2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION CHINA

accellera SYSTEMS INITIATIVE

ANALOG DEVICES

AHEAD OF WHAT'S POSSIBLE™

Exploring Formal Verification: A Journey Through Maturity Levels and Case Studies, with a Glimpse into the Future of Assertion Creation with the Power of GPT

Shawn Zhang

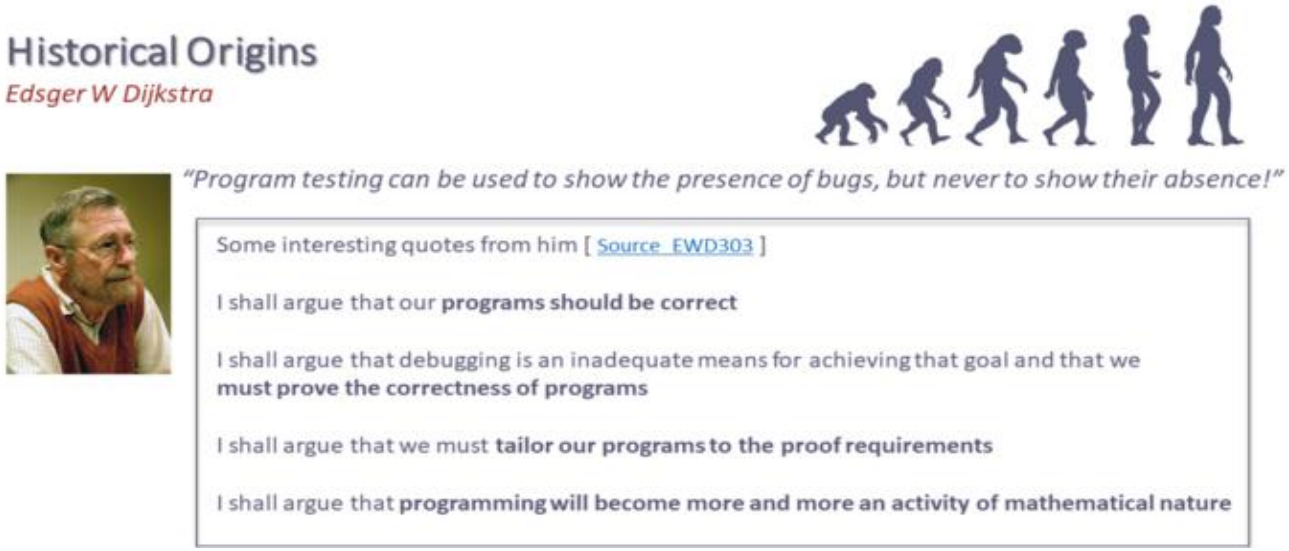
Agenda

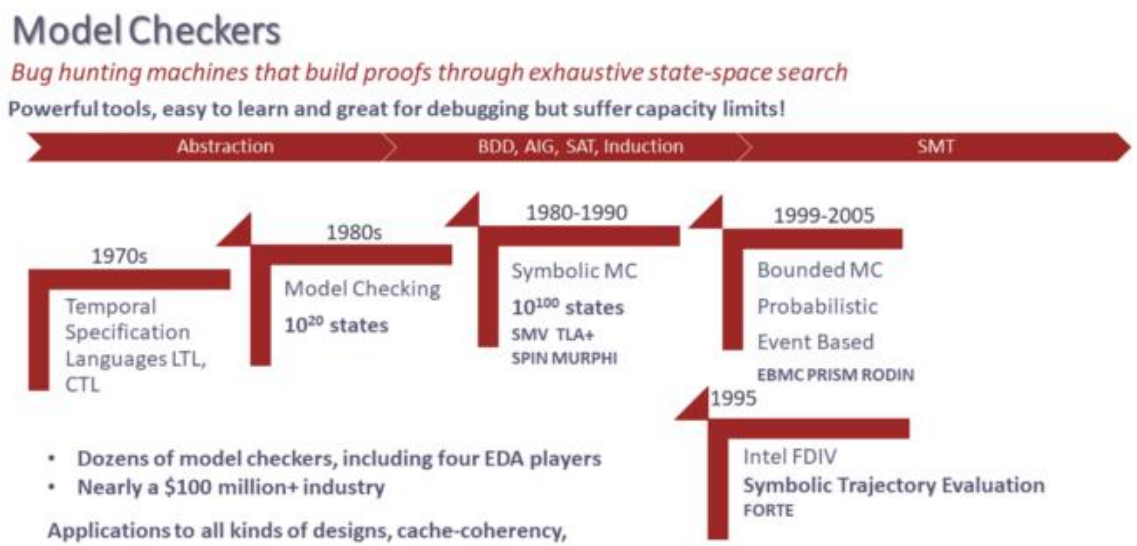
- Introduction
  - History of formal
  - Algorithm of formal
  - Classification of formal
  - Future of formal
- Case Study
  - Functional Safety (Level 1)
  - Connectivity (Level 2)
  - MHDMA (Level 3)
  - Pyro-Fuse (Level 4)
- Autogeneration
  - Open Verification Library
  - ezAssert
  - ChatGPT
- Conclusion
  - Summary
  - Future work
  - Reference

Agenda

- **Introduction**
  - **History of formal**
  - **Algorithm of formal**
  - **Classification of formal**
  - **Future of formal**
- Case Study
  - Functional Safety (Level 1)
  - Connectivity (Level 2)
  - MHDMA (Level 3)
  - Pyro-Fuse (Level 4)
- Autogeneration
  - Open Verification Library
  - ezAssert
  - ChatGPT
- Conclusion
  - Summary
  - Future work
  - Reference

History of formal

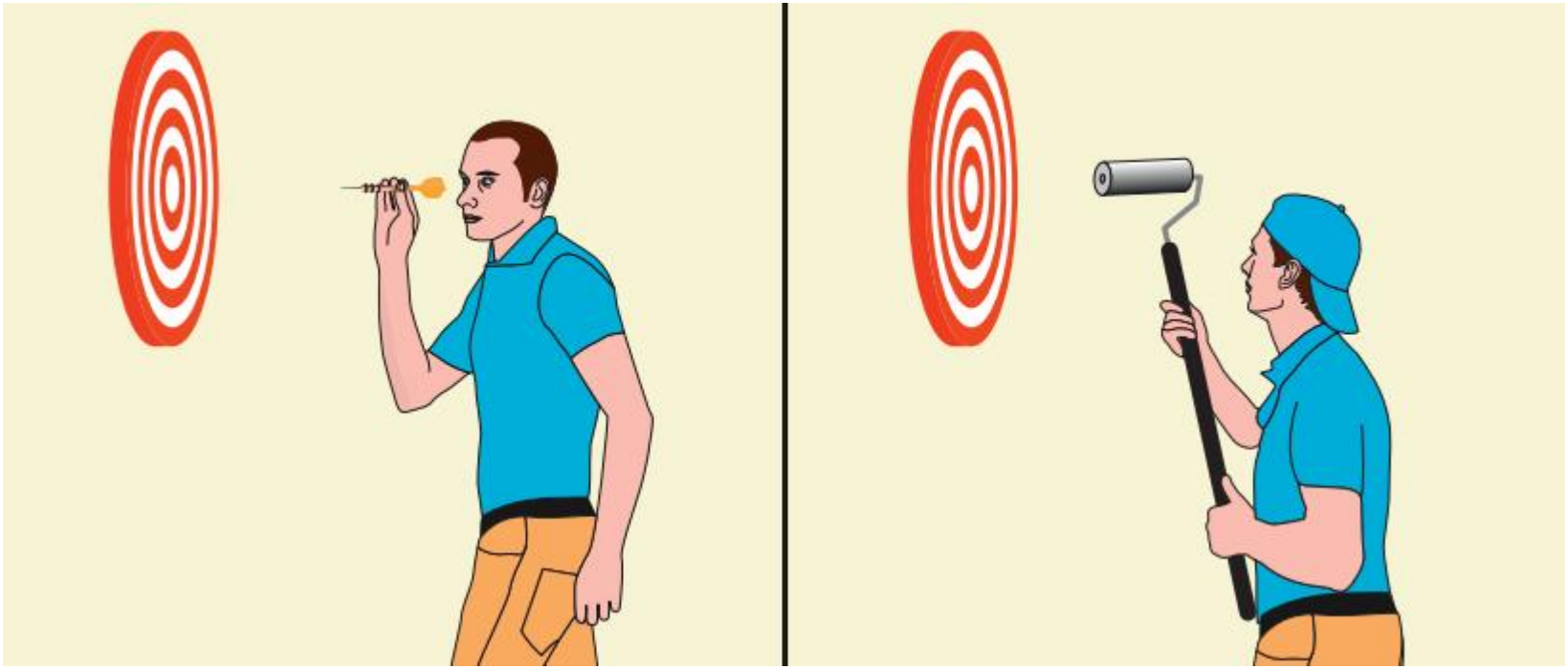


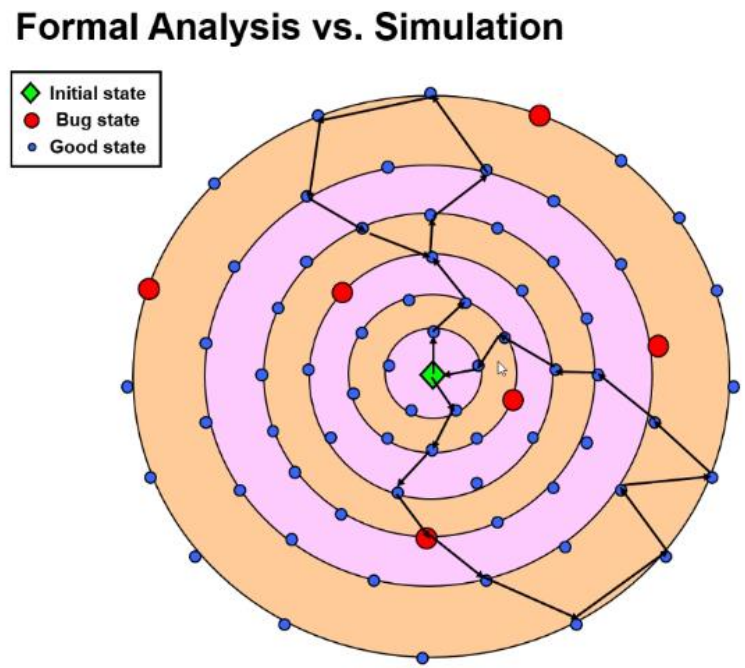


The notion of mathematical proof has been the cornerstone of formal methods. (Source: Axiomise.com)

- In April 1970, Edsger Wybe Dijkstra pointed out "**Testing shows the presence, not the absence, of bugs**".
- In 1981, Emerson and Clarke provided the first automated **model-checking** algorithm.
- After several decades, Model-checking applications for **hardware verification** have come a long way.
- The ratio of the first silicon success has declined to its lowest point in 20 years[1].
- The biggest challenge is insufficient tests to close the coverage.

Formal vs Simulation





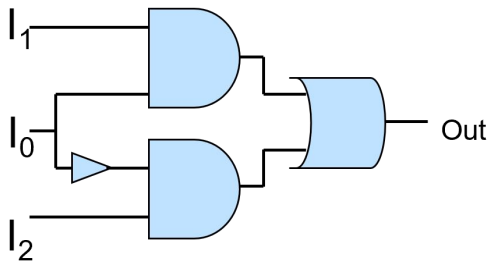
Coverage of your design space by simulation and Formal Verification [2]

Formal	Simulation
Exhaustive States	Exhaustive Stimuli
Model Checking	Constrained Randomization
A Simple Wrapper is enough	A Complex Testbench is necessary
Suitable for control-oriented design	Suitable for long data path design
Property-driven. One loop at one clock cycle	Event-driven. One step at one timeslot

BDD (Binary Decision Diagram)

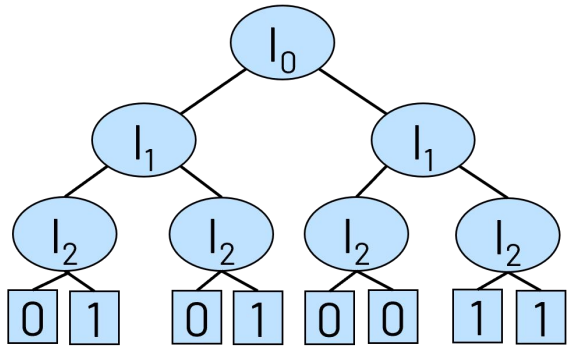
Simulation

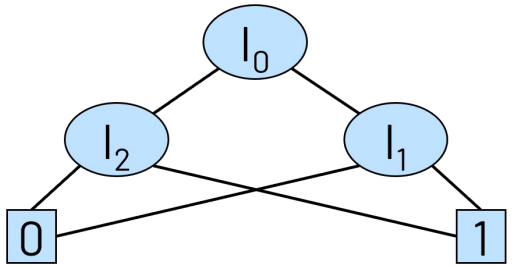
Stimulus			Result
I_0	I_1	I_2	Out
0	0	0	0
0	0	1	1
0	1	0	0
0	**1**	**1**	**1**
1	0	0	0
1	**0**	**1**	**0**
1	1	0	1
1	1	1	1



$Out == (I_0 \&\& I_1) \| (!I_0 \&\& I_2)$

Formal

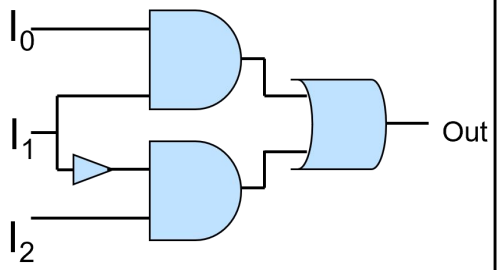




Spec

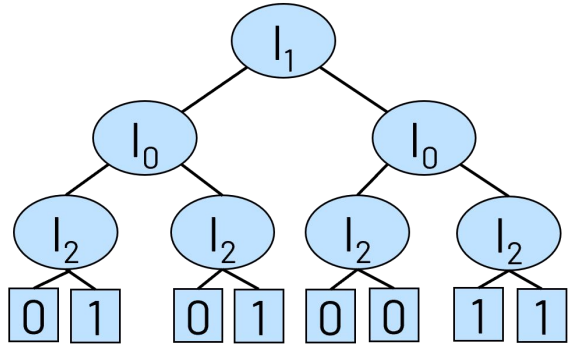
Simulation

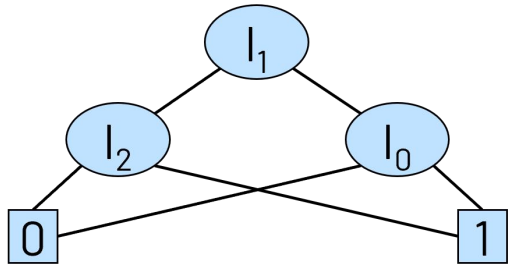
Stimulus			Result
I_0	I_1	I_2	Out
0	0	0	0
0	0	1	1
0	1	0	0
0	**1**	**1**	**0**
1	0	0	0
1	**0**	**1**	**1**
1	1	0	1
1	1	1	1



$Out == (I_0 \&\& I_1) \| (!I_1 \&\& I_2)$

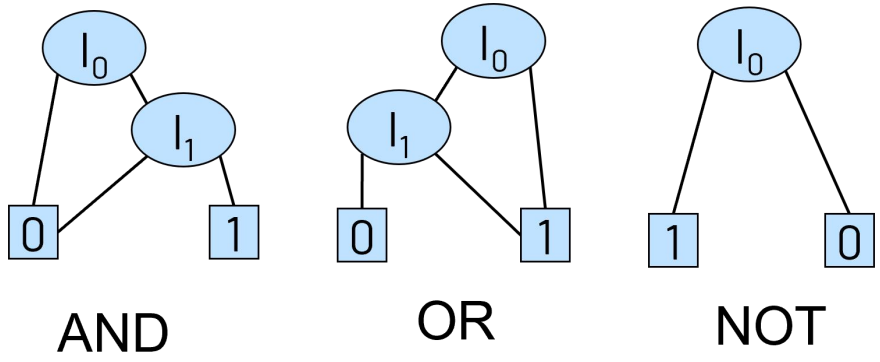
Formal



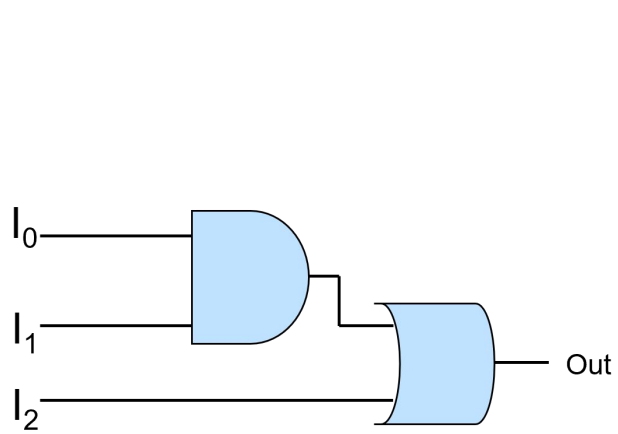


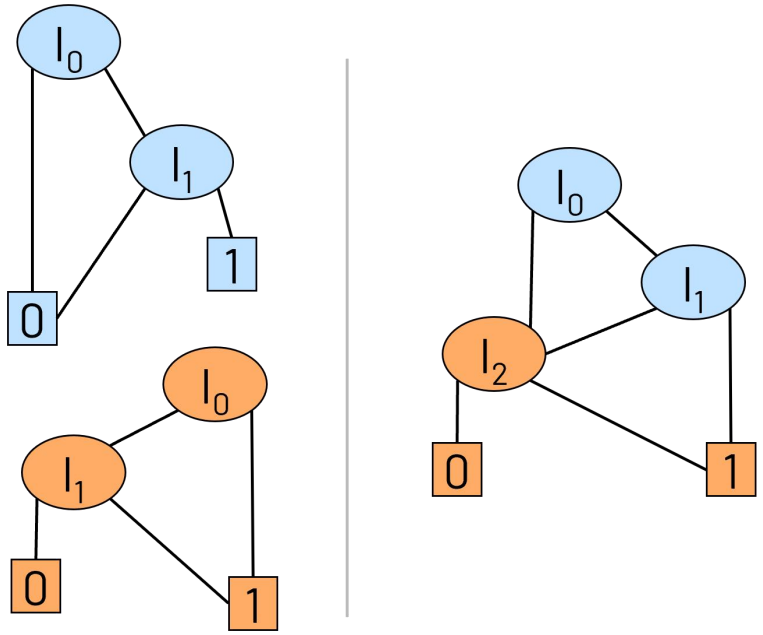
DUT

Model Checking



Primitive BDDs

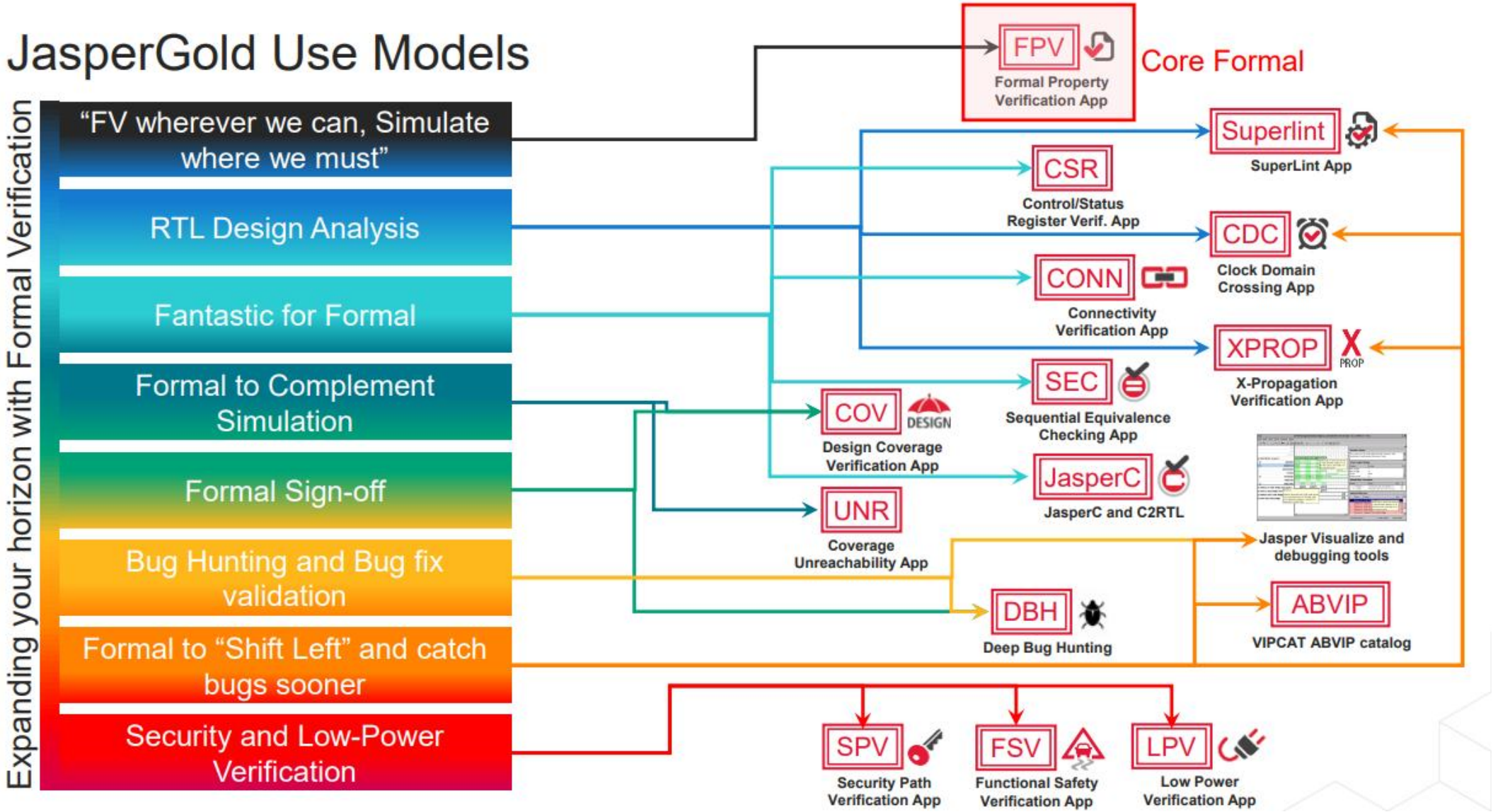




Circuit Computing

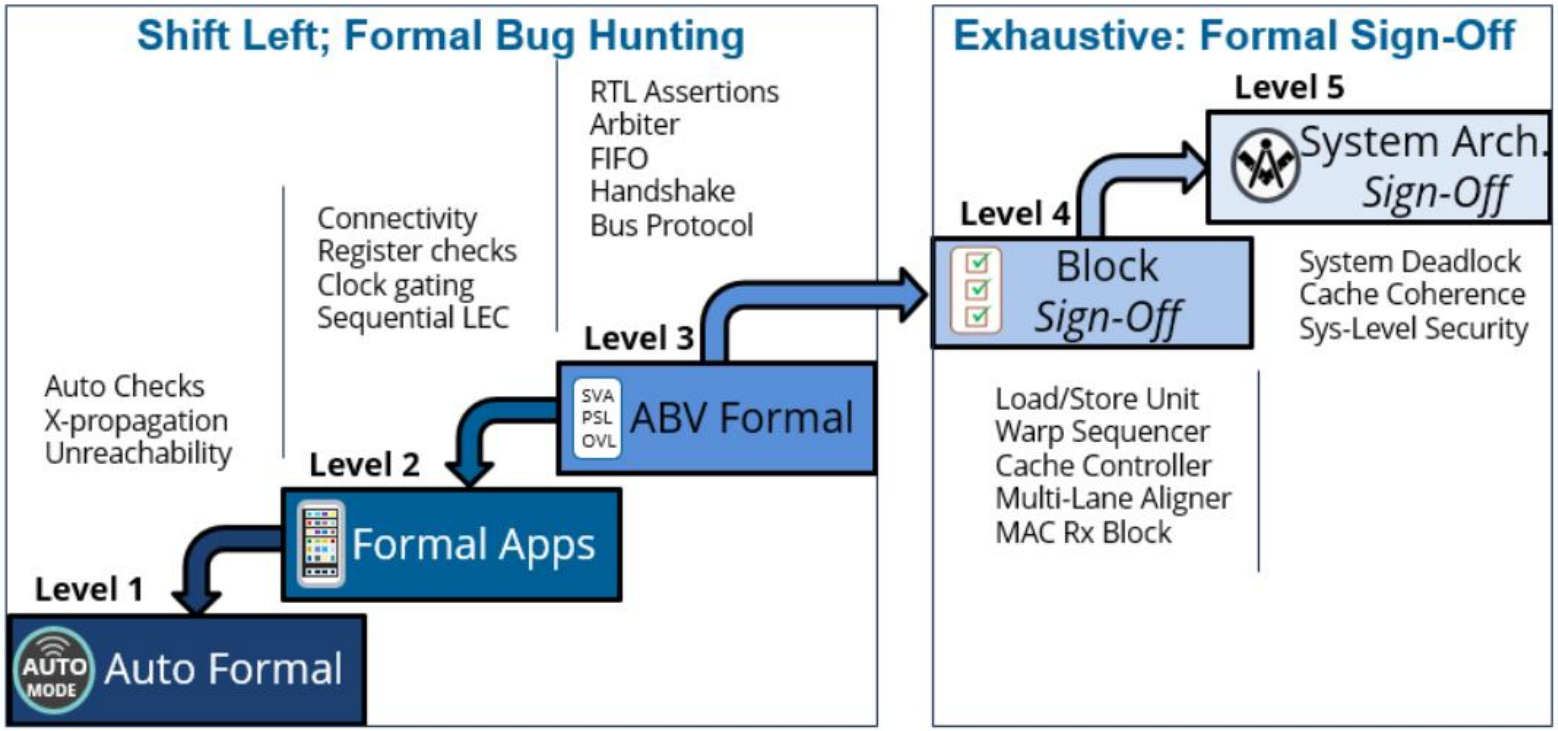
- *Formally, the Model Checking problem can be stated as follows:* ***given a desired property, expressed as a temporal logic formula p, and a structure M with initial state s****, decide if M,s |= p. If M is finite, as it is in hardware, model checking reduces to a graph search.------Wikipedia*
- BDD models can grow **exponentially** with each additional input or state element.
- **Time and memory** are the bottleneck.

Application of formal



JUG2021, Introduction to formal and jasper gold[3]

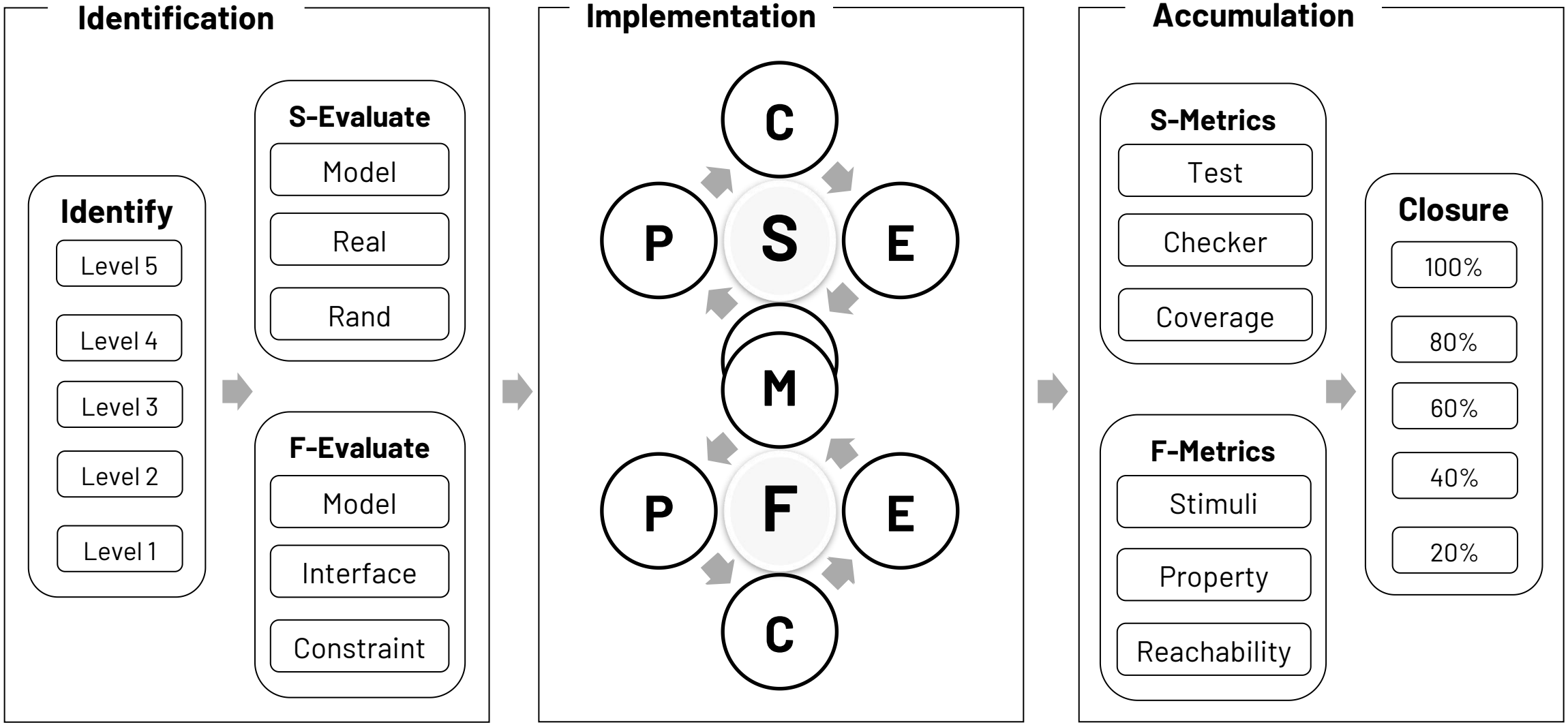
Classification of formal



DV Con 2022. Raising the level of formal signoff with end-to-end checking methodology[4]

- L1: Automatic formal such as auto-checking, formal linting, and dead-code identification
- L2: Formal applications such as connectivity checking, register checking, and X checking
- L3: Assertion-based VIP formal verification such as interface and bus assertions
- L4: Block-level signoff that thoroughly verifies all block-level design behaviors
- L5: System architectural-level verification focusing on specific high-level design

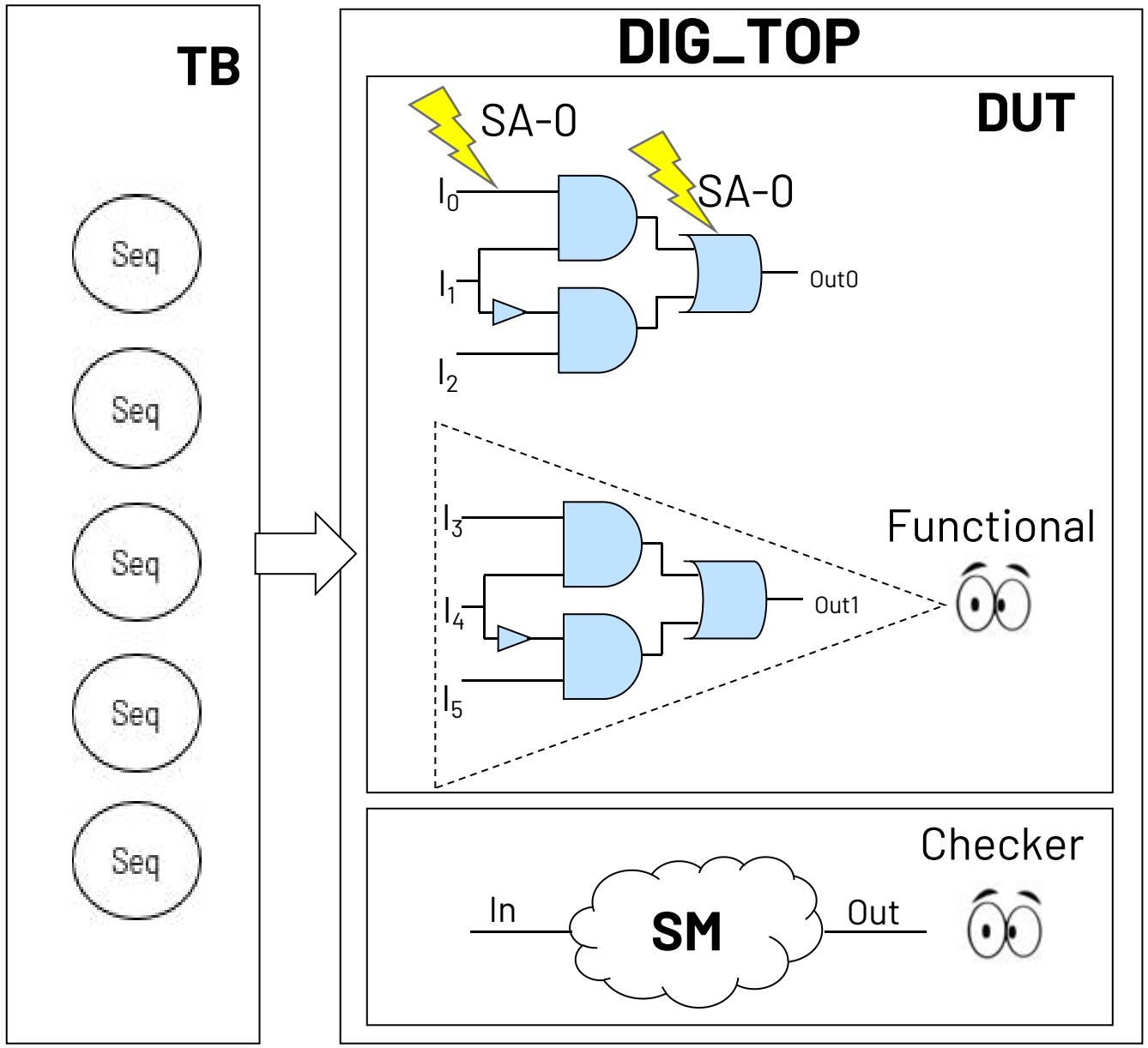
Future of formal



Agenda

- Introduction
  - History of formal
  - Algorithm of formal
  - Classification of formal
  - Future of formal
- **Case Study**
  - **Functional Safety (Level 1)**
  - **Connectivity (Level 2)**
  - **MHDMA (Level 3)**
  - **Pyro-Fuse (Level 4)**
- Autogeneration
  - Open Verification Library
  - ezAssert
  - ChatGPT
- Conclusion
  - Summary
  - Future work
  - Reference

Functional Safety (Level 1)



- Logic Fault Collapsing
  - Equivalence
  - Dominance
- Design-Based Fault Pruning
  - OOCOI
  - Activation
  - Propagation
- Test-Based Fault Pruning
  - Fault Collapsing with constants
  - Activatability analysis with constants
  - Propagability analysis with constants

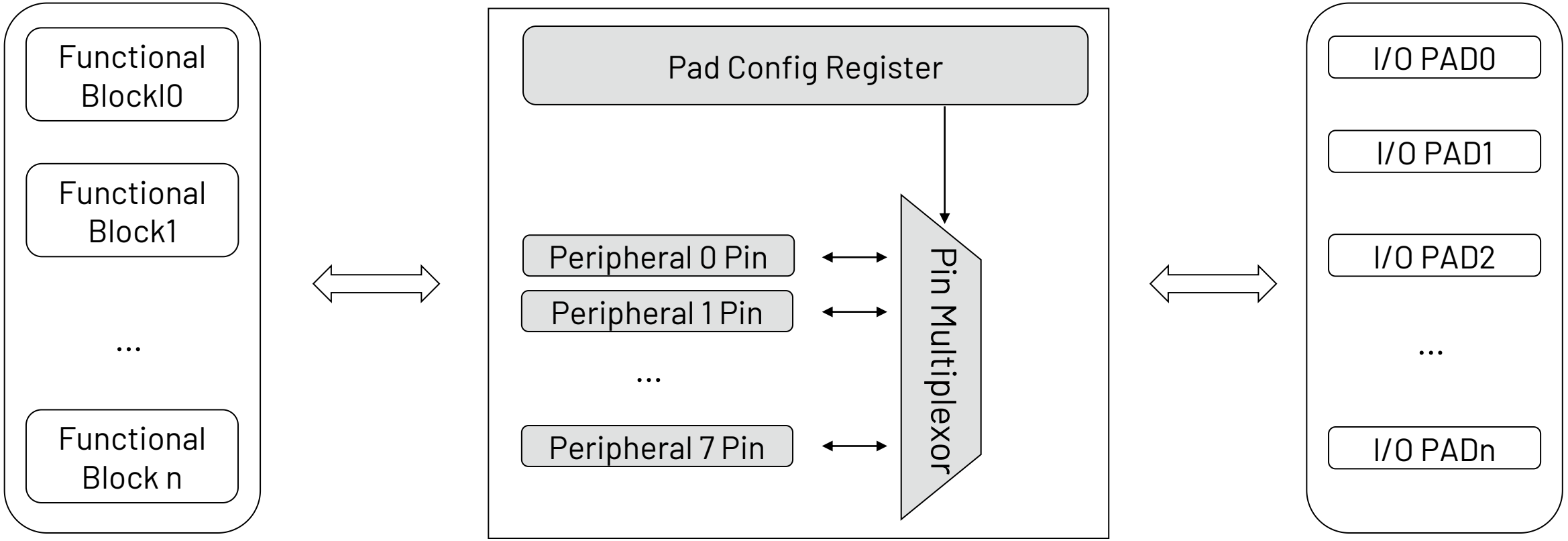
Results

Block A	Total	S	DD	DU	NC	DC	S	SPFM
XFS	16094	20	767	2	15305	99.74%	0.12%	99%
FST+FSV_TC	16094	3860	752	2	11480	99.73%	23.98%	100%

Block B	Total	S	DD	DU	NC	DC	S	SPFM
XFS	23568	218	589	132	22629	81.69%	0.92%	86%
FST+FSV_TC	23568	5399	790	0	17379	100%	22.91%	100%

Block C	Total	S	DD	DU	NC	DC	S	SPFM
XFS	618	0	126	103	389	55.02%	0%	55%
FST+FSV_TC	618	418	64	93	43	40.76%	67.64%	84%

PinMux (Level 2)



- PinMux is a type of multiplexer for connecting multiple peripheral functions to one IO pad controlled by a set of registers as shown in the Figure.
- The verification of PinMux is to exhaustively verify the consistency between the DUT and the connectivity definition in the specification.
- It is tedious, error-prone, and time-consuming.

Conn

# Simple connection	Name	Source instance	Source signal	Destination instance	Destination signal
CONNECTION	axi_csr_addr	axi_brdg	int_addr	csr_blk	int_addr

```
axi_csr_addr: assert (axi_brdg.int_addr == slv3.bus_if.regfile.int_addr);
```

Connection with condition expression

CONNECTION	axi_csr_req	axi_brdg	int_req	csr_blk	int_req[3]
	COND_EXPR	(axi_brdg.b_addr >= 16 && axi_brdg.b_addr < 32)			

```
axi_csr_req: assert (axi_brdg.b_addr >= 16 && axi_brdg.b_addr < 32 |-> axi_brdg.int_req == slv3.bus_if.regfile.int_req[3]);
```

Connection with individual conditions

CONNECTION	axi_intr_overflow	axi_brdg	fifo_full	intr_ctl	overflow
	CONDITION	axi_brdg	int_req	1	
	CONDITION	axi_brdg	int_write	1	

```
axi_intr_overflow: assert (axi_brdg.int_req == 1 && axi_brdg.int_write == 1 |-> axi_brdg.fifo_full == int_ctrl.overflow);
```

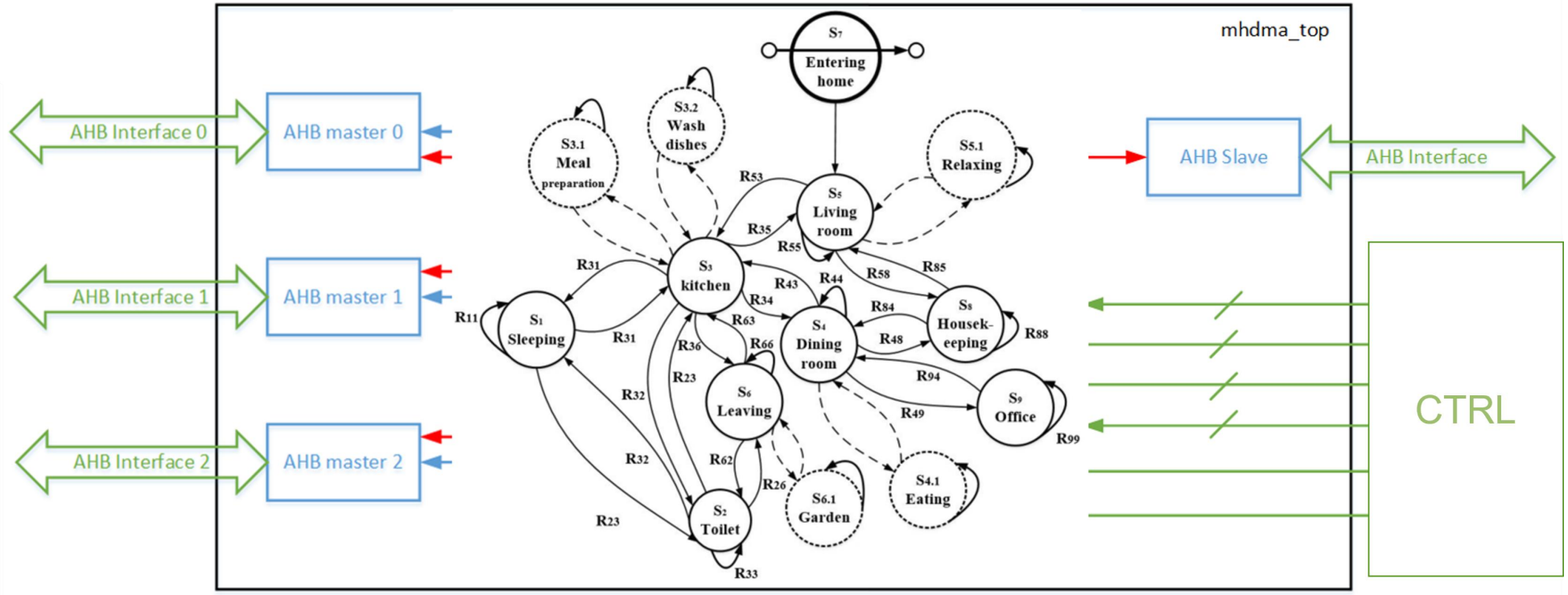
Connections tied to a constant: All bits of bus tied to constant

CONNECTION	intr_tied_low		JDA:LOW	intr_ctl	irq[3:0]
CONNECTION	intr_tied_high		JDA:HIGH	intr_ctl	irq[7:4]

```
intr_tied_low: assert (4'b0000 == intr_ctrl.irq[3:0]);
intr_tied_high: assert (4'b1111 == intr_ctrl.irq[7:4]);
```

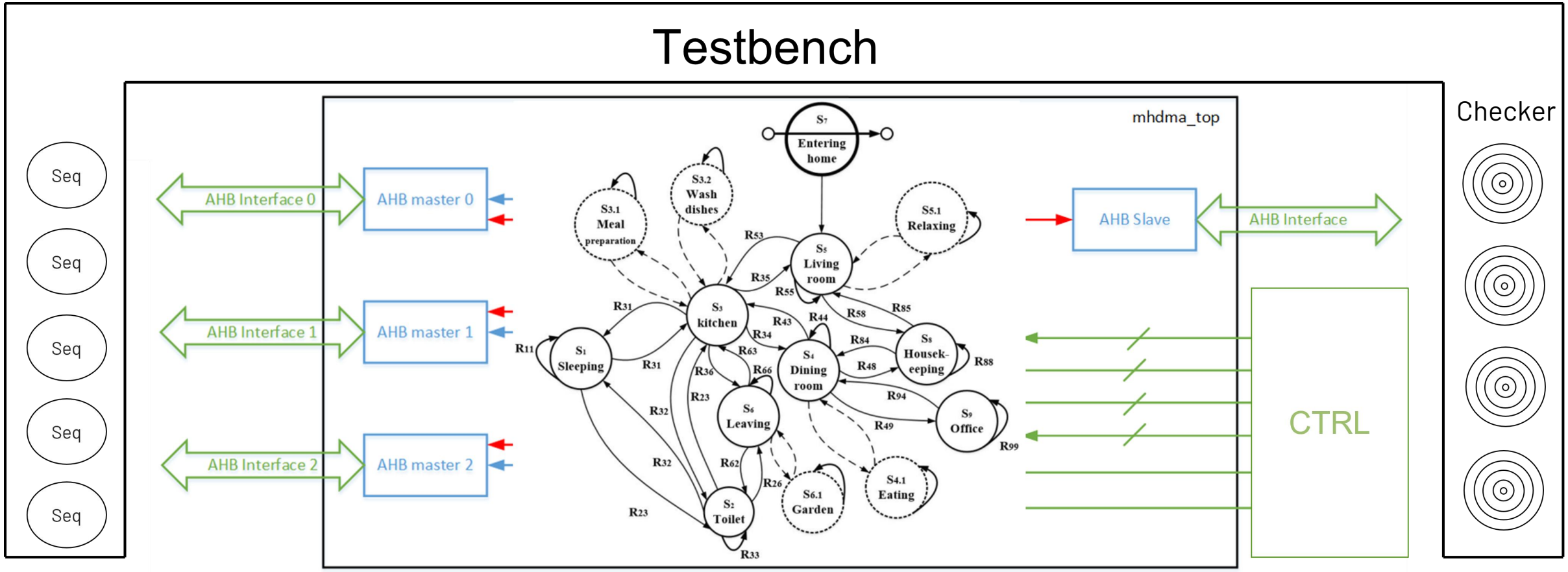
Picture from Cadence

MHDMA (Level 3)



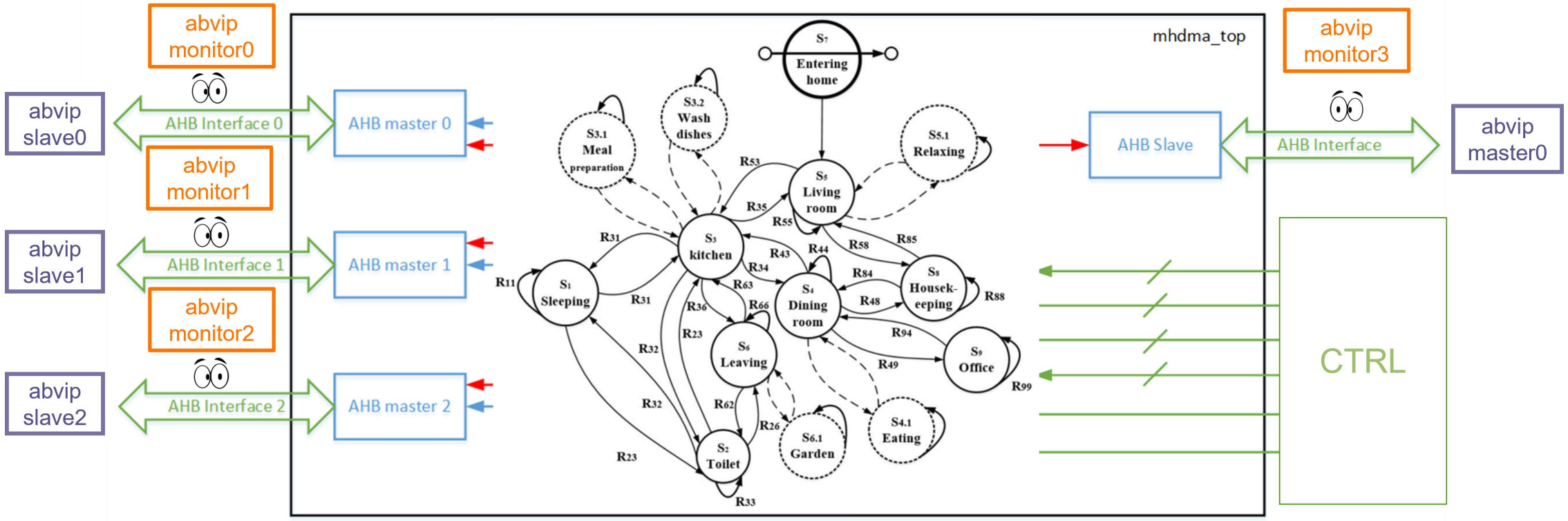
- This DMA block is an AMBA3 compliant IP. It is used to transfer data among mem and peripherals without CPU intervention.
- There are 3 AHB masters used for general data transfer. 1 AHB slave is used for mmr programming.
- The control signals are used to control the arbitration and main FSM.

Current Flow



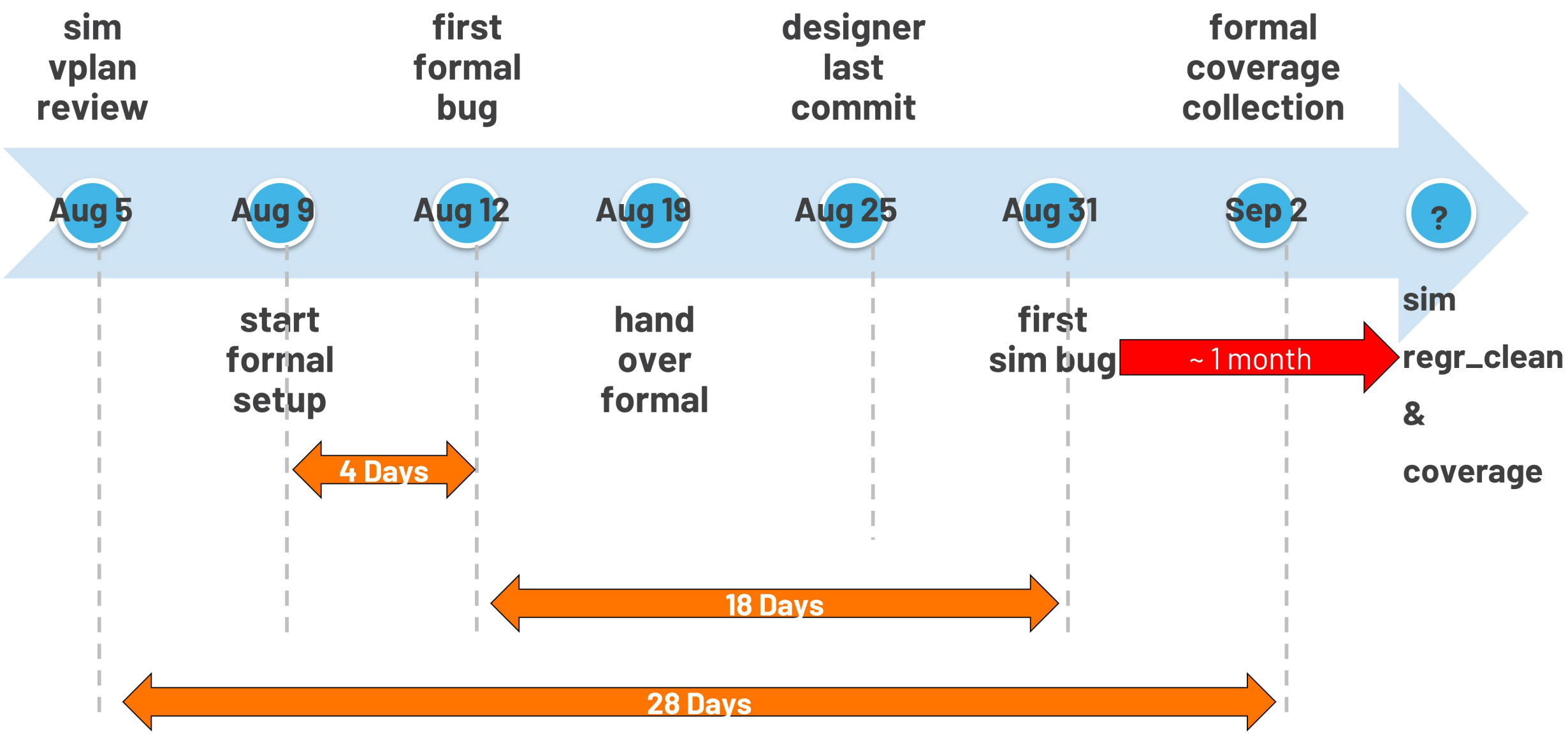
- A UVM-based testbench contains all required components and UVCs. (3~5 days)
- Design a suite of testcases or sequences according to Spec. (1~2weeks)
- Design related checkers. (1~2weeks)
- Debug and pass the first testcase. (1~2days)

New Flow

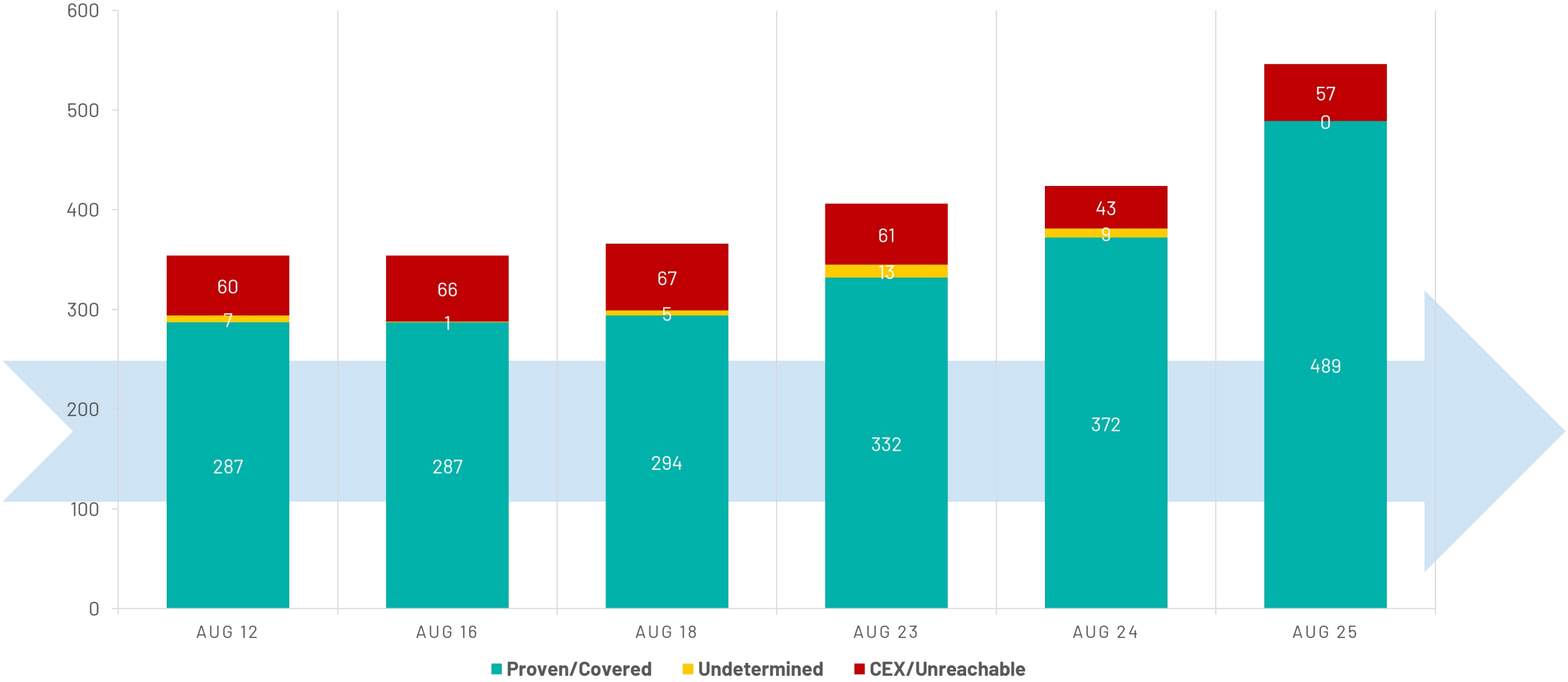


- Ramp up the usage of Assertion-based VIP such as master mode and monitor mode. (1 day)
- Design a simple wrapper to include DUT and ABVIP. Do the configuration such as type/width. (1 day)
- Design the tcl file for Jasper with proper filelists, assumptions and options. (1 day)
- Debug and pass the first assertion. (1 day)

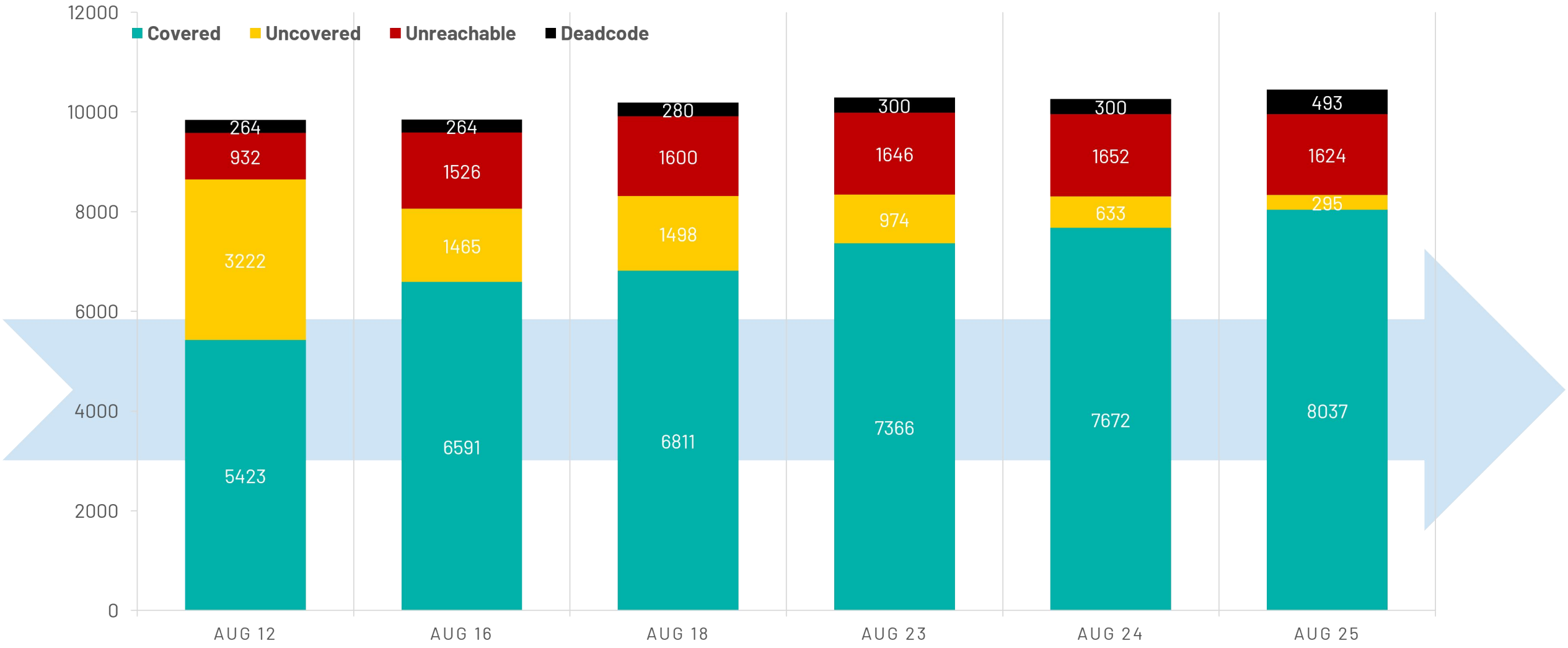
Timeline


sim vplan review
first formal bug
designer last commit
formal coverage collection
Aug 5
Aug 9
Aug 12
Aug 19
Aug 25
Aug 31
Sep 2
?
start formal setup
hand over formal
first sim bug
~ 1 month
sim regr_clean & coverage
4 Days
18 Days
28 Days

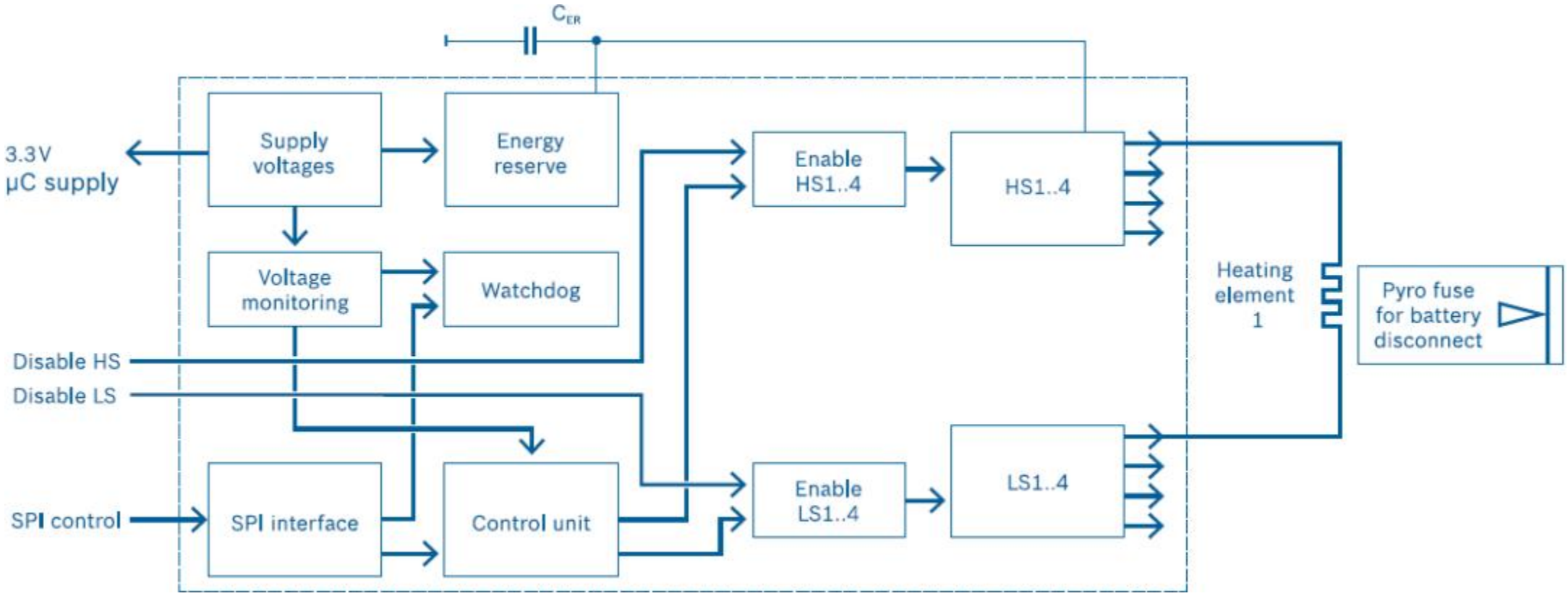
Convergency



Coverage


12000
10000
8000
6000
4000
2000
0
Covered
Uncovered
Unreachable
Deadcode
264
932
3222
5423
264
1526
1465
6591
280
1600
1498
6811
300
1646
974
7366
300
1652
633
7672
493
1624
295
8037
AUG 12
AUG 16
AUG 18
AUG 23
AUG 24
AUG 25

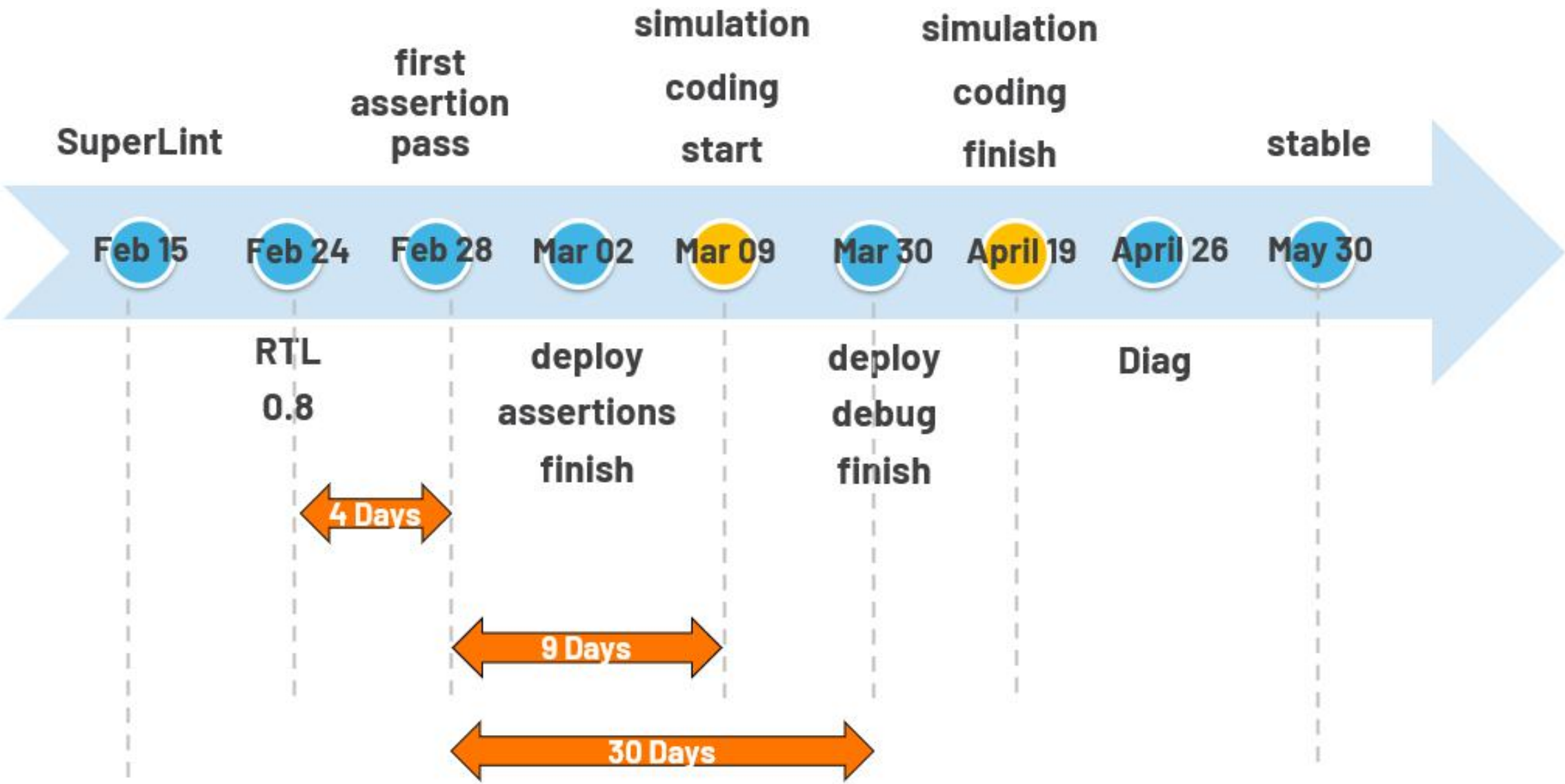
Pyro-Fuse (Level 4)



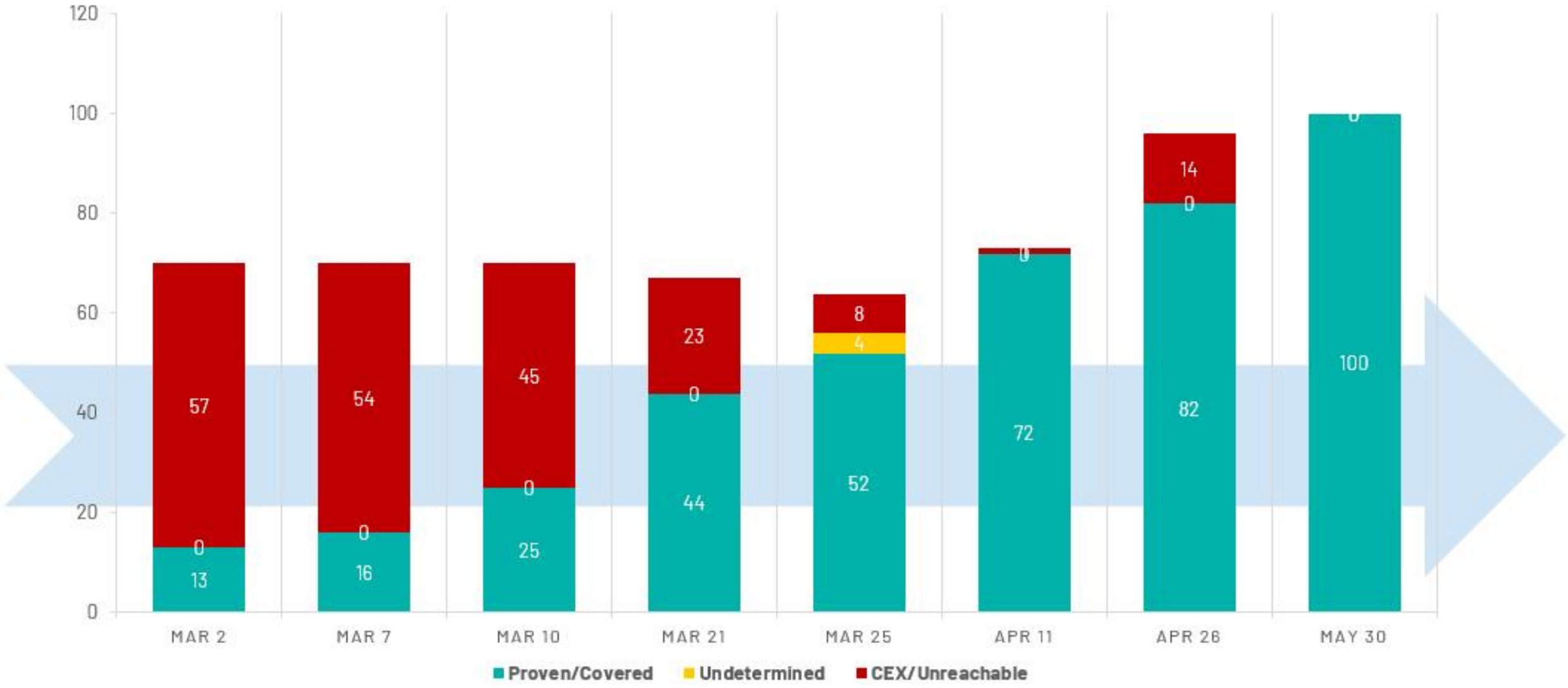
Official Account: New Energy BMS

- Pyro-Fuse is a device that can disconnect a battery from an electrical system.
- It is a state machine that contains a huge state space and many unexpected corner cases.
- No available Assertion-based VIP can be leveraged.

Timeline


SuperLint
first assertion pass
simulation coding start
simulation coding finish
stable
Feb 15
Feb 24
Feb 28
Mar 02
Mar 09
Mar 30
April 19
April 26
May 30
RTL 0.8
deploy assertions finish
deploy debug finish
Diag
4 Days
9 Days
30 Days

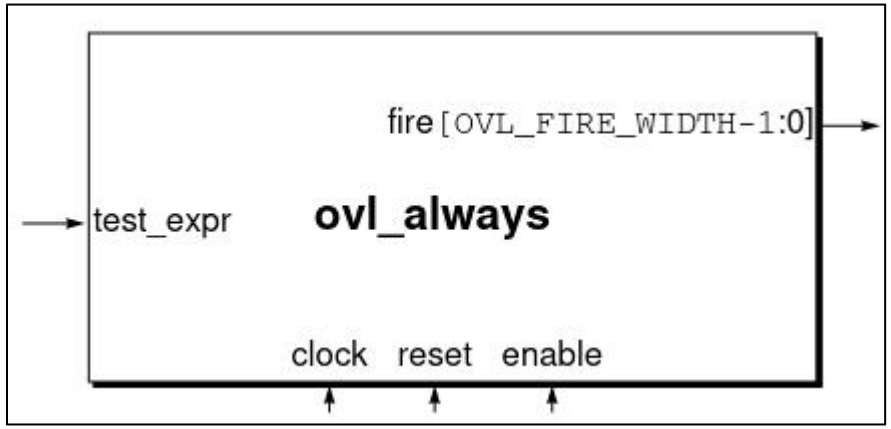
Convergency



Agenda

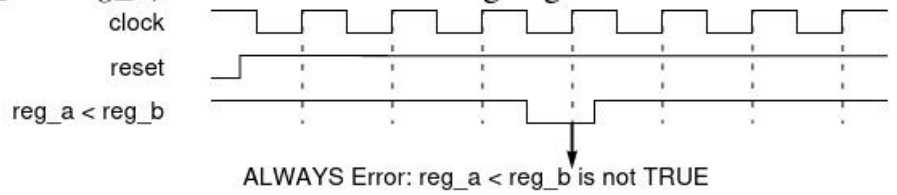
- Introduction
  - History of formal
  - Algorithm of formal
  - Classification of formal
  - Future of formal
- Case Study
  - Functional Safety (Level 1)
  - Connectivity (Level 2)
  - MHDMA (Level 3)
  - Pyro-Fuse (Level 4)
- **Autogeneration**
  - **Open Verification Library**
  - **ezAssert**
  - **ChatGPT**
- Conclusion
  - Summary
  - Future work
  - Reference

Open Verification Library



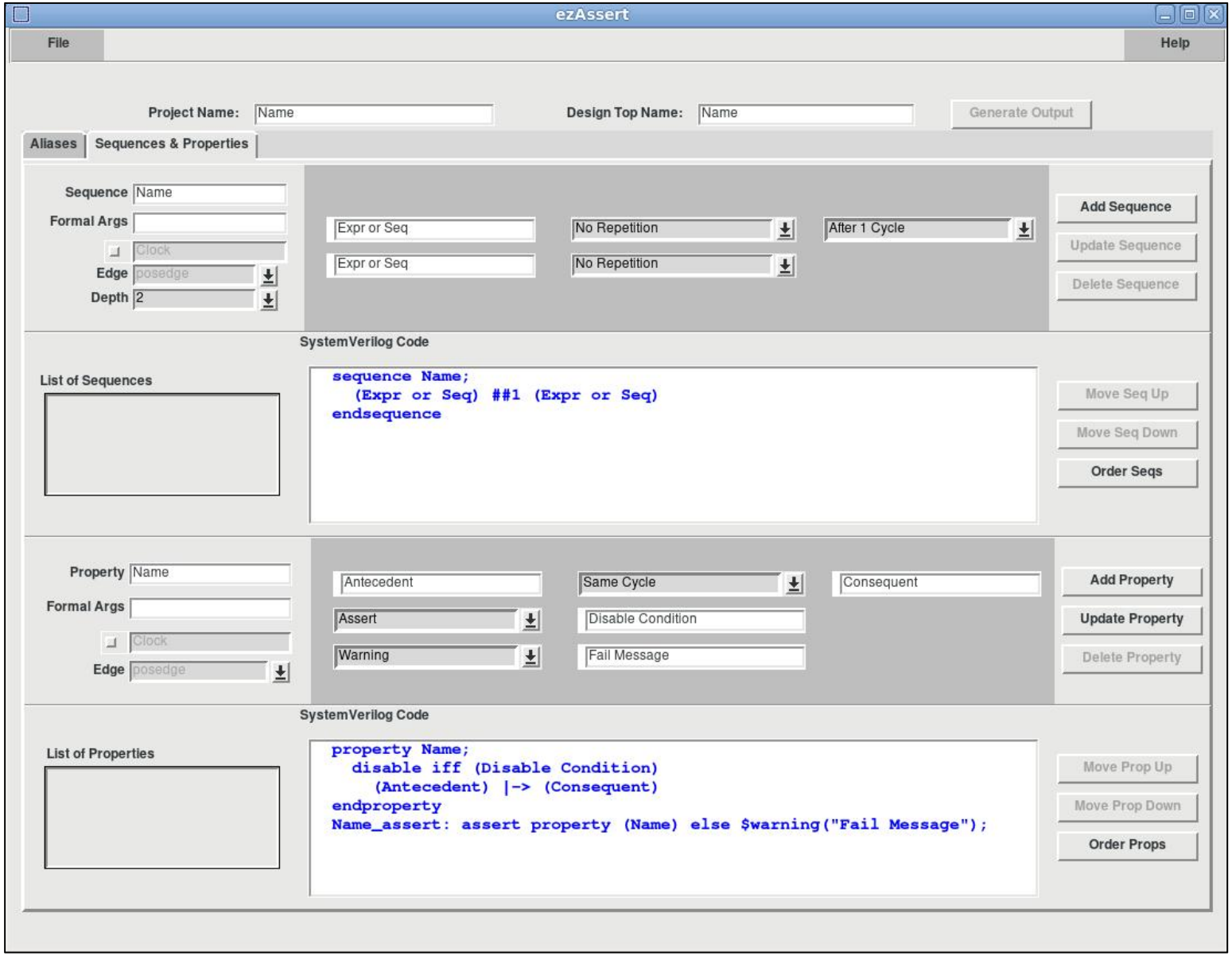

```
ovl_always #(

    `OVL_ERROR,                                  // severity_level
    `OVL_ASSERT,                                 // property_type
    "Error: reg_a < reg_b is not TRUE",          // msg
    `OVL_COVER_NONE,                             // coverage_level
    `OVL_POSEDGE,                                // clock_edge
    `OVL_ACTIVE_LOW,                             // reset_polarity
    `OVL_GATE_CLOCK)                             // gating_type


   reg_a_lt_reg_b (

      clock,                                     // clock
      reset,                                     // reset
      enable,                                    // enable
      reg_a < reg_b,                             // test_expr
      fire);                                     // fire
```

Checks that (*reg_a* < *reg_b*) is TRUE at each rising edge of *clock*.



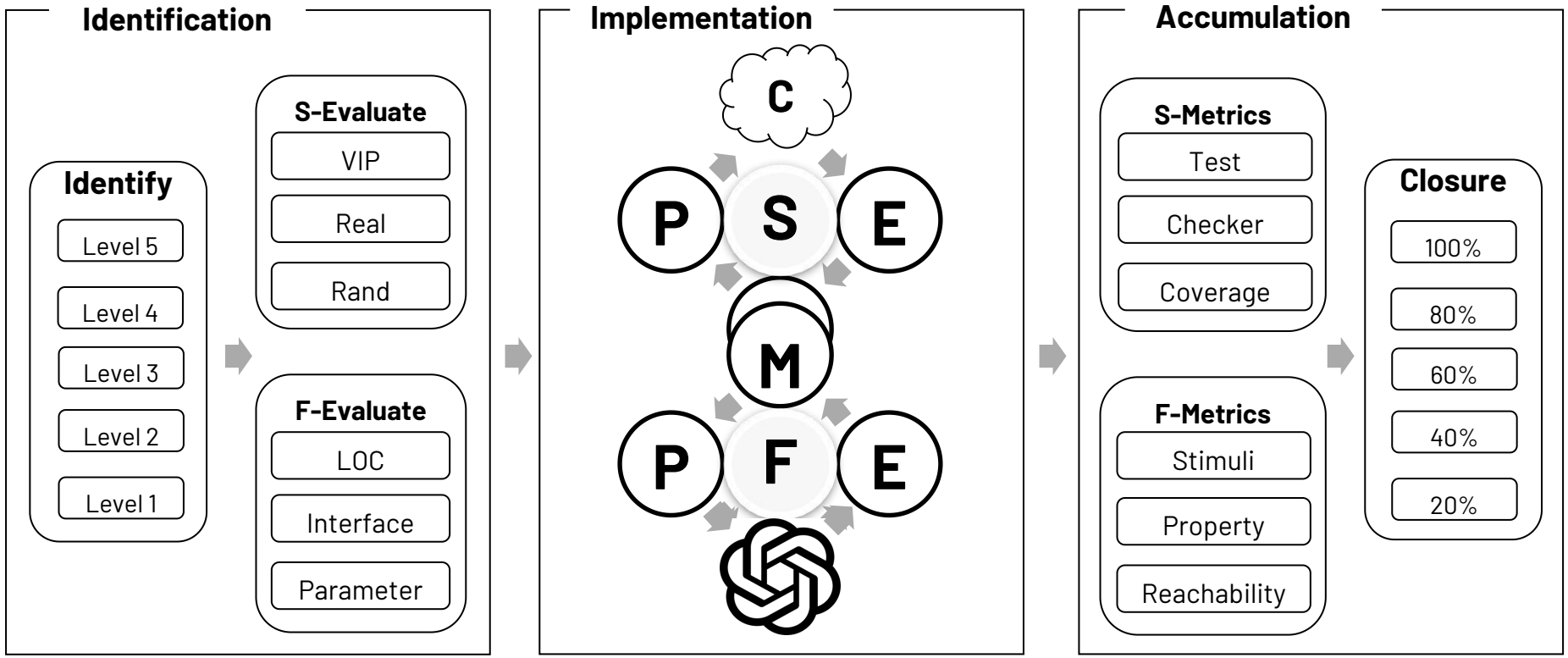
- The OVL is composed of a set of assertion checkers that verify specific properties of a design.
- The ovl_always assertion checks the single-bit expression test_expr at each active edge of the clock.
- Although the OVL can save some time, the limited templates still can't fulfill all the scenarios, especially those highly customized blocks.

ezAssert



- ezAssert is one of our in-house GUI-based assertion development tools
- By using ezAssert, we can implement specification-oriented programming instead of facing the syntax of SVA.
- However, you still need to describe the assertion by using the primitives of ezAssert instead of using SVA directly. And this is the most difficult step.
- ezAssert is not easy and intelligent enough to help you finish the translation from human language to computer language SVA directly.

Formal Added-on MDV Hybrid Strategy



Formal Added-on MDV Hybrid Strategy

- S1: Assertion translation from human language to computer language by using GPT
- S2: Assertion documentation from incomprehensible to understandable by using GPT
- S3: Assertion validation from syntax-wrong to syntax-correct by using GPT
- S4: Assertion optimization from low quality to high quality by using GPT
- S5: Assertion completion from pieces to integrity by using GPT

Translation of assertions

```
Create an assertion to make sure all the signals are stable
during the wait state according to the protocol of AHB3 lite
```

```
// Assertion to check signal stability during the wait state
assert property (
  @(posedge clk)
    disable iff (!hsel || !hready)
    ##1 $stable(hready) && $stable(hsel)
)
$error("AHB3 Lite protocol violation: Signals not stable during wait state at time %t", $time);
```

ARM AMBA
Interconnect Standards

```
property ahb3_lite_master_output_stable;
  @(posedge hclk) disable iff (!hresetn || hready)
    $stable(htrans) && $stable(hsel) && $stable(hwrite) && $stable(hsize) &&
    $stable(hburst) && $stable(hprot) && $stable(haddr) && $stable(hwrite_data);
endproperty

assert property (ahb3_lite_master_output_stable)
  else
    $error("Master output signal changed during wait state");
```

Documentation of assertions

Can you explain the meaning of this assertion?

```
property HS2;
    @(posedge clk) (Req ##1 GntB) |=> (Busy[*1:$] ##1 Done);
endproperty
```

The provided assertion defines a SystemVerilog property named `HS2`.
Let's break down its meaning step by step:

1. `property HS2;`: This line starts the definition of a new property named `HS2`.

2. `@(posedge clk)`: This is an implication operator (`|=>`) that specifies the timing of the property.
   It means that the property is evaluated at the positive edge of the `clk` signal.

3. `(Req ##1 GntB)`: This is the antecedent (left-hand side) of the implication.
   It is a concatenation of two sequences:

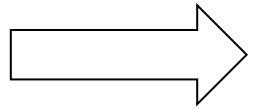
   - `Req`: This represents a signal/event named `Req`.
   - `GntB`: This represents a signal/event named `GntB`.

   The `##1` between them means that `Req` and `GntB` must occur back-to-back, i.e.,
   `Req` must occur one clock cycle before `GntB`.

4. `|=>`: This is the implication operator, which means "implies" or "leads to."
   It indicates that if the antecedent is true, the consequent must also be true.

5. `(Busy[*1:$] ##1 Done)`: This is the consequent (right-hand side) of the implication.
   It is a concatenation of two sequences:

   - `Busy[*1:$]`: This is a non-empty sequence of events represented by the signal/event `Busy`.
   The `[*1:$]` means that `Busy` must occur one or more times continuously.
   - `##1`: This means that there is a gap of exactly one clock cycle between the end of the `Busy` sequence
            and the start of the `Done` sequence.
   - `Done`: This represents a signal/event named `Done`.



Putting it all together, the meaning of the `HS2` property is:

"When at a positive edge of the `clk`, if there is a sequence of events where `Req` and `GntB` occur back-to-back, it implies that there must be a non-empty sequence of `Busy` events followed by a `Done` sequence."

In simpler terms, this property specifies a protocol where when a request (`Req`) is followed immediately by a grant (`GntB`), with a one-clock cycle gap between the last `Busy` and the `Done` events.

Agenda

- Introduction
  - History of formal
  - Algorithm of formal
  - Classification of formal
  - Future of formal
- Case Study
  - Functional Safety (Level 1)
  - Connectivity (Level 2)
  - MHDMA (Level 3)
  - Pyro-Fuse (Level 4)
- Autogeneration
  - Open Verification Library
  - ezAssert
  - ChatGPT
- **Conclusion**
  - **Summary**
  - **Future work**
  - **Reference**

Summary

Author's Viewpoints

- Useful for block-level verification such as DMA, Bus Matrix, Arbiter, Buffer control logic, Cache, MMU, protocol checking etc. A good criterion is ~3000 lines valid RTL code. (Confirmed by Cadence AE)
- Designer friendly. More than the structural errors found by Lint, functional errors can also be detected by Formal. Fill the gap before simulation testbench is ready and implement the shift-left verification.
- Supplyment of simulation for corner cases, unreachable coverage, xprop, connectivity checking etc.
- Replacement of simulation? Not now, due to both human resource and computing power to design maitain and execute the assertions.

- Pros
  - Easy to set up and start the verification. No testbench is needed.
  - Be able to discover errors you never anticipate.
  - AIGC improves the performance of the autogeneration and assertions can be re-used.
  - Many scenarios can be applied with all these Jasper Apps.
- Cons
  - Proving time can't be accepted with a long data path or a huge number of states.
  - RAM-hungry especially enables the coverage collection.
  - FPV convergence issues.

Future work

- Optimize the convergency issue by using helper assertion, abstraction, etc.
- Develop more reusable assertion suites.
- Implement the automation flow.
- Explore the API of ChatGPT.

Reference

[1] Harry Foster, "2022 Wilson Research Group Functional Verification Study", Verification Horizons, 18 December 2022

[2] Erik Seligman, Tom Schubert, M V Achutha Kiran Kumar. (2015). "Formal Verification. An Essential Toolkit for Modern VLSI Design". Elsevier Inc.

[3] Luiza Pena. (2021). "Introduction to formal and jasper gold". Jasper User Group Webinar.

[4] Ping Yeung, Arun Khurana, Dhruv Gupta, Ashutosh Prasad, Achin Mittal. (2022). DV CON US.

Helpful Links:

- https://support.cadence.com
- https://www.eeweb.com/a-brief-history-of-formal-verification/

Thank You!