scientific analog

Putting It All Together: CTLE Modeling Example

Scientific Analog, Inc.

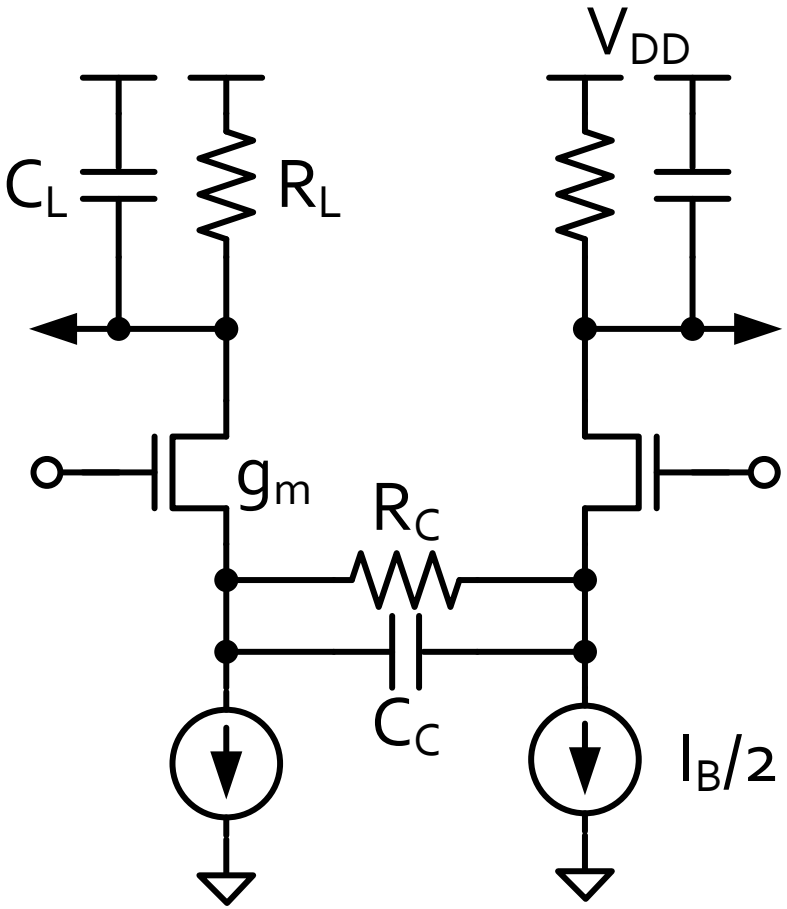
July 2017

Modeling Example: CTLE

- Continuous-time linear equalizer (CTLE) is a key building block in wireline receivers
  - Intent is a linear filter but can be nonlinear (e.g. gain compression)
- Three different ways to model CTLEs with *XMODEL*:
  - Block-level model using function primitives
  - Circuit-level model calibrated using *MODELFIT*
  - Circuit-level model generated using *MODELZEN*

Example CTLE

- CTLE with capacitive source degeneration



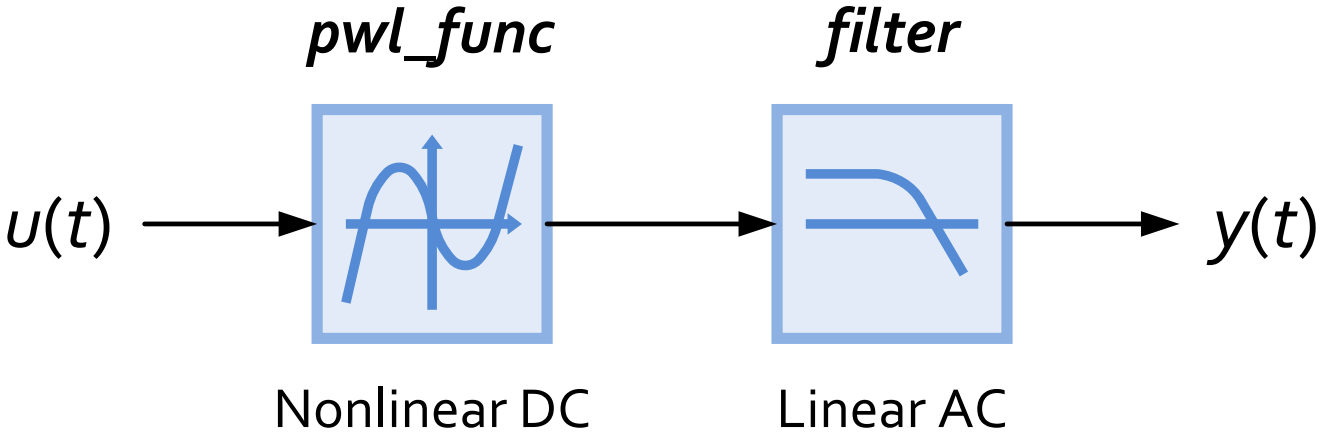
- AC transfer function H(s) has two poles & one zero:

$$H(s) = A_{DC}\frac{1 + s/z_1}{(1 + s/p_1)(1 + s/p_2)}$$

- Each output swing is limited to V_{DD}-$I_B R_L$ ~ V_{DD}
- Tunable by adjusting R_C and C_C
- ***ctle_model:ctle_core: schematic***

CTLE Block-Level Model

- Model a CTLE stage with
  - Large-signal DC characteristics
  - Small-signal AC characteristics



Hammerstein Model

Exercise #1: Model Fitting

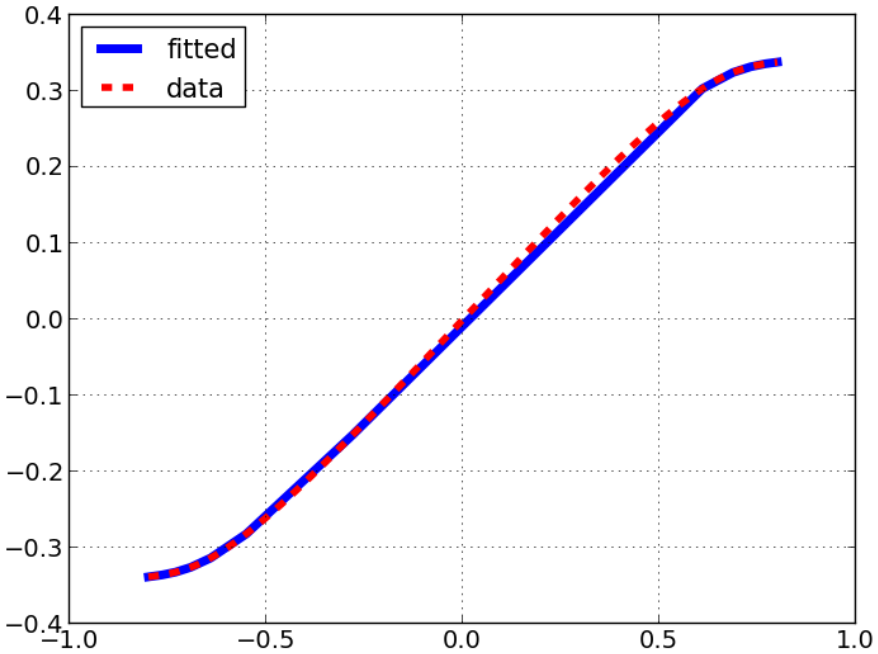
- Example is located in ***ctle_model/ctle_func***
  - Model template: ***ctle_func.sv.empy***
  - Model fitting script: ***fit_ctle.py***
- First, generate netlist for the following two cellviews (i.e. click *GLISTER->Generate Netlist* in Virtuoso):
  - ***ctle_model:ctle_basic:schematic***
  - ***ctle_model:ctle_core:schematic***
- Second, fit the model parameters by executing:

```
$ cd $TUTORIAL_HOME/ctle_model/ctle_func
$ make
```

Model Extraction with *MODELFIT*

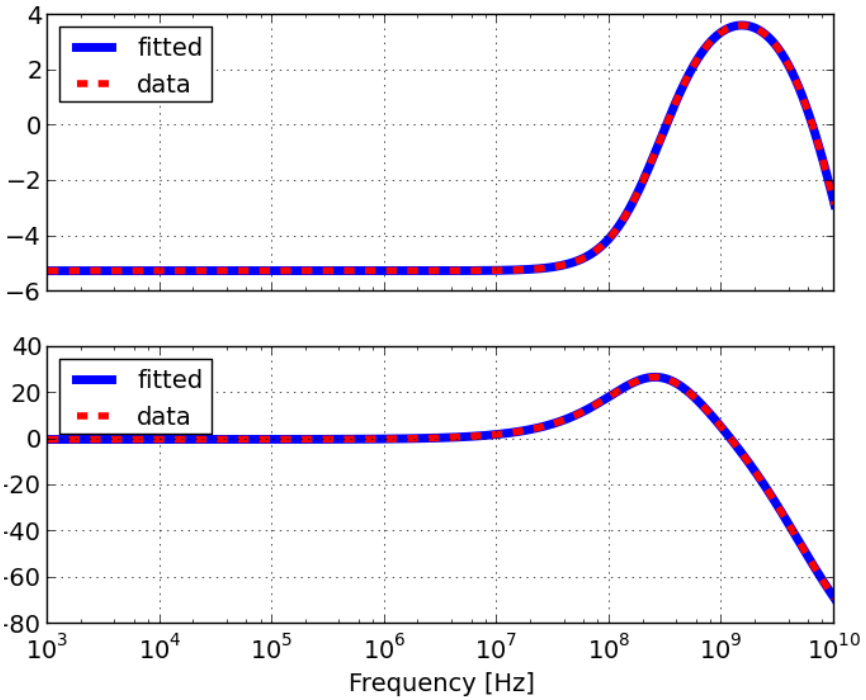
- *MODELFIT* extracts parameters for model templates chosen for DC and AC characteristics

DC Characteristics



modelfit("pwl_func")

AC Characteristics



modelfit("filter")

Exercise #1: Simulation

- Run DC simulation with slowly increasing input:

```
$ cd $TUTORIAL_HOME/ctle_model/ctle_func/tb/dc
$ make
```

- Run transient simulation with PRBS data input:

```
$ cd $TUTORIAL_HOME/ctle_model/ctle_func/tb/prbs
$ make
```

XMODEL-SPICE Cosimulation

- Command-line method to run *XMODEL*-SPICE cosimulation (e.g. VCS + XA):

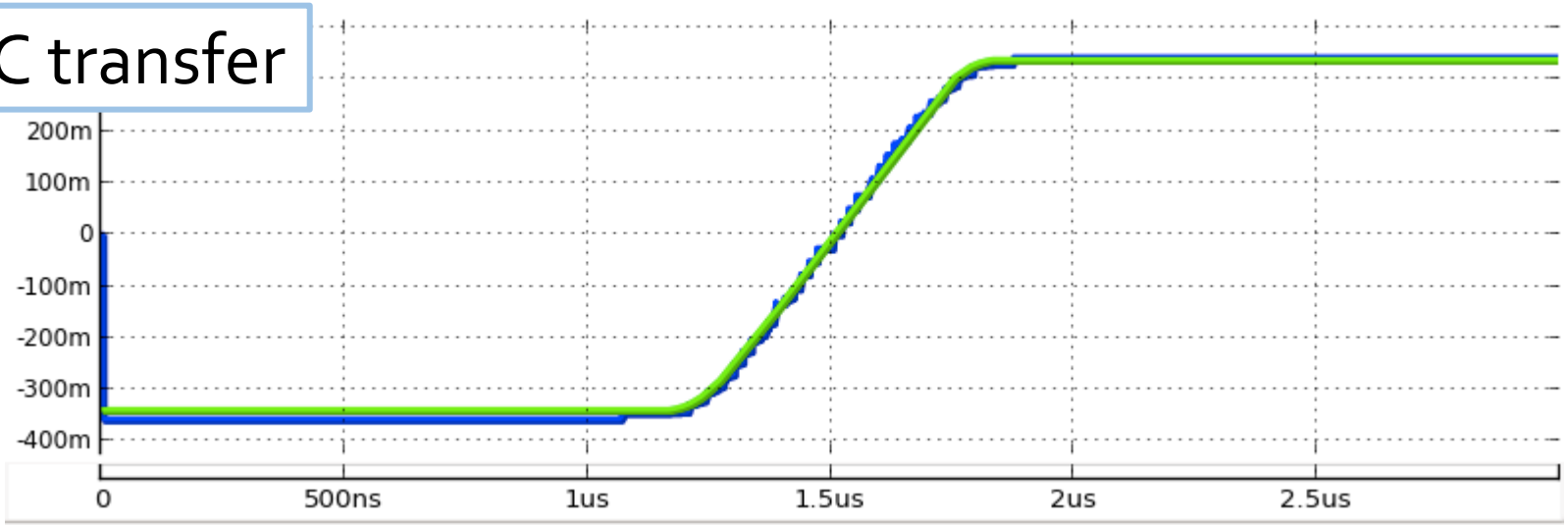
```
xmodel <source files> --top <top modulename>
    --simulator vcs --spice xa --cosimctrl vcsAD.init
```

- vcsAD.init is the cosimulation control file:

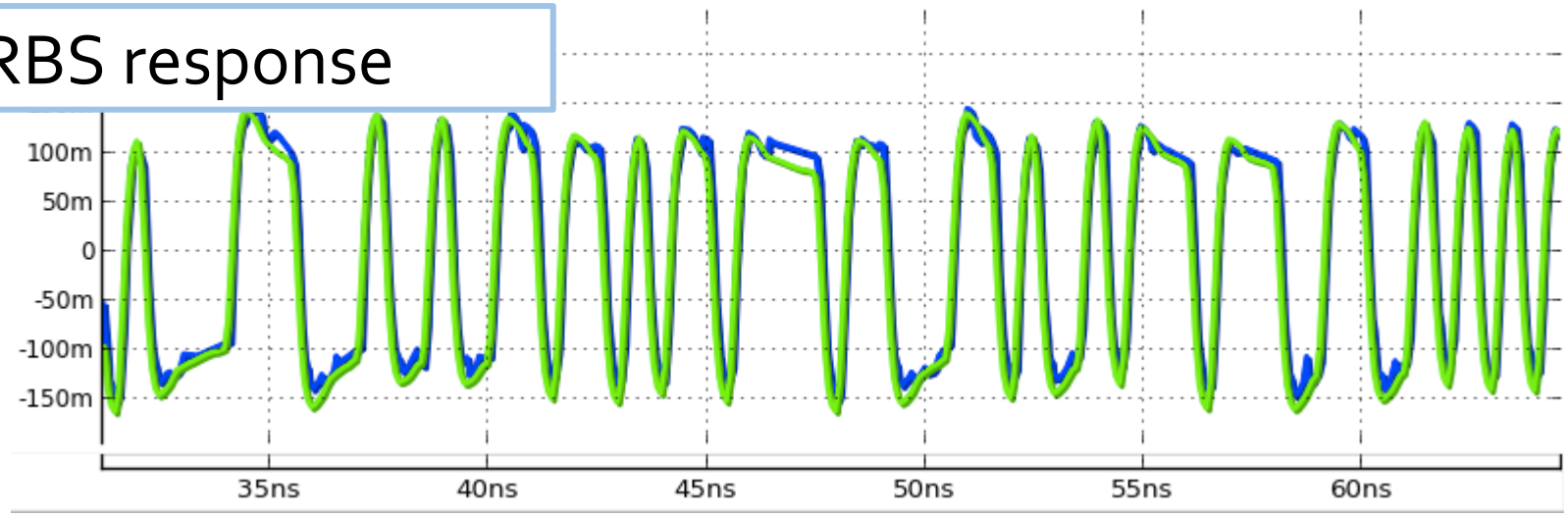
```
use_spice -cell ctle_basic;
choose xa tb_ctle.sp -c xa_config;
```

CTLE Simulation Results

DC transfer



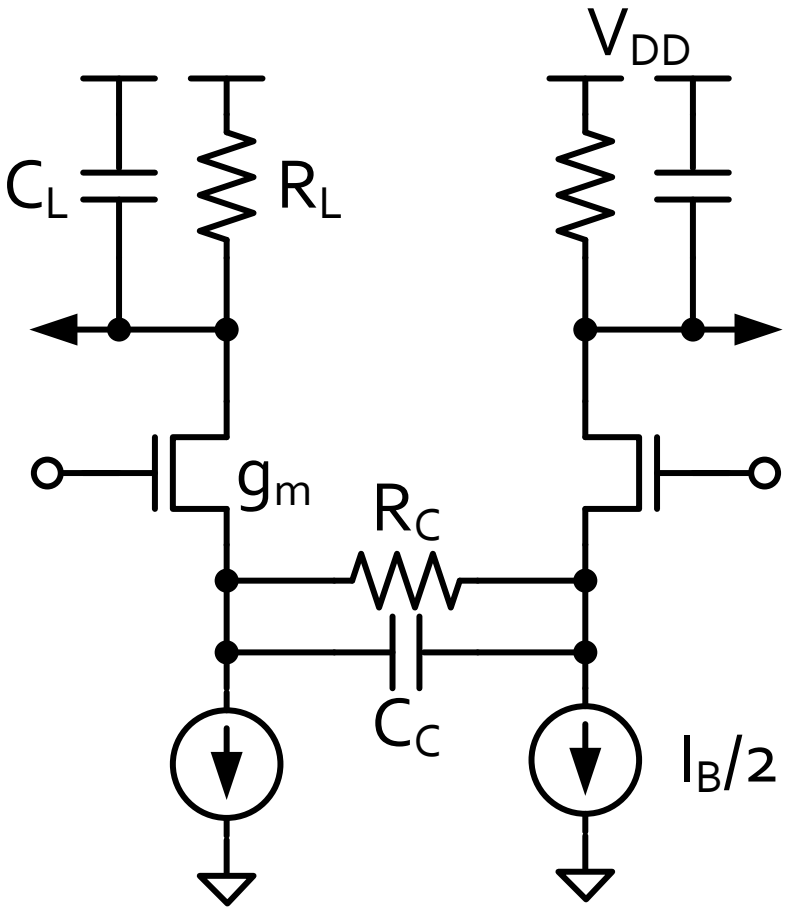
PRBS response



Circuit-Level Modeling of CTLE

- If modeling H(s) is the only requirement, using the ***filter*** primitive would be the easiest
- Considering the requirements on adaptability and nonlinearity, it'd be easier to use ***circuit-level models***
- Circuit-level modeling (CLM) allows you to describe circuits directly with circuit-level components
  - e.g. transistors, resistors, capacitors, …
  - *XMODEL* extracts H(s) at run-time
  - Can easily model tunable R and C and signal clipping

CTLE Model in *XMODEL*




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

// parameter definitions omitted for brevity

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
```

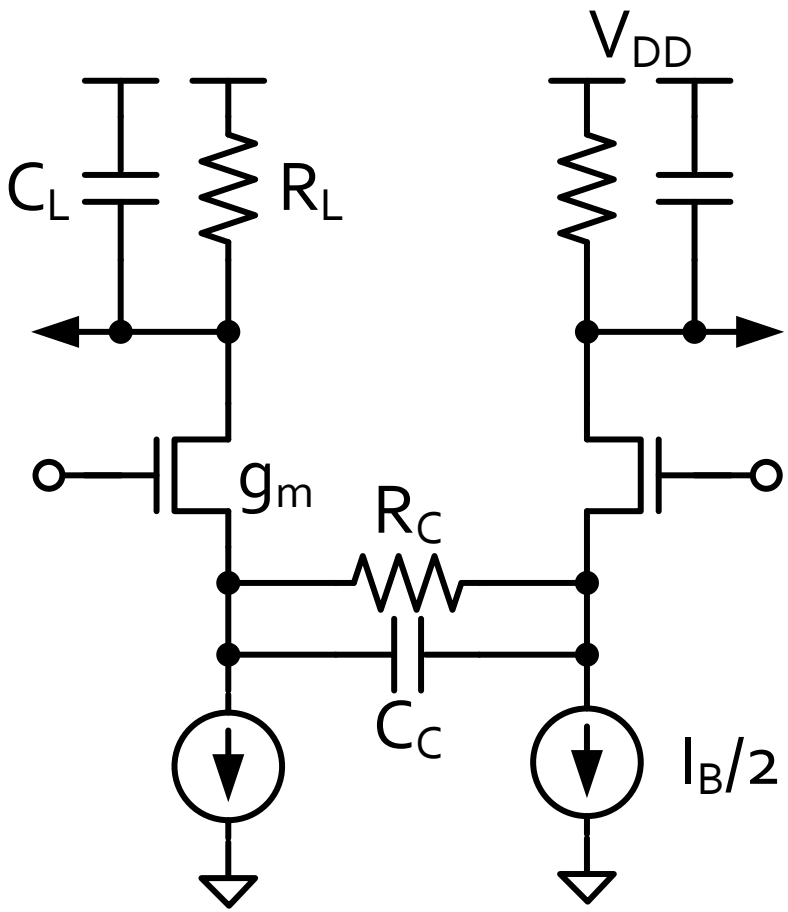
CTLE Model Parameters

- Behavioral parameters
  - **gain**: small-signal DC gain (A_{DC})
  - **pole1**: lower pole frequency (real-valued; p_1)
  - **pole2**: higher pole frequency (real-valued; p_2)
  - **zero**: zero frequency (real-valued; z)
  - **itotal**: current consumption (~1/output resistance)
  - **vout_max**: maximum output voltage (=Vdd)
  - **vout_min**: minimum output voltage
  - **vic_max**: maximum input common-mode voltage
- Circuit parameters
  - **g_m, V_{th}, R_L, C_L, R_C, C_C, I_B, V_{DD}**

AC small-signal param.'s

DC large-signal param.'s

CTLE Circuit Analysis



- AC small-signal param.'s:

$$A_{DC} = \frac{g_m R_L}{1 + g_m R_C/2}$$

$$p_1 = (1 + g_m R_C/2)/R_C C_C$$

$$p_2 = 1/R_L C_L$$

$$z = 1/R_C C_C$$

- DC large-signal param.'s:

$$V_{DD} - I_B R_L \le V_{out} \le V_{DD}$$

$$V_{in,cm} \le V_{out,cm} + V_{th}$$

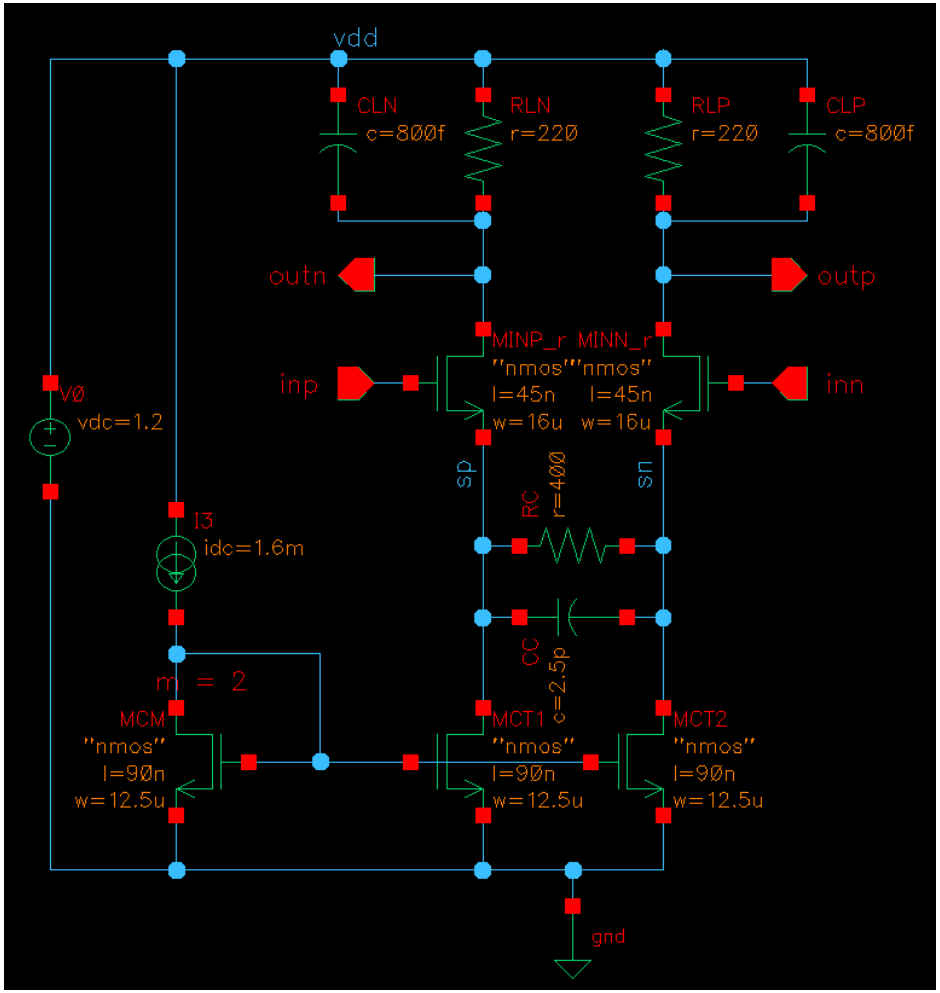
$$= V_{DD} - I_B R_L/2 + V_{th}$$

Parameter Definitions

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

// behavioral parameters
parameter real gain = 2;            // small-signal DC gain
parameter real pole1 = 0.5e9;       // lower pole frequency (in Hz)
parameter real pole2 = 10e9;        // higher pole frequency (in Hz)
parameter real zero = 0.1e9;        // zero frequency (in Hz)
parameter real itotal = 1e-3;       // total current consumption
parameter real vout_max = 1.2;      // maximum output voltage (=Vdd)
parameter real vout_min = 0.8;      // minimum output voltage
parameter real vic_max = 1.0;       // maximum input common-mode voltage

// circuit parameters
parameter real Vdd = vout_max;                      // supply voltage
parameter real Ib = itotal;                         // bias current
parameter real Vth = vic_max-(vout_max+vout_min)/2; // threshold voltage
parameter real Rload = (vout_max-vout_min)/Ib;      // load resistance
parameter real Cload = 1.0/(Rload*2*M_PI*pole2);    // load capacitance
parameter real Gm = gain/Rload * (pole1/zero);      // transconductance
parameter real Rc = (pole1/zero-1)*2/Gm;            // source-degeneration resistance
parameter real Cc = 1.0/(Rc*2*M_PI*zero);           // source-degeneration capacitance

...
```

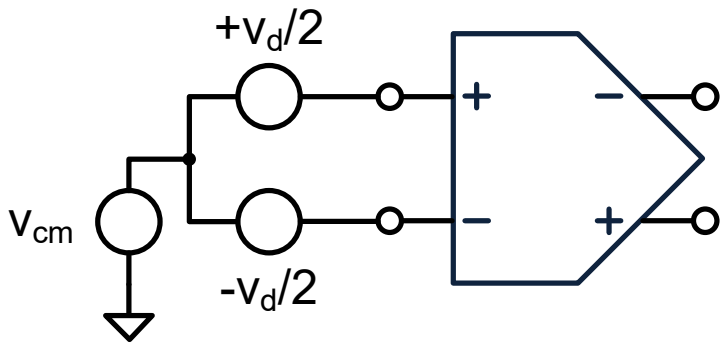
Parameter Extraction from Circuit (1)

- A circuit schematic based on PTM 45nm library
- You can write a model fitting script in Python using *XMULAN* and *MODELFIT*



Parameter Extraction from Circuit (2)

- Script ***fit_ctle.py*** performs three simulations to extract the parameters
  - DC simulation to extract *vic_max*, *itotal* parameter
  - DC simulation to extract *vout_min*
  - AC simulation to extract *gain*, *pole1*, *pole2*, *zero*



Exercise #2: Model Fitting

- Example is located in ***ctle_model/ctle_clm***
  - Model template: ***ctle_clm.sv.empy***
  - Model fitting script: ***fit_ctle.py***

- Fit the model parameters and generate a model by executing:

```
$ cd $TUTORIAL_HOME/ctle_model/ctle_clm
$ make
```

Exercise #2: Simulation

- Run DC simulation with slowly increasing input:

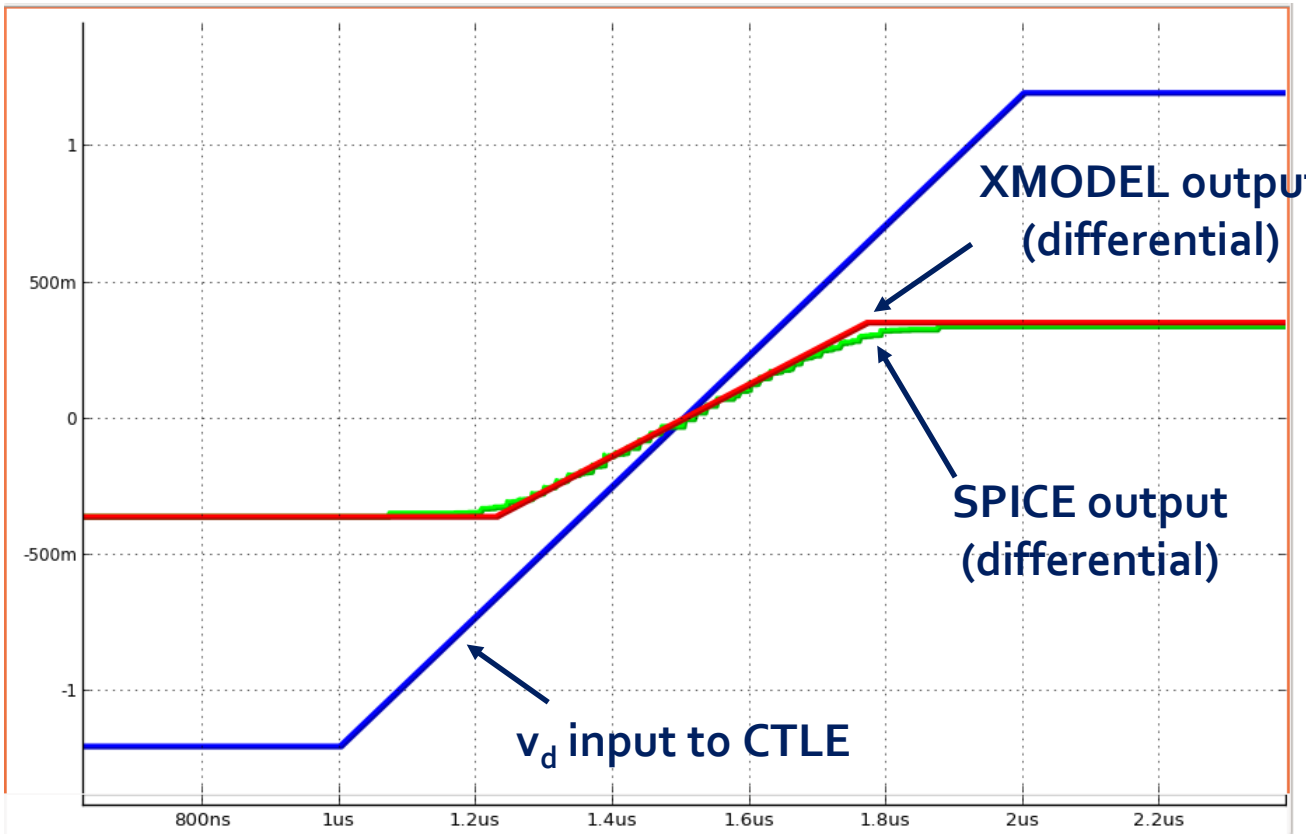
```
$ cd $TUTORIAL_HOME/ctle_model/ctle_clm/tb/dc
$ make
```

- Run transient simulation with PRBS data input:

```
$ cd $TUTORIAL_HOME/ctle_model/ctle_clm/tb/prbs
$ make
```

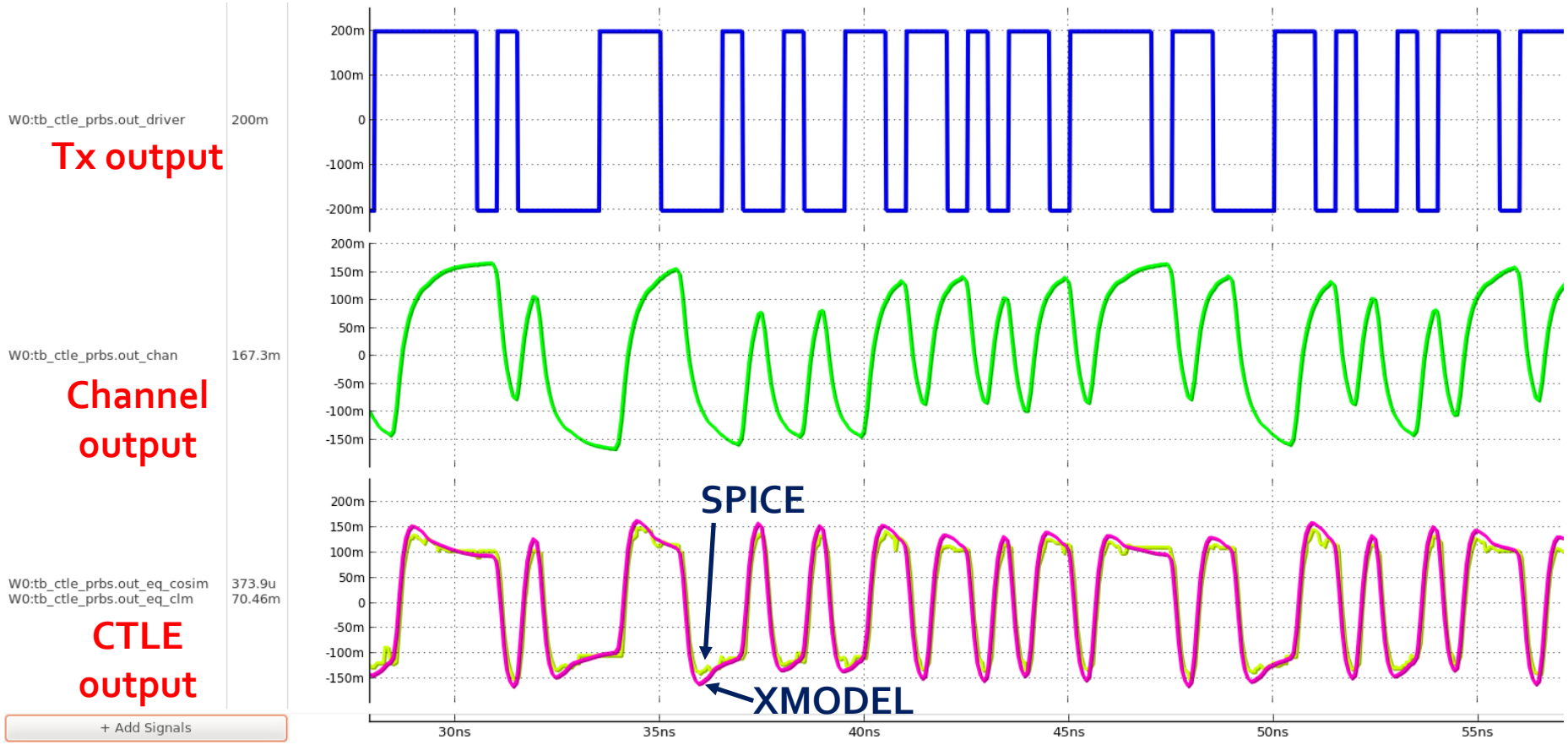
Simulation Results (1)

- DC gain compression effect



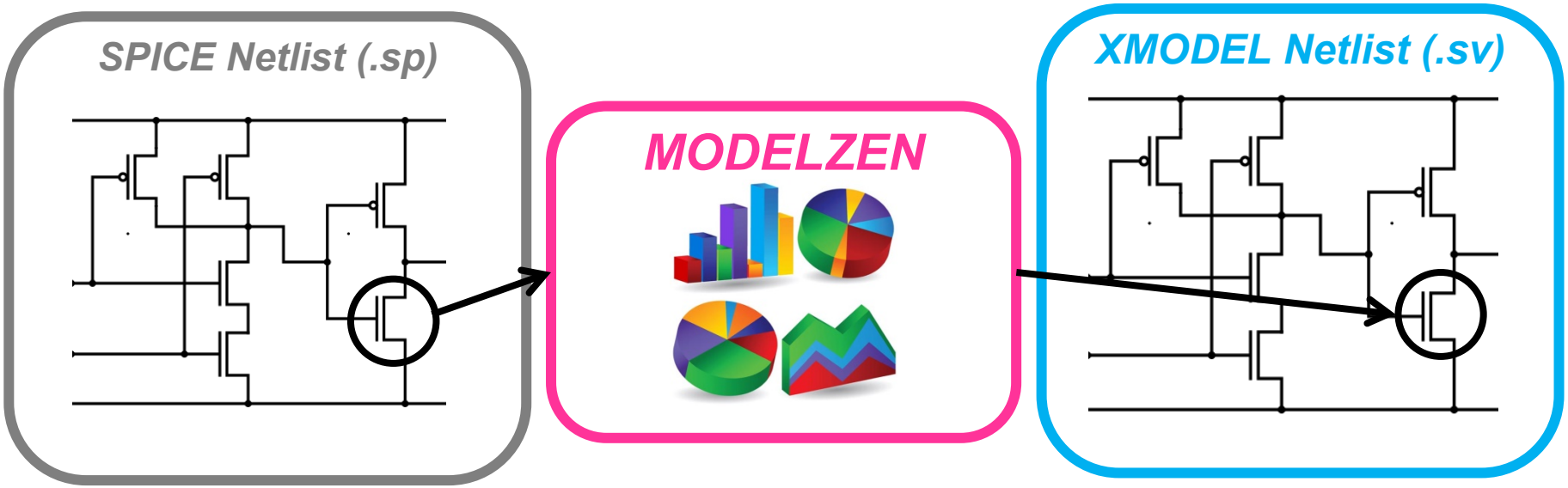
Simulation Results (2)

- Comparison of waveforms w/ PRBS input



Model Generation with *MODELZEN*

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



Exercise #3: Model Generation

- Example is located in ***ctle_model/ctle_mzen***
  - Technology configuration file: ***tech_config.py***

- Generate a model by executing:

```
$ cd $TUTORIAL_HOME/ctle_model/ctle_mzen
$ modelzen -c tech_config.py ctle_core.sp -o ctle_mzen.sv
```

Or, simply:

```
$ cd $TUTORIAL_HOME/ctle_model/ctle_mzen
$ make
```

Exercise #3: Simulation

- Run DC simulation with slowly increasing input:

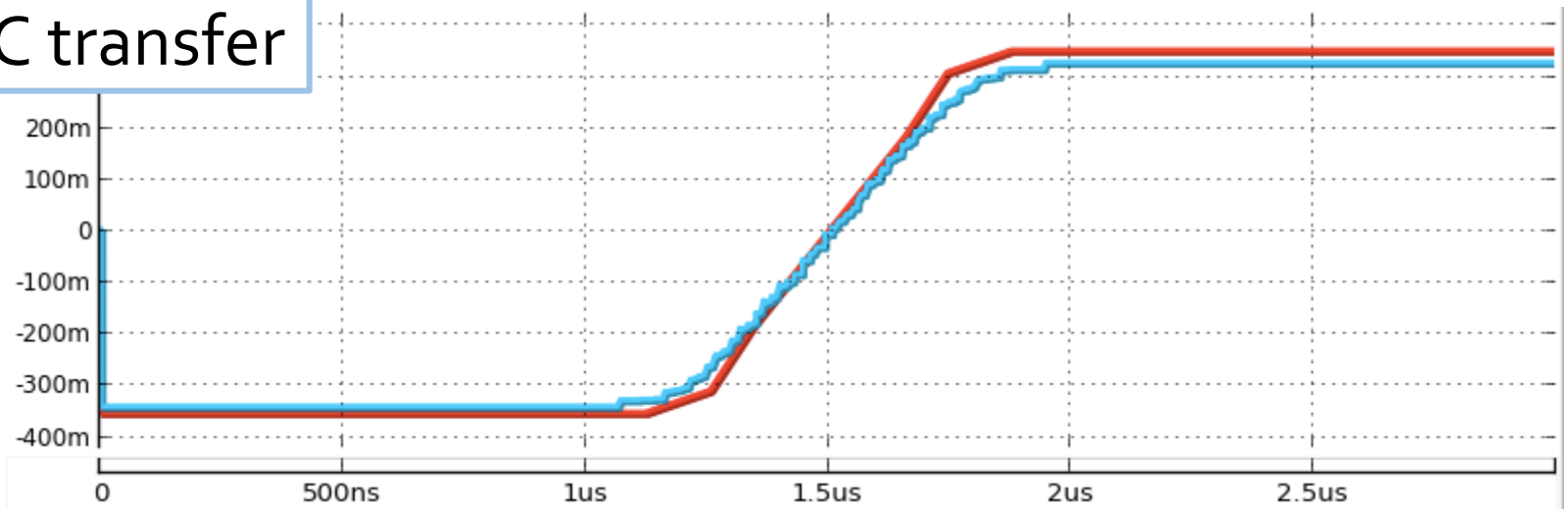
```
$ cd $TUTORIAL_HOME/ctle_model/ctle_mzen/tb/dc
$ make
```

- Run transient simulation with PRBS data input:

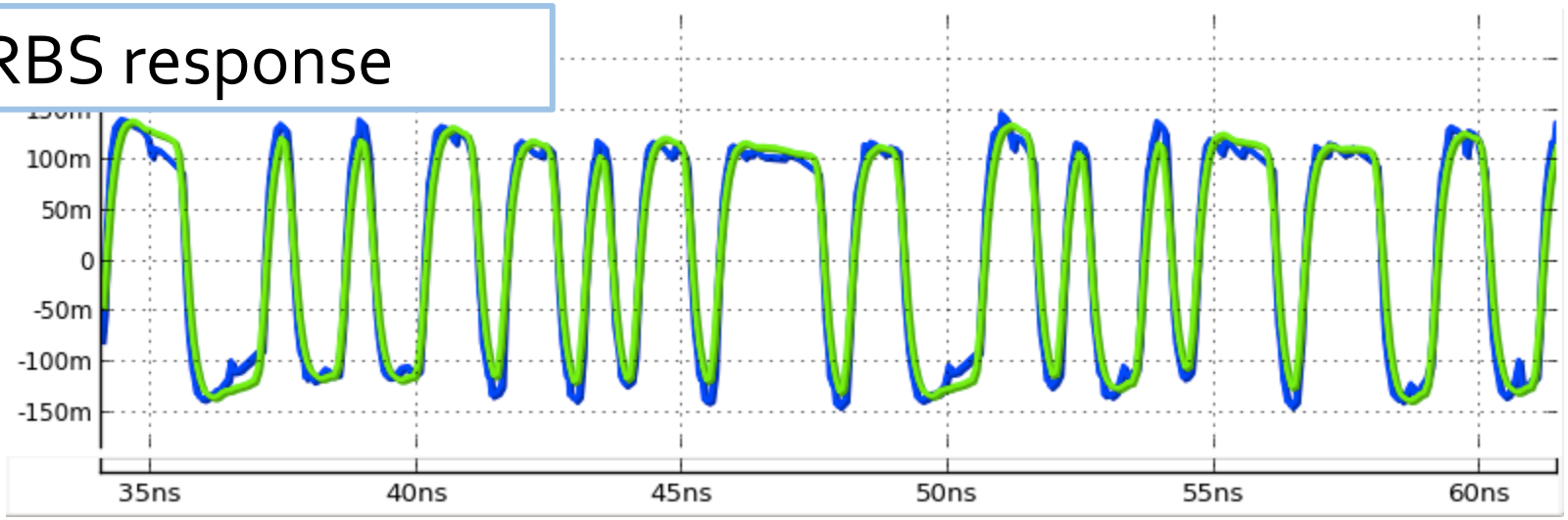
```
$ cd $TUTORIAL_HOME/ctle_model/ctle_mzen/tb/prbs
$ make
```

CTLE Simulation Results

DC transfer

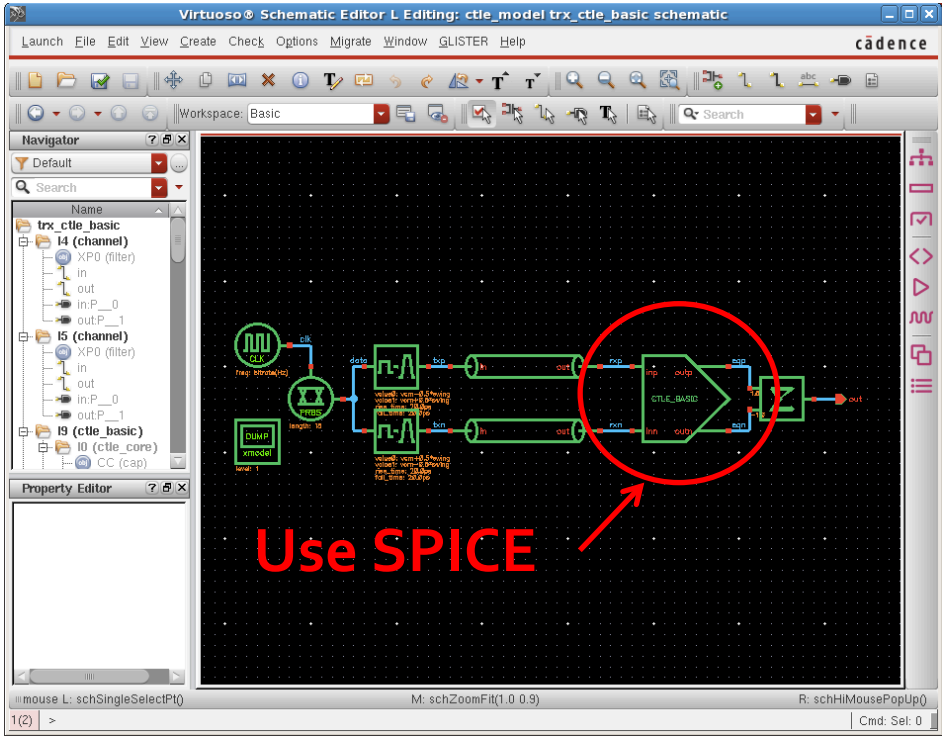


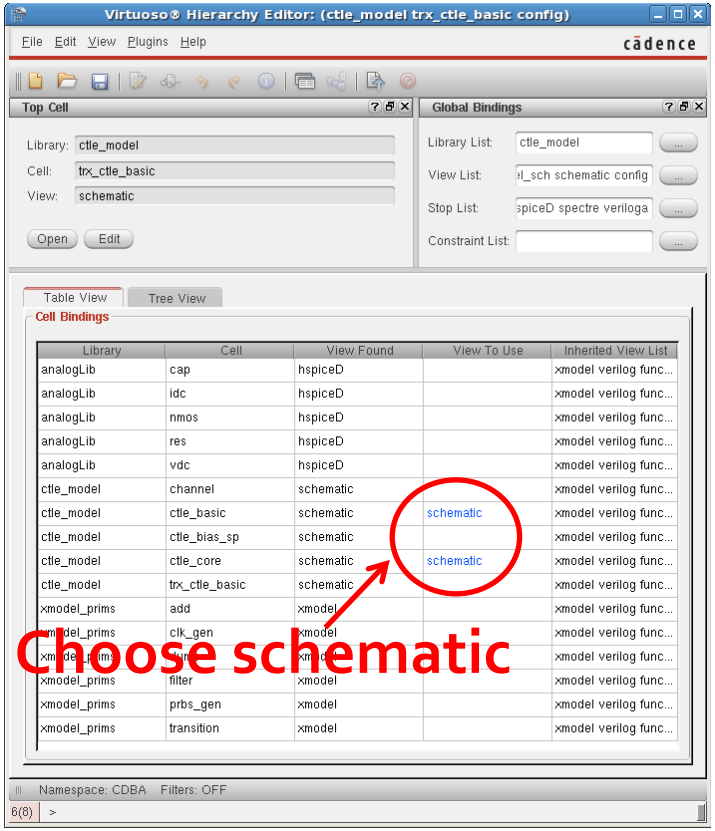
PRBS response



Exercise #5: Cosimulation Netlisting

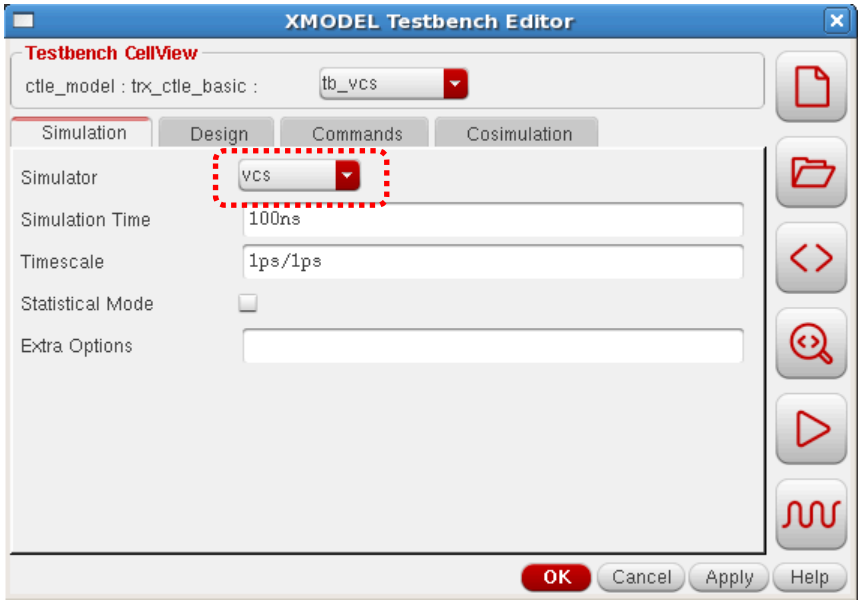
- Generate cosimulation netlist using *GLISTER*
- ***ctle_model:trx_ctle_basic: config***



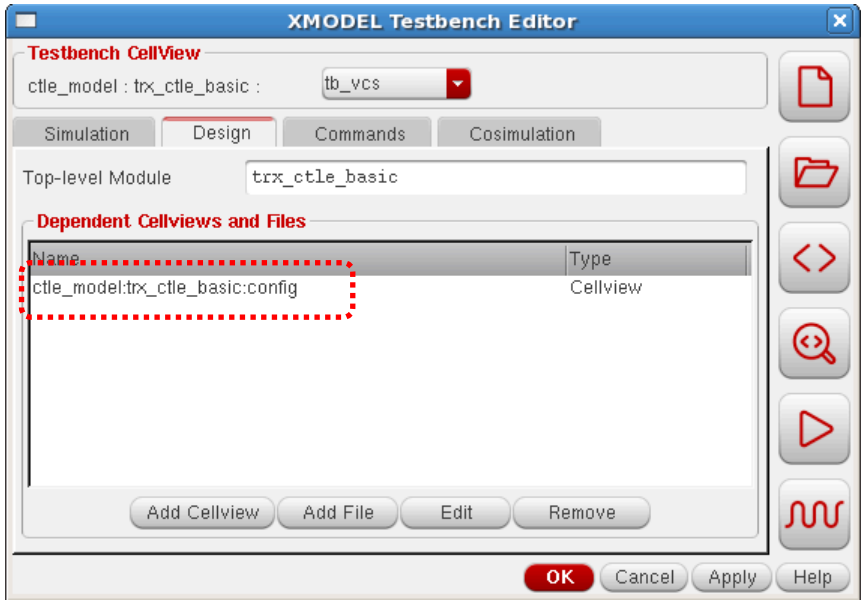


Exercise #5: Cosimulation Testbench

- Choose the simulator and config view for cosimulation
  - VCS-XA setup is in: ***ctle_model:trx_ctle_basic:tb_vcs***



Choose Simulator



Choose Config View Defining the Hierarchy

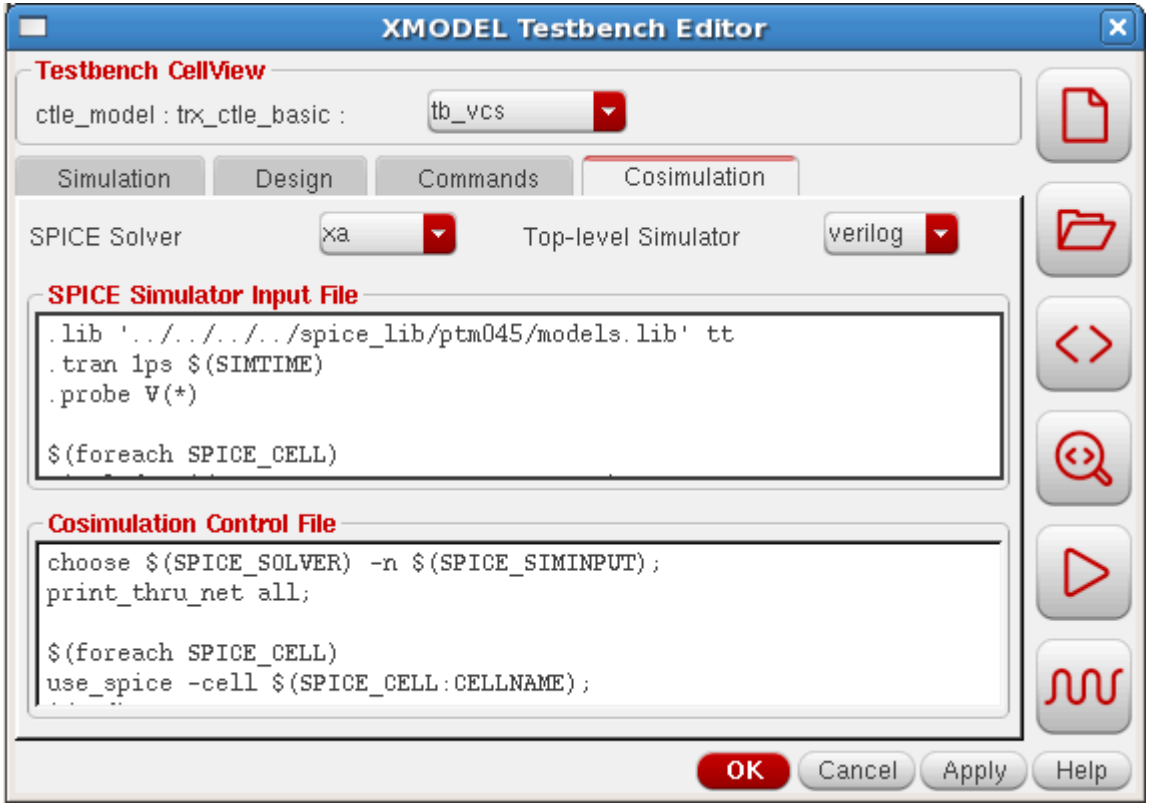
Exercise #5: Cosimulation Setup

- You can generate netlists and run simulation using the same interface of Testbench Editor

SPICE Solver

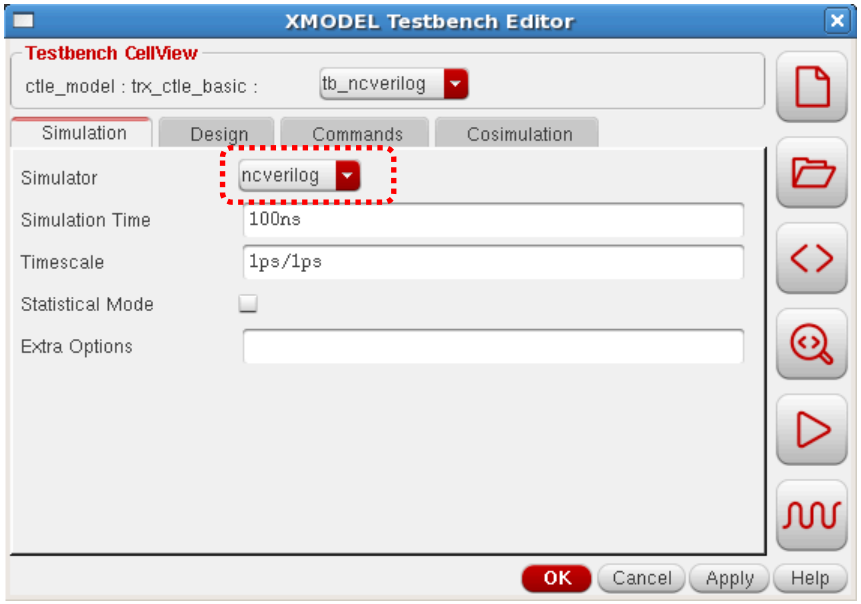
Top-Level SPICE Input File

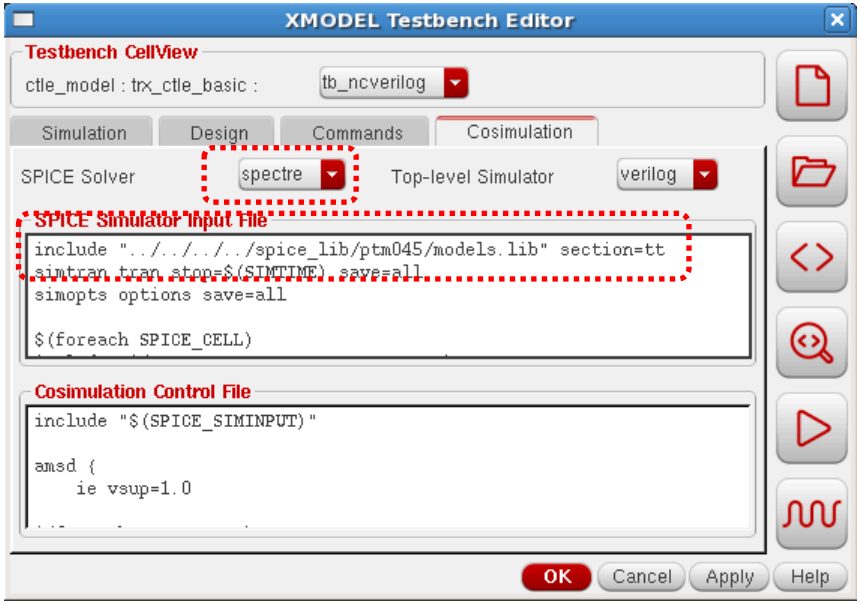
Co-simulation Control File



Exercise #5: Cosimulation Setup (2)

- NCVerilog-Spectre cosimulation setup is found in: ***ctle_model:trx_ctle_basic:tb_ncverilog***





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***, ***MODELZEN***, and ***MODELFIT*** are convenient tools that help you compose analog models and verify them against SPICE