وَمَا أُوتِيتُمْ مِنَ الْعِلْمِ إِلَّا قَلِيلًا

Ain Shams University – Faculty of Engineering – EECE Dept. – Integrated Circuits Lab.
Dr. Hesham Omran

Analog Integrated System Design – Cadence Tools
Lab 08 (Mini-Project 01)
SAR ADC

Intended Learning Objectives

1) To be familiar with Verilog-A modeling of mixed-signal and digital blocks.
2) To be familiar with the design and simulation of SAR ADC.
3) To be familiar with the design and simulation of fully-differential SAR ADC.

PART 1: Verilog-A Behavioral Models

1) Create a behavioral Verilog-A model for an inverter using the code below[1]. Create a symbol for the inverter.

```
`include "discipline.h"
`include "constants.h"

module inv_gate(vin, vout);
input vin;
output vout;
electrical vin, vout;
parameter real vth = 1;
parameter real vhigh = 2 from (vth:inf);
parameter real vlow = 0 from (-inf:vth);
parameter real trf = 1n from (0:inf);
parameter real tdel = 2n from [trf/2:inf);

   real vout_val;

   analog begin

        @ (cross(V(vin) - vth, 0) or initial_step)
                vout_val =  (V(vin) > vth) ?  vlow : vhigh;

      V(vout) <+ transition( vout_val, tdel-trf/2, trf, trf);

   end
endmodule
```

2) Create a behavioral Verilog-A model for a DFF with Set/Reset using the code below. Create a symbol for the DFF.

```
`include "constants.h"
`include "discipline.h"

module lab_07_dff_sr (Q, Q_, CLK, D, R, S);
output Q;
electrical Q;
output Q_;
electrical Q_;
input CLK;
electrical CLK;
input D;
electrical D;
```

[1] To set your text editor as gedit use the following command in Virtuoso main window: editor = "gedit"

```
input R;
electrical R;
input S;
electrical S;

// INSTANCE PARAMETERS:
  parameter real vhi = 2;                // voltage [v] for logic high
  parameter real vlo = 0 from (-inf:vhi) ; // voltage [v] for logic low
  parameter real vthresh = 0.5*(vhi+vlo) ; // switch voltage [v]

  parameter real tcplhq = 1n from (0: 1m] ;      // prop delay from clock to Q, low to high
  parameter real tcphlq = 1n from (0: 1m] ;      // prop delay from clock to Q, high to low
  parameter real tcplhq_ = 1n from (0: 1m] ;     // prop delay from clock to Q_, low to high
  parameter real tcphlq_ = 1n from (0: 1m] ;     // prop delay from clock to Q_, high to low
  parameter real tspq = 1n from (0: 1m] ;        // prop delay from set to Q and Q_
  parameter real trpq = 1n from (0: 1m] ;        // prop delay from reset to Q and Q_
  parameter real tr = tcplhq from (0:2*tcplhq]; // rise time [s] for gate output
  parameter real tf = tcphlq from (0:2*tcphlq]; // fall time [s] for gate output
  parameter integer initval = -1 from [-1:1] ; // initial value for Q (X,0,1)

// LOCAL VARIABLES:
  real tdelq;
  real tdelq_;
  integer s;
  integer r;
  integer d;
  integer qnow;
  integer q_now;
  integer q;
  integer q_;
  integer sr_flag;
  integer clk_flag;

  analog function real tdelay;
     input outnow, out, tplh, tphl, trise, tfall;
     real  tplh, tphl, trise, tfall;
     integer outnow, out;
         case (1)
            (outnow > out) : begin  // high to low on Q
              tdelay = tphl - tf/2.0 ;
              if (tdelay < 0.05*tf) tdelay = 0; // delay too small, neglect.
            end
            (outnow < out) : begin // low to high on Q
              tdelay = tplh - tr/2.0 ;
              if (tdelay < 0.05*tr) tdelay = 0; // delay too small, neglect.
            end
            (outnow == out) :if (tdelay < 0.05*tr) tdelay = 0; // no change
         endcase
  endfunction

  analog begin
     @ (initial_step) begin
        if (vthresh > vhi || vthresh < vlo) begin
                $display
        ("%M: Inconsistent input threshold specification w/logic family.\n");
        end
        case (initval)
           1: begin q = 1; q_ = 0; end
           0: begin q = 0; q_ = 1; end
           -1: begin q = (V(D) > vthresh); q_ = (V(D) <= vthresh); end
           default begin q = (V(D) > vthresh); q_ = (V(D) <= vthresh); end
        endcase
        sr_flag = 1;
        clk_flag = 0;
     end

      @ (cross(V(S) - vthresh, 0) )      sr_flag = 1;
      @ (cross(V(R) - vthresh, 0) )      sr_flag = 1;
      @ (cross(V(CLK) - vthresh, +1) )   clk_flag = 1;

      if (sr_flag) begin
         sr_flag = 0;
         s = V(S) > vthresh;
         r = V(R) > vthresh;
         d = V(D) > vthresh;
         qnow = q;
         q_now = q_;
```

```
         if ( (s==1) && (r==1) ) begin
                   q = 1;
                   q_ = 1;
                   tdelq = tdelay(qnow, q, tspq, tspq, tr, tf);
                   tdelq_ = tdelay(q_now, q_, trpq, trpq, tr, tf);
                   // assume R/S to Q/Q_ delay time for "high to low" and
                   // "low to high" are the same.
         end
         if ( (s==1) && (r==0) ) begin
                   q = 1;
                   q_ = 0;
                   tdelq = tdelay(qnow, q, tspq, tspq, tr, tf);
                   tdelq_ = tdelay(q_now, q_, trpq, trpq, tr, tf);
         end
         if ( (s==0) && (r==1) ) begin
                   q = 0;
                   q_ = 1;
                   tdelq = tdelay(qnow, q, tspq, tspq, tr, tf);
                   tdelq_ = tdelay(q_now, q_, trpq, trpq, tr, tf);
         end
      end

        if (clk_flag) begin
         clk_flag = 0;
         s = V(S) > vthresh;
         r = V(R) > vthresh;
         d = V(D) > vthresh;
         qnow = q;
         q_now = q_;

         if ( (s==0) && (r==0) ) begin
                   if (d) begin
                      q = 1;
                      q_ = 0;
                      tdelq = tdelay(qnow, q, tcplhq, tcphlq, tr, tf);
                      tdelq_ = tdelay(q_now, q_, tcplhq_, tcphlq_, tr, tf);
                   end
                   if (!d) begin
                      q = 0;
                      q_ = 1;
                      tdelq = tdelay(qnow, q, tcplhq, tcphlq, tr, tf);
                      tdelq_ = tdelay(q_now, q_, tcplhq_, tcphlq_, tr, tf);
                   end
         end
      end

      V(Q) <+ transition( q ? vhi : vlo, tdelq, tr, tf);
      V(Q_)<+ transition( q_? vhi : vlo, tdelq_, tr, tf);
    end
endmodule
```

3) Create a behavioral Verilog-A model for a dynamic comparator using the code below. Create a symbol for the comparator.

```
`include "discipline.h"
`include "constants.h"

// model CompLatched - Comparator
//================================================================================

module lab_07_comp (VINP, VINN, CLK, VOP, VON, VGND, VDD) ;
// VGND and VDD are included to be pin-accurate with the schematic view
// PINS
 input VINP, VINN, CLK;
 output VOP, VON;
 inout VGND, VDD;
 electrical VINP, VINN, CLK, VOP, VON, VGND, VDD;

// INSTANCE PARAMETERS:
//     vhigh = Analog value for a Digital 1 on output [V]
//   p_off = Offset voltage added to VINP pin [V]
//   slack = The smallest time interval considered negligible
// (abstol for clock) [S]
//      td = intrinsic delay  from threshold cross to output 50%
// change [S]. H: That's why I use td - trf/2
//   trf = Transistion time for output rise/fall [S]
```

```
//     vth = CLK pin's threshold voltage [V]
//    vlow = Analog value for a Digital 0 on output [V]

   parameter real p_off = 10.0u;
   parameter real trf = 1n  from (0:inf);
   parameter real    td = 1n  from (trf/2:inf);
   parameter real   vhigh = 2.0;
   parameter real  vlow = 0.0;
   parameter real   vth = 1.0;

// LOCAL VARIABLES:
    real vin, outstate;

//================================================================================

    analog begin
      @(initial_step)  begin
        vin =V(VINP,VINN) + p_off;
        outstate =  (vin > 0.0) ?  vhigh : vlow  ;

      end

      // when re-CLKd need to determine if condition has changed

      @(cross(V(CLK)-vth,+1)) begin // on rising edge
        vin =V(VINP,VINN) + p_off;

    outstate =  (vin > 0.0) ?  vhigh : vlow  ;

    // normal and complement outputs
    V(VOP) <+  transition (outstate, td-trf/2, trf, trf );
    V(VON) <+ vhigh+vlow-V(VOP);

    end
endmodule
```

4) Create a behavioral Verilog-A model for an ideal sample and hold with enable using the code below. Create a symbol for the S&H.

```
`include "discipline.h"
`include "constants.h"

module sah_ideal(en, vin, vout, vclk);
input en, vin, vclk;
output vout;
electrical en, vin, vout, vclk;
parameter real vth = 1;
real vout_val;

        analog begin

                @ (cross(V(vclk) - vth, 1.0) or initial_step)
                        if (V(en) > vth) vout_val = V(vin);

                V(vout) <+ vout_val ;
        end
endmodule
```

5) We will also need Verilog-A models for NAND and NOR gates. We may use the gates available in ahdllib, or we may create new Verilog-A models as below.

```
`include "discipline.h"
`include "constants.h"

// INSTANCE parameters
//    vlogic_high = output voltage for high  [V]
//    vlogic_low   = output voltage for high  [V]
//    vth      = voltages above this at input are considered high  [V]
//    tdel, trise, tfall = {usual} [s]
//

module nand_gate(vin1, vin2, vout);
input vin1, vin2;
output vout;
```

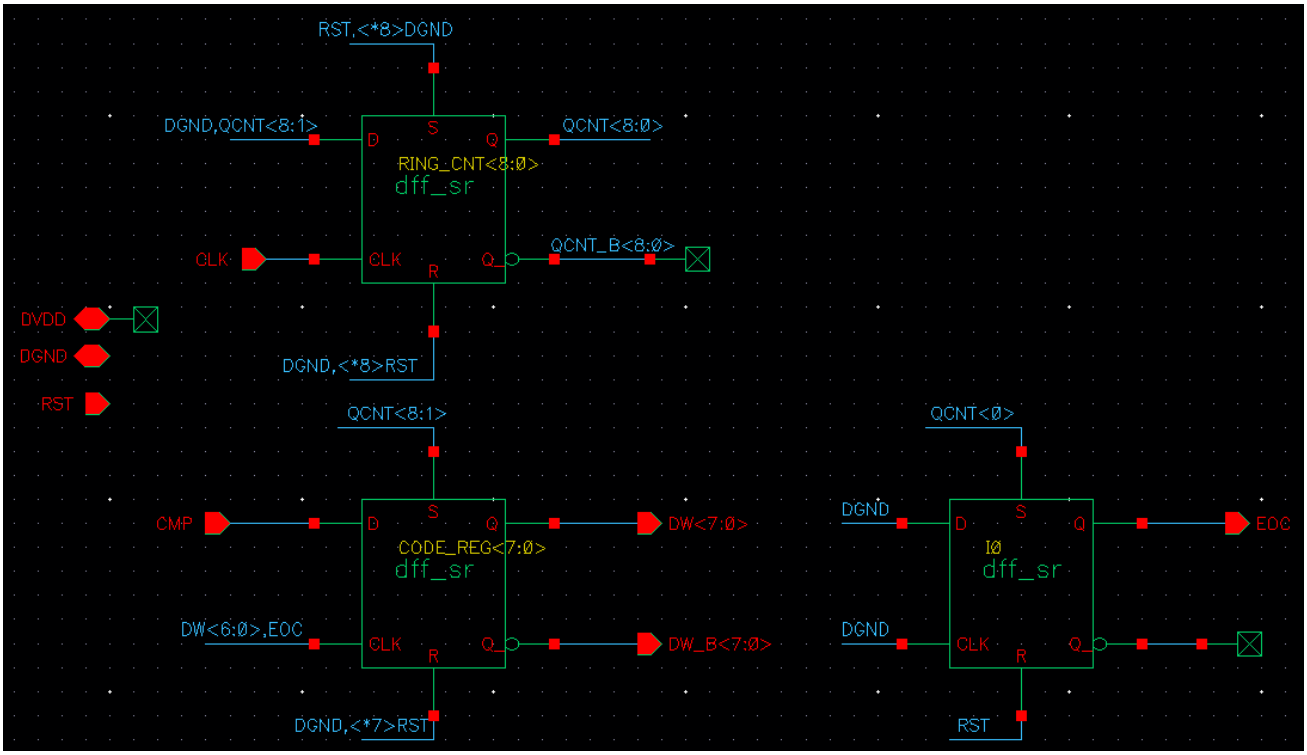
```
electrical vin1, vin2, vout;
parameter real vlogic_high = 2;
parameter real vlogic_low = 0;
parameter real vth = 1;
parameter real tdel = 2n from [0:inf);
parameter real trise = 1n from (0:inf);
parameter real tfall = 1n from (0:inf);

   real vout_val;
   integer logic1, logic2;

   analog begin

      logic1 = V(vin1) > vth;
      logic2 = V(vin2) > vth;

      @ (cross(V(vin1) - vth, 1))  logic1 = 1;
      @ (cross(V(vin1) - vth, -1)) logic1 = 0;

      @ (cross(V(vin2) - vth, 1))  logic2 = 1;
      @ (cross(V(vin2) - vth, -1)) logic2 = 0;

      //
      // define the logic function.
      //
      vout_val = !(logic1 && logic2) ? vlogic_high : vlogic_low;

      V(vout) <+ transition( vout_val, tdel, trise, tfall);
   end
endmodule
```

```
`include "discipline.h"
`include "constants.h"

// INSTANCE parameters
//    vlogic_high = output voltage for high [V]
//    vlogic_low  = output voltage for high [V]
//    vth      = voltages above this at input are considered high [V]
//    tdel, trise, tfall = {usual} [s]
//

module nor_gate(vin1, vin2, vout);
input vin1, vin2;
output vout;
electrical vin1, vin2, vout;
parameter real vlogic_high = 2;
parameter real vlogic_low = 0;
parameter real vth = 1;
parameter real tdel = 2n from [0:inf);
parameter real trise = 1n from (0:inf);
parameter real tfall = 1n from (0:inf);

   real vout_val;
   integer logic1, logic2;

   analog begin

      logic1 = V(vin1) > vth;
      logic2 = V(vin2) > vth;

      @ (cross(V(vin1) - vth, 1))  logic1 = 1;
      @ (cross(V(vin1) - vth, -1)) logic1 = 0;

      @ (cross(V(vin2) - vth, 1))  logic2 = 1;
      @ (cross(V(vin2) - vth, -1)) logic2 = 0;

      // define the logic function.
      vout_val = (!(logic1 || logic2)) ? vlogic_high : vlogic_low;

      V(vout) <+ transition( vout_val, tdel, trise, tfall);
   end
endmodule
```

6) Create appropriate testbenches to verify the operation of the Verilog-A models. Report transient simulation results showing proper operation of each block.

PART 2: SAR Logic

1) Create the schematic of SAR logic as shown below[2]. Study the operation carefully.

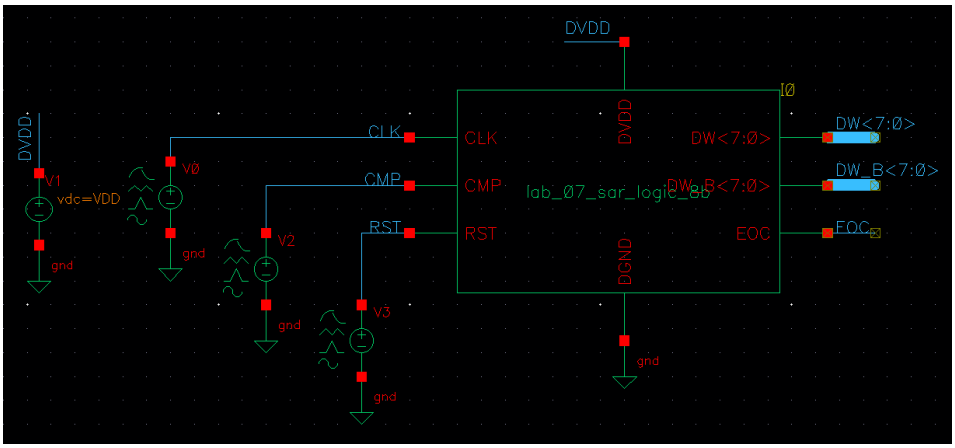


2) Create a testbench to verify SAR logic operation. Set FCLK = (NBIT + 2) * FS, where NBIT = 8 and FS = 1MHz.
3) Set sources (RST and CLK) as shown below.

	SMPL (RST)	CLK
Type	Pulse	Pulse
Zero value	0	0
One value	VDD	VDD
Period	TS	TCLK
Pulse width	TCLK	TCLK/2
Delay	TCLK/2 + TRF	TCLK/2
Rise/fall time	TRF	TRF

4) Report transient simulation results for the ring counter output, the code register output, and the EOC signal for the following cases:
    a. CMP is all zeros
    b. CMP is all ones
    c. CMP is alternating ones and zeros (period = 2*TCLK and pulse width = TCLK)

[2] In Mentor Pyxis bus notation use […] instead of <…> and ";" instead of ",".



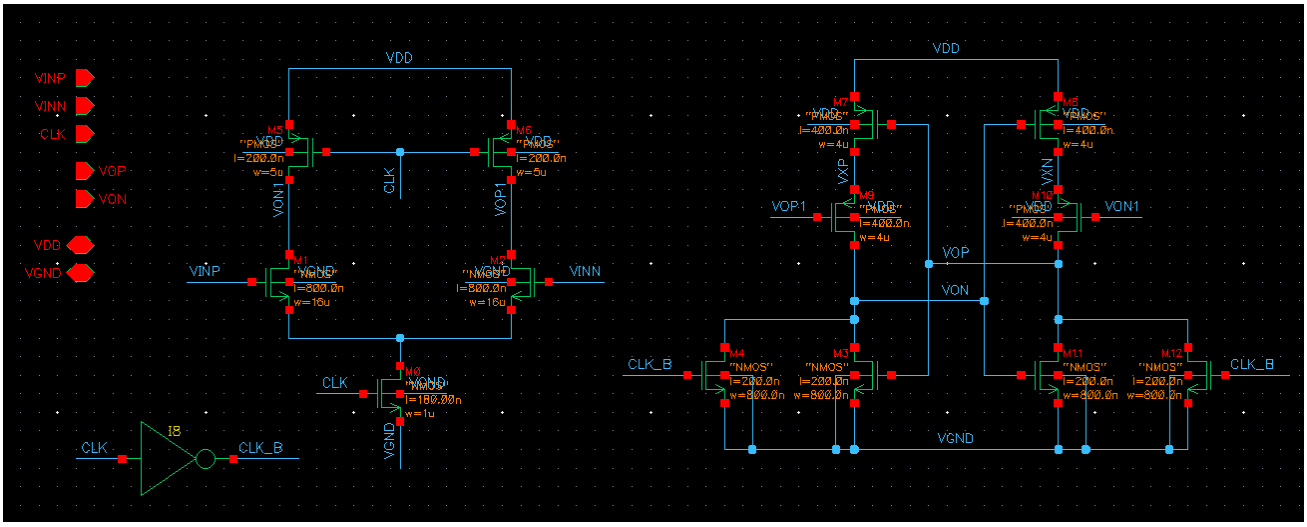
5) Briefly explain **<u>in your own words</u>** how does this design work and how does it implement the successive approximation algorithm[3]. Support your explanation by filling the state table below.

Clock cycle	DW<7>	DW<6>	DW<5>	DW<4>	DW<3>	DW<2>	DW<1>	DW<0>	CMP
1 (reset)									
2									B7
3									B6
4									B5
5									B4
6									B3
7									B2
8									B1
9									B0
10									

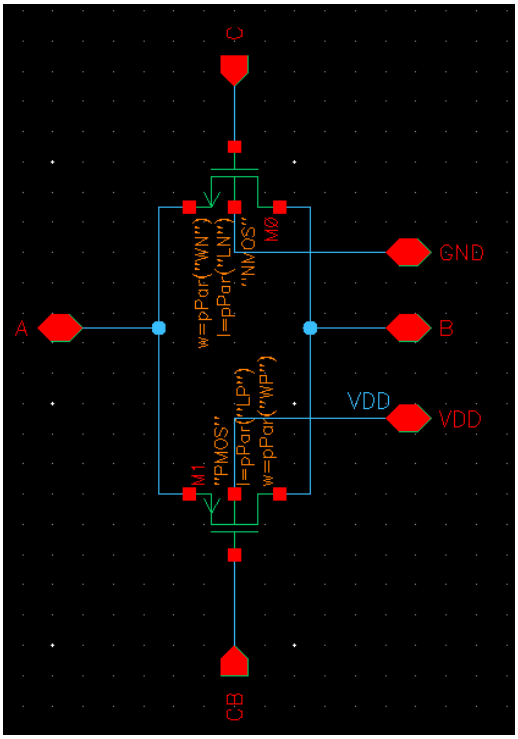
[3] You may find this reference useful (but be aware of typos and mistakes): Hedayati, Raheleh. "A study of Successive Approximation Registers and implementation of an ultra-low power 10-bit SAR ADC in 65nm CMOS technology." (2011).

PART 3: SAR ADC Testbench

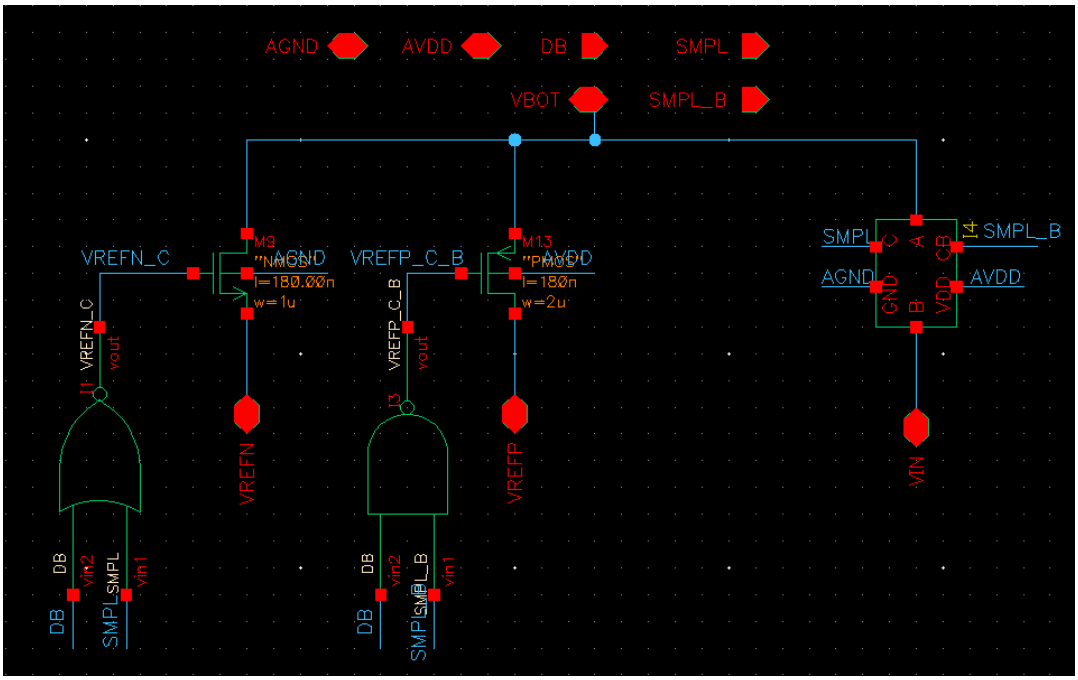
1) Create a new schematic view for the comparator cell in addition to the Verilog-A view. In the schematic view, copy the schematic of the comparator you created in Lab 06. Make sure the schematic and Verilog-A views are “pin-accurate”.



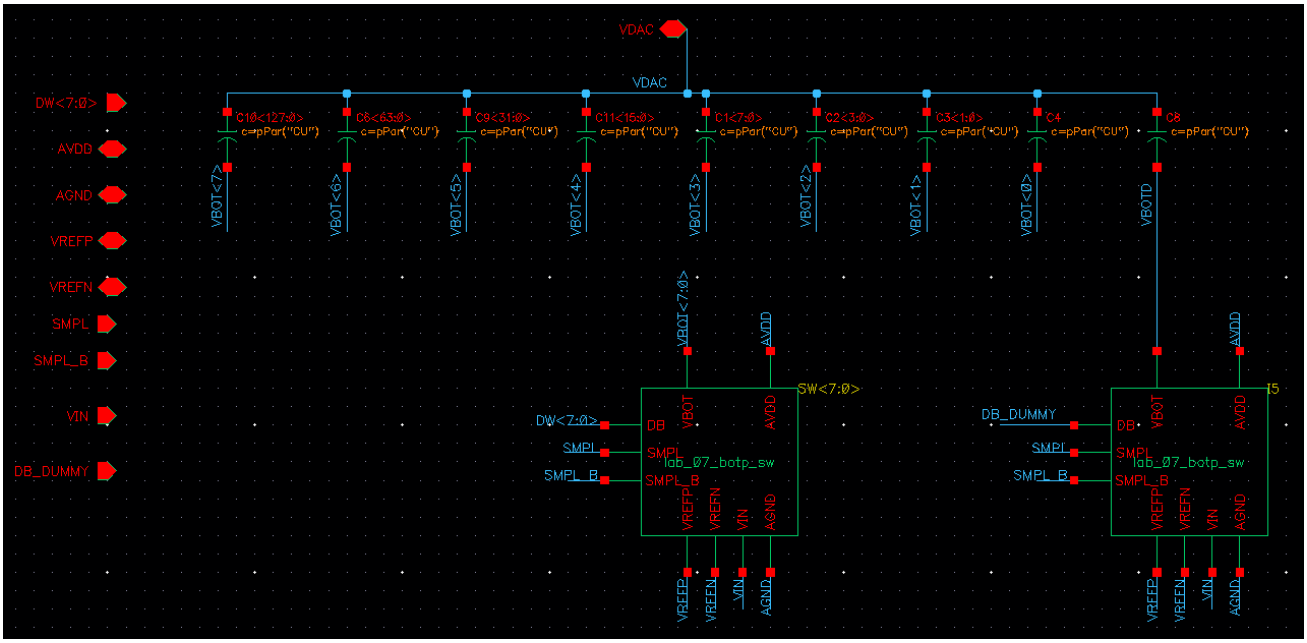
2) Transmission gate schematic. Set LN and LP to technology minimum. Set WN and WP to 1um.



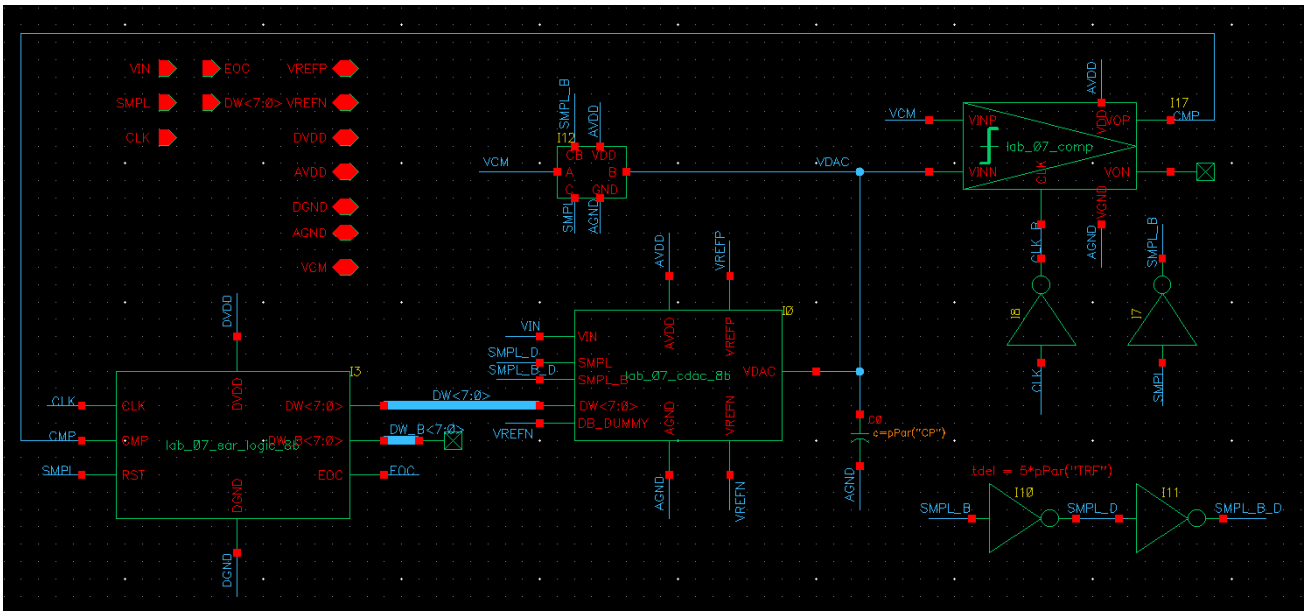
3) Bottom-plate switch schematic.



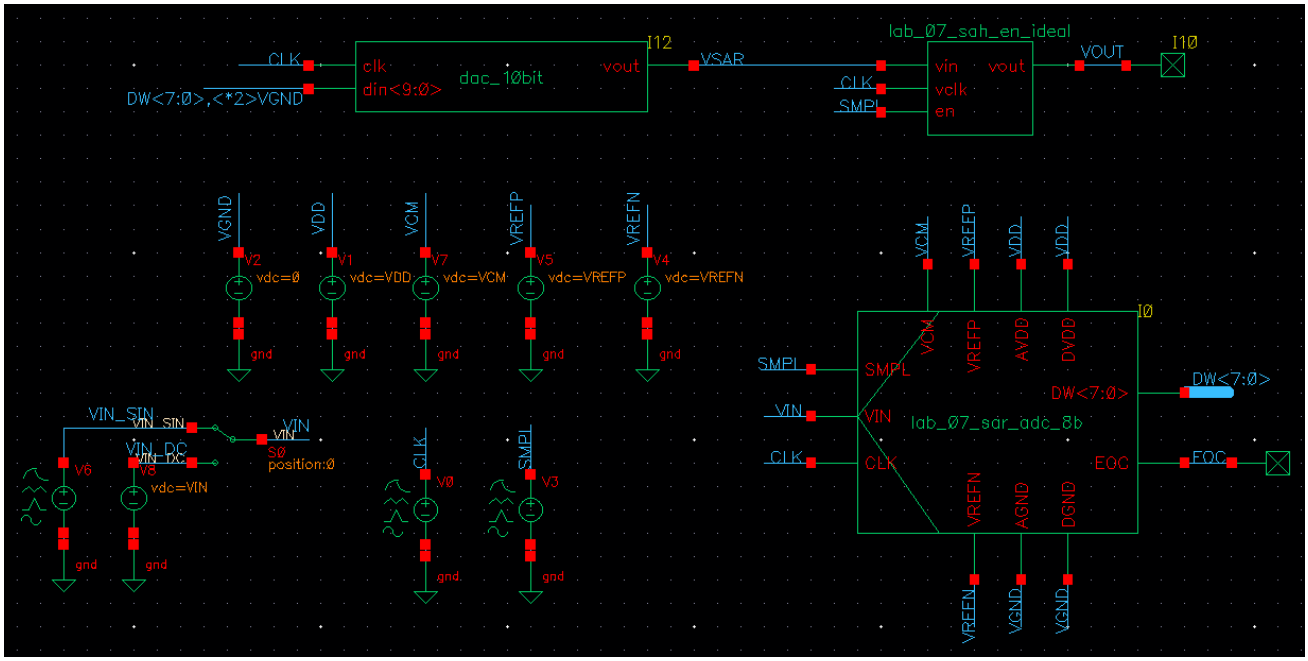
4) Capacitive DAC schematic.



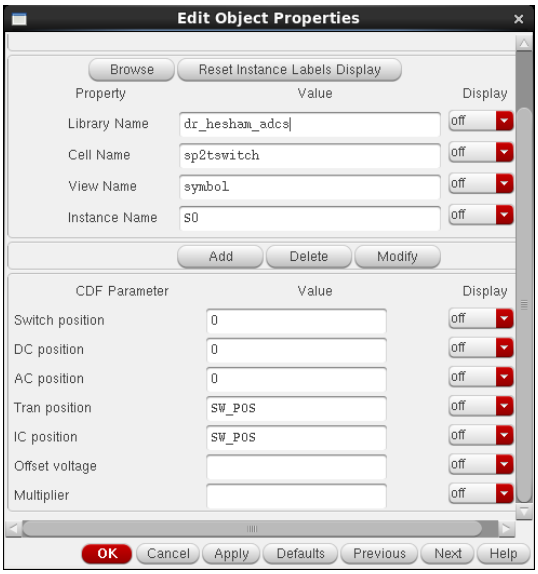
5) SAR ADC schematic. Set rise/fall/delay of all behavioral blocks to be TRF except the one shown below that has tdel = 5*TRF.



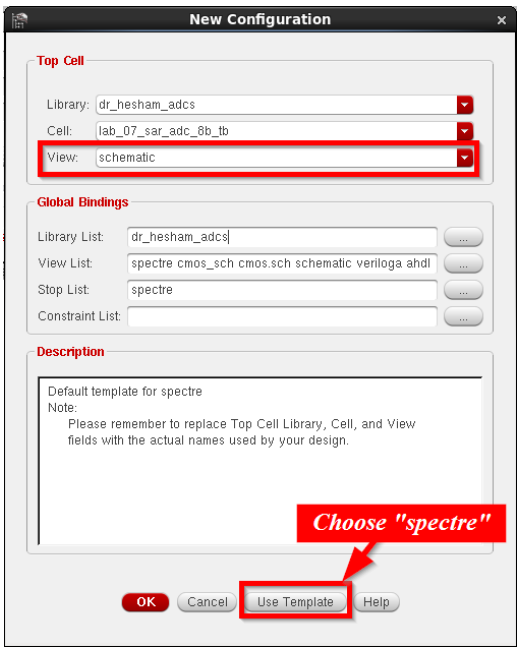
6) SAR ADC testbench.



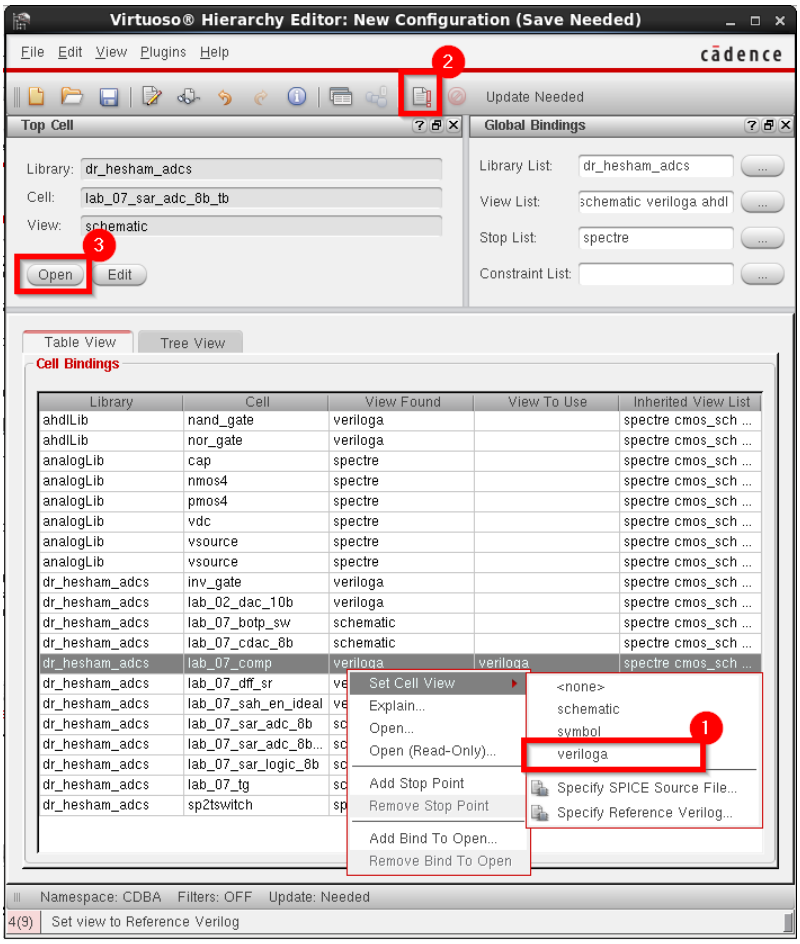
7) We use sp2tswitch (from analogLib) to select the input signal source (sin or dc) through global variables without changing the schematic. We control the switch position using SW_POS variable as shown below. If you use old version of Cadence, you first need to copy the sp2tswitch to your library and edit its CDF properties (from main Virtuoso window: Tools -> CDF -> Edit) to make the cell parameters “string” instead of “list”.



8) Close the window of the SAR ADC test bench schematic. Create a new view for the same test bench cell you have just created. Choose “config” as the view type.
   - The config view opens in “Hierarchy Editor”. Choose “spectre” template from the dropdown list.

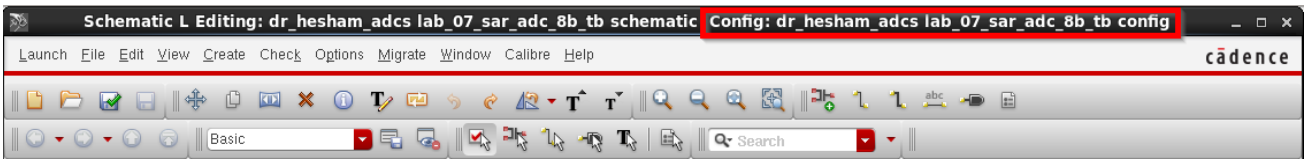


   - The comparator cell has two views. One is the transistor level schematic in the previous lab, and the other is the behavioral schematic you created in this lab. The hierarchy editor enables you to select which one you want to use for simulation. This technique is very powerful, and is used extensively to simulate different views of the same cell (behavioral, schematic, post layout, etc.).
   - From hierarchy editor, (1) choose the behavioral view of the comparator, (2) update the hierarchy (you need to update each time you edit your testbench), and (3) open the schematic from inside the Hierarchy Editor as shown below.

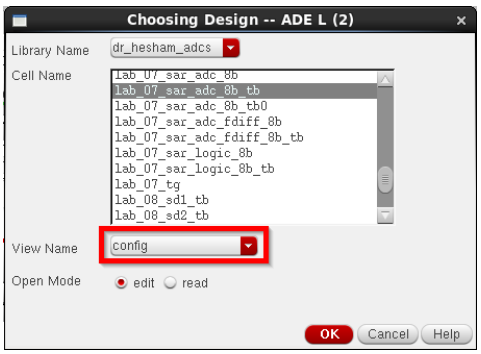


- **Note that every time you open your design, you must open the config view first. Do NOT open the schematic view directly from adexl.**

9) In the schematic view, make sure that the word “config” is shown in the title bar as shown below.



10) Create a new adexl view. Make sure that the design to be simulated by adexl is the config view as shown below.



PART 4: DC Functional Test

1) Set CLK and SMPL (RST) as in Part 2. Set global variables as shown below.

CU	20f
TCLK	100n
TRF	100p
TS	(NBIT+2)*TCLK
VOS	0
VDD	2
VCM	0.5*(VREFP + VREFN)
VREFP	0.75*VDD
VREFN	0.25*VDD
NBIT	8
VLSB	(VREFP – VREFN) / 2**NBIT
CP	0.05*CU*2**NBIT
TDROP	TS
TSTOP	3*TS
VIN	See below
SW_POS	2

2) Run transient simulation for three cases of VIN:
    a. VIN = VREFN → output will be all zeros
    b. VIN = VREFN + (128+32+8+2+0.5)*VLSB → output will be alternating zeros and ones
    c. VIN = VREFP → output will be all ones
3) Report the waveforms of VIN and VSAR overlaid for the three previous cases.
4) Report the waveforms of SMPL, CLK, CMP, and EOC for the second case.
5) Replace the behavioral comparator with the schematic view (use Hierarchy Editor). Repeat the transient simulation.

PART 5: Sine Wave Test

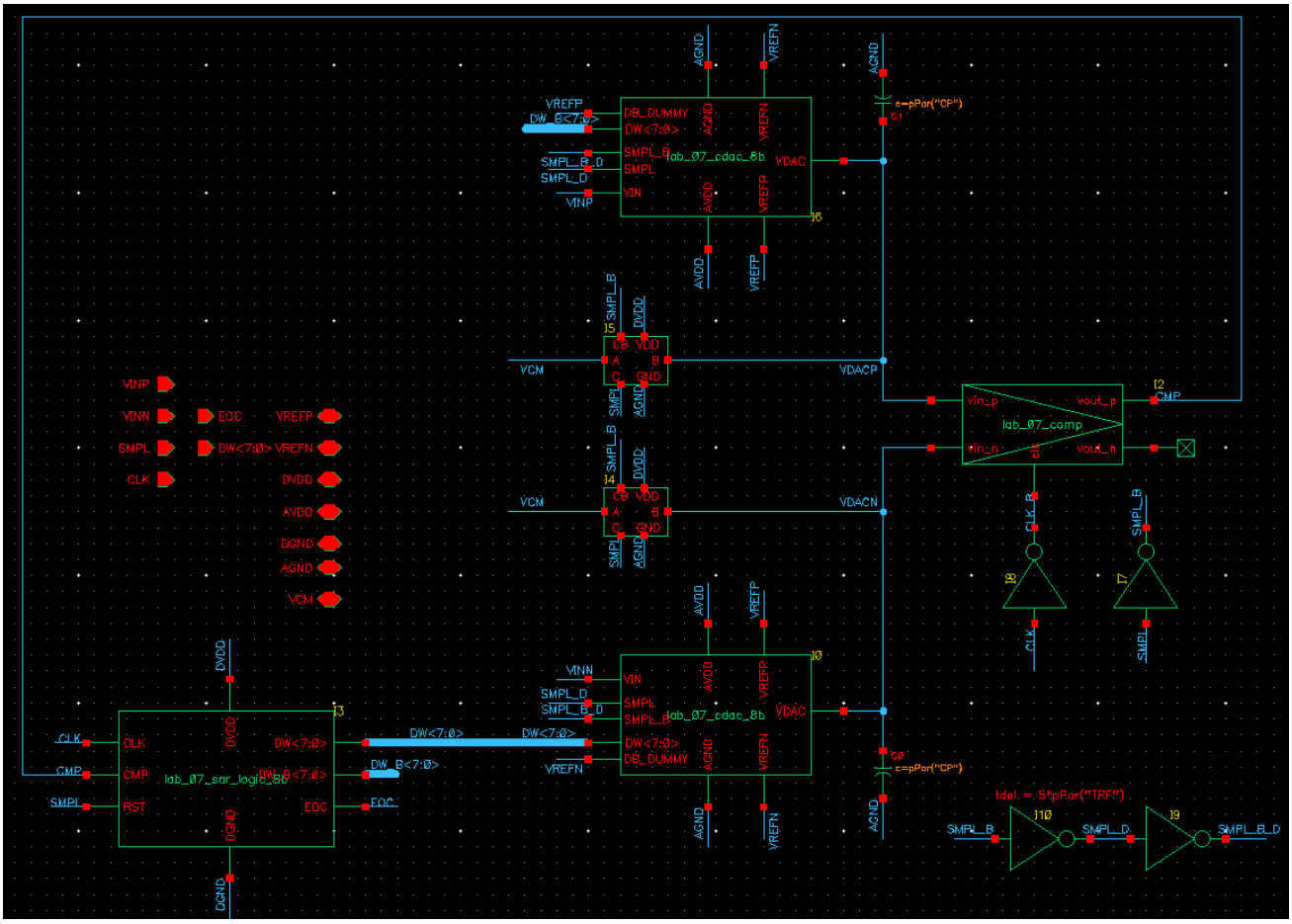
1) Switch back to the behavioral comparator (use Hierarchy Editor).
2) Set global variables as shown below. We may use a relatively small number of FFT points to speed up the simulation.

CU	20f
TCLK	100n
TRF	100p
TS	(NBIT+2)*TCLK
VOS	0
VDD	2
VCM	0.5*(VREFP + VREFN)
VREFP	0.75*VDD
VREFN	0.25*VDD
NBIT	8
VLSB	(VREFP – VREFN) / 2**NBIT
CP	0.05*CU*2**NBIT
NFFT	2**6
NCYC	5
FIN	(NCYC/NFFT)/TS
VPK	VDD/4
TDROP	0.5/FIN
TSTOP	NCYC/FIN + TDROP
SW_POS	1

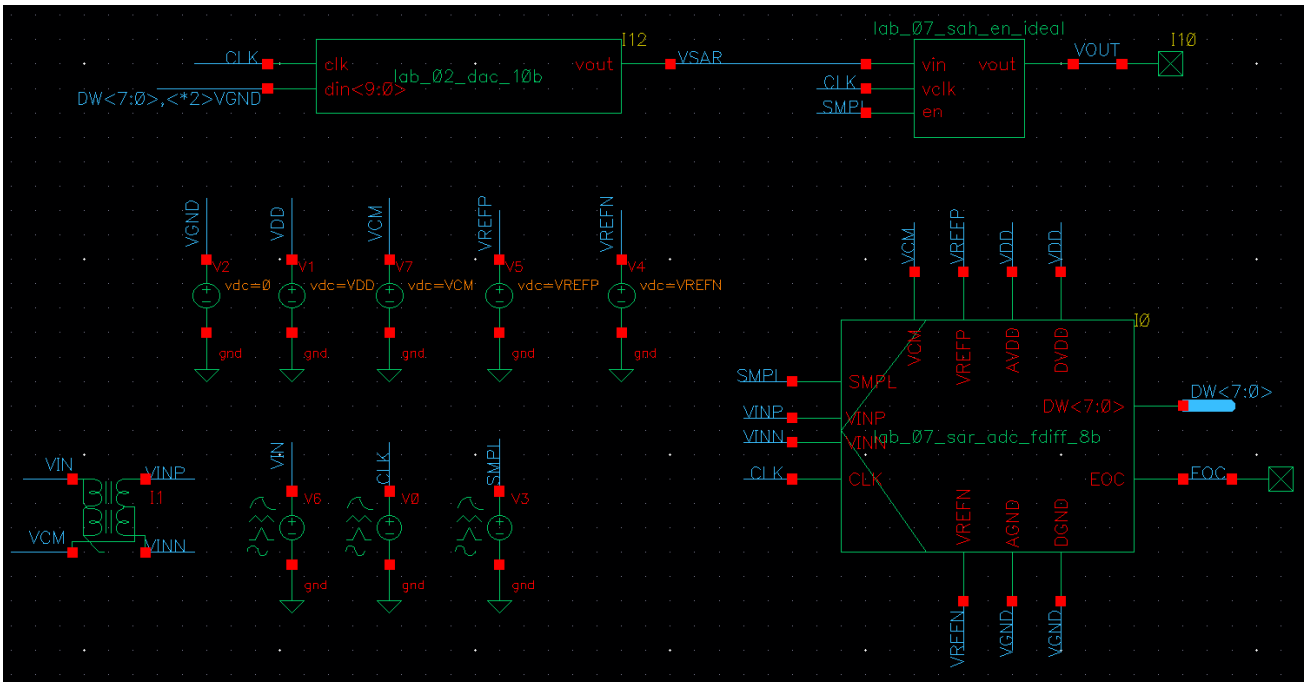
3) Run transient analysis. It is recommended that you first run the transient simulation with “liberal” setting, then repeat it with “conservative” setting after you make sure all outputs are as expected.
4) Plot transient waveforms of VIN and VOUT. Plot the FFT of the VOUT to measure the ENOB and other performance parameters.
5) Replace the behavioral comparator with the schematic view (use Hierarchy Editor). Repeat the transient simulation and plot the FFT.

PART 6: Fully-Differential SAR ADC

1) Create a new schematic for fully-differential SAR ADC as shown below (copy the SE schematic then edit it).



2) Create a new schematic for fully-differential SAR ADC testbench as shown below (copy the SE schematic then edit it).



3) Switch back to the behavioral comparator (use Hierarchy Editor).
4) Set global variables as shown below. We use a relatively small number of FFT points to speed up the simulation. Note that the differential operation doubles the amplitude. Remove the DC level for the differential sinusoidal input.

CU	20f
TCLK	100n
TRF	100p
TS	(NBIT+2)*TCLK
VOS	0
VDD	2
VCM	0.5*(VREFP + VREFN)
VREFP	0.75*VDD
VREFN	0.25*VDD
NBIT	8
VLSB	(VREFP – VREFN) / 2**NBIT
CP	0.05*CU*2**NBIT
NFFT	2**6
NCYC	5
FIN	(NCYC/NFFT)/TS
VPK	VDD/2
TDROP	0.5/FIN
TSTOP	NCYC/FIN + TDROP

5) Run transient analysis. It is recommended that you first run the transient simulation with “liberal” setting, then repeat it with “conservative” setting after you make sure all outputs are as expected.
6) Plot transient waveforms of VIN and VOUT. Plot the FFT of the VOUT to measure the ENOB and other performance parameters.
7) Replace the behavioral comparator with the schematic view (use Hierarchy Editor). Repeat the transient simulation and plot the FFT.