2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Migrating from UVM to UVM-AMS

Accellera UVM-AMS Working Group
Tom Fitzpatrick, Siemens EDA, UVM-AMS WG Chair
Abhijit Madhu Kumar, Cadence Design Systems

accellera SYSTEMS INITIATIVE

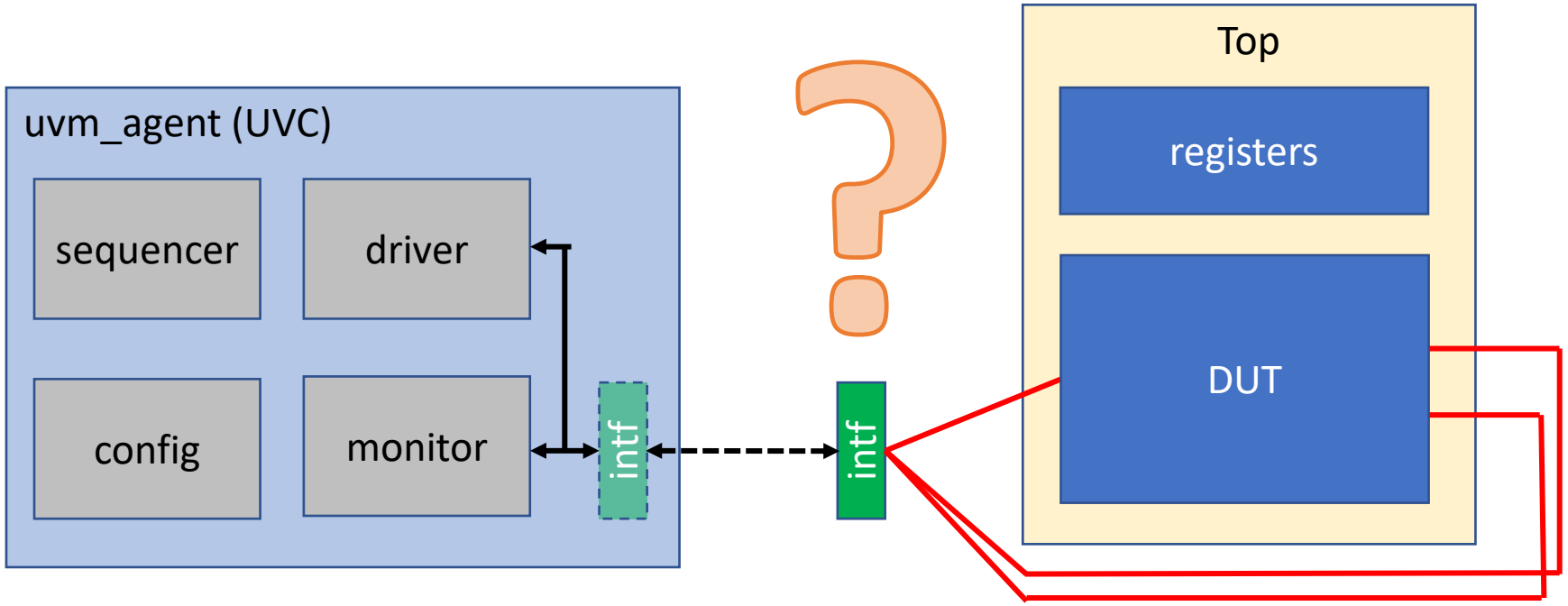
UVM-AMS WG Member Companies

- Cadence
- NXP
- Qualcomm
- Renesas
- Siemens
- Synopsys
- Texas Instruments

What Are We Trying to Do?

- Define a way to extend UVM to AMS/DMS
  - Modular, reusable testbench components
  - Sequence-based stimulus
  - Take advantage of UVM infrastructure as much as possible
- Reuse as much UVM as possible as DUT is refined from digital to AMS
  - Use extension/factory as much as possible
  - Support UVM architecture for DMS/AMS DUT from the start
- Define standard architecture for D/AMS interaction
  - Minimize traffic across boundary
  - Enable development of D/AMS VIP libraries & ecosystem

Classical UVM Example



Terminology

- Analog Mixed-Signal (AMS) simulation and verification refers to systems that can simulate/verify analog/mixed-signal designs as a co-simulation of digital + analog (electrical) solvers
- Digital Mixed-Signal (DMS) simulation and verification refers to systems that can simulate/verify analog/mixed-signal designs within a discrete event-driven solver as digital (logic) and real number models

Requirements

- Minimal changes to UVC to add AMS capabilities (driver, monitor, sequence item) that can be applied using set_type_override_by_type
- Define analog behavior based on a set of parameters defined in a sequence item and generate that analog signal using an analog resource (MS Bridge)
- Measure the properties of the analog signal, return them to a monitor, and package those properties into a sequence item
- Drive and monitor configurations, controlled by dedicated sequence items and support easy integration into multi-channel test sequences
- Controls can also be set by way of constraints for pre-run configurations.
- Collect/check coverage in the monitor based on property values returned from analog resource or add checkers in analog resource

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

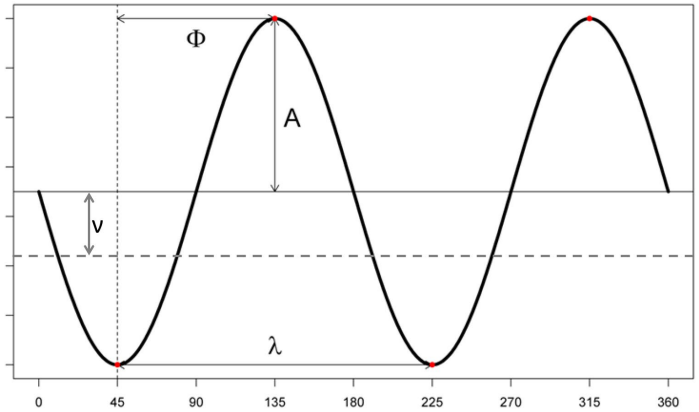
Now the Real Work Begins

Abhijit Madhu Kumar, Cadence Design Systems

accellera SYSTEMS INITIATIVE

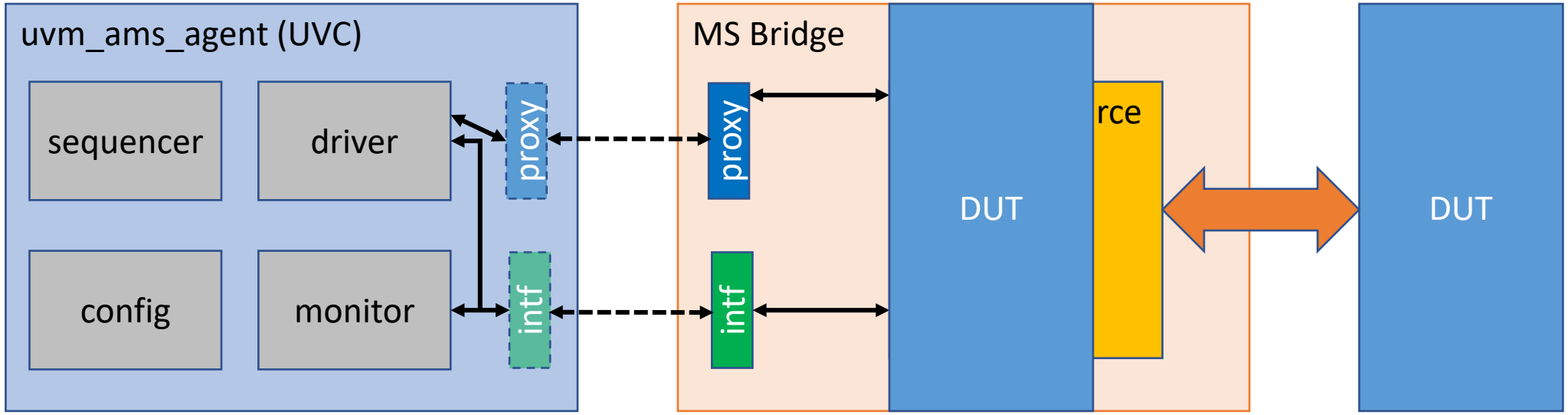
Generating/Driving Continuous Analog Signals

- An analog signal that is not simple DC or a slow changing signal, needs to be a periodic waveform like a sine wave or a sawtooth, or some composition of such sources.
- For example, a signal generator for a sine wave can be controlled by four control values determining the freq(λ), phase(Φ), amplitude(A), and DC bias(v) of the generated signal.



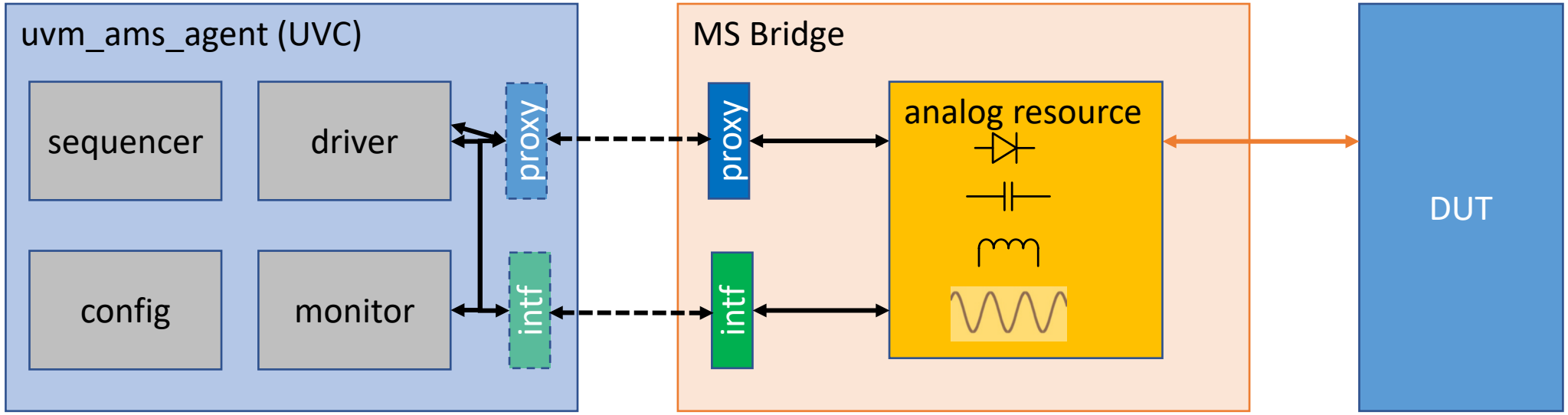
- The properties of the analog signal being driven are controlled by real values, generated by the sequencer
- A UVM sequence_item contains fields for all the control parameters.
- The driver converts the transaction to a setting for the signal generator.

Overall UVM-AMS Methodology



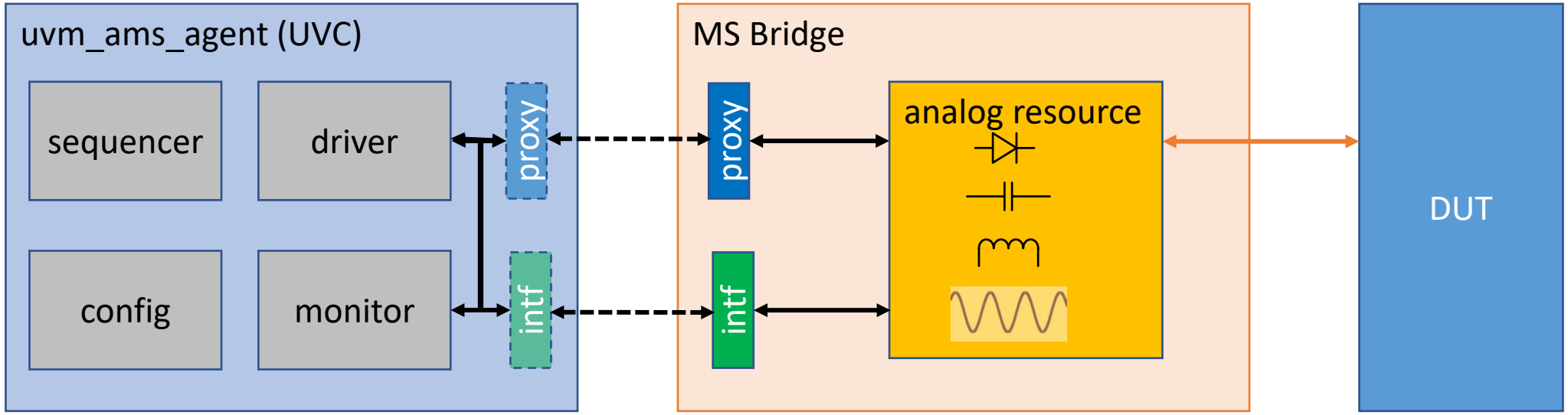
- MS Bridge is the proposed layer that sits between the UVC and the (A)MS DUT
- MS Bridge is a SV module that consists of a proxy API, SV interface, and an analog resource module
- The ‘proxy’ is an API that conveys analog attributes between the UVC and the MS Bridge
- The SV ‘intf’ passes digital/discrete signal values (logic, real, nettype/RNM) between UVC and MS Bridge
- Both ‘proxy’ and ‘intf’ can be used together or individually
- The analog resource (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-AMS Analog Resource



- MS testbench may require the behavior and presence of analog components that a typical UVM-RTL testbench could not include. These could be:
  - Capacitors, Resistors, Inductors, Diodes, current/voltage sources etc. Or a complex passive network for multiple DUT pins.
  - A piece of Verilog-AMS code
  - Such components will be used to model the analog behavior of PADs, lossy transmission lines, loads/impedances, or any other voltage/current conditioning required to accurately model the signals connecting to the ports of DUT
  - Those components can be placed inside the analog resource to be controlled by proxy.

UVM-AMS Analog Resource



- Proxy is an API used to interact with analog resource to perform the following
  - Push / pull electrical values such as voltage, current, component values.
  - Event generation
  - Arbitrary sampling of a continuous signal to update a variable in the proxy.
- The analog resource would have the same number of ports as the DUT for a one-to-one connectivity between the ports of analog resource and the DUT
- The API between the bridge and the analog resource must support Verilog-AMS language constructs to support all possible analog resource views (VAMS, SV, etc.)

Proxy “hook-up”

Proxy Template (API)

UVC package

```
virtual class pga_bridge_proxy;
...
  pure virtual function void config_wave(…);
...
endclass
```

Implement

Proxy instance in MS Bridge module

```
module pga_bridge(…);
…
pga_bridge_core #(…) core (…); // AMS model
…
class proxy extends pga_bridge_proxy;
  function void config_wave(input real ampl, bias, phase, freq);
    core.ampl_in  = ampl;
    core.bias_in  = bias;
    core.phase_in = phase;
    core.freq_in  = freq;
  endfunction
endclass

proxy bridge_proxy;
…
endmodule
```

Instance of analog resource

Passes values to analog resource to “program” waveform

UVM config setting

```
module top;
…
  osc_bridge osc_bridge(.clk_outp, .clk_outn, .clk_in);
…
  initial begin
    uvm_config_db#(pga_bridge_proxy)::set(null,"*freq_adpt*","bridge_proxy",top.osc_bridge.bridge_proxy);
    run_test();
  end
endmodule
```

Proxy ←→ Analog Resource

MS Bridge

Push

Pull

Monitored

```
class proxy … ;
  function void config_wave(…);
    core.ampl_in  = ampl;
    core.bias_in  = bias;
    core.phase_in = phase;
    core.freq_in  = freq;
  endfunction

  function void get_measures(...);
    ampl  = core.ampl_out;
    bias  = core.bias_out;
    phase = core.phase_out;
    freq  = core.freq_out;
  endfunction
  real min, max;
endclass
```

```
always_comb begin
  bridge_proxy.min = core.min_a;
  bridge_proxy.max = core.max_a;
end
```

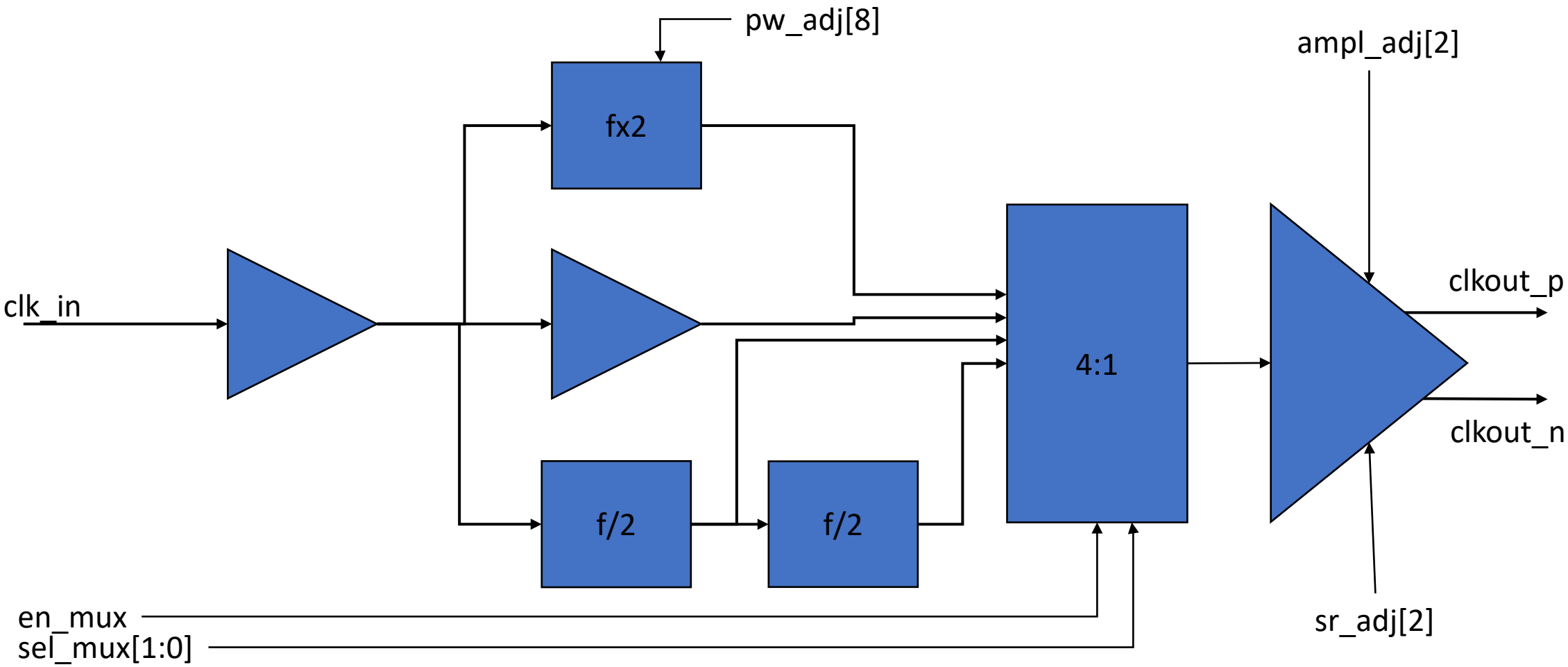
```
pga_bridge_core (...);
  ...
  real ampl_in;
  real bias_in;
  real phase_in;
  real freq_in;

  analog begin
    vsin = (ampl_in * sin(`M_TWO_PI * freq_in * $abstime);
    ...
  end

  real  ampl_out;
  real  bias_out;
  real  freq_out;
  real  phase_out;

  Vsig = V(sig);
  if (Vsig > max_a)
    max_a = Vsig;
  else if (Vsig < min_a)
    min_a = Vsig;
```

If target is different, it's seen as a D2A event

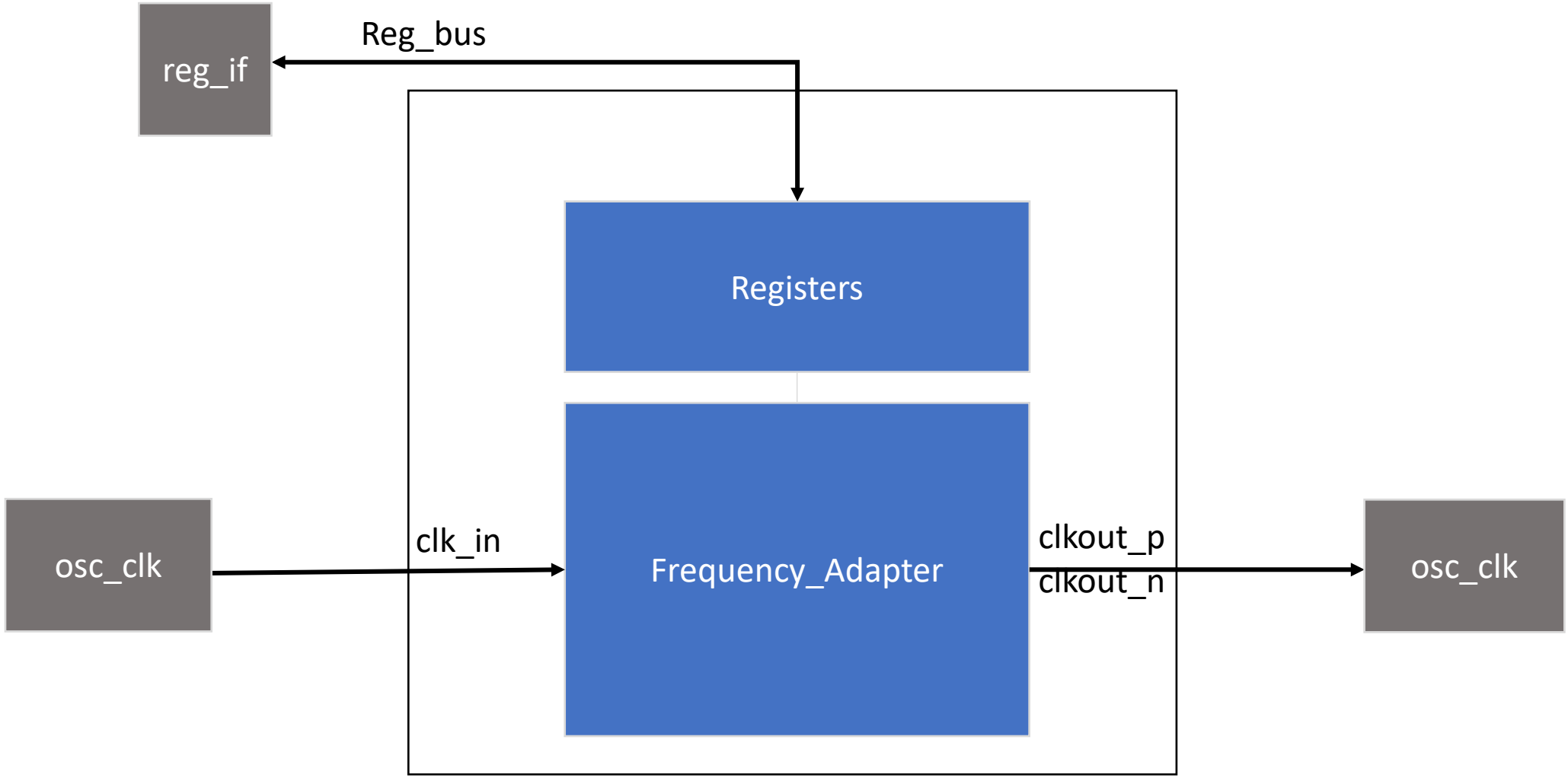
Interpolated value

Analog generates update

Frequency_Adapter DUT

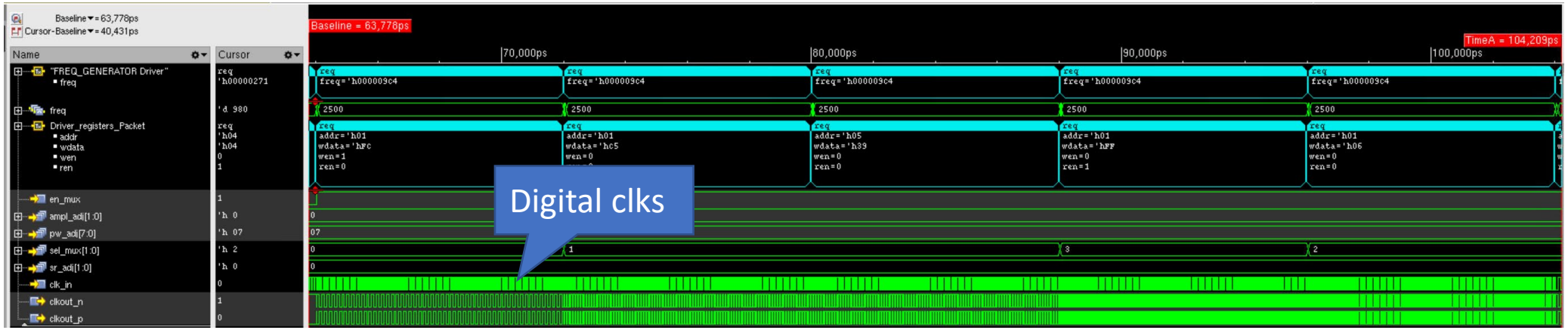


UVM TB – add analog capability

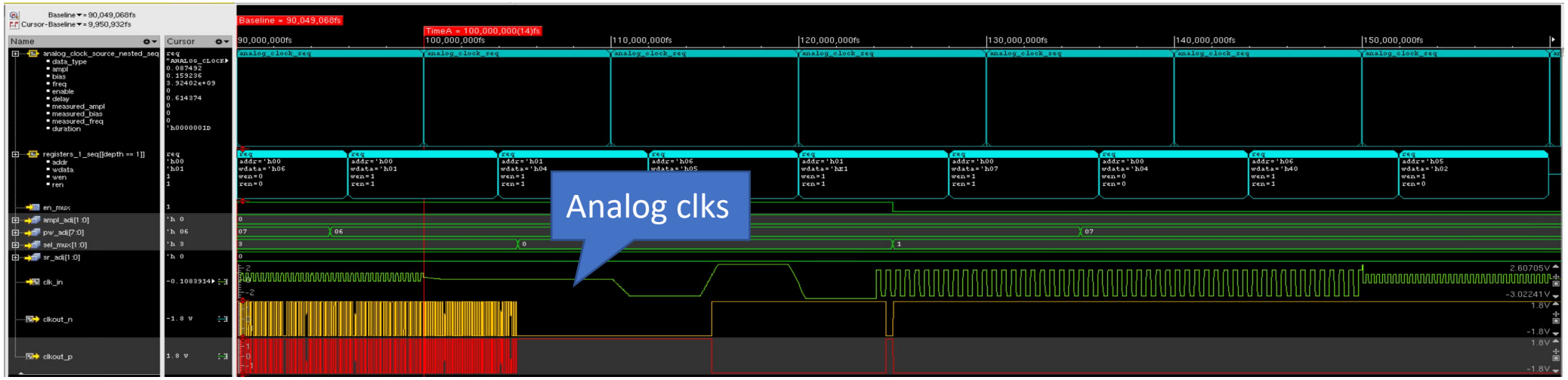


Freq_adapter Waveforms

Digital

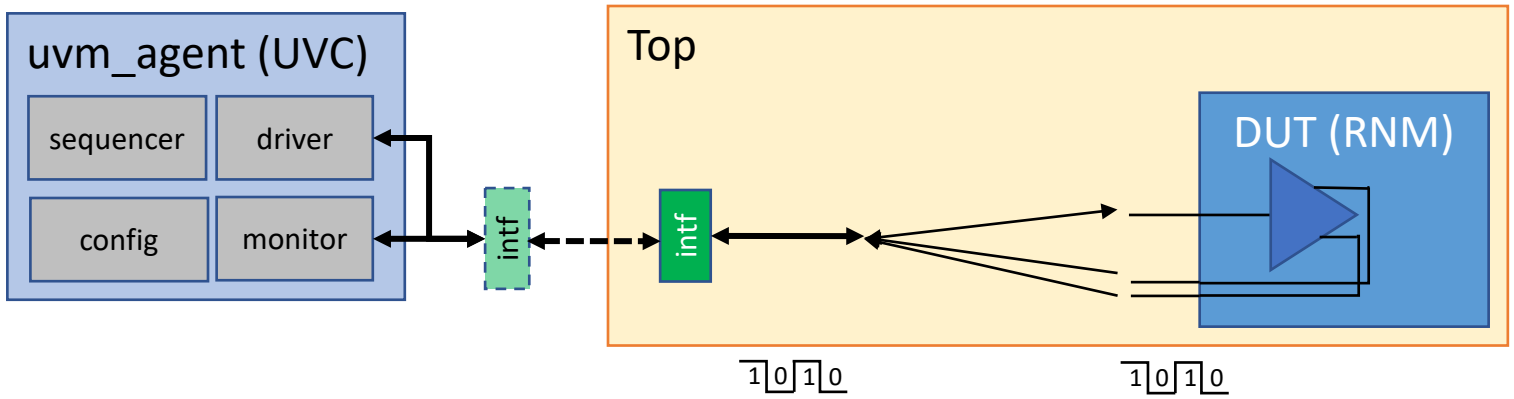


VAMS



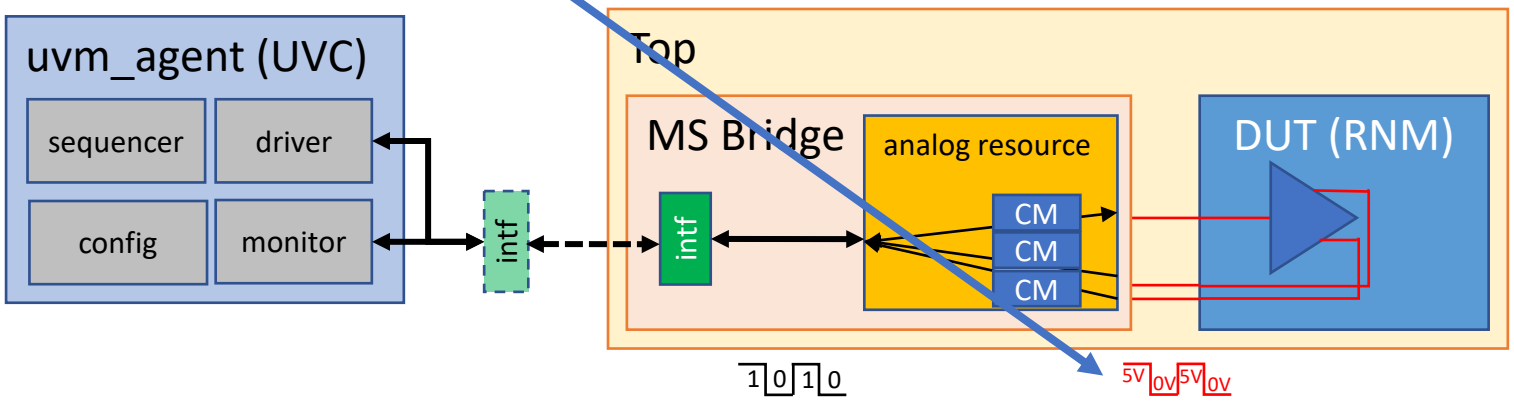
Model of Frequency Adapter Ports in SV

```
module freq_adapter (
  output logic CLKOUT_P,CLKOUT_N;  // differential output
  input  logic CLK_IN;             // clock input
  input  logic en_mux, [1:0] sel_mux; // register control
  input  logic [7:0] pw_adj, [1:0] sr_adj, ampl_adj;
);
```



Model of Frequency Adapter Ports in SV RNM

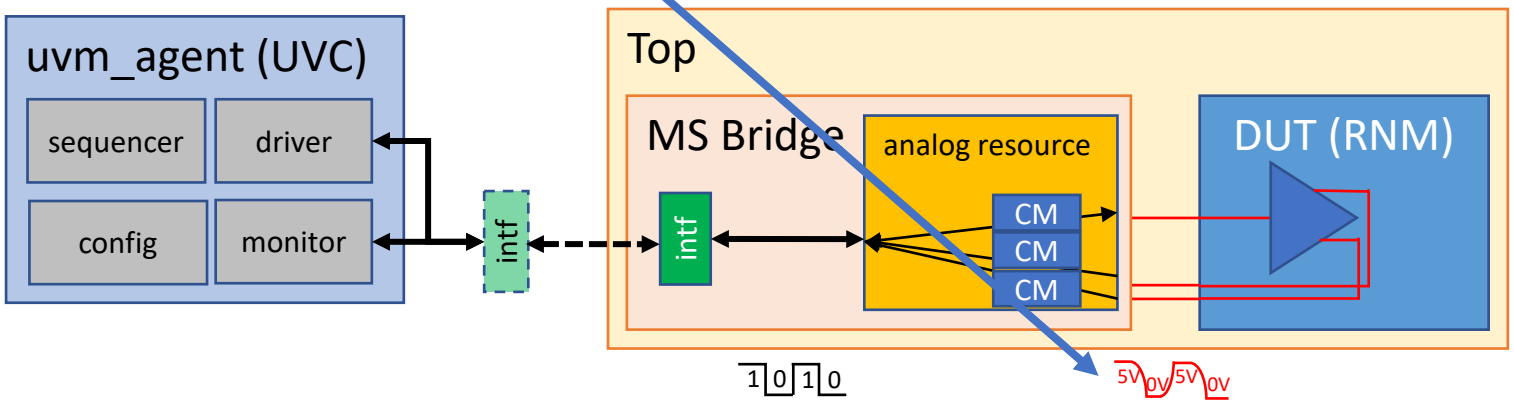
```
module freq_adapter import rnm_pkg::*; (
  output real_net CLKOUT_P,CLKOUT_N;  // differential output
  input  real_net CLK_IN;             // clock input
  input  logic en_mux, [1:0] sel_mux; // register control
  input  logic [7:0] pw_adj, [1:0] sr_adj, ampl_adj;
);
```



RNM uses event solver so just need to convert logic to real voltage

Model of Frequency Adapter Ports in VAMS

```
module freq_adapter (CLKOUT_P,CLKOUT_N,CLK_IN,en_mux,sel_mux,pw_adj,sr_adj,ampl_adj);
  output CLKOUT_P,CLKOUT_N; electrical CLKOUT_P,CLKOUT_N;    // differential output
  input CLK_IN; electrical CLK_IN;                           // clock input
  input wire [2:0] en_mux, [1:0] sel_mux;                    // register control
  input [7:0] pw_adj, [1:0] sr_adj, ampl_adj;                // digital control voltage
```

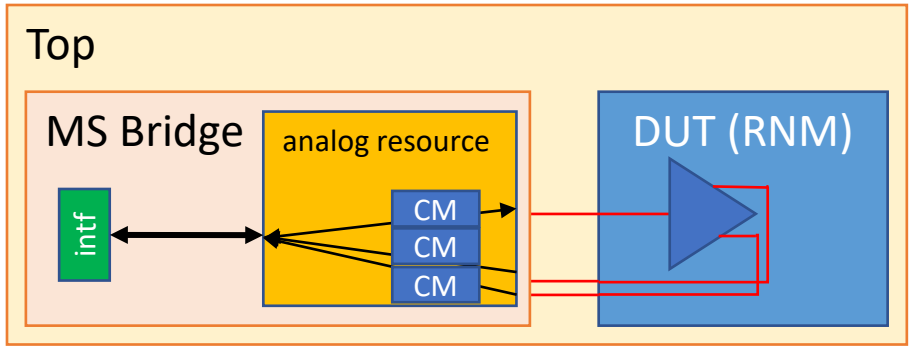


electrical uses analog solver that takes into account VIR

Analog Resource for SV-RNM/VAMS

Option 1

- Automatically inserted Connect Modules (CM) converts logic signal values to SV-RNM or electrical equivalents (depending on the DUT)
  - Simple to use but many non-standard requirements such as supply connection, DRS, etc.
  - No fine control on the analog resources ‘electrical’ interface
  - No changes required to UVM driver

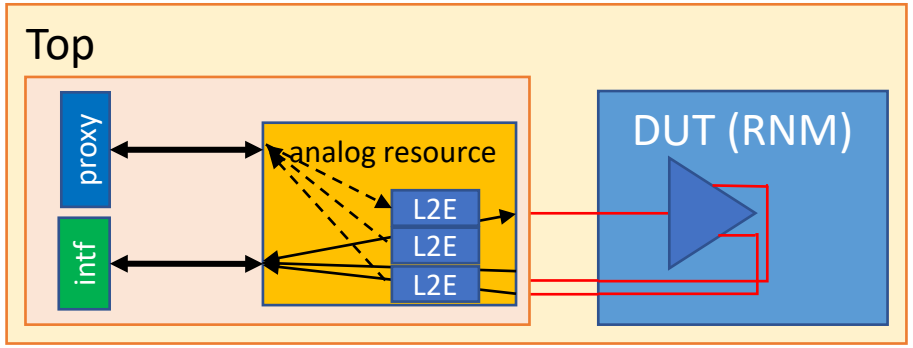


Not recommended where control over critical analog signals needed

Analog Resource for VAMS

Option 2

- User generated code for L2E converts logic signal values to electrical equivalents
  - Proxy used to pass supply value used by analog resource to determine voltage value of logic 1
  - Same UVC/MS Bridge with VAMS analog resource for electrical signals and RNM analog resource for RNM signals
  - Requires new functionality in UVM driver to access proxy and generate values



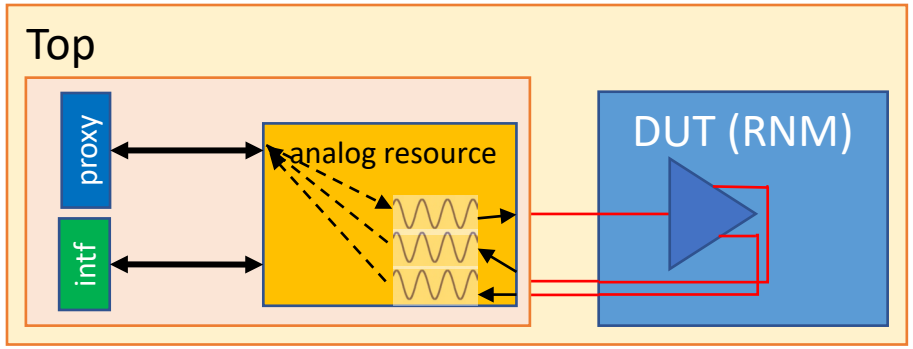
Recommended for protocol signals such as I^2C

Analog Resource for VAMS

Option 3

- Analog resource uses proxy attributes to generate analog signal algorithmically
  - Proxy used to pass attributes that define type and shape of analog signal
  - Same UVC/MS Bridge with VAMS analog resource for electrical signals and RNM analog resource for RNM signals
  - Requires override of UVM driver and sequence item to change functionality from driving signals to passing values through proxy

This is the option used for the demo



Recommended for continuously changing signals such as sine wave

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

Example Walk-through

UVM digital to UVM-AMS

accellera SYSTEMS INITIATIVE

Steps

- Create Bridge module
  - Contains Analog Resource, Interface, and Proxy (optional)
- Extend classes for Driver, Monitor, and Sequence Item
  - Use set_type_override_by_type to use extended classes
- Create Proxy class if needed

analog_clk_bridge

```
3  module osc_bridge ( input osc_clk, output osc_clk_p, osc_clk_n
4    import osc_pkg::*;
5
6    class proxy extends osc_bridge_proxy;
7      function void config_wave(input real ampl, bias, freq, ena
8        core.ampl_in  = ampl;
9        core.bias_in  = bias;
10       core.freq_in  = freq;
11       core.enable   = enable;
12     endfunction
13     //Signals to send to core sampler
14     real   delay_in;
15     int    duration_in;
16     bit    sampling_do;
17     //Measurements to send up reported values to monitor
18     real   sampling_done;
19     real   ampl_out;
20     real   bias_out;
21     real   freq_out;
22   endclass
23
24   proxy bridge_proxy = new();
```

```
26   always @(bridge_proxy.delay_in, bridge_proxy.duration_in,
27     core.delay_in    = bridge_proxy.delay_in;
28     core.duration_in = bridge_proxy.duration_in;
29     core.sampling_do = bridge_proxy.sampling_do;
30   end
31
32   always_comb begin
33     bridge_proxy.sampling_done = core.sampling_done;
34     bridge_proxy.ampl_out = core.ampl_out;
35     bridge_proxy.bias_out = core.bias_out;
36     bridge_proxy.freq_out = core.freq_out;
37   end
38
39   osc_bridge_core #(.diff_sel(diff_sel)) core (
40     .osc_clk(osc_clk),
41     .osc_clk_p(osc_clk_p),
42     .osc_clk_n(osc_clk_n)
43     );
44
45 endmodule
```

analog_clk_driver

UVM

```
class osc_driver extends uvm_driver #(osc_transaction);
  // The virtual interface used to drive and view HDL signals.
  virtual interface osc_if vif;

  // period of the generated clock
  real period;

  // component macro
  `uvm_component_utils_begin(osc_driver)
    `uvm_field_real(period, UVM_ALL_ON)
  `uvm_component_utils_end

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

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

  function void connect_phase(uvm_phase phase);
    if (!uvm_config_db#(virtual osc_if)::get(this,"","vif", vif))
      `uvm_error("NOVIF",{"virtual interface must be set for: ",get_fu
  endfunction: connect_phase

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

UVM-AMS

```
class osc_ms_driver extends osc_driver;
  protected osc_bridge_proxy bridge_proxy;

  osc_ms_transaction ms_req;

  `uvm_component_utils(osc_ms_driver)

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

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    if(!uvm_config_db#(osc_bridge_proxy)::get(this,"","bridge_proxy",bridge_proxy))
      `uvm_error(get_type_name(),"bridge proxy not configured");
  endfunction

  task get_and_drive();
    forever begin
      seq_item_port.get_next_item(req);
      $cast(ms_req,req);
      drive_transaction(ms_req);
      seq_item_port.item_done();
      fork
        #(20*1ns); //Time for transaction
        begin : sample_thread
          #(1ns) bridge_proxy.sampling_do = 1;
          #(1ns) bridge_proxy.sampling_do = 0;
        end
      join
    end
  endtask : get_and_drive
```

analog_clk_trans

UVM

```
 7 class osc_transaction extends uvm_sequence_item;
 8   rand real freq; // frequency of input clock
 9
10   `uvm_object_utils_begin(osc_transaction)
11     `uvm_field_real(freq, UVM_ALL_ON)
12   `uvm_object_utils_end
13
14   function new (string name = "osc_transaction");
15     super.new(name);
16   endfunction : new
17
18   constraint freq_c { freq inside {625, 1250, 2500}; }
19
20 endclass : osc_transaction
```

UVM-AMS

```
38 class osc_ms_transaction extends osc_transaction;
39
40   // Bridge Proxy fields
41   rand real ampl;
42   //rand real freq; // already exists in base class
43   rand real bias;
44   rand real period;
45
46   rand bit  enable;
47   rand real delay;    //Delay in ns
48   rand int  duration;
49
50   real measured_ampl;
51   real measured_bias;
52   real measured_freq;
53
54   constraint default_drive_trans_c {
55     freq > 5e8;
56     freq < 1e9;
57     ampl > 0;
58     ampl < 1.65;
59     bias inside {[-0.5:0.5]};
60     enable dist { 1'b0 := 1 , 1'b1 := 5 };
61   }
62   constraint default_measurement_trans_c {
63     duration > 20;
64     duration < 32;
65     delay > 0.0;
66     delay < 1.0;
67   }
```

analog_clk_tb

UVM

```
class freq_adpt_tb extends uvm_env;
  `uvm_component_utils(freq_adpt_tb)

  registers_env registers;
  osc_env freq_generator;
  osc_env freq_detector;
  freq_adpt_scoreboard freq_adpt_sb;

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

  function void build_phase(uvm_phase phase);
    uvm_config_db#(virtual osc_if)::set(this,"freq_generator*","vif", top.g
    uvm_config_db#(virtual osc_if)::set(this,"freq_detector*", "vif", top.d
    uvm_config_db#(virtual registers_if)::set(this,"registers.reg_agent.*",
    uvm_config_db#(virtual registers_if)::set(this,"freq_adpt_sb",


    super.build_phase(phase);

    freq_generator = osc_env::type_id::create("freq_generator",  this);
    freq_detector  = osc_env::type_id::create("freq_detector",   this);
    registers      = registers_env::type_id::create("registers", this);
    freq_adpt_sb   = freq_adpt_scoreboard::type_id::create("freq_adpt_sb",   this);

  endfunction : build_phase

  function void connect_phase(uvm_phase phase);
    registers.reg_agent.monitor.item_collected_port.connect(freq_adpt_sb.sb_registers_in);
    freq_generator.agent.monitor.item_collected_port.connect(freq_adpt_sb.sb_osc_gen);
    freq_detector.agent.monitor.item_collected_port.connect(freq_adpt_sb.sb_osc_det);
  endfunction : connect_phase

endclass : freq_adpt_tb
```

UVM-AMS

```
class freq_adpt_ms_tb extends freq_adpt_tb;
  `uvm_component_utils(freq_adpt_ms_tb)

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

  function void build_phase(uvm_phase phase);
    uvm_config_db #(osc_bridge_proxy)::set(this,"freq_gen.agent.*","proxy", top.gen_bridge.bridge_proxy);
    uvm_config_db #(osc_bridge_proxy)::set(this,"freq_det.agent.*","proxy", top.det_bridge.bridge_proxy);
    // override driver, monitor, and scoreboard with UVM-AMS versions
    set_type_override_by_type(osc_transaction::get_type(),osc_ms_transaction::get_type());
    set_type_override_by_type(osc_driver::get_type(),     osc_ms_driver::get_type());
    set_type_override_by_type(osc_monitor::get_type(),     osc_ms_monitor::get_type());
    super.build_phase(phase);
  endfunction

endclass : freq_adpt_ms_tb
```

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Demo

accellera SYSTEMS INITIATIVE

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

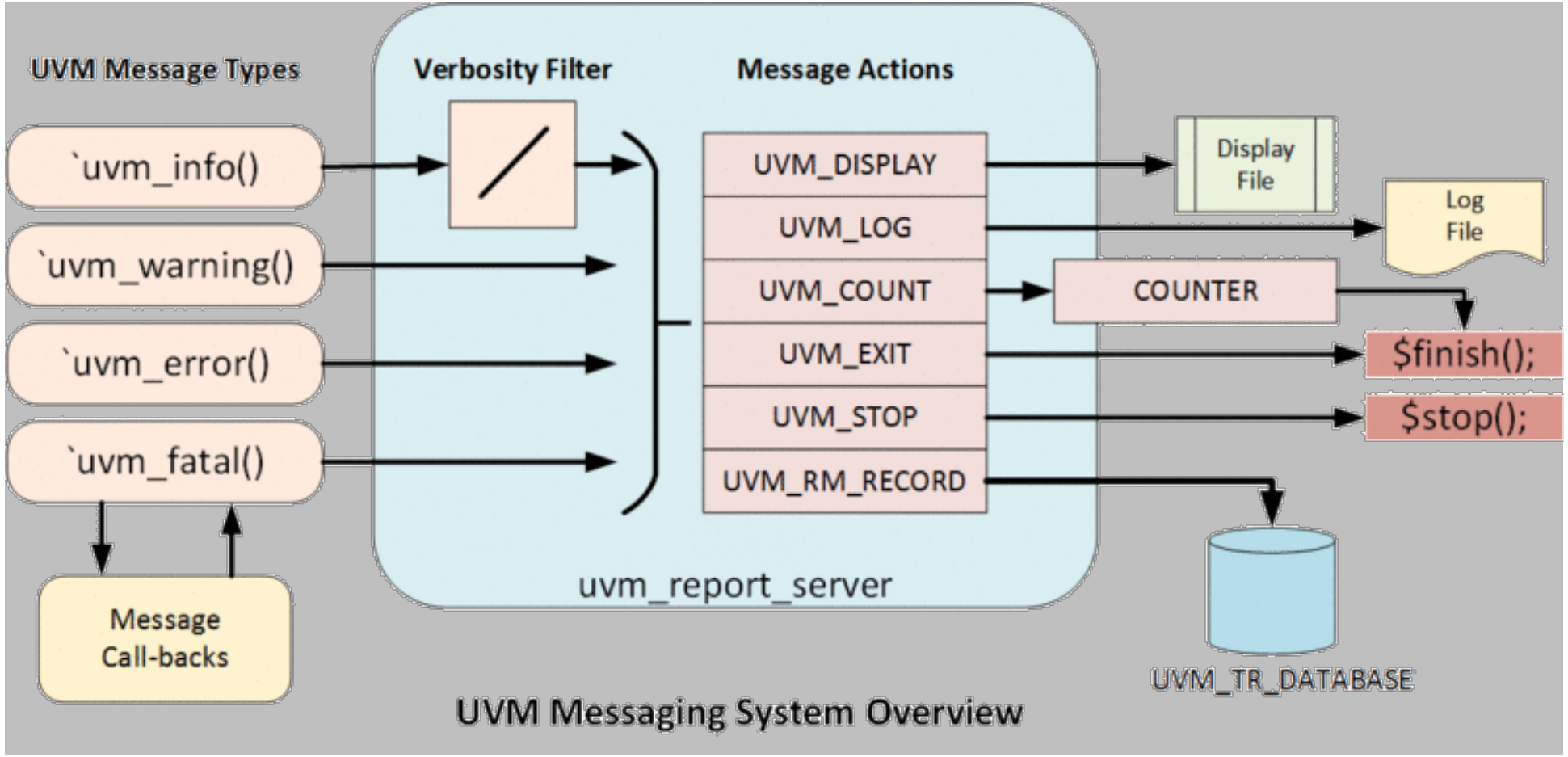
UVM Messaging

accellera SYSTEMS INITIATIVE

Messages for Debug and Error Reporting

- Debugging activity inside a large environment with many UVCs is critical.
- Need to report:
  - Errors
  - Debug
  - Progress
- Messages need to be categorized via severity:
  - Fatal, Error, Warning, Info
- Need to link actions with messages
  - Stop simulation on fatal or after four errors
  - Summarize number of messages reported
- Need a different mechanism than simulator messages to avoid filtering effects

UVM Messaging System



UVM Messaging from Analog Resource

- UVM Reporting macros not supported in Verilog-AMS modules.
- Take advantage of up-scoping to provide solution. (1364-2001 LRM)
- `include "uvm_ams.vamsh“ in Verilog-AMS file (analog resource)
  - localparams to define UVM Verbosity levels as integers to match UVM enum
- `include "uvm_ams.svh“ in SV file (MS Bridge)
  - Void functions that wrap `uvm_*() reporting macros into functions of the same name
- Within a digital block of a Verilog-AMS file users call; uvm_[info|warning|error|fatal](…)
  - Up scoping means it find the function in the MS Bridge file
- Within analog block, many solutions so here is one (calling of digital functions not allowed)
  - Set string value and toggle integer
  - Use absdelta to trigger on toggle and read string to call up-scoping function

UVM Message – Analog block

VAMS

```
localparam string uvm_path = $sformat(uvm_path,”%m”);
localparam string message = $sformat("The Current is above the threshold @ %eA",I_PLUS);
uvm_info(P__TYPE,message,UVM_MEDIUM,uvm_path);
```

SV Bridge

```
function void uvm_info(string id, string message, int verbosity_level, string uvm_path);
`uvm_info_context(id,message,uvm_verbosity'(verbosity_level),uvm_root::get().find(uvm_path))
endfunction: uvm_info
```

- Hold UVM component hierarchy path string in proxy class via get_full_name()
- Use *_context reporting macros to direct message to relevant component

```
UVM_INFO ../../include/uvm_ams.svh(26) @ 52001.098068ns: uvm_test_top.env.v_agent [i_bridge]
The Current is above the threshold @ 1.178812e+00A
```

Conclusions

- There is a need for more advanced, standard methodologies for scalable, reusable and metric-driven mixed-signal (AMS/DMS) verification
- The UVM-AMS proposal addresses the gaps in current verification methodology standards
- Extend UVM class-based approach to seamlessly support the module-based approach (MS Bridge) needed for mixed-signal verification
  - Targeting analog/mixed-signal contents (RNM, electrical/SPICE)
  - Application and extension of existing UVM concepts and components
    - Sequencer, Driver, Monitor
    - MS Bridge / Analog resources
    - UVM Messaging System

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

Questions?

accellera
SYSTEMS INITIATIVE