February 25-28, 2013
DoubleTree, San Jose

accellera SYSTEMS INITIATIVE

LOCKHEED MARTIN

Weathering the Verification Storm

Methodology Enhancements used on a Next Generation Weather Satellite CDH Program

By

Michael Horn

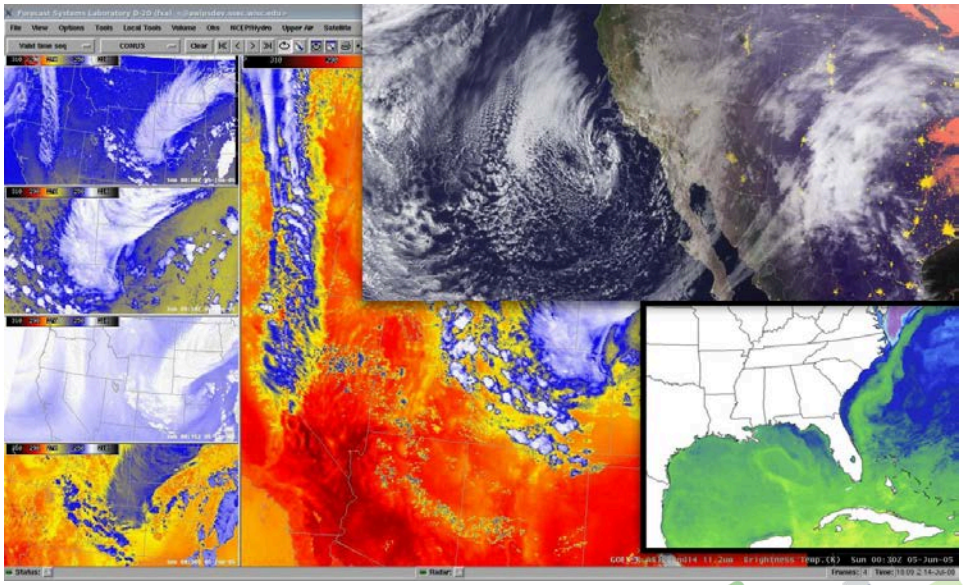
Principal Verification Architect

Mentor Graphics

Mentor Graphics®

What Verification Storm?

- Next Generation Weather Satellite
- Used for detecting and tracking severe weather such as hurricanes



What Verification Storm?

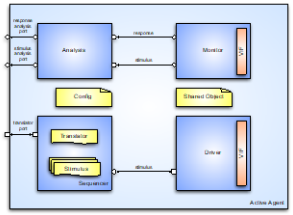
- 8 FPGAs in Command and Data Handler
- Advanced Verification Techniques
  - OVM™
  - SystemVerilog Assertions

```
assert property (@(posedge Clock) Req |-> ##[1:2] Ack);
```

  - Independent Verification Team

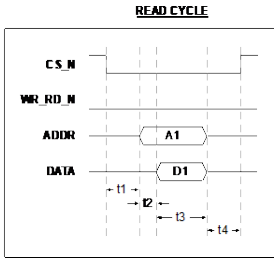
Three Methodology Enhancements

1. Parameterized Base Agent



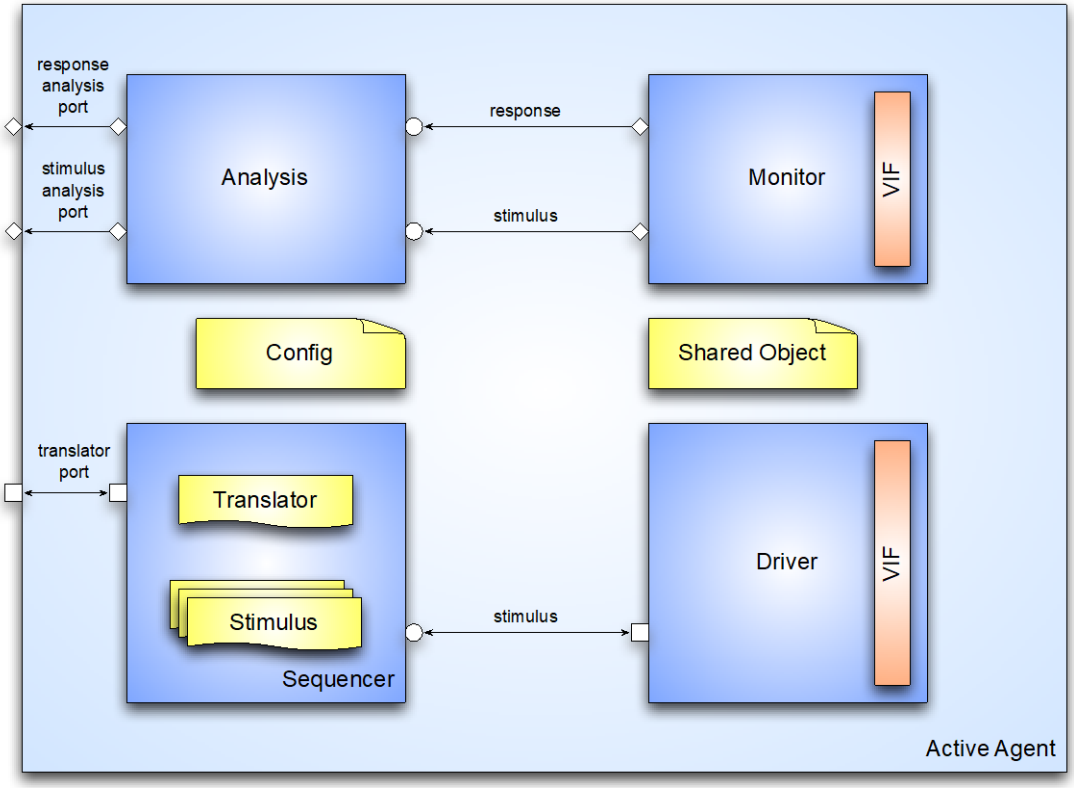
2. Driver Level Data Rate Controls

3. Interface Timing Controls



Parameterized Base Agent

- UVM Agents All Look Alike



Make an Agent Configurable

DVCon 2013 Design & Verification Conference & Exhibition
Sponsored By:
accellera SYSTEMS INITIATIVE

```
class agent_base extends uvm_agent;
  // members
  …
  <driver_class> driver;

  function void build_phase(uvm_phase phase);
    …
    driver = <driver_class>::type_id::create("driver", this);
  endfunction : build_phase
  …
endclass : agent_base
```

Make an Agent Configurable

DVCon 2013 Design & Verification Conference & Exhibition
Sponsored By:
accellera SYSTEMS INITIATIVE

```
class agent_base #(type DRV = agent_drv_base) extends uvm_agent;
  // members
  …
  DRV driver;

  function void build_phase(uvm_phase phase);
    …
    driver = DRV::type_id::create("driver", this);
  endfunction : build_phase
  …
endclass : agent_base
```

Make an Agent Configurable

DVCon 2013 Design & Verification Conference & Exhibition
Sponsored By:
accellera SYSTEMS INITIATIVE

```
class agent_base #(type CFG = agent_drv_base,
                   type VIF = agent_vif,
                   type TXN = agent_txn_base,
                   type ANL = agent_anl_base,
                   type MON = agent_mon_base,
                   type DRV = agent_drv_base) extends uvm_agent;
  // members
  CFG config;
  VIF virt_if;
  uvm_sequencer #(TXN) seqr;
  ANL analysis;
  MON monitor;
  DRV driver;

  …
endclass : agent_base
```

Using the Configurable Agent

DVCon 2013 Design & Verification Conference & Exhibition
Sponsored By:
accellera SYSTEMS INITIATIVE

```
typedef agent_base #(
  .CFG (myagent_cfg),
  .VIF (virtual myagent_if),
  .TXN (myagent_txn),
  .ANL (myagent_analysis),
  .MON (myagent_monitor),
  .DRV (myagent_driver)
) myagent;
```

```
class myagent extends agent_base #(
  .CFG (myagent_cfg),
  .VIF (virtual myagent_if),
  .TXN (myagent_txn),
  .ANL (myagent_analysis),
  .MON (myagent_monitor),
  .DRV (myagent_driver)
);
  …
endclass : myagent
```

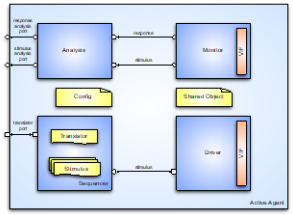
Pros & Cons

- Eliminates redundant, mundane code 
- Enforces a standard architecture for Agents 
- Code consistency 

- Debug is difficult 

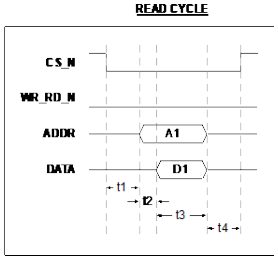
Three Methodology Enhancements

1. Parameterized Base Agent



2. Driver Level Data Rate Controls

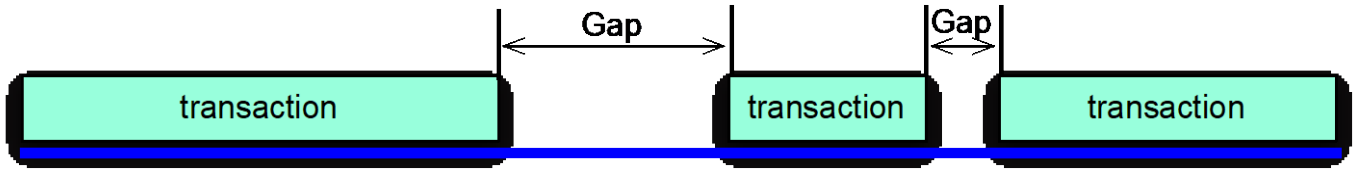
3. Interface Timing Controls



Driver Level Data Rate Controls

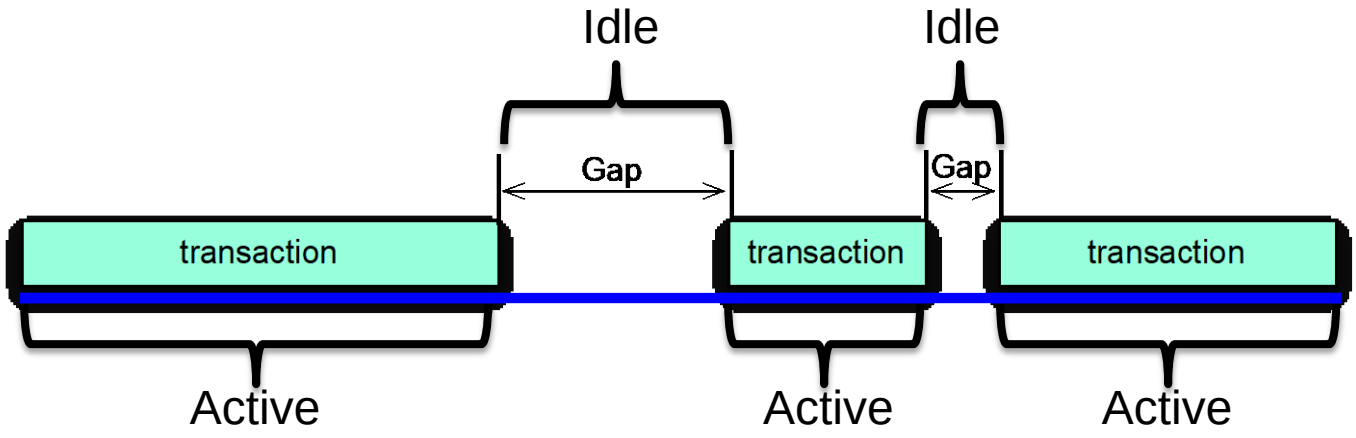
- Variable timing between transactions is usually desirable
- Aggregate throughput of transactions is still needed

- Use a throttle object to provide these capabilities
  - Instantiated in a driver
  - Controls gap using a Poisson distribution



Driver's Responsibilities

- The driver has three responsibilities when using a throttle object
  1. Initialization of the throttle object
  2. Informing the throttle object when idle
  3. Informing the throttle object when active



Using the Throttle Class

- Process sequence_items in the driver

```
task my_drv::process_item_nb();
  my_txn req_txn, rsp_txn;

  do begin
    seq_item_port.try_next_item(req_txn);
    if (req_txn == null) begin
      idle_cycle();
    end
  end while (req_txn == null);

  // wiggle pins
  active_cycle(req_txn, rsp_txn);
  seq_item_port.item_done();
  …
  seq_item_port.put(rsp_txn);
endtask : process_item_nb
```

Called when a sequence_item is not available

Called when data is available to process

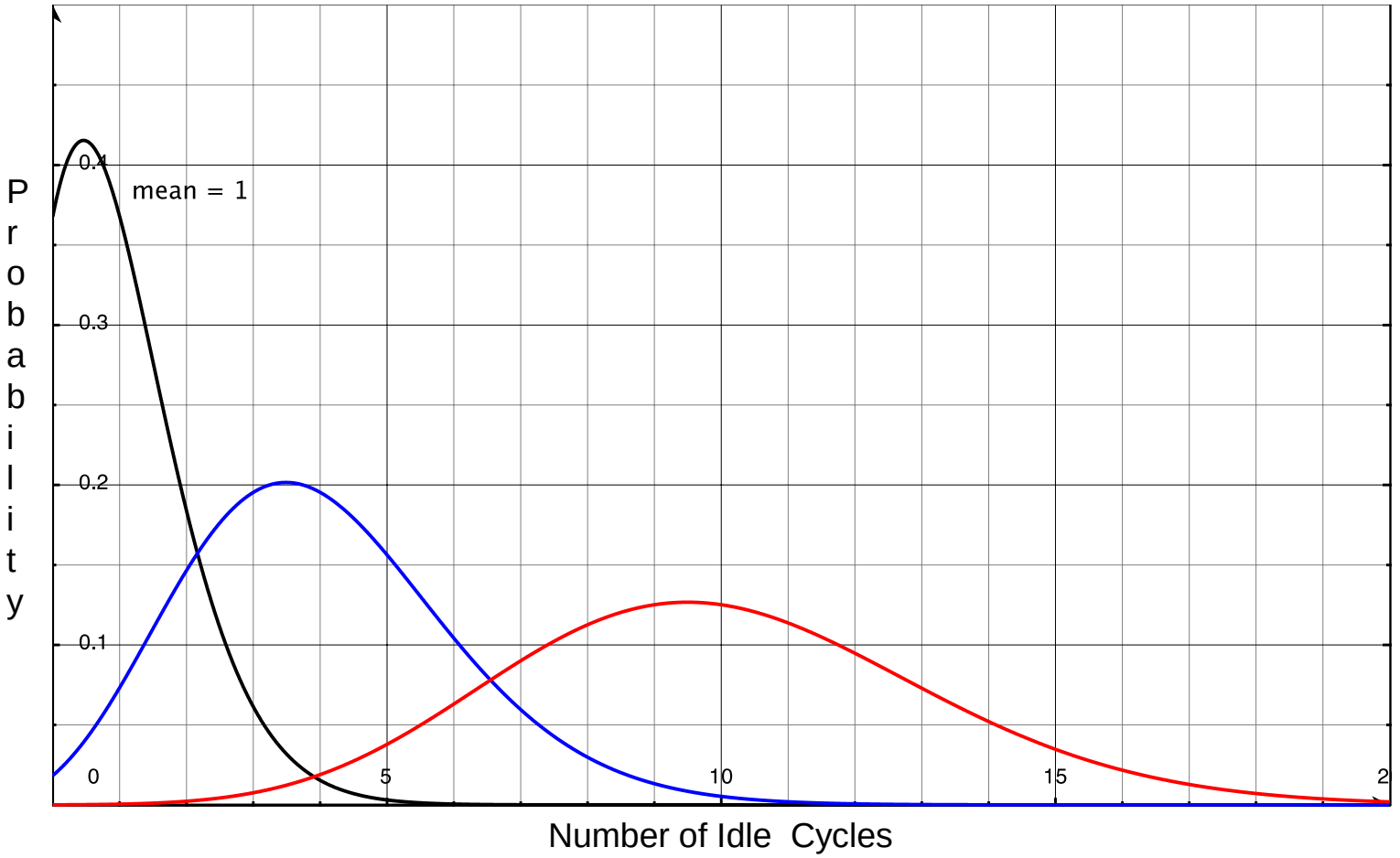
Throttle Class

DVCon 2013 Design & Verification Conference & Exhibition
Sponsored By:
accellera SYSTEMS INITIATIVE

```
function void throttle::idle(int count = 1);
  // record idle cycle(s)
  idles += count;
endfunction : idle
```

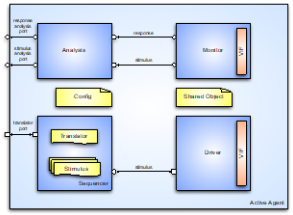
```
function int throttle::active(int count);
  int target, mean, cycles;

  // record active cycle(s)
  actives += count;

  // calculate the target number of active cycles
  target = throughput * (actives+idles) / 100;

  // calculate the mean of the Poisson distribution
  mean = (actives > target) ?(actives - target) : 1;

  // randomize the number of idle cycles needed
  return ($dist_poisson(seed, mean));
endfunction : active
```

Poisson Distributions



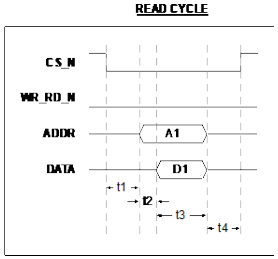
Three Methodology Enhancements

1. Parameterized Base Agent



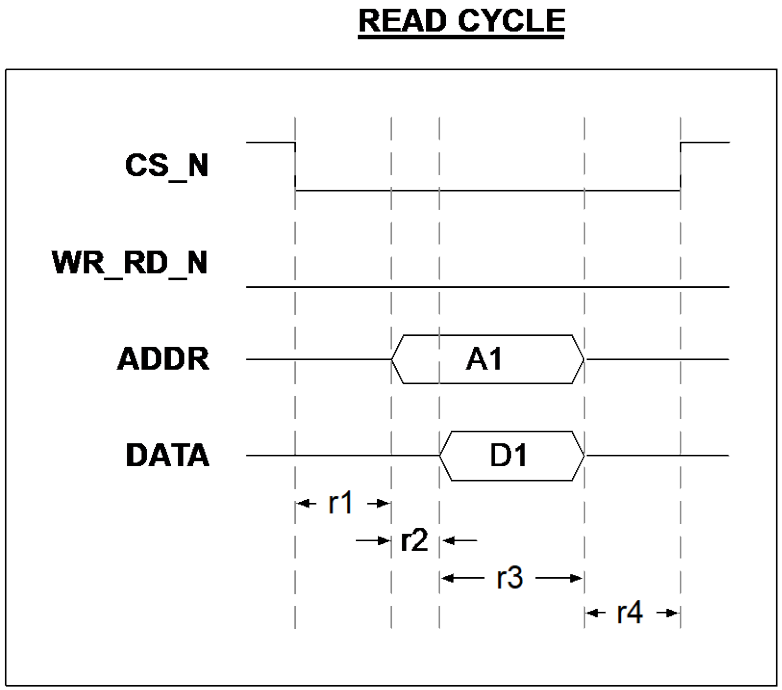
2. Driver Level Data Rate Controls

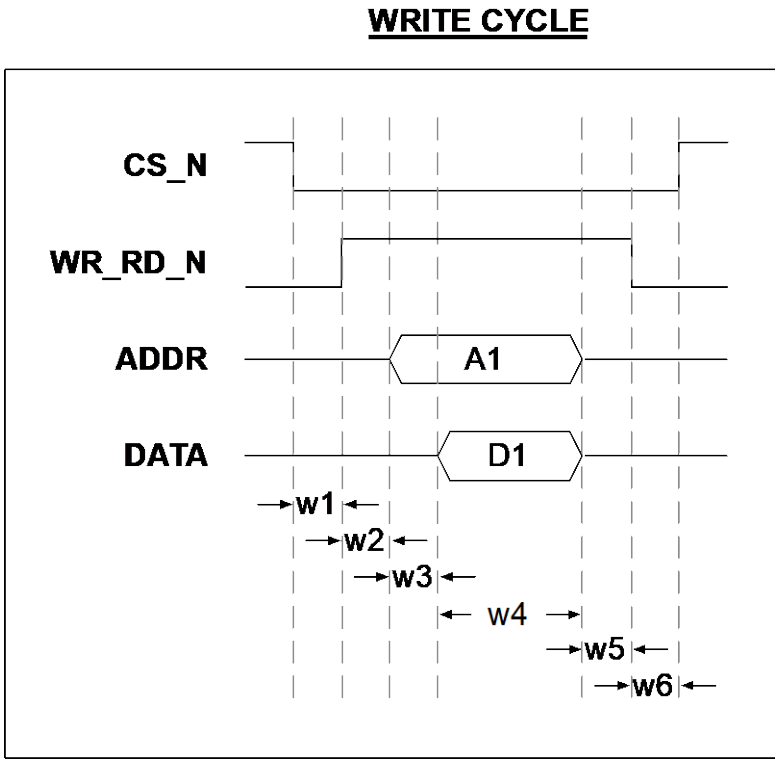
3. Interface Timing Controls



Interface Timing Controls

- Varying timing between interface signals can catch bugs



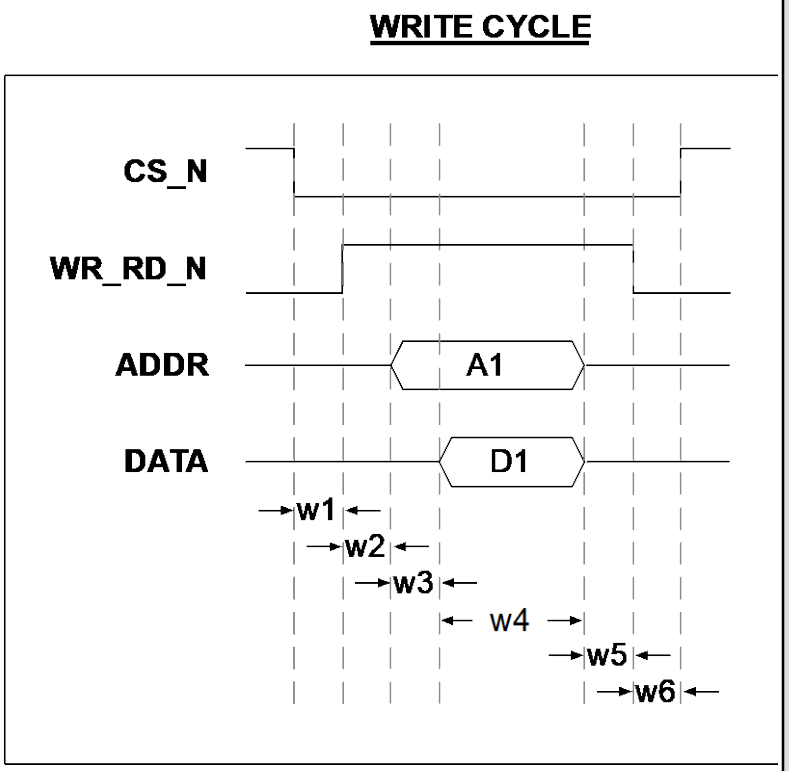


Interface Timing Class

- Encapsulate timing information in a class
  - Interface timing can be randomized at any interval in the simulation
  - Constraint-based timing definition
  - Weighted distributions generate interesting values
  - Driver code is highly succinct, readable and maintainable
  - Applies to both asynchronous and synchronous interfaces

Interface Timing Class: Timing Parameters

DVCon 2013 Design & Verification Conference & Exhibition
Sponsored By:
accellera SYSTEMS INITIATIVE




```
//=====================================
// write cycle timing specs, in ns
//=====================================
const int W1_MIN    = 100;
const int W1_MAX    = 200;

const int W2_MIN    = 150;
const int W2_MAX    = W2_MIN*2;

const int W3_FIXED  = 300;

const int W4_MAX    = 200;
const int W4_MIN    = W5_MAX/2;

…

//=====================================
// timing specs, randomized each cycle
//=====================================

rand int w1, w2, w3, w4, …;
```

Interface Timing Class: Timing Constraints

DVCon 2013 Design & Verification Conference & Exhibition
Sponsored By:
accellera SYSTEMS INITIATIVE

```
//=============================================
// write cycle timing constraints
//=============================================
constraint w1_range { w1 inside {[W1_MIN : W1_MAX]}; };
constraint w2_range { w2 inside {[W2_MIN : W2_MAX]}; };
constraint w3_fixed { w3 ==  W3_FIXED; };
constraint w4_range { w4 inside {[W4_MIN : W4_MAX]}; };

constraint w1_w2_phase  { w1 < w2; };
…
```

Interface Timing Class: Timing Distributions

DVCon 2013 Design & Verification Conference & Exhibition
Sponsored By:
accellera SYSTEMS INITIATIVE

```
//=============================================
// write cycle timing distribution
//=============================================
constraint sc_w1_dist  { w1 dist {W1_MIN := 10,
                                  [W1_MIN : W1_MAX] :/ 80,
                                  W1_MAX := 10 }; }
constraint sc_w2_dist  { w2 dist {W2_MIN := 10,
                                  [W2_MIN : W2_MAX] :/ 80,
                                  W2_MAX := 10 }; }
// w3 is fixed, no dist necessary
constraint sc_w4_dist  { w4 dist {W4_MIN := 10,
                                  [W4_MIN : W4_MAX] :/ 80,
                                  W4_MAX := 10 }; }
```

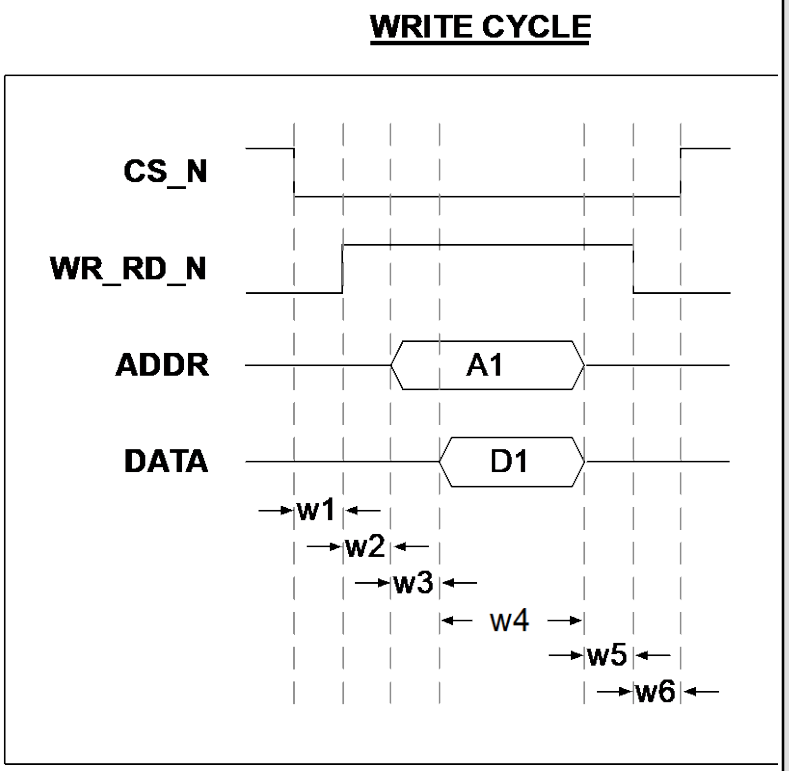
Interface Timing Class: Covergroup

DVCon 2013 Design & Verification Conference & Exhibition
Sponsored By:
accellera SYSTEMS INITIATIVE

```
//=============================================
// write cycle timing coverage
//=============================================
covergroup wr_cvg();
  w1_cvp: coverpoint w1 {
   bins min             = {W1_MIN};
   bins range[RNG_BINS] = {[W1_MIN : W1_MAX]};
   bins max             = {W1_MAX}; }
  w2_cvp: coverpoint w2 {
   bins min             = {W2_MIN};
   bins range[RNG_BINS] = {[W2_MIN : W2_MAX]};
   bins max             = {W2_MAX}; }
  // no cvp for w3, it is fixed
  w4_cvp: coverpoint w4 {
   bins min             = {W4_MIN};
   bins range[RNG_BINS] = {[W4_MIN : W4_MAX]};
   bins max             = {W4_MAX}; }
  …
endgroup: wr_cvg
```

Interface Timing Class: Driver Code

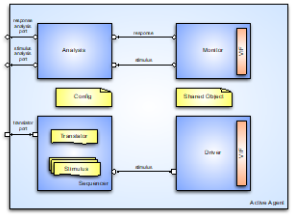
DVCon 2013 Design & Verification Conference & Exhibition
Sponsored By:
accellera SYSTEMS INITIATIVE




```
task drv::wr_cycle();
  //Randomize timing for this cycle
  if(!timing.randomize()) `uvm_error(…)

  //initiate cycle
  vif.cs_n = 'b0;
  #(timing.w1);
  vif.wr_rd_n = 'b1;
  #(timing.w2);

  //send addr and data
  vif.addr = req_txn.addr;
  #(timing.w3);
  vif.data = req_txn.data;
  #(timing.w4);
  …
  //collect coverage
  timing.wr_cvg.sample();
endtask :wr_cycle
```

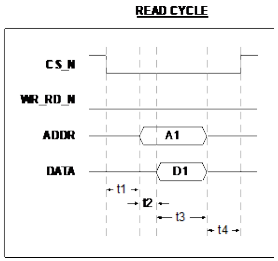
Three Methodology Enhancements

1. Parameterized Base Agent



2. Driver Level Data Rate Controls

3. Interface Timing Controls



Acknowledgments

- Michael Donnelly – Verification Engineer – Lockheed Martin
- Doug Krening – Verification Consultant
- Geoff Koch – Technical Writer - Mentor Graphics

26 of