Verifying Functional, Safety and Security Requirements (for Standards Compliance)

Mike Bartley (TVS)
in collaboration with

OneSpin Solutions

Tortuga Logic

TVS
Test and Verification Solutions

TVS Agenda

- 11.00 Introductions
- 11.05 TVS
- 11.20 OneSpin
- 11.55 Tortuga Logic
- 12.10 TVS
- 12.25 Q&A (TVS, OneSpin, Tortuga Logic)

Introduction to your speakers

- **Dr Mike Bartley, CEO of TVS**
  - PhD in Mathematical Logic, MSc in Software Engineering, MBA
  - 25 years in Software Testing and Hardware Verification
  - Started TVS (Test and Verification Solutions) in 2008
    - 125 engineers worldwide
    - UK, France, Germany, Italy, Sweden, Turkey, India, Singapore, Korea, China, US
    - Delivering SW test and HW verification products and services
      - » Focus on reliability, safety, security
  - asureSIGN
    - "Requirements Driven Test and Verification" methodology
    - Define requirements and refine them to verification plans and capture sign-off

Jörg Große

Product Manager for Functional Safety, OneSpin Solutions

Jörg Große recently joined OneSpin Solution as a Product Manager for Functional Safety.
He has more than 20 years of experience in EDA, functional verification and ASIC design, having served at companies in Europe, the United States and New Zealand.
As co-founder of a successful Silicon Valley based startup, he was central in developing the concept of fault/mutation testing into a state-of-the-art EDA tool.
He deployed this technology in many leading semiconductor companies, increasing the quality of their functional verification.
He holds a Dipl.-Ing.(FH) in Electrical Engineering from the University of Applied Science Anhalt.

Dr. Ryan Kastner

co-founder of Tortuga Logic

Dr. Ryan Kastner is a co-founder of Tortuga Logic and has over 10 years of experience in realm of hardware security. He has served as a principal investigator on various government and industrial grants related to hardware security (over $3 million in toto). This includes the National Science Foundation Innovation Corps award, which focuses on commercializing technology from academia. Dr. Kastner is a professor in the Computer Science and Engineering Department at UCSD. He received a PhD in Computer Science at UCLA, a masters degree (MS) in engineering and bachelor degrees (BS) in both Electrical Engineering and Computer Engineering, all from Northwestern University.

TVS Agenda

- **11.00 Introductions**
- **11.05 TVS**
  - Safety and security in Hardware and Software
  - Requirements Driven Test and Verification (RDTV)
  - Using an ECC example and breaking it down into a test plan
- **11.20 OneSpin**
- **11.55 Tortuga Logic**
- **12.10**
  - Analysing the results and signoff
  - Advantages of RDTV
- **12.25 Q&A (TVS, OneSpin, Tortuga Logic)**

Why are Safety and Security important?

- **IC Insights research**
  - The automotive industry is set to drive chip demand over the coming years.
  - IC Insights research suggests the demand from automotive is expected to exhibit average annual growth of 10.8% into at least 2018.
  - Demand will come from safety features that are increasingly becoming mandatory, such as backup cameras or eCall, and the near-ubiquitous driver-assistance systems.
- **IoT**
  - Drones (avionics), autonomous cars, robots, ….
  - Connected devices have potential security threats
- **TTTech**
  - By 2020 50% of all ICs will be safety-related
  - By 2020 50% of all ICs will be connected

Safety Standards

- **IEC61508:** Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems
- **DO254/DO178:** Hardware/Software considerations in airborne systems and equipment certification
- **EN50128:** Software for railway control and protection systems
- **IEC60880:** Software aspects for computer-based systems performing category A functions
- **IEC62304:** Medical device software -- Software life cycle processes
- **ISO26262:** Road vehicles – Functional safety

Safety

Safety

"Freedom from unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment"

Functional Safety

"That part of the overall safety that depends on a system or equipment operating correctly in response to its inputs"

How Systems Fail

- **Random failures**
  - Can usually predict (statistically)
  - Can undertake preventative activities
- **Systematic failures**
  - Specified, designed or implemented incorrectly
  - Can't usually predict
- **Systemic failures**
  - Shortcomings in culture or practices

Focus Here Today

TVS Has Expertise

Basics of Safety Standards

- **The life cycle processes are identified**
- **Objectives and outputs for each process are described**
  - Objectives are mandatory
  - But vary by Integrity Level
  - For higher Integrity Levels, some Objectives require Independence

IEC61508

- **Dynamic analysis and testing**

Technique	SIL 1	SIL 2	SIL 3	SIL 4
Structural test coverage (entry points) 100%	HR	HR	HR	HR
Structural test coverage (statements) 100%	R	HR	HR	HR
Structural test coverage (branches) 100%	R	R	HR	HR
Structural test coverage (conditions, MC/DC) 100%	R	R	R	HR
Test case execution from boundary value analysis	R	HR	HR	HR
Test case execution from error guessing	R	R	R	R
Test case execution from error seeding	-	R	R	R
Test case execution from model-based test case generation	R	R	HR	HR
Performance modelling	R	R	R	HR
Equivalence classes and input partition testing	R	R	R	HR

Key Processes

- Plans & Standards
- Requirements
- Design Specifications
- Reviews and Analyses
- Testing (against specifications)
  - At different levels of hierarchy
- Test Coverage Criteria
- Requirements Traceability
- Independence

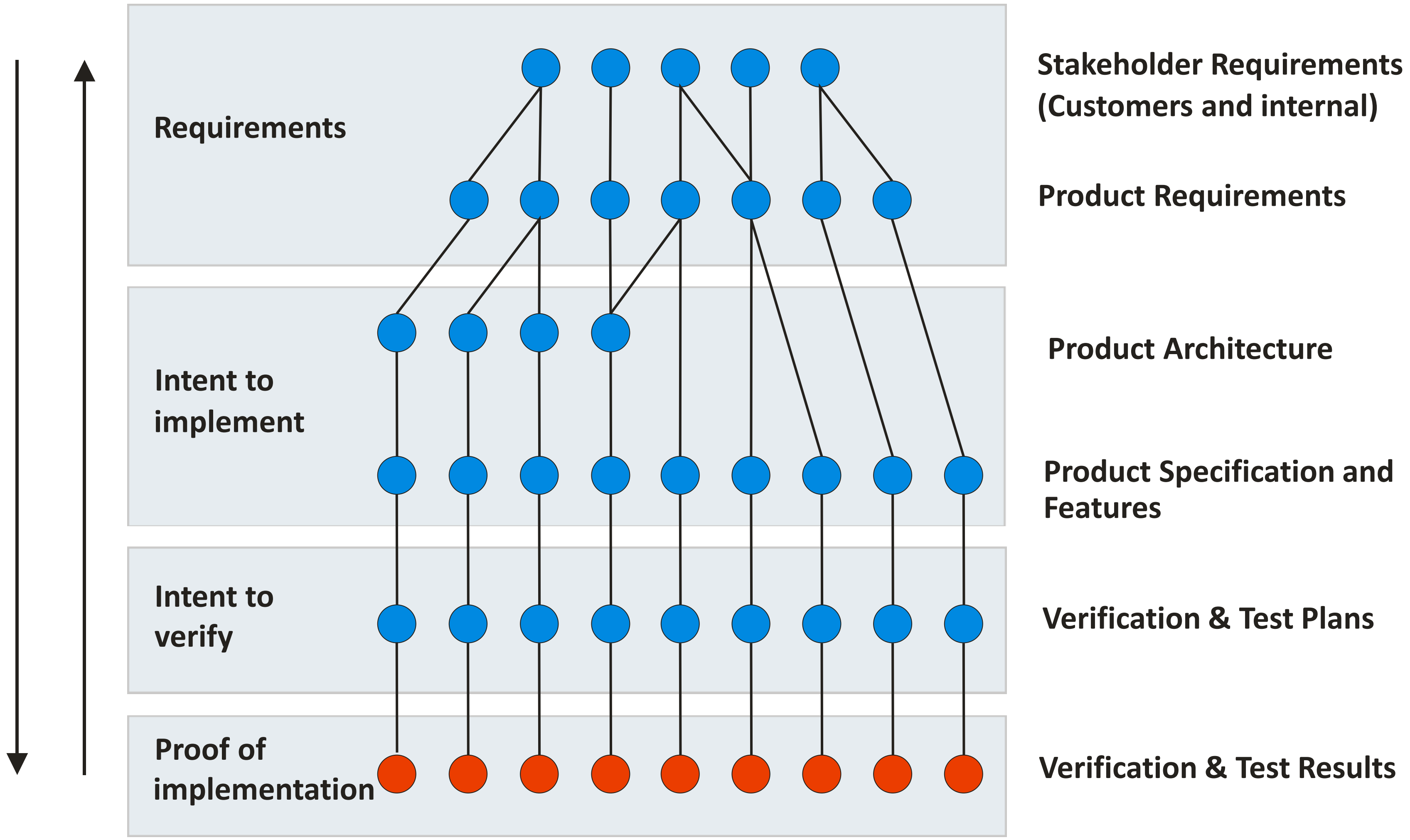
Key Deliverables

- Verification Plan
- Validation and Verification Standards
- Traceability Data
- Review and Analysis Procedures
- Review and Analysis Results
- Test Procedures
- Test Results
- Acceptance Test Criteria
- Problem Reports
- Configuration Management Records
- Process Assurance Records

Key Deliverables

- Verification Plan
- Validation and Verification Standards
- Traceability Data
- Review and Analysis Procedures
- Review and Analysis Results
- Test Procedures
- Test Results
- Acceptance Test Criteria
- Problem Reports
- Configuration Management Records
- Process Assurance Records

Traceability in Practice

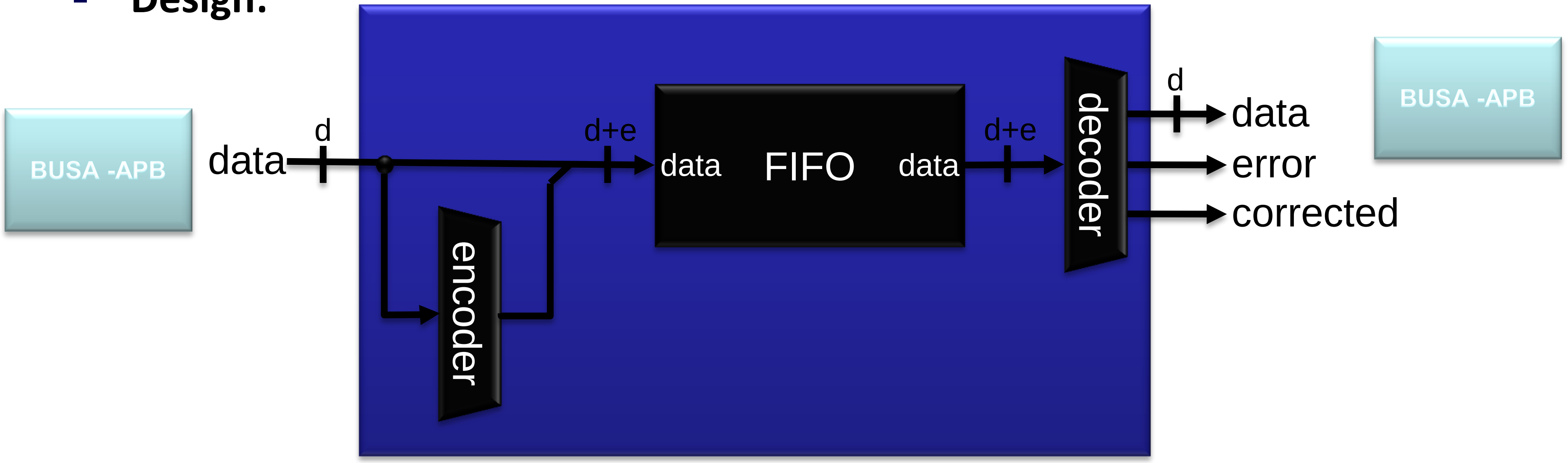

Requirements
Intent to implement
Intent to verify
Proof of implementation
Stakeholder Requirements (Customers and internal)
Product Requirements
Product Architecture
Product Specification and Features
Verification & Test Plans
Verification & Test Results

Shows a mapping from features to verification and test plans

Example – Safeguarding a FIFO

- **Safety Function**
  - Detect 1-bit errors and correct them
  - Detect 2-bit errors and raise alarm

- **Design:**



  - Encoder adds **e** data bits stored in RAM
  - Decoder detects & corrects 1-bit faults on read (`error=0`, `corrected=1`)
  - Decoder detects 2-bit faults on read (`error=1`)

A full ? set of requirements

R1	FIFO_SINGLE_BIT	The FIFO will be able to detect and correct single bit errors.
R2	ERR_REPORT_CPU	Single bit errors must be reported to the CPU
R3	MULT_ERR_CPU	The FIFO will be able to detect and report multiple bit errors to the CPU
R4	FIFO_NOT_FULL	Data arriving on the write interface shall be written in to the FIFO as long as it is not full
R5	FIFO_NOT_EMPTY	Requests to read data shall return the oldest data in the FIFO as long as it is not empty
R6	FIFO_EMPTY_READ	Read attempts from an empty FIFO shall be reported to the CPU
R7	FIFO_WRITE_FULL	Write attempts to a full FIFO shall be reported to the CPU
R8	WRITE_APB_INTERFACE	Write data shall come across an APB interface
R9	READ_APB_INTERFACE	Read data shall be send across an APB interface
R10	STATUS_REG_SINGLE_ERR	A status register will record a single bit error
R11	STATUS_REG_MULTI_ERR	A status register will record a multibit error bit error
R12	STATUS_REG_FIFO_FULL_	A status register will indicate a FULL fifo
R13	STATUS_REG_FIFO_EMPTY	A status register will indicate an empty fifo
R14	STATUS_REG_FIFO_OVERFLOW	A status register will indicate overflow
R15	STATUS_BIT_OVERFLOW	A status bit will record underflow
R16	PRIVILEGE_LEVEL_1	only users with privilege level 1 can read from the FIFO
R17	PRIVILIGE_LEVEL_1_2	only users with privilege level 1 or 2

Safety
Functional
Security

Safety Requirement Decomposition (example)

Req: Safeguard Design against single bit soft errors

Sub-Concept/Req: Safeguarde each FIFO

Safety Reqirements for FIFO / Concept:

- Use ECC FIFO
- Detect 1-bit errors and correct them
- Detect 2-bit errors and raise alarm

Safety Verification Requirement for ECC FIFO Implmementation

- If no error occurs, nothing is flagged and the data is uncorrupted
- If one error occurs, no error is flagged, the data is uncorrupted and the correction is flagged
- If two errors occur, an error is flagged, but no correction

Formal Safety Properties to verify Implementation

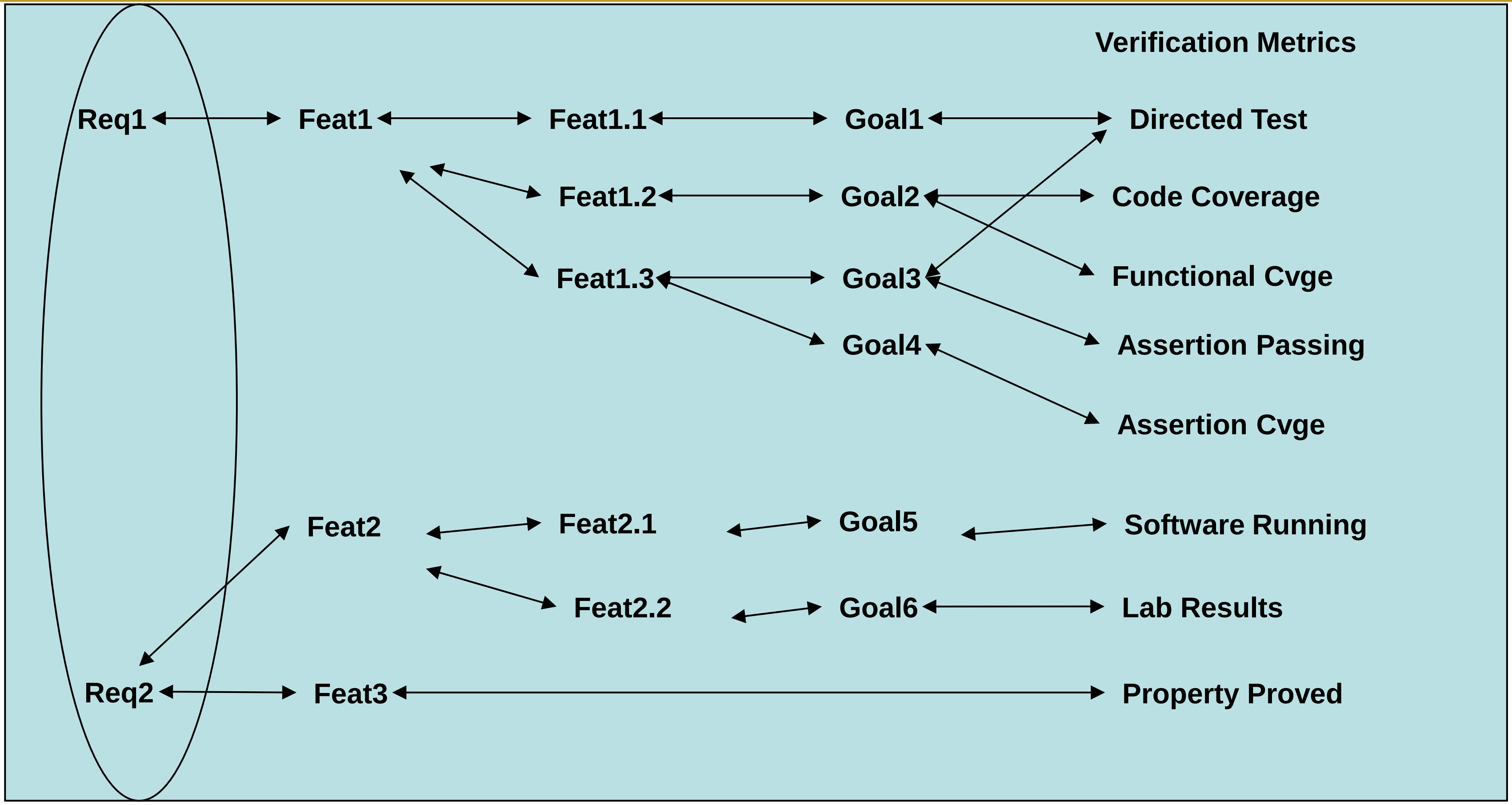
- Separate slide

Mapping Security Requirements to Features

- **R16 - PRIVILEGE_LEVEL_1: only users with privilege level 1 can read from the FIFO**

ECC_SECURITY_1	Reads without privilege level	Reads without privilege level 1 or 2 will cause a bus error
ECC_SECURITY_2	Reads with privilege level	Reads with privilege level 1 or 2 will be successful

Mapping Requirements to Verification Metrics



Relationships can be:

- Bi-directional
- Many-many

Metrics can be:

- From HW verification
- From Silicon validation
- From SW testing

asureSIGN Demo

- **Mapping the requirements to a test plan**

TVS Agenda

- **11.00 Introductions**
- **11.05 TVS**
  - Safety and security in Hardware and Software
  - Requirements Driven Test and Verification (RDTV)
  - Using an ECC example and breaking it down into a test plan
- **11.20 OneSpin**
- **11.55 Tortuga Logic**
- **12.10 TVS**
  - Analysing the results and signoff
  - Advantages of RDTV
- **12.25 Q&A (TVS, OneSpin, Tortuga Logic)**

Requirement Driven Verification for Safety & High Reliability

Jörg Große

Why Safety & Reliability Verification is important - Risk Drivers

- Cars are computer on wheels
  - But reset is not an option, especially not when diving at high speeds

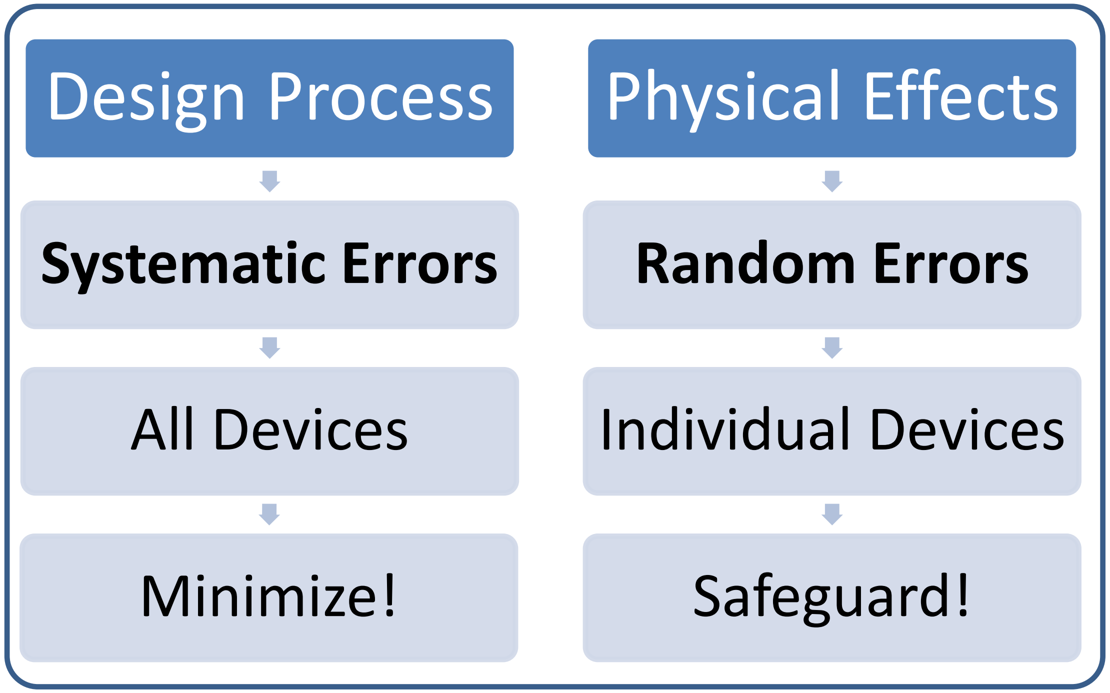
Systematic Errors

- Machine Errors
  - Synthesis bugs, ..
- Human Errors
  - Implementation bugs
  - Design bugs
- Driven by
  - Ever increasing complexity
  - Time to market and budget

Random Errors

- Hard Errors
  - Latch-ups
  - Burnouts (struck-at faults)
- Soft Errors
  - Transients (glitches, bit flips)
- Driven by
  - Decreasing geometries
  - Decreasing supply voltage
  - Increasing area

Consequence?

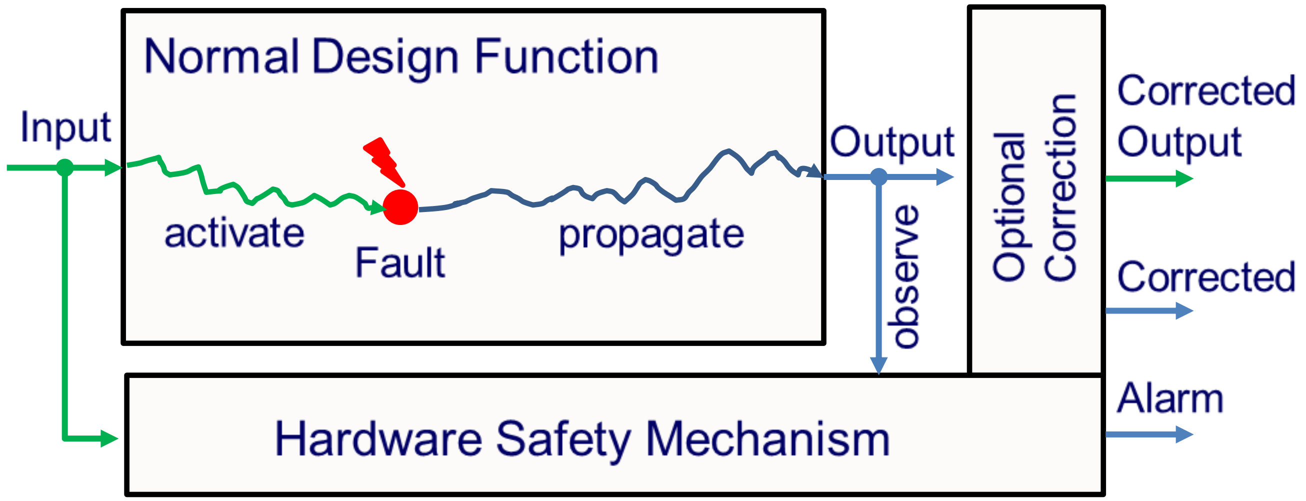


Functional Verification + Safeguard Verification

=

Functional Safety Verification

Safeguarding against Random Errors



Fault Detection

- Raise alarm

Fault Handling

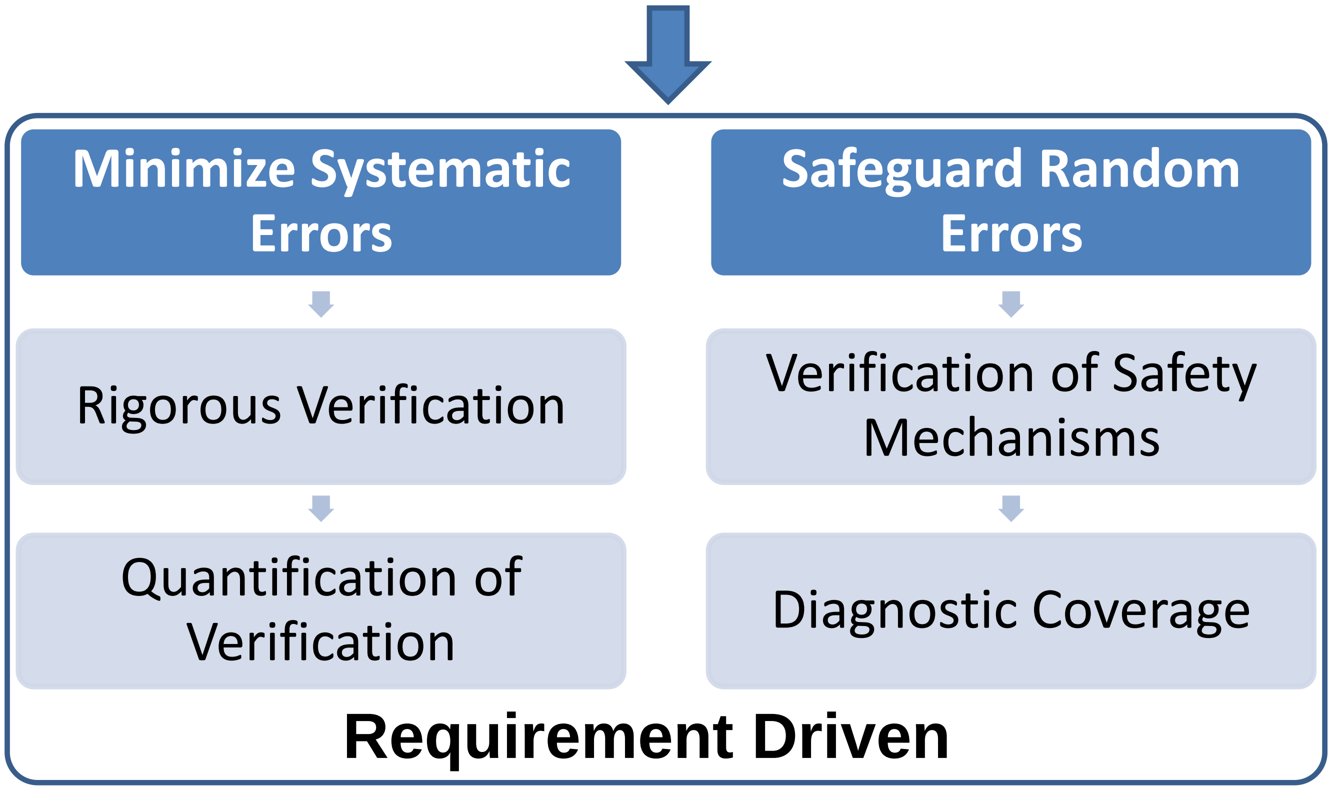
- Enter into safe mode
- Or correct erroneous output

Examples

- Parity, ECC, lock-step

Additional Verification Effort for

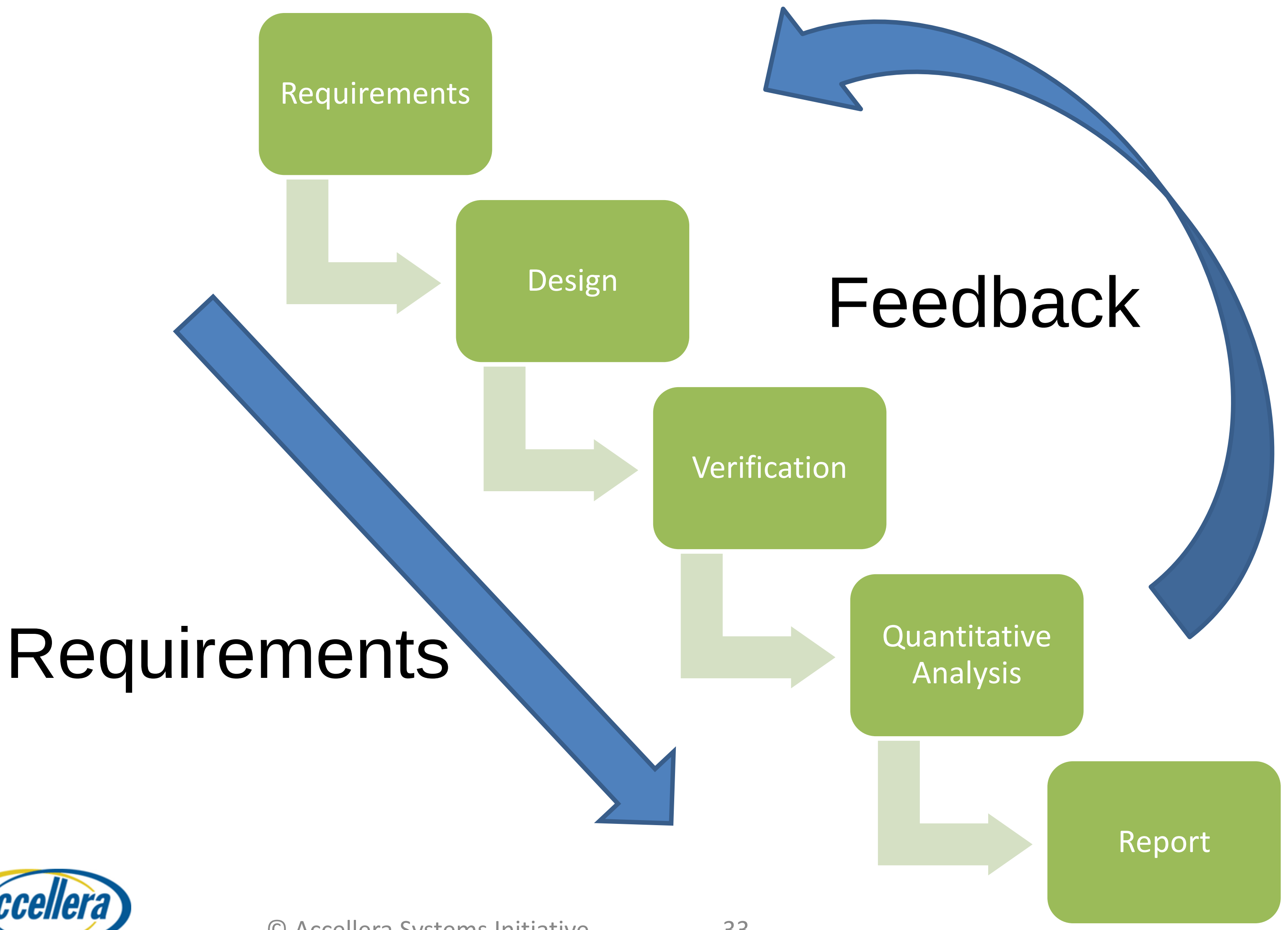
Functional Safety Verification


Minimize Systematic Errors
Rigorous Verification
Quantification of Verification
Safeguard Random Errors
Verification of Safety Mechanisms
Diagnostic Coverage
Requirement Driven

Puts additional pressure on Time-to-Market & Budget!
=> Automation

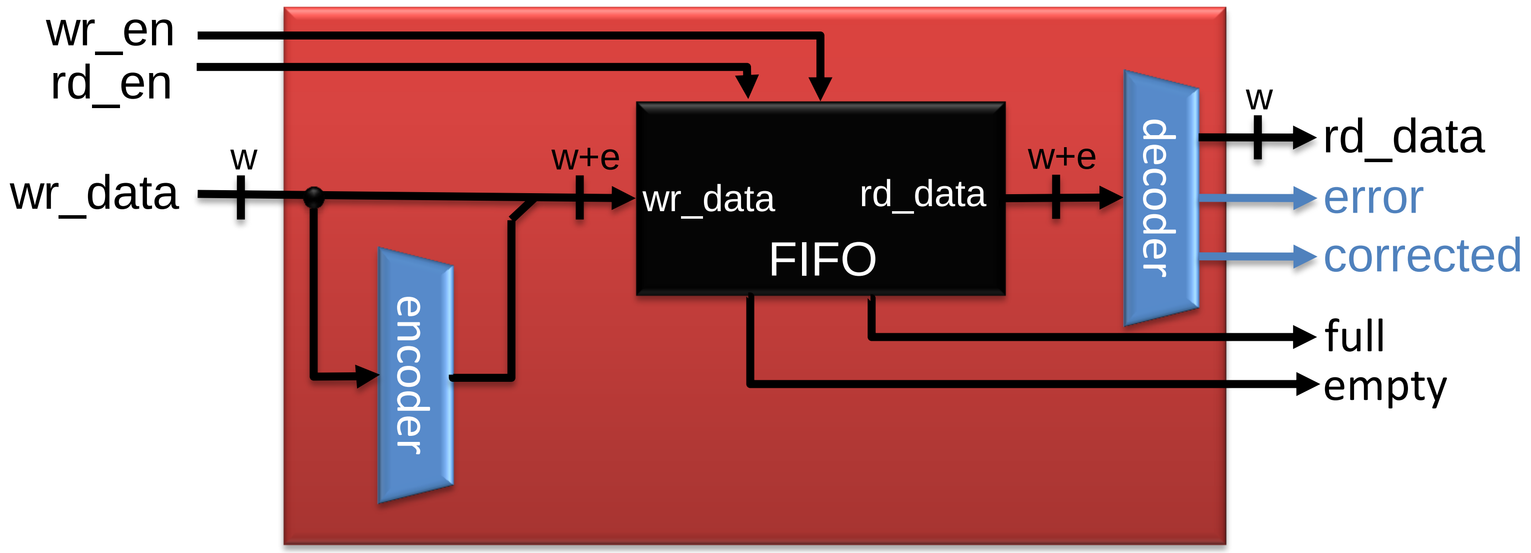
Minimizing Systematic Errors with Rigorous Requirement Based Verification

Generic Verification Flow with Requirement Tracing



Example Design
Safeguarding a FIFO with ECC

- Safety Functions
  - Detect 1-bit errors and correct them
  - Detect 2-bit errors and raise alarm
- Design:



- Encoder adds **e** data bits stored in FIFO

Functional & Safety Requirements

Functional Requirements

- The FIFO is not full and empty at the same time
- The FIFO is empty after DEPTH many reads without writes
- The FIFO is full after DEPTH many writes without reads
- The FIFO is no longer empty after a write
- The first data written to an empty FIFO leaves the FIFO unmodified on the first read

Safety Requirements

- If no error occurs, nothing is flagged and the data is uncorrupted
- If one error occurs, no error is flagged, the data is uncorrupted and the correction is flagged
- If two errors occur, an error is flagged, but no correction

Mapping Requirements to Properties

Functional Requirements

The FIFO is not full and empty at the same time

The FIFO is empty after DEPTH many reads without writes

The FIFO is full after DEPTH many writes without reads

The FIFO is no longer empty after a write

The first data written to an empty FIFO leaves the FIFO unmodified on the first read

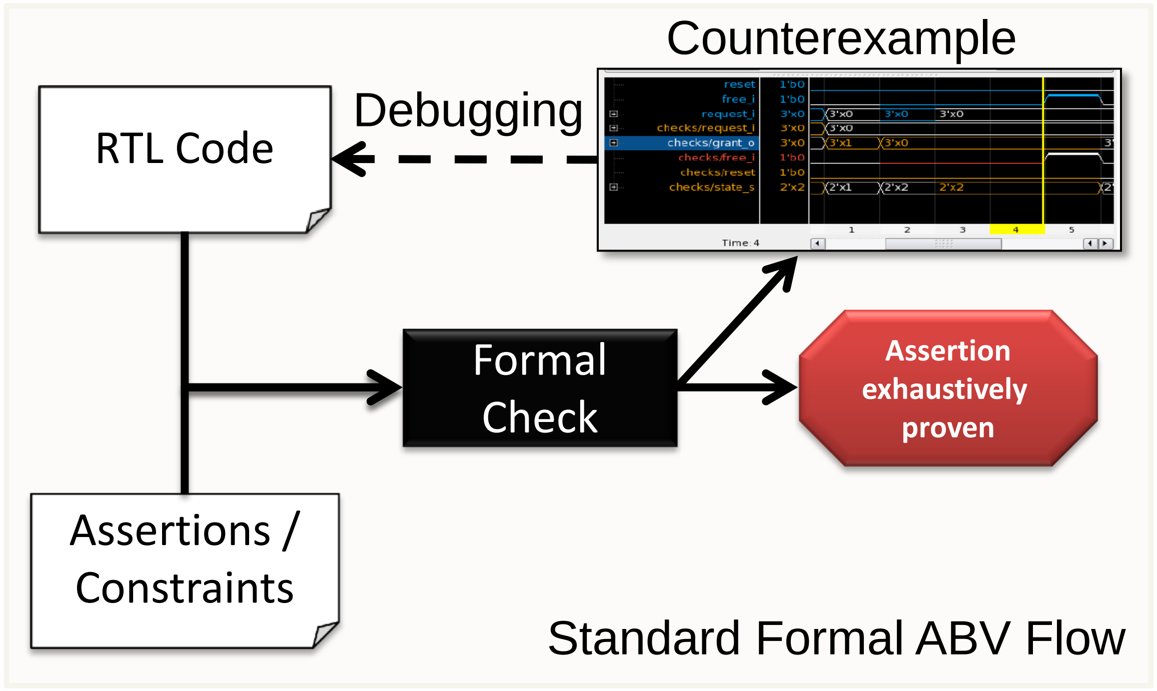
Formal Property

Requirement based verification
→ Create assertions for each requirement!

```
not_empty_after_write_a: assert property
 (disable iff (!FIFO.reset_n) wr_en |=> !empty);
```

Example: assert.not_empty_after_write_a
“The FIFO is no longer empty after a write”

Formal Assertion Based Verification



- **Early:** No stimulus or testbench is needed
- **Efficient:** Typically check-debug-fix in minutes
- **Exhaustive:** If assertion holds -> no simulation needed

Mapping Requirements to Properties

Functional Requirements

The FIFO is not full and empty at the same time

The FIFO is empty after DEPTH many reads without writes

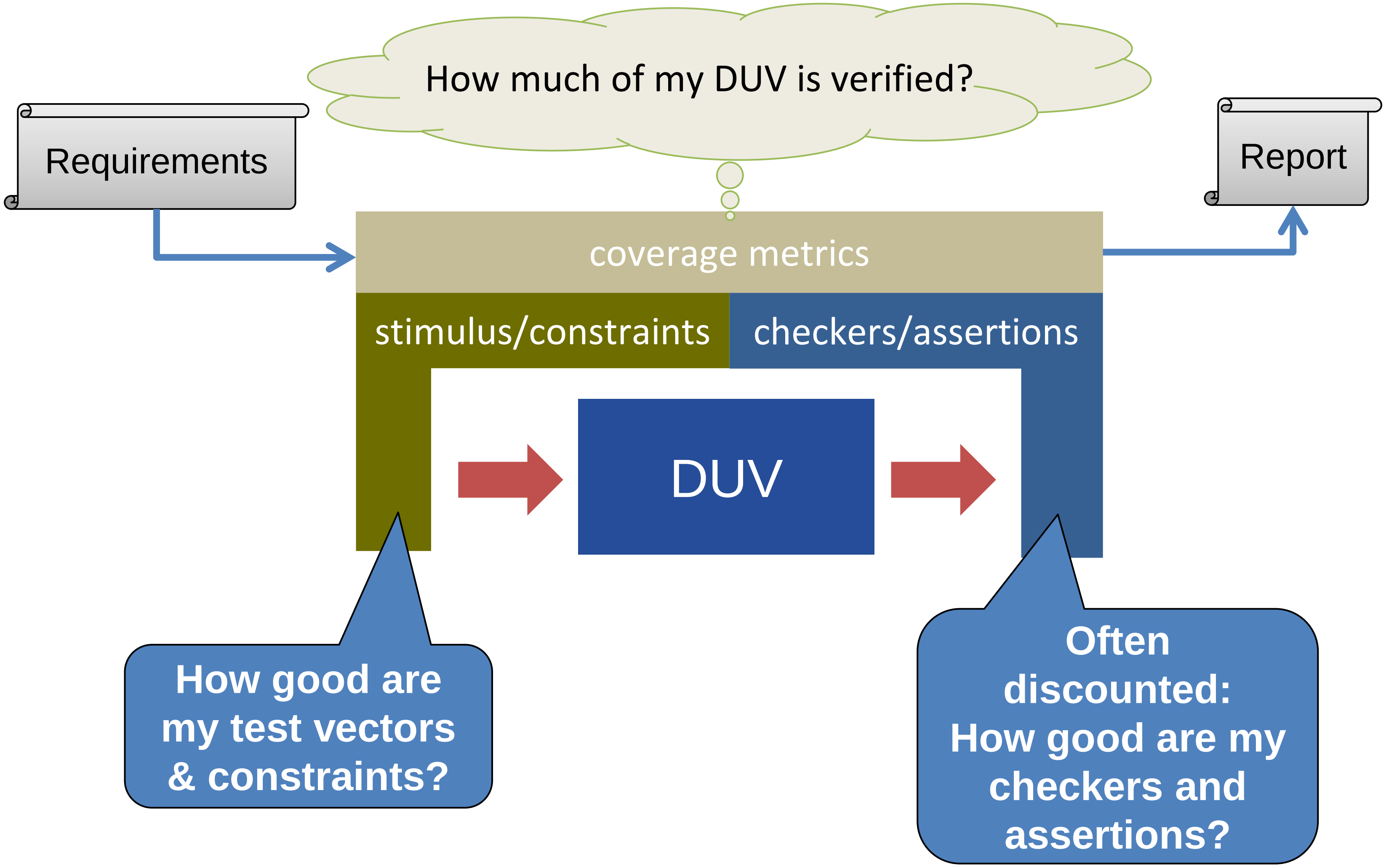
The FIFO is full after DEPTH many writes without reads

The FIFO is no longer empty after a write ✓

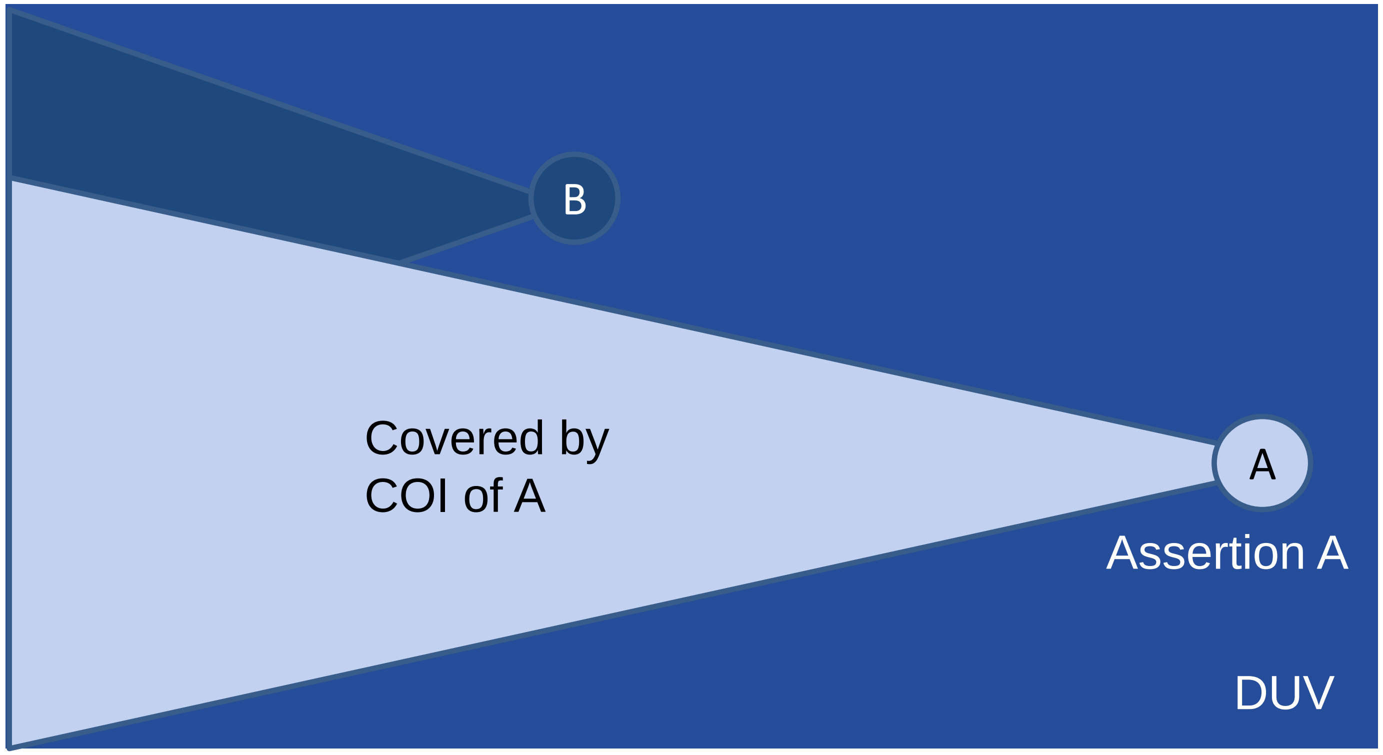
The first data written to an empty FIFO leaves the FIFO unmodified on the first read

Reliable Quantification of Formal Assertion Sets
Coverage Reloaded

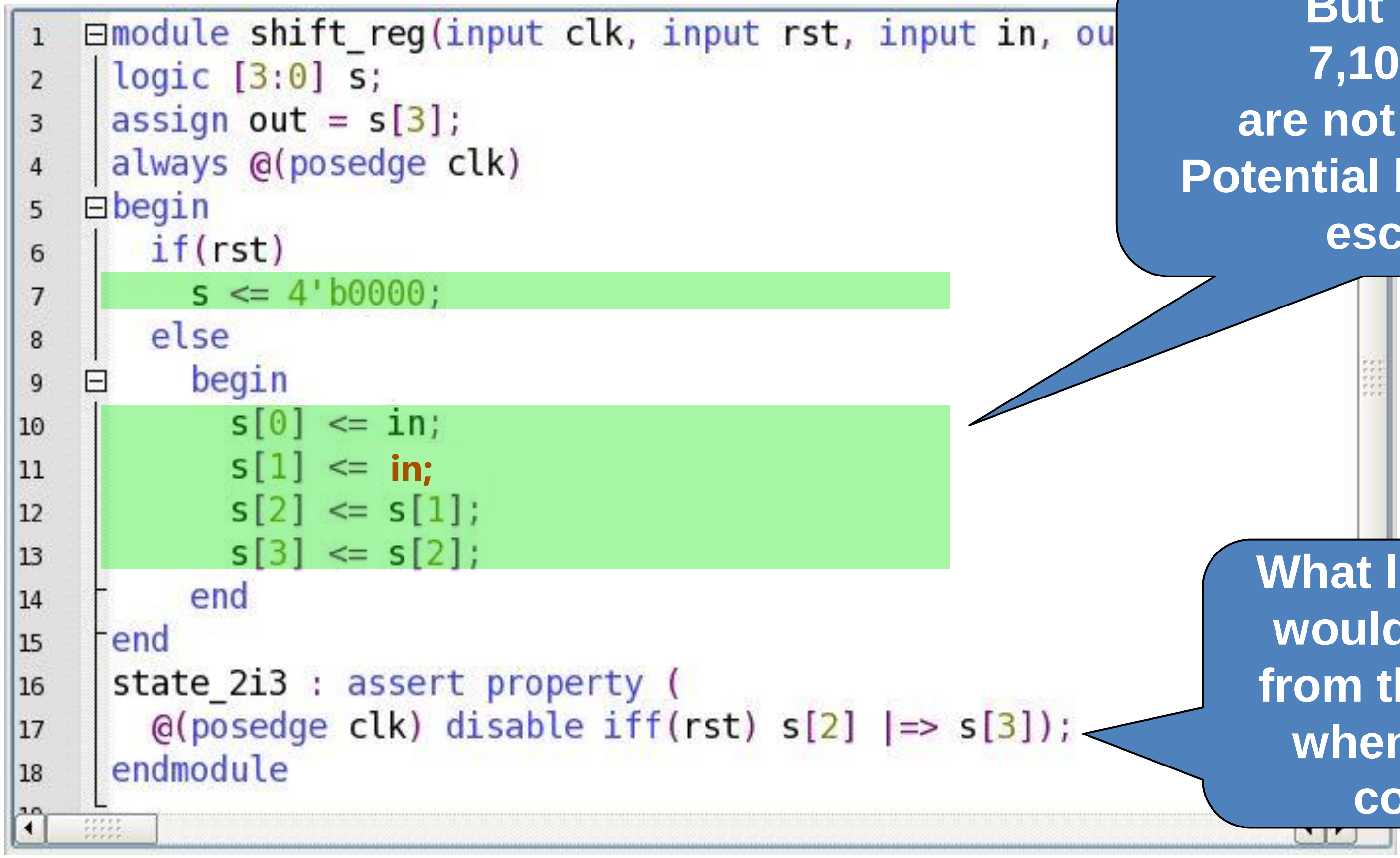
Quantitative Analysis of Verification


How much of my DUV is verified?
Requirements
Report
coverage metrics
stimulus/constraints
checkers/assertions
DUV
How good are my test vectors & constraints?
Often discounted: How good are my checkers and assertions?

Cone-of-Influence Coverage



A Trivial Example – COI Coverage

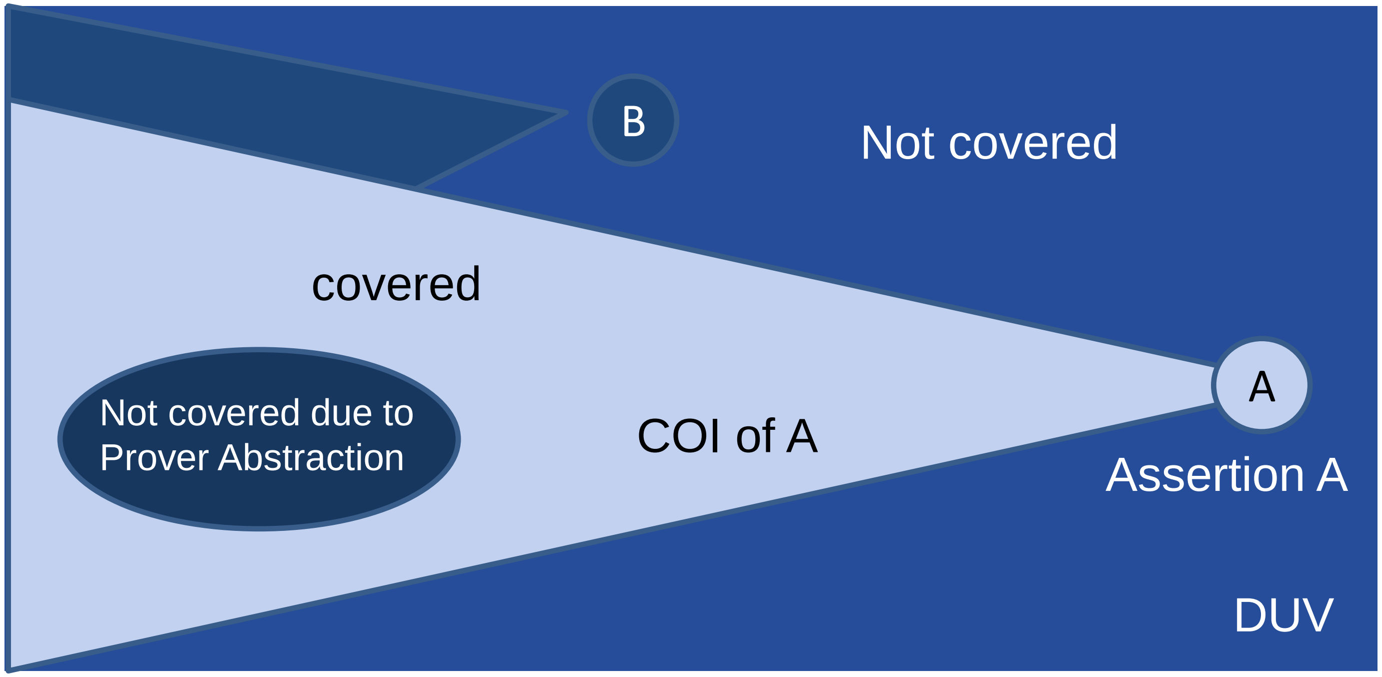


```
1   module shift_reg(input clk, input rst, input in, ou
2   logic [3:0] s;
3   assign out = s[3];
4   always @(posedge clk)
5   begin
6     if(rst)
7       s <= 4'b0000;
8     else
9       begin
10        s[0] <= in;
11        s[1] <= in;
12        s[2] <= s[1];
13        s[3] <= s[2];
14      end
15  end
16  state_2i3 : assert property (
17    @(posedge clk) disable iff(rst) s[2] |=> s[3]);
18  endmodule
```

But lines: 7,10,11,12 are not verified. Potential bugs could escape!

What line coverage would you expect from this assertion when using COI coverage?

Prover Coverage



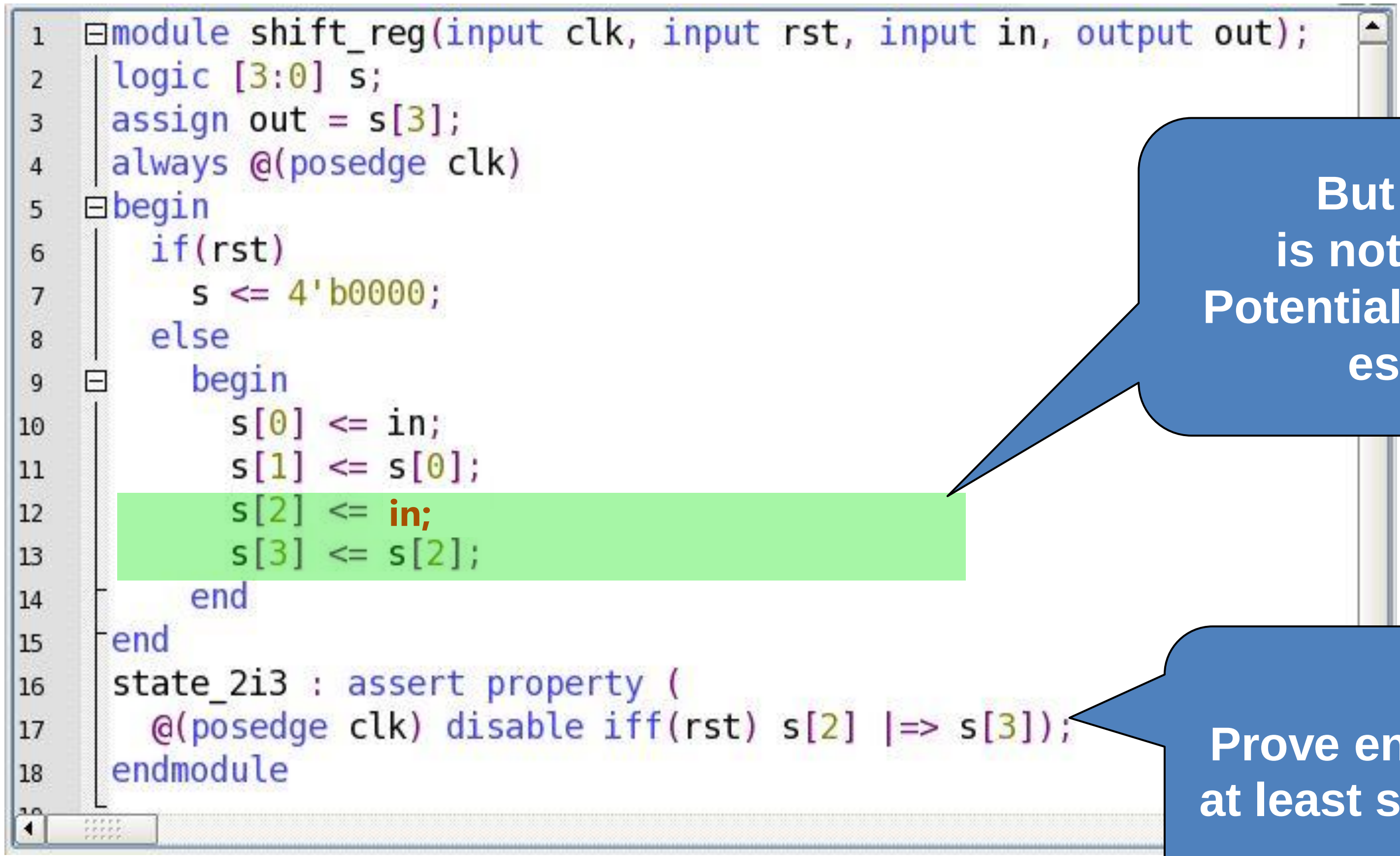
Whatever the prove engine needs is considered covered.

Corresponds to abstractions inside prove engines.

Each prove engine uses different abstractions.

No guarantee that what the prove engine needs is fully covered!

A Trivial Example – Prover Coverage

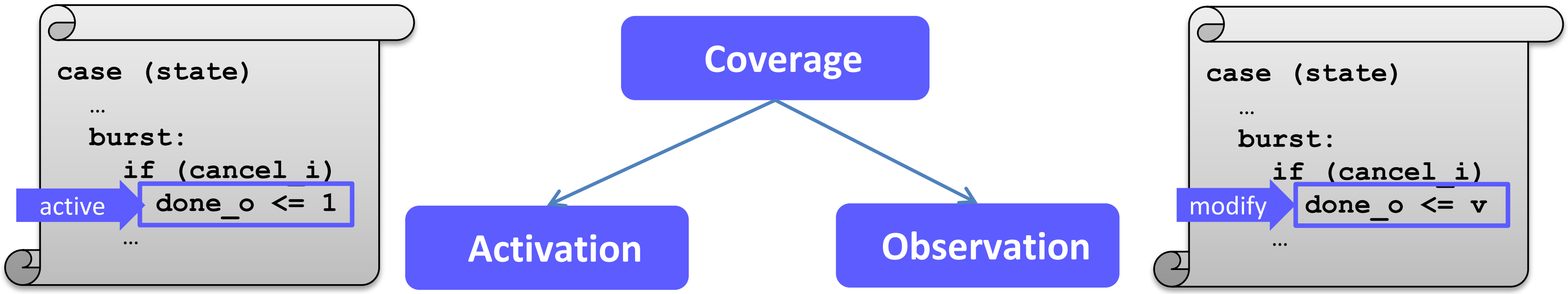


```
1  module shift_reg(input clk, input rst, input in, output out);
2  logic [3:0] s;
3  assign out = s[3];
4  always @(posedge clk)
5  begin
6    if(rst)
7      s <= 4'b0000;
8    else
9      begin
10       s[0] <= in;
11       s[1] <= s[0];
12       s[2] <= in;
13       s[3] <= s[2];
14     end
15 end
16 state_2i3 : assert property (
17   @(posedge clk) disable iff(rst) s[2] |=> s[3]);
18 endmodule
```

But line 12
is not verified.
Potential bugs could
escape!

Prove engine needs
at least s[2] and s[3].

Observation Coverage Principle



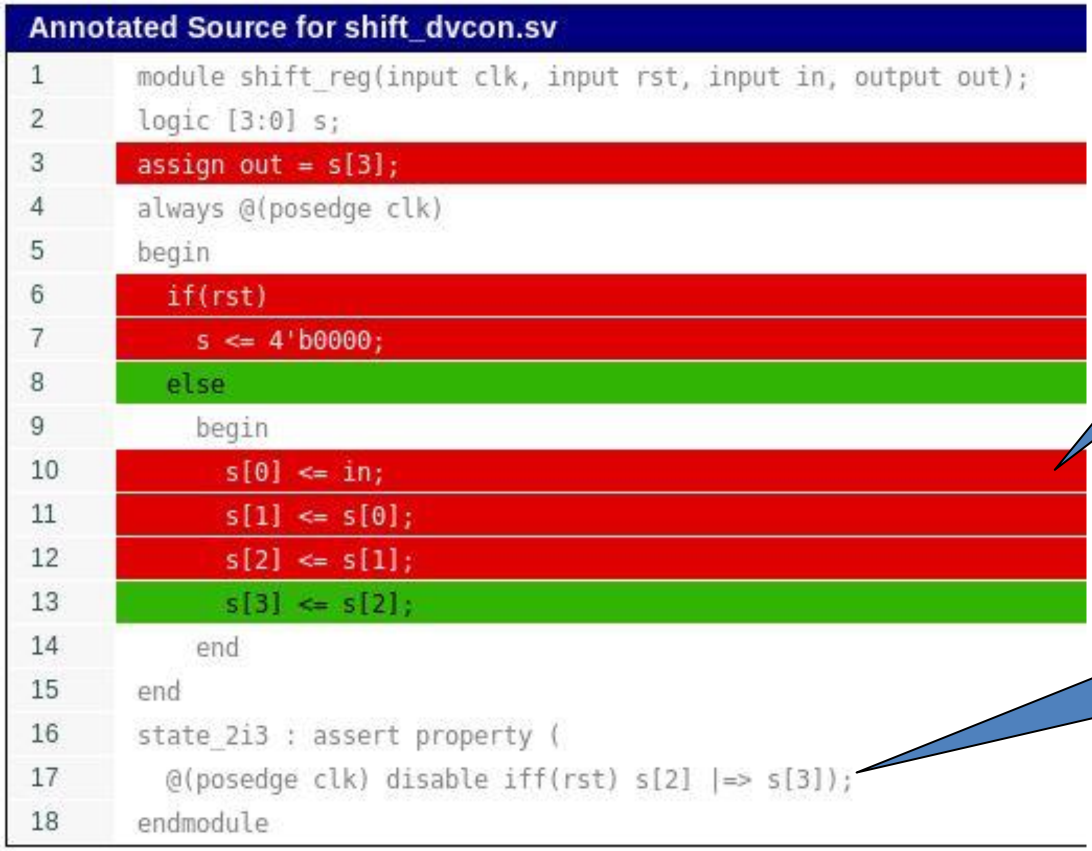
Example: Statement Coverage

- Has the statement been activated?
- If a statement has not been activated during verification, it can't break a check.
- Measures reachability.

- Has the effect been observed?
- If a statement is modified and activated, some assertion should fail.
- Measures quality of assertions.

Been there! Done that!

Using Observation Coverage



Annotated Source for shift_dvcon.sv

```
module shift_reg(input clk, input rst, input in, output out);
logic [3:0] s;
assign out = s[3];
always @(posedge clk)
begin
  if(rst)
    s <= 4'b0000;
  else
    begin
      s[0] <= in;
      s[1] <= s[0];
      s[2] <= s[1];
      s[3] <= s[2];
    end
end
state_2i3 : assert property (
  @(posedge clk) disable iff(rst) s[2] |=> s[3]);
endmodule
```

Unlike COI coverage, observation coverage identifies all unchecked assignments.

Need better or more assertion(s).

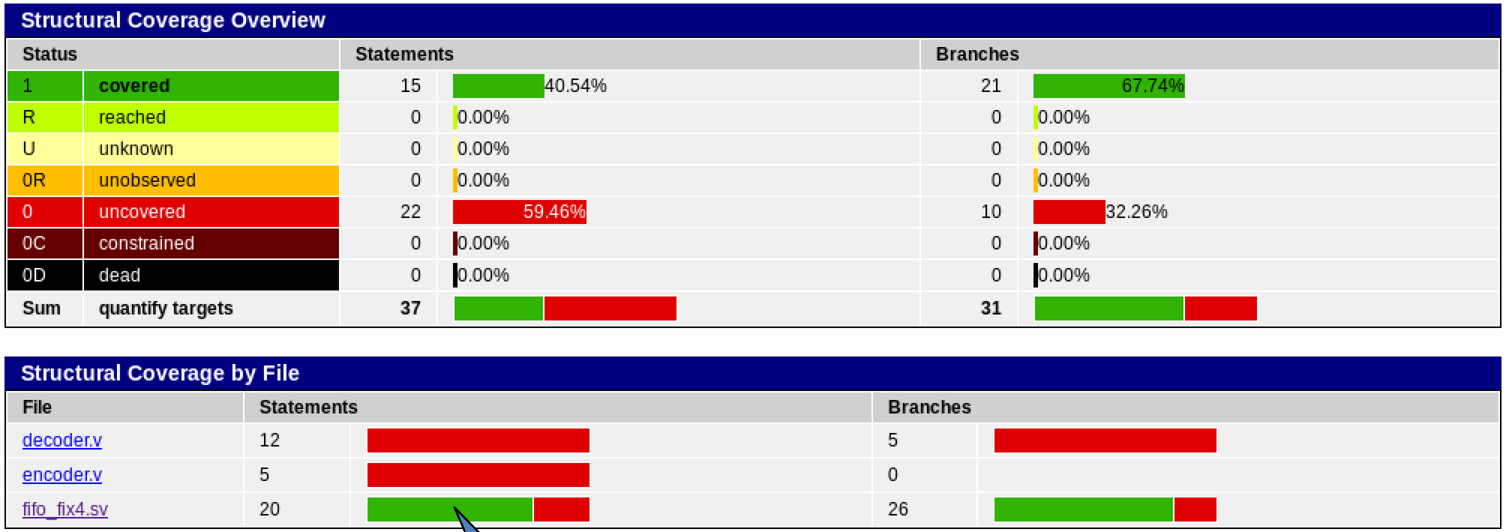
Quantification of Properties

Functional Requirements	**Assertions Hold**
The FIFO is not full and empty at the same time	✓
The FIFO is empty after DEPTH many reads without writes	✓
The FIFO is full after DEPTH many writes without reads	✓
The FIFO is no longer empty after a write	✓
The first data written to an empty FIFO leaves the FIFO unmodified on the first read	✓

All assertions are proven, but how good are they?

Apply OneSpin's Quantify observation coverage technology.

Quantify Coverage Report



Structural Coverage Overview

Status		Statements		Branches	
1	covered	15	40.54%	21	67.74%
R	reached	0	0.00%	0	0.00%
U	unknown	0	0.00%	0	0.00%
0R	unobserved	0	0.00%	0	0.00%
0	uncovered	22	59.46%	10	32.26%
0C	constrained	0	0.00%	0	0.00%
0D	dead	0	0.00%	0	0.00%
Sum	**quantify targets**	**37**	🟩🟥	**31**	🟩🟥

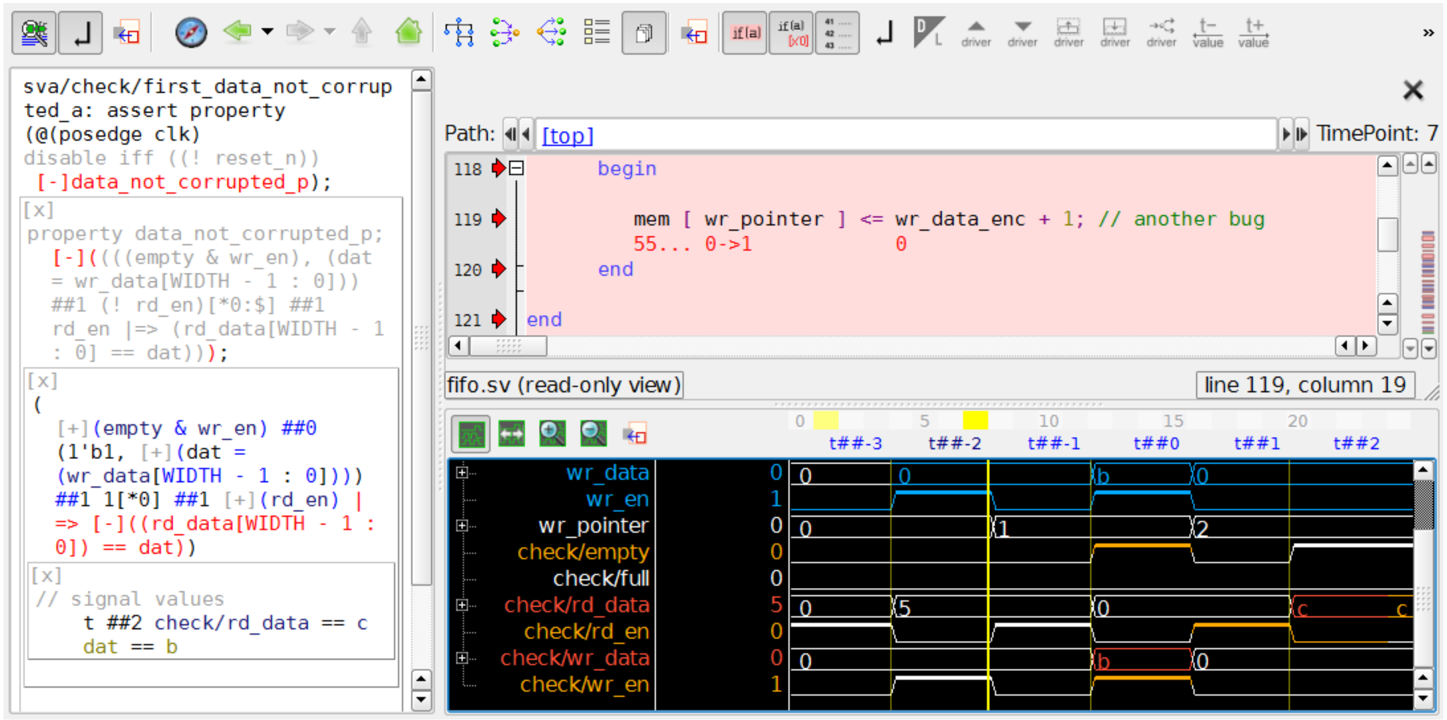
Structural Coverage by File

File	Statements		Branches	
decoder.v	12	🟥	5	🟥
encoder.v	5	🟥	0	
fifo_fix4.sv	20	🟩🟥	26	🟩🟥

Expecting FIFO to be fully covered!

Stronger Assertion Exposes Bug

```
property data_not_corrupted_p;
... (empty & wr_en, dat=wr_data[WIDTH-1:0]) ##1
!rd_en[*0:$] ##1 rd_en |=> rd_data[WIDTH-1:0]==dat
|| (!full | empty); // Bad!
```



```
sva/check/first_data_not_corrup
ted_a: assert property
(@(posedge clk)
disable iff ((! reset_n))
 [-]data_not_corrupted_p);
[x]
property data_not_corrupted_p;
  [-]((((empty & wr_en), (dat
  = wr_data[WIDTH - 1 : 0]))
  ##1 (! rd_en)[*0:$] ##1
  rd_en |=> (rd_data[WIDTH - 1
  : 0] == dat));
[x]
(
  [+](empty & wr_en) ##0
  (1'b1, [+](dat =
  (wr_data[WIDTH - 1 : 0])))
  ##1 1[*0] ##1 [+](rd_en) |
  => [-]((rd_data[WIDTH - 1 :
  0]) == dat))
[x]
// signal values
     t ##2 check/rd_data == c
     dat == b
```

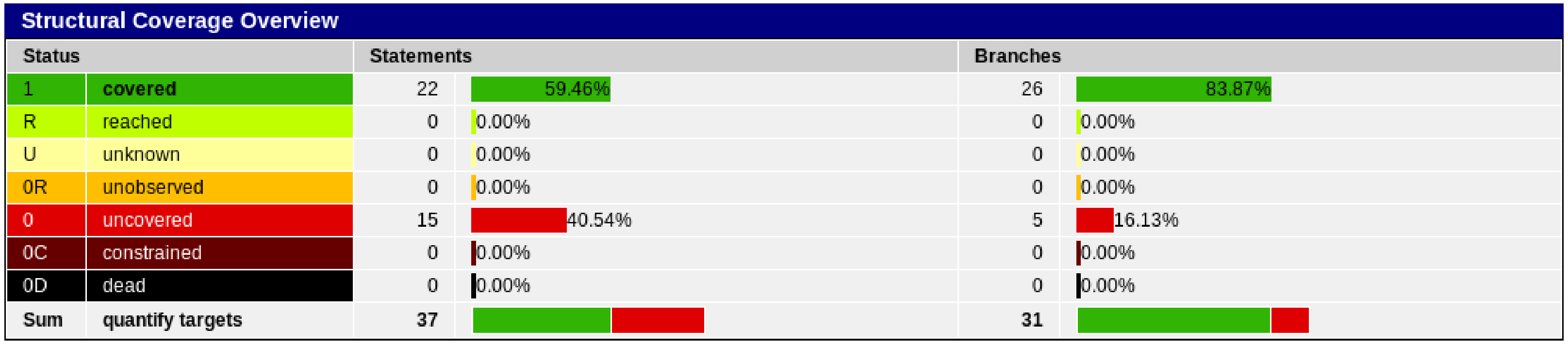
Path: [top] TimePoint: 7

```
118      begin
119         mem [ wr_pointer ] <= wr_data_enc + 1; // another bug
            55... 0->1            0
120      end
121   end
```

fifo.sv (read-only view)   line 119, column 19

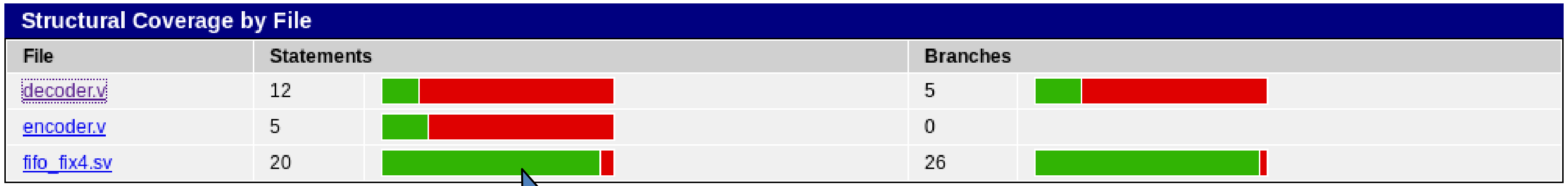
```
                 0     5     10     15     20
              t##-3 t##-2 t##-1  t##0  t##1  t##2
wr_data        0   0     0           b    0
wr_en          1
wr_pointer     0   0           1          2
check/empty    0
check/full     0
check/rd_data  5   0     5           0          c      c
check/rd_en    0
check/wr_data  0   0                 b    0
check/wr_en    1
```

Quantify Coverage Report



Structural Coverage Overview

Status		Statements		Branches	
1	covered	22	59.46%	26	83.87%
R	reached	0	0.00%	0	0.00%
U	unknown	0	0.00%	0	0.00%
0R	unobserved	0	0.00%	0	0.00%
0	uncovered	15	40.54%	5	16.13%
0C	constrained	0	0.00%	0	0.00%
0D	dead	0	0.00%	0	0.00%
Sum	**quantify targets**	**37**	🟩🟥	**31**	🟩🟥



Structural Coverage by File

File	Statements		Branches	
decoder.v	12	🟩🟥	5	🟩🟥
encoder.v	5	🟩🟥	0	
fifo_fix4.sv	20	🟩🟥	26	🟩🟥

Much better after fix!
But still something wrong.
Visit our booth P6 for full demo!

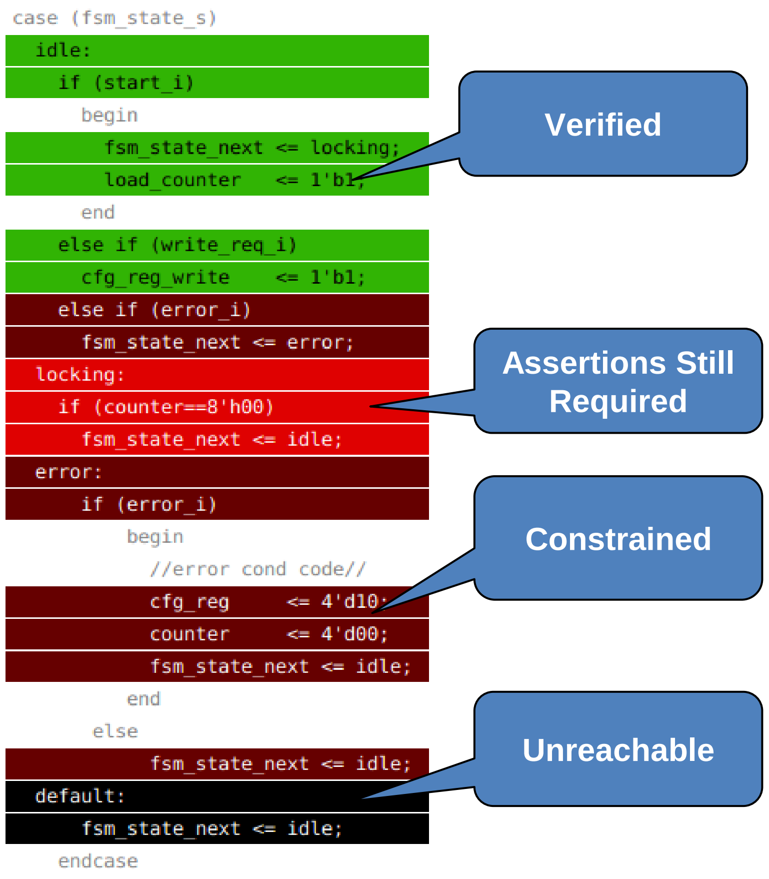
Quantify Properties

Functional Requirements	Assertions Hold	Coverage Achieved
The FIFO is not full and empty at the same time	✓	✓
The FIFO is empty after DEPTH many reads without writes	✓	✓
The FIFO is full after DEPTH many writes without reads	✓	✓
The FIFO is no longer empty after a write	✓	✓
The first data written to an empty FIFO leaves the FIFO unmodified on the first read	✓	✓

Summary Observation Coverage with Quantify

- Observation coverage algorithm drives precise coverage metric
  - Qualifies for safety-critical
  - Also identifies dead code and over-constrained code
  - Provides comprehensive progress metric

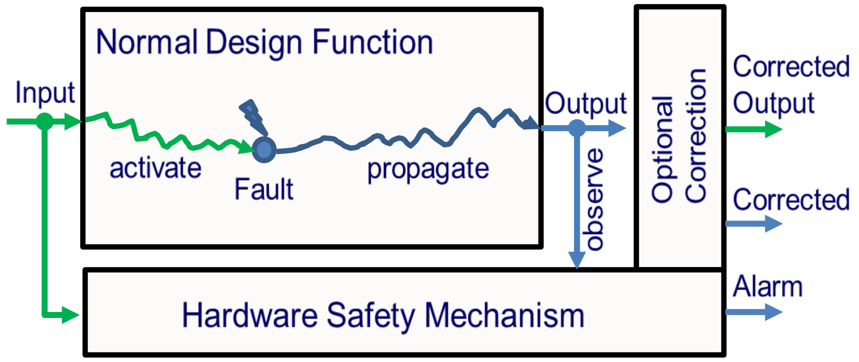
- Don't trust COI coverage
  - Maybe good for sanity/quick check
  - But not for safety-critical
- Prover coverage is also problematic for safety-critical
  - Not objective, results depend on prove engine



```
case (fsm_state_s)
  idle:
    if (start_i)
      begin
        fsm_state_next <= locking;
        load_counter   <= 1'b1;
      end
    else if (write_req_i)
      cfg_reg_write    <= 1'b1;
    else if (error_i)
      fsm_state_next <= error;
  locking:
    if (counter==8'h00)
      fsm_state_next <= idle;
  error:
      if (error_i)
          begin
            //error cond code//
            cfg_reg        <= 4'd10;
            counter        <= 4'd00;
            fsm_state_next <= idle;
          end
       else
            fsm_state_next <= idle;
  default:
      fsm_state_next <= idle;
    endcase
```

Verification of Safety Mechanism

Efficient Verification of Safety Functions



Safety Verification Problem

- Safety functions are **inactive under normal operation**!
- Artificially **inject faults** into verification to activate

Fault Injection complexity for bit vectors:

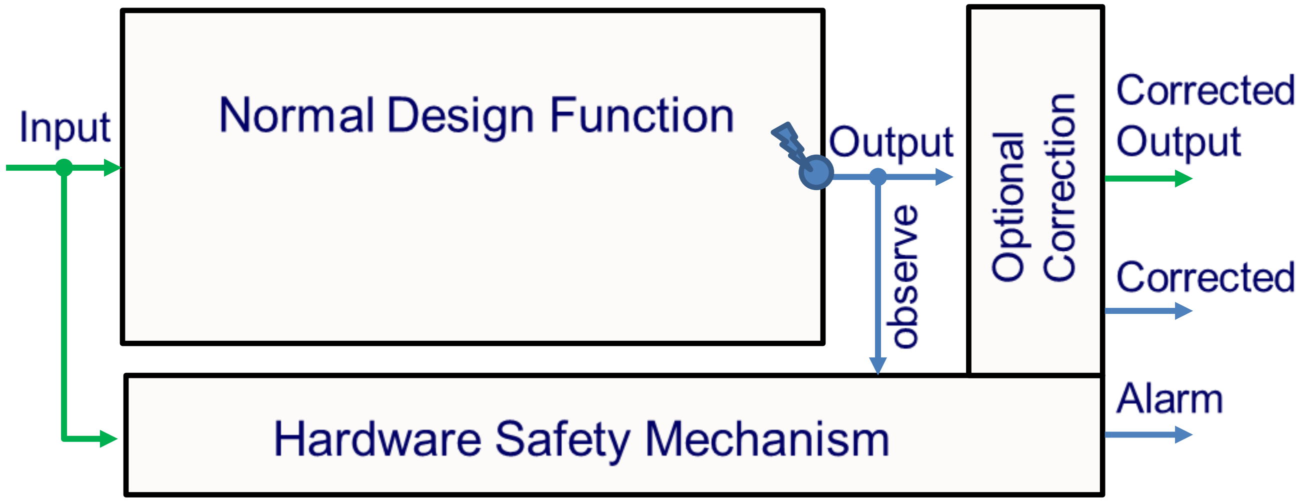
- *2^{width}* possible data input combinations
- *(width)* 1-bit errors
- *(width* width-1)* 2-bit errors

Simulation Based Verification is not a good solution:

- Hard to anticipate all relevant conditions
- **Hard to deal with huge number of faults + combinations!**
- No exhaustive testing feasible

Formal ABV with fault injection

Three Simple Steps to Success



1. Describe expected behavior with no fault injected and prove that it holds. ✓
   ```
   property (<antecedent> |=> !Alarm)
   ```
2. Describe expected behavior with the fault(s) injected, inject the fault(s) and prove that it holds. ✓
   ```
   property (<antecedent> |=> Alarm)
   ```
3. Describe expected behavior with correctable faults injected, inject the correctable faults and prove that it holds. ✓
   ```
   property (<antecedent> |=> Input'== CorrectedOutput
                 & Corrected & !Alarm
   ```

FIFO Safety Requirements

Functional Requirements

The FIFO is not full and empty at the same time

The FIFO is empty after DEPTH many reads without writes

The FIFO is full after DEPTH many writes without reads

The FIFO is no longer empty after a write

The first data written to an empty FIFO leaves the FIFO unmodified on the first read

Safety Requirements

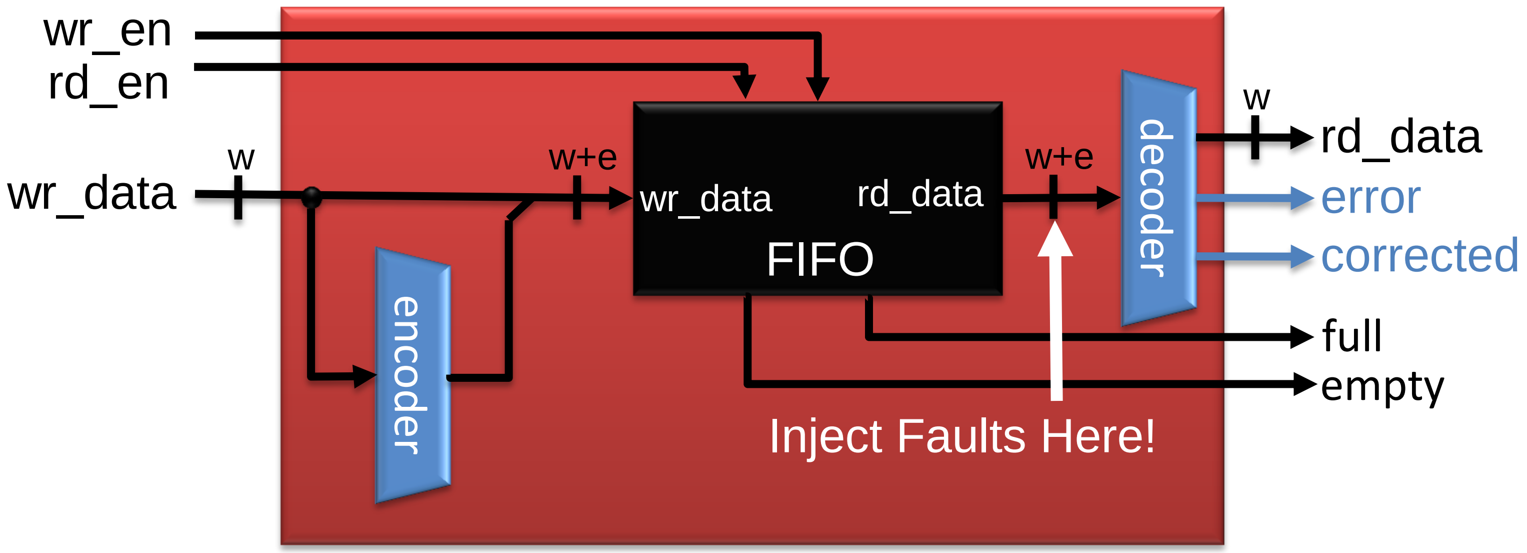
If no error occurs, nothing is flagged and the data is uncorrupted

If one error occurs, no error is flagged, the data is uncorrupted and the correction is flagged

If two errors occur, an error is flagged, but no correction

Formal ABV with Fault Injection Application Scenario: FIFO

- For FIFO Example:
  - Create no_error, corrected_no_error and error assertions according to the safety requirements
  - Depening on the assertion, inject Bit-Flip faults at the FIFO output



Application Scenario: FIFO
SV Assertions for Safety Features

No error → nothing flagged, data uncorrupted:

```
no_error: assert property (disable iff (!reset_n)
                    empty & wr_en ##1 rd_en
             |=> rd_data == $past(wr_data,2) & !rd_error & !rd_corrected);
```

One error → no error flagged, data uncorrupted, correction flagged:

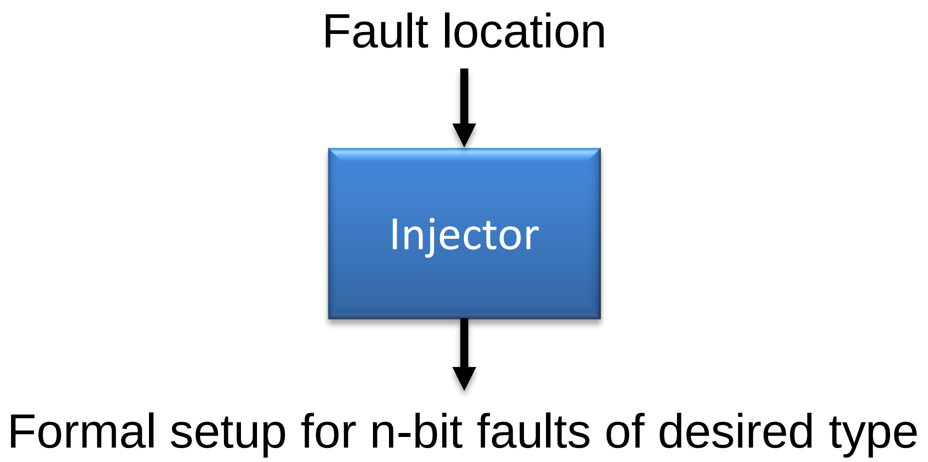
```
corrected_no_error: assert property (disable iff (!reset_n)
                    empty & wr_en ##1 rd_en
             |=> rd_data == $past(wr_data,2) & !rd_error & rd_corrected);
```

Two errors → error flagged, no correction flagged:

```
error: assert property (disable iff (!reset_n)
                    empty & wr_en ##1 rd_en
             |=> rd_error && !rd_corrected);
```

How to inject the faults?

- Conveniently use formal fault injection:



- User can automatically enable different number/kind of faults for individual assertions
- Possible to verify generic assertions like “a 2-bit fault gets detected”
- Supporting FLIP, ST0, ST1, OPEN

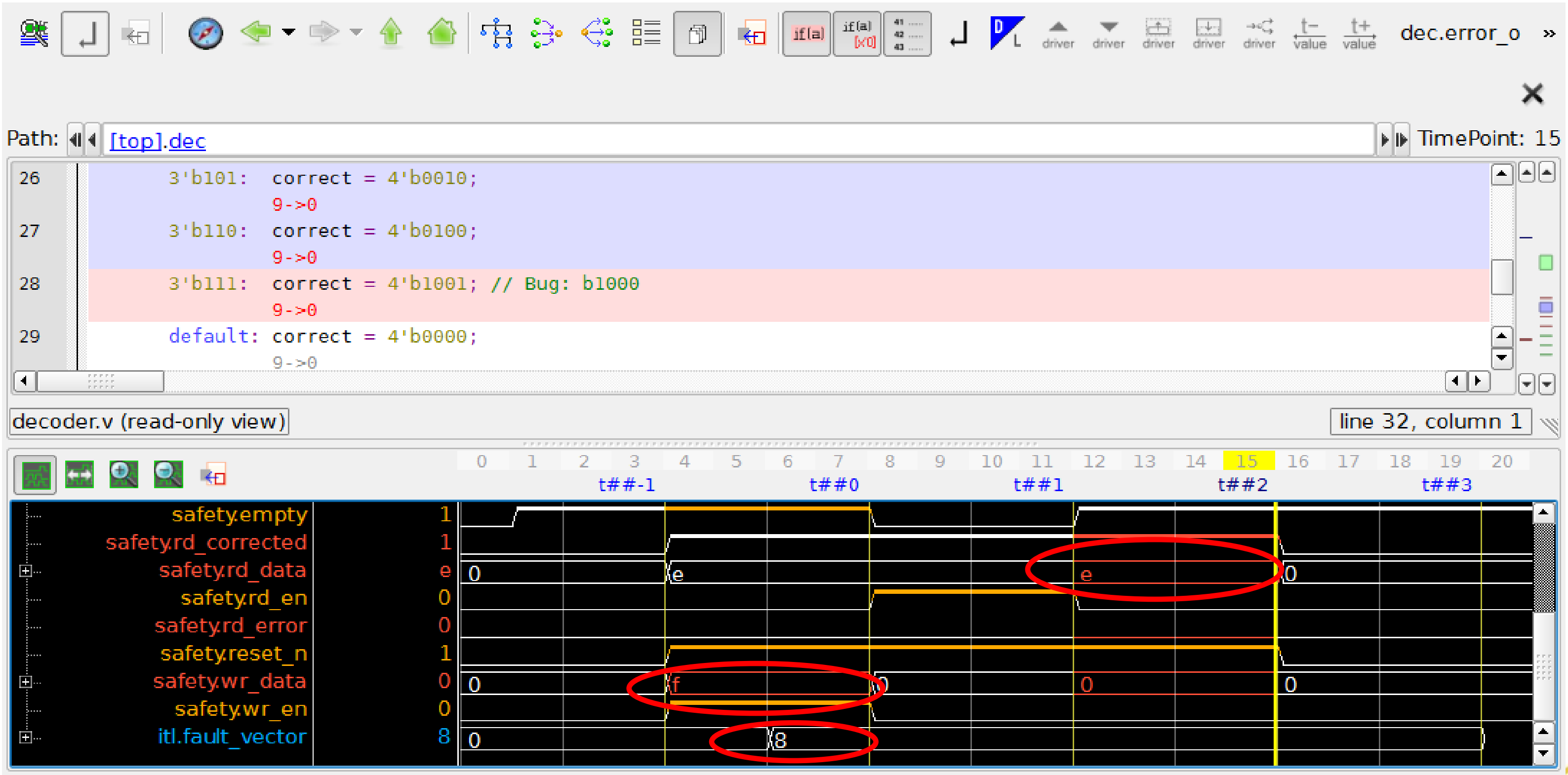
Using the Formal Fault Injection

```
inject_fault -location rd_data_FIFO -type <type> -assert <assertion>
```

Assertion	Inject Fault Type	Expect
`safety.no_error`	NONE	HOLD
`safety.corrected_no_error`	FLIP 1 bit	HOLD
`safety.error`	FLIP 2 bit	HOLD

Application Scenario: FIFO Failing Assertion for Safety Feature

Assertion	Inject Fault Type	Got
`safety.corrected_no_error`	FLIP 1 bit	FAIL


Path: [top].dec TimePoint: 15

```
26        3'b101:  correct = 4'b0010;
                   9->0
27        3'b110:  correct = 4'b0100;
                   9->0
28        3'b111:  correct = 4'b1001; // Bug: b1000
                   9->0
29        default: correct = 4'b0000;
                   9->0
```

decoder.v (read-only view) line 32, column 1

```
safety.empty         1
safety.rd_corrected  1
safety.rd_data       e
safety.rd_en         0
safety.rd_error      0
safety.reset_n       1
safety.wr_data       0
safety.wr_en         0
itl.fault_vector     8
```

FIFO Safety Requirements

Functional Requirements

- The FIFO is not full and empty at the same time
- The FIFO is empty after DEPTH many reads without writes
- The FIFO is full after DEPTH many writes without reads
- The FIFO is no longer empty after a write
- The first data written to an empty FIFO leaves the FIFO unmodified on the first read

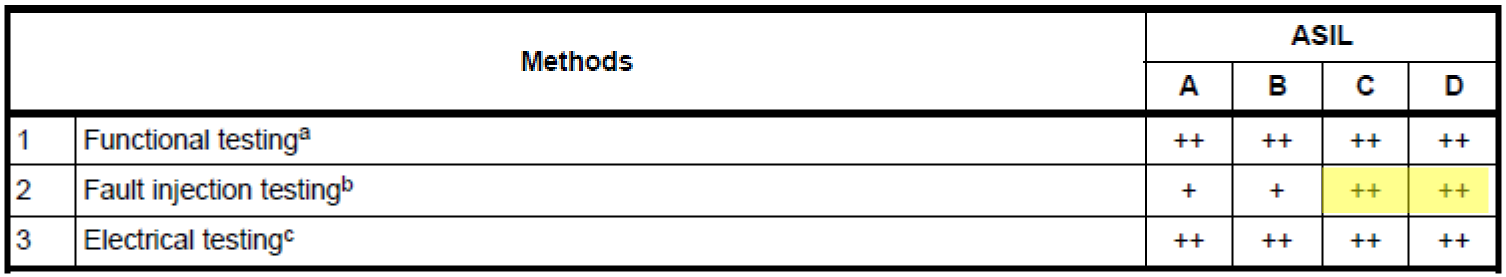
Safety Requirements

- If no error occurs, nothing is flagged and the data is uncorrupted ✓
- If one error occurs, no error is flagged, the data is uncorrupted and the correction is flagged ✓
- If two errors occur, an error is flagged, but no correction ✓

Summary Verification of Safety Mechanism

- ISO 26262-5 (page 28) highly recommends to apply model based fault injection testing:

Table 11 — Hardware integration tests to verify the completeness and correctness of the safety mechanisms implementation with respect to the hardware safety requirements



Methods		ASIL			
		A	B	C	D
1	Functional testing[a]	++	++	++	++
2	Fault injection testing[b]	+	+	++	++
3	Electrical testing[c]	++	++	++	++

- OneSpin provides formal fault injection to meet ISO 26262 and verify safety mechanisms
  - No modification of source code required
  - Supports different fault types and number of faults
  - Unlike simulation, it provides complete proof of all faults in one step
  - Easily maps assertions to faults and checks them

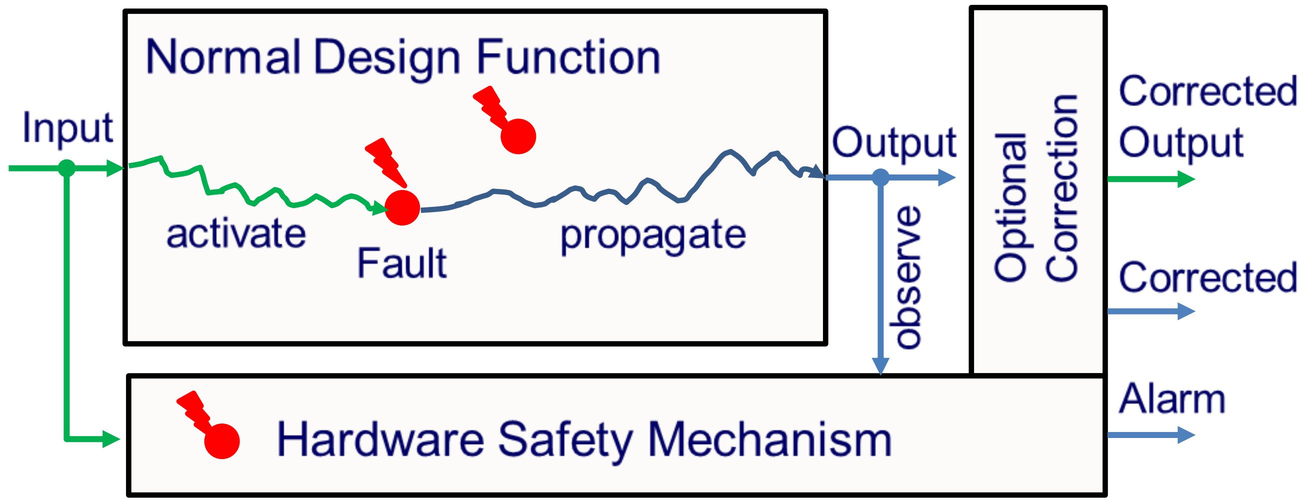
Diagnostic Coverage

Diagnostic Coverage
ISO 26262 Analysis Requirements

- Diagnostic coverage: proportion of hardware element failure rate that is detected or controlled by safety mechanisms

- High diagnostic coverage is needed to achieve a high Automotive Safety Integrity Level (ASIL)

Discussing Diagnostic Coverage of Safety Mechanisms



Fault Classification in Semiconductor Context

- Safe faults
  - Faults which cannot propagate
  - Faults which only propagate to non-safety-critical functions (don't violate a safety goal)
  - Faults which are detected by a safety mechanism before they can cause harm
- Unsafe faults
  - Faults which propagate to a safety-critical function without being detected
  - Faults with unknown behavior

Minimize Unsafe Faults → **Increase Diagnostic Coverage**

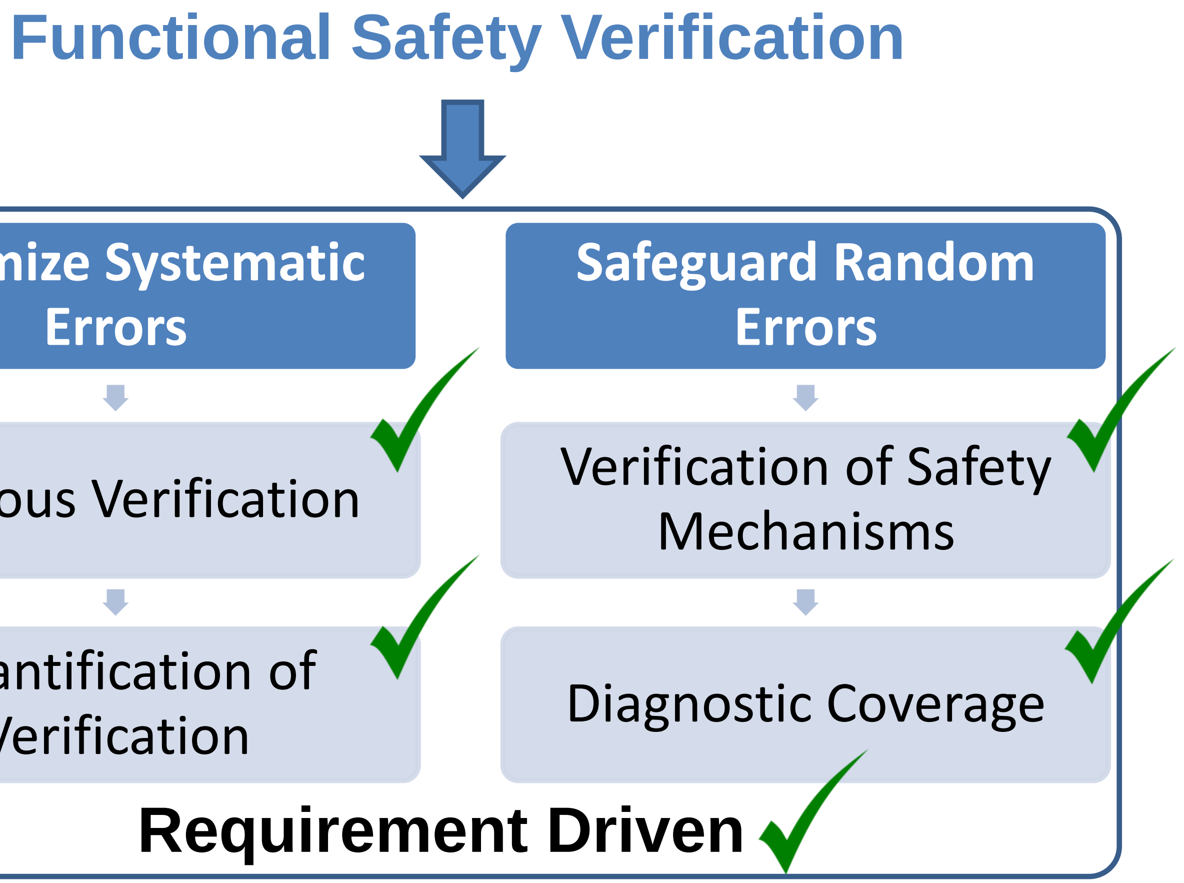
Formal Propagation Analysis Summary

- Formal propagation analysis can identify
  - Faults which cannot propagate
  - Whether a fault propagates to a safety-critical function
  - Whether a fault propagates to a safety mechanism

- This information helps to classify faults as safe or unsafe and creates more precise diagnostic coverage of the safety mechanism

More Precise Diagnostic Coverage → **Meet Safety Goal**

Summary


Functional Safety Verification
Minimize Systematic Errors
Rigorous Verification
Quantification of Verification
Safeguard Random Errors
Verification of Safety Mechanisms
Diagnostic Coverage
Requirement Driven

Thank you!

To learn more about safety critical design & verification:

- Read Safety Critical News
  - http://safetycritical.onespin-solutions.com/
- Visit us at Booth P6

Questions

Finalize slide set with questions slide

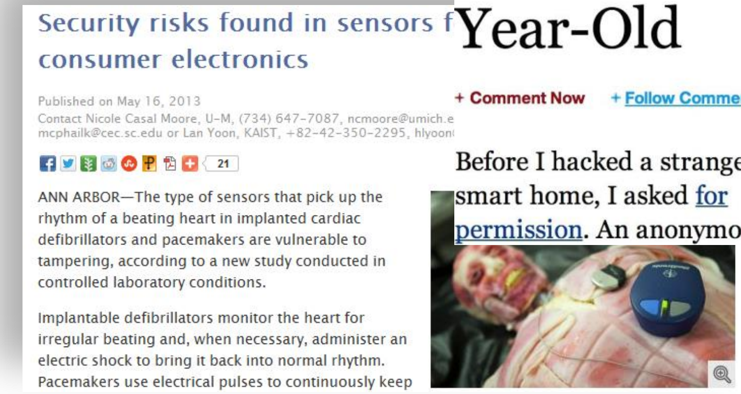
TVS Agenda

- **11.00 Introductions**
- **11.05 TVS**
  - Safety and security in Hardware and Software
  - Requirements Driven Test and Verification (RDTV)
  - Using an ECC example and breaking it down into a test plan
- **11.20 OneSpin**
- **11.55 Tortuga Logic**
- **12.10 TVS**
  - Analysing the results and signoff
  - Advantages of RDTV
- **12.25 Q&A (TVS, OneSpin, Tortuga Logic)**

Medtronic

mazda

Not just these devices….



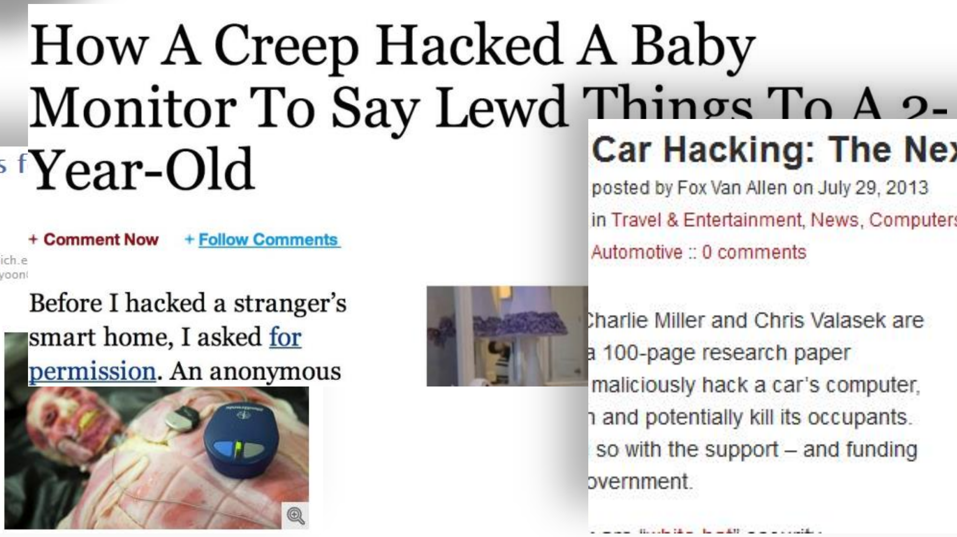
Security risks found in sensors f
consumer electronics

Published on May 16, 2013
Contact Nicole Casal Moore, U-M, (734) 647-7087, ncmoore@umich.e
mcphailk@cec.sc.edu or Lan Yoon, KAIST, +82-42-350-2295, hlyoon

21

ANN ARBOR—The type of sensors that pick up the rhythm of a beating heart in implanted cardiac defibrillators and pacemakers are vulnerable to tampering, according to a new study conducted in controlled laboratory conditions.

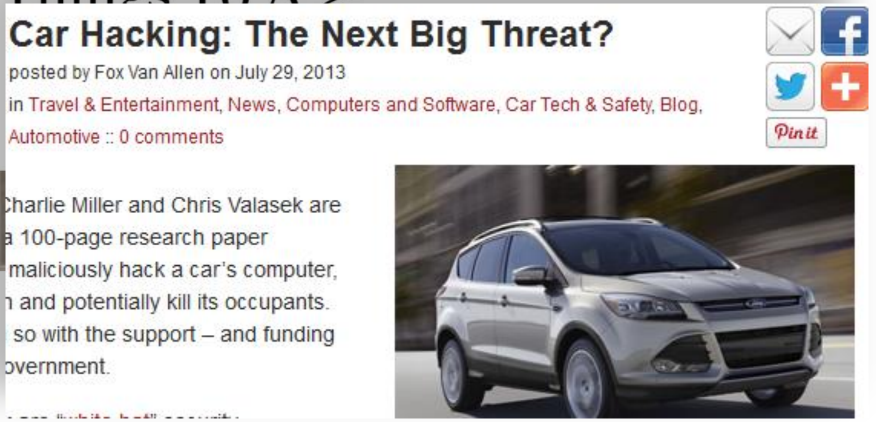
Implantable defibrillators monitor the heart for irregular beating and, when necessary, administer an electric shock to bring it back into normal rhythm. Pacemakers use electrical pulses to continuously keep



How A Creep Hacked A Baby Monitor To Say Lewd Things To A 2-Year-Old

+ Comment Now + Follow Comments

Before I hacked a stranger's smart home, I asked for permission. An anonymous



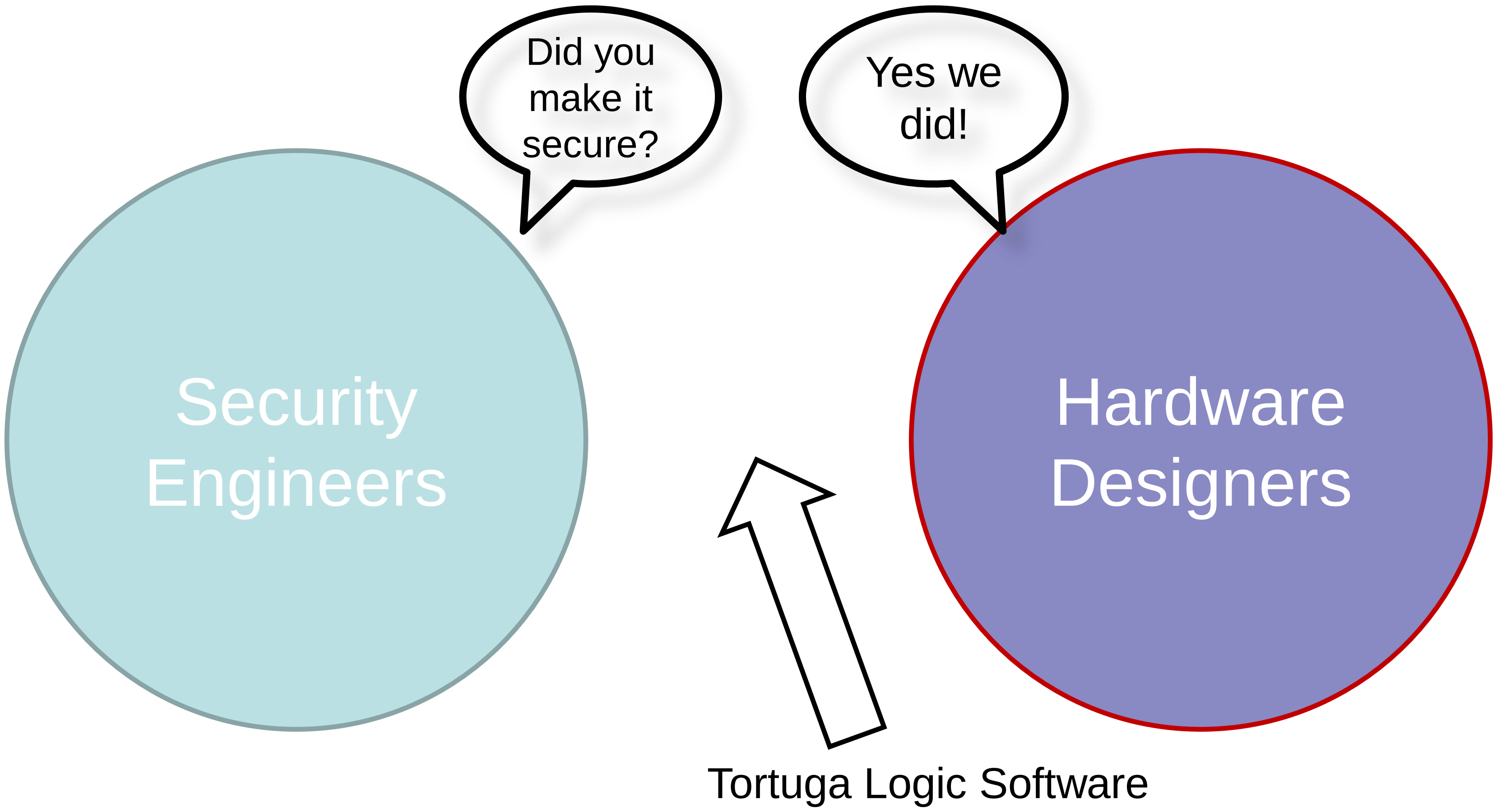
Car Hacking: The Next Big Threat?

posted by Fox Van Allen on July 29, 2013
in Travel & Entertainment, News, Computers and Software, Car Tech & Safety, Blog, Automotive :: 0 comments

Charlie Miller and Chris Valasek are
a 100-page research paper
maliciously hack a car's computer,
n and potentially kill its occupants.
so with the support – and funding
overnment.

Hackers are now focusing on hardware

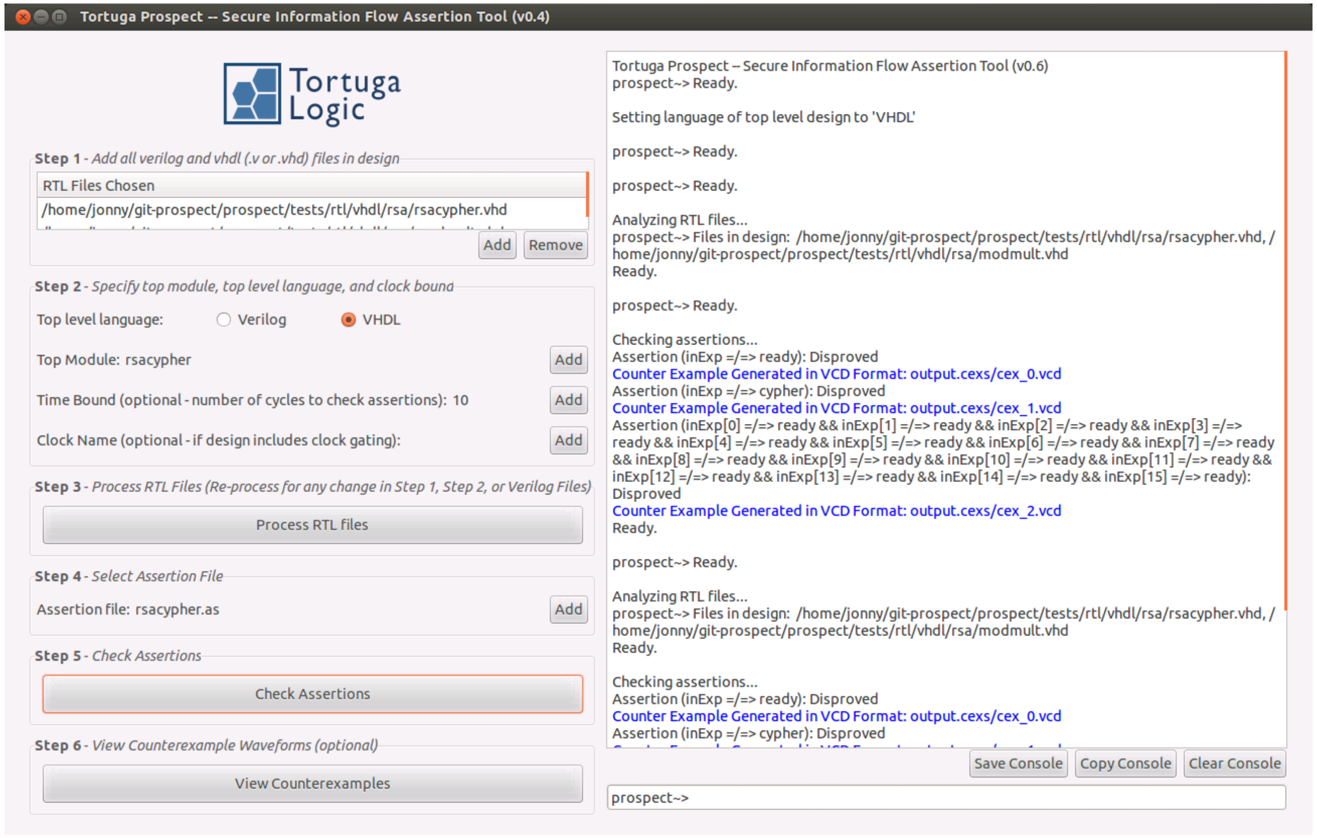
Current "State-Of-The-Art"
Designing Secure Hardware



Tortuga Logic's PROSPECT

Enable "Design-for-Security" from the ground up to minimize security breaches in hardware and systems

Tortuga Logic's **PROSPECT *Software Solution***


Tortuga Prospect -- Secure Information Flow Assertion Tool (v0.4)
Tortuga Logic

Step 1 - *Add all verilog and vhdl (.v or .vhd) files in design*

RTL Files Chosen
/home/jonny/git-prospect/prospect/tests/rtl/vhdl/rsa/rsacypher.vhd

Step 2 - *Specify top module, top level language, and clock bound*

Top level language: Verilog [✓ VHDL]

Top Module: rsacypher

Time Bound (optional - number of cycles to check assertions): 10

Clock Name (optional - if design includes clock gating):

Step 3 - *Process RTL Files (Re-process for any change in Step 1, Step 2, or Verilog Files)*

Step 4 - *Select Assertion File*

Assertion file: rsacypher.as

Step 5 - *Check Assertions*

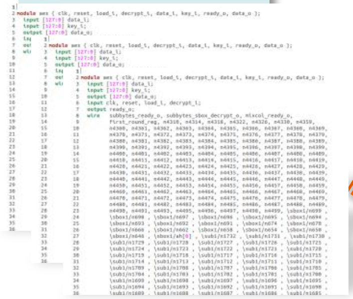
Step 6 - *View Counterexample Waveforms (optional)*

```
Tortuga Prospect – Secure Information Flow Assertion Tool (v0.6)
prospect~> Ready.

Setting language of top level design to 'VHDL'

prospect~> Ready.

prospect~> Ready.

Analyzing RTL files...
prospect~> Files in design: /home/jonny/git-prospect/prospect/tests/rtl/vhdl/rsa/rsacypher.vhd, /
home/jonny/git-prospect/prospect/tests/rtl/vhdl/rsa/modmult.vhd
Ready.

prospect~> Ready.

Checking assertions...
Assertion (inExp =/=> ready): Disproved
Counter Example Generated in VCD Format: output.cexs/cex_0.vcd
Assertion (inExp =/=> cypher): Disproved
Counter Example Generated in VCD Format: output.cexs/cex_1.vcd
Assertion (inExp[0] =/=> ready && inExp[1] =/=> ready && inExp[2] =/=> ready && inExp[3] =/=>
ready && inExp[4] =/=> ready && inExp[5] =/=> ready && inExp[6] =/=> ready && inExp[7] =/=> ready
&& inExp[8] =/=> ready && inExp[9] =/=> ready && inExp[10] =/=> ready && inExp[11] =/=> ready &&
inExp[12] =/=> ready && inExp[13] =/=> ready && inExp[14] =/=> ready && inExp[15] =/=> ready):
Disproved
Counter Example Generated in VCD Format: output.cexs/cex_2.vcd
Ready.

prospect~> Ready.

Analyzing RTL files...
prospect~> Files in design: /home/jonny/git-prospect/prospect/tests/rtl/vhdl/rsa/rsacypher.vhd, /
home/jonny/git-prospect/prospect/tests/rtl/vhdl/rsa/modmult.vhd
Ready.

Checking assertions...
Assertion (inExp =/=> ready): Disproved
Counter Example Generated in VCD Format: output.cexs/cex_0.vcd
Assertion (inExp =/=> cypher): Disproved
```

Tortuga Logic's PROSPECT

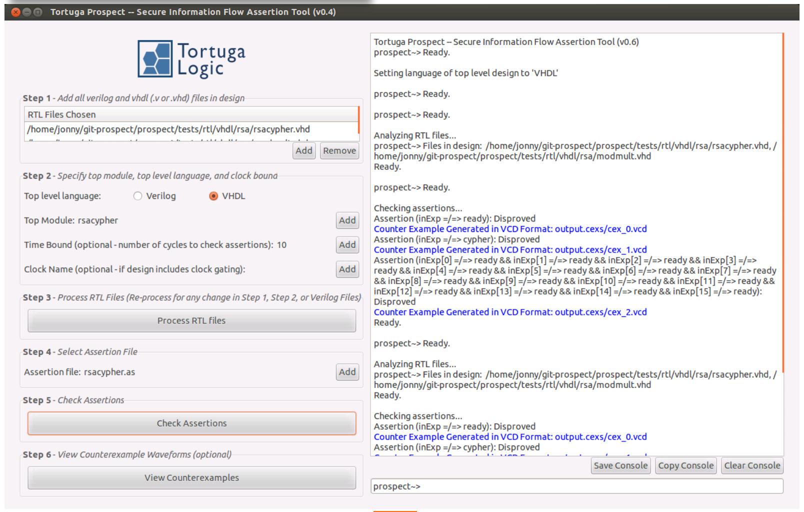
- **Prospect Tool flow**

RTL



```
module aes ( clk, reset, load_i, decrypt_i, data_i, key_i, ready_o, data_o );
  input [127:0] data_i;
  input [127:0] key_i;
  output [127:0] data_o;
  input clk, reset, load_i, decrypt_i;
  output ready_o;
  wire    subbytes_ready_o, subbytes_sbox_decrypt_o, mixcol_ready_o,
          first_round_reg, n4310, n4314, n4316, n4322, n4326, n4330, n4359,
          [illegible]
```

Prospect GUI



Step 1 - *Add all verilog and vhdl (.v or .vhd) files in design*

RTL Files Chosen
/home/jonny/git-prospect/prospect/tests/rtl/vhdl/rsa/rsacypher.vhd

Step 2 - *Specify top module, top level language, and clock bound*

Top level language: Verilog [✓ VHDL]

Top Module: rsacypher

Time Bound (optional - number of cycles to check assertions): 10

Clock Name (optional - if design includes clock gating):

Step 3 - *Process RTL Files (Re-process for any change in Step 1, Step 2, or Verilog Files)*

Step 4 - *Select Assertion File*

Assertion file: rsacypher.as

Step 5 - *Check Assertions*

Step 6 - *View Counterexample Waveforms (optional)*

```
Tortuga Prospect -- Secure Information Flow Assertion Tool (v0.6)
prospect~> Ready.

Setting language of top level design to 'VHDL'

prospect~> Ready.

prospect~> Ready.

Analyzing RTL files...
prospect~> Files in design:  /home/jonny/git-prospect/prospect/tests/rtl/vhdl/rsa/rsacypher.vhd, /
home/jonny/git-prospect/prospect/tests/rtl/vhdl/rsa/modmult.vhd
Ready.

prospect~> Ready.

Checking assertions...
Assertion (inExp =/=> ready): Disproved
Counter Example Generated in VCD Format: output.cexs/cex_0.vcd
Assertion (inExp =/=> cypher): Disproved
Counter Example Generated in VCD Format: output.cexs/cex_1.vcd
Assertion (inExp[0] =/=> ready && inExp[1] =/=> ready && inExp[2] =/=> ready && inExp[3] =/=>
ready && inExp[4] =/=> ready && inExp[5] =/=> ready && inExp[6] =/=> ready && inExp[7] =/=> ready
&& inExp[8] =/=> ready && inExp[9] =/=> ready && inExp[10] =/=> ready && inExp[11] =/=> ready &&
inExp[12] =/=> ready && inExp[13] =/=> ready && inExp[14] =/=> ready && inExp[15] =/=> ready):
Disproved
Counter Example Generated in VCD Format: output.cexs/cex_2.vcd
Ready.

prospect~> Ready.

Analyzing RTL files...
prospect~> Files in design:  /home/jonny/git-prospect/prospect/tests/rtl/vhdl/rsa/rsacypher.vhd, /
home/jonny/git-prospect/prospect/tests/rtl/vhdl/rsa/modmult.vhd
Ready.

Checking assertions...
Assertion (inExp =/=> ready): Disproved
Counter Example Generated in VCD Format: output.cexs/cex_0.vcd
Assertion (inExp =/=> cypher): Disproved
```

```
prospect~>
```

Security Properties

key =/=> out
coreA =/=> coreB
. . .

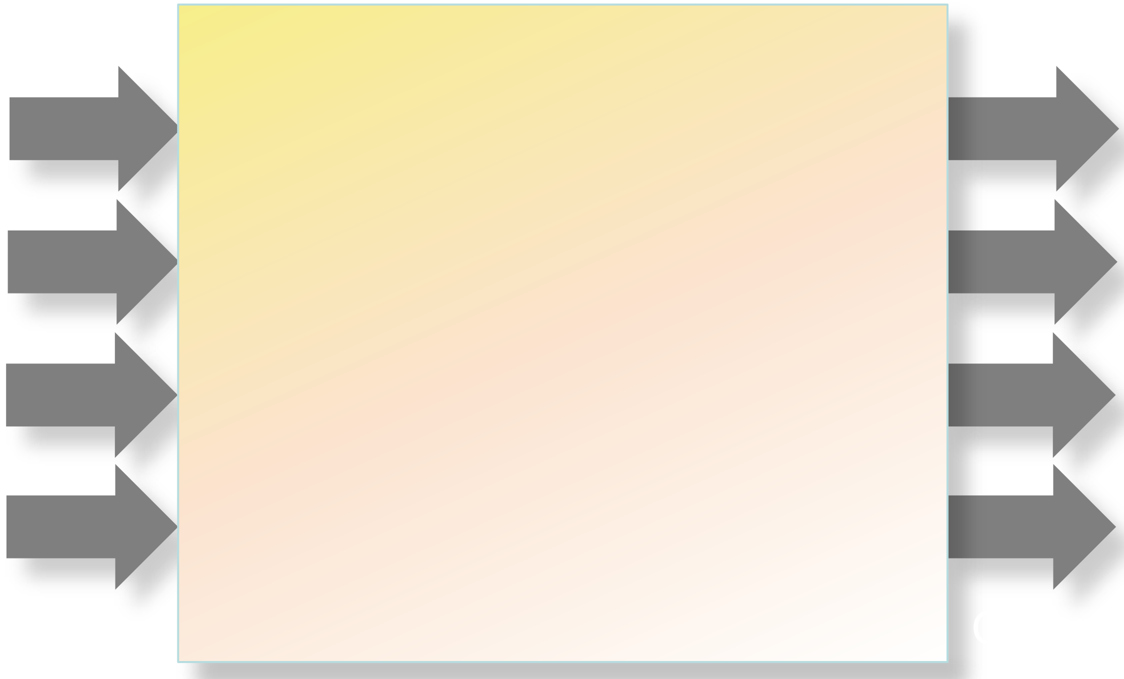
Results and Debug feedback

PROSPECT: Key Values

- **Automates HW security design**
  - Reduce security validation from months to hours
  - Significant cost savings for certification
- **Increase security coverage and reduce risk**
  - Many checks cannot be done manually
- **Makes design for security a priority**

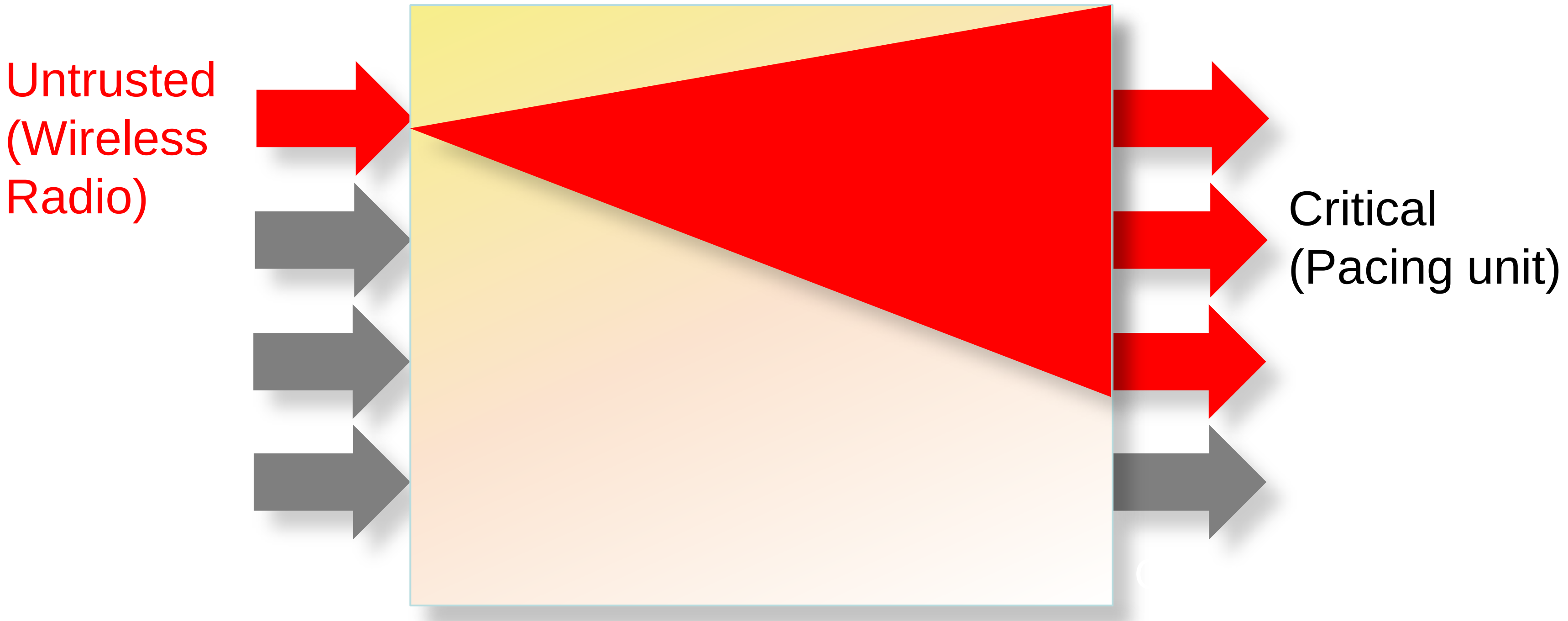
Tortuga Logic's PROSPECT

- **Types of addressable security properties**



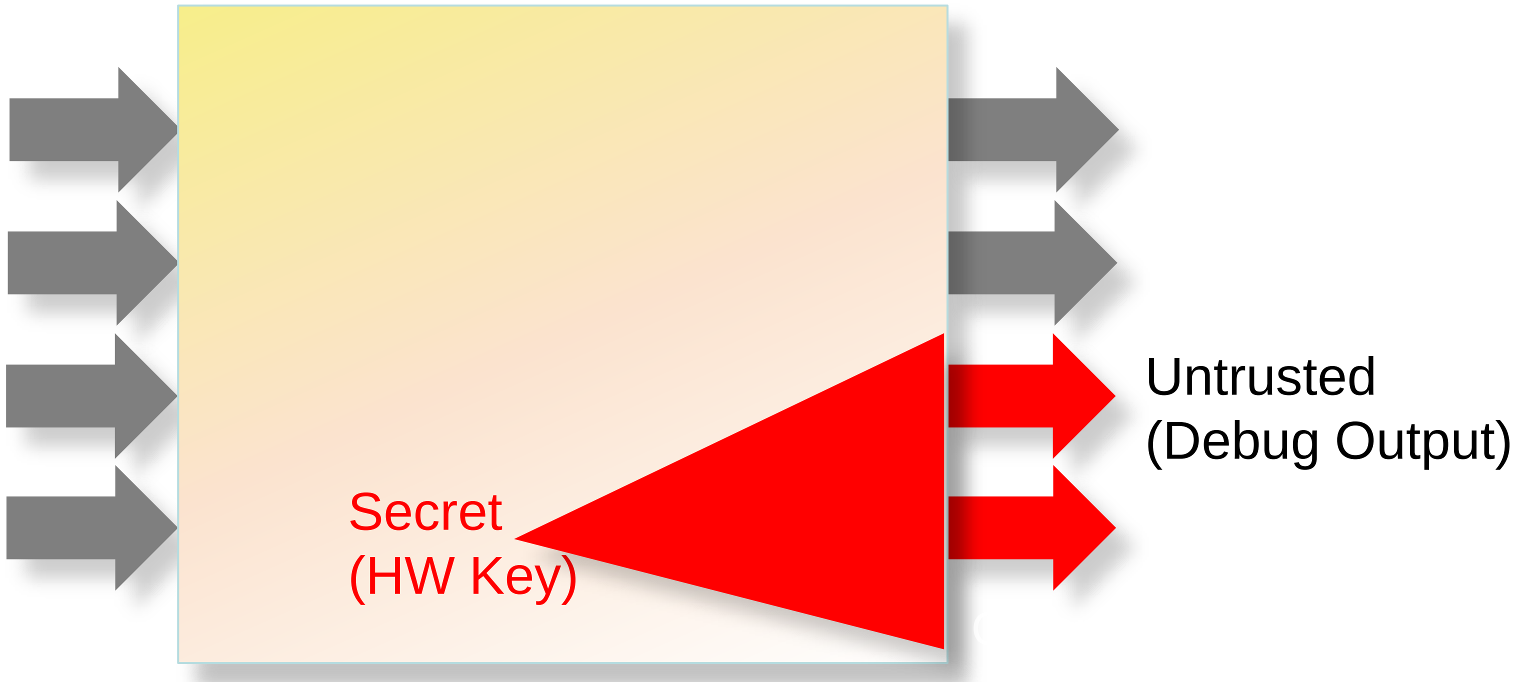
Tortuga Logic's PROSPECT

- **Critical component is adversely affected**



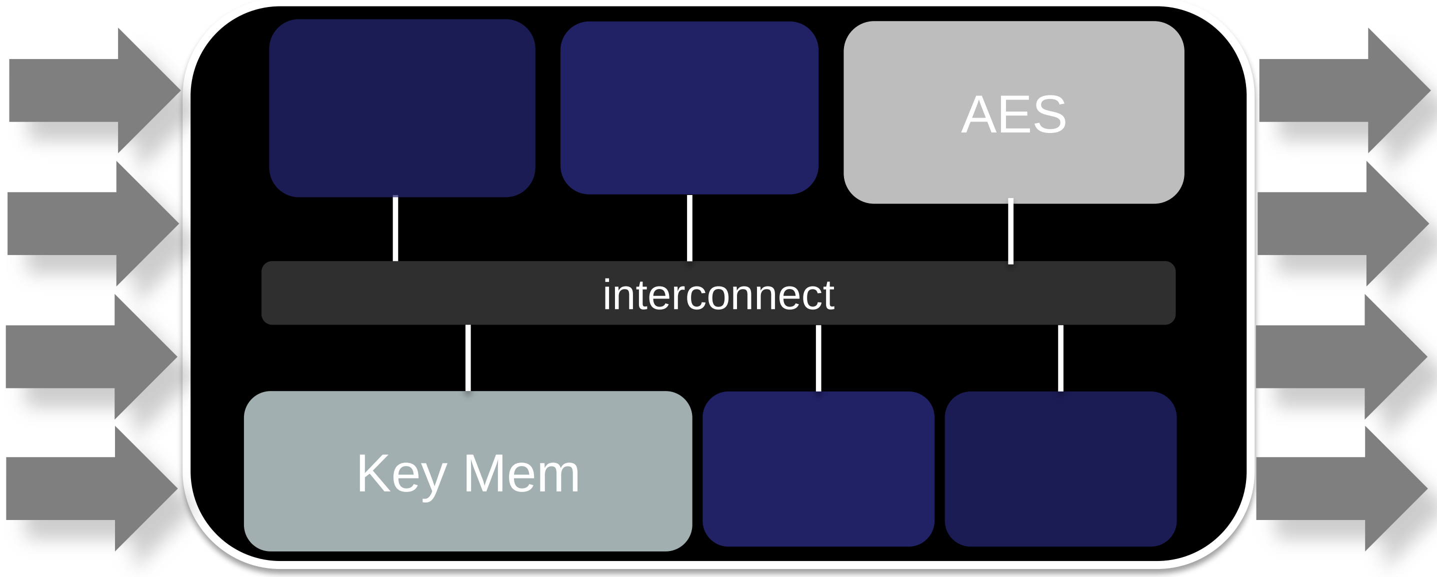
Tortuga Logic's PROSPECT

- **Secret data is unintentionally leaked**



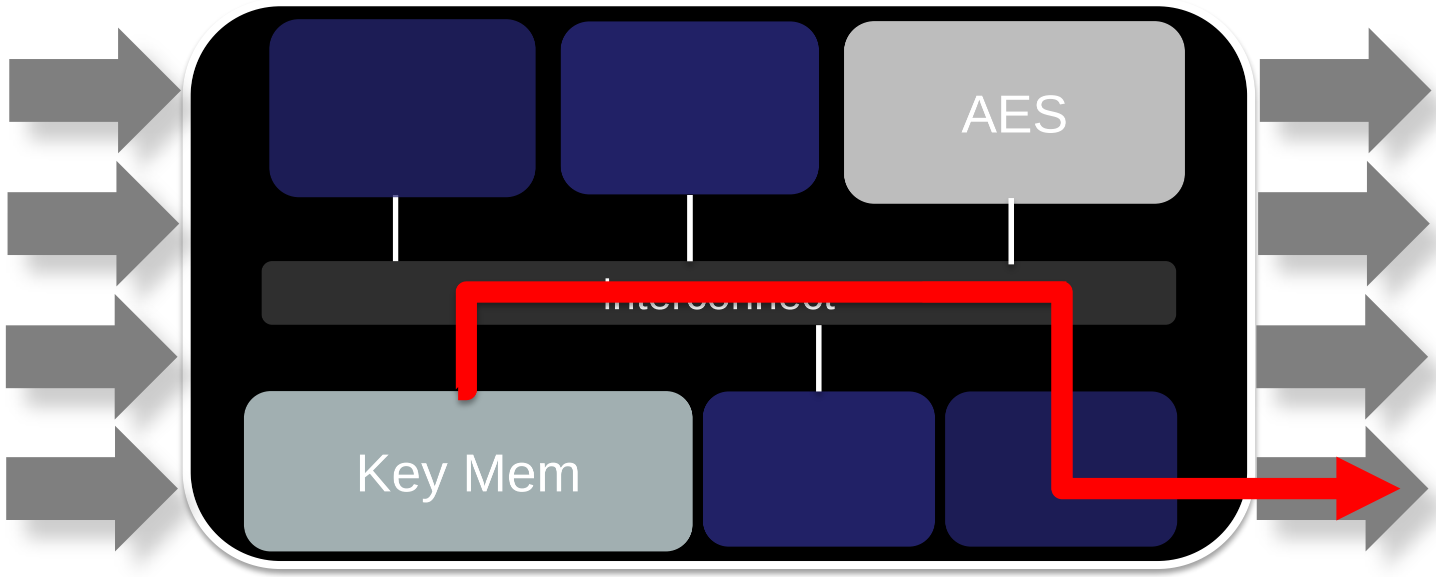
Case Study – Top-25 Semi Company Key Flowing Out Of Design

- **Assertion: Key only flows through AES**
  - *assert iflow (key =/=> $all_outputs ignoring aes.$all_outputs);*
  - If assertion holds, key only flows to outputs through AES first
- **Real world results**
  - State-of-the-art design with over 10 million gates
  - Actual required properties, impossible to visually inspect



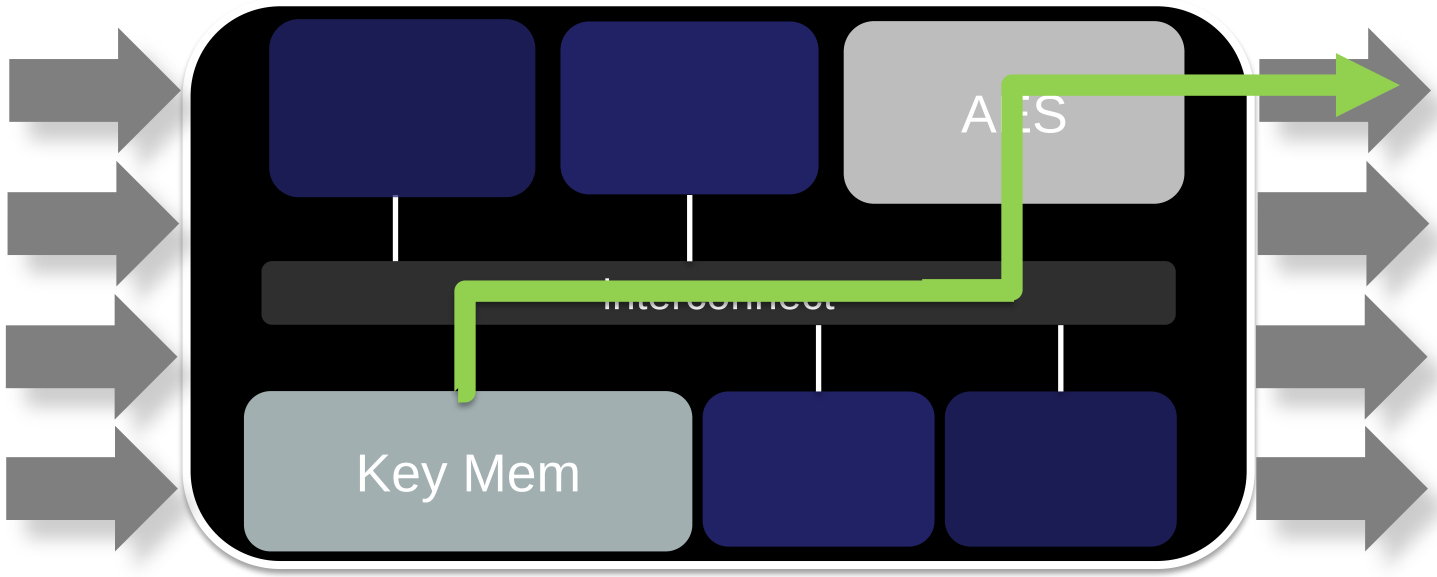
Case Study – Top-25 Semi Company Key Flowing Out Of Design

- **Assertion: Key only flows through AES**
  - *assert iflow (key =/=> $all_outputs ignoring aes.$all_outputs);*
  - If assertion holds, key only flows to outputs through AES first
- **Real world results**
  - State-of-the-art design with over 10 million gates
  - Actual required properties, impossible to visually inspect



Case Study – Top-25 Semi Company Key Flowing Out Of Design

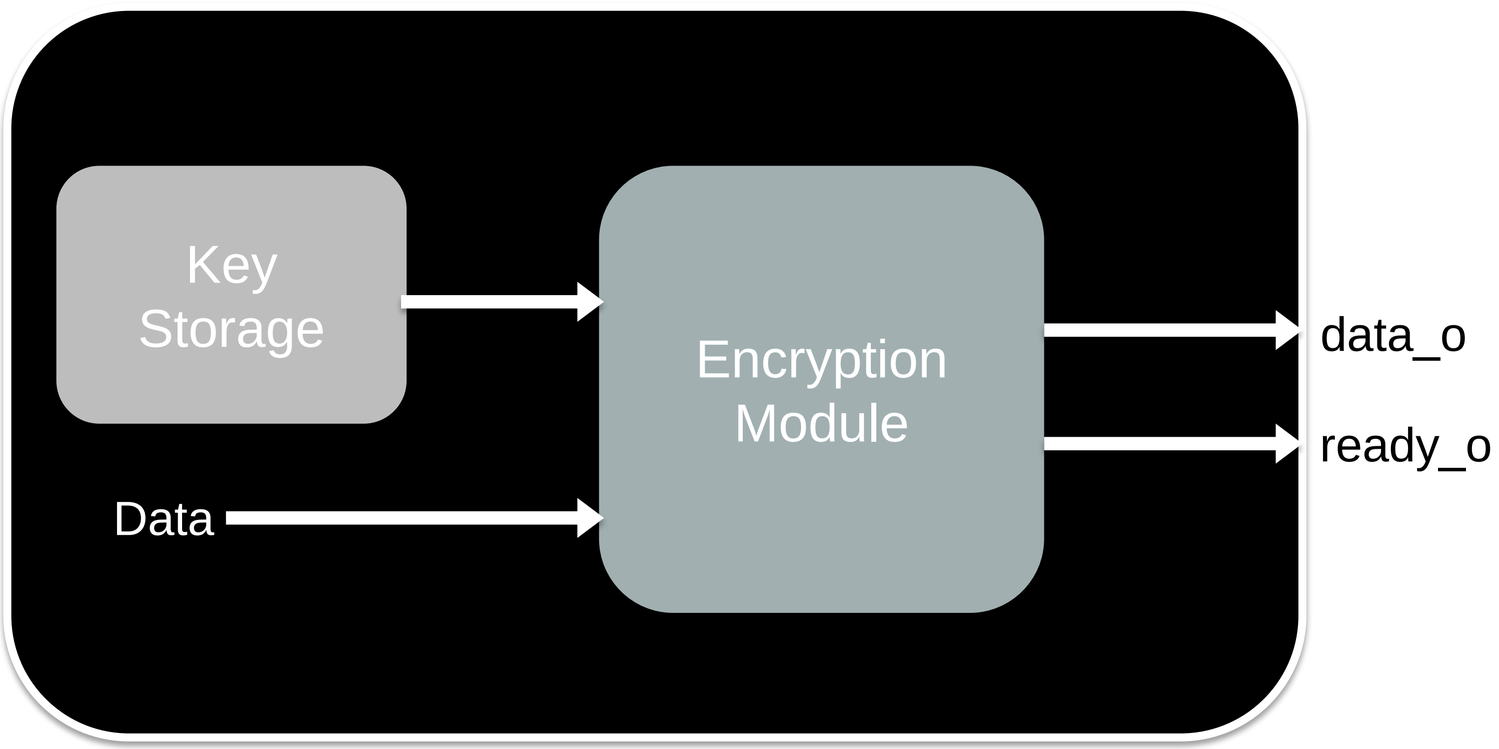
- **Assertion: Key only flows through AES**
  - *assert iflow (key =/=> $all_outputs ignoring aes.$all_outputs);*
  - If assertion holds, key only flows to outputs through AES first
- **Real world results**
  - State-of-the-art design with over 10 million gates
  - Actual required properties, impossible to visually inspect



Demo: AES Key Leakage

Property:
assert iflow (*key =/=> data_o*);

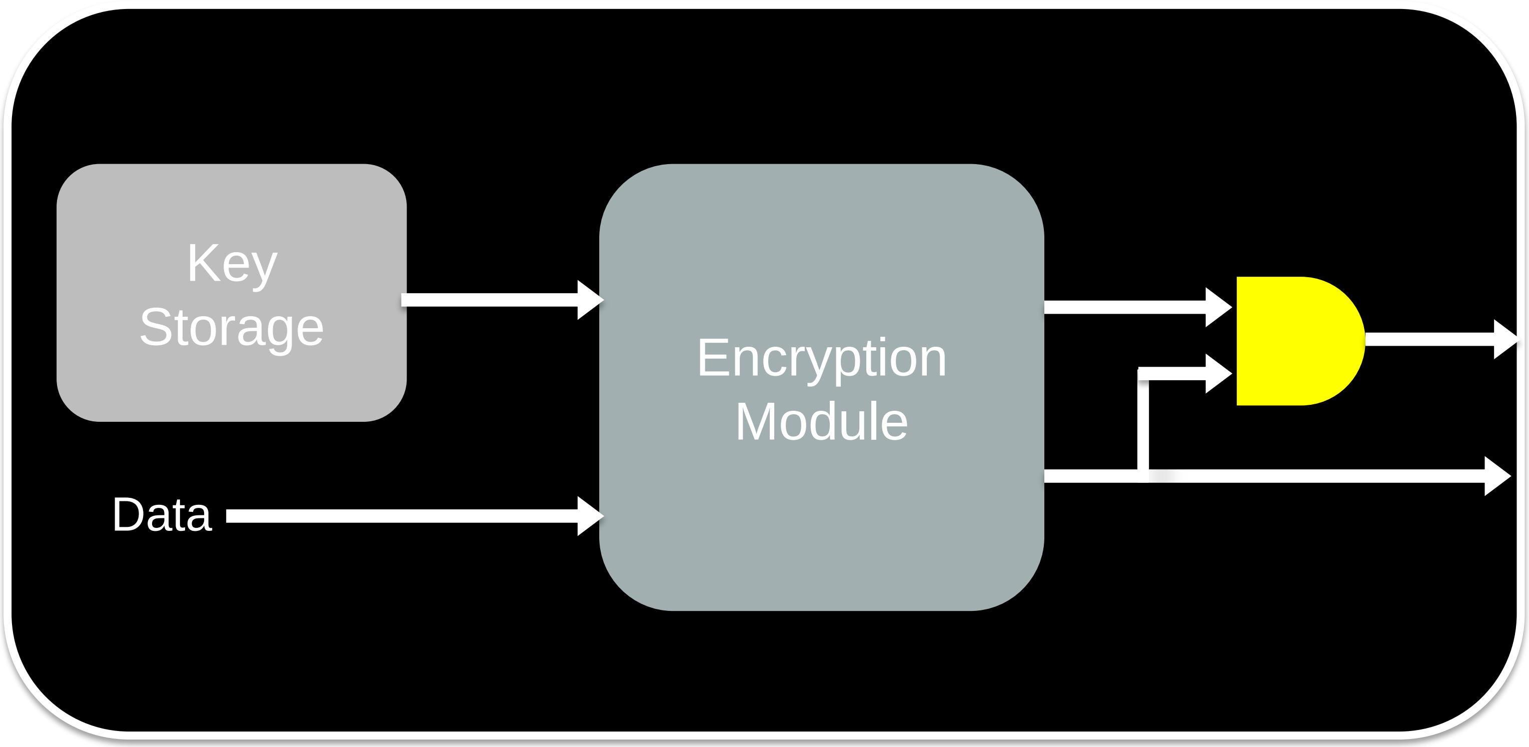
Fails in 4 cycles
Key XOR Data flows to pins, security flaw



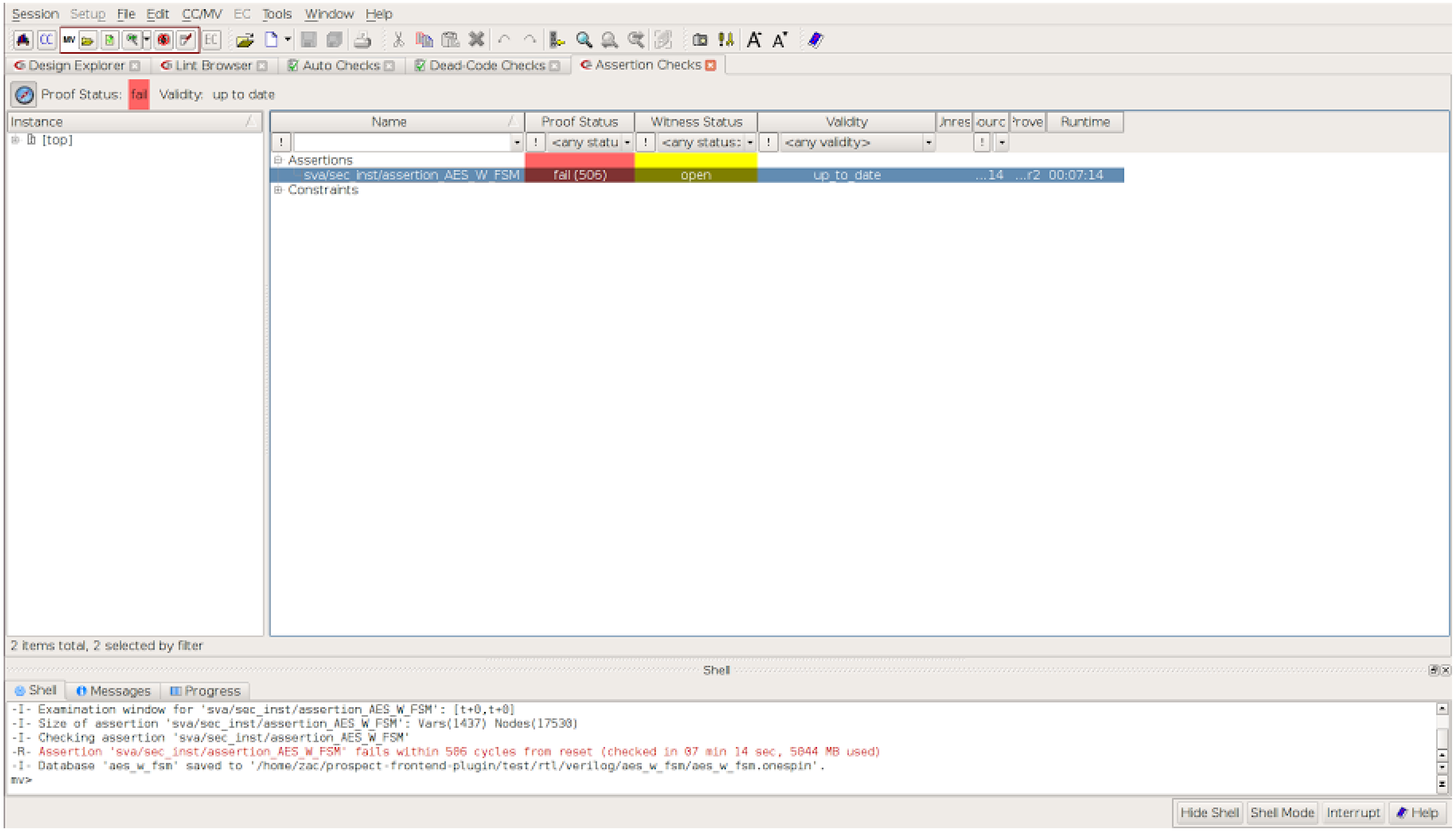
Demo: AES Key Leakage

Property:
assert iflow (*key =/=> data_o*);

Fails in 506 cycles
Encrypted data flows to pins
Flow is allowed, ready_o



Demo: AES Key Leakage



Name	Proof Status	Witness Status	Validity	Jnres	ourc	'rove	Runtime
Assertions							
sva/sec_inst/assertion_AES_W_FSM	fail (506)	open	up_to_date	...14	...r2	00:07:14	
Constraints							

```
-I- Examination window for 'sva/sec_inst/assertion_AES_W_FSM': [t+0,t+0]
-I- Size of assertion 'sva/sec_inst/assertion_AES_W_FSM': Vars(1437) Nodes(17530)
-I- Checking assertion 'sva/sec_inst/assertion_AES_W_FSM'
-R- Assertion 'sva/sec_inst/assertion_AES_W_FSM' fails within 586 cycles from reset (checked in 07 min 14 sec, 5044 MB used)
-I- Database 'aes_w_fsm' saved to '/home/zac/prospect-frontend-plugin/test/rtl/verilog/aes_w_fsm/aes_w_fsm.onespin'.
mv>
```

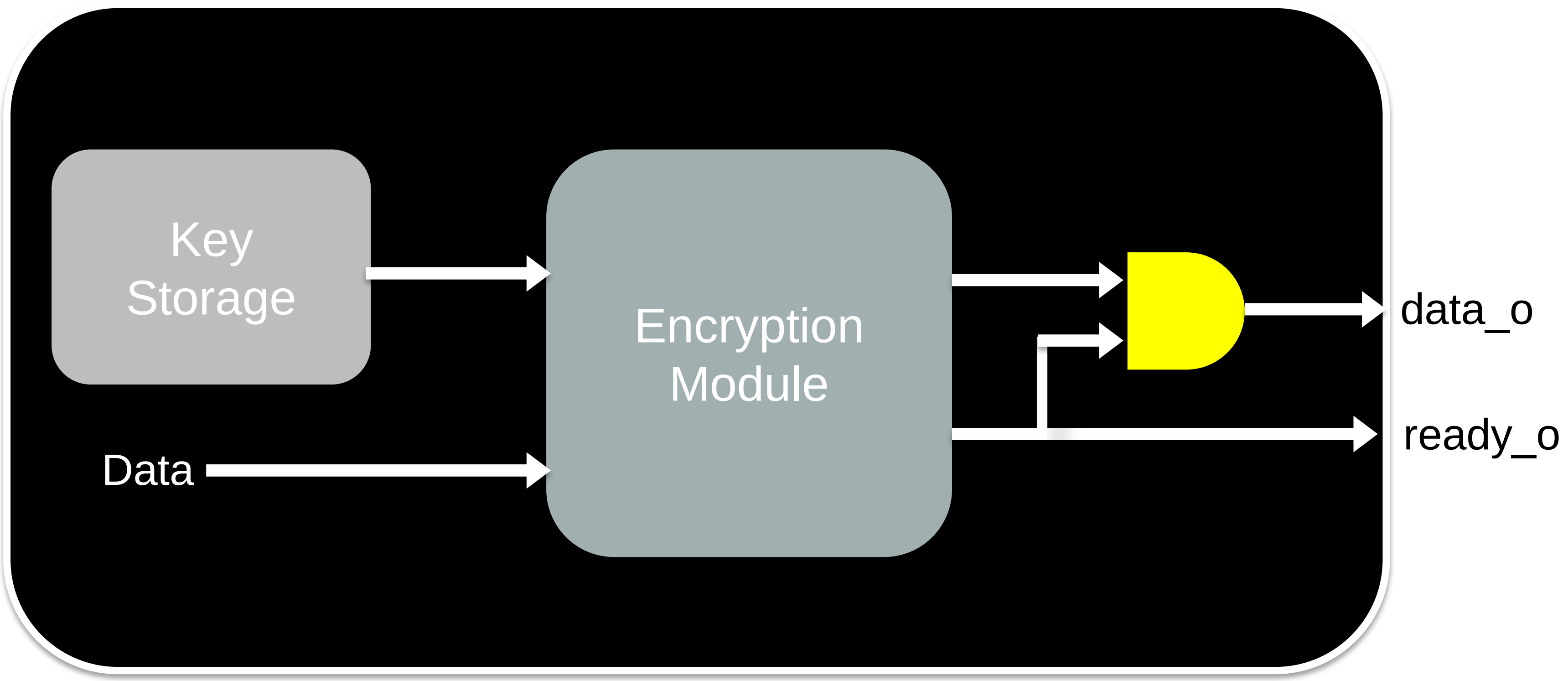
Demo: AES Key Leakage

Property:

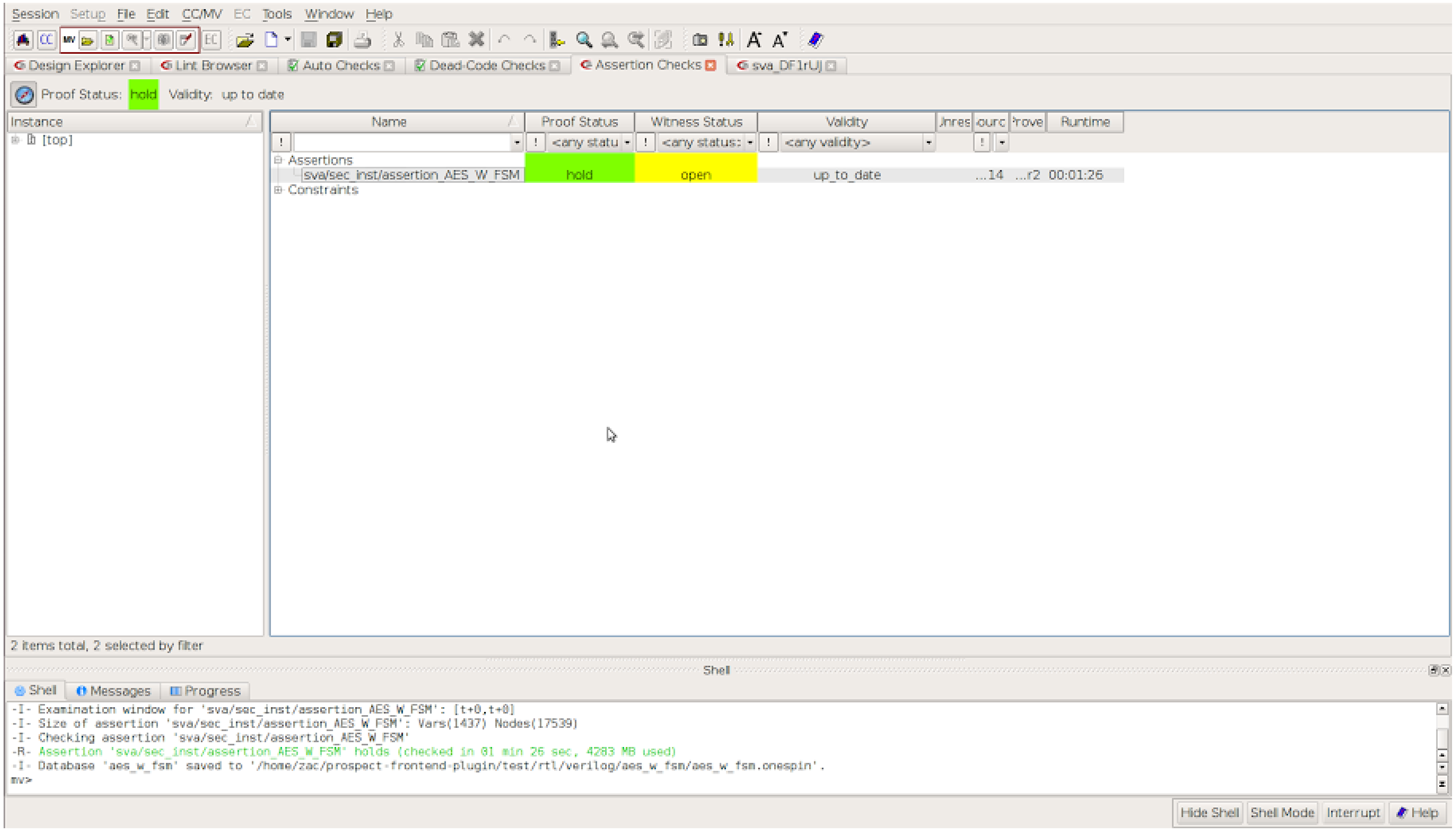
assert iflow (*key* =/=> *data_o*) || *ready_o*;

Result

Assertion Holds



Demo: AES Key Leakage



Proof Status: hold Validity: up to date

Name	Proof Status	Witness Status	Validity	Jnres	ourc	rove	Runtime
Assertions							
sva/sec_inst/assertion_AES_W_FSM	hold	open	up_to_date		...14	...r2	00:01:26
Constraints							

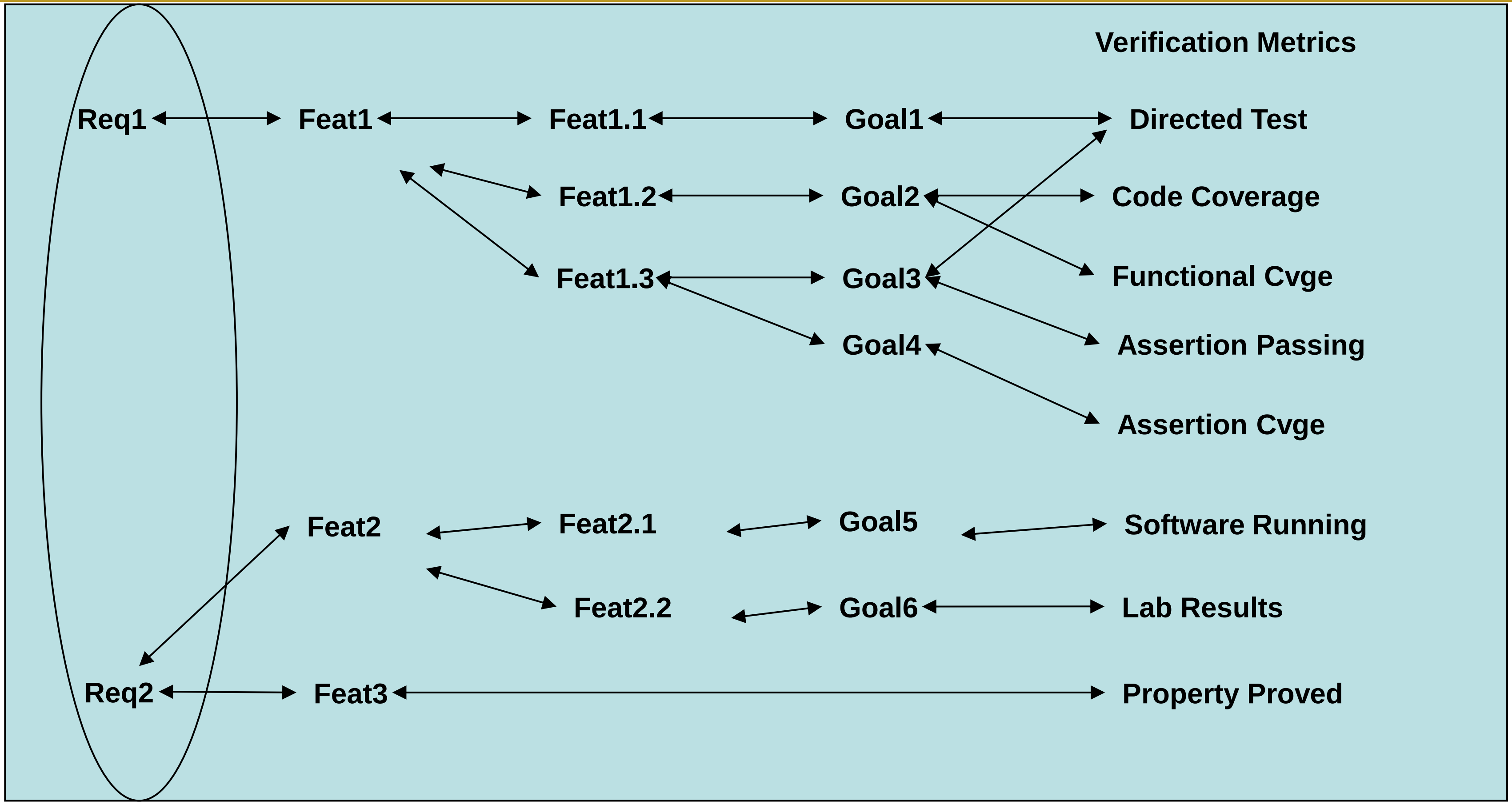
2 items total, 2 selected by filter

```
-I- Examination window for 'sva/sec_inst/assertion_AES_W_FSM': [t+0,t+0]
-I- Size of assertion 'sva/sec_inst/assertion_AES_W_FSM': Vars(1437) Nodes(17539)
-I- Checking assertion 'sva/sec_inst/assertion_AES_W_FSM'
-R- Assertion 'sva/sec_inst/assertion_AES_W_FSM' holds (checked in 01 min 26 sec, 4283 MB used)
-I- Database 'aes_w_fsm' saved to '/home/zac/prospect-frontend-plugin/test/rtl/verilog/aes_w_fsm/aes_w_fsm.onespin'.
mv>
```

TVS Agenda

- **11.00 Introductions**
- **11.05 TVS**
  - Safety and security in Hardware and Software
  - Requirements Driven Test and Verification (RDTV)
  - Using an ECC example and breaking it down into a test plan
- **11.20 OneSpin**
- **11.55 Tortuga Logic**
- **12.10 TVS**
  - Analysing the results and signoff
  - Advantages of RDTV
- **12.25 Q&A (TVS, OneSpin, Tortuga Logic)**

Mapping Requirements to Verification Metrics



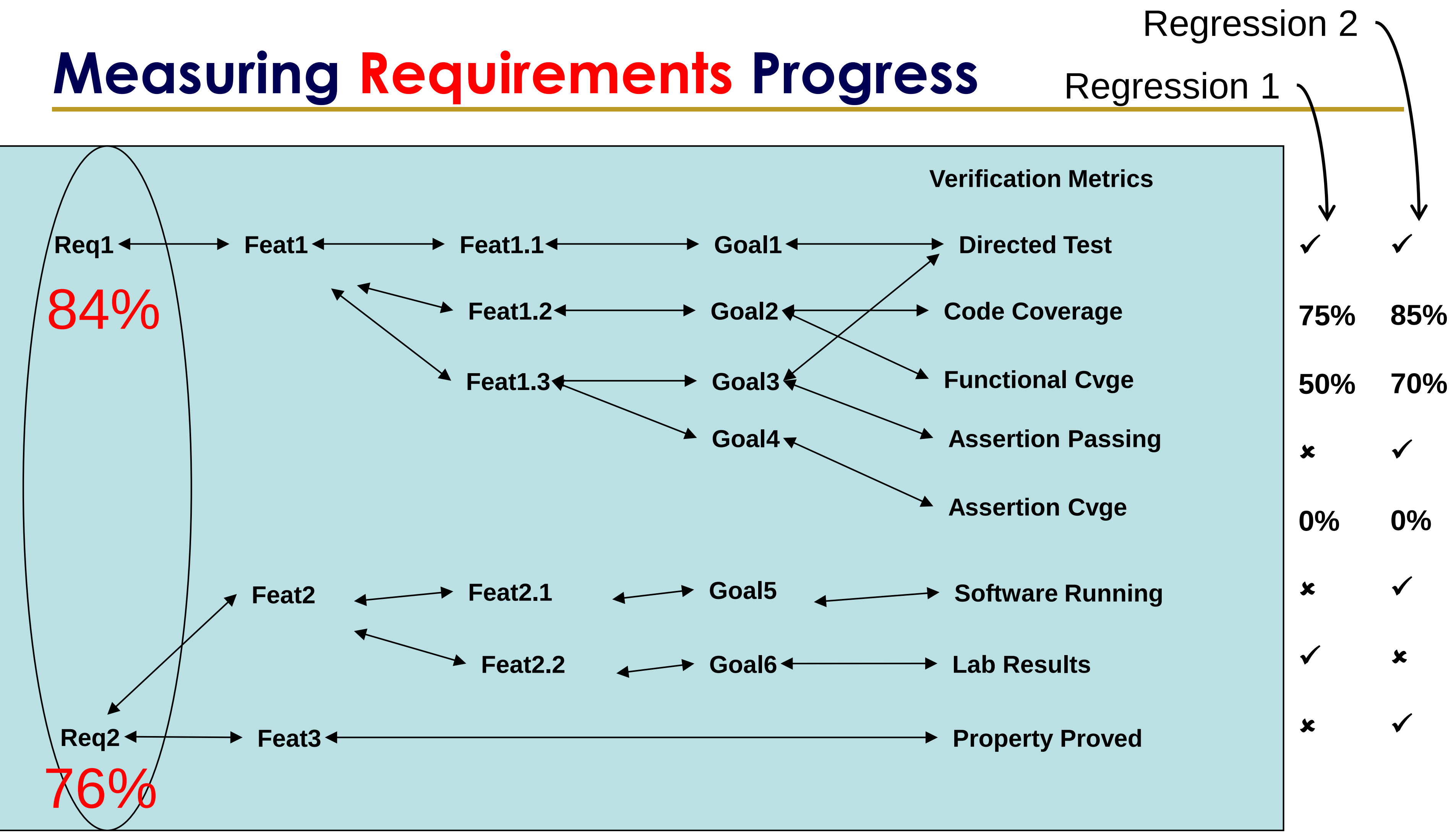
Relationships can be:

- **Bi-directional**
- **Many-many**

Metrics can be:

- **From HW verification**
- **From Silicon validation**
- **From SW testing**

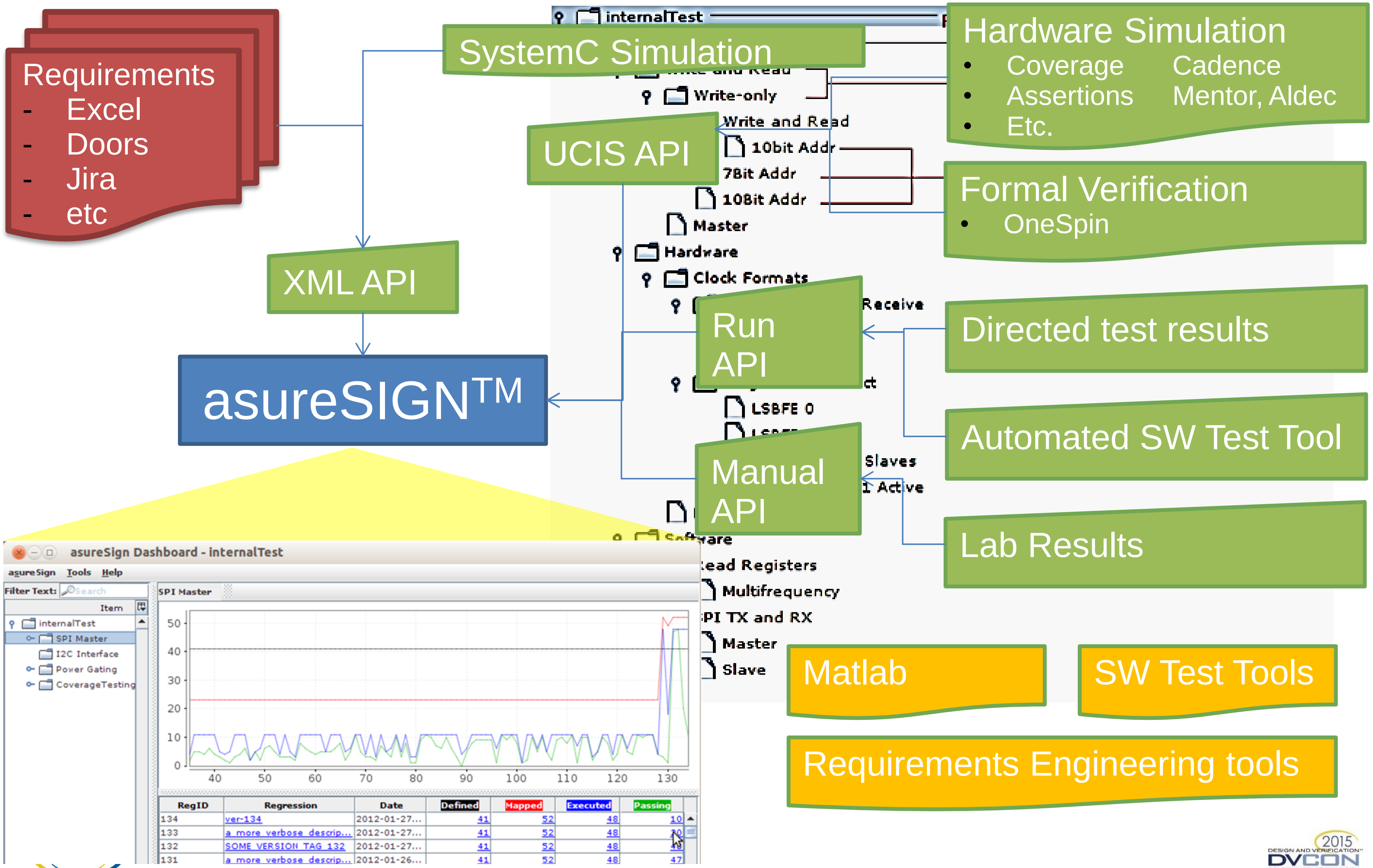
Measuring Requirements Progress



Use a bi-directional mapping to track backwards

Use an SQL database to hold the mappings and results

asureSIGN™ at the heart of HW/SW V&V



RegID	Regression	Date	Defined	Mapped	Executed	Passing
134	ver-134	2012-01-27...	41	52	48	10
133	a more verbose descrip...	2012-01-27...	41	52	48	20
132	SOME VERSION TAG 132	2012-01-27...	41	52	48	48
131	a more verbose descrip...	2012-01-26...	41	52	48	47

SYSTEMS INITIATIVE

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Supporting Hierarchical Verification

- A requirement might be signed off at multiple levels of hierarchy during the hardware development
  - Block
  - Subsystem
  - SoC
  - System
    - Including Software
  - Post Silicon

asureSIGN Demo

- **Mapping the results to the test plan**

Retention of Verification Results (DO 254)

- **Verification records should contain a clear correlation to the pass/fail criteria**
  - These verification records should contain the author/reviewer, date, and any items used in the including their versions.
  - Any failures or issues found should be correlated to the standard that has been violated.
- **Test results should be clearly linked to their associated tests and requirements**
- **Test Results should be reviewed to be sure that the actual and expect results are giving the correct results and that the tests are passing.**

Requirements Driven Verification

- **Compliance to various safety standards**
  - hardware and software (and systems)
- **Some advantages**
  - Identify test holes and test orphans
  - Retention of verification results
    - Build historical perspective for more accurate predictions
  - Better reporting of requirements status
  - Risk-based testing
  - Prioritisation and Risk Analysis
  - Filtering Requirements based on Customers and releases
  - Impact and conflict analysis

Any questions ?

I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?
E-mail: SCOTTADAMS@AOL.COM
© 2006 Scott Adams, Inc./Dist. by UFS, Inc.
www.dilbert.com