UNIVERSITÀ DI PISA

Simulated fault injections of Soft Errors in the Configuration Memory of SRAM-Based FPGAs using Mobius

Cinzia Bernardeschi

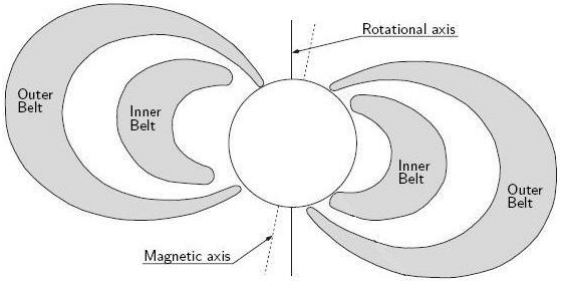
Soft errors

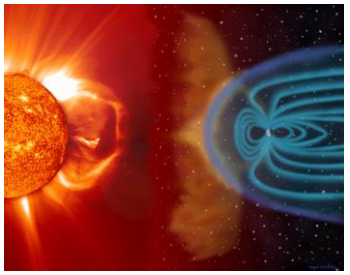
- Soft errors are caused by high-energy particles that change the stored charge in memory cells of electronic devices
- In particular, a bit flip, or Single Event Upset (SEU), in a memory cell may bring the system to an erroneous state until the cell is rewritten.
- SEUs, therefore, are usually transient as they are likely to be overwritten at the next clock cycle, but

they may have permanent, or at least long term, effects in configurable devices such as SRAM-based FPGAs, since the functions and the interconnections of these devices are controlled by a configuration memory

Radiation in a nutshell

Sources:
- Van Allen belts
- Solar flares





Duration:
Long-term effects: Total Ionizing Dose (TID)
Short-term effects: Single Event Effects (SEEs)

Single Event Effects classification:
- Permanent (Hard Errors)
  - Single Event Burnouts (SEBs)
  - Single Event Gate Ruptures (SEGRs)
  - …
- Transient (Soft Errors)
  - Single Event Upsets (SEUs)
  - Single Event Transients (SETs)

FIT = Failure in time rate (in 10^9 hours)

~100 FIT/MByte at ground level
~1 SEU/year per device in space

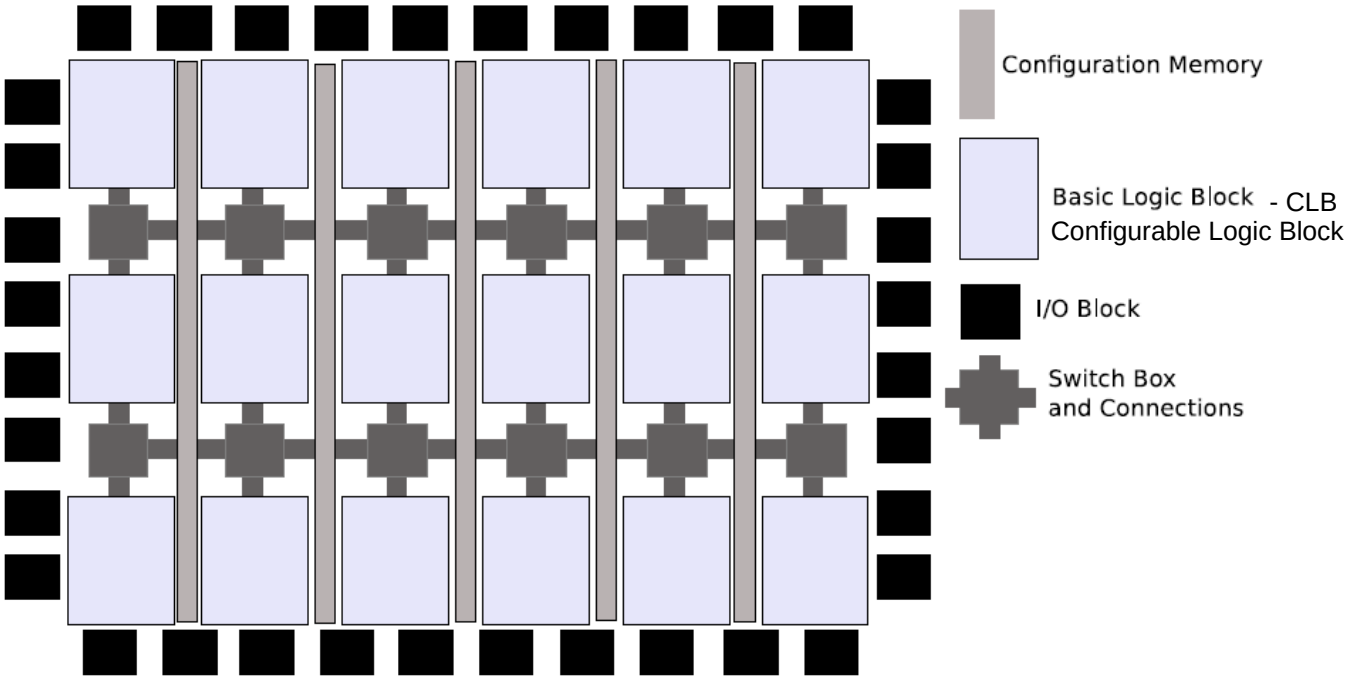
SRAM-FPGA

An FPGA is an array of programmable logic blocks, interconnected through a programmable routing architecture and communicating with the output through programmable I/O pads

Conductive tracks (or wires) are grouped into channels running in two orthogonal directions and forming a grid whose meshes enclose the logic blocks.

The areas where vertical and horizontal channels cross are called switch boxes and contain the programmable circuitry that connects tracks of one channel to tracks of the same or the other channel.

Overall structure of an FPGA architecture

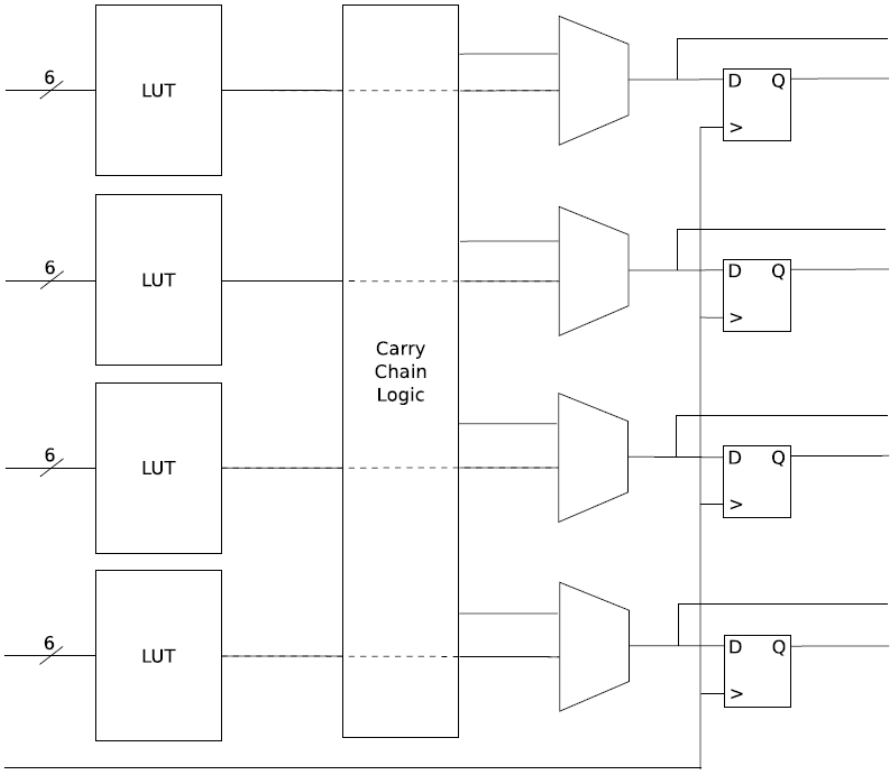


- Regular layout of the logic blocks, the routing architecture, and the configuration memory.
- The configuration memory bits control and program both logic and routing resources of the FPGA

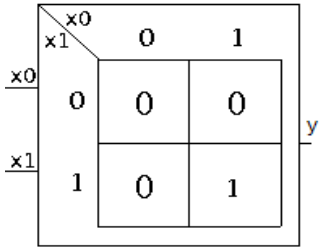
Basic logic block structure

Configurable Logic Blocks - CLB:

- n m- input LUTs implementing the required functions
- a carry chain logic that can be used to implement arithmetic functions,
- a number of multiplexers that allow selecting among the outputs of the LUTs and those of the carry chain logic
- and a number of flip-flops



LUT = Look-Up Table



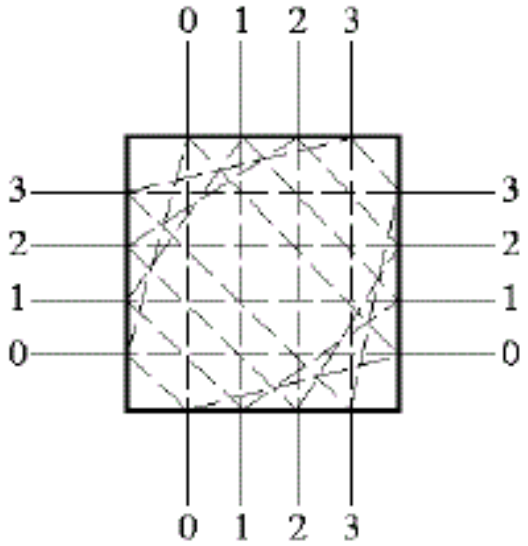
2-input LUT

Switch box structure

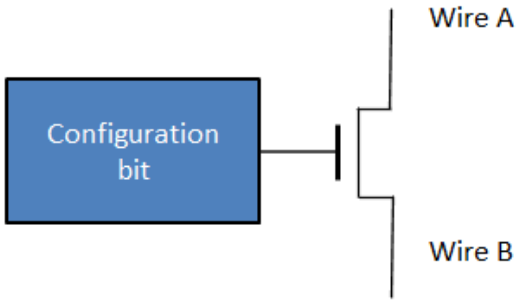
Switch matrix, by which it is possible to drive signals inside an FPGA.

Routing is obtained by means of Programmable Interconnection Points (PIPs), which are CMOS transistors that can be activated and deactivated in order to get a custom path within a switch matrix.

Switch matrices are linked together by fixed wires.



Programmable Interconnection Point (PIP)

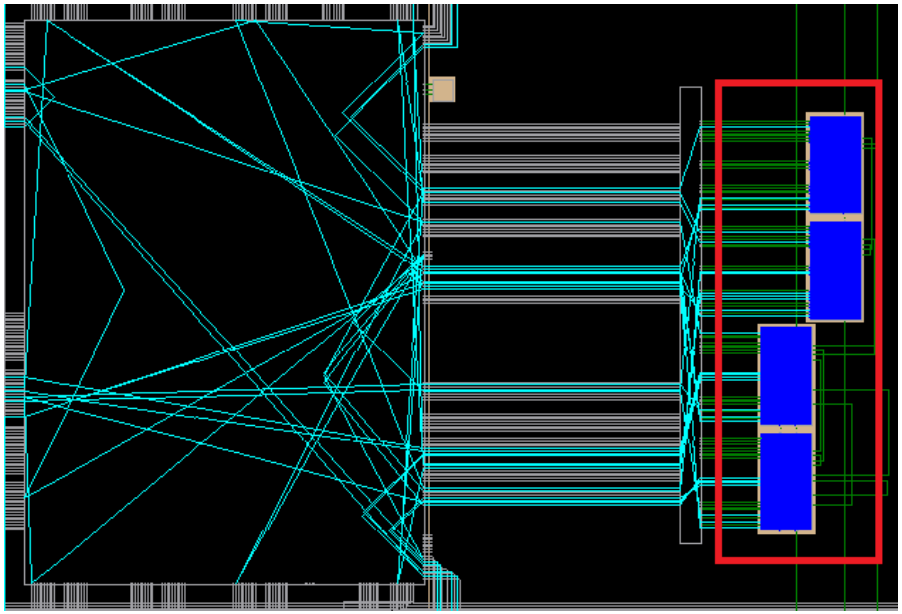


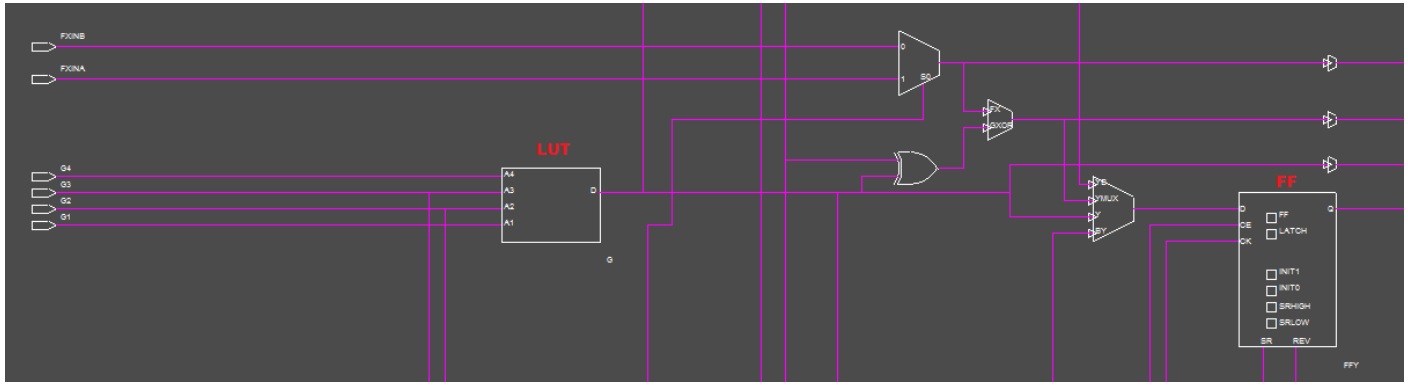
Regarding an FPGA architecture the programmable routing infrastructure consists in a set of wire segments, which can be interconnected by means of programmable elements: PIPs.

A PIP is a switched element, whose state is determined by the value contained in a configuration cell.

Xilinx SRAM-FPGA

The CLB structure depends on vendor and on FPGA families. Example of Xilinx Virtex-4 CLB: consists of 4 slices





A slice is split in part F and G. Each part consists, from a simplified viewpoint, in a 4-input Look Up Table(LUT) and a Flip-Flop(FF)

SRAM-FPGA programming

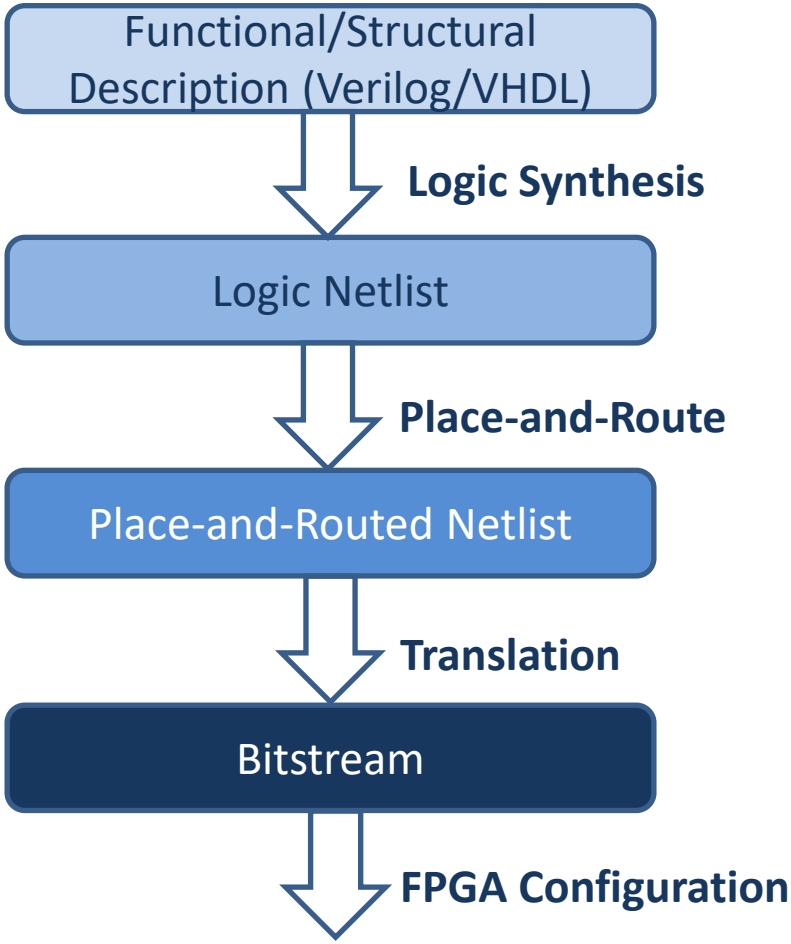
1. Functional/Structural Description
FPGAs are programmable devices, for this reason their behaviour has to be defined by the end user, which can do that by means of hardware description language. The most used HDLs are Verilog and VHDL.

2. Logic Netlist
using a design automation tool, which are typically supplied free of charge by vendors, a technology-mapped netlist is generated. A netlist is a textual description of a circuit diagram, which provides a map of how its elements are interconnected.

3. Place-and-Routed Netlist Then a process, called Place-and-Route, can be performed in order to adapt a netlist to an actual FPGA architecture. On the result of the previous process, several verification methodologies, such as timing analysis and simulation, can be performed.

SRAM-FPGA programming

Standard FPGA design process



SRAM-FPGA store configuration data in SRAM memory

SRAM-FPGA programming

4. Bitstream

once that the result is verified a binary file, called bitstream, is generated and loaded into the device in order to configure or reconfigure it.

- FPGAs feature a high flexibility due to the possibility to being reprogrammed via software

- Partial Reconguration(PR) permits to reconfigure a specified part of the device, without affecting parts of the device.

- The bitstream determines the functions of the programmable logic blocks, the internal connections among logic blocks and the external connections among logic blocks and I/O pads.

- SEUs have a long-term effect, as they are likely to be overwritten by a reconfiguration.

Failure probability analysis

widely used approach to evaluate the propagation of faults in digital devices

Fault injection techniques can be divided into:

- prototype based
  accelerated radiation ground testing aims at emulating the effects of SEUs by exposing a prototype of the FPGA-based system to a flux of radiations, originated by either a radioactive source or a particle accelerator

- simulation based
  fault simulation can be applied soon during the design of the system and at different levels of abstraction

Note: *while radiation testing and fault injection allow the analysis of the system's behavior only by observing its output, fault simulation makes it possible to study fault activation and fault propagation, giving designers a deeper insight into the circuit and the causes of system failures.*

Simulation-based fault injection

- Simulation-based fault injection technique for failure probability and fault observability assessment of SRAM-FPGA systems based on a model of FPGA netlists realised with the Stochastic Activity Networks formalism.

- Faults can be injected into the model either stochastically or exhaustively one at a time.

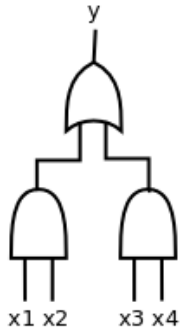
- Fault propagation is traced to the output pins, using a **four-valued logic** that enables faulty signals to be tagged and recognized.

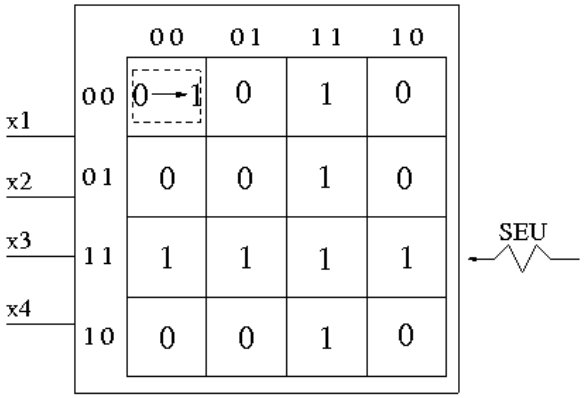
- A golden copy is not needed

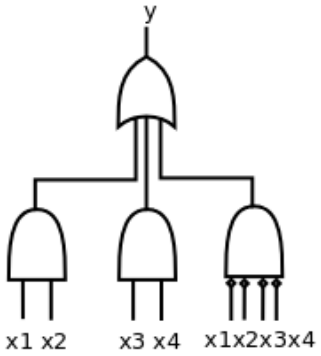
Effects of SEUs in the logic components of an FPGA

Logic Faults

Bit flip of a SRAM cell







Before SEU

After SEU

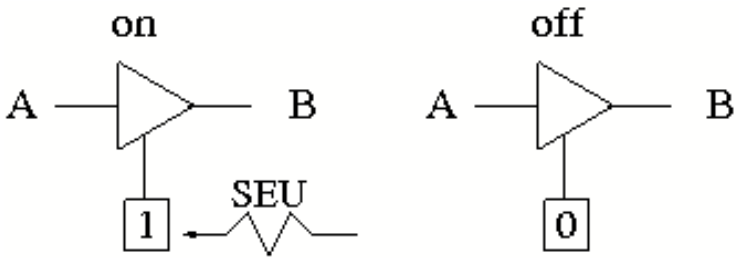
A LUT affected by an SEU in its configuration memory will produce an incorrect value only when the pattern of its input values is the one associated with the faulty configuration bit, while for every other input pattern the faulty LUT will behave correctly

-> different from the stuck-at fault model

Effects of SEUs in the logic components of an FPGA

Logic Faults

Bit flip in an I/O buffer



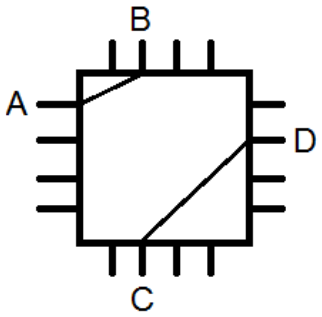
Undesired connection disconnection between two wires

The 1 changes to 0 bit flip causes a disconnection between point A and B.

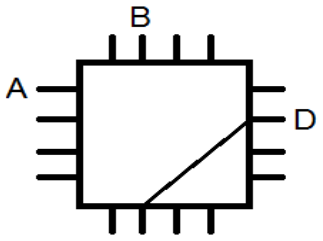
Effects of SEUs in the logic components of an FPGA

Routing Faults

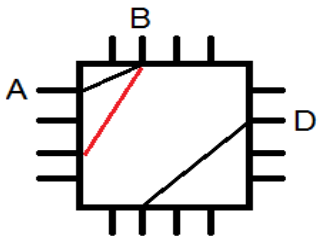
Before SEU



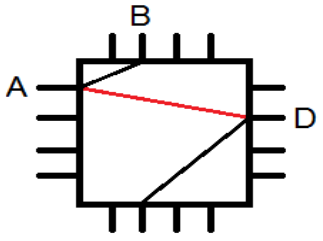
After SEU



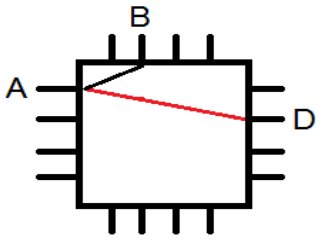
Open Fault



Antenna Fault



Conflict Fault



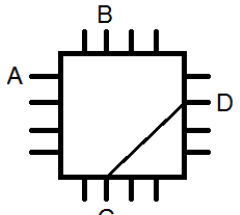
Bridge Fault

M. Violante *et al.* “Reconfigurable Field Programmable Gate Arrays for Mission-Critical Applications”, Springer Science & Business Media, 2011.

Effects of SEUs in the logic components of an FPGA

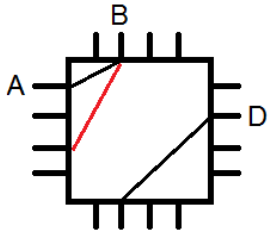
Depending on the position and the electrical properties of the affected PIPs, an SEU in the routing structure can cause the following topological modifications.

• **Open.** The PIP corresponding to a net is not programmed anymore.



Open Fault

• **Input antenna.** A new PIP is added between an unused input node and a used output node. The new PIP can influence the behavior of the output node.

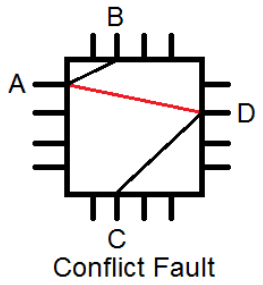


Antenna Fault

• **Output antenna.** A new PIP is added between a used input node and an unused output node. The new PIP does not influence the behavior of the implemented circuit.

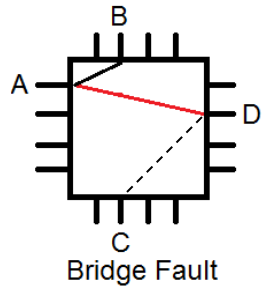
Effects of SEUs in the logic components of an FPGA

• **Conflict.** A new PIP is added between an input node and an already used output node.



Conflict Fault

• **Bridge.** A PIP is disabled (---) while a new PIP is instantiated between a used input node and the output nodes used by the previous interconnection segment.



Bridge Fault

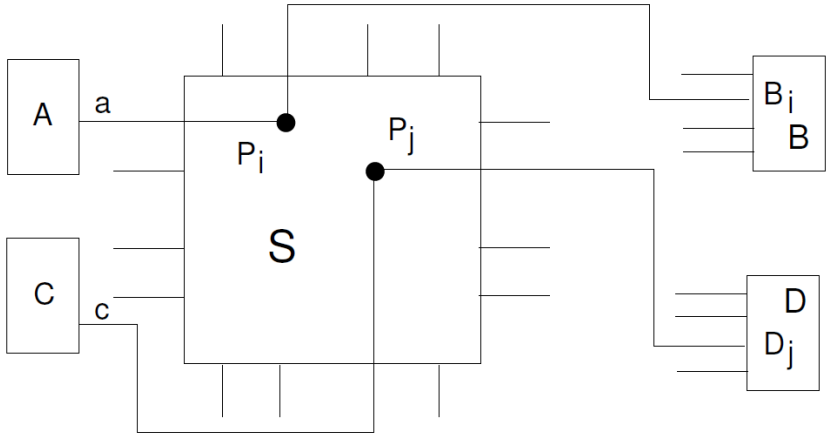
By comparing the results of electrical fault injection with behavioral simulations, it has been possible to identify the logical effects of the topological modifications induced by SEUs into the routing resources, thus defining an equivalent logical model associated with such SEUs.

Logic effects of routing faults (E2STAR)

Logic effects of a SEU in the configuration bit controlling the PIPs

S: a switch box

B and D: two components directly connected to S, Bi and Dj are the input pins of B and D connected to S through the PIPs Pi and Pj , respectively.



Possible logical effects of an SEU in the configuration bit controlling the two PIPs

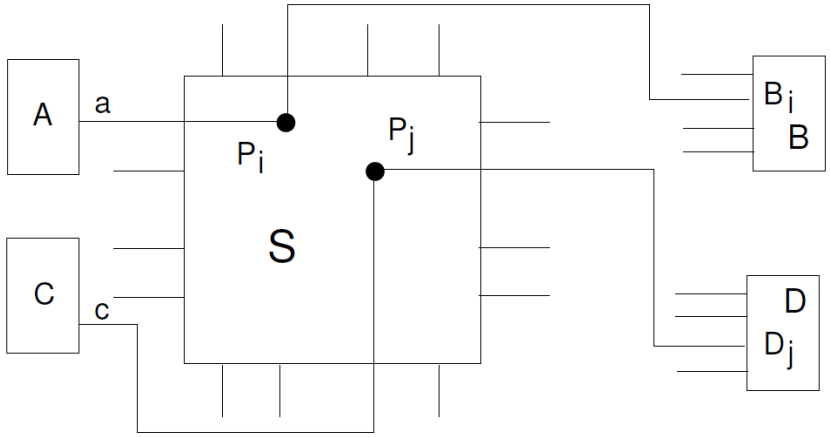
Stuck-at 0 on Bi: The logic signal on Bi is set at zero.

Stuck-at 1 on Bi: The logic signal on Bi is set at one.

Logical bridge between Bi and Dj : The logic values on Bi and Dj are exchanged; or one between Bi and Bj assumes the value of the other.

Logic effects of routing faults (E2STAR)

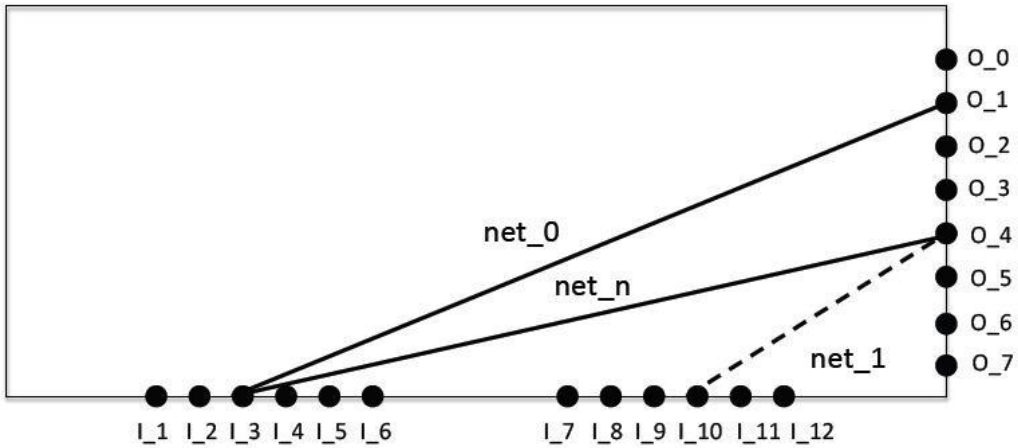
Logic effects of a SEU in the configuration bit controlling the PIPs



Wired-AND (Wired-OR) between Bi and Dj : The logic signals on Bi and on Dj are set to the value a AND (OR) c.

Wired-MIX between Bi and Dj : Bi is set to 1 and Dj is set to 0 if (a != c), while Bi and Dj are left unaltered otherwise.

Bridge fault



the interconnection segment corresponding to net 1 is disabled while a new segment corresponding to the interconnection net n is instantiated between a used input node and the output nodes of the previously used net 1

Propagation of faults: four-valued logic

Four-valued logic is used to propagate faults through the circuit

$D = \{0c, 0f, 1c, 1f\}$
represents correct and faulty Boolean values, namely zero correct ($0c$), one correct ($1c$), zero fault ($0f$), and one faulty ($1f$)

Tracking function:
-models the propagation of values through the netlist
- returns the value that is compute by correct and faulty components (for each input configuration)

Each component of the netlist implements a Boolean function

$$f : B^n \rightarrow B.$$

For such function, its tracking function f* extends f to the four-valued domain D

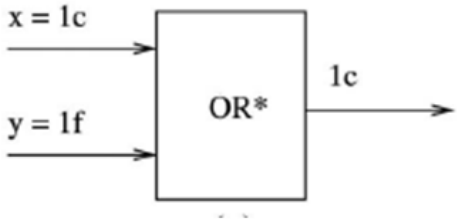
$$f^* : D^n \rightarrow D$$

Tracking function of a component

Given n-tuple of inputs (d1, · · · , dn) in D^n, the tracking function f* of a component applies the function f to the actual inputs and to the input that would have been applied in absence of faults.

Then f* compares the two results and returns the result on the actual input, annotated with c (correct) if the two results are identical, or f (faulty) if the two results differ

CASE 1: CORRECT COMPONENT



Example: LUT-OR* input: 1c, 1f

OR(1,1)=1 actual input
OR(1,0)=1 input in absence of faults → OR*(1c, 1f) = 1c

Fault not propagated

OR gate
input: 1c, - the result is correct, no matter what the other input is

Tracking function of a component

CASE 2: FAULTY COMPONENT

For each possible fault i of the component, let f(i) be the faulty function

$f(i) : B^n \rightarrow B$
describes the behavior of the component in presence of the fault i.

The tracking function

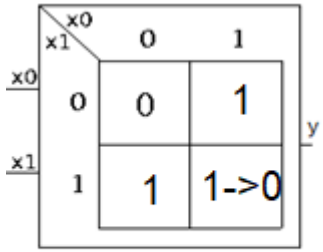
$f^*(i) : D^n \rightarrow D$

of a faulty function f(i) compares the output of the faulty component with the actual inputs to the output of the correct component with correct inputs.

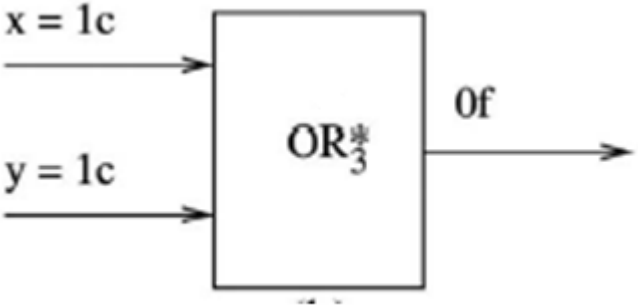
If the two are equal, the result is taken as correct, otherwise the result on the actual input is taken as faulty.

Tracking function of a component

Example: failing 2-input LUT on input configuration (11)



assume OR_3* represents a LUT implementing an OR, which is faulty for input configuration x = 1, y = 1.

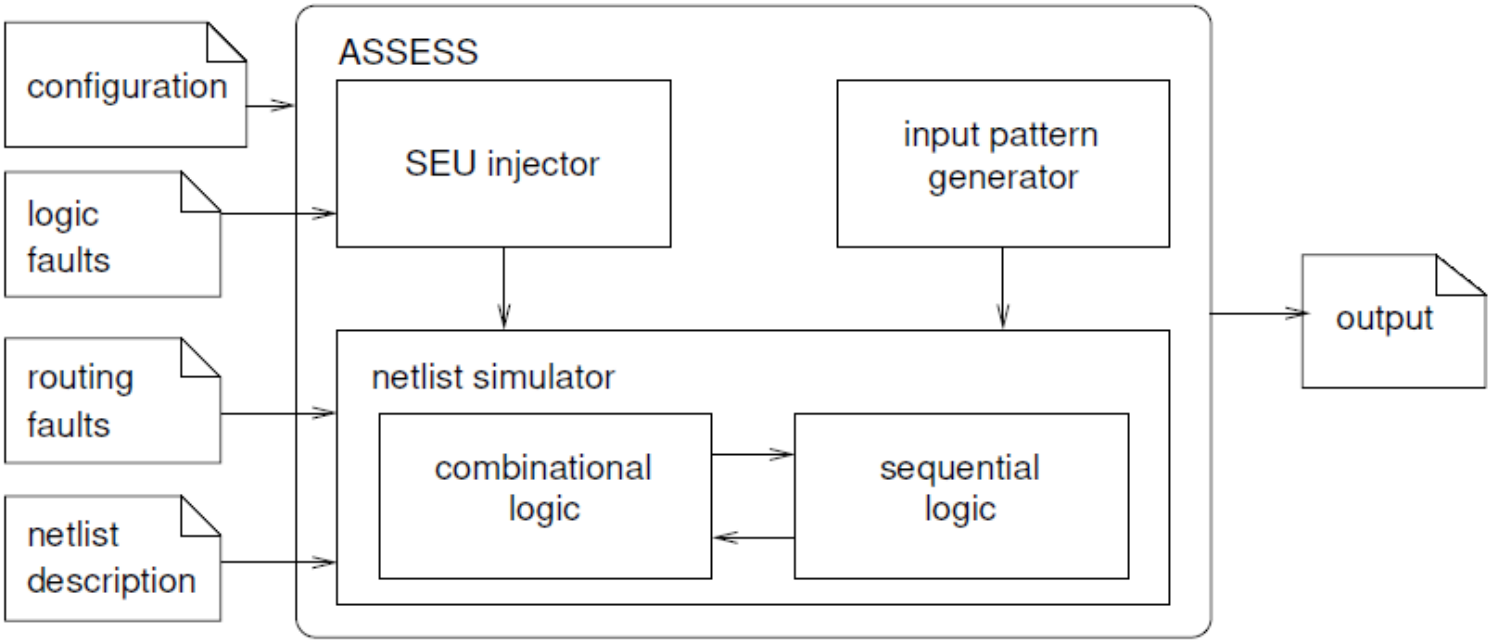


OR(1,1) != OR_3 (1,1) ⇒ therefore OR_3*(1c, 1c) = 0f

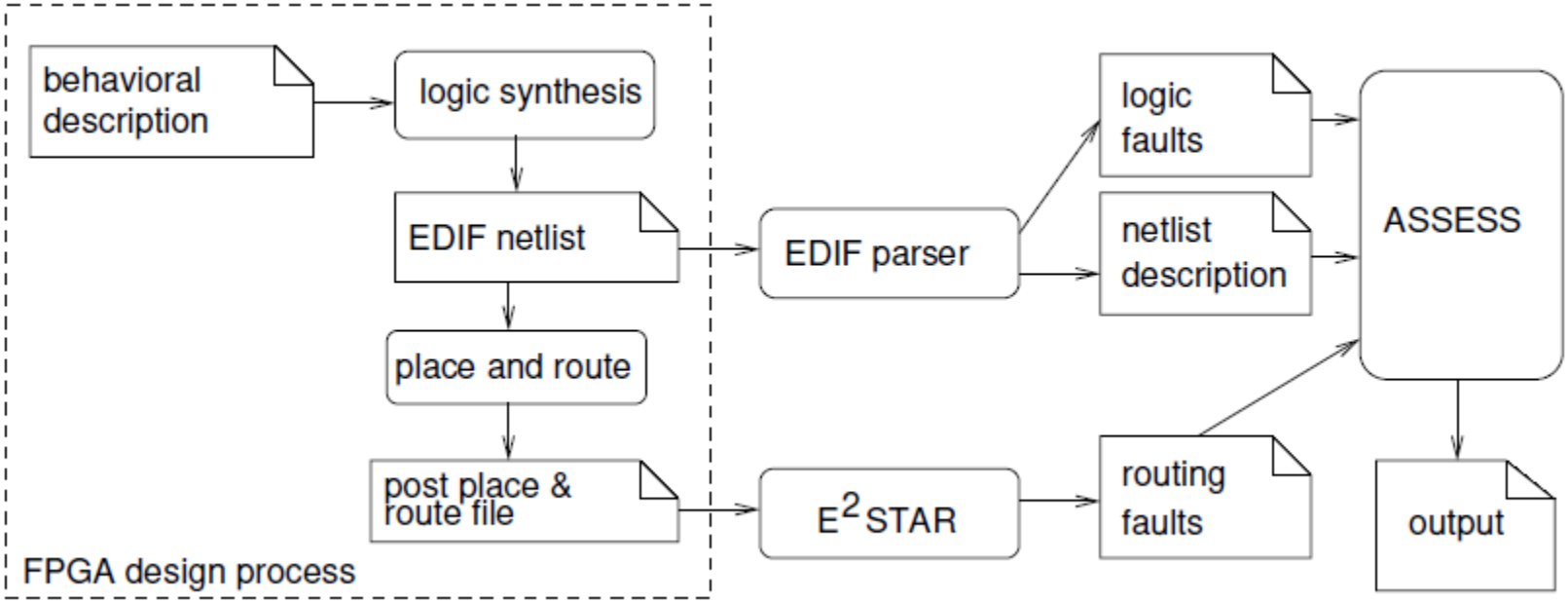
Propagation of the fault

The simulator

- netlist-level simulator (before the place & route)
- logic fault list: SEUs in the configuration memory of logic resources
- routing fault list: logical effetcs of SEUs in the routing (E2STAR tool)
- generate and collect test patterns for detecting SEUs
- analyse the error-failure relation.



Flow Diagram of the Simulation Environment



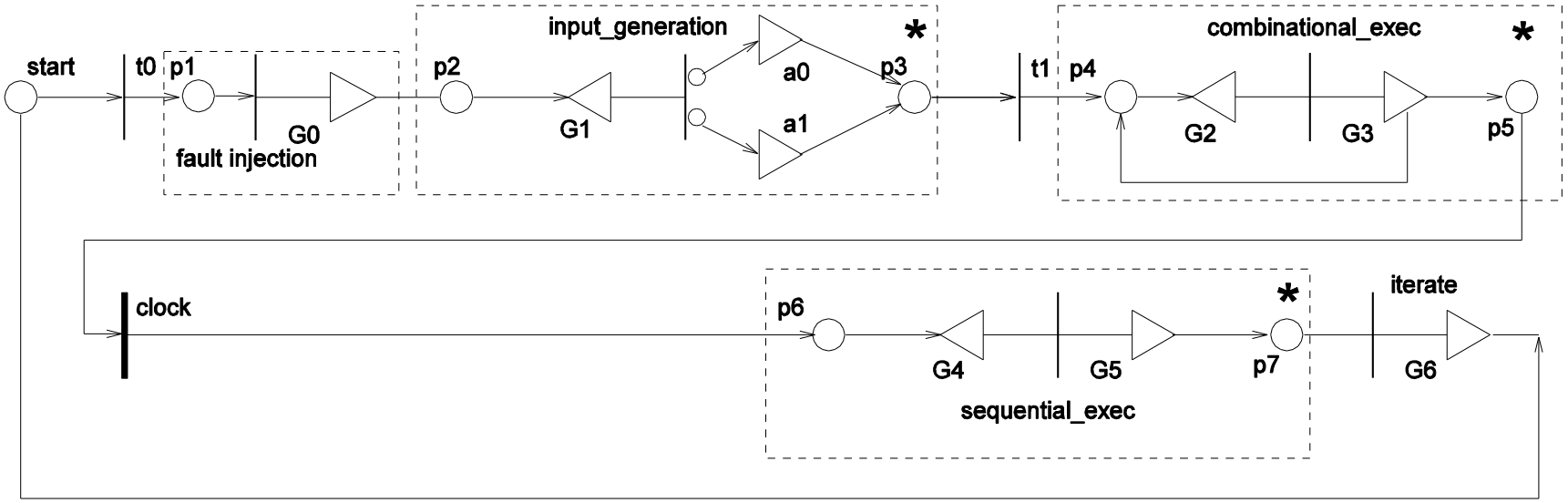
Flow Diagram of the Simulation Environment

- The parser also produces the list of SEUs occurring in configuration bits associated with the logic resources and of their effects, according to the fault model
- E2STAR produces a file containing the list of the SEUs in the configuration bits associated with the routing elements of the FPGA that can alter the structure of the implemented system. For each SEU, the file contains the list of its effects on the logic components
- Simulation can be run before Place-and-Route
- The Input Pattern Generator provides the signals at the input pins at each clock cycle. Signal values may be taken from externally generated test patterns or chosen stochastically according to user-provided signal probabilities.
- By configuring dedicated simulation parameters, ASSESS can generate detailed reports on input data and results. In particular, a detailed SEU report shows, for each injected SEU, whether the SEU has been propagated to at least one output pin.

Stochastic Activity Networks as formal models

- Event driven simulation
- A simulation run is the application of a test pattern, i.e., a sequence of test vectors, to the modeled circuit.
- Performance and dependability analysis
  - analyze the sensitivity of the system to each fault, and to have a detailed and complete picture of the activability, propagability and criticality of each fault.

A first model with Stochastic Activity Networks



Input generator: signal probabilities

Fault injector gate G0: injects a fault with a given probability, choosing randomly the fault location (functions are written in C++)

PARSER: from the EDIF file of the netlist to an internal format

The overall behaviour

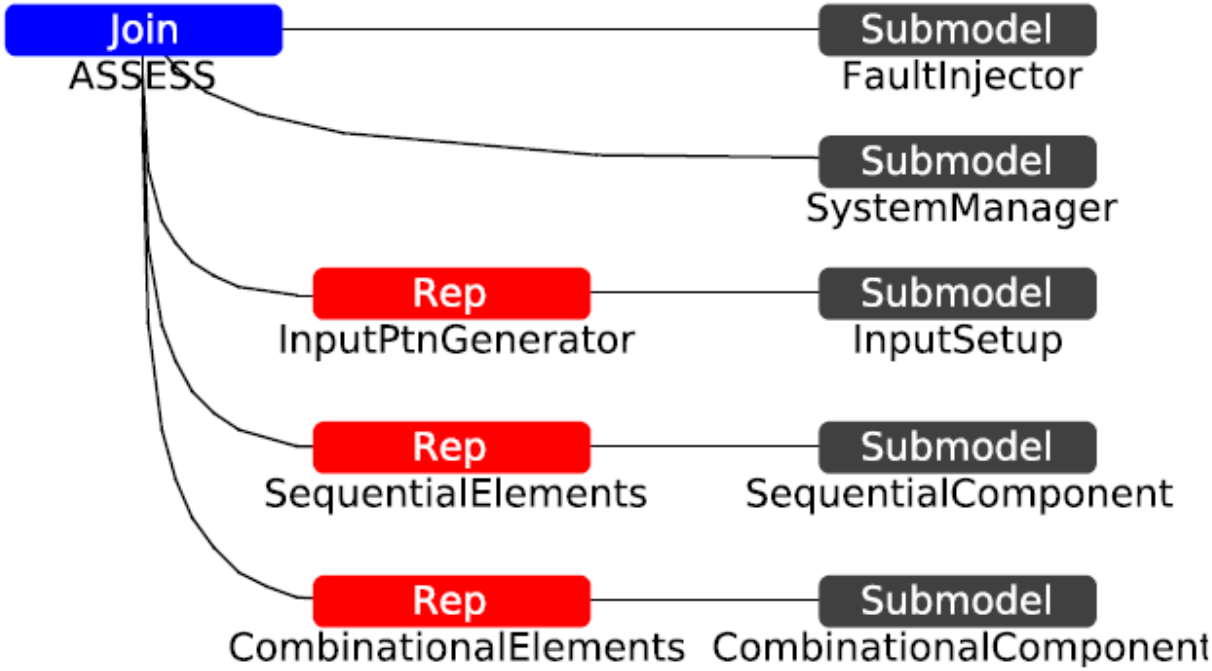
```
call the EDIF parser;
load the netlist description;
load the list of SEUs in logic components;
if routing faults simulation required then
        call E2STAR;
          load the list of SEUs in routing components;
end if
load the configuration;
generate/load test patterns;

if fault-free simulation then
        for each clock cycle n do
            simulate;
          end for
else
```

The overall behaviour

```
if stochastic fault injection mode then
        for each clock cycle n do
                inject an SEU with the configured SEU probability;
                simulate;
                evaluate reward functions;
        end for
else if deterministic fault injection mode then
        for each SEU do
                inject the SEU;
                for each clock cycle do
                        simulate;
                        evaluate reward functions;
                end for
        clear fault;
        end for
end if
collect and output results
```

Stochastic Activity Networks as formal models



SAN submodules interact through shared places. The main shared places are Input Lines, Output Lines, and Internal Lines, holding the values of the input, output, and internal signals, respectively.

Stochastic Activity Networks as formal models

The internal structure of the tool rests on three levels of abstraction:

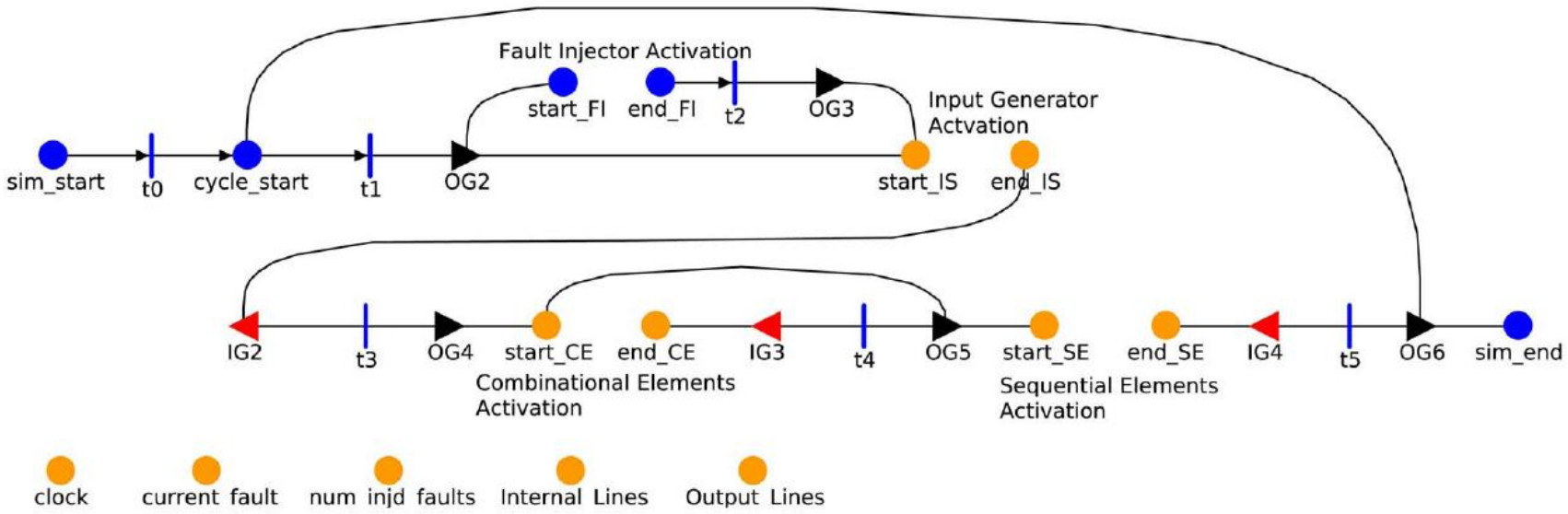
1) A generic SAN model of digital circuit behavior, consisting of a set of interacting

SANs;

2) a set of procedures, attached to the SAN model, that specialize its behavior for the typical logical blocks of FPGA netlists, stored in a library accessed by the Mobius environment;

3) the definitions of the actual systems to be simulated, which comprise the reward model(s), and descriptions of the specific circuit.

The System Manager module

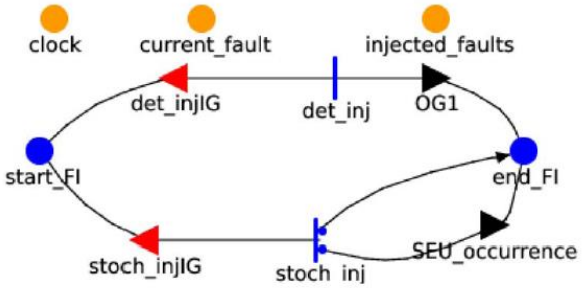


Generic SAN model of a sequential circuit
Set of functions, attached to SAN models, that describe the behaviour of logic blocks of FPGA netlists, stored in a library
Input Generator: signal probabilities
Fault Injector : injects faults

(functions are written in C++)

Stochastic Activity Networks as formal models

Fault Injector SAN model



Three modes of operation are possible:

1) fault-free simulation
2) deterministic fault injection
   all faults are injected, each one at the beginning of a simulation run
3) stochastic fault injection
   the user sets the maximum number of faults to be injected;
   at each clock cycle, a fault is injected with a specified probability q (SEU probability), and then faults are chosen at random

Reward function *failures()*:
increments a performance variable that counts the number of failures detected at the primary outputs and marks as critical each SEU causing such failure

Confidence level 98% and confidence interval 0.1 means that the simulator will execute a number of runs necessary to guarantee the confidence level specified by the user

Measuraments

- The *failure probability* of a circuit is the fraction of simulation runs in which a faulty output is produced, assuming known fault probabilities.

  - random input vectors are generated according to specified signal probabilities of the input, faults are injected according to specified fault probabilities, and at each clock a reward function returns 1 if at least one output value is in the set {0f , 1f }.

- The *fault observability* analysis consists in studying the propagation of faults to the output of the system, and can be evaluated as the fraction of system failures observed in a series of simulation runs over the total number of injectable faults in the configuration bits.

  In each simulation run a test pattern is stochastically generated, and at the end of the simulation the list of applied test patterns and associated discovered faults is available. From this list a set of test patterns able to detect all the observed faults can be obtained.

Measuraments

- The simulator allows error propagation and fault activation to be estimated.
- The error propagability analysis consists in measuring the fraction of simulation runs in which at least one flip-flop in the system stores a faulty value, meaning that the fault has been propagated from the fault location to at least one memory element.
- The fault activability analysis consists in measuring the fraction of simulation runs in which the component affected by the fault produces an incorrect value.

An application: circuits from the ITC'99 benchmark

Characteristics of the benchmark

Circuit	LUTs	FFs	MUXs	IOBs	Function
b01	9	5	0	5	Compare serial flows
b02	4	4	0	3	Recognize binary coded decimal numbers
b03	76	37	0	9	Resource arbiter
b06	9	8	0	9	Interrupt handler
b07	152	51	20	10	Count points on a straight line
b08	40	21	0	14	Find inclusions in sequences of numbers
b09	53	28	0	3	Serial-to-serial converter
b10	52	24	0	18	Voting system
b11	147	38	14	14	Scramble string with var. cipher
b13	106	59	11	21	Interface to meteo sensors

An application: circuits from the ITC'99 benchmark

Effects of SEUs

Circuit	F_l	F_r	SA0	SA1	W-AND	W-Mix	Bridge
b01	129	547	708	2,944	5	7	0
b02	55	304	118	339	5	7	102
b03	963	5,910	8,105	21,661	1,423	1,431	2,320
b06	113	566	372	790	0	18	305
b07	1,730	10,431	18,331	47,762	4,085	3,911	4,739
b08	518	2,689	3,074	8,061	464	496	1,217
b09	695	3,872	6,569	15,948	567	512	1,908
b10	678	3,942	4,603	10,727	482	692	1,498
b11	1,790	10,104	14,059	35,749	3,537	3,536	4,480
b13	1,237	7,203	10,390	27,720	1,143	1,387	3,602

Comparison between simulation and prototype- based fault injection

Circuit	F_t	D_{sim}	D_{fi}	O_{sim} (%)	O_{fi} (%)
b01	676	676	676	100.0	100.0
b02	359	352	350	98.0	97.5
b03	6,873	3,285	3,278	47.8	47.7
b06	679	670	670	98.7	98.7
b07	12,161	927	927	7.6	7.6
b08	3,207	157	157	4.9	4.9
b09	4,567	2,081	2,080	45.5	45.5
b10	4,620	3,548	3,545	76.8	76.7
b11	11,894	7,521	7,519	63.2	63.2
b13	8,440	2,517	2,515	29.8	29.7

- simulation method can accurately reproduce the effects of SEUs affecting any configuration bit of an SRAM-based FPGA system
- error ranging between 0.0% and 0.5% for this benchmark (0.1% on average).

Estimated SEU Observability, Stuck-at Model

same input patterns used in the previous experiment

Circuit	F_{sa}	D_{sa}	O_{sa}
b01	28	28	100.0%
b02	24	24	100.0%
b03	170	78	45.9%
b06	36	36	100.0%
b07	324	17	5.1%
b08	108	23	2.1%
b09	112	42	37.6%
b10	140	107	76.2%
b11	322	259	80.5%
b13	254	69	27.1%

D_{sa} of detected failures with respect to the number F_{sa} of possible stuck-at faults, at zero and at one, in LUTs and buffers, giving the fault observabilities (O_{sa}) reported in the table.