



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Design and Verification of Safe and Secure VLSI Systems*

Juan-Carlos Ruiz-García

jcruizg@disca.upv.es

* Research partially funded by the Ministerio de Economía, Industria y Competitividad de España under grant agreement number TIN2016-81075-R



Who Am I?

**Universitat Politècnica
de València**

www.upv.es

+34 96 387 7000, extension: 75752

jcruizg@disca.upv.es

Department of Computer Engineering (DISCA)
Research Institute ITACA-UPV
Building 1G, office: 2S12
Universitat Politècnica de València (UPV)
Camino de Vera s/n
46022 - Valencia (Spain)



Juan-Carlos Ruiz
Associate Professor



Campuses	3
Centers	13
Students	36,187
Faculty and research personnel	2,843
Personnel of administration and services	2,396
2017 Budget	313.4 M€
R+D+i financial volume	70.5 M€
Departments	41
Research institutions	40
Bachelor's degrees	27
Master's degrees	56
PhD programmes	28
Continuous training courses	1,439



Campus de Vera (Valencia)



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



My (double) profile



- Department of Computer Engineering lecturing
 - Computer engineering and networking
 - Industrial IT
 - Operating systems
 - Dependability and Cibersecurity
- 91 faculty members

■ Courses

- Computer architecture and engineering
- Development of mobile applications
- Security in mobile devices
- Adaptive fault-tolerant systems

Escola Tècnica
Superior d'Enginyeria
Informàtica  etsinf

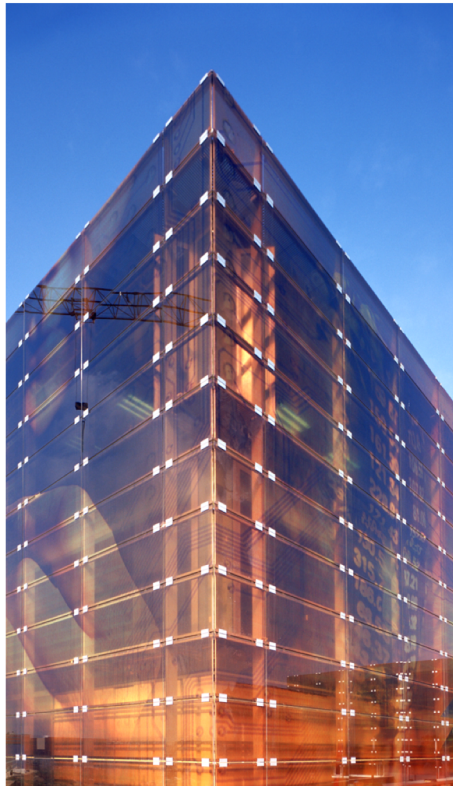
– **TELECOM**
UPV **VLC**

DISCA
DEPARTAMENTO DE INFORMÁTICA
DE SISTEMAS Y COMPUTADORES

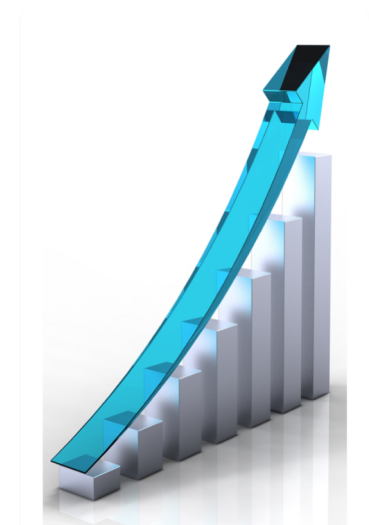
■ Academic coordination of the UPV specialist in cybersecurity

- <https://cibersec.webs.upv.es>

CIBERSEC
Diploma Especialización en Ciberseguridad



- Research institute integrating 11 research groups (≈200 researchers) and 50 partners (including public entities and private companies) that benefit from the R&D activities of the institute
- 2013-2016 activity summary
 - Turnover: 23.8 M€ (25% growth of annual turnover)
 - More than 700 R&D projects
- Results:
 - Intellectual property results: 30 Patents
 - Research results:
 - 65 PhD thesis
 - >400 journal articles
 - >1200 conference papers



- 7 researchers
- Currently I have 2 on-going PhDs
- Research spotlight placed on embedded HW systems
 - Design of (dependable) VLSI systems
 - System (robustness) assessment
 - Optimization and Benchmarking of (resilient) HW designs
- Ongoing project
 - DINAMOS (TIN2016-81075-R): Dependable adaptive mechanisms for Autonomous and connected vehicles



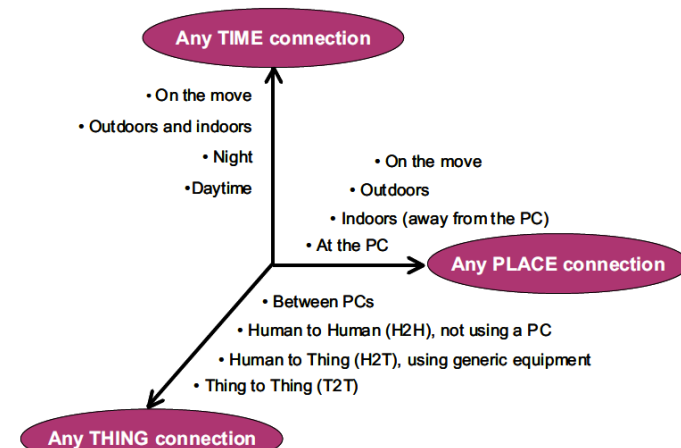
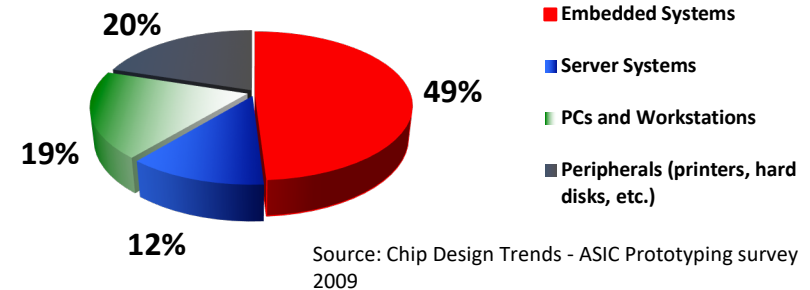
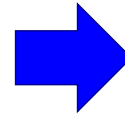
What is this talk about?





Source: 2015 International Technology Roadmap for Semiconductors (<http://www.itrs.net>)

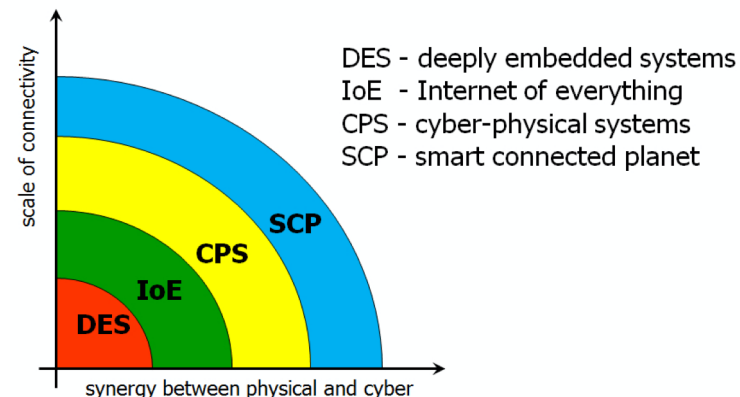
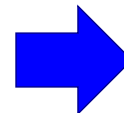
Only few of today's current CPU shipment targets the PC market



Source: ITU adapted from Nomura Research Institute

The integration of computation and physical processes relies on more complex and further interconnected systems to provide ANY * CONNECTION

The scale of such connection matters



Technical viewpoint



Adapted from CASAGRAS results, FP7 EU Project Number 216803 (<http://www.rfidglobal.eu>)

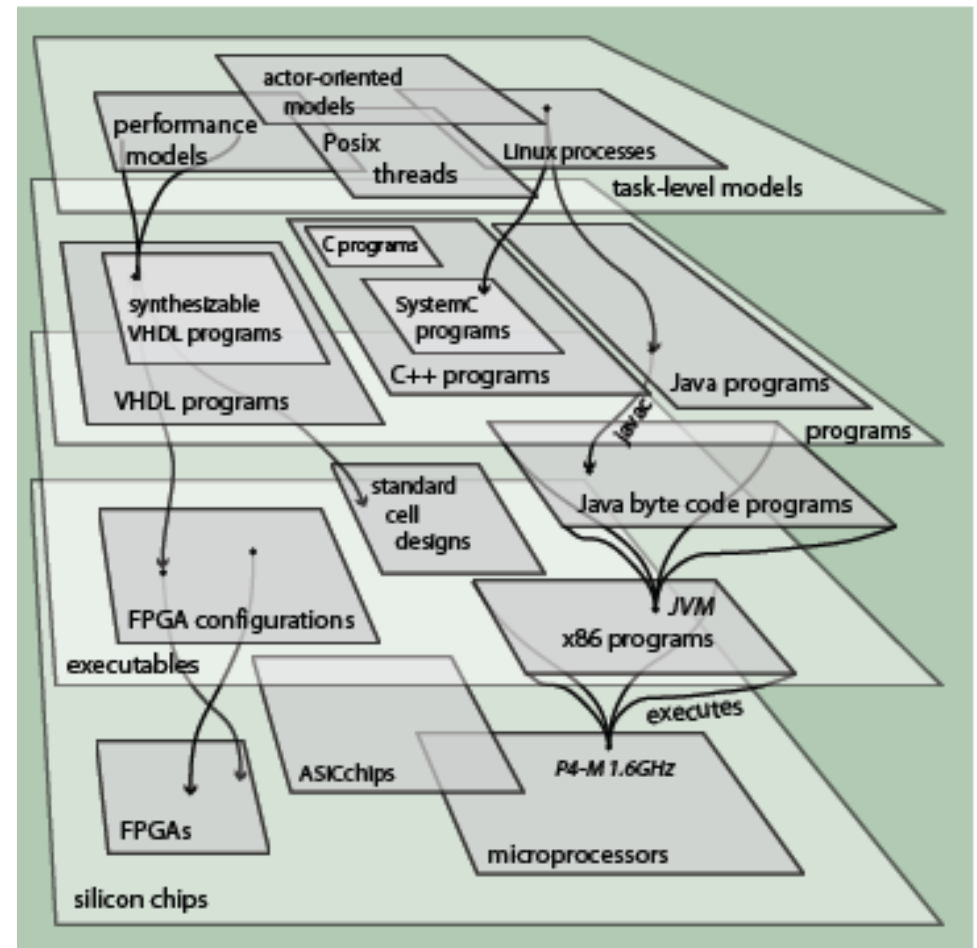
Why are dependability and security so important in this scenario?

- Any CPS solution must be designed with dependability in mind, i.e. considering its safety, security, reliability, availability, confidentiality, integrity ...



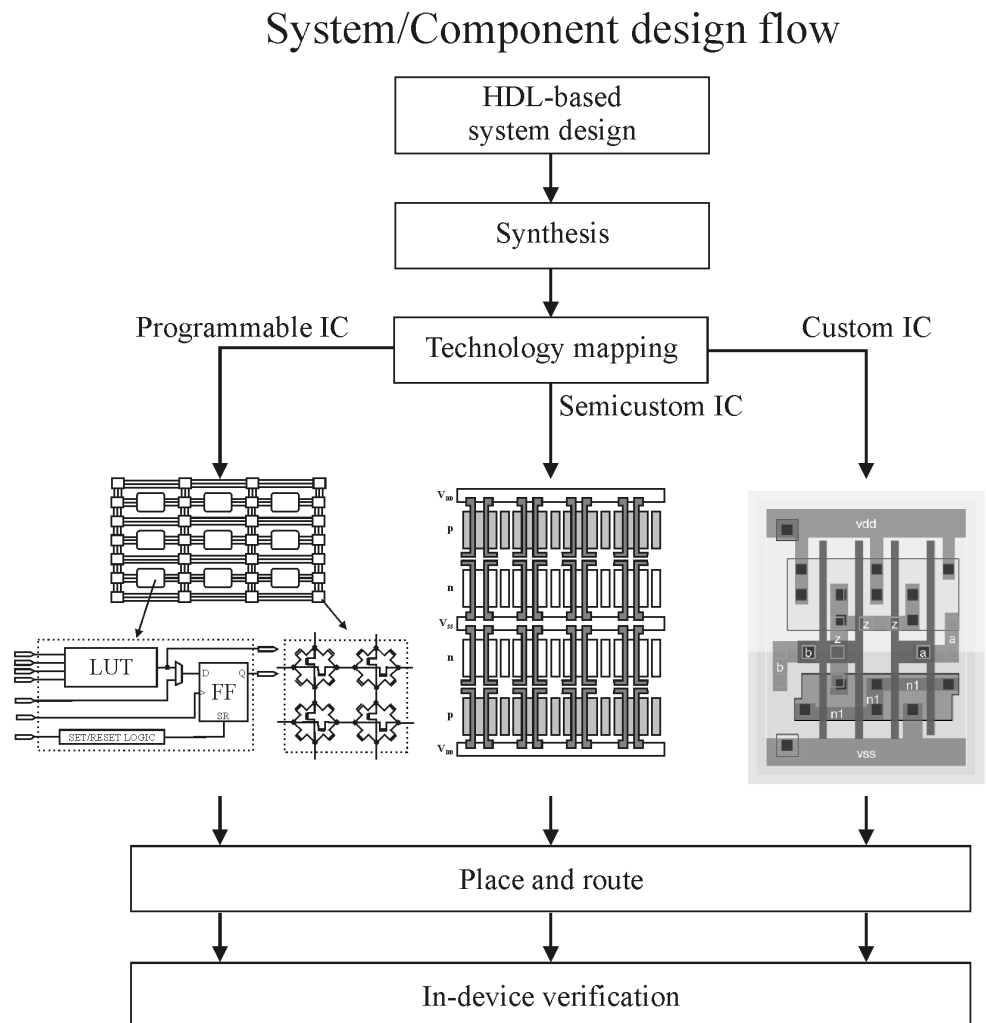
- It is not only a matter of designing dependability and security mechanisms, but also of assessing their proper behaviour [as stated in standards such as ISO 27000 (asset-critical systems) or fault injection for the ISO 26262 (automotive systems)]

- **IP cores** are modeled using **HW description languages**
 - VHDL, Verilog, SystemC, etc.
- Tools transforming models into implementations are called Electronic Design Automation (**EDA**) tools
- The implementation process has several steps defining the so-called Semicustom Design Flow
 - Synthesis
 - Technology mapping
 - Place & Routing



(from Prof. Edward Lee)

- **IP cores** are modeled using **HW description languages**
 - VHDL, Verilog, SystemC, etc.
- Tools transforming models into implementations are called Electronic Design Automation (**EDA**) tools
- The implementation process has several steps defining the so-called Semicustom Design Flow
 - Synthesis
 - Technology mapping
 - Place & Routing



Requirements

- Higher performance → optimization of critical paths, higher levels of parallelism ...
- Smaller devices → use of higher levels of integration ...
- Lower power consumption and Increasing autonomy → operation at low (and very close) voltage levels...
- Reduced (time-to-market and manufacturing) costs → use of IP cores, EDA tools, mastering manufacturing processes, outsourcing production ...

Requirements & Vulnerabilities

- Higher performance → optimization of critical paths, higher levels of parallelism ...
... but heating (wear-out) issues !!!
- Smaller devices → use of higher levels of integration ...
... but manufacturing defects or sensitivity to radiation and electromagnetic fields !!!
- Lower power consumption and Increasing autonomy → operation at low (and very close) voltage levels...
... but higher sensitivity to voltage variations !!!
- Reduced (time-to-market and manufacturing) costs → use of IP cores, EDA tools, mastering manufacturing processes, outsourcing production ...
... but assessment and verification takes time !!!

A great opportunity ...

- The development of CPS offers a great opportunity to HW core designers
 - Shenzhen: The Silicon Valley of Hardware
(<https://www.youtube.com/watch?v=SGJ5cZnoodY>)
- Because many things are changing
 - Manufacturing through integration
 - Increasing number of open source IP cores
 - Quality is becoming a must (even for Chinese products)

... but also a great challenge

- How to improve the dependability and security of HW components?
- How to cope with the simultaneous consideration of performance, power consumption, area and dependability?



- **Rapid (and automatic?) design** of safe and secure IP cores
- **Accurate evaluation**, including dependability assessment
- **Reduce time-to-market**: Use of EDA tools to optimize the implementation of HW designs
- **Fast selection of components**: Integration of *most suitable* IP cores (Benchmarking)

- **Rapid (and automatic?) design** of safe and secure IP cores
- **Accurate evaluation**, including dependability assessment
- **Reduce time-to-market**: Use of EDA tools to optimize the implementation of HW designs
- **Fast selection of components**: Integration of *most suitable* IP cores (Benchmarking)



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Using Open Compilation to Simplify the Design (and automate the deployment) of Safe and Secure VLSI Systems



- Problem Statement & Basic Concepts
- Open Compilation Approach
- NMR Fault Tolerance as a Metaprogram
- Conclusions

- HW systems developed by integrating custom & third-party IP cores
 - Rapid integration of new solutions
 - Reduction of time-to-market costs
- Hardware fault tolerance is a requirement for modern VLSI designs
 - As integration scales decreases, circuit fault sensitivity increases
 - Increasing need of using replication and ciphering techniques for protecting IP cores

- Fault tolerance strategies typically developed by hand and on a case-per-case basis
 - Applying the same strategy on different designs means performing various different developments
 - Development of FT strategies is error prone
- Need of separation of concerns in HW designs
 - Non-skilled designers develop FT mechanisms (the community agrees that this must be avoided)
 - Functional (system core) and non-functional (FT or security) can be separately designed, then they can be tangled



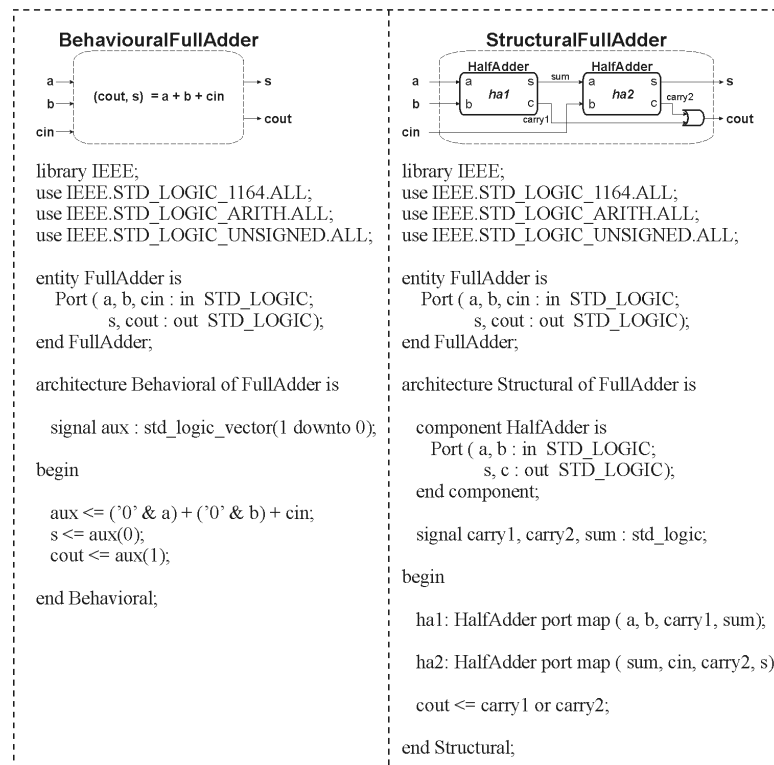
UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



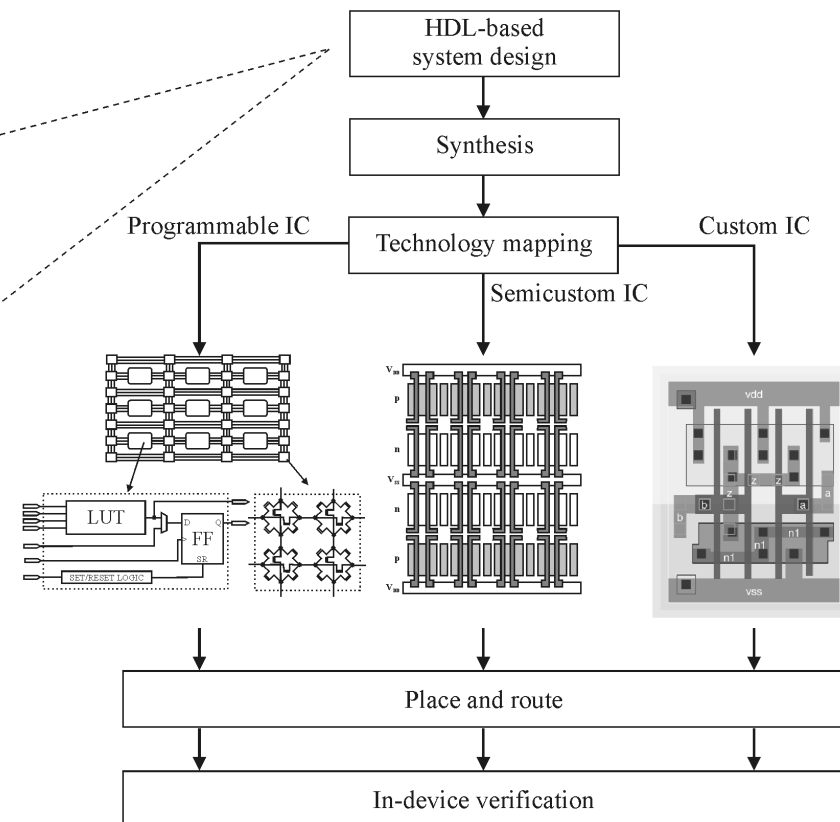
How to cope in practice with such separation of concerns?



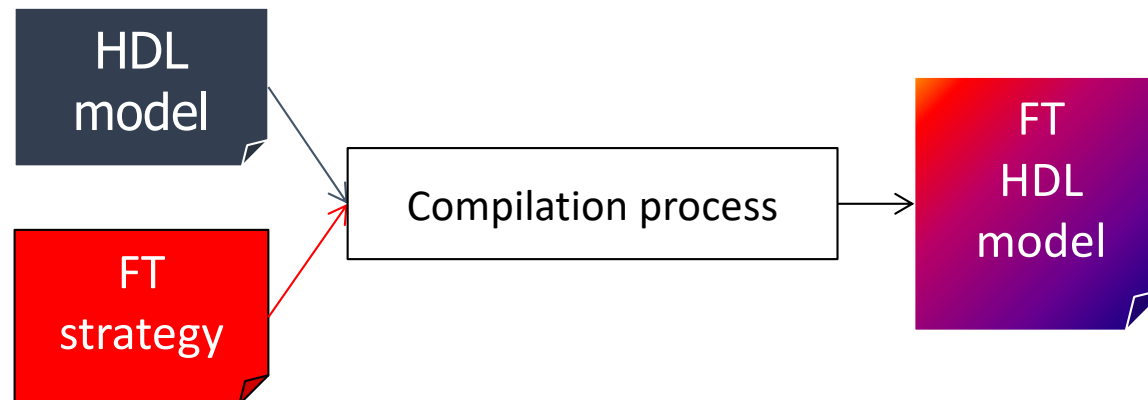
Typical HW design flow

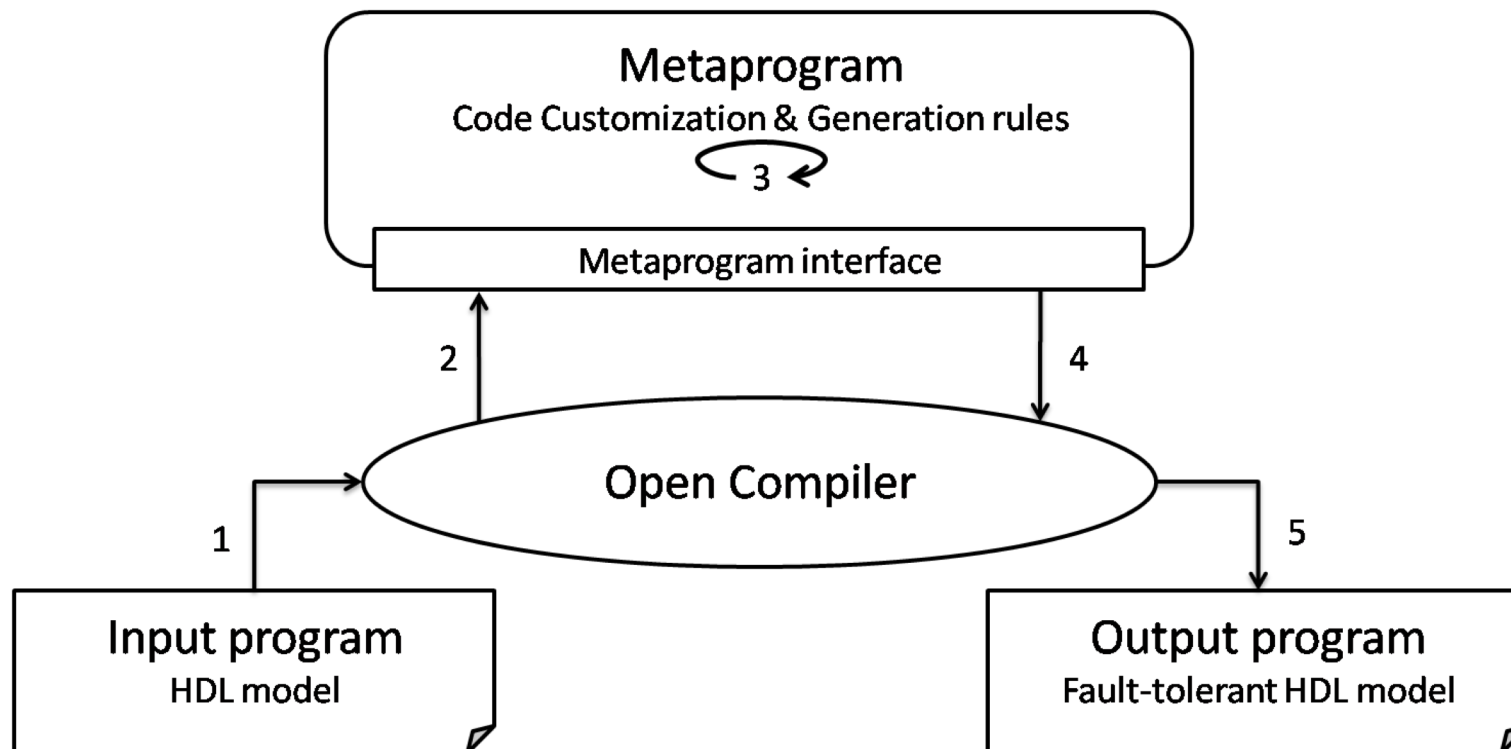


System/Component design flow



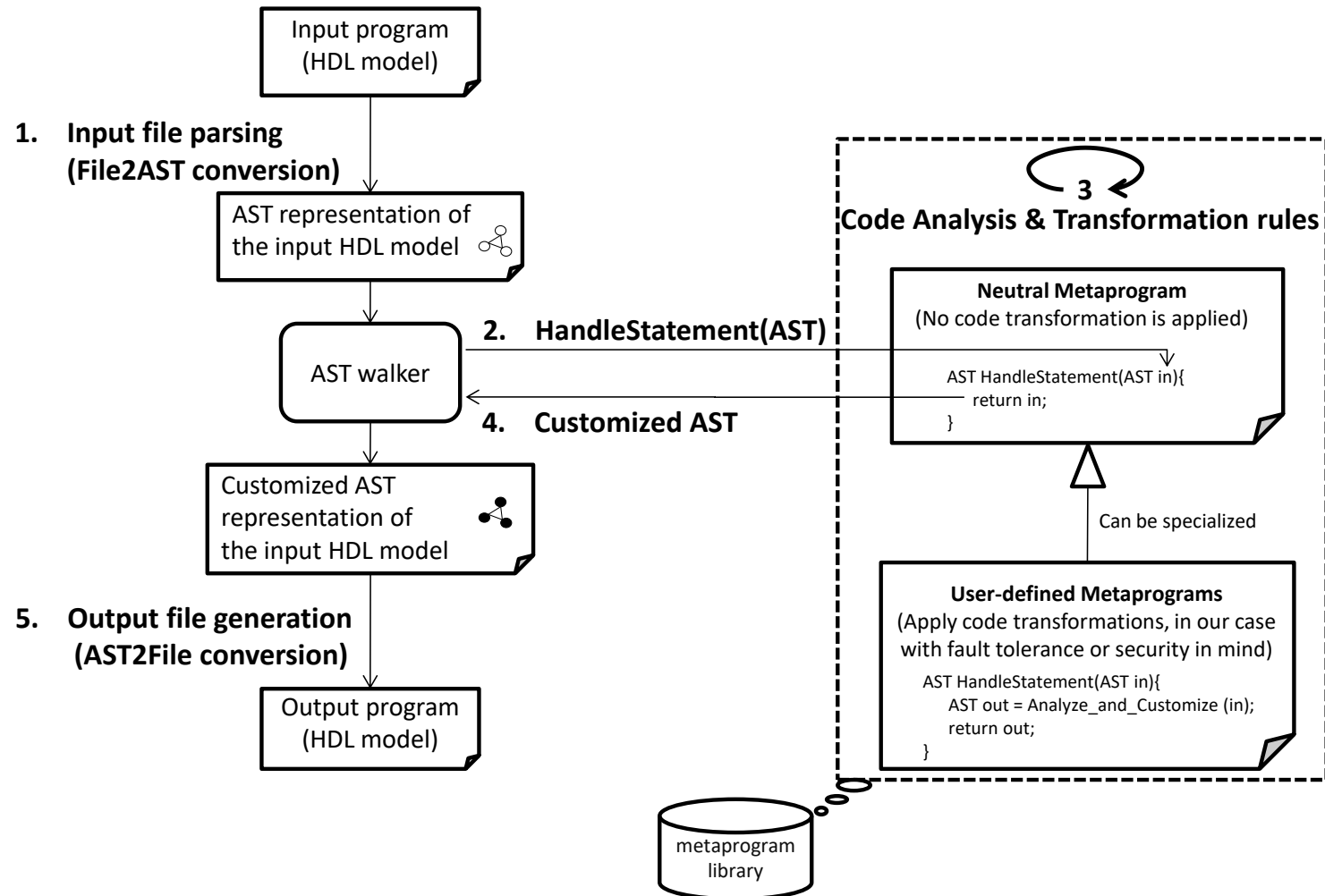
- Convert non-fault-tolerant HDL designs into fault-tolerant ones





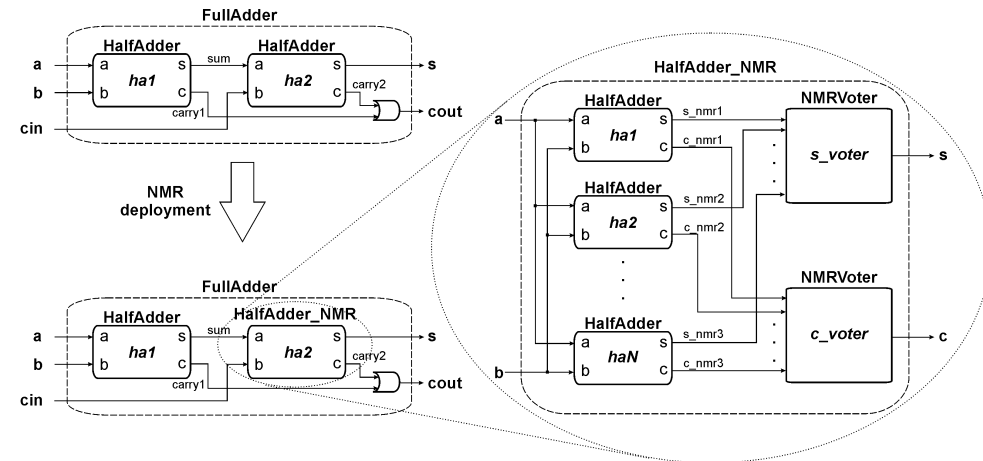
The CODESH Approach

*An open **CO**mpilation
process for the design
of **DE**pendable and
SEcure **HDL**-based
systems (CODESH)*



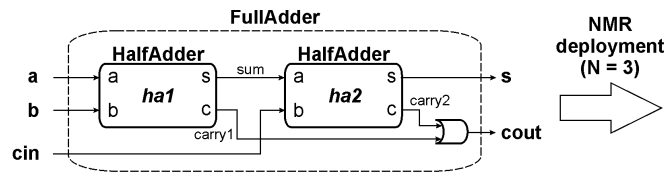
- Designing a FT strategy means developing a metaprogram
 - Which are the program statements that must be handled at the metaprogram?
 - Which are the customization and generation rules that must be applied on each statement?

- Example of a FullAdder core:

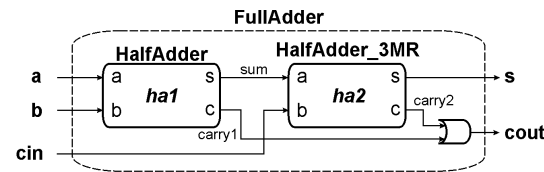


- Code Customisation and Generation rules
 - Customise the original model by replacing selected cores by their corresponding (replicated) versions (rules C1 & C2)
 - Generate the model of the replicated (NMR) component (rule G1)
 - Generate the majority voting infrastructure (rule G2)

Customisation of the original system model



NMR
deployment
(N = 3)



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity FullAdder is
    Port ( a, b, cin : in  STD_LOGIC;
           s, cout : out STD_LOGIC);
end FullAdder;
```

architecture Behavioral of FullAdder is

```
    component HalfAdder is
        Port ( a, b : in  STD_LOGIC;
              s, c : out STD_LOGIC);
    end component;
```

```
    signal carry1, carry2, sum : std_logic;
```

begin

```
    ha1: HalfAdder port map ( a, b, carry1, sum);
    ha2: HalfAdder port map ( sum, cin, carry2, s);

    cout <= carry1 or carry2;
```

end Behavioral;

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity FullAdder is
    Port ( a, b, cin : in  STD_LOGIC;
           s, cout : out STD_LOGIC);
end FullAdder;
```

architecture Behavioral of FullAdder is

```
    component HalfAdder is
        Port ( a, b : in  STD_LOGIC;
              s, c : out STD_LOGIC);
    end component;
```

HalfAdder_3MR
component
declaration

```
component HalfAdder_3MR is
    Port ( a, b : in  STD_LOGIC;
          s, c : out STD_LOGIC);
end component;
```

```
    signal carry1, carry2, sum : std_logic;
```

begin

```
    ha1: HalfAdder port map (a, b, carry1, sum);
    ha2: HalfAdder_3MR port map (sum, cin, carry2, s);

    cout <= carry1 or carry2;
```

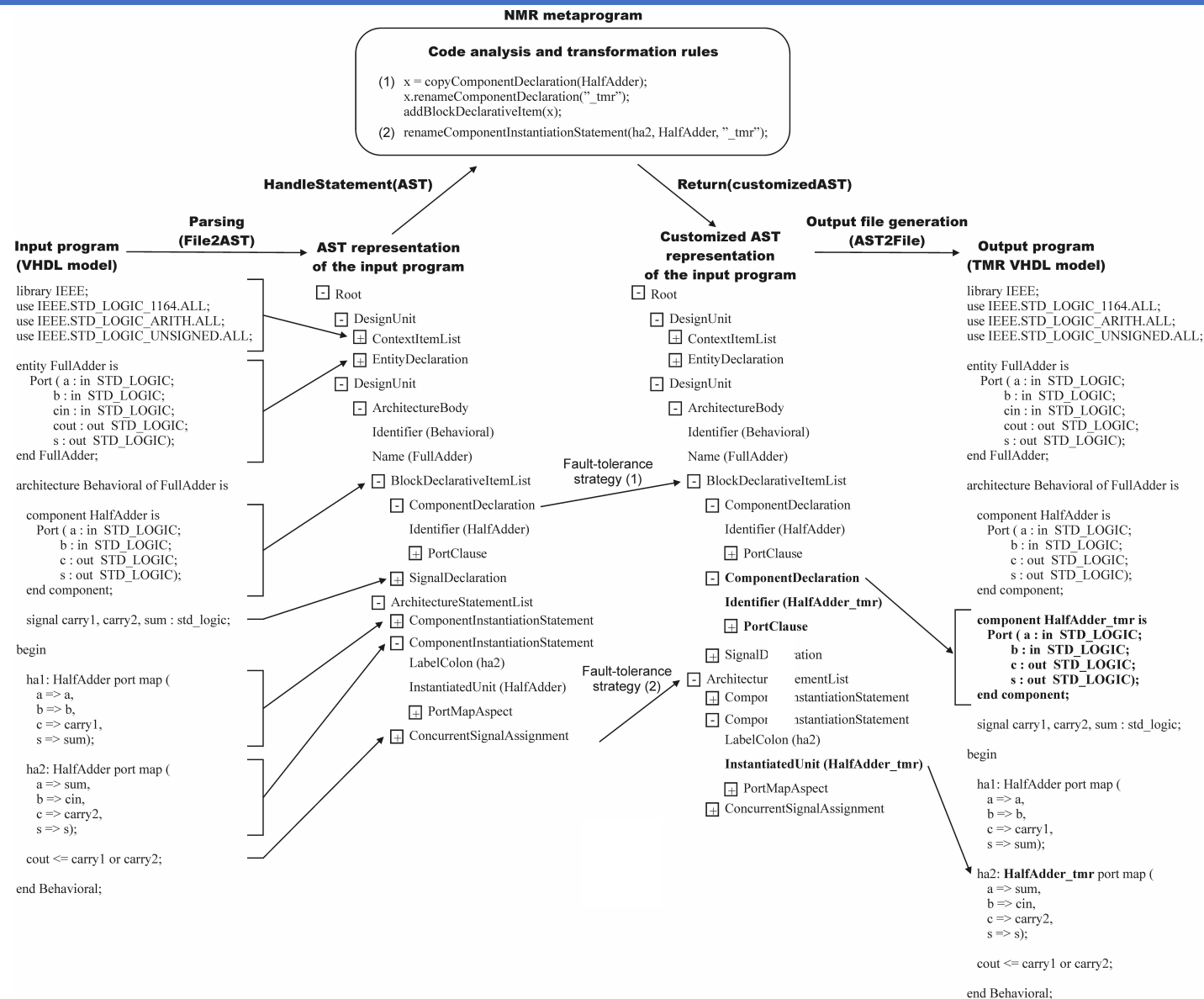
end Behavioral;

Customisation Rules

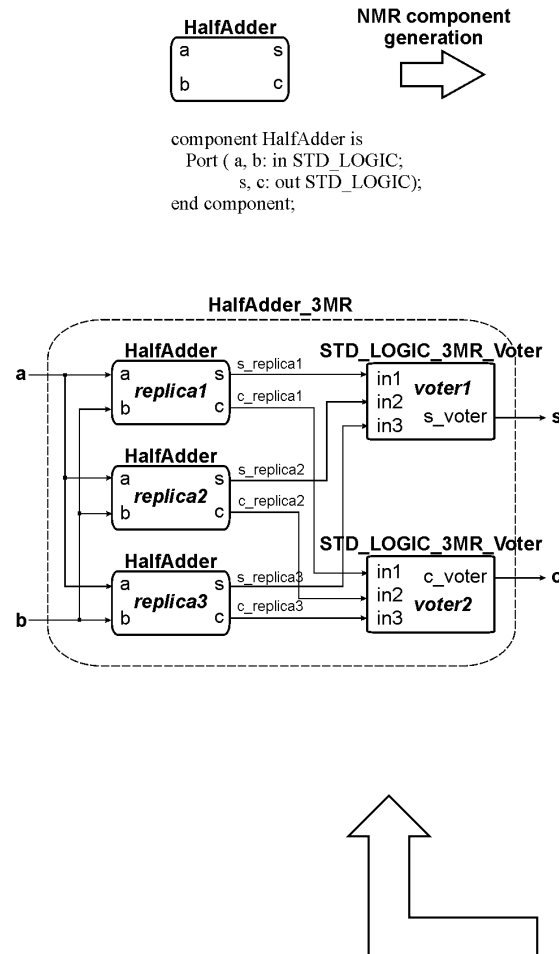
C1: Append the new generated NMR component declaration to the list of component already declared in the model under compilation

C2: Modify the required component instantiation statements to refer to the component NMR version

Customization rules (how are rules C1 and C2 applied?)



Generating the NMR version of the selected component (rule G1)



G1: Generation Rule

Step 0 - Information extraction

- (1) Component name: HalfAdder
- (2) Port name: a, b, s, c
- (3) Port mode: in, in, out, out
- (4) Port type: STD_LOGIC, STD_LOGIC, STD_LOGIC, STD_LOGIC
- (5) Number of replicas: 3

Step 1 - context item list (constant)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

Step 2 - NMR entity declaration (1, 2, 3, 4, 5)

```
entity HalfAdder_3MR is
Port ( a, b: in STD_LOGIC;
s, c: out STD_LOGIC);
end HalfAdder_3MR;
```

Step 3 - architecture identification (1, 5)

architecture HalfAdder_3MR_arch of HalfAdder_3MR is

Step 4 - NMR component declaration (1, 2, 3, 4, 5)

```
component HalfAdder is
Port ( a, b: in STD_LOGIC;
s, c: out STD_LOGIC);
end component;
```

Step 5 - NMR Voter component declaration (4, 5)

```
component STD_LOGIC_3MR_Voter is
Port ( in1, in2, in3 : in STD_LOGIC;
vote: out STD_LOGIC);
end component;
```

Step 6 - auxiliary signals declaration (2, 3, 4, 5)

```
signal c_replica1, c_replica2, c_replica3, c_voter: STD_LOGIC;
signal s_replica1, s_replica2, s_replica3, s_voter: STD_LOGIC;
```

begin

Step 7 - replicas component instantiation (1, 2, 3, 5)

```
replica1: HalfAdder port map(a, b, c_replica1, s_replica1);
replica2: HalfAdder port map(a, b, c_replica2, s_replica2);
replica3: HalfAdder port map(a, b, c_replica3, s_replica3);
```

Step 8 - NMR Voter component instantiation (2, 3, 5)

```
voter1: STD_LOGIC_3MR_Voter port map (c_replica1, c_replica2, c_replica3, c_voter);
voter2: STD_LOGIC_3MR_Voter port map (s_replica1, s_replica2, s_replica3, s_voter);
```

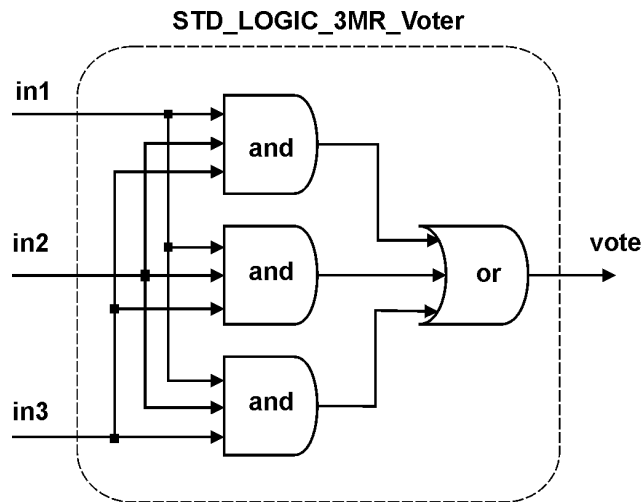
Step 9 - concurrent signal assignment statements (2, 3)

```
c <= c_voter;
s <= s_voter;
```

Step 10 - architecture end (1, 5)

```
end HalfAdder_3MR_arch;
```

Majority voter generation (rule G2)



G2: Generation Rule

Step 0 - Information extraction
(1) Port type: STD_LOGIC
(2) Number of replicas: 3

Step 1 - context item list
(constant)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

Step 2 - NMR Voter
entity declaration
(1, 2)

```
entity STD_LOGIC_3MR_Voter is
port (in1, in2, in3: in STD_LOGIC;
      vote: out STD_LOGIC);
end STD_LOGIC_3MR_Voter;
```

Step 3 - architecture
identification (1, 2)

```
architecture STD_LOGIC_3MR_Voter_archi of STD_LOGIC_3MR_Voter is
begin
```

Step 4 - concurrent signal
assignment statement
(combination(2))

```
vote <=
(in1 and in2) or
(in1 and in3) or
(in2 and in3);
```

Step 5 - architecture end (1, 2)

```
end STD_LOGIC_3MR_Voter_archi;
```

Resulting metaprogram

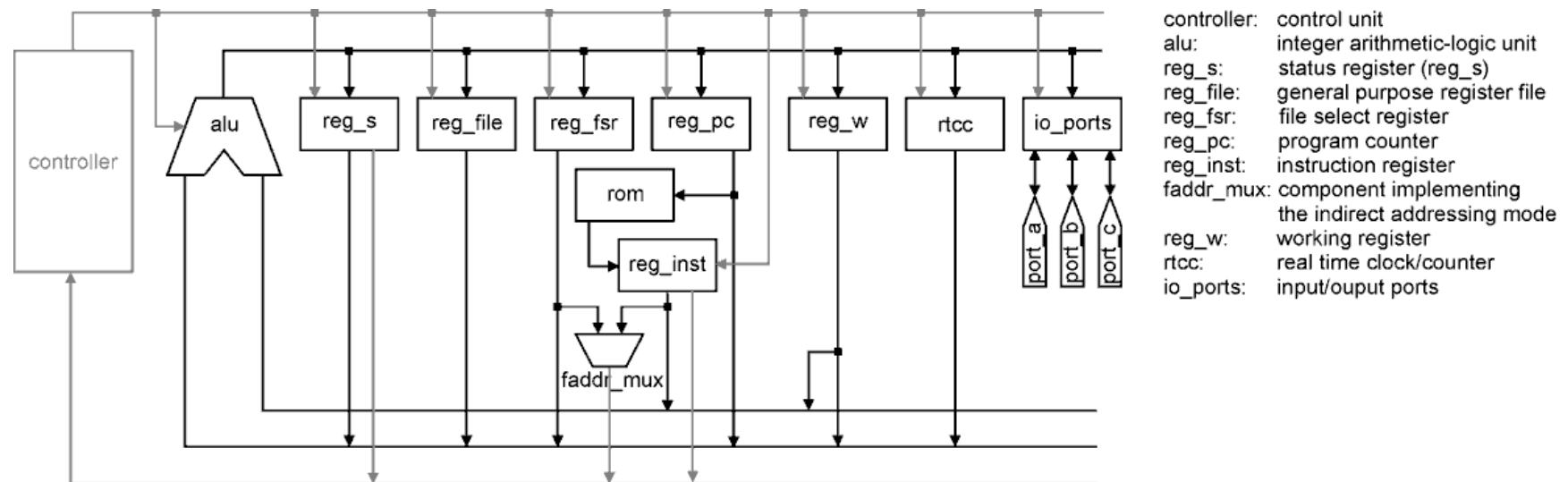


```

class NMR_Metaprogram extends Metaprogram{
    ComponentDeclaration handleComponentDeclarationStatement(ComponentDeclaration cd){
        Apply_Customization_rule_c1(cd, OC.getTargetComponentList(),
                                     OC.getComponentDeclarationList(), OC.getNumberOfReplicas());
        Apply_Generation_rule_g1(cd, OC.getNumberOfReplicas());
        Apply_Generation_rule_g2(cd, OC.getNumberOfReplicas());
    }
    ComponentInstantiation handleComponentInstantiationStatement(ComponentInstantiation ci){
        Apply_Customisation_rule_c2(ci, OC.getComponentInstantiationList(), OC.getNumberOfReplicas());
    }
}
  
```

- The CODESH metaprogram library
 - NMR (including TMR), Hamming codes (ECC, SEC-DED), Time redundancy, Data encryption (DES) ...
- Which is the price to pay for the use of the library?
 - An automatic deployment approach is interesting but consider dependability benefits wrt performance, taken area and power consumed by resulting designs
 - When various alternatives exist consider the use of dependability benchmarking
- Which system component should be replicated?
 - Study the behaviour of the system in the presence of faults in order to detect where fault tolerance is required, suitable or not necessary (use of fault injection techniques)
 - Take into account the different types of faults and fault models that are today representative in CMOS systems

■ Experiments target: PIC 16C5X microcontroller



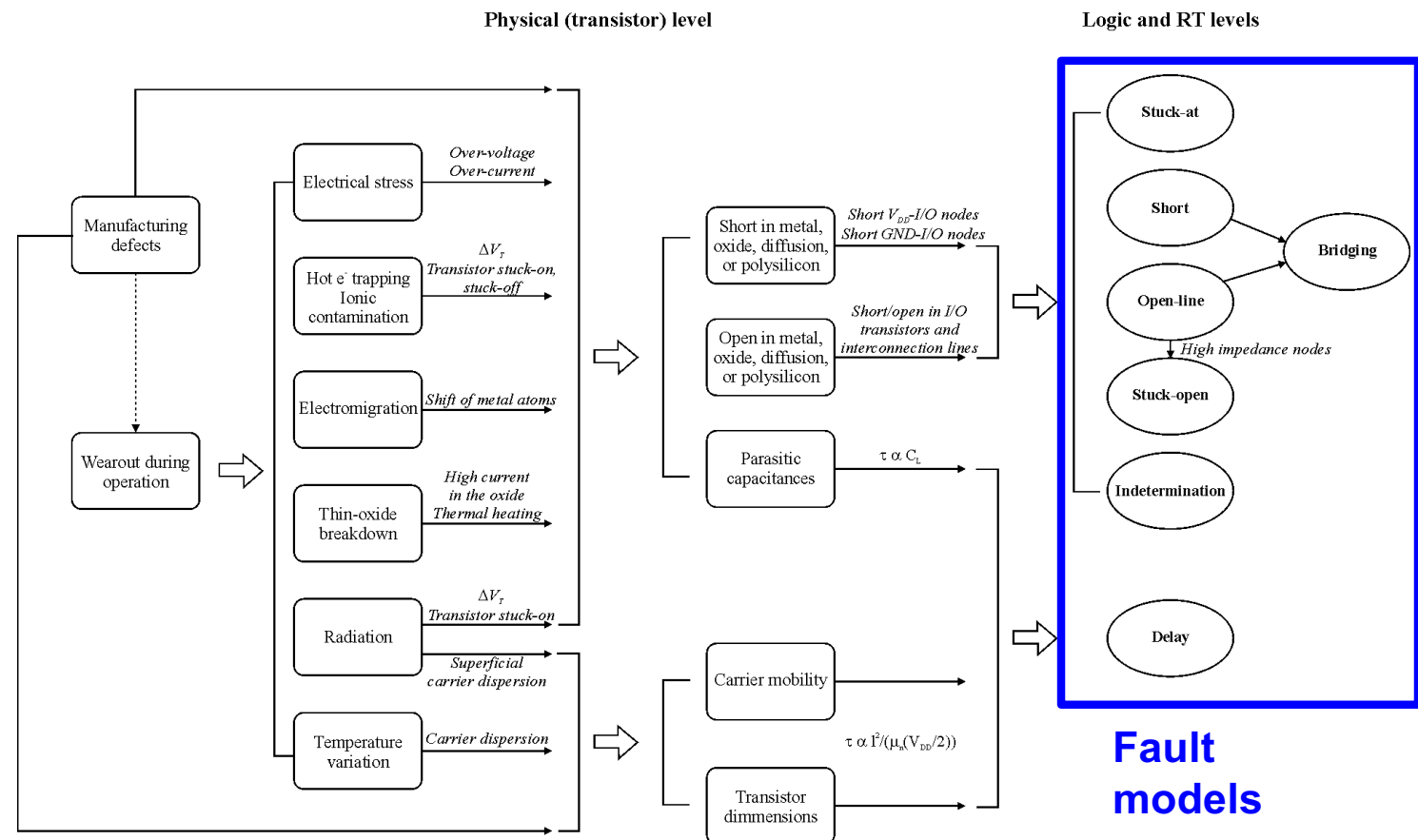
■ Fault injection experiments carried out using the FADES fault injector

- FPGA-based framework for the Analysis of the Dependability of Embedded Systems

- With the continuous downscaling of CMOS technologies, the device reliability has become a major bottleneck in embedded systems

Permanent Faults

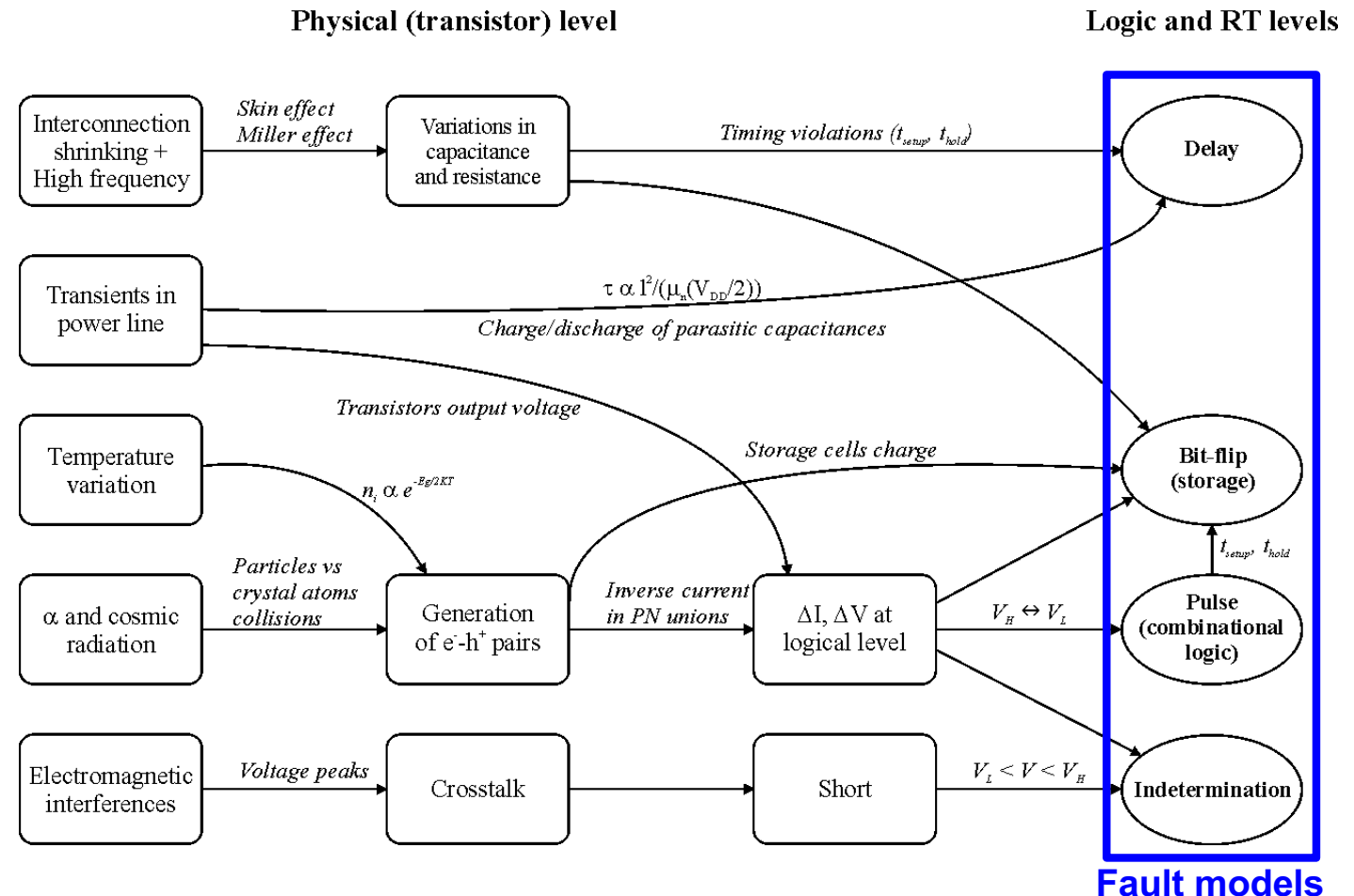
... caused irreversible physical defects during production and operation



- With the continuous downscaling of CMOS technologies, the device reliability has become a major bottleneck in embedded systems

Transient Faults

... caused by environmental conditions like temperature, humidity, pressure, voltage, power supply, vibrations, fluctuations, electromagnetic interference, ground loops, cosmic rays and α -particles



- Workload
 - Signal filtering operations, which typically consist in 2D convolution algorithms.
- Faultload
 - *Bit-flips* for transient faults targeting the sequential core logic
 - *Stuck-at-1s* for permanent faults targeting its combinational logic.
- Number of experiments
 - Determined to obtain an experimentation confidence degree of 0.99 (3309 experiments on sequential logic and 9738 on combinational logic)

//Generation of a PIC microcontroller embedding a 3MR working_register

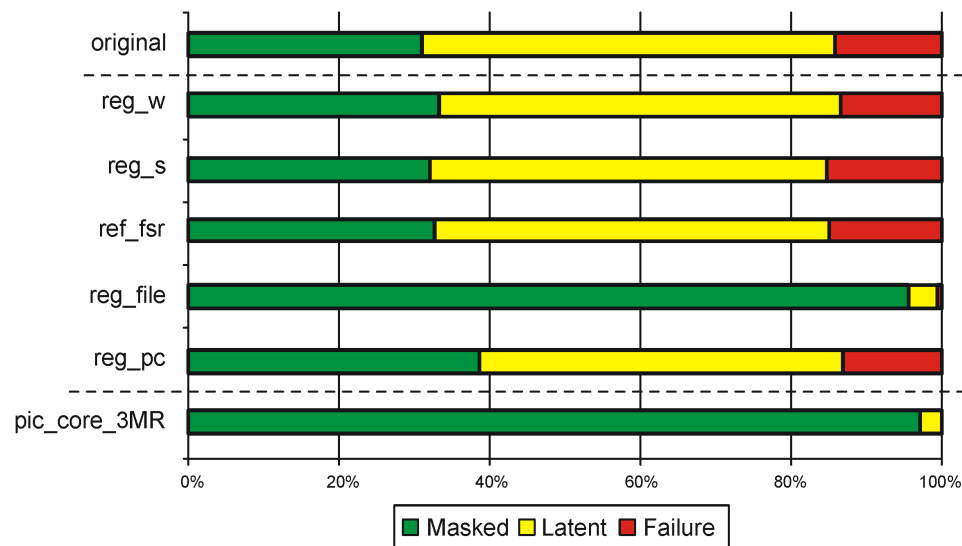
```
$java codesh          --metaclass FTlib.NMR -- inputModel PIC_core.vhd  
                      --outputModel PIC_with_working_register-3MR.vhd ---NumberReplicas 3  
                      ---TargetComponentDeclaration reg_w  
                      ---TargetComponentInstace working_register
```

//Generation of a 3MR version of the PIC microcontroller

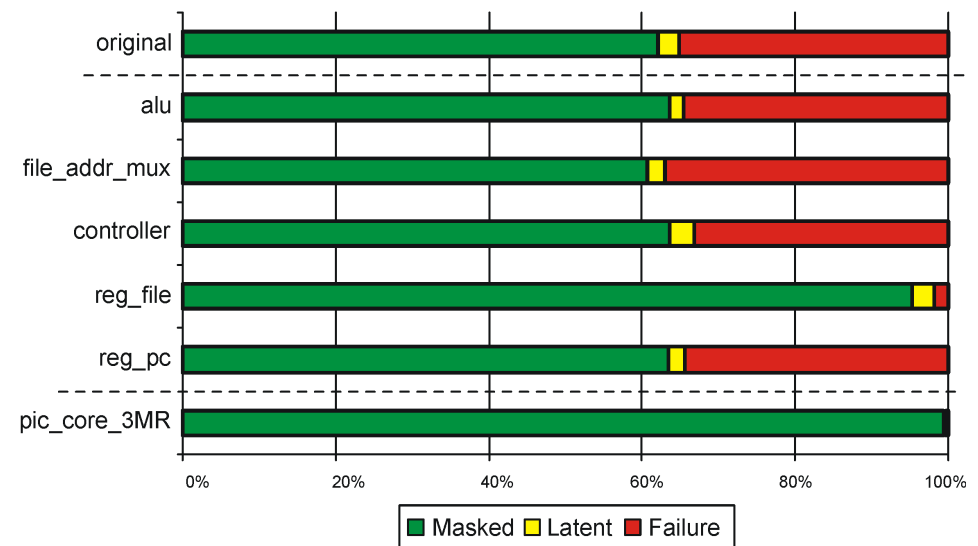
```
$java codesh          --metaclass FTlib.NMR -- inputModel Design_using_PIC_core.vhd  
                      --outputModel PIC_core_3MR.vhd ---NumberReplicas 3  
                      ---TargetComponentDeclaration PIC_core
```

Impact of faults on the 3MR version of the PIC

Sequential logic (targeted by bit-flips)

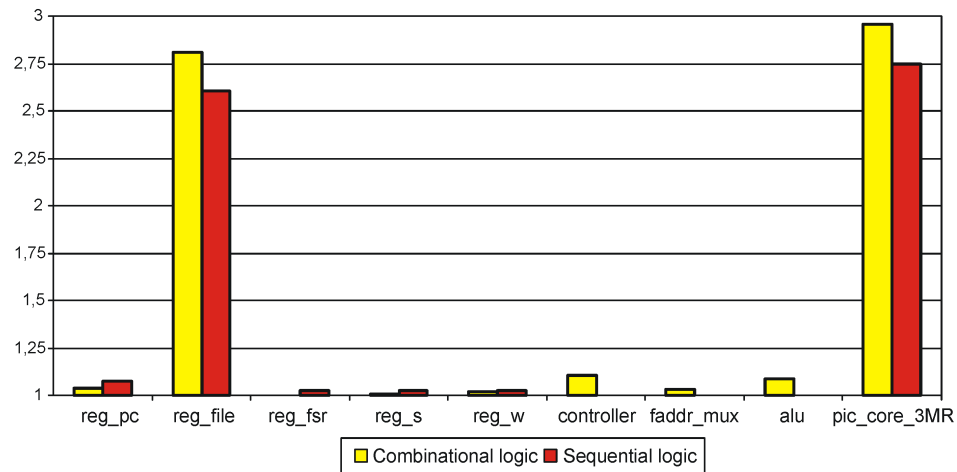


Combinational logic (targeted by stuck-at-1s)

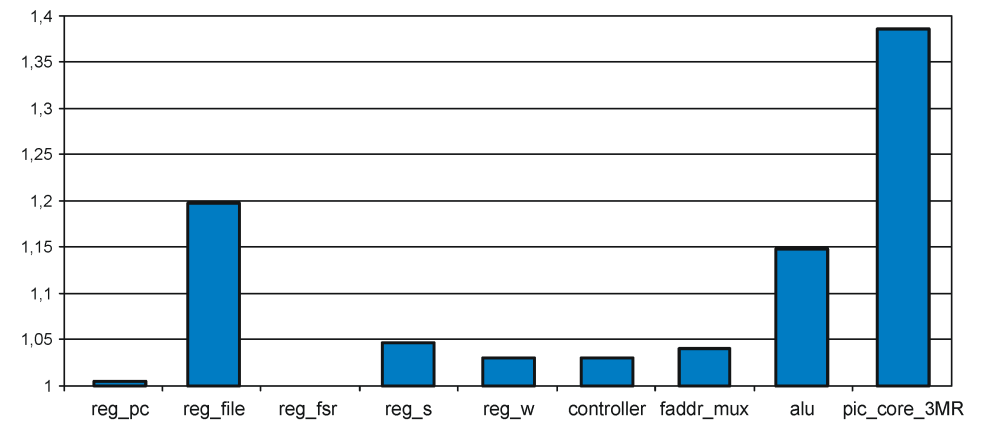


Increasing area and clock speed factor

Increasing area factor

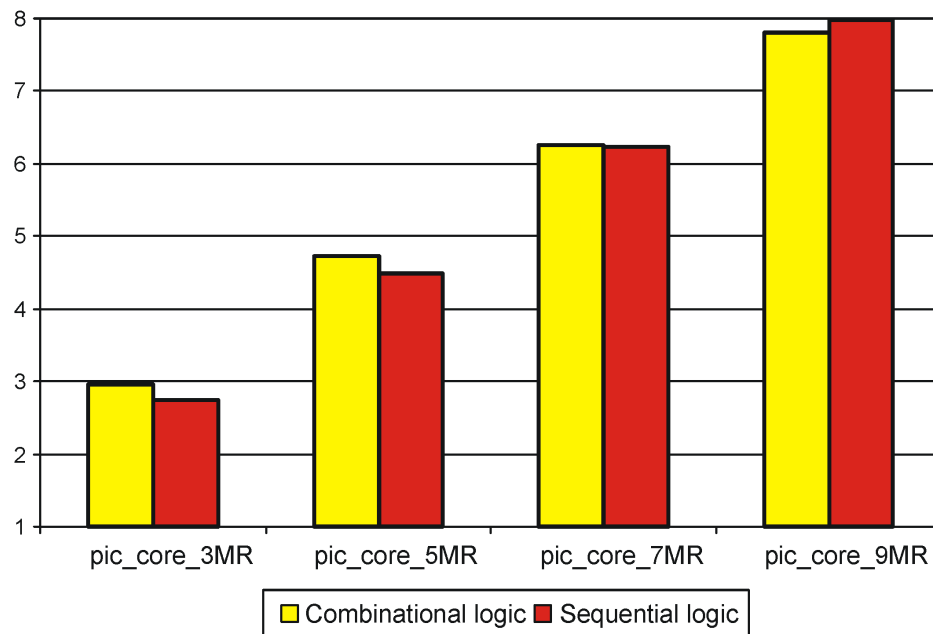


Increasing clock period factor

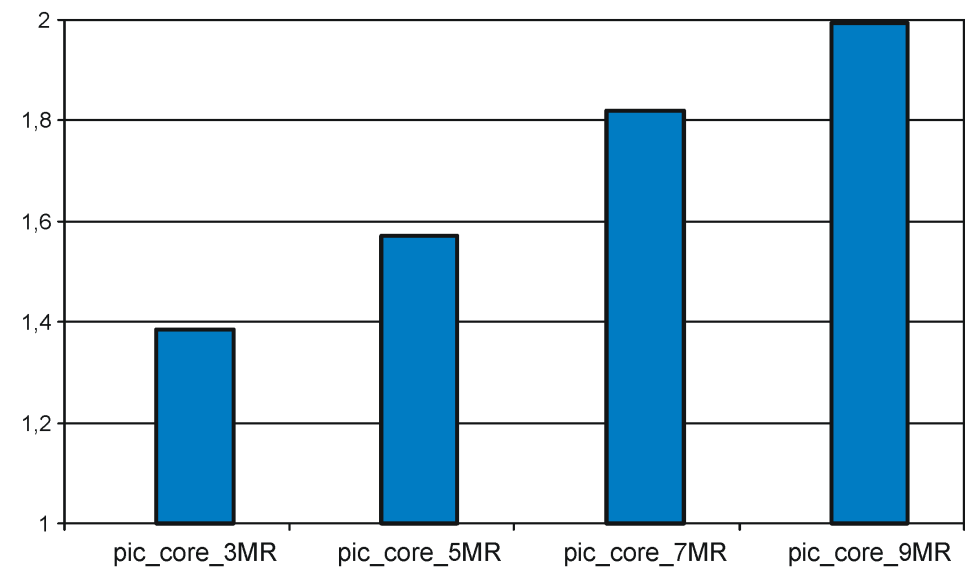


Other generated NMR versions

Increasing area factor



Increasing clock period factor



- Open compilation seems quite promising to increase FT hardware design productivity and efficiency, while reducing time-to-market costs
- HW FT mechanisms can be designed, changed and evolved independently from the rest of the system
- Separation of mechanisms eases the use (and reuse) of a given FT strategy on different VLSI designs
- The proposal simplifies adaptation of VLSI designs attending to a particular set of dependability needs

- **Rapid (and automatic?) design** of safe and secure IP cores
- **Accurate evaluation**, including dependability assessment
- **Reduce time-to-market**: Use of EDA tools to optimize the implementation of HW designs
- **Fast selection of components**: Integration of *most suitable* IP cores (Benchmarking)

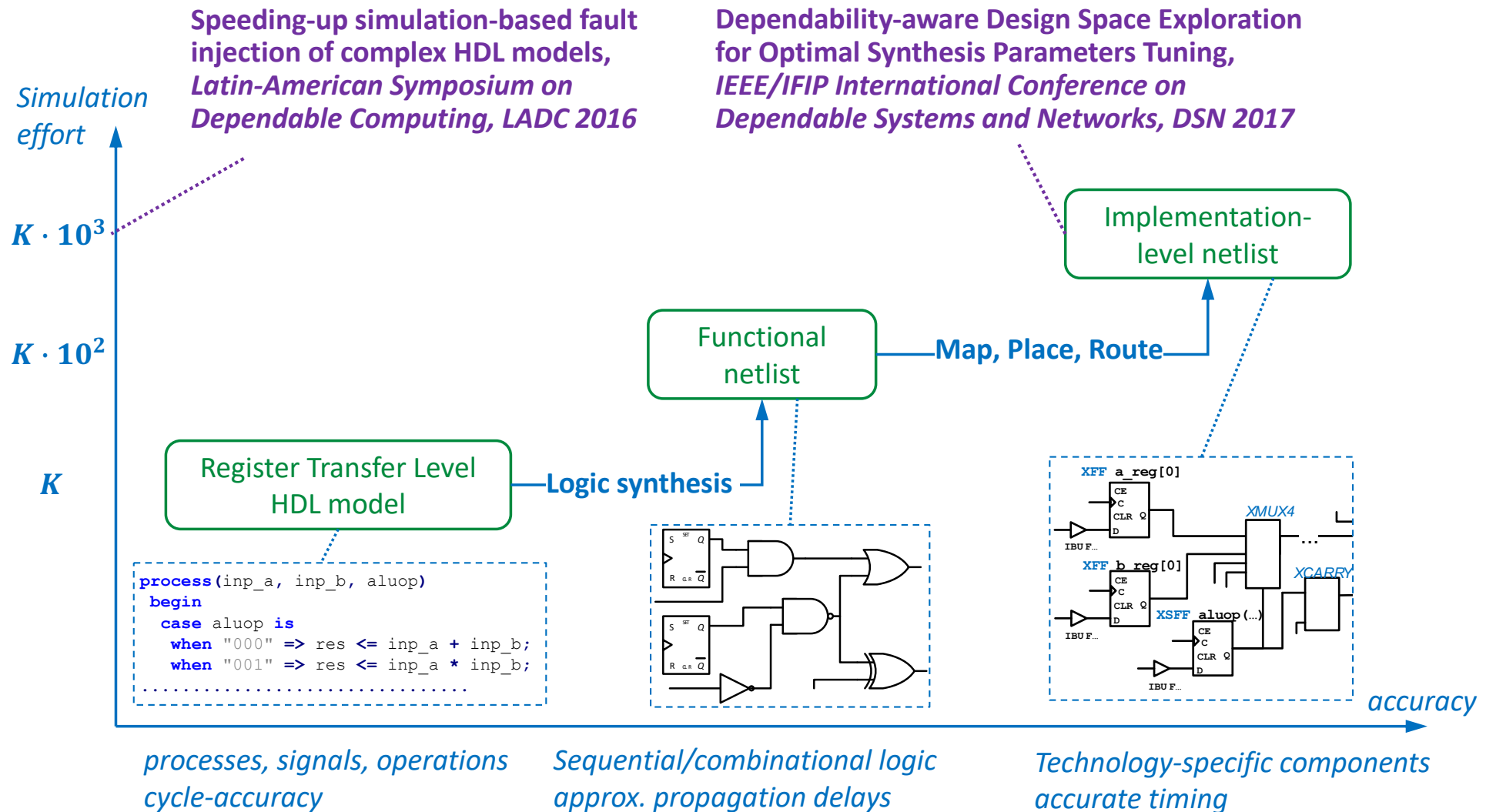


UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Accurately simulating the effects of faults in VHDL models described at the implementation-level



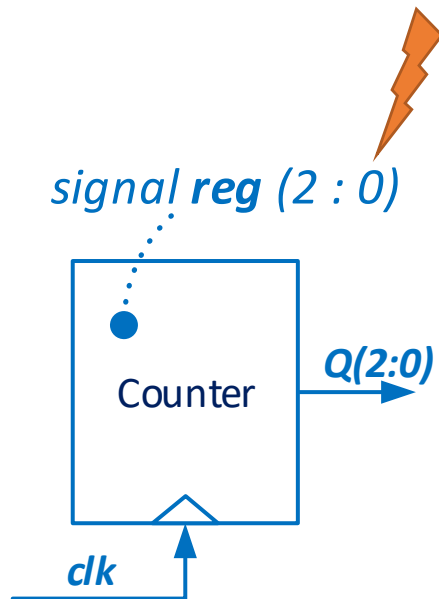


SBFI techniques:

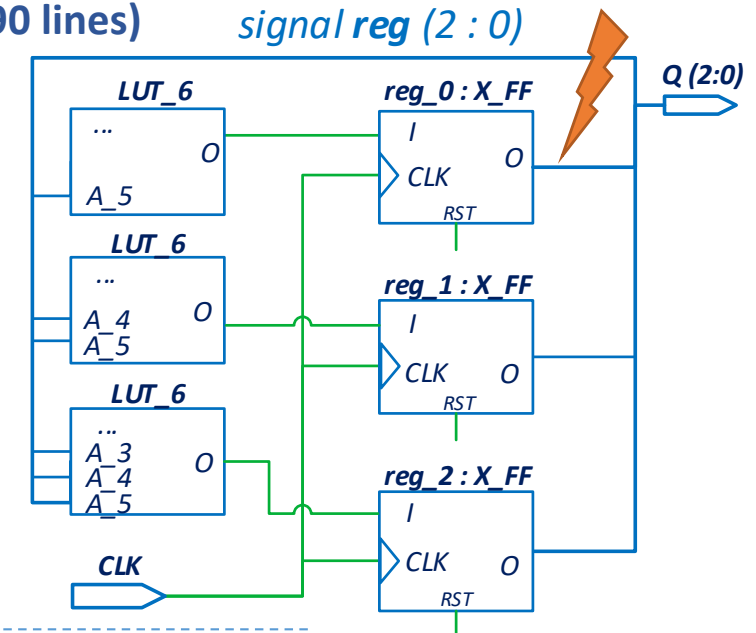
- *Simulator commands*
 - *Mutants*
 - *Saboteurs*
- intrusive

Bit-flip: RTL vs Implementation level

Register Transfer Level (28 lines)

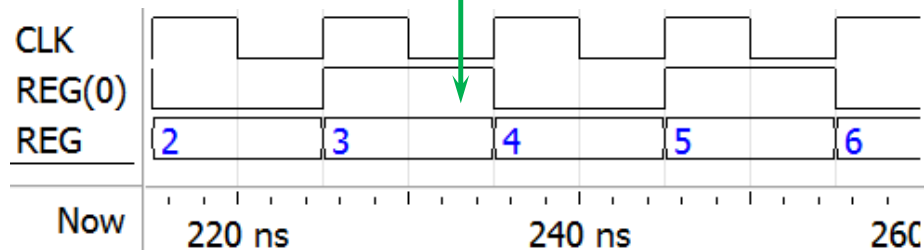


Implementation-level Netlist (290 lines)

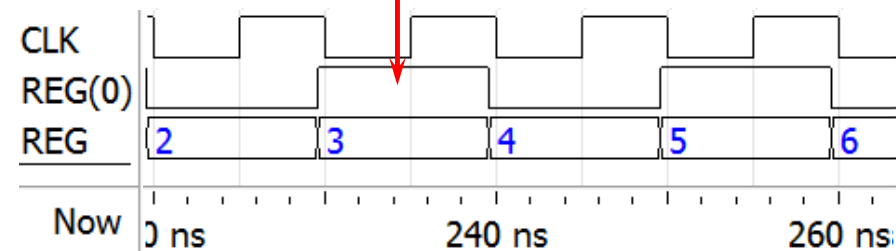


bit-flip

```
set cv [examine cnt/reg(0)]
if {$cv == 1} { set nv 0 } else { set nv 1 }
force -deposit cnt/reg(0) $nv 0
```



stuck-at?



VHDL initiative towards ASIC libraries (VITAL)

- Created by industry consortium
- Accelerate/unify the development of ASIC libraries

Vendors of EDA tools

Mentor Graphics,
Cadence,
...



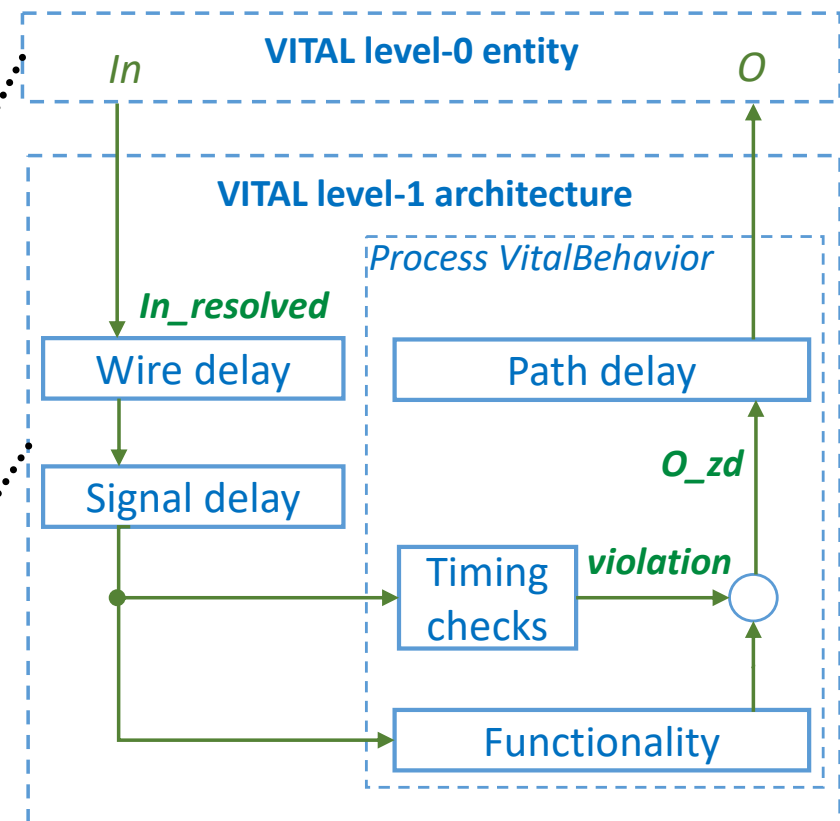
Manufacturers

Xilinx,
Altera,
TSMC
...

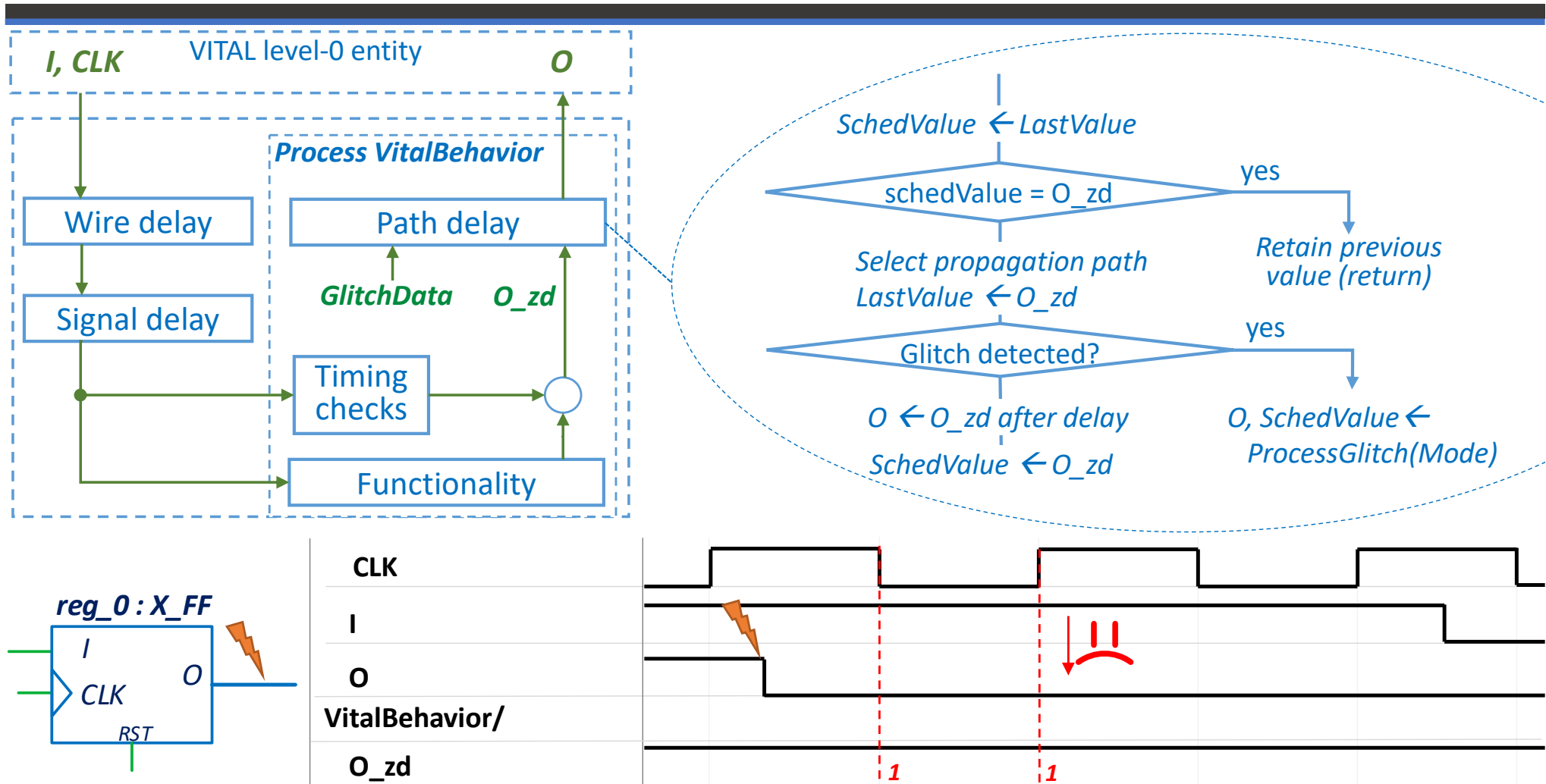
Portability
Interoperability
Annotation timing from Standard Delay
Format files (*SDF – IEEE-1497-2001*)

Simulation performance

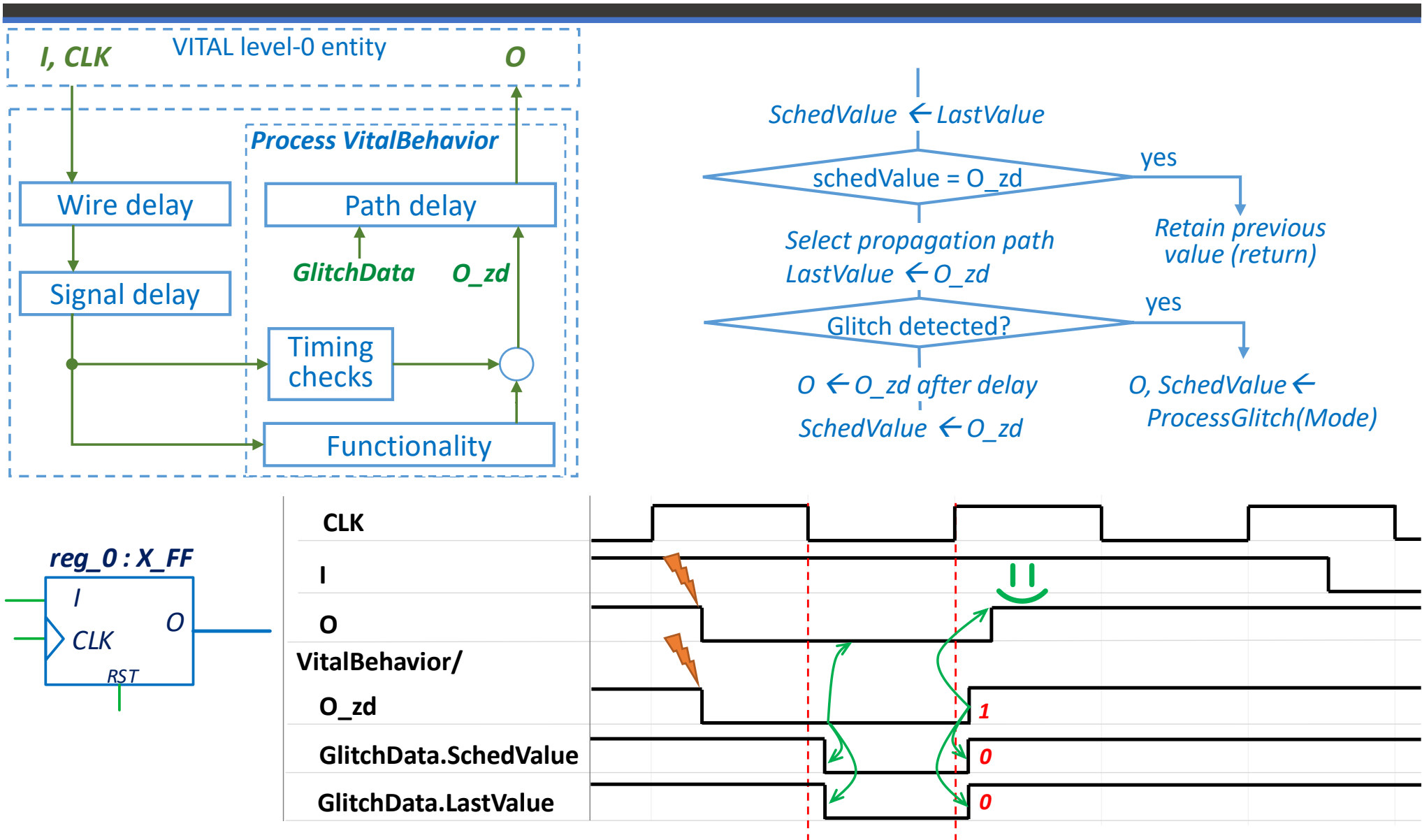
*VITAL compliant macro-cell model
(simulation primitive)*



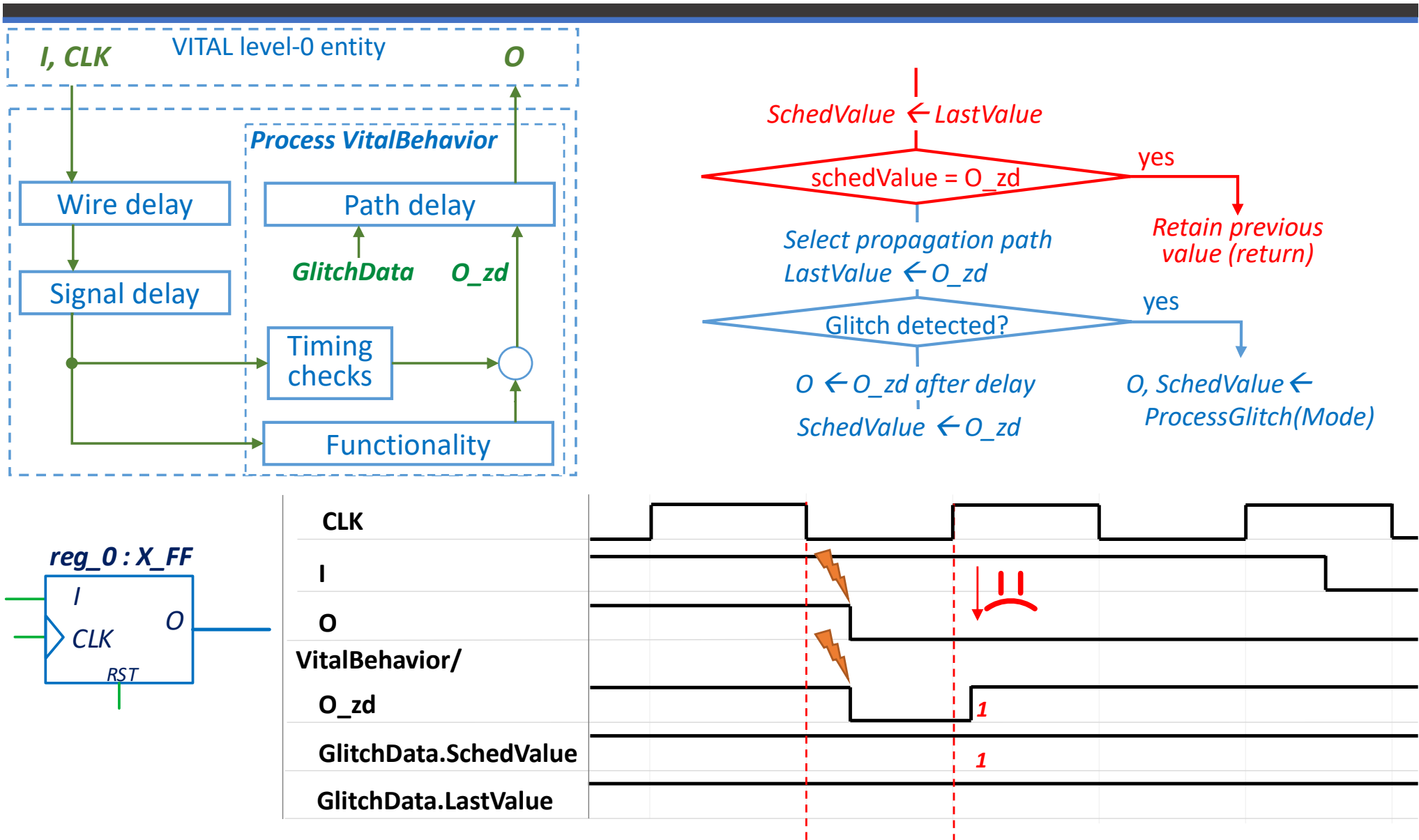
Bit-Flip at Implementation level



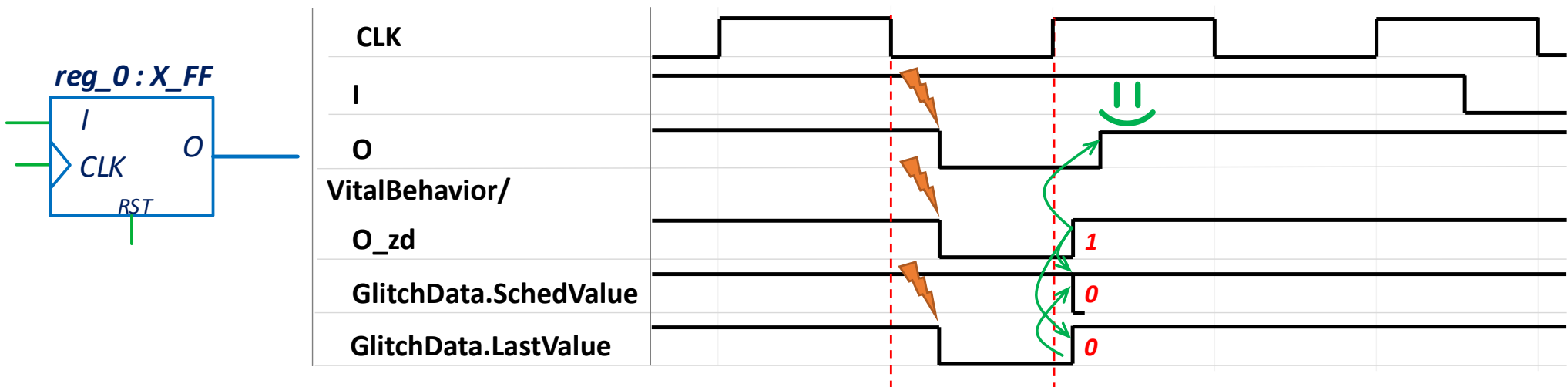
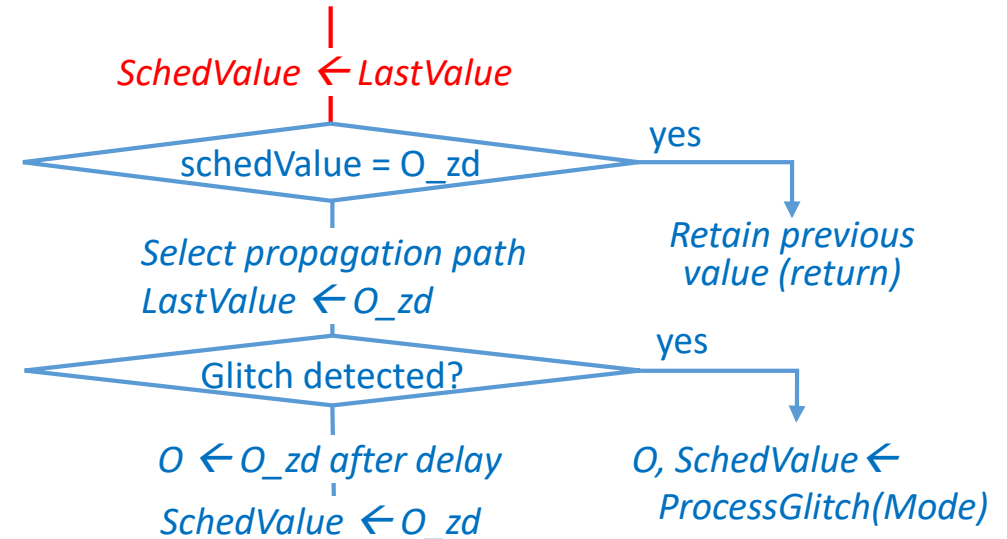
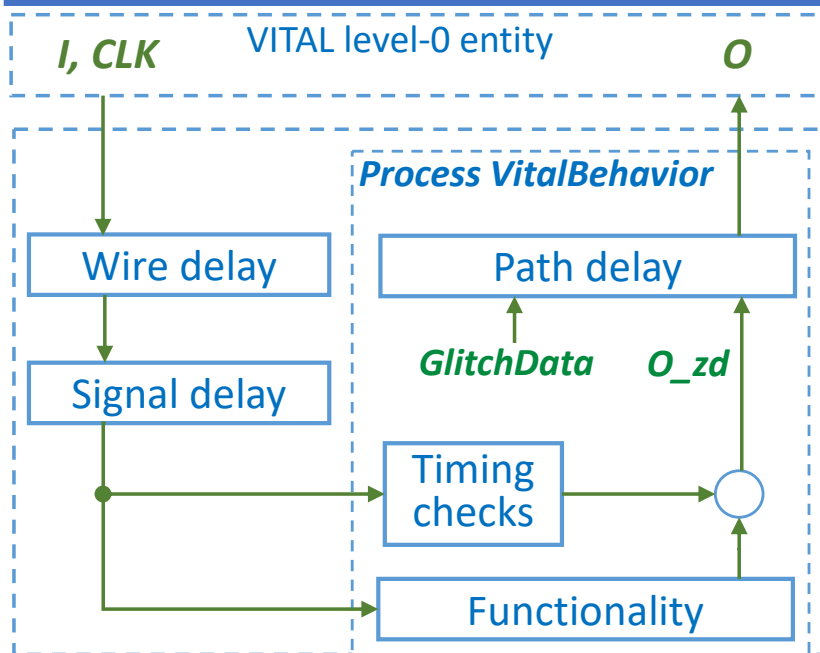
Bit-Flip at Implementation level



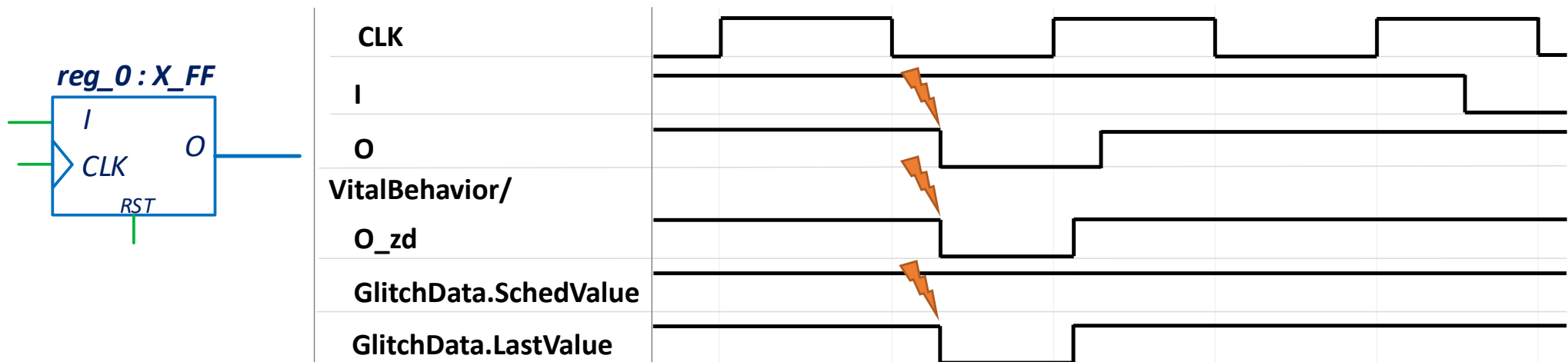
Bit-Flip at Implementation level

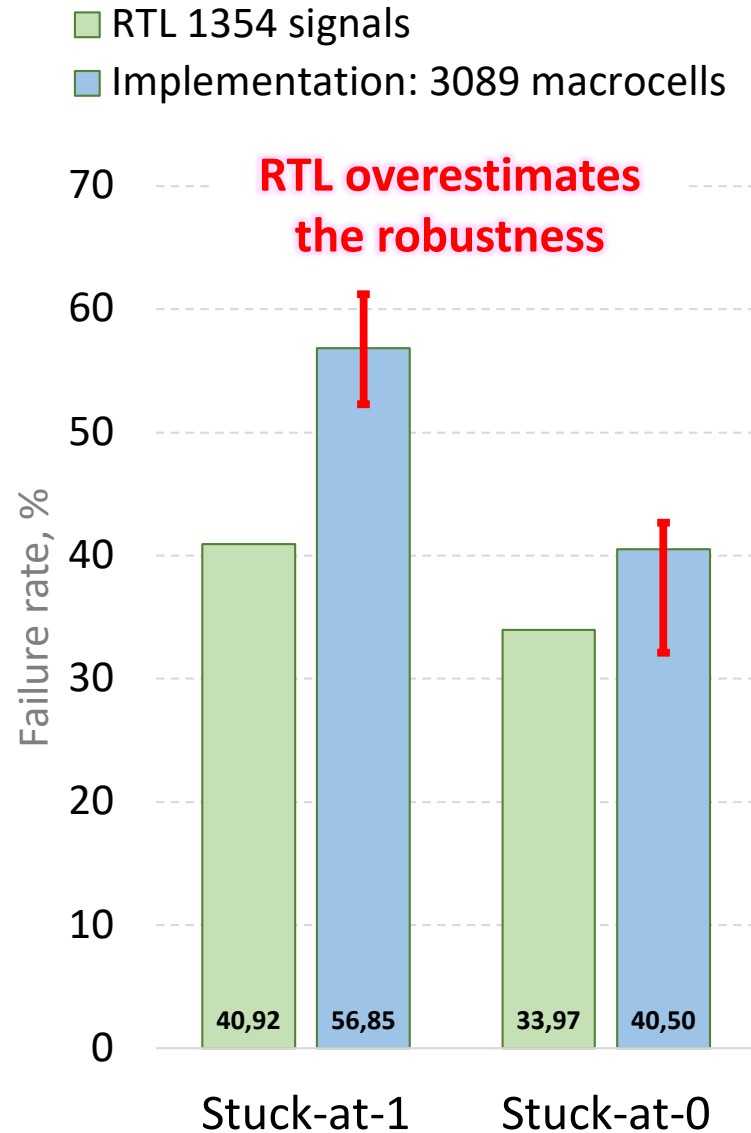


Bit-Flip at Implementation level



```
set CV [examine FF/O]
if {$cv == 1} { set nv 0 } else { set nv 1 }
force -deposit FF/O $nv 0
change FF/VITALBehavior/O_zd $nv
change FF/VITALBehavior/O_GlitchData.LastValue $nv
```





Limitations of robustness assessment at RTL

- RTL may significantly overestimate the robustness (bit-flip and stuck-at) → Increase accuracy by matching sequential logic between RTL and implementation
- Effects of faults in combinational logic (stuck-at and pulse) and delays (sequential and combinational logic) can only be accurately assessed at implementation level

New methodologies for injecting faults at implementation-level

- Use of standard simulator commands for bit-flip injection
- Generation of injectable VITAL-compliant libraries for delay injection
- Low-level of intrusion (neither models nor EDA tools must be modified)

Towards the automatic generation of injectable VITAL-compliant libraries

- Generic solution for any VHDL model, device manufacturer, EDA and SBFI tools
- Further research required for injection of other logic fault models (short, open, bridging ...) into device-specific components (LUT, DSP, RAM blocks ...)

- **Rapid (and automatic?) design** of safe and secure IP cores
- **Accurate evaluation**, including dependability assessment
- **Reduce time-to-market**: Use of EDA tools to optimize the implementation of HW designs
- **Fast selection of components**: Integration of *most suitable* IP cores (Benchmarking)



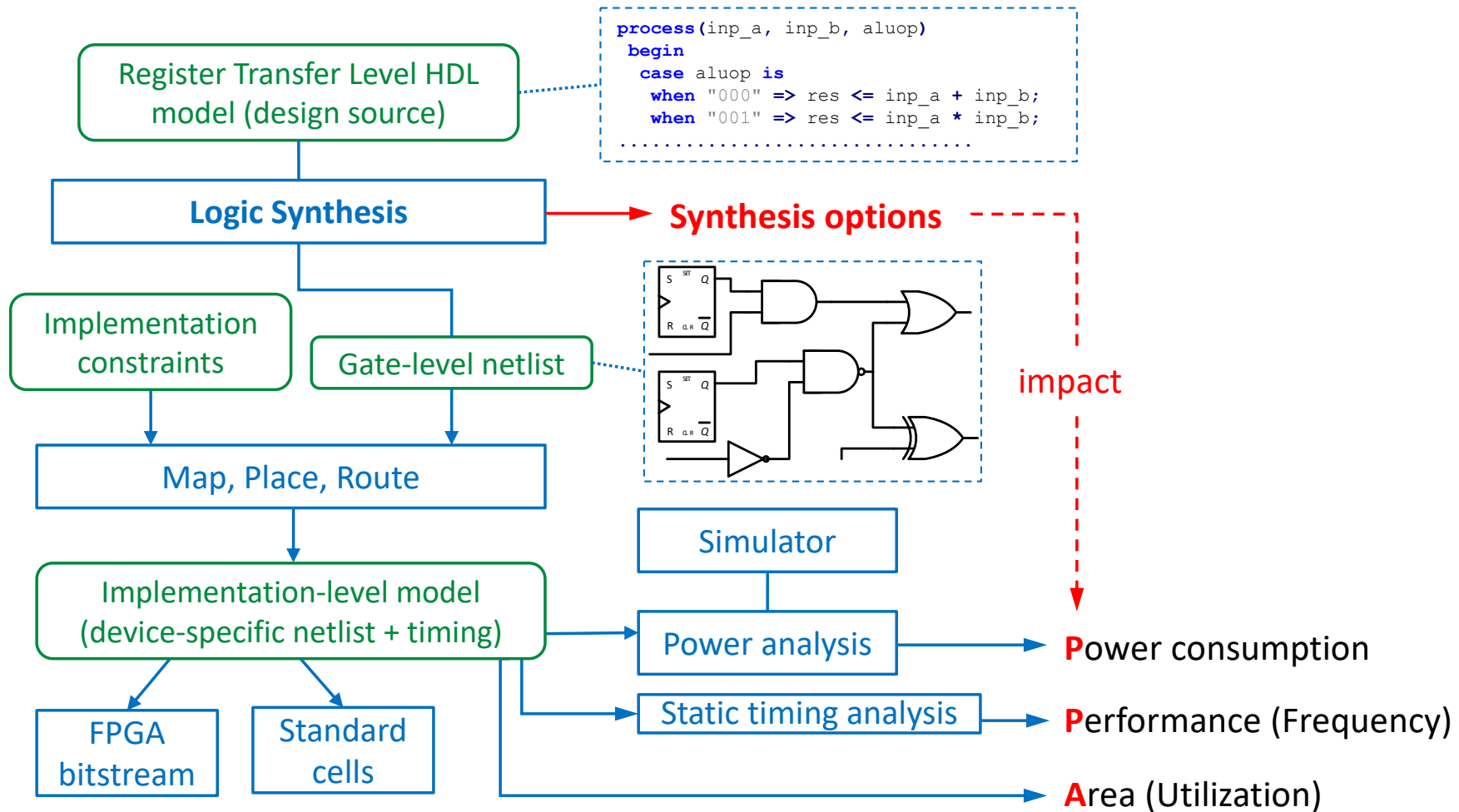
UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Dependability-aware Design Space Exploration for Optimal Synthesis Parameters Tuning



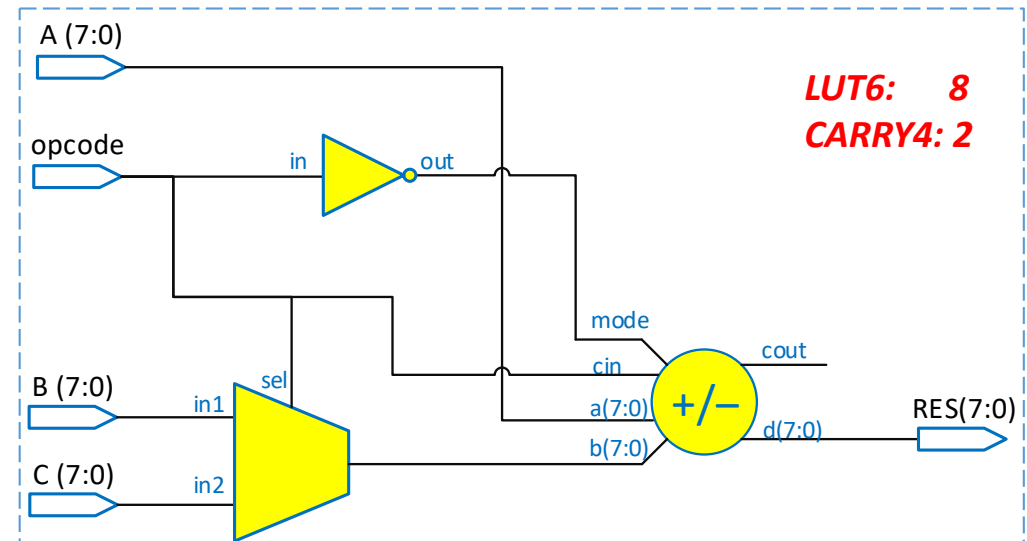
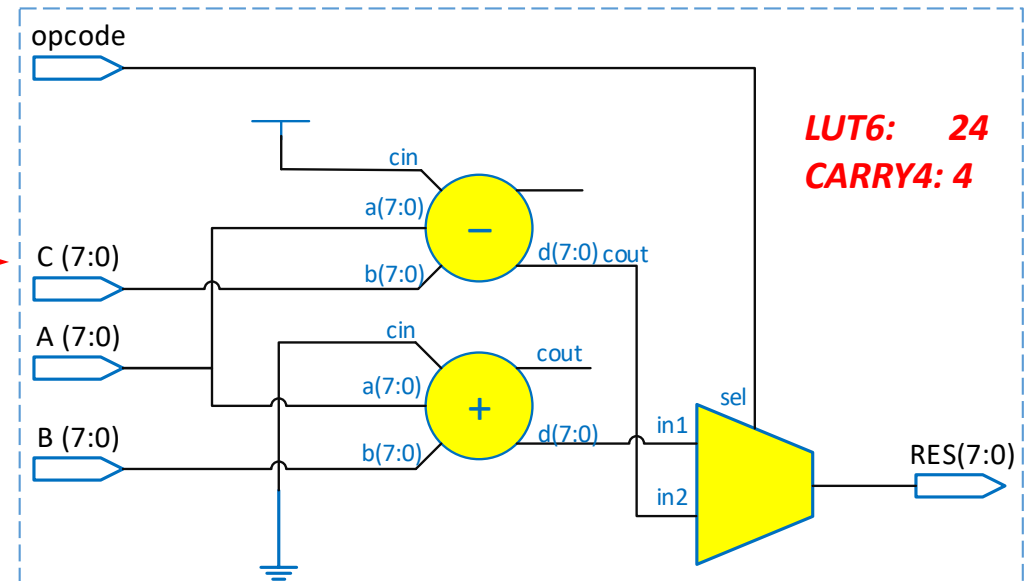
- Motivation
- Dependability-aware design space exploration for logic synthesis
- Case study
- Results
- Conclusions



Resource sharing → NO

```
entity addsub is
  port(
    A,B,C :in  std_logic_vector(7 downto 0);
    opcode :in  std_logic;
    RES :out std_logic_vector(7 downto 0));
end addsub;
architecture behav of addsub is
  begin
    RES <= A + B when opcode = '0'
           else A - C ;
  end behav;
```

Resource sharing → YES



Determining the optimal configuration of synthesis options:

- Huge design space:
 - K options (*factors*),
 - at least at two levels

→ 2^K configurations to explore

- High computational effort to evaluate each configuration:

- Performance, Power, Area (PPA):
 - Circuit implementation
 - Timing/Power analysis
- Dependability attributes (PPA + D)
 - **Fault injection experiments**

Experimentation time ~

- Model complexity
- × Number of Affected elements
- × Number Fault models
- × Number Injection time instances

Unaffordable DSE time

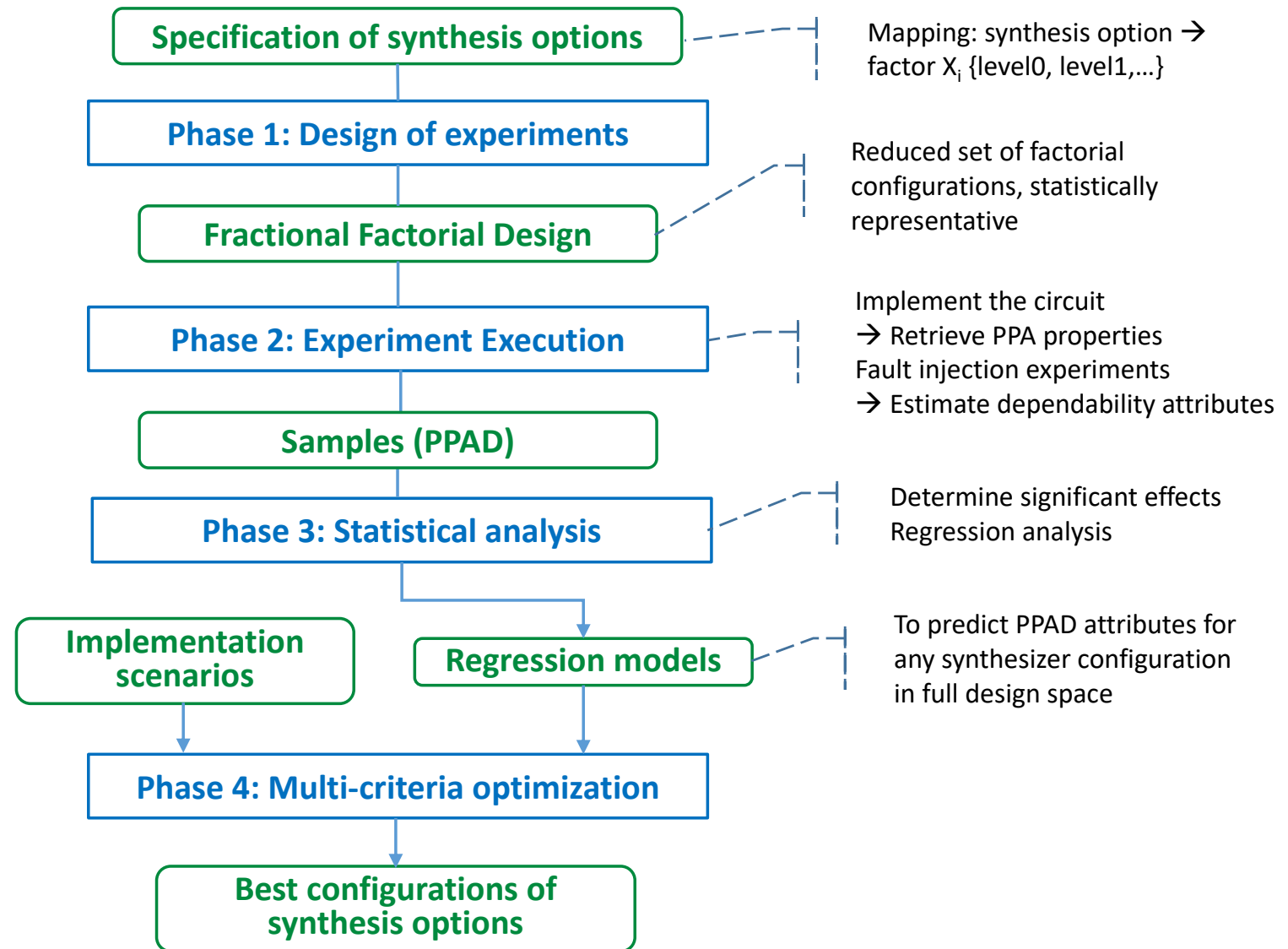
- $K = 31$ Xilinx synthesis options
- Full design space 2^{31} configurations
- 1 hour per each configuration

245 146 years

Hours/Days of simulation time
per each configuration (on Grid)!

- Motivation
- Dependability-aware design space exploration for logic synthesis
- Case study
- Results
- Conclusions

Proposed method – overview



- Motivation
- Dependability-aware design space exploration for logic synthesis
- Case study
- Results
- Conclusions

Case study: Response variables

Performance: maximum clock frequency

Dynamic power consumption: requires simulation of switching activity

Area: amount of used logic (flip-Flops and look-up-tables)

Mission Time: time in which reliability of the system $R(t)$ reaches a given threshold $R_T(t)$

$$R_T(t) = e^{-FIT \times K \times f \times t}$$

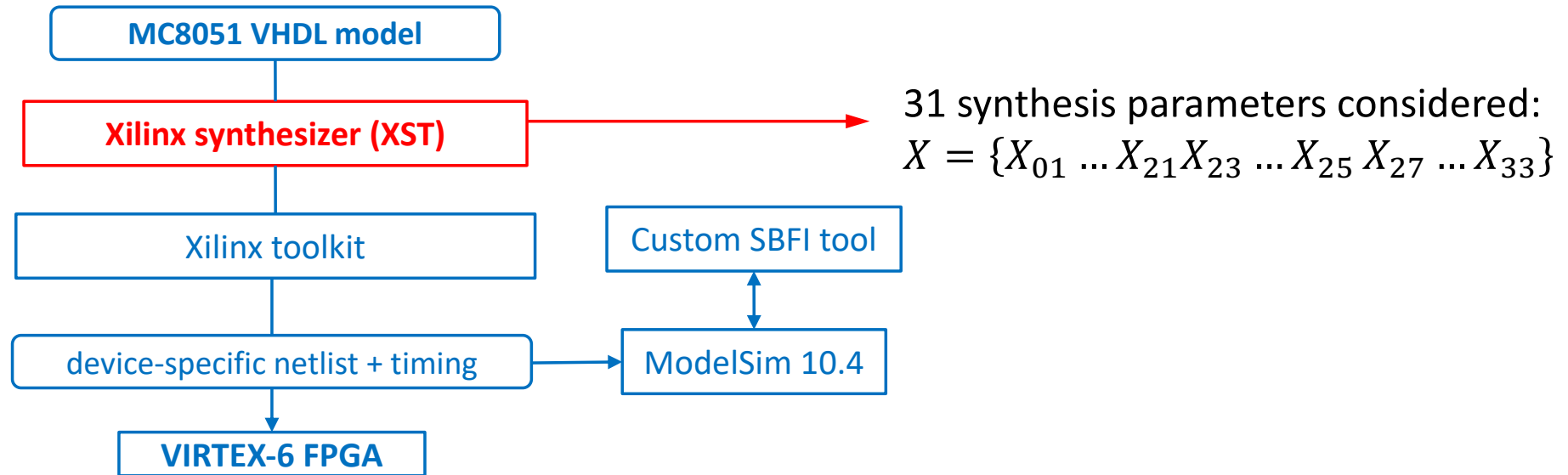
FIT – fault rate at sea-level (Xilinx device reliability report): 12×10^{-9} for Virtex6

K – factor scaling the fault rate according to the altitude (based on Rosetta test results):

- 1.0 at sea level,
- 21 for high-terrestrial altitudes,
- 327 for high-altitude aircraft;

f – percentage of faults leading to a failure – obtained from SBFi experiments

Case Study: Overview

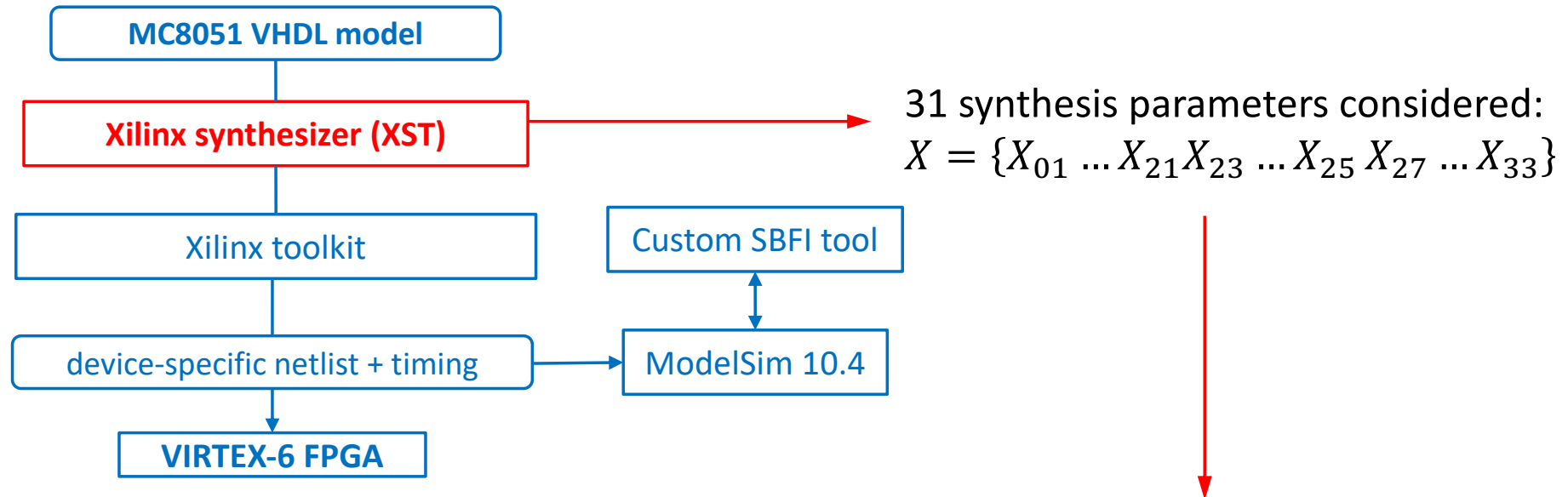


Case study: Synthesis options

Factor	XST option	Selected High/Low Levels
X ₀₁	Optimization Goal	Speed/Area
X ₀₂	Optimization Effort	Normal/High
X ₀₃	Power Reduction	No/Yes
X ₀₄	Keep Hierarchy	No/Yes
X ₀₅	Global Optimization Goal	ClockNets/Delay
X ₀₆	Cross Clock Analysis	No/Yes
X ₀₇	BRAM Utilization Ratio	0%/100%
X ₀₈	DSP Utilization Ratio	0%/100%
X ₀₉	Automatic FSM Extraction	No/Yes
X ₁₀	FSM Encoding Algorithm	Auto/Compact
X ₁₁	Safe Implementation	No/Yes
X ₁₂	FSM Style	LUT/BRAM
X ₁₃	RAM Extraction	No/Yes
X ₁₄	RAM Style	Auto/Distributed
X ₁₅	ROM Extraction	No/Yes
X ₁₆	ROM Style	Auto/Distributed

Factor	XST option	Selected High/Low Levels
X ₁₇	Automatic BRAM Packing	No/Yes
X ₁₈	Shift Register Extraction	No/Yes
X ₁₉	Shift Register Minimum Size	2/8
X ₂₀	Resource Sharing	No/Yes
X ₂₁	Use DSP Blocks	Auto/Automax
X ₂₂	Asynchronous to Synchronous	No/Yes
X ₂₃	Max Fanout	100/100000
X ₂₄	Register Duplication	No/Yes
X ₂₅	Equivalent Register Removal	No/Yes
X ₂₆	Register balancing	No/Yes
X ₂₇	Pack I/O Registers into IOBs	No/Auto
X ₂₈	LUT Combining	Auto/Area
X ₂₉	Reduce Control Sets	No/Auto
X ₃₀	Use Clock Enable	No/Auto
X ₃₁	Use Synchronous Set	No/Auto
X ₃₂	Use Synchronous Reset	No/Auto
X ₃₃	Optimize Primitives	No/Yes

Case Study: Overview



Targets per configuration [min : max]		Fault models	Experiments per target	Injection time
FFs LUTs	[539 : 624] [1972 : 3461]	Stuck-at-1	1	0
		Stuck-at-0	1	0
		Bit-Flip (FFs)	3	[0.05 : 0.15]
		Pulse (LUTs)		[0.30 : 0.40] [0.60 : 0.70]

- 2.45×10^6 fault injection experiments - executed on Grid (150 threads);
- 6 days - total experimentation time for all 128 configurations;
- 180 GB of logs.

- Motivation
- Dependability-aware design space exploration for logic synthesis
- Case study
- **Results**
- Conclusions

Statistically Significant Effects

ANOVA:

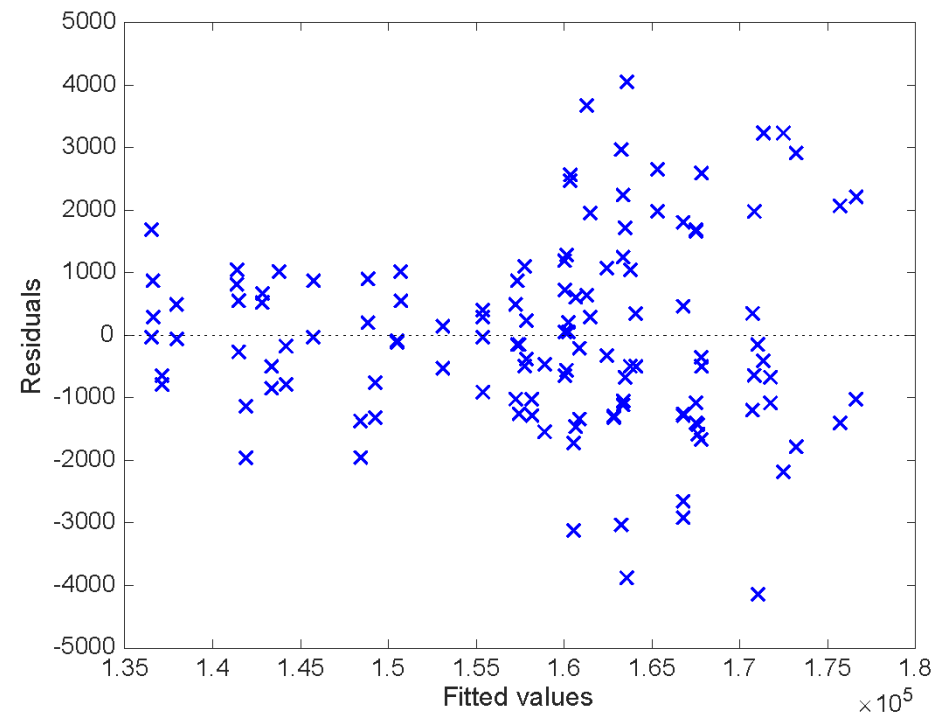
- 12 factors of 31 have statistically significant impact on response variables ($p\text{-value} < 0.05$)
- Main effects:

Factor	Synthesis option	Frequency (MHz)	Power (mW)	FF	LUT	Mission Time (h)
X_{01}	Optimization Goal	-15.95	-30.41	-9	-558	-13213
X_{02}	Optimization Effort	3.65	1.40	2	-72	13208
X_{04}	Keep Hierarchy	-0.04	-3.00	10	-102	2731
X_{05}	Global Optimization Goal	0.73	0.96	1	30	459
X_{08}	DSP Utilization Ratio	0.00	-0.70	0	-105	-4141
X_{19}	Shift Register Minimum Size	-0.54	-2.42	9	-9	-208
X_{21}	Use DSP Blocks	-0.05	-0.63	0	-41	348
X_{23}	Max Fanout	-0.19	-1.10	-3	-91	2892
X_{24}	Register Duplication	-0.54	-1.01	12	-2	432
X_{27}	Pack I/O Registers into IOBs	-2.05	-3.44	-18	-1	204
X_{28}	LUT Combining	-0.17	-0.21	0	-39	8
X_{30}	Use Clock Enable	0.81	-0.54	-2	-381	-3819
Intercept		59.29	121.73	577.93	3337.85	161290

MissionTime =

$$= (161.29 - 12.91 \cdot X_{01} + 5.47 \cdot X_{02} - 3.49 \cdot X_{08} + 2.2 \cdot X_{23} + 8.58 \cdot X_{01}X_{02} - 1.58 \cdot X_{01}X_{08} - 1.36 \cdot X_{01}X_{23} - 7.13 \cdot X_{01}X_{30} + 4.96 \cdot X_{02}X_{04} + 2.09 \cdot X_{02}X_{23} + 0.99 \cdot X_{04}X_{23}) \times 10^3$$

Coefficient of determination
 $R^2 = 0.98$



Resulting model

$$\begin{aligned} MTime = & (161.29 - 12.91 \cdot X_{01} + 5.47 \cdot X_{02} - 0.42 \cdot X_{04} - 3.49 \cdot X_{08} + 2.2 \cdot X_{23} + 0.22 \cdot X_{30} \\ & + 8.58 \cdot X_{01}X_{02} + 0.88 \cdot X_{01}X_{04} - 1.58 \cdot X_{01}X_{08} - 1.36 \cdot X_{01}X_{23} - 7.13 \cdot X_{01}X_{30} \\ & + 4.96 \cdot X_{02}X_{04} + 2.09 \cdot X_{02}X_{23} - 0.17 \cdot X_{01}X_{30} + 0.04 \cdot X_{04}X_{08} + 0.99 \cdot X_{04}X_{23} \\ & - 0.57 \cdot X_{04}X_{30} + 0.04 \cdot X_{08}X_{23} + 0.18 \cdot X_{08}X_{30} - 0.04 \cdot X_{23}X_{30}) \times 10^3 \text{ [Hours]} \end{aligned}$$

$$\begin{aligned} Frequency = & 59.29 - 15.93 \cdot X_{01} + 2.61 \cdot X_{02} - 3.34 \cdot X_{27} - 0.76 \cdot X_{01}X_{02} + 1.88 \cdot X_{01}X_{27} \\ & - 1.16 \cdot X_{01}X_{30} + 0.20 \cdot X_{02}X_{27} + 2.64 \cdot X_{02}X_{30} + 0.50 \cdot X_{27}X_{30} \text{ [MHz]} \end{aligned}$$

$$\begin{aligned} Power = & 121.73 - 40.55 \cdot X_{01} - 9.44 \cdot X_{04} - 2.32 \cdot X_{19} - 7.43 \cdot X_{27} + 12.15 \cdot X_{01}X_{04} \\ & + 2.91 \cdot X_{01}X_{19} + 5.23 \cdot X_{01}X_{28} - 2.55 \cdot X_{04}X_{19} + 3.30 \cdot X_{04}X_{27} - 0.55 \cdot X_{19}X_{27} \text{ [mW]} \end{aligned}$$

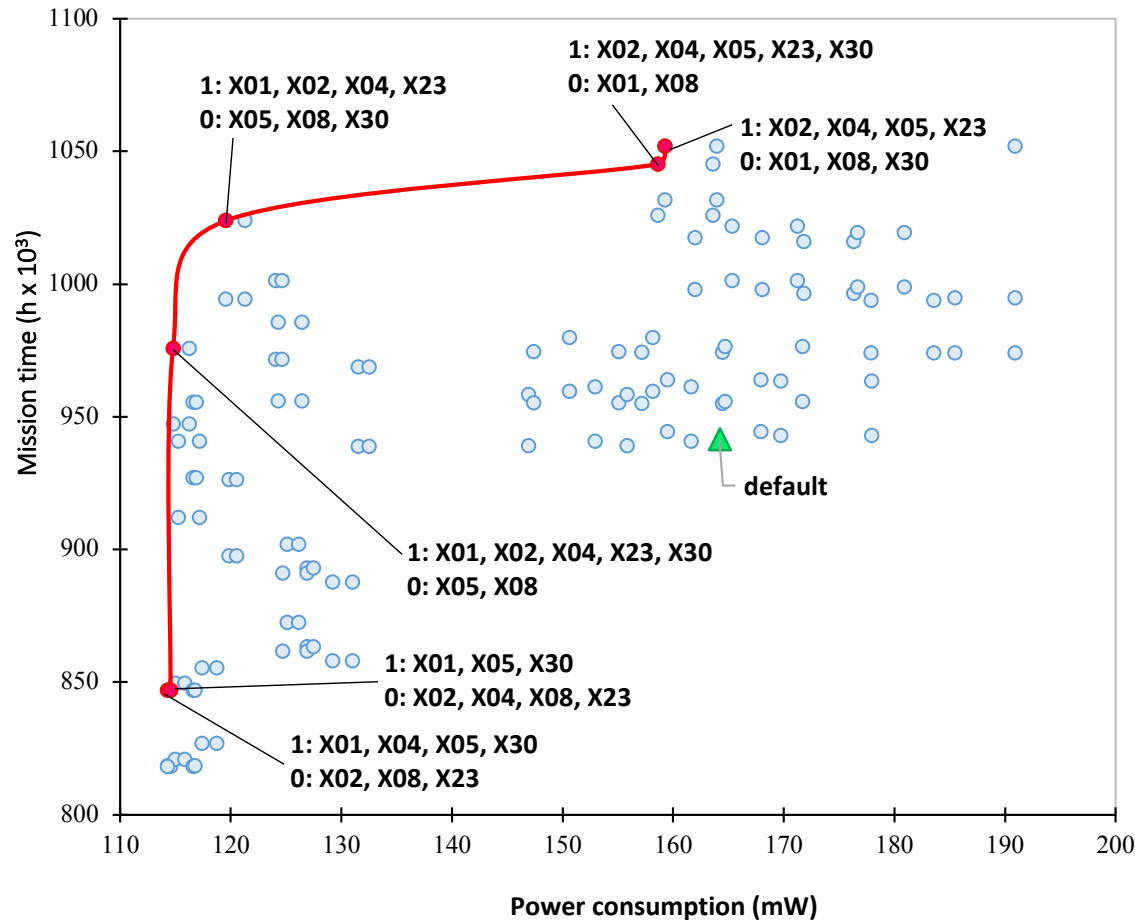
$$\begin{aligned} UtilFF = & \exp(6.35 + 0.02 \cdot X_{01} - 0.01 \cdot X_{04} - 0.01 \cdot X_{19} + 0.02 \cdot X_{24} - 0.06 \cdot X_{27} \\ & - 0.03 \cdot X_{01}X_{19} - 0.04 \cdot X_{01}X_{24} + 0.06 \cdot X_{04}X_{27} + 0.03 \cdot X_{19}X_{24}) \end{aligned}$$

$$\begin{aligned} UtilFF = & \exp(8.11 - 0.2 \cdot X_{01} - 0.04 \cdot X_{02} - 0.04 \cdot X_{04} + 0.02 \cdot X_{05} - 0.01 \cdot X_{08} - 0.03 \cdot X_{23} - 0.02 \cdot X_{28} \\ & - 0.10 \cdot X_{30} + 0.03 \cdot X_{01}X_{02} + 0.04 \cdot X_{01}X_{04} - 0.06 \cdot X_{01}X_{05} + 0.02 \cdot X_{01}X_{08} + 0.01 \cdot X_{01}X_{23} \\ & + 0.02 \cdot X_{01}X_{28} - 0.06 \cdot X_{01}X_{30} - 0.01 \cdot X_{02}X_{23} - 0.01 \cdot X_{04}X_{05} - 0.01 \cdot X_{04}X_{30} - 0.03 \cdot X_{08}X_{21}) \end{aligned}$$

- Regression models to predict responses (PPAD) for any configuration of synthesis options
- Determining best response and synthesizer configuration when optimizing each PPAD property separately

		Implementation Goals*				
		Mission time (h)	Frequency (MHz)	Power (mW)	FF	LUT
Simplex ($R_T=0.98$)	Best	1.77×10^5	64.37	78.97	542	1990
	Worst	1.37×10^5	41.06	121.73	616	3451
	Default	1.58×10^5	59.05	111.09	610	2901

* Best/Worst values correspond to different configurations optimizing each response variable separately



Pareto-optimal configurations:

→ 6 (MT vs Consumption)

→ 88 (all five variables)

Equally good

Which one should be selected?

→ Requirements

→ Subjective preferences

Implementation scenarios

Scenario	R _T	K	MT	F	P		FF	LUT
Consumer (Simplex)	0.98	1.0	C 1/6	M 3/6	W 0	S 1/6	1/6	
Cons. Mobile (Simplex)	0.98	1.0	C 1/8	S 2/8	M 3/8	S 1/8	1/8	
Industrial (TMR)	0.99	21.3	M 3/5	S 2/5	W 0	W 0	0	
Automotive (TMR)	0.99	21.3	M 3/10	M 3/10	S 2/10	S 1/10	1/10	
Aerospace (TMR)	0.999	237.8	M 3/6	S 2/6	C 1/6	W 0	0	

R_T – Reliability threshold, K – Fault rate scale factor, MT – Mission time,
F – Maximum clock frequency, P – Power consumption,
M – Must, S – Should, C – Could, W – Would

Context-aware optimization

Scenario		WSM Score	Mission Time (h)	Frequency (MHz)	Power (mW)	FF	LUT
Consumer (simplex)	Best	0.96	7.38×10 ⁶	64.37	112.28	571	2404
	Worst	0.74	-16%	-54%	-36%	+2%	+11%
	Default	0.81	-9%	-9%	-1%	+6%	+17%
Mobile (simplex)	Best	0.91	6.55×10 ⁶	45.76	78.97	544	2026
	Worst	0.77	+3%	+23%	+34%	+12%	+40%
	Default	0.8	+3%	+23%	+29%	+11%	+30%
Industrial (TMR)	Best	1.00	1.05×10 ⁶	60.19	163.96	1712	9033
	Worst	0.74	-28%	-45%	-43%	+3%	-32%
	Default	0.91	-11%	-8%	0%	-1%	-8%
Automotive (TMR)	Best	0.91	1.04×10 ⁶	60.19	163.57	1712	7875
	Worst	0.80	-22%	-45%	-25%	+3%	+2%
	Default	0.86	-11%	-8%	0%	-1%	+6%
Aerospace (TMR)	Best	0.95	1.96×10 ⁴	60.19	163.96	1712	9033
	Worst	0.78	-28%	-45%	-40%	+1%	-31%
	Default	0.87	-11%	-8%	0%	-1%	-8%

cells in green: less than 5% difference from absolute best

- Motivation
- Dependability-aware design space exploration for logic synthesis
- Case study
- Results
- Conclusions

Proposed approach:

- Quantifies the impact of synthesis options on PPAD properties,
- Simultaneously optimizes PPA and dependability-related properties,
- Handles huge design space within affordable experimentation time,
- Not requiring any modification of HDL design or EDA tools,
- Add-on to all existing approaches and fault tolerant design techniques

Proper tuning of synthesis options improves implementation result with respect to misconfigured tool up to:

+ 30% mission time, + 56% performance, – 42% power consumption; – 35% area

Default configuration of XST tool:

- Tailored to maximize clock frequency
- Close to worst power consumption and area
- 12% less mission time than the best one

- **Rapid (and automatic?) design** of safe and secure IP cores
- **Accurate evaluation**, including dependability assessment
- **Reduce time-to-market**: Use of EDA tools to optimize the implementation of HW designs
- **Fast selection of components**: Integration of *most suitable* IP cores (Benchmarking)



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



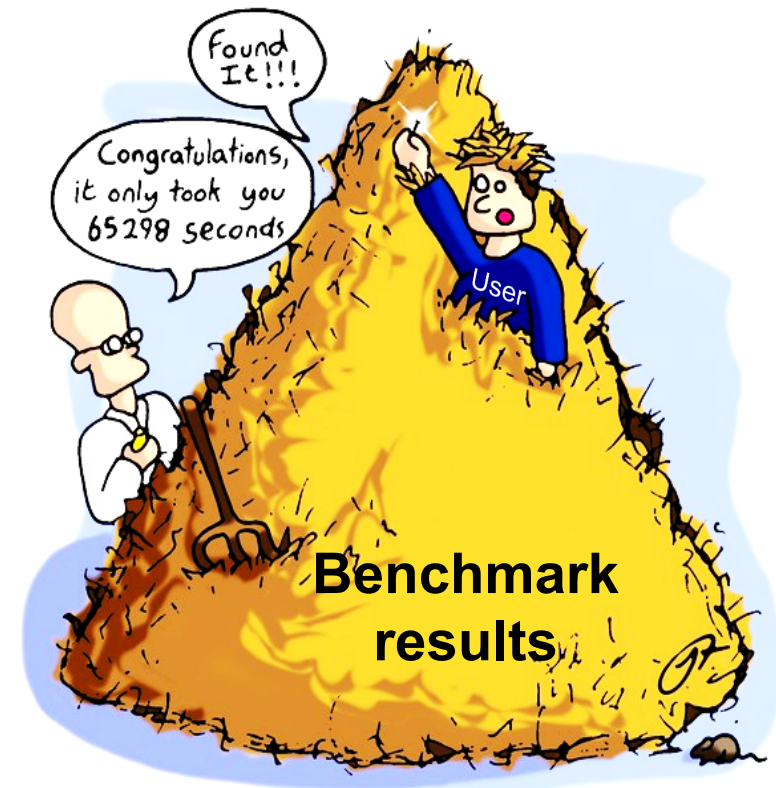
Fast selection of available alternatives through benchmarking



- A challenge when considering a variety of targets, faults/targets, types of measures, failure modes and contexts of use



- Error-prone interpretation
- Limited interest for non-skilled benchmark users

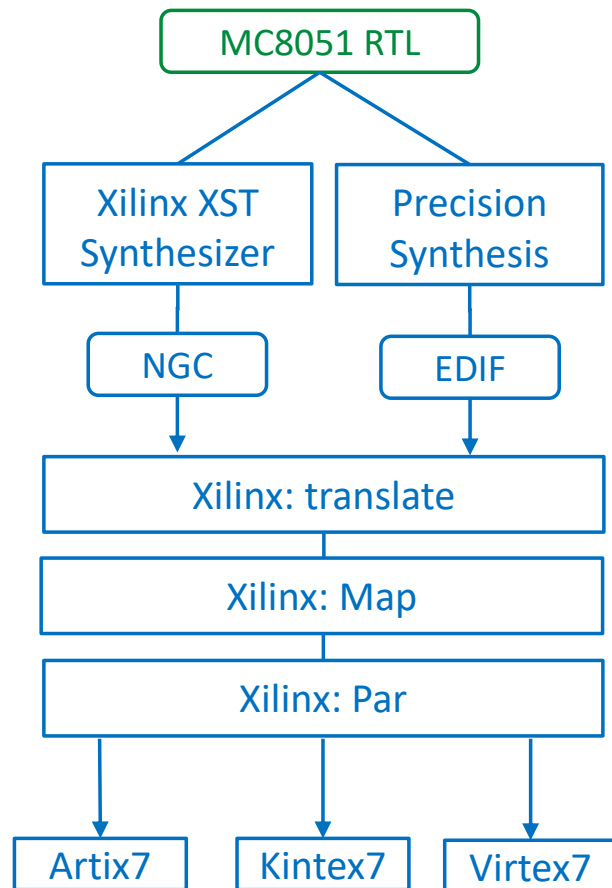


Multi-criteria decision making

- Hierarchical aggregation and analysis of measures
- Multiple alternatives: LSP, AHP, WSM, etc.
- MCDM paradox
- Typically Weighted Sum Model (WSM) is the most simple to interpret, and most easy to accept, analysis method

$$A_i^{WSM-score} = \sum_{j=1}^n w_j a_{ij}$$
$$A_*^{WSM-score} = \max_i \sum_{j=1}^n w_j a_{ij}$$

- Making results public is not enough, the analysis method must be also made explicit → **Reproducibility of conclusions**



6 implementation alternatives

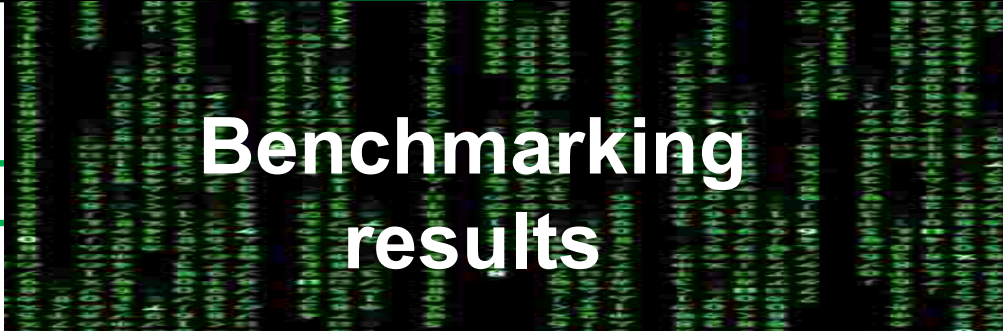
- XST + Artix7 / XST + Kintex7 / XST + Virtex7 / Precision + Artix7 / Precision + Kintex7 / Precision + Virtex7

5 alternative scenarios

Scenario	R _T	K	MT	F	P	FF	LUT
Consumer (Simplex)	0.98	1.0	C 1/6	M 3/6	W 0	S 1/6	1/6
Cons. Mobile (Simplex)	0.98	1.0	C 1/8	S 2/8	M 3/8	S 1/8	1/8
Industrial (TMR)	0.99	21.3	M 3/5	S 2/5	W 0	W 0	0
Automotive (TMR)	0.99	21.3	M 3/10	M 3/10	S 2/10	S 1/10	1/10
Aerospace (TMR)	0.999	237.8	M 3/6	S 2/6	C 1/6	W 0	0

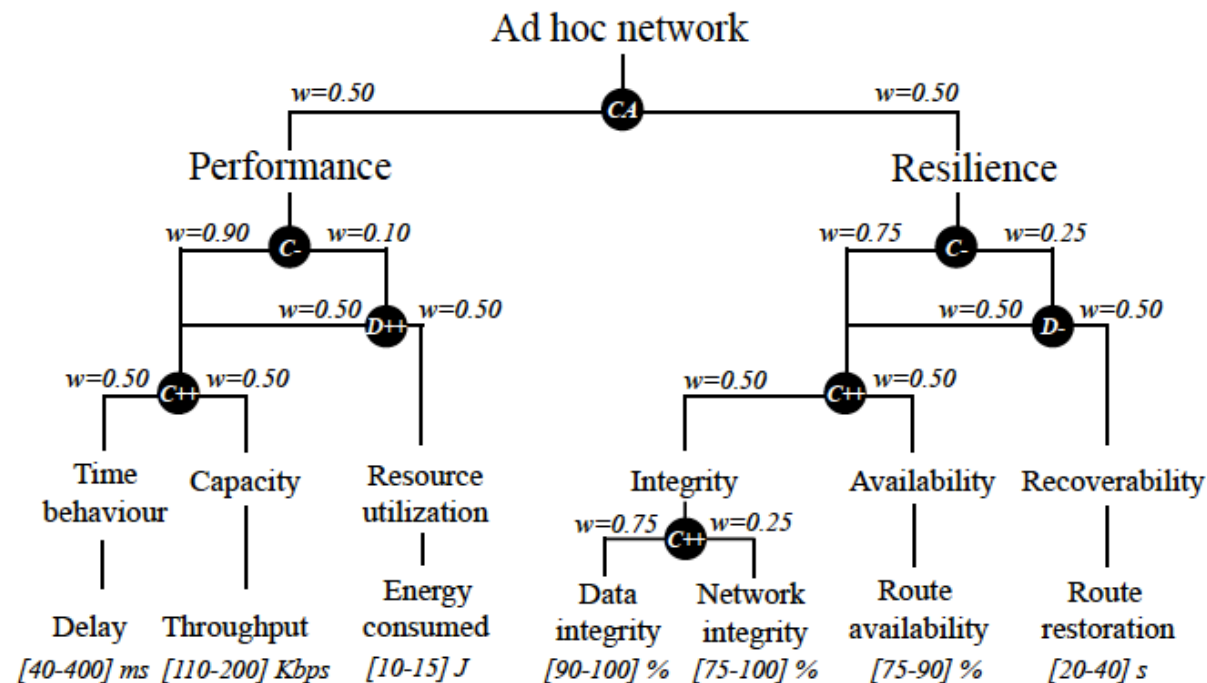
R_T – Reliability threshold, K – Fault rate scale factor, MT – Mission time, F – Maximum clock frequency, P – Power consumption, M – Must, S – Should, C – Could, W – Would

Fault load: Stuck-at, bit-flips and delays

Alternative	Implementation properties				
	Cost (min)	Frequency (max)	Power (min)	SDC (min)	MTTF_K21 (max)
XST_Artix7					
XST_Kintex7					
XST_Virtex7					
Precision_Artix7					
Precision_Kintex7					
Precision_Virtex7					

Alternative	Implementation properties					Score	
	Cost (min)	Frequency (max)	Power (min)	SDC (min)	MTTF_K21 (max)	Automotive	Mobile
XST_Artix7	1.71	55.81	46.01	19.15	7.04	0.78	0.75
XST_Kintex7	2.19	83.59	77.71	18.45	6.93	0.89	0.67
XST_Virtex7	16.57	80.29	74.90	18.42	6.93	0.86	0.63
Precision_Artix7	1.00	50.14	34.79	21.60	9.44	0.80	0.88
Precision_Kintex7	1.19	71.93	49.54	21.27	9.06	0.86	0.78
Precision_Virtex7	10.54	71.45	51.15	21.15	9.11	0.85	0.69

- Example of HW routing components implementing ad hoc routing protocols
- They are used for the deployment of ad hoc networks



Conclusions

- Tiny HW devices are at the core of the CPS revolution
- Separation of concerns provides means to
 - Modularize the design of IP cores
 - Automate the deployment of non-functional mechanisms
- Dependability and security cannot be considered in isolation during verification
 - Accurate assessment is fundamental
 - Approaches should seamlessly integrate into existing semicustom design flows supported by EDA tools
 - Selection of components must be objective and justified
- Beyond the mere electronic design and verification of IP cores there is a myriad of other problems to consider



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Design and Verification of Safe and Secure VLSI Systems*

Juan-Carlos Ruiz-García

jcruizg@disca.upv.es

* Research partially funded by the Ministerio de Economía, Industria y Competitividad de España under grant agreement number TIN2016-81075-R

