how big can you dream?™

cadence

Top Down Modeling and Test Bench Development

Verification Case Study: Pipeline ADC

April 2004 IEEE - Santa Clara Valley – Circuits and Systems chapter presentation

Jonathan David – Mixed Signal Methodology – Cadence

Top Down Modeling and Test Bench Development

Verification Case Study: Pipeline ADC

2002 IEEE International Workshop on Behavioral Modeling and Simulation

October 8, 2002

Jonathan David – Mixed Signal Methodology – Cadence

Top Down Modeling and Test Bench Development

Verification Case Study: Pipeline ADC

Understand

- Role of Behavioral Modeling in Top Down Verification
- Pipeline ADC Circuit Operation
- Top Level Simulations
- Behavioral Models and Test Blocks
- Circuit Level Simulations and Supporting Models

TOP DOWN DESIGN
Still the future of Mixed Signal Design?

- Theoretical Approach
  - Actually in use, Matlab/SPW -> Spec -> Designers
- Bottom Up Design is still Powerful
  - Circuit Knowledge + Creativity = New Approaches

BUT

- Practical Mixed Signal Simulation +
- Design Reuse +
- Decent Verification Environments =
- Top Down Mixed Signal System Verification
  - Starting EARLY in the design process
  - Gives Team Higher Visibility into Design Status

Can Top Down Design help the Analog Team?

- Some Automation for Design and layout is Already Available
  - Allows Reuse for Cells with tested Optimization Scripts
- Building Practical Characterized models of Existing Cells
  - lets System Architects focus resources on New needs

- Some Types of Automated Design are available
  - Filter Networks, Switched Cap, DSP
  - Depend on basic Building blocks requiring creative work!

A Methodology for Top Down Verification

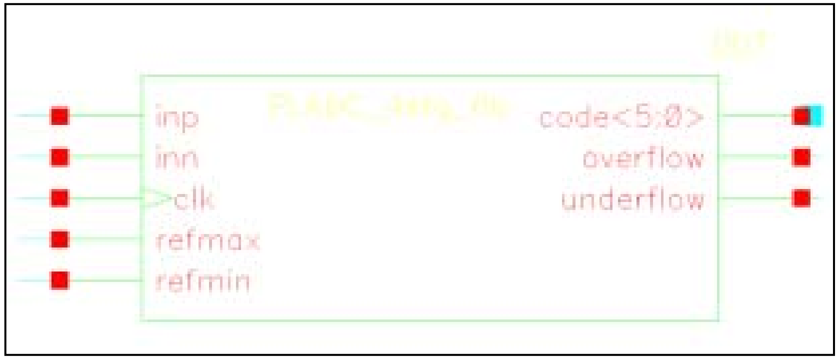
- Specification -> Test Plan
  - Quick Ramp Test
  - INL & DNL via Binning
  - ENOB via Nyquist Rate Sine Test
- Pin Accurate Model
  - Define all known I/O
  - Add behavior later
- Start Test Development & Architecture
  - Calc DNL INL from Bins
  - Calc ENOB from FFT of Sine

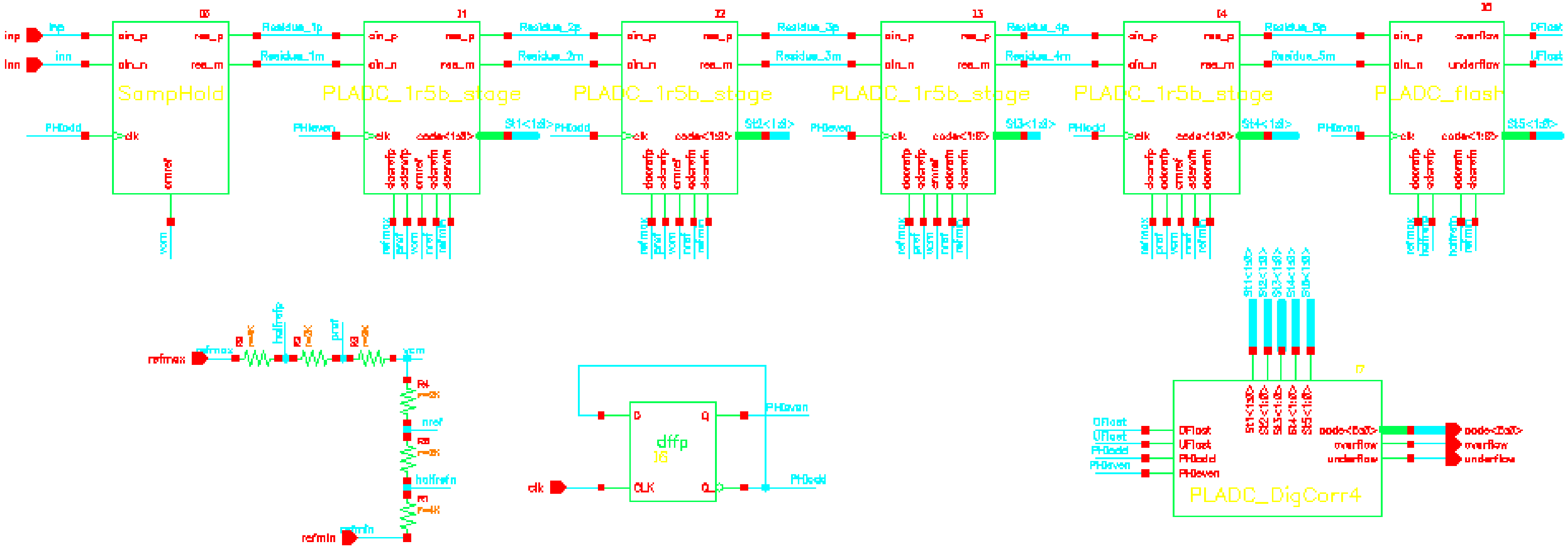
- ADC Specs
  - Input Range tran
  - Supplies and Biasing tran
  - Clocking & Timing tran
  - Output Drive tran
  - Coding tran
  - Linearity tran+random params
  - Noise & Distortion ‘ ‘

cadence

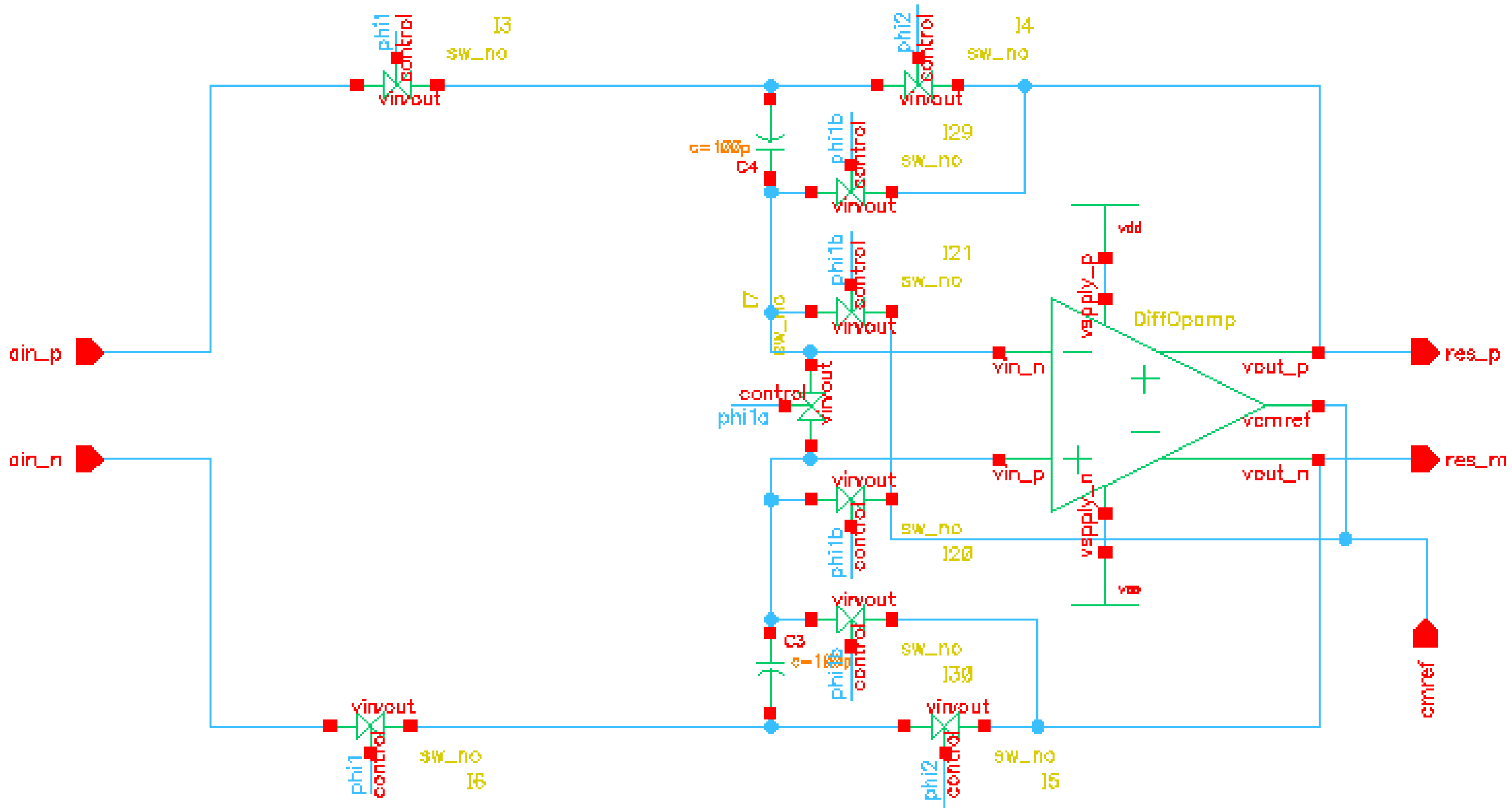
6b 1.5b/stage 4 stage Pipeline ADC.

- 3v supplies
- Differential Signal path .25 – 2.25v
- Vcm = 1.25
- Digital Error Correction



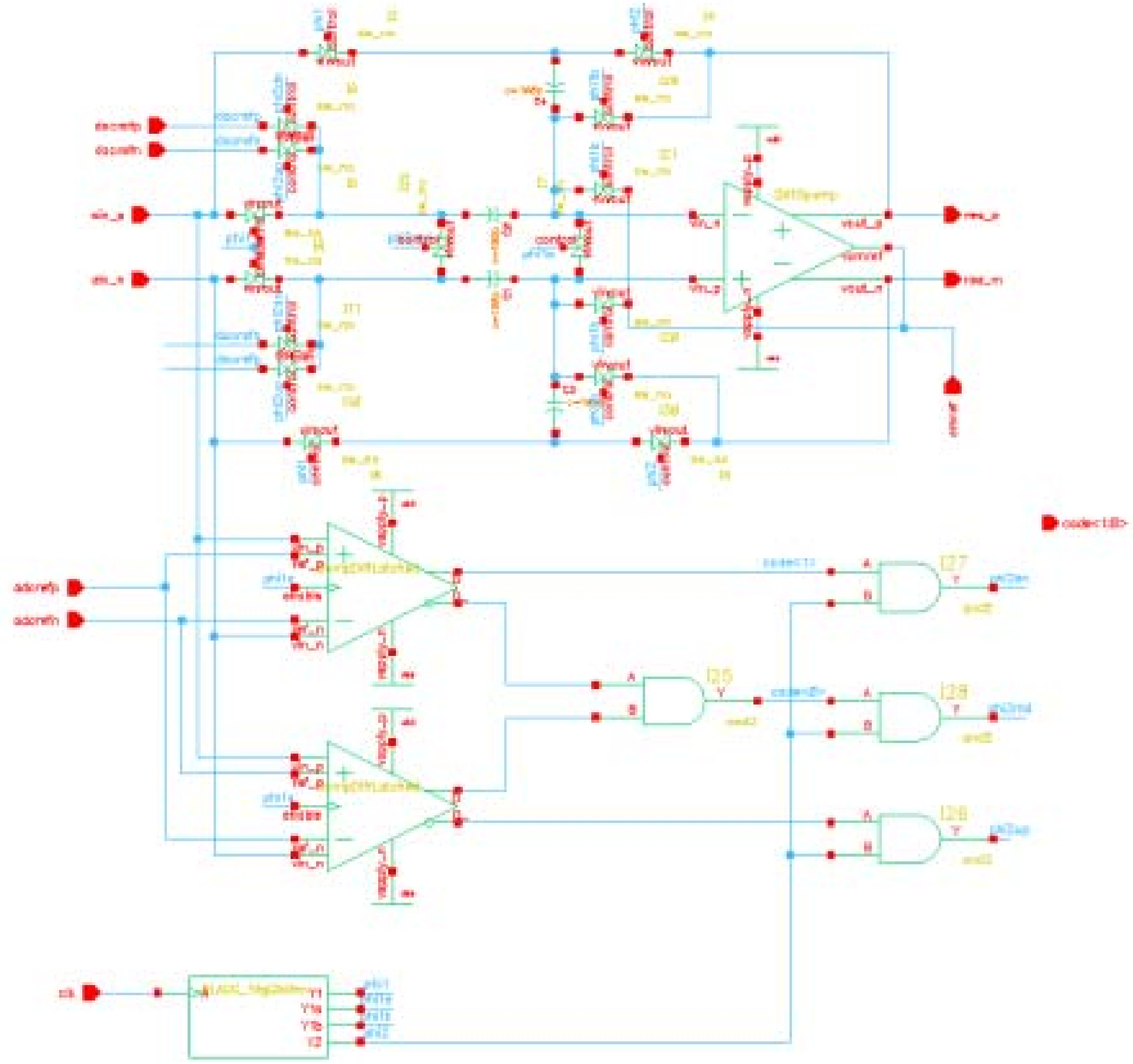


SAH



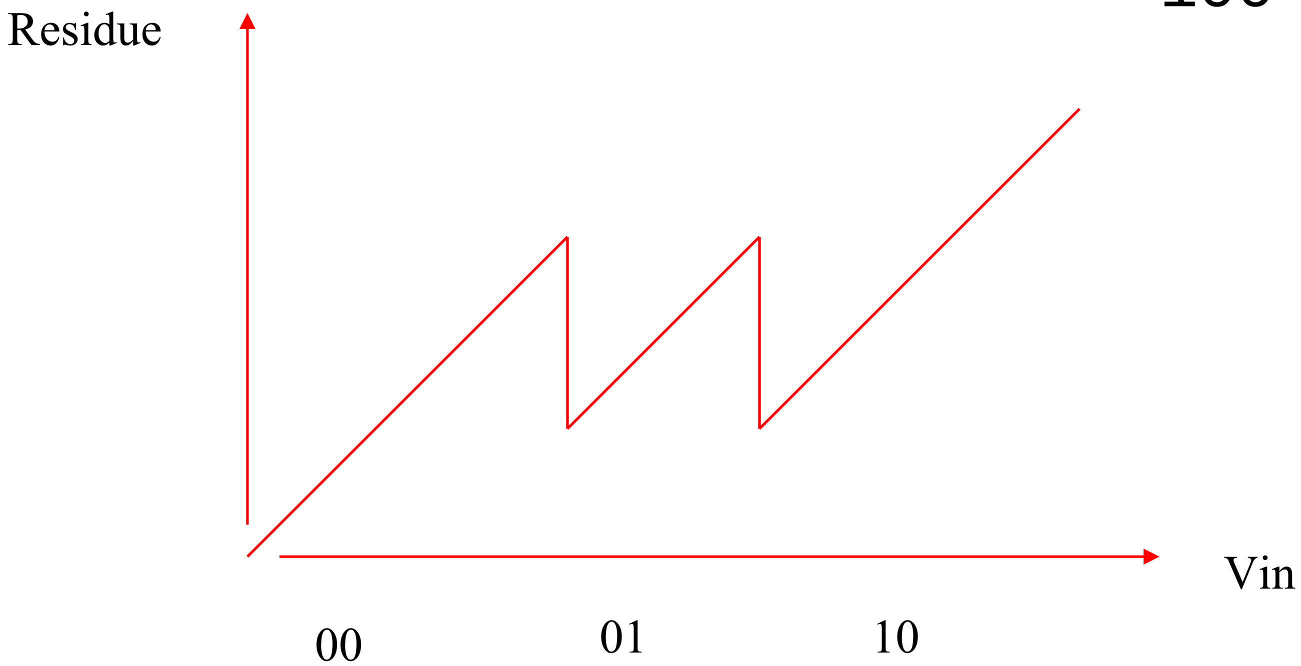
ain_p
ain_n
res_p
res_m
cmref
I3
I4
I5
I6
I7
I20
I21
I29
I30
sw_no
phi1
phi2
phi1b
phi1a
control
vin
vout
C4
C3
DiffOpamp
vin_n
vin_p
vout_p
vout_n
vcmref
vsupply_p
vsupply_n
vdd

1.5b Pipeline Stage

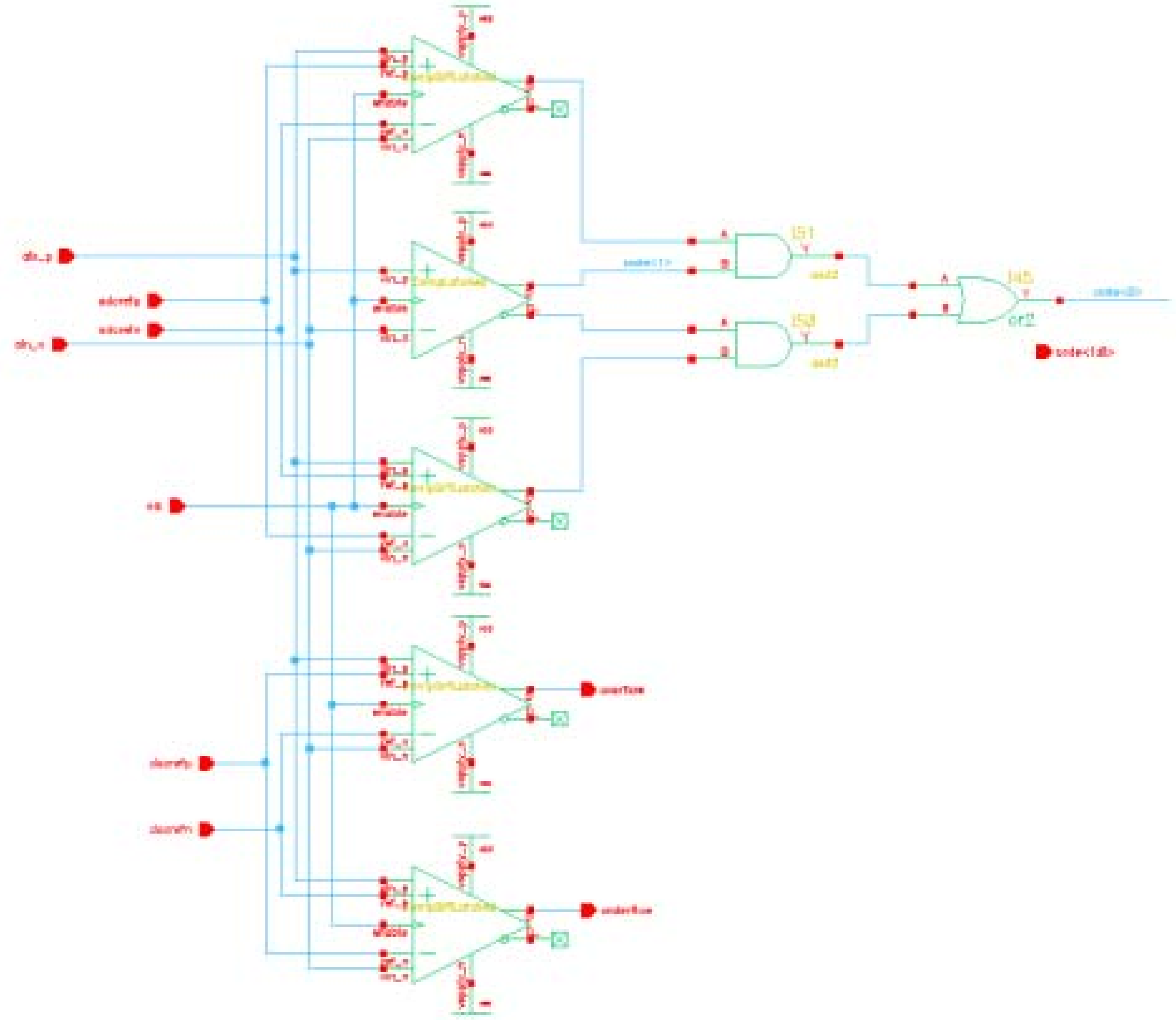


Transfer Function

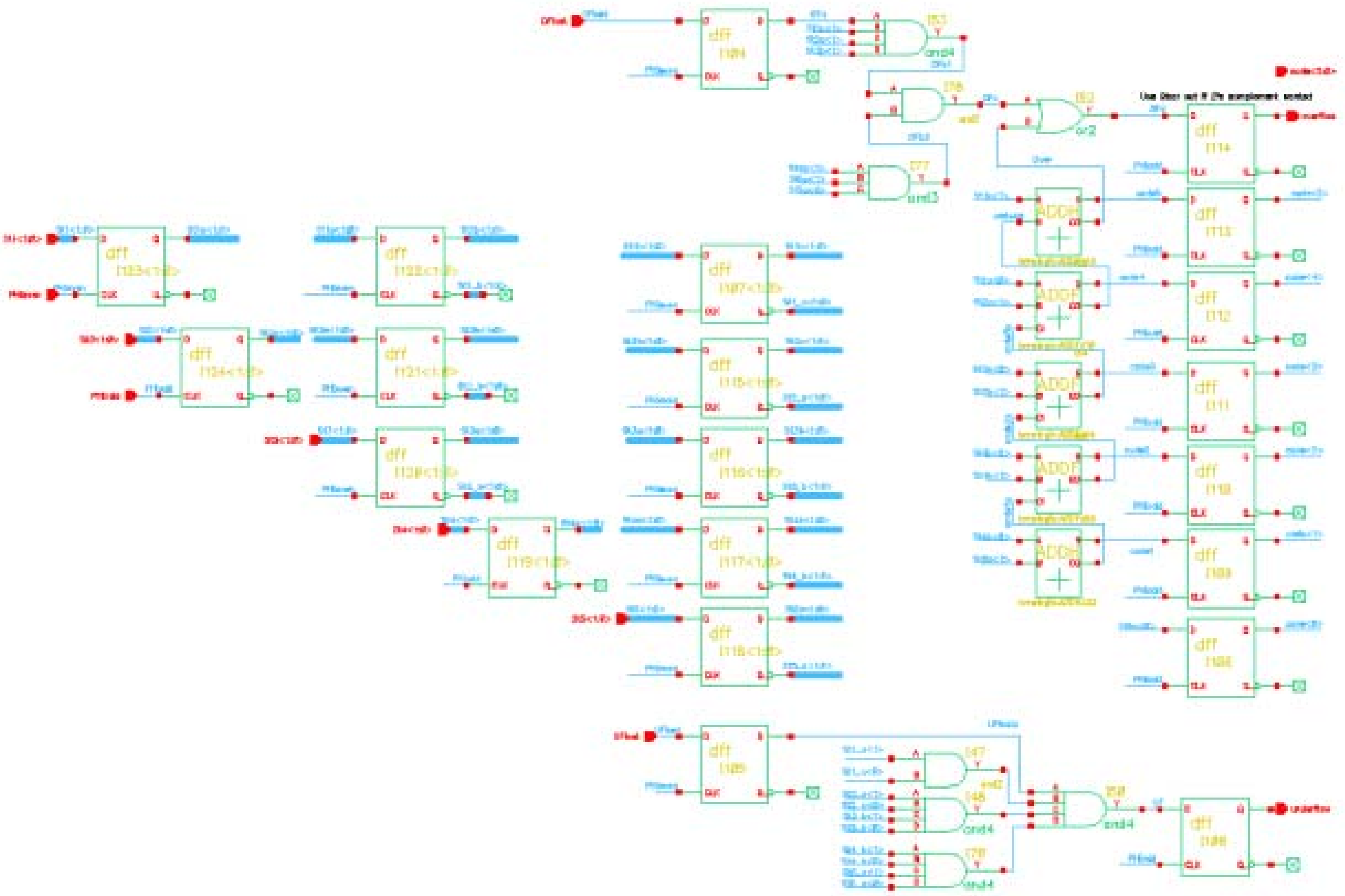
```
10      01
 00      10
---     ---
100     100
```



Flash Stage



Digital Correction Block

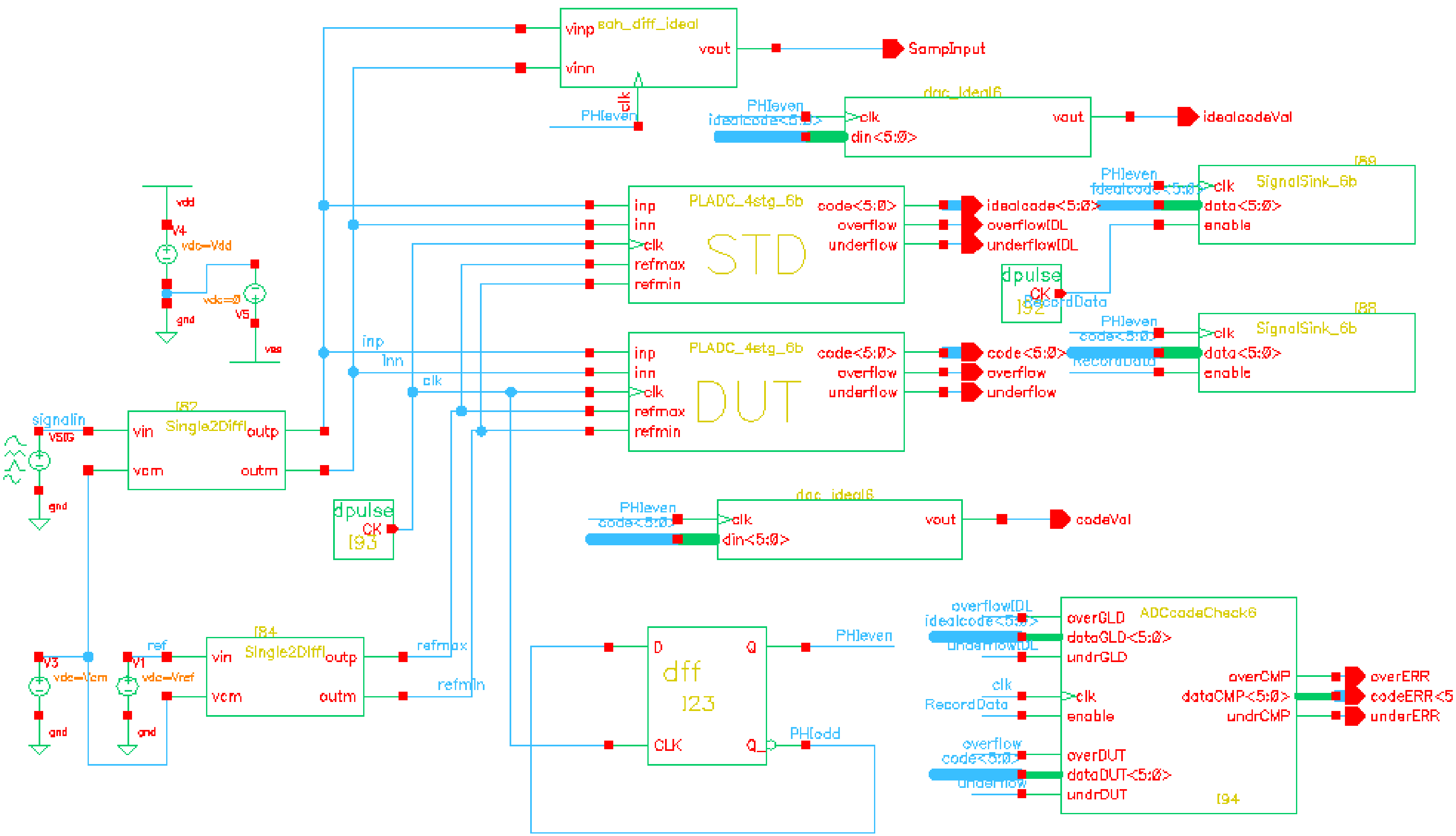


Functional Test Details

- Delays
  - Compare outputs at clock edge and after MAX Tp spec
  - Mismatch at other times indicates wrong prop delays in models
- Input Range
  - Simple Low to High Sweep – to hit all Codes
  - Mismatch in output codes indicates wrong input model/ sampling delay

cadence

Test bench Schematic



First Order Model

- Delays
  - clk –> samp & clk -> out
- ENOB – Noise
- INL-DNL - Random Offsets

```
always @(posedge clk)
       #dT sclk = !sclk;
```

```
always @(posedge sclk) begin
      // add the noise
      vn   = vnoise * ($dist_normal(seedn,0,1000)%6000) / 1000.0;
      vnr  = vnref  * ($dist_normal(seedr,0,1000)%6000) / 1000.0;
      // get the input and reference
      vin  = V(inp,inn)       + vn;
      vref = V(refmax,refmin) + vnr;
end
```

First Order Model

- Real to Bits
  - Store sequence to model latency

```
always @(negedge sclk) begin
   #td code = code1;
   // set the overflow bits
   overflow = of1;
   underflow = uf1;
   code1 = code2; of1  = of2; uf1  = uf2;
   code2 = code3; of2  = of3; uf2  = uf3;
   of3  = vin >  vref;
   uf3  = vin < -vref;
   codeval = (vin/vref/2.0)*(fullscale) + fullscale/2 -0.5;
       // internal storage in 2's complement
   code3 = ( codeval>=(fullscale-1) ? fullscale-1 :
             ( codeval<=0 ? 0 : codeval ));
end
```

Compare Block Details

- **Instantaneous Comparison (SimVision Compare Function is Better)**

```
assign overCMP = overDUT^overGLD;
assign undrCMP = undrDUT^undrGLD;
assign dataCMP = dataDUT-dataGLD; // difference better for the data
```

- **Open a Log File (include date & time in name – Thx to Jon Brenner)**

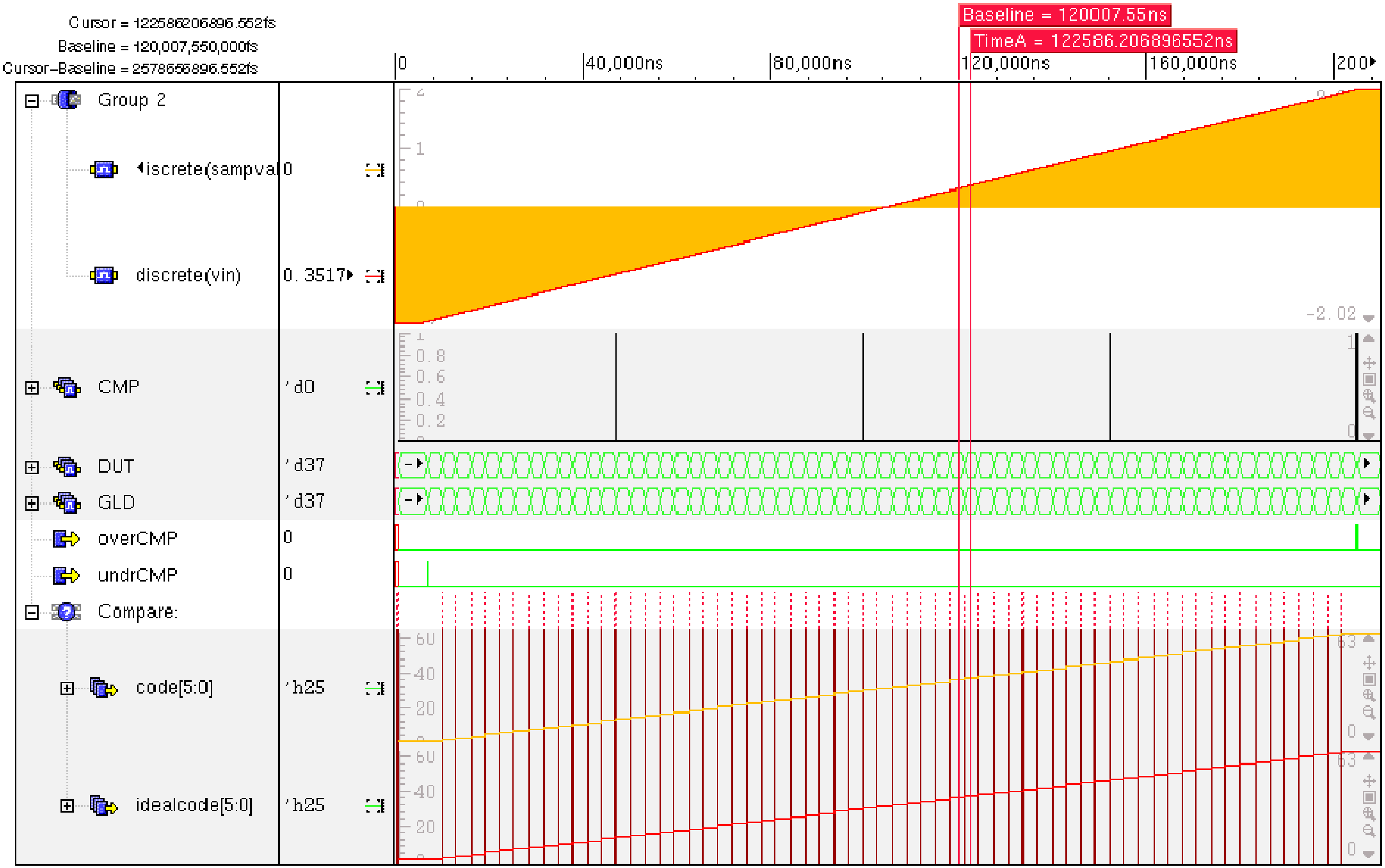
```
initial  begin
   $system("rm now.txt; date +%Y%m%d_%H%M > now.txt"); // update datestring
   datefile = $fopen("now.txt","r"); // open the date for read
   rstat = $fscanf(datefile,"%s",datestring);
   $fclose(datefile);
   filestring = {filename, datestring,".dat"};
   fileid = $fopen(filestring);
   $fstrobe(fileid,
     "ADC Output Data Comparison File : Testblock %M : date %s.%s.%s:%s",
     datestring[8*13:8*9+1],datestring[8*9:8*7+1],datestring[8*7:8*5+1],
     datestring[8*4:1]);
   $fstrobe(fileid, "Time    Delta : O ## U (DUT) sb= (GLD) O ## U");
```

Record Delta Warnings & Failures to File & Log

```
always @(clk) begin
     #Td  GLD = dataGLD+overGLD-undrGLD;
     DUT = dataDUT+overDUT-undrDUT;
     CMP = GLD - DUT;
     if (enable) begin
        $fstrobe(fileid, "%t  %d : %b %d %b (%d) sb= (%d) %b %d %b ",
           $realtime, CMP, overDUT, dataDUT, undrDUT, DUT,
           GLD, overGLD, dataGLD, undrGLD);
        if (  (   dataDUT !== dataGLD)
              || (overDUT !== overGLD)
              || (undrDUT !== undrGLD) ) begin
           if ( CMP >= -TolLsb && CMP <= TolLsb ) begin //specwarn
              $strobe("SPECWARN: %t ADC DUT(%d) != ADC GLD(%d)",
                 $realtime, DUT, GLD);
              $fstrobe(fileid, "SPECWARN: %t ADC DUT(%d) != ADC GLD(%d)",
                 $realtime, DUT, GLD);
           end else begin // specfail will also fail on any X values
              $strobe("SPECFAIL: %t ADC code Delta=%d > spec=%d",
                 $realtime, CMP, TolLsb);
              $fstrobe(fileid, "SPECFAIL: %t ADC code Delta=%d > spec=%d",
                 $realtime, CMP, TolLsb);
```

Wreal Model without Noise or Non Linears

Some Delays are not yet matched –esp Sample



Supplies and Reference Checking

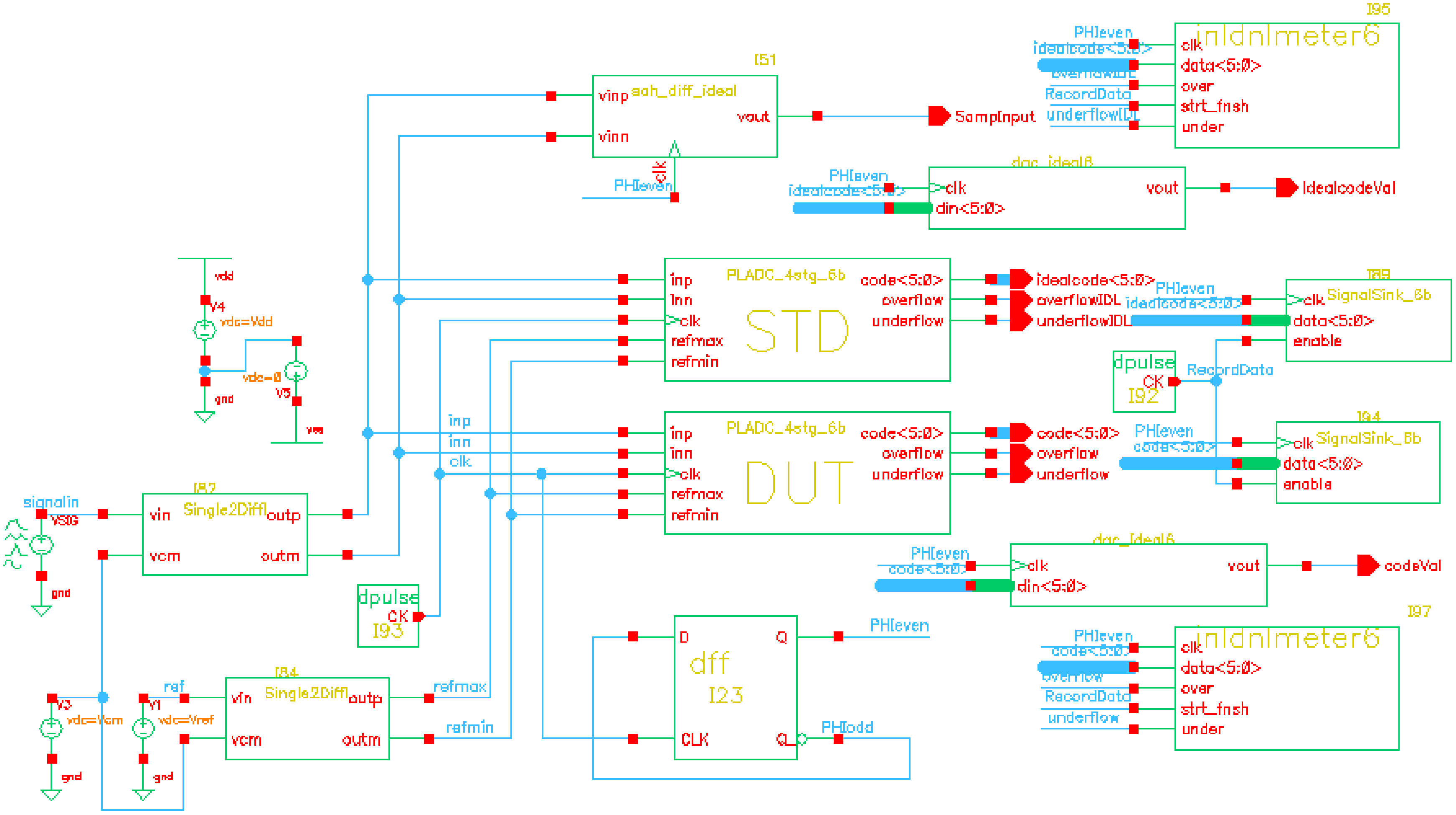
- Can extend to PVT simulations –
  - Repeat this test for all combinations of supplies, references, process corner & Temperature needed.
  - Can extend testbench/probes to record additional nodes to check in results.
- Simple Ramp Check is sufficient to check correctly assembled.
- These checks should be re-run frequently as the design progresses

Ramp Test Details

- INL & DNL Determination
  - Classic method : Determine transition points exactly
  - AutoTest method: Take many Samples, Use Histogram for DNL
    - Need to correct counts based on input wave type unless Ramps are used
- Second method with Ramp Source is present Solution
  - Could be adjusted for Sinusoidal input fairly easily
  - Warn User if endpoints / out-of-range values hit!
- Use endpoint bins?
  - Only if over-range and under-range indicator
  - To separate Out-of-Range values from valid measurements

cadence

Test Bench for INL DNL tests



DNL test – Initialize the bins

```
always @(posedge strt_fnsh) begin
    for (i = 0; i<= maxcode; i = i+1) begin
      bins[i] = 0; dnl[i] = 0; inl[i] = 0;
    end
    dnlmax = 0;
    inlmax = 0;
    totcount = 0;
    counting = 1;
    sum = 0;
    $fdisplay(datafile,
           "        counts        DNL           INL ");
  end
```

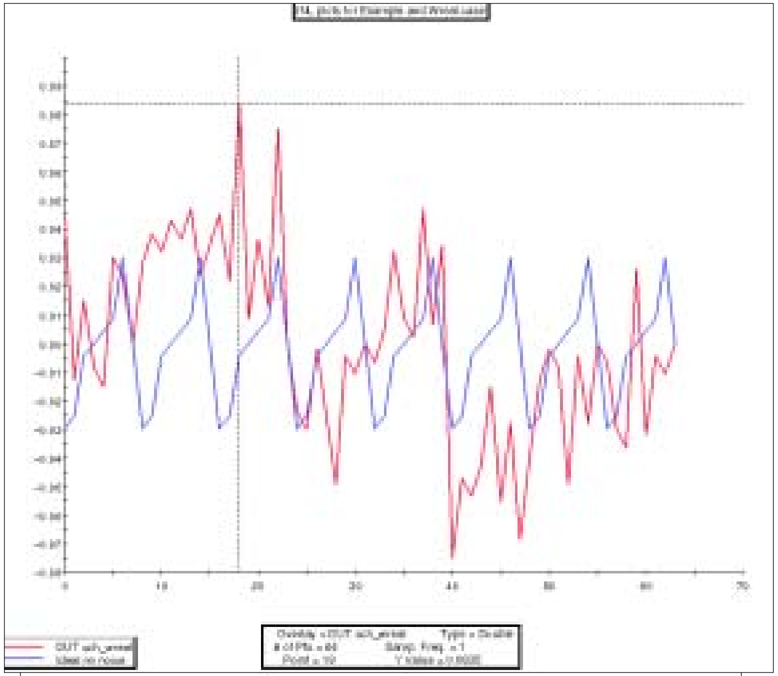
Do the Math!

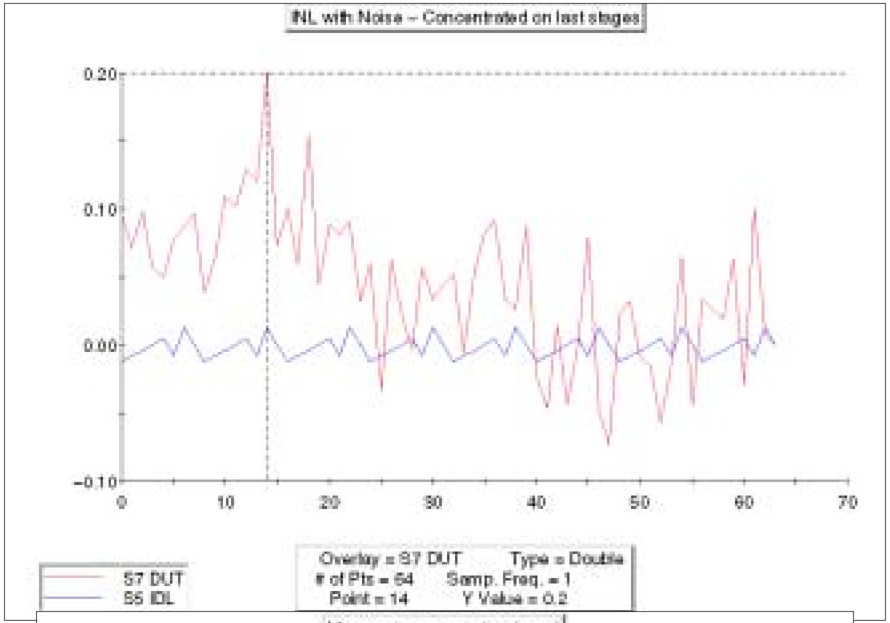
```
always @(negedge strt_fnsh) begin
   counting = 0;
   if (totcount > 0) begin
      idealbin = totcount/numcodes; // numcodes is real
      for (j = 0; j<= maxcode; j = j+1) begin
         dnl[j] = (bins[j]/idealbin) - 1.0;
         if (j>0) inl[j] = inl[j-1] +dnl[j];
         else inl[j] = dnl[j];
         $fdisplay(inlfile, "%10.3g # bin %d",inl[j], j ); // for SPW plotting
         $fdisplay(dnlfile, "%10.3g # bin %d",dnl[j], j ); // for SPW plotting
         $fdisplay(datafile, "%d       %d       %10.3g     %10.3g ",  j, bins[j], dnl[j], inl[j]  );
         if (dnl[j] > dnlmax)  dnlmax = dnl[j];
         else if (-dnl[j] > dnlmax ) dnlmax = -dnl[j];
         if (inl[j] > inlmax)  inlmax = inl[j];
         else if (-inl[j] > inlmax ) inlmax = -inl[j];
      end
      $fstrobe( datafile, "\n max abs dnl: %10.3g  inl: %10.3g ",    dnlmax, inlmax);
      $strobe( "\n %s max abs dnl: %10.3g  inl: %10.3g ",  filename,  dnlmax, inlmax);
   end
end
```

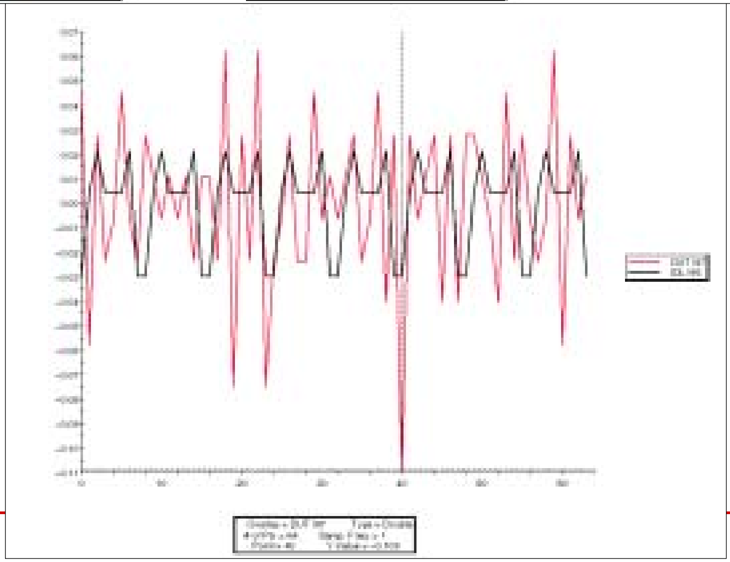
Count Each Code

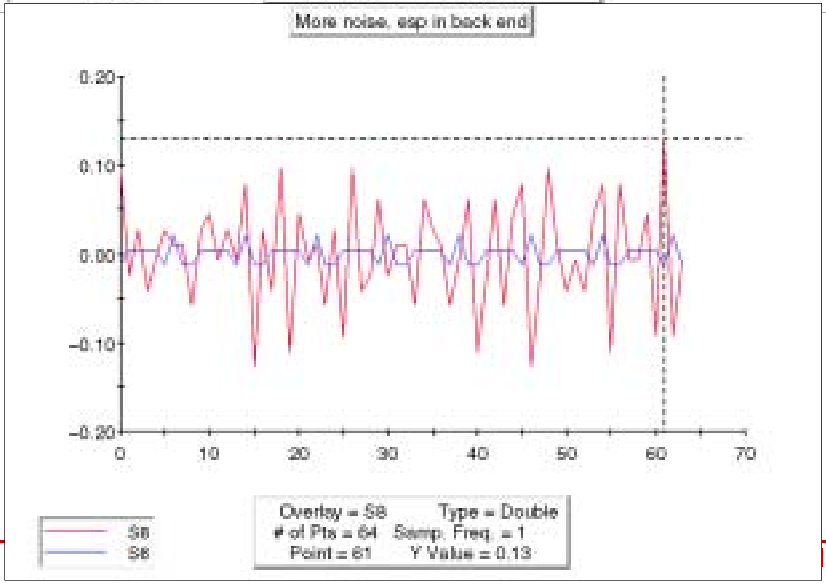
```
always @(posedge clk) begin
   if (counting && ((^data) !== 1'bx) && ((^data) !== 1'bz)
        && !under && !over ) begin
      bins[data] = bins[data] +1;
      //   debug   !!!
      bintest = bins[data];
      //   debug   !!!
      totcount = totcount+1;
      sum = sum + data;
   end
 end
```

DNL & INL Plots



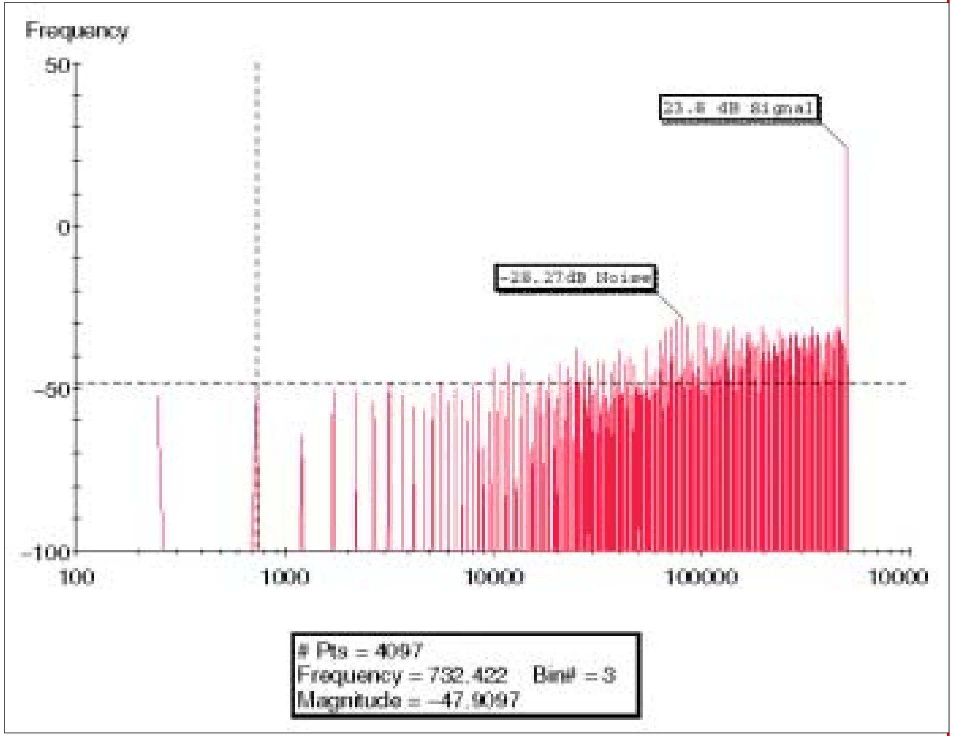




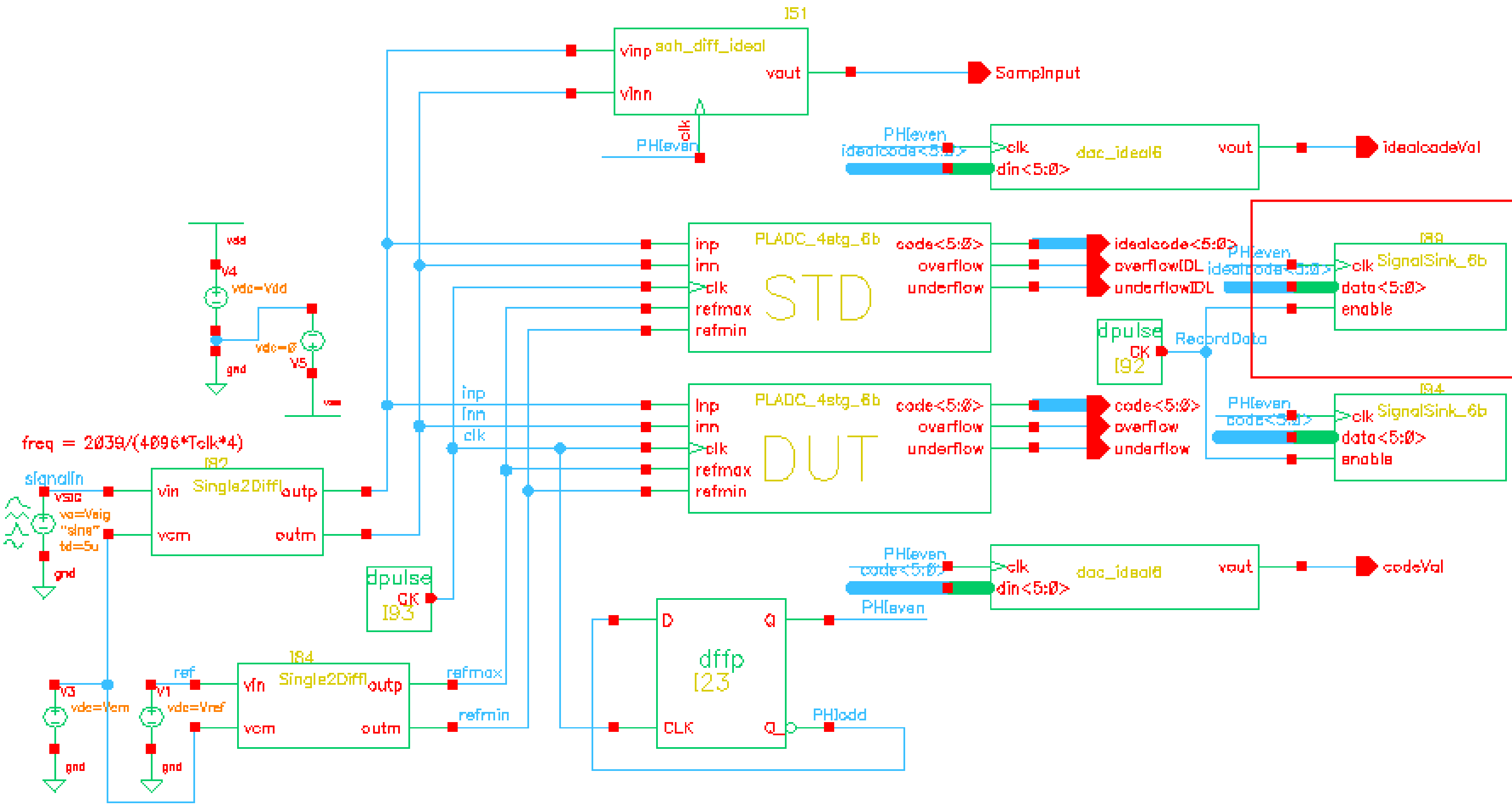


Sine Test Details

- Sample input ≈ Nyquist Rate
- Integer # Cycles in 2^N samples
- NOT Subharmonic of Sample Rate
- -> Need Prime Number ≈ 2^{N-1}
- Input cannot oversaturate codes
  - (no “over” or “under” allowed)
- No Harmonics (can’t measure THD)
- SNR = sample(dB) / RSS noise(dB)
- FFT Methods
  - package FFTW routine for VPI
  - Matlab
  - SPW <- Cadence tool



Test Bench for Sine Test



SPW FFT Plot For 2nd Order Model

S2=fft_transform(S1, fftlen=4096, num_frames=1, overlap=0, window=rectangular, rm_mean, half_spectrum, normalize

S3= abs(conjugate(S2)*S2)

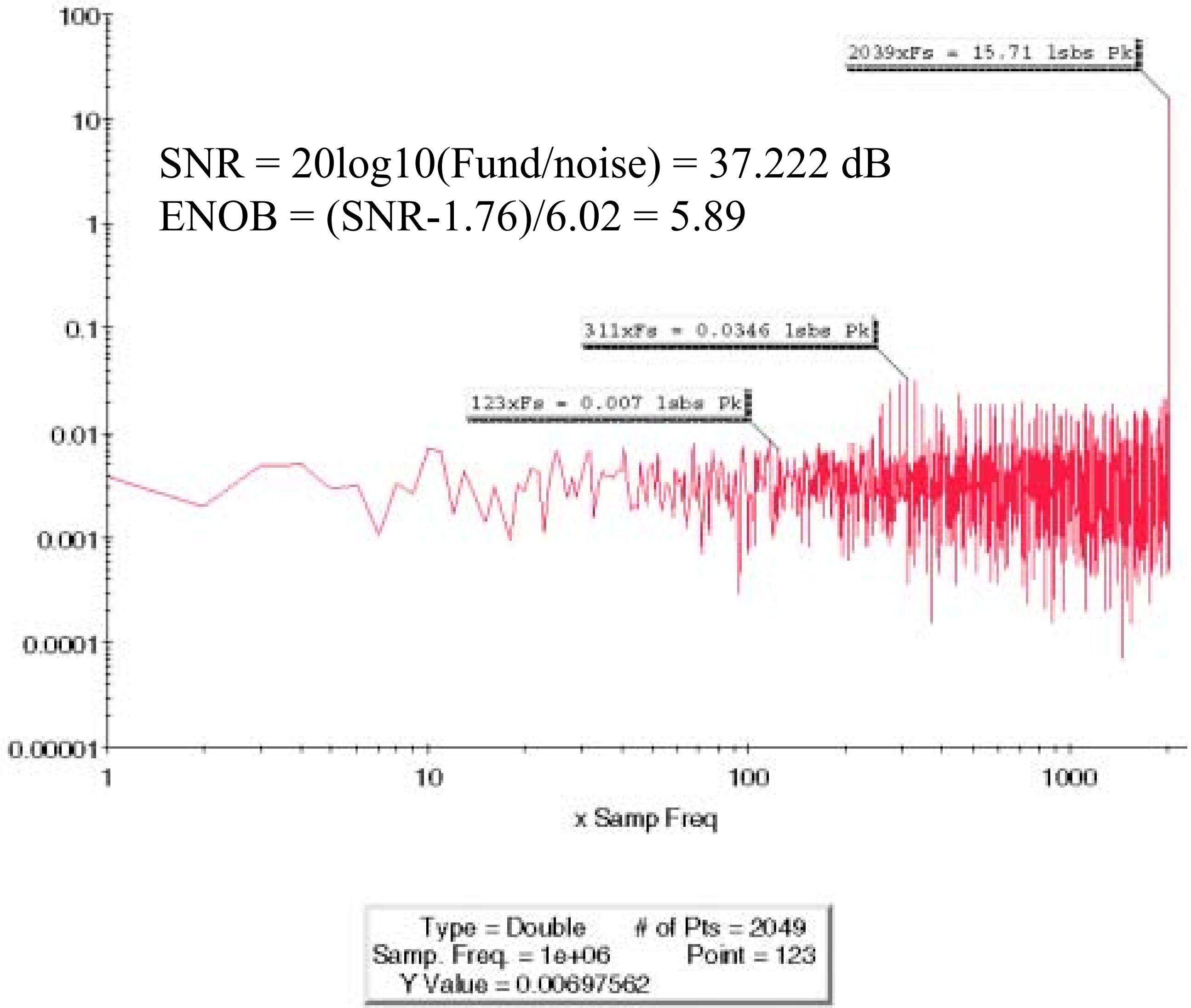
S4=sqrt(S3)

Fund = Peak(S4)= sqrt(S3[2039])

Noise = Sqrt(Sum(S3[0:2038, 2040:2049))

SNR = 20log10(Fund/noise) = 37.222 dB

ENOB = (SNR-1.76)/6.02 = 5.89



Signal Sink – Formatted for SPW

```
initial  begin
   //datestring function was here
   filestring = {libpath,"/",SignalName, datestring,".", viewname};
   fileid = $fopen(filestring);
   $fstrobe(fileid, "$SIGNAL_FILE 9" );
   $fstrobe(fileid, "$USER_COMMENT" );
   $fstrobe(fileid, "Output Data File for ADC %M");
   $fstrobe(fileid, "$COMMON_INFO");
   $fstrobe(fileid, "SPW Version        = 4.81");
   $fstrobe(fileid, "System Type        = solaris2");
   $fstrobe(fileid, "Sampling Frequency = %d",SampRate);
   $fstrobe(fileid, "Starting Time      = 0");
   $fstrobe(fileid, "$DATA_INFO");
   $fstrobe(fileid, "Number of points   = %d", NumPoints);
   $fstrobe(fileid, "Signal Type        = Integer");
   $fstrobe(fileid, "$DATA ASCII");
   $timeformat(-9,," ns",20);
   count = 0;
end
```

Signal Sink – Formatted for SPW

```
always @(posedge clk) begin
   if (((^data) !== 1'bx) && ((^data) !== 1'bz)) decimal_value = data;
   if (enable&& (count<NumPoints)) begin
      $fstrobe(fileid, "%d # %t ", data, $realtime);
      count = count +1;
   end
end
//------------------------------------------------------------
analog begin
   @(initial_step) begin
      outfile = $fopen("SampleInfo%I.%M.%T.dat");
      $fstrobe(outfile, "# Output Data File for ADC %M");
      $fstrobe(outfile, "# Time                    Sample");
      $fstrobe(outfile, "%20.15e  %d",$abstime, decimal_value);
   end
   @(posedge clk) begin
      if (enable) $fstrobe(outfile, "%20.15e %d",
          $abstime, decimal_value);
   end
   @(final_step) $fclose(outfile);
end
```

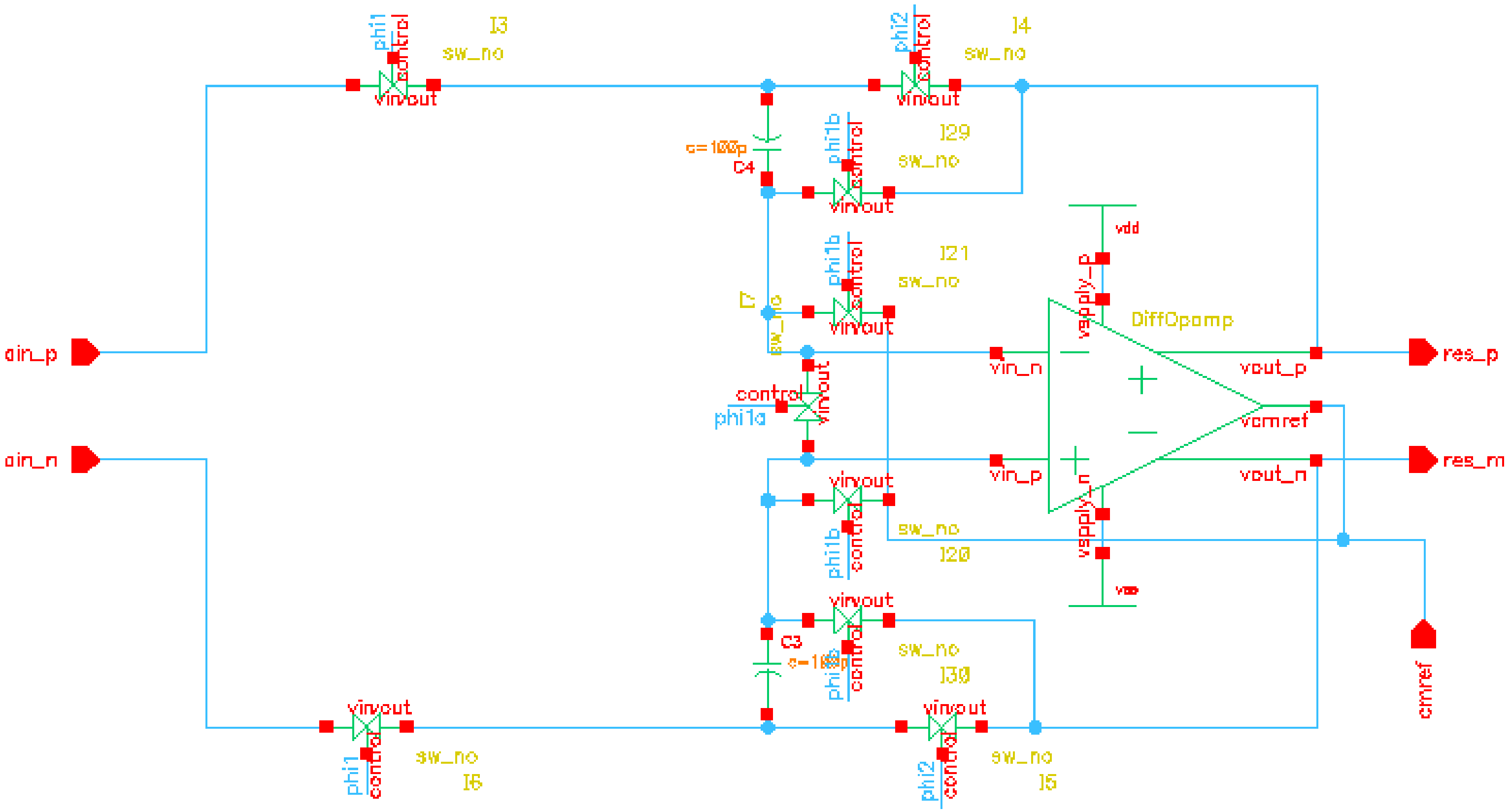
Second Order Model

- Offsets/GainErrors that match Arch.
- Can expand First Order Model
- Or use first Order Models of SubBlocks

Sub-Block Models

- SAH
- 1.5b Pipeline Stage
- 2 bit Flash
- Digital correction Block

SAH



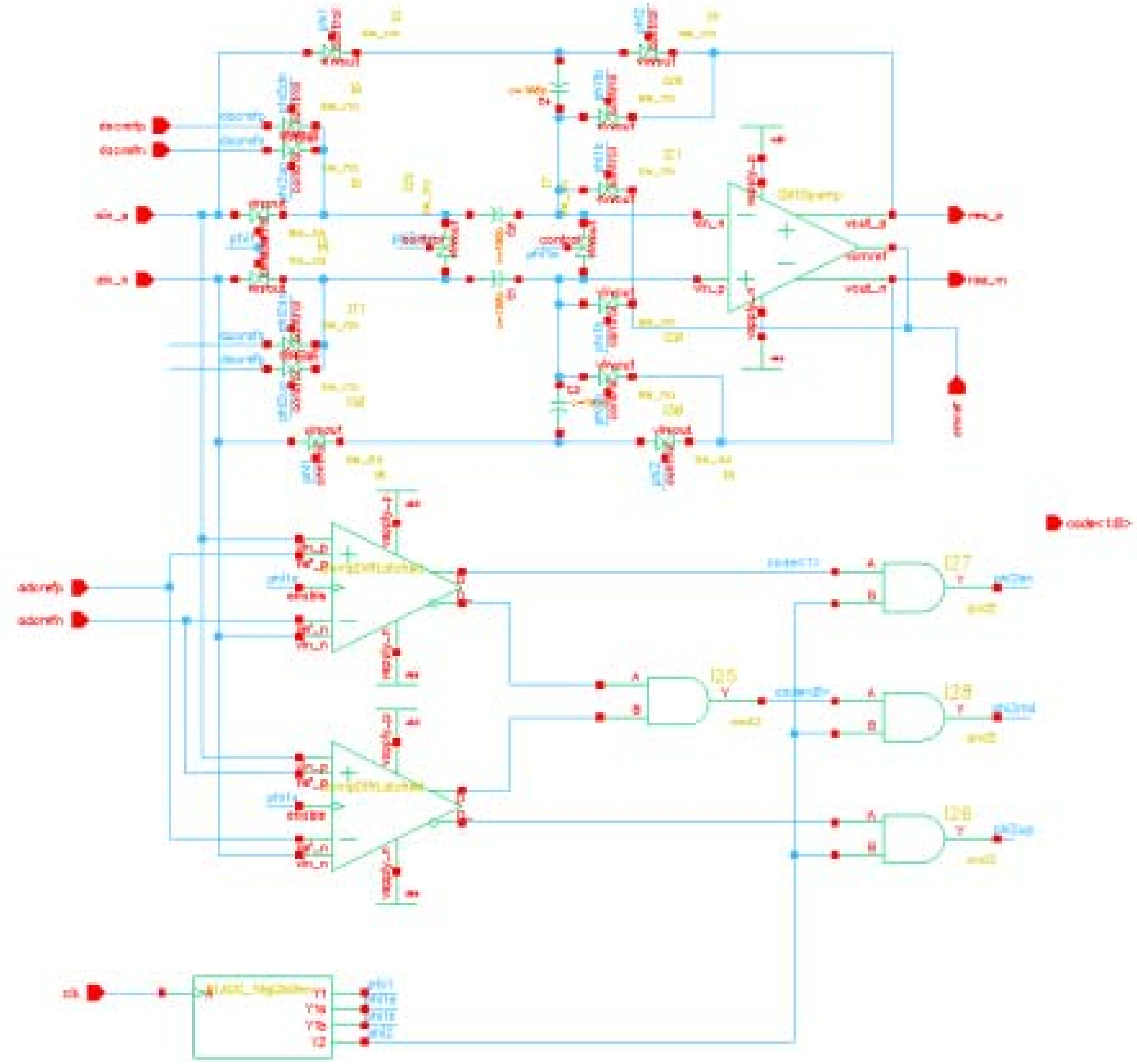
ain_p
ain_n
res_p
res_m
cmref
I3
I4
I5
I6
I7
I20
I21
I29
I30
sw_no
phi1
phi2
phi1a
phi1b
control
vin
vout
C3
C4
c=100p
DiffOpamp
vdd
vsupply_p
vsupply_n
vin_n
vin_p
vout_p
vout_n
vcmref

SampHold.vams

```
real posval, negval;
// Analog Values in Discrete time events
wreal res_p = posval;   wreal res_m = negval;
 // initialize the variables in Initial Block
 always @(negedge clk) begin
    sampval = V(ain_p, ain_n);
    vn   = vnoise * ($dist_normal(seedn,0,1000)%6000) / 1000.0;
    posval = V(cmref) + gain*(sampval+vos+vn)/2;
    if (posval > vhi) posval = vhi;
    else if (posval < vlo ) posval = vlo;
    negval = V(cmref) - gain*(sampval+vos+vn)/2;
    if (negval > vhi) negval = vhi;
    else if (negval < vlo ) negval = vlo;
 end
 always @(posedge clk) begin
    sampval = 0;
    posval = V(cmref);
    negval = V(cmref);
 end
```

- **Models Fixed Gain and Offset errors + Gaussian Noise @ Sample rate**

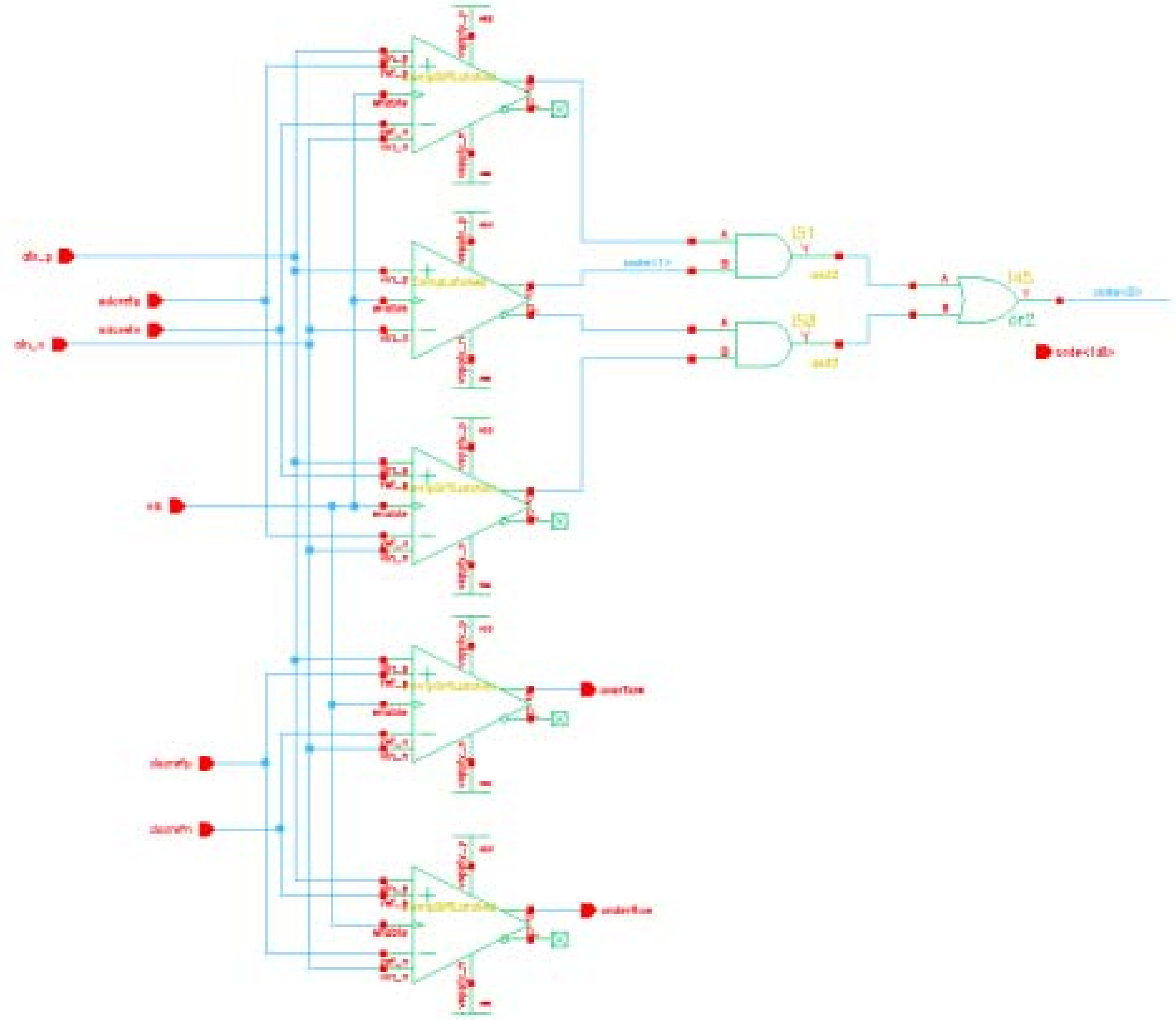
1.5b Pipeline Stage



PLADC_1r5b_stage.vams

```
real res_pl, res_mi; wreal res_p, res_m; wreal ain_p, ain_n;
assign res_p = res_pl;   assign res_m = res_mi;
always @(posedge clk) begin // sample the input
   #(td/1n) code = 2'bx;  // set to unknown until other edge of clock
   res_pl = V(cmref);
   res_mi = V(cmref);
end
always @(negedge clk) begin // evaluate and drive the outputs
   vn   = vnoise * ($dist_normal(seedn,0,1000)%6000) / 1000.0;
   valin = ain_p - ain_n;
   refin = V(adcrefp, adcrefn);
   #(td/1n) code = 1+((valin+vospcomp)>refin)-((valin+vosncomp)<-refin);
   #(td/1n) resout = (valin+vosamp+vn)*Cgain+ (1.0-
                                         code)*V(dacrefp,dacrefn);
   res_pl = V(cmref)+0.5*resout;
   res_mi = V(cmref)-0.5*resout;
end
```

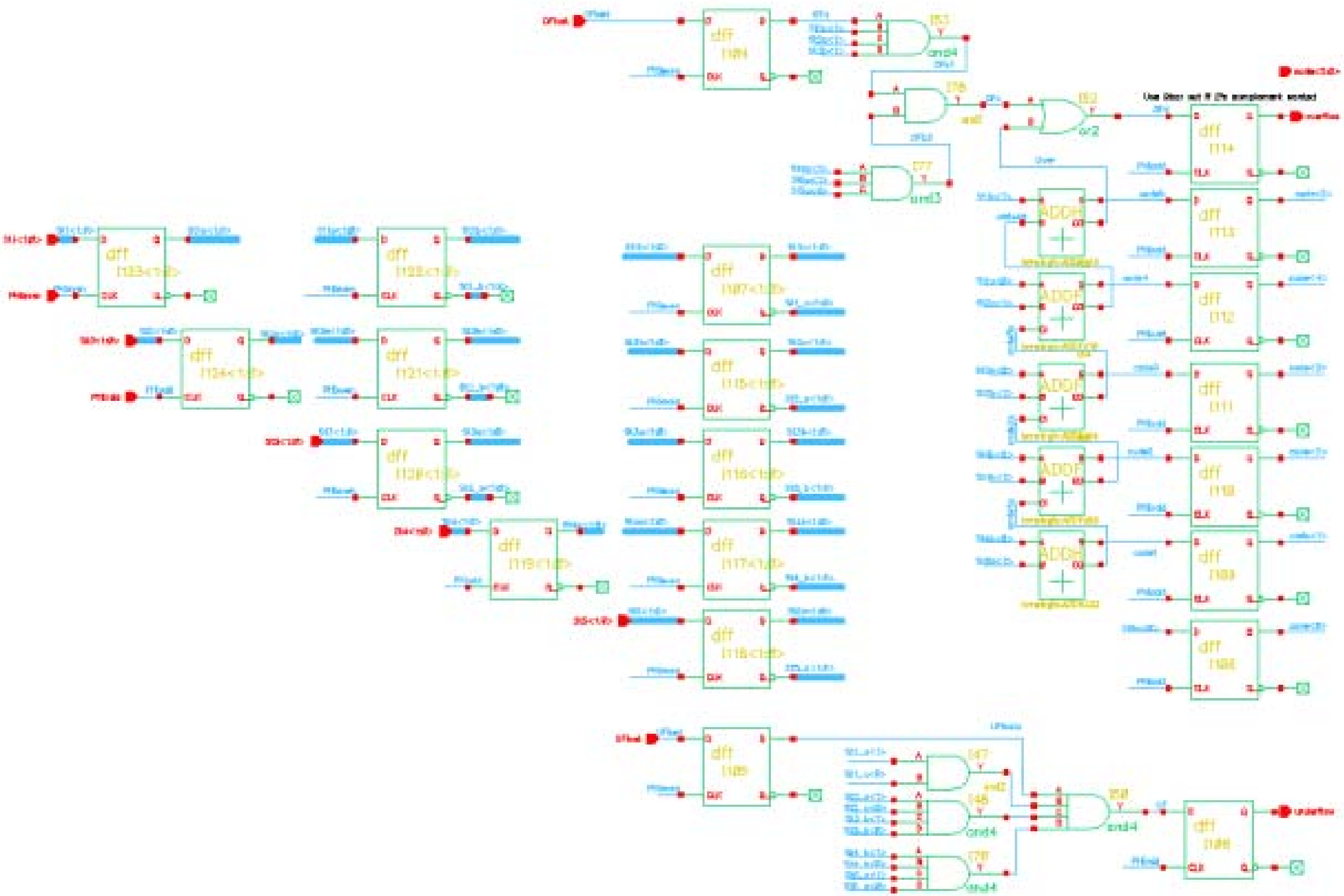
Flash Stage



PLADC_flash.vams

```
wreal ain_p, ain_n;
logic clk;
output overflow, underflow; reg overflow, underflow;
output  [1:0] code; reg [1:0] code;
always @(posedge clk) begin // sample the input
   //valin = ain_p - ain_n;
   //refin = V(adcrefp, adcrefn);
   #(td/1n) code = 2'bx;  // set to unknown
end
always @(negedge clk) begin // evaluate and drive the outputs
   valin = ain_p - ain_n;
   refin = V(adcrefp, adcrefn);
   #(td/1n) code = 1+(valin>0)+(valin>refin)-(valin<-refin);
   overflow = valin>V(dacrefp,dacrefn);
   underflow = valin<-V(dacrefp,dacrefn);
end
```

Digital Correction Block



PLADC_DigCorr4.vams

```
always @(posedge PHIodd) begin
   #Td // just the even bits
   St2a = St2;   St4a = St4;   code = Sum;
   overflow = Over || (OF &&
      St1c[1] && St2c[1] && St3b[1] && St4b[1] && St5a[1] &&
 St5a[0]);
   underflow = UF && !St1c && !St2c && !St3b && !St4b && !St5a ;
end
always @(posedge PHIeven) begin
   #Td // need to do this in order,
       // or use non-blocking with the same delay?
   St1c = St1b;    St1b = St1a;    St1a = St1; //1
   St2c = St2b;    St2b = St2a;                //2
   St3b = St3a;    St3a = St3;                 //3
   St4b = St4a;    St5a = St5;                 //4 & 5
   Sum = St5a + (St4b<<1) + (St3b<<2) + (St2c<<3) + (St1c<<4);
   // these are clocked and EVALUATED on other Edge
   Over = Sum[6];    OF = OFlast;    UF = UFlast;
end
```

Third Order Models – Behavioral Models of Analog Building Blocks

- Switch
- Opamp
- Comparator
- Clock Generation

- 2nd order models allow allocation of gain error, offset and noise specs
- Simulation Is FASTER than electrical but slower than Matlab. WHY repeat this?
- Confirm Matlab conclusions in a TEST BENCH compatible with Extracted simulation
- Other Specs (loading driving, Non ideal Opamp vs Cap Mismatch) cannot be separated out at 2nd order level.

Sw_no.vams

cadence

```
// log Cubic Spline Transition
analog function real lcubefn;
   input x,K; real x,K;
   lcubefn = (x<=0)?1:(x>=1)?K: pow(K,(3-2*x)*x*x);
endfunction
initial Control = 0;
always @(posedge control) Control = 1;
always @(negedge control) Control = 0;
analog begin
   @(initial_step) begin
      if (Control == 1) swres = 0.0; // on means R is minimum
      else swres = 1.0;             // off means R is maximum
   end
   @(posedge Control) swres = 0.0; // on means R is minimum
   @(negedge Control) swres = 1.0; // off means R is maximum
   // RoutExponent calculated from a transistion function
   rsmooth = transition(swres, tdelay,trise,tfall);
   rout = ron*lcubefn(rsmooth,roff/ron);
   V(vin,vout) <+ I(vin,vout)*rout;
end
```

DiffOpamp.va – Start with ModelWriter, Add vcm, outn

cadence

```
analog begin
   @(initial_step) begin // by default ALL analyses included (446+)
      r1 = gain; gm_nom = 1.0;
      c1 = 1/(`M_TWO_PI * pole_freq * gain);  r_rout = rout;
   end
   vin_val=  V(vin_p, vin_n) + vin_offset;
   // ------ Vref  is at Virtual Ground
   V(vref, vspply_n) <+ 0.5*V(vspply_p,vspply_n);
   // ------ Input Stage
   I(vin_p, vin_n) <+ vin_val / rin;
   I(vref, vin_p)  <+ ibias;  I(vref, vin_n)  <+ ibias;
   // ------ GM stage
   I(vref, coutp) <+ gm_nom*vin_val ;  I(vref, coutn) <+ -gm_nom*vin_val ;
   // ------ Dominant Pole.
   I(coutp, vref) <+ 2*c1*ddt(V(coutp, vref));    I(coutp, vref) <+ 2*V(coutp, vref)/r1;
   I(coutn, vref) <+ 2*c1*ddt(V(coutn, vref));    I(coutn, vref) <+ 2*V(coutn, vref)/r1;
   // ------ Output Stage.
   I(vref, vout_p) <+ 2*V(coutp, vref)/r_rout;
   I(vout_p, vref) <+ 2*V(vout_p, vref)/r_rout;
   I(vref, vout_n) <+ 2*V(coutn, vref)/r_rout;
   I(vout_n, vref) <+ 2*V(vout_n, vref)/r_rout;
   // ------ Soft Output Limiting.
   if (V(vout_p) > (V(vspply_p) - vsoft))      I(coutp, vref) <+ gm_nom*(V(vout_p, vspply_p)+vsoft);
   else if (V(vout_p) < (V(vspply_n) + vsoft)) I(coutp, vref) <+ gm_nom*(V(vout_p, vspply_n)-vsoft);
   // ------ Soft Output Limiting.
   if (V(vout_n) > (V(vspply_p) - vsoft))      I(coutn, vref) <+ gm_nom*(V(vout_n, vspply_p)+vsoft);
   else if (V(vout_n) < (V(vspply_n) + vsoft)) I(coutn, vref) <+ gm_nom*(V(vout_n, vspply_n)-vsoft);
end
```

CompDiffLatched.vams

```
reg d, D;
assign D_ = !D;
analog begin
   @(initial_step)  begin
     halfhys =  hys/2.0;
     Tplh = (td + trise/2)/1n; Tphl = (td + tfall/2)/1n;
   end
   vin =V(vin_p,vin_n) - V(ref_p,ref_n) + p_off + n_off;
end
initial begin
   TPlh = 1; TPhl = 1; // value will be corrected soon
   #0.1 TPlh = Tplh; TPhl = Tphl; // until analog initial_step
   d = vin>0; // initialize the register
end
always @(cross(vin - halfhys, +1 ))  if (enable) d = 1;
always @(cross(vin + halfhys, -1 ))  if (enable) d = 0;
always @(posedge enable) begin
   if ((vin < -halfhys)&&(d)) d = 0;
   else if ((vin > halfhys)&&(!d)) d = 1;
end
always @(posedge d) #TPlh D = d;
always @(negedge d) #TPhl D = d;
```

CompLatched.vams is easier!

PLADC_StgClkGen.vams

```
initial begin
   Y2 = 1;
   Y1 = 0;
   Y1a = 0;
   Y1b = 0;
end
always @(posedge A) begin
   #Tdhl2 Y2 = !A;
   #Tdlh1 Y1 = A;
   #Tdlh1a Y1a = A;
   #Tdlh1b Y1b = A;
end
always @(negedge A) begin
   #Tdhl1b Y1b = A;
   #Tdhl1a Y1a = A;
   #Tdhl1 Y1 = A;
   #Tdlh2 Y2 = !A;
end
```

Characterizing The SubBlocks

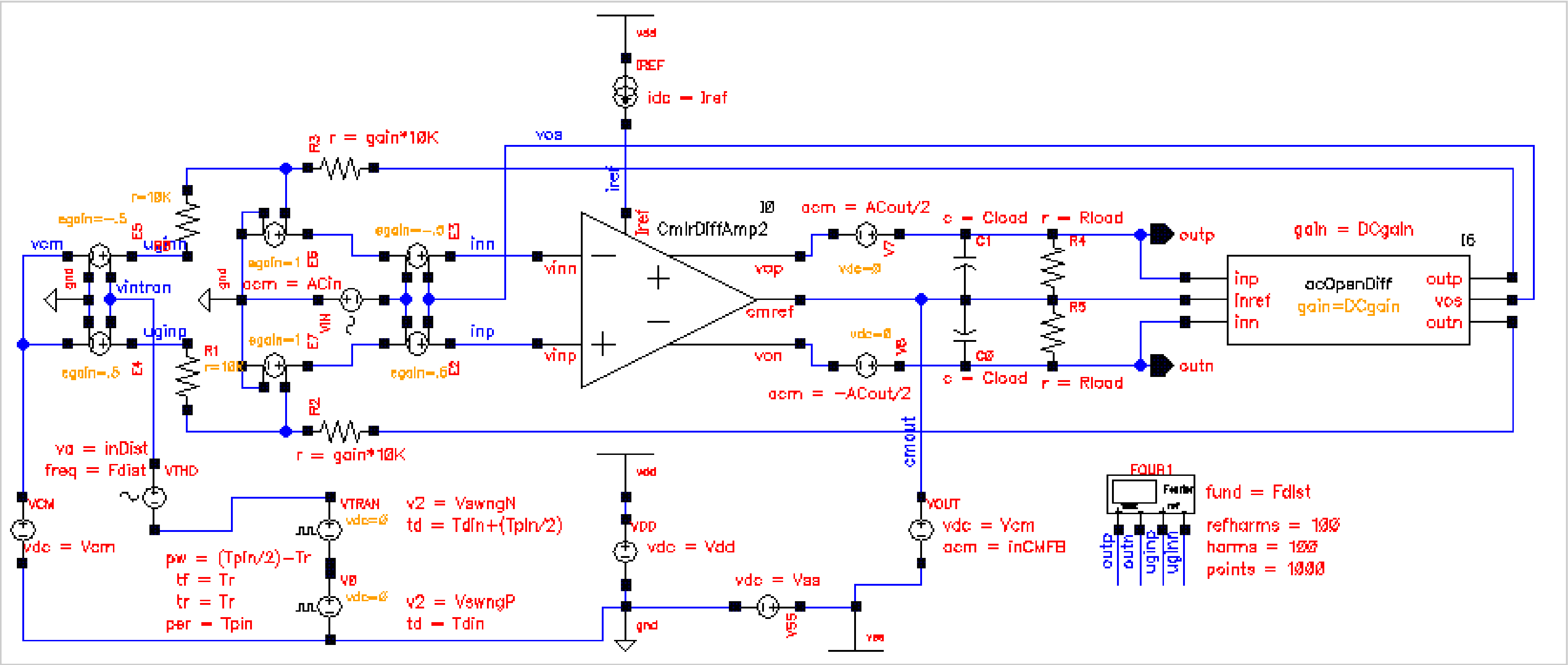
- Gain Errors
- Offsets
- Noise

- A Design Reuse Method will aid this.
- Scripts need to create datafiles accessed by higher level models
- New file access functions allow easier use of data between various simulations

Analog Block Characterization

- OCEAN scripting – Gives ADE users a Batch capability
  - Script can write file to be read or **included** in model
- Aptivia (Virtuoso Specification Driven Environment)
  - Open DCM allows you to define
    - Configurable model with TABLE data
    - test bench and measurements
    - extract data from corner sweeps
  - Apply to variety of similar circuits
- Verilog-A models can also help with measurements
- Consider Design Reuse
  - Build datasheets with scripts after ECO changes.

Variables are key to TB Flexibility



Summary

- Top Down Verification Methodology
- 1st order, Pin Accurate model defined
- Specifications and models to support those were defined
- 2nd Order models were developed, and assembled for more detail. Retested and compared to original model
- 3rd Order model was developed from 1st order model of analog building blocks. – Additional Specifications need to be Allocated to make that part work again.

Questions?

References

- J. Doenberg, HS Lee, DA Hodges, “**Full-Speed Testing of A/D Converters**” IEEE Journal of Solid-State Circuits(1984), Vol. 19, No. 6, Dec. 1984, 820-827
- TE Linnenbrink, SJ Tilden, MT Miller, “**ADC Testing with IEEE-Std 1241-2000**” Proceedings IEEE Instrumentation and Measurement Conference 2001, 1986-1991
- G. Chiorboli, C. Morandi, “**ADC Modeling and Testing**” Proceedings IEEE Instrumentation and Measurement Conference 2001, 1992-1999
- **Nyquist data converter testing and yield analysis using behavioral simulation**
  Liu, E.W.Y.; Sangiovanni-Vincentelli, A.L.
  Computer-Aided Design, 1993. ICCAD-93. Digest of Technical Papers., 1993 IEEE/ACM International Conference on , 1993 Page(s): 341 -348
- **Influence of the architecture on ADC error modeling**
  Arpaia, P.; Daponte, P.; Michaeli, L.
  Instrumentation and Measurement, IEEE Transactions on , Vol.48, Iss.5, 1999
  Pages: 956- 966
- **Metrological characterisation of analog-to-digital converters-a state of the art**
  Arpaia, P.; Cennamo, F.; Daponte, P.
  Advanced A/D and D/A Conversion Techniques and Their Applications, 1999. Third International Conference on (Conf. Publ. No. 466) , 1999
  Page(s): 134 -144
- **Some thoughts on sine wave ADC testing**
  Sugawara, H.; Kobayashi, H.; Arpaia, P.
  Instrumentation and Measurement Technology Conference, 2000. IMTC 2000. Proceedings of the 17th IEEE , 2000
  Page(s): 125 -130 vol.1
- J. David, “**Functional Verification Of A Differential Operational Amplifier**” International Cadence User Group, 2001

More References

- **Behavioral simulation of a 3-bit flash ADC**
  Mantooth, H.A.; Allen, P.E.
  Circuits and Systems, 1990., IEEE International Symposium on , 1990
  Page(s): 1356 -1359 vol.2
- **Behavioral model of pipeline ADC by using SIMULINK(R)**
  Bilhan, E.; Estrada-Gutierrez, P.C.; Valero-Lopez, A.Y.; Maloberti, F.
  Mixed-Signal Design, 2001. SSMSD. 2001 Southwest Symposium on , 2001 Page(s): 147 -151
- **Behavioral modeling and simulation of data converters**
  Liu, E.; Gielen, G.; Chang, H.; Sangiovanni-Vincentelli, A.L.; Gray, P.R.
  Circuits and Systems, 1992. ISCAS '92. Proceedings., 1992 IEEE International Symposium on , Volume: 5 , 1992
  Page(s): 2144 -2147 vol.5
- **IEEE Std 1241: the benefits and risks of ADC histogram testing**
  Max, S.
  Instrumentation and Measurement Technology Conference, 2001. IMTC 2001. Proceedings of the 18th IEEE , Volume: 1 , 2001
  Page(s): 704 -709 vol.1
- **Fast accurate and complete ADC testing**
  Max, S.
  Test Conference, 1989. Proceedings. Meeting the Tests of Time., International , 1989
  Page(s): 111 -117

Bibliography

Don Lewis “Testing Operational Amplifiers” Electronics Test (Benwill Publishing) January 1979

Don Lewis “Compensation of Linear IC Test Loops” Electronics Test (Benwill Publishing) May 1979

David Johns, Ken Martin,”Analog Integrated Circuit Design”, Wiley & Sons, New York 1997, esp chapter 6.

Jacob Millman, “MicroElectronics: Digital and Analog Circuits and Systems” McGraw-Hill 1979

Paul Gray and Robert Meyer “Analysis and Design of Analog Integrated Circuits: 2nd Ed” Wiley & Sons 1984

C.F. Wojslaw & E.A. Moustakas “Operational Amplifiers” Wiley & Sons, NY, 1986

Dan Fitzpatrickj, Ira Miller, “Analog Behavioral Modeling with the Verilog-A Language” Kluwer, Boston, 1998

Samir Palnitkar, “ Verilog HDL” SunSoft, Mountain View, CA 1996

Ken Kundert, “The Designer’s Guide to Spice & Spectre”, Kluwer, Boston, 1995

J.E. Solomon. "The monolithic op amp: a tutorial study." IEEE Journal of Solid-State Circuits(1974) SC-9.6 (Dec. 1974 (Special Issue on Analog Circuits)): 314-332. (Also published as Application Note AN-A from National Semiconductor)

G.Ferri and W. Sansen “A Rail-toRail Constant-g_m Low-Voltage CMOS Operational Transconductance Amplifier.” IEEE Journal of Solid-State Circuits(1997), vol 32, October 1997 1563-1567.

M. Yamatke, “A Simplified Test-Set for Op Amp Characterization” National Semiconductor Application Note 24, April 1986

Bibliography (Cont'd)

K. Bult& G.J.G.M. Geelen, "A Fast-Settling CMOS Op Amp for SC Circuits with 90-dB DC gain. IEEE Journal of Solid-State Circuits(1990), Vol. 25, No. 6, Dec. 1990, 1379-1384

T.C. Choi, R.T. Kaneshiro, R.W. Brodersen, P.R. Gray W.B. Jett & M. Wilcox, "High-Frequency CMOS Switched Capacitor Filters for Communications Application" IEEE Journal of Solid-State Circuits(1983), SC-18 (December 1983) 652-663

M. Banu, J.M. Khoury & Y. Tsividis "Fully Differential Operational Amplifiers with Accurate Output Balancing." IEEE Journal of Solid-State Circuits(1988), Vol. 23, No. 6, Dec. 1988 1410-1414

P.E. Allen and D.R. Holberg, "CMOS Analog Circuit Design" Oxford Univ. Press, New York 1987

K.R. Laker & W.M.C. Sansen, "Design of Analog Integrated Circuits and Systems" McGraw-Hill, New York 1994