PHILIPS

Assertion-based Verification for the SpaceCAKE Multiprocessor
– A Case Study

Milind Kulkarni
Benita Bommi J
Philips Research India - Bangalore

Outline

- Motivation
- Design Overview
- Verification Approaches
- Assertion based Approach
  - Assertion-based Static Verification
  - Assertion-based Dynamic Verification
- Test setup
- Results/Observations
- Conclusion

Motivation

- Complex multiprocessor design
  - Interdependency
- Many design refinements
  - Feature additions/change management
  - Synthesis targets
- Small design and verification teams
  - Maximize RC

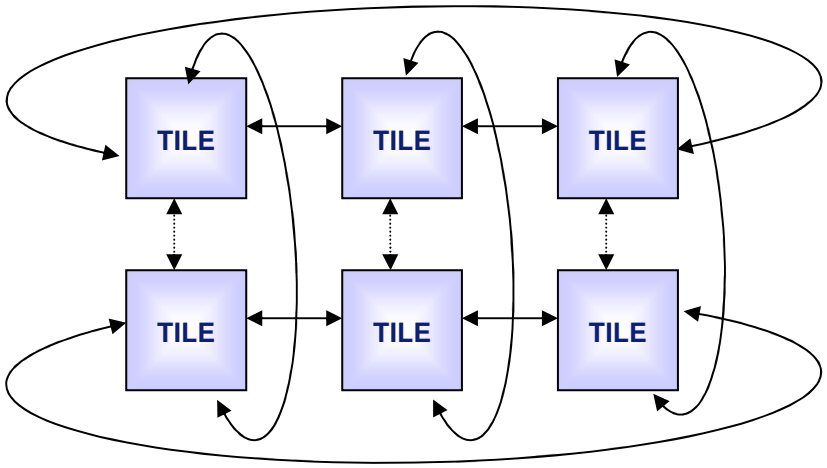
Interdependency, deadlocks, race conditions, re-orderings !!

Testbench refinements !

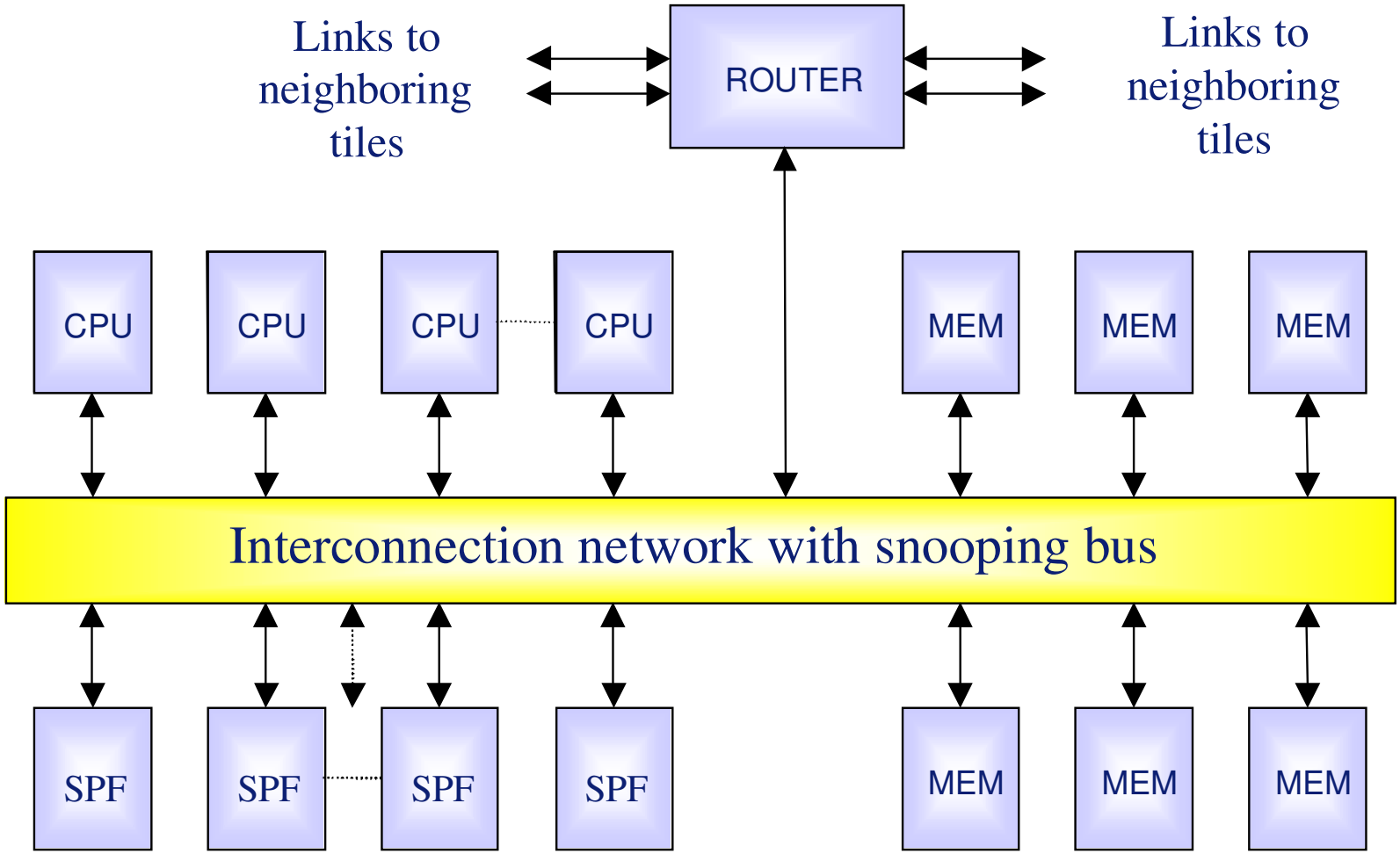
Verification flow and methodology !

Design Overview

- Distributed Shared Memory Architecture
- L1 and L2 cache coherency
- Sequential Consistency
- Shared resource reservation framework
- Software controlled cache operations and debug functionality
- Multiple outstanding requests with out-of-order processing



Design Overview (contd…)



Memory consistency and coherency issues

Coherency:

"Reads return the latest written value on same address."

	CPU1	CPU2
T=0	A=1;	
T=1		B=A; //B should be 1

Not easy:
CPU rd/wr ops are not atomic. SoCs have no global unique time.

Consistency:

"Observe proper ordering between different addresses."

	CPU1	CPU2
T=0	D=0; //data	
T=1	F=0; // flag	
T=2	D=1;	
T=3	F=1;	while (!F);
		C=D; // C should be 1

Not easy:
Different addresses can map to different locations in the architecture, with different access delays.

Problem Description

- Architecture Complexity
  - Complex SoC with high degree of configurability
  - Two levels of cache
  - Complex cache coherence protocols
  - High degree of concurrency
  - Integrated Intellectual Property
  - Distributed Shared Memory

- Dilemma
  - Module-level verification - time consuming
  - Top-level verification - hidden bugs

Identify and implement proficient verification techniques to verify complex SpaceCAKE design

Verification Methodology

- Identify verification hot – spots
- Introduce assertion - based verification
- Verify the design through the use of dynamic verification
- Simulate the design using different configurations
- Use functional coverage and code coverage metrics
- Improvise tests to get better coverage
- Verify the design statically
  - Sub-modules with queues, pipelines, smaller FSMs

Identifying Hot Spots

- Hot spots in design structures
  - Queues – storing transactions
  - Buffers – data storage for out-of-order response
  - Memories – shared L2 cache
  - Arbiters – access to shared resources (L2, datapath)
  - FSMs – distributed controllers
  - Pipelines – overlapping computations with data transfers
- Hot spots in interface protocols
  - Standard interface standards (AXI and MTL)
    - Also used between modules for data transfer
    - Stable over the design cycle

Identifying Hot Spots (Contd…)

- Hot spots in interfaces between modules
  - Specific defined handshake protocol and signal definitions
  - Combinatorial paths for inter-module status updates
- Hot spots in protocols and policies specific to the design
  - Sequential Consistency
  - Memory coherency
  - Out-of-order processing
  - Various system configurations
    - 2x2 Tiles with Tile[0,0] and Tile[0,1] having shared memory configuration and allocate-write L2 caching policy and …..
  - Software driven cache operations

Assertion Types

- Assumptions
  - Not supported features
  - No X's on signal group upon valid "High"
  - Input/response for wrong transaction
  - Unexpected input being in particular state
  - Response time window
  - Reveal integration issues
- Constraints
  - Valid signal values
  - Handshake mechanisms
  - Useful for static verification

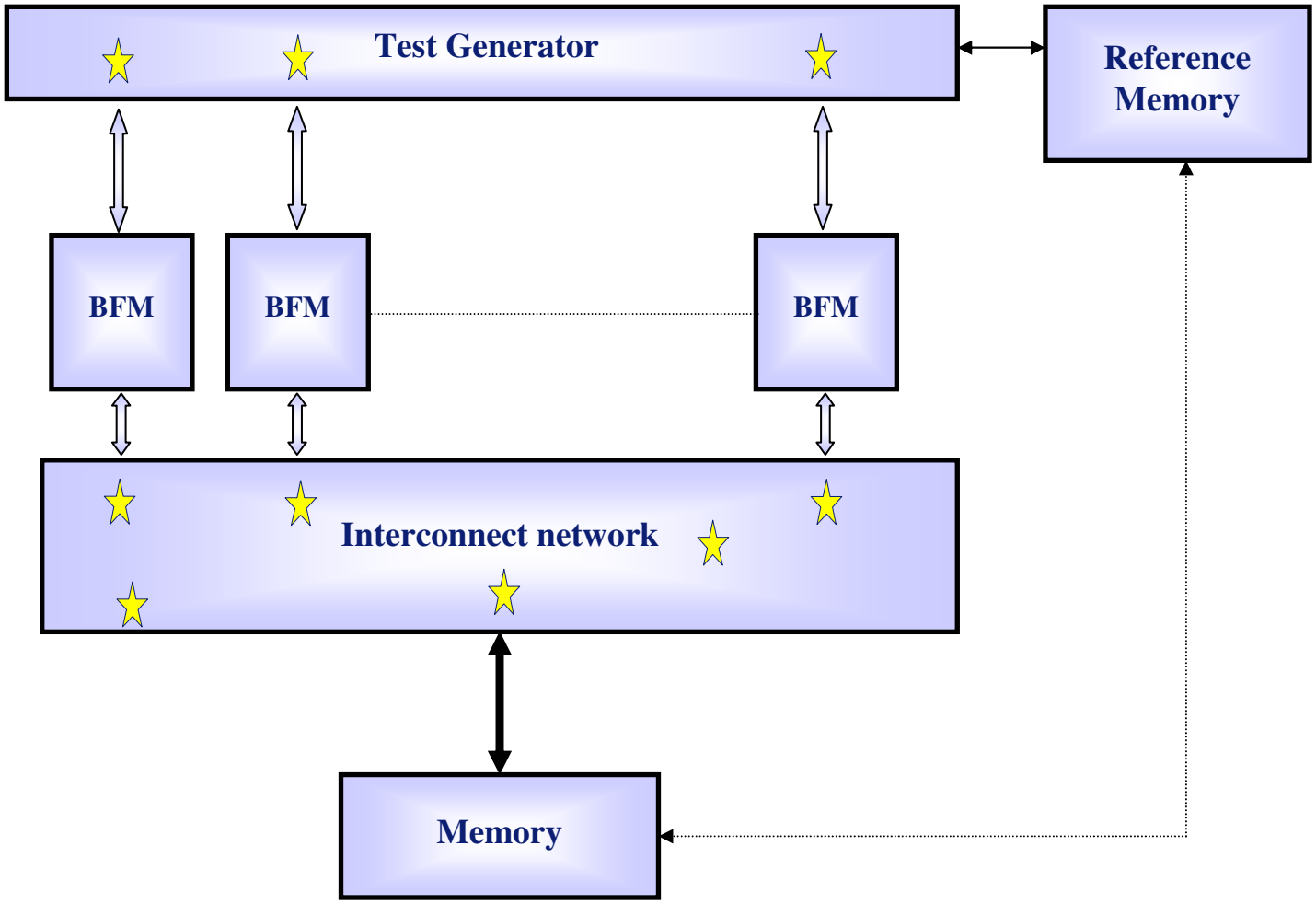
Assertion Types (Contd…)

- Coverage
  - Test coverage
  - Stress effectiveness
  - Effects of different configuration
- Functional
  - Self checking tests
  - Design specific functionality checks e.g. priority queue, random victimization, reservations
- Implementation
  - Assertions implemented as either in OVL or PSL

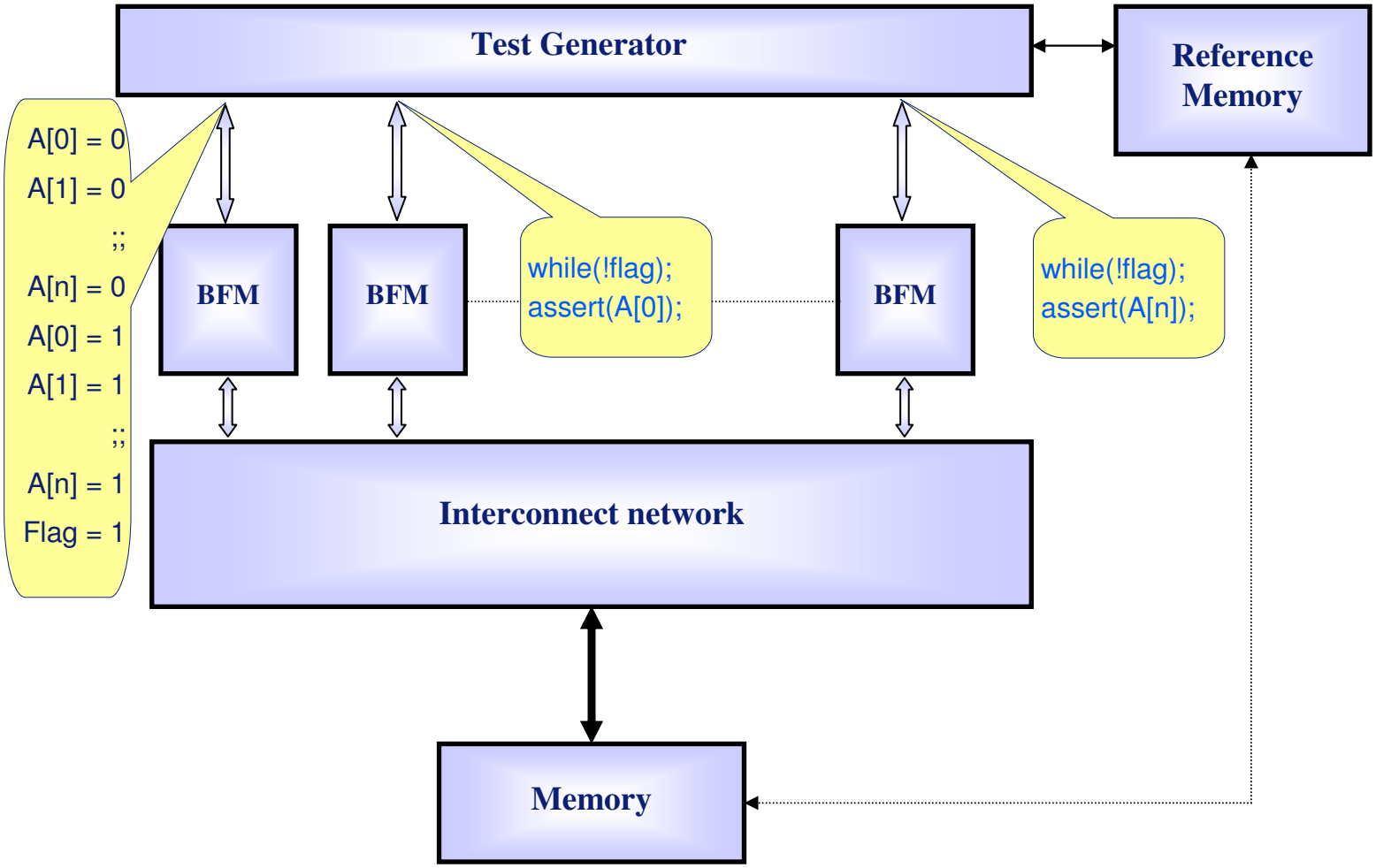
Static Verification of Assertions

- Features
  - Cadence staticverilog tool was used for static verification
  - No requirement to generate testbenches
  - Applied to sub-modules with smaller FSMs, queues and pipelines
  - 10% assertions completely verified to hold
- Drawbacks
  - Lack of support for verilog 2001 constructs which were used in the design
  - Lot of effort to constrain the inputs

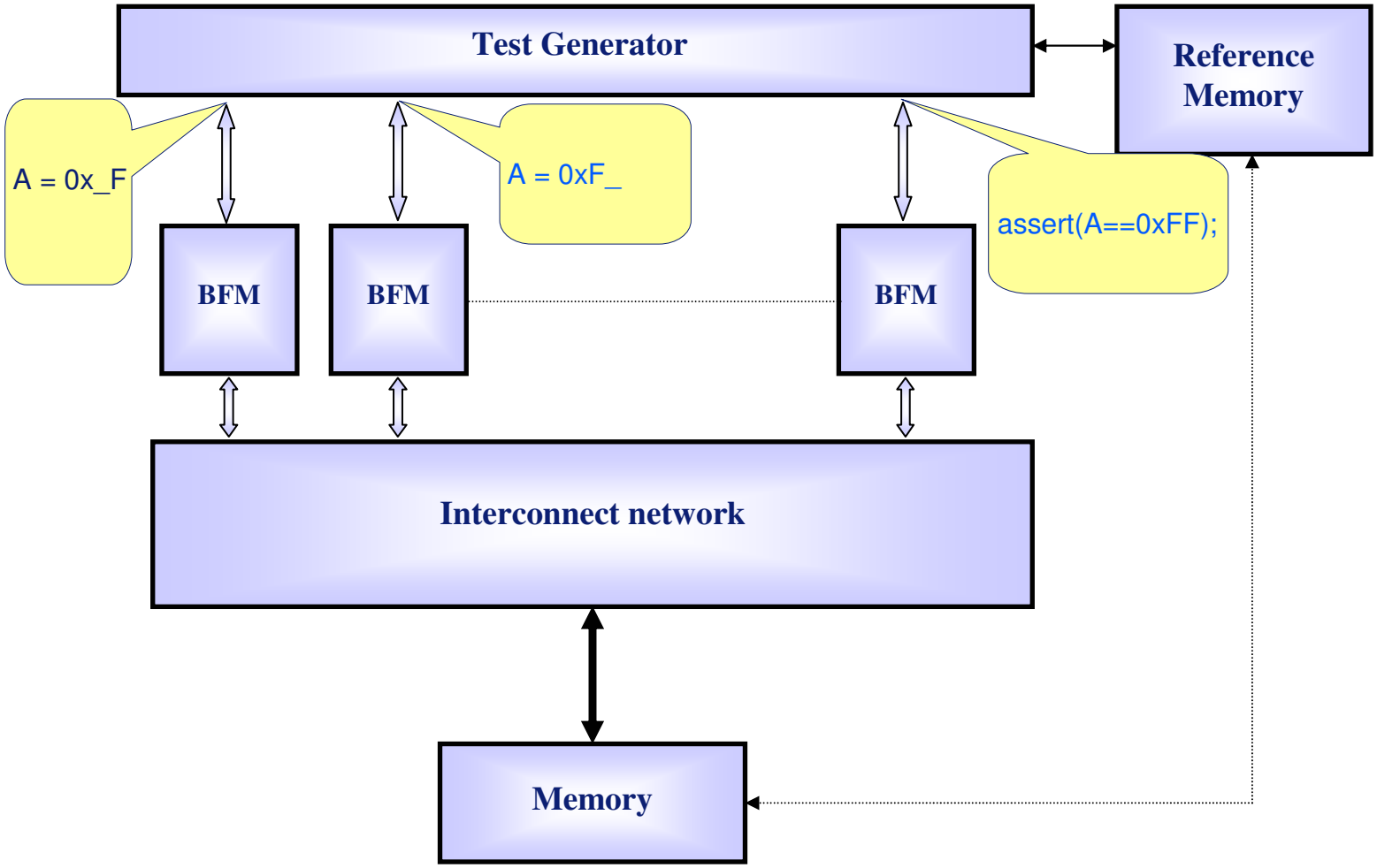
Test Setup for Dynamic Verification



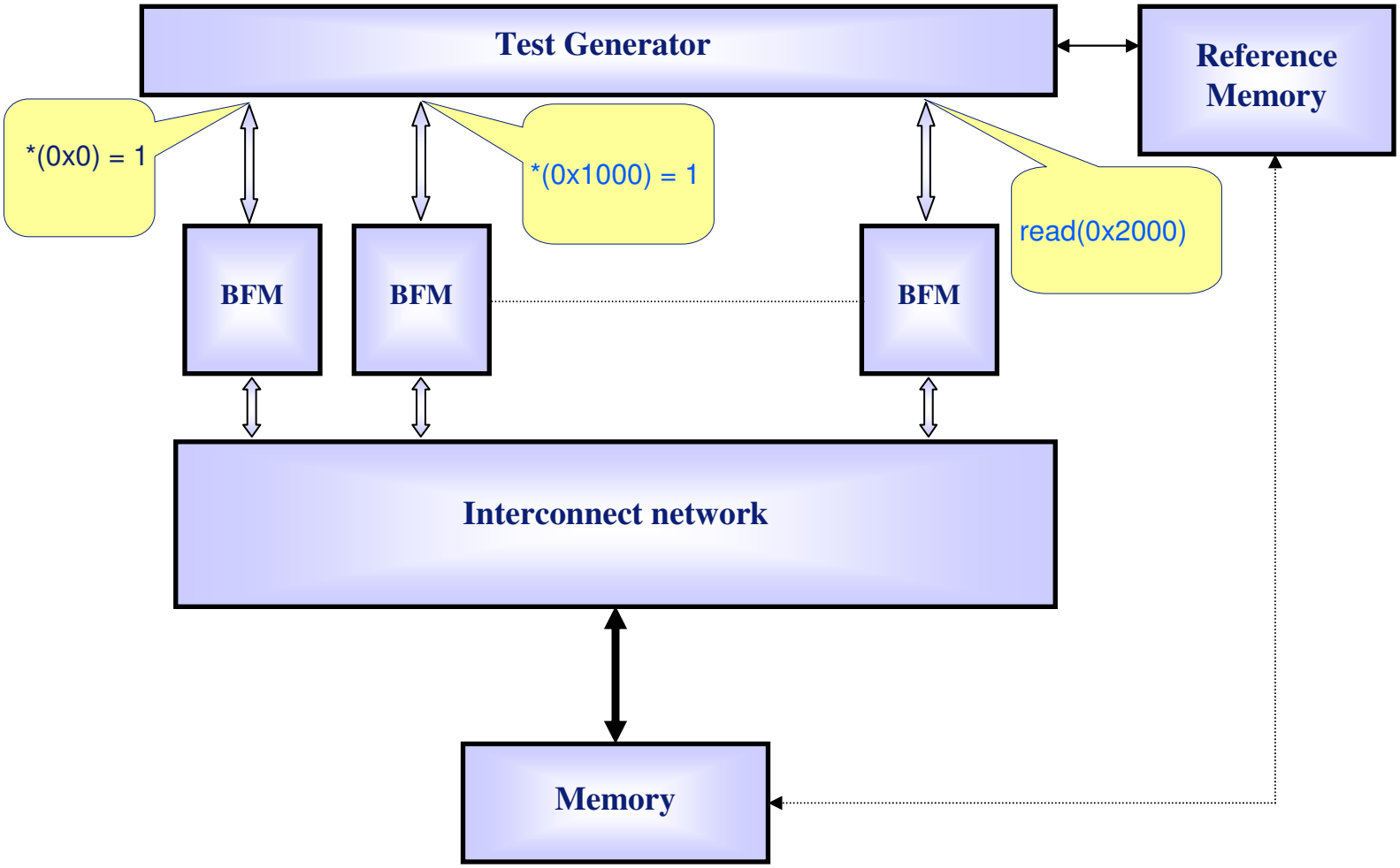
Test setup (Sequential Consistency)



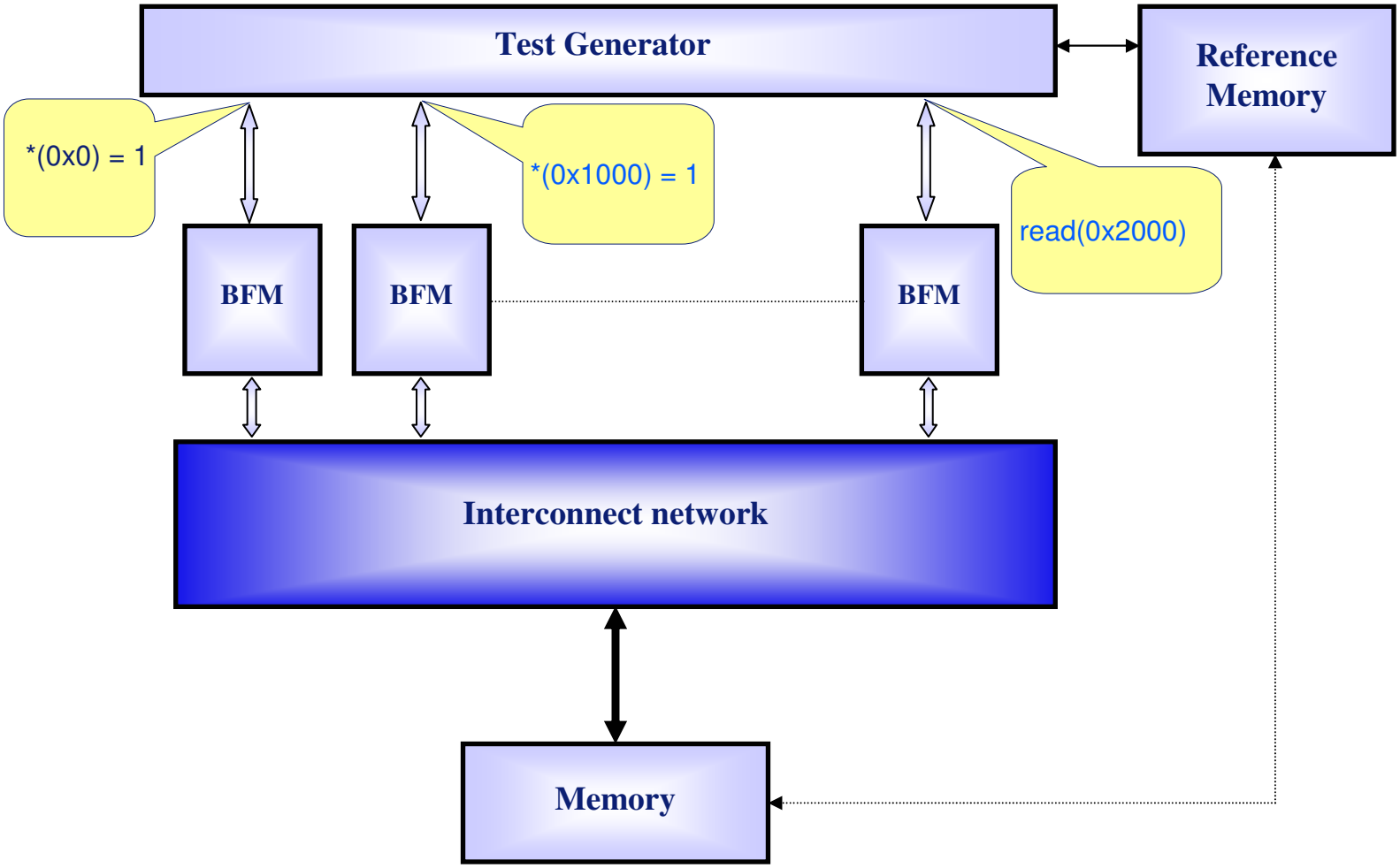
Test setup (Memory Coherency)



Test setup (L2 Cache)



Test setup (Constraining Design)



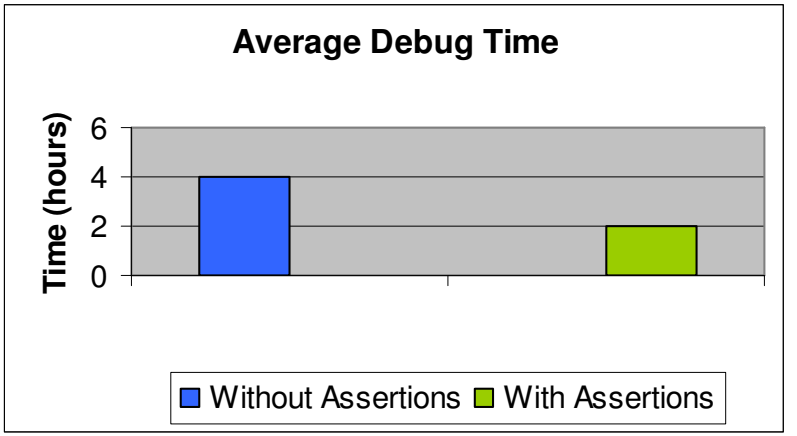
Pros and Cons of Assertion Based Approach

- Advantages
  - Facilitate explicit design specification
  - Supplement traditional verification
  - Capture design intent
  - Enhanced visibility
  - Provide multifaceted approach

- Disadvantages
  - Inconsistent design specifications and frequent design changes
  - False trigger of timing assertions e.g. detecting deadlocks
  - Different global level scenarios are not foreseen and hence assertions cannot be written beforehand
    - Verification engineer cannot be sure that the test has triggered different scenarios/deadlocks
    - Today's bug is tomorrow's assertion
  - Simulation time overhead

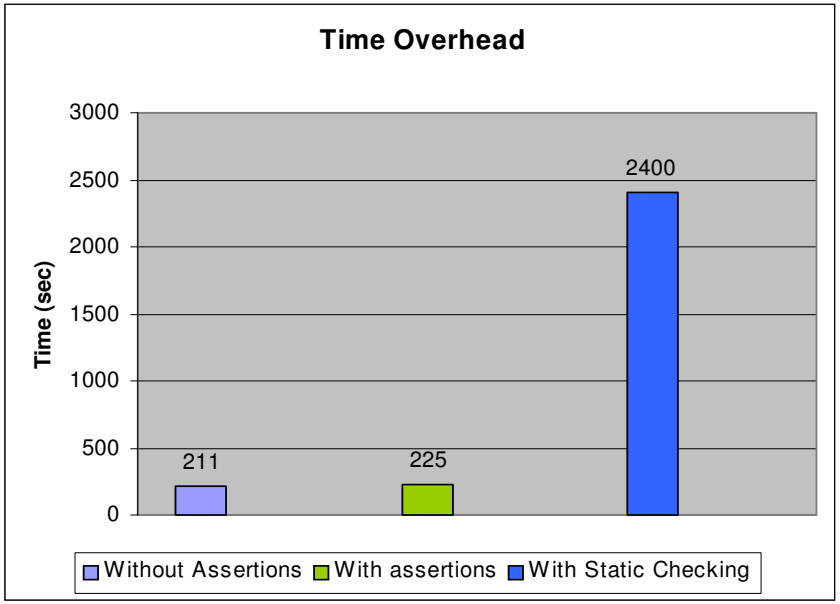
Results/Observations

- Increase in bug detection rate bugs - high visibility
- Different assertions useful at different phases in the design cycle.
- Remarkable reduction in debug time - upto 50%
- Increased scope to catch unexpected bugs – previously successful tests triggered assertions !!!
  - Transparent address translations
  - Speculative reads
  - Not enough stress created by the test



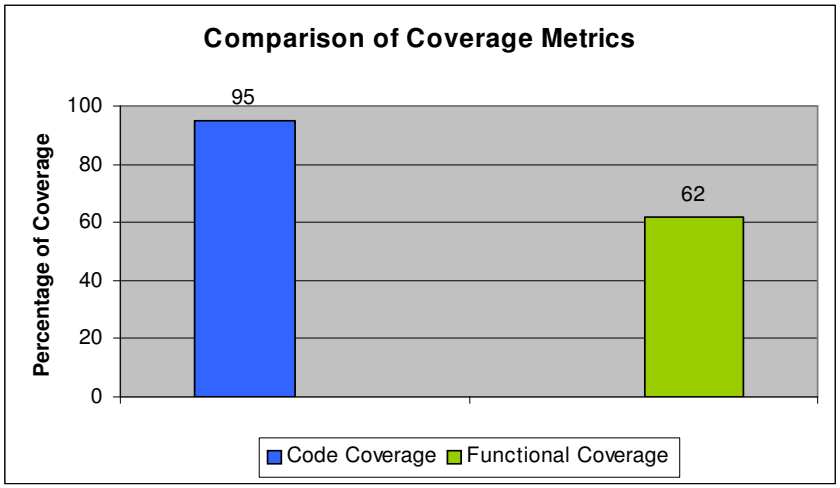
Results/Observations (Contd…)

- Minimal time overhead - 6% increase in simulation time
  - ~150 PSL + OVL assertions
- Density of assertions should be kept in check
- Increase concentration at the hot-spots
- 10% of assertions verified using static checking
- Increased overhead using static assertions



Results/Observations (Contd…)

- 62% functional coverage
- 95% code coverage for DSM functionality
- High code coverage with average functional coverage- indicates that code coverage cannot be relied upon totally



Conclusions

- Assertions enhance design visibility
- Assertion-based coverage driven technique is a cohesive, efficient methodology
- Decrease in verification time and effort
- Moving verification to the functional level, by focusing on protocol monitors,verification hot - spots and coverage metrics proved to be effective in verifying the SpaceCAKE design
- More matured static verification tools from EDA companies is very much needed


PHILIPS