2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Migrating from UVM to UVM-MS

Accellera UVM-AMS Working Group

Tim Pylant, Cadence Design Systems, UVM-AMS WG Vice-Chair

accellera SYSTEMS INITIATIVE

UVM-AMS WG Member Companies

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

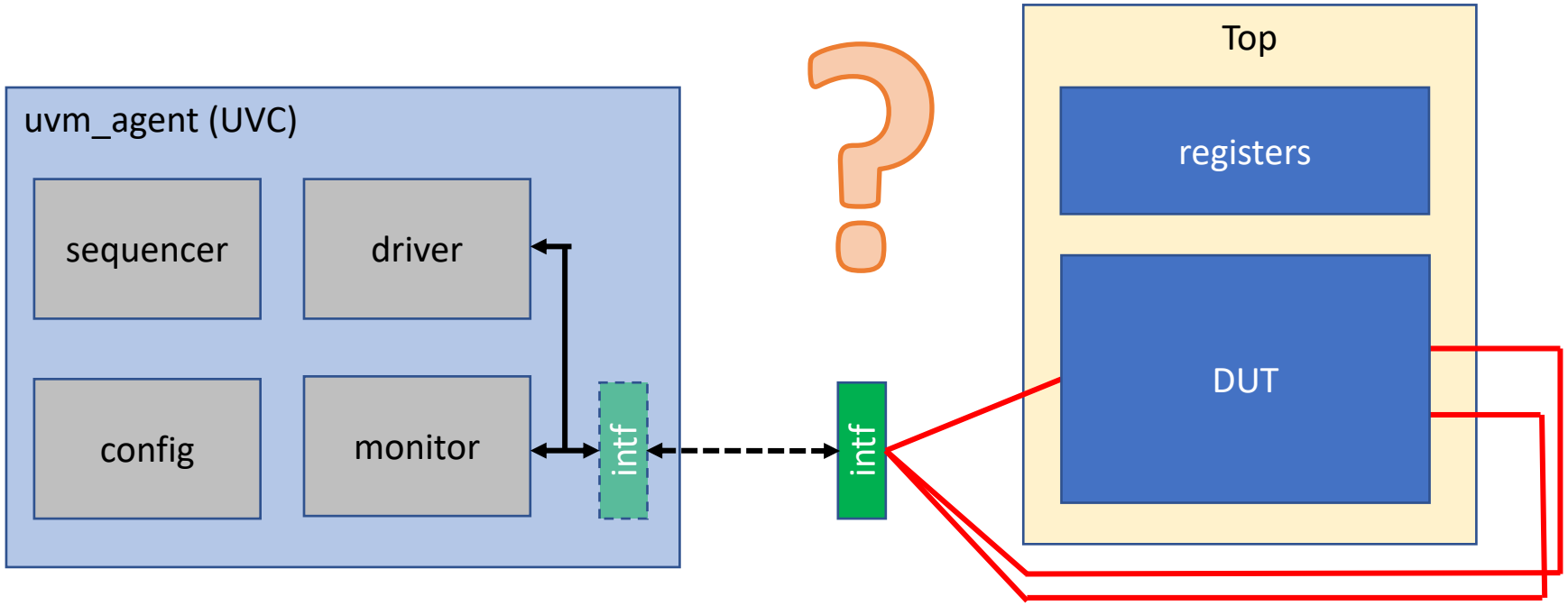
Why UVM-AMS

- Same reason as UVM – explosion of verification needs
  - Verifying analog functionality/connectivity under large set of digital configurations
  - Digital control system transitions interacting with analog functions
  - Dynamic control between analog & digital circuits under wide range of conditions
  - Finding problems with A/D interaction in unexpected corner cases
- Standard methodology
  - Plug & play reuse of existing UVM components
  - Rich debug and messaging scheme integrated with simulator

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 agent to add MS 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

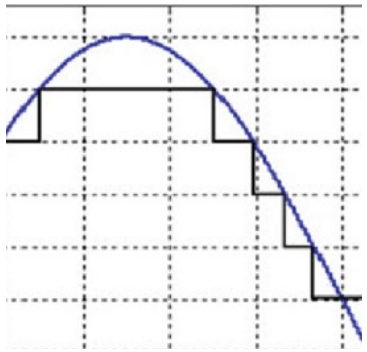
2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

What is needed to move from UVM to UVM-MS

accellera SYSTEMS INITIATIVE

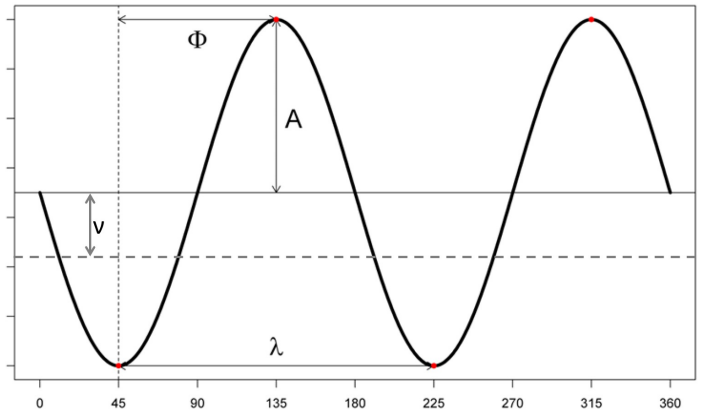
Generating/Driving Discrete Analog Signals

- 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
- Classical RNM would drive real numbers from UVC sequence/driver within the agent
- **In AMS this would generate too many D2A events or not give enough finesse to the signal**



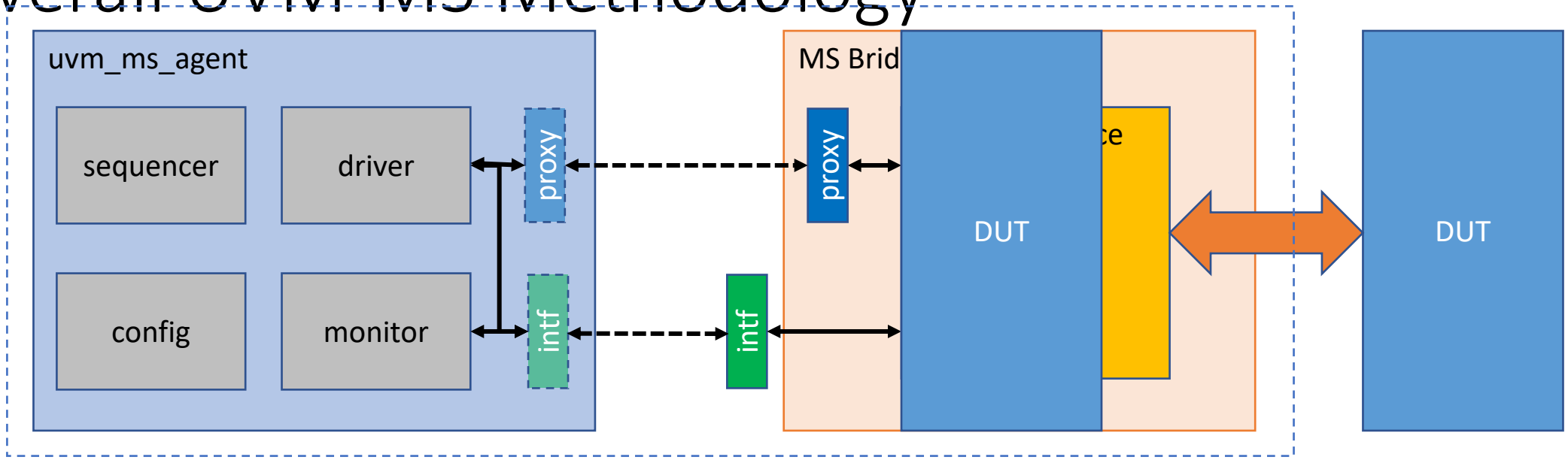
Generating/Driving Continuous Analog Signals

- A signal generator for a continuously changing signal can be controlled by four properties determining the freq(1/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 passes the fields of the sequence_item to the controls for the signal generator

Overall UVM-MS Methodology



- MS Bridge is the proposed layer that sits between the agent and the (A)MS DUT and consists of a proxy API, SV interface, and an analog resource module
- The ‘proxy’ is an API that conveys analog attributes between the agent and the MS Bridge
- The SV ‘intf’ passes digital/discrete signal values (logic, real, nettype/RNM) between agent and MS Bridge – can be left in top or moved to Bridge

Verilog-AMS Simulator DC OP

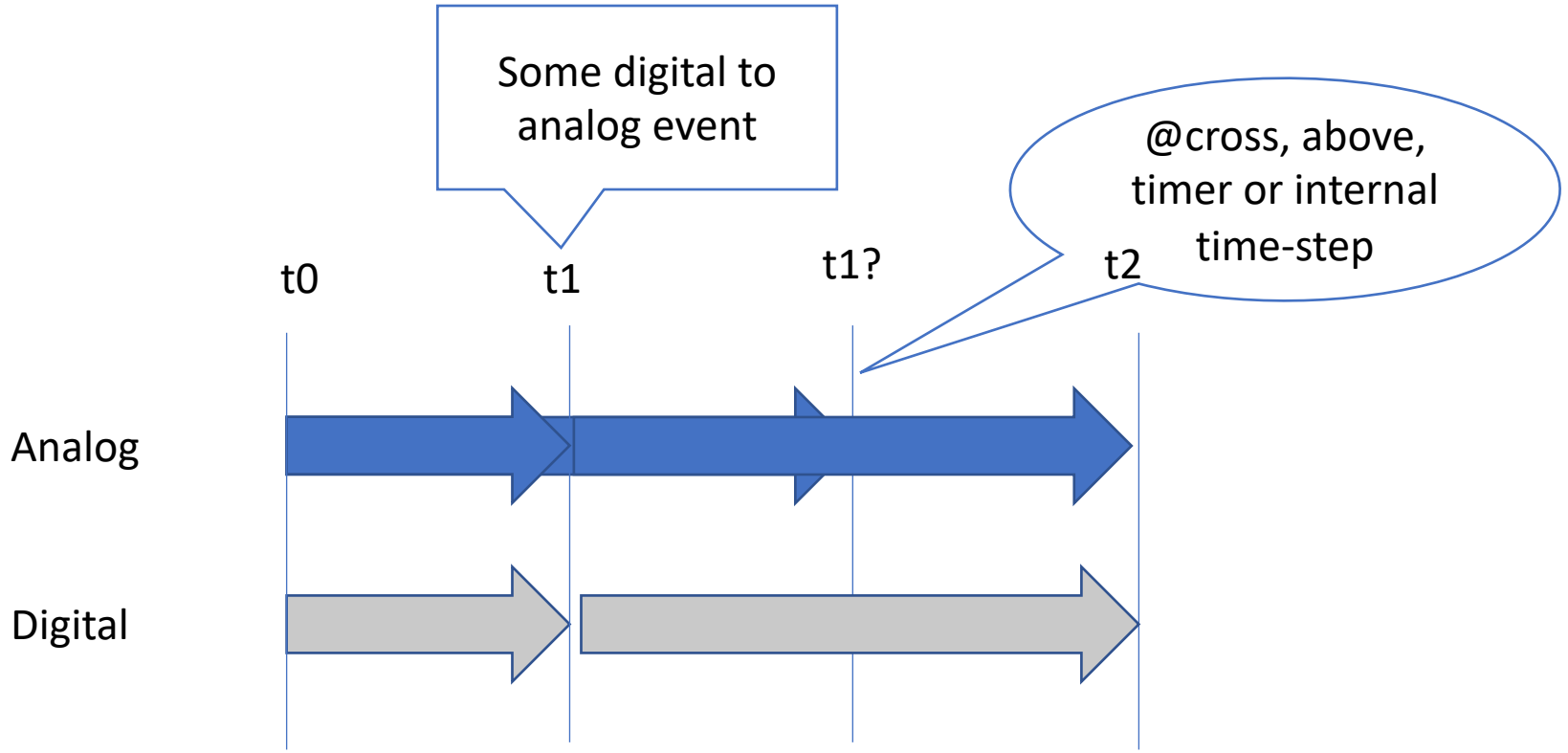
- DC Op – Steady State operating point of all the nodes/branch currents
- Understanding of UVM-MS DC OP is important
  - Need to make sure initial conditions (caps, supplies, timesteps) start with valid values for proper convergence
  - Enable UVM DUT configuration before analog circuit initialization (DC Op).
    - E.g. make a cap open for a particular test before DC op
  - Using #0 is not good practice as it shows poor coding and understanding of the simulator(s) scheduler
- Must raise a UVM objection before DC OP otherwise the simulation finishes

```
virtual task my_ams_test::run_phase(uvm_phase phase);
  …
  phase.raise_objection(this);
  if ($realtime <= 0.0) #1step;
  `uvm_info("TEST", "AMS DC-OP finished", UVM_MEDIUM)
  my_seq.start(my_seqr); // Launch sequence(s)
  …
  phase.drop_objection(this); // Test termination
endtask: my_ams_test
```

Ensures time is consumed

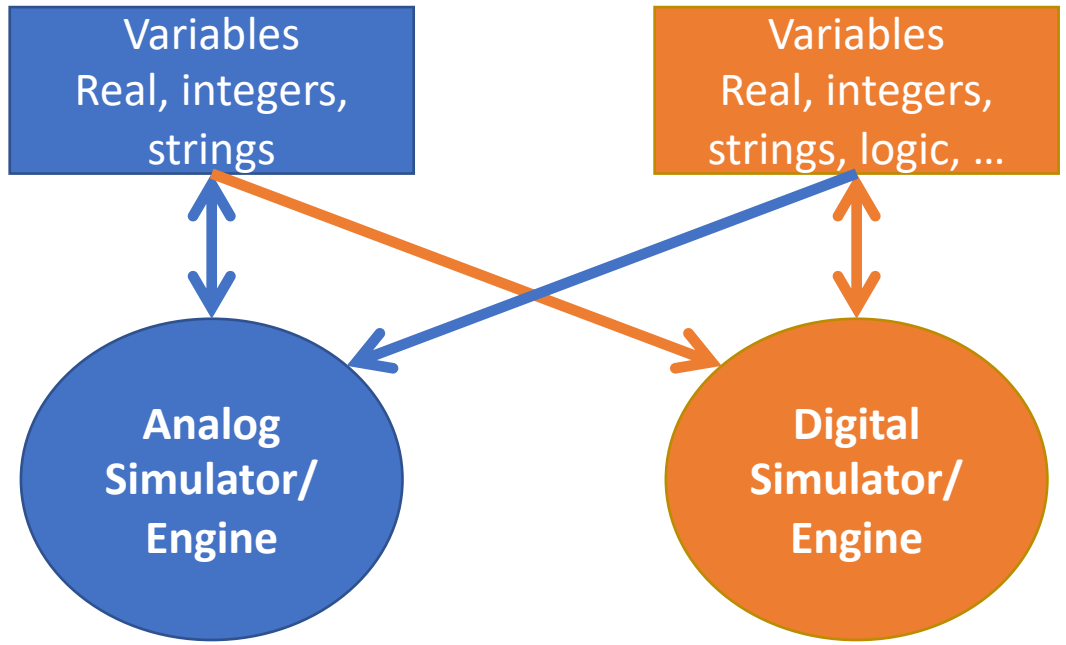
Verilog-AMS Simulator Scheduling - Transient

- Analog engine always leads
- Digital to analog events cause matrix re-evaluation and timestep backtrack
- Most simulators see any digital var in the analog block as a D2A to monitor



Verilog-AMS Best Practices

- Variables are ‘owned’ by one engine but can be read by another
- AMS can’t access digital variables that are dynamic (everything in the matrix is fixed at time 0)
- Generally, avoid ‘string’ datatypes in Verilog-AMS as support is inconsistent and the LRM is not clear
- OOMR to analog-owned variables not allowed – they are not part of the analog matrix



Proxy “hook-up”

UVC package

```
class osc_bridge_proxy extends uvm_ms_proxy;
...
  pure virtual function void config_wave(…);
...
  real freq_out;
...
endclass
```

Implement

UVM config setting

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

Proxy instance in MS Bridge module

```
module osc_bridge(...);
...
osc_bridge_core #(...) core (...); // AMS model
...
class proxy extends osc_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 __uvm_ms_proxy = new();
...
always_comb
  __uvm_ms_proxy.freq_out = core.freq_out;
...
endmodule
```

Instance of analog resource

Passes values to analog resource to “program” waveform

Passes values to agent component to ”monitor” waveform

Proxy ←→ Analog Resource

MS Bridge

Push

Pull

Monitored

```
class osc_bridge_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; //from base class
endclass
```

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

```
osc_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

UVM Phasing Requirements for UVM-MS

- Analog resources will have parameters and UVM should have a means to read/modify/write them before simulation consumes time
- Implement methods getParameters() / setParameters() in proxy
- Use existing UVM phases to guarantee read/modify/write order

UVM Phase	What should happen for AMS resources
build	
connect	Read parameters values from 'SV+VAMS' module (Instrument/Passive) into the agent's configuration.
end_of_elaboration	Modify agents parameters based on test requirements
start_of_simulation	Apply agents parameters to 'SV+VAMS' module (Instrument/Passive)

run	Must consume some time to allow DC OP to happen before agents drive sequence items so that synchronization system works. Recommend
	`task run_phase() ;`
	`  if($realtime <= 0.0) #1step; // cause a DC OP to occur`

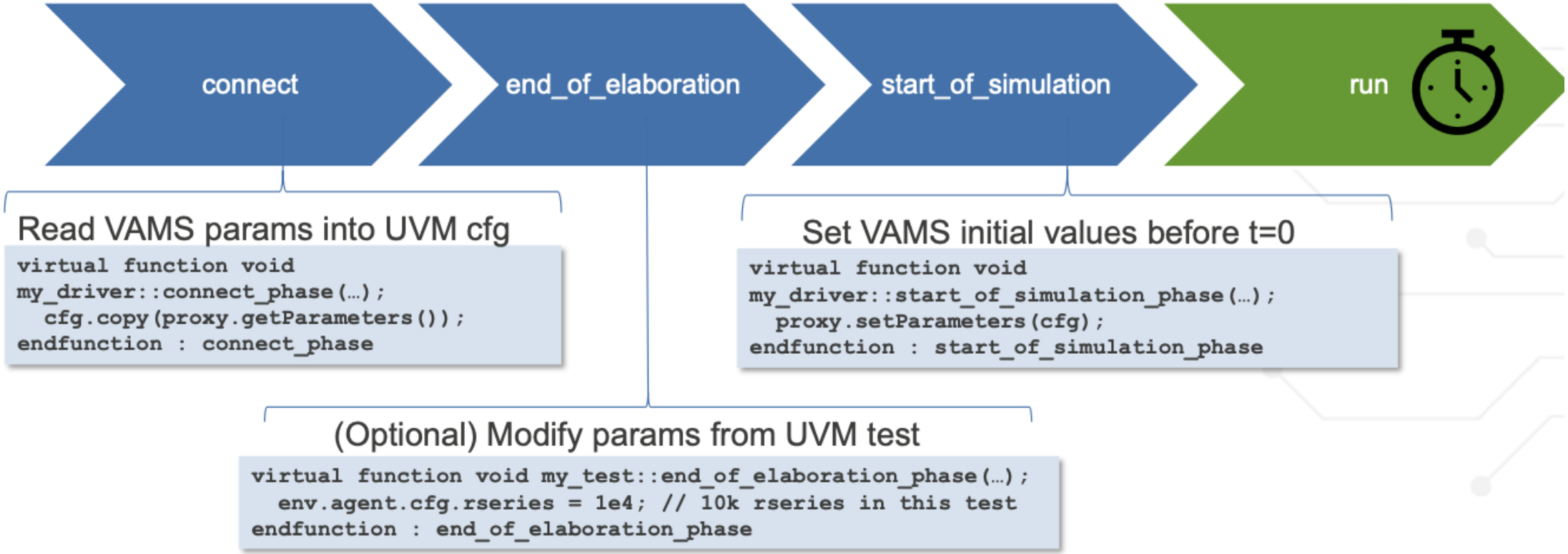
Analog Resource Configuration

- Analog components tend to be placed with initial values as parameters. e.g. a decoupling cap on an LDO output
- Allow the MS Bridge to have parameters that are copied from UVM configuration in connect_phase
- Test cases can override the configuration, which is then set in the analog resource in start_of_simulation_phase

```
class osc_bridge_proxy;
  function res_config getParameters();
    res_config cfg = new();
    cfg.res_val = i_core.rseries_val;
    ...
    return(cfg);
  endfunction
  function void setParameters(res_config cfg);
    i_core.rseries_val = cfg.res_val;
    ...
  endfunction
  ...
```

```
osc_bridge_core (...);
  ...
  parameter res_val = 200;
  ...
  // Initial values set from parameter,
  // then set by setParameter in proxy
  real rseries_val = res_val;
  ...
```

UVM-MS Phasing



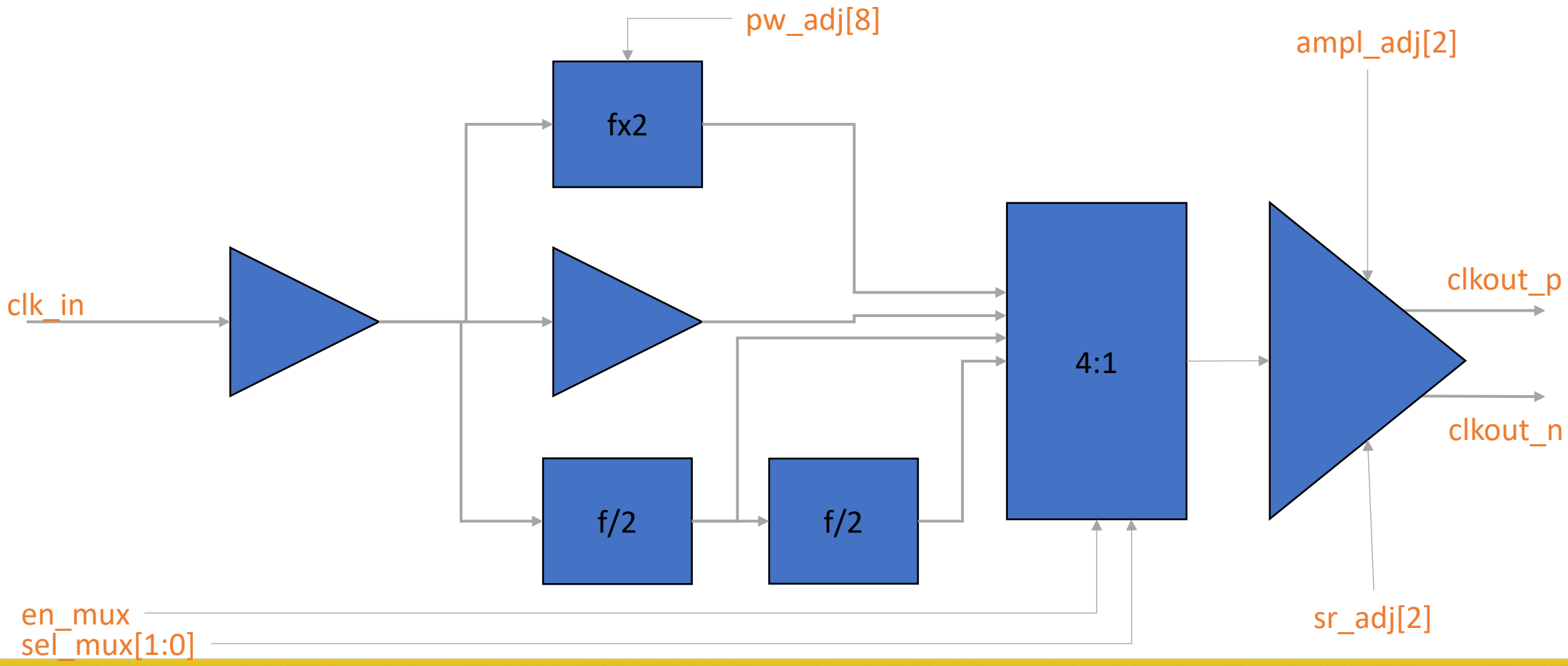
2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Example Walk-through

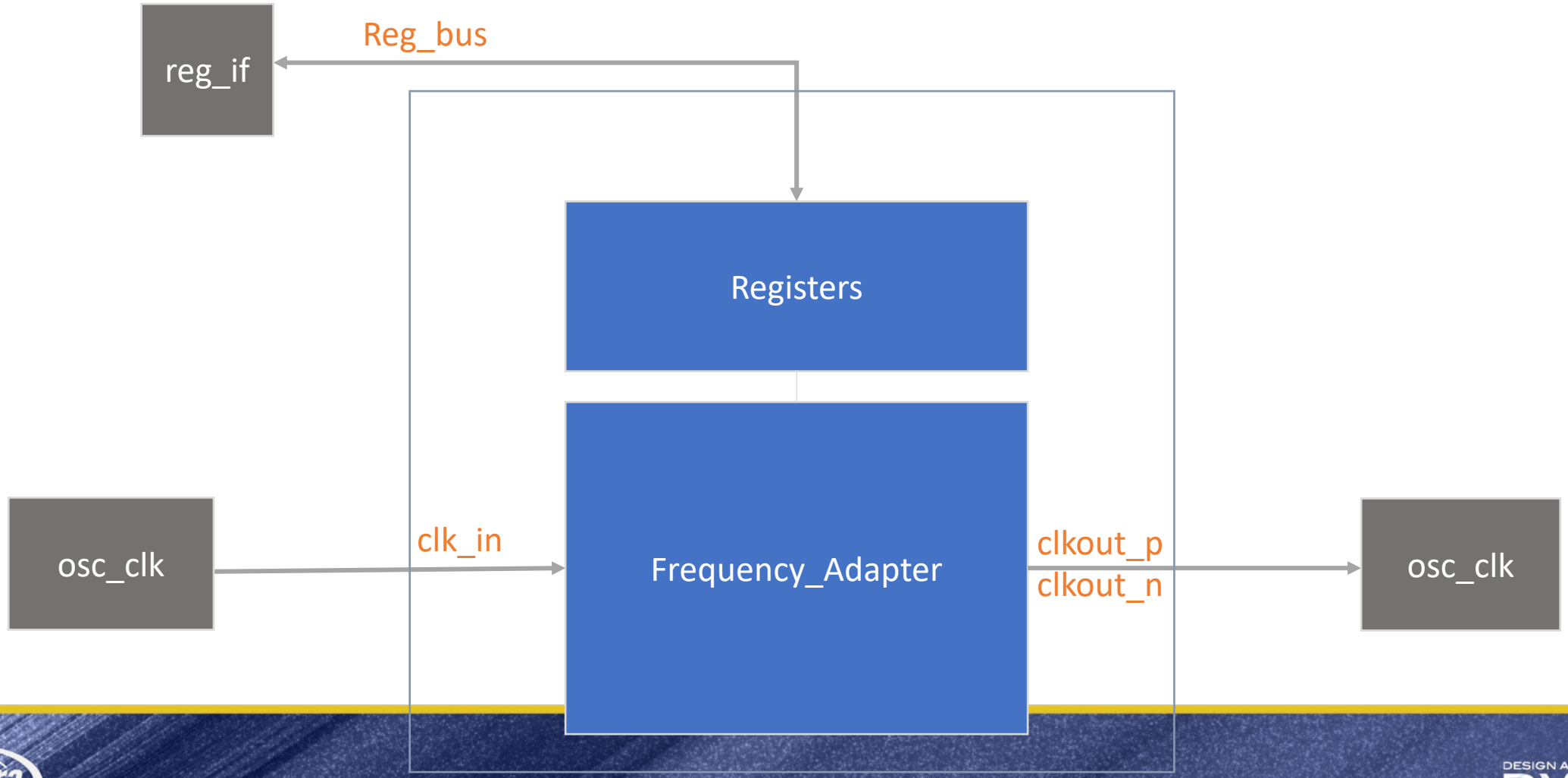
UVM digital to UVM-MS

accellera SYSTEMS INITIATIVE

Frequency_Adapter DUT

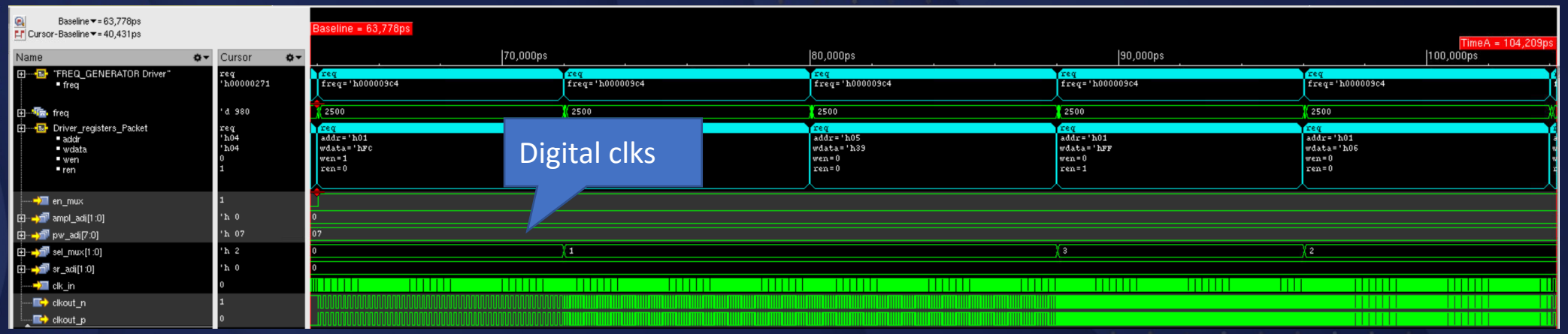


UVM TB – add analog capability

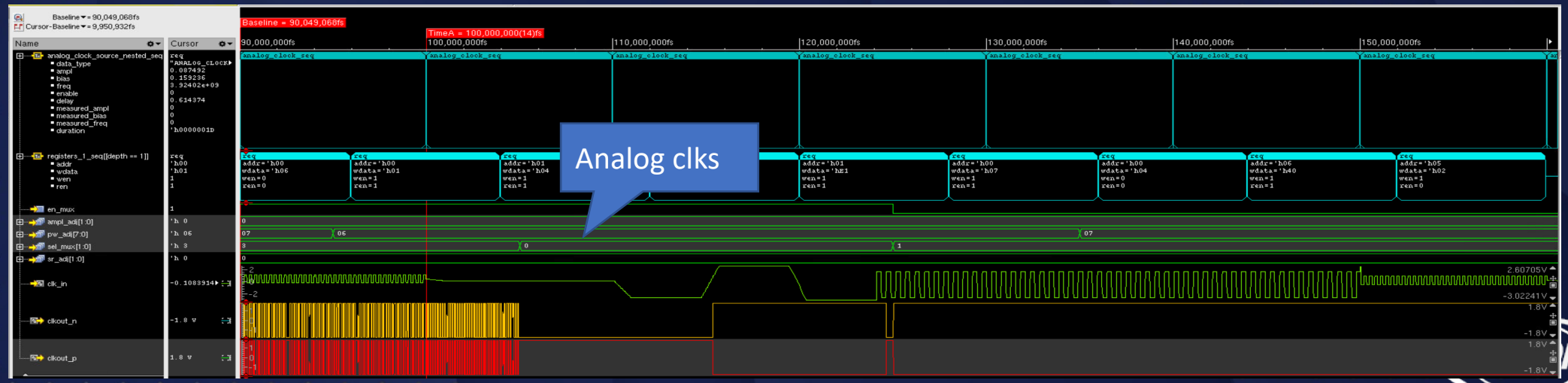


Freq_adapter Waveforms

Digital

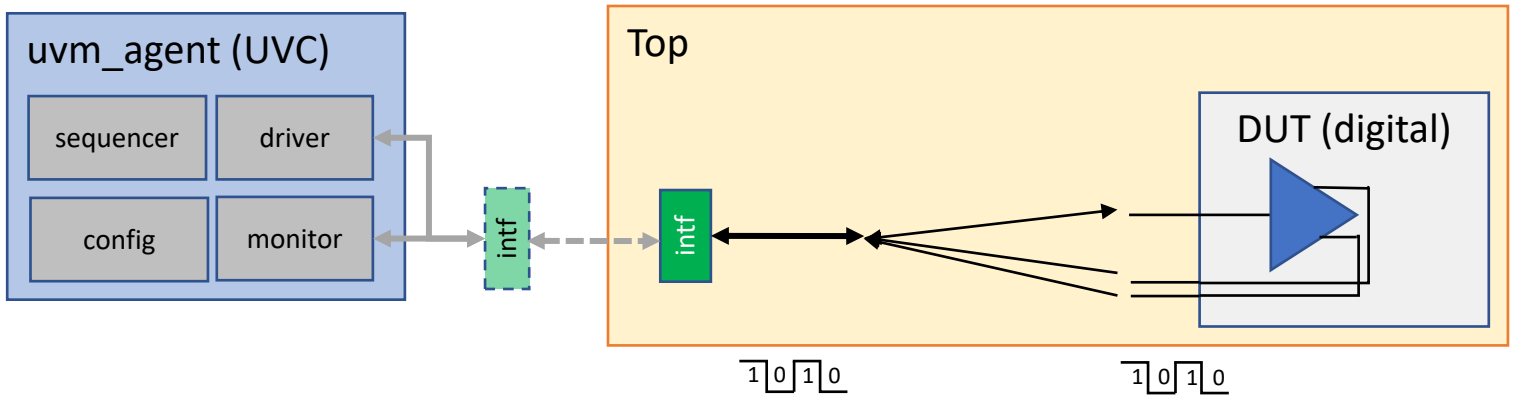


VAMS



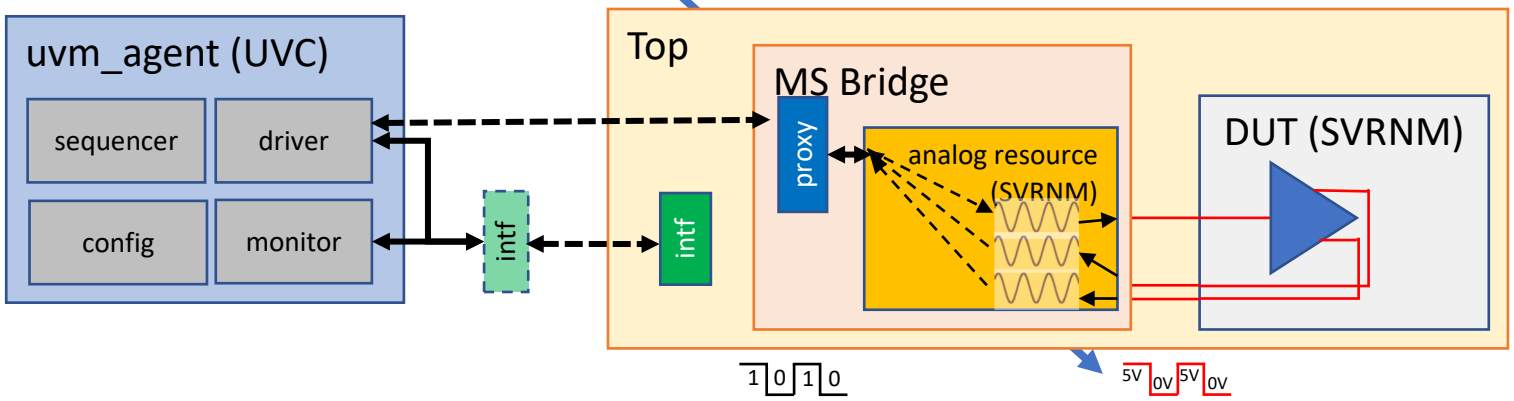
Model of Frequency Adapter Ports in SV

```
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

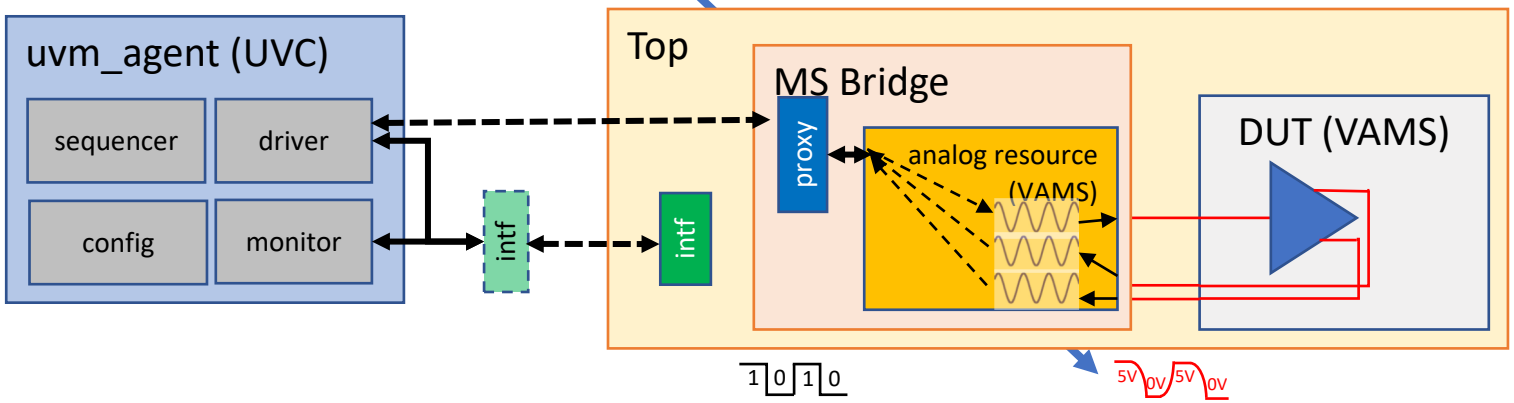
```
module freq_adapter import cds_rnm_pkg::*; (
  output wreal4state CLKOUT_P,CLKOUT_N;  // differential output
  input  wreal4state CLK_IN;             // clock input
  input  logic en_mux, [1:0] sel_mux; // register control
  input  logic [7:0] pw_adj, [1:0] sr_adj, ampl_adj;
);
```



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
```

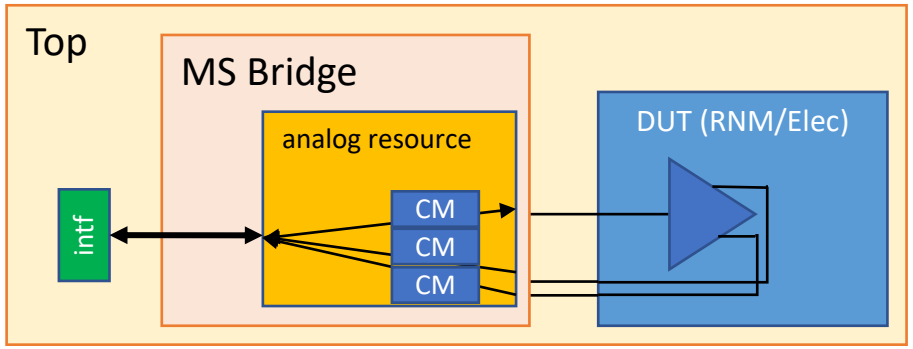


electrical uses analog solver that takes into account VIR

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)
  - IE card parameters used to configure the connect modules inserted (supply voltage, rise time, drive resistance, etc)
  - No changes required to UVM driver



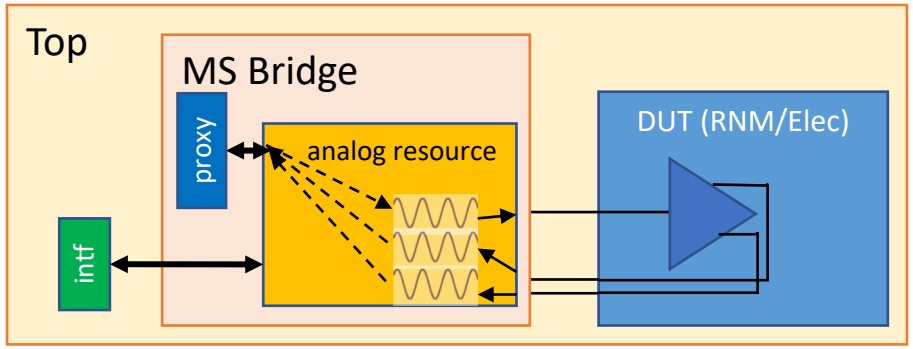
Not recommended where control over critical analog signals needed

Analog Resource for SV-RNM/VAMS

Option 2

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

This is the option used for the demo



Recommended for continuously changing signals such as sine wave

Steps to create a UVM-MS agent

- Create Bridge module
  - Contains Analog Resource and Proxy
- Extend classes for Driver, Monitor and Sequence Item
  - Use set_type_override_by_type to use extended classes

osc_bridge

```
3   module osc_bridge import cds_rnm_pkg::*; (
4     input  wire osc_clk,
5     output wire osc_clk_p,
6     output wire osc_clk_n
7     );
8
9     //UVM + MS extras
10    import uvm_pkg::*;
11    import uvm_ms_pkg::*;
12    `include "uvm_macros.svh"
13    `include "uvm_ms.svh"
14
15    //UVM package for this component
16    import osc_pkg::*;
17
18    //Selection bit to choose between single-ended clock and differential clock
19    parameter bit diff_sel = 0;
20    parameter passive = 0;
21
22    //Class proxy extends the osc_bridge_proxy included in osc_pkg.sv
23    //The implementation for the config_wave push function is defined here
24    class proxy extends osc_bridge_proxy;
25
26      function new(string name = "");
27        super.new(name);
28      endfunction : new
29
30      function void config_wave(input real ampl, bias, freq, enable);
31        core.ampl_in  = ampl;
32        core.bias_in  = bias;
33        core.freq_in  = freq;
34        core.enable   = enable;
35      endfunction
36    endclass
37
38    proxy __uvm_ms_proxy = new("__uvm_ms_proxy");
```

```
40    //Connections from proxy to core
41    always @(__uvm_ms_proxy.delay_in, __uvm_ms_proxy.duration_in, __uvm_ms_proxy.sampling_do) begin
42      core.delay_in = __uvm_ms_proxy.delay_in;
43      core.duration_in = __uvm_ms_proxy.duration_in;
44      core.sampling_do = __uvm_ms_proxy.sampling_do;
45    end
46
47    //Connections from core to proxy
48    always_comb begin
49      __uvm_ms_proxy.sampling_done = core.sampling_done;
50      __uvm_ms_proxy.ampl_out = core.ampl_out;
51      __uvm_ms_proxy.bias_out = core.bias_out;
52      __uvm_ms_proxy.freq_out = core.freq_out;
53    end
54
55    //Analog resource instantiation
56    osc_bridge_core #(.diff_sel(diff_sel), .passive(passive)) core (
57      .osc_clk(osc_clk),
58      .osc_clk_p(osc_clk_p),
59      .osc_clk_n(osc_clk_n)
60      );
61
62  endmodule
```

accellera SYSTEMS INITIATIVE

osc_driver → osc_ms_driver

UVM

```
7   class osc_driver extends uvm_driver #(osc_transaction);
8
9     // The virtual interface used to drive and view HDL signals.
10    virtual interface osc_if vif;
11
12    // Count transactions sent
13    int num_sent;
14
15    // period of the generated clock
16    real period;
17
18    // component macro
19    `uvm_component_utils_begin(osc_driver)
20      `uvm_field_int(num_sent, UVM_ALL_ON)
21      `uvm_field_real(period, UVM_ALL_ON)
22    `uvm_component_utils_end
23
24    int test_config;
25
26    // Constructor - required syntax for UVM automation and utilities
27    function new (string name, uvm_component parent);
28      super.new(name, parent);
29    endfunction : new
30
31    virtual function void build_phase(uvm_phase phase);
32      super.build_phase(phase);
33    endfunction
34
35    function void connect_phase(uvm_phase phase);
36      if (!uvm_config_db#(virtual osc_if)::get(this,"","vif", vif))
37        `uvm_error("NOVIF",{"virtual interface must be set for: ",get_full_name(),".vif"})
38    endfunction: connect_phase
39
40    // UVM run() phase
41    task run_phase(uvm_phase phase);
42      get_and_drive();
43    endtask : run_phase
```

UVM-MS

```
107   class osc_ms_source_driver extends osc_driver;
108
109     protected osc_bridge_proxy bridge_proxy;
110
111     osc_ms_transaction ms_req;
112
113     // Count transactions sent
114     int num_sent;
115     // Get value to drive onto diff_sel
116     bit diff_sel;
117
118     // Provide implmentations of virtual methods such as get_type_name and create
119     `uvm_component_utils_begin(osc_ms_source_driver)
120       `uvm_field_int(diff_sel, UVM_ALL_ON)
121     `uvm_component_utils_end
135     virtual function void build_phase(uvm_phase phase);
136       super.build_phase(phase);
137
138       if(!uvm_config_db#(osc_bridge_proxy)::get(this,"","bridge_proxy",bridge_proxy))
139         `uvm_error(get_type_name(),"bridge proxy not configured");
140     endfunction
```

```
153   // Gets transactions from the sequencer and passes them to the driver.
154   task osc_ms_source_driver::get_and_drive();
155     forever begin
156       // Get new item from the sequencer
157       seq_item_port.get_next_item(req);
158       $cast(ms_req,req);
159       // Drive the item
160       drive_transaction(ms_req);
161       fork
162         #(200*1ns); //Time for transaction
163         begin : sample_thread
164           #(1ns) bridge_proxy.sampling_do = 1;
165           #(1ns) bridge_proxy.sampling_do = 0;
166         end
167       join
168       // Communicate item done to the sequencer
169       seq_item_port.item_done();
170     end
171   endtask : get_and_drive
```

accellera SYSTEMS INITIATIVE

osc_monitor → osc_ms_monitor

UVM

```
6   class osc_monitor extends uvm_monitor;
7
8     // Virtual Interface for monitoring DUT signals
9     virtual interface osc_if vif;
10
38    osc_transaction osc_clk_transaction, osc_clk_p_transaction;
39
40    // This TLM port is used to connect the monitor to the scoreboard
41    uvm_analysis_port #(osc_transaction) item_collected_port;
42
65    function void build_phase(uvm_phase phase);
66      super.build_phase(phase);
67    endfunction: build_phase
68
69    function void connect_phase(uvm_phase phase);
70      if (!uvm_config_db#(virtual osc_if)::get(this, get_full_name(), "vif", vif))
71        `uvm_error("NOVIF",{"virtual interface must be set for: ",get_full_name(),".vif"})
72      if (!uvm_config_int::get(this,"","diff_sel", diff_sel))
73        `uvm_error("NOCONFIG",{"value must be set for: ",get_full_name(),".diff_sel"})
74      else  `uvm_info("CONFIG_CORRECT",{"Value of ",get_full_name(), $sformatf(".diff_sel = %d",diff_sel)
75    endfunction: connect_phase
76
```

UVM-MS

```
201   class osc_ms_source_monitor extends osc_monitor;
202
203     // Virtual Interface for monitoring DUT signals
204     protected osc_bridge_proxy bridge_proxy;
205     // Count transactions collected
206     int num_col;
```

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

```
327   task osc_ms_source_monitor::collect_transaction();
328     // This monitor re-uses its data items for ALL transactions
329     transaction = osc_ms_transaction::type_id::create("transaction", this);
330     forever begin
331       @(posedge bridge_proxy.sampling_done);
332       // Begin transaction recording
333       void'(begin_tr(transaction, "analog_clock source Monitor"));
334       transaction.data_type = OSC_MS_SAMPLE;
335       transaction.ampl = bridge_proxy.ampl_out;
336       transaction.bias = bridge_proxy.bias_out;
337       transaction.freq = bridge_proxy.freq_out;
338       `uvm_info(get_type_name(),
339         $psprintf("source transaction collected :\n%s",
340         transaction.sprint()), UVM_LOW) //Temporarily dropped verbosity
341       if (checks_enable)
342         perform_checks();
343       if (coverage_enable)
344         perform_coverage();
345       // Send transaction to scoreboard via TLM write()
346       item_collected_port.write(transaction);
347       num_col++;
348       fork
349         begin : wait_for_sampling_done
350           @(negedge bridge_proxy.sampling_done);
351           disable wait_for_timeout;
352         end
```

osc_bridge

```
//Connections from core to proxy
always_comb begin
  __uvm_ms_proxy.sampling_done = core.sampling_done;
  __uvm_ms_proxy.ampl_out = core.ampl_out;
  __uvm_ms_proxy.bias_out = core.bias_out;
  __uvm_ms_proxy.freq_out = core.freq_out;
end
```

osc_transaction → osc_ms_transaction

UVM

```
7   class osc_transaction extends uvm_sequence_item;
8
9     rand real freq; // frequency of input clock
10    bit diff_sel;
11
12    `uvm_object_utils_begin(osc_transaction)
13      `uvm_field_real(freq, UVM_ALL_ON)
14    `uvm_object_utils_end
15
16    // Constraints go here
17    constraint default_freq_c {
18      freq > 5e8;
19      freq < 1e9;
20    }
21
22    // Constructor - required syntax for UVM automation and utilities
23    function new (string name = "osc_transaction");
24      super.new(name);
25    endfunction : new
26
27  endclass : osc_transaction
```

UVM-MS

```
35  class osc_ms_transaction extends osc_transaction;
36
37    rand osc_ms_data_type_e data_type;
38
39    // Drive fields
40    rand real ampl;
41    rand real bias;
42    rand bit enable;
43
44    //Measurment fields
45    rand real delay;    //Delay in ns
46    rand int  duration;
47
48    `uvm_object_utils_begin(osc_ms_transaction)
49      `uvm_field_enum(osc_ms_data_type_e, data_type, UVM_DEFAULT)
50      `uvm_field_real(ampl, UVM_DEFAULT)
51      `uvm_field_real(bias, UVM_DEFAULT)
52      `uvm_field_int(enable, UVM_DEFAULT)
53      `uvm_field_real(delay, UVM_DEFAULT)
54      `uvm_field_int(duration, UVM_DEFAULT)
55    `uvm_object_utils_end
56
57    // Constraints go here
58    // To override, use the same constraint name or TCL to disable
59    constraint default_drive_trans_c {
60      ampl > 0.95;
61      ampl < 1.65;
62      bias inside {[-0.05:0.5]};
63      enable dist { 1'b0 := 1 , 1'b1 := 5 };
64    }
65    constraint default_measurement_trans_c {
66      duration > 20;
67      duration < 32;
68      delay > 0.0;
69      delay < 1.0;
70    }
```

accellera SYSTEMS INITIATIVE

freq_adpt_tb → freq_adpt_ms_tb

UVM

```
class freq_adpt_tb extends uvm_env;

  // component macro
  `uvm_component_utils(freq_adpt_tb)

  registers_env registers;
  osc_env freq_generator;
  osc_env freq_detector;

  freq_adpt_scoreboard freq_adpt_sb;

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

  // UVM build() phase
  function void build_phase(uvm_phase phase);
    `uvm_info("MSG","In the build phase",UVM_MEDIUM)

    // set up virtual interfaces for UVCs and scoreboard
    uvm_config_db#(virtual osc_if)::set(this,"freq_generator*","vif", top.generator_if);
    uvm_config_db#(virtual osc_if)::set(this,"freq_detector*", "vif", top.detector_if);
    uvm_config_db#(virtual registers_if)::set(this,"registers.reg_agent.*", "reg_vif", top.reg_if);

    // config the value of diff_sel for freq_generator to 0 - single-ended clock generation
    uvm_config_int::set(this,"freq_generator.agent.*","diff_sel", 0);
    // config the value of diff_sel for freq_detector to 1 - differential clock detection
    uvm_config_int::set(this,"freq_detector.agent.*","diff_sel", 1);

    super.build_phase(phase);

    // create the envs for the generator, detector, registers and scoreboard
    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

  // UVM connect_phase
  function void connect_phase(uvm_phase phase);
    // Connect the TLM ports from the UVCs to the scoreboard
    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
```

UVM-MS

```
class freq_adpt_ms_tb extends freq_adpt_tb;

  // component macro
  `uvm_component_utils(freq_adpt_ms_tb)

  //freq_adpt_ms_scoreboard freq_adpt_sb;

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

  // UVM build() phase
  function void build_phase(uvm_phase phase);
    `ifdef UVM_AMS
    // set up brdige proxy pointer references to generator and detector UVCs
    uvm_config_db #(osc_bridge_proxy)::set(this,"freq_generator.agent.*","bridge_proxy", top.generator_bridge.__uvm_ms_proxy);
    uvm_config_db #(osc_bridge_proxy)::set(this,"freq_detector.agent.*","bridge_proxy", top.detector_bridge.__uvm_ms_proxy);
    `endif

    // 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_source_driver::get_type());
    set_type_override_by_type(osc_monitor::get_type(),osc_ms_source_monitor::get_type());
    set_type_override_by_type(freq_adpt_scoreboard::get_type(),freq_adpt_ms_scoreboard::get_type());

    super.build_phase(phase);

  endfunction

endclass : freq_adpt_ms_tb
```

accellera SYSTEMS INITIATIVE

freq_adpt_scoreboard → freq_adpt_ms_scoreboard

UVM

```
4   class freq_adpt_scoreboard extends uvm_scoreboard;
5
6     cover_e coverage_control = COV_ENABLE;//COV_ENABLE;
7
8     // component utils macro
9     `uvm_component_utils_begin(freq_adpt_scoreboard)
10      `uvm_field_enum(cover_e, coverage_control, UVM_ALL_ON)
11    `uvm_component_utils_end
12
13    // define TLM port imp object
14    `uvm_analysis_imp_decl(_registers)
15    `uvm_analysis_imp_decl(_osc_gen)
16    `uvm_analysis_imp_decl(_osc_det)
17
18    uvm_analysis_imp_registers #(registers_packet, freq_adpt_scoreboard) sb_registers_in;
19    uvm_analysis_imp_osc_gen #(osc_transaction, freq_adpt_scoreboard) sb_osc_gen;
20    uvm_analysis_imp_osc_det #(osc_transaction, freq_adpt_scoreboard) sb_osc_det;
```

```
128   // write()
129   virtual function void write_registers(registers_packet packet);
130     registers_packet sb_packet;
131     // Make a copy for storing in the scoreboard
132     $cast(sb_packet, packet.clone());  // Clone returns uvm_object type
133     reg_packets_in++;
134
135     if(sb_packet.addr > 7)
136       reg_in_drop ++;
137
138     else begin
139       if(sb_packet.wen && !sb_packet.r
140         rdata <= 8'h00;
141         if(sb_packet.addr < 3)
142           INT_REG[sb_packet.addr] <= sl
143       end
144       MUX_REG = INT_REG[1];
145       en_mux = MUX_REG[2];
146       sel_mux = MUX_REG[1:0];
147     end
148   endfunction
```

```
151     virtual function void write_osc_gen(osc_transaction packet);
152       osc_transaction sb_packet;
153       $cast(sb_packet, packet.clone());  // Clone returns uvm_object type
154       // separate the case of diff_sel
155       freq_generator_packets_in++;
156       freq_in_reg = sb_packet.freq;
157     endfunction: write_osc_gen
158
159     // write() function for detector
160     virtual function void write_osc_det(osc_transaction packet);
161       osc_transaction sb_packet;
162       $cast(sb_packet, packet.clone());  // Clone returns uvm_object type
163
164       if(en_mux) begin
```

UVM-MS

```
class freq_adpt_ms_scoreboard extends freq_adpt_scoreboard;
  `uvm_component_utils(freq_adpt_ms_scoreboard)
```

```
283     // write() function for registers is not required
284
285     // write() function for frequency generator
286     virtual function void write_osc_gen(osc_transaction packet);
287       osc_ms_transaction sb_packet;
288       $cast(sb_packet, packet.clone());  // Clone returns uvm_object type
289       freq_generator_packets_in++;
290       freq = sb_packet.freq;
291       ampl = sb_packet.ampl;
292       bias = sb_packet.bias;
293       `uvm_info("WRITE_OSC_GEN",$sformatf("\nFreq = %f\tAmpl = %f \t Bias = %f",freq,ampl,bias),UVM_LOW)
294     endfunction: write_osc_gen
295
296     // write() function for detector
297     virtual function void write_osc_det(osc_transaction packet);
298       osc_ms_transaction sb_packet;
299       $cast(sb_packet, packet.clone());  // Clone returns uvm_object type
300
301       if(en_mux) begin
302         // Compare output freq with expected result calculated from input freq
```

accellera SYSTEMS INITIATIVE

How to check tests?

- Send expectations about ongoing outputs to agents monitoring meters.
  - Use Assertions to report errors when output is not within expected parameters
  - Test sends sequence item to agents as expectations change.
  - Test must be able to predict expectations.
- Reference model gets sequence items affecting the block and predicts expectations.
  - Can be sent to agents monitoring outputs. Where data can be used in assertions.
  - Sent as item to a scoreboard for check against similar item collected from DUT.
  - Scoreboard uses item compare method to determine match / no match.
- How to coordinate item generation between DUT and Reference model?
  - In digital designs the TLM model assumes that output transactions are generated as a result of input transactions.
  - Is such a transaction response the right approach in general for Analog?

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Demo

accellera SYSTEMS INITIATIVE

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

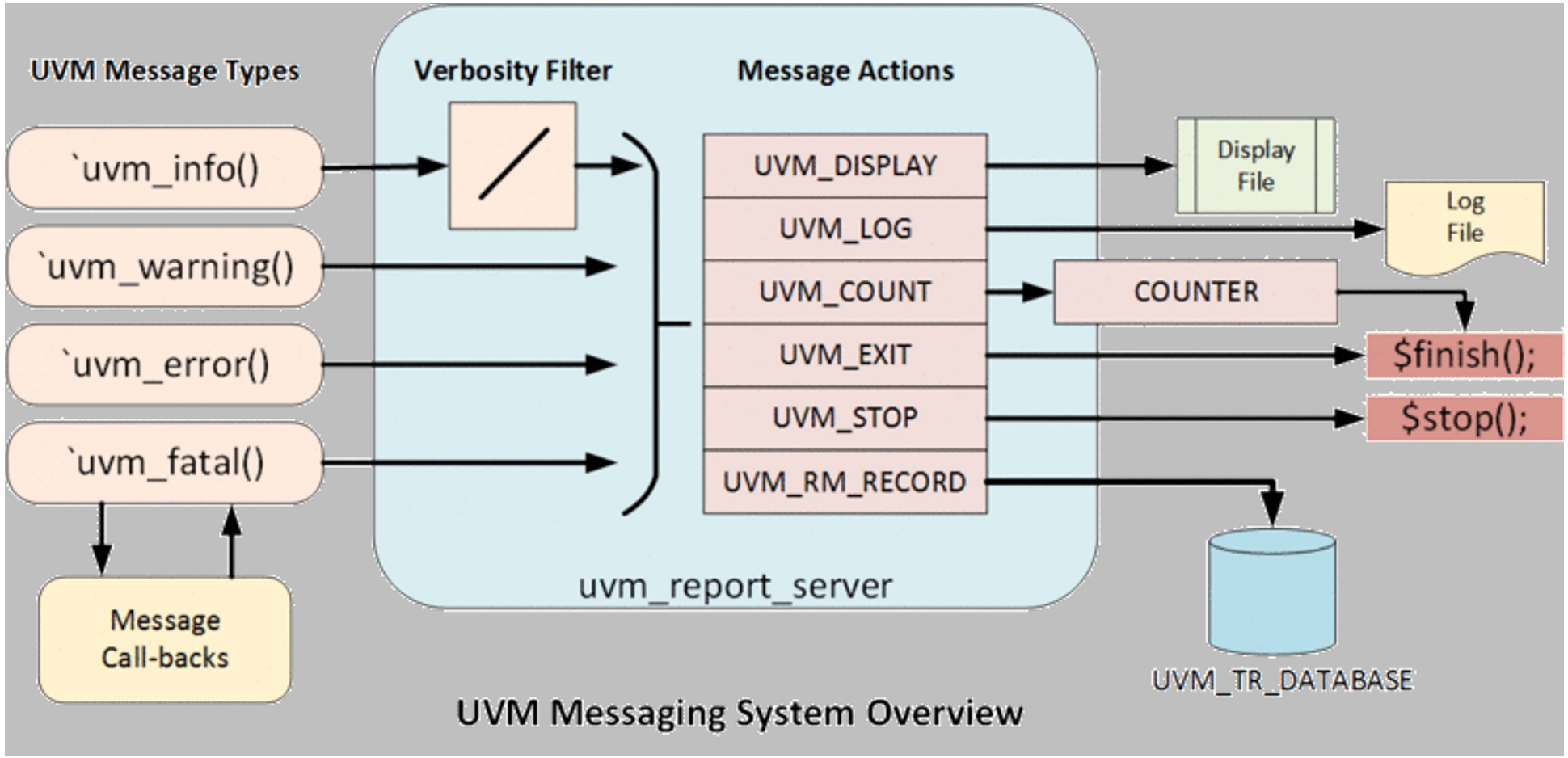
UVM Messaging

accellera SYSTEMS INITIATIVE

Messages for Debug and Error Reporting

- Debugging activity inside a large environment with many agents 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 access SV bridge
- `` `include "uvm_ms.vamsh" `` in Verilog-AMS analog resource or
  `` `include "uvm_ms.vdmsh" `` in SystemVerilog analog resource
  - localparams to define UVM Verbosity levels as integers to match UVM enum
  - Macros to wrap the `uvm_ms_*` reporting function calls defined in uvm_ms.svh
- `` `include "uvm_ms.svh" `` in MS Bridge (SV)
  - Definitions of the functions called by analog resource
  - Provides macros for `` `uvm_ms_[info|warning|error|fatal](…) ``
  - Utilizes the `"__uvm_ms_proxy"` declaration as the originating path for analog resource UVM messages

UVM Messaging Example for Verilog-AMS Resource

- Use analog domain to detect the issue and toggle a flag
- Flag is detected by absdelta to then report the message via the digital engine
- Example

```
analog begin
  if((I_PLUS > 1.0) && !I_thr_triggered) I_thr_triggered = 1;
  else if(I_PLUS < 0.9) I_thr_triggered = 0;
end
//Convert the detection in the analog block to a UVM report.
string message;
always@(absdelta(I_thr_triggered,1,0,0,1)) begin
  $sformat(message,"The Current is above the thresholds @ %e",I_PLUS);
  if(I_thr_triggered) `uvm_ms_error(P__TYPE,message)
end
```

Up-scope function call

UVM Message – Analog block

"uvm_ms.vamsh" uvm_ms_info function is found via up-scope and executed from SV bridge

```
`define uvm_ms_info(id,message,uvm_verbosity) \
        uvm_ms_info(id,message,uvm_verbosity,$sformatf("%m"),`__FILE__ ,`__LINE__ );
```

osc_core.vams

```
`include "uvm_ms.vamsh"
`uvm_ms_info("FREQ_UPDATE",$sformatf("freq=%e Hz period=%e ns", freq_in, out_period),\
UVM_MEDIUM)
```

"uvm_ms.svh"

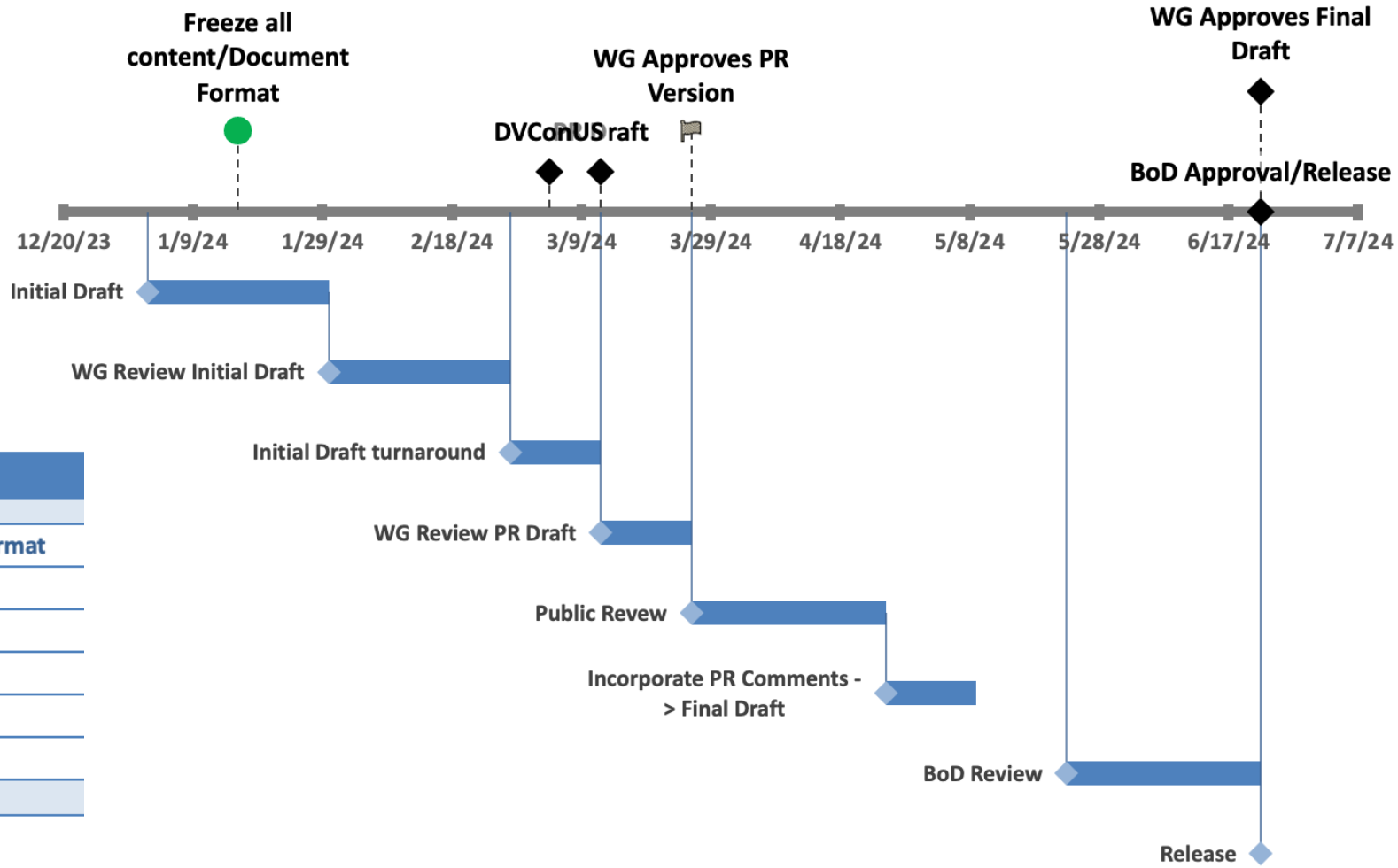
```
function void uvm_ms_info(id,message,uvm_verbosity,uvm_path,`__FILE__ ,`__LINE__ );
  uvm_component CTXT;
  CTXT=uvm_ms_get_bridge_path(uvm_path); // get path to uvm_component in top.bridge
  CTXT.uvm_report_info(id,message,uvm_verbosity'(verbosity_level),file,line);
endfunction: uvm_info
```

osc_bridge.sv

```
`include "uvm_ms.svh"
`uvm_ms_reporter        // instantiates uvm_ms_reporter component to be used with messaging
```

```
UVM_INFO ../uvc_lib/osc/vams/osc_bridge_core.vams(98) @ 52001.098068ns: top.detector_bridge
[FREQ_UPDATE] The Current is above the threshold @ 1.178812e+00A
```

UVM-AMS Standard Release Schedule



Milestones

Date	Label
1/16/24	Freeze all content/Document Format
3/12/24	PR Draft
3/26/24	WG Approves PR Version
3/4/24	DVConUS
6/22/24	WG Approves Final Draft
6/22/24	BoD Approval/Release
	Insert new rows above this one

Conclusions

- There is a need for more advanced, standard methodologies for scalable, reusable and metric-driven mixed-signal (AMS/DMS) verification
- The UVM-AMS WG 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

Changes for UVM-AMS from UVM

- `uvm_*_printer print_real()` uses `%f` formatting, which truncates very small values
  - Override with `uvm_radix_real_exp`
- UVM messaging macros don't work in modules
  - Created macros and methodology to support Analog Resource
- UVM-MS specific include/import files

Statement	Usage	
import uvm_ms_pkg::*;	Within the MS Bridge and uvm_ms_agent	Defines uvm_ms_proxy template class
`include "uvm_ms.vamsh"	Within analog_resource modules defined as Verilog-AMS	Defines UVM-MS messaging macro/functions
`include "uvm_ms.dmsh"	Within analog_resource modules defined as SystemVerilog	Defines UVM-MS messaging macro/functions
`include "uvm_ms.svh"	Within the MS Bridge to enable the messaging from the analog_resource	Requires the MS Proxy instance to be named __uvm_ms_proxy

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Questions?

accellera SYSTEMS INITIATIVE