SYNOPSYS®

Beyond Boolean: Smart Abstraction Choices in Mixed-Signal Verification

Finding the Perfect Balance Between Effort, Accuracy and Speed

Evgeny Vlasov, Sr Staff PE, AMS/DMS Verification

May 2025

Legal Disclosure

CONFIDENTIAL INFORMATION

The information contained in this presentation is the confidential and proprietary information of Synopsys. You are not permitted to disseminate or use any of the information provided to you in this presentation outside of Synopsys without prior written authorization.

IMPORTANT NOTICE

This presentation may include information related to Synopsys' future product or business plans. Such plans are as of the date of this presentation and subject to change. Synopsys is not obligated to update this presentation or develop the products with the features and/or functionality discussed in this presentation. Additionally, Synopsys' products and services may only be offered and purchased pursuant to an authorized quote and purchase order or a mutually agreed upon written contract.

FORWARD LOOKING STATEMENTS

This presentation may include certain statements including, but not limited to, Synopsys' financial targets, expectations and objectives; business and market outlook, business opportunities, strategies and technological trends; and more. These statements are made only as of the date hereof and subject to change. Actual results or events could differ materially from those anticipated in such statements due to a number of factors. Synopsys undertakes no duty to, and does not intend to, update any statement in this presentation, whether as a result of new information, future events or otherwise, unless required by law.

Learning plan

1. Structural electrical modeling with VIR_net
2. Technical overview
3. Boost converter and APLL case studies
4. Where is the speed-up coming from?
5. Future-proof

Real life implications of bad AMS designs

- Recent personal example:
  - a screen I was using in the office has started to produce a very loud high-pitch sound through loudspeakers during a Zoom call. I've muted my loudspeakers in Windows – no effect. I've disconnected USB-C cable – no effect. I've pulled the power cord – it helped.
- If a system interact with user – it is using analog signals. Failure to properly validate AMS systems leads to problems that can affect users physically.

Here's a picture of a robot went 'full terminator' mode and caused workers to dodge

PC users: due to DDR re-training process with XMP profiles my PC takes 30+ seconds to load. It never happened with DDR4!

Car users: my car brakes due to phantom obstacles!

Homeowners: my garage door never opens from the first button press!

SYNOPSYS®

VIR_net

Next step in structural SystemVerilog RNM methodology

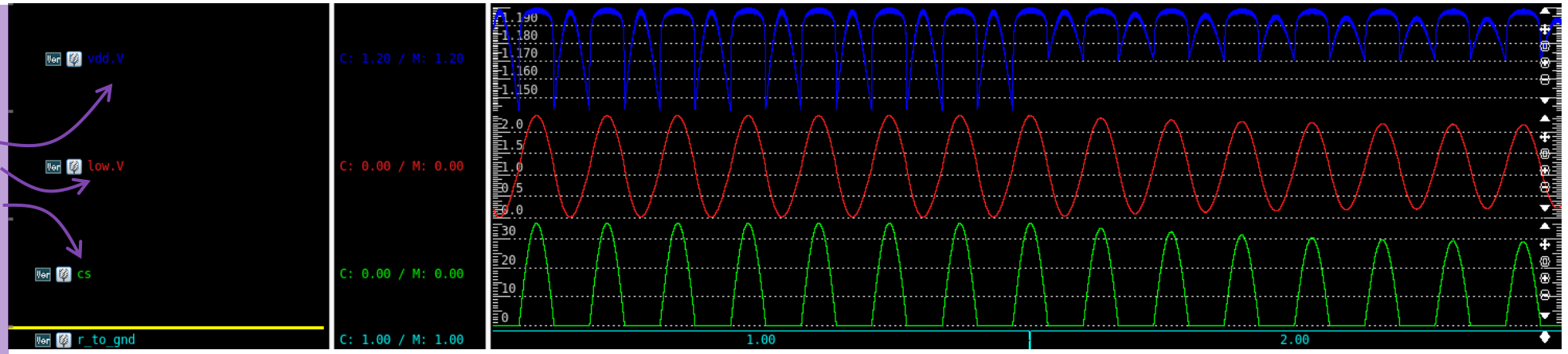
Structural electrical modeling with SystemVerilog

- High accuracy
  - Native modeling of voltages, currents, impedances
  - Support for wide range of analog primitives: transmission gates, capacitors, inductors, diodes, voltage sources, current sources
- Faster verification
  - 1000x test vectors with DMS can help identify which vectors needs to be simulated in AMS
  - Models are reusable and PDK agnostic
- Full reusability of digital regression environment
  - UVM
  - Coverage
  - SVA
- Powerful debugging and profiling in Verdi
  - Debug of convergence cycles with Verdi ‘Event sequence’ or ‘Log-to-wave’
  - Performance profiling with VCS profilers

LC VCO (current-shaping)

Equivalent scheme (single LC tank)

VDD
VOUT
Req (vcr)



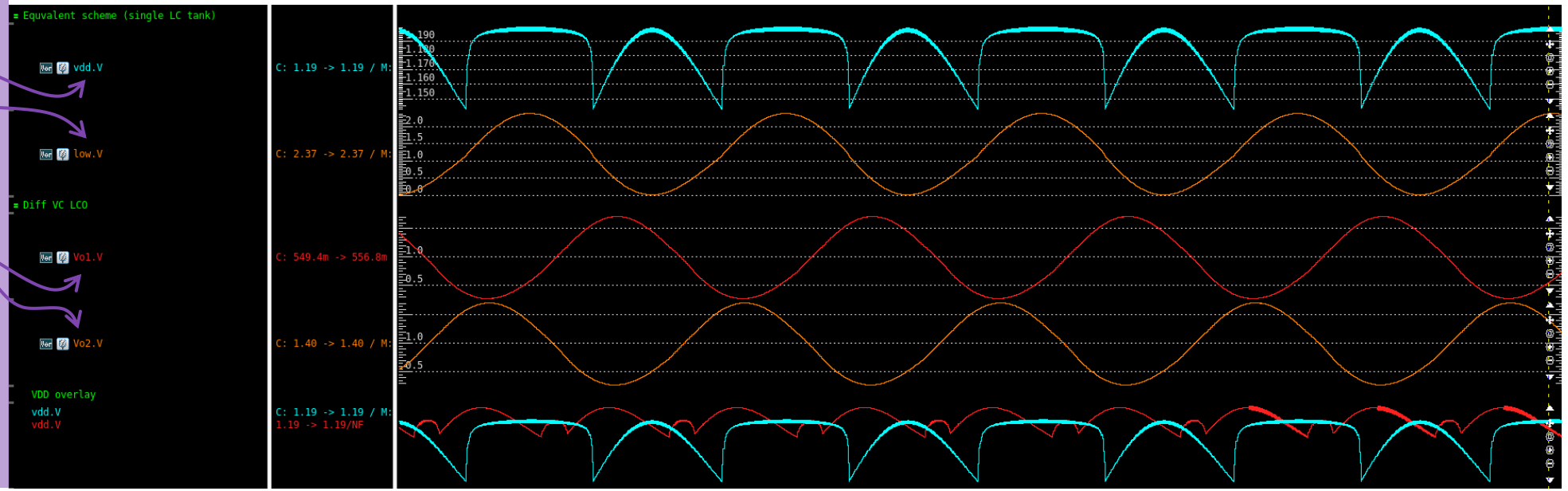
Equivalent scheme

VDD
VOUT

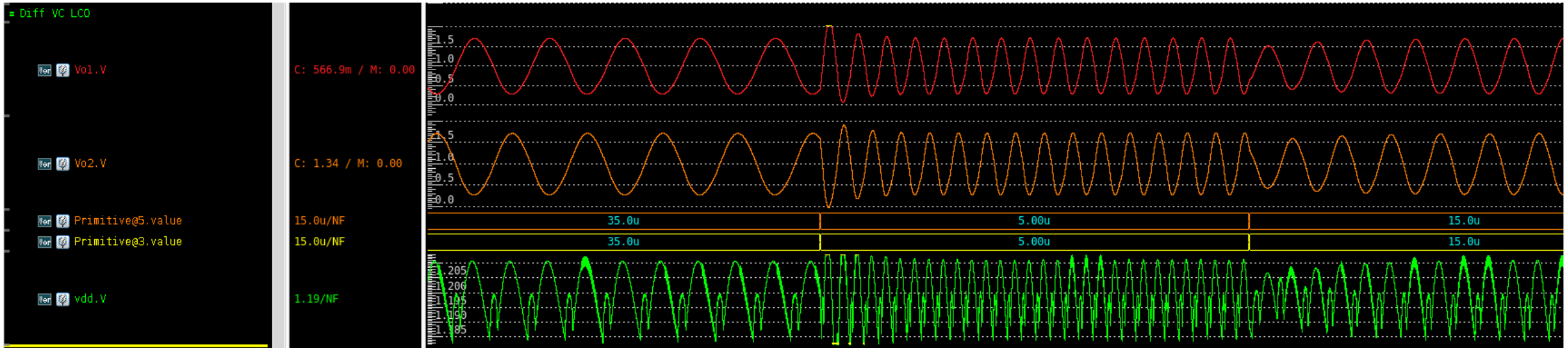
Cross-coupled LC VCO

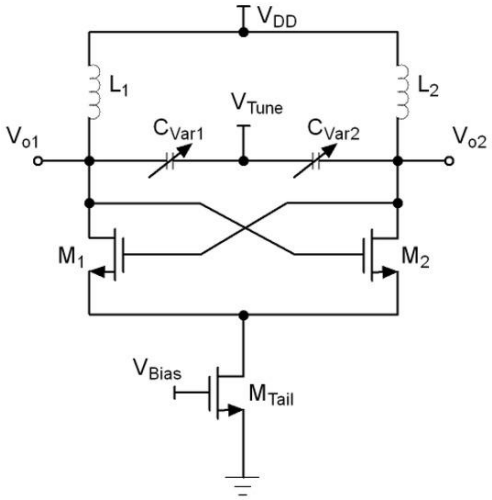
VOUT1
VOUT2

VDD Overlay

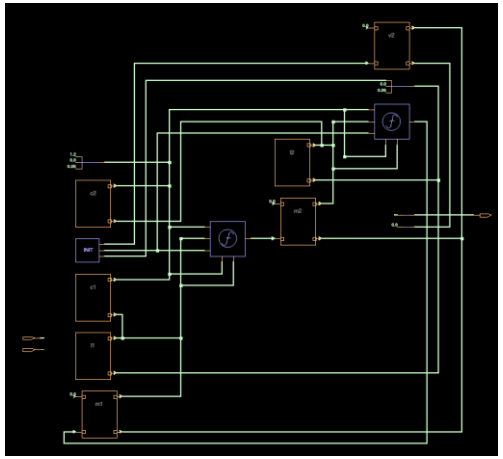


Cross-coupled LC VCO – variable capacitance



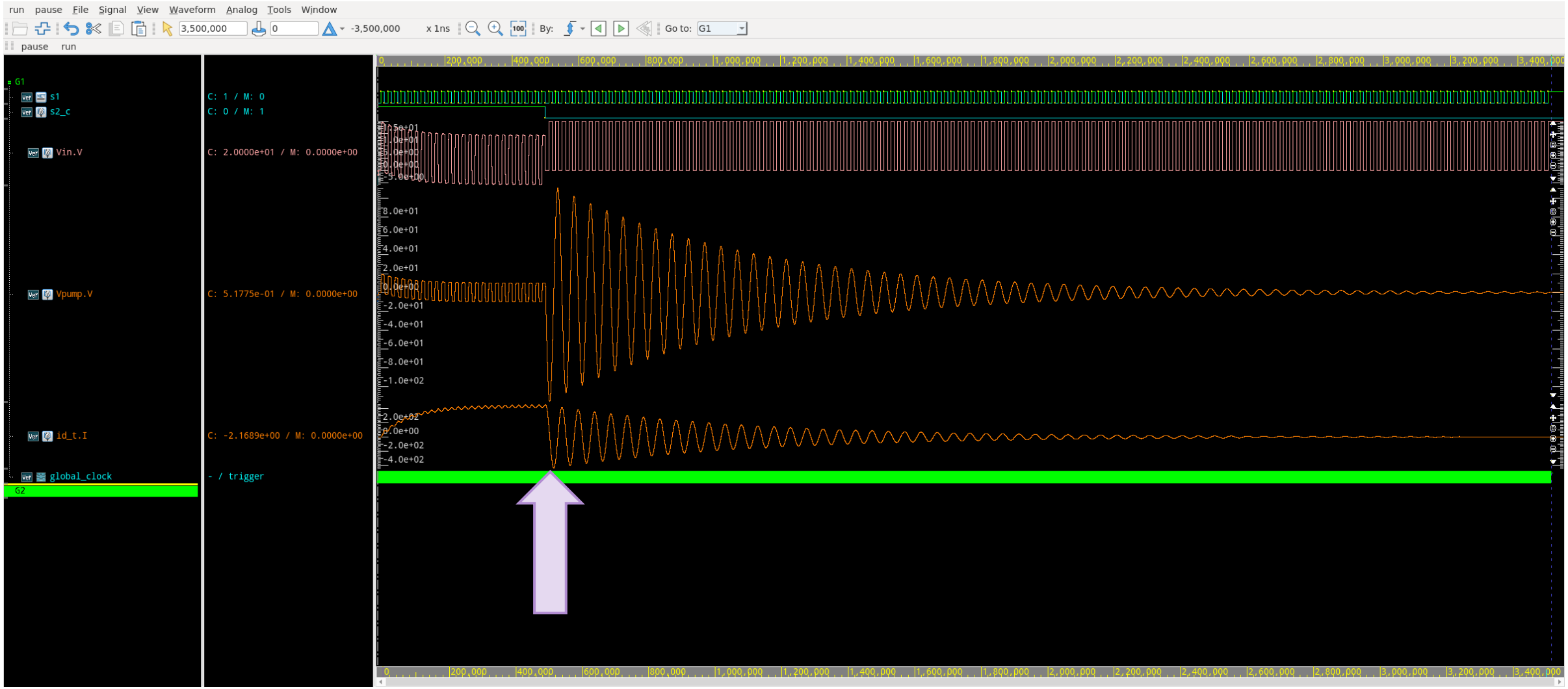


i2	lc_vco
c1	vir_capacitor
c2	vir_capacitor
l1	vir_inductor
l2	vir_inductor
m1	vcvsD
m2	vcvsD
v2	vcvsD

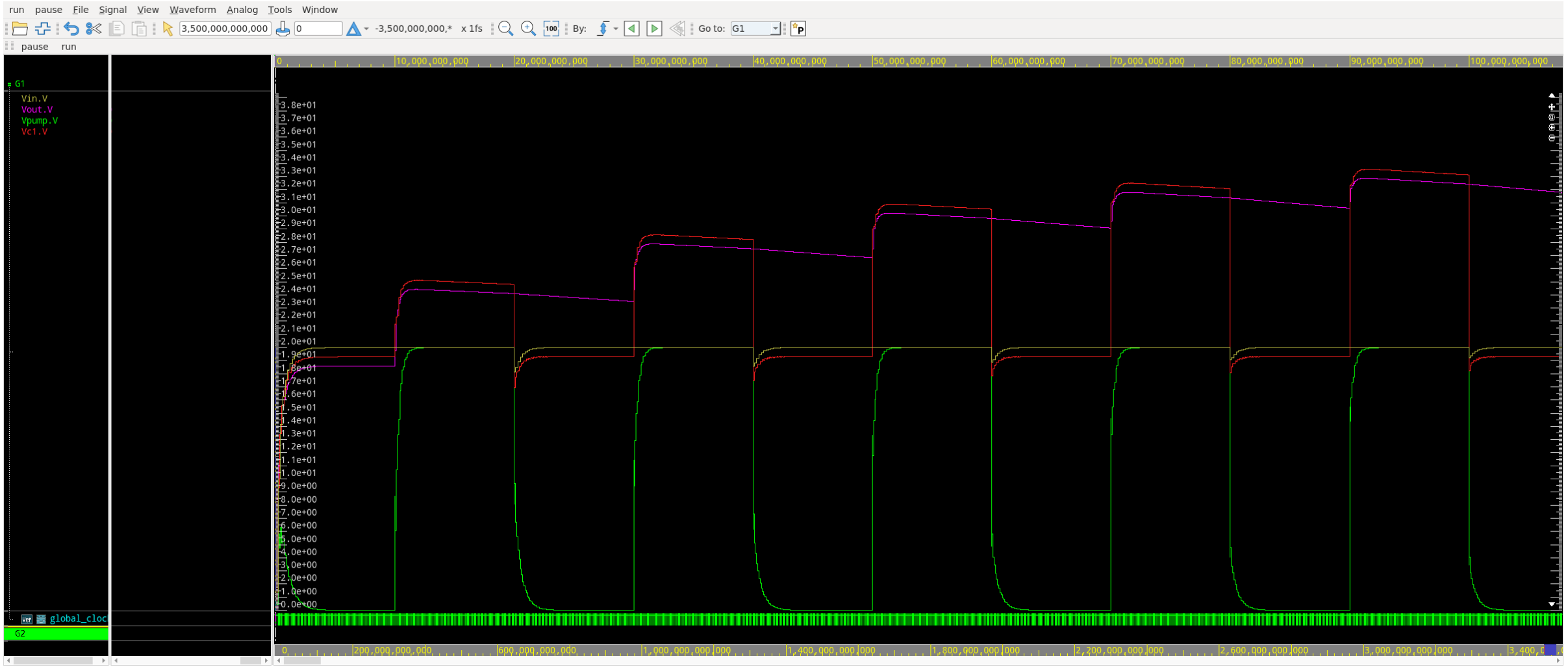


https://www.mdpi.com/sensors/sensors-21-06583/article_deploy/html/images/sensors-21-06583-g001-550.jpg

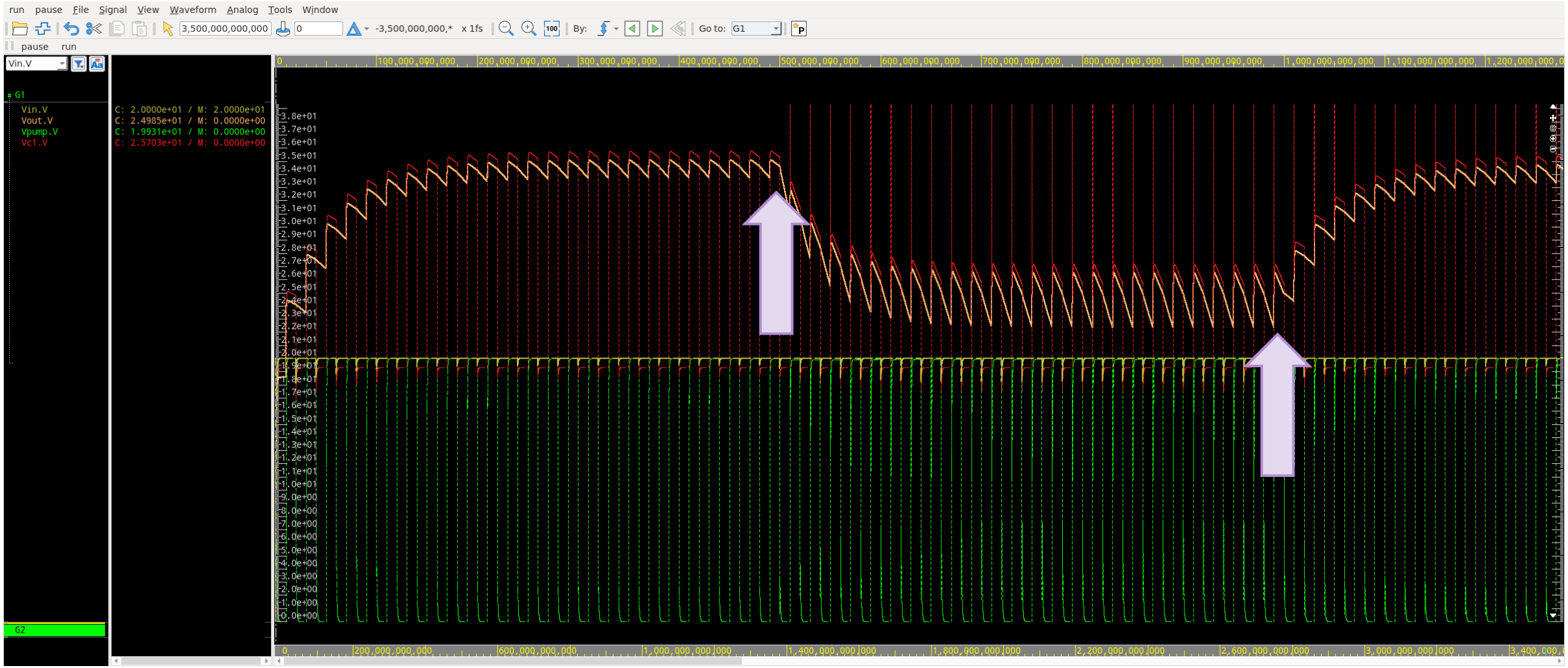
Parallel LC tank – forced charge injection, then free oscillation



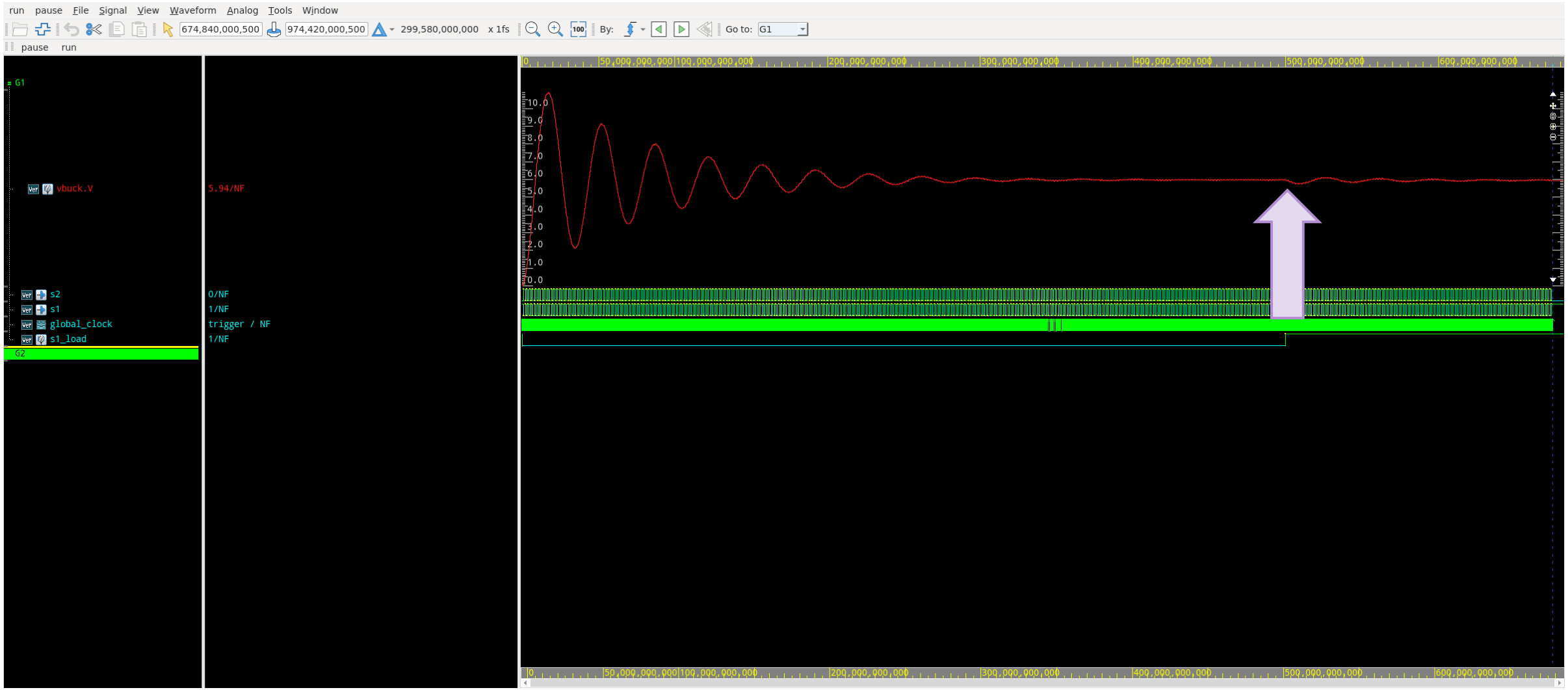
Charge pump - startup



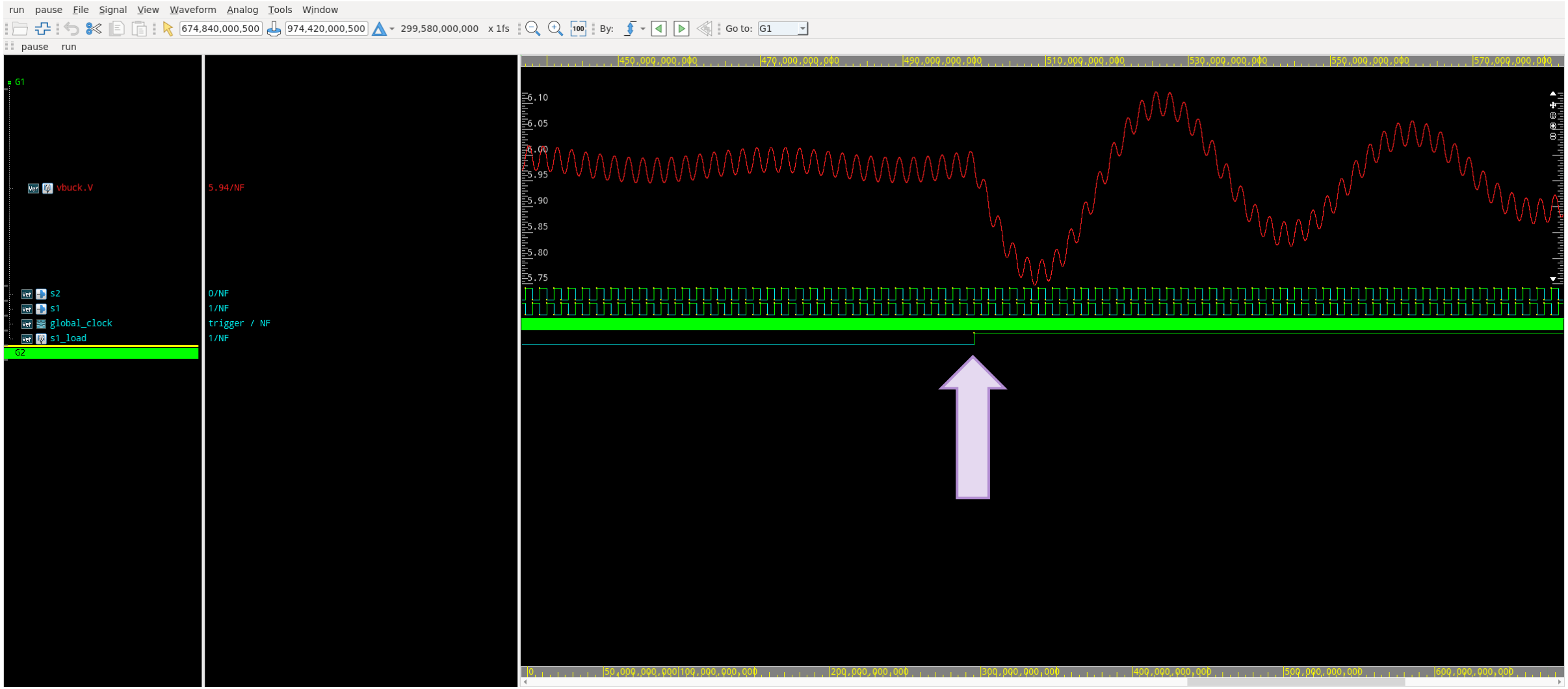
Charge pump – step change of a load



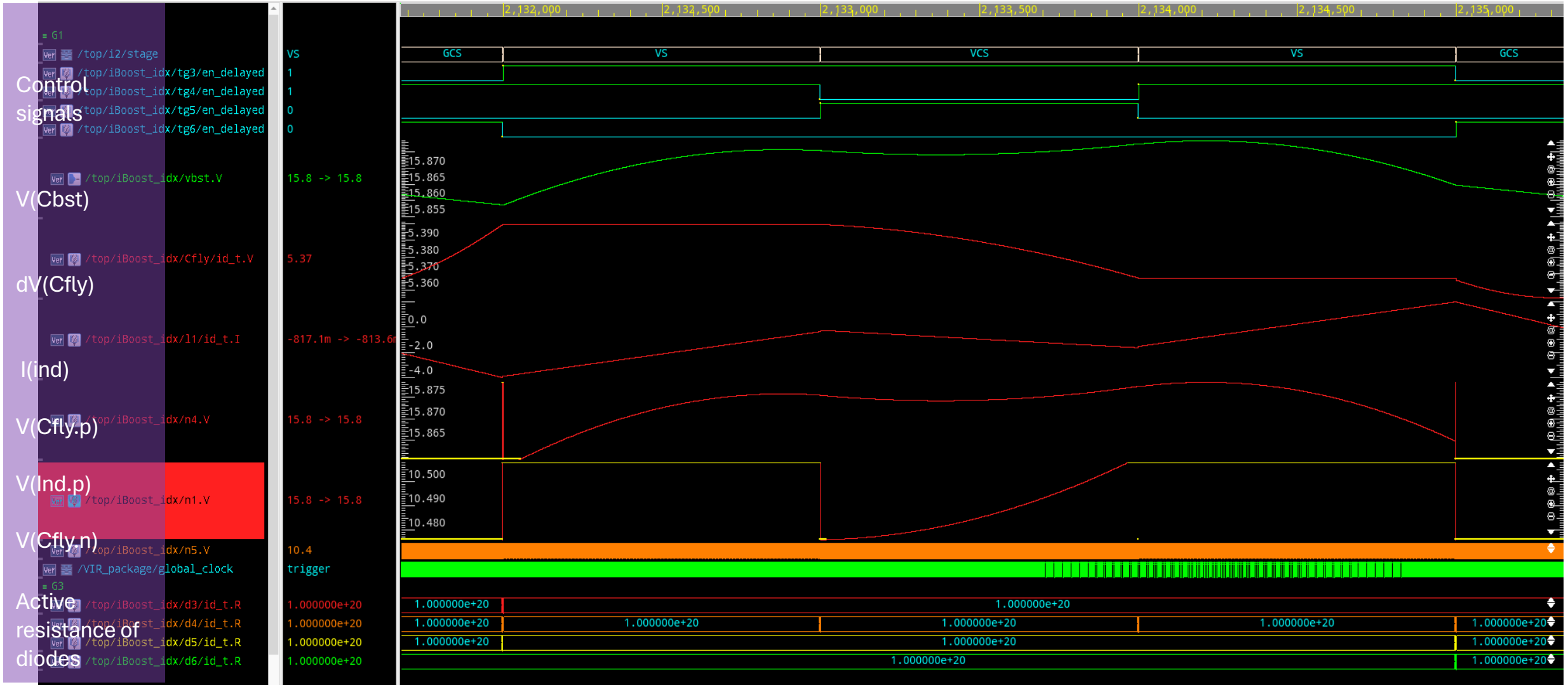
Buck converter - startup



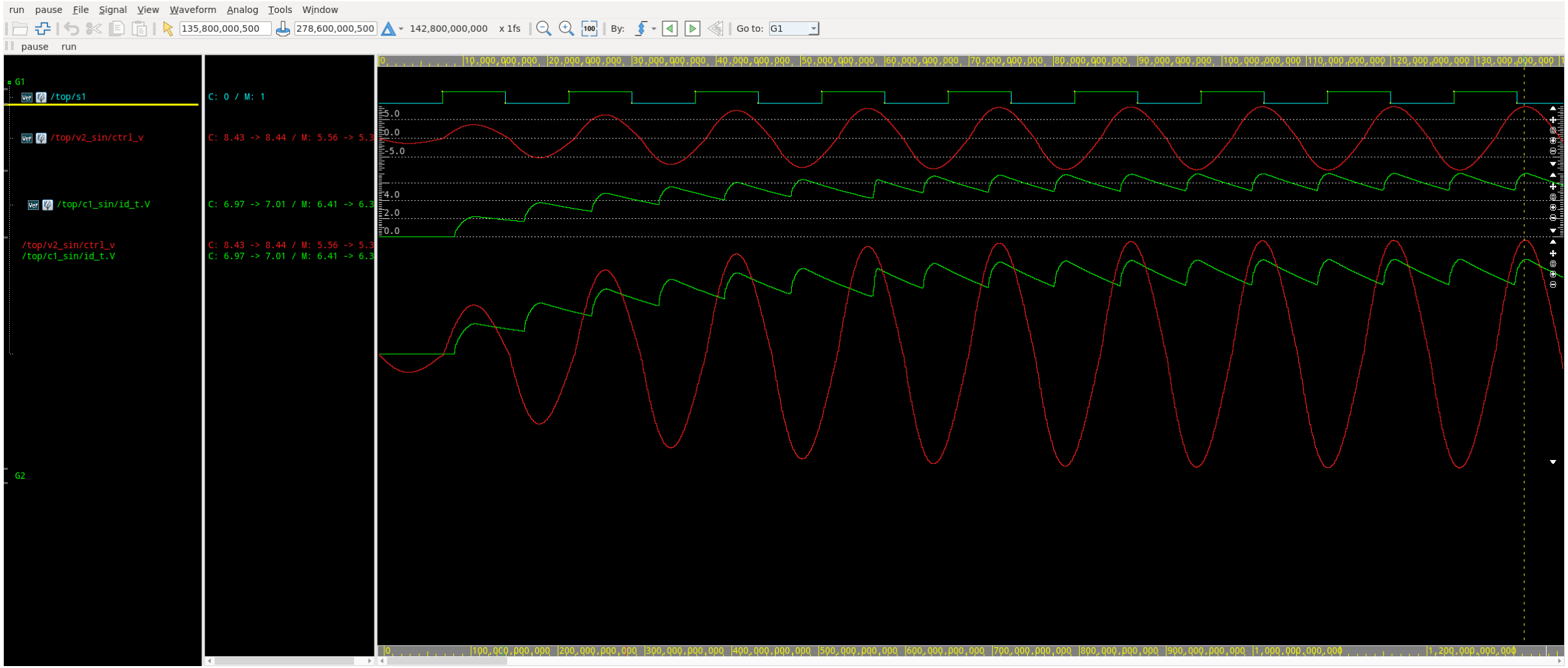
Buck converter – step change of a load



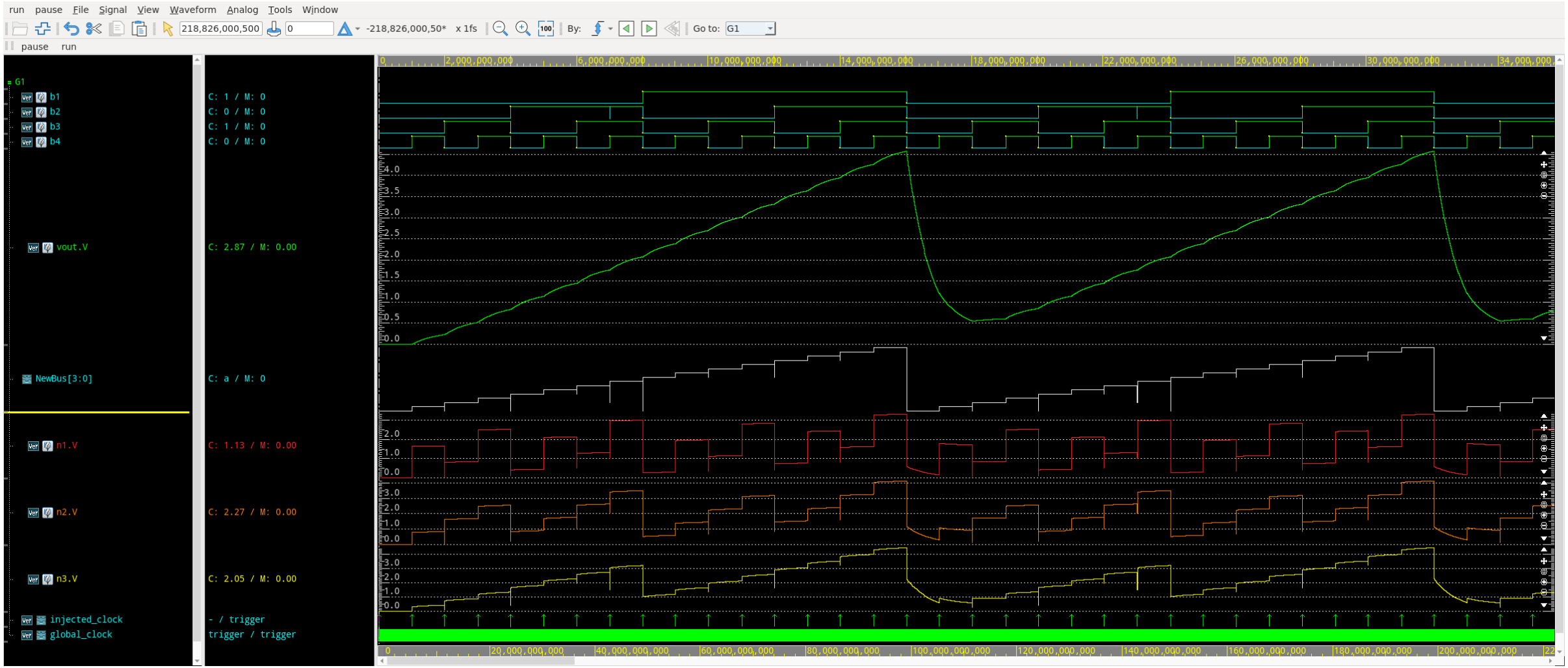
Boost converter with Cfly – one conversion cycle



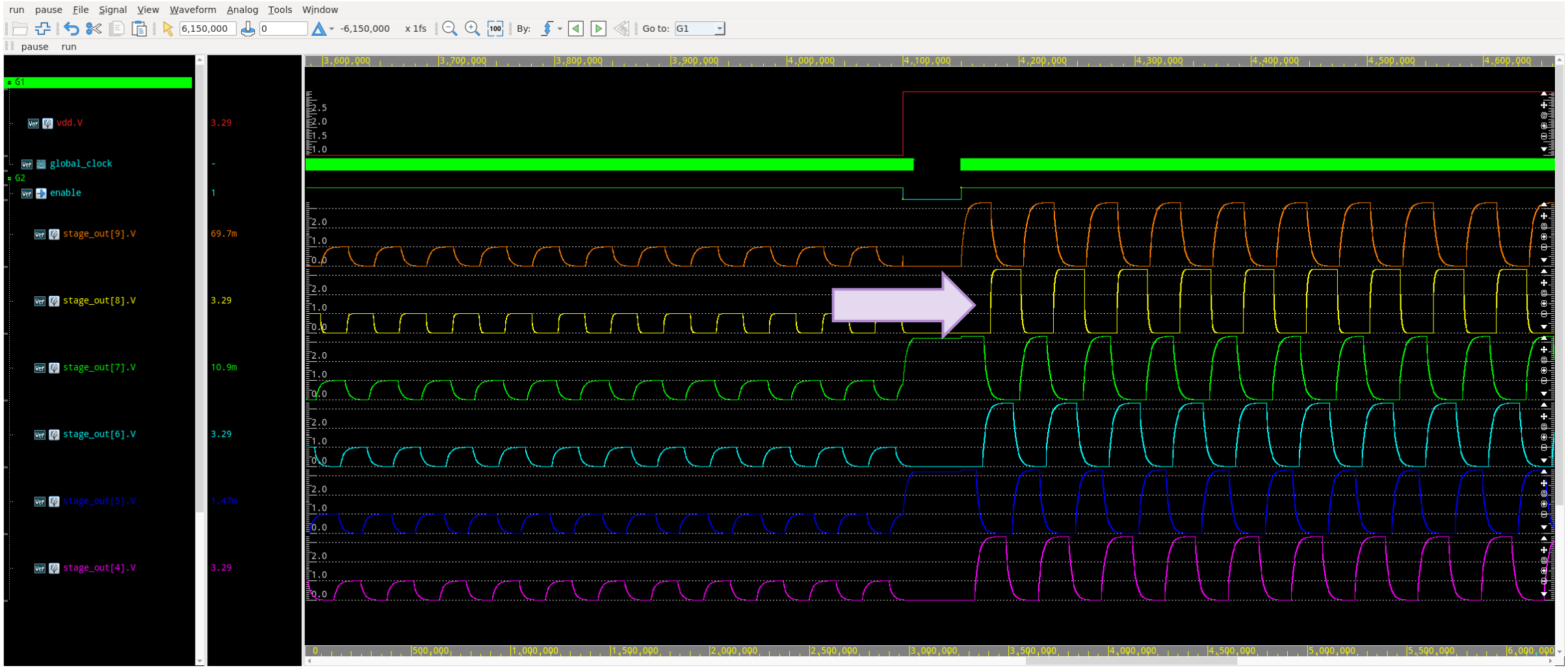
Full wave rectifier



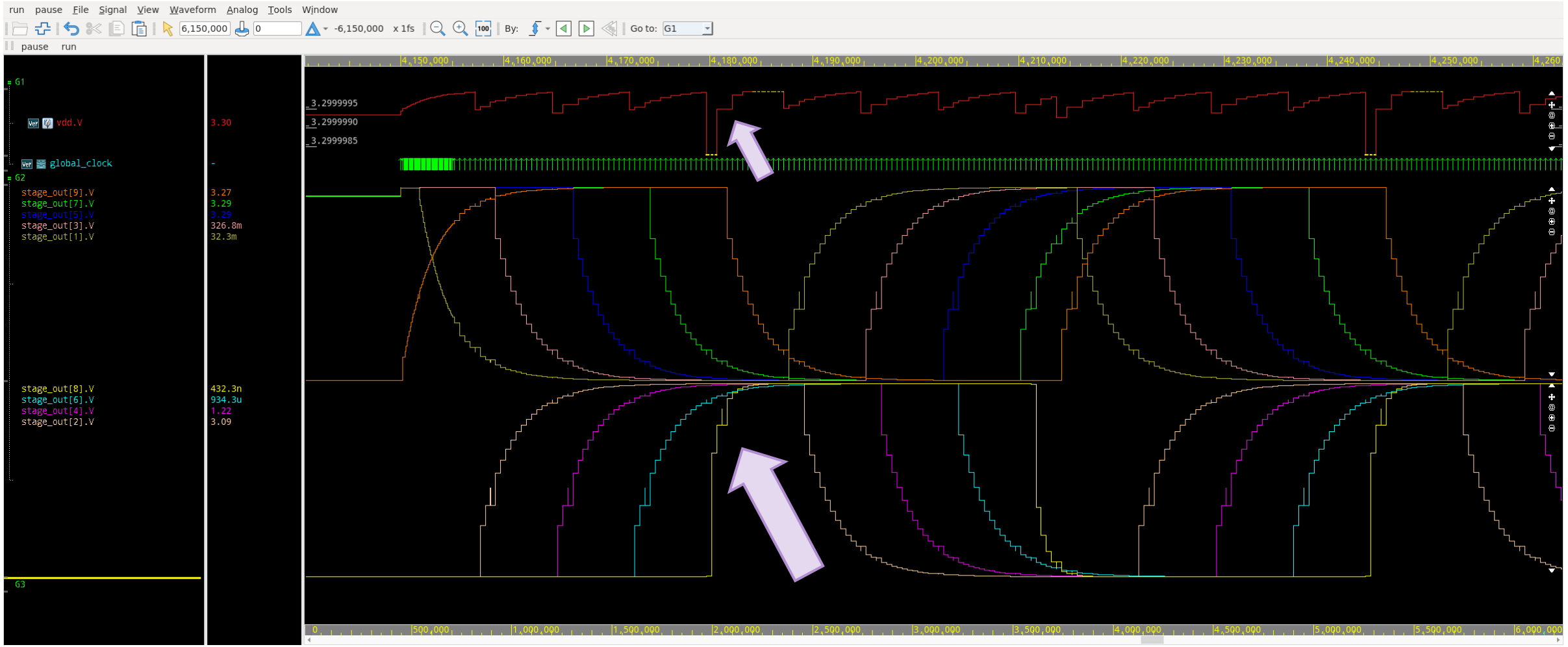
R2R DAC



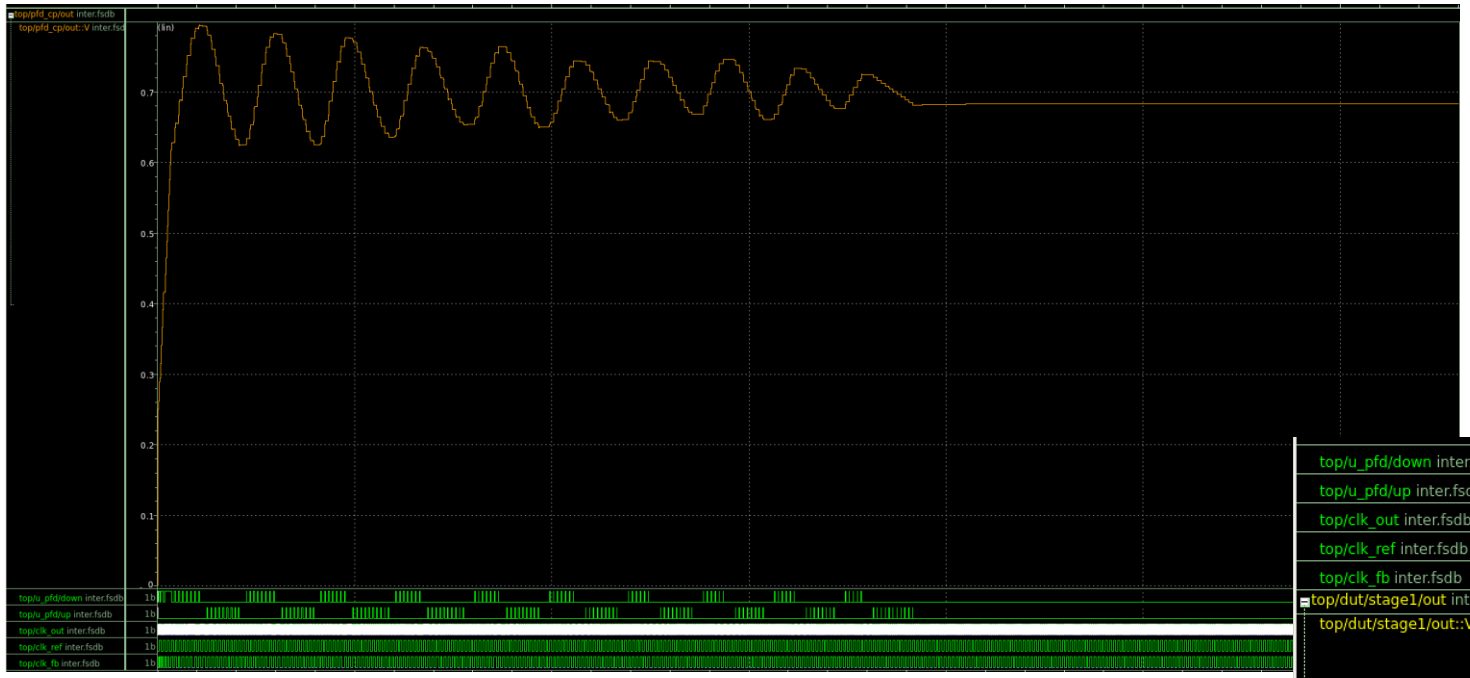
9-stage ring oscillator – various vsup operations (stage 9 with m=2)



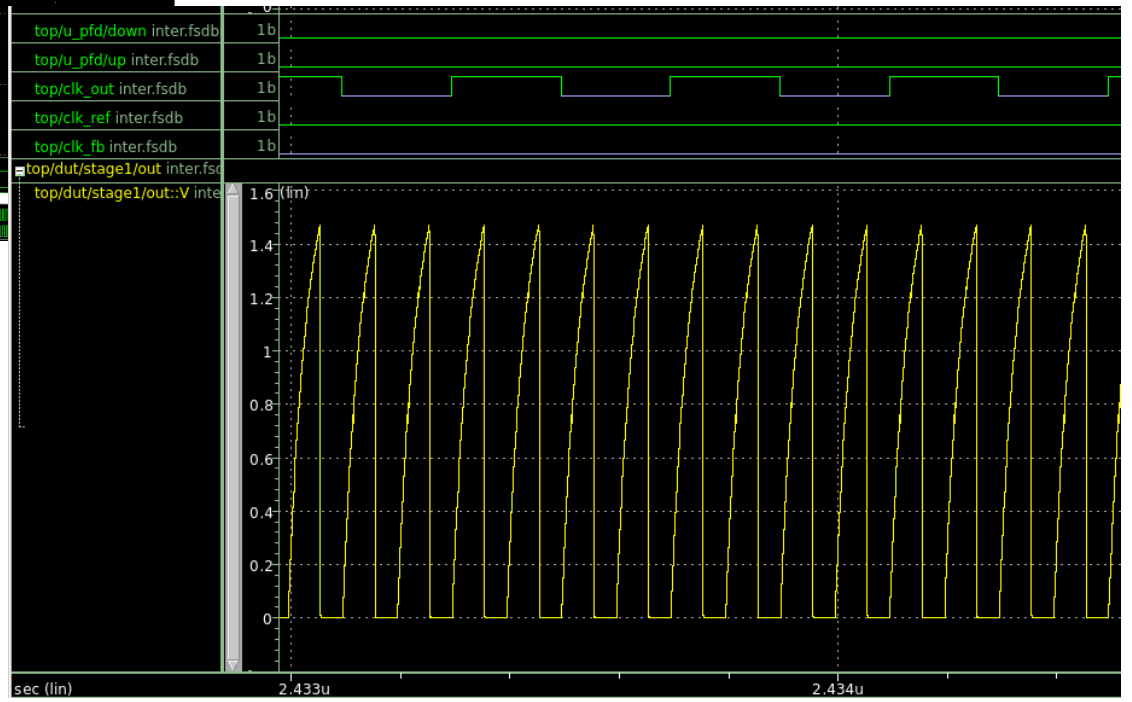
9-stage ring oscillator – supply noise due to recharge of parasitic caps



Analog PLL (10GHz, Nfb=100, Ndiv=4)

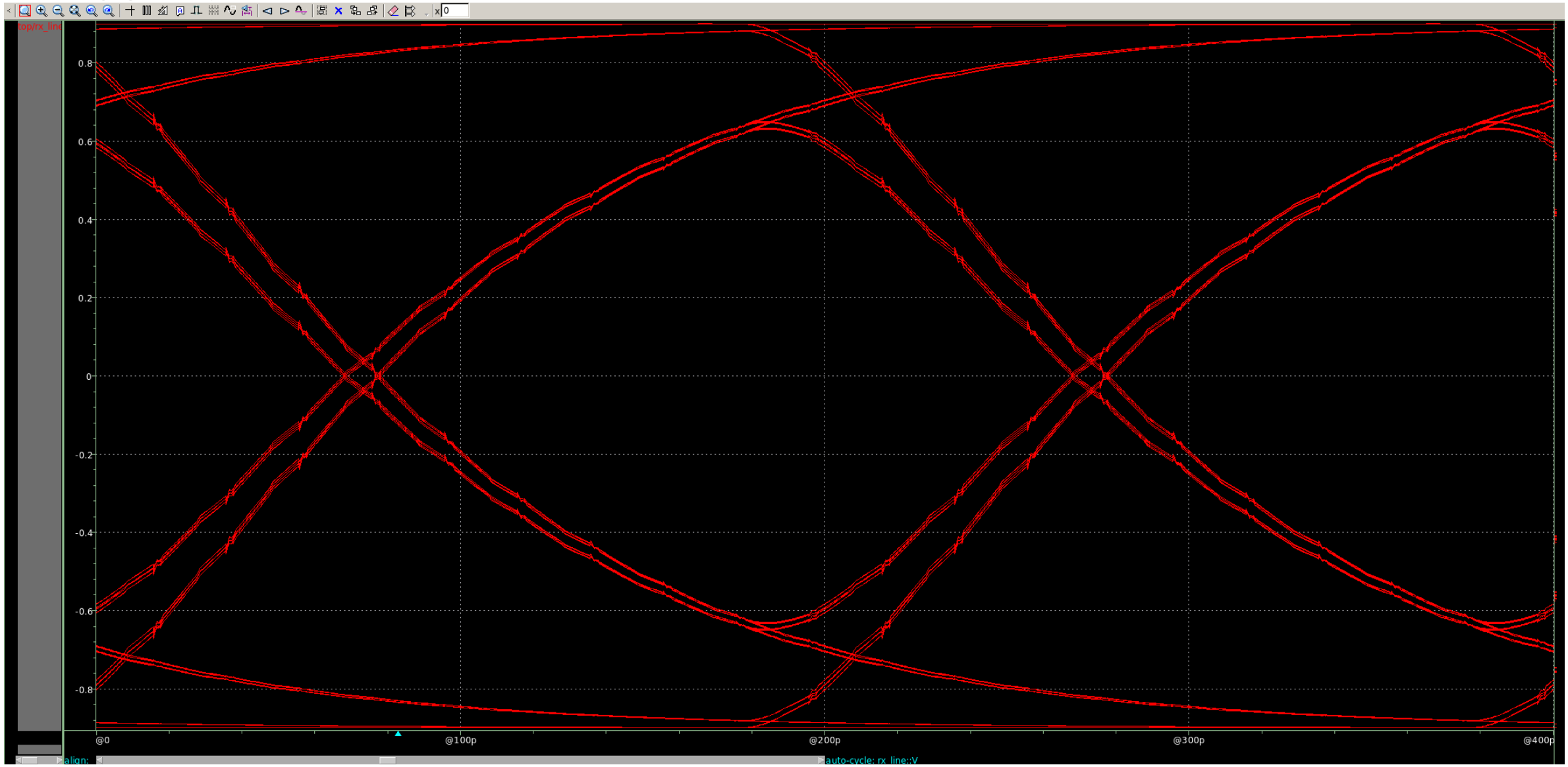


Digital PFD -> Charge-pump -> Analog LPF -> Current-starved ring VCO -> N-div



SerDes TX (voltage mode NRZ, 2.5Gbps)

RX side shown



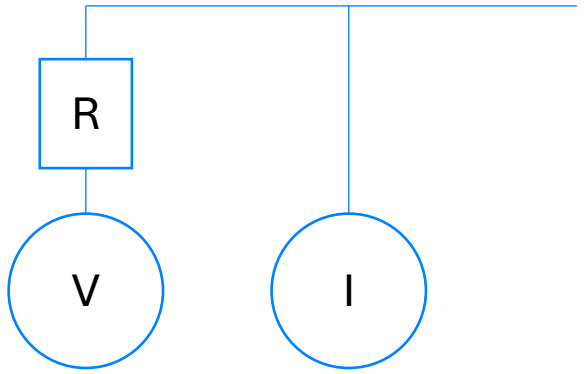
SYNOPSYS®

Crash-course on structural electrical modeling in SystemVerilog

VIR_net

Defined as a user-space package using native SystemVerilog capabilities

```
typedef struct {
  real  V;
  real  I;
  real  R;
} VIR_struct;
```

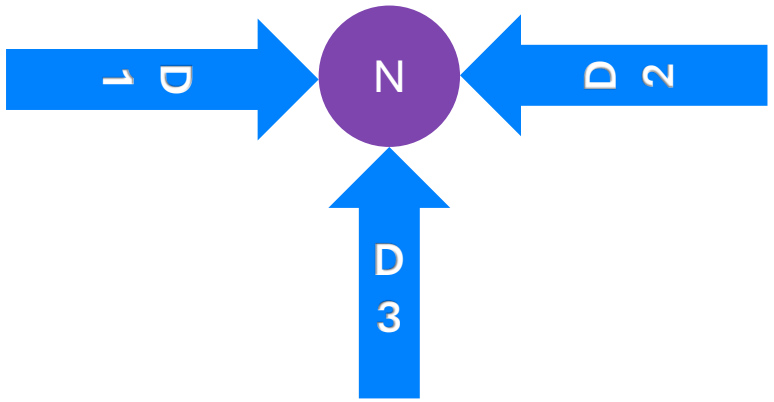



```
function automatic VIR_struct VIR_res (input VIR_struct input[]);

nettype VIR_struct VIR_net with VIR_res;
```

How UDN resolution happens?

A node N's solution: N = f(D1, D2, D3) = {Vn, In, Rn};
In a common case, a node's solution is not equal to any of drivers.

Driver 1:
{V1, I1, R1}



Driver 2:
{V2, I2, R2}

Driver 3:
{V3, I3, R3}

How UDN resolution happens?

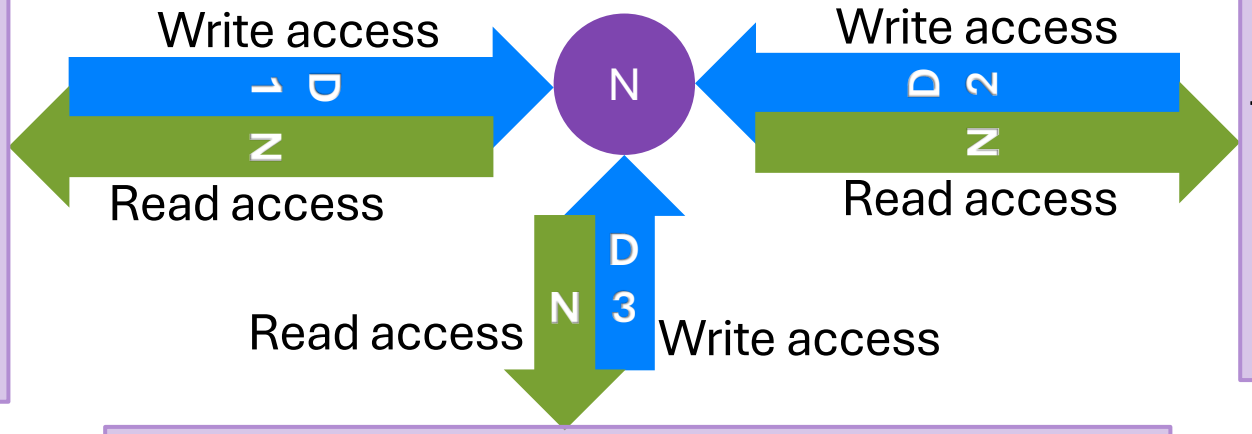
Supported by the simulator (VCS)

A node N's solution: N = f(D1, D2, D3) = {Vn, In, Rn};
In a common case, a node's solution is not equal to any of drivers.

Module 1

Driver 1:
{V1, I1, R1}

Read/sample:
{Vn, In, Rn}



Module 2

Driver 2:
{V2, I2, R2}

Read/sample:
{Vn, In, Rn}

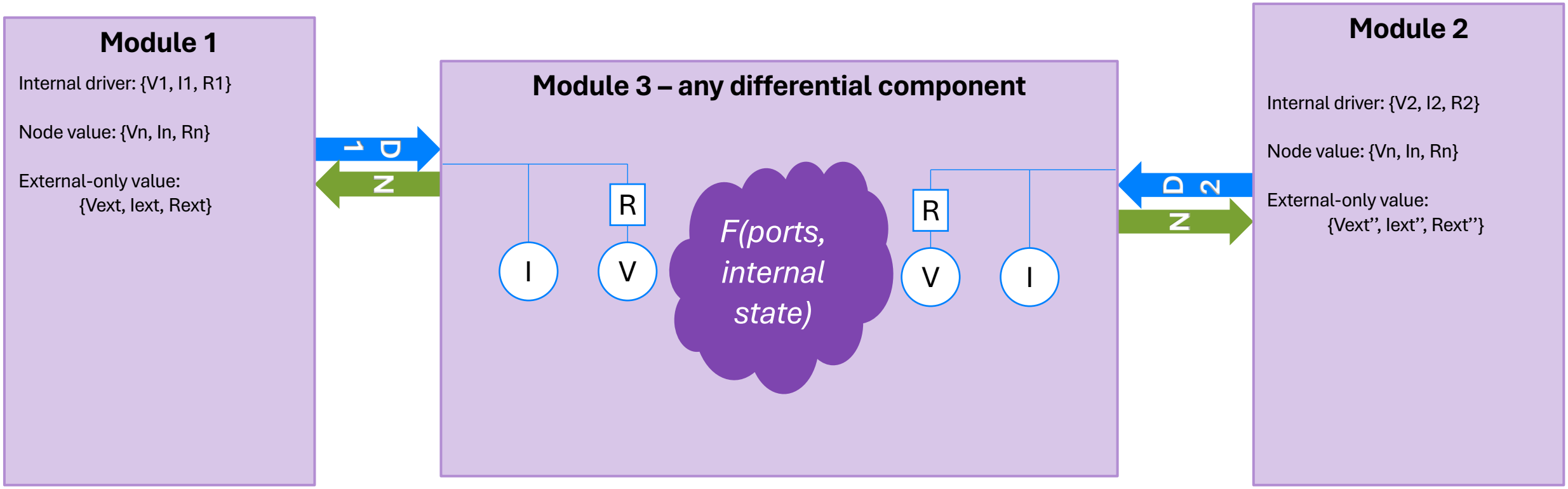
Module 3

Read/sample:
{Vn, In, Rn}

Driver 3:
{V3, I3, R3}

Generic approach to model differential components

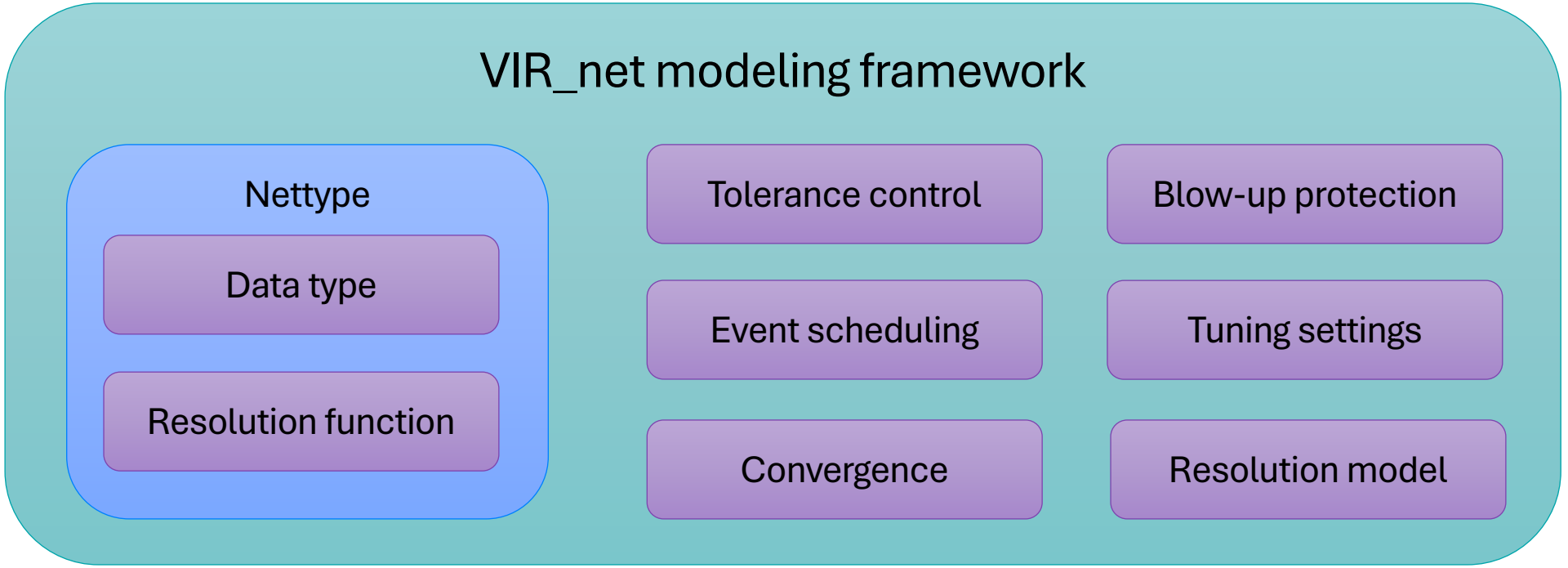
A function of the component to be defined as a class method



Any differential component eventually is represented by a 2-port device, where each port has to generate a {V,I,R}(t) contribution to a respective port such that a reasonable electrical accuracy is achieved throughout the network.

VIR_net – is a modeling framework

- A nettype is simply a combination of a data type, and a resolution function.
- However, when augmented with necessary infrastructure, it becomes a modeling framework.



- Strictly speaking, VIR_net is a PWL electrical simulator with dynamic scheduling written in SystemVerilog.

What is possible to model with the VIR_net framework today?

Tested circuits

- Buck converter
  - Open loop, closed loop, CCM, DCM, HiZ
- 3-level boost converter with a flying capacitor
  - Including activation of protection diodes
- Charge pump source
- Full wave rectifier
- R2R DAC (only matrix of resistors)
- Ring VCO, Cross-couple LC VCO
- Analog PLL
  - Current-starved ring VCO, N-div, PFD, Charge Pump, LPF
- SerDes TX
  - Voltage mode, 2.5Gpbs NRZ
- RC Ladder (3000 sequential RCs) 😤

Available DMS components

- resistor
- capacitor
- inductor
- diode
- tranif1 (a switch)
- vcr
- vcvs
- VIR_GPIO
  - Iterations limiting and tolerance control
  - Scheduling control through a ‘clock_controller’ class.

Well structured architecture of analog primitive modules

Disclaimer: SystemVerilog code snippets throughout the presentation do not reflect production version of the VIR_net modeling framework and serve illustrative purposes only.

```
7  `timescale 1s/1ps
8
9  module capacitorD1
10     import VIR_package::*;
11     import snps_msv_nettype_pkg::*;
12      ( inout VIR_net p, inout VIR_net n);
13
14 parameter real c_val = 1.0e-12;
15 parameter real sampling_rate_s = 1e-9; // Sampling rate in seconds
16 parameter real ic = 0.0;
17
18 VIR_struct ext_p, int_val_p, ext_n, int_val_n, id_t, eq_t;
19 real total_v, total_r, t, Itmp;
20
21 assign total_v = ext_p.V - ext_n.V;
22 assign total_r = ext_p.R + ext_n.R;
23 always @(ext_p.V, ext_n.V, ext_p.R, ext_n.R) eq_t = '{total_v, 0.0, total_r};
24
25 VIR_GPIO p_gpio ( .p(p), .id(int_val_p), .ed(ext_p));
26 VIR_GPIO n_gpio ( .p(n), .id(int_val_n), .ed(ext_n));
27 Primitive prim_inst;
28
29 initial begin : initial_block
30   int_val_p = '{realZ, 0.0, realZ};
31   int_val_n = '{realZ, 0.0, realZ};
32   prim_inst = new("cap", ic, c_val, sampling_rate_s);
33 end
34
35 always @( p_gpio.ed_updated, n_gpio.ed_updated ) begin
36   id_t = prim_inst.update_capacitor_with_esr(eq_t, $realtime - t, p_gpio.ext_HiZ | n_gpio.ext_HiZ, Itmp);
37   int_val_n = '{ext_p.V - id_t.V, 0.0, ext_p.R + id_t.R};
38   int_val_p = '{ext_n.V + id_t.V, 0.0, ext_n.R + id_t.R};
39   t = $realtime();
40 end
41
42 endmodule
```

Header

Parameters

Variables

Structural assigns

Instances

Initialization

Computation loop

1 VIR_GPIO per bidirectional port

1 function call to prim_inst.update_capacitor_with_esr

Easy to extend by defining just one new function

```
552 class Primitive;
553
554   string device;//cap, or ind, or res
555   real dt;
556   VIR_struct id;
557   VIR_struct id_last;
558   real Ieq;
559   real Veq;
560   real value; //for cap - capacitance, for ind - inductance, for diode - Vth, etc.
561   bit status;
562   real ts;
563
564 function new(string device, real ic, real value, real ts);
565   this.device = device;
566   this.id = '{ic, 0.0, 0.0};
567   this.id_last = '{ic, 0.0, 0.0};
568   this.value = value;
569   this.ts = ts;
570   smTs.try_get(1);
571   VIR_package::effective_timestep = this.ts;
572 endfunction
573
574
575 virtual function VIR_struct update_capacitor(input VIR_struct ed, input real dt, input bit HiZ, output real Ieq);
576   if (dt != 0.0 ) begin
577     this.dt = dt;
578     this.id_last = this.id;
579   end
580   if (HiZ) this.id = '{this.id_last.V, 0.0, this.dt/this.value};
581   else this.id = '{(ed.R*this.value*this.id_last.V + this.dt*ed.V)/(ed.R*this.value + this.dt), 0.0, this.dt/this.value};
582   this.Ieq = this.value*(this.id.V - this.id_last.V)/this.dt;
583
584   Ieq = this.Ieq;
585   return this.id;
586 endfunction
```

Internal variables

Constructor

Update function for an ideal capacitor

- Available functions:
  - update_diode
  - update_inductor
  - update_capacitor
  - and more

How switchable components inject time events?

By introducing a 1fs delay

- Untitled-1

```
1  //Inject events if a following pattern:
2  //Trigger -> event
3  //Trigger -> dt -> event
4  //Trigger -> dt*(idx)*DELAY_DURATION_MULT -> event
5  task inject_events(input real dt, input int NUM_DELAYS, input int DELAY_DURATION_MULT);
6    real runtime_dt;
7    int runtime_delays;
8    int runtime_mult;
9
10   if ($value$plusargs("DT=%e", runtime_dt)) dt = runtime_dt;
11   if ($value$plusargs("NUM_DELAYS=%d", runtime_delays)) NUM_DELAYS = runtime_delays;
12   if ($value$plusargs("DELAY_MULT=%d", runtime_mult)) DELAY_DURATION_MULT = runtime_mult;
13
14   -> VIR_package::injected_clock;
15   for (int i = 1; i <= NUM_DELAYS; i++) begin
16     fork
17       automatic int idx = i;
18       begin
19         if (idx < 2) begin
20           #(dt * idx) -> VIR_package::injected_clock;
21         end
22         else #(dt * (idx) * DELAY_DURATION_MULT) -> VIR_package::injected_clock;
23       end
24     join_none
25   end
26 endtask
```

Read simv plusargs

Time event injection based on forked ‘for loop’ iterations

SYNOPSYS®

Case Study: 3-level synchronous boost converter with a ‘flying’ capacitor

Use case: match the reference

Example: 3-level boost converter with Cfly

2025.06-Beta training (Workspace) - Untitled-1

```
/*****************************************************************
 * SYNOPSYS CONFIDENTIAL                                         *
 *                                                               *
 * This is an unpublished, proprietary work of Synopsys, Inc., and *
 * is fully protected under copyright and trade secret laws. You may *
 * not view, use, disclose, copy, or distribute this file or any     *
 * information contained herein except pursuant to a valid written   *
 * license from Synopsys.                                        *
 *****************************************************************/

//3-level boost converter with a flying capacitor
module boost_converter
    import VIR_package::*;
    import snps_msv_nettype_pkg::*;
    (
        input  realnet vin,
        input  bit     s1,
        input  bit     s2,
        input  bit     s3,
        input  bit     s4,
        output VIR_net vbst
    );

parameter real ts = 1e-9; //sampling time target
```

Header, and a target sampling time

```
//Internal nets
VIR_net n3, n1, n4, n5, gnd;

//Ground driver
VIR_struct gnd_dt = '{0.0,0.0,0.0};
assign gnd = gnd_dt;
```

Internal nets, including ground

```
//Convert input voltage real number into a VIR driver woth 1mOhm output impedance
vcvsG v1 ( .out(n3), .control(vin), .control_0(vin), .r_out(0.001), .enable(1'b1) );

//Input serial inductor
inductor #(.l_val( 1.1e-6), .sampling_rate_s(ts)) l1 (n3, n1);
```

Vsrc and serial input inductor

```
//Switching circuitry with protection diodes
tranif1D tg5 (n5, n1, s3);
diode  #(.Vth(0.7)) d5 ( n5, n1 );

tranif1D tg6 (gnd, n5, s4);
diode  #(.Vth(0.7)) d6 ( gnd, n5 );

tranif1D tg4 (n1, n4, s2);
diode  #(.Vth(0.7)) d4 ( n1, n4 );

tranif1D tg3 (n4, vbst, s1);
diode  #(.Vth(0.7)) d3 ( n4, vbst );
```

Switching circuitry and protection diodes

```
//"Flying" capacitor
capacitor #(.c_val(30e-6), .sampling_rate_s(ts), .ic(6.0)) Cfly (n4, n5);

//Output capacitor
capacitor #(.c_val(80e-6), .sampling_rate_s(ts), .ic(5.3)) Cout (vbst, gnd);
```

‘Flying’ and output capacitors

```
46 tranif1D tg4 (n1, n4, s2);
47 diode  #(.Vth(0.7)) d4 ( n1, n4 );
48
49 tranif1D tg3 (n4, vbst, s1);
50 diode  #(.Vth(0.7)) d3 ( n4, vbst );
51
52 //"Flying" capacitor
53 capacitor #(.c_val(30e-6), .sampling_rate_s(ts), .ic(6.0)) Cfly (n4, n5);
54
55 //Output capacitor
56 capacitor #(.c_val(80e-6), .sampling_rate_s(ts), .ic(5.3)) Cout (vbst, gnd);
57
58 //Gshunt
59 assign n1 = '{0.0, 0.0, 1e15};
60 assign n5 = '{0.0, 0.0, 1e15};
61 assign n4 = '{0.0, 0.0, 1e15};
62
63
64 //Dynamic read of internal values
65 always begin
66     #(1ms);
67     pre_sampling(); //Inject a timestep before dumping out probed values
68     $display("[%t]: Vin = %1.3e, Vfly = %1.3e, Vout = %1.3e, Iind = %1.3e, Ts = %1.3
   e", $realtime, n3.V, Cfly.id_t.V, vbst.V, l1.id_t.I, ts);
69 end
70
71 endmodule
```

Shunt resistance

Dynamic reading of internal nets with event injection

Example: 3-level boost converter with Cfly (FCBC)

Liang_Jingyuan_202106_MAS_thesis.pdf

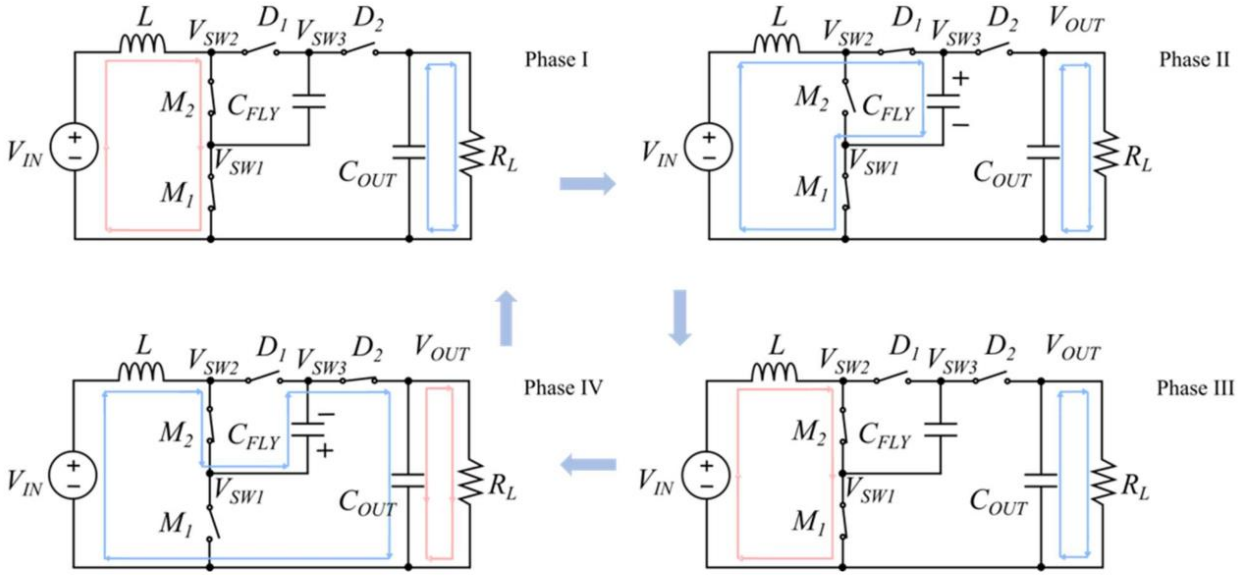


Figure 2.12 Three-level boost converter switching sequence, (a) to (d) are the four phases of the switching sequence.

In phase II and phase IV, the charging and discharging loop of CFLY is denoted in blue. The arrow indicates the direction of the current flow. One thing to be noticed is that the switches involved in the charging and discharging loops are different. During charging, M1 and D1 are in the loop.

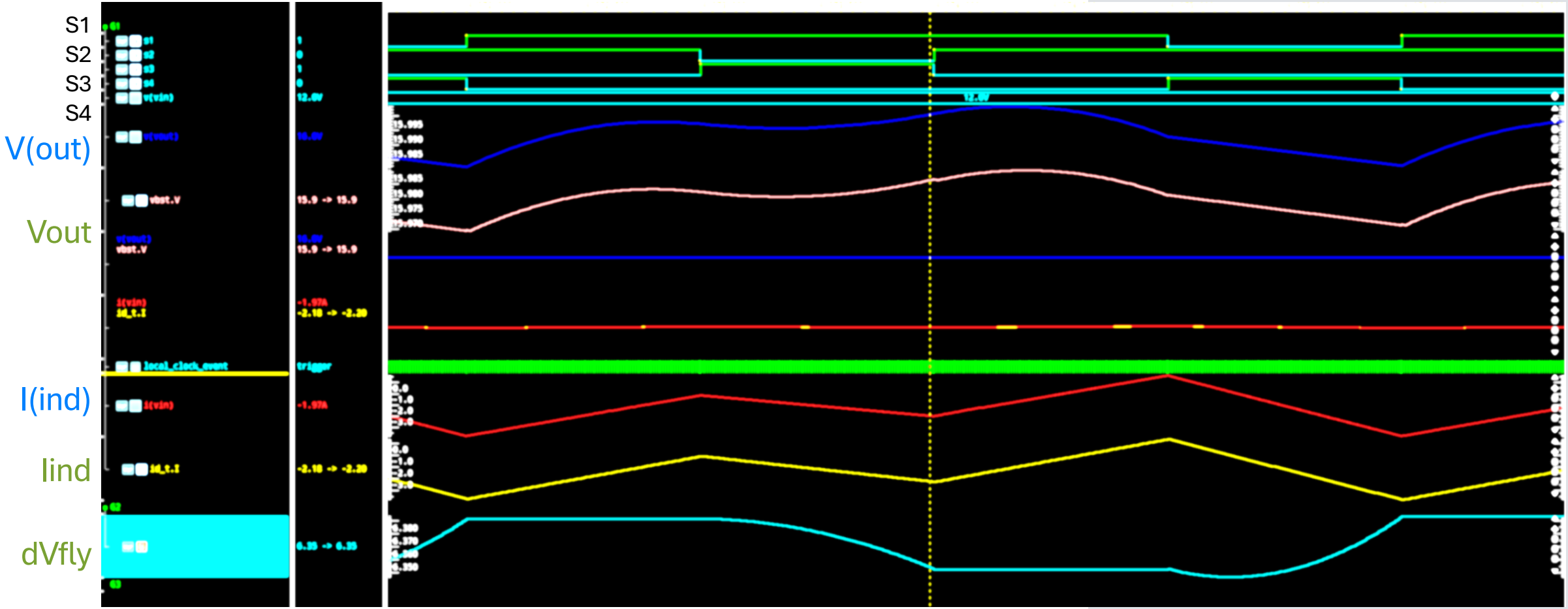
During discharging, M2 and D2 are in the loop. Also, phase IV has an output capacitor COUT in the loop as well. This extra path creates a longer loop, in other words, higher resistance and inductance for discharging CFLY. *Although in an ideal situation, the charging and discharging process should be the same, in the actual implementation, all the above factors will exhibit some differences in the process of charging and discharging the CFLY respectively. This behavior is known as* ***flying capacitor unbalance****, and it imposes a challenge in designing the closed-loop controller for this topology*. One of the control strategies for balancing the flying capacitor can be found in [40]. In this thesis, a manual adjustment on two PWM's duty cycle is performed to achieve a balanced charge on the flying capacitor.

Note: This example is using control phases different from what we have in the testcase. Variety of boost converter architectures is mind blowing, and I haven't been able to find a paper on the example that we've build for a customer. Nevertheless, all architectures are trying to overcome the same challenges.

3-level boost: PrimeSim vs VIR_package

Accuracy validation

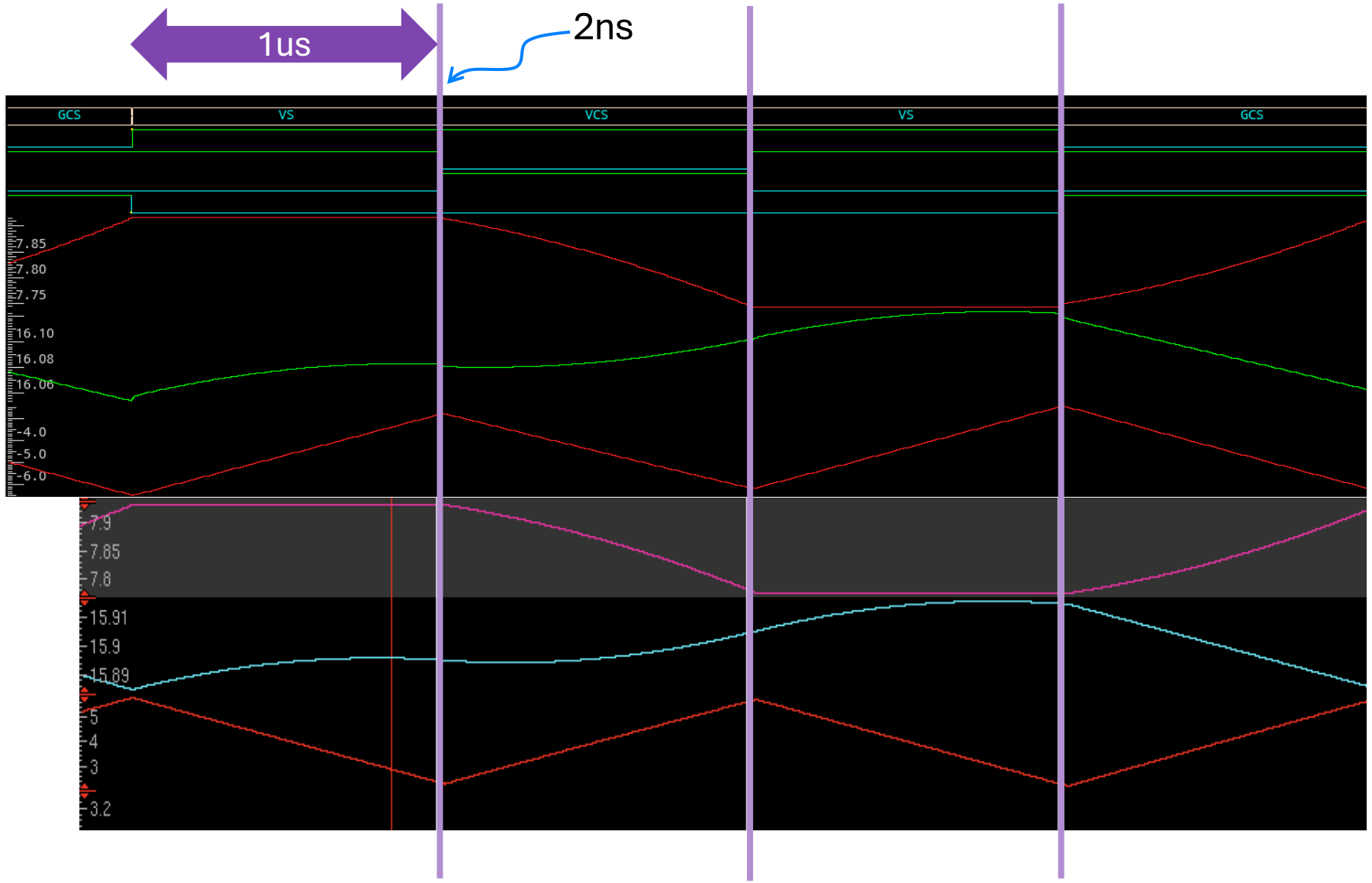
- Ts = 10ns, Vmax = 1kV, max dI/dt (L): 0.1A/ns
- Manually assigned gshunt 1e-15



3-level boost: Synopsys' VIR_net vs 3rd party UDN

Synopsys
Ts=10ns
VIR_net – Synopsys
Primitives – Synopsys

Customer X
Ts=1ns
UDN – 3rd party
Primitives – Customer X



Non-overlapping time of control signals is 2ns; A current signal is inverted.

3-level boost converter with a flying capacitor

- Load: range of 5 Ohm up to 55 Ohm tested.
  - Bigger load reduces Vfly value, Vbst stays the same
- Tran time – **3.5ms** (stability tests performed on 30ms simulations)
- Vfly and Vbst IC = 6.0V
- Vin=12V
- Sampling point – 3ms

Fs target	V(out)	dV(Cfly)	Iind	**CPU time**
1Ghz	15.88	5.75	3.75	**58.6s**
100Mhz	15.83	5.56	4.01	**8.3s**
25Mhz	15.97	5.98	4.30	**3.1s**

- Validated modes: CCM, DCM, HiZ, including electrically correct but unwanted oscillations.

Runtime control of functional parameters

- Runtime control of functional parameters allows for an accuracy change during simulation or through plusargs.
- Support of plusargs enables an easy way to setup parametric sweep to find a sweet spot of accuracy/perf ratio.
- Since simv runs are independent, hundreds of parallel light-weight runs can be launched via LSF.

```
simv +GLOBAL_DT=4e-08  +NUM_DELAYS=5  +DELAY_MULT=500   # ==> VIR_res evals =  10M
simv +GLOBAL_DT=3e-08  +NUM_DELAYS=5  +DELAY_MULT=150   # ==> VIR_res evals =  13M
simv +GLOBAL_DT=2e-08  +NUM_DELAYS=7  +DELAY_MULT=500   # ==> VIR_res evals =  19M
simv +GLOBAL_DT=1e-08  +NUM_DELAYS=7  +DELAY_MULT=150   # ==> VIR_res evals =  31M
simv +GLOBAL_DT=5e-09  +NUM_DELAYS=3  +DELAY_MULT= 10   # ==> VIR_res evals =  54M
simv +GLOBAL_DT=4e-09  +NUM_DELAYS=3  +DELAY_MULT= 50   # ==> VIR_res evals =  67M
simv +GLOBAL_DT=3e-09  +NUM_DELAYS=3  +DELAY_MULT=100   # ==> VIR_res evals =  86M
simv +GLOBAL_DT=2e-09  +NUM_DELAYS=3  +DELAY_MULT=500   # ==> VIR_res evals = 127M
simv +GLOBAL_DT=1e-09  +NUM_DELAYS=7  +DELAY_MULT= 10   # ==> VIR_res evals = 248M
```

- For the given circuit, all sets of parameters listed above are giving correct simulation results (within defined accuracy criteria). It took ~10min of real time to get those parameters, but ~600 LSF jobs.
- VIR_res number directly translates into a simulation performance of the VIR_net-based network. If it's a standalone block-level simulation of the VIR_net-based model, then VIR_res evaluations directly translate into a walltime.

SYNOPSYS®

Case Study: Analog PLL Design

Ground-up functional APLL design in under 4 hours by a person never designed APLL before.

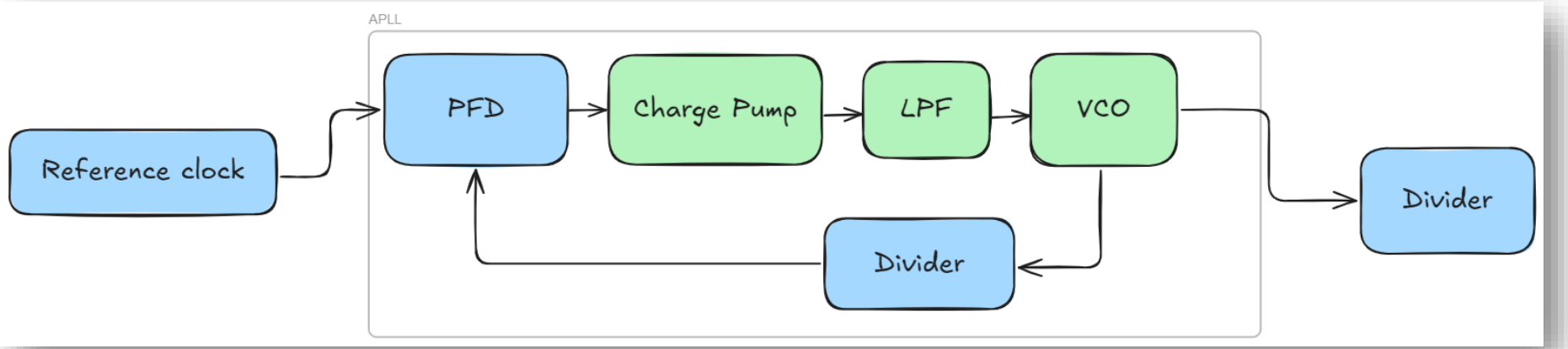
Use case: shift-left in verification cycle

Background on the analog PLL and DMS primitives

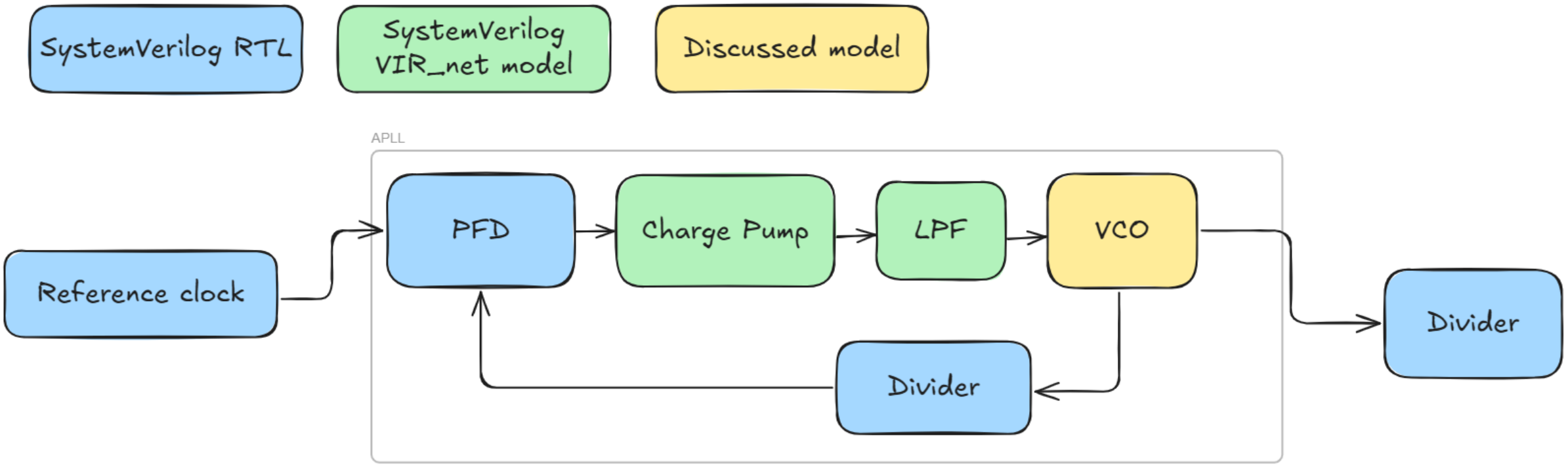
- Necessary analog PLL components
  - Voltage-controlled oscillator
  - Divider
  - Phase Frequency Detector
  - Charge Pump
  - Low-Pass Filter

Available DMS components

- resistor
- capacitor
- inductor
- diode
- tranif1 (a switch)
- vcr
- vcvs



Part 1: Create a VCO



Part 1: Create a VCO, Step 1: Create an inverter

```
1  module inverter
2    import VIR_package::*;
3    import snps_msv_nettype_pkg::*;
4    (
5      inout VIR_net in,
6      inout VIR_net out,
7      inout VIR_net vdd,
8      inout VIR_net gnd
9    );
10
11   parameter real Cout = 0.5e-15; //~~typical output cap for 22nm inv
12   parameter real m = 1;
13
14   bit enable;
15   assign  #(1ps) enable = in.V < Vth + gnd.V ? 1 : 0;
16
17   tranif1D #(.Ron(100/m)) nmost (vdd, out,  enable);
18   tranif1D #(.Ron(100/m)) pmost (gnd, out, ~enable);
19   capacitor #(.c_val(Cout)) C1 (out, gnd);
20
21 endmodule
```

Scoped import

Ports

Parameters

Simple processing of Vth

An inverter

Part 1: Create a VCO, Step 2: Create a ring of inverters, with VCR to vdd

```
module ring_osc
  import VIR_package::*;
  import snps_msv_nettype_pkg::*;
(
  input bit enable,
  inout VIR_net vdd,
  inout VIR_net gnd,
  output VIR_net out
);

VIR_net [5:0] stage_out;
VIR_net [5:0] starved_vdd;
real vctl = 1.0;

vcvsD vcr1 (vdd, starved_vdd[1], 0.0, vctl);
vcvsD vcr2 (vdd, starved_vdd[2], 0.0, vctl);
vcvsD vcr3 (vdd, starved_vdd[3], 0.0, vctl);
vcvsD vcr4 (vdd, starved_vdd[4], 0.0, vctl);
vcvsD vcr5 (vdd, starved_vdd[5], 0.0, vctl);

tranif1D #(.Ron(1e-3)) tg_to_gnd (gnd, stage_out[5], ~enable);

inverter #(.m(0.5)) stage1 (.in(stage_out[5]),.out(stage_out[1]),.vdd(starved_vdd[1]),.gnd(gnd));
inverter #(.m(0.5)) stage2 (.in(stage_out[1]),.out(stage_out[2]),.vdd(starved_vdd[2]),.gnd(gnd));
inverter #(.m(0.5)) stage3 (.in(stage_out[2]),.out(stage_out[3]),.vdd(starved_vdd[3]),.gnd(gnd));
inverter #(.m(0.5)) stage4 (.in(stage_out[3]),.out(stage_out[4]),.vdd(starved_vdd[4]),.gnd(gnd));
inverter #(.m(0.7)) stage5 (.in(stage_out[4]),.out(stage_out[5]),.vdd(starved_vdd[5]),.gnd(gnd));

assign out = stage_out[5];

endmodule
```

Scoped import

Ports

Internal nets

Current ‘starvation’ using vcvsD with variable output impedance

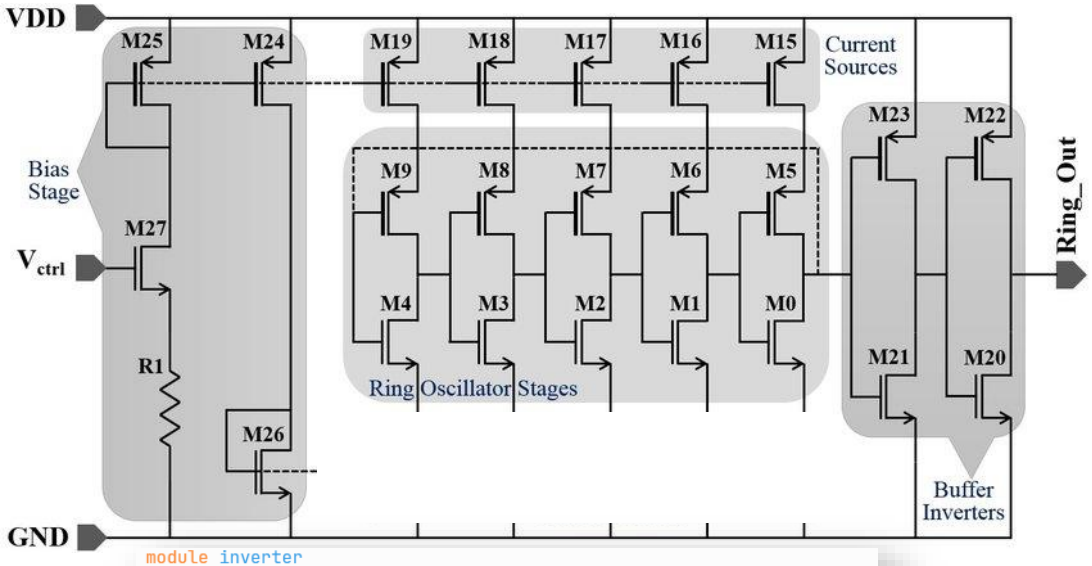
Ports: p, n, **V**ctl, **R**ctl

Enable switch

Actual ring oscillator

Optimization notes

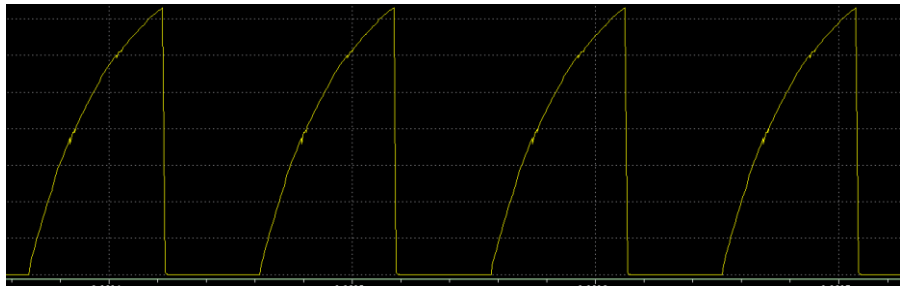
- While structurally this current-starved VCO closely resembles an analog reference, it's not necessary to adhere to the reference.
- In this case, we can redesign an inverter module to have a variable resistance of an open 'channel', and control its resistance directly.
- This eliminates additional components, makes hierarchy more compact, and simulation – faster.




```
vcvsD vcr1 (vdd, starved_vdd[1], 0.0, vctl);
vcvsD vcr2 (vdd, starved_vdd[2], 0.0, vctl);
vcvsD vcr3 (vdd, starved_vdd[3], 0.0, vctl);
vcvsD vcr4 (vdd, starved_vdd[4], 0.0, vctl);
vcvsD vcr5 (vdd, starved_vdd[5], 0.0, vctl);

tranif1D #(.Ron(1e-3)) tg_to_gnd (gnd, stage_out[5], ~enable);

inverter #(.m(0.5)) stage1 (.in(stage_out[5]),.out(stage_out[1]),.vdd(starved_vdd[1]),.gnd(gnd));
inverter #(.m(0.5)) stage2 (.in(stage_out[1]),.out(stage_out[2]),.vdd(starved_vdd[2]),.gnd(gnd));
inverter #(.m(0.5)) stage3 (.in(stage_out[2]),.out(stage_out[3]),.vdd(starved_vdd[3]),.gnd(gnd));
inverter #(.m(0.5)) stage4 (.in(stage_out[3]),.out(stage_out[4]),.vdd(starved_vdd[4]),.gnd(gnd));
inverter #(.m(0.7)) stage5 (.in(stage_out[4]),.out(stage_out[5]),.vdd(starved_vdd[5]),.gnd(gnd));
```

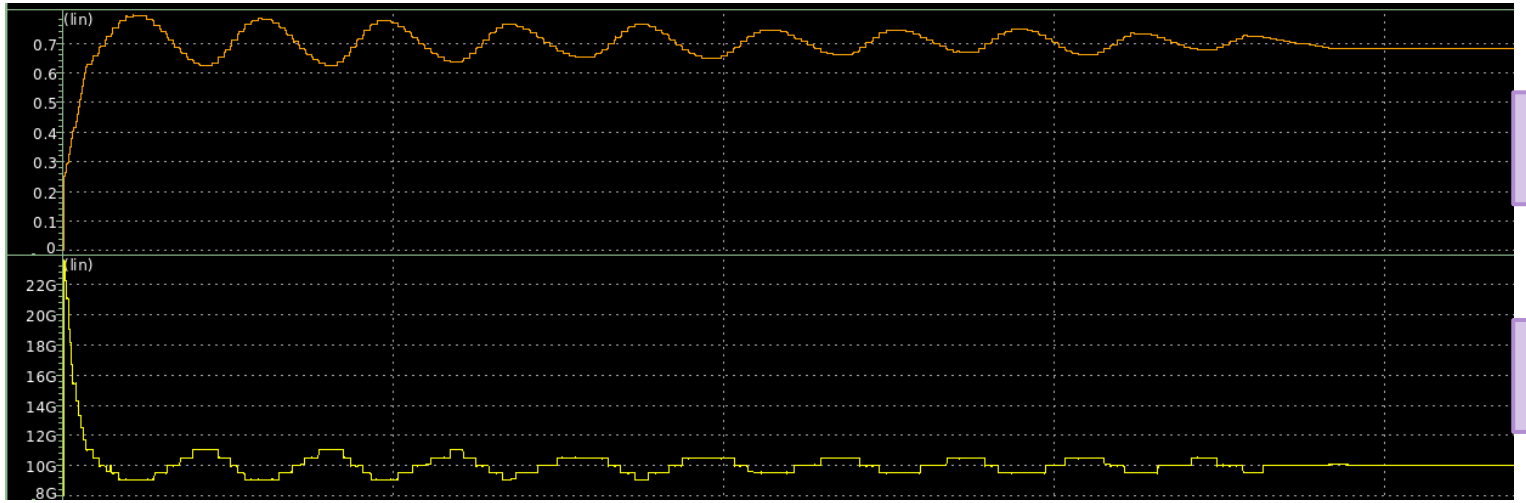
```
module inverter
  import VIR_package::*;
  import snps_msv_nettype_pkg::*;
  (
    inout VIR_net in,
    inout VIR_net out,
    inout VIR_net vdd,
    inout VIR_net gnd
  );

  parameter real Cout = 0.5e-15; //~~typical output cap for 22nm inv
  parameter real m = 1;

  bit enable;
  assign  #(1ps) enable = in.V < Vth + gnd.V ? 1 : 0;

  tranif1D #(.Ron(100/m)) nmost (vdd, out,  enable);
  tranif1D #(.Ron(100/m)) pmost (gnd, out, ~enable);
  capacitor #(.c_val(Cout)) C1 (out, gnd);

endmodule
```

Part 1: Create a VCO, Step 3: Control Vctl variable



```
// VCO control voltage scaling
// Maps charge pump output (typically 0-1.8V) to VCR's target resistance (1k-500k)
// Target working value 0.65V ==> 65k Ohm
always @(pfd_cp_out.V)
  dut.vctl = 100000*pfd_cp_out.V inside {[1e3:500e3]} ? 100000*pfd_cp_out.V : 500e3;
```

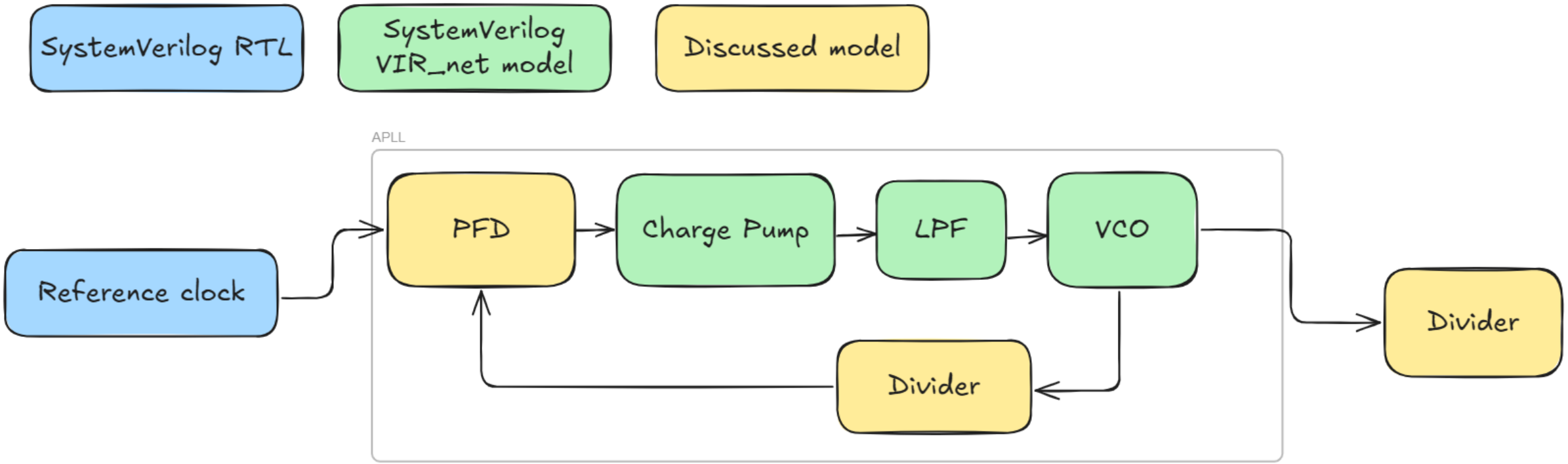
Behavioral control statement for connecting an output of a charge pump to an internal variable that defines a variable resistance of current limiting resistors of a ring oscillator



Vctrl

Frequency

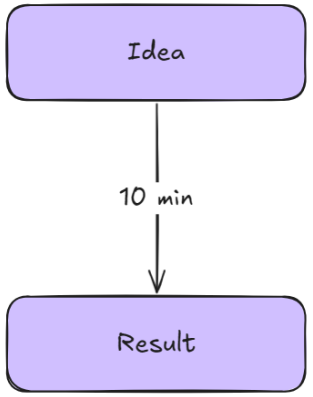
Part 2: Generate digital modules



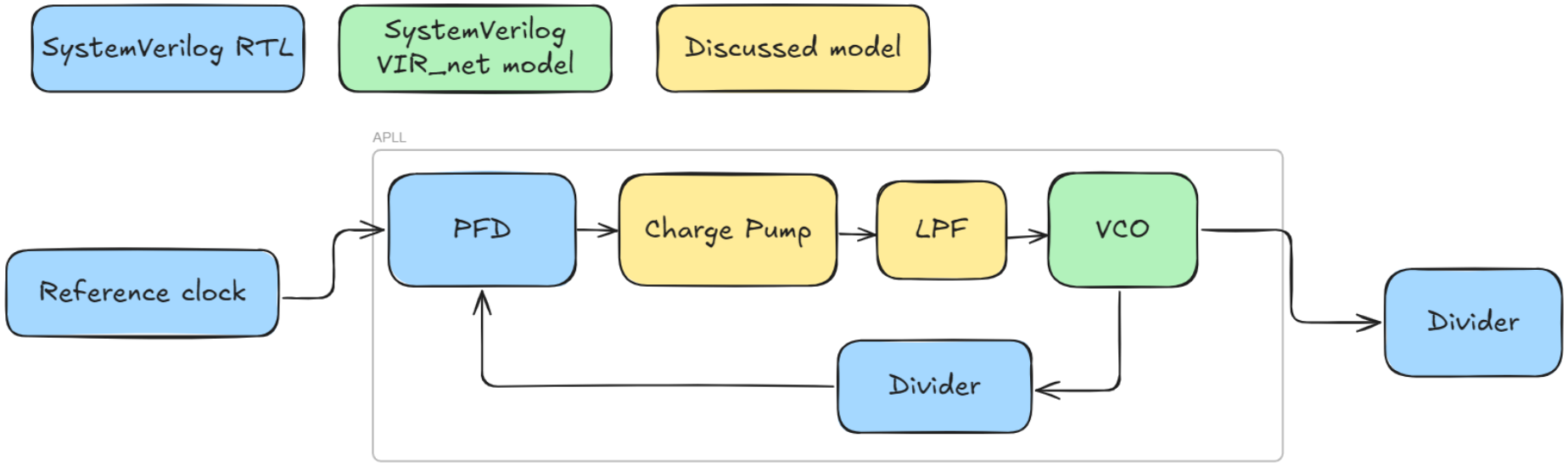
Part 2: Generate digital modules

- Just ask any of available modern LLMs to generate a divider module, and a classic PFD
  - Remind LLM to adhere to SV LRM rules and use a synthesizable subset of the language.

- 2 iterations with LLM



Part 3: Create a charge pump with a built-in LPF



Part 3: Create a charge pump with a built-in LPF

Untitled (Workspace) - Untitled-5

```
module charge_pump
  import VIR_package::*;
  import snps_msv_nettype_pkg::*;
  (
    input logic up,
    input logic down,
    inout VIR_net vdd,
    inout VIR_net gnd,
    inout VIR_net out
  );
parameter real Cout = 1e-9;
parameter real m = 1;

//charge up -> vcr up -> cur down -> freq down
tranif1D #(.Ron(100/m), .NUM_DELAYS(20), .DELAY_DURATION_MULT(500)) nmost (vdd, out,  down);
tranif1D #(.Ron(100/m), .NUM_DELAYS(20), .DELAY_DURATION_MULT(500)) pmost (gnd, out, up);

capacitor #(.c_val(Cout), .sampling_rate_s(1e-12), .ic(0.25)) C1 (out, gnd);

VIR_net t1;
resistor #(.r_val(1.2k), .sampling_rate_s(1e-12)) R1 (out, t1);
capacitor #(.c_val(100p), .sampling_rate_s(1e-12), .ic(0.65)) C2 (t1, gnd);

endmodule
```

Scoped import

Ports

Symmetric charging

Output capacitor

Low pass filter

Verification and design cases that can be studied with this demo

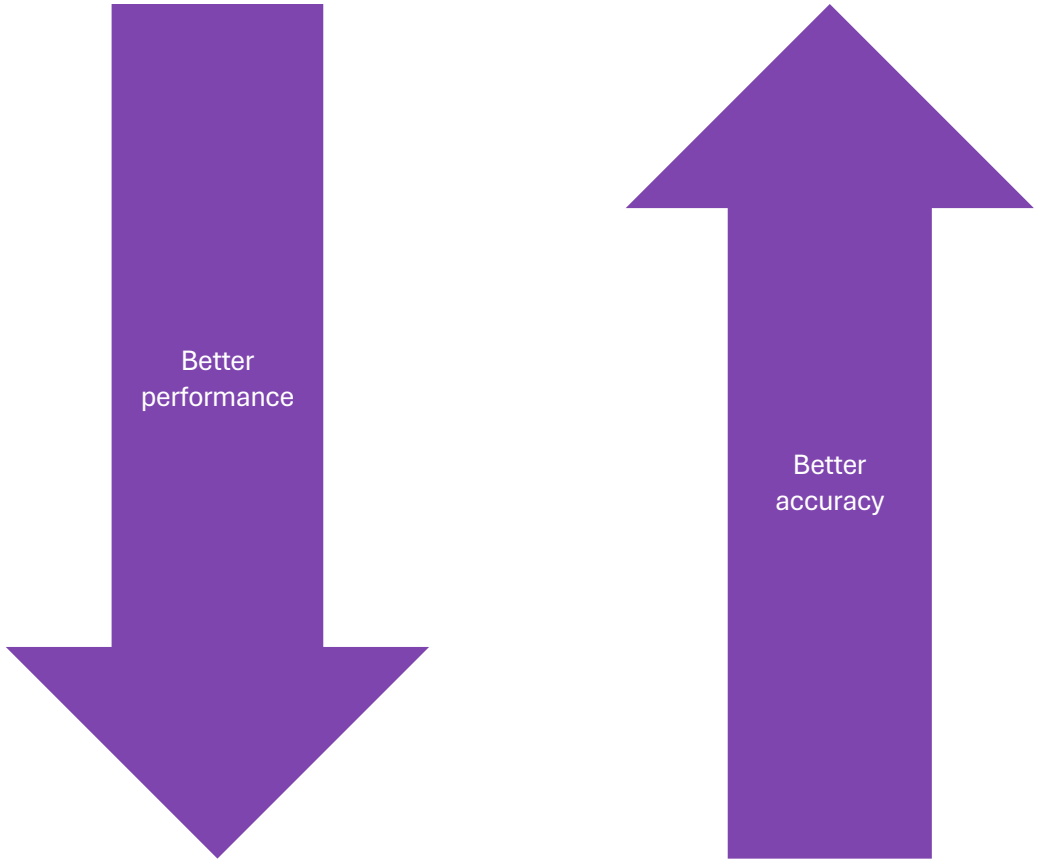
- **Power-up sequence**: which blocks to enable first, do we need handshaking, do we need initial values for currents?
- **VCO design**: should we use ring VCO or LC VCO? Should we have programmable current source? Should we have trimming and in which way?
- **PFD and Divider blocks**: currently they are in RTL – not power-aware(PA). How do we make them PA? Explicit supply ports, UPF, RNM model?
- **What is APLL tuning potential**? Currently Fout = 10Ghz, can we test tuning range?
- Are our abstraction models precise enough to validate VDD noise influence on a process of achieving a lock state?
- Can we replace analog LPF with digital LPF? How?
- Essentially, most of system design and AMS verification methodology related questions can be asked and answered(!) using only SystemVerilog, potentially months ahead of delivery of actual analog blocks.

SYNOPSYS®

Where is the speed-up coming from?

Runtime speed-up per netlist format

- SPF / DSPF (post-layout extracted)
- Spice
- Verilog-A
- Verilog-AMS
- SystemVerilog structural electrical modeling
- SystemVerilog signal flow model
- SystemVerilog GLN
- SystemVerilog RTL



VIR_net is faster than Spice similarly how Spice is faster than DSPF – by abstracting out a lot of functionally irrelevant components considering a required accuracy as an optimization target.
For example, a PVT-stable programmable CMOS current source can consist of hundreds of CMOS transistors + thousands of parasitic components, whereas in VIR_net it's just one primitive – vccs.

Design-time speed-up

- A process of creating a structural VIR_net-based model is very similar to a process of migrating the same design from one PDK to another.
- It can be easily and quickly implemented by a digital verification or analog design engineer with a brief support from each other.
- Support of runtime tuning of simulation parameters allows to run excessive parametric sweeps to get the model which is fast and accurate.
  - Including parameters of the simulation engine, and parameters of used analog primitives (capacitance, inductance, open-channel resistance, etc)

SYNOPSYS®

Future-proof methodology

Easy to extend

Highlights of Synopsys' VIR_package offering

- Extendable architecture
  - User-friendly module's code
  - Class-based equation definition (Backward Euler)
  - Class-based event scheduling
  - Structural awareness (beta)
- Optimized for elaboration and simulation performance
  - Parallel compile, elaborate
  - Multi-core simulation
  - High performance resolution function
  - Fine-grained event control
  - Runtime and file-based save/restore of VIR network
  - Most of functional parameters can be controlled during runtime or through plusargs.
- Global dynamic timestep control
  - Flexible time-step control for optimized simulation precision and stability
  - Time event injection for probing/sampling
  - Non-VIR drivers contribute to timestep selection
  - Period-based recalculation
  - Runtime control of sampling period
- Stability tweaks
  - Handles big step changes of input signals
  - Works with ideal switches and linearized models

SYNOPSYS®

Key Takeaways

Why choose structural modeling with VIR_net over signal-flow models?

Domain	Criteria	VIR_net	Signal flow
Expertise	Analog Design expertise (creation)	😊 Baseline knowledge	😐 To implement specific electrical properties of a circuit, it's a must to know what it is supposed to do
	SystemVerilog expertise (creation)	😊 Baseline syntax knowledge	😐 For high performance models, advanced verification concepts of SystemVerilog are a must
	Easy to understand (reuse)	😊 Yes	😐 No, often internal logic is not trivial
Accuracy	Accuracy of a voltage-mode waveform	😊 High	😊 High
	Can model currents within a system	😊 Yes	😐 Partially
	Built-in support for power nodes noise	😊 Yes	😐 No
Design time	Ease of implementation	😊 High	😐 Low
	Speed of implementation	😳 Mid	😳 Low
	Proof of correctness	😊 Correct by structure	😐 Model validation
	Validation required	😊 Yes, of simulation parameters	😐 Yes, of the core functionality
Simulation time	Simulation speed	😊 High	😊 High
	Performance tuning potential	😊 High	😊 High

Summary

- VIR_net modeling framework is, in essence, a PWL analog simulator written in SystemVerilog.
- A significant turn-around-time gain for a complete verification cycle is achieved by:
  - Lowering an expertise bar of modeling for creation and reuse
  - Offering high accuracy correct electrical responses out-of-the-box
  - Lowering model validation requirements
- VIR_net is a 2nd generation of electrical modeling framework by Synopsys, and is in production since VCS 2025.06
  - Including VIR_package, set of analog primitives, and an extensive library of examples

SYNOPSYS®

Thank you