

2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Next Frontier in Formal Verification

Ping Yeung, Rajesh Rathi, Vaibhav Kumar,
Puneet Anand, Ravindra Aneja



Introductions : Ravindra Aneja (Synopsys)



Ravindra Aneja is Director, Applications Engineering at Synopsys, responsible for formal verification solutions. He has 25+ years of technical marketing and application engineering experience in functional verification domain which includes simulation, hardware acceleration, emulation, assertion based formal verification and clock domain crossing verification. Before Joining Synopsys, Ravindra worked at Atrenta, Mentor Graphics, 0-In Design Automation, IKOS Systems and Interra Inc.

Ravindra is co-chair of DAC Front-end engineering track TPC.

Introductions : Ping Yeung (NVIDIA)



Ping Yeung, Ph.D. is a Senior Manager at Nvidia. He was part of the team that developed and introduced Assertion-Based Formal Verification to the electronic design industry. He has over 25 years of experience, including positions at O-In, Synopsys, Mentor Graphics, and Siemens EDA. He has published 30+ papers and holds 7 patents in the CDC and formal verification area.

Introductions : Rajesh Rathi (Google)



Rajesh Rathi is currently working at Google as a formal verification engineer. He has 16 years of experience doing both simulation and formal verification at companies such as Samsung and Broadcom.

Introductions : Vaibhav Kumar (NXP)



Vaibhav has graduated from Dayalbagh Educational Institute in conjunction with University of Maryland. He has overall 15+ years of experience in SoC verification, validation, emulation and IP verification. Currently he manages team of 15+ IP verification engineers based out at NXP Austin and India.

He drives overall verification methodology adaption for various verification initiatives in Digital IP group at NXP. He has publications and interest in areas of formal verification & functional safety.

Introductions : Puneet Anand (Qualcomm)



Puneet Anand leads the formal verification effort for the Qualcomm GPU organization, across North America, India & Europe. He is keenly focused on deploying traditional and advanced FPV and formal app-based solutions.

He spent last few years working on GPU & ML ASICs DV and Formal Verification. At his previous job at Meta, he was leading Formal Verification for the Inference/Training, Video Transcoding and Connectivity ASICs, as well as leading Simulation DV for Cache subsystem.

Outside of work, his personal interests are Reading, playing and watching Tennis, volunteering as a Level 3 Official in his kids' Swim team and a Jr. FLL robotics Coach.

Formal Verification - Motivation

- Growing trend to look for better verification solutions
 - Current flows not scaling
 - Complexity continues to grow
 - Time to market pressure continues to mount



Does not scale, controllability challenges



Too late and expensive to find bugs

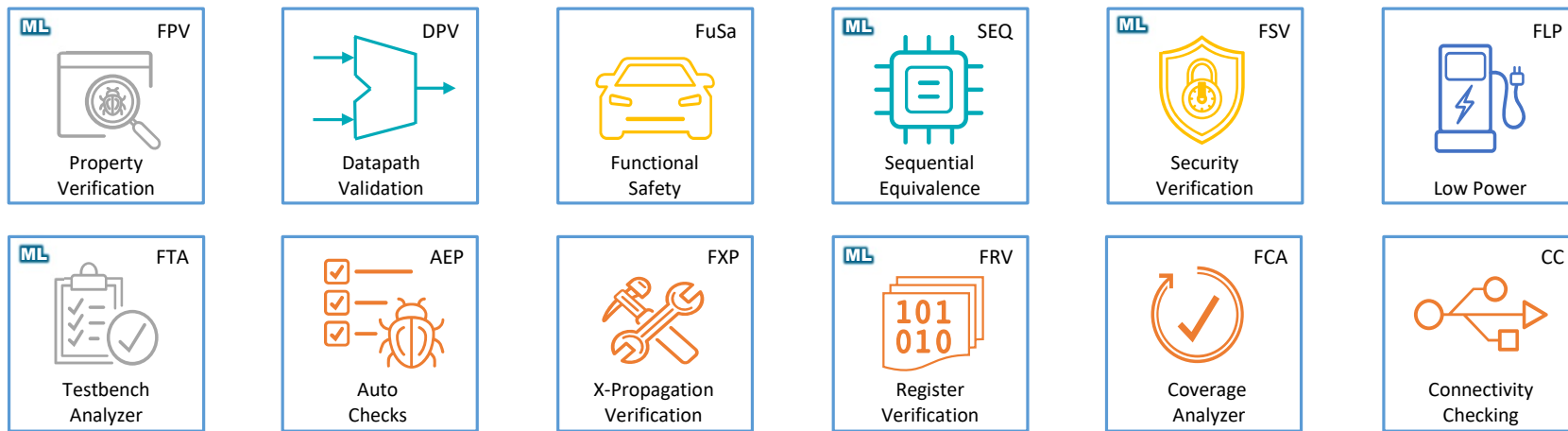
- Background
 - UVM based Simulation environment
 - FPGA prototyping/Emulation

Formal Addresses these Verification Challenges

Factors Driving Formal Adoption

- Availability of easy-to-use Formal Apps
 - Control path and data path
 - Security, Safety, low power
 - Coverage closure
 - Signoff
- Continuous capacity and performance improvements in formal verification products, example: Synopsys VC Formal
- Leveraging AI/ML techniques for faster performance and actionable feedback to the users to manage complexity
- Simulation like coverage-based signoff criteria

Synopsys VC Formal Apps : Targeted for Specific Problems



Block/IP

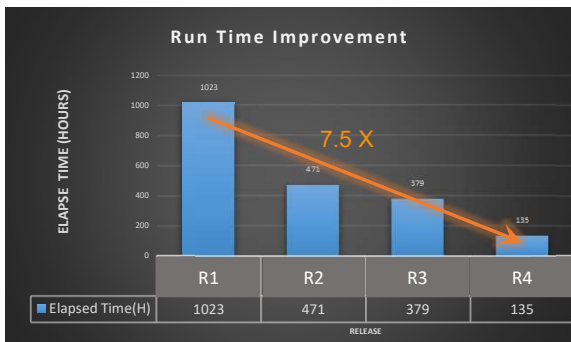
Subsystem

SoC

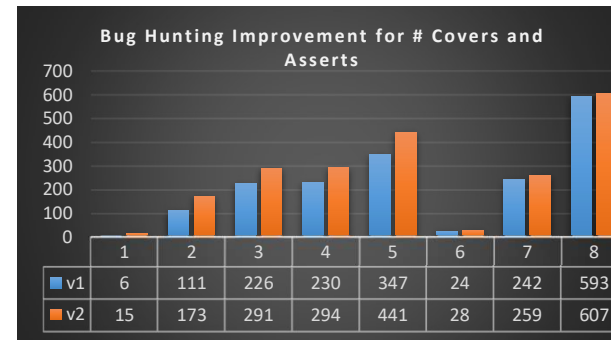
High Performance: ML powered proprietary engines for hard proofs, liveness, and deep bug-hunting
High Confidence Formal Signoff: Native Certitude integration for fast and high-quality Formal Signoff

VC Formal: Performance Improvements

2X Faster Run Time

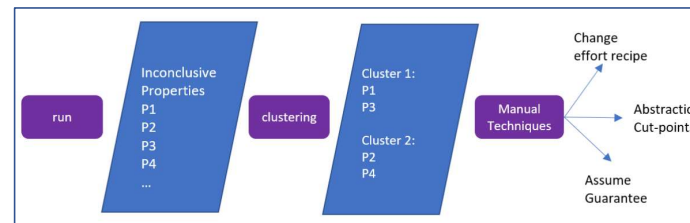
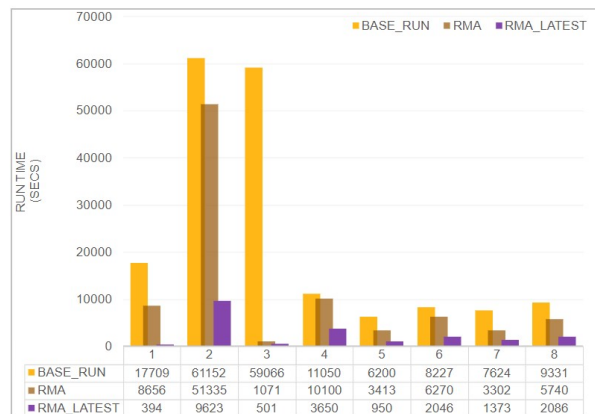


- Up to 7.5X faster TAT
- Faster proofs with new engines
- Release-over-release improvement



- Bug Hunting specific performance improvements
- Automated cover properties generation
- Custom orchestration for expert users

Leverage AI/ML for Faster Convergence



ML for Performance and Debug

- 10X performance speed-up through RMA
- More robust RMA across RTL changes
- ML-based clustering inconclusive properties

Simulation Like Signoff Criteria

- Generate coverage for verified properties
- Merge it with simulation coverage
- Add waivers for non-functional logic

Name	Score	Line	Toggle	FSM	Condition	Branch
soc_tb_top	20.46%	58.72%	3.12%	1.57%	16.88%	42.27%
Bfm0	36.36%					
dut	20.46%	58.72%	3.12%	1.57%	16.88%	42.27%
dcm0_ram	0.00%					
dut	43.22%	87.28%	10.76%	14.12%	36.71%	67.21%
l_axi	50.41%	88.58%	8.28%		33.66%	71.13%
l_axi_a2x	60.30%	87.78%	32.50%		52.61%	68.31%
l_axi_a2x_gic	40.61%	79.42%	0.69%		36.35%	45.99%
l_axi_a2x_pcie	40.63%	79.47%	0.69%		36.38%	45.99%
l_axi_x2p	55.07%	86.06%	52.17%	13.59%	55.78%	67.74%
l_timers	49.79%	85.14%	4.69%		48.50%	60.84%
l_uart	46.86%	80.09%	18.63%	14.86%	58.21%	62.52%
l_alb_mss_mem_l...	67.97%	87.84%	46.47%		55.03%	82.54%
l_coresight	25.09%	59.41%	2.78%	0.65%	28.07%	34.52%
l_ufshc	18.38%	48.17%	2.19%	0.72%	4.33%	36.50%

Assert	Type	Success/Match	FVstatus	FVdepth	Attempt	Incomplete
fsm.a1	Assertion		proved		1	0
fsm.a2	Assertion		proved	65	1	0
fsm.a_complete_frame	Assertion		proved		1	0
fsm.a_loop_break	Assertion		inconclusive	70	1	0
fsm.a_onehot	Assertion		proved		1	0
fsm.a_trans	Assertion		proved		1	0
fsm.c1	Cover Property		proved		1	0
fsm.c2	Cover Property		inconclusive	70	1	0
fsm.c_blk_cnt	Cover Property		covered	4	1	0
fsm.c_onehot	Cover Property		covered		1	0
fsm.c_trans	Cover Property		covered		0	0

Data Path Validation

- Unique formal engines to provide conclusive answers to math problems
 - Widespread used in CPU/GPU/AI/ML designs
- Bring-in traditional model checking features into data path validation world
 - Coverage
 - Vacuity, witness, SVA style properties
 - Visibility into engines performance

New Avenues and Pushing the Envelope

- End-2-end property verification
- Data path validation
- Security and safety verification
- Sequential equivalence for most advanced clock gating designs

2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION

UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Formal Signoff with End-to-End Checking Methodology

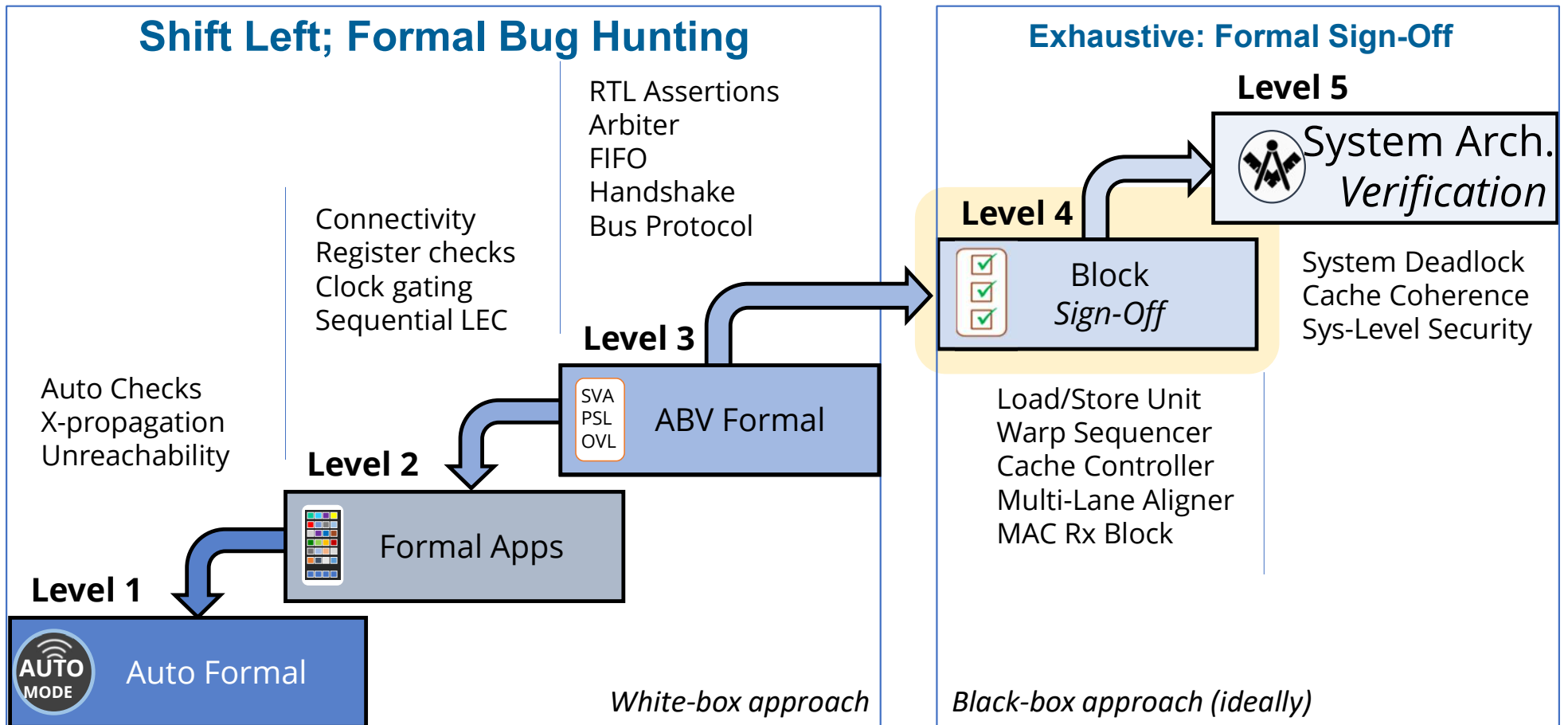
Ping Yeung, Nvidia



Agenda

- Formal Verification Usage Levels
- End-to-End Checking Methodology
- End-to-End Checkers
- Abstraction Techniques and Modeling
- Testcases
 - Parameterized Multi-cast Crossbar Design
 - Request Coalescer Unit
 - NOC Configurable Cache Controller

Formal Verification Usage Levels



Block-Level Formal Signoff

Different from traditional Assertion-based Verification

- Black-box approach; use end-to-end checkers; does not depend on RTL
- Divide-and-conquer with multiple formal testbenches

Early deployment

- Identify incomplete or ambiguous specifications early in the design cycle,
- Provide clear value to the project team because they map directly to the functional specification
- Find more bugs and verify the block while the designer is coding the RTL

Exhaustiveness

- Replace simulation entirely and do a formal signoff of the block,
- Find deep or unaware corner case issues

Reusability

- Reuse the formal testbench with updated RTL to quickly confirm a fix or find new issues

Level 4



Formal Signoff Bugs Found

Project 1

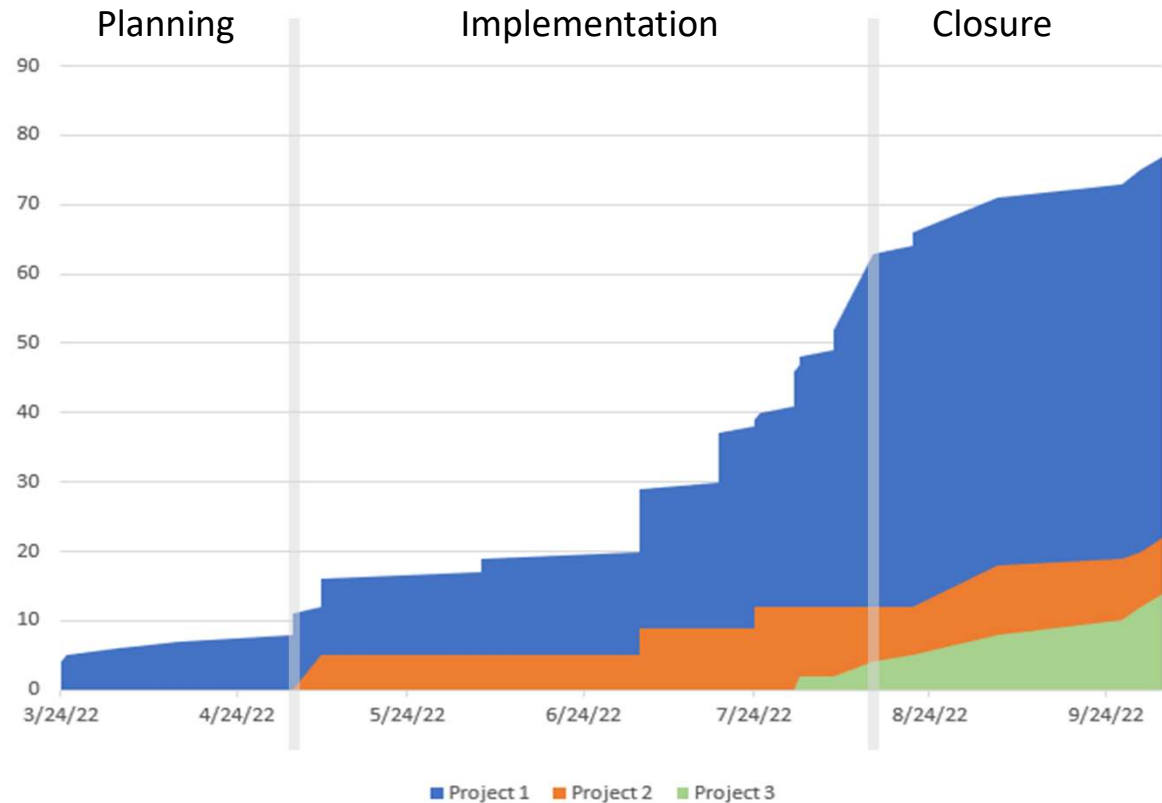
- Engage before RTL coding
- Coding I/Fs + end-to-end checkers when designers coding RTL
- Signoff before top-level simulation
- Found simple + complex issues

Project 2

- Reused block with additional functionalities
- Coding I/Fs + end-to-end checkers when designers *updating* RTL
- Concurrent with existing simulation
- Found complex issues

Project 3

- Updated block with standard interfaces
- Formal VIP + end-to-end checkers
- Found complex issues



Agenda

- Formal Verification Usage Levels
- End-to-End Checking Methodology
- End-to-End Checkers
- Abstraction Techniques and Modeling
- Testcases
 - Parameterized Multi-cast Crossbar Design
 - Request Coalescer Unit
 - NOC Configurable Cache Controller

End-to-End Checking Methodology

Task	Planning	Implementation	Closure
Management	Formal expertise Schedule & milestones	Allocate formal engineer resources	Plan extra compute, vendor resources

Management

- Need a team of formal experts and engineers
 - Formal experts with years of experience required for formal planning
 - Formal engineers required for formal testbench implementation
 - Careful partnering of formal engineers with design team members
- Need compute resources and vendor expertise
 - Server farm environment for formal coverage and final signoff
 - Vendor expertise to address some difficult properties

End-to-End Checking Methodology

Task	Planning
Management	Formal expertise Schedule & milestones
Block	Identify and Evaluate
Function	Describe and Prioritize
Complexity	Decompose and Map

Block

- Identify blocks for E2E formal
- Evaluate to determine effort

Function

- Describe E2E functionality
- Prioritize them based on importance/risk

Complexity

- Decompose, divide-and-conquer
- Map them to one or more formal TBs



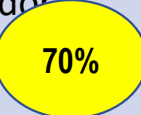







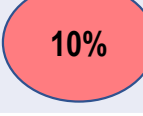

End-to-End Checking Methodology

Task	Planning	Implementation
Management	Formal expertise Schedule & milestones	Allocate formal engineer resources
Block	Identify and Evaluate	Capture Interfaces
Function	Describe and Prioritize	End-to-End Checkers
Complexity	Decompose and Map	Abstraction Techniques

End-to-End Checking Methodology

Task	Planning	Implementation	Closure
Management	Formal expertise Schedule & milestones	Allocate formal engineer resources	Plan extra compute, vendor resources
Block	Identify and Evaluate	Capture Interfaces	Validate Constraints
Function	Describe and Prioritize	End-to-End Checkers	Conclusiveness
Complexity	Decompose and Map	Abstraction Techniques	Formal Coverage

End-to-End Checking Methodology Milestones

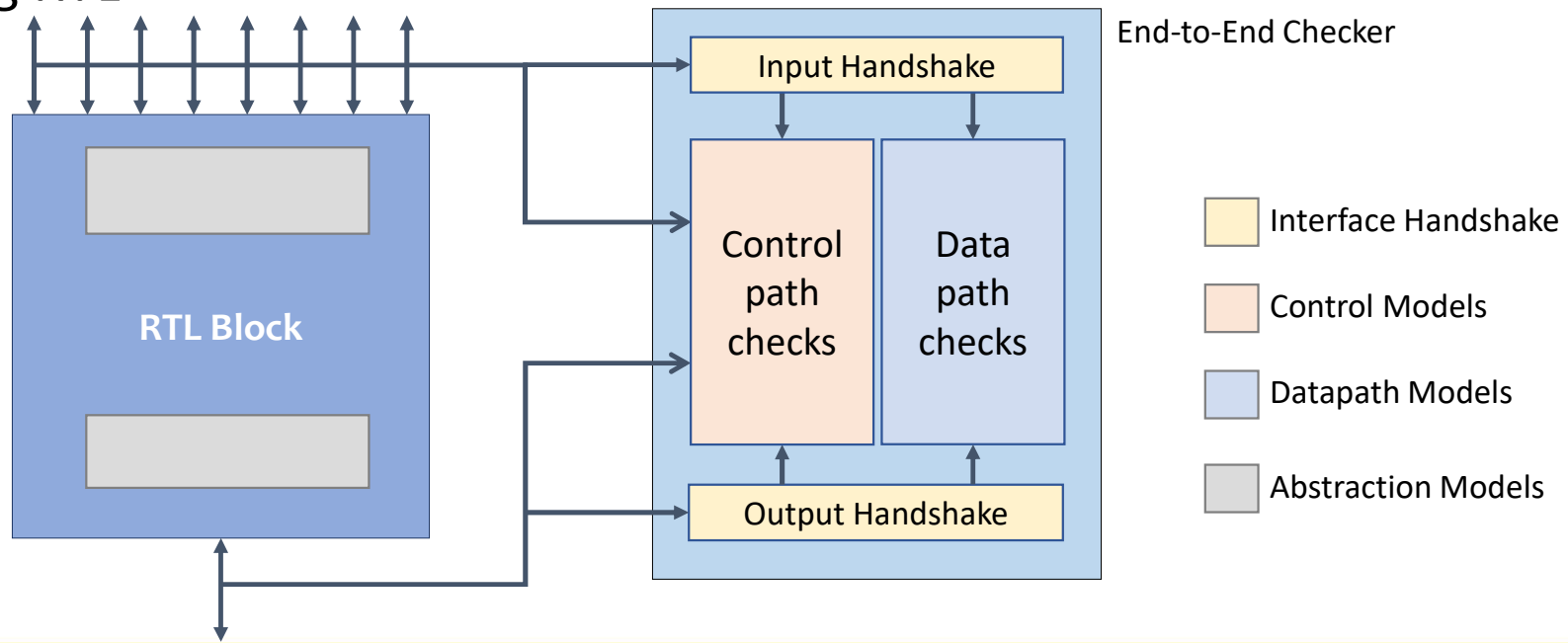
Task	Planning	Implementation	Closure
Management	Formal expertise Schedule & milestones 	Allocate formal engineer resources 	Plan extra compute, vendor resources 
Block	Identify and Evaluate 	Capture Interfaces 	Validate Constraints 
Function	Describe and Prioritize 	End-to-End Checkers 	Conclusiveness 
Complexity	Decompose and Map 	Abstraction Techniques 	Formal Coverage 

Agenda

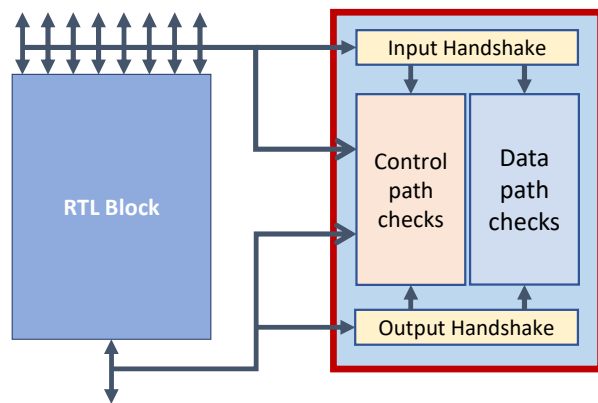
- Formal Verification Usage Levels
- End-to-End Checking Methodology
- End-to-End Checkers
- Abstraction Techniques and Modeling
- Testcases
 - Parameterized Multi-cast Crossbar Design
 - Request Coalescer Unit
 - NOC Configurable Cache Controller

End-to-End Checkers

Developing formal-friendly reference model could be as big an effort as writing RTL

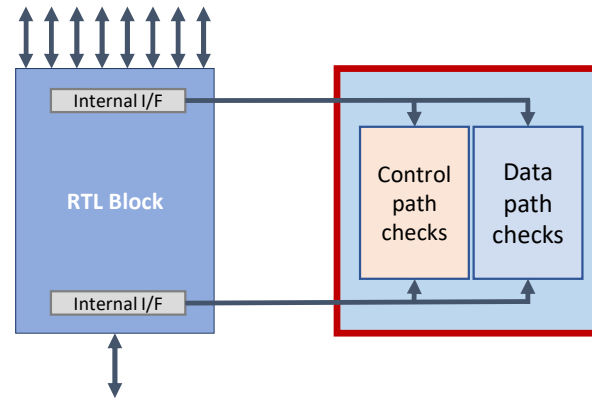


Formal Testbench Configurations



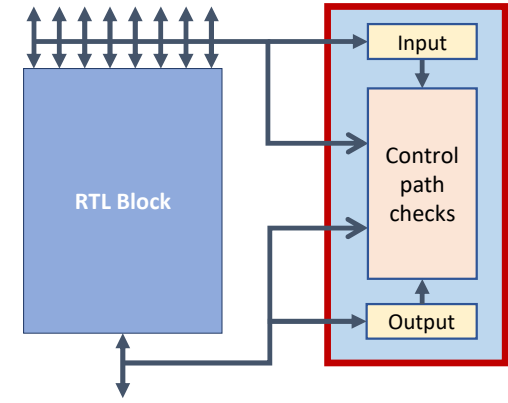
Formal TB = DUT

- ✓ Match formal TB to whole DUT
- ✓ Well defined input/output interfaces
- × May have high complexity



Formal TB = (DUT - Interfaces)

- ✓ Useful to bypass complex interfaces
- ✓ Reduce design/proof depth
- × Undocumented interfaces



Formal TB = (partial DUT)

- ✓ Divide-and-conquer approach
- ✓ Reduce design/formal complexity
- × Incomplete functionality

Abstraction Techniques

Abstraction Technique	Design Complexity	Formal Efficiency
Case splitting	Multiple runs with different cases reducing design complexity per run/case	Reduce COI, reduce state space per run/case
Cut-point/ Black box	Eliminate logic driving cut-points/inside blackbox	Reduce COI, state space; controlled with constraints

Abstraction Techniques

Abstraction Technique	Design Complexity	Formal Efficiency
Case splitting	Multiple runs with different cases reducing design complexity per run/case	Reduce COI, reduce state space per run/case
Cut-point/ Black box	Eliminate logic driving cut-points/inside blackbox	Increase flexibility but controlled with constraints
Reset abstraction	n.a.	Reduce access depth
Counter abstraction	n.a.	Reduce the length of counting

Abstraction Modeling 1

Abstraction Model	Design Complexity	Formal Efficiency
Symmetric data elements	Eliminate multiple dimensional data elements; add single dimension abstraction model	Reduce COI and state space with symmetry

Abstraction Modeling 1

Abstraction Model	Design Complexity	Formal Efficiency
Symmetric data elements	Eliminate multiple dimensional data elements; add single dimension abstraction model	Reduce COI and state space with symmetry
RTL model	Abstraction model	
<pre> element_type [SIZE-1:0] element; element [addr] = wr_data; rd_data = element [addr]; </pre>	<pre> element_type abs_element; if (addr == sym_addr) abs_element = wr_data; if (addr == sym_addr) rd_data = abs_element; \$stable (sym_addr) </pre>	

Abstraction Modeling 2

Abstraction Model	Design Complexity	Formal Efficiency
Memory abstraction	Represent one location instead of the full size of the memory	Reduce COI and state space with symmetry

RTL memory:

```
reg [WIDTH-1:0] mem [DEPTH-1:0];
```

abstraction memory:

```
reg [WIDTH-1:0] mem;
```

assume property:

```
(sym_addr < DEPTH) ##1 $stable(sym_addr)
```

abstraction write:

```
if (wr && (wr_addr == sym_addr)) mem <= wr_data;
```

abstraction read:

```
if (rd && (rd_addr == sym_addr)) rd_data = mem;
```

Abstraction Modeling 3

Abstraction Model	Design Complexity	Formal Efficiency
FIFO	Eliminate logic before cut-points; add abstraction model	Reduce the depth of the FIFO

```
wire [LOG_DEPTH-1:0] sym_depth;  
assume property: (sym_depth > 1 && sym_depth < DEPTH) ##1 $stable(sym_depth)  
abstraction model: if (wr_ptr == sym_depth) wr_ptr <= 0;  
else wr_ptr <= wr_ptr + 1;
```

Abstraction Modeling 4

Abstraction Model	Design Complexity	Formal Efficiency
Data independence (Wolper Coloring)	Eliminate all storage elements; add Wolper FSMs	Reduce COI with pattern

The rules for generating and verifying the Wolper sequence are:

1. If the first 1 is seen, next one should be 1

wolper_1st_1_seen_next_1: (first_one && !second_one && input_valid) |-> (colored_input == 1'b1)

2. If two 1's are seen, only 0's should be seen

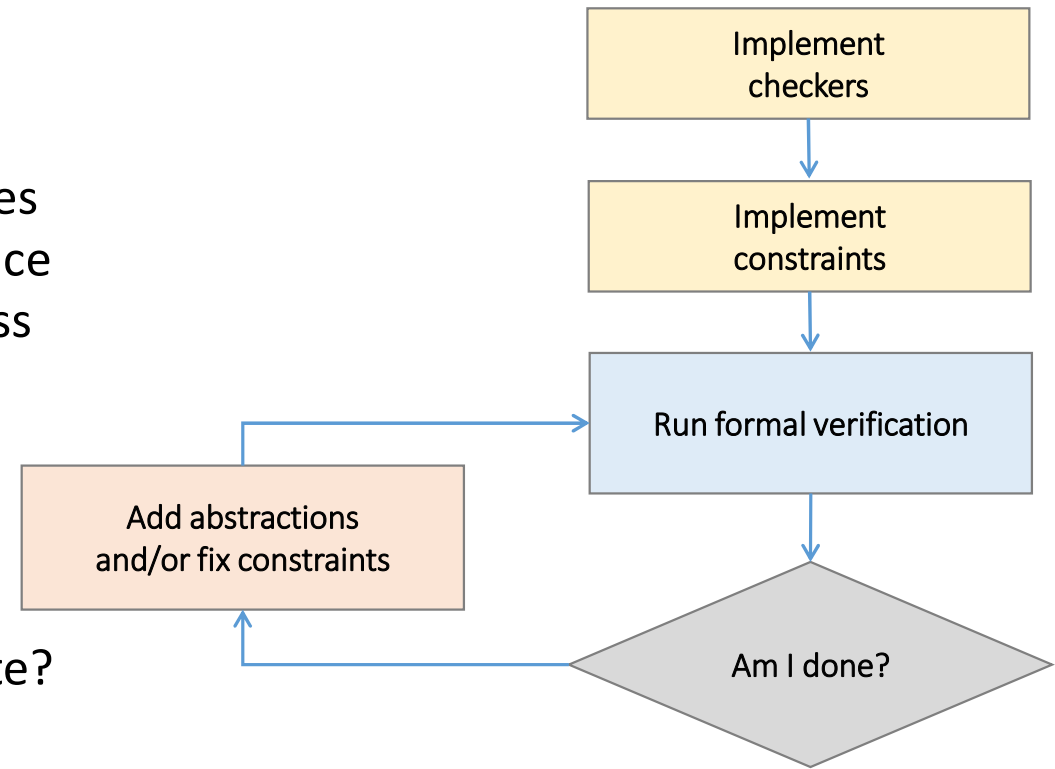
wolper_2nd_1_seen_forever_0: (second_one && input_valid) |-> (colored_input == 1'b0)

Abstraction Modeling Summary

Abstraction Modeling	Design Complexity	Formal Efficiency
Symmetric data elements	Eliminate multiple dimensional data elements; add single dimension abstraction model	Reduce COI and state space with symmetry
Memory abstraction	Represent one location instead of the full size of the memory	Reduce COI and state space with symmetry
FIFO	Eliminate logic before cut-points; add abstraction model	Reduce the depth of the FIFO
Data independence (Wolper Coloring)	Eliminate all storage elements; add Wolper FSMs	Reduce COI with pattern
Tagging	Represent one tag instead of the complete linked list	Reduce COI

Formal Sign-off

- Methodology
 - End-to-End Checkers
 - Constraints to control the interfaces
 - Abstractions to achieve convergence
 - Coverage to measure completeness
- Am I done?
 - Are my Checkers complete?
 - Are my Constraints weak enough?
 - Is my Complexity strategy complete?
 - Is my Coverage goal met?

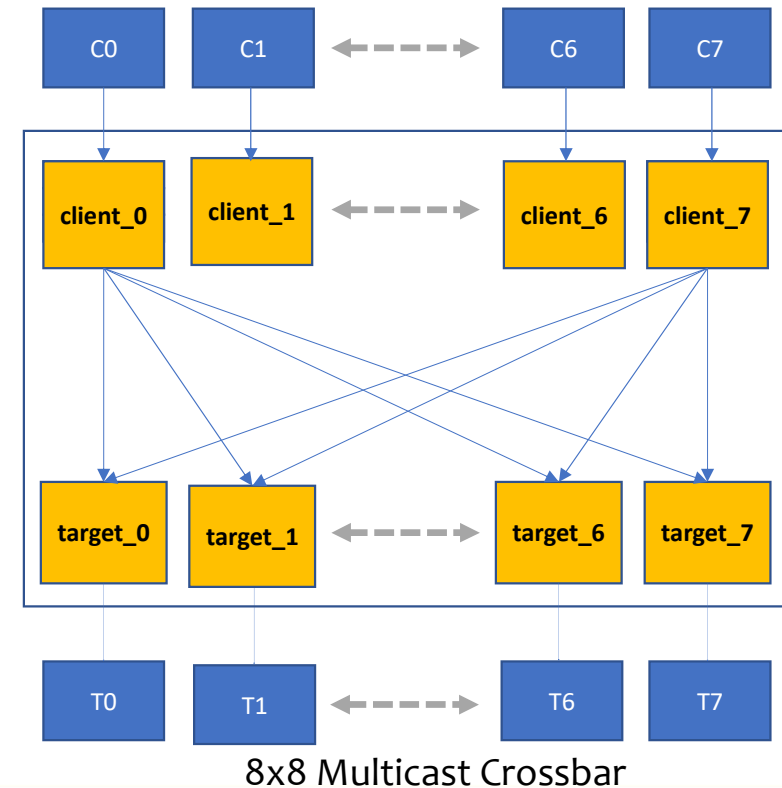


Agenda

- Formal Verification Usage Levels
- End-to-End Checking Methodology
- End-to-End Checkers
- Abstraction Techniques and Modeling
- Testcases
 - Parameterized Multi-cast Crossbar Design
 - Request Coalescer Unit
 - NOC Configurable Cache Controller

Parameterized Multi-cast Crossbar Design

- 8x8 Crossbar design
 - Each client can send request to 1+ targets
 - Each target has an arbiter to decide which request gets forwarded based on priorities
- Abstraction Deployed
 - Symbolic variables used to select a client/target and implemented all of the checkers for the symbolic client and target pair.
 - Formal explore all possible values for the symbolic variables



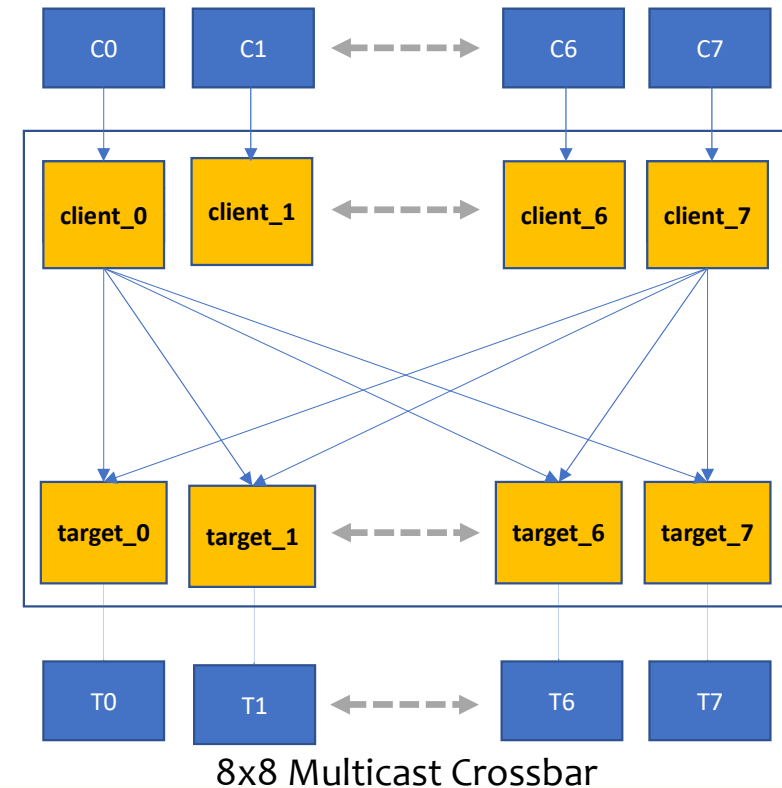
Control Path and Data Path Checkers

Multi-cast Crossbar Design:

- Control path end-to-end checkers:
 - An arbitration checker (a combination of two checkers) for the arbitration scheme
 - A consistency checker to ensure no spurious grant is given to a client
 - Performance checkers to ensure operations are performed in each cycle when the conditions are met.
- Data path end-to-end checkers:
 - Data integrity checkers to ensure correct transfer
 - from read data input port to buffer
 - from buffer to store output port.
 - data is not corrupted, duplicated, reordered, or dropped.
 - Wolper coloring technique: doesn't require data storage

Parameterized Multi-cast Crossbar Design

- Design used in the "Break the Testbench" challenge at DAC 2015.
 - Attendees were invited to insert functional bugs
- All 73 inserted RTL bugs
 - Exposed by one or more checkers
 - Excellent exercise to demonstrate that end-to-end checking is a comprehensive methodology.
- Abstraction Deployed
 - symbolic variables used to select a client/target and implemented all of the checkers for the symbolic client and target pair.
 - Formal explores all possible values for the symbolic variables



End-to-End Checking Methodology

Task	Planning	Implementation	Closure
Management	Formal expertise Schedule & milestones	Allocate formal engineer resources	Plan extra compute, vendor resources
Block	Identify and Evaluate	Capture Interfaces	Validate Constraints
Function	Describe and Prioritize	End-to-End Checkers	Conclusiveness inconclusives
Complexity	Decompose and Map	Abstraction Techniques	Formal Coverage



Summary

- Block-level Formal Signoff with End-to-End Checking Methodology
 - End-to-End Checkers
 - Abstraction Techniques and Modeling
 - Comprehensive for block-level formal signoff
- Major benefits
 - **Reduce time to First Bug:** Shift-Left “Avoidable Bugs”
 - **Reduce time to Last Bug:** Eliminate “Inevitable Bugs”
- Acknowledgement
 - The support of the whole Nvidia Formal Team in Gurugram, India.

Q&A



Datapath Formal Verification of Crypto Accelerators and Multipliers using VC Formal

Rajesh Rathi, Google Inc
Manish Harnur, Google Inc



Agenda

- Motivation
- FP32 Multiplier C-to-RTL
- Galois Field Multiplier C-to-RTL
- Cryptographic accelerator AES C-to-RTL
- AES with obfuscation RTL-to-RTL
- Conclusion

Motivation

- DPV enables delivering higher design quality
 - Impractical to verify datapath designs with huge state space exhaustively with simulation
 - DPV proves absence of bugs
- DPV can be more efficient than simulation
 - In case of AES, simulation reduced the stimuli to a manageable subset and still able to get high confidence in verification
 - Simulation has overheads of testbench development, stimulus generation and coverage closure

FP32 Multiplier C-to-RTL Equivalence

- Off-the-shelf C model from SoftFloat
- Proof did not converge in 24 hours with standard solvers
 - Expected to converge easily
- Likely that the tool struggled to find common points between spec and impl
 - Upon inspection, our design implementation is non-standard

FP32 Multiplier - Convergence Strategy

- Used assume - guarantee
- Prove product of mantissa with HDPS engine.

*lemma impl.partial_prod[47:0](2) == {1'b1, impl.io_x(2)[22:0]} * {1'b1, impl.io_y(2)[22:0]}*

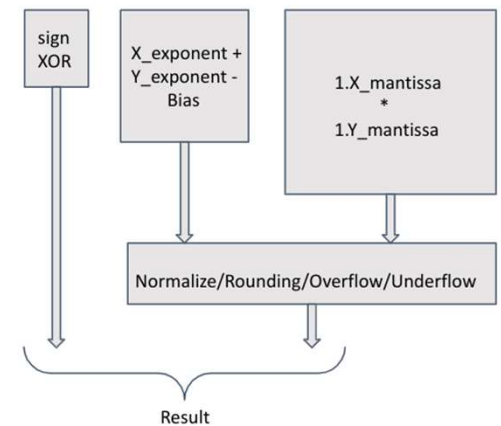
- Then use that as an assumption to prove the rest of the logic with standard solvers.

*assume impl.partial_prod[47:0](2) == {1'b1, impl.io_x(2)[22:0]} * {1'b1, impl.io_y(2)[22:0]}*

lemma z_check = spec.z(1) == impl.mul_out(2)

- Proof converged within 5 minutes

FP32 multiplier implementation



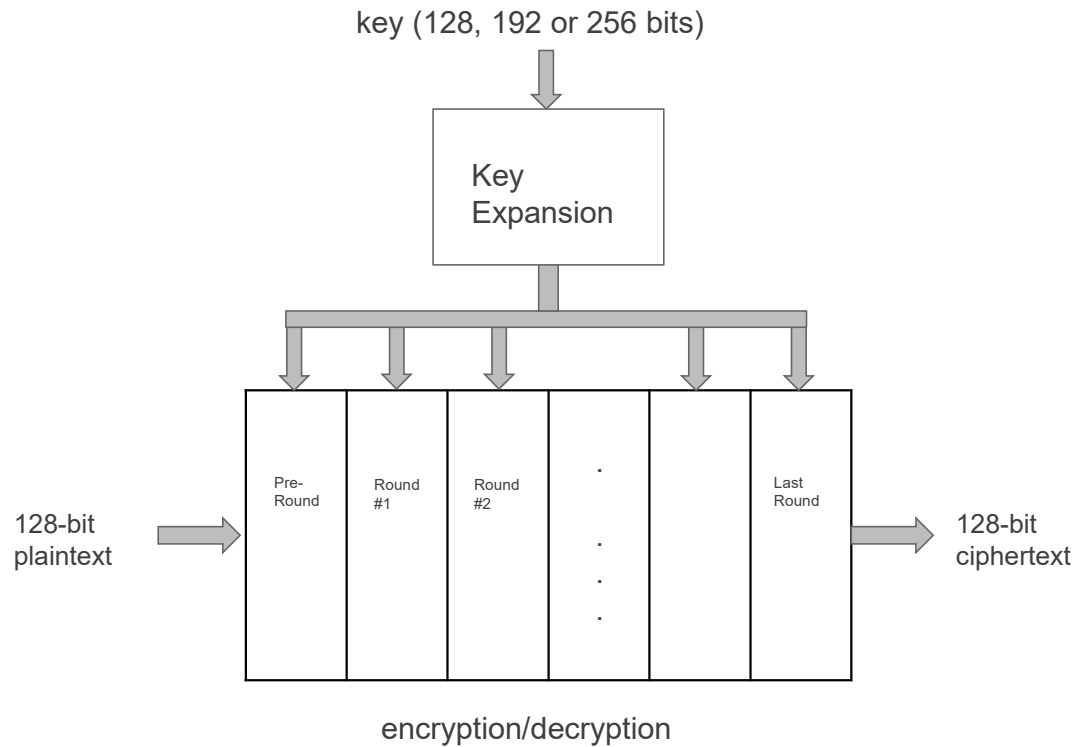
Galois Field Multiplier

- Math building block used in our security designs
- Two designs
 - First design: two inputs of 128 bits each and output of 128 bits
 - Second design: two inputs of 64 bits each and output of 64 bits
- In-house C reference models
 - Coded independently and completely different from each other
 - Both coded without any attempt to be similar to the RTL impl

Galois Field Multiplier - Convergence

- 128-bit design converged easily within minutes
- 64-bit design did not converge even after running for 24h
 - Upon inspection, we found only a few common points between C (spec) and RTL (impl)
- Avoided recoding the C model to avoid introducing bugs to a known good C model (or needing to verify changes)
- Focused on achieving convergence through standard techniques
 - Case split
 - Different solvers for different proofs
- Proof times in the range of couple of minutes to 45 minutes

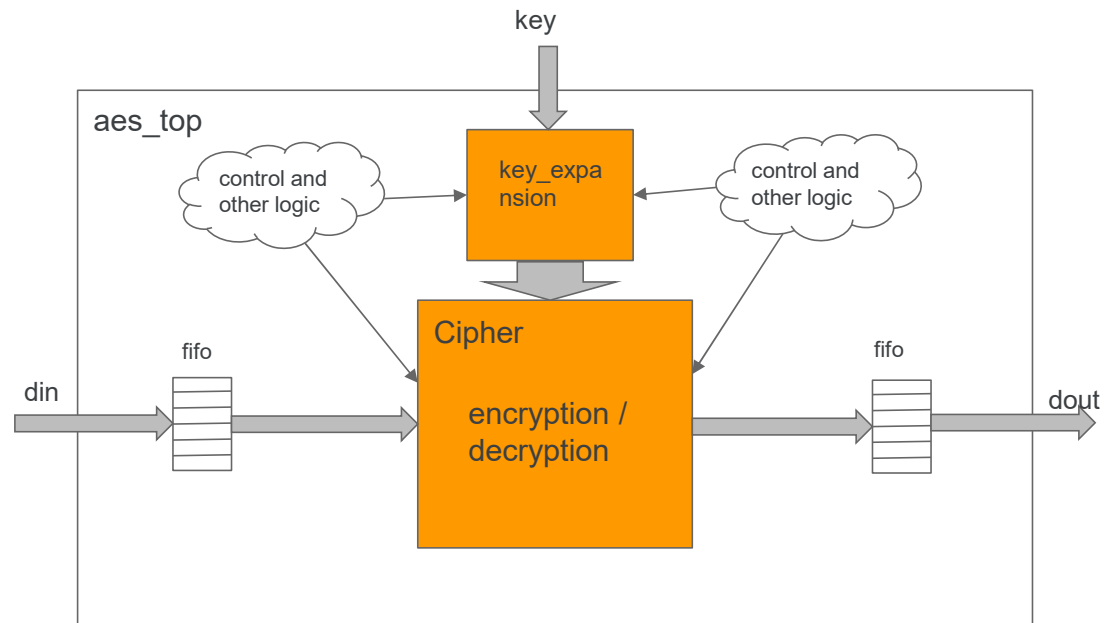
AES Basics



Key Size (bits)	Exp Key Size (bits)	No. of Rounds
128	1408	10
192	1664	12
256	1920	14

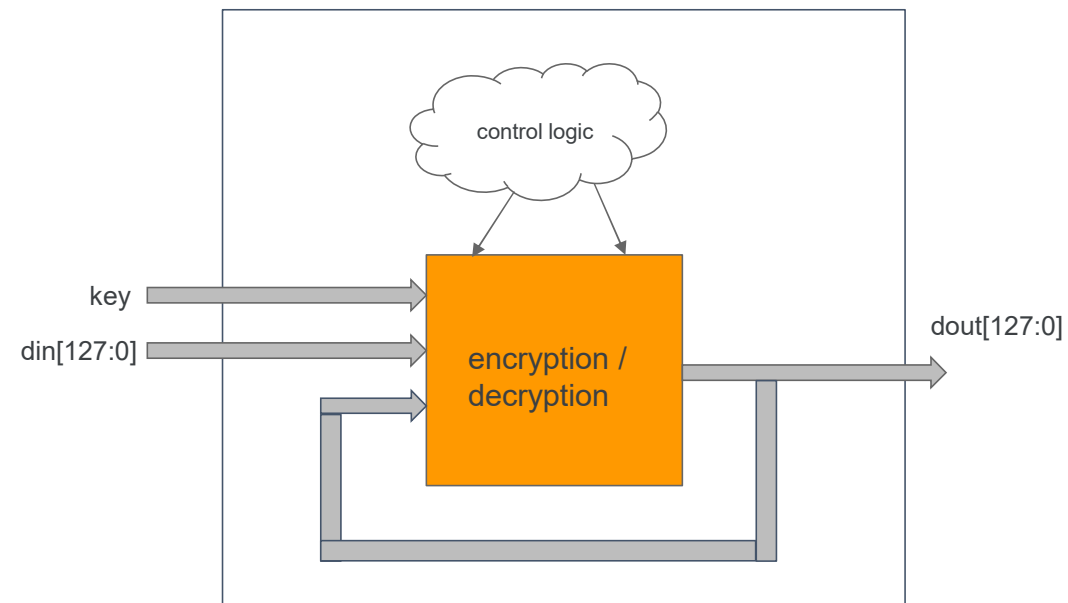
AES Block Implementation

- Not practical to verify the entire design end-to-end
- Verify the 2 datapath blocks standalone



AES Cipher Implementation

- Single stage implementation with data recycling
- End-to-end verification
 - Simple control logic that should lend well to DPV tool
 - Obfuscation techniques require end-to-end testing

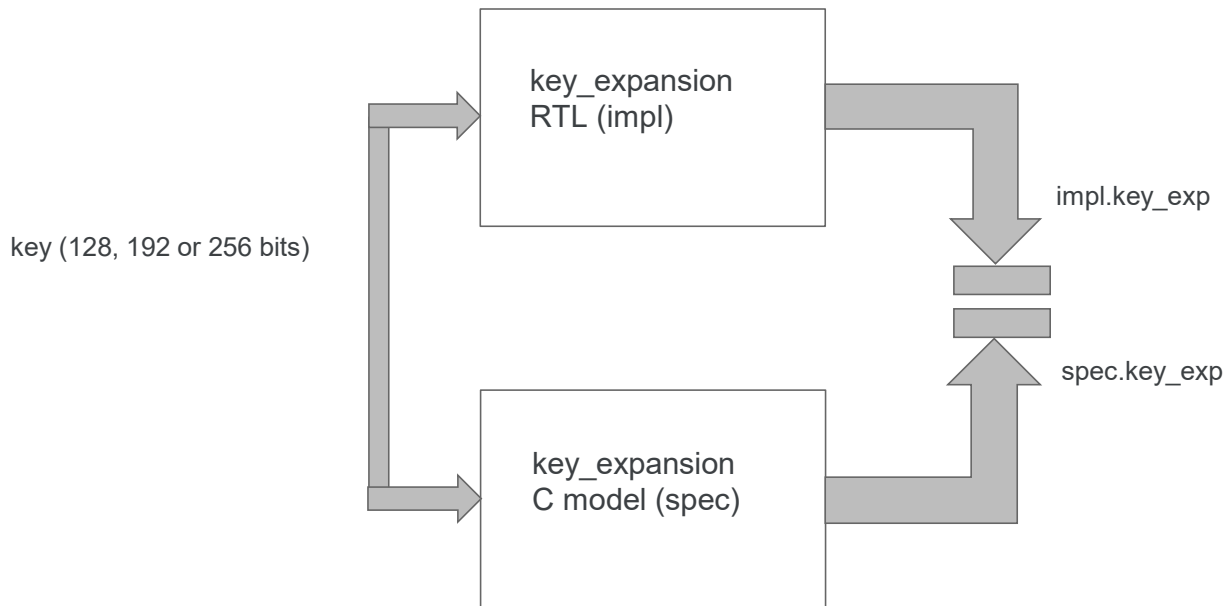


AES DPV - Divide and Conquer

- RTL expands the key for all rounds before the encryption/decryption begins
- Looked at several C models from openssl (open source software library)
 - Some did the key expansion at each round
 - Picked the model that was similar to the RTL: performed the key expansion for all rounds beforehand

AES Key Expansion C-to-RTL Equivalence

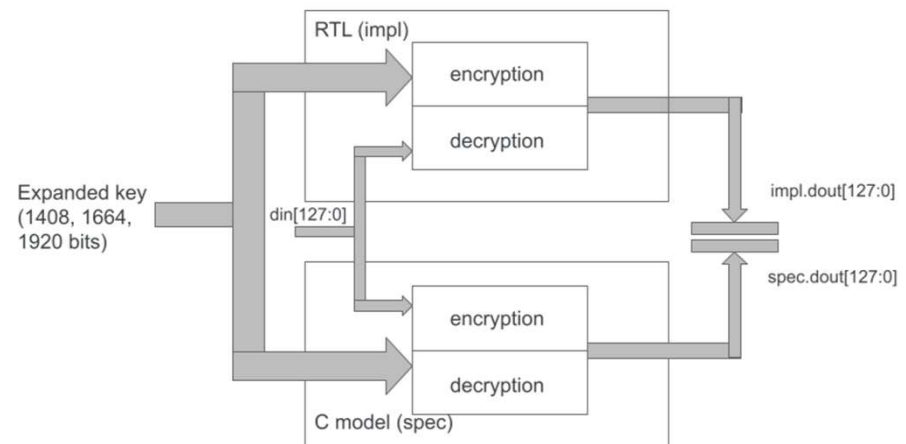
- Proofs converged easily



Cipher C-to-RTL Equivalence

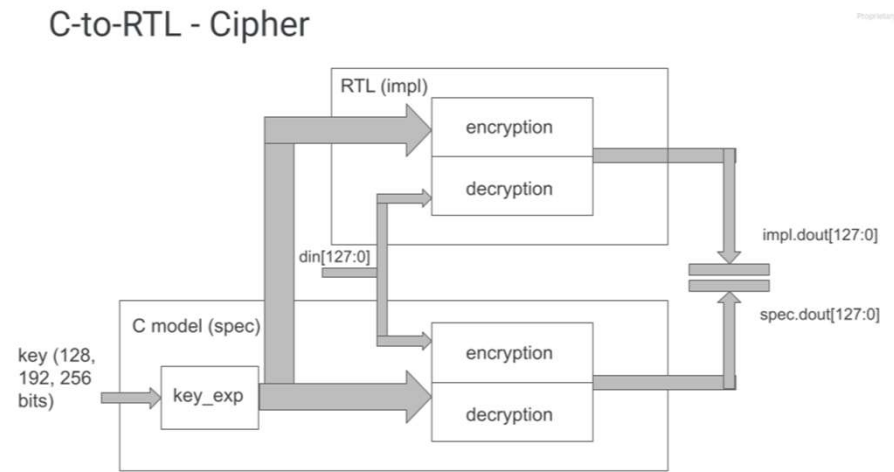
- Proofs did not converge even after running for 24 hours
 - Likely due to huge state space of the expanded_key (1408/1664/1920 bits)
- Exhaustive exploration of entire state space of expanded_key is not required, only a subset
 - 2^{128} of 2^{1408}
 - 2^{192} of 2^{1664}
 - 2^{256} of 2^{1920}

C-to-RTL - Cipher



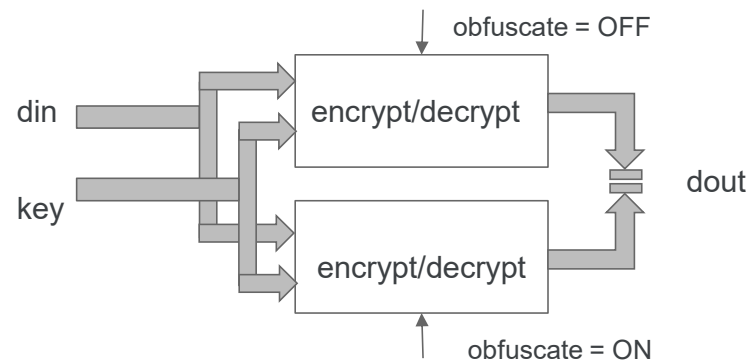
Cipher C-to-RTL - Convergence Strategy

- Include the C model for key expansion
- Feed output of key_expansion C model to both C model and RTL of Cipher
- Significant reduction of state space
- Proofs converged
 - shortest time of 1h for 128-bit encryption
 - longest time of 7h for 256-bit decryption



Obfuscation for Countermeasures

- A hacker can use side channel analysis to extract secrets such as key
- RTL provides a way to obfuscate the internal data as a countermeasure to side channel attacks
- Final output of encryption or decryption with obfuscation should not change



Cipher with obfuscation

- Verification intent: Obfuscation should not corrupt encrypted or decrypted output
- C-to-RTL equivalence
 - C model has no concept of obfuscation
 - Enabled obfuscation on RTL implementation
 - C-to-RTL did not converge even when run for 24 hours
- RTL-to-RTL equivalence
 - More common points between spec and impl; hence greater chances of convergence
 - Spec is RTL with obfuscation disabled; Impl is RTL with obfuscation enabled

AES with Obfuscation RTL-to-RTL Equivalence

- RTL-to-RTL equivalence results
 - Encryption (for all sizes of key) converged in 1 hour or less
 - Decryption did not converge (for any size of key) even after running for several days with standard solvers
- Tried various solutions to converge decryption but no success
- Synopsys provided a customized solver
 - Proof converged for decryption with 256-bit key in 12 hours

Results

- DPV found corner case bugs which would have been hard to find with simulation. Some example bugs found in FP32 multiplier:
 - Bug in normalization and rounding logic
 - Bug in logic that handles the special cases of Denormal
- Full proofs on Galois field multiplier and AES gave us higher confidence
 - Help meet higher industry standards

Conclusion

- Formal on datapath blocks enabled us to deliver high design quality and more efficiently than simulation
- Achieved proof convergence through
 - Standard techniques such as case split, assume-guarantee
 - Careful choice of reference model such that more common points between spec and impl
 - Deployed RTL-to-RTL equivalence where applicable
- Custom solvers for a specific datapath can yield faster convergence

Q&A

2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

How Formal is enabling automotive SoCs to
comply to various metrics of Safety & Security

Vaibhav , Sr Manager, NXP Austin

Gautam, Sr Manager, NXP Noida

Gaurav, Sr Manager, NXP Noida



Table of contents :

- Safety : Motivation for Safety solution
- FuSa basics/Problem statement
- Fault injection plan
- Formal and fault injection tool usage

- Security : SOC with distributed architecture
- Problem Statement for securtiy
- Challenges : Formal & Simulation
- Solution and Implementation

2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

PROBLEM STATEMENT



MOTIVATION

Design Process:

- Functional Safety is a fundamental requirement in the automotive systems to guarantee a tolerable level of risks in accordance with ISO 26262
- Based on FMEA/FMEDA process, IPs should define, implement and comply to the relevant ASIL level safety mechanism
 - This includes the Fault Campaign: identifying fault injection and diagnostic points, and converging on diagnostic coverage metrics

Challenge:

- Verify safety mechanisms comply with the ASIL-D metrics
 - Executing on fault campaign
 - Fault injection via normal simulation method could be too cumbersome and time consuming
 - Closing diagnostic coverage on last few percentage points could be overwhelmingly time consuming, and manual effort is prone to errors

Approach/Value:

- Presenting an optimized flow by combining fault simulation and formal analysis with Z01X and VC Formal, which was used on a memory controller IP
- Faster overall flow and reduction in manual effort for fault analysis

MAIN IDEA

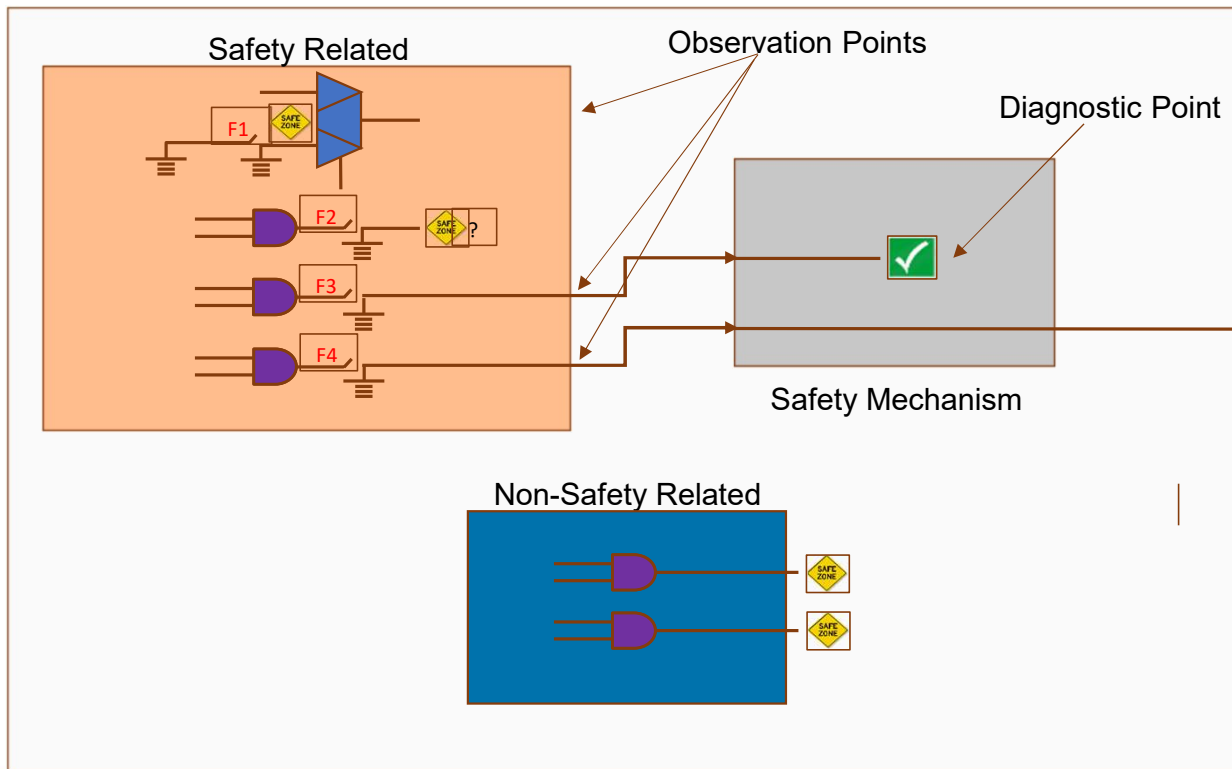
Shift Left:

- Partitioning the design appropriately and excluding standardized safety elements (ECC, lockstep modules, etc.)
- Leveraging the tools' strengths to maximize the ROI
 - Ordering the formal analysis and fault injection tools such that they can use each other's capabilities to the maximum advantage
- User automation makes the flow more efficient
 - Making it more repeatable & portable for several input vectors and across IPs

Tool capabilities:

- Z01X gives the base of fault simulation
 - Testability analysis: grade and reorder tests to provide higher coverage sooner
 - Concurrent fault simulation: simulate 1000's of faults in a single simulation
 - Compute farm job management: coordinate distributed jobs to amplify fault capacity
- VC Formal FuSa improves the flow by analyzing the design & the fault space
 - Structural analysis: reduce the fault space by finding safe faults
 - Formal analysis: further reduce fault space and provide formal proofs of detectability
- Unified Fault Database provides a common platform for the tools
 - Seamless interaction of Z01X and VC Formal FuSa, sharing fault definitions and results
 - Merge results and generate reports

FAULT CLASSIFICATION: DETERMINING THE EFFECT OF ANY FAULT



The devil is in the “details”...

- F1 – **Safe**
- F2 – Assumed **Dangerous**
- F3 – **Dangerous Detected**
- F4 – **Dangerous** Undetected

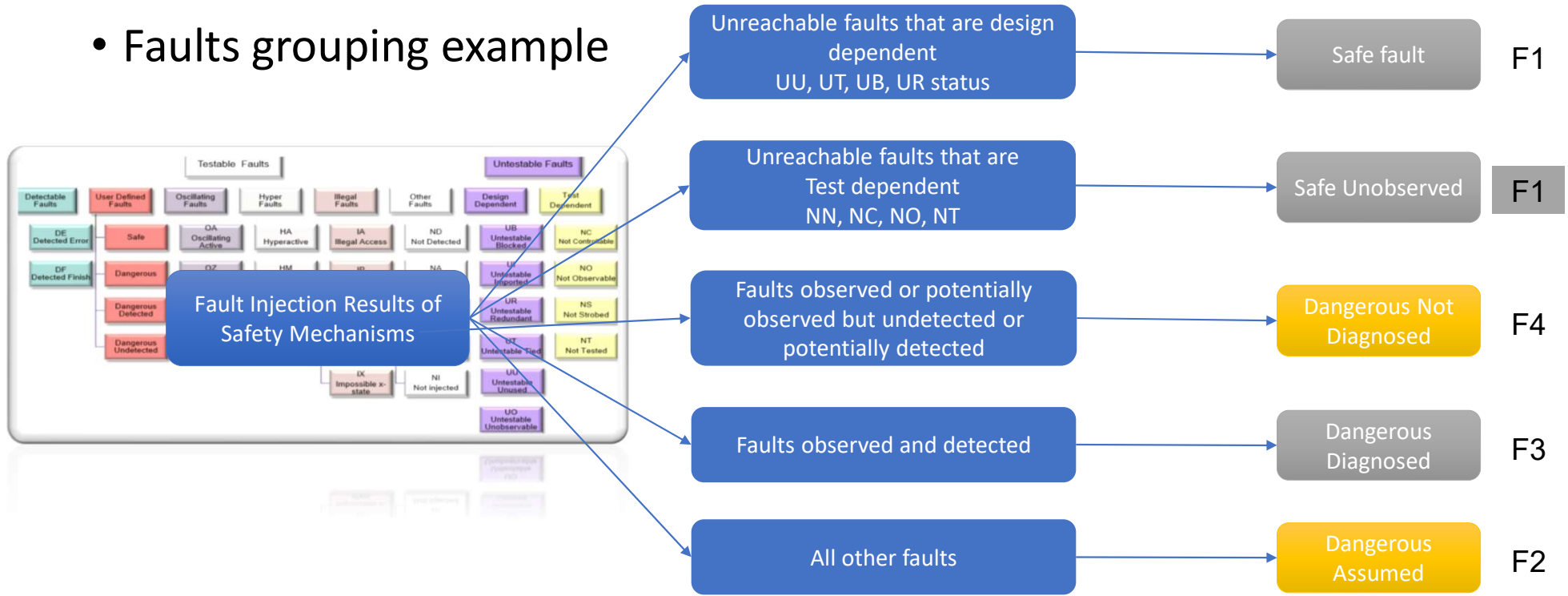
Based on user specification

If a fault was not observed and/or detected (F2), it can be:

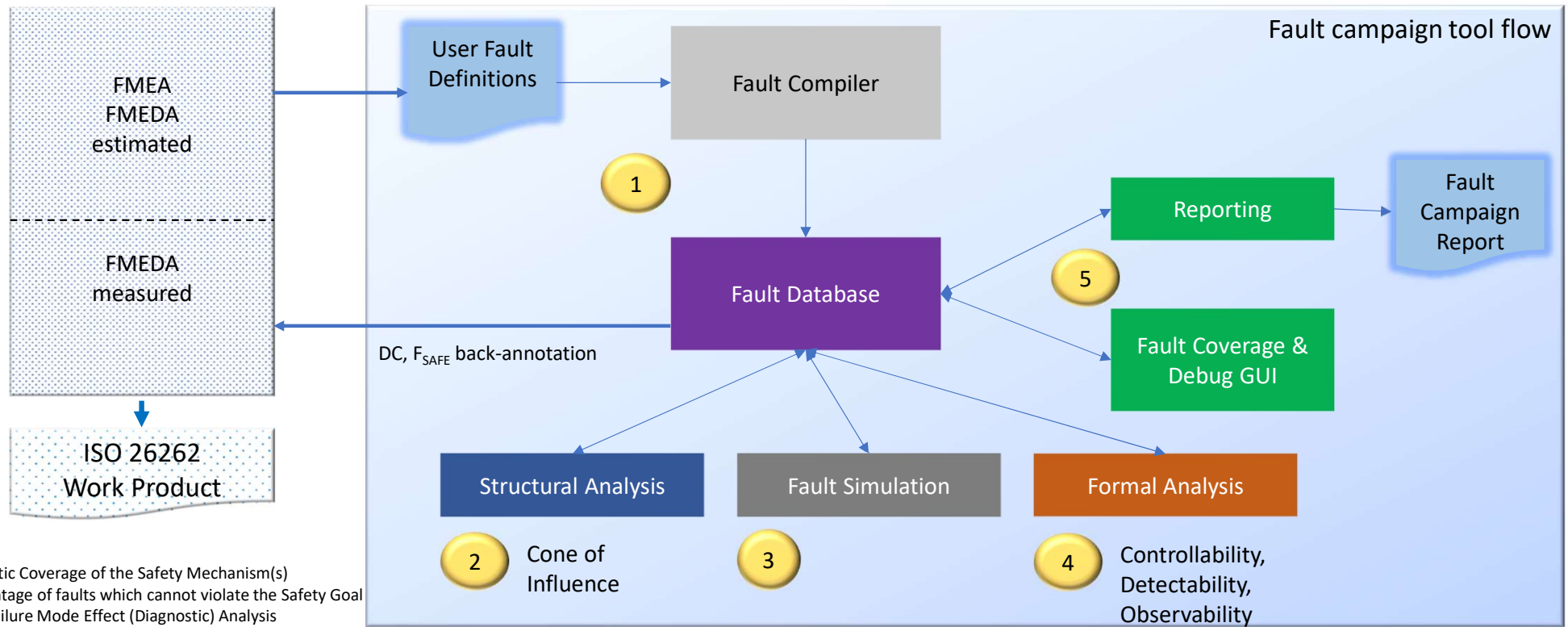
1. A safe fault
2. A dangerous fault which did not propagate due to insufficient stimulus / safety hole

STANDARD FAULT FORMAT FOR FUNCTIONAL SAFETY

- Faults grouping example



FAULT CAMPAIGN FLOW



DC – Diagnostic Coverage of the Safety Mechanism(s)
 F_{SAFE} – Percentage of faults which cannot violate the Safety Goal
 FME(D)A – Failure Mode Effect (Diagnostic) Analysis

FAULT INJECTION PLAN

- Faults :

All hierarchical ports and internal variables in the design will be considered as fault injection points

- **Observation point :**

All outputs to be considered as observation points

- **Fault monitors :**

Sitting outside IP : faults will be monitored by central fault monitoring unit

Various safety mechanism supported in IP

- Exclusions :

All clocks are assumed monitored

Lockstep have no shared physical redundancy thus independent fault injection could be done

- Faults to be covered :

Plan to cover both stuck at & transient faults

Single Point Fault Injections are primary mode of fault injection

- *Only valid multi point fault injection is single stuck at & single transient fault*

2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

SOLUTION AND IMPLEMENTATION



START SAFETY ANALYSIS EARLY

Synopsys VC Formal Environment Ready

Synopsys Z01X Environment Ready

Tool step:

- FCC
- Z01X
- Analysis and Convergence
- VCF (Formal Structural Observability (COI))
- VCF (Formal Control + Observe + Detect)

Flow 1

Flow 2

Flow 3

Shift Left

Shift Left






2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

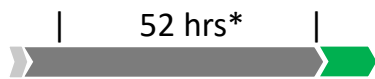
RESULTS



EVIDENCE (FLOW COMPARISON)

Tool step:

-  FCC
-  Z01X
-  Analysis and Convergence
-  VCF (Formal Structural Observability (COI))
-  VCF (Formal Control + Observe + Detect)



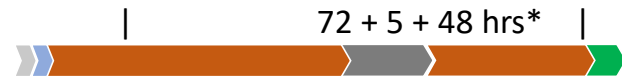
Flow 1

Faults (NA)	SA	DM	DU	DA	DN	DI
0	5730	154877	132997	2054	21402	12940



Flow 2

Faults (NA)	SA	DM	DU	DA	DN	DI
0	166078	78651	51746	630	19172	13549



Flow 3

Faults (NA)	SA	DM	DU	DA	DN	DI
0	167050	75000	50745	326	21784	15095

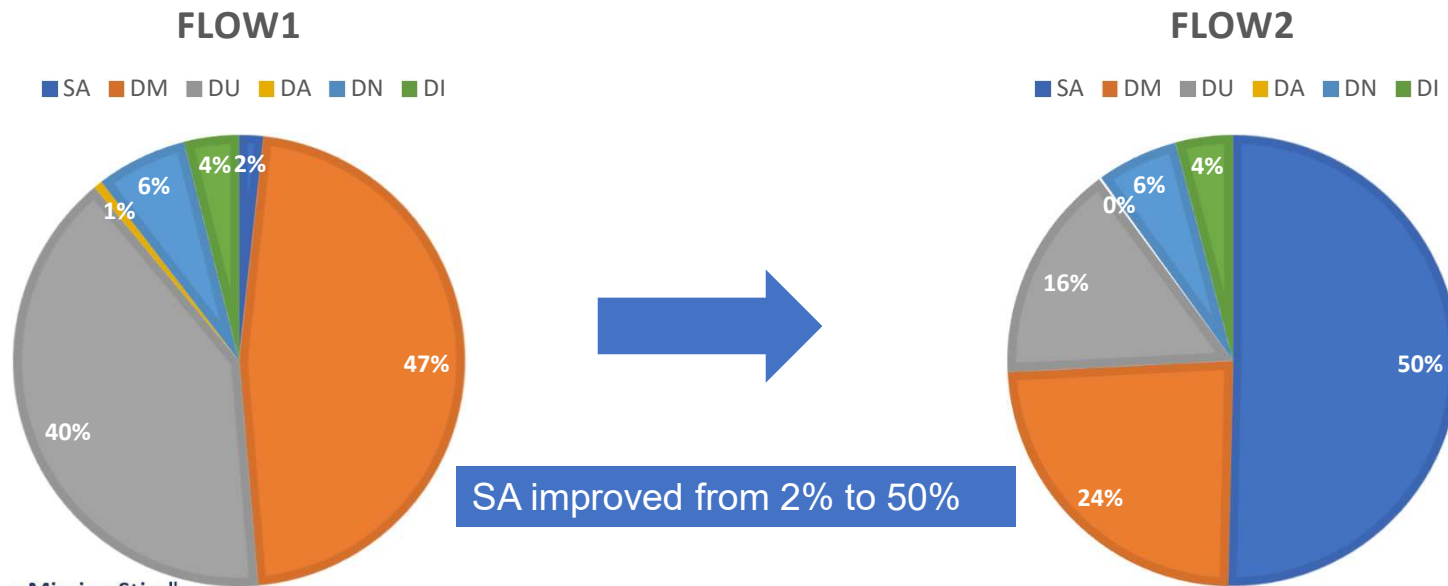
Faults status at end of each flow

Fault Status Groups

- SA "Safe"
- DM "Dangerous Missing Stim"
- DU "Dangerous Unobserved"
- DA "Dangerous Assumed"
- DN "Dangerous Not Diagnosed"
- DI "Dangerous Diagnosed"

*Times measured for Synopsys Z01X and Formal Analysis only

FLOW COMPARISON : FAULT SCOPING – SYNOPSIS Z01X (FLOW1) VS SYNOPSIS VC FORMAL + Z01X (FLOW2)



Fault Status Groups

- SA "Safe"
- DM "Dangerous Missing Stim"
- DU "Dangerous Unobserved"
- DA "Dangerous Assumed"
- DN "Dangerous Not Diagnosed"
- DI "Dangerous Diagnosed"

SA improved from 2% to 50%

- Reduction in overall fault simulation time (by more than 7X)
- Improved productivity by reducing manual debug and analysis efforts

2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

ADVANTAGES



ANALYSIS OF COVERAGE RESULTS – F_{SAFE} , DC

Coverage	Flow 2%
FS1	50.35
FS2	74.20
FS3	89.89
DC1	8.27
DC2	15.92
DC3	41.41

FS1/DC1 : current status

FS2/DC2 & FS3/DC3 : potential status after analyzing and reclassifying DM and DU

SA "Safe"
 DM "Dangerous Missing Stim"
 DU "Dangerous Unobserved"
 DA "Dangerous Assumed"
 DN "Dangerous Not Diagnosed"
 DI "Dangerous Diagnosed"

Further analysis on DM, DU, DA, DN fault status groups

- DM : faults for which propagation is structurally blocked for given stimulus
 - Solutions: apply formal convergence techniques, add stimulus for simulation, and/or document why they are safe
- DU : faults which started propagating, but did not make it to Observation or Detection points
 - Solutions: debug the faults, add stimulus, apply formal techniques, and/or document why they are safe
- DA : simulation issues
 - Rerun Synopsys Z01X with revised settings
- DN : faults which are dangerous but not diagnosed
 - May need to revisit the Safety Mechanisms

```
Coverage {
  FS1 = "( SA          ) / ( NA + SA + DM + DU + DA + DN + DI )";
  FS2 = "( SA + DM      ) / ( NA + SA + DM + DU + DA + DN + DI )";
  FS3 = "( SA + DM + DU ) / ( NA + SA + DM + DU + DA + DN + DI )";
  DC1 = "( DI          ) / ( NA +      DM + DU + DA + DN + DI )";
  DC2 = "( DI          ) / ( NA +      DU + DA + DN + DI )";
  DC3 = "( DI + DA      ) / ( NA +      DA + DN + DI )";
}
```

DC – Diagnostic Coverage of the Safety Mechanism(s)
 F_{SAFE} – Percentage of faults which cannot violate the Safety Goal

SUMMARY (SHIFT LEFT)

- Fault simulation, formal analysis and common fault database are all critical pieces of ASIL fault campaign
- Reducing fault injection scope by efficient use of Synopsys VC Formal FuSa with Z01X leads to significant performance enhancement and thus reducing manhour effort.
- Synopsys VC Formal FuSa impact on functional safety fault campaign
 - Quick setup and high ROI
 - COI structural analysis reduced fault space by 50%, resulting in
 - Reduction in overall fault simulation time (by more than 7X)
 - Improved productivity by reducing manual debug and analysis efforts
 - Formal analysis : proofs for detection without simulation in some cases

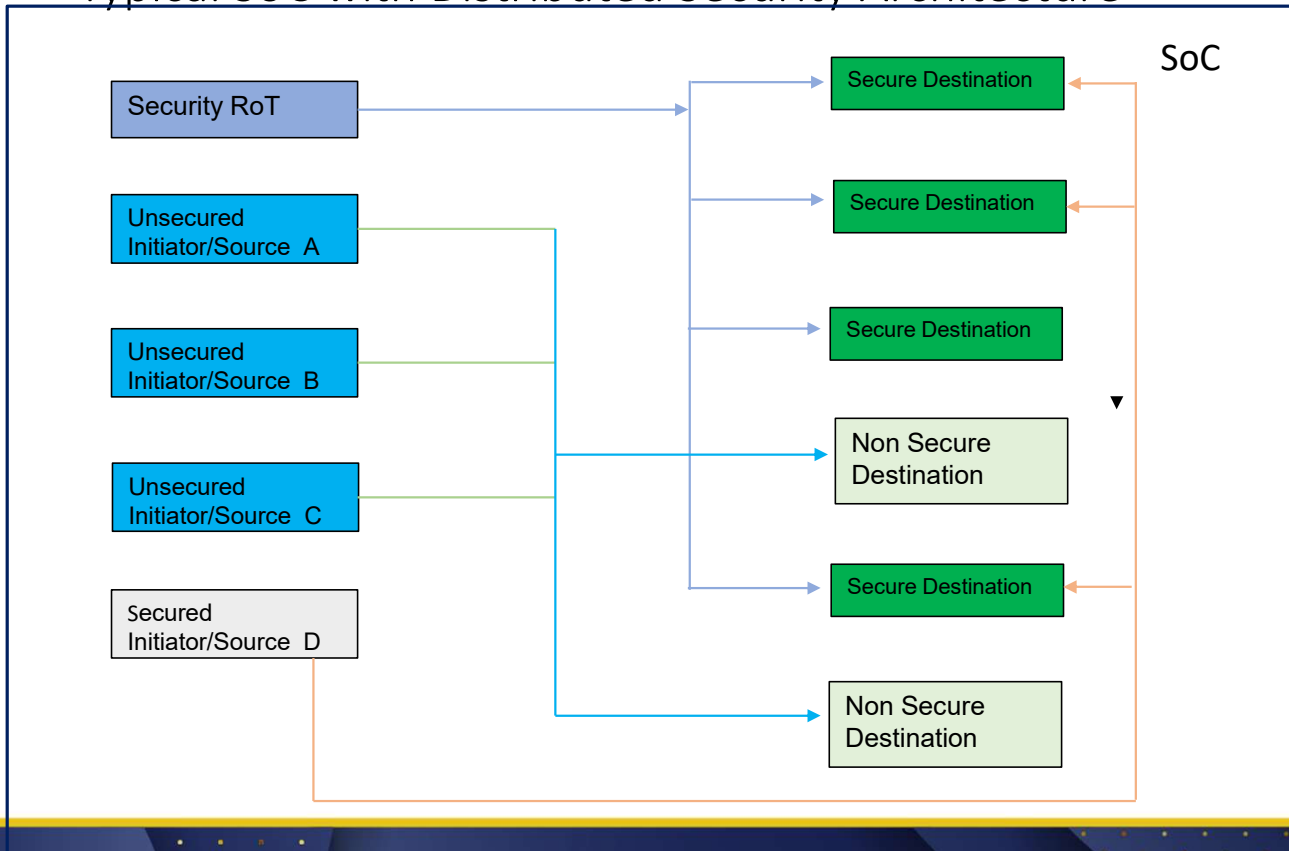
Using both tools in ways suggested in presentation leads to converge efficiently on our design for safety compliance which was further augmented by flow automation making it repeatable & portable

2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

SECURITY : SOC WITH DISTRIBUTED ARCHITECTURE

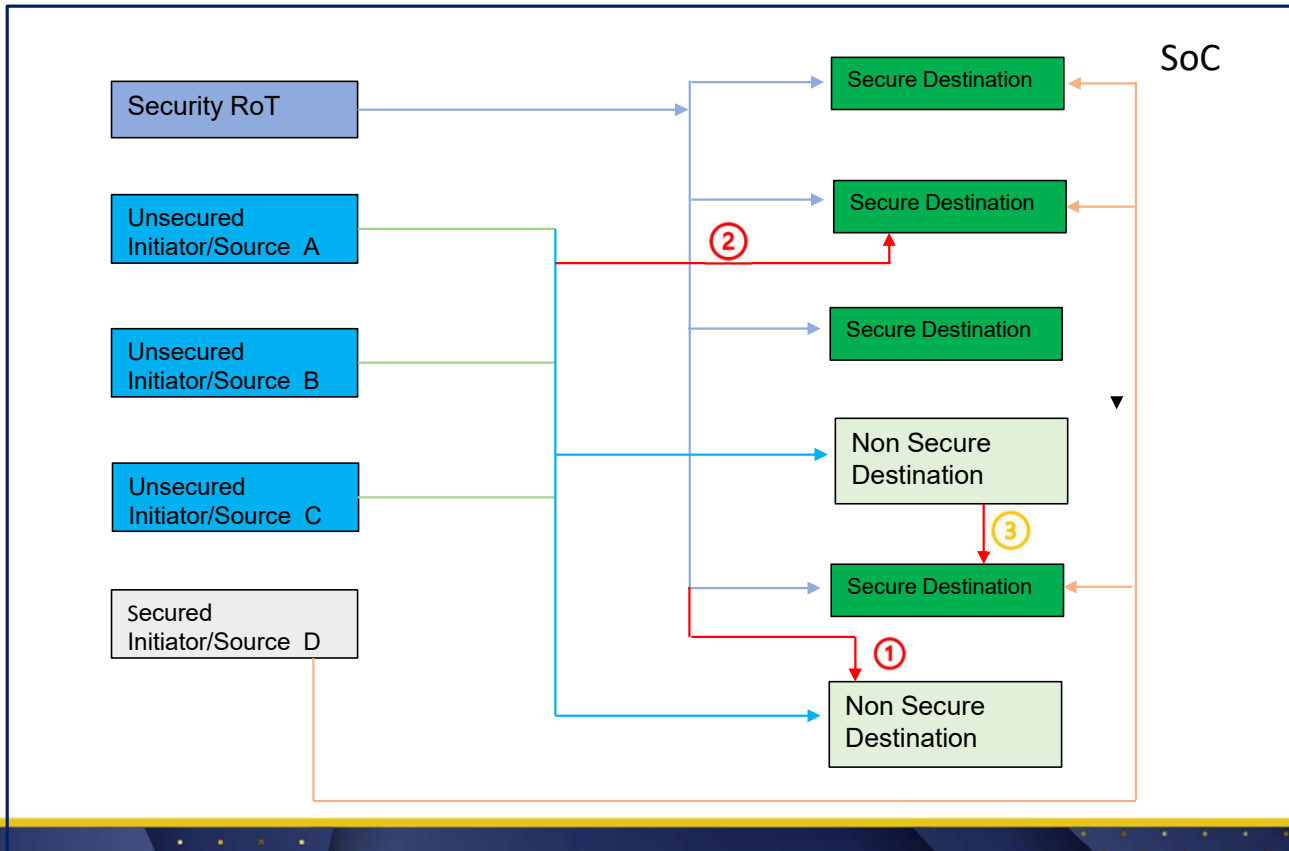


Typical SoC with Distributed Security Architecture



- Security RoT (Root of Trust) – Owner of Keys
- Secure Destination – Security relevant block, consumer of RoT keys
- Secure Initiator/Source – Security block which can read/write security critical information from Secure Destinations.
- Unsecured Initiator – A functional block which is not expected to have access to secure info / destination
- Non-Secure Destination – A function blocks which requires no Security Info (e.g. RoT Keys) to perform its functionality.

Typical SoC with Distributed Security Architecture with potential leakages



- Potential Security Leakages can be because of :**
1. Security Key being distributed to a Non-Secure Destination - ①
 2. Non-Secured Initiator getting access to a Secure Destination - ②
 3. Secure Destination getting accessed via an INDIRECT path from a Non-Secure Destination ③
- Typical reasons of Security Leakages :**

1. Unintentional security hole created while implementing a non-security functionality of SoC (e.g., **test, debug, safety etc.**)
2. Remnant logic from re-use of previous design.
3. Bad design practice which can open up access to a Secure Block, unintentionally. e.g., Multiple hierarchies of muxes.
4. SoC Specific Logic which doesn't fall under any specific verification scope and gets missed all together in functional verification.

2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

PROBLEM STATEMENT



SoC Security Leakage Problem Statement

Problem :

How to deterministically identify the potential leakage of Security critical info in a SoC with distributed security architecture ?

Details :

Security leakage due to logical bugs are difficult to identify and need manual RTL analysis. The process can be iterative, inconclusive and prone to errors. No metric is yet available which can give confidence on the sign-off of such security leakages.

EDA tool solutions are still in nascent stage to solve this problem at SoC Level.

A security leakage can lead to :

- Unintentional leakage of Secure info (e.g., Root Keys) to an unsecure logic problem.
- Protection override of security asset by incorrect un-secure source/signal.

2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

CHALLENGES : FORMAL & SIMULATION



Current Challenges with respect to Security Verification:

With Simulation :

- **Negative verification** is an important aspect of Security Verif Signoff. This is typically an **open-ended requirement** for SoC Verif Engineer, to develop all **negative scenario tests**.
- Dynamic testcases may **expose only limited security leakage** on SOC, based on a Verif Engineer's understanding of Design

With Formal:

- **Setting up the accurate SOC design** on Formal tool is a **known challenge (due to convergence)**.
- ***Constraining SoC Design to identify negative scenarios beyond functional specs is a challenge.***
- **No metric** to know the required logical cone for identifying negative security scenarios on SoC (a.k.a identification of '**ghost logic**' which can impact /compromise the security).
- Detailed design knowledge **required for a new user** to differentiate between secure paths and non secure paths.

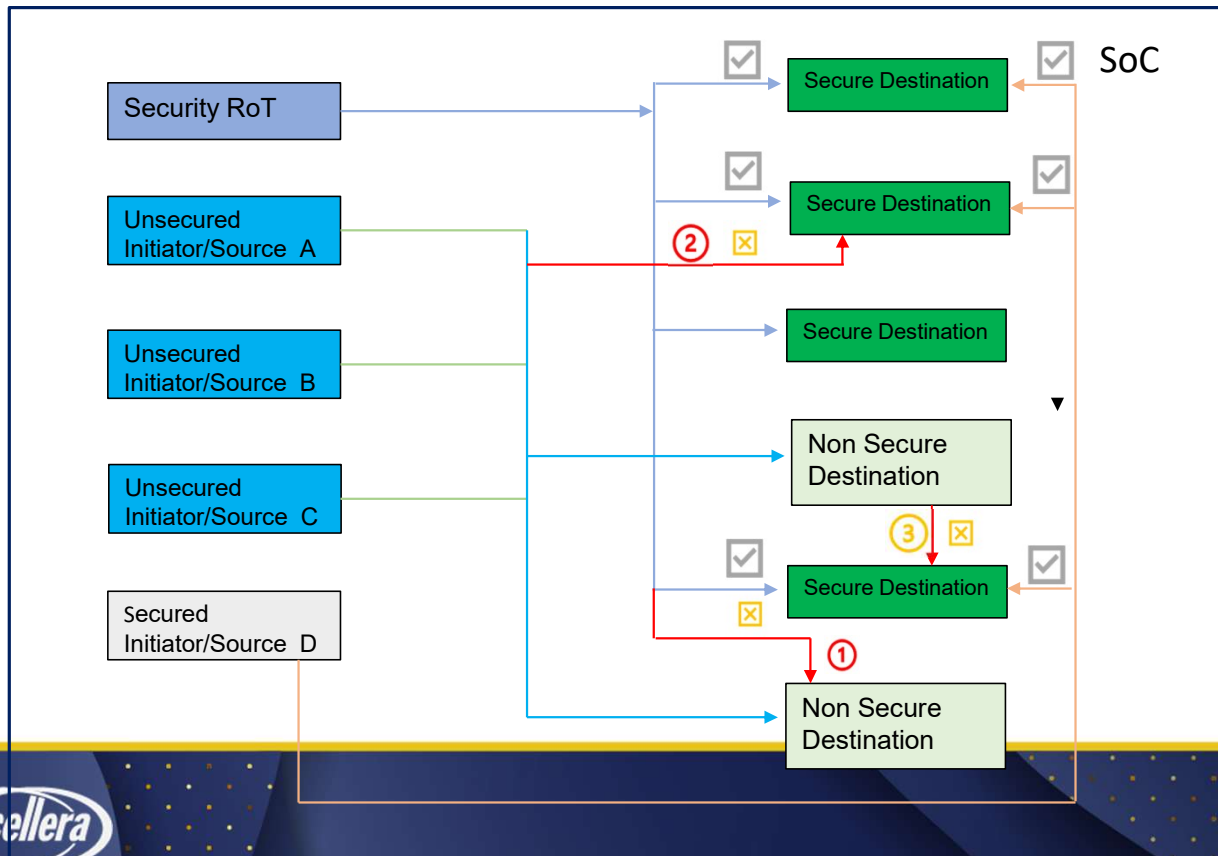
2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

SOLUTION AND IMPLEMENTATION



Solution evaluated using FSV :

- FSV is leveraged to develop a **PoC SoC Security Verif Flow** to identify a sub-set of security leakage causes commonly seen in SoCs, having distributed security architecture:

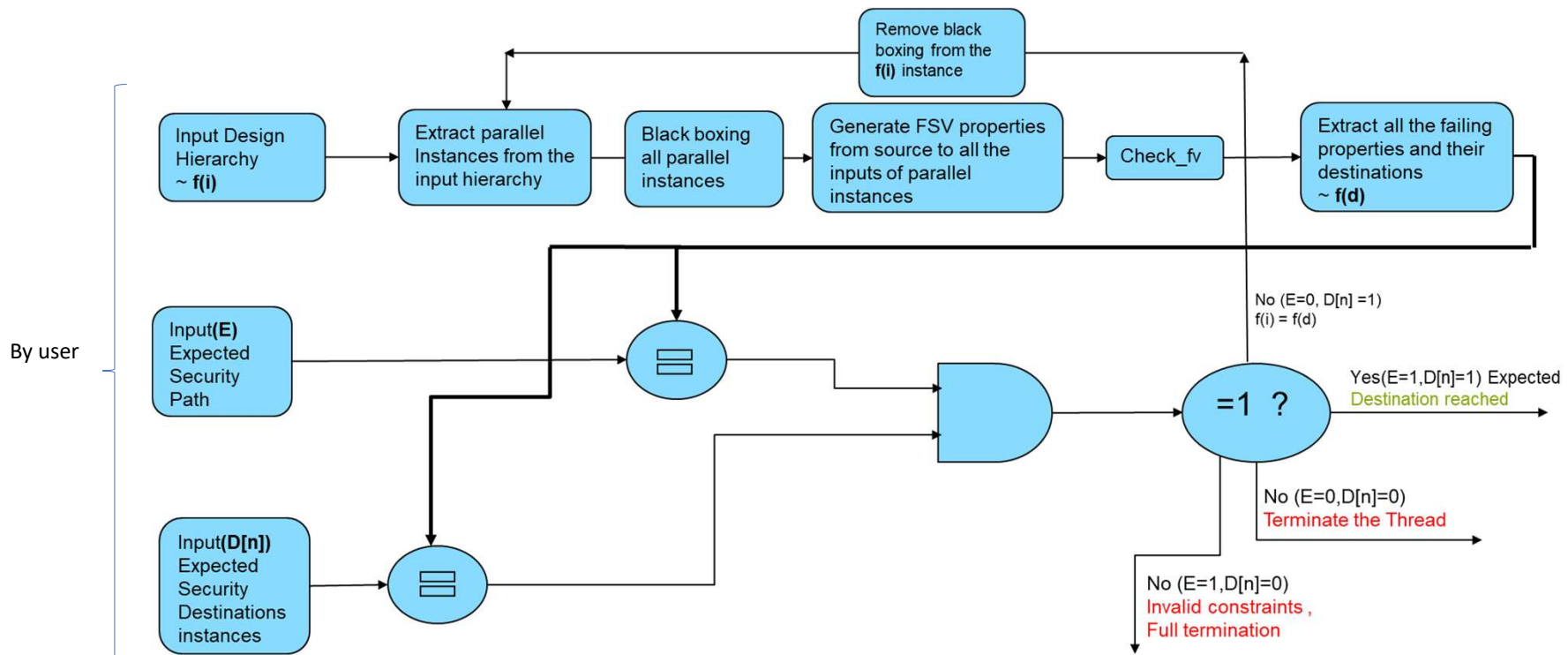


Overview of Flow :

- Identify the set of Secure Initiators
- Identify the set of Secure Destinations
- Identify the security critical signals
- Set the hierarchy to restrict the tool's analysis of the design.
- Run the tool to identify any potential leakages

Note: Custom scripts are used in the flow to pass the required info (mentioned in the above steps) to the tool

Solution evaluated using FSV : (contd)



2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

RESULTS



Results:

- The SoC with 8 master and 14 slaves (approx. 50 million gates) was put through FSV set-up for identifying security leakages on specific signals.
- The design was able to converge FSV in 3 hours.
- 2 signals were identified to be security critical at Initiator , converging on 4 destinations.
- The tool was able to successfully capture the unintended destination, indicating the leakage.

Inputs by User

Runs from each iteration

Results of each iteration

Analysis on results from each iteration

2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

ADVANTAGES



Advantages :

- Verification Flow to identify any Logic in SoC which can result in unintended leakage of the security sensitive info.
- Identifying security vulnerabilities due to incorrect usage of security critical signals at SOC level.
- Reduced probability of verification miss, due to design being over and under constrained by leveraging guided user inputs & FSV black boxing.
- Faster and more elaborate than directed testcase.
- Reusable, can be leverage across multiple different security architectures.
- Easy to use by new user , even with limited design knowledge.

Total Runtime : 3-4hrs (typically depends upon the depth of design hierarchies and)

2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

GOING AHEAD



Going Ahead

- Plan is to **optimize** the number of inputs provided by user .
- CC app utilization should be identified to dump out the number of possible path between one source to destination.
 - **Paths can be fed as an expected path inputs to FSV .**
- Addition of defining the number of iterations by user for unintended leakage paths to help user in debugging.
- Analysis to improve the runtime.
- Temporal flow view-based design picturization of complete leakage path.

Thank You



2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Robust GPU signoff using Comprehensive Formal Verification

Qualcomm GPU
Formal Verification Team

Qualcomm



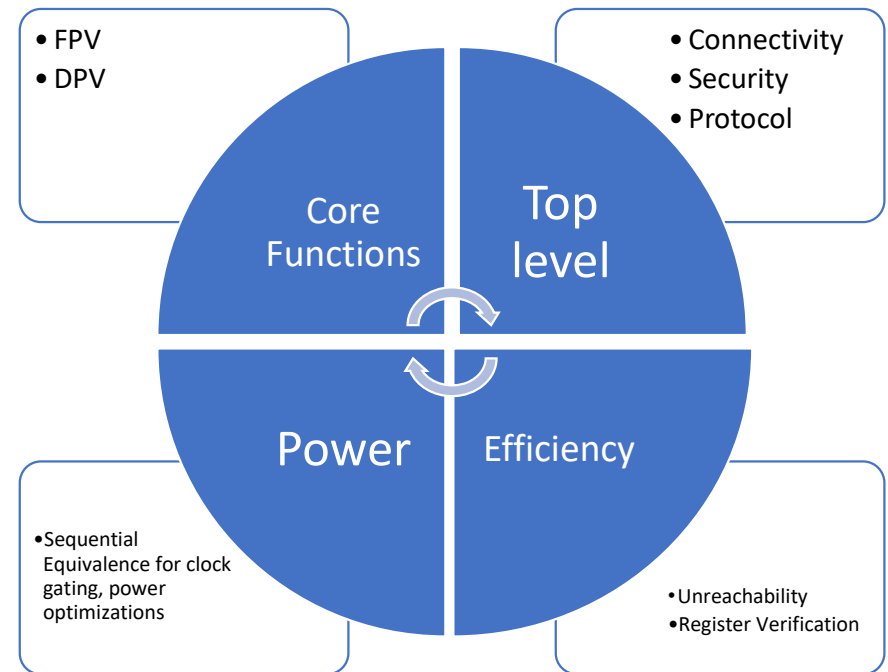
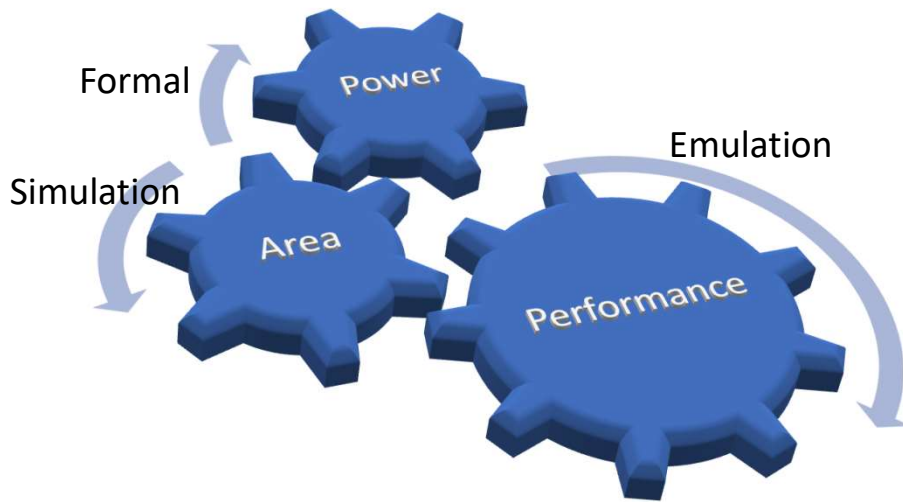
Agenda

- Introduction
- Defining the Success Criteria
- FPV for End to End formal sign off
- Functional and Performance Verification
- Example : Reusable ECC ABVIP
- Datapath Verification
- More App based Verification
- Example – Checker library for PPA optimization
- Case Study - SEQ Methodology for RTL Optimization

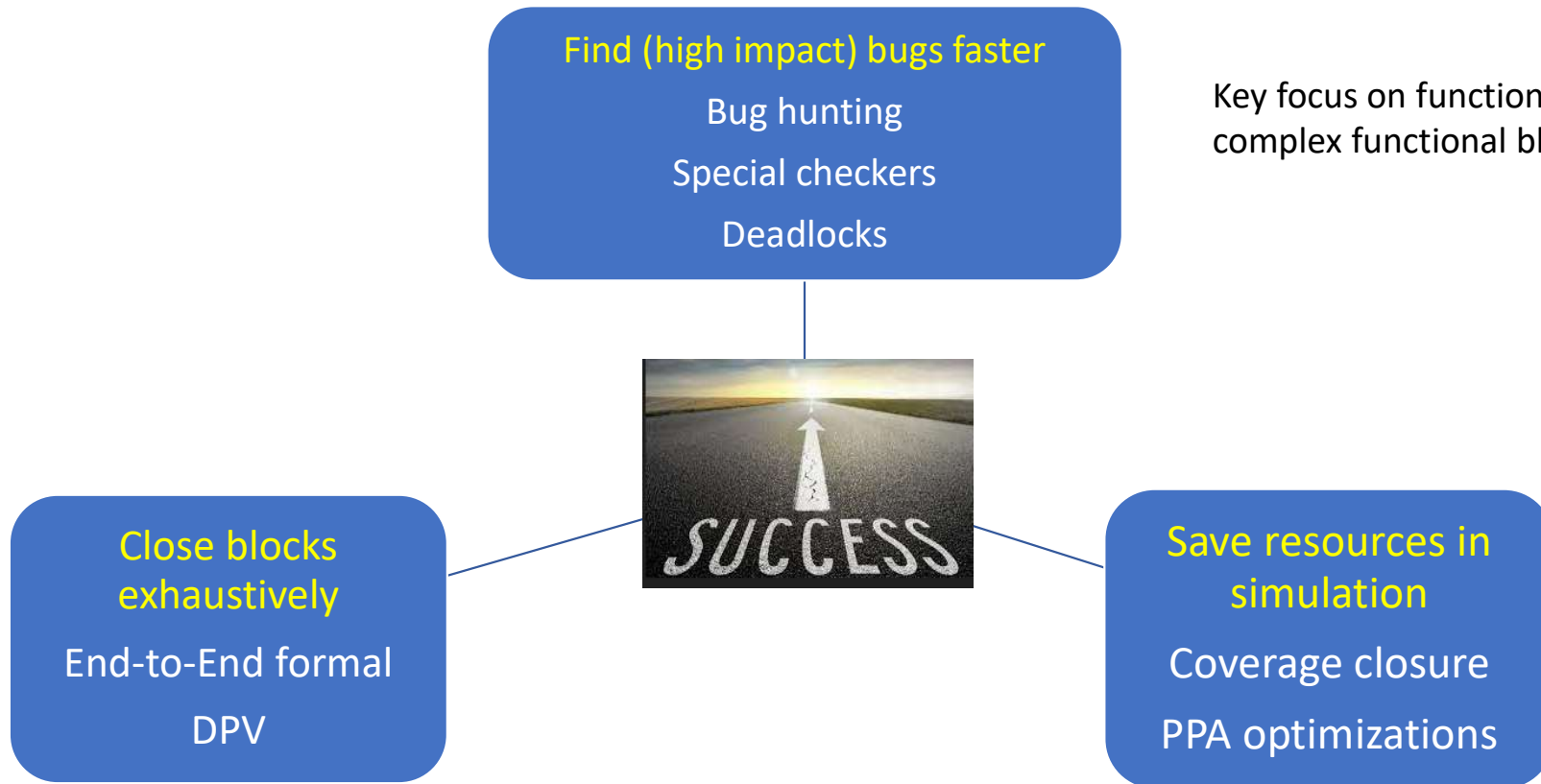
FV for GPU at Qualcomm

- Formal Verification is an integral part of Overall Verification Strategy for Qualcomm GPUs.
- Essential to make the Adreno GPU as the most power efficient mobile GPU (with world class peak performance)
- Design and Verification closure on aggressive PPA goals

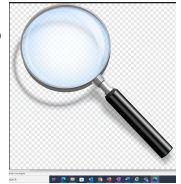
Complementing Simulation and Emulation



Defining the Success Criteria



Key focus on functionality of complex functional blocks



FPV for End-to-End signoff

End to End formal signoff

- Extensive use across cache sub-system verification (L0/L1/L2)

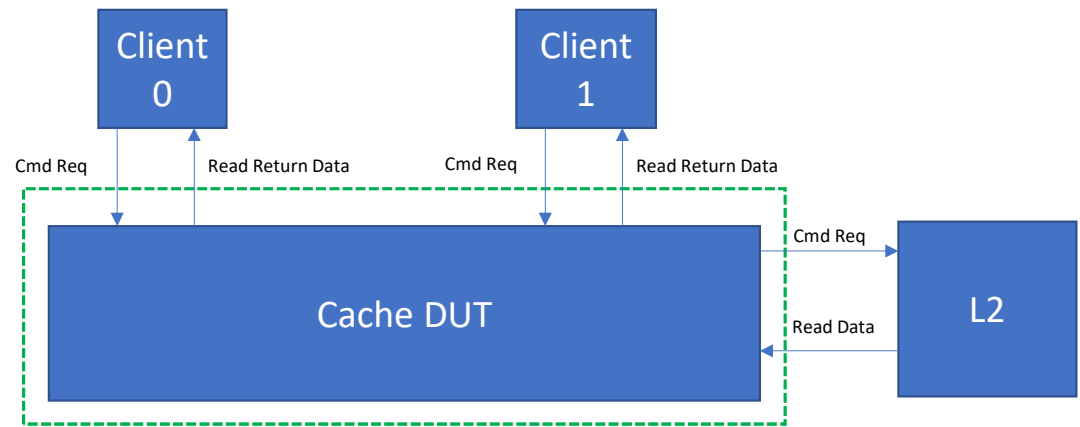
- Features:

<input type="checkbox"/>	Read and write paths, pre-fetch, mis-fetch, LOAD paths	<input type="checkbox"/>
<input type="checkbox"/>	Modes including buffer, cache modes	<input type="checkbox"/>
<input type="checkbox"/>	Read/write address overlap	<input type="checkbox"/>
<input type="checkbox"/>	Coverage across all cache lines	<input type="checkbox"/>
<input type="checkbox"/>	Buffers overlap	<input type="checkbox"/>
<input type="checkbox"/>	Forward progress	<input type="checkbox"/>
<input type="checkbox"/>	Deadlocks	<input type="checkbox"/>

- Sub-unit Testbenches, Unit level testbenches and Reusable properties

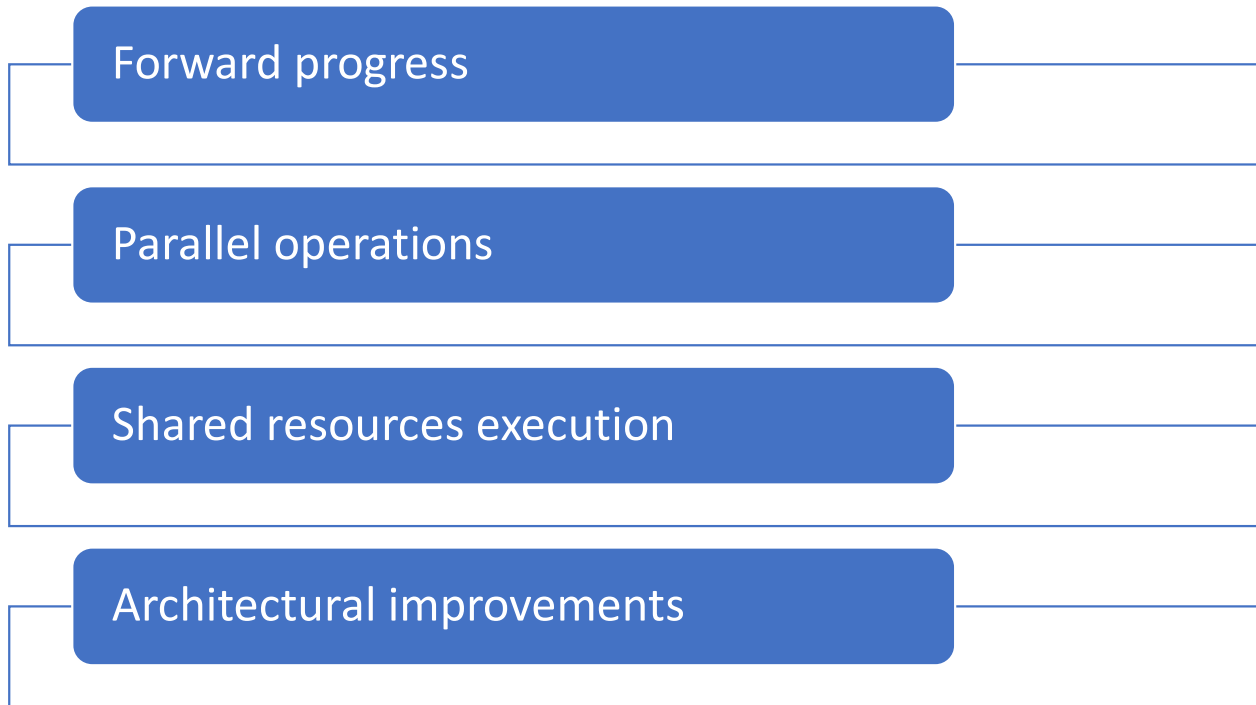
FPV for Performance Verification

- Forward progress checkers helpful in
 - Deadlock verification
 - Performance verification
 - **Observed behavior** : miss request on one client can halt another client if the corresponding Outstanding request fifo for first one is full.
 - **Reason** : In the Request fifo in the pipeline, the two clients are coupled together
 - **Resolution** : to check `osd_fifo_wfull` before client req is written in req



More FPV

- Concurrent graphics operation blocks with heavy control logic (Operations list)



2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

ECC ABVIP (Assertion IP)



Reusable IP deployment

ABVIPs

AXI/AHB/internal protocols

FIFOs

Arbiters

RAM i/f

ECC proofkit

Commonly used interfaces (valid/ready, srrdy/rrdy)

Custom blocks for arbitration/ traffic scheduling

- Forward progress
- deadlocks

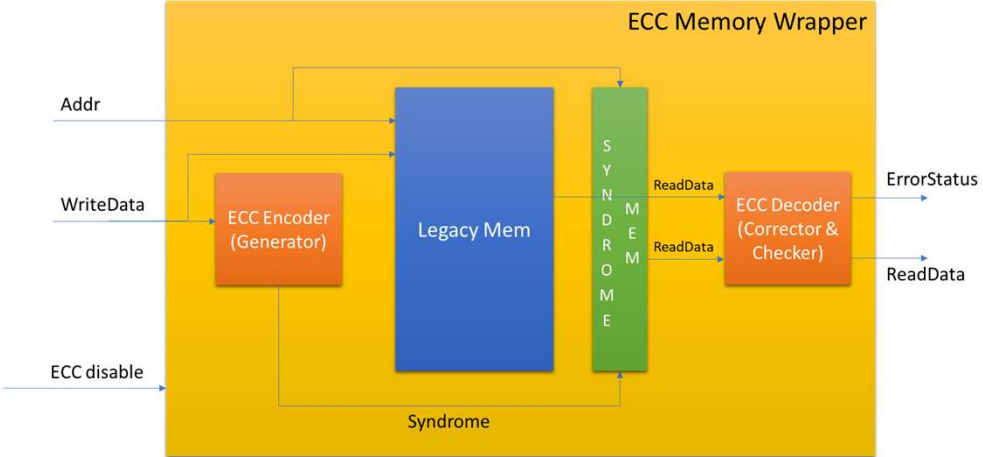
Example: ECC proofkit

ECC Feature overview

- Memory writes generate syndrome, reads compare stored syndrome
- Interrupts generated for SEC and DED

Scope of ECC

- **Large state space**
 - Depth x Width combinations
 - ByteEn supported or not
 - Single / Dual / Triple port
 - Bypass
- Verifying **data integrity**
- **Lack of coverage/confidence** could be a **showstopper for Automotive Safety**



ECC Formal Verification Proof-Kit

Small design (loading 1 ECC memory wrapper at a time)

- friendly to formal tool
- classic ***divide-and-conquer*** strategy employed in formal verification

Simple properties to describe and code

- Single-bit error should always be detected and corrected
- Double/Multi-bit error should always be detected
- In case of no error (injected between memory write and read)
 - Data should remain intact
 - Error shouldn't be reported as detected/corrected

Exhaustive coverage

- Covers complete state space, ensures compliance

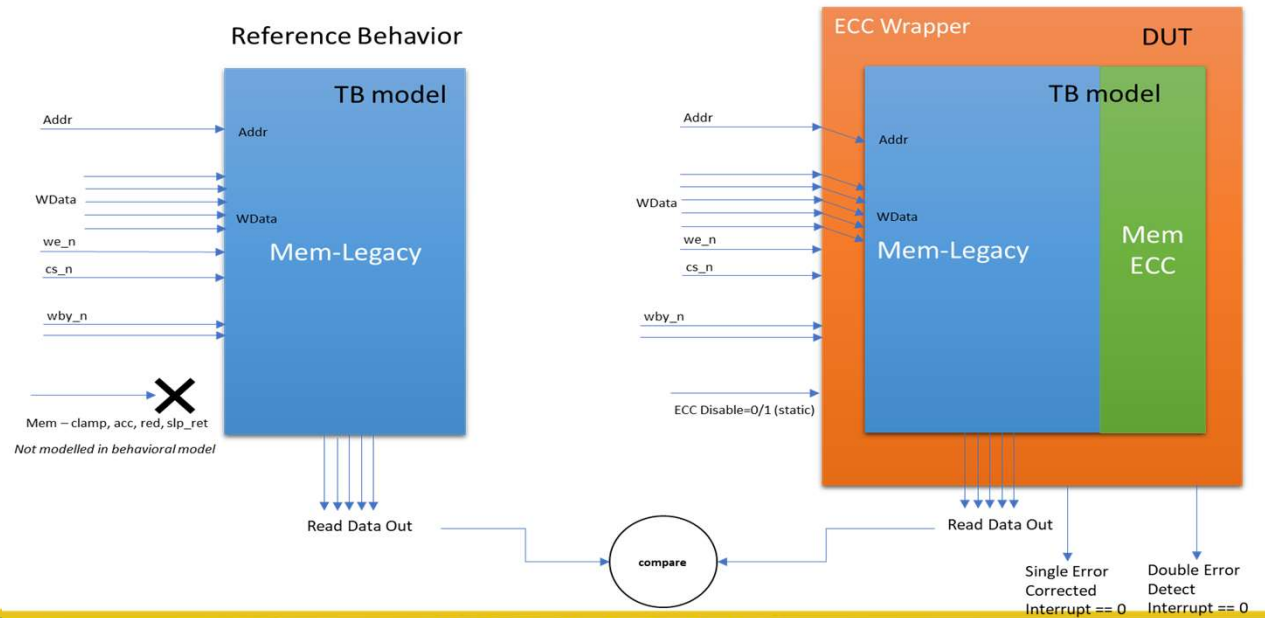
Memory model abstraction

- Lightweight memory model used for formal

Solution - PART I

Proof of Data Integrity (without error injection)

- **Sequential Logic Equivalence**
 - “simple memory model” vs. “memory model with ECC wrapper around”
 - **No error injected** between write and read
 - Prove that “**Read Data outputs are always same** between LHS and RHS”
 - Prove that “**Error is never reported** as detected/corrected”

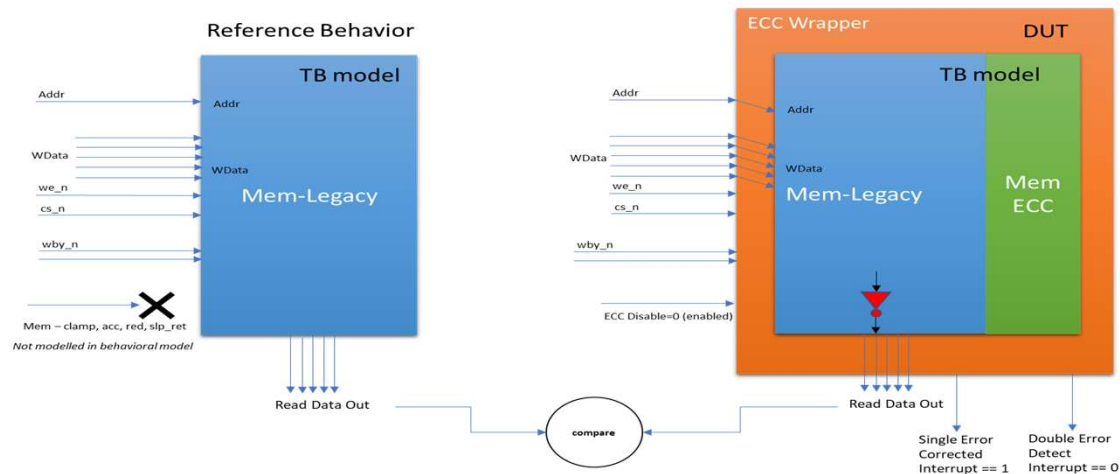


Solution – PART II

Proof of ECC functionality Correctness (with error injection)

- **Sequential Logic Equivalence**

- “simple memory model” vs. “memory model (with 1 or 2 bits flipped) with ECC wrapper around”
- Mimics **corruption** or **error injected** between write and read
 - 1 or 2 bits are inverted at memory Read Data out bus
 - either/both of main and syndrome memory
- Prove that “Read Data outputs are always same between LHS and RHS if only single bit flipped” – single bit error correction
- Prove that “Error is always reported – either as corrected/detected according to 1 or 2 errors injected”
- **Additional custom properties/assertions** on top of SEQ-mapped comparison of output signals



2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Datapath Verification



Datapath Verification

- Extensive use of Synopsys VC Formal DPV solution for exhaustive verification of

Arithmetic blocks in shader unit

- Adders, multipliers, special formats, operations

Data format converters

- IEEE 754 compliant
- Non-standard Floating point and fixed point
- Float2fix, fix2float

Compression/decompression engines logic

- Challenging advanced C++ constructs
- Rewrite of C++ code for DPV support compatibility

Pervasive use across GPU of arithmetic heavy blocks

More about DPV

20+ arithmetic blocks verified

Used from more than 8 years

Found 100s of RTL and Model bugs

Saved 2-4 man months of effort per project

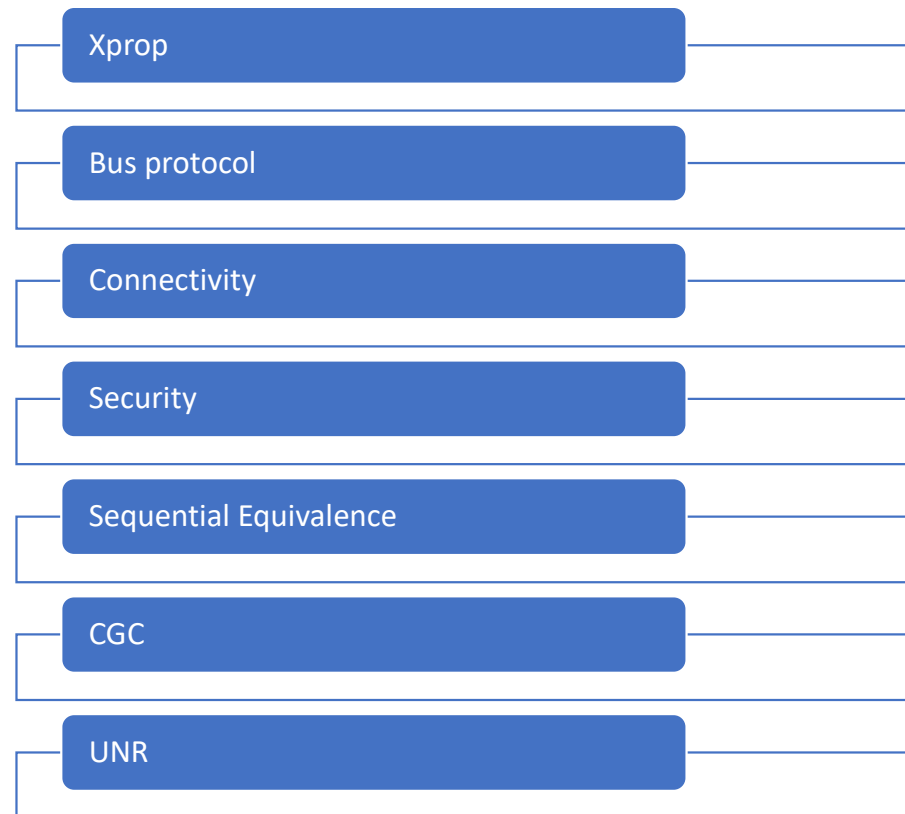
Completely replace Simulation TB for ALU, format converters

More complex blocks

- Compression engine
- Decompression engine

App based formal applications

- Miscellaneous use across GPU of app based formal:



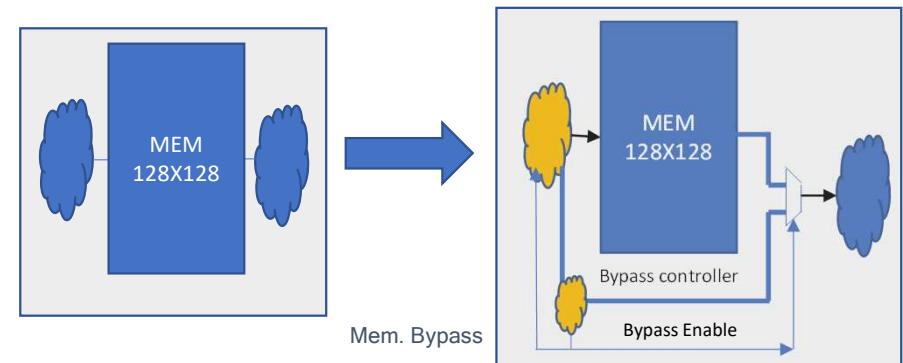
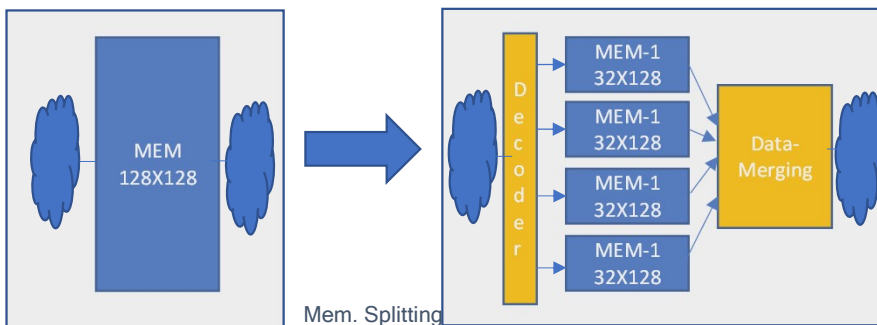
2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Checker Library for PPA optimization



Checker Library for Memory PPA optimizations

- Late design changes in RTL around Memories/Cache of Power, Performance and Area optimization (PPA) are challenging to verify.
- Requirement is to quickly check the new with minimum setup time
 - Need to reduce the re-verification time.
 - No dedicated mem controller entity , as the newly implemented logic is scattered around the design.
- Type of potential changes are like -
 - Memory splitting
 - Memory access removal by changing the control path
 - Memory access removal-based workload analysis



Checkers

- Checkers specific to each scenario can be developed guaranteeing correctness of each scenario
- In some cases, checkers can also verify the effectiveness of the transformations
 - Read access removal : Is there any scenario still exists where same address is enabling the memory ?
- Checkers can be bound to memory control related logic though defined interface
 - Can be used in both simulation and formal verification
 - Can be configured to allow flexibility in usage
- Property based verification by modelling assertion custom checkers around memories as Handshake, Spurious , Data Integrity and Forward progress checks

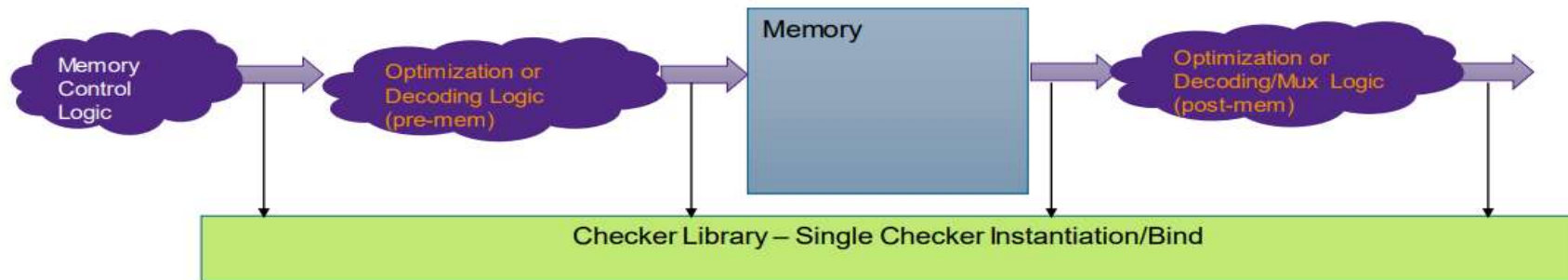
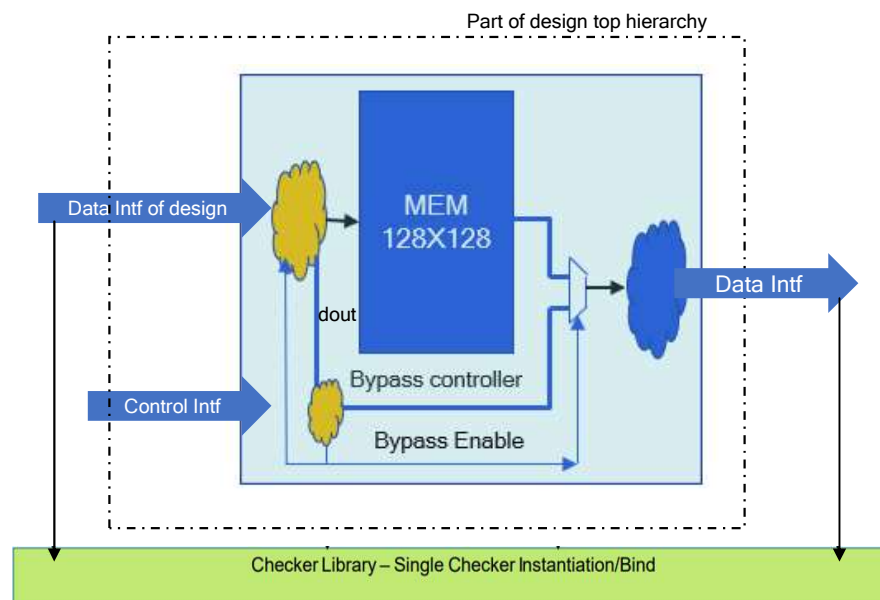


Illustration of implementation



- Design-under-test will be module from top design hierarchy which captured all transformation.
- Properties are configurable/parameterized through TB
- Abstraction for faster turnaround-
 - Symbolic address abstraction
 - FIFO full abstraction
 - Optional : Reset abstraction for WR/RD pointer if depth/size is bigger.

Technique of implementation (Abstractions)

- Symbolic Address Abstraction
 - Use of Symbolic Address
 - Replace DPETHXWIDTH memory with 1XWIDTH
- FIFO Full Abstraction
 - Cut point fifo full
 - Reset abstract WR and RD pointers
 - Add constraints on full

2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

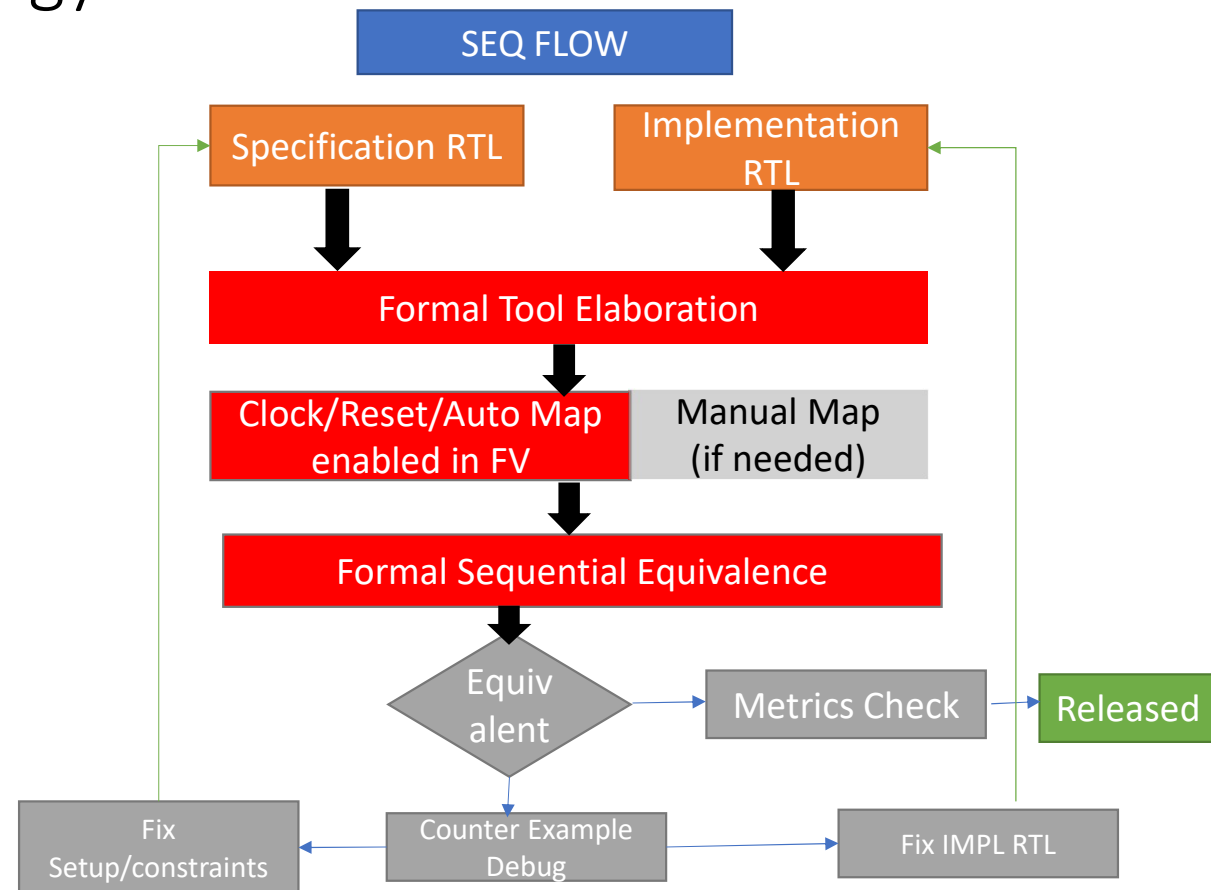
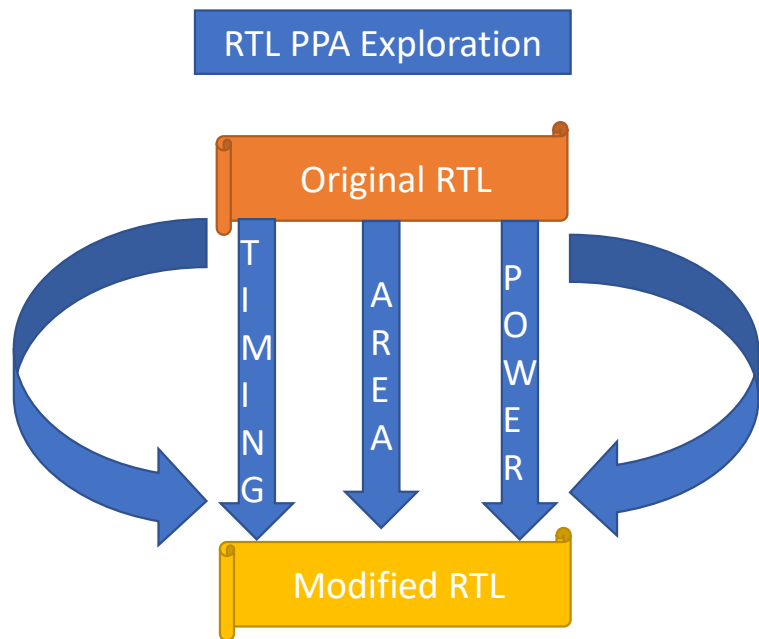
SEQ Methodology for RTL Optimization



Boosting RTL PPA optimization verification

- Design and Verification closure of aggressive area/timing and power goals is challenging due to increase in design-complexity and Time to Market constraints.
- To address PPA(Power-Performance-Area) RTL fixes are MUST , but hard to explore multiple “whatif” ideas at early design cycle due to DV dependency.
- Combinational Equivalence is not efficient as incremental design revisions include sequential(register) changes.
- Formal Sequential Logic Equivalence is the only way to validate temporal PPA changes for new/legacy blocks across design cycle to enable a “LEFT-SHIFT” in verification process.

SEQ Verification Methodology



SEQ Formal Methodology for RTL PPA exploration

- Multiple “What-if” PPA exploration easily verified in few hours by SEQ Working flow
- No need of multiple days of DV regression for multiple incremental RTL changes
- Exhaustive and ease-of-use methodology with complete coverage of the design
- PPA changes implemented in parallel to functional changes without additional regular DV impact

More to be done!

- More End to end FPV across new as well as legacy blocks
- Multiple projects, running in parallel need innovative solutions for efficient formal closure.
 - Ingenious equivalence solutions
 - Reduce simulation test count by proving various sub-configurations equivalent
- ABVIP developments for leaf cells, protocols, blocks
 - Promote wider reuse across formal and with simulation.
- SEQ + FPV blended flows for full closure

Q&A

