Application-Specific Reconfigurable Computing:
Architectures, Applications and Tools

Sergei Sawitzki
saw@fh-wedel.de

NetWare 2019, Nice / Saint Laurent du Var, France

October 28th, 2019

Outline

1. Introduction

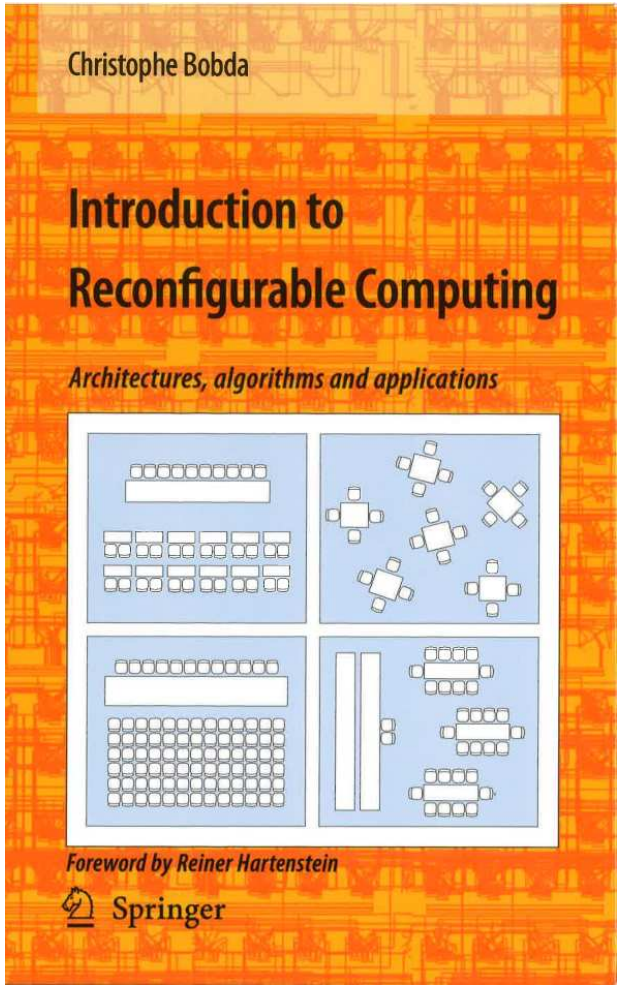
What Does "Reconfigurable Computing" Mean?

Reconfigurable device (reconfigurable processing unit, RPU) is a hardware device able to adapt to the application.

Reconfigurable computing is defined as the study of computation using reconfigurable devices.

Configuration is the process of changing the structure of a reconfigurable device at start-up-time.

Reconfiguration is the process of changing the structure of a reconfigurable device at run-time.



➥ Christophe Bobda, *Introduction to Reconfigurable Computing*, Springer 2007

Field-Programmable Gate-Arrays

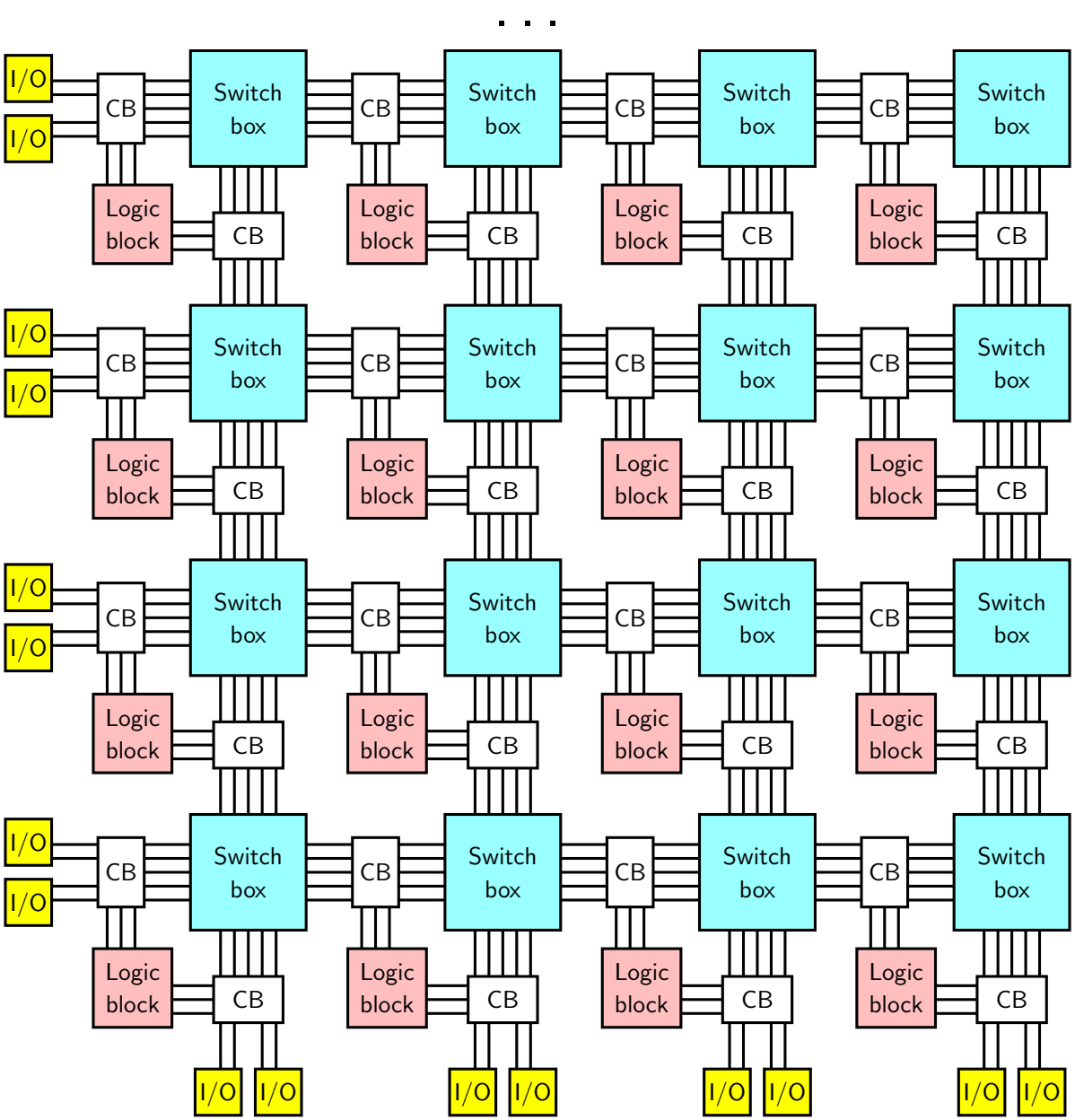
FPGA is the most common type of RPU.

Gate-Array: Logic (transistors) is pre-fabricated, interconnect is added later to implement customer-specific functionality. Both steps are done in the fab. NRE cost reduction, since master wafers fabrication costs are shared among many customers.

FPGA: Logic (look-up-tables) and interconnect are pre-fabricated but not configured, the customer gets an "empty" device and can determine its functionality by configuring it according to the own requirements (hence "field-programmable").

Related terms: Custom Computing Machines (CCM), Reconfigurable Logic (RL), Field-Programmable Logic (FPL), ...

FPGA: Basic Concepts

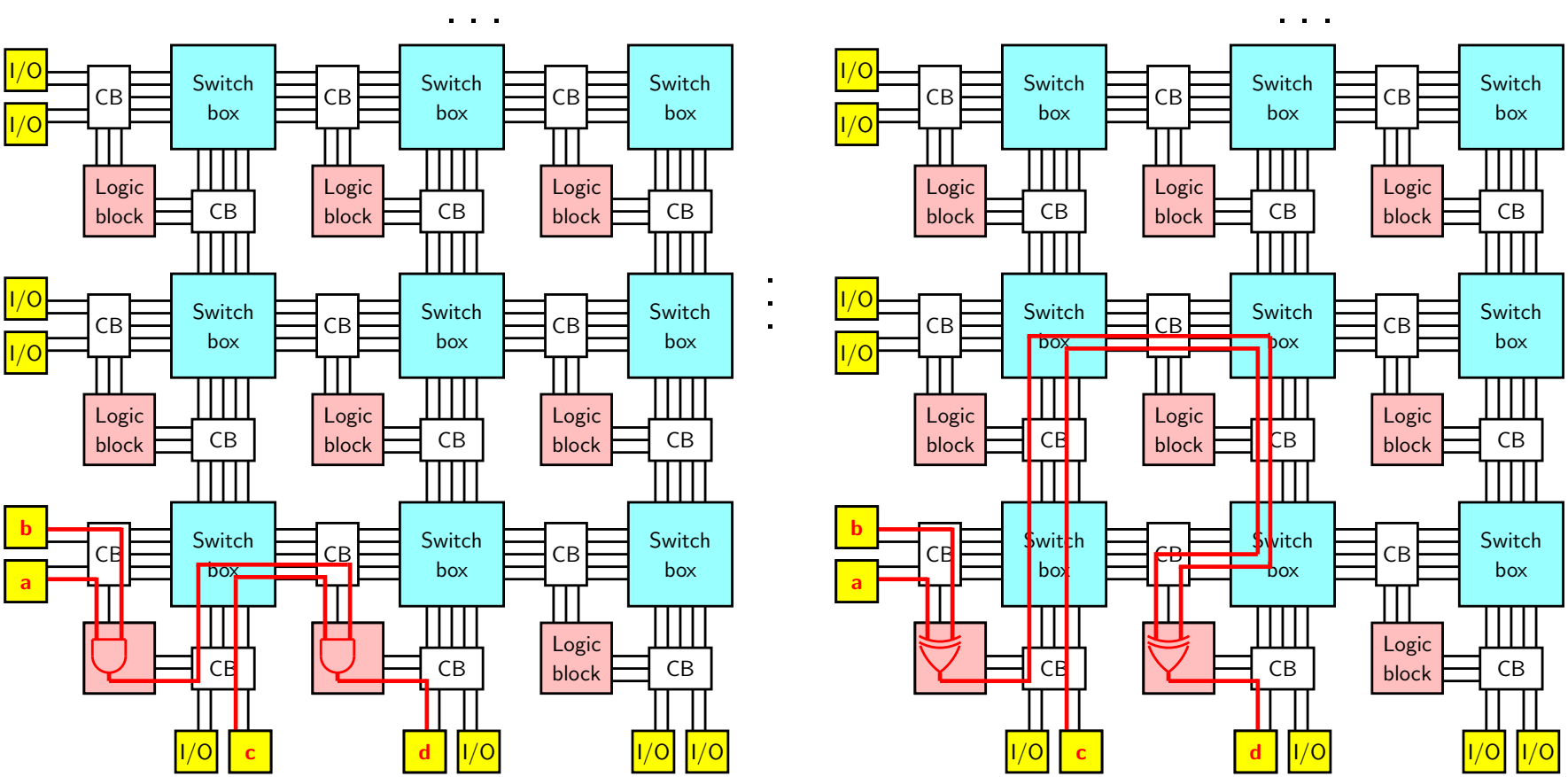


Logic block: universal logic module, in most cases a Look-up-Table (LUT)

Connection block, CB: configurable interconnect (logic to routing channel)

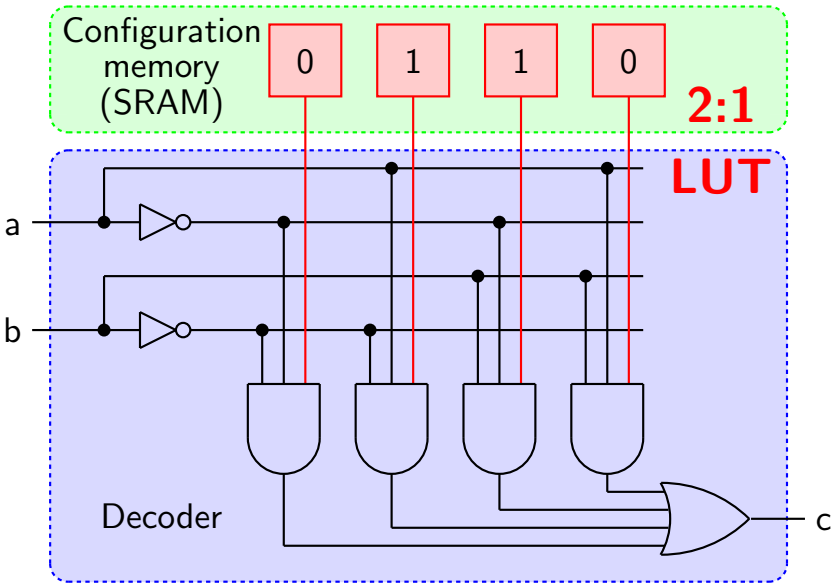
Switch box: configurable interconnect (routing channel to routing channel)

FPGA: Illustrating the Principle

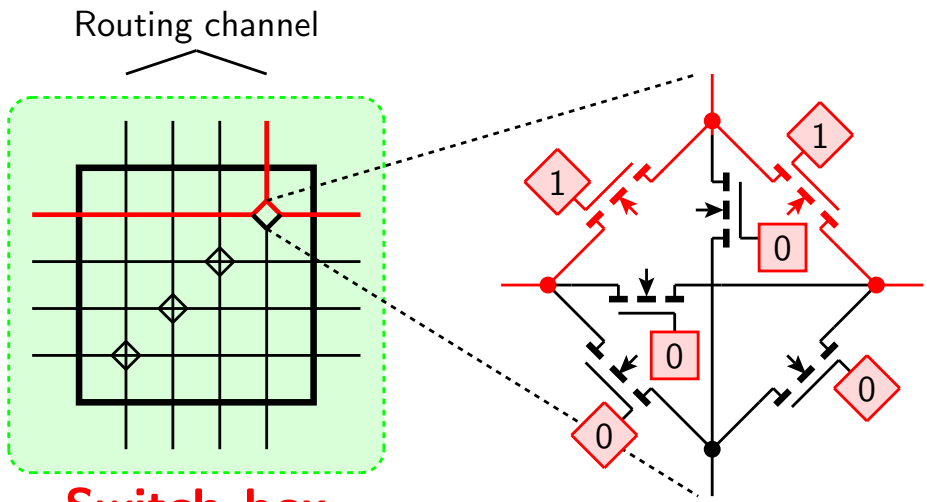


- different logic functions using the same hardware
- functionality is changed by reconfiguration: restructuring the hardware, not changing the software

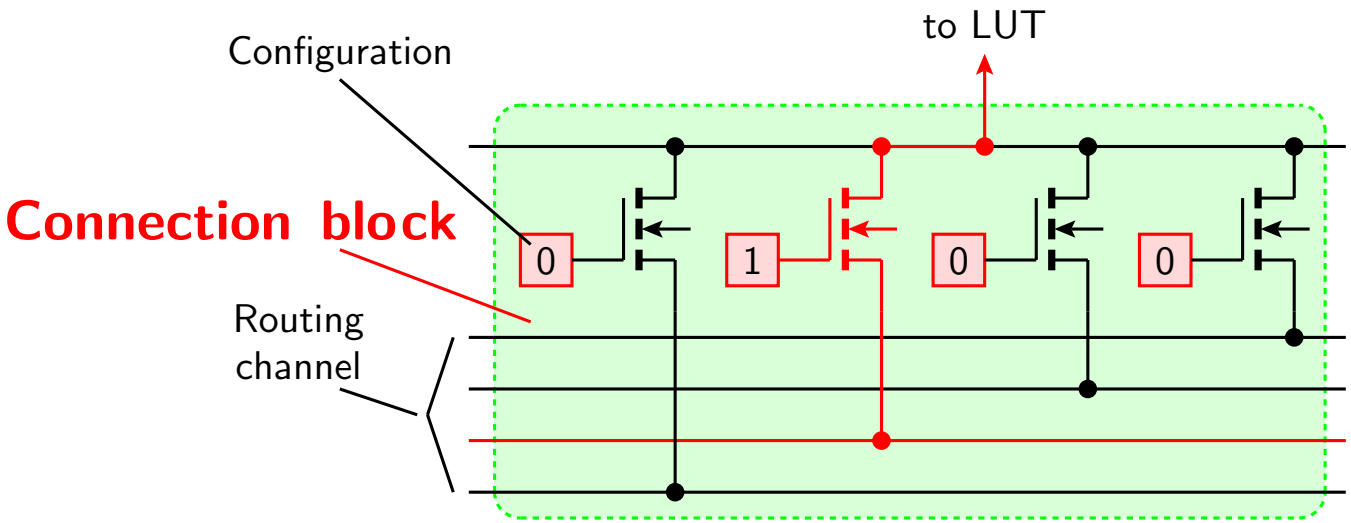
How Do We Reconfigure the Device?



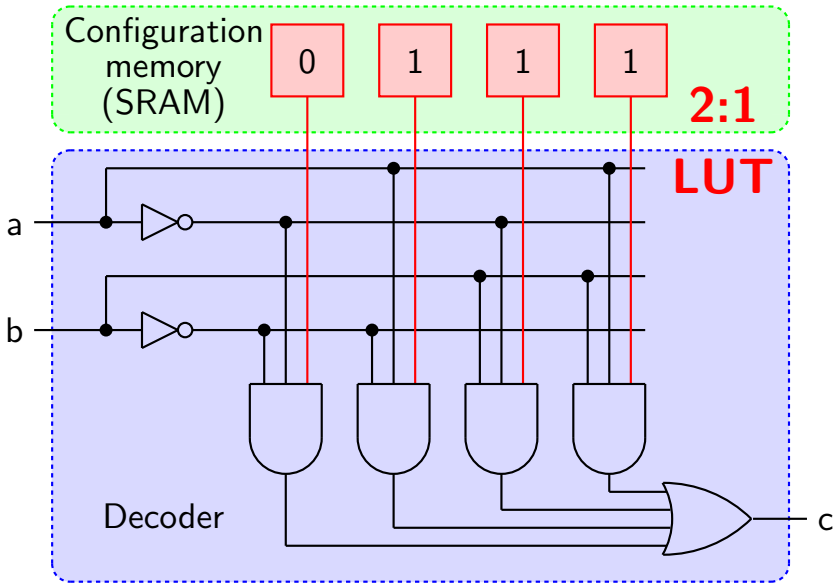
$c = a \oplus b$



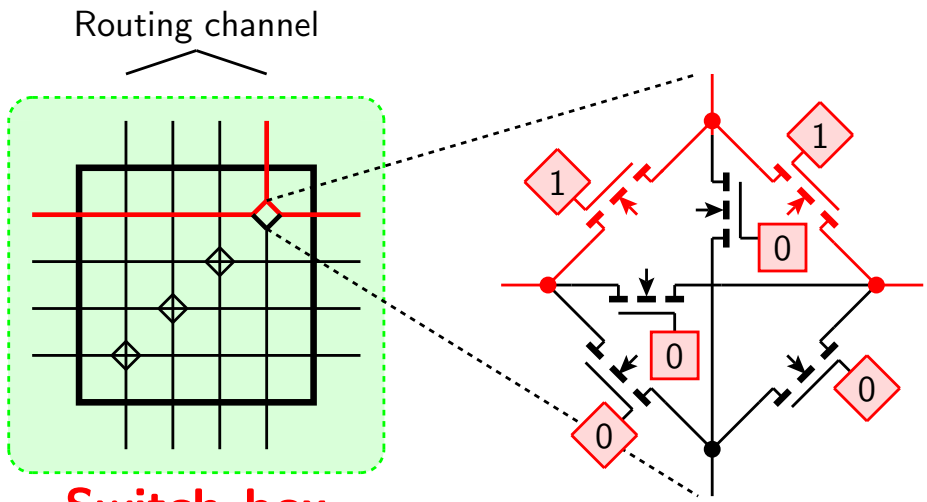
Switch box



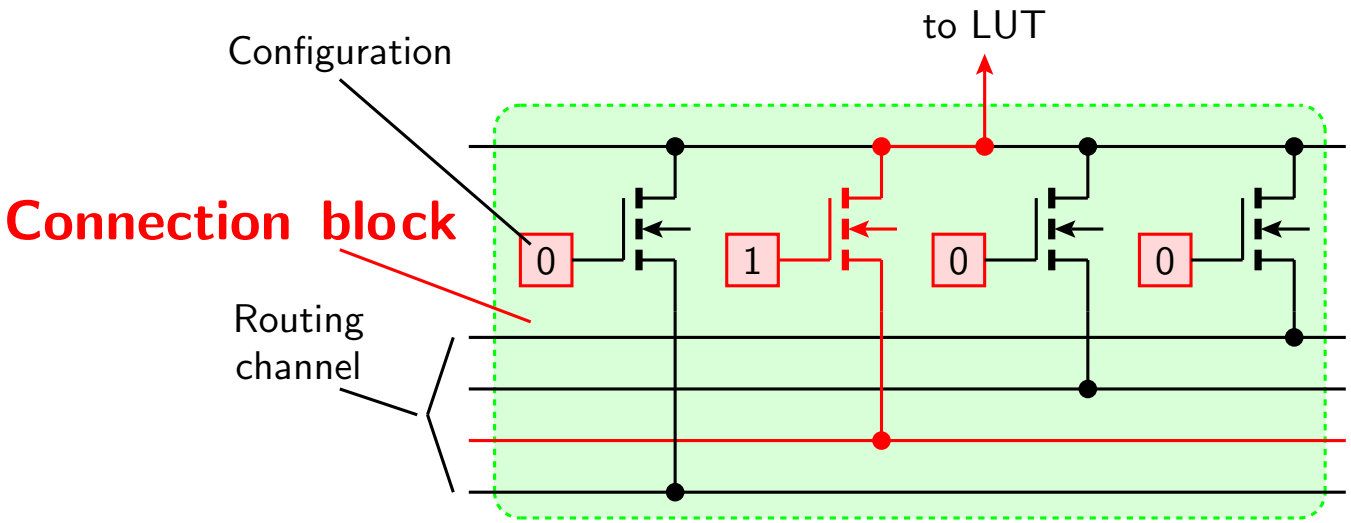
How Do We Reconfigure the Device?



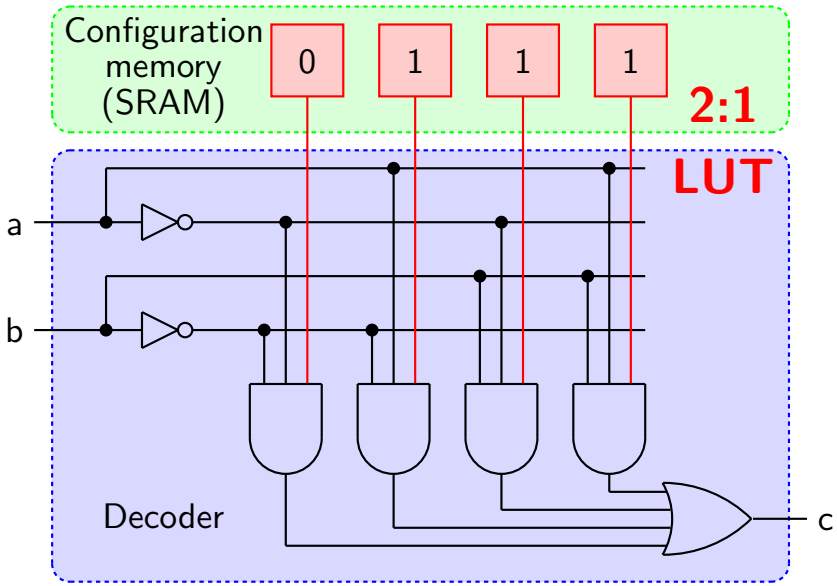
$c = a \vee b$



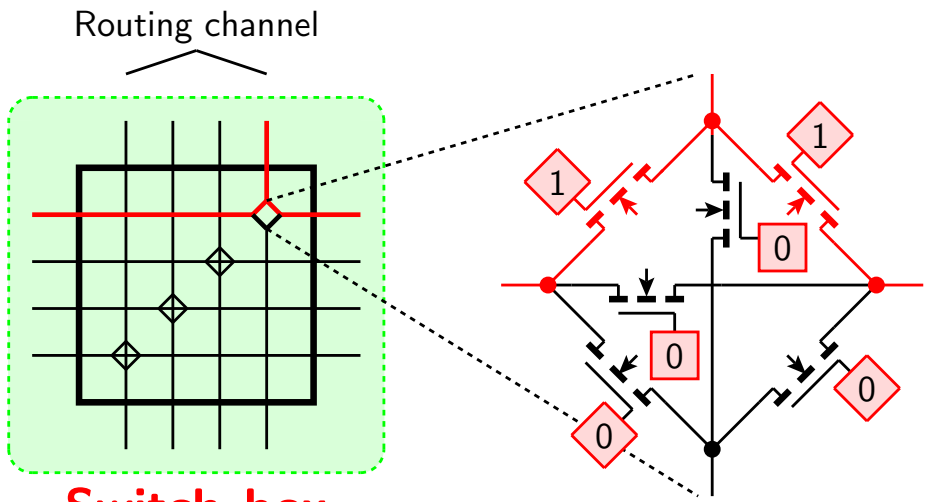
Switch box



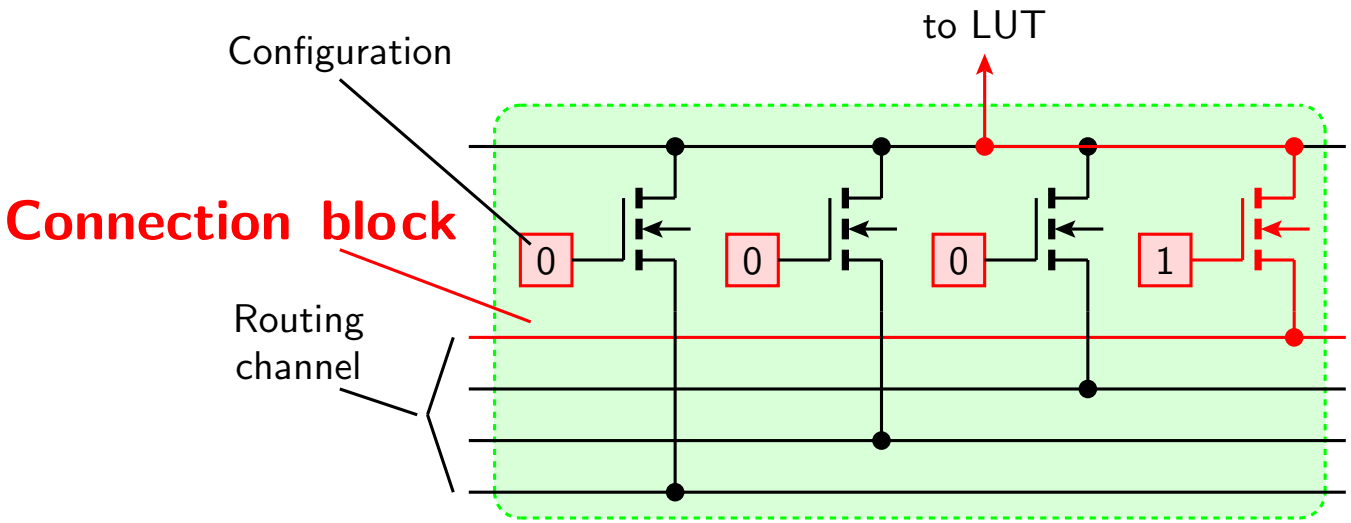
How Do We Reconfigure the Device?



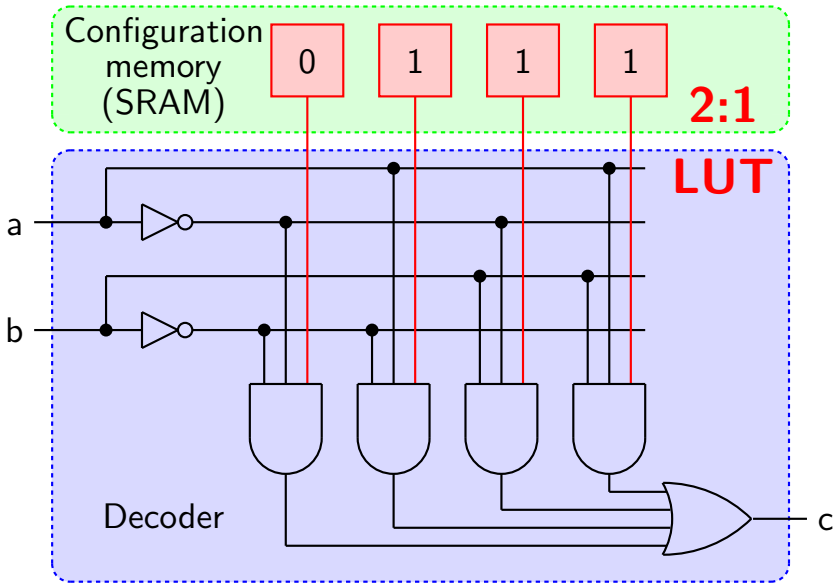
$c = a \vee b$



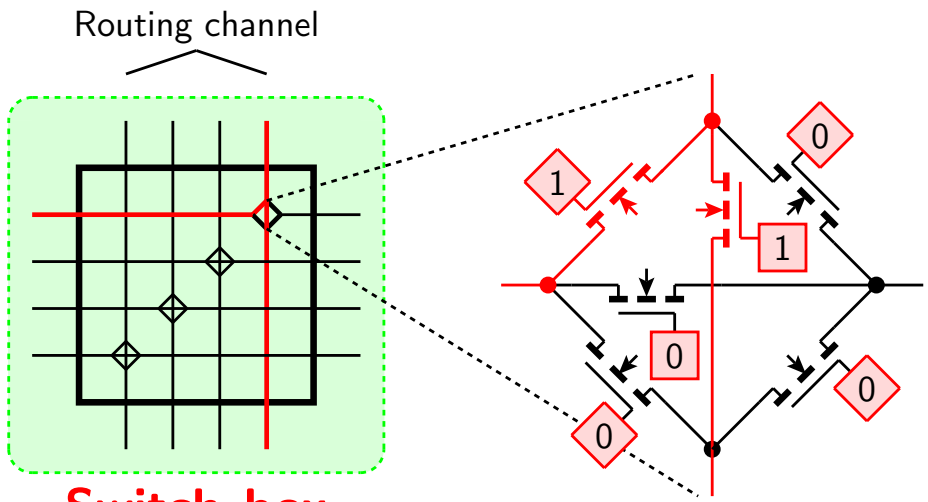
Switch box



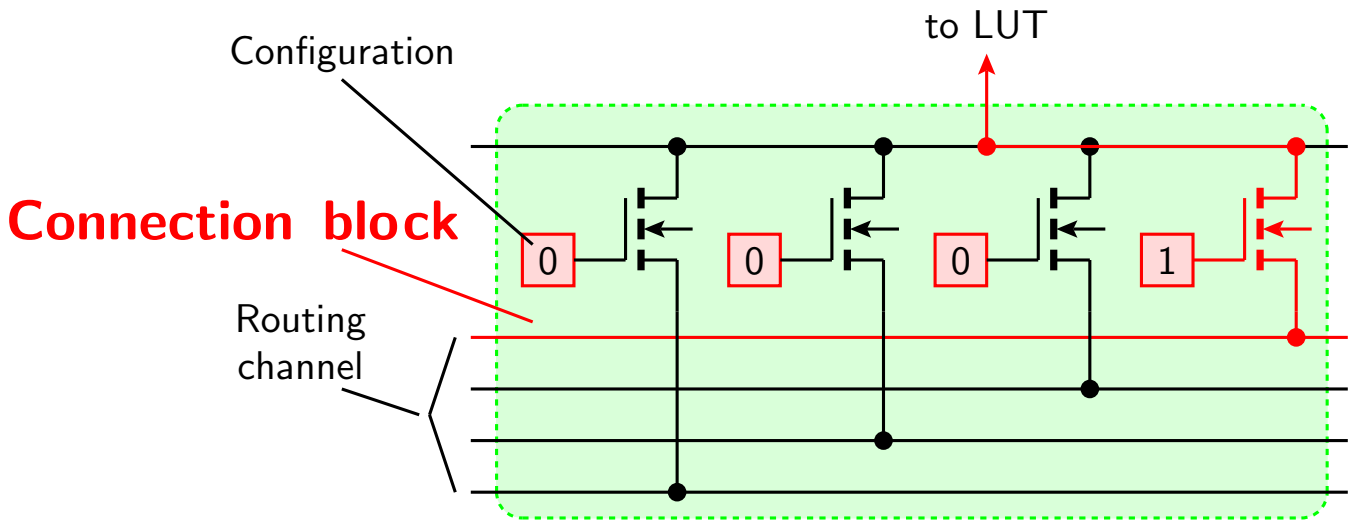
How Do We Reconfigure the Device?



$c = a \vee b$



Switch box

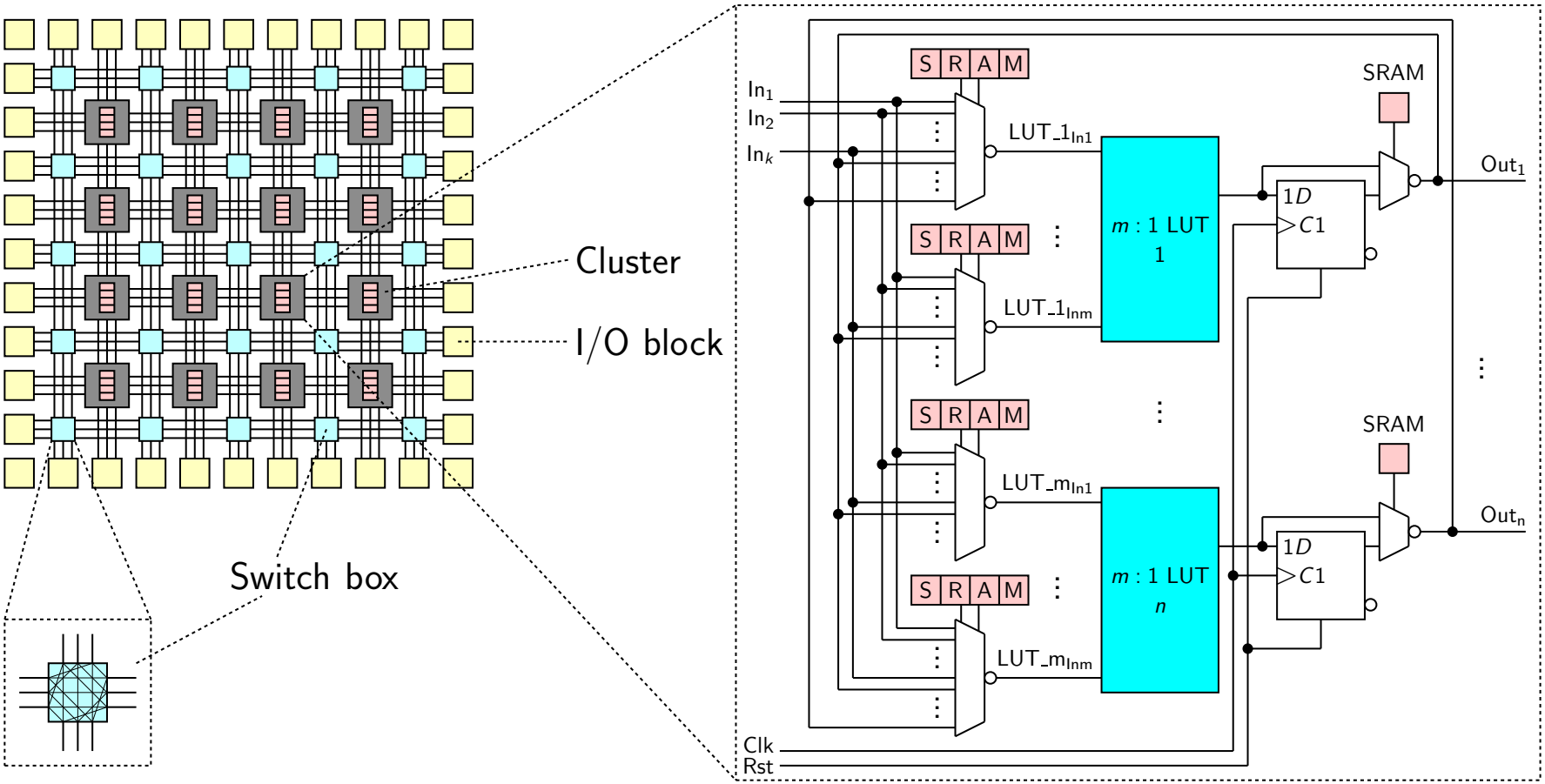


Modern Devices Are Much More Sophisticated

- hierarchical interconnect (LUTs are grouped into clusters, fast local interconnect, slower inter-cluster-interconnect, several clustering levels)
- fracturable LUTs
- embedded memories
- pre-fabricated IP modules
  - transmitter
  - (multi)processor cores
  - PLL, DLL
  - specialized digital signal processing (DSP) logic blocks
  - standard interface modules (PCIe, USB, ...)
  - ...

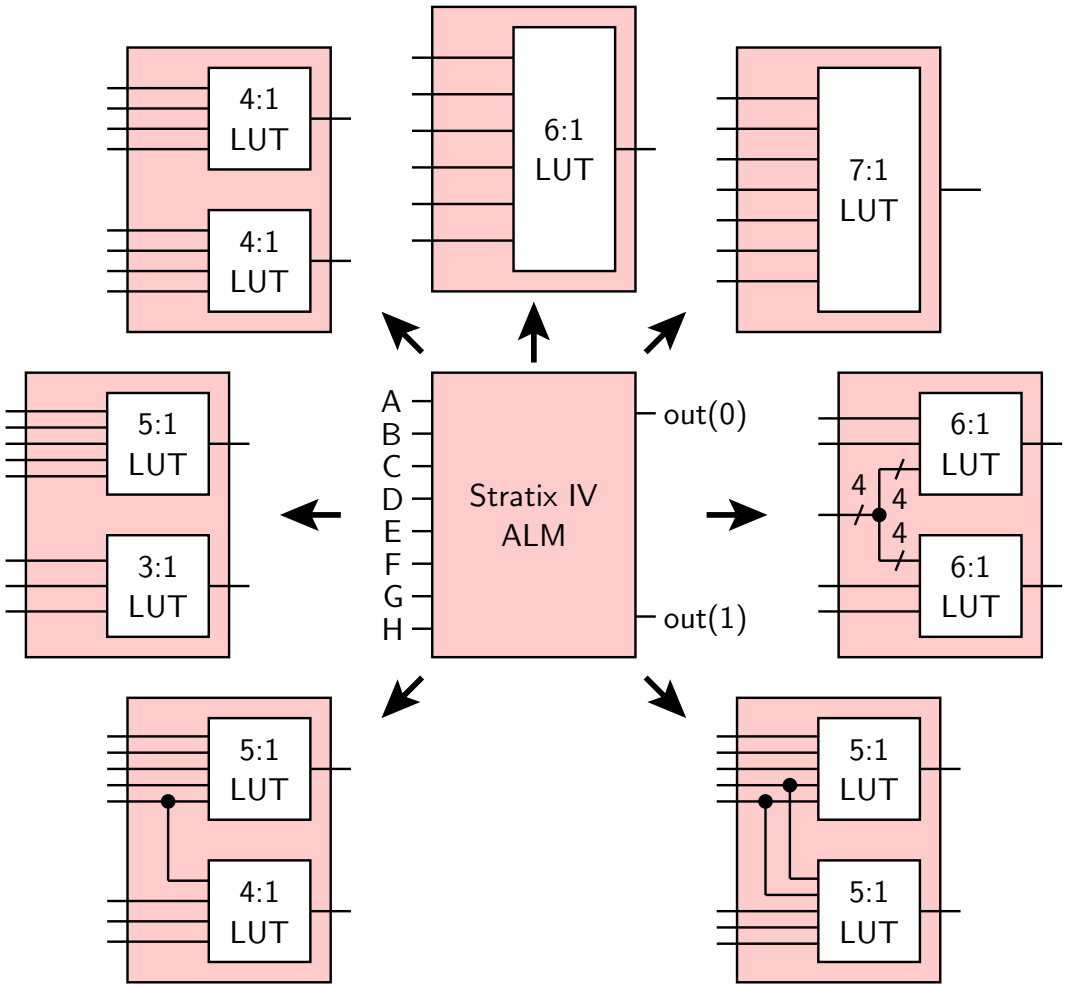
➥ Reconfigurable Systems-on-Chip (SoC) or even Multi-Processor System-on-Chip (MPSoC)

Hierarchical FPGA



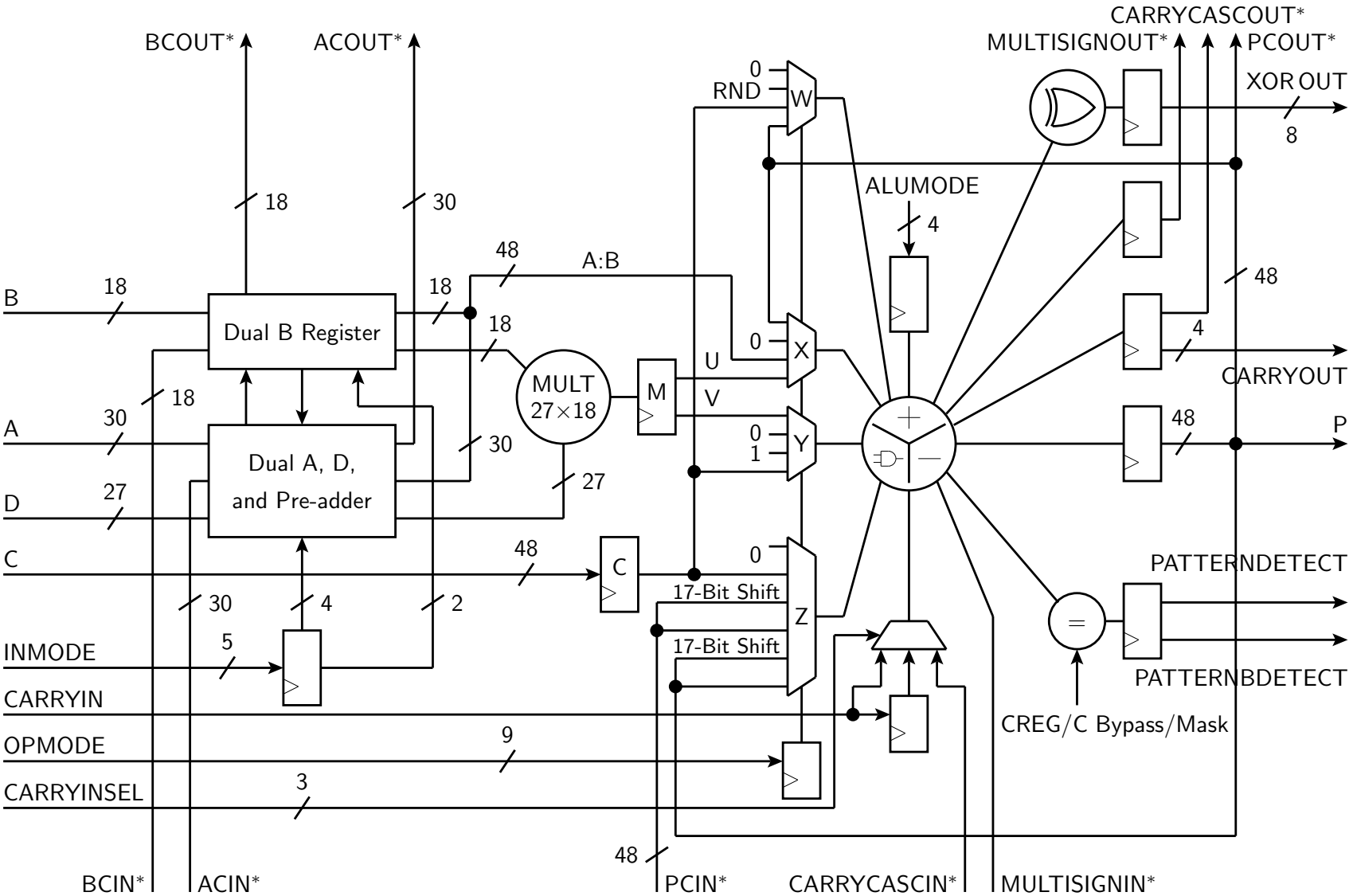
- extendable to more than 2 hierarchy levels
- interconnect gets slower with every additional hierarchy level (fast local vs. slower global)

Fracturable LUT (Altera/Intel)

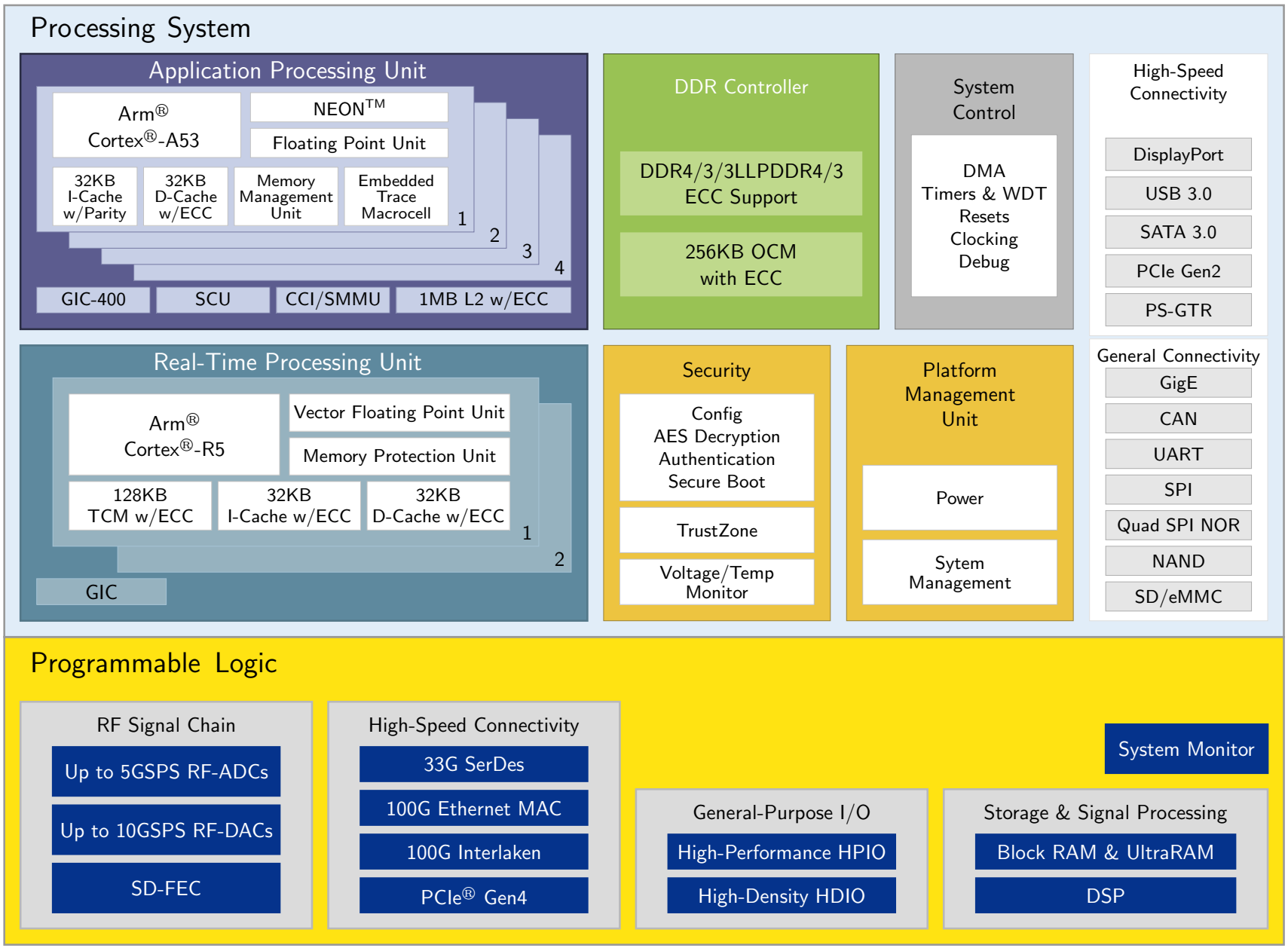


ALM stands for “Adaptive Logic Module”

Xilinx DSP48E2 Block



Xilinx Zynq UltraScale+ RFSoc (www.xilinx.com)



Standardization vs. Specialization

General-purpose FPGAs suite almost any application domain but imply a significant overhead. This issue is addressed by the manufacturers:

- different product families
  - memory-oriented
  - logic-oriented
  - DSP-oriented
  - processor-centered
  - communication-centered
  - AI-support
- different device complexity within one family:
  - more or less logic blocks, memory, DSP blocks etc.
  - different technology nodes for the same device architecture

➥ Still, commercially available FPGAs remain general-purpose computing engines

Case Study 1: Xilinx Inc.

- CPLD device family
  - CoolRunner-II
- 4 FPGA device families
  - Spartan-6 and -7 (low-cost)
  - Artix-7 (low-cost)
  - Kintex-7, Kintex UltraScale, Kintex UltraScale+ (mid-range)
  - Virtex-5, -6 and -7, Virtex UltraScale, Virtex UltraScale+ (high-end)
- SoC and MPSoc
  - Zynq-7000
  - Zynq UltraScale+
- Adaptive Compute Acceleration Platform (ACAP)
  - Versal

All product names are registered trademarks of Xilinx Inc.

Case Study 2: Intel

- 3 FPGA/CPLD device families
  - MAX II (low-cost)
  - MAX V (low-cost)
  - MAX 10 (non-volatile)
- 4 FPGA device families
  - Cyclone III, IV, V and 10 (low-cost)
  - Agilex F (general purpose), I (interface) and M (memory)
  - Arria I, II and V (mid-range)
  - Stratix III, IV, V and 10 (high-end)
- SoC and MPSoc
  - Cyclone V
  - Arria V and 10
  - Stratix 10
  - all Agilex product lines

Mainstream vs. Application-Specific

Mainstream Reconfigurable Computing is application-specific by definition:

- RPUs can be configured to suit the needs of almost any application

➥ but at high price, since a lot of overhead is involved to make them generally applicable

Application-Specific Reconfigurable Computing goes one step further:

- stick to a few (or even a single) application domain (or even just a couple of applications)
- reduce the overhead as far as possible

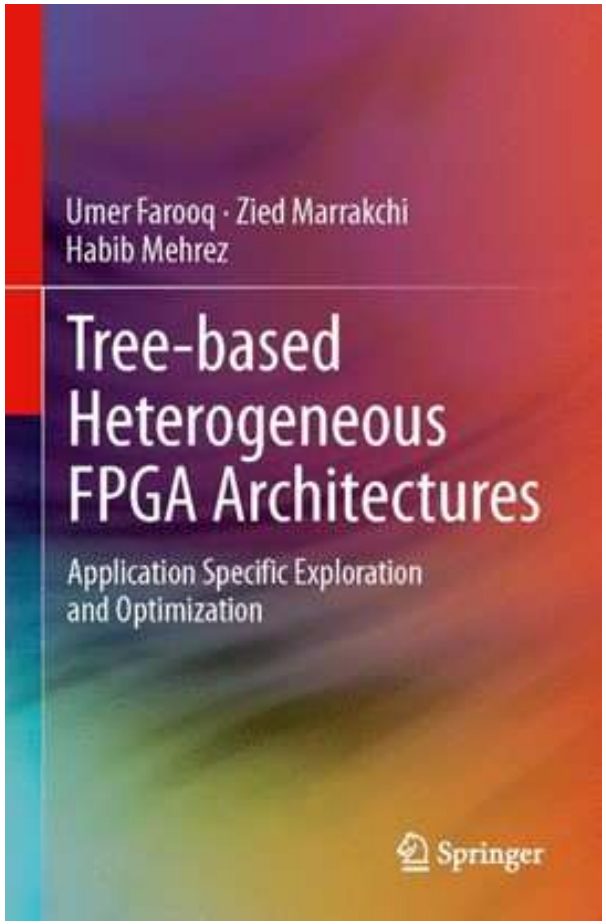
2. Architecture Studies

Application-Specific Inflexible FPGA

Basic approach:

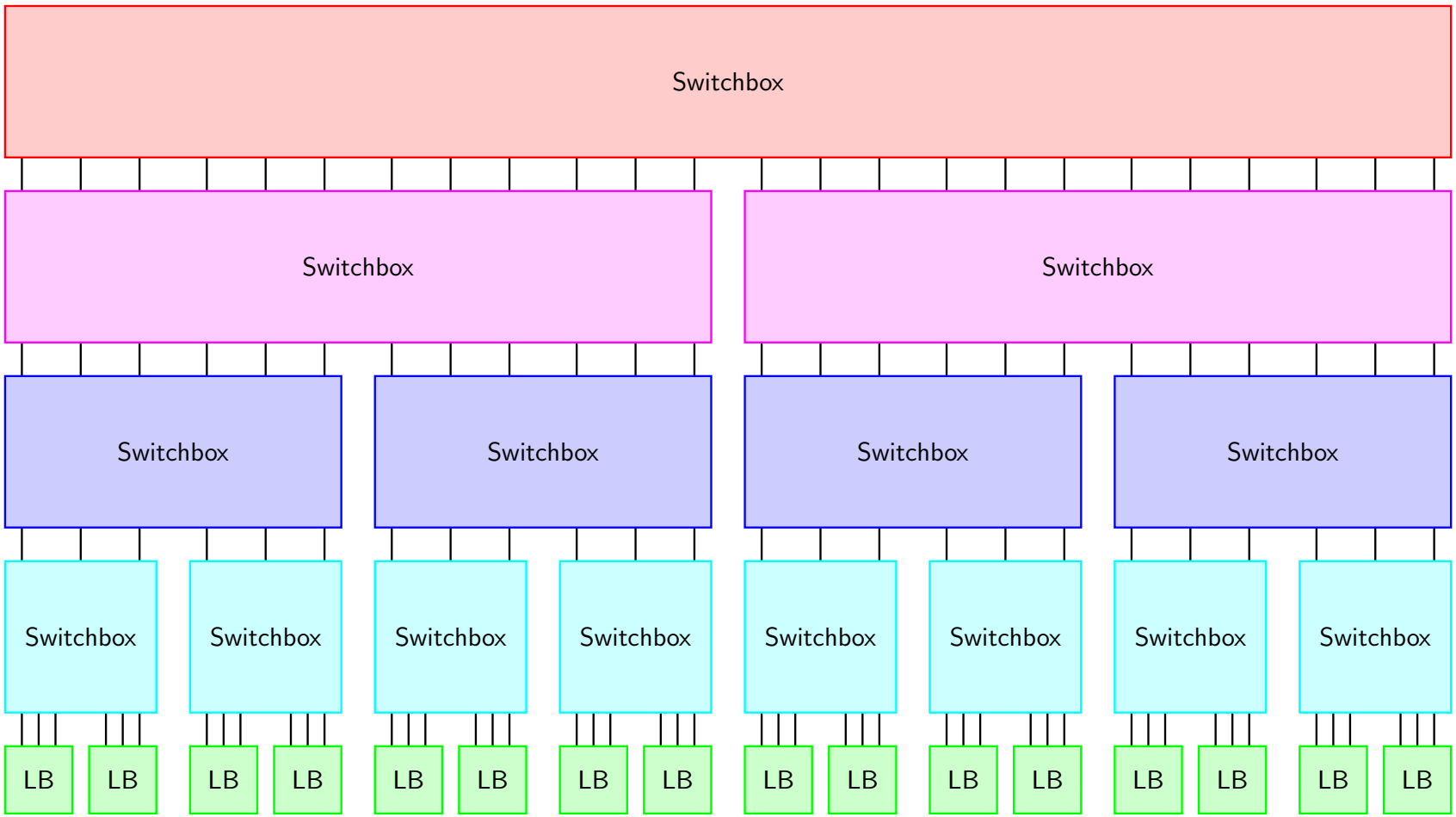
- use hierarchical FPGA architecture with variable number of levels
- optimize interconnect to route a predefined set of netlists only
- replace reconfigurable logic blocks by hard-macros (if possible and useful)
- reconfigure the ASIF for each netlist individually (time-multiplexing)

➥ 70 % area reduction compared to the general-purpose hierarchical FPGA implemented in the same technology

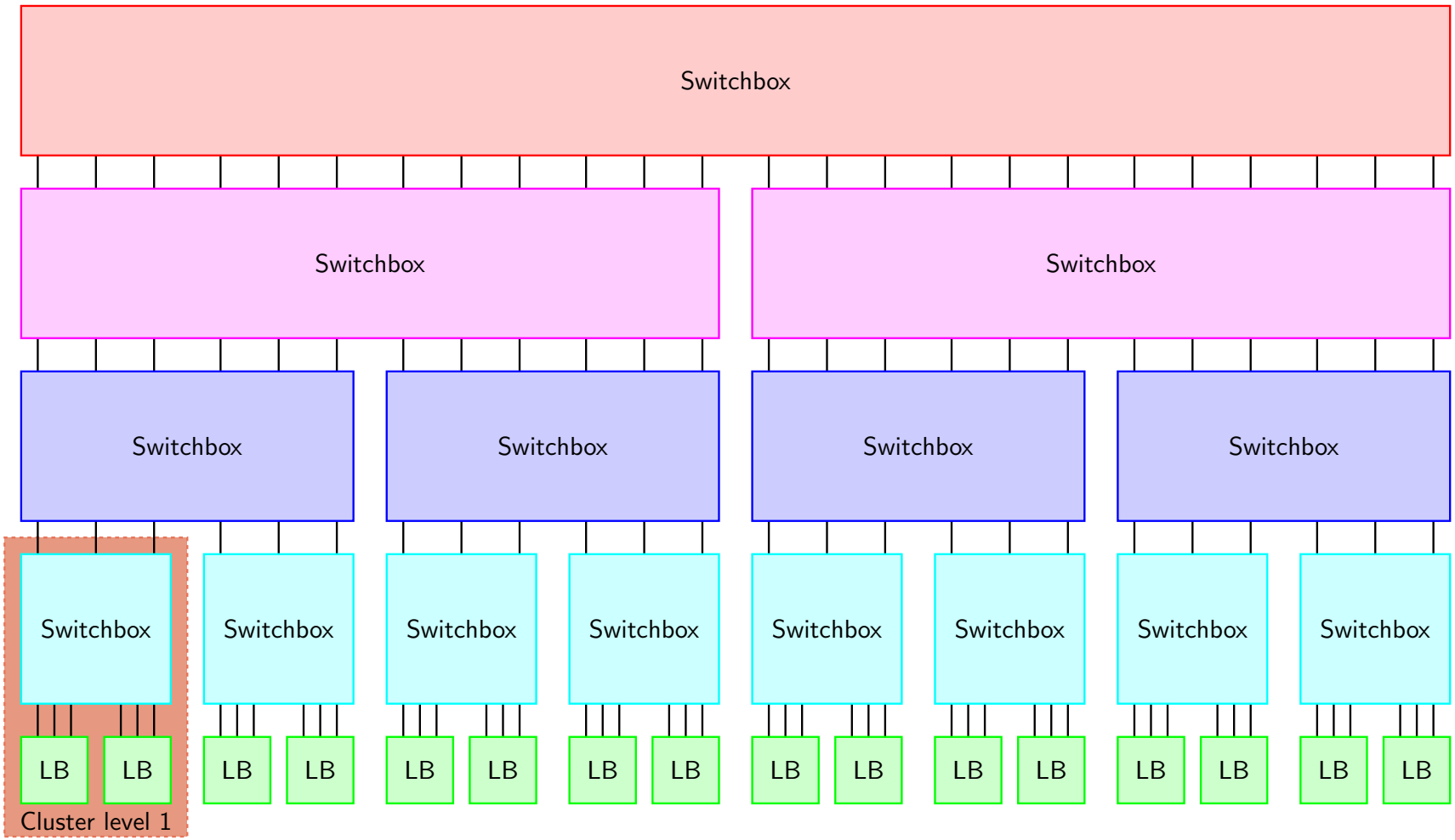


➥ Farooq, Marrakchi, Mehrez, *Tree-based Heterogeneous FPGA Architectures*, Springer 2012

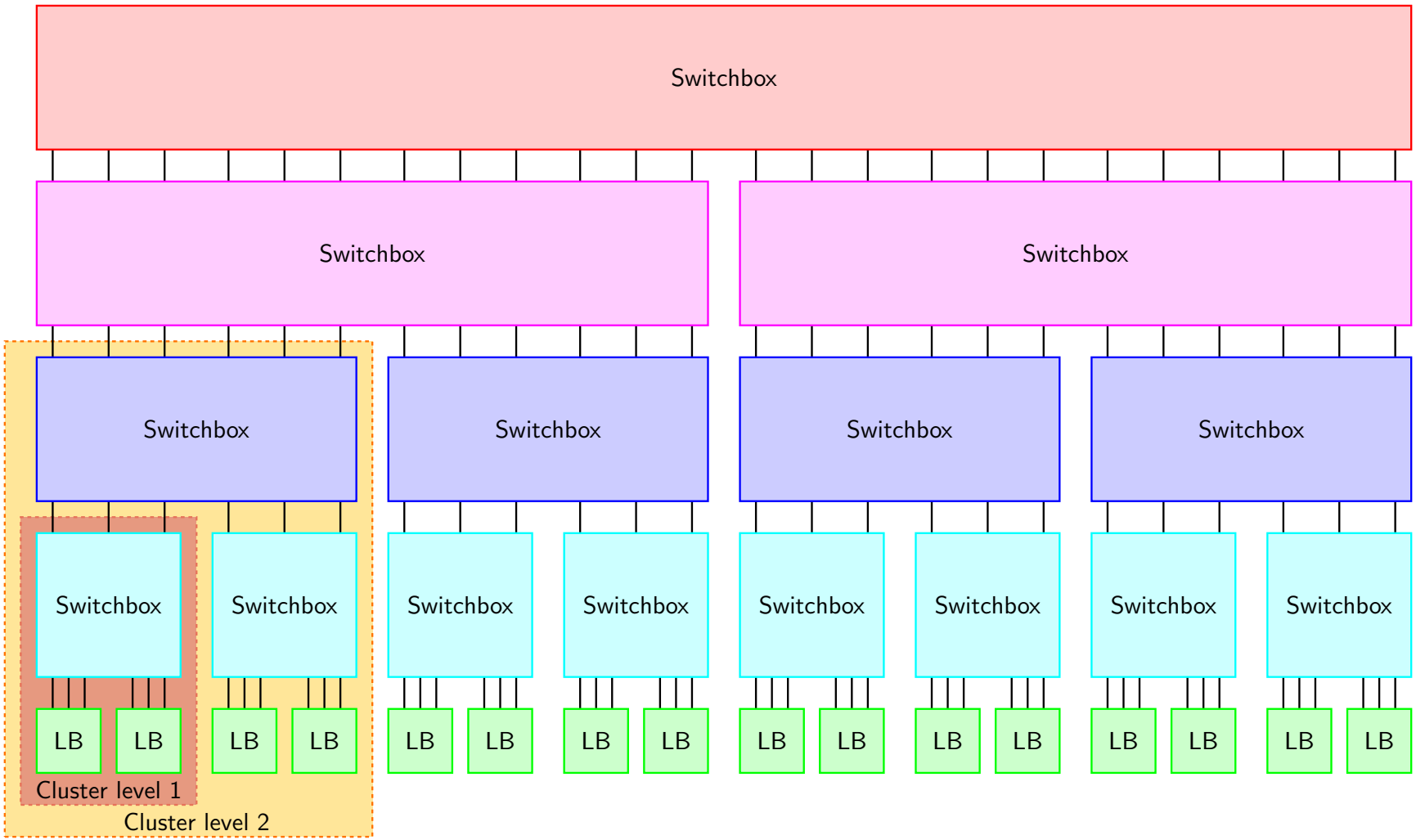
A 4-Level Hierarchical FPGA



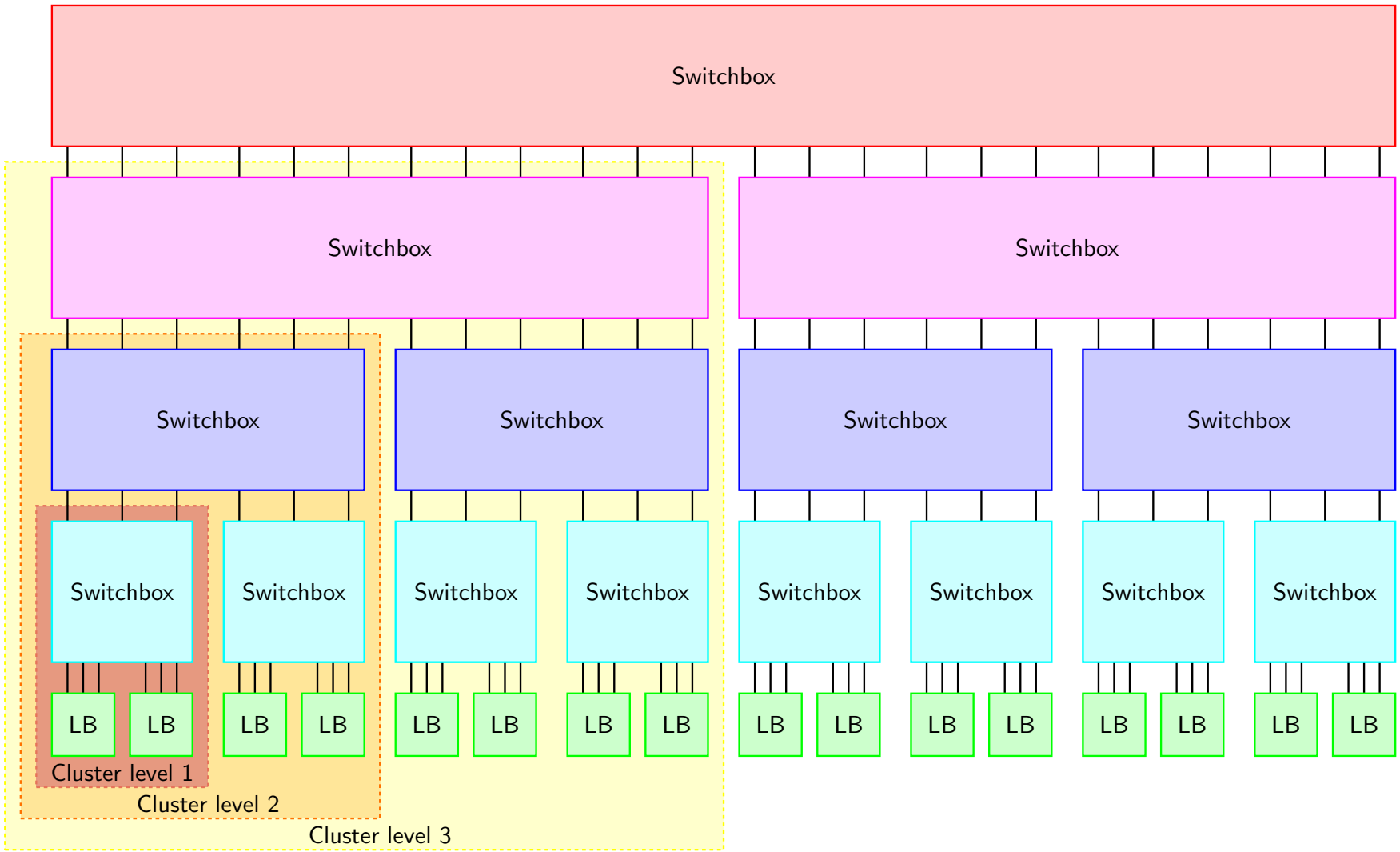
A 4-Level Hierarchical FPGA



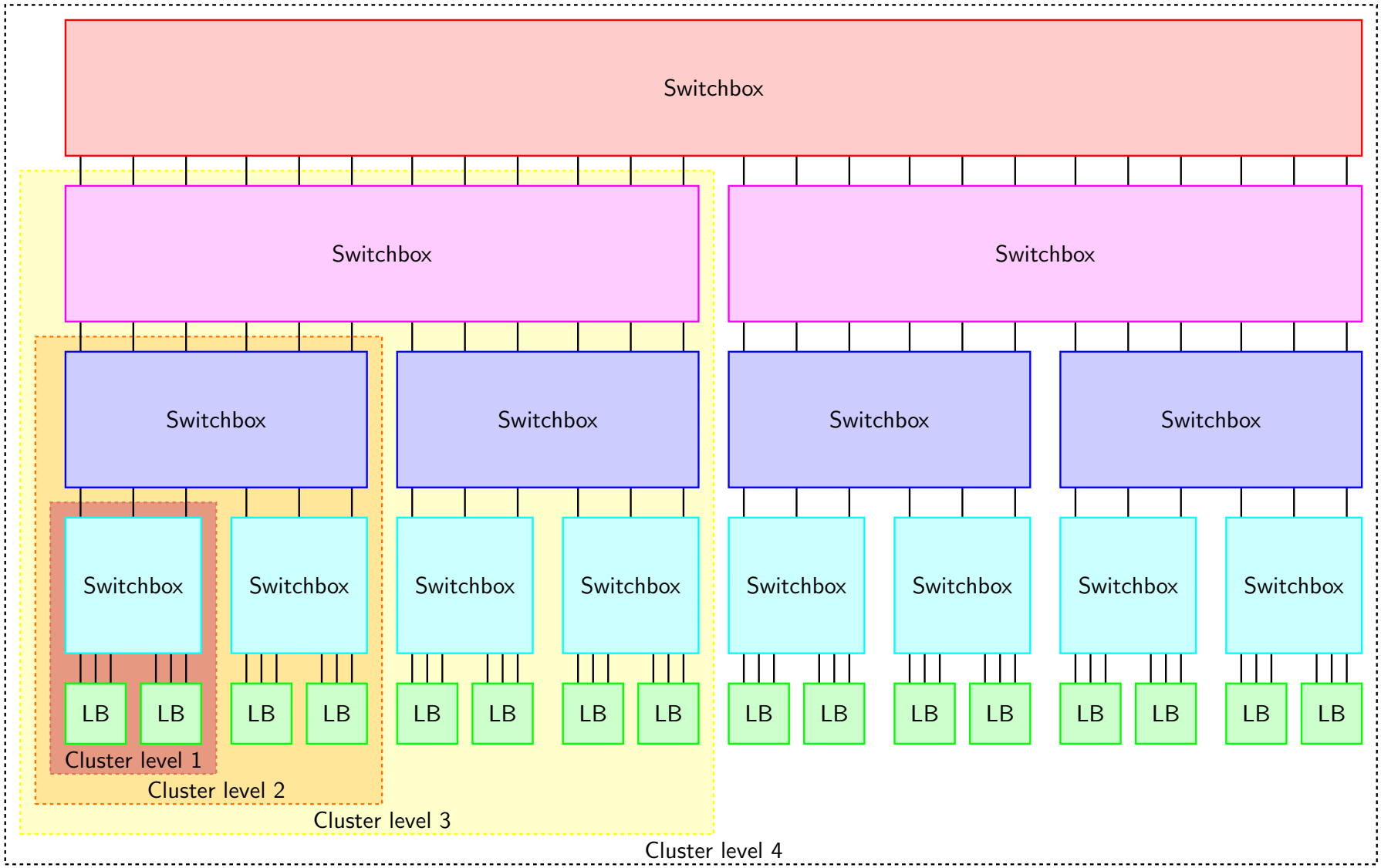
A 4-Level Hierarchical FPGA



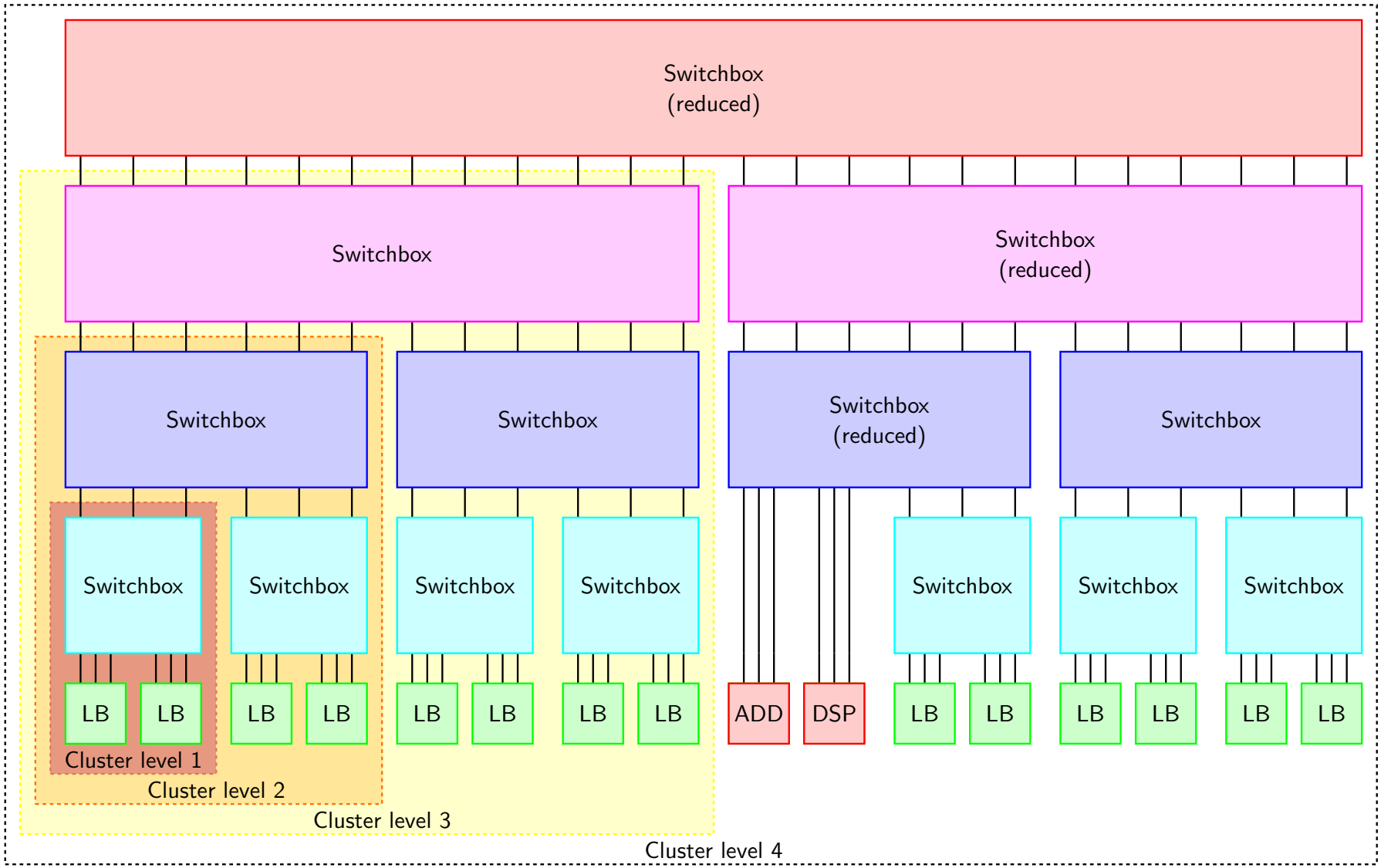
A 4-Level Hierarchical FPGA



A 4-Level Hierarchical FPGA



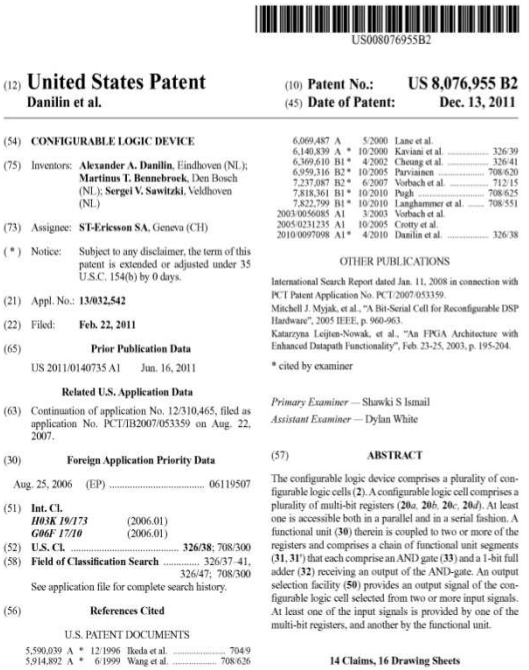
Heterogeneous Tree-Based Architecture



Advanced Space-Time Reconfigurable Architecture

Basic approach:

- keep the classical island-style architecture but
  - separate data flow from control flow
  - make logic blocks operating on words instead of single bits
  - implement global interconnect exclusively for control
  - allow data transfers to adjacent blocks only (reduces the interconnect overhead dramatically)
- implement additional registers to switch between parallel/serial computation within the block (hence "space-time reconfigurable")



US008076955B2

(12) **United States Patent**
Danilin et al.

(10) **Patent No.: US 8,076,955 B2**
(45) **Date of Patent: Dec. 13, 2011**

(54) **CONFIGURABLE LOGIC DEVICE**

(75) Inventors: **Alexander A. Danilin**, Eindhoven (NL); **Martinus T. Bennebroek**, Den Bosch (NL); **Sergei V. Sawitzki**, Veldhoven (NL)

(73) Assignee: **ST-Ericsson SA**, Geneva (CH)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **13/032,542**

(22) Filed: **Feb. 22, 2011**

(65) **Prior Publication Data**

US 2011/0140735 A1 Jun. 16, 2011

Related U.S. Application Data

(63) Continuation of application No. 12/310,465, filed as application No. PCT/IB2007/053359 on Aug. 22, 2007.

(30) **Foreign Application Priority Data**

Aug. 25, 2006 (EP) 06119507

(51) **Int. Cl.**
H03K 19/173 (2006.01)
G06F 17/10 (2006.01)

(52) **U.S. Cl.** **326/38**; 708/300

(58) **Field of Classification Search** 326/37–41, 326/47; 708/300

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,590,039	A *	12/1996	Ikeda et al.	704/9
5,914,892	A *	6/1999	Wang et al.	708/626
6,069,487	A	5/2000	Lane et al.	
6,140,839	A *	10/2000	Kaviani et al.	326/39
6,369,610	B1 *	4/2002	Cheung et al.	326/41
6,959,316	B2 *	10/2005	Parviainen	708/620
7,237,087	B2 *	6/2007	Vorbach et al.	712/15
7,818,361	B1 *	10/2010	Pugh	708/625
7,822,799	B1 *	10/2010	Langhammer et al.	708/551
2003/0056085	A1	3/2003	Vorbach et al.	
2005/0231235	A1	10/2005	Crotty et al.	
2010/0097098	A1 *	4/2010	Danilin et al.	326/38

OTHER PUBLICATIONS

International Search Report dated Jan. 11, 2008 in connection with PCT Patent Application No. PCT/2007/053359.

Mitchell J. Myjak, et al., "A Bit-Serial Cell for Reconfigurable DSP Hardware", 2005 IEEE, p. 960-963.

Katarzyna Leijten-Nowak, et al., "An FPGA Architecture with Enhanced Datapath Functionality", Feb. 23-25, 2003, p. 195-204.

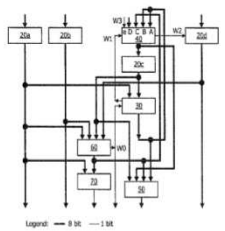
* cited by examiner

Primary Examiner — Shawki S Ismail
Assistant Examiner — Dylan White

(57) **ABSTRACT**

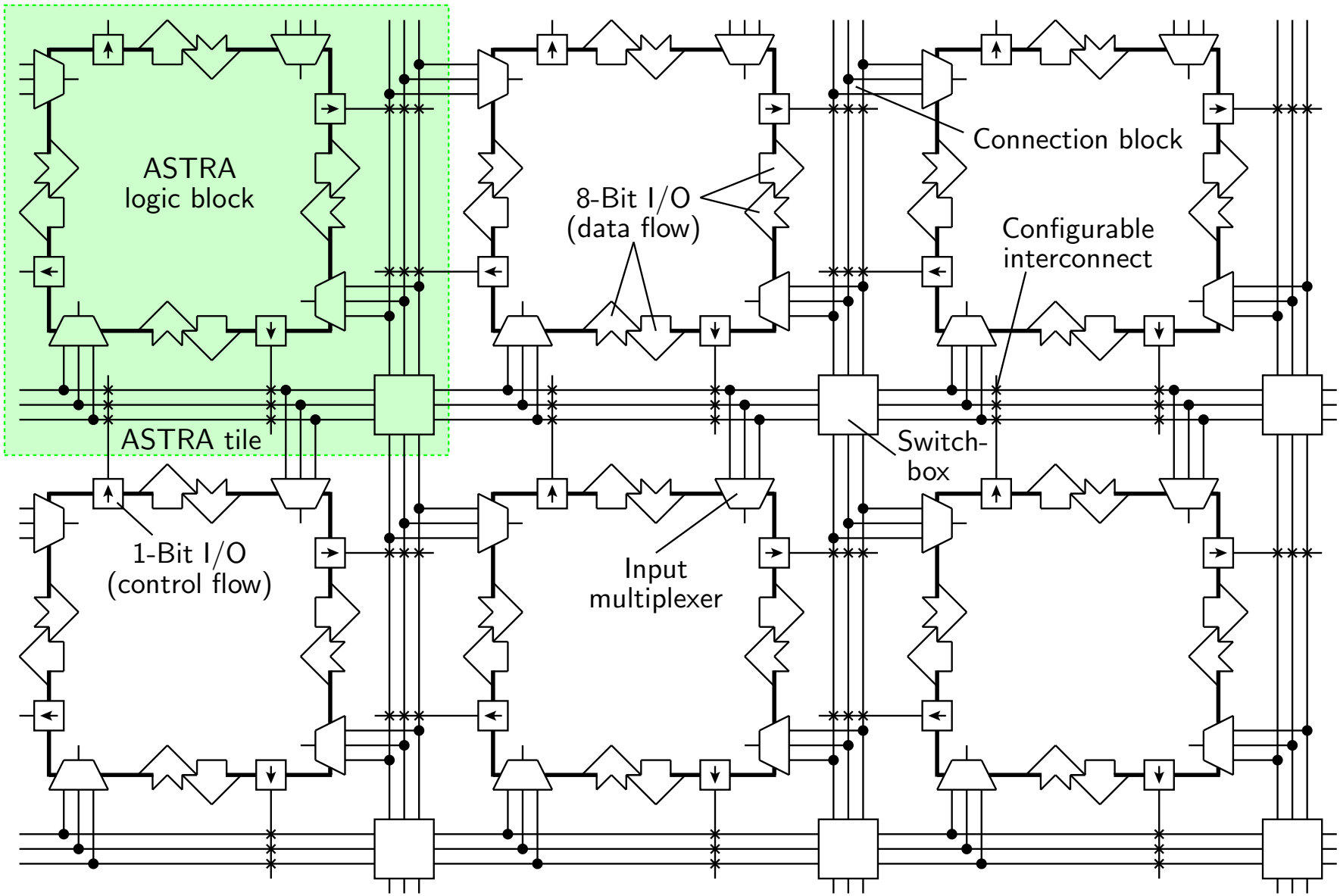
The configurable logic device comprises a plurality of configurable logic cells (**2**). A configurable logic cell comprises a plurality of multi-bit registers (**20***a*, **20***b*, **20***c*, **20***d*). At least one is accessible both in a parallel and in a serial fashion. A functional unit (**30**) therein is coupled to two or more of the registers and comprises a chain of functional unit segments (**31**, **31'**) that each comprise an AND gate (**33**) and a 1-bit full adder (**32**) receiving an output of the AND-gate. An output selection facility (**50**) provides an output signal of the configurable logic cell selected from two or more input signals. At least one of the input signals is provided by one of the multi-bit registers, and another by the functional unit.

14 Claims, 16 Drawing Sheets



➥ Danilin, Bennebroek, Sawitzki, *Configurable Logic Device*, US Patent 8,076,955 B2, 2011.

Top View


ASTRA
logic block
8-Bit I/O
(data flow)
Connection block
Configurable
interconnect
ASTRA tile
Switch-
box
1-Bit I/O
(control flow)
Input
multiplexer

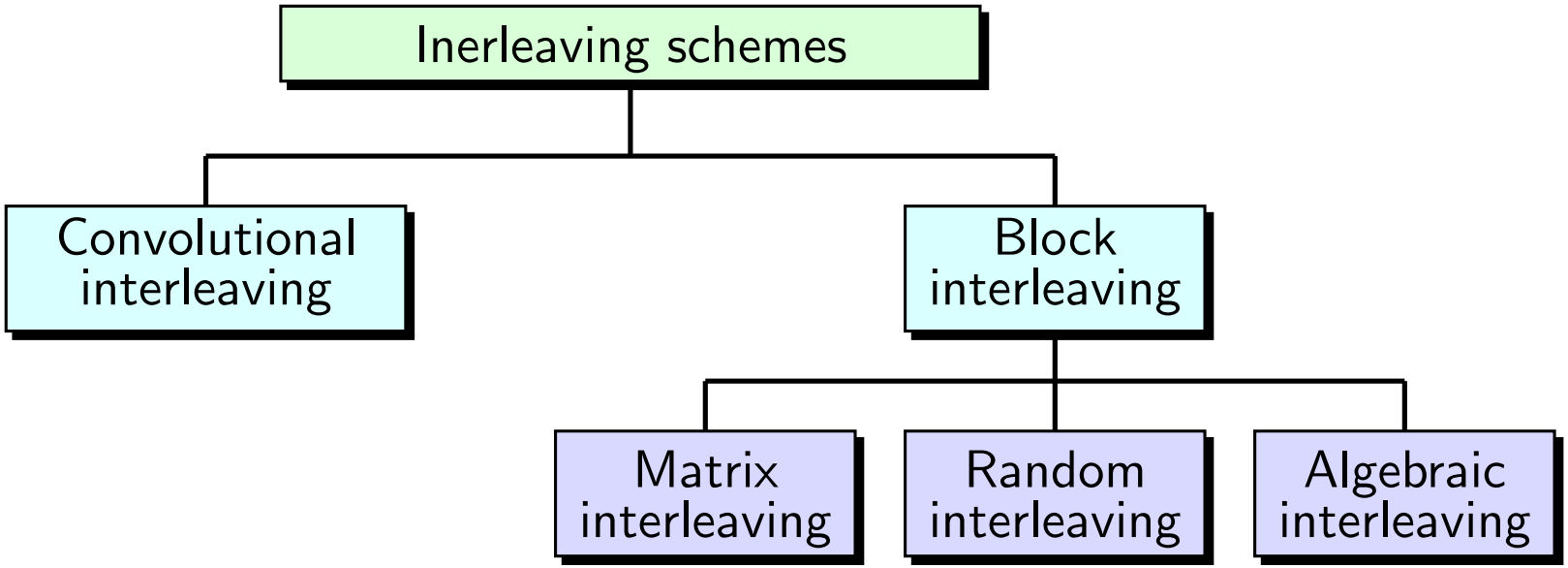
Some Benchmarks

Application	ASTRA-2, temporal		ASTRA-2, spatial		ASIC,
	Area/ mm^2	Logic blocks	Area/ mm^2	Logic blocks	spatial mm^2
VITERBI-Decoder (64 states, 4-bit input)	1,54	220	2,25	352	0,20
8-pt FFT (8-bit input)	1,27	182	3,30	470	0,13
FIR filter (16 tap, 8-bit input)	0,45	64	1,12	160	0,09

- 11–25× silicon area of an equivalent ASIC implementation (for commercial FPGAs this ratio is $\geq$ 35–40× if no special blocks are used)

3. Applications

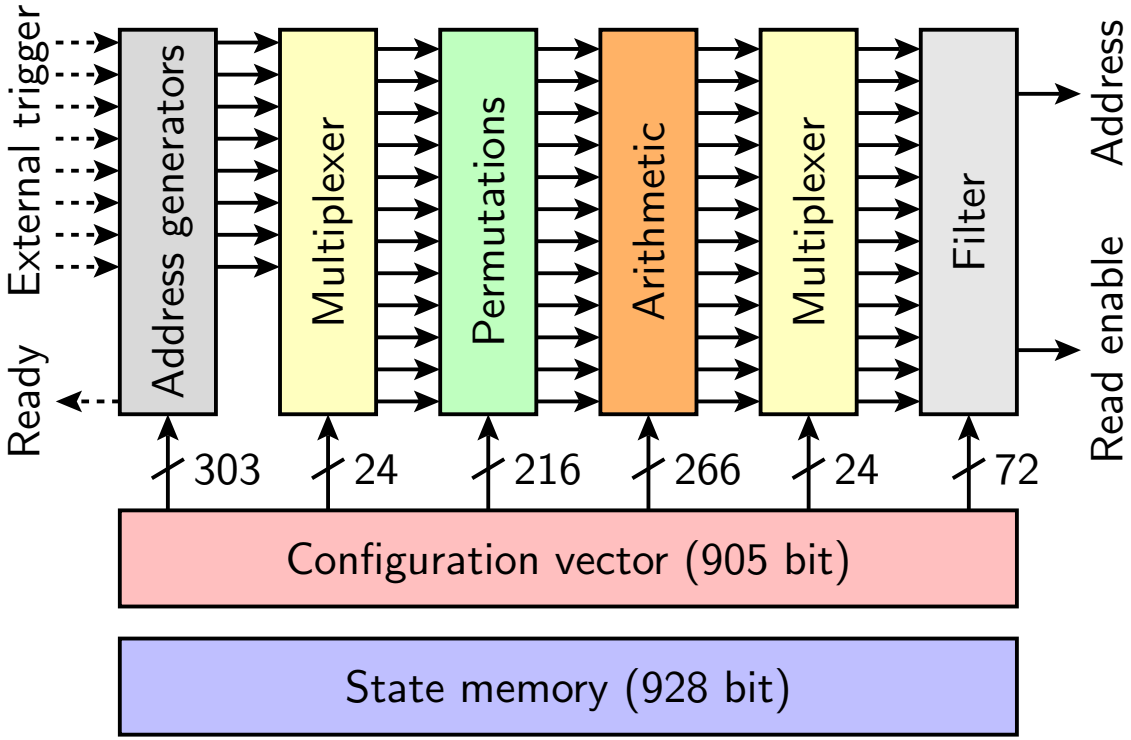
Interleaving in Digital Communication Systems



- Present in almost any relevant standard family: IEEE 802.11, DAB, DVB, LTE, ...
- Used to achieve several goals: Improve the quality of forward error correction, better use of frequency diversity, ...
- Top-view architecture: Memory which is read and written using different address sequences ➠ address generation is the key
- Most address generation schemes can be implemented using only three basic operations: permutation, transposition and bit rotation

Universal Reconfigurable (De)Interleaver

Case study for DAB, DVB, IEEE 802.11a/g and UMTS (HSDPA):



- Does not have much in common with traditional FPGA architectures, still it is (application-specific) reconfigurable computing

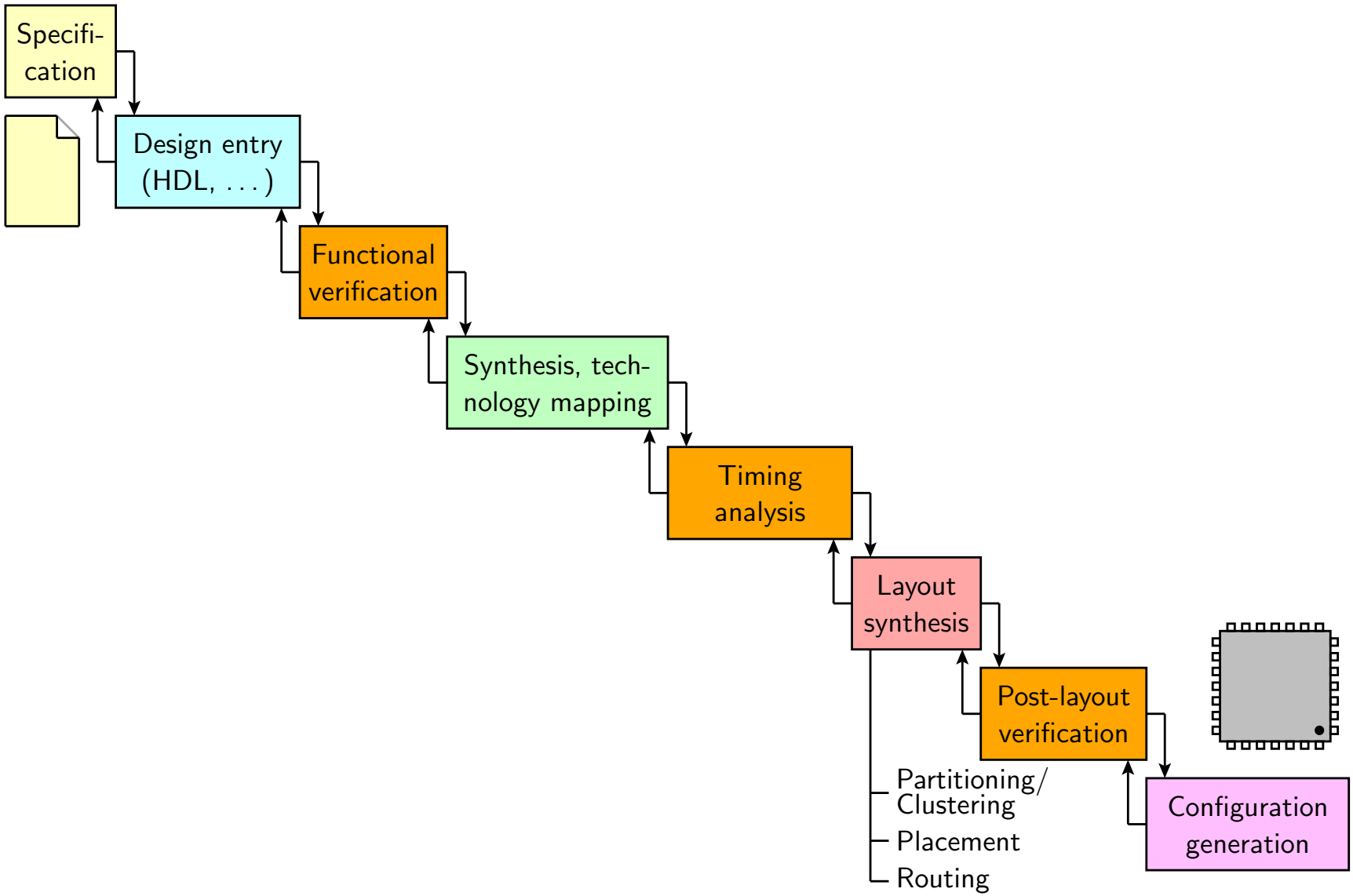
➥ Danilin, Sawitzki, Rijshouwer, *Reconfigurable Cell Architecture for Multi-Standard Interleaving and Deinterleaving in Digital Communication Systems*, FPL'2008.

The Same Study Mapped to ASTRA

Parameter	Universal deinterleaver (last slide)	ASTRA run-time reconfigurable (36 LB)	ASTRA statically reconfigurable (100 LB)
Configuration vector	905 bit	5,2 Kbit	14 Kbit
External configuration memory size (104 configurations)	94 Kbit	542 Kbit	1,5 Mbit
State register size	928 Bit	576 Bit	1,6 KBit
Area (CMOS090)	0,2 mm^2	0,2 mm^2	0,6 mm^2
Configuration loading time	5 clock cycles	variable	variable
Pipeline depth	9 stages	4 stages (variable)	4 stages (variable)

4. Tools

Conventional Design Flow



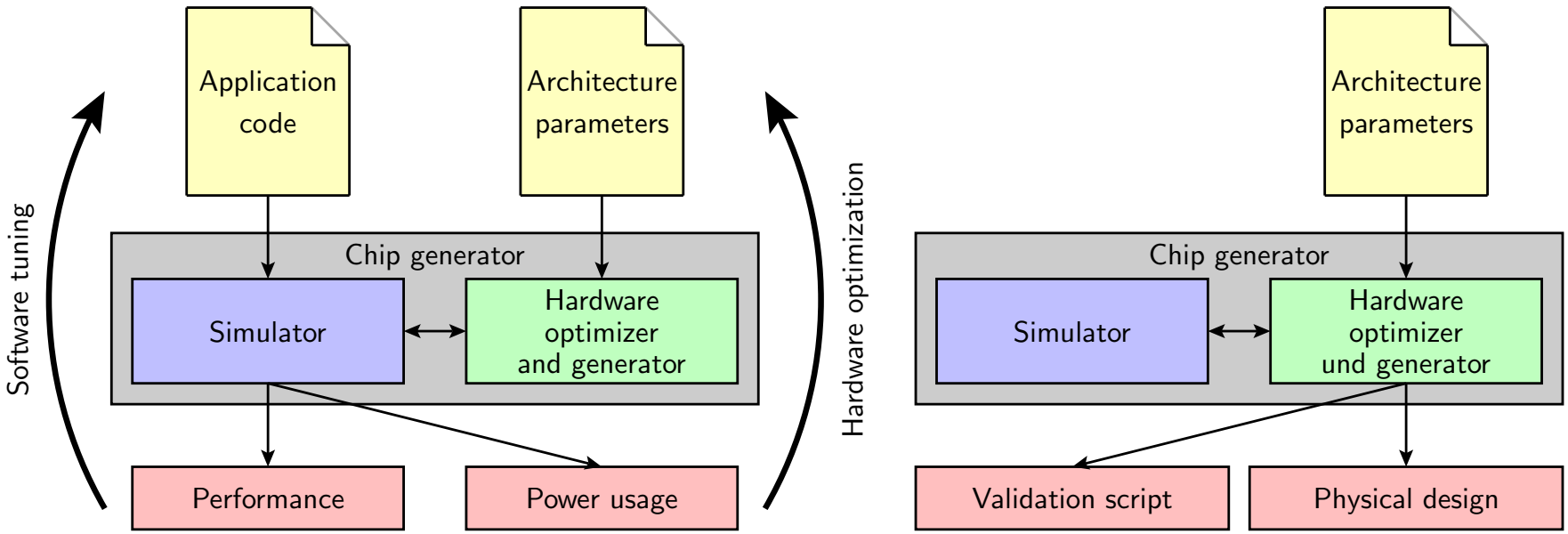
Issues with Application-Specific Design

- While front-end (functional verification, synthesis) can be kept generic, the design flow starts to be vendor-specific starting with the technology mapping step
- Traditional design flow optimizes the given application towards the underlying architecture (Intel, Xilinx, . . .)
- Application-specific approach optimizes the architecture towards the given application (domain):
  1. Start with a more or less generic architecture
  2. Map you application and check the resource utilization
  3. Remove underutilized resources, optimize congested resources
  4. Iterate if needed
  5. Once finished, proceed with a final run using the optimized architecture instance
- Steps 1–4 are called “design space exploration and tuning”, step 5 is called “instance and test generation”

➥ Template-based design

Two Phases of the Template-Based Design

. . . to be found in all following case studies!



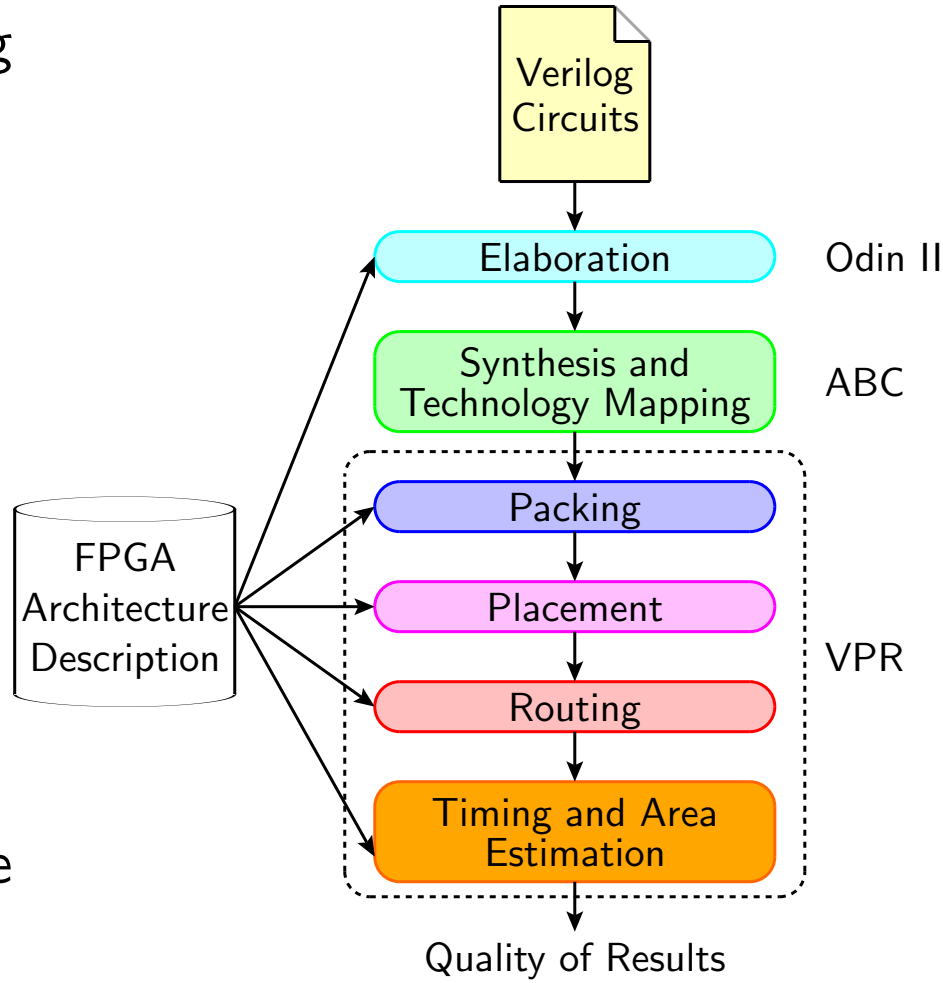
Design space exploration and tuning (Phase 1)

Instance and test generation (Phase 2)

➥ Shacham, Azizi, Wachs et al., *Rethinking Digital Design: Why Design Must Change*, IEEE Micro, Vol. 30(6), 2010

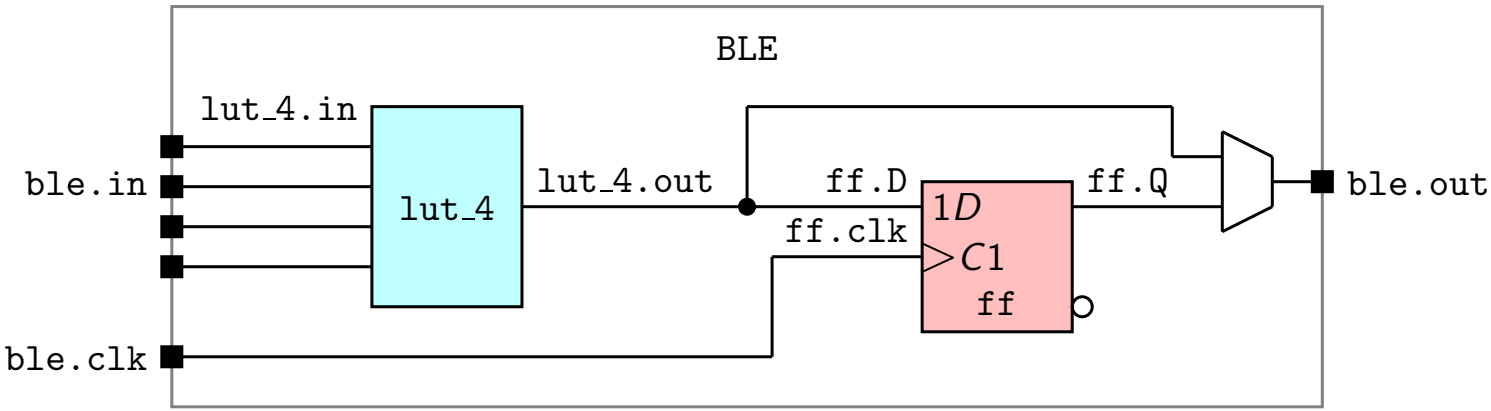
Design Flow

- open source (VTR = Verilog To Routing)
- based on standard FPGA architecture
- can handle most aspects of the modern devices
  - heterogeneous blocks
  - fracturable LUTs
  - complex logic blocks
  - special purpose cells (memories, DSP, ...)
- suitable for both architecture and algorithmic research



➥ Luu et al., *VTR 7.0: Next Generation Architecture and CAD System for FPGAs*, ACM Transactions on Reconfigurable Technology and Systems, Vol. 7(2), 2014.

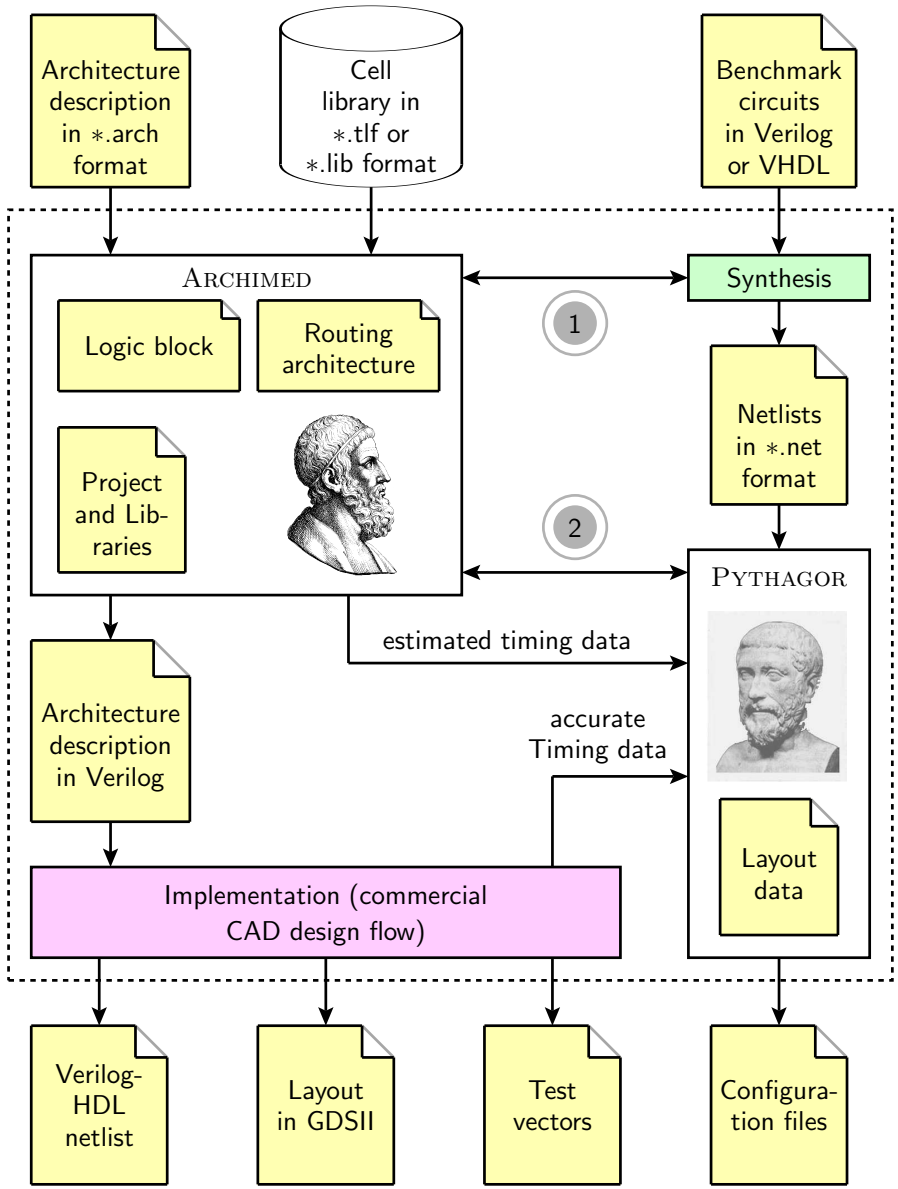
Architecture Description Example




```
<pb_type  name="ble">
  <input  name="in" num_pins="4"/>
  <output  name="out" num_pins="1"/>
  <clock  name="clk"/>
  <pb_type  name="lut_4" blif_model=".names" num_pb="1" class="lut">
    <input  name="in" num_pins="4" port_class="lut_in"/>
    <output  name="out" num_pins="1" port_class="lut_out"/>
  </pb_type>
  <pb_type  name="ff" blif_model=".latch" num_pb="1" class="flipflop">
    <input  name="D" num_pins="1" port_class="D"/>
    <output  name="Q" num_pins="1" port_class="Q"/>
    <clock  name="clk" port_class="clock"/>
  </pb_type>
  <interconnect>
    <direct  input="lut_4.out" output="ff.D"/>
    <direct  input="ble.in" output="lut_4.in"/>
    <mux  input="ff.Q lut_4.out" output="ble.out"/>
    <direct  input="ble.clk" output="ff.clk"/>
  </interconnect>
</pb_type>
```

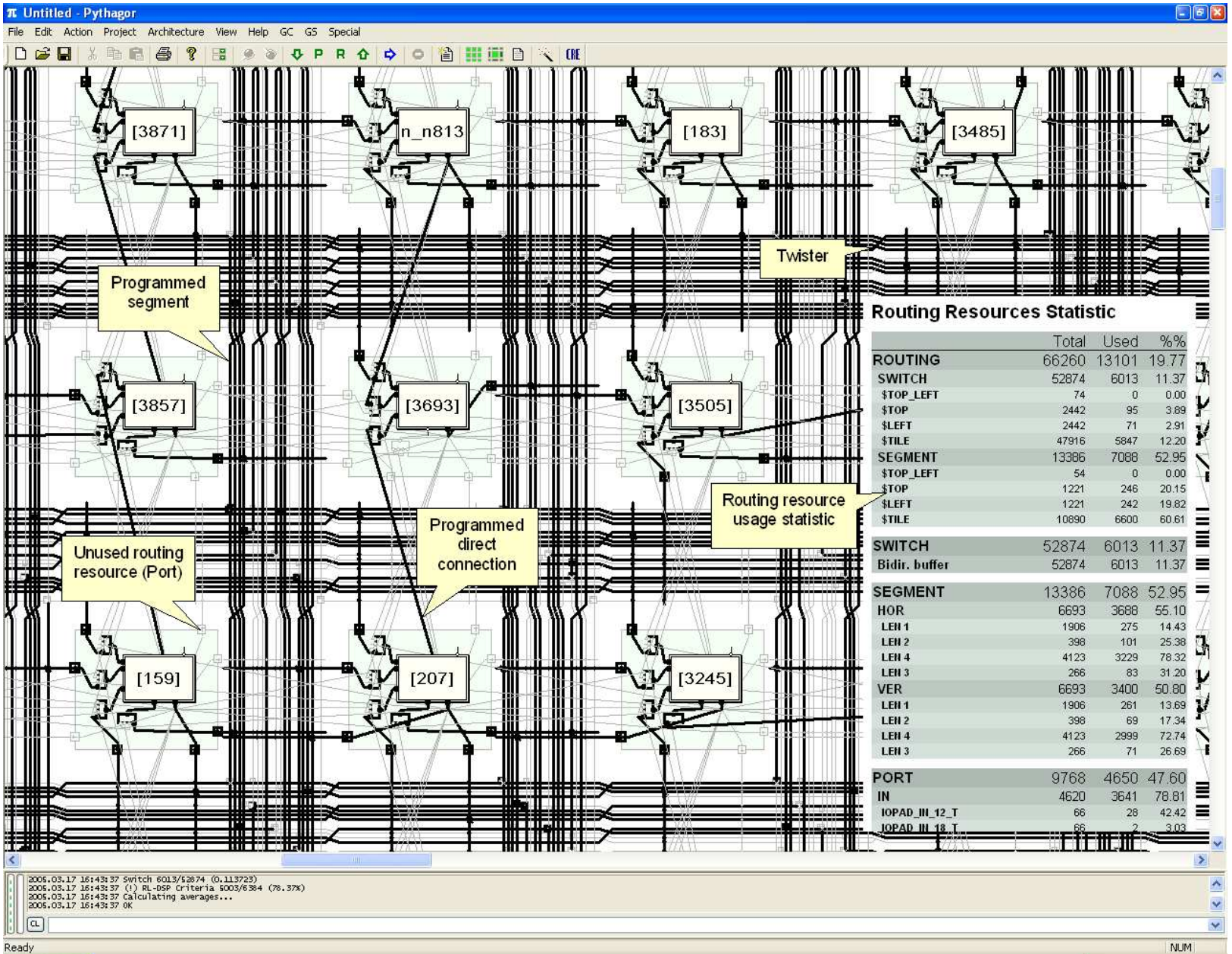
Design Flow

- former research project at Philips Research
- **not** based on standard FPGA architecture
  - any type of logic can be modeled
  - produces ready-for-manufacturing layout
  - was used for several real-world designs
  - includes test data generation
- requires external synthesis/ technology mapping tools

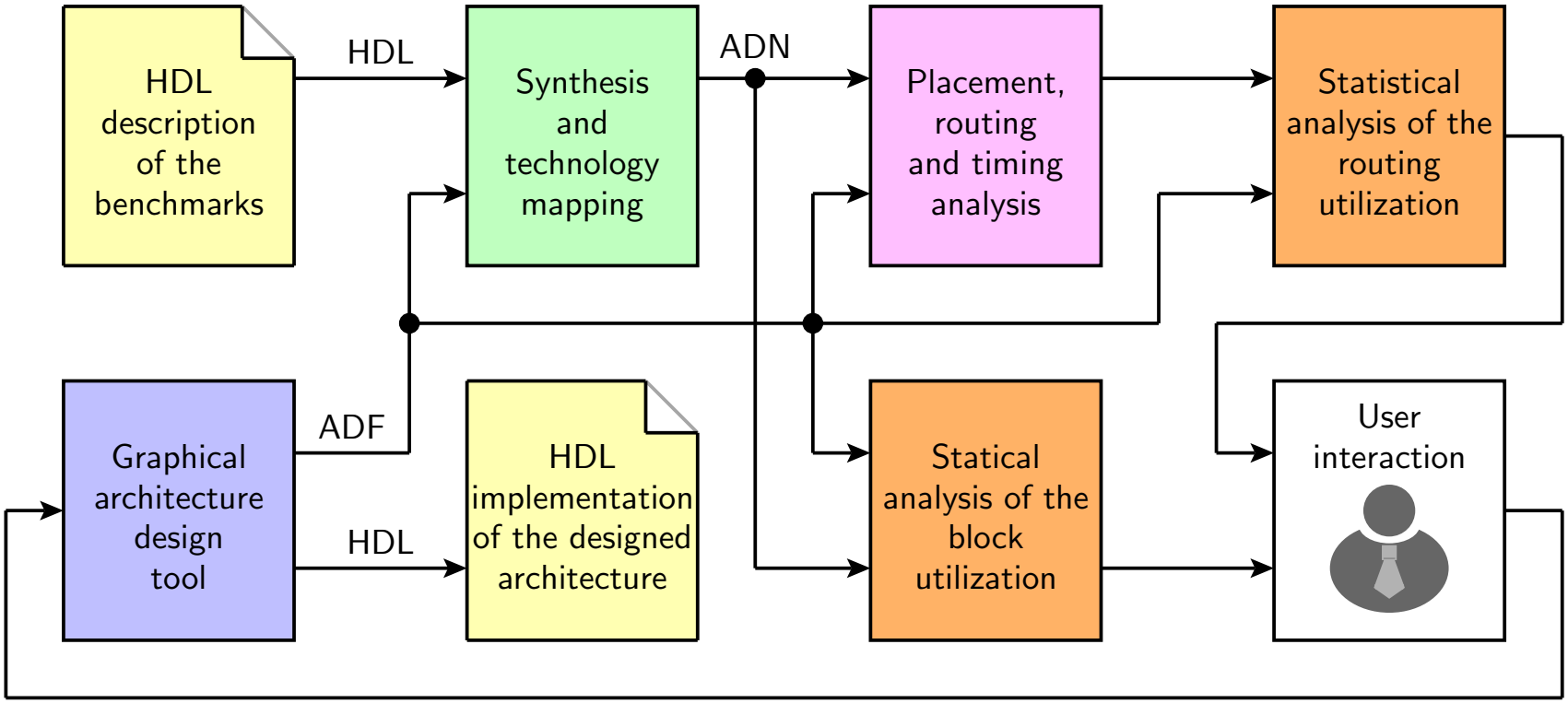


➥ Danilin, Bennebroek, Sawitzki, *A Novel Toolset for the Development of FPGA-like Reconfigurable Logic*, FPL'2005.

Pythagor in Action



Design Flow

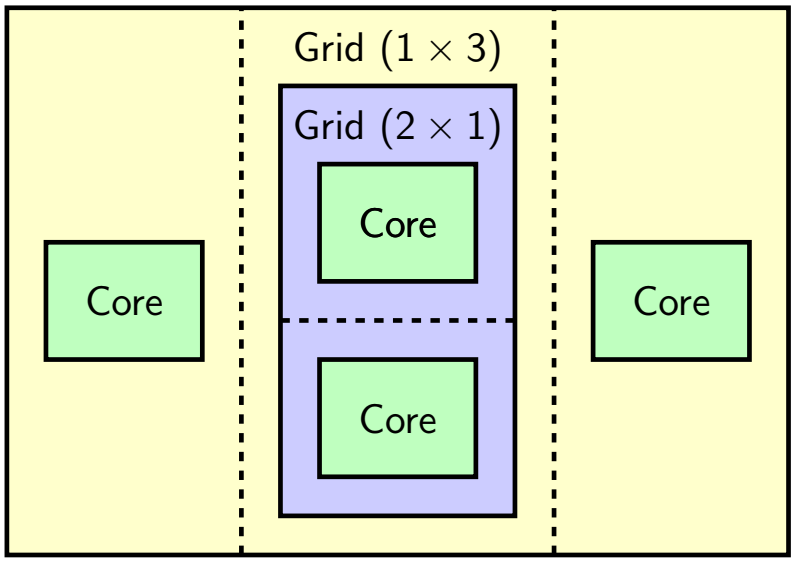


- template-based design methodology
- extremely flexible architecture template

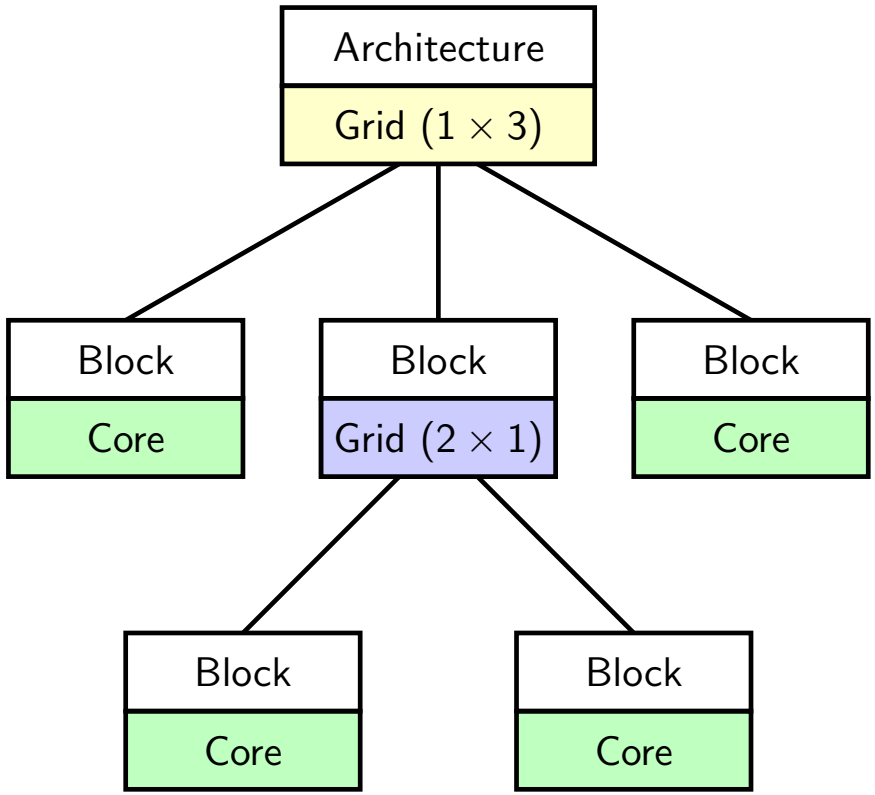
➥ Bostelmann, Sawitzki, *A Heterogeneous Architecture Template for Application Domain Specific Reconfigurable Logic*, Austrochip'2015.

Architecture Template

Grid representation

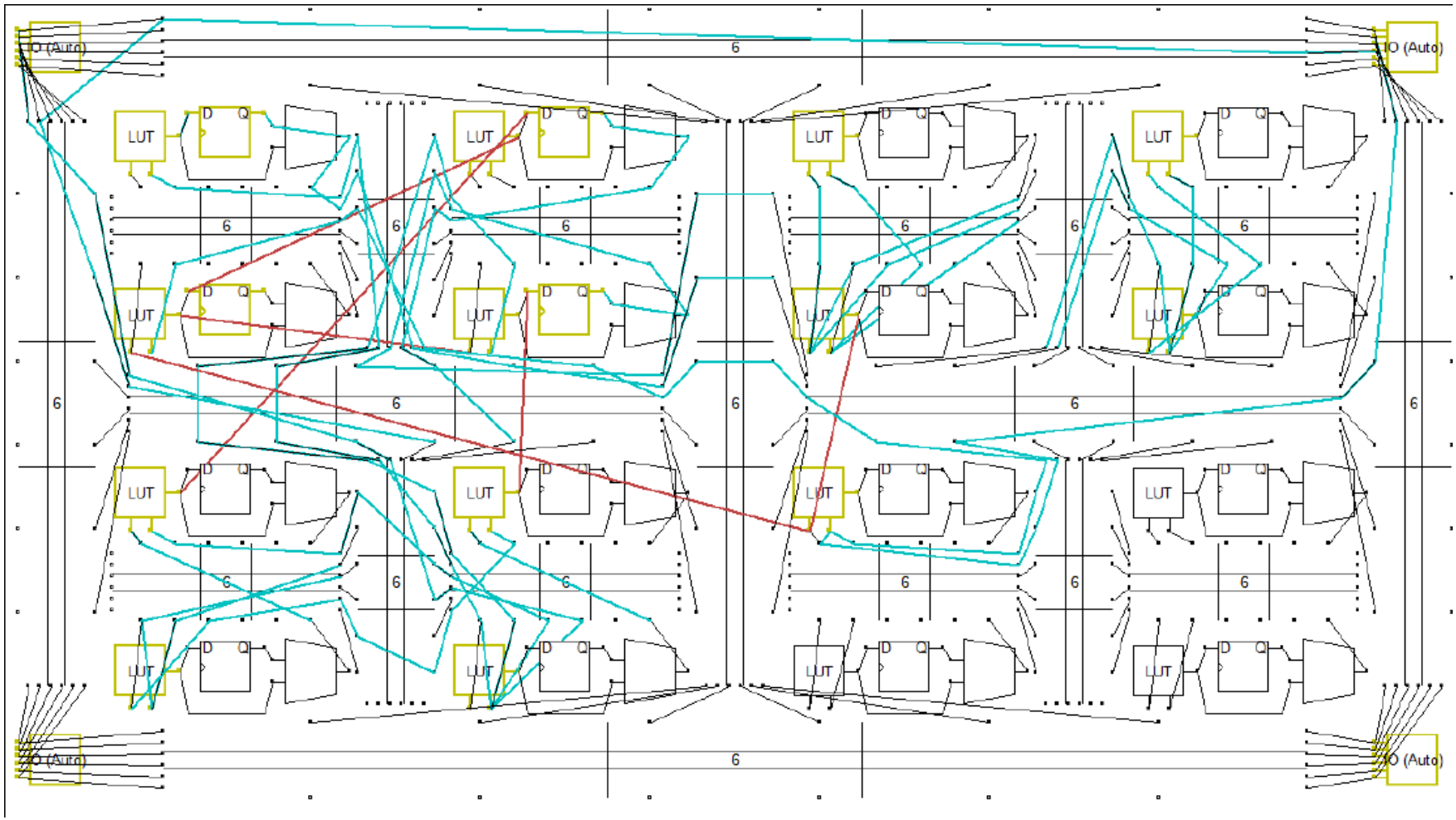


Tree representation



- Block is the only basic data structure which can be instantiated as core, grid or repeater
- Plug-in system with a simple interface (adding a new algorithm to the existing framework ➠ two methods in Python)

CustArD in Action



A placed and (partially) routed 4-bit counter circuit

5. Summary and Conclusions

To Round It Up ...

- Application-specific reconfigurable computing is a promising approach to design digital systems
  - proven by numerous studies
  - well-known in the research community, less appreciated by the industry
- Is not suitable as a replacement for the mainstream, only useful if you need to squeeze the last bit out of your reconfigurable design
- Can be a nice solution, if you do not the full flexibility of the platform-FPGA
- Stable tools and design flows are still a big issue
  - Need to optimize architecture as well as application as well as design automation algorithms!
  - Template-based design is the solution

Thank you for your attention!

Image Credits

Slide 4: Front cover of the book *"Introduction to Reconfigurable Computing"* by Christophe Bobda, copyright by Springer Science+Business Media B. V.

Slide 11: `www.intel.com`, *"Stratix IV FPGA ALM Logic Structure's 8-Input Fracturable LUT"*, copyright by Intel Corporation

Slide 12: *"UltraScale Architecture DSP Slice User Guide v1.9"*, page 14, copyright by Xilinx Inc.

Slide 13: `www.xilinx.com`, *"Zynq UltraScale+ RFSoC"*, copyright by Xilinx Inc.

Slide 19: Front cover of the book *"Tree-based Heterogeneous FPGA Architectures"* by Umer Farooq, Zied Marrakchi and Habib Mehrez, copyright by Springer Science+Business Media New York