Ph.D. Dissertation

Event-Driven Simulation Methodology for Analog/Mixed-Signal Systems

혼성 신호 시스템을 위한

사건 구동 방식 모의 실험 방법

by

Ji Eun Jang

August 2015

School of Electrical Engineering and Computer Science
College of Engineering
Seoul National University

Event-Driven Simulation Methodology for Analog/Mixed-Signal Systems

지도 교수 김 재 하

이 논문을 공학박사 학위논문으로 제출함

2015 년 8 월

서울대학교 대학원

전기·컴퓨터공학부

장 지 은

장지은의 박사 학위논문을 인준함

2015 년 8 월

위 원 장 _____ (인)

부위원장 _____ (인)

위 원 _____ (인)

위 원 _____ (인)

위 원 _____ (인)

Abstract

Recent system-on-chip's (SoCs) are composed of tightly coupled analog and digital components. The resulting mixed-signal systems call for efficient system-level behavioral simulators for fast and systematic verifications. As the system-level verifications rely heavily on digital verification tools, it is desirable to build the mixed-signal simulator based on a digital simulator. However, the existing solutions in digital simulators suffer from a trade-off between simulation speed and accuracy. This work breaks down the trade-off and realizes a fast and accurate analog/mixed-signal behavior simulation in a digital simulator SystemVerilog.

The main difference of the proposed methodology from existing ones is its way of representing continuous-time signals. Specifically, a clock signal expresses accurate timing information by carrying an additional real-value time offset, and an analog signal represents its continuous-time waveform in a functional form by employing a set of coefficients. With these signal representations, the proposed method accurately simulates mixed-signal behaviors independently of a simulator's time-step and achieves a purely event-driven simulation without involving any numerical iteration.

The speed and accuracy of the proposed methodology are examined for various types of analog/mixed-signal systems. First, timing-sensitive circuits (a phase-locked loops and a clock and data recovery loop) and linear analog circuits (a channel and linear equalizers) are simulated in a high-speed I/O interface example. Second, a switched-linear-behavior simulation is demonstrated through switching

power supplies, such as a boost converter and a switched-capacitor converter. Additionally, the proposed method is applied to weakly nonlinear behaviors modeled with a Volterra series for an RF power amplifier and a high-speed I/O linear equalizer. Furthermore, the nonlinear behavior simulation is extended to three different types of injection-locked oscillators exhibiting time-varying nonlinear behaviors. The experimental results show that the proposed simulation methodology achieved tens to hundreds of speed-ups while maintaining the same accuracy as commercial analog simulators.

Keywords : Event-driven simulation, Behavioral modeling, Mixed-signal system, SystemVerilog, High-speed I/O interface, Switching-mode power supply, Volterra series model, Injection-locked oscillator

Student Number : 2011-30974

Contents

List of Figures

List of Tables

Chapter 1

Introduction

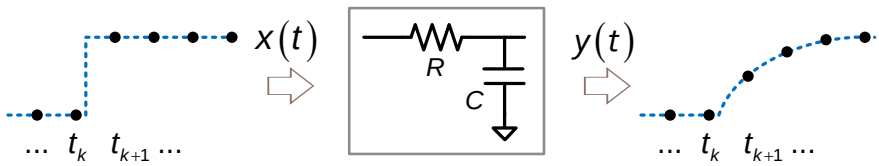
1.1 Background

Today's high-performance system-on-chip’s (SoCs) are complex, consisting of tightly coupled analog and digital components. For example, an RF front-end can include a digital calibration loop that compensates for mismatches between I/Q channels [1]. A high-speed wireline receiver may employ various digital adaptation loops to adjust the timing, voltage offsets, and equalization coefficients [2]. These systems contain multiple feedback loops with complex interactions between analog and digital blocks. Such a mixture of analog and digital presents challenges in verifying them, as it is very time-consuming to verify the entire system using a transistor-level SPICE simulation. A practical solution to achieve reasonable simulation speed is to employ behavioral models.

However, existing analog behavioral simulators such as Verilog-A and Matlab

Simulink are associated with a trade-off between speed and accuracy. The main cause is that they employ ordinary differential equation (ODE) solvers to simulate analog responses. ODE solvers numerically solve differential equations based on a time-integration method (e.g., forward-Euler, backward-Euler, or trapezoidal integration methods), which approximates the derivative of a continuous-time signal with a finite difference between two data points as shown in Fig. 1.1 [3]-[6]. This finite difference approximation is inaccurate once the two data points are positioned too far away. Therefore, the time steps between data points should be fine enough to achieve the accuracy at the cost of the simulation speed.

On the other hand, there have been many efforts to model analog behaviors entirely in digital simulators such as Verilog or VHDL [7]-[10]. However, the analog simulation still poses a speed bottleneck of the system simulation. These so-called real-number modeling (RNM) approaches model analog signals as discrete-time data using *real* data types in digital simulators. The main advantages of the RNM is that it inherits the digital simulator's natural features of fast speed and



$$RC\dot{y}(t)+y(t)=x(t)$$

$$\dot{y}(t_k)\approx\frac{y(t_{k+1})-y(t_k)}{t_{k+1}-t_k}$$

Finite difference approximation

Fig. 1.1 A simple example of a forward-Euler time-integration method for an RC filter.

event-driven simulation. However, the event-driven simulation of analog behaviors cannot be as efficient as digital behaviors due to digital simulator's quantized time steps. Fig. 1.2 illustrates this using a high-speed receiver example. A clock-triggered comparator in the receiver detects 0 or 1 by sampling the analog signal *Vin*(*t*) at the rising edge of the clock *clk*. As *Vin*(*t*) continuously changes, the sampled signal *Vout*(*t*) is highly dependent on the clock timing. However, when both signals are quantized by a fixed time step of the simulator, the accuracy can be lost. For instance, the fixed time step forces the analog signal to be expressed in a piecewise-constant waveform and limits the resolution of the clock jitter that can be expressed. Some techniques can be applied to improve the accuracy [11]-[12]. For instance, [11] uses interpolated values between discrete time points to obtain approximated continuous-time signals. [12] models clock signals as samples having an amplitude proportional to the pulse width in that fixed time interval. The proposed method takes a similar approach with [12] to express accurate clock timing information, yet in a more explicit way.

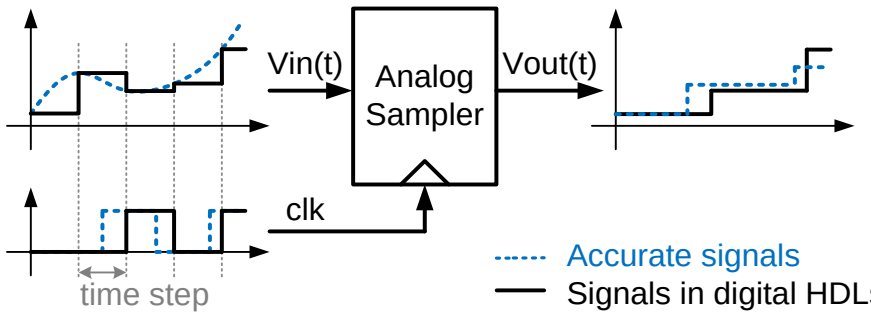


Fig. 1.2 An example of an analog sampler in a high speed receiver. The sampler is triggered by a noisy clock to sample a continuous-time input signal.

Another issue for the analog simulation in the digital simulator is that it is not truly event-driven in the sense that a single event at the input triggers multiple events at the output. Consider a low-pass RC filter that receives an one-time step change at its input (Fig. 1.3). To simulate its exponential response, the output events are required to be triggered multiple times even though there was only one input event. To relieve this inefficiency, some research employed a non-uniform time step [13]-[14]. [13] approximated an analog signal as a piecewise linear (PWL) waveform and updates its slope and offset values only when the approximation error exceeds a certain tolerance. [14] introduced connecting modules that suppress analog events when their changes are small enough. Nonetheless, they still could not realize a truly event-driven simulation.

Some recent research mitigated the speed-accuracy trade-off and realized a truly event-driven simulation of analog behaviors [15]-[17]. [15] built an event-driven model of a digitally-controlled oscillator in VHDL. For digital control codes and oscillator's phase noise characteristics, it computes perturbations of the fundamental oscillator's period. Then, the oscillator's digital output event is scheduled according

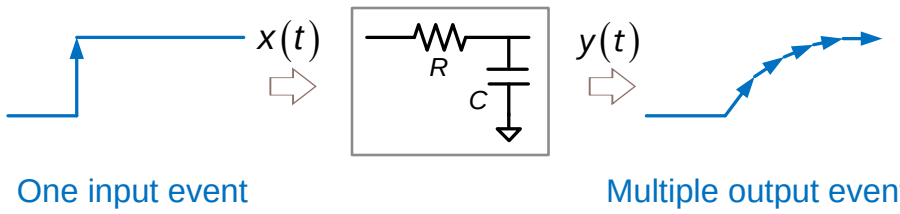


Fig. 1.3 Existing analog simulation methods in digital simulators are not truly event-driven as one input event may trigger multiple output events.

to the period. [16] presented an event-driven model for a channel ISI and a limited sampler bandwidth in Matlab. Fixing the input signal types to a pulse, the step response of a channel followed by a sampler is predetermined in a look-up table. For a series of input pulse events, the output is reconstructed as a sum of responses to each pulses. [17] implemented an event-driven simulator for a 3rd-order charge-pump PLL in C. It solved a set of governing differential equations for a loop filter and a voltage-controlled oscillator to obtain an exact phase expression in a functional form. At every clock event, it updates the phase expression and schedule the next clock switching event. While all these approaches achieved a truly event-driven simulation of analog behaviors, they share the limitations that they are not generally applicable to arbitrary systems or input types.

1.2 Main Contribution

This work proposes an event-driven simulation methodology that accurately simulates analog/mixed-signal behaviors without sacrificing the simulation speed. The key idea is the manner of expressing the clock and analog signals; rather than implicitly relying on fine time steps, it explicitly attaches the key information of interest to the signals. For the clock signal, a *real*-valued variable indicating exactly when the clock made the last transition is attached. For continuous-time analog signals, coefficients describing the signals as time-domain functions (e.g., c_i, m_i, and a_i of the complex exponential function $c_i t^{m_i} e^{-a_i t}$) are supplemented. With these signal representations, the clock and analog signals can be evaluated at any arbitrary time precision regardless of the simulator's time step. In addition, the analog behaviors can be simulated in a truly event-driven fashion without involving numerical time-integration methods.

The proposed methodology is implemented in a single digital hardware description language SystemVerilog. As most of recent SoCs include a large number of digital components, it is desirable to build a mixed-signal simulator based on a digital simulator. Also, the digital simulator inherently provides an event-driven engine supporting the proposed event-driven simulation. The supplemental variables for clock and analog signals are bundled as one signal using a composite data type *struct* in SystemVerilog, which keeps module ports pin-compatible.

The thesis demonstrates that the proposed method can simulate complex mixed-signal systems employing various classes of analog circuit behaviors: time-invariant/time-varying linear behaviors and time-invariant/time-varying nonlinear

behaviors. The first demonstrative example is a high-speed I/O interface with timing sensitive circuits (a phase-locked loop and a clock-and-data recovery loop) and linear analog blocks (a channel and linear equalizers). The second examples are switching power supplies which exhibit time-varying linear behaviors. In addition, weakly nonlinear behaviors are simulated in an RF transmitter and a multi-level pulse amplitude modulation receiver. Finally, the proposed method is extended to time-varying nonlinear behaviors like an injection-locked oscillator. Simulation results show that the proposed method can accurately simulate various mixed-signal systems in a truly event-driven fashion with tens to hundreds of speed-ups compared to existing analog simulators.

1.3 Thesis Organization

Chapter 2 describes the proposed representations of clock and analog signals, and their corresponding type definitions in SystemVerilog. Using those data types, the event-driven simulation of analog/mixed-signal behaviors is presented. Chapter 3 explains modeling details of linear systems on the proposed simulation platform using a demonstrative example of a high-speed I/O interface. Chapter 4 provides modeling examples of linear time-varying systems with various power converters. Chapter 5 applies the proposed method to nonlinear systems modeled with a Volterra series. Chapter 6 further demonstrates nonlinear time-varying system simulations with injection-locked oscillator examples.

Chapter 2

Event-Driven Simulation of Analog/Mixed-Signal Behaviors

This chapter explains main ideas of the proposed event-driven simulation methodology for analog/mixed-signal behaviors. The proposed method introduces new ways to express accurate clock and analog signals in digital hardware description languages (HDLs). To break the speed-accuracy trade-off explained in the introduction, new signal expression explicitly carries the supplementary information: transition timing information to the clock signal and a functional representation to the analog signal. The analog signal representation allows event-driven simulation of analog behaviors modeled in ordinary differential equations. The newly introduced signal types are implemented in a digital HDL, particularly SystemVerilog. These main ideas are demonstrated through a simple RC filter example.

2.1 Proposed Clock and Analog Signal Representations

As described in Chapter 1.1, digital simulators such as Verilog and VHDL use quantized time steps, which limit the accuracy and speed when modeling and simulating timing-critical clock signals and continuous-time analog signals. The proposed method overcomes this limitation by introducing two new signal types: *xbit* for clock signals and *xreal* for analog signals.

An *xbit*-type signal expresses accurate clock transition times by attaching its timing offset explicitly to the signal, as illustrated in Fig. 2.1. In digital simulators, the simulation time is quantized with a fixed time step, and the actual clock transition is snapped to the nearest integer multiple of the time step. The only way to make this apparent transition close to the actual transition is to decrease the time step. In contrast, the timing offset information attached to the *xbit* signal indicates where the actual transition is located relative to the apparent transition time. For instance, if the clock signal switches from 0 to 1 at an apparent time of 10 with the timing offset having a value of -0.4, this implies that the actual transition occurred at 0.4 before 10 (i.e., at a time of 9.6). This timing offset ranges [-1 x time step, 0] and takes a double-precision floating-point value. Therefore, the *xbit*-type signal can achieve a virtually infinite timing resolution regardless of the simulator's time step.

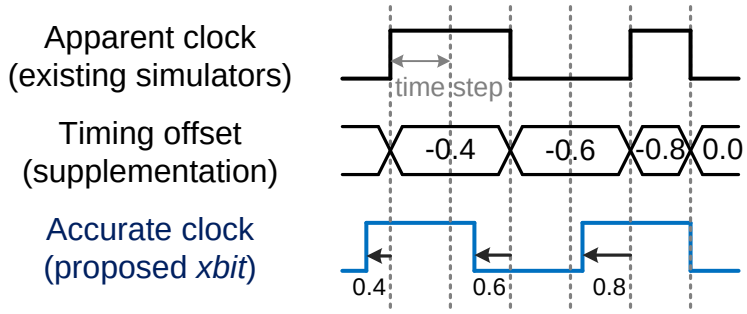


Fig. 2.1 The proposed *xbit*-type expressing an accurate clock waveform by supplementing the timing offset.

On the other hand, an *xreal*-type signal carries a set of coefficients expressing a continuous-time analog waveform in a functional form. Specifically, the analog signal $x(t)$ is represented as a linear combination of complex exponential functions:

$$x(t) = \sum_i c_i t^{m_i} e^{-a_i t} \ . \qquad \textbf{(2-1)}$$

This functional form is parameterized with three real-valued coefficients, c_i, m_i, and a_i. Any change in this set of coefficients constitutes an *xreal* event, and the analog waveform is represented as a series of these events. For example, Fig. 2.2 compares cases in which an exponential and a sinusoidal waveforms are expressed using a set of discrete data points and a series of *xreal* events. The existing digital simulators describe the waveform with the discrete data points (e.g., a *real*-type variable), and it requires fine time steps to improve its accuracy. In contrast, the *xreal* representation expresses the accurate waveform with only two events at t_0 and t_1, and its accuracy is independent of the simulation time step.

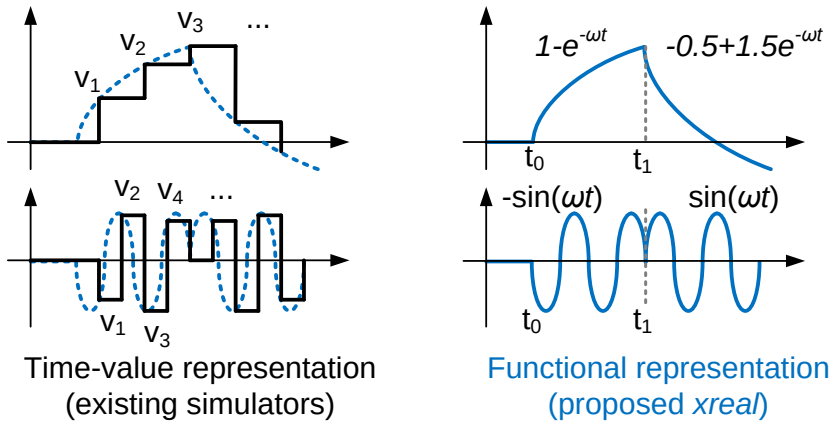


Fig. 2.2 The *xreal*-type expressing the continuous-time analog signal as a sum of complex exponential functions

In addition, the family of the *xreal* functional form in Eq. (2-1) includes most of signal types a designer may encounter in analog circuits. For instance, the *xreal* functional form expresses a family of polynomials with a_i =0, exponentials with a real a_i value, sinusoids with complex c_i and a_i values, and any linear combination of these signal families (Fig. 2.3).

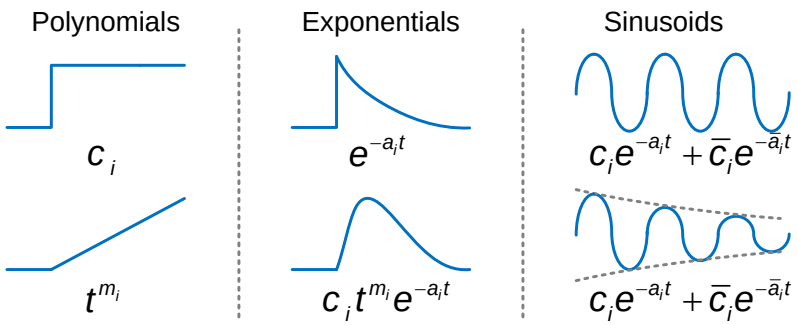


Fig. 2.3 Signal examples of the *xreal* representation.

Another advantage of the *xreal* functional form is that it can express outputs of both linear and nonlinear systems. Scrutinizing Eq. (2-1), the *xreal* functional form is a general expression of a linear system response, where a_i and m_i correspond to the eigenvalues (poles) and their multiplicities of the system, respectively. Therefore, this form encompasses all possible outputs of linear systems with *xreal* inputs. For nonlinear systems, the nonlinear distortions are often modeled as modulations between signals (e.g. Talyor or Volterra seires models [20]-[22]). As the *xreal*-type is in a sum-of-exponential form, multiplication between those exponentials results in the sum-of-exponential form, which is again the *xreal*-type.

2.2 Signal Type Definitions in SystemVerilog

The two proposed signal types *xbit* and *xreal* are defined as the *struct*-type in SystemVerilog. SystemVerilog is a language extension to Verilog [23], which supports a composite data type *struct* bundling multiple variables into one and allows the *struct*-type variable to pass through port boundaries. Therefore, the *struct*-type can attach the supplementary information (i.e., the timing offset in *xbit* and the coefficients in *xreal*) to the signal variables without adding new ports at the module interface.

Fig. 2.4 shows signal definitions of *xbit* and *xreal* in SystemVerilog. The *xbit* type has two member variables: *value* and *t_offset* where *value* denotes the logic level of the signal, and *t_offset* denotes the timing offset of the last transition. On the other hand, the *xreal* type has three member variables: *param_set*, *t_offset*, and *flag*. *param_set* is a C-pointer to the parameter set, which is an array containing the values of three coefficients: c_i, m_i, and a_i. For *param_set*, a dynamic data structure such as linked lists is necessary as the number of elements in the s-domain parameter set can vary from signal to signal. *t_offset* is the timing offset of the last change event. *flag* is an event variable to indicate whether the change event has occurred. Since the member variable *param_set* is merely a C-pointer whose address does not change once it is initialized, a separate variable is necessary to notify the blocks consuming this signal of the change in coefficients. Therefore, a block that

produces an *xreal* signal must trigger the event variable *flag* whenever it updates the parameter set (e.g. '->' operator in Verilog).

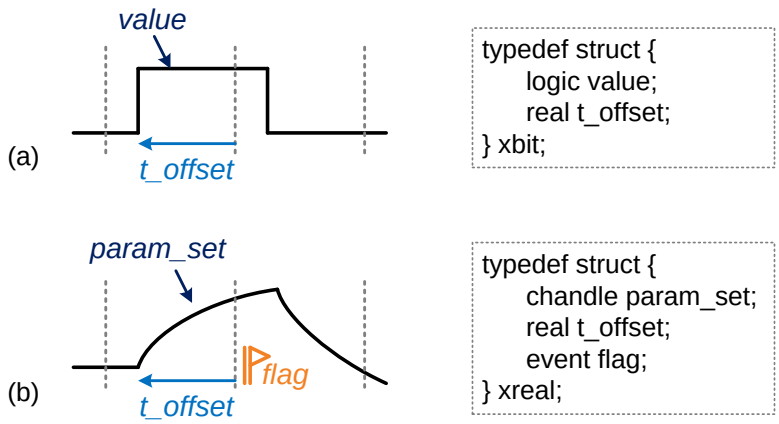


Fig. 2.4 The *struct* data type definitions of (a) *xbit* and (b) *xreal* in SystemVerilog.

2.3 Event-Driven Simulation Methodology

This section explains how the *xreal* representation of continuous-time signals enables a purely event-driven simulation of analog systems without involving any numerical iteration. The *xreal* form in Eq. (2-1) has a Laplace s-domain equivalence:

$$x(t)=\sum_i c_i t^{m_i} e^{-a_i t} \xrightarrow{\mathcal{L}} X(s)=\sum_i \frac{c_i \times m_i!}{(s+a_i)^{m_i+1}} . \qquad \textbf{(2-2)}$$

This implies that if the s-domain transfer function of a linear system $H(s)$ is given, its response to the input $x(t)$ can be computed simply as a product of the two s-domain functions, $X(s)$ and $H(s)$. Assuming that the transfer function is expressed in the residue form,

$$H(s)=\sum_j \frac{q_j}{(s+p_j)^{n_j}} \qquad \textbf{(2-3)}$$

the process of computing the output is simply calculating the cross-products between the terms in $X(s)$ and $H(s)$ and adding them, as expressed by Eq. (2-4). The resulting cross-products can be decomposed back into the residue form via partial-fraction decomposition. It is noteworthy that this computation is fully algebraic; there is no numerical iteration or time-step integration involved.

$$\begin{aligned}\sum_i \frac{b_i}{(s+a_i)^{m_i}} \times \sum_j \frac{q_j}{(s+p_j)^{n_j}} &= \sum_{i,j} \frac{b_i}{(s+a_i)^{m_i}} \times \frac{q_j}{(s+p_j)^{n_j}} \\ &= \sum_{i,j} \left(\sum_{k=1}^{m_i} \frac{c_k}{(s+a_i)^k} + \sum_{l=1}^{n_j} \frac{d_l}{(s+p_j)^l} \right)\end{aligned} \qquad \textbf{(2-4)}$$

$$\text{where, } c_k = \begin{cases} \dfrac{b_i \cdot q_j}{(p_j - a_i)^{n_j}}, & for\ k = m_i \\ \dfrac{b_i \cdot q_j}{(p_j - a_i)^{m_i + n_j - k}} \cdot \dfrac{(-1)^{m_i - k}}{(m_i - k)!} \cdot \prod_{z=i}^{m_i - 1} \left(m_i + n_j - 1 - z \right), & for\ k = m_i - 1, \cdots, 1 \end{cases}$$

$$d_l = \begin{cases} \dfrac{b_i \cdot q_j}{(a_i - p_j)^{m_i}}, & for\ l = n_j \\ \dfrac{b_i \cdot q_j}{(a_i - p_j)^{n_j + m_i - l}} \cdot \dfrac{(-1)^{n_j - l}}{(n_j - l)!} \cdot \prod_{z=i}^{n_j - 1} \left(n_j + m_i - 1 - z \right), & for\ l = n_j - 1, \cdots, 1 \end{cases}$$

The described computation is purely event-driven; computation is required only once when the coefficients describing the input change. In the other words, when there is an event involving a change to the input coefficients, the coefficients of the output are updated according to Eq. (2-4), and the results remain valid until the next input event arrives. This stands in contrast to SPICE, where the output value keeps being updated between the events. In case of multiple input events, the output of the linear system can be evaluated based on the superposition principle; responses to each input event is computed as outlined in Eq. (2-4), and then the final output is a sum of those responses.

Fig. 2.5 illustrates the proposed event-driven computation using an RC filter. When a step input arrives at t_0, which is equivalent to a change in the coefficient c_i, this triggers a computation. As the RC filter has a transfer function of $1/(RCs+1)$, and the step input is $1/s$ in the s-domain, its output is simply a product of these two factors. The resulting output is $1/s - 1/(s+1/RC)$ according to Eq. (2-4), which corresponds to $1-e^{-t/RC}$ in the time-domain. This output is valid until the next input event, realizing therefore an event-driven simulation of analog blocks.

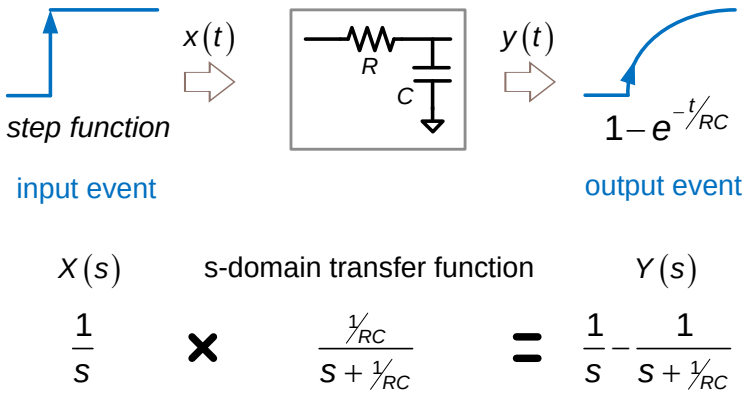


Fig. 2.5 The event-driven, s-domain computation of a linear RC filter response.

While initial conditions are omitted for brevity in Fig. 2.5, their effects should be considered once a system includes reactive elements. For instance, a Laplace transform of the governing differential equation of the RC filter results in two parts: a response to an input X(s)·1/(*RCs*+1) and a response to an initial condition *y*(0)·RC/(RC*s*+1), where *y*(0) is an initial voltage across a capacitor (Fig. 2.6). The response to initial conditions always results in the same form as the *xreal* form in Eq. (2-4), and hence it is additive to the responses to the input events.

Fig. 2.7 shows a pseudo-model of the RC-filter example in SystemVerilog using the proposed event-driven simulation method. The input and output variables, *x* and *y*, are defined as *xreal* types. Whenever an input event occurs (as indicated by *x.flag*), the instructions within the *always* statement are executed to compute the output event. First, initial conditions of the circuits are sampled using a DPI function *sample*(). Note that as analog signals are expressed in a functional form, initial conditions can be accurately computed at any time point. Then, the output responses

to the initial condition and the input event are computed by multiplying the sampled initial condition *y_init* and the input coefficients *x.param_set* with the transfer functions *tf_init* and *tf_in*, respectively. *tf_init* and *tf_in* are transfer functions from the initial condition and from the input event to the output response. *compute_coeff*() is a DPI function that performs the multiplying operation outlined in Eq. (2-4). Because the output event is always triggered simultaneously to the input event, *y.t_offset* is set equal to *x.t_offset*. Finally, the event flag *y.flag* is triggered to notify the subsequent blocks that the signal *y* has been updated.

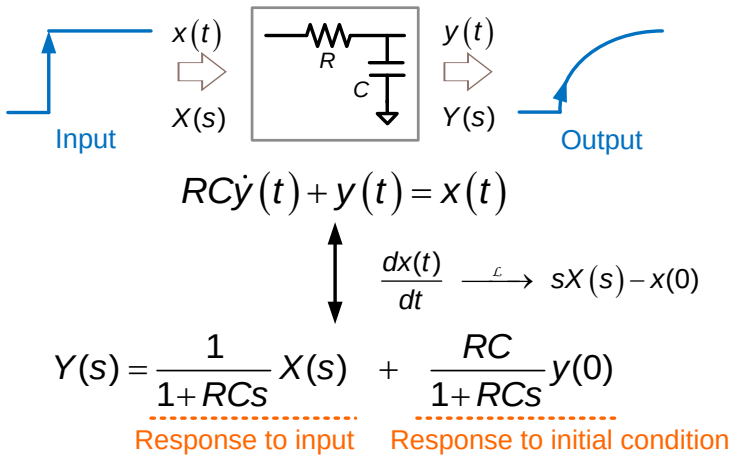


Fig. 2.6 A complete RC filter response with an initial condition included.

```
module rc_filter (
   input xreal x,
   output xreal y);

   always @(x.flag) begin
      y_init = sample( y.param_set );
      y.param_set = compute_coeff( x.param_set, tf_in )
                          +  compute_coeff( y_init, tf_init );
      y.t_offset = x.t_offset;
      -> y.flag;
   end

endmodule
```

Fig. 2.7 A pseudo-model of a linear RC filter in SystemVerilog.

Chapter 3
High-Speed I/O Interface Simulation

This chapter demonstrates how to apply the proposed method to model and simulate systems including time-invariant linear analog blocks through a high-speed I/O interface example. The exemplary high-speed I/O system consists of a transmitter with a charge-pump phase-locked loop (PLL), a receiver with a bangbang clock and data recovery (CDR) loop, continuous/discrete-time equalizers, and a channel (Fig. 3.1). The modeling accuracy of the system relies strongly on the accurate clock timing information and the analog signal waveform. For instance, it requires a sub-pico-second time resolution to measure the jitter performances of a PLL and a CDR. It also demands a sub-mV voltage resolution to simulate channel distortions and equalizer performances. Each sub-section will show that the proposed method can accurately model and simulate those sub-blocks in a truly event-driven fashion. In the final sub-section, the system-level behaviors are simulated and compared with Verilog-A models simulated in HSPICE.

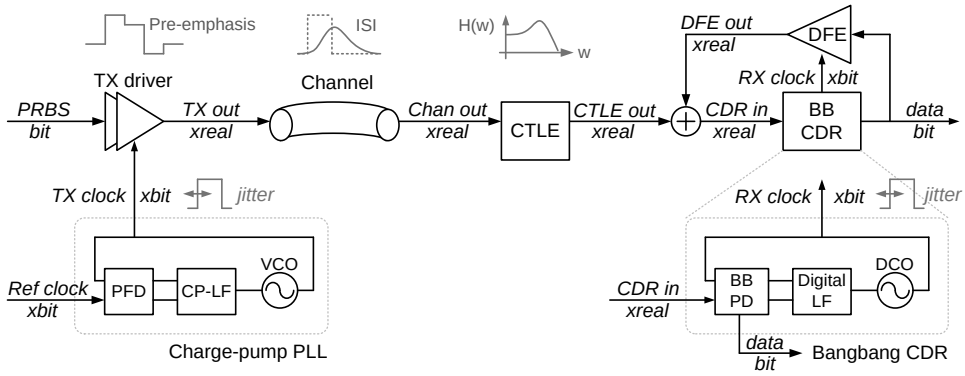


Fig. 3.1 An example of a high-speed I/O interface.

3.1 Charge-Pump Phase-Locked Loop

The first sub-block is a third-order charge-pump phase-locked loop (PLL) on the transmitter side. As the clock signal affects digital system performances, it is desirable to simulate such systems in digital simulators. However, a PLL is analog in nature, requiring an accurate simulation of a clock jitter and a loop filter waveform.

The exemplary charge-pump PLL is composed of a phase/frequency detector (PFD), a charge pump with a second-order loop filter (CP-LF), a voltage-controlled oscillator (VCO), and a frequency divider. The signals carrying the timing information, such as the reference input clock *in*, the VCO output clock *out*, the frequency-divided clock *div*, and the PFD output pulses *up*/*dn* are *xbit*-type signals, while the analog control voltage *vctrl* is an *xreal*-type signal (Fig. 3.2).

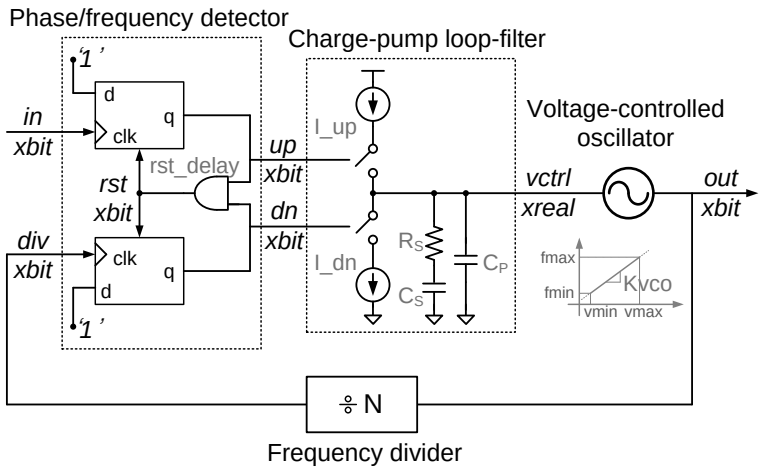


Fig. 3.2 A third-order CP-PLL example; the clock and timing-sensitive signals are defined as the *xbit*, while the analog signal is defined as the *xreal*.

3.1.1 Phase/Frequency Detector

With *xbit* representations, a PFD model can achieve a sub-time-step resolution for the timing difference between the input clock edges. The PFD employs two D flip-flops (DFF) with asynchronous reset and one AND gate. The DFF sets the output to '1' at the rising edge of the clock *clk* and resets the output to '0' at the rising edge of the reset *rst* (Fig. 3.3(a)). Therefore, its output *q* copies the timing offsets of *clk* for the rising edge and *rst* for the falling edge. In this way, all the timing information of *clk* and *rst* can be transferred to the output without degrading the accuracy. The AND gate produces the output '1' when both DFF outputs rise. Its timing offset for the rising edge is determined by the *t_offset* of the signals arriving later, as shown in Fig. 3.3(b). The pseudo-models of the DFF and the AND gate are given in Fig. 3.4 and Fig. 3.5.

Moreover, not only the signal timing offset but also the gate delay can be accurately simulated regardless of the simulator's time step. Especially, the AND gate delay on the reset path, *rst_delay*, plays an important role in the PLL non-idealities [24]. Fig. 3.3(b) depicts a case in which the gate delay is not a multiple integer of the time step; for instance, when the AND gate input arrives at a time of 0.7 with a gate delay of 1.5, the output rises at a time of 2.2. A model of this in SystemVerilog is described in Fig. 3.5. First, the AND gate determines the Verilog delay in a quantized time step (*ceil*(*delay*+*t_offset*)). Then, after this quantized delay, it sets the output value to '1' and computes its timing offset. In Fig. 3.3(b), the AND gate waits for two time steps and sets the output to '1' with a *t_offset* of -0.8.

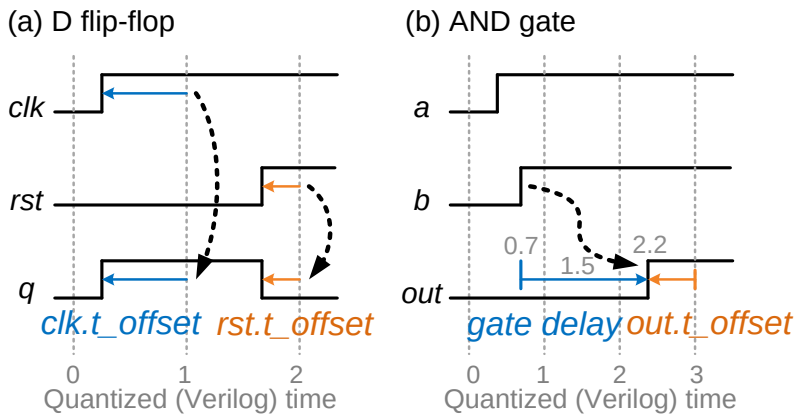


Fig. 3.3 Modeling with accurate timing: (a) the D flip-flop and (b) the AND gate.

```
module dff (
    output xbit q,
    input_xbit d,
    input xbit clk,
    input xbit rst);

    always @(posedge clk.value) begin
        q.value = d.value;
        q.t_offset = clk.t_offset;
    end

    always @(posedge rst.value) begin
        q.value = 0;
        q.t_offset = rst.t_offset;
    end
endmodule
```

Fig. 3.4 A pseudo-model of the D flip-flop in SystemVerilog.

```
module and (
   output xbit out,
   input xbit a,
   input xbit b);

   always @(posedge a.value or posedge b.value) begin
      if (a.value ==1 && b.value ==1) begin
          t_offset = (a.t_offset > b.toffset)?
                                    a.t_offset: b.t_offset;
          #(ceil(delay+t_offset)) out.value = 1;
          out.t_offset = delay+t_offset - ceil(delay+t_offset);
      end
   end

   always @(negedge a.value) begin
          t_offset = a.t_offset;
          #(ceil(delay+t_offset)) out.value = 0;
          out.t_offset = delay+t_offset - ceil(delay+t_offset);
   end

   always @(negedge b.value) begin
          t_offset = b.t_offset;
          #(ceil(delay+t_offset)) out.value = 0;
          out.t_offset = delay+t_offset - ceil(delay+t_offset);
   end
endmodule
```

Fig. 3.5 A pseudo-model of the AND gate in SystemVerilog.

3.1.2 Charge-Pump Loop Filter

A charge-pump loop filter generates the control voltage in an event-driven fashion by computing the filter response only when the *up* or *dn* signal changes. The charge pump applies a positive current when *up* is high and a negative current when *dn* is high (Fig. 3.6). This current flows into the following loop filter, which is composed of a resistor, *Rs*, and two capacitors, *Cs* and *Cp*. The output voltage *vctrl* is a product of the charge-pump current *ictrl* and the impedance transfer function *Z*(*s*):

$$Z(s) = \frac{vctrl(s)}{ictrl(s)} = \frac{R_S C_S s + 1}{R_S C_S C_P s^2 + (C_S + C_P)s} \quad . \qquad \textbf{(3-1)}$$

The charge-pump pseudo-model is given in Fig. 3.7. The charge-pump determines the value of the output current *I_cur* upon every change in the *up*/*dn* signals. The *xreal* representation of this current is a constant with the parameter set of {(*I_cur*, 0, 0)}. The *t_offset* of the output current *ictrl* is identical to that of the *up*/*dn* signal. The change in *ictrl* triggers the computation of the output voltage *vctrl*, which is then computed as a response of a linear filter with a transfer function of *Z*(*s*) to the *ictrl* input as described in Chapter 2.3.

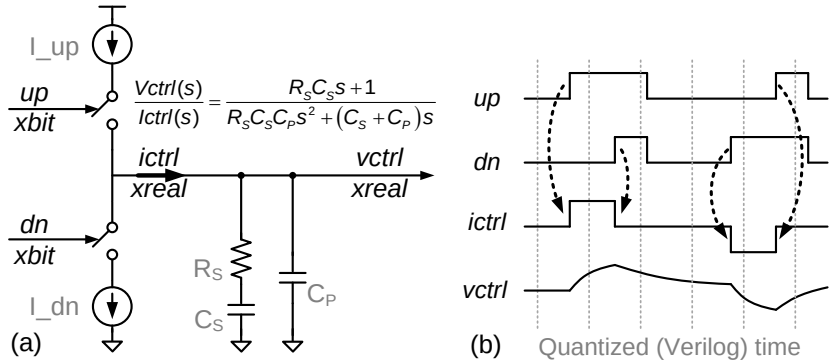


Fig. 3.6 (a) The charge pump with a second-order loop filter, and (b) signal waveforms illustrating its operation.

```
module cp_lf (
   output xreal vctrl,
   input xbit up,
   input xbit dn);

   chandle tf;     // impedance of loop filter

   always @(up.value) begin
      I_cur = I_up*up.value − I_dn*dn.value;
      ictrl.param_set = create_params(I_cur,0,0);
      ictrl.t_offset = up.t_offset;
      ->ictrl.flag;
   end

   always @(dn.value) begin
      I_cur = I_up*up.value − I_dn*dn.value;
      ictrl.param_set = create_params(I_cur,0,0);
      ictrl.t_offset = dn.t_offset;
      ->ictrl.flag;
   end

   always @(Ictrl.flag) begin
      vctrl.param_set = compute_coeff(ictrl.param_set, tf);
      vctrl.t_offset = ictrl.t_offset;
      ->vctrl.flag;
   end
endmodule
```

Fig. 3.7 A pseudo-model of the charge-pump loop filter in SystemVerilog.

3.1.3 Voltage Controlled Oscillator

The voltage-controlled oscillator (VCO) model generates an *xbit*-type clock of which frequency is controlled by an *xreal*-type input *vctrl*. The frequency of the oscillator is a linearly scaled version of the input, and its phase is an integral of that frequency (Fig. 3.8). This integration can be carried by multiplying $1/s$ in the s-domain without time-step integration. This is in contrast to other simulators which must accumulate the frequency for every time step, thus degrading the simulation speed. Finally, the *xbit*-typed output clock is toggled every time the *xreal*-type phase reaches the value of π.

A pseudo-model of the VCO in SystemVerilog is shown in Fig. 3.9. When the input control voltage *vctrl* has an event, the VCO updates its frequency signal by scaling *vctrl* using a DPI function *scale*(). The phase signal is then computed by multiplying the frequency signal by $1/s$ in the s-domain, after which the DPI function *find_cross*() checks when the phase signal crosses π. Based on the crossing time *t_cross*, the timing offset *t_offset* and the quantized time stamp at which the output event needs to be scheduled are determined. For instance, if *t_cross* is 2.5ns and the Verilog time step is 1ns, the VCO model waits for three time steps (3ns) and toggles the output with a *t_offset* of -0.5ns. When the scheduled output event is triggered, the phase is shifted by $-\pi$ using the DPI function *subtract*() to prevent the phase from overflowing. At the same time, the next π-crossing event is scheduled, and the clock continues to toggle whenever the phase crosses π.

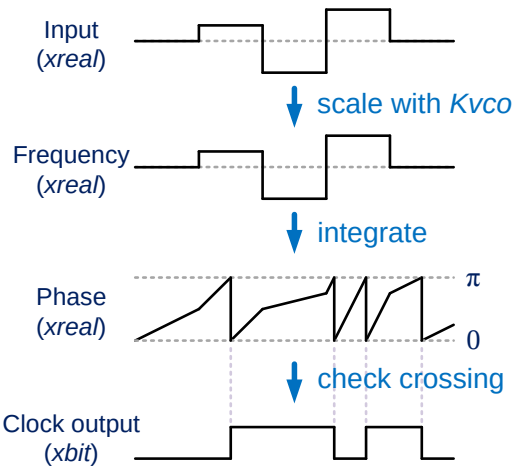


Fig. 3.8 The voltage-controlled oscillator generates its digital clock output based on *xreal*-type frequency and phase signals.

```
module vco (
   output xbit out,
   input xreal vctrl);

   always @(vctrl.flag) begin
      freq.param_set = scale(vctrl.param_set, Kvco);
      tf = create_params(1,0,0);
      phase.param_set = compute_coeff(freq.param_set, tf);
      t_cross = find_cross(phase.param_set);
      #(ceil(t_cross)) -> out_event;
   end

   always @(out_event) begin
      out.value = ~out.value;
      out.t_offset = t_cross - ceil(t_cross);
      phase.param_set = subtract(phase.param_set,pi);
   end

   always @(out.value) begin
      t_cross = find_cross(phase.param_set);
      #(ceil(t_cross)) -> out_event;
   end
endmodule
```

Fig. 3.9 A pseudo-model of the voltage-controlled oscillator in SystemVerilog.

3.1.4 Frequency Divider

A synchronous divider is modeled by aligning the output clock timing to the input clock timing. The divider model counts the number of rising transitions of the input and toggles its output whenever it reaches a predetermined division factor. If there is no delay, the *t_offset* value of the output is set equal to that of the triggering input. Fig. 3.10 illustrates this operation with a division factor of 4. Fig. 3.11 shows a pseudo-model of the frequency divider in SystemVerilog.

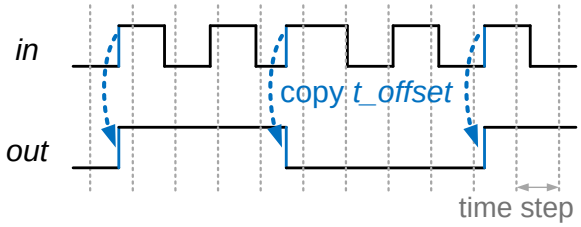


Fig. 3.10 The input and output waveforms of the synchronous frequency divider with a division factor of 4.

```
module freq_div (
    output xbit out,
    input xbit in);

    always @(posedge in.value) begin
        count = count + 1;
        if (count == div_factor/2) begin
            out.value = 1;
            out.t_offset = in.t_offset;
        end
        else if (count == div_factor) begin
            out.value = 0;
            out.t_offset = in.t_offset;
            count = 0;
        end
    end

endmodule
```

Fig. 3.11 A pseudo-model of the frequency divider in SystemVerilog.

3.1.5 Simulation Results

The described charge-pump PLL model is simulated with the design parameters listed in Table 3.1. The initial VCO output frequency is 1.5GHz at a control voltage *vctrl* of 0V. Its target output frequency is 2.0GHz for a reference frequency of 0.5GHz and a frequency division factor of 4. To verify the accuracy, the simulation results are compared with those of Verilog-A models simulated in HSPICE. The Verilog-A models of the PFD with the charge pump, digital logic gates and VCO are those found in [25]-[26], and the loop filter is modeled using the *laplace_nd* function [6].

Fig. 3.12 shows the locking transient waveform of the VCO control voltage *vctrl* in comparison with the Verilog-A waveform. The value of *vctrl* is increased from its initial value of 0V and is locked at 0.5V. This value corresponds to a VCO frequency of 2.0GHz for an initial frequency of 1.5GHz and a *Kvco* of 1GHz/V. The zoomed-in view in Fig. 3.12(b) shows that the simulated waveform is in good agreement with the Verilog-A waveform. The measured maximum and the root

Table 3.1. Design parameters for the charge-pump PLL simulation

Module	Parameter	Description	Value
PFD	rst_delay	AND gate delay on a reset path	30ps
CP-LF	I_up/I_dn	Up/down current amplitude	20μA
	Rs	Series resistance	20kΩ
	Cs	Series capacitance	500fF
	Cp	Parallel capacitance	50fF
VCO	Kvco	Voltage-to-frequency gain	1GHz/V
Freq. Div.	N	Frequency division factor	4

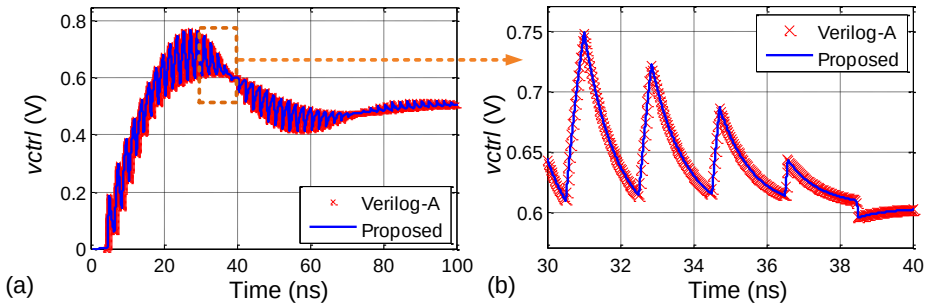


Fig. 3.12 (a) The locking transient waveform of the input control voltage of the VCO, and (b) its zoomed-in view.

mean square (RMS) differences between the two waveforms are 0.74mV and 0.23mV, respectively.

Fig. 3.13 and Fig. 3.14 show the simulated jitter histogram and jitter transfer function. In Fig. 3.13, the jitter histogram of the output clock signal is simulated when the PLL is locked to a reference at 0.5 GHz with a RMS jitter of 4ps. The simulated jitter histogram shows a standard deviation of 1.86ps, which matches well with a Verilog-A result of 1.82ps. Note that the proposed method uses a time step as coarse as 10ps, yet still can obtain a jitter histogram with a fine resolution. Fig. 3.14 shows the jitter transfer function. The jitter transfer function is simulated by applying a sinusoidal jitter to the reference clock and measuring the amplitude of the resulting sinusoidal jitter of the output clock. Fig. 3.14 plots the ratio between the input and output jitter amplitudes when the jitter frequency is swept from 100kHz to 0.5GHz. The simulated results are in good agreement with the theoretical jitter transfer function of a third-order charge-pump PLL, given as:

$$H(s)=\frac{T(s)}{1+T(s)/N},$$

$$\text{where } T(s)=\frac{I_{cp}\cdot Kvco}{2\pi s}\frac{R_S C_S s+1}{R_S C_S C_P s^2+(C_S+C_P)s}. \quad \textbf{(3-2)}$$

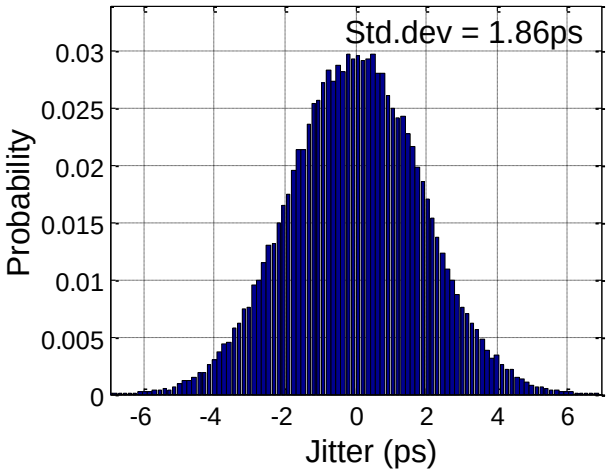


Fig. 3.13 The Simulated jitter histogram of the output clock.

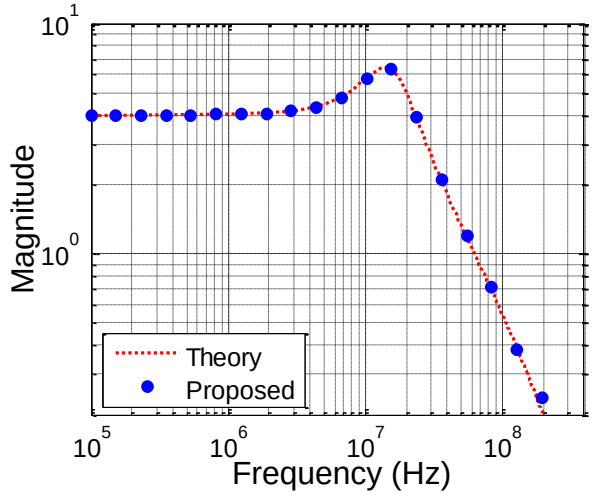


Fig. 3.14 The simulated jitter transfer function of the charge-pump PLL.

To assess the non-ideality effects, the reference spurs and static phase offsets are measured as a function of the mismatch between the up and down currents. Fig. 3.15 shows the reference spurs at 0.5GHz in the power spectral density $S_\phi(f)$ of the output phase when the up and down currents are mismatched by 20%. In Fig. 3.15(a), the current mismatch causes a reference spur of -133dBc, which is in good agreement with the Verilog-A results in Fig. 3.15(b). Fig. 3.16(a) plots the reference spur level as the mismatch varies from 0 to 30%. These results show good agreement with the Verilog-A results except when the mismatch is 5%, where the reference spur is smaller than the noise level of -145dBc. The current mismatch also causes the phase offset between the reference and the output clock signals. Fig. 3.16(b) shows that the static phase offset increases to 25ps when the charge-pump current mismatch reaches 30%.

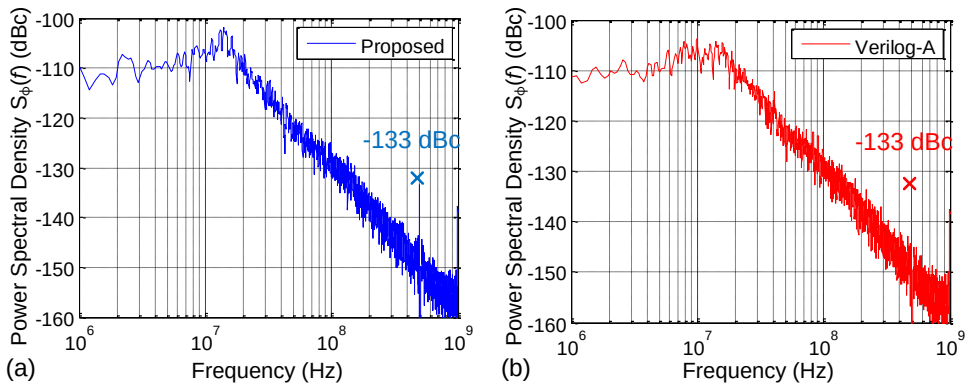


Fig. 3.15 The power spectral densities of the output phase simulated with (a) the proposed method and (b) the Verilog-A model, when the up and down currents show a mismatch of 20%.

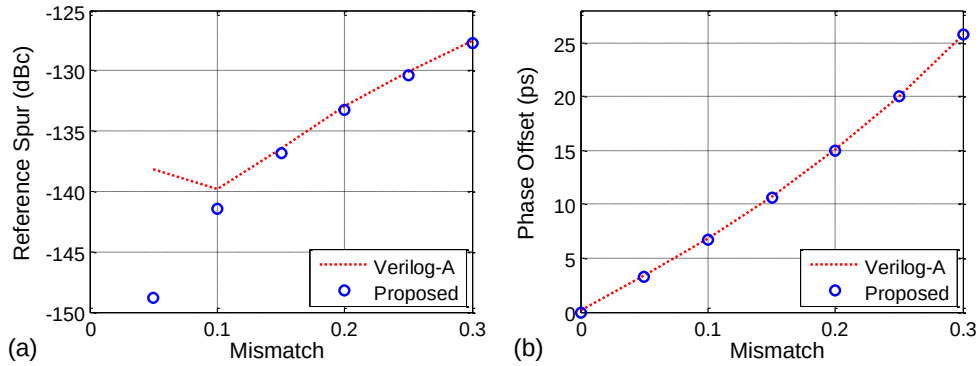


Fig. 3.16 (a) The reference spurs of the output clock and (b) static phase offsets between the input and output clocks as a function of the charge-pump current mismatch.

3.2 Bangbang Clock and Data Recovery

The second sub-block is a bangbang clock and data recovery (CDR) circuit. The bangbang CDR is often considered pure digital circuits as the loop filter is digital and the oscillator is digitally controlled. However, the timing information of clock signals is critical in the phase detector, and the oscillator is still an analog block. Therefore, the bangbang CDR cannot be accurately simulated in a pure digital simulator.

This sub-section provides modeling details and simulation results of a bangbang CDR. An bangbang CDR example consists of three blocks (Fig. 3.17): an Alexander phase detector (PD), a digital loop filter (LF), and a digitally controlled oscillator (DCO). The difference with regard to the previous PLL model is that the PD output takes a binary value and the DCO is controlled by a digital value. Therefore, the loop filter is implemented in digital and described in pure Verilog.

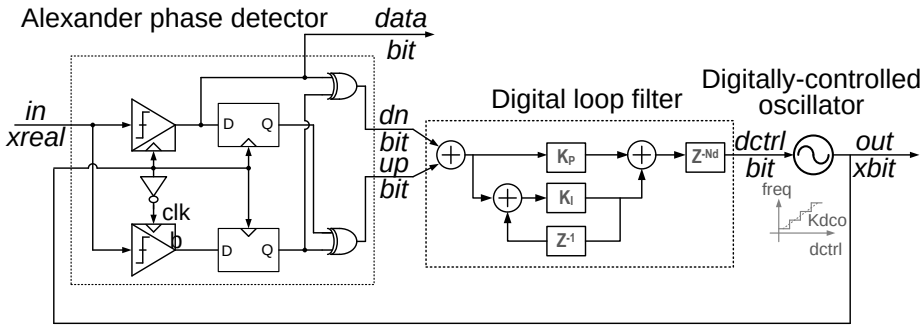


Fig. 3.17 The block diagram of the bangbang clock and data recovery example.

3.2.1 Alexander Phase Detector

The Alexander PD employs two analog comparators, two DFFs, and two XOR gates. One of the comparators determines the data at the clock's rising edges, while the other extracts the edges at the clock's falling edges. The following DFF and XOR gates determine whether the timing is early or late; their models are similar to the logic gate models in the charge-pump PLL example. While these DFFs and XORs consume *xbit*-type signals, their outputs, *data*, *up*, and *dn*, are converted to *bit*-type signals using *assign* (Fig. 3.18). This conversion occurs because the subsequent digital loop filter is purely digital.

Fig. 3.19 outlines the analog comparator model in the Alexander PD. The analog comparator computes the sampled value based on the input's functional representation. Therefore, this model accurately compares the input signal independently of the simulator's time step. For example, if the input is an exponential signal of $exp(-10^9/t)$ and the clock is triggered at a Verilog time of 1ns with a time offset of -0.6ns, the sampled value is $exp(-10^9/0.4ns)$. The sampling is performed in C using the DPI function *sample*(). Finally, the comparator output is determined by comparing the sampled value with the comparator threshold.

```
module alexander_pd (
   output bit data,
   output bit up,
   output bit dn,
   input xbit clk,
   input xreal in);

   comparator data_comp(data_d, in, clk);
   comparator edge_comp(edge_d, in, clkb);

   dff data_dff(data_q, data_d, clk);
   dff edge_dff(edge_q, edge_d, clk);

   inv clk_inv(clkb, clk);
   xor up_xor(up_xbit, data_q, edge_q);
   xor dn_xor(dn_xbit, data_d, edge_q);

   /* casting to bit-type */
   assign data = data_d.value;
   assign up = up_xbit.value;
   assign dn = dn_xbit.value;
endmodule
```

Fig. 3.18 A pseudo-model of the Alexander phase detector in SystemVerilog.

```
module comparator (
   output xbit out,
   input xreal in,
   input xbit clk);

   always @(posedge clk.value) begin
      t_clk = $realtime + clk.t_offset;
      in_value = sample(in.param_set, t_clk);
      out.value = (in_value > threshold)? 1: 0;
      out.t_offset = clk.t_offset;
   end
 endmodule
```

Fig. 3.19 A pseudo-model of the comparator in SystemVerilog.

3.2.2 Digital Loop Filter

The digital loop filter can be described in pure Verilog. In this example, the digital loop filter has a proportional and an integral path (Fig. 3.17). The integral path accumulates the input with an integral gain of *Ki*, and the proportional path scales the input with a proportional gain of *Kp*. These two values added to determine the output, and the output is delayed by *Nd* clock cycles to model the filter delay. Fig. 3.20 shows a digital loop filter model in Verilog.

```
module digital_lf (
   output reg[width_out-1:0] out,
   input xbit clk,
   input bit up,
   input_bit dn);

   reg [width_out-1:0] out_d  [Nd-1:0];
   reg [width_out-1:0] acc;
   reg [width_out-1:0] out_p;

   always @(posedge clk) begin
      acc = acc + Ki*(up - dn);
   end

   always @(acc or up or dn) begin
      out_p = Kp*(up - dn) + acc;
   end

   assign out = out_d[Nd-1];
   always @(posedge clk) begin
      for (i=Nd-1; i>0; i--) out_d[i] = out_d[i-1];
      out_d[0] = out_p;
   end

endmodule
```

Fig. 3.20 The digital loop filter description in pure Verilog.

3.2.3 Digitally Controlled Oscillator

The modeling of a digitally controlled oscillator (DCO) is identical to that of the VCO in the charge-pump PLL example except that its input signal is a *bit*-type signal. The DCO frequency is the input *dctrl* linearly scaled by a factor of *Kdco*. Its phase is obtained by integrating the frequency, and the output *out* is toggled every time the phase crosses π.

```
module dco (
   output xbit out,
   input bit [width_dctrl-1:0] dctrl);

   always @(dctrl) begin
      freq.param_set = create_params(Kdco*real'(dctrl),0,0);
      tf = create_params(1,0,0);
      phase.param_set = compute_coeff(freq.param_set, tf);
      t_cross = find_cross(phase.param_set);
      #(ceil(t_cross)) -> out_event;
   end

   always @(out_event) begin
      out.value = ~out.value;
      out.t_offset = t_cross - ceil(t_cross);
      phase.param_set = subtract(phase.param_set,pi);
   end

   always @(out.value) begin
      t_cross = find_cross(phase.param_set);
      #(ceil(t_cross)) -> out_event;
   end

endmodule
```

Fig. 3.21 A pseudo-model of the digitally-controlled oscillator in SystemVerilog.

3.2.4 Simulation Results

The bangbang CDR example is simulated with the design parameters listed in Table 3.2. The bangbang CDR input is a 2-Gbps pseudo random bit sequence (PRBS) data stream which is transmitted through a lossy channel. The transmitter and channel models are identical to those explained in the next sub-section.

Fig. 3.22 shows the locking transient waveforms of the CDR. Fig. 3.22(a) plots its DCO frequency as a function of time. The DCO frequency is initially 1.984GHz and increases to 2GHz. The sudden changes in the frequency arise from its proportional path, which increases the phase of the DCO instantaneously. Fig. 3.22(b) and (c) compare the recovered data during this locking transient period. The recovered data show errors when the CDR frequency is lower than the input data rate (Fig. 3.22(b)), while the recovered data match the input data when the CDR frequency is at the input data rate (Fig. 3.22(c)).

The effects of the digital loop filter delay and the input transition density on the jitter transfer function are investigated. Fig. 3.23 shows the jitter transfer functions for different digital loop filter delays. As the filter delay increases from 2 to 16, the

Table 3.2. Design parameters for the Bangbang CDR simulation

Module	Parameter	Description	Value
Digital LF	*Kp*	Proportional gain	256
	Ki	Integral gain	1
	Nd	Digital loop filter delay	4 clock cycles
	Width	Bit-width of digital output	14
DCO	*Kdco*	Digital-to-frequency gain	2^{-14} GHz/LSB

peaking of the transfer function increases. Fig. 3.24 compares the jitter transfer functions when two different inputs of a pseudo random bit sequence (PRBS) data pattern and a '1010' data pattern are applied. This figure illustrates that the CDR bandwidth is wider for the '1010' pattern than for the PRBS pattern. This wider bandwidth arises due to the higher transition density of the '1010' pattern, which allows the CDR to lock faster with more frequent updates of the control signals.

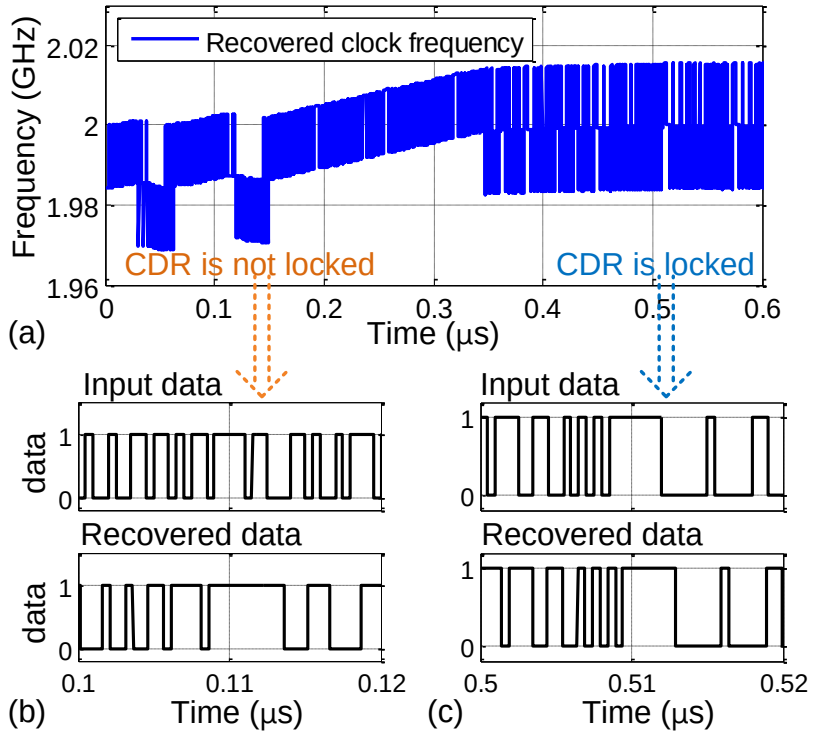


Fig. 3.22 The locking transient waveform of the CDR clock frequency and its recovered data when (b) the CDR is not locked and (c) the CDR is locked.

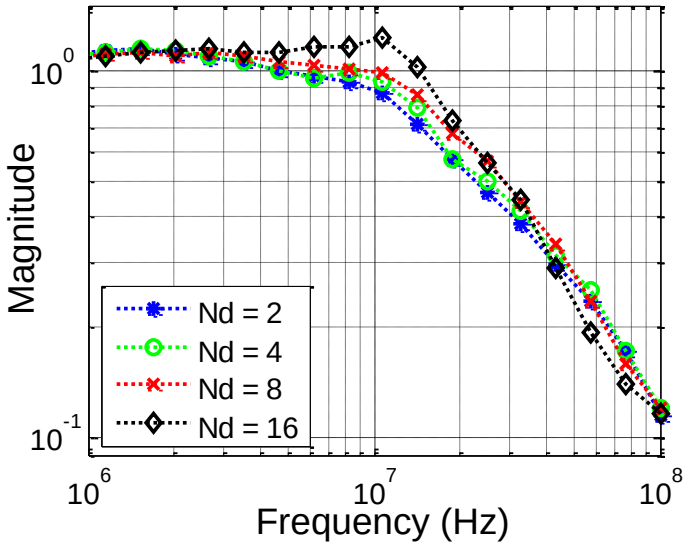


Fig. 3.23 The jitter transfer functions with different digital loop filter delays.

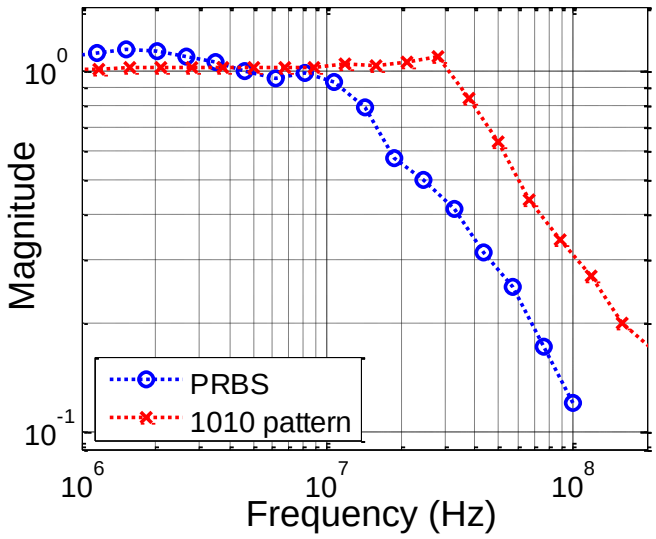


Fig. 3.24 The jitter transfer functions with different input data patterns.

3.3 Channel and Equalizers

This sub-section explains modeling and simulation of a channel and three equalization schemes: a transmitter-side pre-emphasis equalization, and a receiver-side continuous-time linear equalization (CTLE) and a decision-feedback equalization (DFE) as shown in Fig. 3.25.

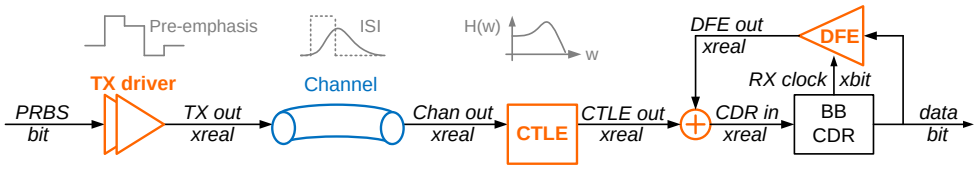


Fig. 3.25 A high-speed I/O interface with three equalization techniques: a pre-emphasis equalizer, a continuous-time linear equalizer (CTLE), and a decision-feedback equalizer (DFE).

3.3.1 Channel and Continuous-Time Linear Equalizer

The channel and CTLE are modeled as linear filters with s-domain transfer functions. The channel transfer function is extracted from the s-parameters by fitting S21 into a rational polynomial form using the *rationalfit* function in Matlab. Fig. 3.26(a) shows the transfer function extracted with 47 poles, when the delay of 3.001ns is factored out. The CTLE transfer function has a high-pass characteristic with one zero and two poles. Fig. 3.26(b) plots its transfer function when it has a zero at 0.5GHz and poles at 1 and 2GHz. Then, these linear analog behavior can be modeled in the same way as described in Section 2.3.

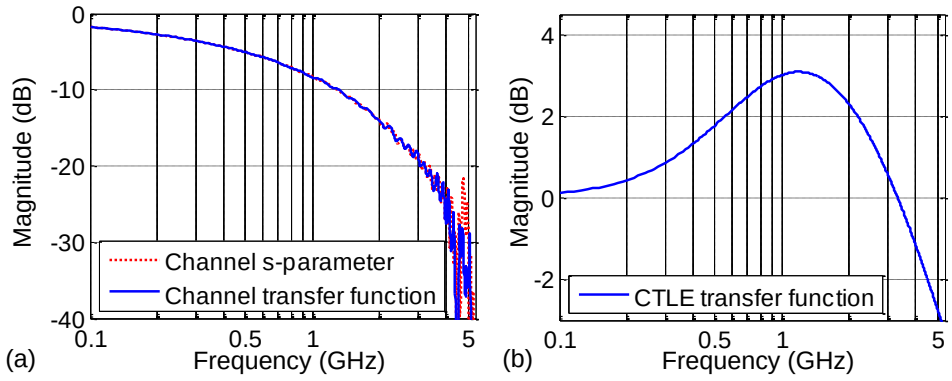


Fig. 3.26 (a) The channel transfer function extracted from its measured S-parameter, and (b) the CTLE transfer function with one zero at 0.5GHz and two poles at 1.0GHz and 2.0GHz, respectively.

3.3.2 Pre-Emphasis and Decision-Feedback Equalizer

The pre-emphasis equalization and DFE can be modeled using a finite impulse response (FIR) filter, which determines the output as a weighted sum of the input data. For instance, Fig. 3.27 shows the output signal with weighting coefficients of {0.8, -0.2}. Fig. 3.28 outlines the FIR filter model in SystemVerilog. The FIR filter computes the output value and triggers an output event at every rising clock edge. If the output has a finite transition time, two events are required to define the start and end points of the transition; the first event starts the transition with a finite transition slope, and the second event ends the transition with the final value after the transition time.

The analog adder combines the CTLE output and the DFE output. As the CTLE and DFE outputs are in the *xreal* functional form, their addition is a linear combination of those functions. For example, if the CTLE output is an exponential function of c_1te^{-at} and the DFE output is a step function of c_2, the added output is a sum of those two functions, $c_1te^{-at} + c_2$, as shown in Fig. 3.29. Therefore, the parameter set of the adder output is a union of the parameter sets of the CTLE and DFE outputs. This operation in SystemVerilog is outlined in Fig. 3.30. Whenever either input changes, the output parameter set is updated to a union of input parameter sets using the DPI function *union*().

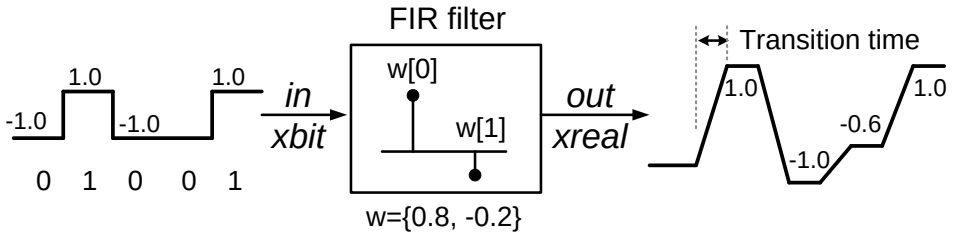


Fig. 3.27 The output signal of the FIR filter with a finite transition time.

```
module fir_filter (
   output xreal out,
   input xbit in,
   input xbit clk);

   always @(posedge clk.value) begin
      value = 0.8*in.value − 0.2*p_in;

      /* start the transition */
      slope = (value − p_value)/t_tran;
      out.param_set = create_params(slope, 0, 1);
      out.t_offset = clk.t_offset;
      ->out.flag;

      /* finish the transition */
      #(t_tran);      // transition time
      out.param_set = create_params(value, 0, 0);
      out.t_offset = clk.t_offset;
      ->out.flag;

      p_value = value; p_in = in.value;
   end
endmodule
```

Fig. 3.28 A pseudo-model of the FIR filter in SystemVerilog.

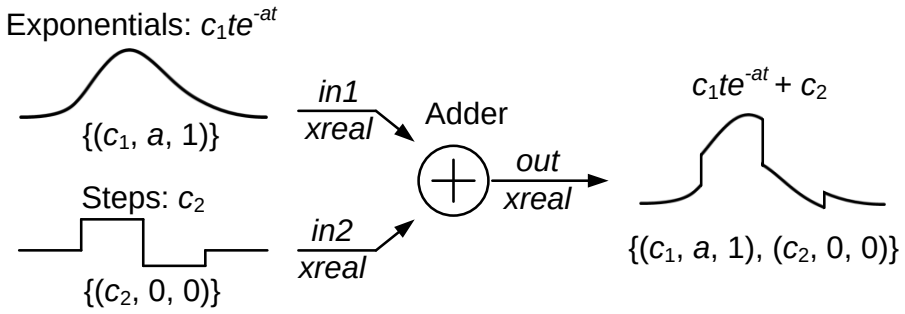


Fig. 3.29 The addition of two *xreal* signals is a linear combination of functions, and can be modeled as a combination of input parameter sets.

```
module adder (
   output xreal out,
   input xreal in1,
   input xreal in2);

   always @(in1.flag) begin
      out.param_set = union(in1.param_set, in2.param_set);
      out.t_offset = in1.t_offset;
      ->out.flag;
   end

   always @(in2.flag) begin
      out.params = union(in1.param_set, in2.param_set);
      out.t_offset = in2.t_offset;
      ->out.flag;
   end
endmodule
```

Fig. 3.30 A pseudo-model of the analog adder in SystemVerilog.

3.3.3 Simulation Results

The design parameters of the high-speed I/O interface example are as follows. The weighting coefficients of the pre-emphasis and decision-feedback equalizers are {0.8, -0.2} and {-0.14, -0.03}. The continuous-time linear equalizer has one zero at 0.5GHz and two poles at 1 and 2GHz. The transmitted data rate is 5Gb/sec.

Fig. 3.31 shows the analog waveforms of the transmitter (TX) driver, channel, CTLE, and DFE adder. The analog events are denoted with the circles, and these events propagate multiple analog blocks without introducing any additional events. In Fig. 3.31(a), the TX driver generates two events per input bit to start and end the ramp signal. In Fig. 3.31(b) and Fig. 3.31(c), these events propagate through the channel and the CTLE without adding any new event, yet still describing accurate continuous-time waveforms. In Fig. 3.31(d), the events from the CTLE and the DFE are combined by the adder. Fig. 3.31(e)-(h) compare the waveforms with Verilog-A models simulated in HSPICE. For the Verilog-A model, the digital logic gates and oscillators are based on the Verilog-A model library provided in [25]-[26], while the analog filters are modeled using the *laplace_nd* function [6]. The simulation results of the proposed method are in good agreement with the Verilog-A models. The measured maximum and the root-mean square (RMS) differences between those waveforms are 0.12mV and 32μV for a time period of [0,100ns], respectively.

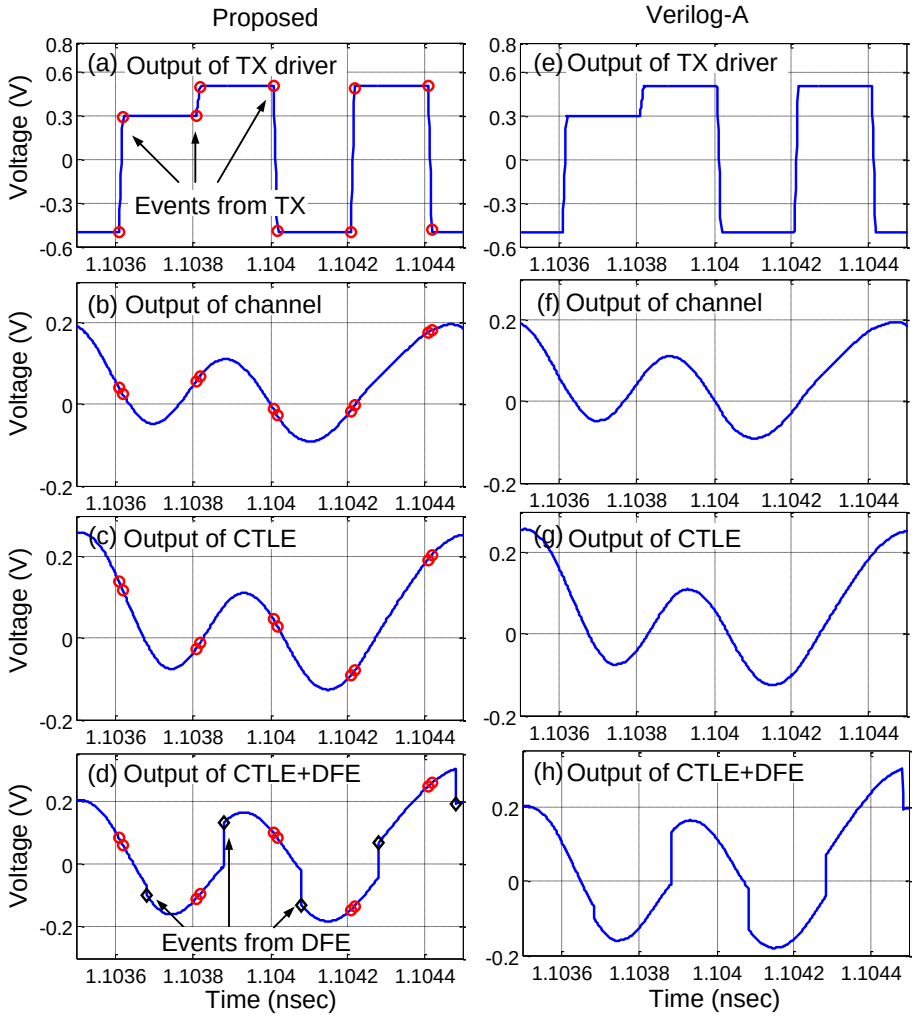


Fig. 3.31 The waveforms of (a) the TX driver output, (b) the channel output, (c) the CTLE output, and (d) the adder output, simulated with the proposed method; (e), (f), (g), and (h) the waveforms of the same signals simulated with Verilog-A models in HSPICE.

3.4 High-Speed I/O System Simulation

This sub-section summarizes system-level simulation results; the simulation includes all the sub-blocks outlined in previous sub-sections. The data rate is increased to 5Gb/s (from 2Gb/s in the bangbang CDR example) by the aid of equalizers. The charge-pump PLL and bangbang CDR use the same design parameters as the preceding examples, except that the center frequencies of the VCO and DCO are 5GHz.

Fig. 3.32 and Fig. 3.33 show simulated eye diagrams in comparison with the Verilog-A model. Fig. 3.32 shows the three eye diagrams of the channel, the CTLE, and the DFE adder, showing that their eye openings are enlarged as the signal goes through the equalizers. Fig. 3.33(a) and Fig. 3.33(b) compare eye diagrams simulated with different time steps of 10ps and 10fs. These two eye diagrams are identical, demonstrating that the accuracy is independent of the simulation time step. Fig. 3.33(b) and Fig. 3.33(c) compare eye diagrams simulated with the proposed method and the Verilog-A model. These figures illustrate that the simulation results of the proposed method are in good agreement with those of the Verilog-A model.

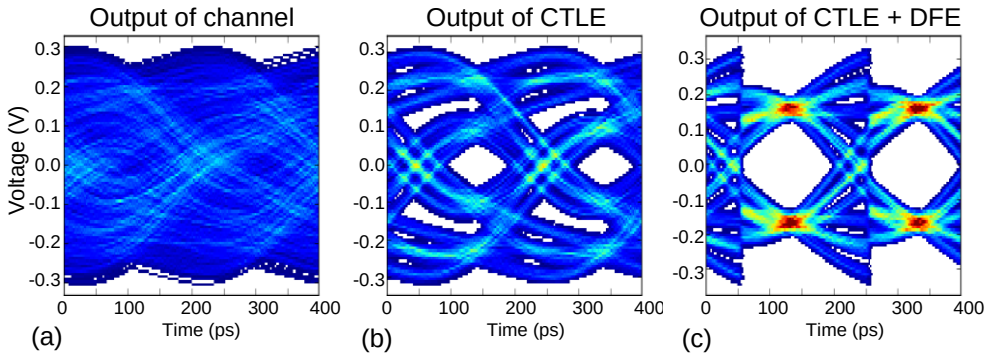


Fig. 3.32 Eye diagrams of (a) the channel output, (b) the CTLE output, and (c) the adder output.

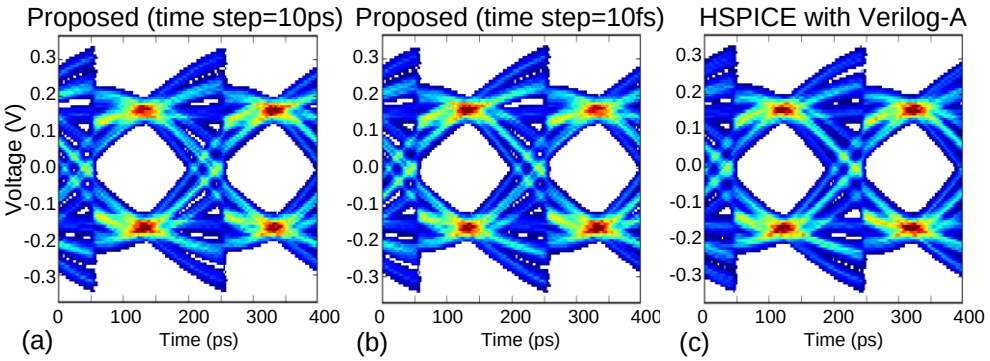


Fig. 3.33 (a), (b) Eye diagrams simulated with the proposed method with simulation time steps of 10ps and 10fs, respectively, and (c) eye diagram simulated with Verilog-A models in HSPICE.

Fig. 3.34(a) shows the number of events processed in the TX driver, channel, CTLE, and DFE adder for one million input data. The number of events in each block remains the same regardless of the simulation time step, demonstrating that the proposed method is truly event-driven. Fig. 3.34(b) summarizes the simulation runtime on a Linux machine with an Intel Core i5-3570 CPU. The runtime of the proposed method for one million bits is 226 sec for a time step of 10ps, which is equivalent to 4400 bits/sec. The runtime increases slightly to 246 sec (by 8%) for a time step of 10fs, but this increase is minimal considering that the time step becomes finer by a factor of 1000. These steady simulation speeds arise because the amount of computation is unaffected by the simulation time step due to the event-driven simulation. For reference, the simulation runtime of the Verilog-A model was 168min for one million bits, which is 45× slower than the proposed method.

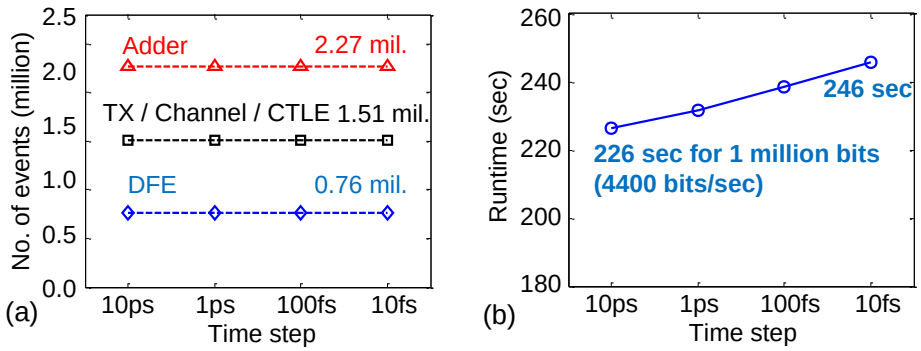


Fig. 3.34 (a) The number of events processed in each block, and (b) the simulation runtimes for one-million bit input with different time steps.

Chapter 4

Switching Power Supply Simulation

A switching power supply includes one or more switches which change the system input-to-output relationship depending on their connections, but the system can be modeled as a linear time-invariant system between switching instants (i.e. a switched linear system). The switching power supplies like a boost converter, a buck converter, and a switched-capacitor converter are typical examples of a switched linear system (Fig. 4.1) [27]. The main difficulty in simulating them is that the time-integration methods of analog simulators cannot efficiently handle such abrupt changes due to switching activities [28]. Moreover, most switching power supplies are characterized by high-frequency switching activities that demand fine-grained simulation for accuracy, but also by slow transients that require long simulation times.

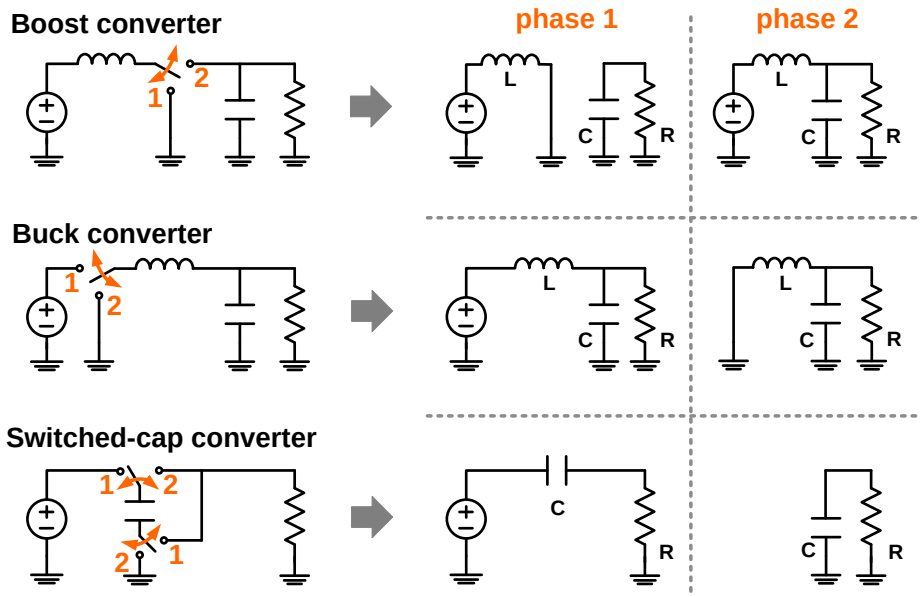


Fig. 4.1 Switched-linear system examples.

This chapter demonstrates how to apply the proposed method to switched linear systems and compares its simulation accuracy and speed with existing analog simulators. The main difference from linear system simulation described in Chapter 3 is that switched linear system is modeled with multiple transfer functions, each of which describes the circuit network during each switching phase. At every switching event, one transfer function is selected depending on the switch connections and compute the output response. In other words, in addition to the input events, the switching event can also trigger an output event. The switched linear system simulation is demonstrated through two power converter examples, a power factor correction (PFC) boost converter [33] and a time-interleaved switched-capacitor DC-DC converter [34].

4.1 Boost Converter

Boost converters are widely used for DC/AC-to-DC power conversion with an output voltage greater than its input voltage. A basic architecture is composed of an inductor, a capacitor, a resistor, and a switch (Fig. 4.2). The boost converter has two operation phases alternated by a switch connection as shown in Fig. 4.2(a). With the switch in position 1 (phase 1), the right-hand side of the inductor is connected to the ground, resulting in the network shown in Fig. 4.2(b). With the switch in position 2 (phase 2), the inductor is connected to the output, leading to the circuit shown in Fig. 4.2(c).

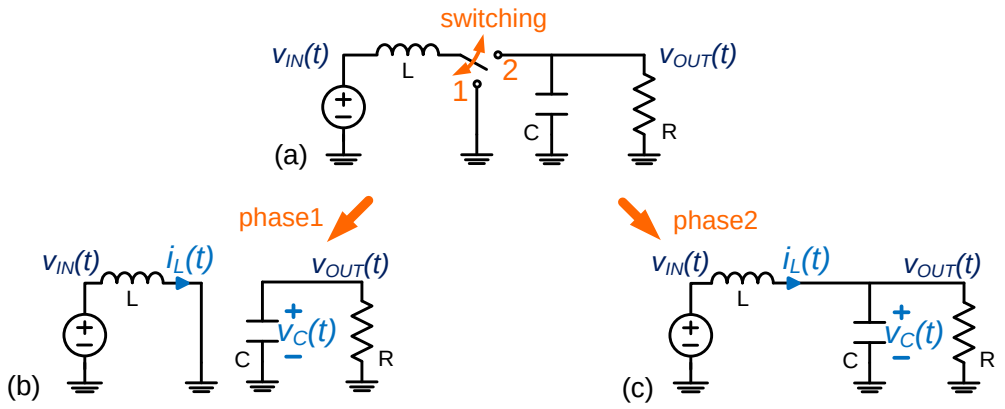


Fig. 4.2 (a) A boost converter circuit and its linear system model in (b) switching phase 1 and (c) switching phase 2.

4.1.1 System Model

Even though the input/output relationship changes at each switching instant, the circuit can be modeled as a linear time-invariant system within each operation phase. For example, the relationship between the input $v_{IN}(t)$ and output $v_{OUT}(t)$ of the boost converter in phases 1 and 2 can be modeled as a set of differential equations listed below:

$$\begin{cases} \dfrac{dv_C(t)}{dt} = -\dfrac{1}{RC} v_C(t) \\ \dfrac{di_L(t)}{dt} = \dfrac{1}{L} v_{IN}(t) \end{cases} \quad \textbf{(4-1)}$$

$$\begin{cases} \dfrac{dv_C(t)}{dt} = \dfrac{1}{C}\dfrac{di_L(t)}{dt} - \dfrac{1}{RC} v_C(t) \\ \dfrac{di_L(t)}{dt} = \dfrac{1}{L} v_{IN}(t) - \dfrac{1}{L} v_C(t) \end{cases} \quad \textbf{(4-2)}$$

This set of differential equations can be converted to Laplace s-domain equivalents, using the Laplace transformation formula for a function derivative in Eq. (4-3), resulting in Eqs. (4-4) and (4-5). Note that the initial conditions of the reactive elements v_C and i_L are made explicit in the s-domain equations as explained in Chapter 2.3. In Eqs. (4-4) and (4-5), the capital letters denote s-domain signals while the italic letters denote their initial conditions in the time domain.

$$\mathcal{L}\left\{ f^n(t) \right\} = s^n \mathcal{L}\{ f(t) \} - \sum_{k=1}^{n} s^{k-1} f^{n-k}(0) \quad \textbf{(4-3)}$$

$$\begin{cases} V_C(s) = \dfrac{RC}{sRC+1} v_C(0) \\ I_L(s) = \dfrac{1}{sL} V_{OUT}(s) + \dfrac{1}{s} i_L(0) \end{cases} \quad \textbf{(4-4)}$$

$$\begin{cases} V_C(s) = \dfrac{1}{s^2LC + sL/C + 1} V_{IN}(s) + \dfrac{sLC}{s^2LC + sL/C + 1} v_C(0) + \dfrac{L}{s^2LC + sL/C + 1} i_L(0) \\ I_L(s) = \dfrac{sC + 1/R}{s^2LC + sL/C + 1} V_{IN}(s) - \dfrac{C}{s^2LC + sL/C + 1} v_C(0) + \dfrac{sLC + L/R}{s^2LC + sL/C + 1} i_L(0) \end{cases} \quad \textbf{(4-5)}$$

With s-domain representations, the boost converter can be simulated by following the steps: 1) sample the initial conditions of the reactive elements, 2) choose the transfer functions corresponding to the operating phases, and 3) compute the output by summing the effects from both the input signal and the initial states of the reactive elements. Fig. 4.3 illustrates the proposed event-driven method applied to the boost converter example. For each phase, the three transfer functions define the relationship between the input (V_{IN}), the initial capacitor voltage ($v_C(0)$), the initial inductor current ($i_L(0)$), and the output ($V_{OUT} = V_C$). Every time the circuit switches between phases, the initial capacitor voltage and the inductor current are sampled and their transfer functions are redefined according to Eqs. (4-4) and (4-5). Then, the output voltage induced by each of the input and initial conditions can be evaluated simply by multiplying each one by its corresponding transfer function. The zero transfer gains for V_{IN} and $i_L(0)$ in phase 1 imply that the output is not related to the input and the inductor current, as they are disconnected from the output during this phase. When the circuit switches to the next phase, however, V_{IN} and the initial current flow in the inductor start to increase V_{OUT} again. Finally, the output is updated as a linear combination of these three components.

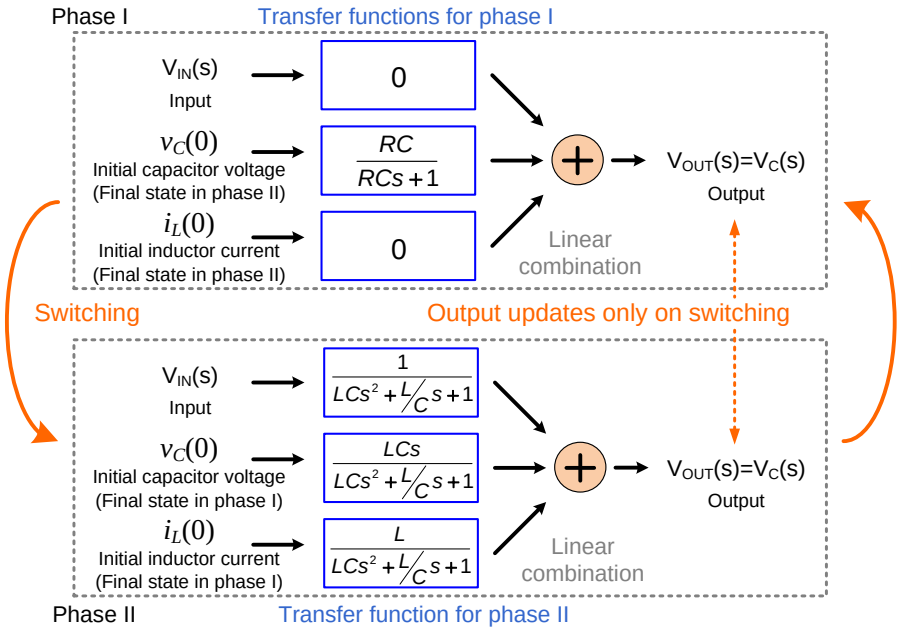


Fig. 4.3 The s-domain event-driven simulation of the boost converter example.

An outline of a boost converter model in SystemVerilog is given in Fig. 4.4. The input and output signals are defined as *xreal*, while the switching input signal is *xbit*. The *always* statement within the module is triggered when the circuit switches between phases (switching events) or the input coefficients change (input events). The initial states of the capacitors and inductors are then sampled and the *param_set* of the output *xreal* signal is updated according to the current input and sampled initial conditions. The *compute_coeff()* function is a DPI function written in C that performs s-domain multiplications of *xreal* signals and transfer functions. As the output update is aligned in time with a switching event or an input event, the *t_offset* of the output has the same value as the one of the switching or the input accordingly.

Once the *param_set* and *t_offset* outputs are updated, the event variable *out.flag* is triggered, thus notifying subsequent blocks of the change event.

```
Module boost_converter(
   input xbit switching,
   input xreal in,
   output xreal out);

   always @(switching.flag or in.flag) begin

      vc0=sample( vc.param_set );    // sampling initial states
      il0=sample( il.param_set );

      if (phase1)                          //switching phase 1
        out.param_set = compute_coeff( In.param_set, tf_in_out_ph1 )
                      + compute_coeff( vc0, tf_vc0_out_ph1 )
                      + compute_coeff( il0,  tf_il0_out_ph1 );
      if (phase2)                          //switching phase 2
        out.param_set = compute_coeff( in.param_set, tf_in_out_ph2 )
                      + compute_coeff( vc0, tf_vc0_out_ph2 )
                      + compute_coeff( il0, tf_il0_out_ph2 );

      if (input_event)       out.t_offset = in.t_offset;
      if (switching_event) oUt.t_offset = switching.t_offset;

      -> out.flag;
   end

endmodule
```

Fig. 4.4 A pseudo-model of the boost converter in SystemVerilog.

4.1.2 Simulation Results

The speed and accuracy of the proposed simulation method are demonstrated using the example of a power factor correction circuit (PFC) composed of a bridge-diode rectifier and a boost converter (Fig. 4.5). The power factor is one of the key performance metrics of AC-DC power converters required by many regulatory standards. It is defined as in Eq. (4-6), which expresses the ratio of the real power flowing to the load and the apparent power in the circuit:

$$power\ factor = \frac{average\ power}{(rms\ voltage)(rms\ current)}. \qquad \textbf{(4-6)}$$

For a high power factor, the circuit should basically behave as a pure resistive load. A boost converter is a widely used topology for power factor correction circuits because the switched inductor at the input conducts a current that is proportional to the input voltage with very low harmonics [33].

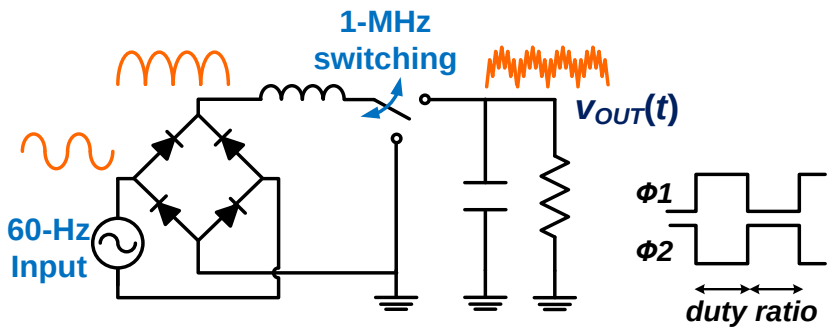


Fig. 4.5 A power factor correction boost converter.

One difficulty in simulating such an AC-DC power converter is that there is a big gap between the input AC frequency and the switching frequency of the boost converter. For instance, in most applications, the input source is 50~60-Hz 110~220V AC power, while the switching frequency is typically in the 100kHz~1MHz range. Therefore, the required simulation time is long, typically several tens of milliseconds, to simulate a few cycles of the 60-Hz AC input.

Fig. 4.6 illustrates the accuracy of the waveforms simulated by the proposed event-driven simulation method. Fig. 4.6(a) is the simulated output voltage of the boost converter, $v_{OUT}(t)$, for one 60-Hz input cycle. The zoom-in waveforms in Fig. 4.6(b) and Fig. 4.6(c), simulated with the proposed method and HSPICE, respectively, demonstrate that they are well matched, and illustrate the switching ripples of the converter. It is noteworthy that HSPICE requires many data points to express the switching ripples (marked by the blue dots in Fig. 4.6(c)) while our event-driven method generates only two events per switching cycle, as indicated by the arrows in Fig. 4.6(b). The power factor can be measured from the simulated input current waveform. The comparison between the simulated power factors as a function of switching frequency and duty cycle in confirms that the proposed method achieves the same level of accuracy as SPICE.

The proposed event-driven simulator demonstrates significant improvements in speed compared with HSPICE, yet retains the equivalent accuracy. On a Linux machine with an AMD Phenom II X4 945 processor, the total execution time to simulate a 0.1-second period with a 100-ns time step is 8.2 seconds. Under the same conditions, the HSPICE simulation takes 920.5 seconds, which is 110× slower.

The execution time of the proposed method varies weakly with the time step (Fig.

4.8(a)). For instance, when the time step is reduced from 100-ns to 10-ps (1/10,000x reduction), the execution time of the proposed method increases by only 15% (from 8.2 to 9.4 seconds), while that of HSPICE increases by 15000%. The reason the execution time hardly varies is that the number of switching events within the 0.1-second period remains the same regardless of the time step, which confirms that the proposed simulation indeed operates in a purely event-driven fashion. This remarkable speed-up does not incur any penalty in accuracy, as the power factor measured using the proposed method is virtually constant, independent of the time step (Fig. 4.8(b)).

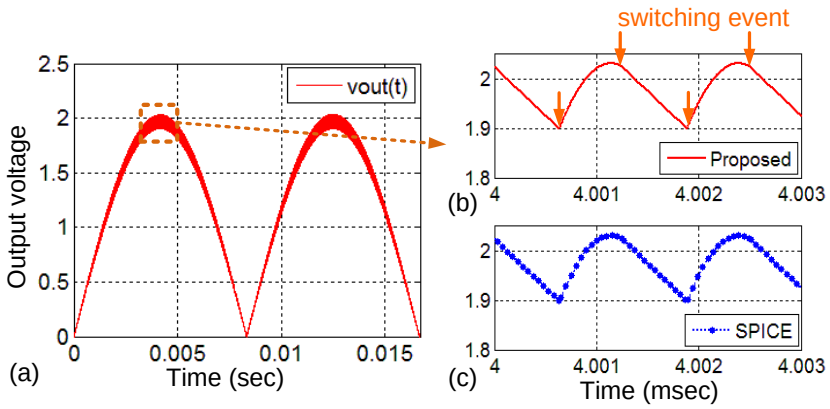


Fig. 4.6 (a) The output voltage waveform $v_{OUT}(t)$ simulated for one 60-Hz input cycle, (b) 5000× zoom-in view of $v_{OUT}(t)$, (c) $v_{OUT}(t)$ simulated by HSPICE.

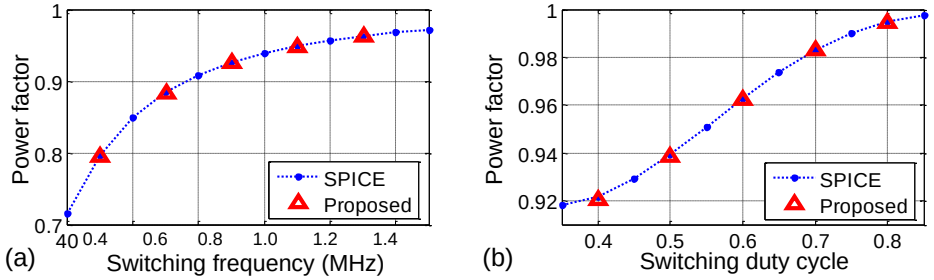


Fig. 4.7 The comparison of the simulated power factors vs. (a) frequency and (b) switching duty cycle.

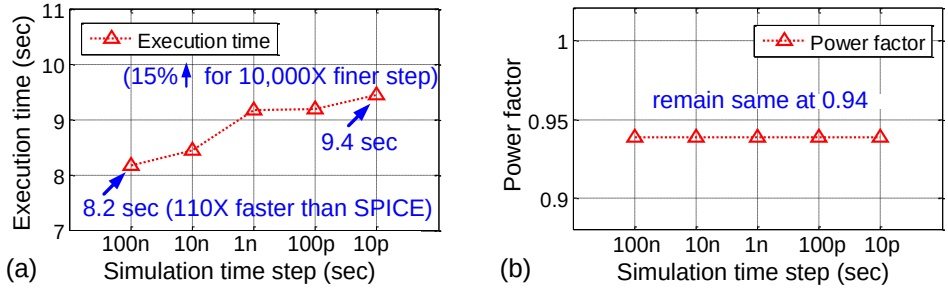


Fig. 4.8 (a) Execution time and (b) simulated power factor for different time steps.

4.2 Time-Interleaved Switched-Capacitor Converter

The second example of a switching power supply is a time-interleaved switched-capacitor (TI-SC) DC-DC converter described and analyzed in [34]. The switched-capacitor DC-DC converter topology is becoming a common choice for power-supplies on chips, as the IC technology is more amenable to integrating high-density capacitors than low-loss inductors. One difficulty in simulating a TI-SC DC-DC converter is that the number of switching activities increases with the number of time-interleaving phases.

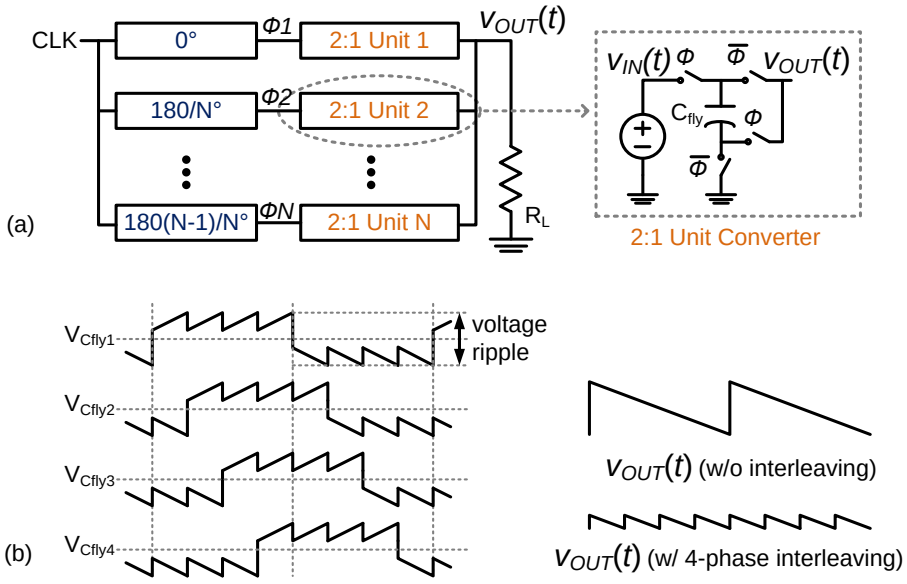


Fig. 4.9 (a) N time-interleaved 2:1 step-down switched-capacitor DC-DC converter, (b) the waveforms of its internal capacitor voltages and final output voltage when N=4 [34].

The TI-SC converter example is composed of N interleaved 2:1 step-down converter units, as shown in Fig. 4.9(a). The total capacitance is divided into a set of small units and the switching is controlled by a set of N equally spaced clocks (Φ1, …, ΦN). Fig. 4.9(b) illustrates the basic operation of a 4-phase TI-SC converter with the output waveforms. The output voltage ripple is inherent in a switched-capacitor converter, and generally decreases as the switching frequency increases. Time-interleaving is an alternate way of reducing the ripples without increasing the switching frequency.

4.2.1 System Model

The TI-SC converter in Fig. 4.9 can be modeled as in Fig. 4.10, where the unit capacitors, C_{fly}, switch their configurations between a series and parallel connections depending on the controlling clock phase (Φ). The model includes the on-resistance of the switches, R_{sw}, and parasitic top- and bottom-plate capacitances, C_{par}, to account for the conduction and switching losses, respectively. To simplify the model, the top- and bottom-plate capacitances are combined into a single capacitor because they experience approximately the same voltage swings in steady states [34]. The s-domain transfer function of each phase can be derived from Eq. (4-7), where $v_{cap}[i]$ denotes the voltage across the *i*-th unit capacitor.

$$V_{OUT}(s) = \frac{\frac{s}{2R_{sw}C_{par}}V_{IN}(s) + \frac{N_S}{4R_{sw}{}^2N_SC}v_{OUT}(0) + \frac{s}{2R_{sw}C_{par}N_S}v_{cap}(0)}{\left\{s^2 + s\left(\frac{R_LN_SC_{par} + 2R_{sw}C + NCR_L}{2R_{sw}CR_LN_SC_{par}}\right) + \frac{1}{2R_{sw}CR_LN_SC_{par}}\right\}}$$

$$\text{where } v_{cap}(0) = \sum_{parallel} v_{cap}[i](0) - \sum_{series} v_{cap}[i](0)$$

$$V_{cap}[i](s) = \begin{cases} \text{for series capacitor:} \\ \frac{1}{2R_{sw}Cs+1}V_{IN}(s) + \frac{1}{2R_{sw}Cs+1}V_{OUT}(s) + \frac{2R_{sw}C}{2R_{sw}Cs+1}v_{cap}[i](0) \\ \text{for parallel capcitor:} \\ -\frac{1}{2R_{sw}Cs+1}V_{OUT}(s) + \frac{2R_{sw}C}{2R_{sw}Cs+1}v_{cap}[i](0) \end{cases} \quad \textbf{(4-7)}$$

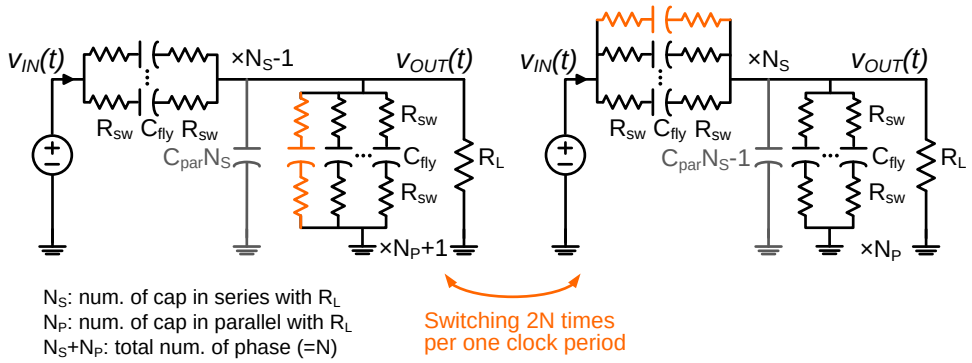


Fig. 4.10 Switched linear circuit model of an N time-interleaved, 2:1 step-down TI-SC converter.

4.2.2 Simulation Results

Fig. 4.11(a) plots the power efficiency of the TI-SC converter as a function of the number of time-interleaving phases, N, and compares the results produced by HSPICE and the proposed method. When the total amount of charge delivered to the load is fixed with N, the output voltage ripple initially decreases by a factor of N, as the amount of charge delivered per clock transition is smaller. Therefore, the better power efficiency can be achieved with a higher N. However, increasing N above a certain value produces diminishing returns because the other losses, such as the conduction loss of the switches and the switching loss of the parasitic capacitors, increase. Fig. 4.11 illustrates this tendency, and the simulation results of the proposed method match well with the SPICE simulation results.

The improvement in speed with the proposed method is moderate compared with the boost converter case, as SPICE is better at simulating switched capacitors than inductors. When simulated on a Linux machine with an AMD Phenom II X4 945 processor, the proposed method shows a ~20X overall speed improvement compared with the HSPICE simulation (see Fig. 4.11(b)).

The proposed method predicts the well-known dependencies of the switching frequency and power efficiency for the output voltage v_{OUT}, described in [34]. Fig. 4.12(a) plots the switching frequency vs. the average v_{OUT} and Fig. 4.12(b) plots the power efficiency vs. the average v_{OUT} when v_{IN} is 2V and N=16. The v_{OUT} dependency on the switching frequency is similar to the IR-drop phenomenon in linear regulators, in which the output voltage drops when the load current is higher than the current that the TI-SC converter can nominally supply. As a result, slower

switching leads to a lower average v_{OUT} and lower power efficiencies (Fig. 4.12). Nonetheless, an excessively high switching frequency is also undesirable as the power efficiency can be degraded due to the loss in the switching capacitors.

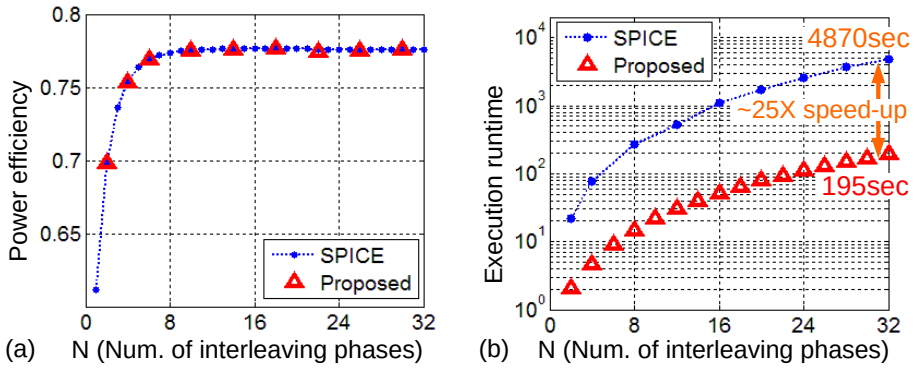


Fig. 4.11 (a) The simulated power efficiency of the TI-SC converter and (b) execution time vs. the number of time-interleaving phases (N).

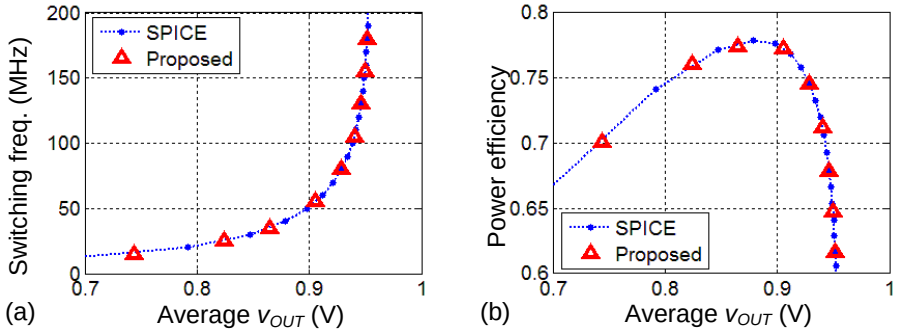


Fig. 4.12 The simulated (a) switching frequency and (b) power efficiency of the TI-SC converter vs. the average output voltage.

Chapter 5

Volterra Series Model Simulation

Even though most analog systems can be modeled as linear or switched-linear systems, there are systems in which weak nonlinearities significantly affect system performances and need to be accurately evaluated. For instance, the nonlinearity of a power amplifier in a RF transmitter introduces cross- and inter-modulation [35] and a continuous-time linear equalizer in a high-speed digital communication system can distort a signal presented to the subsequent data decision blocks [36]. Although such nonlinear responses are orders of magnitude smaller than the desired response, it is important to consider these nonlinearities during system-level verification processes to meet stringent design specifications such as total harmonic distortion of less than –50dBC for a RF system and a bit error rate (BER) of less than 10^{-20} for a high-speed I/O interface.

This chapter demonstrates that the proposed event-driven simulation method can

be extended to simulate such weakly nonlinear behavior of analog circuits. Specifically, this chapter uses the perturbational form of a Volterra series model to simulate the circuit nonlinearities. The Volterra series is one of the most widely used nonlinear system representation. The Volterra series expresses a nonlinear response by a series of polynomial integral operators with increasing degree of nonlinear distortions [20]. Its perturbational form decomposes the nonlinear system equation into a multiple sub-system equation linearized with respect to each nonlinear distortion [35]. Therefore, sub-system equation can be converted to s-domain transfer function and simulated by the presented event-driven method.

In this chapter, two examples are modeled and simulated: a class-A power amplifier for an RF transmitter and a continuous-time linear equalizer (CTLE) for a multi-PAM receiver. The simulation results demonstrate the accuracy and speed of the proposed method compared to SPICE simulation.

5.1 Volterra Series Model

A Volterra series is one of the most widely used nonlinear system representation. The Volterra series expresses a nonlinear system response $y(t)$ to an input $x(t)$ as a sum of partial responses $y_i(t)$:

$$y(t) = y_0 + \sum_{i=1}^{N} y_i(t) = y_0 + y_1 + y_2 + \cdots + y_N \,, \tag{5-1}$$

where $y_i(t)$ is computed as i-times repeated convolution with the i-th order Volterra kernel $h_i(\cdot)$ [20]:

$$y_i(t) = \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} h_i(t_1, t_2, \cdots, t_i) x(t - t_1) x(t - t_2) \cdots x(t - t_i) dt_1 dt_2 \cdots dt_i \,. \tag{5-2}$$

For a weakly nonlinear system, which exhibits only minor deviations from the linear response such as inter- and cross-modulation and gain compression, the output response can be fully described only with the first few orders.

The Volterra formulation provides a way to extract nonlinear transfer functions from the general time-invariant nonlinear differential equations. Without loss of generality, nonlinear circuit behaviors can be formulated in the following nonlinear differential equation:

$$\frac{d}{dt} q\big(y(t)\big) + g\big(y(t)\big) = u\big(x(t)\big) \,. \tag{5-3}$$

where $y(t)$ and $x(t)$ are the output and input, and where $q(\cdot)$, $g(\cdot)$, and $u(\cdot)$ are nonlinear resistive, dynamic and input functions. Considering a small perturbation around a DC operating point at $x = x_0$, $y = y_0$ and expanding $q(\cdot)$ and $g(\cdot)$ at the operating point, Eq. (5-3) becomes:

$$\frac{d}{dt}\left(C_1 y(t) + C_2 y^2(t) + \cdots\right) + \left(G_1 y(t) + G_2 y^2(t) + \cdots\right) = B_1 x(t) + B_2 x^2(t) + \cdots . \quad \textbf{(5-4)}$$

where $C_1 = \left.\frac{\partial^i q}{\partial y^i}\right|_{y=y_0}$, $G_1 = \left.\frac{\partial^i g}{\partial y^i}\right|_{y=y_0}$, and $B_1 = \left.\frac{\partial^i u}{\partial x^i}\right|_{x=x_0}$. Assume that the small perturbation to the input is given as $x(t) = x_0 + \varepsilon \Delta x(t)$, where ε is an arbitrarily small scalar value. According to Eq. (5-1) and (5-2), the output response $y(t)$ should take the form of:

$$y(t) = y_0 + \varepsilon y_1(t) + \varepsilon^2 y_2(t) + \cdots + \varepsilon^n y_n(t). \quad \textbf{(5-5)}$$

As Eq. (5-5) should satisfy the system equation, Eq. (5-4), for an arbitrary value of ε, a set of n differential equations is obtained by equating each of the ε^i-coefficients to zero [37]. For instance, listing only the first three equations:

$$\begin{aligned}
&\frac{\partial}{\partial t}\left(C_1 y_1\right) + G_1 y_1 = Bx \\
&\frac{\partial}{\partial t}\left(C_1 y_2\right) + G_1 y_2 = -\frac{\partial}{\partial t}\left(C_2 {y_1}^2\right) - G_2 {y_1}^2 + B_2 x^2 \\
&\frac{\partial}{\partial t}\left(C_1 y_3\right) + G_1 y_3 = -\frac{\partial}{\partial t}\left(C_3 {y_1}^3 + 2C_2 y_1 y_2\right) - G_3 {y_1}^3 - 2G_2 y_1 y_2 + B_3 x^3
\end{aligned} \quad \textbf{(5-6)}$$

Note that Eqs. (5-6) are linearized with respect to each distortion orders; the first equation is linear with respect to the first-order distortion $y_1(t)$, the second equation is linear with respect to the second-order distortion $y_2(t)$, and so on. Therefore, these equations can be transformed to Laplace s-domain, representing s-domain transfer functions.

For instance, a simple circuit of a nonlinear capacitor in series with a linear resistor demonstrates the procedure of decomposing a nonlinear single-input-single-output (SISO) system equation. Its circuit equation is given as Eq. (5-8), when a

nonlinear capacitance is modeled with a Taylor series, Eq. (5-7).

$$C = C_0 + C_1 v_C(t) \tag{5-7}$$

$$\frac{d}{dt}\left(RC_0 v_C(t) + \frac{RC_1}{2} {v_C}^2(t) \right) + v_C(t) = v_{IN}(t) \tag{5-8}$$

Representing the output response, $v_c(t)$, with a Volterra series, $v_c(t) = v_{c1}(t) + v_{c2}(t) + v_{c3}(t) + \ldots$, the linearized differential equations with respect to the first three order responses are as follows:

$$\begin{aligned}
&RC_0 \frac{dv_{C1}(t)}{dt} + v_{C1}(t) = v_{IN}(t) \\
&RC_0 \frac{dv_{C2}(t)}{dt} + v_{C2}(t) = -\frac{RC_1}{2}\frac{d}{dt}\left[{v_{C1}}^2(t) \right] \\
&RC_0 \frac{dv_{C3}(t)}{dt} + v_{C3}(t) = -RC_1 \frac{d}{dt}\left[v_{C1}(t) v_{C2}(t) \right]
\end{aligned} \tag{5-9}$$

These three linearized differential equations of the nonlinear capacitor example (Eq. (5-9)) can be converted to the s-domain transfer functions as follows:

$$\begin{aligned}
V_{C1}(s) &= \frac{V_{IN}(s) + RC_0 v_{C1}(0)}{sRC_0 + 1} \\
V_{C2}(s) &= \frac{-s\,RC_1 / 2 \cdot \mathcal{L}\left\{ {v_{C1}}^2(t) \right\} + RC_1 / 2 \cdot {v_{C1}}^2(0) + RC_0 \cdot v_{C2}(0)}{sRC_0 + 1} \\
V_{C3}(s) &= \frac{-sRC_1 \cdot \mathcal{L}\left\{ v_{C1}(t) v_{C2}(t) \right\} + RC_1 v_{C1}(0) v_{C2}(0) + RC_0 v_{C3}(0)}{sRC_0 + 1}
\end{aligned} \tag{5-10}$$

For Eq. (5-10), the capital letters denote s-domain signals, the italic letters denote time-domain signals, and terms with *t=0* means its initial state.

Even though Eq. (5-10) includes time-domain multiplications of signals such as ${v_{c1}}^2(t)$ and $v_{c1}(t)v_{c2}(t)$, the multiplication between signals in *xreal* functional form can be efficiently computed in s-domain without additional inverse Laplace transforms.

As explained in Chapter 2.2, one very useful property of our signal representation is that it keeps the Laplace transform of the multiplication between the signals simple, as merely coefficient addition and multiplication are involved. With the two signals $x_1(t)$ and $x_2(t)$ given as Eq. (5-11), a multiplied signal of two signals remains in an identical form, as shown in Eq. (5-12).

$$\begin{aligned} x_1(t) &= \sum_i c_i t^{m_i} e^{-a_i t} \xrightarrow{\mathcal{L}} X_1(s) = \sum_i \frac{c_i \times m_i!}{(s+a_i)^{m_i+1}} \\ x_2(t) &= \sum_j q_j t^{n_j} e^{-p_j t} \xrightarrow{\mathcal{L}} X_2(s) = \sum_j \frac{q_i \times n_i!}{(s+a_i)^{n_i+1}} \end{aligned} \quad \textbf{(5-11)}$$

$$x_1(t)x_2(t) = c_i q_j t^{m_i+n_j} e^{-(a_i+p_j)t} \xrightarrow{\mathcal{L}} A_2[X_1, X_2] = \sum_i \sum_j \frac{c_i q_j \times (m_i+n_j)!}{(s+a_i+p_j)^{m_i+n_j+1}} \quad \textbf{(5-12)}$$

The newly defined s-domain operator, A_2, performs such time-domain multiplication of the two signals via Eq. (5-12). Therefore all the computation can be handled solely in s domain. For instance, Eq. (5-11) can be rewritten as Eq. (5-13) utilizing the operator, A_2.

$$\begin{aligned} V_{C1}(s) &= \frac{V_{IN}(s) + RC_0 v_{C1}(0)}{sRC_0 + 1} \\ V_{C2}(s) &= \frac{-s\,RC_1/2 \cdot A_2[V_{C1}(s), V_{C1}(s)] + RC_1/2 \cdot v_{C1}{}^2(0) + RC_0 v_{C2}(0)}{sRC_0 + 1} \\ V_{C3}(s) &= \frac{-sRC_1 \cdot A_2[V_{C1}(s), V_{C2}(s)] + RC_1 v_{C1}(0) v_{C2}(0) + RC_0 v_{C3}(0)}{sRC_0 + 1} \end{aligned} \quad \textbf{(5-13)}$$

As all the distortion terms, V_{C1}, V_{C2}, and V_{C3}, are expressed in s-domain and they are in the *xreal* form, the Eq. (5-13) can be solved by the proposed event-driven method. First, at every input event, V_{C1}, V_{C2}, and V_{C3} are solved sequentially; V_{C1} is computed based on $V_{IN}(s)$, V_{C2} is solved based on computed V_{C1}, V_{C3} is updated based

on V_{C1} and V_{C2}, and so on. Then, the final output is simply a sum of the distortion terms, $V_C = V_{C1} + V_{C2} + V_{C3}$.

Fig. 5.1 compares the output waveforms simulated with the proposed method (the red solid lines) to a SPICE simulation (the blue dotted line) when a sinusoidal input signal is applied. For the SPICE simulation, the nonlinear differential equation Eq. (5-8) is written in Verilog-A. Fig. 5.1(a), Fig. 5.1(b), and Fig. 5.1(c) show the output waveforms obtained using our method when the first-, second- and third-order terms are added, respectively. Fig. 5.1(d), Fig. 5.1(e), and Fig. 5.1(f) show the error terms compared to the SPICE simulation. With the first three orders, the nonlinear responses are described with an error less than 1mV. It is noteworthy that while SPICE numerically solved the equation at every data point, as represented with dots on the waveforms, the proposed method processed the output response only once, at t=0.

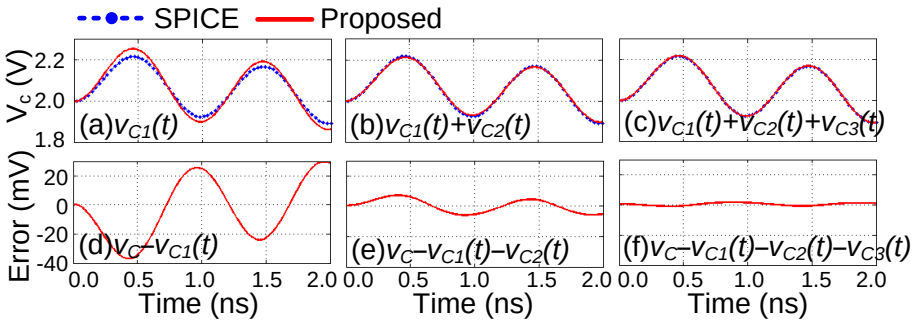


Fig. 5.1 Output response up to (a) the first-order, (b) the second-order, and (c) the third-order responses, and output error with (d) the first-order, (e) the second-order, and (f) the third-order responses.

5.2 Class-A Power Amplifier

The first example of the Volterra series model is an RF transmitter employing a phase-shift keying (PSK) modulation and a class-A power amplifier (Fig. 5.2). The nonlinearity of the power amplifier is a critical factor, as it presents interferers to other users or corrupts its own signal through cross- and inter-modulation. Meeting the stringent design specifications on linearity for RF communication systems, its accurate simulation and verification is important. As the modulation scheme varies with systems, a full-waveform transient simulation is necessary for a rigorous characterization of the PA nonlinearity effect with respect to the scheme. However, a circuit-level simulation with actual modulated inputs takes a very long time, as it requires a fine time resolution for a fast carrier frequency (several GHz) and a long simulation time for a low data rate (several MHz). This chapter demonstrates that the presented event-driven method can accurately simulate a Volterra series model of a class-A power amplifier with a significant speed-up compared to Spectre simulation.

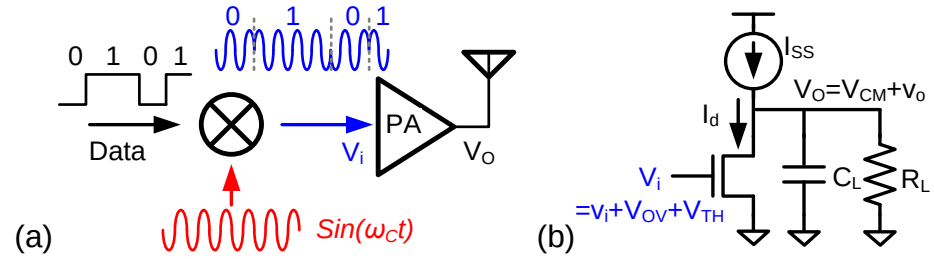


Fig. 5.2 (a) An RF transmitter employing a phase-shift keying modulation scheme and (b) a class-A power amplifier.

5.2.1 System Model

The class-A power amplifier for the exemplary RF transmitter in Fig. 5.2 is modeled in a perturbational form of a Volterra series. There are many effective model extraction and model-order reduction methods for a general Volterra series applied to RF power amplifiers [35]. For demonstrative purposes, the class-A power amplifier is modeled with the nonlinear circuit equation:

$$C_L \frac{dv_o(t)}{dt} + \frac{V_{CM} + v_o(t)}{R_L} + K\left(V_{OV} + v_i(t)\right)^2 \left(1 + \lambda V_o(t)\right) = I_{SS}\,. \qquad \textbf{(5-14)}$$

where the circuit parameters R_L, C_L, K, V_{OV}, and λ are assumed to be 50Ω, 100fF, 1.0A/V^2, 0.1V, and 1/50, respectively. This equation models the distortion due to the square-law dependence and channel-length modulation effect of the MOSFET device. Applying the perturbation method as described in Chapter 5.1, its first-, second-, and third-order transfer functions are given as:

$$\begin{aligned}
V_{o1}(s) &= \frac{-g_m R_L\left(1+\lambda V_{CM}\right)}{sR_LC_L + \lambda KR_L{V_{OV}}^2} V_i(s) + \frac{R_LC_L}{sR_LC_L + \lambda KR_L{V_{OV}}^2} v_{o1}(0) \\
V_{o2}(s) &= -\frac{\lambda g_m R_L}{sR_LC_L + \lambda KR_L{V_{OV}}^2} A_2\left[v_i(t), v_{o1}(t)\right] \\
&\quad -\frac{K\left(1+\lambda V_{CM}\right)R_L}{sR_LC_L + \lambda KR_L{V_{OV}}^2} A_2\left[v_i(t), v_i(t)\right] + \frac{R_LC_L}{sR_LC_L + \lambda KR_L{V_{OV}}^2} v_{o2}(0) \qquad \textbf{(5-15)} \\
V_{o3}(s) &= -\frac{\lambda g_m R_L}{sR_LC_L + \lambda KR_L{V_{OV}}^2} A_2\left[v_i(t), v_{o2}(t)\right] \\
&\quad -\frac{\lambda KR_L}{sR_LC_L + \lambda KR_L{V_{OV}}^2} A_2\left[{v_i}^2(t), v_{o1}(t)\right] + \frac{R_LC_L}{sR_LC_L + \lambda KR_L{V_{OV}}^2} v_{o3}(0)
\end{aligned}$$

5.2.2 Simulation Results

Time-domain simulation is performed to assess a third-order intercept point (IP3) and spectral re-growth property of the transmitter. First, a two-tone test shown in Fig. 5.3(a) is simulated to estimate the third-order intercept point (IP3). Two sinusoidal inputs with frequencies of 0.99GHz and 1.01GHz are applied and the intermodulation power at 1.03GHz is measured. Fig. 5.3(b) shows the simulated main signal power, the third-order inter-modulation power (IM3), and its extrapolated third-order intercept point (IP3), which are in good agreement with the SpectreRF periodic-steady-state (PSS) simulation. Waveforms in Fig. 5.4 depicts a transient output when a two-tone sinusoidal input is applied. For a better visibility, the sinusoidal signals at frequencies with bigger differences (0.9GHz and 1.1GHz) are applied. As shown in the zoom-in view in Fig. 5.4(b), output signal gets distorted as its amplitude is amplified, and this nonlinear effect is well modeled with the third-order Volterra series.

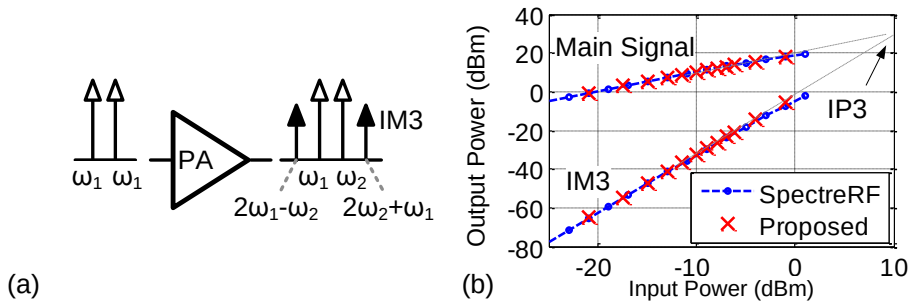


Fig. 5.3 (a) A two-tone testbench for an RF power amplifier and (b) simulated third-order intercept point (IP3).

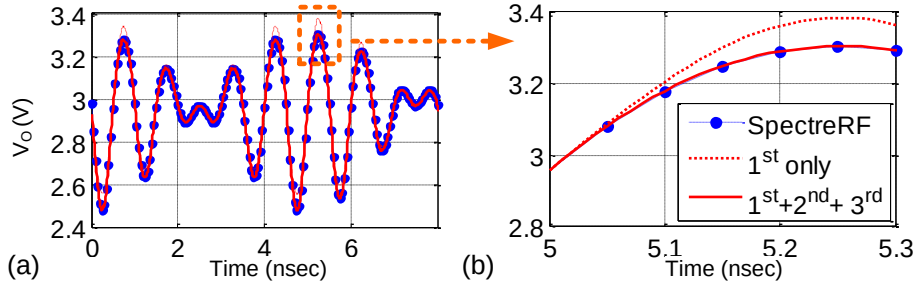


Fig. 5.4 (a) Transient waveforms of the two tone test and (b) their zoom-in view.

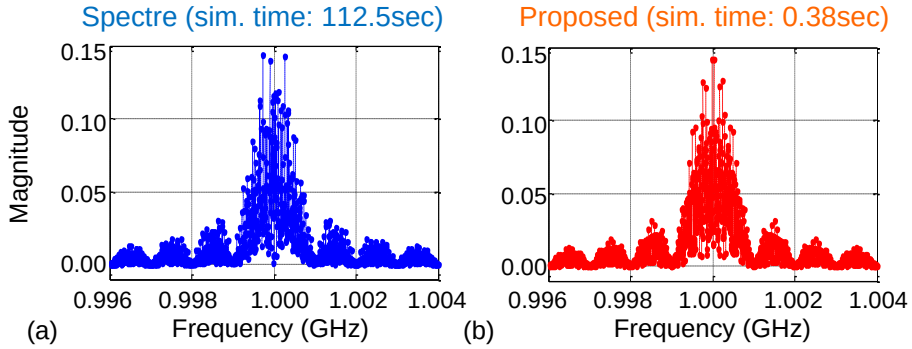


Fig. 5.5 Output spectra of the power amplifier with data-modulated inputs simulated by (a) Spectre and (b) the proposed method.

On the other hand, Fig. 5.5 shows the spectra of the output signal when a 1-GHz carrier signal is PSK-modulated with 1-Mbps data. To obtain the spectra, a transient waveform is simulated for a 512-μsec, which is equivalent to a data length of 512 with 512,000 carrier cycles. The obtained power spectra is in a good agreement with the Cadence Spectre. Thanks to the event-driven simulation, the substantial speed-up of ~300× was observed compared to Cadence Spectre; For a 512-bit data transmission (corresponding to 512μsec), it took only 0.38 seconds for our simulator

while it took 112.5 seconds for Spectre. The speed-up is largely due to the fact that the *xreal* functional form is particularly efficient in expressing a stiff signal such as a PSK modulated signal. For instance, it takes only 512 events for the proposed method to express the signal without any accuracy loss, while it takes one million samples for Spectre to Nyquist-sample the 1-GHz carrier signal.

5.3 Continuous-Time Equalizer

The second example is a multi-level, high-speed data receiver, employing a continuous-time linear equalizer (CTLE) stage and a four-level pulse-amplitude modulation (4PAM) scheme as shown in Fig. 5.6 [36]. The multi-level receiver is particularly sensitive to the distortion, as it experiences non-uniform filtering/amplification depending on the signal level. Resorting to a SPICE-like simulator to assess its effects is however too costly, considering the complexity of a typical high-speed link system and long simulation times required to estimate a BER less than 10^{-12}.

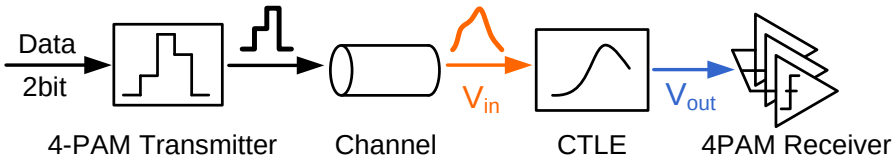


Fig. 5.6 A 4-PAM high-speed I/O interface example.

5.3.1 System Model

The CTLE in Fig. 5.7 is designed to compensate an 8-dB loss at 2GHz with a zero introduced by a source-degeneration capacitor (C_S) and resistor (R_S). The governing circuit equations with input (v_{in}) and transistor's source node voltages (v_{s+} and v_{s-}) are:

$$\begin{aligned} 2I_{SS} &= K\left[\left(V_{OV}+v_{in}-v_{s+}\right)^2+\left(V_{OV}-v_{in}-v_{s-}\right)^2\right] \\ C_s\frac{dv_{s+}-v_{s-}}{dt}+\frac{v_{s+}-v_{s-}}{R} &= g_m\left(v_{in}-v_{s+}\right)+K\left[\left(v_{in}-v_{s+}\right)^2\right] \end{aligned} \quad \textbf{(5-16)}$$

where the circuit parameters g_m, K, C_S, R_S, C_L and R_L have values of 10mS, 0.1A/V^2, 800fF, 400Ω, 100fF and 200Ω, respectively. This equation models the distortion due to the square-law dependence of the MOSFET device. The upper-case letters denote large-signal operating points, while the lower-case letters represent small-signal behaviors of interest.

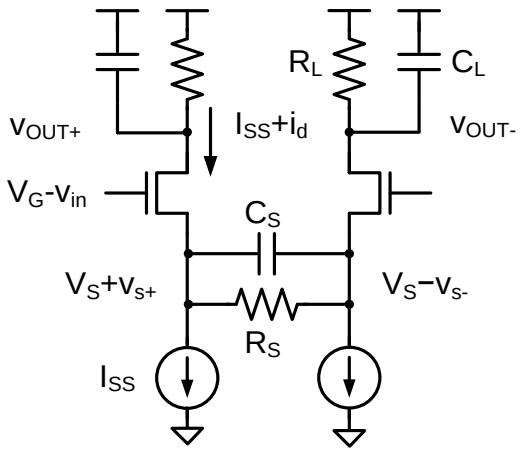


Fig. 5.7 Circuit schematics of the continuous-time linear equalizer.

Applying the perturbation method described in Chapter 5.1, its first-, second-, and third-order transfer functions of v_{s+} are given as:

$$\begin{aligned} V_{s1+}(s) &= \frac{g_m/2C_s \cdot V_{in}(s) + v_{s1+}(0)}{s + (1/R_sC_s + g_m/2C_s)} \\ V_{s2+}(s) &= 2K/g_m \cdot (V_{in}(s) - V_{s1+})^2 \\ V_{s3+}(s) &= \frac{K/C_s \cdot A_2[V_{s2+}(s), V_{s1+}(s) - V_{in}(s)] + v_{s3+}(0)}{s + (1/R_sC_s + g_m/2C_s)} \end{aligned} \quad . \qquad \textbf{(5-17)}$$

Once v_{s+} is solved, then the output v_{out} can be computed by the following differential equation:

$$C_L \frac{dv_{out}}{dt} + \frac{v_{out}}{R} = K(V_{OV} + v_{in} - v_{s+})^2, \qquad \textbf{(5-18)}$$

which has an equivalent s-domain representation:

$$V_{out}(s) = \frac{K \cdot A_2[V_{OV} + V_{in}(s) - V_s(s), V_{OV} + V_{in}(s) - V_s(s)]}{1/R + sC_L} + \frac{C_L \cdot v_{out}(0)}{1/R + sC_L}. \qquad \textbf{(5-19)}$$

5.3.2 Simulation Results

Fig. 5.8 compares the eye-diagrams of the 2-Gbps 4-PAM signals before and after the CTLE stage for the transmit swing of $\pm 30mV_{dpp}$ and $\pm 300mV_{dpp}$. Since the channel is modeled as a linear system with 7 poles, there is no difference in its output eye shapes between Fig. 5.8(a) and (d). The same is true when only the first-order response of the CTLE stage is modeled (Fig. 5.8(b) and (e)). However, when the third-order distortion response is included, the output signal exhibits different amount of distortion depending on the signal swing as shown in Fig. 5.8(c) and (f). In particular, for the input swing of $\pm 300mV_{dpp}$, the top-most and bottom-most eye openings are smaller (115mV) than that of the middle eye (155mV).

Fig. 5.9 compares the eye diagrams obtained with Synopsys HSPICE and the proposed simulator with different time steps of 100ps and 100fs. Note that the eye diagram obtained is identical to the one with HSPICE even with a coarse time step of 100ps (0.2 UI). In addition, simulating 2,000 bits of data pattern takes 26 seconds for the proposed method on a Linux machine with AMD Phenom II X4 945 processor while HSPICE simulation takes 25,720 seconds (~990× speed-up).

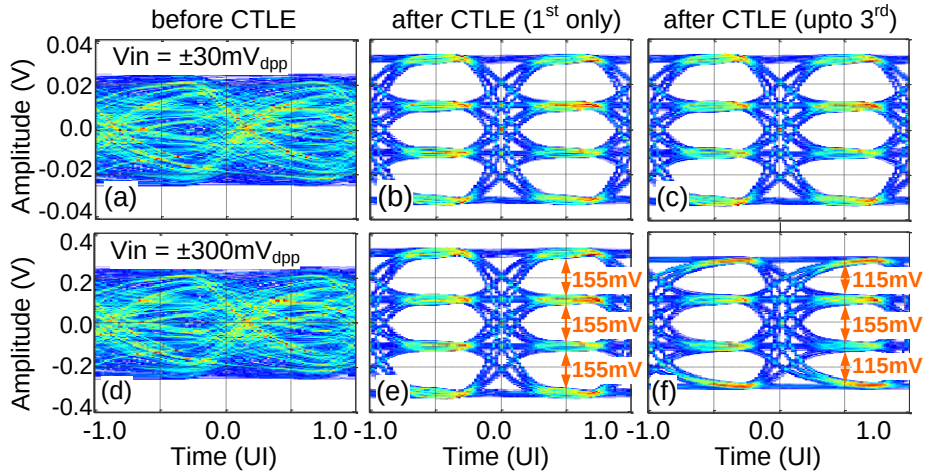


Fig. 5.8 The simulated eye diagrams for two swing levels: ±30mV$_{dpp}$ and ±300mV$_{dpp}$. The eye diagrams before the CTLE (a,d) and after the CTLE without (b,e) and with the third-order distortion included (c,f).

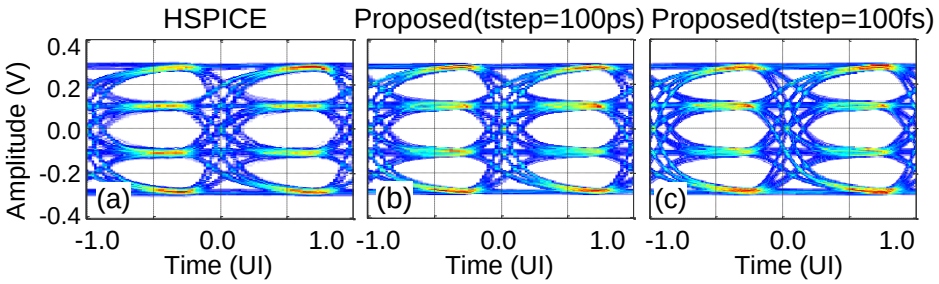


Fig. 5.9 Eye-diagrams of the CTLE output added up to a third-order response simulated by (a) SPICE with a maximum time step of 10ps, (b) the proposed method with a time step of 100ps, and (c) the proposed method with a time step of 100fs.

Chapter 6

Injection-Locked Oscillator Simulation

Injection-locked oscillators (ILOs) are circuits that produce a fundamental or super-/sub-harmonic frequency of the input signal based on an injection locking phenomenon [38]. When the injection input or super-/sub-harmonic frequency is close enough to the oscillator's free-running frequency, the nonlinear interaction within the oscillator forces its frequency and phase to lock to those of the injection input. This injection locking phenomenon is utilized in many wireline and wireless communication applications such as multi-phase clock generation, jitter filtering, frequency multiplication or division, and burst-mode clock recovery [39]-[44]. However, the nonlinear, time-varying oscillator characteristic challenges the modeling and simulation of the injection-locking behavior.

This chapter shows that the presented method can simulate such nonlinear time-varying behaviors in an event-driven fashion. The injection-locking behavior is modeled by a perturbation projection vector (PPV) based approach presented in [45]. The governing ordinary differential equation (ODE) of the PPV-based ILO model is nonlinear and can be expanded by a Volterra series as described in Chapter 5. However, there are two main differences from the nonlinear behavior simulation in Chapter 5: first, the governing ODE does not have an analytical form as the PPV is numerically extracted via simulation in most cases. Second, the governing ODE is periodically time-varying as it describes the oscillator's behavior. To address these issues, the numerically extracted PPV is curve-fitted into a piecewise polynomial (PWP). Then, the PPV-based ILO model can be simulated in the same way as the Volterra series model with additional crossing events at piecewise interval boundaries.

This chapter demonstrates the PPV-based ILO model simulation through three examples: an LC oscillator, a ring oscillator, and a burst-mode CDR. In the LC oscillator example, the dependence of the simulation accuracy and speed on the number of intervals and degree of polynomials in PPV's piecewise polynomial notation is investigated. Additionally, a ring oscillator example with a non-sinusoidal PPV shows that the proposed model can accurately simulate both the lock and quasi-lock behaviors of the ILO and also predict the locking range and static phase offset for different injection frequencies and magnitudes. Furthermore, a burst-mode clock recovery example shows that the proposed method can also simulate the locking transient, lock range, and lock time of the ILO when an aperiodic, pseudo-random pattern signal is injected into it.

6.1 PPV-Based ILO Model

Among various macromodeling approaches for ILOs [45]-[49], this work adopts the perturbation projection vector (PPV) based approach presented in [45] to model the oscillator's response to an external injection input. The simplest and most intuitive model is perhaps the Adler's equation [46]. While it can predict the ILO locking range, it depends on a quality factor Q that limits its use to LC oscillators. The generalized Adler's equation [47] extends the applicability to other oscillator types, but ignores the high-order harmonic effects. Another macromodel approach for ILOs is based on the impulse sensitivity function (ISF) [48]. ISF describes the final phase shift of the oscillator caused by an impulse arriving at different times. While the ISF can capture harmonic effects, it is not suitable for modeling the injection-locking behaviors, since only the final phase shift at steady-states is described and not the instantaneous phase shift during transients. Thus, the ILO model in [49] introduced an additional phase shift term in the ISF-based model. On the other hand, the PPV-based macromodel [45] can predict the oscillator's injection locking behavior most accurately and is generally applicable to all types of oscillators and inputs [50]-[52]. Moreover, the PPV-based model describes the oscillator's phase response to the perturbation using a compact, scalar, nonlinear ordinary differential equation (ODE) that renders its models simple.

The PPV model describes the output response of a perturbed oscillator as:

$$x_p(t) = x_s(t + \alpha(t)) + y(t) \quad \textbf{(6-1)}$$

where $x_s(t)$ is a steady-state response of an unperturbed oscillator. Here, the perturbation is decomposed of the amplitude deviation, $y(t)$, and phase deviation, $\alpha(t)$. The PPV model mainly focuses on the phase deviation as it is the only one that persists over time. In other words, the perturbation input on the oscillators' amplitude is neglected and the perturbed oscillator's signal waveform, $x_p(t)$, is approximated as:

$$x_p(t) \approx x_s(t + \alpha(t)) . \quad \textbf{(6-2)}$$

PPV is a periodically time-varying vector $\vec{v}_1(\cdot)$ that describes the oscillator's phase response to the perturbation input. The PPV-based phase-domain macromodel describes the phase deviation $\alpha(t)$ of the perturbed oscillator using the nonlinear differential equation

$$\dot{\alpha}(t) = \vec{v}_1^T(t + \alpha(t)) \cdot \vec{b}(t) , \quad \textbf{(6-3)}$$

where $\vec{b}(t)$ is an n×1 row vector describing the perturbation inputs to the system and $\vec{v}_1(\cdot)$ is a periodically time-varying n×1 row vector referred to as the perturbation projection vector (PPV). Eq. (6-3) implies that the PPV represents the oscillator's phase sensitivity to the perturbation input depending on its internal phase $t+\alpha(t)$. Due to the periodically time-varying nature of the oscillator, the PPV is also periodic with the self-oscillating period. The ODE in Eq. (6-3) is nonlinear, because the phase variable $\alpha(t)$ is used as an argument to the PPV function.

PPV is a well-established concept in oscillator modeling and various PPV extracting methods from transistor-level circuits are available [45], [53], [54]. For instance, the work in [53] extracts the PPV from the frequency- or time-domain Jacobian matrices computed during steady-state analysis. Some commercial

simulators like Cadence SpectreRF provide options to report the PPVs after the periodic steady-state (PSS) analysis of oscillators.

The main issue in simulating the PPV-based ILO model with the same method as the Volterra series model in Chapter 5 is that the PPV is numerically extracted via simulation and the ODE does not have a polynomial form of Eq. (5-4). One straightforward way is to curve-fit the numerical PPV into a polynomial function. However, a polynomial expansion is not suitable for describing a periodic function like PPV since asymptotically, it is either increasing or decreasing. Instead, a piecewise polynomial (PWP) expansion is adopted to describe the periodic PPV.

The PWP expansion of the PPV turns the periodically time-varying ODE in Eq. (6-3) into a set of nonlinear ODE segments which is more amenable to a Volterra series model. Let's consider an example of a PPV with a single row element, $v_1(t)$, described by a piecewise polynomial function with k phase intervals:

$$v_1(t) = \begin{cases} f_1(t) & \text{for } 0 \le t < t_1 \\ f_2(t) & \text{for } t_1 \le t < t_2 \\ \quad\vdots & \\ f_k(t) & \text{for } t_{k-1} \le t < 1 \end{cases}, \qquad \textbf{(6-4)}$$

where $f_1(\cdot)$, $f_2(\cdot)$, …, $f_k(\cdot)$ are polynomial functions of the input phase argument, t. The PPV continuity and periodicity require

$$\begin{aligned} &f_i(t_i) = f_{i+1}(t_i) \quad \text{for} \quad i = 1, 2, \ldots, k-1 \\ &f_1(0) = f_k(1) \end{aligned} . \qquad \textbf{(6-5)}$$

For instance, Fig. 6.1 shows an exemplary PPV curve-fitted to a third-order piecewise polynomial function with four intervals (k=4). By substituting Eq. (6-4) into Eq. (6-3), the piecewise nonlinear ODE governing the phase deviation response

$\alpha(t)$ of the ILO is given as:

$$\frac{d\alpha(t)}{dt} = \begin{cases} f_1(t+\alpha(t))\cdot b(t) & \text{for } 0 \le t+\alpha(t) < t_1 \\ f_2(t+\alpha(t))\cdot b(t) & \text{for } t_1 \le t+\alpha(t) < t_2 \\ \vdots & \\ f_k(t+\alpha(t))\cdot b(t) & \text{for } t_{k-1} \le t+\alpha(t) < 1 \end{cases} \quad \textbf{(6-6)}$$

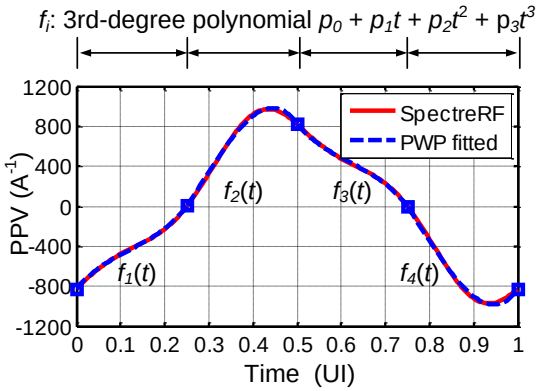


Fig. 6.1 Piecewise polynomial expansion example of a PPV extracted by SpectreRF.

For a given segment of the piecewise nonlinear ODE in Eq. (6-5), the phase response to an input event is computed in the same way as the Volterra series model in Chapter 5. The ODE segments listed in Eq. (6-5) are nonlinear ODEs, since each of their right-hand side expressions contains a polynomial function of the phase deviation, $\alpha(t)$. When the ODE segment response is assumed to be a sum of multiple distortion-order responses (i.e., $\alpha(t)=\alpha_1(t)+\alpha_2(t)+\alpha_3(t)+...$), a perturbation method in [37] decomposes the nonlinear ODE of Eq. (6-5) into a set of linear ODEs, each of which governs a different distortion-order response. If a small perturbation is given as an input, $\varepsilon \cdot b(t)$, with an arbitrarily small scalar value ε, $\alpha(t)$ takes the form of Eq.

(6-7) assuming that the nonlinear distortion terms generate harmonics at their degrees:

$$\alpha(t) = \sum \varepsilon^i \alpha_i(t) = \varepsilon\alpha_1(t) + \varepsilon^2\alpha_2(t) + \varepsilon^3\alpha_3(t) + \cdots \quad \textbf{(6-7)}$$

Since Eq. (6-7) should satisfy the system equation in Eq. (6-5) for an arbitrary ε, a set of differential equations is obtained by equating each ε^i-coefficient to zero. For instance, if the PPV is described using a second-order piecewise polynomial (i.e., $f_i(t)=p_0+p_1t+p_2t^2$) and the phase response, $\alpha(t)$, is approximated up to its third-order distortions, the system equation for a given segment becomes:

$$\frac{d\left(\sum_{i=1}^{3}\varepsilon^i\alpha_i(t)\right)}{dt} = \left\{p_0 + p_1\left(t + \sum_{i=1}^{3}\varepsilon^i\alpha_i(t)\right) + p_2\left(t + \sum_{i=1}^{3}\varepsilon^i\alpha_i(t)\right)^2\right\} \cdot \varepsilon b(t) \quad \textbf{(6-8)}$$

By equating each of the ε^i-coefficients to zero, the set of the decomposed ODEs are

$$\begin{aligned}
\frac{d\alpha_1(t)}{dt} &= \left\{p_0 + p_1t + p_2t^2\right\} \cdot b(t) \\
\frac{d\alpha_2(t)}{dt} &= \left\{p_1\alpha_1(t) + 2p_2t\alpha_1(t)\right\} \cdot b(t) \\
\frac{d\alpha_3(t)}{dt} &= \left\{p_1\alpha_2(t) + 2p_2t\alpha_2(t) + p_2{\alpha_1}^2(t)\right\} \cdot b(t)
\end{aligned} \quad \textbf{(6-9)}$$

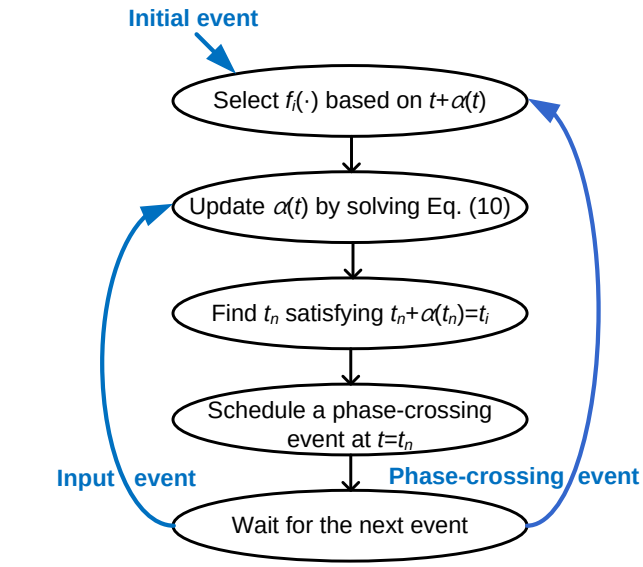
With the input signal $b(t)$ already in a form of Eq. (2-4), right-hand side of Eq. (6-9) also takes the same form as in Eq. (2-4), as they are products between polynomials and exponentials. As $\alpha_i(t)$ corresponds to an integration of each of these expressions, each distortion-order response can be computed in an event-driven fashion by converting the right-hand side expression into Laplace s-domain using Eq. (2-5) and multiplying it by 1/s. The overall response, $\alpha(t)$, of the nonlinear ODE segment is a sum of these multiple distortion-order responses, $\alpha_1(t)$, $\alpha_2(t)$, ... , $\alpha_i(t)$, and its initial condition. The number of distortion-order responses required to accurately describe

the oscillator's behavior depends on how strongly nonlinear the circuit is.

Due to the piecewise representation of the PPV, the phase $t+\alpha(t)$ crossing the piecewise interval boundaries triggers additional phase computations. In other words, every time the phase advances to the new polynomial segment, the phase deviation, $\alpha(t)$, should be re-computed based on a corresponding polynomial in Eq. (6-6). Whenever a new analytical $\alpha(t)$ is computed, the time, t_n, when the phase crosses the current phase interval boundary is found and the next phase computation event is scheduled. Since an ILO typically has a narrow locking range, the crossing time, t_n, can be found via an efficient bisection search algorithm with an narrow initial search range. For instance, it is still conservative to use $(t, t+2/k]$ as the search range where t is the current time in a unit interval (UI) and k is the number of piecewise segments over one period.

Fig. 6.2 summarizes the simulation flow of the aforementioned procedure for computing the phase deviation response, $\alpha(t)$. Assuming that the initial condition for $\alpha(t)$ is given, a proper nonlinear ODE segment that corresponds to the current phase $t+\alpha(t)$ is chosen. Upon the arrival of each input event, the $\alpha(t)$ of the chosen nonlinear ODE segment is computed as a sum of multiple distortion-order responses, each of which can be obtained by solving a linear differential equation in Eq. (6-9). Every time the analytical expression for the phase response $\alpha(t)$ is updated, the time, t_n, when the phase $t+\alpha(t)$ crosses the interval boundary of the current ODE segment is computed and a Verilog event is scheduled at t_n. When the time advances to t_n, the scheduled event is triggered and the phase response, $\alpha(t)$, is re-computed using the next ODE segment while preserving the state variable $\alpha(t)$ (i.e. an initial condition for the next segment). If a new input event arrives before the time reaches t_n, $\alpha(t)$ is

re-evaluated using the current ODE segment and a new t_n value is computed. In this case, the previously-scheduled event at t_n is rescheduled according to this new t_n value.



(a)

Initial
- Trigger a phase-crossing event at t=0

always @(phase-crossing event) begin
- Select $f_i(\cdot)$ based on $t+\alpha(t)$
- Solve the ODE (Eq. (10)) for the new $\alpha(t)$
- Find the next crossing time t_n satisfying $t_n+\alpha(t_n)=t_i$
- Schedule a phase-crossing event at $t=t_n$

end

always @(input event) begin
- Solve the ODE (Eq. (10)) for the new α(t)
- Find the next crossing time t_n satisfying $t_n+\alpha(t_n)=t_i$
- Reschedule a phase-crossing event at $t=t_n$

end

(b)

Fig. 6.2 The procedure for the event-driven simulation of the ILO model using a piecewise polynomial expansion of the PPV: (a) a process flowchart and (b) Verilog pseudo-code.

6.2 LC Oscillator

The first example is an 1-GHz LC oscillator frequently used in many papers on the PPV-based macromodel [51], [55] (Fig. 6.3). The governing differential equations of this LC oscillator are

$$\begin{aligned} -C\frac{d}{dt}v(t) &= \frac{v(t)}{R} + i(t) + S\tanh\left(\frac{G_n v(t)}{S}\right) + b(t) \\ L\frac{d}{dt}i(t) &= v(t) \end{aligned}, \qquad \textbf{(6-10)}$$

where L, R, and C are the inductance, resistance, and capacitance of the LC tank, respectively, and S and G_n are the parameters of the nonlinear negative resistor. The parameter values used for this example are L=4.869x10^{-7}/2π, C=2x10^{-12}/2π, R=100, S=1/R, and G_n= –1.1/R.

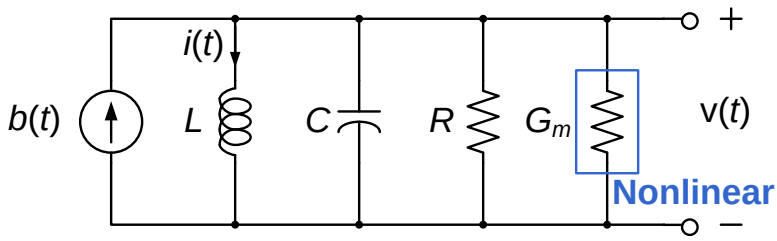


Fig. 6.3 An LC oscillator example [51]

6.2.1 System Model

Fig. 6.4(a) and (b) show the oscillating waveform and PPV of the LC oscillator (Fig. 6.3) at the $v(t)$ node that are both simulated using SpectreRF. The simulated PPV is divided into equally-spaced 16 segments, each curve-fitted to a first-degree polynomial (i.e. piecewise linear) function having a least mean square (LMS) error (Fig. 6.4(b)). While the uniform piecewise interval is demonstrated in this example, the non-uniform piecewise interval can also be used to reduce the PPV-fitting error without increasing the number of piecewise intervals.

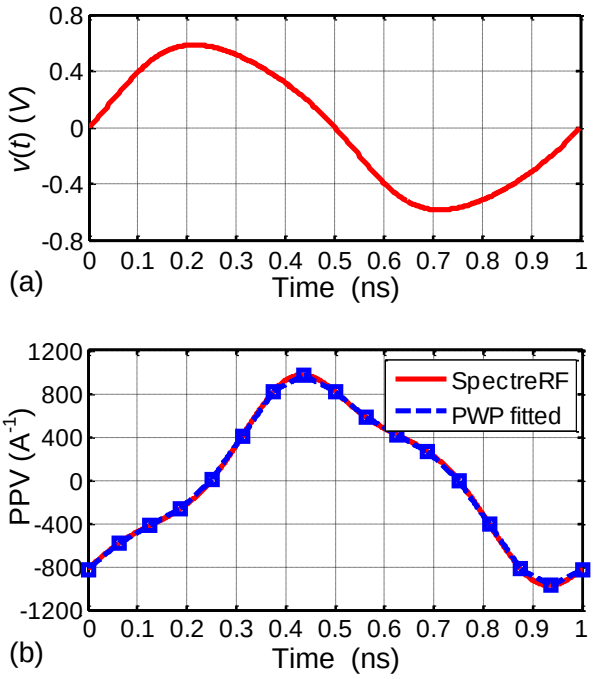


Fig. 6.4 (a) its oscillating output waveform and (b) measured perturbation projection vector (PPV) at node $v(t)$.

6.2.2 Simulation Results

Fig. 6.5 shows that the proposed method accurately simulates the ILO's response to a step change of the input clock phase. First, the oscillator is injection-locked to a sinusoidal signal at a 1GHz frequency. The input signal, $b(t)$, changes its polarity at 10ns (i.e. a phase shift of π), as shown in Fig. 6.5(a). After the input signal's phase change, the oscillator phase starts to decrease following the input signal phase and reaches π at 40ns (Fig. 6.5(b)). The phase change simulated with the proposed method is compared with the Spectre simulation in Fig. 6.5(c). The maximum phase difference between two simulations for [0:50ns] was 0.15-rad.

To examine the accuracy and speed dependencies on the piecewise polynomial (PWP) approximation of the PPV, the maximum phase errors in the aforementioned phase-shift test bench are measured for various numbers of PWP intervals and degrees of polynomials. In this test, the nonlinear ODE is decomposed up to the second-order distortion. Fig. 6.6 shows the measured phase errors between the proposed method and Spectre simulations. The phase error decreases as the number of intervals increases and the degree of polynomials increases (Fig. 6.6(a)). However, the phase error rapidly reaches its minimum as the number of intervals increases. In fact, the case with 32 intervals has the same phase error as the case with 16 intervals. On the other hand, the simulation runtime increases in proportion to the number of intervals and degree of polynomials (Fig. 6.6(b)). This is because the larger number of intervals increases the phase update events and the higher degree of polynomials increases the computation required for each event. With the maximum error tolerance at 0.2-rad, the case with 16 intervals and first-order polynomial has the

fastest simulation time with 9.5sec. For reference, the equivalent simulation in Spectre took 16.2 sec.

To examine the accuracy and speed dependencies on the distortion orders included in the nonlinear ODE decomposition, the worst-case phase error is

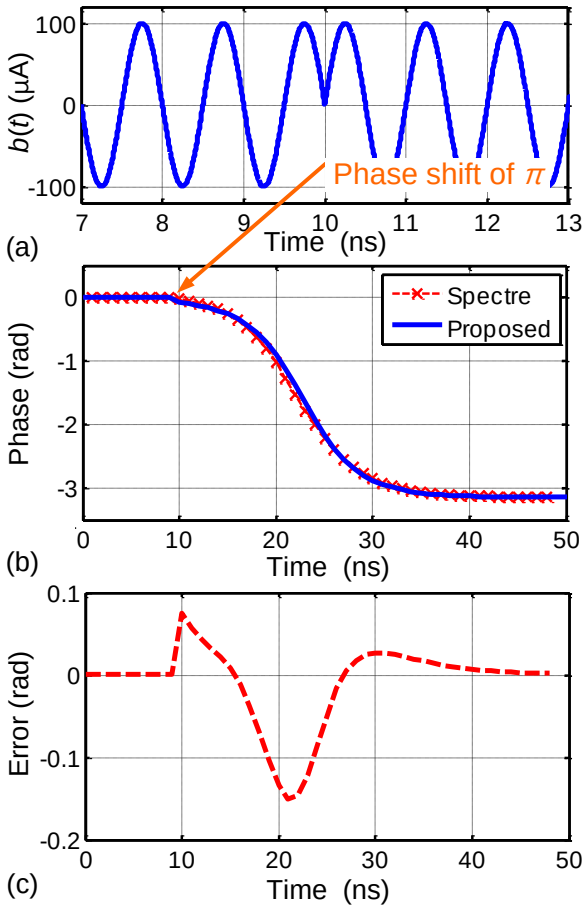


Fig. 6.5 (a) A phase shift of π in the input perturbation $b(t)$, (b) a locking transient of the ILO phase, and (c) a simulation error in ILO's phase response compared to Spectre simulation.

measured while varying the maximum distortion-order included in the phase in Eqs. (6-7)-(6-9). In this case, the PPV is curve-fitted to a piecewise linear function. Fig. 6.6(c) shows that the phase error decreases as the maximum distortion order increases to 2, but has rapidly diminishing returns as the order increases past 2. On the other hand, Fig. 6.6(d) shows that the simulation time grows super-linearly with the maximum distortion order due to the large amount of computation required for the higher distortion-order responses. Considering this trade-off between accuracy and speed, it is best to use the maximum distortion order of 2.

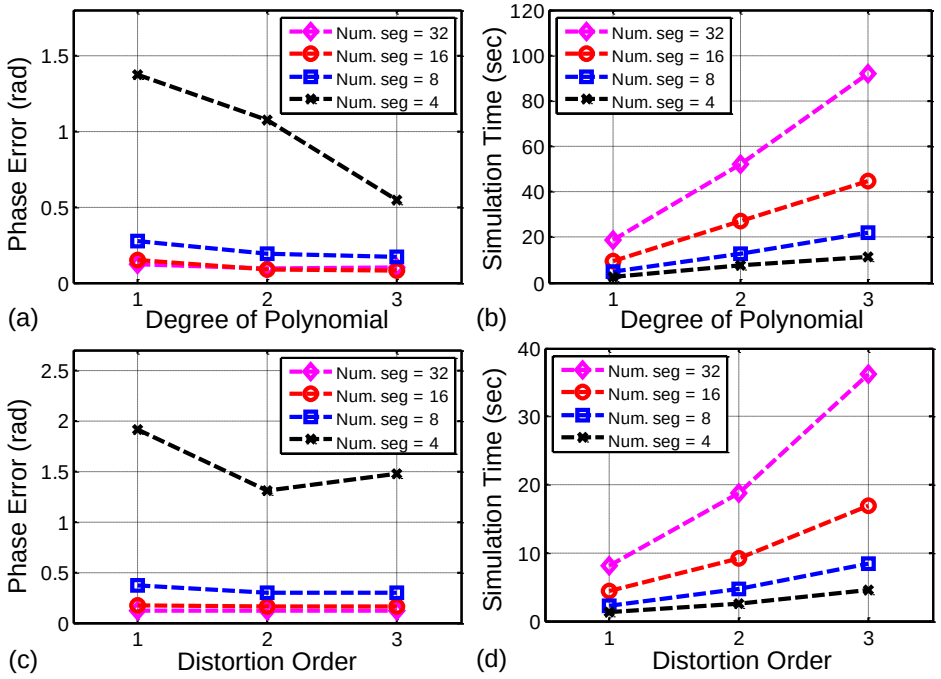


Fig. 6.6 The worst-case phase errors and execution times while varying the degree of piecewise polynomials describing the PPV ((a) and (b), respectively) and maximum distortion order included while solving each piecewise nonlinear ODE ((c) and (d), respectively).

6.3 Ring Oscillator

The second example is a 7.2-GHz, 7-stage ring oscillator in the CMOS process (Fig. 6.7). Injection-locked ring oscillators are frequently used in recent low-power PLL applications, as it can suppress the noise-sensitive ring oscillator jitters by periodically injecting a clean reference signal. However, its limited locking range and static phase offset should be verified for the correct PLL operation. In addition, a ring oscillator has a non-sinusoidal PPV with its peak sensitivities located near the clock edge transition timings. This can demonstrate the applicability of the proposed PPV-based model with the piecewise polynomial expansion.

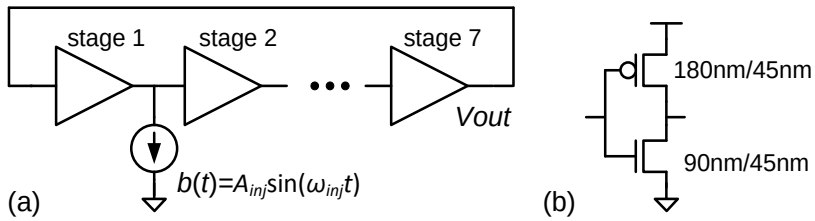


Fig. 6.7 (a) A 7-stage injection-locked ring oscillator and (b) a circuit schematic of each stage.

6.3.1 System Model

For the ring oscillator shown in Fig. 6.7(a), the input injection signal $b(t)=A_{inj}\sin(\omega_{inj}t)$ is connected at the output node of the first stage. Fig. 6.8 shows its oscillating waveform, *Vout*, and measured PPV at the injection node. The PPV is fitted into a piecewise linear function with 16 equally-spaced intervals, as shown in Fig. 6.8(b). The governing nonlinear ODE of the oscillator's phase response is decomposed up to the second-order distortion.

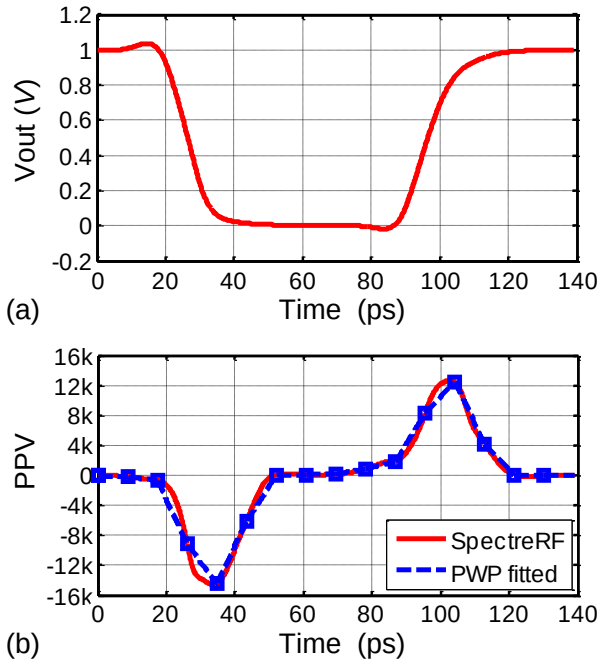


Fig. 6.8 (a) An oscillating waveform of the ring oscillator and (b) a measured perturbation projection vector (PPV) at the injection node.

6.3.2 Simulation Results

Fig. 6.9 shows the simulated injection-locking and pulling behaviors of the ring oscillator in comparison with the Spectre simulation. For the injection-locking case, a 7.27-GHz sinusoidal signal with 4μA amplitude is injected at time 10ns. Under this strong injection, the frequency is locked to the injection frequency and there is no further phase shift after 20ns, as shown in Fig. 6.9(a) and Fig. 6.9(b). On the other hand, Fig. 6.9(c) and Fig. 6.9(d) show the frequency and phase waveforms of the oscillator when a 7.27-GHz sinusoidal signal with 2.5μA amplitude is injected. In this case, the injected signal frequency is just above the lock range and the ring oscillator shows injection-pulling. When the oscillator is injection-pulled, it mostly behaves as if it is locked to the injection (the quasi-lock shown in Fig. 6.9(d)), but periodically undergoes a rapid 2π phase shift returning to the quasi-lock condition (i.e., the phase slip shown in Fig. 6.9(d)) [38]. As shown in Fig. 6.9, the waveforms simulated with the proposed method agree with the Spectre simulation. For the 70-ns simulation period shown in Fig. 6.9, the worst-case phase errors for the injection-locking and injection-pulling cases were 0.074-rad and 0.105-rad, respectively. On the other hand, the execution times of simulating the longer 1μs period were 12.5 seconds for the proposed method and 387.6 seconds for Spectre, which shows a speed-up of 31×. This larger speed-up compared to the previous LC oscillator case is because Spectre simulation is slower for the transistor-level circuits than the RLC passive circuits. In contrast, PPV-based macromodels abstract these circuit-level details and achieve the same simulation speeds that depend only on the polynomial degree and number of piecewise intervals.

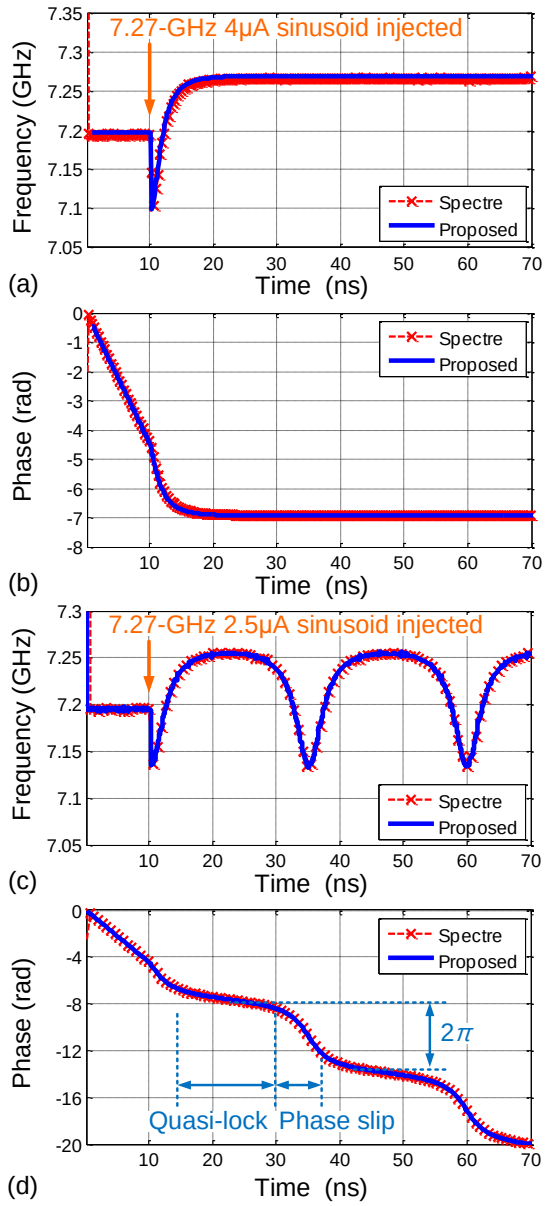


Fig. 6.9 (a) Frequency and (b) phase waveforms when the ring oscillator is injection-locked. (c) Frequency and (d) phase waveforms when the ring oscillator is injection-pulled.

Fig. 6.10 is the minimum injection strength needed to acquire a lock and the simulated static phase offset after lock at various injection frequencies. Fig. 6.10(a) plots the minimum injection amplitude, A_{inj}, required for the oscillator to acquire the lock as a function of injection frequencies normalized to the self-oscillating frequency, ω_0. Fig. 6.10(a) shows that the larger injection amplitude is required as the frequency offset of the injection signal becomes larger. For the same frequency offsets (e.g. ω_{inj}/ω_0=0.96 and 1.04), the ring oscillator can lock to the lower frequency (i.e. ω_{inj}/ω_0=0.96) with the smaller injection amplitude. Fig. 6.10(b) measures the static phase offsets of the output signal compared to the injection signal at different injection frequencies. For injection frequencies higher than ω_0, the output signal lags behind the injection signal. For frequencies lower than ω_0, the output leads the injection signal. As expected, the phase offsets between the output and injection signals are smaller for a stronger injection. All the simulation results agree with the Spectre simulations.

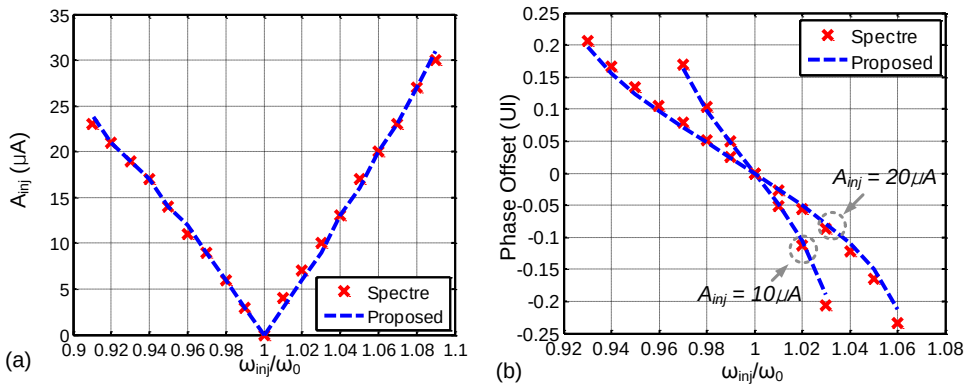


Fig. 6.10 (a) Minimum injection amplitude required for injection-locking and (b) static phase offset between the output and the input injection signal for different injection frequencies.

6.4 Burst-Mode Clock and Data Recovery

The third example is a 20Gb/s burst-mode clock recovery using an LC oscillator [44]. Some communication networks require immediate clock extraction as soon as a data packet arrives (i.e. burst-mode operation). The burst-mode clock recovery is one of ILO's key applications that take advantage of its virtually instantaneous locking capability. In this example, the lock time and lock range are the key performance specifications. Another difference from the previous examples is that the input is an aperiodic random data stream. This example examines if the model can accurately handle such aperiodic perturbations.

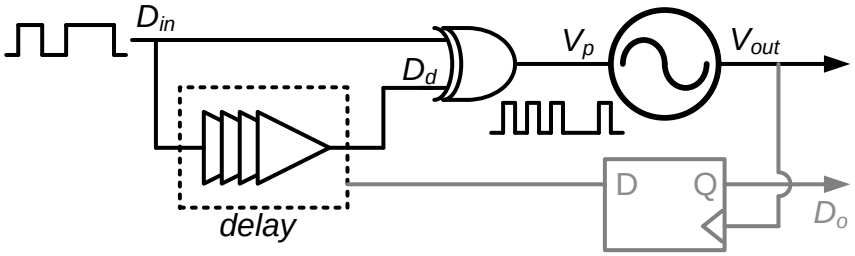
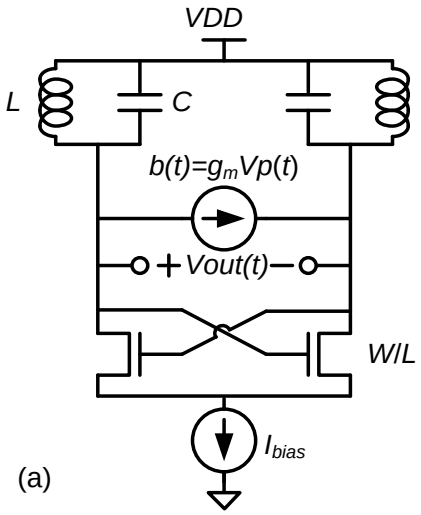


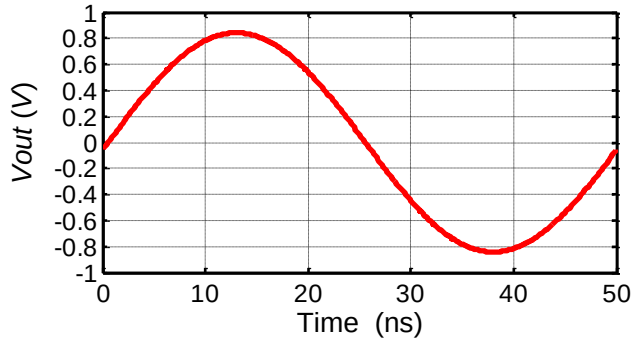
Fig. 6.11 Burst-mode clock recovery circuits [44].

6.4.1 System Model

The block diagram of the burst-mode clock recovery circuits is shown in Fig. 6.11. The input, D_{in}, and its delayed replica, D_d, are XORed to generate pulses, *Vp*, at every data transition. Furthermore, *Vp* has spectral components at the harmonic frequencies of the data rate, which is fed to the LC oscillator to lock its frequency at the data rate. The circuit schematics and parameter values of the LC oscillator used in this example are given in Fig. 6.12(a), and *Vp* is injected through the g_m-stage of $b(t)$. Fig. 6.12(b) and Fig. 6.12(c) show the simulated output waveform and the measured LC oscillator PPV. Again, the PPV is modeled as a piecewise linear function with 16 equally-spaced intervals and the governing nonlinear ODE is decomposed to the second order distortion.



Parameter	Value
L	0.4 nH
C	158 fF
g_m	0.4 mS
W/L	180nm/45nm
VDD	1.5 V
I_{bias}	1.2 mA



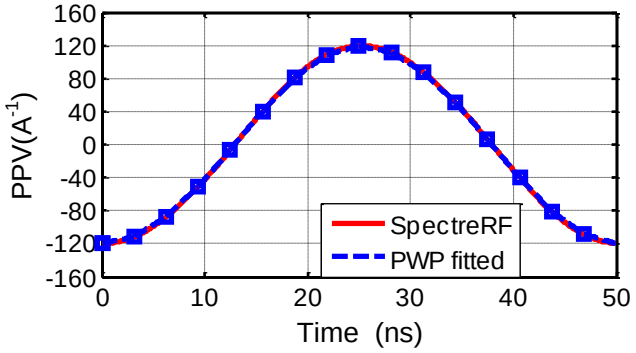


Fig. 6.12 (a) An LC oscillator used for the burst-mode clock recovery, (b) its oscillating waveform, and (c) PPV at the output node.

6.4.2 Simulation Results

Fig. 6.13 shows its phase waveform getting injection-locked to a 20.02-Gb/s pseudo-random bit stream (PRBS) input. Note that the injected data rate is higher than the free-running oscillator frequency. Therefore, in the zoomed-in Fig. 6.13(b) and Fig. 6.13(c), the phase advances to the higher values when there are the input pulses injected into the oscillator and drifts towards the lower values when there is no input pulse. Fig. 6.13 shows the agreement between the proposed method and Spectre simulation. The measured worst-case phase error in this simulation period for 20ns was 0.117-rad. As for the execution times, the proposed method took 31.9secs for 1-μs simulation while Spectre took 134secs, which demonstrated a 4× speed-up.

Fig. 6.14 plots the simulated lock time of the burst-mode clock recovery circuits when the input signal is injected to the LC oscillator with different initial phase offsets. For instance, when the injection signal has an initial phase offset of 0.56UI, the CDR takes the longest at 6.3ns to acquire the phase lock. On the other hand, the CDR can achieve the phase lock almost instantaneously (i.e. lock time=0) when the initial phase offset is 0-UI. Fig. 6.14 shows the discrepancies in the simulated lock times compared to the Spectre simulation for the phase offset values near 0.56UI. In Spectre simulations under these conditions, the LC oscillator experiences considerable amplitude changes during the locking transition shown in Fig. 6.15. Our model cannot capture this behavior since it neglects the amplitude deviation when assuming a sufficiently weak perturbation. Proving that the error is due to the limitation of the phase-domain-only model and not due to the proposed event-driven

method, the PPV-based model written in Verilog-A [55] gave the same results as our event-driven model in Fig. 6.14. For reference, the same 1-μs simulation took 47.7sec for the Verilog-A model.

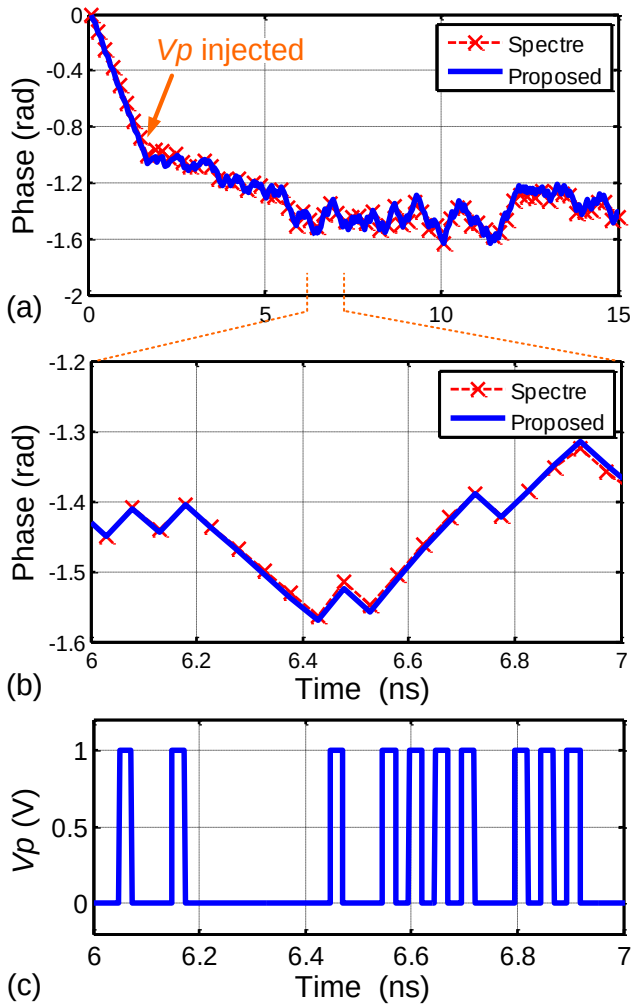


Fig. 6.13 (a) A locking transient of the burst-mode clock recovery and (b) its zoomed-in view (c) with the input signal *Vp*.

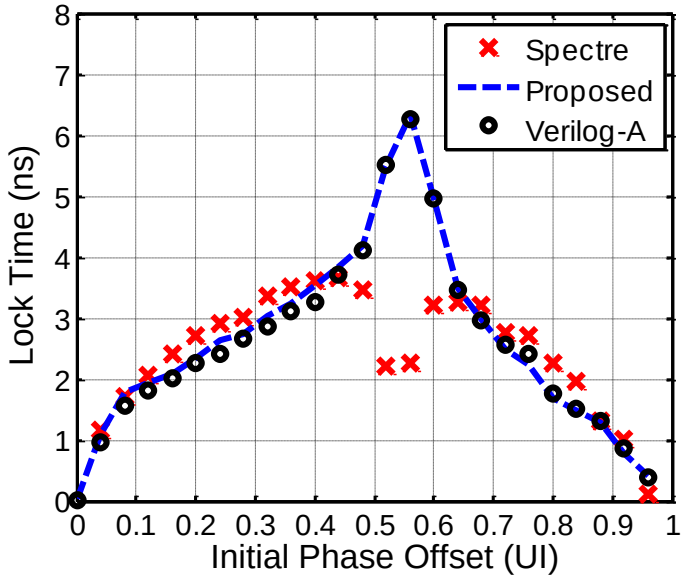


Fig. 6.14 A lock time of the burst-mode clock recovery circuit for different initial phase offsets of the input from the LC oscillator.

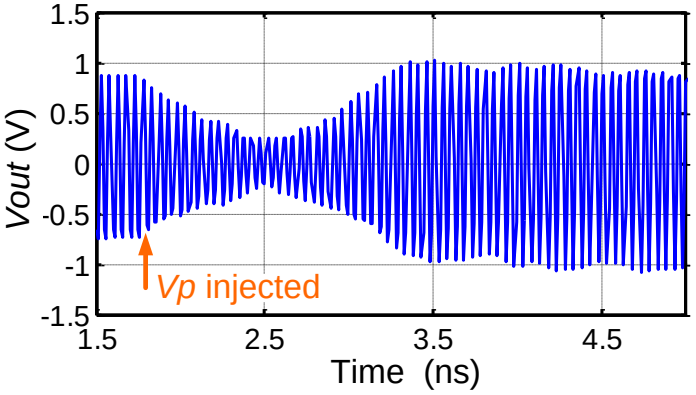


Fig. 6.15 An output waveform of the LC oscillator for the case with an initial phase offset of 0.56UI.

Finally, Fig. 6.16 measures the minimum amplitude of the injection current, $b(t)$, to achieve a phase lock for different input data rates. As expected, the clock recovery circuit requires a larger injection current if the input data rate deviates from its self-oscillating frequency. Fig. 6.16 also shows that the clock recovery circuit needs a large injection current when the input signal, *Vp*, has a short pulse width. This is because the spectral power at the data rate depends on its pulse width, which is controlled by the delay in Fig. 6.11. For instance, the spectral power with a 0.25-UI pulse width is $1/\sqrt{2}$ of that with a 0.5-UI pulse width [44]. Fig. 6.16 agrees well with this theory in that the required amplitude of $b(t)$ with the 0.25-UI delay is $1/\sqrt{2}\times$ lower than that with 0.5-UI delay.

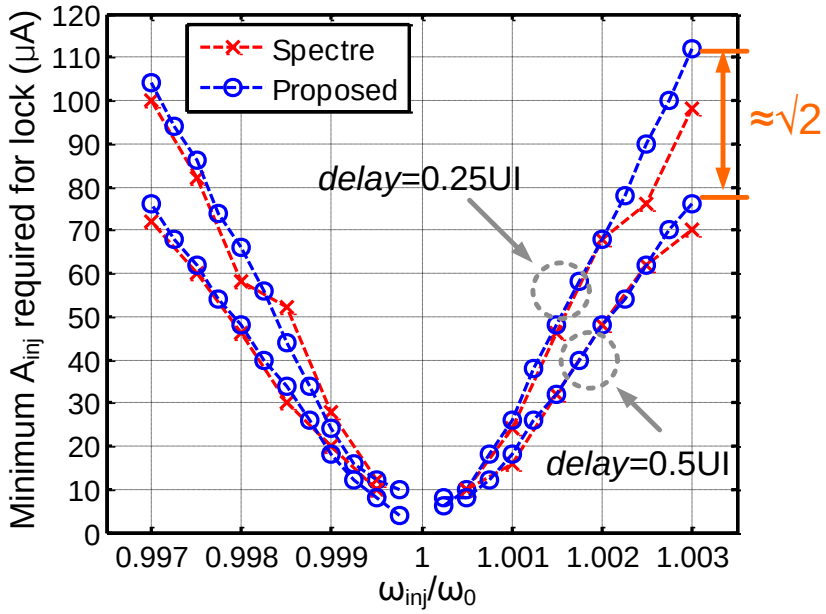


Fig. 6.16. The minimum injection amplitude required for the burst-mode clock recovery circuit to achieve a phase lock for different data rates.

Conclusion

This work proposes an event-driven simulation methodology for analog/mixed-signal behaviors. The proposed method is implemented on a single digital logic simulation platform, SystemVerilog, without relying on an additional analog simulator. Its simulation is fast and independently accurate of the simulator's time-step. This advantage stems from two newly introduced signal-type definitions of clock and analog signals: *xbit* and *xreal*, respectively. By supplementing a real-value time offset to the *xbit*-type signal and a set of coefficients describing the functional form to the *xreal*-type signal, these signals achieve virtually infinite time resolution without relying on a fine time-step. Moreover, this *xreal* functional form enables truly event-driven simulations of analog signals: when there is a change event to the input coefficients, the output coefficients are updated only once. The computation for this update is fully algebraic, and there is no numerical iteration or time-step integration involved.

The proposed method is demonstrated for various analog/mixed-signal systems. First, for a high-speed I/O interface example, the proposed method accurately simulated jitter performances of clock-generating circuits, such as phase-locked loops and clock and data recovery. Also, simulated continuous-time waveforms of linear blocks, such as channels and linear equalizers, were in good agreement with Verilog-A models, achieving a 45x speed-up. Second, for switching power supply

examples, the proposed method showed SPICE-level accuracy with 20~100x faster simulation speed for switched linear systems such as AC-DC boost converters and DC-DC switched-capacitor converters. Furthermore, the proposed method was applied to simulate nonlinear behaviors modeled by a Volterra series. The simulation results showed that it can accurately estimate system performance degradation due to circuit nonlinearities (e.g. a spectral regrowth in an RF transmitter and an eye-opening reduction in a multi-level pulse amplitude modulation transceiver). For both examples, the simulation speed was 300~1000x faster compared to SPICE simulations. Finally, the proposed method was extended to simulate time-varying nonlinear behaviors of injection-locked oscillators. The injection-locked oscillator was modeled based on a perturbation projection vector phase-domain macromodel. Using three examples (an LC oscillator, a ring oscillator, and burst-mode clock recovery circuits), the proposed method achieved the same level of accuracy with 2~30x speed-ups compared to the Spectre simulation.

Bibliography

[1] I. Vassiliou, et al., “A single-chip digitally calibrated 5.15-5.825-GHz 0.18-µm CMOS transceiver for 802.11a wireless LAN,” *IEEE J. Solid-State Circuits*, pp. 2221-2231, Dec. 2003.

[2] S. Son, et al., “A 2.3-mW, 5-Gb/s low-power decision-feedback equalizing receiver front-end and its two-step, minimum bit-error-rate adaptation algorithm,” *IEEE J. Solid-State Circuits*, pp. 2693-2704, Nov. 2013.

[3] R.A. Cottrell, “Event-driven behavioural simulation of analogue transfer functions,” in *Proc. the European Publication Design Automation Conference* (*EDAC*), pp. 240-243. Mar. 1990.

[4] J. Mentzer and T. Wey, “A Verilog mixed signal model of a 10-bit pipeline analog-to-digital converter,” in *Proc. IEEE Behavioral Modeling and Simulation* (*BMAS*) *Workshop*, pp. 115-119, Sep. 2006.

[5] Y. Wang, et al., “Event driven analog modeling for the verification of PLL frequency synthesizers,” in *Proc. IEEE Behavioral Modeling and Simulation* (*BMAS*) *Workshop*, pp. 25-30, Sep. 2009.

[6] K. Kundert and O. Zinke, *The designer's guide to Verilog-AMS*. New York: Springer, 2004.

[7] R. B. Staszewski, “Top-down simulation methodology of a mixed-signal read channel using standard VHDL,” in *Proc. IEEE Dallas Circuits and Systems Workshop on System-on-Chip* (*DCAS*), pp. 1-4, Nov. 2007.

[8] G. D. Cataldo, et al., “Modeling of feedback analog circuits with VHDL,” in *Proc.* 13*th IEEE International Conf. Electronics, Circuits and Systems* (*ICECS*), pp. 882-885, Dec. 2006.

[9] M. Schubert, “An analog-node mode for VHDL-based simulation of RF integrated circuits, ” *IEEE Trans. Circuits and Systems I: Regular Papers*, pp. 2717-2725, Dec. 2009.

[10] C. Wegener, “Method of modeling analog circuits in Verilog for mixed-signal design simulations,” in *Proc. IEEE European Conf. Circuit Theory and Design* (*ECCTD*), pp. 1-5, Sep. 2013.

[11] A. Demir, et al., “Behavioral simulation techniques for phase/delay-locked systems, ” in *IEEE Custom Integrated Circuits Conf.* (*CICC*), pp. 453-456, May 1994.

[12] M.H. Perrot, “Fast and accurate behavioral simulation of fractional-N frequency synthesizers and other PLL/DLL circuits” in *Proc. Design Automation Conf.* (*DAC*), pp. 498 - 503, Jun. 2002.

[13] S. Liao and M. Horowitz, “A Verilog piecewise-linear analog behavior model for mixed-signal validation,” *IEEE Trans. Circuits and Systems I:*

Regular Papers, pp. 2229-2235, Aug. 2014.

[14] S. Joeres, et al,. "Event driven analog modeling of RF frontends" in *Proc. IEEE Behavioral Modeling and Simulation* (*BMAS*) *Workshop*, pp. 46-51, Sep. 2007.

[15] R. B. Staszewski, et al., "Event-driven simulation and modeling of phase noise of an RF oscillator," *IEEE Trans. Circuits and Systems I: Regular Papers*, pp. 723-733, Apr. 2005.

[16] M. Ierssel, et al., "Event-driven modeling of CDR jitter induced by power-supply noise, finite decision-circuit bandwidth, and channel ISI," *IEEE Trans. Circuits and Systems I: Regular Papers*, pp.1306-1315, Jun. 2008.

[17] C. Hedayat, et al., "Modeling and characterization of the 3rd order charge-pump PLL: a fully event-driven approach," *International J. Analog Integrated Circuits and Signal Processing*, pp. 25-45, Apr. 1999.

[18] M.-J. Park, et al., "Fast and accurate event-driven simulation of mixed-signal systems with data supplementation," in *Proc. IEEE Custom Integrated Circuits Conf.* (*CICC*), pp. 1-4, Sep. 2011.

[19] J.-E. Jang, et al., "True event-driven simulation of analog/mixed-signal behaviors in SystemVerilog: a decision-feedback equalizing (DFE) receiver example," in *Proc. IEEE Custom Integrated Circuits Conf.* (*CICC*), pp.1-4, Sep. 2012.

[20] S. A. Maas, *Nonlinear Microwave and RF Circuits*, 2nd. Ed., Norwood, MA: Artech House, 2003, ch. 4.

[21] A. Novak, et al., "Nonlinear system identification using exponential swept-sine signal," *IEEE Trans. Instrumentation and Measurement*, pp. 2220-2229, Aug. 2010.

[22] M. Schoukens, et al., "Parametric identification of parallel Hammerstein systems," *IEEE Trans. Instrumentation and Measurement*, pp. 3931-3938, Dec. 2011.

[23] IEEE Standard for SystemVerilog -- Unified Hardware Design, Specification, and Verification Language, IEEE 1800 SystemVerilog Language Working Group, Feb. 2013.

[24] B. Razavi, *Monolithic phasae-locked loops and clock recovery circuits*, Piscataway, NJ: IEEE Press, 1996.

[25] Verilog-A model library, http://www.designers-guide.org/VerilogAMS, retrieved on 1 Aug. 2014.

[26] M. Takahashi, et al., "VCO jitter simulation and its comparison with measurement," in *Proc. Asia and South Pacific Design Automation Conf.* (*ASP-DAC*), pp. 85-89, Jan. 1999.

[27] R. W. Erickson and D. Maksimovic, *Fundamentals of Power*

Electronics, 2nd Ed., New York: Springer, 2001.

[28] J. H. Alimeling and W. P. Hammer, "PLECS-piece-wise linear electrical circuit simulation for Simulink," in *Proc. IEEE International Conf. Power Electronics and Drive Systems* (*PEDS*), pp. 355-360, Jul. 1999.

[29] D. Maksimovic, et al., "Modeling and simulation of power electronic converters," in *Proc. of the IEEE*, pp. 898-912, Jun. 2001.

[30] H. Jin, "Behavior-mode simulation of power electronic circuits," *IEEE Trans. Power Electronics*, pp. 443-452, Mar. 1997.

[31] G.W. Wester, et al., "Low-frequency characterization of switched dc-dc converters," *IEEE Trans. Aerospace and Electronic Systems*, pp. 376-385, Mar. 1973.

[32] E. Dijk, et al., "PWM-switch modeling of DC-DC converters," *IEEE Trans. Power Electronics*, pp. 659-665, Jun. 1995.

[33] B. Singh, et al., "A review of single-phase improved power quality AC-DC converters," *IEEE Trans. Industrial Electronics*, pp. 962-981, May 2003.

[34] H.-P. Le, et al., "Design techniques for fully integrated switched-capacitor DC-DC converters," *IEEE J. of Solid-State Circuits*, pp. 2120-2131, Sep. 2011.

[35] A. Zhu, et al., "An efficient Volterra-based behavioral model for

wideband RF power amplifiers, ” in *IEEE MTT-S Int'l Microwave Symposium Dig.*, pp. 787-790, Jun. 2003.

[36] T. Toifl, et al., “A 22-Gb/s PAM-4 receiver in 90-nm CMOS SOI technology,” *IEEE J. Solid-State Circuits*, pp. 954-965, Apr. 2006.

[37] J. Roychowdhury, “Reduced-order modeling of time-varying systems,” *IEEE Trans. Circuit and Systems II: Analog and Digital Signal Processing*, pp. 1273-1288, Oct. 1999.

[38] B. Razavi, “A study of injection locking and pulling in oscillators,” *IEEE J. Solid-State Circuits*, pp. 1415-1424, Sep. 2004.

[39] J. Lee and H. Wang, “Study of subharmonically injection-locked PLLs,” *IEEE J. Solid-State Circuits*, pp. 1539-1553, May. 2009.

[40] S. Ye, et al., “A multiple-crystal interface PLL with VCO realignment to reduce phase noise,” *IEEE J. Solid-State Circuits*, pp. 1795-1803, Dec. 2002.

[41] H.R. Rategh and T.H. Lee, “Superharmonic injection-locked frequency dividers,” *IEEE J. Solid-State Circuits*, pp. 813-821, Sep. 1999.

[42] A. Mirzaei, et al., “The quadrature LC oscillator: a complete portrait based on injection locking,” *IEEE J. Solid-State Circuits*, pp. 1916-1931, Sep. 2007.

[43] I.-T. Lee, et al., “A 4.8-GHz dividerless subharmonically injection-

locked all-digital PLL with a FOM of 252.5 dB," *IEEE Trans. Circuits and Systems II: Express Briefs*, pp. 547-551, Jul. 2013.

[44] J. Lee and M. Liu, "A 20-Gb/s burst-mode clock and data recovery circuit using injection-locking technique," *IEEE J. Solid-State Circuits*, pp. 619-630, Mar. 2008.

[45] A. Demir and J. Roychowdhury, "Phase noise in oscillators: a unifying theory and numerical methods for characterization," *IEEE Trans. Circuits and Systems I: Regular Papers*, pp. 655-674, May 2000.

[46] R. Adler, "A study of locking phenomena in oscillators," in *Proc. of the IEEE*, pp. 1380-1385, Oct. 1973.

[47] P. Bhansali and J. Roychowdhury, "Gen-Adler: the generalized Adler's equation for injection locking analysis in oscillators," in *Proc. Asia and South Pacific Design Automation Conf.* (*ASP-DAC*), pp. 522-527, Jan. 2009.

[48] A. Hajimiri and T.H. Lee, "A general theory of phase noise in electrical oscillators," *IEEE J. Solid-State Circuits*, pp. 179-194, Feb. 1998.

[49] Paolo Maffezzoni, "Analysis of oscillator injection locking through phase-domain impulse-response," *IEEE Trans. Circuits and Systems I: Regular Papers*, pp. 1297-1305, Jun. 2008.

[50] P. Vanassche, et al., "On the difference between two widely publicized

methods for analyzing oscillator phase behavior," in *Proc. International Conf. Computer Aided Design* (*ICCAD*), pp. 229-233, Nov. 2002.

[51] X. Lai and J. Roychowdhury, "Capturing oscillator injection locking via nonlinear phase-domain macromodels," *IEEE Trans. Microwave Theory and Techniques*, pp. 2251-2261, Sep. 2004.

[52] X. Lai and J. Roychowdhury, "Fast PLL simulation using nonlinear VCO macromodels for accurate prediction of jitter and cycle-slipping due to loop non-idealities and supply noise," in *Proc. Asia and South Pacific Design Automation Conf.* (*ASP-DAC*), pp. 459-464, Jan. 2005.

[53] A. Demir and J. Roychowdhury, "A reliable and efficient procedure for oscllator PPV computation, with phase noise macromodeling applications," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, pp. 188-197, Feb. 2003.

[54] J. Roychowdhury, "Exact analytical equations for predicting nonlinear phase errors and jitter in ring oscillators," in *Proc. International Conf. VLSI Design* (*ICVD*), pp. 516-521, Jan. 2005.

[55] B. Gu, et al., "Implementing nonlinear oscillator macromodels using Verilog-AMS for accurate prediction of injection locking behaviors of oscillators," in *Proc. IEEE Behavioral Modeling and Simulation* (*BMAS*) *Workshop*, pp. 43-47, Sep. 2005.

초 록

최근 시스템온칩은 아날로그와 디지털 회로가 긴밀하게 연결되어 디자인 되고있다. 따라서 시스템온칩을 빠르고 체계적으로 검증하기 위해서는 효율적인 혼성 신호 시뮬레이터가 필요하다. 하지만, 현존하는 시뮬레이터들은 아날로그 회로를 시뮬레이션할 때 시뮬레이션 속도와 정확도가 서로 상충되어 효율적인 시뮬레이션을 할 수 없다. 본 연구는 이러한 상충을 극복하여 정확하면서 빠른 혼성 신호 행동 시뮬레이터를 구현하는 방법을 제안하였다.

본 연구가 제안하는 방법의 가장 큰 특징은 지속 시간 신호를 표현하는 방식이다. 클럭 신호는 시간 오프셋을 추가하여 신호의 전환 시간을 정확히 표현하며, 아날로그 신호는 함수의 계수들을 추가하여 지속 시간 파형을 함수의 꼴로 표현한다. 이러한 부가적인 정보를 활용하면, 시뮬레이터의 시간 단계에 상관없이 정확한 지속 시간 신호를 시뮬레이션 할 수 있다. 더욱이, 함수로 표현된 지속 시간 파형은 아날로그 회로를 사건 구동 방식으로 시뮬레이션 가능하게 하여 시뮬레이션 속도를 크게 향상 시킬 수 있다.

본 연구는 제안한 혼성 신호 시뮬레이션 방법을 다양한 시스템 예제들을 통하여 검증하였다. 첫번째로, 위상고정루프, 클럭데이터복원기와 같은 시간 정보에 민감한 회로들과 채널, 이퀄라이저와 같은 선형 회로들을 포함하는 고속 입출력 인터페이스 예제를 제안한 방법이 정확히 시뮬레이션 하는 것을 보였다. 두번째로, 부스트컨버터, 스위칭커패시터컨버터와 같은 전원 공급 회로 예제들에서 제안한 방법이 스위칭 선형 시스템에 적용되

는 것을 보였다. 추가하여, 제안한 방식을 볼테라 시리즈로 모델된 약한 비선형 행동에 적용하여 무선 주파수 송신기와 고속 입출력 인터페이스 이퀄라이저를 시뮬레이션하였다. 마지막으로, 다양한 주입 동기 발진 회로를 시뮬레이션하여 제안한 방식이 시변 비선형 회로에도 확대될 수 있음을 확인하였다. 위의 예제들의 시뮬레이션 결과에 기반하여 제안한 방식이 현존하는 아날로그 행동 시뮬레이터들과 같은 정확도를 가지며 수십배에서 수백배까지 빠른 속도로 시뮬레이션 할 수 있음을 검증하였다.

주요어 : 사건 구동 방식의 모의 실험, 행동 모델, 혼성 신호 시스템, 시스템베릴로그, 고속 입출력 인터페이스, 스위칭 전원 공급 회로, 볼테라 시리즈 모델, 주입 동기 발진기

학 번 : 2011-30974