 2003, IEEE.

1-3. Vojin G. Oklobdzua. “Circuit Implementation Techniques For The Magnetic Read/Write Channels”.

1-4. Robert B. Staszewski, Khurram Muhammad, Poras T. Balsara. “A Constrained Asymmetry LMS Algorithm for PRML Disk Drive Channels”, Trans., on circuit and systems. Vol.48, No.8, August 2001. IEEE

1-5. Tertulien Ndjountche and Rolf Unbehauen. “A Low-Power And High-Speed Equalizer For Magnetic Storage Read Channels”

1-6. Patrick K. D. Pai, Anthony D. Brewster, and Asad A. Abidi. “A 160-MHz Analog Front-End IC for EPR-IV PRML Magnetic Storage Read Channels”. JSSC, Vol.31, No.11, November 1996, IEEE

1-7. Derrick C. Wei, Daniel Q. Sun, Asad A. Abidi, “Mostly Analog Disk Drive Read Channel with Practical Depth-of-Two Fixed Delay Tree Search”. Trans., on magnetics, Vol.38, No.6, November 2002. IEEE.

1-8. Roy D. Cideciyan, Francois Dolivo, Reto Hermann, Walter Hirt, Wolfgang Schott. “A PRML System for Digital Magnetic Recording”. In Comm., Vol.10, No.1, January 1992. IEEE.

1-9. S. Hossein Mousavinezhad, Ikhlas M. Abdel-Qader, “Digital Signal Processing In Theory and Practice”. 31st ASEE/IEEE Frontiers in Education Conference.

1-10. Chapter 7 of “Implementing High-Performance DSP Functions in Stratix & Stratix GX Devices”. Altera Corporation. April 2003.

1-11. Application Notes 005, “Implementating an FIR Filter using the VERSA1 MAC”. Goal Semiconductor.

Analog-to-Digital (AD) and Digital-to-Analog (DA) Converter:

2-1. Thomas Conway, Philip Quinlan, Joe Spalding, Kevin McCall. “A CMOS 260Mbps Read Channel with EPRML Performance”. 1998 Symposium on VLSI Circuit Digest of Technical Papers.

2-2. Johns and Martin “Analog Integrated Circuit Design”.

Phase-Locked Loop (PLL):

3-1. Teresa M. Alemeida, Moises S. Piedade. “High Performance Analog And Digital PLL Design”. 1999, IEEE.

3-2. Terng-Yin Hsu, Bai-Jue Shieh, Chen-Yi Lee. “An All-Digital Phase-Locked Loop (ADPLL)-Based Clock Recovery Circuit”. JSSC, Vol.34, No.8, August 1999. IEEE

3-3. Y. Fouzar, M. Sawan, Y. Savaria. “Very Short Locking Time PLL Based on Controlled Gain Technique”. 2000, IEEE.

3-4. Razavi, “Design of Analog CMOS Integrated Circuits”

Analogue Hardware Description Language (AHDL and Verilog-A):

4-1. R. J. Binns, P.Hallam, B. Mack, R. Massara “High-Level Design of Analogue Circuitry Using an Analogue Hardware Description Language”.

4-2. Toshi Murayama, Yuji Gendai. “A Top-Down Mixed-Signal Design Simulator and Analog HDL”. 1996. IEEE

4-3. Robert Sobot, Shawn Stapleton, Marek Syrzycki. “Behavioral Modeling of Continuous Time $\Delta\Sigma$ Modulators”. 2003, IEEE.

4-4. Dan FitzPatrick, Ira Miller. “Analog Behavioral Modeling with the Verilog-A Language”

Miscellaneous:

5-1. Kenichi Ohhata, Fumihiko Arakawa, Toru Masuda, Nobuhiro Shiramizu. “40-Gb/s Analog IC Chipset for Optical Receivers – AGC Amplifier, Full-Wave Rectifier and Decision Circuit “. 2001, IEEE

5-2. Vojin G. Oklobdzija. “Circuit Implementation Techniques for the Magnetic Read/Write Channels”. Project Report 2001-2002 for MICRO Project.

5-3. Asad A. Abidi. “Integrated Circuits in Magnetic Disk Drives”. 20th European SSC Conference. September 20-22, 1994.

Appendix A: Verilog-A Code

(A-1) Adder

```
// VerilogA for Qualifier, adder, veriloga
// Designer: CHING-HONG WANG
// Purpose : Qualifier Assigned Project
`include "constants.h"
`include "discipline.h"
module adder_Noclk(in1,in2,in3,out);
input in1,in2,in3;
output out;
electrical in1,in2,in3;
electrical out;
real out_reg;
analog begin
out_reg = V(in1)+V(in2)+V(in3);
V(out) <+ transition(out_reg,0,0,0);
end
endmodule
```

(A-2) Delay Block

```
// VerilogA for Qualifier, delay, veriloga
// Designer: CHING-HONG WANG
// Purpose : Qualifier Assigned Project
`include "constants.h"
`include "discipline.h"
module dlay(in,clk,out);
input in;
input clk;
output out;
electrical in,clk,out;
parameter real vtrans_clk=2.5;
```

```
real out_reg;
analog begin
@(cross(V(clk)-vtrans_clk, +1)) begin
out_reg = V(in);
end
V(out) <+  transition(out_reg,0,0,0);
end
endmodule
```

(A-3) Multiplier

```
// VerilogA for Qualifier, multiplier, veriloga
// Designer: CHING-HONG WANG
// Purpose : Qualifier Assigned Project
`include "constants.h"
`include "discipline.h"
module multiplier_Noclk(in1,in2,out);
input in1,in2;
output out;
electrical in1,in2,out;
real out_reg;
analog begin
out_reg = V(in1) * V(in2);
V(out) <+ transition(out_reg, 0,0,0);
end
endmodule
```

(A-4) Low-Pass Filter

```
// VerilogA for Qualifier, LPF, veriloga
// Designer : CHING-HONG WANG
// Purpose  : Qualifier Assigned Project
```

```
`include "constants.h"
`include "discipline.h"
`define PI  3.14159265358979328468643383279502884197l
module LPF(vin,vout);
   input   vin;
   output  vout;
   electrical vin, vout;
   parameter real LPF_BW_Hz=100000000;  // BW=100MHz(DAC's fs =10MHz)
   real r;
   real c;
   real Wc;
   analog begin
       @(initial_step("tran","ac","dc")) begin
           r=1k;
           Wc=2*`PI*LPF_BW_Hz;
           c=1/(r*Wc);
       end
       V(vout,vin)  <+  I(vout,vin)*r;
       I(vout)      <+  ddt(V(vout)*c);
   end
endmodule
```

(A-5) 6-Bit Analog-to-Digital Converter (6-Bit ADC)

```
// VerilogA for Qualifier, Serial_6bitADC, veriloga
// Designer : CHING-HONG WANG
// Purpose  : Qualifier Assigned Project
`include "constants.h"
`include "discipline.h"
module SerialADCbit6(vout,vin,vclk,s_out);
    input vin,vclk;
    output [5:0] vout;
    output s_out;
    electrical [5:0] vout;
```

```
    electrical vin, vclk;
    electrical s_out;
    parameter real trise = 0 from [0:inf);
    parameter real tfall = 0 from [0:inf);
    parameter real tdel = 0 from [0:inf);
    parameter real vinh= 5;
    parameter real vinl= 0;
    parameter real voh = 5;
    parameter real vol = 0;
    parameter real vth_clk = 2.5;
    parameter real vth_bit = 2.5;
    real vref;
    real midref;
    real sampledV;
    real vdout[5:0];
    real pow2[6:0];
    real code,power_base,output_range;
    integer i,j,k,m;

analog begin
    @(initial_step) begin
       vref = (vinh-vinl)/2 + vinl;
       midref = vref / 2;
       output_range = (voh-vol);
       generate j(5,0) begin
         vdout[j] = 0;
       end
       pow2[0] = 1.0;
       for (k=1; k<=6;k=k+1)
            pow2[k]=2.0 * pow2[k-1];
    end
    @(cross(V(vclk) - vth_clk, 1)) begin
       sampledV = V(vin);
       for (i = 5; i >= 0 ; i = i - 1) begin
            vdout[i] = 0;
            if (sampledV > vref) begin
```

```
                vdout[i] = voh;
                sampledV = sampledV - vref;
            end
            else begin
                vdout[i] = vol;
            end
            sampledV = sampledV * 2;
        end
        code = 0.0;
         generate m (5,0) begin
              if (vdout[m] < vth_bit)
                 power_base= 0.0;
              else
                 power_base = pow2[m];
                code=code+power_base;
         end
         code = output_range/(pow2[6]-1) * code;
end
generate i (5,0) begin
   V(vout[i]) <+ transition(vdout[i], tdel, trise, tfall );
end
V(s_out)   <+ transition(code,tdel, trise, tfall );
end
endmodule
```

(A-6) 7-Bit Digital-to-Analog Converter (7-Bit DAC)

```
// VerilogA for Qualifier, 7bitSerialIn_DAC, veriloga
// Designer : CHING-HONG WANG
// Purpose  : Qualifier Assigned Project
`include "constants.h"
`include "discipline.h"
module DACbit6_SerialIn(s_in,in,DC_level,vclk,out,s_out);
   input       DC_level,vclk ;
```

```
input       [6:0] in;
input       s_in;
output      out,s_out ;
electrical  [6:0] in;
electrical  s_in, s_out;
electrical  DC_level,out,vclk;
parameter   real   vinh = 5;
parameter   real   vinl = 0;
parameter   real   voh  = 5;
parameter   real   vol  = 0;
parameter   real   vth_clk = 2.5;
parameter   real   output_range = 5 ;
parameter   real   trise        = 1n from [0:inf);
parameter   real   tfall        = 1n from [0:inf);
real        code, power_base;
real        pow2  [7:0];
real        vth_bit;
real        sampledV;
real        vdout[6:0];
real        vin_range, midref, vref;
real        code2, power_base2;
integer     i,j,k,m,n;
analog begin
   @(initial_step) begin
        vin_range = (vinh-vinl);
        vref = (vin_range)/2 + vinl;
        midref = vref / 2;
        generate k(6,0) begin
           vdout[k] = 0;
        end
        vth_bit=(vinh+vinl)/2;
        pow2[0] = 1.0;
        for (i=1; i<=7;i=i+1)
            pow2[i]=2.0 * pow2[i-1];
   end
   @(cross(V(vclk)-vth_clk, 1)) begin
```

```
        code = 0.0;
        generate j (6,0) begin
             if (V(in[j]) < vth_bit)
                power_base= 0.0;
             else
                power_base = pow2[j];
               code=code+power_base;
        end
        code = output_range/(pow2[7]-1) * code;
        sampledV = V(s_in);
        for (m = 6; m >= 0 ; m = m - 1) begin
              vdout[m] = 0;
              if (sampledV > vref) begin
                 vdout[m] = voh;
                 sampledV = sampledV - vref;
              end
              else begin
                 vdout[m] = vol;
              end
              sampledV = sampledV * 2;
        end
        code2 = 0;
        generate n (6,0) begin
              if (vdout[n] < vth_bit)
                   power_base2 = 0.0;
              else
                   power_base2 = pow2[n];
              code2 = code2+power_base2;
        end
        code2 = output_range/(pow2[7]-1) * code2;
     end // cross-begin
   V(out)    <+ V(DC_level) + transition(code,0,trise,tfall) ;
   V(s_out)  <+ V(DC_level) + transition(code2,0,trise,tfall);
   end
endmodule
```

(A-7) 6-Bit Digital-to-Analog Converter (6-Bit DAC)

```
// VerilogA for Qualifier, 6bitSerialIn_DAC, veriloga
// Designer : CHING-HONG WANG
// Purpose  : Qualifier Assigned Project
`include "constants.h"
`include "discipline.h"
module DACbit6_SerialIn(s_in,in,DC_level,vclk,out,s_out);
   input       DC_level,vclk ;
   input       [5:0] in;
   input       s_in;
   output      out,s_out ;
   electrical  [5:0] in;
   electrical  s_in, s_out;
   electrical  DC_level,out,vclk;
   parameter   real   vinh = 5;
   parameter   real   vinl = 0;
   parameter   real   voh  = 5;
   parameter   real   vol  = 0;
   parameter   real   vth_clk = 2.5;
   parameter   real   output_range = 5 ;
   parameter   real   trise        = 1n from [0:inf);
   parameter   real   tfall        = 1n from [0:inf);
   real        code, power_base;
   real        pow2  [6:0];
   real        vth_bit;
   real        sampledV;
   real        vdout[5:0];
   real        vin_range, midref, vref;
   real        code2, power_base2;
   integer     i,j,k,m,n;
   analog begin
      @(initial_step) begin
           vin_range = (vinh-vinl);
           vref = (vin_range)/2 + vinl;
           midref = vref / 2;
```

```
      generate k(5,0) begin
         vdout[k] = 0;
      end
      vth_bit=(vinh+vinl)/2;
      pow2[0] = 1.0;
      for (i=1; i<=6;i=i+1)
          pow2[i]=2.0 * pow2[i-1];
end
@(cross(V(vclk)-vth_clk, 1)) begin
   code = 0.0;
   generate j (5,0) begin
        if (V(in[j]) < vth_bit)
           power_base= 0.0;
        else
           power_base = pow2[j];
           code=code+power_base;
   end
   code = output_range/(pow2[6]-1) * code;
   sampledV = V(s_in);
   for (m = 5; m >= 0 ; m = m - 1) begin
         vdout[m] = 0;
         if (sampledV > vref) begin
            vdout[m] = voh;
            sampledV = sampledV - vref;
         end
         else begin
            vdout[m] = vol;
         end
         sampledV = sampledV * 2;
   end
   code2 = 0;
   generate n (5,0) begin
         if (vdout[n] < vth_bit)
              power_base2 = 0.0;
         else
              power_base2 = pow2[n];
```

```
                code2 = code2+power_base2;
          end
          code2 = output_range/(pow2[6]-1) * code2;
       end // cross-begin
   V(out)    <+ V(DC_level) + transition(code,0,trise,tfall) ;
   V(s_out)  <+ V(DC_level) + transition(code2,0,trise,tfall);
   end
endmodule
```

(A-8) NoiseAdder

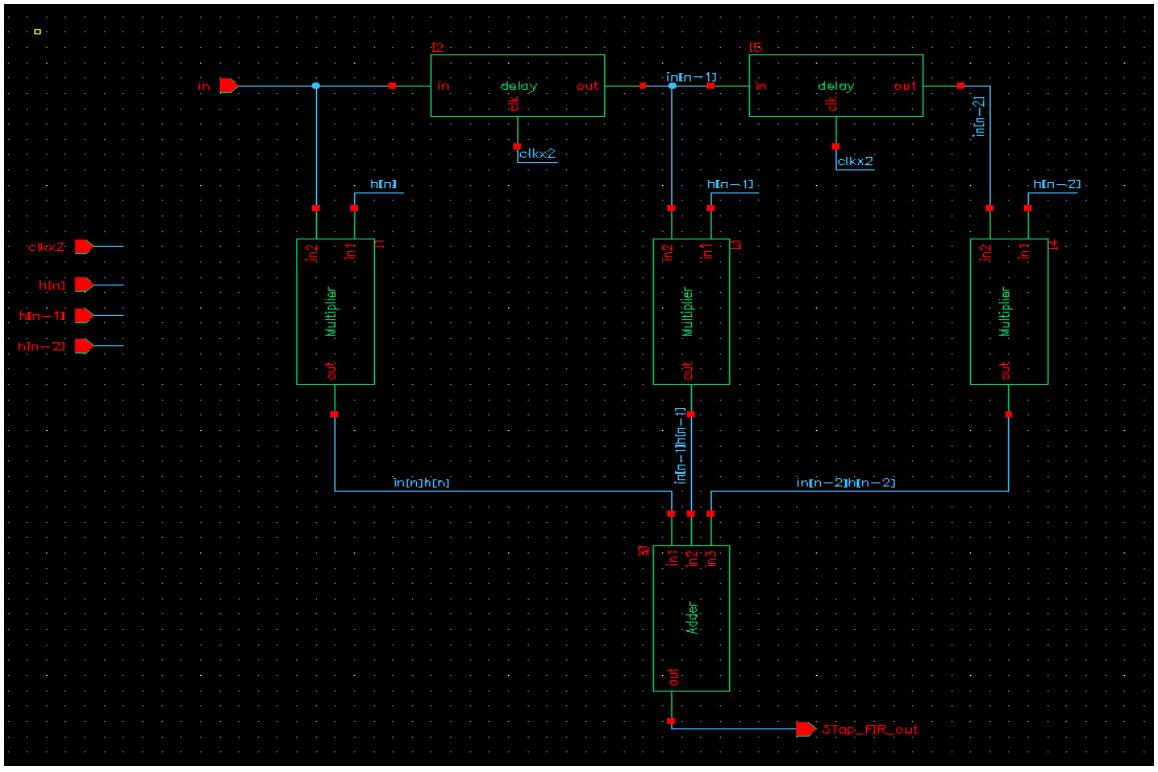
```
// VerilogA for Qualifier, NoiseAdder, veriloga
// Designer: CHING-HONG WANG
// Purpose : Qualifier Assigned Project
`include "constants.h"
`include "discipline.h"
module NoiseAdder(in,noise,out);
     input in, noise;
     output out;
     electrical in, noise, out;
     real in_value;
     real noise_value;
     real out_value;
    analog begin
        @(initial_step) begin
            in_value = 0;
            noise_value =0;
            out_value =0;
        end
        in_value = V(in);
        noise_value = V(noise);
        out_value = in_value + noise_value;
        V(out)  <+ transition(out_value,0,0,0);
     end
endmodule
```

(A-9) LevelShifter

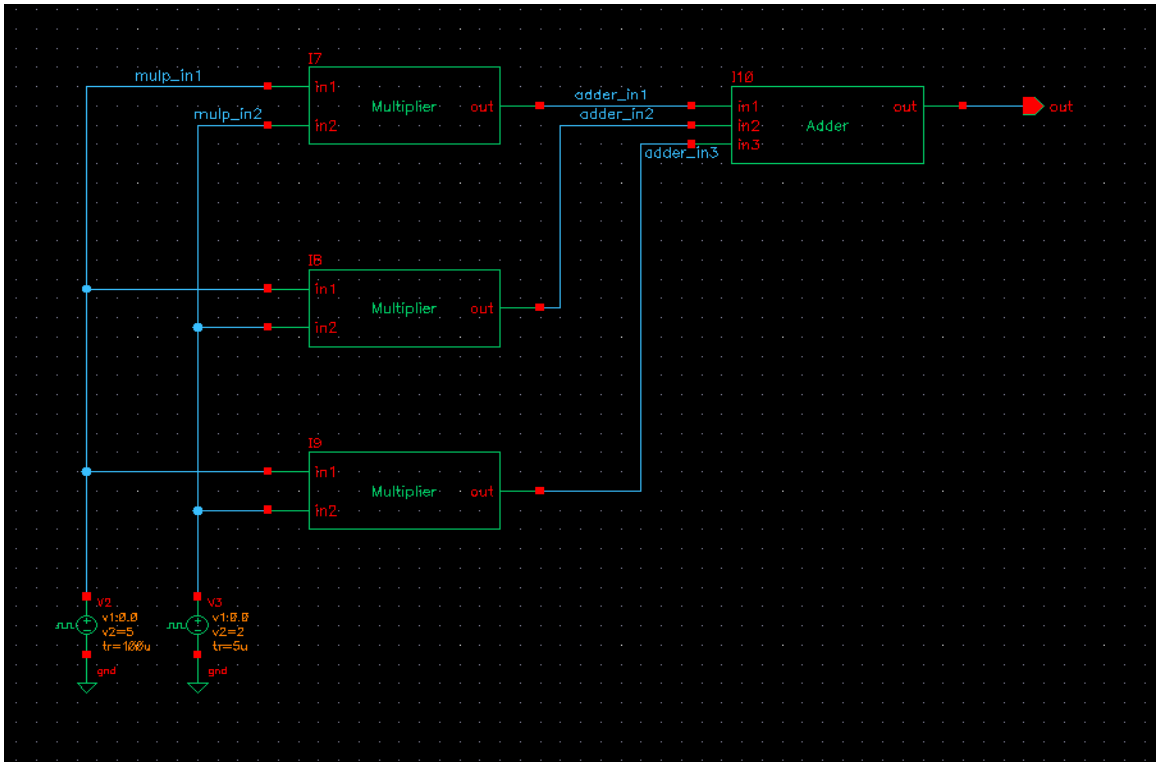
```
// VerilogA for Qualifier, LevelShift, veriloga
// Designer: CHING-HONG WANG
// Purpose : Qualifier Assigned Project
`include "constants.h"
`include "discipline.h"
module LevelShift(in, out);
   input   in;
   output  out;
   electrical in,out;
   parameter  real  shiftvalue=0;
   parameter  real  level_base = 2.5;
   real level_get;
   real level_out;

   analog begin
      level_get = V(in);
      level_out = level_get - level_base;
      V(out) <+ transition(level_out,0,0,0);
   end
endmodule
```

Appendix B:

(B.1) 3-Tap FIR schematic in Cadence Environment



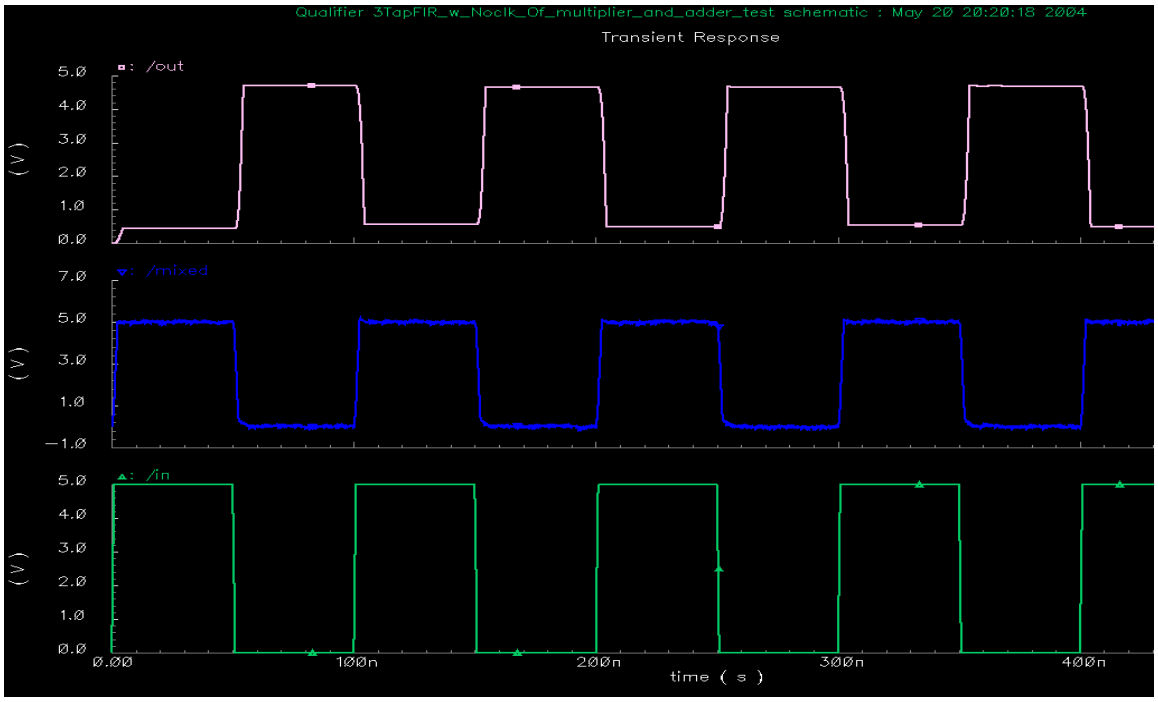
(B-2) Schematic of Adder plus Multiplier



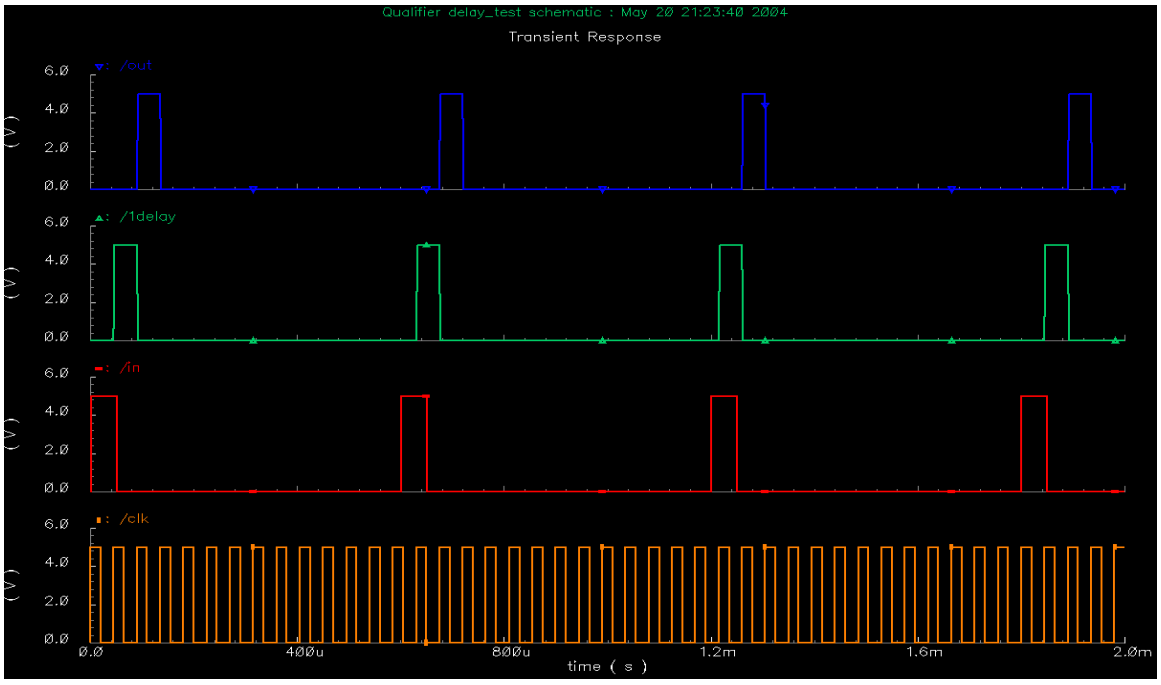
Appendix C:

(C-1) 3-Tap FIR simulation results on a noisy input signal

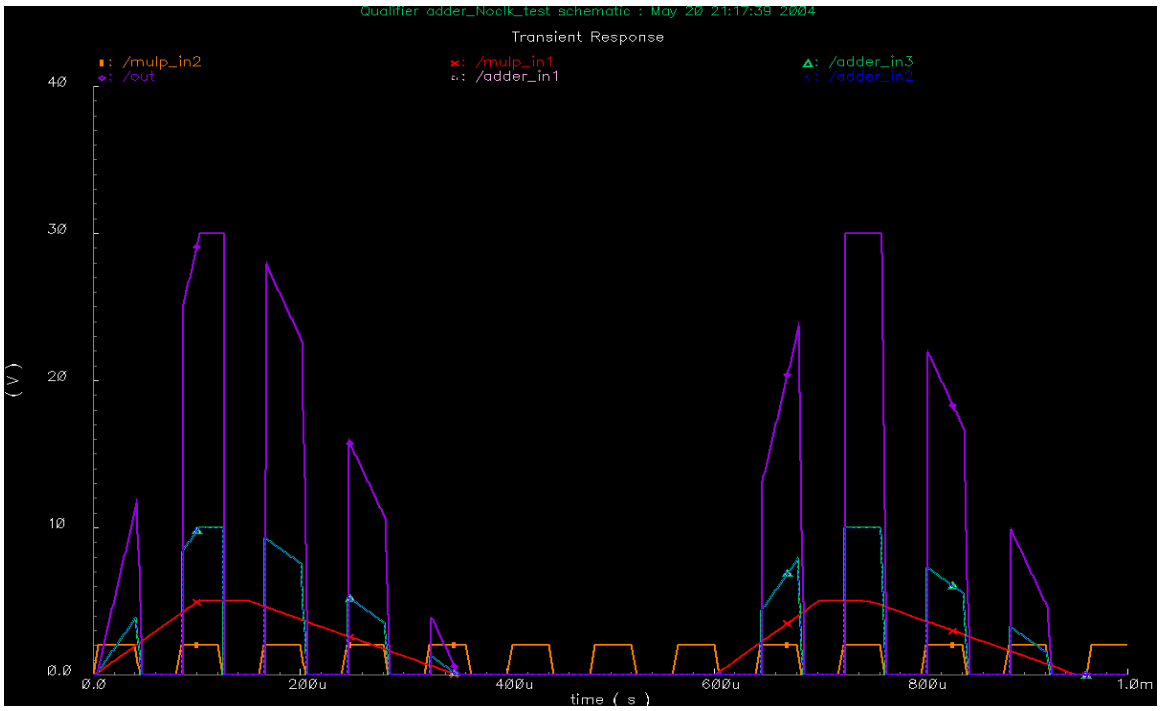
The waveforms from top to bottom are output of 3-Tap FIR filter, noisy input signal and clear input signal respectively.



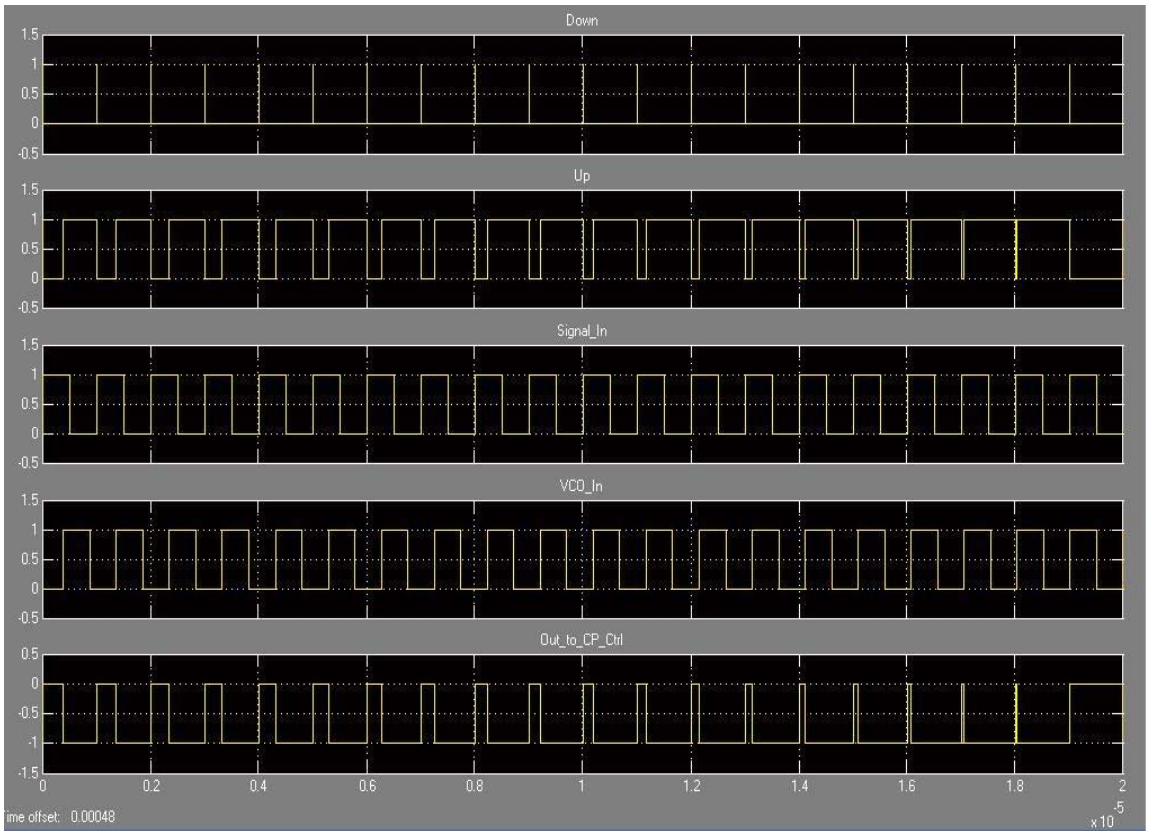
(C-2) Simulation result of the delay block



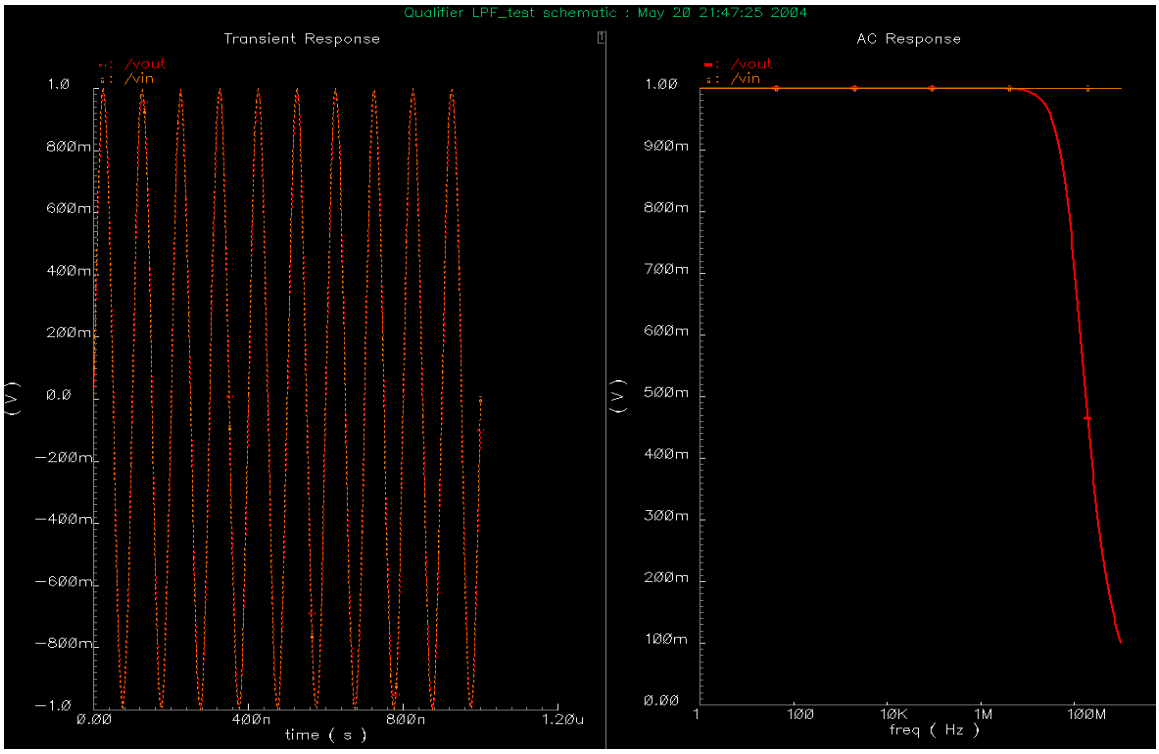
(C-3) Simulation result of the function of multiplier followed by an adder



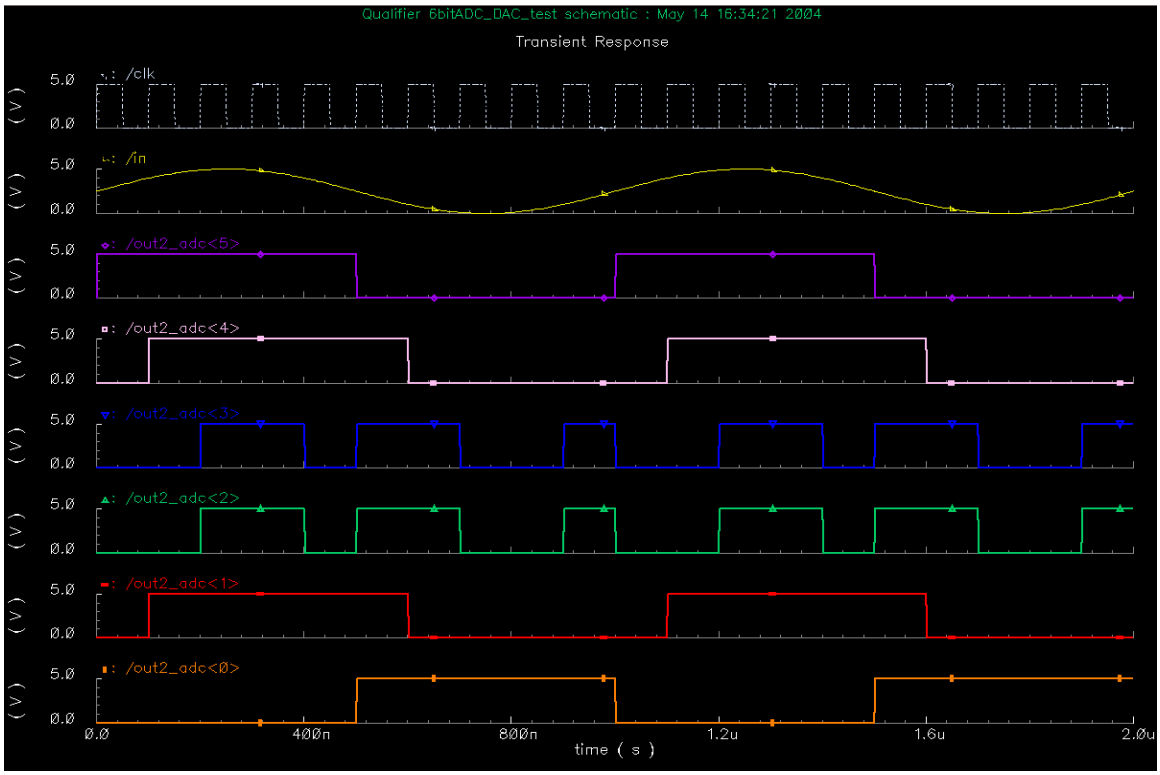
(C-4) Matlab simulation result of a conventional phase-frequency detector



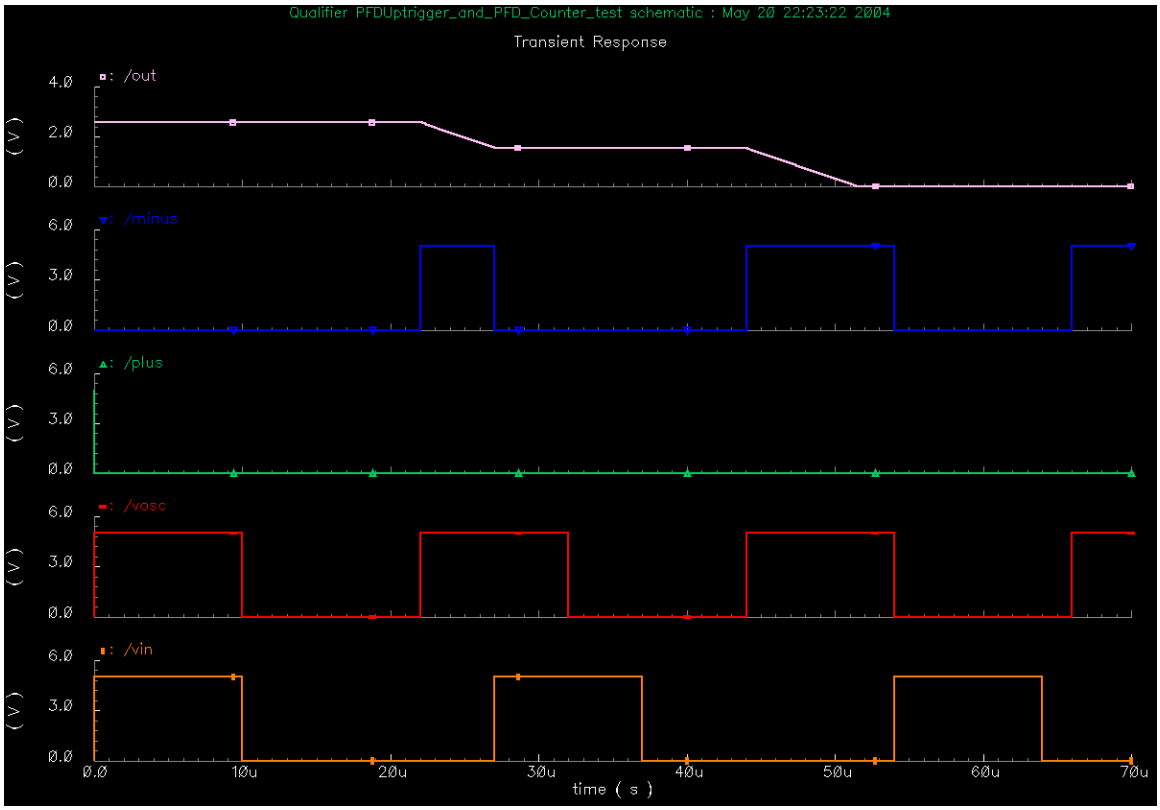
(C-5) Simulation results of a low-pass filter



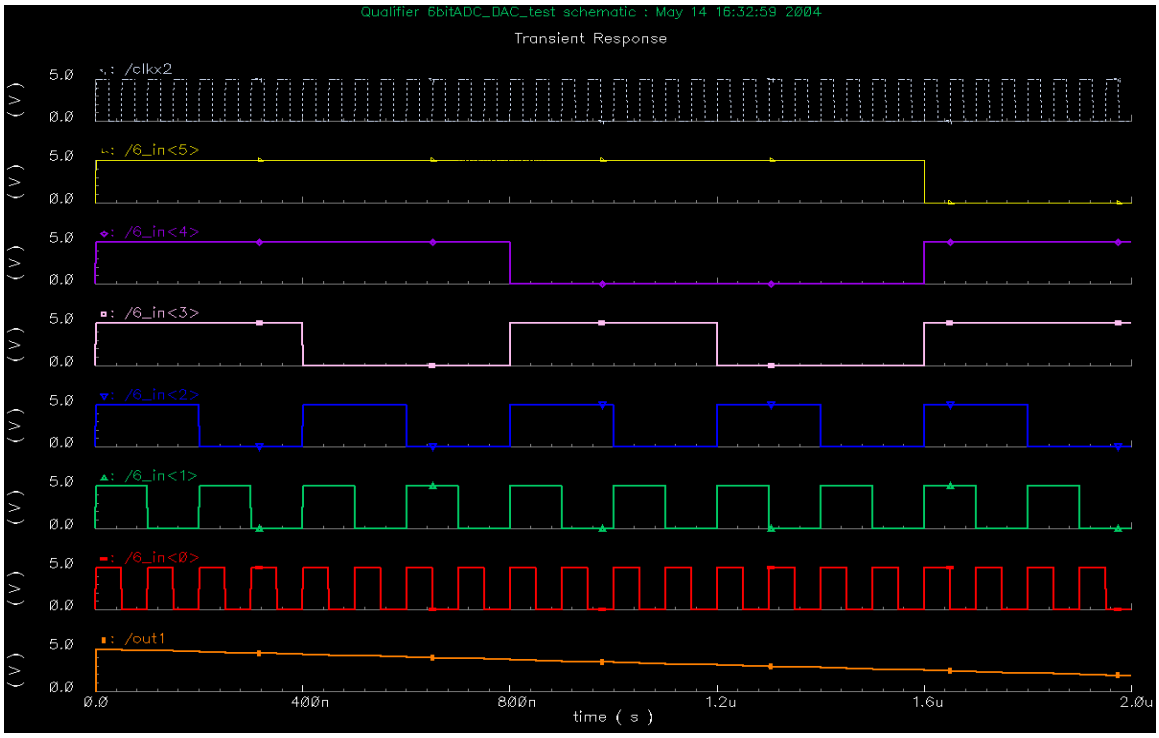
(C-6) Simulation results of a 6-bit analog-to-digital converter (ADC)



(C-7) Simulation results of a phase-frequency detector modeled by Verilog-A



(C-8) Simulation results of a 6-Bit Digital-to-Analog Converter (6-Bit DAC)



Appendix D:

(D-1) Netlist of the Random-Bit Noise Included DAC

```
// Generated for: spectre
// Design library name: Qualifier

simulator lang=spectre
global 0
include "/opt/local/cadence/IC446QSR2/tools.hppa/dfII/samples/artist/ahdlLib/quantity.spectre"
include "/opt/local/cadence/design_kits/NCSU/current/local//models/spectre/nom/allModels.scs"

I39 (NormR_out out_LPF) LPF
I37 (NormR_out out_LPF15) LPF15MHz
I36 (out_7b_DAC NormR_out) Norm_Random
V0 (net49 0) vsource dc=0 type=dc
I34 (_6_in_6 _6_in_5 _6_in_4 _6_in_3 _6_in_2 _6_in_1 _6_in_0 net49 clk \
        out_7b_DAC) DACbit6
R1 (clkx2 0) resistor r=1K
R0 (clk 0) resistor r=1K
V15 (clkx2 0) vsource type=pulse val0=0.0 val1=5 period=50n rise=1n \
        fall=1n width=24n
V17 (clk 0) vsource type=pulse val0=0.0 val1=5 period=100n rise=1n fall=1n \
        width=49n
V7 (_6_in_3 0) vsource type=pulse val0=0.0 val1=5 period=800n rise=1n \
        fall=1n width=399n
V3 (_6_in_2 0) vsource type=pulse val0=0.0 val1=5 period=400n rise=1n \
        fall=1n width=199n
V4 (_6_in_1 0) vsource type=pulse val0=0.0 val1=5 period=200n rise=1n \
        fall=1n width=99n
V5 (_6_in_0 0) vsource type=pulse val0=0.0 val1=5 period=100n rise=1n \
        fall=1n width=49n
V8 (_6_in_4 0) vsource type=pulse val0=0.0 val1=5 period=1.6u rise=1n \
        fall=1n width=799n
V9 (_6_in_5 0) vsource type=pulse val0=0.0 val1=5 period=3.2u rise=1n \
        fall=1n width=1.599u
```

```
V10 (_6_in_6 0) vsource type=pulse val0=0.0 val1=5 period=6.4u rise=1n \
        fall=1n width=3.199u

simulatorOptions options reltol=1e-3 vabstol=1e-6 iabstol=1e-12 temp=27 \
    tnom=27 scalem=1.0 scale=1.0 gmin=1e-12 rforce=1 maxnotes=5 maxwarns=5 \
    digits=5 cols=80 pivrel=1e-3 ckptclock=1800 \
    sensfile="../psf/sens.output"
tran tran stop=4u write="spectre.ic" writefinal="spectre.fc" \
    annotate=status maxiters=5
finalTimeOP info what=oppoint where=rawfile
modelParameter info what=models where=rawfile
element info what=inst where=rawfile
outputParameter info what=output where=rawfile
designParamVals info what=parameters where=rawfile
saveOptions options save=allpub

ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/LPF/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/LPF_15MHz/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/Norm_Random/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/7bitDAC/veriloga/veriloga.va"
```

(D-2) Netlist of the Entire DPLL Design

```
// Generated for: spectre
// Generated on: Feb 23 15:50:42 2005
// Design library name: Qualifier
// Design cell name: Entire_DPLL_Path
// Design view name: schematic
simulator lang=spectre
global 0
include "/opt/local/cadence/IC446QSR2/tools.hppa/dfII/samples/artist/ahdlLib/quantity.spectre"
include "/opt/local/cadence/design_kits/NCSU/current/local//models/spectre/nom/allModels.scs"

// Library name: Qualifier
// Cell name: Divider_xN_Block
// View name: schematic
subckt Divider_xN_Block DividerxN N vco_out_clk
    I16 (vco_out_clk net5 N) DividerN
    I19 (net5 DividerxN) LevelShift_Up_Digital
ends Divider_xN_Block
// End of subcircuit definition.

// Library name: Qualifier
// Cell name: Clock_Block
// View name: schematic
subckt Clock_Block clk vco_out_clk
    I19 (vco_out_clk clk) LevelShift_Up_Digital
ends Clock_Block
// End of subcircuit definition.

// Library name: Qualifier
// Cell name: Clock_x2_Block
// View name: schematic
subckt Clock_x2_Block clkx2 vco_out_clk
    I12 (vco_out_clk net5) Clockx2
    I23 (net5 clkx2) LevelShift_Up_Digital
ends Clock_x2_Block
```

```
// End of subcircuit definition.

// Library name: Qualifier
// Cell name: VCO_fo_10MHz
// View name: schematic
subckt VCO_fo_10MHz vco_in vco_out
    I9 (vco_in vco_out) VCO
ends VCO_fo_10MHz
// End of subcircuit definition.

// Library name: Qualifier
// Cell name: LPF_BW100MHz
// View name: schematic
subckt LPF_BW100MHz in out
    I8 (in out) LPF
ends LPF_BW100MHz
// End of subcircuit definition.

// Library name: Qualifier
// Cell name: 7BitDAC_Block
// View name: schematic
subckt Qualifier_7BitDAC_Block_schematic _7bDAC_in LS_out No_use clkx2
    I11 (DAC_out LS_out) LevelShift
    I5 (_7bDAC_in 0 0 0 0 0 0 0 net7 clkx2 No_use DAC_out) \
        DACbit6_SerialIn
    V2 (net7 0) vsource dc=0 type=dc
ends Qualifier_7BitDAC_Block_schematic
// End of subcircuit definition.

// Library name: Qualifier
// Cell name: PFD_PD_Block
// View name: schematic
subckt PFD_PD_Block out vin vosc_in
    I3 (plus minus TimeCtrl out) PFD_Counter
    I4 (vin vosc_in plus minus) PFDUpTrigger
    V6 (TimeCtrl 0) vsource type=pulse val0=0.0 val1=5 period=7.8125n \
```

```
        delay=0 rise=1n fall=1n width=2.90625n
    V1 (net14 0) vsource type=pulse val0=0.0 val1=5 period=22u delay=2n \
        rise=1n fall=1n width=10u
    R1 (net14 0) resistor r=1K
ends PFD_PD_Block
// End of subcircuit definition.

// Library name: Qualifier
// Cell name: Start_Block
// View name: schematic
subckt Start_Block Dividerx10 signal vin2PFD vosc2PFD
    V7 (net11 0) vsource type=pulse val0=0.0 val1=5 period=100.1 \
        delay=900.0n rise=1n fall=1n width=100
    I20 (net11 signal vin2PFD) and_gate
    I21 (net11 Dividerx10 vosc2PFD) and_gate
ends Start_Block
// End of subcircuit definition.

// Library name: Qualifier
// Cell name: 3TapFIR_w_Noclk_Of_multiplier_and_adder
// View name: schematic
subckt Qualifier_3TapFIR_w_Noclk_Of_multiplier_and_adder_schematic \
        _3Tap_FIR_out clkx2 h\[n\-1\] h\[n\-2\] h\[n\] in
    I5 (in\[n\-1\] clkx2 in\[n\-2\]) dlay
    I2 (in clkx2 in\[n\-1\]) dlay
    I4 (h\[n\-2\] in\[n\-2\] in\[n\-2\]h\[n\-2\]) multiplier_Noclk
    I3 (h\[n\-1\] in\[n\-1\] in\[n\-1\]h\[n\-1\]) multiplier_Noclk
    I1 (h\[n\] in in\[n\]h\[n\]) multiplier_Noclk
    I0 (in\[n\]h\[n\] in\[n\-1\]h\[n\-1\] in\[n\-2\]h\[n\-2\] \
        _3Tap_FIR_out) adder_Noclk
ends Qualifier_3TapFIR_w_Noclk_Of_multiplier_and_adder_schematic
// End of subcircuit definition.

// Library name: Qualifier
// Cell name: Entire_DPLL_Path
// View name: schematic
```

```
I41 (Dividerx10 N vco_out) Divider_xN_Block
I40 (clk vco_out) Clock_Block
I39 (clkx2 vco_out) Clock_x2_Block
I38 (LPF_out vco_out) VCO_fo_10MHz
I37 (DAC_out LPF_out) LPF_BW100MHz
I36 (PFD_out DAC_out No_use clkx2) Qualifier_7BitDAC_Block_schematic
I35 (PFD_out xin xosc) PFD_PD_Block
I34 (Dividerx10 FIR_out xin xosc) Start_Block
V16 (N 0) vsource dc=10 type=dc
V12 (h\[n\-2\] 0) vsource dc=20.00m type=dc
V14 (h\[n\-1\] 0) vsource dc=980.00m type=dc
V13 (h\[n\] 0) vsource dc=20.00m type=dc
I10 (FIR_out clkx2 h\[n\-1\] h\[n\-2\] h\[n\] Latch_out) \
        Qualifier_3TapFIR_w_Noclk_Of_multiplier_and_adder_schematic
V1 (in 0) vsource dc=2.5 type=sine ampl=2.5 freq=1M
I18 (No_5 No_4 No_3 No_2 No_1 No_0 in clk ADC_out) SerialADCbit6
I13 (ADC_out clk Latch_out) Latch10

simulatorOptions options reltol=1e-3 vabstol=1e-6 iabstol=1e-12 temp=27 \
    tnom=27 scalem=1.0 scale=1.0 gmin=1e-12 rforce=1 maxnotes=5 maxwarns=5 \
    digits=5 cols=80 pivrel=1e-3 ckptclock=1800 \
    sensfile="../psf/sens.output"
tran tran stop=5u write="spectre.ic" writefinal="spectre.fc" \
    annotate=status maxiters=5
finalTimeOP info what=oppoint where=rawfile
modelParameter info what=models where=rawfile
element info what=inst where=rawfile
outputParameter info what=output where=rawfile
designParamVals info what=parameters where=rawfile
saveOptions options save=allpub

ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/DividerN/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/LevelShift_Up_Digital/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/Clockx2/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/VCO/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/LPF/veriloga/veriloga.va"
```

```
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/LevelShift/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/Serial_7bitDAC/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/PFD_Counter/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/PFDUpTrigger/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/AND2/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/delay/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/multiplier_Noclk/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/adder_Noclk/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/Serial_6bitADC/veriloga/veriloga.va"
ahdl_include "/rcc4/student/wangc/cadence/NCSU/Qualifier/1bit_10_Latch/veriloga/veriloga.va"
```