



FACULTEIT INGENIEURSWETENSCHAPPEN

Dynamische circuitspecialisatie als stap naar efficiënter hardware-ontwerp van de toekomst


Prof. Dirk Stroobandt, Universiteit Gent, België
Hardware and Embedded Systems group

FPGAs prevalent in Embedded Systems

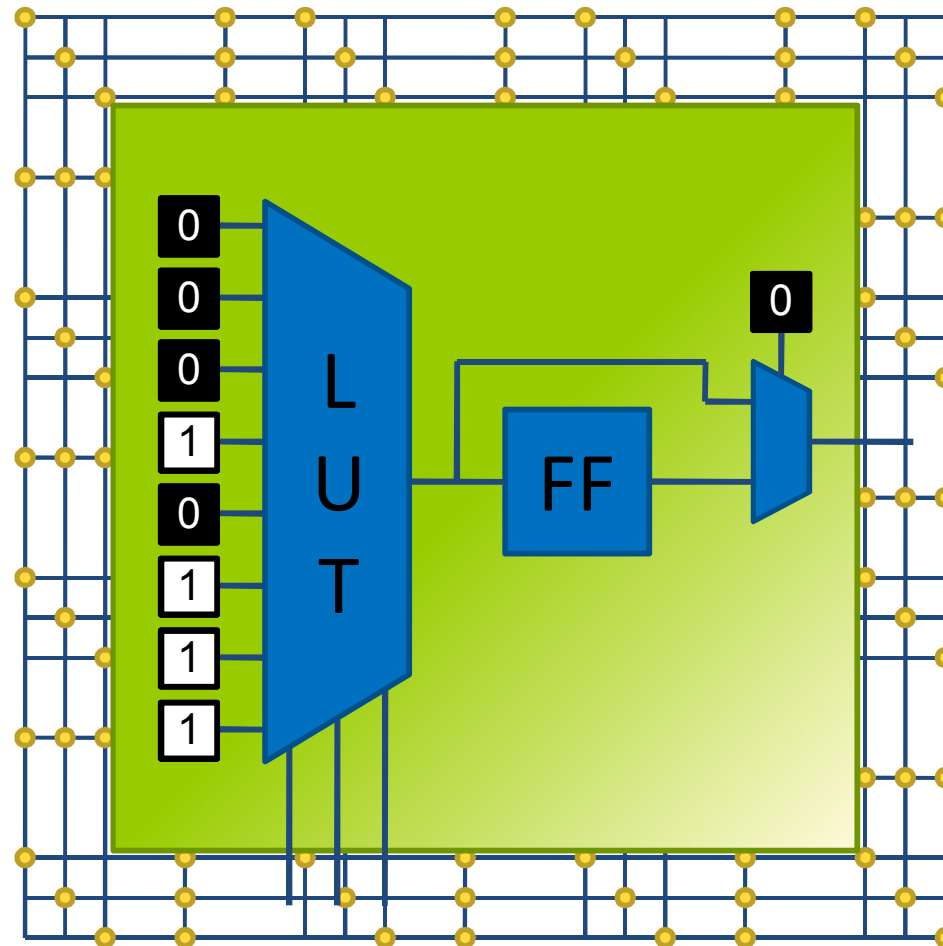
- Embedded Systems often need hardware acceleration
- Hardware needs to be targeted to specific system but flexible enough for updates/improvements
 - ASICs are very power efficient but no flexibility
 - FPGAs cost-effective for relatively low volumes
 - Additional benefit: reconfigurability

Current trends in High Performance Comp.

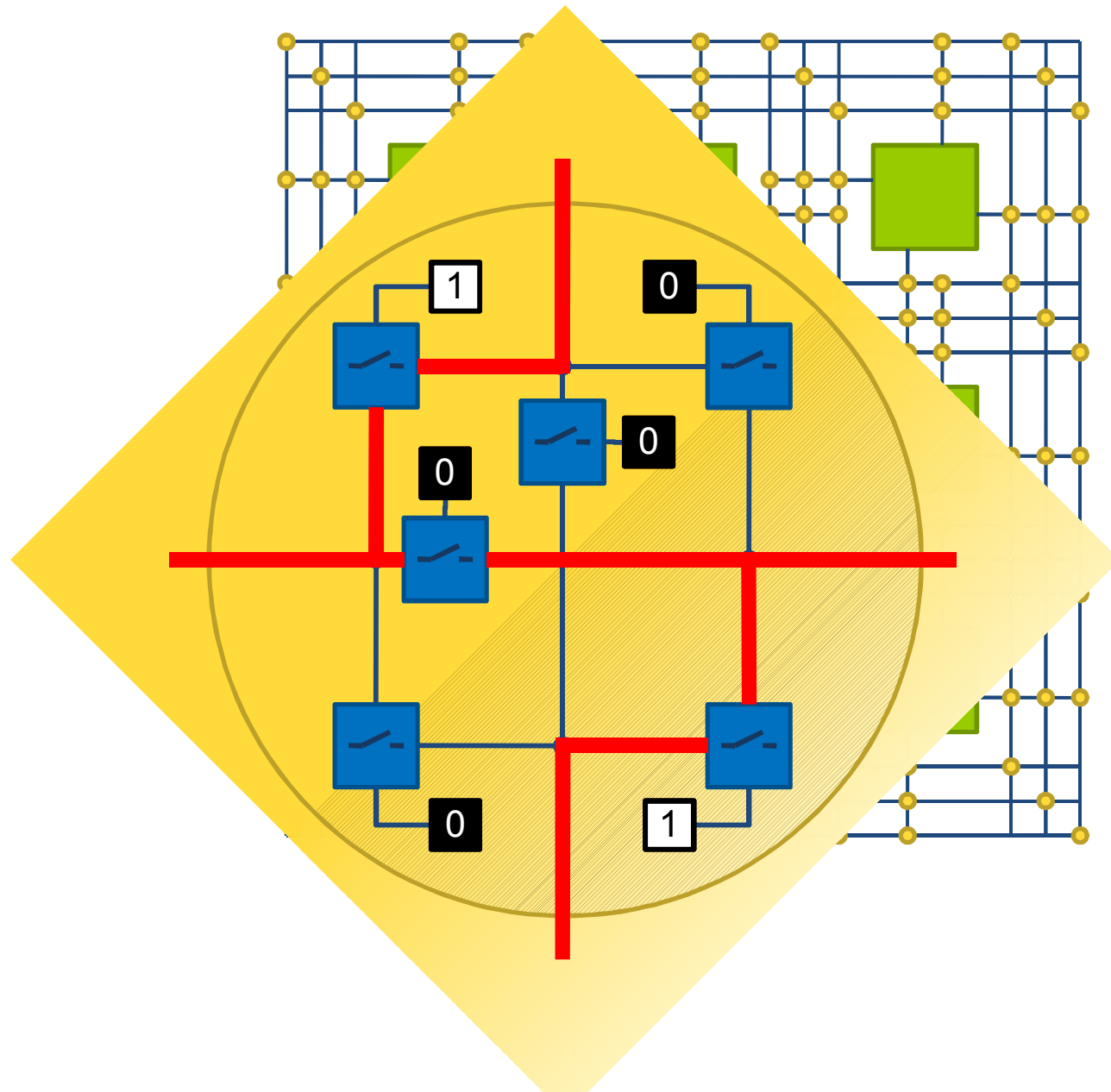
- More and more heterogeneous compute nodes
 - CPU, GPU, ASIP, also FPGA
 - Hardware acceleration often required
- Hardware has to be flexible
 - ASICs are very power efficient but no flexibility
 - Current datacenters: require configurability


- In future systems
 - Power requirements will demand dynamic circuit specialization, i.e., optimizations within tasks
 - Run-time reconfiguration will be main driver

FPGA configurability



FPGA configurability



Outline

- What is **Parameterized Run-time Reconfiguration**?
- **Profiling** your applications for parameters
- How to put this in **practice**?

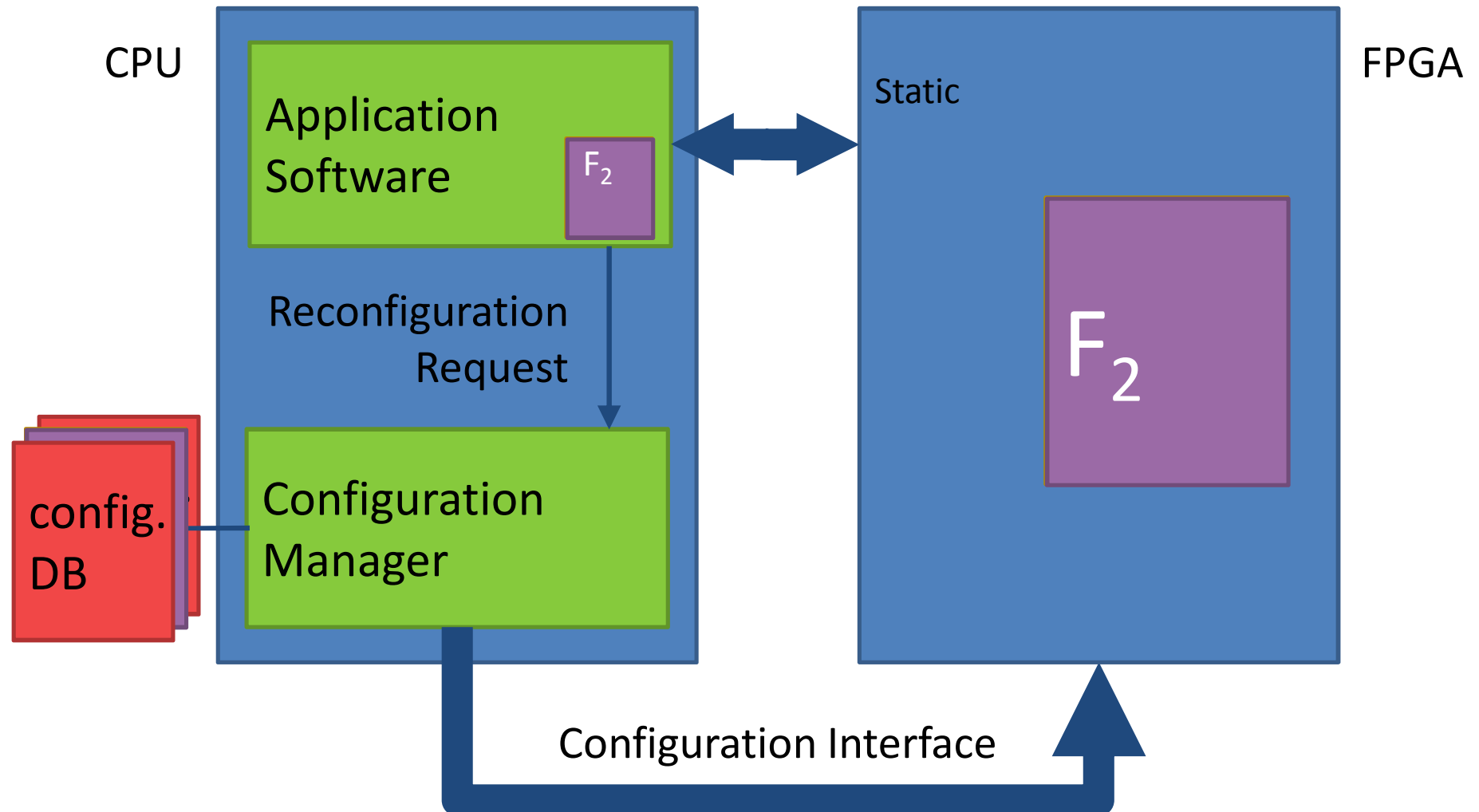
Outline

- What is Parameterized Run-time Reconfiguration?
- Profiling your applications for parameters
- How to put this in practice?

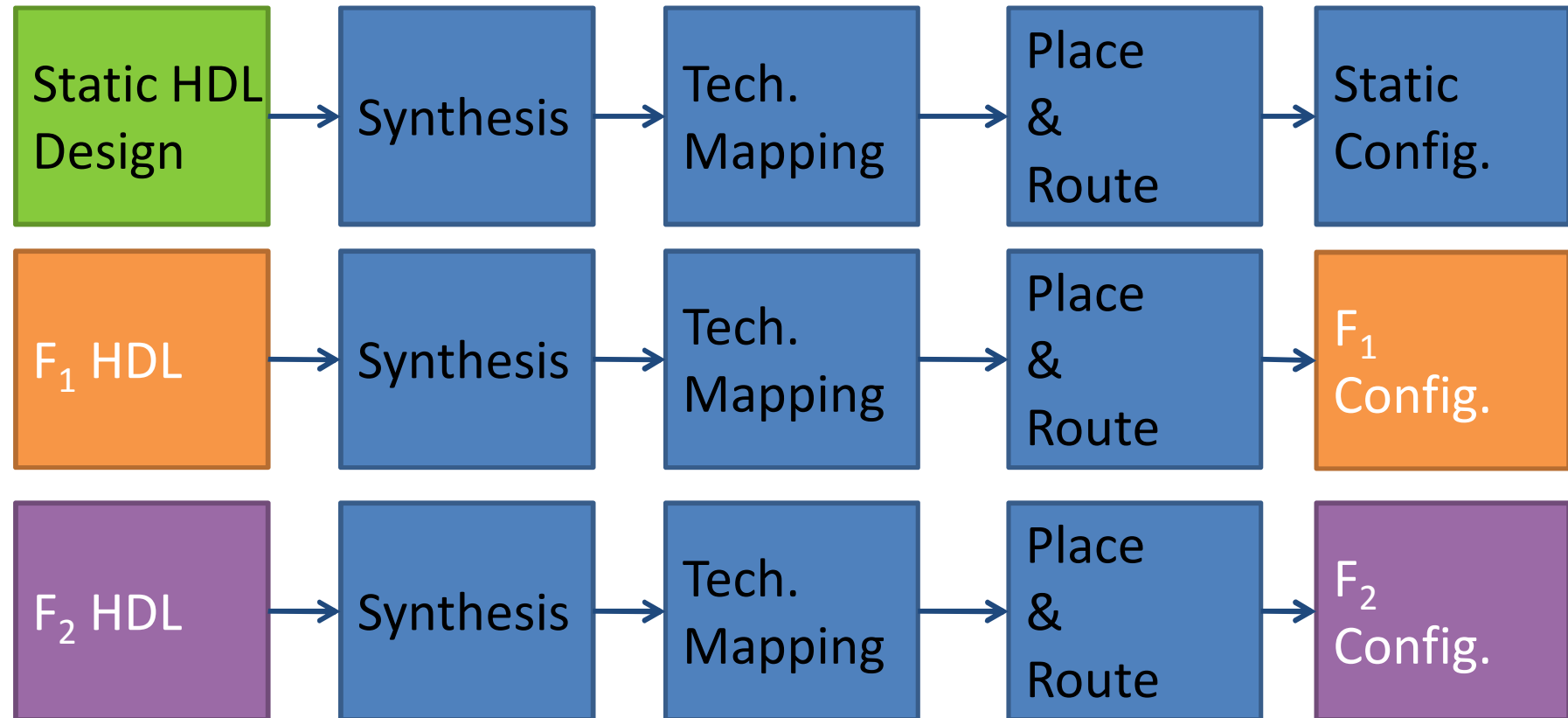
Run-Time Reconfiguration?

- Yesterday: configurability on a **large time scale**
 - Prototyping
 - System update
 - ...
- Today: configurability on a **smaller time scale**
 - Dynamic circuit specialization
 - Frequently changing (regular) inputs vs. infrequently changing **parameters**
 - Parameters trigger a reconfiguration (through **configuration manager**)
 - Goals:
 - Improve performance
 - Reduce area
 - Minimize design effort

Conventional Dynamic Reconfiguration



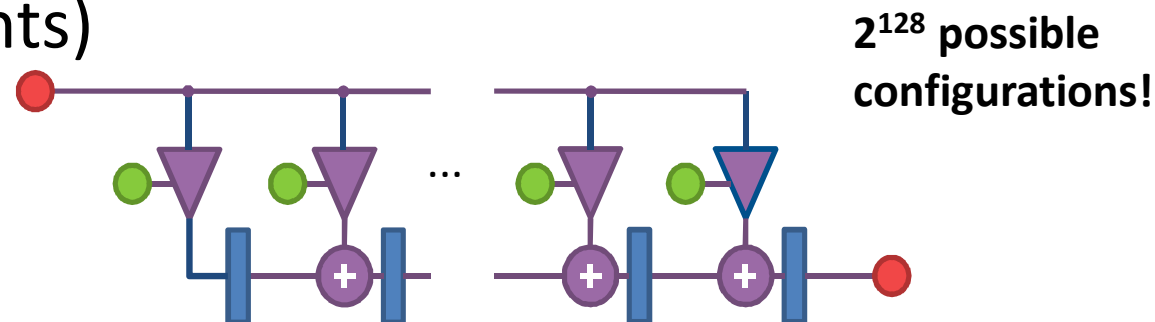
Conventional Tool Flow



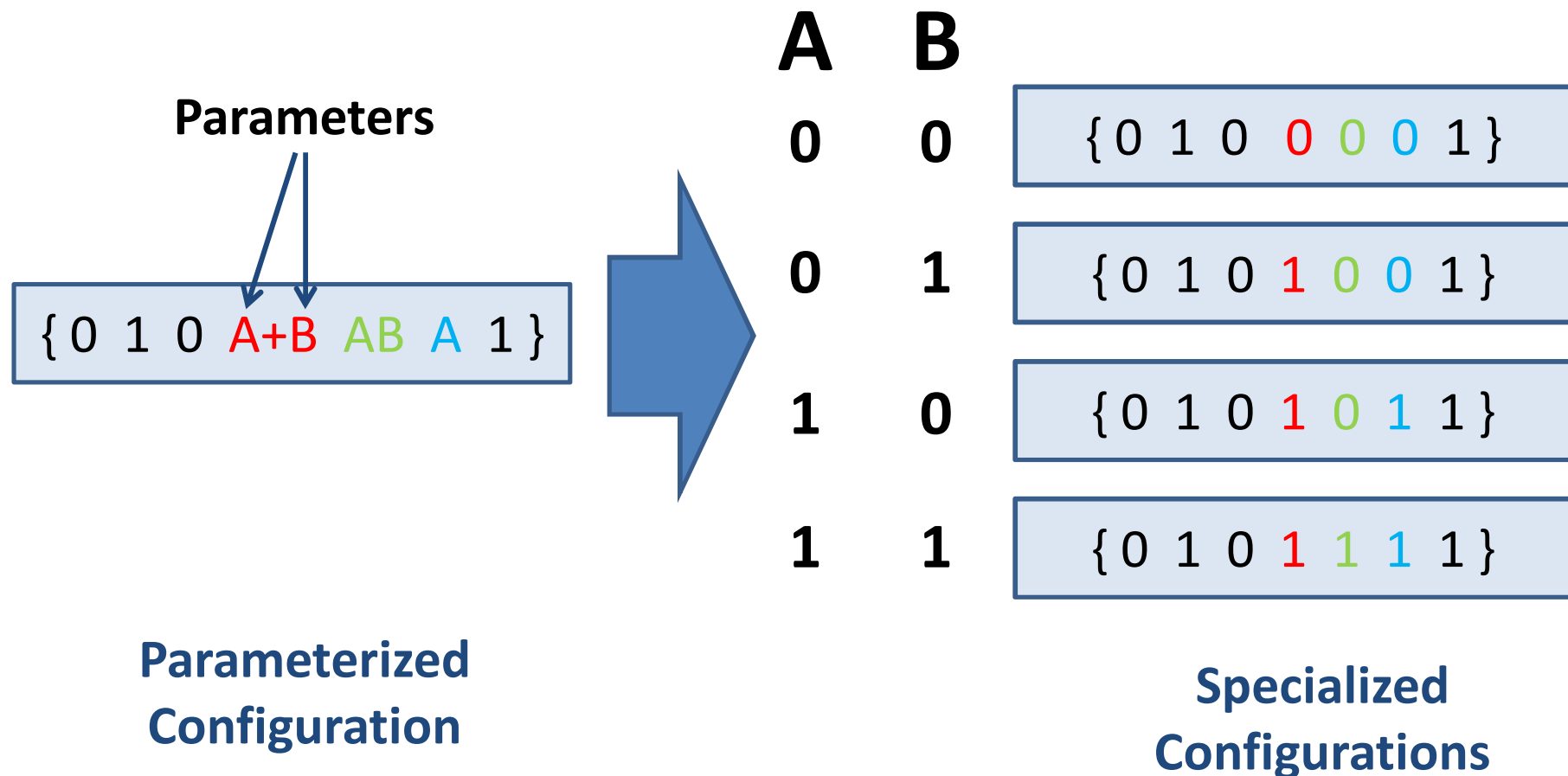
Dynamic Circuit

Specialization not feasible!

- Application where part of the input data changes infrequently
 - Conventional implementation (no reconfiguration):
Generic circuit, Store data in memory, Overwrite memory
 - Dynamic circuit specialization:
Reconfigure with configuration specialized for the data
- Example: Adaptive FIR filter (16-tap, 8-bit coefficients)

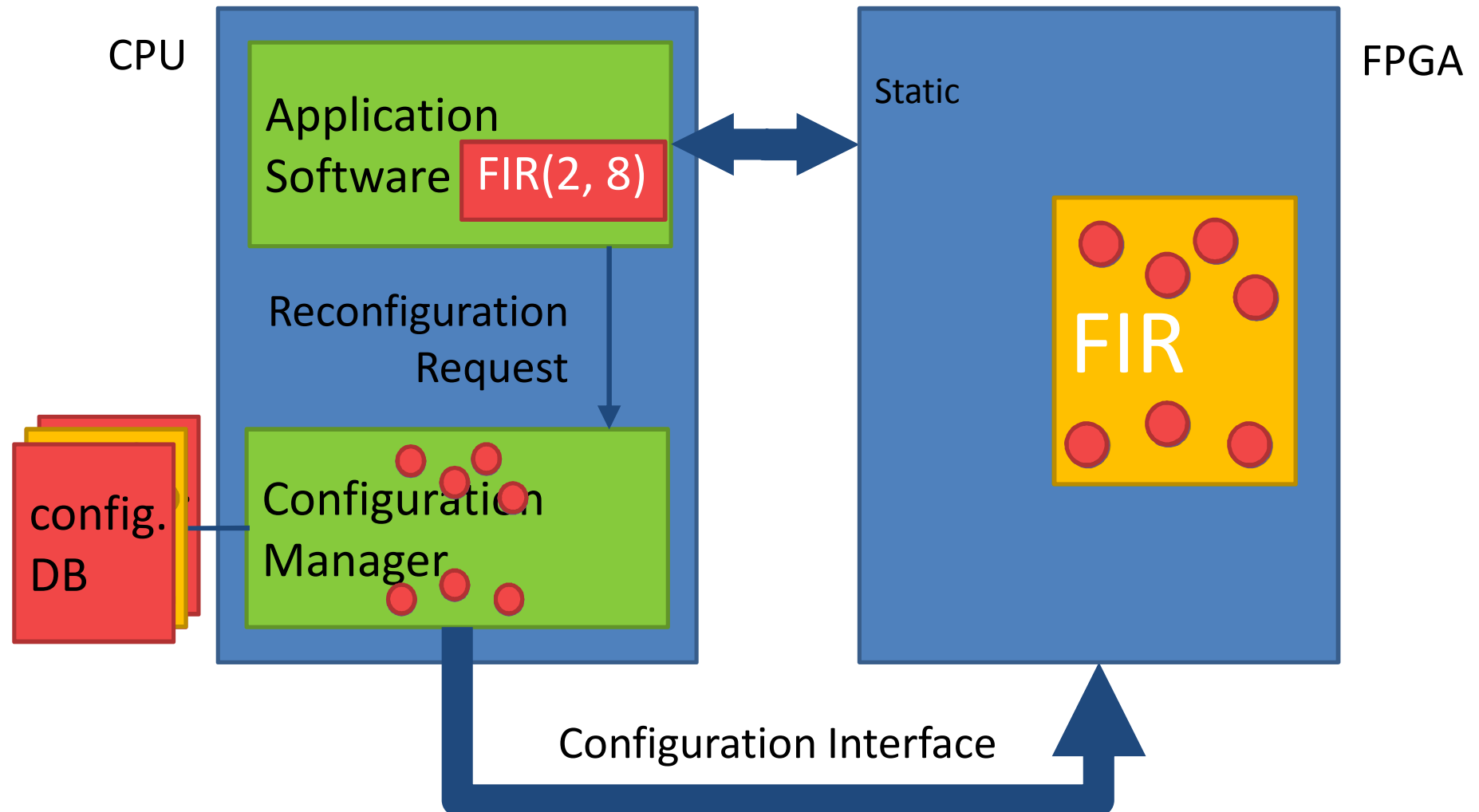


Our solution: Parameterized Configuration



* K. Bruneel and D. Stroobandt, "Automatic Generation of Run-time Parameterizable Configurations," FPL 2008.

Dynamic Circuit Specialization (micro-reconfiguration)



Two stage approach

- Off-line stage:
 - In: **Generic functionality**
 - Specification of the generic functionality
 - Distinction regular and parameter inputs
 - Out: **Parameterizable Configuration**
 - Software function
 - outputs specialized configurations for given parameter values
- On-line stage:
 - In: **Parameter values**
 - Evaluate parameterizable configuration
 - Out: **Specialized Configuration**
 - **Repeat** every time parameters change

Generic
Functionality

Off-line Stage

Parameterizable
Configuration

On-line Stage

Specialized
Configuration

Param. Configuration Tool Flow

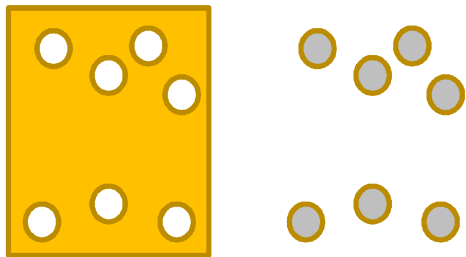
Param. HDL

Synthesis*

Tech. Mapping*

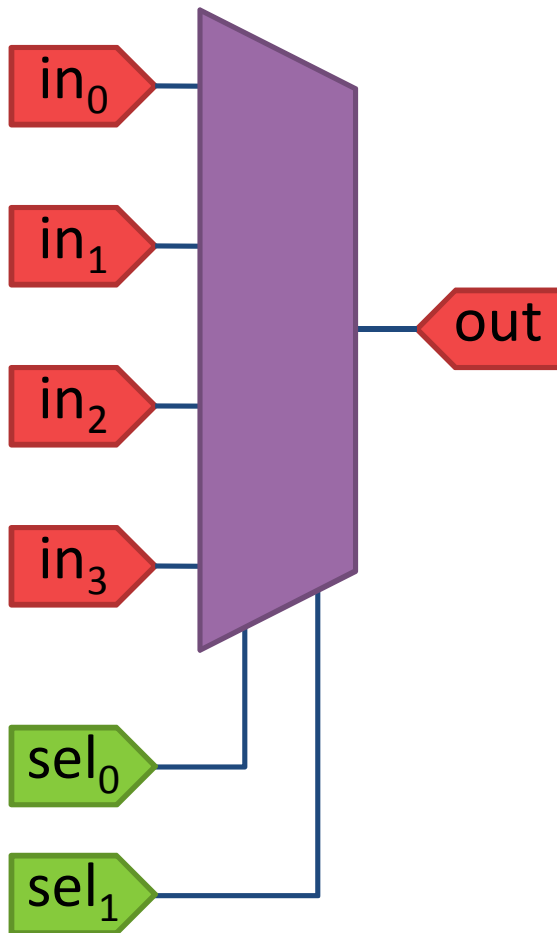
Place* & Route*

Param. Config.



- **Tunable truth table bits**
 - Adapted Tech. Mapper: TMAP
 - Map to Tunable LUTs (TLUTs)
 - [FPL2008], [ReConFig2008], [DATE2009]
- **Tunable routing bits**
 - Adapted Tech. Mapper
 - Adapted Placer
 - Adapted Router

Parameterizable HDL design



```
entity multiplexer is
port(
  --BEGIN PARAM
  sel : in  std_logic_vector(1 downto 0);
  --END PARAM
  in  : in  std_logic_vector(3 downto 0);
  out : out std_logic
);
end multiplexer;

architecture behavior of multiplexer is
begin
  out <= in(conv_integer(sel));
end behavior;
```


Experiment: 16-tap FIR, 8-bit coefficients*

	Generic	Parameterizable configuration	Specialized
area (LUTs)	2999		1146
clock freq. (MHz)	84		119
gen. time (ms)	0		35634
memory (kB)	0		2^{128} conf.

*: old results on Virtex-II Pro device

Experiment: 16-tap FIR, 8-bit coefficients*

	Generic	Parameterizable configuration	Specialized
area (LUTs)	2999	1301 (-56%)	1146
clock freq. (MHz)	84		119
gen. time (ms)	0		35634
memory (kB)	0		2^{128} conf.

Less area **(-56%)**

- More functionality in one TLUT
- Functionality is moved to the tuning functions

Experiment: 16-tap FIR, 8-bit coefficients*

	Generic	Parameterizable configuration	Specialized
area (LUTs)	2999	1301 (-56%)	1146
clock freq. (MHz)	84	115 (+37%)	119
gen. time (ms)	0		35634
memory (kB)	0		2^{128} conf.

Higher clock frequency **(+37%)**

- Less LUTs can be placed closer together
- Less congestion because less nets

Experiment: 16-tap FIR, 8-bit coefficients*

	Generic	Parameterizable configuration	Specialized
area (LUTs)	2999	1301 (-56%)	1146
clock freq. (MHz)	84	115 (+37%)	119
gen. time (ms)	0	0.166	35634
memory (kB)	0		2^{128} conf.

Reduced generation time **(5 orders)**

- No NP-hard problems (place and route) at run-time
- Only evaluation of the tuning functions

Experiment: 16-tap FIR, 8-bit coefficients*

	Generic	Parameterizable configuration	Specialized
area (LUTs)	2999	1301 (-56%)	1146
clock freq. (MHz)	84	115 (+37%)	119
gen. time (ms)	0	0.166	35634
memory (kB)	0	29	2^{128} conf.

Less memory (**only 29kB**)

- TMAP flow finds similarity between configurations
- Compressed form of all configurations

Outline

- What is Parameterized Run-time Reconfiguration?
- **Profiling** your applications for parameters
- How to put this in practice?

Dynamic Circuit Specialization (DCS)

Identifying applications that might benefit from DCS is hard for the designer:

- Know the application very well
(What are the infrequently changing signals?)
- Be very familiar with Circuit Specialization
(What is the impact of choosing these parameters?)

 Requires a lot of low level work

In general, DCS results are **hard to predict** without actually making the DCS implementation

Dynamic Circuit Specialization (DCS)

Solution: Methodology for identifying DCS opportunities.

How?

- By comparing **all** DCS implementations of the same application

Two problems:

- How to compare different DCS implementations?
- Too many possible implementations (one for every possible set of parameters)

How to compare implementations?

Use the Functional Density as a measure for implementation efficiency.

$$FD = \frac{N}{T \cdot A}$$

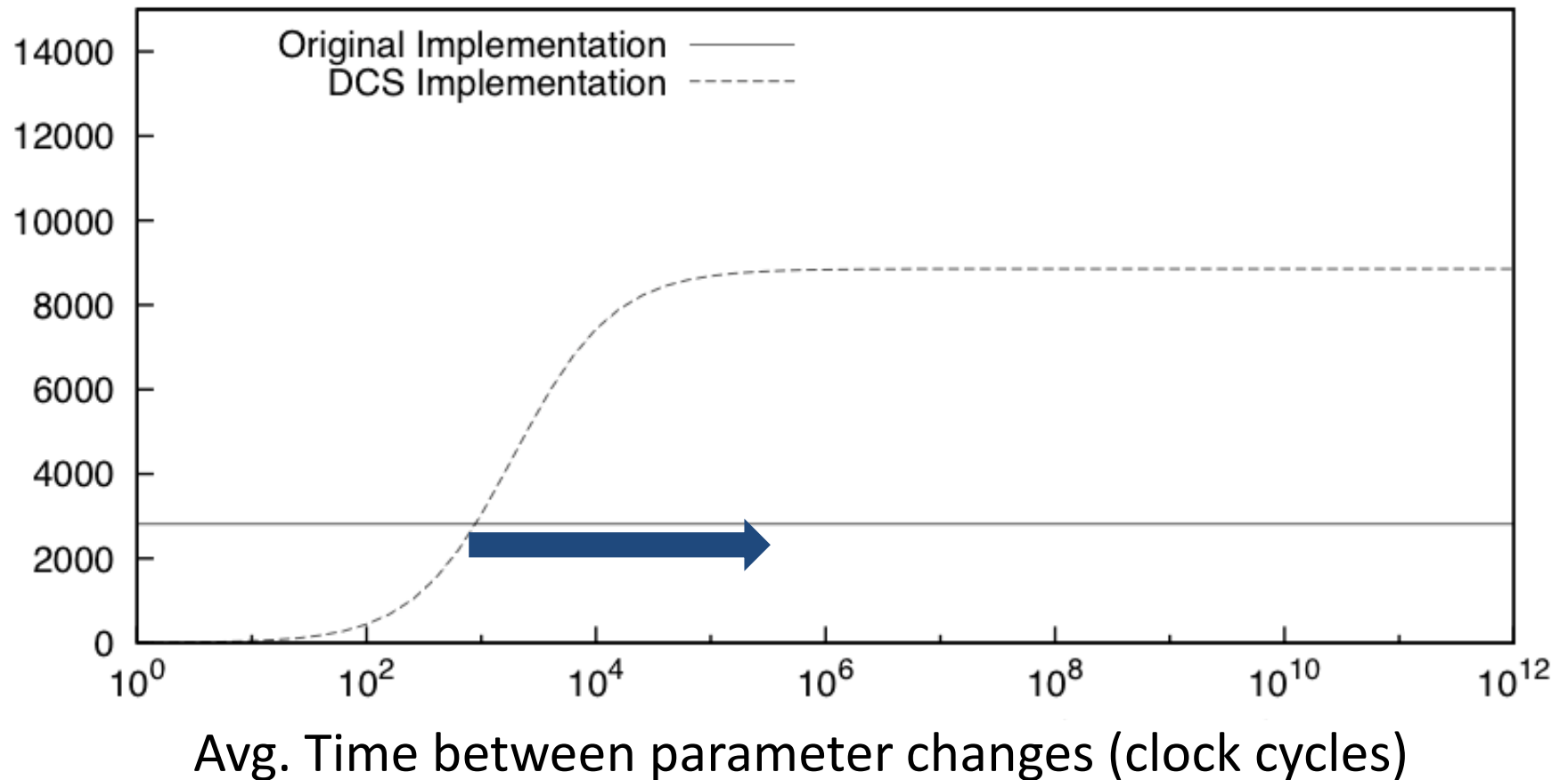
A: The area needed

T: The total execution time

N: The number of operations

*A. M. Dehon, Reconfigurable architectures for general- purpose computing, Massachusetts Institute of Technology, 1996.

Parameter Selection



Profiling the RTL

The DCS-RTL profiler, in three steps:

1. List parameter candidates and their dynamic behavior.
(Using a test bench with real-life data)
2. Reduce the number of parameter candidates
3. Calculate the functional density for each remaining parameter candidate

Execution Time

3 RTL-DCS Profiler implementations:

1. Exact FD calculation, no parameter pruning
2. Exact FD calculation, with parameter pruning
3. Estimated FD, without running Place and Route

Table I. Run times of all three RTL-DCS Profiler implementations

Design	Orig. Size (LUTs)	# cand.	after prun.	Total run time (h:m:s)			Run time Impr.
				Exact FD	Exact FD (prun.)	Est. FD	
16-tap FIR filter (8-bit)	2099	17	1	0:45:47	0:14:31	0:12:31	3.65x
32-tap FIR filter (8-bit)	4399	33	1	2:38:19	1:04:09	1:02:33	2.54x
16-tap FIR filter (16-bit)	8977	17	17	2:38:35	2:38:35	1:39:39	1.59x
32-tap FIR filter (16-bit)	17312	33	32	10:34:20	10:20:45	7:19:50	1.41x
RC6 encryption	2772	2	1	0:18:25	0:16:26	0:13:47	1.33x
RC6 decryption	3017	2	1	0:19:26	0:17:21	0:14:42	1.32x
Twofish 128	5491	48	14	2:51:58	0:56:41	0:22:19	7.70x
Twofish 192	6891	63	19	4:22:07	1:20:35	0:32:38	7.99x
Twofish 256	8270	78	22	5:56:24	1:40:47	0:44:34	8.03x
Pipelined AES	12958	15	5	2:18:58	0:57:20	0:29:23	5.19x

Test machine: Intel Core i7-3770 3.40GHz CPU with 32 GB of RAM

Target: Virtex-5 FPGA (XC5VFX70T-1FF1136)

Profiling Quality

Would DCS be beneficial?

Design	Calc. FD	Calc. FD w. Prun.	Est. FD
16-tap FIR (8 bit)	Yes	Yes	Yes
32-tap FIR (8 bit)	Yes*	Yes*	Yes*
16-tap FIR (16 bit)	Yes*	Yes*	Yes*
32-tap FIR (16 bit)	Yes*	Yes*	Yes*
RC6 encry.	Yes	Yes	Yes
RC6 decry.	Yes	Yes	Yes
Twofish 128	No	No	No
Twofish 192	No	No	No
Twofish 256	No	No	No
Pipelined AES	No	No	No

*SRL-reconfiguration is beneficial, HWICAP not

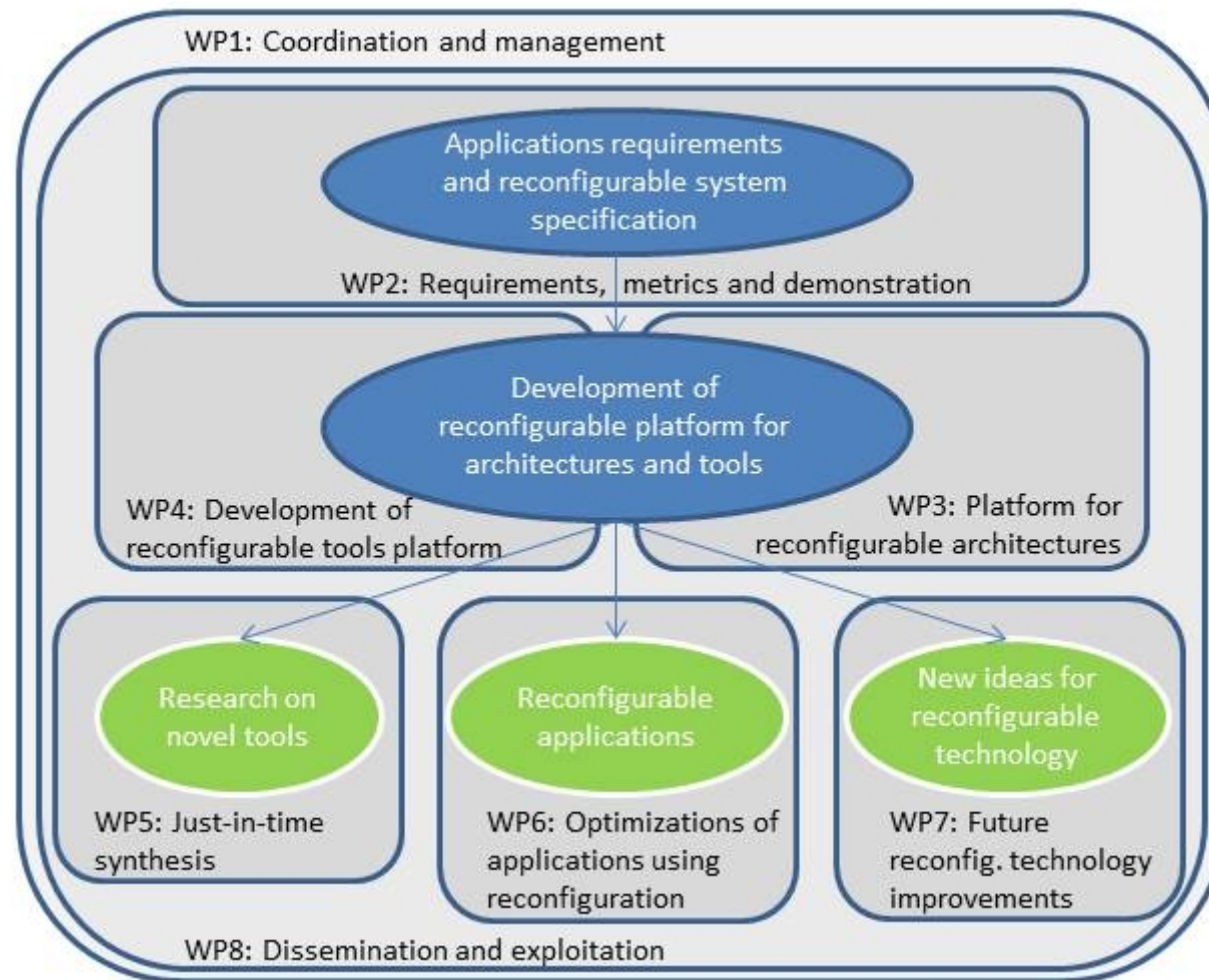
Outline

- What is Parameterized Run-time Reconfiguration?
- Profiling your applications for parameters
- How to put this in practice?

The EXTRA Project

- Exploiting eXascale Technology with Reconfigurable Architectures
- Main objective:
 - Develop an open source research platform for continued research on reconfiguration architecture and tools
- Develop and program HW with run-time reconfiguration as a design concept
- Enable joint optimization of architecture, tools, applications and reconfigurable technology
- Prepare the HPC hardware nodes of the future

EXTRA project structure

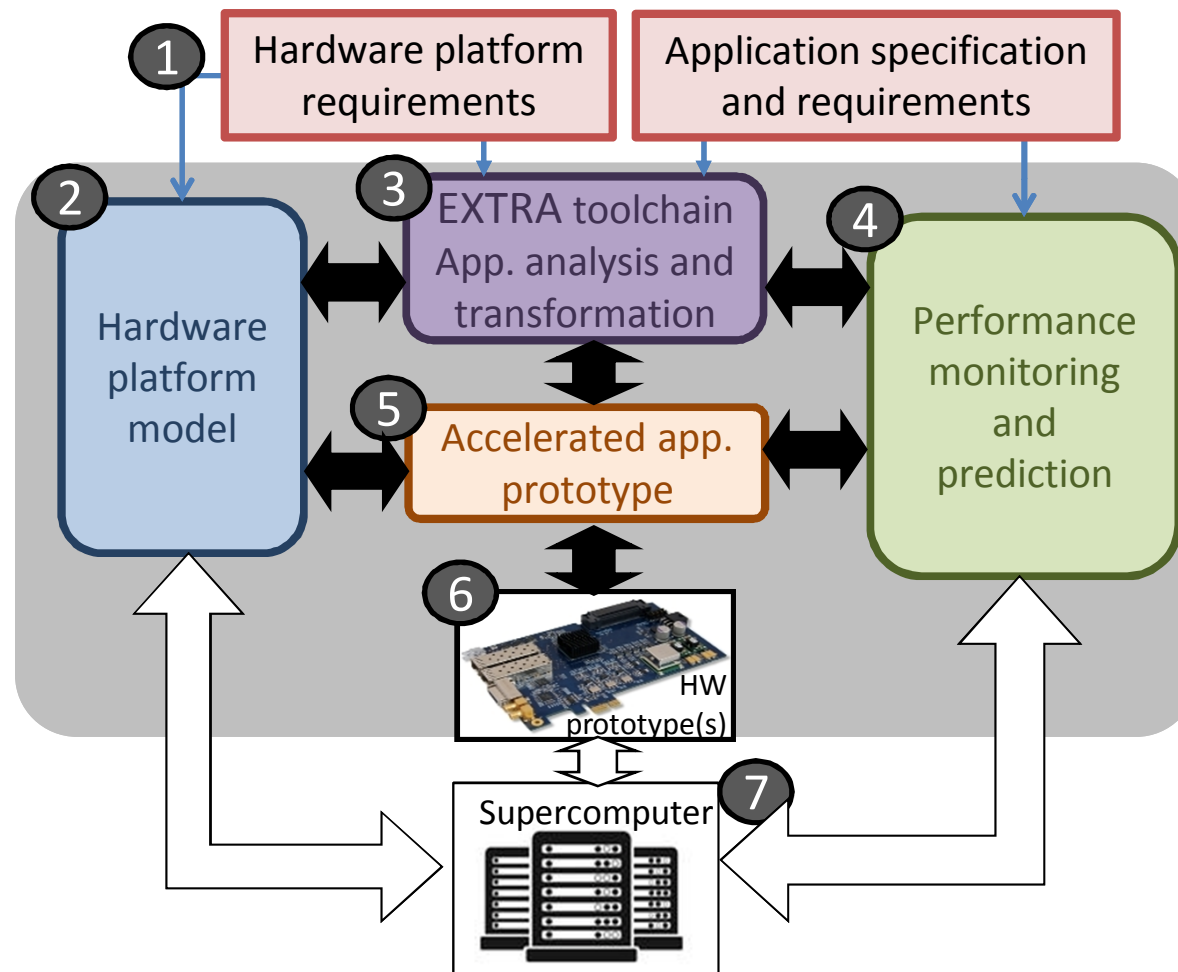


The **basis** of further work in the project

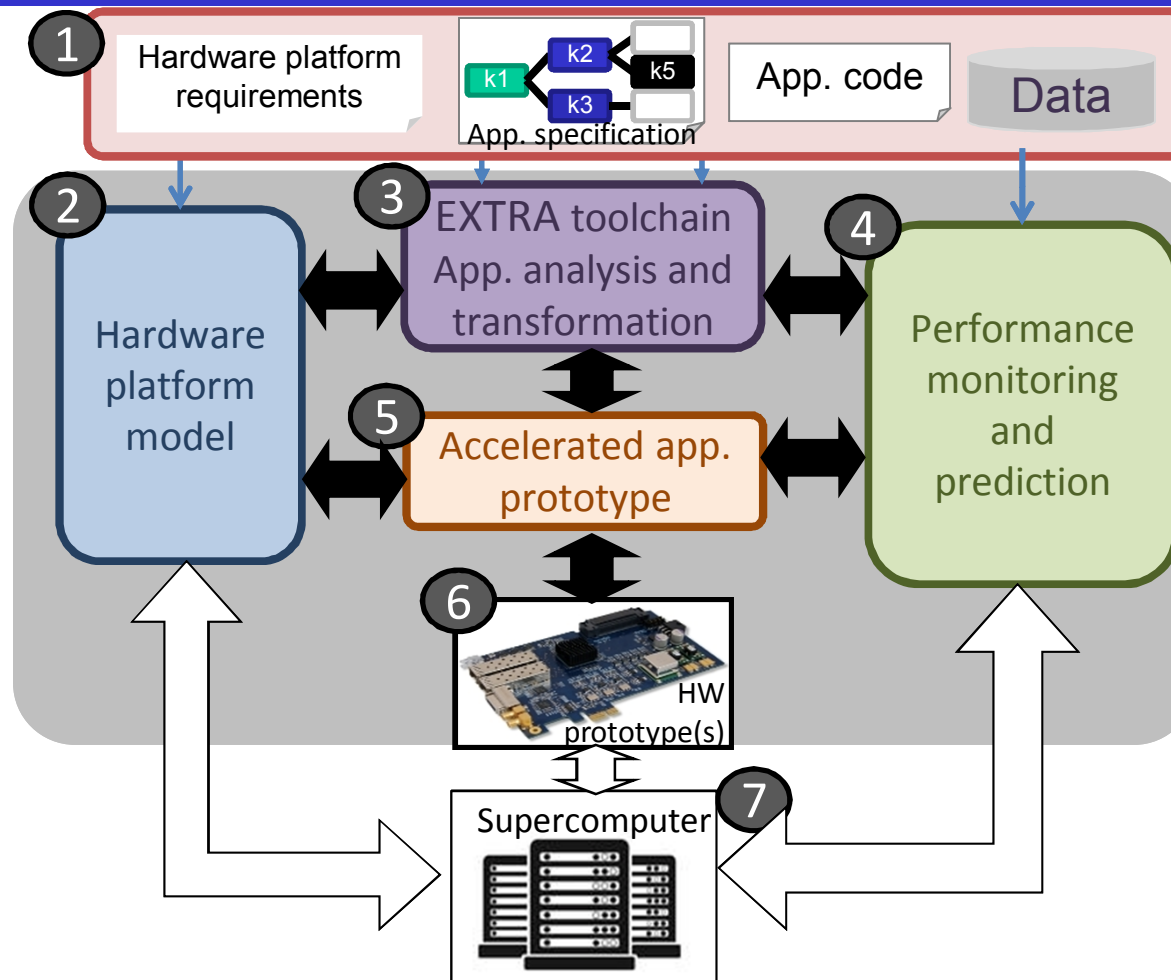
Open source **platform** for researchers

Improvements using the platform

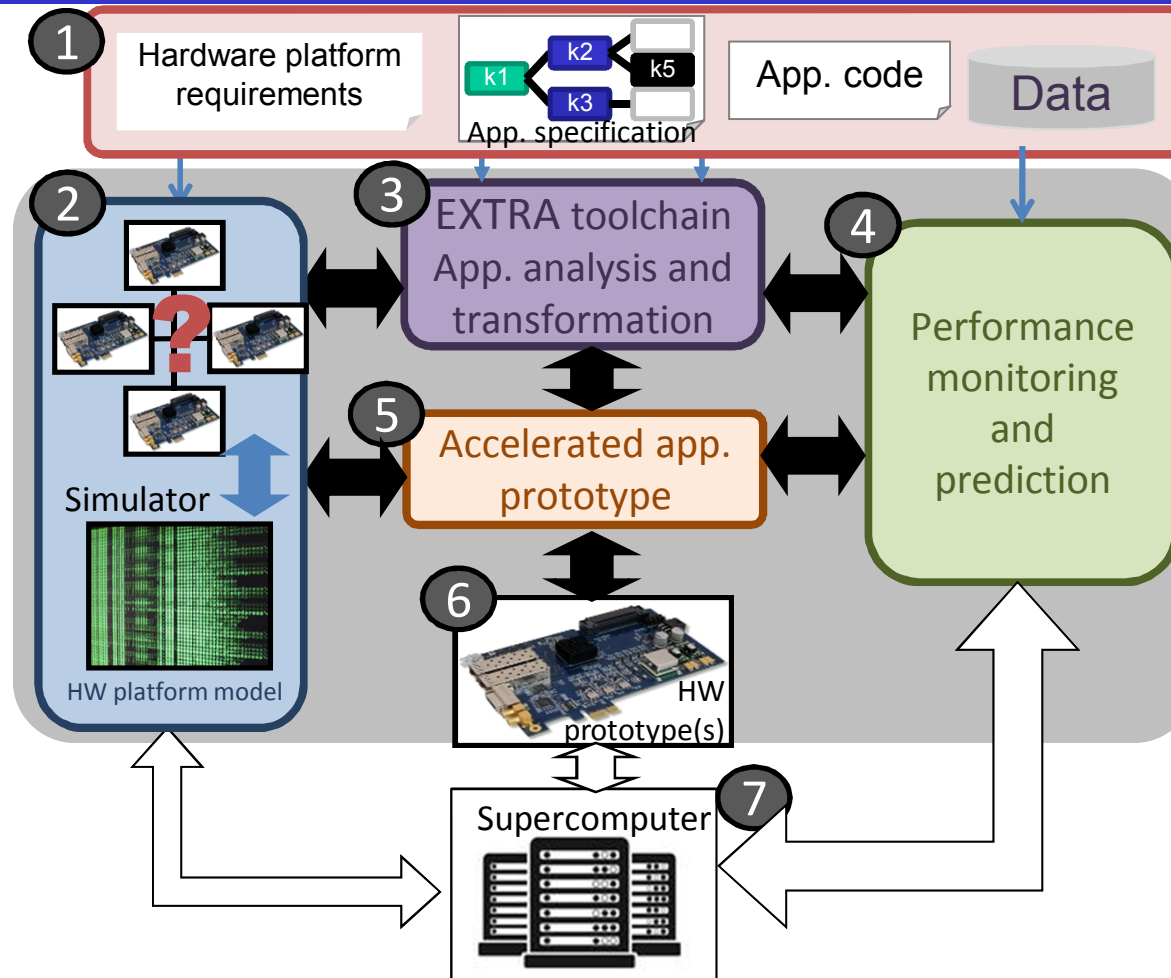
Open research platform



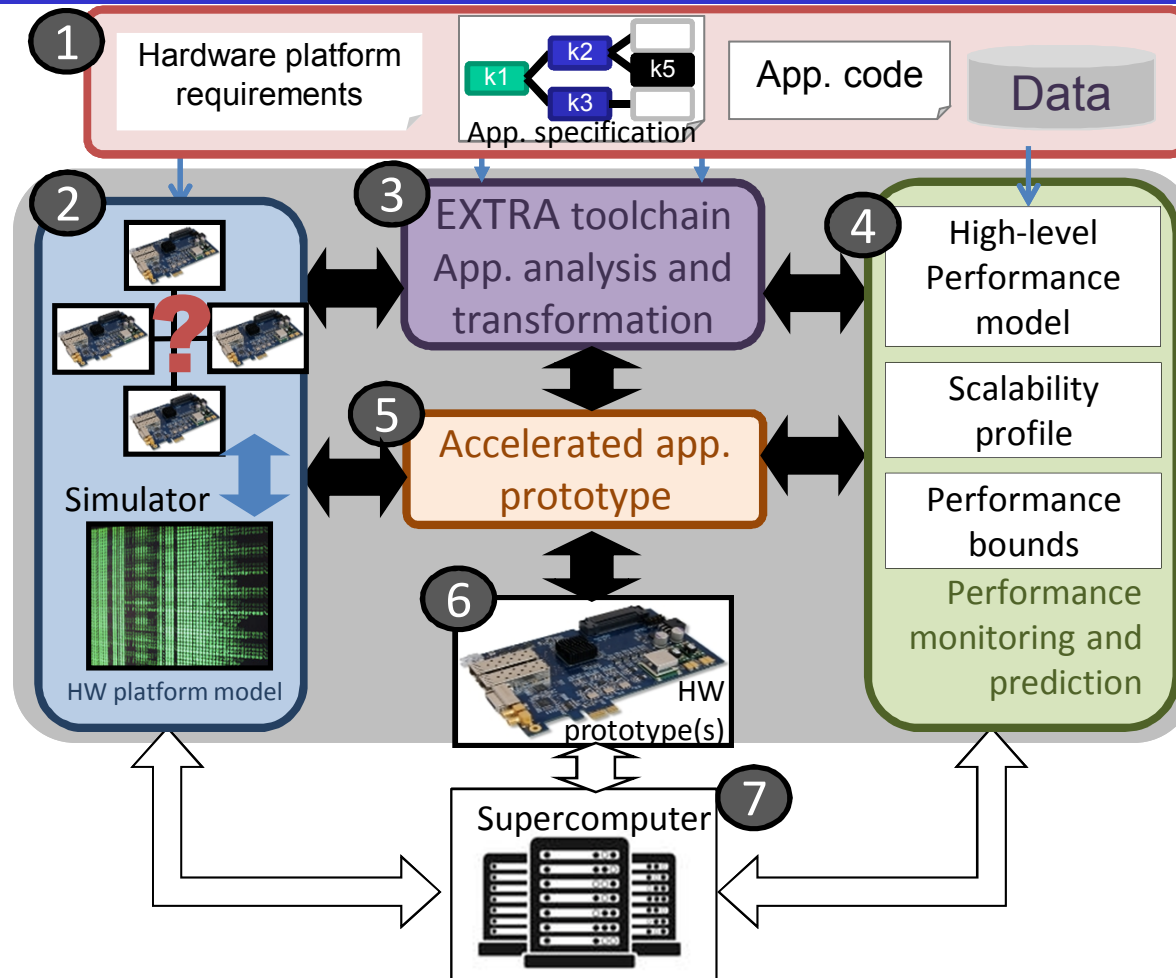
Open research platform



Open research platform



Open research platform



Conclusions

- Run-time reconfiguration will be required in future HPC systems.
- Parameterized run-time reconfiguration allows dynamic circuit specialization.
- Parameter Pruning and FD estimate greatly reduce time required for DCS profiling, but still allow correct identification of DCS opportunities.
- Practical implementations work and are being explored in open research platform of the EXTRA project.

Last slide

- Much of this work was done in the framework of the EU-FP7 project FASTER
- Current work (open research platform) is part of the EU-H2020 project EXTRA (www.extrahpc.eu)
- Questions?
- Tools at https://github.com/UGent-HES/tlut_flow
- More information: <http://hes.elis.ugent.be/>

