IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 27 May 2023, accepted 16 July 2023, date of publication 26 July 2023, date of current version 3 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3299227

RESEARCH ARTICLE

Increasing the Modeling Accuracy of an Analog PLL Device Executed With an Event-Driven Simulator

MARIAM MAURICE[1,2], MOHAMED DESSOUKY[2], AND ASHRAF SALEM[3]

[1]Siemens Digital Industries Software (DISW)—Siemens Electronic Design Automation (EDA), Cairo 75024, Egypt
[2]Electronics and Electrical Communications Engineering (EECE) Department, Ain Shams University (ASU), Cairo 11517, Egypt
[3]Computer and Systems Engineering (CSE) Department, Ain Shams University (ASU), Cairo 11517, Egypt

Corresponding author: Mariam Maurice (mariam.maurice@siemens.com)

This work is supported by Siemens Digital Industries Software (DISW)-Siemens Electronic Design Automation (EDA).

ABSTRACT Real Number Modelling (RNM) has become more common as a part of mixed-signal SoC validation. The paper illustrates modelling Phase Locked Loops (PLL) using SystemVerilog-Real Number Modelling (SV-RNM) as it's one of the essential blocks in any Integrated Circuit (IC) and a feedback loop system. It uses the Piece-Wise Linear (PWL) technique to model the loop filter with higher orders, higher than a first-order Low Pass Filter (LPF). The PWL technique needs both the value and the slope information so that the values between the samples can be interpolated, this is represented by the User-Defined Type (UDT) and Net (UDN). A fractional divider is modelled using the Sigma-Delta modulator of the Multi-stAge noiSe sHaping (MASH) 1-1-1 topology to generate the fractional part. Modelling non-linear effect like the phase noise of each sub-block, which is converted to the Root Mean Square (RMS)-jitter, by using the 'class' datatype that takes complex variables of real and imaginary values then returns some functionalities on these complex variables. Moreover, taking the loading effect due to capacitances and resistances at the output by using the User-Defined Resolved Nets (UDRN). The simulation results ensure that there is an accuracy improvement in the expected PLL outputs compared to the outputs from the transistor level with a much faster simulation time as an event-driven simulator is used.

INDEX TERMS Real number modeling (RNM), SystemVerilog (SV), user-defined types (UDT), user-defined resolved nets (UDRN), phase locked loops (PLL), fractional dividers, piece-wise linear (PWL), phase noise (PN), RMS jitter, hardware description and verification language (HDVL).

I. INTRODUCTION

Nowadays, both the analog and the digital system blocks are located on the same chip. Most digital design engineers want to verify the digital modelled blocks along with the analog models for ensuring the functionality of both systems together. There are many challenges during the mixed-signal design modelling as the simulation performance, the verification efficiency, and the modelling accuracy [1], [2]. Real Number Modelling (RNM) is the process of modelling the behavior of an analog circuit as discrete real data so it's a signal flow-based modelling approach [3], [4], [5]. This means that every output of an analog component is sampled, in a discrete manner, from the inputs and the internal states. The model detects an event and decides the time to carry out a computation. Therefore, no continuous time operation is considered, only sampled, clocked, or event-driven operations. The most familiar Hardware Description Language (HDL) to support RNM constructs in the digital environment is the 'real' datatype in SystemVerilog (SV) [6], [7], and the wire-real 'wreal' in Verilog-AMS [8], [9], and 'real' in VHDL-AMS [10].

There are two main advantages of using the RNM. The first one is that the Real Number Models are faster than

The associate editor coordinating the review of this manuscript and approving it for publication was Yue Zhang.

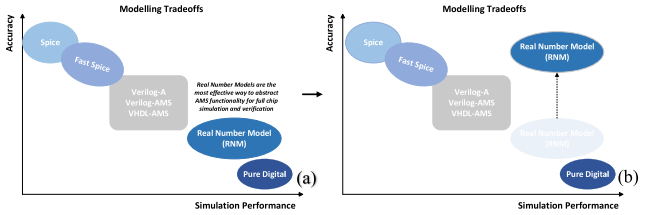


FIGURE 1. Modelling tradeoffs.

SPICE and VERILOG-AMS electrical models. The transistor netlist has the highest accuracy level, but it exhibits the lowest simulation time compared to the other approaches since it uses an analog solver to solve matrices. For the event-driven simulators, there are no matrices solving. Therefore, no matrix conversion as SPICE. Discrete event solver ''not analog solver'' therefore no convergence problems come into the picture. The simulation time of a system using the SPICE simulators takes hours or more than a day to simulate one system block at a typical corner [11], [12]. The classic analog modelling techniques are: (VHDL-AMS, Verilog-A, Verilog-AMS). These techniques reduce the actual simulation time, in comparison with SPICE simulations. But compared to the event-driven simulation, still slower as they are still using an analog solver. The simulation time of a complete system takes a couple of minutes [13], [14], [15], [16], [17].

The second advantage is that the Real Number Models are more accurate than the purely digital model. The pure logic model only represents four states (0,1, X, and Z) but RNM can model any state which boosts the simulation efficiency. There are ways to increase the accuracy of real number models as taking more discrete events by decreasing the time precision within the timescale. The more discrete events are taken, the more analog levels can be modelled (sampling the data with a much higher frequency). But this way could speed down the simulation and could be a tradeoff to the pros of the event-driven simulators. This paper will introduce new techniques that can even increase the accuracy without decreasing the timestep between events. These techniques provide modelling of the analog devices with higher accuracy and the output could be much equivalent to the output from the transistor netlist. These techniques define User defined Nets (UDN) that could hold the electrical values in just one net such as a net that holds the electrical values of voltage, current, resistance, and capacitance. Moreover, the PWL technique is where the UDN holds the electrical value of the signal, slope, and time [18], [19], [20], [21], [22].

Before the real number models are the best for the high-level abstraction of the analog devices but still their accuracy is not much comparable to the output from the transistor level that is simulated by a spice simulator as in Fig. 1(a). The paper introduces new RNM constructs that increase the modelling accuracy of the analog devices with maintaining high simulation performance as an event-driven simulator is used as in Fig. 1(b).

A SystemVerilog behavioral RNM for a PLL is presented to describe the functionality of each sub-block of the PLL as

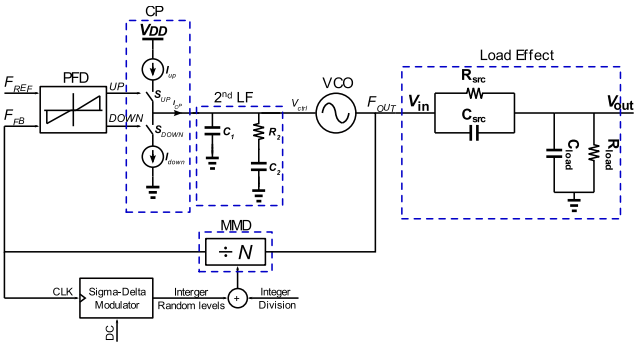


FIGURE 2. 2nd-order fractional PLL structure with a load effect.

TABLE 1. PLL system specifications and requirements.

System Parameter	System_1	System_2
Reference Clock (Hz)	100M	200M
Charge Pump Current (A)	50u	10u
VCO Gain(rad/s/v)	$2\pi * 1E9$	$2\pi * 7E9$
Fractional Divider Levels	47-54	22-29
Loop Bandwidth (rad/s)	2π * 1E6	2π * 4E6
PM (degree)	60	60
Tuning Range (GHz)	4.5-5.5	2-9
C1(pF)	10.6	6
C2(pF)	138	76.8
R2(KΩ)	10.8	2

its circuit nature such that, it will be divided according to its input/output port data type [18]. While the structural model describes each sub-block as its gate function. Moreover, this model takes into consideration the effect of transistors on the output of each sub-block. Therefore, it models non-linear effect like the phase noise of each sub-block and then converted it to the RMS-jitter (seconds) which can be defined as a delay in the output event and the loading effect of each sub-block especially at the PLL output as it will be connected to another loaded circuit. Here, it does not look at the datatype of the input/output port and will model all the ports as of 'real' datatype since there are a lot of analog effects that will be modelled. Fig. 2. represents the second-order Fractional PLL structure by taking the effect of the loading. The system specifications and requirements are represented in Table 1. System_1 serves modelling the VCO block by using an LC-oscillator while System_2 by using a Ring-oscillator.

II. PLL MODEL

A. PHASE FREQUENCY DETECTOR (PFD)

The datatype of the input and the output ports of the PFD is a 'logic' datatype so the behavioral implementation of the PFD is about tracking the information at the transition instant of the input ports. The idea of the implementation is about finding the time difference between the positive edge events of the two input clocks. In this period the output should be equal to '1' else the output is equal to '0'. The output will be named 'phase'. Firstly, the positive edge transition of the input clocks will be tracked using 'event' datatype so now

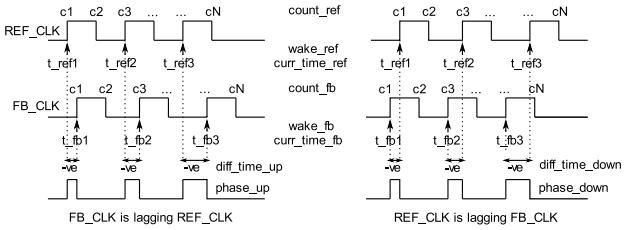


FIGURE 3. Behavioral waveforms description of PFD.

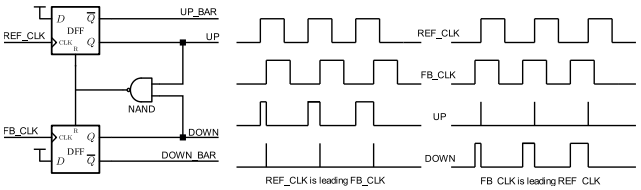


FIGURE 4. Tristate PFD and its operation.

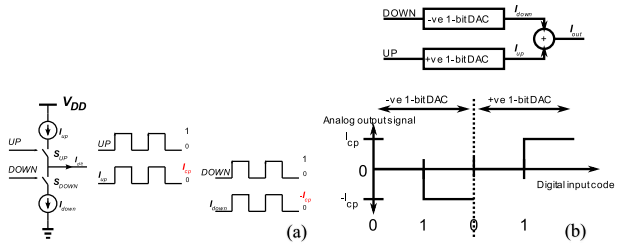


FIGURE 5. (a). Charge pump operation (b). Charge pump circuit & Transfer function of DACs.

TABLE 2. DAC.

DAC	Logic Input	Analog Output
-ve 1-bit DAC Input logic DOWN Output Analog I_{down}	DOWN = 0	$I_{down} = 0$
	DOWN = 1	$I_{down} = -I_{cp}$
+ve 1-bit DAC Input logic UP Output Analog I_{up}	UP = 0	$I_{up} = 0$
	UP = 1	$I_{up} = I_{cp}$

there is information when the input data is changed from low to high. The event where the reference clock changed from low to high will be named 'wake_ref' and the time where the reference clock changed from '0' to '1' will be named 'current_time_ref'. Similarly, the feedback clock will be named 'wake_fb' and 'current_time_fb' where the feedback clock changed from '0' to '1'. Then counting any event for both clocks whatever the data is changed from '0' to '1' or from '1' to '0'. The counters will be named 'count_ref' which counts any transition of the reference clock and 'count_fb' which counts any transition of the feedback clock. Knowing the number of counts will help to know which input data is leading the other. When the count of any counter exceeds the other then the difference between the input clock events can be calculated. The lagging clock should be subtracted from the leading clock this is done when the lagging clock event is subtracted from the leading clock event. If the time difference is equal to a -ve value this means that the next positive edge of the lagging signal clock is still not occurred at the current time, there is only one transition event from '0' to '1' which is the leading signal clock. Therefore, the difference is equal to a -ve value starting from the current time of the positive edge of the leading clock until the positive edge of the lagging clock is occurred. The phase difference between the two clocks could simply be equal to '1' when the time difference between the lagging clock and the leading clock is equal to a -ve value as illustrated in Fig. 3. In the case of the feedback clock is the lagging the reference clock, this means that the counts of the 'count_ref' are greater than the counts of the 'count_fb' then calculate the difference between the 'current_time_fb' and the 'current_time_ref' starting from the 'wake_ref' until the positive edge of the 'fb_clk' is occurred. If the difference 'diff_fb_ref' is equal to a -ve value, then 'phase_up' is equal to '1' else the 'phase_up' is equal to '0'. Similarly, if the counts of the 'counter_fb' are greater than the counts of the 'count_ref' then the difference between the 'current_time_ref' and the 'current_time_fb' starting from the 'wake_fb' until the positive edge of the 'ref_clk' is occurred. If the difference 'diff_ref_fb' is equal to a -ve value, then the 'phase_down' is equal to '1' else the 'phase_down' is equal to '0'.

The structural implementation of the PFD is about choosing a topology for implementing the PFD circuit. The most familiar topology is the tristate PFD [23], [24]. The tristate PFD is essentially consisting of two resettable D-FF and one NAND gate as illustrated in Fig. 4.

B. CHARGE PUMP (CP)

The UP and DOWN pulses generated from the PFD, control the charge pump current that will flow in the pull-up and the pull-down networks. Such that if the UP signal is equal to one logic value, UP = 1, the S_{UP} is closed and the output current I_{out} will equal to the charge pump pull-up current I_{cp}. Similarly, if the DOWN signal is equal to one logic value, DOWN = 1, the S_{DOWN} is closed and the output current I_{out} will equal the charge pump pull-down current $-I_{cp}$. If the UP signal is zero logic value, UP = 0, then I_{out} will be equal to zero. If the DOWN signal is zero logic value, DOWN = 0, then I_{out} will be equal to zero. Briefly, the analog current output value is equal to the desired positive current value, I_{cp}, at the rising edge of the UP signal. The analog current output value is equal to the desired negative current value, -I_{cp}, at the rising edge of the DOWN signal as illustrated in Fig. 5(a). Since the datatype of the input ports of the CP is a 'logic' datatype and the output ports of the CP are a 'real' datatype. Therefore, the charge pump current can be treated as a two 1-bit Digital to Analog Converter (DAC). The first DAC converts a digital input to a negative analog output while the second DAC converts a digital input to a positive analog output as illustrated in Table 2.

Fig. 5(b). illustrates the transfer function graph that relates the analog output with the digital input and a simple circuit representation to the charge pump circuit that consists of two DACs and an adder to add the output currents from both DACs. Only one DAC is ON at the present time either at the rising edge of the UP signal or at the rising edge of the DOWN signal. One of the two currents, I_{up}&I_{down}, will have

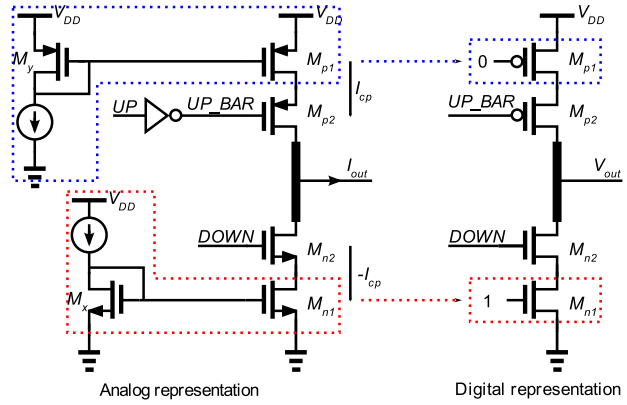


FIGURE 6. Conventional single ended charge pump circuit.

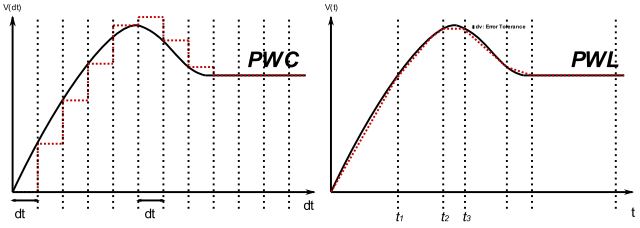


FIGURE 7. PWC/PWL.

a real flow current value while the other will be equal to zero. At this moment, the output current will be equal to I_{cp} if the positive DAC is ON and $-I_{cp}$ if the negative DAC is ON.

If the need is to implement a specific topology for the charge pump circuit like the one illustrated in Fig. 6. for a more structural model. The conventional single ended charge pump circuit depends on the mirroring current from M_x and M_y, then M_{n1} and M_{p1} act now as current sources [25]. The current mirror circuit could be treated in the digital environment as a closed switch or an ON transistor. Where the p-mos current mirror circuit is an ON pull-up transistor, the drain has a low logic value ('0) and the source has a high logic value ('1). The n-mos current mirror circuit is an ON pull-down transistor, the drain has a high logic value ('1) and the source has a low logic value ('0). The current sources are with the same concept in the digital environment where M_{n2} is ON when the DOWN signal is high and M_{p2} is ON when the NOT UP signal is low.

Then the output voltage, V_{out}, will be converted to the output current according to the value of the output voltage. If the output voltage is equal to '1' which means that the transistors of the pull-up network are on, then the output current is equal to I_{cp}. If the output voltage is equal to '0' which means that the transistors of the pull-down network are on, then the output current is equal to $-I_{cp}$.

C. LOOP FILTER (LF)

Most digital designers use the Piece-Wise Constant (PWC) technique to model the LF that depends on sampling the data at a constant delta time [20], [21], [22], [23], [24], [25], [26]. Therefore, the events of the data are defined at specific times such that the time difference between the current data and the previous data is constant delta time (ΔT) as illustrated in Fig. 7(a). To prevent aliasing of the data, the Nyquist condition is applied which means that the data is sampled with a very high rate greater or twice the rate of the input signal to prevent the loss of the information. Since the PWC technique could be inefficient in modelling the RF analog systems even if the data is sampled at a very high rate. That's because it forces the entire analog system block, like the LF block, to be modelled at a single clock rate which is the rate of the Nyquist condition and does not take into consideration the effect of an event that occurs in a time less than the delta time like timing jitter. Even if the digital designer thinks to increase the rate of sampling the data with a clock rate of femtosecond this will lead to slow down the simulation as the whole computation of the system should be occurred each femtosecond.

Therefore, the PWC technique could be just a behavioral model for modelling the LF and it is a way to model the functionality of the block but if the digital designer needs the output signal to be updated only when the input signal is changed or wants to take into consideration more effects like timing jitter then the PWL technique could be used [18], [19], [20], [22], [23], [24], [25], [26], [27]. The PWL technique is used to model the data by sampling the data and interpolating linearly between the sampled points. Instead of evaluating the data at a fixed rate, the analog output event is completely event-driven and the output time-step depends on the given error tolerance in the magnitude of the analog output as illustrated in Fig. 7(b). The error tolerance in the analog output is defined by the error that is expected to be in the analog amplitude due to the non-idealities in the block. The smaller the error tolerance, the more the designer is towards the ideal case model for that block. If the error tolerance has a large value, then the next time step could be far from the previous one to be reached. This means that the next time step according to the previous one, will depend on the error tolerance in the magnitude of the analog output. Therefore, the error tolerance could be defined as the maximum tolerance in the analog amplitude. If the error tolerance is reached, then the next time is defined compared to the previous one. This allows the effect of sampling clock jitter to be computed even if the input signal has a non-constant time step. The datatype of the input and the output ports of the LF is a 'real' datatype. The analog 'real' datatype input is filtered with a transfer function to produce the 'real' datatype output. Since the passive filters are preferred to be used more than the active filters except for the applications that require the VCO tuning voltage to be higher than what the charge pump can provide and with higher orders to reject the noise. Part (III.) will illustrate the PWC and the PWL techniques for a second-order loop filter. These techniques could be applied to any loop passive filter type as far as the designer could calculate the transfer function of the loop filter and could find a relation between the input and the output.

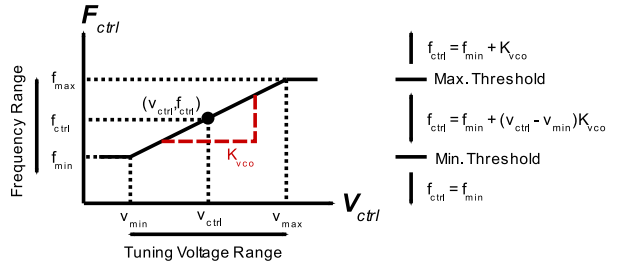


FIGURE 8. Calculation of fctrl according to relation of fctrl versus vctrl.

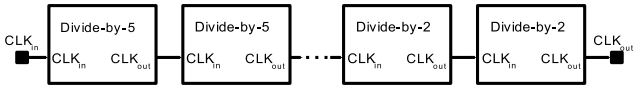


FIGURE 9. Divider consists of divide-by-2 and divide-by-5 Cells.

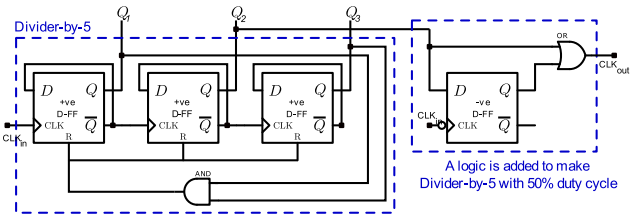


FIGURE 10. Divider-by-5 and logic to make 50% duty cycle.

D. VOLTAGE CONTROLLED OSCILLATOR (VCO)

For the ideal VCO, the output frequency of the VCO is a linear function of its control voltage inside the tuning voltage range. The maximum controlled voltage gives the maximum frequency and above this voltage, the frequency is saturated at the maximum frequency. The minimum controlled voltage gives the minimum frequency and below this voltage, the frequency is saturated at the minimum frequency [28], [29]. The tuning range is dedicated to the needed frequency range of the application and the variation of the required oscillation frequency due to process and temperature. Therefore, the edges of the tuning voltage range could be defined as maximum and minimum thresholds. Between these thresholds, the oscillation output frequency is a linear function of the voltage control ($f_o = f_{ctrl} = K_{vco} v_{ctrl} + f_{min}$) as illustrated in Fig. 8.

The output frequency is then converted to square pulses as the output frequency will be an input to the divider sub-block. The periodic time of the square pulses is simply calculated as a division of one by the control frequency (1/fctrl). Half of the period is equal to '1' and the other half period is equal to '0'.

E. INTEGER DIVIDER

The integer divider (N) that is divisible by five could be implemented as cascaded cells of divider-by-5 and divider-by-2 with the following equation. Where 'a' is the number of the cascaded divider-by-5 cells and 'b' is the number of the cascaded divider-by-2 cells. For example, if the integer number is 100 then two divider-by-5 followed by two divider-by-2 cascaded cells as illustrated in Fig. 9.

$$N = (5 * a) * (2 * b) \tag{1}$$

Any divider circuit could be implemented as cascaded flip-flops. The desired number of flip-flops is calculated according to 2^n. Where (n) is the number of flip-flops. Therefore, divide-by-5 is reached by three cascaded flip-flops. Fig. 10, the used topology depends on a divide-by-5 counter. It counts from 0 to 4 and in the 6th count, the counter resets then it starts again to count from 0 to 4. The reset control signal resets all the flip-flop cells when the output of flip-flops becomes equal to 5 (binary equal to 101), which is the desired division ratio. Since the reset signal is required to be equal to one logic value at the 6th count then this can be done by ANDing the flip-flop outputs that have one logic value. In this case, the outputs that should be taken are $Q_1 \& Q_3$ as these outputs are equal to one logic value in the 6th count. This topology does not have a 50% duty cycle divide-by-5, it is high for two clock phases and low for three clock phases. A sequential logic is added to make the output clock frequency of the divider-by-5 cell (Q_2) have a 50% duty cycle.

F. FRACTIONAL DIVIDER

The concept of the fractional divider (frac-N) is toggling between two or more divisor integer level values (N + <-ve_level>, N, N + < +ve_level>), such that the effective fractional division ratio is achieved by time averaging the divider output. If the desirable division ratio is (N. $<\text{frac}_{\text{no.}}>$) then the fractional divider is controlled dynamically to divide by (N) for (1− $<\text{frac}_{\text{no.}}>$) % of the time and by (N + 1) for (< $\text{frac}_{\text{no.}}>$) % of the time. The time is the reference clock periodic time, and the total number of cycles is the expected output clock cycles per periodic time. Therefore, the division ratio of (N. $<\text{frac}_{\text{no.}}>$) is achieved on the average of the output clock cycles from VCO per reference clock cycle when toggling the division ratio between (N) and (N + 1).

$$\bar{N} = \frac{(N) \times No.\ of\ cycles_1 + (N+1) \times cycles_2}{Total\ No.\ of\ cycles\ per\ time} \tag{2}$$

where,

$$No.\ of\ cycles_1 = \frac{1 - <\text{frac}_{\text{no.}}>}{100} \times Total\ No.\ of\ cycles\ per\ time$$

$$No.\ of\ cycles_2 = \frac{<\text{frac}_{\text{no.}}>}{100} \times Total\ No.\ of\ cycles\ per\ time$$

To reach the above, the fractional-N divider consists of a programmable divider that is programmed to step size between its integer levels. The programmable divider is implemented by the Multi-Modulus Divider (MMD) topology. The MMD uses cascaded 2/3 divider cells. Each 2/3 divider cell is a programable divide-by-2 or divide-by-3 that can be implemented as illustrated in Fig. 11(a).

When the modulus control signal (Mod_{in}) is low, the two lower latches operate as if they do not exist and the output value of the lower positive latch ($\bar{Q}$) is always high. Therefore, the 2/3 divider cell divides the input clock frequency by two. Moreover, when the modulus control signal (Mod_{in}) is high, the output of the 2/3 divider cell is high for two clock

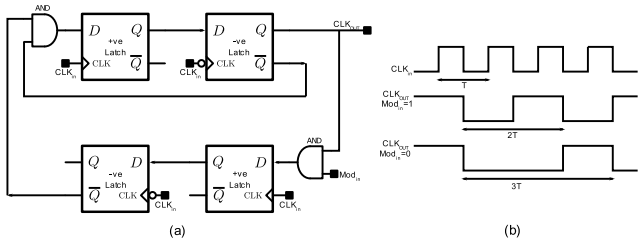


FIGURE 11. (a). Divider-by-2/3 circuit (b). Divider-by-2/3 circuit operation.

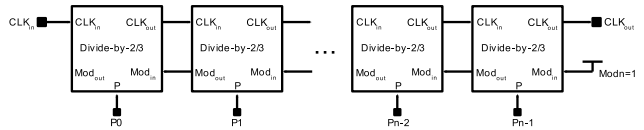


FIGURE 12. Programmable multi-modulus divider from cascaded divider-by-2/3 cells.

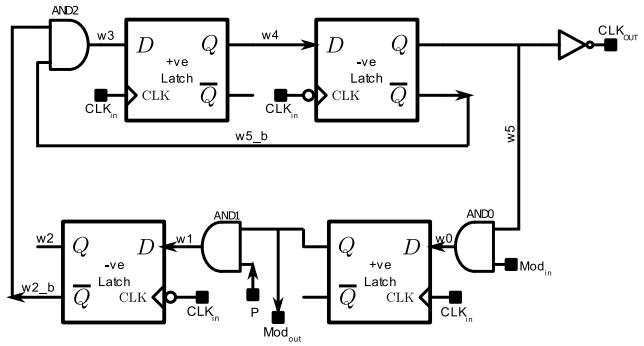


FIGURE 13. Divider-by-2/3 circuit after modification.

phases and low for four clock phases which results in dividing the input clock frequency by three as illustrated in Fig. 11(b). The MMD famous topology uses a cascaded 2/3 divider cell with an n-bit modulus control signal as illustrated in Fig.12. and the MMD division ratio is given by;

$$N = 2^n + P_{n-1} * 2^{n-1} + P_{n-2} * 2^{n-2} + \ldots + P_2 * 2^2 + P_1 * 2^1 + P_0 \quad (3)$$

where (n) represents the number of the cascaded 2/3 divider cells and the programmable divisor integer level is toggling between 2^n to $(2^{n+1} - 1)$. Since the cells are cascaded therefore the 2/3 divider cell in Fig. 11(a) should be modified to interface between the consecutive cells [30], [31], [32], [33]. An input (P) and an output (Mod_{out}) are added for the modification as illustrated in Fig.13.

The divider's modulus control bits (P0, …, Pn) should be changed periodically every reference clock cycle to toggle between the division integer levels (N + <-ve_level>, N, N + < +ve_level>) so that to keep the time-averaged division ratio at the desired fractional number. For every reference clock cycle, the division ratio is changed randomly from one of the desired division integer levels (N + <-ve_level>, N, N + < +ve_level>). Therefore, the feedback clock period is different in its value from one cycle to another which leads to a different phase difference for every reference clock cycle. The difference in this phase difference will inject current pulses in the loop filter which leads the

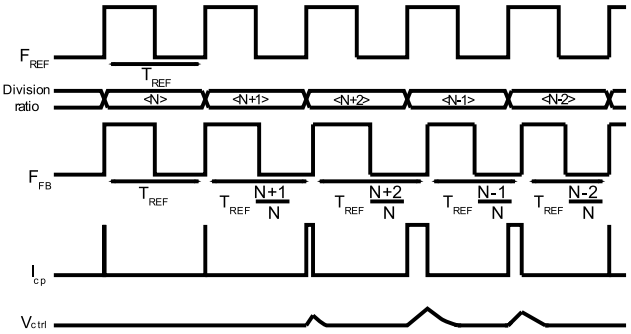


FIGURE 14. Effect of random division ratio at loop locking.

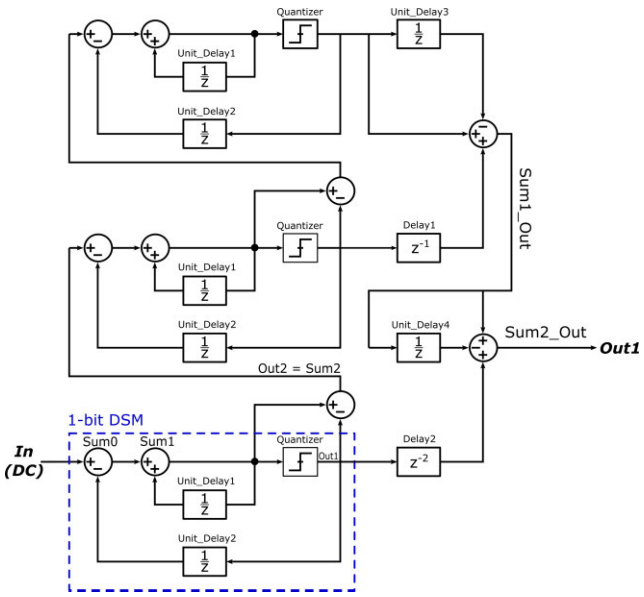


FIGURE 15. Third-order cascaded loop MASH 1-1-1 ΔΣ modulator.

VCO control voltage to oscillate with a small amplitude even at the steady state of the loop. These small amplitude oscillations in the control voltage will modulate the VCO output frequency and then generate undesirable spurs around the output oscillating frequency as illustrated in Fig. 14.

Random switching between the different division levels can break the periodicity of the loop behaviour with time and can result in undesirable phase jitter or spurs near the desired output frequency. The Sigma-Delta (ΔΣ) Modulator is used to generate these random levels (<-ve_level>, 0, < +ve_level>). Moreover, the ΔΣ Modulator performs a high pass filtering effect which moves the undesirable spurs and the in-band noise near the desired output oscillating frequency that results from the switching between different division levels. Then these noises at the high frequency will be filtered by the PLL loop filter which has a low pass filtering effect.

The ΔΣ Modulator is modelled by MASH 1-1-1 topology as illustrated in Fig. 15. The noise spectrum of the MASH 1-1-1 topology has a third-order high pass filtering effect. The order of the ΔΣ Modulator can be selected upon the system noise requirements, the ΔΣ Modulator with an order greater than two leads to more spurs reduction and more

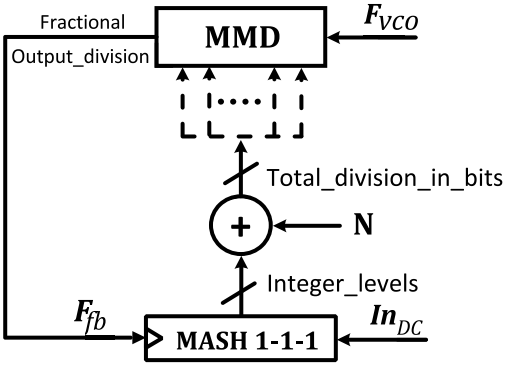


FIGURE 16. Fractional divider structure.

noise shaping therefore better noise performance. The ΔΣ Modulator of orders higher than three does not perform much better noise performance as illustrated by the linear theoretical model and in the real world, the improvement is not linear with the increasing of the ΔΣ Modulator. Therefore, a third order ΔΣ Modulator is always preferred [34], [35], [36].

The third-order ΔΣ Modulator can be implemented by a single-loop modulator technique or a multi-loop modulator technique. The single-loop ΔΣ Modulator includes integrators that their numbers equal to the ΔΣ Modulator order and one quantizer. The multi-loop technique depends on cascading the first-order ΔΣ Modulator, the number of the cascaded 1st-order ΔΣ Modulator depends on the ΔΣ Modulator order. Where the input to the first stage is a DC input voltage and the input to the other stages is the error introduced from the previous stage. This technique is called Multi-stAge noiSe sHaping (MASH). The MASH ΔΣ Modulator does not suffer from the stability issue compared to a single-loop ΔΣ Modulator as each stage of the MASH ΔΣ Modulator has one or two integrators that can cover its stability easily instead of covering a one loop using n/2n-integrators. For 1-bit ΔΣ Modulator, the integrator in the time domain is modelled as a unit delay. The quantizer could act as a comparator switch when the input is greater than zero then the quantizer output is equal to the highest value and when the input is smaller than zero then the output of the quantizer is equal to the lowest value. The Mash 1-1-1 is modelled with three cascaded first-order digital ΔΣ Modulator.

The desired division ratio (N) is then added to the integer levels generated randomly from Mash 1-1-1 then now the total division ratio is ranged from (N-3) to (N+4). The output of the total division should be in bits to be an input for the Multi-Modulus Divider (MMD) as illustrated in Fig. 16.

III. PWC/PWL

A. PIECE-WISE CONSTANT (PWC)

The PWC technique only models and provides the value of the signal. This value has an electric model discipline like the voltage, current, resistance, or capacitance. For LF, the interest is to model the control voltage. The input to the LF is the charge pump current and the output is the control voltage that will control the connected block to the LF output which

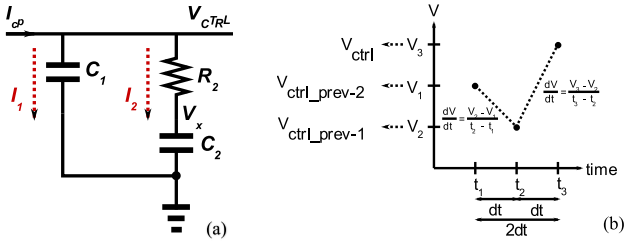


FIGURE 17. (a). Second-order LFP, current flow (b). Numerical second derivative in linear combination of three points.

is the VCO block. Therefore, the target is to find a relation between the input current and the output voltage. In this part, the relation depends on finding Kirchhoff's Voltage Law (KVL) in any closed network and Kirchhoff's Current Law (KCL) at each node. From Fig. 17(a). the charge pump current flows in the two branches and is equal to

$$I_{cp} = I_1 + I_2 = C_1 \frac{dV_{ctrl}}{dt} + \frac{1}{R_2}(V_{ctrl} - V_x) \tag{4}$$

Differentiate (4),

$$\frac{dI_{cp}}{dt} = C_1 \frac{d^2V_{ctrl}}{dt^2} + \frac{1}{R_2}\frac{dV_{ctrl}}{dt} - \frac{1}{R_2}\frac{dV_x}{dt} \tag{5}$$

Since I_2 could also be equal to $C_2 \frac{dV_x}{dt}$ then the charge pump current also is equal to

$$I_{cp} = I_1 + I_2 = C_1 \frac{dV_{ctrl}}{dt} + C_2 \frac{dV_x}{dt} \tag{6}$$

From (6), $\frac{dV_x}{dt}$ is equal to

$$\frac{dV_x}{dt} = \frac{1}{C_2} I_{cp} - \frac{C_1}{C_2}\frac{dV_{ctrl}}{dt} \tag{7}$$

Substitute (7) in (5)

$$\frac{dI_{cp}}{dt} = C_1 \frac{d^2V_{ctrl}}{dt^2} + \frac{1}{R_2}\frac{dV_{ctrl}}{dt} - \frac{1}{R_2C_2} I_{cp} + \frac{C_1}{R_2C_2}\frac{dV_{ctrl}}{dt} \tag{8}$$

The numerical second derivative of data could be represented in a linear combination of three points [46].

$$\frac{d^2V}{dt^2} = \frac{2V_1}{(t_2 - t_1)(t_3 - t_1)} - \frac{2V_2}{(t_3 - t_2)(t_2 - t_1)} + \frac{2V_3}{(t_3 - t_2)(t_3 - t_1)} \tag{9}$$

In the PWC, the difference between the current time and its previous one is constant along the simulation time. Therefore, the time difference between the current time and the previous one is equal to dt as illustrated in Fig. 17(b). Therefore, apply (9) for vctrl;

$$\frac{d^2V_{ctrl}}{dt^2} = \frac{2V_{ctrt_{prev_2}}}{(dt)(2dt)} - \frac{2V_{ctrl_{prev_1}}}{(dt)(dt)} + \frac{2V_{ctrl}}{(dt)(2dt)} = \frac{1}{dt^2}\left(V_{ctrl_{prev_2}} - 2V_{ctrl_{prev_1}} + V_{ctrl}\right) \tag{10}$$

Substitute by (10) in (8) and notice that any first derivative electric discipline, I_{cp}&V_{ctrl}, will be a linear combination of two points;

$$\begin{aligned}&\frac{1}{dt}\left(I_{cp}-I_{cp_{prev_1}}\right)\\&=\frac{C_1}{dt^2}\left(V_{ctrl_{prev_2}}-2V_{ctrl_{prev_1}}+V_{ctrl}\right)\\&\quad+\frac{1}{R_2dt}\left(V_{ctrl}-V_{ctrl_{prev_1}}\right)\\&\quad-\frac{1}{R_2C_2}I_{cp}+\frac{C_1}{R_2C_2}\left(V_{ctrl}-V_{ctrl_{prev_1}}\right)\end{aligned}\tag{11}$$

From (11), the V_{ctrl} is equal

$$V_{ctrl}=\frac{a_2}{a_0}V_{ctrl_{prev_2}}+\frac{a_1}{a_0}V_{ctrl_{prev_1}}+\frac{b_1}{a_0}I_{cp_{prev}}+\frac{b_0}{a_0}I_{cp}\tag{12}$$

where,

$$a_0=\frac{C_1C_2R_2}{dt^2}+\frac{C_1+C_2}{dt},\quad a_1=\frac{2C_1C_2R_2}{dt^2}+\frac{C_1+C_2}{dt},$$
$$a_2=\frac{-C_1C_2R_2}{dt^2},\quad b_0=1+\frac{C_2R_2}{dt},\ b_1=-C_2R_2$$

B. PIECE-WISE LINEAR (PWL)

The PWL technique needs both the value and the slope information so that the values between the samples can be interpolated. To calculate the slope between the samples, the time should be taken into consideration at each event-driven of the signal value as the slope is equal to the value change divided by the time change. Moreover, the value of the signal is a function of time. Therefore, the PWL signal holds the signal, the slope, and the time. This is represented by the user-defined type and creates a net from this defined type, to hold more than one variable in the same net. The signal value should be represented dynamically, as a function of time, to model the change in the output signal when there is a change in the input signal at the current time. Since the LF components are resistors and capacitors that are connected in series and parallel. Moreover, the values of resistors and capacitors do not vary with time. Therefore, the LF system can be considered a Linear Time-Invariant (LTI) system. The linear equation of the analog LF system is reached by putting the model into a linear state-space form, a solution with the Laplace transform, or by finding the linearized model in the Laplace s-domain. To find a linearized relation between the output voltage and the input current of the LF system, the LF transfer function is linearized with a ramp input function.

Fig. 18. illustrates the ramp function in the time domain. Using Laplace transform, the ramp function in the s-domain is equal to $X(s)=\frac{A}{s}+\frac{B}{s^2}$, where B represents the slope value and A represents the constant value. Therefore, the voltage control signal function in the s-domain is a multiplication of the s-domain ramp function with the s-domain of the LF function.

```
package EE_pkg;
//User-defined datatype EEstruct (UDT)
    typedefstruct{
        real volt;
        real curr;
        real resist;
    } EE_struct;

nettype EE_struct EE_net; //User-Defined Net (UDN)
endpackage: EE_pkg

//MODULE (ports)
module LPF (CP_CURR, VCTRL);

//IMPORTING PACKAGES
import EE_pkg::*;

//TYPE OF PORTS (input/output/inout)
    input CP_CURR;
    output VCTRL;

//DATA-TYPE OF PORTS (logic, real, net-type, ..)
    EE_net CP_CURR;
    EE_net VCTRL;

//PARAMETERS
    //GENERAL PARAMETERS
    parameter real M_PI = 3.14159265358979323846264433;
    parameter real TU = 1E-9; //Time Unit

    //PARAMETERS of INPUTS/OPTPUTS (FREQ, FOUT, ICP, KN, KP, KQ, WP, KVCO, WP, QP)
    parameter real FREF = 100E6;  //REFERENCE FREQUENCY (Hz)
    parameter real FOUT = 5E9;   //OUTPUT FREQUENCY (Hz)
    parameter real N = FOUT/FREF; //DIVIDER_RATIO
    parameter real KN = 1/N;
    parameter real KP = 1;
    parameter real ICP = 50E-6;   //CHARGE PUMP CURRENT
    parameter real KQ = ICP/(2*M_PI);
    parameter real KVCO = 2*M_PI*1E9;
    parameter real WP = 2*M_PI*1E6;
    parameter real QP = 60*(M_PI/180);

//VARIABLES
 //CALCULATIONS OF SECOND ORDER FILTER PARAM
    real TauP = ((1/$cos(QP)) - $tan (QP))/WP;
    real TauZ = 1/($pow(WP,2)*TauP);
    real x = TauP/TauZ;
    real y = (KQ*KVCO)/($pow(WP,2)*N);
    real z = $pow ( ((1+$pow(WP*TauZ,2)) /(1+$pow(WP*TauP,2))) ,0.5);

    real C1 = x*y*z;            //Cap 1 value
    real C2 = C1 * ((TauZ/TauP)-1);//Cap 2 Value
    real R2 = TauZ/C2;; //R2 Value

    real TS = 1;        //Samplying time
    real dt = TS * TU;  //Constant Delta-Step


    real b0 = 1 + ((C2 * R2) / dt);
    real b1 = - ((C2 * R2) / dt);
    real a0 = ((C1 * C2 * R2) / $pow(dt,2)) + ((C1+C2) / dt);
    real a1 = ((C1+C2) / dt) + (2*(C1 * C2 * R2) / $pow(dt,2));
    real a2 = - ((C1 * C2 * R2) / $pow(dt,2));

    real cp_curr_prev;//pow(cp_curr,-1)
    real ctrl;
    real ctrl_prev1;   //pow(ctrl,-1)
    real ctrl_prev2;   //pow(ctrl,-2)

   real resis;

//PROCEDURAL BLOCKS
    always #(TS) begin
        ctrl = (b0 * CP_CURR.curr + b1 * cp_curr_prev + a1 * ctrl_prev1 + a2 * ctrl_prev2) / a0;
        cp_curr_prev = CP_CURR.curr;
        ctrl_prev2 = ctrl_prev1;
        ctrl_prev1 = ctrl;
    end

    always @(CP_CURR.curr) begin
        if(CP_CURR.curr == 0) resis  = resis;
        else                  resis  = ctrl/CP_CURR.curr;
    end

    assign VCTRL = EE_struct{ctrl, CP_CURR.curr, resis};

endmodule
```

Listing 1. 1. SV code represents PWC by defining User-Defined Type (UDT) 'EE_struct.

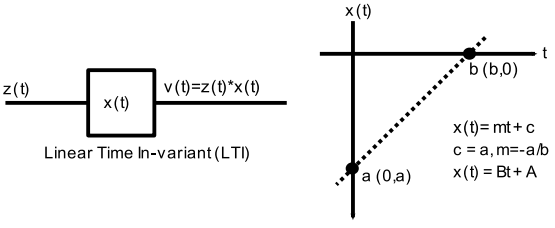


FIGURE 18. System responsible for linearization is ramp function.

$$\begin{aligned}V(s)&=X(s).Z(s)\\&=\left(\frac{A}{s}+\frac{B}{s^2}\right)\left(\frac{1+sC_2R_2}{s(C_1+C_2)+s^2C_1C_2R_2}\right)\end{aligned}\tag{13}$$

To find the time domain for the above equation, the Inverse Laplace transform should be applied. Since the main goal in any dynamic system is to find its response to a given input. Therefore, the inverse Laplace transform of $V(s)$ is a sum of two-component system response: a zero-state response which is caused by the external signals forcing, and a zero-input response which is caused by the system initial conditions or caused by the natural responses. The Inverse Laplace transform will produce both the zero-state and the zero-input components of the system response. In [44], there is information provided to find the zero-state and the zero-input solutions. The zero-state solution is the response of the system to the input with its initial conditions set to zero. Therefore, the zero-state solution is just found the inverse Laplace transform of (13).

$$v_{ZS}(t) = A\left(a_0 + a_1 t + d_0 e^{-wt}\right) + B\left(b_0 + b_1 t + c_1 t^2 + d_1 e^{-wt}\right) \quad (14)$$

The zero-input solution is the response of the system to the initial conditions and with the input set to 0, $Z(s) = \frac{V(s)}{X(s)} = 0$.

$$v_{ZI}(t) = \frac{1}{w} v'\left(0^-\right) + v\left(0^-\right) - \frac{1}{w} e^{-wt} v'\left(0^-\right) \quad (15)$$

where, $v\left(0^-\right)$ is the initial condition of $v(t)$ and $v'\left(0^-\right)$ is the first derivative initial condition of $v(t)$. Where,

$$a_0 = \frac{R_2 C_2^2}{(C_1 + C_2)^2}, \quad b_0 = \frac{-C_1 R_2^2 C_2^3}{(C_1 + C_2)^3}, \quad a_1 = \frac{1}{C_1 + C_2},$$
$$b_1 = \frac{R_2 C_2^2}{(C_1 + C_2)^2}, \quad c_1 = \frac{1}{2(C_1 + C_2)}, \quad d_0 = \frac{-R_2 C_2^2}{(C_1 + C_2)^2},$$
$$d_1 = \frac{C_1 R_2^2 C_2^3}{(C_1 + C_2)^3}, \quad w = \frac{C_1 + C_2}{C_1 C_2 R_2}$$

Therefore, the total response

$$v(t) = v_{ZS}(t) + v_{ZI}(t) \quad (16)$$

Eq. (16) could be simplified as the following

$$v(t) = a + bt + ct^2 + de^{-wt} \quad (17)$$

The first derivation of (17) should be found to calculate the first derivative initial condition, $v'\left(0^-\right)$,

$$v'(t) = b + 2ct - wde^{-wt} \quad (18)$$

where,

$$a = Aa_0 + Bb_0 + v\left(0^-\right) + \frac{1}{w} v'\left(0^-\right), b = Ab_1 + Bb_1,$$
$$c = Bc_1, d = Ad_0 + Bd_1 - \frac{1}{w} v'\left(0^-\right)$$

The parameters (a, b, c, and d) are varying with the simulation since they are depending on 'A' and 'B'. As illustrated before, the parameter 'A' represents the constant value of the input signal. The input signal to the LF is the charge pump current. Therefore 'A' is equal to the charge pump current value from the PWL net. While the parameter 'B' represents the slope of the input signal. Therefore, 'B' is equal to the charge current slope. The PWL method is completely dependent on the event-driven, therefore any change in the output from its value and slope should be executed when a new input comes. For that, the events of the input should be scheduled when occurred and the scheduling of the input events stored by the 'event' datatype. Then these events should be used as a trigger for updating the output signal. After triggering the input events, the current time should be stored when a wake-up input signal occurred. The output at the current time is then executed by evaluating the PWL function. Waiting for the next input event to update the output could take much time and maybe a lot of the output data is lost between two input events. The data lost in the output could be valuable, like timing jitter, according to the error tolerance that is given by the designer. If these data are exceeded the error tolerance. Therefore, they should not be wasted. For that the output should be updated after the delta time step where the delta time is preferred to dynamically be measured in a relation to the error tolerance as illustrated in Fig. 19. The delta time events are triggered to update the output after the delta time step.

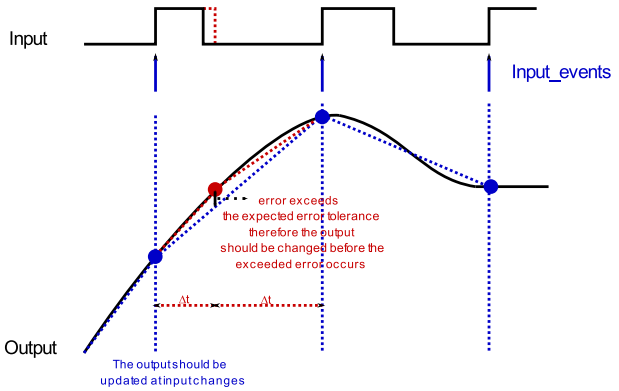


FIGURE 19. Error in output exceeded the expected error tolerance.

The error tolerance that is expected in the linear approximation for the time interval $(t0 \le t \le t0 + \Delta t)$, where $(t0)$ is the current time and $(t0 + \Delta t)$ is the next current time, is bounded by Taylor's error for the linear approximation [45].

$$T_2(t) \le \frac{1}{2}(t - a)^2 v''(c) \quad (19)$$

From the above equation, the maximum error tolerance expected when applying Rolle's theorem [47], If a function is continuous on a closed interval $[t0, t0 + \Delta t]$, differentiable on the open interval $(t0, t0 + \Delta t)$, and it takes the same value at both endpoints of the interval '$v(t0) = v(t0 + \Delta t)$' then somewhere on the open interval the function has a zero derivative, as illustrated in Fig. 20.

$$e_{tol} = \frac{1}{2}(t0 + \Delta t - t0)^2 \left|v''(t)\right| \quad (20)$$

The second derivative of $v(t)$ in (17) is a decaying exponential function plus a constant. Where the constant 'c' is equal to

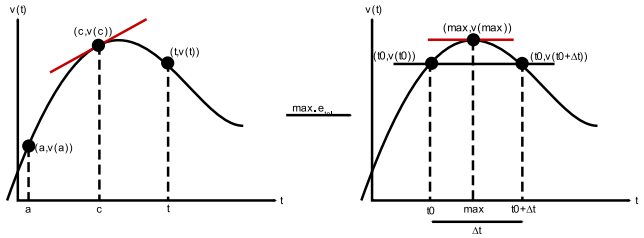


FIGURE 20. Applying Rolle's theorem on taylor's error for linear approximation.

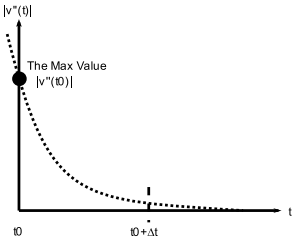


FIGURE 21. Absolute second derivative of v(t).

zero as it's a function of the charge pump slope which is equal to zero. Therefore, $v''(t)$ is considered as a single decaying exponential function and the absolute max. value of $v''(t)$ will be at t0 as illustrated in Fig. 21.

$$v''(t) = 2c + dw^2 e^{-wt} = 2(0) + dw^2 e^{-wt} = \alpha e^{-\beta t} \quad (21)$$

where,

$$\alpha = dw^2, \quad \alpha \in \mathbb{R} \; and \; \beta = w, \beta \in \mathbb{R}^+$$

Then the delta step,

$$\Delta t = \sqrt{\frac{2e_{tol}}{|v''(t)|}} = \sqrt{\frac{2e_{tol}}{|v''(t0)|}} \quad (22)$$

The next output, after the delta time step, could be calculated from (17) but at an input time equal to the current time plus the delta time step '$v(t0 + \Delta t)$'. The output slope could be calculated as $(v(t0 + \Delta t) - v(t0))/\Delta t$.

IV. EFFECTS

A. PHASE NOISE (TIMING JITTER)

For each block, the timing jitter could affect the time when the event could occur. The event could occur late or early due to the timing jitter. This timing jitter of each block can be calculated from the phase noise. The phase noise is in the frequency domain while the RMS jitter is in the time domain. The timing jitter is the integration of the phase noise power under the required frequency band. Since the simulation of the digital tools is event-driven therefore the integration of the phase noise can be modelled as an accumulation of the phase noise of the current frequency to the phase noise of the previous frequency until the required frequency band is covered. First, the noise contributed from each block should be found. Each block could contribute voltage noise, current noise, or phase noise according to the nature and the behavior of this block [37], [38], [39], [40], [41], [42]. Table 3.

```
package PWL_pkg;

 typedefstruct{
    real y;    //signal offset
    real slope; //signal slope
    real t0;   //time offset
 } PWL_struct;

nettype PWL_structPWL_net;
endpackage

//MODULE(ports)
module LPF(CP_CURR,VCTRL);

//IMPORTING PACKAGES
importPWL_pkg::*;

//TYPE OF PORTS(input/output/inout)
    input  CP_CURR;
    output VCTRL;

//DATA-TYPE OF PORT NETS(logic,real,net-type,...)
    PWL_netCP_CURR;
    PWL_netVCTRL;

//PARAMETERS
//GENERAL PARAMETERS
parameter real M_PI = 3.14159265358979323846264 33;

//Parameters of INPUTS/OPTPUTS(FREQ,FOUT,ICP,KN,KP,KQ,WP,KVCO,WP,QP)
parameter real FREF = 100E6;  //REFERENCE FREQUENCY(Hz)
parameter real FOUT = 5E9;    //OUTPUT FREQUENCY(Hz)
parameter real N = FOUT/FREF;//DIVIDER_RATIO
parameter real KN = 1/N;
parameter real KP = 1;
parameter real ICP = 50E-6;   //CHARGE PUMP CURRENT
parameter real KQ = ICP/(2*M_PI);
parameter real KVCO = 2*M_PI*1E9;
parameter real WP = 2*M_PI*1E6;
parameter real QP = 60*(M_PI/180);

//VARIABLES
//CALCULATIONS OF SECOND ORDER FILTER PARAM
real TauP = ((1/$cos(QP)) - $tan(QP))/WP;
real TauZ = 1/($pow(WP,2)*TauP);
real x = TauP/TauZ;
real y = (KQ*KVCO)/($pow(WP,2)*N);
real z = $pow(((1+$pow(WP*TauZ,2))/(1+$pow(WP*TauP,2))),0.5);

real C1 = x*y*z;
real C2 = C1*((TauZ/TauP)-1);
real R2 = TauZ/C2;

//SIGNALS/VARIABLES
eventwakeup;   //signal event(there's a change in input)
real t0;       //time offset
real t_curr;   //currenttime(now)
real t_curr_a;
real dTr;      //dT of PWL waveform in real value format
time dT;       //dT in time format

real etol = 1E-4; //user-defined error tolerance

real TU = 1E-12; //unittime

real out_curr;   //current(currenthere means now) outputsignal value
real out_curr_a;
real out_next;   //outat(t_curr+dT) for PWL outputdata
real out_slope;  //outslope
real out_slope_a;
real out0;       //initial value ofoutwhen a new inputcomes in


PWL_netVCTRL_dt; //differentiated outputsignal value
real out_curr_dt;   //currentdifferentiated outputsignal value
real out_curr_dt_a;
real out_next_dt;   //differentiated outat(t_curr+dT) for PWL differentiated data
real out_slope_dt;  //differtianated outslope
real out_slope_dt_a;
real out0_dt;       //initial value ofdifferentiated outwhen a new inputcomes in

real out_next_a;   //nextoutputsignal value (after currentvalue)
real out_next_dt_a;

real min;   //Variable for comparison

real a0 = (R2*$pow(C2,2))/$pow(C1+C2,2);
real b0 = -(C1*$pow(R2,2)*$pow(C2,3))/$pow(C1+C2,3);

real a1 = 1/(C1+C2);
real b1 = (R2*$pow(C2,2))/$pow(C1+C2,2);

real c1 = 1/(2*(C1+C2));

real d0 = -(R2*$pow(C2,2))/$pow(C1+C2,2);
real d1 = (C1*$pow(R2,2)*$pow(C2,3))/$pow(C1+C2,3);
```

Listing 2. Modified SV code [20] represents PWL for 2nd-order LF by defining user-defined type (UDT) 'PWL_struct'.

```
real w = (C1+C2) /(R2*C2*C1);
real a;  //a = A*a0 + B*b0 + (1/w)*out0_dt+out0
real b;  //b = A*a1 + B*b1
real c;  //c = B*c1
real d;  //d = A*d0*exp(-w*t) + b*d1*exp(-w*t) - (1/w)*(exp(-w*t))*out0_dt

real err;      //error in vcontrol
real err_dt;   //error in differentiated vcontrol

//FUNCTIONS
//Evaluate PWL segment
function real eval_pwl (input PWL_struct in, input real t);
 return in.y + in.slope * (t-in.t0);
endfunction

//vt(vctrl)
function real yt (input real t);
 return (a + b * t + c * $pow(t,2)+ d * $exp(-w*t));
endfunction

//vt_dt(vctrl_dt)
function real yt_dt (input real t);
 return (b + 2 * t* c + d * -w * $exp(-w*t));
endfunction

function real calculate_Tintv_spf (input real etol, input real t);
 real abs_f2max;
 abs_f2max = (d * $pow(w,2) * $exp(-w*t)) > 0 ? (d * $pow(w,2) * $exp (-w*t)) : - (d * $pow(w,2) * $exp(-w*t));
 if(abs_f2max == 0) begin abs_f2max = 1E12;end
 return sqrt((8 * etol)/(abs_f2max));
endfunction

//INITIALIZATION
initial begin
 dT = 1;
end

//PROCEDURAL BLOCKS
always @(CP_CURR.y or CP_CURR.slope) begin
 t0 = $realtime * TU;          //gettime
 out0 = out_next_a;
 out0_dt = out_next_dt_a;

 //Update Parameters
 a = CP_CURR.y * a0 + CP_CURR.slope * b0 + out0 + (1/w)*out0_dt;
 b = CP_CURR.y * a1 + CP_CURR.slope * b1;
 c = CP_CURR.slope * c1;
 d = CP_CURR.y * d0 + CP_CURR.slope * d1 - (1/w) * out0_dt;

 out_curr = eval_pwl (VCTRL,t0);      //currentoffsetvalue ( y = y + slope * (dt))
 out_curr_dt = eval_pwl (VCTRL_dt,t0); //currentoffsetvalue ( y_dt = y_dt + slope * (dt))

 dT = 0;
 ->> wakeup;
end

always @(wakeup) begin
 t_curr = $realtime * TU;
 dTr = calculate_Tintv_spf(etol, t_curr-t0);
 min = (dTr < 1) ? dTr : 1;
 dTr = (TU > min) ? TU : min;

 //current out / current differentiated out
 out_curr  = eval_pwl (VCTRL, t_curr);
 out_curr_dt = eval_pwl (VCTRL_dt, t_curr);

 //next out / next differentiated out
 out_next  = yt(t_curr - t0 + dTr);
 out_next_a = out_next;
 out_next_dt = yt_dt(t_curr - t0  + dTr);
 out_next_dt_a = out_next_dt;

 //slope
 out_slope  = (out_next - out_curr)/dTr;
 out_slope_dt = (out_next_dt - out_curr_dt)/dTr;

 //dT (t+dT)
 dT = time'(dTr/TU);

 //error in out / error in differentiated out
 err  = (dTr*(VCTRL.slope-out_slope)) > 0 ? (dTr*(VCTRL.slope-out_slope)) : - (dTr*(VCTRL.slope-out_slope));
 err_dt = (dTr*(VCTRL_dt.slope-out_slope_dt)) > 0 ? (dTr*(VCTRL_dt.slope-out_slope_dt)) : - (dTr*(VCTRL_dt.slope-out_slope_dt));

 if(err >= 0) begin
  out_curr_a = out_curr;
  out_slope_a = out_slope;
  t_curr_a = t_curr;
 end

 if(err_dt >= 0) begin
  out_curr_dt_a = out_curr_dt;
  out_slope_dt_a = out_slope_dt;
  t_curr_a = t_curr;
 end

 ->> #(dT) wakeup;
end

assign VCTRL  = PWL_struct'{out_curr_a, out_slope_a, t_curr_a};
assign VCTRL_dt = PWL_struct'{out_curr_dt_a, out_slope_dt_a, t_curr_a};
endmodule
```

Listing 2. ***(Continued.)*** **Modified SV code [20] represents PWL for 2nd-order LF by defining user-defined type (UDT) 'PWL_struct'.**

illustrates the noise contributed from each sub-block of the PLL system. Second, the noise of each block will be shaped until it reaches the PLL output. Therefore, the noise transfer function that will reshape the noise should be found. Table 4. illustrates the transfer function that will influence the noise contributed from each block at the output of the PLL. These functions are extracted from the linear model of the PLL by

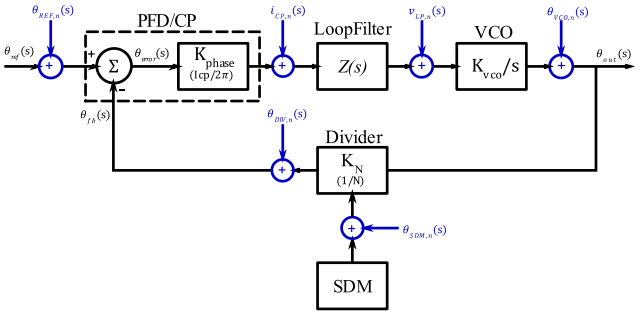


FIGURE 22. **PLl phase noise linear model.**

TABLE 3. **Noise contributed from each block of PLL system.**

Block	Phase/Current/Voltage Noise equation
Crystal Oscillator	$l_{ref}(f) = \frac{FKT}{2P_{sig}}\left(1+\frac{f_c}{f}+\left(\frac{f_o}{2Q_Lf}\right)^2+\frac{{f_o}^2 f_c}{4{Q_L}^2 f^3}\right)$
PFD-CP	${i_{CP,n}}^2(f) = \left(\frac{4KT\gamma(g_{mn}+g_{mp})}{} + \frac{2K_f}{f}\left(\frac{\mu_n}{{L_n}^2 C_{oxn}}+\frac{\mu_p}{{L_p}^2 C_{oxp}}\right)I_{CP}\right)T_{cp}F_{ref}$
LF	${v_{R,n}}^2 = 4KTR$
LC-VCO	$l(f) = \frac{FKT}{2P_{sig}}\left(1+\left(\frac{f_o}{2Q_Lf}\right)^2+\frac{{f_o}^2 f_c}{4{Q_L}^2 f^3}\right)$
RING-VCO	$l(f) = \frac{\lvert N_{out}\rvert^2}{{v_{osc}}^2} = \frac{16}{9{v_{osc}}^2}\frac{\gamma KT}{g_m}\left(\frac{f_{osc}}{f}\right)^2$
DIVIDER	$l(f) \approx \frac{10^{-14\pm1}+10^{-27\pm1}{f_o}^2}{f} + 10^{-16\pm1} + 10^{-22\pm1}f_o$
SDM	$L(f)_{SSB} = \frac{1}{2}\frac{(2\pi)^2}{12f_{ref}}\left(2\sin\left(\pi\frac{f}{f_{ref}}\right)\right)^{2(n-1)}$

TABLE 4. **Transfer function of each block.**

Block	The transfer function that will influence noise contributed from each block at the output of PLL
Crystal Oscillator (Reference Clock)	$H_{ref}(s) = \frac{\theta_{out,n}(s)}{\theta_{REF,n}(s)} = CL(s) = \frac{G(s)}{1+H(s)G(s)}$
PFD-CP	$H_{CP}(s) = \frac{\theta_{out,n}(s)}{i_{CP,n}(s)} = \frac{Z(s)}{1+OL(s)}\times\frac{K_{vco}}{s} = \frac{Z(s)}{1+H(s)G(s)}\times\frac{K_{vco}}{s}$
LF	$H_{LP}(s) = \frac{\theta_{out,n}}{v_{LF,n}} = \frac{1}{1+OL(s)}\times\frac{K_{vco}}{s} = \frac{1}{1+H(s)G(s)}\times\frac{K_{vco}}{s}$
VCO	$H_{VCO}(s) = \frac{\theta_{out,n}}{\theta_{VCO,n}} = \frac{1}{1+OL(s)} = \frac{1}{1+H(s)G(s)}$
DIVIDER	$H_{DIV}(s) = \frac{\theta_{out,n}}{\theta_{DIV,n}} = CL(s) = \frac{-G(s)}{1+H(s)G(s)}$
SDM	$H_{SDM}(s) = \frac{\theta_{out,n}}{\theta_{SDM,n}} = \frac{OL(s)}{1+OL(s)}$

injecting a noise source at the output of each sub-block of the PLL system as illustrated in Fig. 22.

Where,

Forward loop Gain "G(s)"	$G(s) = \frac{\theta_{vco}(s)}{\theta_{error}(s)} = \frac{K_{phase}\ Z(s)\ K_{vco}}{s}$
Reverse loop Gain "H(s)"	$H(s) = \frac{\theta_{fb}}{\theta_{vco}} = \frac{1}{N}$
Open Loop Gain "OL(s)=G(s).H(s)"	$OL(s) = \frac{\theta_{fb}(s)}{\theta_{error}(s)} = \frac{K_{phase}\ Z(s)\ K_{vco}}{N\ s}$
Closed Loop Gain "CL(s)"	$CL(s) = \frac{\theta_{vco}\ (s)}{\theta_{ref}(s)} = \frac{G(s)}{1 + H(s)G(s)}$

There are a lot of transfer functions that are linearized in the s-domain ($s = \sigma + j\omega$). The s-domain has a real part of the value (σ) and an imaginary part of the value (ω). In the digital environment, the s-domain or the frequency domain is not considered. Therefore, the designer could get the real and the imaginary parts through one variable and then create functions on this variable. A SystemVerilog package will be used to do some functions such as add, subtract, multiply, divide, absolute, and power on complex variables. The package has a 'class' named 'complex' that defines two real datatype class members which are the 'Real' and the 'Imag'. The user takes the values of 'Real' and 'Imag' from the user to create a class 'complex' variable and this is done by the class task 'set'. Then the class 'complex' variable becomes an input to one of the global functions such as addition, subtraction, multiplication, division, absolution, and power. For functions like addition, subtraction, multiplication, and division, the user needs to pass two class 'complex' variables to the function and the function will return one class 'complex' variable with the result. Moreover, the functions like absolution and power, the user needs to pass one class 'complex' variable to the function and the function will return one class 'complex' variable with the result. The user then could get these values by using the class function 'get'. The 'complex_pkg' is used to calculate the linear model transfer function of the loop filter and the VCO blocks. Moreover, it's used to calculate the linear model transfer function of the forward loop gain, the reverse loop gain, the open loop gain, and the closed loop gain through a range of frequencies. The most important that this package is built to calculate the <block> noise transfer function that will influence the noise from this <block> at the PLL output. Firstly, declaring a class 'complex' variable for each of the previous transfer functions. For each transfer function, two class 'complex' variables will be declared. One for the transfer function value at the given current frequency (f_i) and the other for the transfer function value at the next frequency ($f_i + increment$).

Now the need is to calculate all the transfer functions, and the noise contributed from each block. Therefore, a 'for loop' is used to calculate the value of these functions from the minimum to the maximum frequency. Then estimate the phase noise contributed from the block at the output of the PLL by multiplying the noise with the square power transfer function.

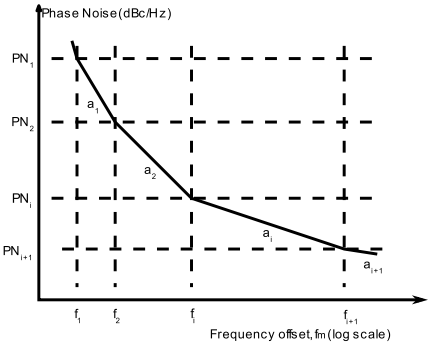


FIGURE 23. Linear piece-wise logarithmic function of PN spectrum.

$$PN = Noise * |TransferFunction|^2\ (dBc/Hz) \tag{23}$$

Finally, converting the phase noise to jitter. In [43], it is assumed that the phase noise PSD response can be approximated by a linear piecewise logarithmic function as illustrated in Fig. 23. The slope is expressed by

$$a_i = \frac{PN_{i+1} - PN_i}{\log_{10}(f_{i+1}) - \log_{10}(f_i)} (dBc/decade) \tag{24}$$

The RMS phase jitter can be calculated by the following

$$J_{block} = \frac{1}{2\pi f_c}\sqrt{2 * \sum J_i} \tag{25}$$

where,

$$J_i = \left(10^{\frac{PN_i}{10}} * f_i^{-\frac{a_i}{10}} * \left(1 + \frac{a_i}{10}\right)^{-1} * (1 + f_i)^{-1}\right) - f_i^{1+\frac{a_i}{10}} \tag{26}$$

All the phase noises contributed from each PLL sub-blocks are added to find the total contributed phase noise at the output of the PLL (dBc/Hz) by the following equation and this is converted to the total RMS timing jitter in (25).

$$PN_{total} = 10log10\left(\begin{array}{l} 10^{\frac{PN_{OUT_{REF}}}{10}} + 10^{\frac{PN_{OUT_{PFD-CP}}}{10}} \\ +10^{\frac{PN_{OUT_{LF}}}{10}} + 10^{\frac{PN_{OUT_{VCO}}}{10}} \\ +10^{\frac{PN_{OUT_{DIV}}}{10}} + 10^{\frac{PN_{OUT_{SDM}}}{10}} \end{array}\right) \tag{27}$$

After the jitter is calculated, the output event should be delayed by the jitter time. This means that the jitter can be translated as a delay in the output signal. The output signal occurs at a certain event 'e.g., always @(event) output = <>'. Therefore, the jitter cannot delay the output signal before when the output should occur as the output occurs due to an event that is driven. That means that the negative jitter cannot be modelled as the negative delay is not modelled in the SystemVerilog Language or any other HDL language that depends on the event-driven simulation ''#(-ve time) does not exist)''. Only the positive jitter can be modelled as illustrated in Fig. 24(a).

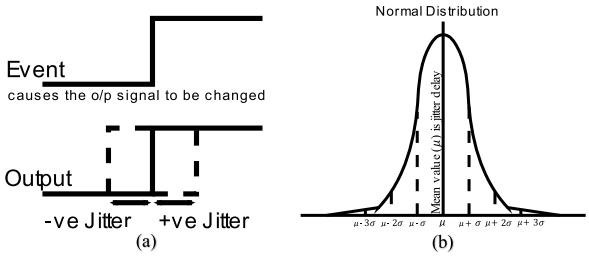


FIGURE 24. (a). Event-Driven models positive jitter only (b). Normal distribution.

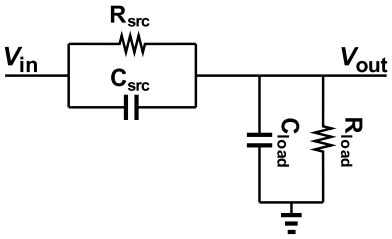


FIGURE 25. Impedance voltage division.

Since the designer is modelling the random jitter. Therefore, the value of the jitter calculated cannot be taken directly as a delay in the output signal. The jitter value that is calculated, does not consider the PVT (Process, Voltage, Temperature) variations. Therefore, the jitter value should be modelled as a Gaussian random distribution process since the Gaussian distribution can represent the PVT variations. At each event, the output signal will take a random delay jitter value according to the Gaussian (Normal) distribution as illustrated in Fig. 24(b).

B. LOADING

The output clock will be an input to another system. The system here acts as a load to the PLL output which means the high-power supply of the output clock will take time to reach its high value and the low power supply of the output clock also will take time to reach its low value. Therefore, the load effect on the output clock should be modelled to know the extent of the load that the output clock can hold to make the system behaves correctly else the load could change the output clock and the behaviour of the system. The load effect can be modelled by a User Defined Resolved Net (UDRN). The UDRNs are useful when the need is to resolve multiple drivers together, driving multiple signals through one net such as driving the load of the PLL and the load of the device that will be connected to the PLL. To take the loading effect, the net will be resolved by an impedance voltage division as illustrated in Fig. 25.

The User-Defined Resolution (UDR) function takes any UDT input connected to the User-Defined Resolved Net (UDRN) from the values of voltages, capacitances, and resistances. Then the net represents the resolved value after considering the effect of the impedance between the driven input voltage and the output that the user is waiting to see its resolved value. The output voltage and the current are

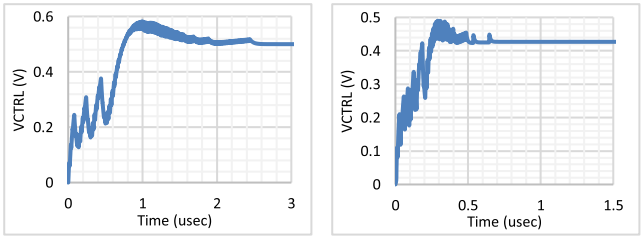


FIGURE 26. PLL locking of behavioral model for system_1 & _2.

resolved by the following equations (31 & 34).

$$I_{R_{scr}} + I_{C_{src}} = I_{R_{Load}} + I_{C_{Load}} \tag{28}$$

$$\frac{Vout - V_{in}}{R_{src}} + \frac{dV_{out} - dV_{in}}{dt} C_{src} = \frac{V_{out}}{R_{load}} + \frac{dV_{out}}{dt} C_{load} \tag{29}$$

The numerical first derivative of the data could be represented in a linear combination of two points ($\frac{dV}{dt} = \frac{V - V_{prev}}{t - t_{prev}}$) and assume that the delta step ($t - t_{prev}$) is constant through the simulation. Therefore, the delta step will be represented as dt.

$$\frac{V_{out} - V_{in}}{R_{src}} + \frac{\left(V_{out} - V_{out_{prev}}\right) - \left(V_{out} - V_{in_{prev}}\right)}{dt} C_{src} = \frac{V_{out}}{R_{load}} + \frac{\left(V_{out} - V_{out_{prev}}\right)}{dt} C_{load} \tag{30}$$

After some simplifications, the resolved output is equal to

$$V_{out} = \frac{1}{A + B + C + D} \begin{pmatrix} (B + D) * V_{in} \\ -(D) * V_{in_{prev}} \\ +(C + D) * V_{out_{prev}} \end{pmatrix} \tag{31}$$

For the Current flow in the output,

$$I_{out} = I_{R_{Load}} + I_{C_{Load}} = \frac{V_{out}}{R_{lood}} + \frac{dV_{out}}{dt} C_{load} \tag{32}$$

$$I_{out} = (A + C) * V_{out} - (C) * V_{out_{prev}} \tag{33}$$

After some simplifications and substitute by V_{out} in (30), the resolved current is equal to

$$I = \frac{A + C}{A + B + C + D} \begin{pmatrix} (B + D) * V_{in} \\ -(D) * V_{in_{prev}} \\ +(C + D) * V_{out_{prev}} \end{pmatrix} - C * V_{out_{prev}} \tag{34}$$

where,

$$A = \frac{1}{R_{Load}}, \quad B = \frac{1}{R_{src}}, \quad C = \frac{C_{Load}}{dt}, \quad D = \frac{C_{src}}{dt}$$

The net can model the effect of the loading at any node, this resolved net models a Low Pass (LP) filtering effect, a High Pass (HP) filtering effect, a resistance-voltage divider effect, a capacitance-voltage divider effect, a resistance-voltage effect, and a capacitance-voltage effect. Table 5. illustrates which capacitor/resistor needed to be set to reach the desired effect.

TABLE 5. Components of resolved net that need to be set.

Effect	R_{Load}	R_{src}	C_{Load}	C_{src}
High Pass (HP)	Value	$\approx \infty$	≈ 0	Value
Low Pass (LP)	$\approx \infty$	Value	Value	≈ 0
Capacitive voltage division	$\approx \infty$	$\approx \infty$	Value	Value
Resistive voltage division	Value	Value	≈ 0	≈ 0
Capacitive	$\approx \infty$	≈ 0	Value	≈ 0
Resistive	Value	≈ 0	≈ 0	≈ 0

```
package EE_pkg;
//User-Defined data-Type "EE_struct" (UDT)
typedefstruct{
real V;      //VOLTAGE
real V_prev;//PREVIOUS VOLT
real I;      //CURRENT
real R;      //RESISTANCE
real C;      //CAPACITANCE
 } EE_struct;

//User-Defined Net-type "EE_net" (UDN)
nettype EE_struct EE_net;

//User-Defined Resolution "EE_func_reslv" (UDR)
function automatic EE_struct EE_func_reslv (input EE_struct driver []);
  //variables declaration
real vin;       //VIN DRIVEN
real vin_prev; //PREVIOUS VIN DRIVEN
real vout_prev; //PREVIOUS VOUT DRIVEN
real src_res;   //SOURCE RESISTANCE (RESITANCE CONNECTED BETWEEN VIN AND VOUT)
real load_res, load_res_inv;
            //LOAD RESISTANCE (RESITANCE CONNECTED BETWEEN VOUT AND GND)
real src_cap;  //SOURCE CAPACITANCE (CAPACITANCE CONNECTED BETWEEN VIN AND VOUT)
real load_cap; //LOAD CAPACITANCE (CAPACITANCE CONNECTED BETWEEN VOUT AND GND)

parameter real TS = 1;
parameter real TU = 1E-9;

real dt = TS * TU; //DELTA-TIME

real A,B,C,D; //CONSTANTS

  //Initialization
EE_func_reslv.V = 0.0;
EE_func_reslv.V_prev = 0.0;
EE_func_reslv.I = 0.0;
EE_func_reslv.R = 0.0;
EE_func_reslv.C = 0.0;

  foreach (driver[i]) begin
   if(driver[i].V !== `wrealZState && driver[i].V_prev !== `wrealZState) begin
     vin_prev = driver[i].V_prev;
     vin = driver[i].V;
     src_res = driver[i].R;
     src_cap = driver[i].C;

     B = 1 / src_res;
     D = src_cap / dt;
   end

   else if(driver[i].V === `wrealZState && driver[i].V_prev === `wrealZState) begin
     load_res_inv += (1.0 / driver[i].R); //LOADED RESISTANCES DRIVEN
     load_res = 1.0 / load_res_inv;     //PARALLEL RESISTANCE
                          //to find out the equivalent loaded resistance
     load_cap += driver[i].C;

     A = 1 / load_res;
     C = load_cap / dt;
   end

   else if(driver[i].V_prev !== `wrealZState && driver[i].V === `wrealZState ) begin
     vout_prev = driver [i].V_prev;
   end
  end

EE_func_reslv.V = (1 / (A+B+C+D)) * ((B+D)*vin - (D)*vin_prev + (C+D)*vout_prev);
EE_func_reslv.V_prev = vout_prev;
EE_func_reslv.I = (A + C) * ((1 / (A+B+C+D)) * ((B+D)*vin - (D)*vin_prev + (C+D)*vout_prev)) - C * vout_prev;
EE_func_reslv.R = (1/A) / ((1/A) + (1/B));
EE_func_reslv.C = (D / (D+C));
endfunction

//User-Defined Resolved Net-type "EE_net_reslv" (UDRN)
nettype EE_struct EE_net_reslv with EE_func_reslv;

endpackage
```

Listing 3. SV code represent Load Effect modelling with UDR/UDRN.

V. PLL SYSTEM INTEGRATION AND EXPERIMENTAL RESULTS

The Behavioral Model uses the behavioral model of the PFD and the CP. The PWC technique for the LF and the integer divider. Fig. 26. illustrates the settling behavior of the VCO control voltage to be settled at 500 mV for System_1 and 420mV for System_2. The settling time is approximately 2.5 μsec for System_1 and 0.8 μsec for System_2.

The Structural Model uses the structural model of the PFD and the CP. The PWL technique for the LF and the fractional divider. Fig. 27. illustrates the PLL locking where, the VCO control voltage reached 99% of its final value at 400mV at 2 μsec for System_1 and at 0.6 μsec for System_2.

There are ripples on the control voltage at settling (locking) of the PLL as illustrated in Fig. 27. because of the instantaneous change in the division ratio, to get the average required output clock frequency of 4.8 GHz as illustrated in Fig. 28. This means that the phase noise contributed from each PLL sub-block that is added to find the total single-side-band phase noise of the PLL (1KHz∼100MHz) and the timing jitter converted from phase noise will be higher than the case of the integer division by the noise of SDM as illustrated in Fig. 29. and Table 6.

The previous table shows the typical values of the RMS jitter while Fig. 30. illustrates the values from the mean value of the RMS jitter at the PLL output clock frequency through the simulation time by using a Gaussian distribution of sigma equal to (0.001). Fig. 31. illustrates the time difference between the clock output frequency with and without jitter.

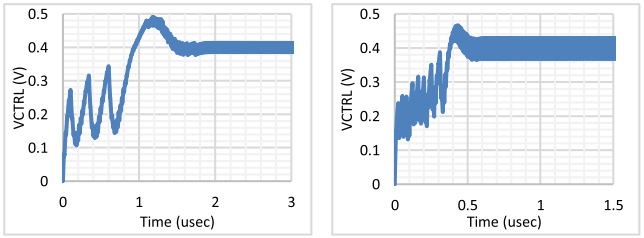


FIGURE 27. PLL locking of structural model for system_1 & _2.

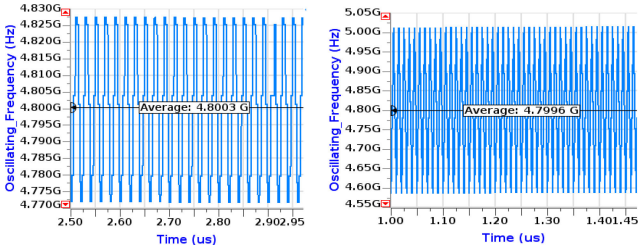


FIGURE 28. Output oscillating frequency through locking time for system_1 & _2.

TABLE 6. Sub-block jitter at PLL output.

Block	System_1	System_2
Reference clock	19.6199 fsec	19.5896 fsec
PFD + CP	339.42 fsec	404.934 fsec
Loop Filter	695.983 fsec	1.29828 psec
VCO	41.9248 fsec	333.408 fsec
DIVIDER	233.605 fsec	233.258 fsec
SDM	460.266 fsec	1.73453 psec
Total	931.935 fsec	2.2415 psec

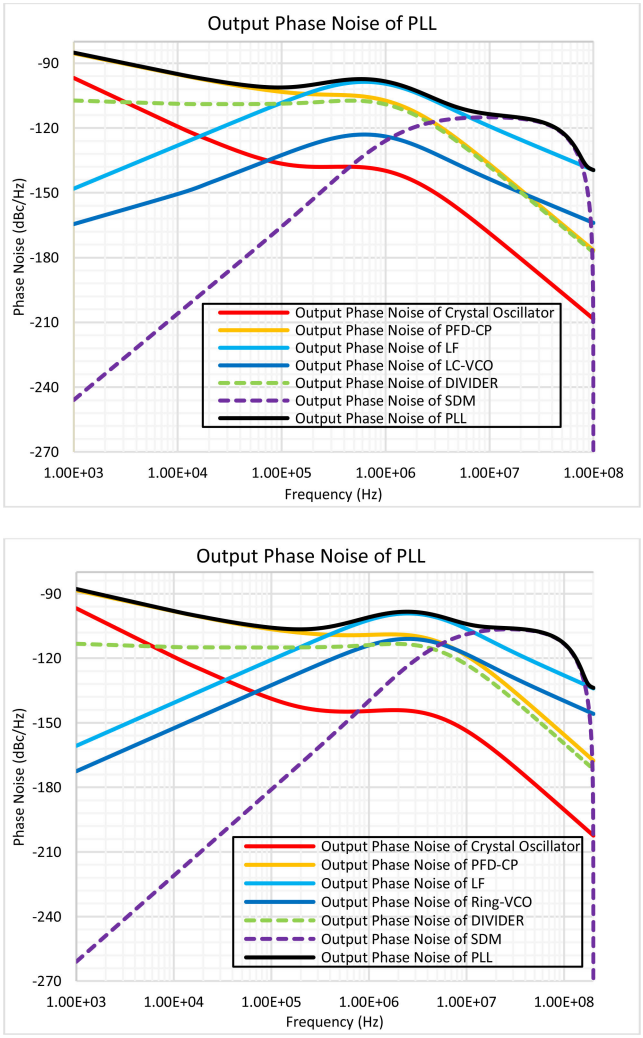


FIGURE 29. Output phase noise of 2nd-order pll for system_1 & _2.

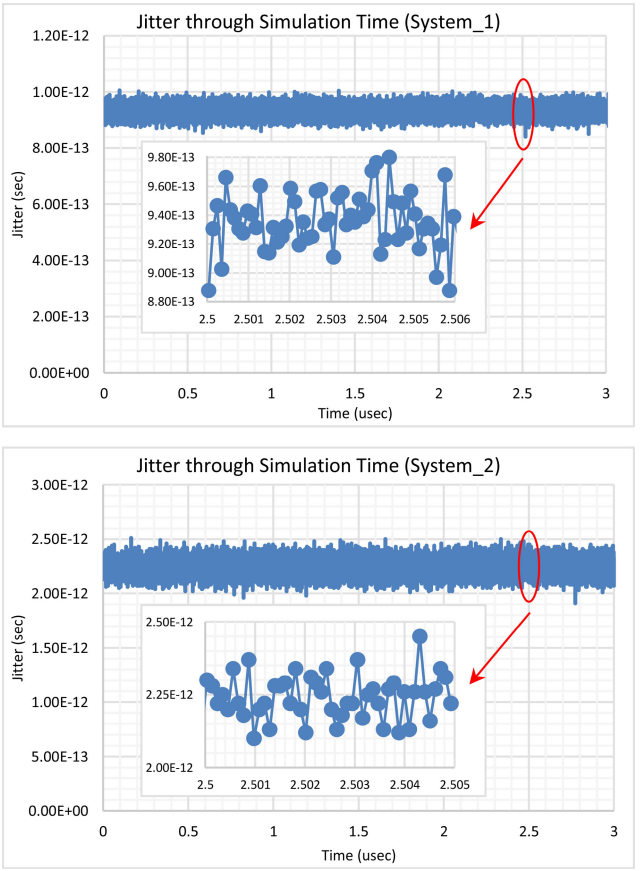


FIGURE 30. Total PLL RMS jitter through simulation time.

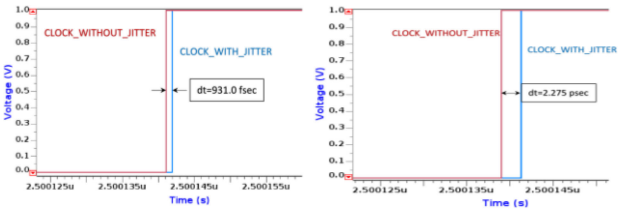


FIGURE 31. Time difference between output clk with and without jitter.

The jitter histogram in Fig. 32. illustrates the total RMS jitter value of 12000 clock cycles of both systems. System_1 contributes rms jitter of mean 928.345 fsec. and System_2 contributes rms jitter of mean 2.33 psec.

For both systems, the need for the carrier frequency or the output oscillating frequency to be equal to 5GHz. The first system contributes less noise power (less rms jitter) than System_2 but System_2 has a smaller area regarding the use of a RING-oscillator instead of an LC-oscillator and the Loop Filter (LF) components are smaller. The specification on each system is driven to serve the VCO block. For example, the loop bandwidth for System_2 is greater than System_1, to reject more noise coming from the VCO block since the RING-oscillator consumes more noise than the LC-oscillator. System_2 consumes less amount of current to decrease the resistance value in the loop filter which could then has a significant noise contribution after the loop bandwidth increased. The division ratio of System_1 is the double ratio of System_2 that is because of the need for a smaller LF area, if the division ratio of System_1 is chosen to be equal to the division ratio of System_2 this will lead to double the area of the LF despite the double division ratio will consume just an extra divider-by-two cell that could be implemented by two flipflops only.

Increasing the loop bandwidth decreases the settling time. Moreover, smaller loop bandwidth leads to reject more noise from the PFD/CP/LF noise but less neglection of noise from the VCO side. This means that there is a trade-off between decreasing the settling time of the PLL which may be a strict requirement in some applications that are applied at very high speed and decreasing the output phase noise. Therefore, the unity gain frequency (wu) which is the loop bandwidth should be adjusted to achieve the minimum output phase noise and the optimum settling time needed for the application.

For the loading effect, Fig. 33(a). illustrates the output clock frequency with different values of (R_{src}) from (10Ω to 500Ω). While the capacitance load (C_{Load}) is 5pF. The R_{src} of 500Ω is not acceptable for the high-speed circuits as it takes time to reach its high value, for that there is always a

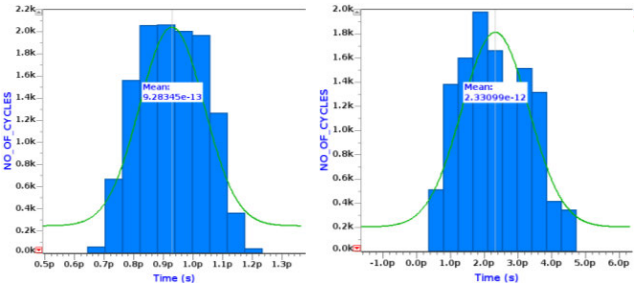


FIGURE 32. Total PLL RMS histogram jitter of 12000 clock cycles for system_1 & _2.

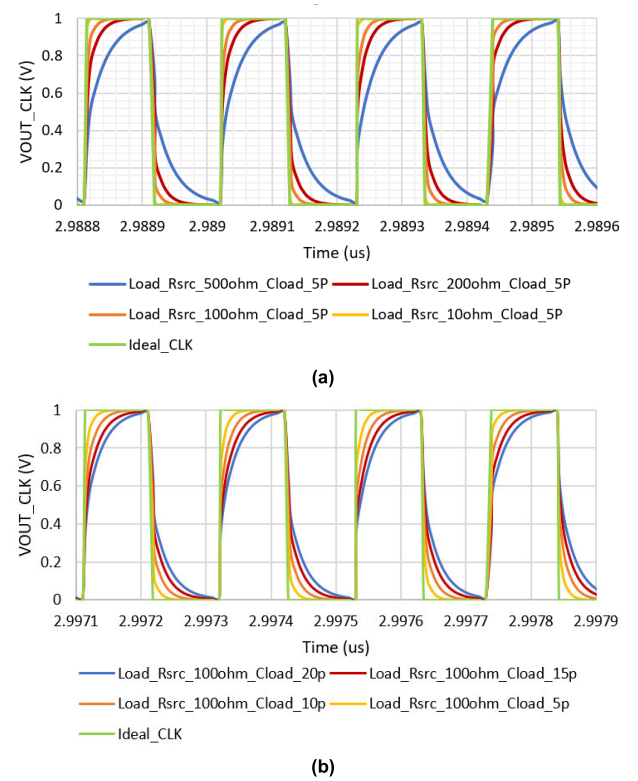


FIGURE 33. (a). Same C_{Load}/Different R_{src} (b). Different C_{Load}/Same R_{src}.

buffer between the PLL output and the device that will be connected to it as the buffer has a small source resistance. The digital design engineer can change this value or even model a filtering effect according to the specifications of the system. Fig. 33(b). illustrates the output clock frequency with (R_{src}) of 100Ω and the capacitance load (C_{Load}) from (5pF to 20pF).

VI. COMPARISON BETWEEN SV-RNM AND TRANSISTOR LEVEL

This partition ensures that the accuracy of modelling PLL with these techniques is increased by comparing the simulation results from the SV-RNM model simulated using an event-driven simulator with the transistor level outputs simulated by a spice simulator. Table 7. represents the control voltage comparison between the transistor level [48] and SV-RNM.

Second, comparing the output techniques of modelling the phase noise produced from [49], [50] with the output

TABLE 7. Control voltage comparison.

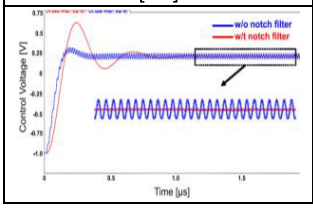
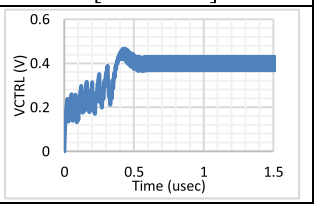
[48]	[This work]
	

TABLE 8. Phase noise comparison.

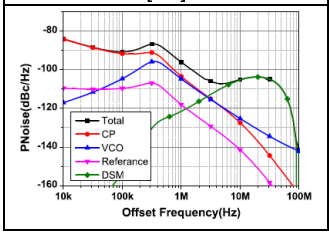
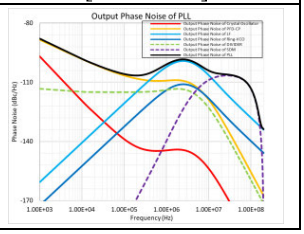
[50]	[This work]
	

TABLE 9. Comparison between simulation time for 5us transient time.

Parameter	Simulation Clock Wall time
SPICE [53]	1.23 h.
SPICE [26]	9.5 h.
FAST SPICE [26]	1.5 h.
Verilog-AMS [15]	16.67 min.
SV-RNM [This Work]	107 sec.

modelled from the SV-RNM. Table 8. proves that the method of evaluating the PN from the block parameters, by building a 'class' datatype that holds the real and the imaginary parts of the s-domain element, is very efficient and precious.

Most digital design engineers do not model the load effect at the PLL output for that the output oscillating frequency is always pure '1' and '0' whatever the HDL used. But the amplitude of the oscillating clock is changed according to the load connected to it and its effectiveness high/low supply values are also changed [51], [52]. Therefore, the UDRNs can model this effect by resolving multiple signals that are driven together in one net.

From the above, it's clear that SV-Real Number Models can achieve higher accuracy modelling for the high-level abstraction of the analog devices with a very high simulation time gain than Verilog-AMS/Verilog-A and SPICE as in Table 9. If the user wants to change a one-block parameter to verify the system, the simulation time will be in minutes for VERILOG-AMS and in hours for SPICE despite in seconds for a high accuracy model Device Under Test (DUT) [54], [55], [56].

VII. CONCLUSION

A lot of techniques could increase the accuracy of modelling analog devices in the digital environment. The PWL technique, a UDN that holds more than one member (value, slope, and time), is very useful when modelling filters in a feedback loop. The high-level abstraction of the fractional

divider captures ripples within the control voltage that could increase the noise contribution compared to the integer divider. Capturing the value of the PN from the block parameters, by building a 'class' datatype that holds the real and the imaginary parts of the s-domain element then using the block parameters and convert the PN to the RMS jitter that helps in predicting the RMS jitter of the PLL system. The UDRNs are helpful in resolving multiple signals that are driven together in one net, an example is the loading effect that helps in predicting the load that the PLL can hold without changing the output clock amplitude and according to the specifications of the PLL that are put to serve the other blocks that PLL will be a clock to them.

REFERENCES

[1] S. X.-D. Tan and C.-J. R. Shi, "Efficient approximation of symbolic expressions for analog behavioral modeling and analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 6, pp. 907–918, Jun. 2004.

[2] B. C. Lim and M. Horowitz, "An analog model template library: Simplifying chip-level, mixed-signal design verification," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 1, pp. 193–204, Jan. 2019.

[3] S. Balasubramanian and P. Hardee, *Solutions for Mixed-Signal SoC Verification Using Real Number Models*. Bengaluru, India: Cadence Design Systems WP, 2019.

[4] C. Sapsanis, M. Villemur, and A. G. Andreou, "Real number modeling of a SAR ADC behavior using SystemVerilog," in *Proc. 18th Int. Conf. Synth., Modeling, Anal. Simulation Methods Appl. Circuit Design (SMACD)*, Jun. 2022, pp. 1–4.

[5] N. Georgoulopoulos and A. Hatzopoulos, "Efficiency evaluation of a SystemVerilog-based real number model," in *Proc. 7th Int. Conf. Modern Circuits Syst. Technol. (MOCAST)*, May 2018, pp. 1–4.

[6] *IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language*, Standard 1800–2017, The Design Automation Standards Committee, IEEE Computer Society and the IEEE Standards Association Corporate Advisory Group, 2018.

[7] S. Sutherland, S. Davidmann, and P. Flake, *SystemVerilog For Design: A Guide to Using SystemVerilog for Hardware Design*. Cham, Switzerland: Springer, 2013.

[8] S. Little, M. O'Leary, and D. Miller, *Verilog-AMS Language Reference Manual*. Elk Grove, CA, USA: Accellera Systems Initiative, May 2009.

[9] K. S. Kundert and O. Zinke, *The Designer's Guide to Verilog-AMS*. Norwell, MA, USA: Kluwer, Jun. 2004.

[10] E. Christen and K. Bakalar, "VHDL-AMS–A hardware description language for analog and mixed-signal applications," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 46, no. 10, pp. 1263–1272, Oct. 1999.

[11] W. Zhang and N.-J. Wu, "A novel hybrid phase-locked-loop frequency synthesizer using single-electron devices and CMOS transistors," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 11, pp. 2516–2527, Nov. 2007.

[12] A. Korobkov, A. Agarwal, and S. Venkateswaran, "Efficient FinFET device model implementation for SPICE simulation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1696–1699, Oct. 2015.

[13] A. Doboli and R. Vemuri, "Behavioral modeling for high-level synthesis of analog and mixed-signal systems from VHDL-AMS," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 11, pp. 1504–1519, Nov. 2003.

[14] F. Pecheux, C. Lallement, and A. Vachoux, "VHDL-AMS and verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 2, pp. 204–225, Feb. 2005.

[15] A. Jakobsson, A. Serban, and S. Gong, "Implementation of quantized-state system models for a PLL loop filter using verilog-AMS," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 3, pp. 680–688, Mar. 2015.

[16] C. C. McAndrew, G. J. Coram, K. K. Gullapalli, J. R. Jones, L. W. Nagel, A. S. Roy, J. Roychowdhury, A. J. Scholten, G. D. J. Smit, X. Wang, and S. Yoshitomi, "Best practices for compact modeling in Verilog–A," *IEEE J. Electron Devices Soc.*, vol. 3, no. 5, pp. 383–396, Sep. 2015.

[17] I. Messaris, A. Serb, S. Stathopoulos, A. Khiat, S. Nikolaidis, and T. Prodromakis, "A data-driven Verilog–A ReRAM model," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 12, pp. 3151–3162, Dec. 2018.

[18] S. Liao and M. Horowitz, "A verilog piecewise-linear analog behavior model for mixed-signal validation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 8, pp. 2229–2235, Aug. 2014.

[19] N. Georgoulopoulos and A. Hatzopoulos, "Parameterizable real number models for mixed-signal designs using SystemVerilog," *J. Electron. Test.*, vol. 37, nos. 5–6, pp. 685–700, Dec. 2021.

[20] B. C. Lim and M. Horowitz, "Error control and limit cycle elimination in event-driven piecewise linear analog functional models," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 1, pp. 23–33, Jan. 2016.

[21] E. Ali, C. Hangmann, C. Hedayat, F. Haddad, W. Rahajandraibe, and U. Hilleringmann, "Event driven modeling and characterization of the second order voltage switched charge pump PLL," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 3, pp. 347–358, Mar. 2016.

[22] M. Louis, M. Dessouky, and A. Salem, "PLL real number modeling in SystemVerilog," in *Proc. 16th Int. Conf. Synth., Modeling, Anal. Simulation Methods Appl. Circuit Design (SMACD)*, Jul. 2019, pp. 257–260.

[23] W.-H. Chen, M. E. Inerowicz, and B. Jung, "Phase frequency detector with minimal blind zone for fast frequency acquisition," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 57, no. 12, pp. 936–940, Dec. 2010.

[24] C.-L. Ti, Y.-H. Liu, and T.-H. Lin, "A 2.4-GHz fractional-N PLL with a PFD/CP linearization and an improved CP circuit," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2008, pp. 1728–1731.

[25] W.-M. Lin, S.-I. Liu, C.-H. Kuo, C.-H. Li, Y.-J. Hsieh, and C.-T. Liu, "A phase-locked loop with self-calibrated charge pumps in 3-μm LTPS-TFT technology," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 56, no. 2, pp. 142–146, Feb. 2009.

[26] A. Lotfy, S. F. S. Farooq, Q. S. Wang, S. Yaldiz, P. Mosalikanti, and N. Kurd, "A system-verilog behavioral model for PLLs for pre-silicon validation and top-down design methodology," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Sep. 2015, pp. 1–4.

[27] D. M. Frías, M. T. S. Pascual, and C. A. D. L. C. Blas, "A CMOS automatic gain control design based on piece-wise linear exponential and logarithm cells," in *Proc. Eur. Conf. Circuit Theory Design (ECCTD)*, Sep. 2017, pp. 1–4.

[28] E. Jimenez-Dominguez, V. R. Gonzalez-Diaz, and A. M. Rodriguez-Dominguez, "Behavioral model of a VCO varying its Kvco with Verilog–A," in *Proc. 13th Int. Conf. Power Electron. (CIEP)*, Jun. 2016, pp. 70–74.

[29] N. Georgoulopoulos, A. Mekras, and A. Hatzopoulos, "Design of a SystemVerilog-based VCO real number model," in *Proc. 8th Int. Conf. Modern Circuits Syst. Technol. (MOCAST)*, May 2019, pp. 1–4.

[30] J. Jin, X. Liu, T. Mo, and J. Zhou, "Quantization noise suppression in fractional-*N* PLLs utilizing glitch-free phase switching multi-modulus frequency divider," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 5, pp. 926–937, May 2012.

[31] A. Elkholy, S. Saxena, R. K. Nandwana, A. Elshazly, and P. K. Hanumolu, "A 2.0–5.5 GHz wide bandwidth ring-based digital fractional-N PLL with extended range multi-modulus divider," *IEEE J. Solid-State Circuits*, vol. 51, no. 8, pp. 1771–1784, Aug. 2016.

[32] H. Wang, P. Brennan, and D. Jiang, "A generic multi-modulus divider architecture for fractional-N frequency synthesisers," in *Proc. IEEE Int. Freq. Control Symp. Joint With 21st Eur. Freq. Time Forum*, Jun. 2007, pp. 261–265.

[33] J. Rogers, C. Plett, and F. Dai, *Integrated Circuit Design for High-Speed Frequency Synthesis*, 2006, Ch. 6.2.4. A Multimodulus Divider.

[34] V. R. Gonzalez-Diaz, M. A. Garcia-Andrade, and G. E. Flores-Verdad, "Optimal dithered digital sigma-delta modulators for fractional-N frequency synthesizers," in *Proc. 18th Eur. Conf. Circuit Theory Design*, Aug. 2007, pp. 563–566.

[35] A. Telli and I. Kale, "Structured tone mitigation in 3^{rd} and 4^{th} order MASH delta-sigma modulators-comparative study," in *Proc. IEEE 10th Annu. Wireless Microw. Technol. Conf.*, Apr. 2009, pp. 1–5.

[36] W.-C. Lai, J.-F. Huang, C.-L. Wen, and W.-T. Lay, "FPGA implementation of a MASH 1–1–1 delta-sigma modulator infractional-N phase locked loop for fuzzy control application," in *Proc. 11th Int. Conf. Fuzzy Syst. Knowl. Discovery (FSKD)*, Aug. 2014, pp. 131–134.

[37] B. Razavi, "A study of phase noise in CMOS oscillators," *IEEE J. Solid-State Circuits*, vol. 31, no. 3, pp. 331–343, Mar. 1996.

[38] R. B. Staszewski, C. Fernando, and P. T. Balsara, "Event-driven simulation and modeling of phase noise of an RF oscillator," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 4, pp. 723–733, Apr. 2005.
[39] S. A. Osmany, F. Herzel, K. Schmalz, and W. Winkler, "Phase noise and jitter modeling for fractional-N PLLs," *Adv. Radio Sci.*, vol. 5, pp. 313–320, Jun. 2007.
[40] S. Limkumnerd and D. Eungdamrong, "Mathematical models and simulations of phase noise in phase-locked loops," *Songklanakarin J. Sci. Technol.*, vol. 29, no. 4, pp. 1017–1028, 2007.
[41] Z. Brezovic and V. Kudjak, "PLL phase-noise modeling by PC," in *Proc. 19th Int. Conf. Radioelektronika*, Apr. 2009, pp. 195–198.
[42] H. Wang, J. Zhou, and B. Li, "Analysis and implementation of Ku-band frequency synthesizer," in *Proc. 2nd IEEE Int. Conf. Comput. Commun. (ICCC)*, Oct. 2016, pp. 2628–2632.
[43] R. Pulikkoonattu, *Oscillator Phase Noise and Sampling Clock Jitter*. Geneva, Switzerland: ST Microelectronics WP, Jun. 2007.
[44] B. P. Lathi, *Signal Processing and Linear Systems*, 2nd ed., Jul. 2004, Ch. 2. Time-Domain Analysis of Continuous-Time Systems.
[45] J. Feldman, A. Rechnitzer, and E. Yeager, *CLP-1 Differential Calculus*, May 2021, Ch. 3.4.9. The Error in the Taylor Polynomial Approximations.
[46] Kintaar, "A numerical second derivative from three points," Math for Mere Mortals, Jan. 2013.
[47] D. Joyce, *The Mean Value Theorem Math 120 Calculus I D Joyce*. Worcester, MA, USA: Clark Univ., Fall 2013.
[48] X. Xu, Z. Wan, W. Rhee, and Z. Wang, "A bias-current-free fractional-N hybrid PLL for low-voltage clock generation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 9, pp. 3611–3620, Sep. 2021.
[49] P. Delos, *System-Level LO Phase Noise Model for Phased Arrays With Distributed Phase-Locked Loops*. Wilmington, MA, USA: Analog Devices, 2018.
[50] Y. Fu, L. Li, Y. Liao, X. Wang, Y. Shi, and D. Wang, "A 32-GHz nested-PLL-based FMCW modulator with 2.16-GHz bandwidth in a 65-nm CMOS process," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 7, pp. 1600–1609, Jul. 2020.
[51] J. Zhao, X. Wang, Y. Zhao, G. M. Xia, A. P. Qiu, Y. Su, and Y. P. Xu, "A 0.23-μg bias instability and 1-$\mu g/\sqrt{}$ Hz acceleration noise density silicon oscillating accelerometer with embedded Frequency-to-Digital converter in PLL," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 1053–1065, Apr. 2017.
[52] S. Kazeminia, K. Hadidi, and A. Khoei, "A 250 MHz to 4 GHz adaptive bias tuned PLL for low-jitter applications based on a fast response VCO oscillating cells," in *Proc. 22nd Iranian Conf. Electr. Eng. (ICEE)*, May 2014, pp. 139–144.
[53] L. Lian-Xi, Y. Yin-Tang, Z. Zhang-Ming, and L. Yani, "Design of PLL system based verilog-AMS behavior models," in *Proc. IEEE Int. Workshop VLSI Design Video Technol.*, May 2005, pp. 67–70.
[54] M. Leoncini, A. Bonfanti, S. Levantino, and A. L. Lacaita, "Efficient behavioral simulation of charge-pump phase-locked loops," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 6, pp. 1968–1980, Jun. 2018.
[55] C. Beyerstedt, J. Meier, F. Speicher, M. Scholl, D. Blase, R. Wunderlich, and S. Heinen, "A fast and accurate true event-driven phase locked loop model," in *Proc. 27th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nov. 2020, pp. 1–4.
[56] H. Li, M. E. Smith, Z. Xing, H. Liu, Y. Yu, Y. Wu, C. Zhao, and K. Kang, "An Event-Driven-Based behavioral modeling for Fractional-N CP-PLL," in *IEEE MTT-S Int. Microw. Symp. Dig.*, Nov. 2022, pp. 1–3.

MARIAM MAURICE received the B.Sc. (Hons.) and M.Sc. degrees in electronics engineering from Ain Shams University (ASU), Cairo, Egypt, in 2018 and 2023, respectively. She is currently a Product Engineer with Questa Simulator—Real Number Modeling and Questa Visualizer Debug Environment, Siemens Digital Industries Software (DISW)—Siemens Electronic Design Automation (EDA), Cairo. From 2018 to 2020, she was a Teaching Assistant (TA) with the Electronics and Communications Engineering (ECE) Department, ASU, and Misr International University (MIU). Her research interests include analog/mixed-signal (AMS) integrated circuits and systems, system-level design, modeling AMS behaviors using hardware description languages, and functional verification.

MOHAMED DESSOUKY received the B.Sc. and M.Sc. degrees in electrical engineering from Ain Shams University, Cairo, Egypt, in 1992 and 1995, respectively, and the Ph.D. degree in electrical engineering from the University of Paris VI, Paris, France, in 2001. In 1992, he joined the Electronics and Electrical Communications Engineering Department, Ain Shams University, where he is currently a Professor and the Director of the Integrated Circuits Laboratory (ICL). He was a Visiting Professor with the University of Paris VI, from 2002 to 2004. From 2004 to 2008, he was on leave from Mentor Graphics, Egypt, where he has been leading the Mixed-Signal Design Team responsible for the design of high-speed serial links. In 2021, he joined Pearl Semiconductor as a VP of engineering. He has coauthored a book chapter of *Analog Layout Synthesis: A Survey of Topological Approaches* (Springer, 2010), in addition to more than 80 papers in refereed journals and conferences. He holds three U.S. patents.

ASHRAF SALEM received the B.Sc. (Hons.) and M.Sc. degrees in computer engineering from Ain Shams University, Cairo, Egypt, in 1983 and 1987, respectively, and the Ph.D. degree in computer engineering from the University of Joseph Fourier, Grenoble, France, in 1992. He is currently a Professor with the Department of Computer and Systems Engineering, Ain Shams University, and the CEO of the CyTwin Laboratory. From 1996 to 2022, he was the Senior Engineering Director of Siemens Digital Industries Software, Cairo.