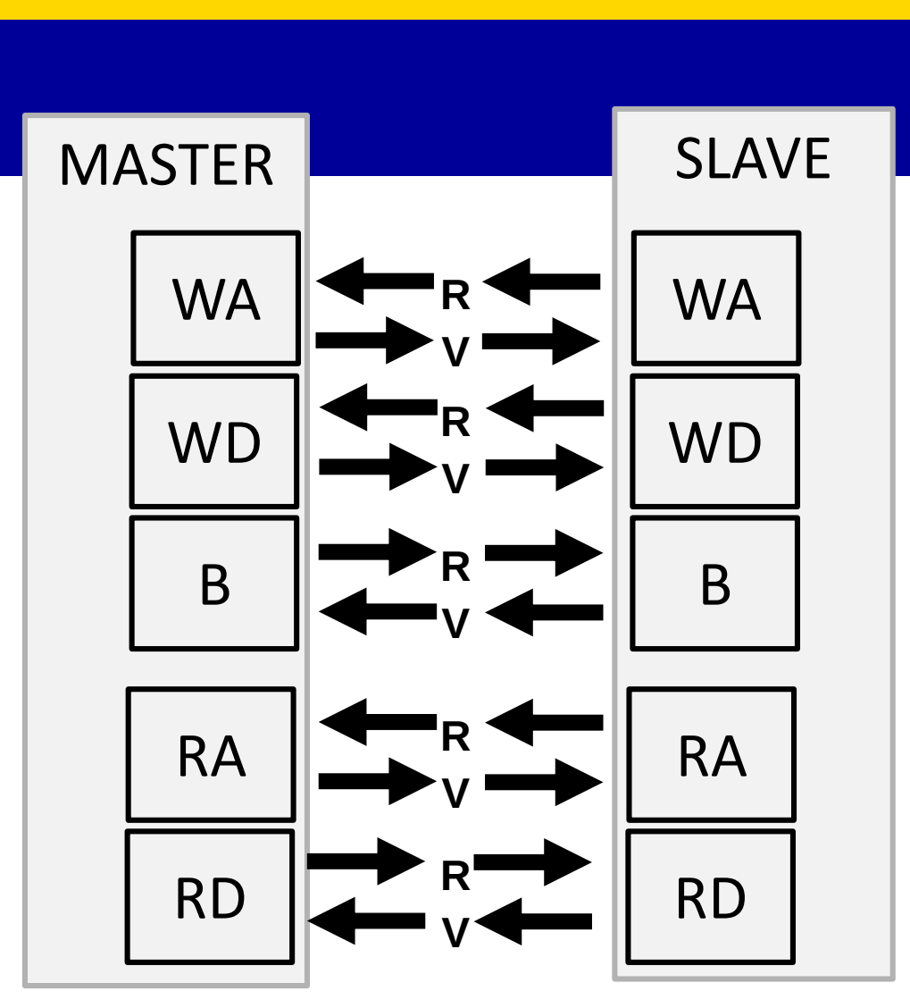
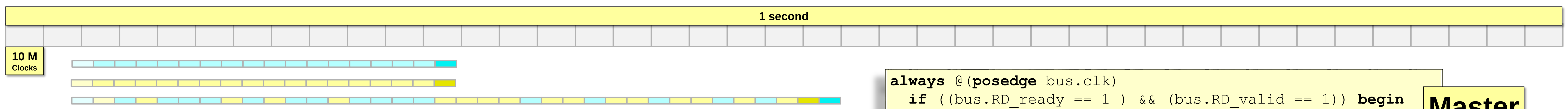
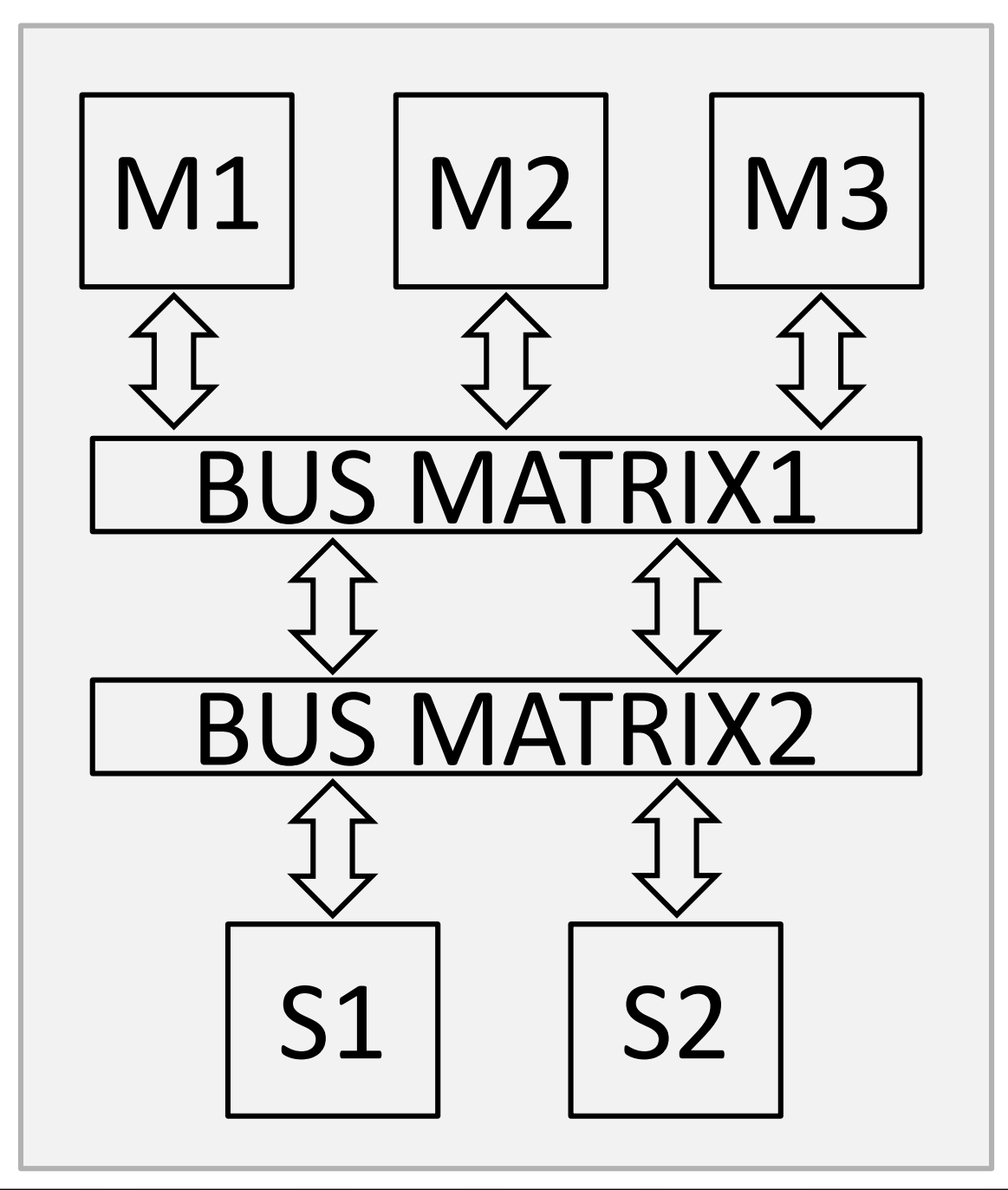
2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

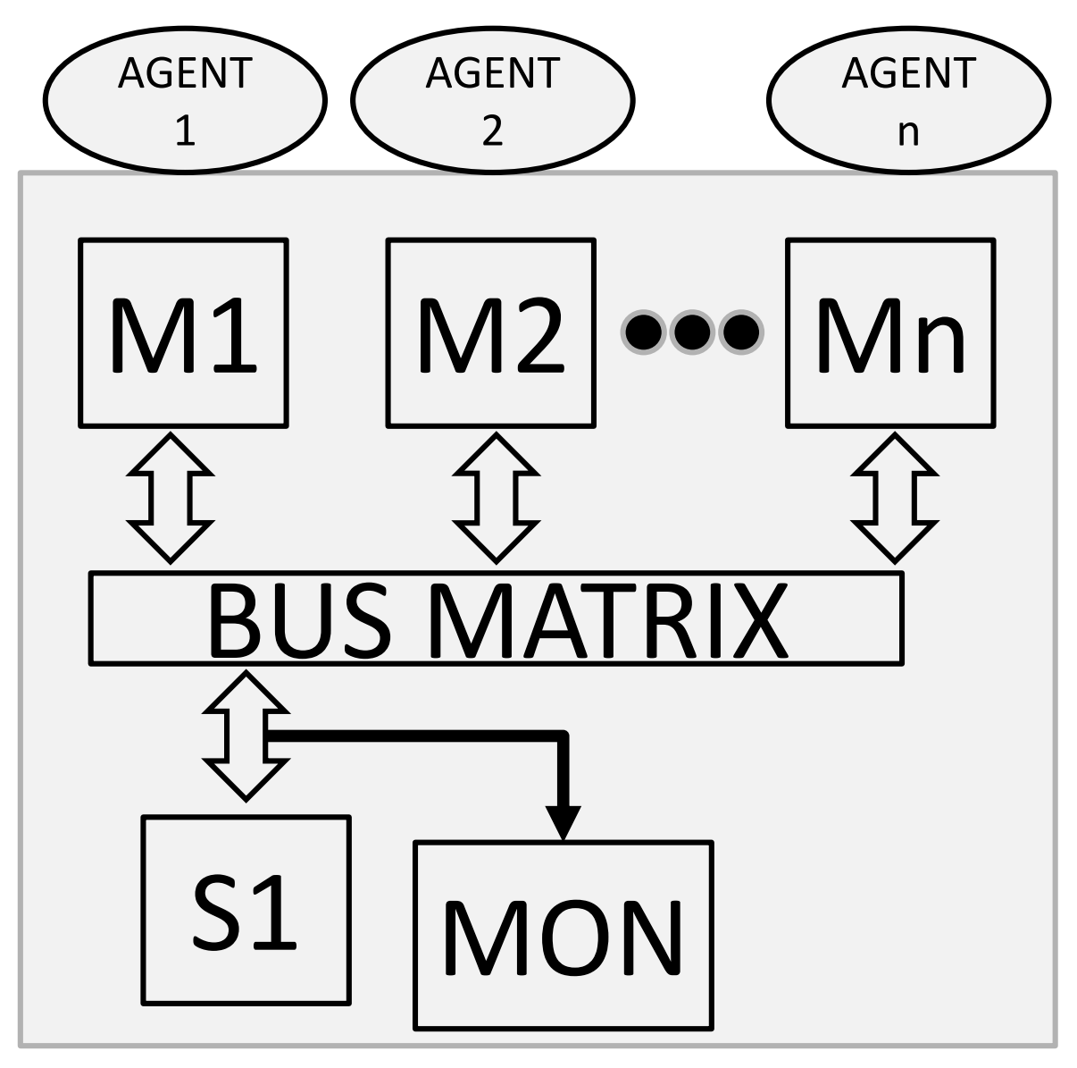
Goldilocks and System Performance Modelling A SystemVerilog Adaptive Rate Control (ARC) Stimulus Generation Methodology

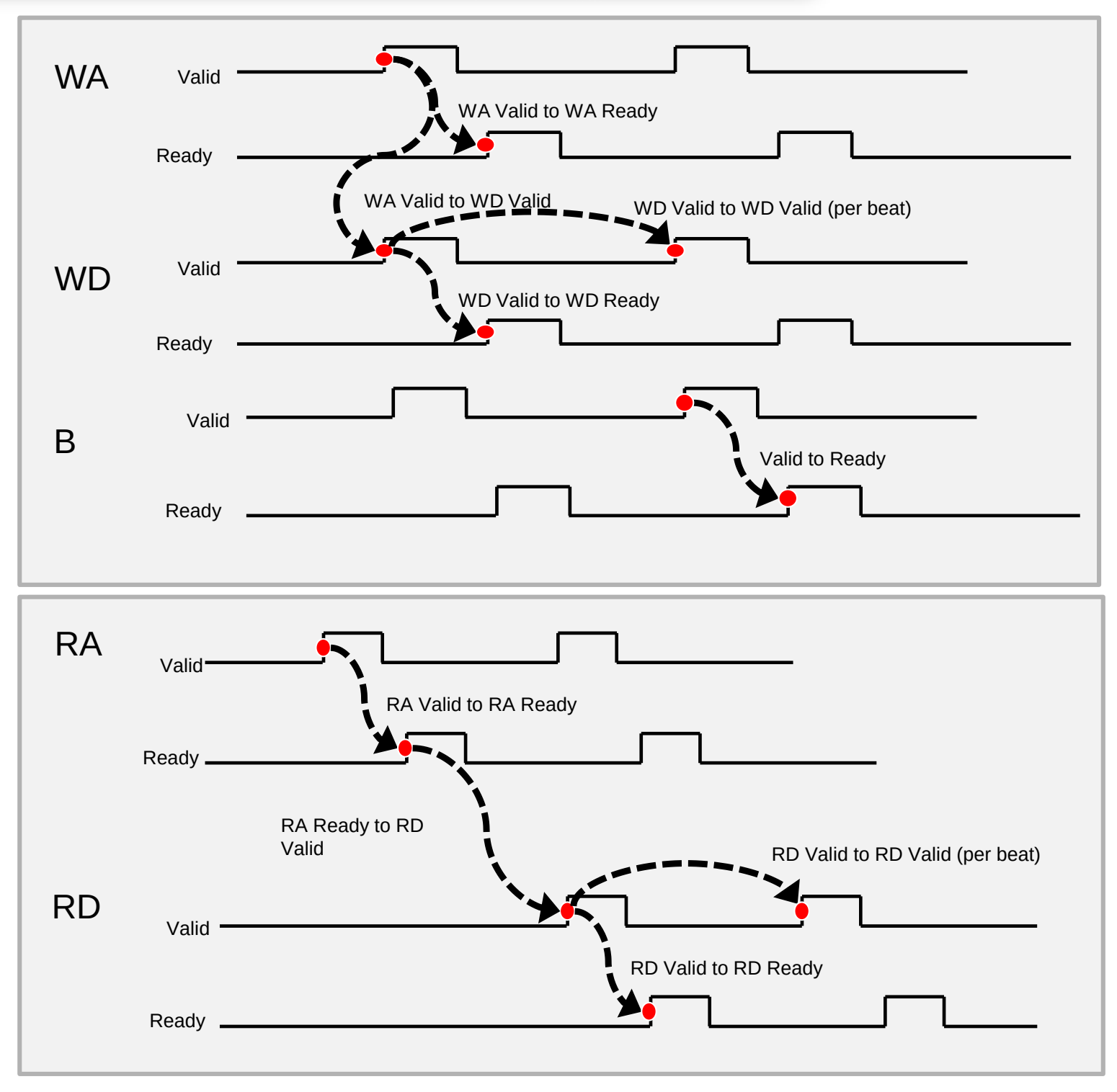




```
Goal: Sustain 300MB/sec transfer rate at the slave.
400Mhz clock = 2.5ns/clock. Payload is 256B. 16 bytes per beat. 16 clocks are needed.
How long can we "wait" and get 400MBps? (256B/400MBps)*400Mhz = 256 clocks
How long can we "wait" and get 300MBps? (256B/300MBps)*400Mhz = 341 clocks
longest # of clocks = (#bytes/BW)*Frequency
```







READ/WRITE

```
task automatic READ(addr_t addr,
    inout array_of_bytes_t data,
       transaction_delay_t transaction_delays);
   id_t tag = generate_new_tag();;
   beats_t beats;
   load_delays(tag, transaction_delays);
   RA(tag, addr, number_of_beats);
   RD(tag, beats);
   pack_beats_to_bytes(tag, beats, data);
 endtask

 task automatic WRITE(addr_t addr,
     input array_of_bytes_t data,
        transaction_delay_t transaction_delays);
   id_t tag = generate_new_tag();
   beats_t beats;
   load_delays(tag, transaction_delays);
   pack_bytes_to_beats(tag, data, beats);
   fork
     WA(tag, addr);
     WD(tag, beats);
     B(tag);
   join
 endtask
```

Top

```
module top;
  reg clk;
  channel            bus(clk);
  slave_interface     si(bus);
  master_interface    mi(bus);
  monitor_interface mon(bus);

  initial begin
    uvm_config_db#(virtual master_interface)::
      set( null, "", "vif", mi);
    uvm_config_db#(virtual monitor_interface)::
      set( null, "", "monitor_vif", mon);

    run_test("test1");
  end
  ...
endmodule
```

Master

```
always @(posedge bus.clk)
  if ((bus.RD_ready == 1 ) && (bus.RD_valid == 1)) begin
    rd[bus.RD_tag][bus.RD_beat_count] = bus.RD_data;
    if (bus.RD_beat_count == 0)
      r_done[bus.RD_tag] = 1;
    @(negedge bus.clk);
    bus.RD_ready = 0;
  end
```

```
always @(posedge bus.RD_valid) begin
  repeat(delay_table.get_per_beat_delay(
    "RD_valid_to_RD_ready",
      bus.RD_tag, bus.RD_beat_count))
      @(posedge bus.clk);
  @(negedge bus.clk);
  bus.RD_ready = 1;
end
```

Test

```
class test1 extends uvm_test;
  `uvm_component_utils(test1)
  agent     agent_h[int];
  sequenceA   seq_h[int];

  int parallel_threads = 16;
  int number_of_transactions = 100;

  function void build_phase(uvm_phase phase);
    $value$plusargs("transactions=%d", number_of_transactions);
    $value$plusargs("threads=%d", parallel_threads);
    for(int i = 0; i < parallel_threads; i++) begin
      agent_h[i] = agent::type_id::create($sformatf("a%05d", i), this);
      agent_h[i].vif = vif;
    end
  endfunction

  task run_phase(uvm_phase phase);
    phase.raise_objection(this);
    for(int i = 0; i < parallel_threads; i++) begin
      seq_h[i] = sequenceA::type_id::create($sformatf("seq%05d", i));
      seq_h[i].sequence_id = i;
      seq_h[i].base_address = (i+1) * 2048;
      seq_h[i].number_of_transactions = number_of_transactions;
    end
    foreach (seq_h[i])
      fork
        automatic int j = i;
        #j seq_h[j].start(agent_h[j].sqr);
      join_none
    wait fork;
    phase.drop_objection(this);
  endtask
endclass
```

Bus

```
interface channel(
    input wire clk);
  logic   RA_ready;
  logic   RA_valid;
  tag_t   RA_tag;
  addr_t RA_addr;
  int     RA_beat_count;

  logic   RD_ready;
  logic   RD_valid;
  tag_t   RD_tag;
  int     RD_beat_count;
  data_t RD_data;

  logic   WA_ready;
  logic   WA_valid;
  tag_t   WA_tag;
  addr_t WA_addr;

  logic   WD_ready;
  logic   WD_valid;
  tag_t   WD_tag;
  int     WD_beat_count;
  data_t WD_data;

  logic   B_ready;
  logic   B_valid;
  tag_t   B_tag;
endinterface
```

Transaction

```
class transaction extends
    uvm_sequence_item;
  `uvm_object_utils(transaction)
  string whence;
  event really_done;

  transaction_delay_t t_delays;

  int sequence_id;
  int transaction_id;

  int tid;
  static int g_tid;

  rw_t rw;
  bit [31:0] addr;
  array_of_bytes_t data;

  function new(string name = "t");
    super.new(name);
    tid = g_tid++;
    t_delays = new();
  endfunction

  function string convert2string();
    string data_string;
    data_string =
      pretty_print_array_of_bytes(data);
    return $sformatf(
  "tid=%0d, %s addr=0x%x, %0d bytes, \
  data=%s,#clocks=(%0d,%0d), whence=%s",
  tid,rw, addr, data.size(), data_string,
  t_delays.actual_number_of_clock_cycles,
      ((rw==1)?
        t_delays.total_read_delay():
        t_delays.total_write_delay()),
       whence);
  endfunction
endclass
```

Transaction Delay

```
class transaction_delay_t;
 id_t tag;

 // The delay that ACTUALLY happened.
 // Set after transaction completes.
 int actual_number_of_clock_cycles = -1;

 // The desired delay.
 // The delays that will be used.
 int           delay[string];
 int per_beat_delay[string][int];
  // per_beat_delay["RD_valid_to_RD_valid"][0]
 function int get_delay(string name);
   return delay[name];
 endfunction

 function int get_per_beat_delay(
     string name, int beat_index);
   return per_beat_delay[name][beat_index];
 endfunction

 function void post_randomize();
   load_string_lookup_table();
 endfunction

constraint value_range {
   ...
   foreach (delay_WD_valid_to_WD_valid[x])
     delay_WD_valid_to_WD_valid[x]
      < max_clock_delay;
 }
 constraint sum_read_delay {
   delay_RD_valid_to_RD_ready.sum() + // Master
   delay_RA_valid_to_RA_ready       + // Slave
   delay_RA_ready_to_RD_valid       +
   delay_RD_valid_to_RD_valid.sum() < read_cycles;
 }
 constraint sum_write_delay {
   delay_WD_valid_to_WD_valid.sum() + // Master
   delay_WA_valid_to_WD_valid       +
   delay_WD_valid_to_WA_valid       +
   delay_WD_valid_to_WD_ready.sum() + // Slave
   delay_B_valid_to_B_ready         +
   delay_WA_valid_to_WA_ready       < write_cycles;
 }
```

Sequence

```
class sequenceA extends uvm_sequence#(transaction);
  `uvm_object_utils(sequenceA)
  int sequence_id;
  bit [31:0] base_address, address;
  int number_of_transactions = 100;
  transaction t;

  task body();
    address = base_address;
    for (int i = 0; i < number_of_transactions; i++) begin
      t = transaction::type_id::create("t");
      t.transaction_id = i;
      t.sequence_id = sequence_id;
      t.whence = {get_full_name(), $sformatf("-t%0d", i)};

      if (!t.t_delays.randomize() with {
          delay_WD_valid_to_WD_valid.size()
       == delay_RD_valid_to_RD_valid.size(); ... })
        ...
      t.addr =  address;

      // Fill in some data. Tag the first four bytes for debug.
      for(int j = 0; j < number_of_bytes; j++) t.data[j] = j;
      for(int j = 0; j < 4; j++) t.data[j] = i;

      t.rw = WRITE;
      start_item(t);
      finish_item(t);
      @(t.really_done);
      write_cycles = t.t_delays.actual_number_of_clock_cycles;

      t.rw = READ;
      start_item(t);
      finish_item(t);
      @(t.really_done);
      read_cycles = t.t_delays.actual_number_of_clock_cycles;

      for(int j = 0; j < t.data.size(); j++) // COMPARE
        if (t.data[j] != bytes_written[j]) ...
    end
  endtask
endclass
```

Driver

```
class driver extends
     uvm_driver#(transaction);
  `uvm_component_utils(driver)
  ...
  virtual master_interface vif;
  transaction t;
  task run_phase(uvm_phase phase);
    forever begin
      @(posedge vif.bus.clk);
      seq_item_port.get_next_item(t);
      vif.put_work(t);
      seq_item_port.item_done();
    end
  endtask
endclass
```

Rich Edelman
rich_edelman@mentor.com
Shashi Bhutada
shashi_bhutada@mentor.com
Mentor Graphics®