Abstracting and Verifying Flash Memories

Sandip Ray
University of Texas at Austin
sandip@cs.utexas.edu

Jayanta Bhadra
Freescale Semiconductor Inc.
jayanta.bhadra@freescale.com

Presentation for NVMTS 2008

Memory Verification

Specification:

- A state machine that reads and writes data at addressed locations.

Implementation:

- A network of transistors.

Verification Problem:

- **Does a given transistor network correctly implement the state machine above?**

Memory verification is a crucial component of a microprocessor or SoC verification.

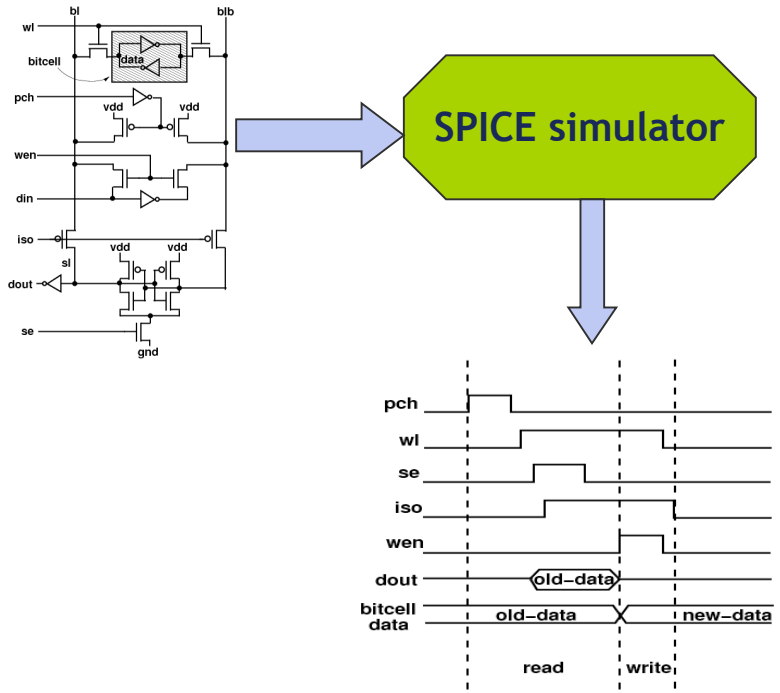
- Memories account for more than 50% of the transistor count of a typical general-purpose microprocessor.
- Custom-designed memories are complex artifacts with subtle and intricate behavior.

Current Practice in Memory Verification

Question 1:

Does a memory bitcell and its associated logic block perform according to specification?

- Analog (SPICE) simulation over a variety of process corners and operating conditions.
- Check transistor implementation against bitcell specifications.



Analog simulation is very detailed and accurate but can be performed only at the level of a single bitcell.

Too expensive to verify an entire memory arrays at this level.

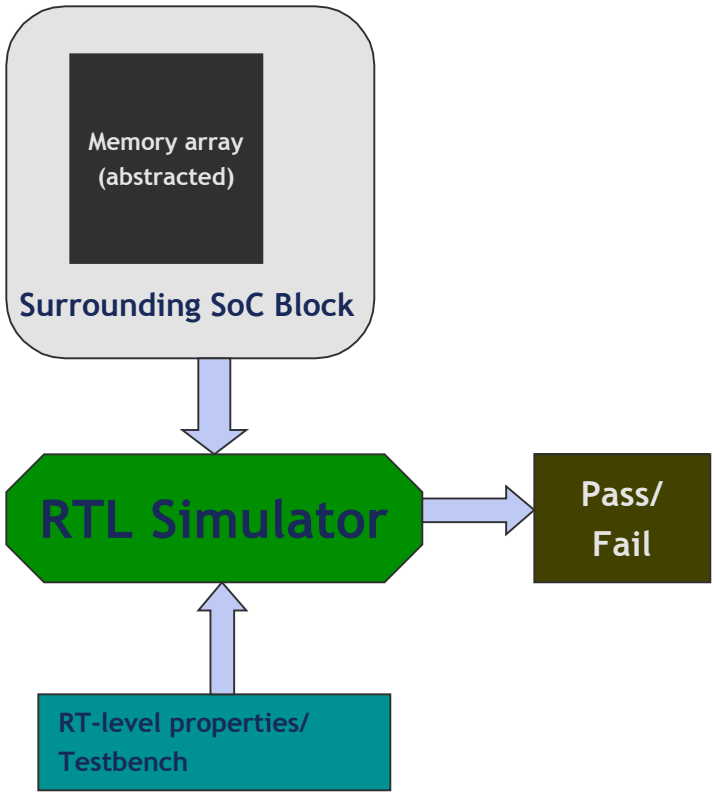
Current Practice in Memory Verification

Question 2:

Does the memory array function correctly within a larger SoC block?

Use fast RTL or high-level simulators

- **Memory is abstracted into a C/C++/other model, that represents the interface of the memory to the surrounding SoC blocks.**
- Simulation checks SoC-level properties.



How do we know that the transistor implementation of the memory array and its C/C++ abstraction?

Traditional Answer: Switch-level Models

Abstract the transistor network into a "switch level model" (Bryant,1984; Bryant et al., 1987)

- Represent the network as a set of nodes connected by transistor switches
  - Each node has state 0, 1, or X
  - Each switch has state open(0), closed(1), or indeterminate(X)
- State transitions are specified by switch equations
  - Typically constructed by partitioning the network into channel connected subcomponents.

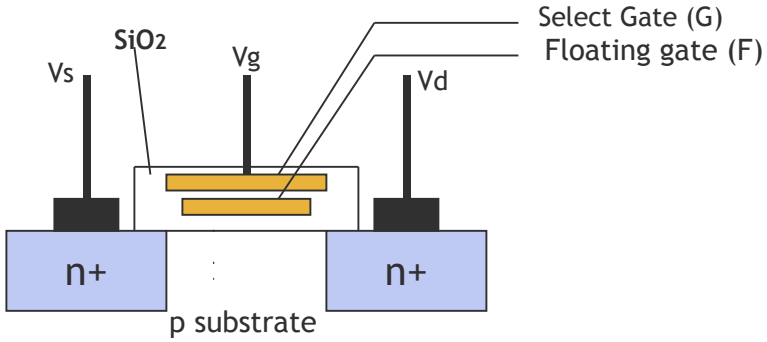
Compare the switch level model with high level (RTL) specification

Switch level analyzers such as ANAMOS accurately capture many aspects of transistor circuits.

Unfortunately, switch-level abstractions do not work for flash.

FLASH Memories

FLASH contains both traditional (MOS) and floating gate (FG) transistors.



The capacitive coupling between G, F, and substrate is used to regulate the threshold voltage by controlling the charge stored.

- Low threshold = Logic 1
- High threshold = Logic 0

The capacitive coupling breaks the view of a transistor as an on/off switch as taken by switch-level analyzers.

Bridging the Gap

The key issue is to develop tractable abstractions of transistor-level memory implementations.

- Goals:
  - Facilitate verification of **full memory arrays** with respect to high-level interface specification.
  - Abstraction must be validated by available data from SPICE-level circuit simulation results

- Key Observation:

  Memory networks are implemented by connecting together analog blocks with well-defined **behavioral characteristics** within the range of operation.

It makes sense for abstractions to capture these behavioral characteristics.

Approach Overview

- **A Hierarchical Approach to Modeling**
  - Develop parameterized **behavioral abstractions** for basic memory blocks (bit cells, sense amplifiers, etc.)
  - Construct abstraction for entire memory array as interactive composition of these blocks.

- **Two key features:**
  - Abstractions for basic memory blocks can be directly validated (even generated) from SPICE-level simulation data.
  - Composition directly corresponds to well-understood circuit hierarchies and block interconnection.

The approach is agnostic to the type of transistors used in the implementation.

Behavioral Abstraction of Bitcell

- **Reading a bitcell:**
  - Apply voltage between low and high thresholds to selected wordline
  - If the bitcell has value 0, transistor does not turn on.
  - Otherwise current is detected at the sense amplifier, reading 1.

- **Programming a flash bitcell:**
  - Raise the FG voltage to high threshold (Channel Hot-Electron Injection).
  - Read with Gate voltage larger than high threshold.
  - A read of 0 means successful programming; otherwise iterate
  - Fail after a certain number of iterations.

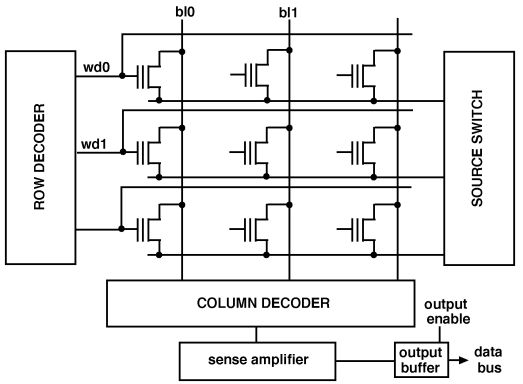
We can clearly model the behavior of the bitcell as a state machine.

- Developed a library of state machines corresponding to each such operation.

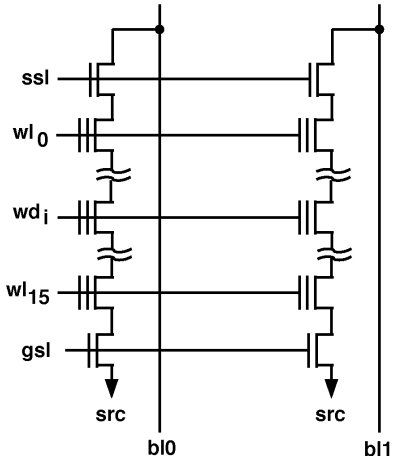
- **The state machines can be validated by SPICE-level simulation.**
  - Correspond exactly with the behaviors and operating constraints that are checked at SPICE level.

Abstracting an Array

- **Consider the NOR configuration**
  - The behavior of the array is simply a composition of behaviors of individual state machines.
  - Composition of behavior corresponds to interconnection of components.

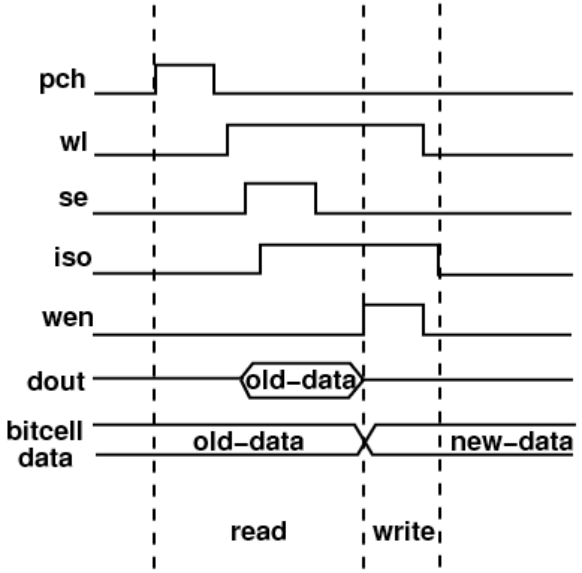


Similar composition works for NAND



Parameterization

Consider a typical memory read/write cycle.



- **Correct functionality requires:**
  - **pch** turns low before **wl** becomes high
  - **se** turns high after **wl** becomes high
  - **iso** turns high after **se**
  - …

It is the relative timing of the events that affects correctness.

Within broad limits, the absolute times don't affect correctness, only efficiency.

The models capture this by parameterizing the behavioral abstractions.

Parameterization (Contd.)

- **Behavioral abstractions are parameterized with respect to a number of metrics.**
  - Relative timing
  - Transistor threshold voltages
  - Array size
  - …

Why is parameterization important?

- Verification of a parameterized abstraction guarantees correctness **of a range of concrete implementation** in one fell swoop.
- Parameterization lets us focus on the aspects of the design that are **really relevant** to functional correctness while abstracting other details.

Completed and Current Work

❑ Completed Work

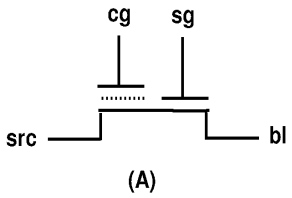
- Developed parameterized behavioral abstraction for SRAM and FLASH memory components.
- Completed verification of a NOR and NAND configuration with respect to interface specification.

❑ Current Work

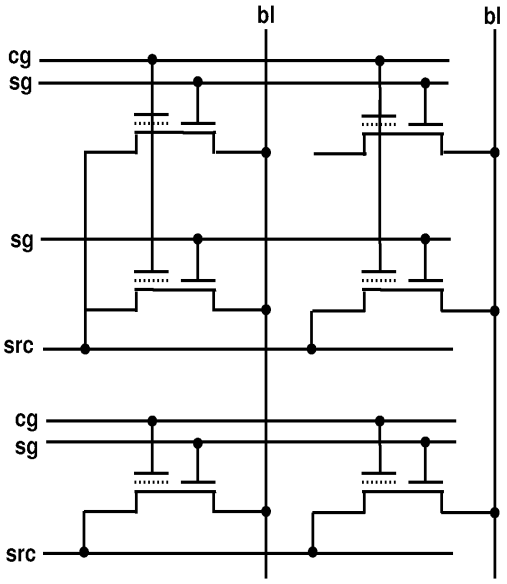
- Modeling and verification of split-gate FLASH models.
  - Modeling completed, verification almost done.
- Integrating behavioral abstractions into functional verification tool flow.
- Extending the approach to other AMS designs.
- Automating construction of behavioral models.

Split-Gate Flash

A somewhat different configuration due to the split between *sg* and *cg*.



(A)



(B)

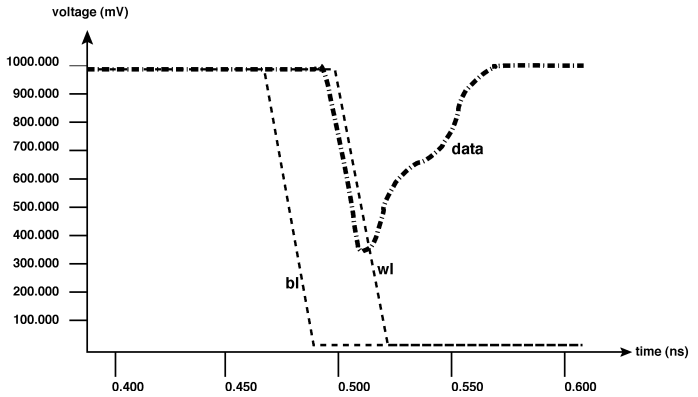
Behavioral models are somewhat different but the same approach works.

Adapting to Functional Verification Flows

- **Since modeling is based on state machines, they are amenable to standard (static and dynamic) verification flows.**
  - Completed verification of a NOR and NAND configuration with respect to interface specification.
  - Work in progress to effectively leverage functional verification tool flows.

A cool aspect is the close correspondence between SPICE-level models.

- **Bug was inserted in an early version --- the setup time was insufficient.**
  - Immediately caught in the assume-guarantee reasoning.
  - Counterexample corresponds directly to SPICE-level trace.



Automating Construction of Behavioral Models

Behavioral models are small, parameterized state machines. But their construction is delicate.

Can we use traces from SPICE-level simulation to automate their construction?

We are exploring a semi-supervised learning framework.

There has been significant recent interest on learning similar state machines. We think we can leverage these techniques to learn behavioral abstractions.

Concluding Remarks

The work provides the first formal platform for functional verification of FLASH designs.

The key novelty is in a different way of looking at the problem.

- Focus on formalizing behaviors, not structure.
- This allows effective formalization of analog designs while circumventing the problem of abstracting analog behavior with switch-level equations.

As more and more AMS components are integrated with digital SoC blocks, it is crucial to develop such techniques to permit effective functional verification of AMS components.

Thank You!

Verification

Specification:

- A state machine S representing memory interface to surrounding SoC block.

Implementation:

- A parameterized state machine I representing composition of behavioral abstractions of memory components.

Verification Problem:

- **Does I correctly implement S?**

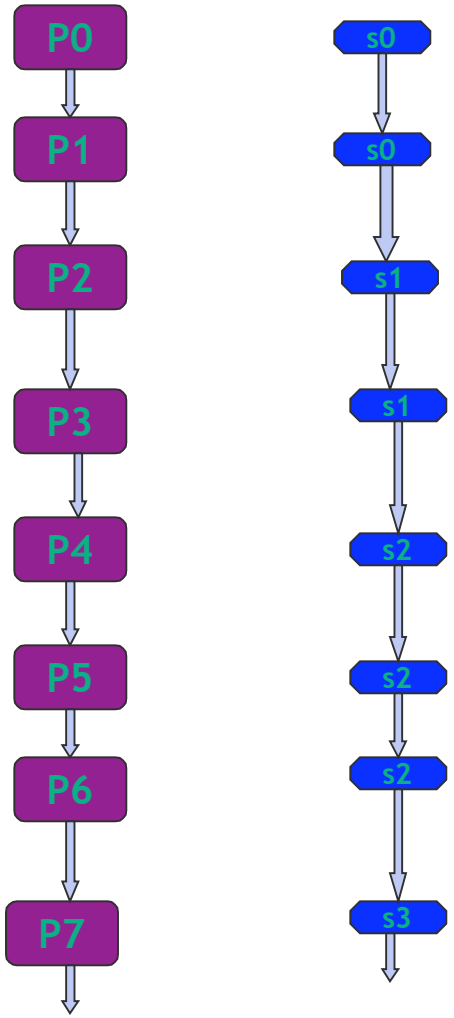
Making the above statement precise requires a formal notion of correspondence between two state machines.

The notion of correspondence used is stuttering trace containment.

Stuttering Trace Containment

Every execution of I can be viewed as an execution of S up to stuttering.

- Well-understood notion of correspondence
- Effective for comparing systems at different levels of abstraction



Stuttering steps correspond to intermediate state changes in the behavioral models and matching steps correspond to completion of memory operations.

Memories and STC

- **Theorem:**
  - I implements S if there exist functions *inv*, *rep*, *commit* and *pick* such that the following formulas are provable:
    - *init(p) implies inv(p)*
    - *inv(p) implies inv(I(p,j))*
    - *inv(p) and commit(p,j) implies rep(I(p,j))=S(rep(p),pick(p,j))*
    - *inv(p) and not commit(p,j) implies rep(I(p,j)) = rep(p)*

- **For memories**
  - *rep* is simply the projection of bitcell values and read buffer.
  - *commit* specifies if the current transition completes the operation.
  - *pick* specifies the kind of operation being completed (read/program/erase) and the relevant memory sectors.

So the non-trivial aspect of the verification is in defining and checking a suitable invariant *inv*.

Completed and Current Work

❑ Completed Work

- Developed parameterized behavioral abstraction for SRAM and FLASH memory components.
- Completed verification of a NOR and NAND configuration with respect to interface specification.

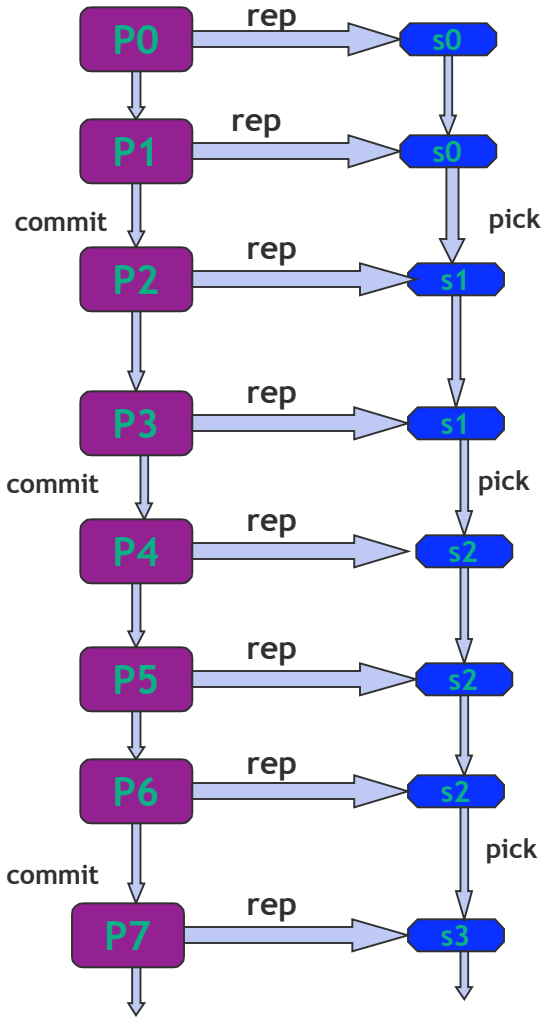
❑ Current Work

- Modeling and verification of split-gate FLASH models.
  - Modeling completed, verification almost done.
- Integrating behavioral abstractions into functional verification tool flow.
- Extending the approach to other AMS designs.
- Automating construction of behavioral models.

Proofs of Stuttering Trace Containment

- **Theorem:**
  - I implements S if there exist functions *inv*, *rep*, *commit* and *pick* such that the following formulas are provable:
    - *init(p) implies inv(p)*
    - *inv(p) implies inv(I(p,j))*
    - *inv(p) and commit(p,j) implies rep(I(p,j))=S(rep(p),pick(p,j))*
    - *inv(p) and not commit(p,j) implies rep(I(p,j)) = rep(p)*

Observe that each obligation involves single steps, not entire execution.



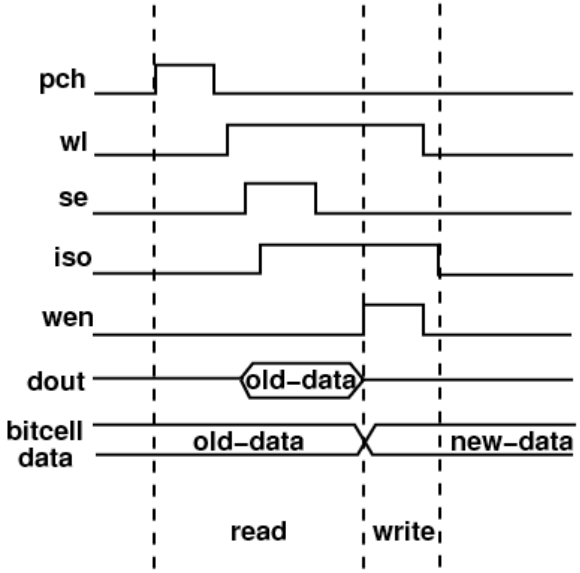
Invariant Proving for Memory Verification

- **Note that the behavioral abstractions are parameterized.**
  - Proving invariants on the model is a *parameterized invariant checking problem*.
  - Parameterized invariant checking is intractable in general (Apt and Kozen, 1986)

But the problem is simplified for memory designs through assume-guarantee reasoning.

Assume-Guarantee Reasoning

- **From behavioral abstraction**
  - A bitcell works correctly if it is provided the right stimulus satisfying all its constraints.
- Read produces the right value in the read buffer, if
  - *iso* is applied appropriately to isolate the bitcell, and
  - *pch*, *wl*, and *se* are appropriately applied.
  - …
- Reduces invariant on read buffer to the invariants on pch, iso, wl, …



In general, each component guarantees its invariant assuming that its surrounding components satisfy their invariants.