2023

DESIGN AND VERIFICATION™

DVCON

CONFERENCE AND EXHIBITION

UNITED STATES

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

Demystifying Formal Testbenches: Tips, Tricks, and Recommendations

MARVELL Dr. Shahid Ikram, Distinguished Engineer, Marvell Semi

SIEMENS Mark Eslinger, Product Engineer, Siemens

accellera SYSTEMS INITIATIVE

Introduction

- Formal Test plans
- Simulation’s Testbench Vs. Formal Testbench
- Components of Formal Testbench
- Advance Topics
- Architecting a Formal Testbench
- Functional Completeness
- Putting it All together

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

Planning

“By failing to prepare, you are preparing to fail.” Benjamin Franklin

Formal Testplanning

- Formal testplanning is important to the success of property checking
- Follow the 7-step flow outline as part of formal testplanning
  - Identify, describe, interface, requirements, properties, strategy, coverage
- Overall project testplanning includes formal and simulation
  - Decide what verification strategy will be applied to what parts of the design
  - Create a written testplan that the formal results will be tracked by

Item	Type	Description	Check	Status
req_eventually	assume / cover	Eventually req each port	s_eventually pend[i] && req[i]	covered
reqs_granted	cover	All reqs granted	req[i] |=> s_eventually gnt[i]	covered
gnts_unique	check / cover	Only 1 gnt active	onehot0(gnt) / cover gnt[i]	passed / covered

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

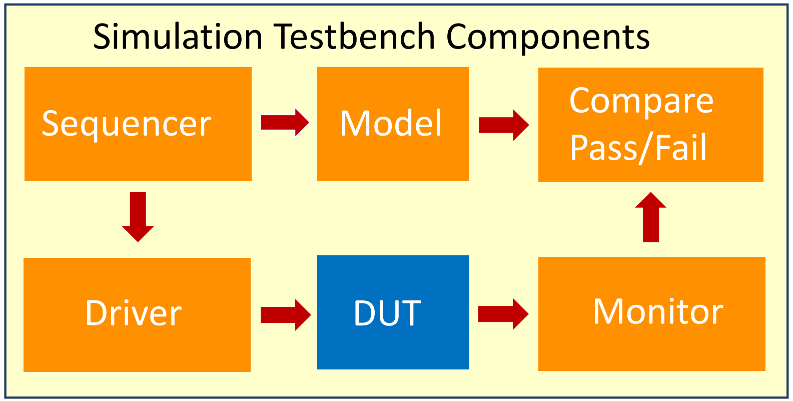
What is different?

A zoom out before a zoom in

accellera
SYSTEMS INITIATIVE

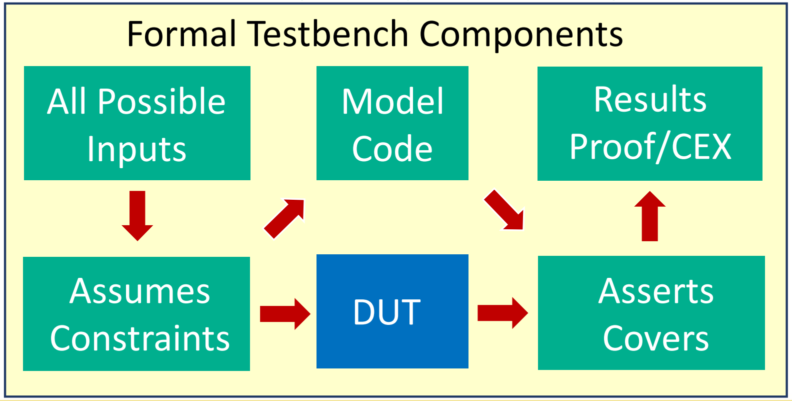
Simulation / Formal Testbench Components

- Simulation and formal testbenches are similar in nature conceptually
- Verification requires 2 models – one of which is the DUT
  - The 2nd model in formal is the modeling code and properties
  - In simulation vectors are driven, in formal the full input space is explored


Simulation Testbench Components

Sequencer → Model → Compare Pass/Fail

Sequencer ↓ Driver → DUT → Monitor ↑ Compare Pass/Fail


Formal Testbench Components

All Possible Inputs ↓ Assumes Constraints → DUT → Asserts Covers ↑ Results Proof/CEX

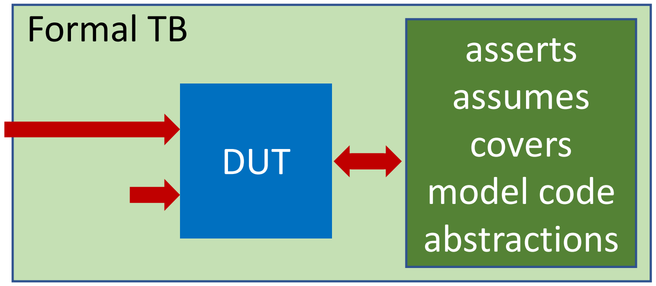
Assumes Constraints ↗ Model Code ↘ Asserts Covers

Comparing Formal / Simulation Testbenches

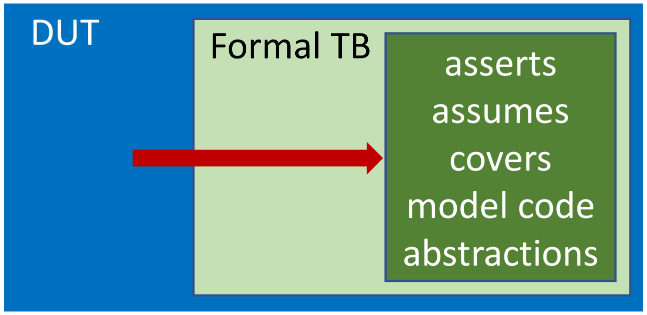
Component	Formal	Simulation
Design RTL (SV/Verilog/VHDL)	Only synthesizable	Synthesizable and behavioral
Properties and coverage	Required	Recommended
Inputs - Vectors	All possible inputs explored by tool	Directed, constrained random, …
Input constraints	Assumptions remove illegal inputs	Part of driver - transactors
Second Model	Assertions and modeling code	UVM, C, …
Results	Assertions proven or have CEX	Design compared to model – pass/fail
Implementation	TB bound to DUT / DUT instance in TB	DUT instantiated in TB
Ports	TB monitor design signals	TB drives inputs and monitors outputs

Formal Testbench Implementations

- Wrapper around the DUT
  - Can be a closed system, only clk/rst as inputs
    - Formal can drive DUT undriven inputs
    - Other signals as inputs OK to help with setup



- Bound with the DUT
  - Often preferred (can be used in sim)
  - All DUT signals can be inputs to formal TB
  - Can bind more to internal modules/instances



Tip: Use the same names in the FTB as the design for simpler binding

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

What makes a Formal Test Bench?

“The whole is Greater than the sum of its parts” Aristotle

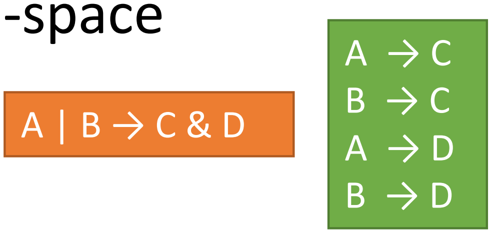
accellera
SYSTEMS INITIATIVE

Formal Testbench Components

- Clocks
  - Formal is cycle based, multi-clock designs require clock definitions
- Initialization sequence
  - Proper design initialization important to formal results
- Properties – required
  - assert / assume / cover / cover bin
- Modeling code
- Abstractions

Properties - assert

- Checks of Design’s behavior
- Adds complexity to verification state-space
  - Keep as simple as possible
    - Divide and conquer, Case analysis
  - Keep as sequentially short as possible
    - Keep it precise by using $rose, $fell
- Meaningful naming
  - Group by names
- Must be checked in Simulation/Emulation at Block and SOC levels
- Back annotation to the Test plan



Tip: Decompose complex properties into a set of simple properties
Trick: Use triggers (e.g. $rose) in antecedent to minimize CEXs

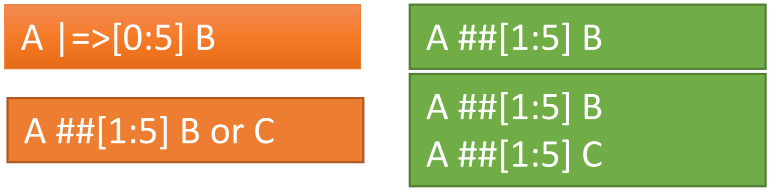
Properties - assume

- Used to restrict the state space
  - Keep them simple
- Global vs. Local
  - Assumptions are global unless restricted through tasks or oracles
- Add assumptions progressively
  - Start with no assumptions
  - Keep these to minimum
- Must be checked in Simulation/Emulation at block/SOC level as assertions
- Back annotation to the Formal Test plan

Recommendation: Use formal VIP to constrain bus interfaces
Trick: Sometimes it is easier to write an assumption on an internal signal

Properties - cover

- Existential Checks of design behavior
- Critical to the Formal Analysis
  - Tendency to over-constrain
- Keep them simple
- No implication only sequence
- Keep them separate
- Follow a strict naming convention
- Back annotation to the Test plan



A |=>[0:5] B

A ##[1:5] B or C

A ##[1:5] B

A ##[1:5] B
A ##[1:5] C

Modeling Code

Modeling code simplifies signals and writing of properties

- Makes properties easier to read and understand
- Often easier to implement that trying to check everything in a property

```
// Requirement: Never > 5 outstanding wr’s (without a rd) and no rd before wr
reg [2:0] my_cnt;
always @(posedge clk or negedge rstn)
if (!rstn)                            my_cnt <= 3’b000;
else      if      ( wr && !rd)  my_cnt <= my_cnt + 1;
          else if (!wr &&  rd)  my_cnt <= my_cnt – 1;
          else                  my_cnt <= my_cnt;
```

```
a_wr_outstanding_le5: assert property (@(posedge clk)    my_cnt <= 3’d5         );
a_no_rd_without_wr:   assert property (@(posedge clk) !((my_cnt == 3’d0) && rd));
```

Recommendation: Use exact/minimal bit widths when defining signals
Trick: Use hierarchical references to access signals in sub hierarchy

Abstractions

Abstractions are all about state space reduction

- Parameter reduction
- Constants
- Blackboxing
- Cutpoints
- Initial value
- Counter/memory/arithmetic

Tip: Use parameters in FTB to match the DUT

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

Tricks of the Trade

“Some people are so busy learning the tricks of the trade that they never learn the trade.” Vernon Law

accellera
SYSTEMS INITIATIVE

Advanced Topics

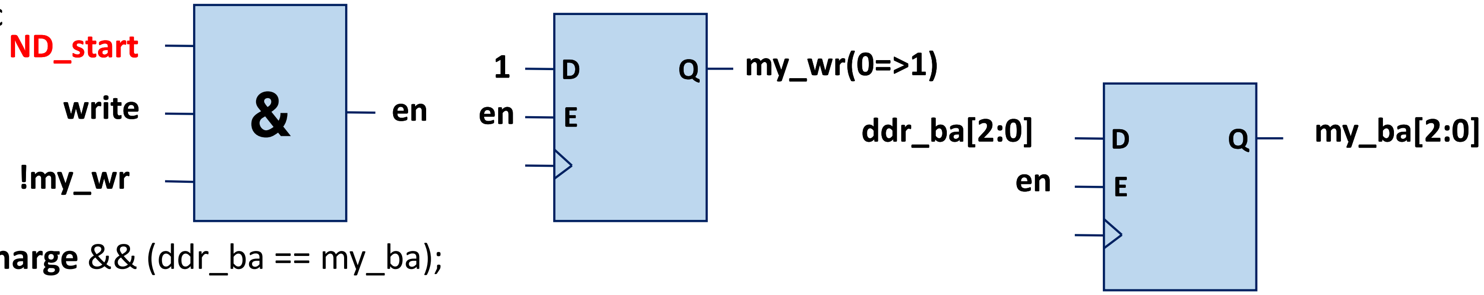
- Non-Determinism(ND)
  - Let formal explore all possibilities, temporally and spatially
- Data Independence(DI)
  - When the datapath is independent of the control logic, can use 1 data bit
- Symbolic Variables
  - Hold value stable during formal run, formal analyzes all possible values
- Phantom Wires
  - Make use of antecedent to constrain values formal can drive

Using Modeling Code and ND for Bug Hunting

DDR requirement: No precharge to same addr as write within 11 cycles

```
// Simplify DDR signals
parameter PRECHARGE = 7'b11_0010_0;                parameter WRITE = 6'b11_0100;
reg pre_cke; always @(posedge ddrclk) pre_cke <= ddr_cke;
wire [6:0] ddr_cmd = {pre_cke,ddr_cke,ddr_cs,ddr_ras,ddr_cas,ddr_we,ddr_addr[10]};
wire precharge = (ddr_cmd == PRECHARGE);           wire write = (ddr_cmd[6:1] == WRITE);
```

// modeling code logic



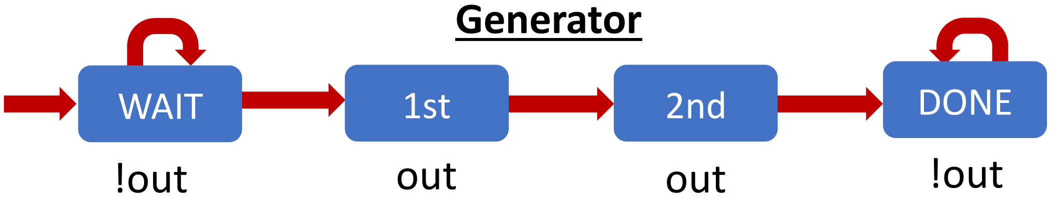
```
wire same_pre = precharge && (ddr_ba == my_ba);

a_wr_to_pre_bug: assert property (@(posedge ddr_clk) $rose(my_wr) |-> (!same_pre)[*11] );
```

Tip: Using ND in bug hunting allows formal the flexibility to find the corner case bug

Data Integrity

- Data Integrity classically makes use of ND and DI
    - ND input: start, DI is lsb: dati[0]
- Generator drives: …00011000…

start



Generator

WAIT

!out

1st

out

2nd

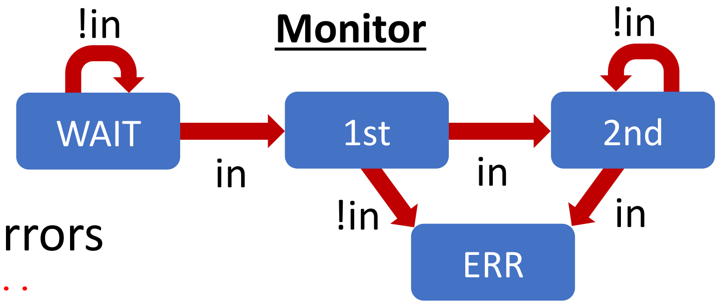
out

DONE

!out

```
m_di0: assume property (@(posedge clk) dati[0] == out );
```

- Monitor checks: …00011000… (in == dato[0])



Monitor

!in

WAIT

in

1st

in

!in

2nd

!in

in

ERR

```
a_no_err: assert property (@(posedge clk) cstate != ERR );
```

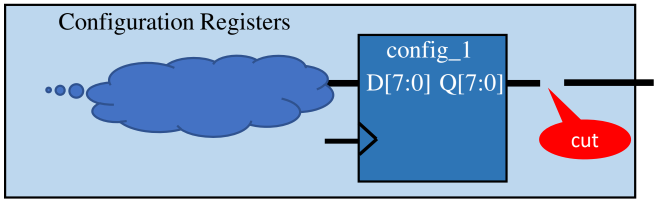
- Catches dropped, duplicated/added, reordered errors

  ..010.. ; ..0011100.. ; ..0010100..

Tip: The principles used here have many applications, only limit is your imagination

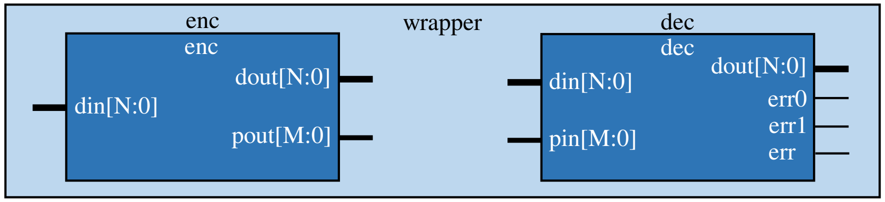
Symbolic Variables and Phantom Wires

- Symbolic Variables
  - Application: Configuration register



```
m_config_1_lo_val:  assume property (@(posedge clk) config_1[3:0] <= 4’h4 );
m_config_1_hi_1hot: assume property (@(posedge clk) $onehot(config_1[7:4]) );
m_config_1_stable:  assume property (@(posedge clk) $stable(config_1) );
```

- Phantom Wires
  - Application: ECC verification



```
// no error
a_0_err_dat:   assert property (@(posedge clk) $countones({(enc.dout^dec.din),(enc.pout^dec.pin)}) == 0 |-> dec.dout == enc.din );
// 1 error (detect and correct)
a_1_err_dat:  assert property (@(posedge clk) $countones({(enc.dout^dec.din),(enc.pout^dec.pin)}) == 1 |-> dec.dout == enc.din );
// 2 errors (detect only, only check error outputs)
a_2_err_err:  assert property (@(posedge clk) $countones({(enc.dout^dec.din),(enc.pout^dec.pin)}) == 2 |-> dec.err );
```

Tip: Symbolic variables can be used on inputs or internal nodes with a cutpoint

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

Architecting a Formal TB

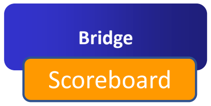
"We shape our buildings: thereafter they shape us."

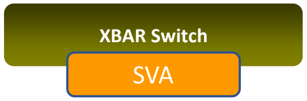
-Winston Churchill

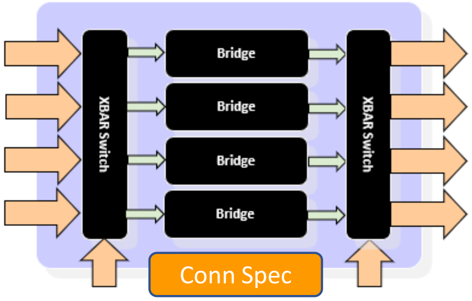
accellera
SYSTEMS INITIATIVE

Formal Testbench Architectures

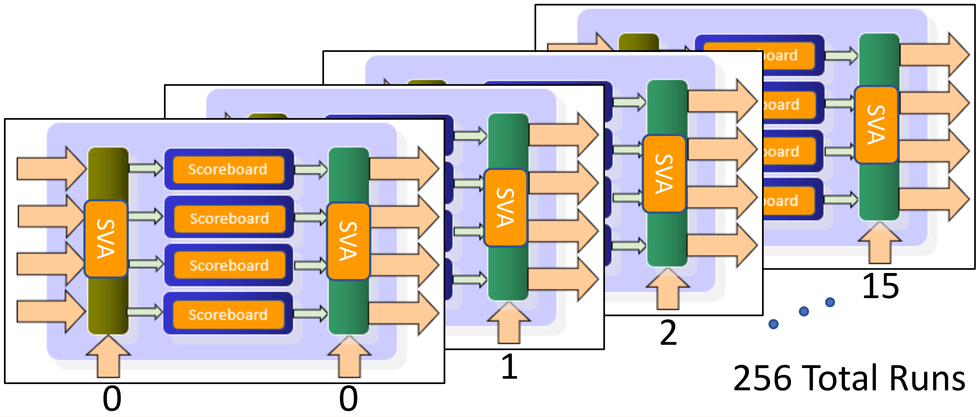
- Divide and Conquer
  - When state space is large
  - Verify each component







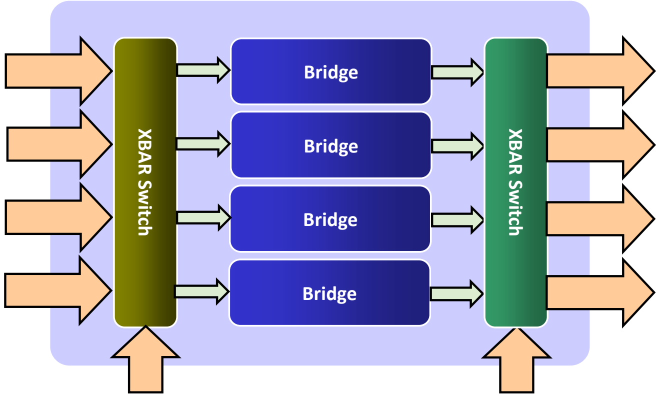
- Brute Force
  - Hard constraints
  - Can parallelize runs



Formal Testbench Architectures - Elegant

Advanced formal techniques allow you to simplify the formal TB

- ND, DI, Symbolic Variables, Formal VIP, modeling code => minimize state

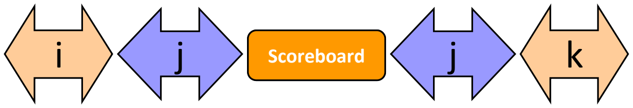


256 combinations
Selects stable during transmission

Data integrity end to end

- Symbolic Variables for input/bridge/output
  - Stable - Determines select value
  - ND – formal picks the path
- Proof – all scenarios good, CEX shows bad path

```
Input  i 0 to 3
Bridge j 0 to 3
Output k 0 to 3
```



selA[j] = i   selB[k] = j

Recommendation: Use the advanced techniques available to minimize state in your FTB

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION

UNITED STATES

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

Are we there yet?

“It always seems impossible until it's done.”

Nelson Mandela

accellera SYSTEMS INITIATIVE

Proof of Completeness

- A subjective target:
  - A Formal Test-plan as a contract among stake holders
- Checkers completeness
  - Full proofs
  - Partial/bounded proofs and design depth
  - Random fault insertions
- Coverage
  - Formal Coverage
  - Formal and Simulation coverage

Putting It All Together

- Proper testplanning is important to ensure success
  - A coverage strategy that is tied back to the testplan is important
- Decide on your formal TB structure
  - Make use of as many techniques as makes sense for what you are checking
  - Each formal TB will be unique based on who is creating it (just like sim!)
- Start where you are and expand from there as you gain experience
- Discuss and share with colleagues your experiences
  - Continue to learn and expand your awareness of these techniques

Questions?