POLITECNICO MILANO 1863

Dependable Systems

Dependability analysis: fault injection

Luca Cassano
luca.cassano@polimi.it
cassano.faculty.polimi.it/ds.html

TOPIC QUESTIONS

How does the system react to the occurrence of a fault?

How reliable or available is the system?

What are the most critical faults?

Overview

What:

Analysis of how the designed/implemented system behaves when faults occur

Testing technique used to test both hardware and software

Overview

What:

Analysis of how the designed/implemented system behaves when faults occur

Testing technique used to test both hardware and software

How:

Controlled experiments where the system is observed after introducing faults on purpose

Aims

- Evaluate system’s susceptibility to faults
- Study fault/error relationship
- Study the behavior of the system when faults occur
- Study faulty behaviors of the target system w.r.t. connected systems
- Evaluate (qualitative/quantitative) fault coverage
- Evaluate the effectiveness of fault tolerance mechanisms
- Assess the system dependability requirements fulfillment
- Identify dependability bottlenecks
- Support new dependability techniques design

Classification

Hardware vs. Software

- Hardware:
  - System components fail
- Software:
  - System software (application or OS) fails
  - Hardware faults are modeled through software erroneous behavior
  - Hardware faults are simulated

Classification | 2

Simulation vs. emulation

- Simulation:
  - a model of the system is developed and faults are introduced into that model
  - corrupted model is then simulated
- Emulation:
  - system is deployed, and some mechanism is found to cause faults
  - execution is then observed to determine the effects of the fault

⬇ slow

⬆ flexible

⬇ fixed

⬆ accurate

Classification | 2

Simulation vs. emulation

- Simulation:
  - a model of the system is developed and faults are introduced into that model
  - corrupted model is then simulated
- Emulation:
  - system is deployed, and some mechanism is found to cause faults
  - execution is then observed to determine the effects of the fault

🔻 slow

🔺 flexible

🔻 fixed

🔺 accurate

What about the accuracy of simulation-based approaches?

Classification | 3

Invasive vs. Non-invasive

- Invasive:
  - injection mechanisms are not transparent
  - footprint of the testing mechanism in the behavior of the system
- Non-invasive:
  - injection mechanisms mask their presence so as to have no effect on the system other than the faults they inject

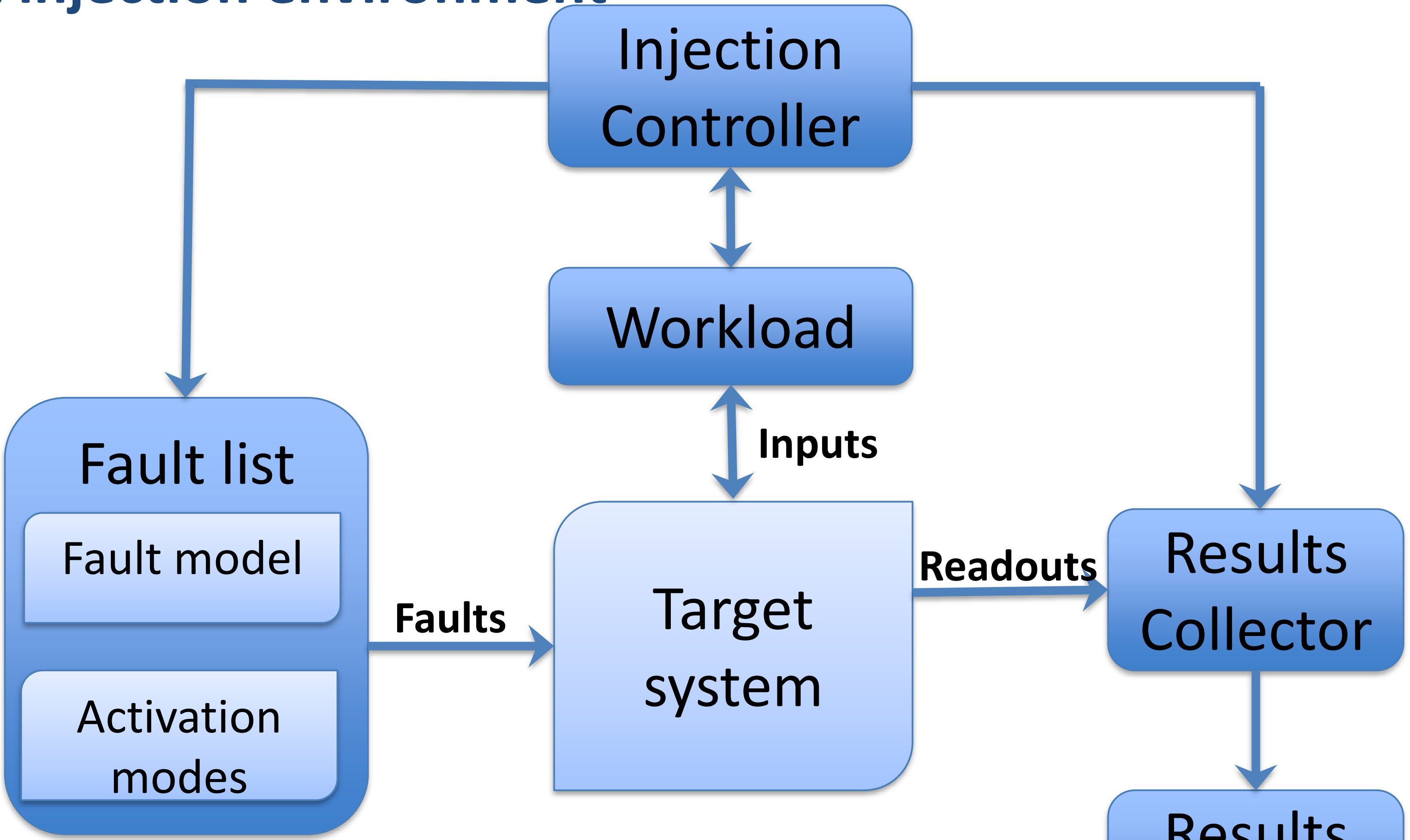
⬇ modified system

⬆ fast

⬇ hard to design

⬆ no impact

Fault injection environment



Measures:
-Fault coverage
-Fault latency

Fault model

Fault models highly depend on the target system (or its model)

Injection controllability impacts on the capability to inject faults

Adoption of behavioral models for the target system requires the definition of functional fault models

Workload, activations and simulation duration

A workload has to be defined to opportunely stimulate the system

Simulation ends according to

- A specified duration
- The occurrence of a specified event

The fault is defined in terms of

- Fault model
- Injection time
- Injection point
- Activation mode

Results collector

Readouts depend on the capability to monitor the target system

- Monitoring of the outputs
- Adoption of probes for reading internal registers

Collection can be

- Continuous
- Final results
- Triggered by specific events

Results analysis

Experimental results are compared against a golden model of the system execution

Several measures can be computed

- Fault coverage
- Fault latency
- Fault susceptibility
- Architecture vulnerability factor

Several analysis can be performed

- Fault/error relationship
- Fault propagation

Result analysis | 2

Experiment classification highly depends on the properties the designer aims at testing

- Silent/detected/failure
- No-effect/critical/not critical
- No-effect/safe/dangerous (EN/IEC61508)

Fault injection campaign

A fault injection campaign is a set of experiments each one described in terms of

- Workload to be applied
- Fault to be injected (fault model, injection condition)
- Readouts to be collected
- Readouts analysis strategy

Fault injection campaign

A fault injection campaign is a set of experiments each one described in terms of

- Workload to be applied
- Fault to be injected (fault model, injection condition)
- Readouts to be collected
- Readouts analysis strategy

Why do we need a campaign?

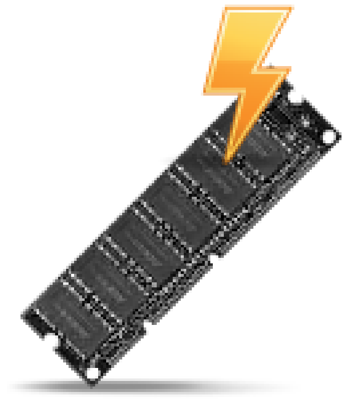
Fault injection campaign

FI strategy

1. The system under test is designed, modeled or implemented
2. A set of faults is selected
3. A workload is determined to stimulate the device
4. Experiment is performed until the fault injection instant
5. Experiment is suspended
5. A fault is injected as “effect” in the implementation
6. Experiment is resumed
7. Outputs is observed for a **certain** period
8. If there is another experiment, reset the system and return to step 4

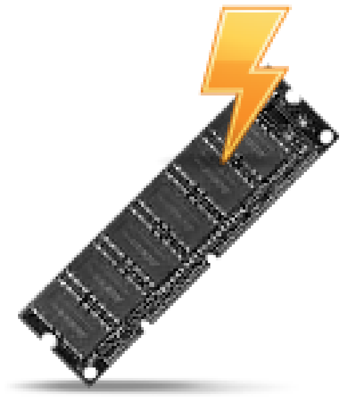
Main approaches

Hardware implemented
fault injection

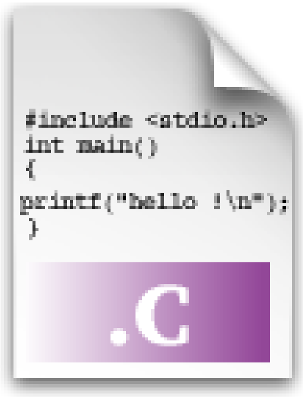


Main approaches

Hardware implemented fault injection



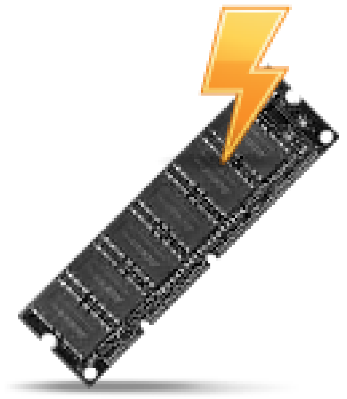
Software-based emulation



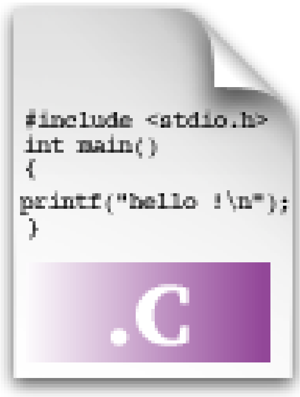
```
#include <stdio.h>
int main()
{
printf("hello !\n");
}
```

Main approaches

Hardware implemented fault injection



Software-based emulation



```
#include <stdio.h>
int main()
{
printf("hello !\n");
}
```

Software-based simulation

Hardware fault injection HWFI

Faults are injected in the circuit after fabrication or in a “close-to-final” prototype of the system under analysis

Additional hardware facilities are used for causing the faults into the system

Hardware fault injection HWFI | 2

Fault model:

- Stuck-at, bridge, … depending on the approach and on the target device (discussed later)

Activation

- Fault injection is triggered at a random time instant

Readouts

- Only board/device outputs can be acquired

Measures

- Statistics on the final results (fault coverage, …)

Hardware fault injection HWFI | 2

Fault model:

- Stuck-at, bridge, … depending on the approach and on the target device (discussed later)

Activation

- Fault injection is triggered at a random time instant

Readouts

- Only board/device outputs can be acquired

Measures

- Statistics on the final results (fault coverage, …)

Can you analyse the effect of aging/ wearout effects?

Hardware fault injection | 3

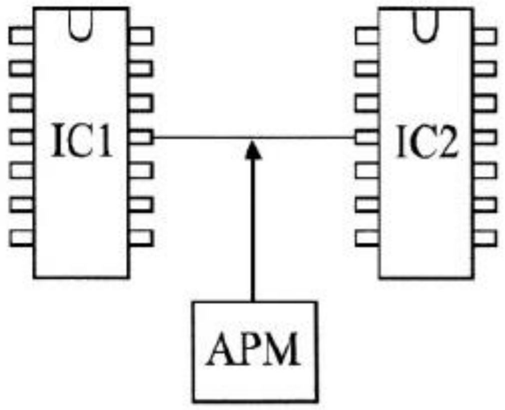
There are two different types of HW fault injection:

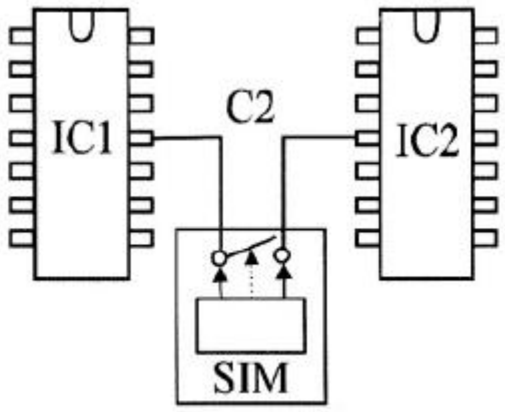
- With contact
- Without contact

Hardware fault injection with contact

With contact (Pin-level FI)

- Active Probes: adds current via the probes attached to the pins
- Socket insertion: socket between the target hardware and its circuit board





Hardware fault injection with contact | 2

Fault models:

- With active probes: stuck-at
- With socket insertion: stuck-at, open line, bridge

Faults can be injected only chip I/O

Particular cases:

- Supply voltage variation
  - Processor’s misinterpret or to skip of an instruction
- Clock variation
  - Erroneous data read/write or instruction miss

Hardware fault injection without contact

Types of approaches:

- Heavy-ion radiation
- White light
- Electromagnetic fields
- Lasers
- X-ray

Fault model

- Transient faults (SEU, SET)

Hardware fault injection without contact | 2

The controllability highly depends on the type of approach

- E.g.: white lights irradiate the whole chip while lasers are able to affect a specific position

Some techniques requires the chip to be depackaged

Most of the techniques accelerate the aging of the device or cause permanent failures

Hardware fault injection

- ⇧ Real hardware faults are injected
- ⇧ Global system validation
- ⇧ Very fast and not intrusive

- ⇩ Risk of damaging the circuit
- ⇩ Limited portability and observability
- ⇩ Limited controllability of injected faults and injection points
- ⇩ High monetary cost

Software-Implemented Fault Injection SWFI

Faults are injected in the software (applications and OS) executed on the system's hardware platform

Fault injection is performed according to the programmer's modeling view of the system

Software-Implemented Fault Injection | 2

Injection at design time
- by manipulating source/assembly code

Injection at run-time
- Time-out and Exception/Trap
  - Specific events trigger the execution of the fault injector
- Code insertion
  - Fault injector injects additional instructions
- The processor debugging unit can be used as fault injector
- System-calls can be intercepted to activate the fault injector

Software-Implemented Fault Injection | 3

Fault locations:

- Data/instruction memory of applications and OS
- Processor registers

Activation of fault injection is

- Time-triggered
  - when a time-out expires
- Event-triggered
  - when a specific instruction/data memory address is accessed

Software-Implemented Fault Injection | 4

Fault model is usually the transient fault

High level fault models can be adopted

- mis-timings
- missing messages/replays
- corrupted memory or registers
- System-call corruption
- Kernel functionalities corruption
- almost any other state the hardware provides access to

Software-Implemented Fault Injection | 5

Readouts

- The application final results are usually collected
- The debugging unit can be used for collecting execution traces
- System-calls and kernel events can also be monitored

Measures

- Application failure modes are usually computed
- Trace analysis can be performed with limited capabilities

Software-Implemented Fault Injection | 6

- ⬆ Can target application and/or OS
- ⬆ No hardware facilities required
- ⬆ Close control of software for validation
- ⬆ Ease to adapt/modify

- ⬇ Limited injection points and fault models
- ⬇ Poor time resolution
- ⬇ Highly intrusive
- ⬇ May introduce timing variations

Hybrid SW-HW implemented FI

SWFI and HWFI can be combined for achieving an advanced fault injection environment

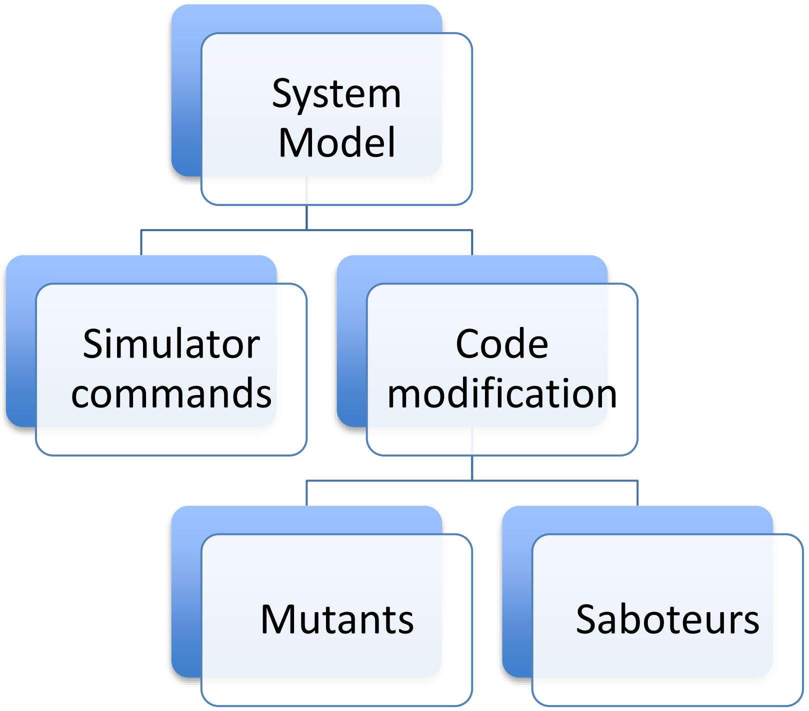
- HWFI is used to inject at pin-level
- SWFI is used to inject into the processor

Simulation-based FI

Injection of logical faults into system model

Fault injectors are usually based on system model simulators (e.g.: modelsim)

Different approaches are available:



Simulation-based FI | 2

The fault model depends on the model of the target system

e.g.:

- In a netlist model it is possible to inject stuck-at
- In a behavioral description of a system, faults are described as misbehaviors of some functionality

Simulation-based FI | 3

Simulator commands

- Exploitation of the simulation environment for the selected HDL modeling language to corrupt the value of signals, memory elements, etc
- No code modification
- Automation by scripting

Simulation-based FI | 4

Mutants

- Use of component descriptions that replace correct ones, when the fault is activated
  - E.g.: modification of an operator, i.e. + becomes –
- It models the effects of the fault, or just a different behavior w.r.t. the expected one
  (depends on abstraction level)

Simulation-based FI | 5

Mutants can be defined by modifying HDL syntactical units in behavioral description

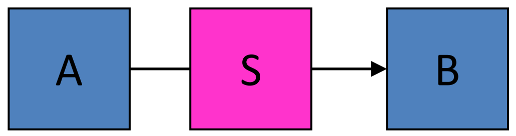
e.g.:

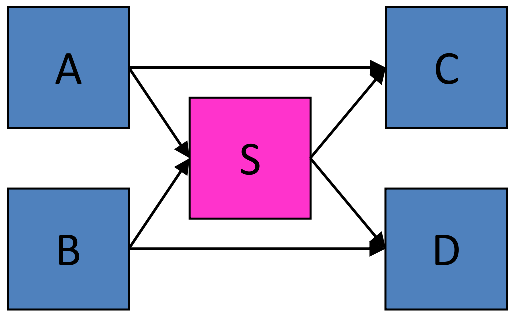
- stuck-then: replace condition with true
- Stuck-else : replace condition with false
- Assignment control: corrupt assignment
- Dead process: eliminate process sensitivity list

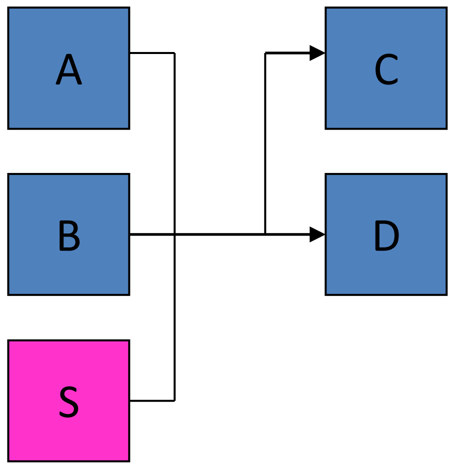
Simulation-based FI | 6

Saboteurs

- Introduces additional elements on the signal paths: when activated, the saboteur alters the value of the signal
- Easy to add, but limited injection points







Simulation-based FI | 8

The high controllability and observability offer large capabilities in fault activation and readouts collection

- Activation of fault injection can be triggered by any events
- It is possible to collect the system status for the overall experiment duration

Simulation-based FI | 9

According to the collected readouts it is possible to define a large set of measures

- Outputs classification according to failure modes
- Analysis of the fault/error relationship
- Error propagation

Experiments can be early interrupted

- e.g.: when the fault effects disappear

Simulation-based FI | 10

- ⬆ Several abstraction levels available
  - Architectural, Functional, Logical, Electrical & Mixed
- ⬆ Applicable to early design stages
- ⬆ High controllability/observability
- ⬆ Low cost automation

- ⬇ Large development effort for the models
- ⬇ Complex mapping between simulated faults and real ones
- ⬇ Time consuming
  - Accuracy/experiment length trade-off

Emulation-based Hardware fault injection

Field Programmable Gate Array (FPGA) platforms are used to implement a prototype of the system

Two different injection approaches are used:

- Instrumentation
- Run-time reconfiguration

Emulation-based FI | 2

- ⬆ Faster than simulation-based FI
- ⬆ More accurate than simulation-based FI

- ⬇ Development of an HDL model of the system
- ⬇ Cost of the FPGA for large designs

Acknowledgements

R. Slater, “Dependable Embedded Systems,”

E. Martins, “Software-based Fault Injection”

L. Frigerio, “Fault injection: techniques and tools”

A. Benso, P. Prinetto, “Fault Injection Techniques and Tools”

TOPIC QUESTIONS

How does the system react to the occurrence of a fault?

How reliable or available is the system?

What are the most critical faults?

TOPICS

Perform fault injection to see how the system evolves when a fault occurs

Simulate, emulate or really “break” it