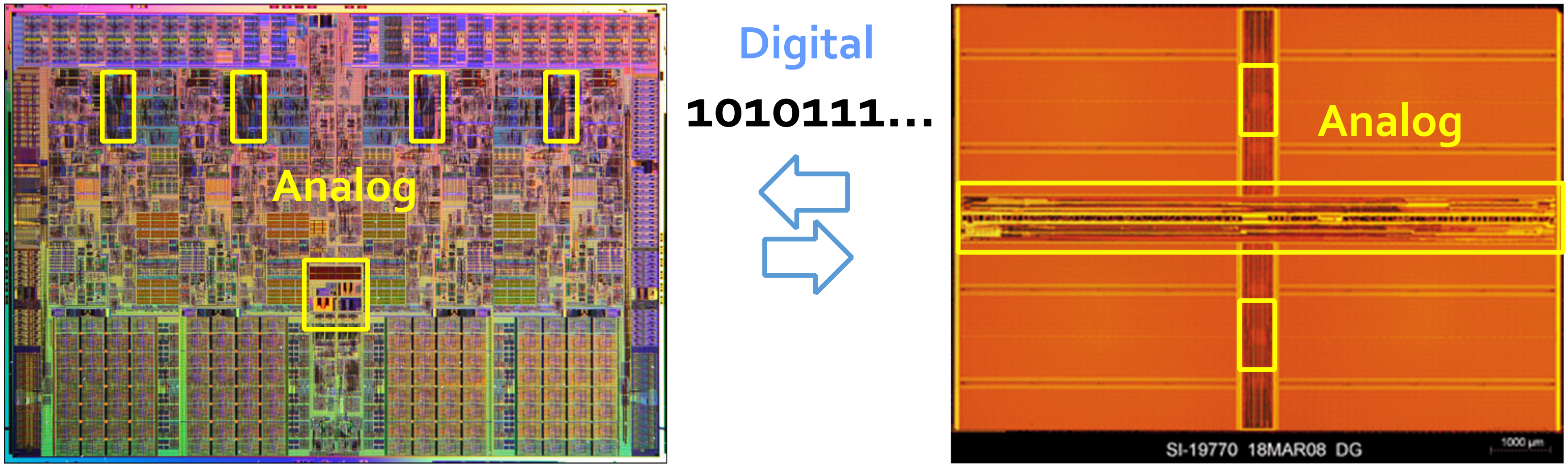
Automatic Generation of SystemVerilog Models from Analog/Mixed-Signal Circuits: A Pipelined ADC Example

Prof. Jaeha Kim
Seoul National University & Scientific Analog, Inc.
September 2021

Today SoC's are Mixed-Signal

- No SoC's are purely digital or purely analog
- Many SoC's are digital on outside and analog on inside
  - Chip-level testbenches are usually in SystemVerilog (UVM)

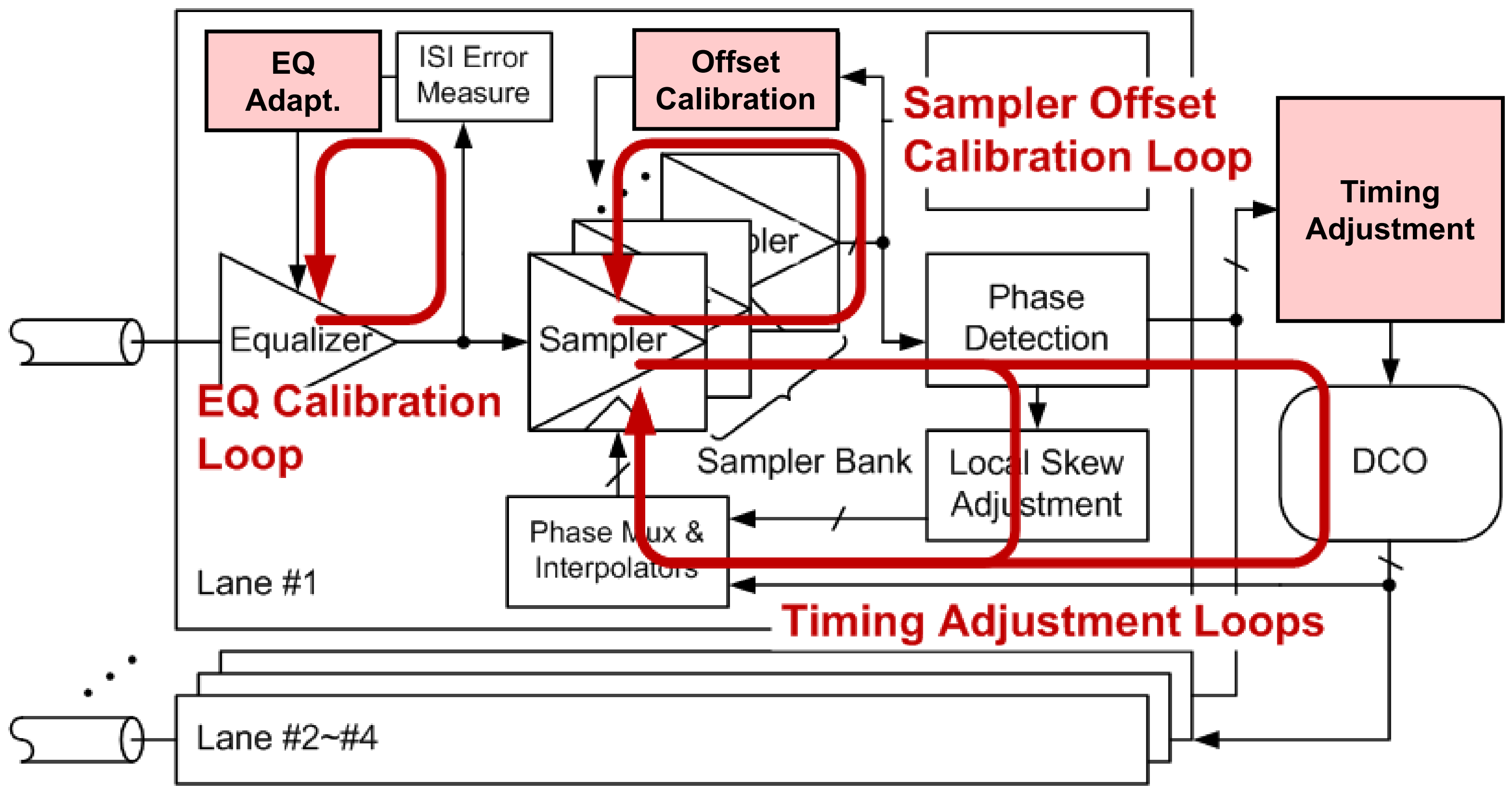


CPU

DRAM

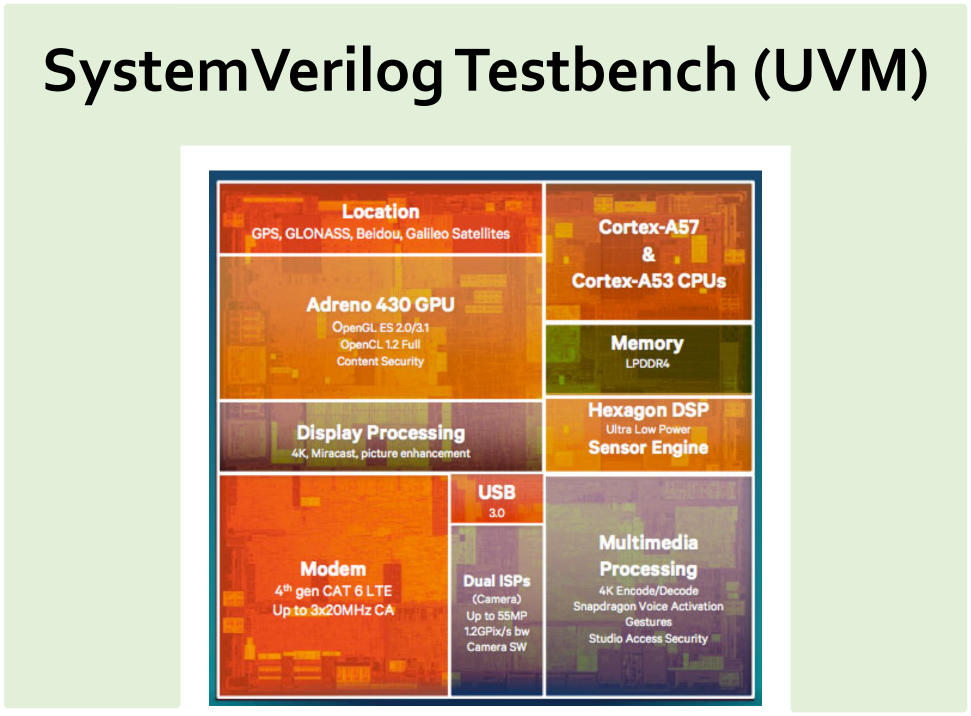
With No Clear A/D Boundaries

- Often, analog and digital parts form a feedback loop; making it difficult to verify one without the others



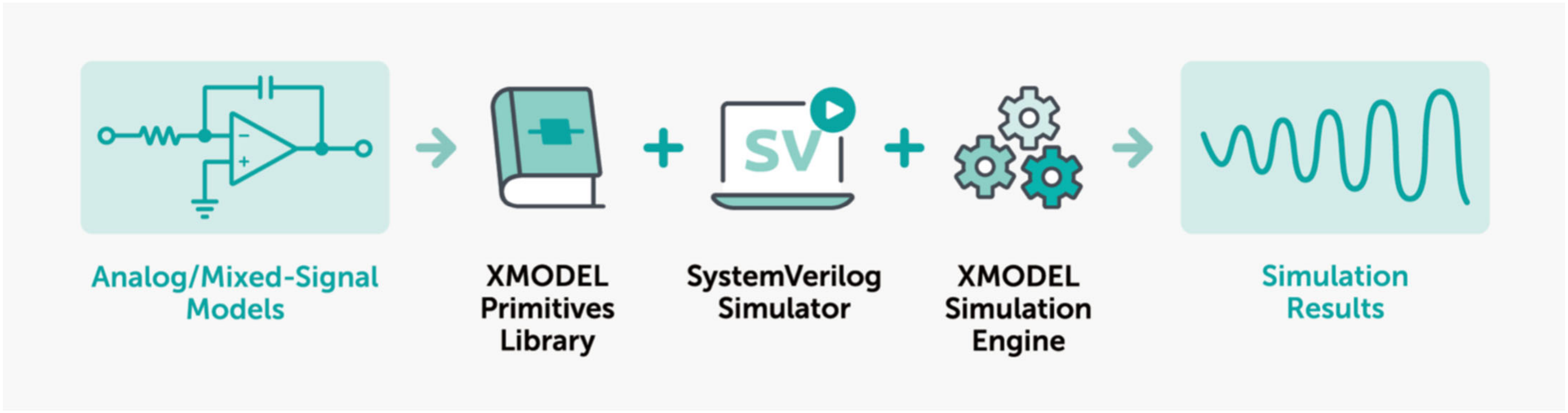
Goal: Verify SoC's in SystemVerilog

- To achieve this goal, we need capabilities to:
  - Simulate analog models in SystemVerilog
  - Auto-extract models from analog circuits



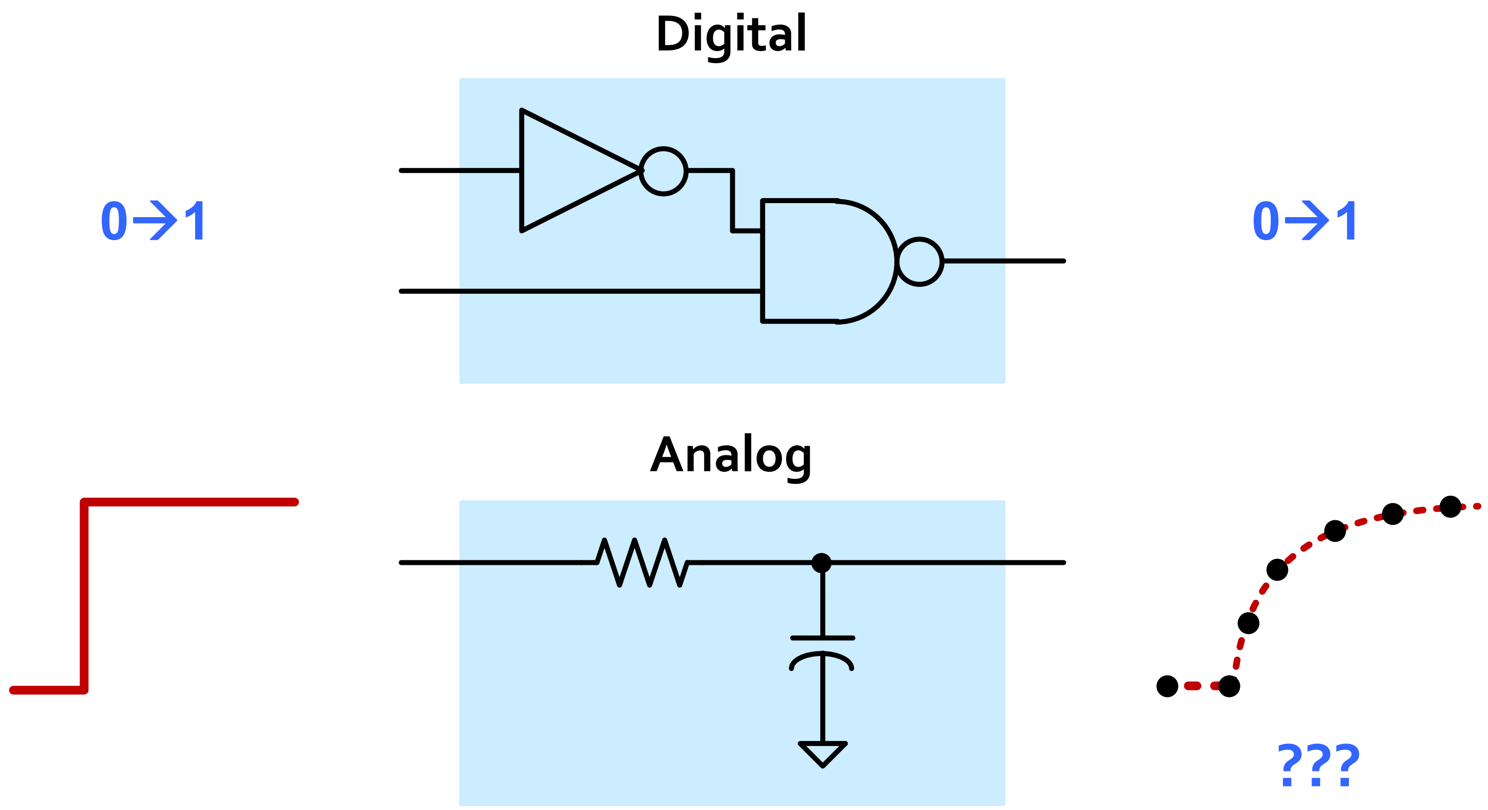
XMODEL: Enable Analog for SV/UVM

- A plug-in extension that enables ***fast and accurate analog/mixed-signal*** simulation in ***SystemVerilog***
  - ***Event-driven***: 10~100x faster than Real-Number Verilog
  - ***Analog***: supporting both functional and circuit-level models
  - ***SystemVerilog***: enabling analog verification in UVM



Event-Driven Simulation of Analog

- How do we extend the Verilog's event-driven algorithm to simulating analog circuits?

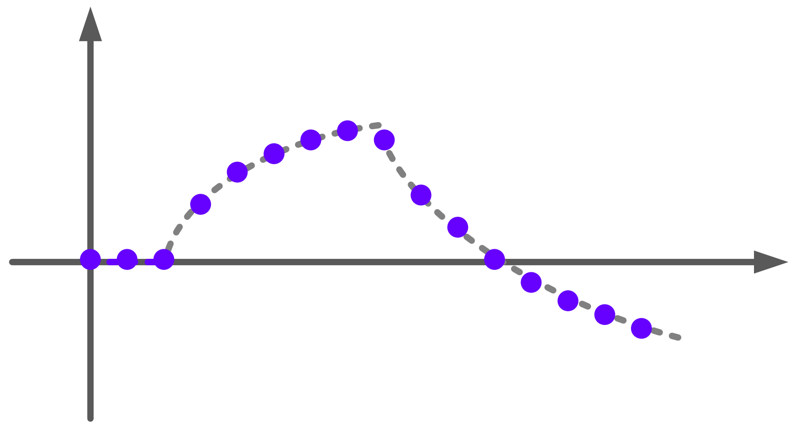


Expressing Analog Events

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

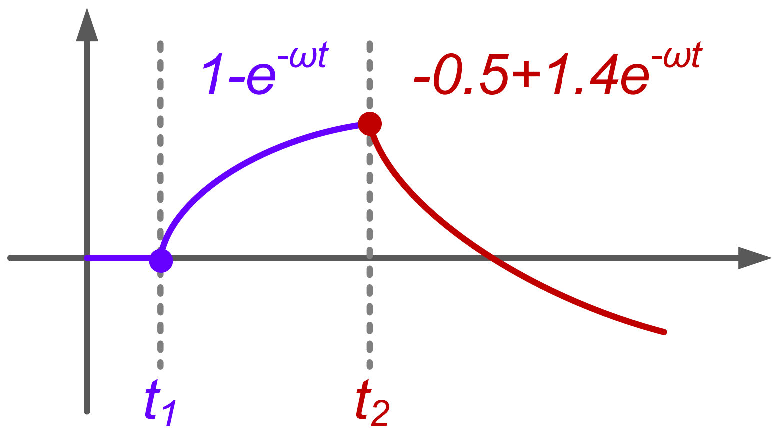
$$x(t) = \sum_i c_i t^{m_i} e^{-a_i t}$$

SPICE



Accuracy relies on fine time step

XMODEL



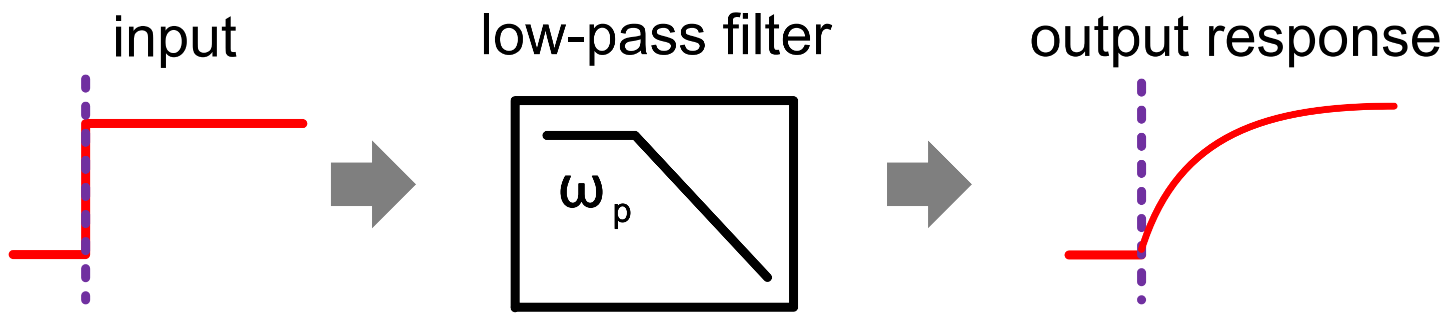
Events occur only when the coefficients are updated

Propagating Analog Events

- With the signals transformed into Laplace s-domain:

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

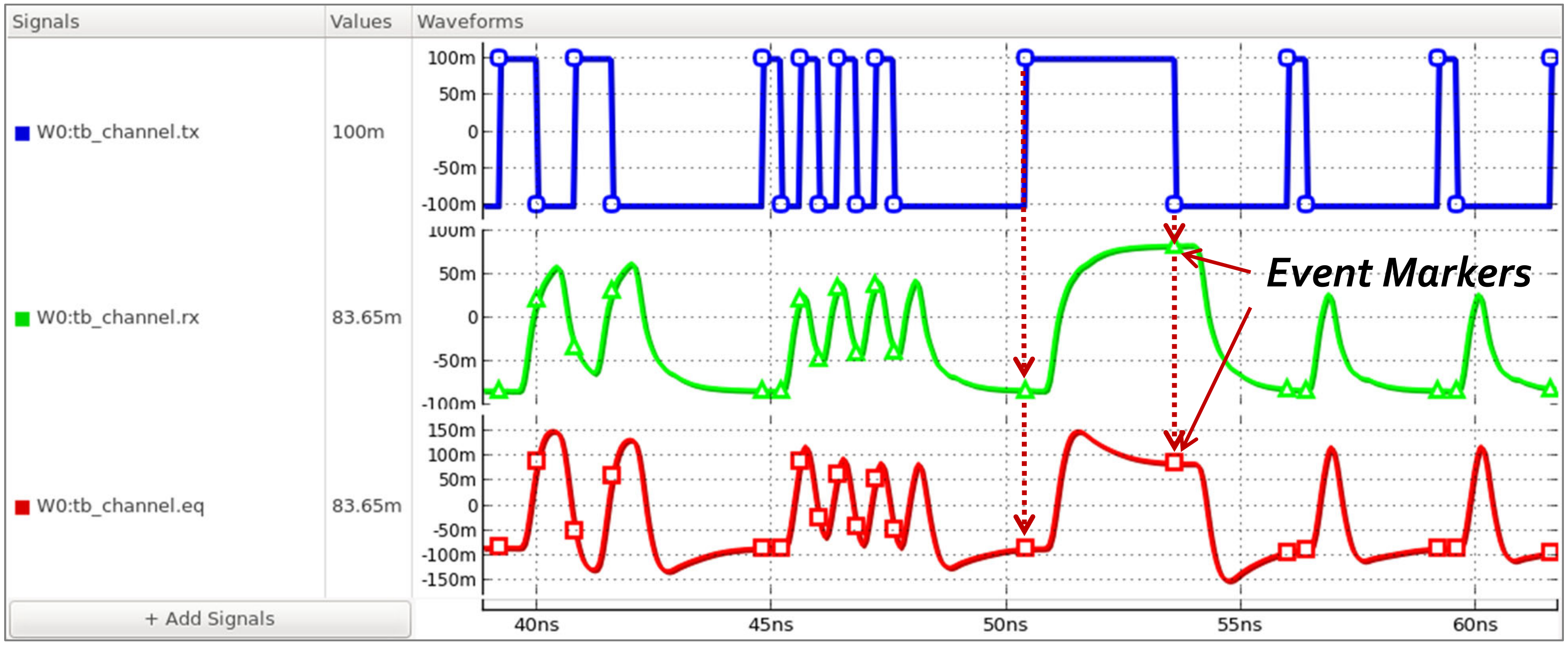
- The response of a system can be computed in an event-driven manner without time-step integration:



$$\frac{1}{s} \times \frac{\omega_p}{\omega_p + s} = \frac{1}{s} - \frac{1}{\omega_p + s}$$

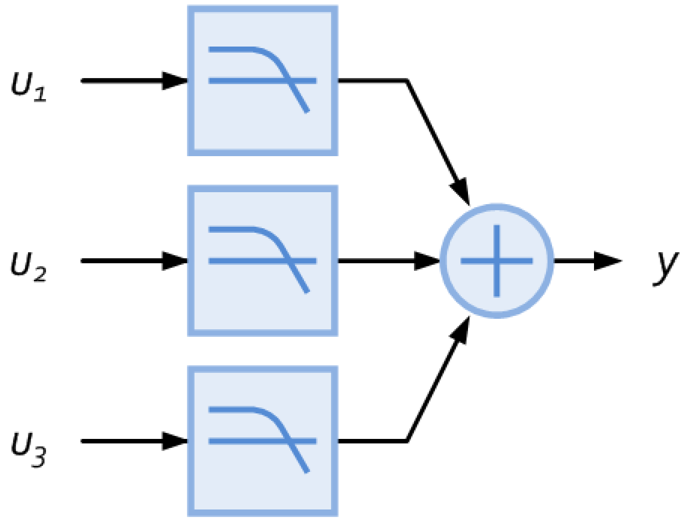
XMODEL's Event-Driven Simulation

- *XMODEL* triggers very few events during simulation and hence achieves very fast speed



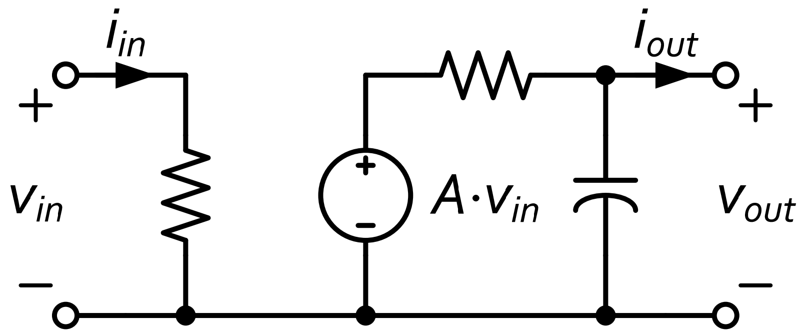
Block-Level vs. Circuit-Level Models

Block-Level Model
(Signal-flow Model)



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

Circuit-Level Model
(Conservative System Model)

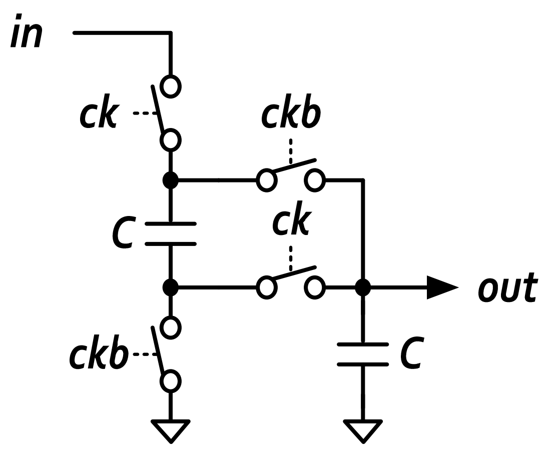


- A network of circuits whose state is described by voltages & currents
- e.g. loading effects

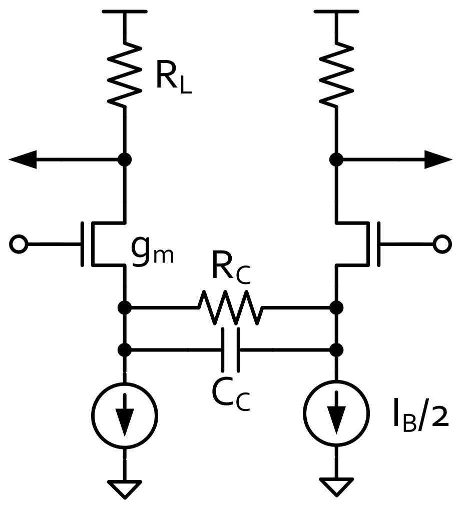
Need for Circuit-Level Models (CLMs)

- Circuit-level models are the most natural way to model switching, nonlinear, and loading effects in analog circuits

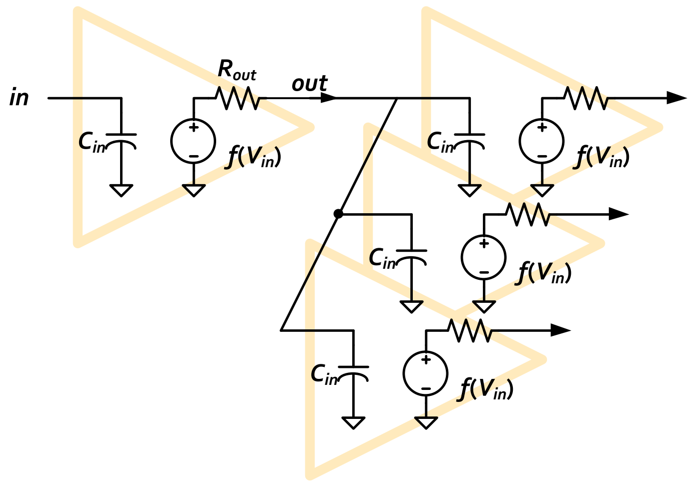
Switching Behaviors



Nonlinear Behaviors

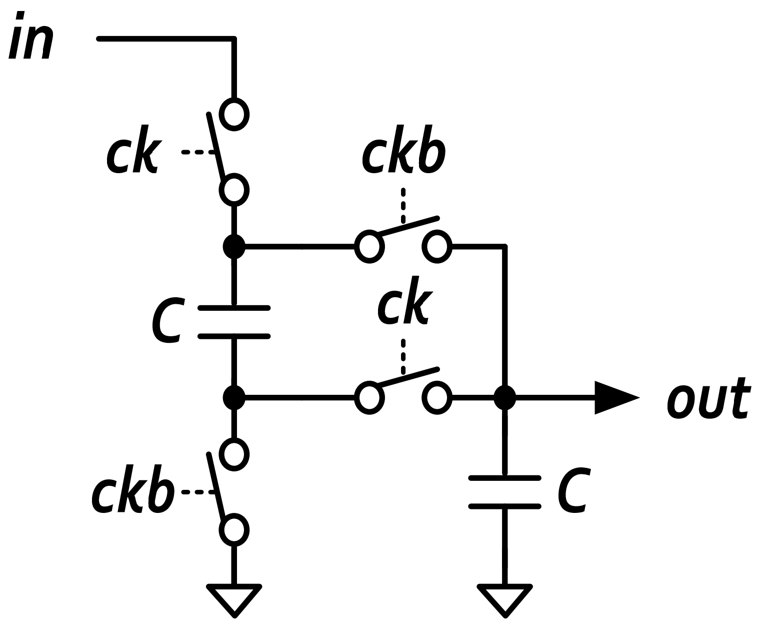


Loading Effects



CLM Support in *XMODEL*

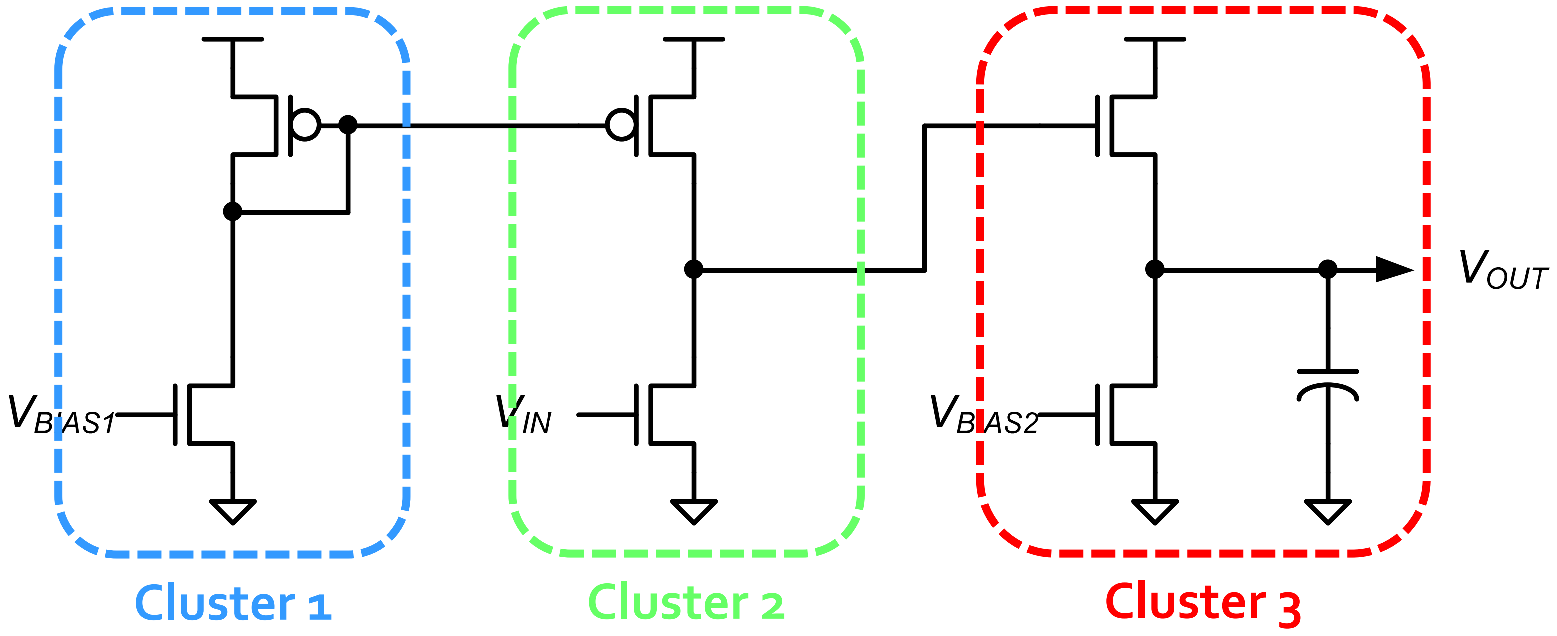
- In *XMODEL*, one can describe analog circuits directly by listing the circuit's elements and devices
- *XMODEL* can simulate these models in SystemVerilog in event-driven fashion without invoking SPICE



```
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
```

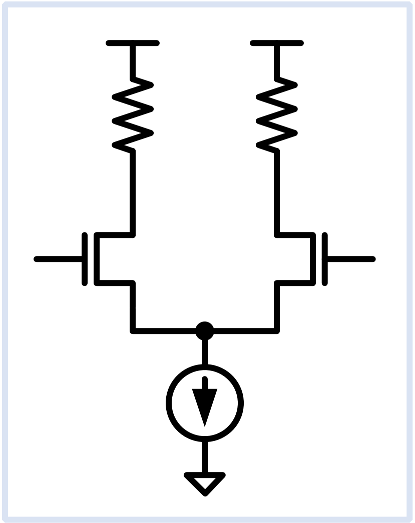
CLM Support (1): Circuit Clustering

- *XMODEL* partitions the circuit into clusters that can be solved separately
  - Signals across the cluster boundaries are unidirectional



CLM Support (2): TF Extraction

- For each cluster, *XMODEL* extracts the transfer function (TF) between its inputs and outputs
- Then, the *XMODEL*'s event-driven algorithm can compute output events from the input events



Circuit Cluster

MNA

$$E\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

Differential Algebraic Equation

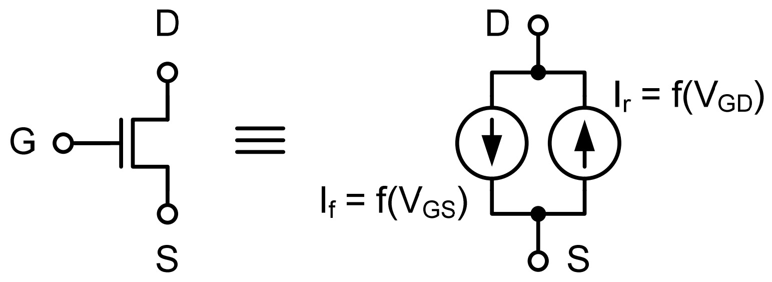
PFE

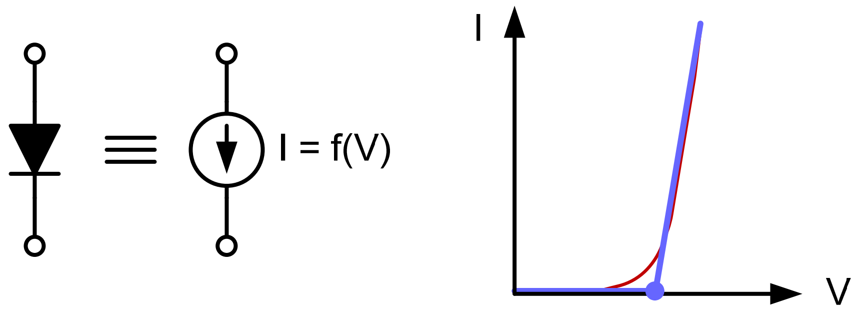
$$H(s) = \sum_i \frac{b_i}{(s+a_i)^{m_i}}$$

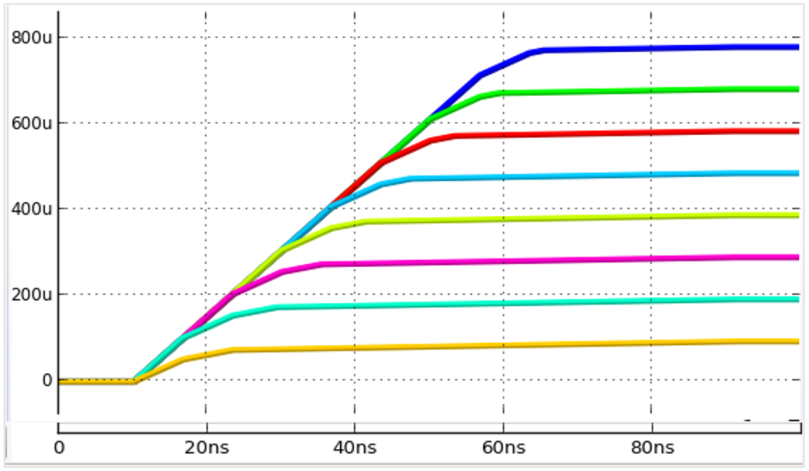
Transfer Function

PWL Models for Nonlinear Elements

- *XMODEL* models nonlinear elements (e.g. diodes and transistors) using piecewise-linear (PWL) models
  - In each operation region, the circuit yields a linear TF
  - Switch into a new TF when the region changes

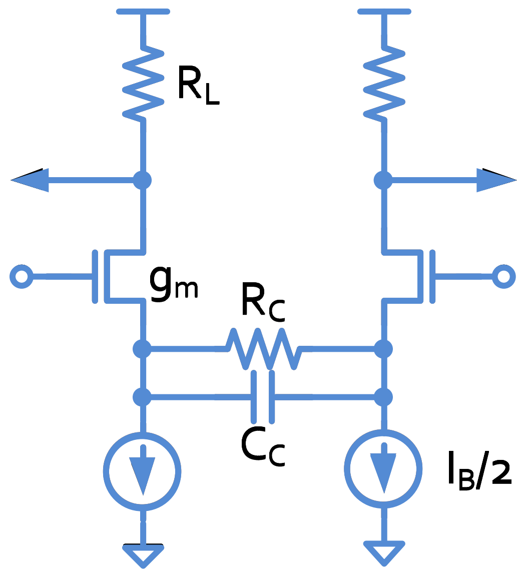






Structural Model Generation

- The first way to auto-generate models from circuits
  - Model each device in the circuit individually
  - Build the circuit model by connecting the device models as in the original circuit



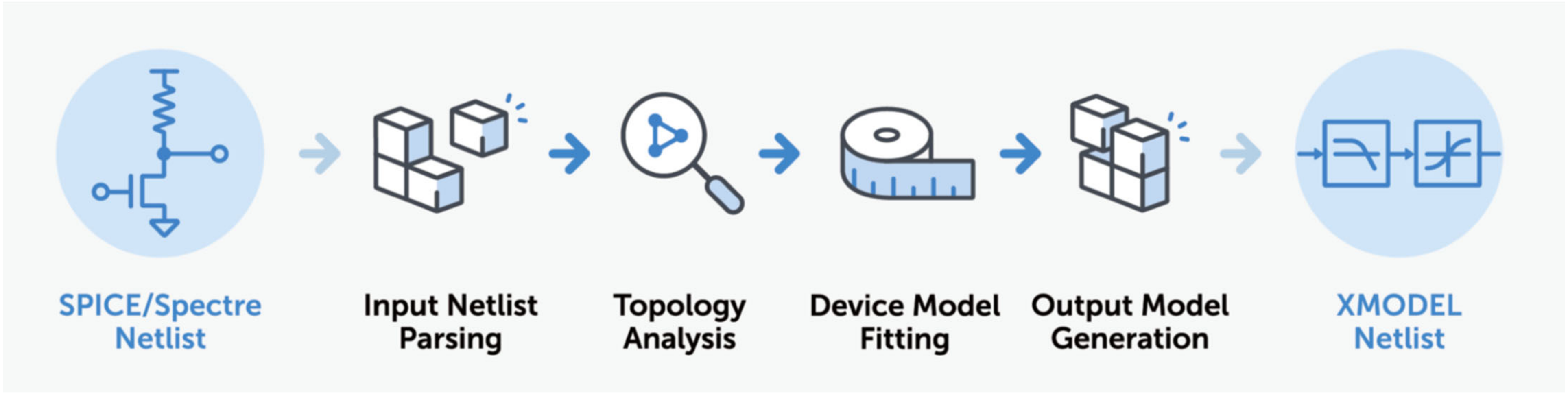
SPICE Elements

XMODEL Primitives

```
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
```

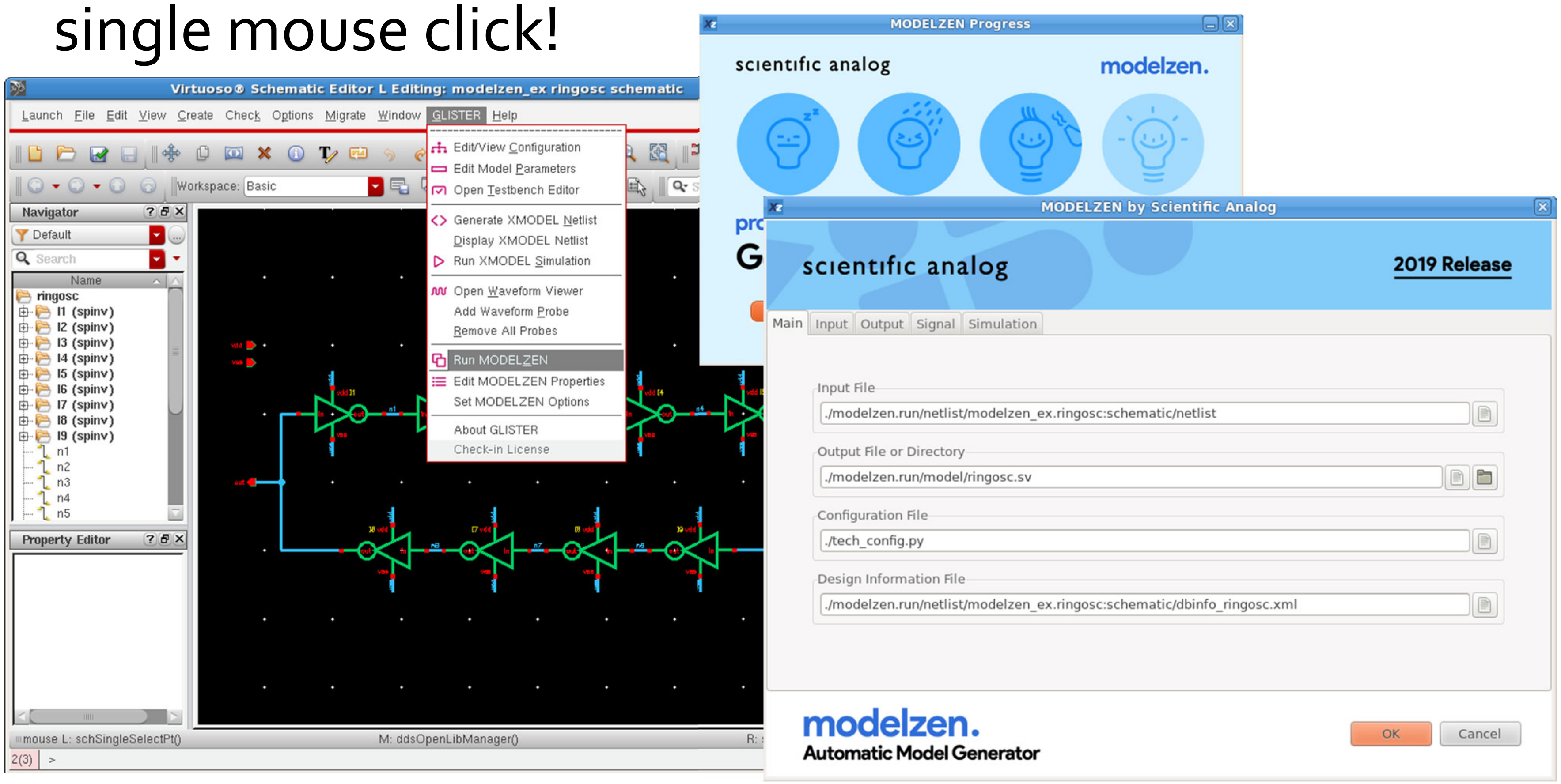
MODELZEN: Extract Bottom-Up Models

- ***MODELZEN*** can automatically generate bottom-up analog models from your circuit schematics or netlists
  - Extracts structural, circuit-level models by default
  - Model parameters are calibrated via SPICE simulations
  - Extracted models also simulate in an event-driven way



MODELZEN in Cadence Virtuoso

- With *GLISTER* and *MODELZEN*, you can auto-create analog models from circuit schematics with just a single mouse click!



Input File
./modelzen.run/netlist/modelzen_ex.ringosc:schematic/netlist

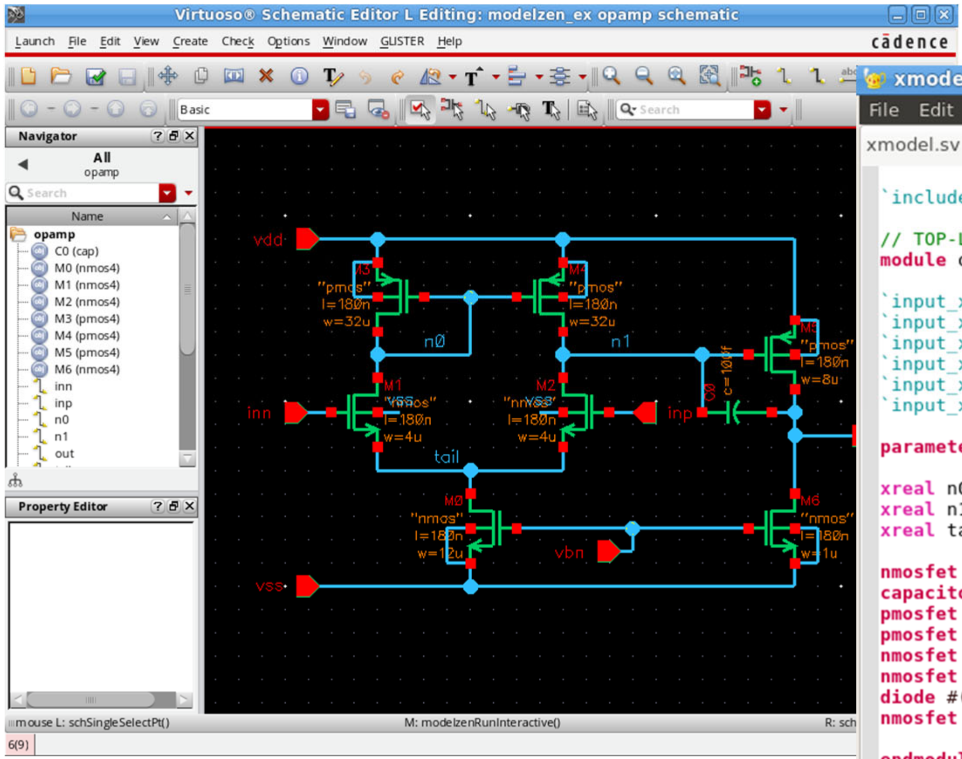
Output File or Directory
./modelzen.run/model/ringosc.sv

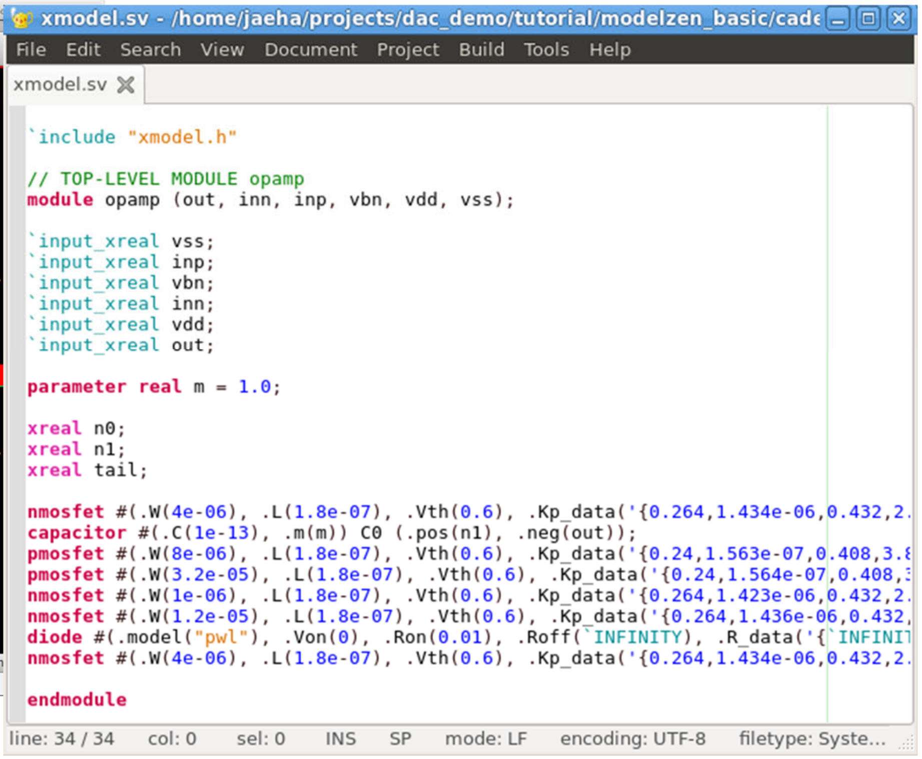
Configuration File
./tech_config.py

Design Information File
./modelzen.run/netlist/modelzen_ex.ringosc:schematic/dbinfo_ringosc.xml

Two-Stage Op Amp Example

- *MODELZEN* generates correct-by-construction, structural models using circuit-level primitives



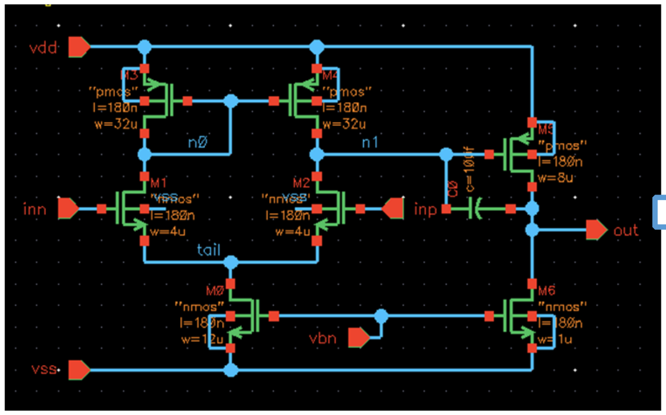

xmodel.sv - /home/jaeha/projects/dac_demo/tutorial/modelzen_basic/cade
File Edit Search View Document Project Build Tools Help
xmodel.sv

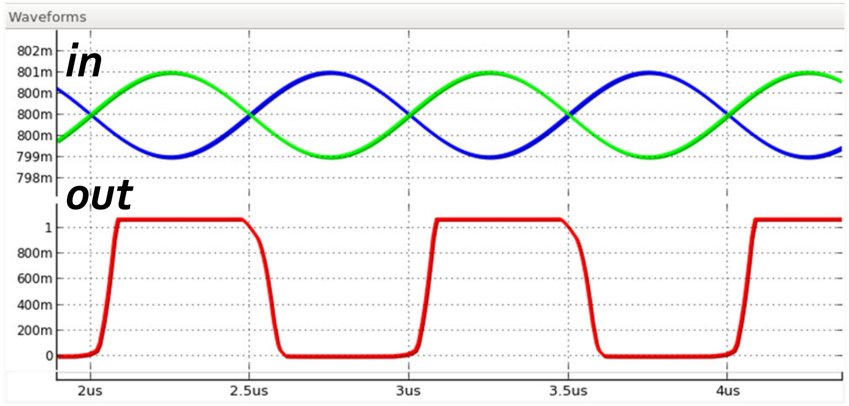
```
`include "xmodel.h"

// TOP-LEVEL MODULE opamp
module opamp (out, inn, inp, vbn, vdd, vss);

`input_xreal vss;
`input_xreal inp;
`input_xreal vbn;
`input_xreal inn;
`input_xreal vdd;
`input_xreal out;

parameter real m = 1.0;

xreal n0;
xreal n1;
xreal tail;

nmosfet #(.W(4e-06), .L(1.8e-07), .Vth(0.6), .Kp_data('{0.264,1.434e-06,0.432,2.
capacitor #(.C(1e-13), .m(m)) C0 (.pos(n1), .neg(out));
pmosfet #(.W(8e-06), .L(1.8e-07), .Vth(0.6), .Kp_data('{0.24,1.563e-07,0.408,3.
pmosfet #(.W(3.2e-05), .L(1.8e-07), .Vth(0.6), .Kp_data('{0.24,1.564e-07,0.408,
nmosfet #(.W(1e-06), .L(1.8e-07), .Vth(0.6), .Kp_data('{0.264,1.423e-06,0.432,2.
nmosfet #(.W(1.2e-05), .L(1.8e-07), .Vth(0.6), .Kp_data('{0.264,1.436e-06,0.432,
diode #(.model("pwl"), .Von(0), .Ron(0.01), .Roff(`INFINITY), .R_data('{`INFINIT
nmosfet #(.W(4e-06), .L(1.8e-07), .Vth(0.6), .Kp_data('{0.264,1.434e-06,0.432,2.

endmodule
```

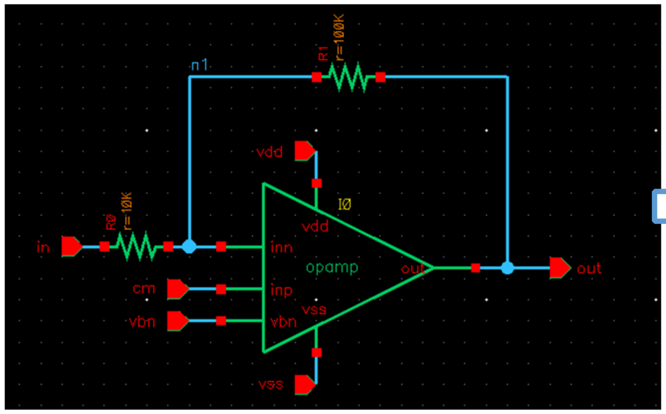
```
line: 34 / 34    col: 0    sel: 0    INS    SP    mode: LF    encoding: UTF-8    filetype: Syste...
```

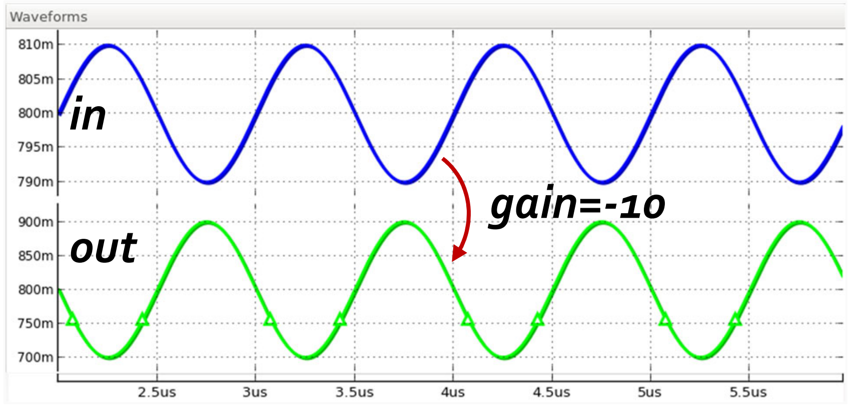
Op Amp Simulation Results

- Models both nonlinear behavior when it's in open loop and linear behavior when in closed-loop feedback



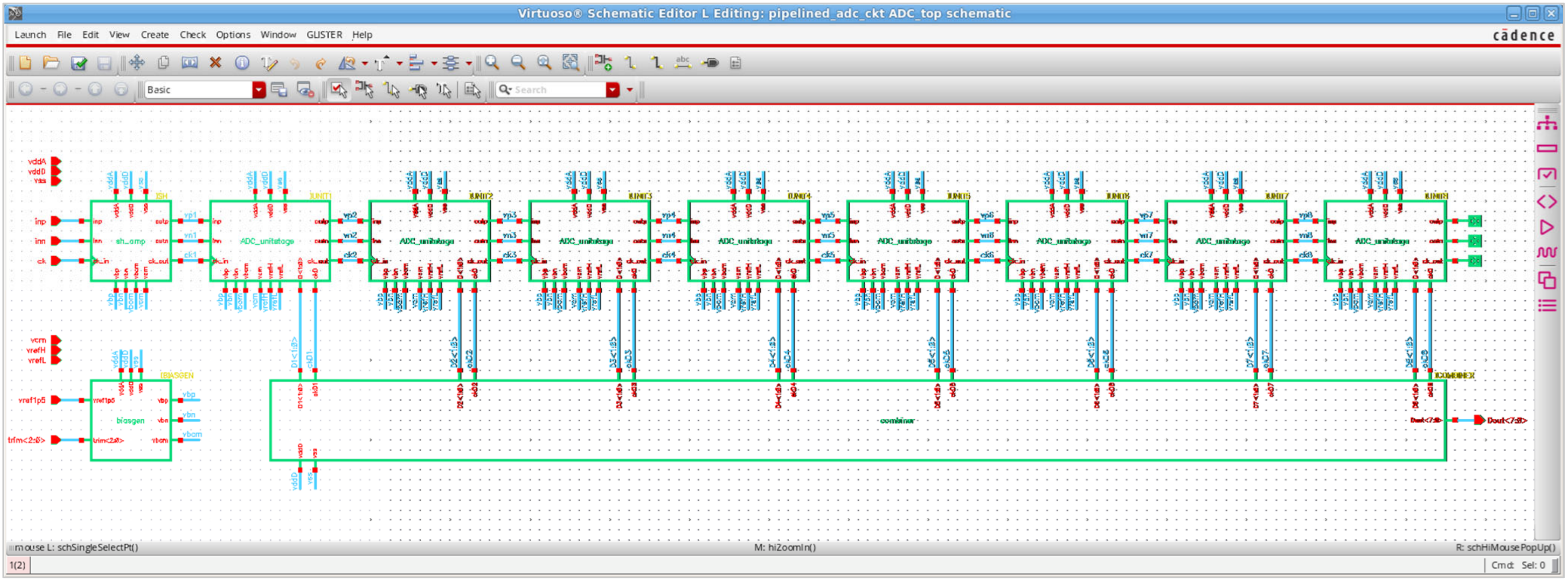






Pipelined ADC Example

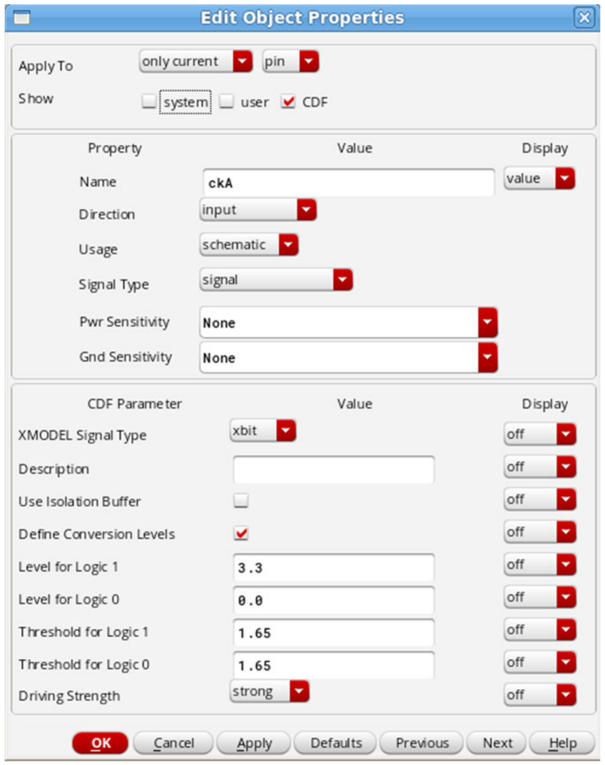
- An 8-bit pipelined ADC designed in PTM 45nm with dual power supplies (***vddD***=1.2V and ***vddA***=3.3V)
  - Operates at 100MHz with full-scale range of 0.5V_{pp}


Virtuoso® Schematic Editor L Editing: pipelined_adc_ckt ADC_top schematic
Launch File Edit View Create Check Options Window GLISTER Help
cādence
Basic
Search
vddA
vddD
vss
inp
inn
clk
sh_amp
ADC_unitstage
vcm
vrefH
vrefL
vref1p5
trim<2:0>
biasgen
combiner
Dout<7:0>
mouse L: schSingleSelectPt()
M: hiZoomIn()
R: schHiMousePopUp()
1(2)
Cmd: Sel: 0

CLM Extraction Approach (1)

- Assign *MODELZEN* pin properties so that digital I/O pins have ***xbit*** or ***bit*** types with proper conversion levels (1.2 or 3.3V)

pipelined_adc_ckt.MDAC:schematic

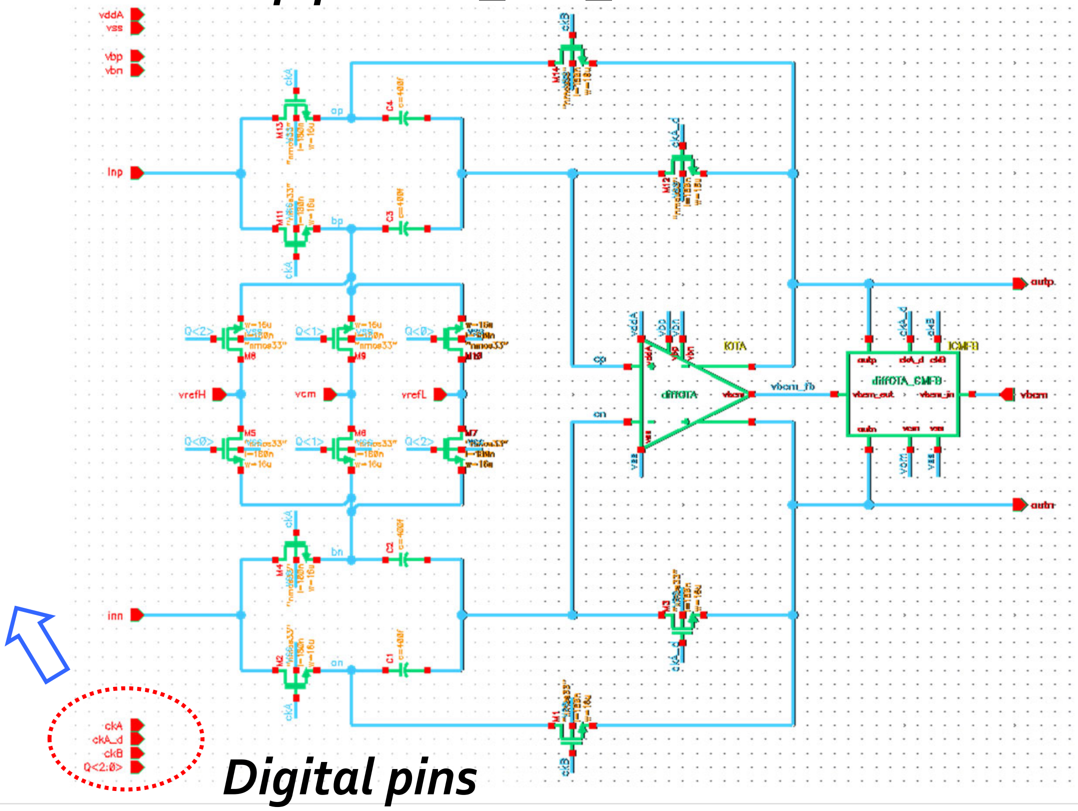


Edit Object Properties

Apply To	only current	pin
Show	☐ system ☐ user ✓ CDF	

Property	Value	Display
Name	ckA	value
Direction	input	
Usage	schematic	
Signal Type	signal	
Pwr Sensitivity	None	
Gnd Sensitivity	None	

CDF Parameter	Value	Display
XMODEL Signal Type	xbit	off
Description		off
Use Isolation Buffer	☐	off
Define Conversion Levels	✓	off
Level for Logic 1	3.3	off
Level for Logic 0	0.0	off
Threshold for Logic 1	1.65	off
Threshold for Logic 0	1.65	off
Driving Strength	strong	off



Digital pins

CLM Extraction Approach (2)

- Specify a list of digital cells in ***tech_config.py*** file so that *MODELZEN* can extract simpler models for them
  - 1.2V logic gates: *std_inv, std_mux, std_fulladd, std_dff, …*
  - 3.3V logic gates: *hv_inv, hv_nand2, hv_nor2, hv_xor, …*
  - 1.2V ↔3.3V level converters: *conv_hi2low, conv_low2hi*

```
# subckt-specific device mapping
cells_digital = [
    wildcard("std_*"),
    wildcard("hv_*"),
    wildcard("conv_*"),
]
devicemap_digital = derive_devicemap(devo_devicemap, convto_digital)
update_subcktmap(cells_digital, devicemap_digital)
```

CLM Extraction Approach (3)

- Specify a list of switch cells in ***tech_config.py*** file so that their transistors can be extracted as switches
  - e.g. switches used for switched-capacitor circuits

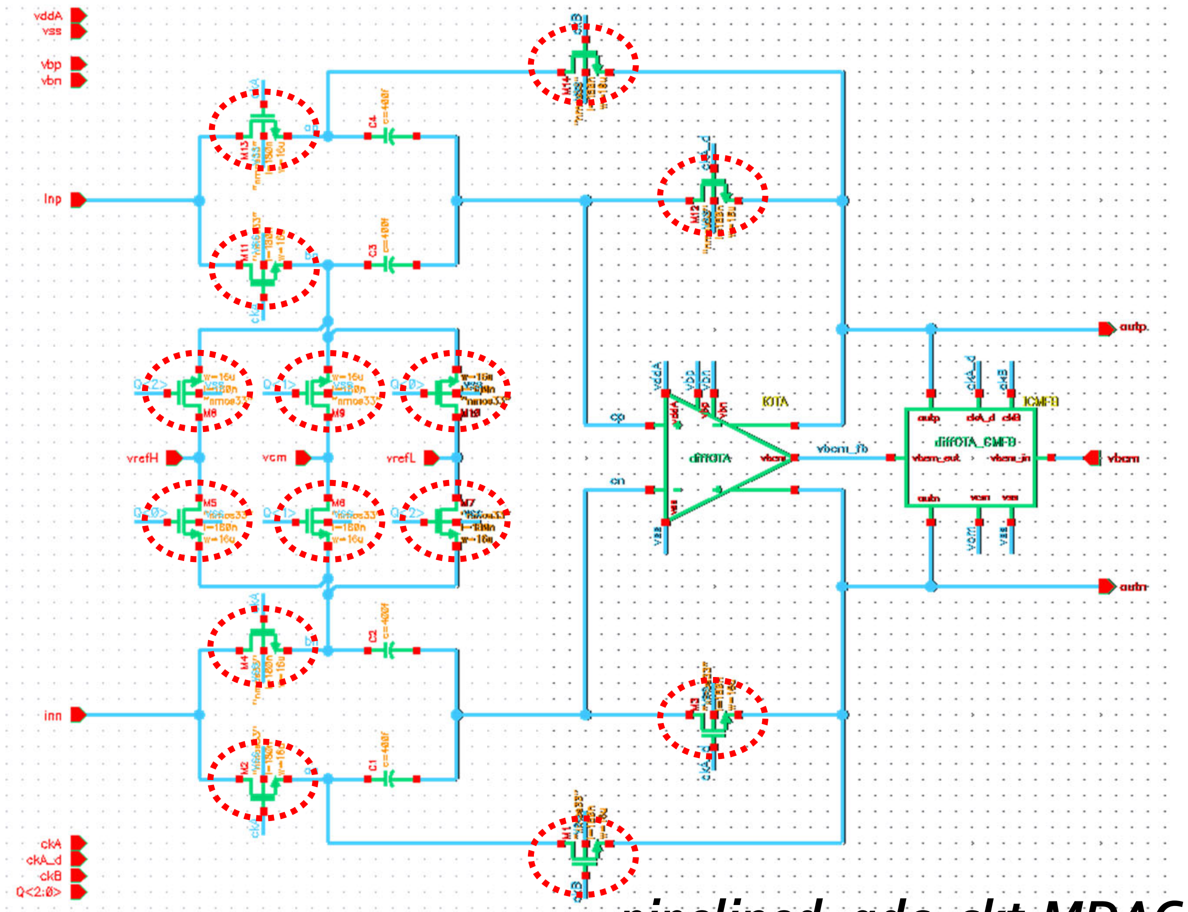
```
cells_switch = [
    "diffOTA_CMFB",
    "sh_amp",
    "MDAC",
]
devicemap_switch = derive_devicemap(devo_devicemap, convto_switch)
update_subcktmap(cells_switch, devicemap_switch)
```

- Adding this option also helps:

```
devo_options['reduce_dev'] += ["nmosfet->switch", "pmosfet->switch"]
```

CLM Extraction Approach (3): MDAC

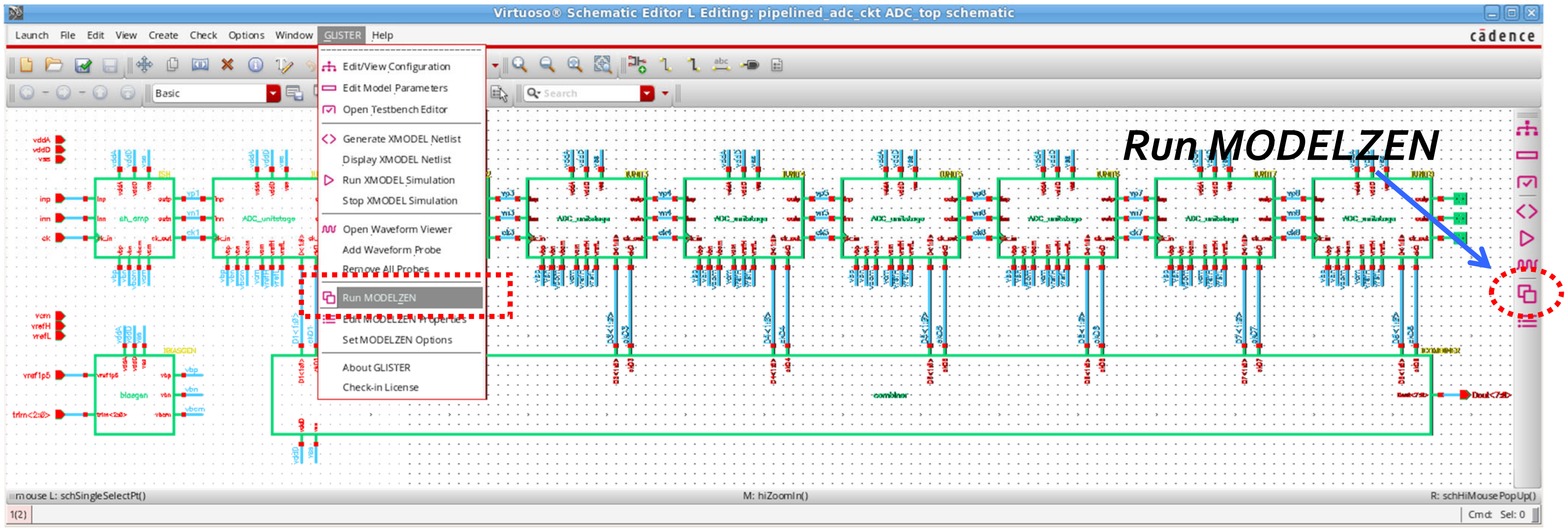
- We want to model the transistors in MDAC as switches



pipelined_adc_ckt.MDAC:schematic

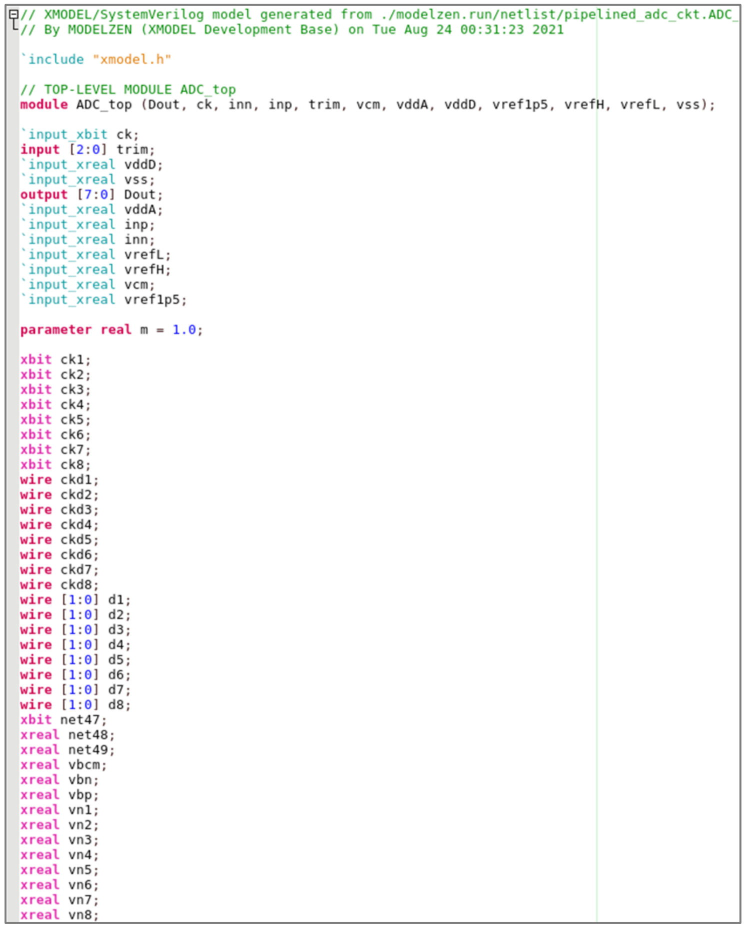
Bottom-up Model Extraction

- Just clicking "*Run MODELZEN*" icon on the top-level schematic extracts models from the entire circuit
- The extracted model is imported to an ***xmodel*** view

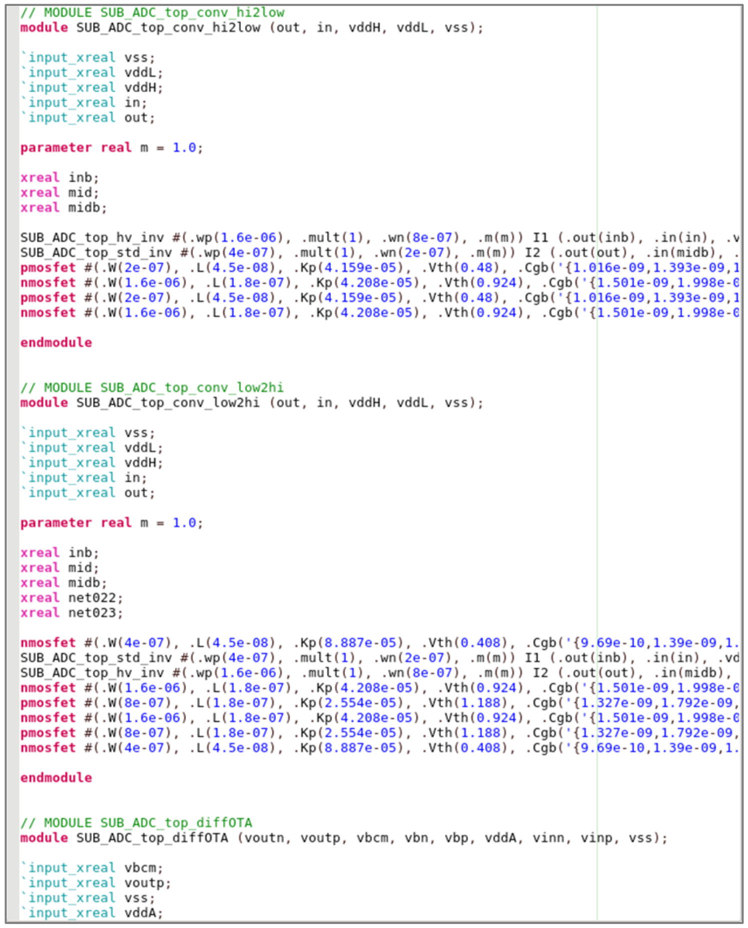


Extracted SystemVerilog Models

- *pipelined_adc_ckt.ADC_top:xmodel*



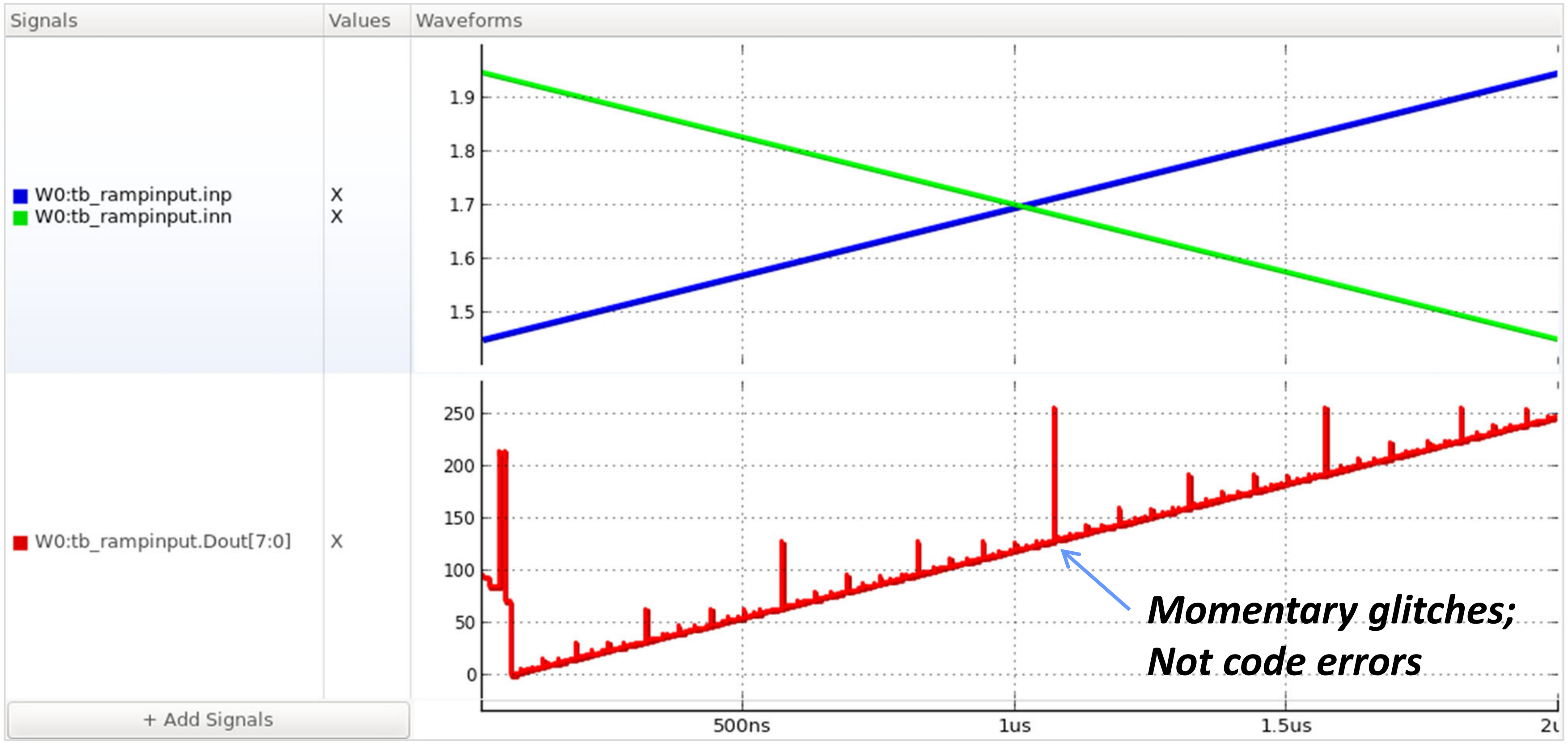
```
// XMODEL/SystemVerilog model generated from ./modelzen.run/netlist/pipelined_adc_ckt.ADC_
// By MODELZEN (XMODEL Development Base) on Tue Aug 24 00:31:23 2021

`include "xmodel.h"

// TOP-LEVEL MODULE ADC_top
module ADC_top (Dout, ck, inn, inp, trim, vcm, vddA, vddD, vref1p5, vrefH, vrefL, vss);

`input_xbit ck;
input [2:0] trim;
`input_xreal vddD;
`input_xreal vss;
output [7:0] Dout;
`input_xreal vddA;
`input_xreal inp;
`input_xreal inn;
`input_xreal vrefL;
`input_xreal vrefH;
`input_xreal vcm;
`input_xreal vref1p5;

parameter real m = 1.0;

xbit ck1;
xbit ck2;
xbit ck3;
xbit ck4;
xbit ck5;
xbit ck6;
xbit ck7;
xbit ck8;
wire ckd1;
wire ckd2;
wire ckd3;
wire ckd4;
wire ckd5;
wire ckd6;
wire ckd7;
wire ckd8;
wire [1:0] d1;
wire [1:0] d2;
wire [1:0] d3;
wire [1:0] d4;
wire [1:0] d5;
wire [1:0] d6;
wire [1:0] d7;
wire [1:0] d8;
xbit net47;
xreal net48;
xreal net49;
xreal vbcm;
xreal vbn;
xreal vbp;
xreal vn1;
xreal vn2;
xreal vn3;
xreal vn4;
xreal vn5;
xreal vn6;
xreal vn7;
xreal vn8;
```



```
// MODULE SUB_ADC_top_conv_hi2low
module SUB_ADC_top_conv_hi2low (out, in, vddH, vddL, vss);

`input_xreal vss;
`input_xreal vddL;
`input_xreal vddH;
`input_xreal in;
`input_xreal out;

parameter real m = 1.0;

xreal inb;
xreal mid;
xreal midb;

SUB_ADC_top_hv_inv #(.wp(1.6e-06), .mult(1), .wn(8e-07), .m(m)) I1 (.out(inb), .in(in), .v
SUB_ADC_top_std_inv #(.wp(4e-07), .mult(1), .wn(2e-07), .m(m)) I2 (.out(out), .in(midb), .
pmosfet #(.W(2e-07), .L(4.5e-08), .Kp(4.159e-05), .Vth(0.48), .Cgb('{1.016e-09,1.393e-09,1
nmosfet #(.W(1.6e-06), .L(1.8e-07), .Kp(4.208e-05), .Vth(0.924), .Cgb('{1.501e-09,1.998e-0
pmosfet #(.W(2e-07), .L(4.5e-08), .Kp(4.159e-05), .Vth(0.48), .Cgb('{1.016e-09,1.393e-09,1
nmosfet #(.W(1.6e-06), .L(1.8e-07), .Kp(4.208e-05), .Vth(0.924), .Cgb('{1.501e-09,1.998e-0

endmodule


// MODULE SUB_ADC_top_conv_low2hi
module SUB_ADC_top_conv_low2hi (out, in, vddH, vddL, vss);

`input_xreal vss;
`input_xreal vddL;
`input_xreal vddH;
`input_xreal in;
`input_xreal out;

parameter real m = 1.0;

xreal inb;
xreal mid;
xreal midb;
xreal net022;
xreal net023;

nmosfet #(.W(4e-07), .L(4.5e-08), .Kp(8.887e-05), .Vth(0.408), .Cgb('{9.69e-10,1.39e-09,1.
SUB_ADC_top_std_inv #(.wp(4e-07), .mult(1), .wn(2e-07), .m(m)) I1 (.out(inb), .in(in), .vd
SUB_ADC_top_hv_inv #(.wp(1.6e-06), .mult(1), .wn(8e-07), .m(m)) I2 (.out(out), .in(midb),
nmosfet #(.W(1.6e-06), .L(1.8e-07), .Kp(4.208e-05), .Vth(0.924), .Cgb('{1.501e-09,1.998e-0
pmosfet #(.W(8e-07), .L(1.8e-07), .Kp(2.554e-05), .Vth(1.188), .Cgb('{1.327e-09,1.792e-09,
nmosfet #(.W(1.6e-06), .L(1.8e-07), .Kp(4.208e-05), .Vth(0.924), .Cgb('{1.501e-09,1.998e-0
pmosfet #(.W(8e-07), .L(1.8e-07), .Kp(2.554e-05), .Vth(1.188), .Cgb('{1.327e-09,1.792e-09,
nmosfet #(.W(4e-07), .L(4.5e-08), .Kp(8.887e-05), .Vth(0.408), .Cgb('{9.69e-10,1.39e-09,1.

endmodule


// MODULE SUB_ADC_top_diffOTA
module SUB_ADC_top_diffOTA (voutn, voutp, vbcm, vbn, vbp, vddA, vinn, vinp, vss);

`input_xreal vbcm;
`input_xreal voutp;
`input_xreal vss;
`input_xreal vddA;
```

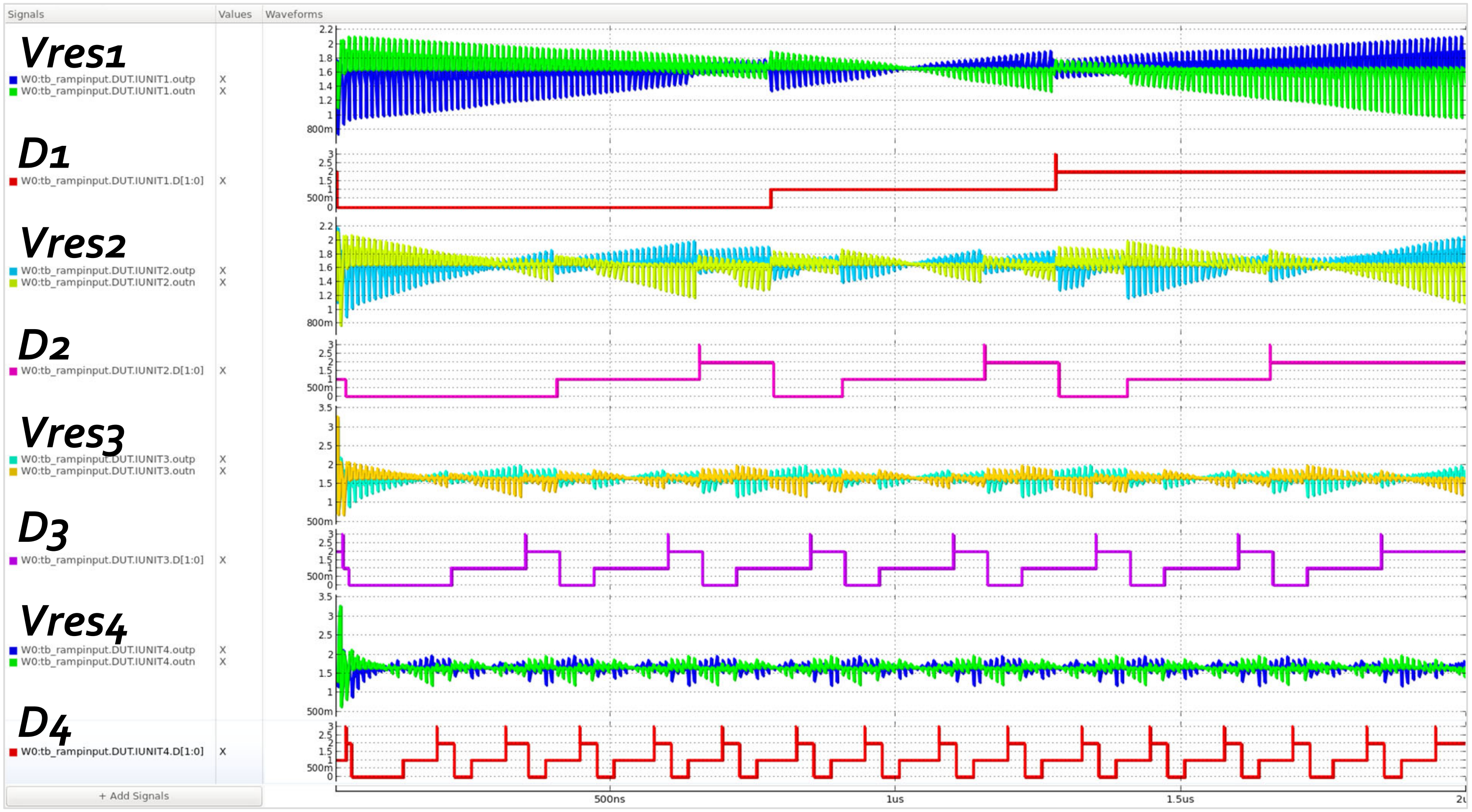
Simulated Results: *tb_rampinput*

- ***pipelined_adc_ckt.ADC_top:tb_rampinput***
- Simulation runtime: ~19 min.



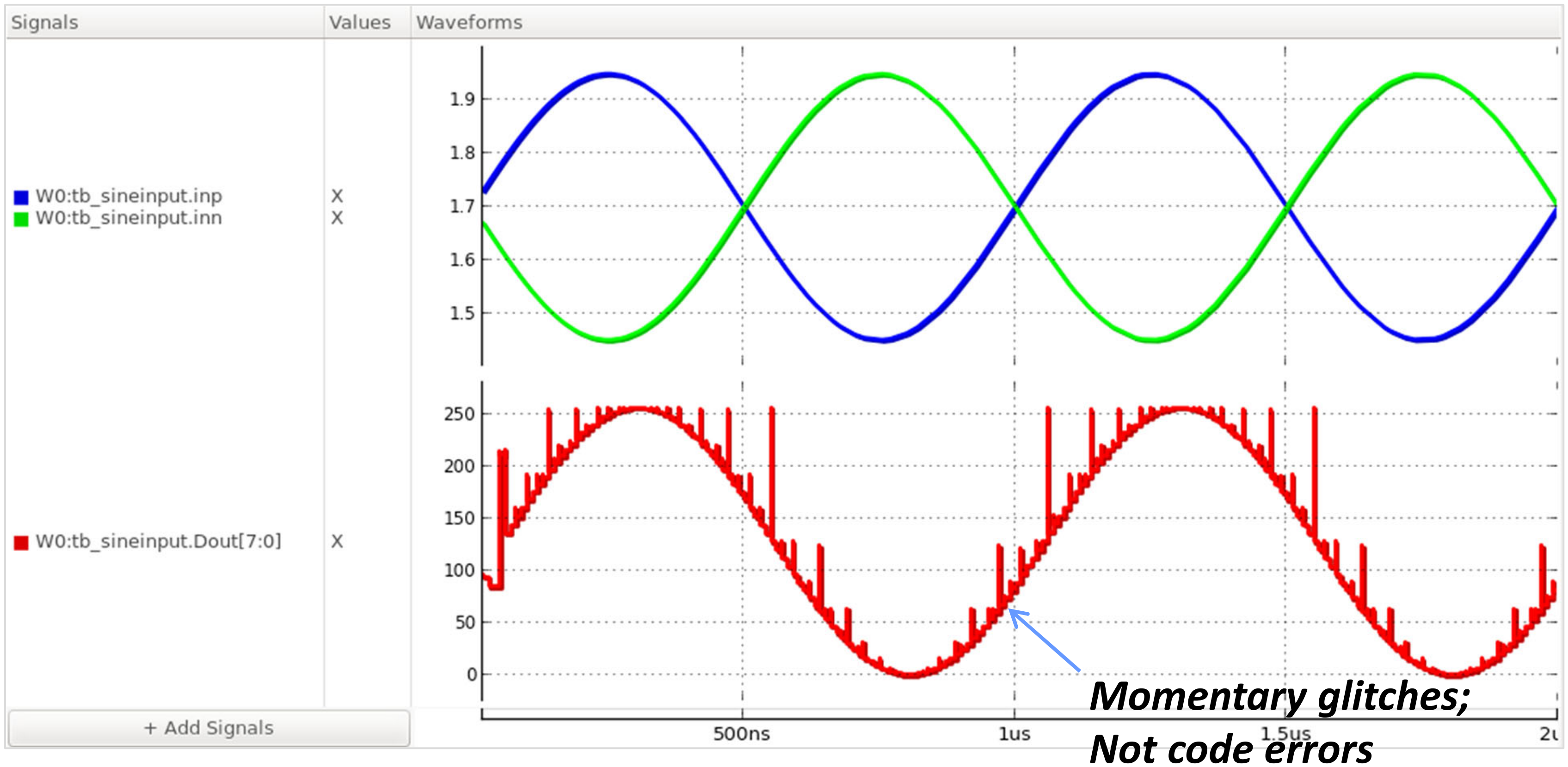
Simulated Results: *tb_rampinput* (2)

- Digital and residual outputs of the individual stages:



Simulated Results: *tb_sineinput*

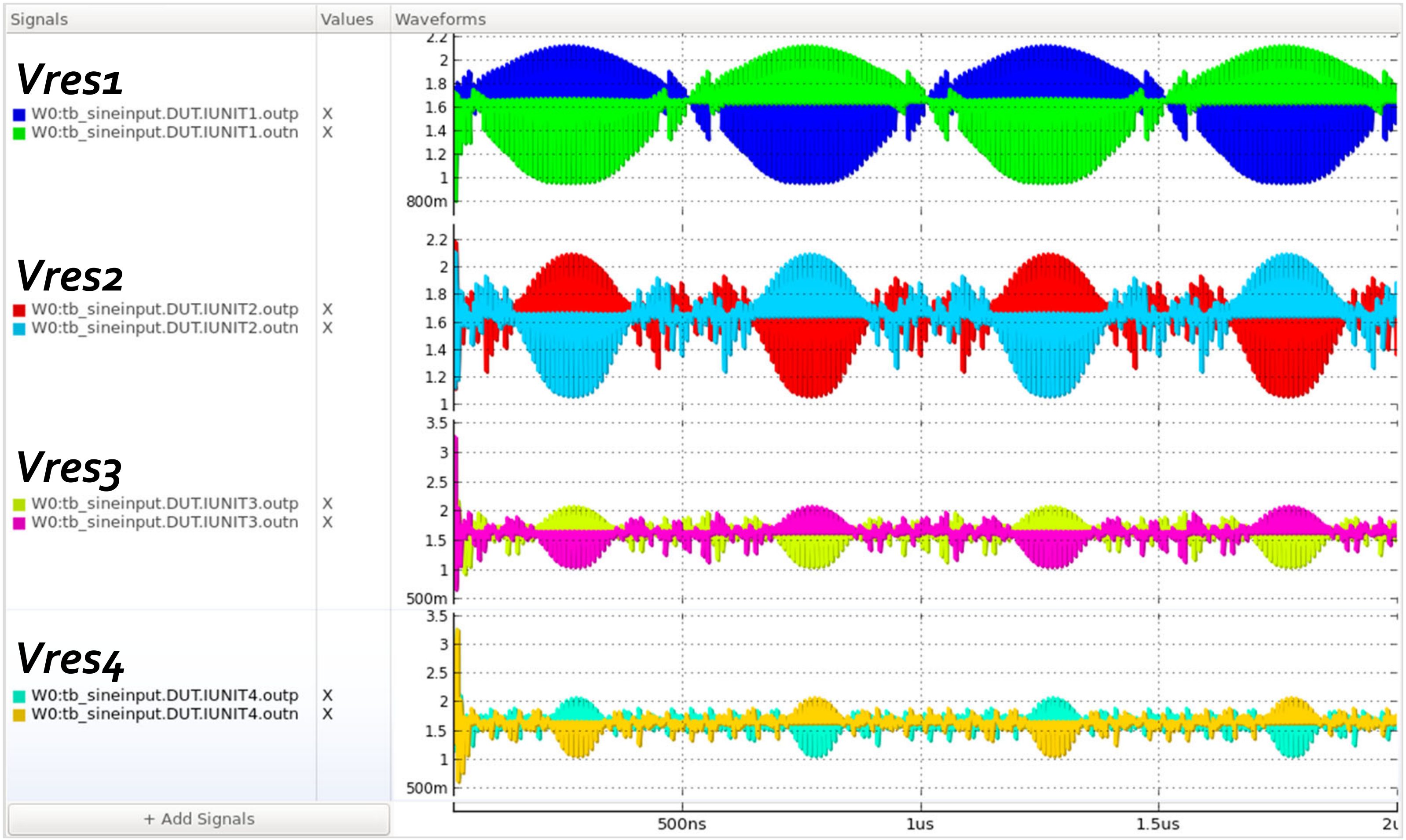
- ***pipelined_adc_ckt.ADC_top:tb_sineinput***
- Simulation runtime: ~20 min.



Momentary glitches; Not code errors

Simulated Results: *tb_sineinput* (2)

- The residual outputs of the individual stages:


Signals
Values
Waveforms
Vres1
🟦 W0:tb_sineinput.DUT.IUNIT1.outp X
🟩 W0:tb_sineinput.DUT.IUNIT1.outn X
Vres2
🟥 W0:tb_sineinput.DUT.IUNIT2.outp X
🟦 W0:tb_sineinput.DUT.IUNIT2.outn X
Vres3
🟩 W0:tb_sineinput.DUT.IUNIT3.outp X
🟪 W0:tb_sineinput.DUT.IUNIT3.outn X
Vres4
🟦 W0:tb_sineinput.DUT.IUNIT4.outp X
🟨 W0:tb_sineinput.DUT.IUNIT4.outn X
+ Add Signals
2.2
2
1.8
1.6
1.4
1.2
1
800m
3.5
3
2.5
1.5
500m
500ns
1us
1.5us

Structural Model Generation

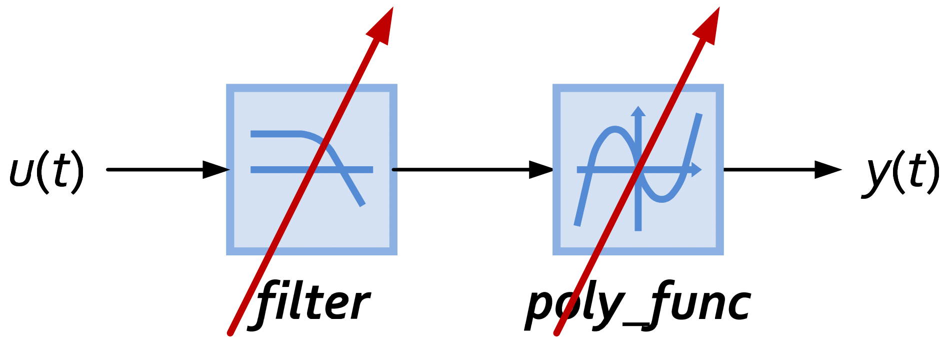
- ***Pros:*** enables a ***push-button flow*** generating correct-by-construction, SPICE-accurate analog models without requiring analog expertise

- ***Cons:*** the resulting circuit-level models are ***low-abstraction, transistor-based models*** that have limited simulation speeds

- ***Q:* Can we generate *functional models* using the *MODELZEN*'s push-button flow?**

Functional Model Generation

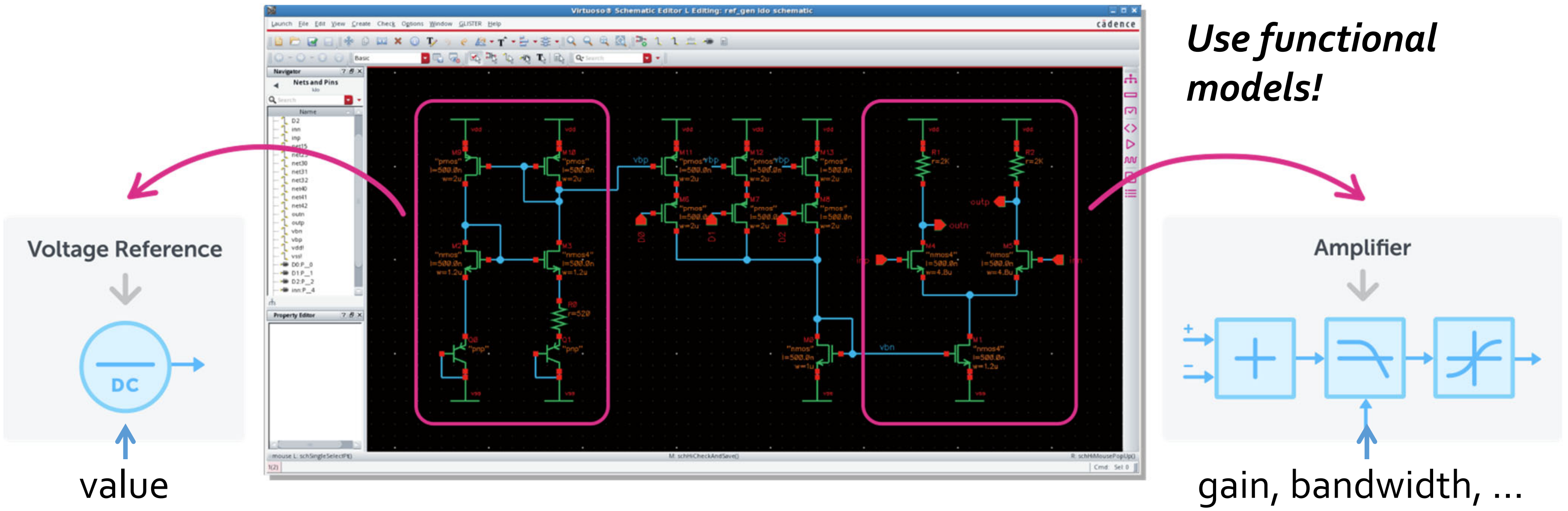
- Functional modeling focuses on the circuit's functions instead of its structure (i.e. topology):
  - Choose a template model based on the circuit's functions
  - Calibrate its model parameters via SPICE characterization



Parameter Calibration

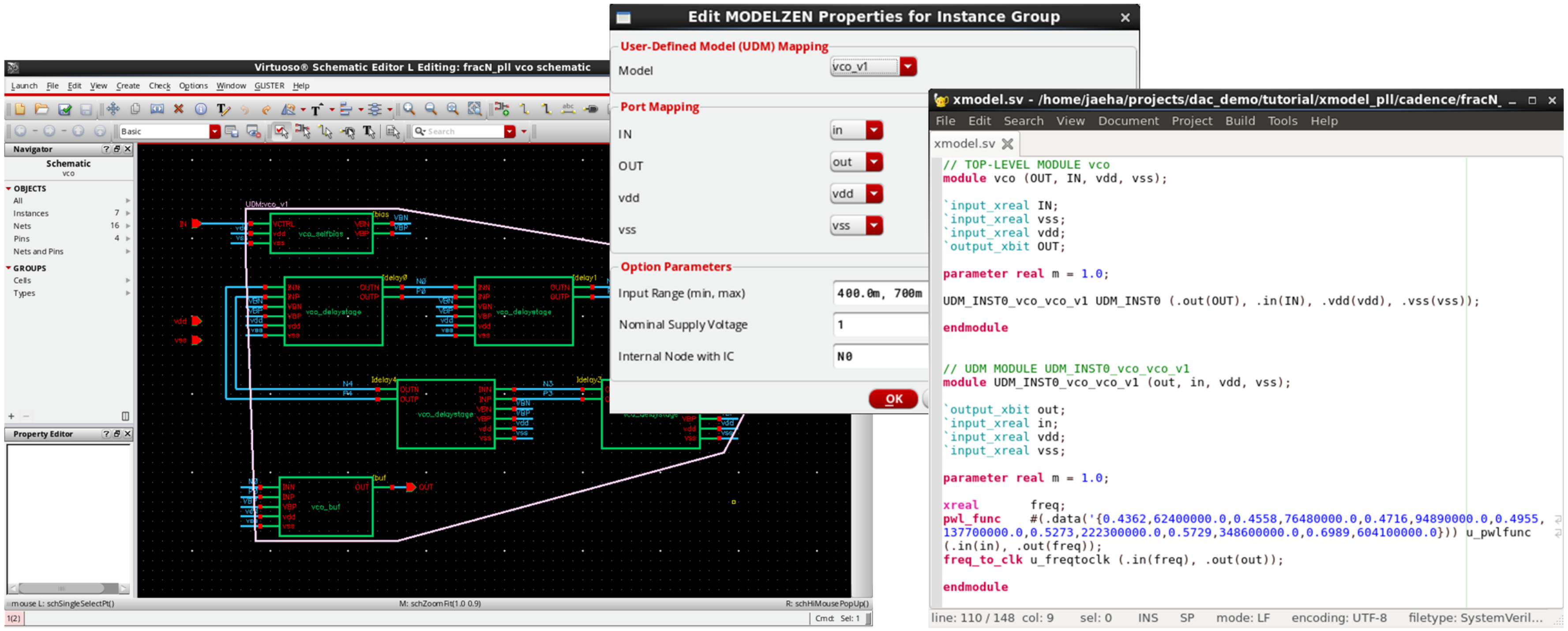
User-Defined Model (UDM)

- The ***User-Defined Model*** **(*UDM*)** interface of *MODELZEN* lets you generate higher-abstraction models with SPICE-calibrated parameters for any selected parts of the circuits



Functional Modeling with UDM

- For instance, you can auto-generate a functional model for an oscillator circuit of which frequency characteristics are calibrated by SPICE simulation


Virtuoso® Schematic Editor L Editing: fracN_pll vco schematic
Edit MODELZEN Properties for Instance Group
xmodel.sv - /home/jaeha/projects/dac_demo/tutorial/xmodel_pll/cadence/fracN_
File Edit Search View Document Project Build Tools Help
OK

User-Defined Model (UDM) Mapping

Model: vco_v1

Port Mapping

IN	in
OUT	out
vdd	vdd
vss	vss

Option Parameters

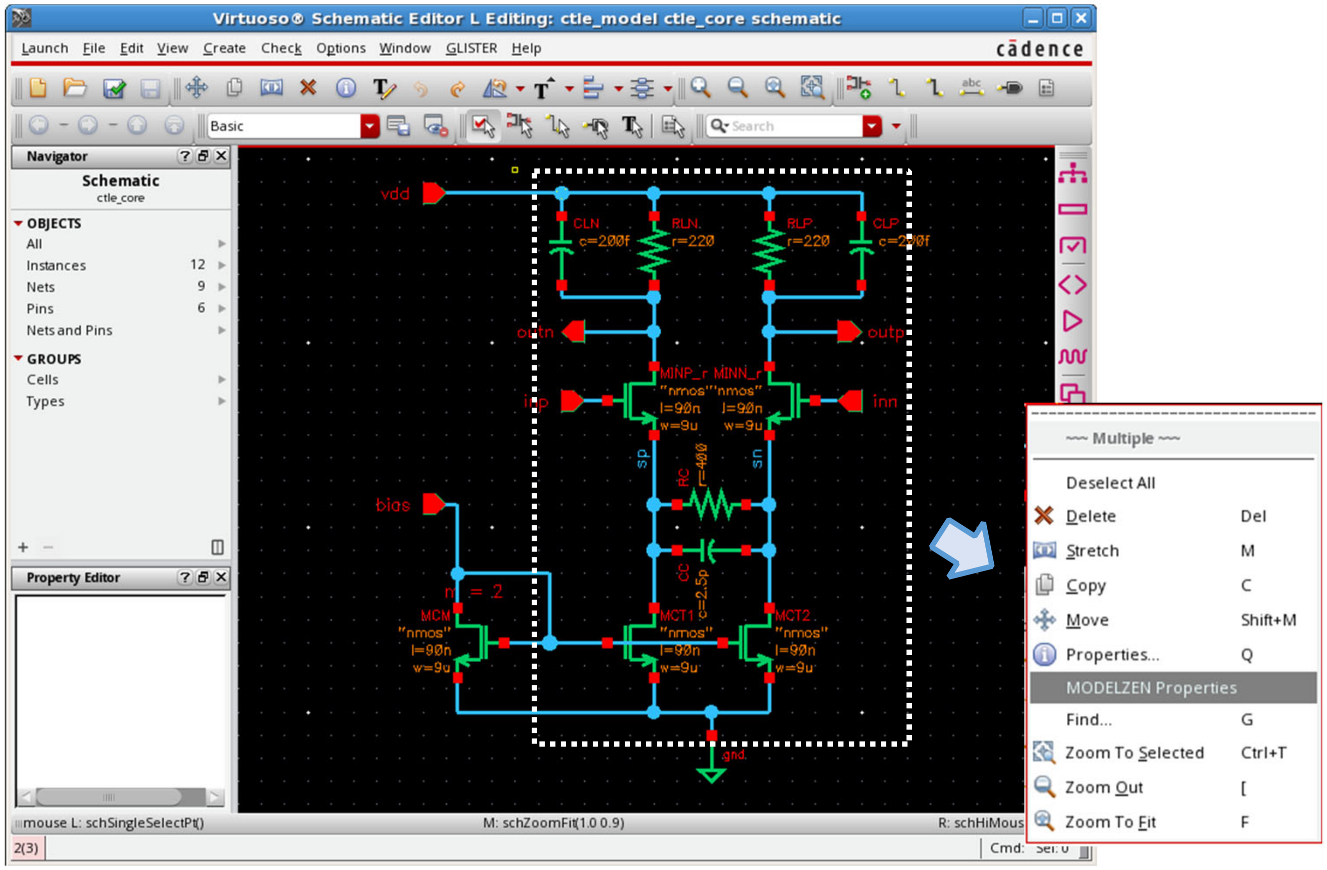
Input Range (min, max)	400.0m, 700m
Nominal Supply Voltage	1
Internal Node with IC	N0

```
// TOP-LEVEL MODULE vco
module vco (OUT, IN, vdd, vss);

`input_xreal IN;
`input_xreal vss;
`input_xreal vdd;
`output_xbit OUT;

parameter real m = 1.0;

UDM_INST0_vco_vco_v1 UDM_INST0 (.out(OUT), .in(IN), .vdd(vdd), .vss(vss));

endmodule


// UDM MODULE UDM_INST0_vco_vco_v1
module UDM_INST0_vco_vco_v1 (out, in, vdd, vss);

`output_xbit out;
`input_xreal in;
`input_xreal vdd;
`input_xreal vss;

parameter real m = 1.0;

xreal       freq;
pwl_func    #(.data('{0.4362,62400000.0,0.4558,76480000.0,0.4716,94890000.0,0.4955,
137700000.0,0.5273,222300000.0,0.5729,348600000.0,0.6989,604100000.0})) u_pwlfunc
(.in(in), .out(freq));
freq_to_clk u_freqtoclk (.in(freq), .out(out));

endmodule
```

```
line: 110 / 148  col: 9     sel: 0     INS     SP     mode: LF     encoding: UTF-8     filetype: SystemVeril...
```

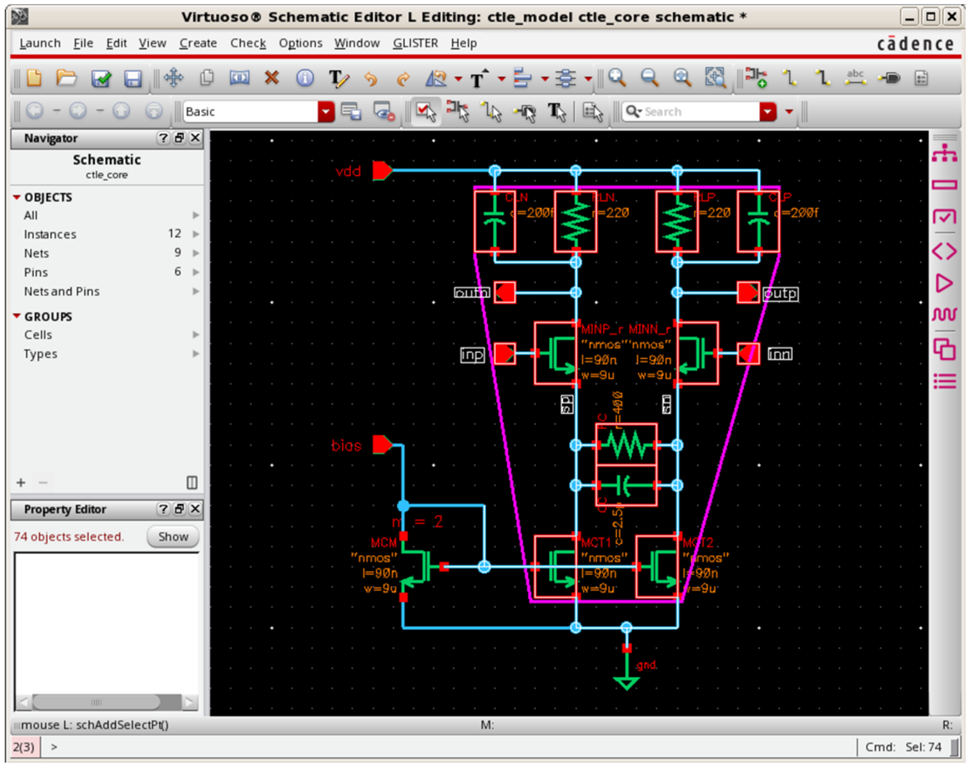
Defining UDM Mapping on Circuits

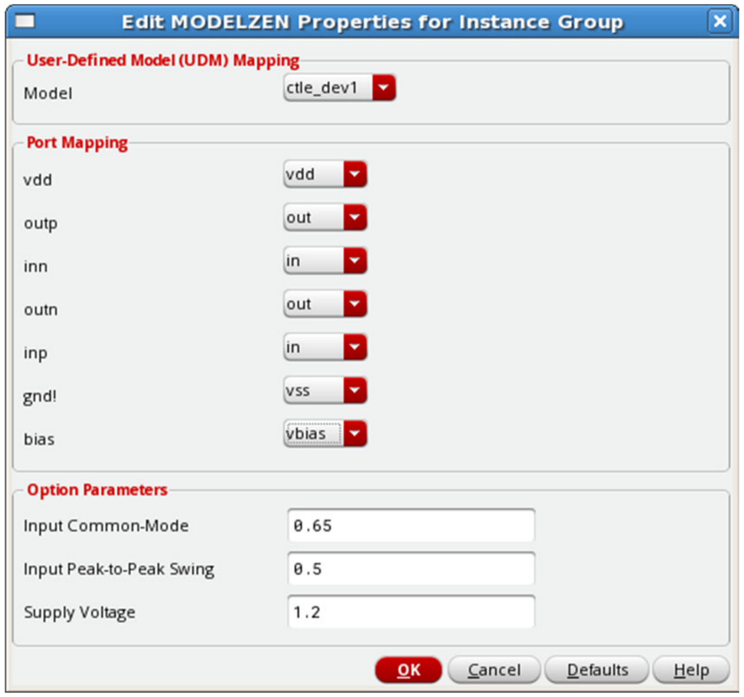
- Select part of the circuits to be mapped to a UDM and right button click on ***"MODELZEN Properties"***



Defining UDM Mapping on Circuits (2)

- *Edit MODELZEN Properties* dialog window will appear where you can map each "pseudo-terminal" to a UDM port and define UDM parameters




Edit MODELZEN Properties for Instance Group

User-Defined Model (UDM) Mapping

Model	ctle_dev1

Port Mapping

vdd	vdd
outp	out
inn	in
outn	out
inp	in
gnd!	vss
bias	vbias

Option Parameters

Input Common-Mode	0.65
Input Peak-to-Peak Swing	0.5
Supply Voltage	1.2

OK | Cancel | Defaults | Help

UDM Definition

- Each UDM is a Python class that defines:
  - A list of terminals and option parameters
  - Functional model template
  - SPICE simulation steps to fit the model parameters

```
class pwm_v1 (devo_udm):
    terms = {
        'in'    : dict(direction="input", sigtype="xreal", width=1),
        'out'   : dict(direction="output", sigtype="xbit", width=1),
        'vpwr'  : dict(direction="input", sigtype="xreal", width=1,
                       paramtype='real', prompt='Value', default=1.2),
        'vgnd'  : dict(direction="input", sigtype="xreal", width=1,
                       paramtype='real', prompt='Value', default=0.0),
        'vbias' : dict(direction="input", sigtype="xreal", width=[0,None],
                       paramtype='real', prompt='Value', default=0.7),
        'ibias' : dict(direction="input", sigtype="xreal", width=[0,None],
                       paramtype='real', prompt='Value', default=0.0),
    }

    params = {
        'range_in'   : dict(type="real_array", prompt="Input Range (min, max)", default=[0.2,0.8]),
    }
```

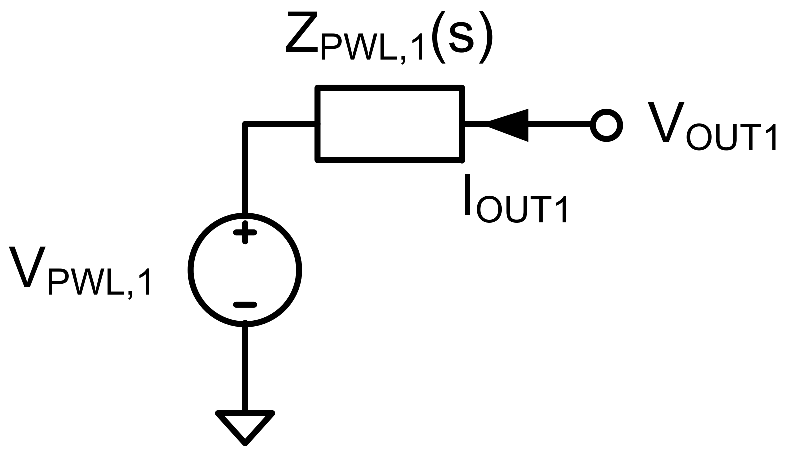
Quest for Analog Model Templates

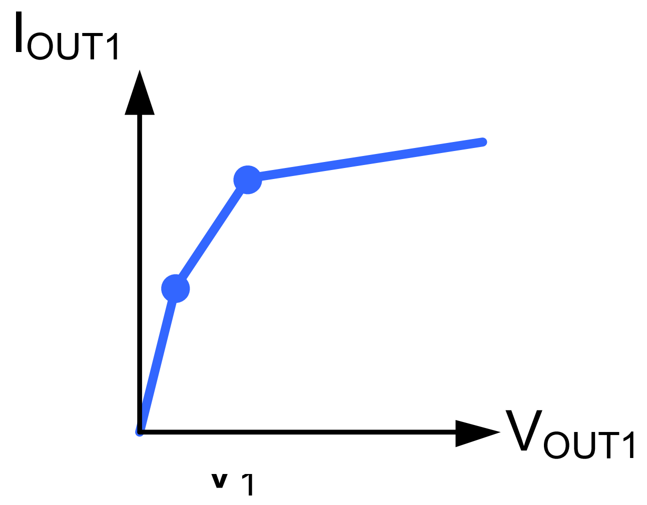
- Many circuits with analog inputs & outputs fall into:
  - One-port model
  - Multi-port impedance model
  - Multi-port amplifier model

- Other circuits with digital inputs or outputs:
  - Comparators and slicers
  - Digital logic gates, flip-flops, and latches
  - Delay lines and oscillators

One-Port Model

- For circuits producing voltages or currents without inputs (e.g. reference generators)
  - Thevenin/Norton-equivalent circuit models finite R_{out}
  - V_1 and R_1 can be PWL functions to model nonlinearity
  - R_1 can be $Z_1(s)$ to model AC impedance





Flexible Port Widths

- Many of our UDM ports have variable widths, so that each UDM can be mapped to a variety of circuits

- For example, our ***refgen*** UDM modeling one-port circuits can have arbitrary number of ***vout***'s and ***iout***'s

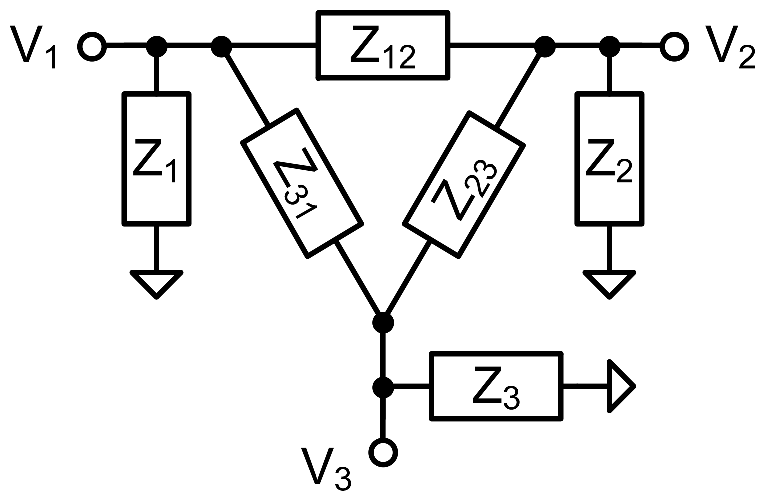
UDM Port	Description	Parameter
vout	Voltage output	Iout bias
iout	Current output	Vout bias
mode	Digital mode input	Voltage level : *level1*[, *level0*]

Digital Mode Inputs

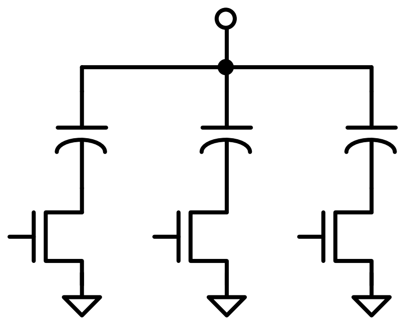
- Many analog circuits have ***digital mode inputs*** for various purposes: e.g. trimming, power-down, enable, …
  - These inputs don't change the model template, but change the model parameter values
  - e.g. changing the output level or impedance
- Our UDMs support arbitrary number of ***'mode'*** inputs
  - Generated model contains a look-up table defining parameter values for each combination of digital modes

Multi-Port Impedance Model

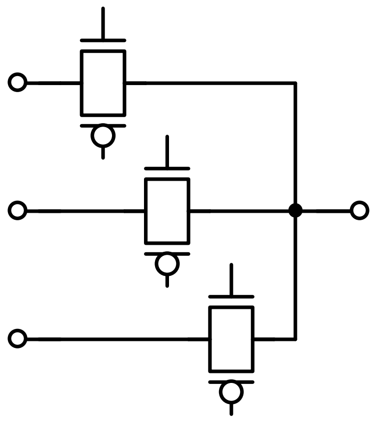
- For circuits having impedances between multiple ports
  - Often, digital modes control their impedance values
  - e.g. digitally-controlled resistors and capacitors, analog multiplexer/demultiplexers (switch networks)



Digitally-adjustable Capacitor

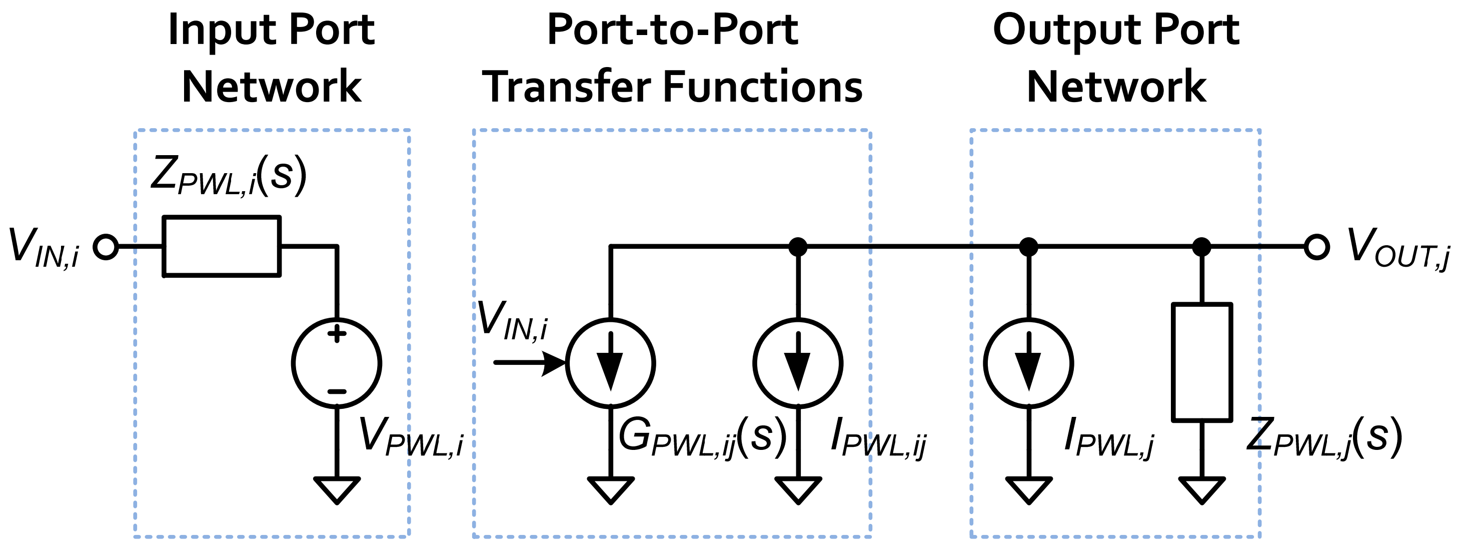


Analog Mux/Demux



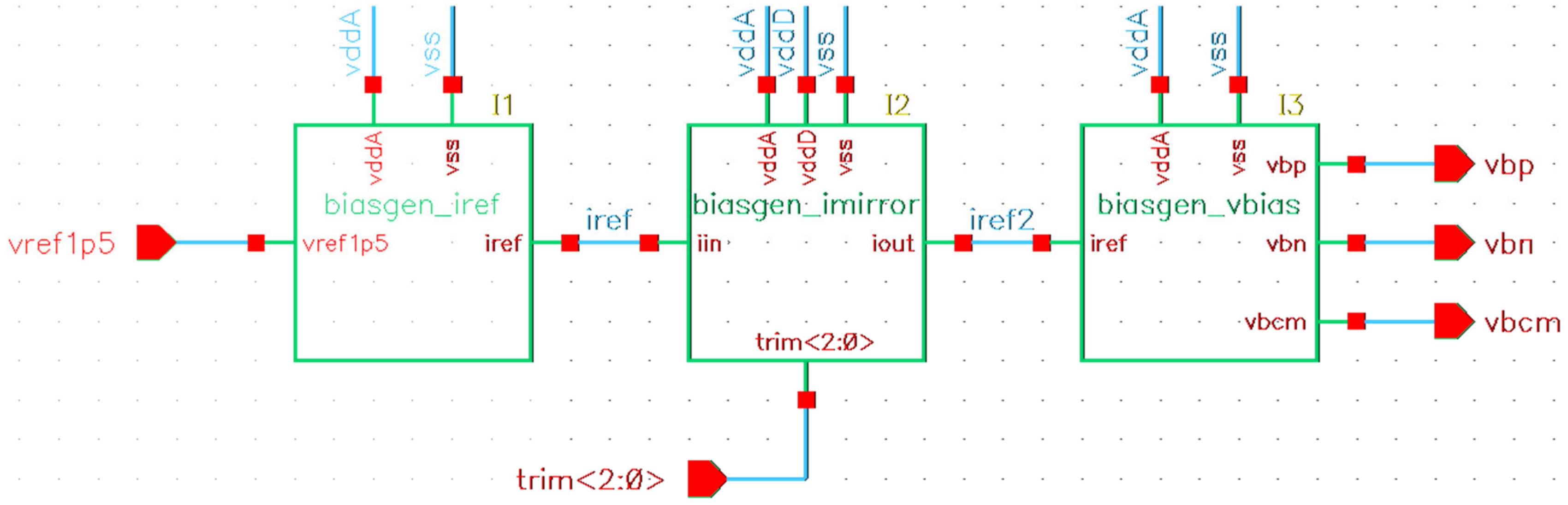
Multi-Port Amplifier Model

- For circuits amplifying/filtering from inputs to outputs
  - Each input/output port network with finite and/or nonlinear $Z_{in}(s)$ or $Z_{out}(s)$
  - Port-to-port transfer functions model DC gain, AC TF, and/or nonlinearity between input & output ports



Bias Generator: *biasgen*

- Generates digitally-trimmable bias voltages via:
  - First stage generating a reference current (***iref***) from ***vref***
  - Second stage scaling the current with ***trim*<2:0>**
  - Third stage converting the current into ***vbp***, ***vbn*** & ***vbcm***

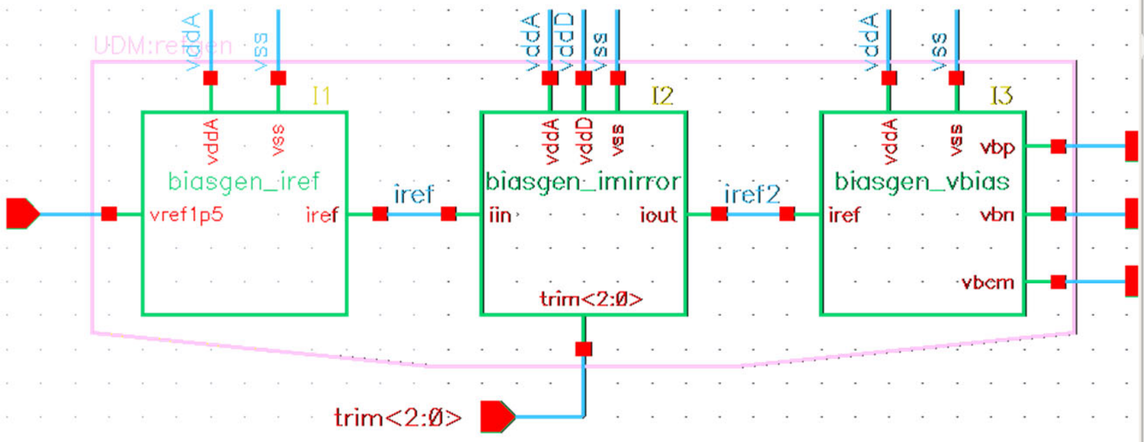


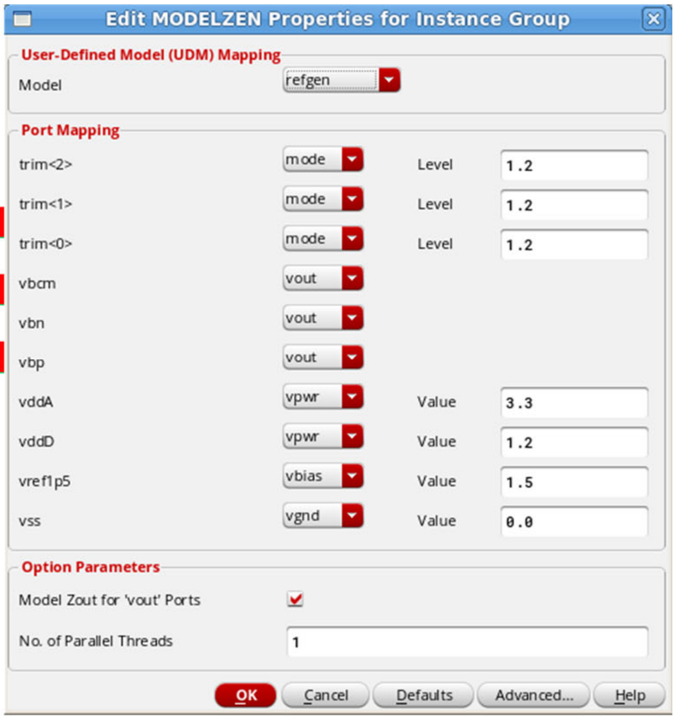
pipelined_adc_ckt.biasgen:schematic

UDM *refgen* for *biasgen*

- Models a circuit generating reference voltages or currents controlled by digital *mode* inputs

UDM:refgen




Edit MODELZEN Properties for Instance Group

User-Defined Model (UDM) Mapping

Model	refgen

Port Mapping

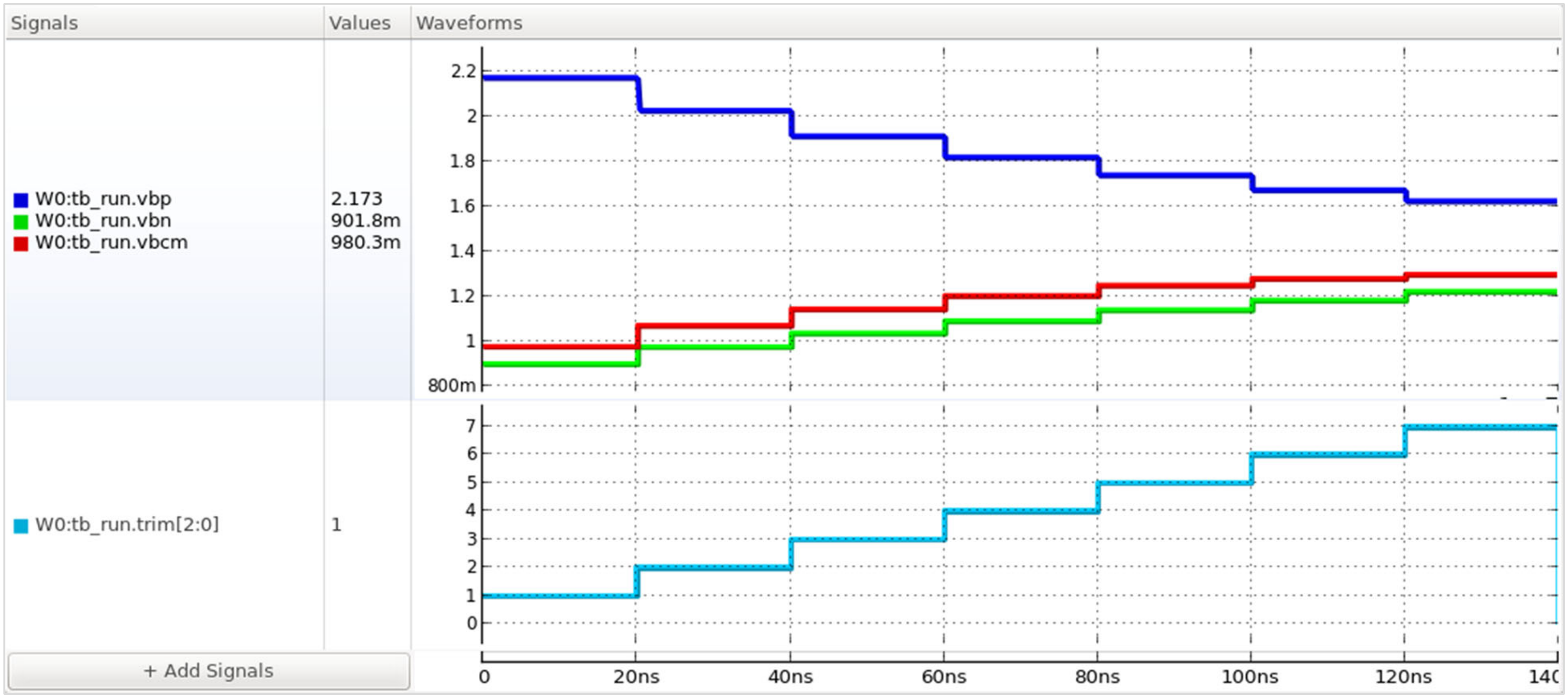
trim<2>	mode	Level	1.2
trim<1>	mode	Level	1.2
trim<0>	mode	Level	1.2
vbcm	vout		
vbn	vout		
vbp	vout		
vddA	vpwr	Value	3.3
vddD	vpwr	Value	1.2
vref1p5	vbias	Value	1.5
vss	vgnd	Value	0.0

Option Parameters

Model Zout for 'vout' Ports	✓
No. of Parallel Threads	1

Simulation Results: *biasgen*

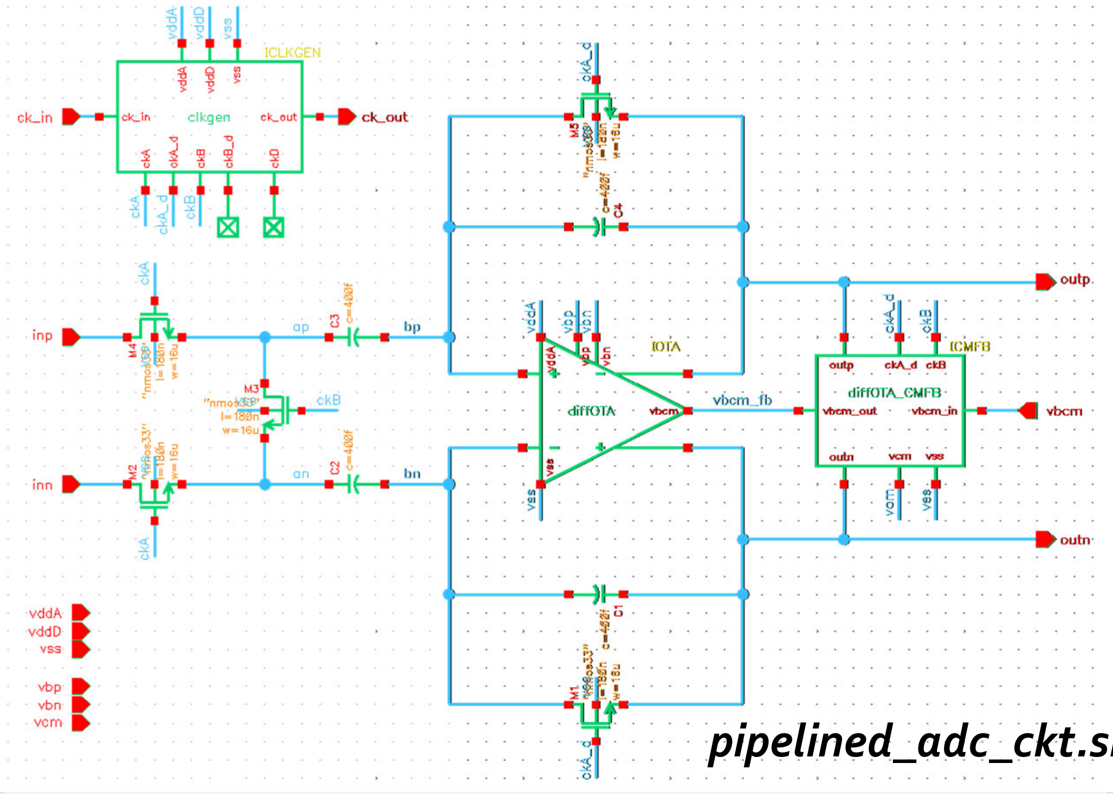
- Testbench: ***pipelined_adc_ckt.biasgen:tb_run***
- Measuring the bias voltage levels while varying ***trim***



Signals	Values
🟦 W0:tb_run.vbp	2.173
🟩 W0:tb_run.vbn	901.8m
🟥 W0:tb_run.vbcm	980.3m
🟦 W0:tb_run.trim[2:0]	1

Sample-and-Hold Stage: *sh_amp*

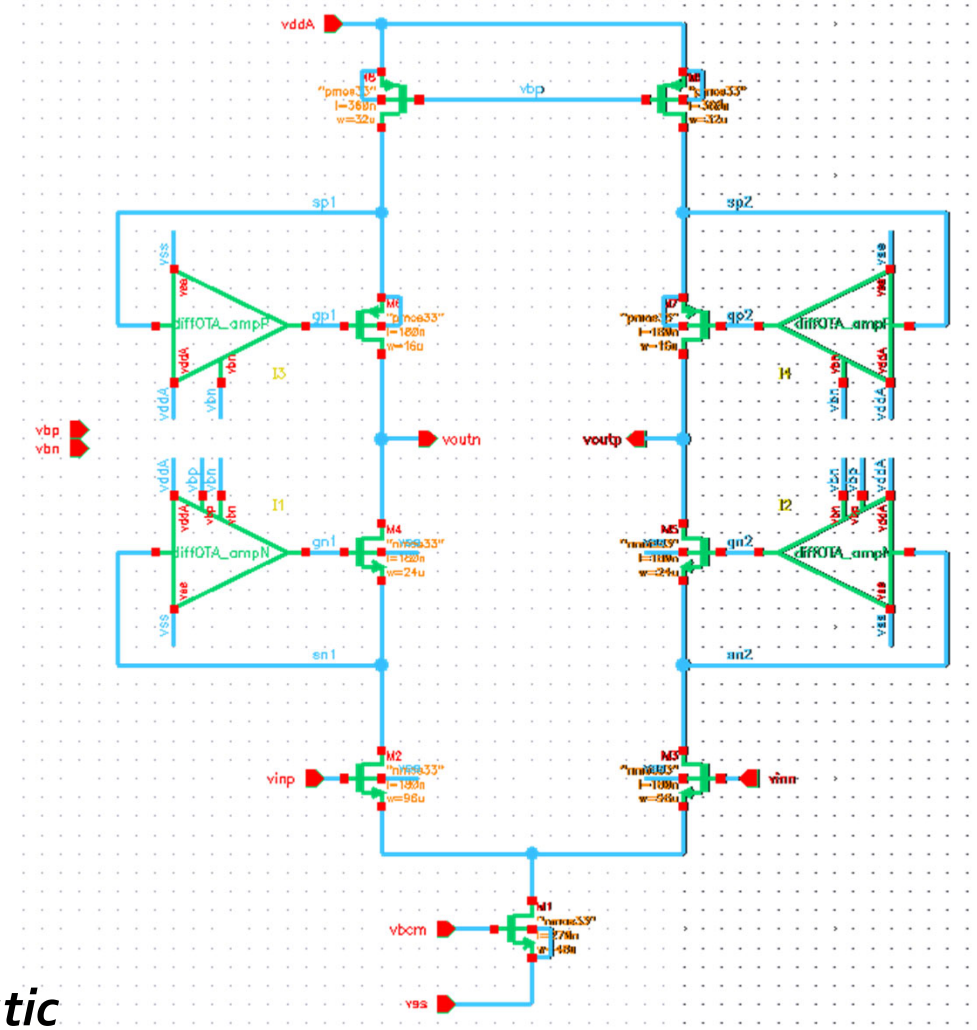
- A switched-capacitor sample-and-hold amplifier
  - A ***common-mode feedback*** (***CMFB***) circuit maintains the output common-mode level at ***vcm***



pipelined_adc_ckt.sh_amp:schematic

Differential OTA: *diffOTA*

- Designed as "***telescopic OTA***" with boosted cascodes
  - Providing a large gain with a single stage

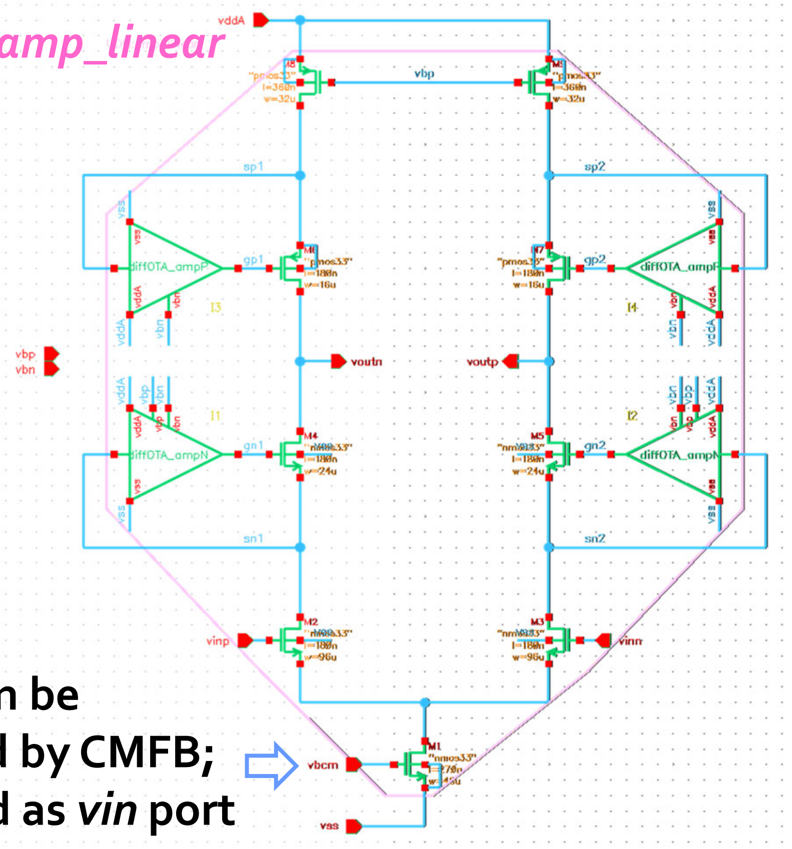


pipelined_adc_ckt.diffOTA:schematic

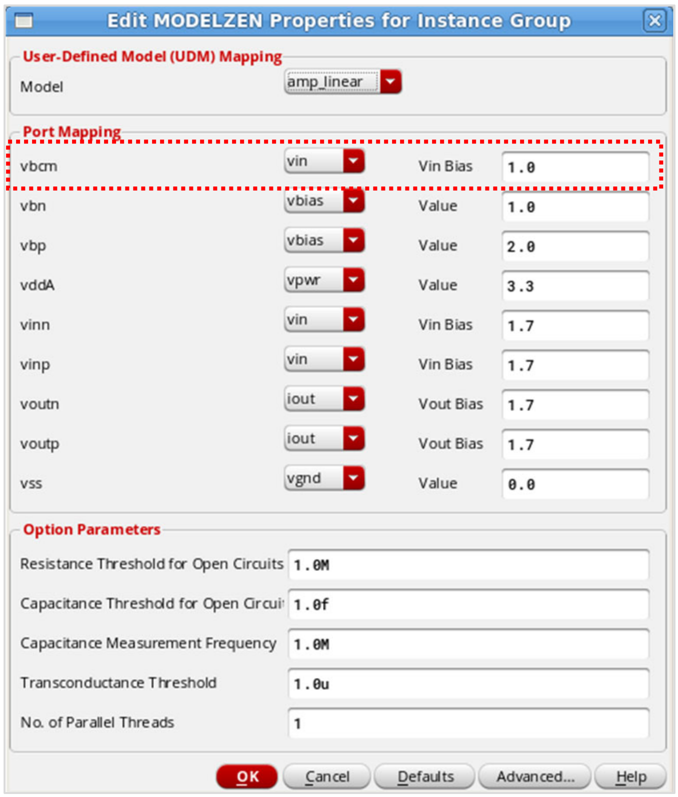
UDM *amp_linear* for *diffOTA*

- Models linear amplifier networks with voltage or current input/output's

UDM:amp_linear



vbcm can be adjusted by CMFB; modeled as *vin* port


Edit MODELZEN Properties for Instance Group

User-Defined Model (UDM) Mapping

Model: amp_linear

Port Mapping

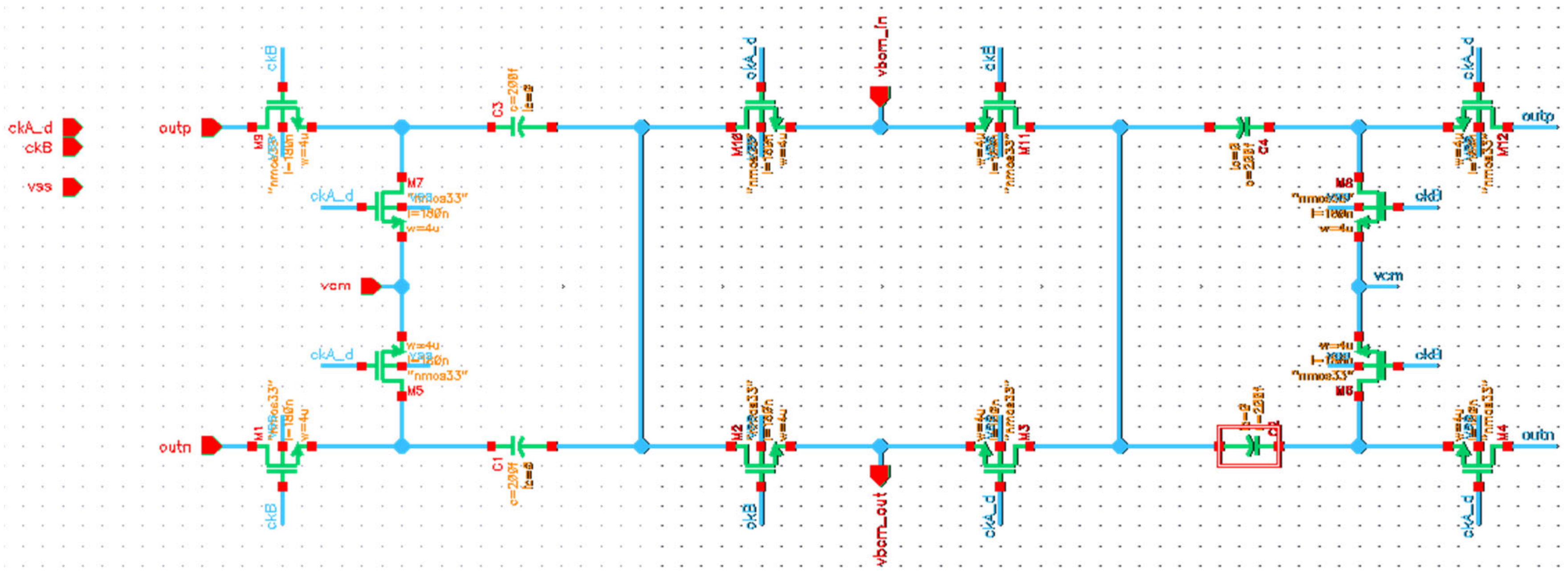
vbcm	vin	Vin Bias	1.0
vbn	vbias	Value	1.0
vbp	vbias	Value	2.0
vddA	vpwr	Value	3.3
vinn	vin	Vin Bias	1.7
vinp	vin	Vin Bias	1.7
voutn	iout	Vout Bias	1.7
voutp	iout	Vout Bias	1.7
vss	vgnd	Value	0.0

Option Parameters

Resistance Threshold for Open Circuits	1.0M
Capacitance Threshold for Open Circui	1.0f
Capacitance Measurement Frequency	1.0M
Transconductance Threshold	1.0u
No. of Parallel Threads	1

CMFB Circuit: *diffOTA_CMFB*

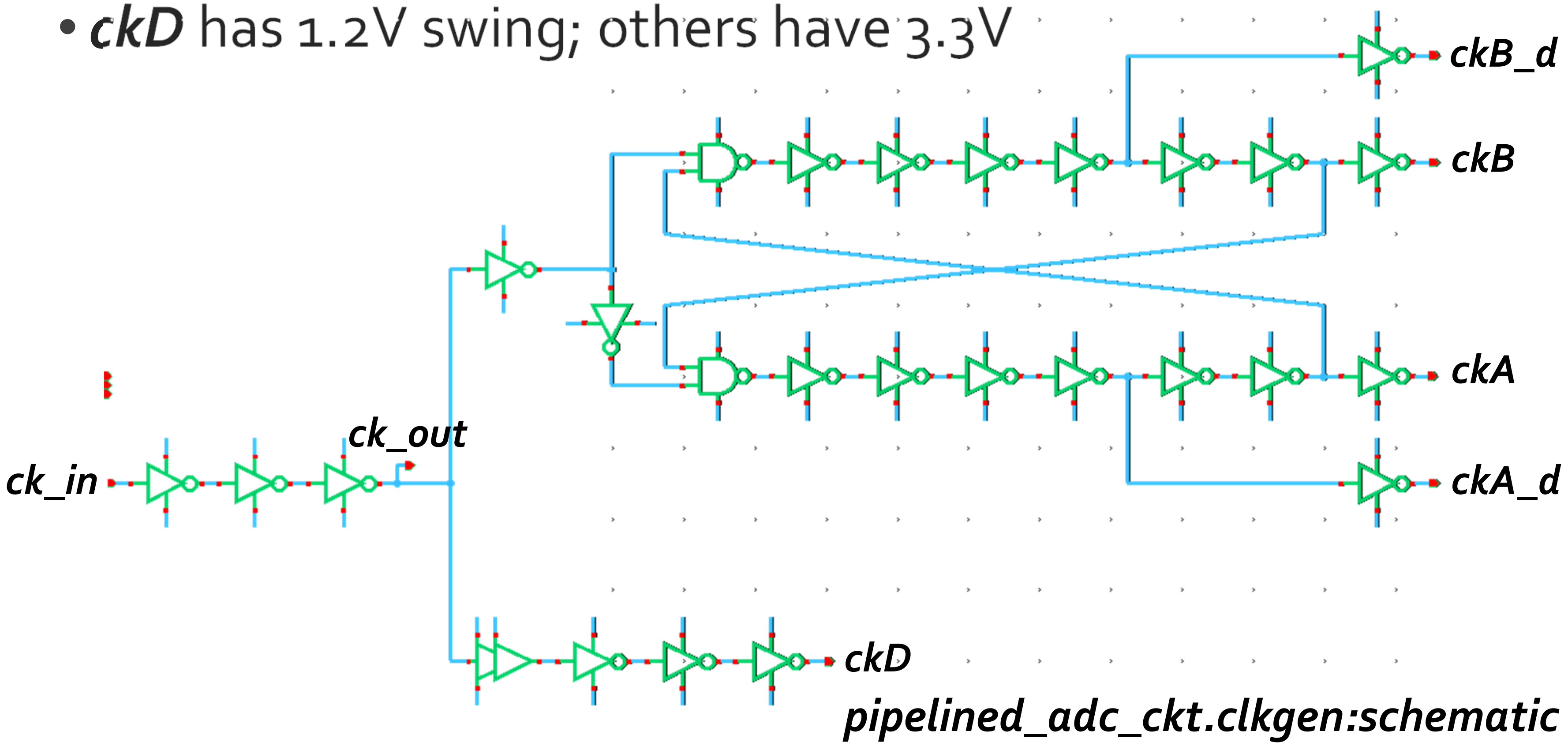
- Applies an offset between ***vbcm_in*** and ***vbcm_out*** which is equal to the difference between the ***outp/outn*** common-mode level and ***vcm***



pipelined_adc_ckt.diffOTA_CMFB:schematic

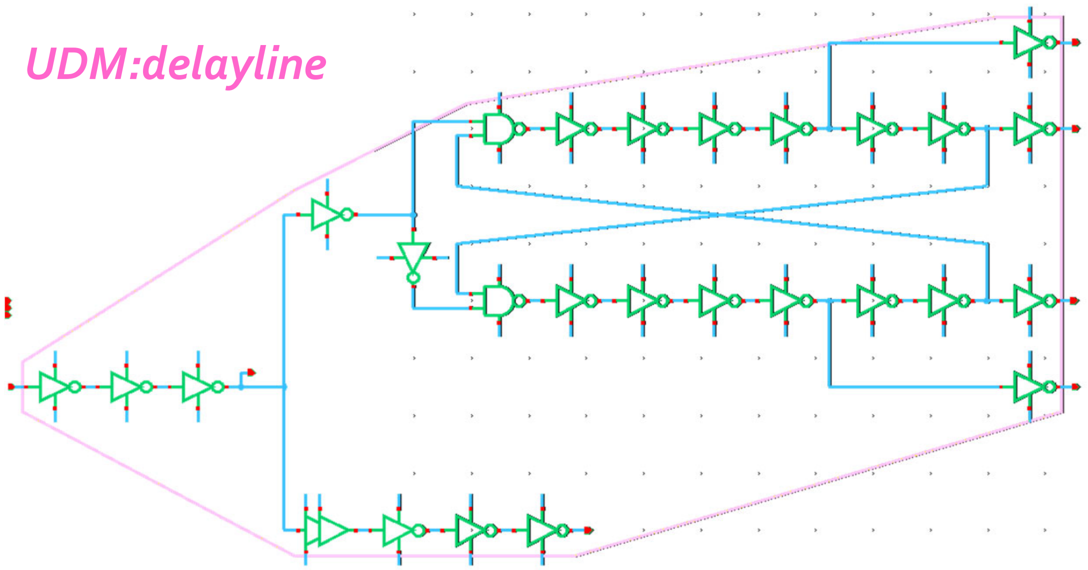
Clock Generator: *clkgen*

- Inverter chains drive the clock loads and add delays defining the non-overlapping periods
  - *ckD* has 1.2V swing; others have 3.3V

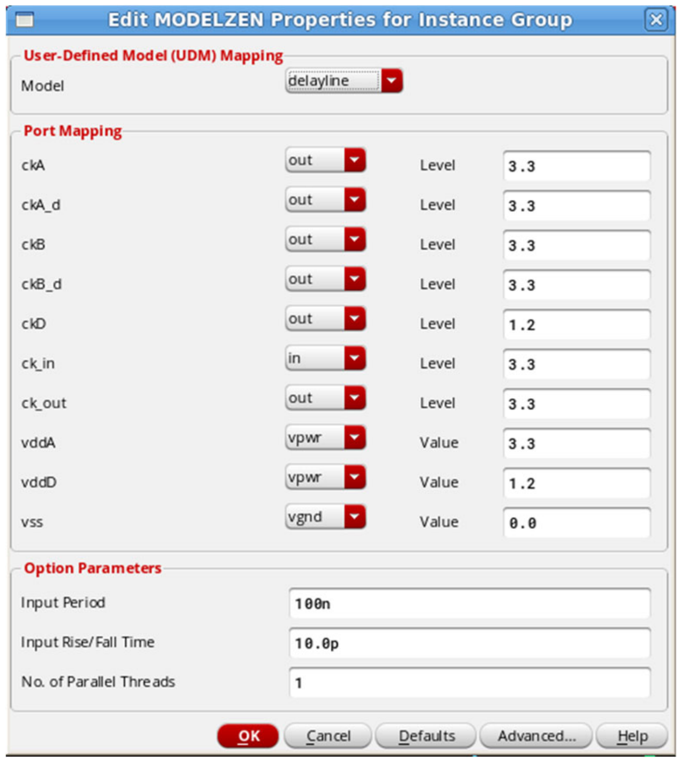


UDM *delayline* for *clkgen*

- Models a delay line with digital input/output's which can be gated or delay-adjusted depending on the digital *mode* inputs



pipelined_adc_sol.clkgen:schematic



User-Defined Model (UDM) Mapping

Model: delayline

Port Mapping

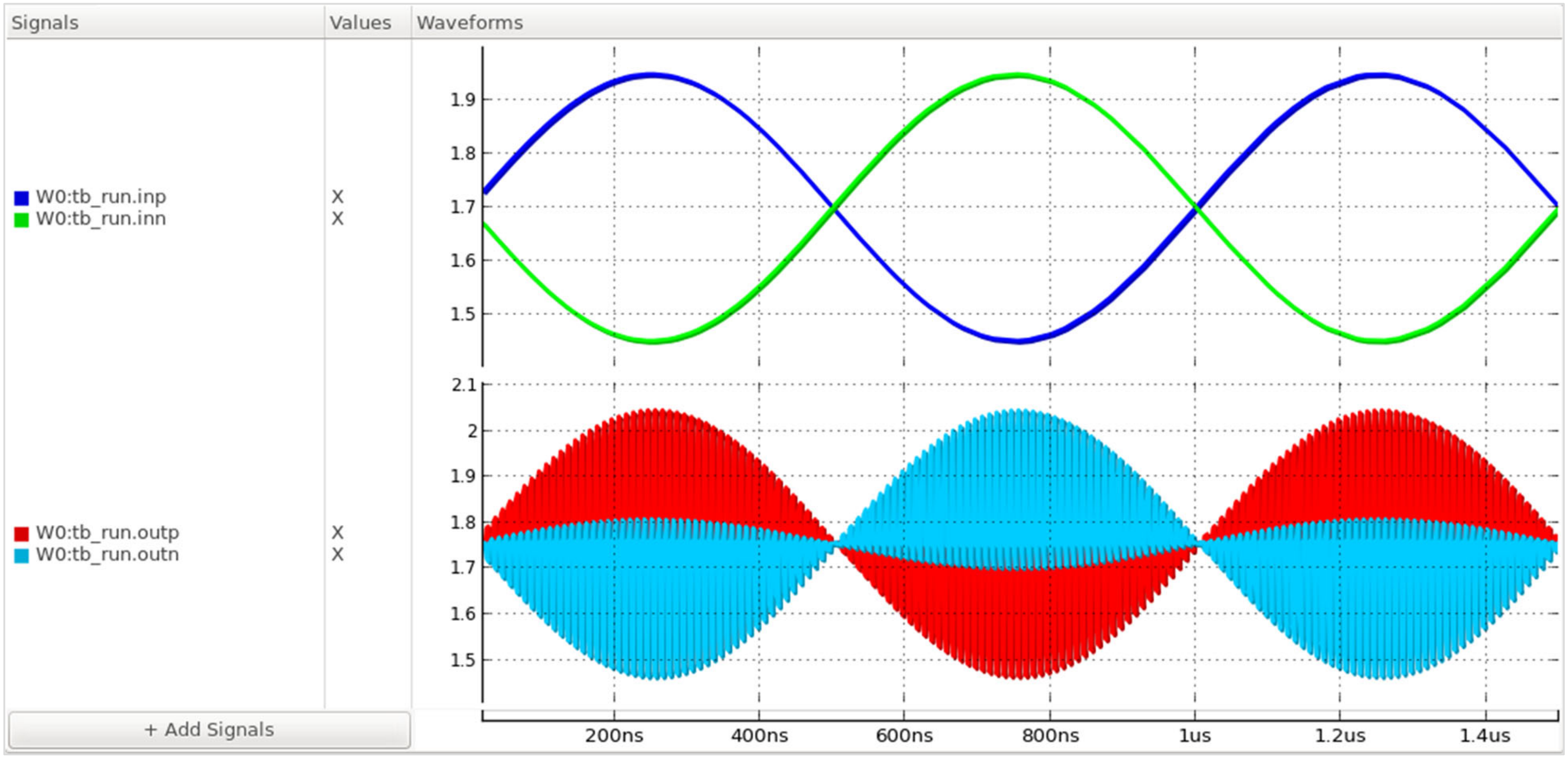
ckA	out	Level	3.3
ckA_d	out	Level	3.3
ckB	out	Level	3.3
ckB_d	out	Level	3.3
ckD	out	Level	1.2
ck_in	in	Level	3.3
ck_out	out	Level	3.3
vddA	vpwr	Value	3.3
vddD	vpwr	Value	1.2
vss	vgnd	Value	0.0

Option Parameters

Input Period	100n
Input Rise/Fall Time	10.0p
No. of Parallel Threads	1

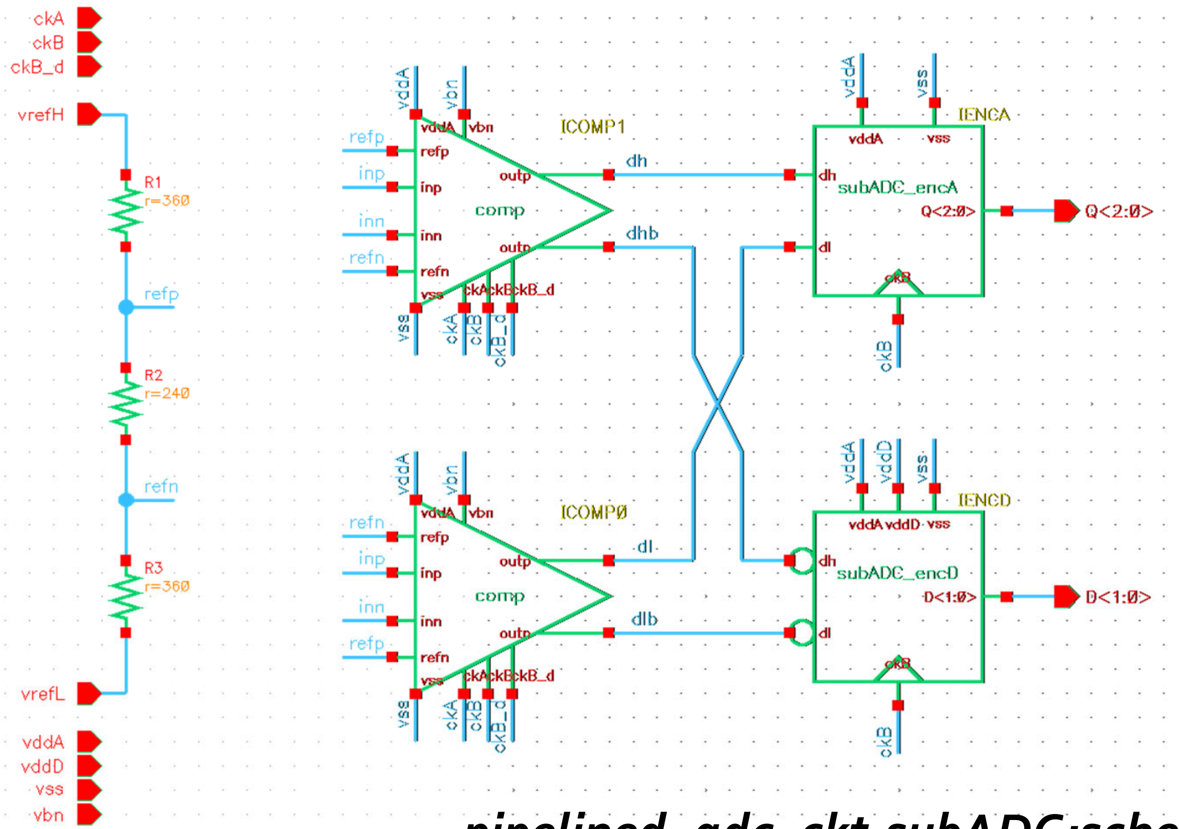
Simulation Results: *sh_amp*

- Now, generate a model for ***sh_amp***
- And run ***pipelined_adc_ckt.sh_amp:tb_run***



1.5-bit Sub-ADC: *subADC*

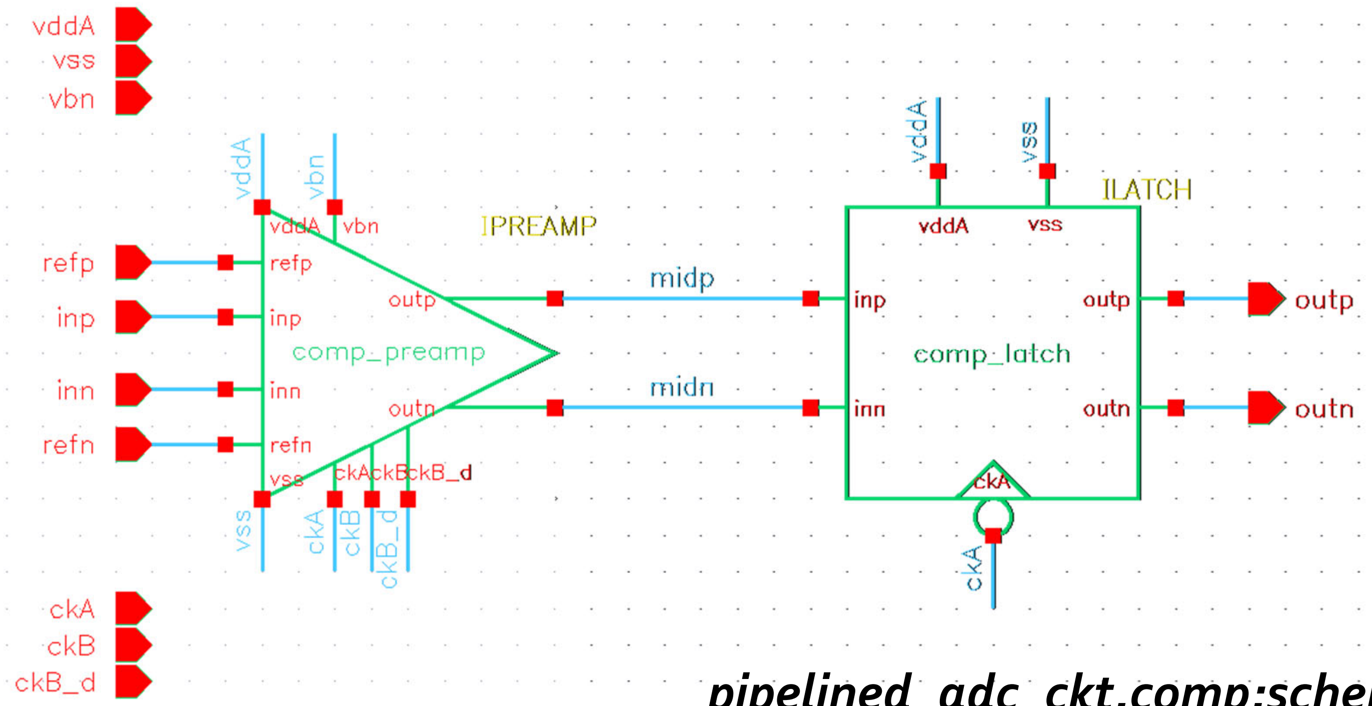
- A flash-type ADC made of R-string reference generator, comparators, and encoder logic



pipelined_adc_ckt.subADC:schematic

Comparator Stage: *comp*

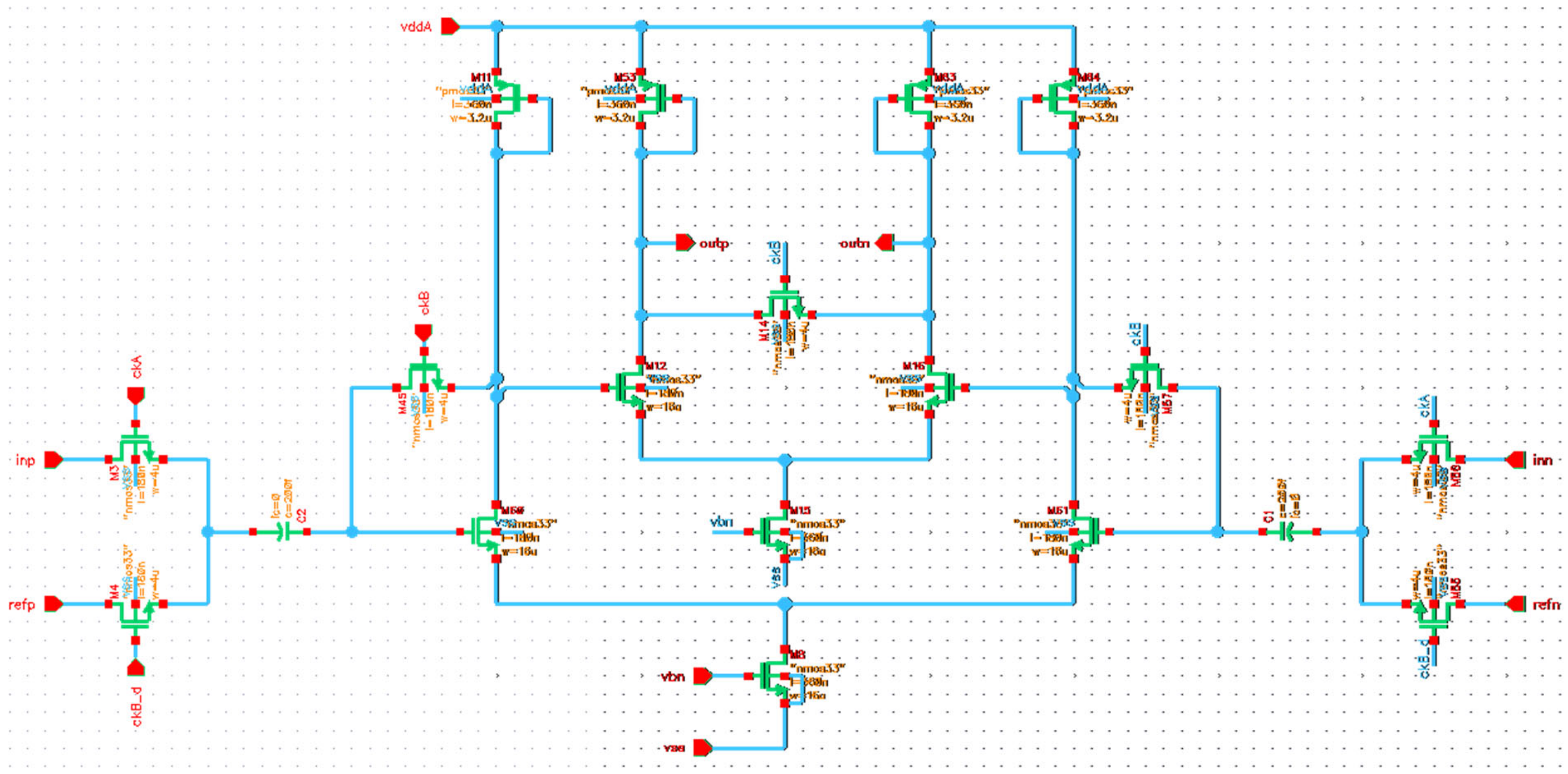
- Made of a pre-amplifier stage amplifying the difference between V_{in}=***inp-inn*** and V_{ref}=***refp-refn*** and a clocked comparator (latch) stage detecting its polarity



pipelined_adc_ckt.comp:schematic

Pre-Amplifier Stage: *comp_preamp*

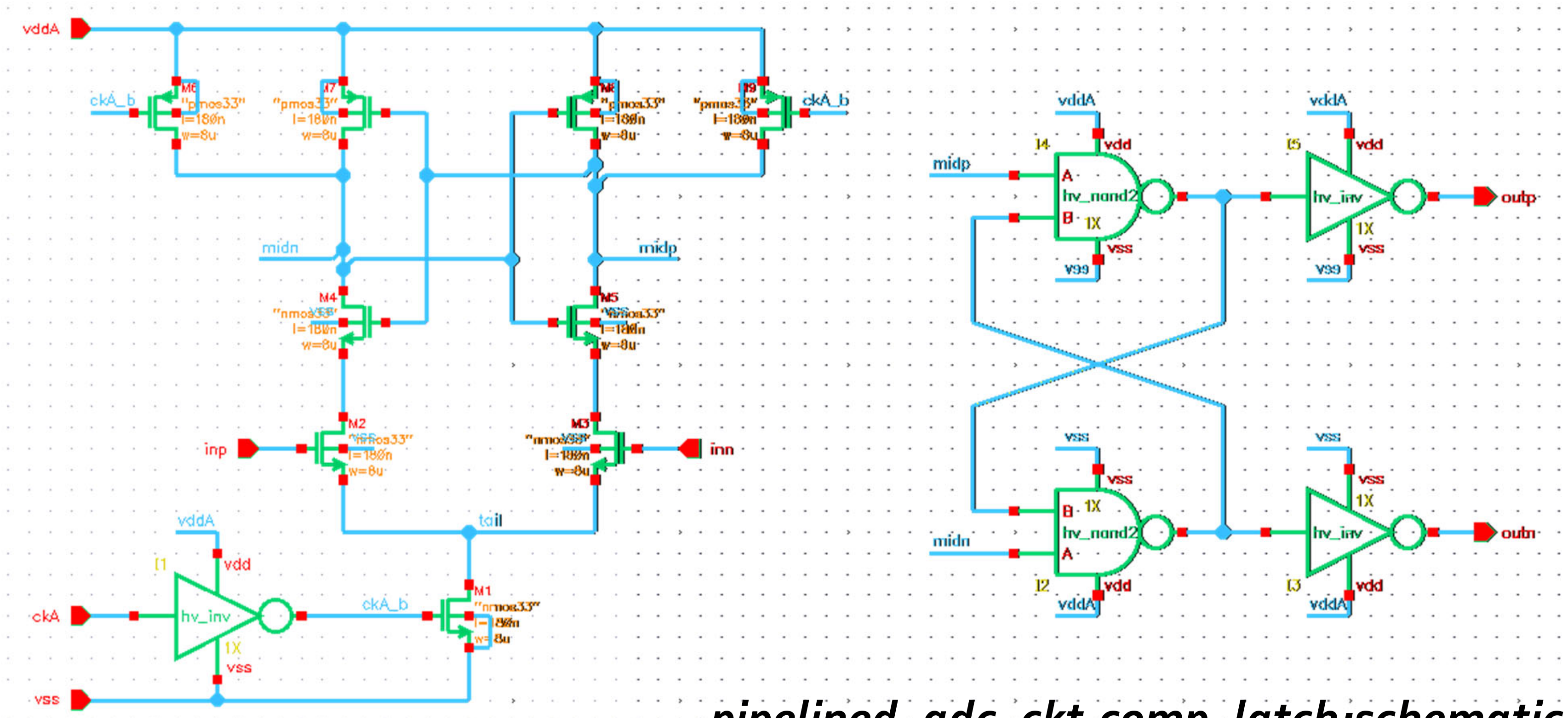
- A switched-capacitor circuit amplifying V_{in}-V_{ref}



pipelined_adc_ckt.comp_preamp:schematic

Latch Stage: *comp_latch*

- A strongArm comparator detecting the polarity of ***inp-inn*** at the falling edge of ***ckA***

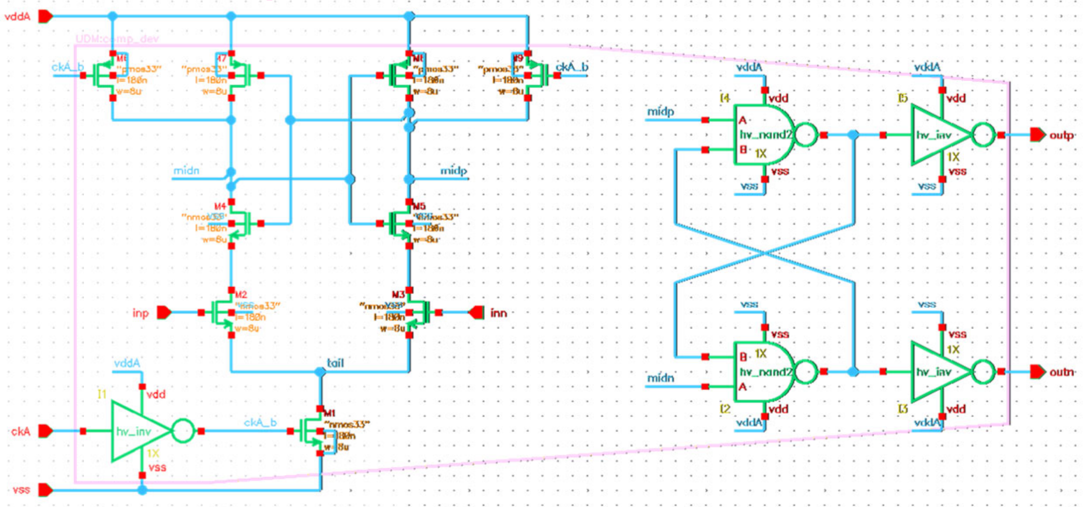


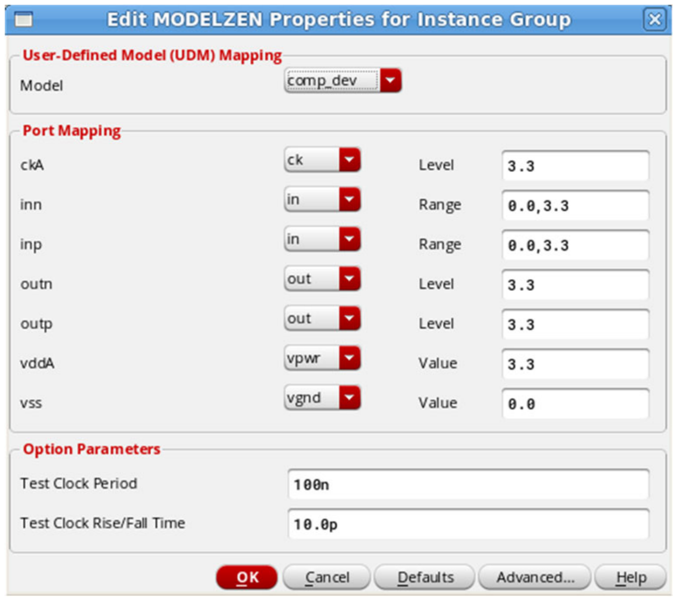
pipelined_adc_ckt.comp_latch:schematic

UDM *comp_dev* for *comp_latch*

- A custom UDM that models a clocked comparator circuit with *xreal*-type analog inputs and *xbit*-type digital outputs

UDM:comp_dev




Edit MODELZEN Properties for Instance Group

User-Defined Model (UDM) Mapping

Model: comp_dev

Port Mapping

ckA	ck	Level	3.3
inn	in	Range	0.0,3.3
inp	in	Range	0.0,3.3
outn	out	Level	3.3
outp	out	Level	3.3
vddA	vpwr	Value	3.3
vss	vgnd	Value	0.0

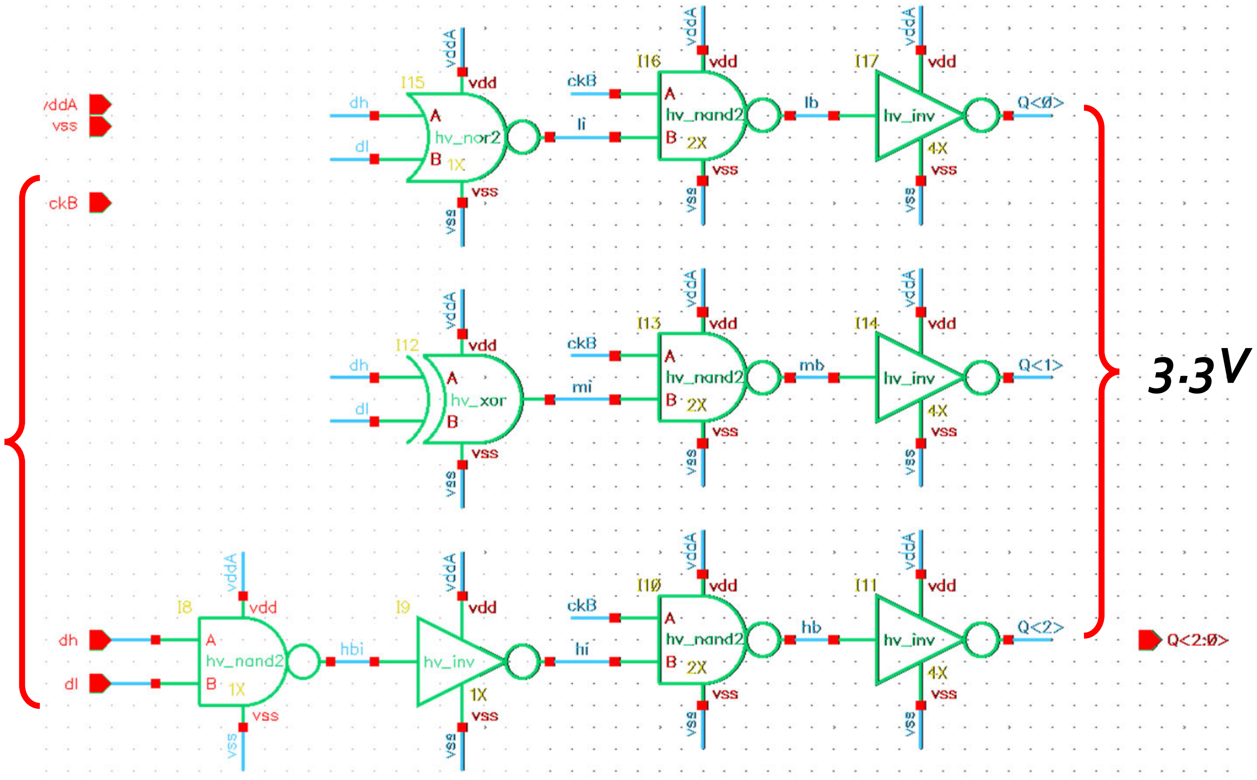
Option Parameters

Test Clock Period	100n
Test Clock Rise/Fall Time	10.0p

Output Encoder Logic: *subADC_encA*

- Converts the thermometer-coded inputs (***dh***, ***dl***) to one-hot-coded outputs ***Q<2:0>***, gated by ***ckB***

3.3V inputs

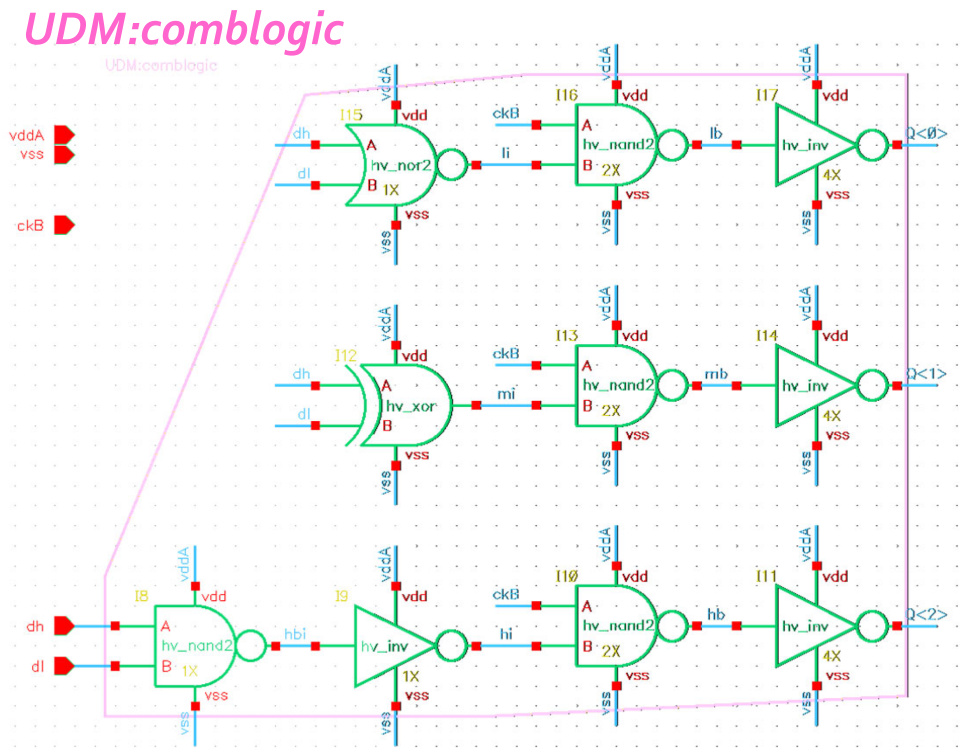


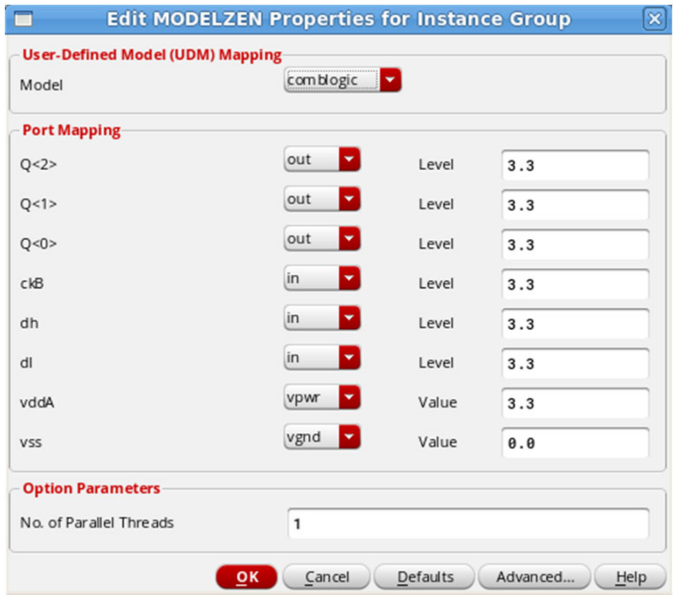
3.3V outputs

pipelined_adc_ckt.subADC_encA:schematic

UDM *comblogic* for *subADC_encA*

- Models arbitrary combinational logic paths




Edit MODELZEN Properties for Instance Group

User-Defined Model (UDM) Mapping

Model: comblogic

Port Mapping

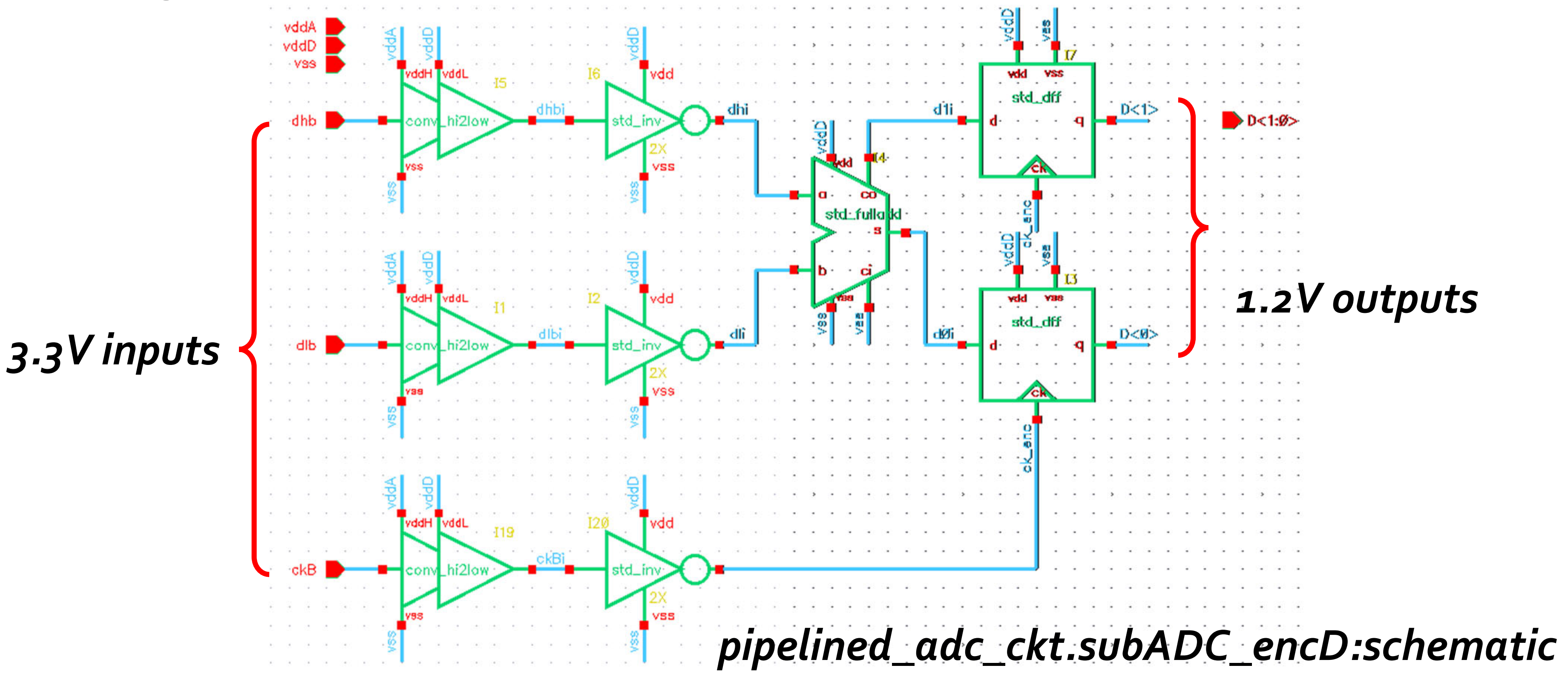
Q<2>	out	Level	3.3
Q<1>	out	Level	3.3
Q<0>	out	Level	3.3
ckB	in	Level	3.3
dh	in	Level	3.3
dl	in	Level	3.3
vddA	vpwr	Value	3.3
vss	vgnd	Value	0.0

Option Parameters

No. of Parallel Threads: 1

Output Encoder Logic: *subADC_encD*

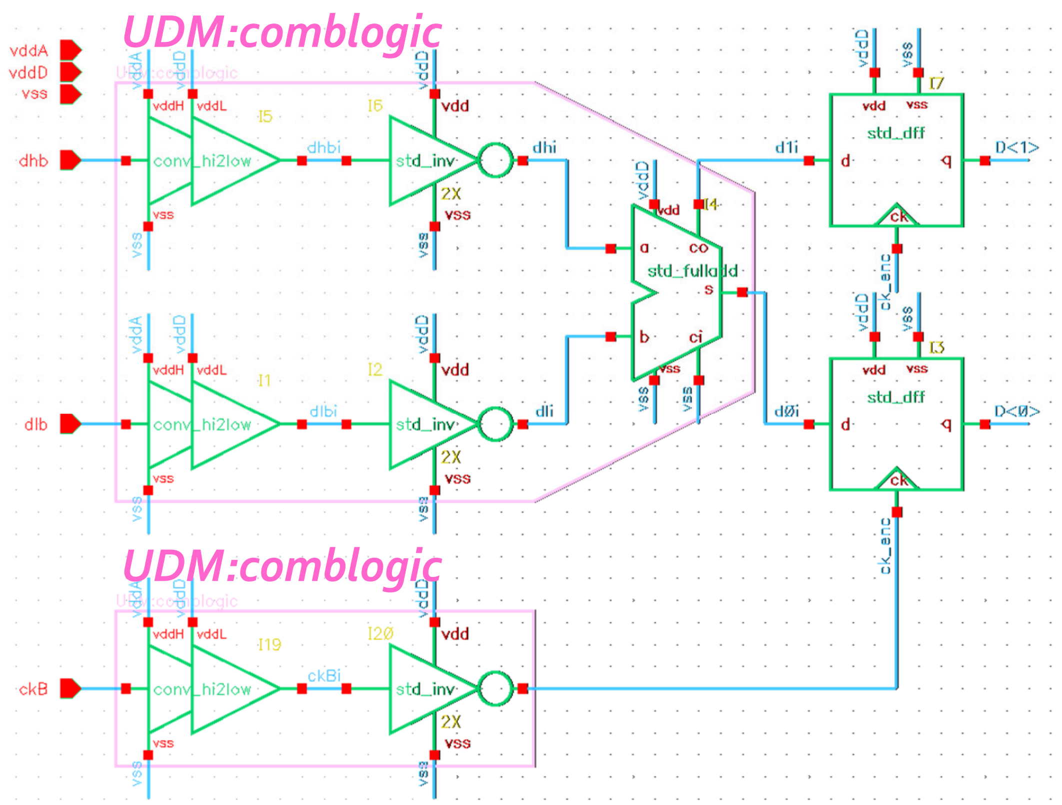
- Converts the thermometer-coded inputs (***dhb***, ***dhl***) to binary-coded outputs ***D<1:0>***, registered at falling edge of ***ckB***

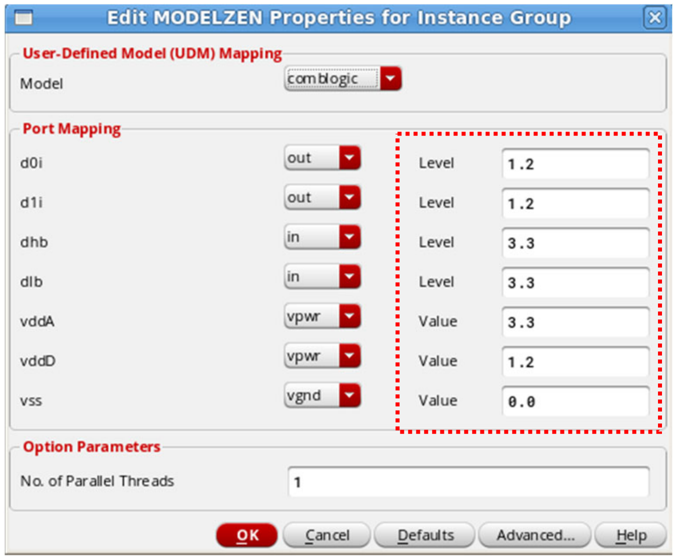


pipelined_adc_ckt.subADC_encD:schematic

UDM *comblogic* for *subADC_encD*

- Can also model level converters by setting input/output levels




Edit MODELZEN Properties for Instance Group

User-Defined Model (UDM) Mapping

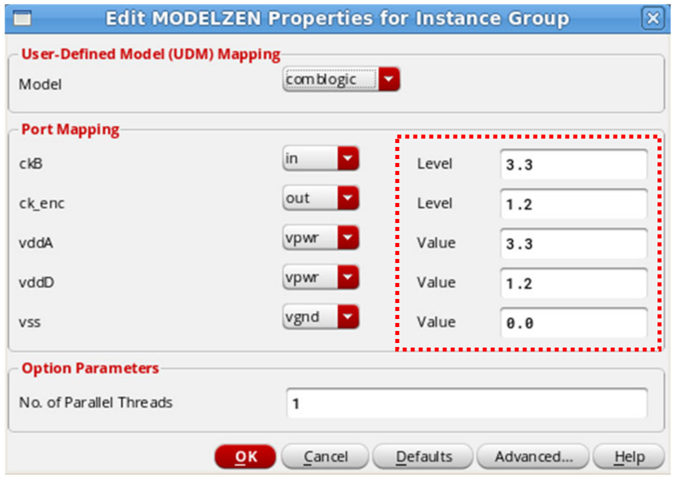
Model: comblogic

Port Mapping

d0i	out	Level	1.2
d1i	out	Level	1.2
dhb	in	Level	3.3
dlb	in	Level	3.3
vddA	vpwr	Value	3.3
vddD	vpwr	Value	1.2
vss	vgnd	Value	0.0

Option Parameters

No. of Parallel Threads: 1


Edit MODELZEN Properties for Instance Group

User-Defined Model (UDM) Mapping

Model: comblogic

Port Mapping

ckB	in	Level	3.3
ck_enc	out	Level	1.2
vddA	vpwr	Value	3.3
vddD	vpwr	Value	1.2
vss	vgnd	Value	0.0

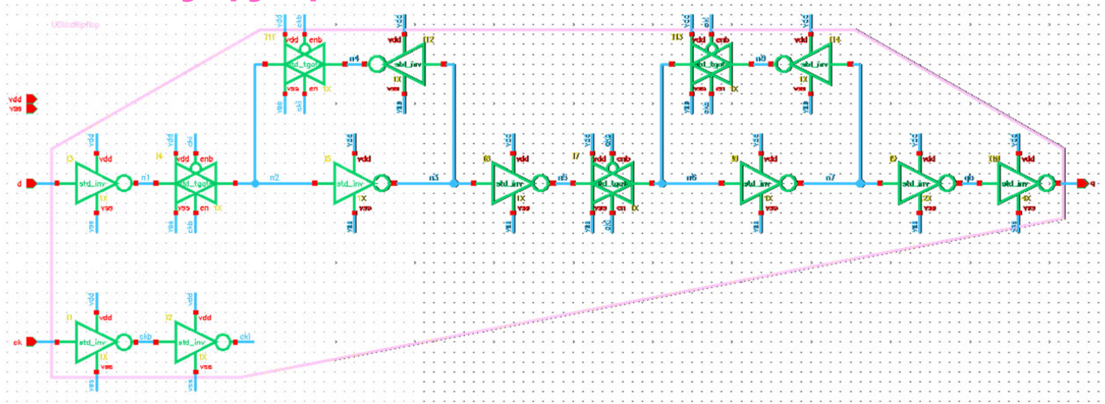
Option Parameters

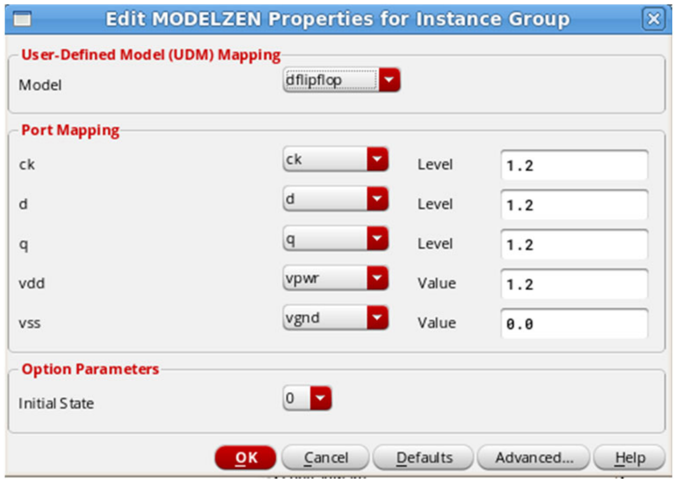
No. of Parallel Threads: 1

UDM *dflipflop* for *std_dff*

- Models D-flipflops with optional set/reset inputs
  - Auto-detects clock, set/reset & output polarities, and sync/async types, …

UDM:dflipflop




Edit MODELZEN Properties for Instance Group

User-Defined Model (UDM) Mapping

Model: dflipflop

Port Mapping

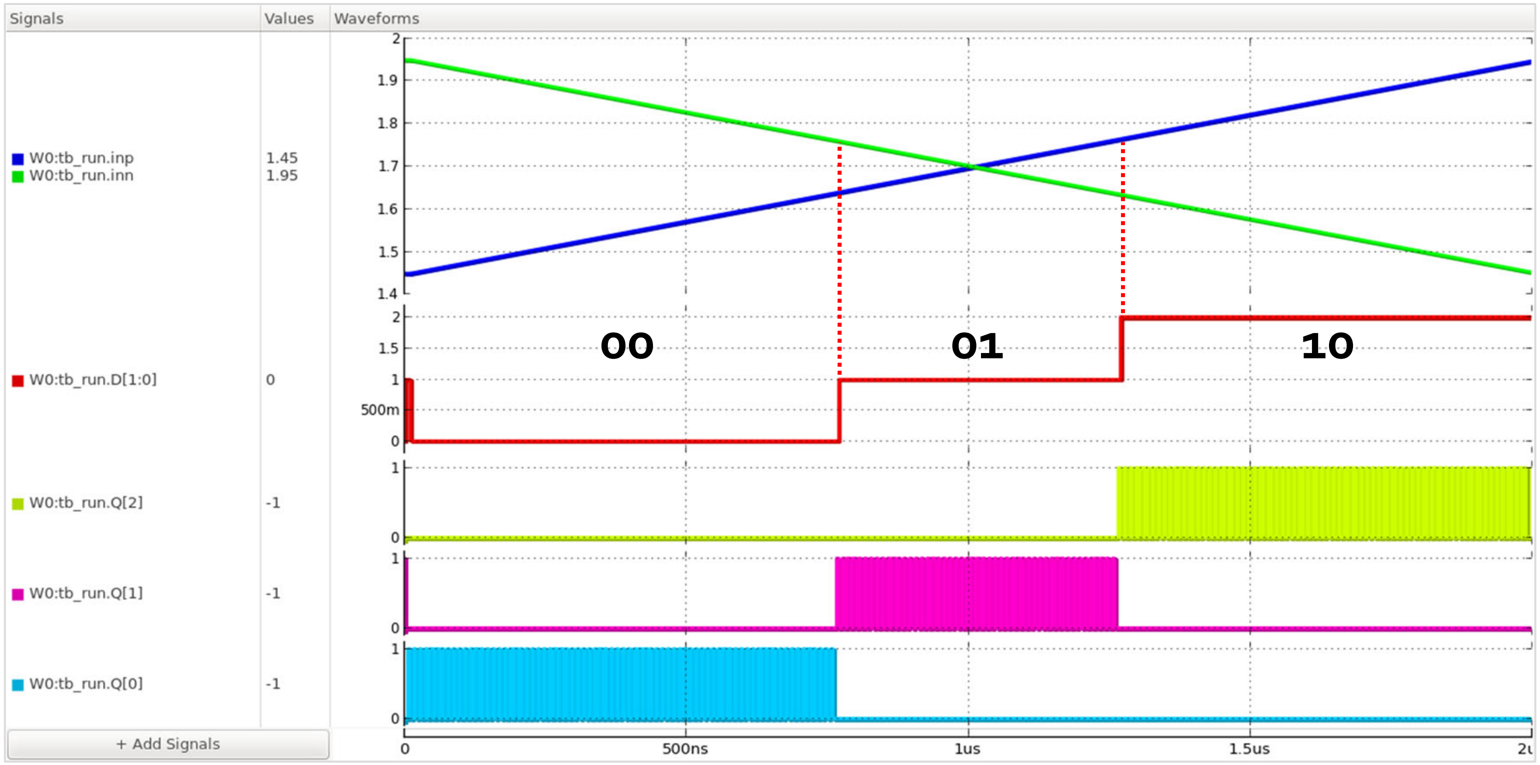
ck	ck	Level	1.2
d	d	Level	1.2
q	q	Level	1.2
vdd	vpwr	Value	1.2
vss	vgnd	Value	0.0

Option Parameters

Initial State: 0

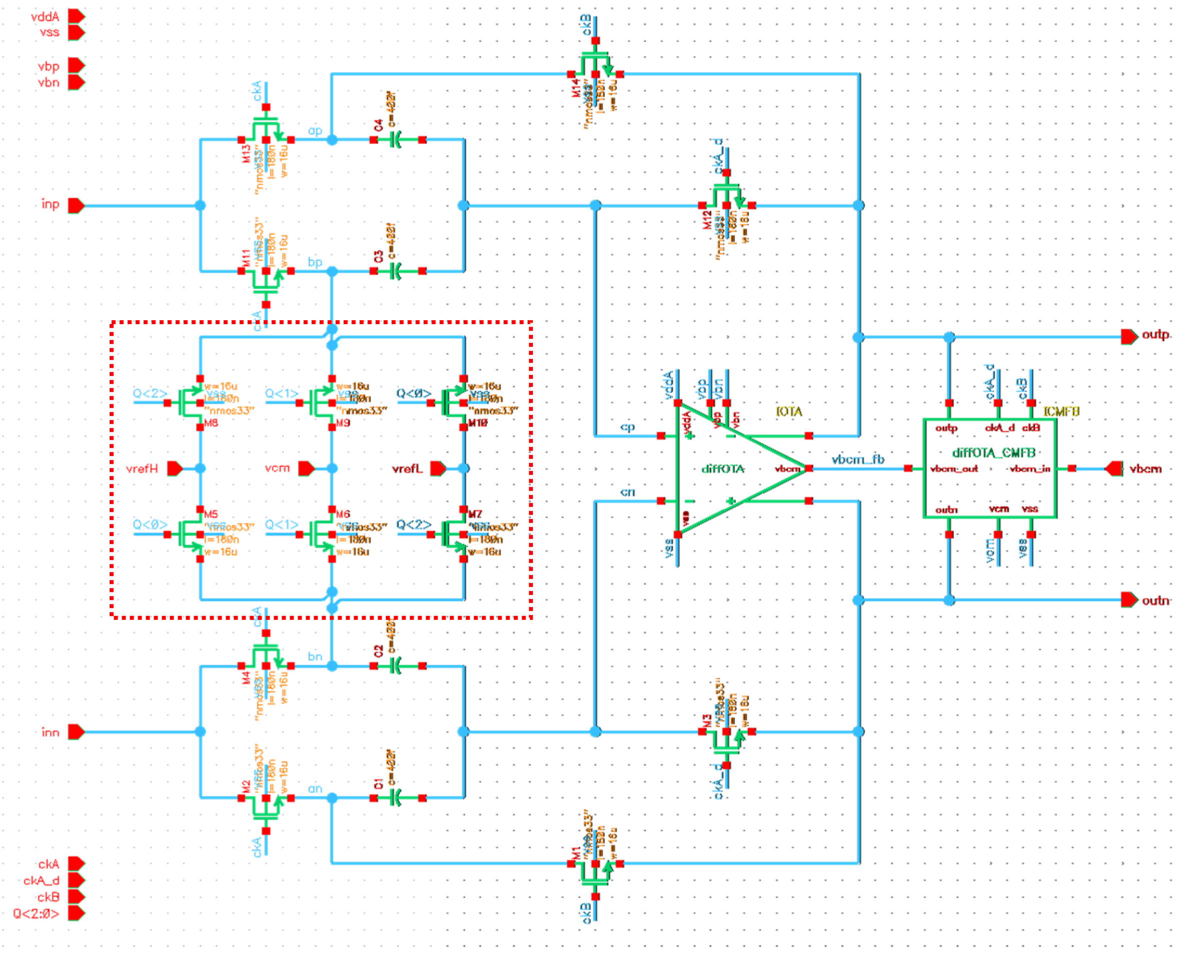
Simulation Results: subADC

- Testbench: ***pipelined_adc_ckt.subADC:tb_run***
- Sub-ADC produces both binary-coded output ***D<1:0>*** and one-hot-coded output ***Q<2:0>***



1.5-bit MDAC: *MDAC*

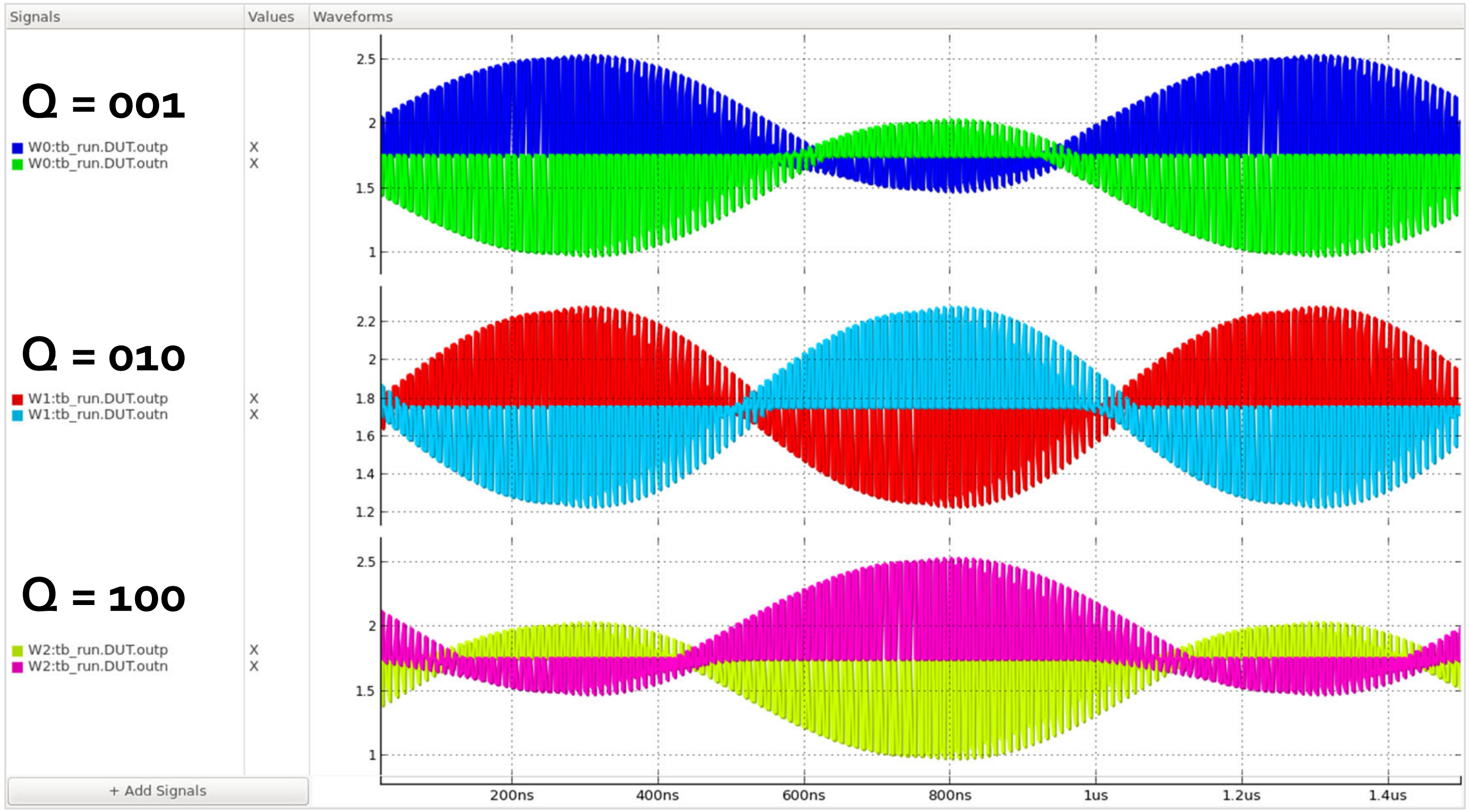
- Sub-DAC selects ***vrefH***, ***vcm***, or ***vrefL*** depending on one-hot coded ***Q<2:0>***
- **Q<2:0>** is qualified by ***ckB***



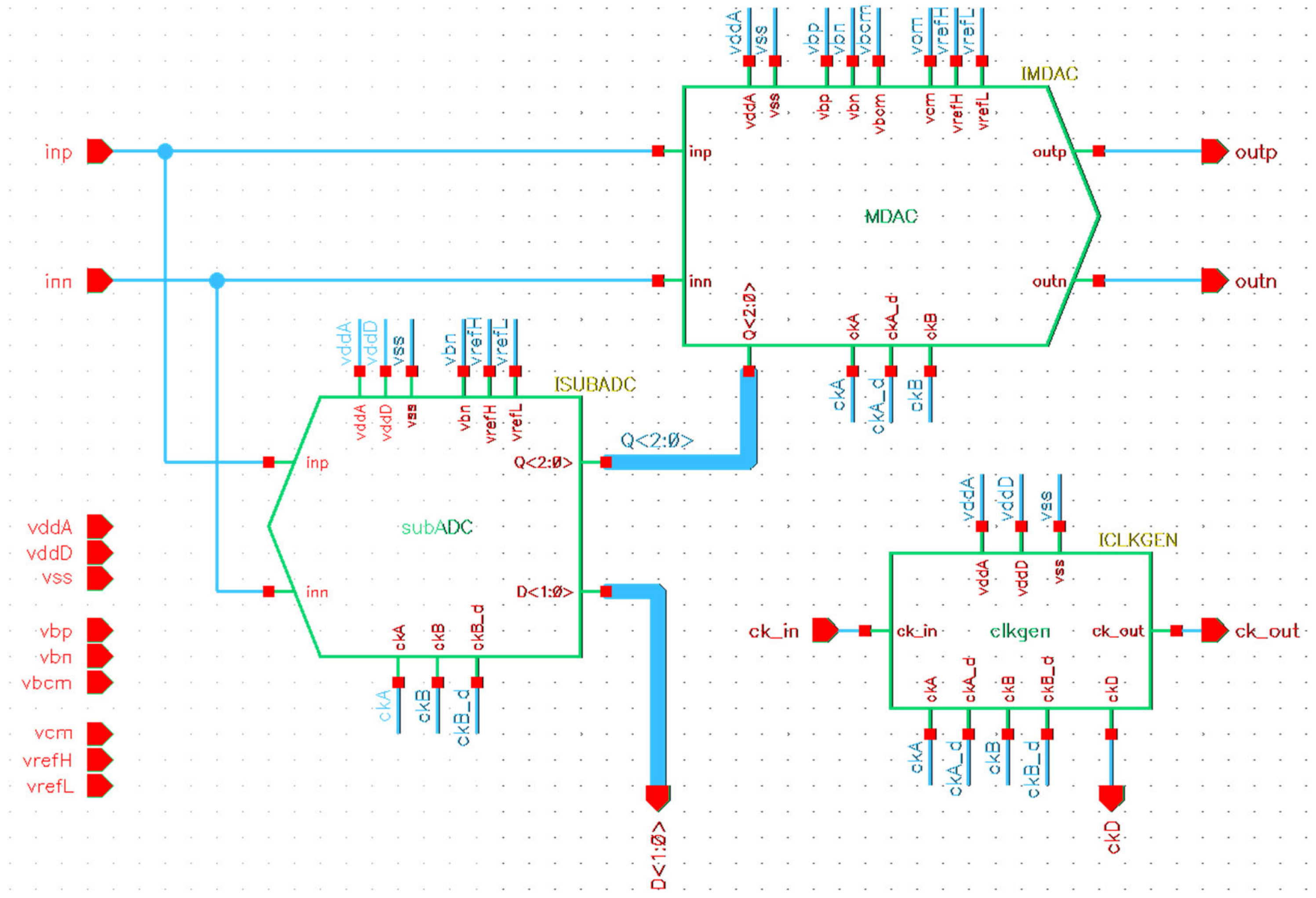
pipelined_adc_ckt.MDAC:schematic

Simulation Results: *MDAC*

- Testbench: ***pipelined_adc_ckt.MDAC:tb_run***
- Confirms gain = 2 and offset = -V_{FS}/2, 0, and +V_{FS}/2



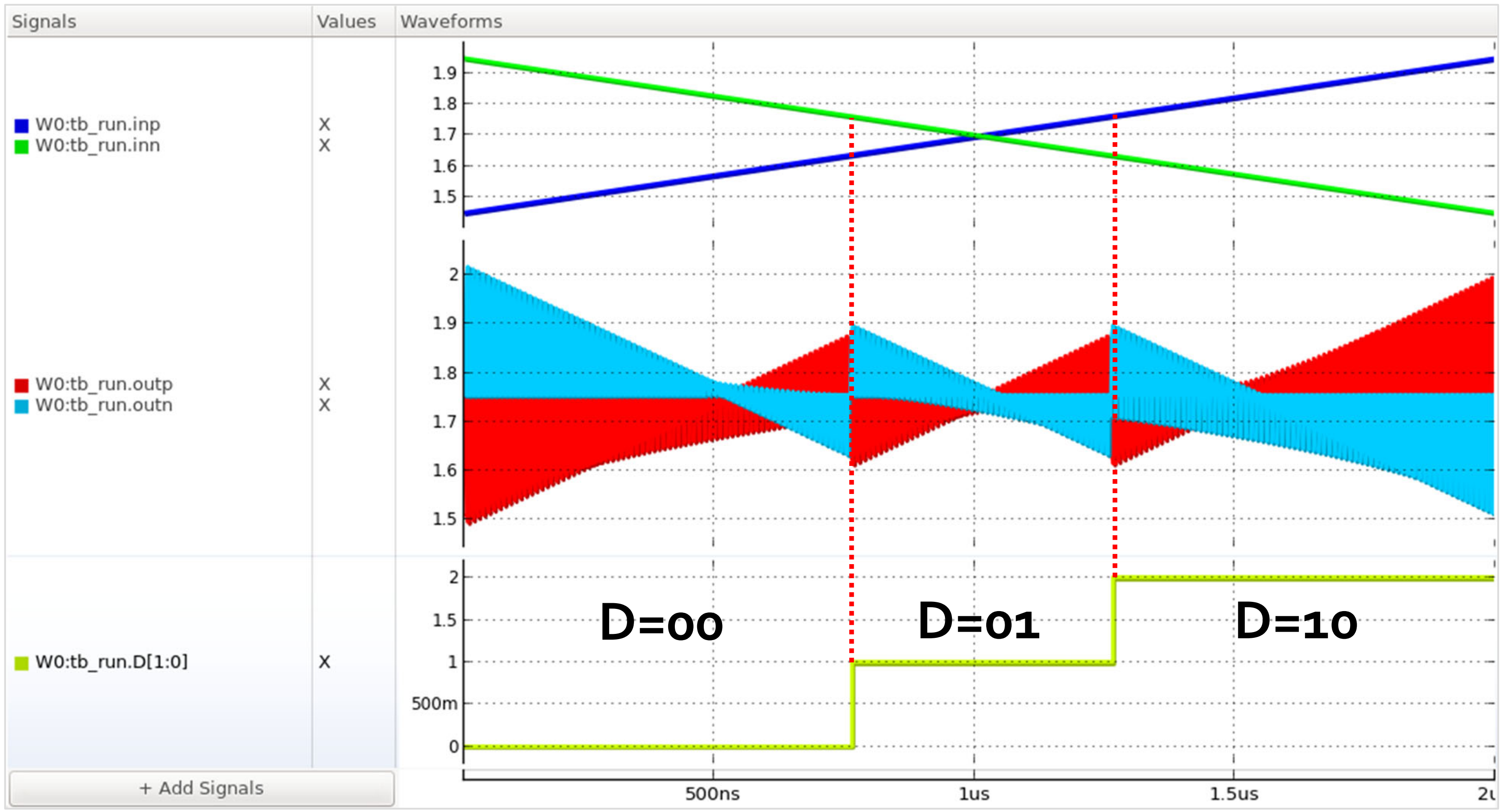
1.5-bit Unit Stage: *ADC_unitstage*



pipelined_adc_ckt.ADC_unitstage:schematic

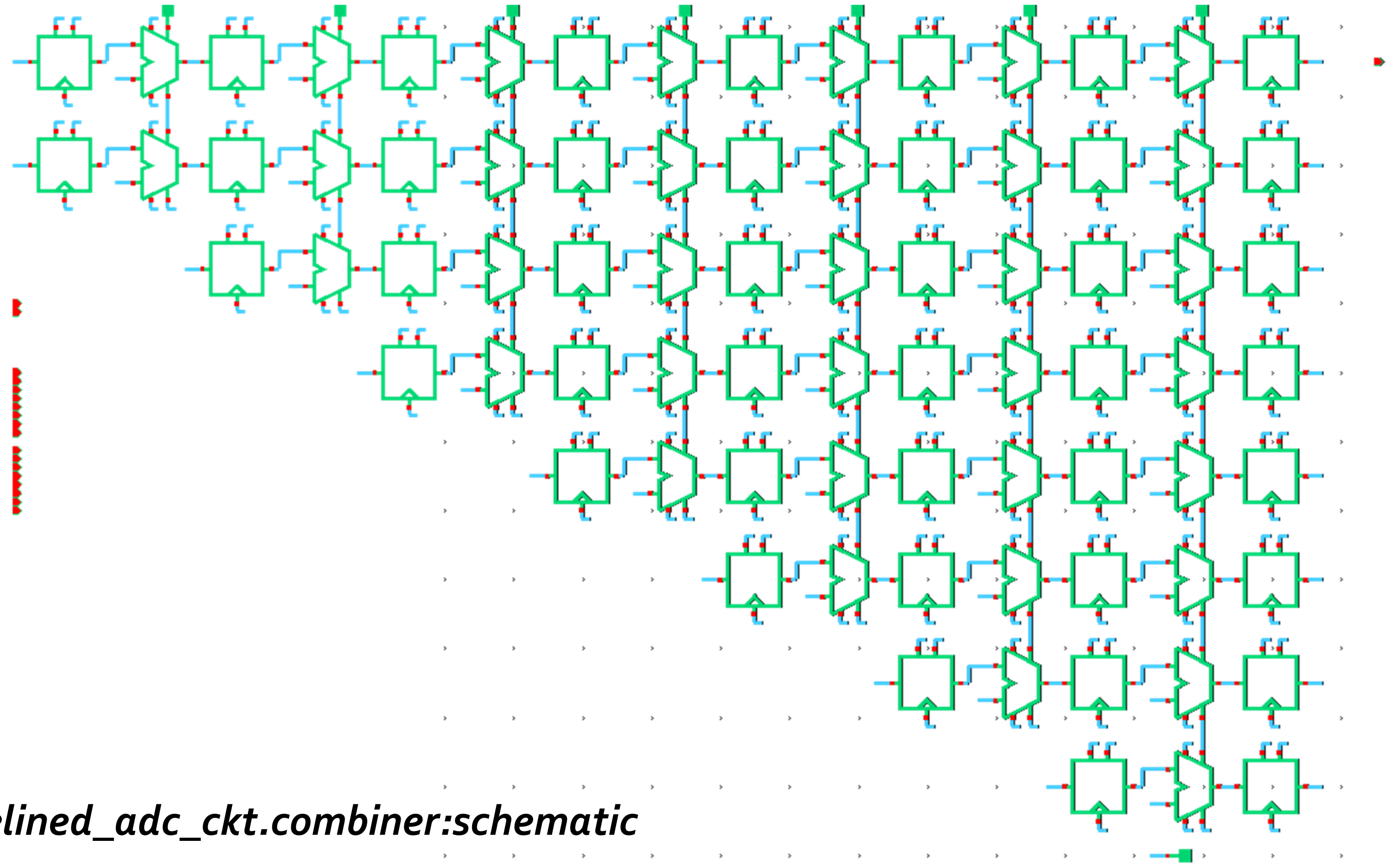
Simulation Results: *ADC_unitstage*

- Testbench: ***pipelined_adc_ckt.ADC_unitstage:tb_run***



Digital Combiner: *combiner*

- A cascade of shift-and-add stages



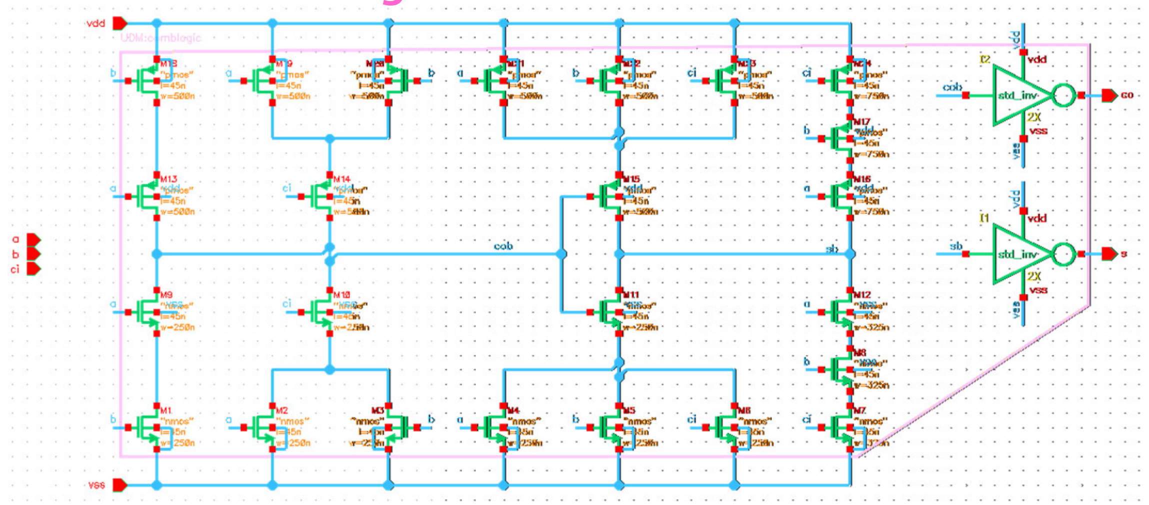
pipelined_adc_ckt.combiner:schematic

UDM *comblogic* for *std_fulladd*

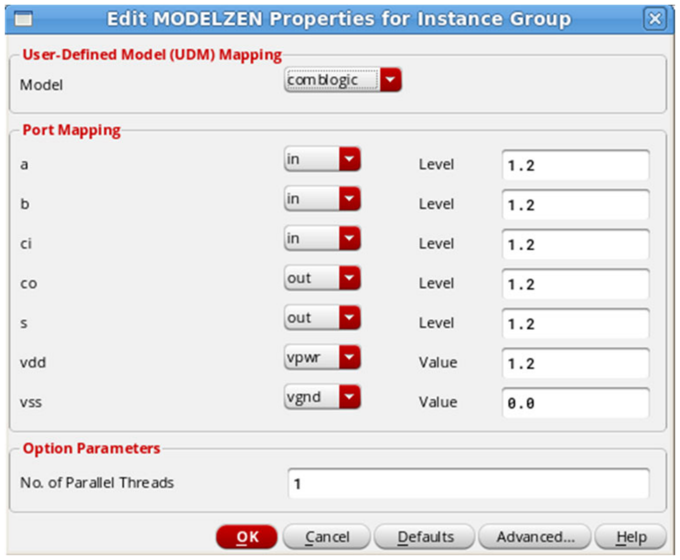
- Static CMOS logic computing:

$$S = A \oplus B \oplus C_i \qquad C_o = AB + BC_i + C_iA$$

UDM:comblogic



pipelined_adc_sol.std_fulladd:schematic



Edit MODELZEN Properties for Instance Group

User-Defined Model (UDM) Mapping

Model	comblogic

Port Mapping

a	in	Level	1.2
b	in	Level	1.2
ci	in	Level	1.2
co	out	Level	1.2
s	out	Level	1.2
vdd	vpwr	Value	1.2
vss	vgnd	Value	0.0

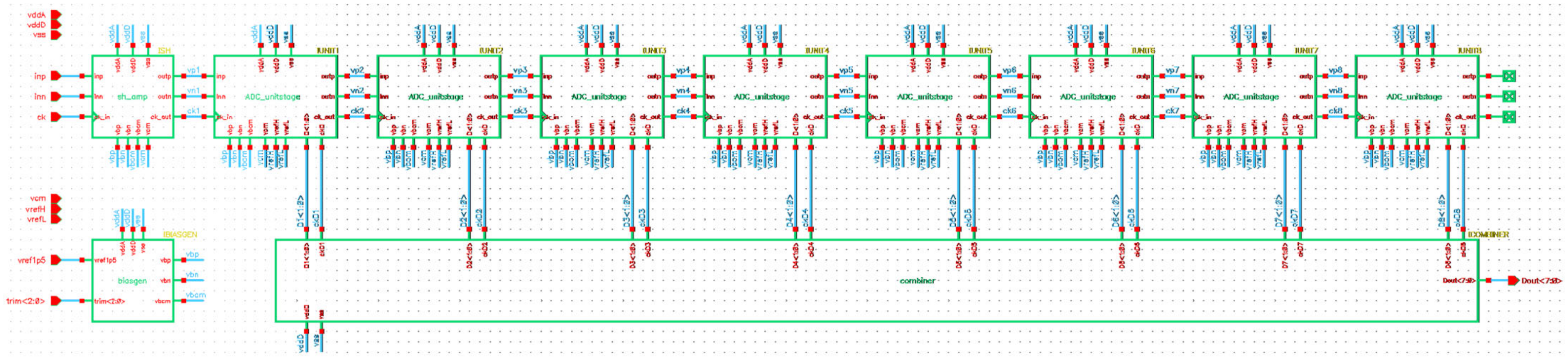
Option Parameters

No. of Parallel Threads	1

Pipelined ADC: *ADC_top*

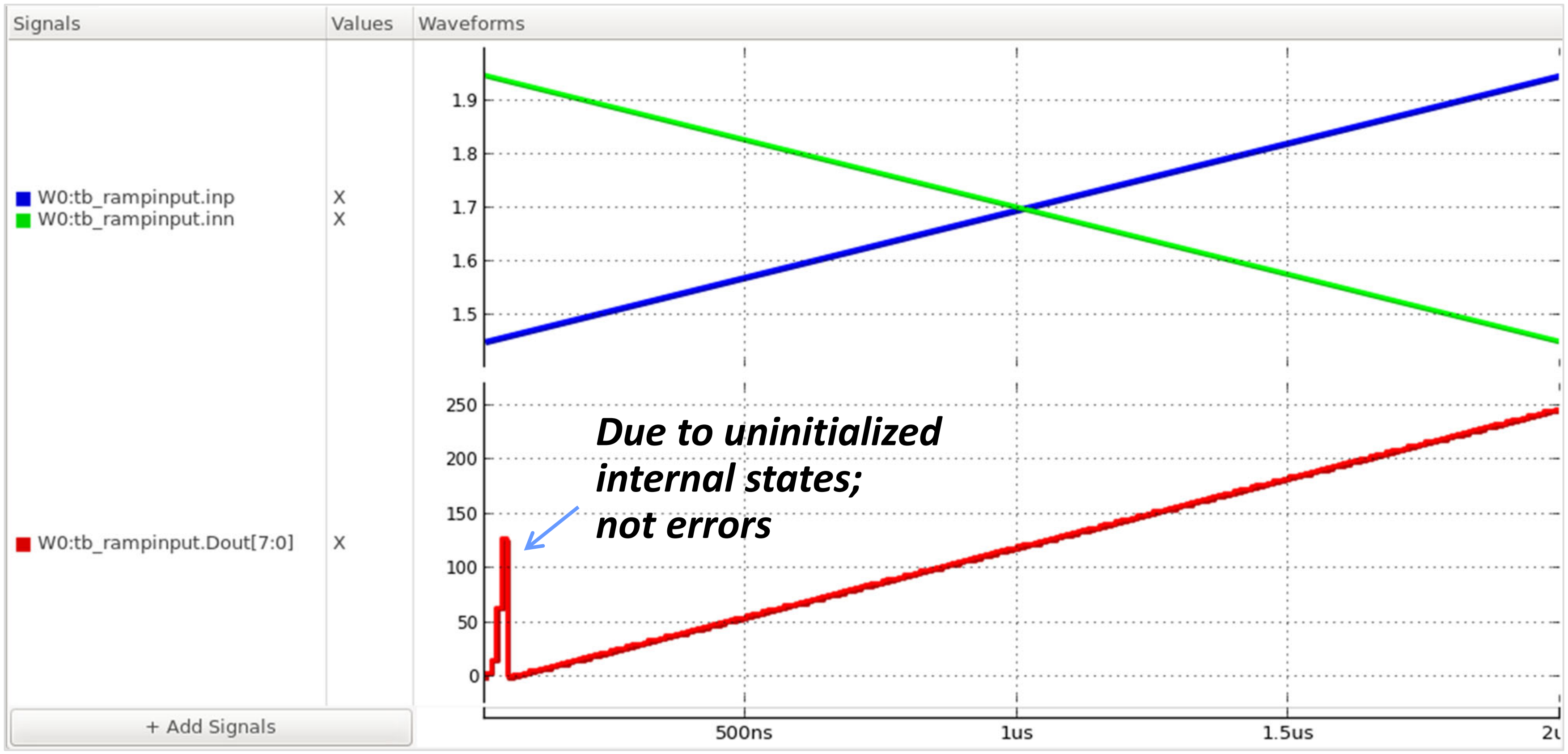
- Now we are ready to extract the top-level model of the pipelined ADC!

pipelined_adc_ckt.ADC_top:schematic:



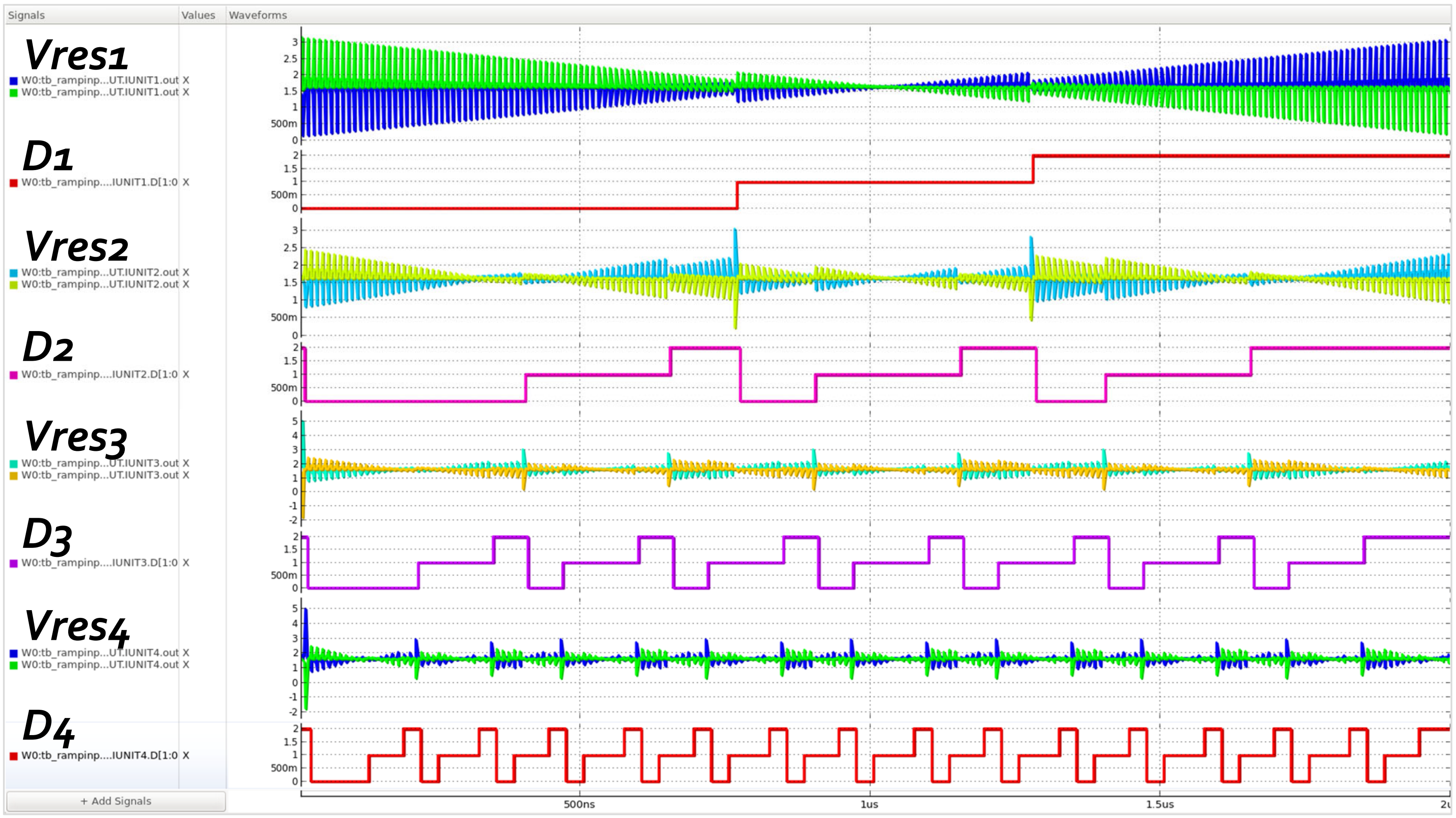
Simulated Results: *tb_rampinput*

- ***pipelined_adc_ckt.ADC_top:tb_rampinput***
- Simulation runtime: 60 sec. (vs. 18.7 min. with CLM)



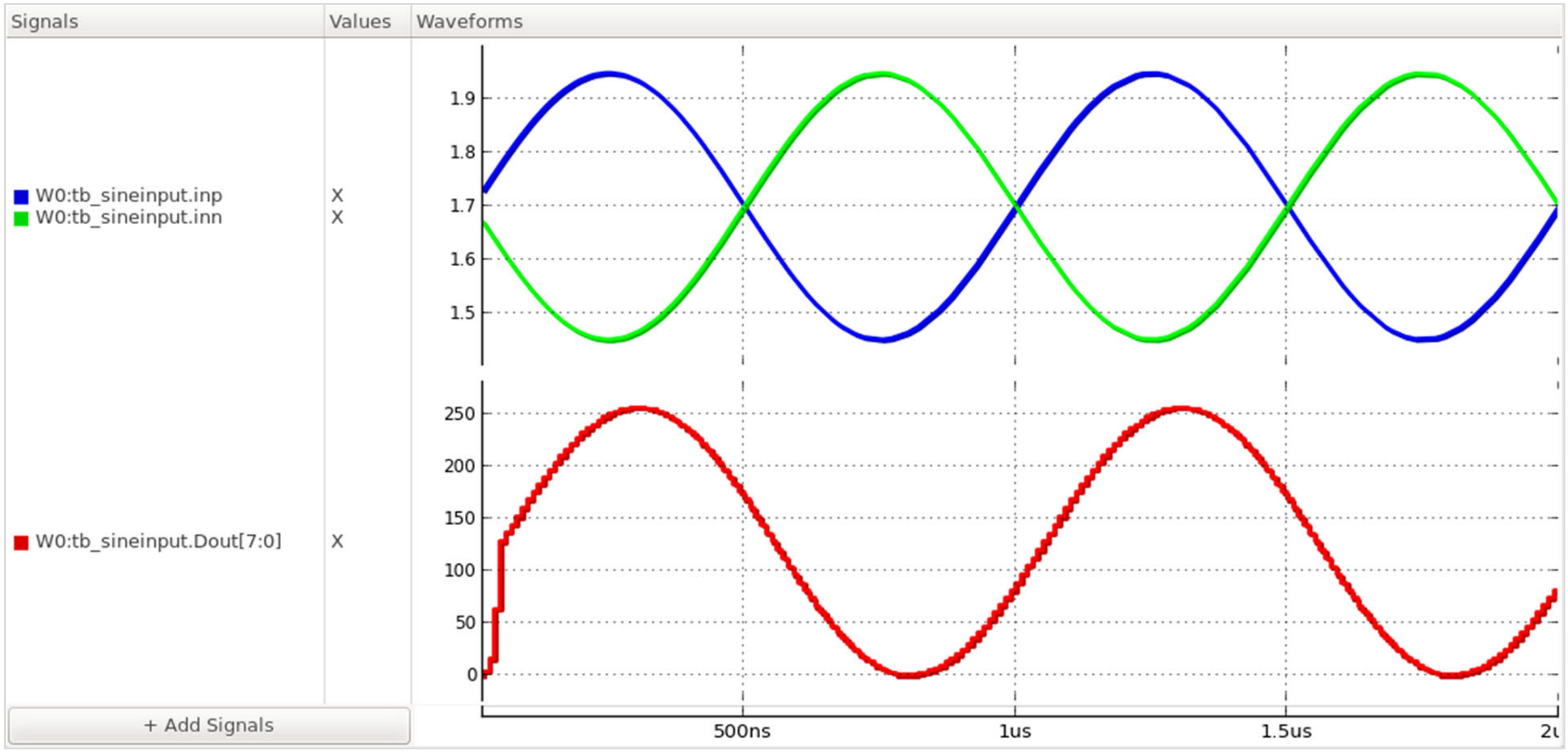
Simulated Results: *tb_rampinput* (2)

- Digital and residual outputs of the individual stages:


Signals
Values
Waveforms
Vres1
W0:tb_rampinp...UT.IUNIT1.out X
W0:tb_rampinp...UT.IUNIT1.out X
D1
W0:tb_rampinp....IUNIT1.D[1:0 X
Vres2
W0:tb_rampinp...UT.IUNIT2.out X
W0:tb_rampinp...UT.IUNIT2.out X
D2
W0:tb_rampinp....IUNIT2.D[1:0 X
Vres3
W0:tb_rampinp...UT.IUNIT3.out X
W0:tb_rampinp...UT.IUNIT3.out X
D3
W0:tb_rampinp....IUNIT3.D[1:0 X
Vres4
W0:tb_rampinp...UT.IUNIT4.out X
W0:tb_rampinp...UT.IUNIT4.out X
D4
W0:tb_rampinp....IUNIT4.D[1:0 X
+ Add Signals
500ns
1us
1.5us

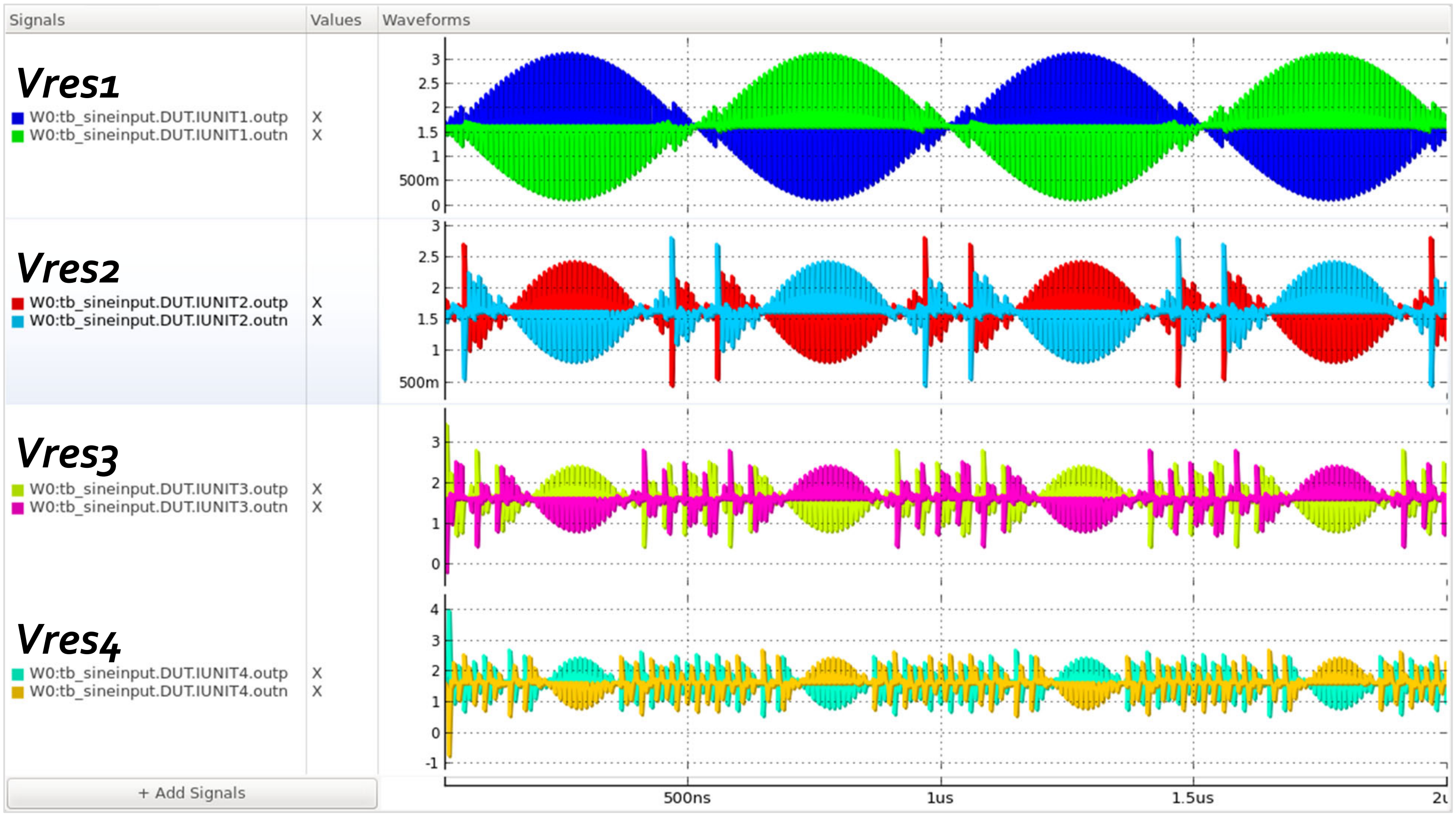
Simulated Results: *tb_sineinput*

- ***pipelined_adc_ckt.ADC_top:tb_sineinput***
- Simulation runtime: 54 sec. (vs. 19.9 min. with CLM)



Simulated Results: *tb_sineinput* (2)

- The residual outputs of the individual stages:



Summary

- Demonstrated two ways of automatically extracting bottom-up models from analog circuits
  - ***Structural modeling*** guarantees correct-by-construction models for all kinds of circuits
  - ***Functional modeling*** yields higher-abstraction models that run much faster
  - Best results can be achieved by combining the two

- Now with the SystemVerilog models for analog circuits, you can perform efficient verification for mixed-signal SoC's all in SystemVerilog/UVM!