2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Automated Modeling Testbench Methodology Tested with four Types of PLL Models

Jun Yan, Josh Baylor

Renesas Electronics

accellera
SYSTEMS INITIATIVE

Agenda

- Modeling scope
- General modeling strategy
- PLL used in our experiment
- 4 types of PLL models
- Automated modeling testbench creation
- Testbench stimulus (STIM_HW, STIM_SW)
- Simulation results, comparison and conclusion

Modeling scope

- Purpose
  - Verify the concept of architecture before design is ready.
  - Improve simulation speed with behavior to emulate design.
  - Verify connectivity of design.
- Type of models
  - Analog: VerilogA/VerilogAMS
  - Discrete: Verilog
  - Real-number: SystemVerilog
    - EEnet
    - wreal

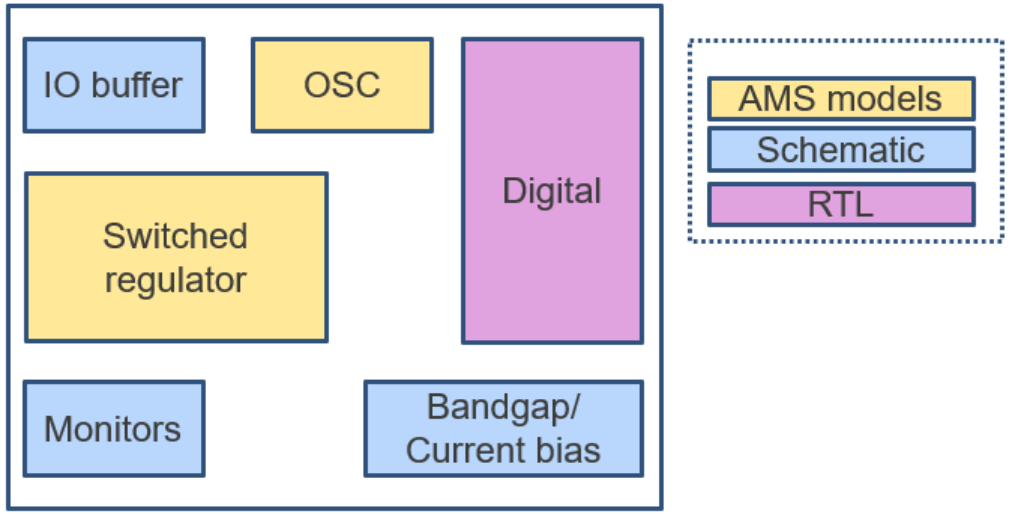
EEnet is a SystemVerilog User Defined Nettype provided by Cadence to emulate V/I/R impedance behavior in DMS simulation.

- Can model switching behavior for PLL/switching regulator.
- It is fast since it is run with DMS.

General modeling strategy

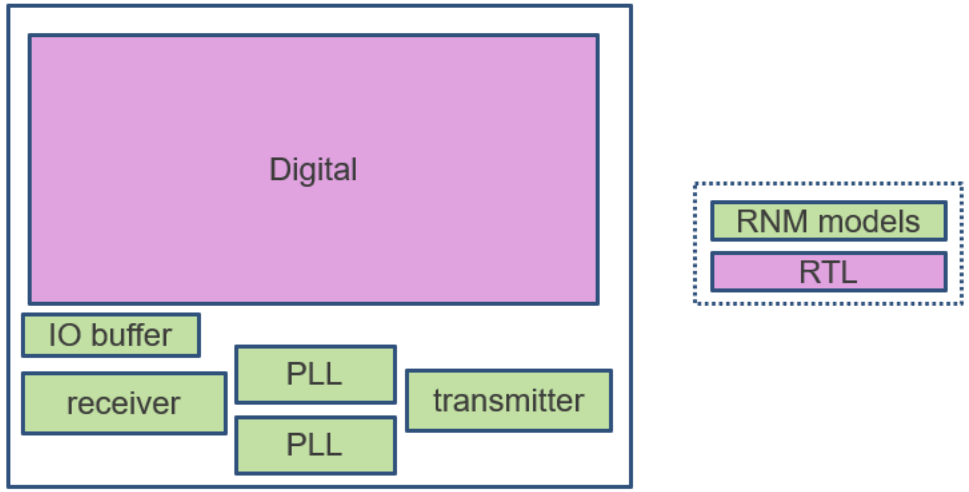
Analog heavy device: PMIC

- DV focus: verify analog design with AMS simulation
- Model switching blocks with AMS models



Digital heavy device: SerDes/VSP(Video Signal Processing)

- DV focus: verify digital design with DMS simulation
- Model all analog blocks with RNM models



PLL modeling in VSP

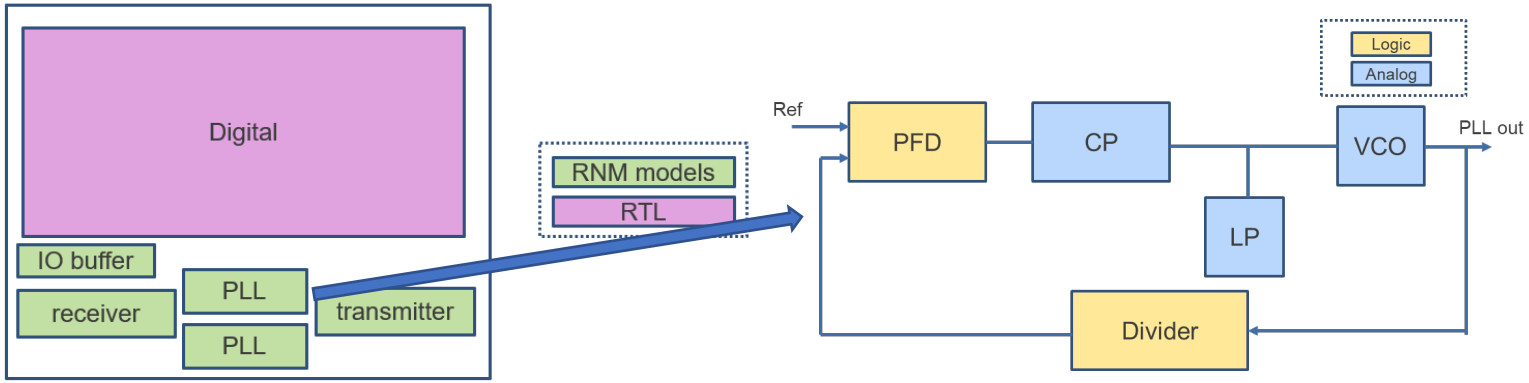
Models to choose

- AMS model with VerilogAMS
- RNM model with SystemVerilog
  - Close-loop model without EEnet
  - Close-loop model with EEnet
  - Open-loop model

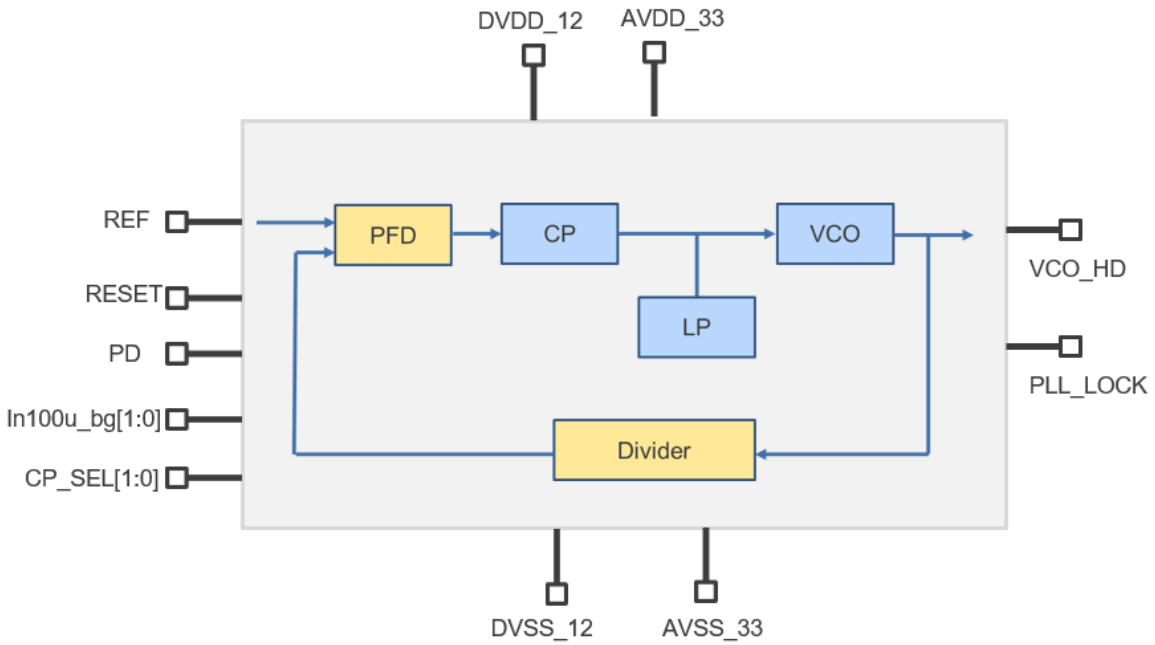
- Use the right/wrong modeling strategy can affect verification efficiency significantly.
- Different voices to support the use of all 4 types of models are heard within the team before the experiment.

How to evaluate models?

- Features included
- Development difficulty
- Change to schematic
- Accuracy of model
- Simulation speed.



PLL used in our experiment



Pin name	Domain	Direction	Function
DVDD_12	Analog	inout	Digital 1.2V supply
DVSS_12	Analog	inout	Digital ground
AVDD_33	Analog	inout	Analog 3.3V supply
AVSS_33	Analog	inout	Analog ground
In100u_bg[1:0]	Analog	inout	Bias current: 100uA
REF	Digital	input	Reference clock: 71.4MHz
RESET	Digital	input	Reset control
PD	Digital	input	Power down control
CP_SEL[1:0]	Digital	input	Select charge pump current
VCO_HD	Digital	output	PLL output: 7x input frequency
PLL_LOCK	Digital	output	Indicate if PLL has locked

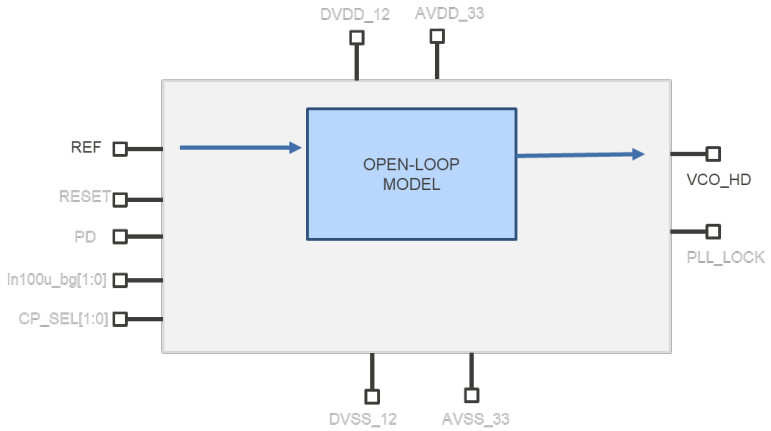
Model examples

Next, we'll describe the four examples:

- Open-loop real number model
- Closed-loop real number model
- VerilogAMS model
- Closed-loop real number model with EEnet

Open-loop Real Number Model

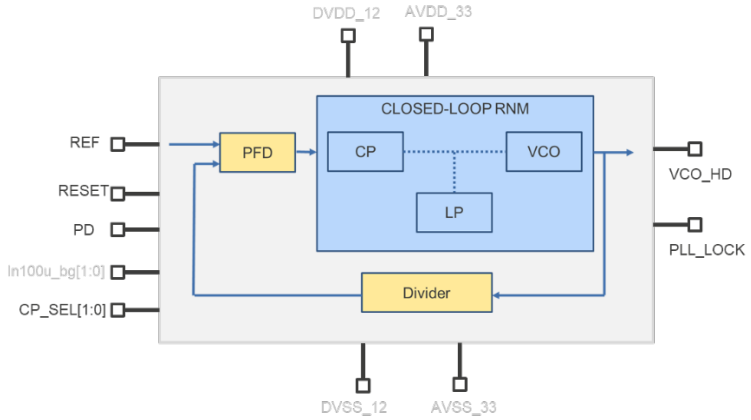
- Model entire PLL without knowing the design implementation.
- Measures input REF clock frequency with $realtime
- Calculates output frequency and then drive the output with a variable delay
- PLL settling response is not modeled




```
33  real time1, time2;
34  real period5x, period;
35  real freq;
36  int acc_cnt;
37
38  initial begin
39    acc_cnt =0;
40    period5x = 0;
41    time1 = 0;
42    time2 = 0;
43  end
44
45  always @(posedge REF) begin
46    time2 = time1;
47    time1 = $realtime;
48    if (time2 != 0 && acc_cnt < 5) period5x = period5x + time1 - time2;
49    acc_cnt = acc_cnt + 1;
50    if(acc_cnt == 5) begin
51      period = period5x / 5.0;
52      acc_cnt = 0;
53      period5x = 0;
54      freq = 1.0/period;
55    end
56  end
57
58  int fb_div=7;
59
60  reg clk_gen;
61  initial begin
62    clk_gen = 1'b0;
63  end
64
65  event sync;
66  always @(posedge REF) -> sync;
67
68  int i;
69  always @(sync)        begin
70    clk_gen = 1'b1;
71        for (i=0; i<fb_div * 2-1; i=i+1 ) begin
72                case(i)
73                        0: DIV_REF=1;
74                        6: DIV_REF=0;
75                endcase
76                # (period/(fb_div * 2)) clk_gen = !clk_gen;
77        end
78  end
79
```

Closed-loop RNM model

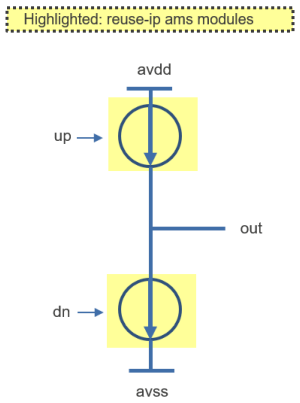
- PLL level schematic needs to be revised for modeling. Combines charge-pump, filter and VCO into a single RNM model block.
- Create a fixed frequency clock named “timeout”.
- At each “timeout” clock edge, re-evaluate the math equations between capacitor charge, voltage and time based on PFD output “up” and “dn” signals.



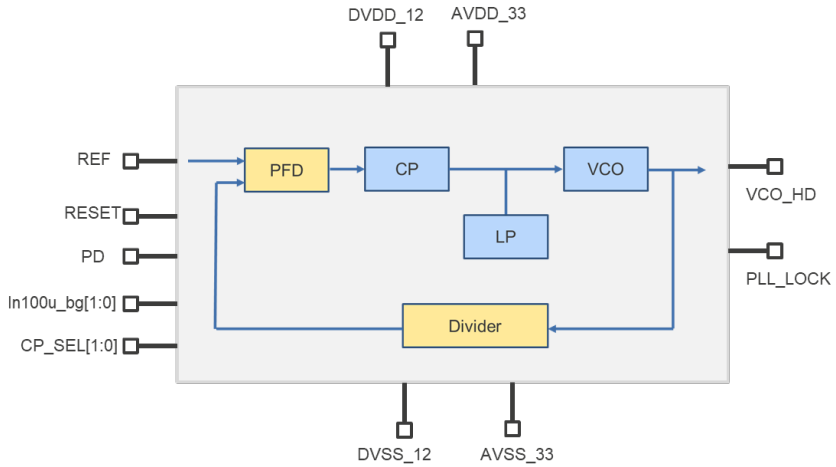

```
always #(timeout_length) timeout = ~timeout*(~pd)*(rstb); // gate timeout on power-down and resetb

always @(up,dn,vcoclk,timeout) begin

        R1_v = C2_v - C1_v;

        casex({state,timeout})
                3'b00x: begin
                                C2_dv = (C1_v - C2_v)*(1-exp(-($abstime - tlastevent)/(R1*C2)));
                                C2_v = C2_v + C2_dv;
                                C1_v = C1_v - C2_dv*C2/C1;
                                tlastevent = $abstime;
                        end
                3'b01x: begin
                                C1_v = C1_v + ($abstime - tlastevent)*(R1_v/R1)/C1;
                                C2_v = C2_v - (Icp+R1_v/R1)/C2*($abstime - tlastevent);
                                tlastevent = $abstime;
                        end
                3'b10x: begin
                                C1_v = C1_v + ($abstime - tlastevent)*(R1_v/R1)/C1;
                                C2_v = C2_v + (Icp-R1_v/R1)/C2*($abstime - tlastevent);
                                tlastevent = $abstime;
                        end
                3'b11x: begin
                                C2_dv = (C1_v - C2_v)*(1-exp(-($abstime - tlastevent)/(R1*C2)));
                                C2_v = C2_v + C2_dv;
                                C1_v = C1_v - C2_dv*C2/C1;
                                tlastevent = $abstime;
                        end
        endcase
        C1_v = C1_v<0 ? 0 : C1_v>vsupply ? vsupply : C1_v; //clamp cap voltage to supply
        C2_v = C2_v<0 ? 0 : C2_v>vsupply ? vsupply : C2_v; //clamp cap voltage to supply
        vctrl = C2_v;
        state = {up,dn};
end
```

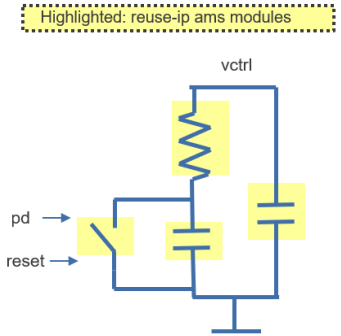
Verilog-AMS model

- Charge pump, loop filter and VCO are modeled separately. PLL model level schematic remained the same for AMS model.
- Pre-developed reuse-ips such as current source, capacitor and resistor are building blocks for the models.




```
1  `include "constants.vams"
2  `include "disciplines.vams"
3
4  module chargepump ( up, dn, avdd, avss, out );
5
6   `define dig_sns (* integer supplySensitivity = "avdd";\
7                      integer groundSensitivity = "avss"; *)
8   input avdd; electrical avdd; input avss; electrical avss;
9   input `dig_sns up; wire up;
10  input `dig_sns dn; wire dn;
11  output out; electrical out;
12
13  reg avdd_ok = 1; reg avss_ok = 1;
14  real i_cp = 10u;
15  wire pwr_ok;
16
17  assign pwr_ok = avdd_ok & avss_ok;
18
19  initial begin
20    I_UP.t_trans = 1n; I_DN.t_trans = 1n;
21  end
22
23  always@(X_CLK.clk) begin
24    avdd_ok = ((V(avdd) < 1.6) && (V(avdd) > 0.8))? 1:0; /
25    avss_ok = ((V(avss) < 0.1) && (V(avss) > -0.1))? 1:0;
26  end
28  always@(*) begin
29    if ((~pwr_ok) || (up&dn)) begin
30      I_UP.value = 0.0; I_DN.value = 0.0;
31    end
32    else if (up == 1'b1) begin
33      I_UP.value = i_cp; I_DN.value = 0.0;
34    end
35    else if (dn == 1'b1) begin
36      I_UP.value = 0.0; I_DN.value = i_cp;
37    end
38    else begin
39      I_UP.value = 0.0; I_DN.value = 0.0;
40    end
41  end
42
43  analog_clock X_CLK ();
44  src_cur_clamp I_UP      (avdd, out);
45  src_cur_clamp I_DN      (out, avss);
46
47
48 endmodule
```

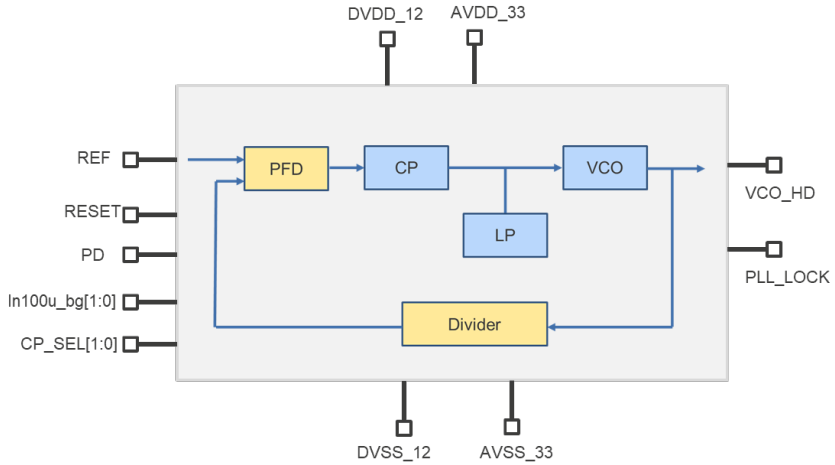





```
1  `include "constants.vams"
2  `include "disciplines.vams"
3
4  module loop_filter ( avss, dvdd, dvss, pd, reset, vctrl );
5
6   `define dig_sns (* integer supplySensitivity = "dvdd";\
7                      integer groundSensitivity = "dvss"; *)
8
9   input vctrl; electrical vctrl;
10  input dvss; electrical dvss;
11  input avss; electrical avss;
12  input dvdd; electrical dvdd;
13
14  input `dig_sns reset; wire reset;
15  input `dig_sns pd; wire pd;
16
17  electrical net_rc;
18
19  always@(*) begin
20    if (pd|reset) SW_RC.close;
21    else  SW_RC.open;
22  end
23
24  res_vams #(.value(1.526k)) R1 (vctrl, net_rc);
25  cap_vams #(.value(476p)) C1 (net_rc, avss);
26  cap_vams #(.value(3.52p)) C2 (vctrl, avss);
27  switch_vams #(.init_state(1),.ron(0.01),.roff(1G)) SW_RC (net_rc, dvss);
28 endmodule
```

Closed-loop RNM EEnet model

- Charge pump, loop filter and VCO are modeled separately. PLL model level schematic remained the same for AMS model.
- EEnet package components such as capacitor (CapGeq) used as the building blocks for the models.

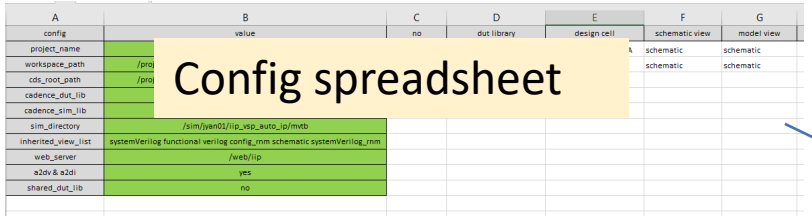


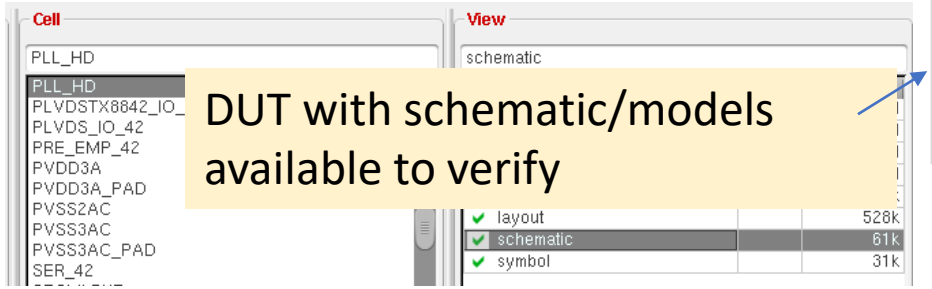

```
 8 module chargepump ( up, dn, avdd, avss, out );
 9
10   input wreal4state avdd;
11   input wreal4state avss;
12   input up;
13   input dn;
14   input pdb;
15   output EEnet out;
16
17   reg timeout=0; always #(5ns) timeout = ~timeout;
18
19   assign  avdd_ok = ((avdd < 1.6) && (avdd > 0.8))? 1:0;
20   assign  avss_ok = ((avss < 0.1) && (avss > -0.1))? 1:0;
21   assign  i10u_ok = ((i10u < 12e-6) && (i10u > 8e-6))? 1:0;
22   assign pwr_ok = avdd_ok & avss_ok;
23   assign int_en = pwr_ok & pdb;
24
25   real vmid, rout, iout;
26
27   initial begin
28     iout = 0;
29     vmid = `wrealZState;
30     rout = `wrealZState;
31   end
32
33   always @(up,dn, int_en, timeout) begin
34     if(int_en) begin
35        if ((up<dn) && (out.V<=0.0)) iout = 0.0;
36        else iout = up*10e-6 - dn*10e-6;
37     end
38     else iout = 0.0;
39   end
40
41   assign out = '{vmid, iout, rout};
42 endmodule
```

```
 1 import cds_rnm_pkg::*;
 2 import EE_pkg::*;
 3
 4 module loop_filter ( avss, dvdd, dvss, pd, reset, vctrl );
 5
 6   input EEnet vctrl;
 7   input wreal4state dvdd;
 8   input wreal4state dvss;
 9   input pd;
10   input reset;
11
12   assign dvdd_ok = ((dvdd < 1.6) && (dvdd > 0.8))? 1:0; // 1.2V
13   assign dvss_ok = ((dvss < 0.1) && (dvss > -0.1))? 1:0; // 0V
14   assign pwr_ok = dvdd_ok & dvss_ok;
15   assign int_en = pwr_ok & (~pd) & (~reset);
16
17   CapGeq #(.c(476e-12), .tinc(5e-9), .rs(1.526e+3)) filtcap1(vctrl);
18   CapGeq #(.c(3.52e-12), .tinc(5e-9)) filtcap2(vctrl);
19 endmodule
```

Automated modeling testbench creation

- Config spreadsheet: specify setup info such as Cadence library path and design blocks (DUTs) names.
- Automation (SKILL):
  - Extract DUT pins from Cadence symbol.
  - Create stimulus module template
  - Create testbench



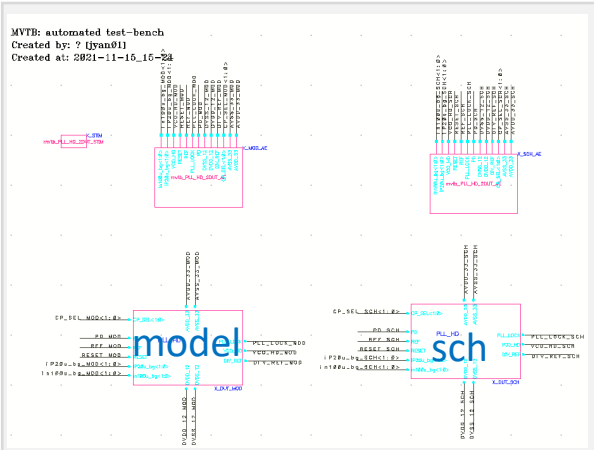


Automated Modeling Testbench script

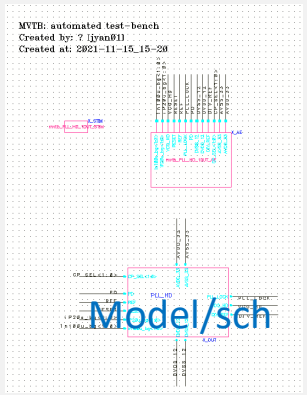
Python (call SKILL command inside)

2 DUT option

1 DUT option

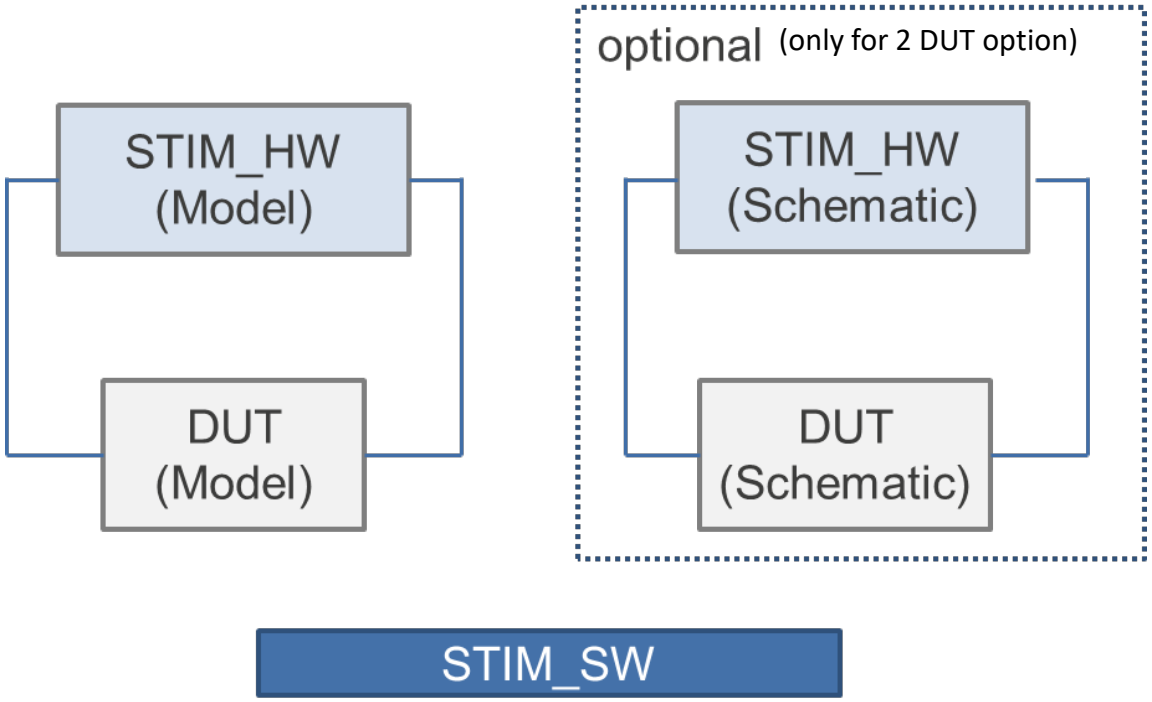


Common approach, suitable for DUT schematic run with fast speed.



More efficiently if need to debug and revise model code for switching DUT.

Automated modeling testbench creation



STIM_HW:

- Stimulus hardware module
- Contain hardware components such as voltage drivers, capacitors, resistors, switches to emulate silicon validation PCB board.
- Ports one to one map to DUT
- In VerilogAMS

STIM_SW:

- Stimulus software module
- Use test sequence to control STIM_HW components by calling the tasks defined in those components
- No ports
- In SystemVerilog

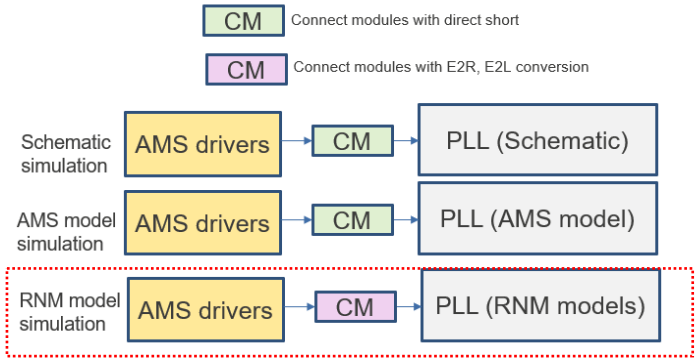
Next: details about two modules

Module template such as ports declaration is automated, but content of the module still need to be hand edited.

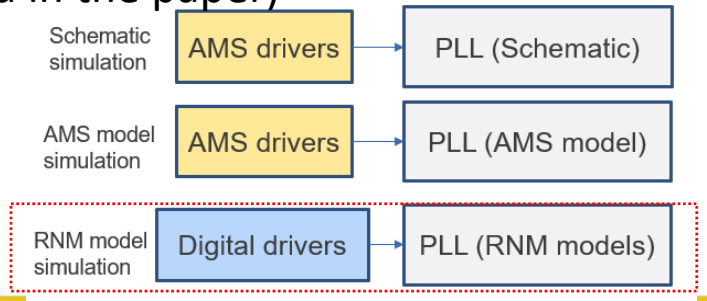
STIM_HW

Challenge: how to choose the right drivers for different models.

Method 1: use same AMS drivers for AMS/RNM models + CM



Method 2: use different AMS/digital drivers for different models (used in the paper)



Solution with method 2

```
7  `ifdef config_schematic
8    `define ams_drivers
9  `elsif config_ams_model
10   `define ams_drivers
11 `elsif config_rnm_closed_loop_model
12   `define digital_drivers
13 `elsif config_rnm_open_loop_model
14   `define digital_drivers
15 `elsif config_rnm_eenet_model
16   `define digital_drivers
17 `endif
18
19 module STIM_HW__PLL(
20 PLL_LOCK,VCO_HD,AVDD_33,AVSS_33,CP_SEL,DVDD_12,DVSS_12,PD,REF,RESET,in100u_bg
21 );
22 // ...
23
24 `ifdef ams_drivers
25 // ...
26 // PLL STIM_HW with AMS drivers
27
28 `elsif digital_drivers
29 // ...
30 // PLL STIM_HW with digital drivers
31
32 `endif
33 //...
```

Map between drivers and different types of models (defined in configs)

Code for two drivers show in the next

STIM_HW

AMS drivers

- Drive analog signal: AMS modules (voltage/current driver)
- Drive digital signal: Set on “reg” and assign to “wire” + supply sensitivity

```
66   // supply and bias pins
67   volt_driver V_AVDD_33 (AVDD_33, GND);
68   volt_driver V_DVDD_12 (DVDD_12, GND);
69   current_driver I_in100u_bg_1 (in100u_bg[1], GND);
70   current_driver I_in100u_bg_0 (in100u_bg[0], GND);
71   gnd_vams X_GND_AVSS_33 (AVSS_33);
72   gnd_vams X_GND_DVSS_12 (DVSS_12);
73
74   // digital input
75   wire [1:0] (* integer supplySensitivity = "DVDD_12"; integer groundSensitivity = "DVSS_12"; *)  CP_SEL;
76   reg [1:0] CP_SEL_dig = 2'b00;
77   assign CP_SEL[1:0]=CP_SEL_dig[1:0];
78   wire (* integer supplySensitivity = "DVDD_12"; integer groundSensitivity = "DVSS_12"; *) PD;
79   reg PD_dig = 1'b0;
80   assign PD=PD_dig;
81   wire (* integer supplySensitivity = "DVDD_12"; integer groundSensitivity = "DVSS_12"; *) REF;
82   reg REF_dig = 1'b0;
83   assign REF=REF_dig;
84   wire (* integer supplySensitivity = "DVDD_12"; integer groundSensitivity = "DVSS_12"; *) RESET;
85   reg RESET_dig = 1'b0;
86   assign RESET=RESET_dig;
87
88   // digital output
89   wire (* integer supplySensitivity = "DVDD_12"; integer groundSensitivity = "DVSS_12"; *) PLL_LOCK;
90   wire (* integer supplySensitivity = "DVDD_12"; integer groundSensitivity = "DVSS_12"; *) VCO_HD;
```

Digital ports (wire/inout type)

Internally created reg type signal

Digital drivers

- Drive analog signal: SystemVerilog modules (wreal driver)
- Drive digital signal: Set on “reg” and assign to “wire”

```
65   wreal_driver V_AVDD_33 (AVDD_33);
66   wreal_driver V_DVDD_12 (DVDD_12);
67   wreal_driver I_in100u_bg_1 (in100u_bg[1]);
68   wreal_driver I_in100u_bg_0 (in100u_bg[0]);
69
70   reg [1:0] CP_SEL_dig = 2'b00;
71   assign CP_SEL[1:0]=CP_SEL_dig[1:0];
72   reg PD_dig = 1'b0;
73   assign PD=PD_dig;
74   reg REF_dig = 1'b0;
75   assign REF=REF_dig;
76   reg RESET_dig = 1'b0;
77   assign RESET=RESET_dig;
```

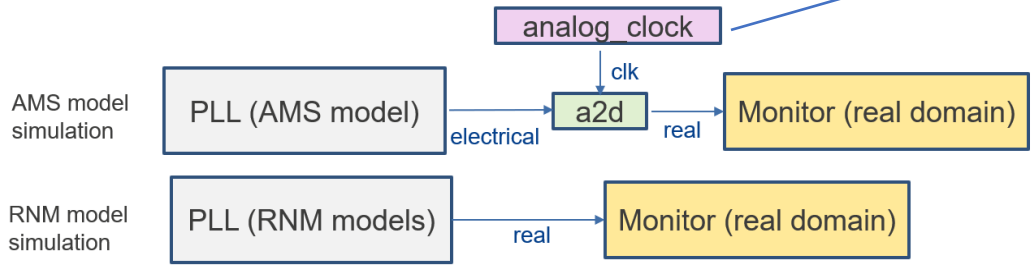
STIM_SW

- STIM_SW is shared between AMS drivers and digital drivers
  - Same tasks name/arguments for corresponding driver modules such as volt_driver(AMS) and wreal_driver(digital): such as “setv” with arguments of “value” and “ramptime”
  - Same instance name for corresponding drivers used in AMS/digital: such as V_AVDD_33, I_in100u_bg_0

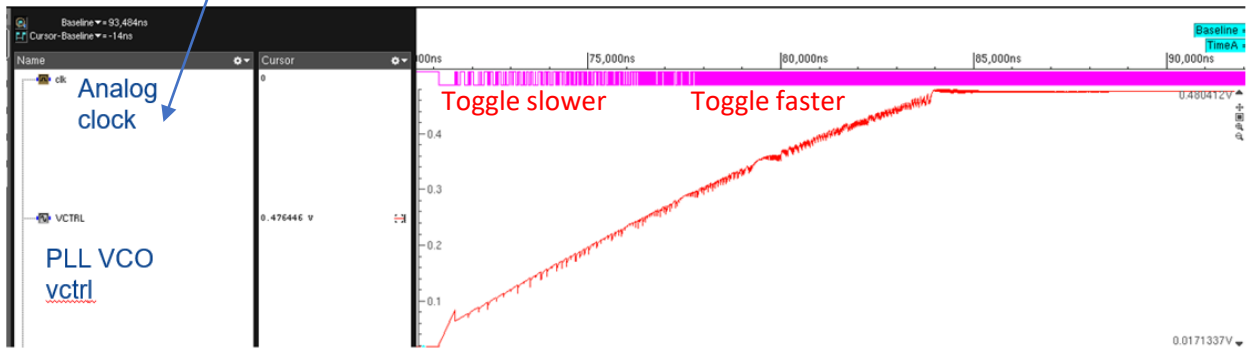
```
 9 module STIM_SW__PLL();
10
11 reg ena_ref=0;
12 real freq = 75e+6;
13 real t_period;
14
15 initial begin
16   t_period = 1.0/freq;
17
18   `log("Ramp supplies");
19   `STIM_HW.V_AVDD_33.setv(.value(3.3), .ramptime(10e-6));
20   `STIM_HW.V_DVDD_12.setv(.value(1.2), .ramptime(10e-6));
21   `STIM_HW.I_in100u_bg_0.seti(.value(10e-6), .ramptime(10e-6));
22   `STIM_HW.I_in100u_bg_1.seti(.value(10e-6), .ramptime(10e-6));
23
24   `log("Set logic input");
25   `STIM_HW.CP_SEL_dig[1:0]=2'b01;
26   `STIM_HW.PD_dig=1'b0;
27   `STIM_HW.RESET_dig=1'b0;
28
29  `log("Enable REF clock");
30   ena_ref=1'b1;
31   #(30us);
32
33   $finish;
34 end
35
36 always @(ena_ref) begin
37    if (ena_ref) begin
38      forever #(t_period/2) `STIM_HW.REF_dig = ~`STIM_HW.REF_dig;
39    end
40    else `STIM_HW.REF_dig = 0;
41 end
42 endmodule
```

Output monitoring

- Monitors (written in SV): take the real type input and measure min/max/peak to peak.
- For RNM, monitors can be connected directly to PLL output.
- For AMS model, A2D conversion is needed to:
  - Method 1: use absdelta function for each electrical output (commonly used)
  - Method 2: use analog_clock to sample each electrical output. (used in this paper)




```
1  `include "disciplines.vams"
2
3  module analog_clock();
4
5      real clk_a;
6      reg clk = 1'b0;
7
8      always @(absdelta(clk_a, 10m, 10p, 1m)) begin
9          if ((clk_a == 1.0) || (clk_a == 0.0)) begin
10             clk = ~clk;
11         end
12     end
13
14     analog begin
15         clk_a = 1.0 - clk_a;
16     end
17
18 endmodule
```



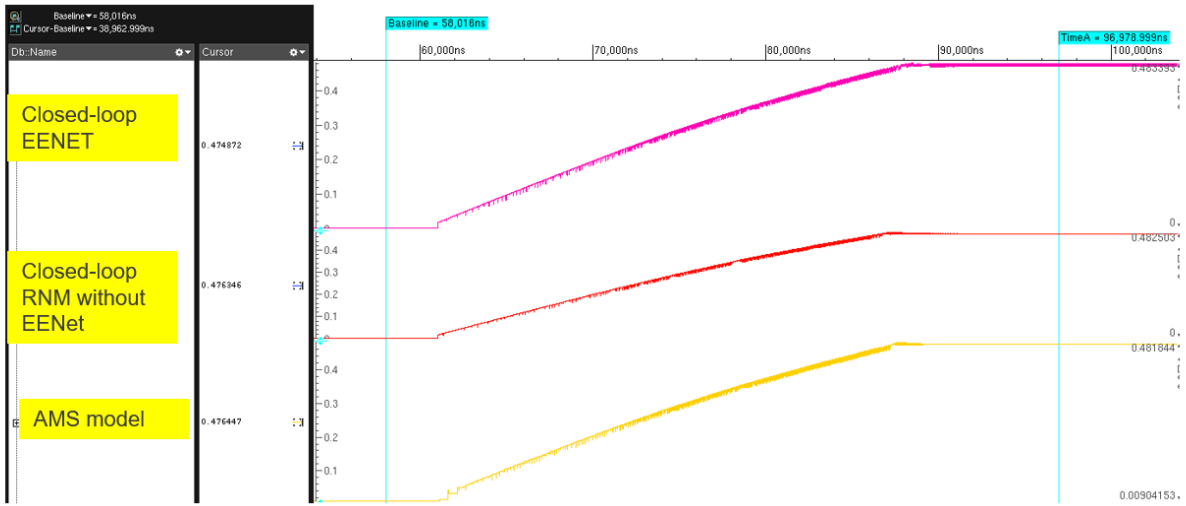
"clk" created in analog_clock module align with all analog time steps. Use "clk" to sample electrical signals.

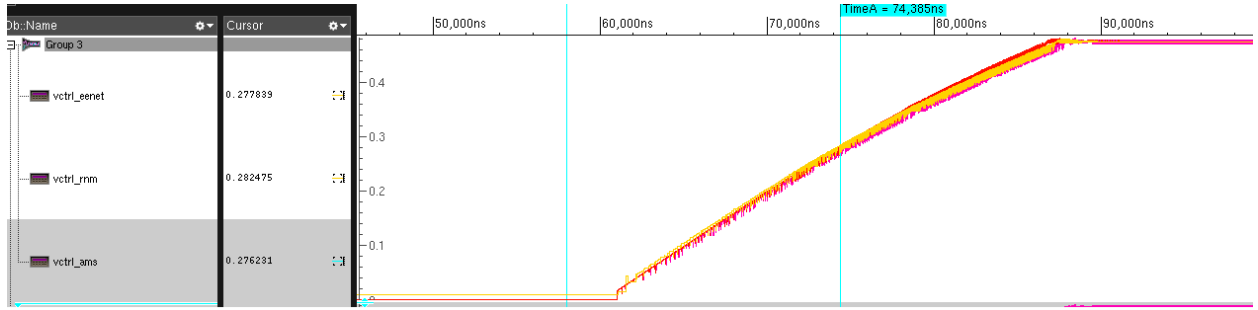
1. High accuracy of data conversion
2. Doesn't slow down in the simulation Since no extra analog time steps are added.

Simulation results

- Use VCO control signal to compare accuracy of three closed-loop models.
- The overlay view shows the three models correlate very well between each other.

VCO control voltage





Simulation results

- Complete comparison for 4 models. (rating: **high**, **medium**, low)

Winner of models for heavy digital simulation.

	AMS model	Open-loop RNM	Closed-loop RNM	Closed-loop RNM with EENet
Development difficulty	Difficult (deal with convergence issue and slow sim optimization)	Easy	Difficult (deal with mathematical equation)	Medium
Change to schematic	No change	Yes, combine all blocks into one model file	Yes, combine charge pump, loop filter.	No change
Accuracy	Highest	Lowest	High	High
Sim time in individual testbench	14 min (can be reduced based on optimization)	~1 second	~1 second	~1 second
Sim time in a top-level verification test: process several video frames	Not tested	16 min 10 s (bench mark)	22 min 18s	21 min 35s

Conclusion

- We are proposing a way to automate the modeling testbench creation.
- We talk about how we create models and stimulus modules, and we compare models from different angles.
- For heavy digital verification, we choose closed-loop RNM EEnet model because:
  - Fast simulation.
  - Accurate: fully model PLL settling response and configuration through control ports.
  - No need to modify PLL schematic to create the model. Design connectivity can be verified.
  - Not difficult to create by using EEnet package provided by Cadence.

Questions

- Thanks for attending today’s presentation!
- Any questions?