



SOFT ERRORS IN DIGITAL CIRCUITS

Luis Entrena
Microelectronic Design & Applications Group
Universidad Carlos III de Madrid

1



Outline

- Introduction to test
- Soft errors
- Design for soft error mitigation. Hardening techniques
- Evaluation of soft error sensitivity



Introduction

Electronic circuits do fail!

- Need to test circuits
- *Test*: an experiment to exercise a circuit and analyze its response to determine whether the circuit works correctly
- *Diagnostic*: analyze the response of a circuit in order to determine the origin of a fault



The importance of testing

- Testing is most important for quality products, i.e. it is the way quality is ensured
- Quality is strongly related to economy:
 - Wrong pieces have to be withdrawn. Withdrawing may be disastrous when a wrong piece has been shipped to customer or failures occur while in operation
 - Testing is expensive: it accounts for an important and increasing percentage of the total cost of a circuit
- Need to find a good balance to satisfy the user's needs at a minimum cost



A simple example

- Testing a 1Kb memory
 - For each memory position:
 - Write a 0
 - Read & Check
 - Write a 1
 - Read & Check
 - Total number of memory accesses = 4K



A simple example

- What are the contents of the rest of memory cells?
- An “improved” testing approach:
 - Repeat the previous algorithm for each possible memory contents
 - Total number of memory accesses = $4K * 2^{1024} = 10^{311}!!!$
(age of the universe is about $4 * 10^{17}$ ns)
- Combinatorial explosion makes exhaustive testing unfeasible even for very small circuits



The difficulty of testing

- Exhaustive testing is not feasible even for very simple circuits
- The complexity of testing grows exponentially with circuit complexity
- Testing is affected by VLSI technology trends:
 - Reduced feature size
 - Increasing device integration
 - Increasing clock rates
 - Reduced noise margin



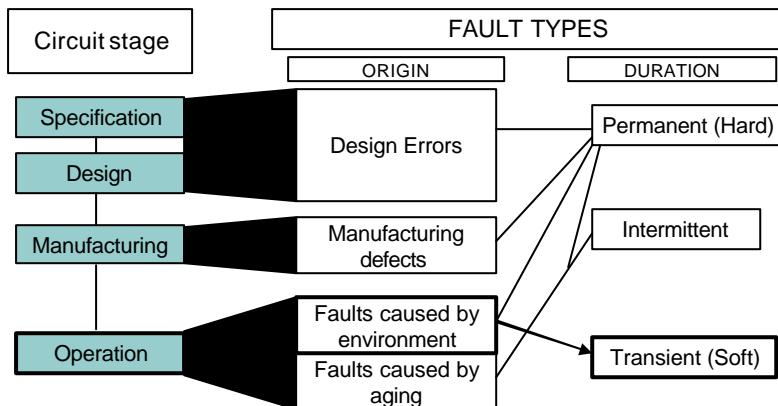
The problem of testing

- Need to find “smart” solutions to keep a good balance between:
 - The cost of testing
 - The capability to detect wrong behaviour and perform correction actions (e.g. repair or mask wrong behaviour)
- Solutions are not only off-chip
 - Testing without some kind of on-chip support may be too difficult or inefficient
 - Designers must use Design For Testability (DFT) and design hardening techniques

Defects, faults, errors and failures

- Defect: the result from an imperfect manufacturing process
- Fault: a physical defect or anything in a circuit that goes wrong
 - A circuit without defects may experience faults due to environmental conditions
- Error: a misbehaviour produced by a fault
 - Not all faults produce errors, some fault effects may be masked
- Failure: an unacceptable effect produced by an error
 - Not all errors produce failures, some errors may be acceptable

Types of faults





Design validation

- Goal: ensure the circuit has no design errors
- Design validation is part of the design cycle
 - Use a model (design) of the circuit
 - Simulate the model to detect design errors
 - Correct the model until no errors are found



Manufacturing test

- Goal: ensure the circuit has no defects and is correctly manufactured
- Manufacturing testing is part of the manufacturing process:
 - Every circuit coming out of the fab must be tested with Automatic Test Equipment (ATE)
- but has strong implications on design!
 - Designer must specify the test to be performed!
 - Designer must consider the use of DFT techniques to facilitate the testing process



On-line testing

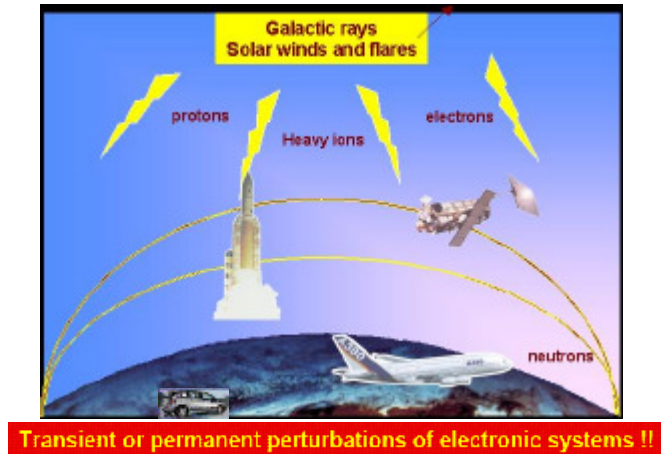
- Goal: check the circuit while in operation to ensure quality of service
- On-line testing deals with failures that occur during the lifetime of the circuit:
 - Aging, electromigration, corrosion, etc.
 - Environment: temperature, humidity, vibrations, electromagnetic interference, noise, radiation



Soft errors

- *soft error*: a random error induced by an event that corrupts the data stored in the device but does not damage the device itself
- Sources of soft errors:
 - Radiation
 - Electromagnetic Interference (EMI)
 - Electrical noise

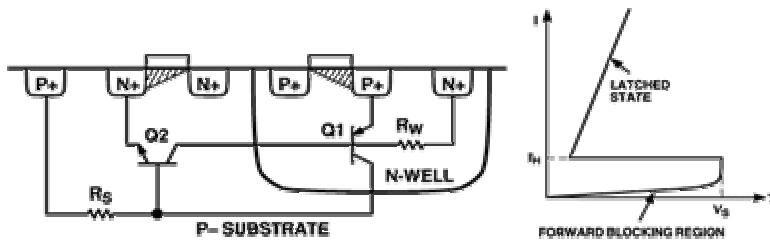
Radiation effects



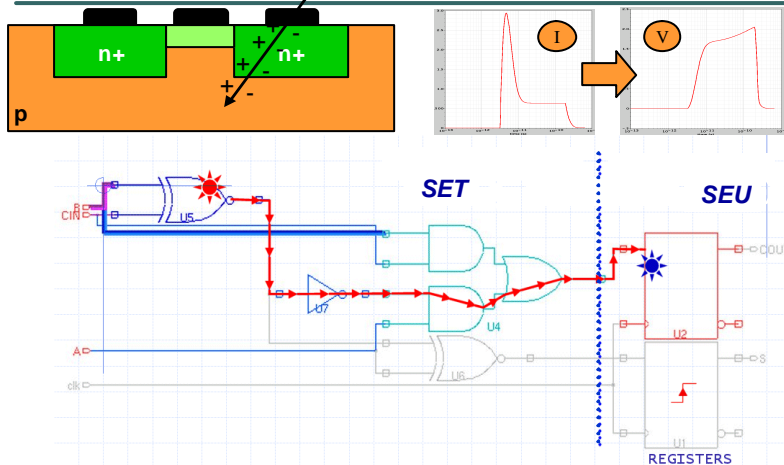
Radiation effects

- *Total Ionizing Dose (TID):* charge accumulated over time that modify device properties
- *Single-Event Latchup (SEL):* single event induced high current state (may become a permanent fault)
- *Single-Event Transient (SET):* a pulse induced in combinational logic
- *Single-Event Upset (SEU):* a state change of a memory element (memory or register bit)
- *Multiple-Bit Upset (MBU):* multiple bit changes produced by a single event

Latchup effect



SET and SEU





Importance of Soft errors

- Traditionally considered in specific sectors, that require high reliability and work in harsh environments:
 - Aerospace applications (satellites, aircrafts)
 - Nuclear plants
 - Military applications
- Now an increasing concern for many applications, even at ground level:
 - Medical
 - Transportation (automotive, railway)
 - Telecom, computing



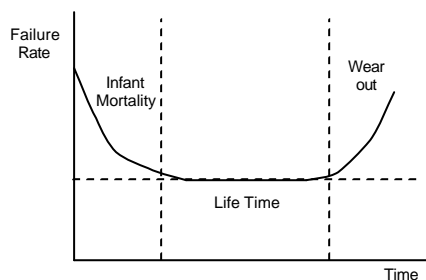
Importance of Soft errors

- Technology trends make circuits more sensitive to soft errors
 - Reduced transistor size and critical charge
 - Reduced supply voltage and noise margin
 - Higher clock frequencies
- Some nasty problems being experienced
 - High-end servers made by Sun Microsystems suddenly crashed in 1999 due to radiation effects
 - Cisco 12000 series routers experienced SEU effects (www.cisco.com)

Fault-tolerance goals

- *Reliability*: probability that a circuit will perform correctly in a given period of time
- *Availability*: probability that a circuit will perform correctly and is available to perform its functions at an instant of time
- *Safety*: probability that a circuit will either perform its functions correctly or will discontinue its functions in a safe manner
- *Maintainability*: probability that a failed system will be restored to an operational state within a specified period of time
- *Testability*: ability to test for certain attributes in a circuit
- *Dependability*: quality of service that a circuit provides

Reliability function





Measuring parameters

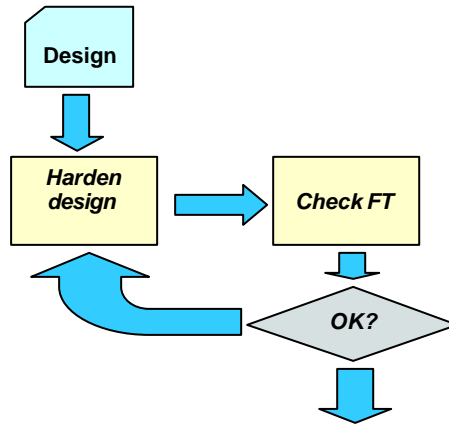
- Failure rate is typically measured in FIT (Failure In Time)
 - 1 FIT = 1 failure in 10^9 hours
- *Mean Time To Failure (MTTF)*: expected time before first failure occurs. It is the inverse of the Failure Rate
- *Mean Time To Repair (MTTR)*: average time for repairing
- *Mean Time Between Failures (MTBF)*: average time between failures = MTTF + MTTR



Example

- Typical PC with 1GB (8192 Mb) of typical DRAM (1000 FIT/Mb)
 - Failure Rate = $1000 * 8192 = 8192000$ FIT
 - MTTF = $1E+9 / 8192000 = 122$ hours ~ 5 days

Design process of a fault-tolerant circuit



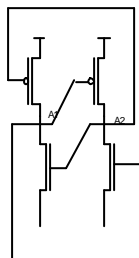
Hardening techniques

Technology solutions

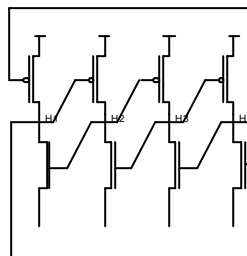
- Modify manufacturing process to reduce sensitivity to radiation
 - e.g. Silicon-On-Sapphire (SOS) and Silicon-On-Insulator technologies
- Rad-hard / Rad-tolerant technologies:
 - Immune to Total Ionizing Dose (up to some level)
 - Reduced sensitivity to SEL
 - Reduced sensitivity to SET and SEU
- Rad-hard / Rad-tolerant technologies can reduce sensitivity to soft errors by several orders of magnitude, but at the expense of higher cost, lower performance and higher power consumption

Transistor-level solutions

- Hardened memory cell



SRAM cell



Dual Interlocked storage Cell (DICE)



Redundancy

- Redundancy is required in order to differentiate correct behaviour from wrong behaviour
- Types of redundancy
 - Hardware redundancy
 - Information redundancy
 - Time redundancy
 - Software redundancy (when applicable)

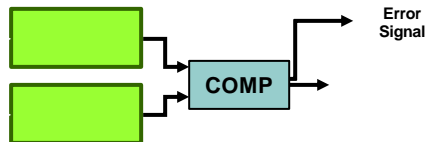


Hardware redundancy

- Replicate hardware components of the circuit and check the result of each replica
- Types of techniques:
 - Passive techniques: mask fault effects
 - Active techniques: detect the occurrence of faults and implement recovery actions
 - Hybrid techniques: use a combination of both

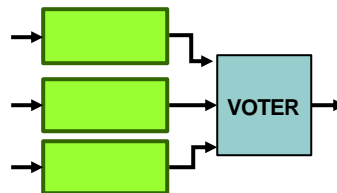
Duplication With Comparison (DWC)

- Replicate the circuit and add a comparator
- Can detect faults when they occur in any of the replicas
- More than 100% overhead



N-Modular Redundancy (NMR)

- Replicate the circuit N times and resolve the output through a majority voter
- Need an odd number of replicas. N=3 is the most typical case and it is called Triple Modular Redundancy (TMR)
- Can mask faults
- More than 200% overhead





Other hardware redundancy techniques

- *Standby Sparing*: use spares to substitute erroneous instances
- *NMR with spares*: mask the fault and substitute the erroneous instance with a spare
- *Watchdog timers*: the system must assert a signal in due time; otherwise, the watchdog timer issues a timeout exception

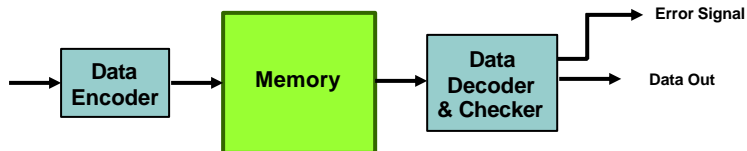


Hardware redundancy

- Can support fault detection (active techniques) or fault masking (passive techniques)
- Large overhead (> 100% for fault detection and > 200% for fault masking)

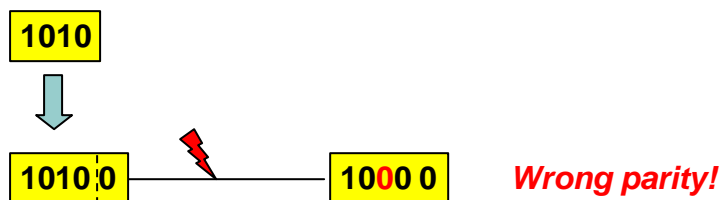
Information redundancy

- Use redundant codes
- Types of codes
 - Error detecting codes (EDCs)
 - Error correcting codes (ECCs)



Parity code

- Add a parity bit



- Can detect any single error (actually all errors of odd multiplicity)

M-out-of-n code

- Code words have n bits and contain exactly m 1s
- Example: 2-out-of-n
- *Unidirectional non-separable code:*
Can detect any single error and multiple unidirectional errors

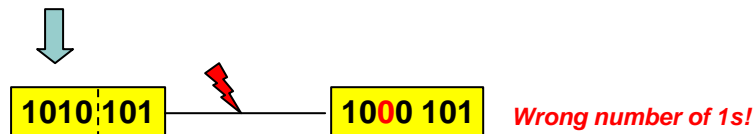
0	00011
1	11000
2	10100
3	01100
4	10010
5	01010
6	00110
7	10001
8	01001
9	00101



Berger Code

- Add check bits = the number of 1s (usually in complemented form)

1010 2 ones => add $\overline{010} = 101$



- *Unidirectional separable code:* Can detect any single error and multiple unidirectional errors

Hamming codes

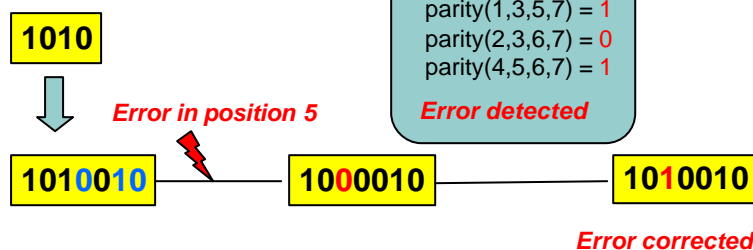
- Use overlapping parity bits to provide error detection capabilities
- Example: Hamming(7, 4)
 - 4 data bits + 3 check bits = 7 bits
 - Place check bits in 2^i positions

7	6	5	4	3	2	1
d_3	d_2	d_1	c_2	d_0	c_1	c_0

- Select c_0, c_1, c_2 as to obtain correct parities
 - $C_0 \Rightarrow$ parity(1,3,5,7)
 - $C_1 \Rightarrow$ parity(2,3,6,7)
 - $C_2 \Rightarrow$ parity(4,5,6,7)

Hamming codes

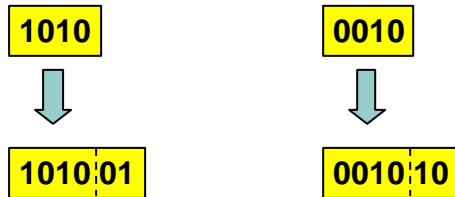
- Example: Hamming(7, 4)



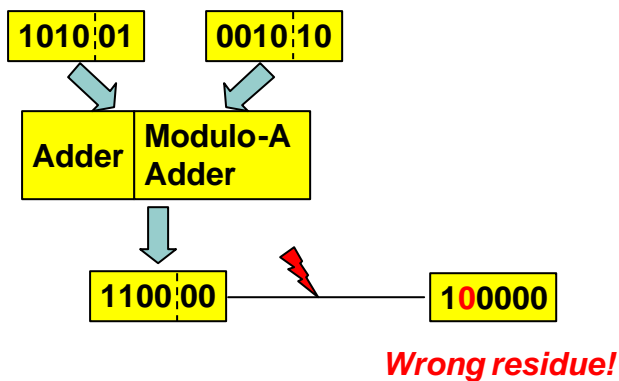
- Check bits identify the position of erroneous bits
- Hamming code can correct any single error

Arithmetic codes

- Use codes that are invariant to arithmetic operations
- Example: Residue code
 - Add the residue to a number A (e.g. A=3)



Residue codes





Information redundancy

- Widely used for memories, although they can be applied to logic as well
- Overhead increases with error detection and correction capabilities:
 - Redundant coded data requires more storage bits
 - Encoders, decoders and checkers require abundant logic

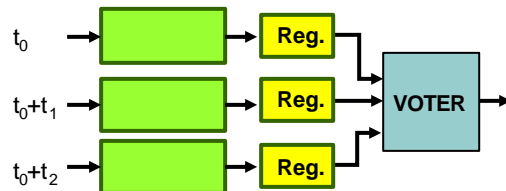


Time redundancy

- Use extra time instead of additional hardware or additional information bits
- Need little extra hardware but reduces performance
- Main techniques:
 - Repeat computations and compare the results
 - Sample a computation at several time instants

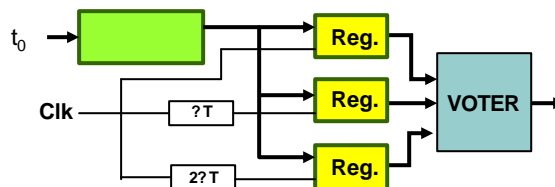
Repeating computations

- Repeat computations two or more times and compare the results



Sampling computations

- Sample a computation several times and compare the results





Time redundancy

- Useful when additional time can be used
- In its simplest form, time redundancy can only be used for transient faults (soft errors)
 - A permanent fault would produce errors for all computations
 - Time redundancy can be used to distinguish between permanent and transient faults
- Can be used for permanent errors if the computations are not exactly the same (e.g., using complemented data)

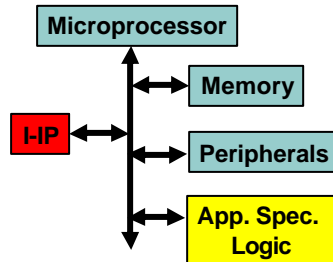


Software redundancy

- Use extra software to increase fault tolerance
- Main features
 - Applicable to microprocessors
 - Easy to use and implement
 - Can be associated to a software implementation of previously mentioned techniques:
 - Repeat instructions (time redundancy)
 - Use coded data (information redundancy)
 - May use specific hardware to accelerate checking

Hardening System-on-Chip (SoC)

- Use Infrastructure IP (I-IP)
 - I-IP are not functional, they are embedded just for reliability purposes
 - I-IP may detect, analyze and correct errors on-line



Conclusions

- Fault tolerance is not for free!
 - A large area or performance overhead is usually involved (even higher than 100% in many cases)
- Full fault tolerance is not feasible nor required in general
 - Only the most critical parts of the design should be hardened
- For critical designs, fault tolerance is a design requisite, as much as area or timing



Conclusions

- Fault tolerant design tasks
 - Identify the most critical parts of the design and apply the best technique for each case
 - Validate the fault tolerant design (check the circuit behavior in the presence of errors)
 - Evaluate the fault tolerance achieved and repeat the process until the design satisfies dependability specs
- What about tools?
 - Unfortunately, commercial tools are still in its infancy
 - Currently, hardening is mostly a manual task



***Evaluation of soft
error sensitivity***



Fault-tolerance evaluation

- Main objective:
 - Verify the circuit satisfies fault-tolerance requirements
- Other objectives:
 - Predict the behavior of the circuit in the presence of faults
 - Identify the critical parts of the circuits



Analytical methods

- Obtain a numerical estimation of dependability parameters by means of mathematical models
 - Combinatorial models, stochastic models, Markov chains, etc.
- Mathematical models are complex and difficult to apply to real circuits
 - Simplifications can be made at the expense of reducing accuracy
 - Cannot be used with IPs, since their internal structure is usually unknown

Experimental methods

- Direct measure of dependability parameters
 - Observe the circuit while in operation to analyze the response to real faults and extract dependability data (e.g. MTTF)
 - The most realistic approach
 - Requires large (and costly) experimentation time, since faults occur very rarely
 - Experiments are usually performed with many circuits, to increase the probability of faults, and in worst conditions (e.g., high altitude to increase the particle flux)
 - Does not provide useful data for diagnosis and dependability improvement
 - Mainly used to validate other approaches

Direct measure example

- Altitude SEE Test European Platform (ASTEP):
“Plateau du Pic de Bure”, French Alps (elevation:
2252m)



Direct measure example

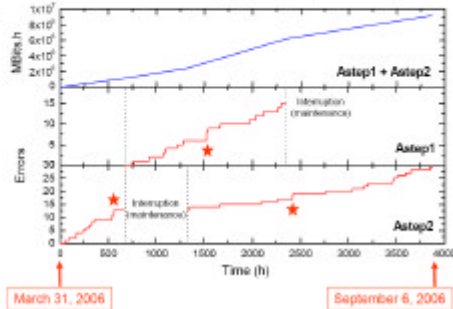


44 fails
including
38 SEU
3 MCU(2) *

physically
adjacent
bit cells in
all cases

First results for CMOS 130nm

Cumulative fail number vs. time
Nominal test conditions ($V_{DD} = 1.2V$, RT, checkerboard)



$4 \text{ Mbits cut under test} \times 1200 = 4.8 \text{ GBits}$

Experimental methods

- Fault injection
 - Accelerate the experimental measure by artificially injecting faults
 - Reduce the time required for the experiments
- Less accurate than direct methods, but widely accepted in practice
- Can be applied:
 - On a hardware implementation of the circuit
 - On a design model, at any design phase and at any level of abstraction

Fault injection on a hardware implementation

- Require at least a circuit prototype and cannot be used during the design phase
- Main techniques
 - Physical fault injection: use a physical means to inject faults
 - Forced radiation
 - Laser fault injection
 - Pin level
 - EMI-based
 - Power supply
 - Logical fault injection: use available circuit resources to inject faults. Examples:
 - JTAG
 - ICE (in microprocessors)
 - Software fault injection (in microprocessors)
 - Circuit reconfiguration (in configurable hardware)

Forced radiation

- Use particle accelerators:
 - Heavy ions, protons, neutrons, etc.
- Usually a requisite to qualify circuits for space applications
- Cannot control location and instant of fault injection
- Very expensive equipment!



Laser fault injection

- Inject faults using a laser beam
- Can select fault injection site
- Less mature technique.

Problems to be solved:

- Control spot size and laser energy
- Circuit is impacted on the back side to avoid laser reflection at metal layers
- Need very precise decap



Fault injection in microprocessors

- Use software fault injection
 - Modify code to emulate faults
- Typical approach:
 - Interrupt program execution
 - Read a selected register or memory position and write back a different value (fault injection)
 - Resume execution
 - Compare the result with golden execution



Fault injection in microprocessors

- Advantages
 - Easy to use
 - Fast
- Disadvantages
 - Intrusive (faulty execution is not the same as golden execution)
 - Faults can only be injected in visible memory elements (internal registers, such as pipeline registers are not accessible by software)

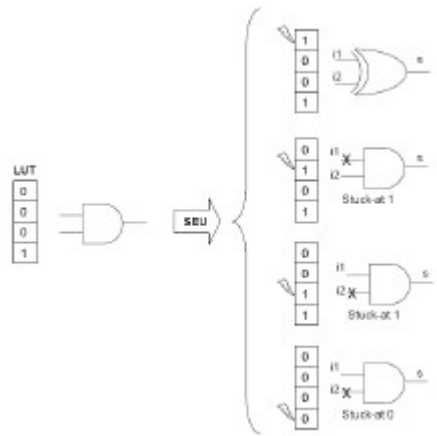


Fault injection in microprocessors

- Use debugging infrastructure
 - Most microprocessors today support OCD (On-Chip Debugging) or ICE (In-Circuit Emulation) capabilities
 - Not intrusive (almost)

Fault injection in configurable hardware

- Use FPGA configuration resources to inject faults
- Actually most faults in FPGAs appear in configuration memory!

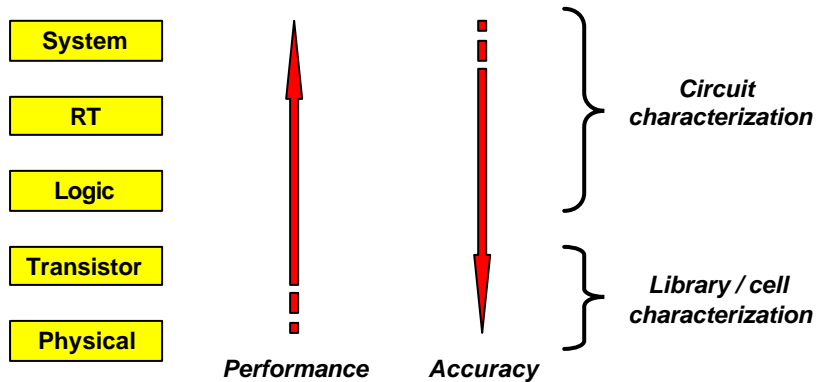


Fault injection on a design model

- Require a circuit model (e.g., an HDL description)
- Mainly intended to be used during the design phase
- Main techniques
 - Simulation: use a circuit simulator (e.g., an HDL simulator)
 - Emulation: use a fast prototyping platform (e.g., an FPGA)

Simulation-based fault injection

- At which level of abstraction?



RT-level fault injection

- Use an RT-level simulator
 - Commercial RTL simulators are widely available and easy to use
- Techniques
 - Modify the code to describe faulty behaviour
 - Use simulator interfaces
 - Modify the simulation tool



RT-level fault injection

- Code modification techniques
 - Saboteur: a component that can change the value of one or more signals
 - Mutant: a component that replaces another model component
- Pros and cons of code modification
 - Very flexible: can implement many fault models
 - Many tools proposed (mainly non-commercial)
 - Highly intrusive
 - Need to recompile for every fault or set of faults



RT-level fault injection

- Simulator interfaces
 - Command line interface:
 - Use scripts of simulator commands to modify signal values and control execution
 - Foreign Language interface (FLI) (e.g. Verilog PLI, Modelsim FLI, etc.)
 - Use FLI capabilities to control and observe signal values
- Advantages
 - Non intrusive
 - Fault injection is performed at run time instead of compile time



Emulation-based fault injection

- Simulation-based fault injection is very flexible, but also rather slow:
 - Typical fault injection rate in the order of 1 fault/sec.
- Solution: improve speed by emulating circuit in a FPGA
 - Much faster: circuit can run nearly at-speed
 - May support interaction with real environments
- Emulation-based fault injection approaches
 - FPGA dynamic reconfiguration
 - Instrumented circuit



Fault injection by reconfiguration

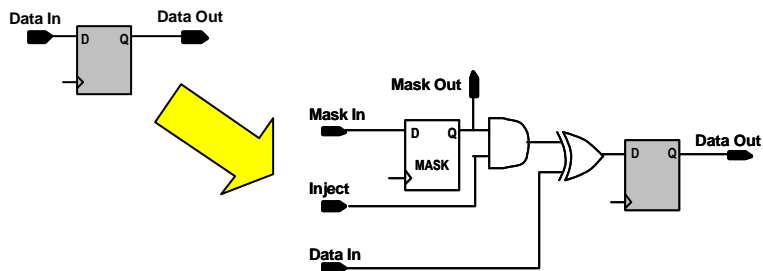
- Use dynamic reconfiguration capabilities to inject faults
 - Run circuit until fault injection time
 - Read configuration bitstream
 - Modify configuration bitstream and reconfigure
 - Resume execution
- Typical performance: 10-100 faults/sec
- Problem:
 - FF contents cannot be modified through the bitstream

Fault injection by reconfiguration

- SEU fault injection approach
 - Run circuit until fault injection time
 - Read the states of all FFs
 - Reconfigure the set/reset switch of each FF:
 - FF = '0' -> reset switch ON
 - FF = '1' -> set switch ON
 - Do the opposite for the faulty FF
 - Pulse Global Reset
 - Reconfigure set/reset switches to original value
 - Resume execution

Fault injection by circuit instrumentation

- Modify the circuit description to introduce fault injection infrastructure





Circuit instrumentation

- Mask FF
 - Set for the faulty FFs
 - Can be loaded serially (Mask FF form a shift register)
- Inject signal
 - Inverts FF data for faulty FFs

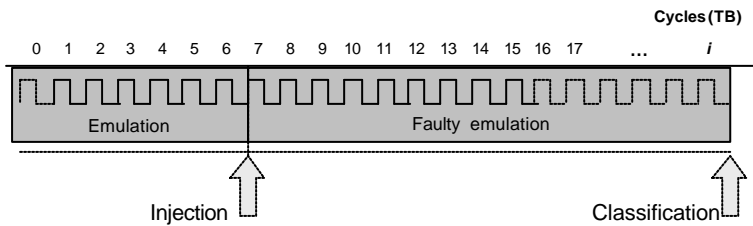


Fault injection by circuit instrumentation

- SEU fault injection approach
 - Set the Fault Injection Mask
 - Run circuit until fault injection time
 - Pulse Inject signal
 - Resume execution
- Setting the fault injection mask is much faster than FPGA reconfiguration
 - Typical performance: 10000 faults/sec

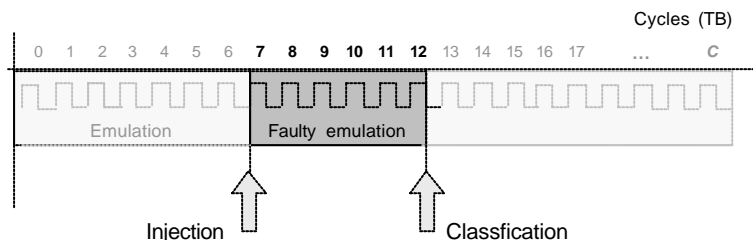
Fault injection optimization

- Run circuit until fault injection time
- Inject fault
- Resume execution
- Classify fault

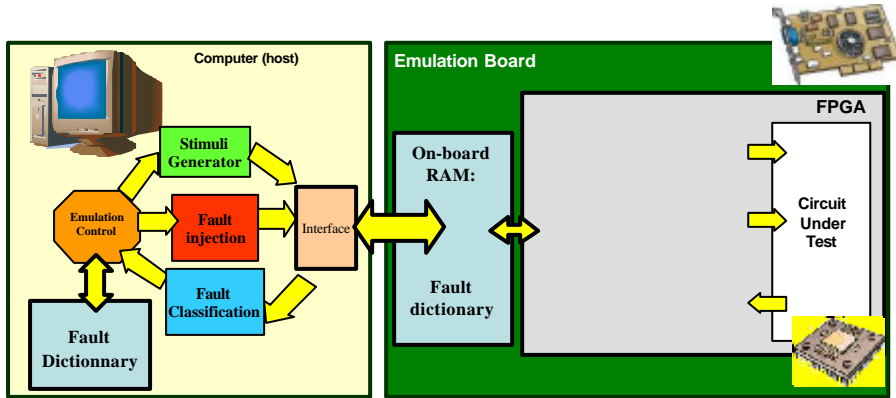


Fault injection optimization

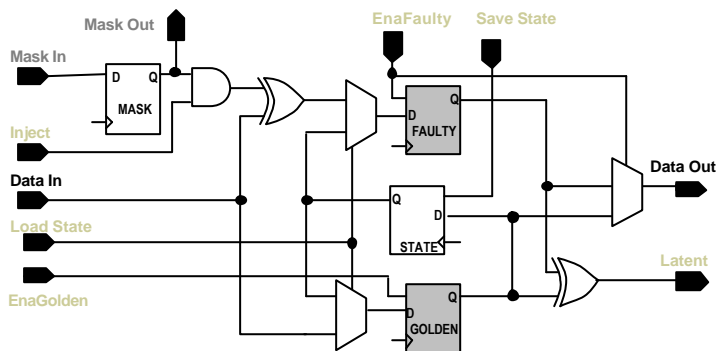
- Load circuit state at fault injection time
- Classify fault as soon as possible



Autonomous Emulation



Optimized circuit instrumentation





Optimized circuit instrumentation

- Use 4 FFs
 - Mask FF: set target FFs for fault injection
 - Golden FF: store golden circuit state
 - Faulty FF: store faulty circuit state
 - State FF: store circuit state previous to fault injection



Optimized circuit instrumentation

- Load circuit state at fault injection time
 - Use State FF to save circuit state before previous fault injection
 - Load required circuit state in a single clock cycle
- Classify fault as soon as possible
 - Compare Faulty FF and Golden FF at every clock cycle
 - Abort execution as soon as fault can be classified:
 - No difference in circuit state => Silent fault
 - Difference at POs => Failure



Conclusions

- Evaluation of the circuit sensitivity to soft errors must be performed for critical applications
- Fault injection in the manufactured circuit is expensive
 - Mainly used for circuit certification
- Fault injection allows detecting the weak areas early in the design process, where correcting actions are easier



A summary example

- Consider the following circuit figures
 - 10^4 memory elements (FFs + RAM bits)
 - Workload: 1ms @ 100 MHz = 10^5 clock cycles
 - Fault space = 10^4 FF x 10^5 CC = 10^9 faults



A summary example

- Typical results of validation test experiments:

Experiment	Expected #events/faults	Test time
While in operation	10^1 (0.0000001 %, 0.01 ppm)	Few weeks / months
Radiation testing	10^2 (0.00001 %, 0.1 ppm)	Few hours
Simulation-based FI	10^4 (0.001 %, 10 ppm)	Few hours
Emulation-based FI	10^4 (0.001 %, 10 ppm)	Few seconds
Autonomous Emulation	10^6 (0.1 %, 1000 ppm)	Few seconds

- As circuit complexity increases, need more efficient methods to evaluate circuit sensitivity



SOFT ERRORS IN DIGITAL CIRCUITS

Luis Entrena
Microelectronic Design & Applications Group
Universidad Carlos III de Madrid