



Verification of CML Circuits in PLL designs with Verilog-AMS

Jonathan David
Mixed Signal Design Verification
Scintera Networks, Inc
j.david@ieee.org



Agenda



- CML Application Background
- Example CML Circuits
- Basic form of Behavioral Model
- Verilog-AMS models
- Connect models
- Results with PLL Application
- Looking Forward

Current Mode Logic Background

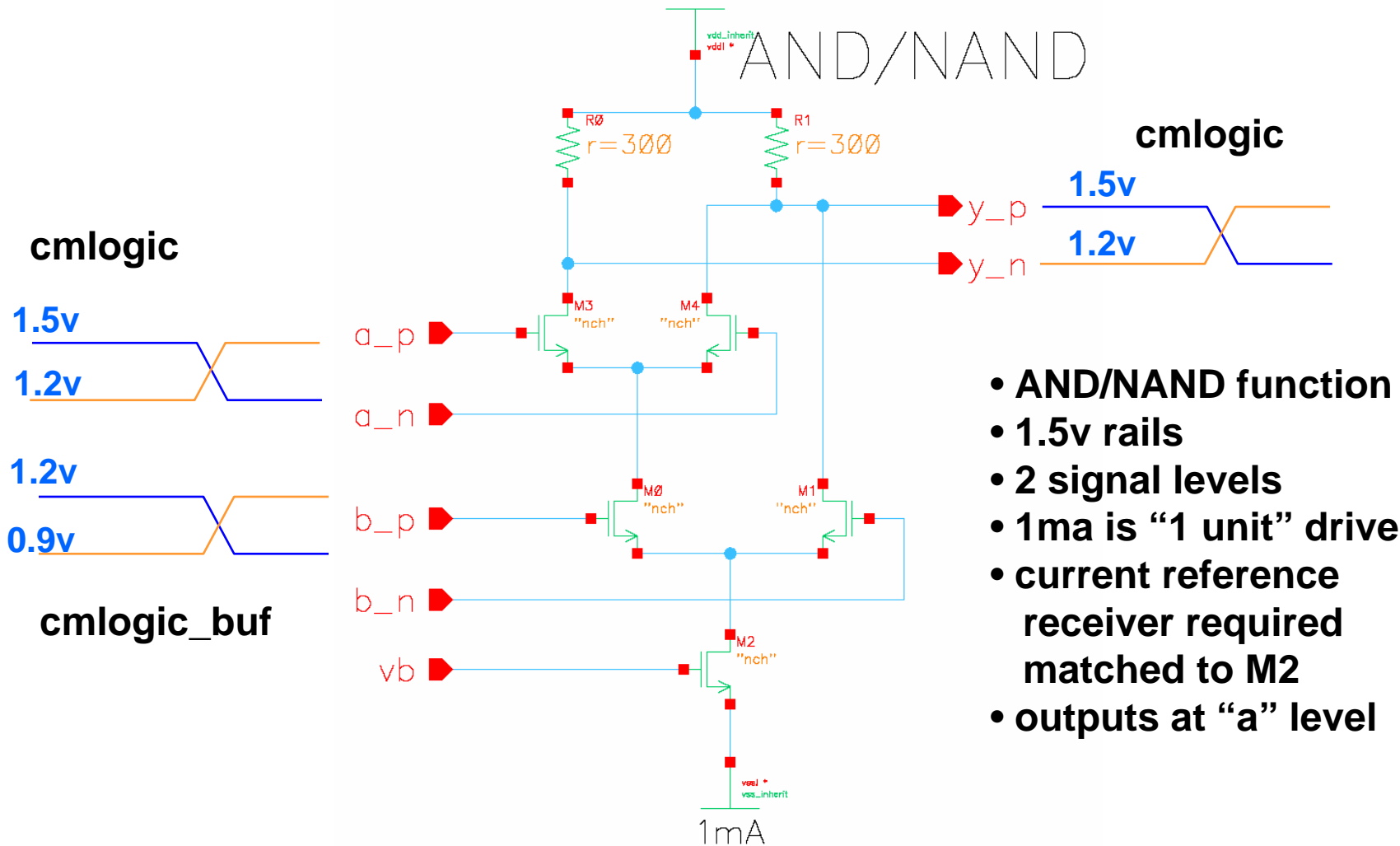


- Current Mode Logic continues to be the choice for the Highest speed signals
 - Remember ECL – an early class of CML
- Low-Voltage Swing
 - Driving transistors remain in Active region
- High power – but essentially constant
- Little difference (except performance) between CMOS and Bipolar.

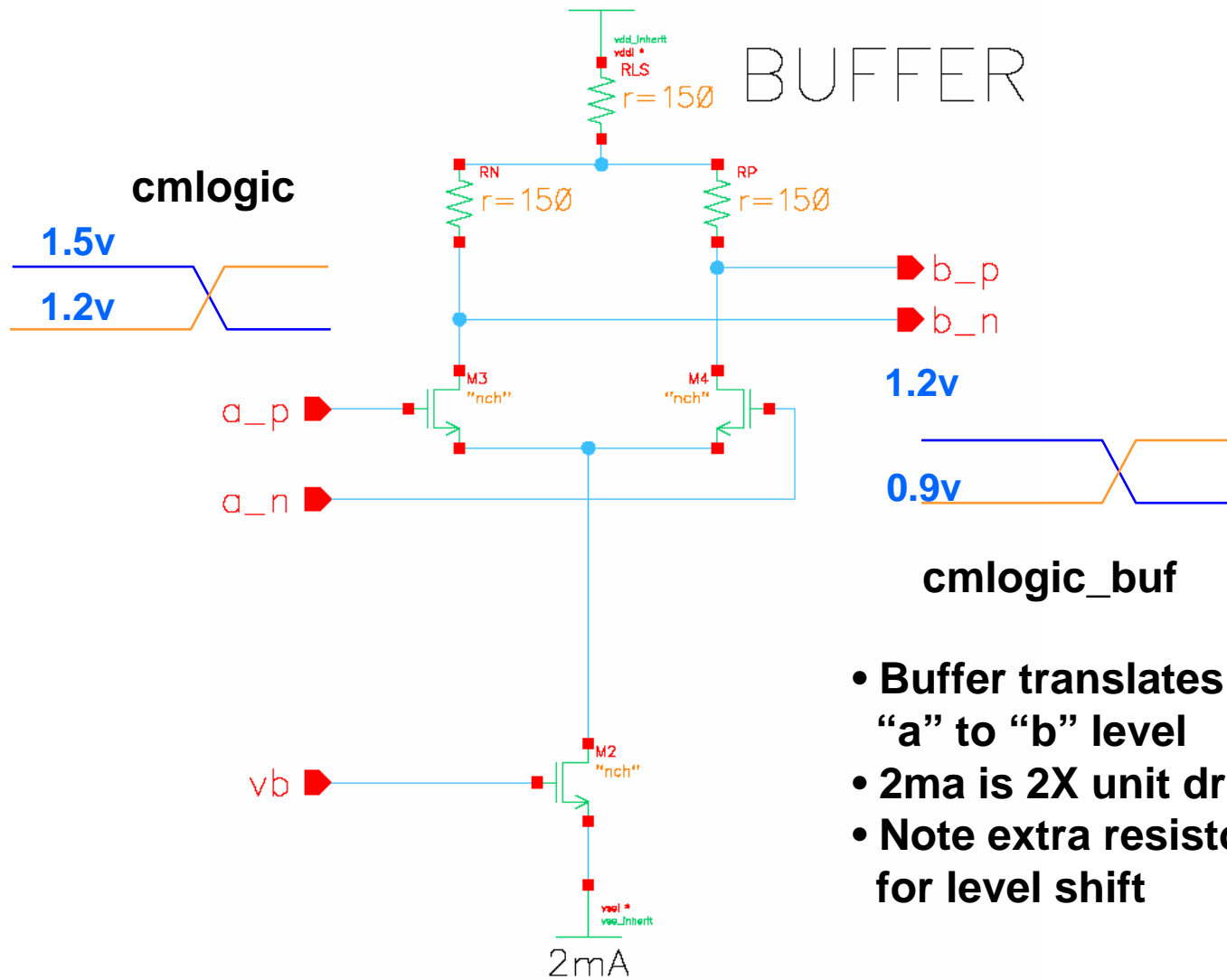
Applications today include

- SERDES based communications protocols
 - PCI-X
 - USB2.0
 - FBDimm
 - XAUI / XFI
 - 10G-Ethernet
- Frequency Synthesis

AND_AX1

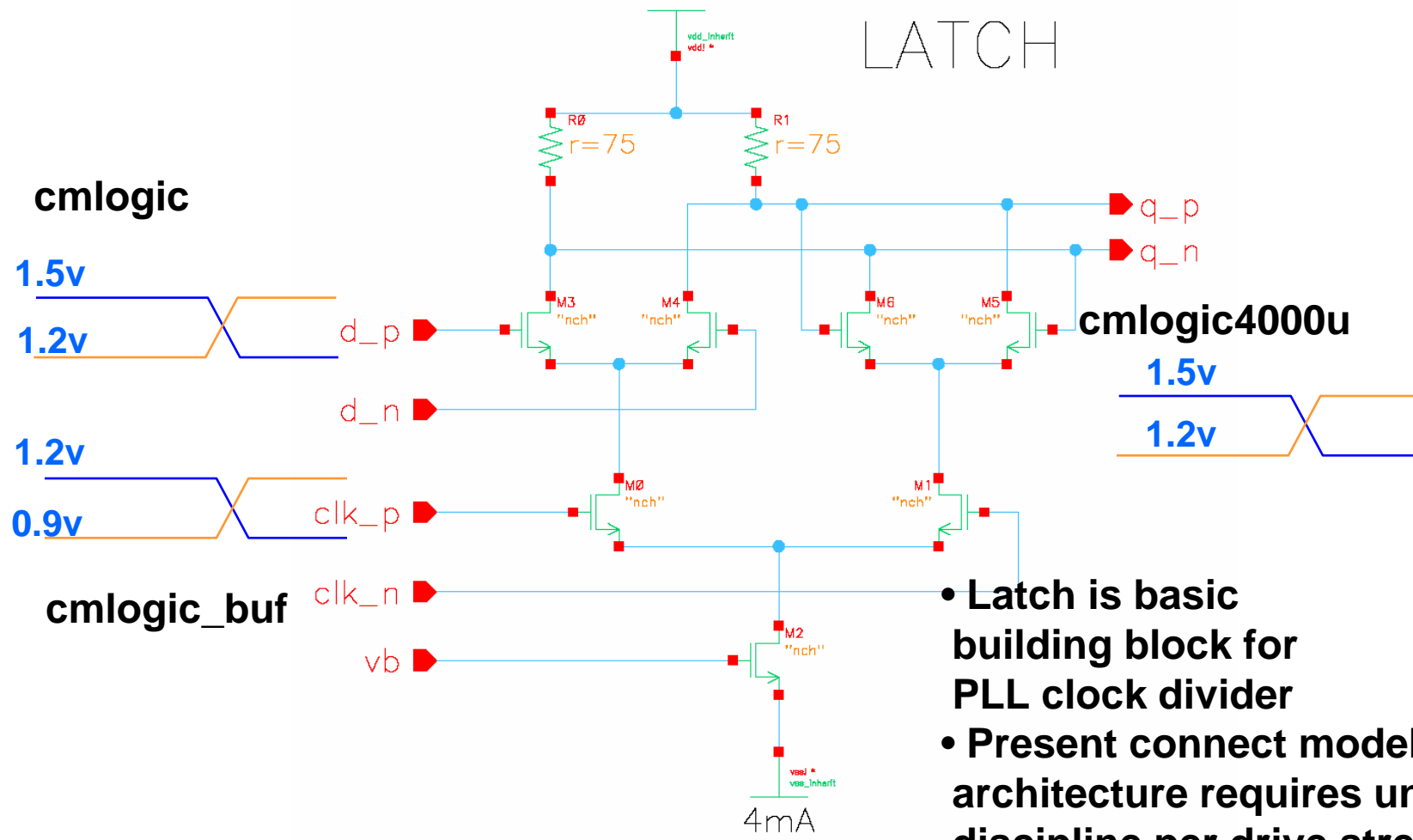


BUF_BX2



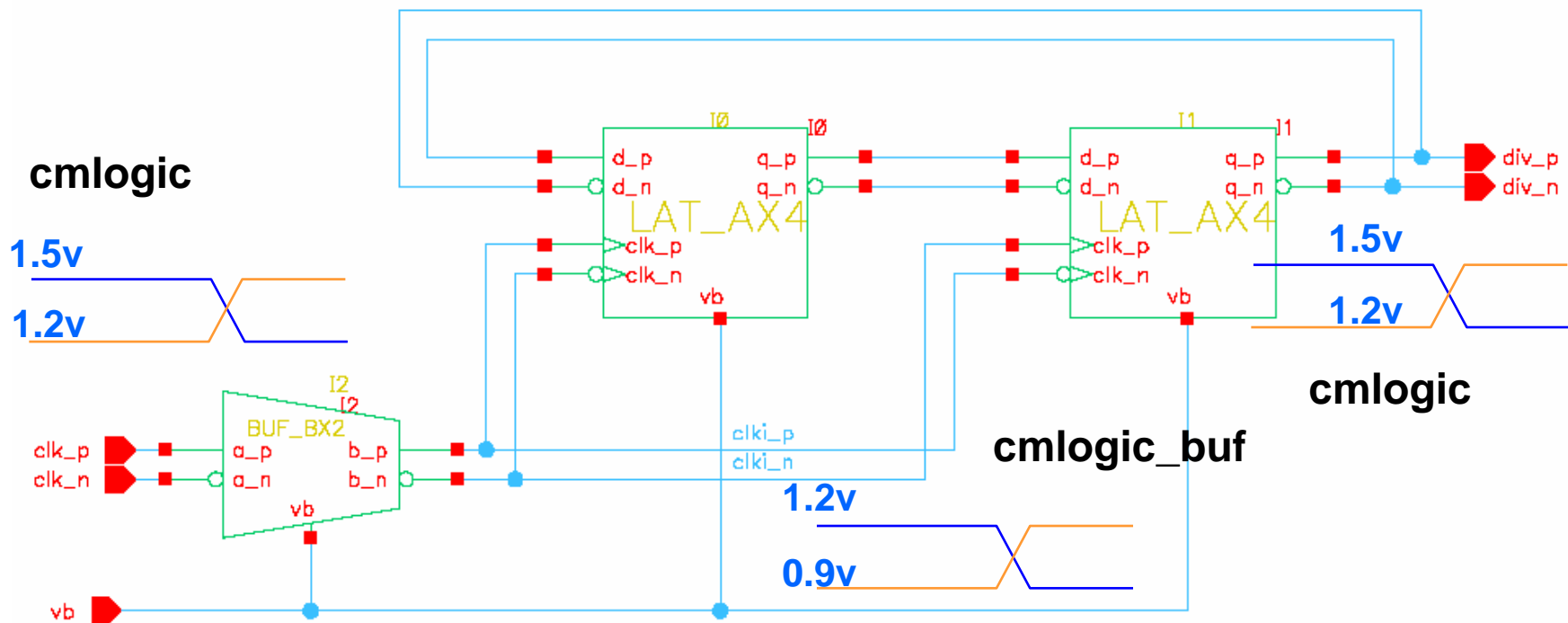
- Buffer translates “a” to “b” level
- 2ma is 2X unit drive
- Note extra resistor for level shift

LAT_AX4



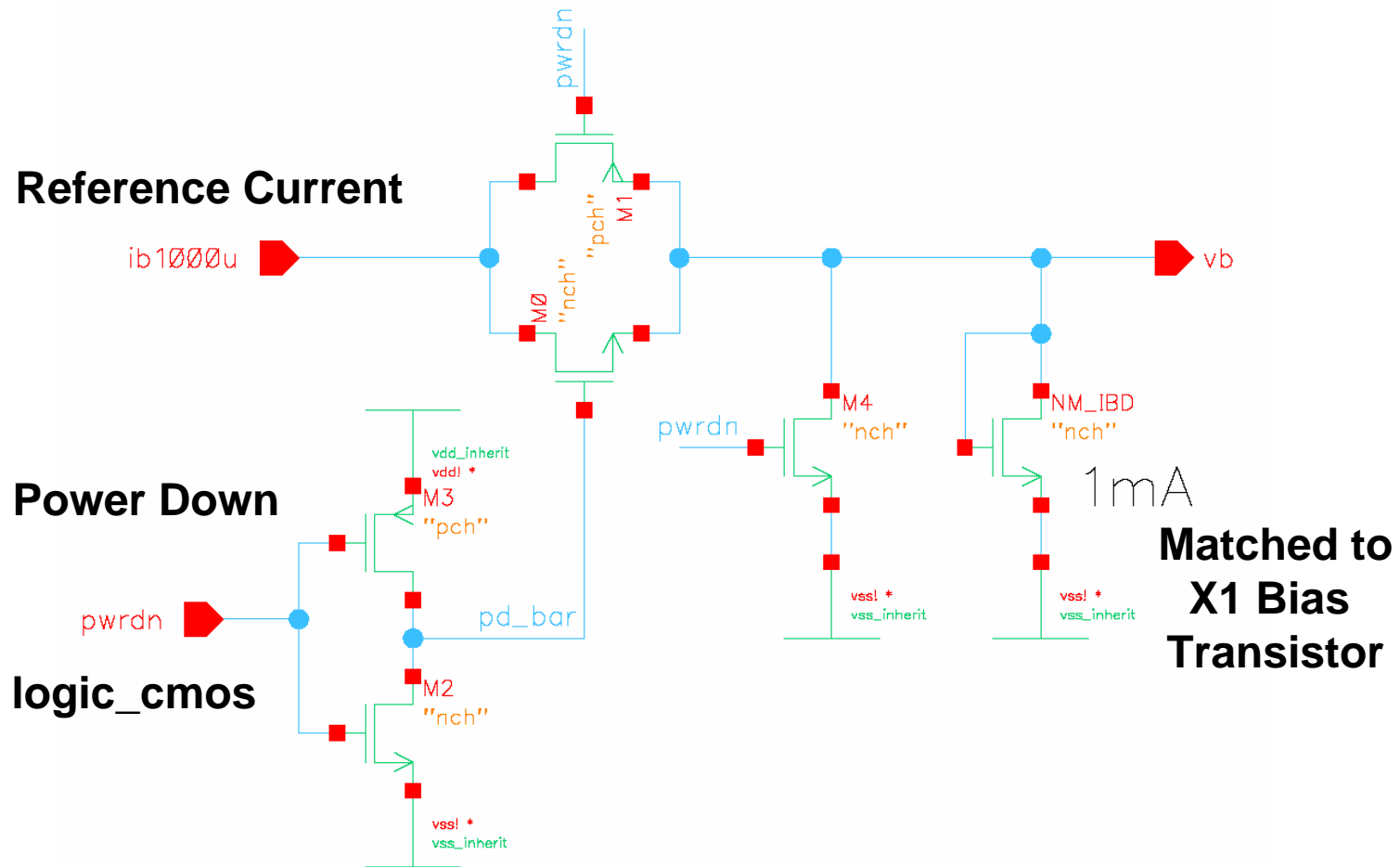
- Latch is basic building block for PLL clock divider
- Present connect model architecture requires unique discipline per drive strength

DFF input is inverted output

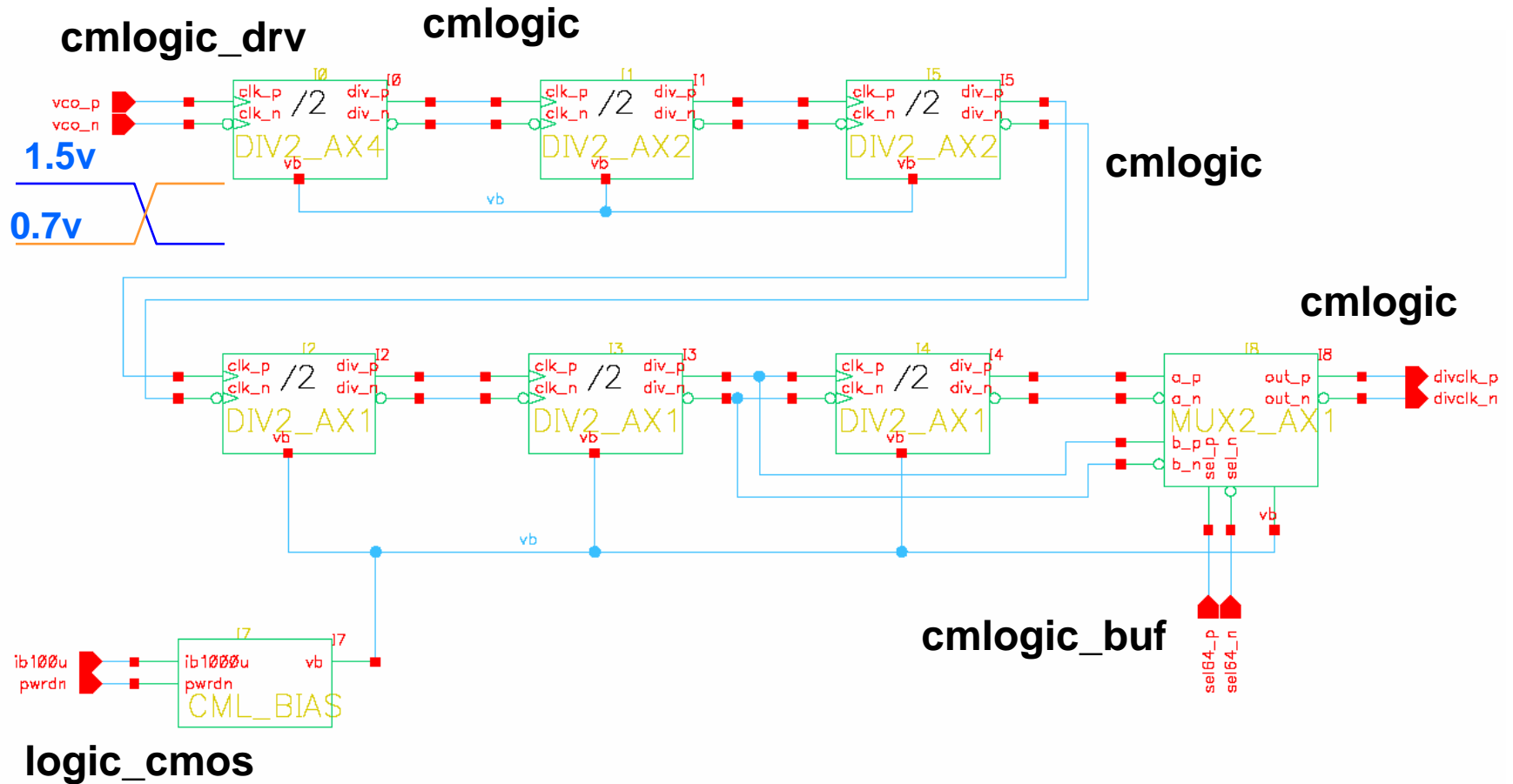


- Toggle FF – composed of two latches + buffer for clock level shift from prior stage

CML_BIAS – the power down control signal is low voltage cmos



CLKDIV64 – First stage input is driven by VCO with a wider swing.

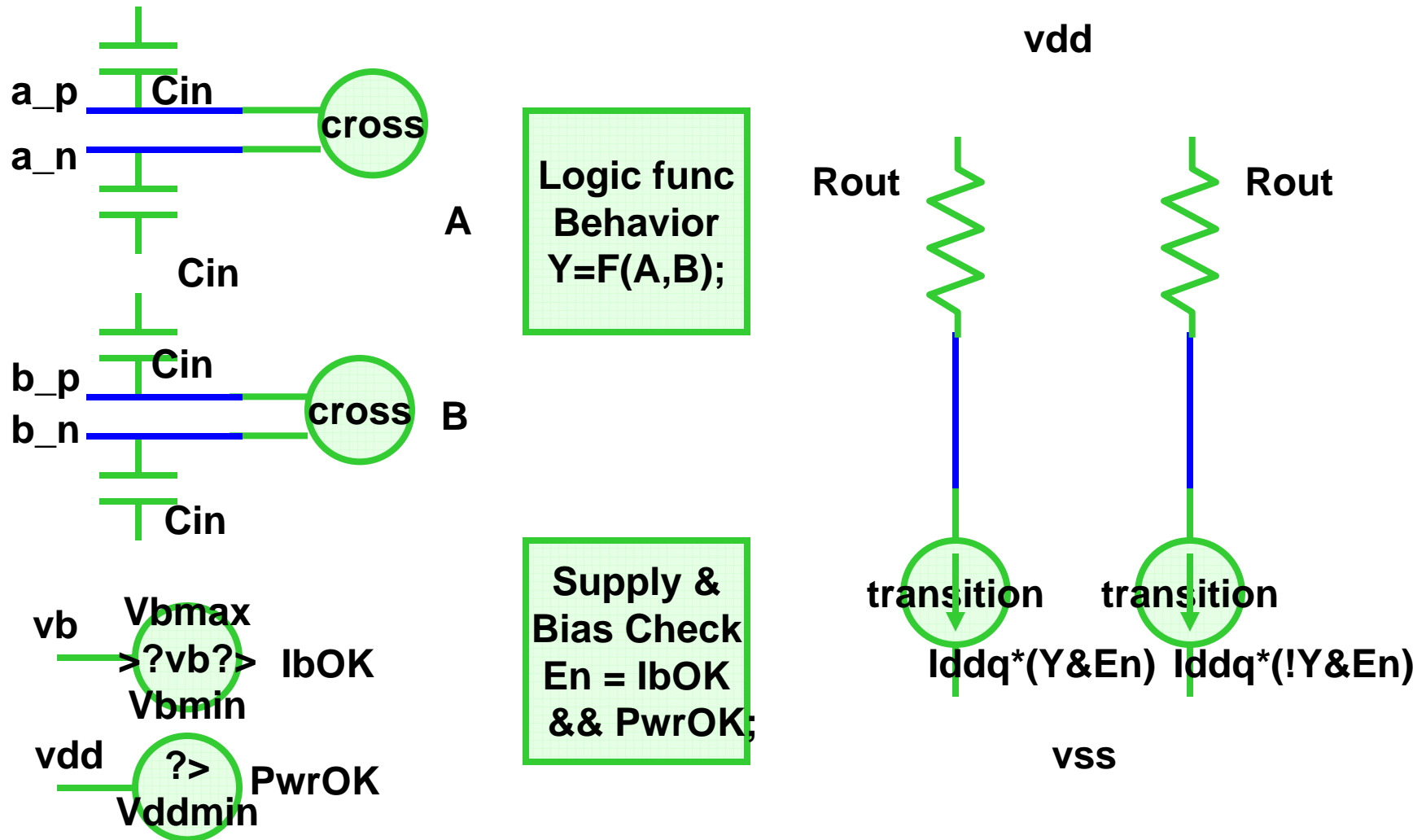


Compositional Approach

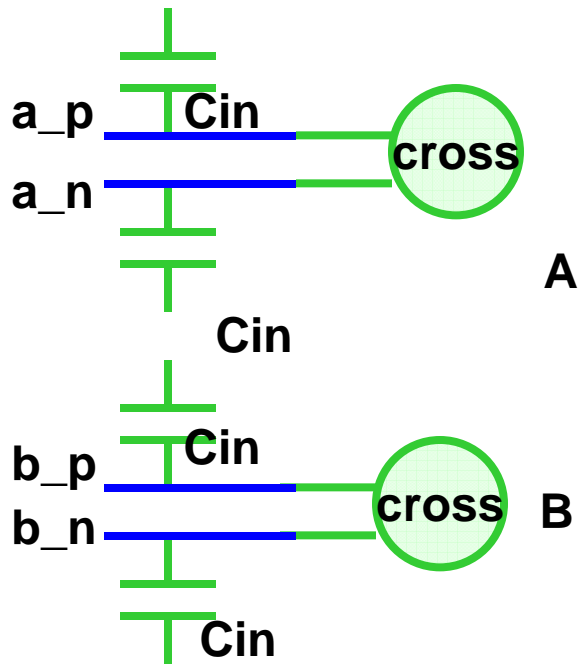


- Start with Verilog-A model of 1 gate
- Partition into Logical model and Interface
- Put Interface into Connect-Models & connect Rules
- Remainder is our Verilog-AMS model

CML Model Architecture - cmlogic

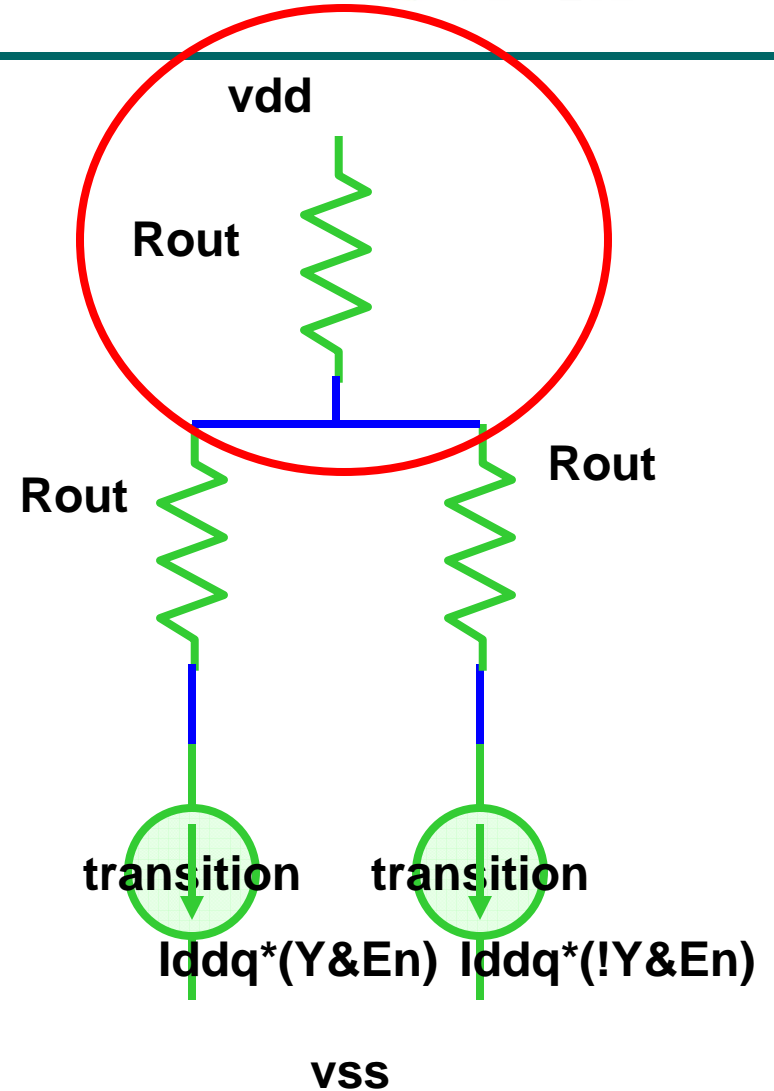
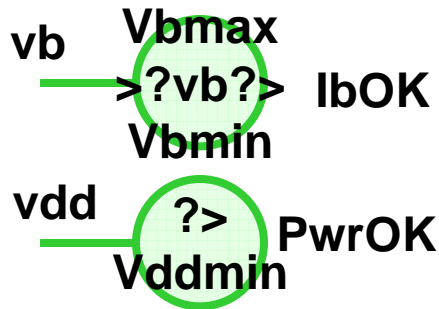


CML Model Architecture – cmlogic_buf



Logic func
Behavior
 $Y=F(A,B);$

Supply &
Bias Check
 $En = IbOK$
&& $PwrOK;$



AND2_AX1 – Verilog-A model



```
// VerilogA for jbd_cml, AND2_AX1, veriloga
// $Revision: 1.9 $
// $Date: 2006/07/02 07:46:34 $
// $Author: j david $
// REVISION HISTORY:
// $Log: veriloga.va,vs $
`include "constants.vams"
`include "disciplines.vams"
//=====
module AND2_AX1( y_n, y_p, a_n, a_p, b_n, b_p, vb );
// REGISTER and WIRE TYPES
output y_n, y_p; electrical y_n, y_p;
input a_n, a_p; electrical a_n, a_p;
input b_n, b_p; electrical b_n, b_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 = 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 lout;
integer Aval, Bval, Yval, Pdown, Enabled;
// STRUCTURE
resistor #(.r(Rout)) R1p (vdd, y_p);
resistor #(.r(Rout)) R1n (vdd, y_n);
```

```
capacitor #(.c(Cin)) C1p (a_p, vss);
capacitor #(.c(Cin)) C1n (a_n, vss);
capacitor #(.c(Cin)) C2p (b_p, vss);
capacitor #(.c(Cin)) C2n (b_n, vss);
//-----
analog begin
// re evaluate the logic when inputs change
@((initial_step)
or (cross(V(a_p,a_n),0,ttol,vtol))
or (cross(V(b_p,b_n),0,ttol,vtol))) begin
// inputs
Aval = V(a_p,a_n) > 0;
Bval = V(b_p,b_n)>0;
// output
Yval = Aval&&Bval; // Heres the actual logic function
end
l(y_p,vss) <+ transition(!Yval&&Enabled?Iddq:0,tp,trf,trf);
l(y_n,vss) <+ transition(Yval&&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
```

AND model Core



```
// STRUCTURE
resistor #(.r(Rout)) R1p (vdd, y_p);
resistor #(.r(Rout)) R1n (vdd, y_n);
analog begin
  // re evalutate the logic when inputs change
  @((initial_step)
  or (cross(V(a_p,a_n),0,ttol,vtol))
  or (cross(V(b_p,b_n),0,ttol,vtol))) begin
    // inputs
    Aval = V(a_p,a_n) > 0;
    Bval = V(b_p,b_n)>0;
    // output
    Yval = Aval&&Bval; //Heres the actual logic function
  end
  l(y_p,vss) <+ transition(!Yval&&Enabled?liddq:0,tp,trf,trf);
  l(y_n,vss) <+ transition(Yval&&Enabled?liddq:0,tp,trf,trf);
  ...
end
```

Assertions monitor bias and supply



// 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;
```

BUF_BX2.va



```
// 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 lout;
  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 evaluate 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
  l(b_p,vss) <+ transition(!Bval&&Enabled?Iddq:0,tp,trf,trf);
  l(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
```


Buffer model

```
// 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
...
end
```

LAT_AX4.va



```
// 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.\lvdd! "; *) vdd;
electrical (* integer inh_conn_prop_name = "vss";
integer inh_conn_def_value = "cds_globals.\lvss! "; *) 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 lout;
integer Dval, CLKval, Qval, Pdown, Enabled;
```

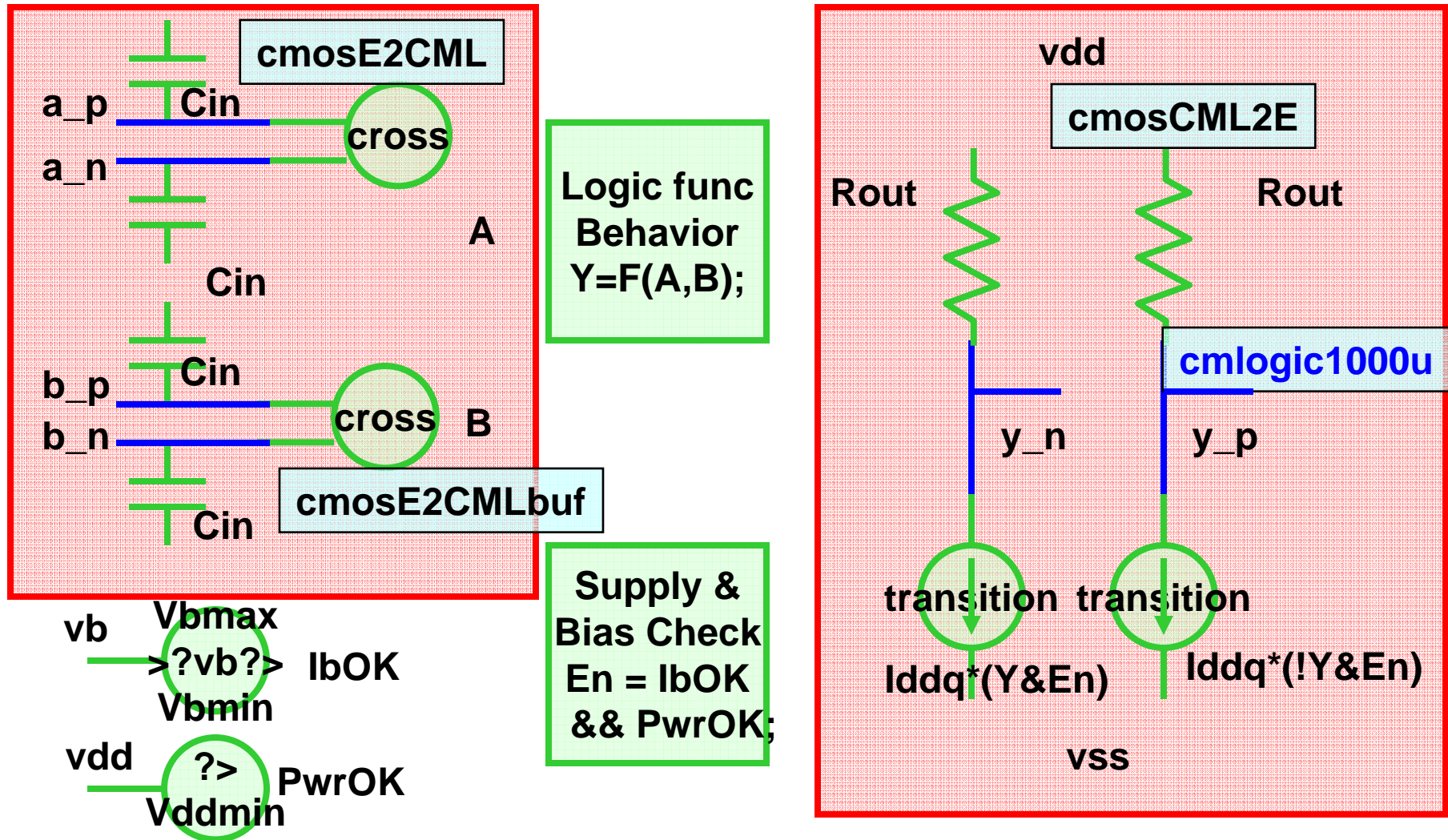
```
// BEHAVIOR -----
analog begin
Pdown = (V(vdd,vss)<Vddmin)|| (V(vb,vss)<Vbmin);
Enabled = !Pdown;
// recalculate the output current on each clock /data change
@((initial_step)
or (cross(V(clk_p,clk_n),0,ttol,vtol)) //if clock = 1
or (cross(V(d_p,d_n),0,ttol,vtol))) begin //if clock = 1
lout = Pdown?0:(V(vb,vss)>Vbmax?2*Iddq:Iddq);
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:lout,tp,trf,trf);
I(q_n,vss) <+ transition(Qval?lout: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
```

Latch model uses behavioral passives needs to retain state information - Qval



```
// BEHAVIOR -----  
analog begin  
  Pdown =  
    (V(vdd,vss)<Vddmin)|| (V(vb,vss)<Vbmin);  
  Enabled = !Pdown;  
  // 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);  
// behavioral parasitics  
  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
```

CML Interface Models Needed



BUF_BX2.vams



```
// Verilog-AMS HDL for jbd_scratch.BUF_BX2:verilogams
// INCLUDE FILES :
`include "constants.vams"
`include "logic_ss.vams"
`include "cml_discipln.vams"
`include "disciplines.vams"
// DEFINE & TIMESCALE :
`timescale 1ns/1ps
//=====
module BUF_BX2 (
  // PINS :
  output (* integer supplySensitivity="\lvdd! ";
    integer groundSensitivity="\lvss! "; *) b_n, b_p,
  input (* integer supplySensitivity="\lvdd! ";
    integer groundSensitivity="\lvss! "; *) a_n, a_p,
  input vb
);
// REGISTER and WIRE TYPES
electrical (* integer inh_conn_prop_name = "vss";
  integer inh_conn_def_value = "cds_globals.\lvss! ";
  *) \vss! ;
electrical (* integer inh_conn_prop_name = "vdd";
  integer inh_conn_def_value = "cds_globals.\lvdd! ";
  *) \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
```

Buffer Declarations

```
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;
```

Buffer.vams model core



```
//-----  
initial begin  
  if (xinhibit == 1) OUT = a_p^a_n ? a_p && !a_n : 0 ;  
    // if we inhibit x - have to choose something  
end  
...{assertions here}  
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
```

Buffer.vams Assertions



```
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
```


LAT_AX4.vams



```
// Verilog-AMS HDL for jbd_scratch.LAT_AX4:verilogams
// INCLUDE FILES :
`include "constants.vams"
`include "logic_ss.vams"
`include "cml_discipln.vams"
`include "disciplines.vams"
// DEFINE & TIMESCALE :
`timescale 1ns/1ps
//===== 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
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 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)
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 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
always @(above(V( vb, lvss! )-vbref_max, ttol_vb )) begin// no small timesteps
    $strobe("ILGLCOND: %M Vbias > max = %g \n",vbref_max);
end
always @(above(vbref_min-V( vb, lvss! ), ttol_vb )) begin // no small timesteps
    $strobe("STATINFO: %M Vbias < min - powering down = %g \n",vbref_max);
    PWRDN = 1;
end
always @(above(V( vb, lvss! )-vbref_min, ttol_vb )) begin // no small timesteps
    $strobe("STATINFO: %M Vbias > min - powering up = %g \n",vbref_max);
    PWRDN = 0;
end
// 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, lvss! ) <+ Igleak + Cin_vb*ddt(V(vb, lvss! ));
    I(\vdd! , lvss! ) <+ Iddq;
end
endmodule
```

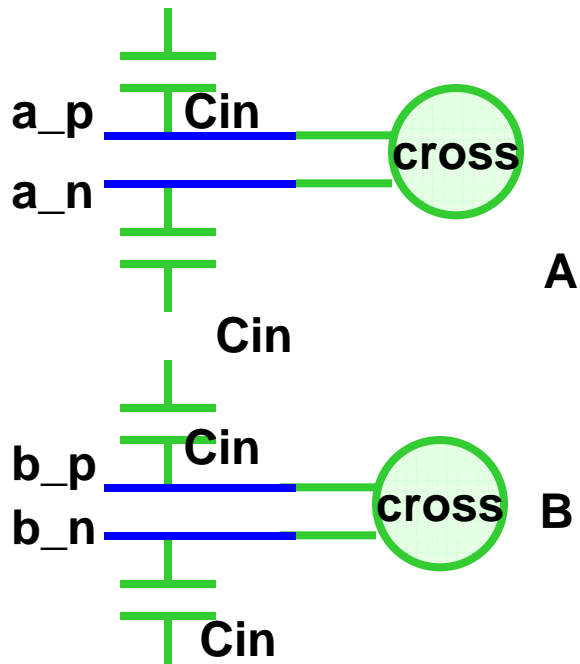
Latch.vams core



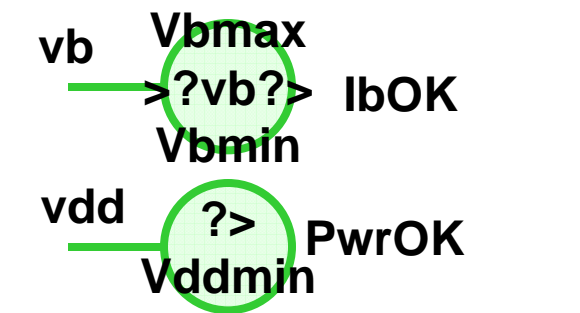
```
// if used in TFF for clock divider, we can't tolerate X input, thus not X output  
always @(clk_p, clk_n, d_p, d_n) if (!PWRDN) begin  
  if ((istate == 0) && clk_p && !clk_n) #Td_ns OUT = d_p && !d_n;  
  else begin // we have to be careful only to output 1 or 0 - no X  
    if ( clk_p && !clk_n ) 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
```

CML Interfaces Needed for Buffer

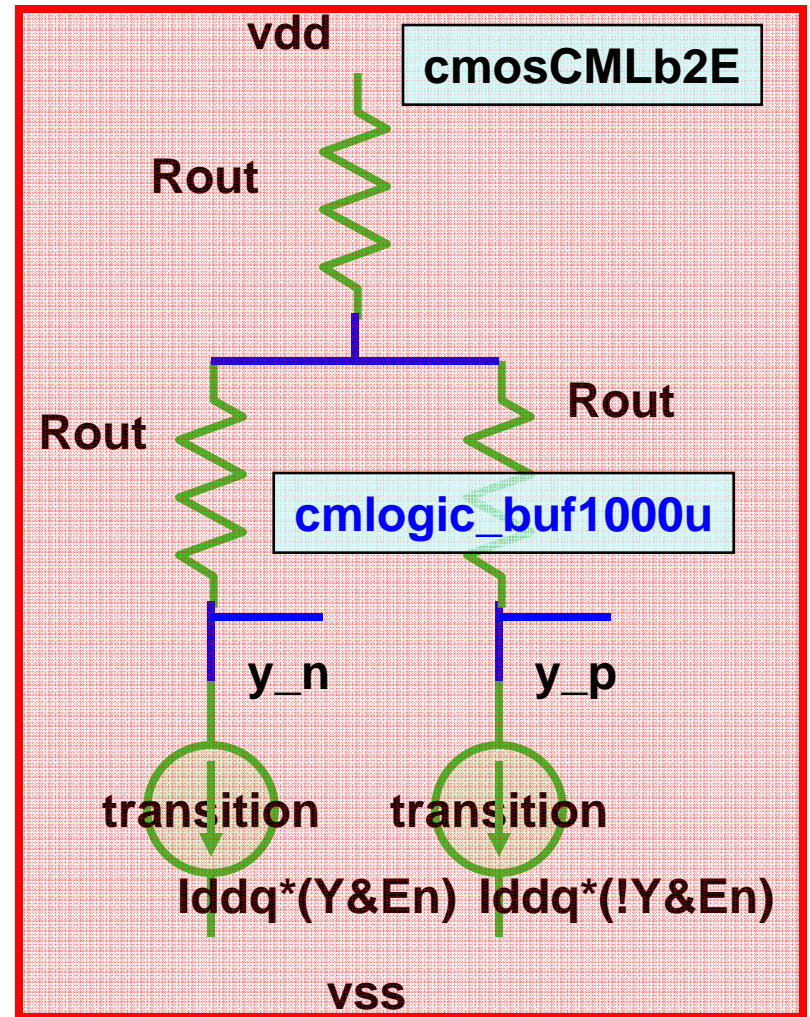
("a" level is simpler)



Logic func
Behavior
 $Y=F(A,B);$



Supply &
Bias Check
 $En = IbOK$
&& $PwrOK;$



Cmos_E2CMLb.vams



```
// '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; //referenece 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 vintresh;    // for CML_buf vintresh = 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
```

Cmos_E2CMLb.vams core



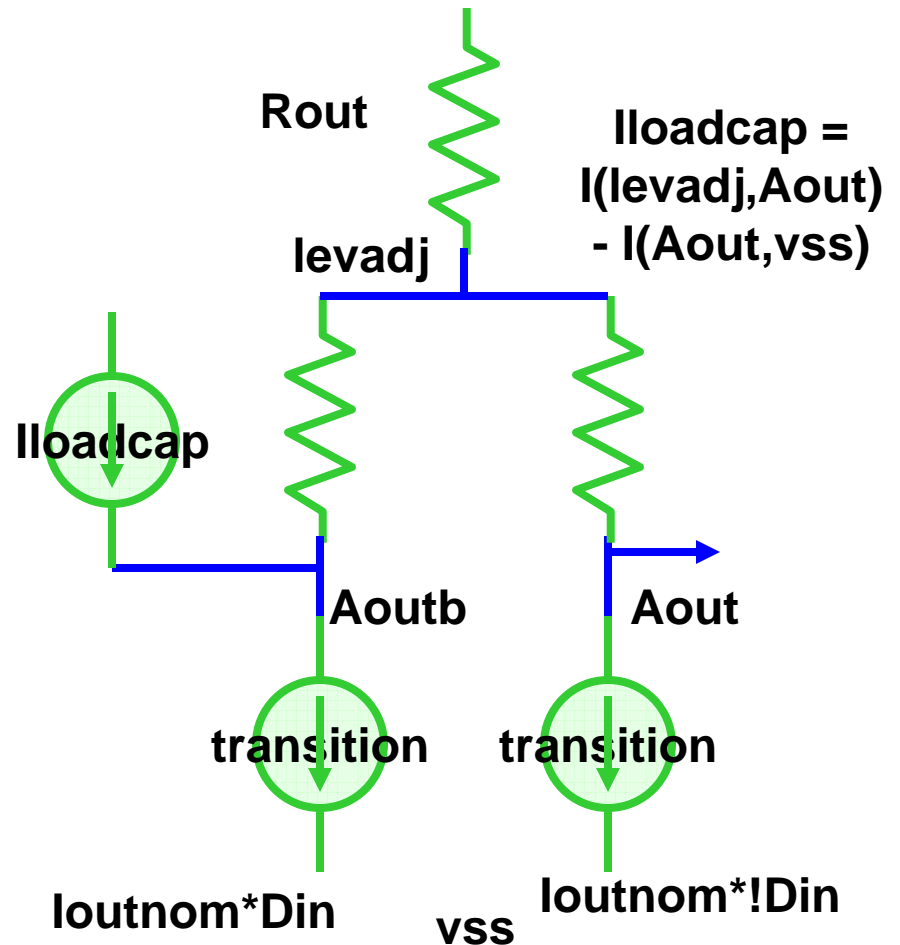
```
//=====
analog begin
  V(avdd,Xref) <+ 1.5*Ampl_nom; // for second level
  I(Ain,avdd) <+ Cin*ddt(V(Ain,avdd)); // input capacitance
end // analog
initial begin
  Dreg= 1'bx; // initial level
end
/ Convert analog signal to high/low - X/notX handled in logic gate
always @(above(V(Ain,Xref)-Ampl_min/2,ttol,vtol))
  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
```

Without “Cross Sensitivity” we can only compare each input To a reference. Need a way to compare them to each other.

Model Architecture for CMLb2E



- CML2E only needs
 - $I_{out} = I_{nom} * !Din$
- For the Level Shifting in CMLb2E
 - Current needs to flow thru R_{out} whether output is high or low
- Solution
 - Make an internal copy of other output
 - No way to pass Load Capacitance connected to A_{out} into connect model.
 - Measure I_{out} , and assume and opposite current transition would occur on A_{outb} .



Cmos_CMLb2E.vams



```
// 'cmos_CMLb2E.vams' - Verilog-AMS connection module.
// INCLUDE FILES:
`include "disciplines.vams"
`include "cml_discipln.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
```

Cmos_CMLb2E.vams CORE



```
always @(Din) begin
  case (Din) // Lookup of V,R,T values for new state
    1'b1: begin // to HIGH
      loutp = 0; // no current = Vhi!
      loutm = ioutnom;
    end
    1'b0: begin // to LOW
      loutm = 0; // no current = Vhi!
      loutp = ioutnom;
    end
    1'bz: begin // to HIGH IMPEDANCE
      $strobe("ILGLCOND %m: CML connect modules don't support Z,
        used X.\n");
      loutp = ioutnom/2;
      loutm = ioutnom/2;
    end
    default: begin // to UNKNOWN
      loutp = ioutnom/2;
      loutm = ioutnom/2;
      $strobe("Warn %m: CML connect modules used X. loutp =
        %g\n", loutp);
    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(loutp ,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(loutm ,0, tr, tr) - Iloadcap;
end
assign Din=Dreg; // assign register to output
endmodule
```


- **CMLOGIC**

// each input has to have its own cm

```
connect cmos_E2CML split
    #(.Ampl_min( Ampl_min),.Ampl_nom( Ampl_nom),
    .ttol( Ttol_cml) ) input electrical, output cmlogic;
```

```
connect cmos_CML2E split
```

```
    #(.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 cmlogic2000u, cmlogic1000u,
    cmlogic4000u, cmlogic
    resolveto cmlogic;
```

- **CMLOGIC_BUF**

// each input has to have its own cm

```
connect cmos_E2CMLb split
    #(.Ampl_min( Ampl_min),.Ampl_nom( Ampl_nom),
    .ttol( Ttol_cml) ) input electrical, output cmlogic_buf;
```

```
connect cmos_CMLb2E split // each has to have its own
```

```
    #(.ioutnom(2000u),.ioutleak(2n),.tr( Tr_cml) )
    input cmlogic_buf2000u , output electrical ;
```

```
connect cmos_CMLb2E split // each has to have its own
```

```
    #(.ioutnom(1000u),.ioutleak(1n),.tr( Tr_cml) )
    input cmlogic_buf1000u , output electrical ;
```

```
connect cmos_CMLb2E split // each has to have its own
```

```
    #(.ioutnom(4000u),.ioutleak(1n),.tr( Tr_cml) )
    input cmlogic_buf4000u , output electrical ;
```

```
connect cmlogic_buf1000u, cmlogic_buf2000u,
    cmlogic_buf4000u, cmlogic_buf
    resolveto cmlogic_buf;
```

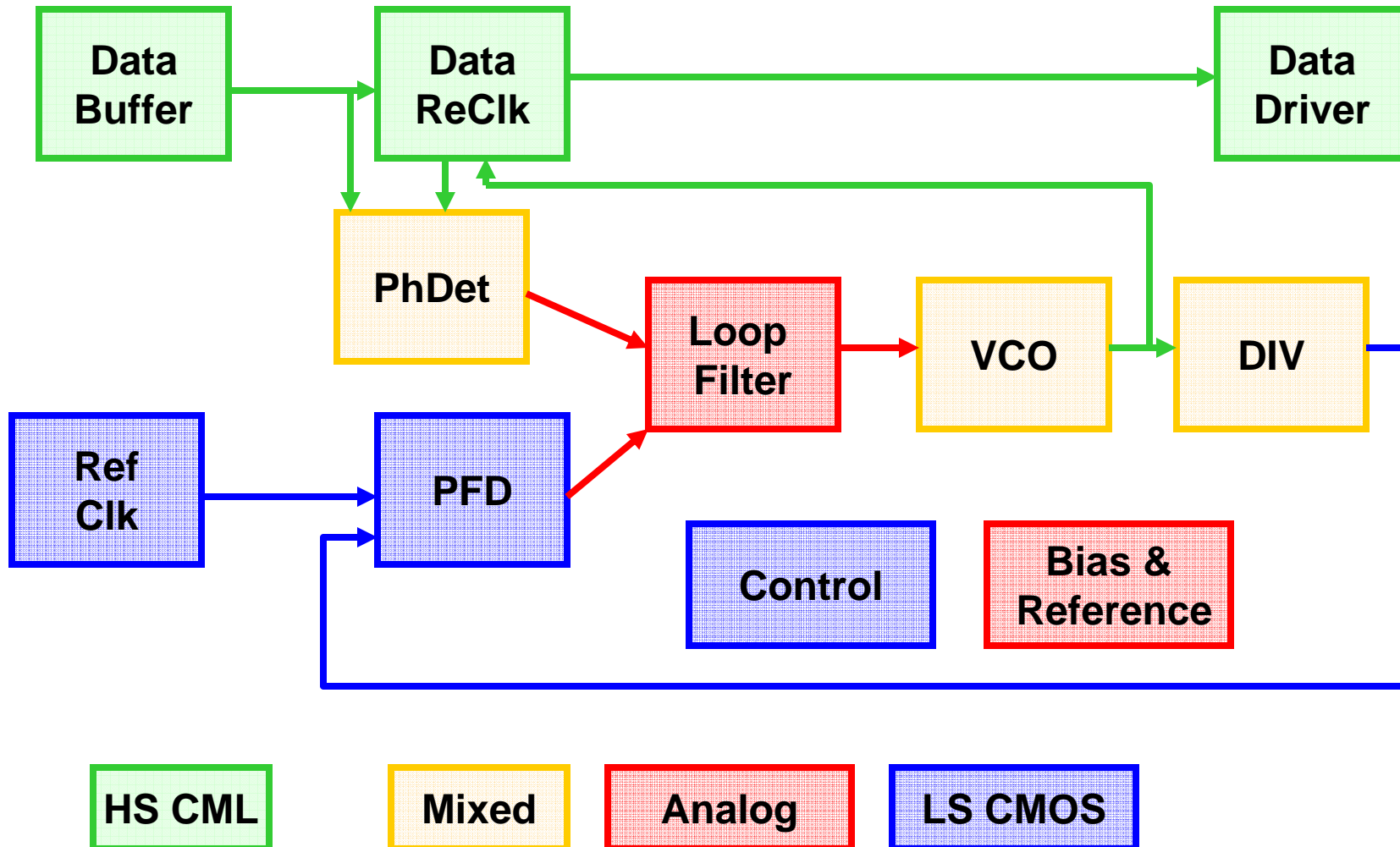
cmosCmlRules.vams



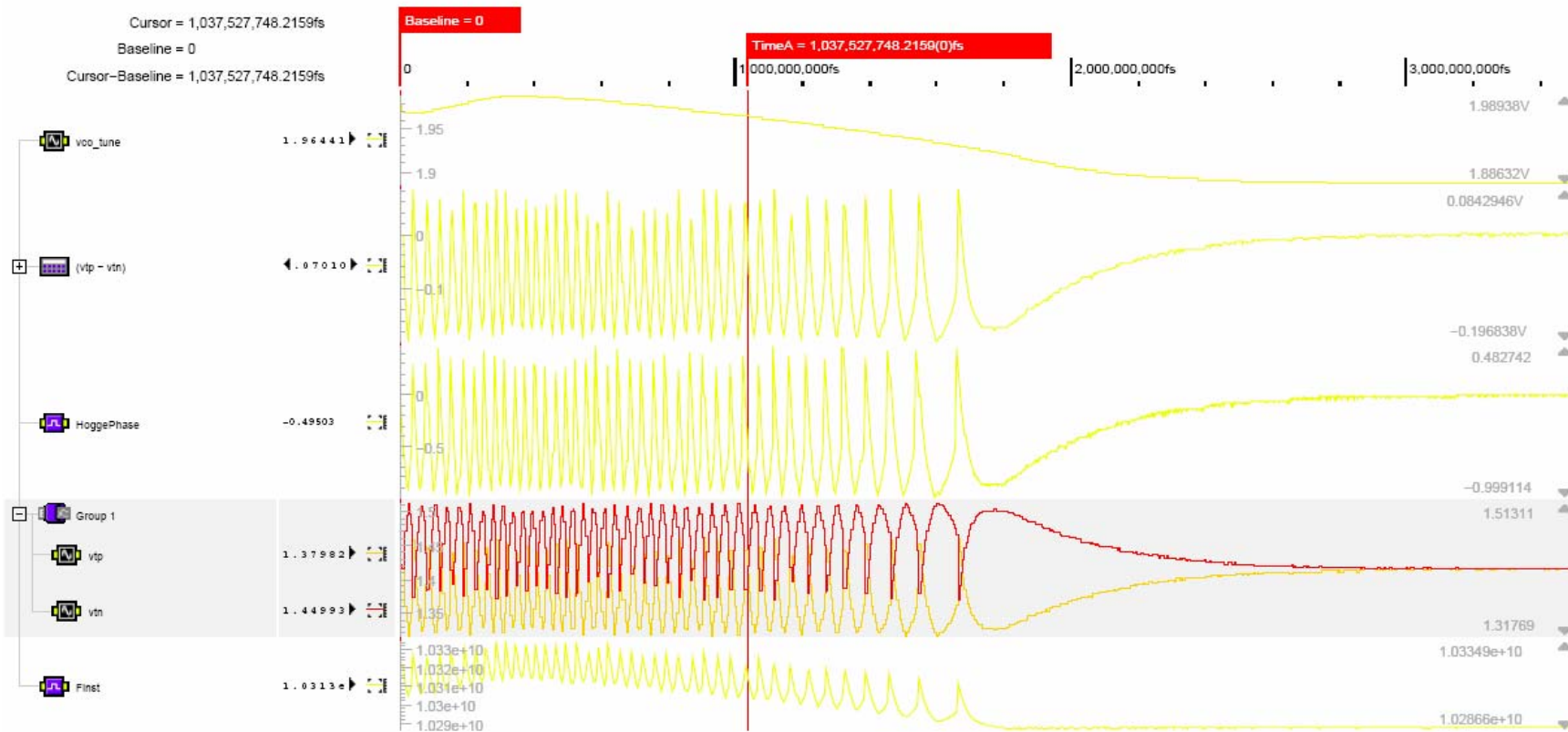
```
// 'cmosRules.vams' - Verilog-AMS supply sensitive connection rules file.
// Include definitions for logic and logic_ss:
`include "disciplines.vams"      `include "logic_ss.vams"
`include "cml_discipln.vams"    `define Vsup 1.5
`define Vthi 1.0                `define Vtlo 0.5
`define Tr 0.2n                 `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>>
```

CDR simulation Setup

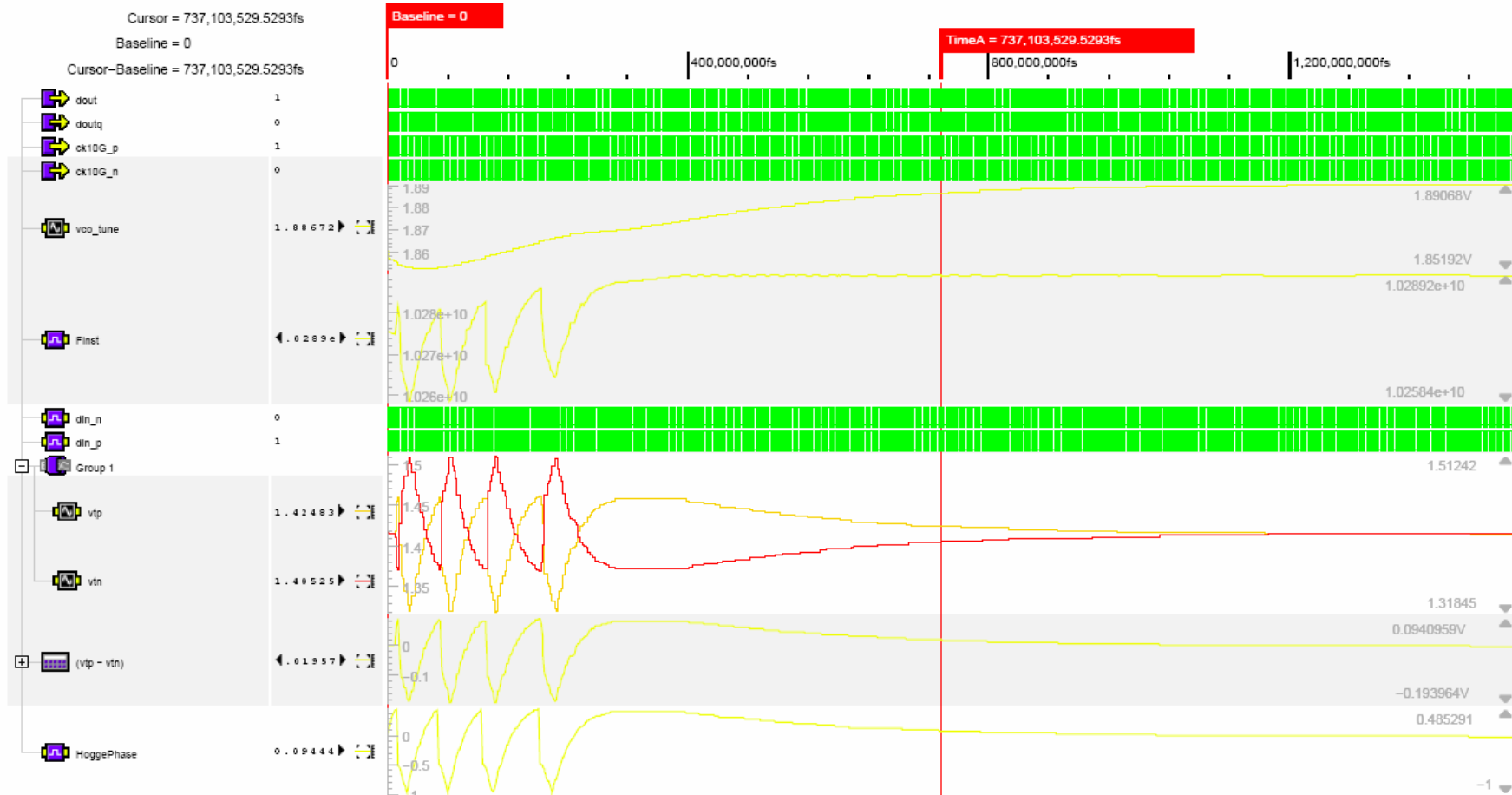


AMS sim of CDR locking to data 10.3G start to 10.288G data rate



**This 45min simulation for 3.5us - LOS block still had analog input
Compares to ~2.5h for 500ns at transistor level + VerilogA VCO**

AMS sim of CDR locking to data 10.27G start to 10.288G data rate



**This 11min simulation for 1.5us
Compares to ~2.5h for 500ns at transistor level**

Conclusions



- Interface Modeling methodology for CML circuits is proven
 - Allows Event Driven simulation
 - Dramatic simulation speed up
 - Transistor level (Verilog-A VCO) 4.5us est 1350 min ($2.5h * 4.5/0.5$)
 - 1 HS net analog 4.5us est 58min ($45min * 4.5/3.5$)
 - All event driven 4.5us ~33min ($11min * 4.5/1.5$)
 - Connection to analog circuits when needed – extracted nets
 - Includes drive strength model
 - Variety of interfaces
 - Interface compatibility is checked in simulation preparation.
 - Assertions check Supply and Bias connections.
- Enables a full chip control signal and bias verification simulation.

Thoughts for the Future



- Cross Sensitivity feature would be nice
 - Enables a CROSS on the differential inputs
 - Enables steering the output current between differential outputs
 - Rather than making a copy of the opposite polarity output in each connect module
- Can we remove the discipline declaration and connect model parameters from the connect rules?
 - Should be able to read these from the liberty/tlf file for each gate.
 - Cmos connect models should be able to use ECSM info from liberty file.
 - Can we get the liberty file format standardized? How about Spice?
- Could we standardize CML Gate and Signal Names?
 - Probably only if a significant Synthesis, Place and Route effort is made.
 - Very High Speed Signal Processors..
 - Power concerns drive significant custom design into this space, and low volume means CAD companies are not interested.