

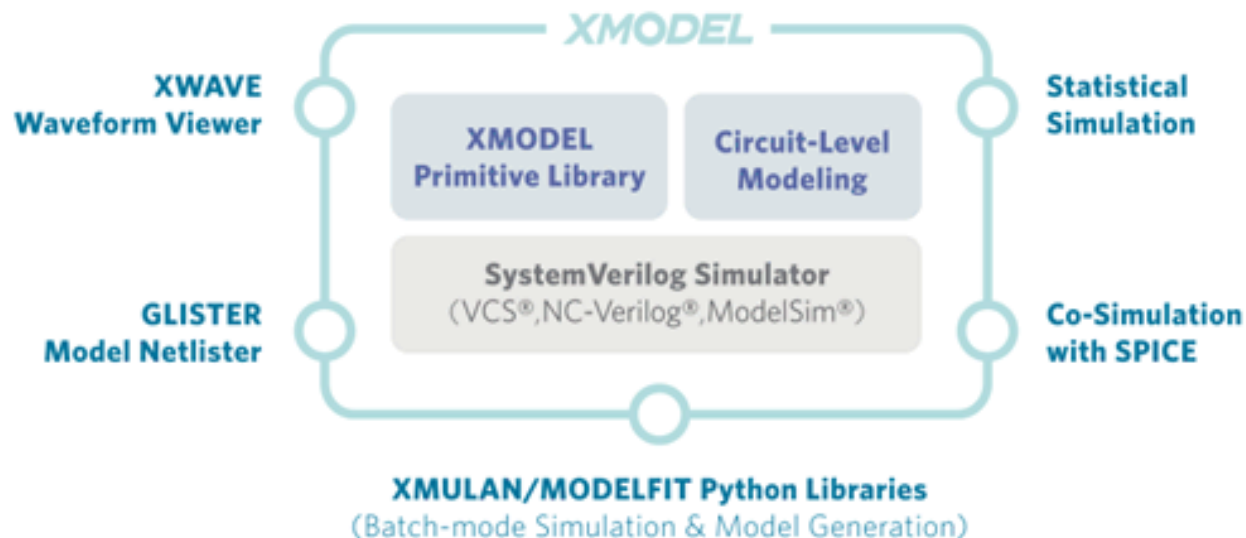
Analog/Mixed-Signal Modeling and Simulation with XMODEL

Jaeha Kim, Sung-Joon Lee, and Eunseo Kim
Mixed-Signal IC and System Group
Seoul National University
May 19. 2016

What is XMODEL?

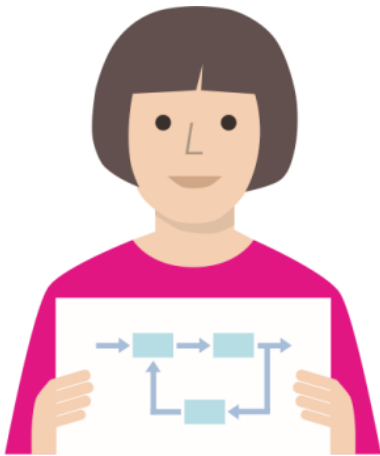
XMODEL is...

- 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)

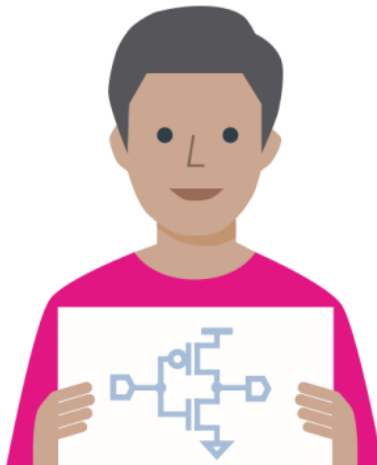


XMODEL is for Everyone!

- XMODEL is a versatile platform that can be used in multiple ways:
 - As a top-down tool, bottom-up tool, and final sign-off tool



System Architects



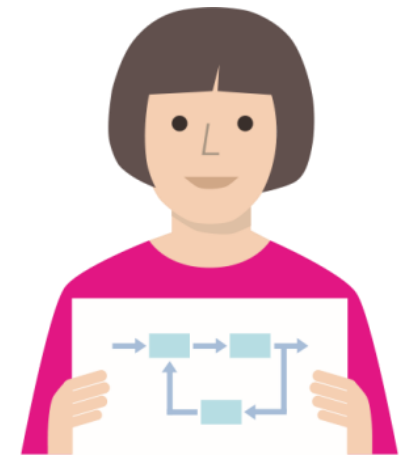
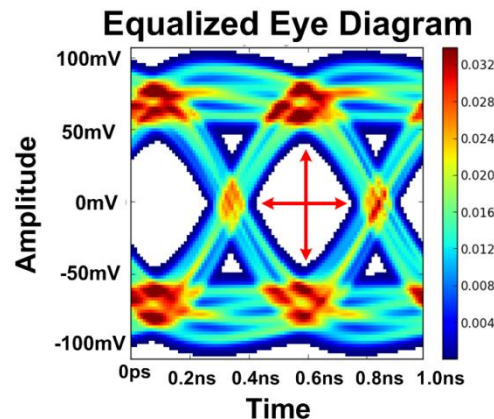
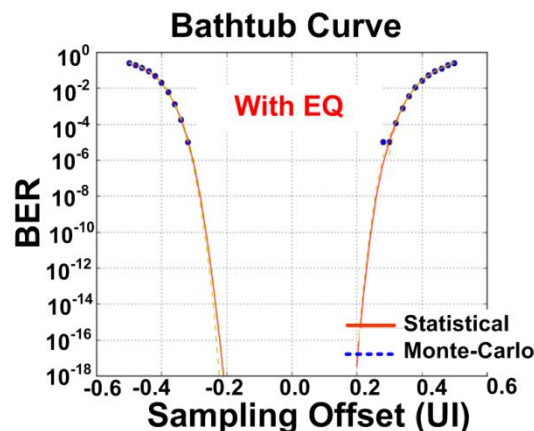
Circuit Designers



Verification Engineers

XMODEL as Top-Down Tool

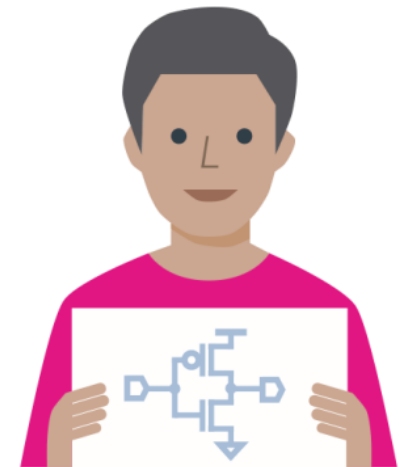
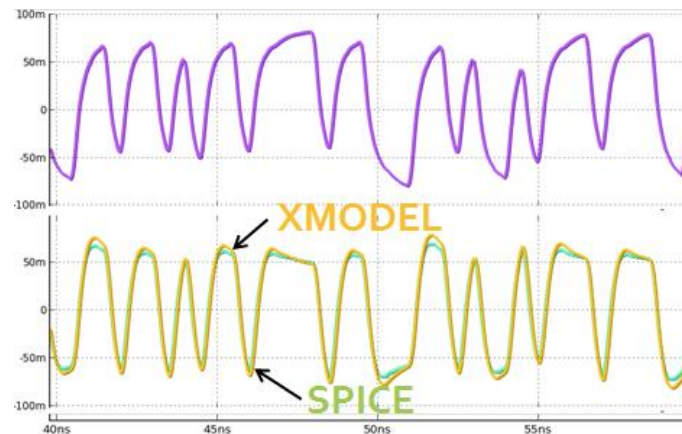
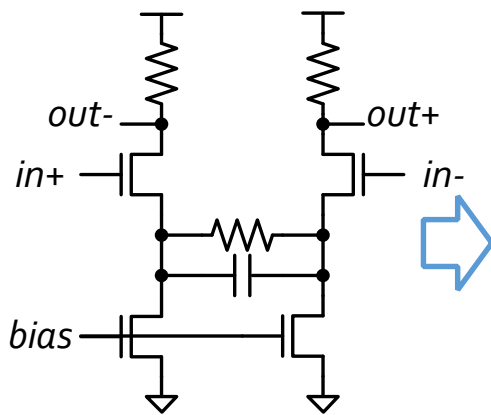
- Explore architectural choices using XMODEL as fast analog/mixed-signal functional simulator
 - e.g. compare BERs of high-speed links with different clocking and equalization schemes
 - Faster speed compared to SPICE, Verilog-A/MS
 - Digital models are directly synthesizable after verification (vs.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



System Architects

XMODEL as Bottom-Up Tool

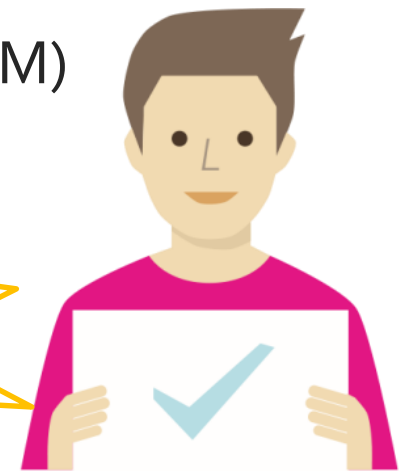
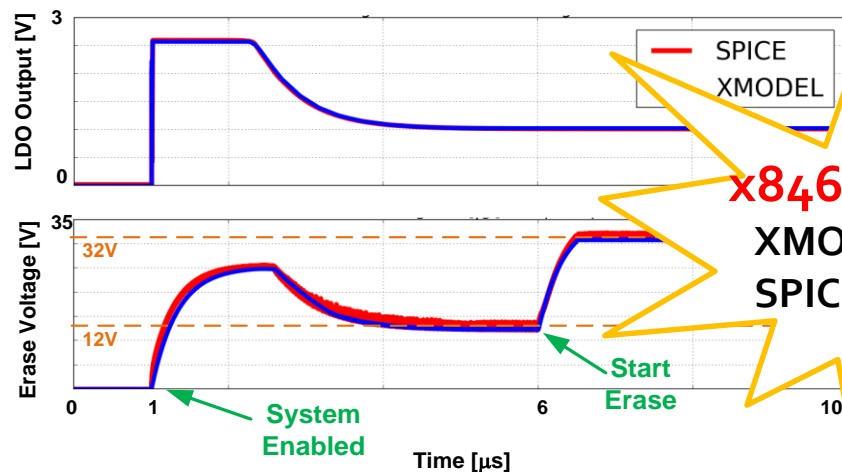
- Write or extract analog models easily with XMODEL
 - Write functional models with a rich set of primitives
 - Use circuit-level modeling (CLM) for structural models
 - Calibrate functional models or extract structural models using *MODELFIT* and *MODELGEN*



Circuit Designers

XMODEL as Sign-Off Tool

- Verify chip-level functionality 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)



Verification Engineers

***XMODEL* Also Includes:**

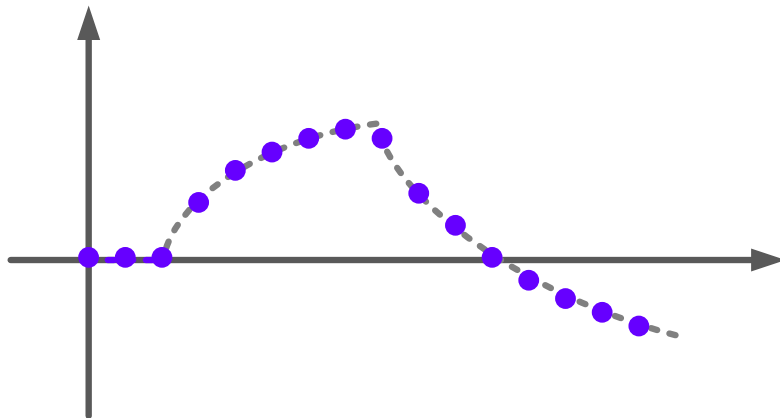
- **Statistical simulation support (STATSIM)**
 - Simulate probabilities of statistical events such as BER, jitter, phase noise, etc., during time-domain simulation
- **Cadence Virtuoso integration support (GLISTER)**
 - Create models graphically with a schematic editor
- **Model extraction and generation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

Key Concepts in *XMODEL*

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

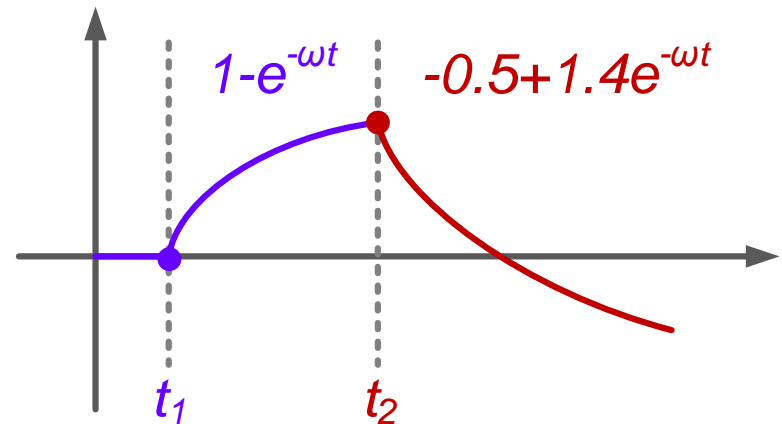
$$x(t) = \sum_i c_i t^{m_i} e^{-a_i t}$$

SPICE



Accuracy relies on fine time step

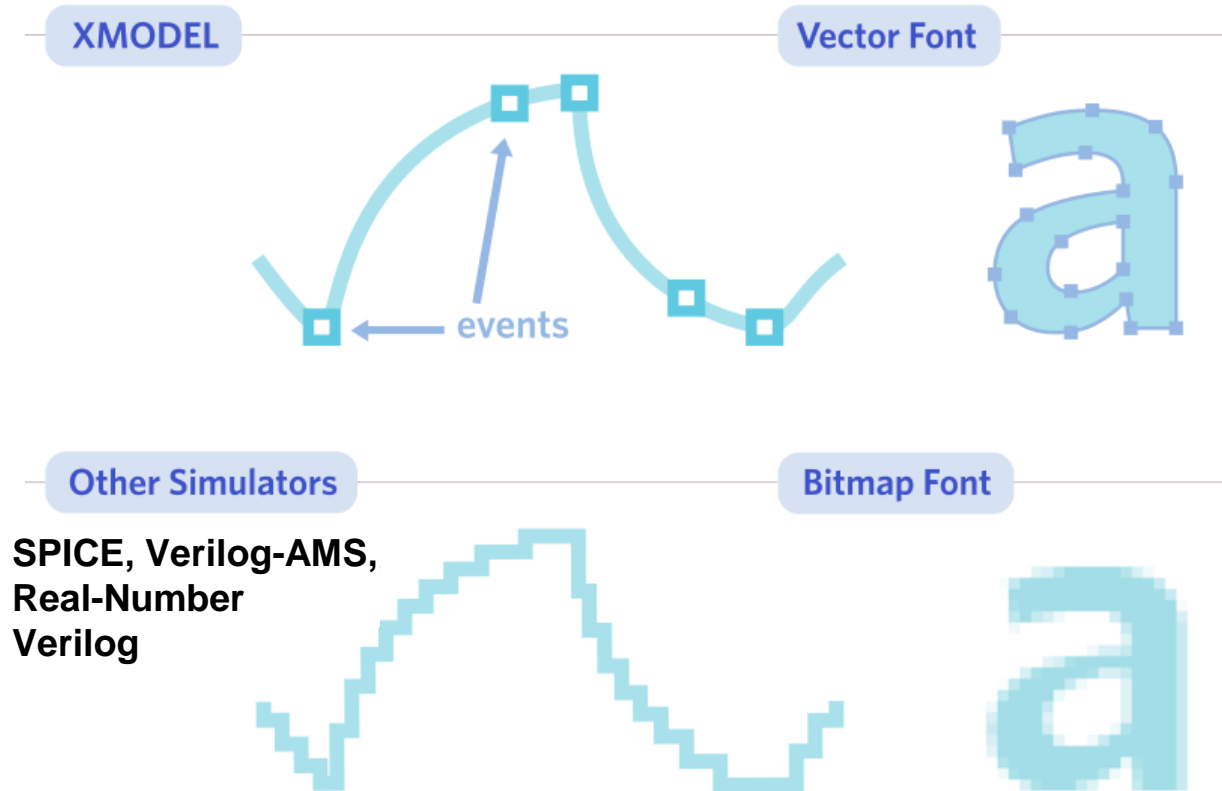
XMODEL



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

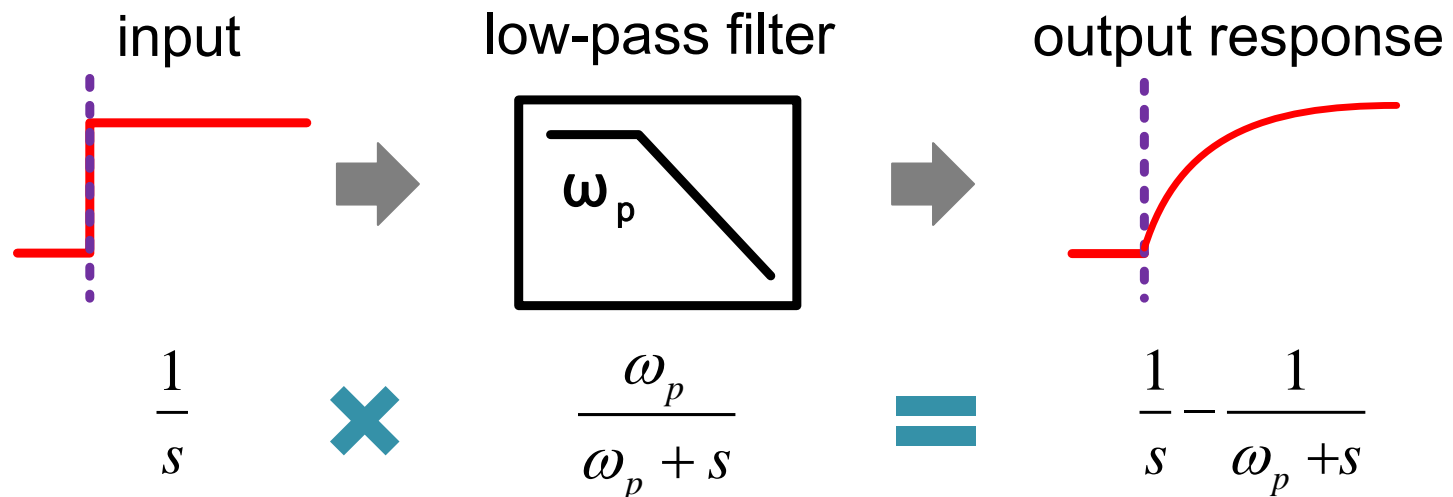


Simulating Analog System Responses

- Signals have an alternate s-domain representation:

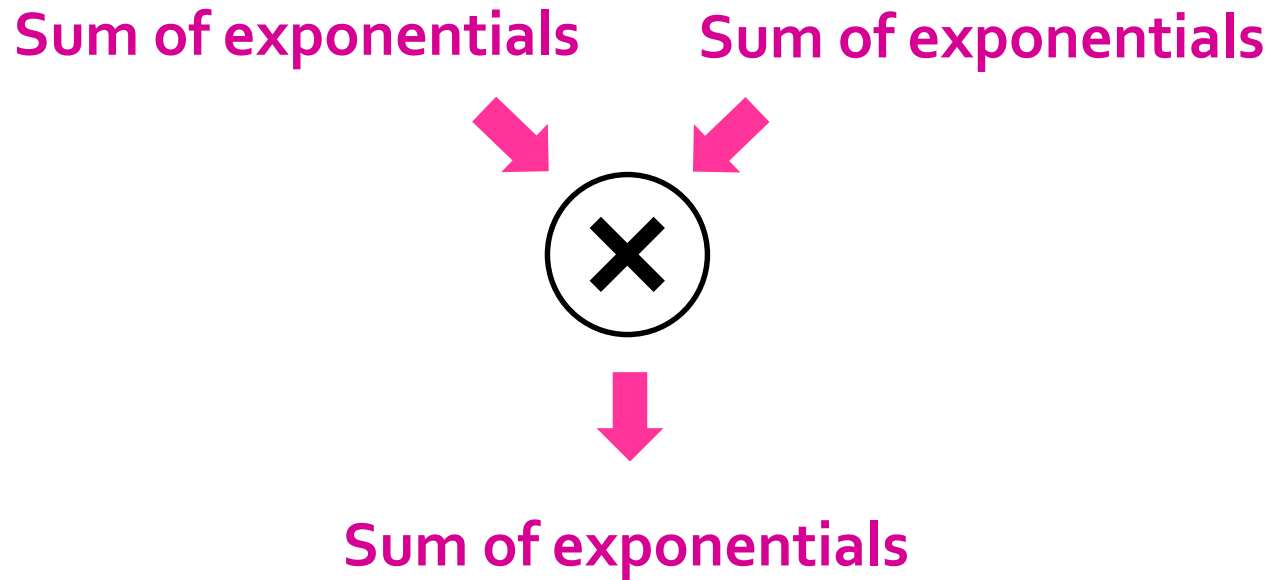
$$x(t) = \sum_i c_i t^{m_i-1} e^{-a_i t} u(t) \xrightarrow{\mathcal{L}} X(s) = \sum_i \frac{b_i}{(s + a_i)^{m_i}}$$

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



Modeling Nonlinear Responses

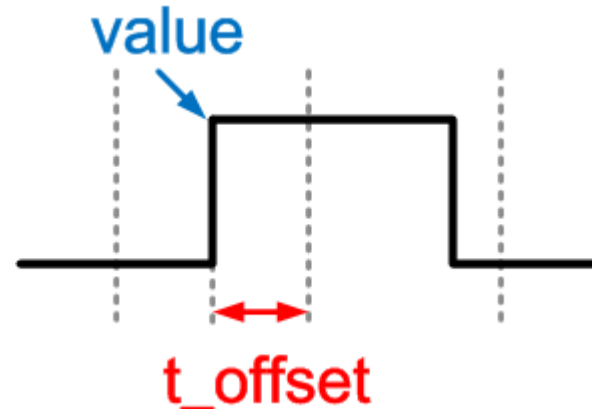
- The functional expression of the signal and event-driven nature of its computation can be maintained even with nonlinear systems
 - Example of mixing two analog signals:



XMODEL Data Types: *xbit* and *xreal*

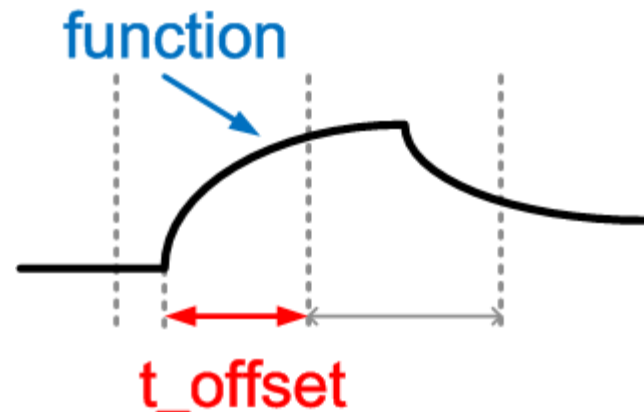
- ***xbit*** for digital signals

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



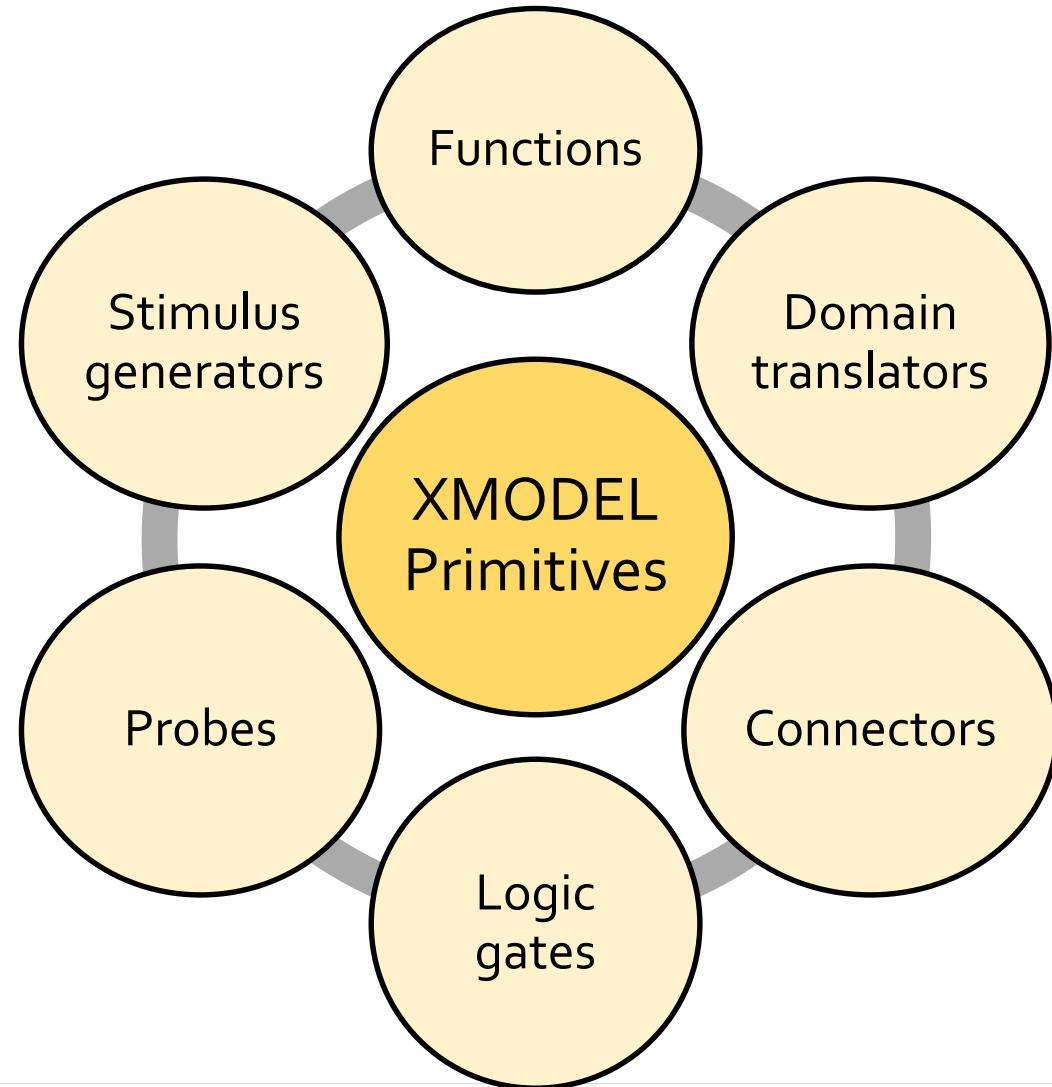
- ***xreal*** for analog signals

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



XMODEL Primitives

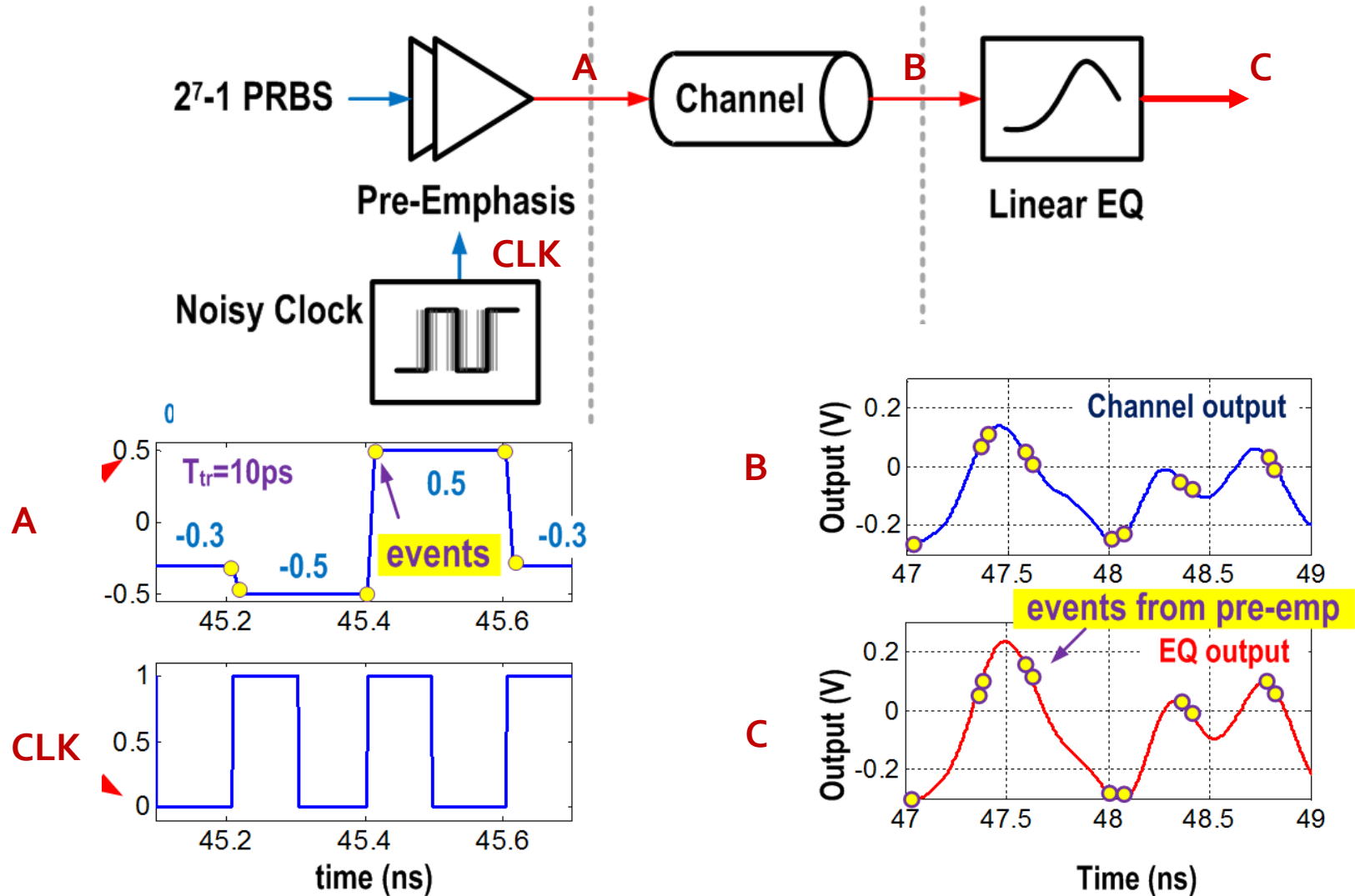
- XMODEL package includes 6 different types of primitives
- Users can easily describe AMS systems without understanding the XMODEL algorithms



XMODEL Primitive Examples

- Functions
 - add, scale, multiply, deriv, integ, integ_mod, filter, delay, select, limit, power, pwl_func, poly_func, transition, sample, compare, dac, adc, ...
- Stimulus generators
 - dc_gen, noise_gen, step_gen, exp_gen, sin_gen, pwl_gen, clk_gen, pulse_gen, pat_gen, prbs_gen, ...
- Domain translators
 - clk_to_freq, clk_to_phase, clk_to_period, clk_to_duty, clk_to_delay, freq_to_clk, phase_to_clk, period_to_clk, duty_to_clk, delay_to_clk

Channel Simulation Example



XMODEL Code Example

```
module channel_and_EQ;
```

```
  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),.value0(value0),.value1(value1))
    driver(.in(in_prbs),.out(out_drv));
```

```
  // channel
```

Description of System Behaviors

```
  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));
```

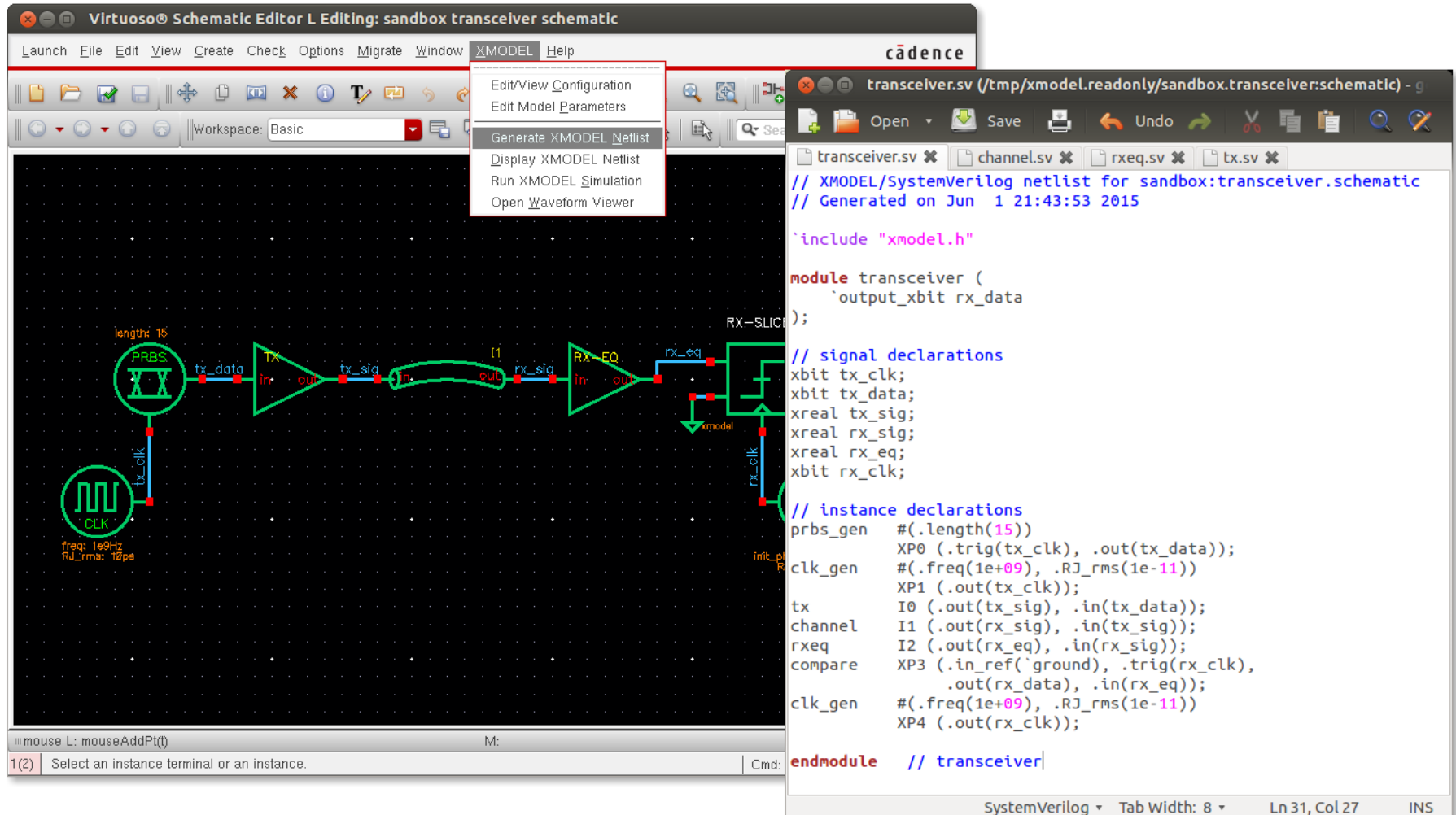
```
  // probes
```

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

Waveform Recording

```
endmodule
```

18



Comparison with Real-Number Verilog

XMODEL

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

transition #(.value0(0), .value1(l_up))
    cp_up(up, i_up);
transition #(.value0(0), .value1(-l_dn))
    cp_dn(dn, i_dn);
add cp_out('{i_up, i_dn}, lfilter);

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

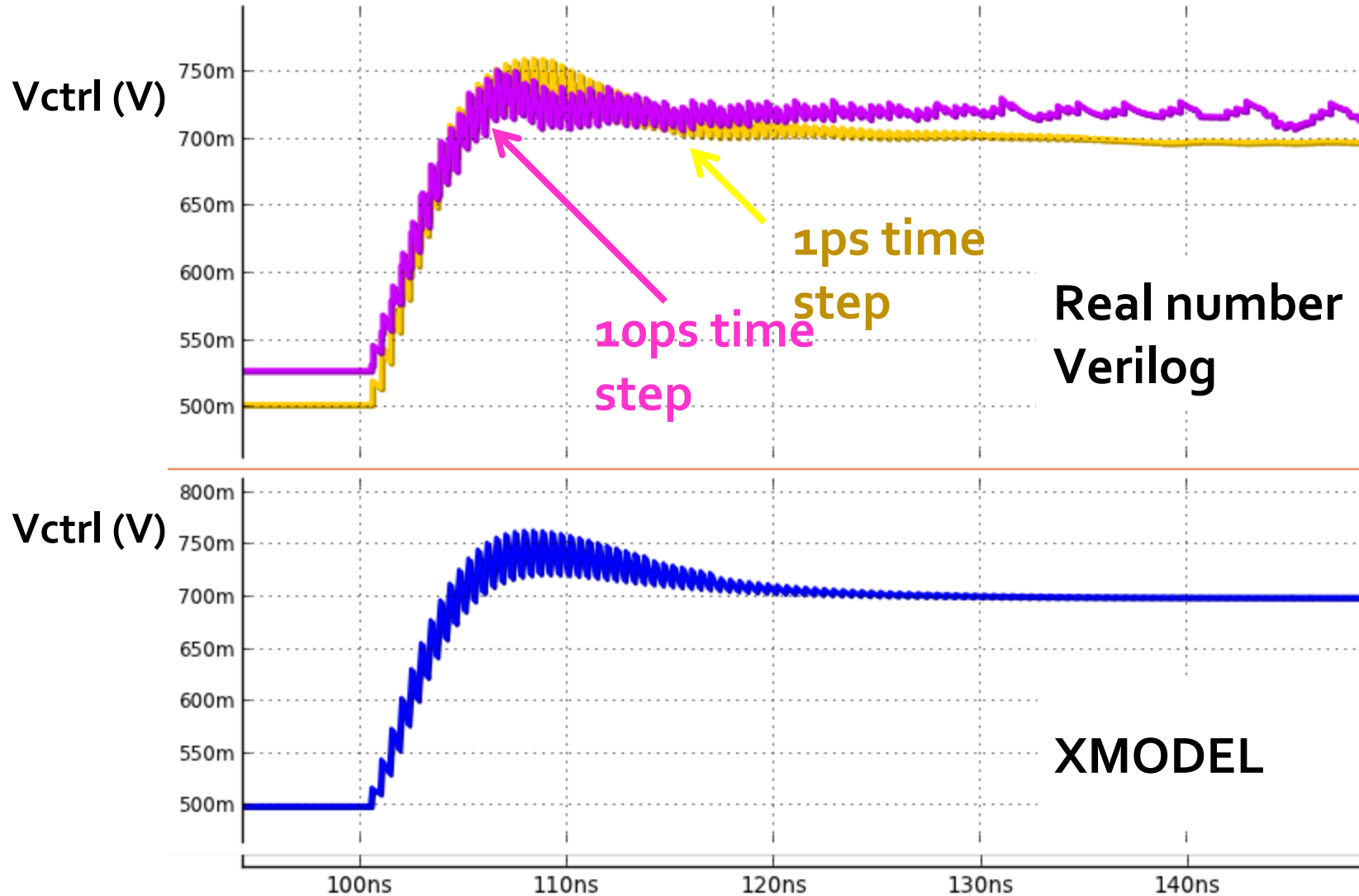
endmodule
```

Real-Number Verilog

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

always begin
    #1;
    lfilter = up*iup - dn*ldn;
    vctrl = (A*(u_post + t_step*lfilter
        + 1/z1 * lfilter)
        + 1/p1 *t_step *vctrl_post)
        / (1 + 1/p1 * t_step) ;
    vctrl_post = vctrl;
    u_post = u_post + t_step* lfilter;
end
endmodule
```

Charge-Pump PLL Locking Transients



Charge-Pump PLL Jitter Histogram

Time Step

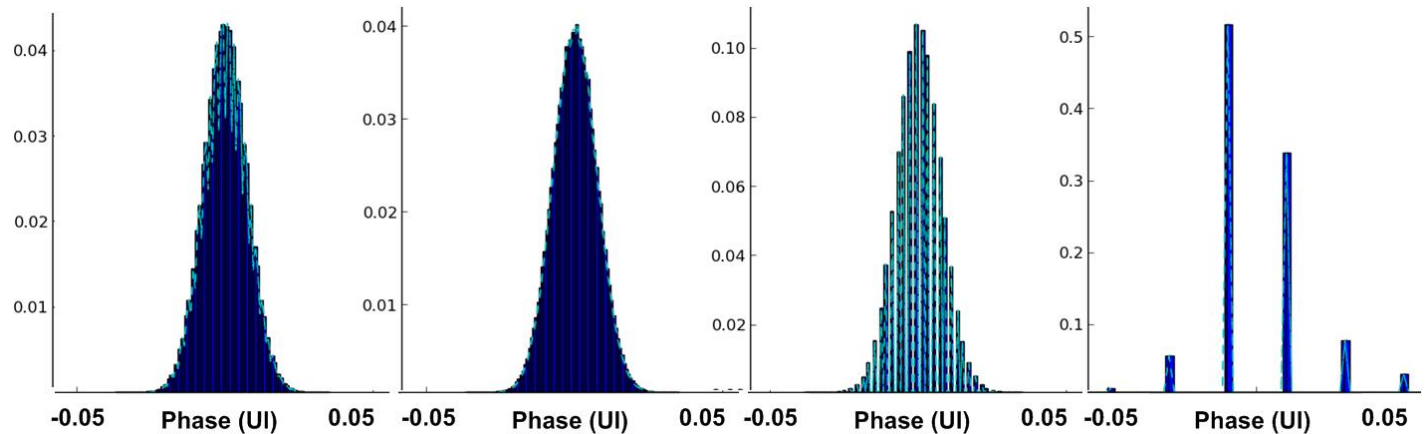
0.01ps

0.1ps

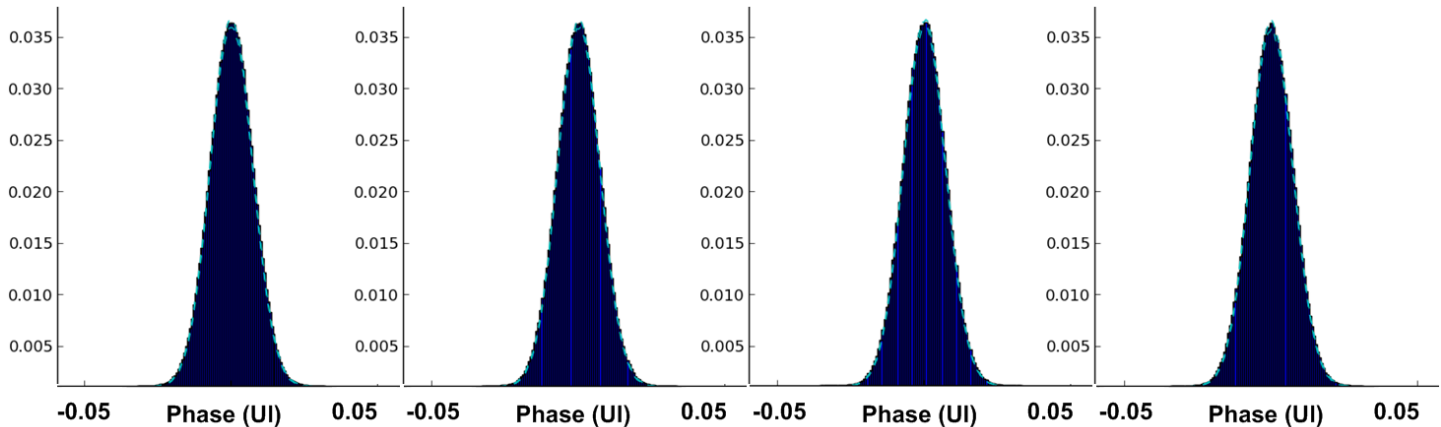
1ps

10ps

Real-Number
Verilog

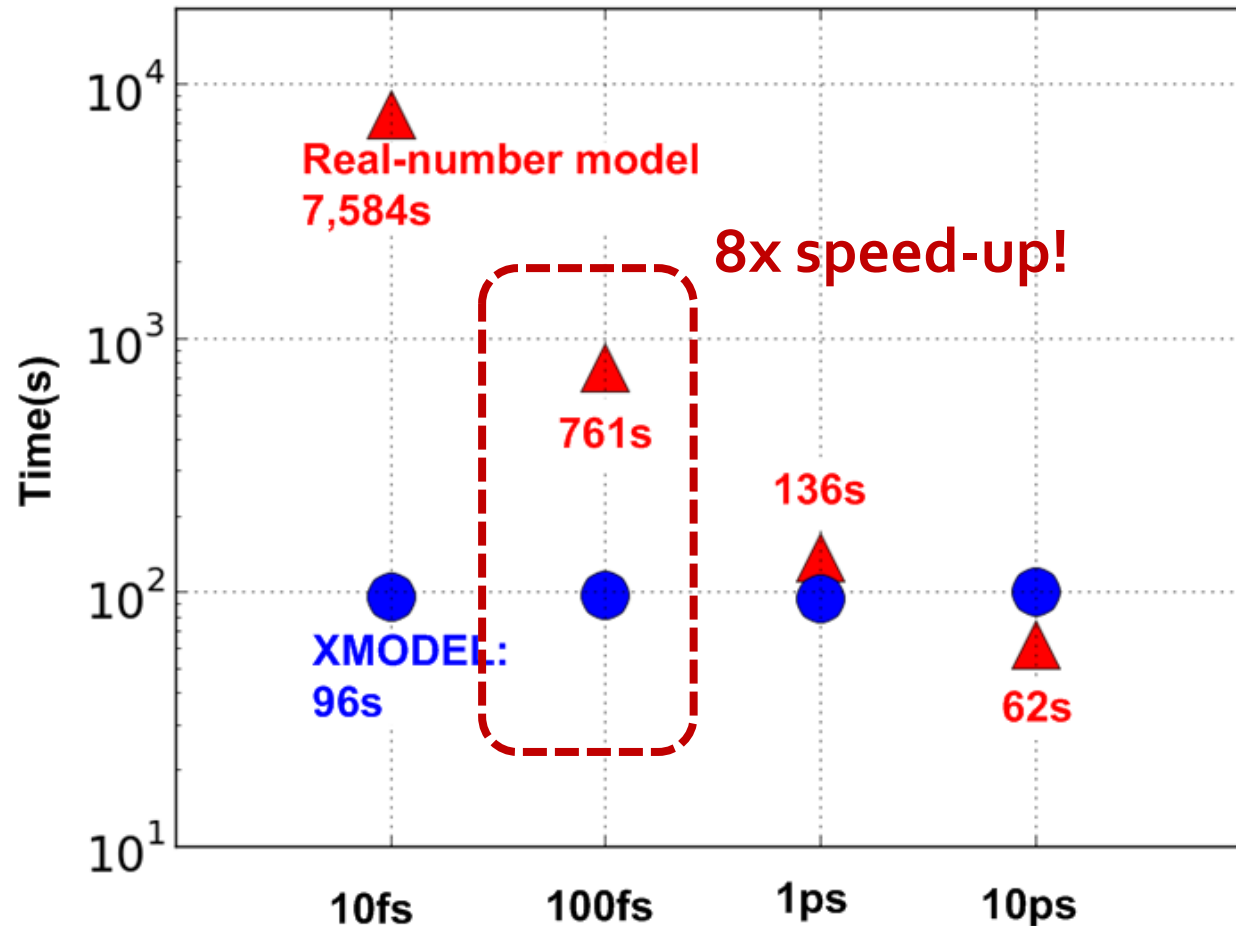


XMODEL



Simulation Speed Comparison

- Execution time of 10^6 clock cycle simulation

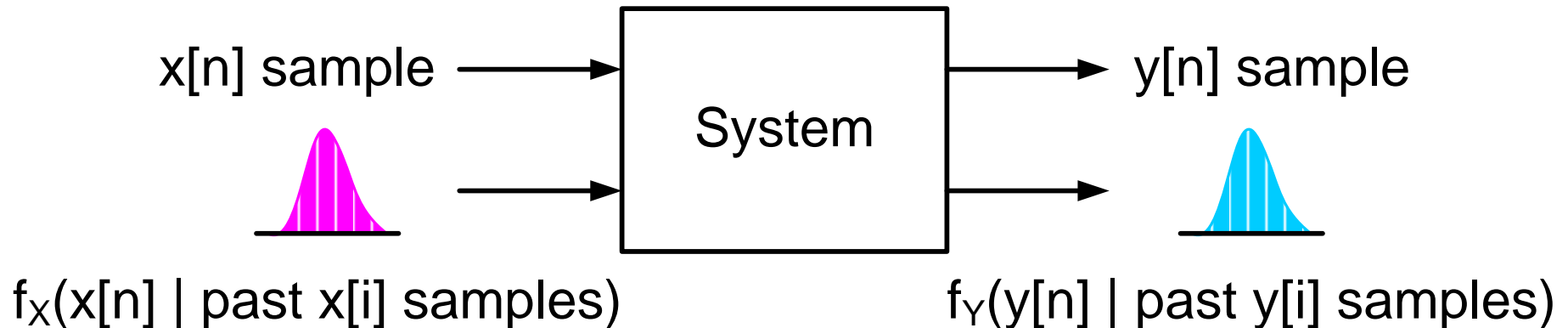


Comparison Table

| | 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, ModelGen, ...) | Moderate (Arana) | Moderate (SMG, Arana) | Easy (MATLAB) |

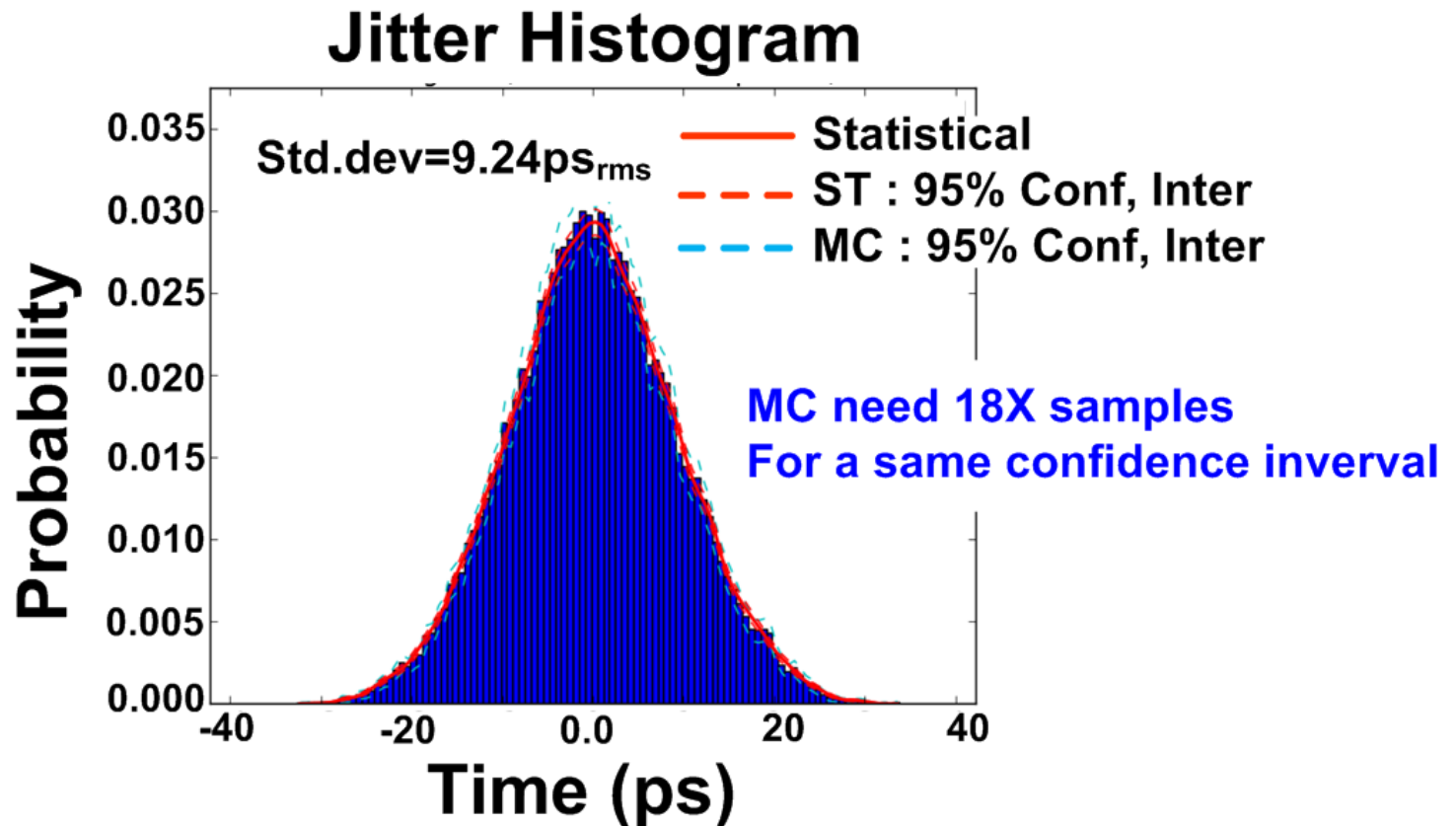
Statistical Simulation in *XMODEL*

- 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 Jitter Histogram

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

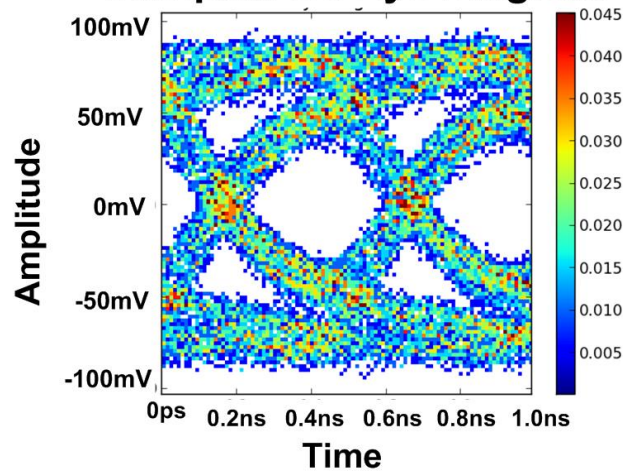


Statistical Eye Diagram

Before
EQ

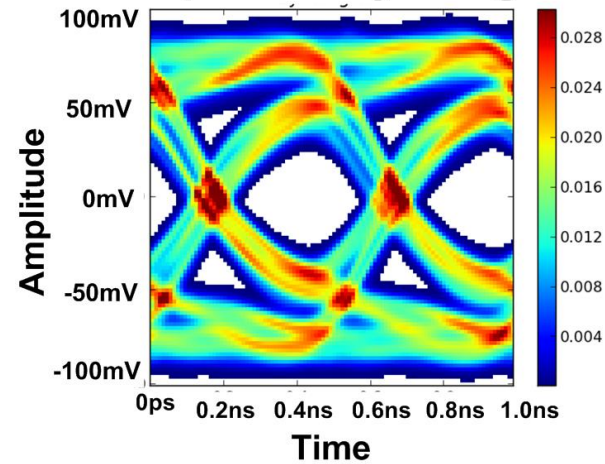
Monte-Carlo

Unequalized Eye Diagram



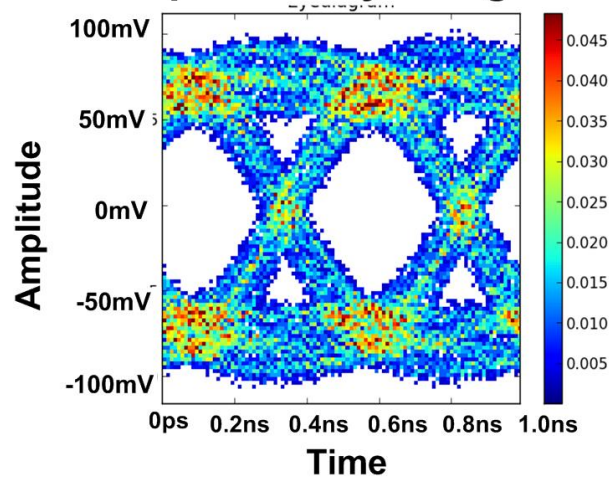
CPDF (Statistical)

Unequalized Eye Diagram

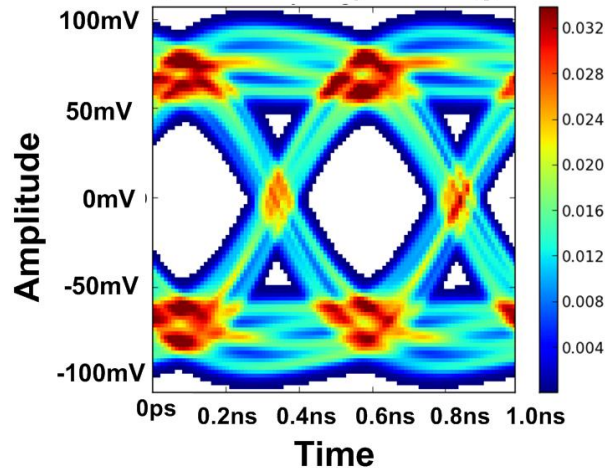


After
EQ

Equalized Eye Diagram

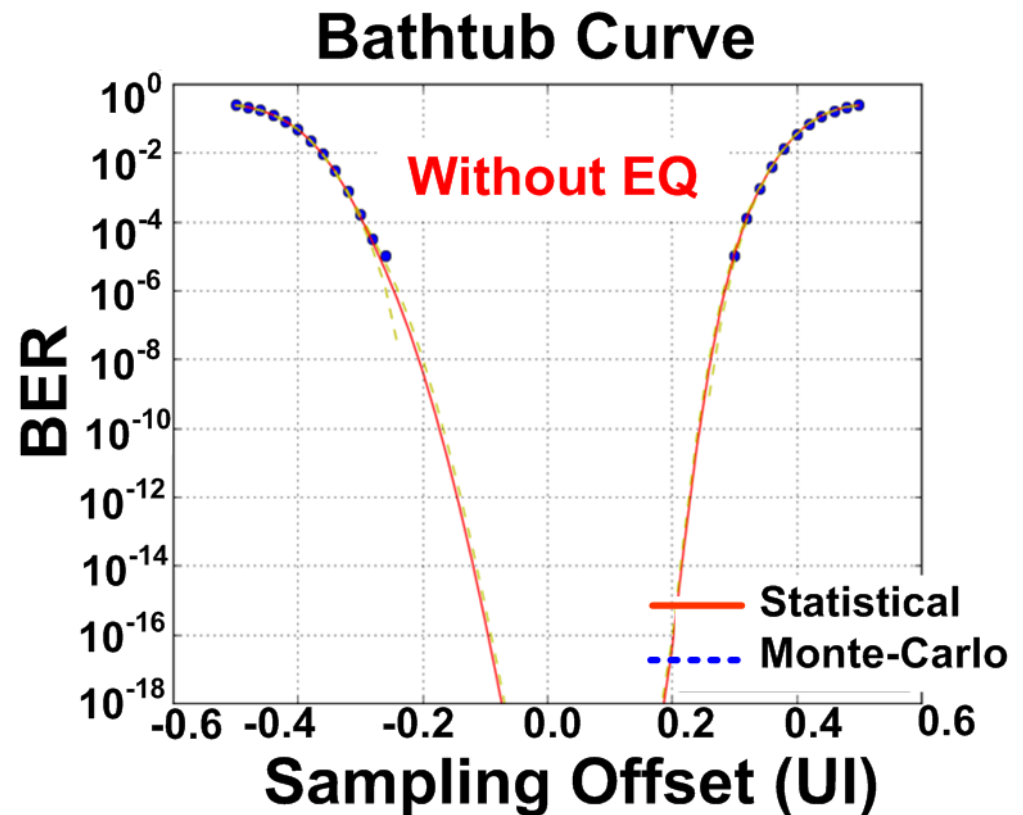


Equalized Eye Diagram



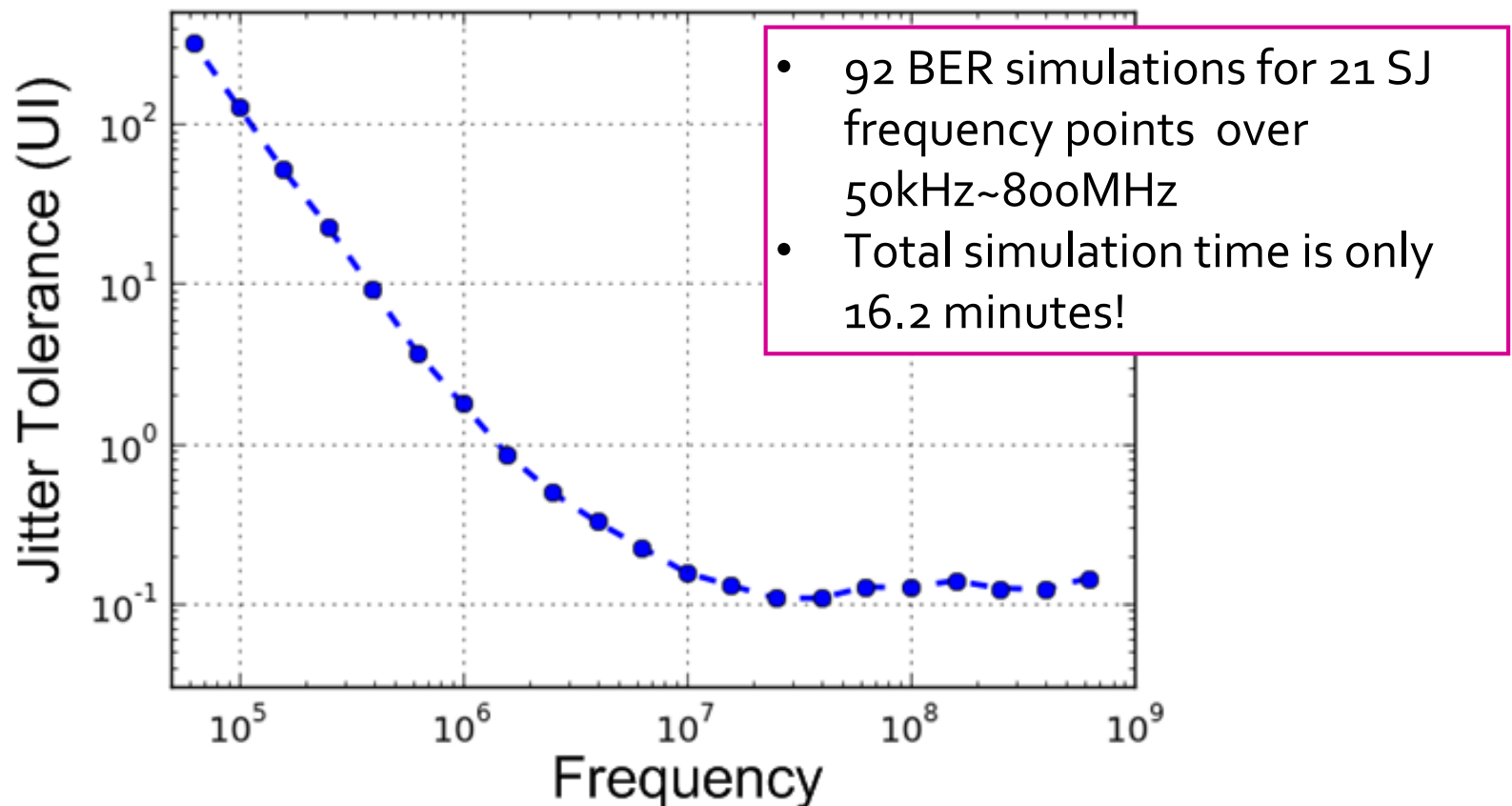
BER Bathtub Curve

- 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)



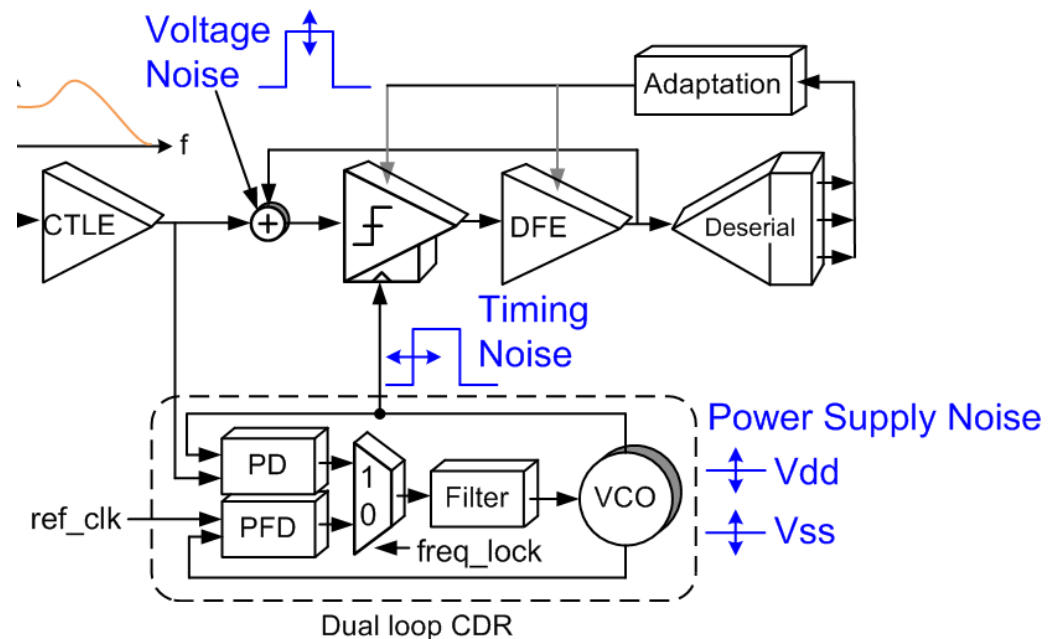
Receiver Jitter Tolerance (JTOL)

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



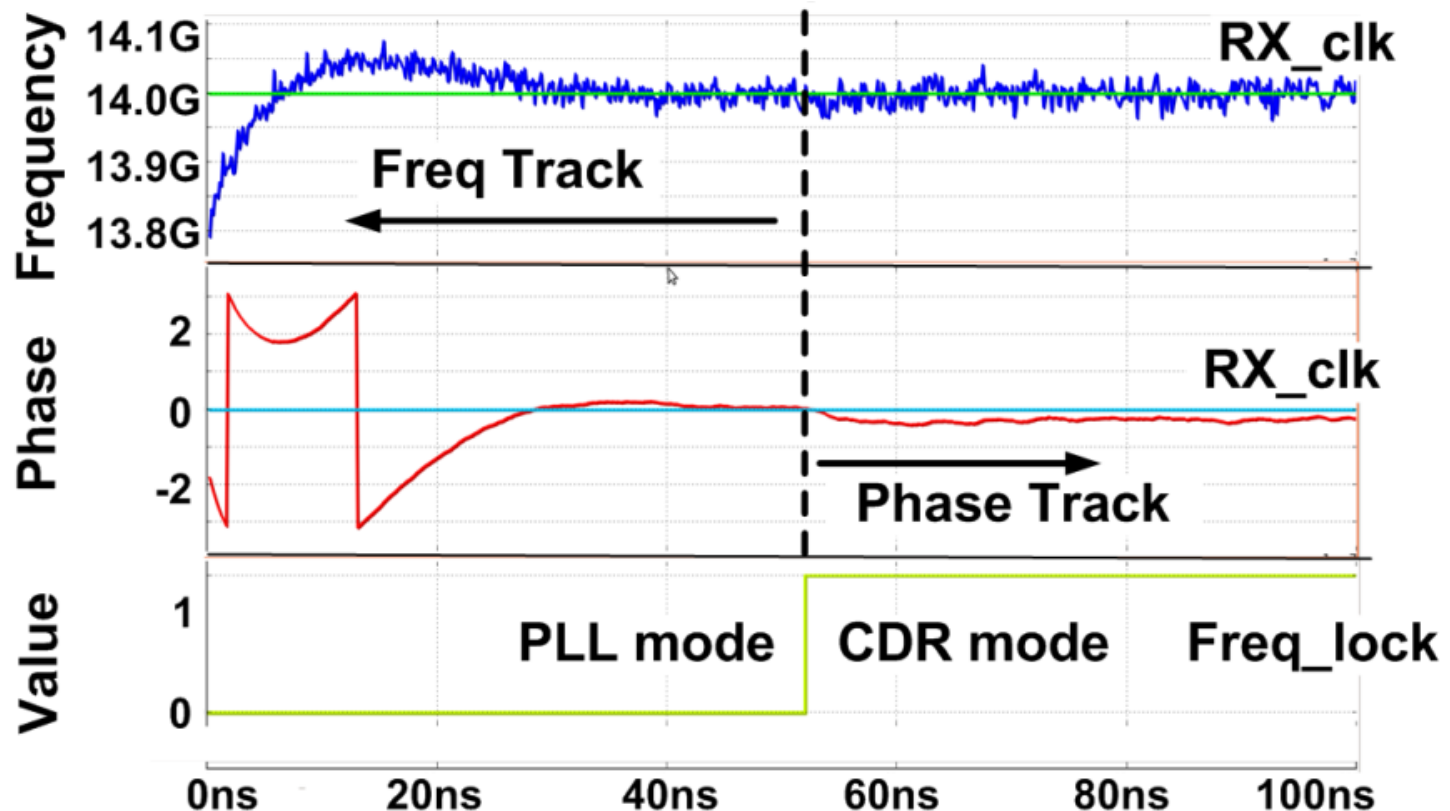
EQ/CDR Adaptation Example

- 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



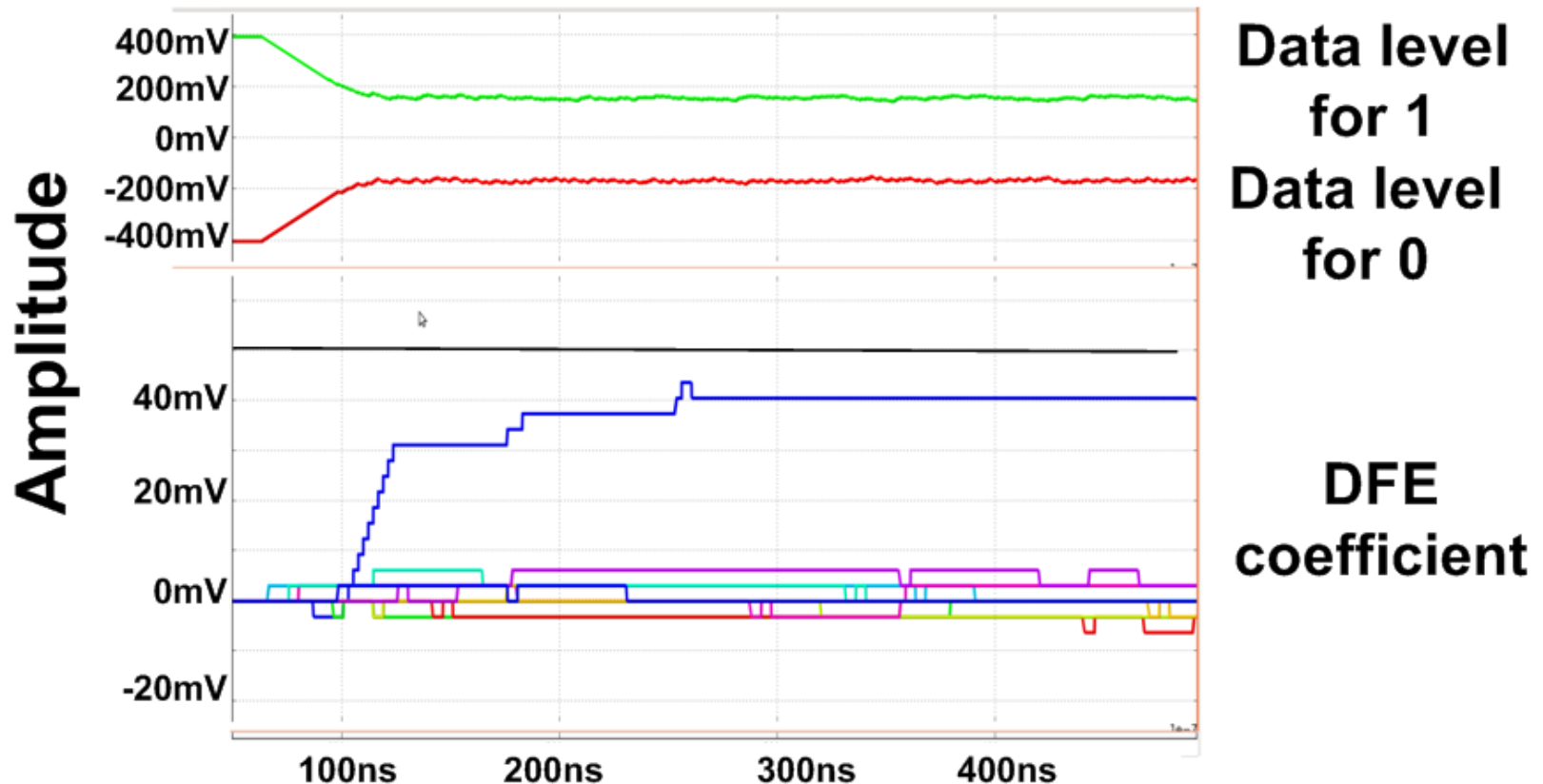
CDR Locking Transient

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



DFE Adaptation Transient

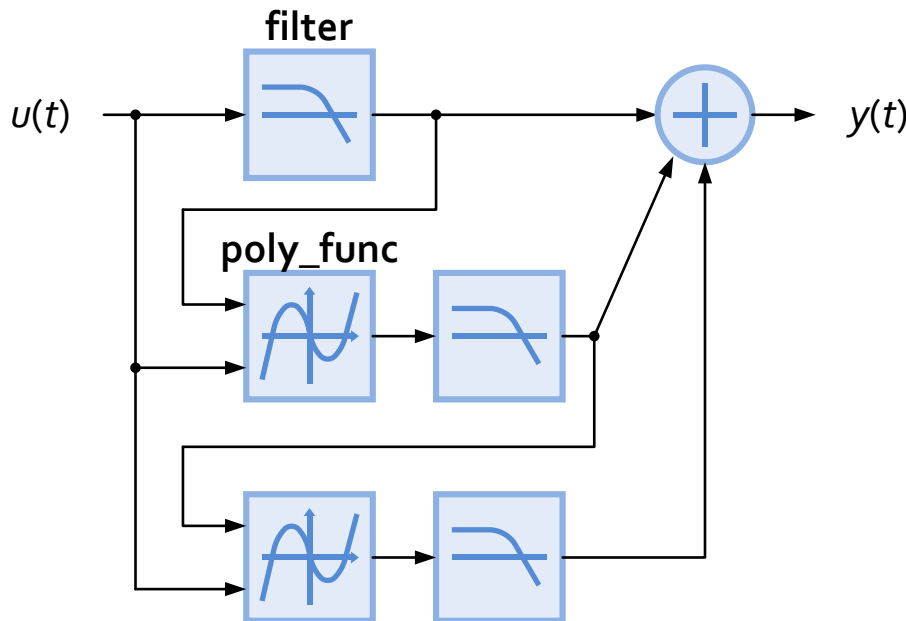
- DFE tap coefficients and data levels are trained using sign-sign LMS algorithm



Nonlinear System Modeling

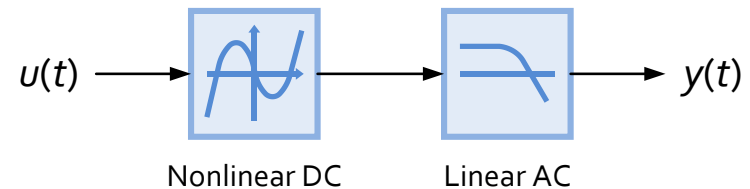
- With XMODEL primitives, you can construct various nonlinear models and calibrate their parameters

Volterra Series Model

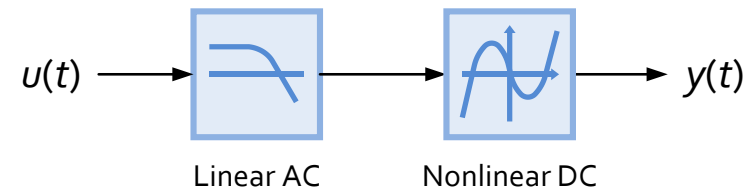


[Roychowdhury,1999]

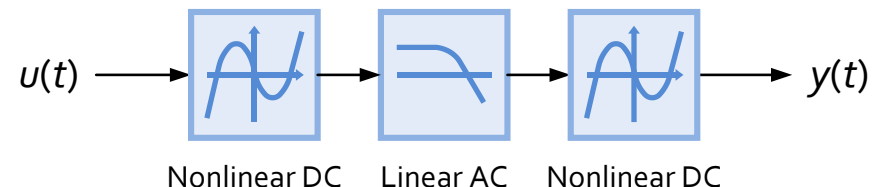
Hammerstein Model



Wiener Model



Hammerstein-Wiener Model



Nonlinear CTLE

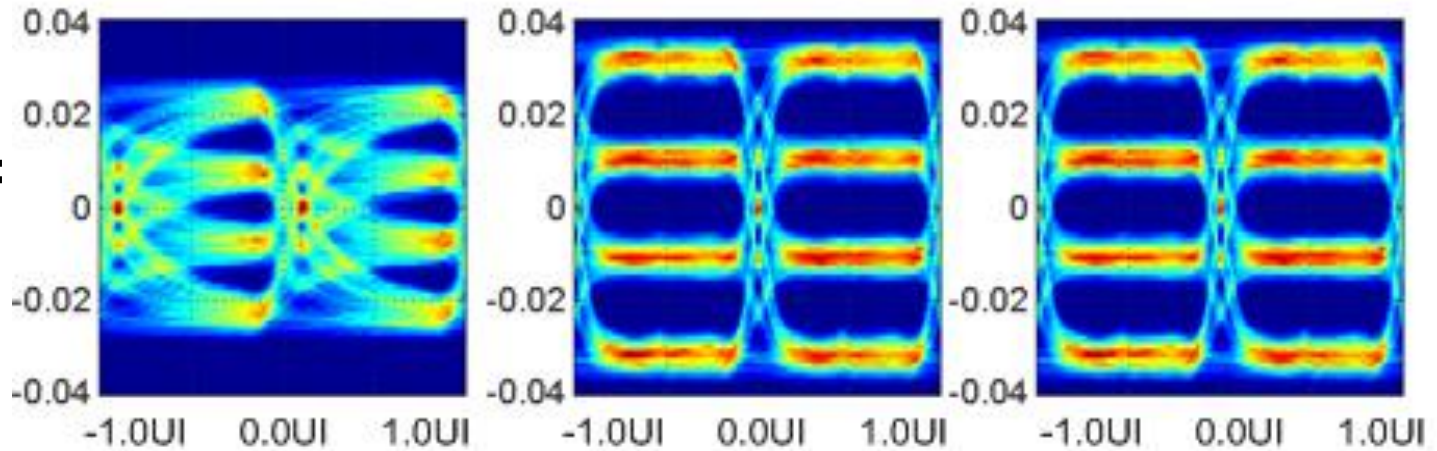
[Jang'13]

Channel output

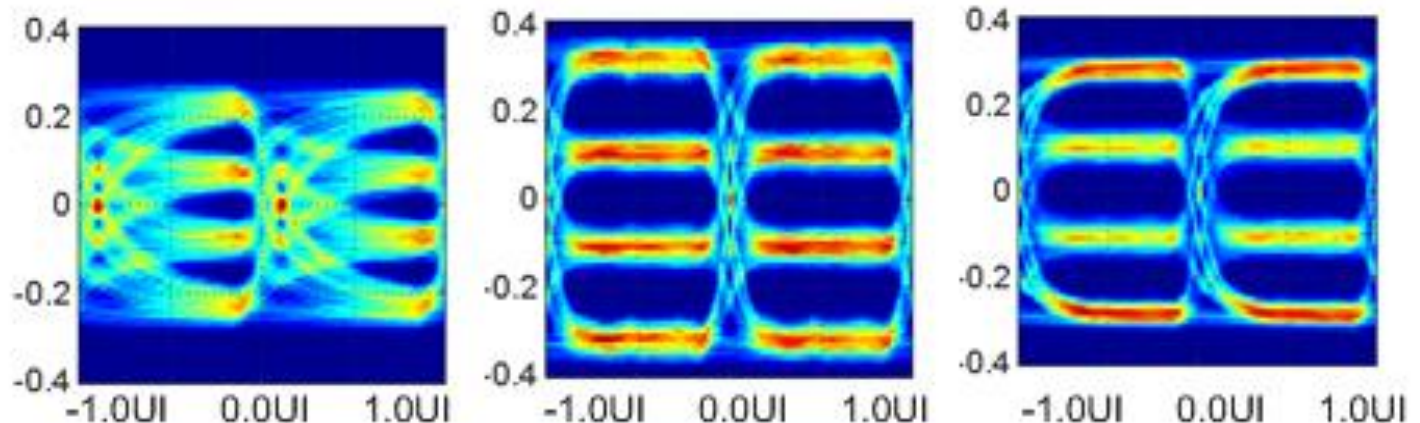
CTLE output (1st)

CTLE output (1st+3rd)

Signal swing:
 $\pm 30\text{mV}_{\text{dpp}}$



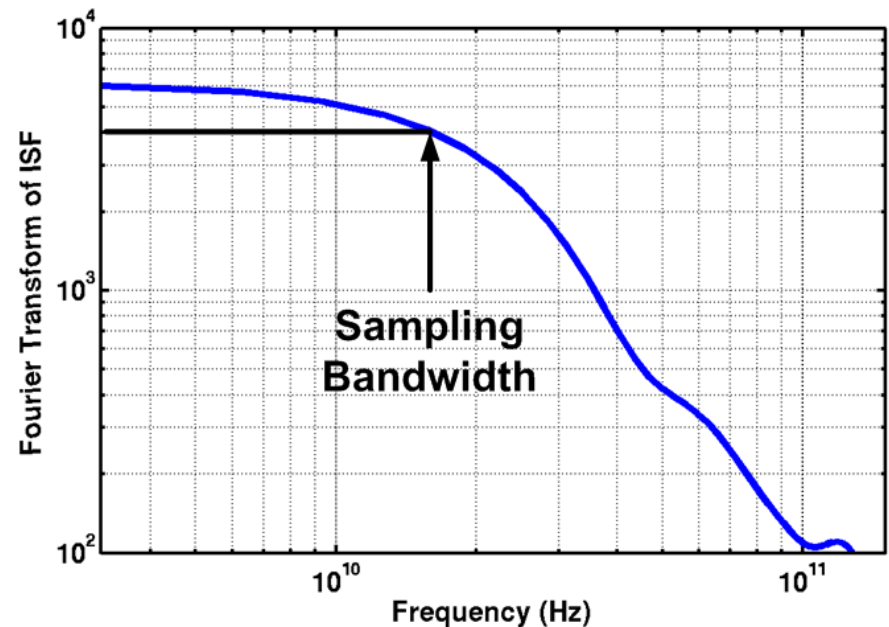
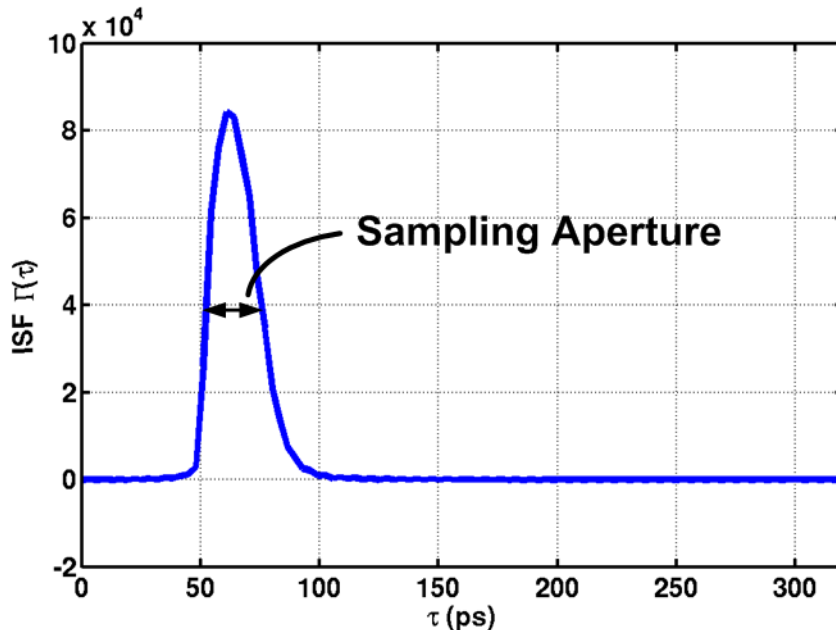
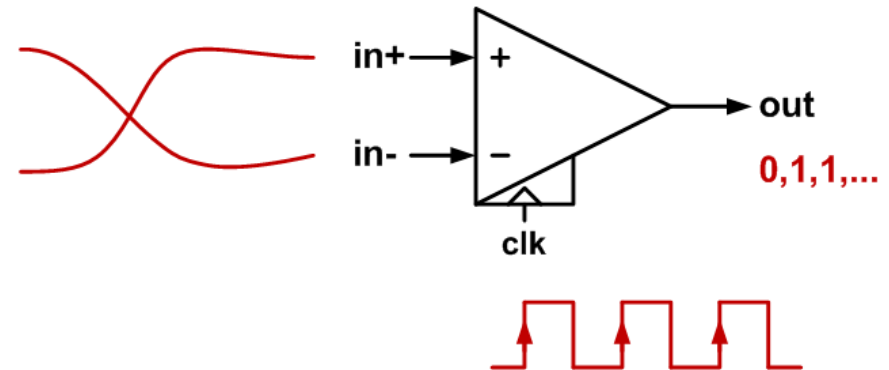
Signal swing:
 $\pm 300\text{mV}_{\text{dpp}}$



Finite-Aperture Comparator

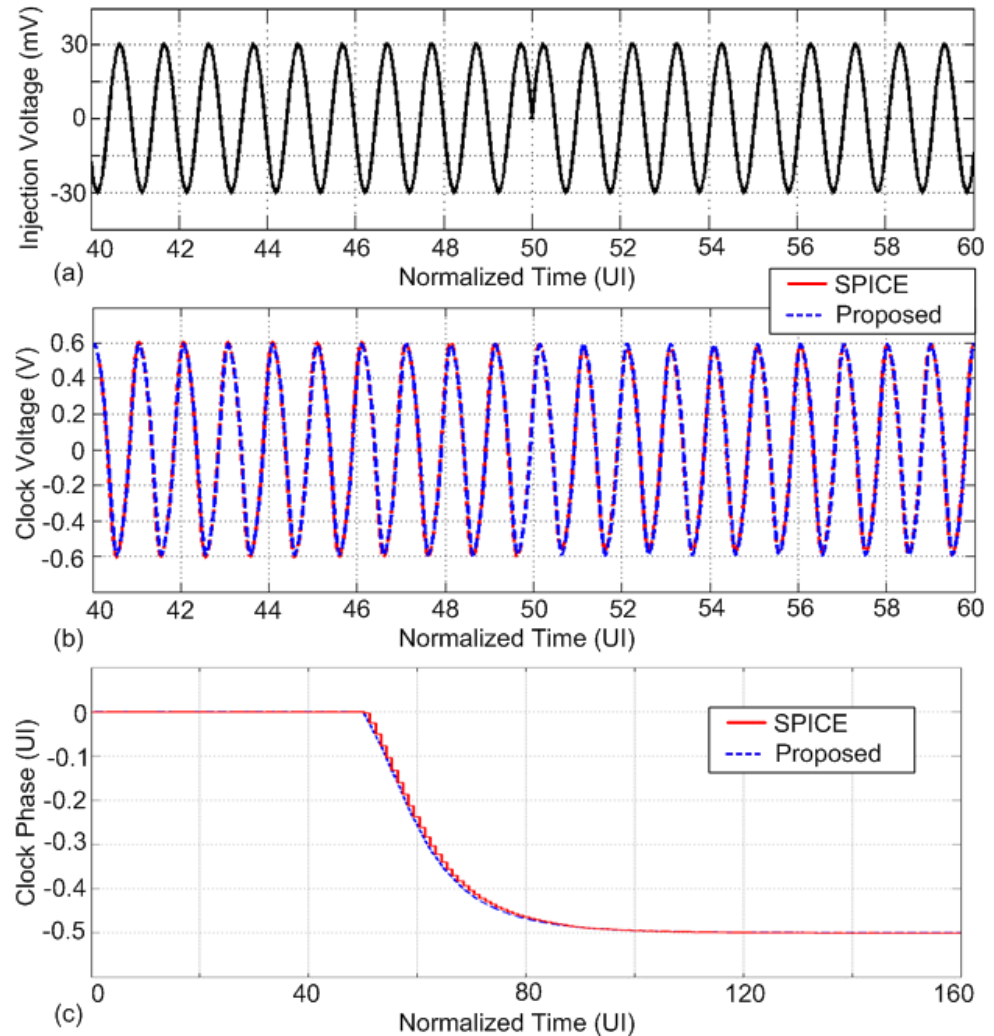
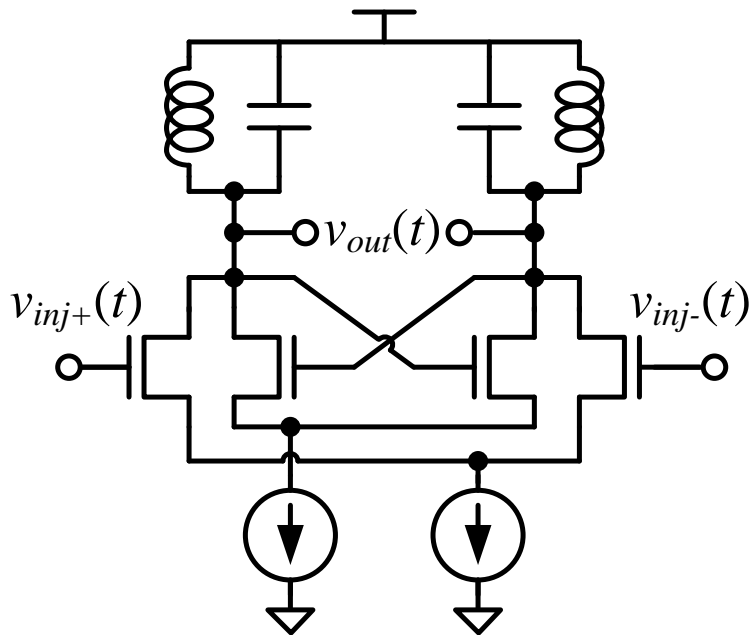
- Clocked comparators have:

- Finite sampling aperture
- Finite regeneration gain
- Random decision error



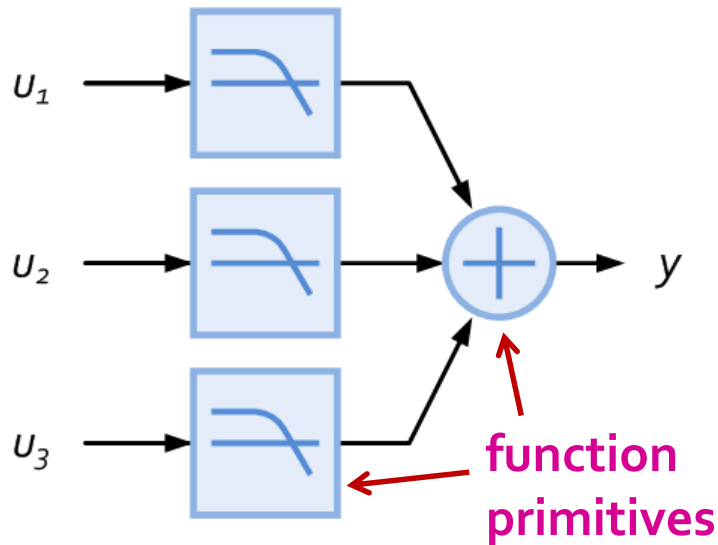
Injection-Locked Oscillator (ILO)

- PPV-based ILO modeling is possible with XMODEL
 - Using 'ilo' primitive



Block-Level vs. Circuit-Level Models

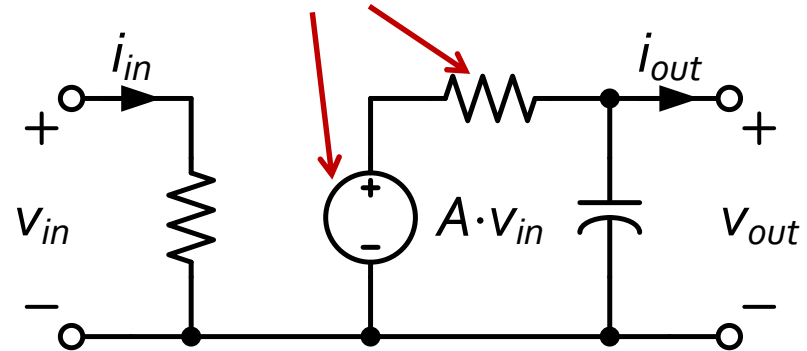
Block-Level Model (Signal-flow Model)



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

Circuit-Level Model (Conservative System Model)

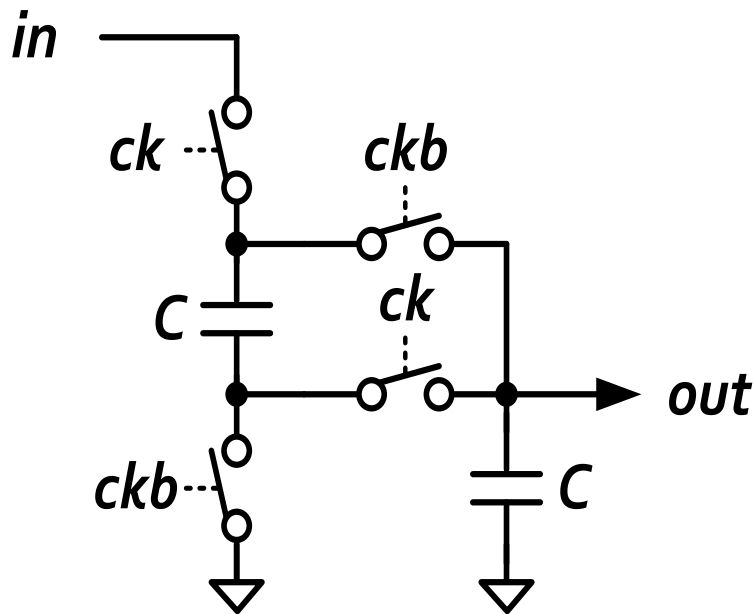
circuit primitives



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

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



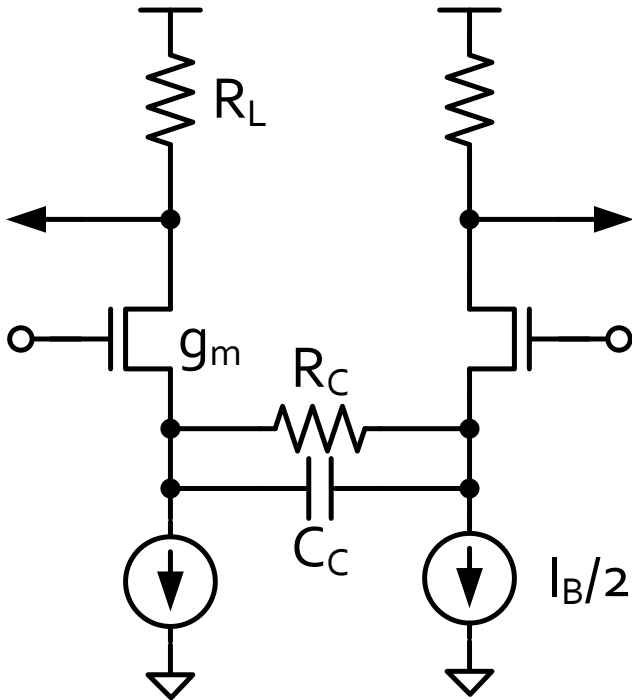
```

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));

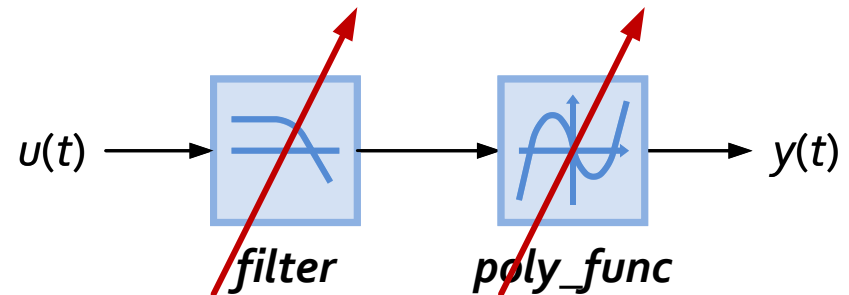
endmodule

```

Functional vs. Structural Modeling

• Functional modeling

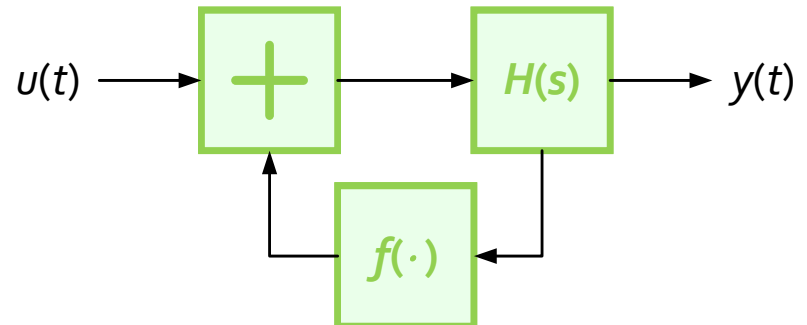
- Choose a template model based on the circuit's functions
- Calibrate its parameters so that the simulations match using **MODELFIT**



Parameter Calibration

• Structural modeling

- Characterize individual circuit components into models
- Models a system as a list of components and their connections using **MODELGEN**



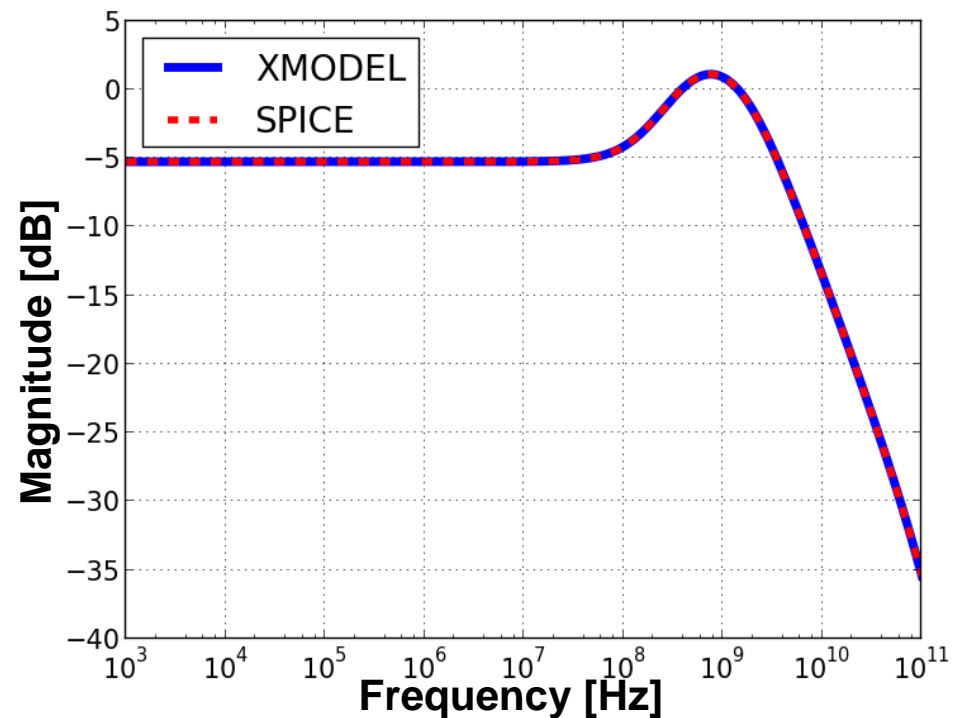
MODELFIT

- A Python script to extract the parameter values for a *filter* primitive from the AC simulation results

```
from xmulan import rowml
from modelfit import modelfit

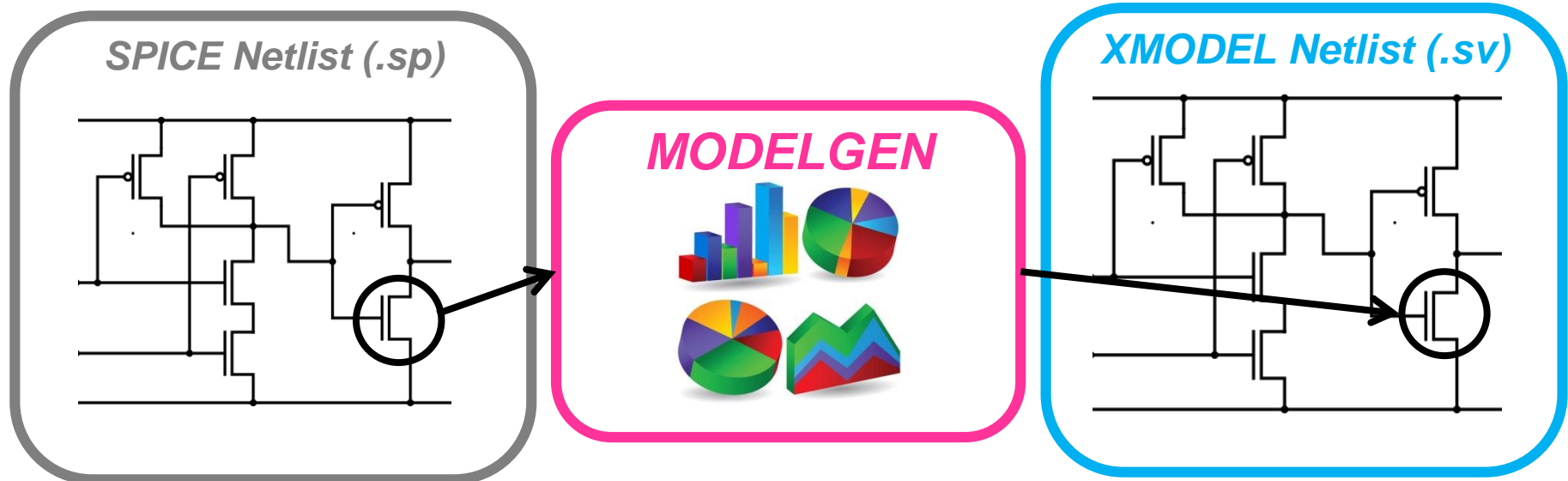
row = rowml()
row.readmeas("rx_eq.aco")
freq = row['HERTZ']
data = row['v:out']

modelfit(model="filter",
          in_=freq, out=data,
          filename="rxeq_tf.dat")
```



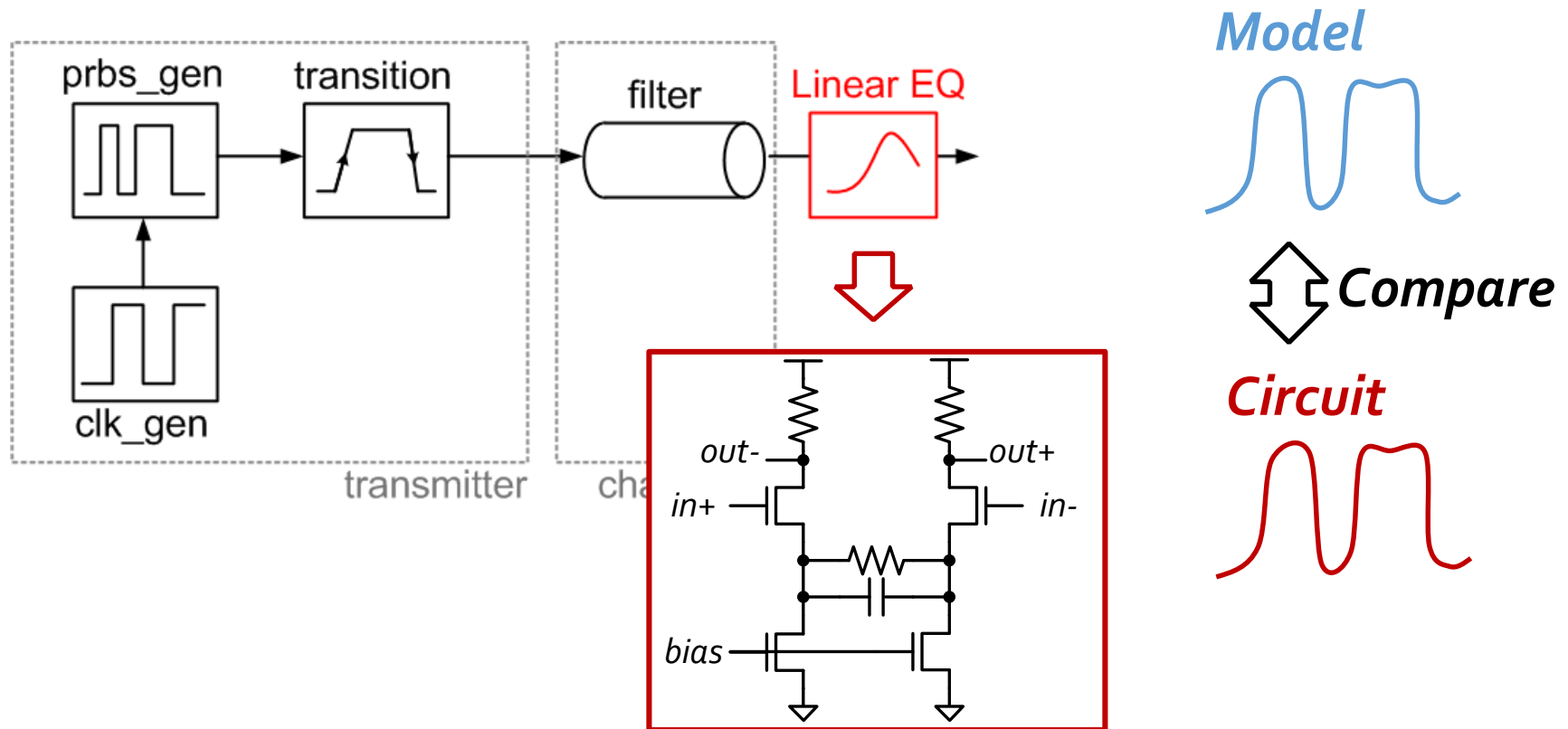
MODELGEN

- **MODELGEN** 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



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 Cadence Hierarchy Editor, you can easily generate netlists for XMODEL-SPICE co-simulation

The image displays two windows from the Cadence Virtuoso suite. The left window, titled 'Virtuoso® Schematic Editor L Editing: sandbox transceiver schematic Config: sandbox transceiver config', shows a schematic diagram of a transceiver. It includes a PRBS signal source, a TX block, a channel block, an RX block, and an RX-SLICER block. The schematic is connected to a clock source (CLK) and a frequency source (freq: 1e9Hz, RJ_rms: 10ps). The right window, titled 'Virtuoso® Hierarchy Editor: (sandbox transceiver config)', shows the hierarchy editor interface. It includes a 'Top Cell' section with fields for Library (sandbox), Cell (transceiver), and View (schematic). Below this is a 'Global Bindings' section with fields for Library List, View List, Stop List, and Constraint List. The 'Cell Bindings' section is open, showing a table of bindings and a context menu.

| Library | Cell | View Found | View To Use | Inherited View List |
|---------------|-------------|------------|-------------|---------------------|
| sandbox | rxreq | schematic | | |
| sandbox | transceiver | schematic | | |
| sandbox | tx | schematic | | |
| xmodel_blocks | channel | schematic | | |
| xmodel_prims | clk_gen | xmodel | | |
| xmodel_prims | compare | xmodel | | |
| xmodel_prims | filter | xmodel | | |
| xmodel_prims | prbs_gen | xmodel | | |
| xmodel_prims | transition | xmodel | | |

The context menu for the 'Cell Bindings' table includes the following options:

- Set Cell View
- Explain...
- Open...
- Open (Read-Only)...
- Add Stop Point
- Remove Stop Point
- Add Bind To Open...
- Remove Bind To Open...

The 'Inherited View List' column shows the following values:

- <none>
- schematic
- symbol
- xmodel
- Specify SPICE Source File...
- Specify Reference Verilog...
- verilog functional hspice...
- verilog functional hspice...

Summary

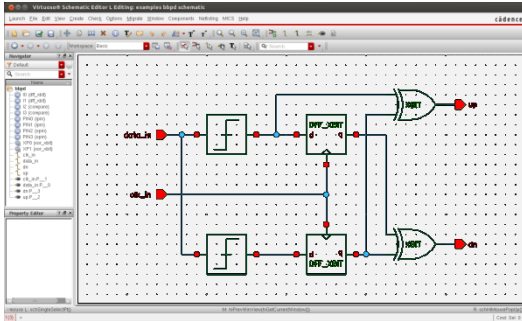
- ***XMODEL*** is a fast analog/mixed-signal simulator that works seamlessly with digital verification flows (SystemVerilog, UVM, ...)
 - Achieves both speed and accuracy (\Leftrightarrow Verilog-A, RNV)
 - Supports both block- and circuit-level models (\Leftrightarrow RNV)
- ***MODELFIT***, ***MODELGEN***, and ***GLISTER*** are convenient tools that help you compose analog models and verify them against SPICE

Getting Started with XMODEL in Cadence Virtuoso Environment

Scientific Analog, Inc.

XMODEL Overview

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**

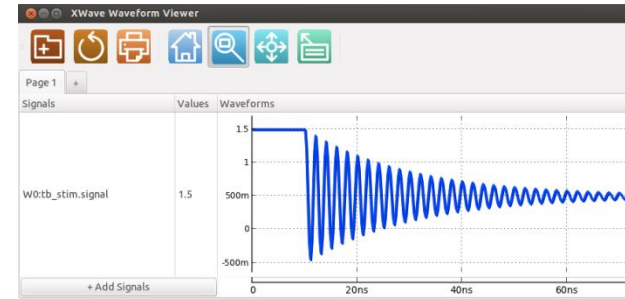


**SystemVerilog
Simulator
(VCS, Questa,
NC-Verilog)**



**XMODEL
Event-Driven
Simulation Engine**

Waveforms

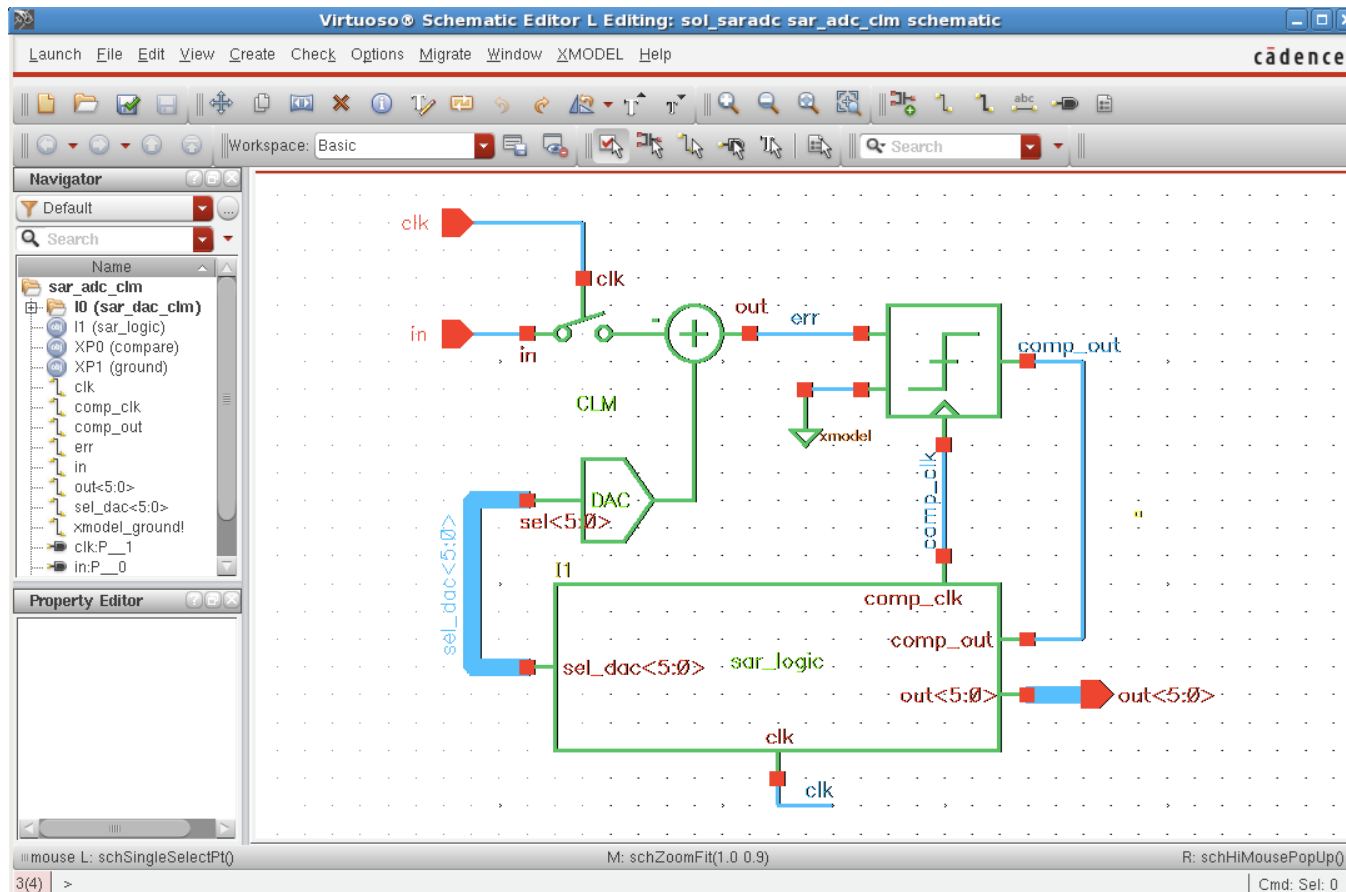


Simulation Results Reports

```
#  
# (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
```

Virtuoso Schematic Editor

- ▶ GLISTER enables you to create XMODEL models and simulate them in Cadence Virtuoso environments



Get Started

- ▶ To launch Cadence Virtuoso, type:

```
> cd ~/IDEC_CBDF/xmodel_dp11/cadence  
> virtuoso &
```

- ▶ It will open the Command Interpreter Window (CIW):



Virtuoso Setup for GLISTER

- Insert XMODEL libraries to your **cds.lib** file:

```
INCLUDE ${XMODEL_HOME}/cadence/etc/cds.lib
```

or:

```
DEFINE xmodel_prims    ${XMODEL_HOME}/cadence/xmodel_prims  
DEFINE xmodel_blocks  ${XMODEL_HOME}/cadence/xmodel_blocks
```

- And insert these lines to your **.cdsinit** file

```
printf("Loading XMODEL/Virtuoso Integration...\n")  
XMODEL_HOME = getShellEnvVar("XMODEL_HOME")  
loadContext( strcat( XMODEL_HOME "/cadence/etc/xmodel.cxt" ) )  
callInitProc( "xmodel" )
```

Virtuoso Setup for GLISTER (2)

- ▶ Copy the XMODEL registration file to your Cadence working directory (\$CDS_PROJECT):

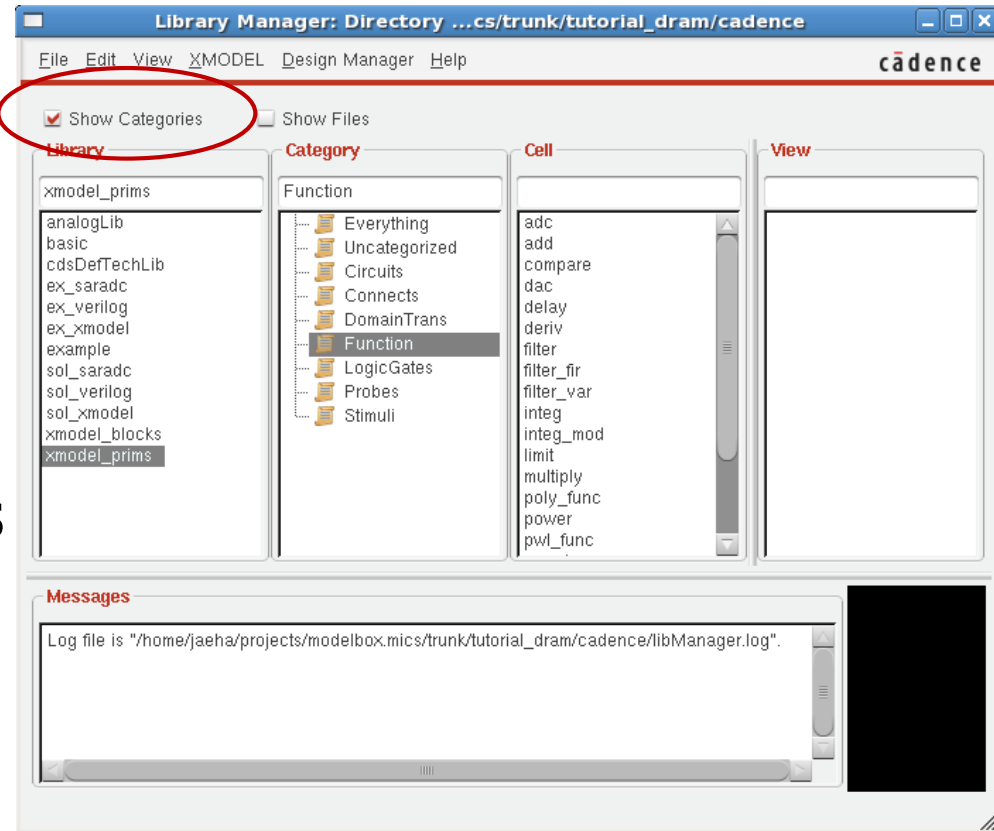
```
$ cp $XMODEL_HOME/cadence/etc/data.reg $CDS_PROJECT
```

- ▶ You don't need this step if your CAD team has installed the registration files in the Cadence installation directory
- ▶ Define \$XMODEL_SIMDIR environment variable to set where the model files are netlisted to and simulated

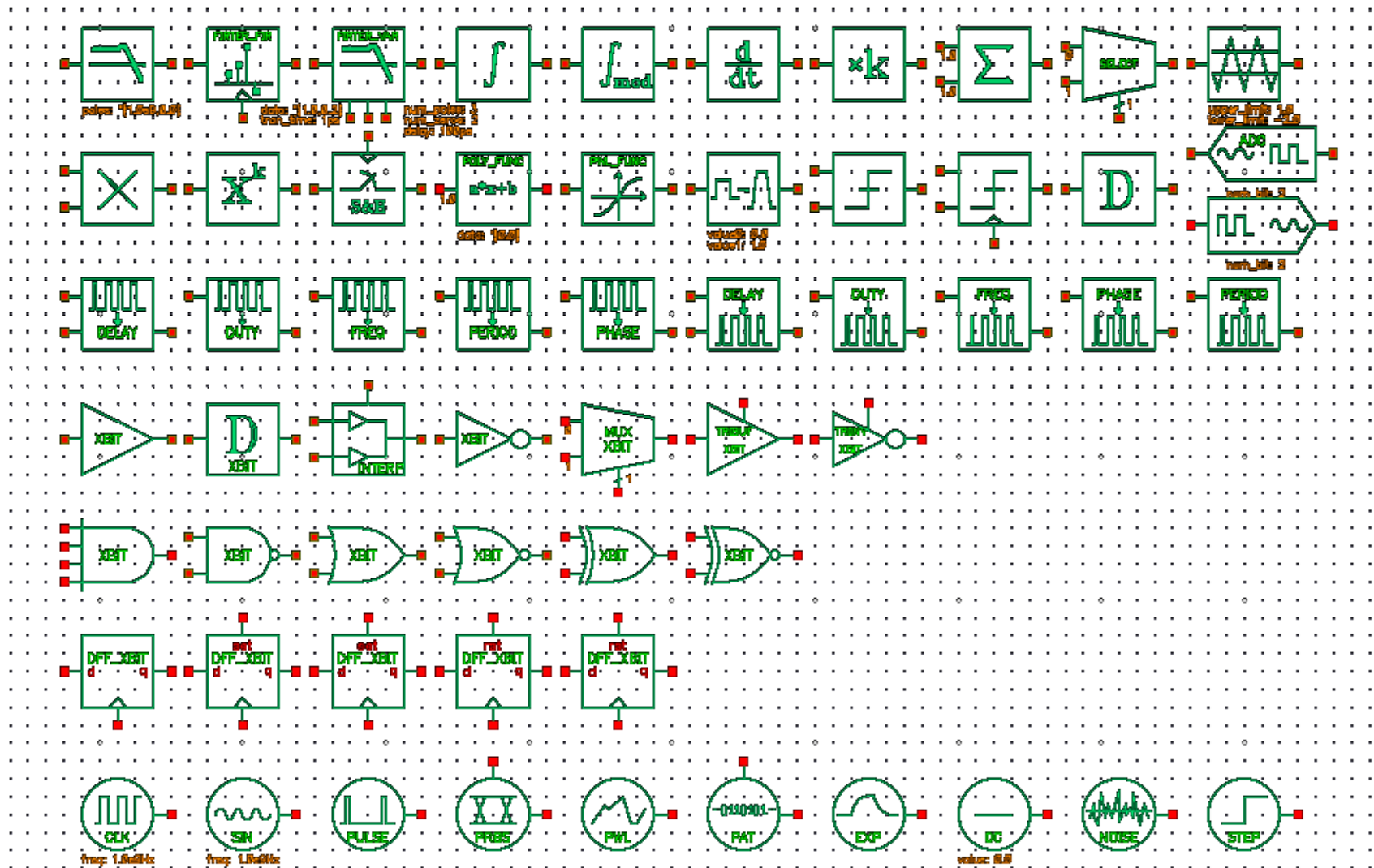
```
$ setenv XMODEL_SIMDIR $CDS_PROJECT/xmodel.sim
```

XMODEL Libraries

- ▶ Click **Tools -> Library Manager** in CIW
- ▶ Select **xmodel_prims** and browse the available XMODEL primitives
- ▶ Check **Show Categories** option to browse different categories of XMODEL primitives



XMODEL Primitive Symbols



XMODEL On-Line Documentations

- ▶ You can access the on-line documentation by passing “-h” option to the “xmodel” command:

```
$ xmodel -h
```

```
...
```

```
list of help topics:
```

| | |
|----------|---------------------|
| function | Functions |
| gate | Logic gates |
| stim | Stimulus generators |
| meas | Probes |
| vdt | Domain translators |
| connect | Connectors |

XMODEL On-Line Documentations (2)

- And use '-h TOPIC' to get a list of primitives of each category:

```
$ xmodel -h stim
```

```
=====
```

```
TOPIC stim
```

```
=====
```

The XMODEL stimulus generator primitives provide means to generate various stimulus waveforms both in analog and digital format.

list of stimulus generator primitives:

| | |
|-----------|-------------------------------------|
| clk_gen | A digital clock generator. |
| dc_gen | Analog DC generator |
| exp_gen | Analog exponential signal generator |
| noise_gen | Noise generator |

...

XMODEL On-Line Documentations (3)

- ▶ Finally, use '-h PRIMITIVE' to get the documentation on each primitive:

```
$ xmodel -h sin_gen
```

```
=====
```

```
PRIMITIVE sin_gen
```

```
=====
```

```
Analog sinusoid generator
```

The 'sin_gen' primitive generates a sinusoidal signal that can optionally be exponentially decaying or frequency/amplitude-modulated.

The generated stimulus waveform $V(t)$ is defined as follows:

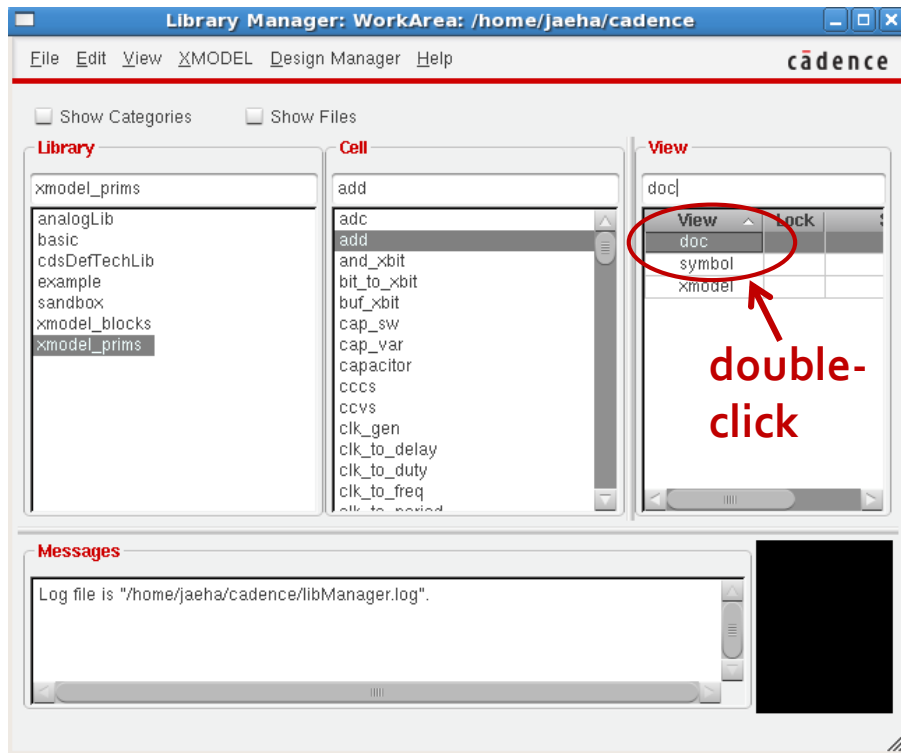
for $t < \text{delay}$:

$$V(t) = \text{offset} + \text{amp} * \text{AM_offset} * \sin(\text{init_phase})$$

...

XMODEL On-Line Documentations (4)

- ▶ The alternate way is to open the “doc” view of each XMODEL primitive from the Library Manager



The screenshot shows the Mozilla Firefox browser displaying the XMODEL documentation for the 'add' primitive. The page title is 'XMODEL'. The main content area shows the primitive name 'add' in a large pink font, followed by a description: ': A multiple-input adder for xreal-typed signals.' To the right of the text is a schematic diagram of the 'add' primitive, which is a blue square with a summation symbol (Σ) inside. It has two input terminals on the left, each labeled '1.0', and one output terminal on the right. Below the description, there is a paragraph explaining the primitive's function: 'The 'add' primitive produces an xreal signal which is a sum of multiple xreal-typed inputs. Using the parameters 'num_in' and 'scale', one can set the number of input signals to be added and their respective scale factors.'

Input/Output Terminals

| Name | I/O | Type | Description |
|------|--------|-------|---------------|
| out | output | xreal | output signal |
| in | input | xreal | input signals |

Parameters

| Name | Type | Default | Unit | Description |
|--------|------------|------------|------|---------------------|
| num_in | integer | 2 | None | number of inputs |
| scale | real array | {1.0, 1.0} | None | input scale factors |

XMODEL Off-Line Documentations

- ▶ The off-line documentations are also available and located in:

`${XMODEL_HOME}/doc`

- ▶ `$XMODEL_HOME` is the XMODEL installation directory
- ▶ You can access them in Linux environment with a PDF viewer
 - ▶ For instance, to open the reference manual, type:

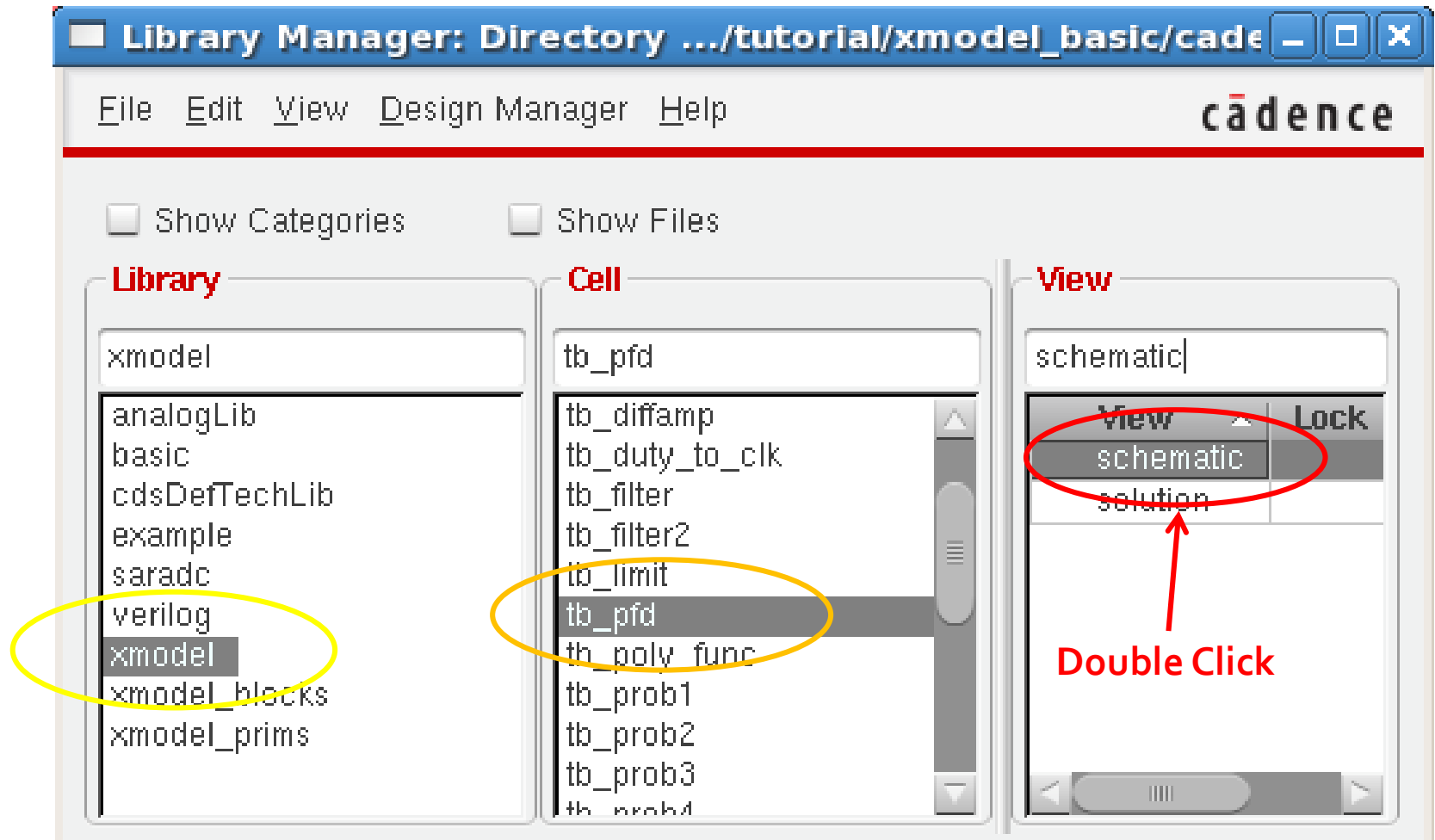
```
$ evince ${XMODEL_HOME}/doc/XMODEL_Reference_Manual.pdf
```

Conventions

- ▶ Each cellview in Cadence Virtuoso is denoted by three fields (e.g. “xmodel:tb_pfd:schematic”)
 - ▶ Library name: **xmodel**
 - ▶ Cell name: **tb_pfd**
 - ▶ View name: **schematic**
- ▶ In this tutorial, you will also use the following prefixes for cell names:
 - ▶ Testbenches: **tb_***
 - ▶ Stimuli generators: **stm_***
 - ▶ Output monitors: **mon_***

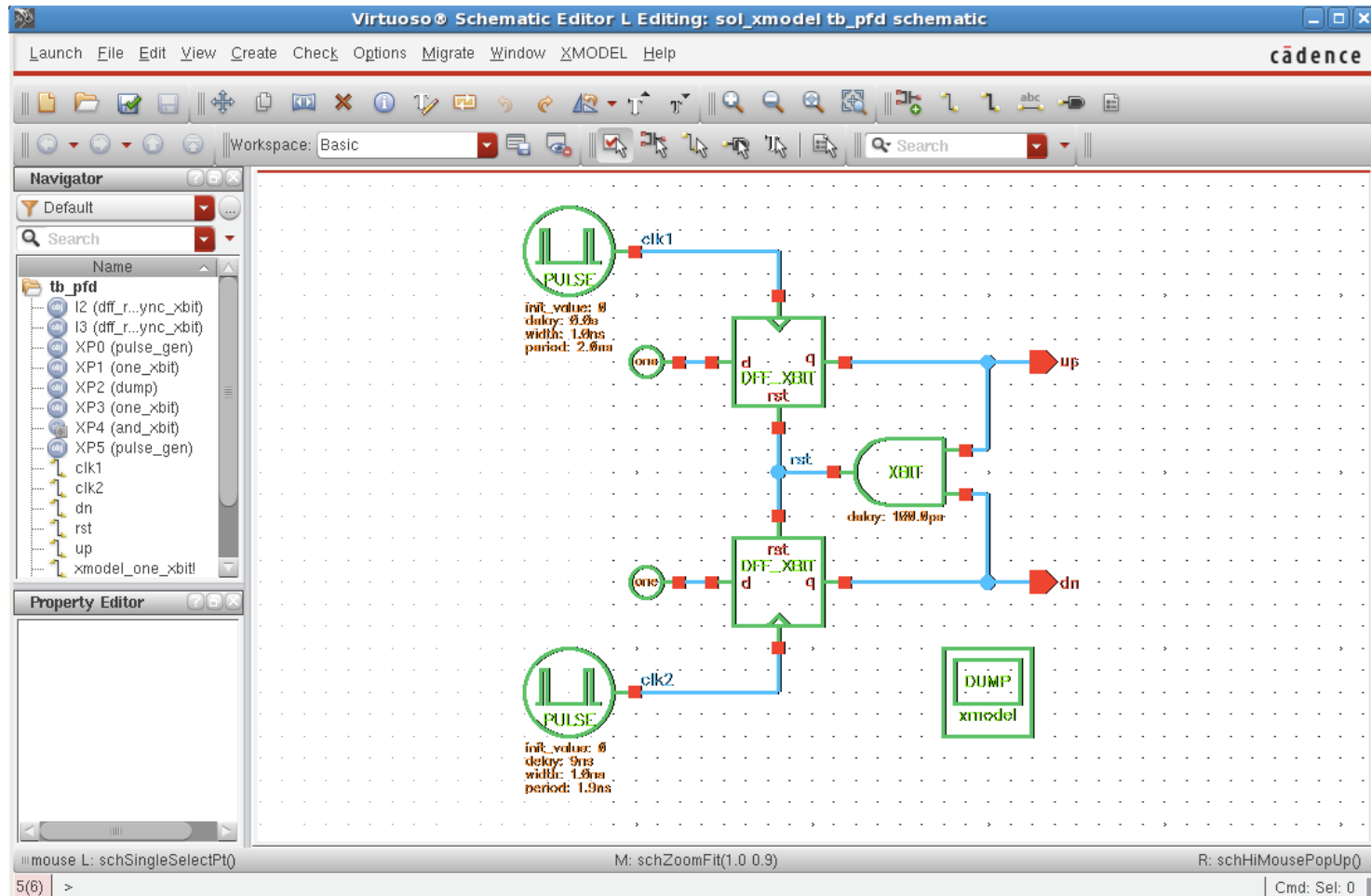
Open Schematic Editor

- Choose library, cell, and double-click the view



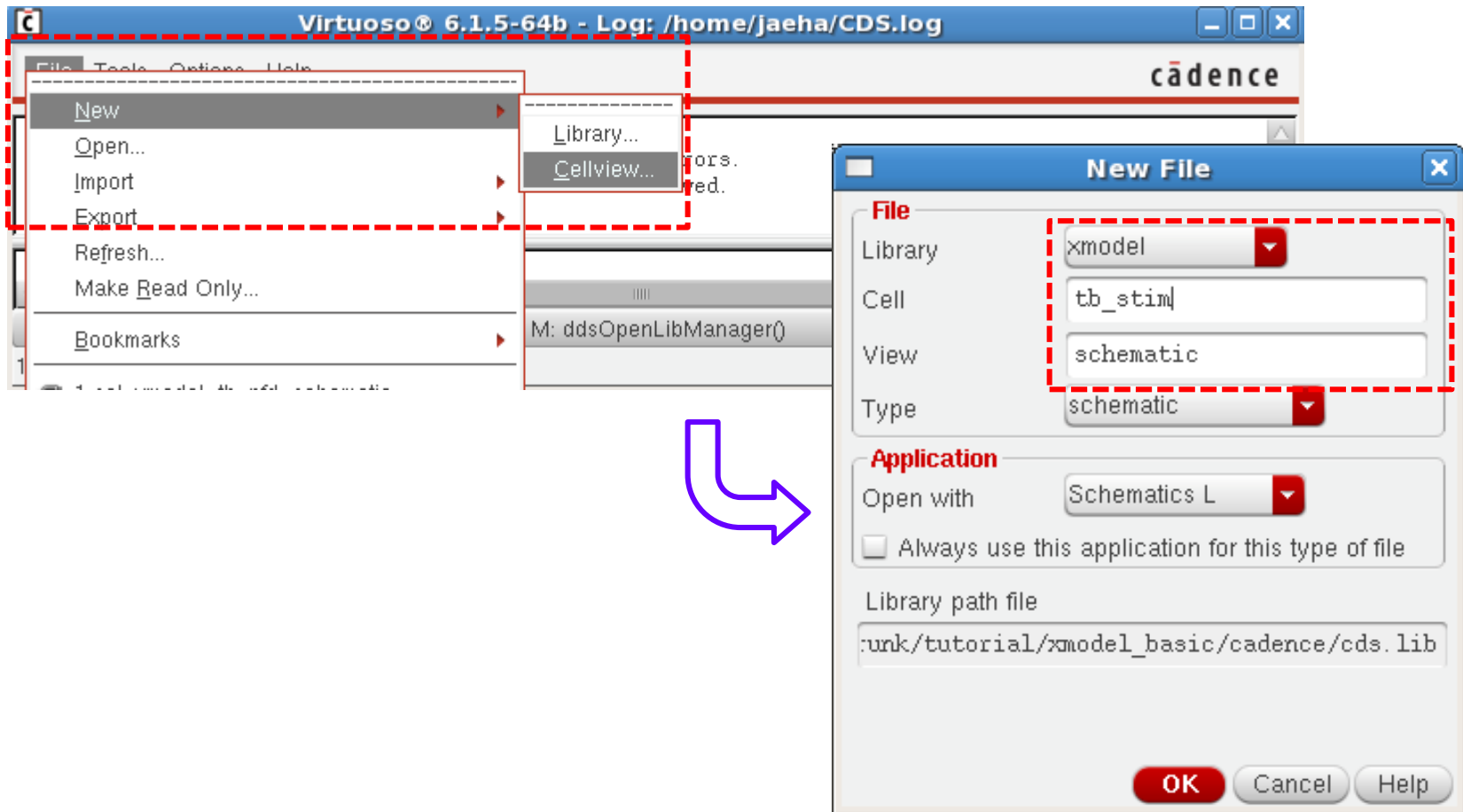
Schematic Editor Window

- Open xmodel:tb_pfd:schematic cellview



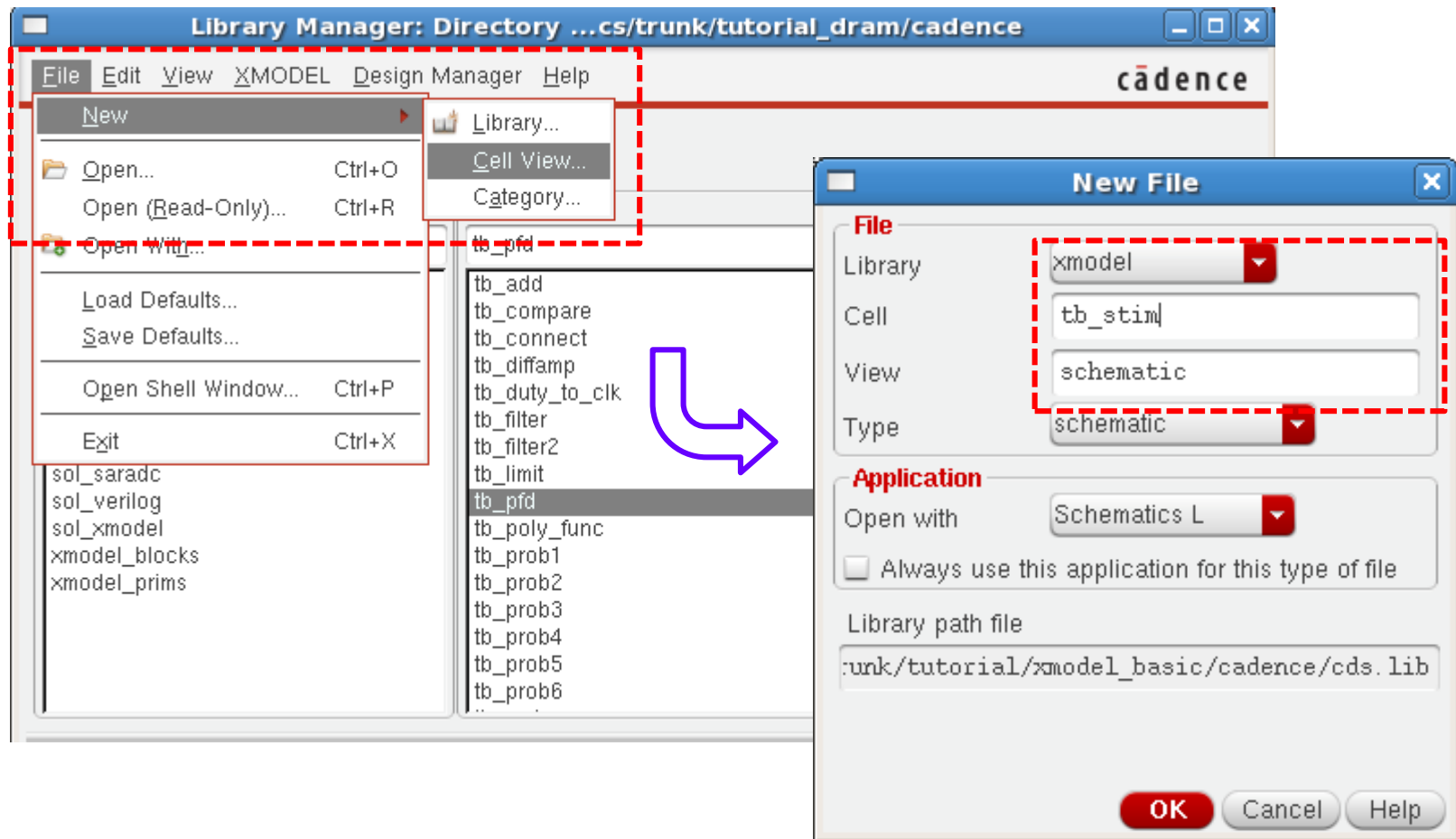
Creating a New Cellview

- From Command Interpreter Window (CIW):



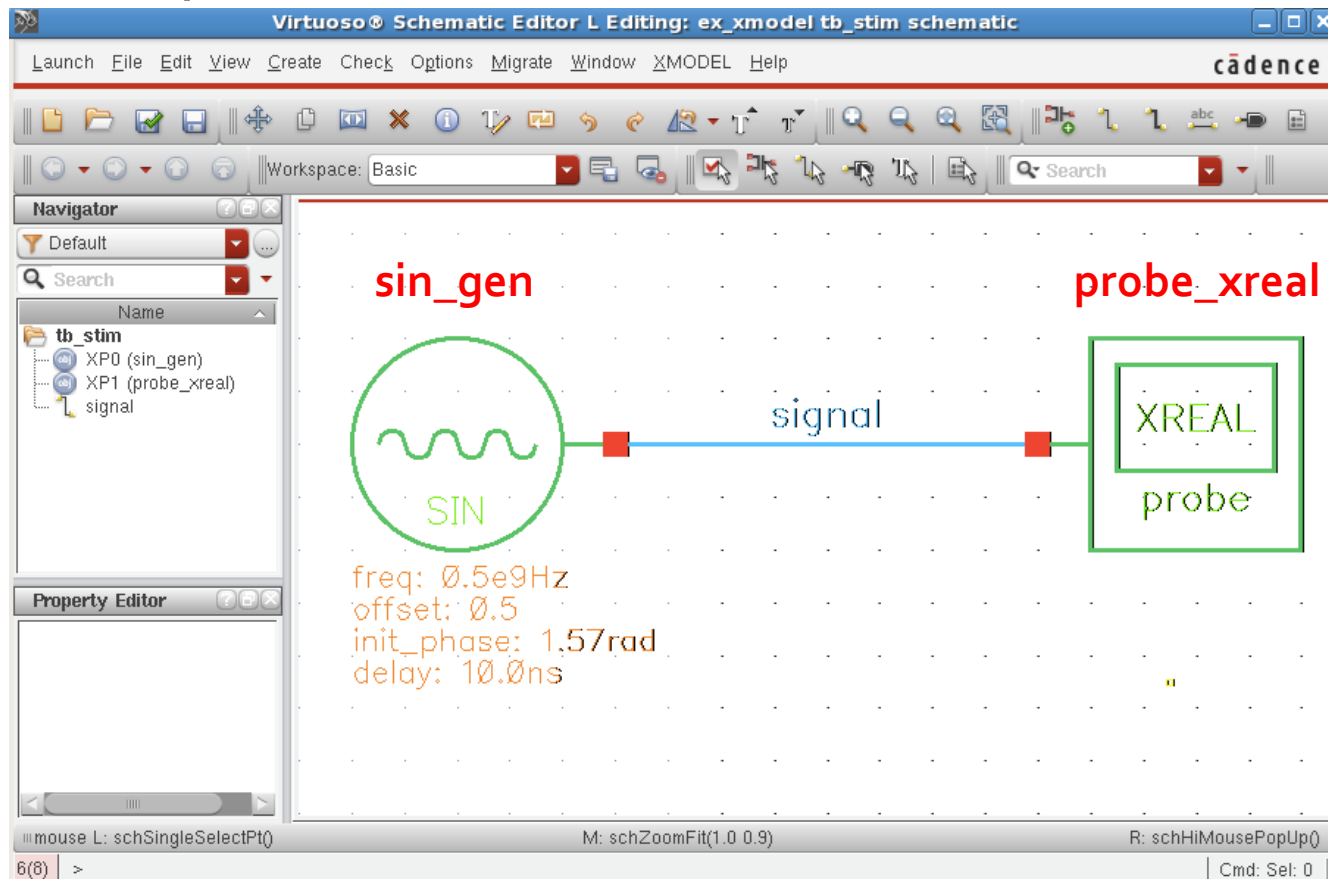
Creating a New Cellview (2)

► From Library Manager:



Very First Exercise

- ▶ Create a new cellview **xmodel:tb_stim:schematic**
- ▶ And compose a model schematic as below:



Add Instances

- ▶ To add an instance, click Create->Instance or type “i”
 - ▶ Select library, cell, and view
 - ▶ Enter the parameter values
- ▶ Then move the cursor on the schematic editor window and place the instance

**Parameter values
for sin_gen instance**

Add Instance

Library: xmodel_prims
Cell: sin_gen
View: symbol
Names:

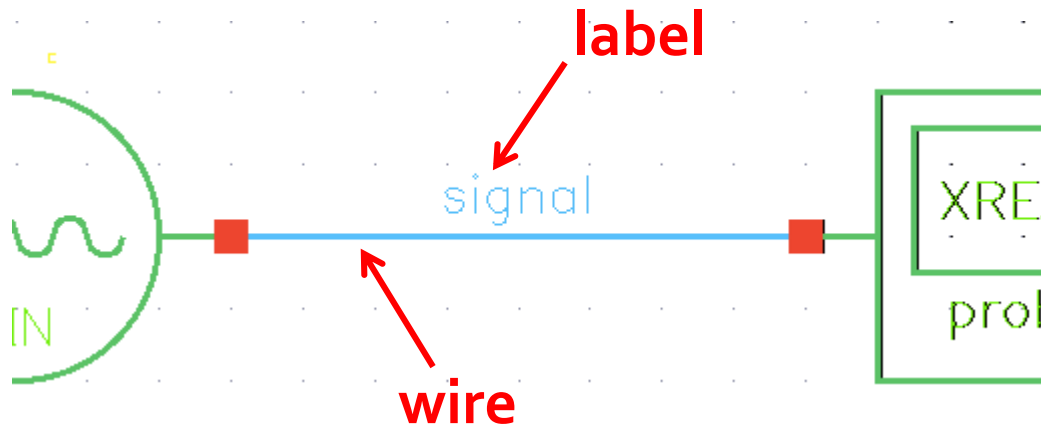
☒ Add Wire Stubs at:
☐ all terminals ☒ registered terminals only

Array: Rows: 1 Columns: 1

| | | |
|------------------------|-------|--------------|
| Offset | 0.5 | 0.5 |
| Amplitude | 1.0 | 1.0 |
| Frequency | 0.5G | 0.5G |
| Initial Delay | 10.0n | 10n |
| Damping Factor (1/sec) | 0.05G | 0.05G |
| Initial Phase | 1.57 | 1.57 |
| AM Offset | 1.0 | |
| AM Amplitude | 0.0 | |
| AM Frequency | 0.0 | |
| FM Index | 0.0 | |
| FM Frequency | 0.0 | |

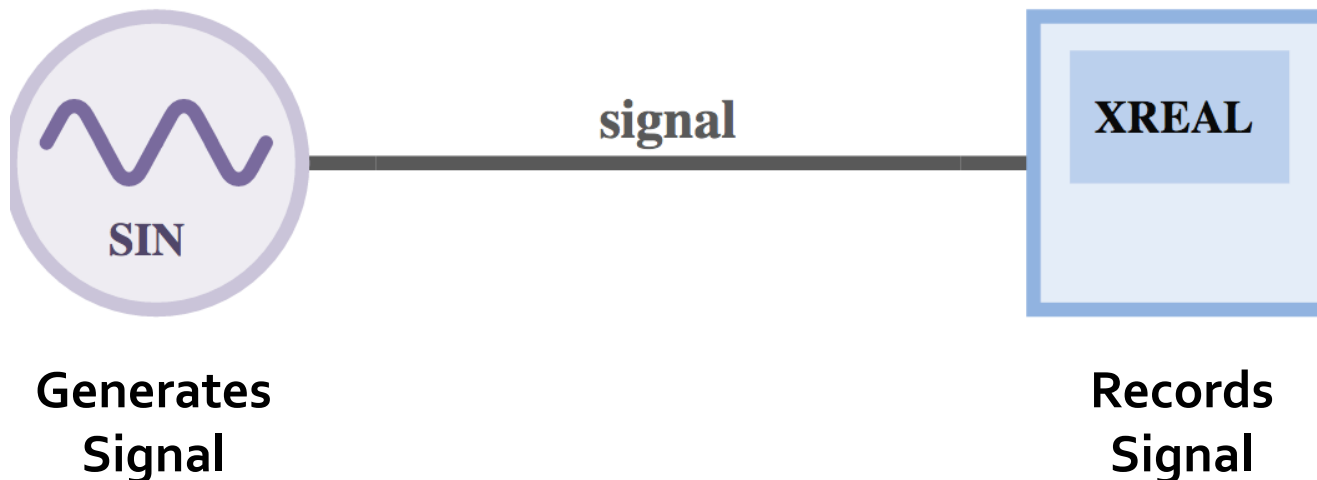
Connecting and Labeling

- ▶ To add a wire, click Create->Wire or type "W"
- ▶ Click one point and then click the other
- ▶ To add a label, click Create->Wire Name or type "L"
- ▶ Enter a name and place the label on the wire



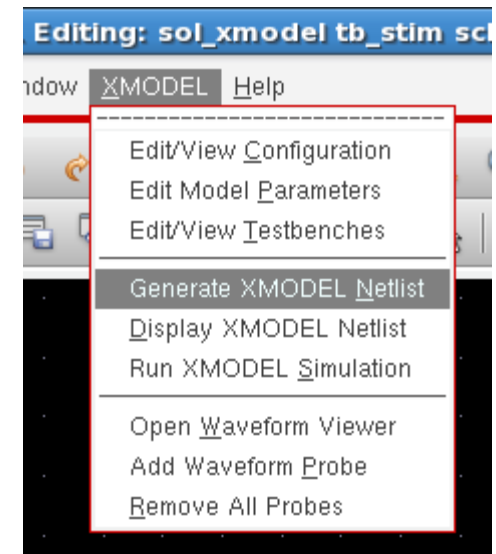
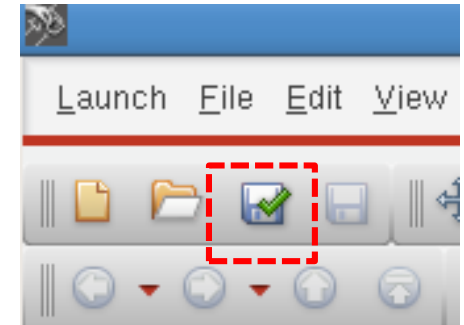
tb_stim Explained

- ▶ What did we just make?
- ▶ *sin_gen* generates a (damping) sinusoidal signal
- ▶ *probe_xreal* records the waveform into a file

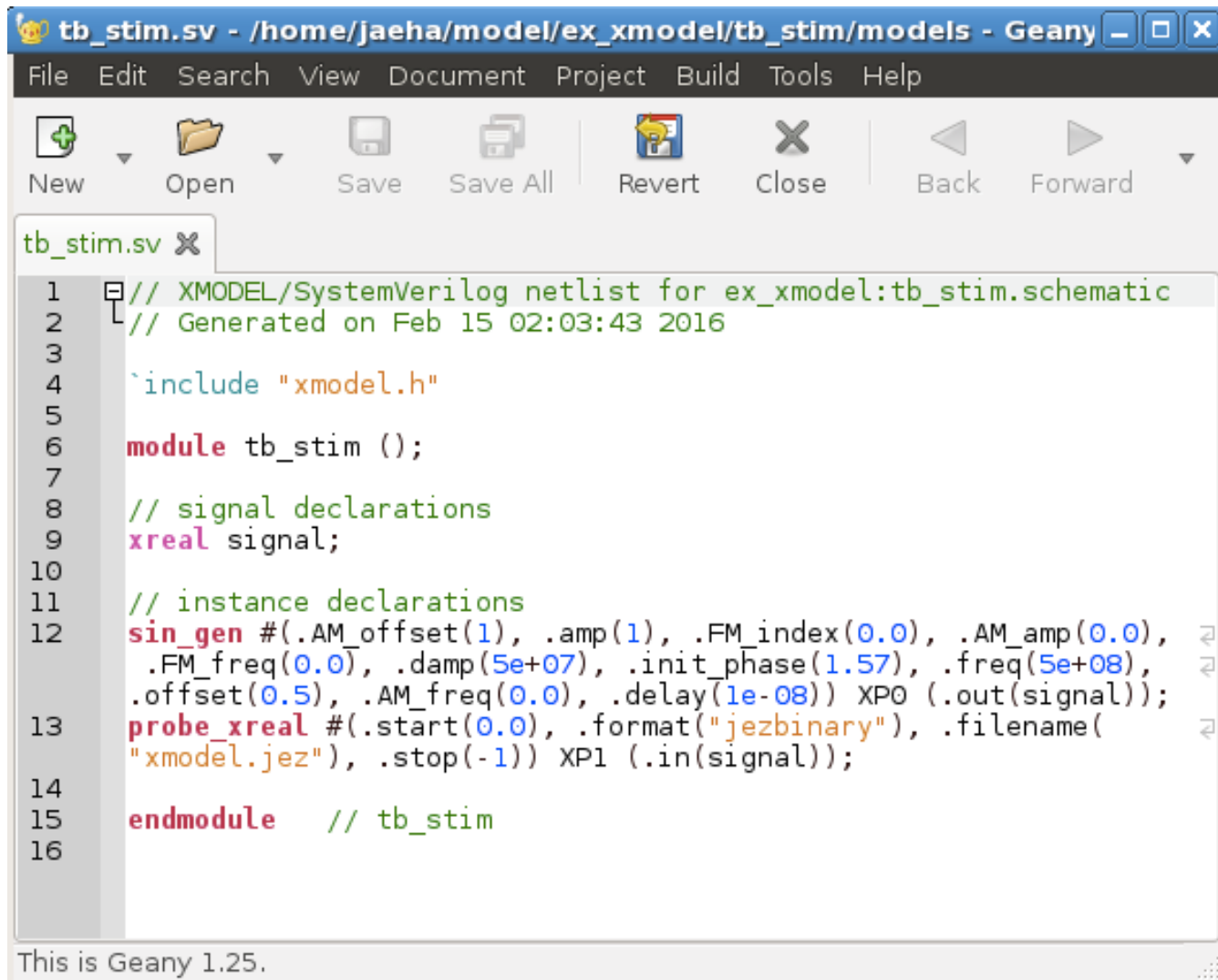


Netlisting and Simulation

- ▶ First, save your design by clicking ***Check-and-Save*** button on the toolbar
- ▶ Using the XMODEL menu, you can:
 - ▶ Generate XMODEL netlist
 - ▶ Display XMODEL netlist
 - ▶ And run XMODEL simulation



XMODEL Netlist for *tb_stim*



```
tb_stim.sv x
1 // XMODEL/SystemVerilog netlist for ex_xmodel:tb_stim.schematic
2 // Generated on Feb 15 02:03:43 2016
3
4 `include "xmodel.h"
5
6 module tb_stim ();
7
8 // signal declarations
9 xreal signal;
10
11 // instance declarations
12 sin_gen #(.AM_offset(1), .amp(1), .FM_index(0.0), .AM_amp(0.0),
    .FM_freq(0.0), .damp(5e+07), .init_phase(1.57), .freq(5e+08),
    .offset(0.5), .AM_freq(0.0), .delay(1e-08)) XP0 (.out(signal));
13 probe_xreal #(.start(0.0), .format("jezbinary"), .filename(
    "xmodel.jez"), .stop(-1)) XP1 (.in(signal));
14
15 endmodule // tb_stim
16
```

This is Geany 1.25.

tb_stim.sv Explained

```
`include "xmodel.h"
```

Include XMODEL header file

```
module tb_stim();
```

Module declaration

```
    xreal signal;
```

Variable Declaration

```
    sin_gen #(.delay(10e-9), .offset(0.5), .damp(0.05e9),  
              .amp(1.0), .freq(0.5e9), .init_phase(1.57))
```

```
        my_gen(signal);
```

```
    probe_xreal #(.filename("xmodel.jez"))
```

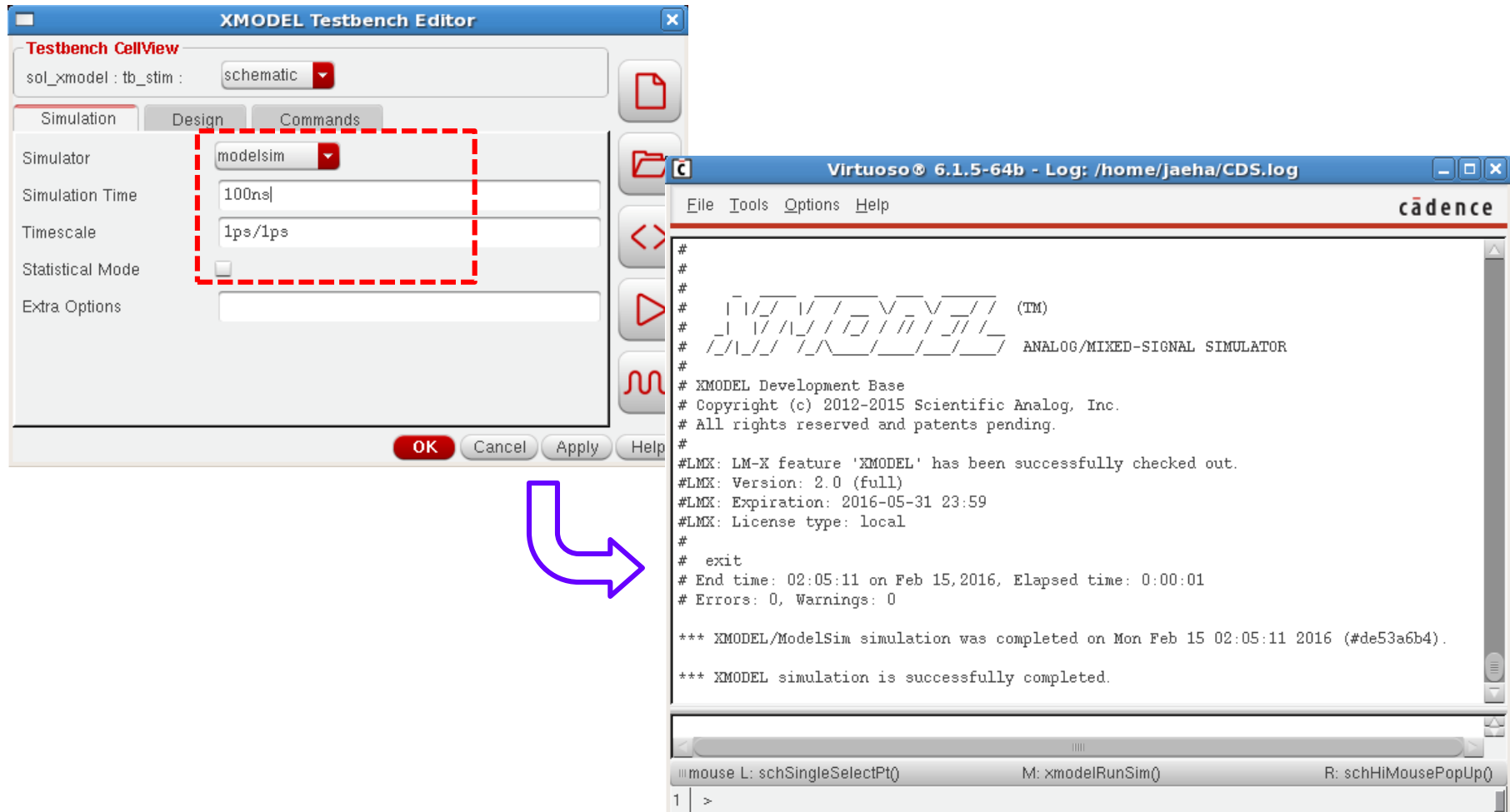
```
        my_probe(signal);
```

XMODEL primitive instances

```
endmodule
```

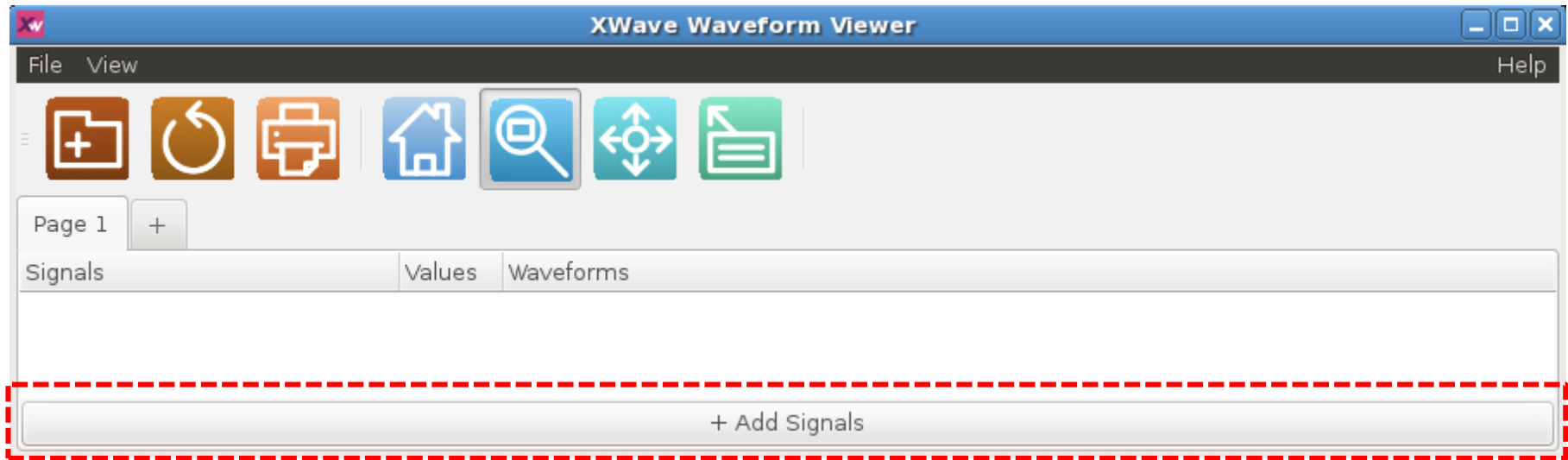
XMODEL Simulation

► XMODEL->Run XMODEL Simulation



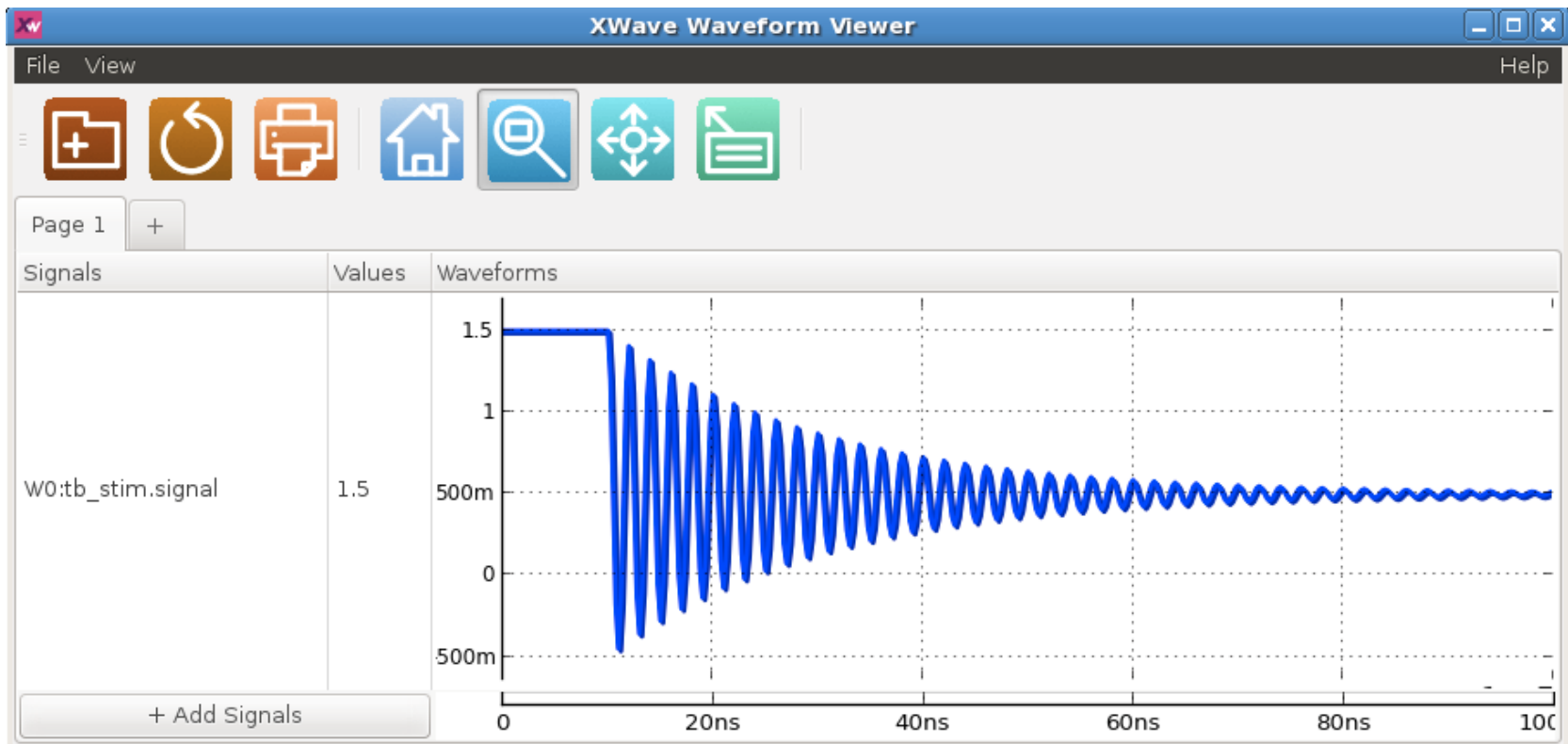
Viewing Waveforms

- ▶ Click XMODEL->Open Waveform Viewer and start XWAVE
- ▶ Click "+ Add Signals" button on the bottom to select a signal to display



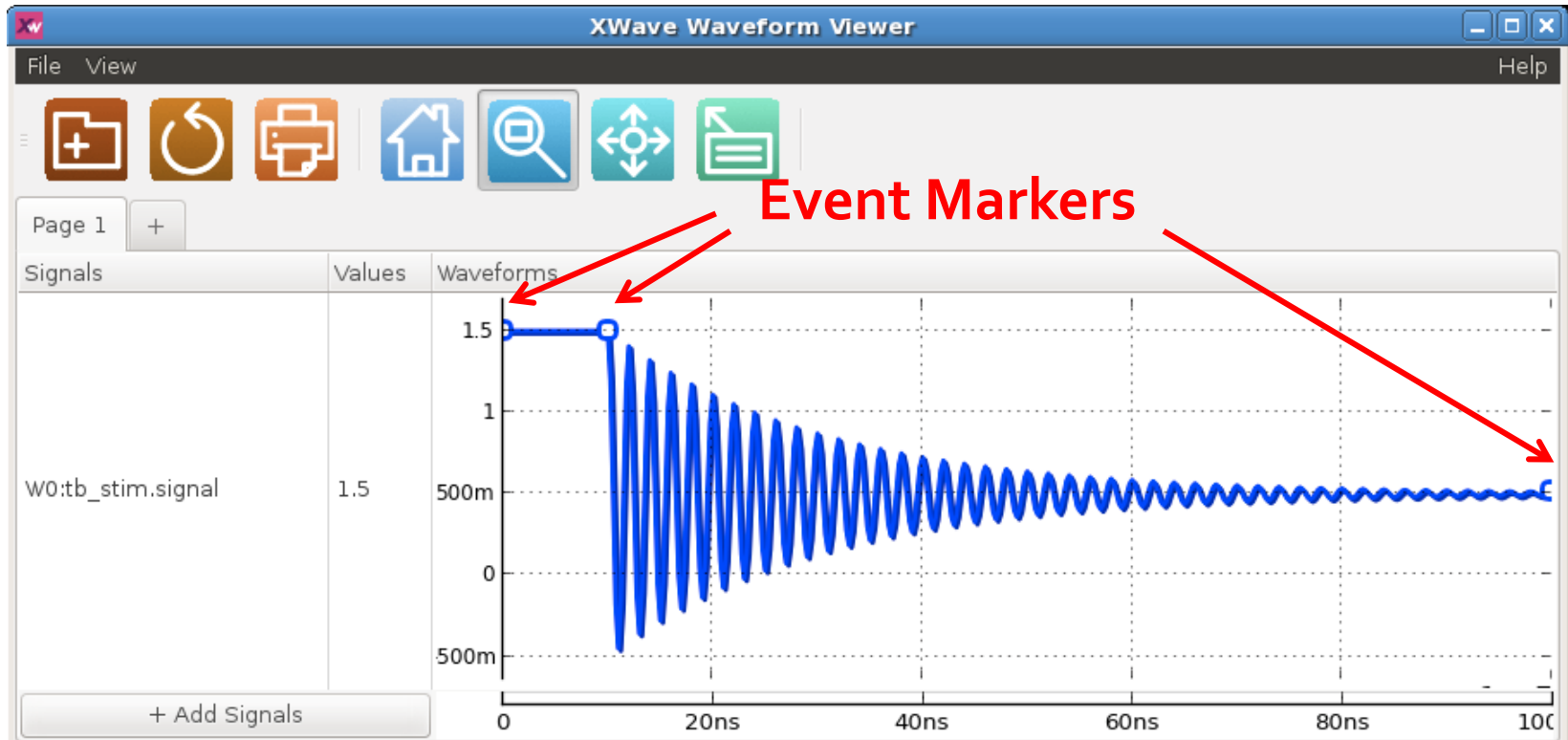
Probing Signals on Schematics

- ▶ You can also select signals on the schematic window by clicking XMODEL->Add Waveform Probe and clicking the wires



Toggling Event Markers

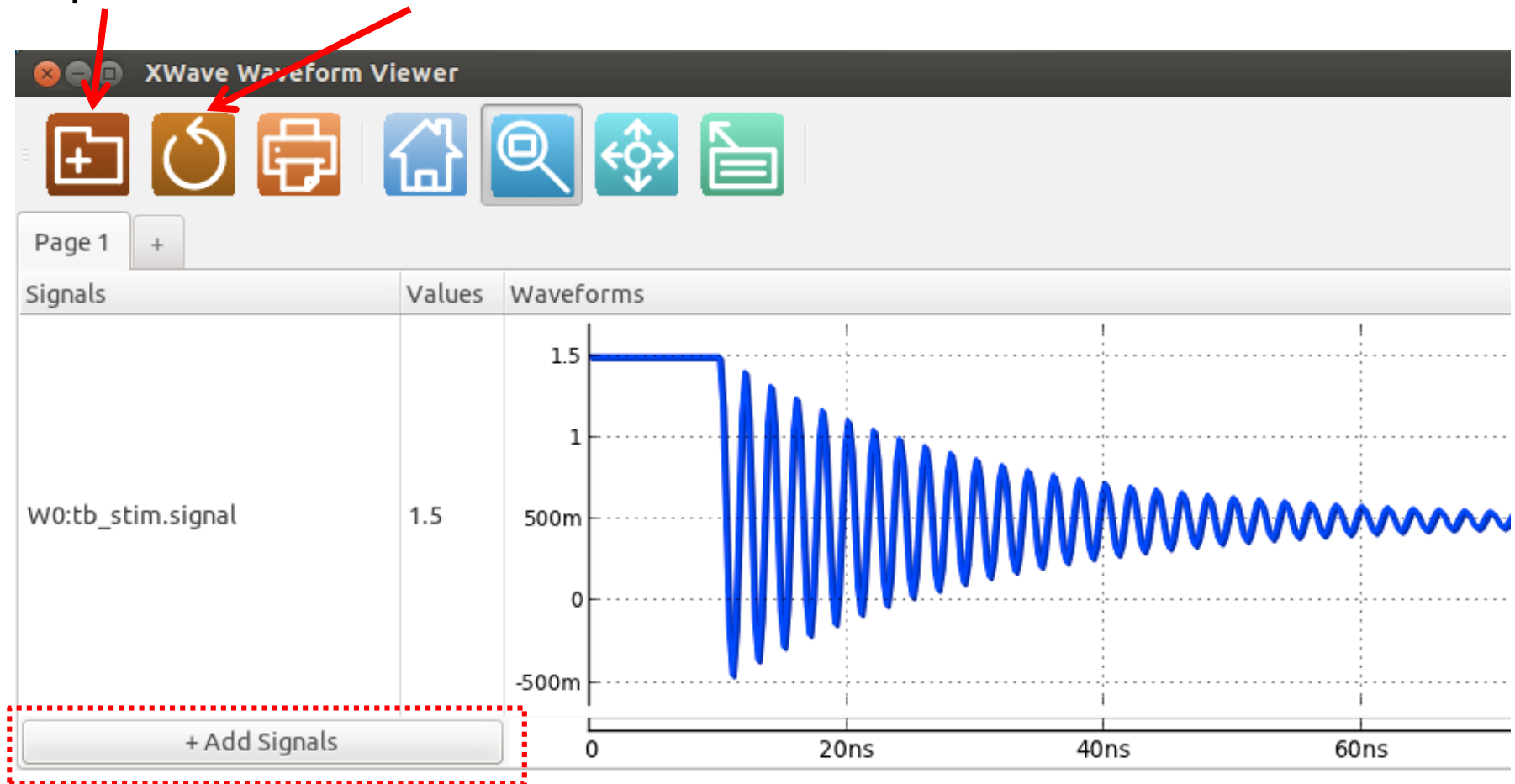
- ▶ Press “M” key to display the event markers
- ▶ These are the only points at which XMODEL performs computation – it’s why it is so fast!



XWave Commands

Open file

Reload file



Add signals

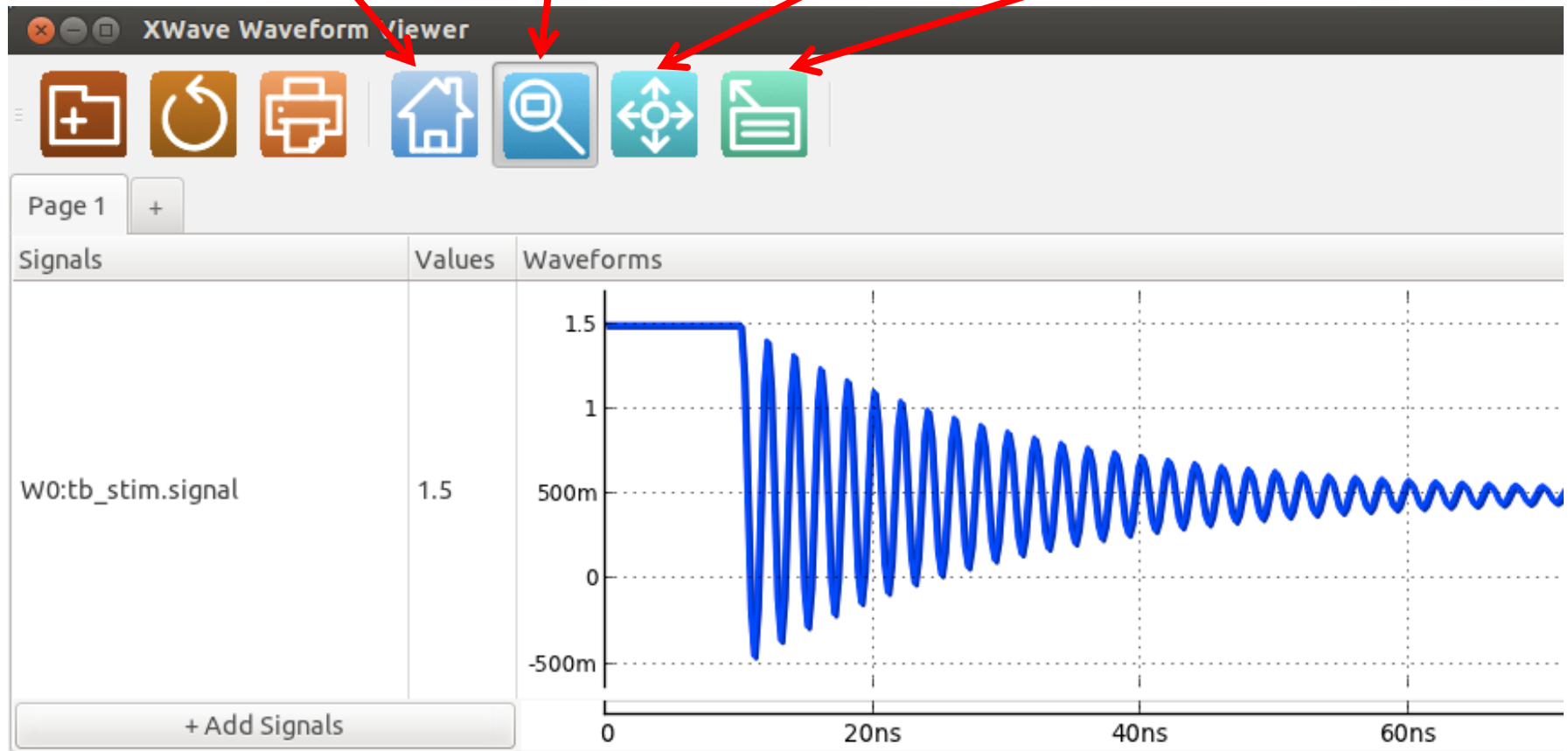
XWave Commands (2)

Zoom to fit

Zoom area

Panning

Cursor



- Try zooming in to see if you see any resolution effects

Running XMODEL in Command Line

- ▶ What you just did in this exercise is equivalent to:

```
$ cd $XMODEL_SIMDIR/xmodel/tb_stim  
$ xmodel -f sources.f --top tb_stim --simtime 100ns
```

and

```
$ xwave xmodel.jez
```

- ▶ For more information on the command-line usage, please refer to the *XMODEL User's Guide*