Preliminary Exam: Dr Samuel Palermo

Younghoon Song

1. **Build a macromodel of a 4-bit parallel PRBS generator and verifier. The PRBS sequence length should be 2^7-1. Refer to Appendix A in Ken Yang's thesis for some intro material on parallel PRBS implementations.**

For very high-speed generation of PRBS sequences, it is useful to know which architecture is optimal for a particular application. The different options that can be considered are parallel versus series PRBS generator architectures and the level of multiplexing. The level of multiplexing determines how much slower, relative to the final output, the core generator is operated, thus requiring proportionally less power. However, if the multiplexing level is too deep, too much power might be spent in the multiplexer itself. Figure 1 shows serial PRBS generation and detection configuration.

A. Serial PRBS Generation and Error Detection

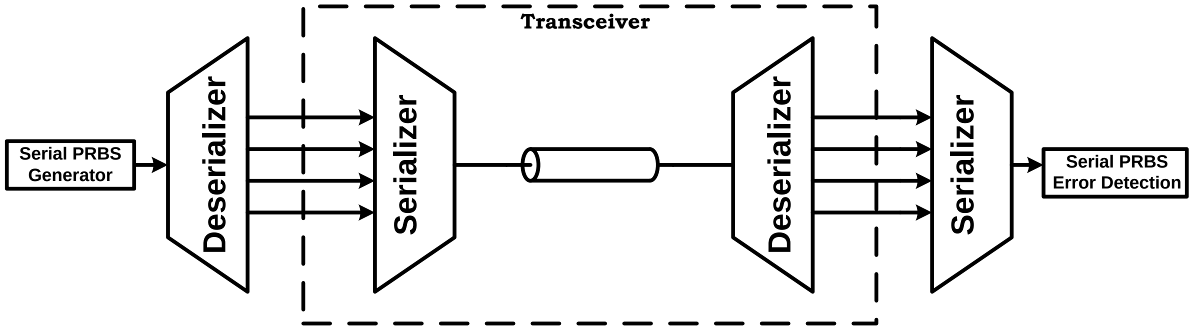


Figure 1 Serial PRBS generation and Error Detection

Series PRBS generators are linear feedback shift registers, where the length of the register n and the feedback function determine the length of the sequence p=2^n -1[1]. For multiplexing the sequence to q times the original bit rate, original sequences, spaced apart by (p-1)/q bits in phase are required [2]. An efficient, algorithm exists for obtaining the phase shifts, nevertheless, the number of XOR gates required to implement the phase shifts in hardware grows exponentially with q, which is shown in Figure2.

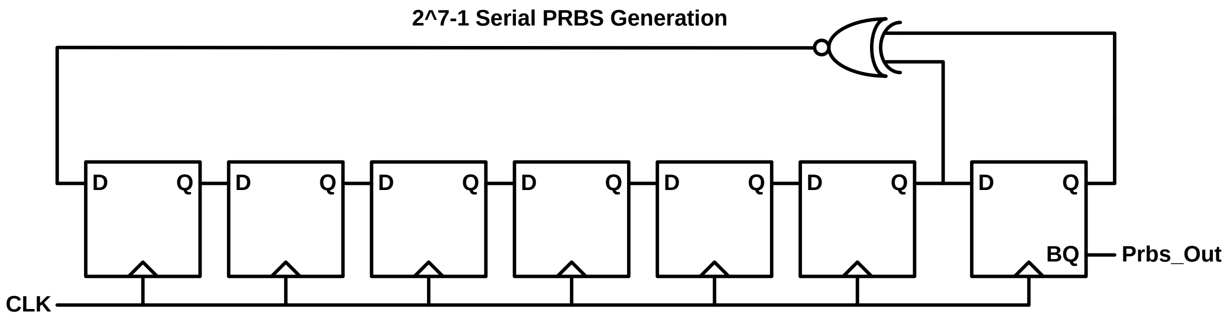


Figure 2 2^7-1 Serial PRBS Generation

In simulation, the initial condition was zero, therefore the resister remains in a degenerate state. XNOR gate was used instead of some method of initialization.

The error detection is based on the principle of multiplication by a reciprocal polynomial. The technique uses the same length shift register chain **as** in the generating polynomial. In Figure 3**,** the serial detection scheme is shown. The shift register is continuously fed by the incoming bit stream and the shift register outputs are XNOR in the same manner as in the generating polynomial. The XNOR output, which is inverse with Din compare with Din XOR gate. If there bits are inverse, the checker generates a no error flag, BQ, for the incoming data. If the bits same, an error is flagged. The technique allows self synchronization to the incoming bit stream, without using any additional hardware.

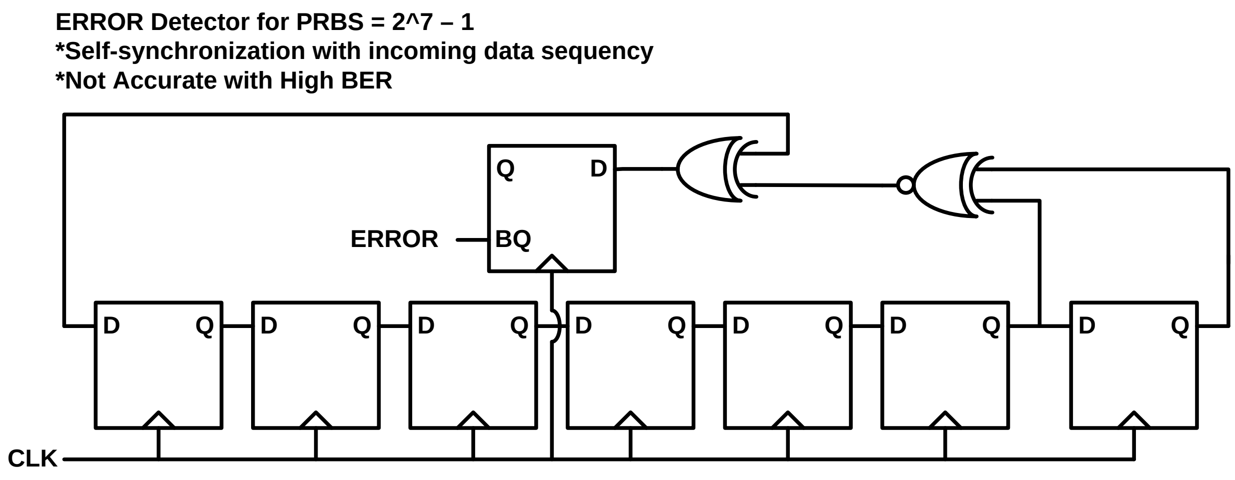

ERROR Detector for PRBS = 2^7 – 1
*Self-synchronization with incoming data sequency
*Not Accurate with High BER
Q
D
ERROR
BQ
D
Q
CLK

Figure 3 ERROR Detection for PRBS = 2^7-1

Simulation Result: PRBS Generation, 1 to 4 Demux, 4 to 1 MUX, and PRBS detection

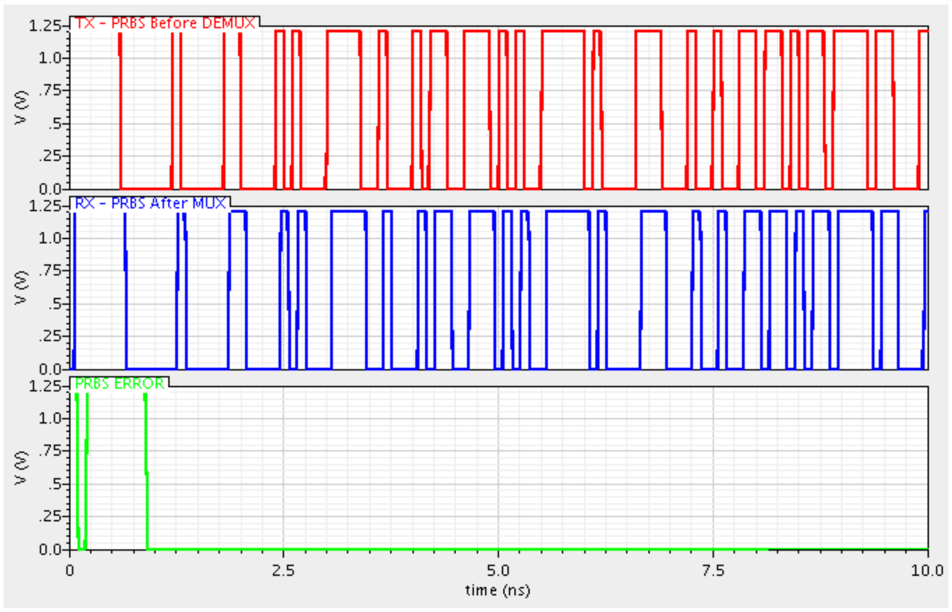


Figure 4 10Gbps 2^7 – 1 PRBS Generation and Detection

Figure 4 shows simulation result. Initially error exists; however it is not valid time for error detection. Therefore we can ignore those errors.

B. 4 bits Parallel PRBS Generation and Purposed Error Detection

In the parallel PRBS generator and detector architecture, which is shown figure 5, the phase shifted sequences are available directly from the generator. The n*m transition matrix T, which can be obtained from the characteristic polynomial of the PRBS, proves useful for constructing parallel PRBS generators. A procedure for translating into the PRBS generator schematic with parallel outputs is given in [3]. The resulting outputs are phase shifted appropriately for direct multiplexing.

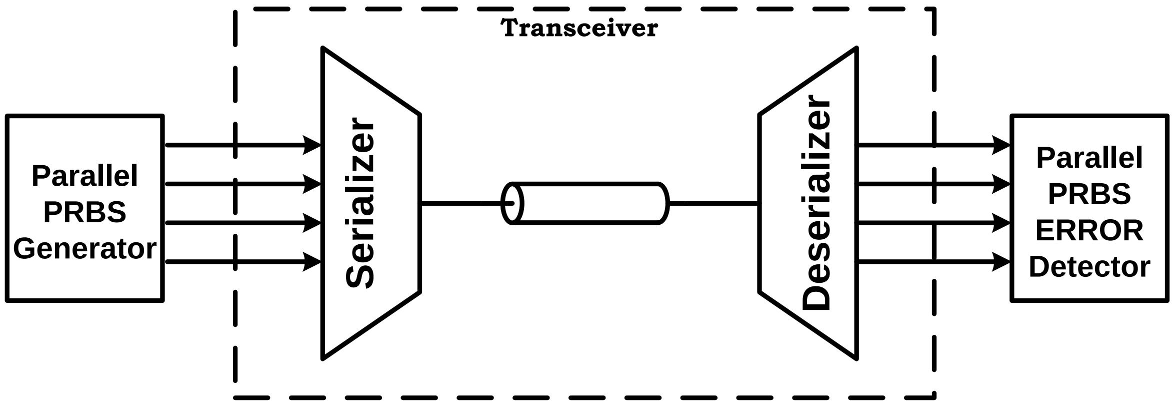


Figure 5 Parallel PRBS generator and detector architecture

The circuit requires *n* flip-flops as in the traditional method, but *m* XOR gates are needed to generate parallel m-bit data streams. Figure 6 shows the connections among flip-flops and XOR gates to form a parallel generator. The flip-flops are connected as a parallel register with the output fed back to the input through the XOR gates. The XOR gates are connected so each bit of the new word is generated according to a given polynomial equation. In this paper, we have used 4-to-I multiplexer to form a single high-speed bit stream using the low-speed parallel data generated by 4-bit parallel PRBS generators.
The XOR gates are connected so each bit of the new word is generated according to the polynomial equation. For example, on the first clock cycle, bit six of the PRBS is generated by XOR bit 1 and bit 3, which are presently held in the register. At the same time, bit *7* is generated from bits *2* and 4. A slight irregularity occurs at the bottom of the register. In order to calculate bit 9, bits 4 and *6* are required. Bit *6* is not held in the register, so it must be obtained from an earlier XOR, where it is being calculated.

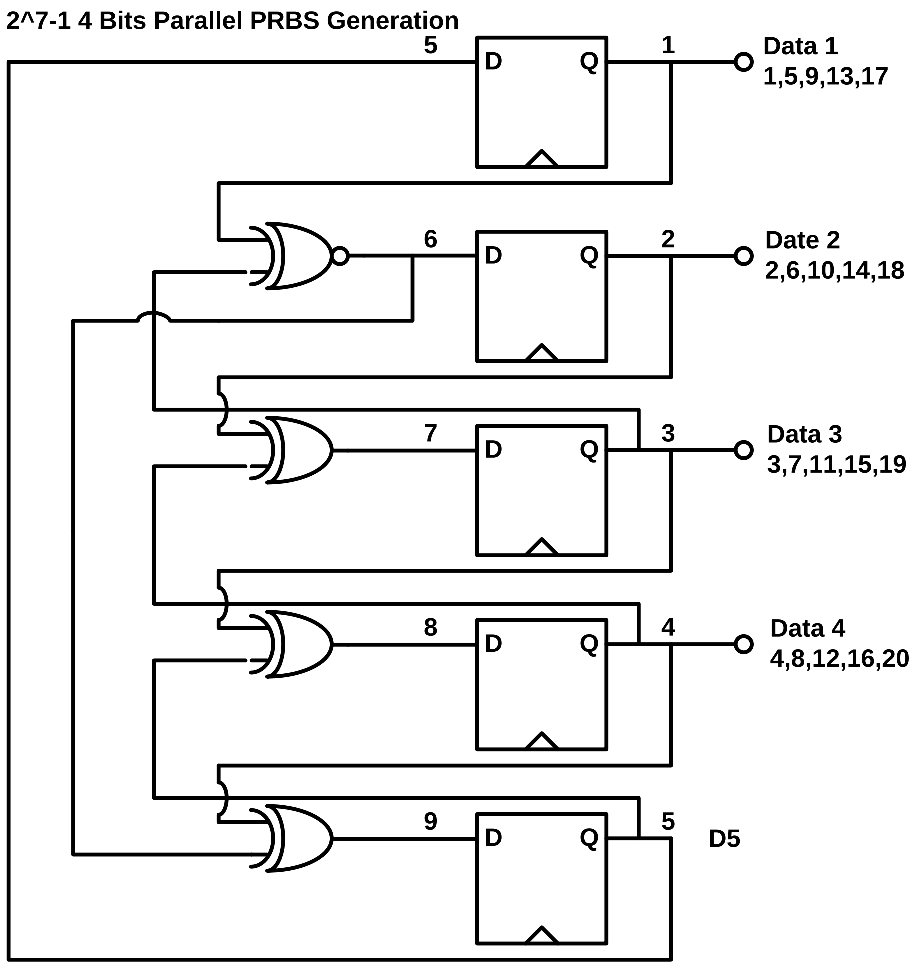


Figure 6 N=5, W=4 2^7-1 Parallel PRBS Generator [3]

Again, to prevent it remains in a degenerate state. XNOR gate was used instead of some method of initialization.

Data 1[n-1] = D5 [n]

Data2 [n-1] = XNOR (Data1[n], Data3[n])

Data3 [n-1] = XOR (Data2[n],Data4[n])

Data4 [n-1] = XOR (Data3[n], Data[5])

D5 [n-1] = XOR (Data4[n], XNOR (Data1[n+1], Data3[n+1]))

D5 internally used for 4 bit Parallel PRBS generation.

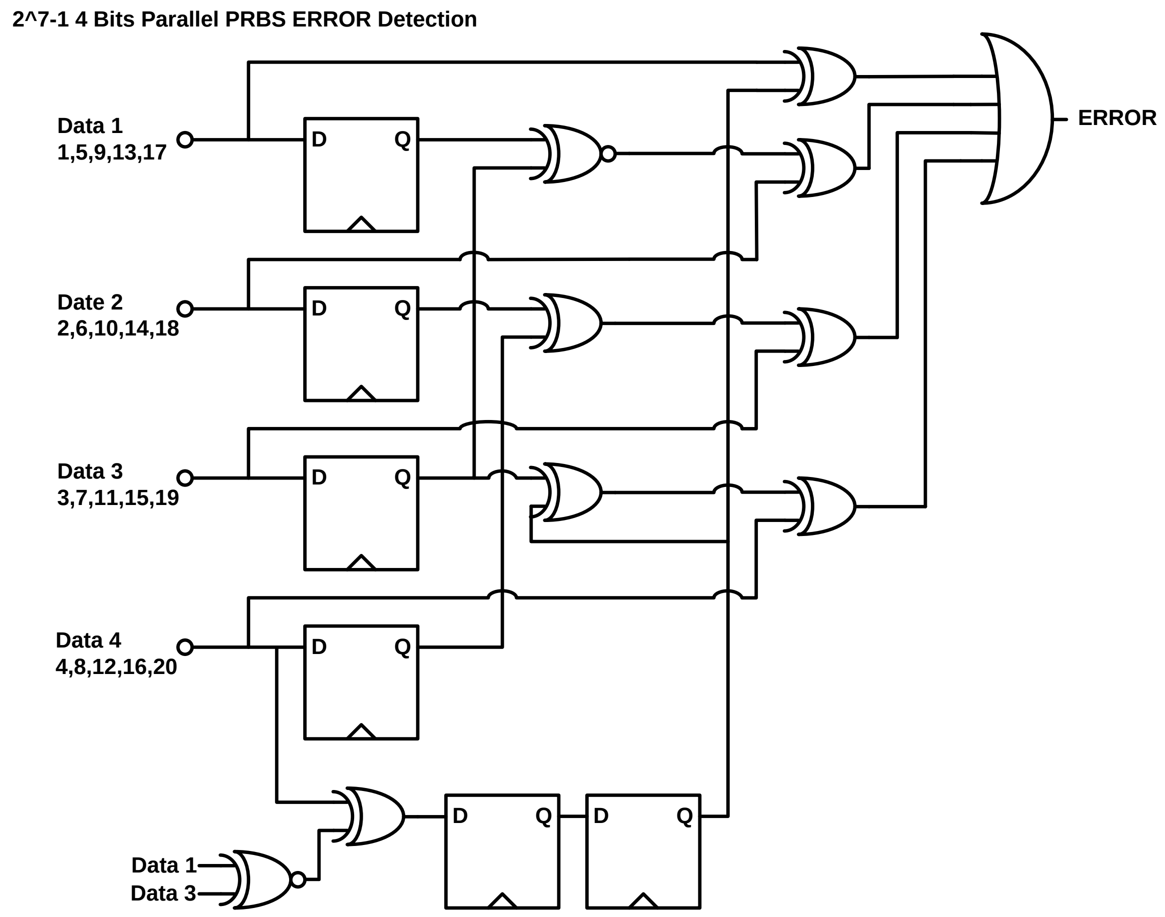


Figure 7 2^7-1 4 Bits Purposed Parallel PRBS ERROR Detection

Purposed Error detection configuration is simple. We know how to generate Data 1 – 4 based on PRBS generation equation. Therefore it simply compares incoming data and internally generation data. They are XORed, thus it will generate Zero, if they are identical. Finally 4 data are ORed, therefore if one of data has error, it will generate ERROR signal. However, this configuration has to be well design for timing such as match gate delay.

Simulation Result: 2.5Gbps 4 bits parallel PRBS Generation and PRBS Error detection

2^7-1 PRBS 2.5Gbps 4 bits was generate. Figure 8 show if generation data match with receiving date, as we expect, there have no error.

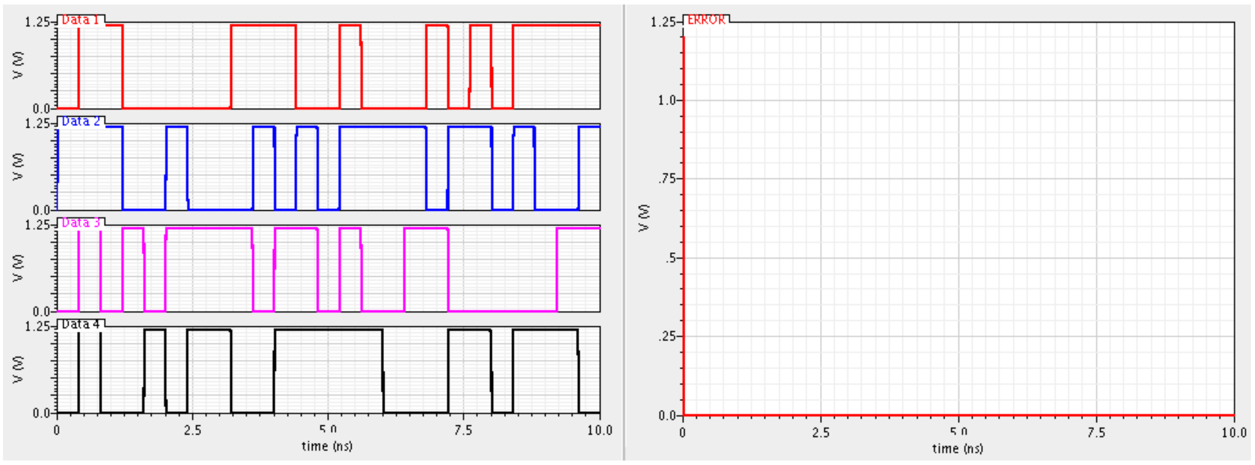


Figure 8 PRBS generation and Detection without ERROR

To see error detection performance, RX data 2 error bit was generated, which is shown in Figure 9. As we can see PRBS error detection detects this ERROR.

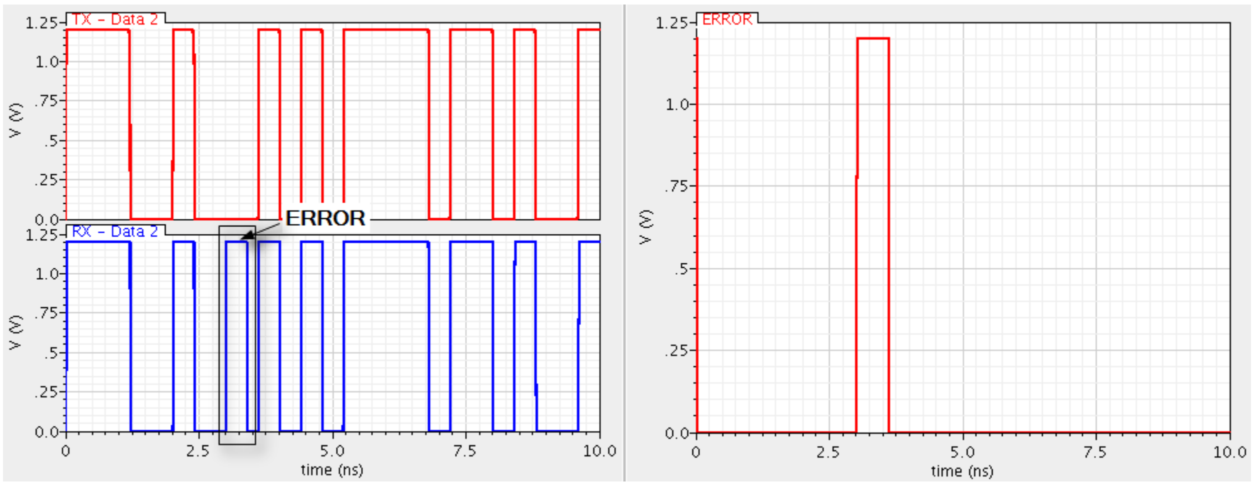


Figure 9 PRBS generation and Detection with ERROR

Reference

[1] S. W. Golomb, *Shift Register Sequences*. San Francisco, CA: Holden-Day, 1967.

[2] F. Sinnesbichler, A. Ebberg, A. Felder, and R. Weigel, “Generation of high-speed pseudorandom sequences using multiplex techniques,” *IEEE Trans. Microw. Theory Tech.*, vol. 44, no. 12, pp. 2738–2742, Dec. 1996.

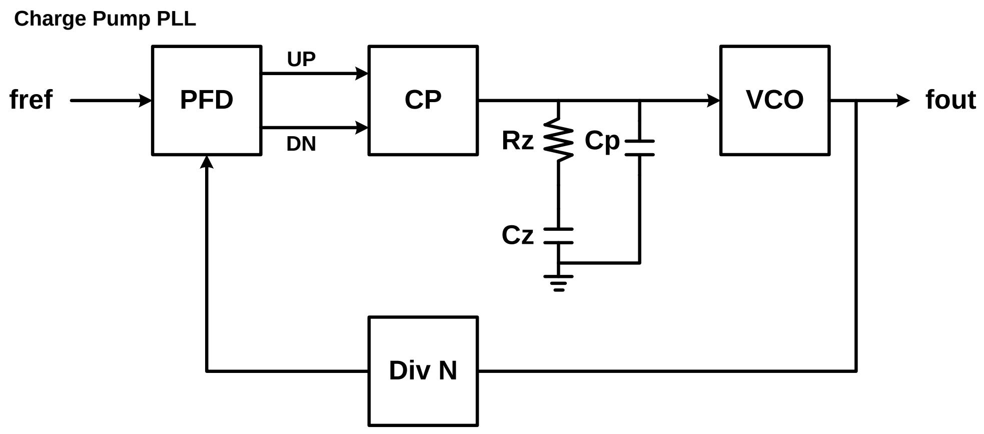
[3] Seongwon Kim “45-Gb/s SiGe BICMOS PRBS Generator and PRBS Checker” IEEE CICC 2003

2. **Build both a time-domain and frequency domain macromodel for a charge-pump PLL with a standard filter consisting of 1 resistor and 2 capacitors.**
    a. **The models should have all major PLL components programmable, i.e. reference clock frequency, charge-pump current, filter R and Cs, VCO gain and center frequency, loop division factor, etc.**
    b. **The models can be implemented in either Matlab, Simulink, or CppSim.**
    c. **For the frequency domain PLL model, have the ability to input various noise spectrums at the PLL input, VCO input, and VCO output. The model should integrate the output phase noise over a user programmable bandwidth to calculate the random jitter rms value. Provide test cases with noise spectrums injected at the PLL input, VCO input, and VCO output.**

A. PLL Modeling (Part a and b)

1. Charge Pump PLL Design

Fig. 1 is a block diagram of a charge pump based PLL



Frequency	2.4GHz
Div N	480
Fref	5 MHz
Settling Time	< 100us

Figure 1 Block diagram of a charge pump based PLL and PLL specification

Free running frequency and a gain of VCO are given as 2.4 GHz and 1 GHz/V, respectively.

$$f_{freerun} = 2.4GHz$$
$$K_{VCO} = 2\pi \times 1GHz/V$$

F_{REF} can be 5 MHz, and it enables to use integer frequency divider.
Required dividing ratios would be

N = 480

Now, loop parameters should be determined. Since loop has one zero and two poles (one is at zero frequency), a zero and pole frequency should be determined.

$$\omega_z = \frac{1}{RzCz}, \quad \omega_p = \frac{(Cz+Cp)}{RzCzCp} \cong \frac{1}{RzCp}$$

Open-loop phase transfer function from an input of PFD to the output of divider is

$$H_{open}(s) = \frac{I_{CP}}{2\pi}\frac{K_{VCO}}{s}\frac{1}{N}\frac{1+s/\omega_z}{sC_Z\left(1+s/\omega_p\right)} = \frac{I_{CP}K_{VCO}}{2\pi Ns^2}\frac{1+sRzCz}{\left(RzCzCps+(Cz+Cp)\right)}$$

where 1/2π is a PFD gain, Icp is a CP gain and K_{VCO} is a VCO gain.

Starting from Gardner's stability limit, cross-over frequency (ω_c) should be less enough than reference frequency.

$$\omega_c < \frac{\omega_{REF}}{10} = \frac{2\pi \times 5MHz}{10} = 2\pi \times 500KHz$$

Damping factor is usually in between 0.707 and 2. Damping factor of 1 means that two poles in closed-loop transfer function would be overlapped. Damping factor less than 1 is that two poles are complex conjugated.
For an optimal settling time and bandwidth, damping factor of 0.707 (under damped) is chosen.

$$\xi = 0.707$$

Natural frequency can be decided by cross-over frequency and damping factor

$$\omega_n = \frac{\omega_c}{2\xi} = \frac{2\pi \times 500KHz}{1.414} \cong 2\pi \times 350KHz$$

With calculated frequencies in above equations, pole and zero frequencies can be decided.

$$\begin{aligned}
&\textit{Rule}\text{ of Thumb}\\
&\omega_c{}^2 = \omega_p \omega_z\\
&\omega_p = \omega_c \times 4\xi^2 = 2\pi \times 1MHz\\
&\omega_z = \frac{\omega_c}{4\xi^2} = 2\pi \times 250KHz
\end{aligned}$$

As seen above, pole frequency is 4 times higher than zero frequency. However, system stability is affected by pole frequency through phase margin; it is desirable to separate pole frequency from at least 8 times higher than zero frequency. Hence, pole frequency is adjusted by factor of 2.

$$\omega_p = 2\pi \times 2MHz$$

Note that small damping factor makes the separation of pole and zero to be smaller, which makes the system fast but not much stable.

Settling time can be estimated as

$$t_s \cong \frac{1}{\xi\omega_n} \ln \frac{\Delta f}{\alpha f_0 \sqrt{1-\xi^2}} = 4.8\mu\sec$$

where f_0 is the frequency from which the synthesizer starts transition, Δf is the amount of frequency jump and α is the settling accuracy. With f_0 = 2.405 GHz, Δf = 75 MHz and α = 25×10^{-6}, the settling time would be 4.8 µsec which is well below than the specification with a good margin.
Closed loop system transfer function is

$$H_{closed}(s) = \frac{H_{open}(s)}{1+H_{open}(s)} \cong \frac{\frac{IKvco}{2\pi NCz}\left(1+s/\omega_z\right)}{s^2 + \frac{IKvco}{2\pi NCz}\frac{1}{\omega_z}s + \frac{IKvco}{2\pi NCz}}$$

Transfer function can be approximated as a second order with the assumption that ω_p is much higher than ω_n.
From natural frequency, Cz, Cp and Rz can be determined

$$Cz = \frac{IcpKvco}{2\pi N\omega_n{}^2}, \quad Rz = \frac{1}{\omega_z Cz}, \quad Cp \cong \frac{1}{\omega_p R_z}$$

In order to calculate the numbers of loop filter, current of charge pump should be decided. Considering power consumption of charge pump, 0.1mA would be 0.12 mW if VDD = 1.2V, which is reasonable number.

$$I=0.1mA$$

To calculate Cz, Rz, Cp, N is assumed to be the 480.

$$Cz = 42\,pF, \quad Rz = 15K\Omega, \quad Cp \cong 5.3\,pF$$

2. Simulation Result

Input parameter by Users

1. Input Clock Frequency
2. Charge Pump parameters
   - Charge pump Current
   - Error up
   - Error Down
   - Slew rate
3. Loop Filter parameters
   - Rz, Cz, and Cp
4. KVCO parameters.
   - KVCO
   - Free running Frequency
5. Divider Parameter

Time Domain Test model by Simulink

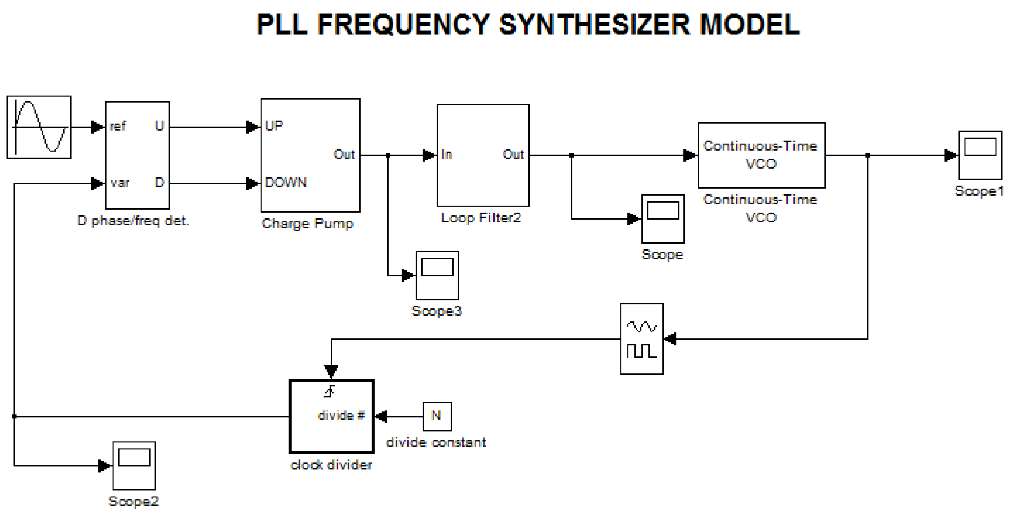
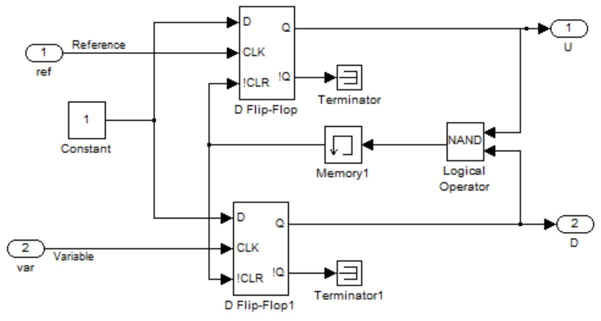
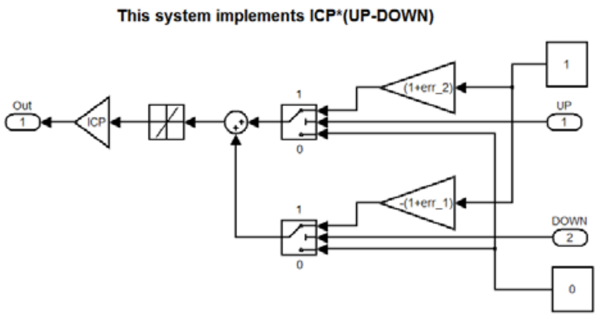


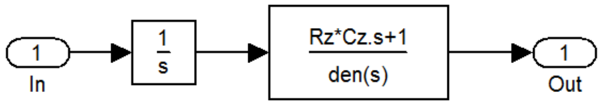
Figure 2 Time Domain Test model – Simulink



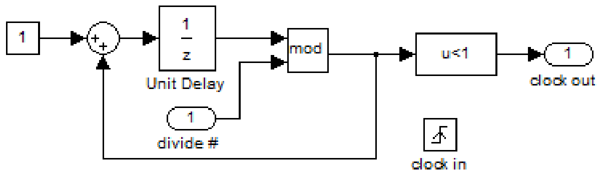
(a) PFD



(b) CP



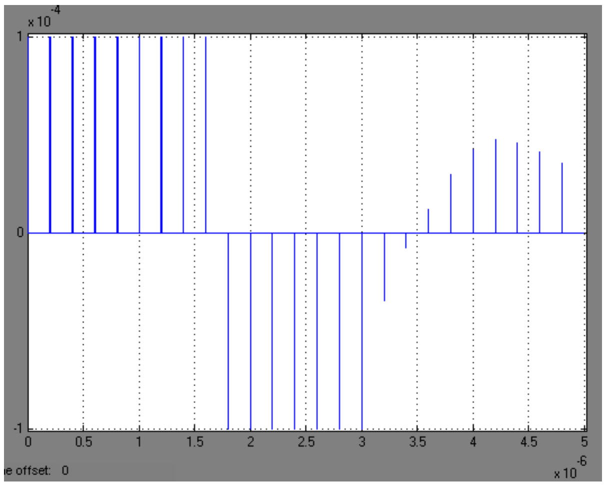
(c) Loop Filter



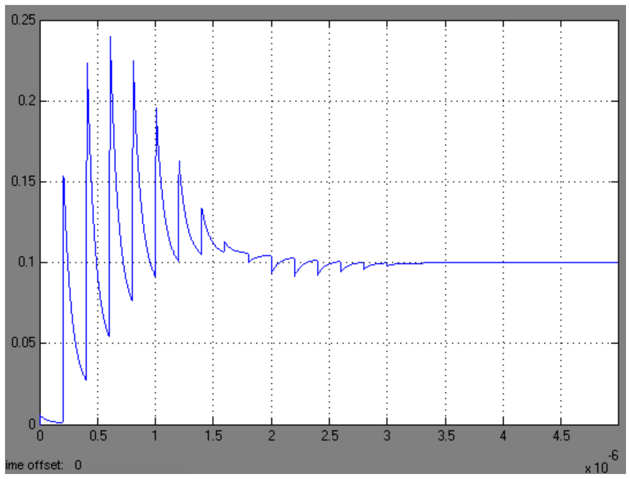
(d) N divider

Figure 3 PLL Each Block configuration

Figure 2 and 3 shows PLL implementation for time domain analysis.



(a)



(b)

Figure 4 a) The output of CP and b)Controle Voltage of VCO

Figure 4 shows charge pump output and loop filter output. Initial VCO free runing frequency was set 2.3GHz (Kvco = 1GHz/V). Due to Feedback loop, controle voltage converge to 0.1V.

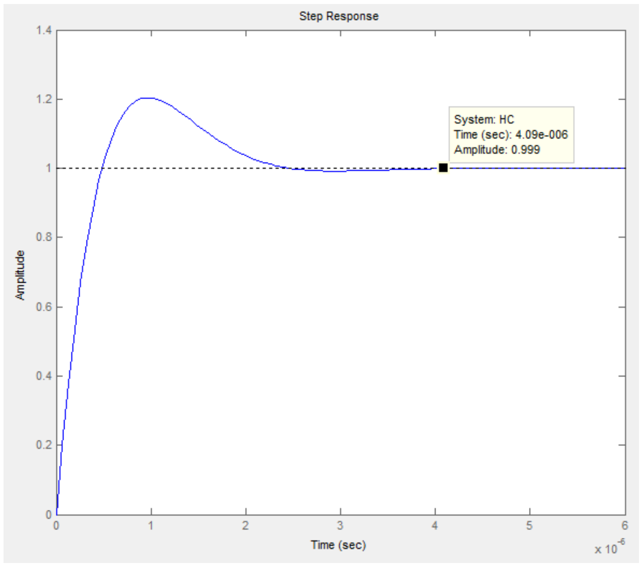


Figure 5 Closed Loop Step Response

Figure 5 shows that settling time for 0.1% accuracy is 4.2 µsec and it is closed to calculated value of 4.8 µsec.

Frequency Domain Analysis by Matlab

Figure 6 is the magnitude and phase response plot. It shows the two poles at zero frequency (phase is -180) and there are zero and pole, corresponding to peak at phase response.

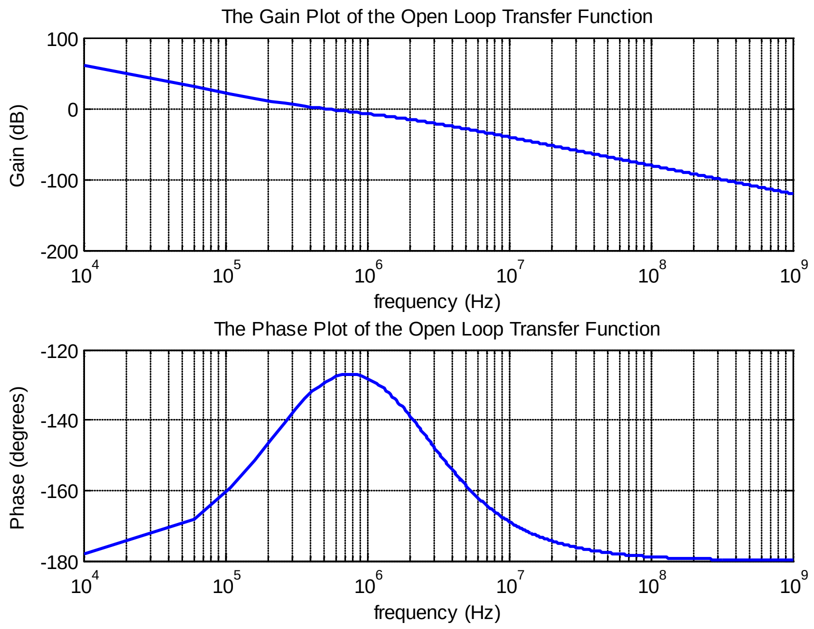


Figure 6 Open Loop Transfer function

Phase Margin = 51 deg, fzero = 253KHz, fcross=510KHz, and fpole(max)=2.25MHz

B. PLL RMS Jitter Calculation Based on 3 different input noise (part c)

1. Noise Contributors in a PLL

All the blocks of the PLL (shown in Figure 1) contribute towards random noise. But some contribute it more than others. The contribution of noise by the divider, PFD and charge pump is very minimal. One reason for this is that, once the PLL has locked, the PU and the PD signals driving the charge pump are on for a very short duration.

On the other hand, the Loop Filter resistance and the VCO contribute the most noise. One more thing to keep in mind is that the VCO output noise is high pass filtered, while the (PFD + Charge pump) input referred noise is low pass filtered to the output. So these blocks contribute to different components of jitter. While the Loop Filter and VCO contribute significantly to short term jitter, the PFD and Charge Pump contribute to long term jitter.

2. Jitter

Jitter is nothing, but the time domain equivalent of phase noise. There are essentially two ways of characterizing jitter: long term (or cycle jitter) and short term (or cycle to cycle jitter). Based on the application, one or both might be important. Both random and deterministic noise, contribute to long term and short term jitter. If the application is serial communication, where one is more interested in bit error rate, then long term jitter is a proper characteristic. If the application is digital signal processing, then short term jitter is of more use, as it would decide the clock period and timing margins.

3. Definitions of Jitter

We consider the output voltage Vout(t) of an oscillator in the steady state.

$$Vout(t) = A(t)f[\omega_o t + \theta(t)]$$

The time point of the *n*th zero crossing of Vout(t) is referred to as tn. The *n*th period is then defined as Tn = tn+1 - tn. For an ideal oscillator, this time difference is independent of *n*, but in reality it varies with *n* as a result of noise in the circuit. This results in a deviationΔTn = Tn - T from the mean period T. The quantity ΔTn is an indication of jitter.

More specifically, absolute jitter or long-term jitter

$$\Delta Tabs(N) = \sum_{n=1}^{N} \Delta Tn$$

is often used to quantify the jitter of phase-locked loops. Modeling the total phase error with respect to an ideal oscillator, absolute jitter is nonetheless ill suited to describing the performance of *oscillators* because, the variance of ΔTabs diverges with time.

A better figure of merit for oscillators is cycle jitter, defined as the rms value of the timing error Δ Tn

$$\Delta Tc = \lim_{n\to\infty} \sqrt{\left(\frac{1}{N}\sum_{n=1}^{N} \Delta T n^2\right)}$$

Cycle jitter describes the magnitude of the period fluctuations, but it contains no information about the dynamics.

The third type of jitter is cycle-to-cycle jitter given by

$$\Delta Tcc = \lim_{n\to\infty} \sqrt{\left(\frac{1}{N}\sum_{n=1}^{N} (T_{n+1} - T_n)^2\right)}$$

Note the difference between the cycle jitter and the cycle-to-cycle jitter: the former compares the oscillation period with the *mean* period and the latter compares the period with the *preceding* period. Hence, in contrast to cycle jitter, cycle-to-cycle jitter describes the short term dynamics of the period. The long-term dynamics, on the other hand, are not characterized by cycle-to-cycle jitter. With respect to the zero crossings, the cycle-to-cycle jitter is a double-differential quantity in that three zero crossings of the output voltage is related to each other. We should note that an oscillator embedded in a phase-locked loop periodically receives correction pulses from the phase detector and charge pump, and hence its long-term jitter strongly depends on the PLL dynamics. Thus, for the analysis of a free-running oscillator, cycle jitter and cycle-to-cycle jitter are more meaningful, particularly because the latter type hardly changes when the oscillator is placed in the loop.

4. Frequency Domain Jitter Analysis [1]

Relationship between timing jitter and noise power spectral density

The relationship between the timing jitter, $\sigma_{\Delta T}^2$ and noise power spectral density, $S_\phi(f)$ is

$$\sigma_{\Delta T}^2 = \frac{8}{\omega_0^2}\int_0^{00} S_\phi(f)\sin^2(\pi f \Delta T)df$$

At long cycles ($\Delta T \to oo$), the expression is simplified as :

$$\sigma_{\Delta T}^2 = \frac{4}{\omega_0^2}\int_0^{00} S_\phi(f)df$$

ΔT = cycles * 1/(output frequency)

Timing jitter is called short-term jitter for small ΔT and long term jitter as ΔT goes to infinity.

Noise Transfer Function 3-order PLL

Low Pass Transfer-Function from input to Output

$$\mathrm{H}_{\mathrm{Low-Pass}} = \frac{\mathrm{NKvcoIcp}}{2\pi}\left(\frac{1+\mathrm{RzCzCps}}{\mathrm{NRzCzCps}^3+\mathrm{N(Cz+Cp)s}^2+\frac{\mathrm{KvcoIcp}}{2\pi}\mathrm{RzCzs}+\frac{\mathrm{KvcoIcp}}{2\pi}}\right)$$

High Pass Transfer Function from VCO output to Output

$$\mathrm{H}_{\mathrm{High-Pass}} = \mathrm{N}\left(\frac{\mathrm{RzCzCps}^3+\mathrm{(Cz+Cp)s}^2}{\mathrm{NRzCzCps}^3+\mathrm{N(Cz+Cp)s}^2+\frac{\mathrm{KvcoIcp}}{2\pi}\mathrm{RzCzs}+\frac{\mathrm{KvcoIcp}}{2\pi}}\right)$$

Band Pass Transfer Function from VCO input to Output

$$\mathrm{H}_{\mathrm{Band-Pass}} = \mathrm{KvcoN}\left(\frac{\mathrm{RzCzCps}^2+\mathrm{(Cz+Cp)s}}{\mathrm{NRzCzCps}^3+\mathrm{N(Cz+Cp)s}^2+\frac{\mathrm{KvcoIcp}}{2\pi}\mathrm{RzCzs}+\frac{\mathrm{KvcoIcp}}{2\pi}}\right)$$

Crystal Noise

$$V_{crystal,out}^2 = \left(H_{lowpass}\right)^2 V_{n,crystal}^2$$

VCO output Noise

$$V_{VCO,out}^2 = \left(H_{Highpass}\right)^2 V_{n,vco\_out}^2$$

Resistor Noise in Loop Filter

$$V_{Resistor,out}^2 = \left(H_{bandpass}\right)^2\left(\frac{4KT}{Rz}\right)\left(\frac{RzCz}{RzCzCps+(Cz+Cp)}\right)^2$$

Figure 7 shows the noise transfer fuctions from the input phase to the output phase,the noise transfer fucntions from VCO input to output, and clock buffer noise.

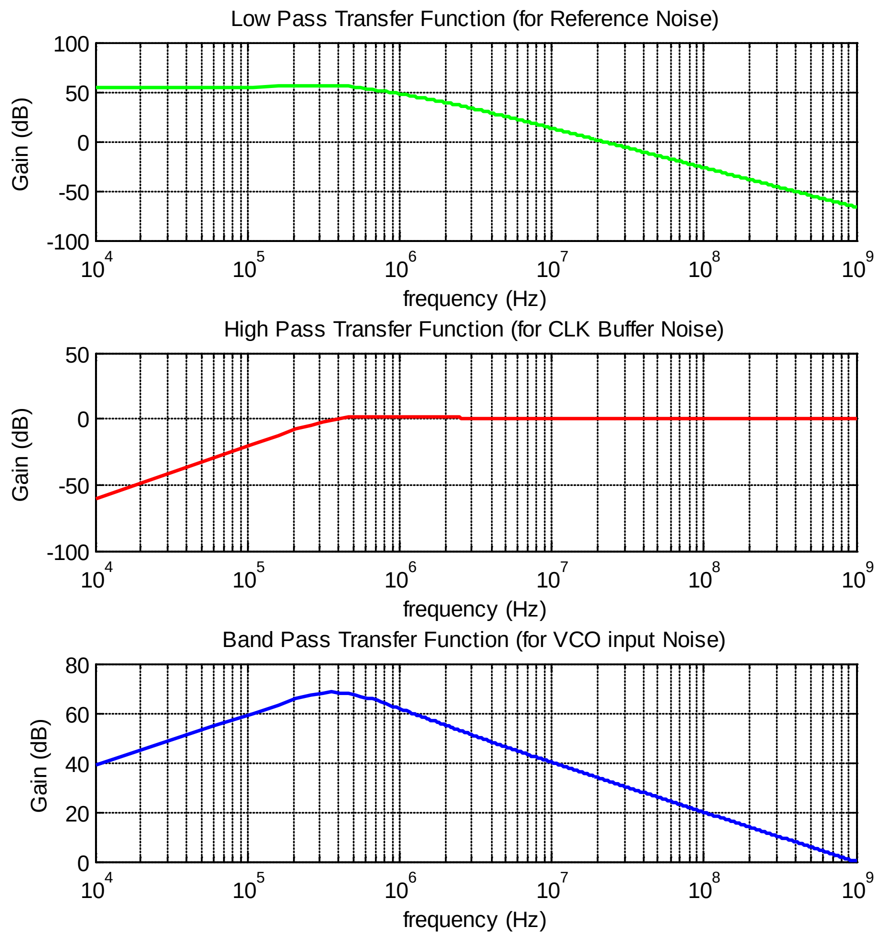


Figure 7 Noise Transfer Functions

Jitter Estimation by applying Effective 2nd-order Model to Any PLL[1]

Although a complete 3rd-order model of a PLL is needed to understand the jitter contribution of different loop parameters, it will be analysis by 2nd-order PLL model for simplification. In addition, the reference mention that analytical results and measurements have found that tracking jitter due to VCO noise for a particular design can be easily estimated by simply using the second-order equations.

The critical parameters that determine the jitter are the f-3dB and the peaking in the NTF.
In a higher-order loop, the parameters such as ζ and fn cannot be directly applied to the equations for the second-order loop because the resulting frequency response can differ greatly. To still use the equation, for a given frequency response, **we find an effective fn and effective ζ that result in the same bandwidth and peaking.**

The NTFs for VCO output (clock buffer noise) is high-pass filters while the NTF for input clock noise is a low-pass filter. In addition, the NTFs for VCO input is band-pass filters. Multiplying each noise source's NTF with the transfer function of the correspondent block provides the overall transfer function from any voltage (or current) noise to the PLL output:

5. Simulation Result

A. Relationship Between Output Jitter and Input Clock Noise

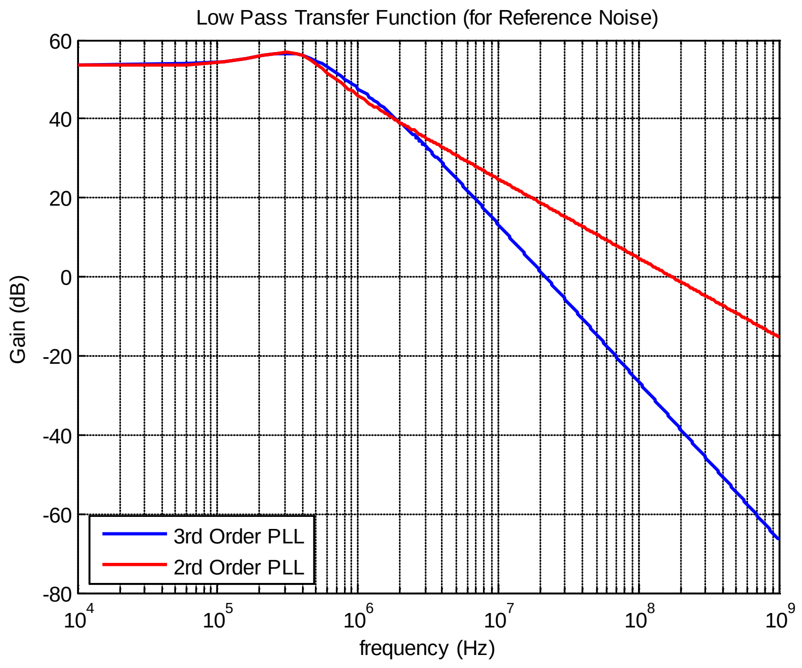


Figure 8 input referred NTF of 3rd order and 2rd order PLL

First find an effiective wn and effective damping factor that result in the same bandwidth and peaking. Wn = 2.3e6 rad/s and Damping factor = 0.5 were found by optimization based on Figure 8. It shows the peaking and bandwidth matches. There are two kinds noise source for wite noise and 1/f^2 noise source.

2rd Order NTF for input noise

$$H_{Lowpass} = N_{div}\left(\frac{2\xi\omega_n s + \omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2}\right)$$

$$\omega_n = \sqrt{\frac{K_{Loop}}{N}}, \xi = \frac{RzCz}{2}\omega_n, and \ \mathrm{K}_{loop} = K_{PFD}K_{VCO}I_{CP} / (2\pi Cz)$$

White Noise Assumption : $S_{\phi nin}(f) = N_{clk-in}$

$$\sigma^2_{\Delta T\to\infty} = \frac{2}{\omega_o^2} N_{clk-in}\left(\omega_n \frac{4\xi^2+1}{4\xi}\right)$$

$$RMS\ jitter = \sqrt{\sigma^2_{\Delta T\to 00} * Ndiv}$$

Ndiv=480, Fout=2.4GHz, and Nclk_in = 1e-6

RMS jitter = 311ns for infinite cycles by white noise

1/f^2 Noise Assumption : $\mathrm{S}_{\phi\mathrm{nin}}(\mathrm{f}) = \frac{\mathrm{N}_{\mathrm{clk-in}}}{\mathrm{f}^2}$

Timing Jitter conversion (This example Damping factor less than 1)

$$\sigma^2_{\Delta T} = \underbrace{\kappa^2\cdot\Delta T}\cdot\begin{cases} 1+\dfrac{1}{2\zeta\omega_n\cdot\Delta T}+\dfrac{e^{-\zeta\omega_n\Delta T}}{\Delta T}\cdot\left(\dfrac{\sin(\omega_d\Delta T+\theta)}{2(1-\zeta^2)\omega_n}-\dfrac{\cos(\omega_d\Delta T)}{2(1-\zeta^2)\zeta\omega n}-\dfrac{2\sin(\omega_d\Delta T)}{\omega_d}\right) & \zeta<1 \\ 1+\dfrac{1}{2\zeta\omega_n\cdot\Delta T}+\dfrac{e^{-a\Delta T}}{\Delta T}\left(\dfrac{2\alpha}{a}-\dfrac{2\alpha\beta}{a+b}-\dfrac{\alpha^2}{a}\right)+\dfrac{e^{-b\Delta T}}{\Delta T}\left(\dfrac{2\beta}{b}-\dfrac{2\alpha\beta}{a+b}-\dfrac{\beta^2}{b}\right) & \zeta\ge 1 \end{cases}$$

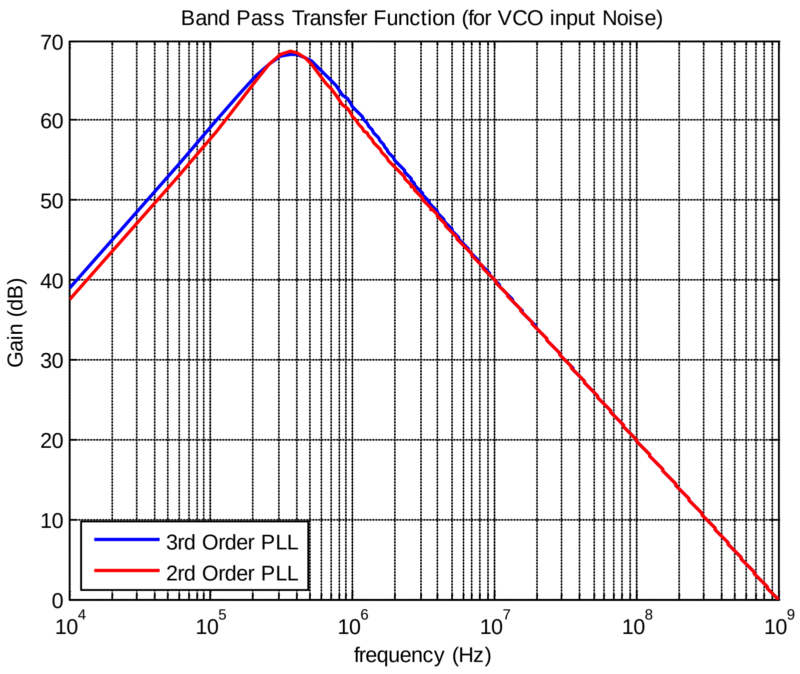
$$k^2 = \frac{4\pi N_{VCO}N_{div}}{\omega_o^{\ 2}}, and\ \ \omega_d = \omega_n\sqrt{1-\xi^2}$$

Delta T = 10000*1/2.4GHz

Nclk_in = 1e-6

Timing Jitter RMS = 0.2ps

B. Relationship Between Output Jitter and VCO Noise



First find an effiective wn and effective damping factor that result in the same bandwidth and peaking.
Wn = 2.3e6 rad/s and Damping factor = 0.5
VCO input noise NTF for 2-rd order PLL

$$H_{Bandpass} = \left(\frac{K_{VCO} s}{s^2 + 2\xi\omega_n s + \omega_n^2} \right)$$

$$\text{where } \omega_d = \omega_n \cdot \sqrt{1-\zeta^2} \text{ and } \cos\theta = \sqrt{1-\zeta^2}$$

$$\sigma_{\Delta T}^2 = \frac{4\pi^2 N_{VCO}}{\omega_0^2} \cdot \begin{cases} \dfrac{1}{2\zeta\omega_n} + \dfrac{e^{-\zeta\omega_n \Delta T}}{2(1-\zeta^2)} \cdot \left(\dfrac{\sin(\omega_d \Delta T + \theta)}{\omega_n} - \dfrac{\cos(\omega_d \Delta T)}{\zeta\omega_n} \right) & \zeta < 1 \\ \dfrac{1}{2\zeta\omega_n} - e^{-a\Delta T}\left(\dfrac{2\alpha\beta}{a+b} + \dfrac{\alpha^2}{a} \right) - e^{-b\Delta T}\left(\dfrac{2\alpha\beta}{a+b} + \dfrac{\beta^2}{b} \right) & \zeta \geq 1 \end{cases}$$

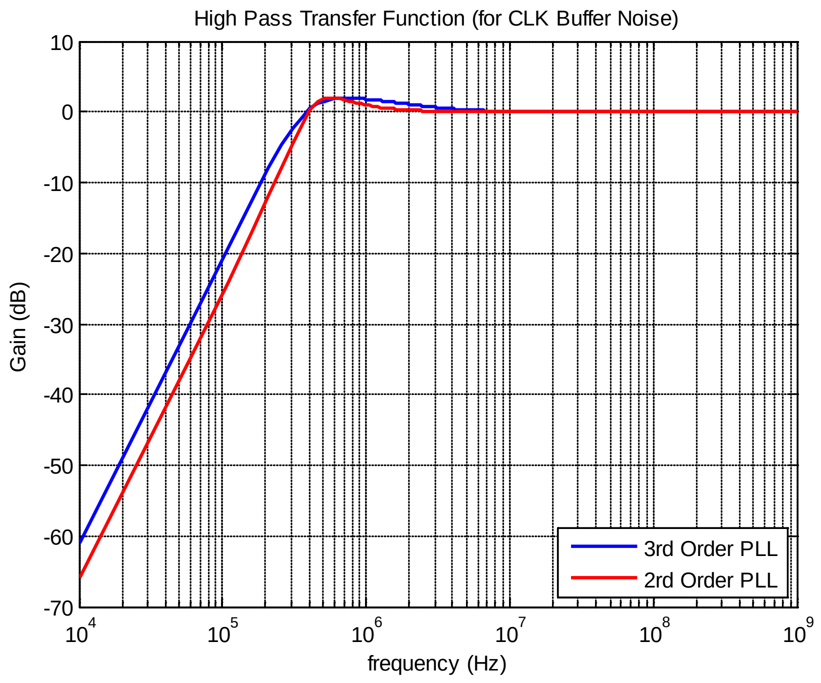
$$a, b = \zeta\omega_n \mp \omega_n \cdot \sqrt{\zeta^2 - 1}, \ \alpha = \frac{-a}{b-a} \text{ and } \beta = \frac{b}{b-a}.$$

For 10000 cycles

NVCO = 1e-3

Timing Jitter RMS = 8.7e-15s

C. Relationship Between Output Jitter and Clock Buffer Noise (VCO output noise)



First find an effiective wn and effective damping factor that result in the same bandwidth and peaking.

Wn = 2.8e6 rad/s and Damping factor = 0.44

For the clock buffer noise or VCO output noise NTF 2rd order PLL

$$H_{Highpass} = \left(\frac{s^2}{s^2 + 2\xi\omega_n s + \omega_n^2} \right)$$

This problem you are ingnore the clock buffer, therefore wbuf = 0 and Nbuf means VCO ouput noise.

Timing jitter equation

$$\sigma_{\Delta T}^2 = \frac{N_{Buf}}{\omega_0^2} \cdot \left(\begin{array}{ll} \left\{ \begin{array}{c} \omega_{Buf} + \omega_n \frac{1-12\zeta^2}{2\zeta} - e^{-\omega_{Buf}\Delta T}(\omega_{Buf} - 4\zeta\omega_n) \\ -e^{-\zeta\omega_n\Delta T}\left(-\frac{\omega_n \sin(\omega_d \Delta T + 3\theta - \pi)}{2(1-\zeta^2)} + \frac{\omega_n \cos(\omega_d \Delta T)}{2(1-\zeta^2)\zeta} - \frac{2\omega_n \sin(\omega_d \Delta T + 2\theta)}{\sqrt{1-\zeta^2}} \right) \end{array} \right\} & \zeta < 1 \\ \left\{ \begin{array}{c} \omega_{Buf} + \frac{\gamma^2}{a} + \frac{\upsilon^2}{b} + \frac{4\upsilon\gamma}{a+b} + \frac{4\omega_{Buf}\upsilon}{a+\omega_{Buf}} + \frac{4\omega_{Buf}\gamma}{b+\omega_{Buf}} - e^{-a\Delta T}\left(\frac{2\upsilon\omega_{Buf}}{a+\omega_{Buf}} + \frac{2\upsilon\gamma}{a+b} + \frac{\upsilon^2}{a} \right) \\ -e^{-b\Delta T}\left(\frac{2\gamma\omega_{Buf}}{b+\omega_{Buf}} + \frac{2\upsilon\gamma}{a+b} + \frac{\gamma^2}{b} \right) - e^{-\omega_{Buf}\Delta T}\left(\omega_{Buf} + \frac{2\upsilon\omega_{Buf}}{a+\omega_{Buf}} + \frac{2\gamma\omega_{Buf}}{b+\omega_{Buf}} \right) \end{array} \right\} & \zeta \geq 1 \end{array} \right.$$

where $\upsilon = \frac{2\zeta\omega_n a - \omega_n^2}{b-a}$ and $\gamma = \frac{-2\zeta\omega_n b + \omega_n^2}{b-a}$.

For 10000 cycles

NBuffer (VCO output noise) = 1e-6

Timing Jitter RMS = 55ps

Reference

[1] Mozhgan Mansuri “Low-Power Low-Jitter On-Chip Clock Generation” thesis UCLA

Appendix

```
% -----------------------------------------------------------------------
% ********************INITIAL SETUP********************

clear all;
format long;

% -----------------------------------------------------------------------
% ********************PLL PARAMETERS********************
f = linspace (10000, 1e9, 20000);
w = 2*pi*f;
% Input Clock Frequency
FIN = 5e6;
fs = 1/10e-12;

% Charge Pump parametere
ICP = 100e-6;
err_1 = 0;
err_2 = 0;
SR = 10e9; %Slew Rate is 1V in 100ps

% Loop Filter parameters
Rz = 15e3;
Cz = 42e-12;
Cp = 5.3e-12;


% KVCO parameters.
Kv = 1e9; %In Hz/V for simulink
KVCO = 1e9*2*pi; %In rad/V
KvcoOffset = 0;
Fo = 2.3e9;


% Divider Parameters
N = 480;

% -----------------------------------------------------------------------
% ********************START SIMULATION********************

options=simset('MaxStep',1/fs,...
    'RelTol',3e-3/fs,'AbsTol',1e-4/fs, ...
    'Solver','ode45',...
    'ZeroCross','on' );

sim('PLL_ab',[0 5e-6], options)
```

```
% ----------------------------------------------------------------------
% ********************POST PROCESSING********************

        Kpd = ICP/(2*pi);
        Kfwd = Kpd*KVCO;
        Kloop = Kfwd/N ;
        a1 = Rz*Cz;
        b21 = Rz*Cz*Cp;
        b22 = (Cp+Cz);

        Aopenloop_num = Kloop*[a1 1];
        Aopenloop_den = [b21 b22 0 0];
        Aopenloop = freqs(Aopenloop_num,Aopenloop_den,w);
        Gain_Open_Loop = db(abs(Aopenloop));
        Phase_Open_Loop = (180/pi)*angle(Aopenloop);

        Zero = abs(roots(Aopenloop_num)/(2*pi));
        Pole = abs(roots(Aopenloop_den)/(2*pi));
        BW = Kpd*KVCO*Rz/N*(Cz/(Cz+Cp))/(2*pi);
        DF = Rz/2*(Kpd*KVCO*Cz/N)^(1/2);

        [y x1] = min(abs(Gain_Open_Loop));
        Unity_Gain_Frequency = f(x1);
        Phase_Margin = Phase_Open_Loop(x1) + 180;
        Natural_freq = Unity_Gain_Frequency/(2*DF);
         figure(1)
         subplot(2,1,1)
         semilogx(f,Gain_Open_Loop,'Color','b','LineWidth',2);
         title('The Gain Plot of the Open Loop Transfer Function');
         xlabel('frequency (Hz)');
         ylabel('Gain (dB)');
         grid on;

         subplot(2,1,2)
         semilogx(f,Phase_Open_Loop,'Color','b','LineWidth',2);
         title('The Phase Plot of the Open Loop Transfer Function');
         xlabel('frequency (Hz)');
         ylabel('Phase (degrees)');
         grid on;

 % Closed Loop Analysis of the VCO (Low Pass Transfer Function)

        Alowpass_num = Kfwd*N*[a1 1];
        Alowpass_den = [N*b21 N*b22 a1*Kfwd Kfwd];
        Alowpass = freqs(Alowpass_num,Alowpass_den,w);
        Gain_Low_Pass = db(abs(Alowpass));

% Closed Loop Analysis of the VCO (High Pass Transfer Function from the VCO
% input to the output)

        Ahighpass_num = (N)*[b21 b22 0 0];
        Ahighpass_den = [N*b21 N*b22 a1*Kfwd Kfwd];
        Ahighpass = freqs(Ahighpass_num,Ahighpass_den,w);
        Gain_High_Pass = db(abs(Ahighpass));
```

```
% Closed Loop Analysis of the VCO (Band Pass Transfer Function from the VCO
% output to the output)

        Abandpass_num =  N*KVCO*[b21 b22 0];
        Abandpass_den = [N*b21 N*b22 a1*Kfwd Kfwd];
        Abandpass = freqs(Abandpass_num,Abandpass_den,w);
        Gain_Band_Pass = db(abs(Abandpass));

         figure
         subplot(3,1,1)
         semilogx(f,Gain_Low_Pass,'Color','g','LineWidth',2 );
         Title('Low Pass Transfer Function (for Reference Noise)');
         xlabel('frequency (Hz)');
         ylabel('Gain (dB)');
         grid on;
%
         subplot(3,1,2)
         semilogx(f,Gain_High_Pass,'Color','r','LineWidth',2);
         Title('High Pass Transfer Function (for CLK Buffer Noise)');
         xlabel('frequency (Hz)');
         ylabel('Gain (dB)');
         grid on;

         subplot(3,1,3)
         semilogx(f,Gain_Band_Pass,'Color','b','LineWidth',2);
         Title('Band Pass Transfer Function (for VCO input Noise)');
         xlabel('frequency (Hz)');
         ylabel('Gain (dB)');
         grid on;

%figure;

%semilogx(f,Gain_Low_Pass,'Color','g','LineWidth',2);
%hold on; grid on;
%semilogx(f,Gain_High_Pass,'Color','r','LineWidth',2);
%hold on; grid on;
%semilogx(f,Gain_Band_Pass,'Color','b','LineWidth',2);
%legend ('Gain Low Pass','Gain High Pass','Gain Band Pass',4);
%Title('Noise Transfer function, N=480' );
%         xlabel('frequency (Hz)');
%         ylabel('Noise Transfer function (dB)');
%         grid on;



wz = 2*pi*Zero ;
wp = 2*pi*max(Pole) ;
wn = 2*pi*Natural_freq ;
wd = wn*(1-DF^2)^0.5;

% figure;
% s = tf('s') ;
% HO = wn^2 * (1 + s/wz) / (s^2 * (1 + s/wp)) ;
% bode(HO) ;
% HC = wn^2 * (1 + s/wz) / (s^2 + ((wn^2)/wz)*s + wn^2) ;
% ltiview(HC) ;

%Jitter due to input ref Noise in an Ideal Second-Order PLL


        figure;
        semilogx(f,Gain_Low_Pass,'Color','b','LineWidth',2 );
         Title('Low Pass Transfer Function (for Reference Noise)');
         xlabel('frequency (Hz)');
         ylabel('Gain (dB)');
         hold on;
         grid on;

         wn_l = 2.3e6;
         df_l = 0.5;
         wd_l = wn_l*(1-df_l^2)^0.5;
         Num_l = N*[2*df_l*wn_l wn_l^2];
         den_l = [1 2*df_l*wn_l wn_l^2];
         NTF_LP=freqs(Num_l,den_l,w);
         Gain_LP=db(abs(NTF_LP));
         semilogx(f,Gain_LP,'Color','r','LineWidth',2);
         legend ('3rd Order PLL','2rd Order PLL',3);

fout = 2.4e9;
wout = 2*pi*fout;
cycles = 10000;
T = 1/fout;
d_T = cycles*T;
Nclk_in = 1e-6;
k_l = (4*pi^2*Nclk_in*N/(wout)^2)^0.5;

% Jitter due by 1/f^2 noise
RMS_jitter_ref2 = (k_l^2*d_T*(1+1/(2*df_l*wn_l*d_T)+exp(-
df_l*wn_l*d_T)/d_T*(sin(wd_l*d_T+pi/6)/(2*(1-df_l^2)*wn_l)-
cos(wd_l*d_T)/(2*(1-df_l^2)*df_l*wn_l)-2*sin(wd_l*d_T)/wd_l)))^0.5

% Jitter due by white noise and T => 00
RMS_jitter_ref=(N*2/(wout)^2*Nclk_in*(wn_l*(4*df_l^2+1)/(4*df_l)))^0.5




%Jitter due to VCO Input Noise in an Ideal Second-Order PLL

figure;
semilogx(f,Gain_Band_Pass,'Color','b','LineWidth',2);
         Title('Band Pass Transfer Function (for VCO input Noise)');
         xlabel('frequency (Hz)');
         ylabel('Gain (dB)');
         Hold on;
         grid on;
         wn_s = 2.3e6;
         df_s = 0.5;
         wd_s = wn_s*(1-df_s^2)^0.5;
         Num = [KVCO 0];
         den = [1 2*df_s*wn_s wn_s^2];
         second_band=freqs(Num,den,w);
         Gain_second_band=db(abs(second_band));
         semilogx(f,Gain_second_band, 'Color','r','LineWidth',2);
         legend ('3rd Order PLL','2rd Order PLL',3);



NVCO=1e-3;
k = (4*pi^2*NVCO/(wout)^2)^0.5;
Jitter_VCO=(k^2*(1/(2*df_s*wn_s)+(exp(-df_s*wn_s*d_T)/(2*(1-
df_s^2)))*(sin(wd_s*d_T+pi/2)/wn_s-cos(wd_s*d_T)/(df_s*wn_s))))^(1/2)

%Jitter due to VCOoutput(CLK Buff) Noise in an Ideal Second-Order PLL

figure;
semilogx(f,Gain_High_Pass,'Color','b','LineWidth',2);
         Title('High Pass Transfer Function (for CLK Buffer Noise)');
         xlabel('frequency (Hz)');
         ylabel('Gain (dB)');
         Hold on;
         grid on;

         wn_b = 2.8e6;
         df_b = 0.44;
         wd_b = wn_b*(1-df_b^2)^0.5;
         Num_b = [1 0 0];
         den_b = [1 2*df_b*wn_b wn_b^2];
         second_high=freqs(Num_b,den_b,w);
         Gain_second_high=db(abs(second_high));
         semilogx(f,Gain_second_high, 'Color','r','LineWidth',2);
         legend ('3rd Order PLL','2rd Order PLL',4);
wbuf = 0;
Nbuf = 1e-6;
Jitter_Buff=(Nbuf/wout^2*(wbuf+wn_b*(1-12*df_b^2)/(2*df_b)-exp(-
wbuf*d_T)*(wbuf-4*df_b*wn_b)-exp(-df_b*wn_b*d_T)*(-
wn_b*sin(wd_b*d_T+3*pi/6.9-pi)/(2*(1-df_b^2))+wn_b*cos(wd_b*d_T)/(2*(1-
df_b^2)*df_b)-2*wn_b*sin(wd_b*d_T+2*pi/6.9)/((1-df_b^2)^0.5))))^0.5
```