CONCRETE EXAMPLE OF UVM-MS FOR A DAC

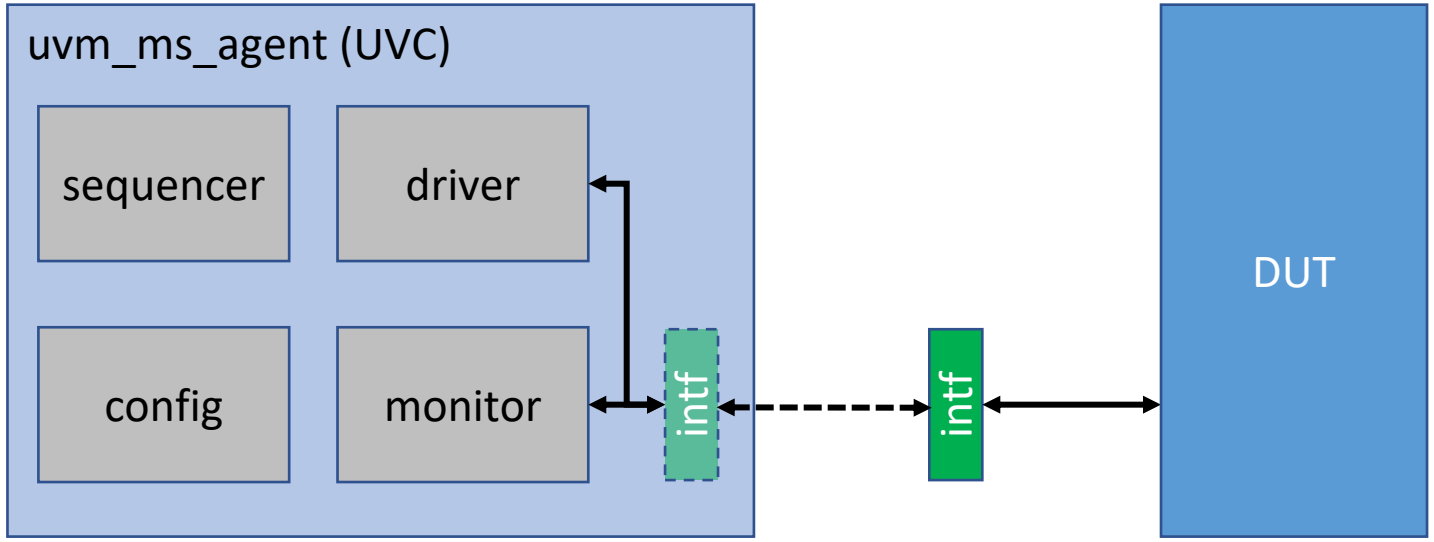
SUBLINE

DATE
STEVEN HOLLOWAY
POSITION, DEPARTMENT,
UNIT
RENESAS ELECTRONICS CORPORATION

AGENDA

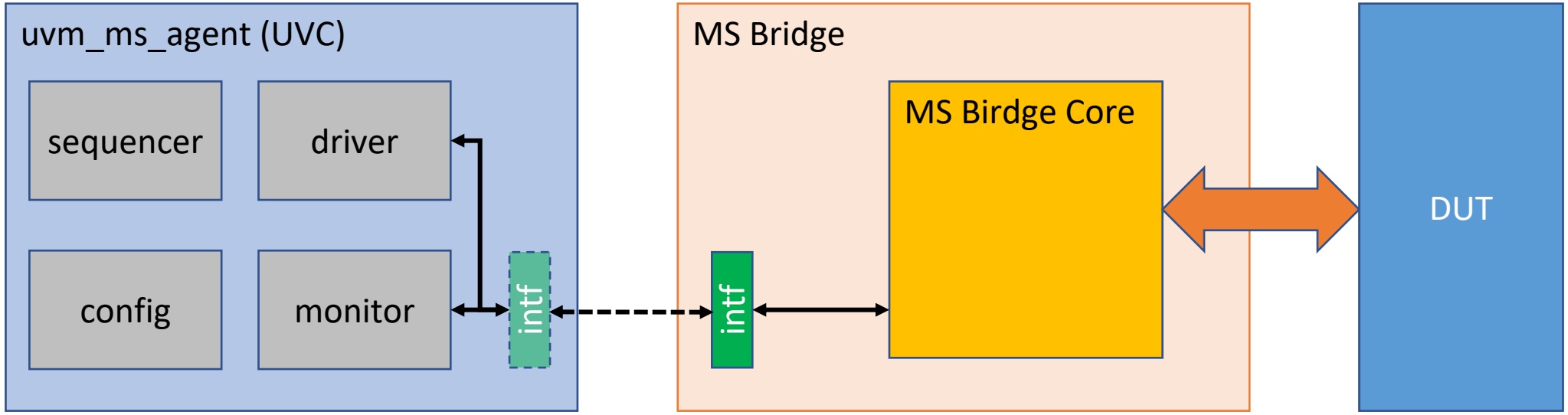
UVM VS UVM-MS

HTTPS://WWW.ACCELLERA.ORG/IMAGES/DOWNLOADS/STANDARDS/UVM-MS/UVM-MS_1.0.PDF



UVM VS UVM-MS

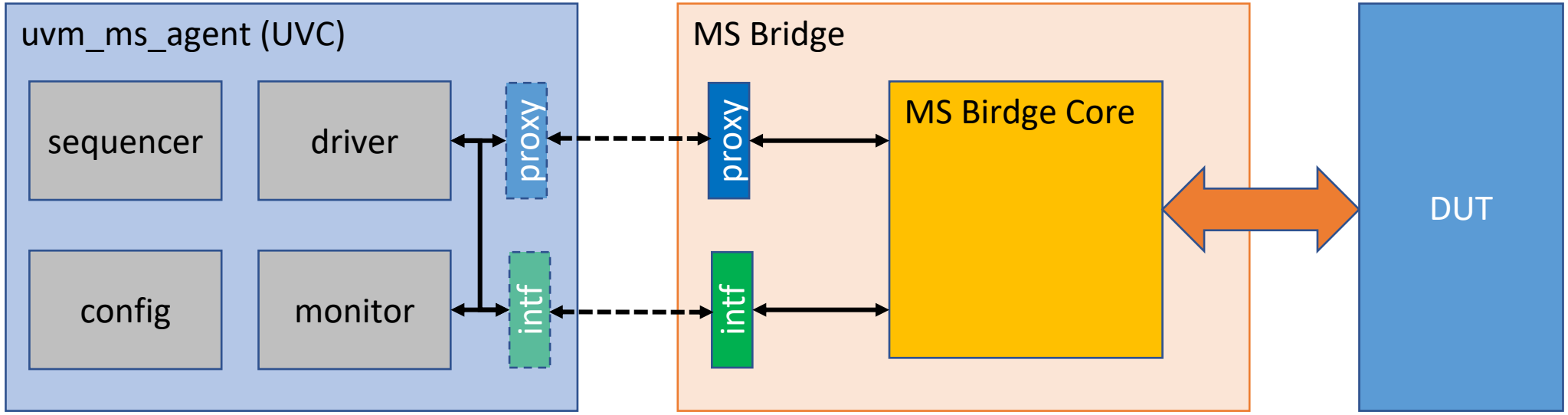
HTTPS://WWW.ACCELLERA.ORG/IMAGES/DOWNLOADS/STANDARDS/UVM-MS/UVM-MS_1.0.PDF



- MS Bridge is the proposed layer that sits between the UVC and the (A)MS DUT
- MS Bridge is a SV module that consists of a proxy API, SV interface, and an analog resource module
- The ‘proxy’ is an API that conveys analog attributes between the UVC and the MS Bridge
- The SV ‘intf’ passes digital/discrete signal values (logic, real, nettype/RNM) between UVC and MS Bridge
- Both ‘proxy’ and ‘intf’ can be used together or individually and use OOMR to MS Bridge Core
- The MS Bridge Core (SV, Verilog or Verilog-AMS)
  - Communication layer between intf/proxy and the ports of DUT
  - Uses the analog attributes from proxy to generate continuously changing values (e.g. ramping voltage supply, electrically modeling drive strengths or cap/res loading, etc.)

UVM VS UVM-MS

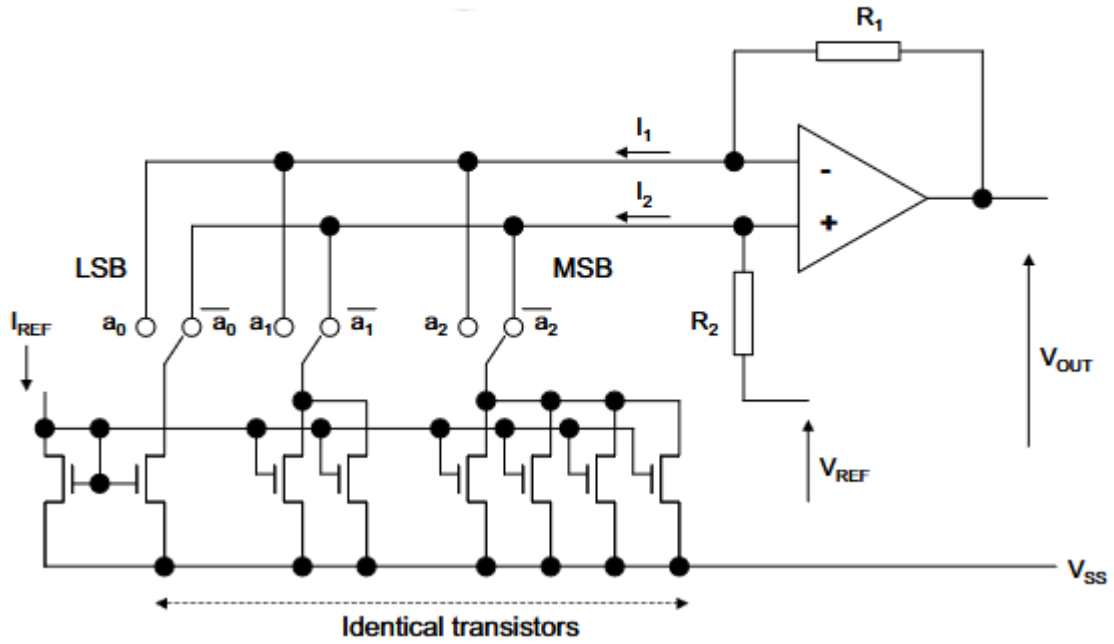
HTTPS://WWW.ACCELLERA.ORG/IMAGES/DOWNLOADS/STANDARDS/UVM-MS/UVM-MS_1.0.PDF



- MS Bridge is the proposed layer that sits between the UVC and the (A)MS DUT
- MS Bridge is a SV module that consists of a proxy API, SV interface, and an analog resource module
- The ‘proxy’ is an API that conveys analog attributes between the UVC and the MS Bridge
- The SV ‘intf’ passes digital/discrete signal values (logic, real, nettype/RNM) between UVC and MS Bridge
- Both ‘proxy’ and ‘intf’ can be used together or individually and use OOMR to MS Bridge Core
- The MS Bridge Core (SV, Verilog or Verilog-AMS)
  - Communication layer between intf/proxy and the ports of DUT
  - Uses the analog attributes from proxy to generate continuously changing values (e.g. ramping voltage supply, electrically modeling drive strengths or cap/res loading, etc.)

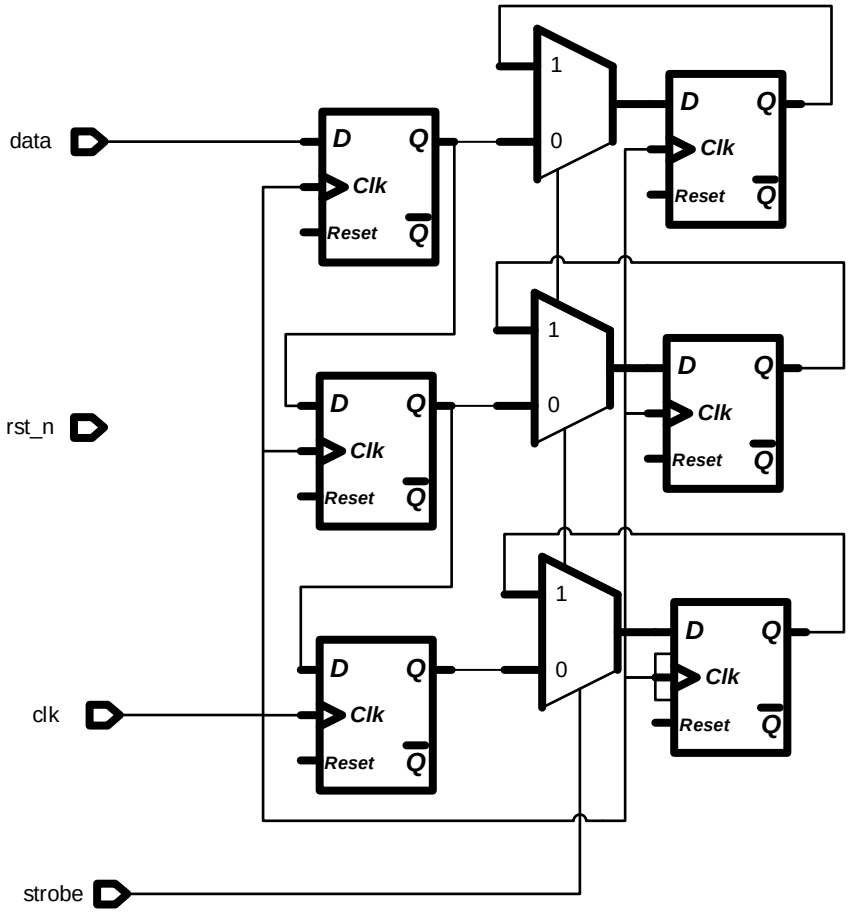
ANALOG 3-BIT CURRENT WEIGHTED DAC

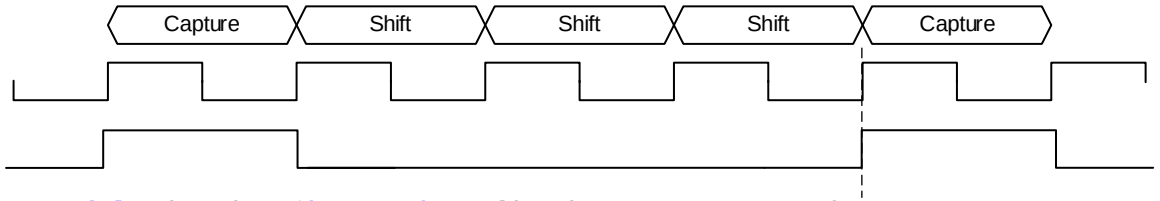
$$V_{OUT} = (V_{REF} - I_2.R_2) + (I_1.R_1)$$




```
module dac(input wire in0, in0n, in1, in1n, in2, in2n,
           input real vdd, vss, vref, iref,
           output real out, output reg clip);

   localparam real R1 = 20.0e3;
   localparam real R2 = 20.0e3;

   real I1, I2;

   always@(*) begin
      //Decode input
      I1 = iref * ($itor(in2 *4.0) + $itor(in1 *2.0) + $itor(in0 *1.0));
      I2 = iref * ($itor(in2n*4.0) + $itor(in1n*2.0) + $itor(in0n*1.0));

      //Calculate output Voltage of Op-AMP
      out = (vref-(I2*R2))+(I1*R1);

      //Output is clipped
      clip = (out > vdd) || (out < vss);

      //clamp to Vdd/Vss
      out = (out > vdd) ? vdd:out;
      out = (out < vss) ? vss:out;
   end
endmodule
```

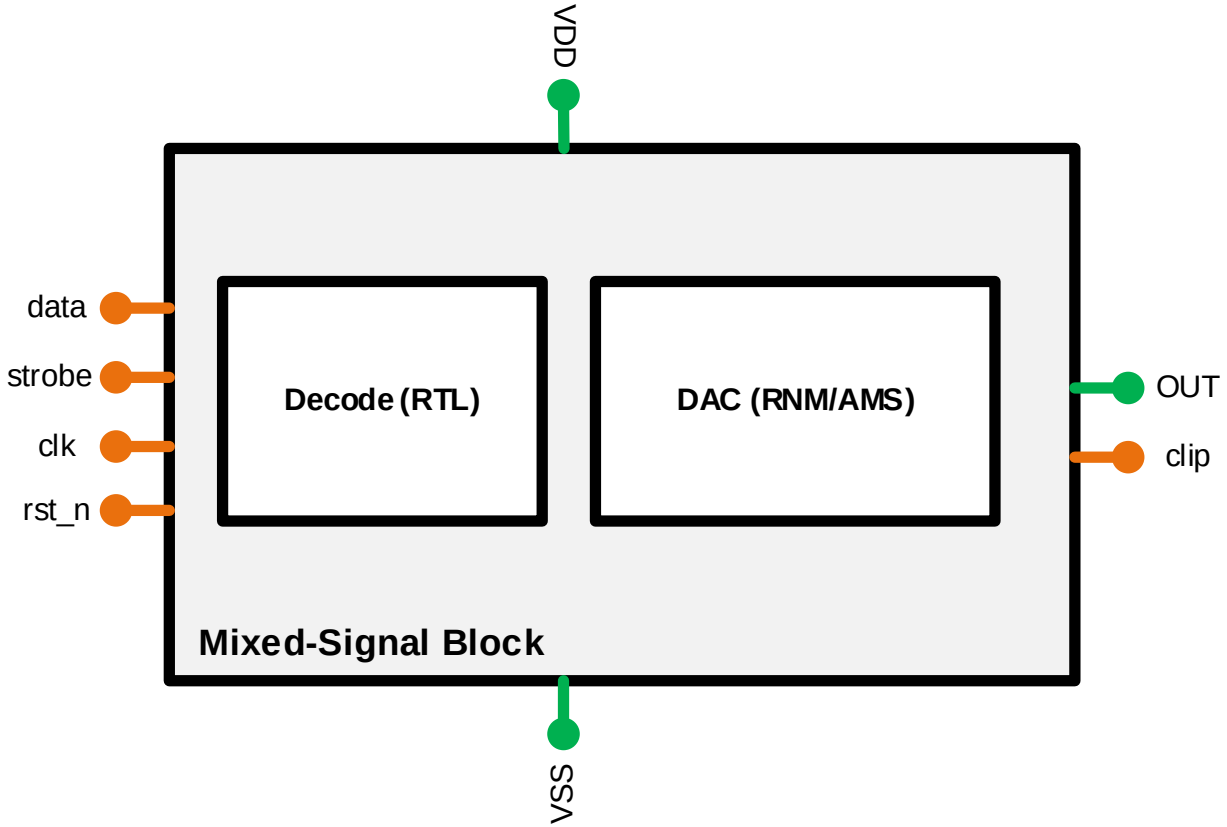
DIGITAL SERIAL TO PARALLEL DECODE






```
module decoder (input wire clk, data, rst_n, strobe,
                output reg in0, in0n, in1,
                           in1n, in2, in2n );
   reg [2:0] sr;
   always @(posedge clk or negedge rst_n) begin
      if (~rst_n) begin
         sr  <= 3'b000;
         in0 <= 1'b0;
         in1 <= 1'b0;
         in2 <= 1'b0;
      end
      else if (strobe) begin
         in0 <= sr[0];
         in1 <= sr[1];
         in2 <= sr[2];
      end
      else begin
         sr <= {sr[1:0], data};
      end
   end
   assign in0n = ~in0;
   assign in1n = ~in1;
   assign in2n = ~in2;
endmodule
```

TOP CELL



- Decoder could be RTL, GLS or transistors
- DAC could be RNM, AMS or transistors
- IO's can have different abstraction based on mixture.
- Can Connect Modules be avoided or limited

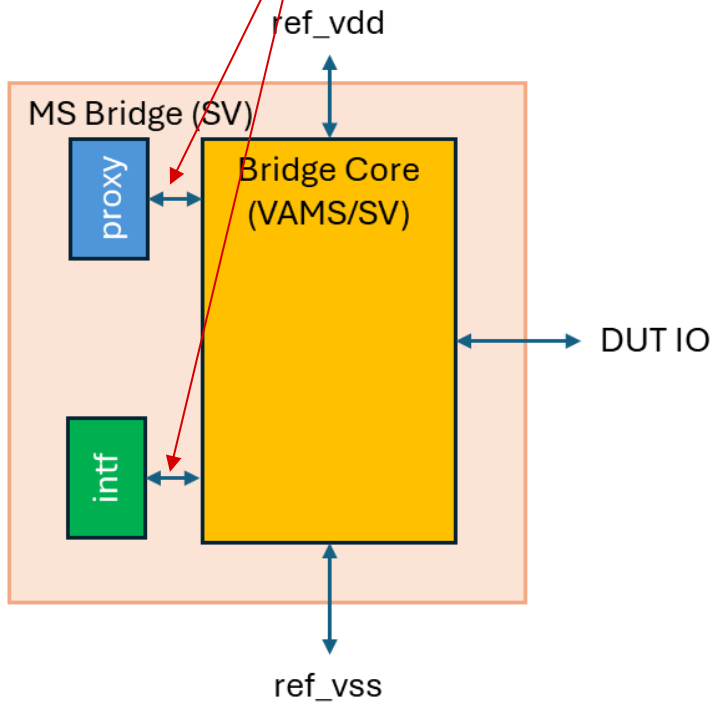
IO could be RNM or Electrical

IO could be Logic, Electrical (or RNM?)

UVM_BRIDGE – LOGIC IO'S

USING TABLE 1 : BRIDGE CONFIGURATIONS OPTION 2 GIVES US THE MOST FLEXIBILITY

Proxy/INTF use **OOMR** to Bridge Core (Section 4.2)

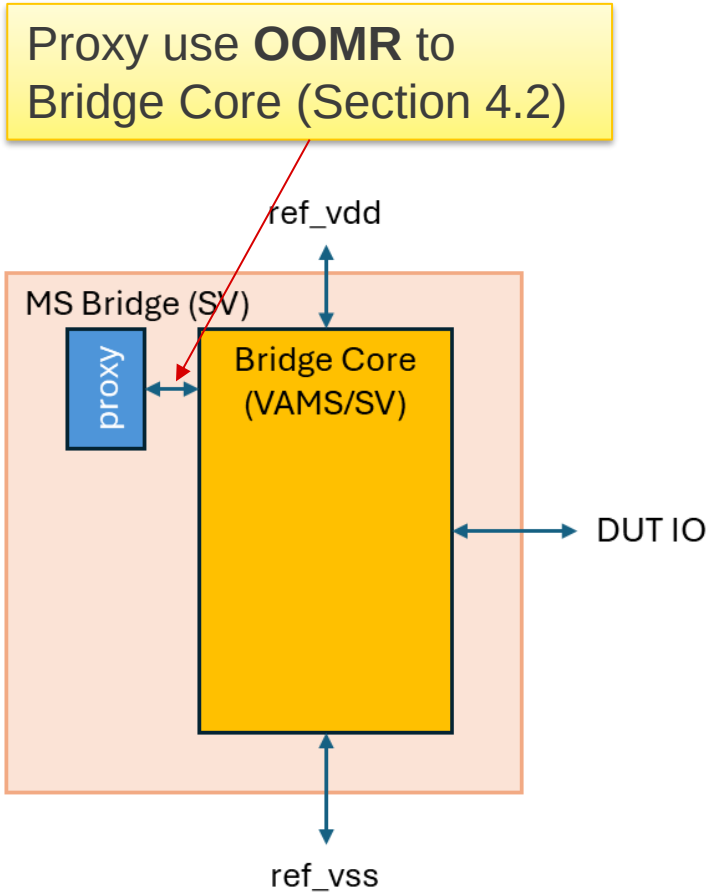


Dut IO	Bridge Core	Proxy	INTF
Logic	SystemVerilog alias between intf and Bridge IO	No used	Logic stimulus
Electrical	Verilog-AMS A2D/D2A converter	Controls Logic to Electrical conversion	
RNM	SystemVerilog R2D/D2R converter	Controls Logic to RNM conversion	

While a Connect Module could be used the ability to control aspect if it dynamically are lost.

UVM_BRIDGE – REAL/ELECTRICAL

USING TABLE 1 : BRIDGE CONFIGURATIONS OPTION 3 GIVES US THE MOST FLEXIBILITY



Dut IO	Bridge Core	Proxy
Electrical	Verilog-AMS	Controls RNM Stimulus
RNM	SystemVerilog	Controls Electrical Stimulus

Proxy provides properties to the Bridge Core to generate stimulus.

- Voltage/Current Source or Probe only
- Value to drive

MS PROXY EXAMPLE

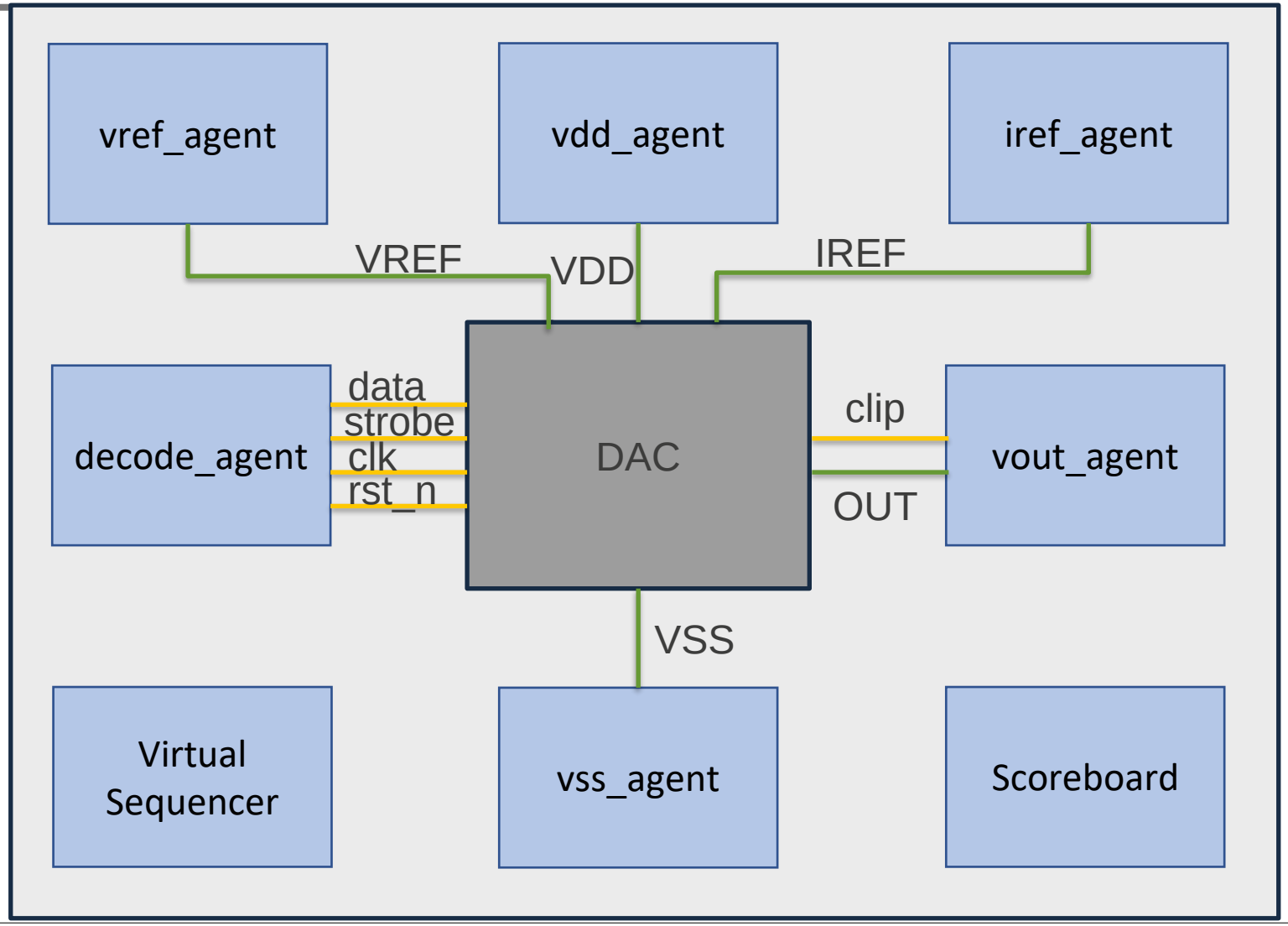
```
virtual class vi_driver_proxy;
   pure virtual function vi_driver_config getParameters();
   pure virtual function void setParameters(vi_driver_config cfg);
   pure virtual function void setResistor (real val, real tr = 1e-9, real tf = 1e-9);
   pure virtual function void setRampVoltage (real val, real tr = 1e-9, real tf = 1e-9);
   pure virtual function void setSinusVoltage(real val, real tr = 1e-9, real tf = 1e-9, real freq = 0.0);
   pure virtual function void setRampCurrent (real val, real tr = 1e-9, real tf = 1e-9);
   pure virtual function void setSinusCurrent(real val, real tr = 1e-9, real tf = 1e-9, real freq = 0.0);
   pure virtual function real getVoltage (string sval = "");
   pure virtual function real getCurrent (string sval = "");
   pure virtual function real getResistor(string sval = "");
endclass: vi_driver_proxy
```

Template (API)

```
class AMSProxy extends vi_driver_proxy;
   function automatic vi_driver_config getParameters();
      vi_driver_config cfg= new();
      cfg.vi_mode = i_core.P__VI_MODE;
      cfg.vdc = i_core.P__VDC;
      ...
      return(cfg);
   endfunction

   function automatic void setParameters(vi_driver_config cfg);
      i_core.vi_mode = cfg.vi_mode;
      ...
   endfunction
   ...
endclass
```

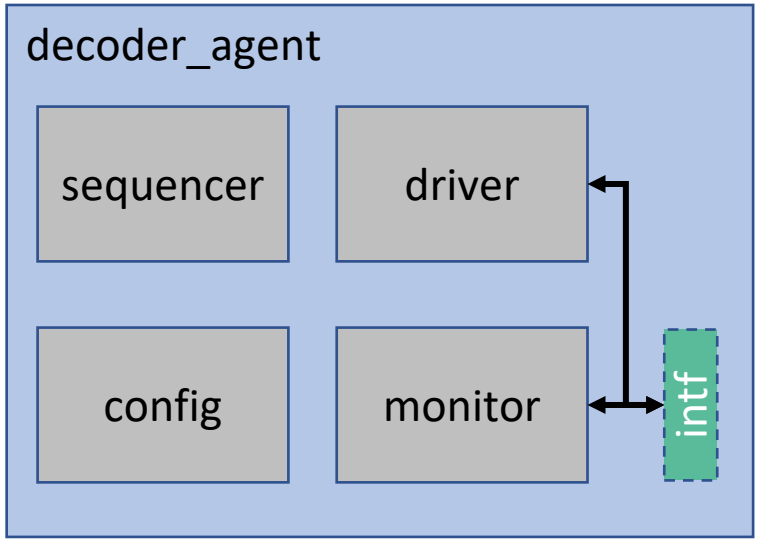
Method Implementation
(in MS Bridge Module)

UVM ARCHITECTURE



DECODER AGENT

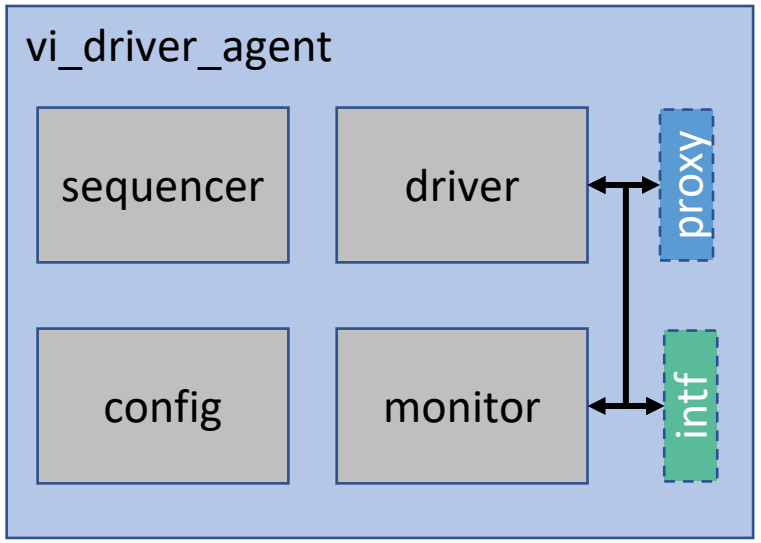
```
class decoder_agent extends uvm_agent;

   decoder_sequencer sqr;
   decoder_driver    drv;
   decoder_monitor   mon;
   decoder_vif       vif;

   `uvm_component_utils(decoder_agent)

   . . .
endclass
```




```
class decoder_driver extends uvm_driver#(decoder_transaction);

  decoder_vif vif;

  `uvm_component_utils (decoder_driver)

  function new (string name, uvm_component parent);
    super.new (name, parent);
  endfunction: new

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    . . .
  endfunction: build_phase

  task run_phase(uvm_phase phase);
    get_and_drive();
  endtask: run_phase

  virtual task get_and_drive();
    . . .
  endtask: get_and_drive

endclass: decoder_driver
```

VI DRIVER AGENT

```
class vi_driver_agent extends uvm_agent;

  vi_driver_config              cfg;
  vi_driver_sequencer           sqr;
  vi_driver_driver              driver;
  vi_driver_monitor             monitor;
  vi_driver_proxy               proxy;
  virtual vi_driver_if          vif;

  `uvm_component_utils(vi_driver_agent)
   . . .
endclass: vi_driver_agent
```



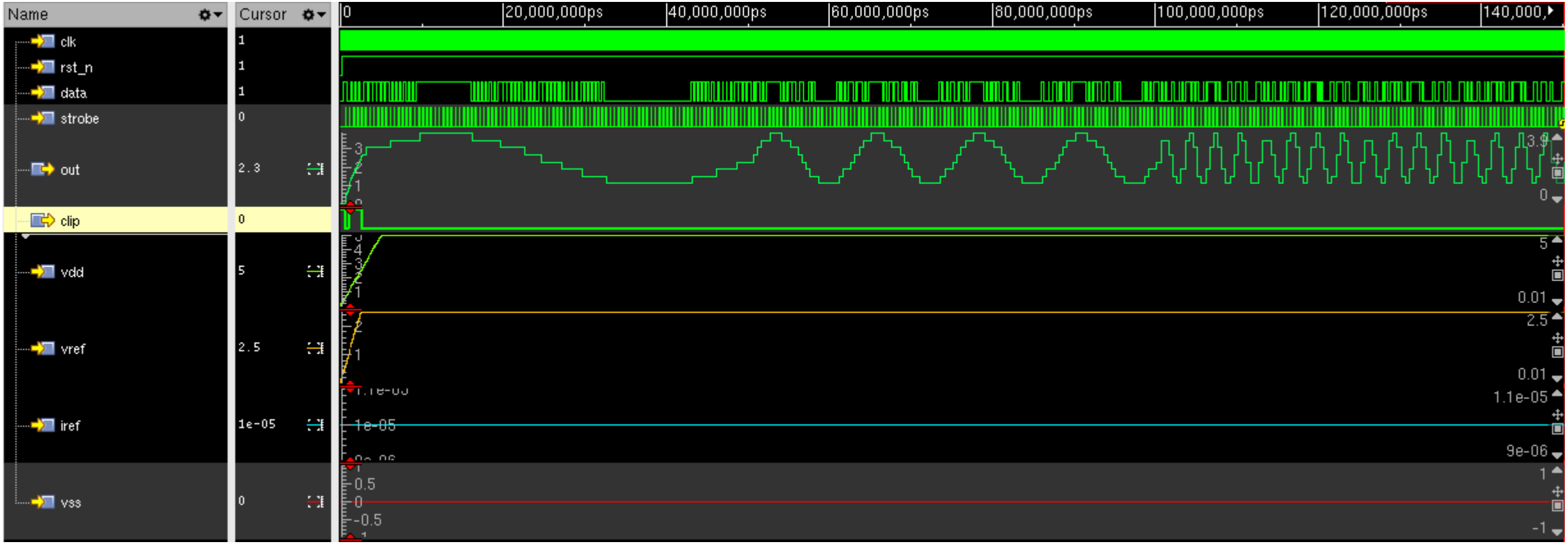

```
class vi_driver_driver extends uvm_driver#(vi_driver_txn);

  vi_driver_proxy      proxy;
  virtual vi_driver_if vif;

  `uvm_component_utils (vi_driver_driver)

  . . .
  virtual task drive_txn(vi_driver_txn txn);

    // Execute txn drive delay
    #txn.delay;

    // Analogue drives via proxy - example scheme
    if (txn.ohms) begin
        proxy.setResistor(txn.ohms, txn.trise, txn.tfall);
        wait(proxy.res_eot); // VAMS handshake in proxy
    end

    // Non-blocking drive for sinus
    proxy.setSinusVoltage(txn.vsin, 1e-9, 1e-9, txn.hertz);

    // Blocking drive for DC offset (until vdc reached)
    proxy.setRampVoltage(txn.vdc, txn.trise, txn.tfall);
    wait(proxy.vdc_eot); // VAMS handshake in proxy

    // Logic drive via interface
    vif.logic_mode_en = txn.logic_mode_en;
    vif.sda_en = txn.sda_out;
  endtask : drive_txn

endclass: vi_driver_driver
```

SEQUENCES

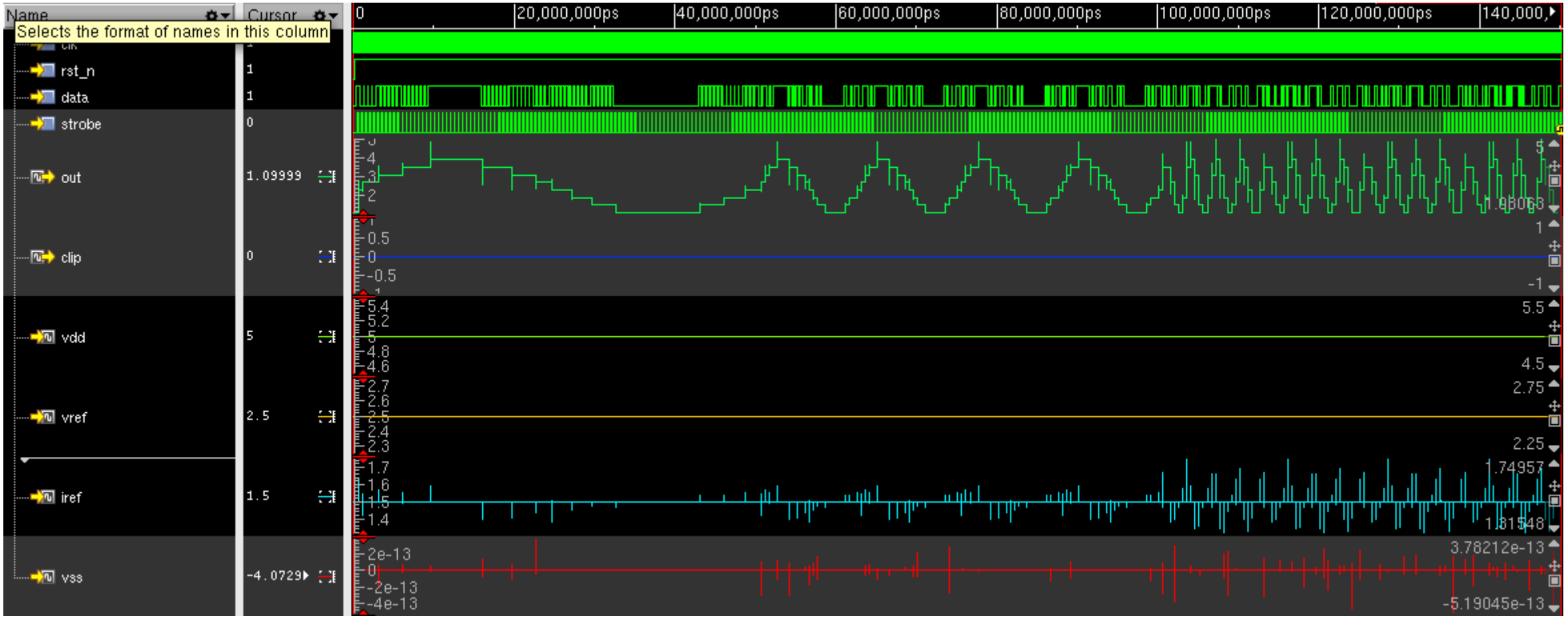
```
class decoder_sin_seq extends decoder_base_seq;
  `uvm_object_utils(decoder_sin_seq)

  rand int  srate;
  rand real freq;
  ...
  task body;
    ...
    repeat (samples) begin
      item = decoder_transaction::type_id::create("item");
      start_item(item);
      ...
      sample = int'((amp * $sin(`TWO_PI * freq * t)) + amp);
      if(!item.randomize() with { data == sample; } ) begin
        `uvm_error("SEQ", "item randomization failed")
      end
      finish_item(item);
    end
  endtask: body

endclass: decoder_sin_seq
```

```
class vsaw_seq extends uvm_sequence #(vi_driver_txn);

  `uvm_object_utils(vsaw_seq)

  real vlower;
  real vupper;
  real trise;
  real tfall;
  int unsigned cycles;
  ...
  virtual task body();
    req = vi_driver_txn::type_id::create("req");
    req.trise  = trise;
    req.tfall  = tfall;

    repeat (cycles) begin
      req.vdc = vupper;
      start_item(req);
      finish_item(req);

      req.vdc = vlower;
      start_item(req);
      finish_item(req);
    end

  endtask: body

endclass: vsaw_seq
```

RNM SIMULATION



AMS SIMULATION



CONCLUSIONS

UVM-MS is the standardisation of analogue/mixed signal extensions for UVM

- Allows UVM to be more mixed-signal aware
- Improved verification of analogue/mixed-signal designs
- Same degree of thoroughness for both analogue and digital parts

Metric-driven verification suits following objectives due to verification space size

- Verifying analogue performance under large set of digital configurations
- Digital control system transitions interacting with analogue functions
- Dynamic control between analogue & digital circuits under wide range of conditions
- Finding problems with A/D interaction in unexpected corner cases

Named UVM-MS as the focus is to support any MS system; DMS, RNM, Spice or a mixture.

Randomisation is not mandatory and benefits are gained even when using directed tests

- Standard methodology
- Plug & play reuse of existing UVM components
- Rich debug & messaging scheme integrated with simulator

Renesas.com