Institut für Technische Informatik
Chair for Embedded Systems - Prof. Dr. J. Henkel

Vorlesung im SS 2016

Reconfigurable and Adaptive Systems (RAS)

Marvin Damschen, Lars Bauer, Jörg Henkel

Institut für Technische Informatik
Chair for Embedded Systems - Prof. Dr. J. Henkel

Reconfigurable and Adaptive Systems (RAS)

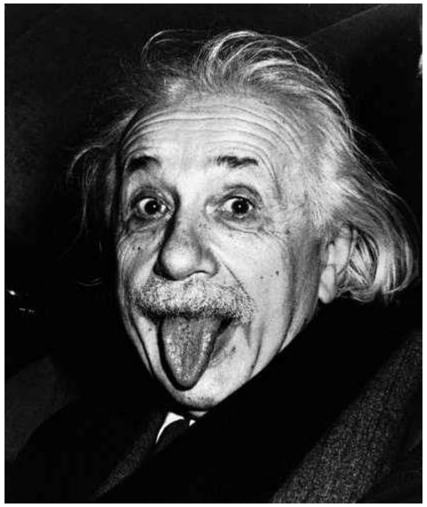
2. Overview and Definitions

Definition: *'Reconfigurable System'*

- "Computing System that can (partially) change the functionality of its hardware"
- This definition implies the use of so-called Reconfigurable Hardware
- Other definitions exist (e.g. changing the software, changing the task mapping & task scheduling etc.) but in the scope of this lecture we will focus on those approaches that use reconfigurable hardware

Definition: *'Time'*

- Here, we do not mean 'time' as a physical unit or a continuous flow
  - Rather, in the following we need to distinguish three distinct points in time
- Design time: The system is specified, the architecture is designed and the IC is taped out and sold to customers
- Compile time: Software (e.g. application) is compiled for the design-time fixed IC; it can be simulated, profiled etc.
- Run time: The application executes on the IC and faces varying situations (input data, other applications etc.)
  - Startup time: when the system boots, application starts etc.


src: Einstein, wissen.de

Definition: *'Adaptive System'*

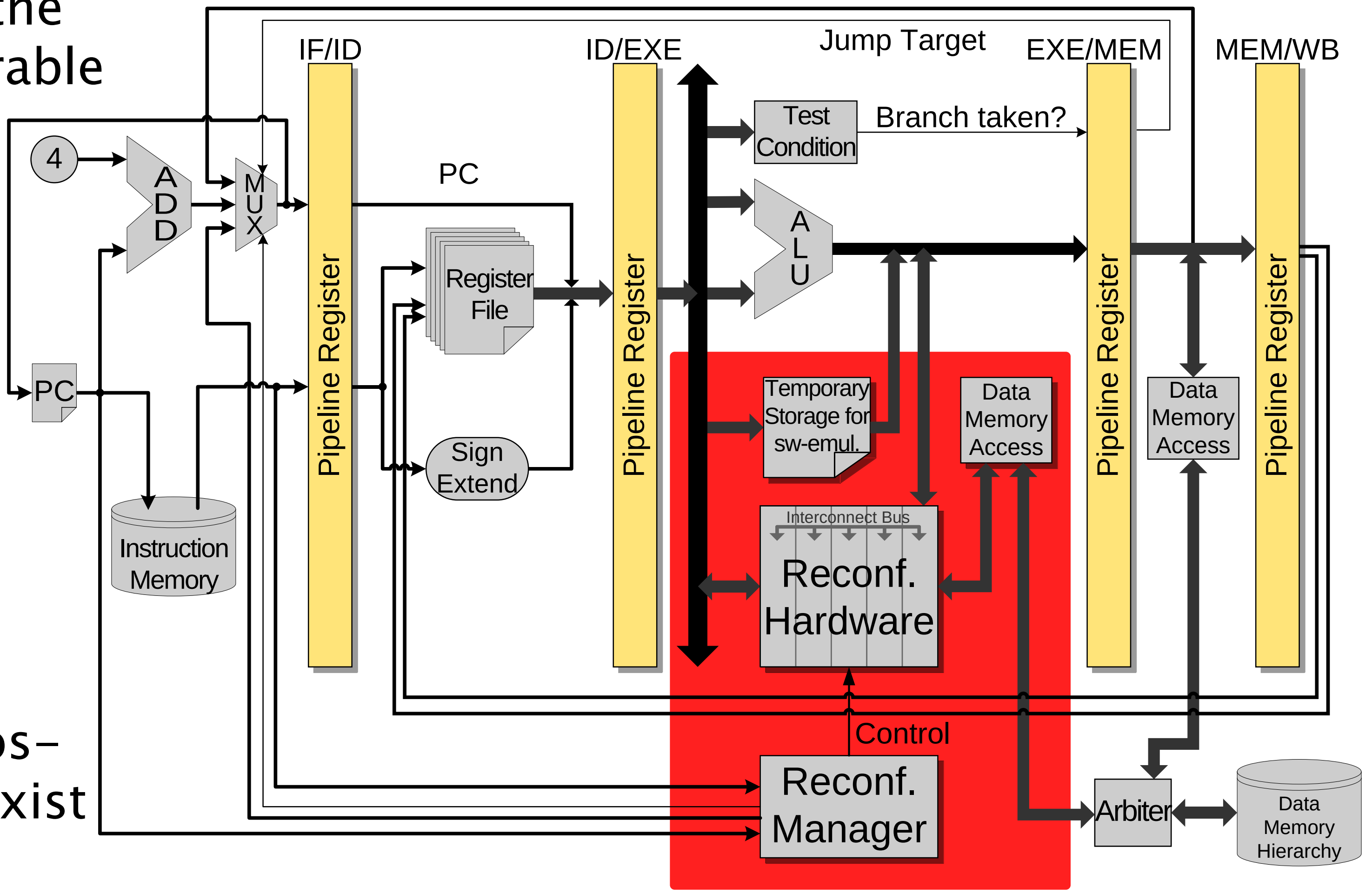
- "Adaptive computing refers to the capability of a computing system to autonomously adapt one or more of its properties (e.g. performance) during run time."
- Reconfigurable Hardware is one of the key paradigms that enable Adaptive Systems
- Not all reconfigurable systems are adaptive
  - they don't need to perform run-time reconfiguration
  - or they might only perform compile-time predetermined run-time reconfigurations
- Not all adaptive systems rely on reconfigurable hardware (e.g. they might use clever software or OS/middleware to adapt their properties)

Definition: *'Application Scenario'*

- An description of the particular work load of the system for a particular time
- Which tasks are executing?
- How do these tasks depend on each other?
  - Data dependencies in a task graph
  - Resource conflicts, e.g. cache or periphery
- What are the deadlines for the tasks?
- What are the priorities for the tasks?
- What is the input data for the tasks?
- What are the requirements of the tasks (computational power, energy consumption, demand for hardware accelerators etc.)

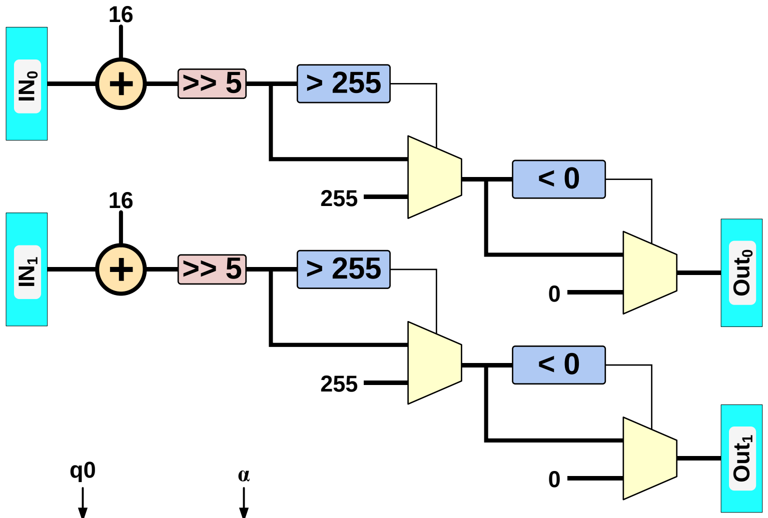
Example for Using Reconfigurable Hardware to Accelerate Applications

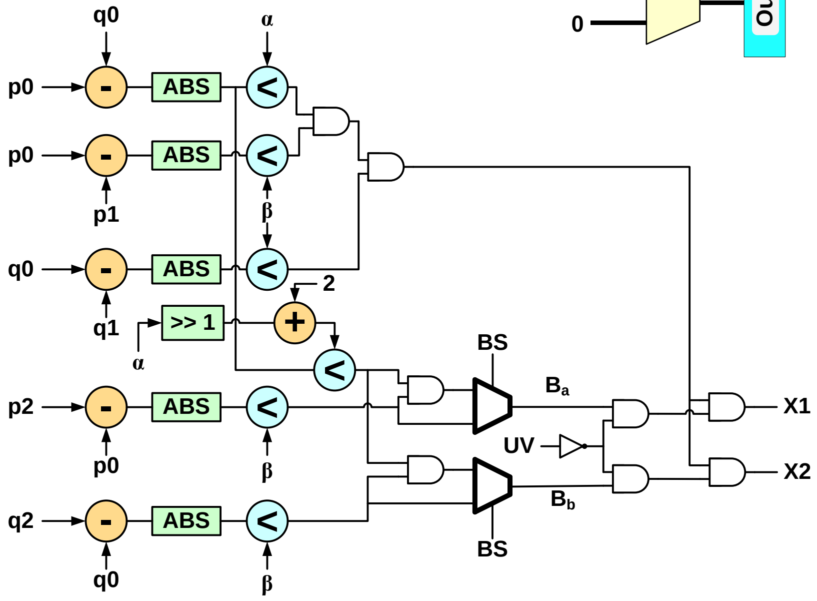
- Integrate the reconfigurable HW into the pipe-line of a pro-cessor
- Use it as a reconfi-gurable functional unit (RFU)
- Further pos-sibilities exist



Examples for Accelerators

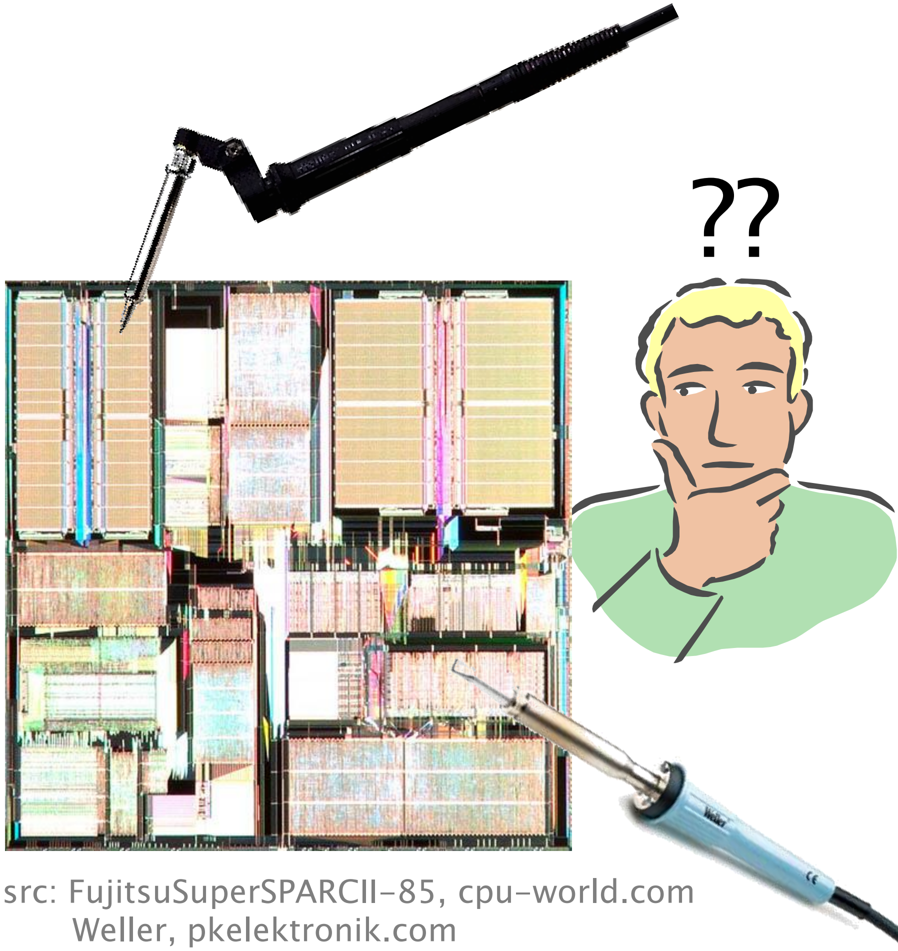
- The reconf. hardware can be used to implement application specific accelerators on demand
- The accelerators exploit:
  - parallelism (multiple independent operations are executed at the same time in parallel) and
  - operator chaining (multiple data-dependent operations are executed right after each other in the same cycle) to achieve speedup





Open Issue: How to perform hardware reconfiguration?

- It is hardly possible to physically change the transistors (N-P doping etc.) and the metal layers after fabrication
- Changing them fast (for run-time reconfiguration) and in a meaningful way can be considered impossible
- So, that's it??

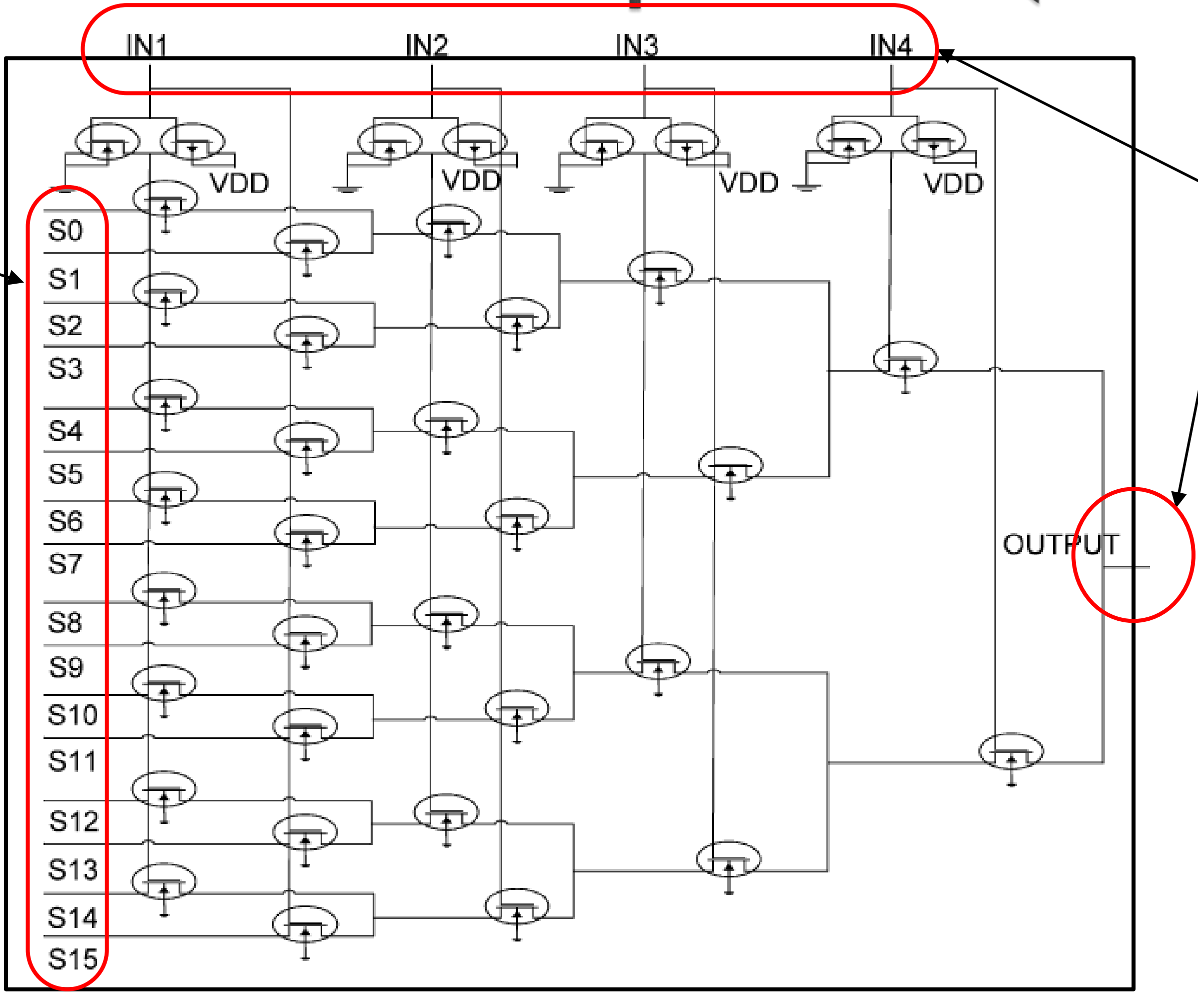


src: FujitsuSuperSPARCII-85, cpu-world.com
Weller, pkelektronik.com

Fine-grained Reconfigurable Hardware: Look-up Table (LUT)

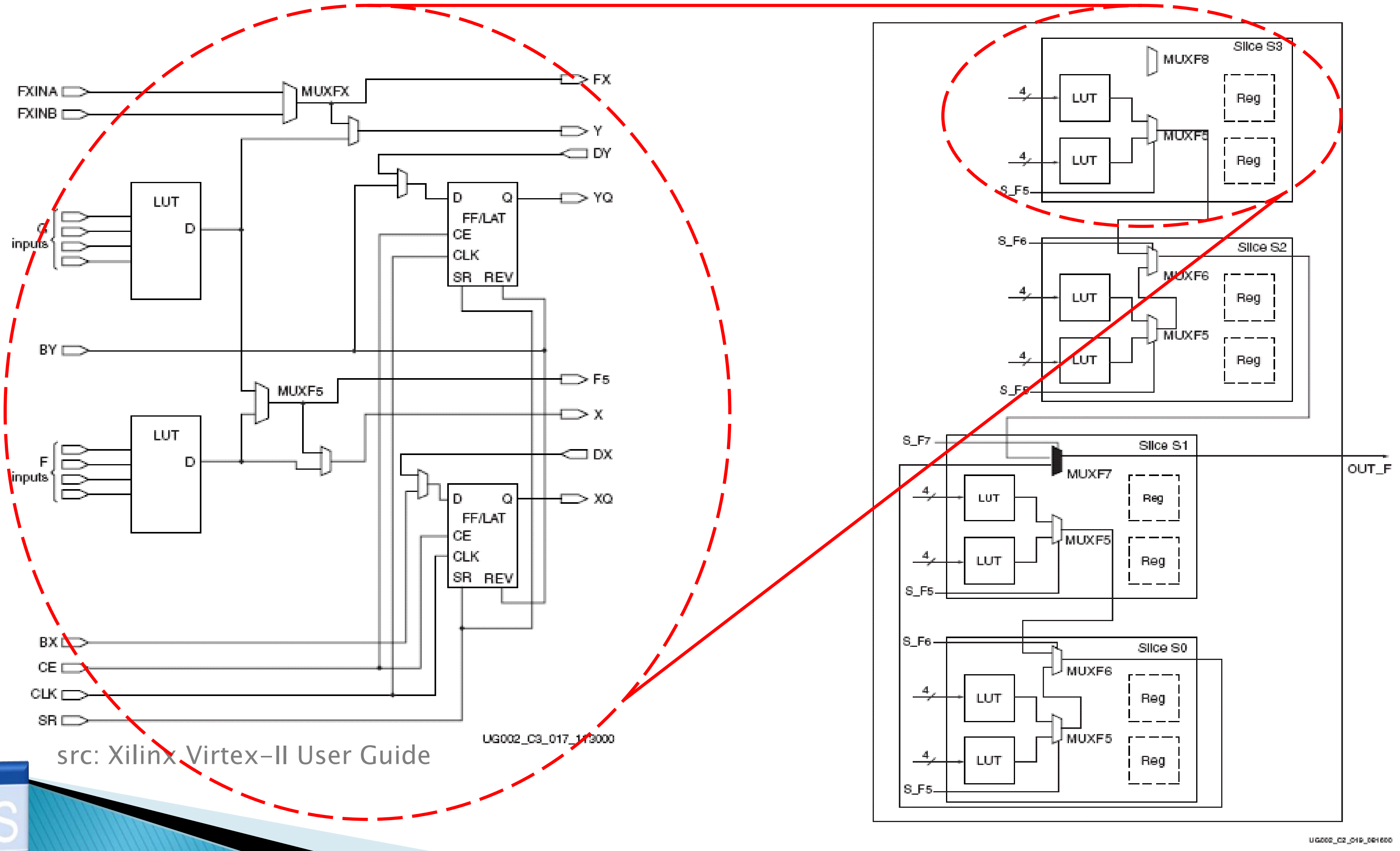
Configura-
tion Data

User I/O



src: Kalenteridis et al. “A complete platform and toolset for system implementation on fine-grained reconfigurable hardware“, Microprocessors and Microsystems 2004

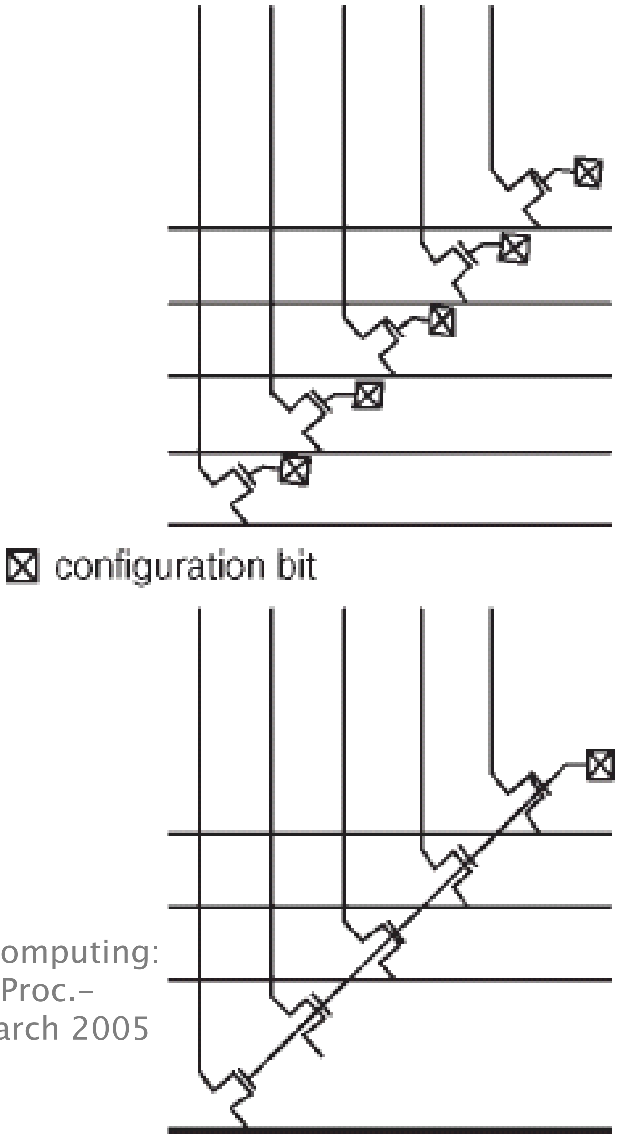
Building Larger Reconfigurable Blocks, so-called Slices and CLBs



src: Xilinx Virtex-II User Guide

Reconfigurable Connections

- Two crossing lines are either connected or not
  - Control Bit decides
- Fine Grained: Each bit line can be configured independently
- Coarse Grained: Multiple bit lines (bus) together



src: T.J. Todman et al.: "Reconfigurable computing: architectures and design methods", IEEE Proc.-Comput. Digit. Tech., Vol. 152, No. 2, March 2005

Array of reconfigurable logic gates

CLB: Configurable
Logic Block

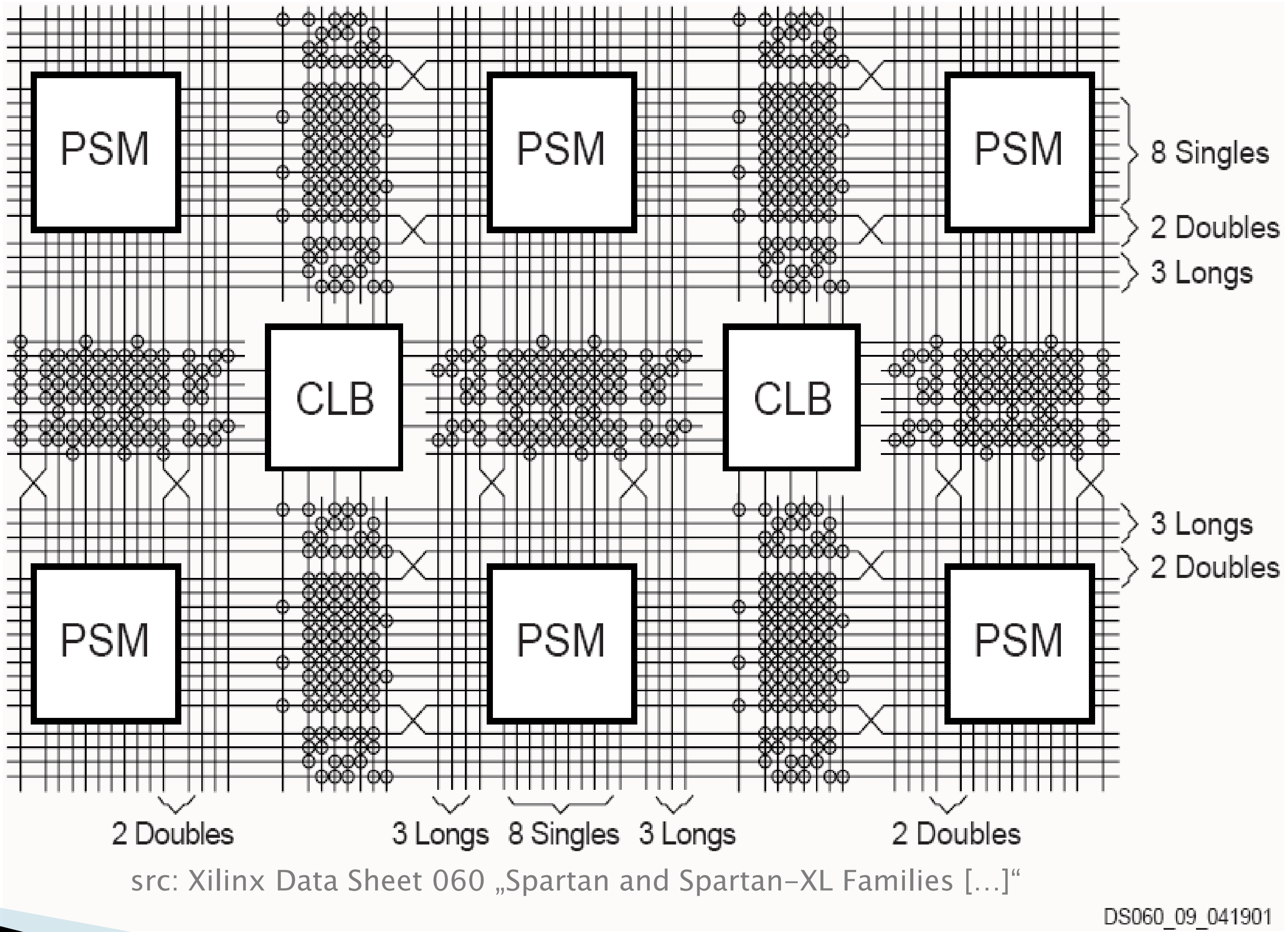
PSM: Programmable
Switch Matrix

Additionally:
I/O Blocks,
RAM Blocks,
Multiplier,
CPUs, …

Virtex-II 6000:
96x88 CLBs
→8.448 CLBs
→67.584 LUTs

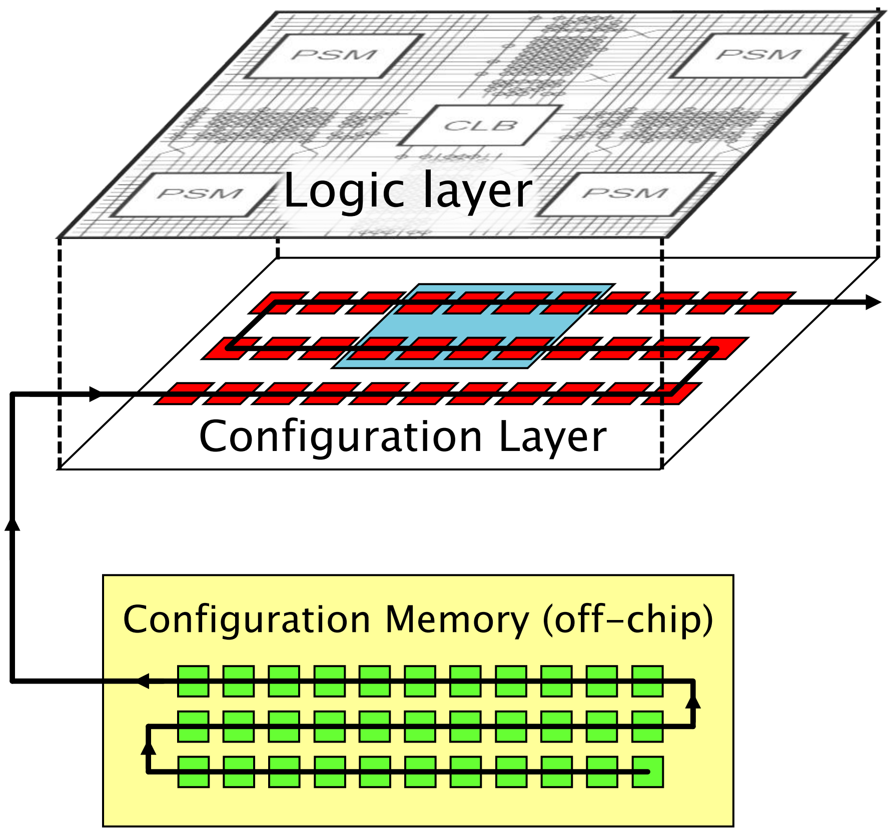
Virtex 4 LX 160:
192x88 CLBs
→16.896 CLBs
→135.168 LUTs

Virtex 7 2000T:
→ 1.221.600 LUTs



src: Xilinx Data Sheet 060 „Spartan and Spartan-XL Families […]"

Partial Run-time Reconfiguration



- Logic Layer: perform the actual computation
- Configuration Layer: determine the kind of computation that shall be performed
  - Is typically configured from external memory
  - May also provide some con-figuration cache inside the FPGA
- May allow reconfiguration of parts of the area
  → partial reconfiguration
  - This allows placing a logic inside the FPGA that recon-figures another part of the FPGA
    → Self-reconfiguration

Internal Configuration memory

- PROM based (Fuse, Anti-Fuse)
  - — Only writeable one time
- (E)EPROM/Flash based (Floating-Gate)
  - + Non-volatile → immediately configured after boot up
  - + Configuration data not (necessarily) readable outside the FPGA → Security; Intellectual Property (IP) protection
  - + Low power consumption
  - — Limited re-writeability (i.e. only good for a limited number of reconfigurations)
  - — Slow write access → not suitable for run-time reconfiguration / self-reconfiguration

Internal Configuration memory (cont'd)

- SRAM based
  - + Allows arbitrary number of reconfigurations
    → good for prototyping
  - + Fast reconfiguration
    → Allows for run-time reconfiguration and self-reconfiguration
  - — Needs to be reconfigured after every boot up
    → high power consumption
    → Security problem, as everyone can observe the configuration data (possible solution: bitstream encryption)
- Hybrid (both EEPROM and SRAM on the die / in the package)
  - + Allows fast run-time reconfiguration (SRAM) and does not need external configuration data after boot up (automatically copying EEPROM to SRAM)
  - — Still high power consumption during boot up
  - — Needs larger chip area

Reconfiguration Time

- Def.: '*Bitstream*': configuration data that is copied to the configuration layer
- Def.: '*Partial Bitstream*' / '*Full Bitstream*': a Bitstream that configures 'only certain parts of' / 'the entire' FPGA
- A Bitstreams can become rather large:
  - Full Bitstream depends on the FPGA, e.g. 2-20 MB for Virtex-6
  - Partial Bitstream depend on the design, e.g. 100 KB – 1 MB
- Definition '*Reconfiguration Bandwidth*': the average bandwidth to copy the Bitstream from the external memory to the Configuration Layer (MB/s)
  - Virtex-II was specified for 50 MB/s and was demonstrated to work at 100 MB/s
  - More recent FPGAs allow faster reconf. bandwidths (e.g. 32 bit @ 100 MHz = 400 MB/s), but memory may become the bottleneck

Reconfiguration Time (cont’d)

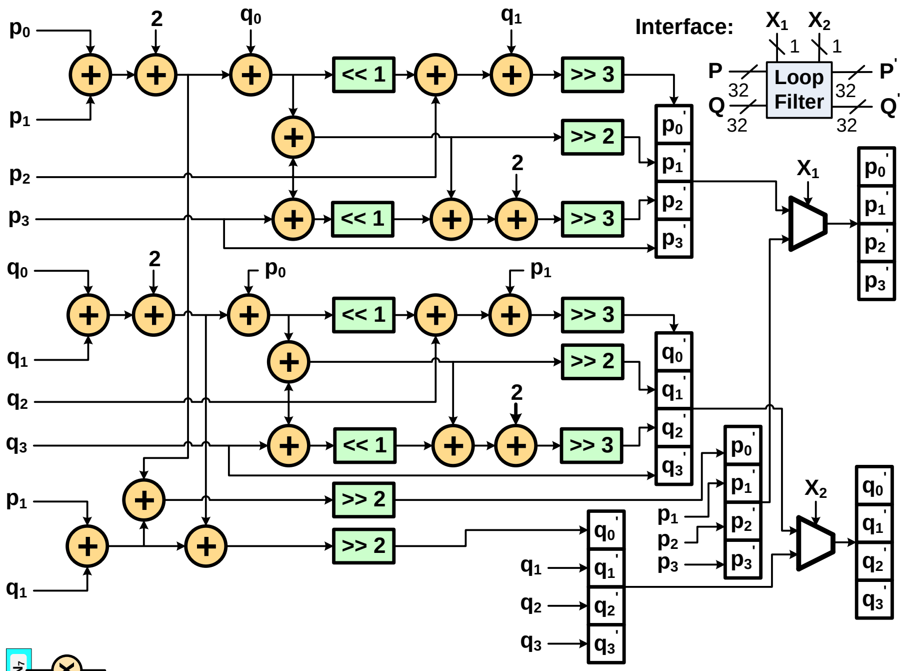
- Practically, the bandwidth is limited by the external memory
  - In CES demonstrator for RISPP project we used external EEPROM that provides on avg. 36 MB/s
  - Alternatively, the system DDR RAM might be used to store the partial Bitstreams
    - Reduces the system’s memory performance during reconfiguration

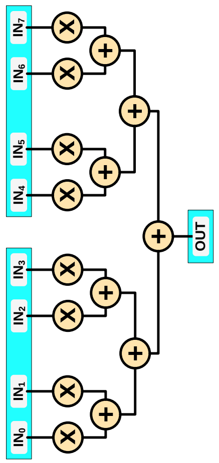
Reconfiguration Time (cont'd)

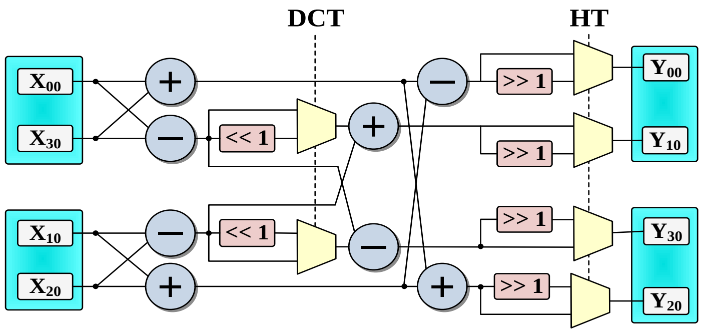
- Resulting Reconfiguration time
  - Typically 1 ms – 10 ms if fast configuration ports are used
    - Note: 1 MB/s corresponds to 1 KB/ms
  - 100 KB @ 100 MB/s → 1 ms
  - 1 MB @ 200 MB/s → 5 ms
  - In CES demonstrator typically 30–40 KB @ 36 MB/s → 0.8 – 1.1 ms
- How long is 1ms?
  - 100,000 cycles of a 100 MHz CPU
  - 1 million cycles of a 1 GHz CPU
  - Task switch time (time slice) in Linux : ~10 ms (depends on task scheduler, kernel version etc.; to some degree configurable; could be 1ms, could be 100ms)
    - → it's a rather long time for a CPU

Reconfiguration Time (cont'd)

- The rather slow reconfi-guration time is due to the large amount of configu-ration data
- For instance, the examples on the right demand 8-16 bit Adds, Subs & Mults
- Many LUTs need to be configured and connected to implement an Adder etc.
  - This leads to rather large partial bitstreams
  - It also affects the area requirements and the maximal frequency

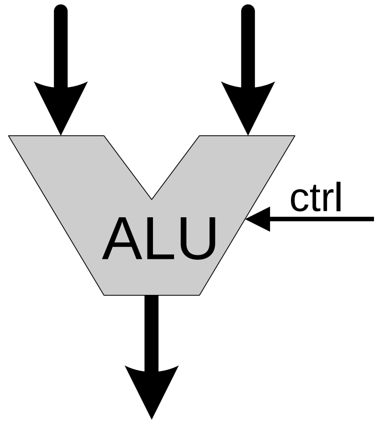


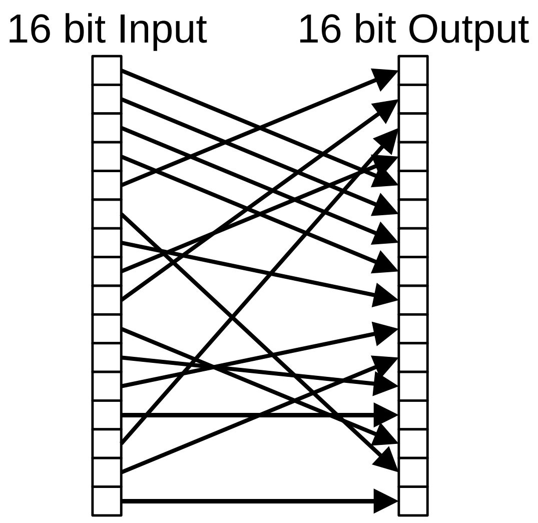




Alternative for LUTs: Arithmetic and Logic Units

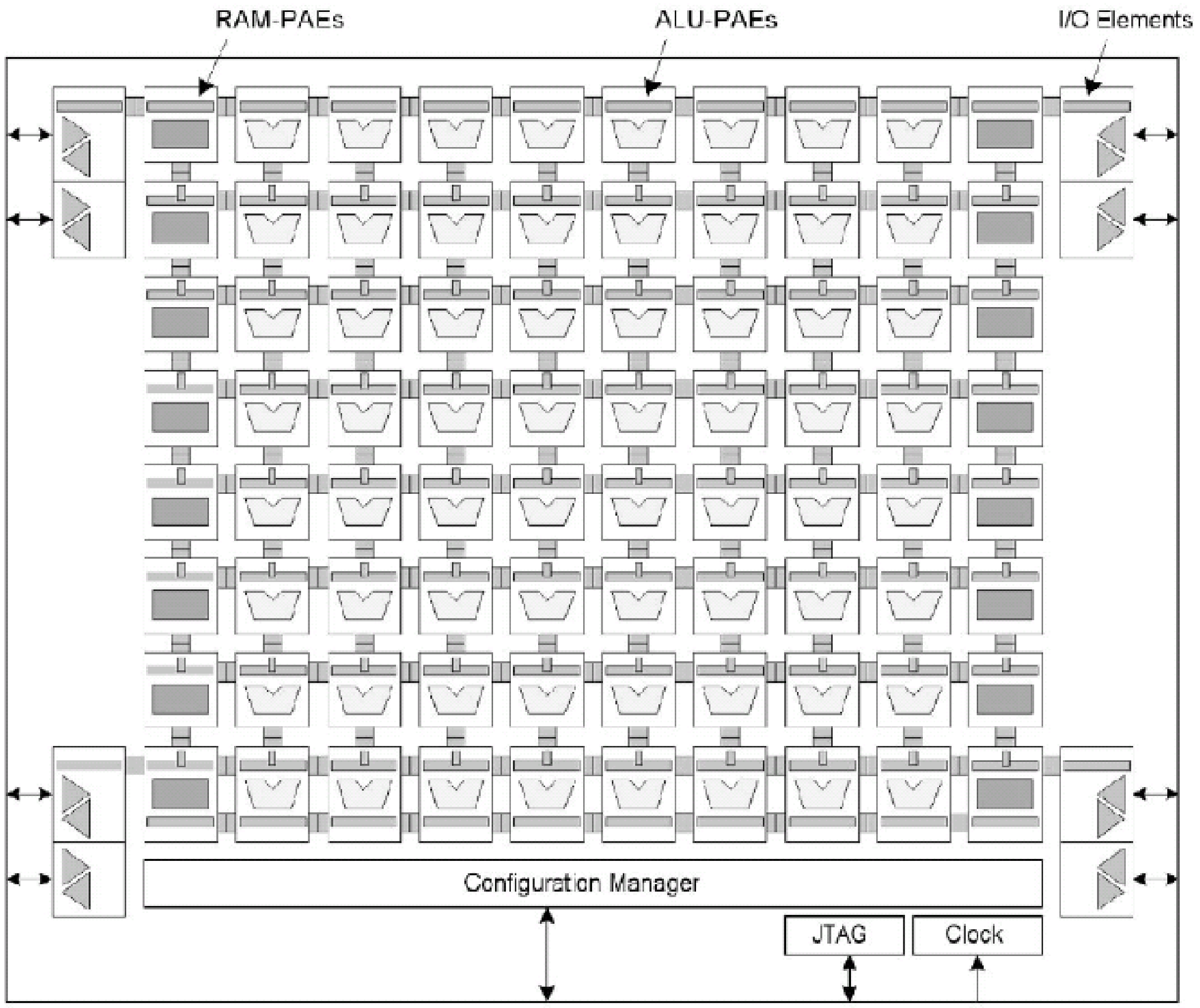
- If multiple arithmetic and/or logical operations need to be performed, then an ALU might outperform LUTs:
- Differences:
  - + Significantly less configuration data
  - + Smaller area footprint (for the operations it implements)
  - + Higher Frequency
  - — Reduced efficiency when facing non-arithmetic operations or bit-level operations (resulting in increased area requirements and/or increased latency)
    - E.g. bit shuffling: how many cycles are needed to perform the operation shown on the right side with one ALU? Or: How many ALUs are needed to pipeline the operation?





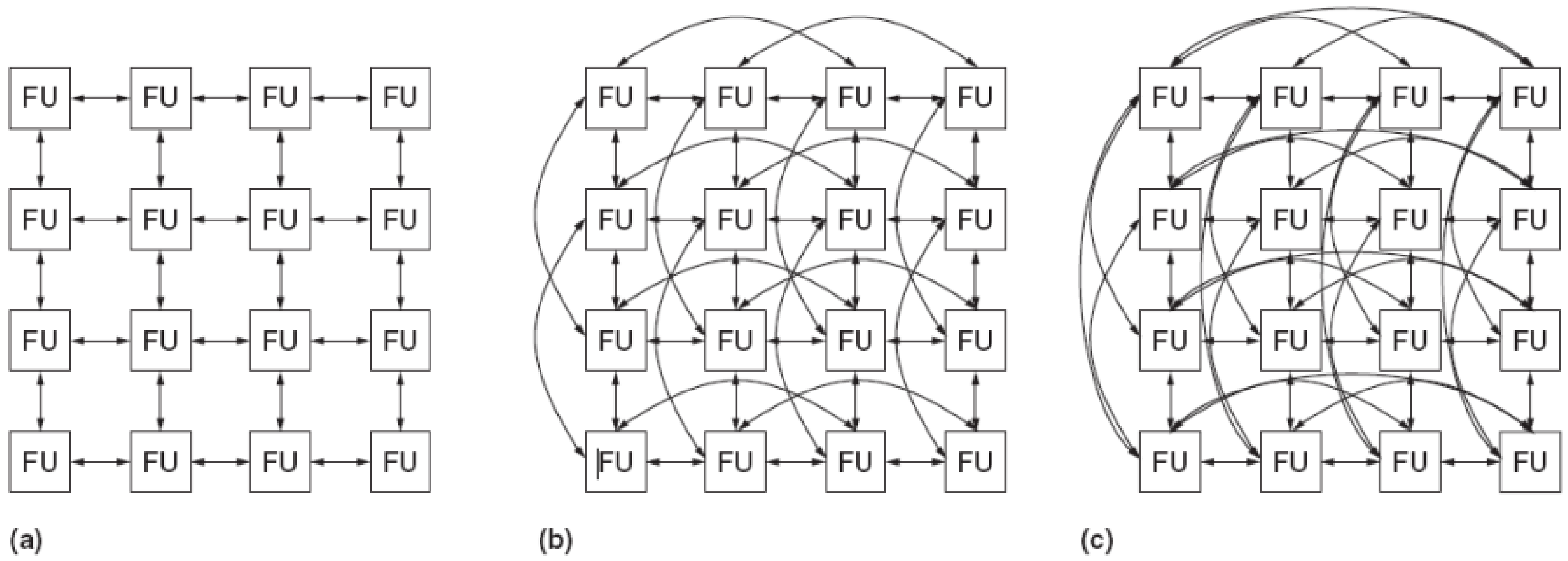
Coarse-grained Reconfigurable Array

- 2-D array of connected ALUs
- Connections often limited to direct neighbors
- Sometimes data may only move downwards (starting at the top of the array and ending at the bottom)



src: PACT's XPP 64-A1 architecture

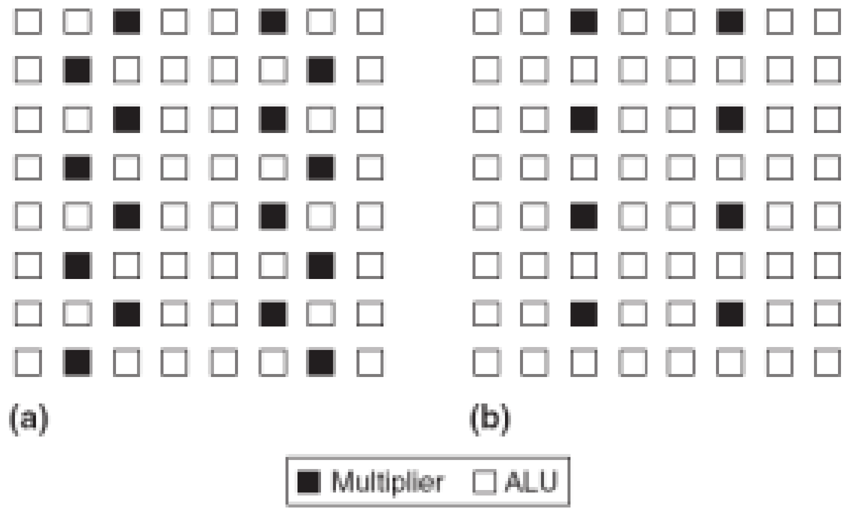
Connection Topology



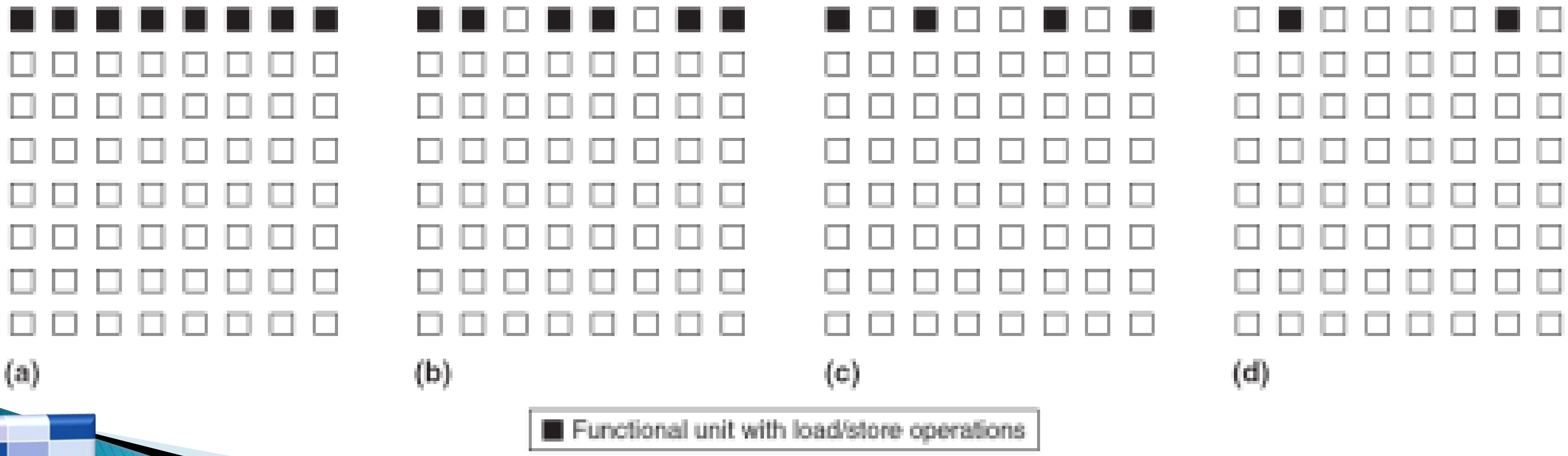
src: B. Mei et al. "Architecture Exploration for a Reconfigurable Architecture Template", IEEE Design and Test of Computers, vol. 22, no. 2, pp. 90-101, 2005

- Different connection Topologies: Performance vs. Area
  - 2D Mesh (1 step Manhattan neighborship, also called von Neumann neighborship)
  - Extended Mesh (2 step orthogonal Manhattan neighborship)
  - Full orthogonal neighborship (each FU can access all other FUs in the same column and the same row)

Heterogeneity: Special Units



- Left side: Number and placement of Multipliers (more expensive than ALUs)
- Bottom side: Number and placement of load/ store units



src: B. Mei et al. "Architecture Exploration for a Reconfigurable Architecture Template", IEEE Design and Test of Computers, vol. 22, no. 2, pp. 90-101, 2005

Summary

- Reconfigurable Hardware can be used to implement accelerators
  - Connected to CPU
  - Used by applications
- Reconfigurable hardware is implemented by
  - Fine-grained structures (LUT array) or
  - Coarse-grained structures (ALU array)
  - They differ in their efficiency, depending on the required operations (bit/byte level vs. word level)
- Configuration data and configuration time have to be kept in mind to exploit the advantages of run-time reconfiguration