

GSRC

GIGASCALE SYSTEMS RESEARCH CENTER

# The Future of EDA: Methodology, Tools and Solutions

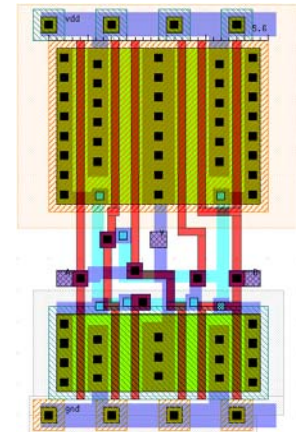
Sharad Malik  
Princeton University

NSF Future of EDA Workshop  
July 8-9, 2009

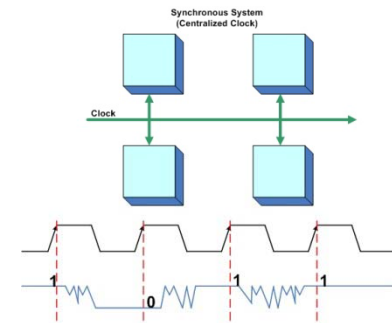


# Essence of EDA

- Tools follow methodology
- ASIC Design *Methodology*
  - Standard Cells
  - Synchronous TimingDefined sub-problems based on what needed to be solved, *and* what could be reasonably solved
- Tools support methodology
  - Provide
    - Design productivity
    - Design quality



Source: vlsitechnology.org



Source: chipdesignhome.com

# Design in the Late- and Post-Silicon Era

## Our Charter

- Enable Moore's Law
  - Reduce cost/unit-function
  - Functionality includes all aspects of design quality
    - power, performance, reliability, usability
  - Significant threats to all aspects of reducing cost and increasing functionality
    - Design verification and test
    - Staying within power budgets
    - Reliable designs on unreliable fabrics
    - Usability through efficient programmability

# Moore's Law and Design Verification

Moore's Law: Growth rate of transistors/IC is exponential

- Corollary 1: Growth rate of state bits/IC is exponential
- Corollary 2: Growth rate of state space (proxy for complexity) is doubly exponential

But...

- Corollary 3: Growth rate of compute power is exponential

Thus...

- Growth rate of complexity is still doubly exponential relative to our ability to deal with it

Design methodology must adapt to deal with this.

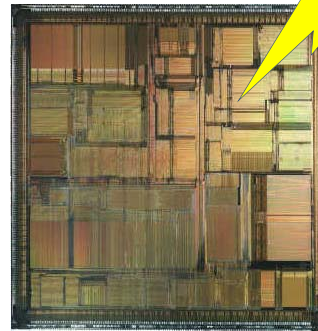
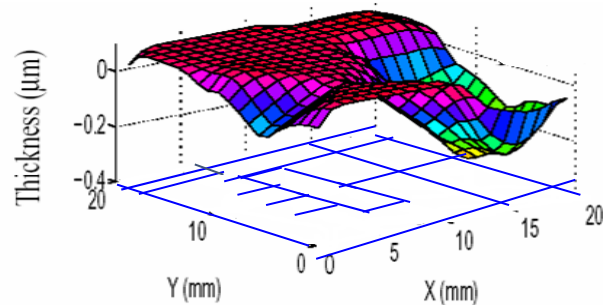
## Possible Solution Direction: Runtime Validation

- Increasingly need to reconcile ourselves to the fact that hardware like software will be shipped with bugs
- Runtime validation (through error detection and recovery) offers a potentially scalable solution
  - Provide robustness in the face of inevitable bug escapes
- Significantly reduce verification costs
  - Verify chips “to life” rather than “to death”

### Parametric Variability

*(Uncertainty in device and environment)*

Intra-die variations in ILD thickness



### Transient Faults due to Cosmic Rays & Alpha Particles

*(Increase exponentially with number of devices on chip)*

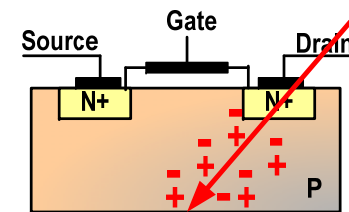


Figure Source: T. Austin

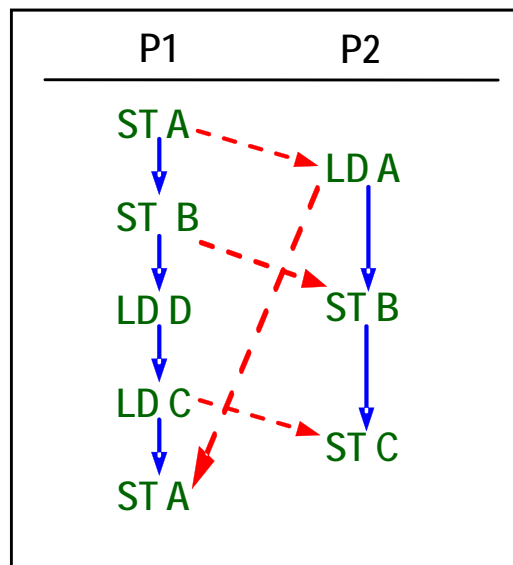
- Dynamic errors which occur at runtime
- Will need runtime solutions
- Combine with runtime solutions for functional errors (design bugs)

# Example: Checking Memory Consistency

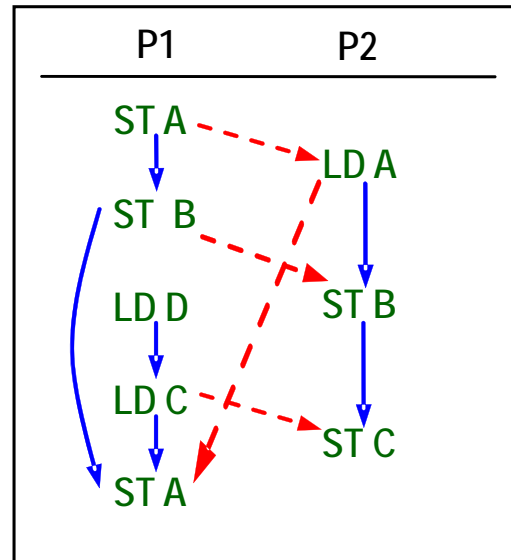
[D. Shasha *et al.*, TOPLAS'88] [H. W. Cain *et al.*, PACT'03]

- A directed graph that models memory ordering constraints
  - **Vertices**: dynamic memory instruction instances
  - **Edges**:
    - **Consistency edges**
    - **Dependence edges**

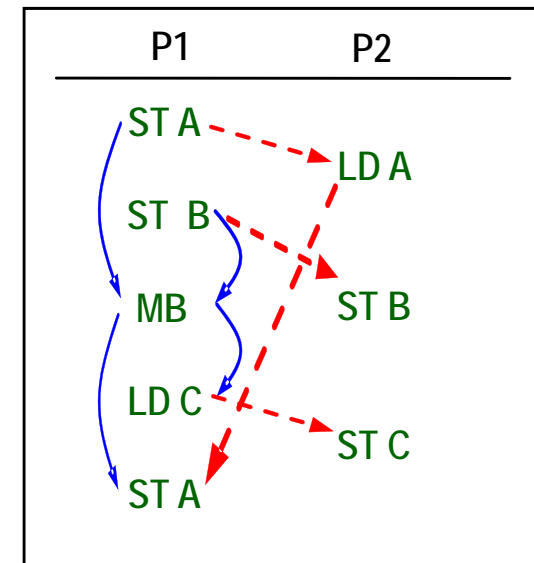
A cycle in the graph indicates a memory ordering violation



Sequential Consistency



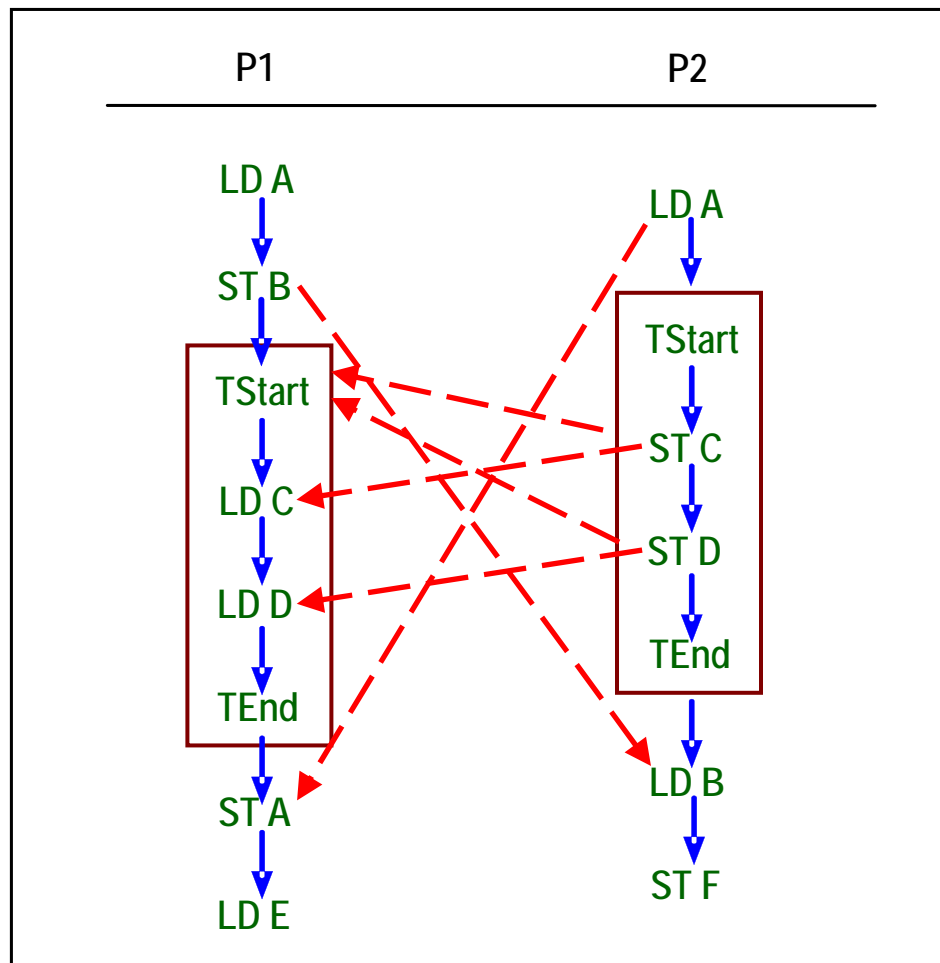
Total Store Ordering



Weak Ordering

# Extensions for Transactional Memory

- Extended constraint graph for transaction semantics
  - Non-transactional code assumes Sequential Consistency



## **TransOpOp:**

$$[Op1; Op2] \Rightarrow Op1 \leq Op2$$

## **TransMembar:**

$$Op1; [Op2] \Rightarrow Op1 \leq Op2$$

$$[Op1]; Op2 \Rightarrow Op1 \leq Op2$$

## **TransAtomicity:**

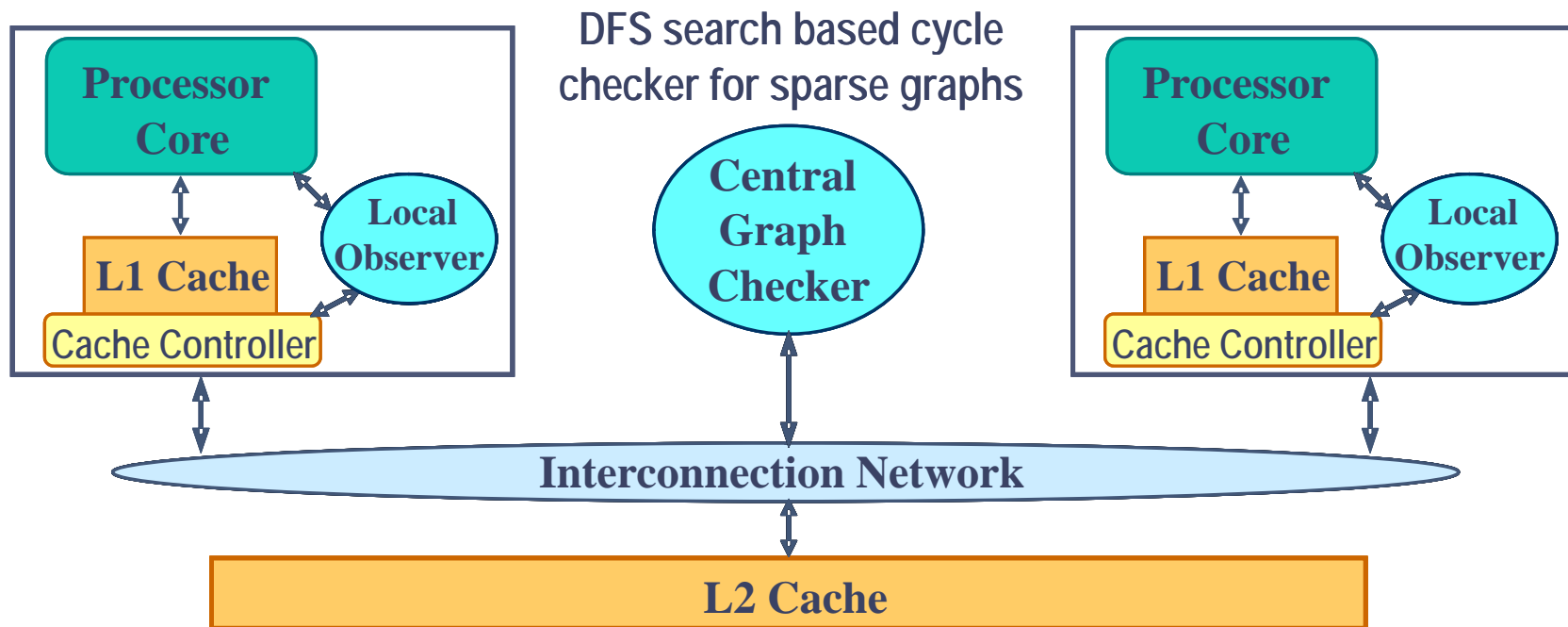
$$[Op1; Op2] \wedge \neg [Op1; Op; Op2]$$

$\Rightarrow$

$$(Op \leq Op1) \vee (Op2 \leq Op)$$

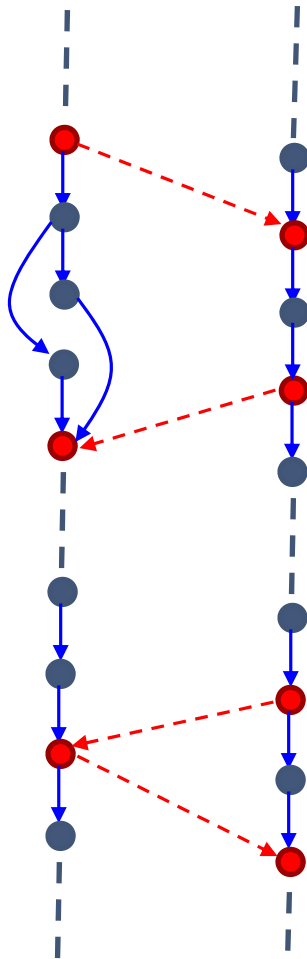


# On-the-fly Graph Checking



- **Local observer:**
- **Central checker:**
  - Local instruction ordering
  - Build the global constraint graph
  - Local access history
  - Check for the acyclic property
  - Locally observed inter-processor edges

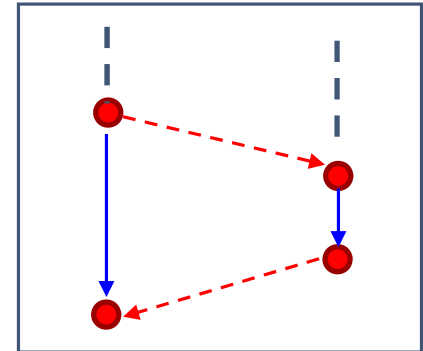
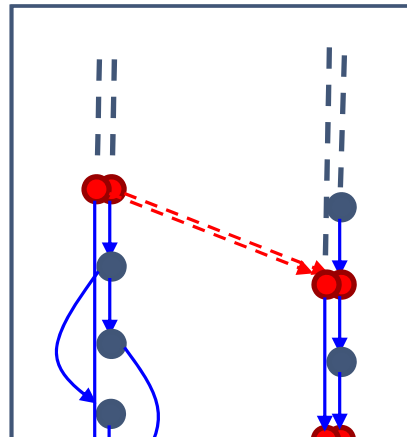
# Practical Design Challenges



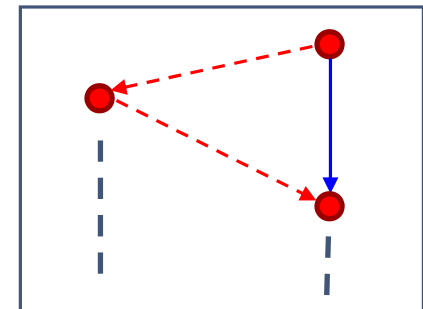
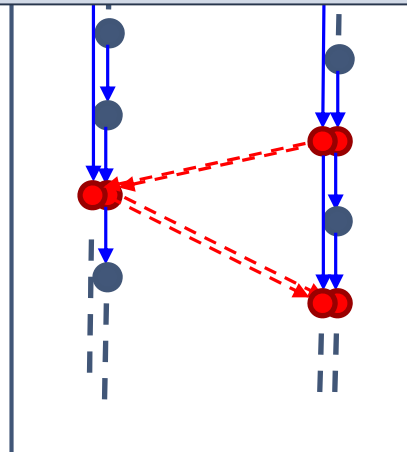
A naively built constraint graph that includes all executed memory instructions

- **Billions of vertices**
- **Unbounded graph size**

# Key Enabling Techniques



Enables checking of graphs of a few hundred vertices every 10K cycles



## Runtime Validation: Key Advantages

- Common framework for a range of defects
- Manage pre-silicon verification costs
  - Have *predictable* verification schedules
  - Support bug escapes through runtime validation
- Complexity, Performance Tradeoffs
  - Common mode
    - High performance, high complexity
  - (Infrequent) Recovery mode
    - Low complexity, low performance
- Leverage check-pointing support
  - Backward error recovery through rollback
  - Relevant for high-performance to support speculation

## Pre-Silicon vs. Runtime Validation

- Complementary Strengths
  - Large state space
    - Pre-silicon: Incomplete formal verification, simulation
    - Runtime: Easy - observe only actual state
  - State observability
    - Runtime: Challenging to observe
      - Distributed state, large number of variables
    - Pre-Silicon: Easy – just variables in software models for simulation or formal verification

# Future Challenges

- Keep costs low, with increasing complexity and failure modes
- A discipline for runtime validation?
  - Mature from one-off solutions to a general methodology
  - General checking and recovery mechanisms
    - Checking
      - Design assertions
    - Recovery
      - Generalized check-pointing and rollback
  - Analysis and synthesis tool support for the above