

Partial Reconfiguration on FPGAs



Dirk Koch (koch@ifi.uio.no)

Introduction: Terms and Definitions

Definition of the term „Reconfigurable Computing“ (RC)

- A good definition for a reconfigurable hardware system was introduced with the Rammig Machine (by Franz Rammig 1977):
... a system, which, with no manual or mechanical interference, permits the building, changing, processing and destruction of real (not simulated) digital hardware
- Reconfigurable computing (RC) is defined as
the study of computation using reconfigurable devices
This includes architectures, algorithms and applications
- The term RC is often used to express that computation is carried out using dedicated hardware structures (often utilizing a high level of parallelism) which are mapped on reconfigurable hardware (this is opposed to the sequential von Neumann computer paradigm!!!).

Introduction: Example

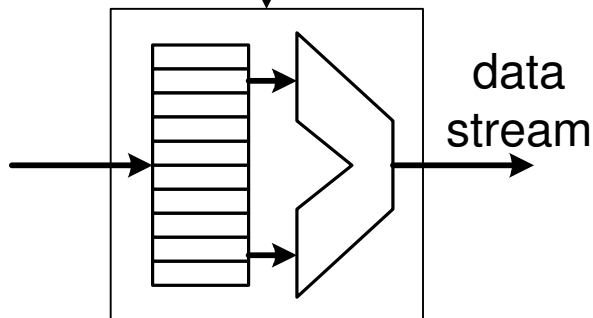
```
for i = 0 to 7 do {  
    tmp = A[i] & x"F";  
    tmp = tmp + 42;  
    Q[i] = tmp * 24;  
}
```

~~slow and power hungry~~

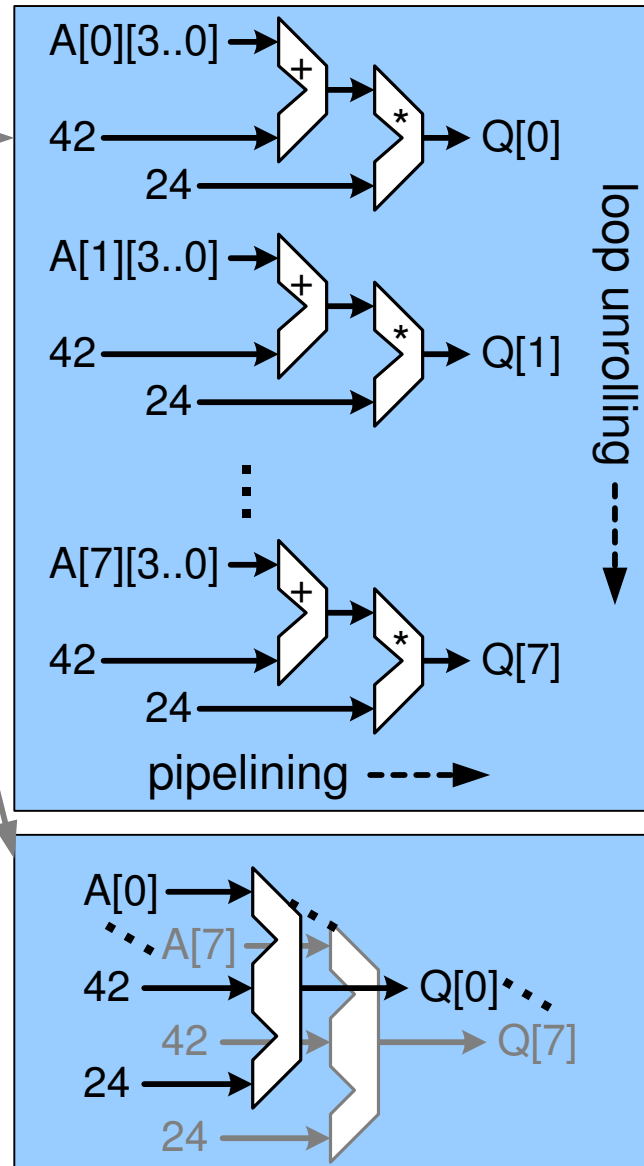
von Neumann computer

```
LDI reg_i, 0  
L1: ANDI r_tmp, $i, xF  
...  
BLI reg_i, L1
```

instruction stream



reconfigurable computing



RC benefits among von Neumann machines:

- fast parallel processing
 - pipelining
 - loop transform.
- no instr. fetch (no extra memory access)
- no instr. decode
- possibility of dedicated instr. (e.g., MAC)
- lower power

Introduction: Example (Benefits)

Reconfigurable computing permits to tradeoff between performance (speed and/or latency) and area (number of used primitives) of the reconfigurable architecture. This requires to solve the following steps:

- **Allocation**: defining the resources / functional blocks which are allowed for implementation
- **Binding**: defining which operation is executed on a particular allocated resource
- **Scheduling**: defining the time when an operation is executed

Allocation, binding, and scheduling are fundamental problems that have to be solved at different level of abstraction (e.g., system level, architecture level, or all refinements).

This holds for both the hardware and the software part!

Further: RC removes architectural limitations (e.g., like shared memory communication in GPUs)

Introduction: Terms and Definitions

These RC benefits exist also for dedicated hardware (ASIC¹, ASIP²), but reconfigurable computing allows more:

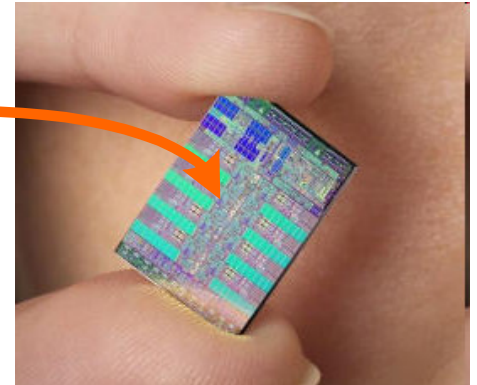
- **Adaptability**: react on environment changes or different workload scenarios by **adapting the behavior and structure of a system** (e.g., scaling a system with configuring more instances of an accelerator module to a recent device)
- **Customization** (post fabrication): allows for different features for individual systems
- **Updatability**: update to new standards, bug fixes, after sales business with new features „hardware apps“

Possible by (re)configuration: **Configuration (and respectively reconfiguration) is the process of changing the structure of a reconfigurable device at start-up-time (respectively run-time).** Mostly this means: sending new configurations to the device

Introduction: Terms and Definitions

Reconfigurable architectures

- **Coarse grained:**
ALU-like primitives with word sized routing channels
 - Examples: NEC-DRC, PACT XPP, Silicon Hive, Ambric, Picochip, TILERA, Nvidia GPGPU
 - Advantage: extreme performance for domain specific tasks
- **Fine grained:** bit level primitives (e.g., look-up tables (LUTs)) and single wire routing
 - Examples: plenty of academic architectures, Atmel FPGAs
 - Advantage: can virtually implement anything
 - **But** often poor performance and/or chip utilization
- **Hybrid:** fine-grained fabric with additional coarse-grained primitives (e.g., hardware multipliers or CPUs)
 - Examples: Xilinx Virtex families (some with embedded PPC)
 - Aims at combining the advantages of both



Introduction: the FPGA-ASIC Gap

Hybrid FPGAs are dominating reconfigurable market, but there is a

- Gap between reconfigurable FPGAs and dedicated ASICs

@ 90nm process*	FPGA versus ASIC
chip area	~ 18 x larger
dynamic power	~ 14 x more
clock speed	~ 3-5 x slower
*Kuon & Rose: Measuring the Gap Between FPGAs and ASICs, in Tr. On CAD, 2007.	

- Note that the gap towards a programmable von Neumann machine could be even orders of magnitude higher!
- also: lack of productive design tools (and skilled engineers)

Solution: **partial run-time reconfiguration (PR)**:

reusing the resources of a reconfigurable architecture by multiple modules over time. Only parts of a system might be updated while continuing operation of the remaining system.

FPGA-based Systems everywhere, but not PR

- FPGA-based systems are omnipresent in our daily life.



Each A380 contains more than 700 Actel FPGAs, e.g., for:

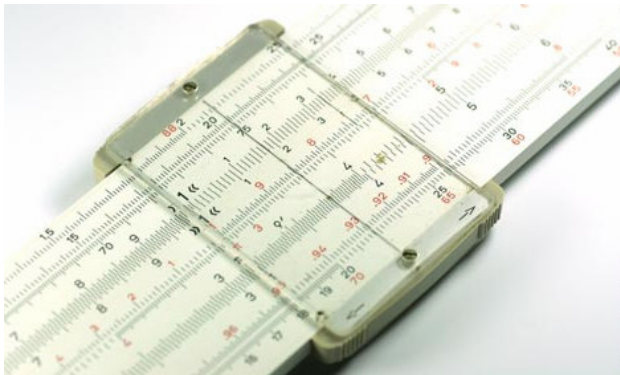
- Engine control & monitoring
- flight computers
- braking systems
- safety warning systems

What we should know about FPGAs

- Slow (~300 MHz), but highly parallel execution >1000 Operations
- Moderate I/O throughput, but >1MB @ >1TB/sec (on-chip)
- Difficult VHDL programming, but C++ is coming up

data flow oriented

```
for i=1 to  $\infty$  loop  
  numbercrunching;
```



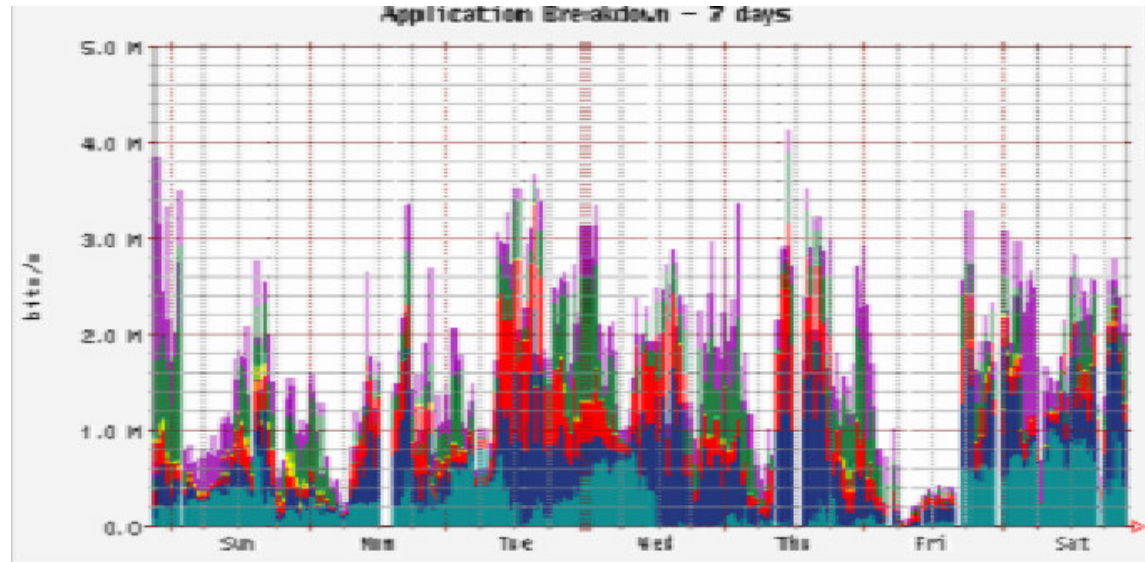
vs. control flow dominated

```
if (old_position) then  
  case position is  
    42: if free then
```

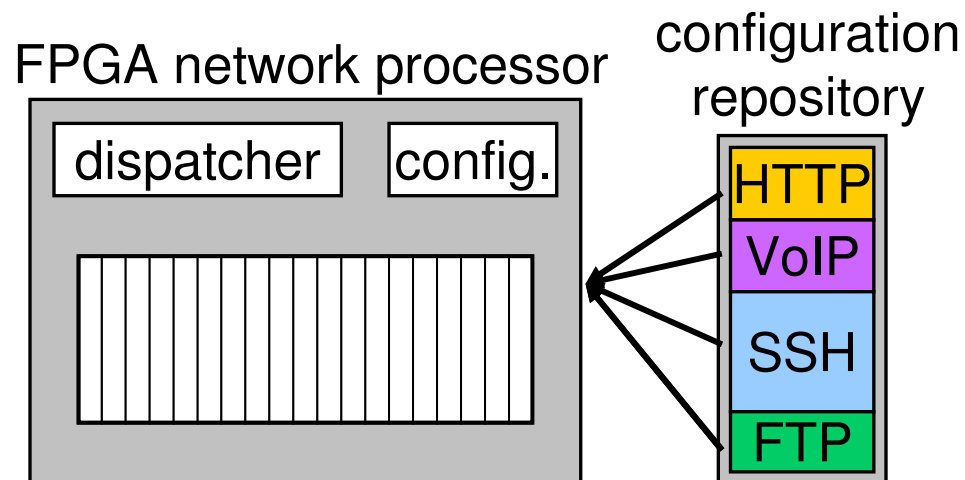


PR Advantages: Area Saving

- Networking:
Adapt to changing protocols over time
- Encapsulated design of the processing modules

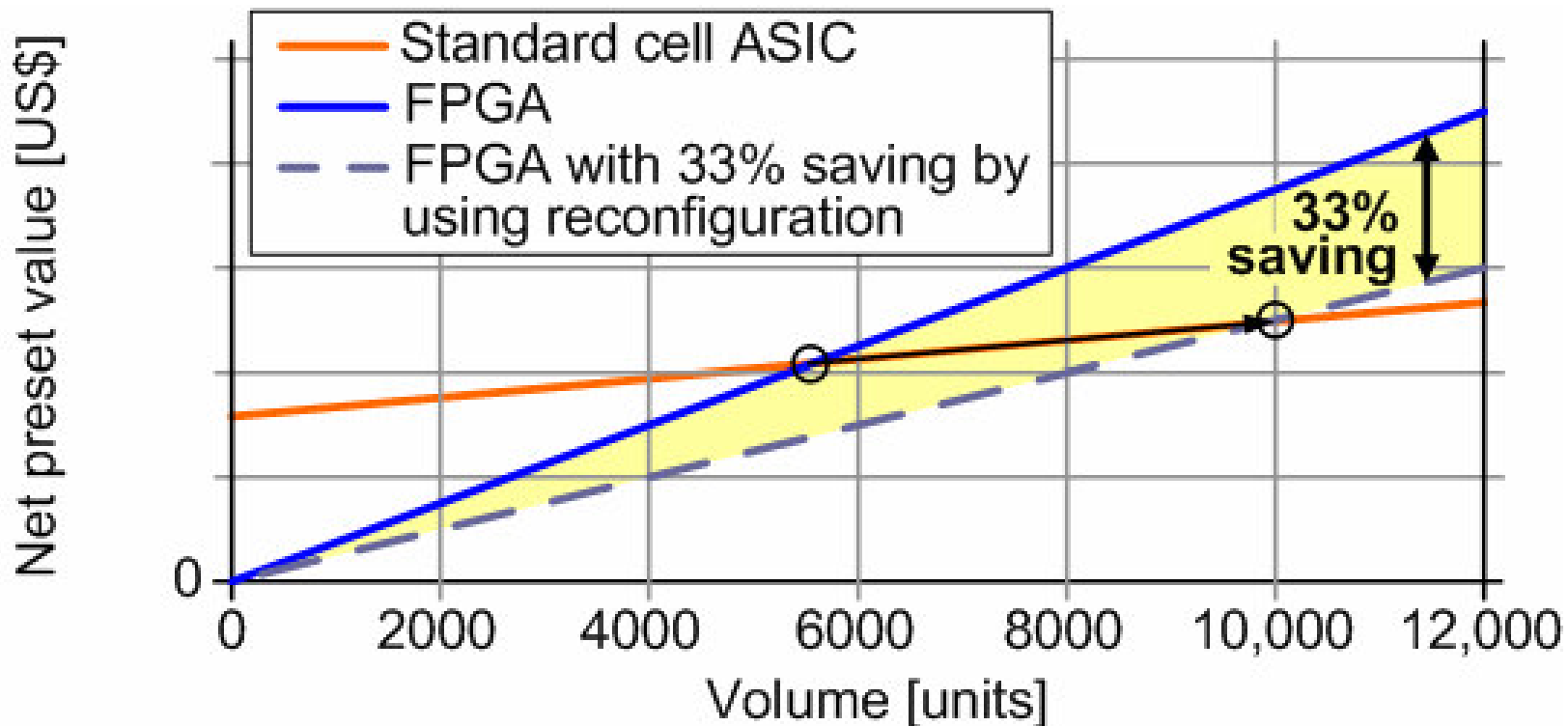


source: www.caida.org



PR Advantages: Area Saving

- Economics of ASIC- and FPGA designs

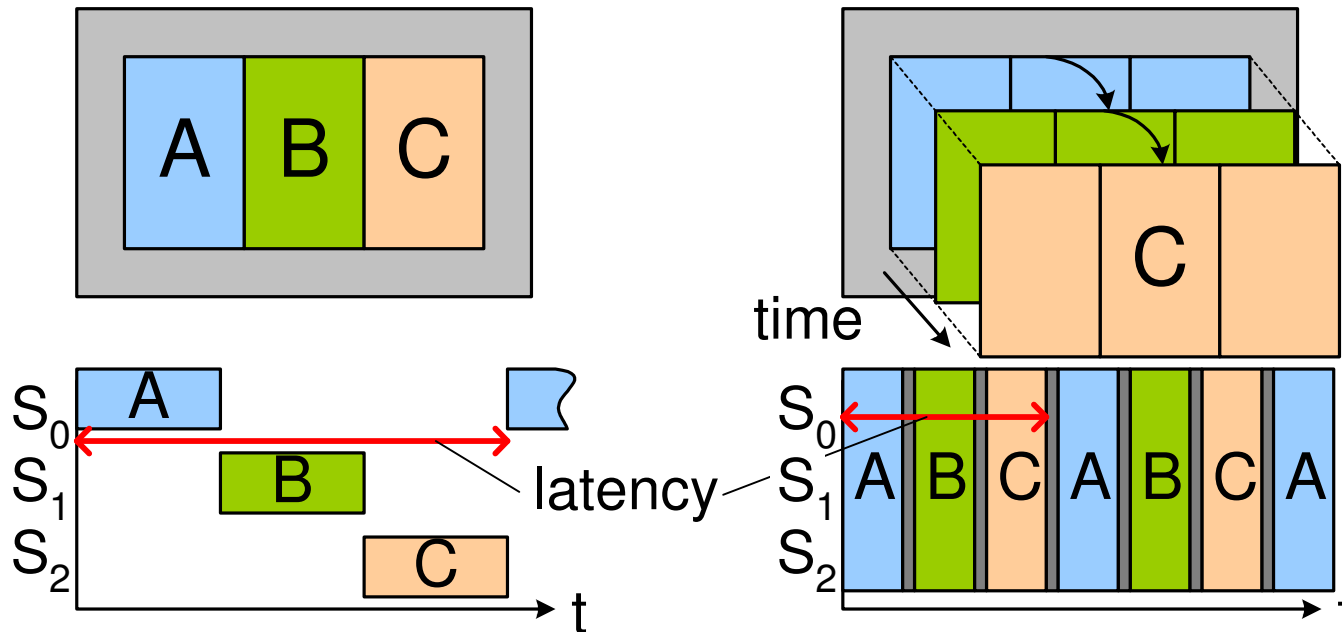


Source: Electronic News 16.03.2006

- FPGA buyers: - reduce unit cost
- after sail business
- FPGA vendors: more attractive for high volume designs

PR Advantages: Acceleration

- Reduce latency by spending more area on submodules

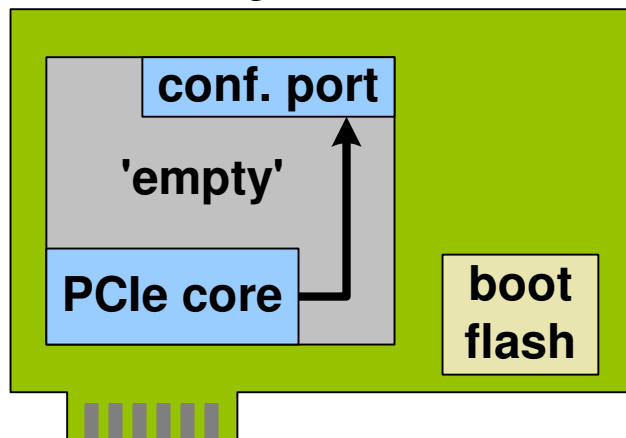


- May alternatively allow to reduce clock frequency (and power)
- Lower latency might reduce buffer sizes
- Example: TLS/SSL, sorting (database acceleration)
- May also increase throughput

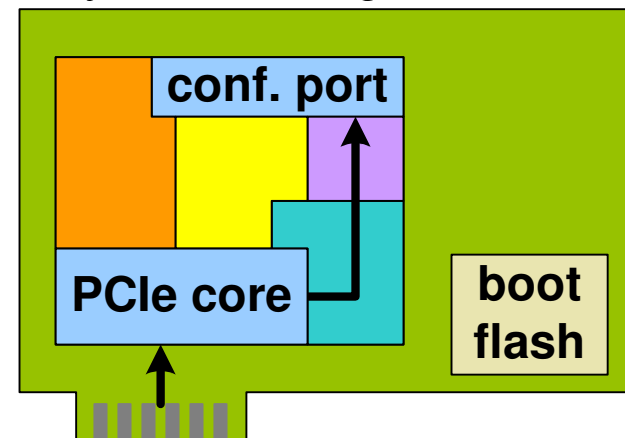
PR Advantages: Faster Configuration

- Full FPGA bistream can currently be > 20 MB
- Flash memory performance 10-20 MB/s
(special high-speed Flash memories reach up to 100 MB/s)
- → Full initial configuration ~ 1 -2 seconds in practice
an order of magnitude too slow for PCIe (setup within 100 ms)
- Solution: Bootstrapping using PR

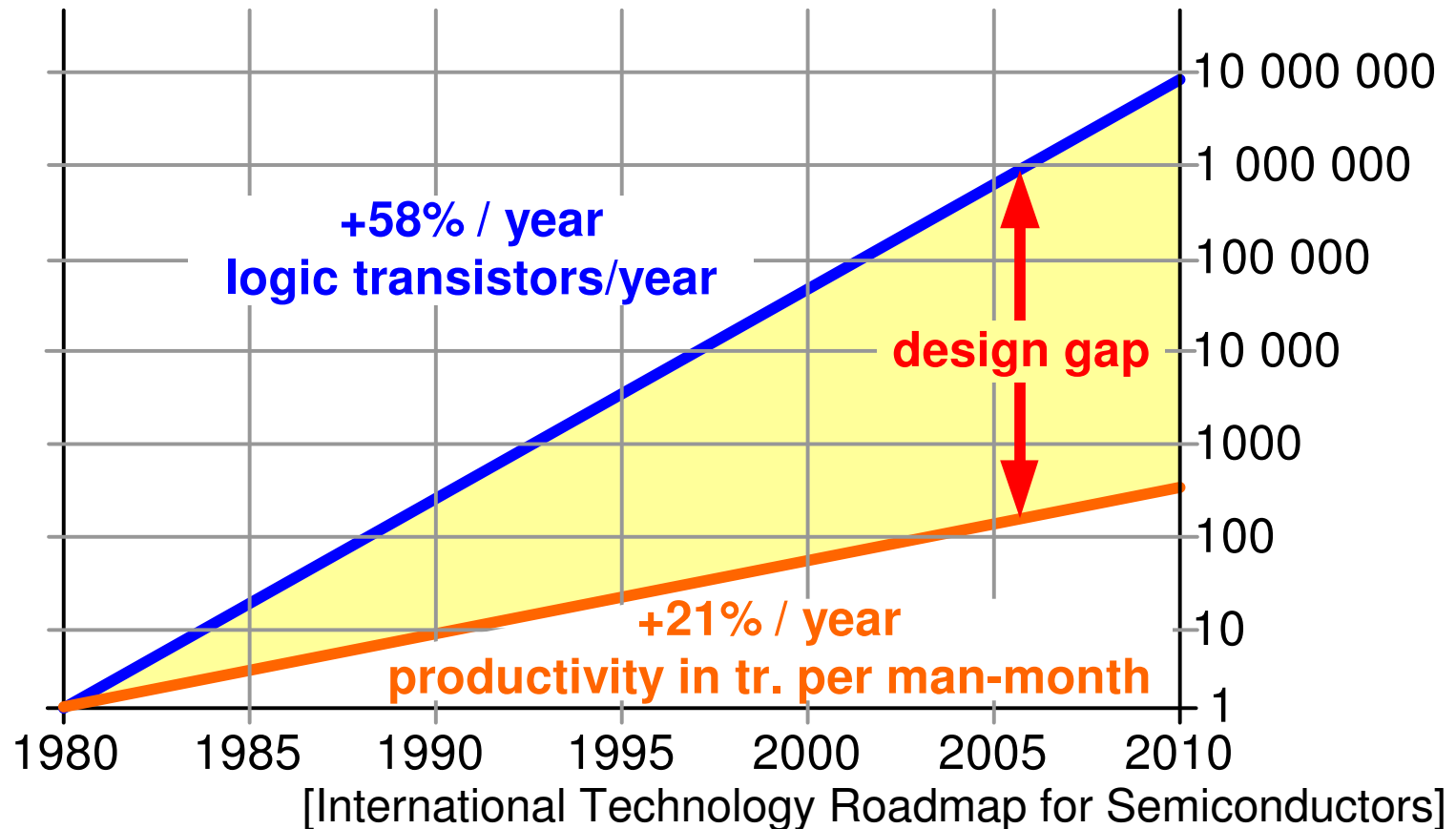
Initial config. from boot flash



System config. via PCIe



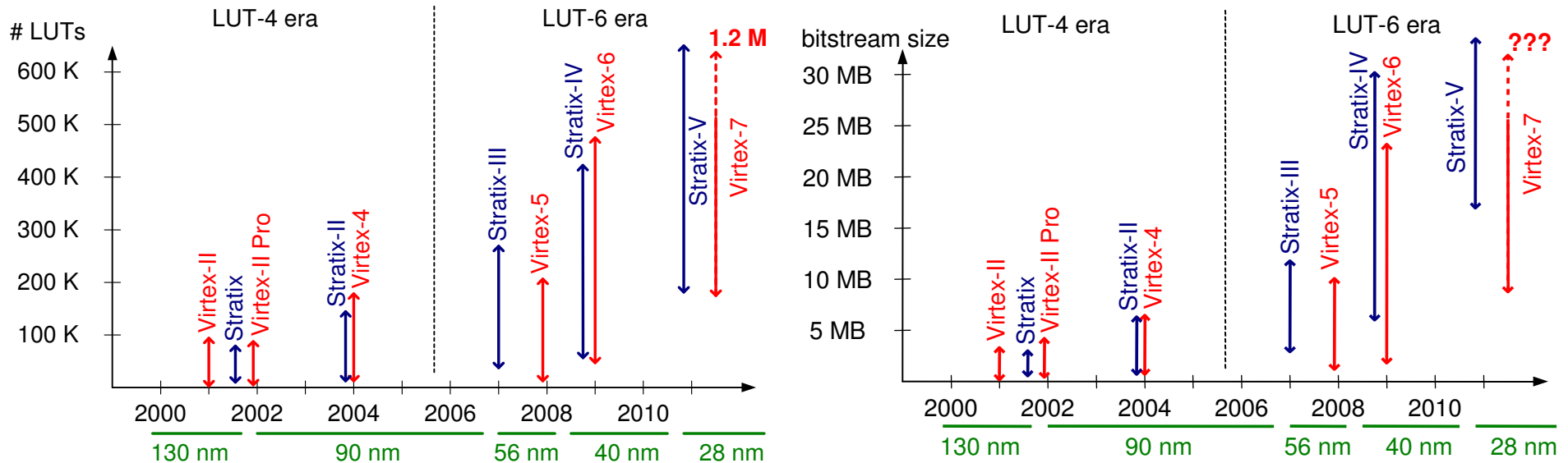
PR Advantages: IP Reuse



**High level of IP reuse → Adapt the component-based system
PR design flow for a general design methodology**

Idea: take as much as possible from an existing environment and add only the application specific parts.

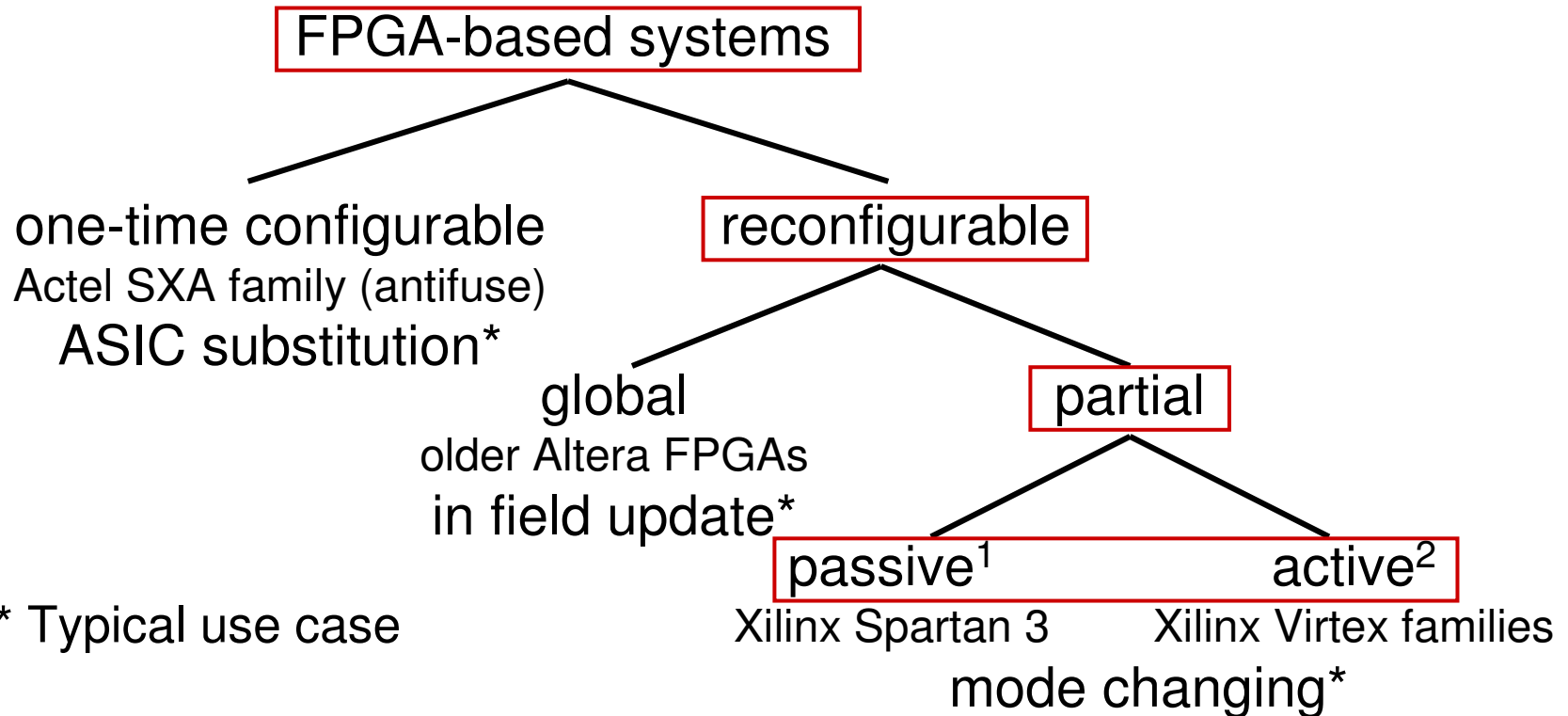
PR Advantages: SEU* Compensation



- Smaller configuration SRAM cells
- Exponential rise in the total amount
- → Increased risk of *single event upsets (SEU)
- Solution: Configuration Scrubbing
 - Continuous reconfiguration during operation (repair)
 - Readback for SEU detection (before committing a result)

RC on FPGAs (Classification)

- Classification of (run-time) reconfigurable FPGA-based systems

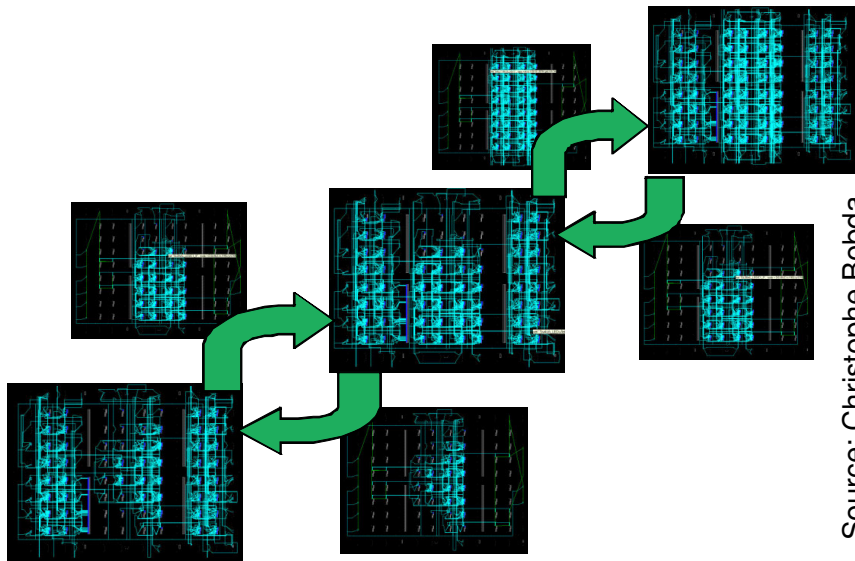


- This lecture focuses on passive¹ partial reconfiguration (interrupt whole FPGA during reconfiguration) and active partial reconfiguration² (untouched parts continue execution) on FPGAs.

Context-Switching on FPGAs

- Partial reconfiguration is also referred as context switching.
- What is the Context of an FPGA?
 - “Context” denotes a “state” which is stored in memory
Located in: 1) FPGA fabric (technology level)
2) Modules (logic level)

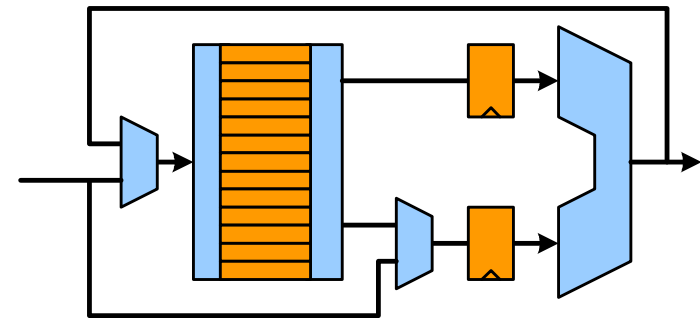
1) Present FPGA configuration



Access via configuration port

2) State of a module

- Register snapshot
- RAM blocks
- External state



Access via configuration port or extra logic (e.g., scan-chain)

Context-Switching on FPGAs

Classification

Technology level (FPGA)

static

dynamic

Logic level (module)

static

- Module runs forever
- Single configuration/
module context
- ASIC-like
(e.g., memory controller)

- Configuration swapping
- Run-to-completion model
(no module context is
considered at start)
(e.g., motion-JPEG)

dynamic

- Multiple module contexts
- on a single configuration
(e.g., multi channel crypto)

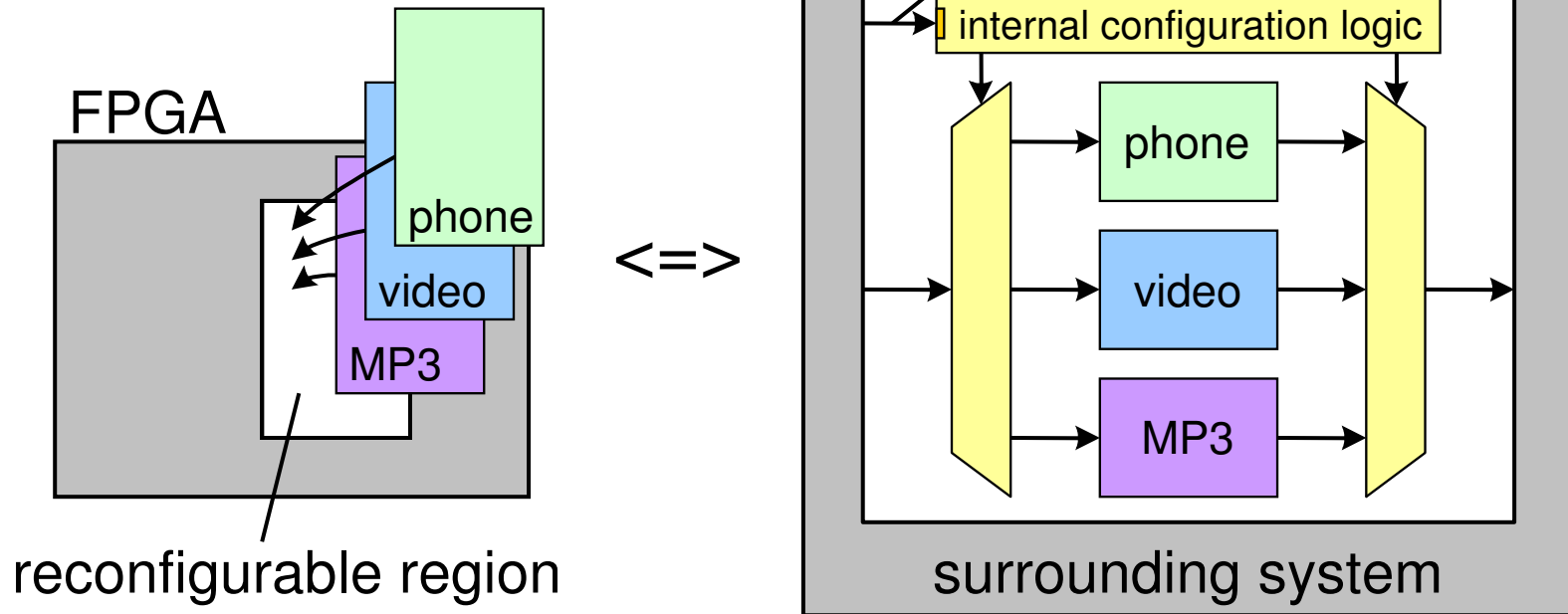
- module preemption and
resuming
- Configuration swapping
- Transparent (like software)
- Examined at UIO in the
COSRECOS project*

- All variants may co-exist in a reconfigurable SoC

*Website: <http://www.matnat.uio.no/forskning/prosjekter/crc>

Baseline Model of Partial Reconfiguration

The time-multiplex model:

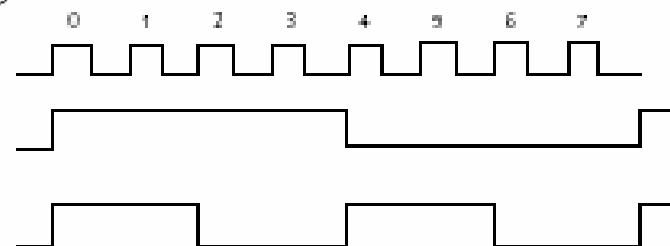
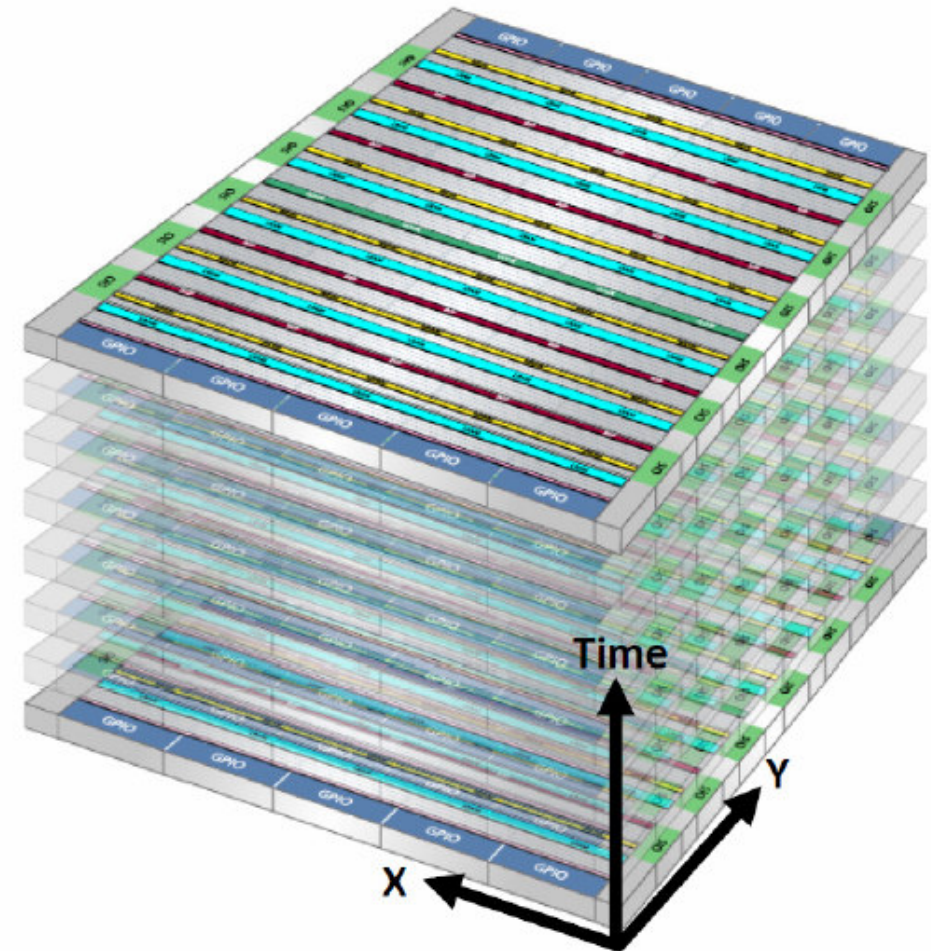
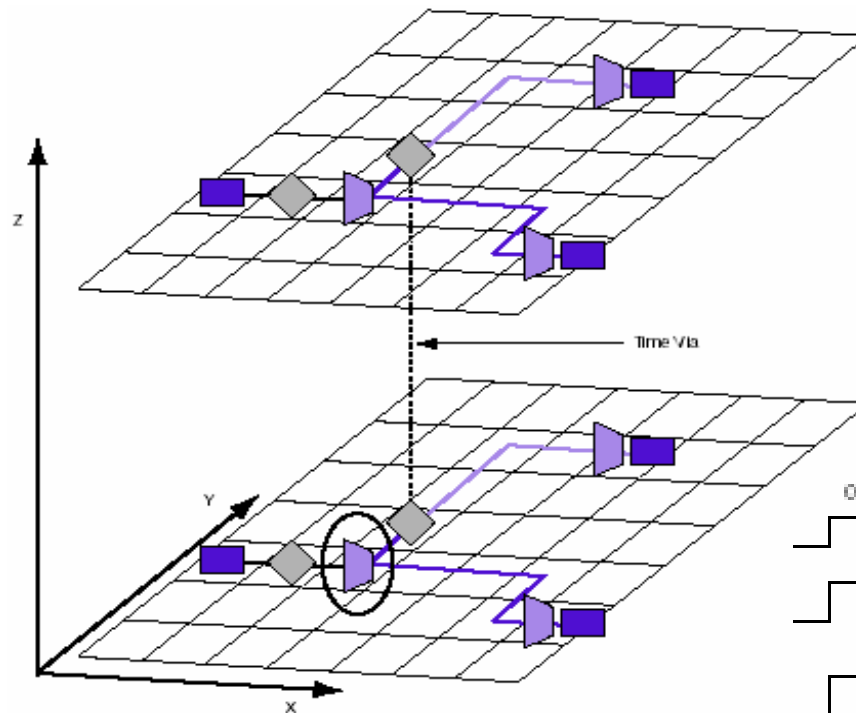


- Activate **one module exclusively** within a reconfigurable region
- Swapping between modules by writing a partial bitstream to a configuration port (defines the configuration time!)
- Bitstream might be written by the FPGA itself → **selfreconfiguration**
- Used by the tools from Xilinx and Altera

PR Time-Granularity (sub-cycle)

Tabula's 3D Architecture

- 8 configuration planes
- Reconfiguration @ 1.6 GHz
- Within netlist reconfiguration (uses forwarding registers called „time via“)



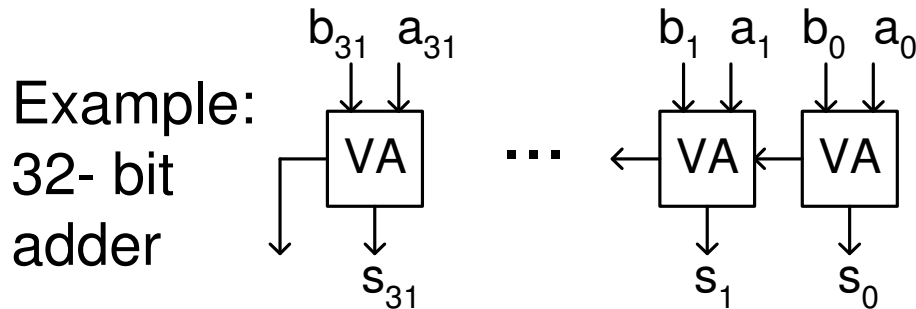
8 folds @ 1.6 Ghz

200 MHz user clock

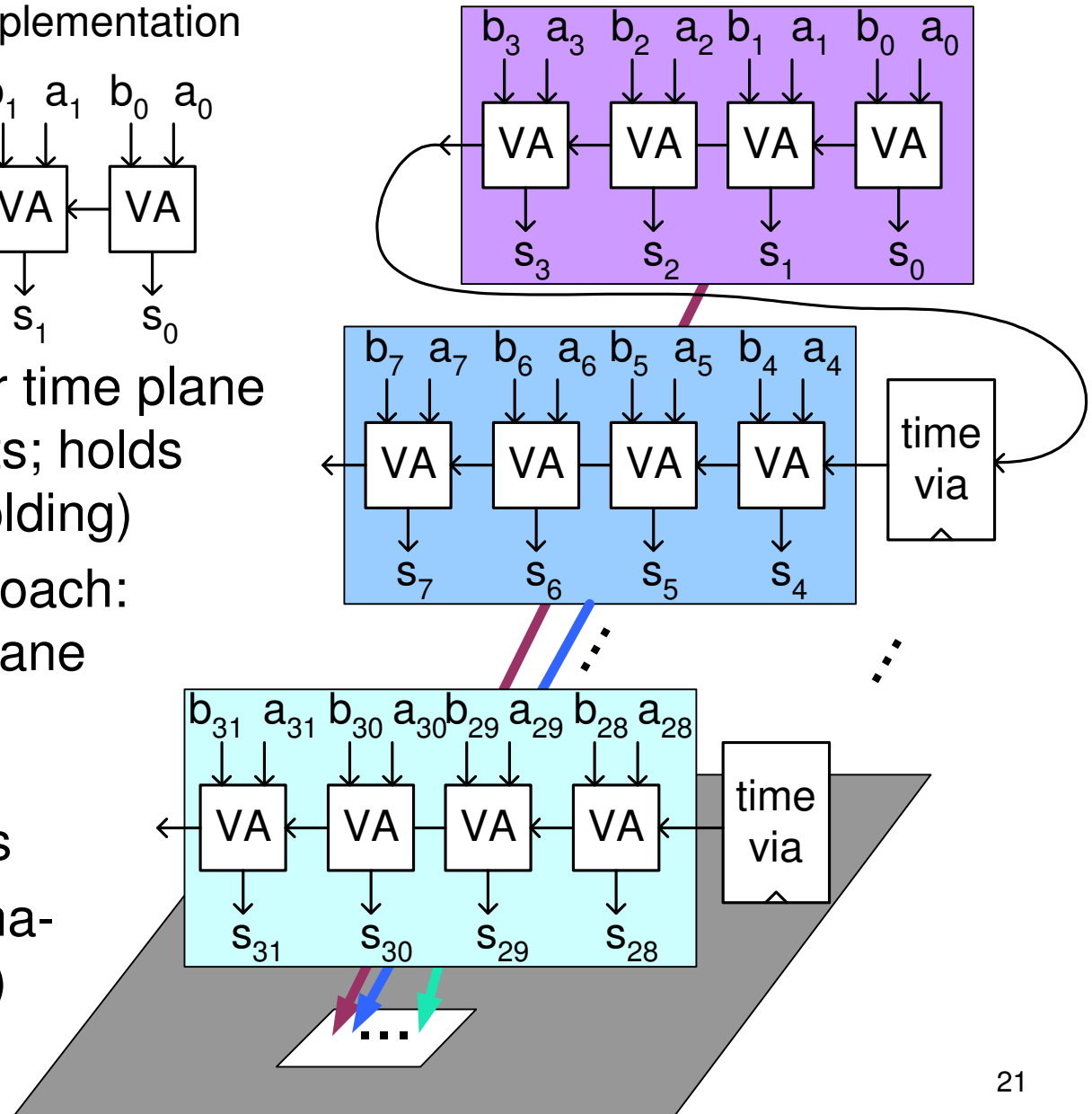
400 MHz user clock

PR Time-Granularity (sub-cycle)

traditional FPGA implementation



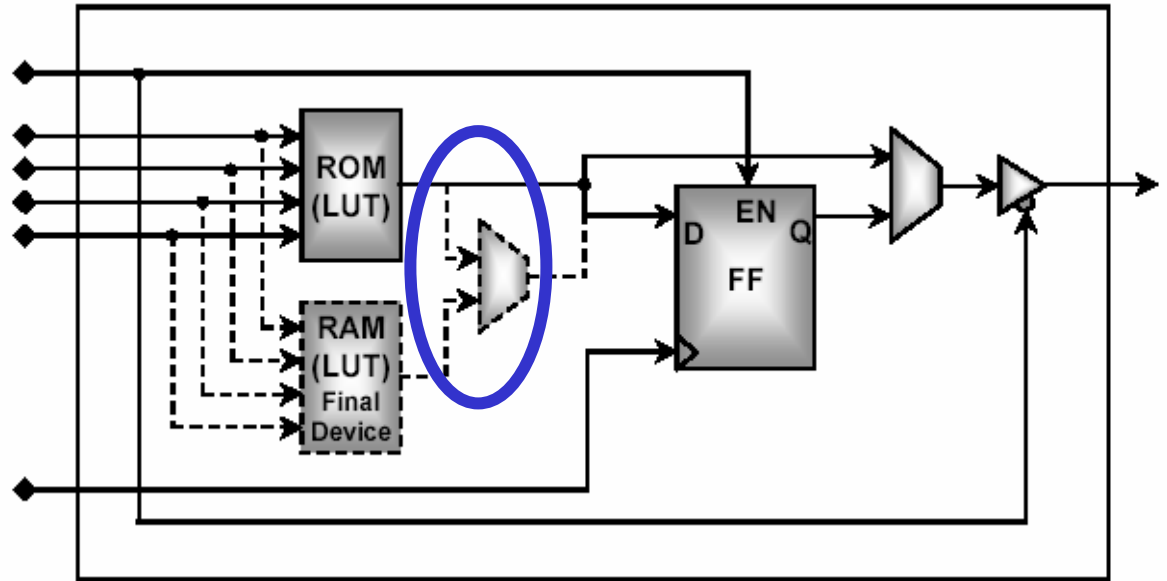
- one memory access per time plane (virtually 8 memory ports; holds only when not space folding)
- Difficult to rate this approach: extra multiplexer for plane switching have to be mapped on a 2D chip → longer routing paths
- Difficult tools (manual manipulation or simulation)
- <http://www.tabula.com>



PR Time-Granularity (single-cycle)

Multi-context FPGAs

- originally proposed by Scalera & Trimberger
- single cycle configuration swapping
- idea: duplicating all configuration bits for each “plane” and multiplexing between planes
- Problem: extra multiplexer required for each configuration bit
- All planes have to be mapped on a 2D chip (3D → 2D mapping) → longer routing between the primitives <=> lower performance
- Bad idea for FPGAs: most of the FPGA die area is spent on configuration SRAM cells) usefull only for coarse-grained architectures
- Better: multiplexing between different areas on the FPGA



PR Time-Granularity (multi-cycle)

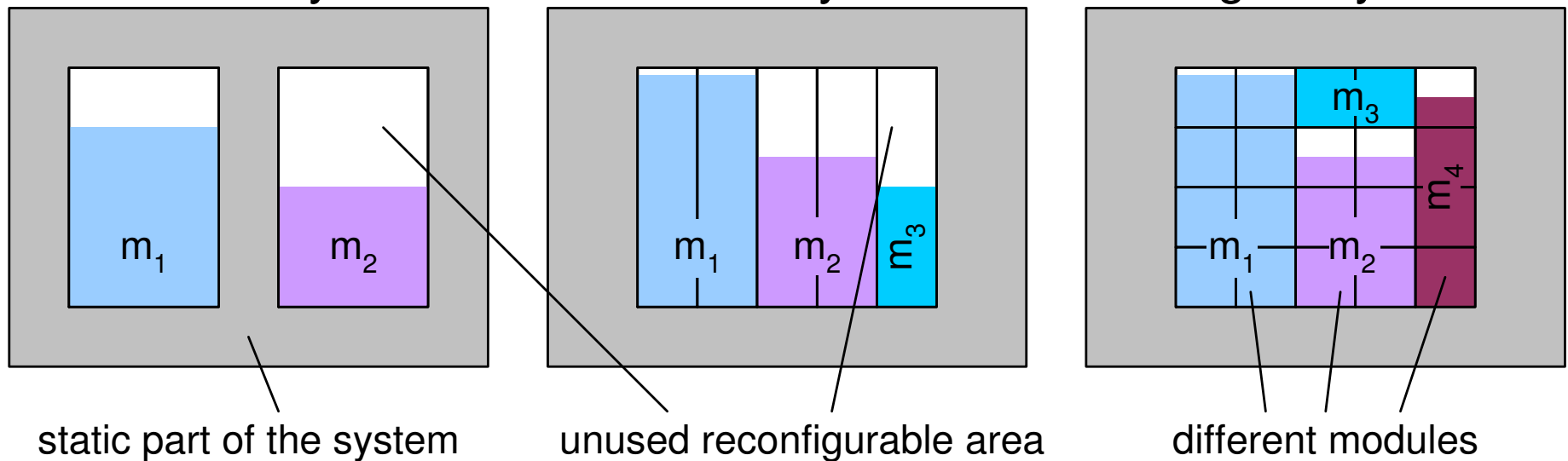
Configuration by writing a new configuration bitstream to the device

- normal case for all FPGAs from Xilinx and Altera (starting with the Stratix-5 family)
- rapid partial module swapping
(e.g., swapping within a frame in a video processing system)
- mode changing / field update
(typically used in combination with full FPGA reconfiguration, e.g., in measurement equipment when changing settings or for prototyping (ASIC emulation))

PR in Time and Space

So far, we have only considered to have one module exclusively placed with a reconfigurable region (temporal partial reconfiguration) → extension to **multi-module placement** of partially reconfigurable modules (spatial partial reconfiguration)

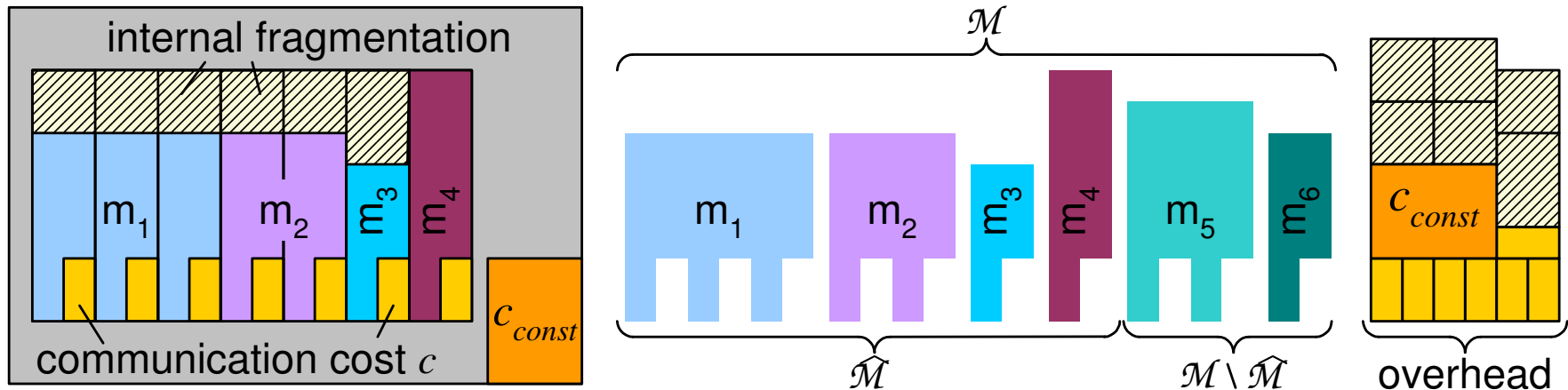
Possibilities for tiling the reconfigurable area into resource slots:
island style slot style grid style



As smaller the slots, as lower the internal fragmentation (the waste of logic resulting from fitting any sized module into a tile-grid (i.e., clustering the FPGA area into regular groups of resources))

PR in Time and Space: Efficiency

PR paradox: Runtime reconfiguration is brilliant, but not used!



- Internal fragmentation is dominating the overhead
- Can be optimized with small slots \rightarrow 2D placement (but might result in additional cost for the communication)
- 2D enhances BRAM/DSP utilization
- 2D is obligatory for newer FPGA Architectures (Virtex-5/6)

Requires adequate on-FPGA communication architectures

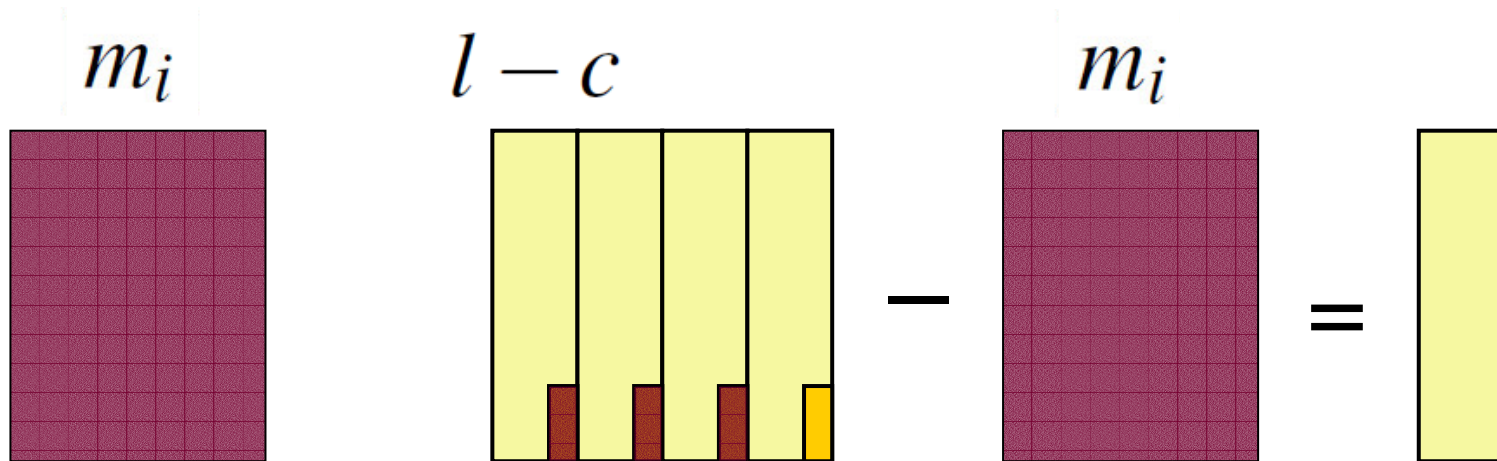
- Buses
- Point-to-point connections

Optimal Resource Slot Size

- Internal fragmentation results from fitting modules into a grid of fixed resource slots.
- Analog: storing files in a filesystem with fixed clusters
- Average overhead of a module set of modules:

$$\overline{O} = \frac{1}{|\mathcal{M}|} \cdot \sum_{i=1}^{|\mathcal{M}|} \left(\left\lceil \frac{m_i}{l-c} \right\rceil \cdot l - m_i \right)$$

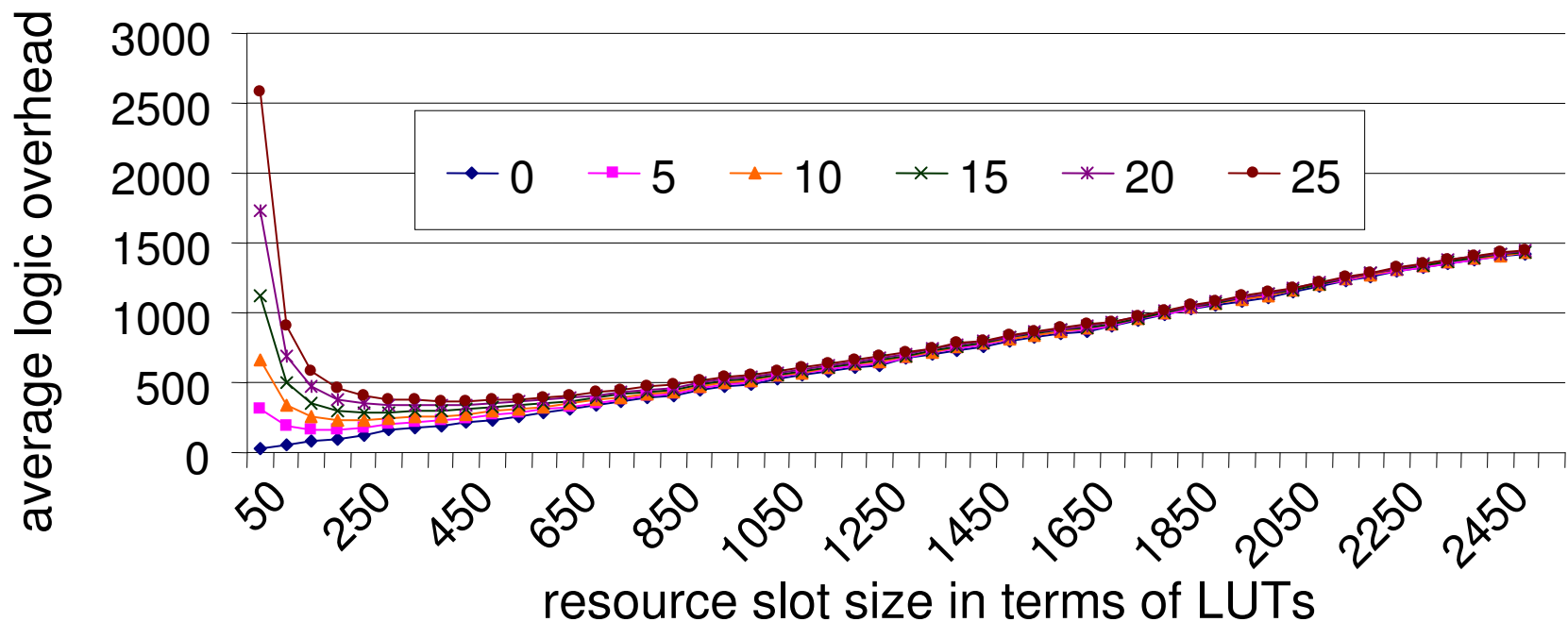
l : resources in a slot
 c : communication
 m_i : resources of module i



Optimal slot size depends on the modules and communication cost

Optimal Resource Slot Size

- Impact of the resource slot size and the communication cost on the average module overhead
- Scenario: 9701 modules with 300, 301, ..., 10000 LUTs with a communication cost of 0, 5, ..., 25 LUTs per slot



Result: optimal slot size ~200–300 LUTs or ~25–40 CLBs

Optimal Resource Slot Size

$$\bar{O} = \frac{1}{|\mathcal{M}|} \cdot \sum_{i=1}^{|\mathcal{M}|} \left(\left\lceil \frac{m_i}{l-c} \right\rceil \cdot l - m_i \right)$$

l : resources in a slot
 c : communication
 m_i : resources of module i

Discussion:

- If $m_i \gg l$ the overhead converges to $l/(l-c)$, meaning that for large modules (with many resource slots) the internal fragmentation becomes negligible.
- The optimal slot size can be computed by differentiating the average module overhead \bar{O} with respect to the slot size l .
- As the ceiling function is discontinuous, its bounds are considered:

$$\frac{|m_i|}{l-c} \leq \left\lceil \frac{|m_i|}{l-c} \right\rceil < \frac{|m_i|}{l-c} + 1$$

- Lower bound: perfect fit
 - Upper bound: one slot is almost unused
- $$\hat{O} = \frac{1}{|\mathcal{M}|} \cdot \sum_{i=1}^{|\mathcal{M}|} \left(\left(\frac{|m_i|}{l-c} + 1 \right) \cdot l - |m_i| \right)$$

Optimal Resource Slot Size

Discussion:

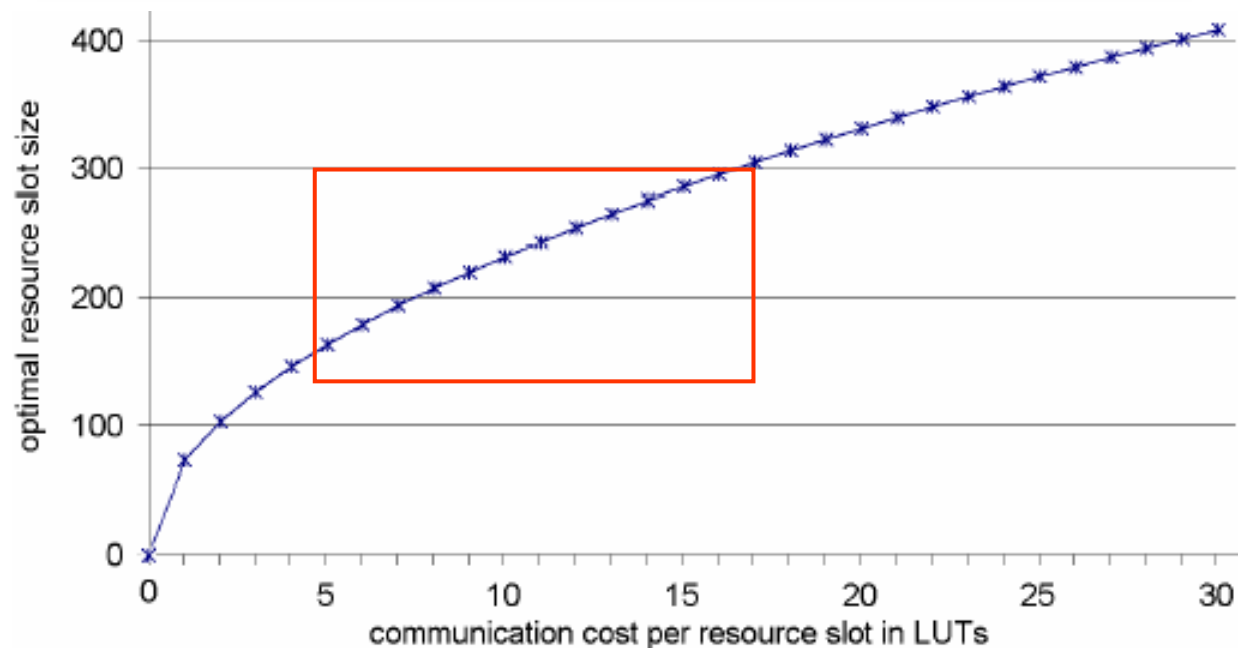
- Upper bound: one slot is almost unused

$$\hat{O} = \frac{1}{|\mathcal{M}|} \cdot \sum_{i=1}^{|\mathcal{M}|} \left(\left(\frac{|m_i|}{l-c} + 1 \right) \cdot l - |m_i| \right)$$

- Worst case:

$$\frac{\partial \hat{O}(l)}{\partial l} = 0 \Rightarrow \hat{l}_{opt} = c \pm \sqrt{\frac{1}{|\mathcal{M}|} \cdot c \cdot \sum_{i=1}^{|\mathcal{M}|} |m_i|}.$$

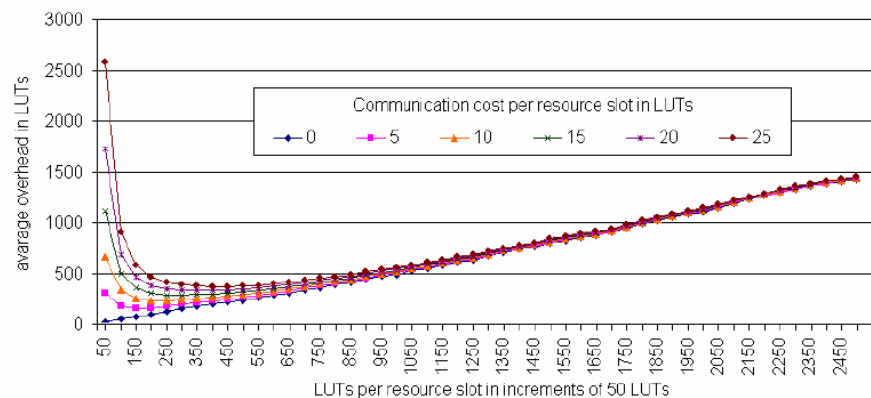
- Average case:
(achievable only with 2D grid style placement)



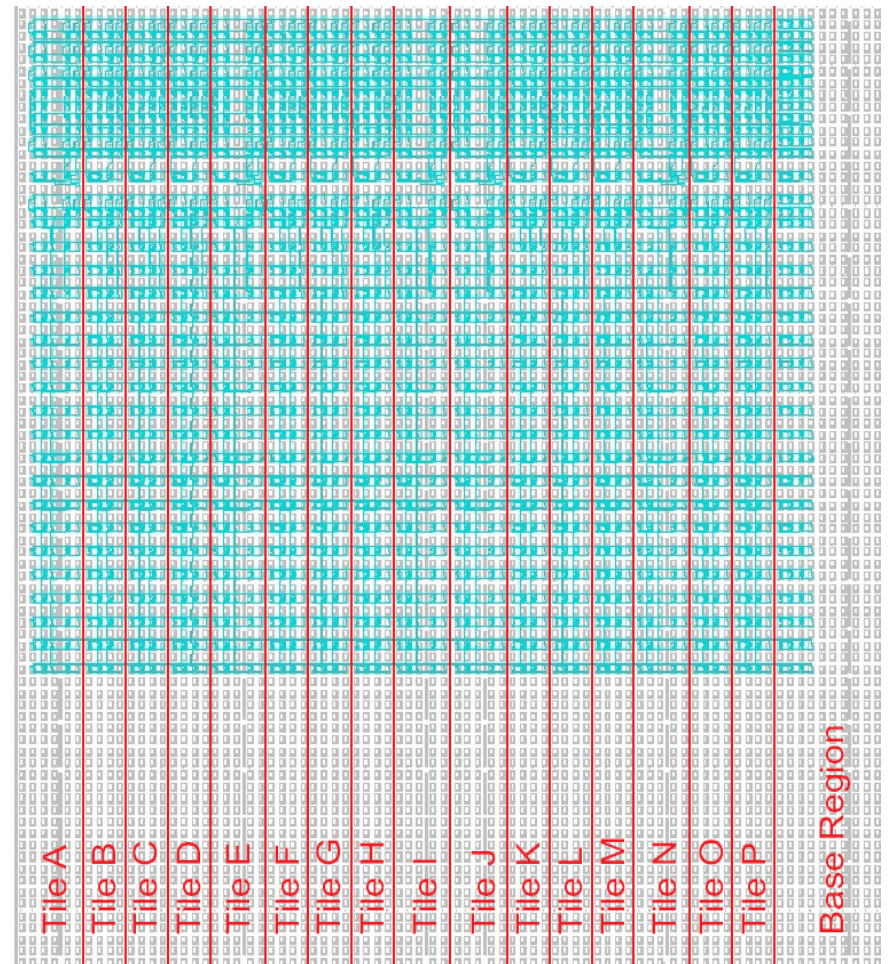
Optimal Resource Slot Size

One of the best published solutions:

- Hagemeyer et al., *Design of Homogeneous Communication Infrastructures for Partially Reconfigurable FPGAs*
ERSA, USA 2007.
- Master and slave support (32 bit)
- 16 sockets (XC2V4000)
- Communication cost: 8554 LUTs (~three 32-bit CPU-cores)
- No I/O support
- Resource slot size: 2560 LUTs



x

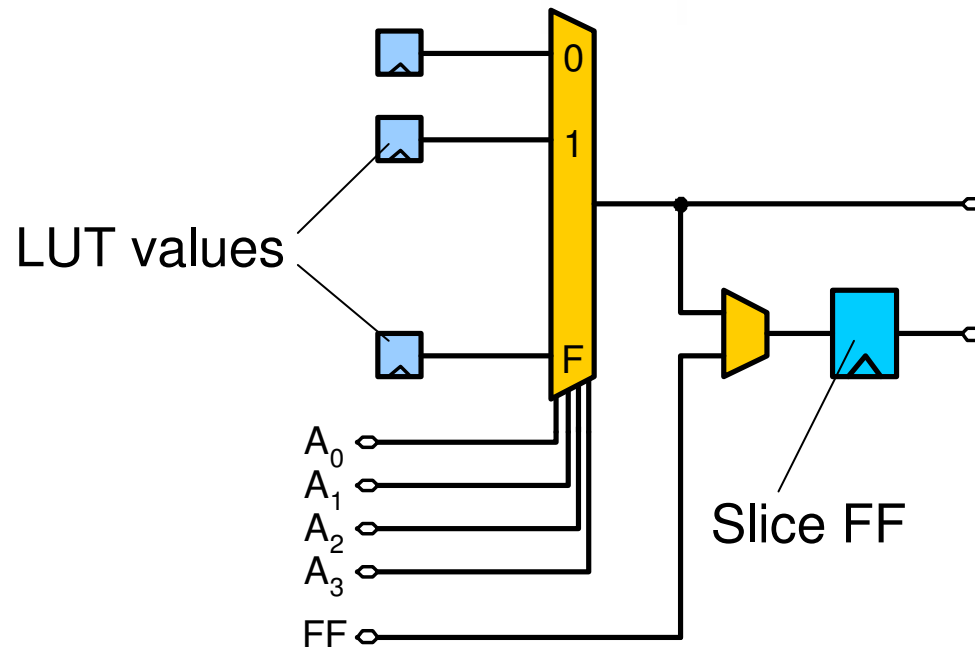


Catastrophic communication cost and too large resource slots

PR Space-Granularity (bitstream)

The behavior or structure of a system can be changed by small manipulations of the configuration bitstream.

- Manipulation of the routing (switch matrix multiplexer)
- Changing logic functions
example: AND \rightarrow OR

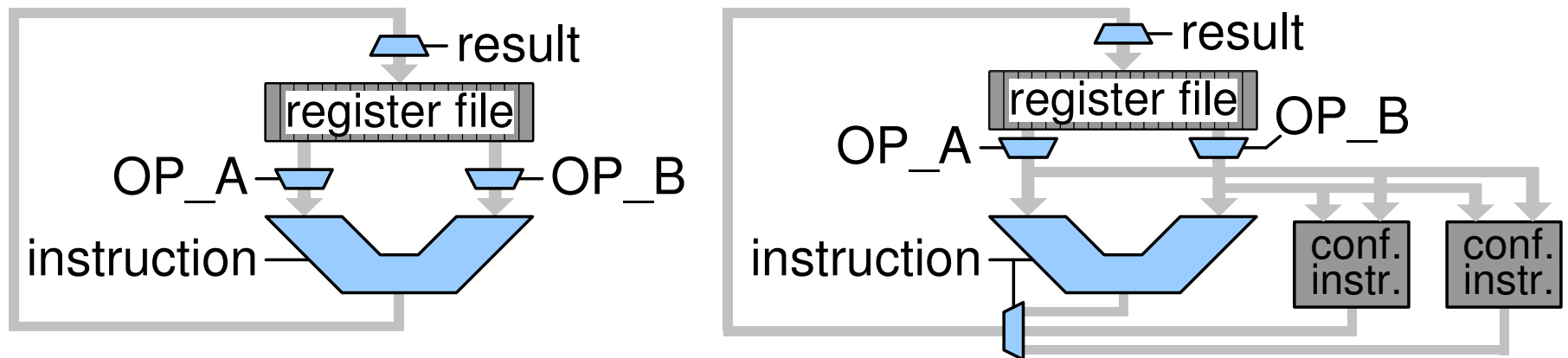


	A ₃ , A ₂ , A ₁ , A ₀	LUT-value AND gate	LUT-value OR gate
0	0000	0	0
1	0001	0	1
2	0010	0	1
3	0011	0	1
4	0100	0	1
5	0101	0	1
6	0110	0	1
7	0111	0	1
8	1000	0	1
9	1001	0	1
A	1010	0	1
B	1011	0	1
C	1100	0	1
D	1101	0	1
E	1110	0	1
F	1111	1	1

PR Space-Granularity (small modules)

Sometimes, even small modules can materially speed-up a system.

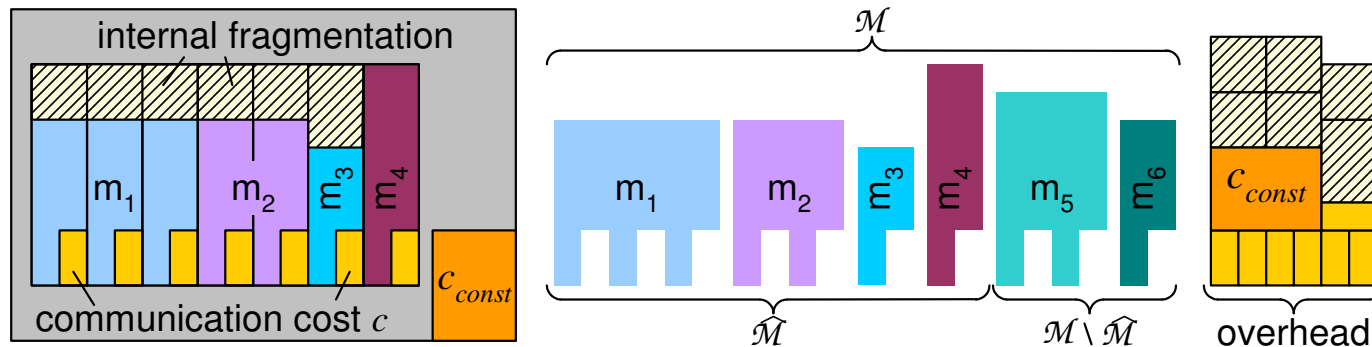
- Example: reconfigurable customized instruction set extensions (e.g., with instructions for CRC, DES round, bit swapping)



- Relatively small configurable instructions can speed up execution by at least an order of magnitude. (NIOS, GARP, DISC)
- Typically non concurrent operation (blocking the ALU)
- Difficulty: instructions have a high pin count per logic
→ Interfaces have to be ultra efficient!
- Different logic requirements → flexible instruction placement

PR Space-Granularity (large modules)

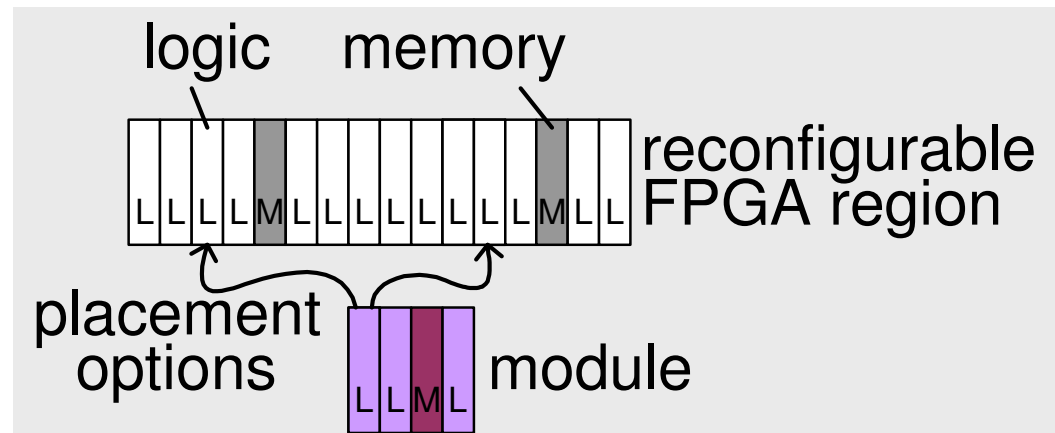
Typically, systems consist of multiple concurrently working modules.



- Difficulty: modules have different resource requirements

- Logic
- Memory
- Multipliers

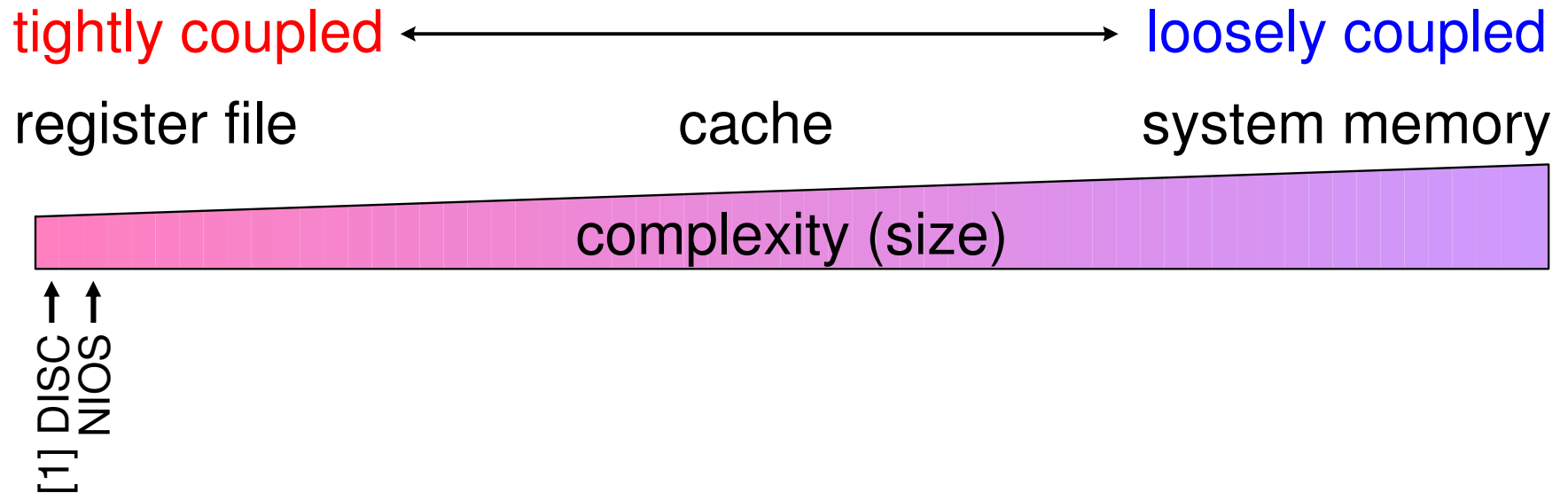
- Placement restrictions (string matching problem)



→ Interfaces should allow two-dimensional module placement

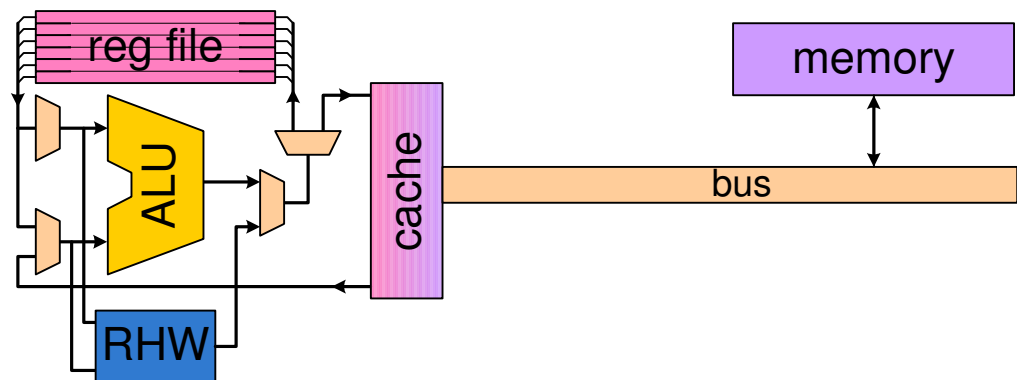
- Further: placement impacts the communication!

PR Space-Granularity (module coupling)

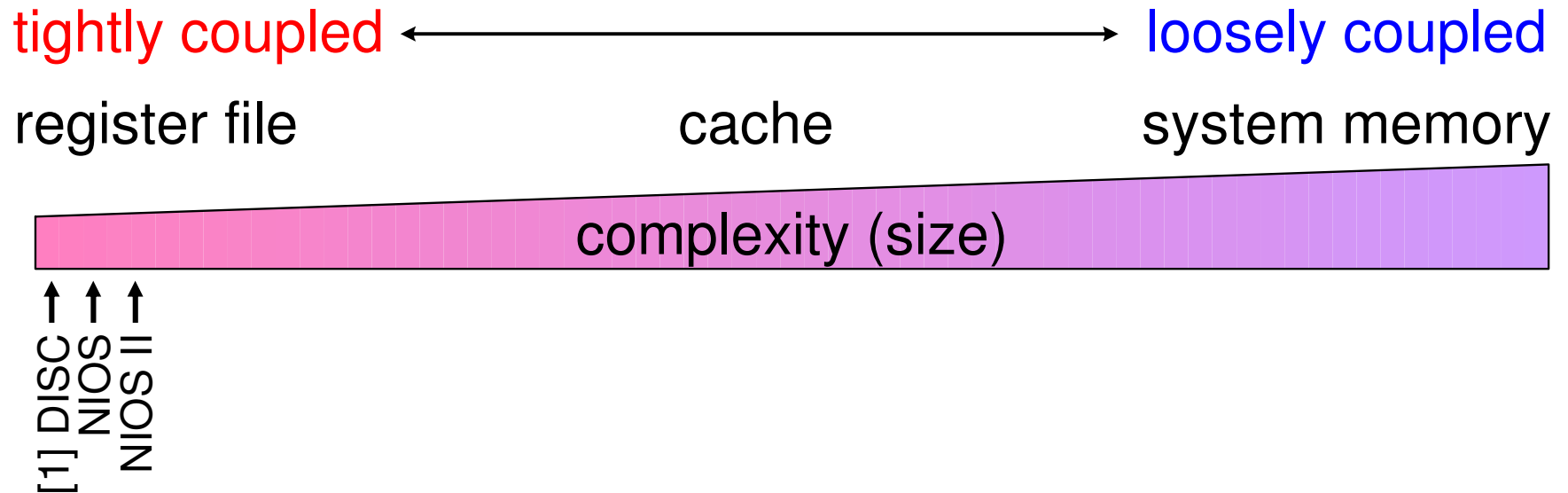


- Reconfigurable HW in parallel to the ALU

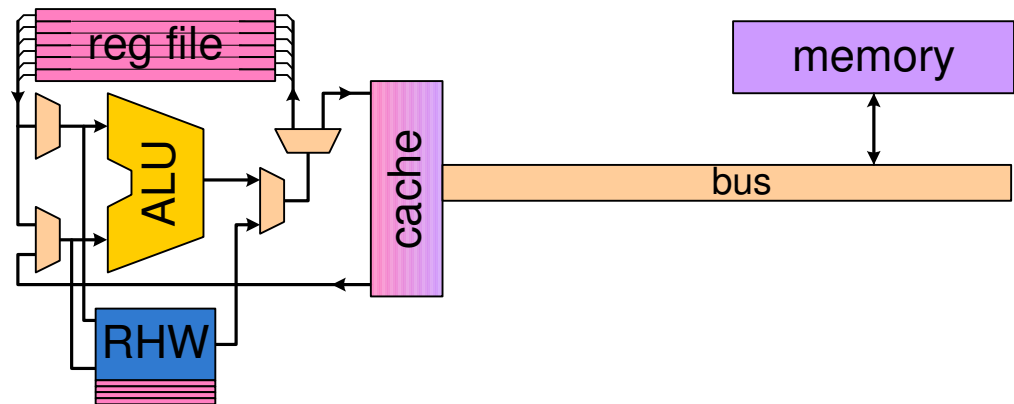
[1] Wirthlin and Hutchings:
*DISC: Dynamic Instruction
Set Computer* (FCCM 1995)



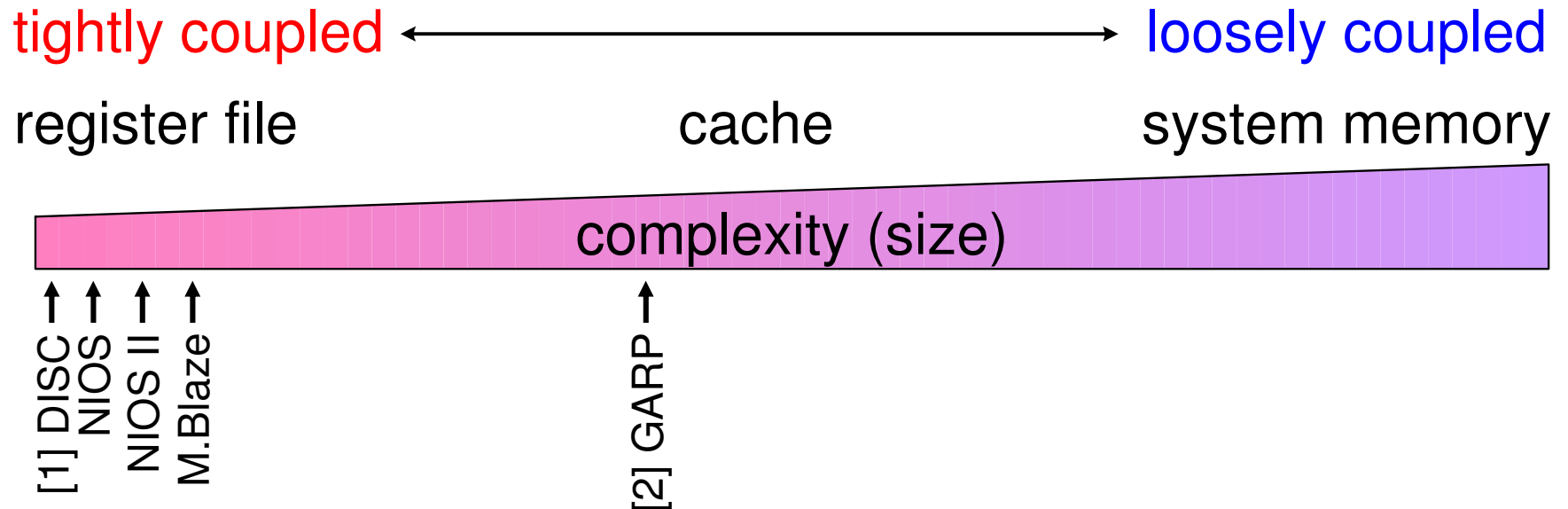
PR Space-Granularity (module coupling)



- Reconfigurable HW in parallel to the ALU
- Module may contain own register file

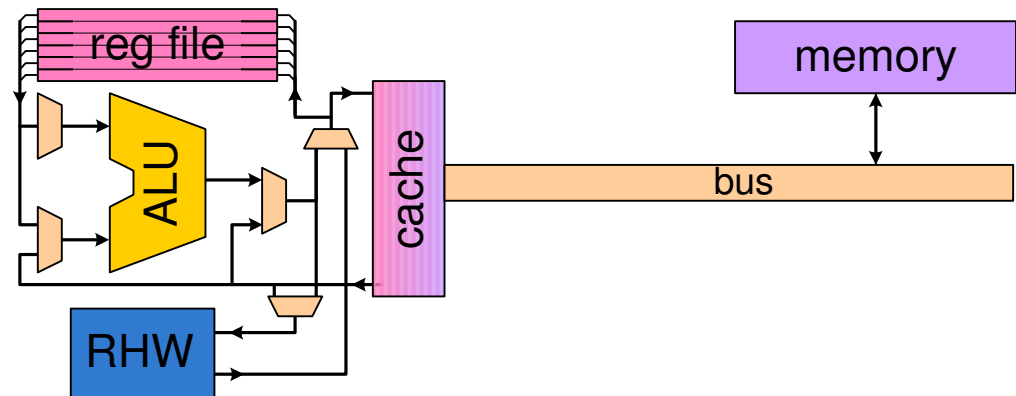


PR Space-Granularity (module coupling)

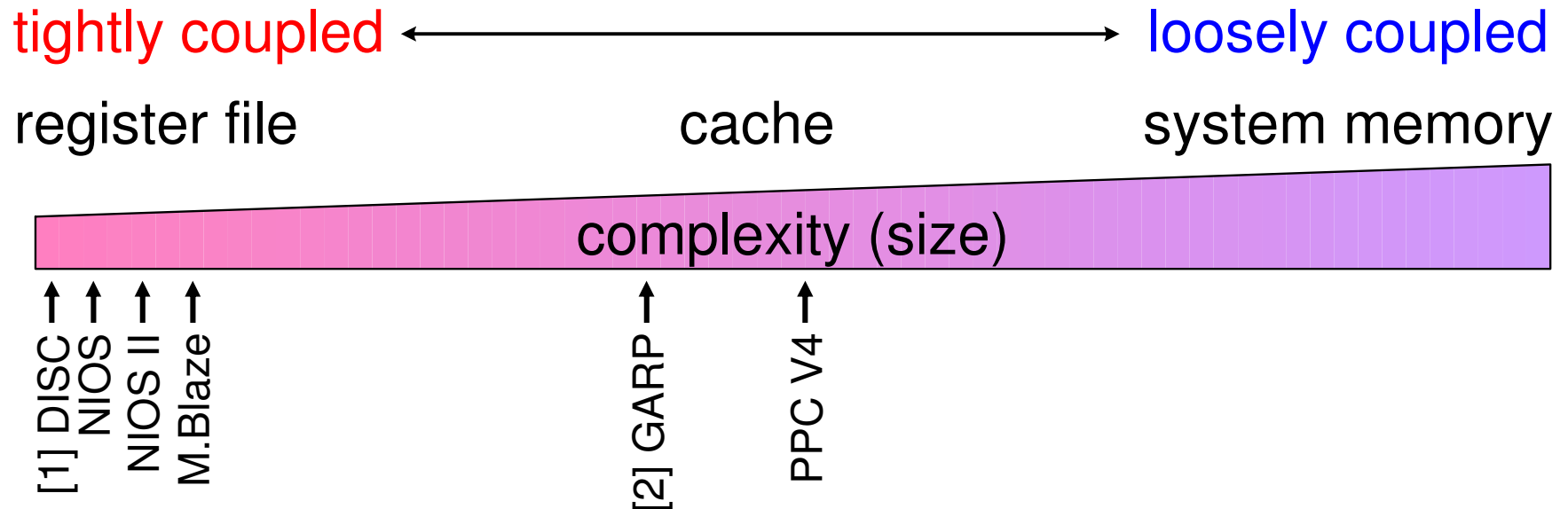


- Coprocessor-like coupling of the reconfigurable HW

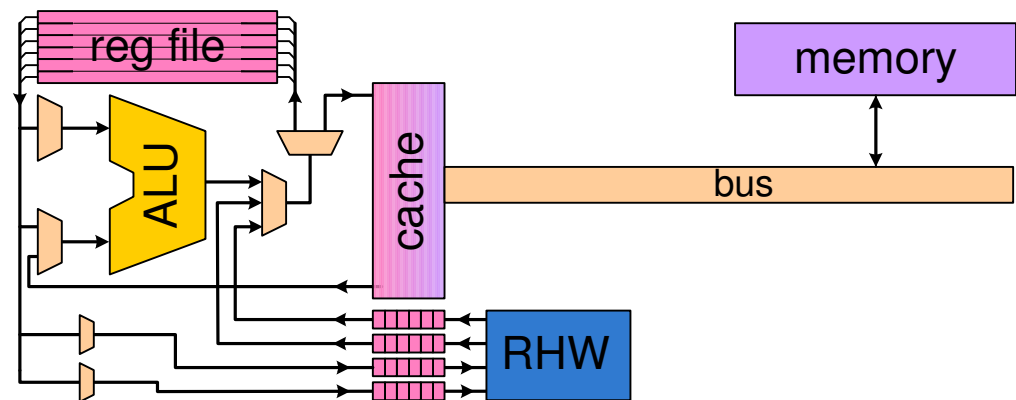
[2] Hauser and Wawrzynek (FCCM 97):
GARP: A MIPS Processor with a Reconfigurable Coprocessor



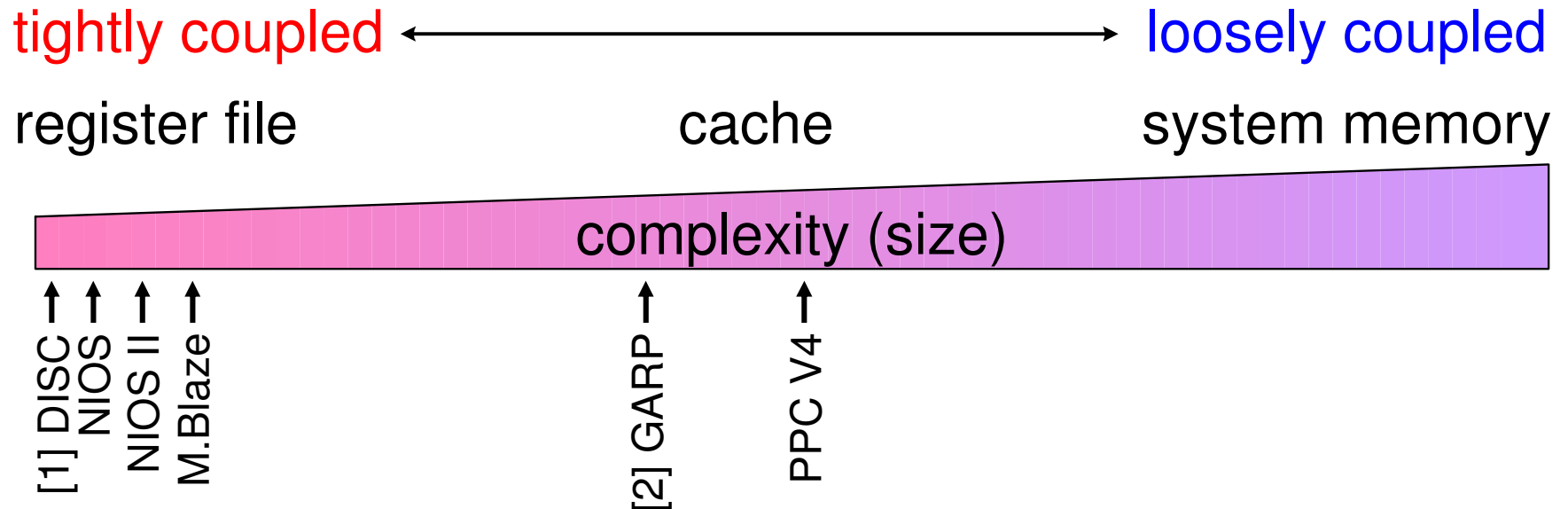
PR Space-Granularity (module coupling)



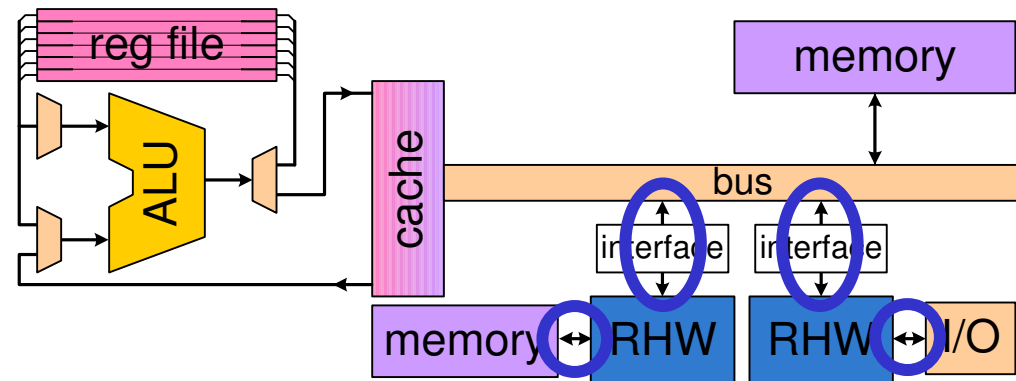
- Coprocessor-like coupling of the reconfigurable HW
- Decoupled by Fifo channels (FSL-Fifo)
- Parallel execution



PR Space-Granularity (module coupling)



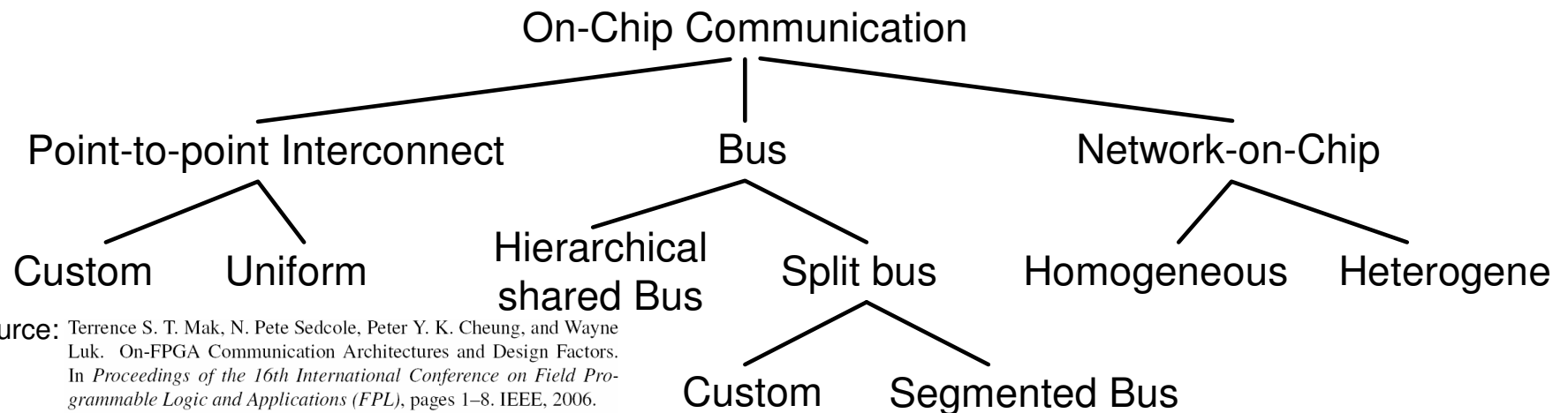
- Connect reconfigurable HW to the memory bus
- Common FPGA-based approaches require an interface (in the easiest case a “bus-macro”)



On-FPGA Communication

Goal: an efficient on-FPGA communication architecture that supports the grid-style module placement.

- Classification of different on-chip communication architectures:

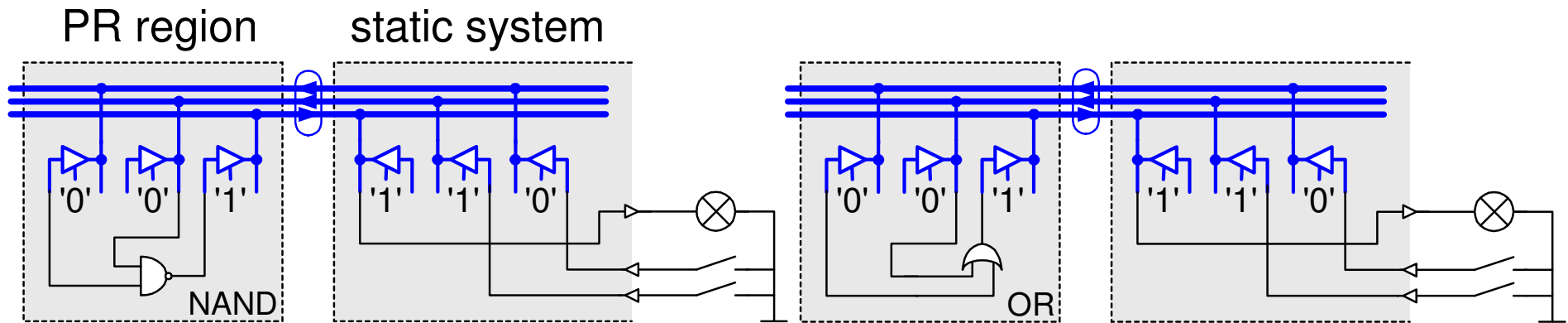


source: Terrence S. T. Mak, N. Pete Sedcole, Peter Y. K. Cheung, and Wayne Luk. On-FPGA Communication Architectures and Design Factors. In *Proceedings of the 16th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE, 2006.

- for FPGAs:
 - buses (reading / writing of registerfiles and DMA)
 - point-to-point links
(I/O-pin connection and data streaming)

On-FPGA Communication (History)

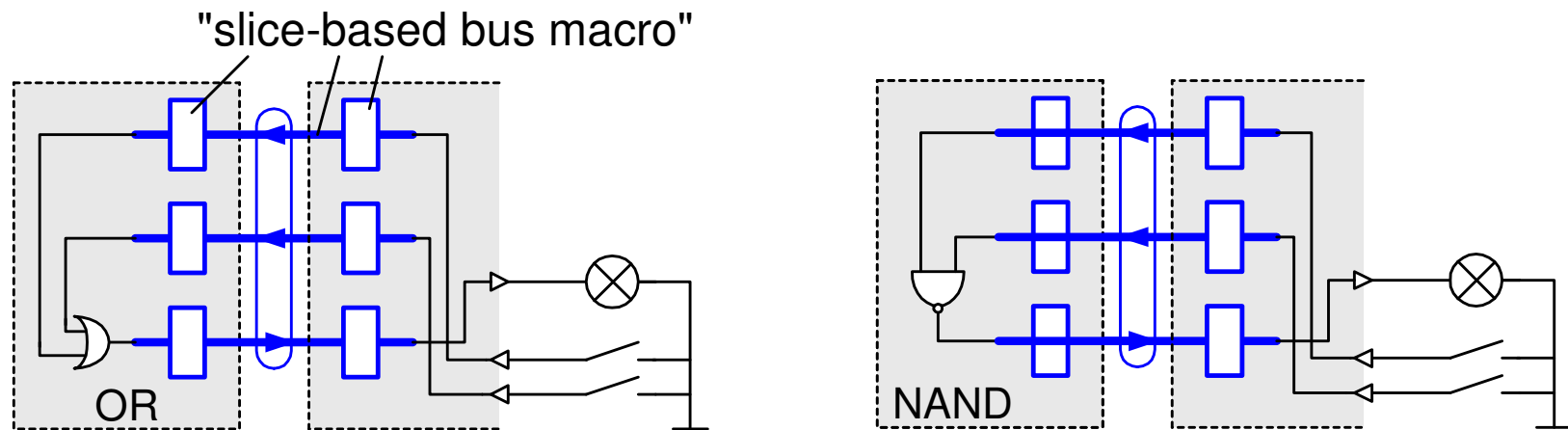
- Progress in Partial reconfiguration (physical implementation) using the Xilinx tools over the last decade:
- Fundamental problem: binding of the partial module entity signals to fixed routing resources of the FPGA fabric „module plug“



- „Xilinx Bus Macros“ for constraining the routing between the static system and one or more PR regions (introduced 2002)
 - Costs two TBUFs per signal wire (in terms of latency and area)
 - Placement restrictions & device support

On-FPGA Communication (History)

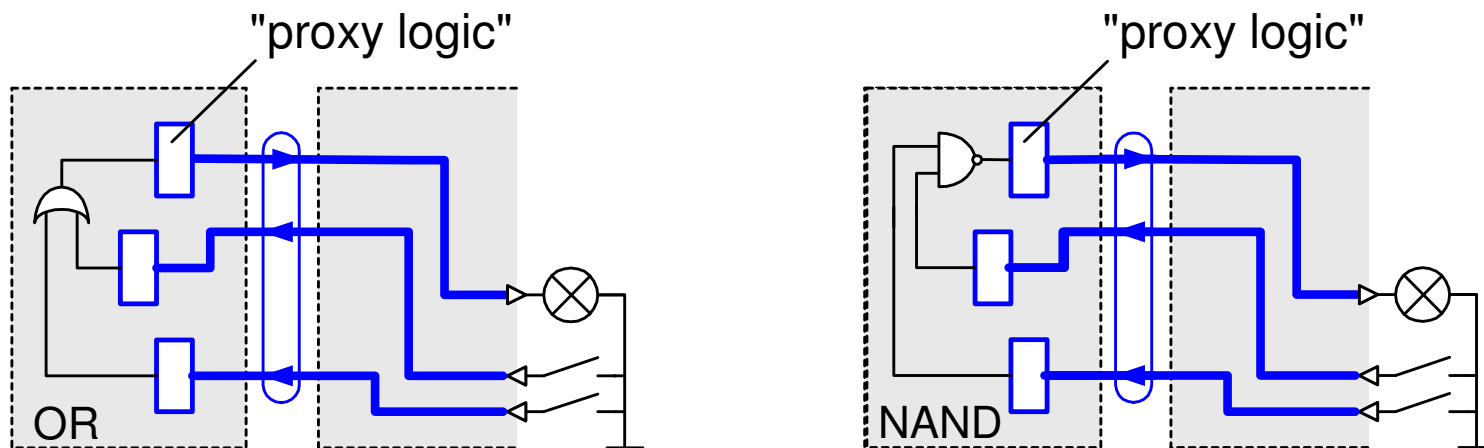
- Progress in Partial reconfiguration using the Xilinx tools over the last decade:



- „Slice-based Bus Macros“ (proposed by Hübner et al. in 2004)
 - More flexible (higher density of wires, more placement options)
 - Works with all Xilinx FPGAs (Virtex-II Pro: last FPGA with TBUFs)
 - Costs two LUTs per signal wire (in terms of latency and area)

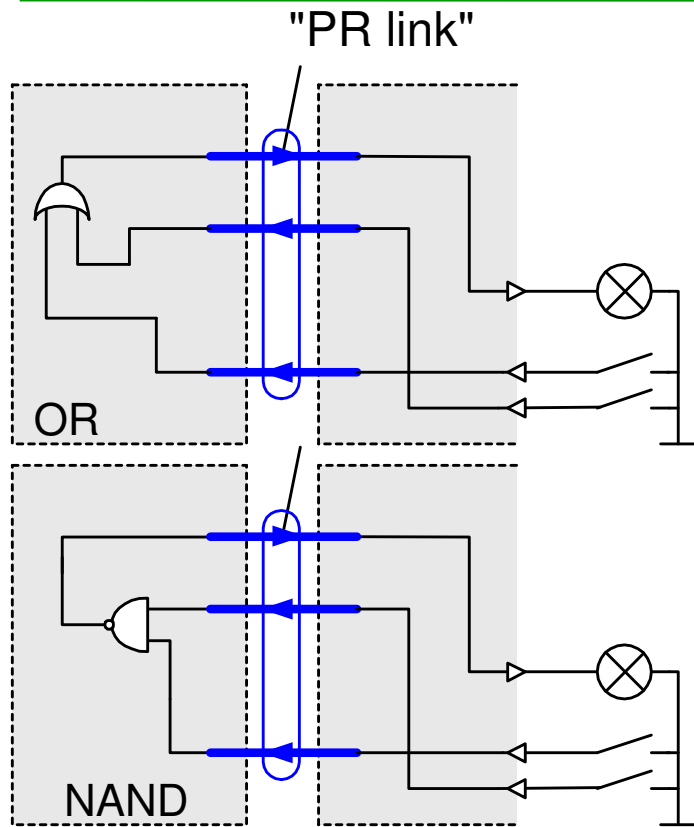
On-FPGA Communication (History)

- Progress in Partial reconfiguration using the Xilinx tools over the last decade:



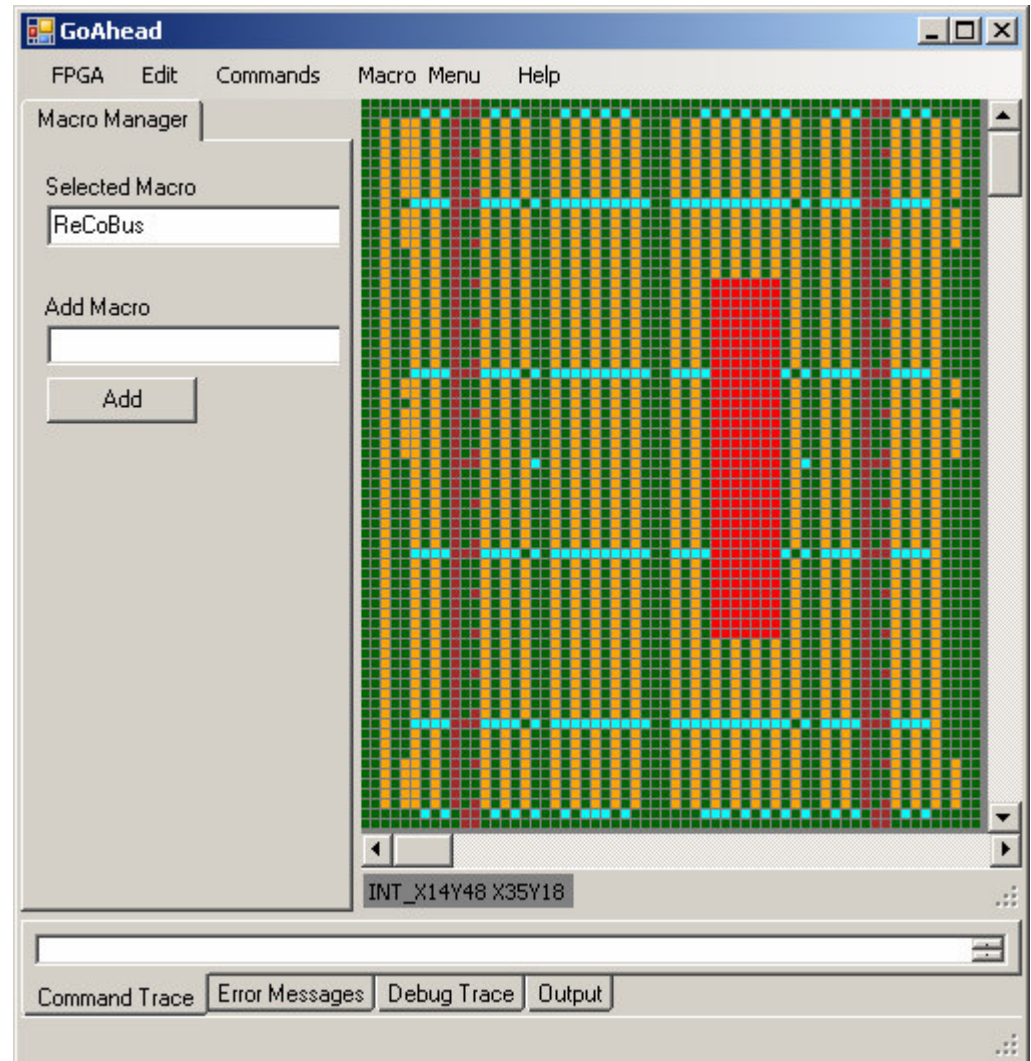
- „Proxy logic“ (released for some devices by Xilinx in 2009)
 - Automatic placement of anchor primitives
 - Costs one LUT per signal wire (in terms of latency and area)
 - Only provided for some devices
- Same approach is used in the upcoming Altera PR flow

On-FPGA Communication



„PR links“

Binding entity signals to the wires crossing the border to a reconfigurable module.



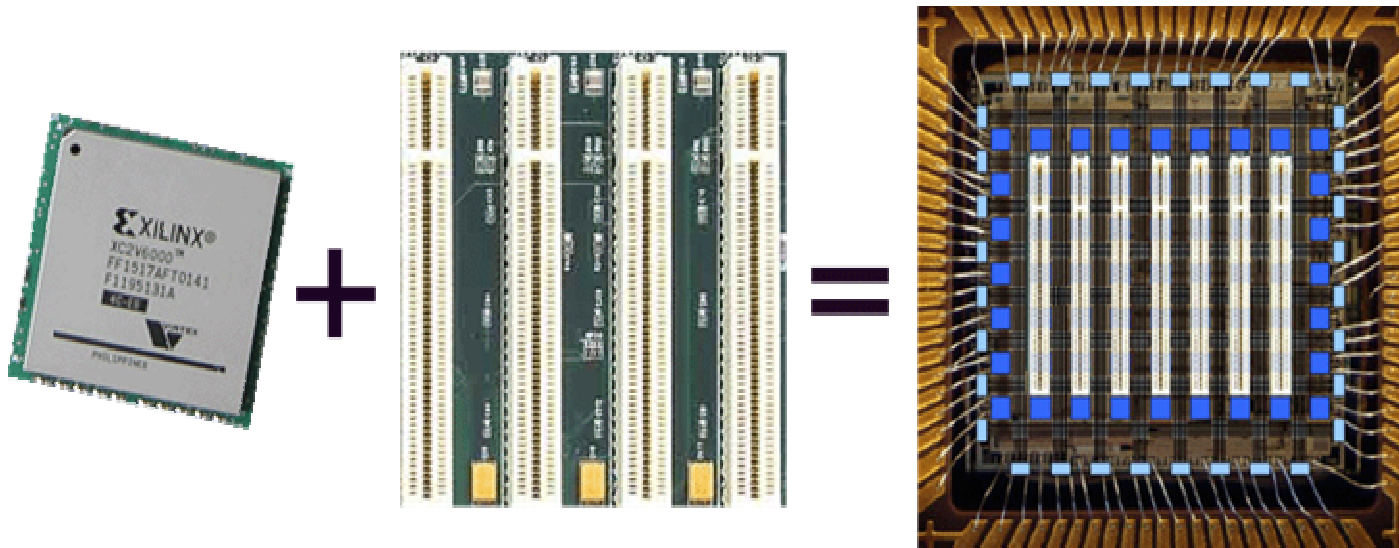
➤ No logic overhead, cleaner design flow, supports S6 (V5, V6)

On-FPGA Communication: Buses

Bus macros are best suited to integrate modules into islands!
The following slides present structured communication architectures for slot-based (1D) or grid-style (2D) module placement

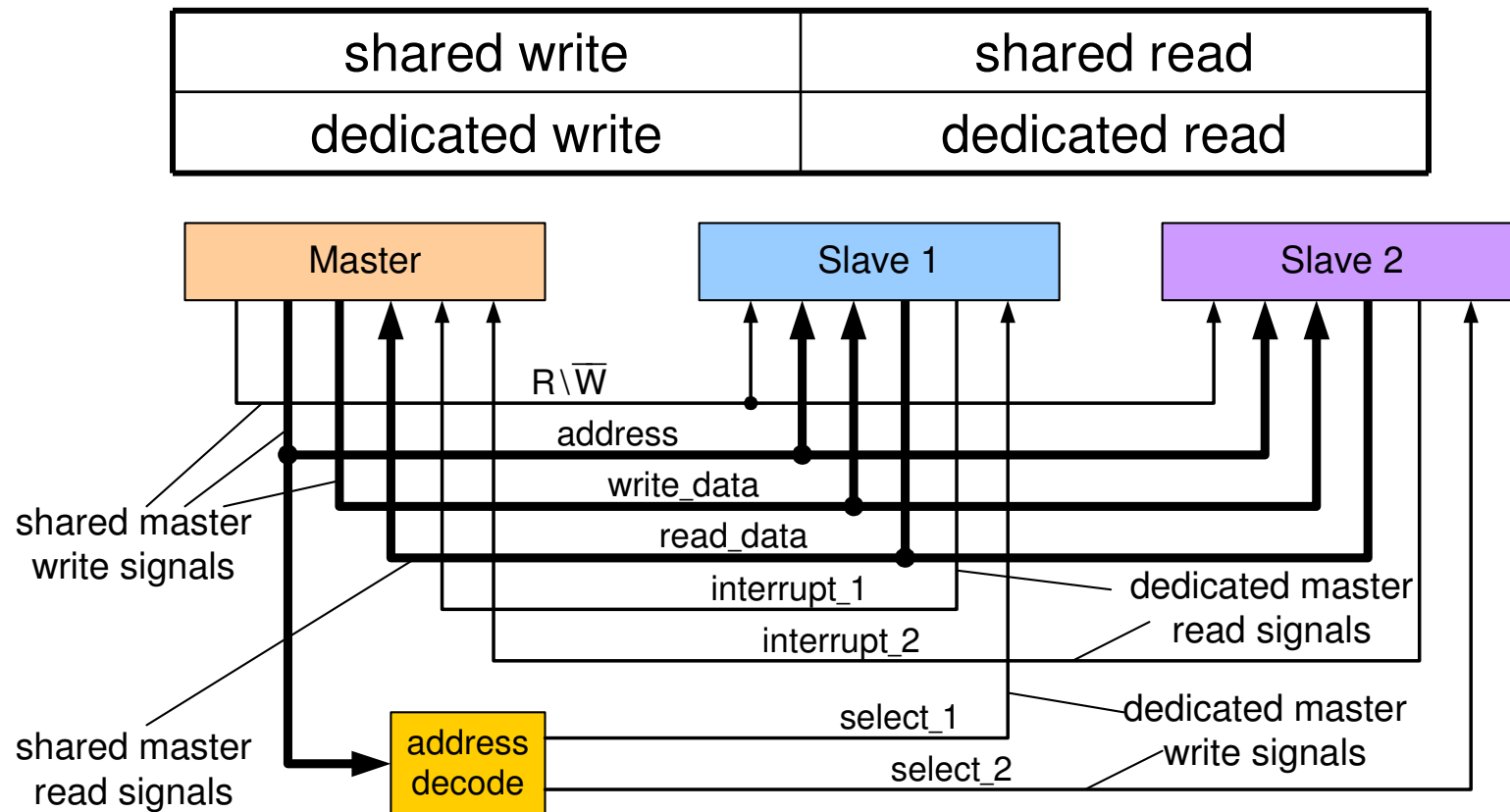


The Simple Formula for Building Bus-based Reconfigurable Systems



ReCoBus Communication

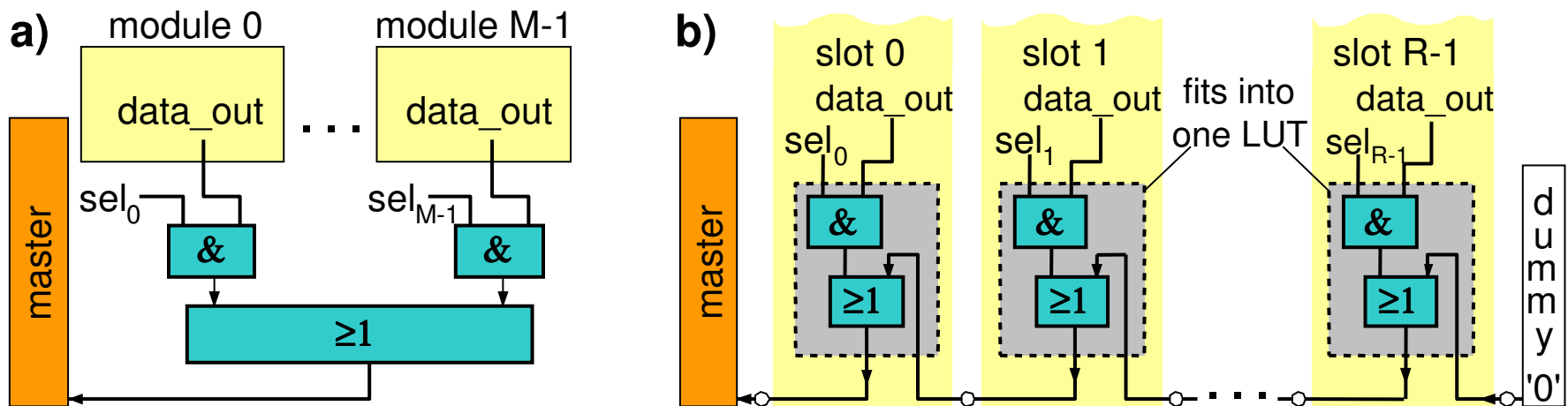
All bus protocols can be implemented by the use of four signal classes:



- Example: connecting an interrupt signal from a slave to an interrupt controller is basically the same problem as connecting a bus request from a master module to an arbiter

ReCoBus: Shared Read

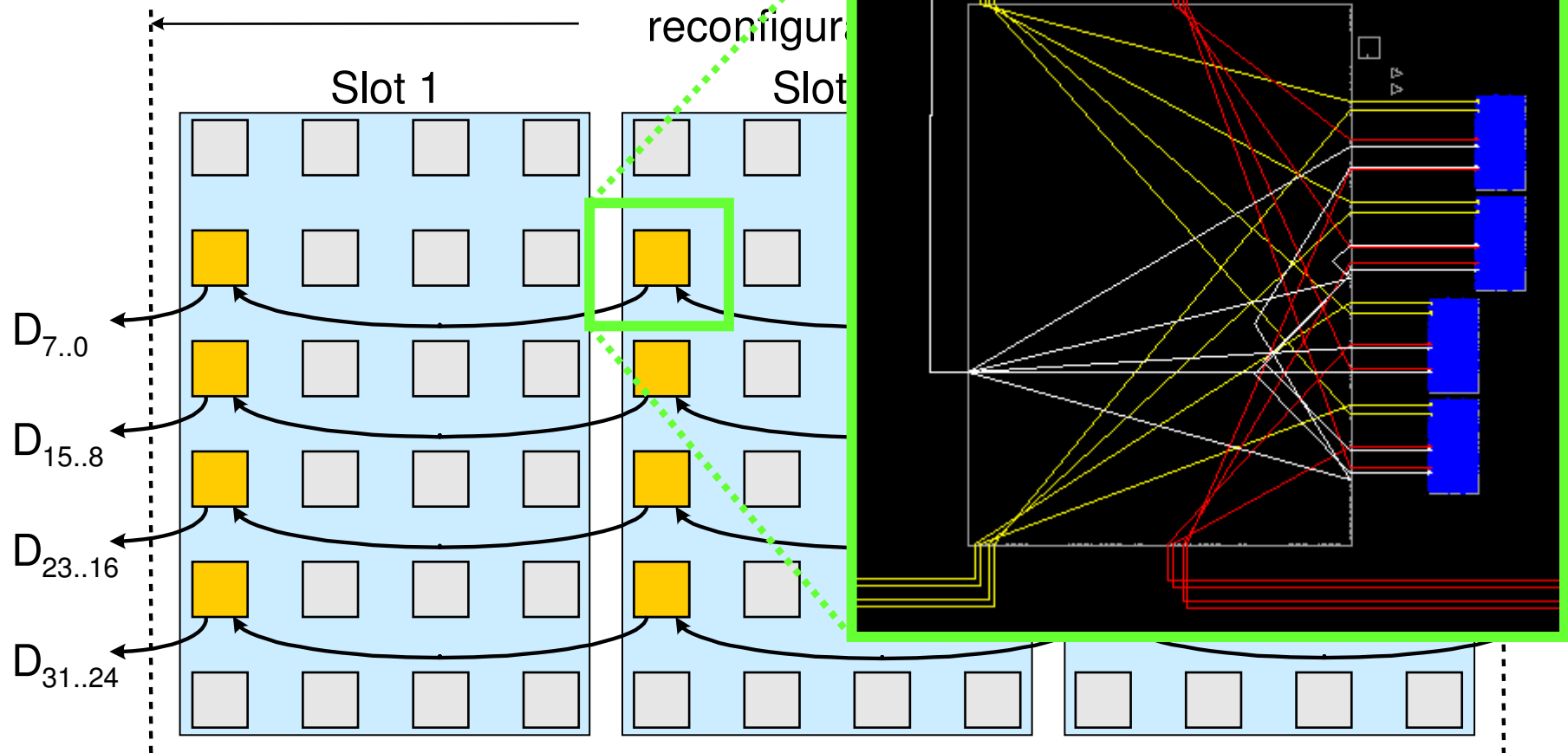
Shared read signals for connecting one selected module with the static system:



- Homogeneous (=identical) logic and routing footprint inside each resource slot
- Free module placement
- Deep combinatory path (slow)
- Massive resource overhead (has to be replicated for each bit signal)
- Only suitable for a coarse-grained placement grid
→ internal fragmentation

ReCoBus: Interleaving

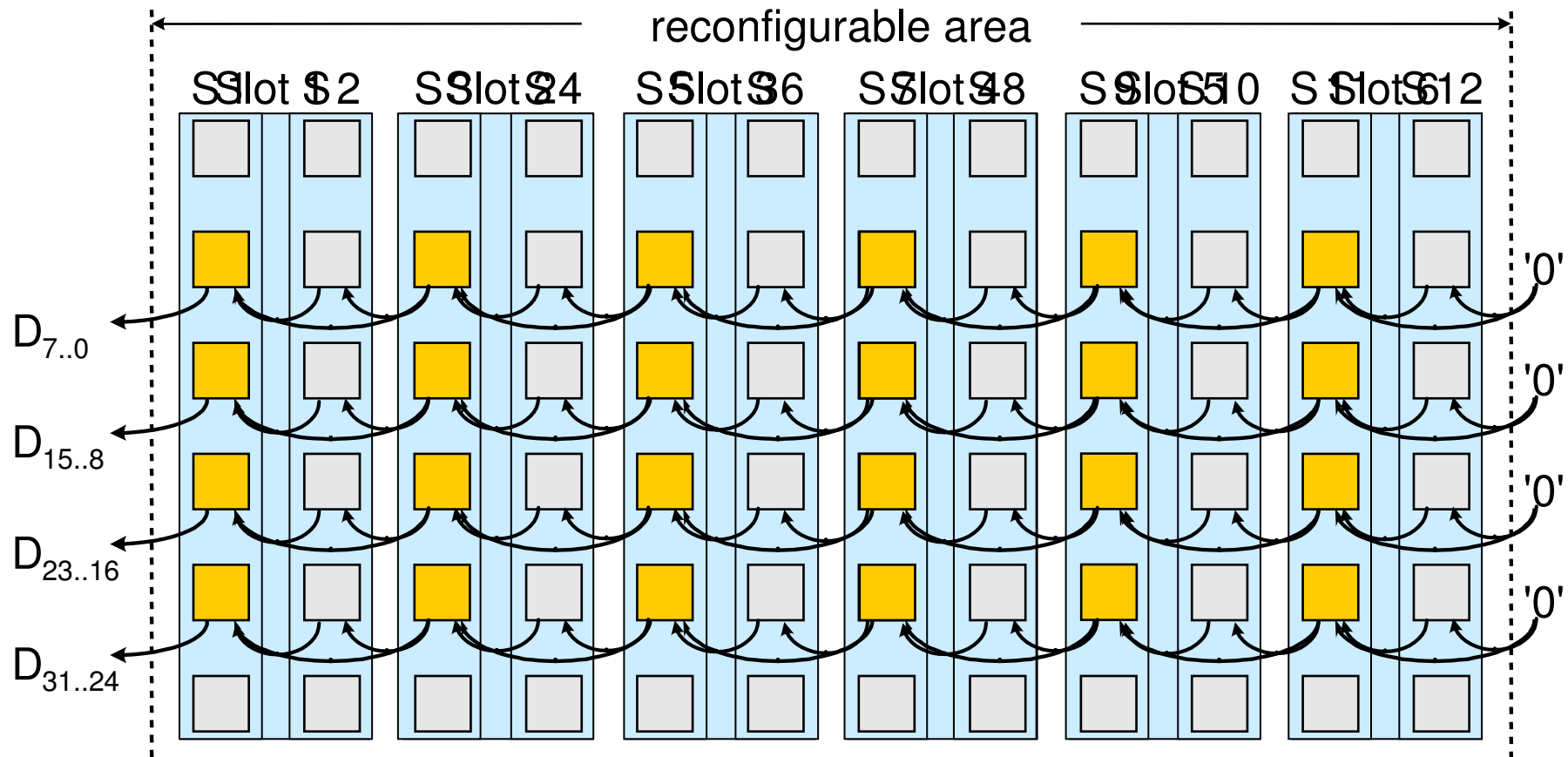
- Problem: the structure of a distributed read multiplexer chain is unlikely for very fine-grained resource



- Logic overhead: $4/24 = 17\%$

ReCoBus: Interleaving

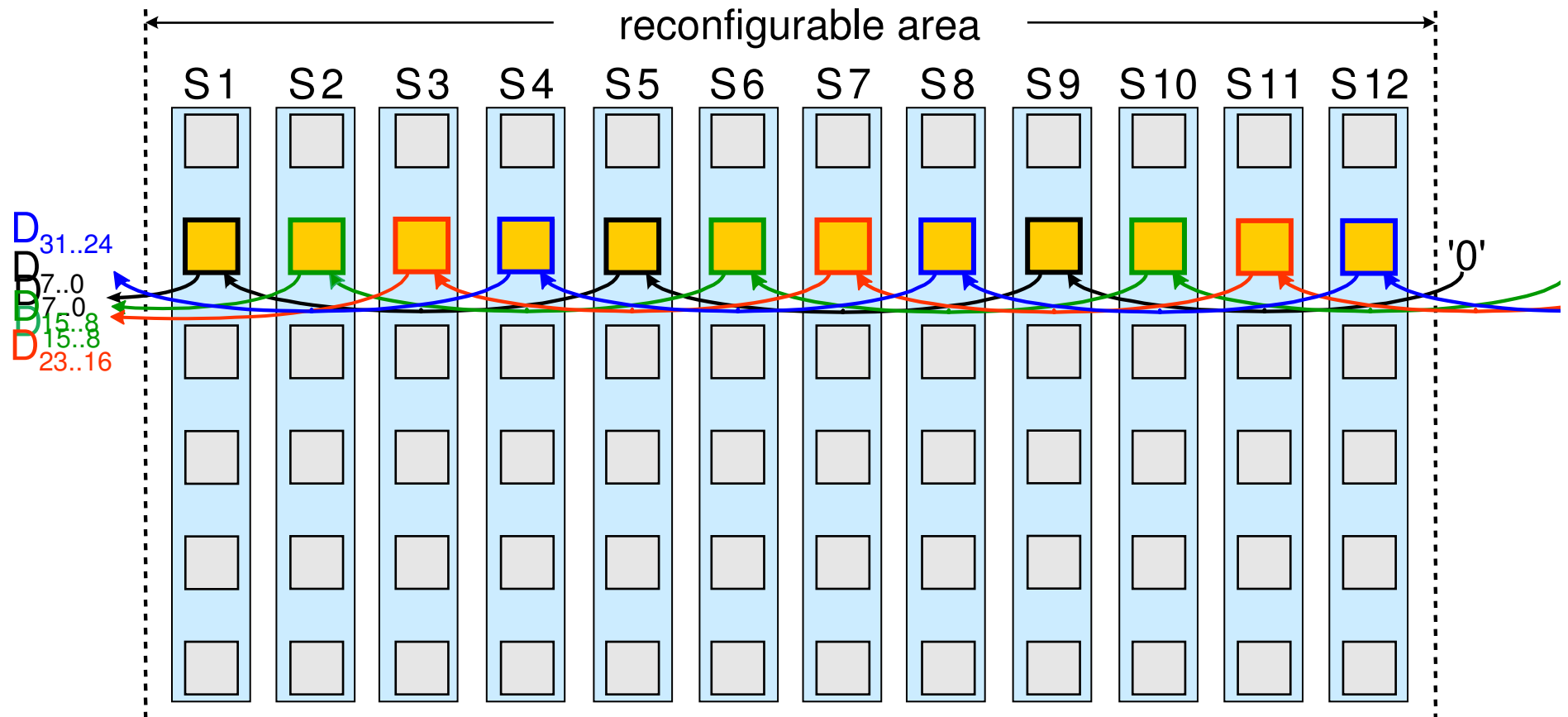
- Problem: the structure of a distributed read multiplexer chain is unlikely for very fine-grained resource slot layouts:



- Logic overhead: $4/6 = 66\%$; **very high latency!**

ReCoBus: Interleaving

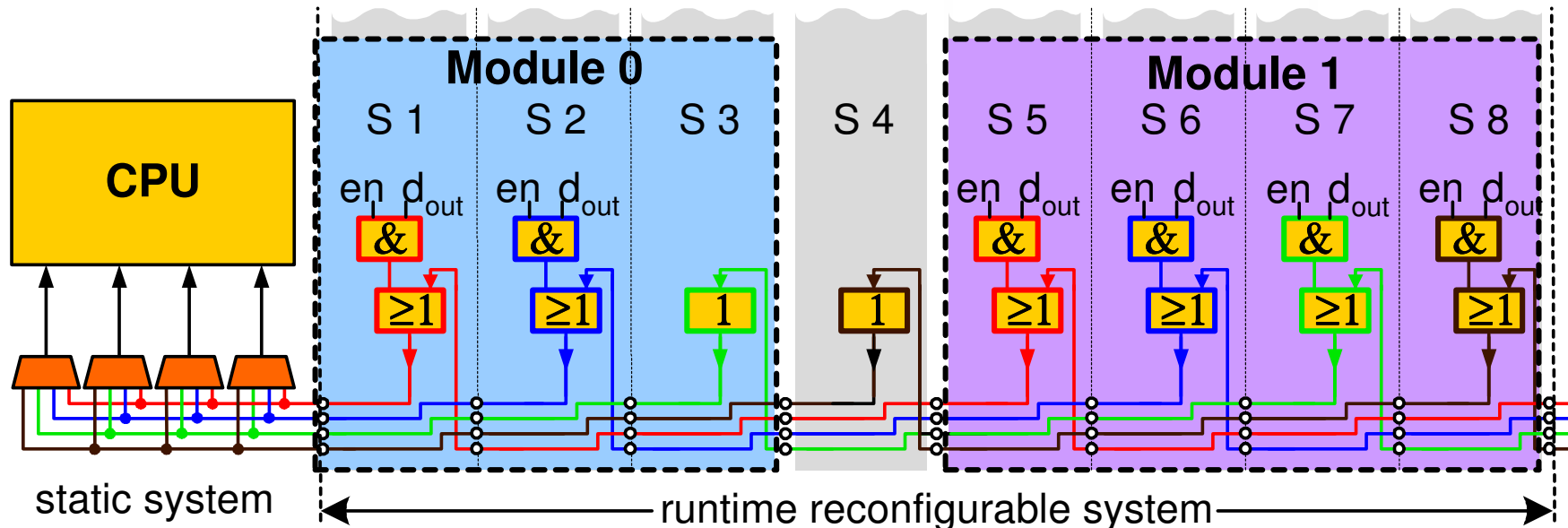
- Solution: multiple interleaved read multiplexer chains



- **Low logic overhead, low latency and fine granularity!**

ReCoBus: Signal Alignment (1D)

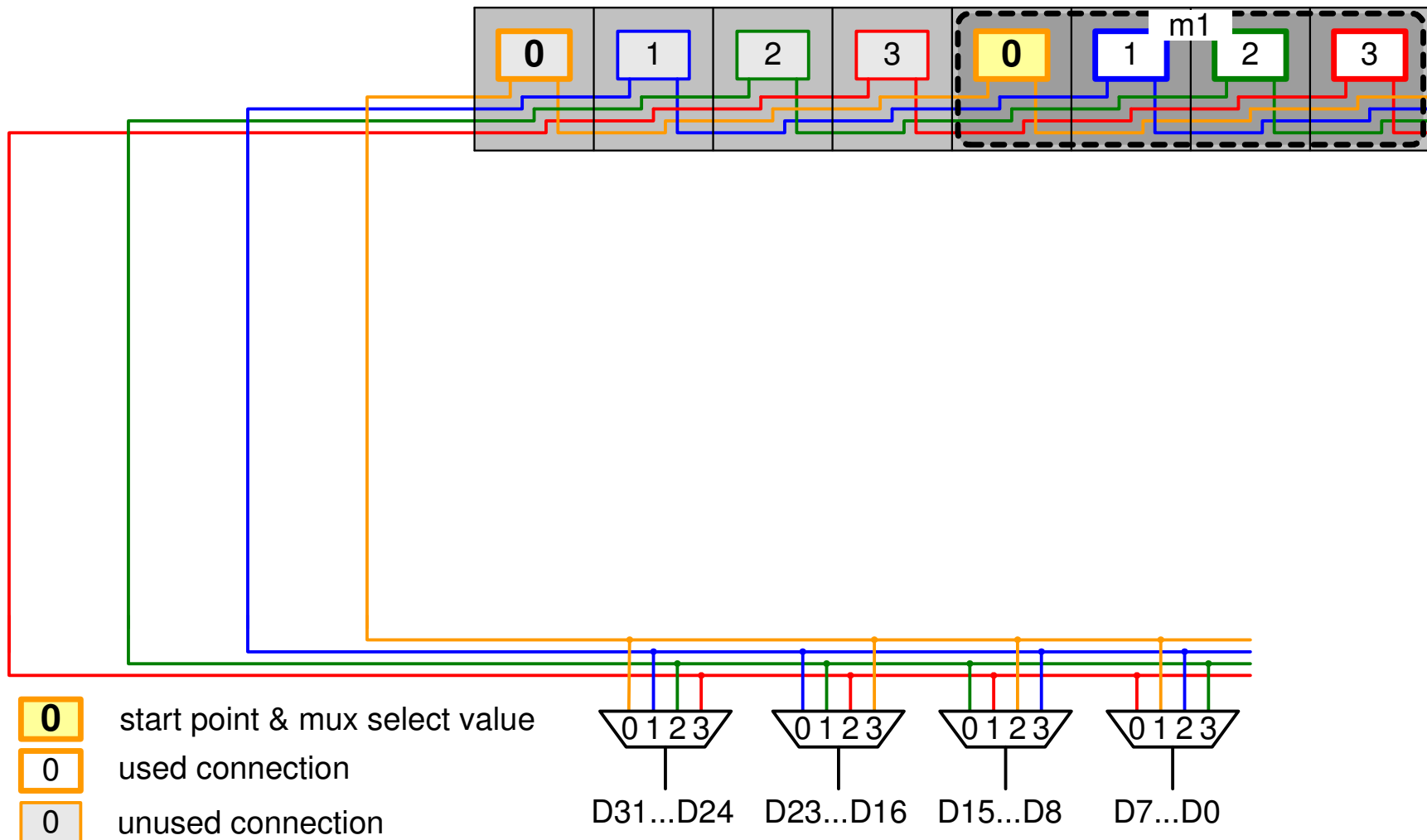
- Example system:



- Alignment-multiplexer allow free module placement
- Interface grows together with the module complexity (size)
For example: a small UART might be connected using an 8-bit data bus and a more complex Ethernet adapter with 32-bit
- The first LUT function of each chain (here rightmost) must be changed to an AND gate or an external source is needed

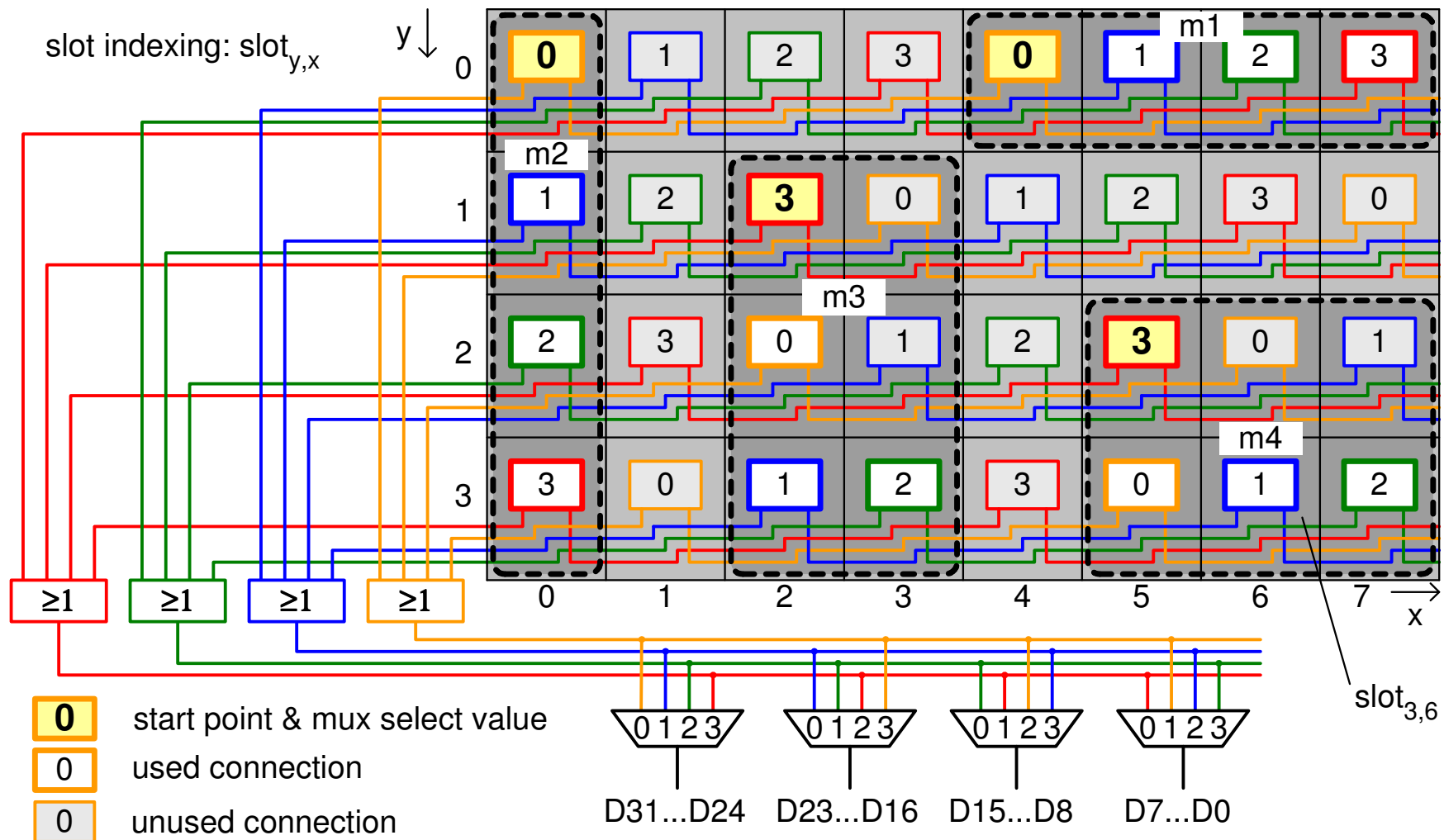
ReCoBus: Signal Alignment (1D)

- Assuming an 8-bit interface per slot, it takes at least four consecutive slots to provide the full interface size



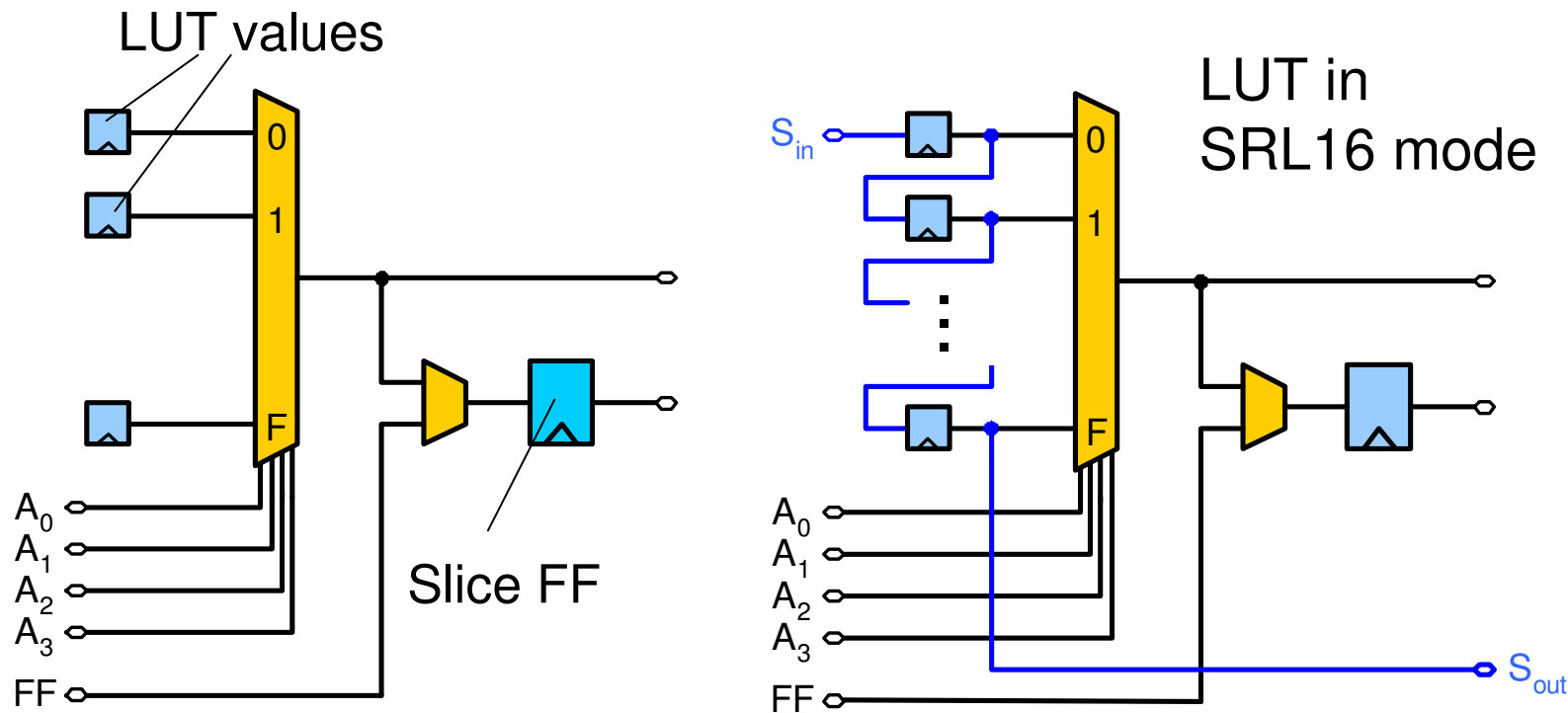
ReCoBus: Signal Alignment (2D)

- The signal interleaving scheme can be extended to implement buses allowing to integrate modules in a 2D grid style.



ReCoBus: Dedicated Write Signals

- LUTs can be used to decode an address within the bus (the table contains then a one-hot value, e.g. for addr. 0xA)

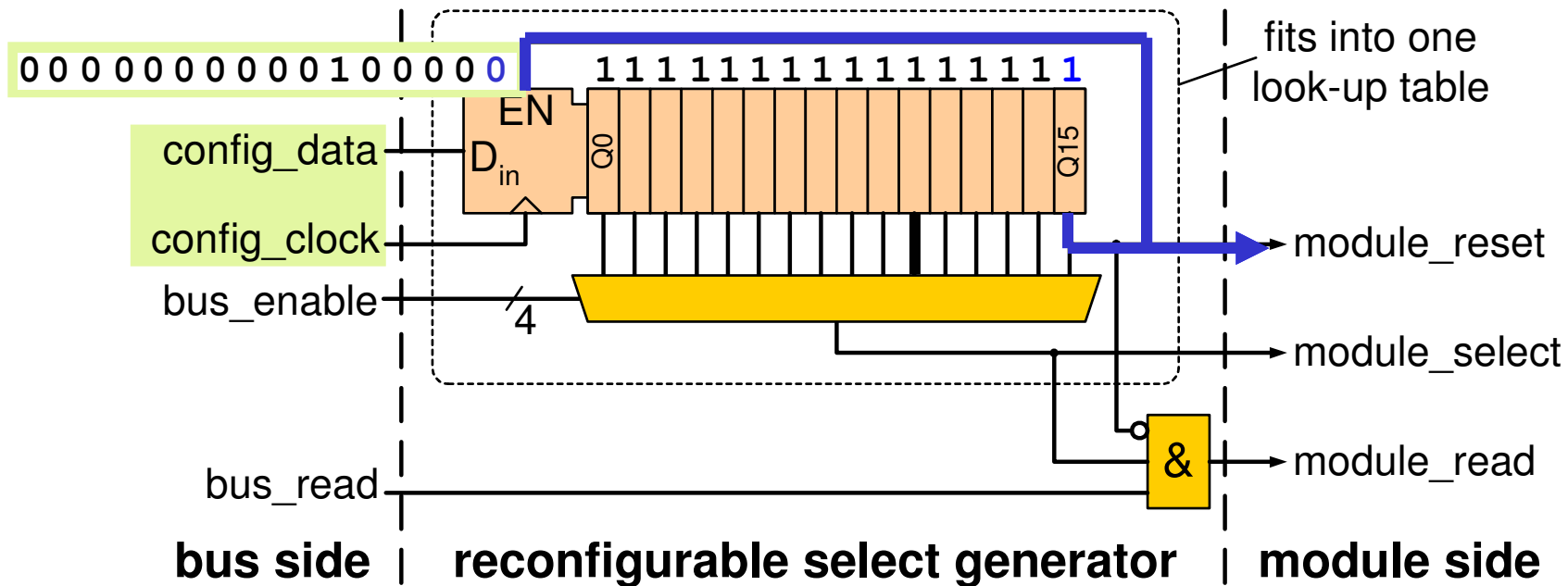


	value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
A	1
B	0
C	0
D	0
E	0
F	0

- For setting an address, LUT values can be exchanged:
 - Using the configuration port,
 - Accessing the table with the user logic:
SRL16 shift register primitive or distributed memory

ReCoBus: Dedicated Write Signals

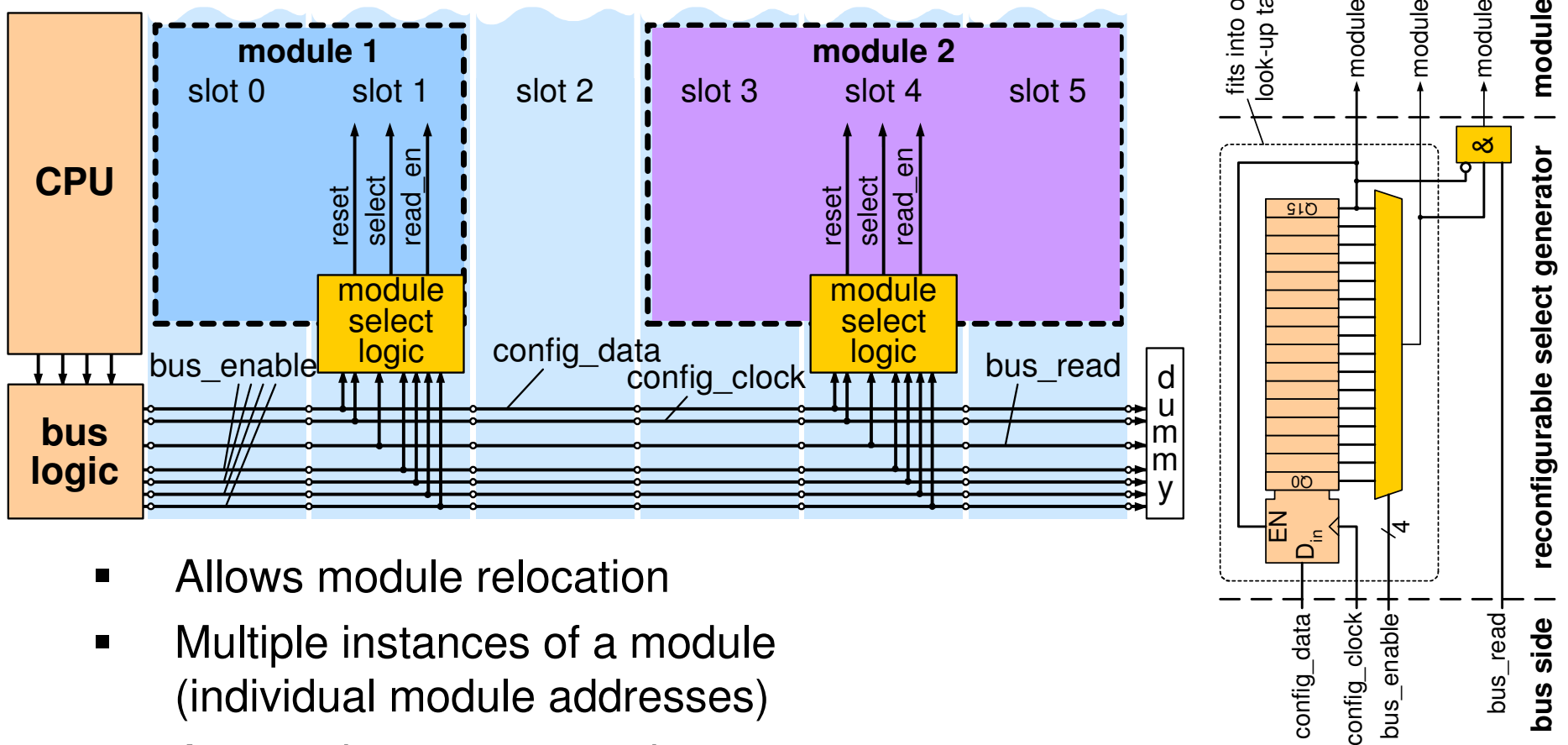
- Architecture: uniform distributed address comparator inside the bus (implemented by SRL16 shift register primitives)



- Two-stage reconfiguration:
 1. FPGA: initialize the shift register with 0xFFFF
 2. Logic: configure address comparator and activate module

ReCoBus: Dedicated Write Signals

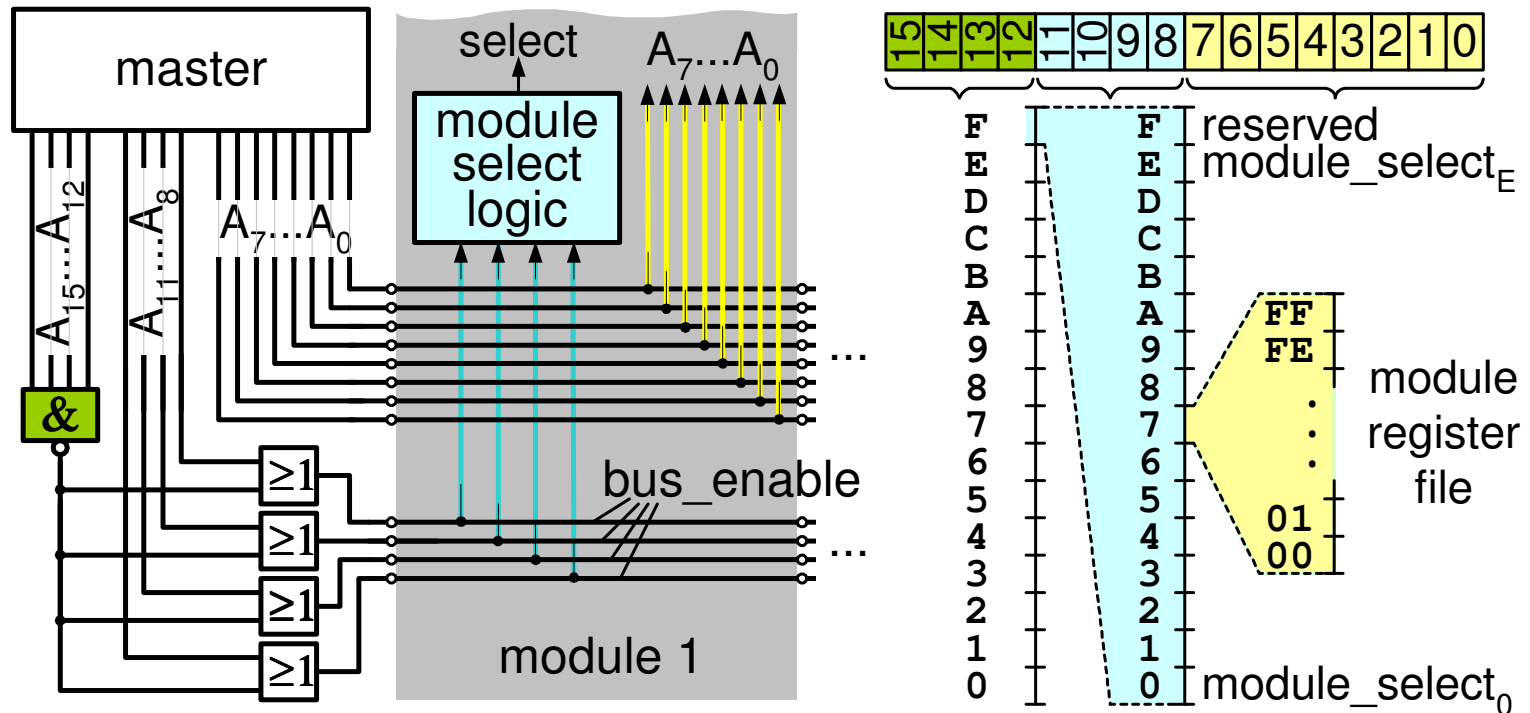
- Arrangement of the address comparators



- Allows module relocation
- Multiple instances of a module (individual module addresses)
- Automatic reset generation
- No interference by the reconfiguration process (Hot-Plug)
- Extra register file look-up for alignment multiplexer control

ReCoBus: Dedicated Write Signals

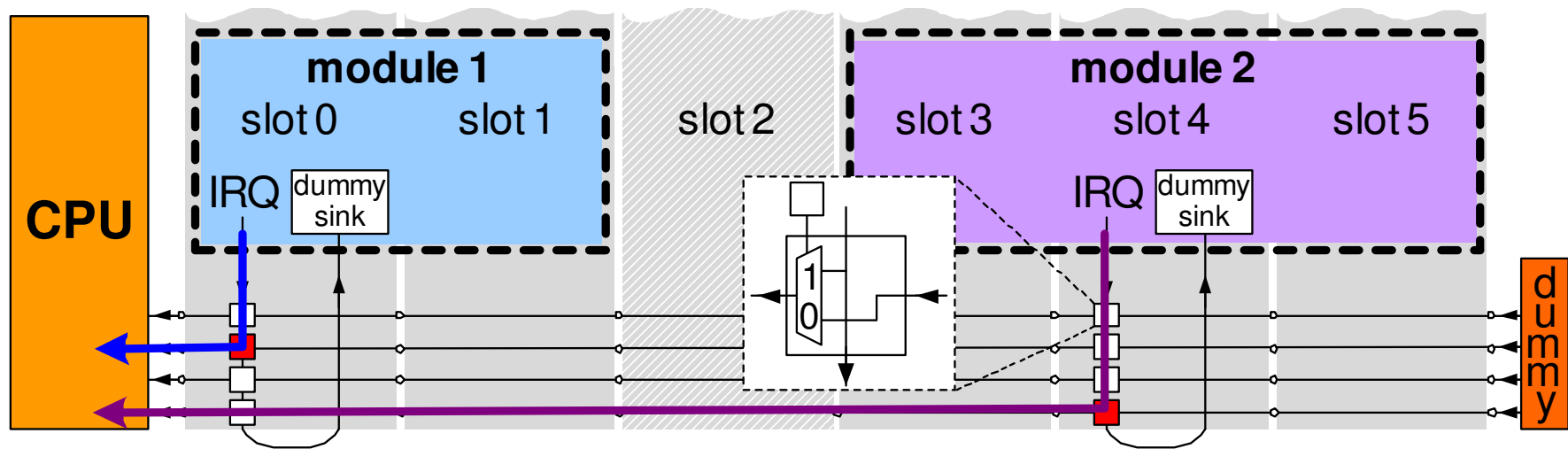
- Assuming an 8-bit interface pro slot, it takes at least four consecutive slots to provide the full interface size



- Address mapping: the whole ReCoBus subsystem appears like one module in the address space of the system
- Up to 15 modules can be addressed (one encoding (0xF--) is used for the case that no module is selected)
- Wildcard addressing for multi cast operation (wired OR on read)

ReCoBus: Dedicated Read Signals

- Dedicated master read signals (`interrupt`)



- Idea: set connection within a module to an internal homogeneously routed interrupt wire by bitstream manipulation
- The number of internal interrupt lines scales with the number of modules (allows many tiny slots)
- Crosspoints are directly implemented in the FPGA routing fabric (no extra logic required)
- In practice: internal wire sharing for interrupt and bus arbitration (also: signal interleaving and masking in the static system)

ReCoBus Properties

- Direct connection of a module to the bus
- Compatible to all established standards (AMBA, PLB, ...)
- Module relocation & flexible module placement
- Variable module sizes
- Multiple instances of the same module
- Very low logic overhead
- Allows high speed / high throughput
- Hot-swap module exchange: The reconfiguration process is completely transparent for all bus transactions.

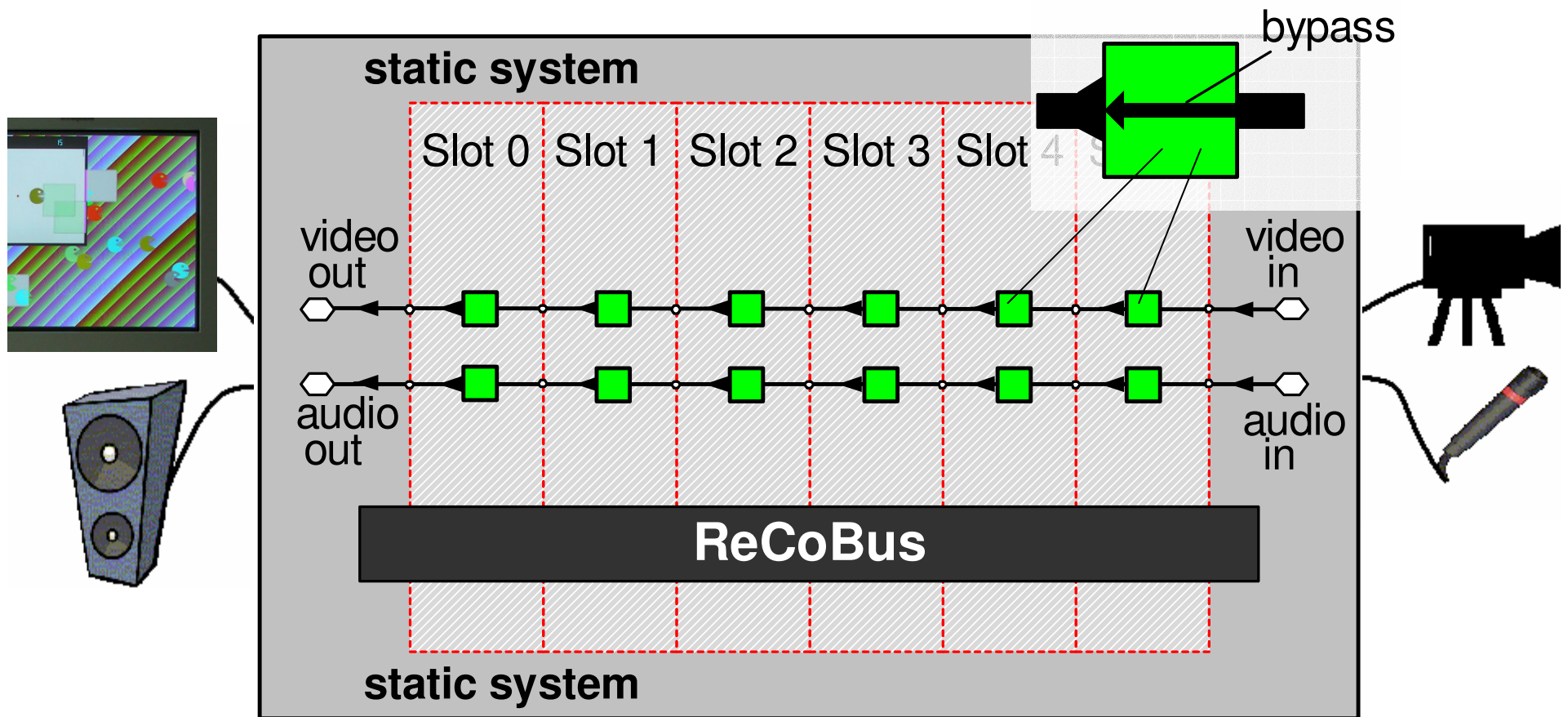
I/O-Bars

OK - We have a suitable Bus.

What about dedicated links or I/O?

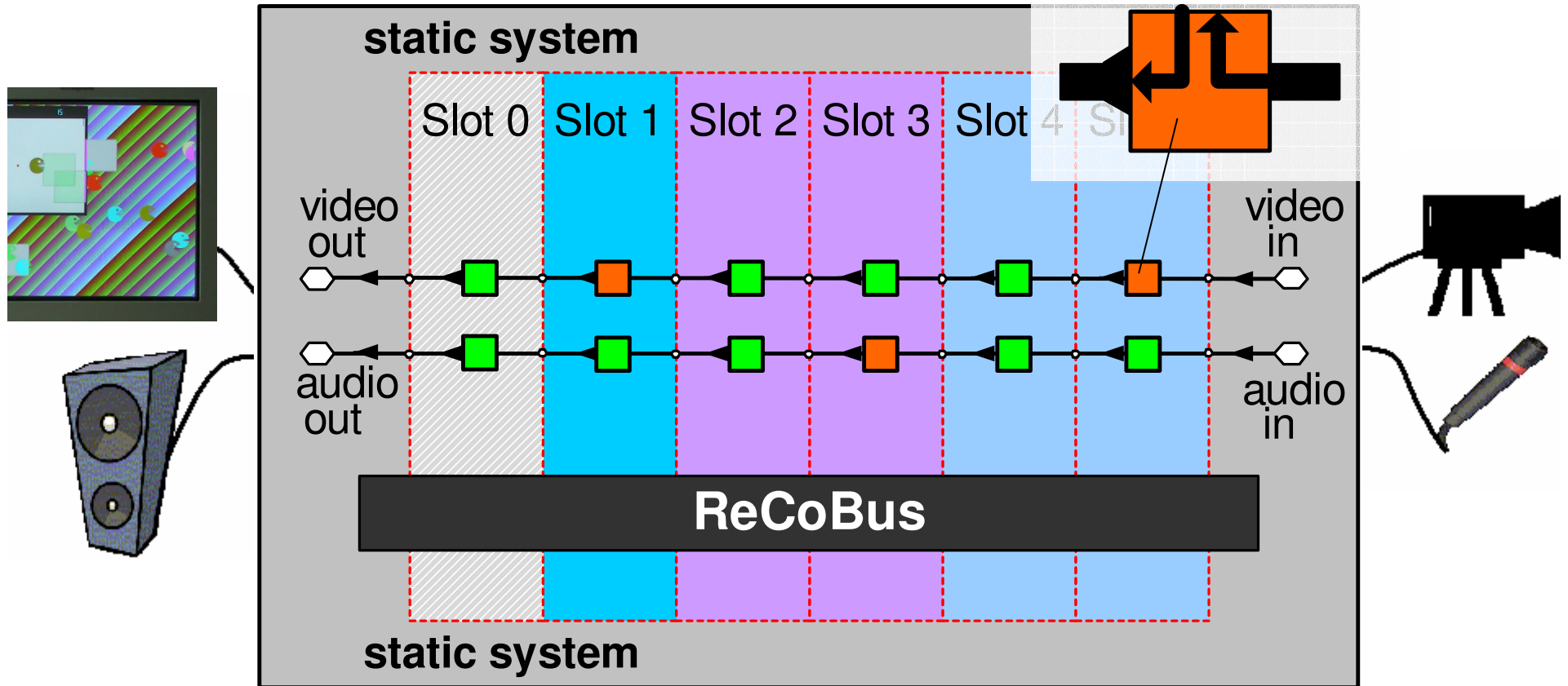
I/O-Bars for Point-to-Point Links

- Horizontal routing track within the reconfigurable area
- Connections are set by modifying switch matrices
- One bar per interface requirement (e.g., video, audio)



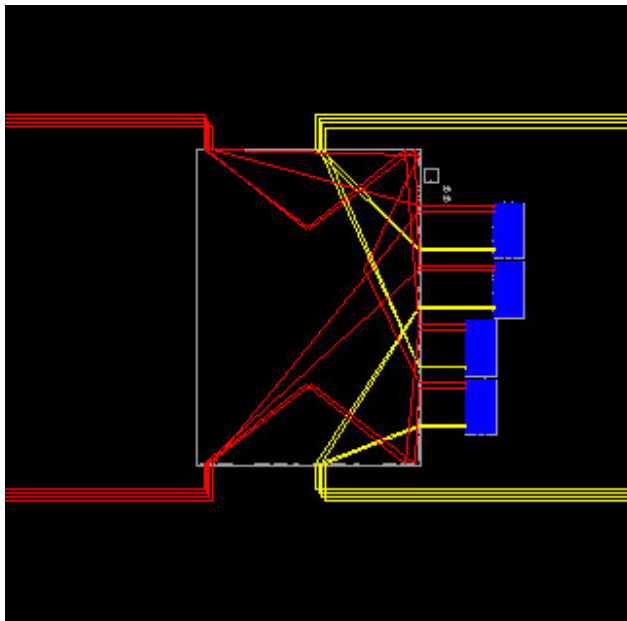
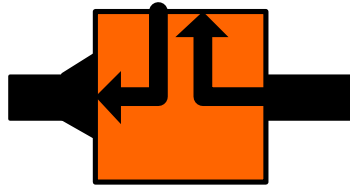
I/O-Bars for Point-to-Point Links

- Read-modify-write connection
- Ideal for data streaming

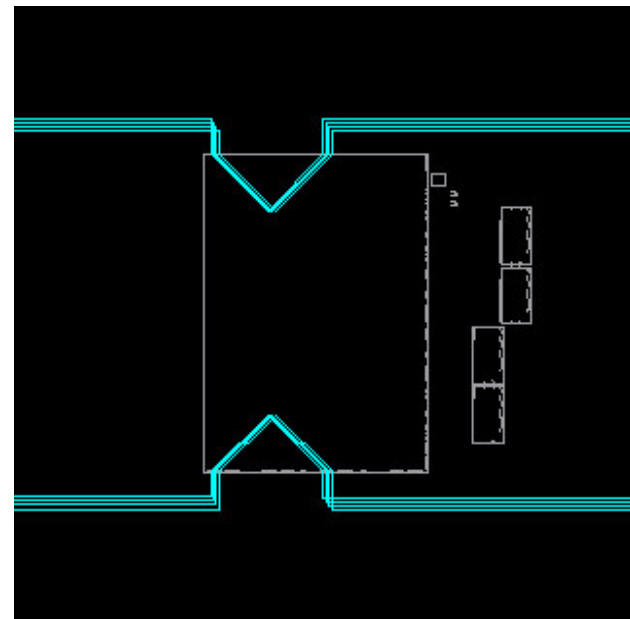
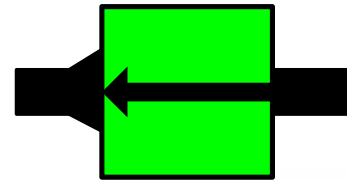


I/O-Bars for Point-to-Point Links

- I/O bar implementation



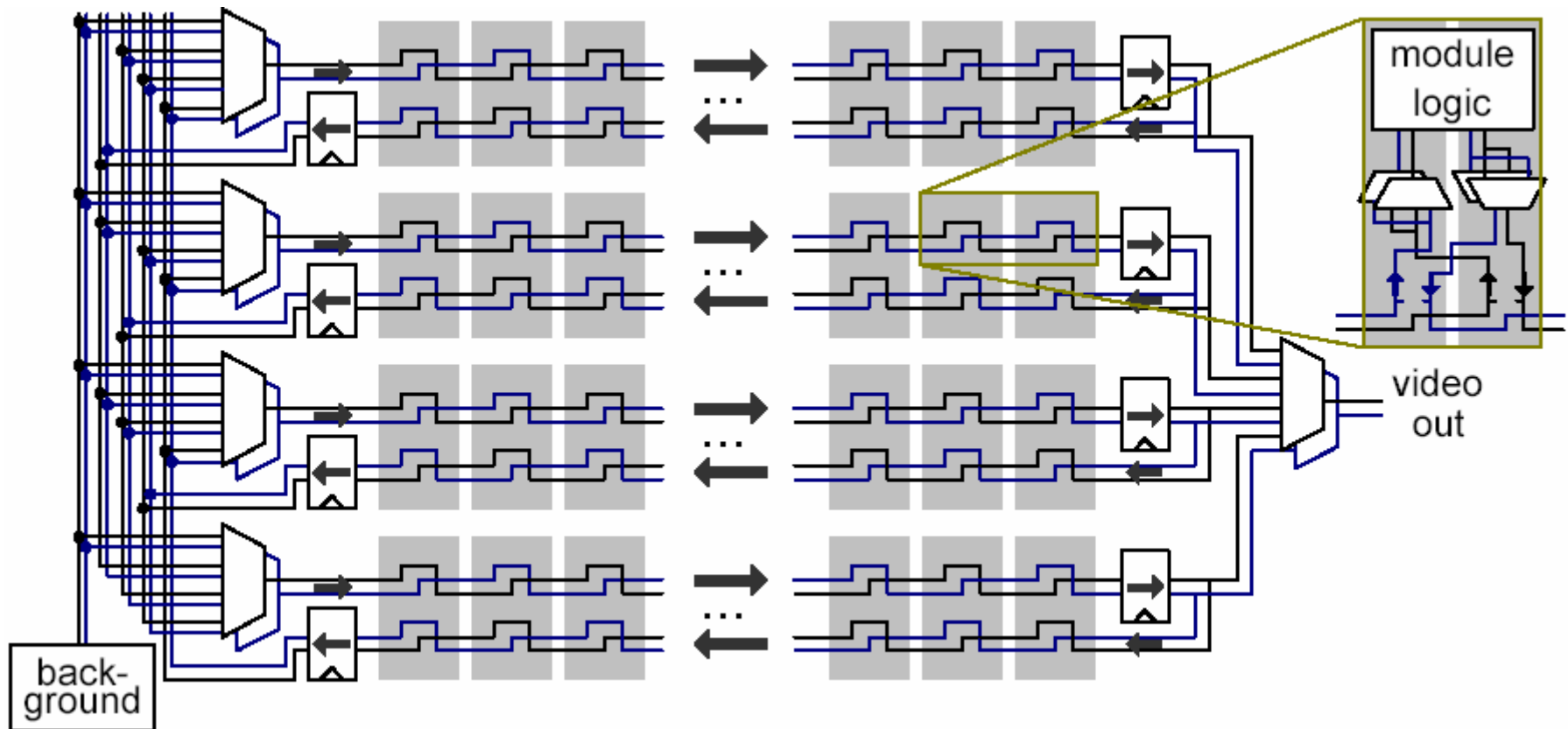
— Incoming signals
— Outgoing signals



— Route through signals

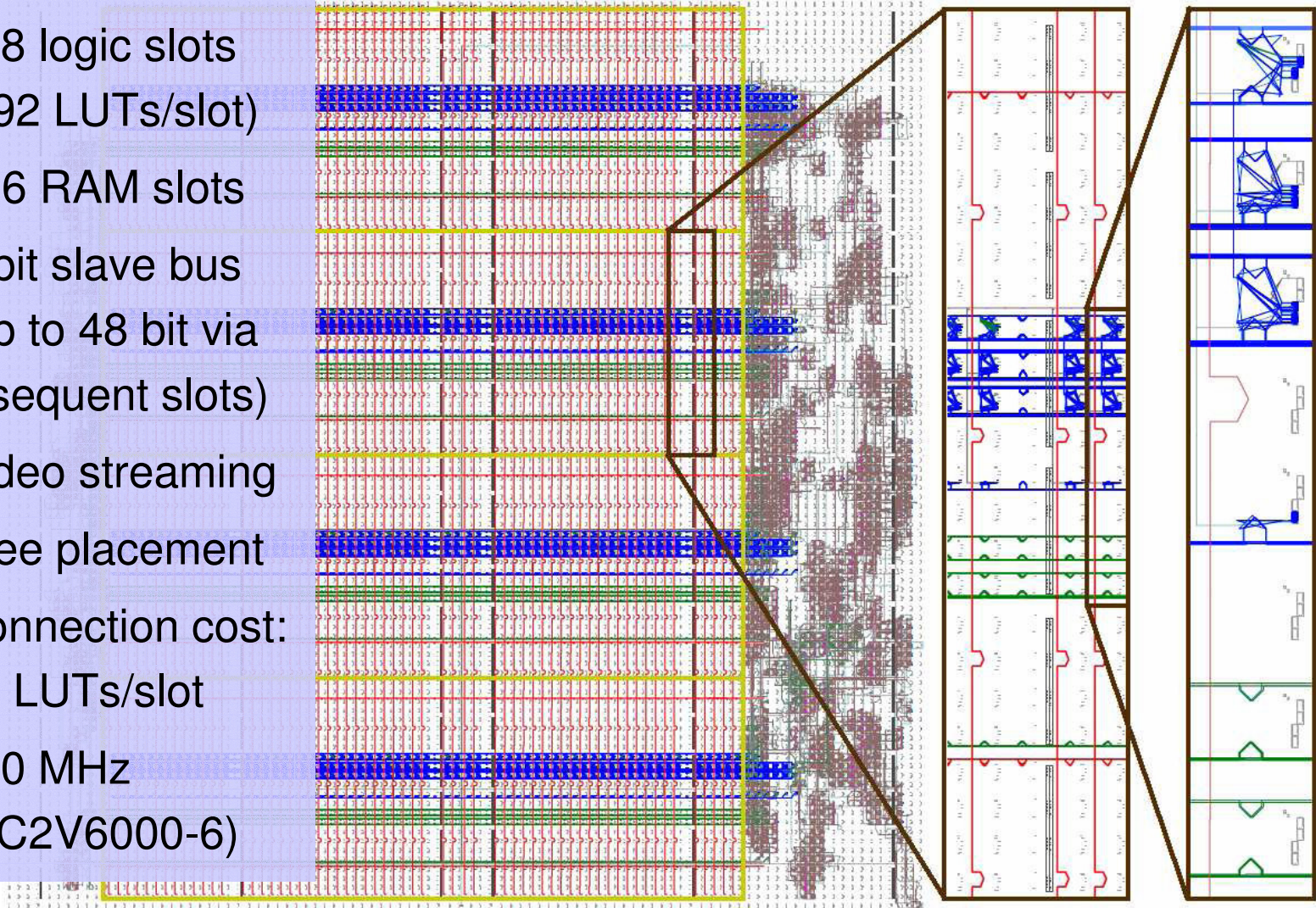
I/O-Bars for Point-to-Point Links

- I/O-Bar implementation for 2D
- Vertical routing is accomplished in the static part
- Can be used with interleaving for decreasing latency (requires signal alignment in each module)



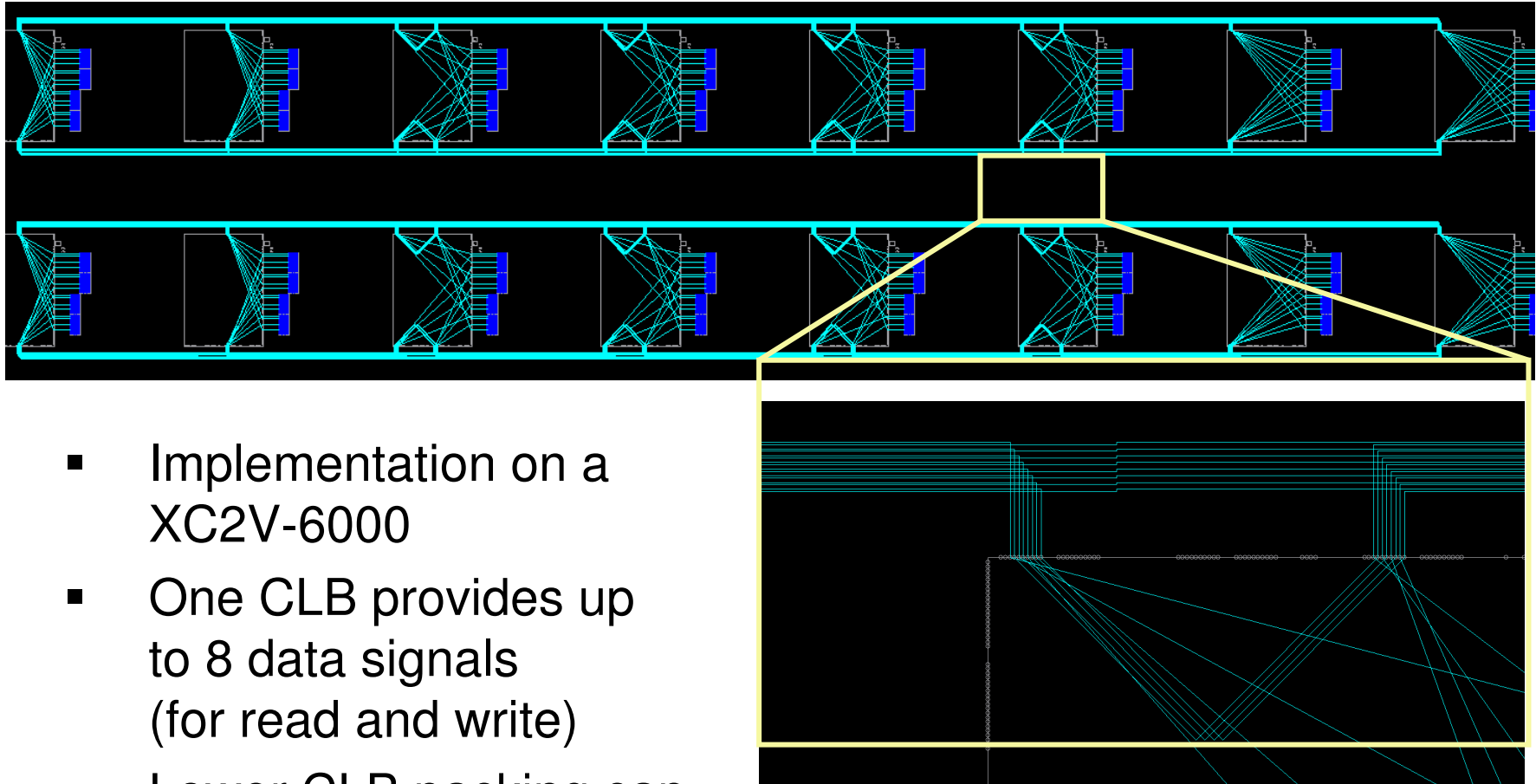
Demo System

- 248 logic slots
(192 LUTs/slot)
- +16 RAM slots
- 8-bit slave bus
(up to 48 bit via
6 sequent slots)
- Video streaming
- Free placement
- Connection cost:
14 LUTs/slot
- 100 MHz
(XC2V6000-6)



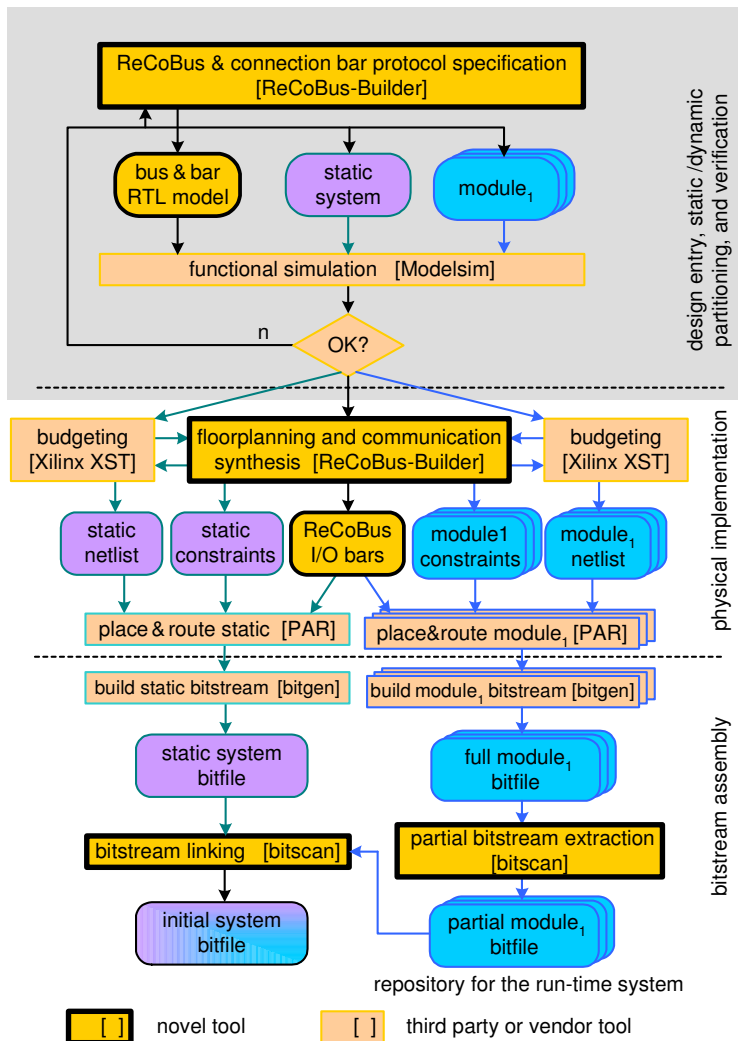
Demo System

- Regular structured ReCoBus macro (a macro contains logic and routing and is instantiated like any other VHDL module)

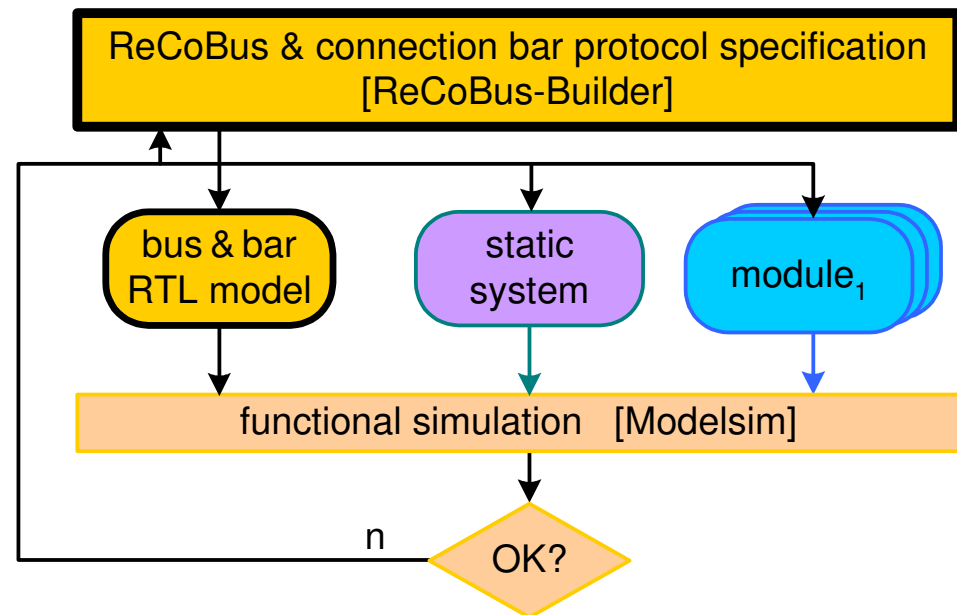


- Implementation on a XC2V-6000
- One CLB provides up to 8 data signals (for read and write)
- Lower CLB packing can improve routing (congestion around the connecting resources)

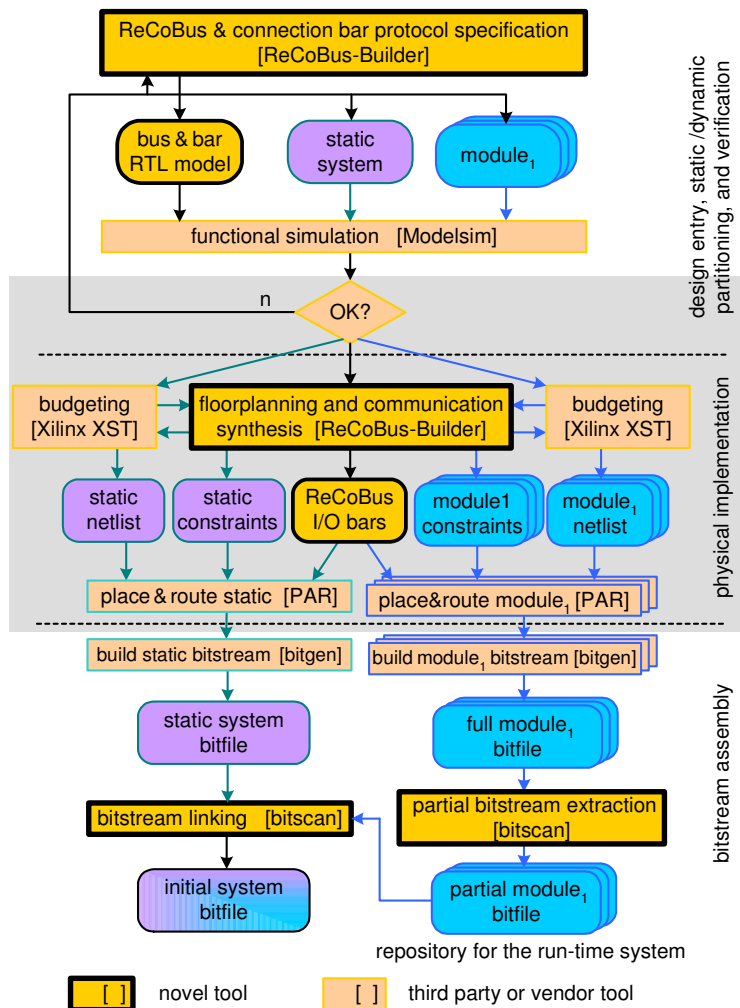
Design Flow



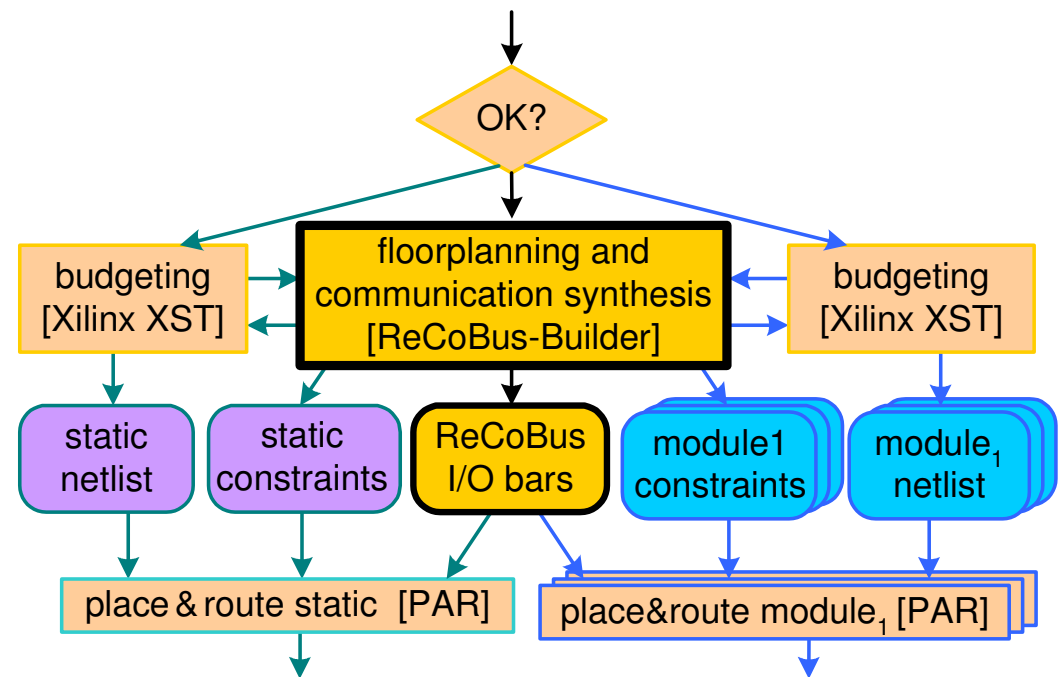
Design Entry, Static/Dynamic Partitioning, and Verification



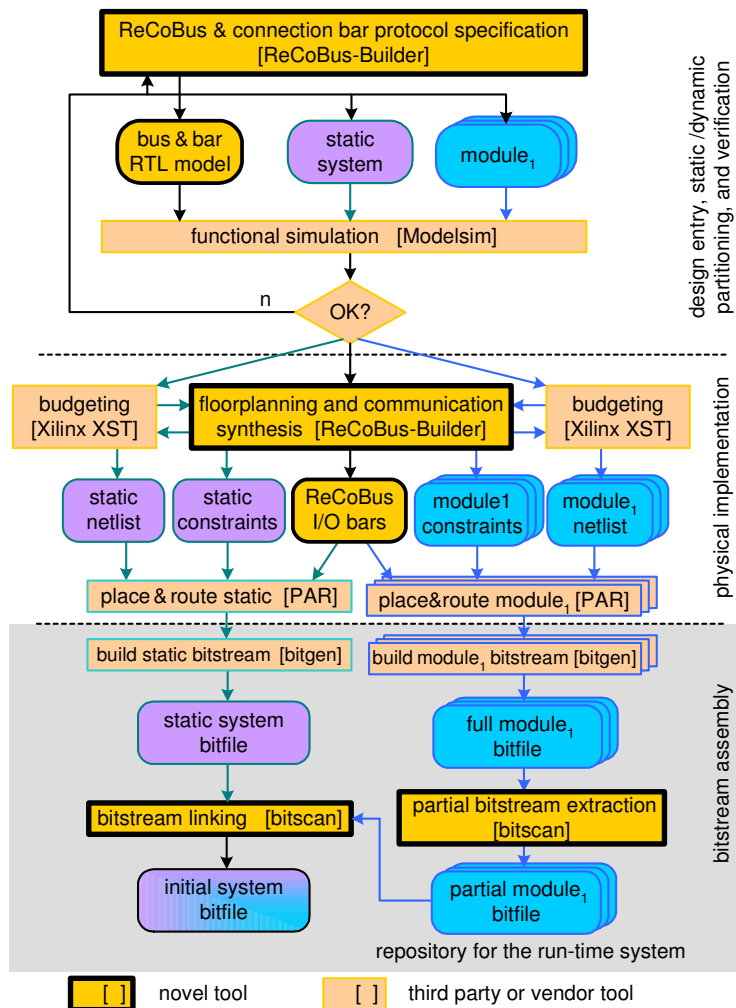
Design Flow



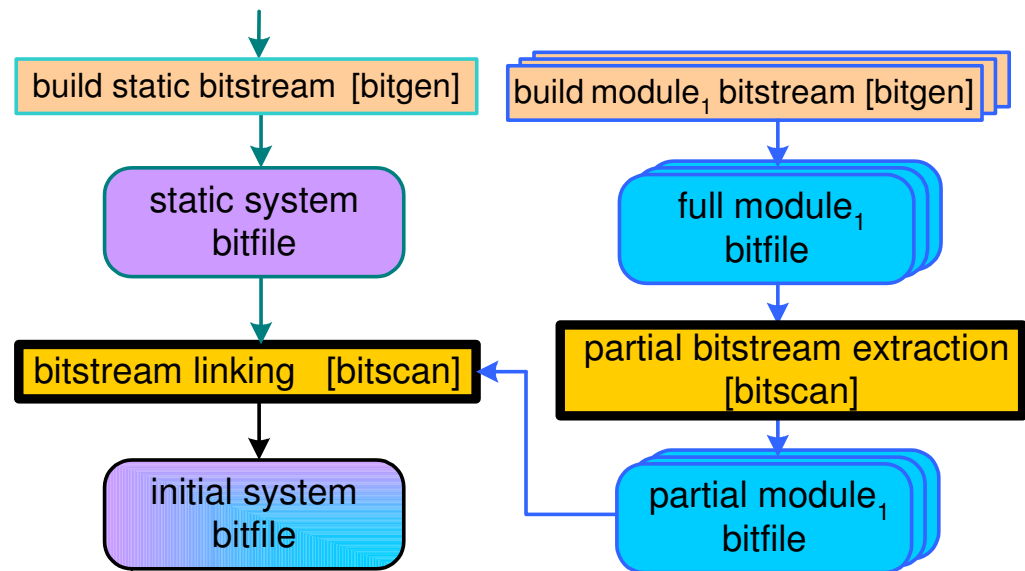
Physical Implementation



Design Flow

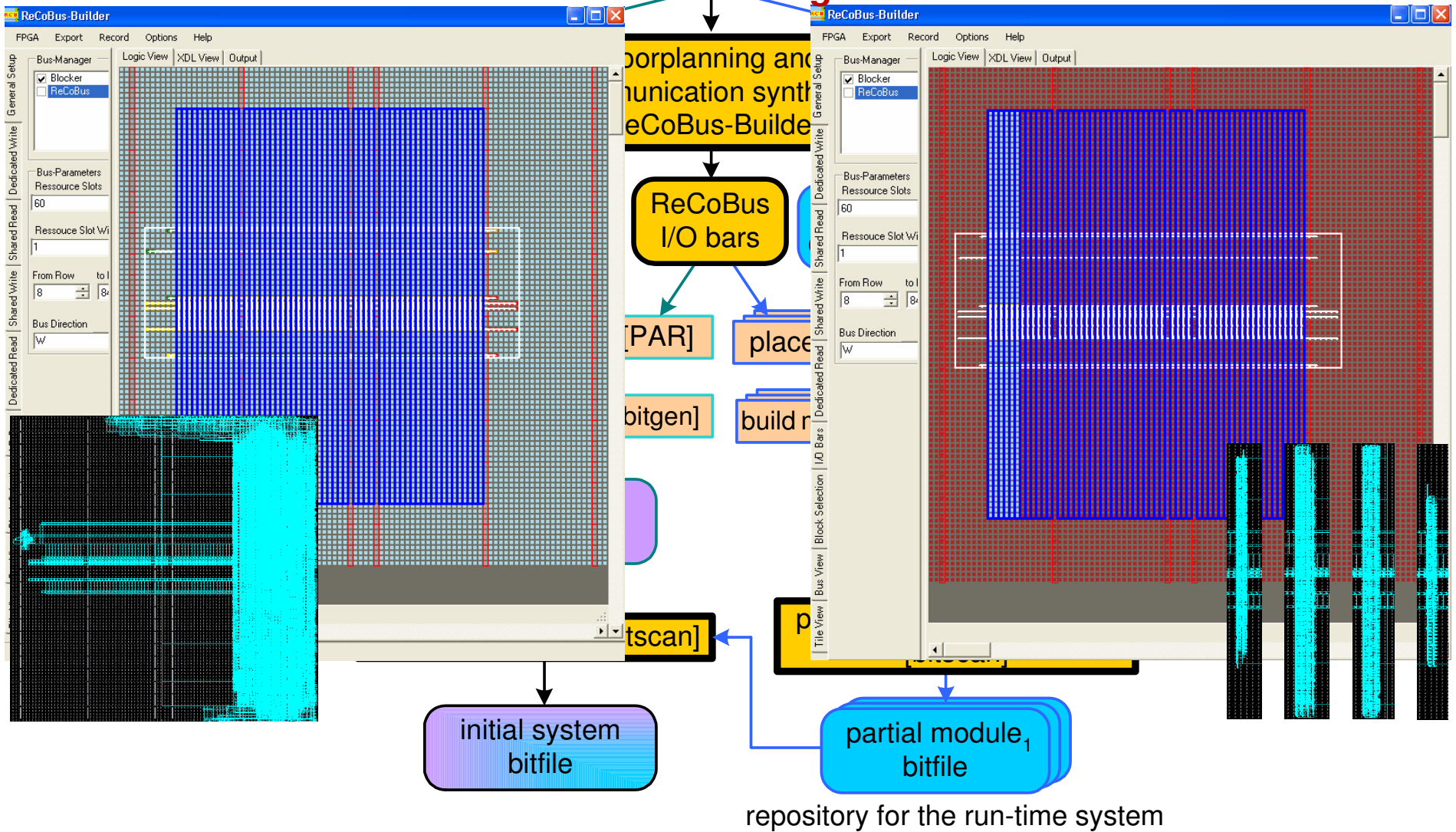


Bitstream Assembly



Design Flow

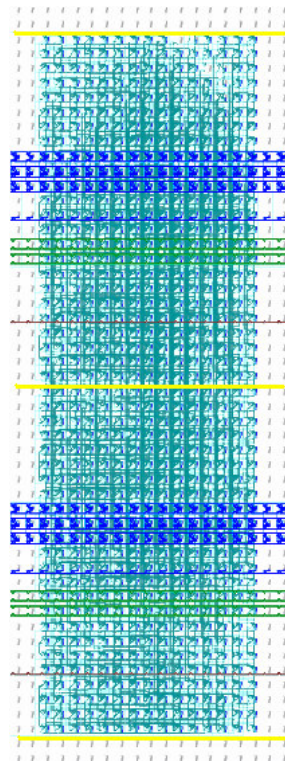
Tested Design



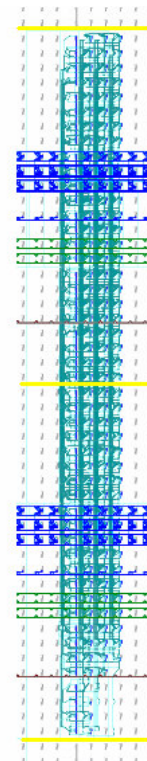
```
bitlink module.bit X Y \
static.bit initial.bit
```


Design Flow

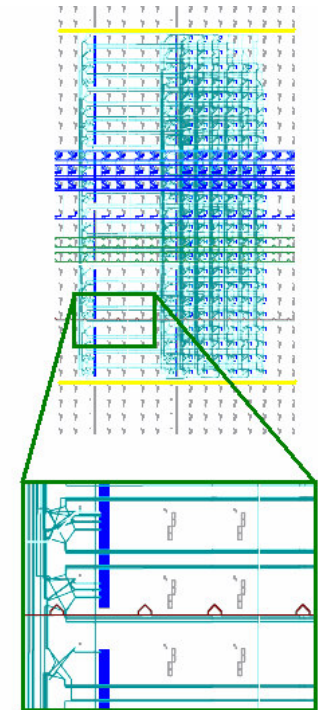
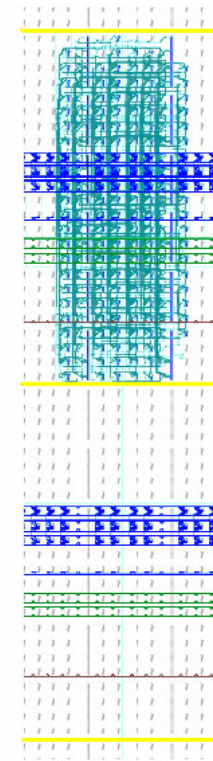
- Modules might be implemented using different shapes/resources (design alternatives)
- Goal: higher utilization
- Interesting for component based system design (no place and route)
- Simplified system integration based on standardized interfaces
- Enhanced IP-reuse



logic only
30 slots

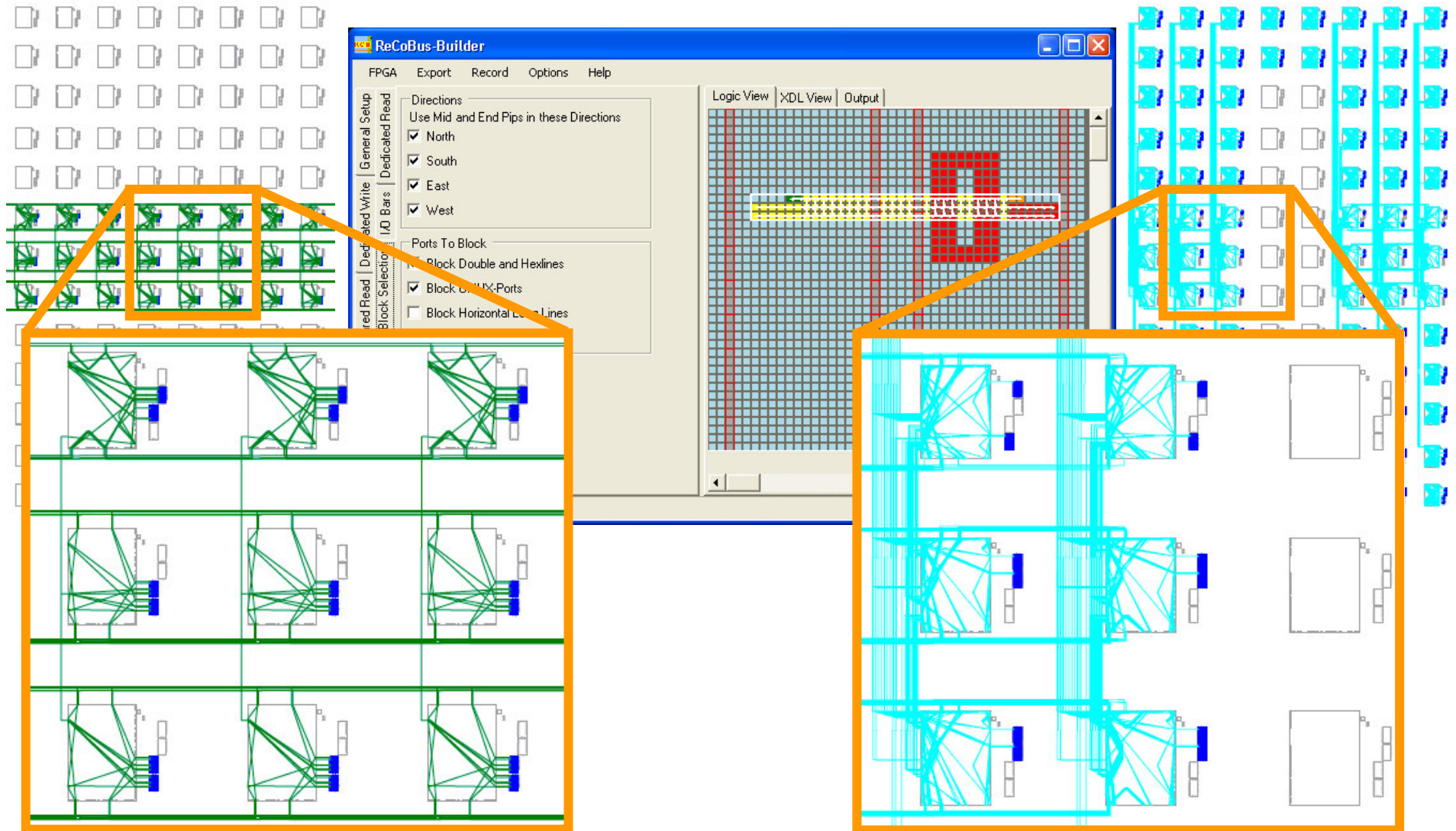


2 multiplier,
6 logic slots



2 multiplier,
6 logic slots
(includes gap)

Design Flow: Blocking



Design Flow: Xilinx PlanAhead

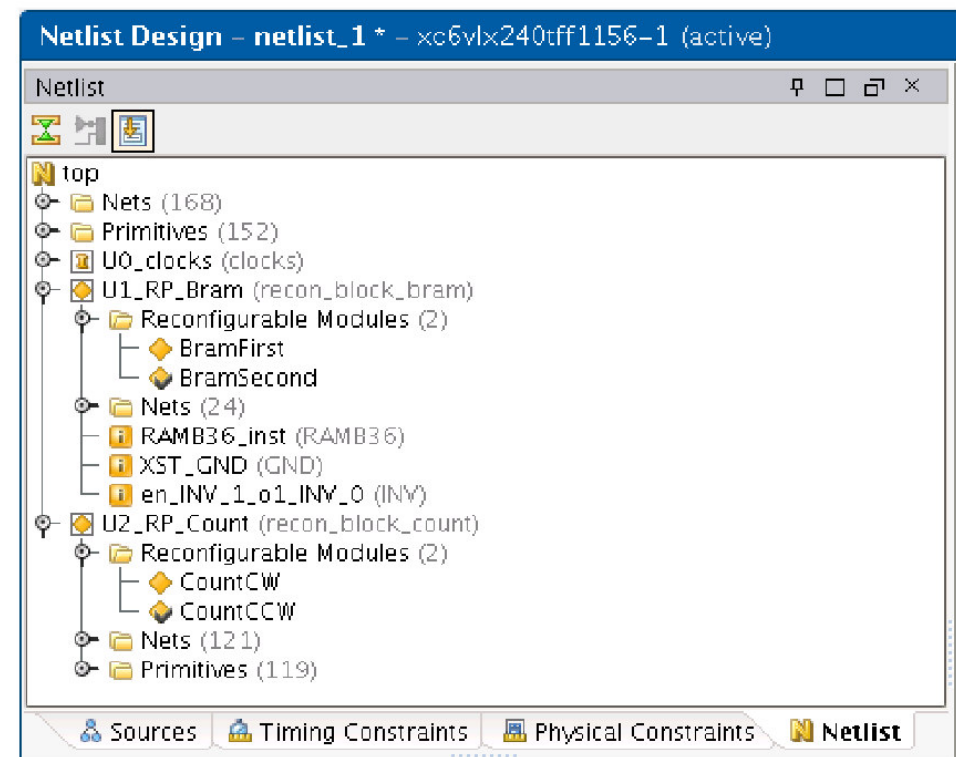
- New advanced GUI for the complete FPGA design flow
- Project management
- Floorplanning
- Critical path analysis (timing)
- Implementation viewer
- Integration of the vendor specific partial flow



Source: Xilinx

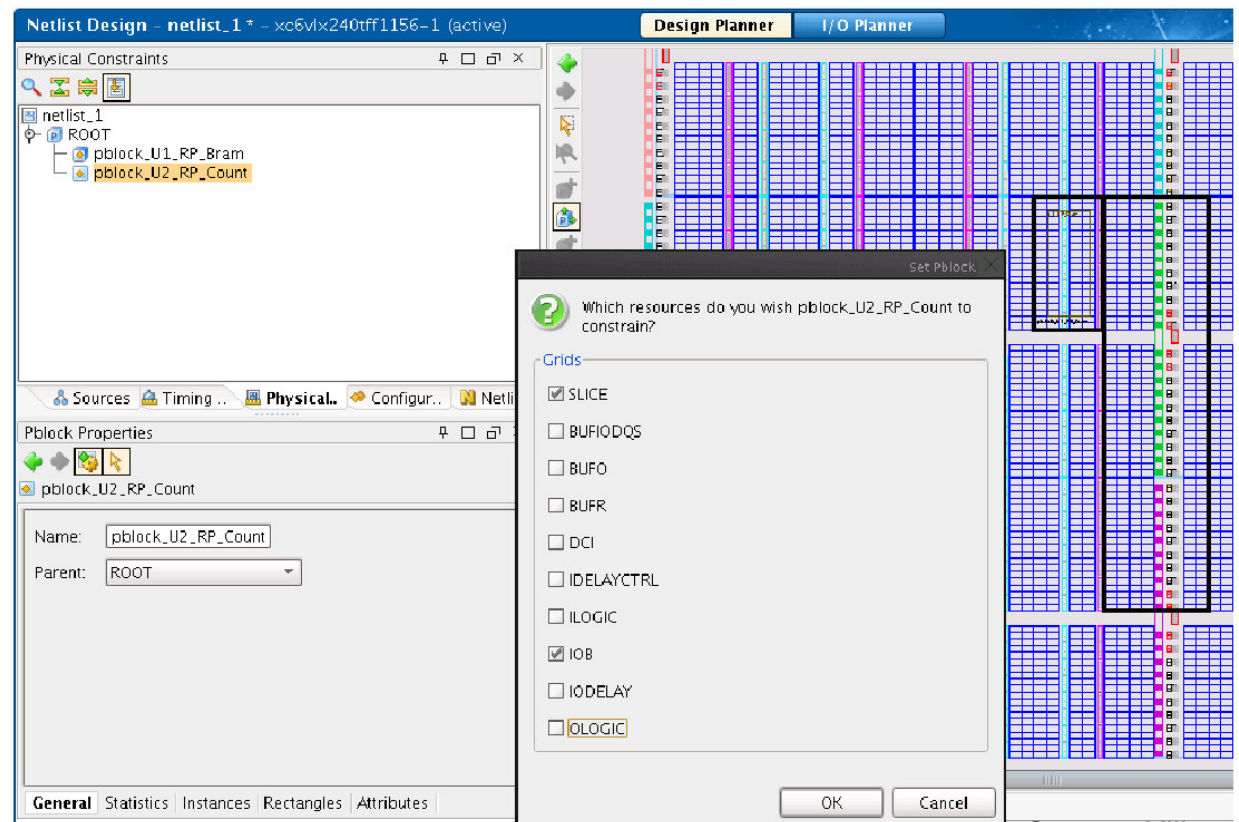
Design Flow: Xilinx PlanAhead

- **1. Step:** Synthesis of all partial and static modules in individual netlists
(Static netlist has black boxes for the modules)
- **2. Step:** Creation of a new PlanAhead project
- **3. Step:** Creation of Reconfigurable Partitions
 - A reconfigurable partition (RP) consists of several reconfigurable modules (RM)
 - Assign a partial netlist to each RM
 - A RM can also be a black box (empty module)



Design Flow: Xilinx PlanAhead

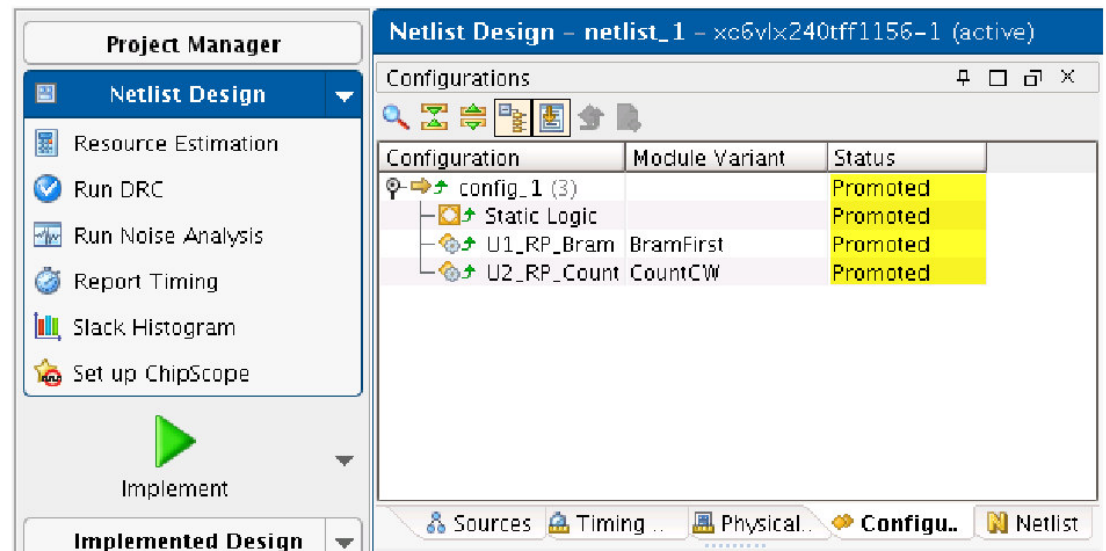
- **4. Step:** Floor planning of the reconfigurable partitions
 - Create Area Groups
 - PlanAhead automatically creates the communication ports for the reconfigurable partition
- Port proxy logic:
 - LUT1 (anchor required for physical implementation)
- PlanAhead automatically creates the user constraints file (UCF) with the bounding box definitions of the RPs



Source: Xilinx

Design Flow: Xilinx PlanAhead

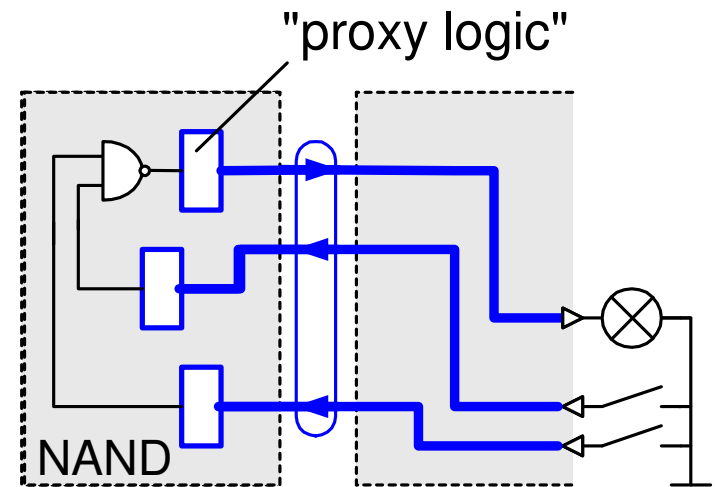
- **5. Step:** Run design rule check (DRC) to verify the design
- **6. Step:** Create the first reconfigurable configuration
 - Consisting of the static module and for each RP a RM
 - Implement this configuration
 - Promote this configuration
- **7. Step:** Create further configurations for each module in a RP:
 - Import the static design
 - Implement the partial module
- **8. Step:** Create the static and partial configuration bitfiles



Design Flow: Xilinx PlanAhead

Differences between the ReCoBus-Builder approach and PlanAhead:

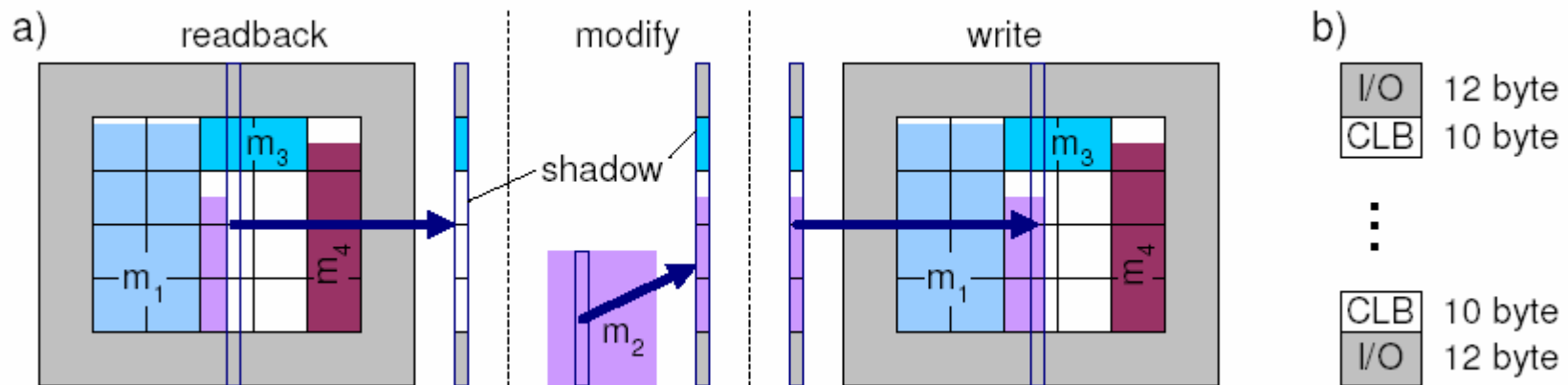
- Slot-style or grid-style vs. island style reconfiguration (island style has no external fragmentation problem → simple placement)
- ReCoBus allows module relocation and multi module instantiation
- Proxy logic bounds a module to a particular fixed region (RP)
- Example: 3 islands and 4 kinds of modules requires $3 \times 4 = 12$ physical implementations (place&route)
- All partial modules have to be re-implemented in case of changes in the static system (does not scale for complex systems)



Design Flow: FPGA Issues

In Xilinx FPGAs, the smallest atomic piece of configuration data is a configuration frame that contains data for all (older devices) or a set of vertical aligned CLBs (newer devices)

- Arbitrary configuration update is possible using readback-modify-write

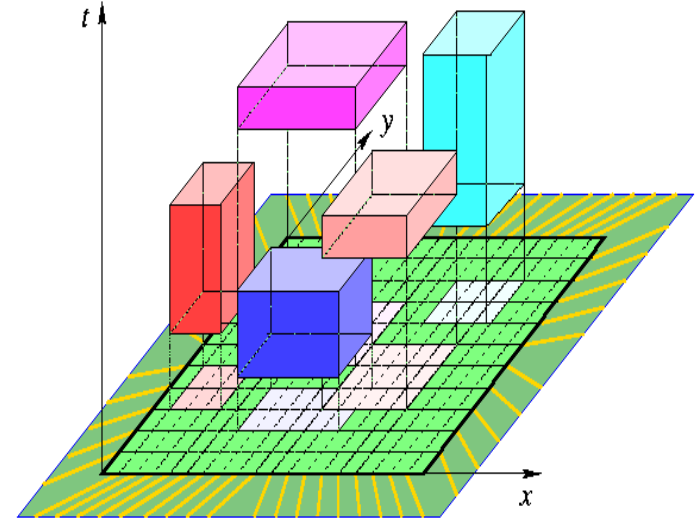
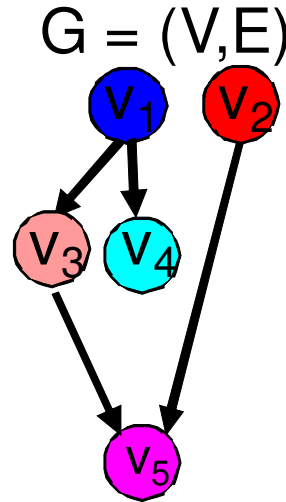


- Instead of readback, a configuration image might be stored in memory to avoid the relatively slow readback process
- **Warning:** Using LUTs as memory elements (e.g., SRL16 mode) might result in side effects, when updating modules above or below these primitives because **LUT values get overwritten**.

Run-time Management

- Main problem: online temporal module placement

Problem: map a DFG
onto a reconfigurable
area such that the
schedule is feasible
and the total execution
time is minimized.

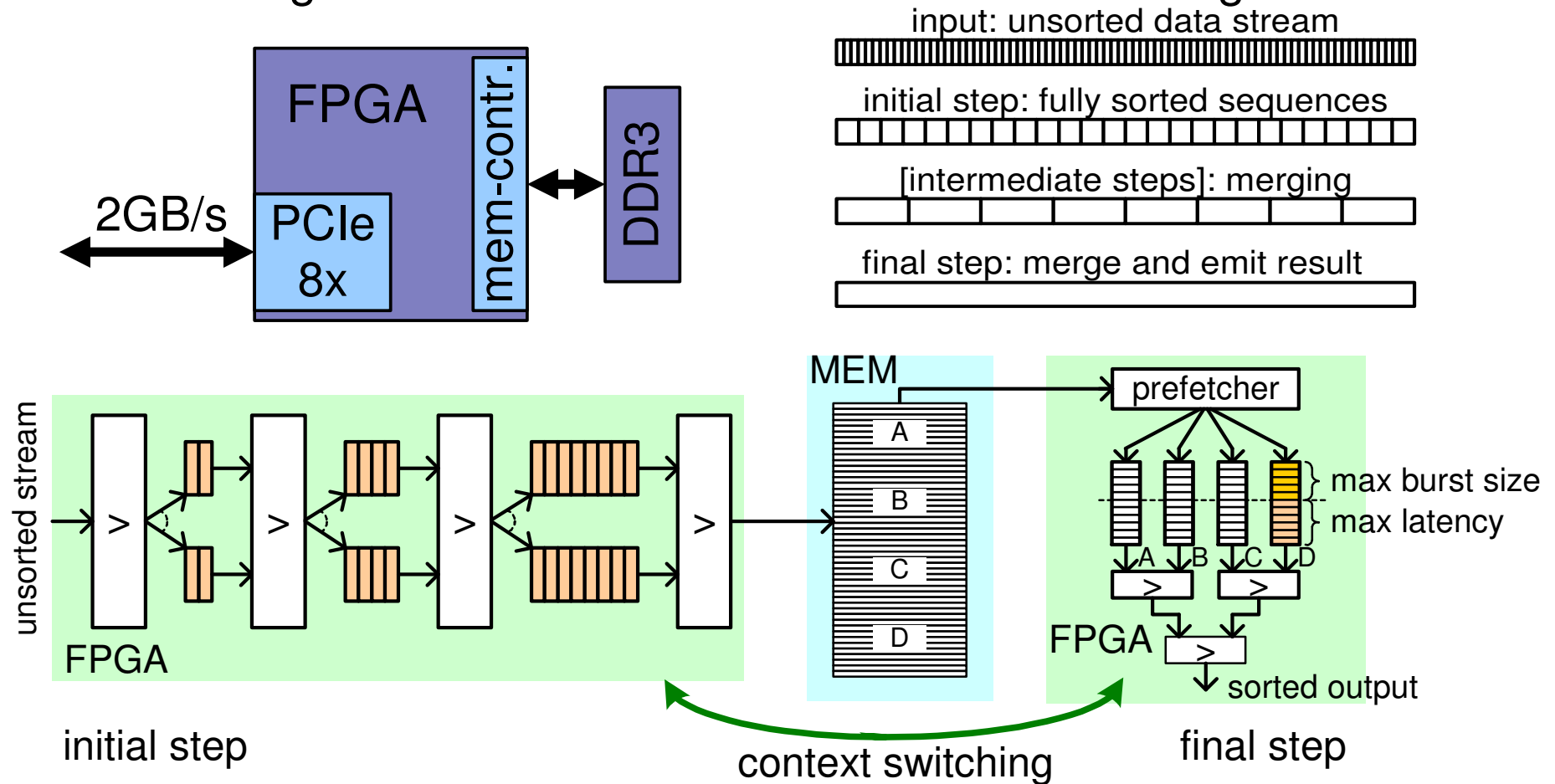


In other words:

- computing module placement positions
- and schedules
- Question: predictable (offline) vs. unpredictable (online) problem

PR example: Sorting for Database Acceleration

- Sorting contributes to 30% of the CPU time in huge databases

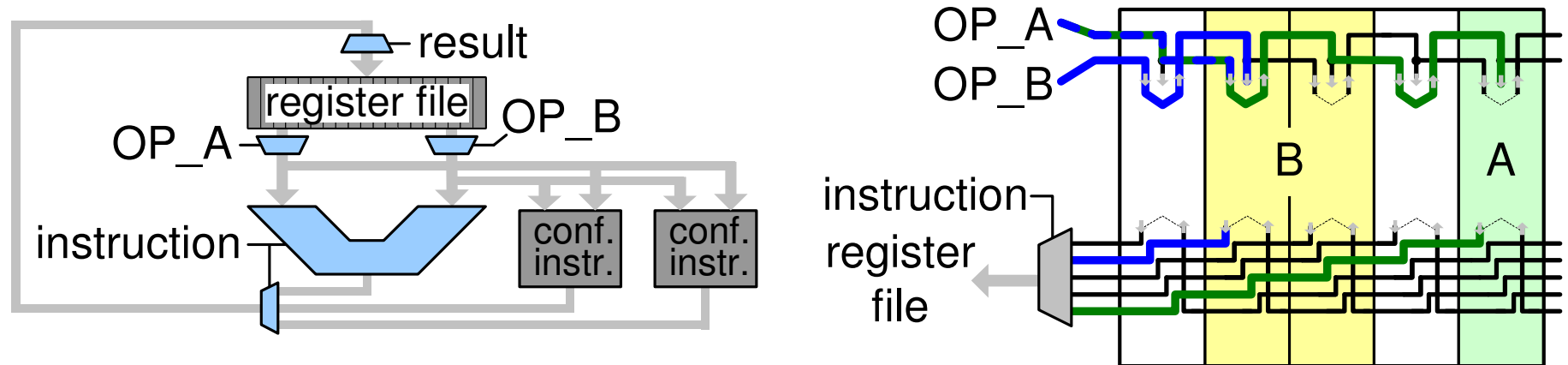


50% area saving or 4 times larger problems as compared to a static design

- Next step: hierarchical reconfiguration: swap comparator cells for different data types (integer, text, ...)

PR example: Custom instructions

- Fine-grained communication architecture for flexible instruction placement



- Identical routing for OPs and results in each slot
- Both operands are available in each slot (end point & middle access)
- Commutative instructions (e.g., $A > B$)
 - Implementation alternative
 - Bitstream manipulation

PR example: Custom instructions

instruction	slices	slots	bitstream	latency (max/av)
64-bit XOR gate	19 (40%)	1	2.64 KB	7.04 / 5.95 ns
CCITT CRC	33 (34%)	2	5.28 KB	5.32 / 3.98 ns
sat. add/sub	70 (73%)	2	5.28 KB	9.89 / 7.81 ns
barrel shifter	90 (94%)	2	5.28 KB	11.07 / 7.88 ns
'1'-bit counter	214 (89%)	5	13.2 KB	11.37 / 8.25 ns
mask & permute	16 (33%)	1	2.64 KB	5.94 / 4.05 ns

- Direct connection (no „proxy logic“)
- Swapping of instructions:
 - Dedicated load commands
 - Triggered by a trap handler