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 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
- Pressure to make them smaller, work at lower energy levels, less power consumption, etc.
CURRENT PRACTICE IN MEMORY VERIFICATION

Question 1:

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

Current Answer:

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

Problem 1:

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 array at this level.

CURRENT PRACTICE IN MEMORY VERIFICATION

Question 2:

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

Current Answer:

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

Problem 2:

How do we know that the transistor implementation of the memory array behaves like its C/C++ abstraction?
MEMORY MODELING

A New Approach

THE DIODE AND ITS DEPLETION REGION

Depletion region has depleted its majority carriers
THE DIODE AND BIASES

Forward bias:
- Diffusion current (majority carriers) dominant
- Typically avoided in digital ICs

Reverse bias:
- Drift current (minority carriers) dominant
- Operating mode (general)

THE MOS TRANSISTOR

Typically $V_S, V_D > V_{sub}$ and so there are two depletion regions due to the two reverse biased diodes

But if we keep $V_{gs} = 0$ then there is no depletion under the gate
THE MOS TRANSISTOR

As $V_{gs}$ increases beyond Threshold Voltage ($V_{t}$):
- $G$ and $S$ become capacitor plates with SiO2 as dielectric
- Since $V_{gs} > 0$, $G$ plate is positive and repels majority carriers in the substrate away from $G$
- This creates a depletion region under $G$ in the channel

But if $V_{ds} = 0$ then there cannot be any current in the channel
As $V_{gs}$ increases the channel intensifies

CURRENT-VOLTAGE RELATIONS

Source: “Digital Integrated Circuits: A design Perspective”
Rabaey, Chandrakasan, Nikolic
**THE THRESHOLD VOLTAGE**

\[
V_T = \Phi_{mS} - 2\Phi_F - \frac{\Phi_B}{C_{ox}} - \frac{\Phi_{SS}}{C_{ox}} - \frac{\Phi_I}{C_{ox}}
\]

\[
V_T = V_{T0} + \gamma \left( \sqrt{-2\Phi_F + V_{SB}} - \sqrt{-2\Phi_F} \right)
\]

with

\[
V_{T0} = \Phi_{mS} - 2\Phi_F - \frac{\Phi_B}{C_{ox}} - \frac{\Phi_{SS}}{C_{ox}} - \frac{\Phi_I}{C_{ox}}
\]

and

\[
\gamma = \frac{q\Phi_{Si}N_A}{C_{ox}}
\]

Source: “Digital Integrated Circuits: A Design Perspective”
Babaev, Chandrakasan, Nikolic

---

**CMOS INVERTER**

\[
V_{in} \rightarrow \text{N-channel MOSFET} \rightarrow V_{out} \rightarrow C_L \rightarrow V_{DD}
\]

\[
V_{out} = \begin{cases} 
0 & V_{in} < 0.5 \\
1 & V_{in} > 2.5 \\
V_{in} & 0.5 \leq V_{in} \leq 2.5 
\end{cases}
\]
ONE TYPE OF ABSTRACTION

SWITCH-LEVEL MODELING

Traditional Answer: Abstract the transistor network into a “switch level model” (Bryant, 1984)

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

Switch level analyzer such as ANAMOS accurately capture many aspects of transistor circuits.
DEFICIENCIES OF SWITCH LEVEL MODELS

Switch level analyzers ignore many analog effects.

- Strength assignment in ANAMOS produces significant mismatch with detailed analog simulation if transistors have closely matching but different strengths.

- The deficiencies have been addressed by designing more and more sophisticated analyzers

(Agarwal, 1990; Krishnamurthy et al., 2000)

Fundamental Problem:

Crisply approximate analog behaviors with equations in discrete algebra

BEHAVIORAL ABSTRACTION OF SRAM READ

The problem is exacerbated by the advent of FLASH memories
FLASH BITCELL

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 stored charge.

FLASH BITCELL BASICS

For $V_{gs} = V_t$, $\Rightarrow$ channel forms.

With charge in FG, for the same $V_{gs} = V_t$, $\Rightarrow$ no channel.

For $V_{gs} = V_{th} > V_t$, $\Rightarrow$ channel forms.

With no charge in FG, for the same $V_{gs} = V_t$, $\Rightarrow$ channel forms.

Call this $V_{tL}, V_{tL} < V_{th}$.
**FLASH BITCELL OPERATIONS**

Low threshold (VtL) \( \Rightarrow \) signifies Logic 1 (erase state, E)
High threshold (VtH) \( \Rightarrow \) signifies Logic 0 (program/write state, P)

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

**VT DISTRIBUTION FOR FG BITCELS**

- VtL and VtH are distributions, not exact values
- Apply Vgs such that VtL < Vgs < VtH.
  - Channel forms and current flows if and only if there is no charge in FG, i.e., the bitcell is erased/stores Logic 1
- Multi-level Flash bitcells through multi-level Vt's
FLASH BITCELL OPERATIONS

Multiple voltage levels break the view of nodes being at 1/0/X as taken by switch-level analyzers

SO THE GAP REMAINS

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

* Neither equivalence checking nor formal verification is used for Flash memory verification
* Potential silicon bugs could cost a huge debug effort – great # of engineers for several weeks
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 behavior 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 clock interconnection

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

BEHAVIORAL ABSTRACTION OF BITCELL

• Reading a Flash bitcell:
  □ Apply voltage between low and high thresholds to selected wordline
  □ If the bitcell is erased (value 1), transistor turns on.
  □ Current is detected at the sense amplifier, reading 1.
**BEHAVIORAL ABSTRACTION OF BITCELL (CONT.)**

- Programming a Flash bitcell:
  - Raise the FG voltage to high threshold (through Channel HotElectron Injection).
  - Read the bitcell.
  - If its an unsuccessful programming then iterate
  - Fail after a certain number of iterations.

**BEHAVIORAL ABSTRACTION OF BITCELL (CONT.)**

- Erasing a Flash bitcell:
  - Typically done on entire sectors
  - Raise threshold voltage of all the bitcells by programming
  - Remove charge (through Fowler-Nordheim Tunneling) to lower the threshold voltage of all bitcells to the lower bound
  - Soft programming to correct the threshold voltage has have fallen too far below the VtL bell-curve.
BEHAVIORAL ABSTRACTION OF BITCELL (CONT.)

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 across readily available SPICE level simulations.

- Correspond exactly with the behaviors and operating constraints that are checked at SPICE level.

MODELING BEHAVIOR UNDER UNCERTAINTY

- Programming a Flash bitcell:
  - Raise the FG voltage to high threshold (through Channel HotElectron Injection).
  - Read the bitcell.
  - If its an unsuccessful programming then iterate
  - Fail after a certain number of iterations.

- **We model uncertainty through non-determinism.**
  - Each state machine has an additional “oracle” stimulus.
  - The oracle determines if the current iteration is successful.
  - The read succeeds if and only if the oracle says “successful”.

Non-determinism is also necessary for erase to determine the subset of bitcells whose threshold has fallen too far.
ABSTRACTING AN ENTIRE 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**

PARAMETRIZATION

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 **parametrizing** the behavioral abstractions.
PARAMETRIZATION (CONT.)

- Behavioral abstractions are parameterized with respect to a number of metrics.
  - Relative timing
  - Transistor threshold voltages
  - Array size
  - Program/Erase voltage values
  - ...

Why is parameterization important?

1. Verification of a parameterized abstraction guarantees correctness of a range of concrete implementation in one fell swoop.

2. Parameterization lets us focus on the aspects of the design that are really relevant to functional correctness while abstracting other details
### 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.
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:

- $\text{init}(p)$ implies $inv(p)$
- $inv(p)$ implies $inv(I(p,j))$
- $inv(p)$ and $\text{commit}(p,j)$ implies $rep(I(p,j)) = S(rep(p), pick(p,j))$
- $inv(p)$ and not $\text{commit}(p,j)$ implies $rep(I(p,j)) = rep(p)$

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

MEMORIES AND STC

Theorem:

$I$ implements $S$ if there exist functions $inv, rep, commit$ and $pick$ such that the following formulas are provable:

- $\text{init}(p)$ implies $inv(p)$
- $inv(p)$ implies $inv(I(p,j))$
- $inv(p)$ and $\text{commit}(p,j)$ implies $rep(I(p,j)) = S(rep(p), pick(p,j))$
- $inv(p)$ and not $\text{commit}(p,j)$ 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 (being read/program/erase) and the relevant memory sectors.

The non-trivial aspect of the verification is in defining and checking suitable invariant $inv$
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
BUGS

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

SPLIT-GATE FLASH

A more efficient cousin of the FG Flash bitcell
**SPLIT-GATE FLASH**

A somewhat different configuration and more complex electrical characteristics due to independent controls SG and CG.

---

**SPLIT-GATE FLASH**

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.
- A challenge is to effectively leverage this technique in industrial functional verification tool flows.

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.
WORK DETAILS

Developed parameterized behavioral abstraction for SRAM and FLASH memory components.

– Developed parameterized behavioral abstraction for SRAM and FLASH memory components.
– Completed verification of FG NOR and NAND configuration with respect to interface specification.
– Verified a NOR configuration of split-gate flash memory.
– Integrating behavioral abstractions into functional verification tool flow.

Used ACL2 theorem prover

– Industrial-strength proof system with applications in diverse system verification.
– All behavioral models formalized as theories in ACL2
– Makes use of ACL2’s powerful logic, induction principles, and existing infrastructure to reason about state machines

CONCLUSIONS