February 25-28, 2013
DoubleTree, San Jose

accellera SYSTEMS INITIATIVE

XILINX
ALL PROGRAMMABLE™

How to Kill 4 Birds with 1 Stone: Using Formal Verification to Validate Legal Configurations, Find Design Bugs, and Improve Testbench and Software Specifications

Saurabh Shrivastava, Kavita Dangi, Darrow Chu, Mukesh Sharma

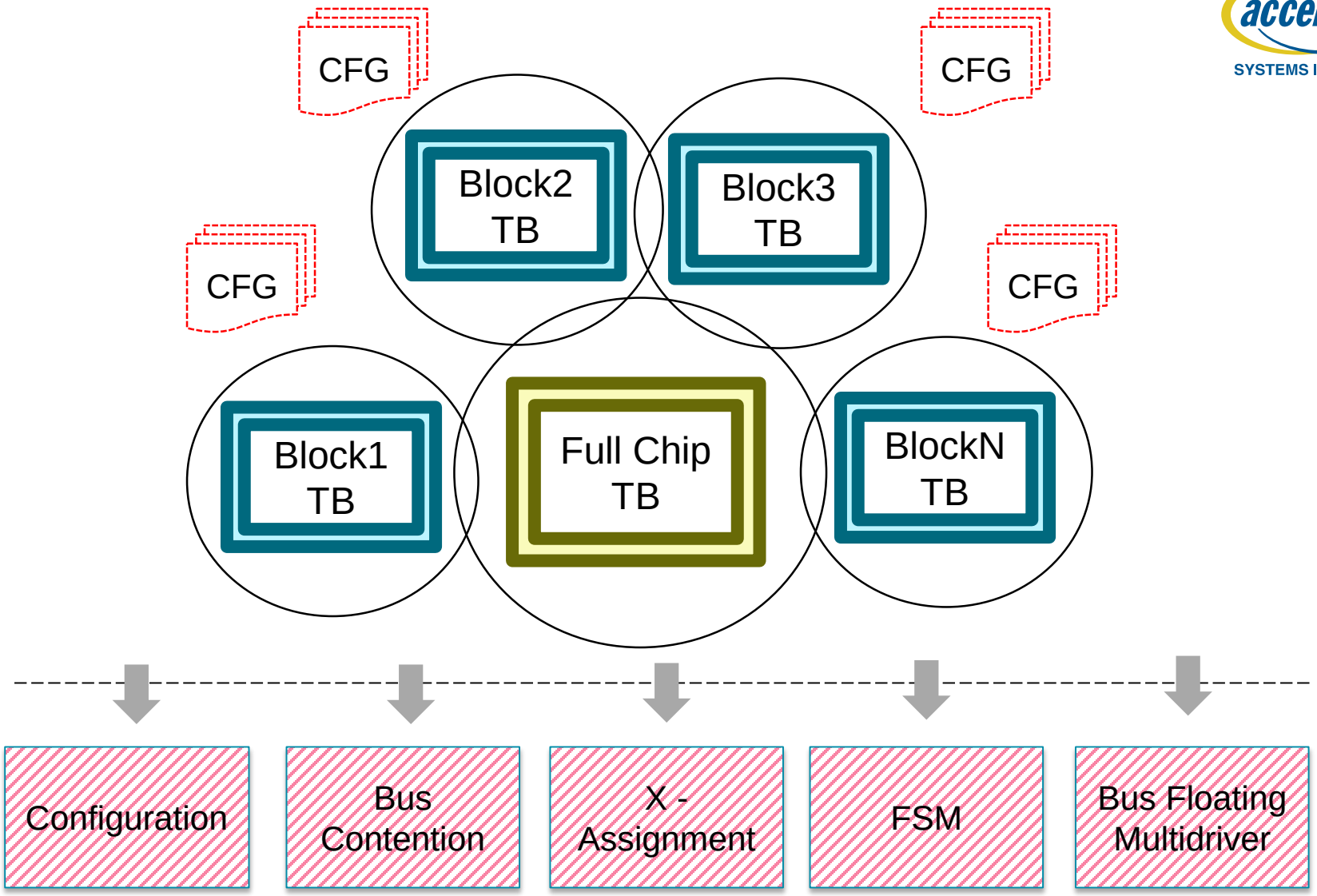
Motivation

- How to get a head start on verification
- How to augment simulation based verification
- How to validate/verify device configuration for
  - Better testbench simulations
  - Better software output
- How to weed out basic bugs:
  - Bus contention, x-assignments, floating-bus, FSM

```
Assign out = SEL[0] ? in0 : 1'bz;
Assign out = SEL[1] ? in1 : 1'bz;
```

```
Case(sel)
   0: out = in0;
   2: out = in1;
   default: out = 1'bx;
```

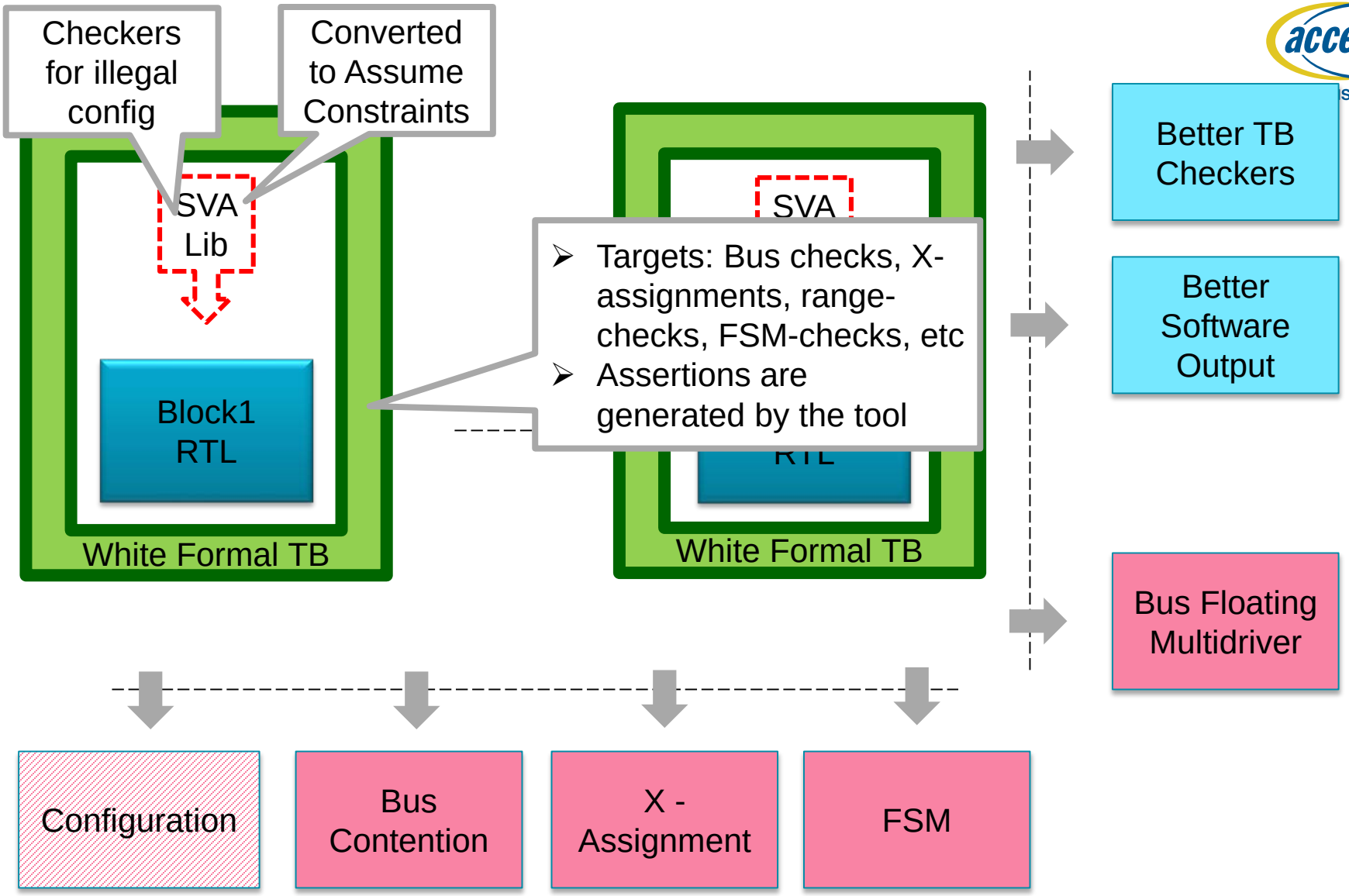
Older Approach



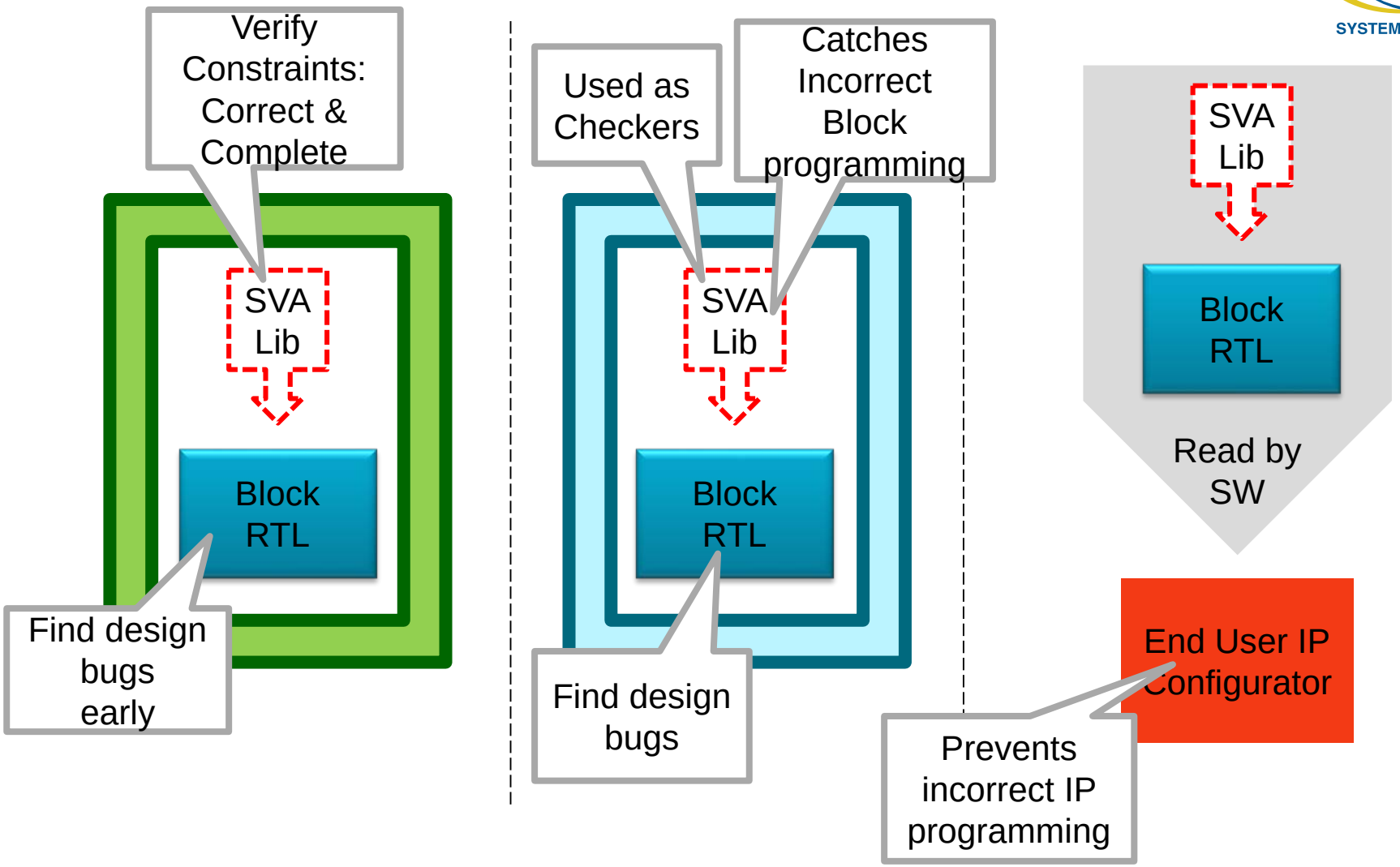
Limitations of Older Approach

- **Bugs are discovered late**
- **Block and Chip level simulations are as good as stimulus**
- **Difficult to hit all scenarios due to:**
  - Big configuration space (only handful are verified)
  - Big design
- **Interesting bugs found when test benches are sophisticated!**

New Approach



New Verification Flow



Assertion Library Example

fileA_assert_lib.sv

```
reg [3:0] varA;
reg [5:0] widthA;
reg modeA;
…
`LIB_ASSERT_ONE_HOT(22,”varA value
is not compliant”,varA)

`LIB_ASSERT_CHECK(23,”Incorect
width for modeA=0”,(modeA==0 &&
widthA==16))
```

assert_lib_define.svh

```
`define LIB_ASSERT_ONE_HOT(INST,MSG,a)
   `ifdef FORMAL \
      assume_one_hot_``INST``: assume property
(@(posedge clk) $onehot0(a)); \
   `else \
      assert_one_hot_``INST``: assert property
(@(posedge clk) $onehot0(a)) \
       else begin $error ("%s: one_hot fails
for a=%h\n",MSG,a); \
   `endif
```

run.log

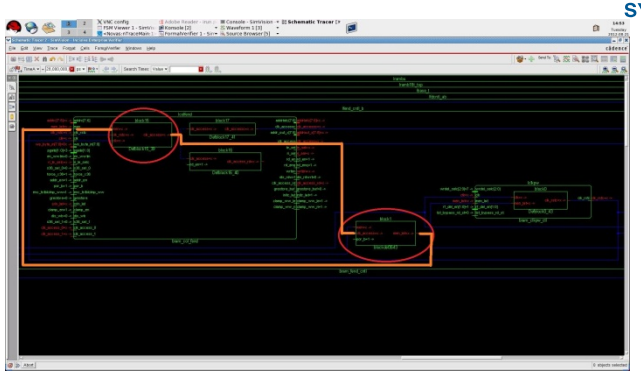
```
ncsim: *E,ASRTST (lib_assert_macros.sv,17):
(time 17 NS) Assertion
top.dut.modA.assert_one_hot_22 has failed
varA value is not compliant: one_hot fails
for a=b
```

Bugs Found

- Incorrect library assertion causing bus contention

```
Assign out = SEL[0] ? in0 : 1'bz;
Assign out = ~SEL[1] ? in1 : 1'bz;
Assign out = SEL[2] ? in2 : 1'bz;
Assign out = SEL[3] ? in3 : 1'bz;
`LIB_ASSERT_CHECK(sel,"Select values are incorrect",(SEL==0 || SEL==1 || SEL==6 || SEL==10)
```

- Combinatorial loops



- Range overflow
- Missing library assertions causing
  - bus contention
  - X-assignment failures
  - Bus floating failures
- State reachability and transition
- State deadlock issues
- Bus floating failures

Comparison: Old v/s New

	Old Approach	New Approach
Early bug discovery		
Exhaustive		
Setup time	> 1 Month	1-3 days
Setup effort		
Test coding		No assertion coding
Scope		Targeted
Configuration verification	Manual approach, limited	Does not verify anything out of cone of logic, but verifies ALL legal cfgs for properties
Tool capacity		
Tool issues		
Tool learning		

- **Only opportunistic!**

Future Work

- Multi-block verification
- Tool kit
- Enable designers to run the flow
- Gate level Verilog verification

Summary

- Finding early design bug and verifying configurations is an issue
- Simulation based approaches have limitations
- White Formal is used along with SVA library to solve this problem
- New approach finds early bugs and verifies configuration space
- As a result we get better testbench and better software output

White Formal

- X-assignment example

Older Approach

- Block and Chip level testbenches
- Directed & pseudo-random tests for:
  - bus contention, X-assignment , FSM-checks
- Static lint for bus floating and multi-driver
- Configuration verification
  - Manual
  - Directed simulations

Limitations of New Approach

- Only opportunistic!
- Works only for targeted checks
- Configurations outside cone of logic not verified
- Tool capacity: cannot be applied at chip level
- Tool issues
- Tool learning