scientific analog

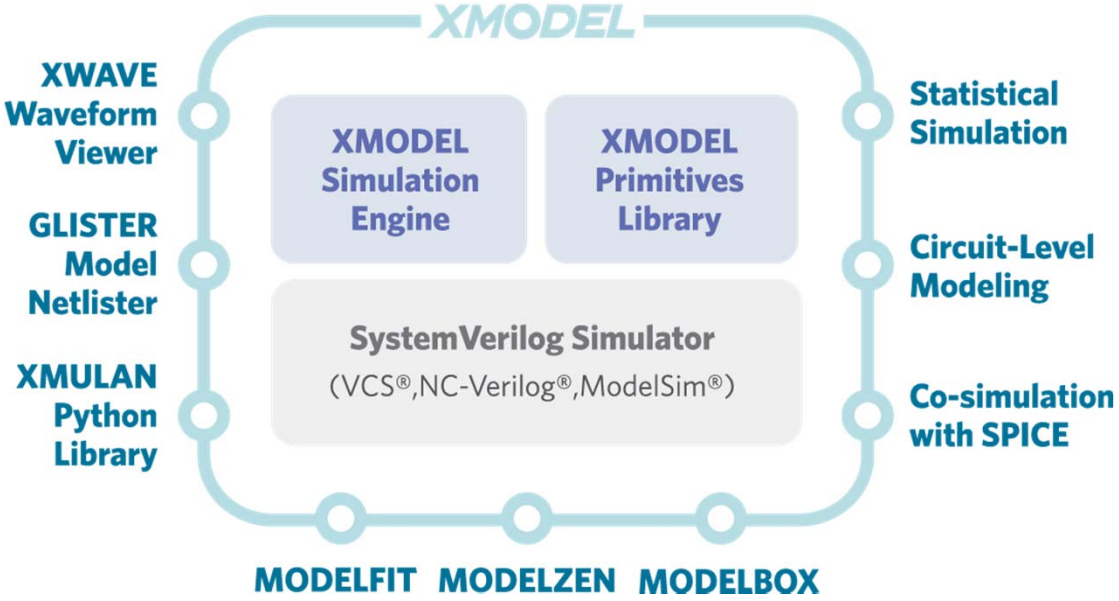
# *XMODEL:* A Fast and Accurate Analog/Mixed-Signal Simulator in SystemVerilog

**Scientific Analog, Inc.**

July 2017

# *XMODEL* is…

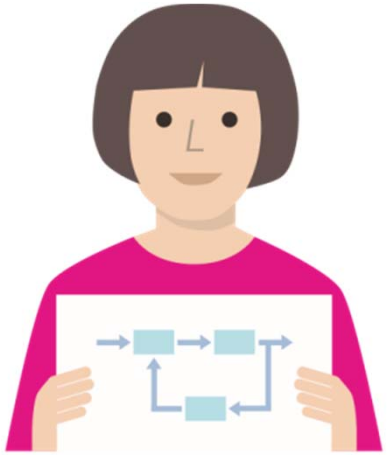
- *An* ***event-driven*** analog/mixed-signal behavioral simulator that is entirely based on ***SystemVerilog***
  - ***Event-driven***: both fast and accurate in simulating analog
  - ***SystemVerilog***: implementation-friendly and compliant with most digital verification flows (e.g. UVM)



# *XMODEL* is for Everyone!

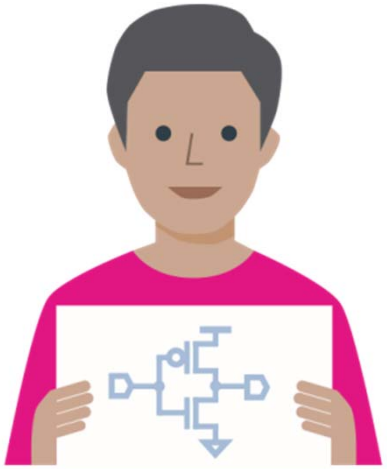
- *XMODEL* is a versatile platform that can be used for multiple purposes in a unified way

**Top-Down Tool**



System Architects

**Bottom-Up Tool**



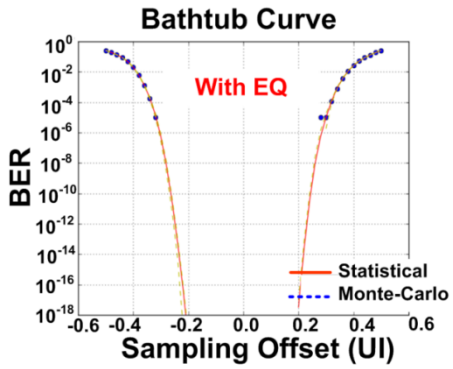
Circuit Designers

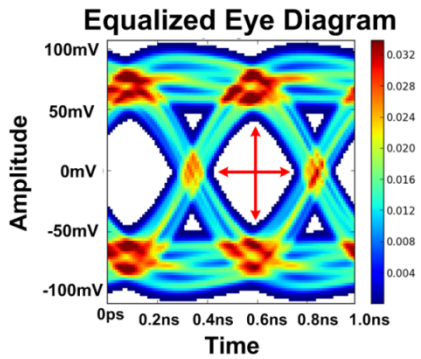
**Sign-off Tool**

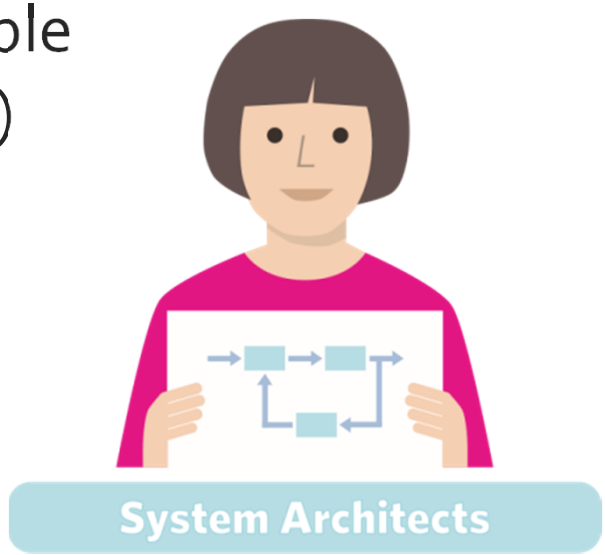
Verification Engineers

# *XMODEL* as Top-Down Tool

- Explore architectural choices using *XMODEL* as fast analog/mixed-signal functional simulator
  - e.g. compare BERs of high-speed links with different clocking and equalization schemes
  - Faster speed compared to SPICE, Verilog-A/MS
  - Digital models are directly synthesizable after verification (vs. Matlab/Simulink)

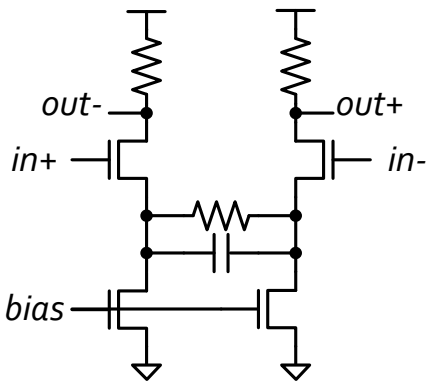


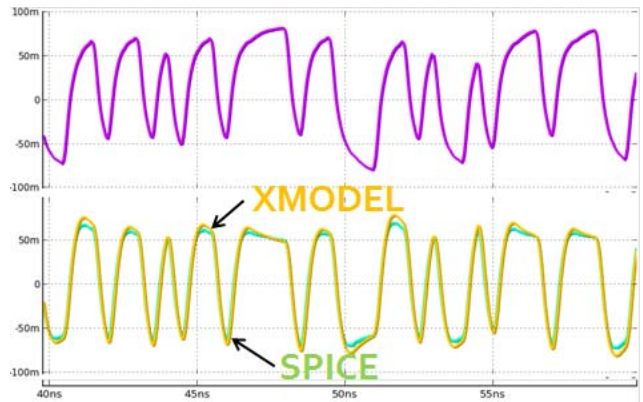



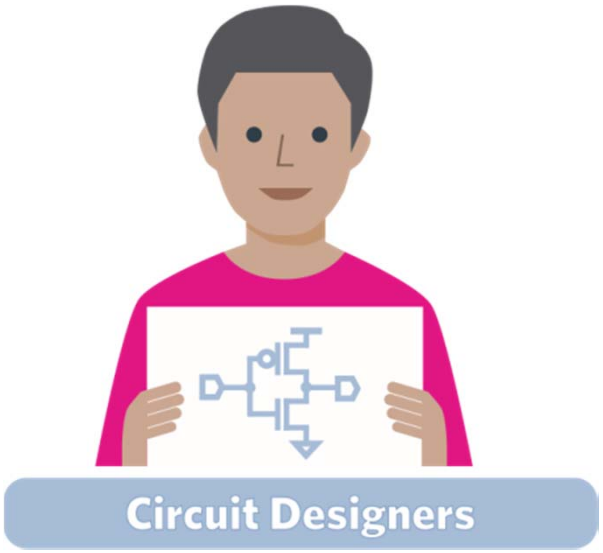



# *XMODEL* as Bottom-Up Tool

- Write or extract analog models easily with *XMODEL*
  - Write functional models with a rich set of primitives
  - Use circuit-level modeling (CLM) for structural models
  - Calibrate functional models or extract structural models using *MODELFIT* and *MODELZEN*

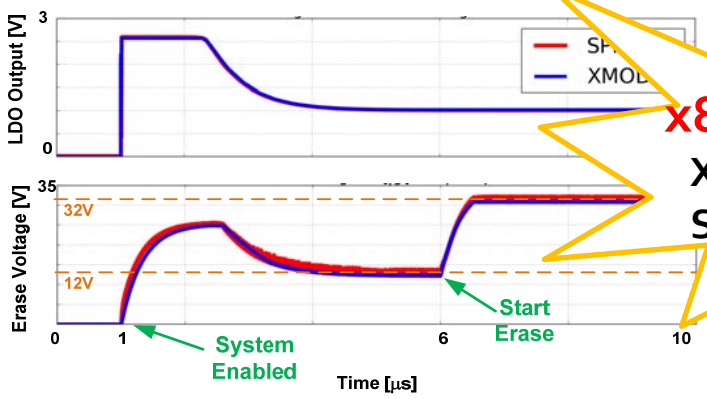


# *XMODEL* as Sign-Off Tool

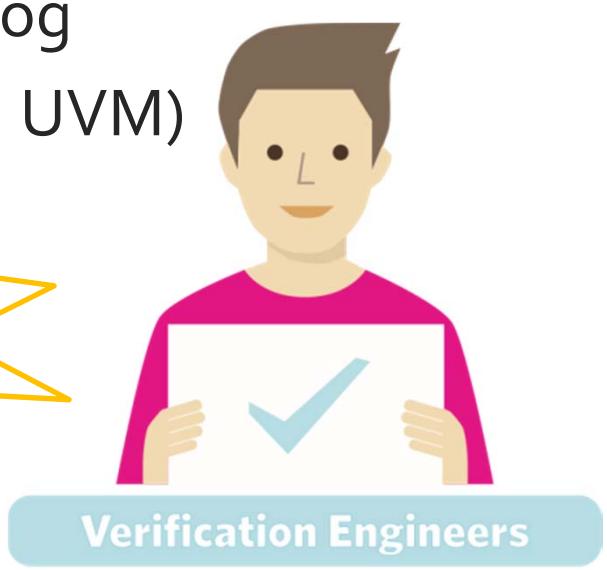
- Verify chip-level functionality of large-scale systems consisting of both digital and analog components
  - e.g. processors with high-speed I/Os, NAND flash memories with HV charge-pumps, …
  - 10~100x faster simulation compared to Verilog-AMS and Real-Number Verilog
  - Fully compatible with digital flows (e.g. UVM)




**x846 Speed-Up!**

**XMODEL: 0.83 sec**

**SPICE : 11.7 min**



# Advanced Features

- **Cadence Virtuoso integration support (GLISTER)**
  - Create models graphically with a schematic editor
- **Model extraction and generation support (*MODELFIT/MODELZEN*)**
  - Calibrate or extrct models that agree well with SPICE
- **Statistical simulation support (STATSIM)**
  - Simulate probabilities of statistical events such as BER, jitter, phase noise, etc., during time-domain simulation
- **Verilog-SPICE co-simulation support**
  - Run *XMODEL* models with SPICE

# 3 Main Products of Scientific Analog

XMODEL

- ***XMODEL*** enables fast and accurate simulation of analog/mixed-signal systems entirely in SystemVerilog

GLISTER

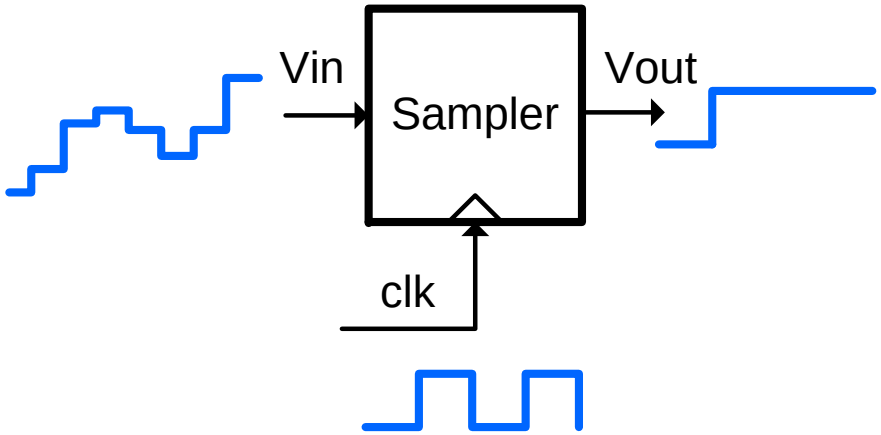
- ***GLISTER*** is a graphical user interface of *XMODEL* and *MODELZEN* integrated into Cadence Virtuoso Schematic Editor

MODELZEN

- ***MODELZEN*** can automatically generate *XMODEL* models from your circuit schematics or netlists

# Basic Concepts of *XMODEL*

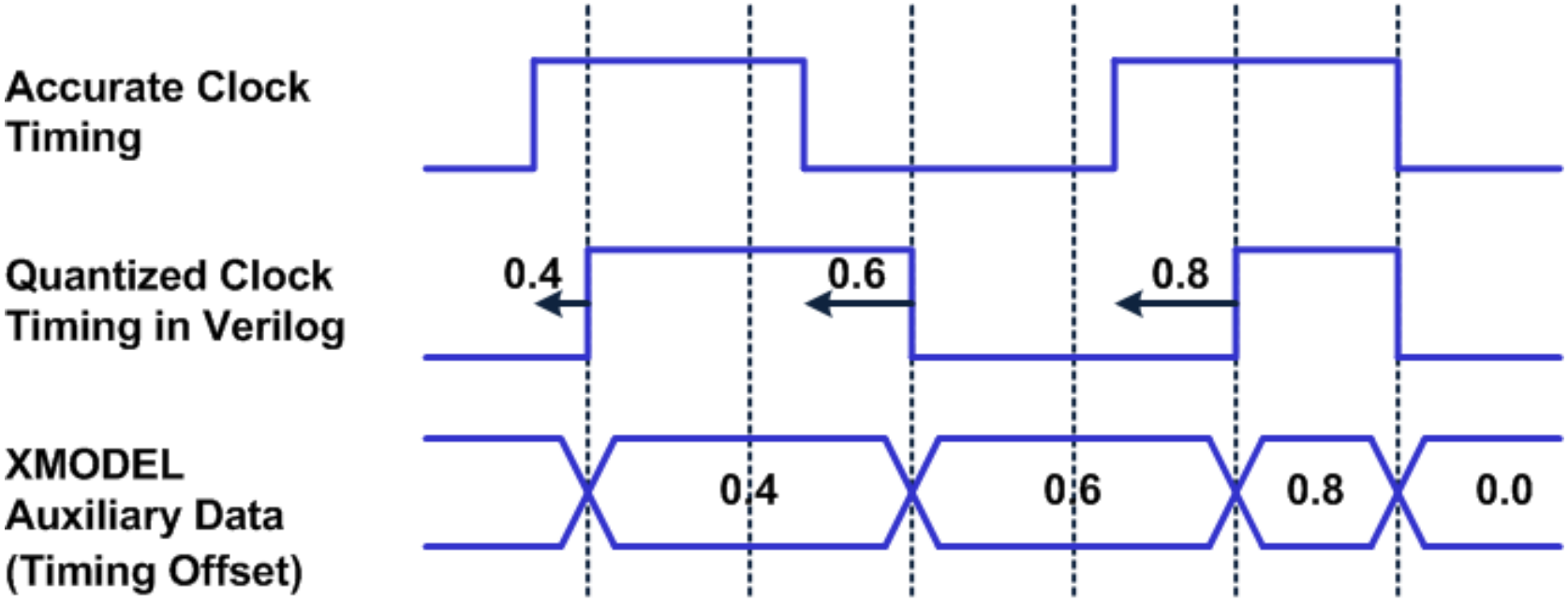
- *XMODEL* started with a motivation to enable efficient event-driven analog simulation in Verilog
  - Since fixed-timestep modeling in Verilog typically encounter conflicting trade-offs between speed and accuracy




```
`timescale 10ps/10ps

module sampler(clk, Vin, Vout);
    always @(posedge clk) begin
        Vout <= (Vin>Vth) ? 1:0;
    end
endmodule
```

# Case 1: Expressing Clock Timing

- In Verilog, to express accurate clock timing one needs to use very fine simulation time steps
- *XMODEL* removes this dependency by adding explicit timing information to the signal (***xbit***)

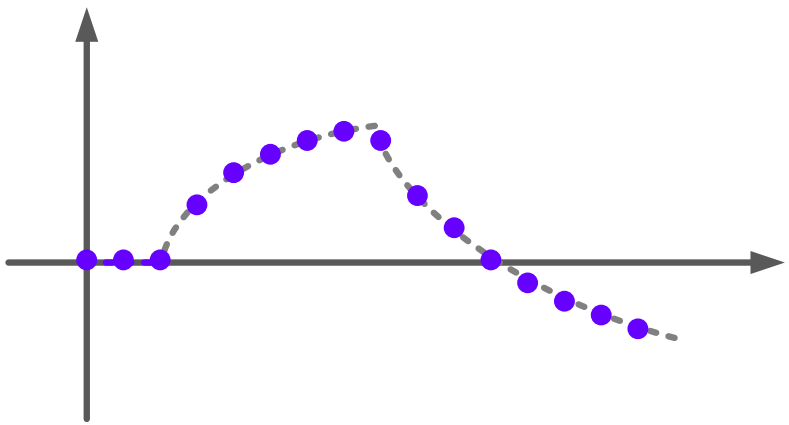


# Case 2: Expressing Analog Signals

- *XMODEL* expresses analog signals in functional forms instead of using a series of time-value pairs:

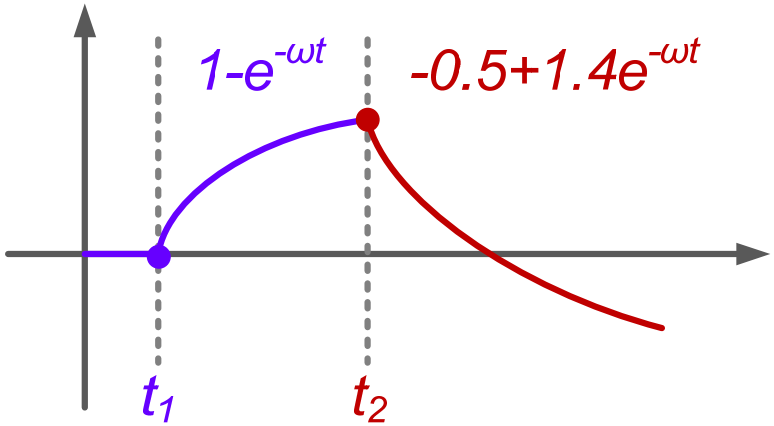
$$x(t) = \sum_i c_i t^{m_i} e^{-a_i t}$$

**SPICE**



Accuracy relies on fine time step

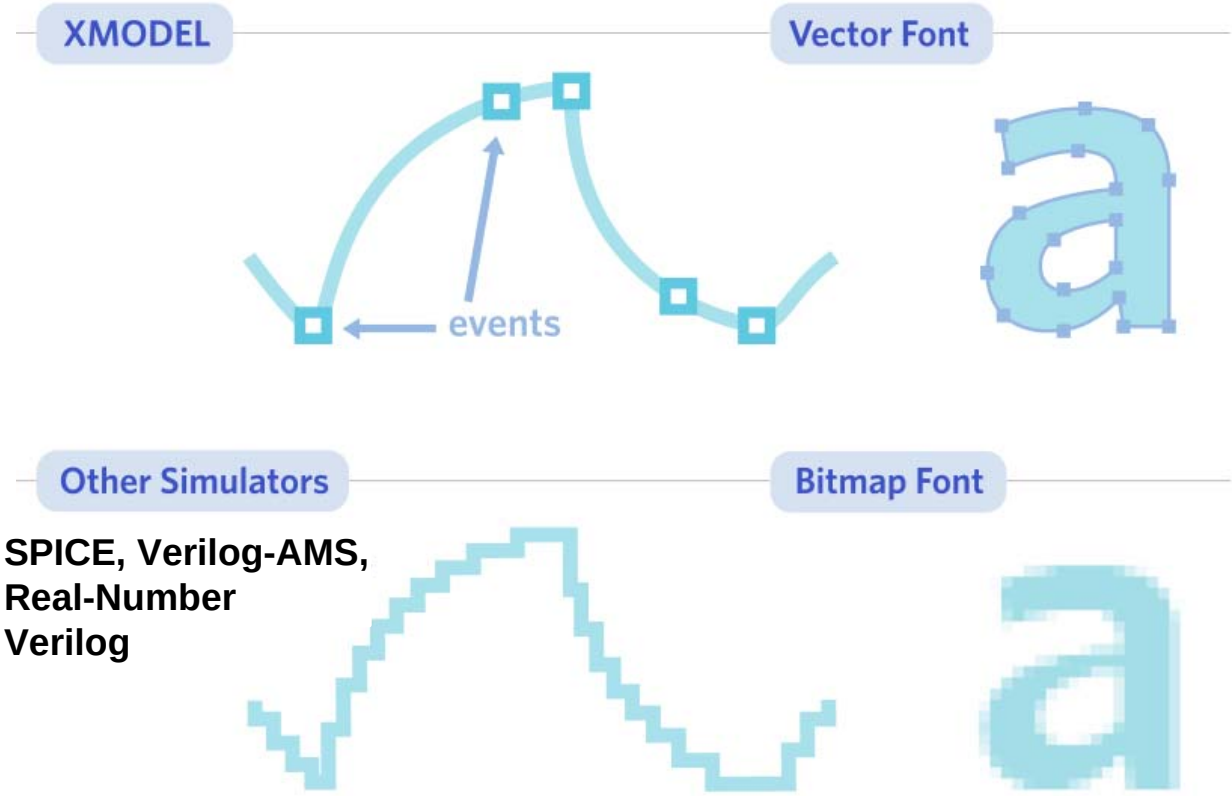
**XMODEL**




Events occur only when the coefficients are updated

# Analogy with Vector Fonts

- *XMODEL* is like vector fonts, achieving the best accuracy and speed regardless of time steps

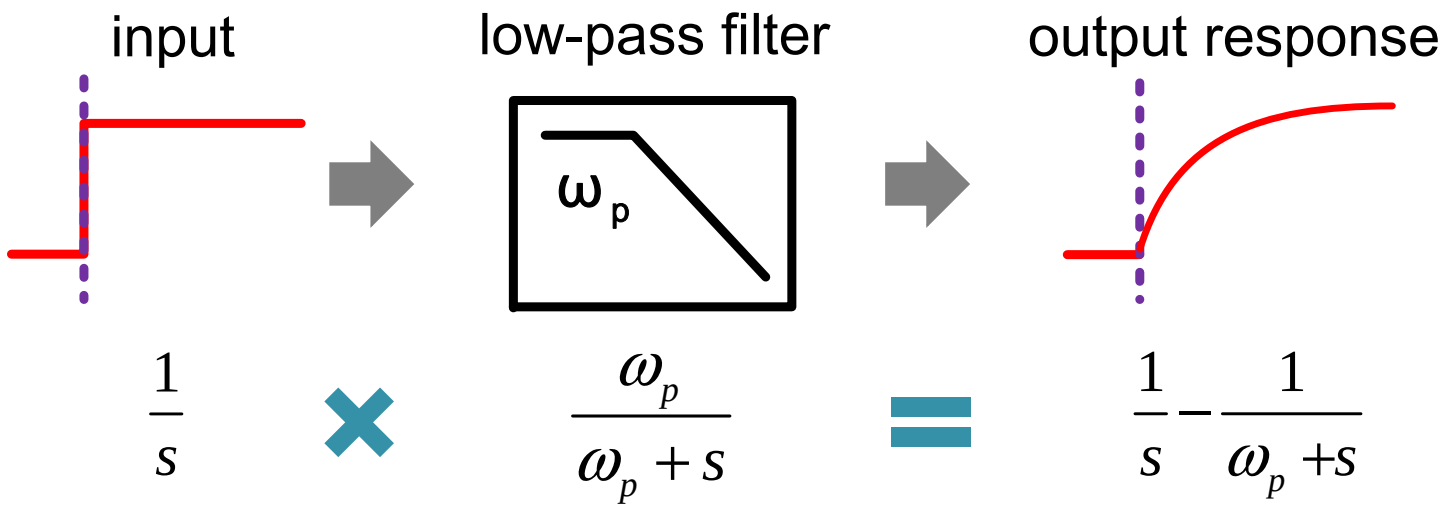


# Simulating Analog System Responses

- Signals have an alternate s-domain representation:

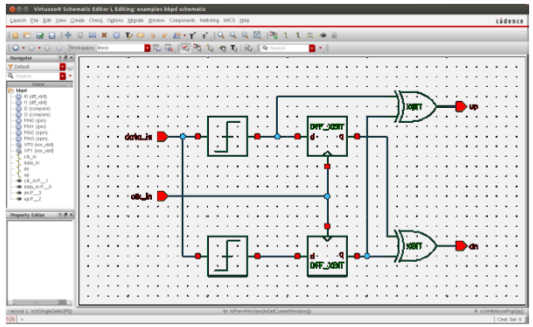
$$x(t) = \sum_i c_i t^{m_i - 1} e^{-a_i t} u(t) \xrightarrow{\mathcal{L}} X(s) = \sum_i \frac{b_i}{(s + a_i)^{m_i}}$$

- With this, the response of a linear system can be computed algebraically without integrating ODE:



# Usage Model of *XMODEL*

Schematics



**Model & Testbenches**

Verilog Sources

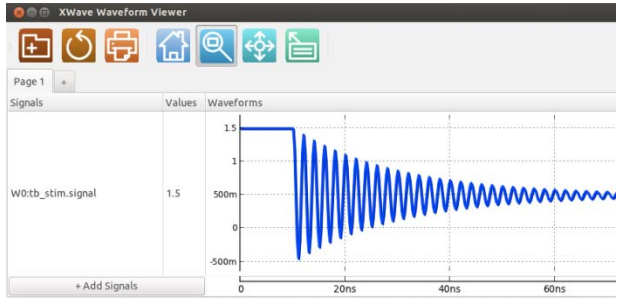
```
module sc_converter(
      input xreal in,
      output xreal out,
      input xbit ck, ckb
);
xreal       n1, n2;
switch      sw1(.pos(in), .neg(n1), .ctrl(ck));
switch      sw2(.pos(n1), .neg(out), .ctrl(ckb));
switch      sw3(.pos(n2), .neg(out), .ctrl(ck));
switch      sw4(.pos(n2), .neg(`ground), .ctrl(ckb));
capacitor   #(.C(1e-12)) C1(.pos(n1), .neg(n2));
capacitor   #(.C(1e-12)) C2(.pos(n2), .neg(`ground));
endmodule
```

***XMODEL* Primitives**

**SystemVerilog Simulator**
(VCS, Questa, NC-Verilog)

***XMODEL* Event-Driven Simulation Engine**

Waveforms



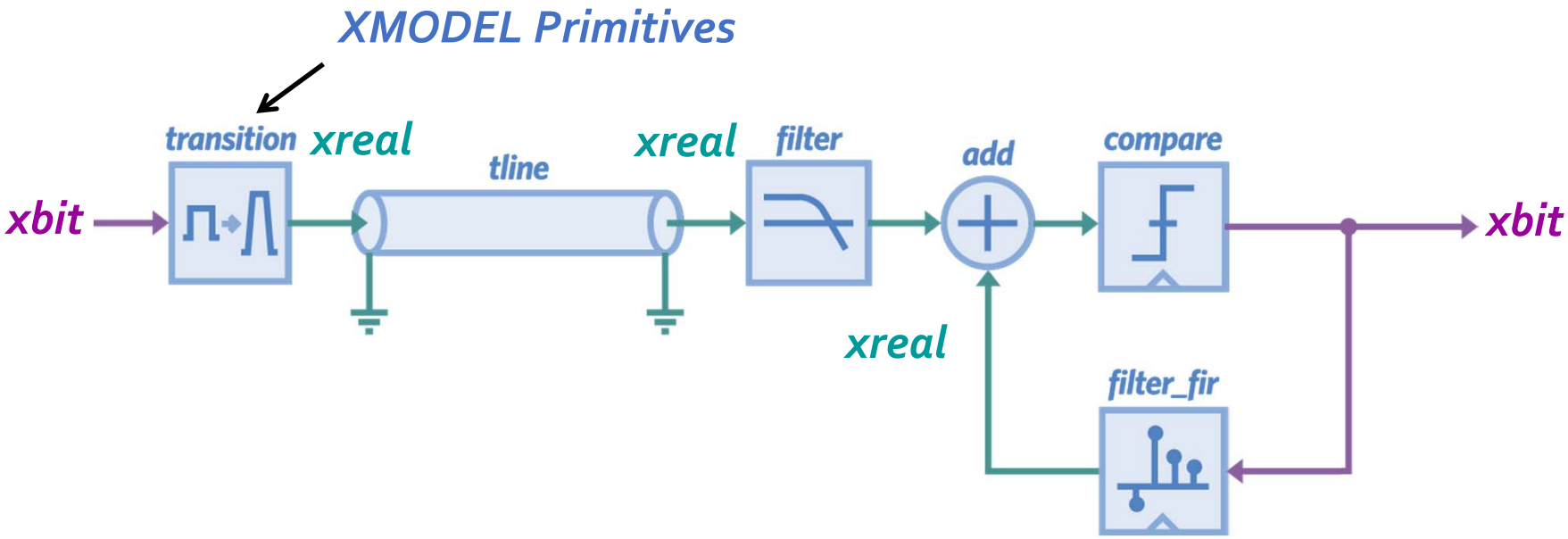

**Simulation Results**

Reports

```
#     _  ____   _____   __   _____
#    | |/_/  |/  / __ \/ _ \/ __/ /  (TM)
#   _|  |/ /|_/ / /_/ / // / _// /__
#  /_/|_/_/  /_/\____/____/___/____/  ANALOG/MIXED-SIGNAL SIMULATOR
#
# XMODEL Development Base
# Copyright (c) 2012-2015 Scientific Analog, Inc.
# All rights reserved and patents pending.
#
#LMX: LM-X feature 'XMODEL' has been successfully checked out.
#LMX: Version: 2.0 (full)
#LMX: Expiration: 2016-05-31 23:59
#LMX: License type: local
#
# WRONG   : A=xxxx, B=xxxx: S=xxxx, C=x
# CORRECT: A=0100, B=1101: S=0001, C=1
# CORRECT: A=1101, B=0110: S=0011, C=1
# CORRECT: A=0011, B=0110: S=1001, C=0
# CORRECT: A=1011, B=1110: S=1001, C=1
# CORRECT: A=1110, B=0001: S=1111, C=0
# CORRECT: A=0001, B=1000: S=1001, C=0
# CORRECT: A=0010, B=1001: S=1011, C=0
```

# Composing Models with *XMODEL*

- Use a rich set of *XMODEL* primitives as building blocks
- Connect them using ***xbit*** or ***xreal*** typed signals
  - ***xbit***: timing-accurate binary pulses (e.g. clocks)
  - ***xreal***: continuous-time analog signals (including V & I)



# *XMODEL* Primitives

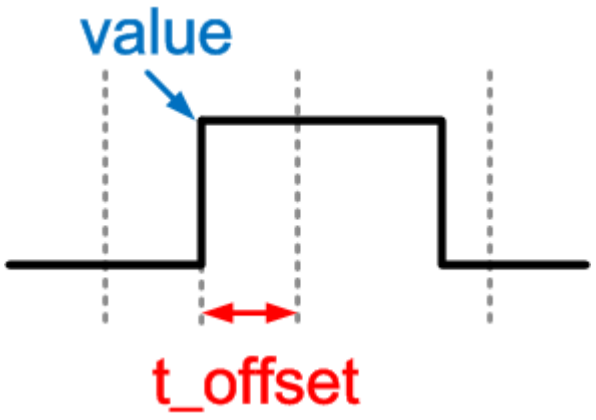
- About 100 primitives in 7 categories
- Superset of SPICE elements, Verilog-A operators, …

```
`include "xmodel.h"

module tb_txeq();

parameter real tr = 125e-12;

// parameter definitions and signal declarations
xbit          clk, data_in;
xbit          [1:0] data;
xreal         [1:0] drv;
xreal         tx_out;

// clock and data generation
clk_gen       #(.freq(1.0e9), .RJ_rms(1.0e-12)) clk_tx(clk);
prbs_gen      #(.length(15)) data_gen(.trig(clk), .out(data_in));
dff_xbit      dff0(.d(data_in), .q(data[0]), .clk(clk));
dff_xbit      dff1(.d(data[0]), .q(data[1]), .clk(clk));

// transmitter with FIR equalizer
transition    #(.value0(-1.0), .value1(1.0),
                .rise_time(tr), .fall_time(tr))
              trans0(.in(data[0]), .out(drv[0]));
transition    #(.value0(-1.0), .value1(1.0),
                .rise_time(tr), .fall_time(tr))
              trans1(.in(data[1]), .out(drv[1]));
add           #(.num_in(2), .scale('{-0.1, 0.4}))
              add(.in(drv), .out(tx_out));
endmodule
```

# *XMODEL* Data Types: *xbit* and *xreal*

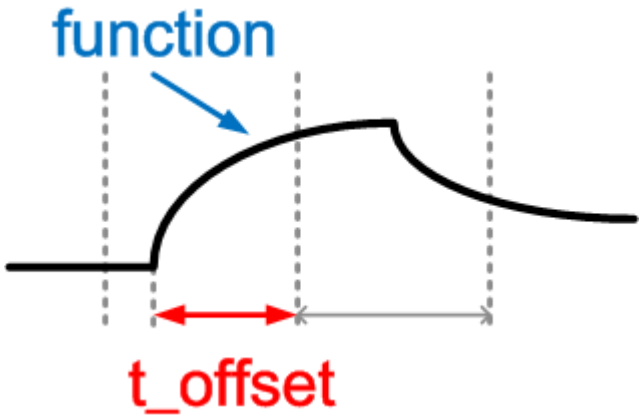
- ***xbit*** for digital signals

```
typedef struct {
    bit value;
    real t_offset;
} xbit;
```




- ***xreal*** for analog signals

```
typedef struct {
    chandle param_set;
    real t_offset;
    event flag;
} xreal;
```



# *XMODEL* Code Example

```
module transceiver;
  xbit        tx_clk, in_prbs;
  xreal       out_drv, out_ch, out_eq;
```

**Signal Declarations**

```
  // transmitter
```

**Stimuli Generation**

```
  clk_gen     #(.freq(ref_freq), .RJ_rms(RJ_rms))  clk_tx(tx_clk);
  prbs_gen    prbs_gen(.trig(tx_clk),.out(in_prbs));
  transition  #(.rise_time(Tr),.fall_time(Tr),.valueo(valueo),.value1(value1))
              driver(.in(in_prbs),.out(out_drv));
```

**Description of System Behaviors**

```
  // channel
  filter      #(.filename("./channel.dat")) channel(.in(out_drv), .out(out_ch));
  // equalizer
  filter      #(.gain(0.66), .poles('{0.5e9, 0.0, 1e9, 0.0}), .zeros('{0.15e9,0.0}))
              equalizer(.in(out_ch), .out(out_eq));
```

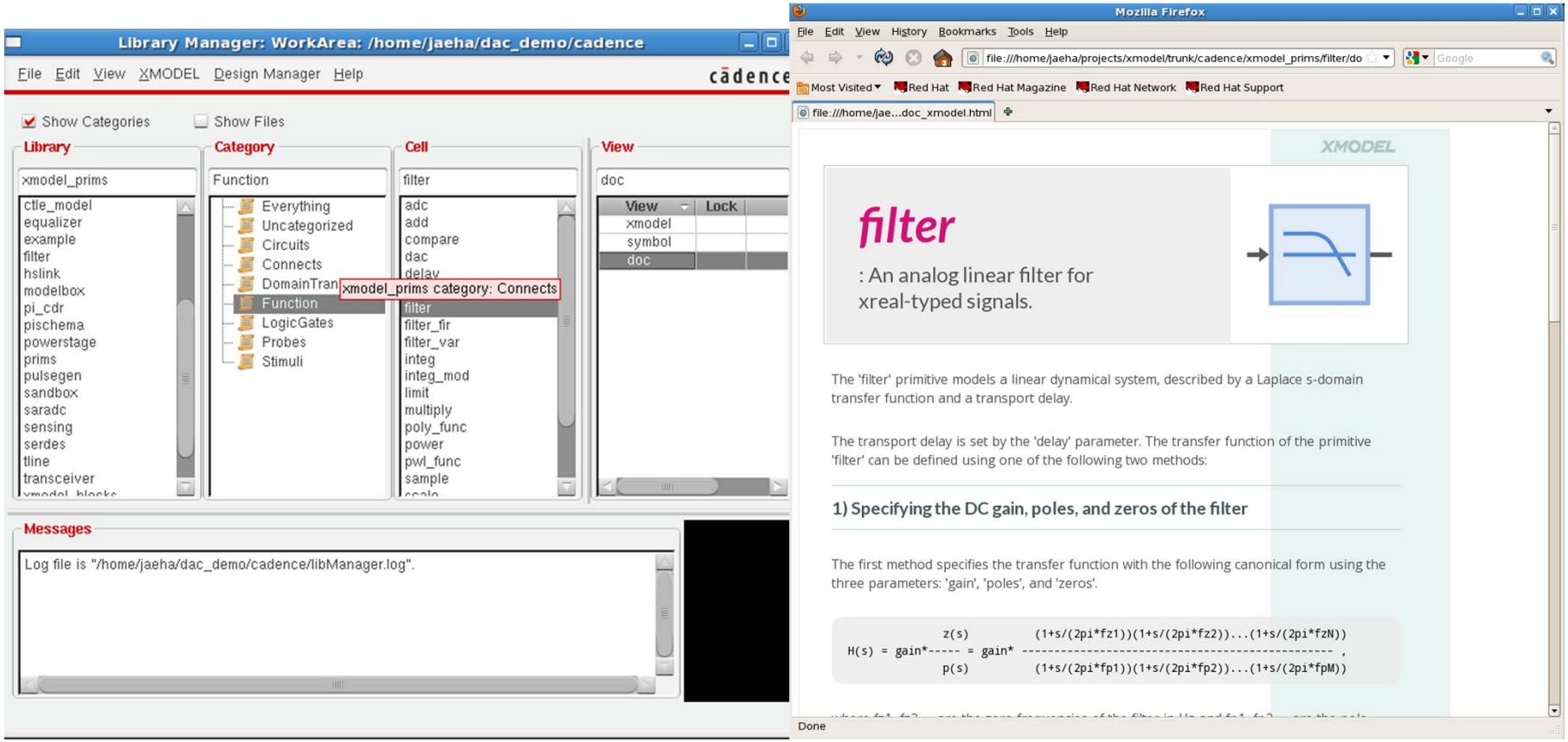
```
  // probes
  probe_xreal eq_output(out_eq);
  probe_xreal ch_output(out_ch);
```

**Waveform Recording**

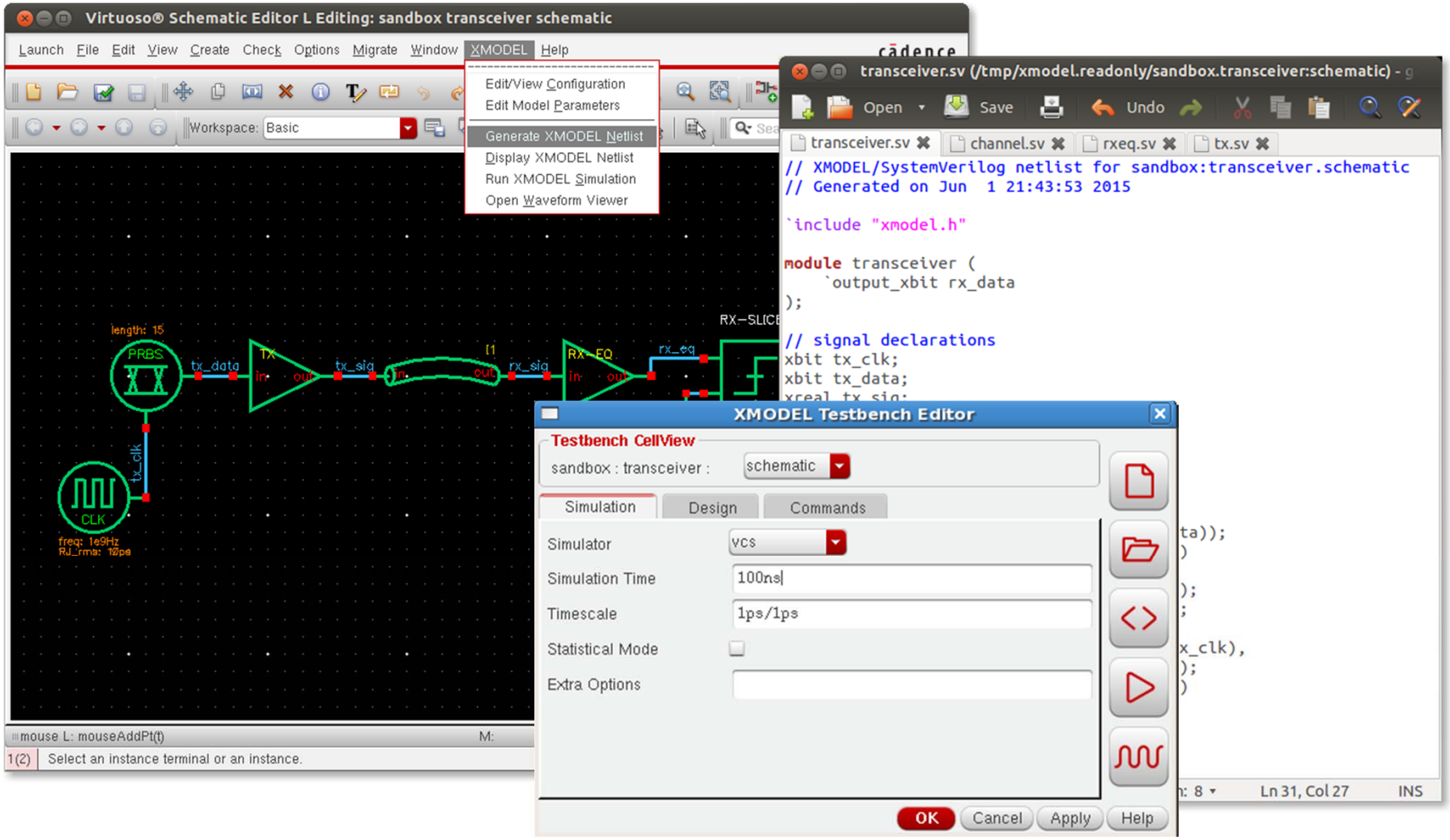
```
endmodule
```

# *XMODEL* Primitives in Virtuoso

- All the *XMODEL* primitives are available as symbols in Cadence Virtuoso environment

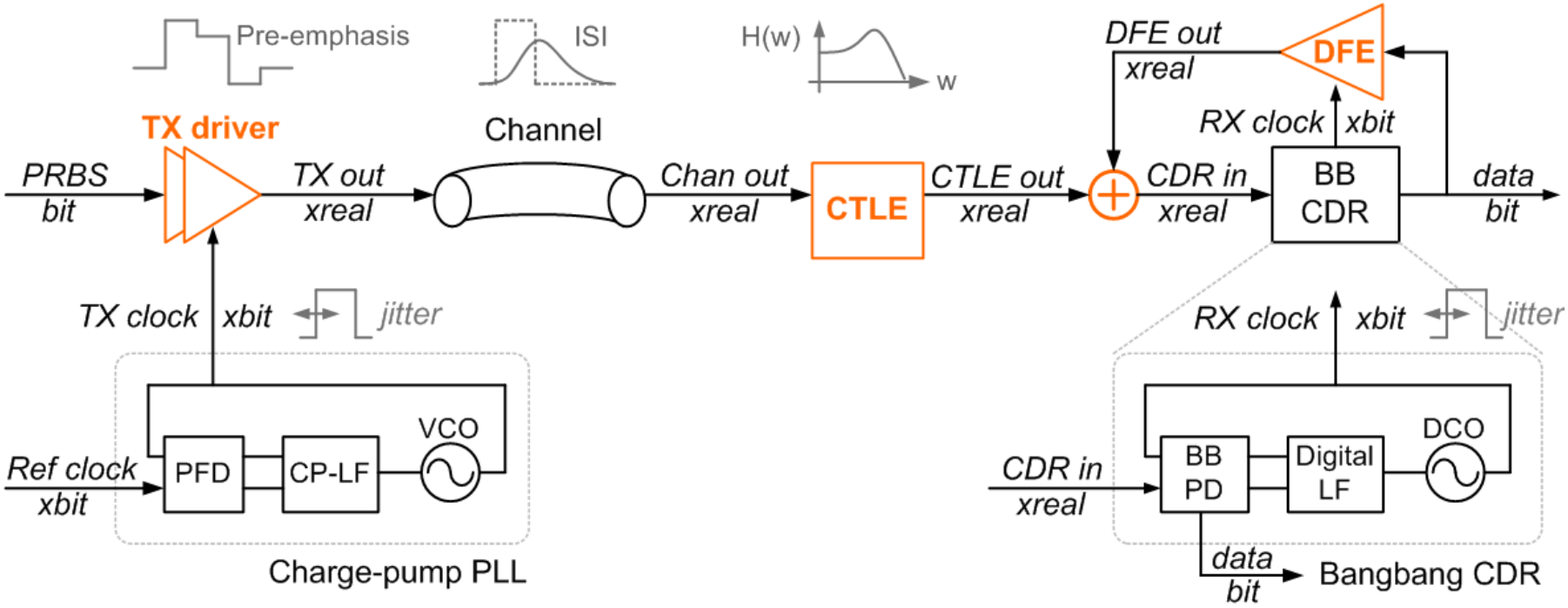


# *GLISTER*: Virtuoso Integration

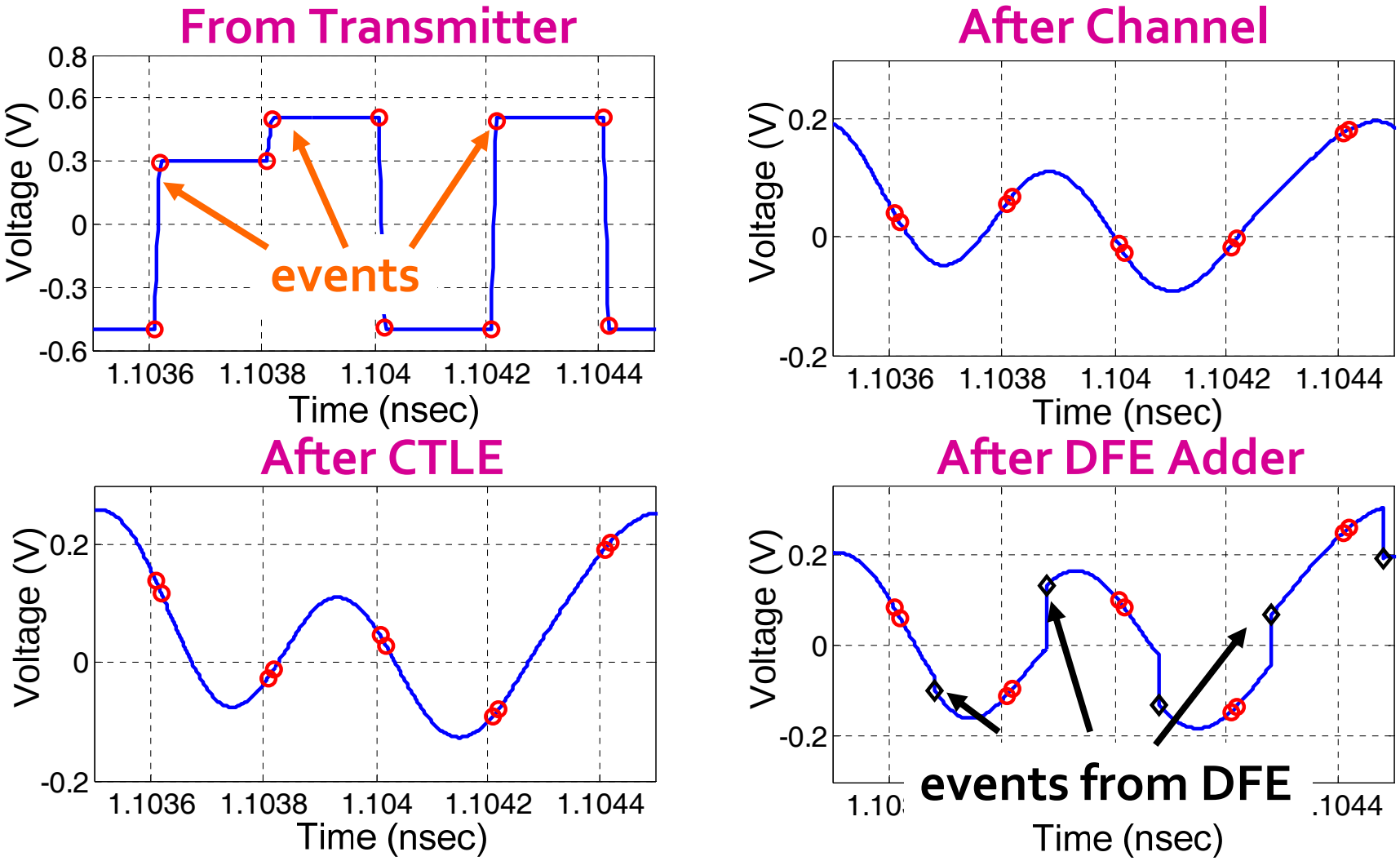


# A High-Speed I/O Link Example

- A high-speed I/O link including:
  - Transmitter, receiver, PLL, and CDR
  - Various equalization schemes: TxEQ, CTLE, DFE

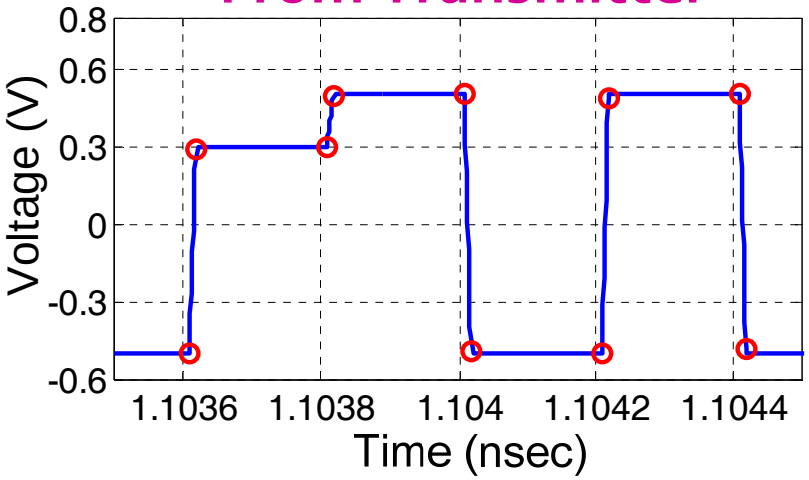


# Simulated Waveforms

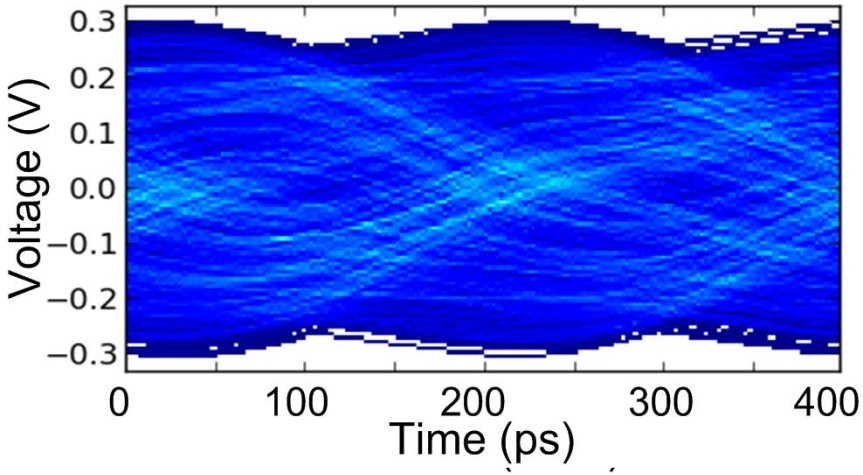


# Simulated Eye Diagrams

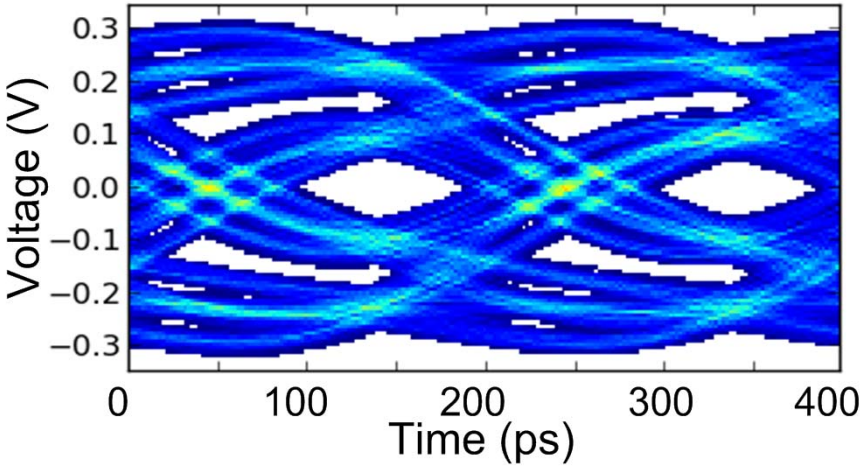
## From Transmitter



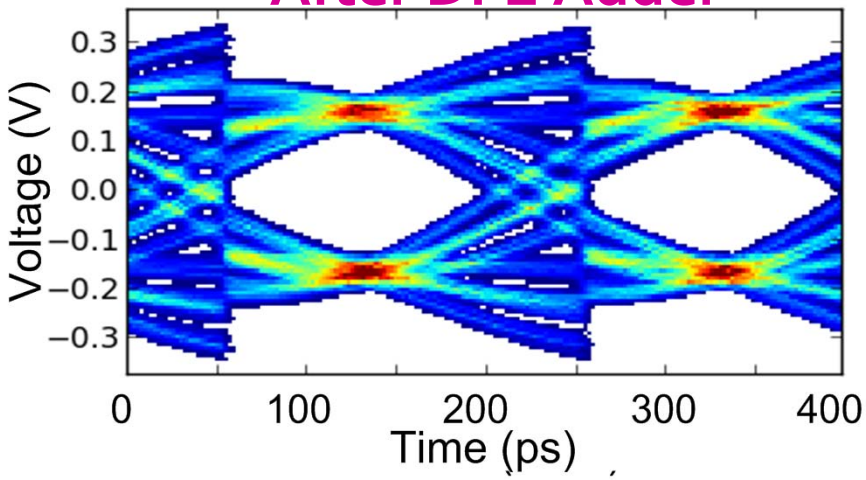

## After Channel




## After CTLE

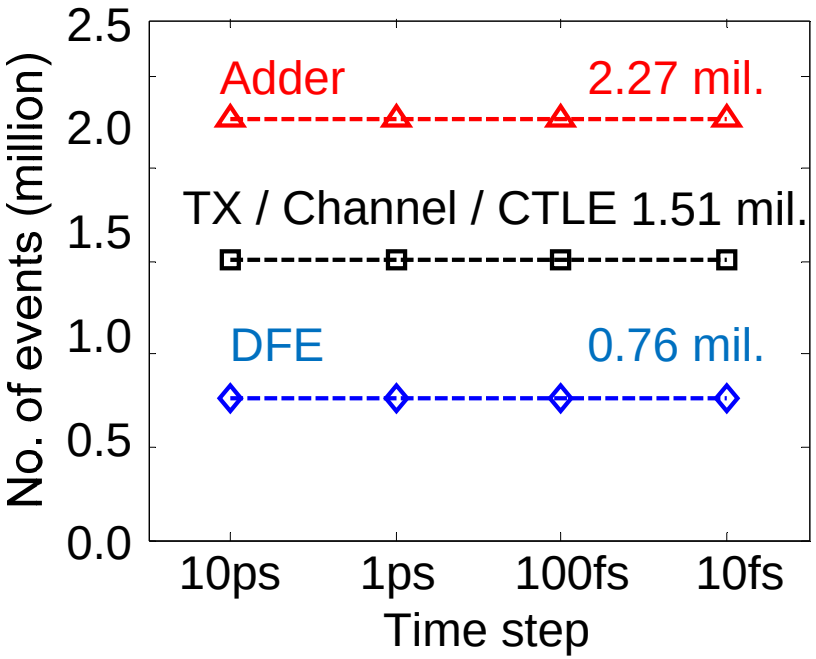


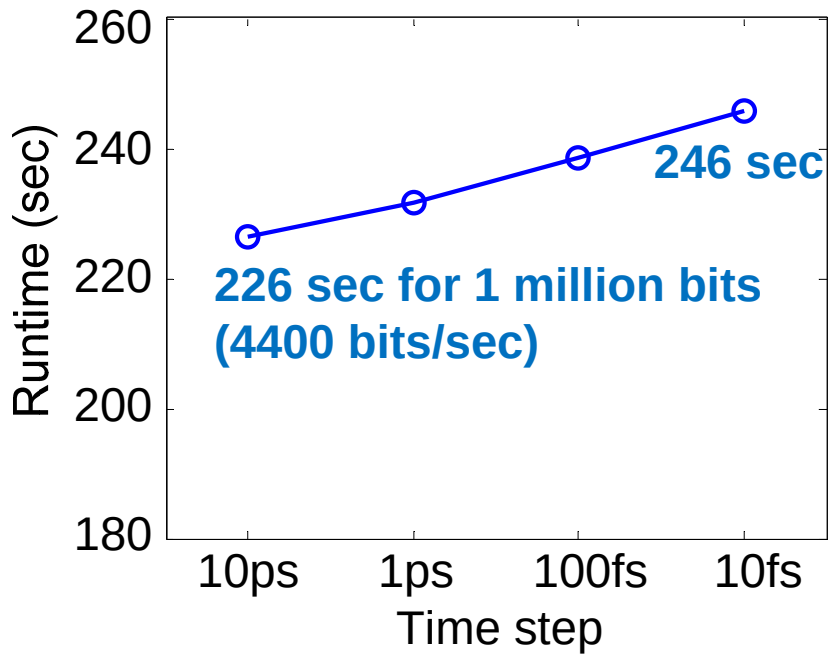

## After DFE Adder



# Speed Independent of Time Step

- The number of events processed and simulation time remain nearly constant even with 1000x change in the time step

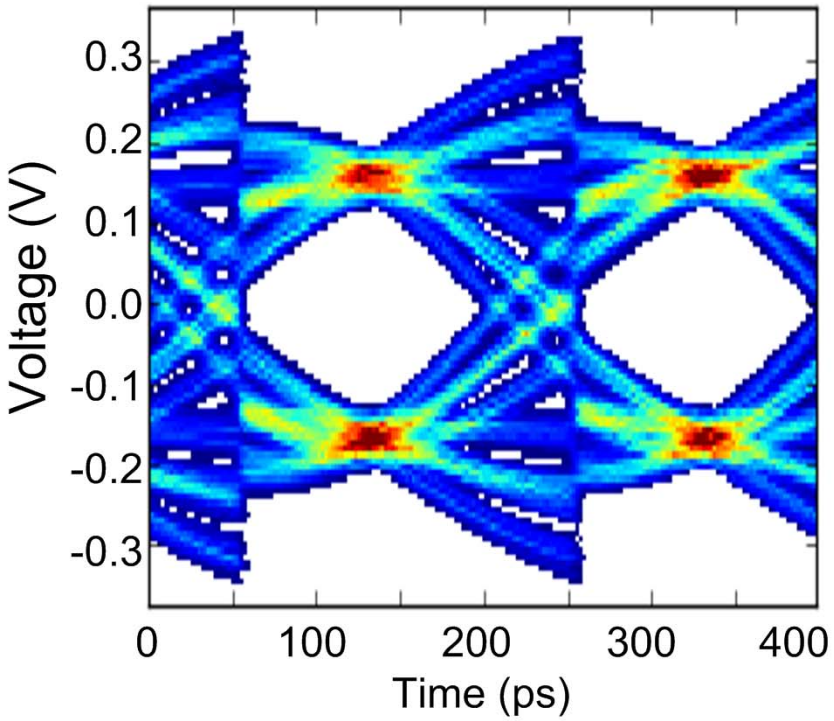


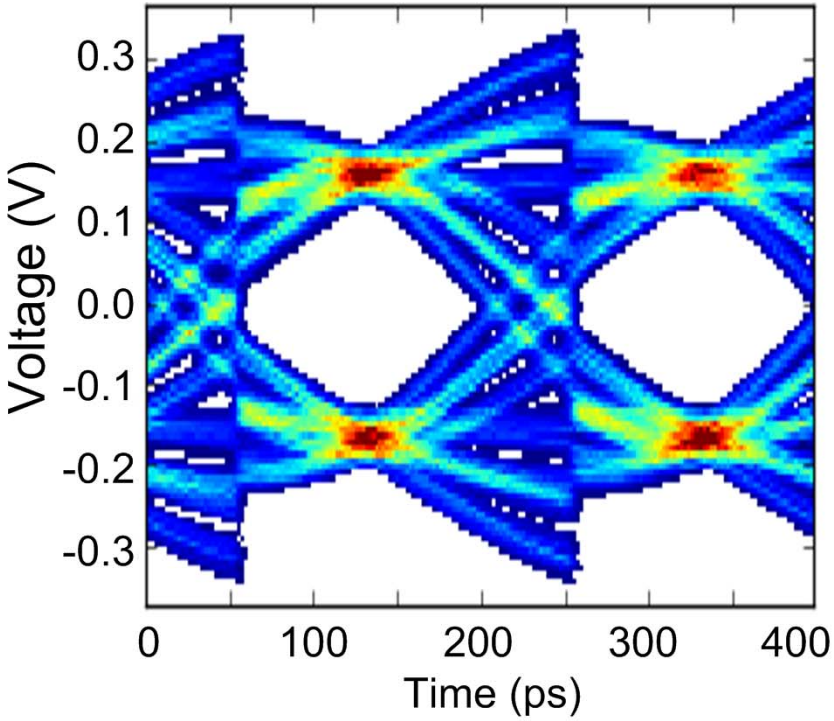
# Accuracy Independent of Time Step

- The shape and quality of the eye diagrams do not depend on the time step

**Time step of 10ps**

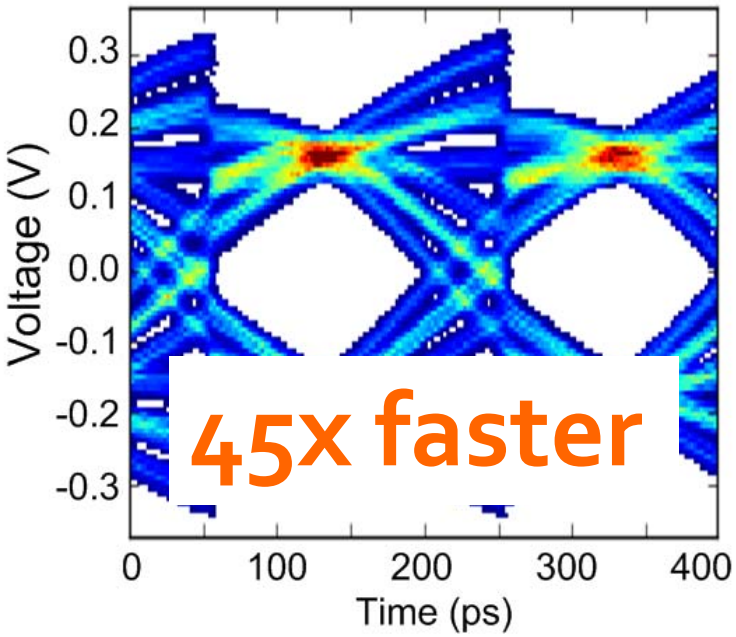



**Time step of 10fs**

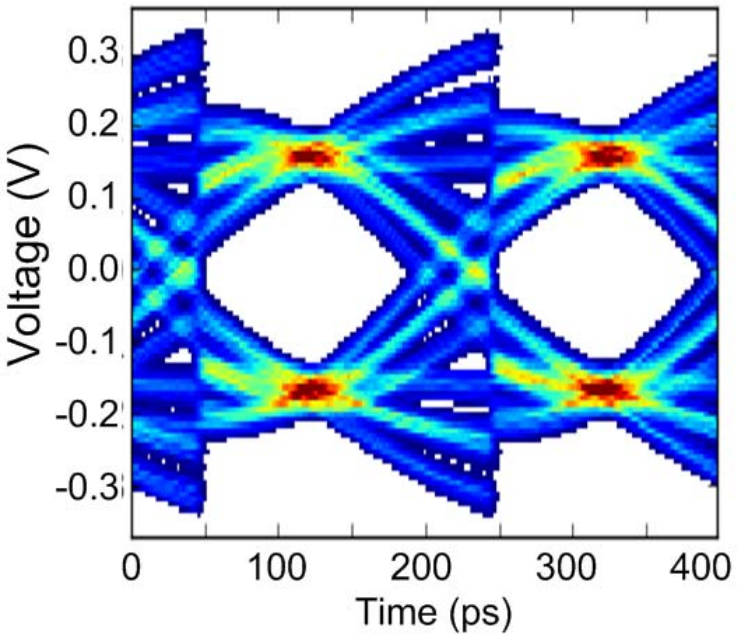


# Comparison with Verilog-A

*XMODEL*




**Verilog-A**




- For 100nsec, the max. and RMS differences are 0.12mV and 0.032mV, respectively

# Comparison with Real-Number Verilog

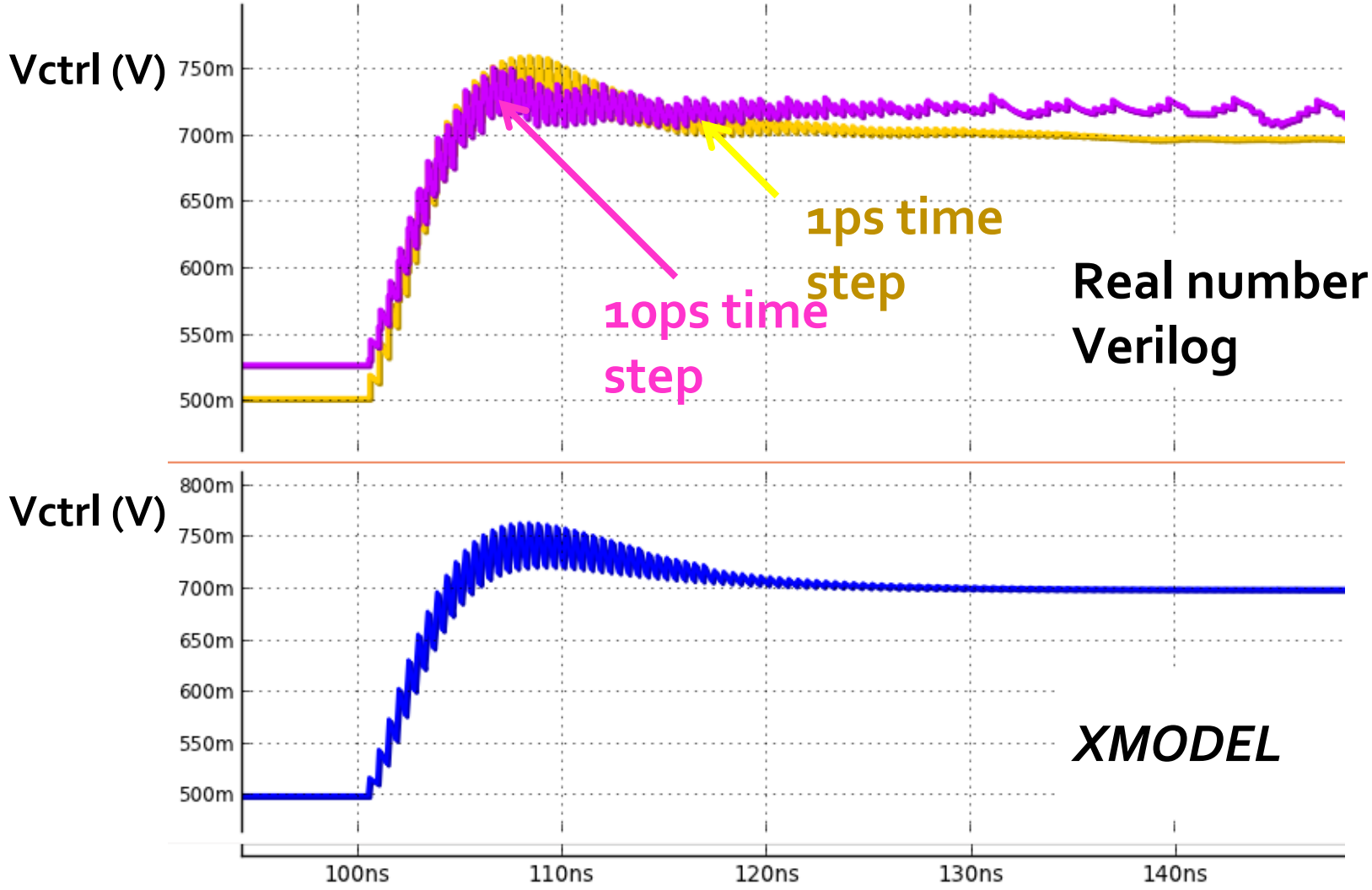
## *XMODEL*

```
`timescale 1ps/1ps
module cp_filter(up, dn, vctrl);
xreal vctrl, i_up, i_dn, Ifilter;

transition #(.value0(0), .value1(I_up))
            cp_up(up, i_up);
transition #(.value0(0), .value1(-I_dn))
            cp_dn(dn, i_dn);
add  cp_out('{i_up, i_dn}, Ifilter);

filter #(.gain(A), .poles(0, p1), .zeros(z1))
      lp_filter (.in(Ifilter), .out(vctrl));

endmodule
```

## Real-Number Verilog

```
`timescale 1ps/1ps
module cp_filter(up, dn, vctrl);
real vctrl, vctrl_post, u_post, Ifilter;
real t_step = 1e-12;

always begin
   #1;
   Ifilter = up*Iup – dn*Idn;
   vctrl = (A*(u_post + t_step*Ifilter
            + 1/z1 * Ifilter)
            + 1/p1 *t_step *vctrl_post)
              / (1 + 1/p1 * t_step) ;
   vctrl_post = vctrl;
   u_post = u_post + t_step* Ifilter;
end
endmodule
```

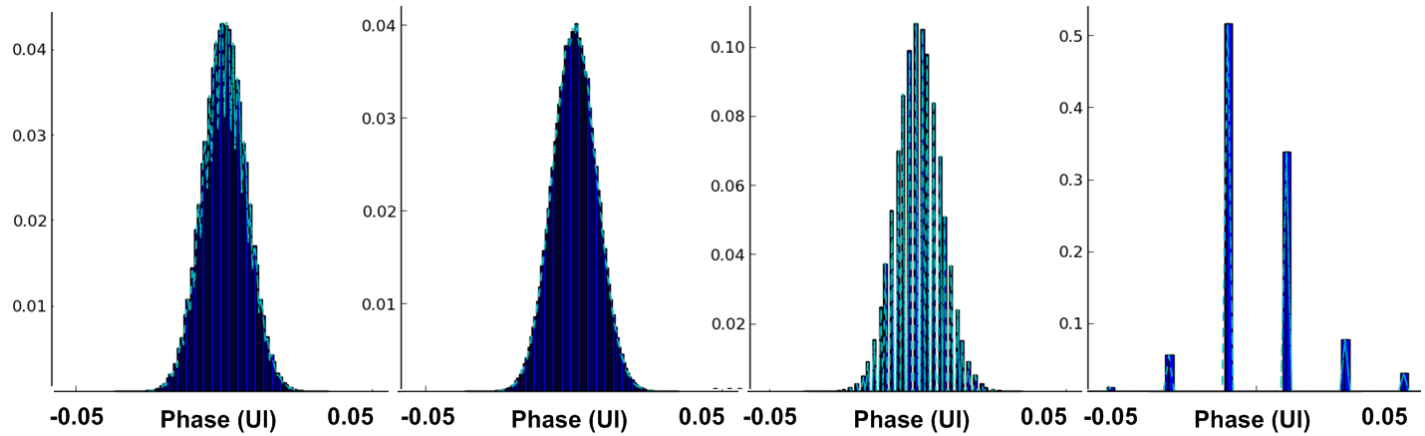
# Charge-Pump PLL Locking Transients



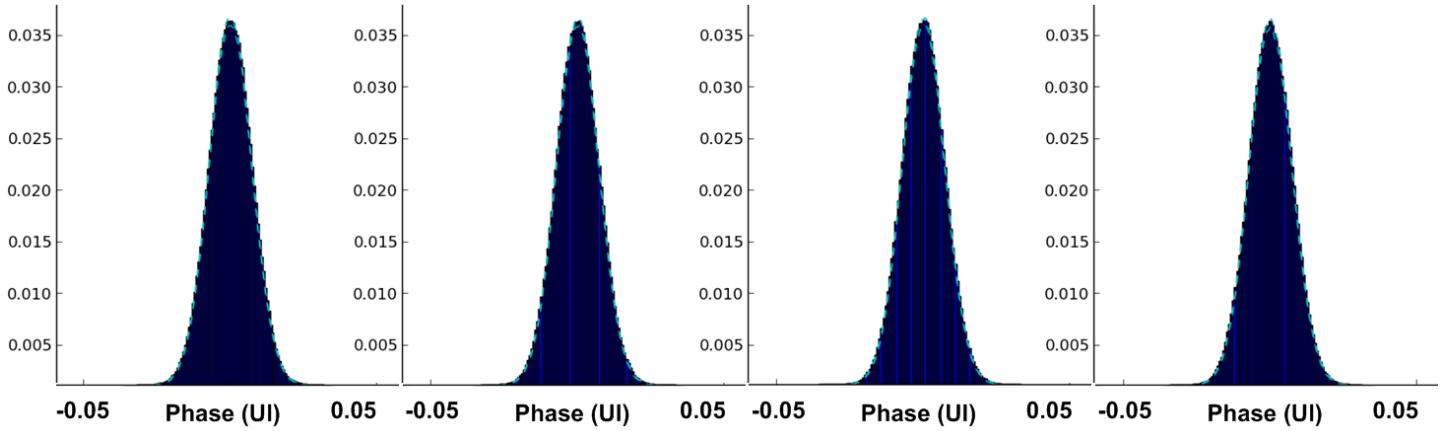
# Charge-Pump PLL Jitter Histogram

**Time Step** 0.01ps 0.1ps 1ps 10ps

**Real-Number Verilog**

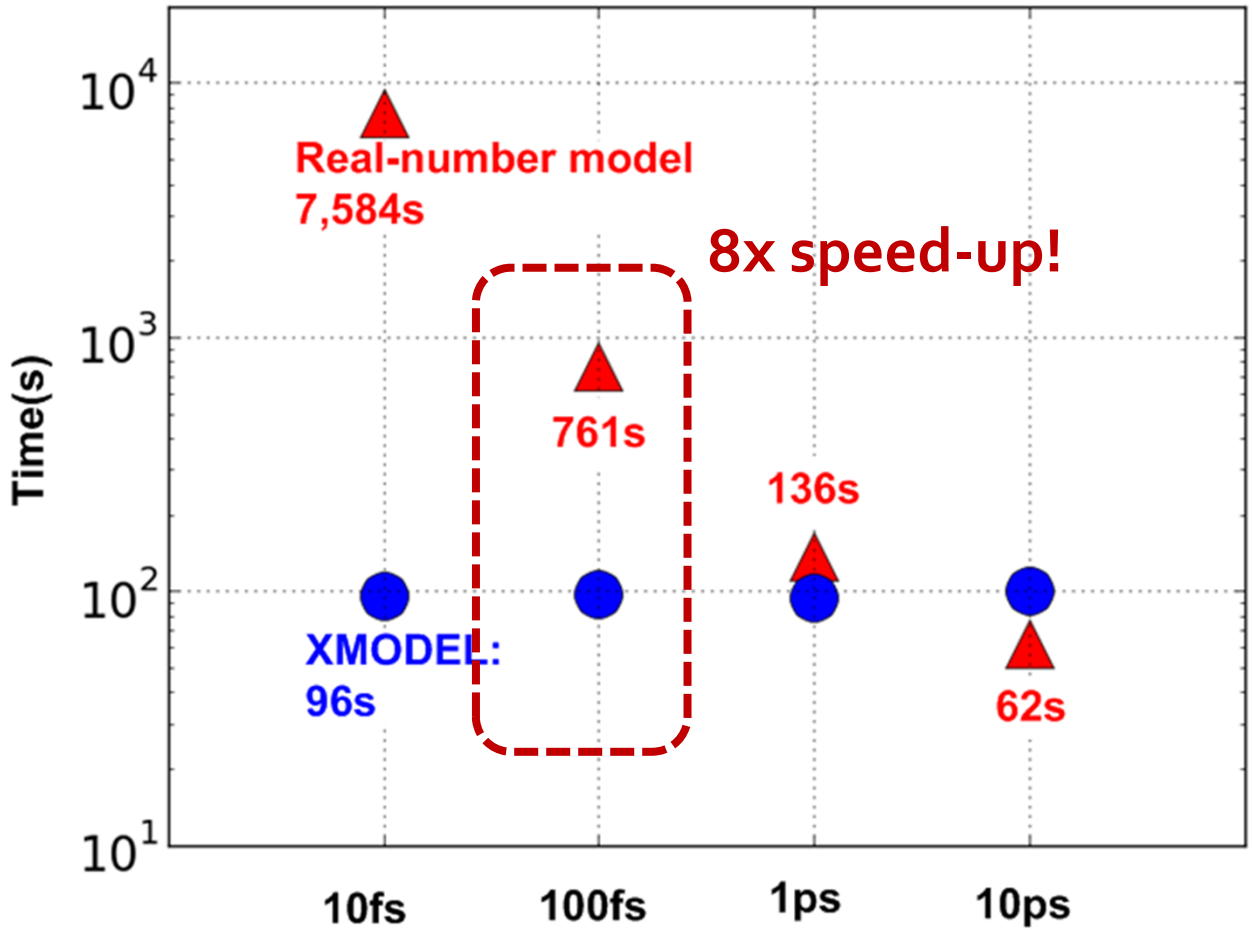



***XMODEL***



# Simulation Speed Comparison

- Execution time of $10^6$ clock cycle simulation

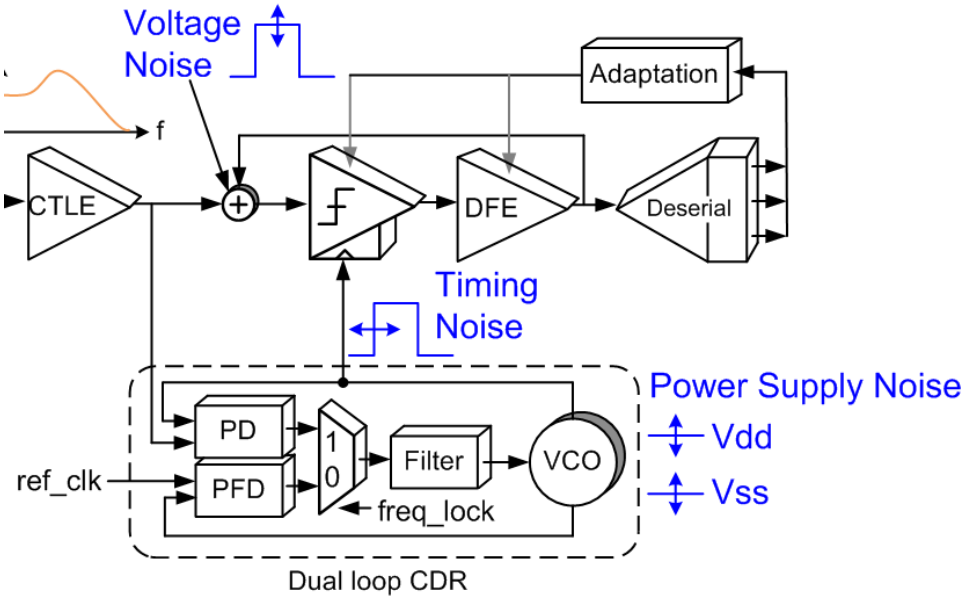


# Comparison with Other Simulators

| | *XMODEL* | Real-Number Modeling | Verilog-A | Simulink |
|---|---|---|---|---|
| **Simulation Algorithm** | **Event-Driven** | **Fixed Time-Step** | Variable Time-Step ODE | Variable Time-Step ODE |
| **Simulation Platform** | **SystemVerilog** | SystemVerilog or Verilog-AMS | **SPICE** | Matlab |
| **Speed** | **Fastest (x50)** | Moderate (x5) | **Slow (x1)** | Fast (x25) |
| **Accuracy** | **High regardless of speed** | **Degrades fast with speed** | Degrades with speed | Degrades with speed |
| **Modeling Capability** | **Functional & Circuit-level** | **Functional only** | Functional & Circuit-level | Functional only |
| **Modeling Difficulty** | **Easy** (GLISTER, MODELZEN, ...) | **Moderate** (Arana) | **Moderate** (SMG, Arana) | Easy (MATLAB) |

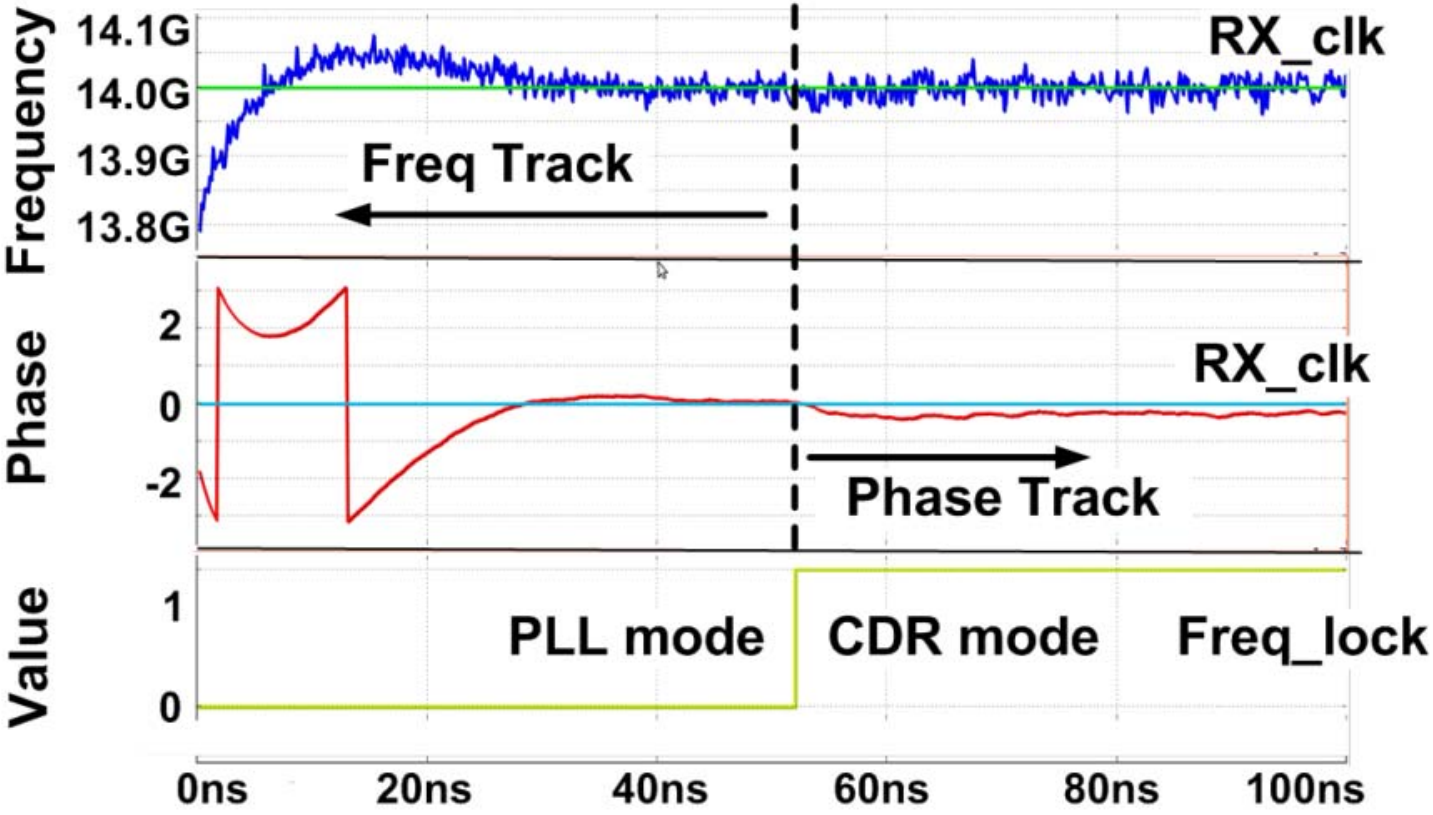
# More System Examples

- *XMODEL* is a powerful tool to verify your digital controller and calibration engines in context of analog/mixed-signal systems
  - Timing calibration, offset calibration, EQ adaptation, …
  - *XMODEL* can model analog parts accurately while retaining the speed
- Example:
  - CDR frequency acquisition loop
  - DFE EQ adaptation loop



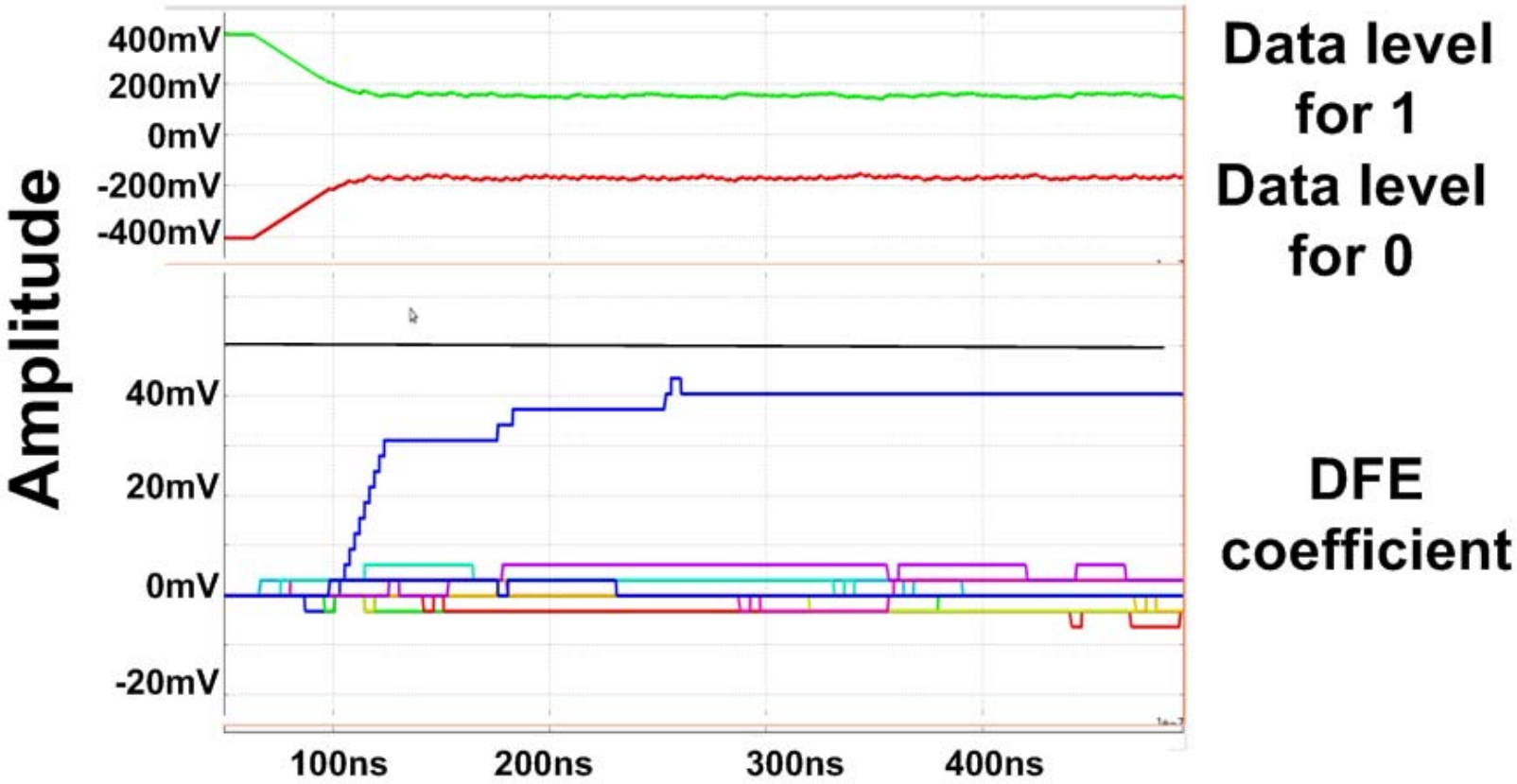
# Clock and Data Recovery (CDR)

- CDR first acquires frequency by operating as a PLL
- CDR then starts tracking phase of the incoming data



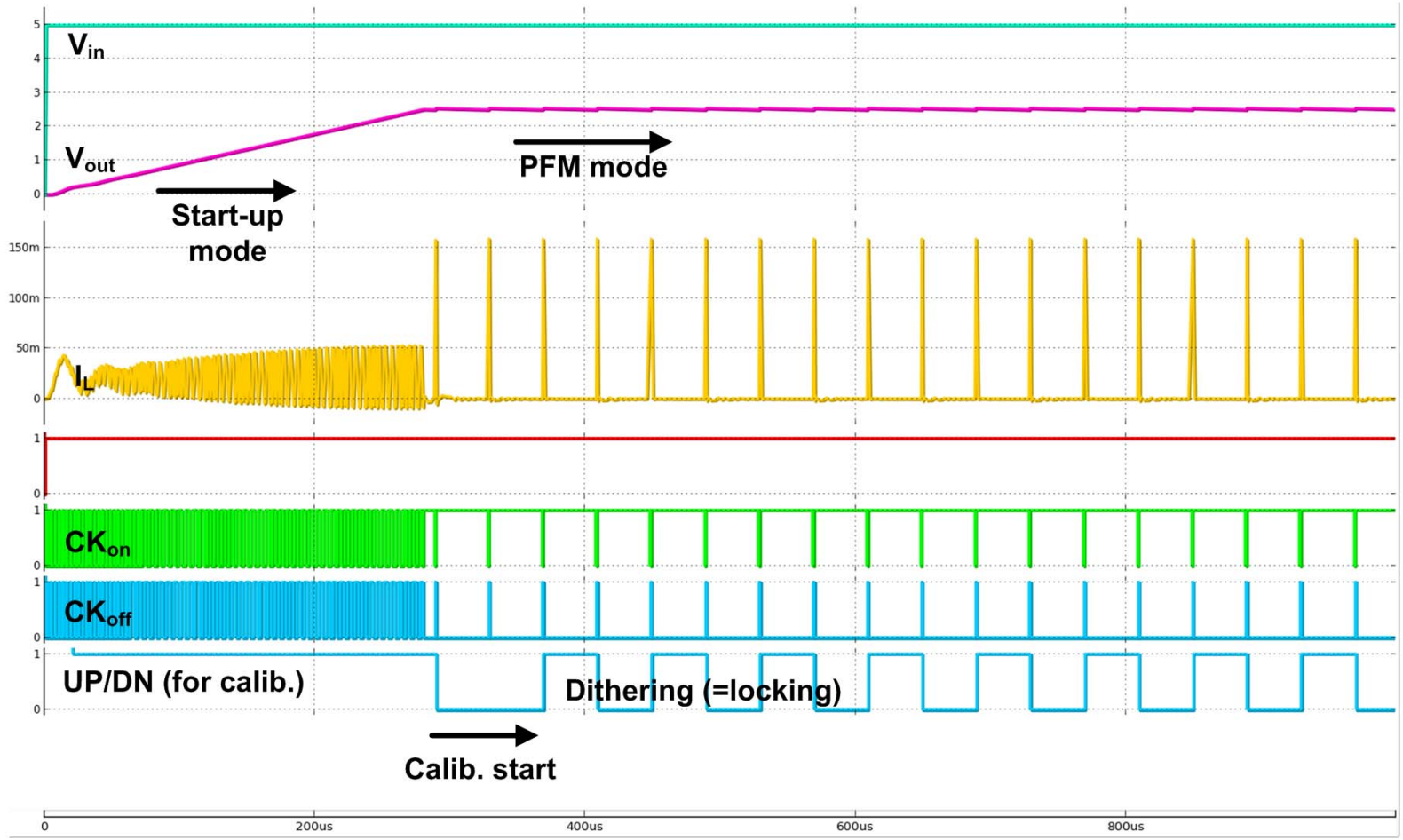
# Adaptive Decision Feedback Equalizer

- DFE tap coefficients and data levels are trained using sign-sign LMS algorithm

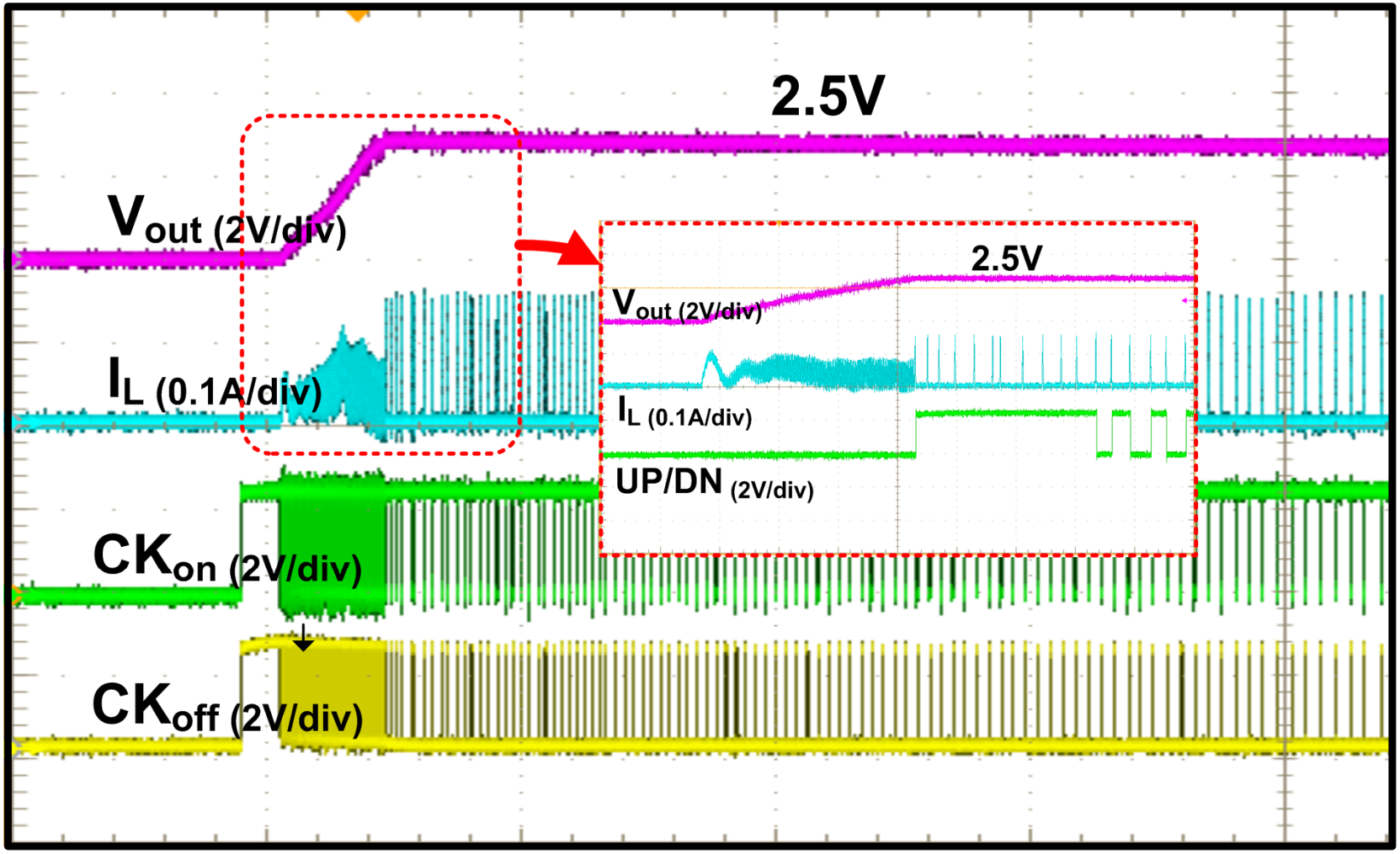


# PFM Buck Regulator Simulation

- 1-ms simulation takes only 2.4 seconds!

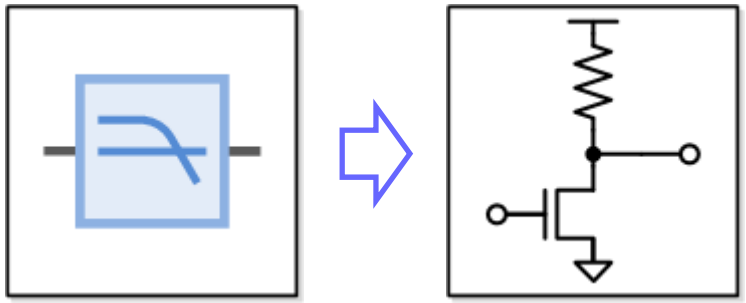


# Comparison with Measurement

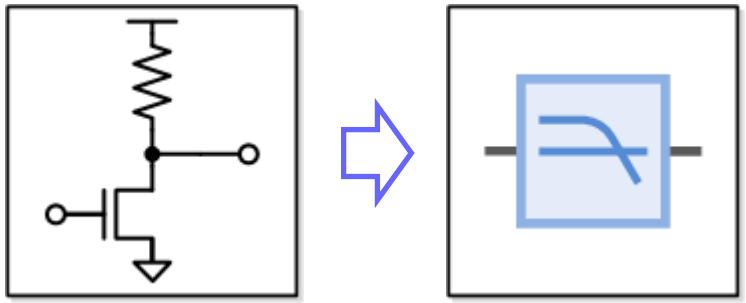


# Writing Models with *XMODEL*

- *XMODEL* supports both top-down and bottom-up modeling
- **Top-down modeling**
  - From model to circuit
  - e.g. explore different architectures before circuit implementation

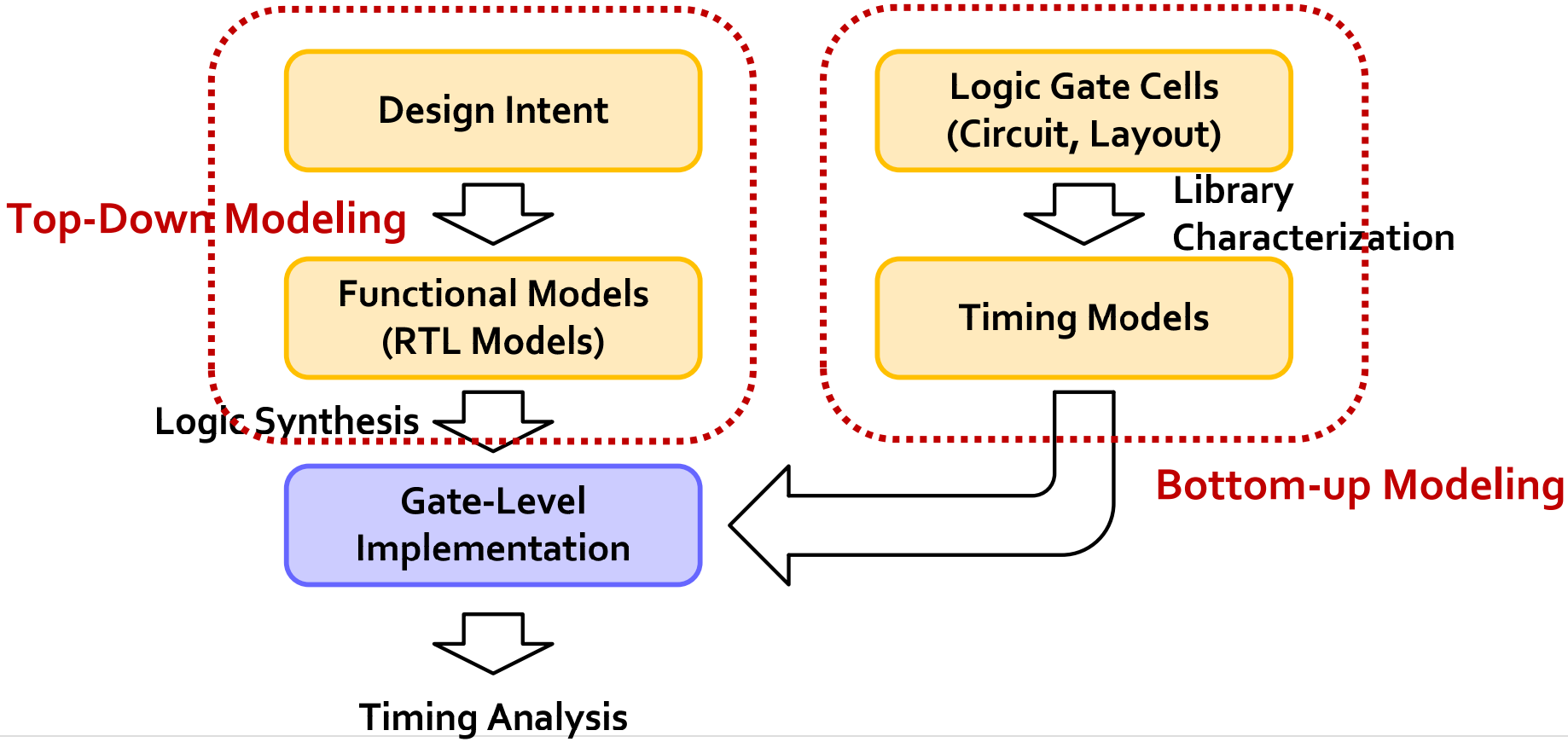


- **Bottom-up modeling**
  - From circuit to model
  - e.g. use models instead of circuits to enable efficient top-level verification



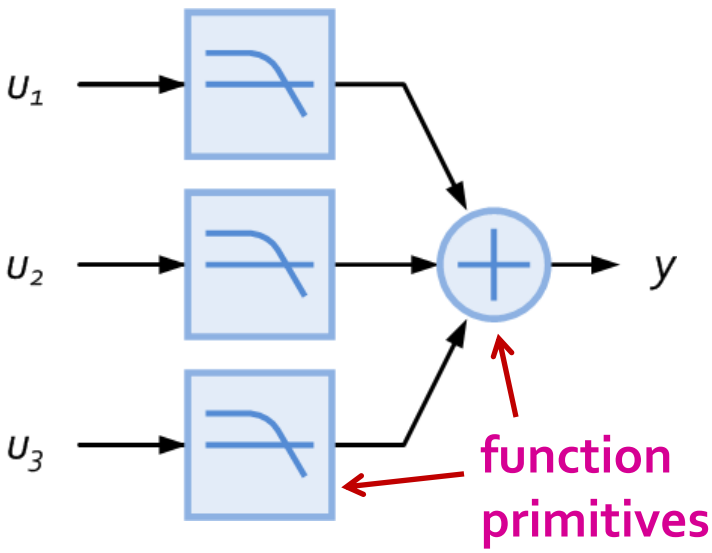
# Top-Down and Bottom-Up in Digital

- Top-down models capture the intended functionality
- Bottom-down models capture the real characteristics



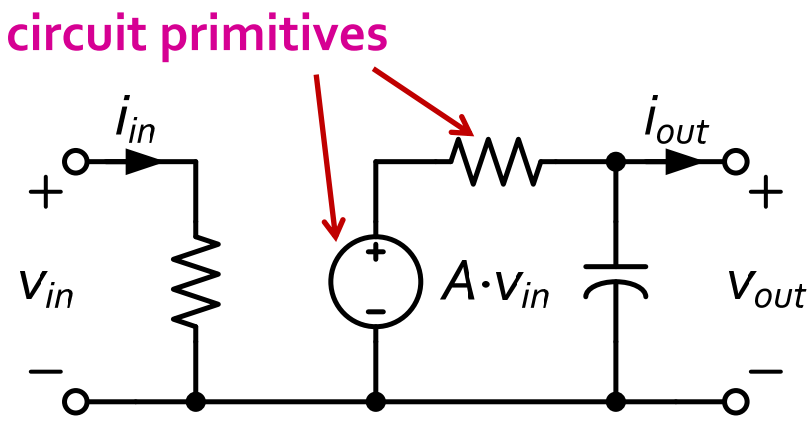
# Block-Level vs. Circuit-Level Models

**Block-Level Model**
(Signal-flow Model)




- A network of blocks where signals flow in one direction only

**Circuit-Level Model**
(Conservative System Model)

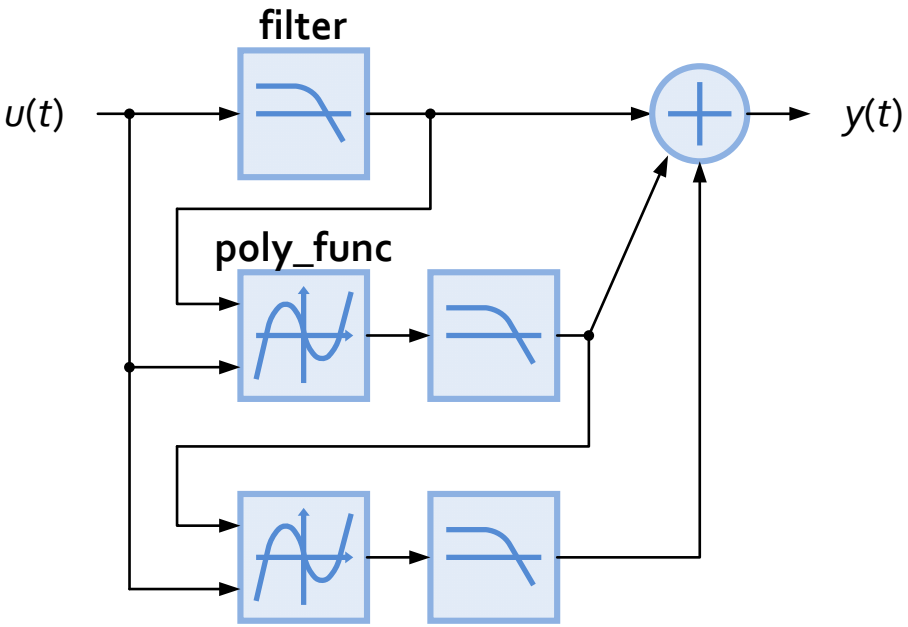



- A network of circuits whose state is described by voltages & currents
- e.g. loading effects

# BLM of Nonlinear Systems

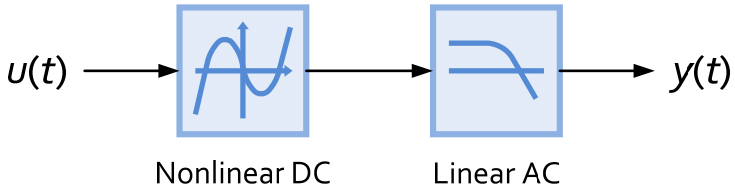
- With *XMODEL* primitives, you can construct various nonlinear models that simulate in event-driven way

**Volterra Series Model**

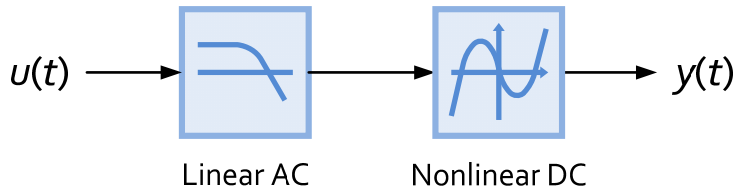



[Roychowdhury,1999]

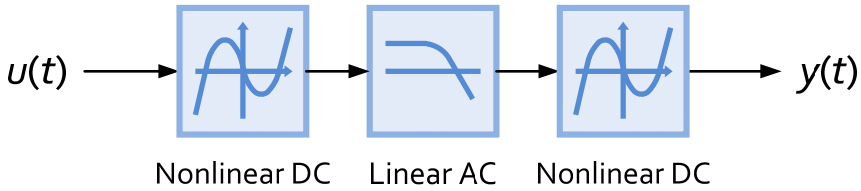
**Hammerstein Model**




**Wiener Model**




**Hammerstein-Wiener Model**

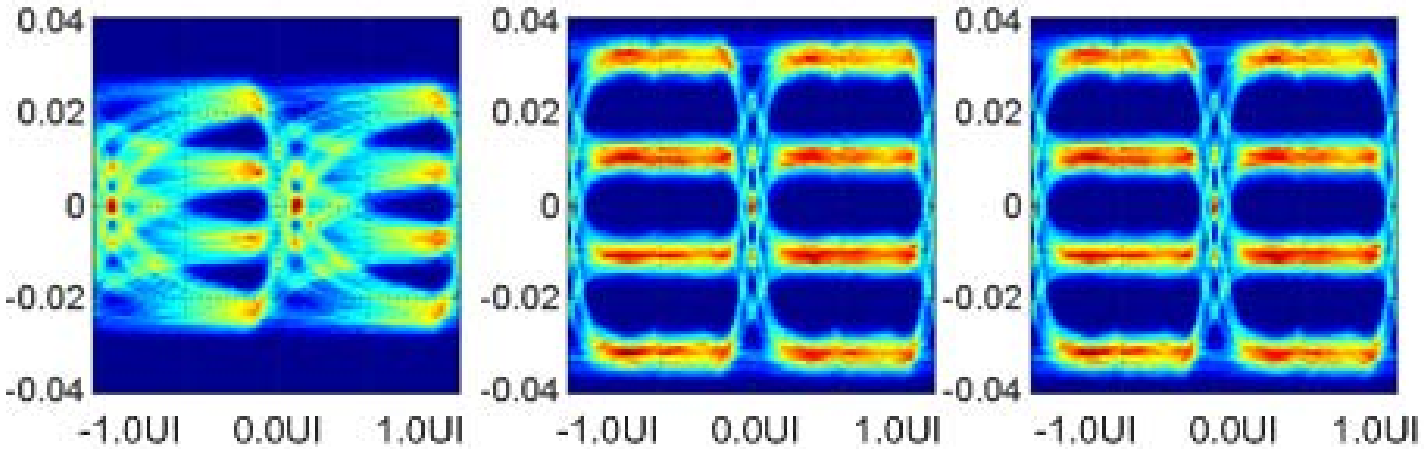


# Nonlinear CTLE Simulation Results

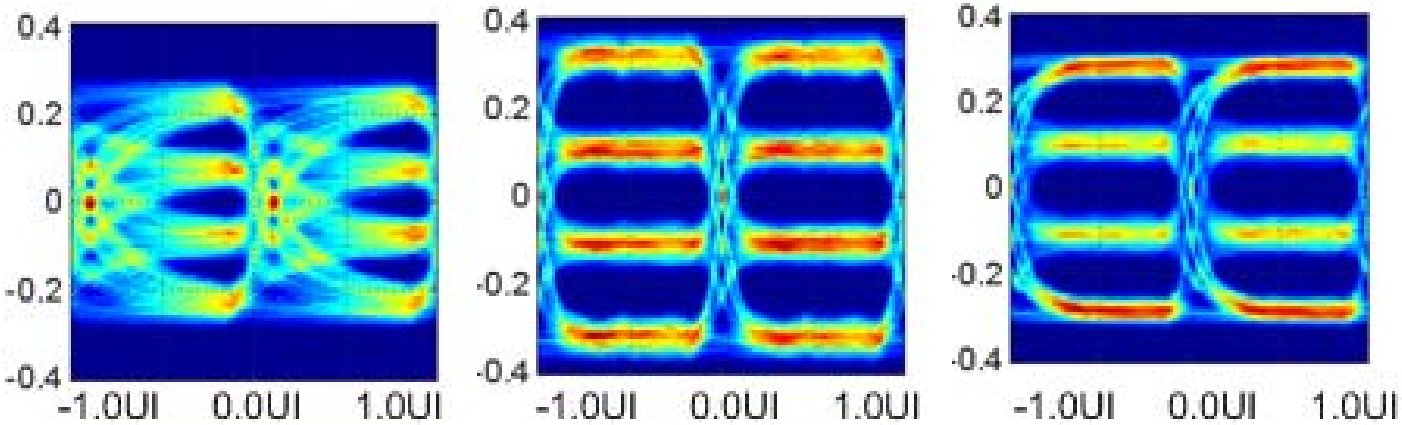
[Jang'13]

**Channel output** **CTLE output (1st)** **CTLE output (1st+3rd)**

**Signal swing: ±30mV$_{dpp}$**

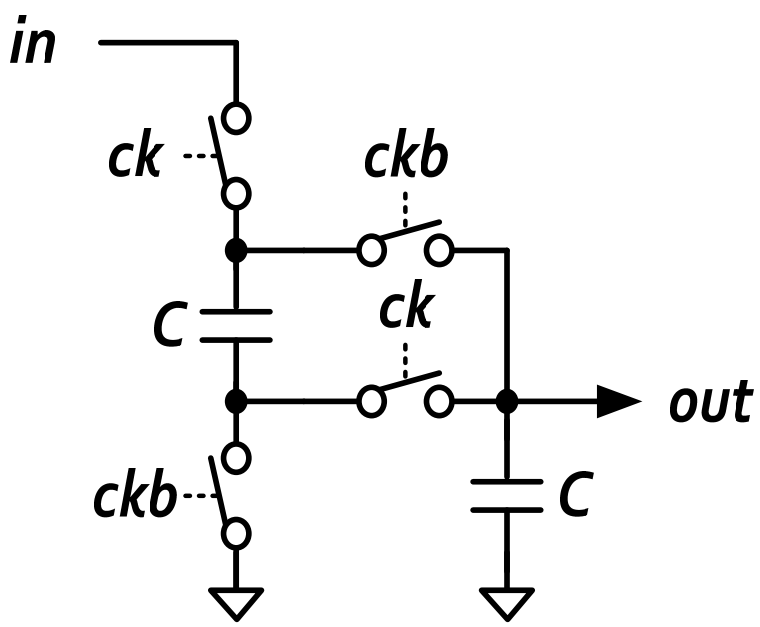



**Signal swing: ±300mV$_{dpp}$**



# Circuit-Level Modeling (CLM)

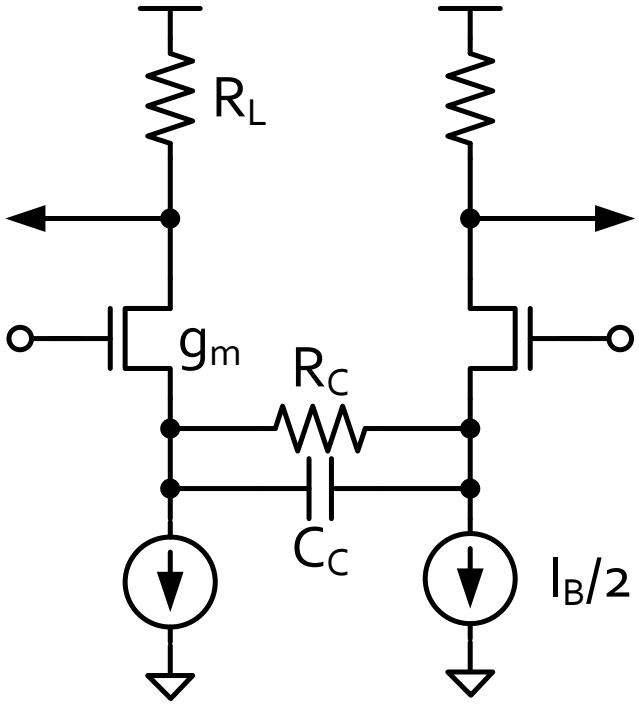
- The CLM feature in *XMODEL* allows you to list circuit elements directly in SystemVerilog!
- *XMODEL* can simulate CLM models in SystemVerilog in event-driven fashion without invoking SPICE




```
module sc_converter(
      input xreal in,
      output xreal out,
      input xbit ck, ckb
);
xreal      n1, n2;
switch     sw1(.pos(in), .neg(n1), .ctrl(ck));
switch     sw2(.pos(n1), .neg(out), .ctrl(ckb));
switch     sw3(.pos(n2), .neg(out), .ctrl(ck));
switch     sw4(.pos(n2), .neg(`ground), .ctrl(ckb));
capacitor  #(.C(1e-12)) C1(.pos(n1), .neg(n2));
capacitor  #(.C(1e-12)) C2(.pos(n2), .neg(`ground));
endmodule
```

# CLM for Nonlinear Modeling

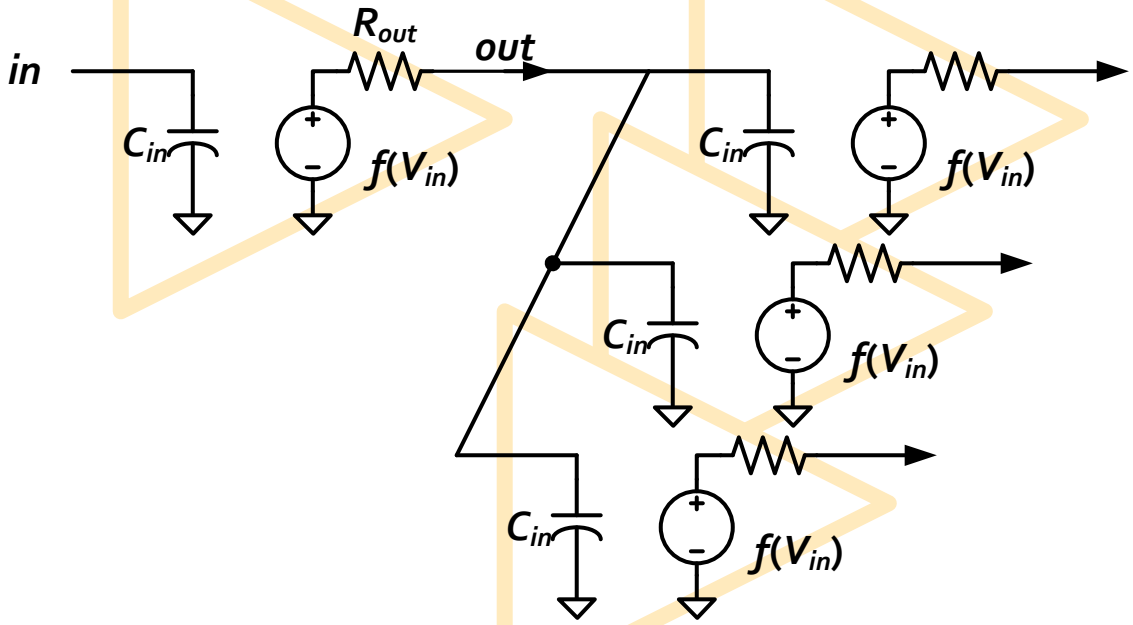
- Using the nonlinear CLM primitives such as transistors is an easy way to model nonlinear circuits




```
module ctle (
    `input_xreal inp, inn,          // input signals
    `output_xreal outp, outn        // output signals
);

xreal sp, sn;
xreal vdd;

vsource     #(.mode("dc"), .dc(Vdd))
            V1(.pos(vdd), .neg(`ground), .in(`ground));
isource     #(.mode("dc"), .dc(Ib/2))
            I1(.pos(sp), .neg(`ground), .in(`ground));
            I2(.pos(sn), .neg(`ground), .in(`ground));

nmosfet      #(.Kp(Gm), .Vth(Vth))
            M1(.d(outn), .g(inp), .s(sp), .b(`ground)),
            M2(.d(outp), .g(inn), .s(sn), .b(`ground));

resistor    #(.R(Rload))
            RL1(.pos(vdd), .neg(outp)),
            RL2(.pos(vdd), .neg(outn));
capacitor   #(.C(Cload))
            CL1(.pos(vdd), .neg(outp)),
            CL2(.pos(vdd), .neg(outn));
resistor    #(.R(Rc))   RC1(.pos(sp), .neg(sn));
capacitor   #(.C(Cc))   CC1(.pos(sp), .neg(sn));

endmodule
```

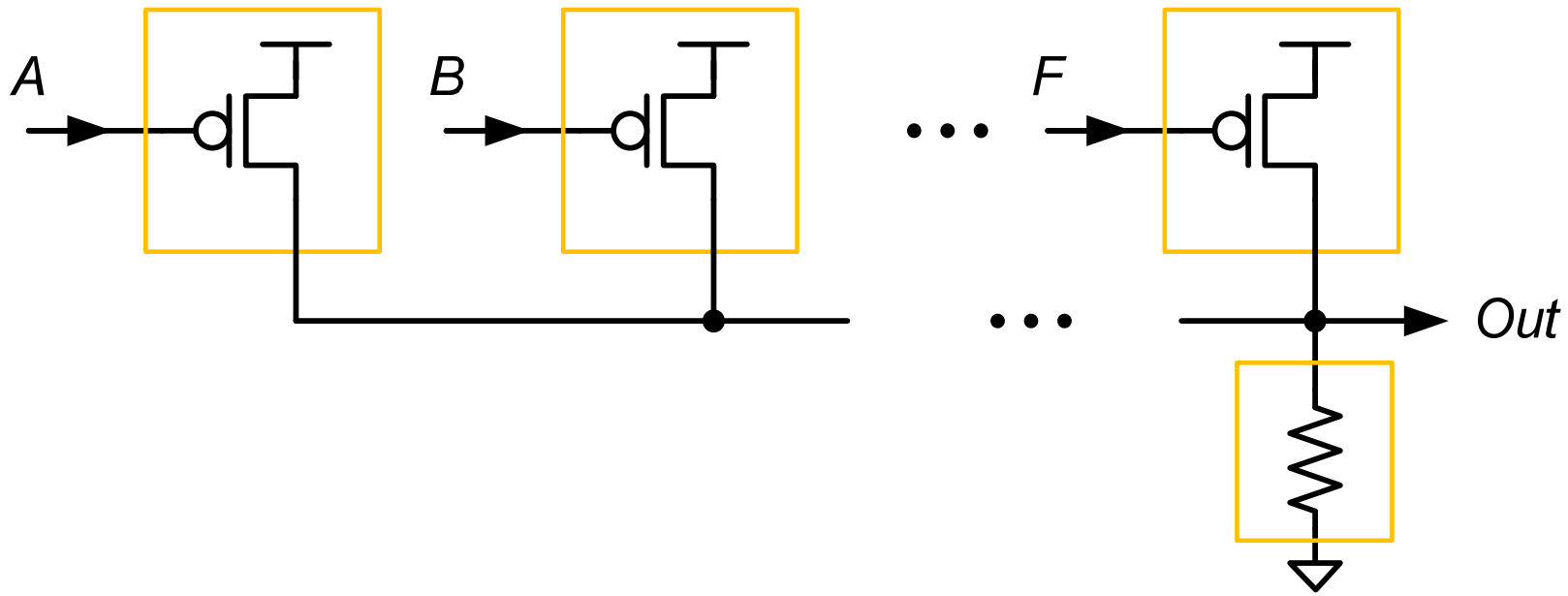
# CLM for Loading Effects

- CLM is the most natural way to model *loading effects*
- e.g. delay varying with $C_{load}$, $V_{out}$ changing with $I_{load}$, …



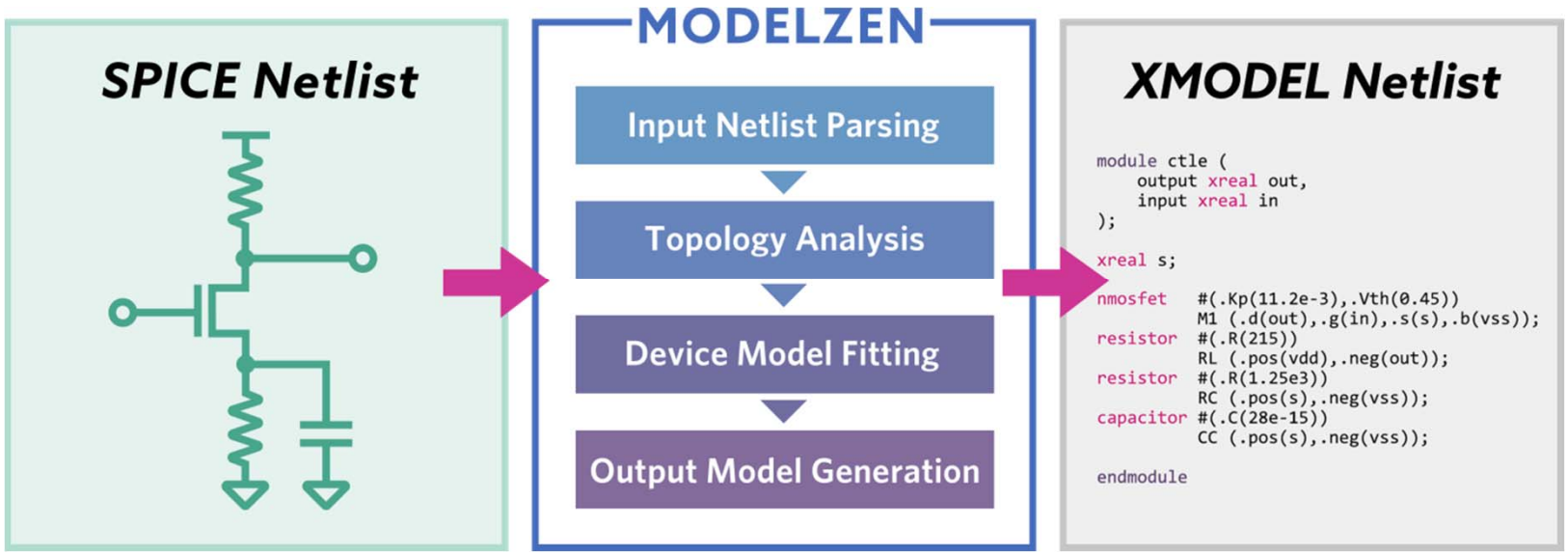
# CLM for Pin-Compatible Modeling

- CLM allows you to use the same cell partitions and pin interfaces between circuits and models
  - e.g. multiple open-drain driver cells connected to a single R
- No more awkward *wreal resolution functions*!

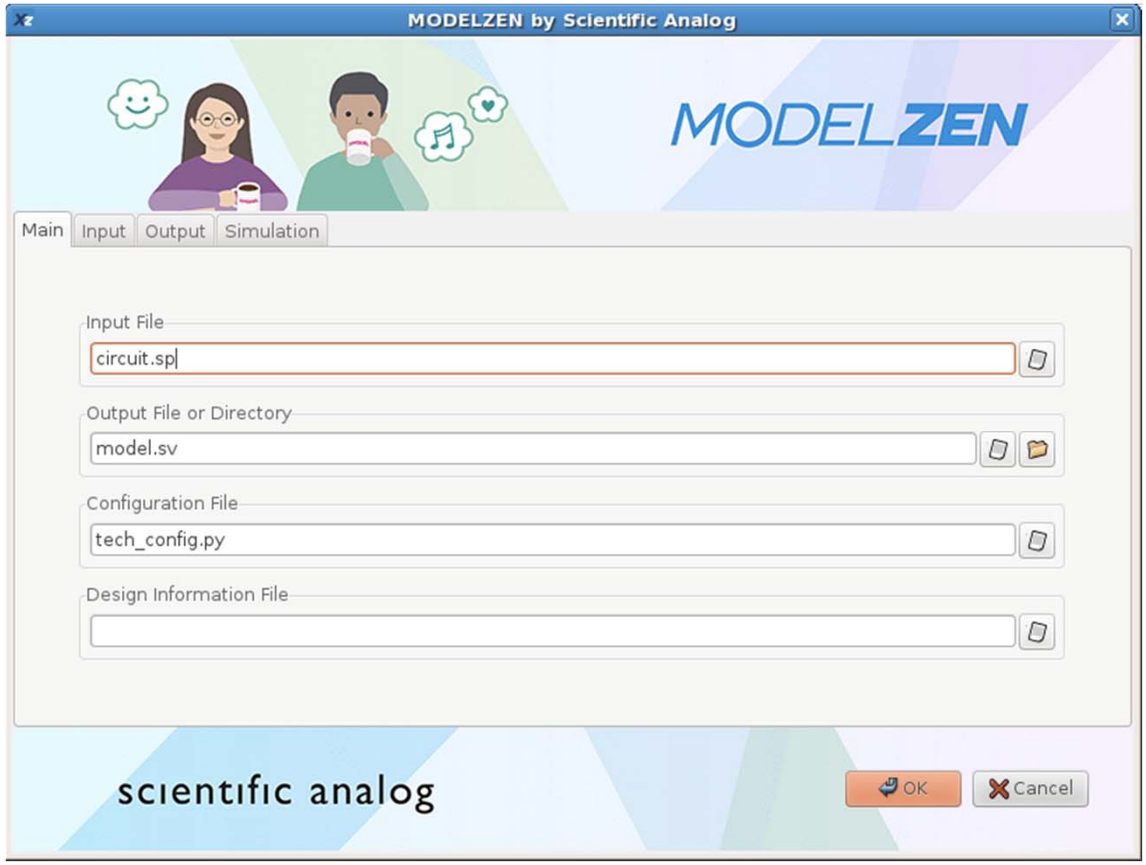


# Structural Model Generation

- ***MODELZEN*** can automatically generate structural models from SPICE netlists
  - Individual device parameters are calibrated via SPICE sim.
  - Useful for bottom-up modeling of complex analog circuits
  - The extracted models also simulate in an event-driven way



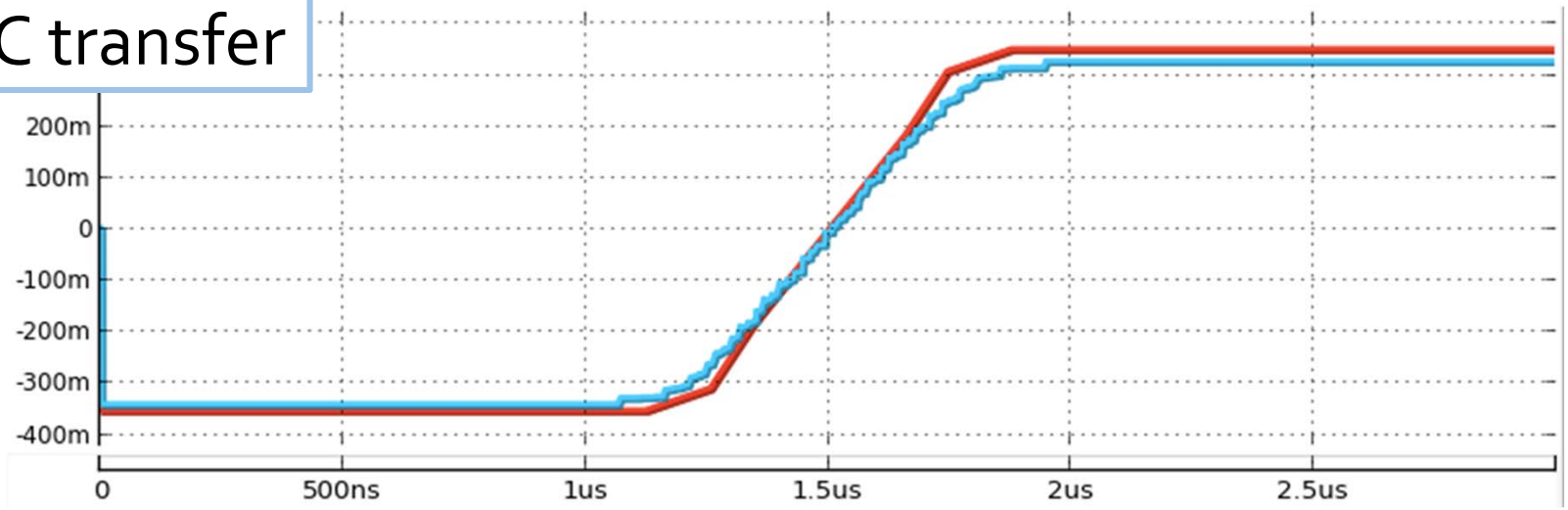
# MODELZEN



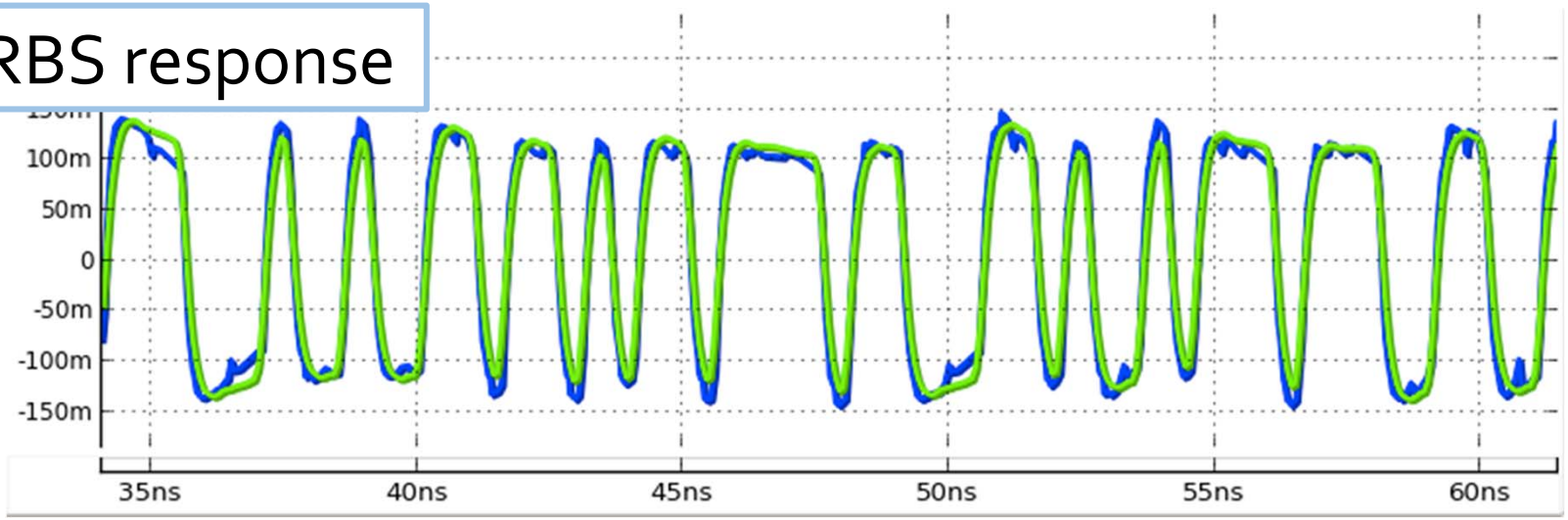

- Both in GUI and in command-line
- Launches a set of SPICE simulations to characterize devices
- Configuration file defines device mapping for tech.
- Multithreading support

# CTLE Simulation Results

DC transfer

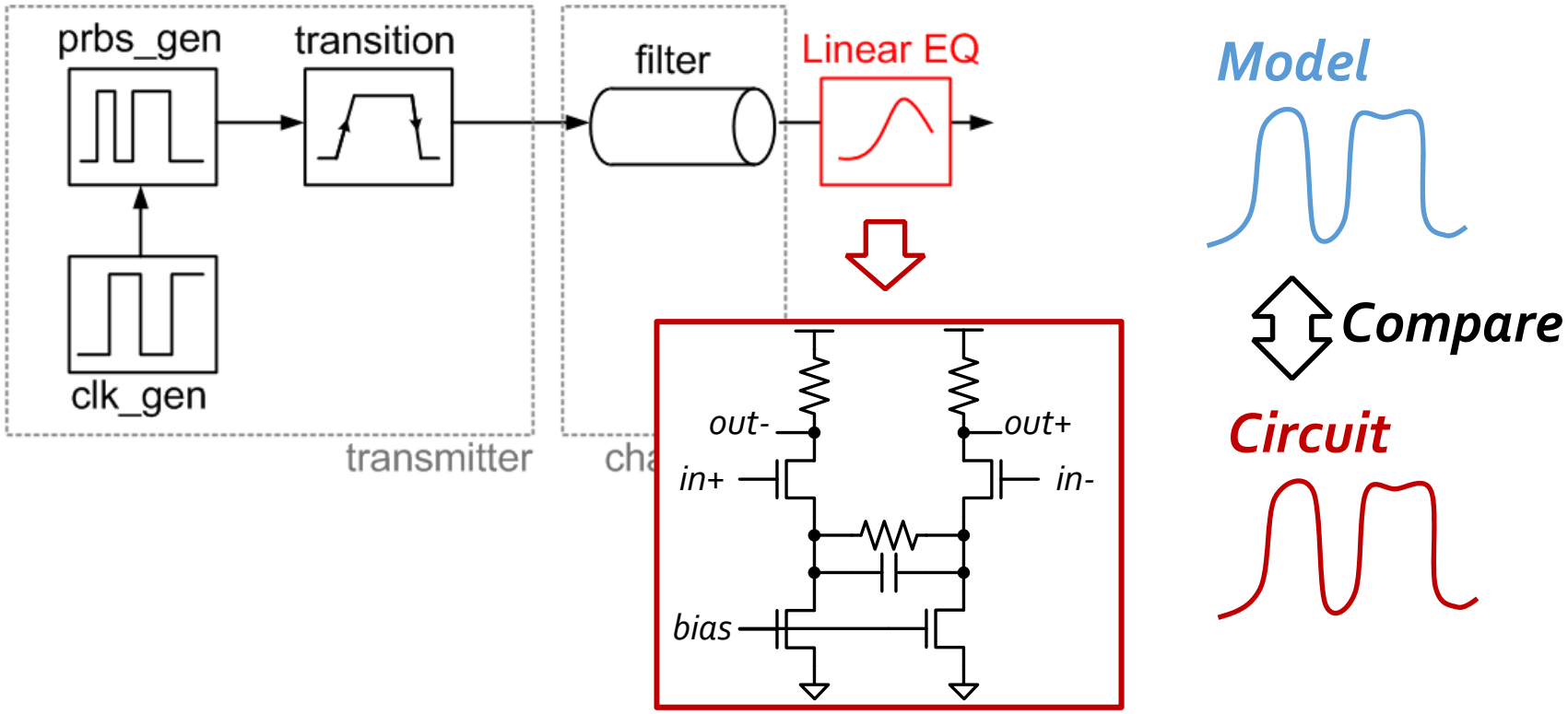



PRBS response



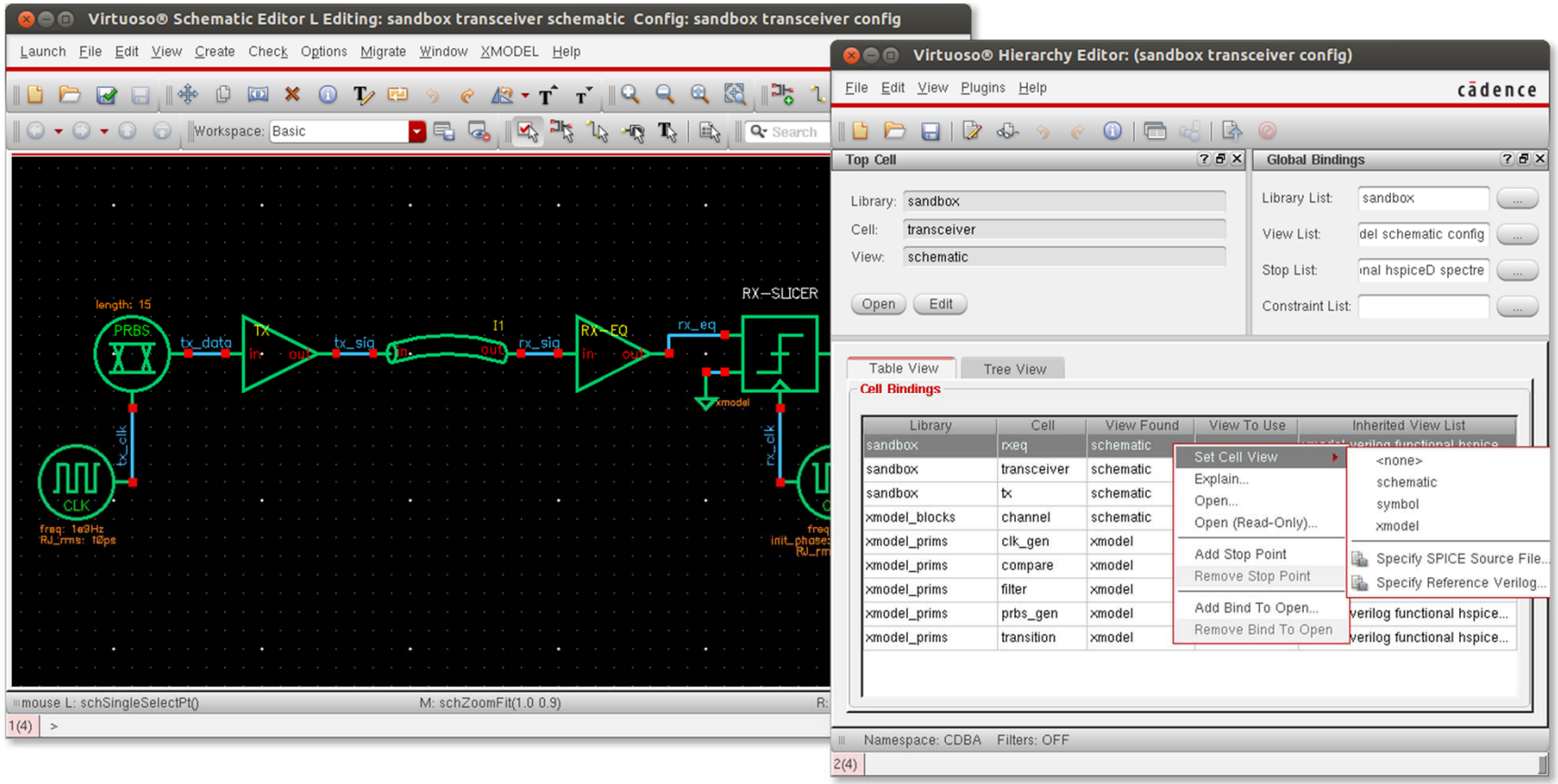
# Model Verification via Co-Simulation

- One way to verify model-circuit equivalence is to run the same *XMODEL* testbench on them and compare



# Generating Co-Simulation Netlist

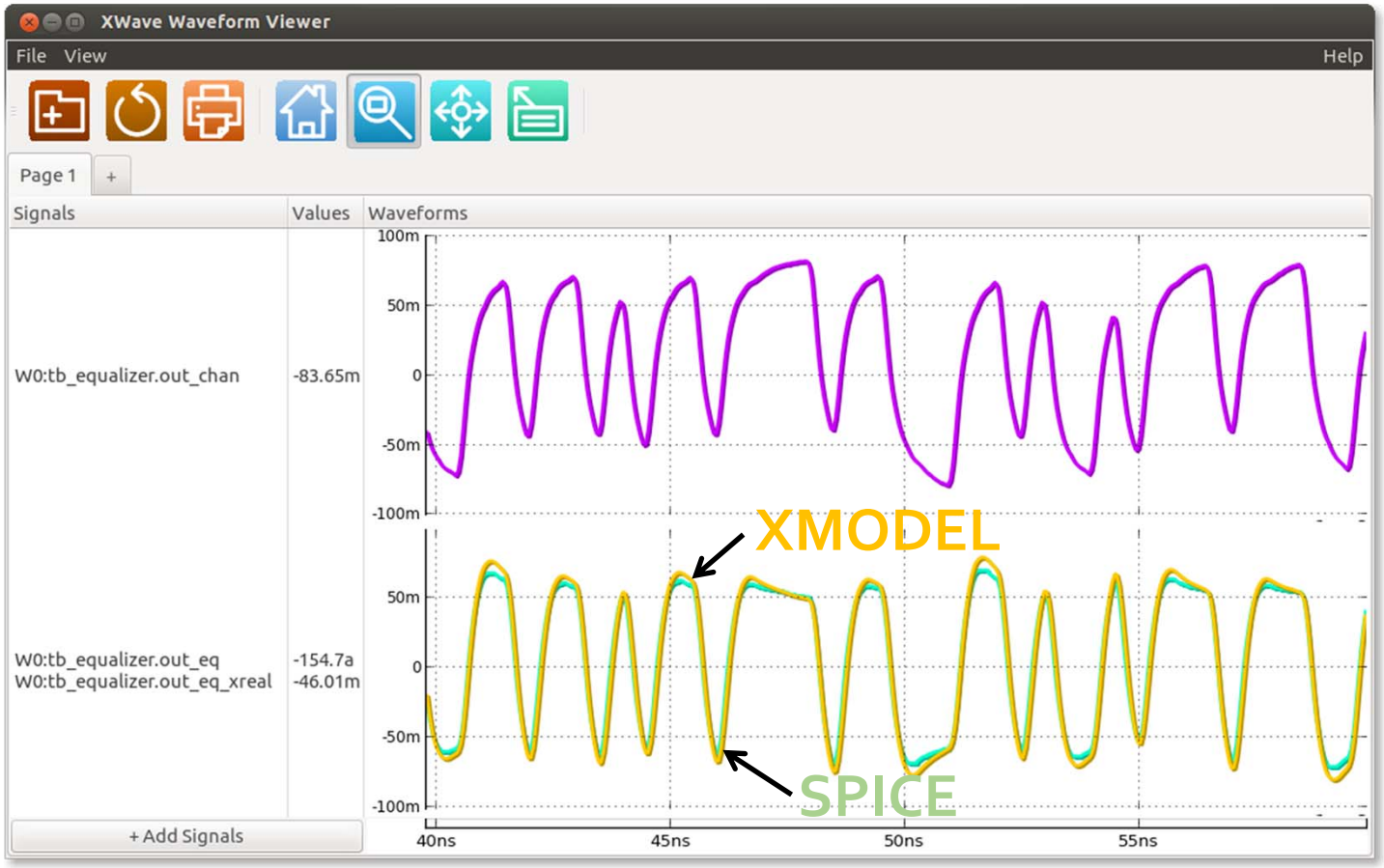
- With Hierarchy Editor and GLISTER, it's easy to generate netlists for *XMODEL*-SPICE co-simulation



# *XMODEL*-SPICE Co-Simulation

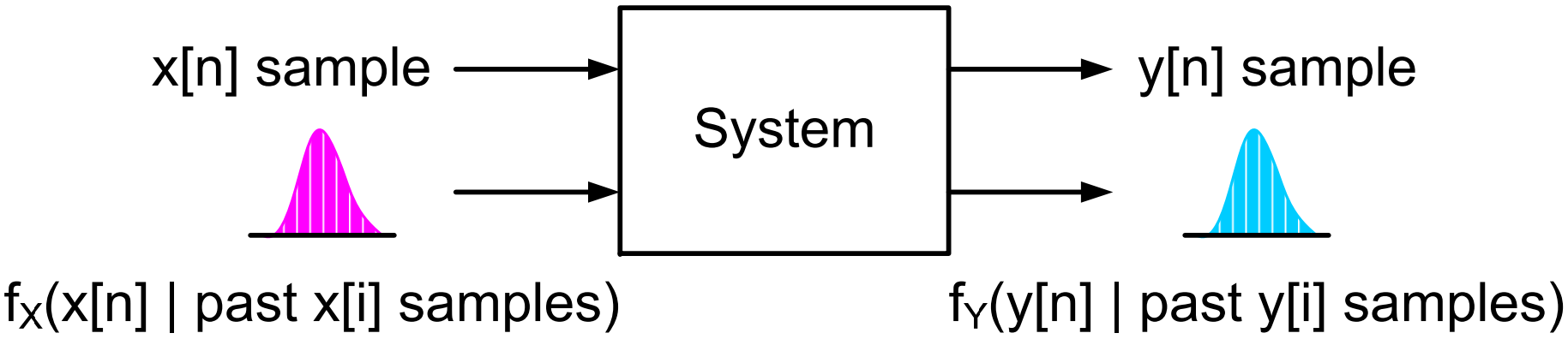
**Equalizer Input**

**Equalizer Outputs**



# Statistical Simulation in *XMODEL*

- *XMODEL* can simulate statistical characteristics of analog systems more efficiently than Monte-Carlo
- Again, the key is to add more information to signals
  - e.g. conditional probability distribution function (CPDF)



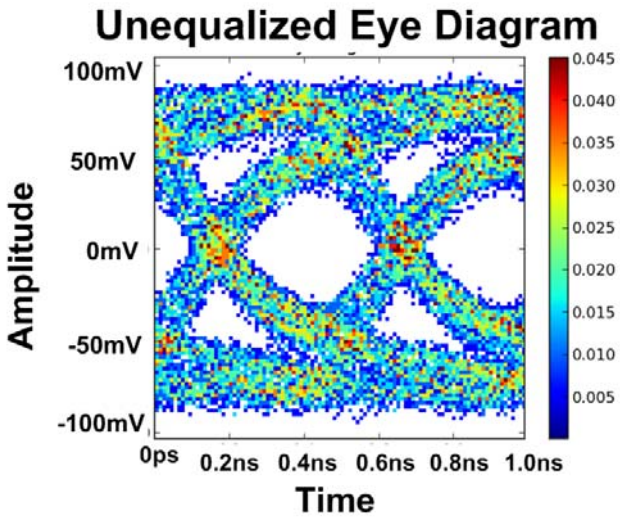
# Statistical Eye Diagram

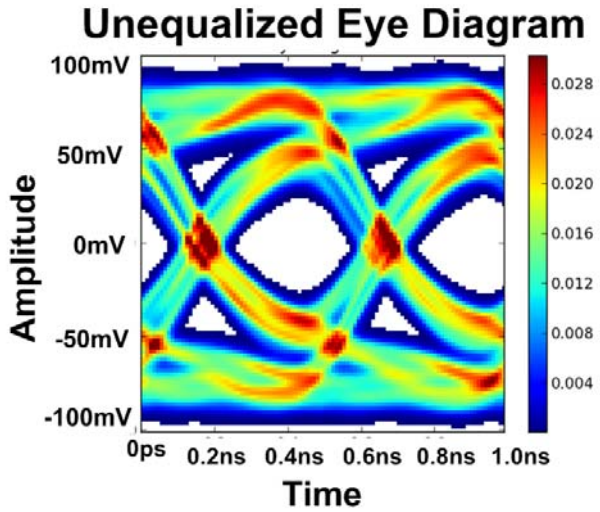
Monte-Carlo

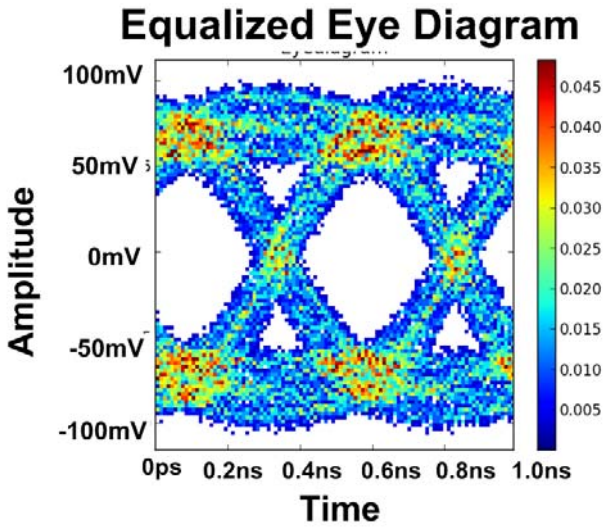
CPDF (Statistical)

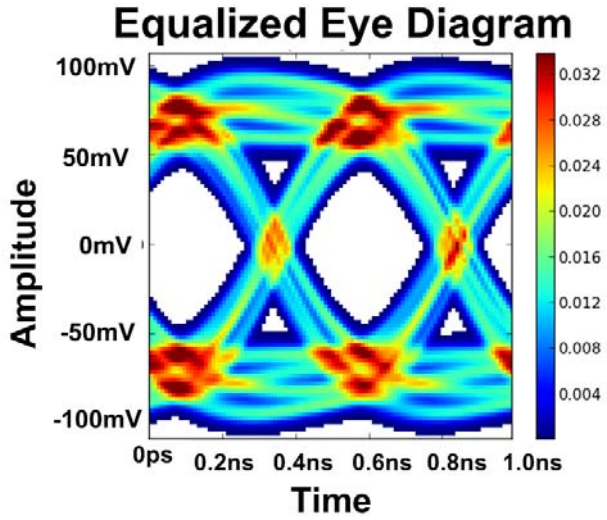
Before EQ

After EQ



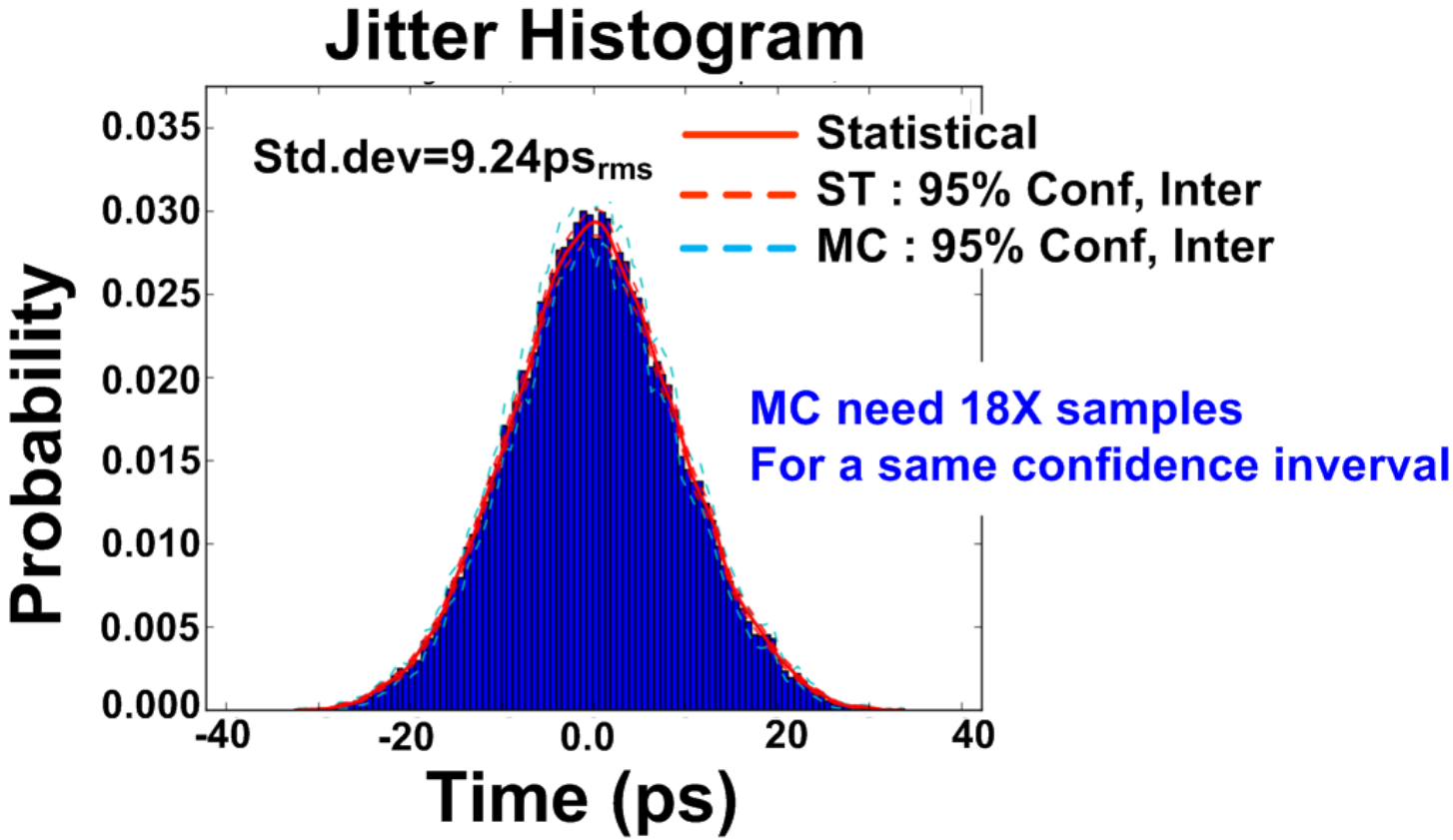



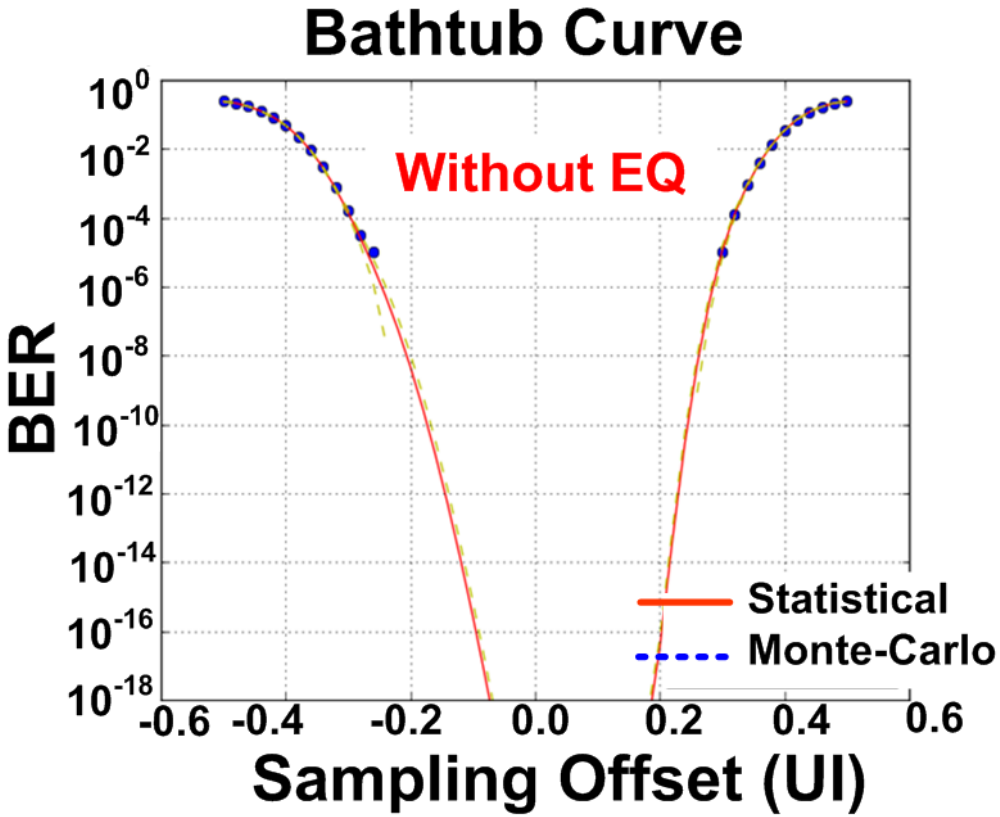
# Statistical Jitter Histogram

- CPDF method can estimate clock jitter histogram with higher confidence using fewer MC samples

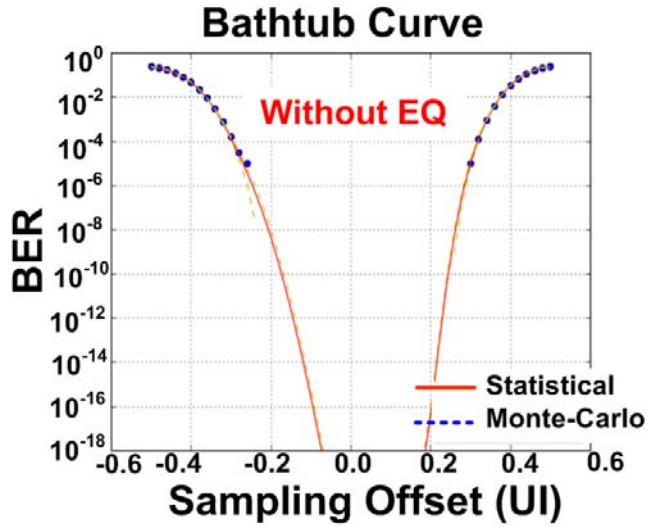


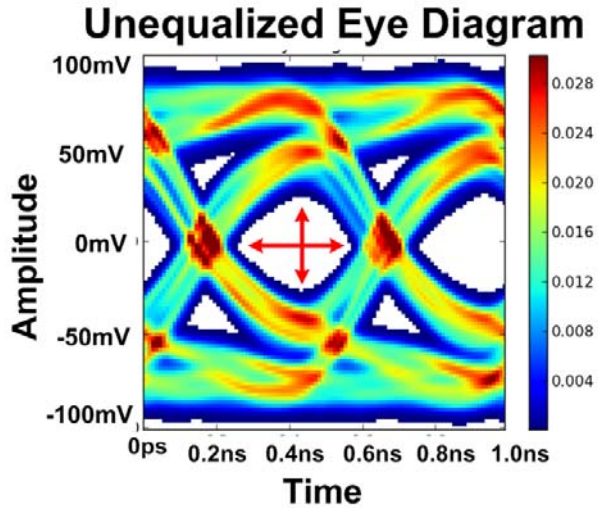
# BER Bathtub Curve

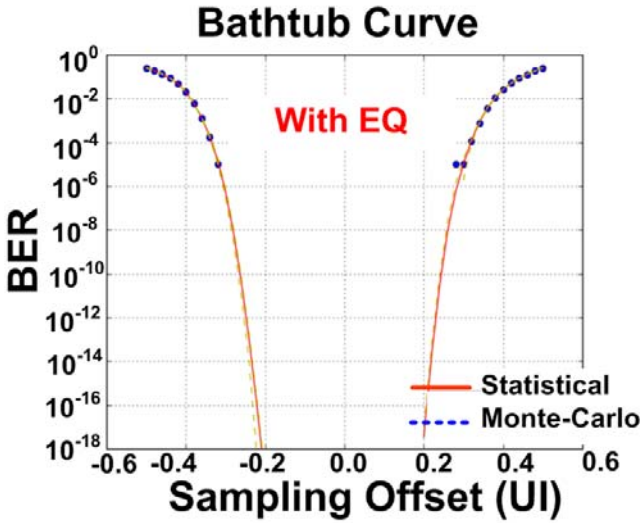
- Plotting BER as a function of timing offset
  - # of samples is $10^5$ for each point (~9.7 sec each)
  - Total simulation time is 8 minutes (50 points)

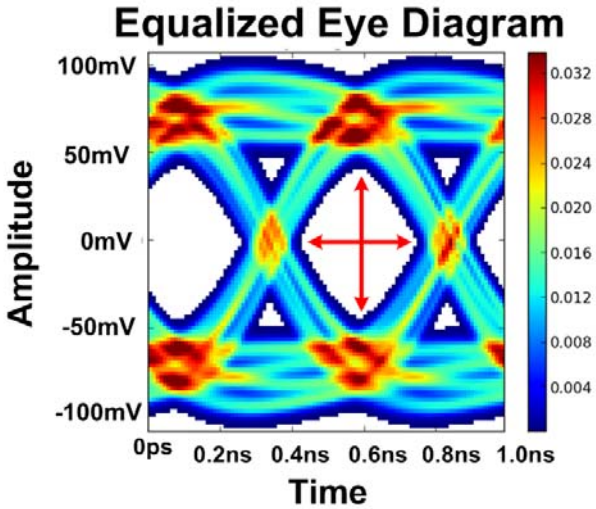


# Bathtub Curve vs. Eye Diagrams



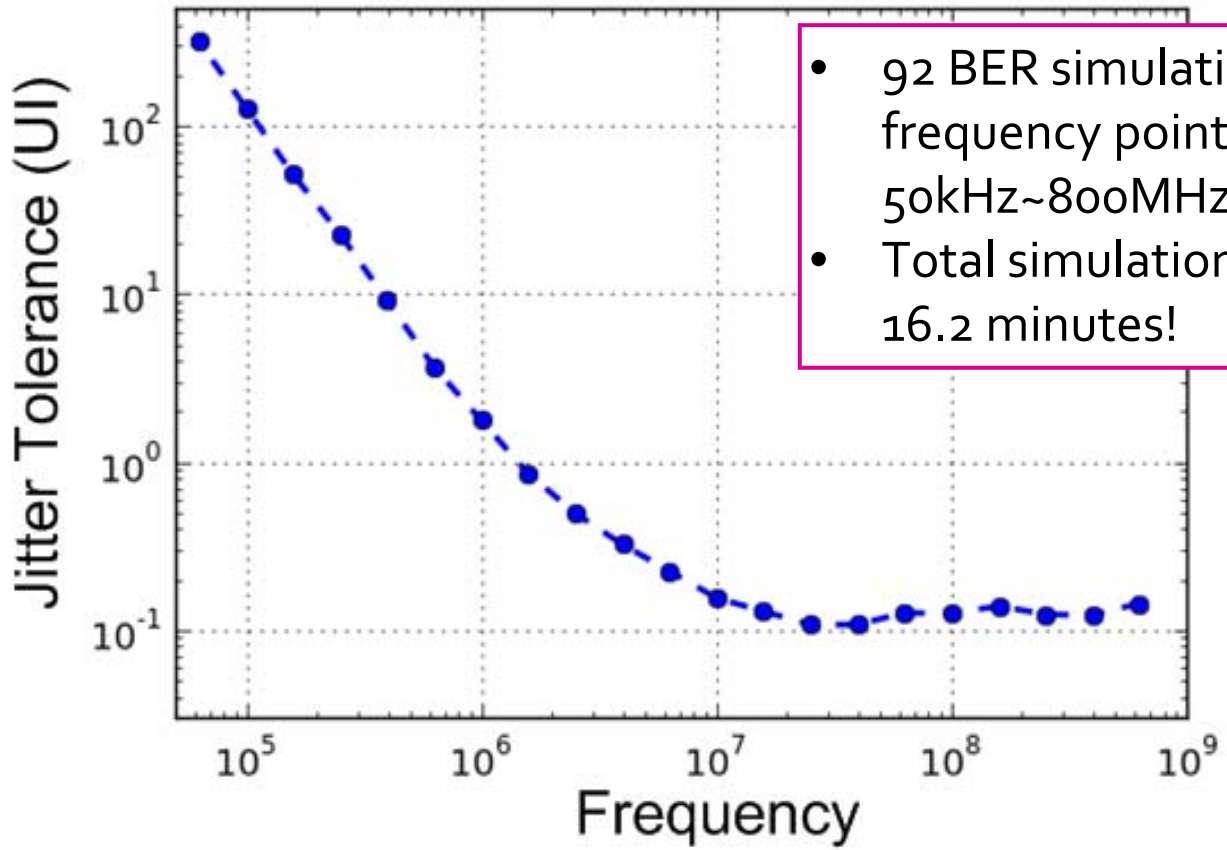



# Receiver Jitter Tolerance (JTOL)

- JTOL measures the largest sinusoidal jitter (SJ) a receiver can tolerate with BER < $10^{-12}$




- 92 BER simulations for 21 SJ frequency points over 50kHz~800MHz
- Total simulation time is only 16.2 minutes!

# Summary

- ***XMODEL*** is a fast analog/mixed-signal simulator that works seamlessly with digital verification flows (SystemVerilog, UVM, …)
  - Achieves both speed and accuracy (⇔Verilog-A, RNV)
  - Supports both block- and circuit-level models (⇔RNV)

- ***GLISTER*** and ***MODELZEN*** provide convenient interfaces to compose or auto-generate models and simulate them in GUI environment