

Behavioral Simulation of an Amplitude Resolution Improvement for an RF-DAC Employing PWM scheme

Min Park

December 2010

Copyright © 2010 by Min Park
All rights reserved.

Table of Contents

| | |
|--|----|
| Setup | 2 |
| Introduction..... | 5 |
| Centered PWM Polar Transmitter Simulations Using Sue2 and CppSimView | 8 |
| A. Opening Sue2 Schematics..... | 8 |
| B. Running the CppSim Simulation | 13 |
| C. Plotting the Time Domain Results..... | 15 |
| Post-Processing the Simulation Results Using a MATLAB Script..... | 17 |
| Centered PWM Method with Pulse Position Offset | 18 |
| A. Modifying the Testbench Schematic | 18 |
| B. Post-Processing the Simulation Results..... | 19 |
| Comparison of Centered PWM with and without LUT..... | 20 |
| A. Opening Sue2 Schematics of a Testbench..... | 20 |
| B. Post-Processing the Simulation Results..... | 21 |
| CppSim Simulations for Non-Centered PWM | 22 |
| A. Opening Sue2 Schematics of a Testbench..... | 22 |
| B. Post-Processing the Simulation Resultsng a Testbench..... | 25 |
| Comparison of Simplified LUT and Full-Size LUT for Non-Centered PWM..... | 26 |
| A. Opening Sue2 Schematics of a Testbench..... | 26 |
| B. Post-Processing the Simulation Results..... | 28 |
| Non-Centered PWM with Different Phase Resolutions | 30 |
| A. Opening Sue2 Schematics of a Testbench..... | 31 |
| B. Post-Processing the Simulation Results..... | 31 |
| Conclusion | 33 |
| Reference | 33 |

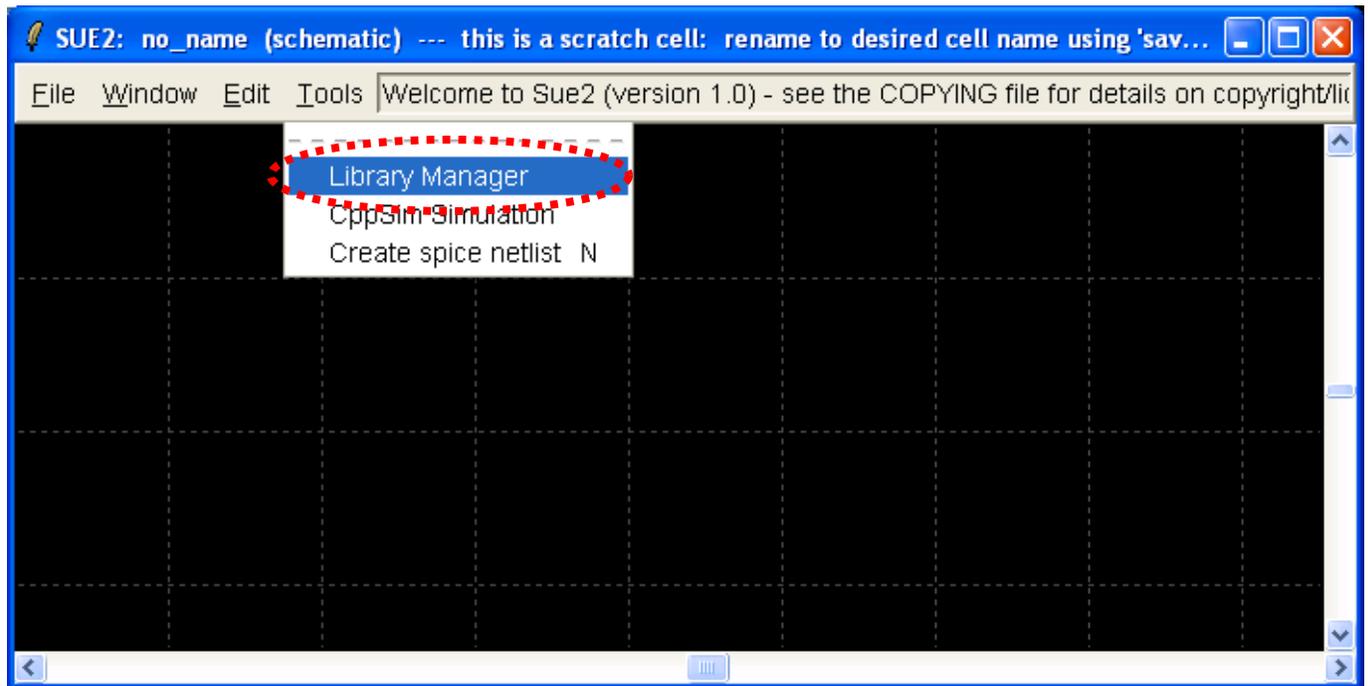
Setup

Download and install the CppSim Version 4 package (i.e., download and run the self-extracting file named **setup_cppsim4.exe**) located at:

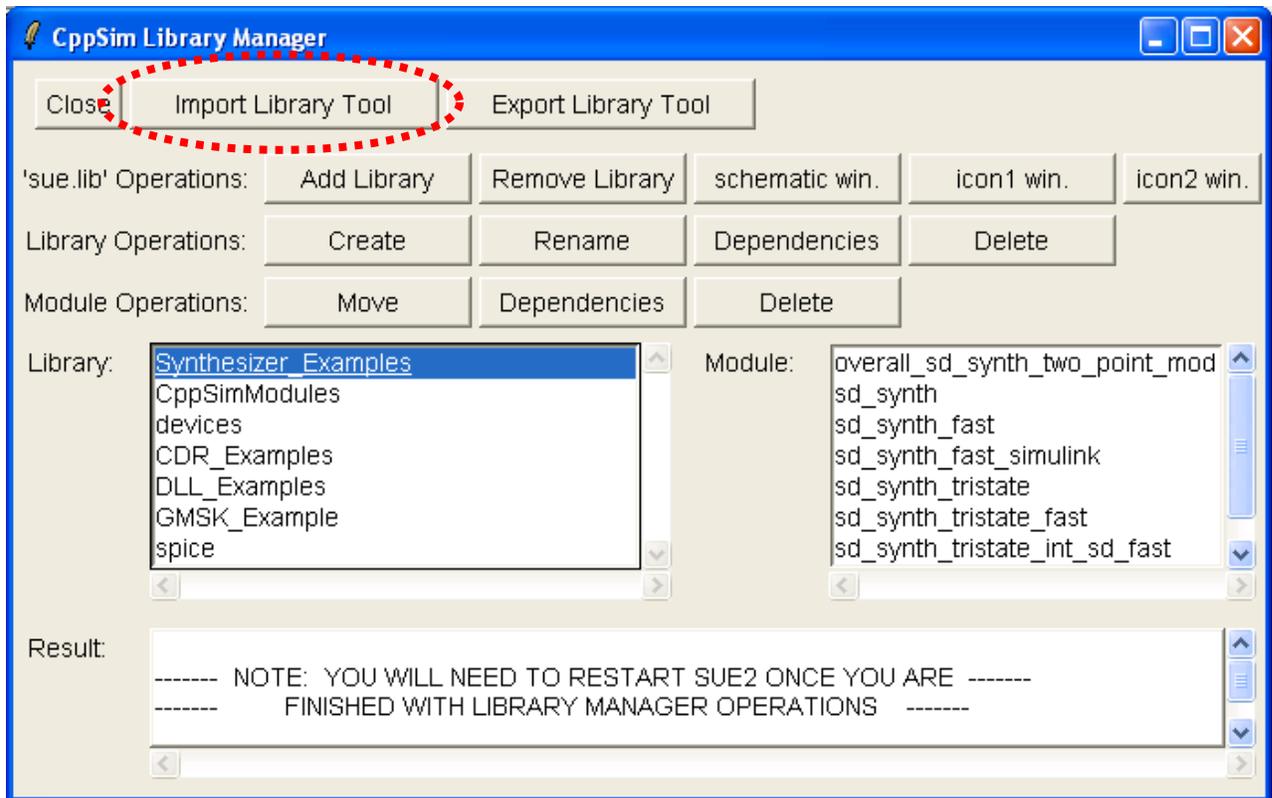
<http://www.cppsim.com>

Upon completion of the installation, you will see icons on the Windows desktop corresponding to the PLL Design Assistant, CppSimView, GTKWave, and Sue2. Please read the “**CppSim (Version 4) Primer**” document, which is also at the same web address, to become acquainted with CppSim and its various components.

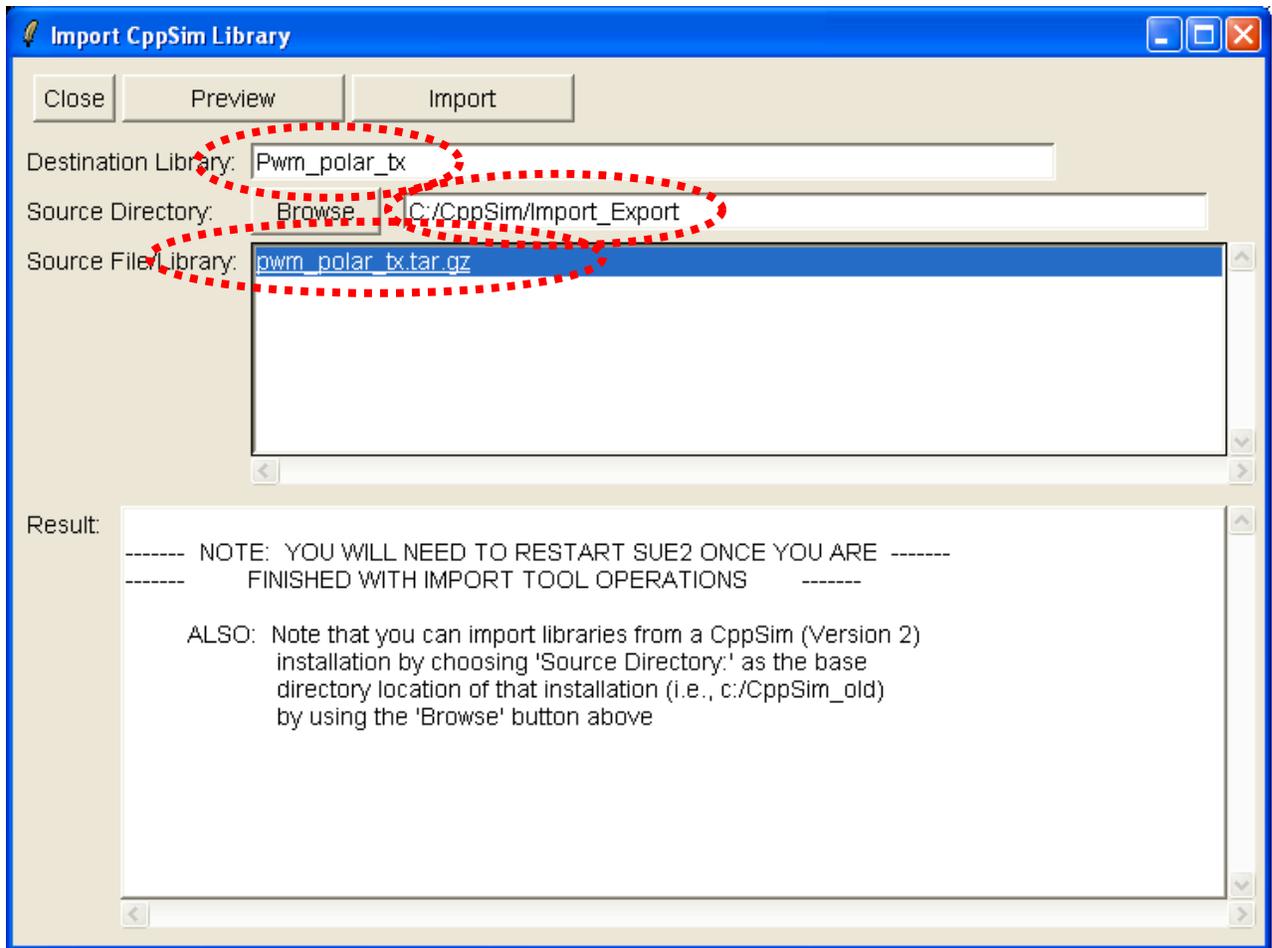
To run this tutorial, you will also need to download the file **pwm_polar_tx.tar.gz** available at <http://www.cppsim.com>, and place it in the **Import_Export** directory of CppSim (assumed to be **c:/CppSim/Import_Export**). Once you do so, start up **Sue2** by clicking on its icon, and then click on **Tools->Library Manager** as shown in the figure below.



In the **CppSim Library Manager** window that appears, click on the **Import Library Tool** button as shown in the figure below.



In the **Import CppSim Library** window that appears, change the **Destination Library** to **Pwm_polar_tx**, click on the **Source File/Library** labeled as **pwm_polar_tx.tar.gz**, and then press the **Import** button as shown in the figure below. Note that if **pwm_polar_tx.tar.gz** does not appear as an option in the **Source File/Library** selection listbox, then you need to place this file (downloaded from <http://www.cppsim.com>) in the **c:/CppSim/Import_Export** directory.



Once you have completed the above steps, restart **Sue2** as directed in the above figure.

Introduction

Nowadays, most wireless applications require extensive digital functionality. Ideally, integrating RF circuits with digital logic is desired for lower cost, form factor, and power dissipation. Some RF transmitters presented in recent publications have suggested using an RF digital-to-analog converter (RF-DAC) for easier system-on-a-chip (SoC) implementation [1]–[4]. Among them, the commercial single-chip Global System for Mobile Communications (GSM)/Enhanced Data rates for GSM Evolution (EDGE) transceiver in [1] and [2] is unique in that it uses an array of unit-weighted transistor switches to control the output RF amplitude instead of using a traditional current-source-based digital-to-analog converter (DAC) structure.

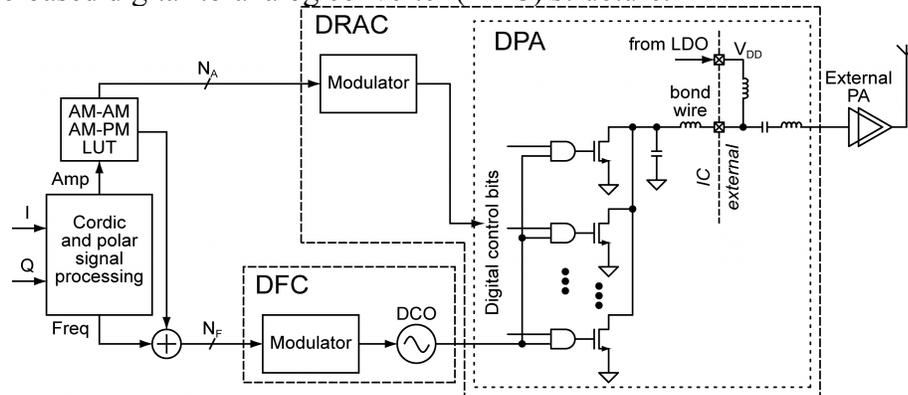


Figure 1 Polar transmitter based on a DCO and a DPA circuits.

Figure 1 illustrates the polar transmitter introduced in [1] and [2]. The in-phase/quadrature baseband data are converted into amplitude and phase/frequency polar components. The frequency signal is fed into the digitally controlled oscillator (DCO)-based N_F -bit digital-to-frequency converter. The amplitude signal drives the N_A -bit digital-to-RF-amplitude converter (DRAC), which includes a digitally controlled power amplifier (DPA). The amplitude of the output RF carrier signal is controlled by the number of engaged switching devices. The DRAC controls the envelope of the phase-modulated RF carrier, and thus, it is considered an RF-DAC.

The approach in [1] and [2] proved that the architecture in Figure 1 is feasible for SoC, and it meets all GSM and EDGE specifications. However, the resolution of the amplitude path is limited by lithography and RF mismatches of the switching devices in the DPA, and consequently, the polar transmitter has little margin in the far-out (i.e., the associated receiving (RX) band) noise limit of the surface acoustic wave(SAW)-less operation for EDGE.

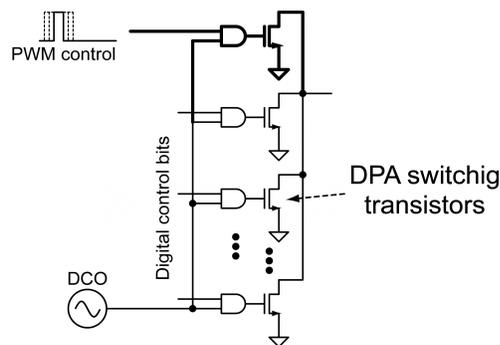


Figure 2 Amplitude resolution improvement by adding a PWM-driven transistor.

The amplitude resolution of the original DPA from Figure 1 can improve simply by adding one additional switching device that is driven by a PWM signal as shown in Figure 2. The resolution

improves by turning on the switching devices for only a short time interval within the positive half-cycle of the RF period. The RF output amplitude will be controlled by the time interval, and the resolution is determined by the time precision of the turning-on signal. The amplitude resolution of the DPA in Figure 1 is now limited by the time resolution of the PWM. In modern CMOS processes, the switching time gets improved by 0.7x per node; hence, achieving higher resolution in the time-domain is easier than in the voltage/current domain [5]. The output amplitude of a PWM signal at the frequency of interest, however, is incorrect if the pulse width is chosen in a straightforward way such that the DC amplitude of the PWM signal is correct [8], [9]. As a result, this inaccurate RF output level at the carrier frequency turns out to limit the resolution improvement.

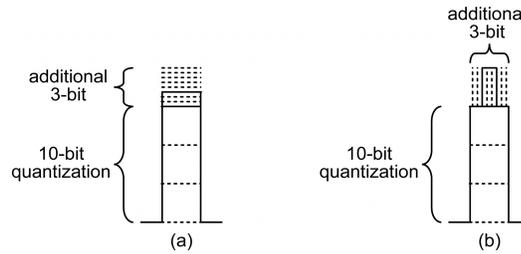


Figure 3 Amplitude resolution improvement by (a) horizontal slicing (b) vertical slicing.

To explain this issue, Figure 3 illustrates two different quantization methods for adding an extra 3-bit resolution in either a voltage or current signal. Intuitively, both signals from Figure 3(a) and (b) are the same in terms of power because the total area of the signals are the same. However, they are equivalent only at DC. The powers of those signals are not the same at the carrier frequency [6], [7]. Therefore, the signal in Figure 3(b) creates an incorrect RF signal even though it creates an accurate DC signal. An inaccurate RF signal from an RF-DAC leads to higher quantization noise. Therefore, resolution improvement by PWM is severely impaired if the pulse width is chosen in a straightforward way [6], [7]. The position of a PWM signal also affects the accuracy of an RF signal driven by PWM. When the pulse position is not *at the center*, the phase of the RF signal is distorted [7].

The inaccurate RF signal from an RF-DAC driven by a *centered PWM* or a *non-centered PWM* signal can be corrected by predistortion digital *Look-Up-Tables* (LUTs) which can be easily implemented in digital CMOS processes [6], [7].

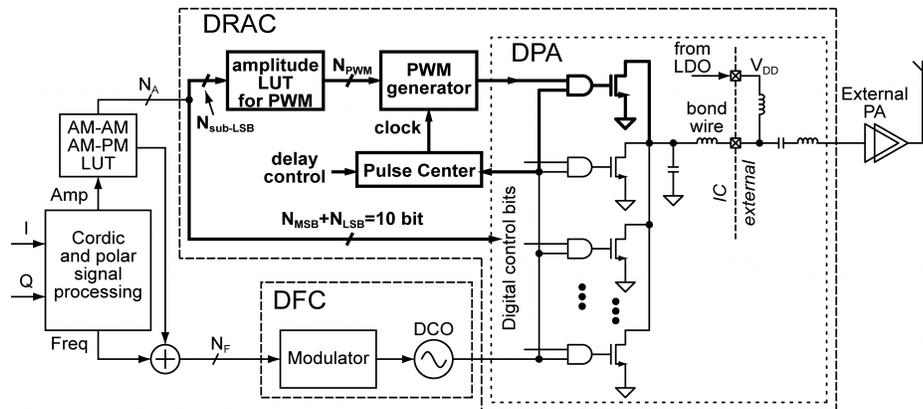


Figure 4 Polar transmitter employing centered PWM to increase the amplitude resolution.

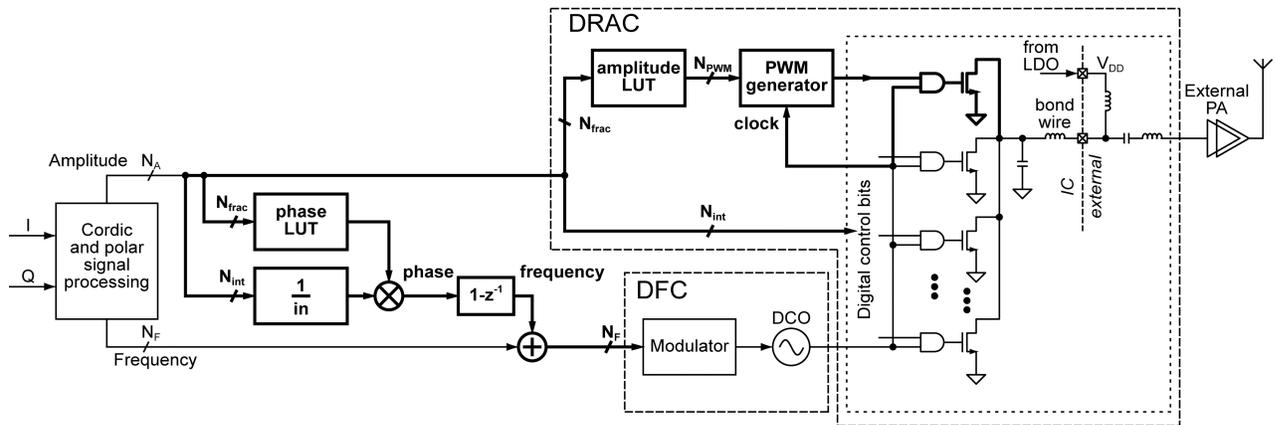


Figure 5 Polar transmitter employing non-centered PWM to increase the amplitude resolution.

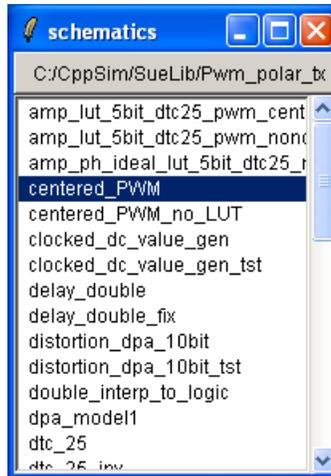
Figure 4 illustrates the proposed polar transmitter architecture with centered PWM introduced in [6]. Figure 5 illustrates the one with non-centered PWM introduced in [7]. In this tutorial, we will see how these proposed transmitter architecture can be modeled in CppSim, and run the simulations to verify the performance of the proposed PWM schemes. We will also use MATLAB post-processing scripts to reproduce the simulation results in [6] and [7].

Centered PWM Polar Transmitter Simulations Using Sue2 and CppSimView

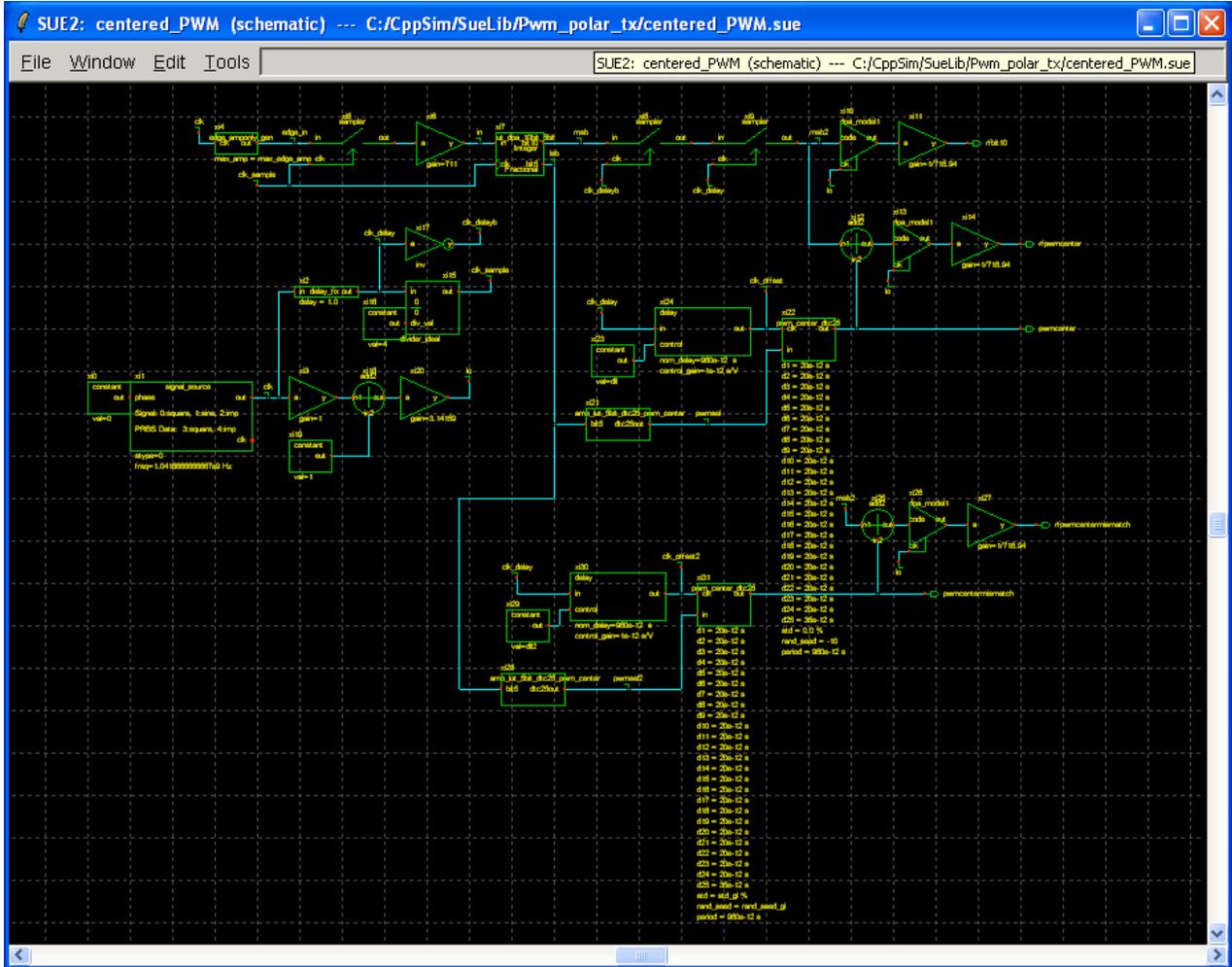
In this section, the user will be guided through basic tasks such as opening the PWM polar transmitter example employing centered PWM within the Sue2 schematic editor and running basic CppSim simulations.

A. Opening Sue2 Schematics

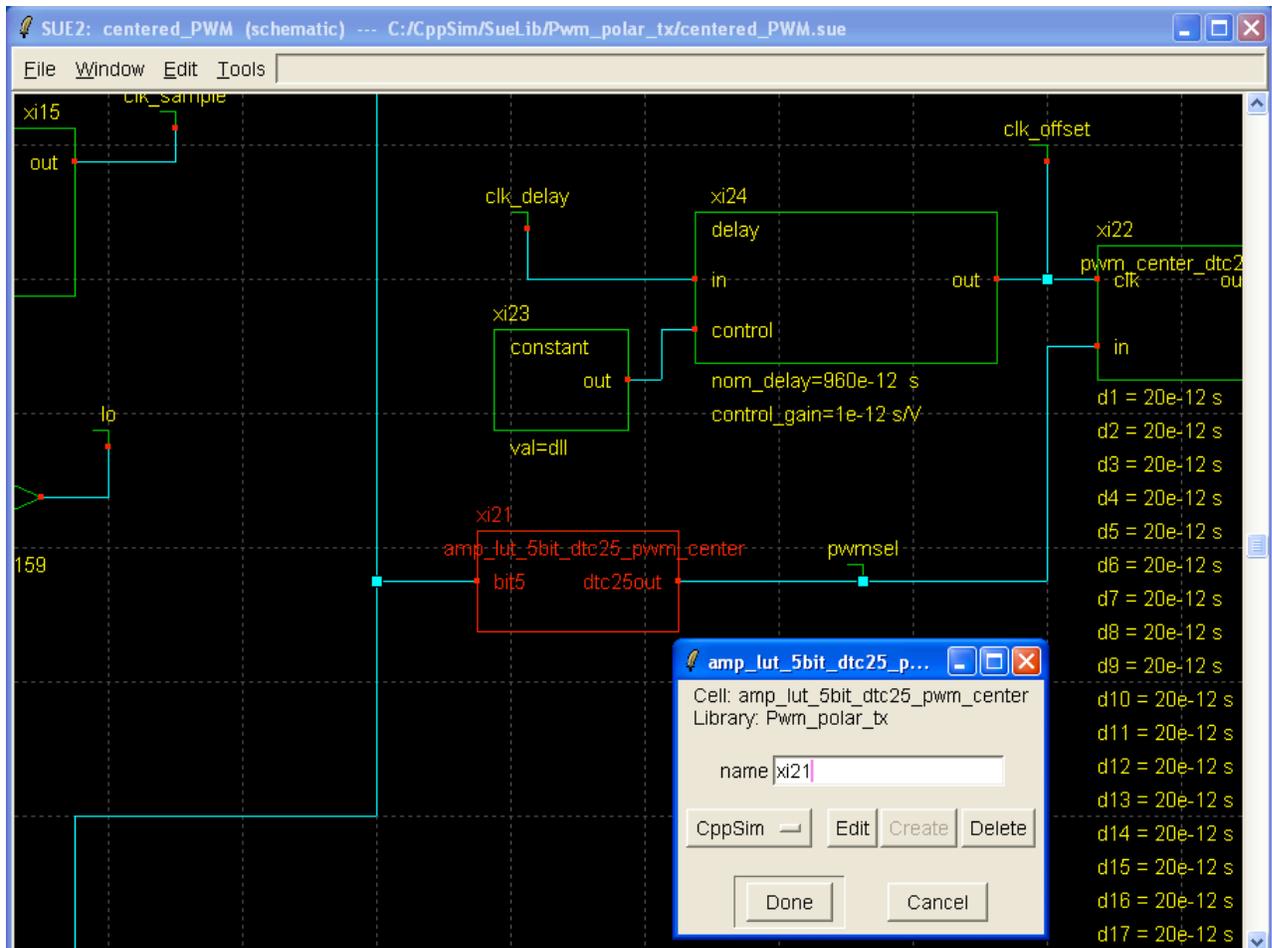
- Click on the Sue2 icon to start Sue2, and then select the **Pwm_polar_tx** library from the **schematic listbox**. The **schematic listbox** should now look as follows:



- Click on the **centered_PWM** schematic. The schematic window should now display the testbench of the EDGE transmitter model employing the centered PWM scheme as shown below:



- The baseband EDGE amplitude signal is generated by **edge_amponly_gen** cell. The main purpose of the PWM scheme introduced here is to improve the amplitude resolution of a digital polar transmitter. Therefore, only amplitude signal path is modeled in this testbench. The amplitude baseband signal goes into the first amplitude LUT, **lut_dpa_10bit_5bit**. This LUT compensates for the DPA's amplitude distortion, and it is already included in the original polar transmitter system in [1]. The output is 15-bit digital signal, and the 10-bit integer part drives directly a 10-bit DPA, and the 5-bit fractional part generates the centered PWM signal.
- The output DPA is modeled by **dpa_model1** cell. The 10-bit input corresponds to the digital input of the DPA. The clock input of **dpa_model1** upconverts the 10-bit digital input signal. The DPA's amplitude non-linearity is also modeled in **dpa_model1** cell.
- **rfbit10** is the upconverted EDGE signal from the 10-bit DPA. We will compare the quantization noise of **rfbit10** with that of the centered PWM signal.
- The 5-bit fractional part of the baseband signal goes into the LUT for centered PWM, **amp_lut_5bit_dtc25_pwm_center**, which is a module with coded functionality. Double click on **xi21** cell, then a new window pops up as shown below:



- Click on **Edit** button. Then, the following emacs window pops up:

```
C:/CppSim/CadenceLib/Pwm_polar_tx/amp_lut_5bit_dtc25_pwm_center/cppsim/text.txt
File Edit Options Buffers Tools Help
module: amp_lut_5bit_dtc25_pwm_center
description: Digital Predistortion for PWM center 5bit(Fractional part) input.
            For dtc25
            0 <= bit6 < 32
            Coded by Min Park 03/16/09
            (converted 10/08/10)

parameters:
inputs:  double bit5
outputs: double dtc25out
classes:
static_variables: double err int nn double err_dis[12]
                  double tau double v_eq double lut[32]
                  double err_q5 int ind int mnn double minerrdiff

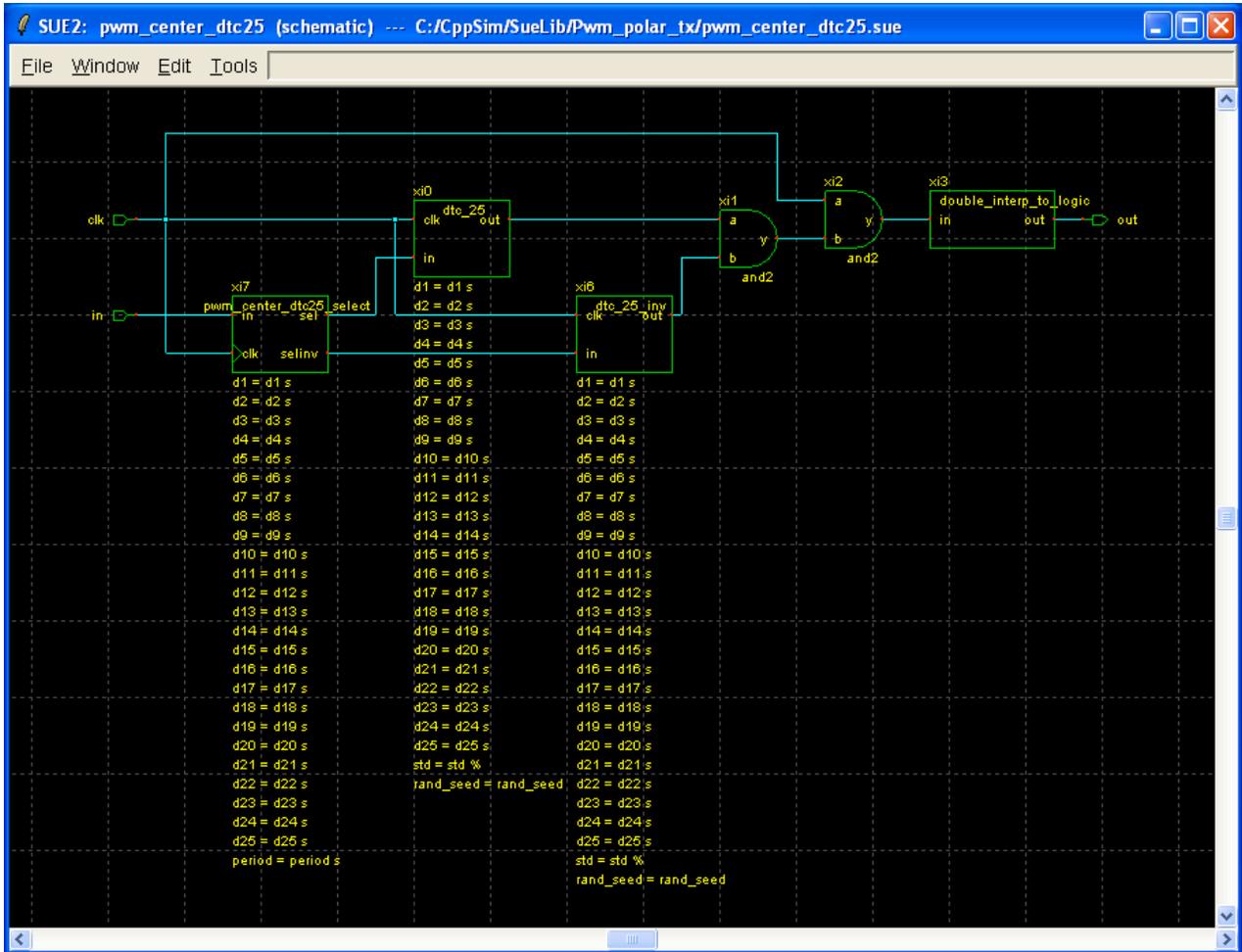
init:

    tau = 40*12/2;
    for( nn=0; nn<12 ; nn++){
        v_eq = cos( 3.13159 * tau / 480);
        printf("v_eq = %e\n", v_eq);
        err_dis[nn] = v_eq;
        // Assuming the maximum q error is 1.
        tau -= 20;
    }

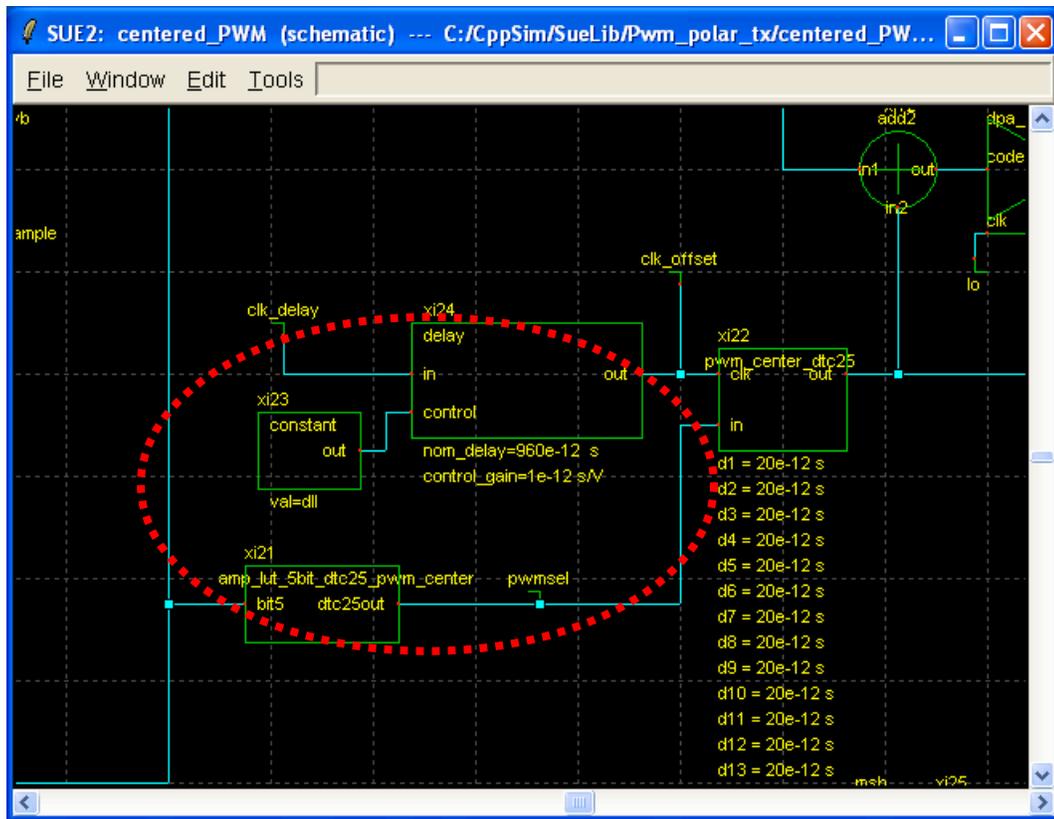
    err_q5 = 0.0;
    for( nn=0; nn<32 ; nn++){
        ind = 0;
        minerrdiff = 1e6;
        for( mnn=0 ; mnn < 12; mnn++){
            if( fabs( err_q5 - err_dis[mnn]) < minerrdiff ){
                minerrdiff = fabs( err_q5 - err_dis[mnn] );
                ind = mnn;
            }
        }
        lut[nn] = (double)ind;
        printf("lut[%d] = %e\n",nn,lut[nn]);
        err_q5 += 1.0/32.0;
        // Assuming the maximum q error is 1.
    }
}

--\-- text.txt (Text)--L1--Top--
For information about the GNU Project and its goals, type C-h C-p.
```

- The module description shown above is based on the amplitude LUT formula explained in [6].
- The output of the LUT goes into **pwm_center_dtc25** which generates a centered PWM based on 25-stage delay cells. After clicking on **pwm_center_dtc25**, press **e** to descend into its schematic. You will see the schematic below. It is also briefly explained in [6] and [7]. You may now press **Ctrl+e** to return to the testbench top level.



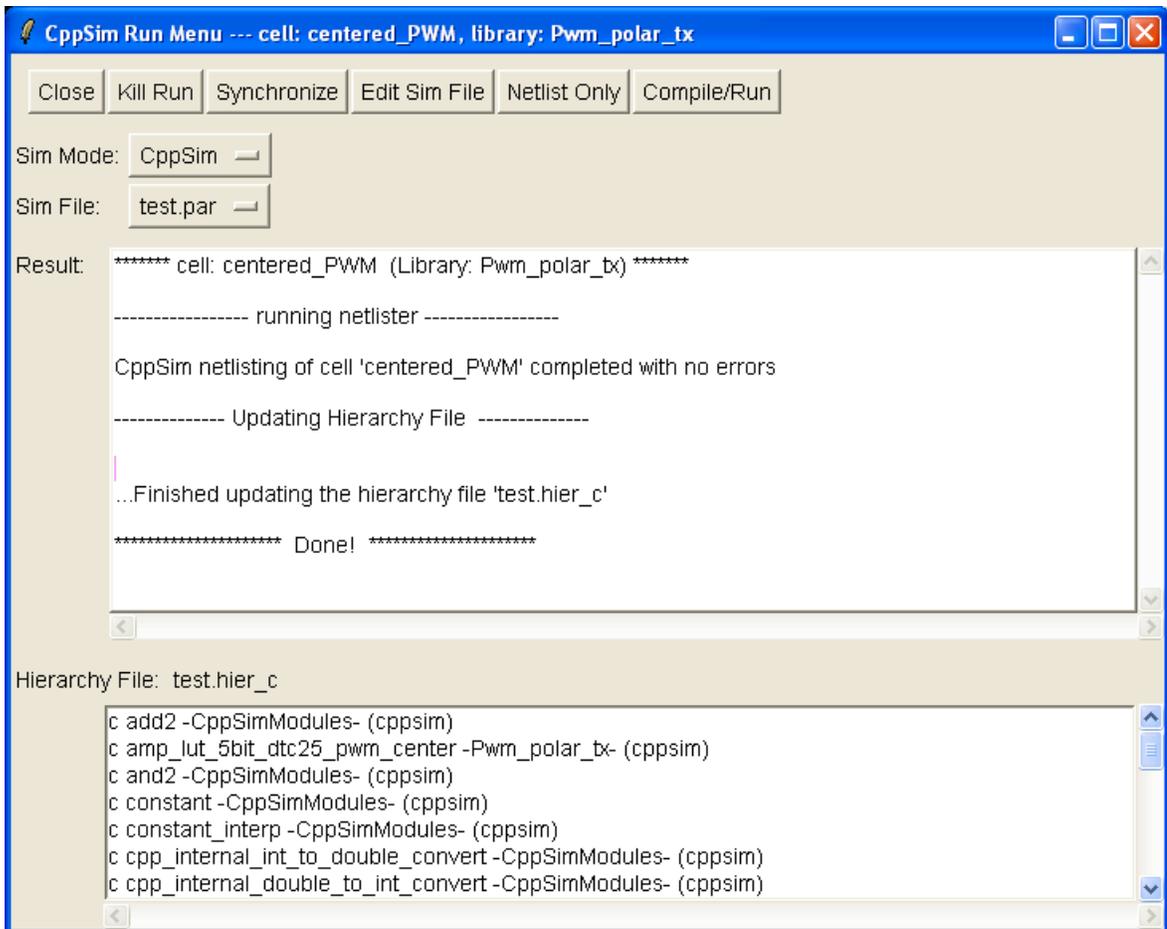
- The output of the LUT goes into `pwm_center_dtc25` which generates a centered PWM based on 25-stage delay cells. After clicking on `pwm_center_dtc25`, press `e` to descend into its schematic. You will see the schematic below. It is also briefly explained in [6] and [7]. You may now press `Ctrl+e` to return to the testbench top level.
- The pulse position of the PWM signal created by `pwm_center_dtc25` is referenced to the input clock of the module. In the testbench, the input clock is controlled by the delay element as shown below. By adjusting the delay (by global parameter `dll`), we can precisely control the pulse position of the centered PWM signal. This block is introduced as **Pulse Center** block in [6] and [7].



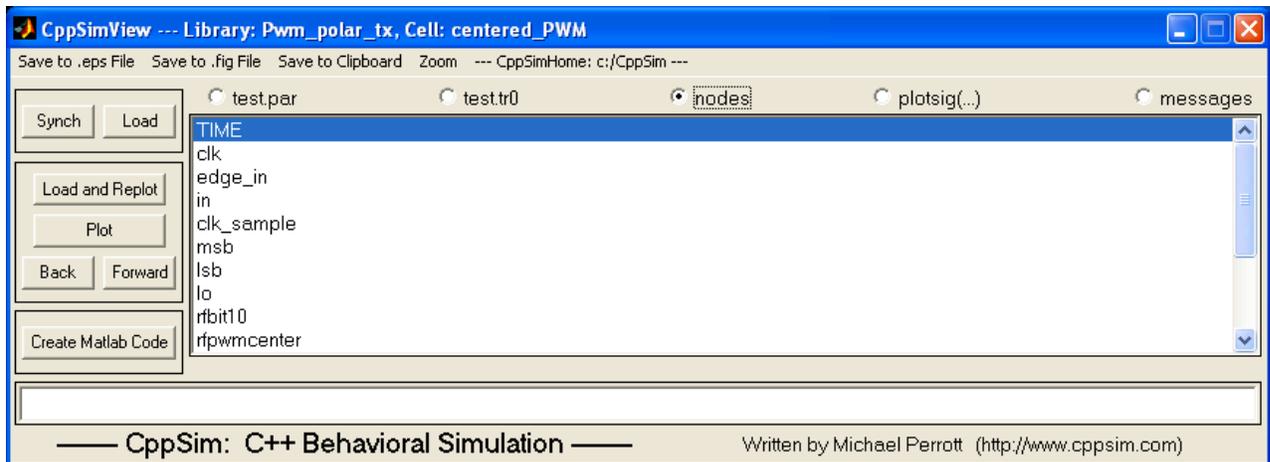
- The generated centered PWM signal is added to the original 10-bit DPA signal creating **rfpwmcenter**. In other words, the centered PWM signal is added on top of the 10-bit amplitude signal such that the resolution of the output RF signal improves beyond 10-bit.
- **rfpwmcenter** will be compared with **rfbit10**, which is the output of the original 10-bit DPA output, in terms of quantization noise.
- **rfpwmcentermismatch** signal is the same as **rfpwmcenter** except that it also includes the delay mismatch effect of the centered PWM generator.

B. Running the CppSim Simulation

- Within Sue2, Click on **Tools** and then **CppSim Simulation**. The **CppSim Simulation Window** should appear as shown below.



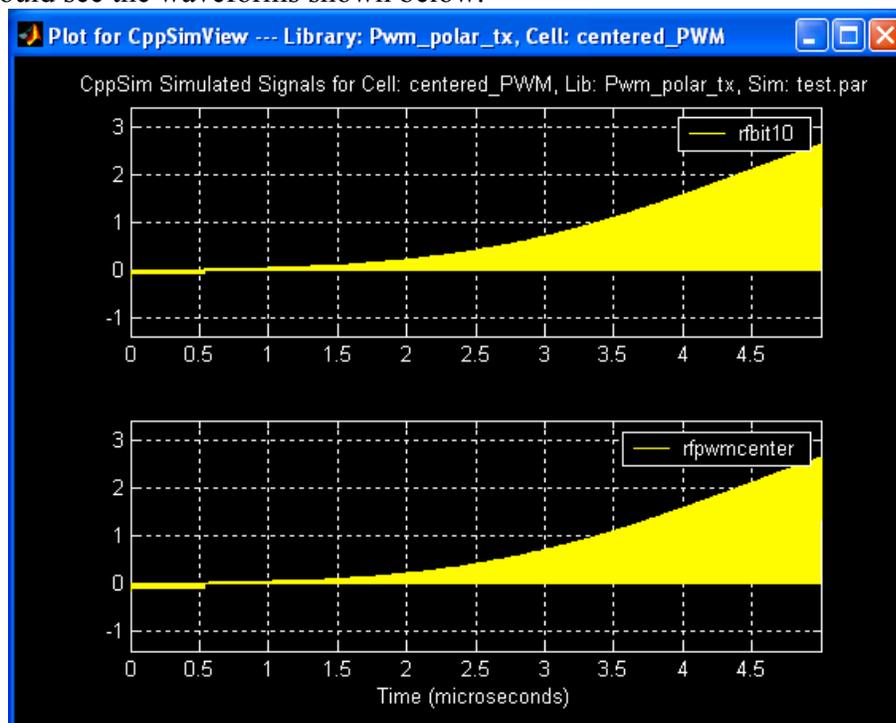
- Double click on the **Edit Sim File** button. An emacs window should appear that indicates that the number of simulation steps, **num_sim_steps**, is set to $1e7$ and the timestep, **Ts**, is set to $10e-12$. We need the finer time step to model the precise PWM waveform including delay mismatch. However, CppSim employs the signal discretization technique introduced in [13] such that it can deal with femto second of time resolution in the simulation even with 10ps of time step.
- Also note that the simulation will save two output files, **test.tr0** and **testfft.tr0**. **test.tr0** will store the first $5e5$ samples only, so that we can look at the time-domain waveforms quickly. **testfft.tr0** will store the entire simulation results except for the first $1e4$ samples. We will use the saved waveforms in testfft.tr0 to see the spectrum of the output signals. The first $1e4$ samples are not saved in order to remove the initial transient effect.
- Click on the **Compile/Run** button to launch the simulation. Note that it can take 10 minutes or more to finish the simulation depending on the system CppSim is running on.
- Click on the **CppSimView** icon on your desktop to start the CppSim viewer. (You don't have to wait until the simulation is completed!)
- Click on the **No Output File** radio button and select **test.tr0** as the output file.
- Click on the **No Nodes** radio button to load in the simulated signals. CppSimView should now appear as shown below.



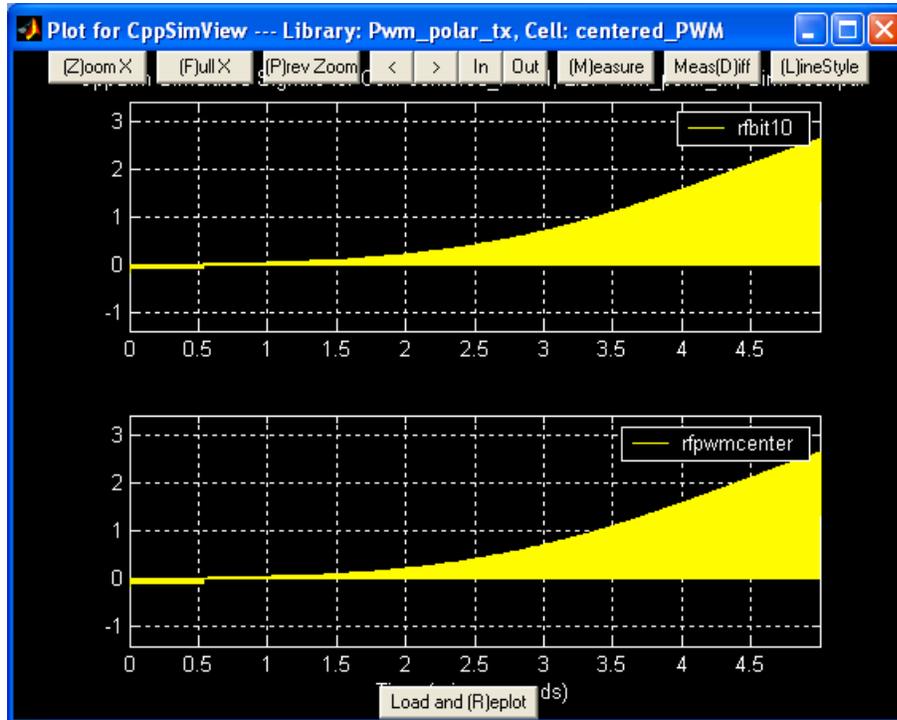
C. Plotting the Time Domain Results

Given the above simulation, we will now take a look at the output signals.

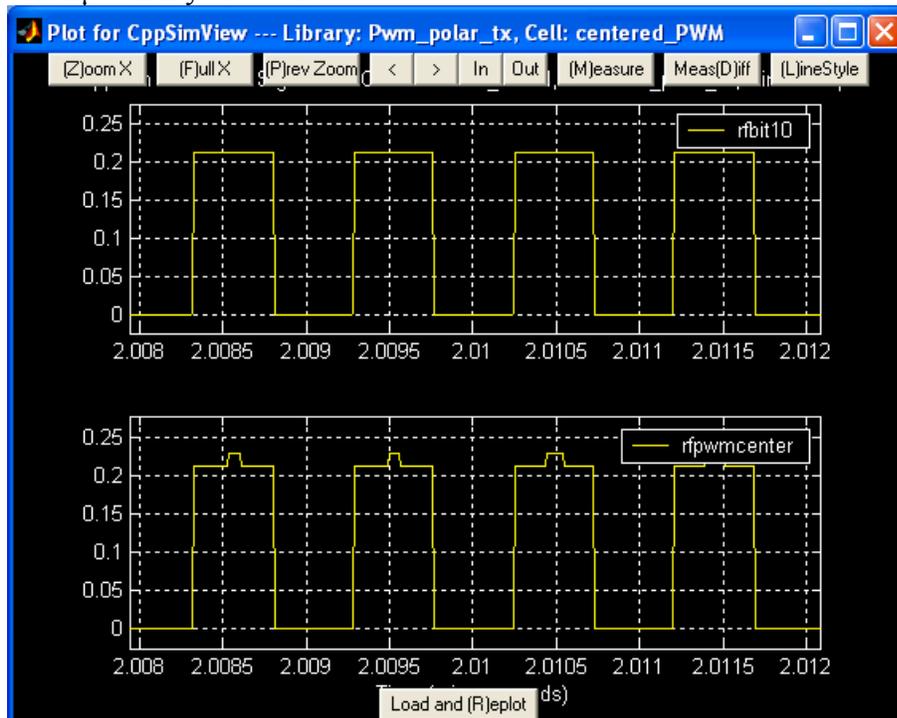
- In the CppSimView window, double-click on signals **rfsbit10** and **rfpwmcenter**. You should see the waveforms shown below.



- In the CppSimView window, click on **Zoom**. The waveform window will include buttons as shown below.



- Click on the **Zoom** button in the waveform window, and then zoom in several times around $2\mu\text{s}$ until you observe the waveform below:



- Notice that `rfpwmcenter` signal shows centered PWM signals on top of the original 10-bit amplitude output.

Post-Processing the Simulation Results Using a MATLAB Script

While CppSim has a few built-in functions for frequency domain analysis, in this case we find it convenient to use a customized script in MATLAB. The provided scripts for the each testbench show the spectrum of the output signal such that we can compare the quantization noise of different output signals.

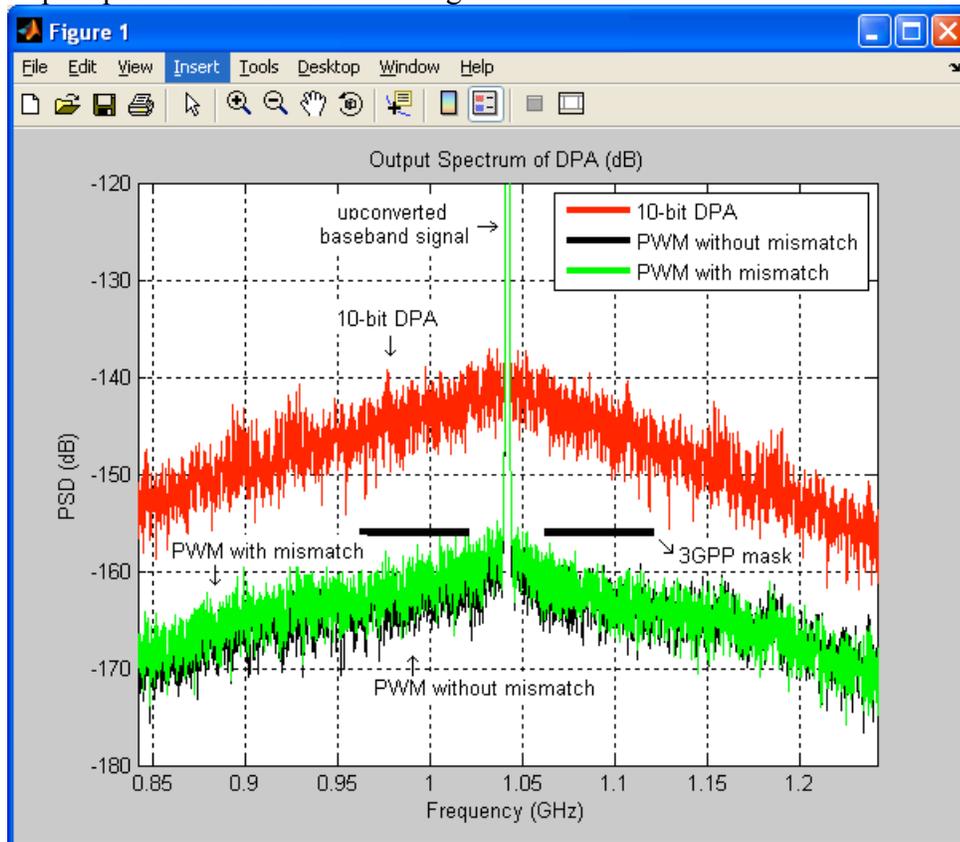
- Open MATLAB and set the working directory to the simulation results folder. For example:

```
>> cd C:\CppSim\SimRuns\Pwm_polar_tx\centered_PWM
```

- Run the post-processing script **centered_PWM.m** by typing post into the command window

```
>> centered_PWM
```

- Since the simulation result, **testfft.tr0**, is a large file, it can take 4 minutes or more to see the final results from the post-processing script depending on the system MATLAB is running on. After MATLAB completes processing the data, you should see the output spectrum for different RF signals from the testbench as shown below:



- The spectrum in red shows the quantization noise of the original 10-bit DPA output. The spectrum in black and green show the noise skirts of the DPA employing centered PWM scheme without and with delay mismatch of the PWM signals.

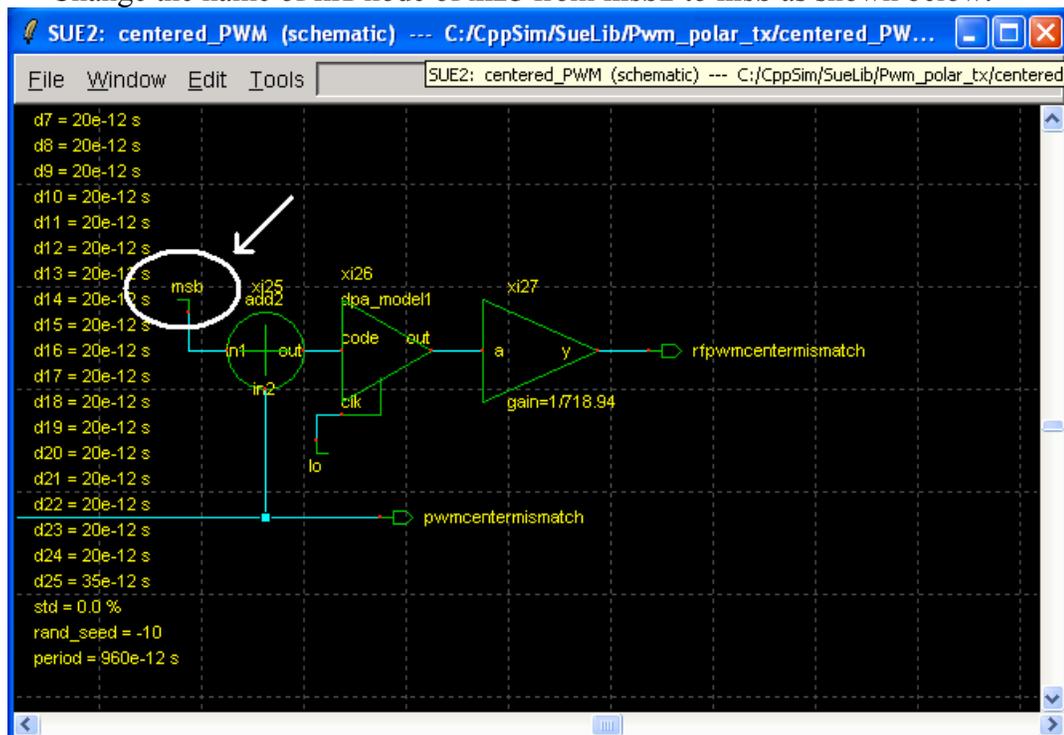
- We observe that the RF output employing the centered PWM scheme improves the quantization noise by about 18dB compared with the original DPA meeting 3GPP spectral mask.

Centered PWM Method with Pulse Position Offset

As mentioned in [6] and [7], the pulse position of the PWM plays an important role. The **delay** blocks, **xi24** and **xi30**, in **centered_PWM** testbench are employed to make the pulse positions center. However, the quantization noise increases when the pulse position of the PWM signal is not at the center. In this section, we will see the effect of the pulse position offset by modifying **centered_PWM** testbench and rerun the simulation.

A. Modifying the Testbench Schematic

- Open the previous testbench by clicking on **centered_PWM** schematic from the **schematic listbox**.
- Change the name of **in1** node of **xi25** from **msb2** to **msb** as shown below:



- Open **CppSim Simulation**, and click on **Edit Sim File** button. Change a global parameter **dll2** value as shown below:

```

C:\CppSim\SimRuns\Pwm_polar_tx\centered_PWM\test.par
File Edit Options Buffers Tools Help
clk_delay

output: testfft start_sample=1e4
probe: rfbt10 rfpwmcenter rfpwmcentermismatch

////////////////////////////////////
// Note: Items below can be kept blank if desired
////////////////////////////////////

// Values for global parameters used in schematic
// Example: global_param: in_g1=92.1 delta_g1=0.6 step_time_g1=100e3*Ts
global_param: max_edge_amp = 1 dll=-14 dll12=-14+18 rand_seed_g1=-10 std_g1=10

// dll=-14 dll12=-14-17.87
// +17.86ps offset is worse than -17.87ps. msb2 should be changed to msb
// for the second PWM generator to use +17.87ps offset.

// Rerun simulation with different global parameter values
// Example: alter: in_g1 = 90:2:98

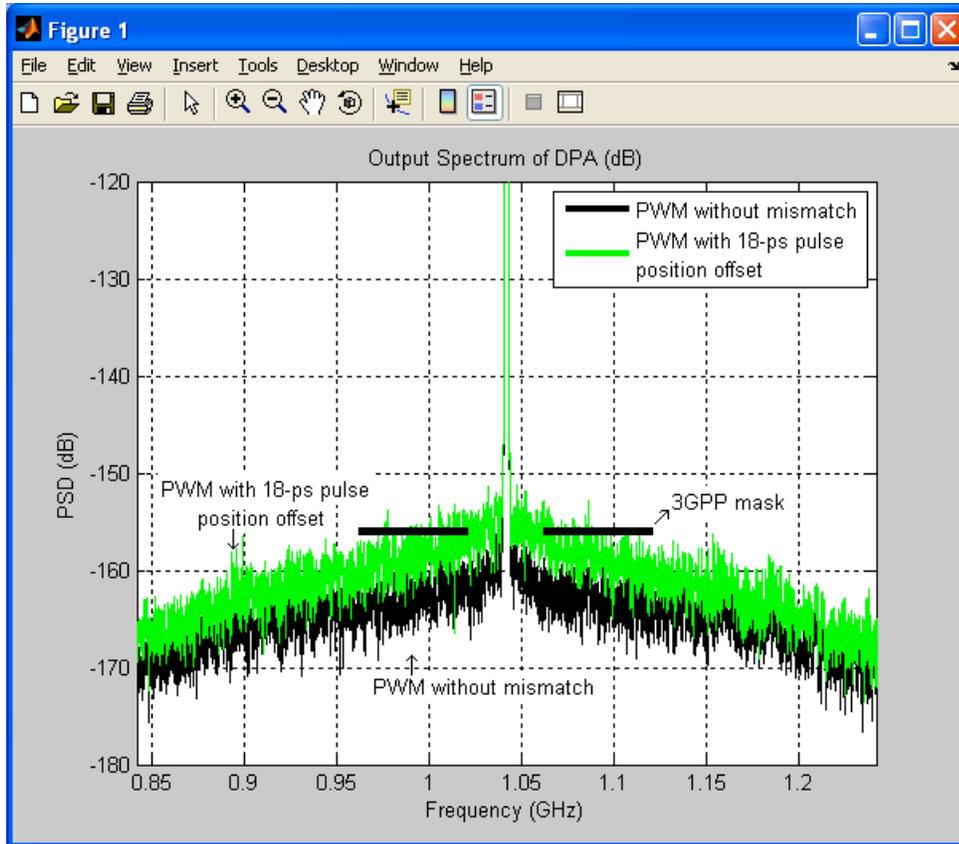
--\-- test.par (Fundamental)--L36--52%
Wrote c:\CppSim\SimRuns\Pwm_polar_tx\centered_PWM\test.par

```

- Now, the centered PWM in the testbench will have 18ps of pulse position offset. Save **test.par** file, and run the simulation.

B. Post-Processing the Simulation Results

- Open MATLAB and set the working directory to the simulation results folder, such as C:\CppSim\SimRuns\Pwm_polar_tx\centered_PWM. Run the post-processing script **centered_PWM_positionoffset.m** in the command window.
- After MATLAB completes processing the data, you should see the following output spectrum for the centered PWM signals with and without the position offset:



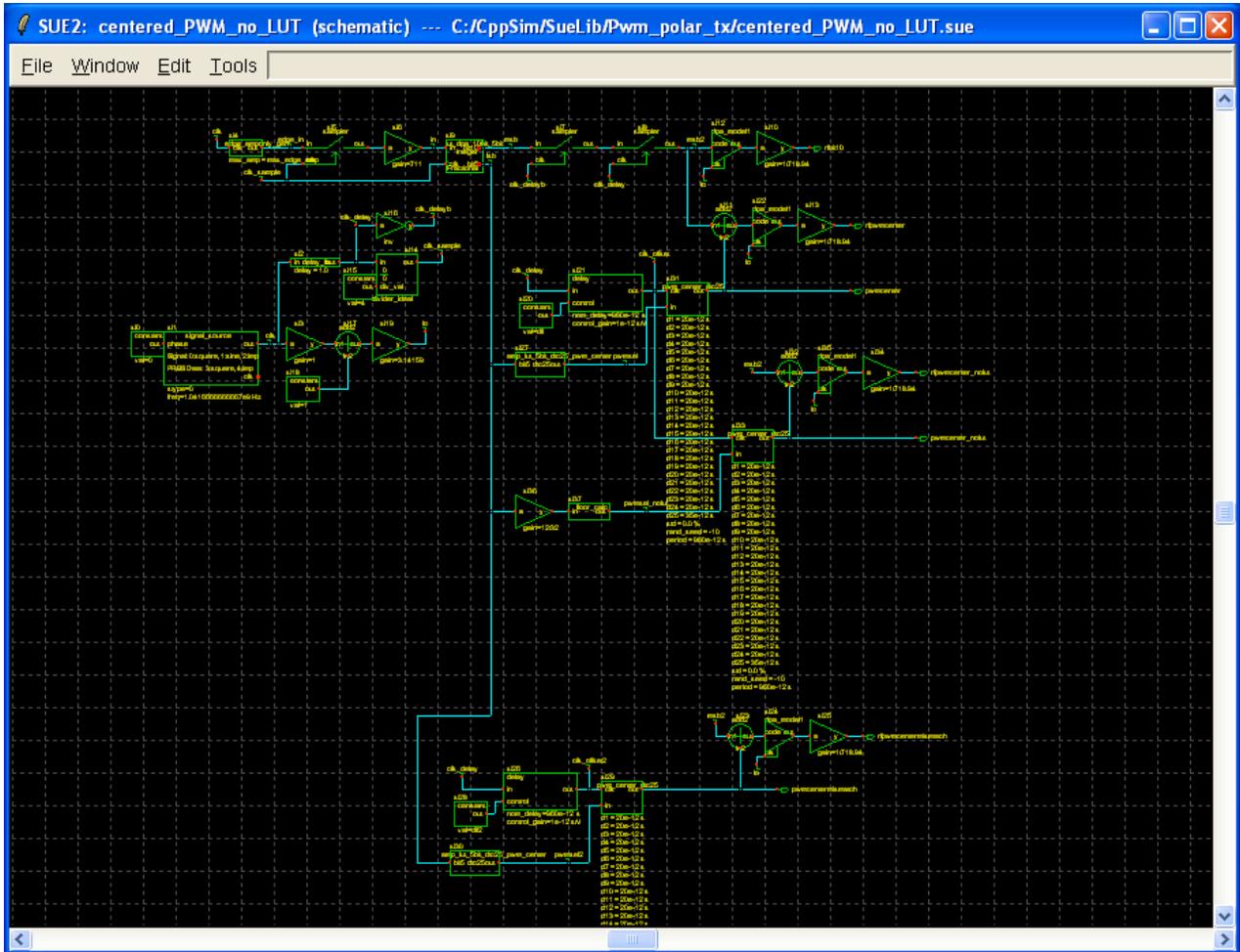
- The spectrum in green shows that the output spectrum violates the 3GPP specification if the pulse position of the PWM signal is 18ps off from center.

Comparison of Centered PWM with and without LUT

As pointed out in [6] and [7], the straightforward PWM does not improve the amplitude resolution much due to the non-linear relationship between the pulsewidth and the RF power at the carrier frequency. In this section, we will compare the centered PWM scheme with and without the amplitude LUT.

A. Opening Sue2 Schematics of a Testbench

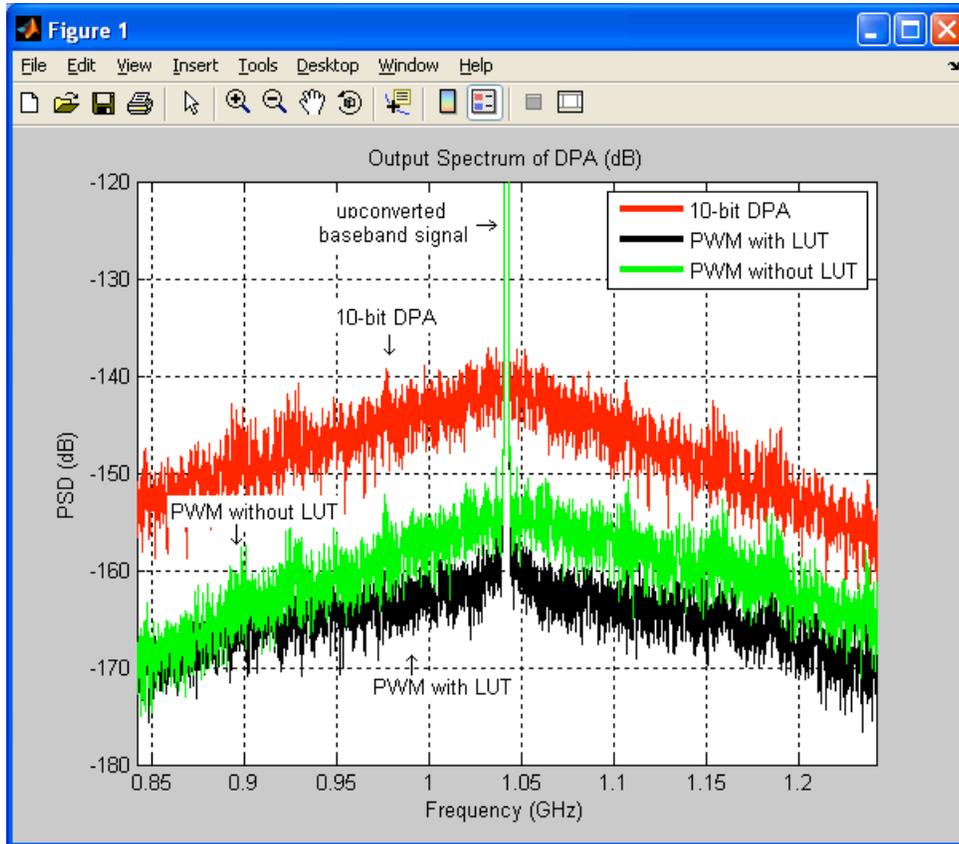
- Open a new testbench by clicking on **centered_PWM_no_LUT** schematic from the **schematic listbox**. You will see the following schematic:



- Run a simulation by clicking on **Compile/Run** button in **CppSim Simulation** window.

B. Post-Processing the Simulation Results

- Open MATLAB and set the working directory to the simulation results folder, such as C:\CppSim\SimRuns\Pwm_polar_tx\centered_PWM_LUT. Run the post-processing script **centered_PWM_LUT.m** in the command window.
- After MATLAB completes processing the data, you should see the following output spectrum for the centered PWM signals with and without the LUT.



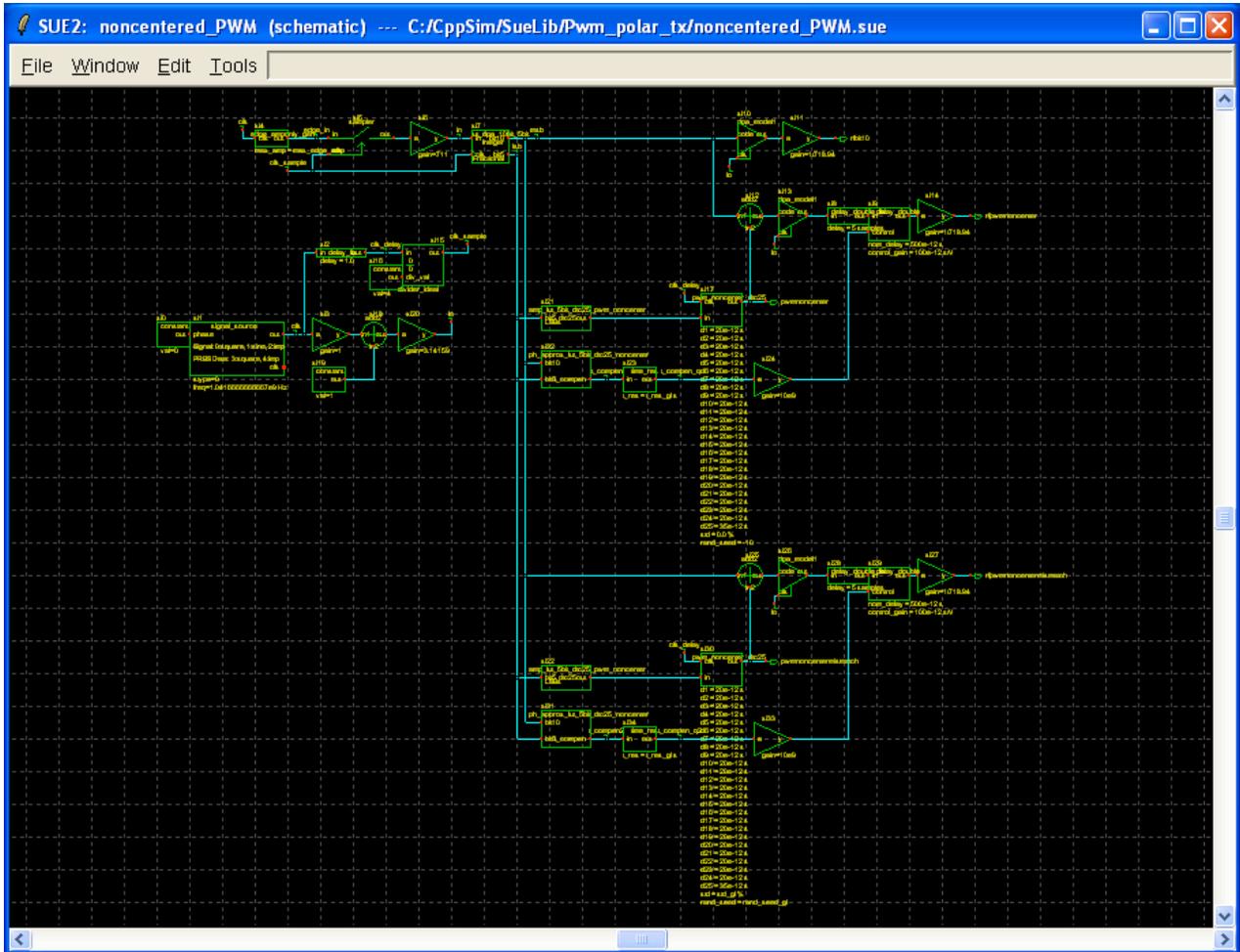
- The spectrum in green shows that the output spectrum of the centered PWM without LUT raises the quantization noise by almost 10dB compared with the PWM with LUT.

CppSim Simulations for Non-Centered PWM

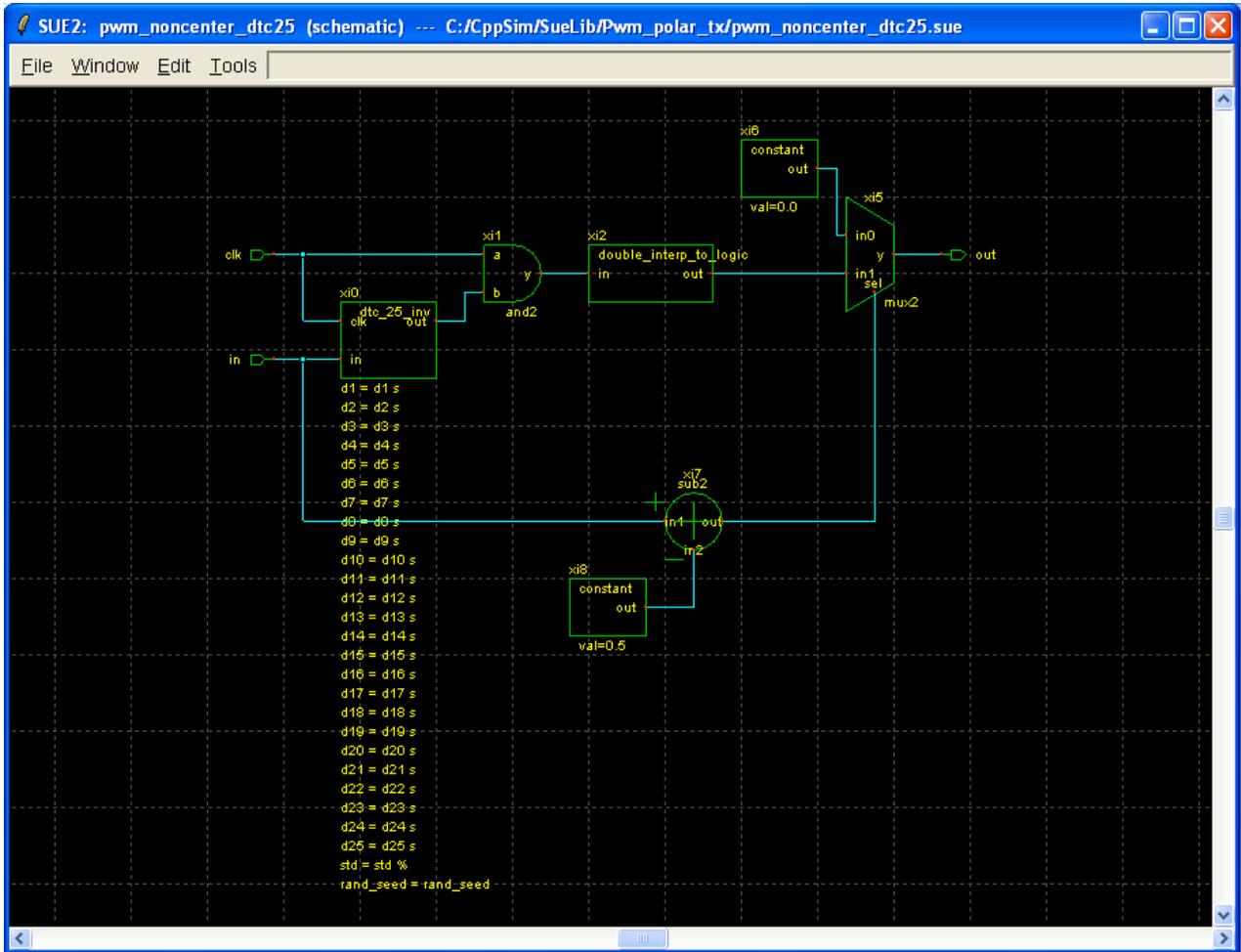
In this section, we will see the CppSim simulation example of a polar transmitter employing non-centered which is proposed in [7].

A. Opening Sue2 Schematics of a Testbench

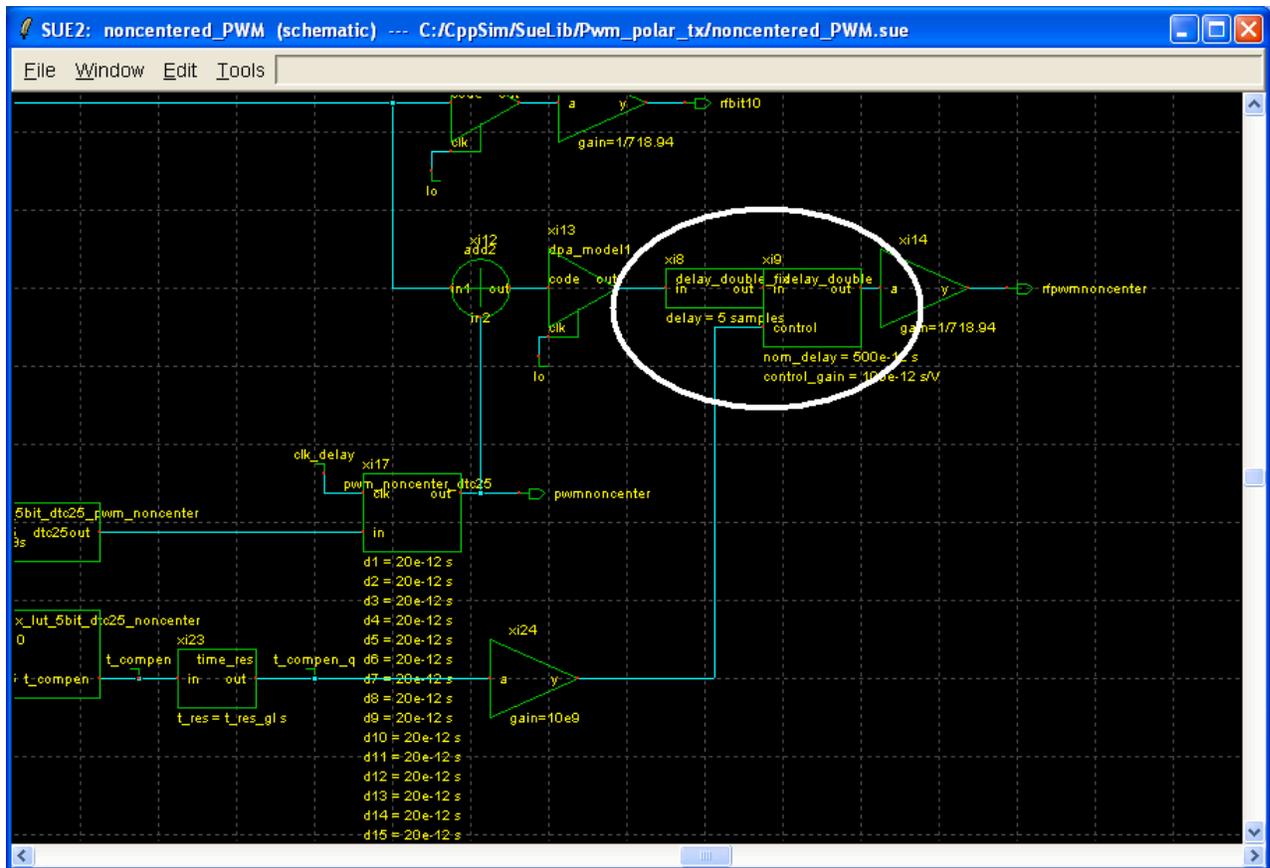
- Open a new testbench by clicking on **noncentered_PWM** schematic from the **schematic listbox**. You will see the following schematic:



- In this schematic, only amplitude path of the EDGE signal is modeled as it is in centered PWM testbenches.
- The EDGE amplitude baseband signal goes into the first amplitude LUT, **lut_dpa_10bit_5bit** which compensates for the DPA's amplitude distortion. The output is 15-bit digital signal, and the 10-bit integer part drives directly a 10-bit DPA, and the 5-bit fractional part generates the non-centered PWM signal.
- **rfbit10** is the upconverted EDGE signal from the 10-bit DPA. We will compare the quantization noise of **rfbit10** with that of a non-centered PWM signal.
- Both the 10-bit integer part and the 5-bit fractional part go into the phase LUT for non-centered PWM. The phase LUT, **ph_approx_lut_5bit_dtc25_noncenter**, is realized based on the LUT approximation method described in [7]. We can see the module description by double clicking **ph_approx_lut_5bit_dtc25_noncenter** followed by clicking **Edit** button in the new window popped up.
- The 5-bit fractional part of the baseband signal also goes into the amplitude LUT for non-centered PWM, **amp_lut_5bit_dtc25_pwm_noncenter**, which is again based on the LUT approximation method explained in [7].
- The output of the amplitude LUT, **amp_lut_5bit_dtc25_pwm_noncenter**, goes into **pwm_noncenter_dtc25**, which generates a non-centered PWM based on 25-stage delay cells. After clicking on **pwm_noncenter_dtc25**, press **e** to descend into its schematic. You will see the schematic below. It is also briefly explained in [4]. You may now press **Ctrl+E** to return to the testbench top level.



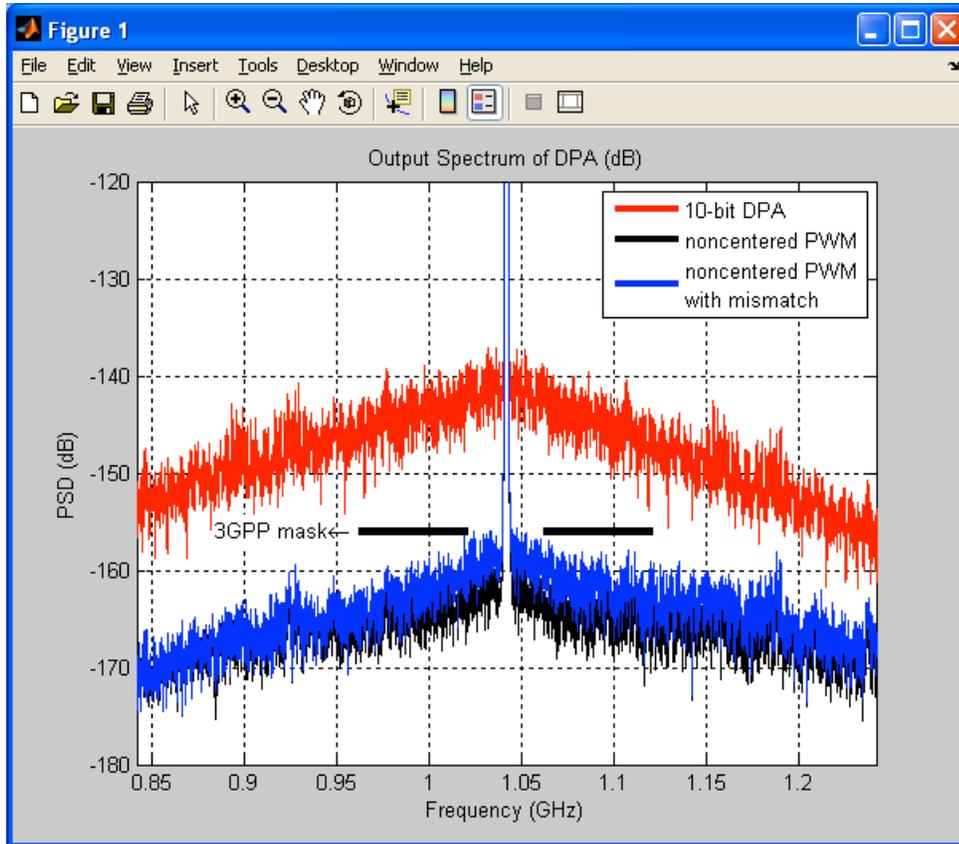
- Phase compensation of a polar transmitter is done by phase modulation using PLL, such as two-point modulation [10]. However, in this CppSim testbench, phase compensation is done by delaying the output waveform as shown below:



- The variable delay cell, **xi9**, is controlled by the output of the phase LUT such that the overall RF output signal is phase-compensated creating **rfpwmnoncenter**.
- **rfpwmnoncenter** will be compared with **rbit10**, which is the output of the original 10-bit DPA output, in terms of quantization noise.
- **rfpwmnoncentermismatch** signal is the same as **rfpwmnoncenter** except that it also includes the delay mismatch effect of the non-centered PWM generator.
- Run a simulation by clicking on **Compile/Run** button in **CppSim Simulation** window.
- Note that the simulation can take 10 minutes or more depending on the system CppSim simulation is running on.

B. Post-Processing the Simulation Results

- Open MATLAB and set the working directory to the simulation results folder, such as `C:\CppSim\SimRuns\Pwm_polar_tx\noncentered_PWM`. Run the post-processing script **noncentered_PWM.m** in the command window.
- Since the simulation result, **testfft.tr0**, is a large file, it can take 4 minutes or more to see the final results from the post-processing script depending on the system MATLAB is running on. After MATLAB completes processing the data, you should see the following output spectrum for the non-centered PWM signals with and without delay mismatch.



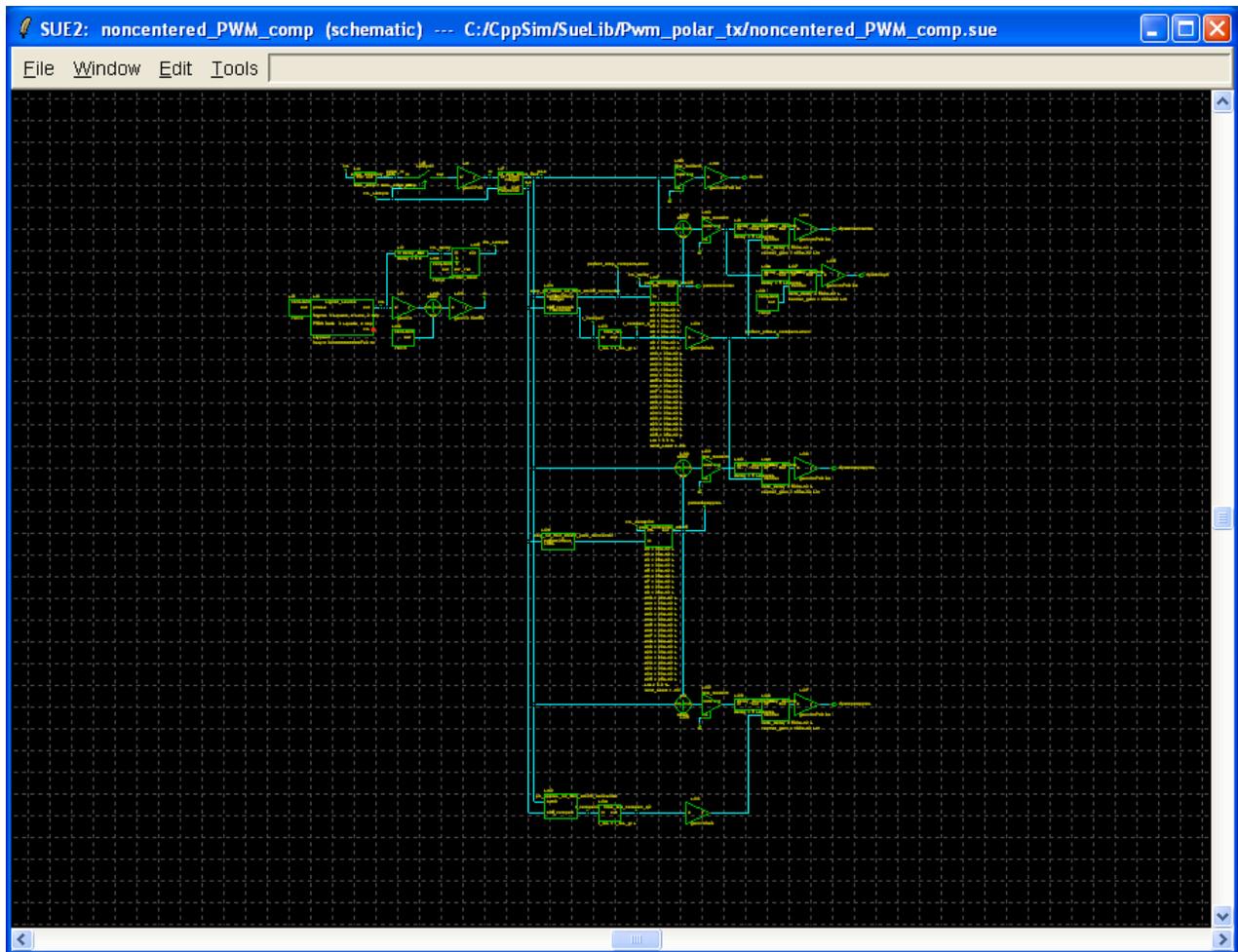
- The black line shows the spectrum of the non-centered PWM using both simplified amplitude and phase LUTs. But, the delay mismatch of the PWM signal generator is not included. The blue line shows the spectrum of the non-centered PWM with delay mismatch. Note that the delay mismatch increased the quantization noise, but it still satisfies 3GPP specification.
-

Comparison of Simplified LUT and Full-Size LUT for Non-Centered PWM

Ideally, the amplitude and the phase LUT for the non-centered PWM require both integer and fractional word as an input. However, we can use simplified LUT to reduce the size of the LUTs for EDGE amplitude signal [7]. In this section, we will compare the quantization noise of the non-centered PWM with the simplified LUTs and the full-size LUTs. In addition, we will also see the performance degradation without the phase LUT.

A. Opening Sue2 Schematics of a Testbench

- Open a new testbench by clicking on **noncentered_PWM_comp** schematic from the **schematic listbox**. You will see the following schematic.

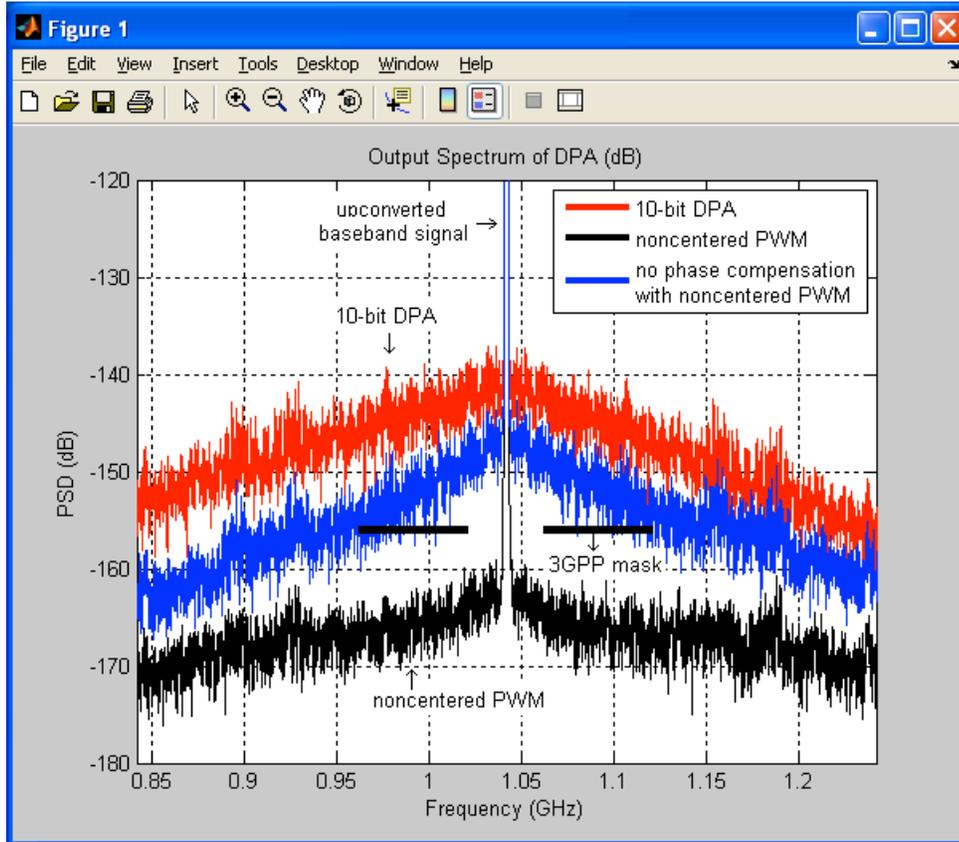


- In this schematic, **rfpwmnoncenter** signal is generated from the full-size LUT, **amp_ph_ideal_lut_5bit_dtc25_noncenter**, which includes both amplitude and phase LUTs. The LUT is modeled based on the ideal LUT formula for the amplitude and phase compensation for non-centered PWM described in [7].
- **rfpwmampapprox** signal is, on the other hand, generated using the simplified amplitude LUT, **amp_lut_5bit_dtc25_pwm_noncenter**. It takes 5-bit input, and calculates amplitude compensation using the approximation method introduced in [4]. Phase compensation is still done by the full-size LUT.
- **rfpwmphapprox** signal is generated using the simplified phase LUT, **ph_approx_lut_5bit_dtc25_noncenter** which also uses the approximation method for phase compensation. Amplitude compensation is also done by the simplified LUT, **amp_lut_5bit_dtc25_pwm_noncenter**. Thus, **rfpwmphapprox** signal is generated using both simplified amplitude and phase LUTs.
- We will compare these three signals to see how the simplified LUTs perform for EDGE.
- **rfpwmnoph** signal is created using only amplitude compensation. We will see the effect of phase compensation for non-centered PWM by comparing it with **rfpwmnoncenter**.
- Run a simulation by clicking on **Compile/Run** button in **CppSim Simulation** window.

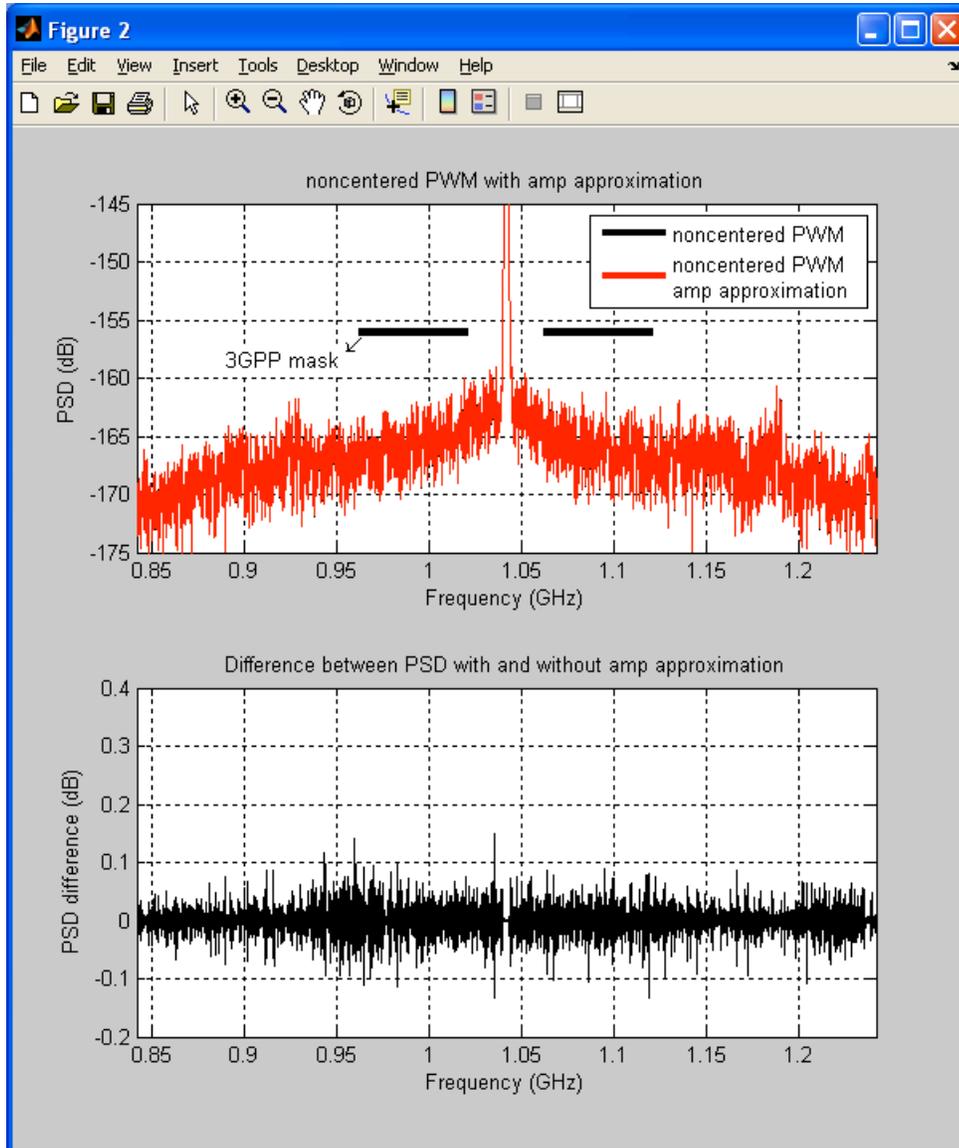
B. Post-Processing the Simulation Results

- Open MATLAB and set the working directory to the simulation results folder, such as C:\CppSim\SimRuns\Pwm_polar_tx\noncentered_PWM_comp. Run the post-processing script **noncentered_PWM_LUT_comp.m** in the command window.
- After MATLAB completes processing the data, you should see the three figures.

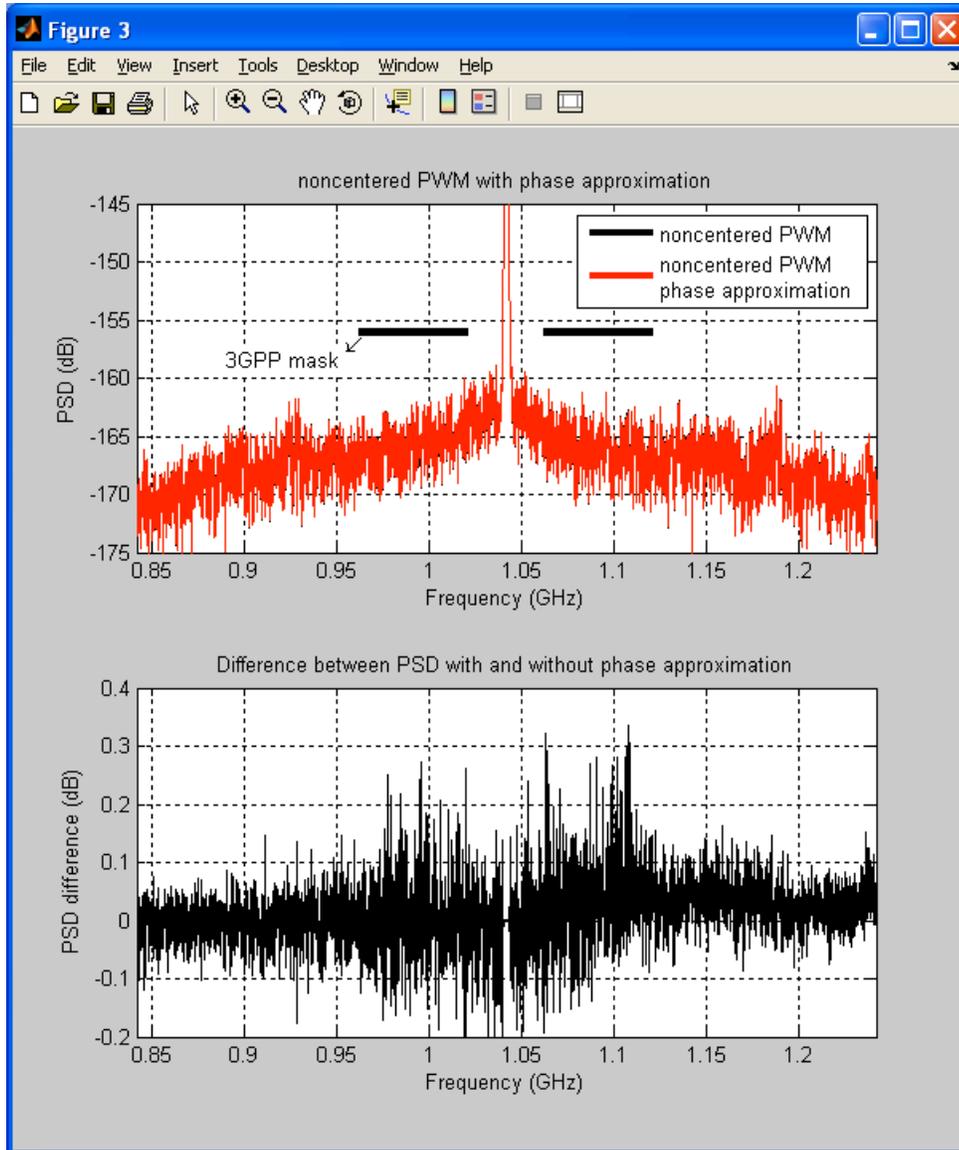
Figure 1 compares the non-centered PWM with and without the phase LUT as shown below:



- The blue line shows the spectrum of the non-centered PWM without the phase LUT. Its quantization noise is much higher than one with the phase LUT. Therefore, the phase compensation is crucial for non-centered PWM.
- **Figure 2** compares the full-size amplitude LUT and the simplified amplitude LUT as shown below:



- The output spectrum of the non-centered PWM with the simplified amplitude LUT and the full-size LUT are plotted at the top. Since the spectrum with the simplified LUT is almost same as that with the full-size LUT, we only see the red lines. At the bottom, the difference between them is plotted. The maximum difference is less than 0.2 dB as shown. Therefore, the simplified LUT with amplitude approximation in [7] does not degraded the performance of the non-centered PWM.
- **Figure 3** compares the full-size phase LUT and the simplified phase LUT as shown below:



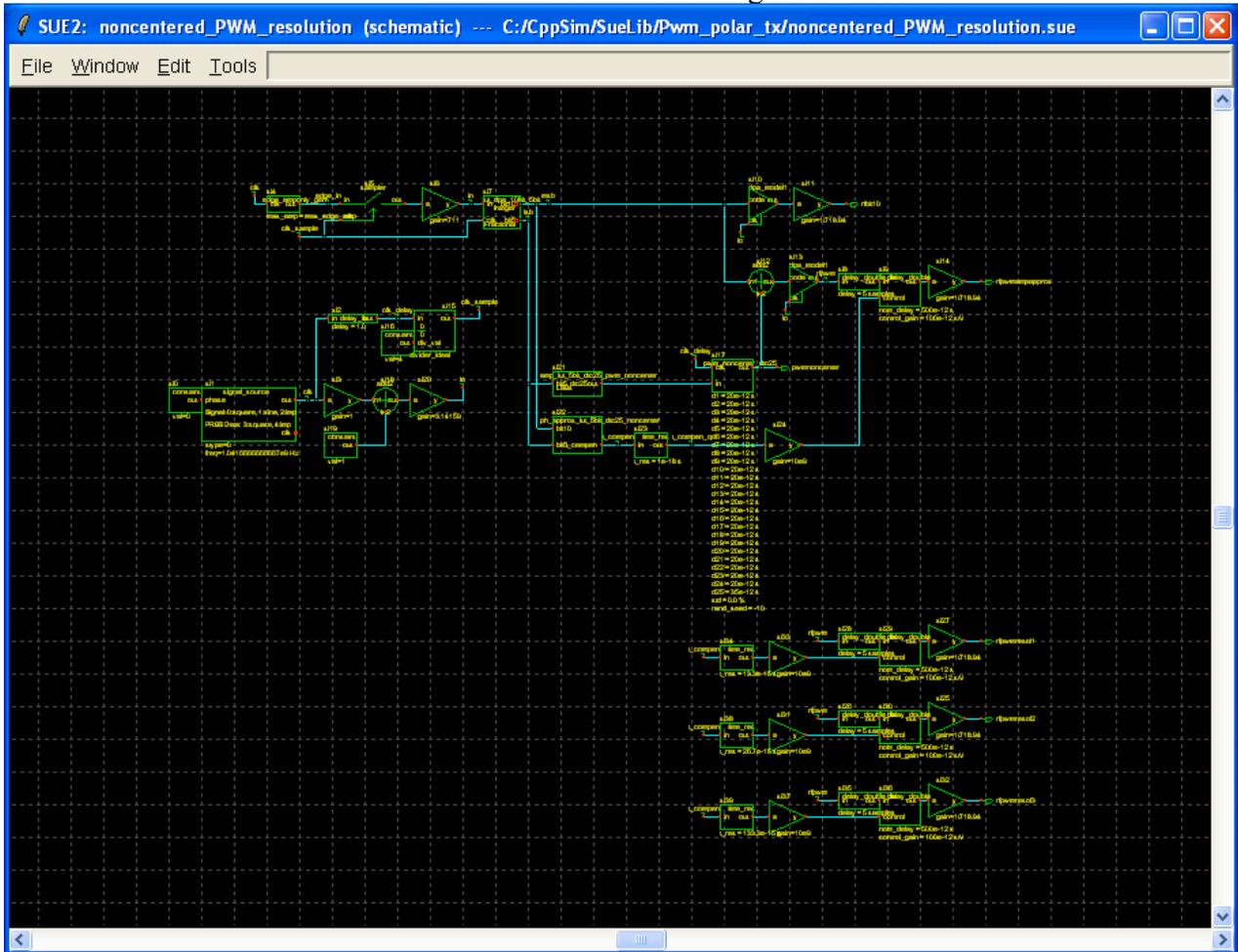
- The output spectrum of the non-centered PWM with the simplified phase LUT and the full-size phase LUT are plotted at the top. Since the spectrum with the simplified LUT is almost same as that with the full-size LUT, we only see the red lines. At the bottom, the difference between them is plotted. The maximum difference is less than 0.4 dB as shown. Therefore, the simplified LUT with phase approximation in [7] does not degraded the performance of the non-centered PWM.

Non-Centered PWM with Different Phase Resolutions

As shown in the previous section, phase compensation is crucial for non-centered PWM. In a digital polar transmitter, the accuracy of phase compensation is affected by the phase resolution of the system. In this section, we will see how the quantization noise of non-centered PWM signal changes by the phase resolution.

A. Opening Sue2 Schematics of a Testbench

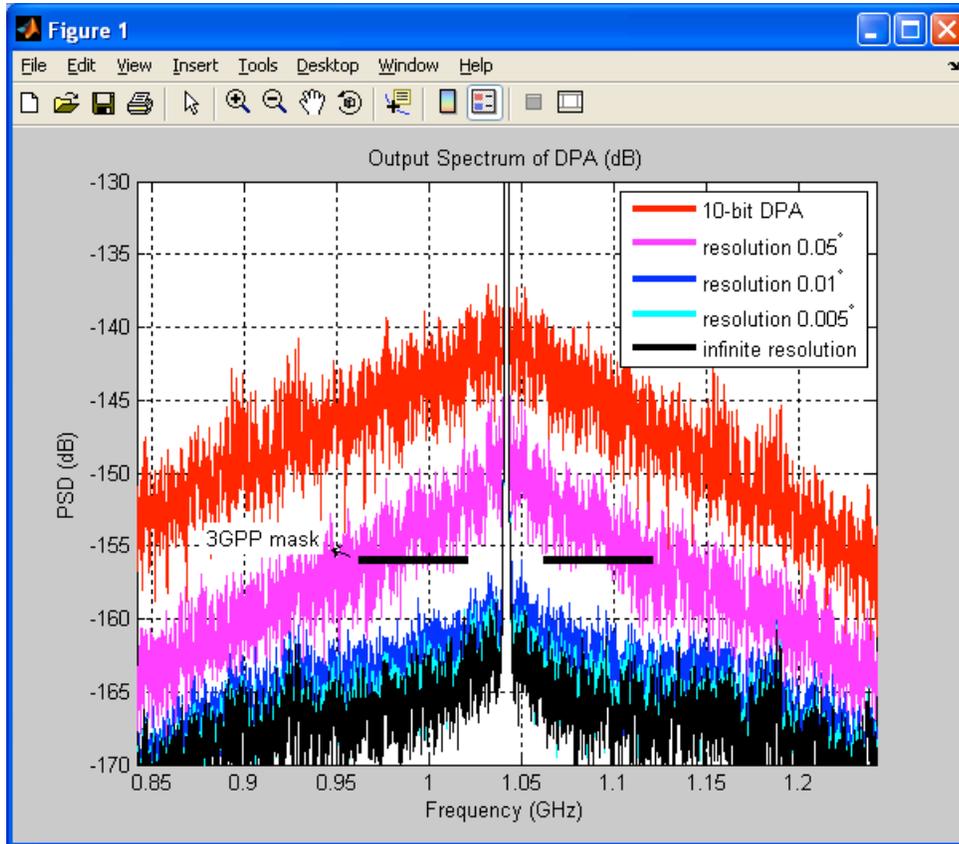
- Open a new testbench by clicking on **noncentered_PWM_resolution** schematic from the **schematic listbox**. You will see the following schematic.



- In this schematic, the resolution for phase compensation is limited by **time resolution**. For example, **rfpwmresol1** is compensated by 13.3fs of time step, which is approximately 0.005° when the period of the carrier signal is 960ps.
- In a same way, **rfpwmresol2** and **rfpwmresol3** are compensated by 26.7fs and 133.3fs of time step, respectively. Thus, the phase resolutions for **rfpwmresol2** and **rfpwmresol3** are 0.01° and 0.05° , respectively.
- Run a simulation by clicking on **Compile/Run** button in **CppSim Simulation** window.

B. Post-Processing the Simulation Results

- Open MATLAB and set the working directory to the simulation results folder, such as C:\CppSim\SimRuns\Pwm_polar_tx\noncentered_PWM_resolution. Run the post-processing script **noncentered_PWM_LUT_resolution.m** in the command window.
- After MATLAB completes processing the data, you should see the following output spectrum:



- The quantization noise increases as the phase resolution decreases. The simulation result shown above shows that the non-centered PWM method can satisfy 3GPP specification with the phase resolution of 0.01° or above.

Conclusion

In this tutorial, we have presented the simulations of digital polar transmitters employing the PWM methods to improve the amplitude resolution. We have shown the examples of centered PWM and non-centered PWM schemes. In order to examine the performance of the proposed PWM schemes quantitatively, we have used MATLAB scripts to plot the spectrum of the simulation results. With the simulation examples in this tutorial, the user can easily reproduce the simulation results shown in [6] and [7], and also explore the proposed PWM scheme further.

Reference

- [1] R. Staszewski, J. Wallberg, S. Rezeq, C.-M. Hung, O. Eliezer, S. Vemulapalli, C. Fernando, K. Maggio, R. Staszewski, N. Barton, M.-C. Lee, P. Cruise, M. Entezari, K. Muhammad, and D. Leipold, "All-digital PLL and transmitter for mobile phones," *IEEE J. Solid-State Circuits*, vol. 40, no. 12, pp. 2469–2482, Dec. 2005.
- [2] J. Mehta, R. B. Staszewski, O. Eliezer, S. Rezeq, K. Waheed, M. Entezari, G. Feygin, S. Vemulapalli, V. Zoicas, C.-M. Hung, N. Barton, I. Bashir, K. Maggio, M. Frechette, M.-C. Lee, J. Wallberg, P. Cruise, N. Yanduru, "A 0.8 mm² all-digital SAW-less polar transmitter in 65nm EDGE SoC," *Proc. of IEEE Solid-State Circuits Conf.*, pp. 58–59, Feb. 2010.
- [3] A. Jerng and C. G. Sodini, "A wideband $\Delta\Sigma$ digital-RF modulator for high data rate transmitters," *IEEE J. Solid-State Circuits*, vol. 42, no. 8, pp. 1710–1722, August 2007.
- [4] P. Eloranta, P. Seppinen, S. Kallioinen, T. Saarela, and A. P. Arssinen, "A Multimode Transmitter in 0.13 μm CMOS Using Direct-Digital RF Modulator," *IEEE J. Solid-State Circuits*, vol. 42, no. 12, pp. 2774–2784, Dec. 2007.
- [5] R. B. Staszewski, "Digital deep-submicron CMOS frequency synthesis for RF wireless applications," Ph.D. thesis, Univ. of Texas at Dallas, Aug. 2002.
- [6] M. Park, M. H. Perrott, and R. B. Staszewski, "A Time-Domain Resolution Improvement of an RF-DAC," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 7, pp. 517–521, July 2010.
- [7] M. Park, M. H. Perrott, and R. B. Staszewski, "An Amplitude Resolution Improvement of an RF-DAC Employing Pulse Width Modulation," submitted for review in *IEEE Trans. Circuits Syst. I*.
- [8] S. E. Meninger and M. H. Perrott, "A fractional-N frequency synthesizer architecture utilizing a mismatch compensated PFD/DAC structure for reduced quantization-induced phase noise," *IEEE Trans. Circuits Syst. II*, vol. 50, issue 11, pp. 839–849, Nov. 2003.
- [9] M. Nielsen and T. Larsen, "An RF Pulse Width Modulator for Switch-Mode Power Amplification of Varying Envelope Signals," *Proc. Silicon Monolithic Integrated Circuits in RF Syst. Top. Meeting*, pp. 277–280, Jan. 2007.
- [10] R. B. Staszewski, D. Leipold, and P. T. Balsara, "Just-in-time gain estimation of an RF digitally-controlled oscillator for digital direct frequency modulation," *IEEE Trans. Circuits Syst. II*, vol. 50, issue 11, pp. 887–892, Nov. 2003.
- [11] M. Perrott, "Cpssim behavior simulator package." [Online]. Available: <http://www.cpssim.com>
- [12] J. Mehta, V. Zoicas, O. Eliezer, R. B. Staszewski, S. Rezeq, M. Entezari, and P. T. Balsara, "An efficient linearization scheme for a digital polar EDGE transmitter," *IEEE Trans. on Circuits Syst. II*, vol. 57, no. 3, pp. 193–197, Mar. 2010.
- [13] M. Perrott, "Fast and accurate behavioral simulation of fractional-n frequency synthesizers and other pll/dll circuits," *Proc. 39th Design Automation Conf.*, 2002, pp. 498–503.