Verification of CML circuits used in PLL contexts with Verilog-AMS

Jonathan David
Scintera Networks, Inc.
4340 Stevens Creek Blvd, Suite 260
San Jose, CA 95129
j.david@ieee.org
Senior Member IEEE

ABSTRACT
CML (Current-Mode Logic) circuits are used in very high-speed applications where standard CMOS gates will not perform. Applications of CML logic are introduced, focusing on the Clock Divider for PLL's operating above 1-2GHz common in communication circuits. An overview of CML circuits and their operation is provided. Starting with a Verilog-A model of a basic gate, the development of connect elements and rules are explained, and applied to the resulting mixed signal model of the gate. Finally the application of this circuit to the PLL and the resulting simulation performance improvements presented.

1. INTRODUCTION
While CMOS logic enabled the ASIC and System-on-Chip revolutions, Current Mode Logic (CML) continues to be useful in the highest speed applications. Whether implemented in CMOS, Bipolar, Bi-CMOS or SiGe-Cmos, the gate and logic design is relatively straightforward. Buffers and receivers, especially for longer lines, are more challenging, frequently requiring a departure from the pure "logic" of the design, to include pre-emphasis for drivers and analog equalization, and even gain control for receivers. Differential signaling offers the speed for the logic, and the signal headroom where the signal is more analog in nature.

CAD tool support for a synthesis, place and route type flow in this area doesn't exist yet, in part due to the tool provider's focus on the high-volume CMOS design flow, and in part because of the significant "hand-crafted" design required in highest speed circuits. On the simulation side, rather than a separate focus on verification and performance (i.e. timing, signal integrity) circuit-level simulation remains the dominant methodology, despite the impact to simulation time at the top level. This effectively limits the size of circuits that can be designed to the size of circuits that can be simulated in a reasonable amount of time.

A somewhat extreme example of this limit occurs when a relatively simple (from a logic viewpoint) clock divider is used in the feedback loop for a PLL. Validation of PLL control signals, especially those affecting the loop parameters is a lengthy process if the VCO and Clock-dividers dominate the simulation time. By putting the VCO output and clock dividers into a logical domain, we can decrease the simulation time enough so that all the control signals can be validated.

Using Verilog-AMS[1,2,3,4,5] as the simulation language we show methods to increase that size by an order of magnitude over analog behavioral modeling. These methods will also aid in the analog-parts of these circuits.

But first we'll review the logic gates, a latch and then assemble these into a clock-divider. It should be noted that these designs are one way to design, and a significant variety design styles are in use for CML circuits.

1.1 CML AND/NAND GATE
The LATCH in Figure 1 has an output swing determined by the combination the resistor size and the bias current. Where higher drive capability is required, a 2x, (or, as shown, 4x) is created which has twice (4x) the tail current and half (1/4) the load resistance to keep the same output swing. The d_p d_n inputs accept the same common-mode and swing that this gate provides, while the clk_p clk_n inputs require a lower common mode, which leads us to another style output. Logic design libraries will typically include an AND/NAND, OR/NOT, XOR and MUX.

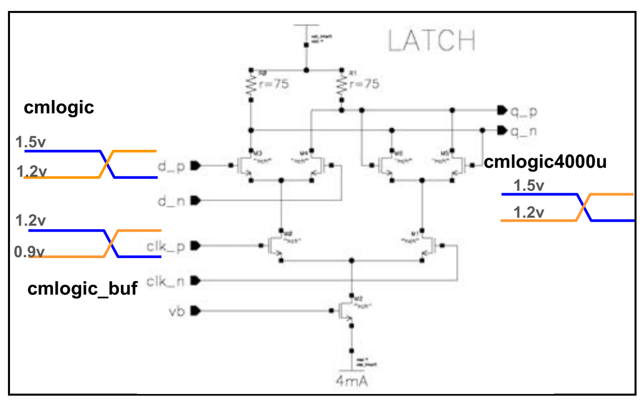

Figure 1 LAT_AX4 Schematic

For simplicity, we only consider resistive loads in these examples.

1.2 CML LEVEL-SHIFTER
Typically the inversion properties of this gate, shown in Figure 2 are not required in CML design, because inputs can be connected backwards to provide the inversion. The resistor RLS

provides the common-mode shift that is needed to drive the b inputs of AND2_AX1.

In cases where an overlap between the signal levels is needed, the common mode adjustment resistor, RLS, will not have the same value as the Line Termination resistors.

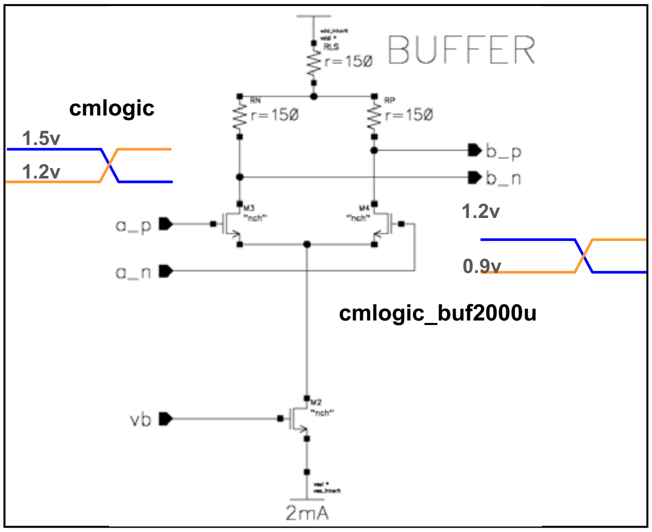

Figure 2 BUF_BX2 schematic

1.3 COMPOSITE LOGIC CIRCUIT

Our basic divider circuit uses two latches and a buffer to provide the level shift and drive increase for the clock.

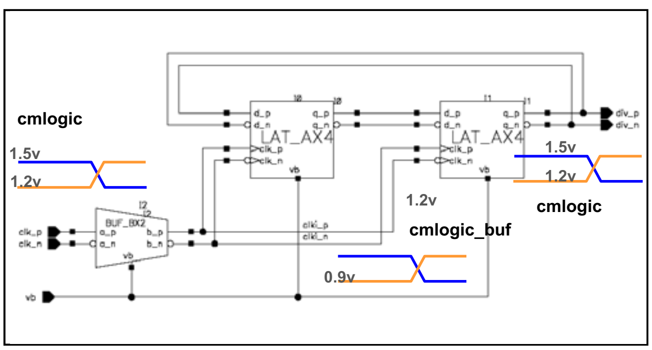

Figure 3 DIV2_AX4 Schematic

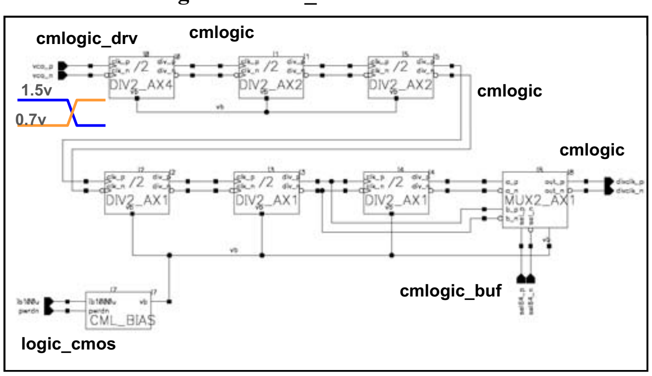

Figure 4 CLKDIV64 Schematic.

We use 6 of these and a mux for our divider so we can switch between 32 and 64 bit modes. The input Connect Discipline used reflects the fact that the VCO output has wider swing. The power-down control, driven by low voltage cmos, is implemented in the bias current block For Functional validation the behavioral gate models need to implement a power-down function based on the bias voltage so that power down control can be validated.

2. GATE MODELS

For Design Verification purposes, we need to validate that we are connecting inputs to compatible drivers, and that the supplies and biases provided are in the valid range. In addition to checking these, our model needs to provide the correct logic function, output swing, and load on the supply line. When running timing simulation with parasitic capacitance and resistance, we'll want the model to support analog inputs and outputs. We'll start with a Verilog-A version of the model, using the architecture shown in Figure 5, that meets these needs, and then transform it to an AMS model, which will support event driven simulation for speed in the functional flow, We complete the design by showing the connect modules and connect_rules, that are used in the Verilog-AMS models to support both timing and functional simulation flows.

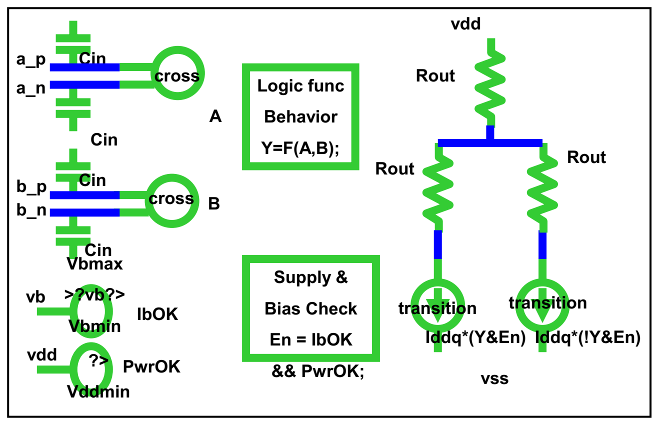

Figure 5 Architecture of CML gate model with level shifted output.

2.1 VERILOG-A MODELS

In addition to the behavior the model needs to support, we add a number of validation checks to this model. In a digital flow (written in system verilog or PSL) these could be kept separate from the model or circuit so that they can be applied (or not) to any model used, whether behavioral, rtl, or gate-netlist.

2.1.1 BUF_BX2.va

The model in Listing 1 implements the behavior of the gate in Figure 2. Note that if the supplies or bias are outside the specified ranges, the behavior is modified to show that the circuit is not operational. When the tail current is zero, both outputs are resistively pulled to the supply rail.

Listing 1 BUF_BX2.va : Level Shifter Buffer

```
// VerilogA for cml_xmpl, BUF_BX2, veriloga
`include "constants.vams"
`include "disciplines.vams"
```

```
module BUF_BX2( b_n, b_p,   a_n, a_p,   vb);
// REGISTER and WIRE TYPES
  output b_n, b_p;  electrical b_n, b_p;
  input a_n, a_p;  electrical a_n, a_p;
  input vb;  electrical vb;
// inherited supply connections
  electrical (* integer inh_conn_prop_name = "vdd";
    integer inh_conn_def_value = "cds_globals.\\vdd! "; *)  vdd;
  electrical (* integer inh_conn_prop_name = "vss";
    integer inh_conn_def_value = "cds_globals.\\vss! "; *)  vss;
  electrical vshift;
// PARAMETERS:  (Comment each one)

  parameter real Iddq = 1m; // 1m *1
  parameter real Rout = 300; // 300/1
  parameter real Cin = 15f;
  parameter integer outinit = 1 from [0:1];     // initial value of output
  parameter real Vddmin = 1.0;              //Vdd below which circuit is off
  parameter real Vbmin = 0.35;              // Vb below which circuit is off
  parameter real Vbmax = 0.5;               // Vb above which circuit doesn't work
  parameter real tp = 75p;              // delay from input clock to divided edge
  parameter real trf = 100p;              // risetime of 156Mhz output
  parameter real vtol = 50m;              // voltage tolerance on input
  parameter real ttol = 20p;             //a 1% of Fnom error in edge is pretty big.
  parameter real ttol_vb = 10n;          // time tolerance for vb check
// LOCAL VARIABLES:  (Comment each one)
  real Iout;
  integer Aval, Bval,  Pdown, Enabled;
// STRUCTURE
  resistor #(.r(Rout)) RLS (vdd, vshift);
  resistor #(.r(Rout)) R1p (vshift, b_p);
  resistor #(.r(Rout)) R1n (vshift, b_n);
  capacitor #(.c(Cin)) C1p (a_p, vss);
  capacitor #(.c(Cin)) C1n (a_n, vss);
// BEHAVIOR-----------------------------------------------------
  analog begin
    // re evalutate the logic when inputs change
    @((initial_step)
    or (cross(V(a_p,a_n),0,ttol,vtol)) ) begin
      // inputs
      Aval = V(a_p,a_n) > 0;
      // output
      Bval = Aval;
    end
    I(b_p,vss) <+ transition(!Bval&&Enabled?Iddq:0,tp,trf,trf);
    I(b_n,vss) <+ transition(Bval&&Enabled?Iddq:0,tp,trf,trf);
    // ASSERTIONS
    @(above(V( vb, vss )-Vbmax, ttol_vb )) begin  // don't want small time steps
         $strobe("ILGLCOND: %M Vbias > max = %g \n",Vbmax);
    end
    @(above(Vbmin-V( vb, vss ), ttol_vb )) begin  // don't want small time steps
        $strobe("STATINFO: %M Vbias < min - powering down = %g \n",Vbmin);
    end
    @(above(V( vb, vss )-Vbmin, ttol_vb )) begin  // don't want small time steps
        $strobe("STATINFO: %M Vbias > min - powering up = %g \n",Vbmin);
    end
    @(above(V( vdd, vss )-Vddmin, ttol_vb )) begin  // don't want small time steps
       $strobe("STATINFO: %M Vdd > min - powering up = %g \n",Vddmin);
    end
    @(above(Vddmin-V( vdd, vss ), ttol_vb )) begin  // don't want small time steps
       $strobe("STATINFO: %M Vdd < min - powering down = %g \n",Vddmin);
    end
    Pdown = (V(vdd,vss)<Vddmin)||(V(vb,vss)<Vbmin);
    Enabled = !Pdown;
  end
endmodule
```

2.1.2 LAT_AX4.va

The model in Listing 2 implements the behavior of the latch in Figure 1. Here, we have provided a parameter to determine the initial output value for the case where the initial condition is latched. This model implements the input capacitance and load resistors behaviorally rather than structurally.

Listing 2 LAT_AX4: Latch

```
// VerilogA for cml_xmpl, LAT_AX4, veriloga
`include "constants.vams"
`include "disciplines.vams"

//=======================================================
module LAT_AX4( q_n, q_p,   clk_n, clk_p,   d_n, d_p,   vb  );
// REGISTER and WIRE TYPES
  output q_n, q_p;        electrical q_n, q_p;
  input clk_n, clk_p;     electrical clk_n, clk_p;
  input d_n, d_p;         electrical d_n, d_p;
  input vb;               electrical vb;
// inherited supply connections
  electrical (* integer inh_conn_prop_name = "vdd";
    integer inh_conn_def_value = "cds_globals.\\vdd! "; *)  vdd;
  electrical (* integer inh_conn_prop_name = "vss";
    integer inh_conn_def_value = "cds_globals.\\vss! "; *)  vss;
// PARAMETERS:  (Comment each one)
  parameter real Iddq = 4m;                      // 1m *4
  parameter real Rout = 75;                      // 300/4
  parameter real Cin = 55f;
  parameter real Cin_clk = 55f;
  parameter integer outinit = 1 from [0:1];   // initial value of output
  parameter real Vddmin = 1.0;              //Vdd below which circuit is off
  parameter real Vbmin = 0.35;              // Vb below which circuit is off
  parameter real Vbmax = 0.5;            // Vb above which circuit doesn't work
  parameter real tp = 75p;             // delay from input clock to divided edge
  parameter real trf = 100p;           // risetime of Divided output
  parameter real vtol = 50m;           // voltage tolerance on input
  parameter real ttol = 20p;         //a 1% of Fnom error in edge is pretty big.
// LOCAL VARIABLES:  (Comment each one)
  real Iout;
  integer Dval, CLKval, Qval, Pdown, Enabled;
// BEHAVIOR  ----------------------------------------------------
  analog begin
    Pdown = (V(vdd,vss)<Vddmin)||(V(vb,vss)<Vbmin);
    Enabled = !Pdown;
    Dval = V(d_p,d_n) > 0;
    CLKval = V(clk_p,clk_n)>0;
    // recalculate the output current on each clock /data change
    @((initial_step)
    or (cross(V(clk_p,clk_n),0,ttol,vtol))  //if clock = 1
    or (cross(V(d_p,d_n),0,ttol,vtol))) begin  //if clock = 1
      Iout = Pdown?0:(V(vb,vss)>Vbmax?2*Iddq:Iddq);
    end
    // inputs
    Dval = V(d_p,d_n) > 0;
    CLKval = V(clk_p,clk_n)>0;
    // output
    Qval = CLKval ? Dval&&Enabled: Qval&&Enabled; //latch while clock = 1
    V(vdd,q_p) <+ I(vdd,q_p)*Rout;              // load resistor
    V(vdd,q_n) <+ I(vdd,q_n)*Rout;              // load resistor
    I(q_p,vss) <+ transition(Qval?0:Iout,tp,trf,trf);
    I(q_n,vss) <+ transition(Qval?Iout:0,tp,trf,trf);
    I(d_p,vss) <+ Cin * ddt(V(d_p,vss));              //Input Cap
    I(d_n,vss) <+ Cin * ddt(V(d_n,vss));              //Input Cap
    I(clk_p,vss) <+ Cin_clk * ddt(V(clk_p,vss));      //Input Cap
    I(clk_n,vss) <+ Cin_clk * ddt(V(clk_n,vss));      //Input Cap
  end
endmodule
```

2.2 VERILOG-AMS MODELS

2.2.1 BUF_BX1.vams

The model in listing implements the behavior of the buffer in figure. Note that nominal resistances are used to set the common mode.

Listing 3 BUF_BX2.vams: Buffer

```
// Verilog-AMS HDL for jbd_scratch.BUF_BX2:verilogams
// INCLUDE FILES :
`include "constants.vams"
`include "logic_ss.vams"
`include "cml_discpln.vams"
`include "disciplines.vams"
// DEFINE & TIMESCALE :
`timescale 1ns/1ps
//======================================================
module BUF_BX2 (
 // PINS :
 output (* integer supplySensitivity="\\vdd! ";
            integer groundSensitivity="\\vss! "; *) b_n, b_p,
 input  (* integer supplySensitivity="\\vdd! ";
            integer groundSensitivity="\\vss! "; *) a_n, a_p,
 input vb
);
// REGISTER and WIRE TYPES
 electrical (* integer inh_conn_prop_name = "vss";
   integer inh_conn_def_value = "cds_globals.\\vss! ";
       *) \vss! ;
 electrical (* integer inh_conn_prop_name = "vdd";
   integer inh_conn_def_value = "cds_globals.\\vdd! ";
       *) \vdd! ;
 cmlogic a_n, a_p;
 electrical vb;
 cmlogic_buf2000u  b_n, b_p;
// INTERNAL NODES :
reg OUT, PWRDN;
assign b_p = OUT||PWRDN;       // both outputs High if off
assign b_n = !OUT||PWRDN;      // both outputs High if off
// PARAMETERS:  (Comment each one)
 parameter real vbref_max = 0.7;      // volts
 parameter real vbref_min = 0.15;     // volts - below which its disabled
 parameter real Cin_vb = 10f;         //farads
 parameter real Igleak = 0;           // amps
 parameter real vddmin = 1.1;
 parameter real Td_ns = 0.010 ;       // ns to ps timescale
 parameter real ttol_vb = 10n;        // time tolerance for vb check
 parameter real Iddq = 800u;          //amps supply current
 parameter integer xinhibit = 0 from [0:1];
         // 0 is not set, X allowed..1 prevents X out
//-------------------------------------------------------------------------
 initial begin
  if (xinhibit == 1) OUT = a_p^a_n ? a_p && !a_n : 0 ;
   // if we inibit x - have to choose something
 end
 always @(above(V( vb, \vss! )-vbref_max, ttol_vb )) begin// no small timesteps
      $strobe("ILGLCOND: %M Vbias > max = %g \n",vbref_max);
 end
 always @(above(vbref_min-V( vb, \vss! ), ttol_vb )) begin // no small timesteps
  $strobe("STATINFO: %M Vbias < min - powering down = %g \n",vbref_max);
  PWRDN = 1;
 end
 always @(above(V( vb, \vss! )-vbref_min, ttol_vb )) begin // no small timesteps
  $strobe("STATINFO: %M Vbias > min - powering up = %g \n",vbref_max);
  PWRDN = 0;
 end
 always @(a_n, a_p) begin
```

```
  if (!PWRDN) begin
   if (xinhibit == 0)  #Td_ns OUT = a_p && !a_n;
   else begin  // we have to be careful only to output 1 or 0 - no X
     if  (a_p ^ a_n) #Td_ns OUT = a_p && !a_n;
   end
  end
 end // edge sensitive model of level sensitive behavior
//-------------------------------------------------------------------------
 analog begin
  I(vb, \vss! )  <+ Igleak + Cin_vb*ddt(V(vb, \vss! ));
  I( \vdd! , \vss! ) <+ Iddq;
 end
endmodule
```

2.2.2 LAT_AX1.vams

The model in listing implements the behavior of the latch in figure.. Here, we have provided a parameter to determine the initial output value for the case where the initial condition is latched, as this is required for a divider configuration.

Listing 4 LAT_AX4.vams: LATCH ams model

```
// Verilog-AMS HDL for jbd_scratch.LAT_AX4:verilogams
/*<<<< Same Header as BUF_BX2 >>>>>*/
module LAT_AX4 (            // PINS :
 output  (* integer supplySensitivity="\\vdd! ";
    integer groundSensitivity="\\vss! "; *)  q_n, q_p,
 input  (* integer supplySensitivity="\\vdd! ";
    integer groundSensitivity="\\vss! "; *)   clk_n, clk_p, d_n, d_p,
 input vb    );          // end of port declarations
// REGISTER and WIRE TYPES
/*<<<< Same Inherited Supplies as BUF_BX2 >>>>>*/
 cmlogic        d_n, d_p;
 cmlogic_buf     clk_n, clk_p;
 electrical      vb;
 cmlogic2400u    q_n, q_p;
// INTERNAL NODES :
reg OUT, PWRDN;
assign q_p = OUT||PWRDN;  // both outputs High if off
assign q_n = !OUT||PWRDN;   // both outputs High if off
// PARAMETERS:  (Comment each one)
/*<<<< Same Parameters  as BUF_BX2
except istate replaces xinhibit  >>>>>*/
 parameter integer istate = 0 from [-1:1];    // 0 is not set, X allowed..
//-------------------------------------------------------------------------
 initial begin
  if (istate == 1) OUT = 1'b1;
  else if (istate == -1) OUT = 1'b0;
 end
/*<<<< Same Assertions as BUF_BX2 >>>>>*/
// if used in TFF for clock divider, we can't tolerate X input, thus not X output
 always @(clk_p, clk_n, d_p, d_n) if (!PWRDN) begin
  if ((istate == 0) && clk_p && !clk_n) #Td_ns OUT = d_p && !d_n;
  else begin  // we have to be careful only to output 1 or 0 - no X
   if ( clk_p && !clk_n ) begin  // while clock is high
     if  (d_p ^ d_n) #Td_ns OUT = d_p && !d_n;
   end
  end
 end // edge sensitive model of level sensitive behavior
//-------------------------------------------------------------------------
 analog begin
  I(vb, \vss! )  <+ Igleak + Cin_vb*ddt(V(vb, \vss! ));
  I( \vdd! , \vss! ) <+ Iddq;
 end
endmodule
```

2.3 CONNECT MODELS

2.3.1 cmos_E2CMLb.vams

This model simply detects when the input signal crosses the threshold with enough margin. Ideally the reference for the threshold would be the other side of the differential pair, but this is not possible with the existing sensitivity attributes. Since the un-shifted version is simpler, we'll focus our attention on the level shifted models.

Listing 5 cmos_E2CMLb.vams: Buffered Input Connect Model

```
// 'cmos_E2CMLb.vams' - Verilog-AMS connection module.
// INCLUDE FILES:
`include "disciplines.vams"
`include "logic_ss.vams"
`include "cml_discpln.vams"
`timescale 1ns / 1ps
//=======================================================
connectmodule cmos_E2CMLb (Ain, Dout);
  input Ain; electrical Ain;                  // electrical input
  output Dout; cmlogic_buf Dout;              // logic output
// Supply Sensitivity attributes
  electrical (* integer supplySensitivity = "cds_globals.\\VDD! "  ; *) avdd;
  electrical (* integer groundSensitivity = "cds_globals.\\VSS! " ; *) agnd;
  electrical Xref; //refereence level for input cross
// INSTANCE PARAMETERS:
  parameter real Ampl_min = 100m;   // volts min input amplitude
  parameter real Ampl_nom = 300m;   // volts nom input amplitude
  parameter real Iin = 0;       // nom. input base current in active branch
  parameter real Cin = 10f;      // nominal cml gate input capacitance
  parameter real ttol=1p;        // time tolerance of crossing
  parameter real vtol=Ampl_min/2.0; // voltage tolerance of input event
// LOCAL VARIABLES:
  reg Dreg;          // output register
  real ibase;        // actual input base current in active branch
  real vinthresh;        // for CML_buf vinthresh = vdd-1.5*Ampl_nom
//=======================================================
  analog begin
    V(avdd,Xref) <+ 1.5*Ampl_nom; // for second level
    I(Ain,avdd ) <+ Cin*ddt(V(Ain,avdd)); // input capacitance
  end // analog
  initial begin
    Dreg= 1'bx;  // initial level
  end
/ Convert analog signal to high/low - X/notX handled in logic gate
  always @(above(V(Ain,Xref)-Ampl_min/2,ttol,vtol))
      begin Dreg=1; end                // +1 direction
  always @(above(V(Xref,Ain)-Ampl_min/2,ttol,vtol))
      begin Dreg=0; end                // -1 direction
  assign  Dout=Dreg;                   // assign register to output
endmodule
```

2.3.2 cmos_CMLb2E.vams

Considering the level shifted output driver, we need a way to have the output current always flowing in resistor RLS [see Figure 5]. Rather than depend on these currents to model the total supply current of the gate output, especially since there is no access to the bias level information, we cancel all the connectmodule supply current, allowing us to implement a more accurate Iddq in the gate model.

Supply Sensitivity is required as we need to connect Rout's to the rail, and sink current to the ground. Unlike most L2E cm's this is NOT a Logic to VOLTAGE conversion, rather we do a "logic-to-current-through-Rout" conversion, so that capacitive loading between gates can correctly impact the signal characteristics.

Listing 6 cmos_CMLb2E.vams: Buffered Output CM

```
// 'cmos_CMLb2E.vams' - Verilog-AMS connection module.
// INCLUDE FILES:
`include "disciplines.vams"
`include "cml_discpln.vams"
`timescale 1ns / 1ps
//=======================================================
connectmodule cmos_CMLb2E (Din, Aout);
  input Din; cmlogic_buf Din;              // input logic
  output Aout; electrical Aout;            // output electrical
// Supply Sensitivity attributes
  electrical (* integer supplySensitivity = "cds_globals.\\VDD! " ; *) avdd;
  electrical (* integer groundSensitivity = "cds_globals.\\VSS! " ; *) agnd;
  electrical levadj; // for the level shift resistor
  electrical Aoutb; // for the other side of the output
// INSTANCE PARAMETERS:
  parameter real tr=1.5p from (0:1u);       // risetime (L/X to Hi)
  parameter real ioutnom=1m from(0:1);      // switched part of the output
  parameter real ioutleak=1n from(0:1);     // fixed part of output current
  parameter real routhi=300*1m/ioutnom;     // current always flows in this one
                                            // shifting vhi down 1 level
  parameter real rout=  300*1m/ioutnom;     // switched current flows in this one
// LOCAL VARIABLES:
  reg Dreg;            // register to writeback logic
  real Ioutp, Ioutm;
  real Iloadcap;
//=======================================================
initial begin
    Ioutp = ioutnom/2;
    Ioutm = ioutnom/2;
  end
  always @(Din) begin
    case (Din)              // Lookup of V,R,T values for new state
      1'b1:  begin          // to HIGH
         Ioutp = 0; // no current = Vhi!
         Ioutm = ioutnom;
      end
      1'b0: begin           // to LOW
         Ioutm = 0; // no current = Vhi!
         Ioutp = ioutnom;
      end
      1'bz:  begin          // to HIGH IMPEDANCE
    $strobe("ILGLCOND %m: CML connect modules don't support Z, used X.\n");
        Ioutp = ioutnom/2;
        Ioutm = ioutnom/2;
      end
      default:  begin       // to UNKNOWN
        Ioutp = ioutnom/2;
        Ioutm = ioutnom/2;
        $strobe("Warn %m: CML connect modules used X. Ioutp = %g\n", Ioutp);
      end
    endcase
    Dreg=Din;
  end
  analog begin
    I(levadj,Aout) <+ V(levadj,Aout)/rout;
    I(levadj,Aoutb) <+ V(levadj,Aoutb)/rout;
    I(levadj,avdd) <+ V(levadj,avdd)/routhi;
    I(avdd,agnd) <+ -ioutnom - 2*ioutleak; // current out
    I(Aout,agnd) <+ ioutleak + transition(Ioutp ,0, tr, tr);
    // without Cload on Aoutb we didn't get a good match to reality..
    // rather than measure Cload, just get it's current
    Iloadcap = I(levadj,Aout)-I(Aout,agnd);
    // and source it into the opposite branch
    I(Aoutb,agnd) <+ ioutleak + transition(Ioutm ,0, tr, tr) - Iloadcap;
  end
```

```
  assign  Din=Dreg;                        // assign register to output
endmodule
```

2.3.3 CmosCMLrules.vams

This connectrules module defines standard 1.5V rules for standard logic, and supply sensitive rules for discipline of type "logic_ss", which would be enough for a dual supply CMOS design. With this project we add a set of rules for 1mA, 2mA and 4mA (X1, X2, X4, 8) CML gates with 2 signal levels, normal (cmlogic), and low (cmlogic_buf).

Listing 7 cmosCmlRules.vams: Connect Rules for CML

```
// 'cmosRules.vams' - Verilog-AMS supply sensitive connection rules file.
// Include definitions for logic and logic_ss:
`include "disciplines.vams"       `include "logic_ss.vams"
`include "cml_discpln.vams"       `define Vsup  1.5
`define Vthi  1.0                 `define Vtlo  0.5
`define Tr    0.2n                `define Rlo   200
`define Rhi   200                 `define Rx    40
`define Rz    10M                 `define Ampl_min 100m
`define Ampl_nom 300m             `define Tr_cml 1.5p
`define Ttol_cml 1p               `define Vcb_bias_min 0.30
`define Vcm_A_min_wrt_vdd  400m
`define Vcm_A_max_wrt_vdd  0
`define Vcm_B_min_wrt_vdd  700m
`define Vcm_B_max_wrt_vdd  200m
connectrules cmosCmlRules;
// Standard "logic" uses 1.5V rules:  - logic_cmos
  connect L2E #(
    .vsup(`Vsup), .vthi(`Vthi), .vtlo(`Vtlo),
    .tr(`Tr), .tf(`Tr), .tx(`Tr), .tz(`Tr),
    .rlo(`Rlo), .rhi(`Rhi), .rx(`Rx), .rz(`Rz) );
  connect E2L #(
    .vsup(`Vsup), .vthi(`Vthi), .vtlo(`Vtlo), .tr(`Tr) );
  connect Bidir #(
    .vsup(`Vsup), .vthi(`Vthi), .vtlo(`Vtlo),
    .tr(`Tr), .tf(`Tr), .tx(`Tr), .tz(`Tr),
    .rlo(`Rlo), .rhi(`Rhi), .rx(`Rx), .rz(`Rz) );
// Special "logic_ss" discipline uses supply sensitive modules:
  connect cmos_L2E_ss split #( .vthi(`Vthi/`Vsup), .vtlo(`Vtlo/`Vsup),
    .tr(`Tr), .tf(`Tr), .tx(`Tr), .tz(`Tr),
    .rlo(`Rlo), .rhi(`Rhi), .rx(`Rx), .rz(`Rz) );
  connect cmos_E2L_ss split #( .vthi(`Vthi/`Vsup), .vtlo(`Vtlo/`Vsup), .tr(`Tr) );
  connect cmos_Bidir_ss split #( .vthi(`Vthi/`Vsup), .vtlo(`Vtlo/`Vsup),
    .tr(`Tr), .tf(`Tr), .tx(`Tr), .tz(`Tr),
    .rlo(`Rlo), .rhi(`Rhi), .rx(`Rx), .rz(`Rz) );
// CML connections
  connect cmos_E2CML split // each has to have its own cm
    #(.Ampl_min(`Ampl_min),.Ampl_nom(`Ampl_nom),
    .ttol(`Ttol_cml) ) input electrical, output cmlogic;
  connect cmos_E2CMLb split // each has to have its own cm
    #(.Ampl_min(`Ampl_min),.Ampl_nom(`Ampl_nom),
    .ttol(`Ttol_cml) ) input electrical, output cmlogic_buf;
  // outputs are another matter.
  connect cmos_CML2E split // each has to have its own cm
    #(.ioutnom(1m),.ioutleak(2n),.tr(`Tr_cml) )
    input cmlogic1000u , output electrical ;
  connect cmos_CML2E split // each has to have its own cm
    #(.ioutnom(2m),.ioutleak(2n),.tr(`Tr_cml) )
    input cmlogic2000u , output electrical ;
  connect cmos_CML2E split // each has to have its own cm
    #(.ioutnom(4000u),.ioutleak(4n),.tr(`Tr_cml) )
    input cmlogic4000u , output electrical ;
  connect cmos_CMLb2E split // each has to have its own cm
    #(.ioutnom(2000u),.ioutleak(2n), .tr(`Tr_cml) )
    input cmlogic_buf2000u , output electrical ;
  connect cmos_CMLb2E split // each has to have its own cm
    #(.ioutnom(1000u),.ioutleak(1n), .tr(`Tr_cml) )
    input cmlogic_buf1000u , output electrical ;
  connect cmos_CMLb2E split // each has to have its own cm
    #(.ioutnom(4000u),.ioutleak(1n), .tr(`Tr_cml) )
    input cmlogic_buf4000u , output electrical ;
 connect cmlogic_buf1000u,  cmlogic_buf2000u,  cmlogic_buf4000u,
  cmlogic_buf     resolveto cmlogic_buf;
  connect cmlogic2000u,  cmlogic1000u, cmlogic4000u,  cmlogic
        resolveto cmlogic;
endconnectrules
`undef  <<ALL defines>>
```

3. PLL CASE-STUDY

When implemented on a typical HS CDR PLL, using this type of disciplines for the Data and clock channels, functional simulation run times improved from 2.5hours for 500ns at the transistor level, to 45min for 1.5us with 1 High-speed signal at the transistor level to 11 min for 4.5us with the entire High-speed signal path in the event driven simulator. By keeping the data and clock paths in the logic domain, the PLL data lock acquisition can be simulated approximately 125x faster than at transistor level, and more than 10x faster than with Verilog-A models. The performance improvement becomes more significant as additional low-speed circuits are added into the simulation.

4. CONCLUSIONS

A methodology for discrete value event driven modeling of differential circuits has been developed and applied. Application of this methodology to a PLL design facilitates enough of a simulation speed improvement to allow full functional simulations[6] to be run on the circuit prior to tapeout.

5. ACKNOWLEDGMENTS

My thanks to a Ron Vogelsong at Cadence for assistance with connectmodels, and to my co-workers at Scintera Networks, who needed this completed and documented.

6. REFERENCES

[1] *IEEE Standard Verilog® Hardware Description Language, IEEE Std. 1364-2001 rev C,* New York, 2001

[2] K. Kundert, O. Zinke, *The Designer's Guide to Verilog-AMS*, Boston:Kluwer, 2004

[3] Dan Fitzpatrick, Ira Miller, *Analog Behavioral Modeling with the Verilog-A Language* Boston: Kluwer, 1998

[4] Samir Palnitkar, *Verilog HDL* Mountain View: SunSoft, 1996

[5] *Verilog-AMS Language Reference ManualVersion 2.2, November 2004* Napa: Accelera, 2004

[6] G. Bonfini, M. Chiavacci, R. Mariani, E. Pescari, "An Analog Mixed-Signal Verification Kit for Verification of Analog-Digital Circuits" *CDNLive, June 2006* http://www.cdnusers.org/community/incisive/Tp_amsvkit_cdnusers.pdf