Dependable Systems

Hardware Dependability - Testing

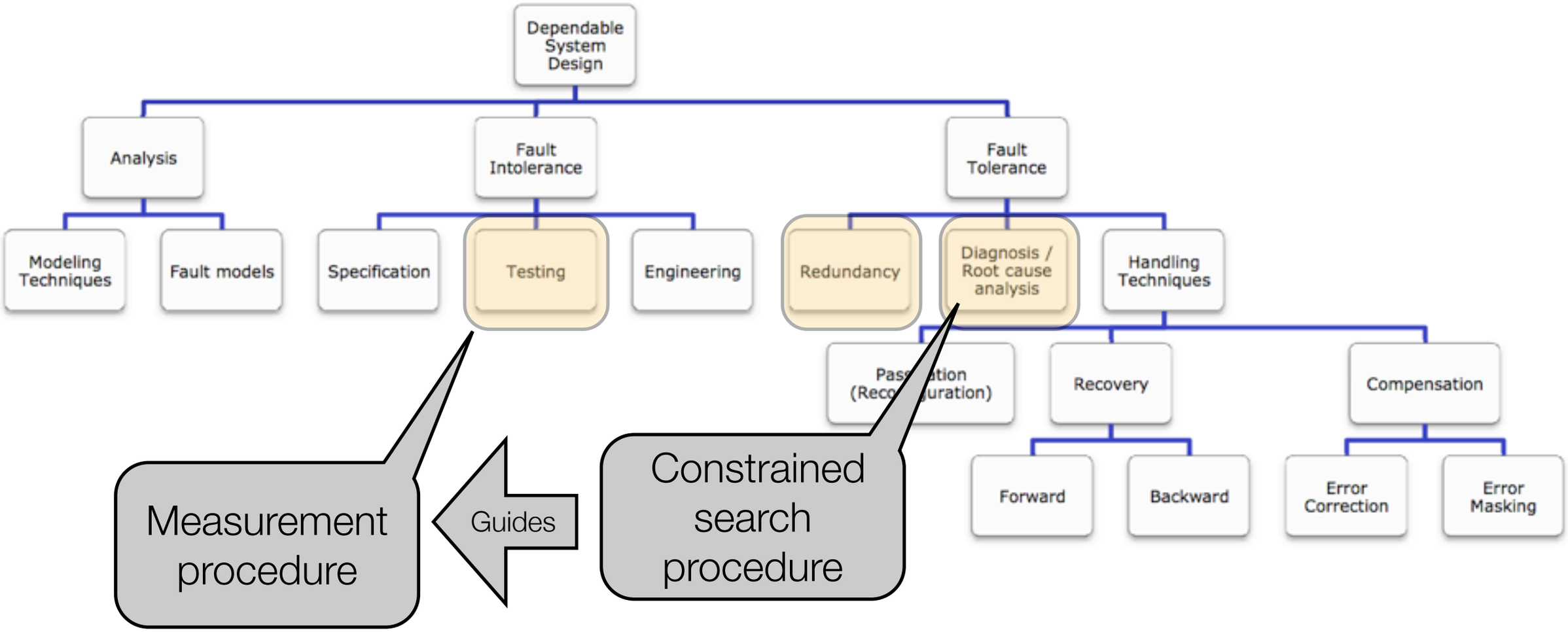
Dr. Peter Tröger

Sources:

Siewiorek, Daniel P.; Swarz, Robert S.:
Reliable Computer Systems. third. Wellesley, MA : A. K. Peters, Ltd., 1998. ,
156881092X

Ziade, Haissam; Ayoubi, Rafic A.; Velazco, Raoul:
A Survey on Fault Injection Techniques. In: Int. Arab J. Inf. Technol. 1 (2004), Nr. 2,
S. 171-186

Dependable System Design (Echtle)



Testing [Sieworek & Swartz]

- Core of specification-based diagnosis is testing - black box approach
- Type of fault influences test procedure - logical fault vs. structural fault
- Development phase influences test procedure - functional vs. acceptance test

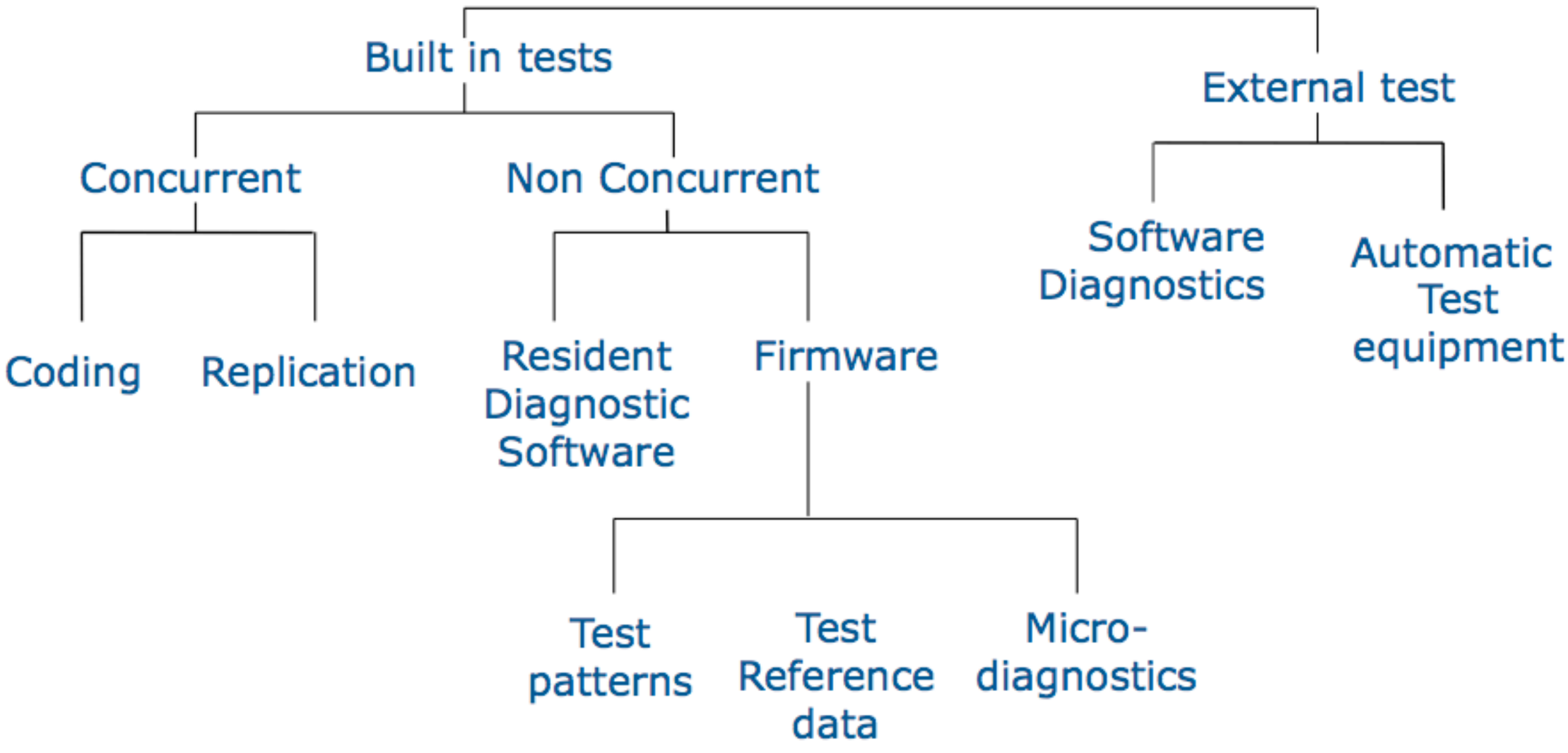
Hierarchical Level	Design	Production	Operational
System	Design maturity test - Reach specification goals	Process maturity test	Synthetic load
Logic	Simulation	Acceptance test	Built-in test
Circuit	Simulation	Parametric test, up to stress tests	Margining

Temporal Stage

Hardware Testing

- What makes hardware testing today difficult and expensive ?
  - Large number of faults and fault classes
  - Limited observability and controllability
  - Pattern sensitivity in large density circuits
  - Increasing circuit speeds
  - Increasing system complexity
  - Exponentially growing number of test patterns
  - Incomplete information about a system
    - Companies often do not disclose system description

Taxonomy of Computer Test Approaches



Hardware Design for Testability

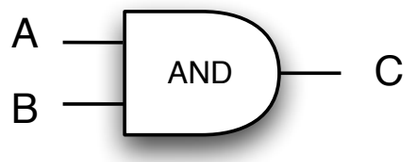
- Testing problem - Test generation vs. test verification
  - No formal proof so far, fault simulation as option for measuring test effectivness
- Design for testability
  - Accept the risk of shipping a defect product, but support testing in a better way
    - Ad-hoc approaches: Partitioning, extra test points at board level, bus systems
    - Structured approaches: Allow observability for state variables in the system
- Stress testing approaches for production phase to force infant mortalities
  - Vibration, over-voltage, burn-in, thermal shock, ...
- Largest body of theory exists for logic-level acceptance testing
  - Usually, single structural stuck-at faults are assumed
  - Define unit under test, stimulus can be on-chip / off-chip

Hardware Design for Testability

- Some recommendations
  - Allow all memory elements to be initialized before testing begins, preferably via a single reset line
  - Provide means for opening feedback loops during testing
  - Allow external access to the clock circuits to permit the tester to synchronize with, or disable, the unit under test
  - Insert multiplexers to increase the number of internal points which can be controlled or observed from the external pins
  - Use synchronous circuitry whenever possible
- With more complexity, built-in tests (per component) become attractive
  - Coding or self-testing hardware

Processor Testing

- Typical approach is logic testing - over 50.000 papers
- One of the popular approaches are enhanced D-algorithms (branch-and-bound)
  - First algorithm for test generation that was designed to be programmable
  - Developed by Roth at IBM in 1966, based on D notation, single stuck-at fault
    - D - „1“ in the good circuit, „0“ in the faulty
    - D‘ - „0“ in the good circuit, „1“ in the faulty
  - Primitive D cube of failure (PDCF) - set of input that will trigger a failure
  - Propagation D cube - set of inputs that will propagate a D to the outputs



Fault	PDCF		
	A	B	C
A stuck-at-0	1	1	D
B stuck-at-0	1	1	D
C stuck-at-0	1	1	D
A stuck-at-1	0	1	D‘
B stuck-at-1	1	0	D‘
C stuck-at-1	X	0	D‘

Processor Testing

- Functional testing (Thatte/Abraham, 1978)
  - Data flow graph model
    - Nodes represent registers or active units, links correspond to data transfer paths in the processor hardware
    - Three classes of instructions (fault model is defined for each class)
      - Sequencing & control
      - Data storage and transfer
        - Example fault model: Any number of lines can be stuck-at-1 or 0
      - Data manipulation
- Functional testing is usually supplemented by additional pseudorandom tests

Memory Testing

- More and more important issue
  - Key component for electronic systems, embedded memory eats most transistors
  - About 30% of the semiconductor market
- Multitude of memory testing approaches
  - Caching problem
    - Memory testing is performed by the processor
    - Disable caches or use appropriate techniques, such as modulo
  - Layout problem
    - Testing procedure must consider organization of RAM hardware
- Testing time vs. fault coverage trade off, additional runtime monitoring

Fault Model for Semiconductor Memories

- **Stuck-at**-1 or stuck-at-0 (hard) faults, **transition faults** (0->1, 1->0)
- **Open and short circuits** - Too much or too little metallization; Also open bonds
- **Input and output leakage** - Leakage current in excess of the specified limit
- **Multiple writing** - Data written into more than one cell when writing into one cell
- **Pattern sensitivity** - Device does not perform reliably with certain test pattern(s)
- **Refresh dysfunction** - Data are lost during the specified minimum refresh time
- **Write recovery** - Write followed by reading/writing at different location resulting in reading/writing at same location
- **Sense amplifier recovery** - Data accessed for a number of cycles are the same and then suddenly changed, sense amplifier tends to stay in the same state
- **Sleeping sickness** - Memory loses information in less than the stated hold time (typically tens of milliseconds)

Fault Model for Semiconductor Memories

- **Decoder malfunction** - Inability to address same portions of the array
  - No cell accessed by certain address, multiple cells accessed by certain address
  - Certain cell not accessed by any address
  - Certain cell accessed by multiple addresses
- **Bridging fault** - Short between cells, AND type or OR type
- **State coupling fault** - Coupled (victim) cell is forced to 0 or 1 if coupling (aggressor) cell is in given state
- **Inversion coupling fault** - Transition in coupling cell inverts coupled cell
- **Idempotent coupling fault** - Coupled cell is forced to 0 or 1 if coupling cell transits from 0 to 1 or 1 to 0
- **Disturb fault** - Victim cell forced to 0 or 1 if we read or write aggressor cell (may be the same cell)

RAM Testing

- Categories: Functional (1‘s and 0‘s), DC parametric (signal timing, fall and rise) and AC parametric (access times, setup and hold times and cycle times)
- Exhaustive test
  - Total number of bit configurations is 2^N
  - N! potential address sequences
  - 16 bit RAM
    - 65536 words possible
    - 2.092279e+13 possible sequential writing sequences
    - Total of 1.371196e+18 combinations !

Test	Complexity
MSCAN	4n
Column Bars	4n
Checkerboard	4n
Marching 1‘s / 0‘s	12n
Shifted Diagonal	$4n^{3/2}$
Ping Pong	n^2
Walking 1‘s / 0‘s	$2n^2 + 6n$
Galloping 1‘s & 0‘s	$2n^2 + 8n$

RAM Testing

- **Memory Scan (MSCAN)**
  - For all cells: Write 0, read, write 1, read
  - Can detect any stuck-at fault in the memory, memory data register, or logic
  - Will not detect stuck-at in the memory address register or in the decoder
- **Checkerboard / Volatility Test Pattern**
  - Write 1‘s in all even locations and 0‘s in all odd locations
  - Wait
  - Read and compare all
  - Repeat with complementary pattern
  - Can check for hold time in dynamic memories

RAM Testing

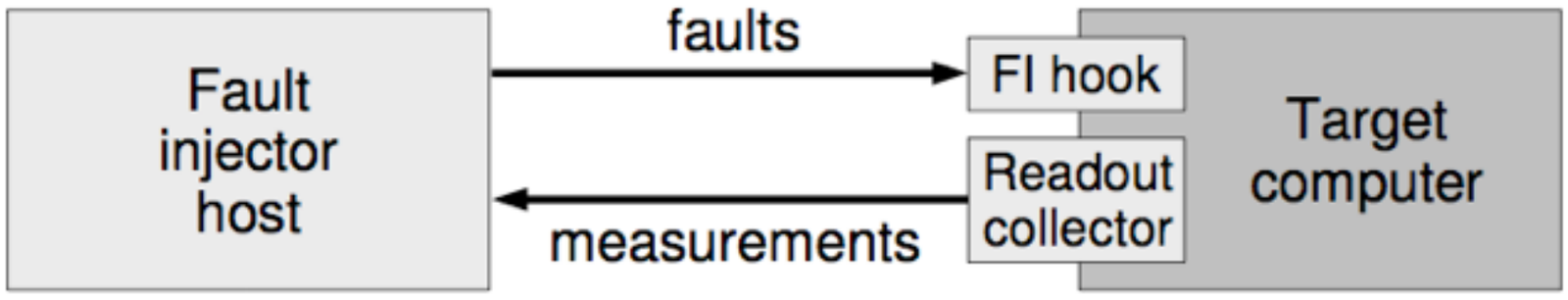
- **Marching 1‘s / 0‘s**
  - Step 1 - For all cells: Write 0
  - Step 2 - For all cells: Read and compare, write 1, read and compare
  - Step 3 - For all cells backward: Read and compare, write 0, read and compare
  - Step 4 - Repeat steps 1 to 3 with complementary pattern
  - Detection of many decoder errors, minimal check on cell interaction
- **Ping Pong**
  - Chose designated test cell, test read / write interaction with all other cells
  - Detect pattern sensity and the interaction between pair cells

RAM Testing

- **Walking 1‘s / 0‘s**
  - Step 1 - For all cells: Write 0
  - Step 2 - For all cells: Write 1 to test cell, read and compare all others, read and compare test cell, write 0 to test cell
    - Checks for independence of cell operations and that decoder addressing works
    - Detects also sense amplifier problems
- **Galloping 1‘s / 0‘s**
  - Same as above, but read 0 test is always followed by read 1 test in test cell
- Complexity Problem: 2 GB RAM = $2*10^9$ Bytes = $16*10^9$ Bits
  - Time effort with checkerboard (4n) and 10ns access time: 10,6 min
  - Reduction by chip-level parallel tests

Fault Injection / Insertion

- Typically low failure rates in standard hardware systems
  - Testing procedures (especially for fault tolerance mechanisms) might take an unacceptable amount of time
- Idea: Accelerate the failure rate by fault injection (in hardware or software)
  - Usually controlled from a second computer
  - FI hook demands trigger for injection process
  - Readout collector obtains data (and verifies injection activity)



Fault Injection Advantages [Ziade et al.]

- Understanding of the effects of real faults and the related system behavior
- Assessment of the efficiency of fault tolerance mechanisms (design faults)
- Encompassing a measurement of the coverage of the fault tolerance mechanism
- Estimate failure coverage and latency
- Explore effects of different workloads on the effectiveness of the FT mechanisms
- Study fault propagation and degree of error containment
- Prove fault model correctness and coverage

Fault Injection

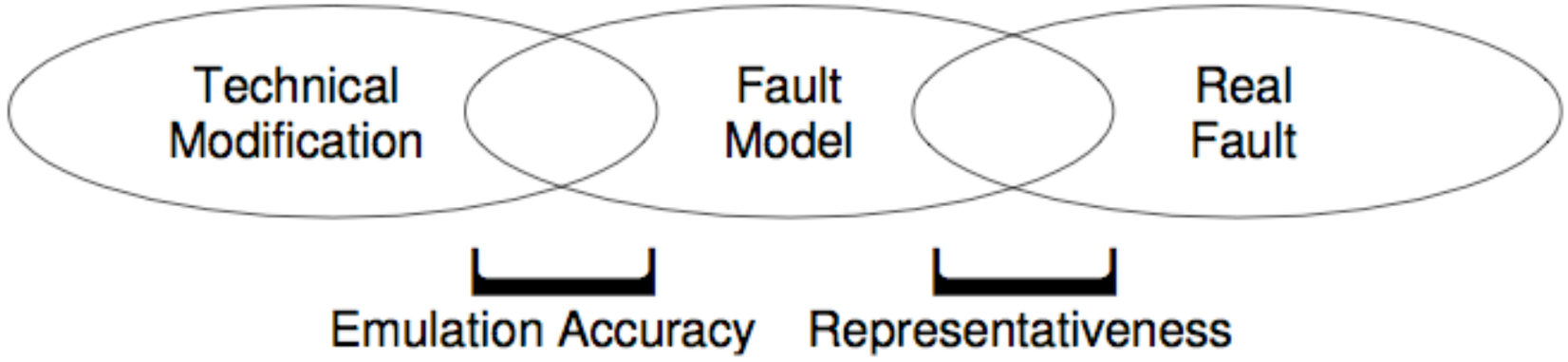
- Typically low failure rates in standard hardware systems
  - Testing procedures (especially for fault tolerance mechanisms) might take an unacceptable amount of time
- Idea: Accelerate the failure rate by fault injection (in hardware and software)
  - Hardware fault injection - With contact (putting unspecified voltages to pins, ...) or without contact (radiation, heat, particle beams, interferences, laser, ...)
  - Simulation-based methods - In high-level models (e.g. VHDL)
  - Emulation-based methods - Use of FPGAs for effective circuit simulation
  - SoftWare-Implemented Fault Injection (SWIFI) - Modification of real system to „emulate“ faults
  - Hybrid approaches - Mix hardware injection with software-based monitoring
- Another categorization: Invasive vs. non-invasive

(HW) Fault Insertion Methods [Sieworek & Swartz]

	Method			
	Software Simulation	Hardware Emulation	Fault Emulation	Physical Insertion
Pro	Access to system at any detail level, fault types and control are unlimited	Representative hardware with favorable access and monitoring	True hardware and software in use	True hardware and software in use
Cons	Simulation time explosion	Implementation and other parameters will change with deployed system	Fault types are limited	Hardware form factor limits access and monitoring points, difficult

Faults vs. Fault Injection

- The set of injected faults usually does not cover the set of targeted real faults
- The fault injection typically can only partly emulate the selected fault model
- The fault model usually covers only parts of the relevant real faults



- Testing with fault injection
  - Problem of repeatability and dependability chain analysis

Hardware-Based Fault Injection

- With contact - Injector has direct physical contact with the target system
  - Pin-level active probes for stuck-at and bridging (probe across two pins) faults
  - Socket insertion for stuck-at, open, or more complex logic faults
    - Inverting signal; Logical combination with other pins or previous signals
    - Supports transient / permanent / random / non-random faults
- Without contact - External source produces physical phenomenon for disturbances
- Benefits
  - Unique locations, supports high resolution systems, accuracy, fast experiments, broad support for fault classes, support for permanent faults
- Drawbacks
  - High risk of damage, dense packaging issues, low portability and observability, high setup time, expensive

Simulation-Based Fault Injection

- Construction of a simulation model for the hardware under test
- Typically with VHDL: Widespread use, hierarchical description capabilities, structure and behavior description in one place
- VHDL code modification vs. saboteurs insertion
  - Latter mutate existing component descriptions, supported by VHDL tool set
  - Saboteurs can model most faults, including environmental conditions (ESD)
- Benefits
  - Can support all levels (electrical, logical, functional, architectural), not intrusive, broad fault models, without extra hardware, integrated in normal design flow, support for timing-related faults, maximum observability
- Drawbacks
  - Large development effort, long experiments, model accuracy

Emulation-Based Fault Injection

- Cope with time limitations of simulation by synthesizing VHDL descriptions
- Typically, necessary injection modifications are included in the model
- External trigger from connected development machine
- Alternative: Rely on reconfiguration capabilities of the FPGA itself
- Benefits
  - Injection is faster than with simulation, low cost approach
- Drawbacks
  - VHDL must be synthesizable, only good to investigate functional failures, timing failures are not supported, restricted by I/O capabilities of the FPGA

SoftWare-Implemented Fault Injection (SWIFI)

- Use software to emulate hardware faults by representative modifications
- Additional software is expected to be independent from the rest of the system
- Compile-time vs. run-time injection
  - Latter demands trigger: Time-out, exception / trap, code insertion, debug register
- Benefits
  - Can target applications and operating systems, testing the production system, no special-purpose hardware required, no model development needed
- Drawbacks
  - Limitation to assembly instruction level, only locations that are accessible for software, additional code that does not exist with the real fault, limited controllability (e.g. processor pipeline), permanent faults are problematic

Current Research at OSM / HPI

- Software-Implemented Fault Injection at Firmware Level - a new SWIFI approach
  - New fault locations (instruction pointer, trap controller state, hypervisor, ...)
  - Better portability
  - Advanced fault triggers