2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Ironic But Effective: How Formal Analysis Can Perfect Your Simulation Constraints

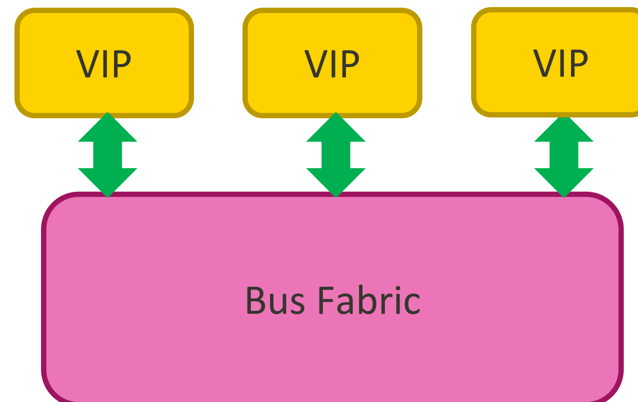
Penny Yang, Jin Hou, Yuya Kao,
Nan-Sheng Huang, Ping Yeung, Joe Hupcey

MEDIATEK

Mentor Graphics®

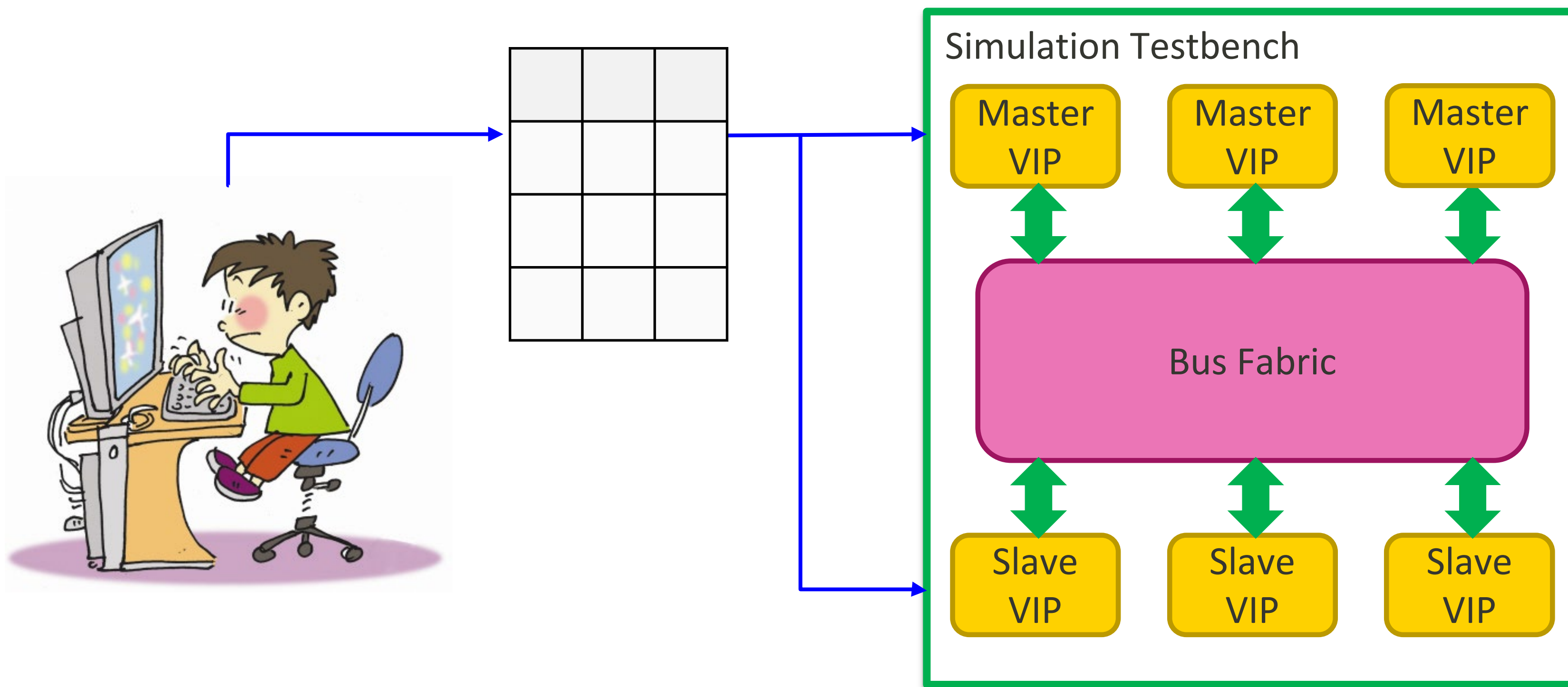
Introduction

- As SoC size increases, bus fabric design becomes more complicated and verification IPs (VIP) are needed in constrained random simulations.
  - Isolate the issues
  - Before designs are ready
  - Flexible and faster



Introduction (Cont.)

- These VIPs are usually constrained according to the information provided by DUT owners.

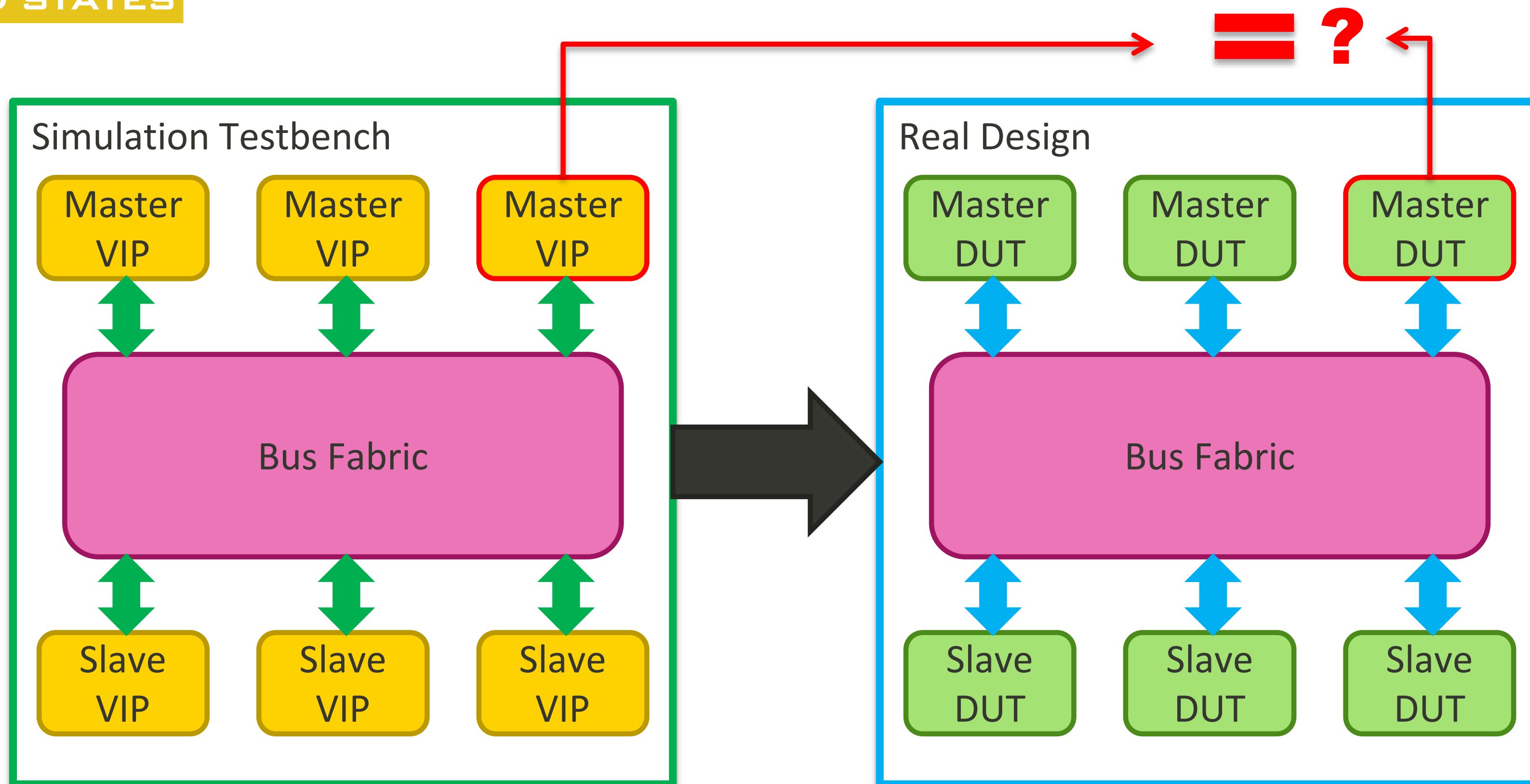


DESIGN AND VERIFICATION™ DVCON 2017 CONFERENCE AND EXHIBITION UNITED STATES

Introduction (Cont.)

- VIPs should behave the same as their substituted design modules. If not the same, the outcomes are:
  - VIP has **less** functionalities than DUT
    - the VIP over-constrains the environment of the bus fabric, and may hide extra functionalities and bugs in the bus fabric
  - VIP has **more** functionalities than DUT
    - the VIP under-constrains the environment of the bus fabric, resulting in false firings in simulation runs that only waste simulation and debug time

Introduction (Cont.)



- How to verify the correctness of the configuration information for VIPs?

Methodology

- Collect VIP configuration specifications from block designers and create a table for each VIP in our database.
- Code the configuration specifications using SVA and bind them to the design modules
- After the RTL design is ready, setup formal verification environment using scripts.
- Run SVA against the substituted design modules using formal tool
- Analyze results

Collect VIP Configuration Spec

- For each VIP used in the simulation environment, list its configurations in a table.

No.	AXI Capability	Property
1	Read/Write	cover
2	Burst Length	cover
3	Burst Size	cover
4	Burst Type	cover
5	Burst cross 256B	assert
6	Burst cross 4KB	assert
7	LOCK Access	cover
8	Exclusive Access	cover
9	AxID permitted value	cover
10	Address aligned to transfer size	assert
11	Write Outstanding #	cover
12	Read Outstanding #	cover
13	Wstrb all ones	cover
14	Write Data Interleaving	cover
15	Write-data-before-addr	cover

No.	AHB Capability	Property
1	Read/Write	cover
2	AHB Transfer Size	cover
3	AHB Transfer Type	cover
4	AHB Burst Type	cover
5	AHB Burst cross 1KB boundary	assert
6	AHB Early Burst Termination	cover
7	AHB Split Transfer Support	cover
8	AHB Retry Transfer Support	cover
10	Address aligned	assert
11	Insert BUSY cycles	cover

For legal behavior, we use cover property to check if it does exist.
For illegal behavior or something always true, we use assert property for it.

Code SVA

- Examples
  - AXI burst length: AXI burst length can be from 1 to 16 defined by *awlen* from 0 to 15

```
default clocking @(posedge clock); endclocking
default disable iff (~reset);
generate
  for (genvar i=0; i<16; i++) begin
    Cover_burst_size: cover property (awvalid && awlen==i);
  end
endgenerate
```

Code SVA (Cont.)

– Address aligned:

```
Assert_addr_aligned: assert property (
             htrans==NSEQ |-> addr_align);
```

– The SVA assertions are in separate files and connected to the design modules using SVA bind command

```
module AXI_assertions (clock, resetn,…);
…
endmodule
bind AXI_design1 AXI_assertions AXI_inst
(.clock(aclk),
 .resetn(aresetn),
 …
);
```

Create Formal Script

- Example of Makefile to run Questa Propcheck
- Automatically generate Makefiles using perl script

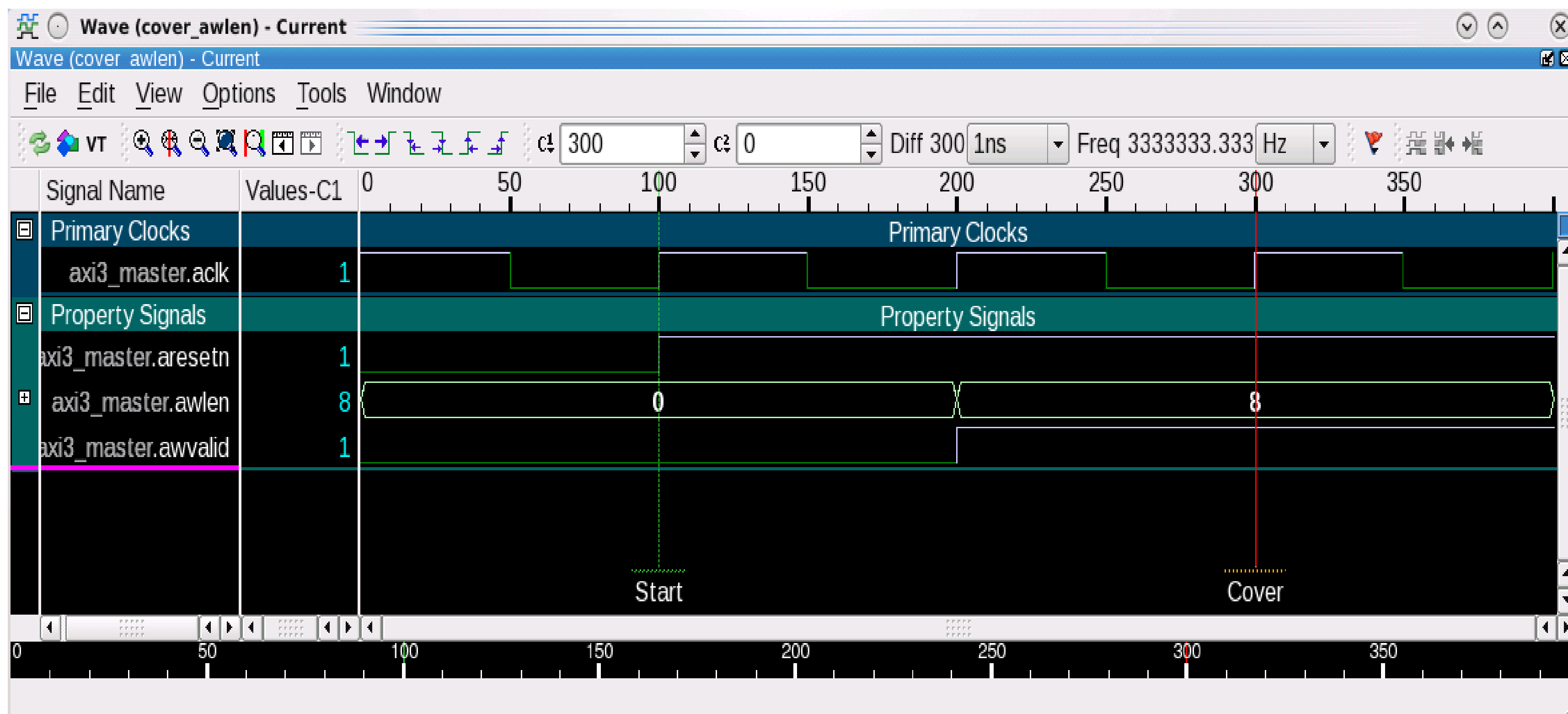
```
Run: compile formal
compile:
        vlib work
        vlog -f filelist
formal:
        qformal -c -od log -do " \
        do directives.tcl; \
        formal compile -d dut; \
        formal verify -init init_file; \
        exit"
```

Other Questa formal commands defining clock, reset, constant pins and etc.

Initial sequence

Waveform

- GUI can show counterexamples of fired properties and the sequences of covered properties.
  - Example: The burst length (awlen==8) is covered



Result Analysis

- Create a table for each VIP and its substituted design module to compare the configurations
  - First row lists all configuration types, and the first column lists all design module names.
  - Values outside of brackets are the VIP configurations, but actual design functions are inside the brackets

module	R/W	Burst Length	Burst Size	Burst Type	Burst cross 4KB	LOCK Access	Exclusive Access	AxID permitted value	Address aligned	Write Outstanding	Read Outstanding	Write Data Interleaving	Write-data-before-addr
A	ALL	ALL (R:!2~16)	8/16/32/64	ALL(INCR)	N	N	0	252/254 (252)	N(W:Y)	1	1	1	N
B	R	1~8(R:16)	64	INCR	N(R:Y)	N	0	0	N(Y)	32(0)	32(0)	16(1)	N

Example: The item of module A's burst type is 'ALL(INCR)', which means that the VIP is set to support all burst types, but module A only supports type INCR. Formal can prove that other types are uncoverable in the design, but simulation cannot.

Result Analysis (Cont.)

– Green items are the ones DUT==VIP, pink items are DUT>VIP, yellow items are DUT<VIP, orange items are inconclusive

– Red letters are real errors confirmed by designers, green letters are mismatches that can be ignored

DUT > VIP
VIP's test item (DUT's capability)

DUT < VIP
VIP's test item (DUT's capability)

module	R/W	Burst Length	Burst Size	Burst Type	Burst cross 4KB	LOCK Access	Exclusive Access	AxID permitted value	Address aligned	Write Outstanding	Read Outstanding	Write Data Interleaving	Write-data-before-addr
A	ALL	ALL (R:!2~16)	8/16/32/64	ALL(INCR)	N	N	0	252/254 (252)	N(W:Y)	1	1	1	N
B	R	1~8(R:16)	64	INCR	N(R:Y)	N	0	0	N(Y)	32(0)	32(0)	16(1)	N
F	ALL	ALL	8/16/32/64 (R:64)	INCR	N	N	0	0~2(0)	N (R:Y)	4	4	1	N

DUT == VIP

Inconclusive

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2017

Results

- We applied this methodology on a smart phone project.
- Tested 17 configurations of AXI VIPs for 18 design modules.
- Tested 11 configurations of AHB VIPs for 9 design modules.

Protocol	Total tests	DUT == VIP		DUT > VIP		DUT < VIP		Inconclusive	
	#	#	%	#	%	#	%	#	%
AXI	306	205	67%	64 (11+53)	21% (4%+17%)	24 (6+18)	8% (2%+6%)	13	4%
AHB	99	75	76%	13 (3+10)	13% (3%+10%)	7 (6+1)	7% (6%+1%)	4	4%

Error False Alarm

Results (Cont.)

- Spent 3 days to implement the SVA assertions.
- Spent 2 days to setup the formal environment for 27 design modules which was 20x faster than simulation
- Run-time of 96% properties is less than 1 hour
- Spent 1 week to get all the results in the previous table.
- Found 26 real errors in the configurations of the VIPs that could produce incorrect verification results of the bus fabric.

Conclusion

- Formal verification is an efficient and effective way to verify the correctness of the constrained VIPs used in simulation environment.
  - No simulation testbenches are needed that can save a lot of time.
  - Formal can verify the situation of VIP<DUT or VIP>DUT.
  - Formal environment is easy to setup.
  - Formal post process debugging is easy.
- Formal verification really works to perfect your simulation constraints!

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Future Work

- Reduce false alarms
  - Apply correct constraints with AIP
- Reduce inconclusive properties
  - Follow tool vendor's guidelines
- Raise success rate
  - Prove dozens of DUTs at once
  - Bug Hunting first
- Together with simulation to make a robust design