
  
Karlsruhe Institute of Technology

# Reliable Computing I

## Lecture 3: Faults, Errors, Failures

Instructor: Mehdi Tahoori


INSTITUTE OF COMPUTER ENGINEERING (ITEC) – CHAIR FOR DEPENDABLE NANO COMPUTING (CDNC)



KIT – University of the State of Baden-Wuerttemberg and  
National Research Center of the Helmholtz Association

[www.kit.edu](http://www.kit.edu)

## Today's Lecture

  
Karlsruhe Institute of Technology

- Terminology and classification
- Causes of Faults and Trends
- Fault Modeling
  - Hardware
  - Software

(c) 2019, Mehdi Tahoori

Reliable Computing I: Lecture 3

2

## Faults



- **Fault**: incorrect state of hardware or software resulting from physical defect, design flaw, or operator error
- Faults introduced during system design
  - Pentium's incorrect floating point division design
  - Bug in software could cause infinite loop
- Faults introduced during manufacturing
  - Bad solder connection between chip pin and motherboard
  - Broken wire within chip
- Faults that occur during operation
  - Cosmic ray knocks charge off DRAM cell
  - System administrator incorrectly installs new software

## Errors



- **Error**: manifestation of a fault
  - Bit in main memory is a 0 instead of a 1 (due to cosmic ray)
  - Software pointer that mistakenly points to NULL (due to bug)
- But not all faults lead to errors!
  - Trees falling in empty forests don't make sounds
- Examples of **masked** faults
  - Cosmic ray knocks charge off logic signal, but after it had been correctly latched in and saved
  - Buggy software that isn't reached

## Failures

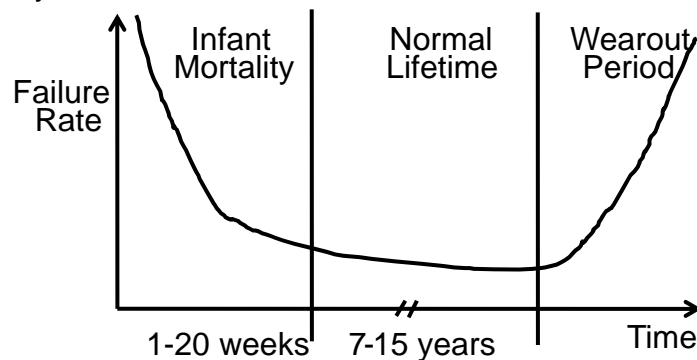


- **Failure**: system level effect of an error (user-visible)
  - System produces incorrect result of computation (e.g.,  $2+2=5$ )
  - System “hangs” (e.g., Blue Screen of Death)
- Not all errors lead to failures!
- Examples of **masked** errors
  - Bit flip in memory location that’s not accessed again
  - NULL pointer that’s not referenced again

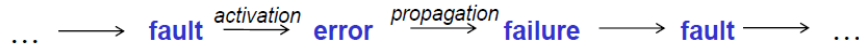
## Physical Failures During Lifetime



- Three phases of system lifetime
  - Infant mortality
  - Normal lifetime
  - Wear-out period
- Physical failures follow famous “bathtub curve”



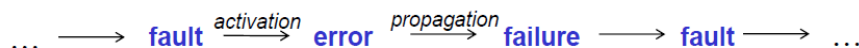
## Fundamental Chain of Dependability



### ■ Example 1

- A short in an integrated circuit is a failure (with respect to the function of the circuit)
- The consequence (e.g., stuck at a Boolean value) is a fault that stays dormant until activated
- Upon activation (invoking the faulty component by applying an input) the fault becomes active and produces an error
- If and when the propagated error affects the delivered service (e.g., information content), a failure occurs

## Fundamental Chain of Dependability



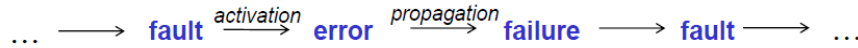
### ■ Example 2

- The result of an error by a programmer leads to a failure to write the correct instruction or data
- This results in a dormant fault in the written software (e.g., faulty instruction)
- Upon activation the fault become active and produces an error
- When the error affects the delivered service , a failure occurs

### ■ Example 3

- An inappropriate human-system interaction performed by an operator is an external fault (from the system view point)
- Resulting altered processed data is an error, .....

## Fundamental Chain of Dependability



### ■ Example 4

- Cosmic ray knocks charge off of DRAM cell
  - Error: bit flip in memory
  - Failure: computation produces incorrect result

### ■ Example 5

- Software bug *could* allow for NULL pointer
  - Bug gets exercised and we get NULL pointer
  - Program seg faults when it tries to access pointer

## Propagation and Masking



- Impact of faults can spread throughout the system
  - If a chip shorts power to ground, it may cause nearby chips to fail as well
  - Common clock and power net
  - Independence of modules is a strong simplification
- Error propagation: Erroneous results used in subsequent computations
  - Containment upon detection important
- Masking
  - Electrical, logical, temporal, behavioral

## Masking



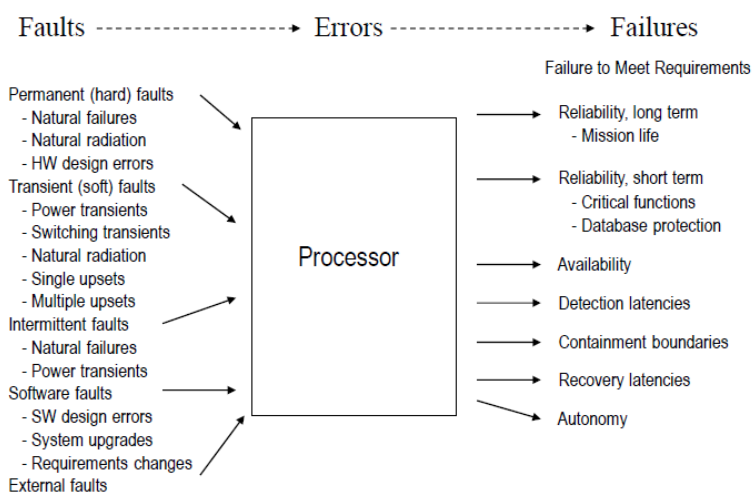
### ■ Logical

- E.g., if a fault flips a bit from 0 to 1 and it is then ANDed with a bit that is 0, then the fault cannot manifest itself as an error


### ■ Functional

- E.g., incorrect data is produced by an instruction that gets squashed due to a branch misprediction
- E.g., the destination register of a NOP is corrupted by a fault

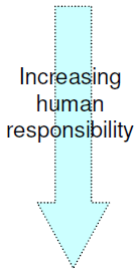
## Faults, Errors, and Failures



## Origin of Defects in Objects



- (hardware or software)
- Good object wearing out with age
  - Hardware (software can age too)
  - Incorrect maintenance/operation
- Good object, unforeseen hostile environment
  - Environmental fault
- Marginal object: occasionally fails in target environment
  - Tight design/bad inputs
- Implementation mistakes
- Specification mistakes




Increasing human responsibility

(c) 2019, Mehdi Tahoori

Reliable Computing I: Lecture 3

13

## Fault Classes: Temporal persistence



- **Permanent** faults, whose presence is continuous and stable.
  - E.g., Broken connection → always open circuit
- **Intermittent** faults, whose presence is only occasional due to unstable hardware or varying hardware and software states (e.g., as a function of load or activity)
  - E.g., Loose connection → occasionally open circuit
  - E.g., Bug in little-used software for rounding → incorrect data
- **Transient** faults, resulting from temporary environmental conditions.
  - E.g., Cosmic ray knocks charge off transistor → bit flip
  - Tend to be due to transient physical phenomena
  - Also known as Single Event Upset (SEU)

(c) 2019, Mehdi Tahoori

Reliable Computing I: Lecture 3

14

## Fault Classes



### ■ Based on the origin

#### ■ Physical faults

- Stemming from physical phenomena internal to the system,
  - such as threshold change, shorts, opens, etc.,
- or from external changes,
  - such as environmental, electromagnetic, vibration, etc.

#### ■ Human-made faults

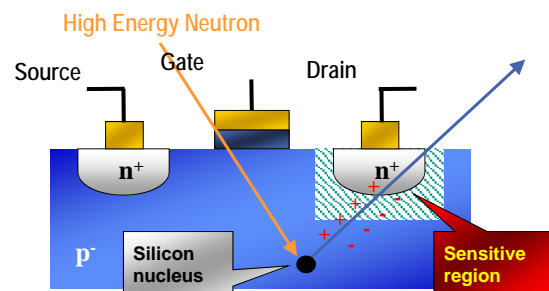
- Design faults,
  - introduced during system design, modification, or establishment of operating procedures,
- Interaction faults,
  - violation of operating or maintenance procedures

## Physical Defects: Transient Phenomena



### ■ Cosmic radiation

- High energy particles that constantly bombard Earth
- May have enough energy to disrupt charge on transistor ( $Q_{crit}$ )
- Used to be only a problem for DRAM, but becoming a problem for SRAM and even for logic (as  $Q_{crit}$  decreases)





## Physical Defects: Transient Phenomena

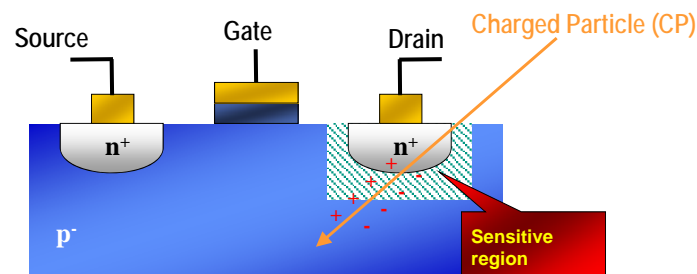


- Cosmic radiation trends:
  - Qcrit decreasing
  - Probability increasing that a cosmic ray that hits a transistor will disrupt its charge
  - Transistor size decreasing → smaller probability that a cosmic ray will hit a particular transistor
  - More transistors per system → greater probability of fault

## Physical Defects: Transient Phenomena



- Alpha particle radiation
  - Similar to cosmic rays, but radiation comes from metal decay
  - Often, the metal housing of the computer is the source
  - Lead solder joints also a problem → want to use “old lead”
  - Trends (same as for cosmic radiation):



## Physical Defects: Transient Phenomena




- Electromagnetic Interference (EMI)
  - Electromagnetic waves from other sources (e.g., microwave oven, power lines, etc.) can cause transient disruptions
  - EMI can be created by the circuit itself! Called “crosstalk”
  - EMI can induce electrical current on wires and thus change the signals on wires
- There are other sources of transient faults, but they tend to be less significant

## Physical Defects: Manufacturing Defects

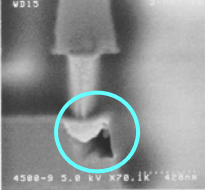


- Manufacturing is not a perfect process, especially for microprocessors
  - It's not easy to manufacture something with dimensions on the order of 10nm
  - Many stages of chip processing which have to be done perfectly and avoid contamination
- And testing doesn't filter out all defective systems
  - Often impossible to test for every possible defect in a reasonable amount of time
  - Also, testing won't detect defects that don't manifest immediately
- Nanotechnology makes this problem even worse

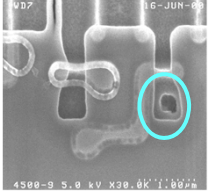
## Evidence of Manufacturing Defects



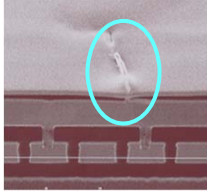
**Void under anchor**



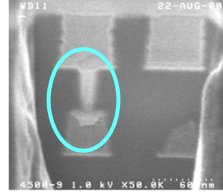
**Silicon damage**



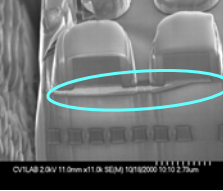
**Metal2 extrusion/  
ILD2 crack**



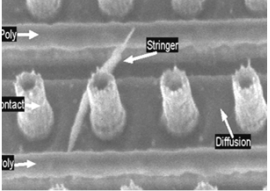
**Metal 1 Shelving**



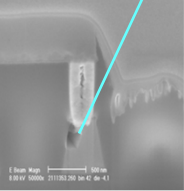
**M4-M4 Short**

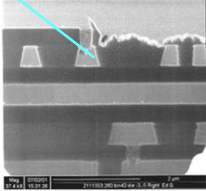


**Poly stringer**



**M4 Void Formations**






(c) 2019, Mehdi Tahoori

Reliable Computing I: Lecture 3

21

## Physical Defects: Manufacturing Defects



- Manufacturing flaws
  - Bad solder connection between chip and board
  - VLSI defects
  - Trends:
    - Flaws may decrease as manufacturing process matures
    - But flaws increase at start of each new process
    - Tougher to avoid VLSI defects as dimensions shrink

(c) 2019, Mehdi Tahoori

Reliable Computing I: Lecture 3

22

## Physical Defects: Manufacturing Defects



- VLSI fabrication process variability
  - During fab, there's some amount of variability in dimensions
    - Thickness of gate oxide dielectric
    - Length of channel
    - Area of via
- Variability can lead to undesirable behavior
  - Gate thickness falls below usable threshold → extra leakage current
  - Wire resistance is too high → signal too slow for clock
- Trend: variability rising as VLSI dimensions shrink
  - When dimensions are on the order of a handful of atoms, it doesn't take much variability to cause significant problems

## Physical Defects: Operational Defects




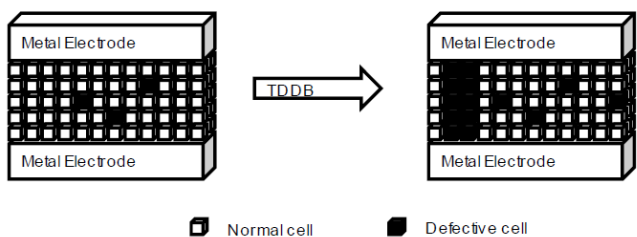
- Permanent (hard) defects can occur during operation
- Electromigration
  - Movement of metal atoms due to electron flow and temperature
    - Increases with current density and temperature
    - Unidirectional current: Power rails
  - Trend: getting worse as wires become smaller and chips become hotter



## Physical Defects: Operational Defects

- Time Dependent Dielectric Breakdown (TDDB)
  - MOSFET transistor has a gate oxide that insulates the gate from the channel
  - If this oxide breaks down, will get a short between gate and channel
  - Trend: getting worse as gate oxides become thinner (only a handful of atoms thick!)






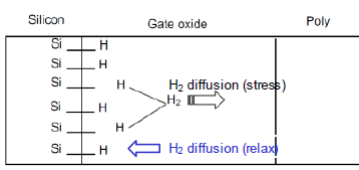
Normal cell
 Defective cell

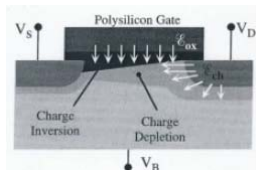
(c) 2019, Mehdi Tahoori
Reliable Computing I: Lecture 3
25

## Physical Defects: Operational Defects

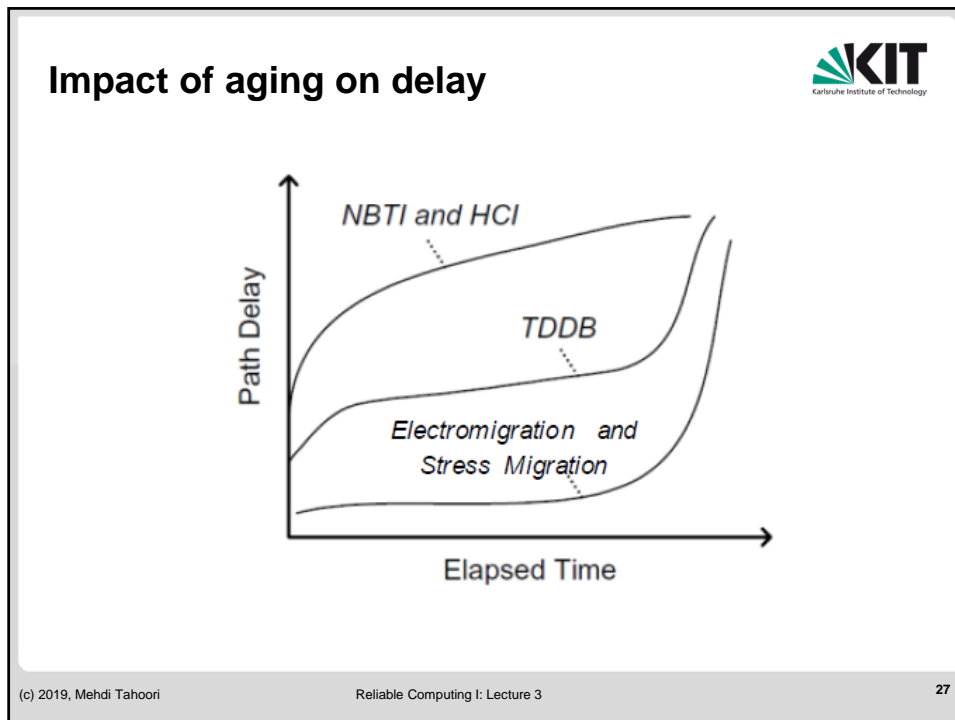
- Transistor aging
  - Causes
    - Negative and Positive Bias Temperature Instability (NBTI and PBTI)
    - Hot Carrier Injection (HCI)
  - Effects:
    - Change of transistor's threshold voltage over time → Reduced current → Transistors become slower → cause timing failures
  - Trend: getting worse with technology scaling








(c) 2019, Mehdi Tahoori
Reliable Computing I: Lecture 3
26



### Hardware Design Flaws: Logical Bugs



- Famous examples:
  - Intel Pentium floating point divide didn't work in every single case due to bug in design
    - very costly recall
  - Sun UltraSPARC III had design flaw in a special cache that meant that it couldn't be used
    - loss in performance
  - AMD's quad-core Barcelona chip had design bug in TLB hardware
    - Long, expensive delay in shipping chips

(c) 2019, Mehdi Tahoori      Reliable Computing I: Lecture 3      28

## Hardware Design Flaws: Timing Bugs



- Logic is fine, but the timing analysis is flawed
- Example: clocking a processor at 4 GHz when there's a slow path in the pipeline that can only run at 3.8 GHz
- Timing analysis must consider critical path delay and environmental effects (operating temperature, EMI, cross-talk, etc.) to determine the maximum operating speed
- This problem is exacerbated by process variability

## Design Flaws: Software



- We all know that software has bugs
- Types of bugs
  - Incorrect algorithm
  - Memory leak (C, C++, but not Java)
    - Allocating memory, but not deallocating it
  - Reference to NULL pointer (C, C++, but not Java)
    - This usually leads to a seg fault and core dump
  - Incorrect synchronization in multithreaded code
  - Allowing more than 1 thread in critical section at a time

## Operator Error



- It has been argued that operator error is the leading cause of computer system failures
- Examples
  - `rm -R *` (in the wrong directory)
  - Incorrect installation of software
  - Frying a board when installing new memory chips
  - Dropping the laptop (and/or kicking it)

## Purpose of Fault Modeling



- Model = abstraction of physical phenomenon
- Simple, tractable way to analyze effects of faults
- Limitations
  - Model multiple defects (loss of resolution)
  - May not distinguish defects or miss defects
  - May not be realistic





## Fault Modeling: Example



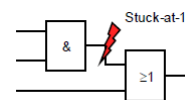
- “fail-stop” network switch
  - if a fault occurs, the switch will just stop doing anything
- Model reflects the behavior of many potential underlying faults
  - E.g., switch has short from power to ground, switch is on fire, etc.
- Easier to work with this model than to consider all possible faults
- Fail-stop fault model for network switch doesn't handle case where switch starts routing packets incorrectly
  - And this fault model represents several realistic underlying faults

## Fault Models for Digital Circuits



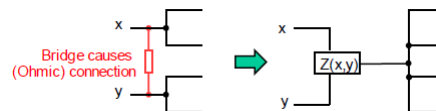
### Traditional fault models

- Stuck-at faults
  - A line is always at a fixed value (1 or 0)



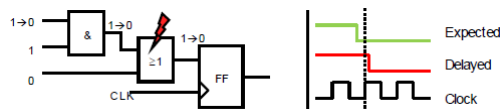
### Bridges

- Lines shorted together
- $Z(X,Y)$ : modeled as OR, AND



### Transition delay faults


- Transition arrives too late
- Slow-to-rise, slow-to-fall

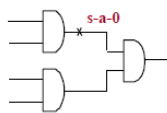
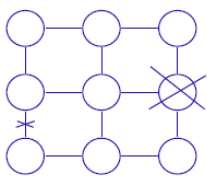


### Fault models for nano-scale circuits

- Crosstalk, small/path delay faults, resistive opens and bridges


## Hardware Fault Models



Stack-at	Module level	Functional level	System level
<p><b>Example:</b> physical failures in circuits</p> <p>Lines in a gate level stuck at 0 or 1</p> <p>Faulty contact</p> <p>Transistor stuck open or closed</p> <p>Metal lines open</p> <p>Shorts between adjacent metal lines</p> 	<p><b>Example:</b> decoder</p> <p>No output lines activated</p> <p>An incorrect line activated instead of desired line</p> <p>An incorrect line activated in addition to desired line</p>	<p><b>Example:</b> Memories</p> <p>One or more cells are stuck at 0 or 1</p> <p>One or more cells fail to undergo 0-1 or 1-0 transition</p> <p>Two or more cells are coupled</p> <p>A 1-0 transition in one cell changes contents in another cell</p> <p>More than one cell is accessed during READ or WRITE</p> <p>A wrong cell is accessed during READ or WRITE</p>	<p><b>Example:</b> a parallel processor topology</p> <p>View machine as a graph</p> <ul style="list-style-type: none"> <li>- nodes correspond to processors</li> <li>- edges correspond to links</li> </ul> <p><b>Fault Model:</b></p> <p>A processor (node) or link (edge) faulty</p> 

(c) 2019, Mehdi Tahoori
Reliable Computing I: Lecture 3
35

## How Many Faults at Once?



- Many fault models include the assumption that only one fault can occur at a given instance
  - Helps to make analysis more tractable
  - E.g., “single stuck-at fault” model
- Reasonable assumption if:
  - Faults are rare
  - System doesn't require extreme reliability
  - **Faults are detected and, if necessary, removed quickly**
- The problem with latent faults
  - Fault occurs, but isn't detected
  - Later, a “single” fault occurs, but this is now a double fault scenario
  - If you only plan for single faults, then this situation is a problem

(c) 2019, Mehdi Tahoori
Reliable Computing I: Lecture 3
36

## Software Fault Models



- **Allocation management** : Memory region used after deallocation
- **Copying overrun**: Program copies data past end of a buffer
- **Pointer management**: Variable containing data address corrupted
- **Wrong algorithm**: Program works executes but uses wrong algorithm
- **Uninitialized variable**: Variable used before initialization
- **Undefined state**: System goes into unanticipated state
- **Data error**: Program produces or reads wrong data
- **Statement logic**: Statements executed in wrong order or omitted
- **Interface error**: A module's interface incorrectly defined or incorrectly used
- **Memory leak**: Program does not deallocate memory it has allocated
- **Synchronization**: Error in locking or synchronization code

## Software Fault Models



- **Incorrect computation**: Arithmetic overflow or an incorrect arithmetic function
- **Data fault**: Incorrect constant or variable
- **Data definition fault**: Fault in declaring data or data structure
- **Missing operation**: Omission of a few lines of source code
- **Side effect of code update**: Not all dependencies between software modules considered when updating software
- **Unexpected situation**: Not providing routines to handle rare but legitimate operational scenarios