The SymPLe Project

Achieving Verifiable and High Integrity Embedded Digital Devices Through Complexity Awareness and Constrained Design

12th International Workshop on Application of

Field Programmable Gate Arrays in Nuclear Power Plants

October 14-16, 2019

Dr. Carl Elks, Rick Hite, Jason Moore (Mathworks) Athira Jayakumar, Smitha Gautham, Chris Deloglos, Andrew Nack (Paragon)and Dr. Ashraf Tantawy

Virginia Commonwealth University
Department of ECE
Richmond, Virginia, USA

Matt Gibson Lead PI,
Electric Power Research Institute, Charlotte, NC.

Sponsored by DOE Office of Nuclear Energy, Advanced Sensors and Instrumentation under the NEET-2 Program

The Team

- POC information: crelks@VCU.edu, mgibson@epri.com ,
- Matt Gibson, Program PI, EPRI - mgibson@epri.com
- Dr. Carl Elks, Co-PI, Assistant Professor of ECE, Virginia Commonwealth University – crelks@vcu.edu
- Dr. Ashraf Tantawy, Associate Research Professor, Virginia Commonwealth University
- Jason Moore, Consulting Services Mathworks
- Andrew Nack, Commercial Grade Dedication, Paragon Inc.
- VCU Students
  - Rick Hite, PhD Candidate - Lead Architect
  - Smitha Gautham, PhD Candidate - HW V&V and Formal Methods
  - Chris Deloglos, PhD Candidate - Architect
  - Athira Jayakumar MS Student - Model Based V&V and Testing
- Future collaborators –
  - Dr. Patrick Schaumont, Va Tech – HW cyber security
  - Dave Landol – OneSpin VLSI formal Verification

SymPLe Project: Overview

- The talk today is about two things: (1) A rapid overview of the SymPLe architecture concept, (2) and the formal model based design assurance activities with respect to IEC 61508.
- Phase 1 sponsored by EPRI and DOE Office of Nuclear Energy, Advanced Sensors and Instrumentation under the NEET-2 Program from 2015 to 2019.
  - The program objectives were to research effective methods to significantly reduce and mitigate Software Common Cause Faults (CCF) in digital I&C.
- Our approach to addressing SCCF was unusual – we avoided software !
- SymPLe is as much a *way of thinking* about designing critical systems as it is as about the SymPle architecture itself.
- The *way* you design tells a lot about *what* you design.

VCU College of Engineering

EPRI ELECTRIC POWER RESEARCH INSTITUTE

Critical Systems Thinking "paraphrased from J. Rushby, *Composing Safe Systems*"

- We build systems from components and platforms, but what makes something a *system* is that its architected properties and behaviors are distinct from those of its components.
  - We have become good at this, *most of the time*.
- For critical systems, we need nearly *<u>all of the time</u> behavior.*

- *As SW IP, SW languages, HW components become complex then we may inherit unwanted or undesirable component interactions, and not know it.*
- How do we compose complex systems and not inherit undesired emergent properties and interactions? This is a open and very active research question.
- SymPLe architecture project are steps in this direction.

Complexity Awareness for Verifiable Systems

- It is out of these insights that we cultivated the notions of "*Complexity Awareness*", "*Constrained Behavior*" and "*Roots of Trust*" to support verifiable and cost effective I&C devices and systems.
- Fact: Most nuclear I&C safety functions are not computationally demanding.
- In the context of nuclear power <u>we often do not</u> need derivatives of "software intensive" systems and by extension, not carrying the complexity associated with these devices and systems.
- We assert, reducing complexity and enhancing reasoning about a system provides a tenable foundation for <u>justifying the trust</u> in the system.

COMPLEXITY AWARE CONCEPT

- A <u>complexity aware design</u> avoids (or can't) encroach into the unknown state space (the light yellow space).
- Desirable to stay on valid and correct state space paths (green and dark yellow).
- By limiting complexity (by design), paths into the unknown state space are limited to the point we can reason about the design and its implemented behavior.
- **This can only occur through architectural solutions**

Domain of behavior state space diagram

Design flaw or omission flaw

Normal Execution - ***Path exercised continuously in normal situations***

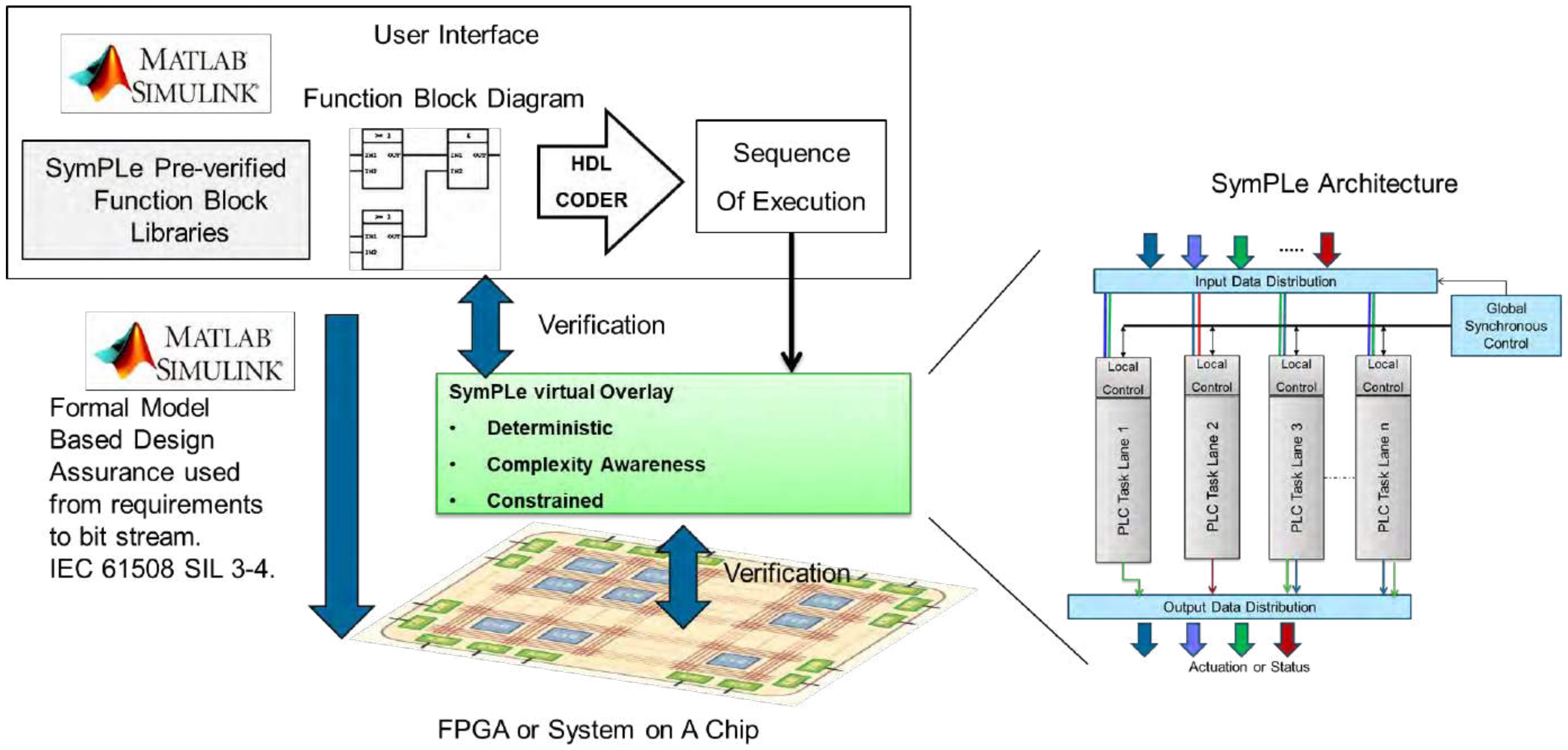
valid, but exception execution - ***Path exercised in occasional but tested situations***

Unforeseen and unknown execution, **not tested resulting in erroneous behavior**

VCU College of Engineering

EPRI ELECTRIC POWER RESEARCH INSTITUTE

SymPle Architecture Concept



- ***SymPLe is a HW solution.*** It allows programmability and computation at low orthogonality.
- ***Engineer Accessible***: By adopting overlay architecture, we hope to make SymPLe accessible like a CPU based architecture – but without its complexity - function blocks are the execution functions.
- *SymPLe* is a ***architectural viewpoint*** that seeks to maximize reasoning, transparency and evidence while avoiding unnecessary complexity

Basic Design Tenants of SymPLe

- ***Model based formal Design and Verification*** – Maximize the transparency and evidence about design assurance, development, and implementation.
- *Verifiability* - SymPLe is limited in what it can do – it trades computational power for verifiability.
- *Composability* - The behavior of "composed" element is a composition of the behaviors of its constituent elements, with well-defined, unambiguous rules of composition.
  - –Interfaces of elements are unambiguously specified, including behavior.
  - –Interactions across elements occur only through their specified interfaces
  - – Assume guarantee reasoning
- *Orthogonal* - The system is modularized using principles of separation of concerns, considering orthogonality[1] of functions and data.
  - Think of Lego blocks – can only interact in a few restricted ways.
  - Only required interactions are allowed. The architecture precludes unwanted interactions and unwanted, unknown hidden coupling or dependencies.
  - Each element (e.g., a FB unit) is internally well-architected and relatively simple.
- *Determinism* - The system is architected (satisfying conditions above) to be predictable and synchronous in it's execution behavior.

These Tenants along with a formal model based design and verification methodology allows CCFs to be identified before deployment, and enhances the ability to reason about system during qualification.

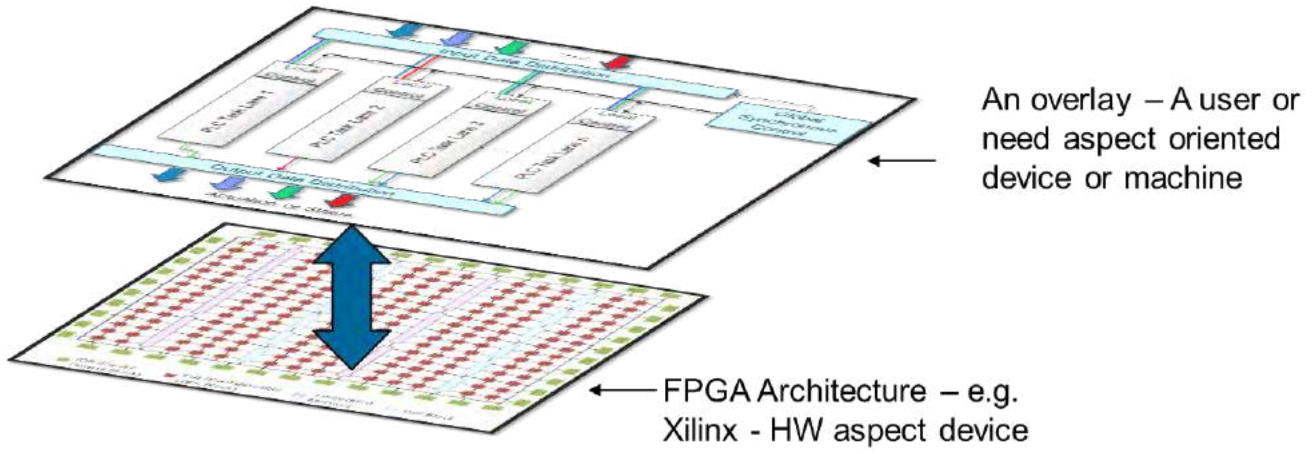
1 = a relatively small set of primitive constructs can be combined in a relatively small number of ways to build the control and data structures of program behavior

VCU College of Engineering

EPRI ELECTRIC POWER RESEARCH INSTITUTE

SymPLe: Complexity Awareness Design + FPGA Overlay Architectures + Model Based Engineering

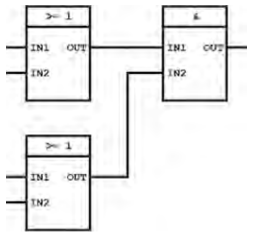
- Overlay architectures are computational systems that are designed on top of a traditional FPGA fabric.
- Overlays are not “fixed” designs or reverse engineered old designs:
  - They employ a computational model
  - They represent a user domain
  - They encode requirements from the users domain.
- Overlay architectures allow a domain community to decide what is “important” them.
- For the nuclear I&C community – we know what is important
  - Verifiability, Safety, Security and evidence for trust.


An overlay – A user or need aspect oriented device or machine
FPGA Architecture – e.g. Xilinx - HW aspect device
Input Data Distribution
Output Data Distribution
Global Synchronous Control

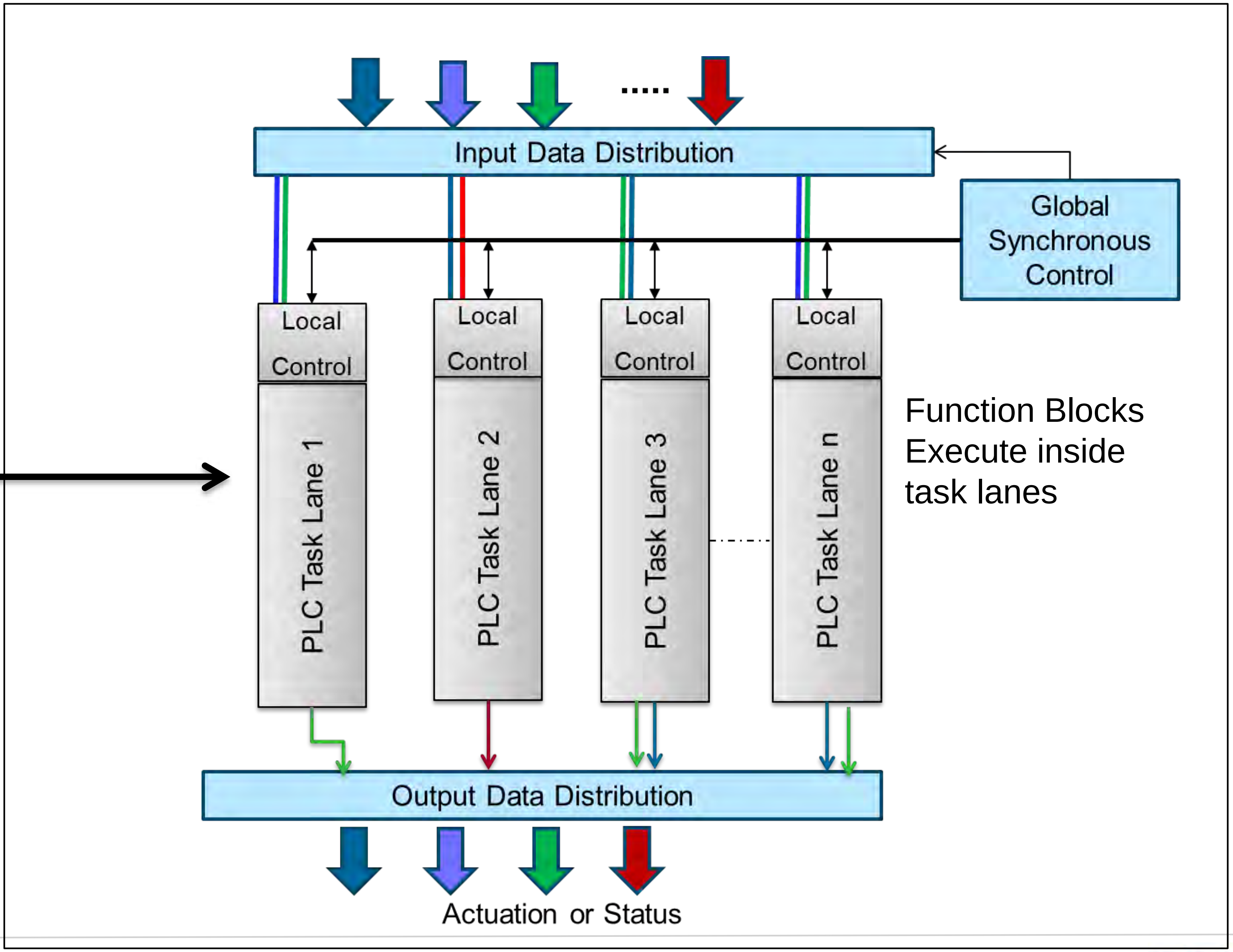
High Level Architecture Model of SymPLe

SymPLe Pre-verified FB Libraries

HDL CODER

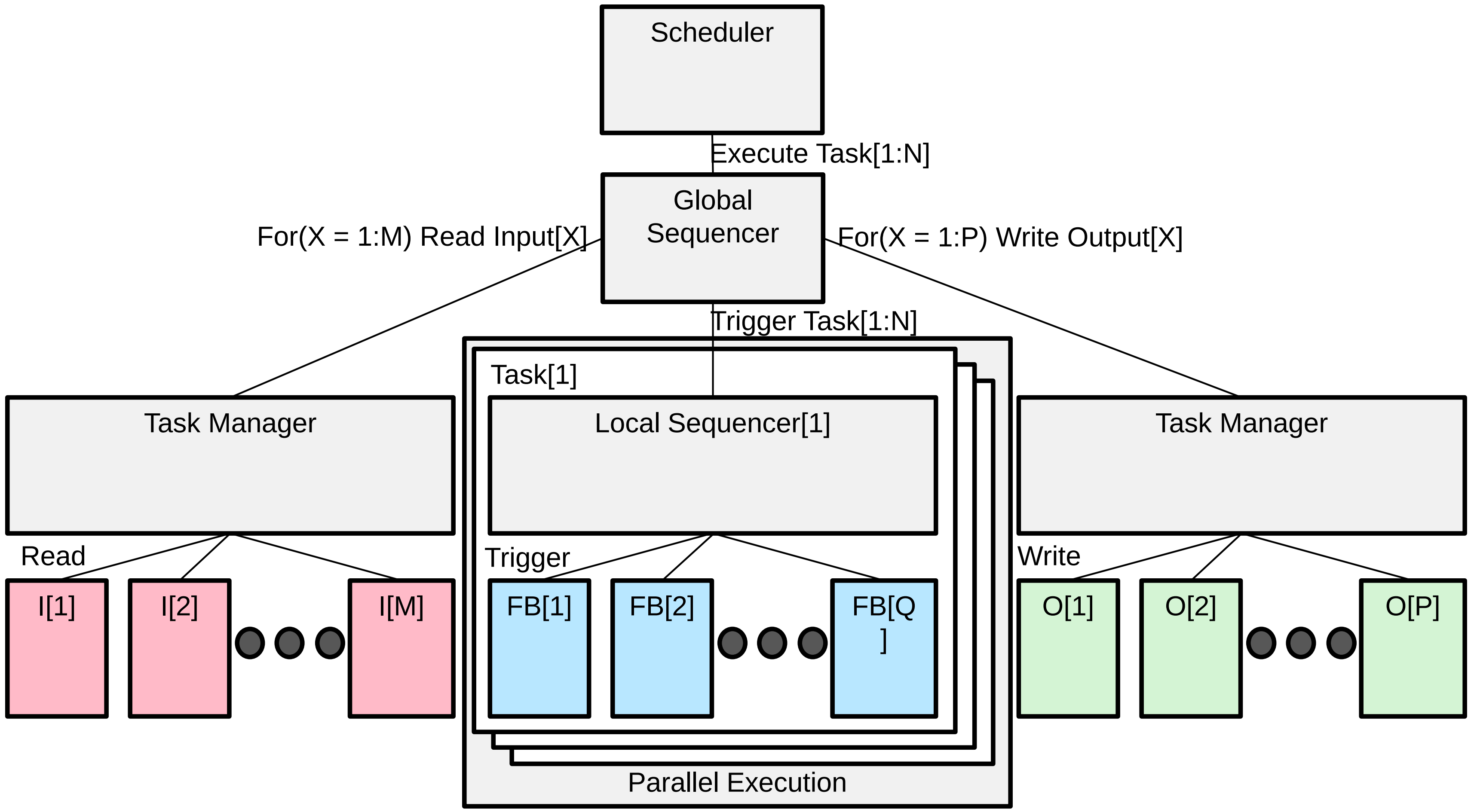


Sequence of FB executions



Function Blocks Execute inside task lanes

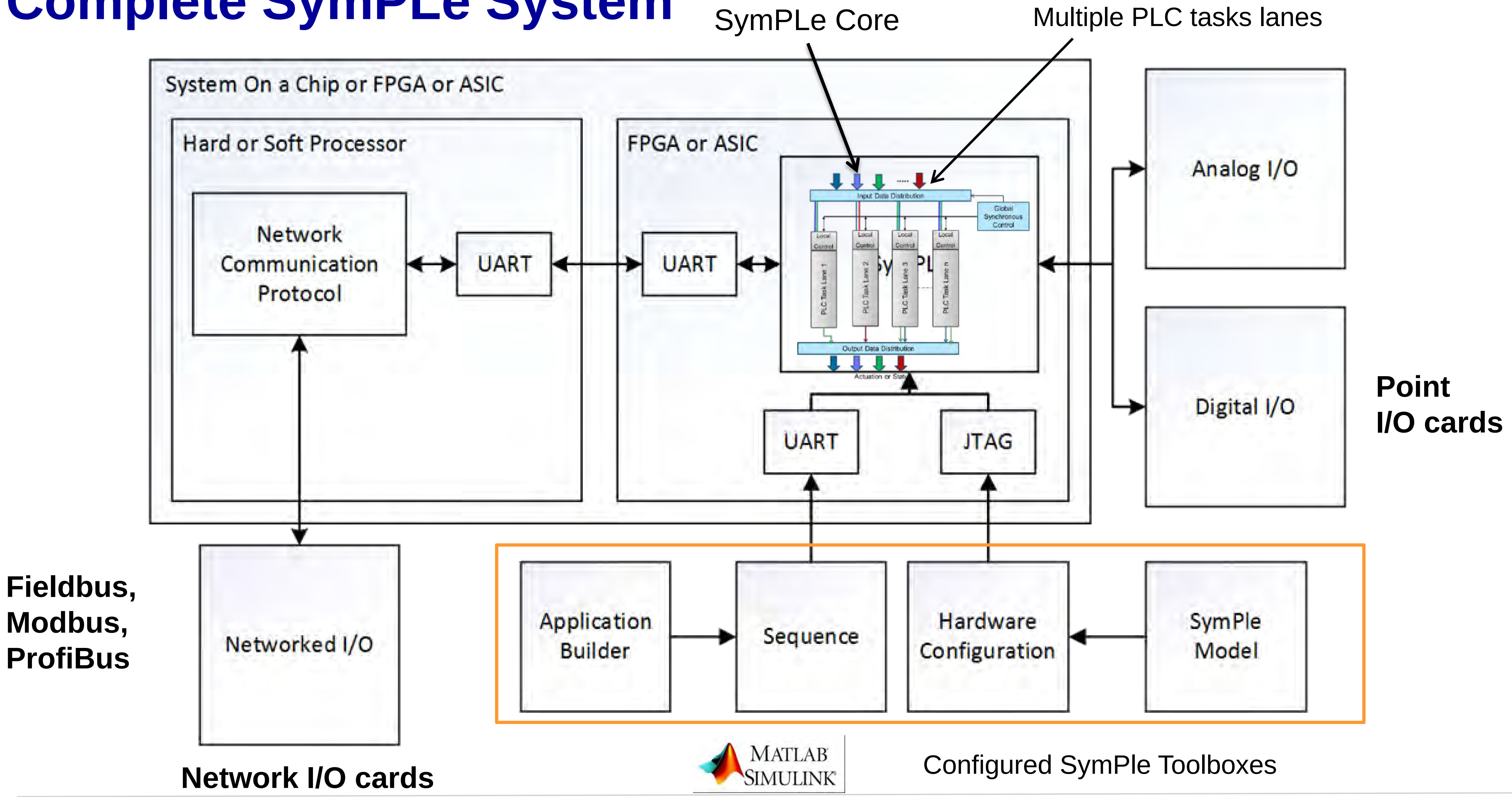
Illustration of SymPLe FB Execution



College of Engineering

ELECTRIC POWER RESEARCH INSTITUTE

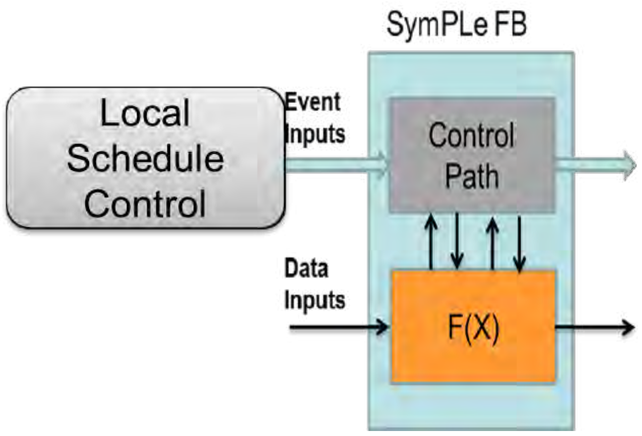
Complete SymPLe System



SymPLe Function Blocks

Most safety critical I&C applications in NPP are not computationally challenging. Don't need complex operations.

Function Block Architecture



- Inspired by IEC 61131-3 and IEC 61499-1
- Deterministic, synchronous behavior.
- Separation of control and dataflow with clear and defined interconnections
- Formal semantics

Elementary FBs

Instruction	Description
AND, OR, NOT, XOR, NAND, NOR	Logical Operators
BAND, BOR, BNOT, BXOR, BNAND, BNOR, BSL, BSR	Bitwise Logical Operators
MAX, MIN, MUX	Selection Operators
GT, GE, EQ, LT, LE, NE	Comparison Operators
ADD, SUB, MUL, DIV	Arithmetic Operators
MOVE	System Operators
QMNtoINT, INTtoQMN, BOOLtoSAFEBOOL, SAFEBOOLtoBOOL	Type Conversion

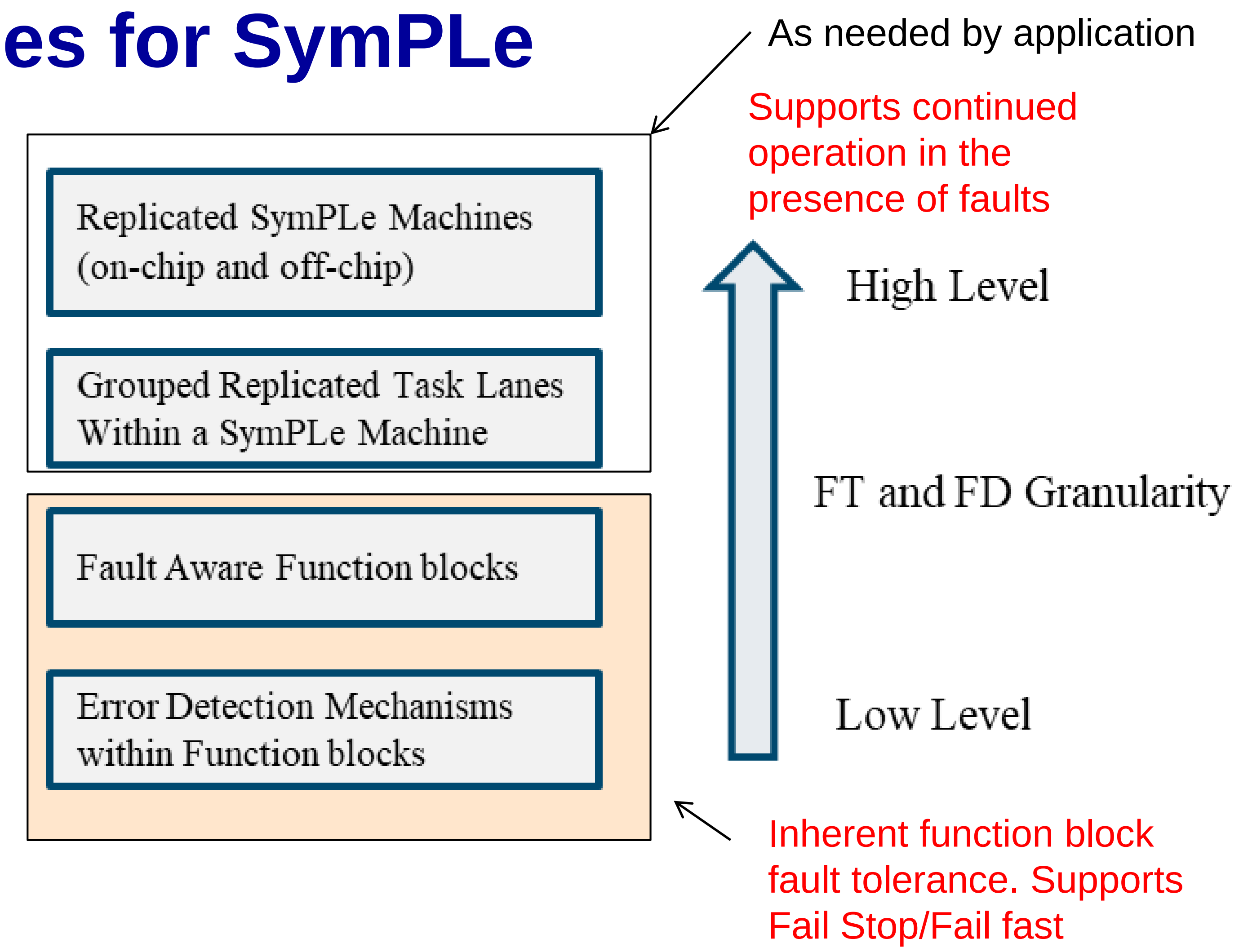
Built-up FBs

Instruction	Description
PID	Control Operator
TON, TOF	Timer Operator
FXTOI, ITOFX	Data Conversion Operator
AVOTE	Analog Voting
BVOTE	Digital Voting
MEM	Memory Operator
LOG	Logging Operator

All V 2.0 Function blocks have undergone formal verification and extensive testing.

Fault Tolerance Strategies for SymPLe

- Our approach to enhancing reliability and supporting safety requirements was a hierarchical approach to fault detection and tolerance.
- We enforce low level fault detection to support a fail-fast/fail-stop state behavior before the error propagates beyond boundaries of the system.
- Higher levels of redundancy are driven by application needs (TMR) and designer chooses when to use them.


As needed by application
Supports continued operation in the presence of faults
Replicated SymPLe Machines (on-chip and off-chip)
Grouped Replicated Task Lanes Within a SymPLe Machine
Fault Aware Function blocks
Error Detection Mechanisms within Function blocks
High Level
FT and FD Granularity
Low Level
Inherent function block fault tolerance. Supports Fail Stop/Fail fast

Model Based Design Assurance and Verification for SymPLe IEC 61508 Process

Athira Jayakumar[2], Smitha Gautham[1] , Jason Moore[3]
Virginia Commonwealth University
Department of ECE
Richmond VA
[3]Mathworks Consulting Group
VCU PhD candidate[1], VCU MS candidate[2]

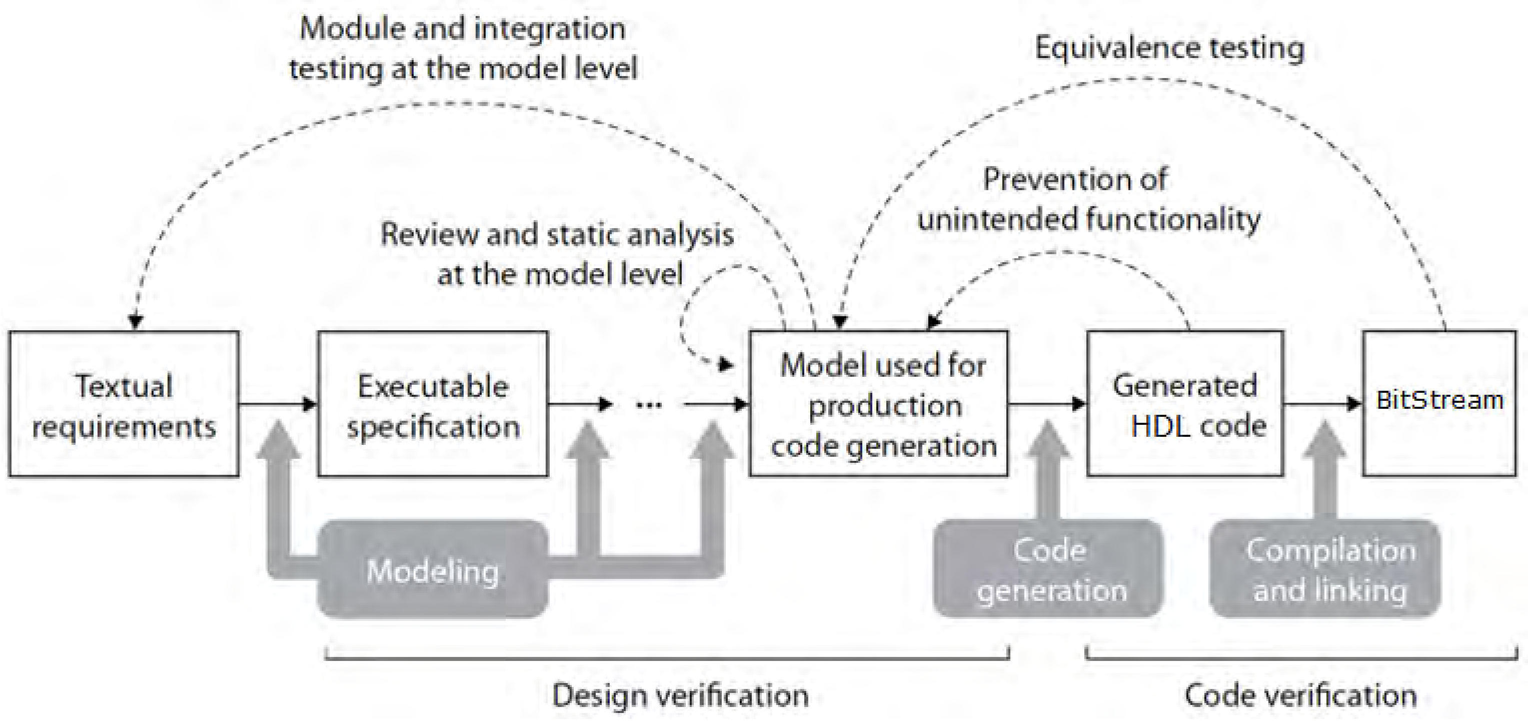
Model Based Design Verification Workflow

Model Verification

Discover design errors at design time

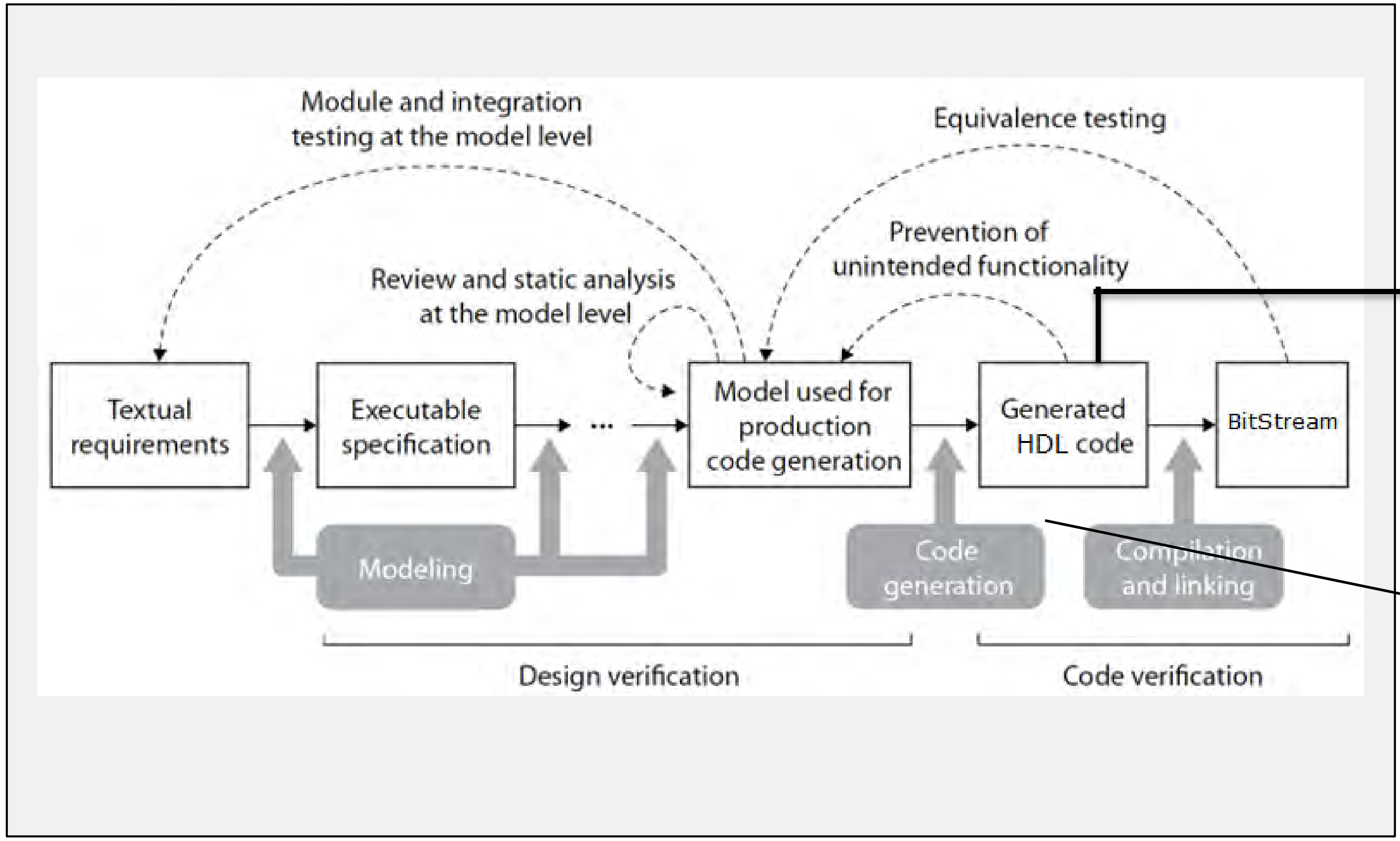
Code Verification

Gain confidence in the generated code

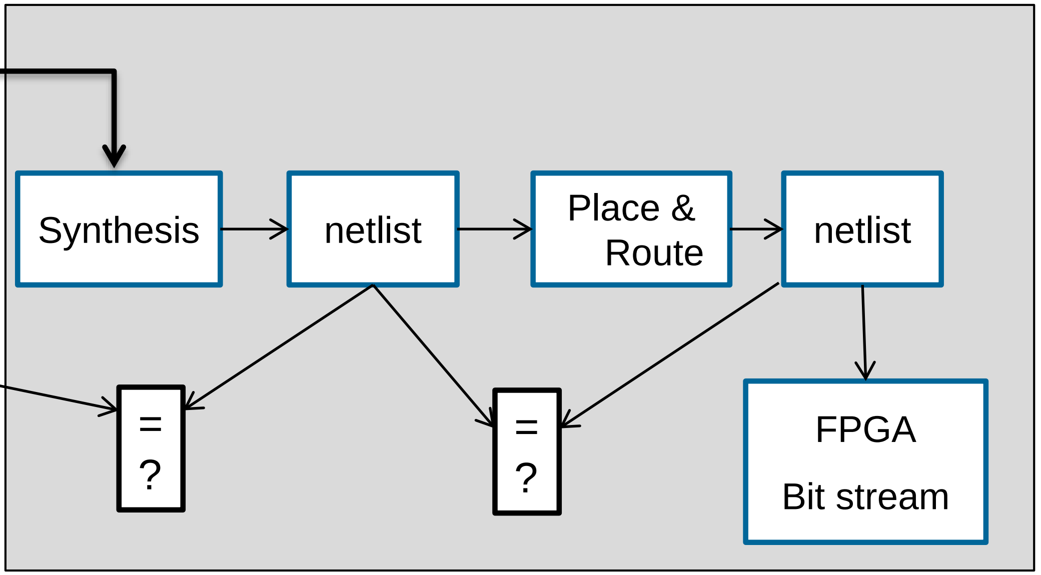


The Major Tool Flows

MATLAB SIMULINK



onespin assuring IC integrity XILINX



OneSpin just acquired – Future workflow

IEC 61508 Guidance

- 61508 Section 3 Annex Table
- There are many of these tables
- These tables provide guidance on appropriate practices, techniques needed for compliance.
- The 61508 standard provides no guidance with respect to "how" or "methodology".
- Huge gap between standard and (methods and tools).
- Selecting IEC 61508 qualified tools is important.

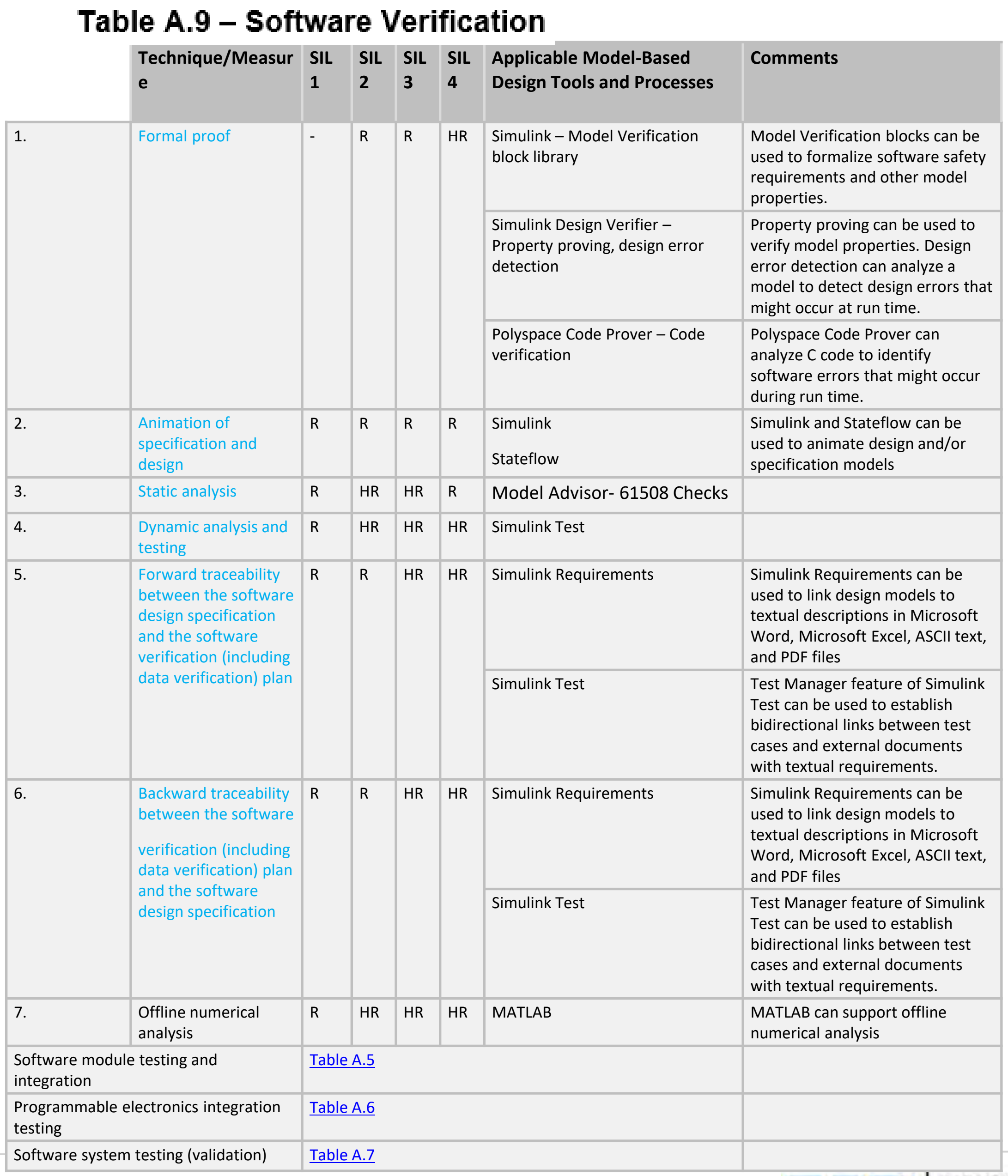


Table A.9 – Software Verification

	Technique/Measure	SIL 1	SIL 2	SIL 3	SIL 4	Applicable Model-Based Design Tools and Processes	Comments
1.	Formal proof	-	R	R	HR	Simulink – Model Verification block library	Model Verification blocks can be used to formalize software safety requirements and other model properties.
						Simulink Design Verifier – Property proving, design error detection	Property proving can be used to verify model properties. Design error detection can analyze a model to detect design errors that might occur at run time.
						Polyspace Code Prover – Code verification	Polyspace Code Prover can analyze C code to identify software errors that might occur during run time.
2.	Animation of specification and design	R	R	R	R	Simulink Stateflow	Simulink and Stateflow can be used to animate design and/or specification models
3.	Static analysis	R	HR	HR	R	Model Advisor- 61508 Checks	
4.	Dynamic analysis and testing	R	HR	HR	HR	Simulink Test	
5.	Forward traceability between the software design specification and the software verification (including data verification) plan	R	R	HR	HR	Simulink Requirements	Simulink Requirements can be used to link design models to textual descriptions in Microsoft Word, Microsoft Excel, ASCII text, and PDF files
						Simulink Test	Test Manager feature of Simulink Test can be used to establish bidirectional links between test cases and external documents with textual requirements.
6.	Backward traceability between the software verification (including data verification) plan and the software design specification	R	R	HR	HR	Simulink Requirements	Simulink Requirements can be used to link design models to textual descriptions in Microsoft Word, Microsoft Excel, ASCII text, and PDF files
						Simulink Test	Test Manager feature of Simulink Test can be used to establish bidirectional links between test cases and external documents with textual requirements.
7.	Offline numerical analysis	R	HR	HR	HR	MATLAB	MATLAB can support offline numerical analysis
Software module testing and integration		Table A.5					
Programmable electronics integration testing		Table A.6					
Software system testing (validation)		Table A.7					

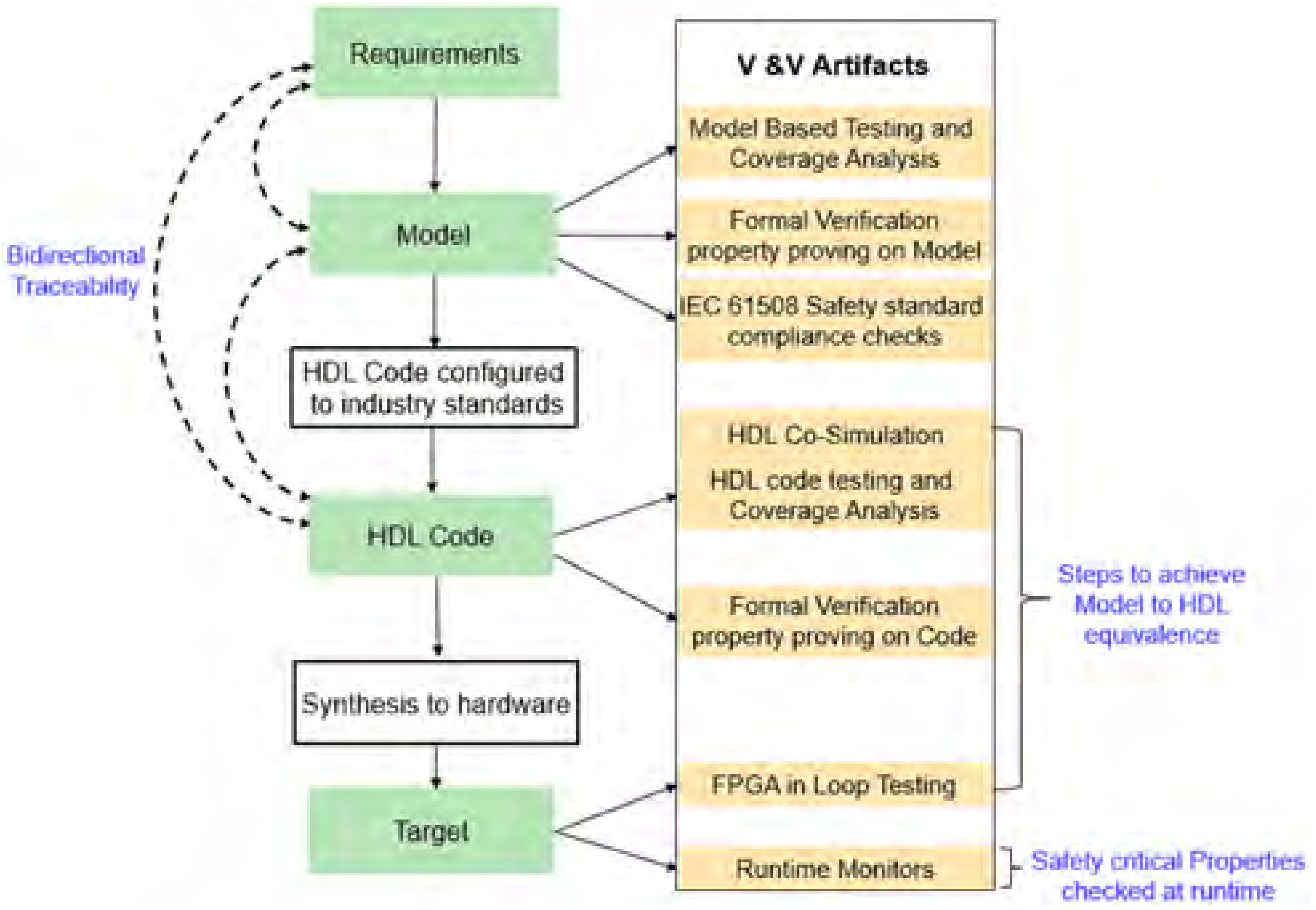
VCU College of Engineering

EPRI ELECTRIC POWER RESEARCH INSTITUTE

SymPLe Design Assurance V & V Workflow

- The Design Assurance workflow we developed and adopted was by far the most challenging part of the project
- IEC 61508 - International functional safety standard for Electronic/Programmable Electronic Safety related systems.
- SymPLe – Goal is SIL Level 4
- Need for a Systematic Verification Process with Evidence Reports

Evidence and Artifacts Perspective

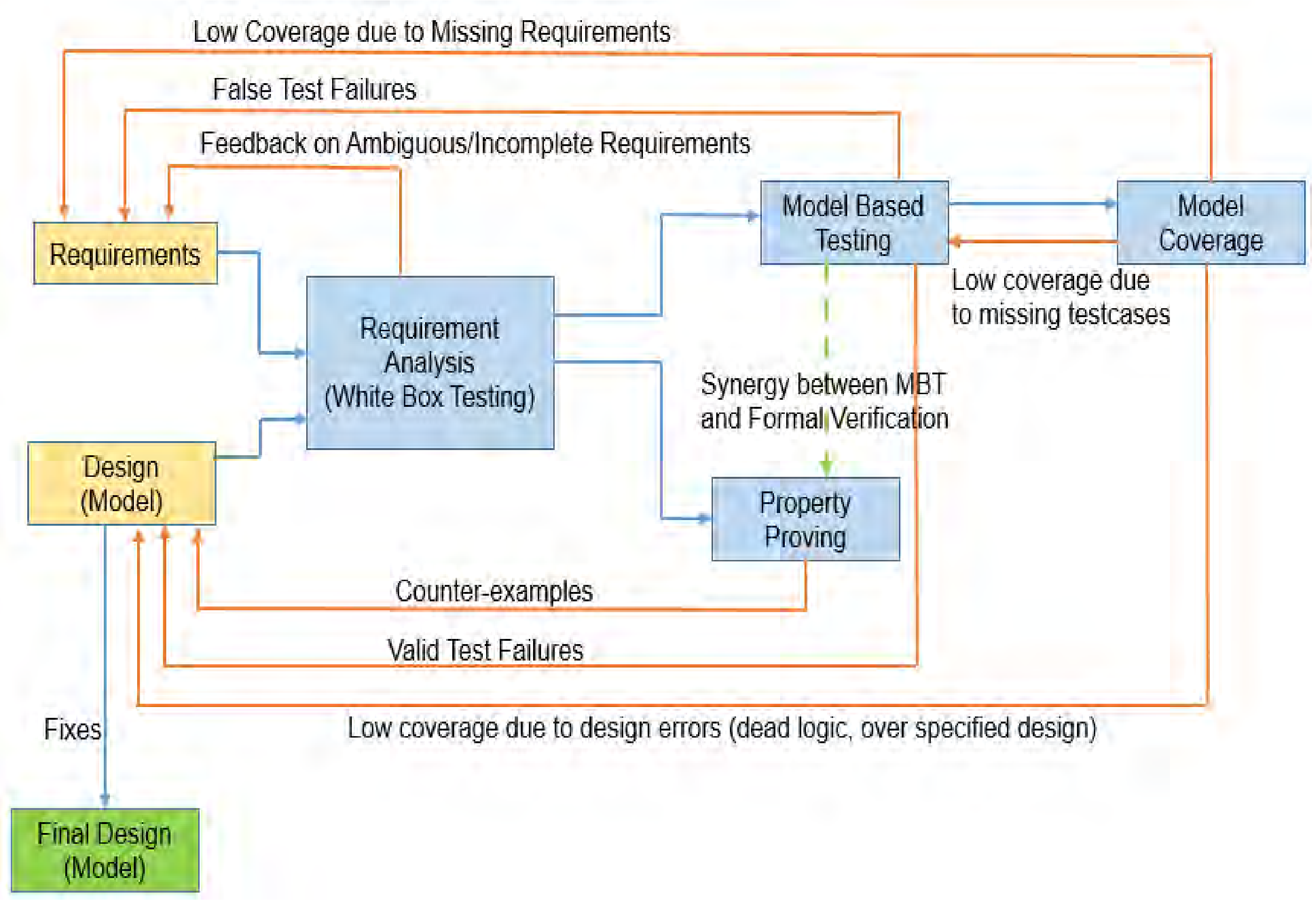

Requirements
Model
HDL Code configured to industry standards
HDL Code
Synthesis to hardware
Target
Bidirectional Traceability
V &V Artifacts
Model Based Testing and Coverage Analysis
Formal Verification property proving on Model
IEC 61508 Safety standard compliance checks
HDL Co-Simulation
HDL code testing and Coverage Analysis
Formal Verification property proving on Code
FPGA in Loop Testing
Runtime Monitors
Steps to achieve Model to HDL equivalence
Safety critical Properties checked at runtime

VCU College of Engineering

EPRI ELECTRIC POWER RESEARCH INSTITUTE

Design Assurance Process Perspective of V&V Workflow

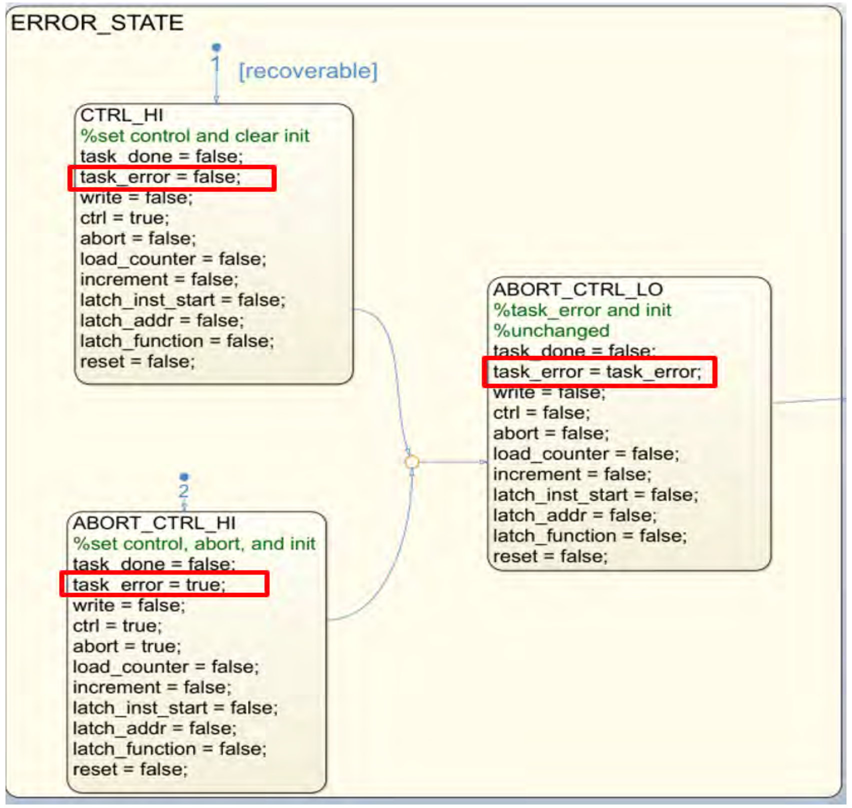
- Most V-Models portray a linear flow from requirements to design (waterfall).
- Note, the V- model aspects of our workflow are not linear. We have feedback loops.
- Requirements/specifications are refined by gaining more understanding of the desired (and undesired) system behaviors.
- This is the way it works in real life.



Low Coverage due to Missing Requirements
False Test Failures
Feedback on Ambiguous/Incomplete Requirements
Requirements
Model Based Testing
Model Coverage
Low coverage due to missing testcases
Requirement Analysis (White Box Testing)
Synergy between MBT and Formal Verification
Design (Model)
Property Proving
Counter-examples
Valid Test Failures
Low coverage due to design errors (dead logic, over specified design)
Fixes
Final Design (Model)

A few examples from V&V process

- Due to time limits, we can't go through every aspect of the model based V&V workflow today.
- We select a few, and jump to the main findings.
- The intention of the stateflow implementation as seen in figure is to keep the task_error true when there is a non-recoverable error.

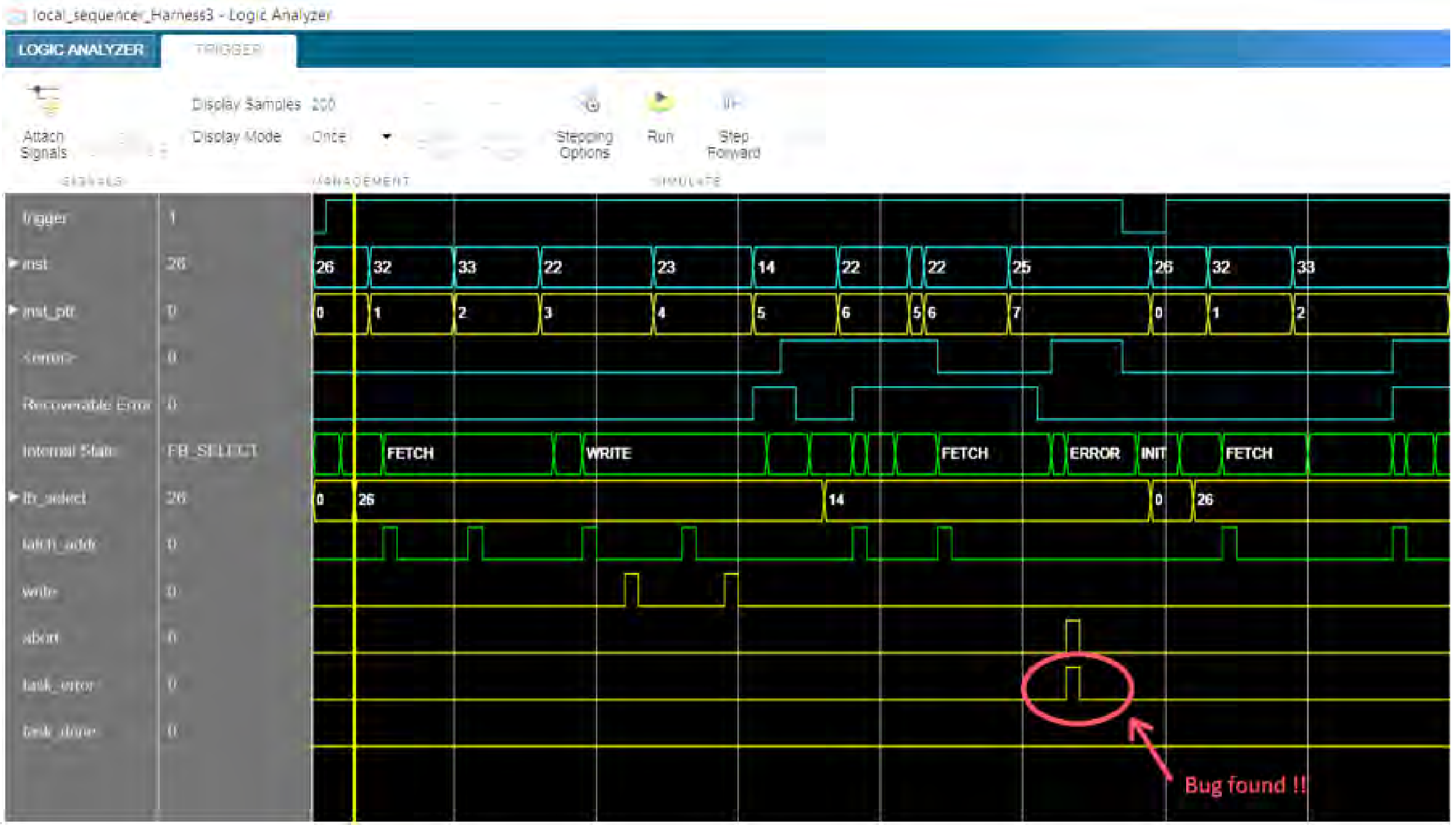


```
ERROR_STATE
1 [recoverable]

CTRL_HI
%set control and clear init
task_done = false;
task_error = false;
write = false;
ctrl = true;
abort = false;
load_counter = false;
increment = false;
latch_inst_start = false;
latch_addr = false;
latch_function = false;
reset = false;

ABORT_CTRL_LO
%task_error and init
%unchanged
task_done = false;
task_error = task_error;
write = false;
ctrl = false;
abort = false;
load_counter = false;
increment = false;
latch_inst_start = false;
latch_addr = false;
latch_function = false;
reset = false;

2
ABORT_CTRL_HI
%set control, abort, and init
task_done = false;
task_error = true;
write = false;
ctrl = true;
abort = true;
load_counter = false;
increment = false;
latch_inst_start = false;
latch_addr = false;
latch_function = false;
reset = false;
```



VCU College of Engineering

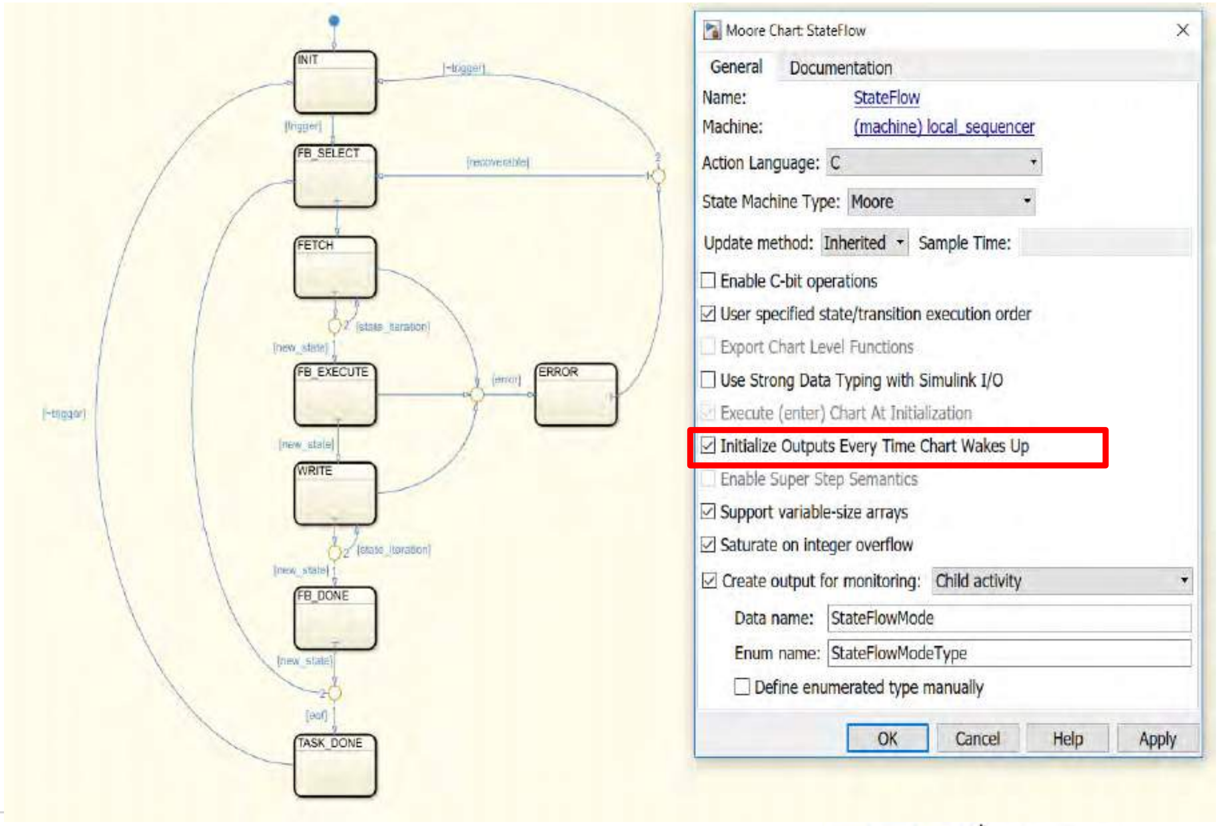
EPRI ELECTRIC POWER RESEARCH INSTITUTE

Example1 – State Chart configuration issue

- A wrong state chart setting “Initialize Outputs Every Time Chart Wakes Up” resulted in incorrect output behavior even though the stateflow implementation is perfectly alright.
- State chart setting caused the output to be initialized to zero each time the state chart woke up which was on each clock cycle.
- Model Based testing was effective enough to find even the hardest configuration related errors.
- 78% design faults were caught at Model Based Testing.

- Note:

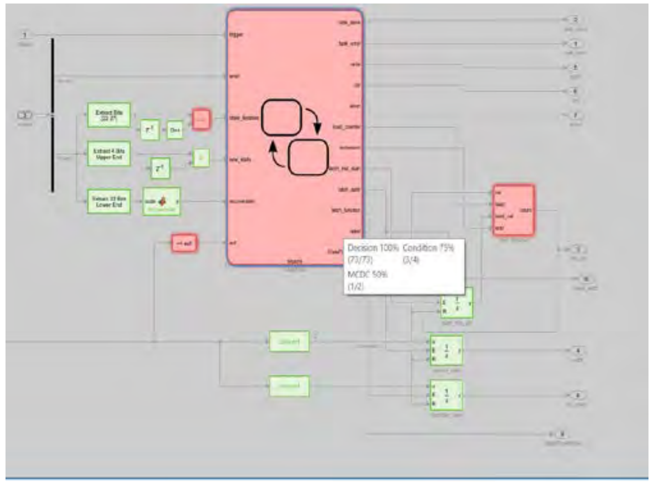
The Bug was found well before HDL code generation !!

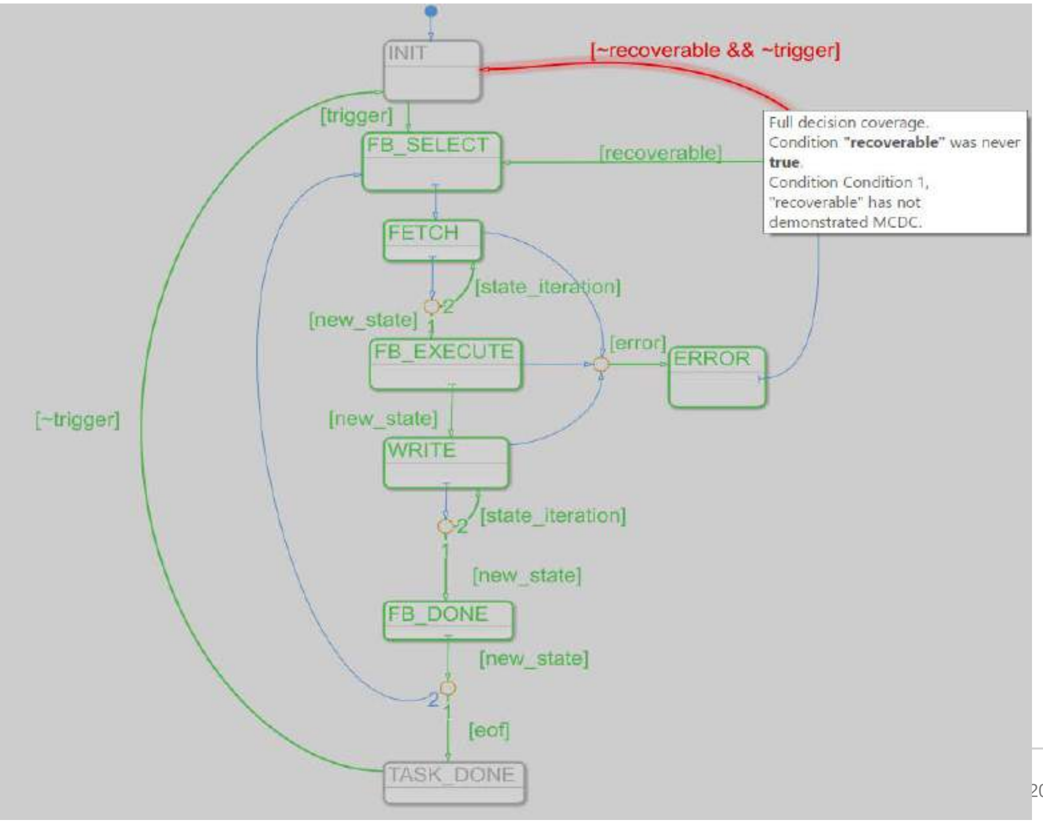


Name: StateFlow
Machine: (machine) local_sequencer
Action Language: C
State Machine Type: Moore
Update method: Inherited Sample Time:
☐ Enable C-bit operations
☑ User specified state/transition execution order
☐ Export Chart Level Functions
☐ Use Strong Data Typing with Simulink I/O
☑ Execute (enter) Chart At Initialization
☑ Initialize Outputs Every Time Chart Wakes Up
☐ Enable Super Step Semantics
☑ Support variable-size arrays
☑ Saturate on integer overflow
☑ Create output for monitoring: Child activity
Data name: StateFlowMode
Enum name: StateFlowModeType
☐ Define enumerated type manually

Structural Coverage

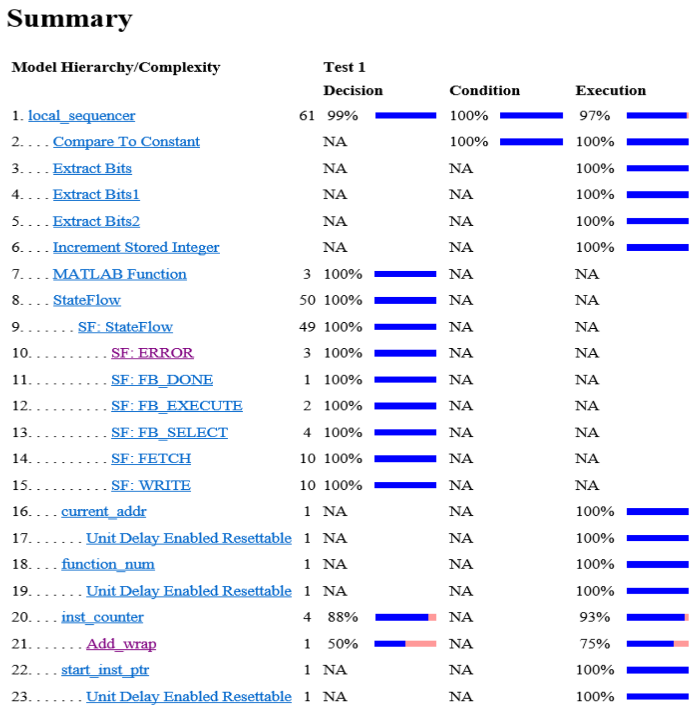
- Structural Model Coverage – How much of the model has been exercised by testcases?
  - To determine design errors like dead Logic branches or over-specified design
  - To determine if sufficient test vectors have been created
  - To determine if existing requirements are sufficient.
- Used Modified Condition/Decision Coverage (MC/DC) coverage criteria.





Full decision coverage.
Condition **"recoverable"** was never **true**.
Condition Condition 1, "recoverable" has not demonstrated MCDC.

- An example of an over-specified design causing less coverage.
- After removing the redundant condition in the transition, MC/DC coverage is no more applicable for the model due to absence of multiple condition decision statements or state transitions.
- Depicts simplicity of design.

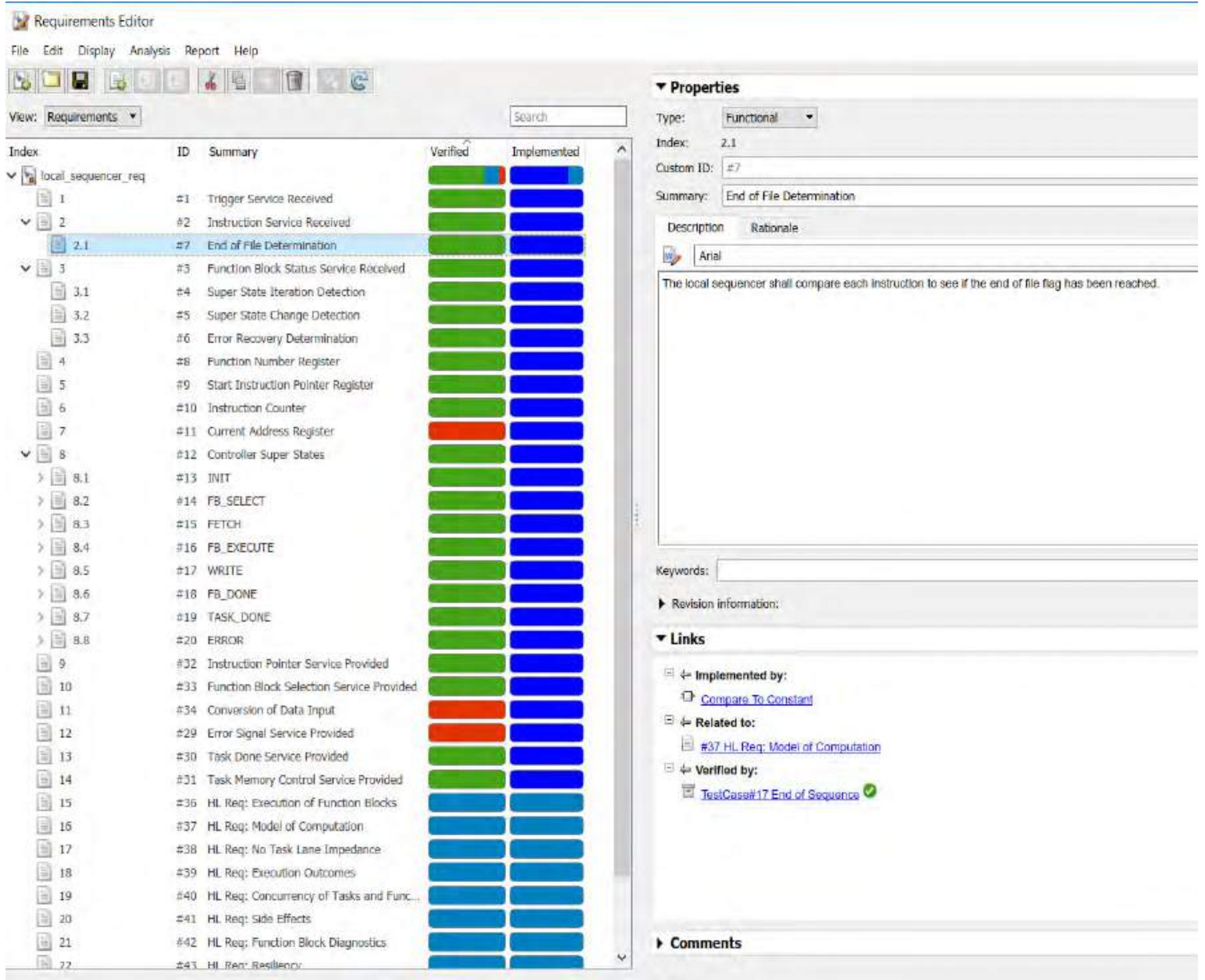


Summary

Model Hierarchy/Complexity		Test 1 Decision	Condition	Execution
1. local_sequencer	61	99%	100%	97%
2. . . . Compare To Constant		NA	100%	100%
3. . . . Extract Bits		NA	NA	100%
4. . . . Extract Bits1		NA	NA	100%
5. . . . Extract Bits2		NA	NA	100%
6. . . . Increment Stored Integer		NA	NA	100%
7. . . . MATLAB Function	3	100%	NA	NA
8. . . . StateFlow	50	100%	NA	NA
9. . . . . . . SF: StateFlow	49	100%	NA	NA
10. . . . . SF: ERROR	3	100%	NA	NA
11. . . . . SF: FB_DONE	1	100%	NA	NA
12. . . . . SF: FB_EXECUTE	2	100%	NA	NA
13. . . . . SF: FB_SELECT	4	100%	NA	NA
14. . . . . SF: FETCH	10	100%	NA	NA
15. . . . . SF: WRITE	10	100%	NA	NA
16. . . . current_addr	1	NA	NA	100%
17. . . . . . . Unit Delay Enabled Resettable	1	NA	NA	100%
18. . . . function_num	1	NA	NA	100%
19. . . . . . . Unit Delay Enabled Resettable	1	NA	NA	100%
20. . . . inst_counter	4	88%	NA	93%
21. . . . . . . Add_wrap	1	50%	NA	75%
22. . . . start_inst_ptr	1	NA	NA	100%
23. . . . . . . Unit Delay Enabled Resettable	1	NA	NA	100%

Functional Coverage

- Bi-directional traceability links between Testcases and Requirements
- Ensures 100% Requirements coverage


Requirements Editor

Index	ID	Summary	Verified	Implemented
local_sequencer_req			🟩	🟦
1	#1	Trigger Service Received	🟩	🟦
2	#2	Instruction Service Received	🟩	🟦
2.1	#7	End of File Determination	🟩	🟦
3	#3	Function Block Status Service Received	🟩	🟦
3.1	#4	Super State Iteration Detection	🟩	🟦
3.2	#5	Super State Change Detection	🟩	🟦
3.3	#6	Error Recovery Determination	🟩	🟦
4	#8	Function Number Register	🟩	🟦
5	#9	Start Instruction Pointer Register	🟩	🟦
6	#10	Instruction Counter	🟩	🟦
7	#11	Current Address Register	🟥	🟦
8	#12	Controller Super States	🟩	🟦
8.1	#13	INIT	🟩	🟦
8.2	#14	FB_SELECT	🟩	🟦
8.3	#15	FETCH	🟩	🟦
8.4	#16	FB_EXECUTE	🟩	🟦
8.5	#17	WRITE	🟩	🟦
8.6	#18	FB_DONE	🟩	🟦
8.7	#19	TASK_DONE	🟩	🟦
8.8	#20	ERROR	🟩	🟦
9	#32	Instruction Pointer Service Provided	🟩	🟦
10	#33	Function Block Selection Service Provided	🟩	🟦
11	#34	Conversion of Data Input	🟥	🟦
12	#29	Error Signal Service Provided	🟥	🟦
13	#30	Task Done Service Provided	🟩	🟦
14	#31	Task Memory Control Service Provided	🟩	🟦
15	#36	HL Req: Execution of Function Blocks	🟦	🟦
16	#37	HL Req: Model of Computation	🟦	🟦
17	#38	HL Req: No Task Lane Impedance	🟦	🟦
18	#39	HL Req: Execution Outcomes	🟦	🟦
19	#40	HL Req: Concurrency of Tasks and Func...	🟦	🟦
20	#41	HL Req: Side Effects	🟦	🟦
21	#42	HL Req: Function Block Diagnostics	🟦	🟦
22	#43	HL Req: Resiliency	🟦	🟦

Properties

Type: Functional
Index: 2.1
Custom ID: #7
Summary: End of File Determination

Description | Rationale

The local sequencer shall compare each instruction to see if the end of file flag has been reached.

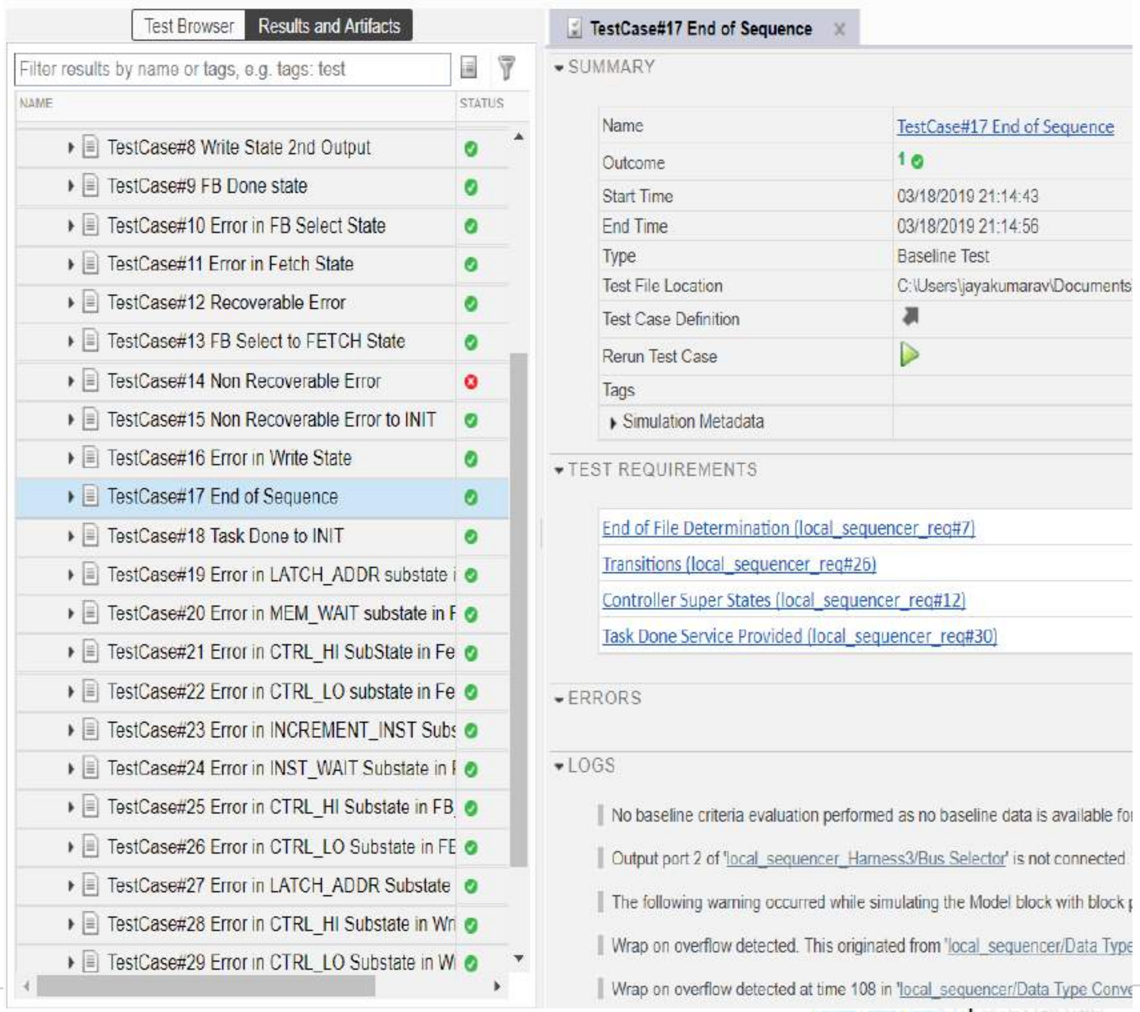
Keywords:

Revision information:

Links

Implemented by:
Compare To Constant
Related to:
#37 HL Req: Model of Computation
Verified by:
TestCase#17 End of Sequence ✓

Comments



NAME	STATUS
TestCase#8 Write State 2nd Output	✓
TestCase#9 FB Done state	✓
TestCase#10 Error in FB Select State	✓
TestCase#11 Error in Fetch State	✓
TestCase#12 Recoverable Error	✓
TestCase#13 FB Select to FETCH State	✓
TestCase#14 Non Recoverable Error	✗
TestCase#15 Non Recoverable Error to INIT	✓
TestCase#16 Error in Write State	✓
TestCase#17 End of Sequence	✓
TestCase#18 Task Done to INIT	✓
TestCase#19 Error in LATCH_ADDR substate i	✓
TestCase#20 Error in MEM_WAIT substate in F	✓
TestCase#21 Error in CTRL_HI SubState in Fe	✓
TestCase#22 Error in CTRL_LO substate in Fe	✓
TestCase#23 Error in INCREMENT_INST Subs	✓
TestCase#24 Error in INST_WAIT Substate in I	✓
TestCase#25 Error in CTRL_HI Substate in FB	✓
TestCase#26 Error in CTRL_LO Substate in FE	✓
TestCase#27 Error in LATCH_ADDR Substate	✓
TestCase#28 Error in CTRL_HI Substate in Wr	✓
TestCase#29 Error in CTRL_LO Substate in W	✓

TestCase#17 End of Sequence

SUMMARY

Name	TestCase#17 End of Sequence
Outcome	1 ✓
Start Time	03/18/2019 21:14:43
End Time	03/18/2019 21:14:56
Type	Baseline Test
Test File Location	C:\Users\jayakumarav\Documents
Test Case Definition	
Rerun Test Case	
Tags	
Simulation Metadata	

TEST REQUIREMENTS

End of File Determination (local_sequencer_req#7)
Transitions (local_sequencer_req#26)
Controller Super States (local_sequencer_req#12)
Task Done Service Provided (local_sequencer_req#30)

ERRORS

LOGS

No baseline criteria evaluation performed as no baseline data is available fo
Output port 2 of 'local_sequencer_Harness3/Bus Selector' is not connected.
The following warning occurred while simulating the Model block with block
Wrap on overflow detected. This originated from 'local_sequencer/Data Type
Wrap on overflow detected at time 108 in 'local_sequencer/Data Type Conve

VCU College of Engineering

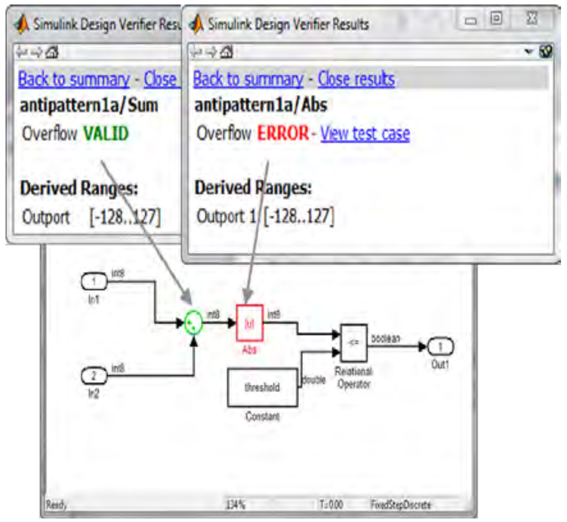
EPRI ELECTRIC POWER RESEARCH INSTITUTE

Formal Methods: Simulink Design Verifier – Main Components

Uses formal verification – Model Checking

Design Error Detection

- **Uncover** hard to find dead logic and design flaws



Simulink Design Verifier Results

Back to summary - Close results

antipattern1a/Sum
Overflow VALID

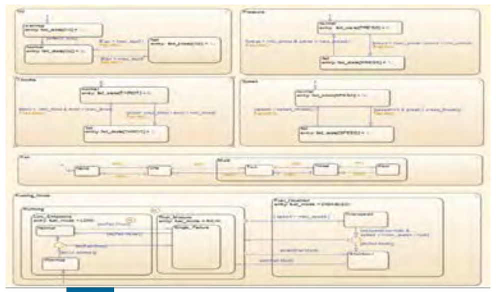
Derived Ranges:
Outport [-128..127]

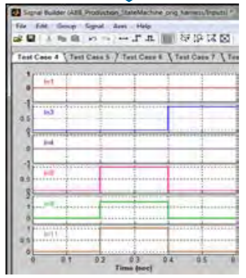
antipattern1a/Abs
Overflow ERROR - View test case

Derived Ranges:
Outport 1 [-128..127]

Test Generation

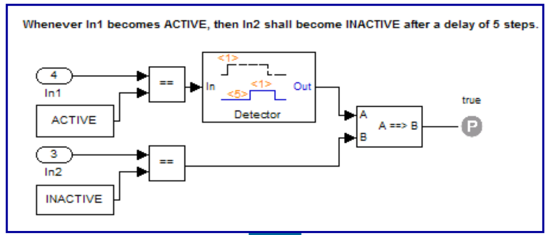
- **Automate** test case generation for coverage completion or functional tests



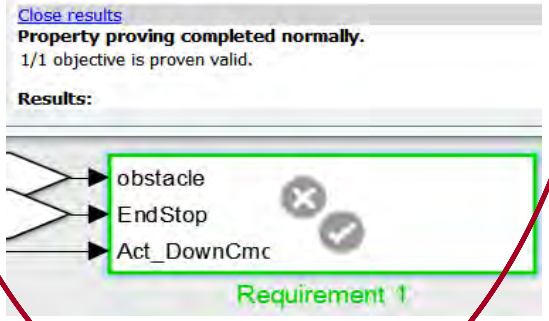


Property Proving

- **Prove** design meets requirements



Whenever In1 becomes ACTIVE, then In2 shall become INACTIVE after a delay of 5 steps.



Close results

Property proving completed normally.

1/1 objective is proven valid.

Results:

Example of Property for a Higher Level Requirement

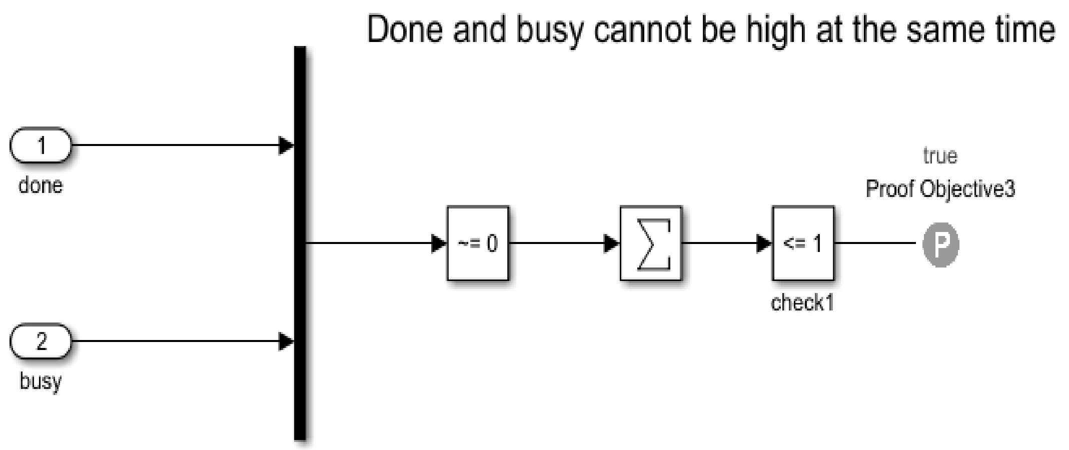
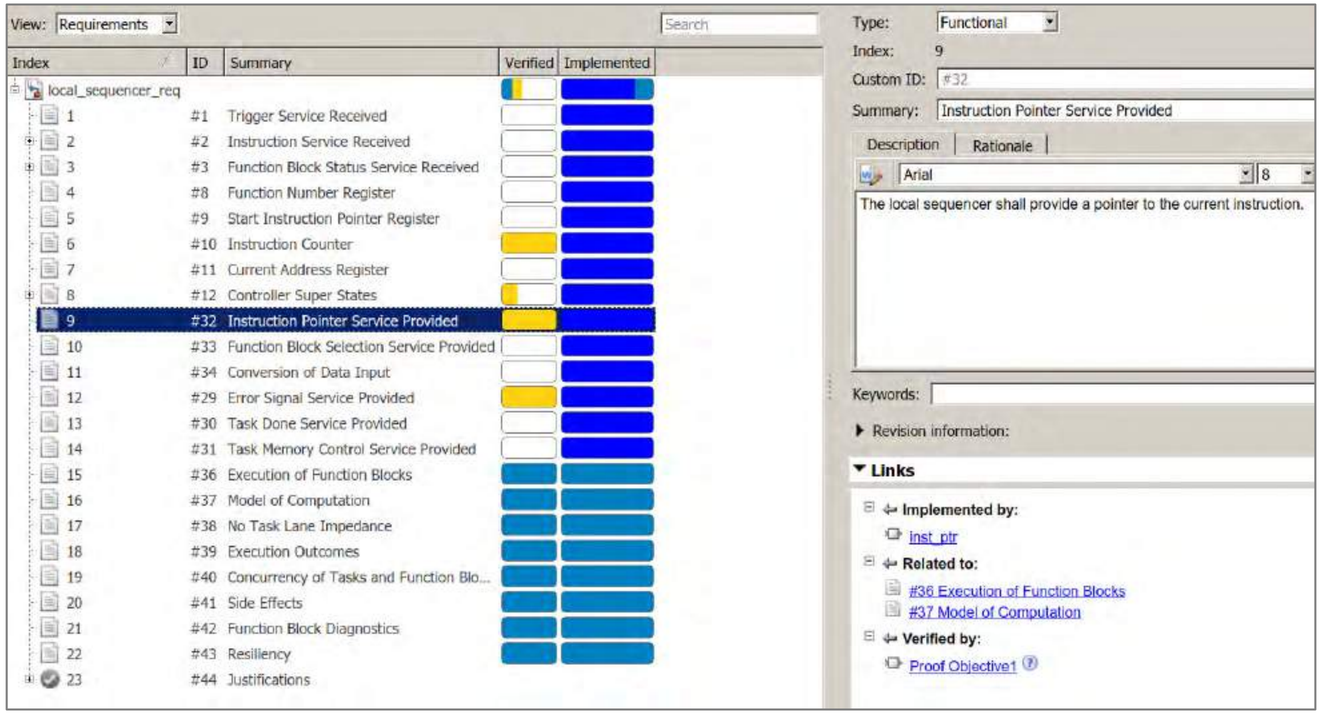


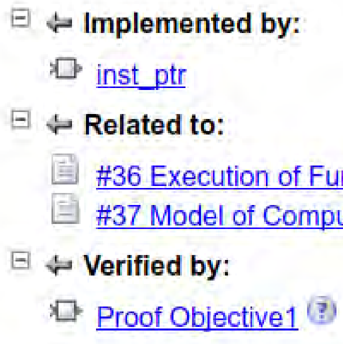
Fig: Example of a property at a higher level



View: Requirements　Search

Index	ID	Summary	Verified	Implemented
local_sequencer_req			🟨	🟦
1	#1	Trigger Service Received	⬜	🟦
2	#2	Instruction Service Received	⬜	🟦
3	#3	Function Block Status Service Received	⬜	🟦
4	#8	Function Number Register	⬜	🟦
5	#9	Start Instruction Pointer Register	⬜	🟦
6	#10	Instruction Counter	🟨	🟦
7	#11	Current Address Register	⬜	🟦
8	#12	Controller Super States	🟨	🟦
9	#32	Instruction Pointer Service Provided	🟨	🟦
10	#33	Function Block Selection Service Provided	⬜	🟦
11	#34	Conversion of Data Input	⬜	🟦
12	#29	Error Signal Service Provided	🟨	🟦
13	#30	Task Done Service Provided	⬜	🟦
14	#31	Task Memory Control Service Provided	⬜	🟦
15	#36	Execution of Function Blocks	🟦	🟦
16	#37	Model of Computation	🟦	🟦
17	#38	No Task Lane Impedance	🟦	🟦
18	#39	Execution Outcomes	🟦	🟦
19	#40	Concurrency of Tasks and Function Blo...	🟦	🟦
20	#41	Side Effects	🟦	🟦
21	#42	Function Block Diagnostics	🟦	🟦
22	#43	Resiliency	🟦	🟦
23	#44	Justifications		

Type: Functional
Index: 9
Custom ID: #32
Summary: Instruction Pointer Service Provided
Description　Rationale
Arial　8
The local sequencer shall provide a pointer to the current instruction.
Keywords:
Revision information:
Links
Implemented by:
inst_ptr
Related to:
#36 Execution of Function Blocks
#37 Model of Computation
Verified by:
Proof Objective1



Implemented by:
inst_ptr
Related to:
#36 Execution of Function Blocks
#37 Model of Computation
Verified by:
Proof Objective1

Fig: Properties are Hyper- linked to requirements

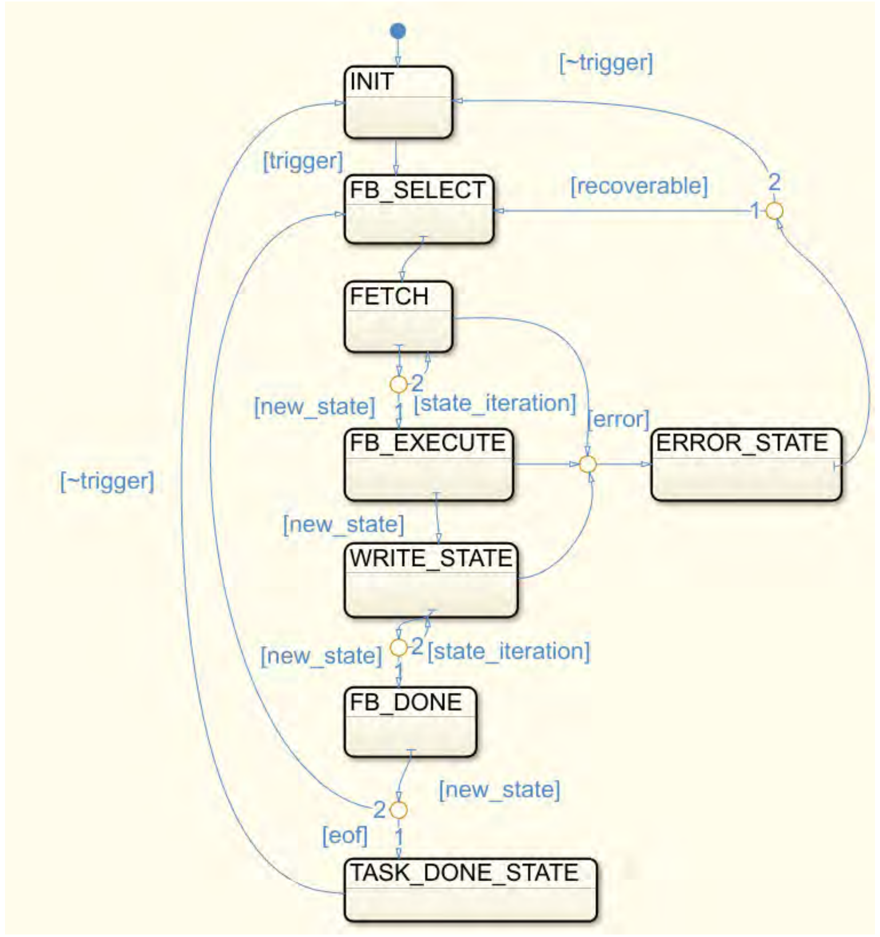
- Safety Property for a high level Requirement: done and busy -> only one of the signal is high at a given time
- Bidirectional Traceability – IEC 61508 requirement

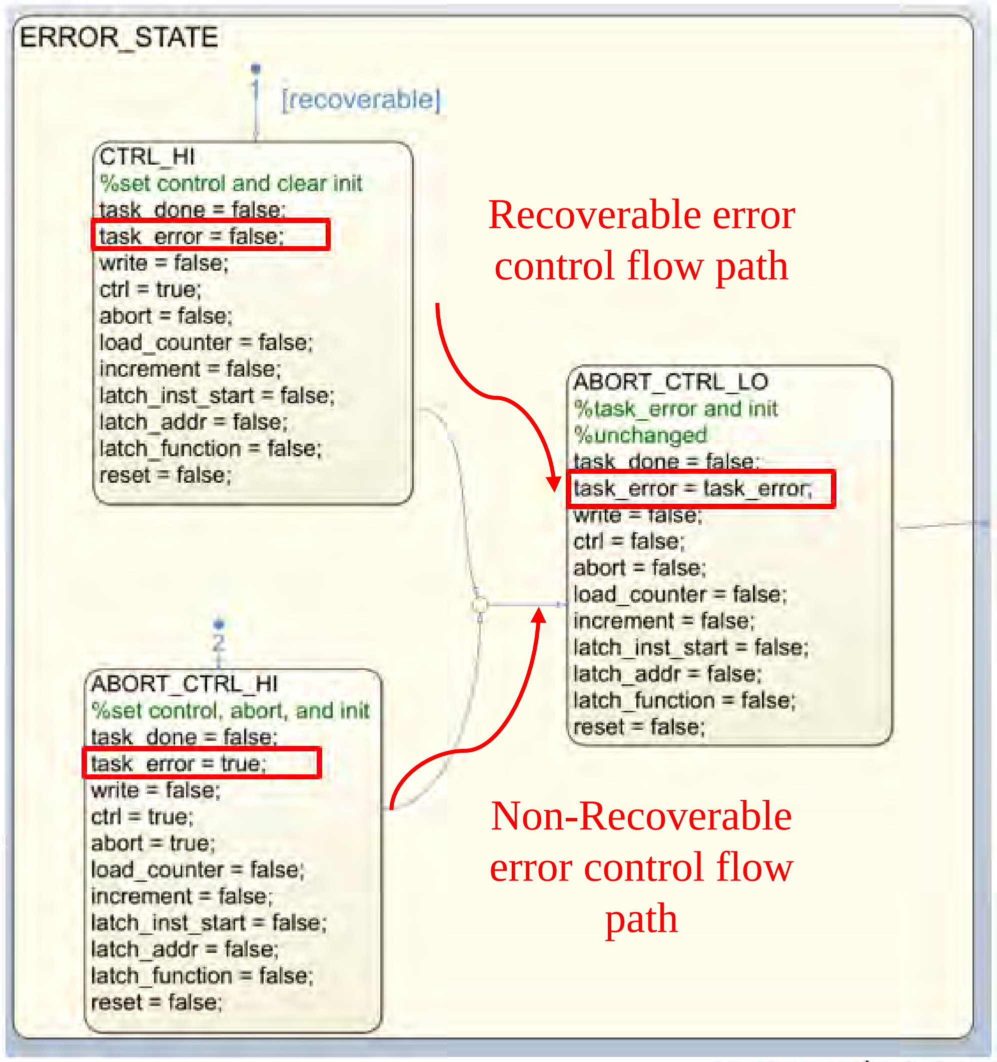
VCU College of Engineering

EPRI ELECTRIC POWER RESEARCH INSTITUTE

Example of Property for a Low Level Requirement

- "If the state is FB_SELECT, FETCH or FB_EXECUTE and an error is detected, the task_error should be high for 2 clock cycles if error is non-recoverable (recoverable=0)"




ERROR_STATE
1
[recoverable]
2
Recoverable error control flow path
Non-Recoverable error control flow path

CTRL_HI
%set control and clear init
task_done = false;
task_error = false;
write = false;
ctrl = true;
abort = false;
load_counter = false;
increment = false;
latch_inst_start = false;
latch_addr = false;
latch_function = false;
reset = false;

ABORT_CTRL_LO
%task_error and init
%unchanged
task_done = false;
task_error = task_error;
write = false;
ctrl = false;
abort = false;
load_counter = false;
increment = false;
latch_inst_start = false;
latch_addr = false;
latch_function = false;
reset = false;

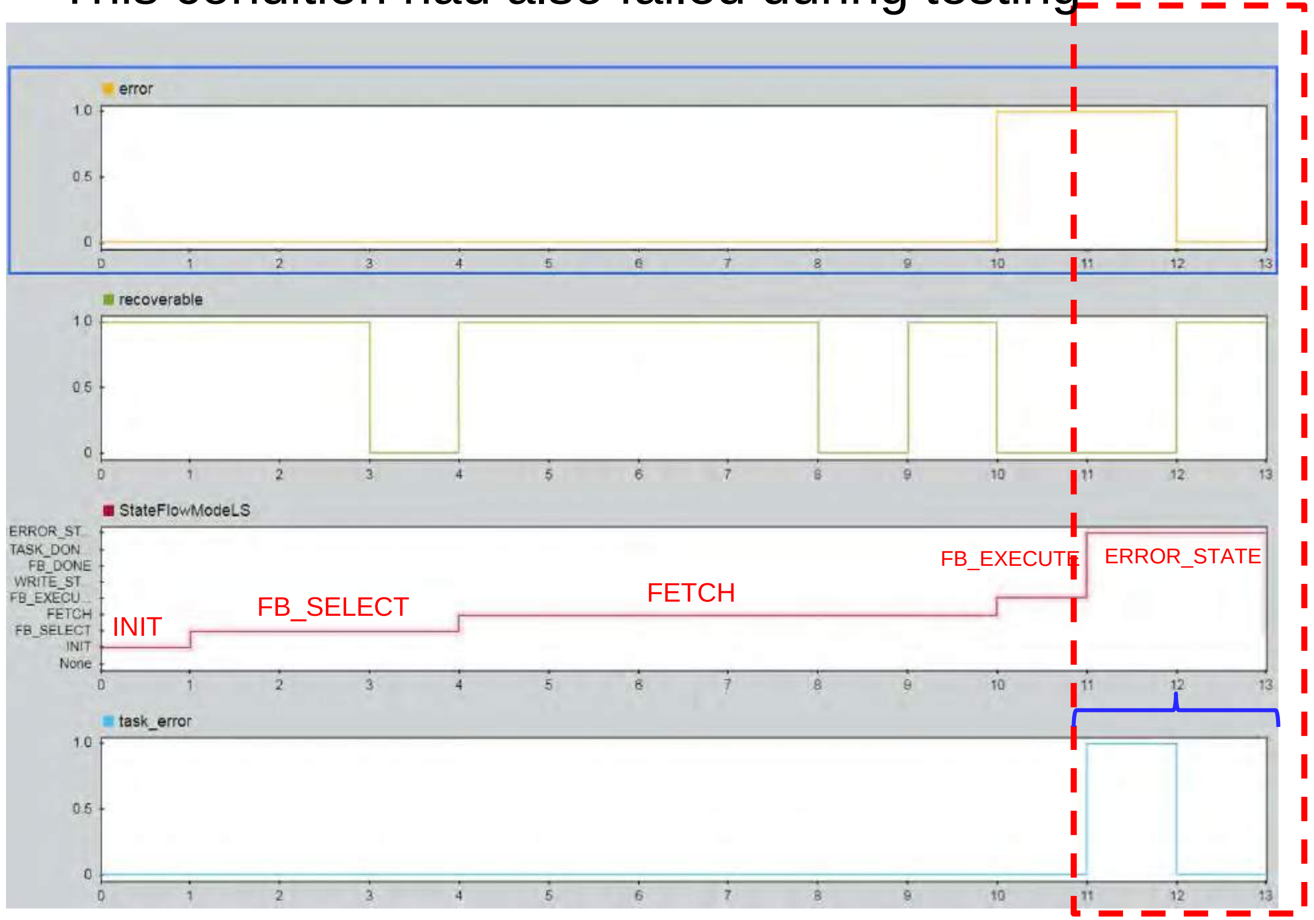
ABORT_CTRL_HI
%set control, abort, and init
task_done = false;
task_error = true;
write = false;
ctrl = true;
abort = true;
load_counter = false;
increment = false;
latch_inst_start = false;
latch_addr = false;
latch_function = false;
reset = false;

VCU College of Engineering

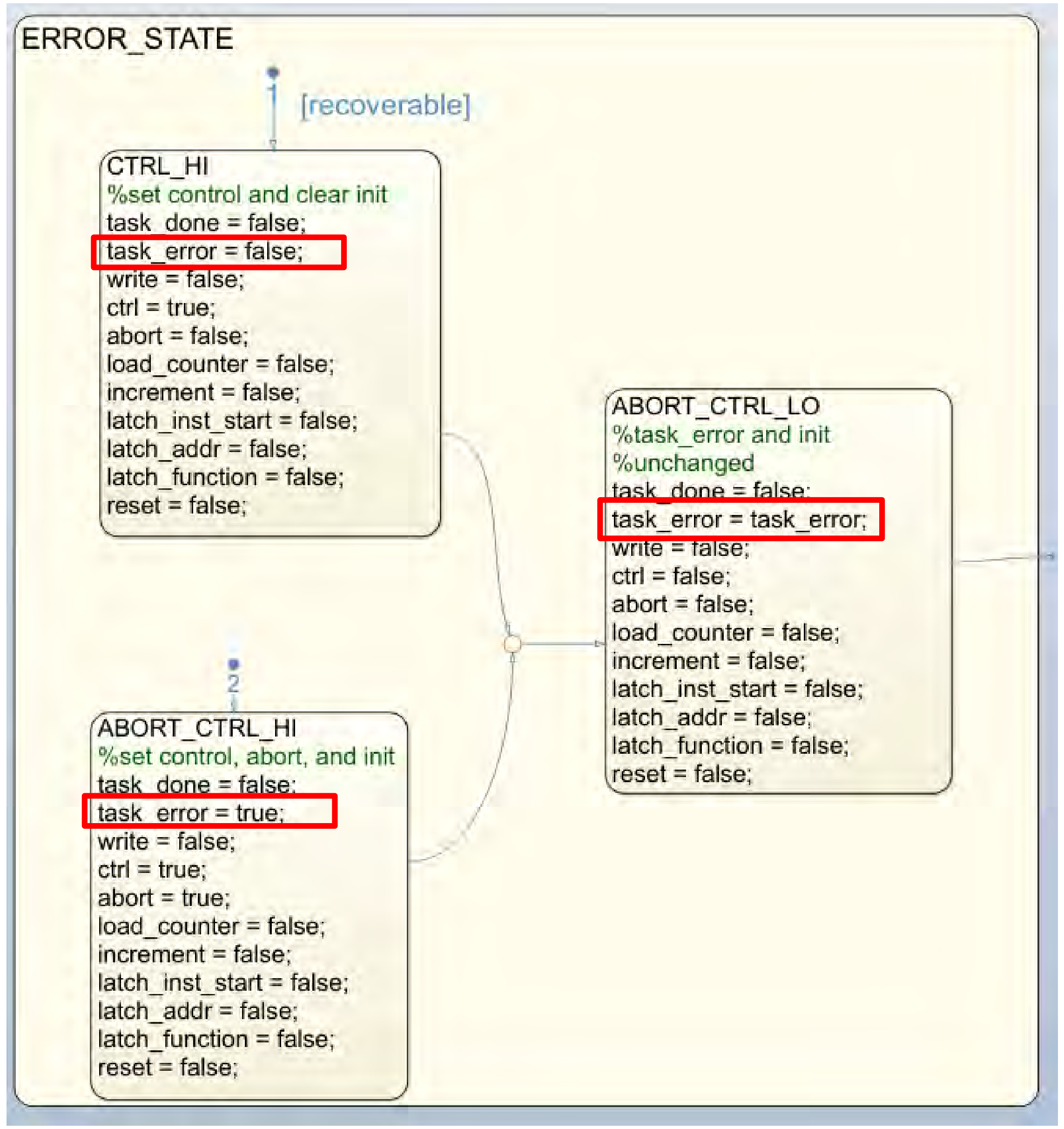
EPRI ELECTRIC POWER RESEARCH INSTITUTE

Counterexample

- Example of synergism between testing and formal verification.
- Inconsistent values in task_error
- This condition had also failed during testing



Task error changes



```
CTRL_HI
%set control and clear init
task_done = false;
task_error = false;
write = false;
ctrl = true;
abort = false;
load_counter = false;
increment = false;
latch_inst_start = false;
latch_addr = false;
latch_function = false;
reset = false;
```

```
ABORT_CTRL_LO
%task_error and init
%unchanged
task_done = false;
task_error = task_error;
write = false;
ctrl = false;
abort = false;
load_counter = false;
increment = false;
latch_inst_start = false;
latch_addr = false;
latch_function = false;
reset = false;
```

```
ABORT_CTRL_HI
%set control, abort, and init
task_done = false;
task_error = true;
write = false;
ctrl = true;
abort = true;
load_counter = false;
increment = false;
latch_inst_start = false;
latch_addr = false;
latch_function = false;
reset = false;
```

Synergism between testing and formal verification

A key insight on more effective use of formal verification:

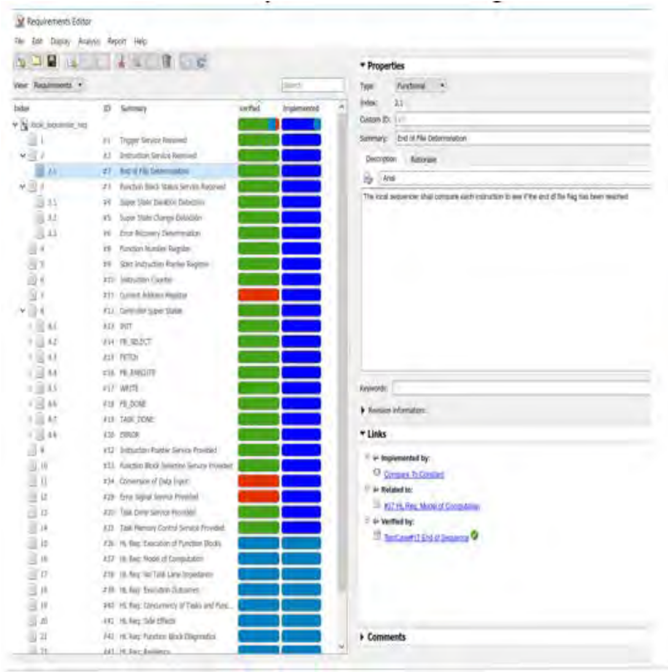
- Do functional testing first to inform formal methods: more subtle complex errors
- This insight was confirmed or validated by discussions with the formal methods engineering group at NASA.

VCU College of Engineering

EPRI ELECTRIC POWER RESEARCH INSTITUTE

End-to-end Traceability

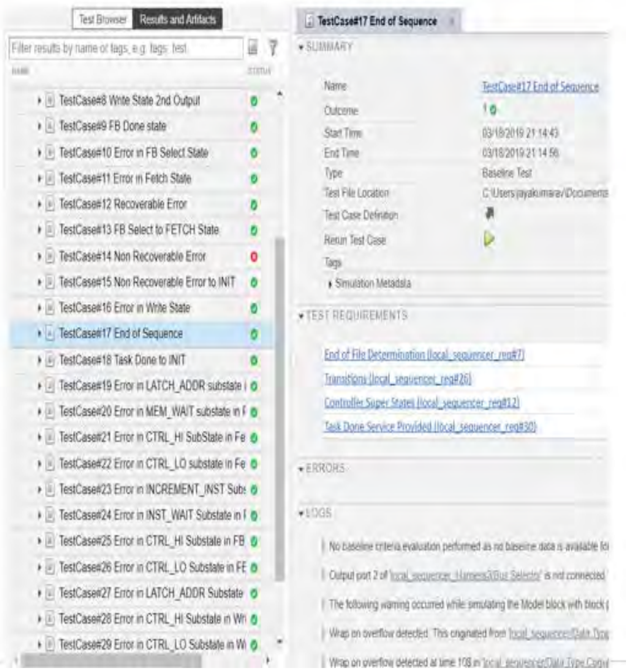
- SymPLe is verified at every incremental level including the Simulink model, generated HDL code, and hardware implementation
- MBD offers traceability at each of these incremental levels thereby offering a chain of evidence to support verification.



Index	ID	Summary	Verified	Implemented
local_sequencer_req			🟩	🟦
1	#1	Trigger Service Received	🟩	🟦
2	#2	Instruction Service Received	🟩	🟦
2.1	#7	End of File Determination	🟩	🟦
3	#3	Function Block Status Service Received	🟩	🟦
3.1	#4	Super State Duration Detection	🟩	🟦
3.2	#5	Super State Change Detection	🟩	🟦
3.3	#6	Error Recovery Determination	🟩	🟦
4	#8	Function Number Register	🟩	🟦
5	#9	Start Instruction Pointer Register	🟩	🟦
6	#10	Instruction Counter	🟩	🟦
7	#11	Current Address Register	🟥	🟦
8	#12	Controller Super States	🟩	🟦
8.1	#13	INIT	🟩	🟦
8.2	#14	FB_SELECT	🟩	🟦
8.3	#15	FETCH	🟩	🟦
8.4	#16	FB_EXECUTE	🟩	🟦
8.5	#17	WRITE	🟩	🟦
8.6	#18	FB_DONE	🟩	🟦
8.7	#19	TASK_DONE	🟩	🟦
8.8	#20	ERROR	🟩	🟦
9	#32	Instruction Pointer Service Provided	🟩	🟦
10	#33	Function Block Selection Service Provided	🟩	🟦
11	#34	Conversion of Data Input	🟥	🟦
12	#29	Error Signal Service Provided	🟥	🟦
13	#30	Task Done Service Provided	🟩	🟦
14	#31	Task Memory Control Service Provided	🟩	🟦
15	#36	HL Req: Execution of Function Blocks	🟦	🟦
16	#37	HL Req: Model of Computation	🟦	🟦
17	#38	HL Req: Val Task Lane Impedance	🟦	🟦
18	#39	HL Req: Execution Outcomes	🟦	🟦
19	#40	HL Req: Concurrency of Tasks and Fun...	🟦	🟦
20	#41	HL Req: Side Effects	🟦	🟦
21	#42	HL Req: Function Block Diagnostics	🟦	🟦
22	#43	HL Req: Resiliency	🟦	🟦

Properties
Type: Functional
Index: 2.1
Custom ID:
Summary: End of File Determination
Description | Rationale
Arial
The local sequencer shall compare each instruction to see if the end of file flag has been reached

Keywords:
Revision information:
Links
Implemented by:
Compare To Constant
Related to:
#37 HL Req: Model of Computation
Verified by:
TestCase#17 End of Sequence ✓
Comments



NAME	STATUS
TestCase#8 Write State 2nd Output	●
TestCase#9 FB Done state	●
TestCase#10 Error in FB Select State	●
TestCase#11 Error in Fetch State	●
TestCase#12 Recoverable Error	●
TestCase#13 FB Select to FETCH State	●
TestCase#14 Non Recoverable Error	●
TestCase#15 Non Recoverable Error to INIT	●
TestCase#16 Error in Write State	●
TestCase#17 End of Sequence	●
TestCase#18 Task Done to INIT	●
TestCase#19 Error in LATCH_ADDR substate i	●
TestCase#20 Error in MEM_WAIT substate in F	●
TestCase#21 Error in CTRL_HI SubState in Fe	●
TestCase#22 Error in CTRL_LO substate in Fe	●
TestCase#23 Error in INCREMENT_INST Subs	●
TestCase#24 Error in INST_WAIT Substate in F	●
TestCase#25 Error in CTRL_HI Substate in FB	●
TestCase#26 Error in CTRL_LO Substate in FE	●
TestCase#27 Error in LATCH_ADDR Substate	●
TestCase#28 Error in CTRL_HI Substate in Wr	●
TestCase#29 Error in CTRL_LO Substate in Wi	●

TestCase#17 End of Sequence

SUMMARY

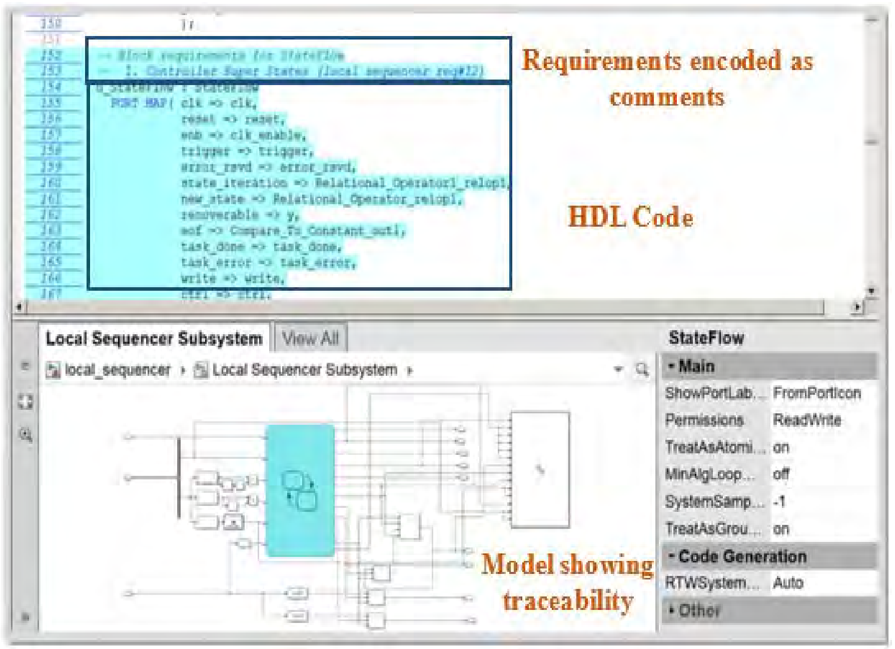
Name	TestCase#17 End of Sequence
Outcome	1 ●
Start Time	03/18/2019 21:14:43
End Time	03/18/2019 21:14:56
Type	Baseline Test
Test File Location	C:\Users\jayakumara\Documents
Test Case Definition	
Rerun Test Case	
Tags	

Simulation Metadata

TEST REQUIREMENTS
End of File Determination (local_sequencer_req#7)
Transitions (local_sequencer_req#26)
Controller Super States (local_sequencer_req#12)
Task Done Service Provided (local_sequencer_req#30)

ERRORS

LOGS
No baseline criteria evaluation performed as no baseline data is available fo
Output port 2 of 'local_sequencer_Harness3/Bus Selector' is not connected
The following warning occurred while simulating the Model block with block (
Wrap on overflow detected. This originated from 'local_sequencer/Data Typ
Wrap on overflow detected at time 108 in 'local_sequencer/Data Type Conv


Requirements encoded as comments
HDL Code
Local Sequencer Subsystem
View All
local_sequencer
Local Sequencer Subsystem
Model showing traceability
StateFlow

```
  );
-- Block requirements for StateFlow
--   1. Controller Super States (local_sequencer_req#12)
u_StateFlow : StateFlow
  PORT MAP( clk => clk,
            reset => reset,
            enb => clk_enable,
            trigger => trigger,
            error_rsvd => error_rsvd,
            state_iteration => Relational_Operator1_relop1,
            new_state => Relational_Operator_relop1,
            recoverable => y,
            eof => Compare_To_Constant_out1,
            task_done => task_done,
            task_error => task_error,
            write => write,
            ctrl => ctrl,
```

Main

ShowPortLab...	FromPortIcon
Permissions	ReadWrite
TreatAsAtomi...	on
MinAlgLoop...	off
SystemSamp...	-1
TreatAsGrou...	on

Code Generation

RTWSystem...	Auto

Other

Bidirectional Traceability between Requirements, test cases, properties , Model and HDL code

Findings and Results of the Model Based V&V

- 24 design faults found during V&V.
- Fig shows the V&V stage where the faults were first identified. The faults were classified into low, medium and high impact faults based on severity.

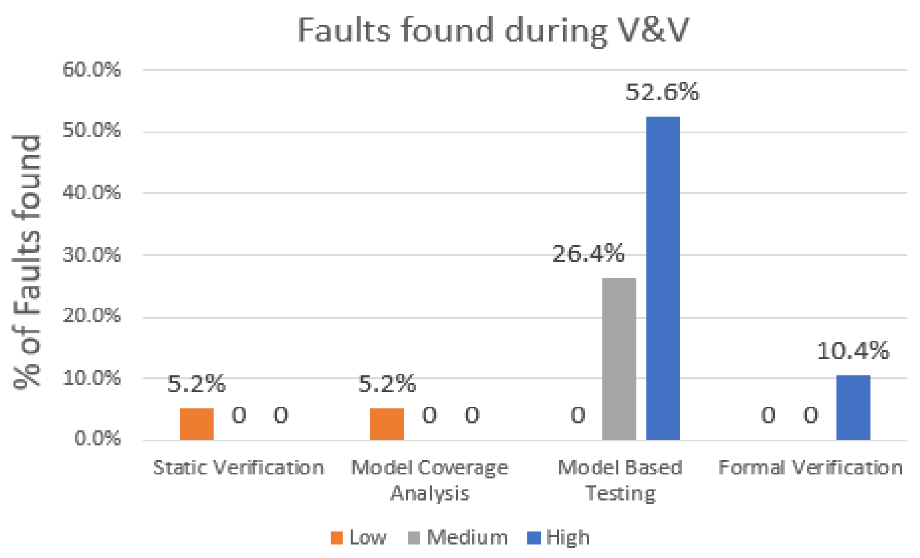
- 78% of the faults were first identified during model based testing

- High severity faults found during formal verification

- Synergy between testing and formal verification
  - o helped find subtle and hard to detect bugs in corner case scenarios that could be overlooked during testing.
  - o Added strong confirmatory evidence that formal methods paired with testing is effective in detecting difficult SCCFs.

- Traceability between requirements, model, code and proofs provide a effective verification environment and support IEC 61508 compliance.

- Constrained and complexity aware principles and design requirements of SymPLe significantly improved the V&V efforts and time spent on verification.



Faults found in SymPLe architecture by V&V process

Phase II of SymPLe

- Phase II will focus on synergistically integrating security and safety into SymPLe.
- We did not address security in Phase I.
- Phase II starting fall and winter of 2019.
- Specifically,
  - Constrained architectures (like SymPLe) provide a technical and pragmatic foundation for establishing very small attack surface devices.
  - Developing and implementing primitive security building blocks from PUFs (Physically Unclonable Functions) for SymPLe will provide a foundation for secure services against sophisticated life cycle attacks.
    - Equipment hijack/tampering
    - design/IP theft
    - data corruption/theft
    - Fingerprinting computations
  - Raising the TRL for SymPLe.
- For phase II we have teamed with Dr. Patrick Schaumont of ECE department at VA tech, Mathworks and with OneSpin solutions (USA).
- Phase II sponsored by Idaho National Labs and EPRI.

Where to find our work

- Report Publications
- ERPI report – free to public
  - https://www.epri.com/#/pages/product/000000003002015754/?lang=en-US
- US Department of Energy ASI (soon to be released) or email me..
  - Matt Gibson and Carl Elks, *Main Achieving Verifiable and High Integrity Instrumentation and Control Systems through Complexity Awareness and Constrained Design*, **Final Technical Report,** M2CA-15-CA-EPRI-0703-0221, NEET-2 Project No. 15-8044 2019.
- Journal and Conference Publications:
- Near Submission: IEEE Transactions on SW Engineering, *Model Based Design and Assurance of a I&C Architecture for Safety Critical Nuclear Power Applications*.
- In preparation: SymPLe: *High Integrity Instrumentation and Control Systems through Complexity Awareness and Constrained Design*, IEEE Transactions on Nuclear Science.
- In preparation: *Lessons Learned from a Model Based Verification and Validation effort for IEC 61508,* Dependable Systems and Networks Conference 2020.
- *ANS NPIC – In progress.*
- ***Welcome to visit our Lab at VCU and see SymPLe in action.***

Discussion

VCU College of Engineering

EPRI ELECTRIC POWER RESEARCH INSTITUTE

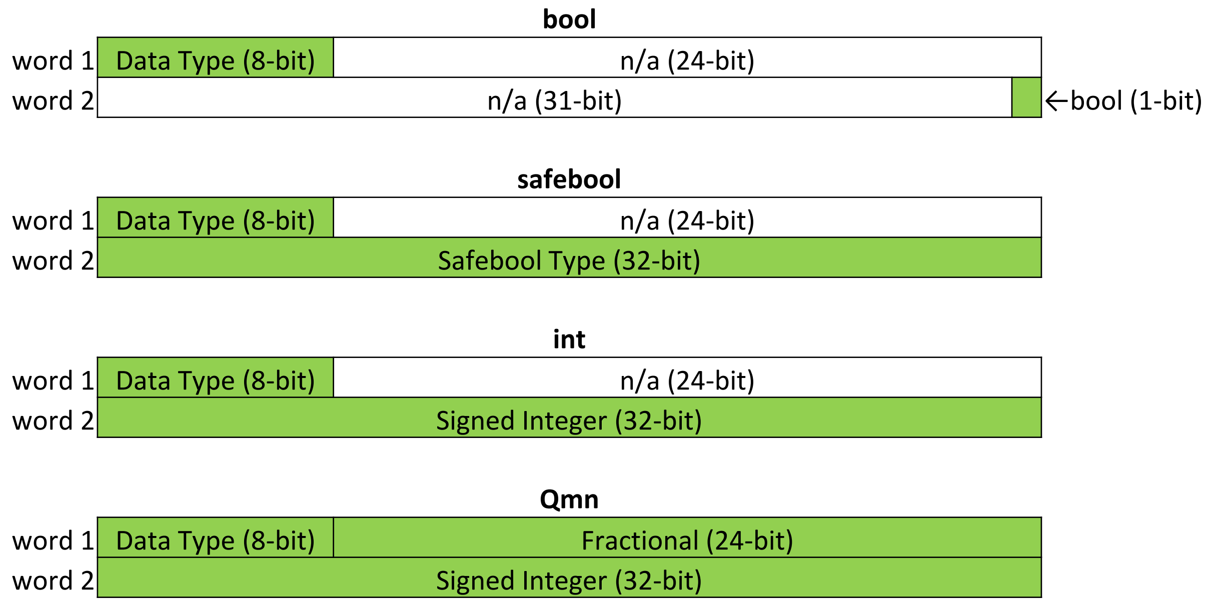
Function Block Error Detection

- Monitors SymPLe execution errors and Logic execution errors
- *FB Controller manages execution recovery for transient errors
- SymPLe Execution Errors
  - Function Block Execution Timeout*
  - Input Register Read Overflow
  - Output Register Write Overflow
  - Data Type Error*
- Logic Execution Errors
  - Duplex Divergence Error (single-event upset)*
  - Arithmetic Overflow
  - Arithmetic Underflow
  - Arithmetic Divide-By-0
- Logic execution errors are unique to each functionality

Error Type	Variable Name	Value	Binary Value
Duplex Divergence Error	stateErrorBit	1	0b1
Function Block Execution Timeout	timeoutBit	2	0b10
Arithmetic Overflow	overflowBit	4	0b100
Arithmetic Underflow	underflowBit	8	0b1000
Arithmetic Division by 0	divideByZeroBit	16	0b10000
Input Register Read Overflow	inputOverflowBit	32	0b100000
Output Register Write Overflow	outputOverflowBit	64	0b1000000
Data Type Error	typeBit	256	0b100000000

SymPLe Data Types

- All SymPLe data transmitted as two int32 words
- SymPLe utilizes type-masking for standard data communication protocol
- Simulink type inheritance allows for easy type modification and future work



SymPLe Types	Description	Min	Max
Boolean	1-bit logical	0	1
Safebool	32-bit logical	01010101010101010101010101010101	10101010101010101010101010101010
Integer	int32	-2147483648	2147483647
Qmn	Fixed point (32.24 prec)	-2147483648	2147483648