February 25-28, 2013
DoubleTree, San Jose

accellera SYSTEMS INITIATIVE

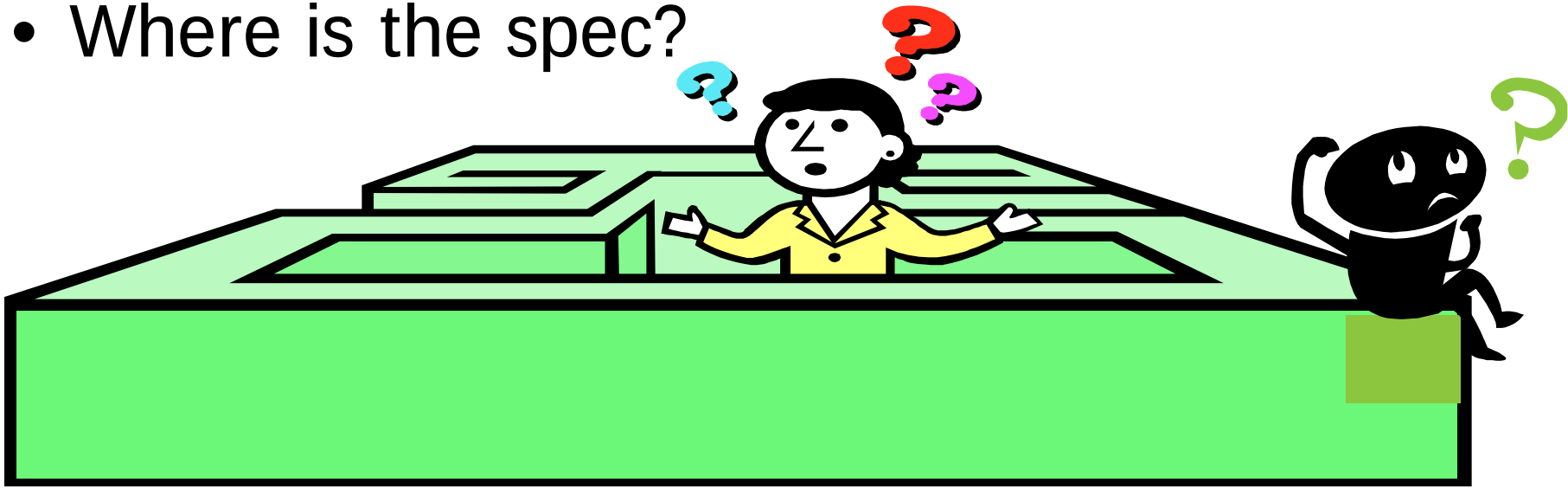
Who's Watching the Watchmen?

The Time has Come to Objectively Measure the Quality of Your Verification

by
David Brownell
Design Verification
Analog Devices Inc.

Verification is Hard

- Am I checking everything?
- Are my tests covering all the scenarios?
- Am I done?
- What did I forget?
- Where is the spec?



Verification Complexity Increasing

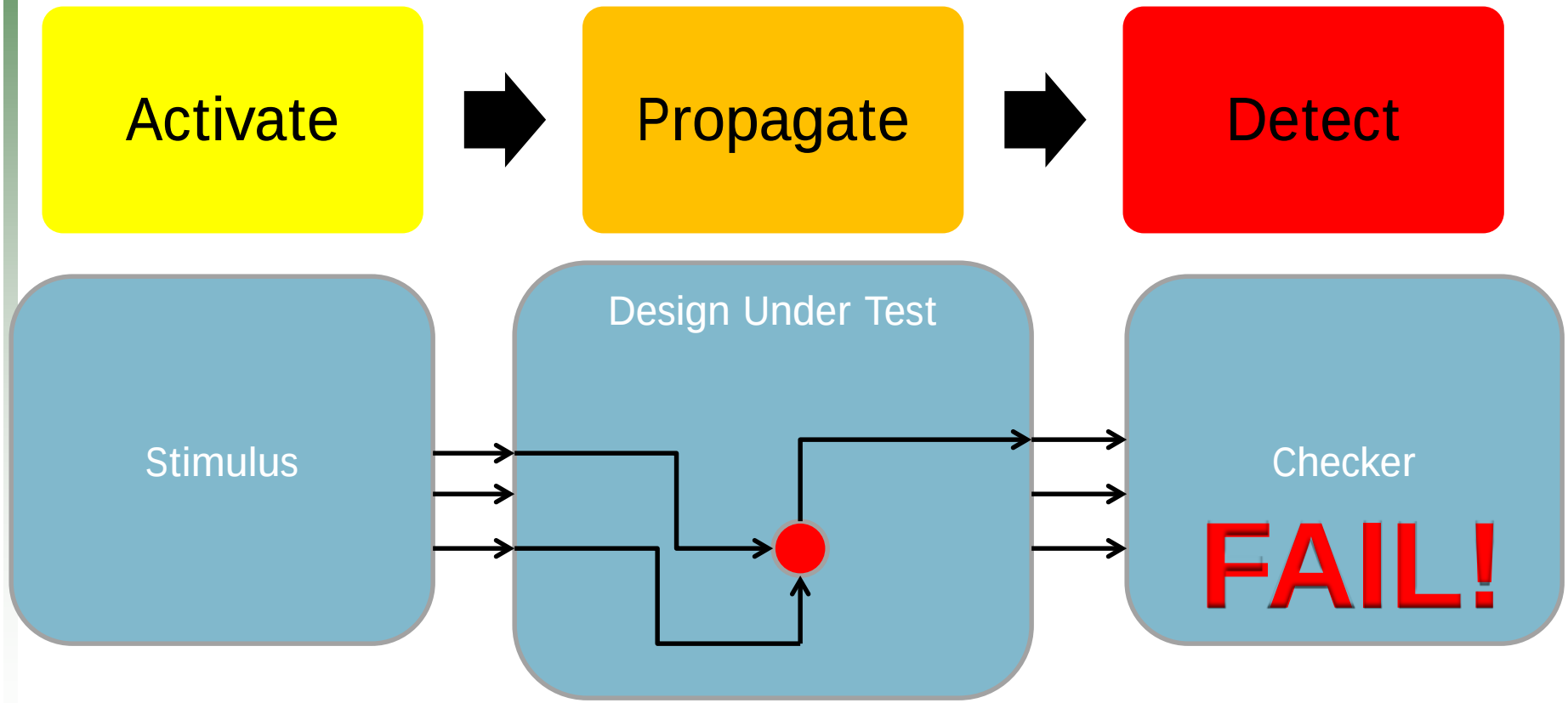
Block/Chip Name	RTL Line Count	Testbench Line Count
Peripheral Block	~9K	~14K
Processor Control Block	~15K	~22K
Memory Controller	~18K	~42K
Small SOC	~90K	~17K
Large SOC		**>250K**

Bugs very likely in Testbench code as well as RTL!

Bugs in Testbench can mask bugs in RTL!

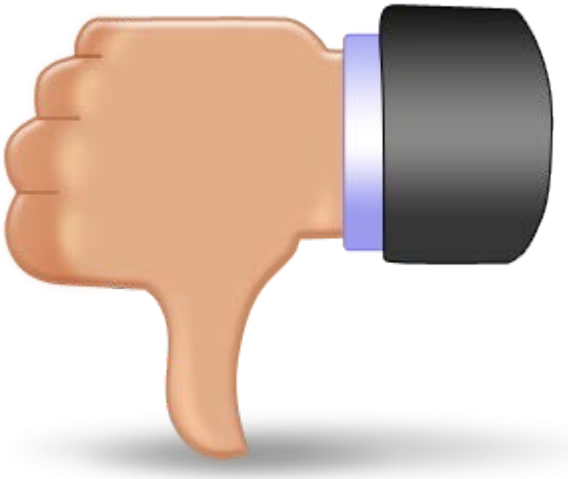
Effective Verification

For any bug that can exist in RTL the DV Environment must be able to:



Current Verification Metrics

- Code Coverage
- Functional Coverage
- Verification Plans



- Good but not good enough
  - Focused on Activation
  - No Information about Propagation or Detection
  - Ignore Testbench completely

Functional Qualification is the Answer

- Systematically insert artificial bugs into the RTL
- Run tests to see if fault is detected
- Provides metrics for every fault as to whether:
  - Fault is activated
  - Fault is propagated
  - Fault is detected

Functional Qualification is the Answer

- Identifies Holes/Weaknesses in DV Environment
  - Inadequate Tests
  - Bad or Missing Checkers
- Objective measure of overall DV quality
- Results and Experiences presented today based Springsoft/Synopsys Certitude

Example of Faults Inserted

Sponsored By:

Port Faults	Input : Stuck at 0, Stuck at 1, Negated Output : Stuck at 0, Stuck at 1, Negated
Condition Faults	Condition True, Condition False, Negated
Dead Faults	Dead Assign, Dead Else
Bus Faults	Flip First Bit, Flip Last Bit, Negate Bus,
Operator faults	Swap operators

[illegible]

```
[illegible]
   c  <= d;
Else
   c <= e;
```

Changed into:

```
[illegible]
   c  <= d;
Else
   c <= e;
```

Functional Qualification Phases

Model

- Parse RTL Files to determine faults to insert
- Search for unreachable faults
- Determine cones of influence
- Create Instrumented RTL Files for next 2 phases

Activate

- Run every test once
- Determine which tests activate each fault
- Determine which faults are not activated

Detect

- Insert each fault into the RTL
- Simulate tests that activated fault
- Determine if any test is capable of propagating and detecting each fault

Fault Classifications

Category	Description
Non-Activated	No test capable of activating the fault
Non-Propagated	Fault Activated, but not propagated to a checker
Non-Detected	Fault propagated to checker, but no fail reported
Detected	At least one test reported a failure

Qualification Results

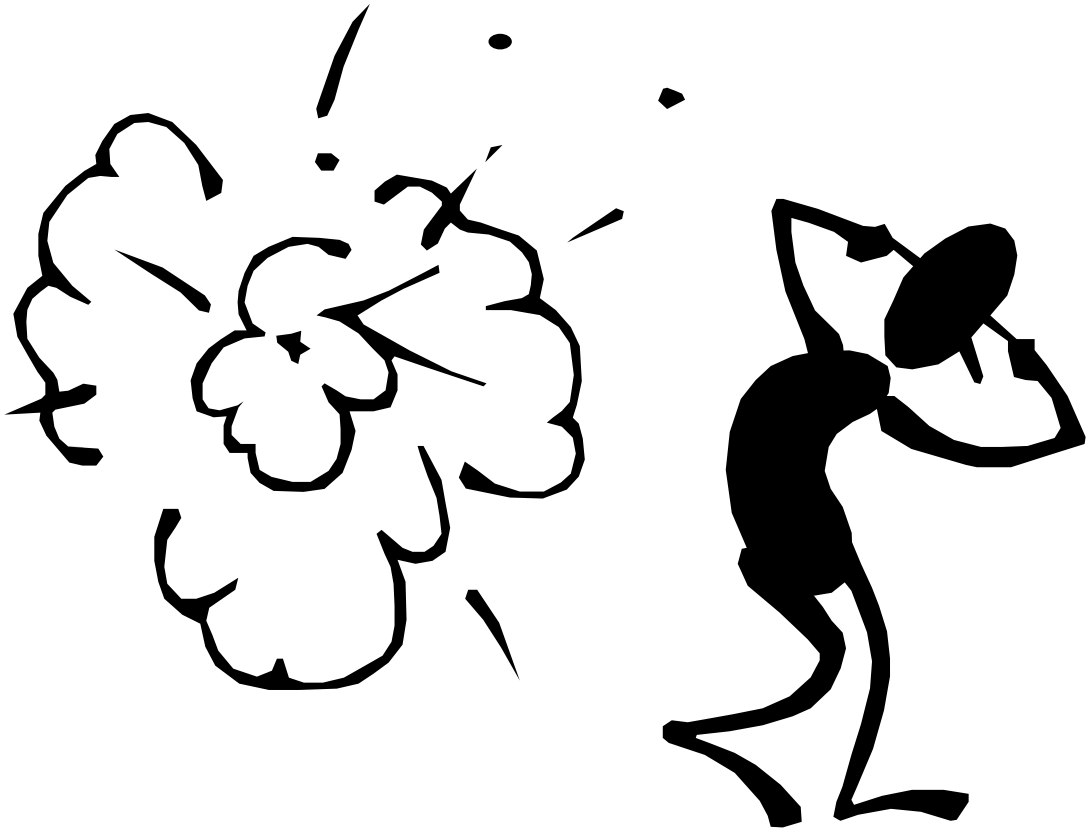
Qualification Status... | Fault Classes... | Source Files... | Testcases... | Probes... | Waveforms... | Help ▾

Fault classes for 'seq_top'

This report was generated on: 2011-08-05 at 15:44:38

Class name	Faults in Design	Faults in List	Non-Activated	Non-Propagated	Detected	Non-Detected	Disabled By Certitude	Disabled By User	Dropped	Not Yet Qualified
ConnectivityOutput	180	180	0	0	180	0	0	0	0	0
ResetConditionTrue	233	233	0	13	202	0	4	13	1	0
SynchronousControlFlow	864	847	0	39	663	4	7	115	19	0
ConnectivityInput	525	521	0	3	430	0	39	33	16	0
SynchronousDeadAssign	220	218	0	8	162	1	10	37	0	0
ComboLogicControlFlow	202	202	0	6	129	0	10	31	26	0
SynchronousLogic	811	799	0	13	636	0	11	118	21	0
ComboLogic	3634	3617	0	39	2856	2	156	305	259	0
OtherFaults	1627	4	0	0	0	0	4	0	0	0
All Fault Classes (9)	**8296**	**6621**	**0**	**121**	**5258**	**7**	**241**	**652**	**342**	**0**

Issues Found During Functional Qualification



Processor Control Example #1

- Fault number 11757
- Force condition to always evaluate to false
- Only 1 test activated this fault

- Weakness – Checker Error : written to fire “if (A && !A)”

- Impact – Potential Design Bug miss

Fault detail

File name: /proj/snapper/users/brownell/blocks/dag/rtl/dag_mlbreg_reg.v

	Action	Fault ID	Fault Type	Status	Detected By	Distance to Output
→	Disable	11757	ConditionFalse	Non-Detected		2
	Disable	11758	ConditionTrue	Detected	stall_test	
	Disable	11759	NegatedCondition	Detected	stall_test	

With the fault 11757 of type 'ConditionFalse', the code:

```
81 assign rout[31:0] = wr_imm_e1f   ? imm_imlbdata_e1f[31:0] :
```

Analyze with Verdi

Is changed into:

```
81 assign rout[31:0] = 1'b0  ? imm_imlbdata_e1f[31:0] :
```

Testcases that activate and propagate the fault:

Testcase	Runtime	Fault propagated to	Debug
REGMV_only_basic_test	00:00:33.280	work.dag_top.dag_xfer_f1f	Dump Waveforms

Update Comment

Processor Control Example #2

- Fault number 6163
- Combo Logic fault – bitflip from a 0/1 to a 1/0
- 5 tests activated this fault

- Weakness – Address outputs driven to “0” when no instruction in F stage. Testbench not checking address 0 when no active instruction in F stage

- Impact – Potential over design, could save power if bus not required to be driven to “0”.

Fault detail

File name: /proj/snapper/users/brownell/blocks/dag/rtl/dag_adr0_dp.v

	Action	Fault ID	Fault Type	Status	Reason	Distance to Output
	Disable	6161	BitFlip	Dropped by fault	6163	
	Disable	6162	FlipFirst	Dropped by fault	6163	
→	Disable	6163	FlipLast	Non-Detected		1

With the fault 6163 of type 'FlipLast', the code:

```
271    3'b000   : addr0_e1f[31:0] =  32'b0;
```

Analyze with Verdi

Is changed into:

```
271    3'b000   : addr0_e1f[31:0] = 32'b00000000000000000000000000000001;
```

Testcases that activate and propagate the fault:

Testcase	Runtime	Fault propagated to	Debug
DSPLDST_only_basic_test	00:00:38.540	work.dag_top.dag0_adr_f1f	Dump Waveforms
MULTI_only_basic_test	00:00:54.970	work.dag_top.dag0_adr_f1f	Dump Waveforms
PROGCTL_only_basic_test	00:00:30.690	work.dag_top.dag0_adr_f1f	Dump Waveforms
basic_test	00:00:41.580	work.dag_top.dag0_adr_f1f	Dump Waveforms
debug_basic_test	00:00:43.610	work.dag_top.dag0_adr_f1f	Dump Waveforms

```
255 //*****************************************************************************
256 // res0_e1f MUX
257
258 assign res0_e1f[31:0] = sel_res0_rev_e1f ? brsum_e1f[31:0] : premod_add_e1f[31:0];
259
260 //*****************************************************************************
261 // Input address mux
262 // Select between i0_e1f and p0_e1f
263
264 assign byteop_clr_e1f[0] =  byteops_e1f && (!dag0_16bit_b_e1f || !dag0_32bit_b_e1f);
265 assign byteop_clr_e1f[1] =  byteops_e1f &&  !dag0_32bit_b_e1f;
266
267 always @(sel_addr0_pmod_e1f or sel_addr0_iadd_e1f or sel_addr0_iind_e1f or
268 p0_e1f or i0_e1f or premod_add_e1f or byteop_clr_e1f)
269 begin
270   case({sel_addr0_pmod_e1f,sel_addr0_iadd_e1f,sel_addr0_iind_e1f})
271   3'b000  : addr0_e1f[31:0] = 32'b0;
272   3'b001  : addr0_e1f[31:0] = p0_e1f[31:0];
273   3'b010  : addr0_e1f[31:0] = {i0_e1f[31:2],(i0_e1f[1:0] & ~byteop_clr_e1f[1:0])};
274   3'b100  : addr0_e1f[31:0] = premod_add_e1f[31:0];
275   default : addr0_e1f[31:0] = 32'bx;
276   endcase
277 end
278
279 //*****************************************************************************
280 // Page   ..... Only for premodify constant ... optimise???
281 // p_eff_e1f + m_eff_e1f == prvaddr0_f1f
282
283 wire [31:0] p_eff_e1f       = sel_addr0_iind_e1f ? p0_e1f[31:0] : i0_e1f[31:0];
284 wire [31:0] m_eff_e1f       = sel_addr0_pmod_e1f ? pimm0_e1f : 32'b0;
285
286 //SLOWER ..more area
287 //wire [33:0] pagesum_e1f     =  p_eff_e1f[31:0] +
288 //                               m_eff_e1f[31:0] +
```

Common Issues Found

- Missing checks on top level outputs
  - Most often heard phrase during FQ: “I meant to check that”
- Missing reset cases
  - No test which asserts reset multiple times
  - Almost every TB we check has this problem
- Logic activated but poorly propagated
- Missing Negative checks
  - Checkers often written to check when a signal should be asserted but they do not check that it is de-asserted the rest of the time
- Bad checkers
  - Checks written or called incorrectly

I

Functional Qualification!

Experiences with FQ Setup (The Bad)

- **First time setting up will likely take 1-2 days**
  - Testbench/scripts must be able to do separate compile and run!
    - Must take argument to define unique logname
  - Defining pass/fail for FQ is critical and easy to get wrong
    - If this is wrong all results are invalid
    - Sometimes difficult to tell when this incorrect
  - Separate compile/executable for each test can be painful!
    - The instrumented code can increase the compile time a lot
  - Occasional SV/Verilog syntax issues require minor RTL changes

Experiences with FQ Setup (The Good)

- **Once you have done one, very easy to do the next project**
  - Typical setup for a new project is a couple of hours
  - Process automated within ADI so setup is minutes
- **Time to results very quick after setup**
  - Model phase
    - Typically <15minutes
  - Activate phase
    - Typically <1hr
  - Detect phase to first non-detected fault
    - Typically <1hr
- **Interactive mode so you can debug faults while tool running**

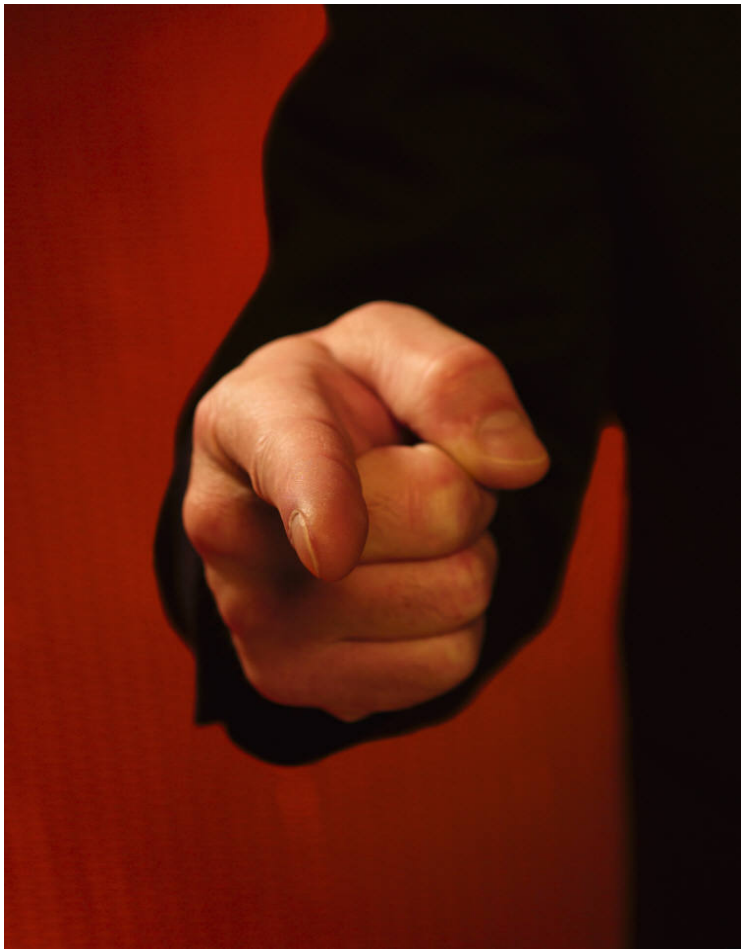
Lessons Learned

Lessons Learned

- **Expect to be offended**
  - Every TB has issues.
  - Functional Qualification will find them ☺.
- **Don’t Run before your environment is complete**
  - If you know you are missing a check on some pins Functional Qualification is just going to point that out.
- **Have a clear plan before you start:**
  - 0 non-activated and 0 non-detected
  - What faults are critical

Who Should be using FQ?

- You
- Any style testbench
- Block to System Level
- Only requirements
  - Self Checking Env/Tests
  - Design is RTL



Why Should you use FQ

- **Higher quality verification environment**
  - More likely to catch RTL bugs
- **Higher quality designs**
  - Less chance of discovering bugs late in the process or after tape-out
- **More confidence in RTL sign-off**
  - Objective, quantifiable criteria
- **Earlier identification of problems**
  - Achieve robust verification environment more quickly
- **Difference Between Thinking or Knowing**
  - That your Environment is working

- Norwood DV and Design Team
- Springsoft/Synopsys Marty Rowe/Myles Glisson