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 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
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
- Supplies and Biasing
- Clocking & Timing
- Output Drive
- Linearity
- Noise & Distortion

6b 1.5b/stage 4 stage Pipeline ADC

- 3v supplies
- Differential Signal path .25 – 2.25v
- Vcm = 1.25
- Digital Error Correction
Top Down Modeling and Test Bench
Development: A Verification Case
Study of a Pipeline ADC

Trap to Avoid

Early Simulation Results aren't the same thing as a Top-Down Verification program!

First Order Model

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

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

always @(posedge sclk) begin
  // add the noise
  vn = vnoise * ($dist_normal(seedn,0,1000)/6000) / 1000.0;
  vnr = vref * ($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

```verbatim
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
```

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
Top Down Modeling and Test Bench
Development: A Verification Case
Study of a Pipeline ADC

Test bench Schematic

Compare Block Details

- **Instantaneous Comparison (SimVision Compare Function is Better)**
  
  assign overCMP = overDUT*overGLD;
  assign underCMP = underDUT*underGLD;
  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) %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
                $fstrobe(fileid, "SPECWARN: %t ADC DUT(%d) != ADC GLD(%d)",
                    $realtime, CMP, DUT, GLD);
                $fstrobe(fileid, "SPECWARN: %t ADC DUT(%d) != ADC GLD(%d)",
                    $realtime, CMP, DUT, GLD);
            end else begin // specfail will also fail on any X values
                $fstrobe("SPECFAIL: %t ADC code Delta=%d > spec=%d",$
                    $realtime, CMP, TolLsb);
                $fstrobe(fileid, "SPECFAIL: %t ADC code Delta=%d > spec=%d",$
                    $realtime, CMP, TolLsb);
            end
        end
    end
end

Wreal Model without Noise or Non Linears

Functional Results – Ideal vs Wreal model

2002 IEEE International Workshop on
Behavioral Modeling and Simulation
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
Top Down Modeling and Test Bench
Development: A Verification Case
Study of a Pipeline ADC

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

Count Each Code

always @(posedge clk) begin
    if (counting && (^data) !== 1'bx) && (^data) !== 1'bz)
        && (under && lover ) begin
        bins[data] = bins[data] +1;
        // debug !!
        bintest = bins[data];
        // debug !!
        totcount = totcount+1;
        sum = sum + data;
    end
end
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
      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 (dnlfile, "max abs dnl: %10.3g", dnlmax);
    $fstrobe (inlfile, "max abs inl: %10.3g", inlmax);
    $fstrobe (filename, "max abs dnl: %10.3g", dnlmax);
    $fstrobe (filename, "max abs inl: %10.3g", inlmax);
  end
end
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

\[ S2 = \text{ fft\_transform}(S1, \text{ fftlen}=4096, \text{ num\_frames}=1, \text{ overlap}=0, \text{ window}=\text{rectangular}, \text{ rm\_mean}, \text{ half\_spectrum}, \text{ normalize}) \]

\[ S3 = \text{ abs}(\text{conjugate}(S2) \ast S2) \]

\[ S4 = \sqrt{S3} \]

\[ \text{Fund} = \text{Peak}(S4) = \sqrt{\text{Sum}(S3)} \]

\[ \text{Noise} = \sqrt{\text{Sum}(S3[0:2038, 2040:2049])} \]

\[ \text{SNR} = 20\log_{10}(\text{Fund}/\text{noise}) = 37.222 \text{ dB} \]

\[ \text{ENOB} = (\text{SNR}-1.76)/6.02 = 5.89 \]

Signal Sink – Formatted for SPW

```plaintext
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

```
cadence
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

// resurrected version of original program

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) $fclose(outfile);
end
```
real posval, negval;
// Analog Values in Discrete time events
wreal res_p = posval;  wreal res_m = negval;
// initialize the variables in Initial Block
always @(negedge clk) begin
    sampval = V(ain_p, ain_n);
    vn = vnoise * ($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

Transfer Function

Residue

Vin

00 01 10

00 100

10 01

100 100
PLADC_1r5b_stage.vams

```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'hx; // 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+vospamp+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
Top Down Modeling and Test Bench Development: A Verification Case Study of a Pipeline ADC

PLADC_flash.vams

```verilog
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;
    //relin = 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/ln) code = 1+(valin>0)+(valin>relin)-(valin<-relin);
    overflow = valin>V(dacrefp,dacrefn);
    underflow = valin<-V(dacrefp,dacrefn);
end
```

Digital Correction Block
Top Down Modeling and Test Bench Development: A Verification Case Study of a Pipeline ADC

**PLADC_DigCorr4.vams**

```vhdl
always @(posedge PHIodd) begin
    St2a = St2;  St4a = St4;  code = Sum;
    overflow = Over || (OF &&
        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

// 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
  rout = ron*lcubefn(rsmooth, roff/ron );
  V(v(in,vout)) <+ I(v(in,vout))*rout;
end

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

analog begin
  @(initial_step) begin // by default ALL analyses included (446+)
    r1 = gain; gm_nom = 1.0;
    r1 = 1/[M_TWO_PI * pole_freq * gain];  r_rout = rout;
  end
  vin_val=  V(vin_p, vin_n) + vin_offset;
  // ------ 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

```verilog
reg d, D;
assign D_ = !D;

analog begin
    @(initial_step) begin
        halfhys = hys/2.0;
        Tplh = (td + trise/2)/ln; Tphl = (td + tfall/2)/ln;
    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

```verilog
initial begin
    Y2 = 1;
    Y1 = 0;
    Y1a = 0;
    Y1b = 0;
end

always @(posedge A) begin
    #Tdhl2 Y2 = !A;
    #Tdih1 Y1 = A;
    #Tdihla Y1a = A;
    #Tdihlb Y1b = A;
end

always @(negedge A) begin
    #Tdhlb Y1b = A;
    #Tdihla Y1a = A;
    #Tdih1 Y1 = A;
    #Tdhl2 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

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**

- Nyquist data converter testing and yield analysis using behavioral simulation
  Liu, E.W.Y.; Sangiovanni-Vincentelli, A.L.
- Influence of the architecture on ADC error modeling
  Arpaia, P.; Daponte, P.; Michaeli, L.
- Metrological characterisation of analog-to-digital converters-a state of the art
  Arpaia, P.; Cennamo, F.; Daponte, P.
- Some thoughts on sine wave ADC testing
  Sugawara, H.; Kobayashi, H.; Arpaia, P.
More References

- Behavioral simulation of a 3-bit flash ADC
  Mantooth, H.A.; Allen, P.E.
  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.
  Page(s): 147 -151

- Behavioral modeling and simulation of data converters
  Liu, E.; Gielen, G.; Chang, H.; Sangiovanni-Vincentelli, A.L.; Gray, P.R.
  Volume: 5, 1992
  Page(s): 2144 -2147 vol.5

- IEEE Std 1241: the benefits and risks of ADC histogram testing
  Max, S.
  Volume: 1, 2001
  Page(s): 764 -769 vol.1

- Fast accurate and complete ADC testing
  Max, S.
  Page(s): 111 -117

- FFT Testing of ADCs
  Odom, Bill
  Conference on Analog and Mixed-Signal Applications, 1998 Proceedings
  Page(s): 201-203

- New High Speed Technique for Pipeline ADC Design
  Sarraj, Maher
  Conference on Analog and Mixed-Signal Applications, 1998 Proceedings
  Page(s): 205-208

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
Samir Pahlkhar, "Verilog HDL" SunSoft, Mountain View, CA 1996
(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-gm, Low-Voltage CMOS Operational Transconductance Amplifier.” IEEE
M. Yamatke, “A Simplified Test-Set for Op Amp Characterization” National Semiconductor Application Note 24, April
1986