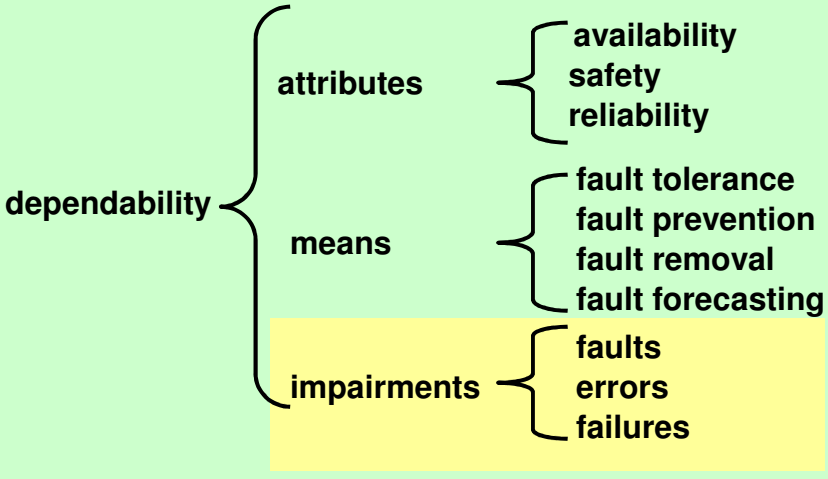
KUNGL TEKNISKA HÖGSKOLAN

International Master Program in System-on-Chip Design

Faults, Errors and Failures

Dependability tree



Examples of failures

- eBay Crash
- Ariane 5 Rocket Crash

eBay Crash

- eBay: giant internet auction house
  - A top 10 internet business
  - Market value of $22 billion
  - 3.8 million users as of March 1999
  - Access allowed 24 hours 7 days a week
- June 6, 1999
  - eBay system is unavailable for 22 hours with problems ongoing for several days
  - Stock drops by 6.5%, $3-5 billion lost revenues
  - Problems blamed on Sun server software

Ariane 5 Rocket Crash

- Ariane 5 rocket exploided 37 seconds after lift-off on June 4, 1996
- Error due to software bug:
  - Conversion of a 64-bit floating point number to a 16-bit integer resulted in an overflow
  - In response to the overflow, the computer cleared its memory
  - Ariane 5 interpreted the memory dump as an instruction to its rocket nozzles
- Testing of full system under actual conditions not done due to budget limits
- Estimated cost: 60 million $

Fault

Fault is a physical defect, imperfection or flaw that occurs in hardware or software

Example: - short between wires

- break in transistor

- infinite program loop

Error

Error is a deviation from correctness or accuracy

Example: Suppose a line is physically shortened to 0 (there is a fault). As long as the value on line is supposed to be 0, there is no error.

Errors are usually associated with incorrect values in the system state.

Failure

Failure is a non-performance of some action that is due or expected

Example: Suppose a circuit controls a lamp (0 = turn off, 1 = turn on) and the output is physically shortened to 0 (there is a fault). As long as the user wants the lamp off, there is no failure.

A system is said to have a failure if the service it delivers to the user deviates from compliance with the system specification.

Cause-and-effect relationship

- Faults can result in errors. Errors can lead to system failures
- Errors are the effect of faults. Failures are the effect of errors

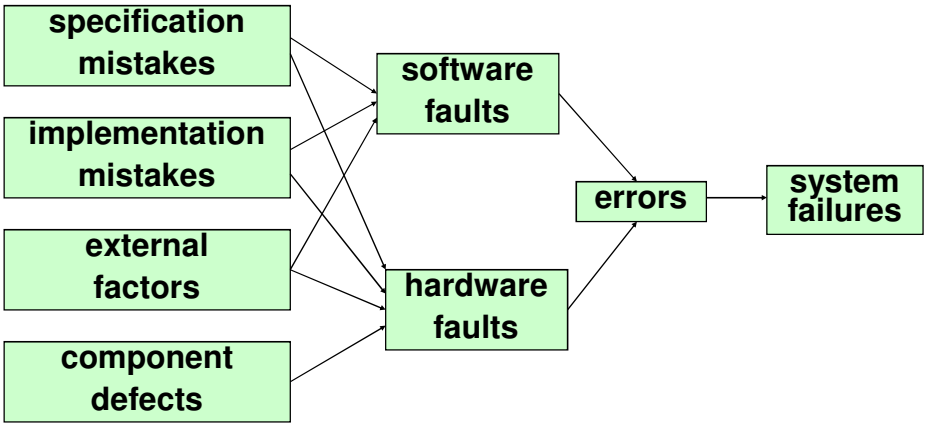
Software

- Definitions of physical, computational and system levels are more confusing when applied to software
  - physical level = program code
  - computational level = values of the program state
  - system level = software system running the program
- Bug in a program is a fault. Possible incorrect values caused by this bug is an error. Possible crush of the operating system is a failure.

Origins of faults

- specification mistakes
  - incorrect algorithms, incorrectly specified requirements (timing, power, environmental)
- implementation mistakes
  - poor design, software coding mistakes
- component defects
  - manufacturing imperfections, random device defects, components wear-outs
- external factors
  - radiation, lightning, operator mistakes

Cause-and-effect relationship



Common-mode faults

- A common-mode fault is a fault which occur simultaneously in two or more redundant components
- Caused by phenomena that create dependancies between components
  - common communication bus
  - shared environmental conditions
  - common source of power
  - design mistake
- Design diversity is the implementation of one or more variant of the redundant component

Hardware faults

- Fault duration specifies the length of time that a fault is active
  - permanent fault
    - remains in existence indefinitely if no corrective action is taken (stuck-at fault)
  - transient fault
    - can appear and disappear within a very short period of time (caused by lightning)
  - intermittent fault
    - appear, disappears and then reappears repeatedly (weak solder joint)

Fault models

- It is very difficult to analyze a system without assuming some fault models
  - hard to design test procedures
  - hard to simulate faults
- To make the problem more manageable, we need to restrict our attention to a subset of all faults what can occur

Fault models

- Fault model is a logical abstraction describing the functional effect of physical defect
- Different levels of modeling
  - high, logic, transistor, layout
- Different fault models
  - stuck-at, transition, coupling

Logic stuck-at fault model

- most commonly used model
- the effect of the fault is modeled by having a line in the circuit permanently fixed to 0 or 1 value
- the basic functionality of the circuit is not changed
  - gates remain the same
  - combinational circuit is not transformed to sequential

Test set

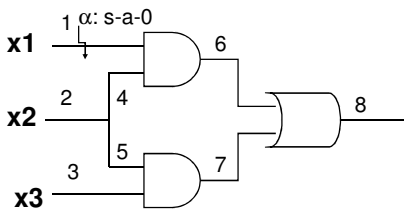
- Test for a given fault is an assignment of values for input variables, detecting this fault
- Complete test set is a set of tests detecting all faults in the circuit (of a specified type)
- Minimal complete test set is a complete test set with the minimal number of tests

Truth-table based method for finding tests for stuck-at faults

To find tests for some stuck-at fault α:

- Write truth tables for the function without fault, f, and the function with fault, f^{α}
- All input assignments of the truth table for which $f \neq f^{\alpha}$ are tests for the fault α

Example



f = x1x2 + x2x3

f^{α} = x2x3

x_1	x_2	x_3	f	f^{α}
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

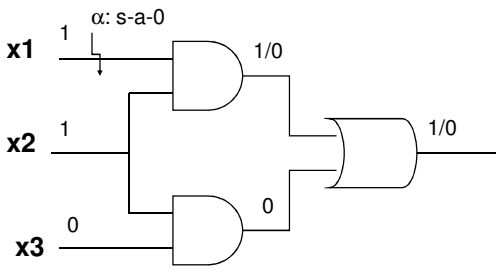
test for α

Circuit-based method for finding tests for stuck-at faults

To find tests for a stuck-at fault α on some line *i*:

- Put on *i* a value opposite to α
- Make the output sensitive to *i* by selecting a path and assigning values to other inputs of gates along this path (1 for AND, 0 for OR, don't care for XOR)
- Try to assign values to other gates in the circuit so that there are no conflicts
- If not possible, choose another path
- If not possible for all paths, α is undetectable

Example



(110) is the test for α

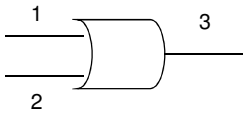
There are no other tests for α

1/0 means that the value is 1 in fault-free circuit and 0 in faulty circuit

How to find a complete minimal test set

- Find all tests for all stuck-at faults in the circuit
- Make a table
  - One row for each fault (2·number of lines)
  - One column for each test (2^n, n = number of inputs)
- Put a star is a test detects a fault
- Select a minimal number of tests which detect all faults (i.e. choose a minimal subset of columns which covers all rows)

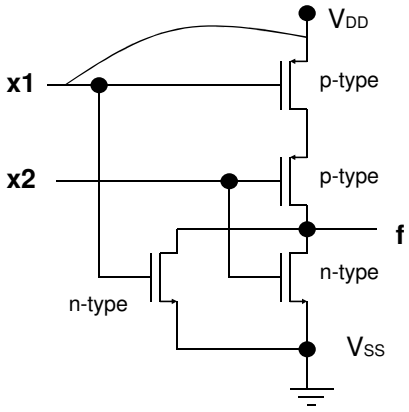
Example

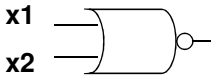


	00	01	10	11
1: s-a-0			*	
1: s-a-1	*			
2: s-a-0		*		
2: s-a-1	*			
3: s-a-0		*	*	*
3: s-a-1	*			

The complete minimal test set is {(00),(01),(10)}

Examples of a faults covered by stuck-at fault model



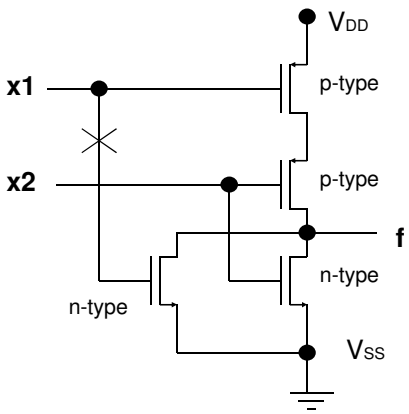


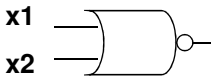
f = (x1 + x2)'

1) The fault caused by x1 shorted to Vdd can be modeled as stuck-at-1 fault at x1.

2) The fault caused by the drain and source of one of the n-type transistors shorted together can be modeled as stuck-at-0 fault at the output.

Example of a fault not covered by stuck-at fault model





f = (x1 + x2)'

The fault cause by the marked broken line cannot be modeled by stuck-at fault model. If the input combination x1x2 = 10 is applied, neither n-type nor p-type transistors are conducting. The output remains in the state defined by the previous inputs (sequential behavior).

Transition fault

- A line in a circuit or a cell in a memory cannot change from a particular state to another state
  - suppose a memory cell containes a 0
  - a 1 is written in the memory successfully
  - if a 0 is attempted to be written to the cell, the cell remains 1
  - there is a 1-to-0 transition fault

Coupling fault

- depend on more than one line
  - short-circuit between two adjacent word lines in a memory
  - writing a value to a memory cell connected to one word line also results in writing that value to the memory cell connected to the other word line
- More difficult to test compared to stuck-at and transition faults

Software faults

- Software differs from hardware in several aspects:
  - it does not age or wear out
  - it cannot be deformed or broken
  - it cannot be affected by environmental factors
  - if deterministic, it always performs the same way in the same circuimstances

Software faults

- Software may undergo several upgrades during system life cycle
  - reliability upgrade – aims to enhace software reliability of security. Done by re-designing some modules using better approaches
  - feature upgrade – aims to enhace software functionality. Likely to increase complexity and thus decrease reliability by introducing new bugs

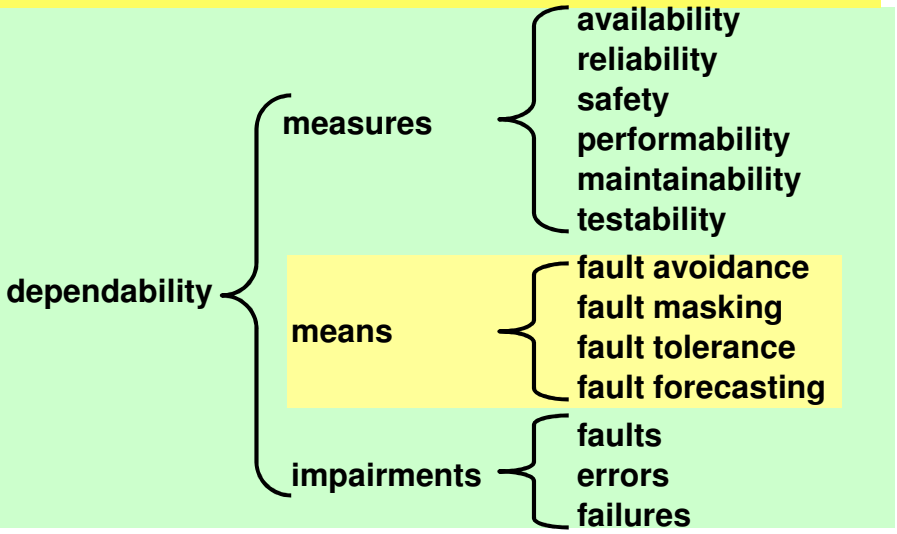
Software faults

- Fixing bugs does not necessarily make software more reliable
  - new bugs may be introduced
    - in 1991, a change f 3 lines of code in a program containing millions lines of code caused a local telephone system in California to stop
- Software is inherently more complex and less regular than hardware
  - achieving sufficient verification coverage is very difficult

Statistic

- 60-65% of software faults originate from
  - incomplete, missing, inadequate, inconsistent, unclear requirements
- 35-40% of software faults originate from
  - coding mistakes
  - proportional to
    - size of code
    - number of paths in code

Dependability tree



Dependability means

- Dependability means are techniques enabling the development of a dependable system:
  - fault tolerance
  - fault prevention
  - fault removal
  - fault forecasting

Fault prevention

- avoid occurrence or introduction of faults
- quality control methods to avoid specification or implementation mistakes and component defects
  - design reviewes
  - component screening
  - testing

Fault prevention

- human-made faults
  - can be reduced by training
  - or by decreasing the amount of information

Fault prevention

- software design faults
  - structured programming, well-defined interface
  - modularization
  - extensive testing in realistic environment
  - formal verification
  - re-use old software
- deliberate malicious faults caused by viruses or hackers
  - firewalls or other security means

Fault prevention

- transient hardware faults
  - prevent external disturbances
    - shielding, grounding
  - power problems
    - filter, separate distribution
  - α -radiation
    - radiation-tolerant components

Fault prevention

- intermittent hardware faults
  - overheating
    - ventilation
  - bad contacts
    - avoid vibrations
  - metastability (oscillation between 0 and 1)
    - good synchronisation

Fault prevention

- permanent hardware faults
  - component failure
    - burn-in
      (H/L temperature, H/L humidity, vibrate)
    - avoid extreme conditions
    - early replacement
  - power supply failures
    - UPS (uninterruptable power supply), for life-critical applications, have a battery
- design faults
  - modularity and testing

Fault removal

- Performed during the development stage as well as during the operational life of a system
  - development stage:
    - verification, diagnosis and correction
  - operational stage:
    - corrective and preventive maintenance

Fault forecasting

- estimate faults
  - present number
  - future number
  - consequences
- qualitatively
  - search for causes of faults
- quantitatively
  - failure rate, time to failure, time between failures

Fault tolerance

- targets development of a system which functions correctly in presence of faults
- achieved by some kind of redundancy
  - redundancy allows either to detect or to mask a fault
- Fault detection/masking are followed by fault location, containment and recovery
  - the goal is to reconfigure system to remove faulty components

Fault detection

Fault detection is the process of recognising that a fault has occurred

Fault location

Fault location is the process of determining where a fault has occurred

Fault containment

Fault containment is the process of isolating a fault and preventing its effect to propagate throughout a system

Fault recovery

Fault recovery is the process of regaining operational status

Summary

- fault detection
  - identify that a fault has occurred
- fault location
  - find where the fault is
- fault containment
  - prevent propagation of the fault
- fault recovery
  - modify structure to remove faulty component
  - graceful degradation – continue operation with a degraded performance

Next lecture

- Evaluation techniques

Read chapter 3

of the text book