# Automated Modeling Testbench Methodology Tested with four Types of PLL Models

Jun Yan, Josh Baylor
Renesas Electronics

***Abstract*-Models are widely used in verification testbenches to solve different challenges in mixed signal verification but choosing the right model can be a tough decision for verification engineers. We may choose different types of models for design with heavy digital or heavy analog. Even with the same design, we may also choose different types of models for different tests. Choosing the right modeling strategy will make the verification work easier and more efficient, while choosing the wrong modeling strategy can potentially add up verification effort such as modeling maintenance and can impact verification work schedule. In this paper, we will talk about an experiment of how to choose the right modeling strategy for a VSP (Video Signal Processing) design with multiple PLLs, and we simulate the same PLL block with an AMS model, an open-loop RNM, a closed-loop RNM without EENet, a closed-loop RNM with EENet. In the end of the paper, we will show our observations and conclusion based on the experiment.**

## I. INTRODUCTION

Models are widely used in verification testbenches to solve different challenges in mixed signal verification, but choosing the right model can be a tough decision for verification engineers. We may choose different types of models for design with heavy digital or heavy analog. Even with the same design, we may also choose different types of models for different tests. Choosing the right modeling strategy will make the verification work easier and more efficient, while choosing the wrong modeling strategy can potentially add up verification effort such as modeling maintenance and can impact verification work schedule. In this paper, we will talk about an experiment of how to choose the right modeling strategy for an VSP (Virtual Signal Processing) design with multiple PLLs, and we simulate the same PLL block with an AMS model, an open-loop RNM, a closed-loop RNM without EENet, a closed-loop RNM with EENet. In the end of the paper, we will show our observation and conclusion based on the experiment.

## II. PURPOSE AND OPTIONS FOR MODELING

### *A. Purpose of modeling*

In mixed signal verification, there are many reasons to use models in the simulation, and we name critical ones as below.

First, models can help verify the concept of architecture before the design is put in. The models can be written with different languages and simulated with different tools. It can be Matlab models, Pspice models or Verilog/VerilogA/SystemVerilog/VerilogAMS models. Since in our application, we have already had design ready, so the modeling for architecture is out of scope for our discussion in this paper.

Second, models can help improve the simulation speed with behavior to emulate the actual design. This is the most critical benefit of modeling used in mixed signal verification. Good modeling strategy can help us reduce the verification development cycle time, catch design bugs early, and reduce the cost of simulator license and computing resources.

Third, models can help verify the connectivity of the design. This benefit is especially important for PLL closed-loop modeling. Basically, the idea is we want to keep the schematic as it is unless it touches transistor level. Doing so, we can verify the connectivity of the design and catch the related bugs, including the controlling signals from block ports, inter-connectivity between the sub-modules in the PLL design.

### *B. General modeling strategy*

Which models to choose can depend on the types of design and types of tests to run.

VerilogAMS is very common to use in analog heavy design projects, such as PMIC (Power Manager IC), Since primary purpose of verification is used to verify the correctness of the design, so we need to use design as much as possible. We only build VerilogAMS models for switching blocks such as oscillators and switching regulators to make most verification tests finish within a reasonable time range, typically from 30 minutes to 2 hours based on the complexity of the design. Another good benefit of VerilogAMS models is that they can be connected directly to

other device modules with actual design in transistor level. Basically, they can be easily plugged into a design, and this is a big advantage over RNM. Another benefit is that VerilogAMS supports electrical signal type and can accurately reflect the loading effect compared to RNM. The drawback of VerilogAMS models is that they are slower than RNM when using the electrical type since they invoke the analog solver. When VerilogAMS models are running along with transistor-level schematic views, the simulation speed is usually limited by the schematic views instead of the VerilogAMS model views. For a small percentage of verification tests, we also want to set the switching blocks in the design view so we can verify the design of those blocks as well. Figure 1 shows an example of a common config used in PMIC device.

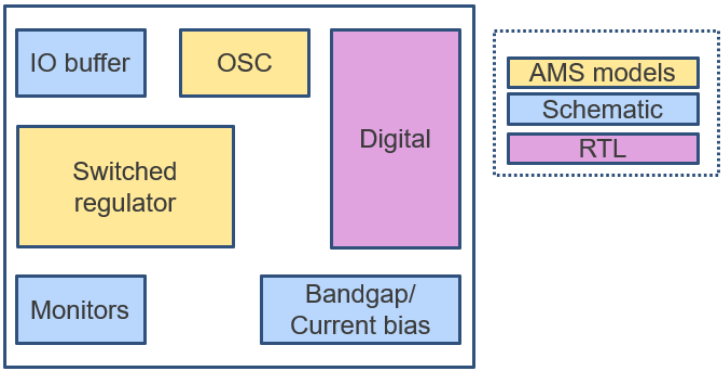



Figure 1. AMS modeling strategy for PMIC device

RNM modeling is very common to use in digital heavy design projects, such as SerDes (Serializer/De-serializer). Since the digital core is big with complex digital signal processing, the primary goal is to verify the algorithms and design configurations within the RTL code. The modeling requirement for analog blocks such as buffers, PLLs, receivers and transmitters, is to emulate the design behavior as much as possible without slowing down the simulation. For example, assume we have a verification test finishes in 1 minute when running with digital core only. By adding analog block models to enable mixed-signal co-simulation from top level, if the simulation time only increases by a few seconds is acceptable while increasing by a few minutes can be too long. So, in this application, RNM models instead of VerilogAMS models can be used here to verify the digital core of the design by adding small overhead to the simulation time. Figure 2 shows an example of a common config used in SerDes device.

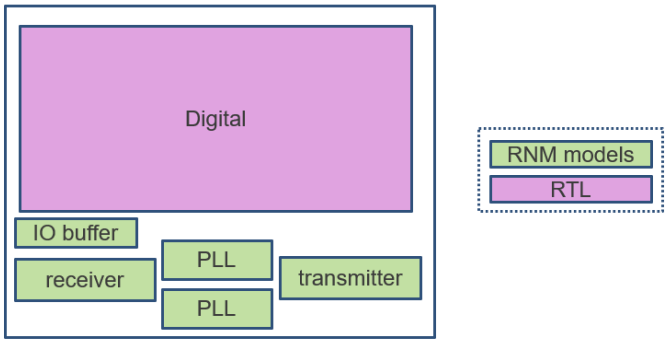



Figure 2. RNM modeling strategy for SerDes device

### *C. Our use case: PLL modeling*

Our VSP (Video Signal Processing) design discussed in this paper is very similar to the SerDes device in Figure 2. Among the analog blocks, PLLs are the most challenging blocks for modeling. We have several types of PLL. One type of PLL is used in the receiver to de-serialize the incoming data, another type of PLL is used to create a free-running high speed clock for signal processing, and yet another type of PLL is used in transmitter to serialize the data and clock the output. The architectures of the PLLs are very similar except some have fixed frequency divider ratios and others require multiple divider ratios and spread spectrum clocking. A conceptual block diagram of the

PLL is shown below in Figure 3. The PFD and frequency divider are designed with pure logic gates and flip-flops, so we can use the primitive SystemVerilog models for those logic components without additional effort to model. The challenges are how to model CP (charge pump), LF (Loop Filter) and VCO (voltage control oscillator) to achieve an optimum balance of simulation speed and accuracy.

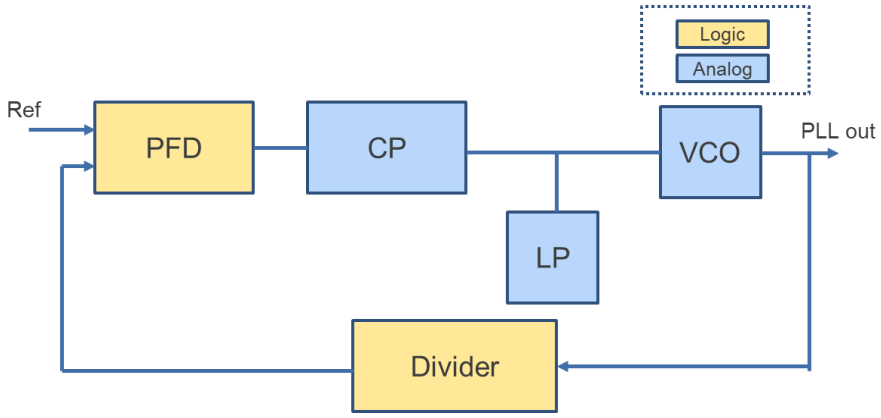



Figure 3. A typical PLL design

Though using RNM in modeling for PLL is a common choice in this type of digital device, we still have questions are below:

a. Which type of RNM should we use? Simple open-loop RNM model, complex closed-loop RNM model or advanced closed-loop RNM model with EENet?
b. Can we use Verilog-AMS model instead of RNM if it is not adding much penalty to the simulation speed?

After doing the experiment of the four types of PLL modeling, we will give our conclusion in the end.

## III. Automated Modeling Testbench

Since we might have multiple types of models for a block like the PLL in our example, we need to check the correctness of those models. We have created an automated flow to standardize and facilitate the model testbench creation process (Fig. 4).

First, we need to fill in a config spreadsheet to specify where the target blocks are located including Cadence library path/name and cell view name. Then the flow uses a SKILL script to extract the list of pins from the symbol of the target block and automate the testbench in schematic view with SKILL. The testbench creation has two options: single DUT and two DUT. With the two DUT option we can check model against schematic or compare two different models.

As for the testbench, we have STIM_HW (hardware module) which instantiates the hardware components such as voltage drivers, capacitors, resistors, and supplies to emulate silicon validation PCB board. STIM_HW is written in VerilogAMS which allows us to use electrical signal handling inside the module. STIM_HW has the same pins and connects to the DUT directly. In addition, STIM_SW module is used to control the hardware components inside STIM_HW, such as ramp voltage source to power up, and enable the DUT through pins. Also, measurement and checkers are implemented inside STIM_SW, and STIM_SW is written with SystemVerilog. STIM_HW and STIM_SW module skeleton code and testbench schematic are automated by the script, and users need to add content to STIM_HW and STIM_SW to complete the test.

The automated model testbench methodology allows us to check modeling of multiple critical blocks within a project in the most effective and efficient way, which ensures us to have reliable approach to verify models.

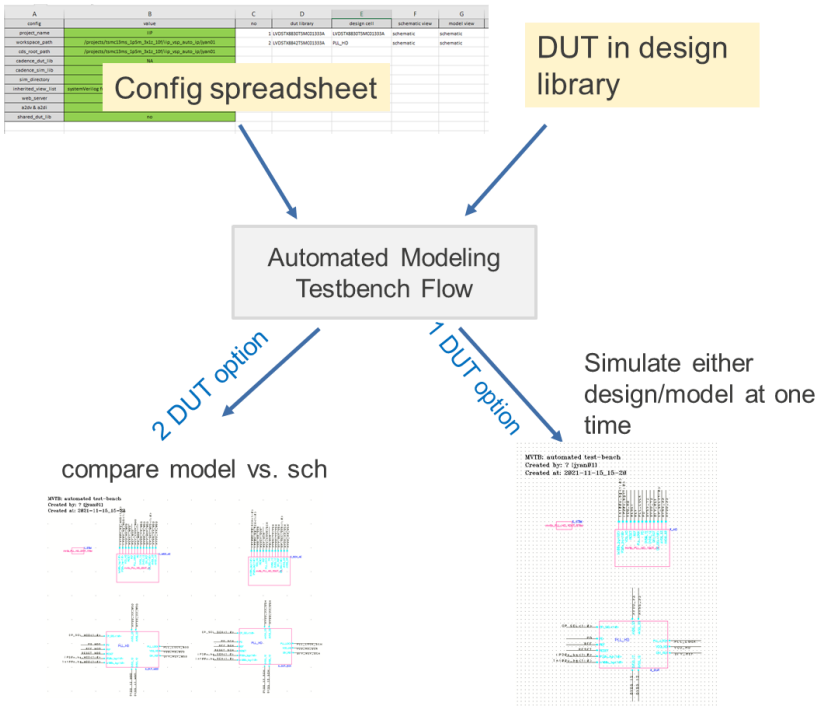



Figure 4. Automated Modeling Testbench flow diagram

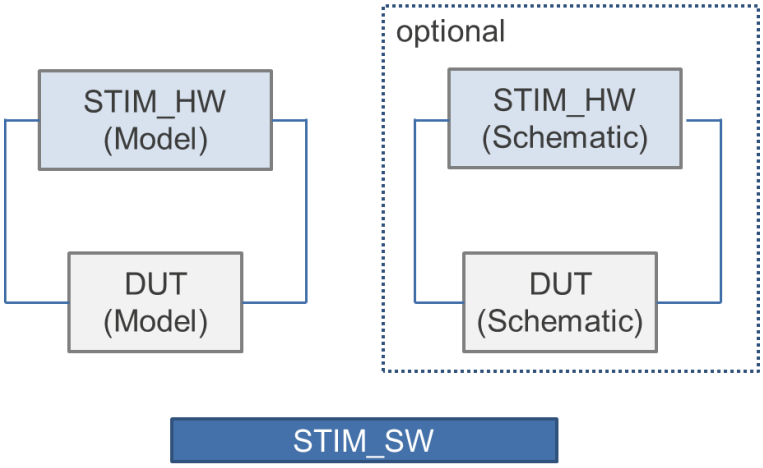



Figure 5. Modeling testbench structure

### *D. 1 DUT versus 2 DUT options*

Traditionally in modeling verification, people tend to put model and schematic side by side in the same testbench because it seems to be straightforward, and it is easier to compare the output of model and schematic in this way. For switching designs like PLLs or switching regulators, the simulation of the schematic design by itself can take a long time, potentially taking hours to run from startup to the settled state. In comparison, AMS models or RNM models may take only seconds or minutes to finish. One purpose of model verification is to repeatedly update model code to make it match the schematic behavior, especially when the schematic design has been delivered by analog designers and verified in analog Spectre testbench. In this scenario, the 1 DUT option is a more efficient solution.

Basically, the process of developing models with the 1 DUT testbench option is:

- Create 1 DUT testbench and use DUT as schematic in the testbench config view. Instantiate the hardware components in STIM_HW and develop software stimulus to exercise DUT inputs until the correct outputs of DUT are observed. The simulation might be long for switching design.

- Within the same testbench, use DUT as model in a different config view. Run simulation with preliminary DUT model, and plot model outputs. If the model outputs behavior differently from the previous schematic simulation, we may need to update the model code and rerun or leave as it is if the difference is out of scope for the features to be modeled. Since the simulation time is fast by simulating model by itself, so we can afford multiple repeated runs and iterations of models within a given work time.

### *E. PLL design explanation*

To help readers understand the testbench code explained in the rest of the paper, we give a brief introduction to the PLL design, especially the in/out ports.

This PLL design has reference clock to be 71.4MHz (period = 14ns), and frequency divider ratio as 7, so the expected PLL output is 500MHz after settled down.

TABLE I
PLL IN/OUT PORTS

| 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 | Pull down control |
| CP_SEL[1:0] | Digital | input | Select charge pump current |
| VCO_HD | Digital | output | PLL output: should be 500MHz (frequency divider ratio=7) |
| PLL_LOCK | Digital | output | Indicate if PLL has locked |

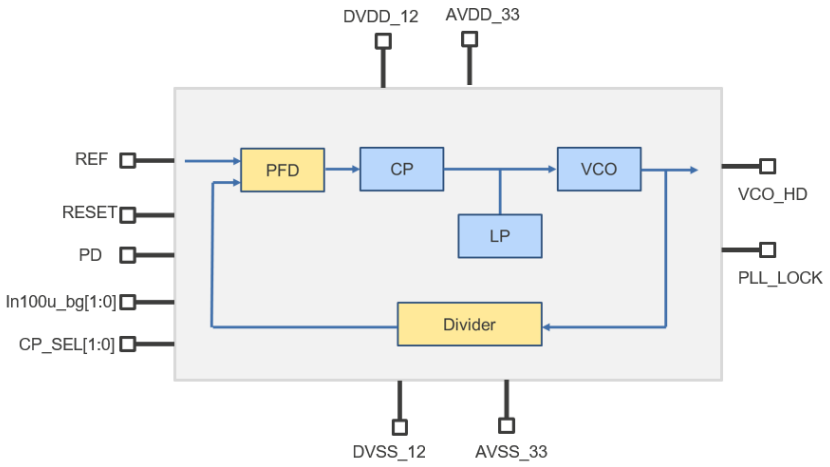



Figure 6. PLL diagram

### *F. STIM_HW (hardware)*

STIM_HW is used to put testbench hardware components and connectivity, such as voltage/current driver, ground connectivity, resistors, capacitors, and inductors. Besides that, for AMS simulation, we also use STIM_HW to define the supply sensitivity for all digital pins, which means we need to tell simulator the supply/ground level for digital pins to enable the correct A2D or D2A conversion during the simulation.

Figure 7 shows all ports in STIM_HW which match the DUT ports for the PLL. You may notice that all ports are inout type. Since inout type ports can be connected to input, output or inout ports of other modules, choosing inout type helps with the consistency of the testbench creation.

```
 7 module STIM_HW__PLL(
 8 PLL_LOCK,VCO_HD,AVDD_33,AVSS_33,CP_SEL,DVDD_12,DVSS_12,PD,REF,RESET,in100u_bg
 9 );
10
11     inout PLL_LOCK;
12     inout VCO_HD;
13     inout AVDD_33;
14     inout AVSS_33;
15     inout [1:0] CP_SEL;
16     inout DVDD_12;
17     inout DVSS_12;
18     inout PD;
19     inout REF;
20     inout RESET;
21     inout [1:0] in100u_bg;
```

Figure 7. PLL STIM_HW (schematic or AMS model simulation)

Since we are building STM_HW for DUT in the view of schematic, AMS model, RNM models, simulation compatibility such as how to handle the signal conversion in different domain is a challenge which we need to deal with. There are two solutions that we can choose to use.

Figure 8 shows solution A. Since PLLs in either schematic view or AMS model view have signals in electrical domain, the same AMS drivers written with VerilogAMS language can drive both. For PLL in RNM models, all signals are in digital domain, so we need digital drivers instead.

Figure 9 shows solution B. By inserting CM (connect modules) between STIM_HW and PLL DUT, we can use the same AMS drivers to drive PLL in different model views. For example, since AMS driver for VDD pins are creating output in electrical domain, if we connect the AMS driver to PLL DUT as AMS model, which has VDD pins as electrical as well, we just need to configure CM as the direct connect view. However, if it is connected to VDD in RNM of PLL DUT, which has VDD pin in the real type, then CM needs to be configured as E2R (electrical to real conversion) view.

The benefit of solution B is that we can use the same hardware drivers for all model configs. However, adding and configuring the CMs adds to the complexity of the modeling verification setup. Then we decide to choose solution A.

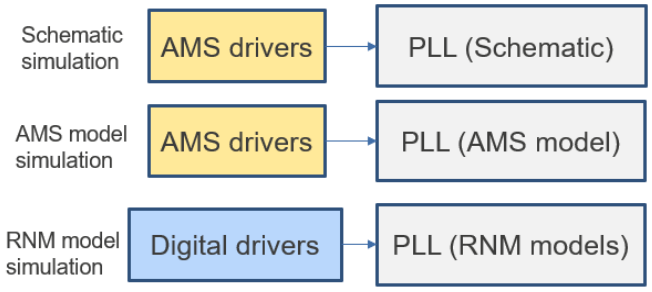



Figure 8. STIM_HW solution A

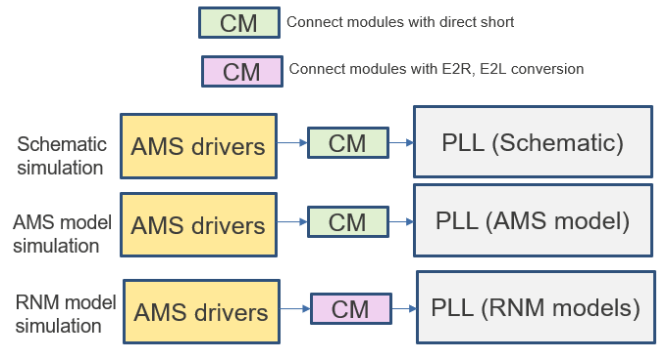



Figure 9. STIM_HW solution B

In solution A, since we have multiple testbench config views choosing two sets of drivers, either AMS drivers or digital drivers, we are using the example code in Figure 10 to show how to choose different drivers for different config views.

```
`ifdef config_schematic
  `define ams_drivers
`elsif config_ams_model
  `define ams_drivers
`elsif config_rnm_closed_loop_model
  `define digital_drivers
`elsif config_rnm_open_loop_model
  `define digital_drivers
`elsif config_rnm_eenet_model
  `define digital_drivers
`endif

module STIM_HW__PLL(
PLL_LOCK,VCO_HD,AVDD_33,AVSS_33,CP_SEL,DVDD_12,DVSS_12,PD,REF,RESET,in100u_bg
);
// ...

`ifdef ams_drivers
// ...
// PLL STIM_HW with AMS drivers

`elsif digital_drivers
// ...
// PLL STIM_HW with digital drivers

`endif
//...
```

Figure 10. choose drivers for different config views

The code for AMS drivers is shown in Figure 11. There are several primitive VerilogAMS reuse-ip modules used in code. “volt_driver” is an AMS voltage source module. “current_driver” is an AMS current source module. “gnd_vams” is an AMS ground module. Those AMS modules are used to connect the supply/bias pins.

For digital input and output pins, we need to specify the supplySensitivity and groundSensitivity to enable correct A2D conversions by the simulator. In addition, we add a corresponding “_dig” version with “reg” type for each digital input pin. For example, for RESET pin, we added a signal named RESET_dig with “reg” type. The reason is that all digital pins are with “inout” type are with “wire” type so we cannot directly set the pin level to be 0 or 1. So we can set the value of RESET_dig with “reg” type and assign RESET_dig to RESET pin with “wire” type.

```
  // supply and bias pins
  volt_driver V_AVDD_33 (AVDD_33, GND);
  volt_driver V_DVDD_12 (DVDD_12, GND);
  current_driver I_in100u_bg_1 (in100u_bg[1], GND);
  current_driver I_in100u_bg_0 (in100u_bg[0], GND);
  gnd_vams X_GND_AVSS_33 (AVSS_33);
  gnd_vams X_GND_DVSS_12 (DVSS_12);

  // digital input
  wire [1:0] (* integer supplySensitivity = "DVDD_12"; integer groundSensitivity = "DVSS_12"; *)  CP_SEL;
  reg [1:0] CP_SEL_dig = 2'b00;
  assign CP_SEL[1:0]=CP_SEL_dig[1:0];
  wire (* integer supplySensitivity = "DVDD_12"; integer groundSensitivity = "DVSS_12"; *) PD;
  reg PD_dig = 1'b0;
  assign PD=PD_dig;
  wire (* integer supplySensitivity = "DVDD_12"; integer groundSensitivity = "DVSS_12"; *) REF;
  reg REF_dig = 1'b0;
  assign REF=REF_dig;
  wire (* integer supplySensitivity = "DVDD_12"; integer groundSensitivity = "DVSS_12"; *) RESET;
  reg RESET_dig = 1'b0;
  assign RESET=RESET_dig;

  // digital output
  wire (* integer supplySensitivity = "DVDD_12"; integer groundSensitivity = "DVSS_12"; *) PLL_LOCK;
  wire (* integer supplySensitivity = "DVDD_12"; integer groundSensitivity = "DVSS_12"; *) VCO_HD;
```

Figure 11. PLL STIM_HW with AMS drivers (schematic or AMS model simulation)

The code for digital drivers is shown in Figure 12. Since both supply voltage pins and bias current pins in RNM only have one property, voltage or current, they can be connected directly to the same wreal_driver, which is used to generate and control a real type output.

We don't need to specify the supplySensitivity and groundSensitivity for digital pins since all digital pins will remain digital domain everywhere in the testbench structure.

```
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;
```

Figure 12. PLL STIM_HW with digital drivers (RNM model simulation)

### *G. STIM_SW (software)*

STIM_SW is used to define the test sequence by controlling the hardware components defined in STIM_HW. We can either set the voltage/current and ramp time of voltage/current source or set the logic inputs to be 0 or 1. In addition, we feed a clock to the REF pin to provide the clock reference to PLL.

To enable we can use the common test sequence code in STIM_SW for both AMS drivers and digital drivers, we are using the same task name and arguments, such as "setv" for both voltage_driver (AMS driver module) and wreal_driver (digital driver module).

```
 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
```

Figure 13. PLL STIM_SW

In STIM_SW, besides driving the inputs of DUT in the test sequence, we also need to measure the DUT output and compare with expected value. The monitor modules we have developed are written with SystemVerilog code

and take the input with only digital types (bit, logic or real). Since AMS model has some electrical type outputs, we need to convert them to real type and send them to monitor modules which can monitor real type.

The A2D conversion method we are using is to sample electrical signals with "clk" created within analog_clock module (Figure 15). This clock can align with analog steps created by simulator, so the sampled signal doesn't lose the precision after A2D conversion. Figure 14 provides a good example of how analog_clock is created [1]. In the waveform shown in Figure 16, Vctrl from VCO in PLL is a fast-changing signal and the major simulation speed limiting factor in the AMS model simulation. In the beginning, Vctrl updates comparatively infrequently and then changes more often when close to or after PLL settling. Similarly, "clk" from analog_clock module toggles slower in the beginning and faster afterwards, which fully align Vctrl toggling response.

The real type monitors consist of a group of self-developed IP modules to monitor real type signal, min/max/average/peak to peak value, or digital type signal like frequency measurement. Those monitors are connected to corresponding signals based on the user case.

```
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
```

Figure 14. analog_clock module

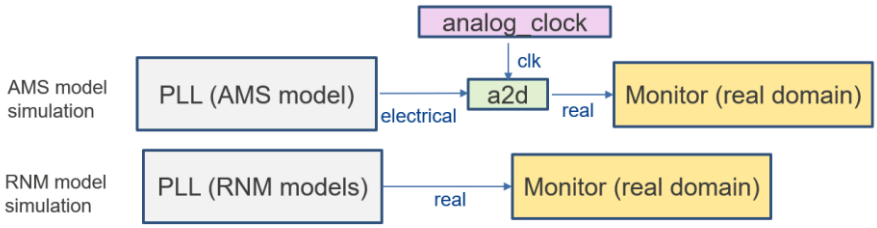



Figure 15. auto-checker mechanism

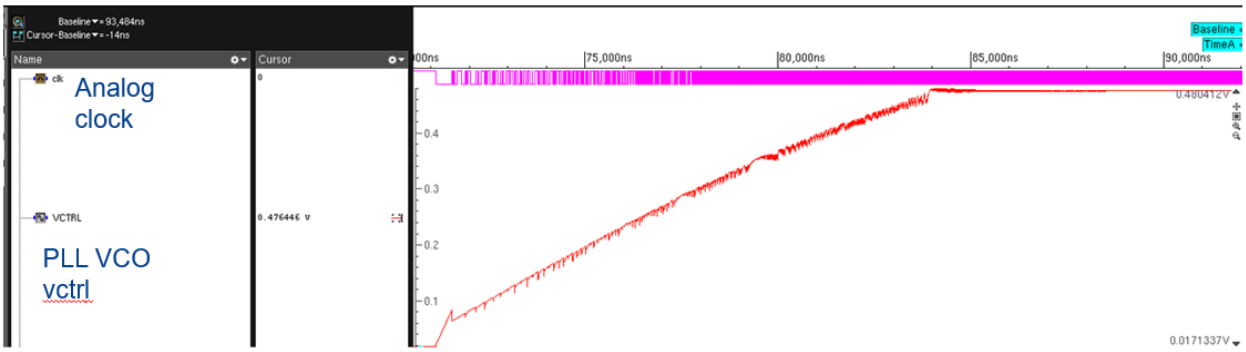



Figure 16. Analog clock and VCO Vctrl in AMS model simulation

### *H. How to run simulation*

The most straightforward way of running modeling verification is to create Maestro views in Cadence Virtuoso and directly run simulation there. To manage the simulation result directories in a more organized approach and ease

with post-processing, we use Makefile to netlist and run simulation. Figure 17 shows the modeling verification simulation directory structure we used to run PLL simulation.

```
Modeling_verification
 - PLL
  - netlist
   - config_ams_model
    - netlist
   - config_rnm_closed_loop_model
    - netlist
  - sim
   - config_ams_model
    - Job_1.0
     - psf
   - config_rnm_closed_loop_model
    - Job_1.0
     - psf
```

Figure 17. Modeling verification simulation directory structure

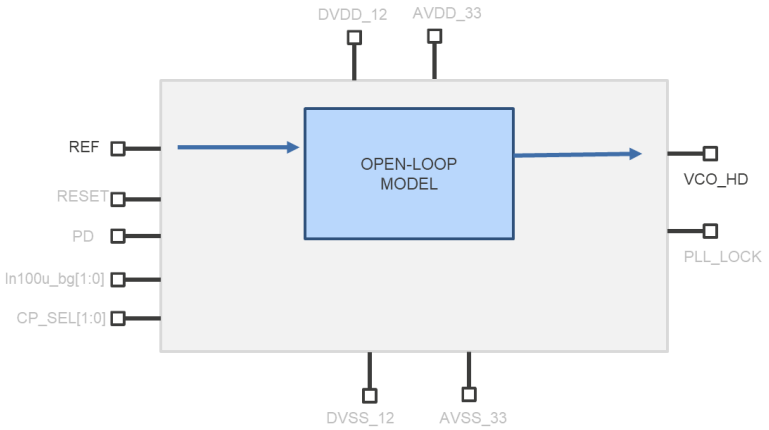
## III. MODELING IMPLEMENTATION

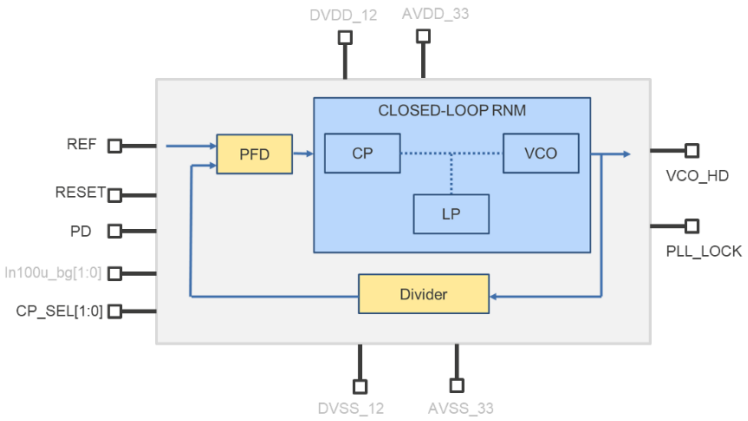
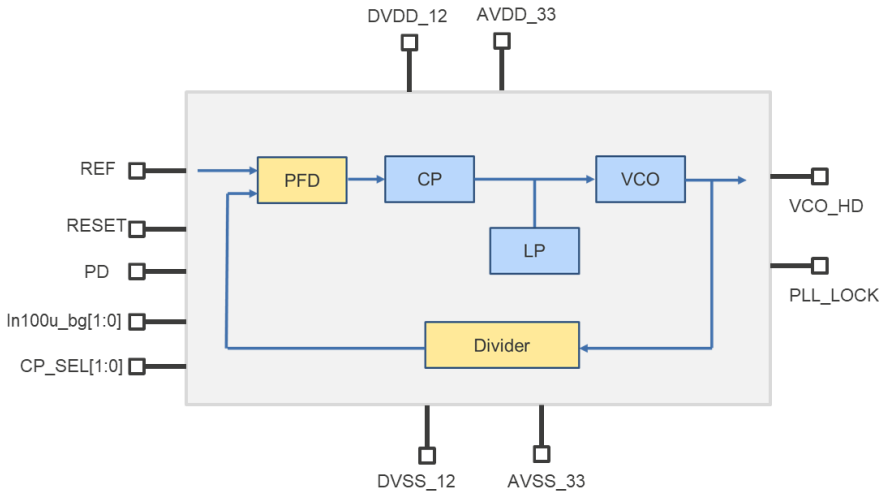
In this section, we are going to talk about the implementation of all four models for PLL. Since the charge pump and loop filter are the keys and most challenging part in the modeling, we will show example code and brief explanations below. We also discuss hierarchy conformance for modeling.

### *A. Hierarchy conformance*

We use models in multiple design phases: prototyping, design optimization, and verification. Interconnect verification is often left out in the early design phases when the design implementation is not finalized. During the design phase, prototype and design optimization models help with system architecture decisions but may not be suited for detailed verification. The design optimization model helps to check that the PLL loop parameters are appropriate for the system. In the verification phase, a hierarchically conforming model is critical for checking signal and port connections for proper functionality. For hierarchy conformance, we rate models as low, medium, or high in the table below.

TABLE II
PLL MODEL RESULTS COMPARISON

| Hierarchy Conformance | Use | |
|---|---|---|
| Low | Prototyping |  |
| | | Not all the top port functionality is modeled. For example, instead of using an input port's current or voltage a model parameter controls a function directly. PLL implementation details may be unknown at this level. |

| | | |
|---|---|---|
| Medium | Design Optimization |  <br>Top port control functionality is modeled but sub-block implementation uses a simplified hierarchy. This is true for the closed-loop RNM. The closed-loop RNM combines the charge-pump, loop filter, and VCO into a single block that captures the dynamics of interest for type-II PLLs but does not scale with bias current. |
| High | Verification |  <br>Detailed port functionality and interconnect are modeled from top-level to the lowest level blocks of interest. Here, EEnet models outshine wreal and AMS/electrical types because the EEnet discipline is simple to implement and fast to simulate for interconnect checks. |

*B. Open loop RNM model*

The open loop RNM model will use single SystemVerilog model code to emulate the entire PLL behavior without knowing the design implementation of the PLL block. Figure 18 shows an example of open loop model code. Similarly, we monitor whether the supplies are within the operating range or not. Then we use mathematical equation to measure the input REF clock frequency and calculate the PLL output frequency which is 7 times of the REF clock. The code implementation of the calculation is skipped in the paper.

```
import cds_rnm_pkg::*;
module PLL ( PLL_LOCK, VCO_HD, AVDD_33, AVSS_33, CP_SEL, DVDD_12, DVSS_12,
PD, REF, RESET, in100u_bg );

  input wreal4state DVDD_12;
  input wreal4state DVSS_12;
  input wreal4state AVDD_33;
  input wreal4state AVSS_33;
  input wreal4state in100u_bg[1:0];

  input REF;
  input [1:0] CP_SEL;
  input RESET;
  input PD;

  output VCO_HD;
  output PLL_LOCK;

  assign  AVDD_33_ok = ((AVDD_33 < 3.6) && (AVDD_33 > 3.0))? 1:0; // 3.3V
  assign  AVSS_33_ok = ((AVSS_33 < 0.1) && (AVSS_33 > -0.1))? 1:0; // 0V

  assign  DVDD_12_ok = ((DVDD_12 < 1.6) && (DVDD_12 > 0.8))? 1:0; // 1.2V
  assign  DVSS_12_ok = ((DVSS_12 < 0.1) && (DVSS_12 > -0.1))? 1:0; // 0V

  assign pwr_ok = AVDD_33_ok & AVSS_33_ok & DVDD_12_ok & DVSS_12_ok;
  assign int_en = pwr_ok & (~PD) & (~RESET);

  // Measure REF clock frequency
  // When int_en is 1, create VCO_HD with 7 times REF clock
```

Figure 18. PLL open loop model code

## C. *Closed loop RNM*

There are many ways to model a PLL loop filter. The example below uses real variables C1_v and C2_v to store the charge values on the two capacitors. It's important to note that the 'up' and 'dn' signals from the PFD are needed in the sensitivity list for this model to work.

```
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
```

Figure 19. RNM Filter Code

To make the 'up' and 'dn' signals visible to the filter, the closed-loop RNM combines the charge-pump, filter, and VCO into a single block. This is good for a general-purpose PLL model but prevents this model from fully verifying signals to each sub block. The wreal net type cannot simultaneously relay the charge-pump current with 'up' 'dn' sensitivity and the control voltage. The EEnet net type can transport simultaneous voltage and current information and is better suited for maintaining the hierarchy and interconnect of the charge-pump, filter, and VCO.

### *D. Verilog-AMS models*

For AMS models, we still want to keep the logic gates in sub-blocks such as PFD (phase and frequency detector) and frequency divider as it is but use logic gates in Verilog or Verilog-AMS view. Then the true design implementation of those blocks can be achieved in the model without slowing down the simulation. To emulate device analog behavior, there are two approaches to create Verilog-AMS models in general. One approach is to use voltage, current equations in analog procedure blocks to model voltage source, current source, capacitor, or resistor behavior. But this approach is not straightforward for users especially when we need to model complex network of components. The other approach is to leverage pre-created reuse-ip modules for components such as voltage source, current source, capacitor, and resistor, and use them as building blocks to put them together. There are some open source AMS models online [2]. Then we can use the pre-defined tasks within those reuse-ip modules to control them.

Figure 20 shows the diagram for the charge-pump within the PLL block. The modeling of the charge-pump is basically to connect two current sources, and to use input logic control “up” and “dn” to control current source value.

Similarly, as the AMS driver part mentioned in the last session, we need to specify the supplySensitivity and groundSensitivity for all digital pins.

Another critical feature for the model is that we need to detect when the supplies of the block go out of range. For example, the operation range for “avdd” pin is between 0.8V and 1.6V, so when “avdd” pin goes out of the range, we need to shut off the charge-pump. Here we used the same “clk” signal from analog_clock module to sample “avdd” voltage, because it is a dynamic clock to track analog time setups without introducing the penalty of simulation slowness. The alternative solution is to use “cross” or “above” function, but it will introduce additional time steps when tested signal is getting close to the threshold value, and it can slow down the simulation.

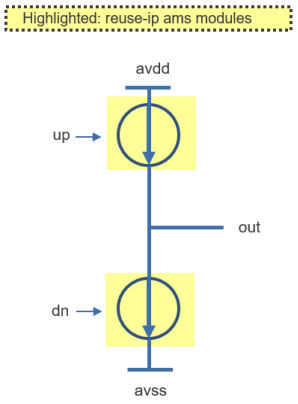



Figure 20. Charge pump modeling diagram

```
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; // 1.2V
25    avss_ok = ((V(avss) < 0.1) && (V(avss) > -0.1))? 1:0; // 0V
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
```

Figure 21. Charge pump model code

Figure 22 and 23 show the block diagram and model code for the loop filter. Similarly, we are building blocks of switch, capacitor and resistor, and the model code is not hard to create and easy to read.

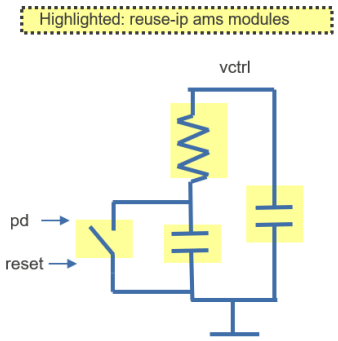



Figure 22. Loop filter diagram

```
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
```

Figure 23. Loop filter model code

### *E. EEnet model*

EEnet modeling technique is a mix of VerilogAMS modeling and regular RNM modeling. EEnet is a new signal type in RNM modeling, and EEnet nodes have three properties: voltage, current, resistance. It can be used to model impedance effects. In our PLL example, only EEnet is used to model VCTRL node which connects between charge pump, loop filter and VCO. More EEnet nodes can potentially slow down the simulation, so we use EEnet nodes if only necessary to model the impedance effect. All the other nodes without EEnet are used in the same way as in other RNM models. Figure 24 shows the example code of charge pump models in EEnet, and you can see only “out” port is EEnet which is connected to loop filter.

Since EEnet capacitors, resistors, switches are provided by vendor Cadence, so we can directly instantiate those components in the similar way s VerilogAMS model. This is a big advantage over closed-loop RNM model, and you can see in loop filter model (Figure. 25), CapGeq is used to build the loop filter network.

```
module chargepump ( up, dn, avdd, avss, out );

  input wreal4state avdd;
  input wreal4state avss;
  input up;
  input dn;
  input pdb;
  output EEnet out;

  reg timeout=0; always #(5ns) timeout = ~timeout;

  assign  avdd_ok = ((avdd < 1.6) && (avdd > 0.8))? 1:0;
  assign  avss_ok = ((avss < 0.1) && (avss > -0.1))? 1:0;
  assign  i10u_ok = ((i10u < 12e-6) && (i10u > 8e-6))? 1:0;
  assign pwr_ok = avdd_ok & avss_ok;
  assign int_en = pwr_ok & pdb;

  real vmid, rout, iout;

  initial begin
    iout = 0;
    vmid = `wrealZState;
    rout = `wrealZState;
  end

  always @(up,dn, int_en, timeout) begin
    if(int_en) begin
       if ((up<dn) && (out.V<=0.0)) iout = 0.0;
       else iout = up*10e-6 - dn*10e-6;
    end
    else iout = 0.0;
  end

  assign out = '{vmid, iout, rout};
endmodule
```

Figure 24. Charge pump model code

```
import cds_rnm_pkg::*;
import EE_pkg::*;

module loop_filter ( avss, dvdd, dvss, pd, reset, vctrl );

  input EEnet vctrl;
  input wreal4state dvdd;
  input wreal4state dvss;
  input pd;
  input reset;

  assign dvdd_ok = ((dvdd < 1.6) && (dvdd > 0.8))? 1:0; // 1.2V
  assign dvss_ok = ((dvss < 0.1) && (dvss > -0.1))? 1:0; // 0V
  assign pwr_ok = dvdd_ok & dvss_ok;
  assign int_en = pwr_ok & (~pd) & (~reset);

  CapGeq #(.c(476e-12), .tinc(5e-9), .rs(1.526e+3)) filtcap1(vctrl);
  CapGeq #(.c(3.52e-12), .tinc(5e-9)) filtcap2(vctrl);
endmodule
```

Figure 25. Loop filter model code

## IV. SIMULATION RESULTS COMPARISON

The four PLL models are developed and tested in the automated modeling testbench with the 1 DUT setup and the comparison of the four models is shown in table II.

### *A. Comparison analysis*

As for the development difficulty, AMS model is the most difficult one, especially when we need to handle the connectivity for all control, supply voltage and bias current. For example, if we need to sink or source bias current into the DUT, and the load of the bias current can be either a valid load in the model, which can be modeled with resistor or a voltage source, but also the load can be open circuit. If a bias current goes to the open circuit, the voltage keeps increasing and goes to infinity. Then we need to add clamp to the current source, but it can slow down the simulation. Also dealing with convergence issue is a challenge, especially when we need lots of AMS modeling in the DUT. Open-loop RNM model is a behavior model without considering the true implementation of the design and it is easy to create. Closed-loop RNM without EENet is difficult to create as we need to use either Laplace function or Bilinear transform, KCL (Kirchhoff's current law) and KVL (Kirchhoff's voltage law) to express the

circuit with mathematical equation in discrete time domain. Also, the equation can be very complicated if the complexity of circuit network increases. Building closed-loop RNM with EENet is to plugin EENet components such as capacitor, resistor, voltage source, switch models provided by vendor. It is very convenient for users though it is the similar equations and theories behind the scenes.

The second factor compare is that if we need to modify the PLL schematic for modeling. When we use PLL schematic without change, we can keep the connectivity between sub-modules (such as PFD, CP, VCO) for PLL as it is. The benefits are: 1. We can save the effort of updating the model schematic if there is any change to the design schematic. 2. We can verify the connectivity of PLL schematic to catch potential design bugs. AMS model and closed-loop RNM with EENet can allow us to use the original PLL schematic. For closed-loop RNM without EENet, since we are using the equation to emulate the behavior of charge pump sinking/sourcing current into the loop filter, then we need to combine the two blocks when doing modeling. The open-loop RNM model itself is one single model code file.

The third factor is the accuracy of the model. Since the AMS model is simulated in the analog domain, it can have very good accuracy, and time step profile tracks the switching behavior of blocks. When the PLL is toggling fast, more time steps are created, and simulation is slowing down. When PLL is not toggling, fewer time steps are created, and simulation is running fast. So, AMS model accuracy is highest. Both closed-loop RNM without and with EENet can still model the PLL settling process with pretty good accuracy by using a fixed sampling clock to digitalize the circuit behavior. The accuracy of the two closed-loop models is not as high as that of the AMS model, but they are still decently good. The open-loop RNM doesn't model PLL settling response so the accuracy is low.

TABLE III
PLL MODEL RESULTS COMPARISON

| | **AMS model** | **Open-loop RNM** | **Closed-loop RNM without EENet** | **Closed-loop RNM with EENet** |
|---|---|---|---|---|
| What feature to be modeled? | Accurate frequency divider ratio, settling response, charge pump current selection, DFT | Accurate frequency divider ratio | Accurate frequency divider ratio, settling response | Accurate frequency divider ratio, settling response |
| 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 and VCO (optional) | No change |
| Driver | AMS driver | Digital driver | Digital driver | Digital driver |
| 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: data path | Not tested | 16 min 10 s | 22 min 18s | 21 min 35s |

When simulating those models in automated verification testbench with 1 DUT option, we can compare VCTRL which is the control voltage for VCO to check the accuracy of those models. Figure 26 below shows the comparison of closed-loop RNM with and without EENet, and AMS model. From ramp time, settling voltage and jitter size, the three models correlate very well with each other. Since open-loop RNM does not include VCO modeling so it is not compared here. Simulation time for open-loop RNM, closed-loop RNM with and without EENet is all very fast, and can finish around 1 second. AMS model simulation takes 14 minutes to finish. Though some speed optimization methods can be used to further improve the speed, simulation time is still far slower than other RNM models.

In addition, we put those PLL models into a top-level verification test with heavy data transaction and compare the simulation speed. Since the AMS model takes long to run even by itself, so we don't put it in the experiment. The simulation with open-loop RNM takes 16 min 10s to finish, while closed-loop RNM with and without EENet are performing in a similar speed, around 22 min, which is a little bit slower than open-loop RNM model.

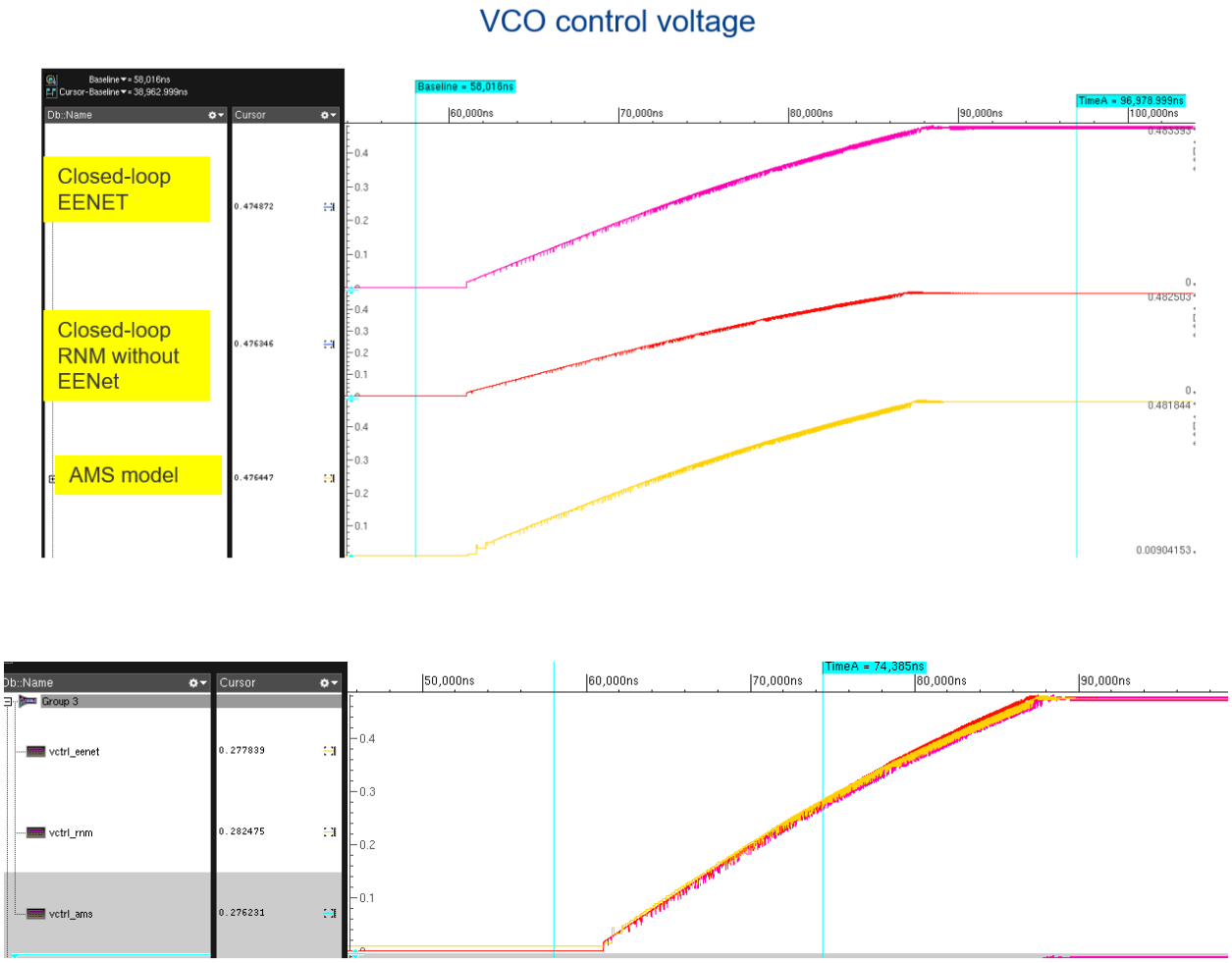



Figure 26. PLL modeling waveform

### *B. Conclusion*

Before the experiment, we were thinking the open-loop model should be used in heavy digital simulation due to its simplicity. However, closed-loop RNM models with and without EENet both can be simulated with speed comparable to open-loop model and has a big benefit of high accuracy. Between those two RNM models, we would pick the one with EENet since it is easier to develop without the need to change PLL schematic.

So, the conclusion is that for a new project, we would use PLL EENet models in all digital simulation, and still use AMS model for a few analog related simulations from top level verification.

## I. REFERENCES

[1] JAROSLAV HRADIL "Modern verification methods of mixed-signal integrated circuits", Master Thesis, 2014
[2] Designer's Guide Community :: Verilog-AMS (designers-guide.org)