Coping with Imperfections in Digital Logic

Igor L. Markov
University of Michigan, Dept. of EECS

http://www.eecs.umich.edu/~imarkov

Transient Failures

- Random changes $0 \rightarrow 1$ or $1 \rightarrow 0$
  - Very rare
  - Do not repeat soon, difficult to trace
  - May lead to computer crashes
- Car electronics, statistics from Ford and GM:
  - More failures at high altitudes
- Electronics in airplanes and satellites
  - Must withstand even very rare crashes
- Servers that process credit card payments
  - 1 min of downtime costs millions of $\$$

Soft Errors

- Transient faults $\rightarrow$ soft errors
  - Mostly caused by neutron hits
  - Radiation in lead packaging, cosmic radiation
  - Became noticeable as transistors became smaller
- Incidence varies with altitude
  $\Rightarrow$ Much more pronounced in aerospace applications
  - Especially during solar flares
- Rumors
  - Sun Micro didn’t turn on ECC on L2 cache by mistake
  - Lost customers

Where Neutrons Come From

How Neutrons Can Generate Charged Particles in any IC

Note: These alphas and other charged particles are being generated right in the silicon itself. Unlike packaging-induced alphas, they do not have to penetrate the top metalization. They can be generated right where they can do the most harm.
How Neutron Flux Varies
with altitude
with latitude

Neutron Models: Flux vs. Altitude
1-10 MeV Atmospheric Neutron Flux

Neutron Models: Flux vs. Latitude
1-10 MeV Atmospheric Neutron Flux

Averaged Over Longitude

Reliability vs Power

Variation of SER with Chip Voltage

T.I. Alpha SER
T.I. Cosmic SER
T.I. BPSG SET
IBM 8286 Cosmic SER
IBM 8278 BPSG SER
IBM 8278 Alpha SER

Normalized SER

Chip Voltage

Impact of Particle Strikes

What NSEU testing was done by Xilinx

- Proton Cross Sections
  - Taken at Crocker (Davis) and Texas A&M
  - Correlation with neutron data was disappointing
- Neutron Cross Sections
  - Taken at the LANSCE facility at Los Alamos
  - Evaluating contribution of energy spectrum models
- Atmospheric Neutron Testing (Rosetta)
  - Large population of parts
  - Tested at three altitudes
  - Correlated with LANSCE results

Some Faults Do Not Propagate

- Controlling values, don’t-cares and logic masking
- Temporal masking - soft fault before transition
- Electrical masking - glitch attenuation

Error-correction for Memories

- 1970s: random bit-flips in memories seen
  - about once a year
  - Can lead to computer crashes
- IBM mainframes:
  - 10+ years without crashing
  - Credit-card transactions, etc
  - Heavy use of redundancy
- Error-correcting codes
  - Example: triple redundancy
    0→000, 1→111
  - Any one bit-flip can be corrected by voting
Challenges

- Circuits grow more susceptible to soft errors at newer technology nodes
- Novel, sub-nanometer circuit devices can be inherently unreliable

⇒ We ought to:
  - Evaluate reliability early (account for masking)
  - Design preventively, because error correction is difficult for combinational logic
  - Develop circuit test techniques for soft errors

Obstacles to Logic-level SER Evaluation

- Scalability
  - Must support incremental re-evaluation
- Technology & layout independence
  - Detailed electrical, timing info unavailable
  - Need to focus on logic masking
- Fault Modeling
  - Detailed glitch waveforms too slow for synthesis & test
  - Need simple probabilistic models

Simulating Transient Faults

- Fault models used in CAD tools today are geared towards manufacturing defects
  - Need inherently probabilistic fault models
- Example: probabilistic AND gate
  - Faultiness of a circuit depends on circuit structure and gate fault models
  - Wire faults can be modeled by fake faulty buffers

Probabilistic Transfer Matrix (PTM)

- Row indices represent outputs values
- Column indices represent inputs values
- Matrix elements capture pairwise transition probabilities

\[
\begin{bmatrix}
0 & 1 - p & 1 - p & 1 - p & p \\
0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 \\
1 & p & p & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
\end{bmatrix}
\]

\[\text{Prob[output}=1\text{]}\text{ when input is 10}\]

Examples of PTMs

- ITM
  - Stochastic
  - S-a-1

- PTM 1
  - S-a-1

- PTM 2
  - Stochastic
  - S-a-1
  - (one-way)

- PTM 3
  - Wrong-gate
  - (NAND→AND)

Wiring PTMs

- Buffers and wires: \( I \) (Identity matrix)

- Fanout PTM \( F_2 \) (0→00, 1→11)

- Wire swap (01→10, 10→01)
Input Distributions

- For $k$ wires, consider a $2^k$-dim vector
  - Each element gives the probability of a particular input
- Example for 2 wires: $(\frac{1}{2}, 0, \frac{1}{4}, \frac{1}{4})$
  - 00 with probability $\frac{1}{2}$
  - 01 with probability 0
  - 10 with probability $\frac{1}{4}$
  - 11 with probability $\frac{1}{4}$
- PTM $\cdot$ input distribution = output distribution
- Easy to compute $\text{Prob}[\text{error}]$ as a $\Sigma$

Estimating Reliability Based on PTM

- Two basic methods of gate composition: serial and parallel

Computing Circuit PTMs

- Two key algebraic operations
  - Serial Connection
  - Parallel Connection

Tensor Product

- Tensor Product matrix $M \otimes N$ has entries given by $m_{ij} \cdot n_{kl}$

Circuit PTMs: Example

Compact Matrix Representation
Compact Matrix Representation

Binary Decision Diagram

\[
\begin{bmatrix}
1/2 & 1/2 & 1/2 & 1/2 \\
1/2 & 1/2 & 1/2 & 1/2 \\
1/2 & 1/2 & 1/2 & 1/2 \\
1/2 & 1/2 & 1/2 & 1/2
\end{bmatrix}
\]

Operations on Decision Diagrams

- Operations are performed on compressed matrices and vectors, w/o decompressing them
- Graph traversal algorithms based on Apply
  - Inner/outer product, scalar operations
  - Tensor product, matrix multiplication, matrix addition
- Runtime and memory usage of all operations depend on the size of DD operands (i.e., # of nodes)
  - Polynomial-sized DDs lead to polynomial runtime

Empirical Results with PTMs

Gate Susceptibility

- Calculating susceptibility
  - Add an error to the gate being evaluated
  - Leave all other gates ideal
  - Calculate the probability of error
- Find the most critical gates and reduce their error probability (e.g., from 0.05 to .005)
- Calculate improvement in reliability

Gate Susceptibility Results

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Gates</th>
<th>Aggs</th>
<th>Outputs</th>
<th>Width</th>
<th>Performance, (p=0)</th>
<th>Performance, (p=0.05)</th>
<th>Reliability, (p=0.05)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C17</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0.634</td>
<td>0.305</td>
<td>0.319</td>
</tr>
<tr>
<td>and</td>
<td>6</td>
<td>21</td>
<td>1</td>
<td>23</td>
<td>0.13</td>
<td>0.23</td>
<td>0.319</td>
</tr>
<tr>
<td>and1</td>
<td>8</td>
<td>7</td>
<td>20</td>
<td>22</td>
<td>0.13</td>
<td>0.23</td>
<td>0.319</td>
</tr>
<tr>
<td>xor</td>
<td>12</td>
<td>10</td>
<td>7</td>
<td>23</td>
<td>0.13</td>
<td>0.23</td>
<td>0.319</td>
</tr>
<tr>
<td>parity</td>
<td>15</td>
<td>16</td>
<td>1</td>
<td>23</td>
<td>0.13</td>
<td>0.23</td>
<td>0.319</td>
</tr>
<tr>
<td>psele</td>
<td>16</td>
<td>19</td>
<td>9</td>
<td>16</td>
<td>0.13</td>
<td>0.23</td>
<td>0.319</td>
</tr>
<tr>
<td>decode</td>
<td>18</td>
<td>5</td>
<td>16</td>
<td>13</td>
<td>0.13</td>
<td>0.23</td>
<td>0.319</td>
</tr>
<tr>
<td>cs</td>
<td>23</td>
<td>14</td>
<td>11</td>
<td>23</td>
<td>0.13</td>
<td>0.23</td>
<td>0.319</td>
</tr>
<tr>
<td>pm1</td>
<td>24</td>
<td>17</td>
<td>17</td>
<td>27</td>
<td>0.13</td>
<td>0.23</td>
<td>0.319</td>
</tr>
</tbody>
</table>

Probabilistic Fault Testing

- Traditional (deterministic) circuit test
  - Any fault can be detected by some input vector
- Probabilistic errors in circuits caused by external radiation, supply voltage fluctuations, etc.
  - May not be detected by any single input vector
- Idea: probabilistically repeat test vectors
  - How many times?
Example

- Gates X1 and AND1 have bit-flip prob’y 10% on all their inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0.9</td>
</tr>
<tr>
<td>01</td>
<td>0.18</td>
</tr>
<tr>
<td>10</td>
<td>0.0</td>
</tr>
<tr>
<td>11</td>
<td>0.82</td>
</tr>
</tbody>
</table>

Input 00 detects fault with probability 0.9
Input 10 detects fault with probability 0.0

Using PTMs in Circuit Test

- **Strategy 1:** one fault at a time
  - Inject fault into a gate
  - Calculate the resulting PTM
  - Read off a row (input combination) with the highest error probability
- This identifies input vectors best for each individual fault
  - May not be the best overall

Using PTMs in Circuit Test

- **Strategy 2**
  - First, consider conventional stuck-at faults
  - Use an existing ATPG tool to find a set of test vectors
  - Re-evaluate these vectors for probabilistic faults using PTMs
  - Compute multiplicity for each vector
- No need to use whole PTM
  - Only the rows suggested by ATPG

Test Vector Generation

- **Goal 1**
  - Minimize total number of test vectors for fixed probability threshold $p_{th}$ of detection
- **Goal 2**
  - Maximize $p_{th}$ for a fixed number $V$ of test vectors

Greedy Algorithms

- Pick vectors one by one
  - Cover the least covered fault(s) so as to ensure higher probability of detection
- Stopping criteria
  - For **Goal 1:** when $p_{th}$ is reached
  - For **Goal 2:** when $V$ is reached

Experimental Results

<table>
<thead>
<tr>
<th>Circuit</th>
<th>No. of inputs</th>
<th>$d_{th}=5$</th>
<th>$d_{th}=95$</th>
<th>$d_{th}=99$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. of vectors</td>
<td>Time (s)</td>
<td>No. of vectors</td>
<td>Time (s)</td>
</tr>
<tr>
<td>c6288</td>
<td>32</td>
<td>0.03</td>
<td>236</td>
<td>0.03</td>
</tr>
<tr>
<td>c1908</td>
<td>33</td>
<td>0.04</td>
<td>2161</td>
<td>0.10</td>
</tr>
<tr>
<td>c432</td>
<td>36</td>
<td>0.03</td>
<td>1947</td>
<td>1.43</td>
</tr>
<tr>
<td>c7552</td>
<td>178</td>
<td>0.64</td>
<td>3599</td>
<td>32.20</td>
</tr>
<tr>
<td>c2670</td>
<td>233</td>
<td>0.07</td>
<td>3729</td>
<td>49.45</td>
</tr>
</tbody>
</table>

Time (s)

<table>
<thead>
<tr>
<th>No. of vectors</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>0.03</td>
</tr>
<tr>
<td>54</td>
<td>0.08</td>
</tr>
<tr>
<td>105</td>
<td>0.13</td>
</tr>
<tr>
<td>268</td>
<td>0.19</td>
</tr>
<tr>
<td>4041</td>
<td>0.45</td>
</tr>
<tr>
<td>50</td>
<td>0.05</td>
</tr>
<tr>
<td>411</td>
<td>0.04</td>
</tr>
<tr>
<td>854</td>
<td>0.64</td>
</tr>
<tr>
<td>8680</td>
<td>1.68</td>
</tr>
<tr>
<td>233</td>
<td>0.07</td>
</tr>
<tr>
<td>884</td>
<td>0.08</td>
</tr>
<tr>
<td>1729</td>
<td>0.28</td>
</tr>
<tr>
<td>5682</td>
<td>0.49</td>
</tr>
</tbody>
</table>
Further Steps

- Speed up reliability evaluation
- Improve circuits to be more robust
- Guide logic synthesis tools toward reliable circuits

Our Methodology

- Linear-time reliability evaluator for use in synthesis
  - Driven by logic simulation
  - Incremental re-evaluation strategy
- Novel synthesis techniques
  - Minimal-overhead logic cloning
  - Fast local rewriting

SER Calculations

We perform single and multiple-error analysis.

\[ P_{err}(a) \times P_{err}(O_1) \]
\[ P_{err}(a) \times P_{err}(O_1) \times P_{err}(O_2) \]

Transient single stuck-at: Single-error-per-cycle assumption, each gate has error rate denoted gerr.

Transient multiple stuck-at: Gates have independent probabilities of error

Signatures and Bit-parallel Logic Simulation

- Signature: partial truth table associated with each node in a circuit
- Stimulate inputs with random simulation vectors
- Generate signatures through bit-parallel simulation

Philosophy ...

- If somewhere in a circuit a gate output turns to 1, ...
  - When does this cause wrong output values?
- Two conditions (testability)
  - The output must have been at 0
  - The change 0 → 1 must be observable

Controllability vs. Observability

- Probability of controlling a signal to a 0 or 1
- Ex: Input vectors 110 and 111 control x to 1
- Probability that an internal signal controls an output
- Ex: x is not observed for input vectors 000 and 100
  - Captured by two 1s in ODC mask
- NumOnes(ODC(f)) / NumSim
Fast Approximate Observability
Don’t Care Analysis

- Linear traversal from POs to PIs
- Exact without reconvergence

Algorithm
1. Examine each of target's fanout
2. AND ODC(fanout) with local ODC for each fanout
3. OR ODCs for each fanout

$$\text{ODC(Target)} = \{0\ldots\}$$

Testability
- Testability: The probability that a signal is both controlled and observed
- Ex: x is both controlled to 0 and observed for vector 010 and 011

$$\text{test0(x)} = \frac{\text{NumOnes}(\text{Sig}(x) \& \text{ODC}(x))}{\text{Num Simulations}}$$
$$\text{test1(x)} = \frac{\text{NumOnes}(\neg\text{Sig}(x) \& \text{ODC}(x))}{\text{Num Simulations}}$$

ODC-based Logic Cloning
- Idea: increase logic masking by decreasing observability
- Find two gates with $$g \subseteq f$$ (up to ODCs)
- Protect the fanout of $$f$$ from glitches by a new OR gate
- No need for complete logic replication or mutually exclusive fanin
- Covers are verified using a theorem prover (SAT)

Experimental Set-up
- Using ISCAS and IWLS OpenCore benchmarks
- 2048 simulations with sets of 64 parallel
- Validate results using PTM-based and ATALANTA-based exact computation multiple and single errors respectively
- Use MiniSAT to verify
- ODC-based logic covers
- Modified ABC to perform reliability-guided rewriting

Single-Error SER
- We compute reliability using testability:
  1. Generate signatures
  2. Derive ODC masks
  3. Derive testability measures (test0, test1)
  4. Weight the testability by probability of error to obtain SER
- $$\text{SER} = \frac{G_{\text{err1}}}{3} + \frac{G_{\text{err0}}}{3}$$

Logic Rewriting
- Rewrite local cuts with more reliable cut
  ABC [Mischenko 06]
- Allows for synthesis on large designs
- Optimize for area and reliability simultaneously
- Incrementally re-evaluate fanin/fanout cone of cut
- Can try hundreds of candidate changes

Experimental Set-up
- Using ISCAS and IWLS OpenCore benchmarks
- 2048 simulations with sets of 64 parallel
- Validate results using PTM-based and ATALANTA-based exact computation multiple and single errors respectively
- Use MiniSAT to verify
- ODC-based logic covers
- Modified ABC to perform reliability-guided rewriting
Empirical Results: SER Evaluation

<table>
<thead>
<tr>
<th>Circuit</th>
<th>SER</th>
<th>FASER</th>
<th>FASER</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>1.0</td>
<td>20.0</td>
<td>22.0</td>
</tr>
<tr>
<td>C880</td>
<td>1.0</td>
<td>10.0</td>
<td>120.0</td>
</tr>
<tr>
<td>C1355</td>
<td>1.0</td>
<td>20.0</td>
<td>40.0</td>
</tr>
<tr>
<td>C1908</td>
<td>1.0</td>
<td>20.0</td>
<td>66.0</td>
</tr>
<tr>
<td>C3540</td>
<td>1.0</td>
<td>60.0</td>
<td>149.0</td>
</tr>
<tr>
<td>C6280</td>
<td>1.0</td>
<td>120.0</td>
<td>278.0</td>
</tr>
</tbody>
</table>

- Both algorithms run in linear time
- Largest benchmarks on which SER has been reported

Empirical Results: Logic Cloning & Rewriting

<table>
<thead>
<tr>
<th>Circuit</th>
<th>SER</th>
<th>% Area Overhead</th>
<th>% SER Improvt</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>4.2E-4</td>
<td>58.7</td>
<td>13.7</td>
</tr>
<tr>
<td>C880</td>
<td>9.4E-3</td>
<td>0.6</td>
<td>2.6</td>
</tr>
<tr>
<td>C1355</td>
<td>1.1E-2</td>
<td>3.6</td>
<td>45.4</td>
</tr>
<tr>
<td>C3540</td>
<td>1.7E-2</td>
<td>3.6</td>
<td>45.4</td>
</tr>
</tbody>
</table>

- Both algorithms run in linear time
- Largest benchmarks on which SER has been reported

Error Analysis Reviewed

- **Probability Transfer Matrices**
  - Exponential size in #wires (unless compressed)
  - Capture all possible correlations of signals, errors explicitly & exactly
  - Accurately capture logic masking

- **Simulation-driven analysis**
  - Linear memory, linear runtime
  - Also capture correlations, but through sampling
  - **Caveat:** thousands of samples required
  - Efficiency achieved through bit-parallel simulation

Conclusions

- **Algebraic formalism for soft errors**
  - Accurate calculations, but slow
  - Useful for probabilistic circuit test
  - Used to “bootstrap” a faster approximate evaluator

- **Efficient linear-time evaluator**
  - Orders of magnitude improvement over prior work
  - Yet reasonably accurate
  - Incremental evaluation supported (used during synthesis)

- **Empirical results for resynthesis**
  - Improved reliability with minimal overhead

References

Available at [http://www.eecs.umich.edu/~imarkov/pubs/](http://www.eecs.umich.edu/~imarkov/pubs/)