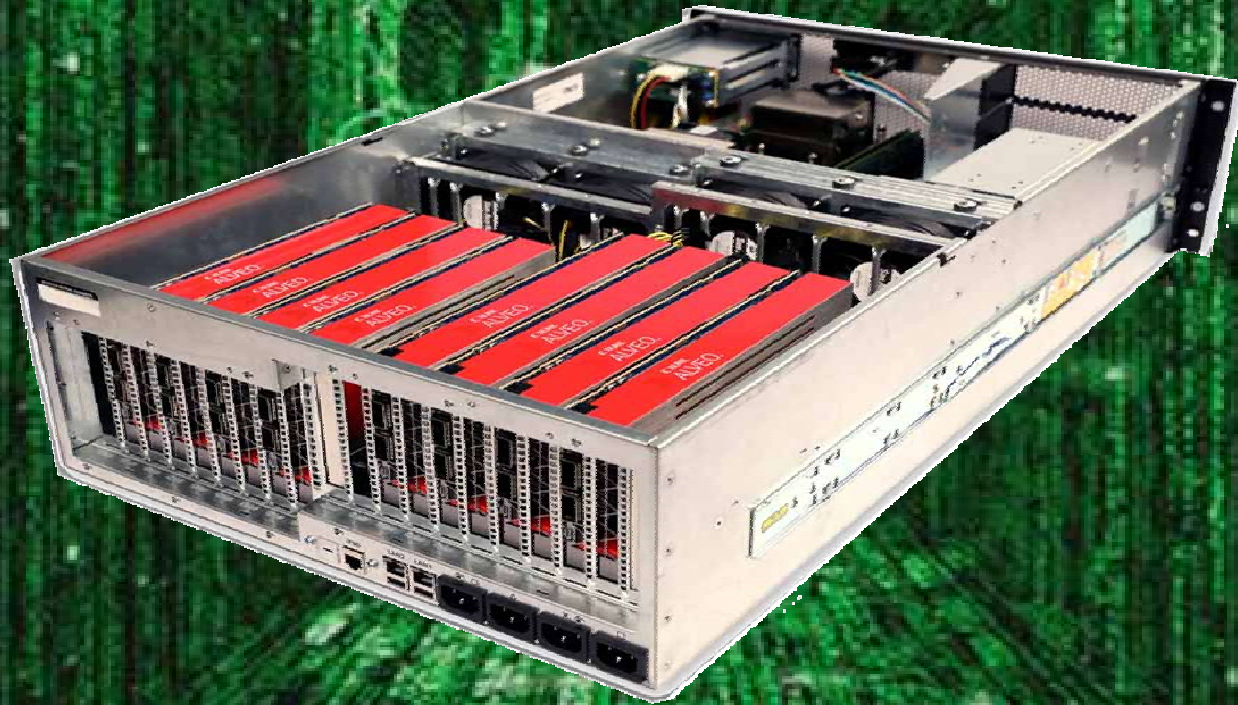


Better Performance through Resource-Elastic Dynamic Stream Processing on FPGAs



Dirk Koch, dirk.koch@ziti.uni-heidelberg.de
Heidelberg University (Germany), The University of Manchester (UK)

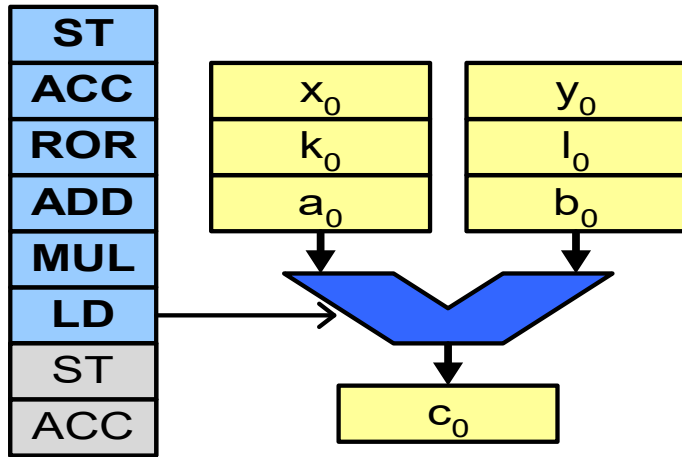
Dynamic FPGA Acceleration in Data Centers

- **Security**
(important but not the core of this talk)
- **“Plumbing”**
(How to implement infrastructure and user modules)
- **Runtimes and Services**
(APIs, programming models, drivers,...)
- **Applications**
(data analytics and database acceleration)

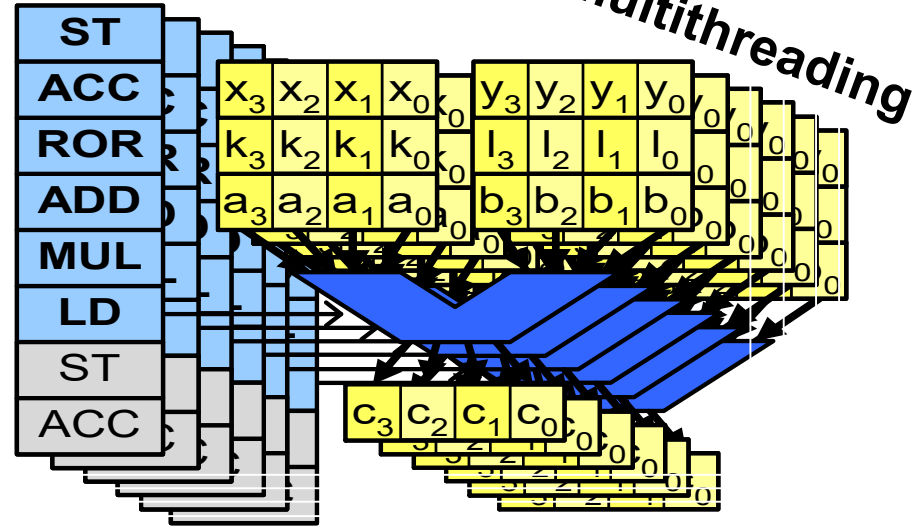


CPU vs. GPU vs. FPGAs

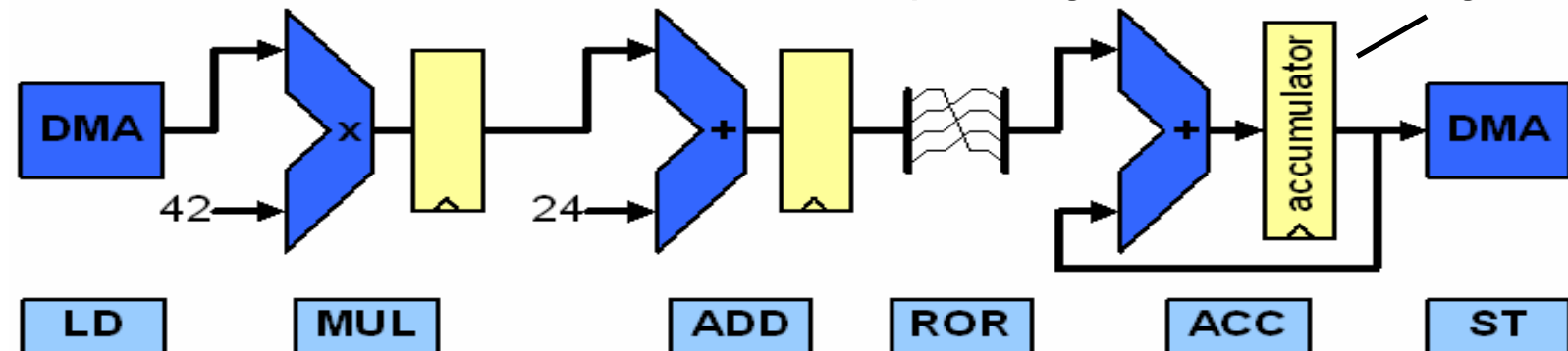
Scalar processing



SIMD processing (e.g. GPU)



Dataflow processing / pipelining (FPGAs)

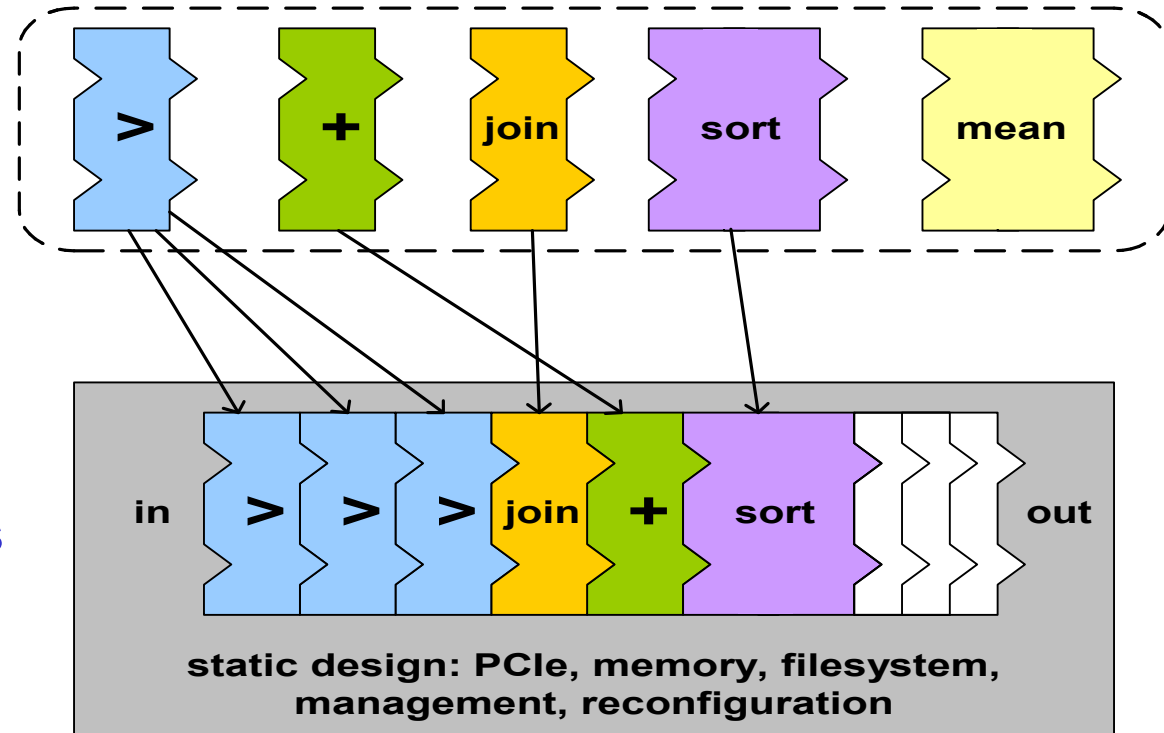


Pipelining often better for global states

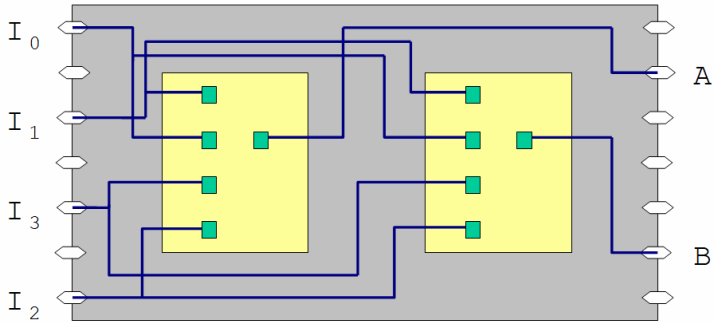
FPGAs are more than updatable ASICs!

Database acceleration example

- ASIC-like (fixed) accelerators often require an over-provisioning (poor resource usage)
- Better: use FPGA reconfiguration to just load currently needed modules
→ provides more resources to the currently running tasks (faster!)
- What about having a database or data analytics system where we can plug together operator modules to solve problems that we may only know at runtime?
→ **Dynamic Stream Processing** (uses partial reconfiguration)



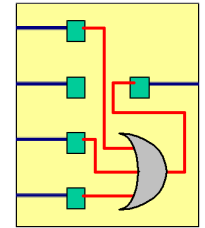
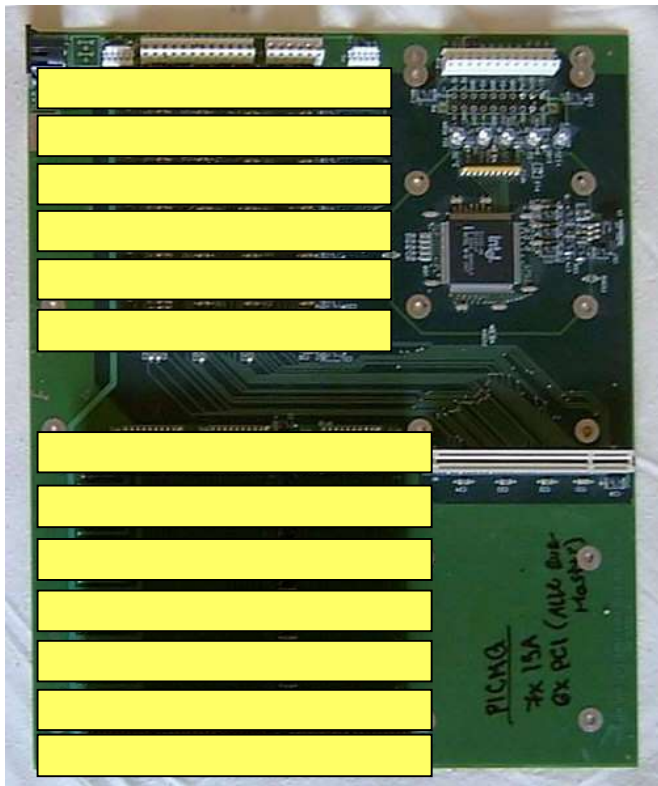
GoAhead PR Design Flow



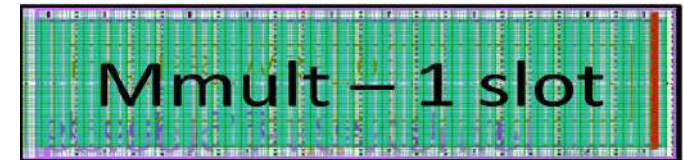
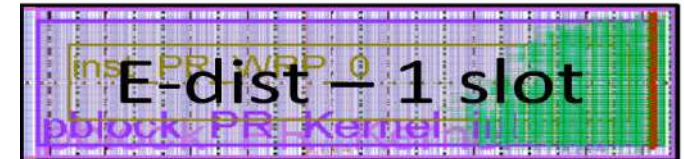
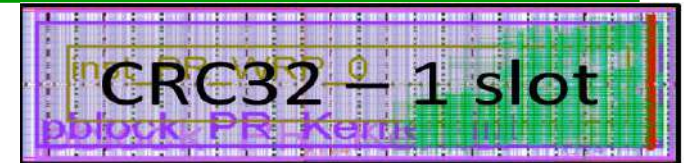
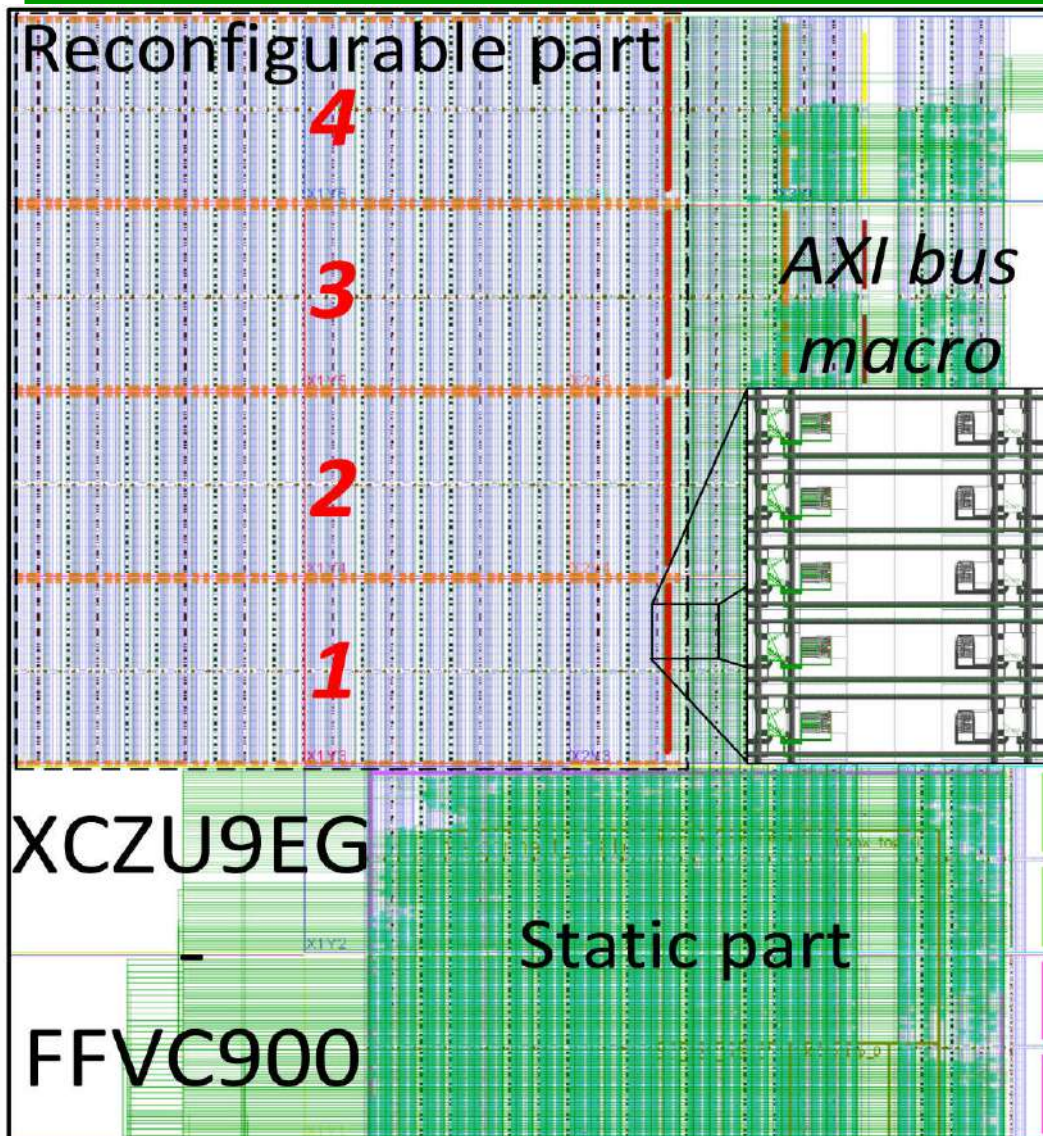
- The partial regions in a static system correspond to (ISA/PCI) slots on a PCB
- Modules are the equivalent to hot-swappable cards

■ Properties:

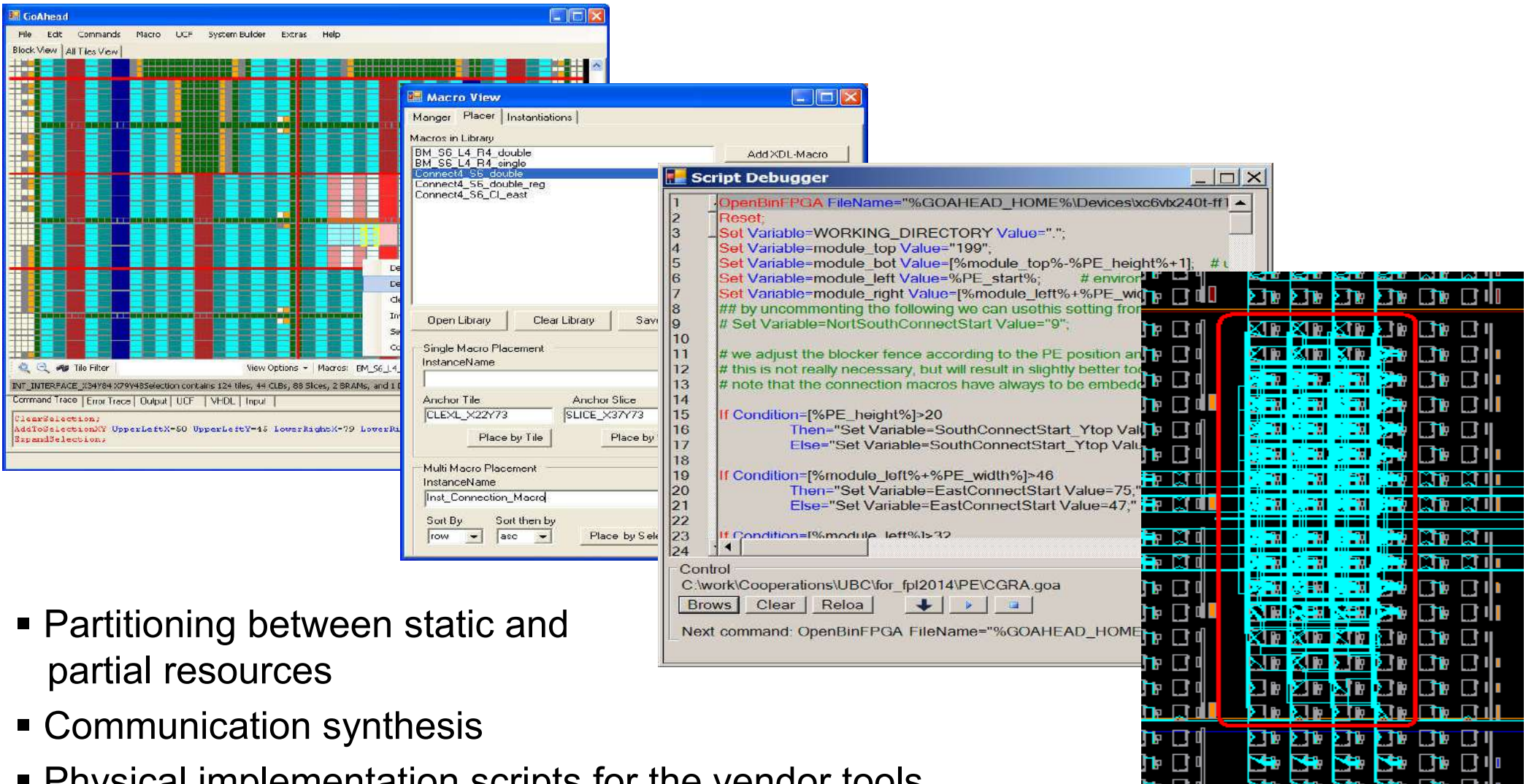
- Static system and modules implemented independently
- Simple integration through netlist or bitstream linking (including partial reconfiguration)
- Arbitrarily interchangeable



ZUCL (running OpenCL on Zynq UltraScale+, FSP'18)



GoAhead: a tool for implementing partially reconfigurable systems



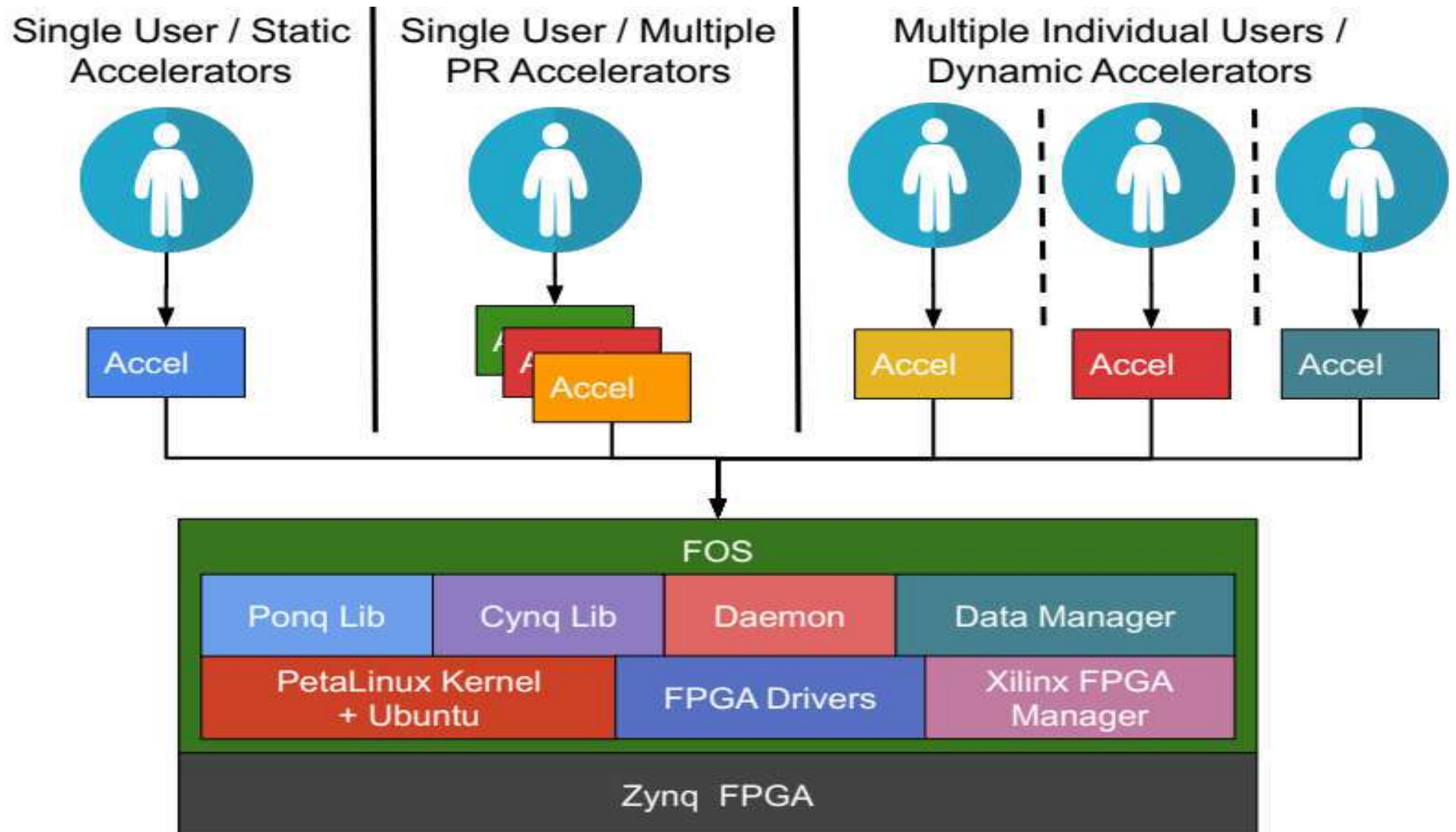
The image displays the GoAhead software interface, which is used for implementing partially reconfigurable systems. The main window shows a grid-based design with various colored blocks (red, blue, green, yellow) representing different components or resources. Overlaid on this are several windows:

- Macro View:** A window showing a list of macros in a library, including "BM_S6_L4_R4_double", "BM_S6_L4_R4_single", "Connect4_S6_double", "Connect4_S6_double_reg", and "Connect4_S6_CL_east". It also includes options for "Open Library", "Clear Library", and "Save".
- Script Debugger:** A window displaying a script with various commands and conditional logic. The script includes commands like "OpenBinFPGA", "Reset", "Set Variable", and "If Condition".
- Macro Placement:** A window showing options for "Single Macro Placement" and "Multi Macro Placement", including fields for "InstanceName", "Anchor Tile", and "Anchor Slice".

On the right side, a detailed circuit diagram is visible, showing a grid of components with a red box highlighting a specific area of interest.

- Partitioning between static and partial resources
- Communication synthesis
- Physical implementation scripts for the vendor tools

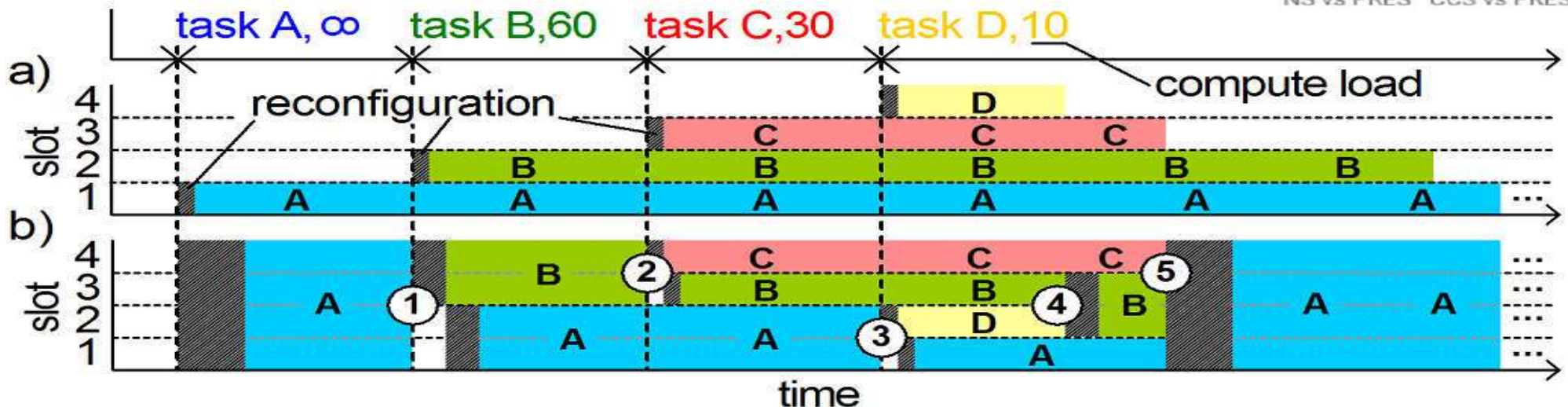
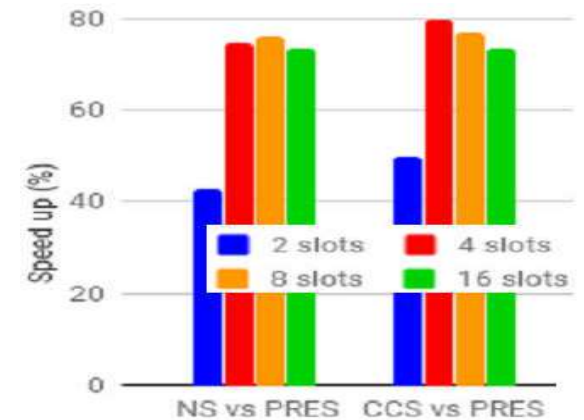
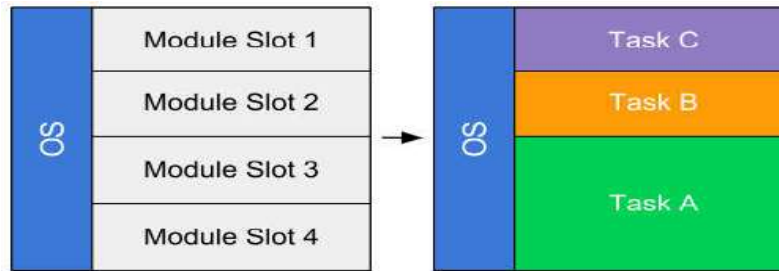
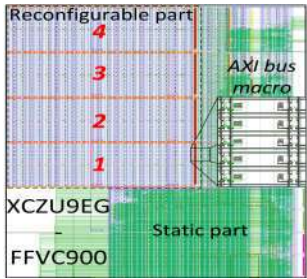
The FOS (FPGA Operating System) Stack



Resource Elastic Virtualization on FOS

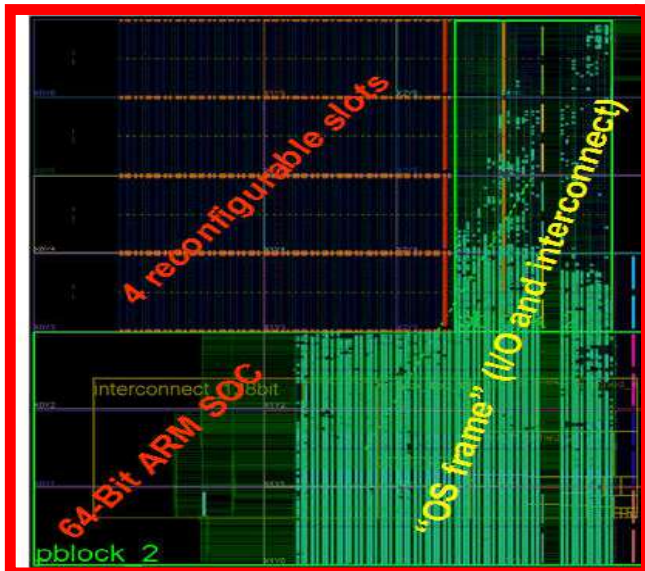
Resource Elastic Virtualization for FPGAs for OpenCL

- Using aggressive reconfiguration to keep utilization high in a dynamic scenario
- Virtualization in the space-domain (time-domain fallback, if needed)



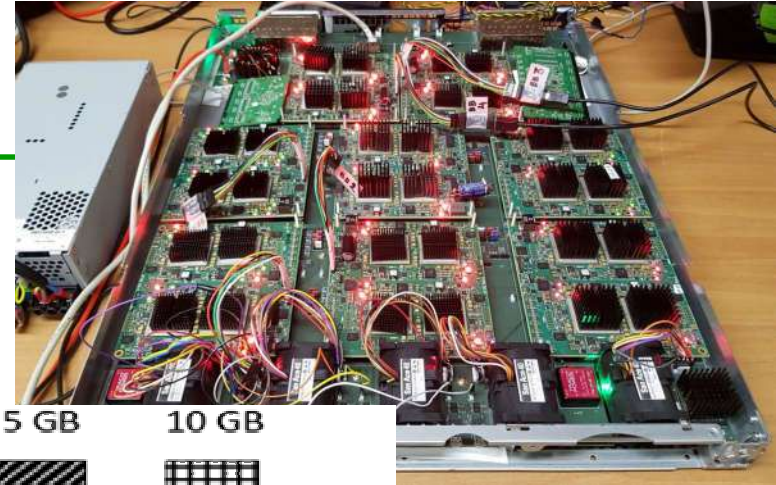
FPGAs for Datacenters (H2020 ECOSCALE)

ECOSCALE demonstrator fully-populated 1u blade with
32 x Zynq UltraScale+ (ZU9EG with 16GB/FPGA or 512 GB/blade)

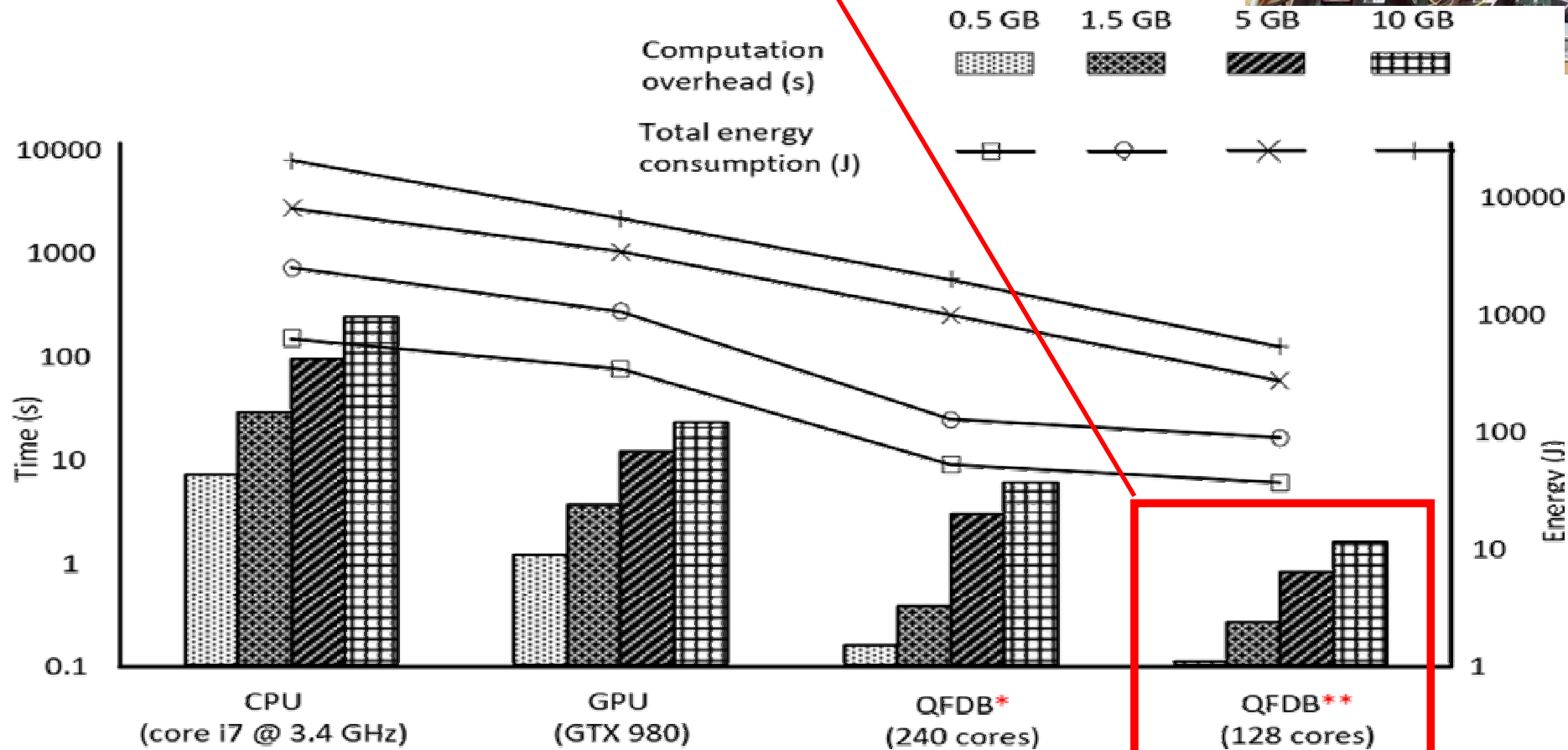


www.ecoscale.eu

FPGAs for Datacenters



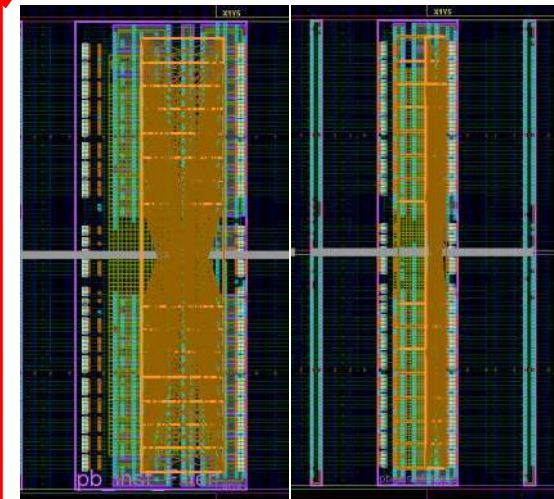
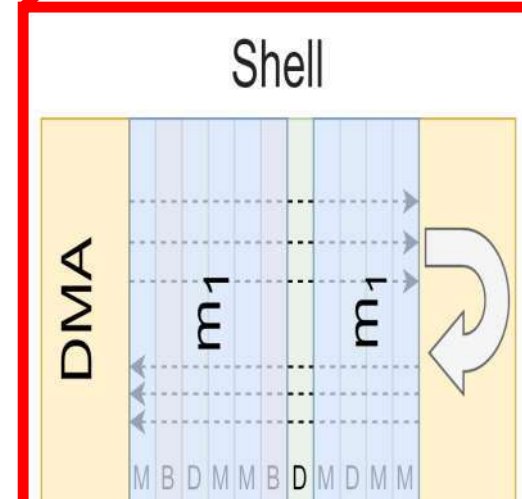
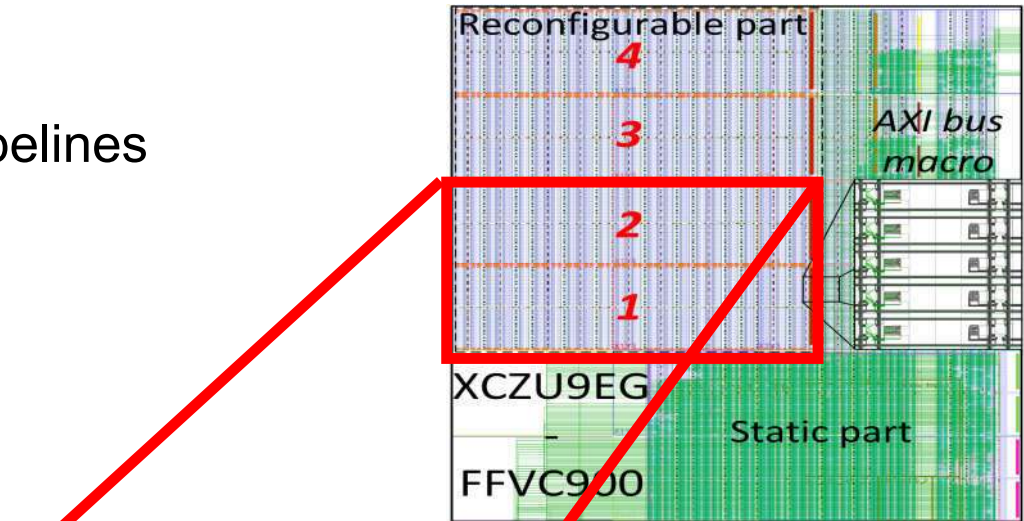
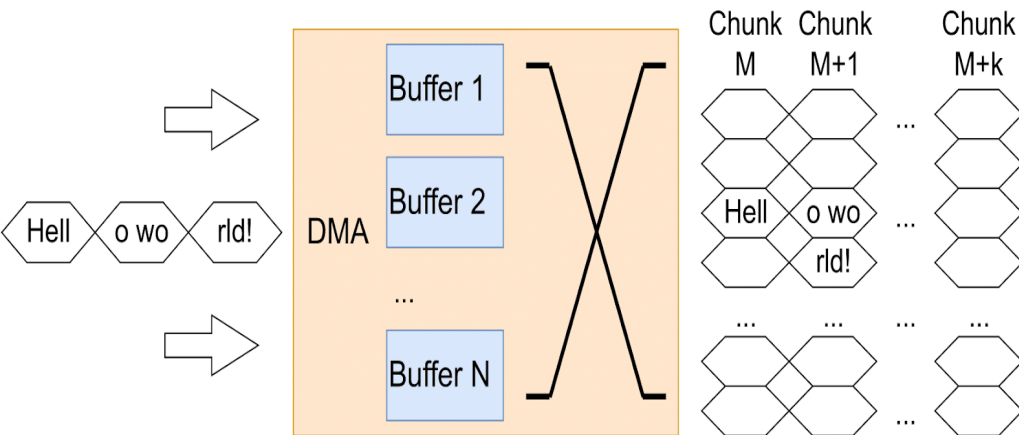
Partial reconfiguration allows moving compute to data → huge energy savings



Dynamic Stream Processing

Integrated into FOS

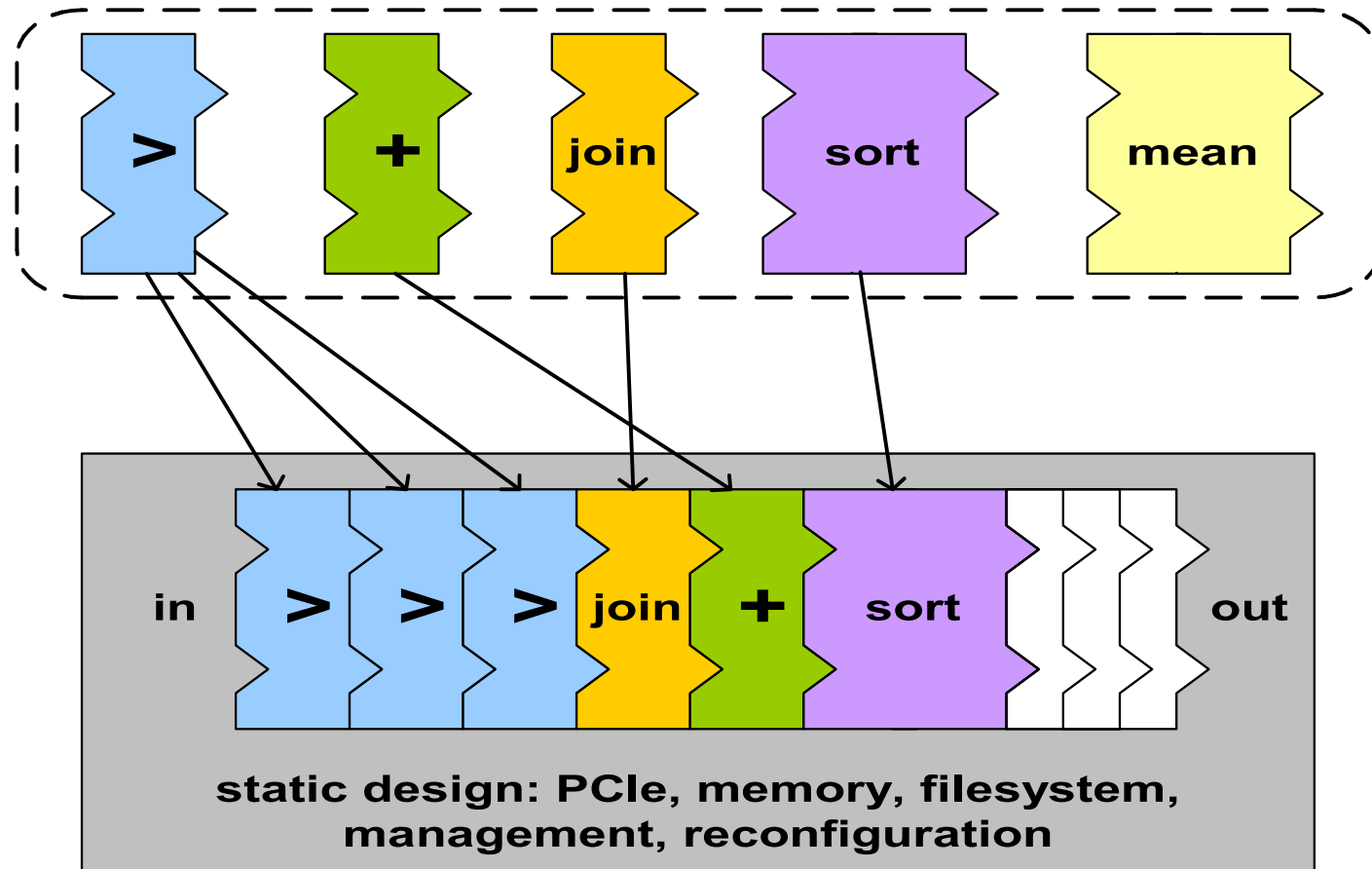
- Run monolithic accelerators **and...**
- ...Run composable stream processing pipelines
 - supported by a DMA engine and dedicated communication protocol
 - Virtual streams
 - Credit-based flow control
 - Register-file read/write access



Dynamic Stream Processing

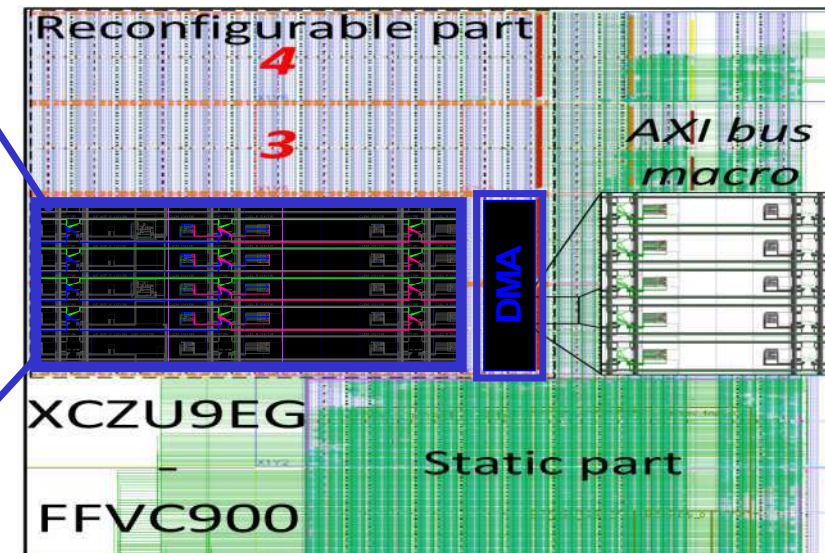
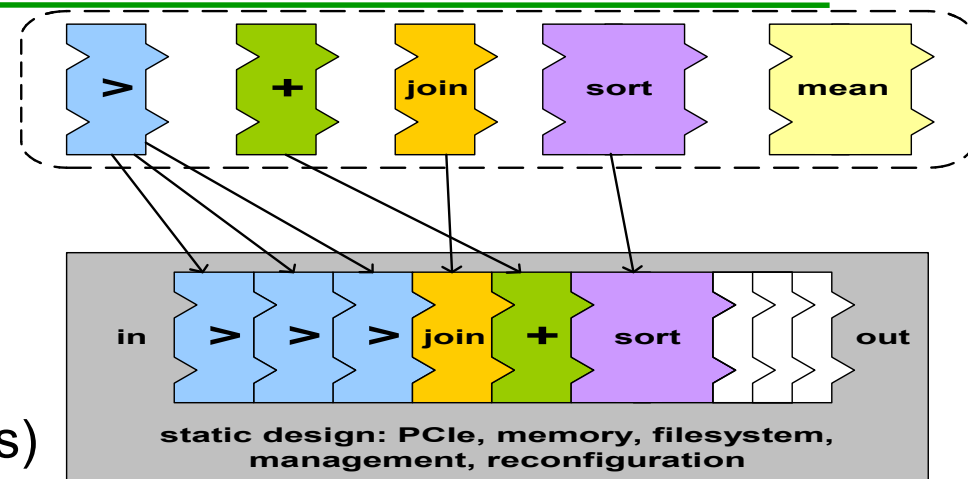
Database acceleration example

- Build library with SQL operators
- Compose optimized datapath at run-time
- Blocking operators are natural reconfiguration point!
→ operators before and after a sorter run mutual exclusive (never together)



Dynamic Stream Processing

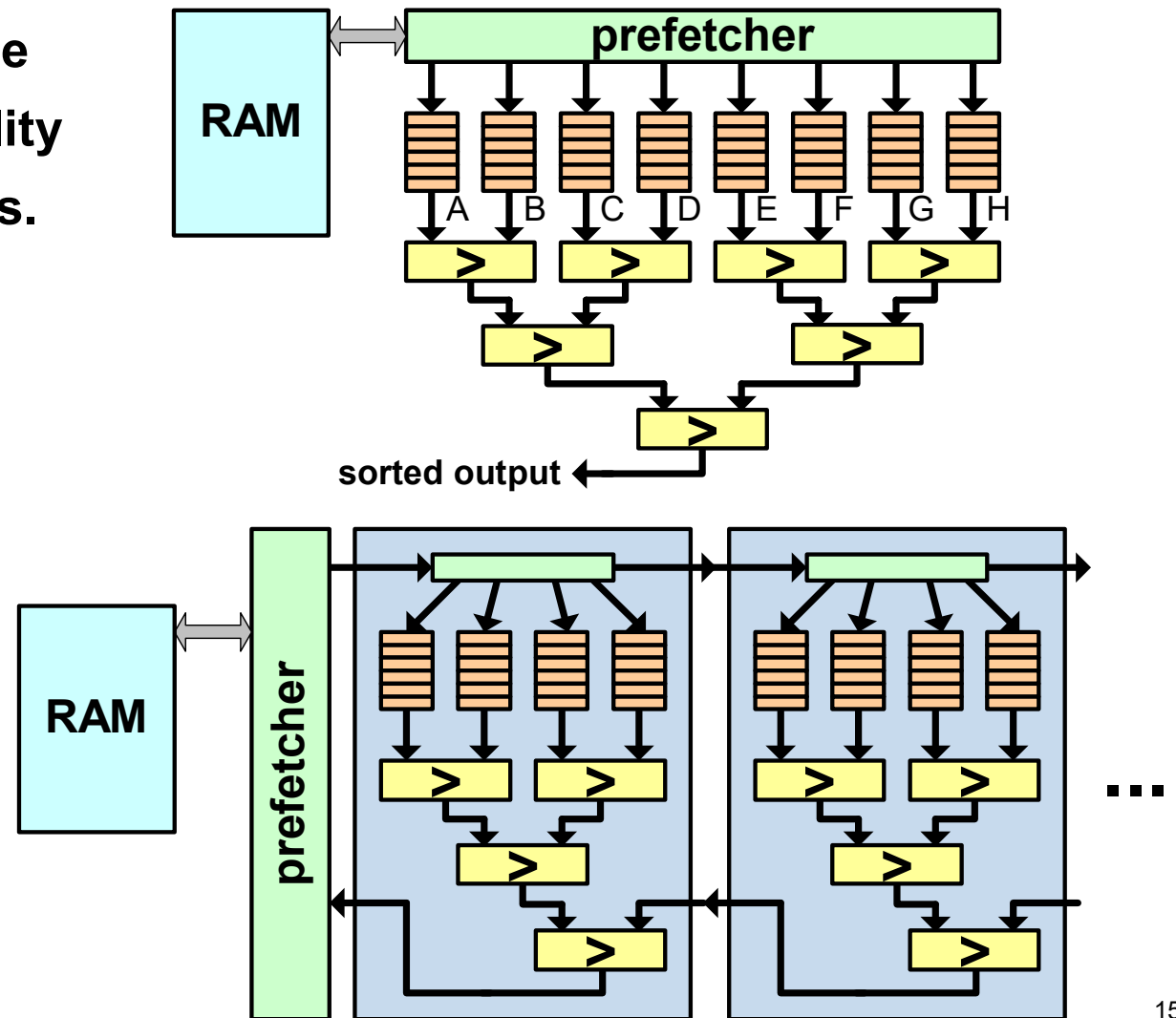
- Can run all TCP-H queries
- Modules with different area-performance/utility tradeoffs
- Modules with different resource footprints (improves placement for heterogeneous resources)



Dynamic Stream Processing – Resource Elasticity

Resource Elasticity allows a runtime system to maximize throughput/utility for the currently available resources.

- Example Sorting: build larger sorters from composable accelerator PEs → more work-per-run → fewer runs (High-utility sorting can merge thousands of streams in one run!)
- Works for WHERE clauses (number and complexity of clauses/regular expressions)
- JOIN: larger buffering reduces runtime



```

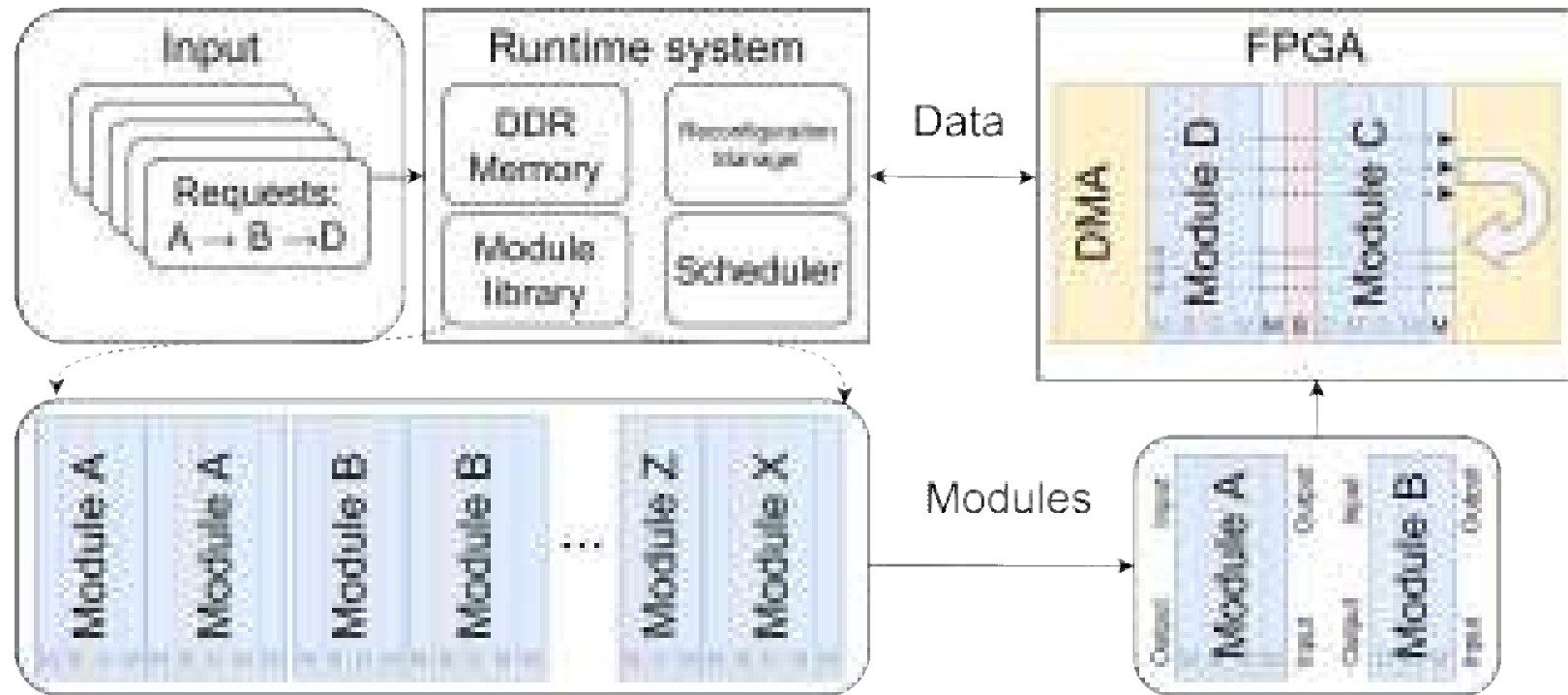
SELECT Sum(l_extendedprice * ( 1 - l_discount )) AS revenue
FROM   lineitem,
       part
WHERE  ( p_partkey = l_partkey
        AND p_brand = 'Brand#12'
        AND p_container IN ( 'SM CASE', 'SM BOX', 'SM PACK', 'SM PKG' )
        AND l_quantity >= 1
        AND l_quantity <= 1 + 10
        AND p_size BETWEEN 1 AND 5
        AND l_shipmode IN ( 'AIR', 'AIR REG' )
        AND l_shipinstruct = 'DELIVER IN PERSON' )
OR ( p_partkey = l_partkey
    AND p_brand = 'Brand#23'
    AND p_container IN ( 'MED BAG', 'MED BOX', 'MED PKG', 'MED PACK' )
    AND l_quantity >= 10
    AND l_quantity <= 10 + 10
    AND p_size BETWEEN 1 AND 10
    AND l_shipmode IN ( 'AIR', 'AIR REG' )
    AND l_shipinstruct = 'DELIVER IN PERSON' )
OR ( p_partkey = l_partkey
    AND p_brand = 'Brand#34'
    AND p_container IN ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG' )
    AND l_quantity >= 20
    AND l_quantity <= 20 + 10
    AND p_size BETWEEN 1 AND 15
    AND l_shipmode IN ( 'AIR', 'AIR REG' )
    AND l_shipinstruct = 'DELIVER IN PERSON' );

```

We can execute TCP-H Query 19,
Xilinx cannot (Vivado HLS) 😊

Stream Processing Ecosystem

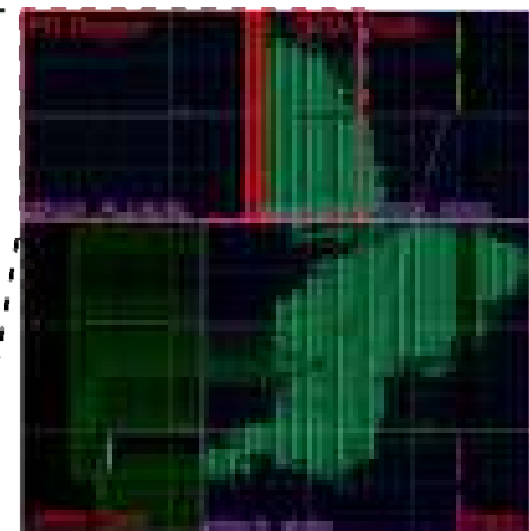
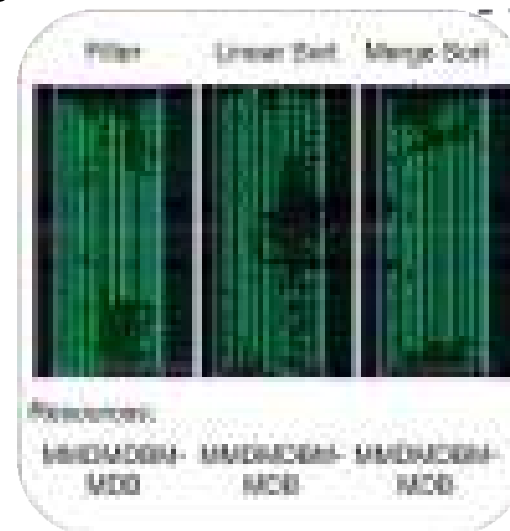
- Frontend
 - Parser
- Middleware
 - Scheduler
 - Operation management
- Hardware
 - Static
 - Module library
(including variants for different placement positions and area-performance tradeoffs)



Dynamic Stream Processing

Integrated into FOS

- Frontend:
 - Parse SQL – PostgreSQL
 - Create execution graph
- Middleware:
 - Schedule modules - Branch and Bound
 - Configure FPGA – Xilinx FPGA manager
 - Initialise modules with generic drivers
- Hardware:
 - Execute
 - Collect results



Resource-Elastic Dynamic Stream Processing

- FPGA acceleration allows us to tailor the I/O and memory subsystem to our problems (e.g., we can sort thousands of streams in a single run, thereby minimizing #runs)
→ design pattern fits to many big data stream processing problems and allows us to maximize the compute work per unit I/O (I/O is key for low power and for everything else)
- Our dynamic stream processing was demonstrated for SQL (and additional data analytics), video processing and some ML acceleration.
- Our modular approach is ideal for data scientists exploring various combinations of accelerator cores (you can plug bitstreams together in tens of milliseconds)
- Load accelerators as needed (e.g., dedicated de-compression for different data types)
- Take advantage of natural configuration points (again: sorting is a blocking operation)
- Our framework could work as a SQL / data analytics domain compiler (+ software stack)

Contributors

Malte Vesper malte.vesper@manchester.ac.uk (SSD stream processing infrastructure & applications)

Kaspar Matas kaspar.matas@manchester.ac.uk (Middleware for dynamic stream processing)

Christian Beckhoff (GoAhead support)

Khoa Pham khoa.pham@manchester.ac.uk (HLS support for PR and runtime management)

Anuj Vaishnav anuj.vaishnav@manchester.ac.uk (Resource-elastic FPGA virtualization & cloud infrastructure)

Kristiyan Manev kristiyan.manev@postgrad.manchester.ac.uk (Resource-elastic stream processing)

Babis Kritikakis babis_k4@hotmail.com (Dynamic Dataflow on Maxeler)

Tuan La tuan.la@manchester.ac.uk (FPGA hardware security)

Joe Powell joe-powell@manchester.ac.uk (FPGA hardware security)

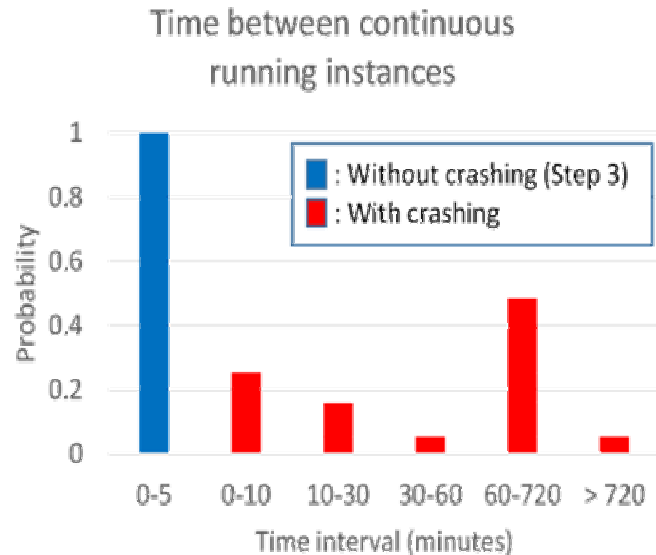
Dirk Koch dirk.koch@ziti.uni-heidelberg.de

Find all our projects on git: <https://github.com/orgs/FPGA-Research-Manchester/>

- **GoAhead** <https://github.com/FPGA-Research-Manchester/GoAhead>
- **FOS** <https://github.com/FPGA-Research-Manchester/fos>
- **Dynamic Streams** <https://github.com/FPGA-Research-Manchester/OrkhestraFPGAStream>
- **FPGA Virus Scanning** <https://github.com/FPGA-Research-Manchester/FPGAVirusScanner>

One slide on our FPGA security work

- Sidechannel attacks: no real threat (but fault injection, possibly Trojans)
- Power-Hammering Potential (Alveo U200)
 - Oscillators and glitch amplifiers > 2 KW
 - Wires > 5 KW
 - Flops > 2 KW
 - + BRAMs (write collisions), DSPs, Gbit transceivers...



- DoS attack on AWS F1 (we crashed >100 F1s, see TCHES 2021)
- Bitstream virus scanning → reject malicious designs (TRETS 2020)
- FPGA TEEs (FCCM 2021)
- FPGA health monitoring

MANCHESTER
1824
The University of Manchester

FPGA Bitstream Virus Scanning
Joseph Powell, Kaspar Matas, Kristijan Manev, Dirk Koch
jpowell@man.ac.uk, kmatas@man.ac.uk, kristijan.manev@man.ac.uk, dirk.koch@man.ac.uk
School of Computer Science, The University of Manchester, UK

Introduction
FPGAs enjoy a large range of users and applications, with both sensitive commercial and military uses. Often FPGAs are provided as a service from a cloud service provider. These operating environments raise important security concerns in regards to confidential data and IP loss, as well as service and even hardware compromise.

There are many attacks on FPGAs including:

- Side channel data extraction
- Attacks on power supply infrastructure
- Injection of hardware trojans
- Cloning of bitstreams
- Accelerated FPGA fabric aging

These attacks tend to use the same kind of primitives in their designs, when including:

- Oscillators (ring, carry-chain, DSP, latch-based)
- Short circuits (small configuration encoding)
- Combinational glitch amplification (glitch generation)
- Excessively high signal rates
- Use of prohibited resources (IO, CAP, PCIE, CTX)
- Use of prohibited fabric area (bouncing buffers)

By detecting the use of these primitives in an FPGA bitstream, we can gain an insight into the safety and security of a bitstream before we program FPGAs with it.

Bitman
Bitman (and similar tools likeStreamParser) is capable of performing bitstream configuration command decoding, decoding the configuration register settings, and the configuration data blocks.

Bitman can remove, copy, and erase the configuration data, as well as perform boolean operations with other data.

Finally, Bitman can export the new configuration data blocks as valid bitstream files.

CarpenteriUT
CarpenteriUT is a library to convert binary FPGA configuration data into LUT configurations, FIP settings, and other primitive configurations.

The extracted primitive configurations are combined with the basic FPGA routing topology for that chip to generate an implemented design as a large graph.

The netlist graph can then be used by other applications, or imported and exported to files.

FPGA Virus Scanner
The Virus Scanner is software that scans the CarpenteriUT netlists to detect malicious structures. The software contains a number of signatures, each of which is to detect and report on the different kind of malicious structures.

Each of the signatures generates a score, which can be used to estimate the severity of the findings as well as any extra information of note.

The Virus Scanner can be used via an online service, allowing the software to be used remotely by anyone.