Architekturen und Entwurf von Rechnersystemen Wintersemester 2018/2019

TaPaSCo – The Task Parallel System Composer

TECHNISCHE UNIVERSITÄT DARMSTADT

Perspective:
FPGAs - Why bother?

TECHNISCHE UNIVERSITÄT DARMSTADT

Every two years, FPGAs are 'finally going to arrive'.

Peter Lee, VP of Microsoft Research

FPGAs are for kids and academics. Everyone who needs to do real work uses ASICs.

Anonymous

Perspective:
The Zoo of Computing Devices I

TECHNISCHE UNIVERSITÄT DARMSTADT

- **ASIC** - Application Specific Integrated Circuit
  Examples: Bitcoin Mining, Encryption, Audio Codec, Apple M7/8/9/M10, ...
- **μ-Controller** - Instruction Set Architecture (ISA), limited scope
  Examples: Arduino, ARM Cortex M-series, Atmel AVR series, ...
- **System-on-Chip** - small-scale computing architecture
  Examples: Raspberry Pi (BCM28xx), Altera D0-series, Xilinx Zynq-Series, ...
- **Low-Power CPU** - desktop-class CPUs w/focus on energy-efficiency
  Examples: Intel Celeron, Intel Core i3, ARM big.LITTLE, ...

Perspective:
The Zoo of Computing Devices II

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Multi-Core CPU / SoC** - desktop class and server class CPUs, SoCs
  Examples: Intel Core i5/i7-series, AMD Ryzen, Intel Xeon E5-series, ...
- **GPGPU** - General Purpose Graphics Processing Unit
  Examples: NVIDIA GeForce-series, AMD R-series, NVIDIA Tesla P-series, ...
- **Many-Core** - massively parallel CPUs (Intel Xeon Phi)
  Examples: Intel Xeon Phi & Knight's Landing, Sunway TaihuLight, Kalray MPPA, ...
- **DSP** - Digital Signal Processors, massively parallel arithmetic units
  Examples: Texas Instruments Cx000-series

Perspective:
The Zoo of Computing Devices III

TECHNISCHE UNIVERSITÄT DARMSTADT

. . . and last but not least:

- **FPGA** - Field Programmable Gate Array
  Examples: Xilinx Virtex-series, Altera Stratix-series, Lattice ICE-series, . . .

It's confusing! Why are there so many different technologies? Which ones should we use?

Perspective:
An attempt at a classification (incomplete)

TECHNISCHE UNIVERSITÄT DARMSTADT

Commodity ISAs *μ-Controller, LPCPUs, Multi-Core CPUs / SoCs*

- standardized instruction set
- extensive tool support (compilers, debuggers, . . .)
- programmable in mainstream languages (e.g., C)

Specialized ISAs *GPGPUs, DSPs, Many-Cores*

- non-standardized instruction set (e.g., NVIDIA PTX)
- limited tool support (vendor tools)
- programmable in specialized languages (e.g., OpenMP, CUDA, OpenCL, . . .)

Perspective:
An attempt at a classification (incomplete)

TECHNISCHE UNIVERSITÄT DARMSTADT

Reconfigurable Technology *PLDs, FPGAs*

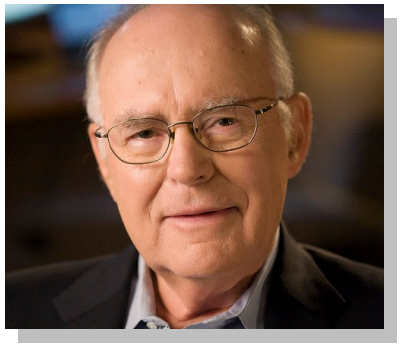
- full-custom design, non-standard
- limited tool support (vendor tools)
- require hardware design languages (e.g., Verilog, VHDL, Bluespec, . . .)
- limited support for mainstream and specialized languages (e.g., C, C++, OpenCL)

ASICs *application specific hardware, e.g., Bitcoin mining, encryption*

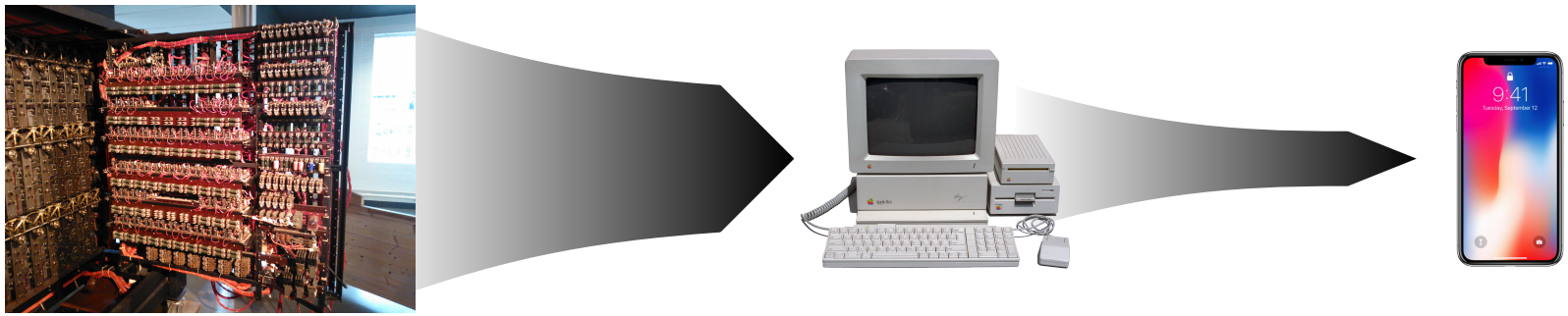
- full-custom design, non-standard
- no compiler support (need to develop OS integration, API)
- require hardware design languages
- (user-)programmable only in low-level languages / via HW interface

Moore's Law

TECHNISCHE UNIVERSITÄT DARMSTADT



- **number of transistors/area doubles every two years**[1]
- originally proposed for 10 years, held several decades
- alternative formulation (David House, Intel?)
  computing power doubles every 18 months
- we have grown accustomed to exponential growth



[1] Moore'65: "Cramming more components onto integrated circuits"

Death of Moore's Law

TECHNISCHE UNIVERSITÄT DARMSTADT

- Moore's Law turned from observation to imperative:
- **International Technology Roadmap for Semiconductors**
- specifies manufacturing process nodes
  - 1971: $10\mu m$
  - ...
  - 2001: $130nm$
  - ...
  - 2012: $22nm$
  - ...
  - 2014: $14nm \Leftarrow$ doubtful
  - ...
  - 2017: $10nm \Leftarrow$ did not happen!
  - ...
  - 2020: $5nm \Leftarrow$?

Patterson's Walls

TECHNISCHE UNIVERSITÄT DARMSTADT

- recent nodes have all missed their deadlines
- nicely summarized by Prof. Dr. David Patterson
- technological advances met *Patterson's Walls*[2]:

Power Wall + Memory Wall + ILP Wall

=

Brick Wall

[2] Patterson'06: "Future of Computer Architecture"

Power Wall

TECHNISCHE UNIVERSITÄT DARMSTADT

- Lumped Circuit Abstraction (LCA): nets are **capacitors**
- increasing operating frequency requires faster toggles
- in other words: need to charge/discharge faster

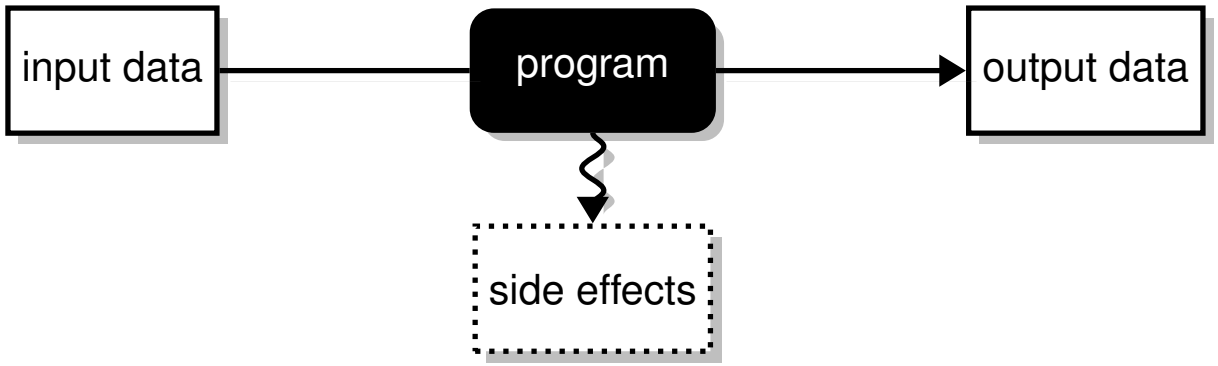
$$\Rightarrow \text{higher currents!}$$

- limited by
  - available power
  - heat dissipation (thermodynamic limits)

Memory Wall

TECHNISCHE UNIVERSITÄT DARMSTADT

- a program is a **filter**:



- memory wall: access latencies + transfer speed
- memory is slow, **bottleneck** in most HPC applications

Memory	≈Access Time (ns)	≈Clock Cycles @ 4GHz
On-chip L1	0.5	2
On-chip L2	7.0	28
Static RAM (SRAM)	10.0 – 20.0	40
DDR SDRAM	60.0 – 100.0	240

Instruction-Level Parallelism Wall

TECHNISCHE UNIVERSITÄT DARMSTADT

inherent parallelism of instruction set architectures is limited

- parallel execution of instructions $\Rightarrow$ increased throughput
- limited by number of **non-contentious hardware resource requirements**
- *example seq: float mul, float mul, . . .*
  requires 2$\times$ float ALU to execute in parallel
- could increase # of hardware units $\Rightarrow$ frequency drops

Brick Wall

TECHNISCHE UNIVERSITÄT DARMSTADT

- Patterson's conjecture:
  If you circumvent two walls, you'll hit the third.

- true for all computer systems that have been built so far
- explains the major trade-offs in HPC:
  - energy-efficiency vs. performance
  - performance vs. memory
  - ...

Beyond Moore's Law

TECHNISCHE UNIVERSITÄT DARMSTADT

- demise of Moore's Law was masked by lateral movements:
  - multi-core architectures
  - parallel computing
  - distributed computing
- . . . but also gave rise to new approaches:
  - GPGPUs
  - DSPs
  - FPGAs
- no clear "winner" emerged - different advantages
- seems obvious to consider **heterogeneous architectures**

Why are there so many different devices? Beyond Moore's Law

TECHNISCHE UNIVERSITÄT DARMSTADT

Beyond Moore's Law

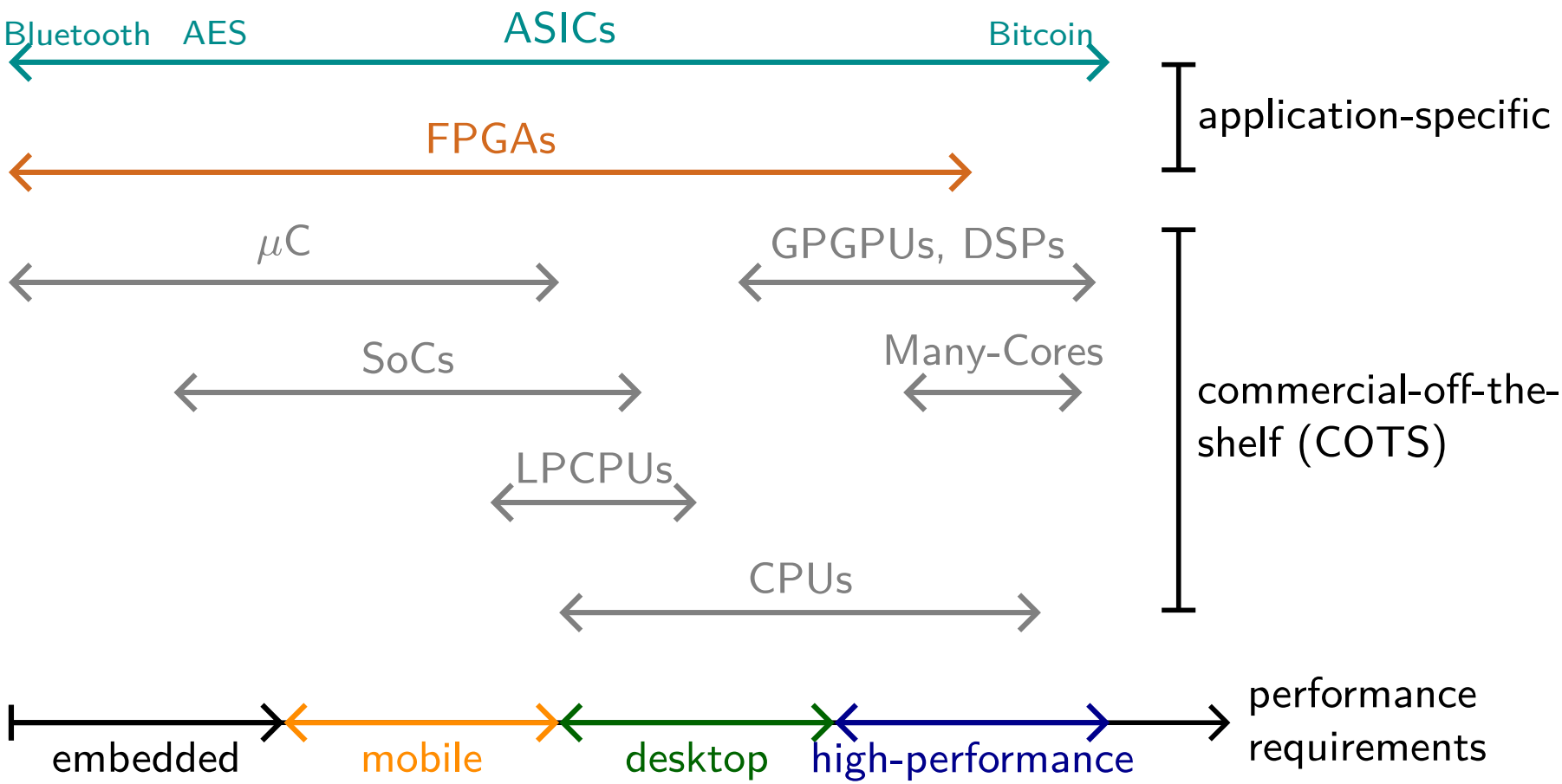
- novel computing approaches to circumvent demise of Moore's Law
- *no silver bullet!* - each approach has its merits and demerits
- technology should be matched to the computational problem

Problem Dimensions

- **performance requirements** – is execution time critical?
- **energy sensitivity** – is power consumption critical?
- **development time/cost** – how quickly changes the problem/algorithm?

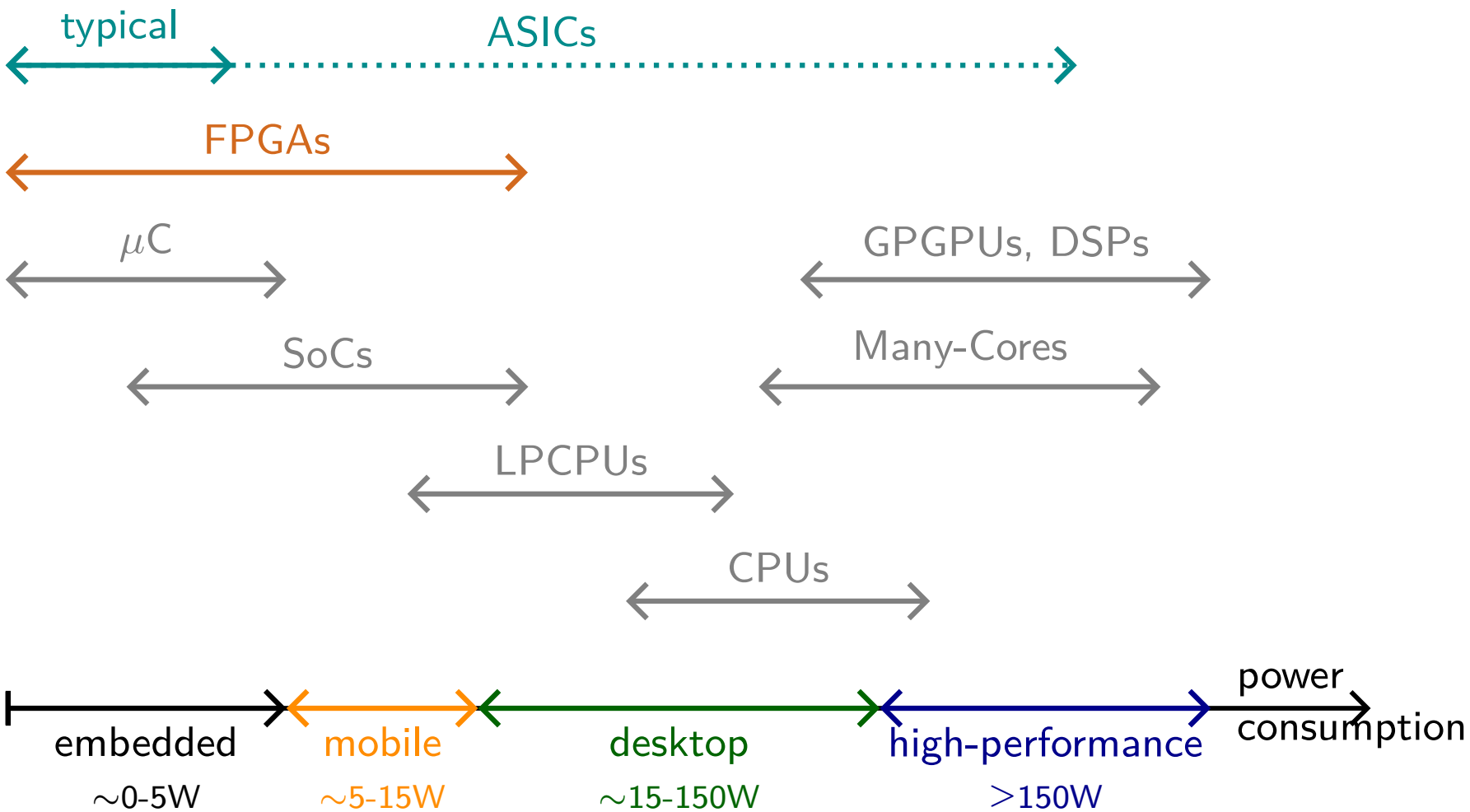
Which technology should we use?
Depends on the kind of computing

TECHNISCHE UNIVERSITÄT DARMSTADT



Which technology should we use?
Depends on the available energy

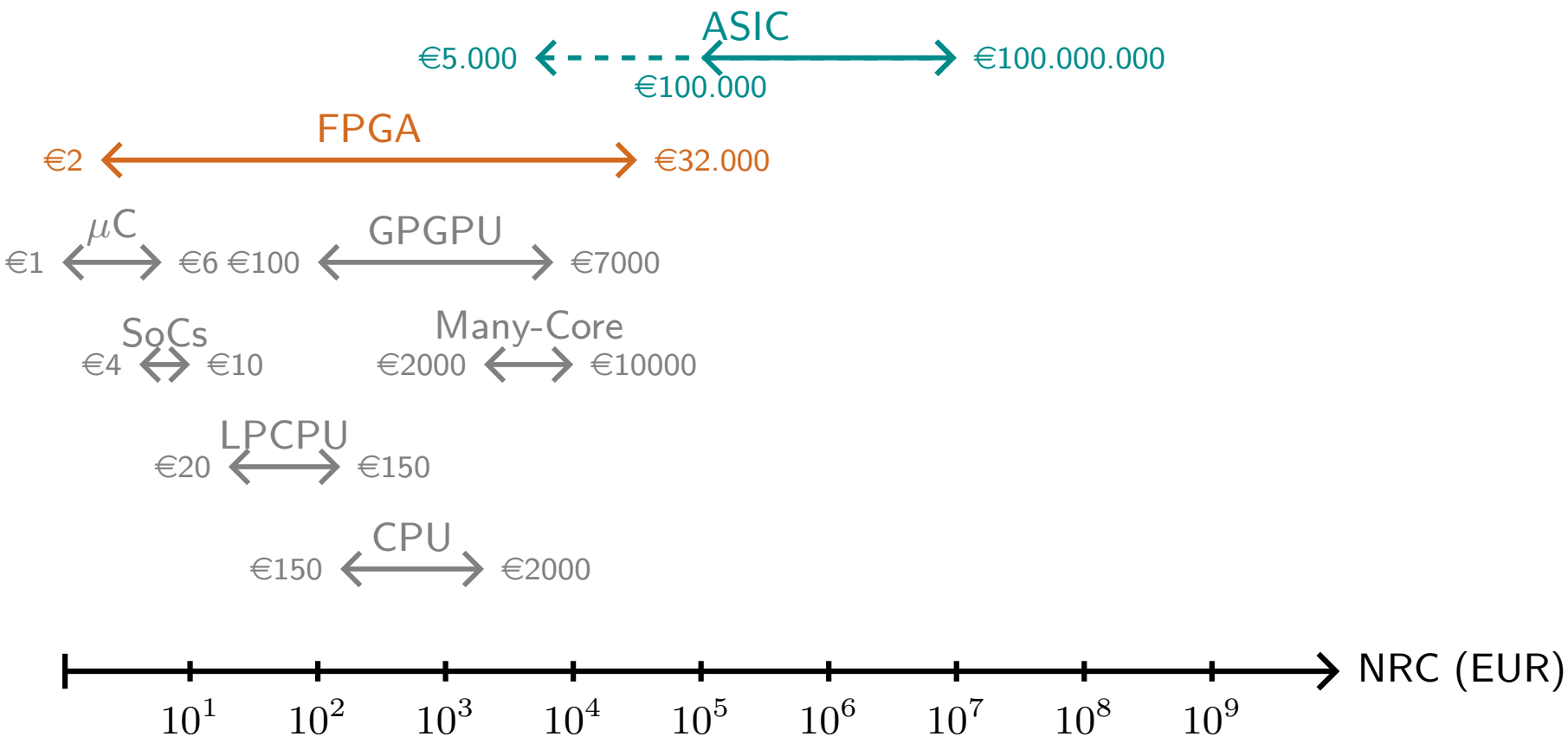
TECHNISCHE UNIVERSITÄT DARMSTADT



Which technology should we use?
Depends on the development cost

TECHNISCHE UNIVERSITÄT DARMSTADT

Non-Recurring Costs (Materials)



Which technology should we use? Depends on the development cost

TECHNISCHE UNIVERSITÄT DARMSTADT

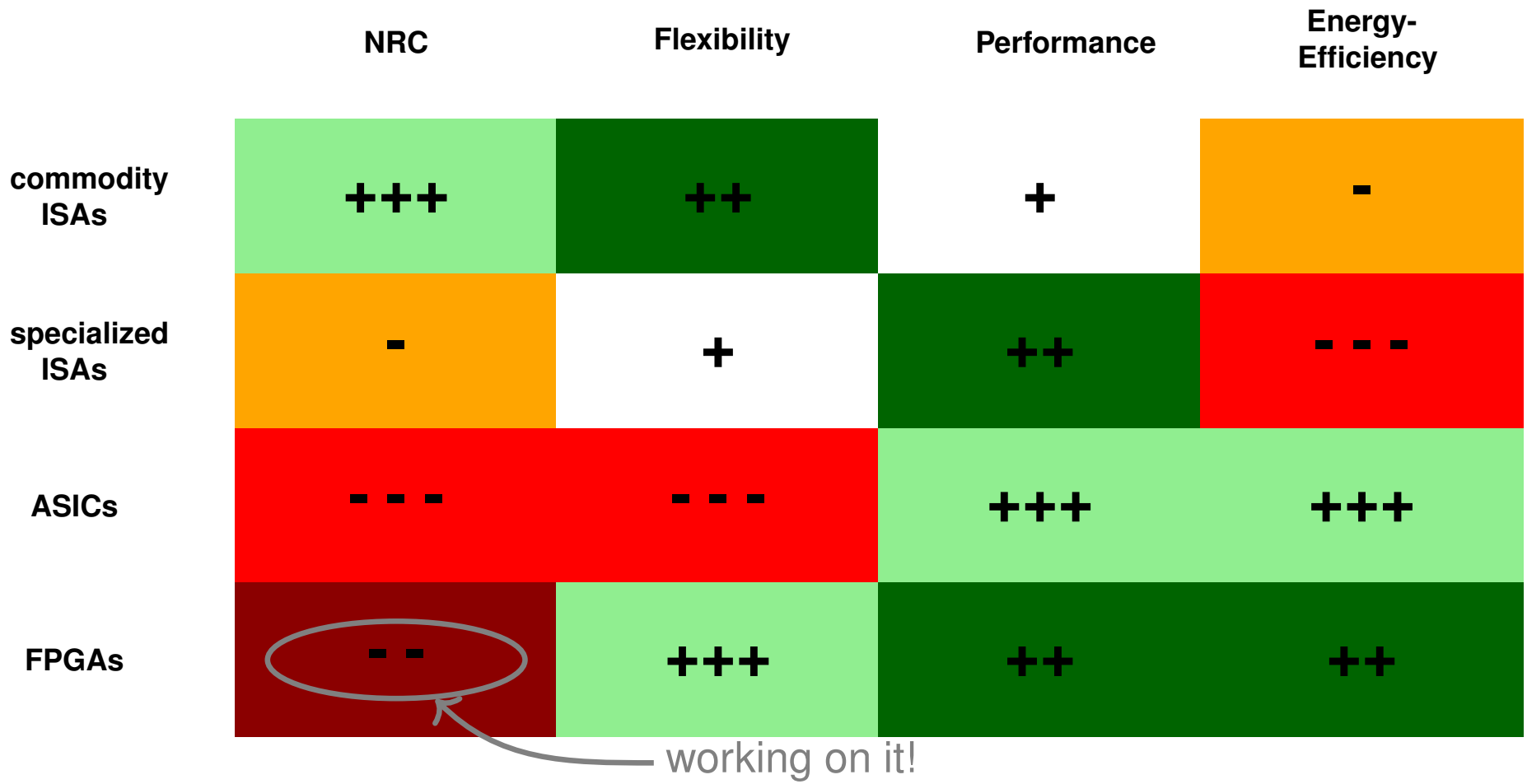
Non-Recurring Development Costs (not including materials!)

- can be split into three components:
  1. cost of developers (€/h)
  2. time required for each development iteration ($\sim$ speed / flexibility)

1. cost of developers $\sim$ avail. number of developers trained in tech
2. development time $\sim$ level of customization
   - COTS solutions w/commodity ISAs have fastest turn-around (days)
   - COTS solutions w/specialized ISAs slightly slower (days, weeks)
   - FPGA solutions often slower by 10x or more (months)
   - ASIC solutions require manufacturing (months, years)

Which technology should we use? Summary

TECHNISCHE UNIVERSITÄT DARMSTADT



Recap quotes
True?

TECHNISCHE UNIVERSITÄT DARMSTADT

"FPGAs are for kids and academics. Everyone who needs to do real work uses ASICs."

Anonymous

Misunderstanding of the technology

- problems that qualify for an ASIC need to
  - change very little
  - stay relevant

  for the next **5-10 years!** (to amortize NRC)
- *no silver bullet* – different problems require different solutions

Recap quotes
True?

TECHNISCHE UNIVERSITÄT DARMSTADT

Every two years, FPGAs are 'finally going to arrive'.

Peter Lee, VP of Microsoft Research

Project Catapult

- supervised by the same Peter Lee
- started in 2010 as datacenter tech to accelerate Bing w/FPGAs
- 2012: large scale pilot program, custom FPGA board (1623 servers)
- 2016: "Configurable Cloud" in nearly every production server (MICRO 2016)

Recap quotes
True?

TECHNISCHE UNIVERSITÄT DARMSTADT

Every two years, FPGAs are 'finally going to arrive'.

Peter Lee, VP of Microsoft Research

Amazon Elastic Compute Cloud (EC2)

- cloud computing datacenters now use FPGAs
- new "F1" instance provides user-configurable FPGAs (Xilinx UltraScale+)
- now in full production use, marketplace for FPGA accelerations functions

Recap quotes
True?

TECHNISCHE UNIVERSITÄT DARMSTADT

Every two years, FPGAs are 'finally going to arrive'.

Peter Lee, VP of Microsoft Research

Intel buys Altera (2015)

- one of two largest FPGA vendors, for approx. $16.7 Billion
- merging Intel Xeons with Altera Stratix FPGAs
- shrinking their traditional divisions by 11% at the same time (2016)

Recap quotes
True?

TECHNISCHE UNIVERSITÄT DARMSTADT

Every two years, FPGAs are 'finally going to arrive'.

Peter Lee, VP of Microsoft Research

... not sure when they are "going to arrive", but:

- are not going away any time soon now
- FPGA devs and engineers are already in short supply
- will be a much sought-after skillset in years to come

Typical Workflow for an FPGA-based solution

TECHNISCHE UNIVERSITÄT DARMSTADT

1. select or build a board
2. build or adapt a base design
3. implement custom logic
4. simulate the design
5. place and route
6. exercise on real hardware

hardware bring-up

logic design

hardware synthesis

Typical Workflow for an FPGA-based solution

1 select or build a board

TECHNISCHE UNIVERSITÄT DARMSTADT

High flexibility comes at a cost:

- COTS hardware differs in details, but is mostly homogeneous
- FPGA-based solutions are much more heterogeneous!
- even off-the-shelf FPGA boards vary wildly:
  1. **on-chip**, i.e., in the FPGA itself
  2. **off-chip**, i.e., in the peripheral components

Typical Workflow for an FPGA-based solution
Details: Differences between FPGAs I

TECHNISCHE UNIVERSITÄT DARMSTADT

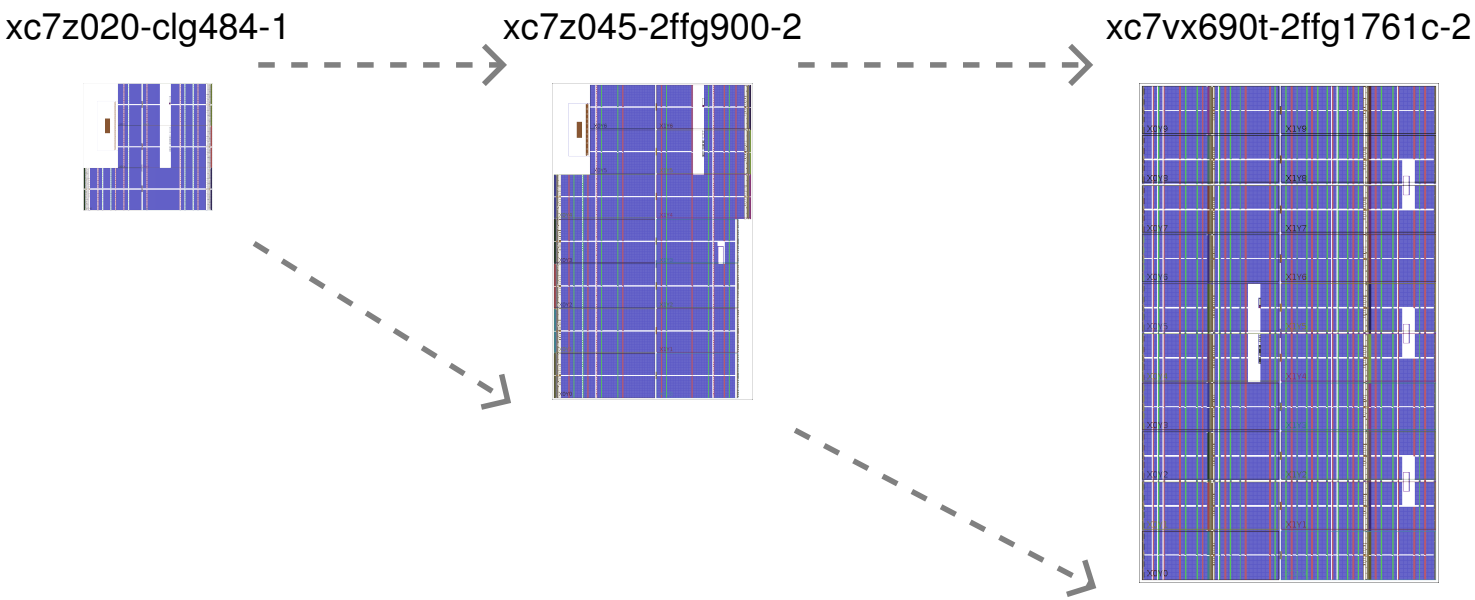
FPGA vary in their characteristics regarding

- architectural family (on-chip resources)
- speed grade ($\sim$ max. operating frequency)
- area and slice composition *(more on next slide . . .)*
- miscellaneous other properties:
  - energy consumption
  - space hardening
  - industrial
  - military
  - . . .

Typical Workflow for an FPGA-based solution Details: Differences between FPGAs II

TECHNISCHE UNIVERSITÄT DARMSTADT

FPGAs differ in the size of the reconfigurable area:

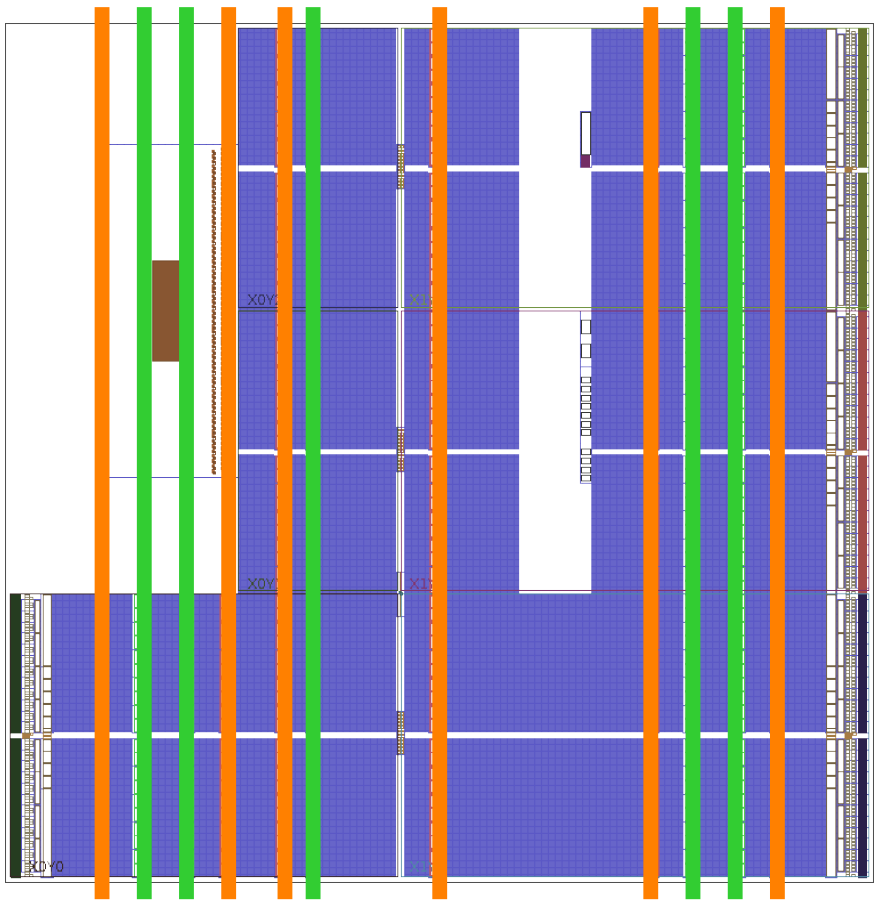


Typical Workflow for an FPGA-based solution
Details: Differences between FPGAs III

TECHNISCHE UNIVERSITÄT DARMSTADT

FPGAs differ in the composition of the reconfigurable area:

- Configurable Logic Blocks (CLBs) are interspersed with *hard blocks*:
  - block RAM (BRAM)
    on-chip memory (kBits)
  - digital signal processors (DSPs)
    dedicated arithmetic units
  - shift registers, Muxes, . . .
- proportion, arrangement vary between *FPGA families*
- families target classes of applications
- variants address specific application requirements:
  - space hardened, automotive, military, . . .



Typical Workflow for an FPGA-based solution
Details: Differences in the periphery

TECHNISCHE UNIVERSITÄT DARMSTADT

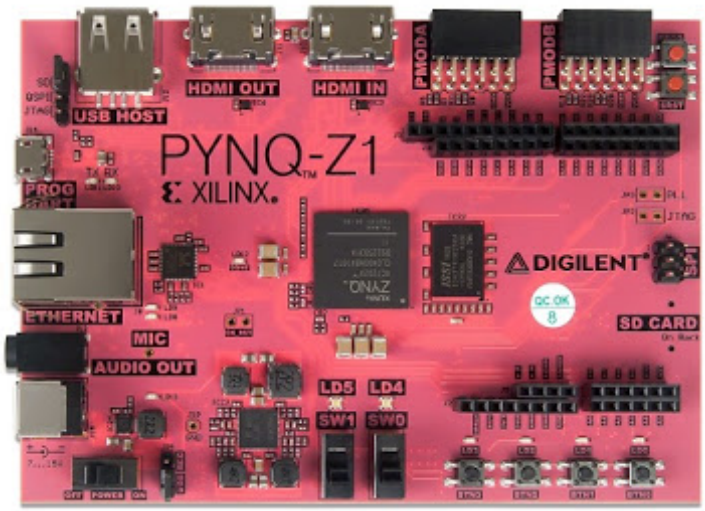
Off-the-shelf FPGA boards provide different peripheral components:

- connectivity to host / outside world, e.g.:
  - PCIe Gen2/3/4
  - Interlaken, RapidIO
  - Intel FSB, QPI, UPI
  - AMD Hypertransport, Infinity Fabric (?)
  - Gen-Z, CCIX
  - Network
  - Shared Memory
  - ...
- accessible memories, caches, e.g.:
  - DDRx, LPDDRx, ...
  - RLDRAM, NVRAM, ...
  - HMC and HBM 3D-Memory, ...
- and secondary peripherals, e.g.:
  LEDs, Displays, Video In/Out, Sensors, ...

Typical Workflow for an FPGA-based solution
Details: Differences between FPGA boards

TECHNISCHE UNIVERSITÄT DARMSTADT

Xilinx PyNQ Z1 – Zynq-series FPGA-SoC (Artix-7)

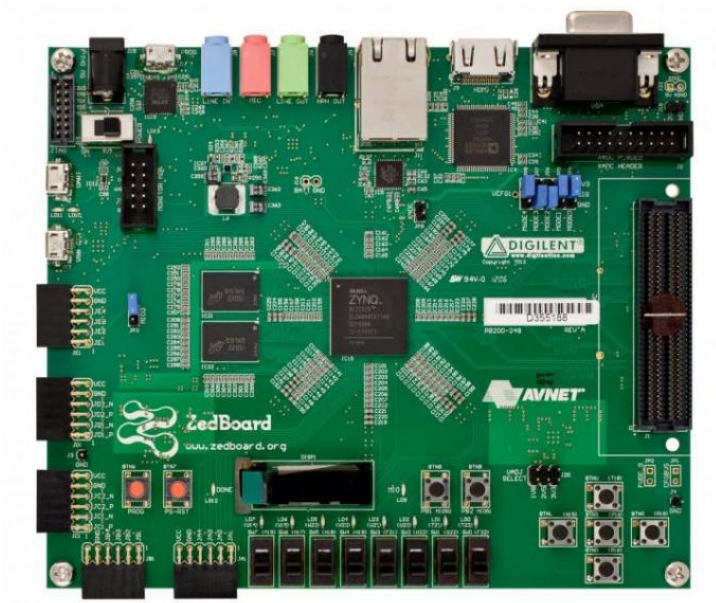


Host connectivity:
AXI3 hard IP (SoC)
Memory:
512MB DDR3, SD card
Other Periphery:
1G Network, HDMI In+Out, Analog Audio, Microphone, LEDs, Switches, Buttons, ...
Size:
∼85.000 Logic Cells
Price:
∼229 USD — **65 USD (academic)**

Typical Workflow for an FPGA-based solution
Details: Differences between FPGA boards

TECHNISCHE UNIVERSITÄT DARMSTADT

Digilent ZedBoard – Zynq-series FPGA-SoC (Artix-7)

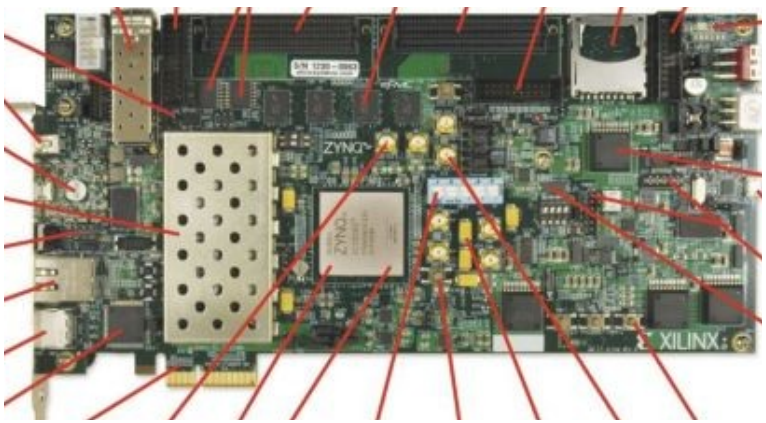


Host connectivity:
AXI3 hard IP (SoC)
Memory:
512MB DDR3, 256 Mbit Quad-SPI Flash, SD card
Other Periphery:
1G Network, FMC, VGA Out, Analog Audio/Video, PMOD, OLED display (128x32), LEDs, Switches, Buttons, . . .
Size:
∼85.000 Logic Cells
Price:
∼495 USD — **319 USD (academic)**

Typical Workflow for an FPGA-based solution
Details: Differences between FPGA boards

TECHNISCHE UNIVERSITÄT DARMSTADT

Xilinx ZC706 – Zynq-series FPGA-SoC (Kintex-7) w/PCIe Gen2.0 x4

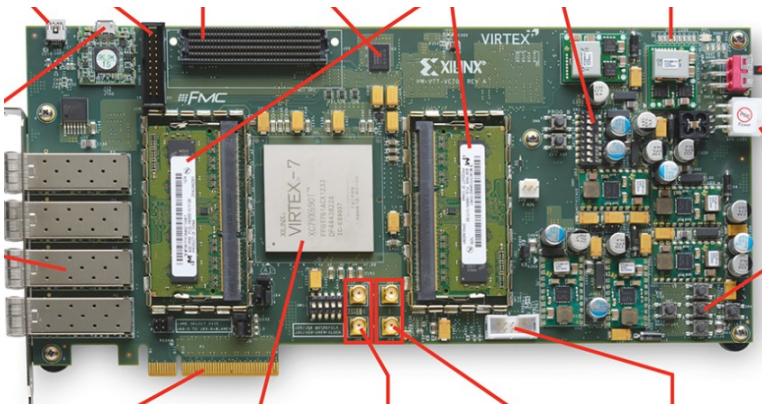


Host connectivity:
AXI3 hard IP (SoC), PCIe Gen2 x4, 1x SFP+ transceiver
Memory:
1GB DDR3 (PL), 1GB DDR3 (PS), 2x128Mbit (Dual) Quad-SPI, SD card
Other Periphery:
2x FMC, Differential InOuts, ...
Size:
$\sim$350.000 Logic Cells
Price:
$\sim$2.850 USD

Typical Workflow for an FPGA-based solution
Details: Differences between FPGA boards

TECHNISCHE UNIVERSITÄT DARMSTADT

Xilinx VC709 – Virtex-7 FPGA w/PCIe Gen3.0 x8



Host connectivity:
PCIe Gen3 x8, 4x SFP+ transceiver
Memory:
2x 4GB DDR3, 128 MB BPI flash
Other Periphery:
1x FMC, Differential InOuts, 8pole DIP switch, ...
Size:
∼690.000 Logic Cells
Price:
∼5.650 USD

Typical Workflow for an FPGA-based solution Recap

TECHNISCHE UNIVERSITÄT DARMSTADT

1. select or build a board
2. build or adapt a base design
3. implement custom logic
4. simulate the design
5. place and route
6. exercise on real hardware

hardware bring-up

logic design

hardware synthesis

Typical Workflow for an FPGA-based solution

2 build or adapt a base design

TECHNISCHE UNIVERSITÄT DARMSTADT

What is a base design?

- the board PCB design controls the pin connections on the *physical level*
- the FPGA bitstream controls the pin connections on the *logical level*
  - which physical pin is connected to which input/output in my design?
  - this is called a *pin constraint*
  - *Example: drive an LED from a logical output*
- great – almost complete wiring freedom, but:
  - you actually need to connect memories (e.g., DDR) on this level!
  - same is true for the connection to the host
  - completely different for each FPGA board
- a *base design* is an empty design with default connections (DDR, . . .)
  - done by vendor for off-the-shelf boards
  - must be done manually in case of custom PCB

Typical Workflow for an FPGA-based solution

3-4 Logic Design

TECHNISCHE UNIVERSITÄT DARMSTADT

Develop the custom logic

- develop a hardware module for the algorithm
- usually done in HDLs, e.g., Verilog/SystemVerilog, VHDL
- more recently: Bluespec, Chisel, . . . (next-generation HDLs)

Test functional correctness by simulation

- behavioral hardware descriptions are simulated on RTL level
- similar to unit testing in software
- aim for coverage to gain confidence in functional correctness

Independent of target hardware (in theory)

- RTL is abstract; can be mapped to any FPGA, or ASIC
- in practice: scalability and portability issues

Typical Workflow for an FPGA-based solution

5-6 Hardware Synthesis

TECHNISCHE UNIVERSITÄT DARMSTADT

Place and route

- two step process:
  1. *place* step maps operations to hardware resources
  2. *route* step performs the wiring and checks timing constraints
- NP-complete problem!
- number of possible mappings grows exponentially with area
- many different algorithmic approaches:
  - heuristic approach with iterative backtracking
  - simulated annealing
  - analytical methods
- most time-consuming step (anywhere from 15min to days)

Exercise the design on real hardware

- necessary! several uncertainties in the process
- real-world perfomance is determined by many outside factors, e.g., OS interactions

Typical Workflow for an FPGA-based solution
Place and Route Example

TECHNISCHE UNIVERSITÄT DARMSTADT

A simple module: 2-bit counter

```
‘timescale 1ns / 1ps
module blinkenlights(
    input wire rst,
    input wire clk,
    output reg[1:0] cnt
    );
  initial     cnt <= ’d0;
  always @(posedge clk) begin
    if (rst) cnt <= ’d0;
    else     cnt <= cnt + 1;
  end
endmodule
```

rst	cnt[1]	cnt[0]	cnt’[1]	cnt’[0]
0	0	0	**0**	**1**
0	0	1	**1**	**0**
0	1	0	**1**	**1**
0	1	1	**0**	**0**
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

rst=1

Typical Workflow for an FPGA-based solution
Place and Route Example

TECHNISCHE UNIVERSITÄT DARMSTADT

A simple module: 2-bit counter

LUT2

rst	cnt[0]	cnt'[0]
0	0	**1**
0	1	**0**
0	0	**1**
0	1	**0**
1	0	0
1	1	0
1	0	0
1	1	0

rst=1

LUT3

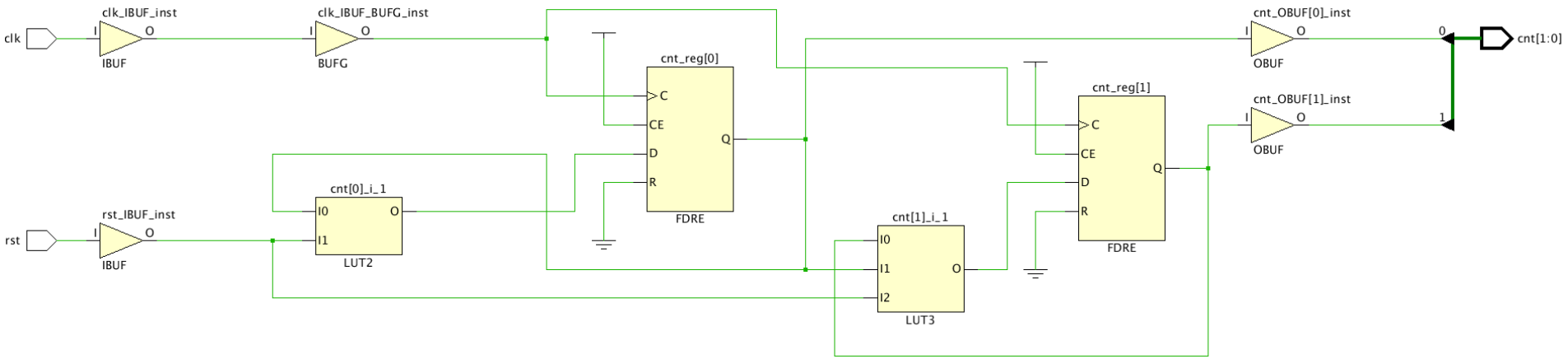
rst	cnt[1]	cnt[0]	cnt'[1]
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

rst=1

Typical Workflow for an FPGA-based solution
Place and Route Example

TECHNISCHE UNIVERSITÄT DARMSTADT

Synthesis result for module



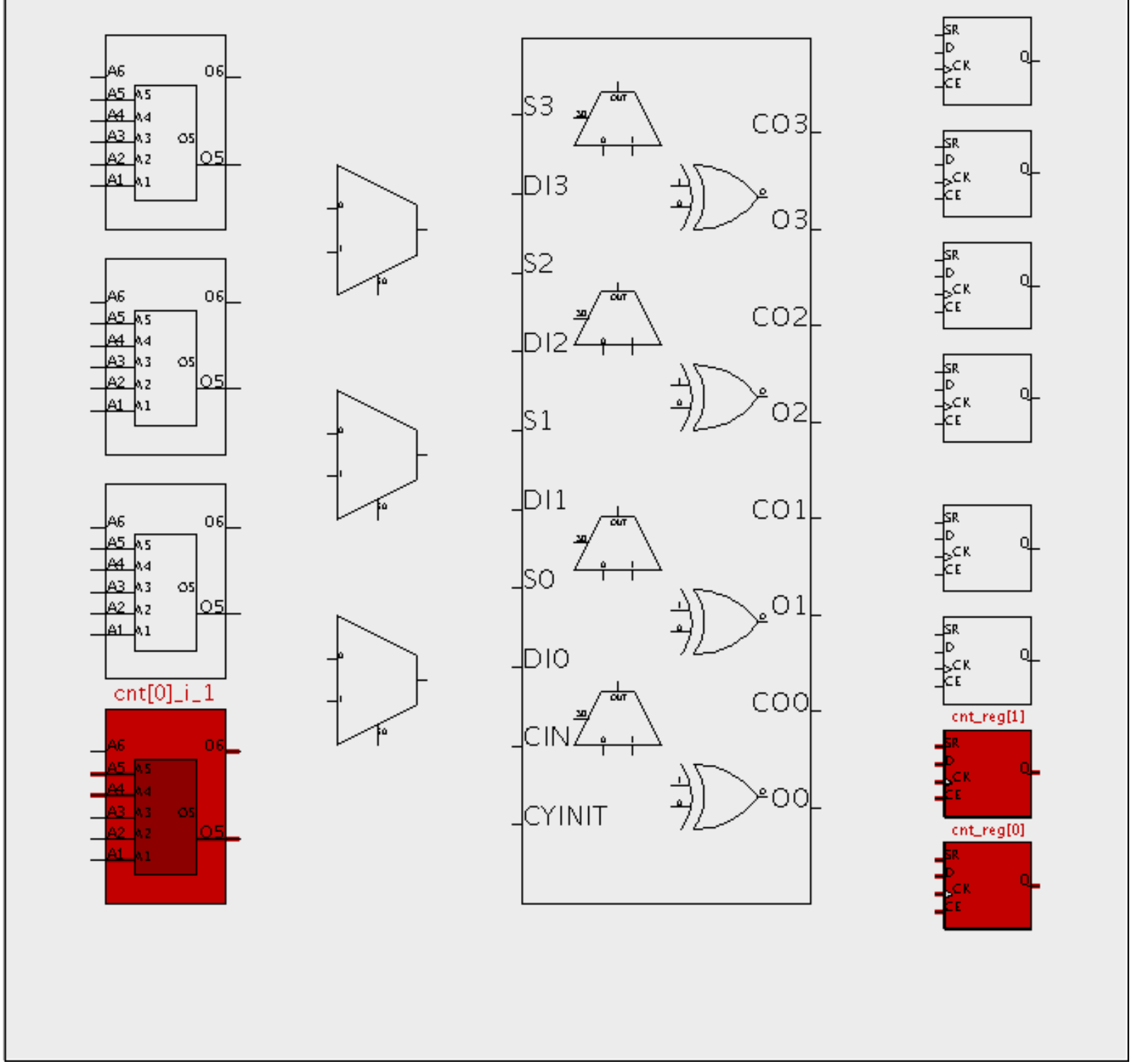
- result of synthesis is called **netlist**
- consists of primitive **instances** and **nets**
- instances here as expected: 1x LUT2, 1x LUT3 + 2x FlipFlops
- **placing** is the process of mapping instances to hardware resources

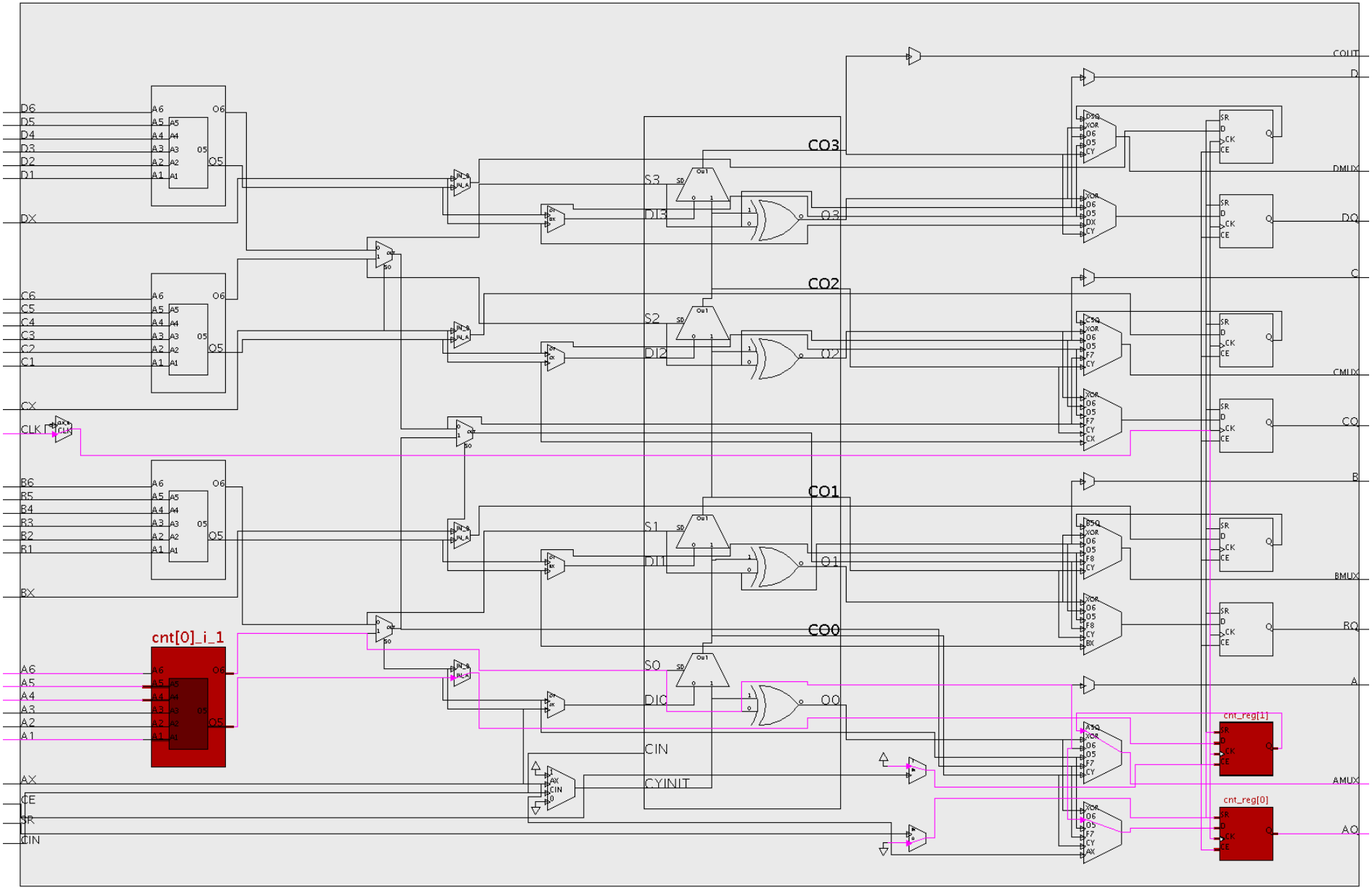
Typical Workflow for an FPGA-based solution
Place and Route Example

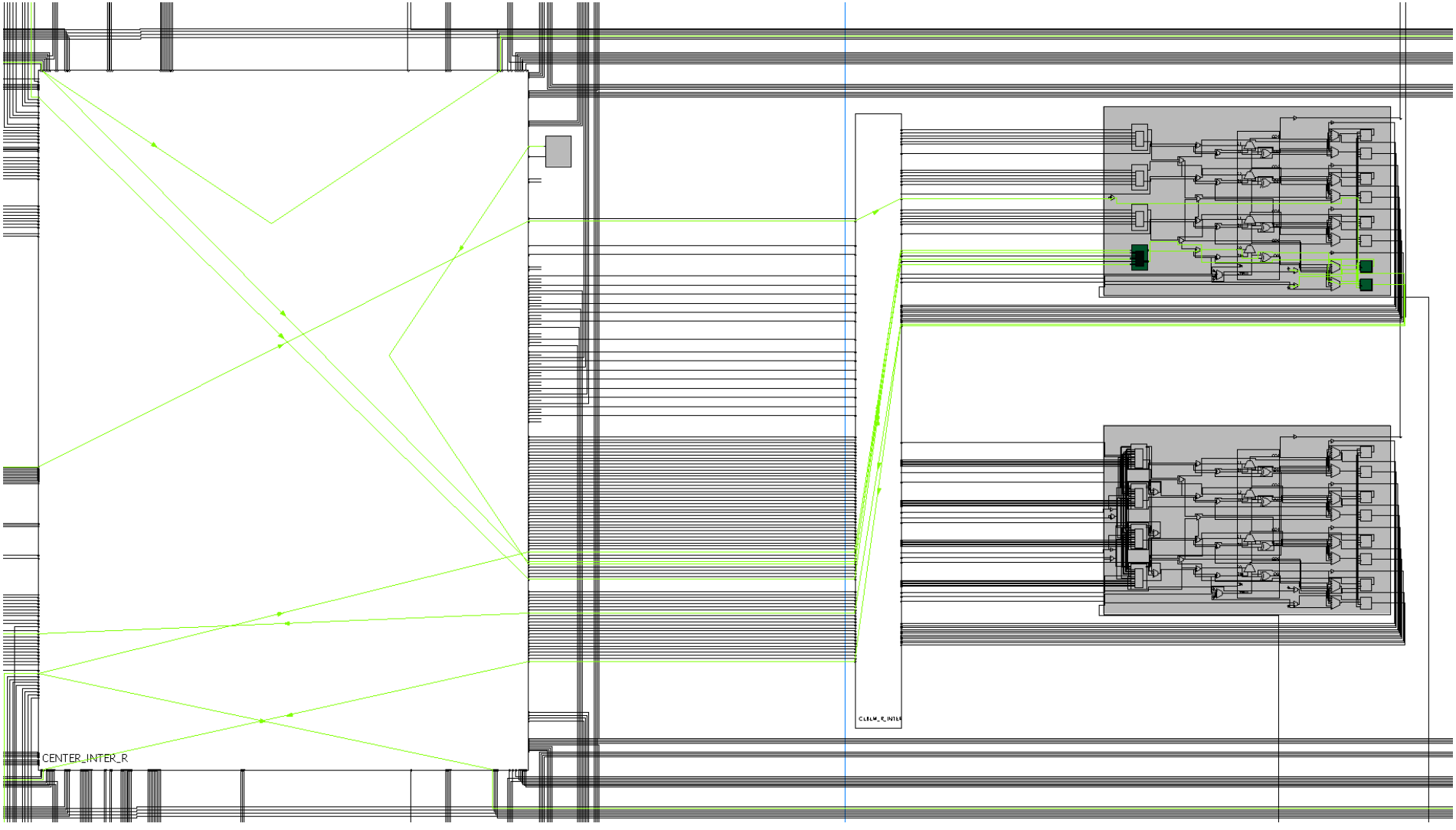
TECHNISCHE UNIVERSITÄT DARMSTADT

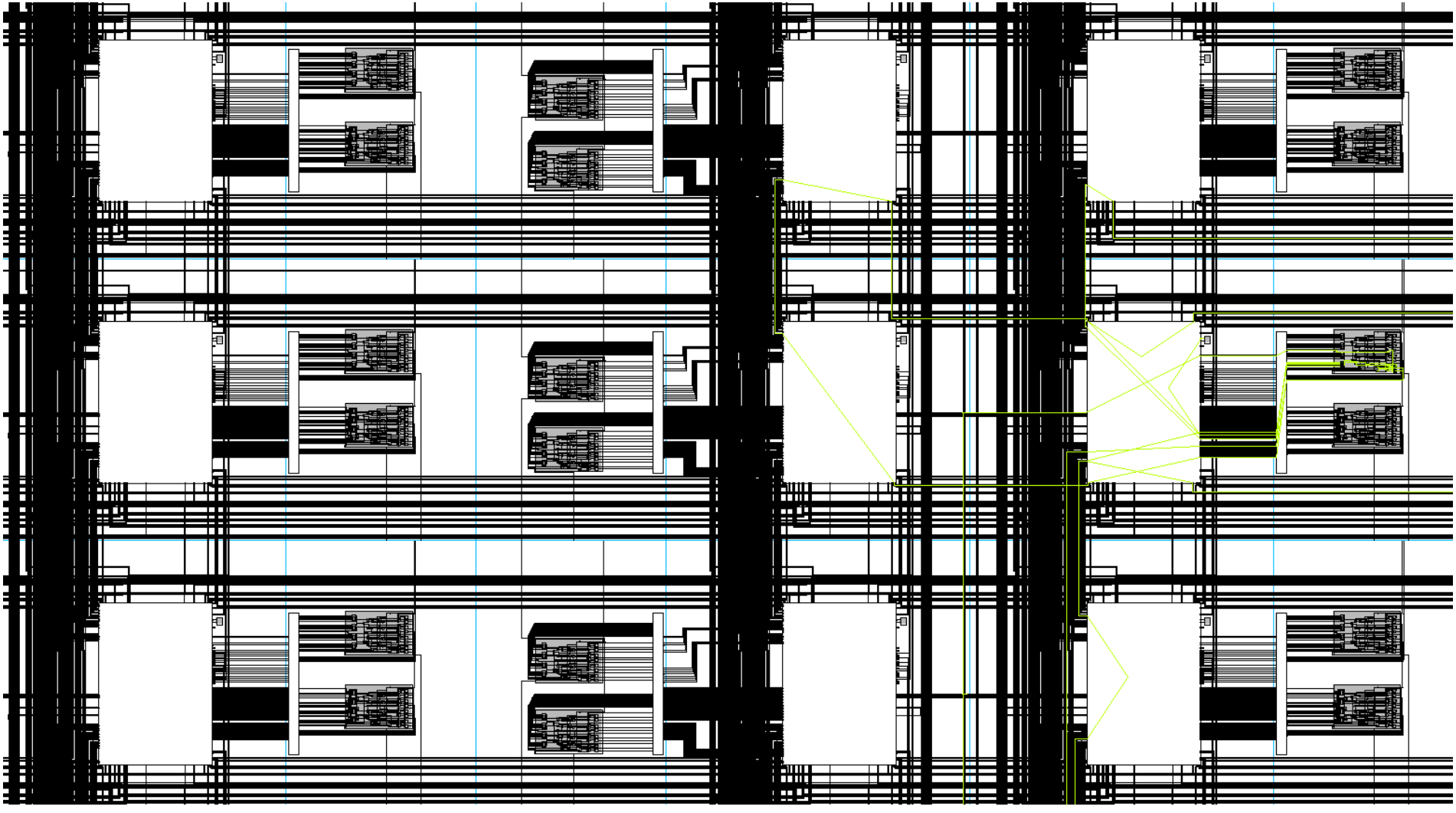
Recall: Configurable Logic Blocks (CLB)

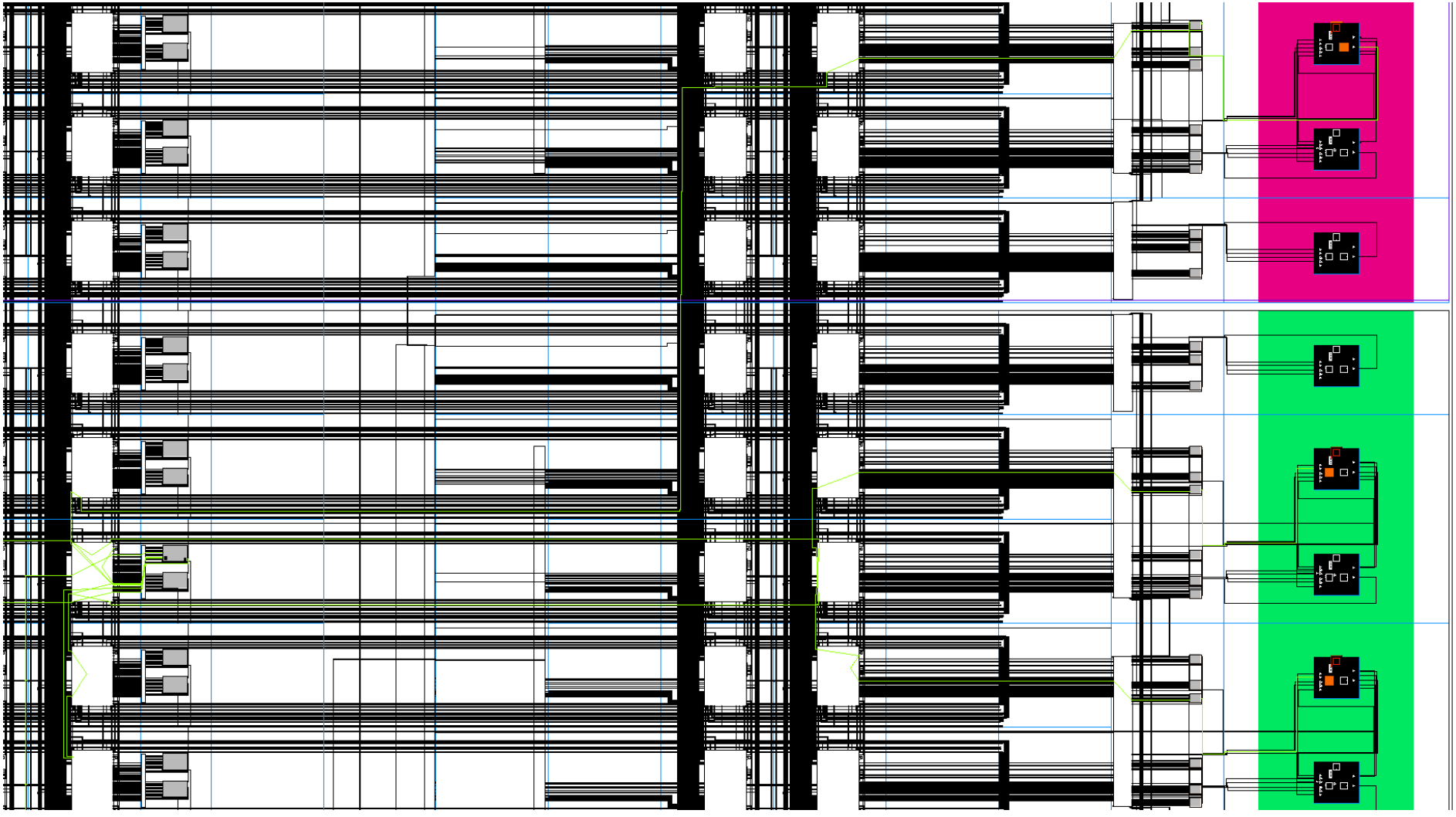
- basic elements of configurable logic
- on Xilinx devices, CLBs usually contain
  1. 4x LUT6
  2. 8x Storage elements
  3. 3x Wide-function Multiplexer (MUX)
  4. Carry logic
- LUT6 can be configured as
  1. 1x 6-input-1-output, or
  2. 2x $\leq$5-input-2-output
- multiple LUTs can be combined with
  1. F7-MUXes to generate up to 2x 7-input-1-output LUTs
  2. F8-MUX to generate 1x 8-input-1-output LUT

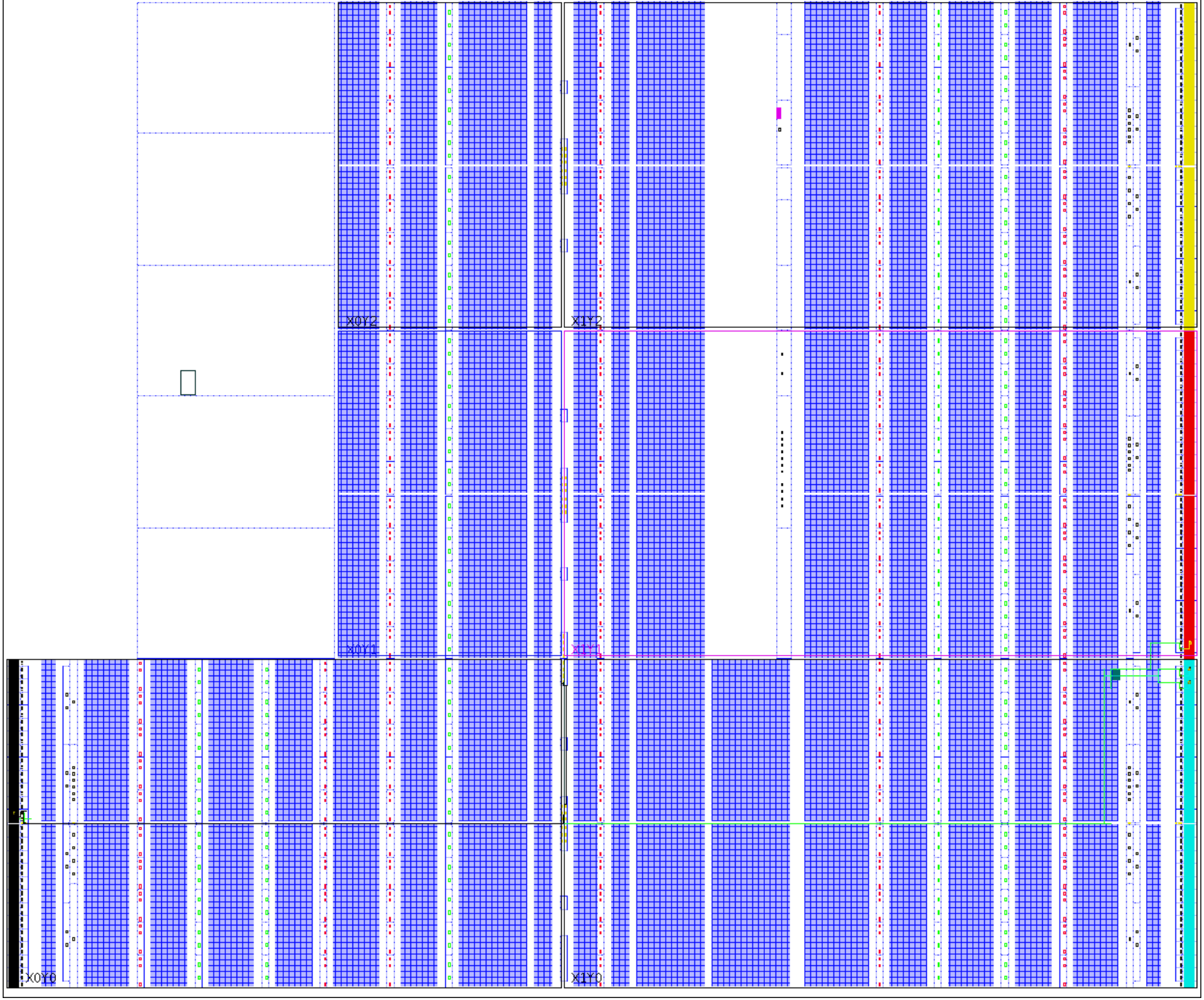

A6
A5
A4
A3
A2
A1
O6
O5
S3
DI3
S2
DI2
S1
DI1
S0
DI0
CIN
CYINIT
CO3
O3
CO2
O2
CO1
O1
CO0
O0
SR
D
CK
CE
Q
cnt[0]_i_1
cnt_reg[1]
cnt_reg[0]


D6
D5
D4
D3
D2
D1
DX
C6
C5
C4
C3
C2
C1
CX
CLK
B6
B5
B4
B3
B2
B1
BX
A6
A5
A4
A3
A2
A1
AX
CE
SR
CIN
cnt[0]_i_1
CO3
CO2
CO1
CO0
S3
S2
S1
S0
DI3
DI2
DI1
DI0
O3
O2
O1
O0
CYINIT
COUT
D
DMUX
DQ
C
CMUX
CQ
B
BMUX
BQ
A
AMUX
AQ
cnt_reg[1]
cnt_reg[0]


CENTER_INTER_R






X0Y2
X1Y2
X0Y1
X1Y1
X0Y0
X1Y0

Typical Workflow for an FPGA-based solution
Recap: Current Problems

TECHNISCHE UNIVERSITÄT DARMSTADT

but most time is spent here

this should be the focus!

1. select or build a board
2. build or adapt a base design
3. implement custom logic
4. simulate the design
5. place and route
6. exercise on real hardware

hardware bring-up

logic design

hardware synthesis

Typical Workflow for an FPGA-based solution
Recap: Current Problems

TECHNISCHE UNIVERSITÄT DARMSTADT

Two major issues driving up the NRC in FPGA-based solutions

Portability

- hardware choices made at start influence the entire workflow, result
- change of hw leads to repetition of most time-consuming steps
- makes choice of hardware **both critical and difficult**

⇓

addressed by **TaPaSCo**
(this lecture)

Scalability

- resources vary significantly; hardware designs often cannot scale the full range
- classic HDLs (Verilog, VHDL) are too verbose and not expressive enough
- good test coverage is hard to obtain; repetitive development efforts

⇓

addressed by Bluespec, **Chisel**, . . .
but also **TaPaSCo** (coarse-grain)

TaPaSCo
The Task-Parallel System Composer

TECHNISCHE UNIVERSITÄT DARMSTADT

- automated tool-flow generating bitstreams for four *Platforms*:
  - Zynq: Xilinx PyNQ, Digilent ZedBoard, Xilinx ZC706
  - Virtex-7: Xilinx VC707, Xilinx VC709, NetFPGA-SUME, ...
  - UltraScale+: VCU118, VCU1525, ...
- automated *high-level synthesis (HLS)* pass to generate hardware from C/C++
- uniform *application programming interface (API)* - write software once!
- based on Scala and Vivado IP Integrator Tcl
- **free software, LGPL license**

TaPaSCo
Underlying model and assumptions

TECHNISCHE UNIVERSITÄT DARMSTADT

Task Parallelism

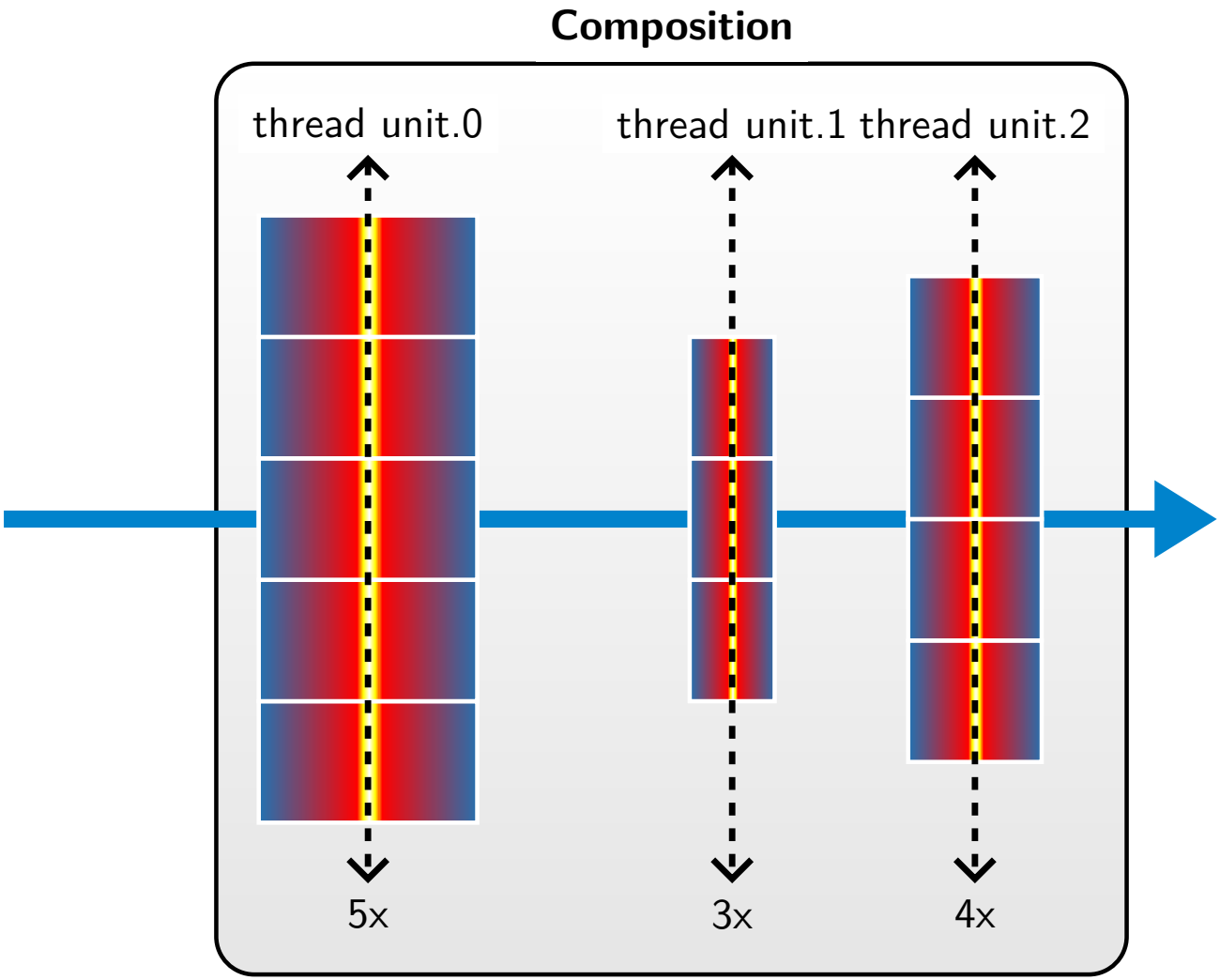
- TaPaSCo implements a **task parallel model of computation**
- computation is grouped into phases called **tasks**
- tasks may depend on the outputs of their predecessors (workflow graph)
- data for each task can be partitioned into **jobs**
- parallelization across multiple **processing elements (PEs)**
- jobs are scheduled on the PEs asynchronously

Assumptions / Requirements

- problem can be decomposed into independent jobs
- jobs can be executed in parallel, communication is limited:
  - inter-task communication is not performance critical (cf. Amdahl's Law)
  - no inter-job communication, jobs are truly independent

TaPaSCo
Parallelising a sequential program

TECHNISCHE UNIVERSITÄT DARMSTADT



- profile sequential program
- identify **computational hotspots (kernels)**
- isolate kernel code, execute non-critical code on main thread
- replicate hardware for each kernel spatially: **parallel processing elements (PE)**
- PEs for each kernel are called **thread unit**
- **Composition**: size of thread units for each kernel
  here: (5, 3, 4)

TaPaSCo
Parallelism in TaPaSCo

TECHNISCHE UNIVERSITÄT DARMSTADT

- many terms for this type of parallelism:
  - task parallelism
  - coarse-grained parallelism
  - fork-join parallelism
  - work group parallelism
  - ...
- we use the term **thread pool**
  - conceptual abstraction over a *Composition*:
    - a pool consists of a number of thread units for kernels
    - each thread unit consists of a number of PEs (instantiations of kernels)
  - operational abstraction:
    - jobs for each kernel can be submitted to the pool
    - are executed on first available PE
    - results can be collected out-of-order
  - no inter-PE communication (handled on main-thread)

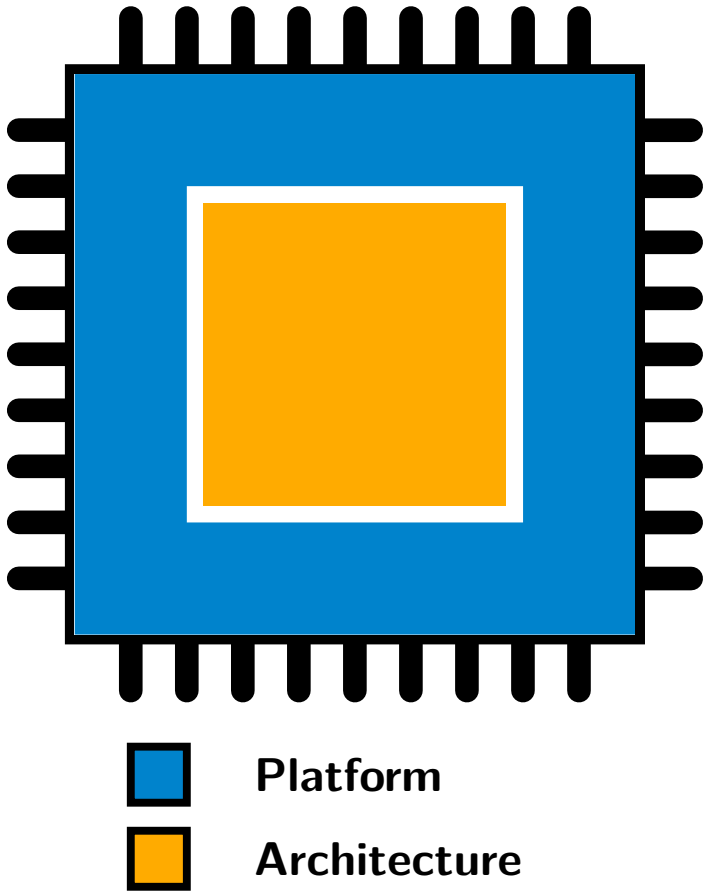
TaPaSCo
Compilation Flow

TECHNISCHE UNIVERSITÄT DARMSTADT

TaPaSCo
Design Abstractions

TECHNISCHE UNIVERSITÄT DARMSTADT

- TaPaSCo designs are split into two distinct parts:
  1. **Architecture**
     - thread units and PEs, according to Composition
     - interconnections (e.g., AXI4-based)
     - independent of target platform / board
  2. **Platform**
     - connection to host and memory
     - hardware-dependent
- advantage: hardware-dependent parts are isolated in **Platform**
- represented in TaPaSCo as plug-in scripts
  - easy to modify / re-use existing scripts
  - easy to add new **Platforms** and **Architectures**



TaPaSCo
Architecture Scripts

TECHNISCHE UNIVERSITÄT DARMSTADT

Tasks

1. define basic interface of PEs (HLS directives)
2. instantiate PEs according to composition
3. combine PEs into thread units + perform wiring
4. combine thread units into thread pool + perform wiring

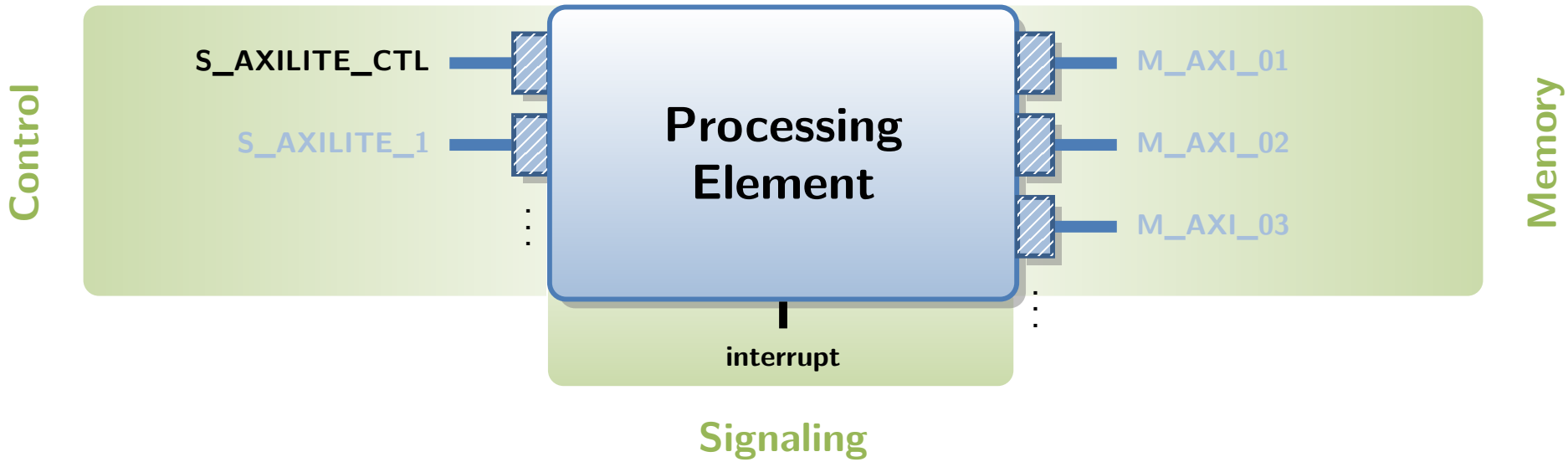
Example Architecture: baseline (AXI4)

- uses AXI4 interfaces for communication
- uses AXI4 interconnects to facilitate communication between host and PEs

TaPaSCo
Example Architecture: baseline

TECHNISCHE UNIVERSITÄT DARMSTADT

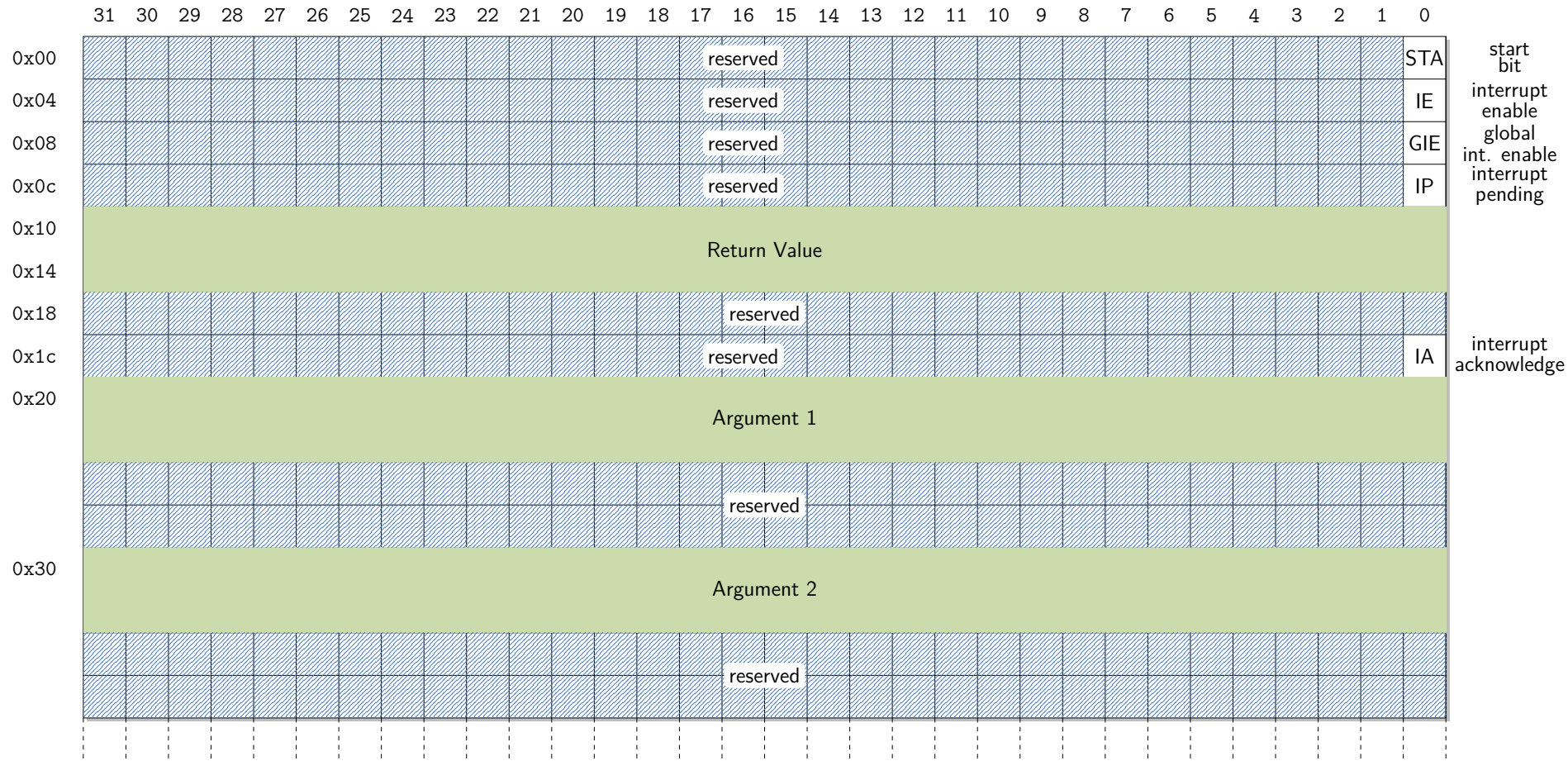
PE Interface: AXI4-based



- AXI4Lite slave interface (memory-mapped)
- standardized control register file
- level-sensitive interrupt line
- logic 1 $\Leftrightarrow$ completed execution awaits acknowledgement
- *optional:* AXI4/AXI4Lite master interfaces

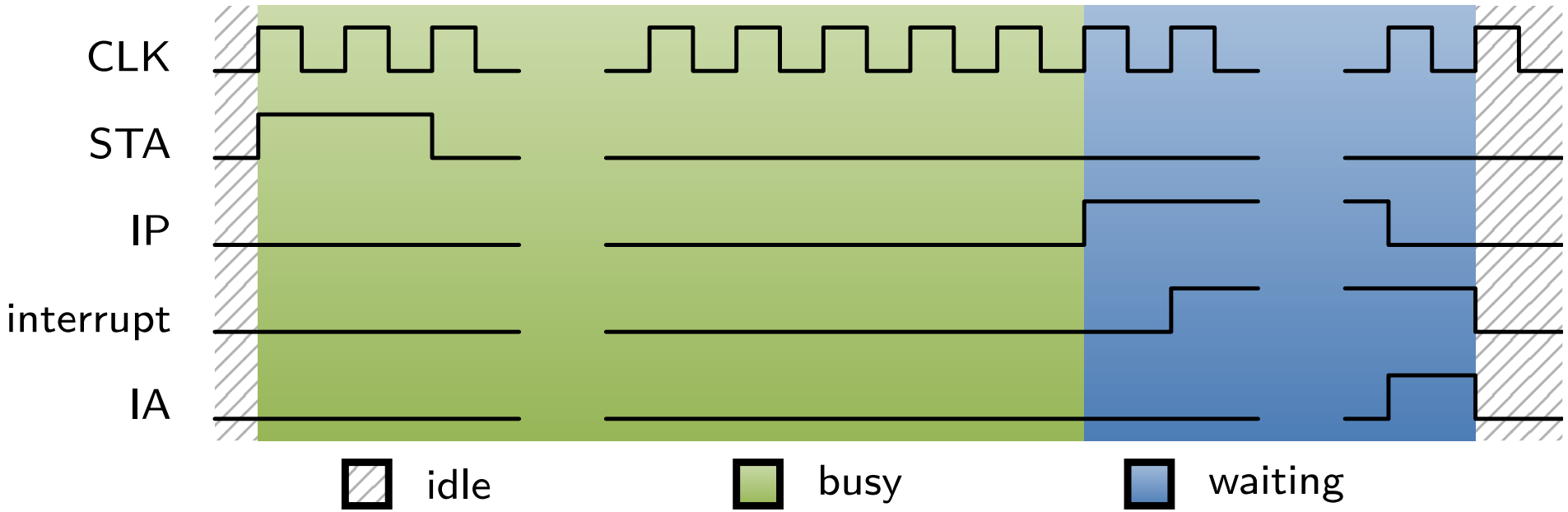
TaPaSCo
Memory-Mapped AXI4Lite Control Register File

TECHNISCHE UNIVERSITÄT DARMSTADT



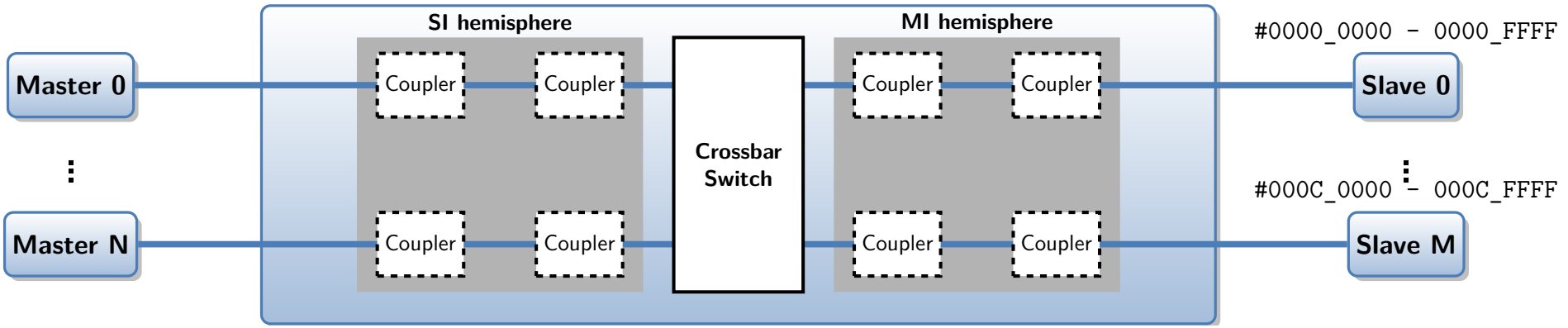
TaPaSCo
PE Timing Diagram

TECHNISCHE UNIVERSITÄT DARMSTADT



TaPaSCo
AXI4 Interconnect IP (Xilinx)

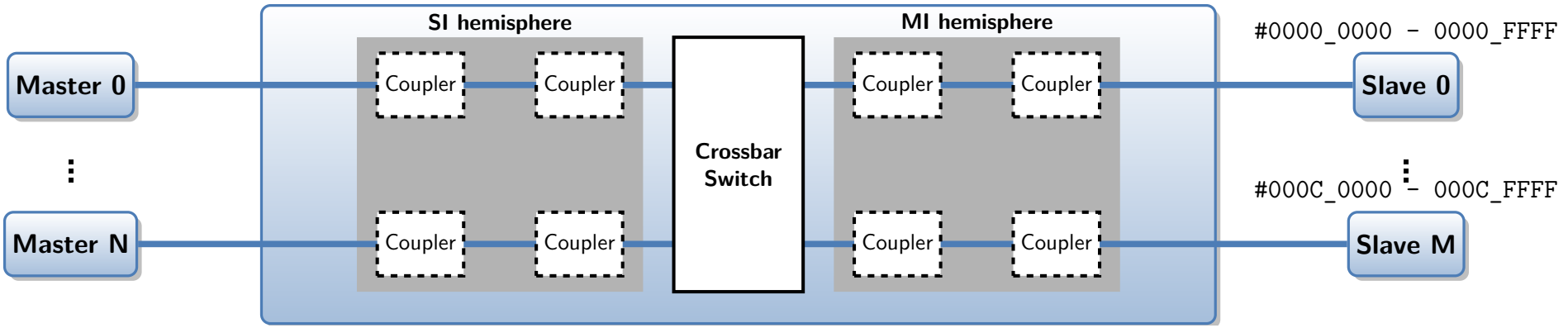
TECHNISCHE UNIVERSITÄT DARMSTADT



- connects up to 16 master interfaces with up to 16 slave interfaces
- slaves are mapped into master adress spaces to distinguish accesses
- **Shared-Address-Multiple-Data (SAMD)** topology
- recall: *AXI is not a bus!* point-to-point
- crossbar implements *time-multiplexed arbitration scheme*
- e.g., round-robin, priority-based, . . .
- switches occur at end of bursts
- dual channel, read/write separate

TaPaSCo
AXI4 Interconnect IP (Xilinx)

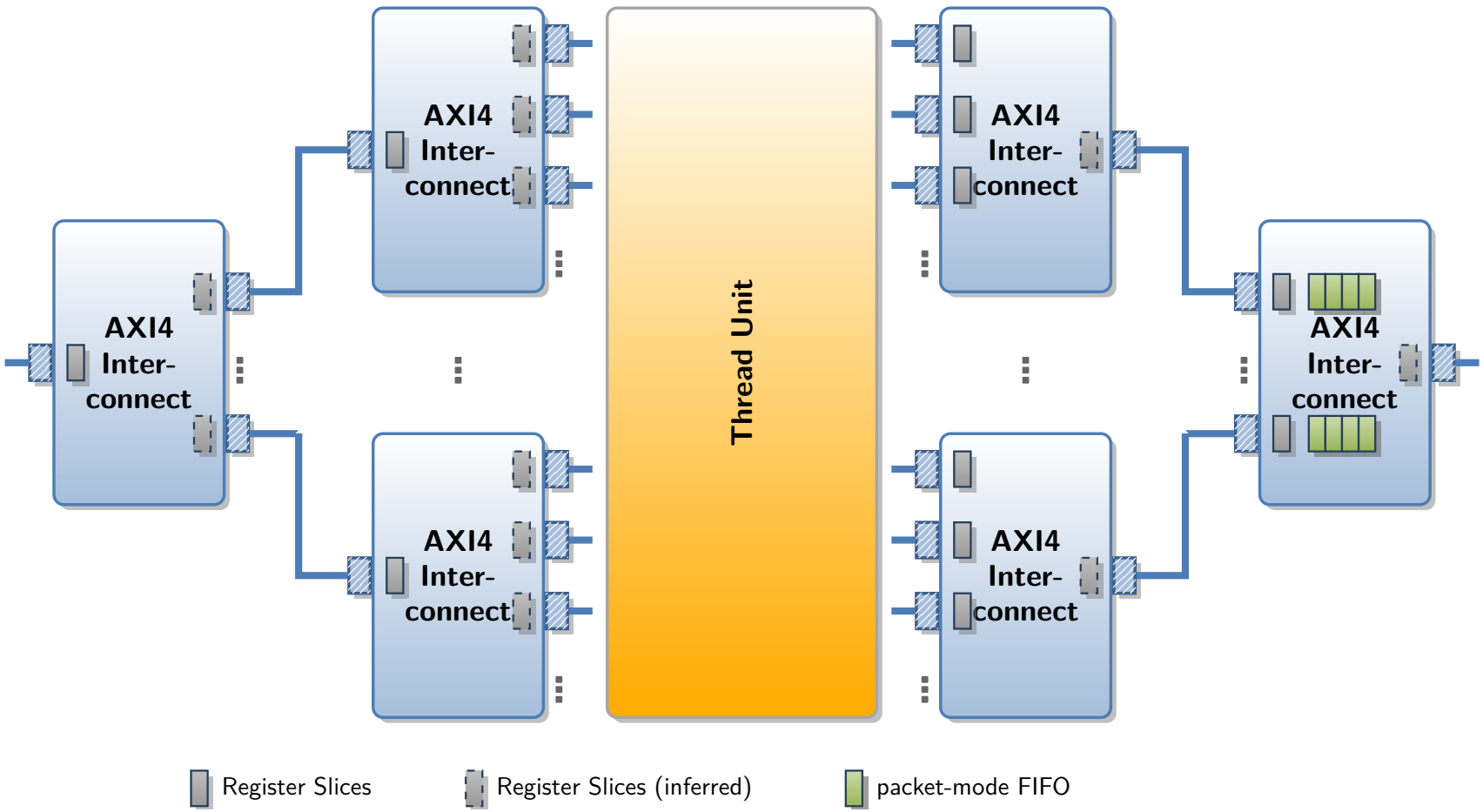
TECHNISCHE UNIVERSITÄT DARMSTADT

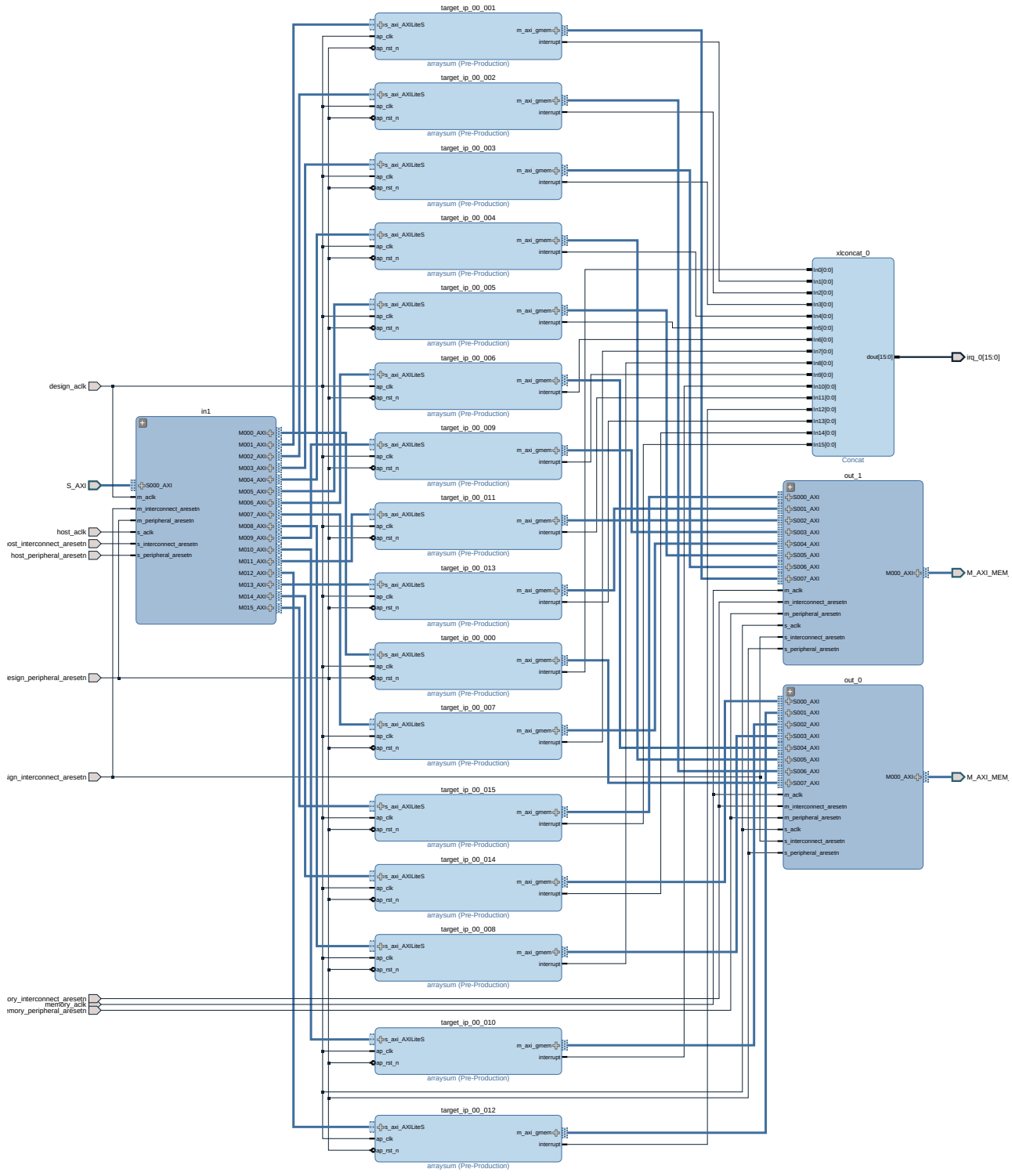


- more complicated: provides optional **couplers**
  - data width converter *(e.g., 32b → 8b)*
  - clock converter *(e.g., 250 MHz → 100 Mhz)*
  - protocol converter *(e.g., AXI4 → AXI4Lite)*
  - register slice *(relax critical paths)*
  - shallow/deep data FIFO *(latency-insensitive decoupling)*
  - AXI MMU
- AXI workhorse — highly versatile, high performance module
- simplifies AXI designs significantly

TaPaSCo
Concrete Architecture: baseline

TECHNISCHE UNIVERSITÄT DARMSTADT




target_ip_00_001
target_ip_00_002
target_ip_00_003
target_ip_00_004
target_ip_00_005
target_ip_00_006
target_ip_00_009
target_ip_00_011
target_ip_00_013
target_ip_00_000
target_ip_00_007
target_ip_00_015
target_ip_00_014
target_ip_00_008
target_ip_00_010
target_ip_00_012
s_axi_AXILiteS
ap_clk
ap_rst_n
m_axi_gmem
interrupt
arraysum (Pre-Production)
xlconcat_0
dout[15:0]
Concat
irq_0[15:0]
design_aclk
in1
S_AXI
host_aclk
M000_AXI
M001_AXI
M002_AXI
M003_AXI
M004_AXI
M005_AXI
M006_AXI
M007_AXI
M008_AXI
M009_AXI
M010_AXI
M011_AXI
M012_AXI
M013_AXI
M014_AXI
M015_AXI
S000_AXI
m_aclk
m_interconnect_aresetn
m_peripheral_aresetn
s_aclk
s_interconnect_aresetn
s_peripheral_aresetn
out_1
out_0
M_AXI_MEM_
S001_AXI
S002_AXI
S003_AXI
S004_AXI
S005_AXI
S006_AXI
S007_AXI

TaPaSCo
Address Map: PE Slaves and Masters

TECHNISCHE UNIVERSITÄT DARMSTADT

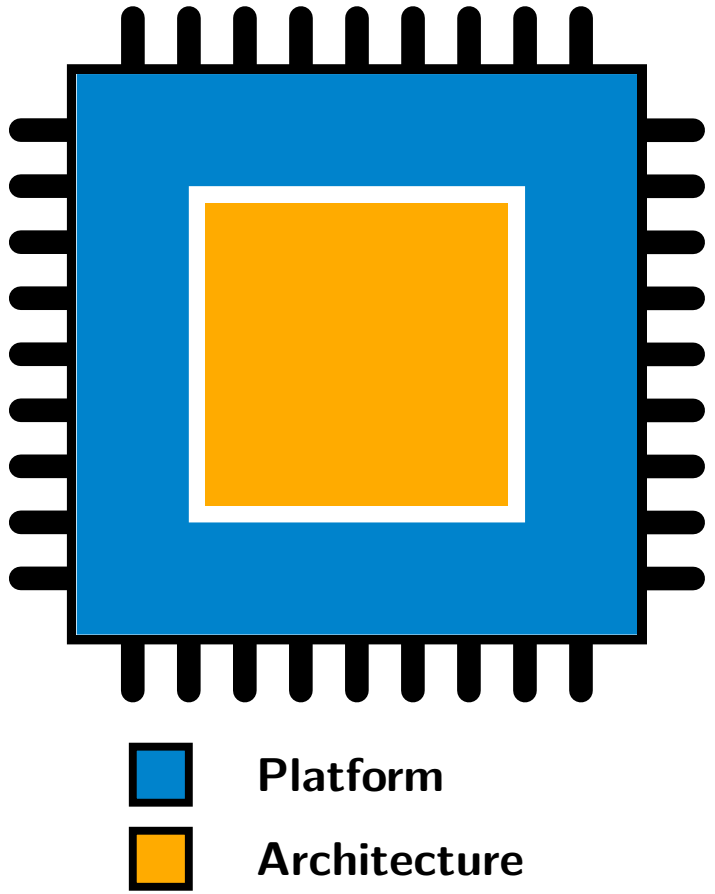
Host/ps7					
Data (32 address bits : 0x40000000 [1G] ,0x80000000 [1G])					
InterruptControl/axi_intc_00	s_axi	Reg	0x8180_0000	64K	0x8180_FFFF
Threadpool/target_ip_00_000	s_axi_AXILiteS	Reg	0x43C0_0000	64K	0x43C0_FFFF
Threadpool/target_ip_00_001	s_axi_AXILiteS	Reg	0x43C1_0000	64K	0x43C1_FFFF
Threadpool/target_ip_00_002	s_axi_AXILiteS	Reg	0x43C2_0000	64K	0x43C2_FFFF
Threadpool/target_ip_00_003	s_axi_AXILiteS	Reg	0x43C3_0000	64K	0x43C3_FFFF
Threadpool/target_ip_00_004	s_axi_AXILiteS	Reg	0x43C4_0000	64K	0x43C4_FFFF
Threadpool/target_ip_00_005	s_axi_AXILiteS	Reg	0x43C5_0000	64K	0x43C5_FFFF
Threadpool/target_ip_00_006	s_axi_AXILiteS	Reg	0x43C6_0000	64K	0x43C6_FFFF
Threadpool/target_ip_00_007	s_axi_AXILiteS	Reg	0x43C7_0000	64K	0x43C7_FFFF
Threadpool/target_ip_00_008	s_axi_AXILiteS	Reg	0x43C8_0000	64K	0x43C8_FFFF
Threadpool/target_ip_00_009	s_axi_AXILiteS	Reg	0x43C9_0000	64K	0x43C9_FFFF
Threadpool/target_ip_00_010	s_axi_AXILiteS	Reg	0x43CA_0000	64K	0x43CA_FFFF
Threadpool/target_ip_00_011	s_axi_AXILiteS	Reg	0x43CB_0000	64K	0x43CB_FFFF
Threadpool/target_ip_00_012	s_axi_AXILiteS	Reg	0x43CC_0000	64K	0x43CC_FFFF
Threadpool/target_ip_00_013	s_axi_AXILiteS	Reg	0x43CD_0000	64K	0x43CD_FFFF
Threadpool/target_ip_00_014	s_axi_AXILiteS	Reg	0x43CE_0000	64K	0x43CE_FFFF
Threadpool/target_ip_00_015	s_axi_AXILiteS	Reg	0x43CF_0000	64K	0x43CF_FFFF
tpc_status	S00_AXI	S00_AXI_reg	0x7777_0000	64K	0x7777_FFFF
Unconnected Slaves					
Host/ps7	S_AXI_ACP	ACP_DDR_LO...			
Host/ps7	S_AXI_ACP	ACP_QSPI_LIN...			
Host/ps7	S_AXI_ACP	ACP_IOP			
Host/ps7	S_AXI_ACP	ACP_M_AXI_G...			
Host/ps7	S_AXI_ACP	ACP_M_AXI_G...			

Threadpool/target_ip_00_000						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP0	HP0_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_001						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP2	HP2_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_002						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP0	HP0_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_003						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP2	HP2_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_004						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP0	HP0_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_005						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP2	HP2_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_006						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP0	HP0_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_007						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP2	HP2_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_008						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP0	HP0_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_009						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP2	HP2_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_010						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP0	HP0_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_011						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP2	HP2_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_012						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP0	HP0_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_013						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP2	HP2_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_014						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP0	HP0_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF
Threadpool/target_ip_00_015						
Data_m_axi_gmem (32 address bits : 4G)						
Host/ps7	S_AXI_HP2	HP2_DDR_LO...	0x0000_0000	512M		0x1FFF_FFFF

TaPaSCo
Design Abstractions

TECHNISCHE UNIVERSITÄT DARMSTADT

- TaPaSCo designs are split into two distinct parts:
  1. **Architecture**
     - thread units and PEs, according to Composition
     - interconnections (e.g., AXI4-based)
     - independent of target platform / board
  2. **Platform**
     - connection to host and memory
     - hardware-dependent
- advantage: hardware-dependent parts are isolated in **Platform**
- represented in TaPaSCo as plug-in scripts
  - easy to modify / re-use existing scripts
  - easy to add new **Platforms** and **Architectures**



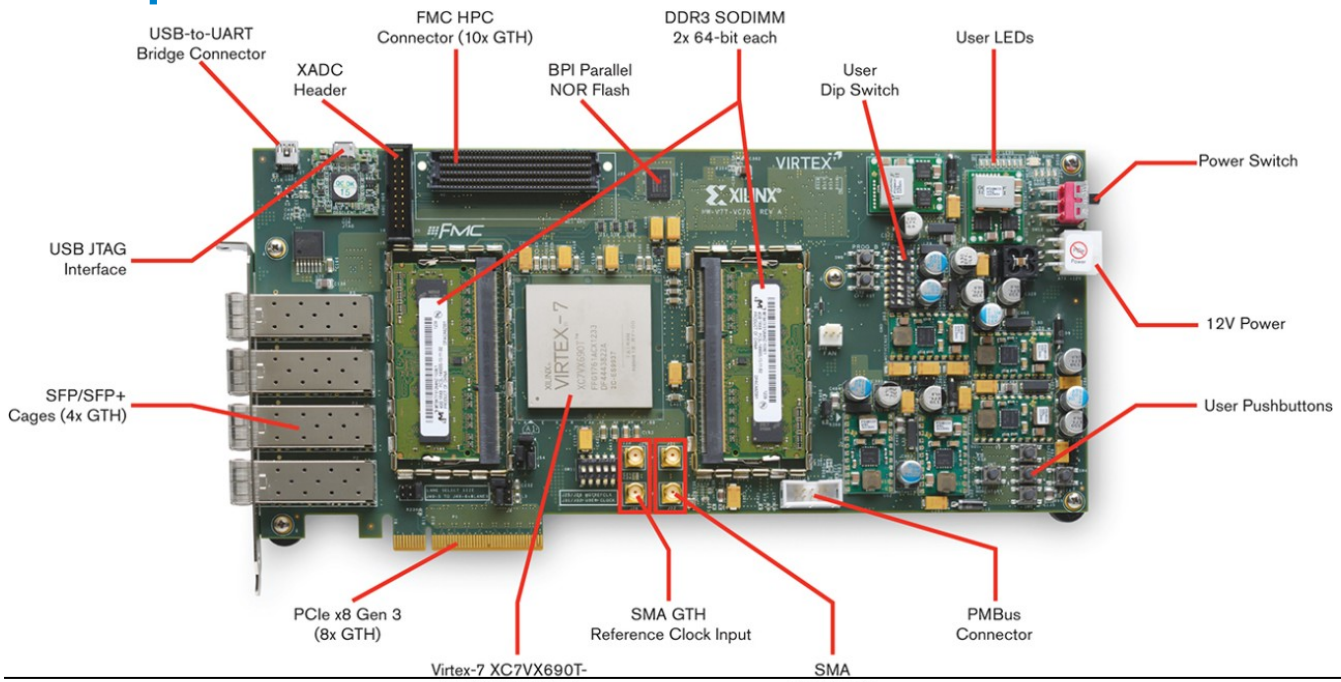
TaPaSCo
Platform Scripts

TECHNISCHE UNIVERSITÄT DARMSTADT

Tasks

1. provide control connection from host to PEs
2. provide PEs with access to memory
3. provide signaling interface from PEs to host
4. *optional: instantiate additional hardware infrastructure*

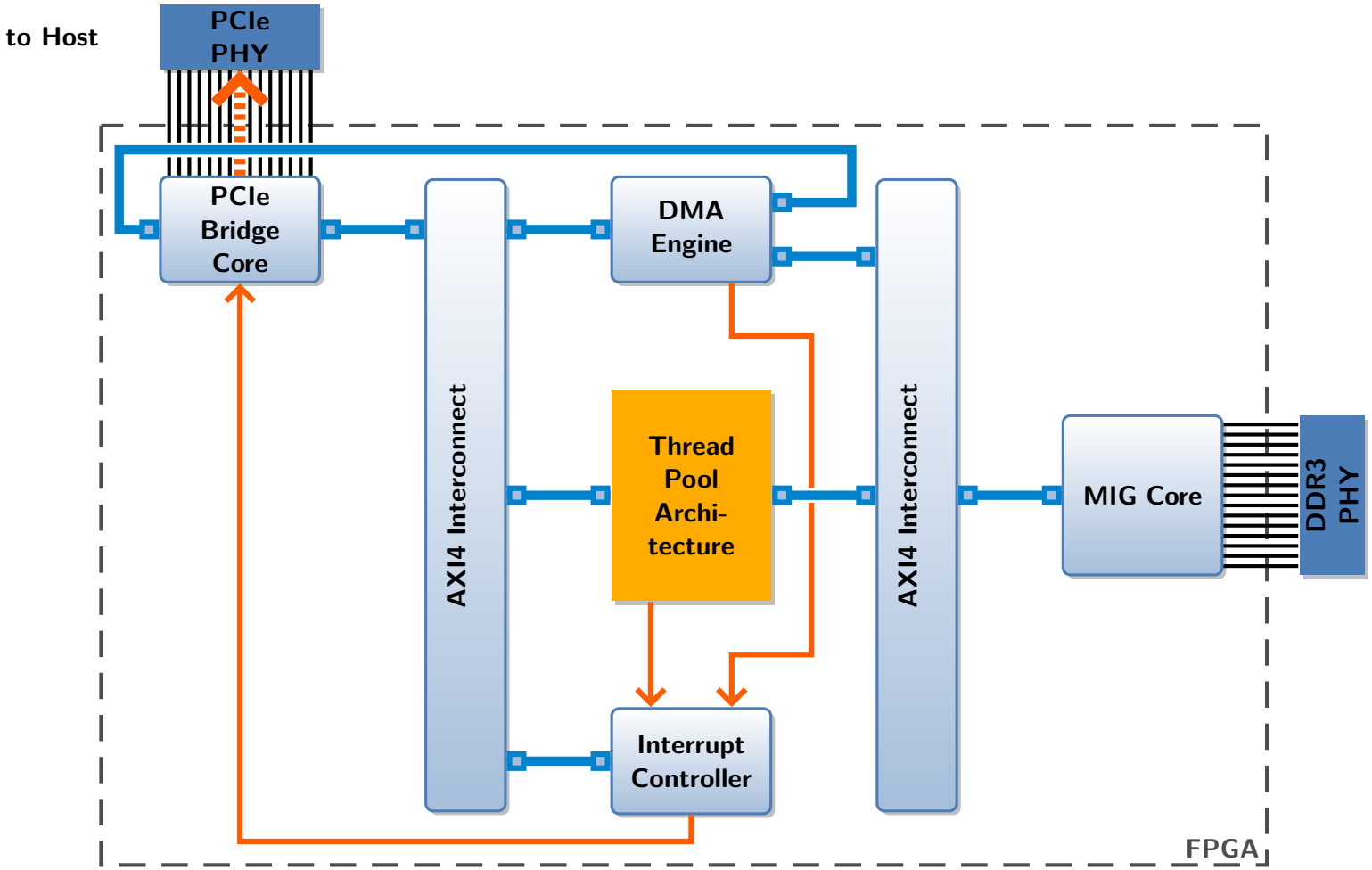
Example Platform: VC709



- *host connection:* PCIe Gen3 x8
- *memory:* 2×4 GiB DDR3
- *signaling:* MSI-X packetized in-band interrupts
- *other:* User LEDs (GPIO)

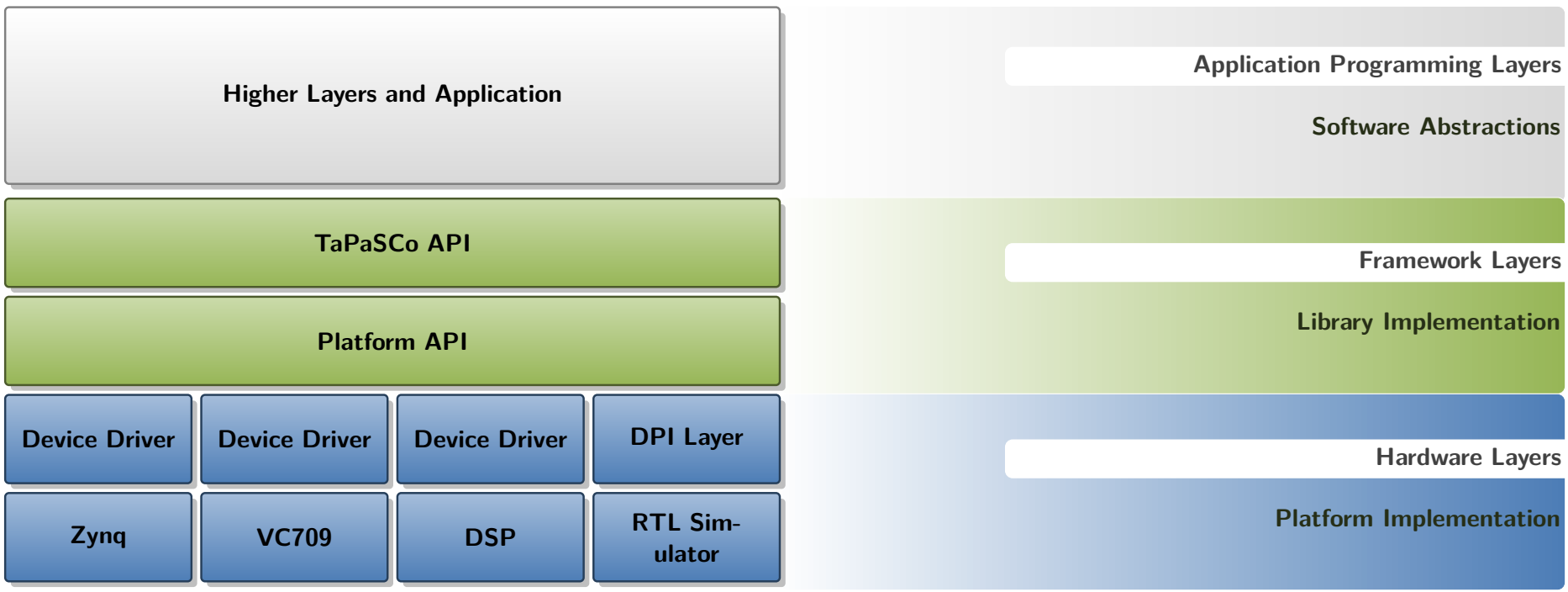
TaPaSCo
Concrete Platform: VC709

TECHNISCHE UNIVERSITÄT DARMSTADT



TaPaSCo
Software Stack

TECHNISCHE UNIVERSITÄT DARMSTADT



TaPaSCo
OS-level Integration

TECHNISCHE UNIVERSITÄT DARMSTADT

- **device driver** provides OS-level integration
  - on-chip address space is mapped into global/bus address space
  - each slave interface has *physical address range*
  - hardware functions are accessible for all bus devices (exactly like "regular" hardware)
- in case of monolithic kernels (Linux, Windows, Mac OS):
  - driver code becomes part of the kernel, executes with Ring-0 privilege
  - security-critical, can crash entire system
  - tedious to develop!
- OS provides facilities to allow and control userspace access
  - physical ranges can be mapped into *virtual address spaces*
  - driver can permit access to *special files*
- *. . . no details here, will be discussed in OS lecture*

TaPaSCo
Platform Library

TECHNISCHE UNIVERSITÄT DARMSTADT

Platform API

- software counterpart to Platform on-chip
- minimal userspace abstraction over the device driver layer
- low-level integration tasks:
  - read/write control registers
    1. `platform_read_ctl`
    2. `platform_write_ctl`
  - manage + read/write device memory
    1. `platform_alloc`
    2. `platform_dealloc`
    3. `platform_read_mem`
    4. `platform_write_mem`
  - wait for signals
    1. `platform_wait_for_irq`
    2. `platform_write_ctl_and_wait`
  - query device address space
    1. `platform_address_get_pe_base`
    2. `platform_address_get_infrastructure_base`
  - . . . and initialization/administrative functions

TaPaSCo
Architecture Library

TECHNISCHE UNIVERSITÄT DARMSTADT

TaPaSCo API

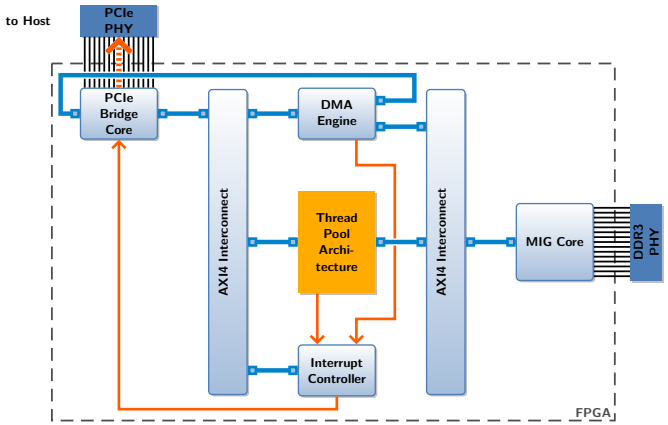
- user-facing API for TaPaSCo applications
- device memory management
  1. `tapasco_device_alloc` — wrapper for `platform_alloc`
  2. `tapasco_device_dealloc` — wrapper for `platform_dealloc`
  3. `tapasco_device_copy_to` — wrapper for `platform_write_mem`
  4. `tapasco_device_copy_from` — wrapper for `platform_read_mem`
- high-level task management
  1. `tapasco_device_acquire_job_id`
  2. `tapasco_device_release_job_id`
  3. `tapasco_device_job_set_arg`
  4. `tapasco_device_job_get_arg`
  5. `tapasco_device_job_get_return`
  6. `tapasco_device_job_launch`
- . . . and initialization/administrative functions

TaPaSCo
Example Application
int some_kernel(uint8_t data[1024], const int x)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
tapasco_device_t dev = ...;
const int x = 42;
tapasco_handle_t h = tapasco_device_alloc(dev, 1024, 0);

tapasco_device_copy_to(dev, data, h, 1024, TAPASCO_BLOCKING_MODE);

tapasco_job_id_t j_id = tapasco_device_acquire_job_id(dev, "some_kernel");

tapasco_device_job_set_arg(dev, j_id, 0, sizeof(h), &h);
tapasco_device_job_set_arg(dev, j_id, 1, sizeof(x), &x);

tapasco_device_job_launch(dev, j_id, TAPASCO_BLOCKING_MODE);

int r = 0;
tapasco_device_job_get_return(dev, j_id, sizeof(r), &r);
tapasco_device_copy_from(dev, h, &data, 1024, TAPASCO_BLOCKING_MODE);

printf("result of job: %d\n", r);

tapasco_device_release_job_id(dev, j_id);

tapasco_device_free(dev, h);
```

TaPaSCo
Example Application
int some_kernel(uint8_t data[1024], const int x)

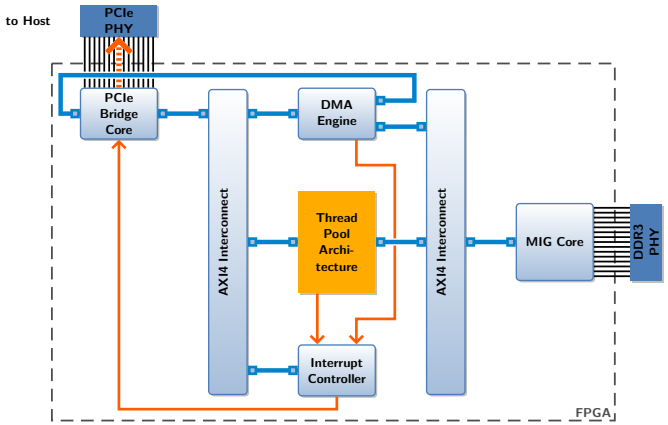



```
tapasco_device_t dev = ...;
const int x = 42;
tapasco_handle_t h = tapasco_device_alloc(dev, 1024, 0);

tapasco_device_copy_to(dev, data, h, 1024, TAPASCO_BLOCKING_MODE);

tapasco_job_id_t j_id = tapasco_device_acquire_job_id(dev, "some_kernel");

tapasco_device_job_set_arg(dev, j_id, 0, sizeof(h), &h);
tapasco_device_job_set_arg(dev, j_id, 1, sizeof(x), &x);

tapasco_device_job_launch(dev, j_id, TAPASCO_BLOCKING_MODE);

int r = 0;
tapasco_device_job_get_return(dev, j_id, sizeof(r), &r);
tapasco_device_copy_from(dev, h, &data, 1024, TAPASCO_BLOCKING_MODE);

printf("result of job: %d\n", r);

tapasco_device_release_job_id(dev, j_id);

tapasco_device_free(dev, h);
```

`tapasco_device_alloc`

1. TaPaSCo library forwards memory allocation request to Platform library
2. Platform library allocates memory on device (via *three-tiered buddy allocation*)
   - `tapasco_handle_t` contains the device address

TaPaSCo
Example Application
int some_kernel(uint8_t data[1024], const int x)

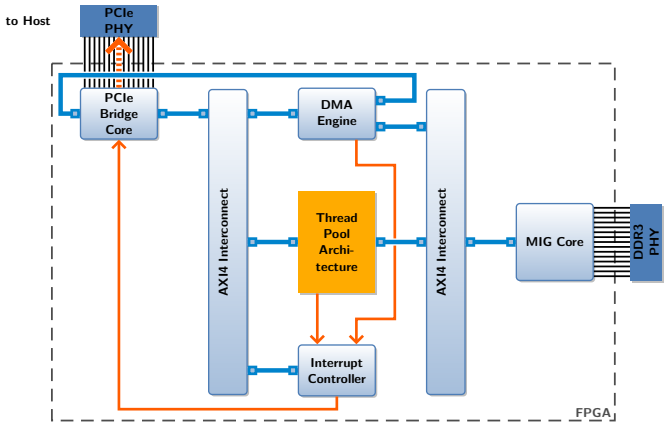



```
tapasco_device_t dev = ...;
const int x = 42;
tapasco_handle_t h = tapasco_device_alloc(dev, 1024, 0);

tapasco_device_copy_to(dev, data, h, 1024, TAPASCO_BLOCKING_MODE);

tapasco_job_id_t j_id = tapasco_device_acquire_job_id(dev, "some_kernel");

tapasco_device_job_set_arg(dev, j_id, 0, sizeof(h), &h);
tapasco_device_job_set_arg(dev, j_id, 1, sizeof(x), &x);

tapasco_device_job_launch(dev, j_id, TAPASCO_BLOCKING_MODE);

int r = 0;
tapasco_device_job_get_return(dev, j_id, sizeof(r), &r);
tapasco_device_copy_from(dev, h, &data, 1024, TAPASCO_BLOCKING_MODE);

printf("result of job: %d\n", r);

tapasco_device_release_job_id(dev, j_id);

tapasco_device_free(dev, h);
```

`tapasco_device_copy_to`

1. device driver copies data from virtual address in user space to *kernel space buffer*
2. device driver translates *kernel virtual address* to *physical/bus address*
3. device driver sets DMA engine registers:
   - 3.1 kernel buffer physical base address
   - 3.2 kernel buffer length
   - 3.3 device memory destination address
   - 3.4 copy direction: to device
4. DMA engine copies data
5. DMA engine raises interrupt to signal host

TaPaSCo
Example Application
int some_kernel(uint8_t data[1024], const int x)

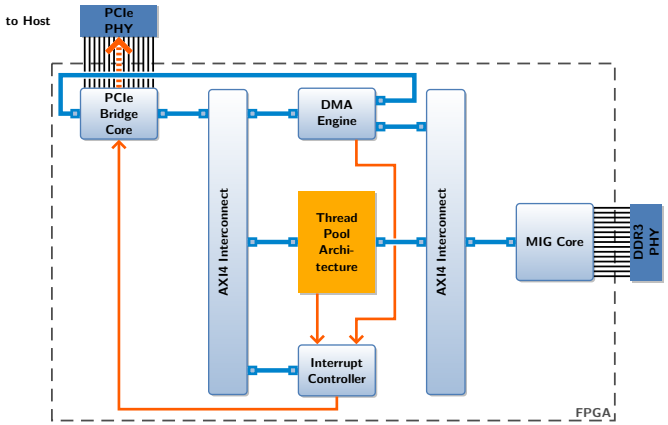


tapasco_device_acquire_job_id

1. TaPaSCo library acquires *job object* userspace struct, contains
   - thread unit ID
   - buffers for all argument values

```
tapasco_device_t dev = ...;
const int x = 42;
tapasco_handle_t h = tapasco_device_alloc(dev, 1024, 0);

tapasco_device_copy_to(dev, data, h, 1024, TAPASCO_BLOCKING_MODE);

tapasco_job_id_t j_id = tapasco_device_acquire_job_id(dev, "some_kernel");

tapasco_device_job_set_arg(dev, j_id, 0, sizeof(h), &h);
tapasco_device_job_set_arg(dev, j_id, 1, sizeof(x), &x);

tapasco_device_job_launch(dev, j_id, TAPASCO_BLOCKING_MODE);

int r = 0;
tapasco_device_job_get_return(dev, j_id, sizeof(r), &r);
tapasco_device_copy_from(dev, h, &data, 1024, TAPASCO_BLOCKING_MODE);

printf("result of job: %d\n", r);

tapasco_device_release_job_id(dev, j_id);

tapasco_device_free(dev, h);
```

TaPaSCo
Example Application
int some_kernel(uint8_t data[1024], const int x)

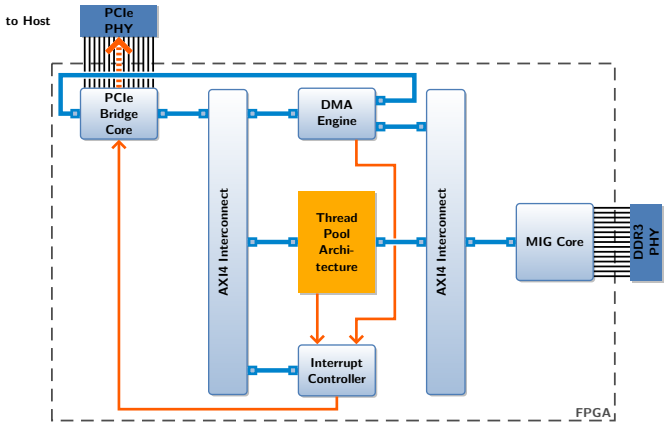


tapasco_device_job_set_arg

1. application prepares job in memory
   - sets argument values in job object

```
tapasco_device_t dev = ...;
const int x = 42;
tapasco_handle_t h = tapasco_device_alloc(dev, 1024, 0);

tapasco_device_copy_to(dev, data, h, 1024, TAPASCO_BLOCKING_MODE);

tapasco_job_id_t j_id = tapasco_device_acquire_job_id(dev, "some_kernel");

tapasco_device_job_set_arg(dev, j_id, 0, sizeof(h), &h);
tapasco_device_job_set_arg(dev, j_id, 1, sizeof(x), &x);

tapasco_device_job_launch(dev, j_id, TAPASCO_BLOCKING_MODE);

int r = 0;
tapasco_device_job_get_return(dev, j_id, sizeof(r), &r);
tapasco_device_copy_from(dev, h, &data, 1024, TAPASCO_BLOCKING_MODE);

printf("result of job: %d\n", r);

tapasco_device_release_job_id(dev, j_id);

tapasco_device_free(dev, h);
```

TaPaSCo
Example Application
int some_kernel(uint8_t data[1024], const int x)

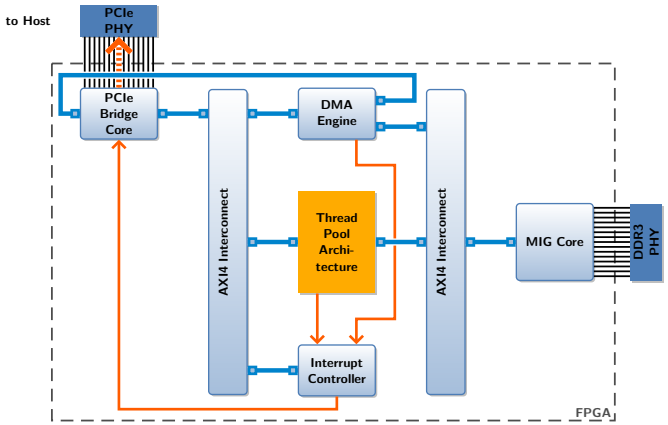



```
tapasco_device_t dev = ...;
const int x = 42;
tapasco_handle_t h = tapasco_device_alloc(dev, 1024, 0);

tapasco_device_copy_to(dev, data, h, 1024, TAPASCO_BLOCKING_MODE);

tapasco_job_id_t j_id = tapasco_device_acquire_job_id(dev, "some_kernel");

tapasco_device_job_set_arg(dev, j_id, 0, sizeof(h), &h);
tapasco_device_job_set_arg(dev, j_id, 1, sizeof(x), &x);

tapasco_device_job_launch(dev, j_id, TAPASCO_BLOCKING_MODE);

int r = 0;
tapasco_device_job_get_return(dev, j_id, sizeof(r), &r);
tapasco_device_copy_from(dev, h, &data, 1024, TAPASCO_BLOCKING_MODE);

printf("result of job: %d\n", r);

tapasco_device_release_job_id(dev, j_id);

tapasco_device_free(dev, h);
```

tapasco_device_job_launch

1. application submits job object to pool
2. TaPaSCo library acquires first idle PE in the thread unit
3. TaPaSCo library writes register values (AXI4Lite register file) via Platform library
4. TaPaSCo library writes start bit and waits for interrupt via Platform library
5. TaPaSCo library copies back any register values and arguments to the job object
6. TaPaSCo library releases the PE

TaPaSCo
Example Application
int some_kernel(uint8_t data[1024], const int x)

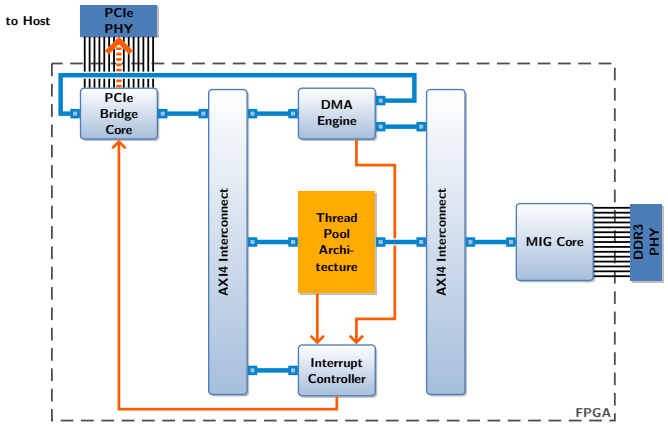


tapasco_device_job_get_return

1. application fetches return value from job object

```
tapasco_device_t dev = ...;
const int x = 42;
tapasco_handle_t h = tapasco_device_alloc(dev, 1024, 0);

tapasco_device_copy_to(dev, data, h, 1024, TAPASCO_BLOCKING_MODE);

tapasco_job_id_t j_id = tapasco_device_acquire_job_id(dev, "some_kernel");

tapasco_device_job_set_arg(dev, j_id, 0, sizeof(h), &h);
tapasco_device_job_set_arg(dev, j_id, 1, sizeof(x), &x);

tapasco_device_job_launch(dev, j_id, TAPASCO_BLOCKING_MODE);

int r = 0;
tapasco_device_job_get_return(dev, j_id, sizeof(r), &r);
tapasco_device_copy_from(dev, h, &data, 1024, TAPASCO_BLOCKING_MODE);

printf("result of job: %d\n", r);

tapasco_device_release_job_id(dev, j_id);

tapasco_device_free(dev, h);
```

TaPaSCo
Example Application
int some_kernel(uint8_t data[1024], const int x)

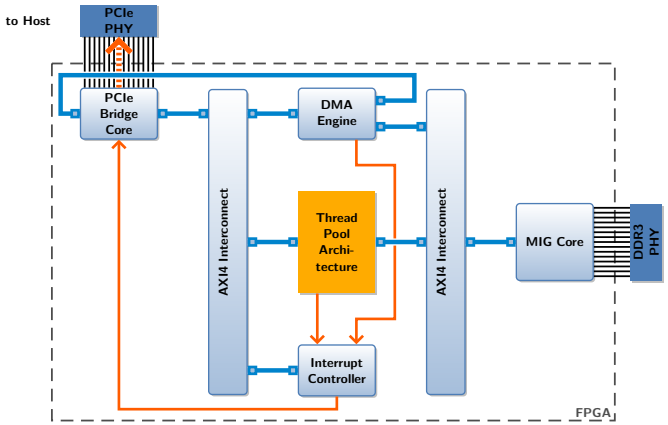



```
tapasco_device_t dev = ...;
const int x = 42;
tapasco_handle_t h = tapasco_device_alloc(dev, 1024, 0);

tapasco_device_copy_to(dev, data, h, 1024, TAPASCO_BLOCKING_MODE);

tapasco_job_id_t j_id = tapasco_device_acquire_job_id(dev, "some_kernel");

tapasco_device_job_set_arg(dev, j_id, 0, sizeof(h), &h);
tapasco_device_job_set_arg(dev, j_id, 1, sizeof(x), &x);

tapasco_device_job_launch(dev, j_id, TAPASCO_BLOCKING_MODE);

int r = 0;
tapasco_device_job_get_return(dev, j_id, sizeof(r), &r);
tapasco_device_copy_from(dev, h, &data, 1024, TAPASCO_BLOCKING_MODE);

printf("result of job: %d\n", r);

tapasco_device_release_job_id(dev, j_id);

tapasco_device_free(dev, h);
```

`tapasco_device_copy_from`

1. device driver reserves kernel buffer
2. device driver sets DMA engine registers:
   - 2.1 device memory source address
   - 2.2 kernel buffer length
   - 2.3 kernel buffer physical base address
   - 2.4 copy direction: from device
3. DMA engine copies data
4. DMA engine raises interrupt to signal host

TaPaSCo
Example Application
int some_kernel(uint8_t data[1024], const int x)

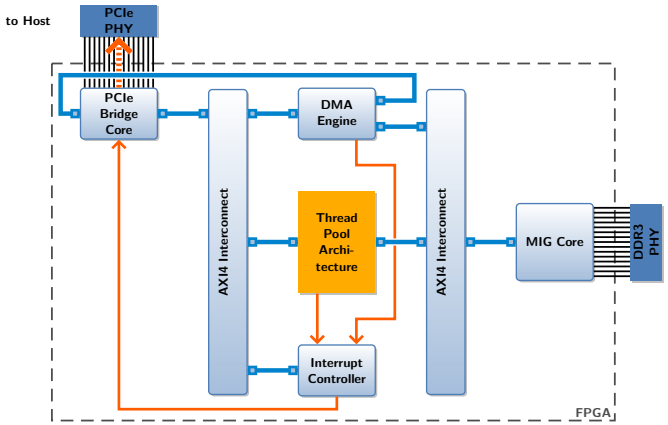


tapasco_device_release_job_id

1. TaPaSCo library releases job object

```
tapasco_device_t dev = ...;
const int x = 42;
tapasco_handle_t h = tapasco_device_alloc(dev, 1024, 0);

tapasco_device_copy_to(dev, data, h, 1024, TAPASCO_BLOCKING_MODE);

tapasco_job_id_t j_id = tapasco_device_acquire_job_id(dev, "some_kernel");

tapasco_device_job_set_arg(dev, j_id, 0, sizeof(h), &h);
tapasco_device_job_set_arg(dev, j_id, 1, sizeof(x), &x);

tapasco_device_job_launch(dev, j_id, TAPASCO_BLOCKING_MODE);

int r = 0;
tapasco_device_job_get_return(dev, j_id, sizeof(r), &r);
tapasco_device_copy_from(dev, h, &data, 1024, TAPASCO_BLOCKING_MODE);

printf("result of job: %d\n", r);

tapasco_device_release_job_id(dev, j_id);

tapasco_device_free(dev, h);
```

TaPaSCo
Example Application
int some_kernel(uint8_t data[1024], const int x)



tapasco_device_free

1. TaPaSCo library forwards memory release to Platform library
2. Platform library marks block as free in buddy allocator

```
tapasco_device_t dev = ...;
const int x = 42;
tapasco_handle_t h = tapasco_device_alloc(dev, 1024, 0);

tapasco_device_copy_to(dev, data, h, 1024, TAPASCO_BLOCKING_MODE);

tapasco_job_id_t j_id = tapasco_device_acquire_job_id(dev, "some_kernel");

tapasco_device_job_set_arg(dev, j_id, 0, sizeof(h), &h);
tapasco_device_job_set_arg(dev, j_id, 1, sizeof(x), &x);

tapasco_device_job_launch(dev, j_id, TAPASCO_BLOCKING_MODE);

int r = 0;
tapasco_device_job_get_return(dev, j_id, sizeof(r), &r);
tapasco_device_copy_from(dev, h, &data, 1024, TAPASCO_BLOCKING_MODE);

printf("result of job: %d\n", r);

tapasco_device_release_job_id(dev, j_id);

tapasco_device_free(dev, h);
```

TaPaSCo
Example Application
int some_kernel(uint8_t data[1024], const int x)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
tapasco_device_t dev = ...;
const int x = 42;
tapasco_handle_t h = tapasco_device_alloc(dev, 1024, 0);

tapasco_device_copy_to(dev, data, h, 1024, TAPASCO_BLOCKING_MODE);

tapasco_job_id_t j_id = tapasco_device_acquire_job_id(dev, "some_kernel");

tapasco_device_job_set_arg(dev, j_id, 0, sizeof(h), &h);
tapasco_device_job_set_arg(dev, j_id, 1, sizeof(x), &x);

tapasco_device_job_launch(dev, j_id, TAPASCO_BLOCKING_MODE);

int r = 0;
tapasco_device_job_get_return(dev, j_id, sizeof(r), &r);
tapasco_device_copy_from(dev, h, &data, 1024, TAPASCO_BLOCKING_MODE);

printf("result of job: %d\n", r);

tapasco_device_release_job_id(dev, j_id);

tapasco_device_free(dev, h);
```

TaPaSCo++ API: Example

C++11 to the rescue!

TECHNISCHE UNIVERSITÄT DARMSTADT

```
TaPaSCo tapasco();

int r = tapasco.launch("some_kernel", &data, x);

std::cout << "result of job: " << r << std::endl;
```

- C++ API facilitates rapid experimentation
- C API provides full access to lower levels for corner cases
- C API resembles *OpenCL* - no coincidence, reduce learning curve for developers

TaPaSCo
Challenges addressed by TaPaSCo

TECHNISCHE UNIVERSITÄT DARMSTADT

Portability

- isolated Platform from Architecture
- can reuse Architectures on all Platforms
- can reuse software code via API hierarchy

Scalability

- can control design frequency
- can control area utilization via composition
- can generate design automatically for frequency + composition pairs

. . . but what is the maximal pair for any given Platform?

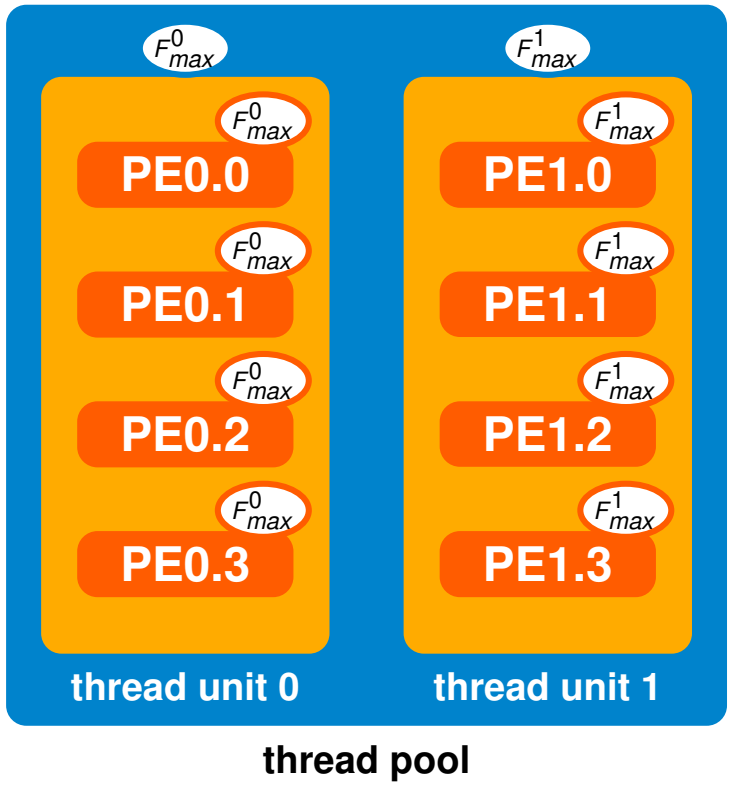
TaPaSCo
Estimating Upper Bounds: Design Frequency

TECHNISCHE UNIVERSITÄT DARMSTADT

Hierarchical Approach

- upper bound for each thread unit is given by upper bound of its PEs
- upper bound for pool is given by lowest upper bound of thread units

$$F_{max} = min(F^0_{max}, F^1_{max}, ...)$$



How to obtain an upper bound of F_{max} for a PE?

- perform synthesis + place and route in **out-of-context mode**
  - synthesize netlist for PE
  - place and route in fixed location
  - length of critical path $\rightarrow$ estimate for upper bound of frequency
  - also yields estimation of area!
- very few constraints for place and route, can be done quickly (minutes)
- ideal scenario $\rightarrow$ optimistic approximation

TaPaSCo
Estimating Upper Bounds: Area Utilization

TECHNISCHE UNIVERSITÄT DARMSTADT

Area Utilization

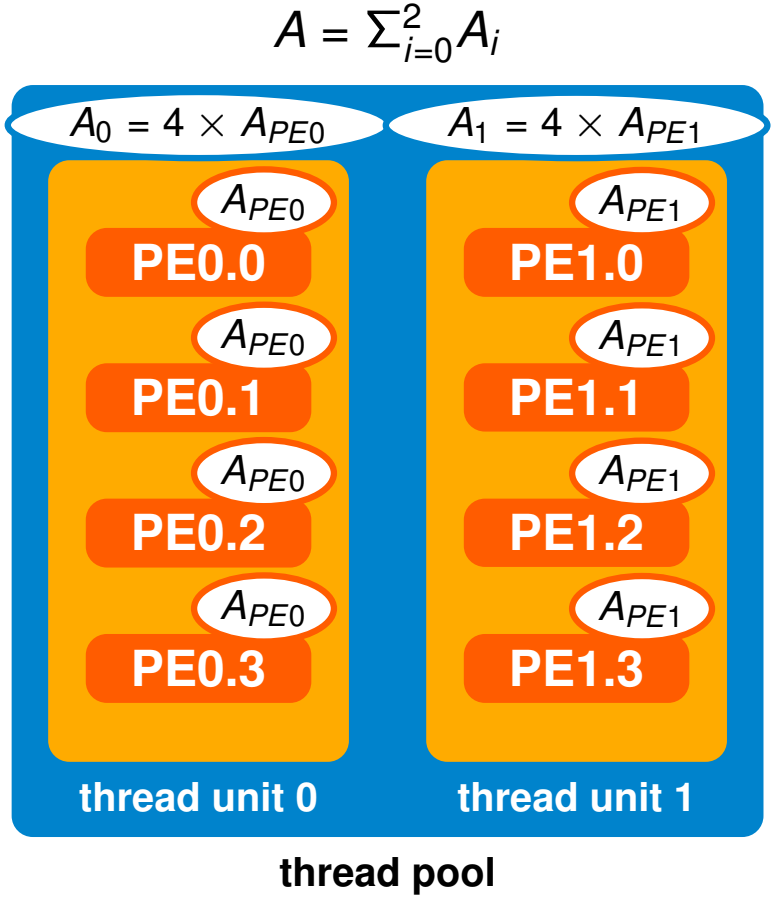
- measures the utilization of limited FPGA hardware resources
- area utilization $\sim$ spatial parallelism $\sim$ performance
- more precisely: one utilization factor per resource type
  - LUTs
  - FlipFlops
  - MUXes
  - BRAM
  - DSP slices
  - ...
- good approximation for amount of logic: LUTs

TaPaSCo
Estimating Upper Bounds: Area Utilization

TECHNISCHE UNIVERSITÄT DARMSTADT

Hierarchical Approach

- area estimation for each thread unit
  - area estimation for PEs $\times$ number of instances
  - plus Architecture overhead estimation
- area estimation for each thread pool
  - sum of area estimations for each thread unit
  - plus Architecture overhead estimation
- compare area estimation for pool to available resources
  - **feasibility:** $A \leq 1$
  - **optimality:** $A_{opt} = 1 - A$



TaPaSCo
Optimization Problem

TECHNISCHE UNIVERSITÄT DARMSTADT

Optimize Performance

- optimization of multiple variables
  - maximization of area utilization
  - maximization of design frequency
- standard approach: **heuristic function**
  - define mathematical function of all variables that maps to single value ($\sim$ performance estimation)
  - optimize this function instead (one dimension)
  - example: use product of max. frequency and area optimality
    $h : (F_{max}, A_{opt}) \mapsto F_{max} \cdot A_{opt}$
- **but: variables here are not independent!**
  - critical path length increases with area utilization $\Rightarrow$ design frequency decreases
  - increasing design frequency limits length of critical path $\Rightarrow$ limits area utilization

TaPaSCo
Optimization Problem

TECHNISCHE UNIVERSITÄT DARMSTADT

Optimize Performance

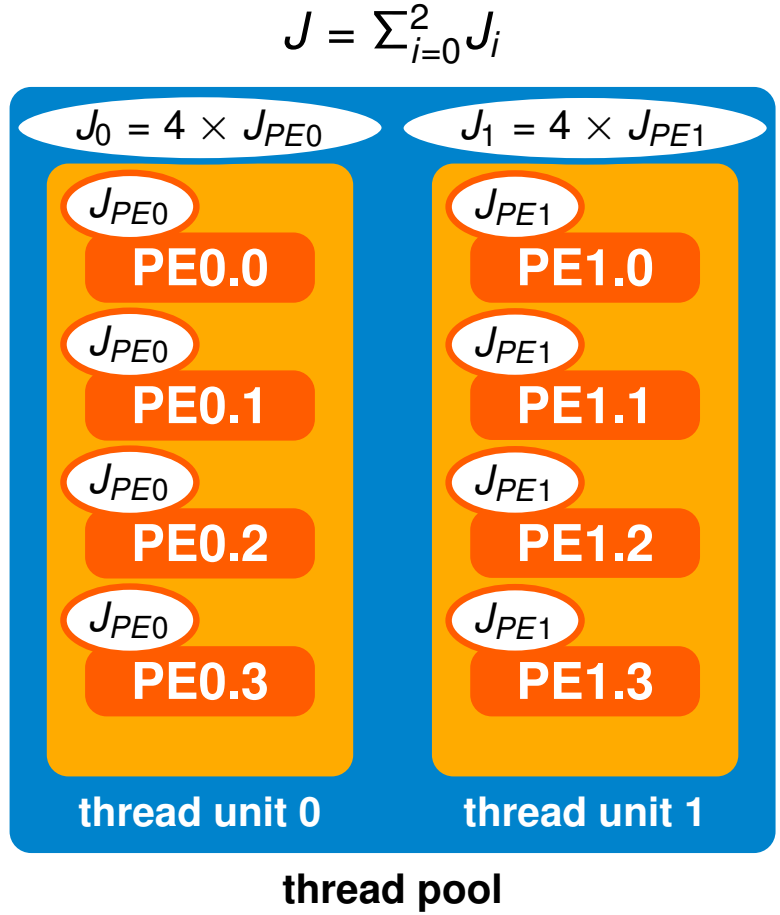
- step back: recall that TaPaSCo implements the *task parallel model* of computation
- what is a measure of performance for task parallelism?
- **job throughput** (average jobs/s)
- optimize job throughput instead

TaPaSCo
Estimating Upper Bounds: Job Throughput

TECHNISCHE UNIVERSITÄT DARMSTADT

Hierarchical Approach

- job throughput estimation for each unit
  - job throughput PEs $\times$ number of instances
  - minus Architecture/Platform overhead est.
- job throughput estimation for thread pool
  - sum of job throughput estimations for each thread unit
  - minus Architecture/Platform overhead est.

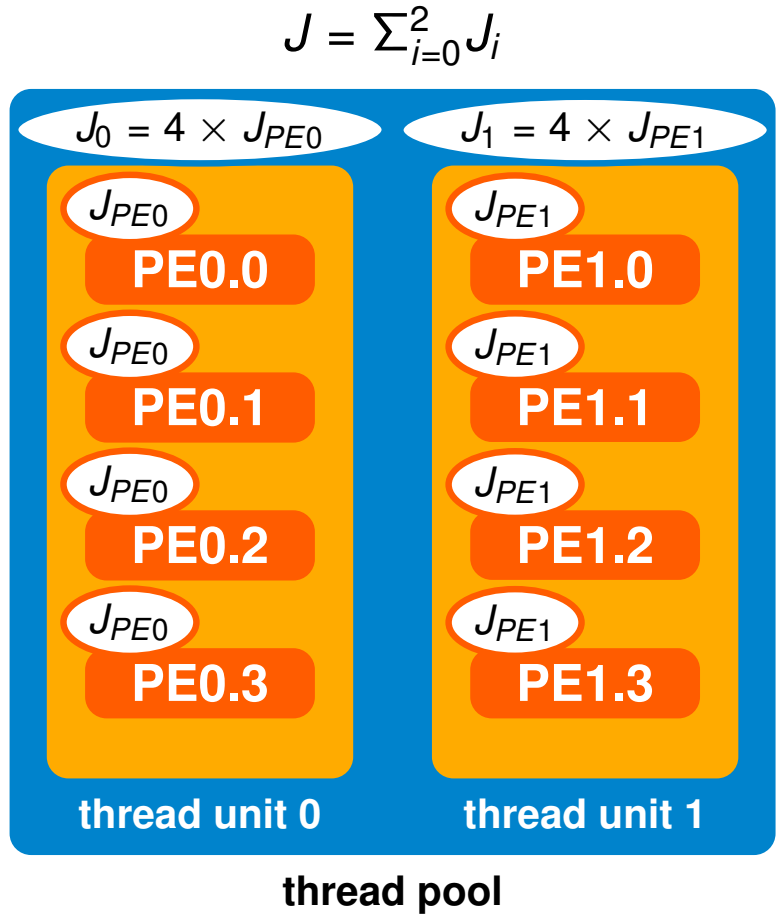


TaPaSCo
Estimating Upper Bounds: Job Throughput

TECHNISCHE UNIVERSITÄT DARMSTADT

Job Throughput for single PE

- job frequency = number of cycles per job * clock frequency
- number of cycles can be obtained by RTL simulation of PE
- max. frequency can be obtained by out-of-context approach
- what if number of cycles are input-dependent?
  - analyze typical workload for the application
  - use average number of cycles in typical workload for approximation



TaPaSCo
Example: PE Analysis

TECHNISCHE UNIVERSITÄT DARMSTADT

Out-of-context results for a simple PE

Clock Period (max.)		5.125	ns
Clock Frequency (max.)		195	MHz
	used / available		
LUTs	443 / 53200	0.833	%
FlipFlops	753 / 106400	0.708	%
BRAM Tiles	0.5 / 140	0.357	%
DSPs	0 / 220	0.000	%
Feasible #	$\lfloor 53200/443 \rfloor =$	120	PEs
Clock Cycles/Job		556	cycles
Job Period	$556 * 5.125ns =$	2849.5	ns
Job Frequency		$\approx$ 351	kHz
Heuristic Value		42 112 651	jobs/s

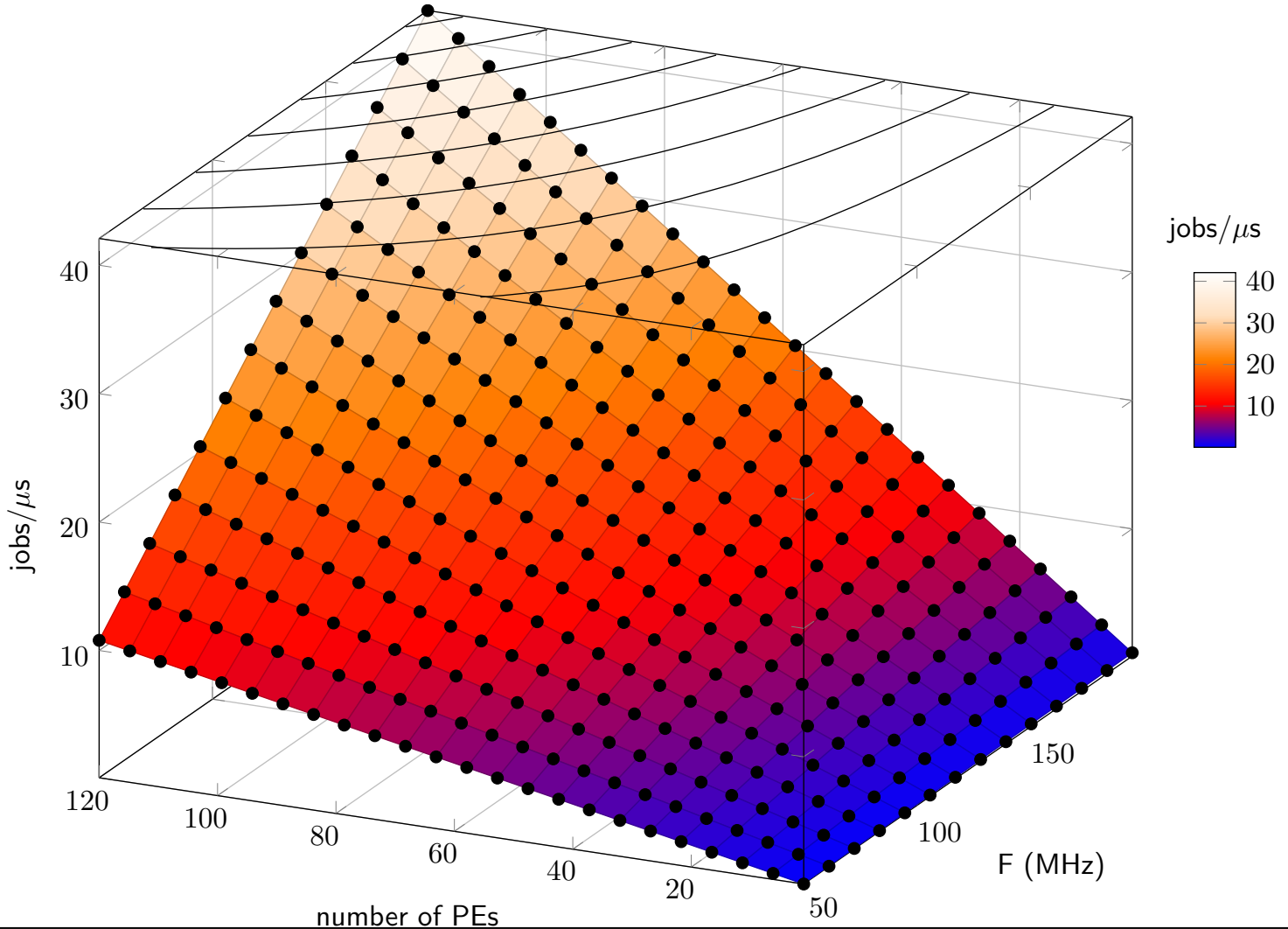
$\Rightarrow$ **120 PEs @ 195 MHz (best heuristic performance)**

Area utilization $\approx$ 99%, will not achieve timing closure

TaPaSCo
Exploring the Design Space

TECHNISCHE UNIVERSITÄT DARMSTADT

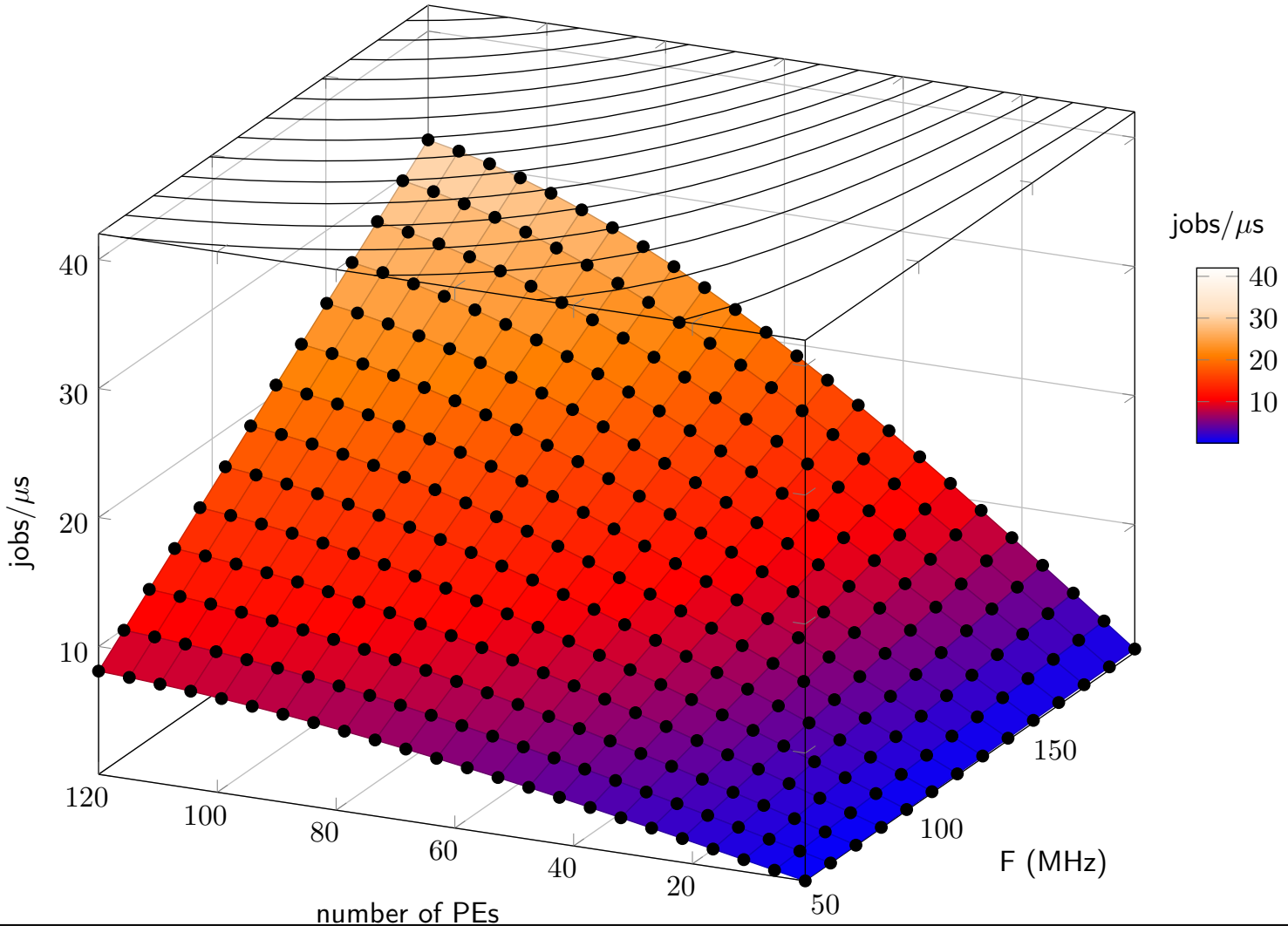
Assuming worst case performance penalty=0



TaPaSCo
Exploring the Design Space

TECHNISCHE UNIVERSITÄT DARMSTADT

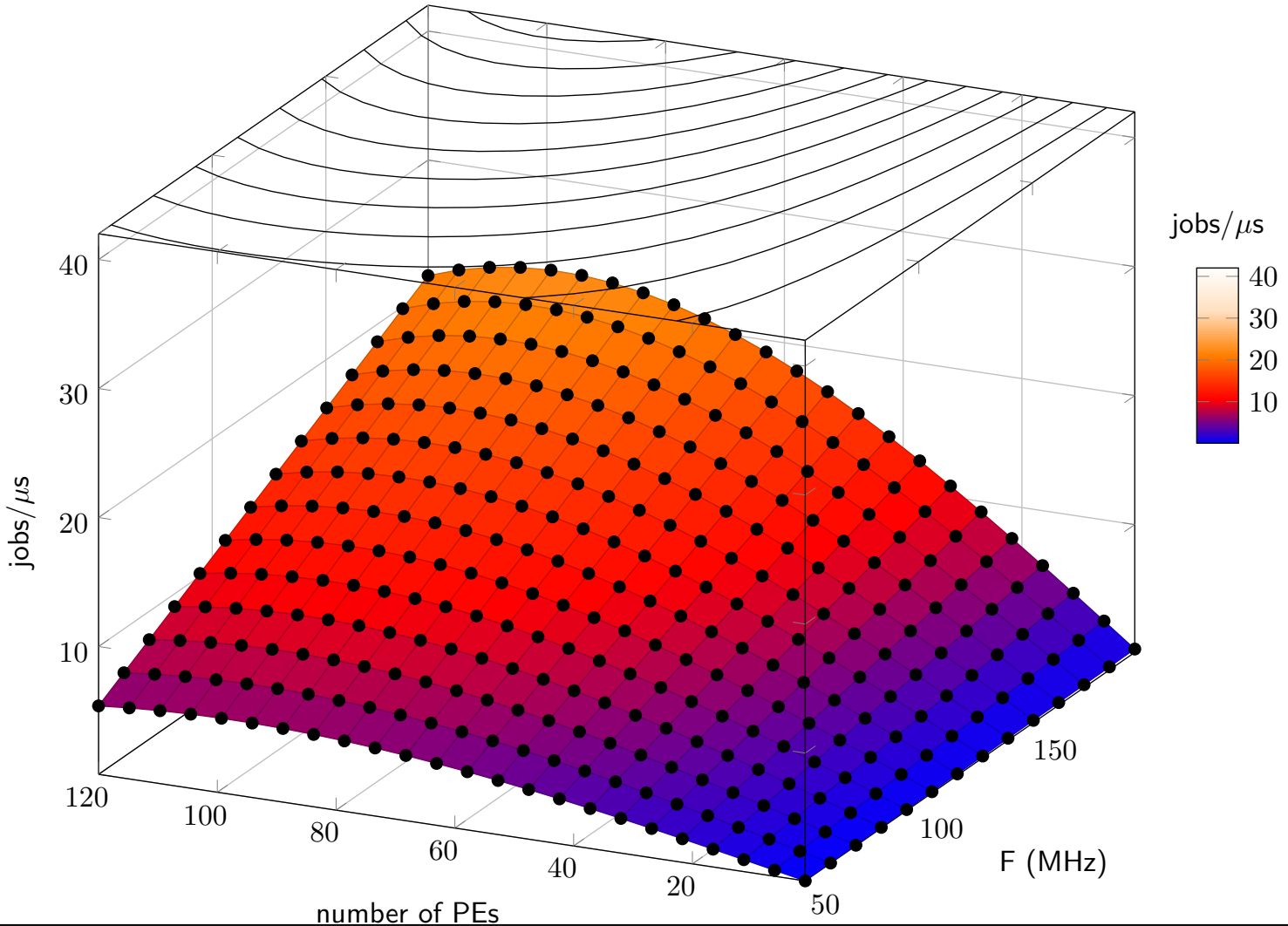
Assuming worst case performance penalty=25%



TaPaSCo
Exploring the Design Space

TECHNISCHE UNIVERSITÄT DARMSTADT

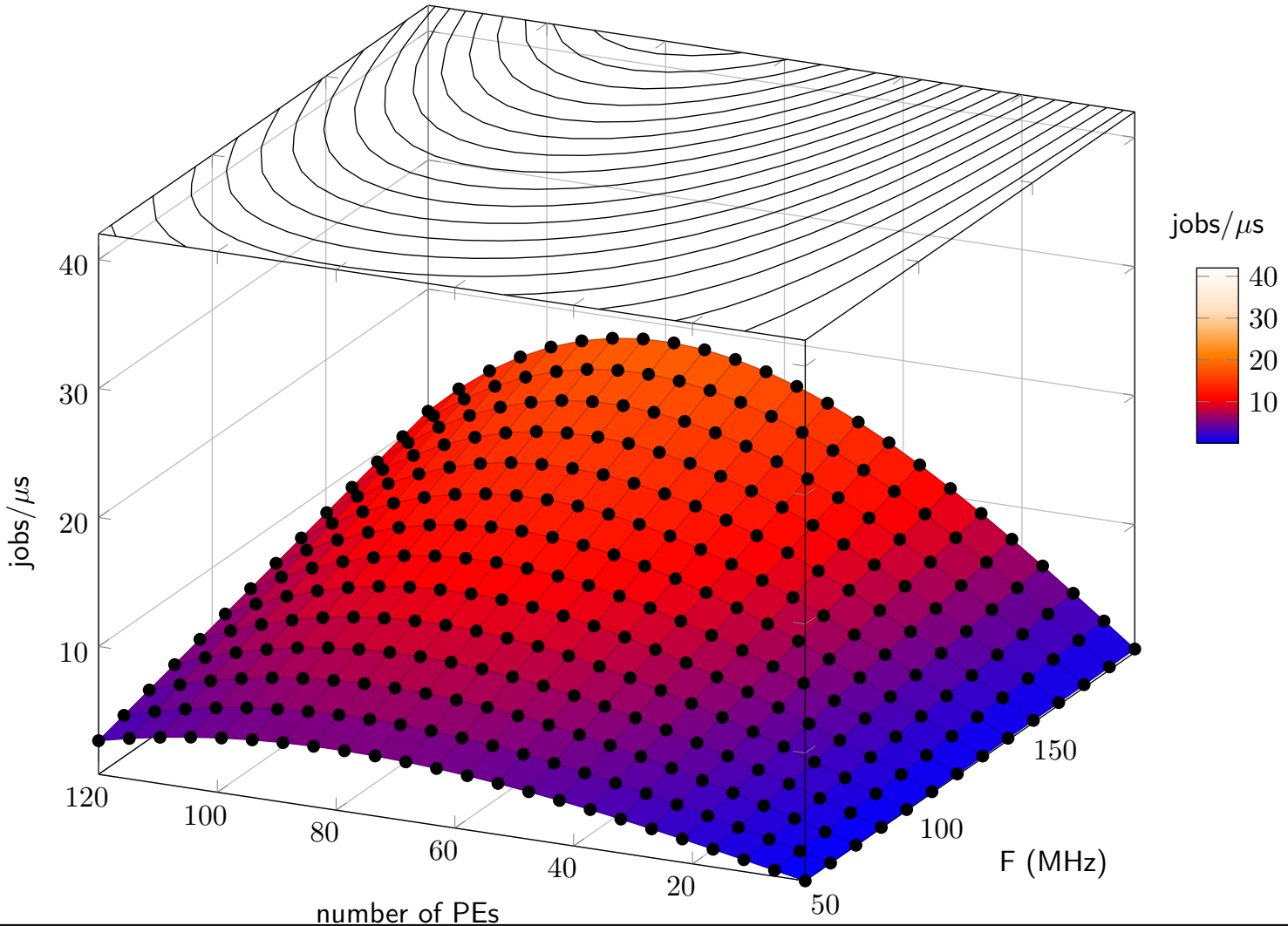
Assuming worst case performance penalty=50%



TaPaSCo
Exploring the Design Space

TECHNISCHE UNIVERSITÄT DARMSTADT

Assuming worst case performance penalty=75%

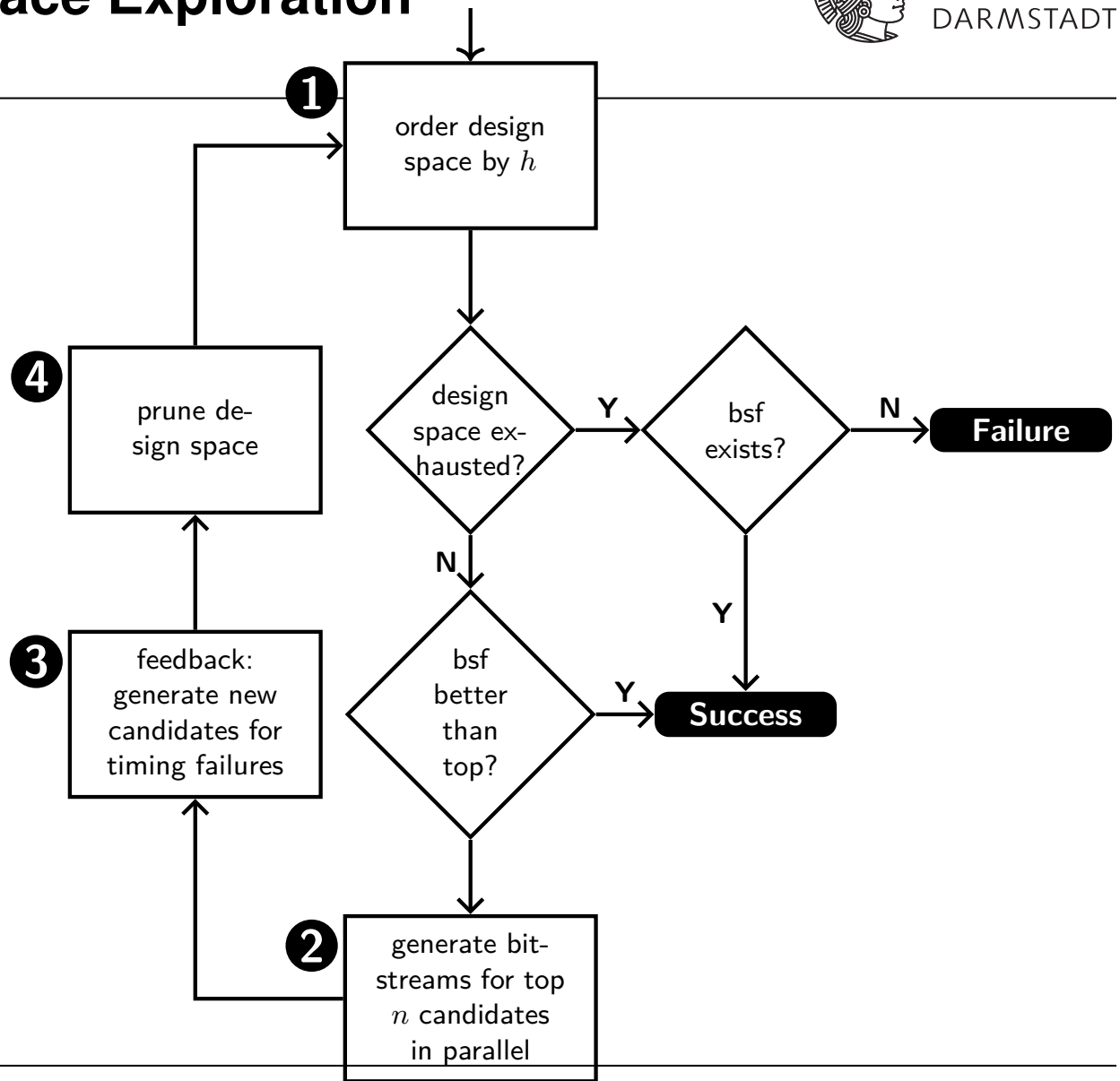


TaPaSCo
Automatic Design Space Exploration

TECHNISCHE UNIVERSITÄT DARMSTADT

Automatic DSE

- fully automatic process in TaPaSCo
- heuristic *h* (configurable)
- batch size *n* (configurable)
- bsf = best-so-far
- top = best-in-batch



TaPaSCo
More Design Space Exploration

TECHNISCHE UNIVERSITÄT DARMSTADT

Automatic DSE

- very useful, automates tedious process
- can be run on compute clusters (e.g., Lichtenberg Cluster)
- increases portablity *and* performance

More DSE: Variants

- variables so far: composition ($\sim$ area), target frequency
- TaPaSCo supports multiple **variants of PEs**
  - different implementations of the algorithm
  - different area requirements
  - different levels of parallelism (fine-grained)
- automated search across all variants of a composition
- already found several non-obvious solutions

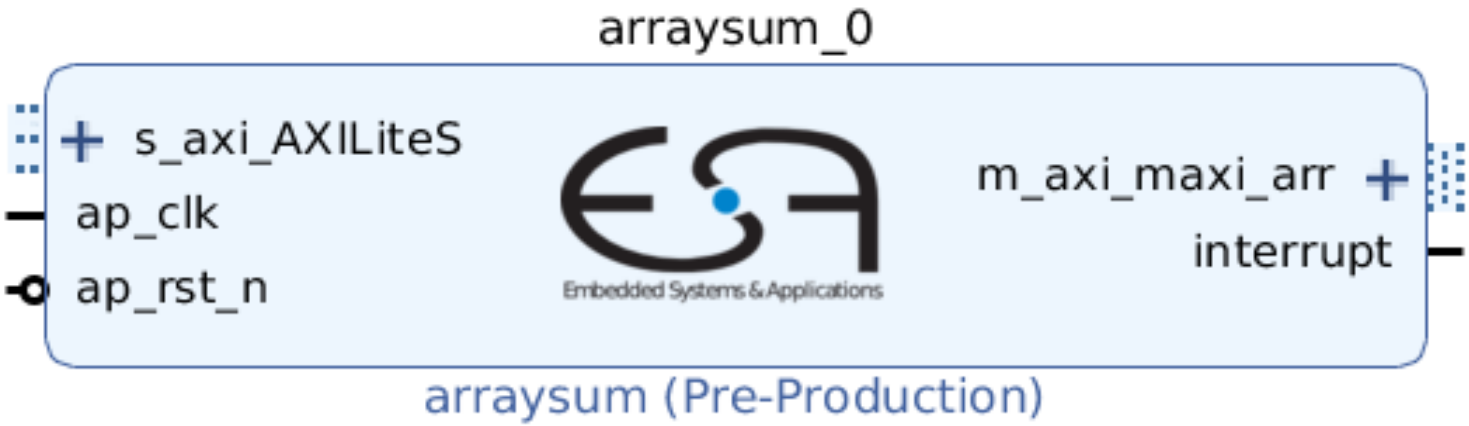
Complete Flow Example

TECHNISCHE UNIVERSITÄT DARMSTADT

```
#include "arraysum.h"

int arraysum(int arr[SZ])
{
        int i = 0, res = 0;
        for (; i < SZ; ++i)
                res += arr[i];
        return res;
}
```

Complete Flow Example

TECHNISCHE UNIVERSITÄT DARMSTADT

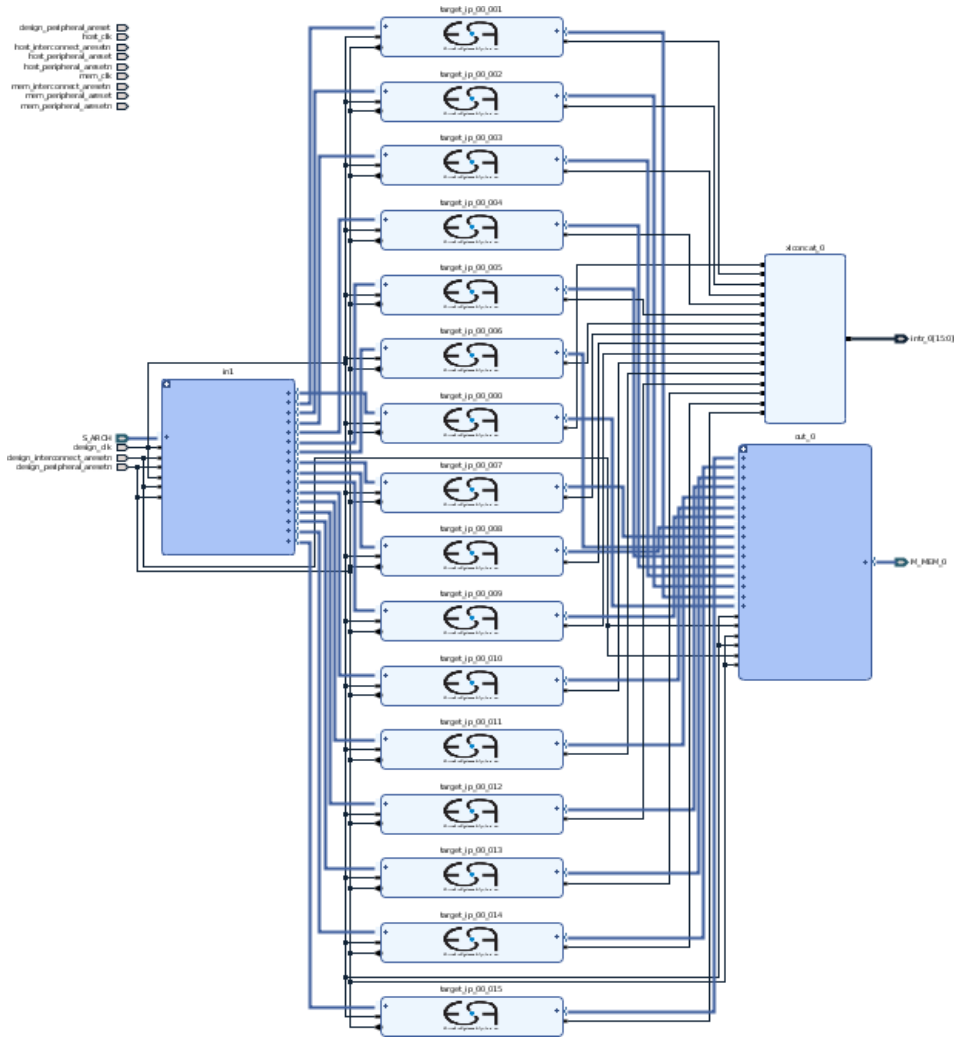


LUTs	504	
FlipFlops	780	
BRAM	0.5	
DSPs	0	
T_{min}	2.211	ns
F_{max}	$\approx$ 452.3	MHz

Table: Results from out-of-context synthesis

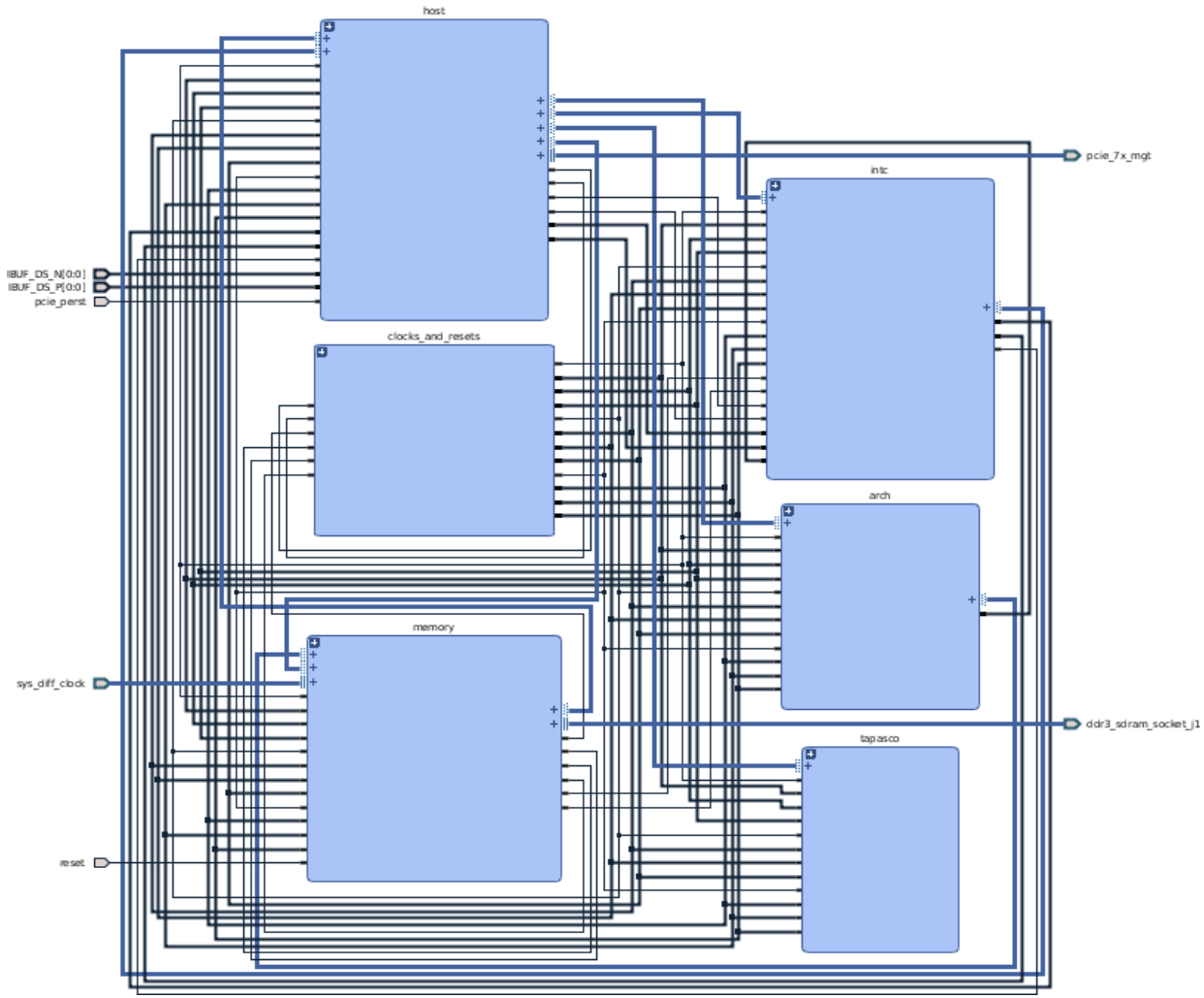
Complete Flow Example

TECHNISCHE UNIVERSITÄT DARMSTADT



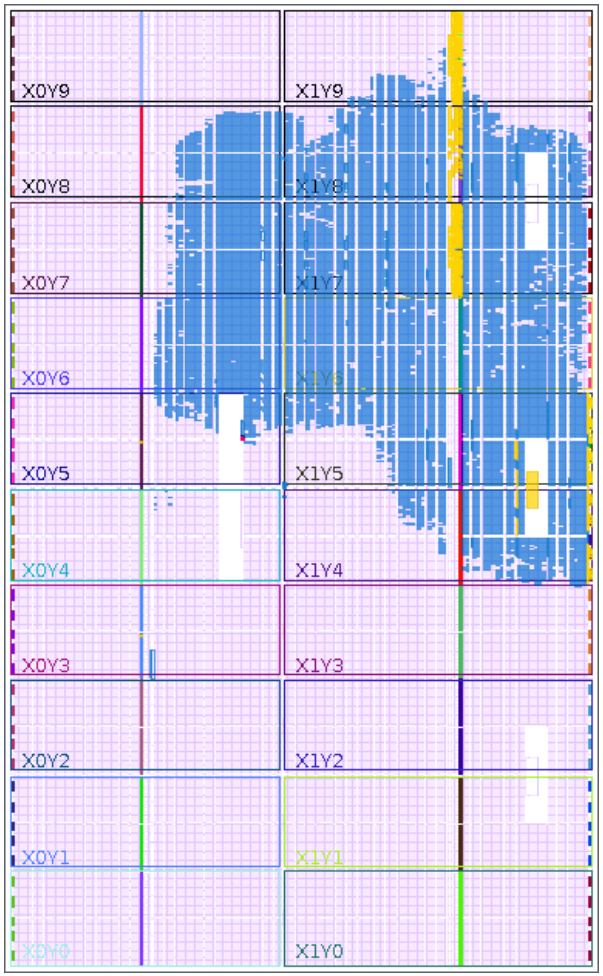
Complete Flow Example

TECHNISCHE UNIVERSITÄT DARMSTADT



Complete Flow Example

TECHNISCHE UNIVERSITÄT DARMSTADT



Place and route results

Xilinx VC709, 16 PEs

OOC:	**LUTs**	*8064*	504
	FlipFlops	*12480*	780
	BRAM	*8*	0.5
	DSPs	*0*	0
Total:	**LUTs**	8027	501.69
	FlipFlops	7755	484.69
	BRAM	48	3
	DSPs	0	0

T_{min}	4.67	ns
F_{max}	$\approx$ 214	MHz

Application Example I

TECHNISCHE UNIVERSITÄT DARMSTADT

```
#include "arraysum.h"

// [...]

typedef int job_t[SZ];
static constexpr size_t FILE_SZ = 8192 * 1024;
static constexpr size_t JOBS    = FILE_SZ / SZ;

job_t *data_in  = new job_t[JOBS];
int   *data_out = new int[JOBS];
```

Application Example II

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// [... fetching data from file ...]

for (size_t i = 0; i < JOBS; ++i) {
        data_out[i] = arraysum(data_in[i]);
}

// [... clean up, free memory, exit ... ]
```

TaPaSCo Application I

TECHNISCHE UNIVERSITÄT DARMSTADT

```
#include <tapasco.hpp>
#include "arraysum.h"
Tapasco tapasco;
// [...]
typedef int job_t[SZ];
static constexpr size_t FILE_SZ = 8192 * 1024;
static constexpr size_t JOBS    = FILE_SZ / SZ;

job_t *data_in  = new job_t[JOBS];
int   *data_out = new int[JOBS];
```

TaPaSCo Application II

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// [... fetching data from file ...]

for (size_t i = 0; i < JOBS; ++i) {
        tapasco.launch(ID,
                       data_out[i],
                       data_in[i]);
}

// [... clean up, free memory, exit ... ]
```

TaPaSCo

TECHNISCHE UNIVERSITÄT DARMSTADT

. . . is free software[3]!

- complete source on our public **GitLab**

  `https://git.esa.informatik.tu-darmstadt.de/tapasco/tapasco`

- free software under LGPLv2 license

. . . is lacking extensive documentation

- but the code is well-documented inline
- some more information can be found in our papers:

  `https://www.esa.informatik.tu-darmstadt.de/twiki/bin/view/Staff/JensKorinthDe.html`

- mostly outdated, but some in-depth info can also be found here:
  REPARA Project, Work Package 5 Deliverables

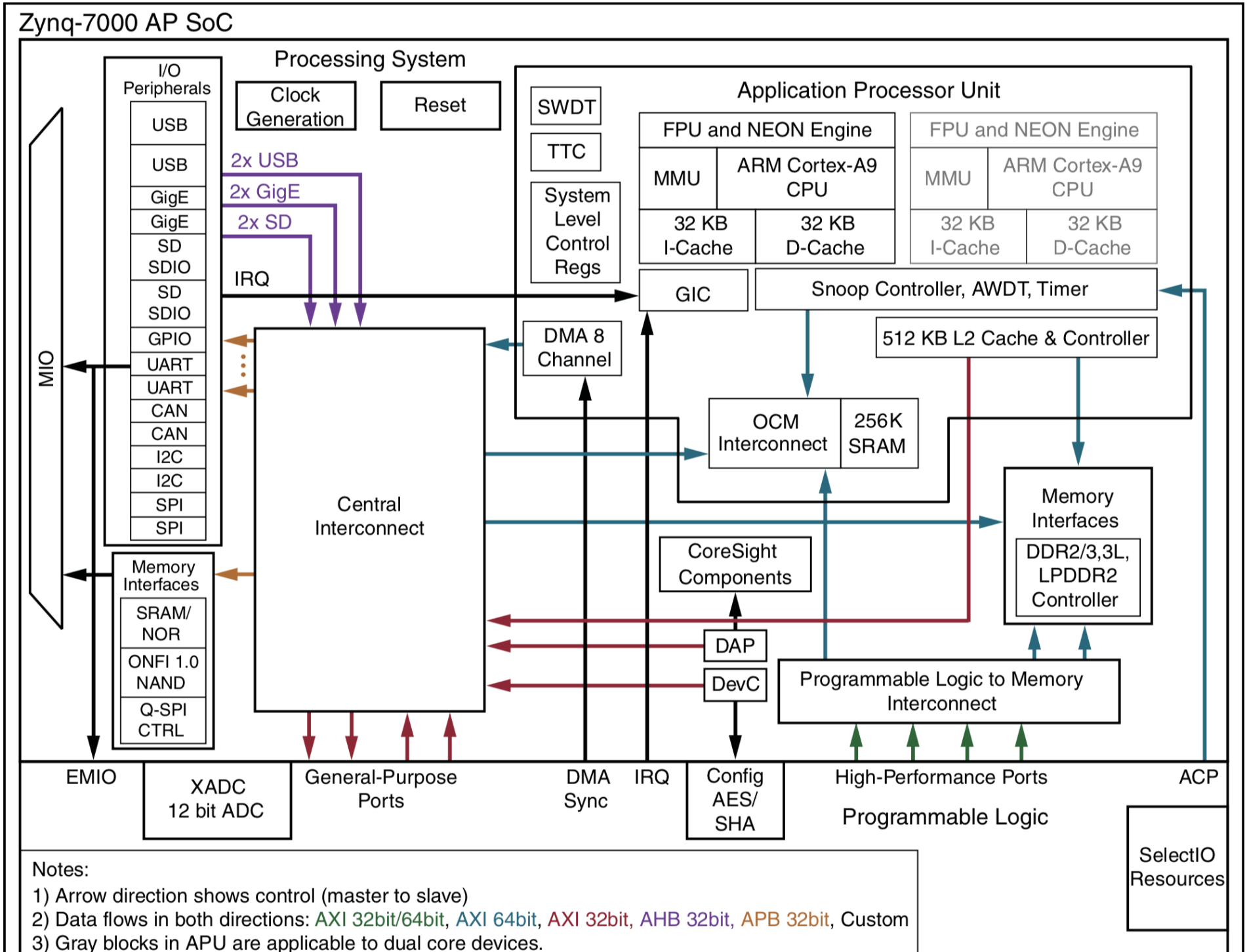
  `http://repara-project.eu/?cat=7`

[3] requires Vivado Design Suite

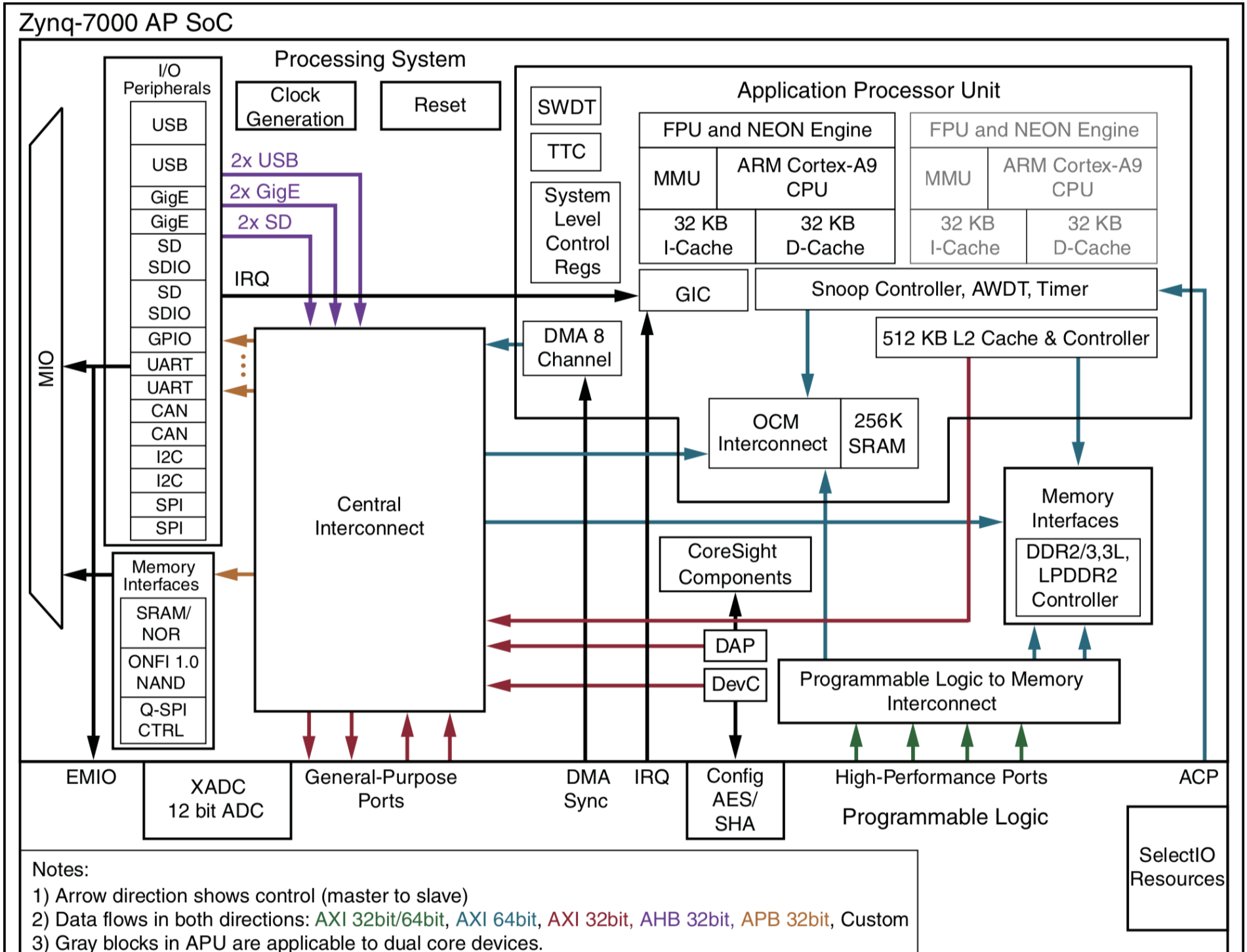
Completing the puzzle: Address Spaces and Address Maps

TECHNISCHE UNIVERSITÄT DARMSTADT

- What we've seen so far:
  - devices are controlled via **registers**
  - registers are organized in a **local address map**
  - local address map is **mapped** into a controlling master at a **base address**
- But how does the CPU access the controller?
  - it's turtles all the way down!
  - classic **Memory-as-I/O** abstraction:
    - CPU controls memory masters, which access memory addresses
    - most ranges map to **memory**
    - some ranges map to **other devices**
    - e.g., general purpose AXI controllers (GPx)
  - an interconnect controls the global address map

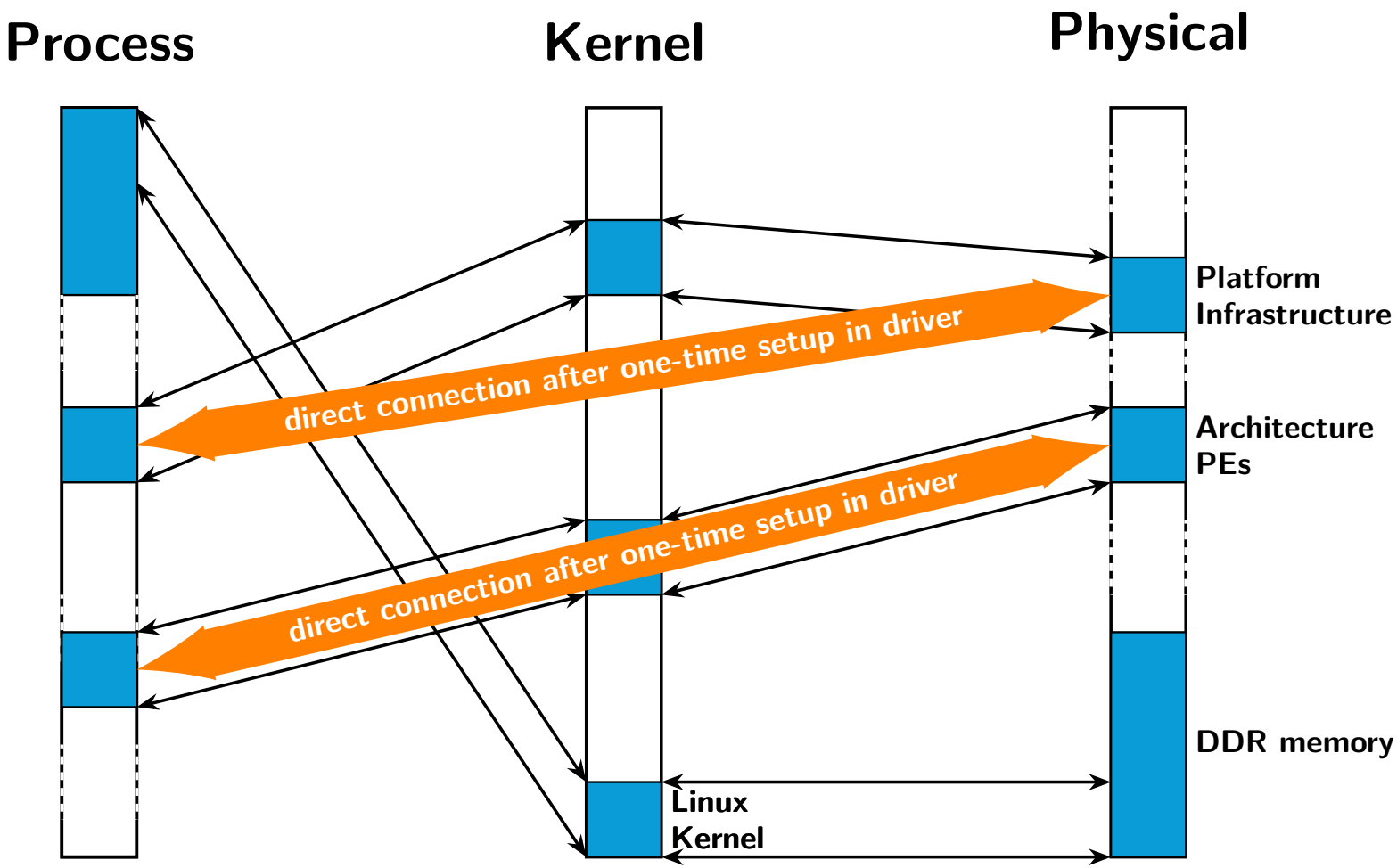

Zynq-7000 AP SoC
Processing System
I/O Peripherals
USB
USB
GigE
GigE
SD SDIO
SD SDIO
GPIO
UART
UART
CAN
CAN
I2C
I2C
SPI
SPI
MIO
Clock Generation
Reset
2x USB
2x GigE
2x SD
IRQ
SWDT
TTC
System Level Control Regs
Application Processor Unit
FPU and NEON Engine
MMU
ARM Cortex-A9 CPU
32 KB I-Cache
32 KB D-Cache
FPU and NEON Engine
MMU
ARM Cortex-A9 CPU
32 KB I-Cache
32 KB D-Cache
GIC
Snoop Controller, AWDT, Timer
512 KB L2 Cache & Controller
DMA 8 Channel
OCM Interconnect
256K SRAM
Central Interconnect
Memory Interfaces
DDR2/3,3L, LPDDR2 Controller
CoreSight Components
DAP
DevC
Programmable Logic to Memory Interconnect
Memory Interfaces
SRAM/ NOR
ONFI 1.0 NAND
Q-SPI CTRL
EMIO
XADC 12 bit ADC
General-Purpose Ports
DMA Sync
IRQ
Config AES/ SHA
High-Performance Ports
ACP
Programmable Logic
SelectIO Resources
Notes:
1) Arrow direction shows control (master to slave)
2) Data flows in both directions: AXI 32bit/64bit, AXI 64bit, AXI 32bit, AHB 32bit, APB 32bit, Custom
3) Gray blocks in APU are applicable to dual core devices.

Address Range	CPUs and ACP	AXI_HP	Other Bus Masters[1]	Notes
0000_0000 to 0003_FFFF[2]	OCM	OCM	OCM	Address not filtered by SCU and OCM is mapped low
	DDR	OCM	OCM	Address filtered by SCU and OCM is mapped low
	DDR			Address filtered by SCU and OCM is not mapped low
				Address not filtered by SCU and OCM is not mapped low
0004_0000 to 0007_FFFF	DDR			Address filtered by SCU
				Address not filtered by SCU
0008_0000 to 000F_FFFF	DDR	DDR	DDR	Address filtered by SCU
		DDR	DDR	Address not filtered by SCU[3]
0010_0000 to 3FFF_FFFF	DDR	DDR	DDR	Accessible to all interconnect masters
4000_0000 to 7FFF_FFFF	PL		PL	General Purpose Port #0 to the PL, M_AXI_GP0
8000_0000 to BFFF_FFFF	PL		PL	General Purpose Port #1 to the PL, M_AXI_GP1
E000_0000 to E02F_FFFF	IOP		IOP	I/O Peripheral registers, see Table 4-6
E100_0000 to E5FF_FFFF	SMC		SMC	SMC Memories, see Table 4-5
F800_0000 to F800_0BFF	SLCR		SLCR	SLCR registers, see Table 4-3
F800_1000 to F880_FFFF	PS		PS	PS System registers, see Table 4-7
F890_0000 to F8F0_2FFF	CPU			CPU Private registers, see Table 4-4


Zynq-7000 AP SoC
Processing System
I/O Peripherals
USB
USB
GigE
GigE
SD SDIO
SD SDIO
GPIO
UART
UART
CAN
CAN
I2C
I2C
SPI
SPI
MIO
Clock Generation
Reset
2x USB
2x GigE
2x SD
IRQ
SWDT
TTC
System Level Control Regs
Application Processor Unit
FPU and NEON Engine
MMU
ARM Cortex-A9 CPU
32 KB I-Cache
32 KB D-Cache
FPU and NEON Engine
MMU
ARM Cortex-A9 CPU
32 KB I-Cache
32 KB D-Cache
GIC
Snoop Controller, AWDT, Timer
512 KB L2 Cache & Controller
DMA 8 Channel
OCM Interconnect
256K SRAM
Central Interconnect
Memory Interfaces
DDR2/3,3L, LPDDR2 Controller
CoreSight Components
DAP
DevC
Programmable Logic to Memory Interconnect
Memory Interfaces
SRAM/ NOR
ONFI 1.0 NAND
Q-SPI CTRL
EMIO
XADC 12 bit ADC
General-Purpose Ports
DMA Sync
IRQ
Config AES/ SHA
High-Performance Ports
ACP
Programmable Logic
SelectIO Resources
Notes:
1) Arrow direction shows control (master to slave)
2) Data flows in both directions: AXI 32bit/64bit, AXI 64bit, AXI 32bit, AHB 32bit, APB 32bit, Custom
3) Gray blocks in APU are applicable to dual core devices.

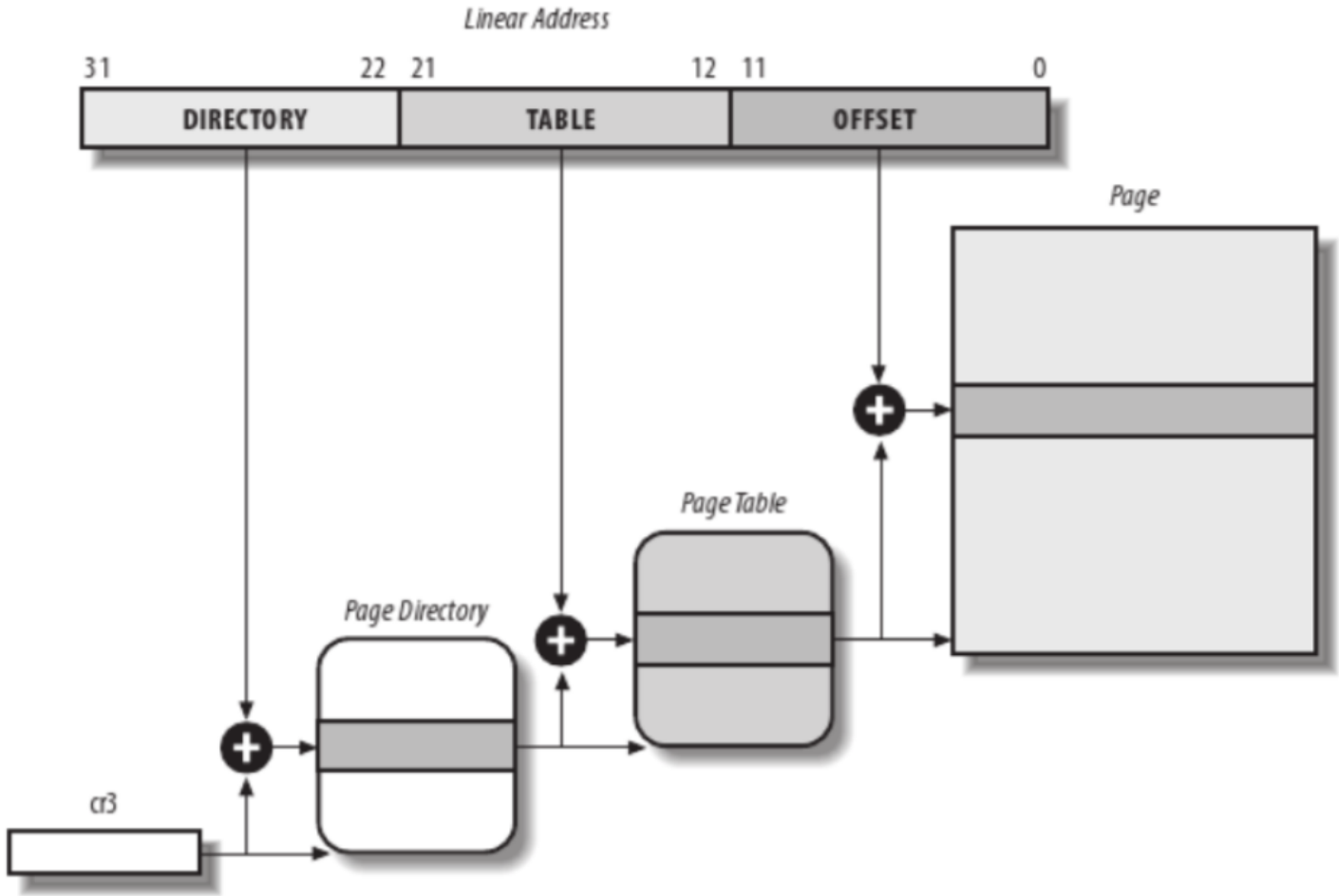
Address Spaces
Page Table Mapping

TECHNISCHE UNIVERSITÄT DARMSTADT



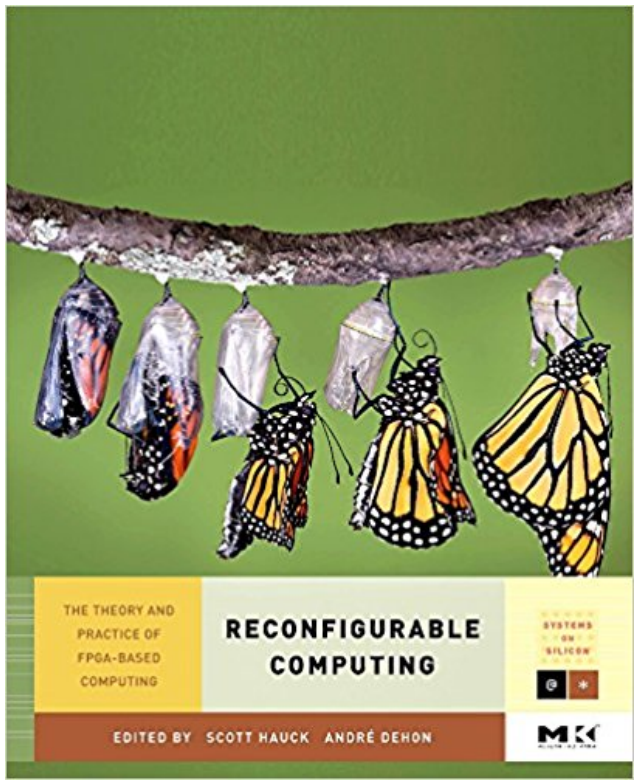
Address Spaces
Page Table Mapping

TECHNISCHE UNIVERSITÄT DARMSTADT



FPGA Structures
Reconfigurable Computing

TECHNISCHE UNIVERSITÄT DARMSTADT



- all images in this chapter are taken from this book:
- Scott Hauck, André DeHon: Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation (Systems on Silicon), ISBN: 978-0123705228
- heavily outdated, but some nice basics

FPGA Structures
Look-Up Tables (LUT)

TECHNISCHE UNIVERSITÄT DARMSTADT

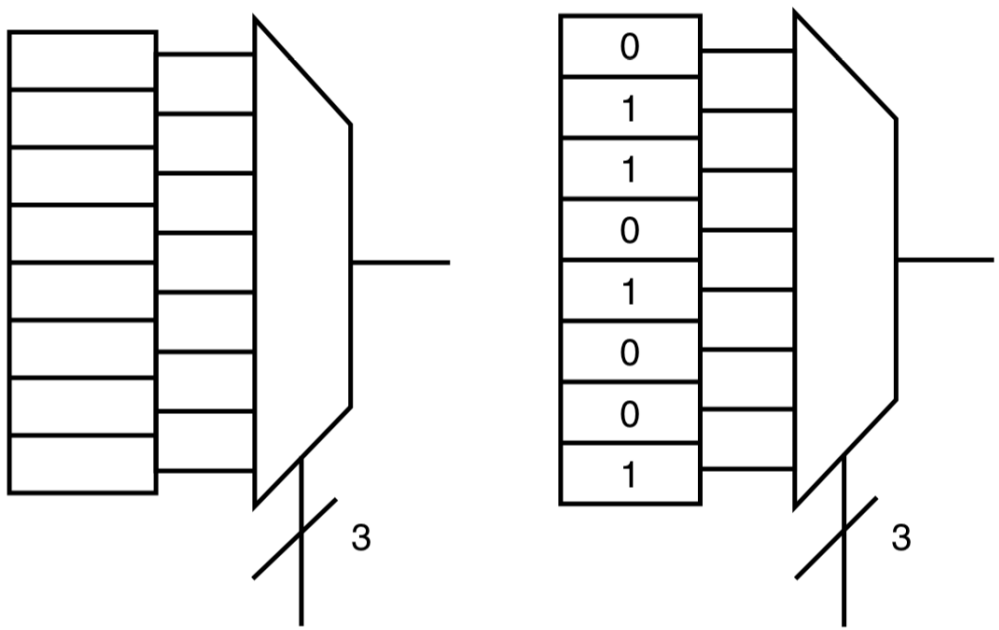


FIGURE 1.1 ■ A 3-LUT schematic (a) and the corresponding 3-LUT symbol and truth table (b) for a logical XOR.

FPGA Structures
Look-Up Tables (LUT)

TECHNISCHE UNIVERSITÄT DARMSTADT

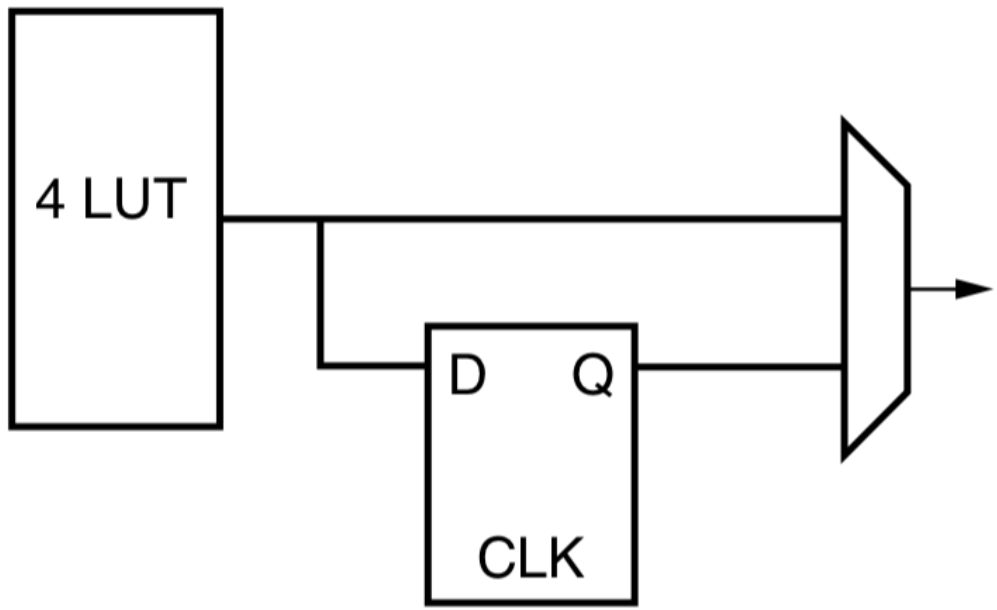
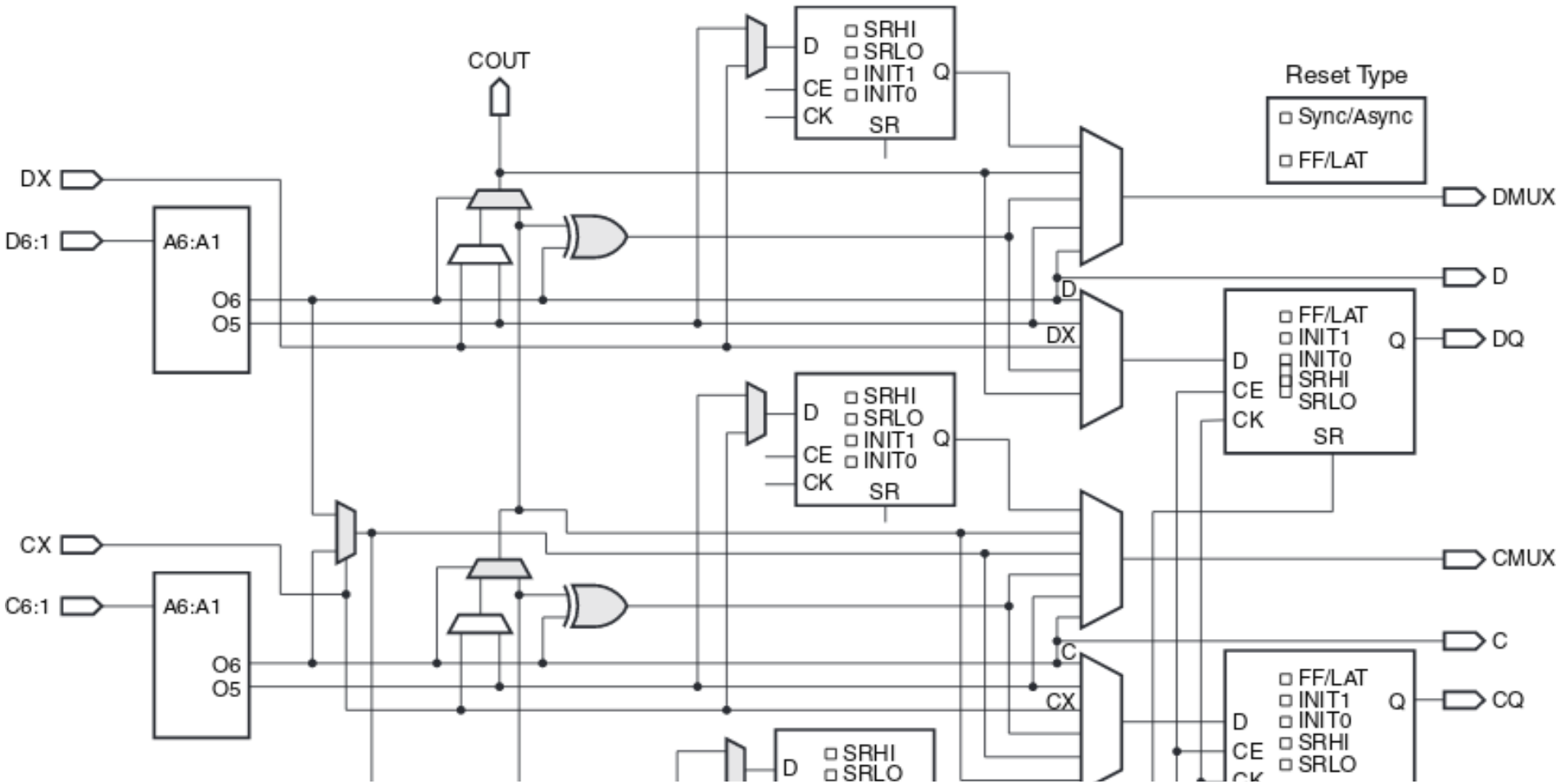


FIGURE 1.2 ■ A simple lookup table logic block.

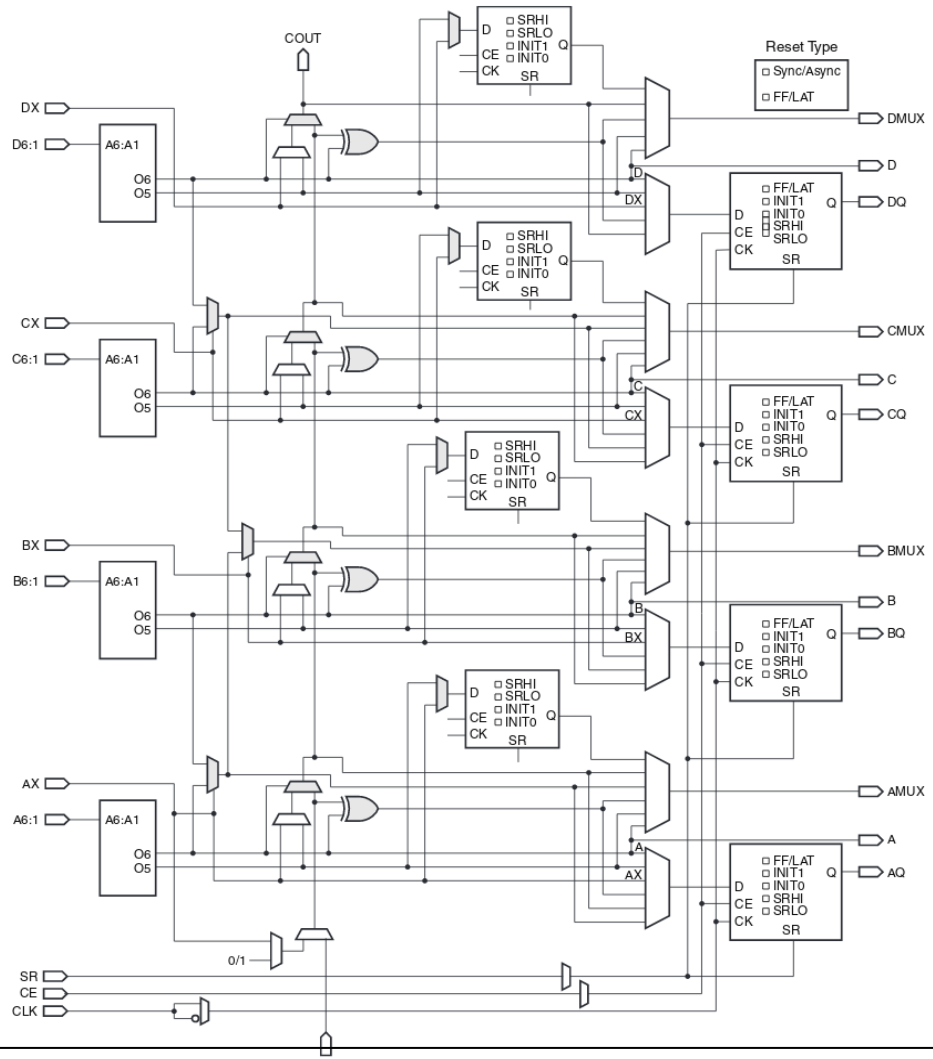
FPGA Structures
Xilinx 7-Series CLB (UG474)

TECHNISCHE UNIVERSITÄT DARMSTADT



FPGA Structures
Xilinx 7-Series CLB (UG474)

TECHNISCHE UNIVERSITÄT DARMSTADT



FPGA Structures
Interconnection

TECHNISCHE UNIVERSITÄT DARMSTADT

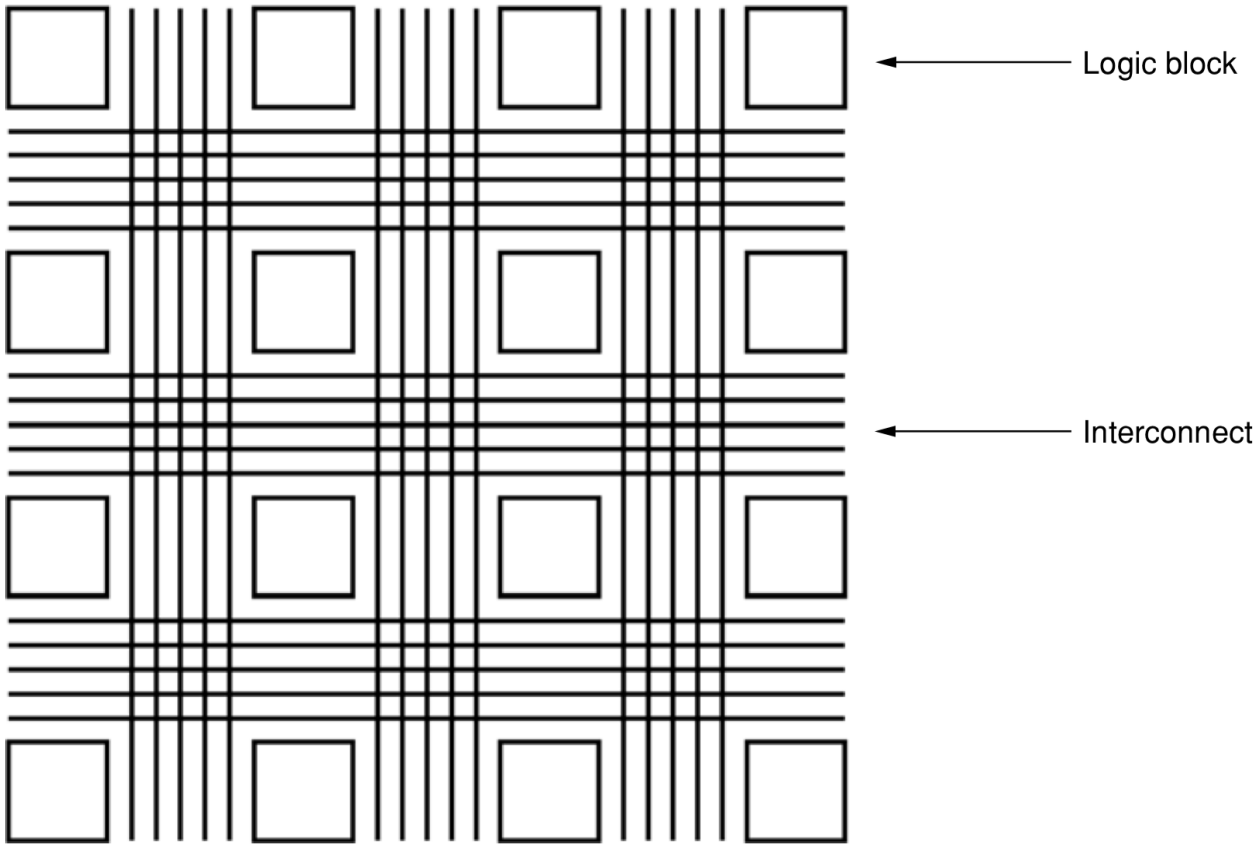


FIGURE 1.3 ■ The island-style FPGA architecture. The interconnect shown here is not representative of structures actually used.

FPGA Structures
Island Style Architecture

TECHNISCHE UNIVERSITÄT DARMSTADT

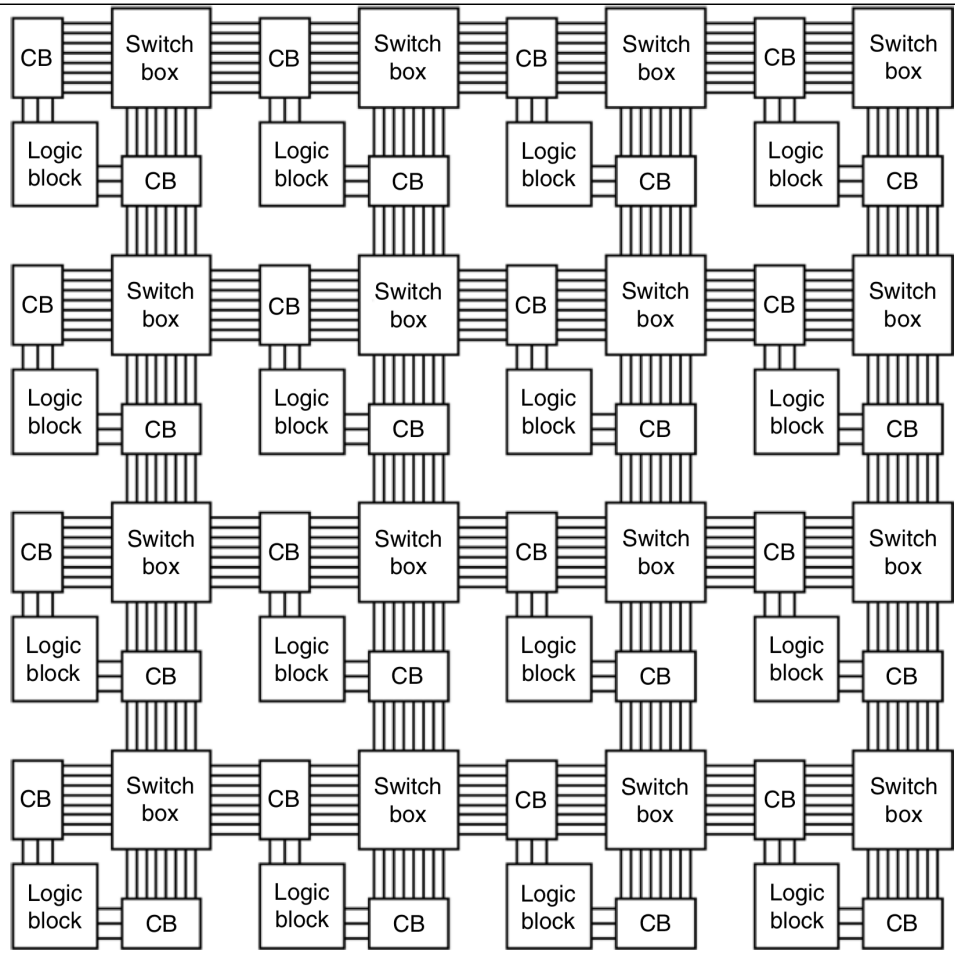


FIGURE 1.5 ■ An island-style architecture with connect blocks and switch boxes to support more complex routing structures. (The difference in relative sizes of the blocks is for visual differentiation.)

FPGA Structures
Connection Blocks

TECHNISCHE UNIVERSITÄT DARMSTADT

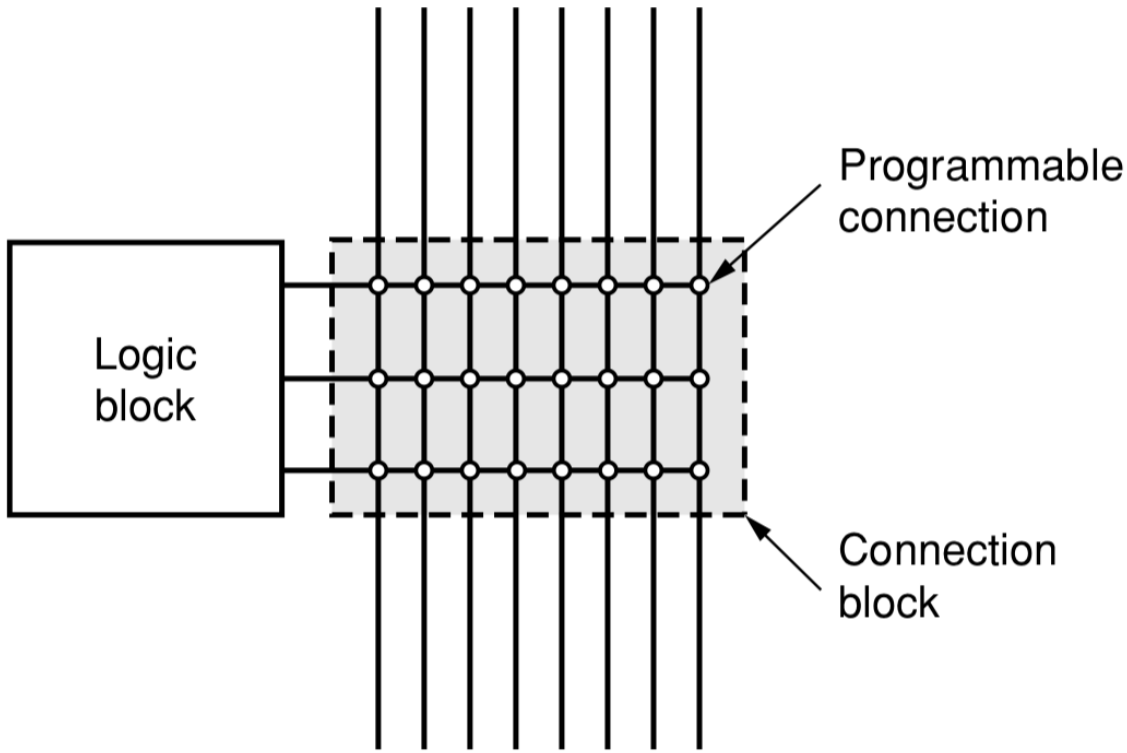


FIGURE 1.6 ■ Detail of a connection block.

FPGA Structures
Switch Blocks

TECHNISCHE UNIVERSITÄT DARMSTADT

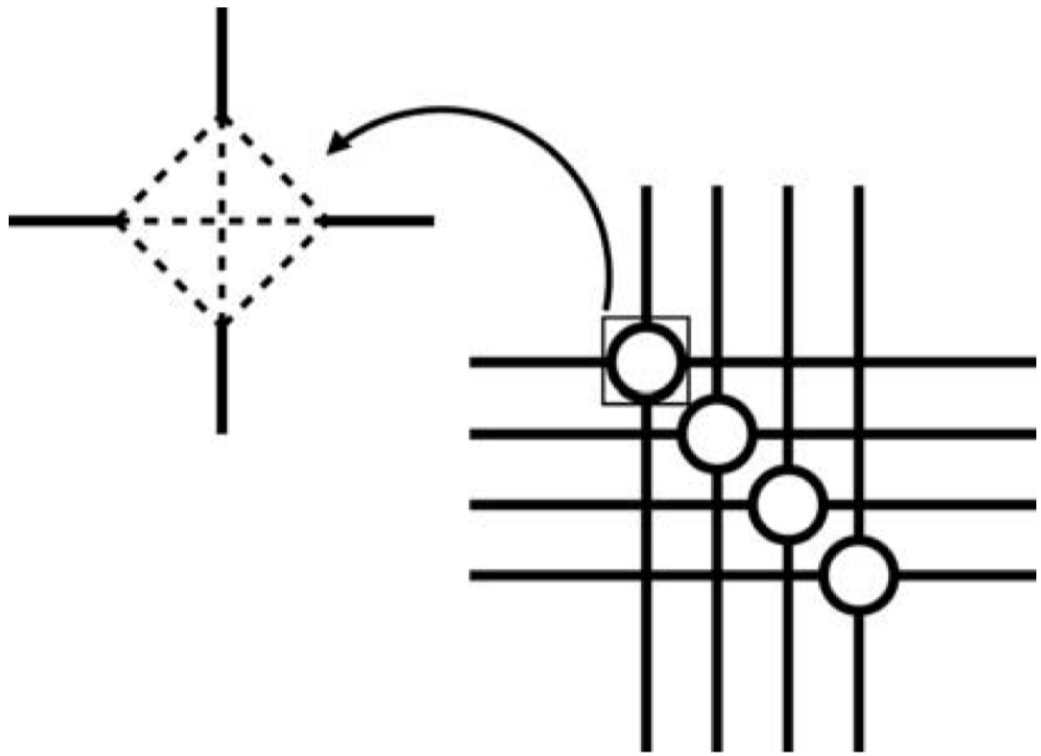


FIGURE 1.7 ■ An example of a common switch block architecture.

FPGA Structures
Island Style Architecture

TECHNISCHE UNIVERSITÄT DARMSTADT

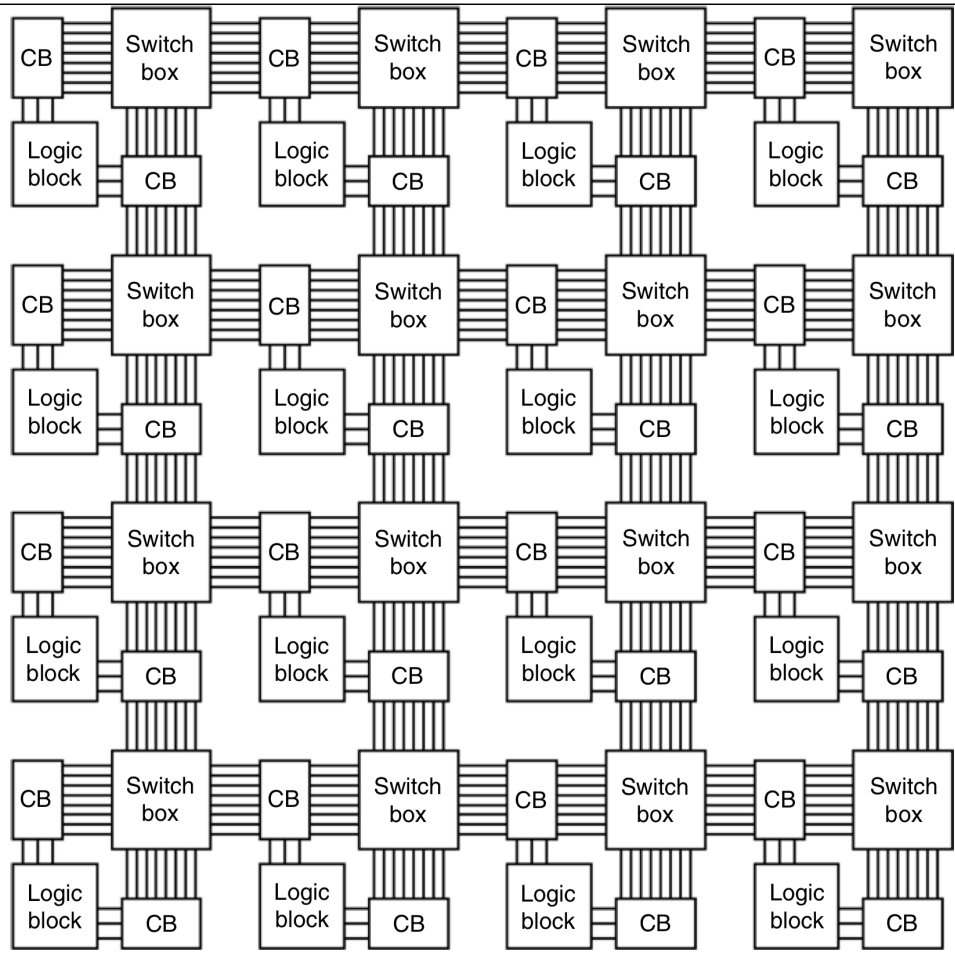


FIGURE 1.5 ■ An island-style architecture with connect blocks and switch boxes to support more complex routing structures. (The difference in relative sizes of the blocks is for visual differentiation.)

FPGA Structures
Interconnect Hierarchical Connections

TECHNISCHE UNIVERSITÄT DARMSTADT

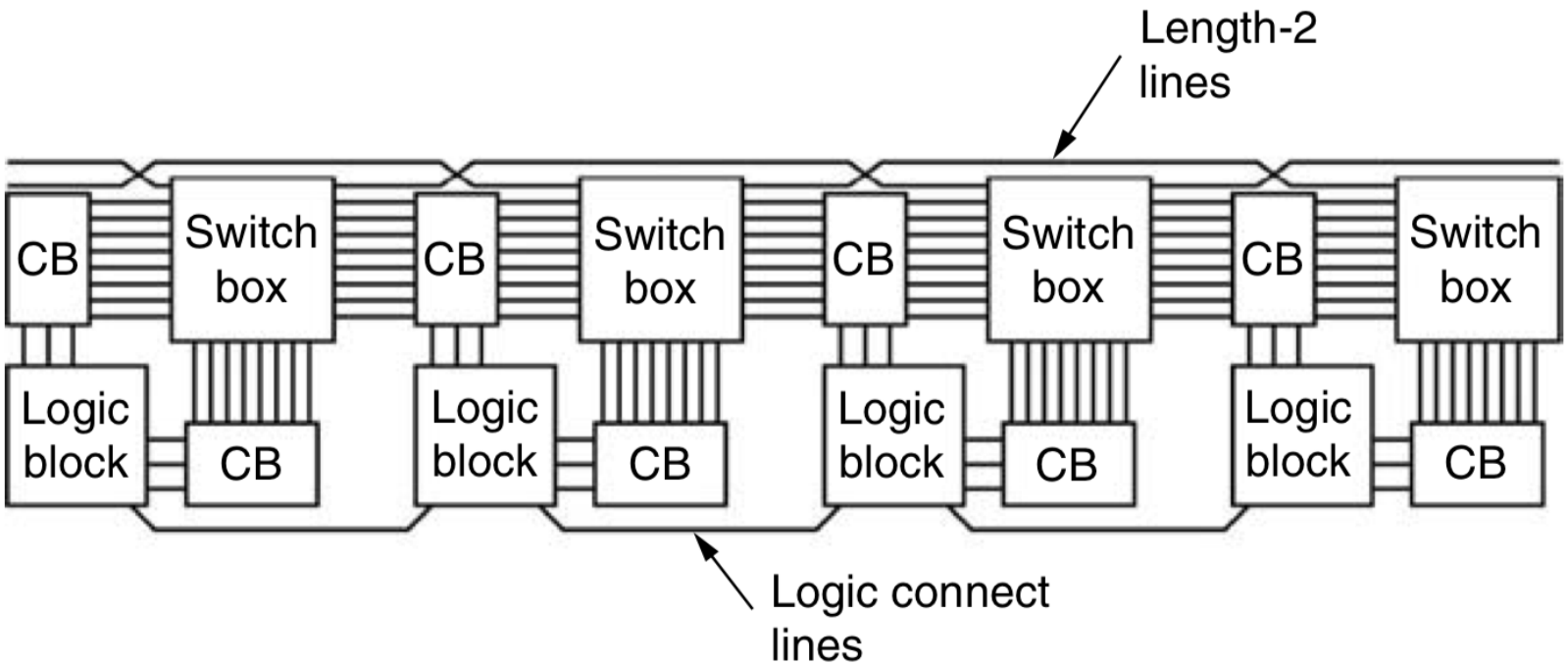


FIGURE 1.8 ■ Local (direct) connections and L2 connections augmenting a switched interconnect.

FPGA Structures
Programmable Routing (electrical level)

TECHNISCHE UNIVERSITÄT DARMSTADT