Karlsruhe Institute of Technology
Institut für Technische Informatik
Chair for Embedded Systems – Prof. Dr. J. Henkel

Vorlesung im SS 2016

# Reconfigurable and Adaptive Systems (RAS)

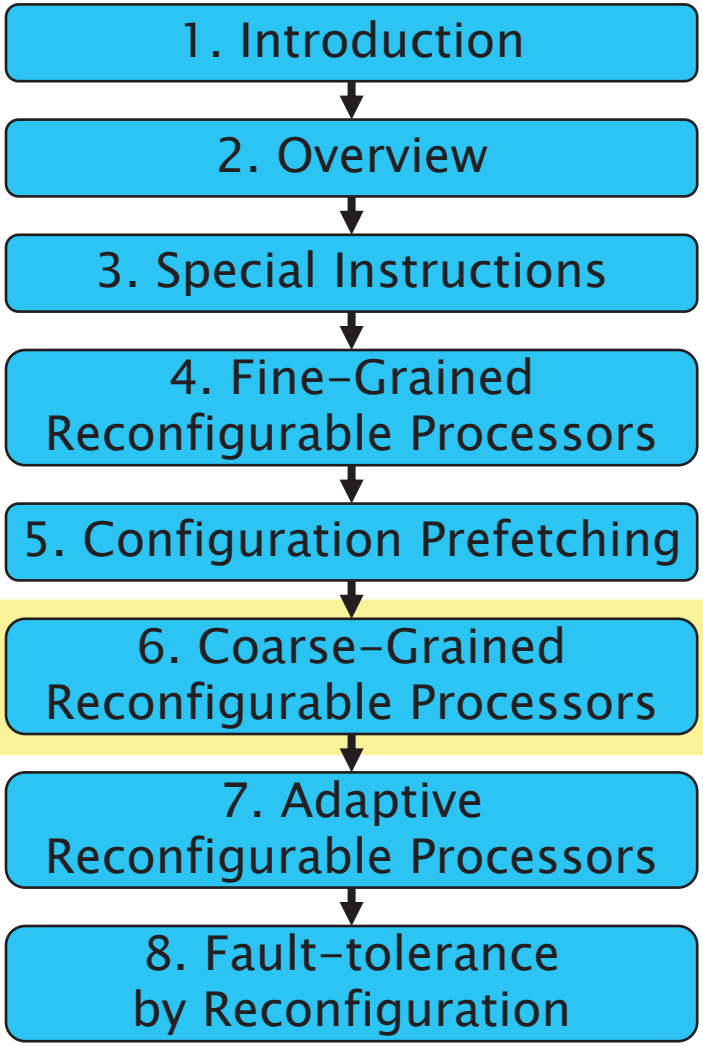
Marvin Damschen, Lars Bauer, Jörg Henkel

Karlsruhe Institute of Technology
Institut für Technische Informatik
Chair for Embedded Systems – Prof. Dr. J. Henkel

# Reconfigurable and Adaptive Systems (RAS)

## 6. Coarse-Grained Reconfigurable Processors

# RAS Topic Overview




- Chameleon SoC with Montium core
- ADRES
- MT-ADRES
- PipeRench

# 6.1 The Chameleon SoC with the Montium Tile Processor

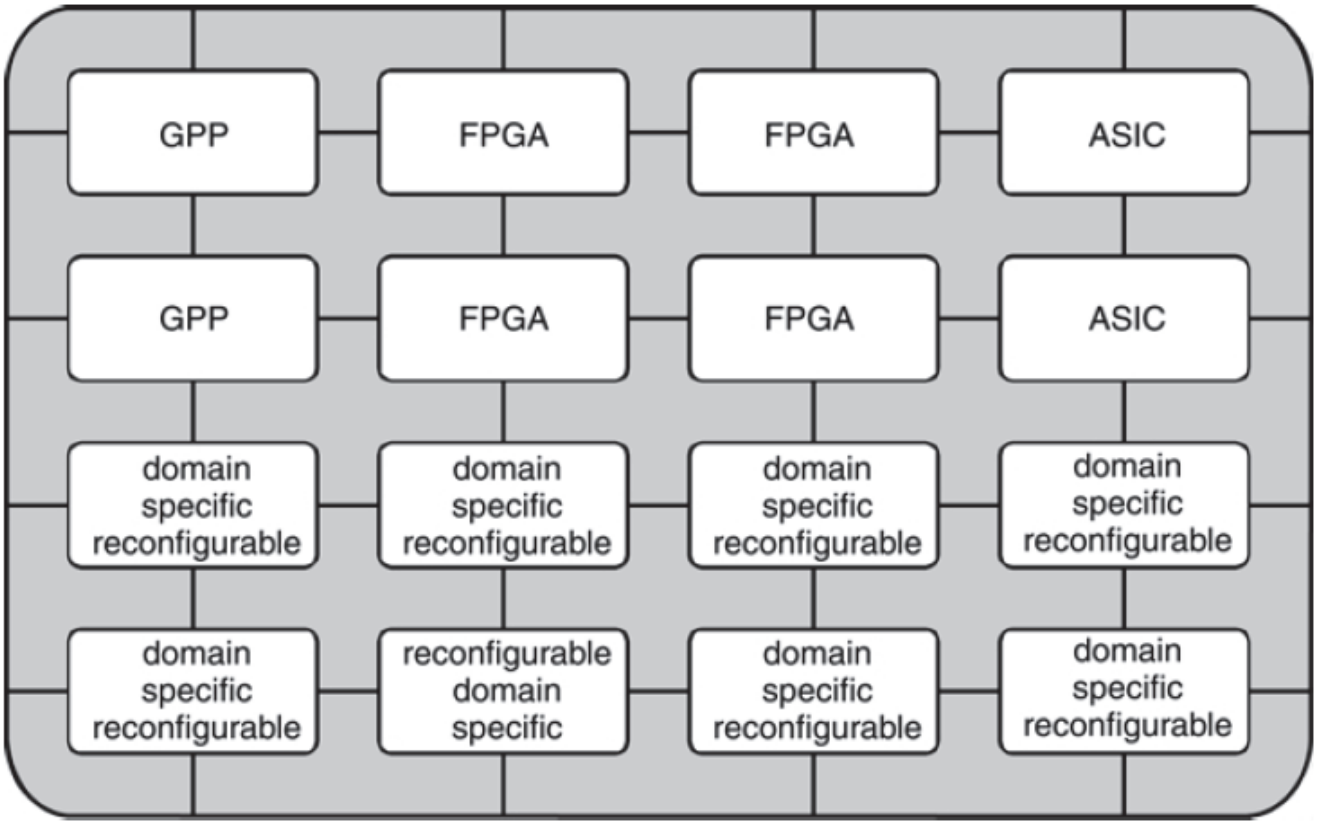

src: recoresystems.com

# Overview

- A coarse-grained reconfigurable processing tile
  - Intended to be integrated with other processing tiles into a System-on-Chip
- Developed at University of Twente, Netherlands
  - Now company: Recore Systems
- Aims at combining flexibility with efficiency
  - Reduced flexibility in comparison with fine-grained reconfigurable logic
  - But increased efficiency if the application requirements match the provided flexibility

# Chameleon System

- Applications are either implemented as ASICs or the tasks are prepared as a GPP implementation, FPGA implementation, and/or Montium implementation (domain specific reconfigurable)
  - Heterogeneous SoC
- Tiles are connected with an on-chip network
- Run-time system decides on which tile a task shall execute



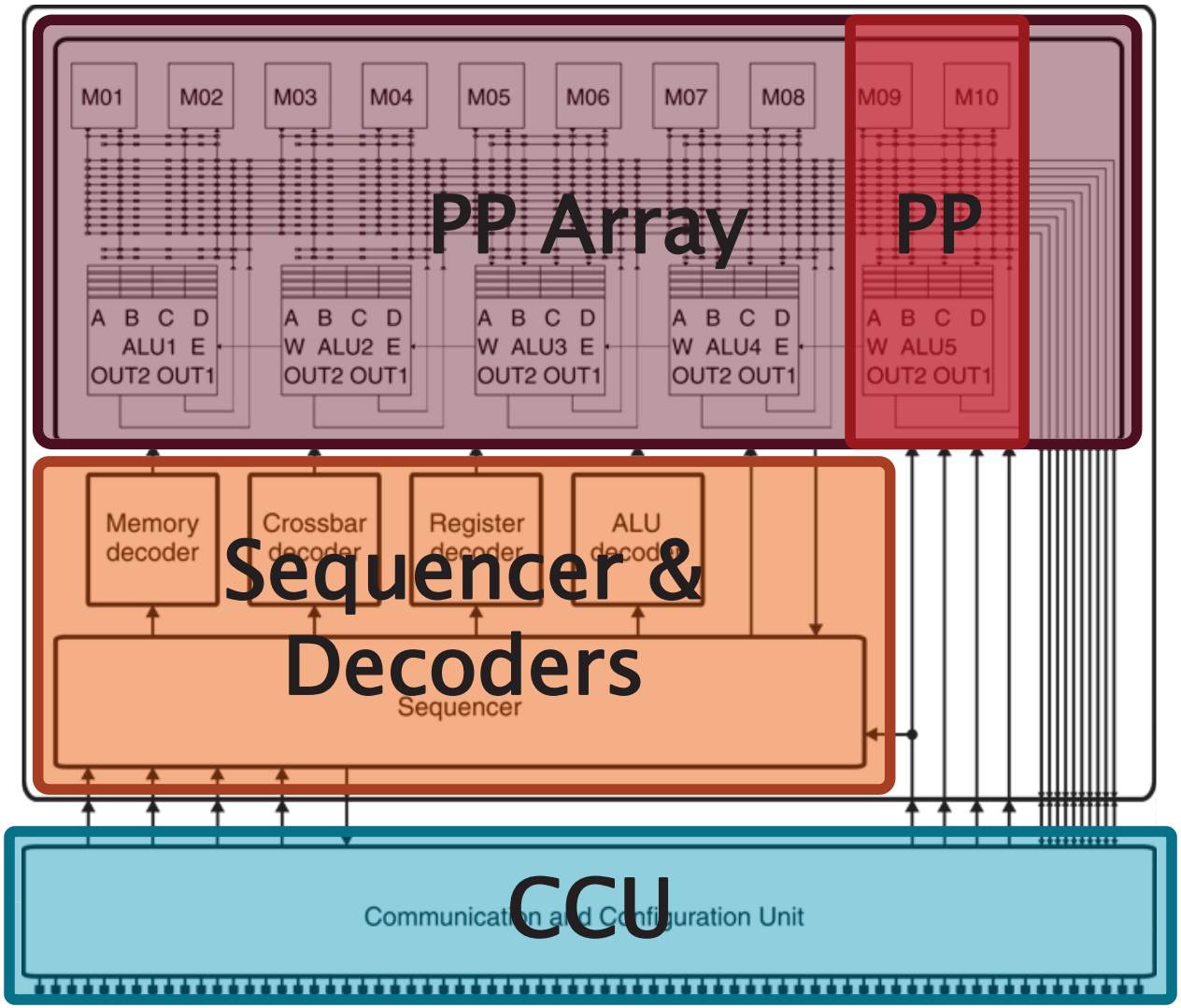

src: [HSM03]

# Montium Processing Tile

- Optimized for specific domains
  - Calculating typical DSP algorithms, e.g.
  - Fast Fourier Transformation (FFT)
  - Finite Impulse Response Filters (FIR-Filters)
  - Software Defined Radio: Rake Finger, HiperLAN/2, Turbo Coding (UMTS)
- Provides sufficient flexibility to implement this application domain and optimized for efficiency (e.g. energy wise)
- Montium can be used to accelerate kernels within the scope of (larger) applications that are distributed over the Chameleon SoC
  - Montium acts as a loosely coupled co-processor

# Montium Processing Tile (cont'd)

- Communication and Configuration Unit (CCU): external interface
- Sequencer / Decoders: Control and Configuration
- Processing Part (PP): computation
- PP Array



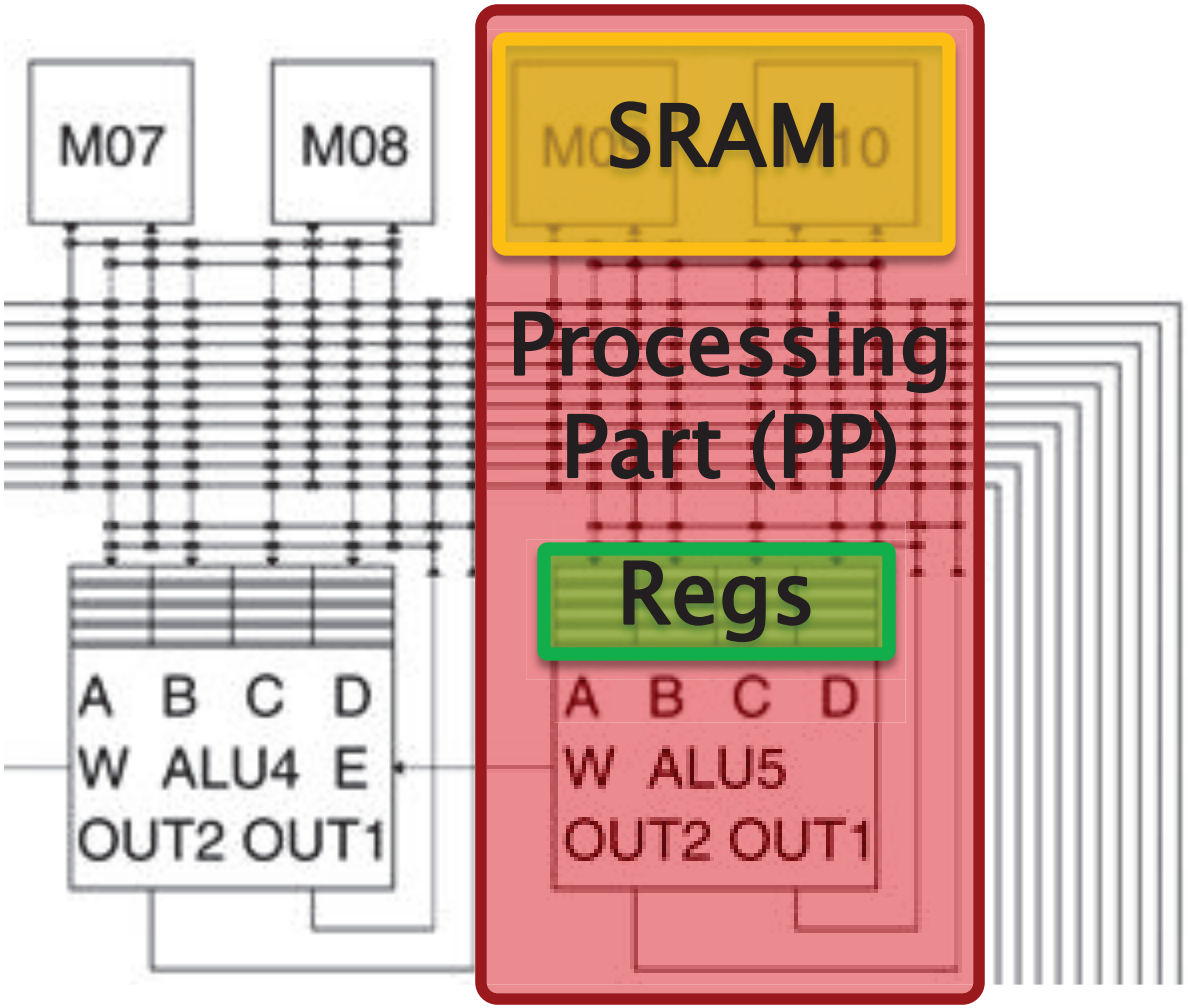

src: [HSM03]

# Montium Processing Tile (cont'd)

- 10 local memories provide high memory bandwidth
- Processing Part (PP) contains a coarse-grained reconfigurable ALU (more complex than a normal ALU), input register file, and parts of the interconnections
- 10 busses for inter-PP communication
- The CCU is also connected to the 10 busses to provide access to external input/output data
- The configuration of the interconnection network and the PP computation can change at every clock cycle

# Processing Part

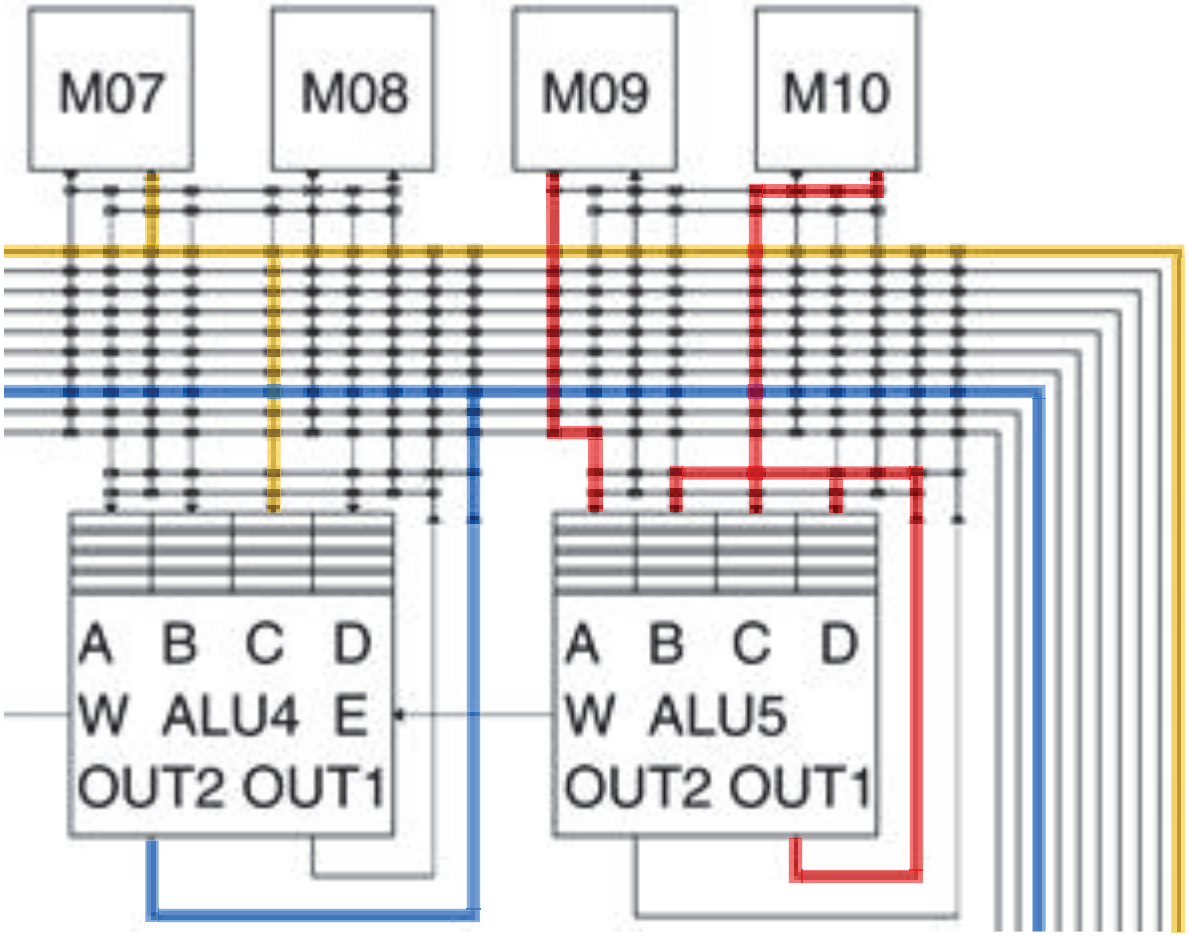
- 2 local 16-bit SRAM memories with 512 entries
- Each ALU input has a private input register file that can store up to 16 operands




src: [HSM03]

# Interconnects

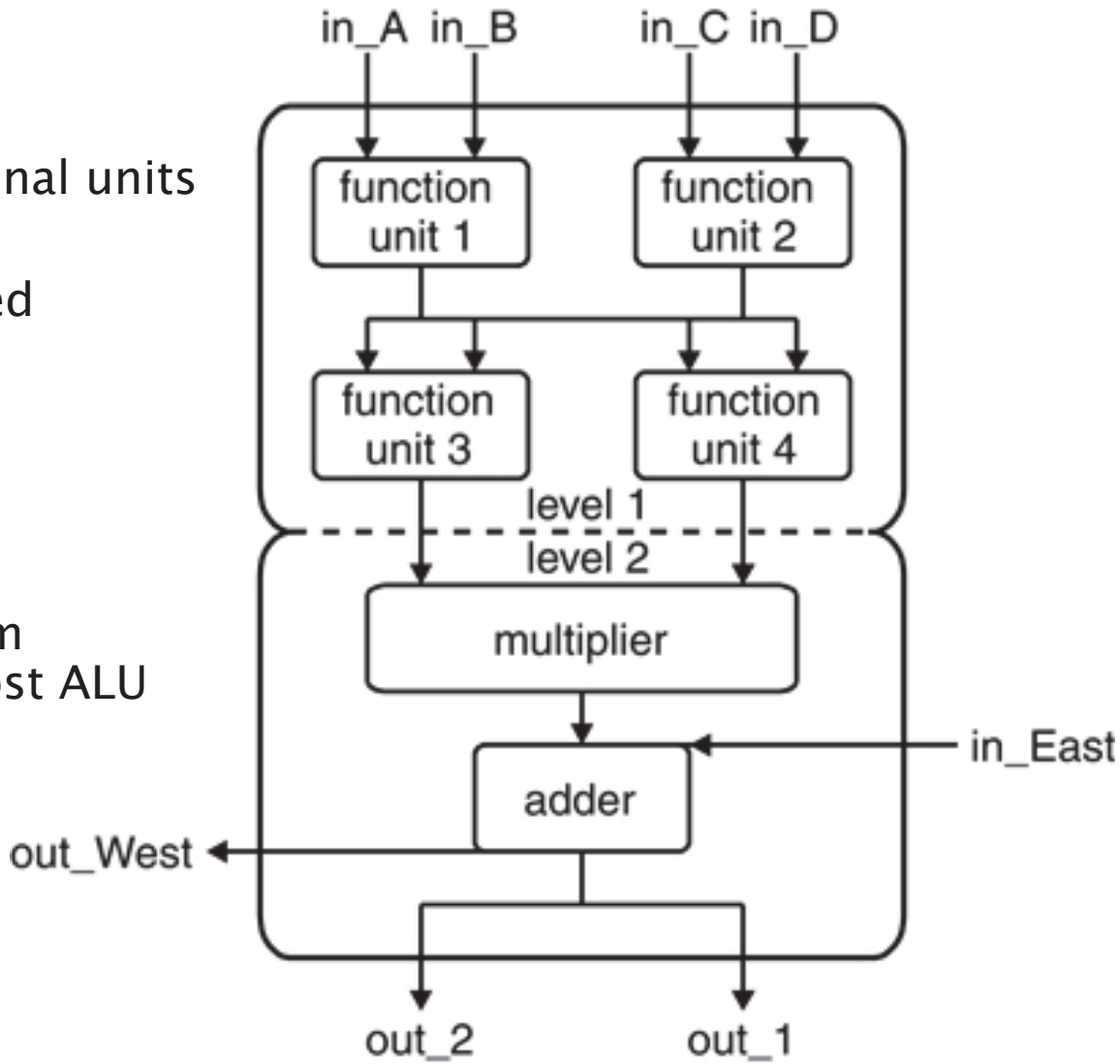
- External DMA Write
- Results to Global Bus
- Local References




src: [HSM03]

# ALU

- Two-tiered
  - Level 1: 16 bit functional units
  - Level 2: 32 bit MAC
  - Levels can be bypassed
- Input: 4 x 16 bit
- Output: 2 x 16 bit
- East to West: 32 bit
  - Critical Path goes from right-most to left-most ALU
- Single status output bit that can be tested by the sequencer



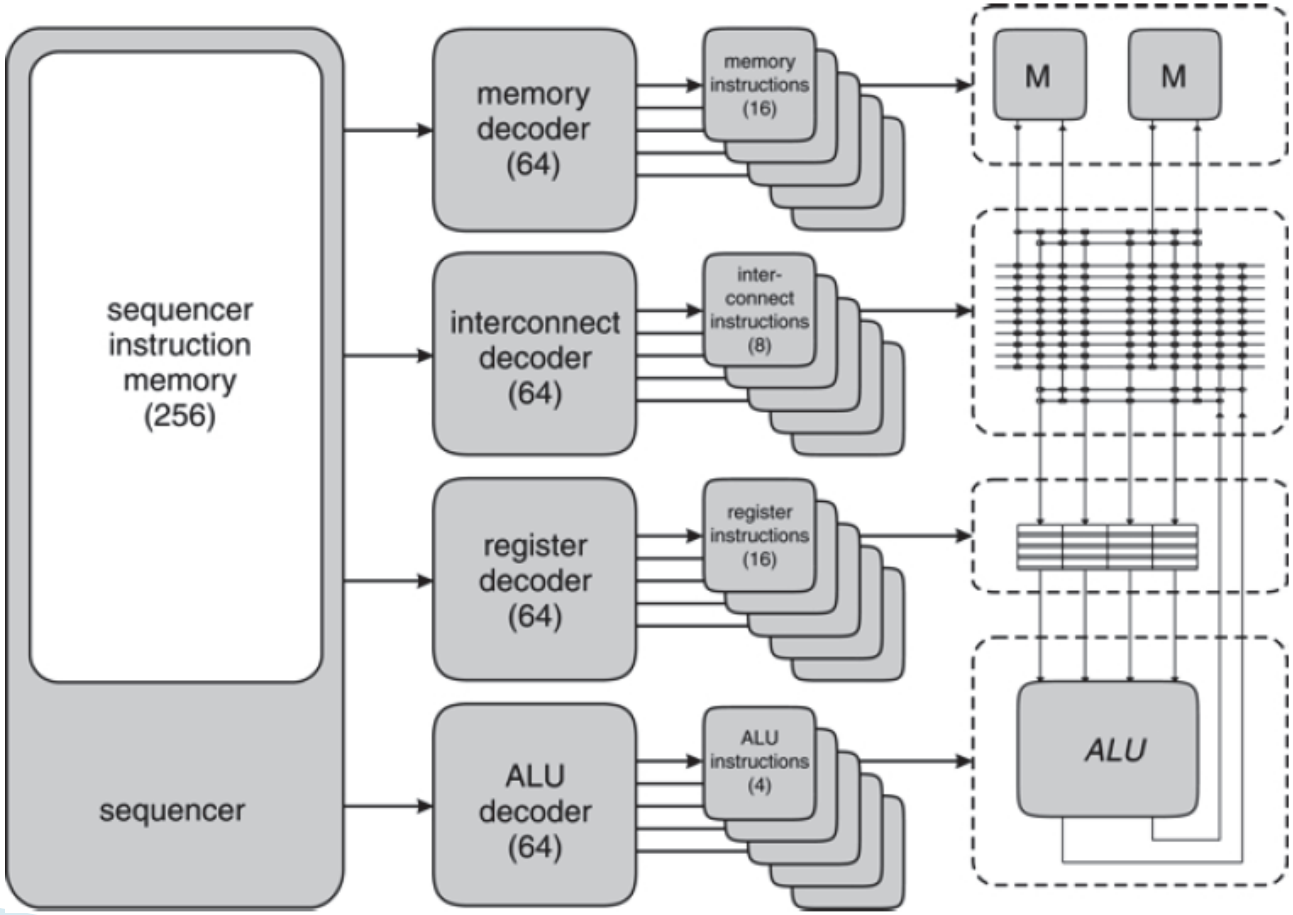

src: [HSM03]

# Sequencer

- The Sequencer controls the cycle-by-cycle reconfiguration of the PP Array, interconnects etc.
- The Sequencer has a small instruction set that is used to implement a state machine
  - Supports conditional execution and can test the ALU status outputs, handshake signals from the CCU, and internal flags
  - Supports up to 2 nested loops and non-nested conditional subroutine calls
  - Can store up to 256 instructions
- But: the flexibility of the PP Array results in a vast amount of control signals
- To reduce this overhead, a hierarchy of small decoders is used

# Sequencer (cont'd)

- Example: ALU Decoder
  - Each ALU contains a configuration register that contains up to 4 instructions that the ALU can currently execute
  - The ALU Decoder simply chooses one of these 4 instructions
  - Similar for input registers, interconnects, and memories




src: [HSM03]

# Communication and Configuration Unit (CCU)

- Interface for off-tile communication
- Typical use case:
  1. Remote configuration manager sends configuration binary to CCU
  2. CCU uses that binary to configure the Montium Tile Processor (TP)
     - Might even reconfigure parts of the CCU as well
  3. CCU receives input data and writes it into the local memories from the TP
  4. CCU signals the sequencer to start the operations
  5. At the end, CCU receives results from local memories and forwards them to off-tile destination

# Results: Application Kernels

| | Configura-tion: size of binary [byte] | Configura-tion: time [cycles] | Configura-tion: Total energy [nJ] | | Executi-on: time [cycles] | Executi-on: Total energy [nJ] |
|---|---|---|---|---|---|---|
| FFT64 | 946 | 473 | 182.8 | | 205 | 110.94 |
| FFT 1024 | 1432 | 716 | 276.34 | | 5141 | 2960 |
| FIR5 | 246 | 123 | 47.01 | | 515 | 192.63 |
| FIR20 | 540 | 270 | 104.95 | | 2055 + 2054 | 860.83 + 866.46 |

src: [H04]

# Results: Chip Area

- Synthesized for 130 nm technology from Philips
- Estimated 10% additional area requirements for wiring

| Montium Tile Processor | | |
|---|---|---|
| **Subcomponent** | **Area [mm$^2$]** | **Percentage** |
| ALUs | 0.32 | 17 |
| Interconnect | 0.14 | 7 |
| Register files | 0.09 | 5 |
| Memories | 0.51 | 29 |
| AGUs | 0.09 | 5 |
| Configuration registers | 0.28 | 15 |
| Decoders | 0.33 | 18 |
| Sequencer | 0.07 | 4 |
| **Total area** | 1.83 | 100 |

src: [HSM04]

# Results: Energy requirements

| Algorithm | Energy [nJ] | | | |
|---|---|---|---|---|
| | Configure | Load | Execute | Retrieve |
| FFT64 | 183 | 7 | 111 | 7 |
| FFT1024 | 276 | 116 | 2964 | 104 |
| FIR5 | 47 | 76 | 193 | 77 |
| FIR20 | 105 | 77 | 864 | 78 |

| Function | Energy[nJ] | Percentage |
|---|---|---|
| Computation | 996 | 34 |
| Storage | 839 | 28 |
| Interconnect | 339 | 11 |
| Control | 319 | 11 |
| Clock | 468 | 16 |
| Total | 2964 | 100 |

src: [HSM04]

# Chameleon/Montium Summary

- Heterogeneous System on Chip with on-chip network
  - GPP, FPGA, domain-specific reconfigurable processor (Montium), ASIC
- Montium Processing Tile
  - Optimized for application kernels
  - 5 Processing Parts with ALUs, memories, registers etc.
- Sequencer to control the execution
  - Hierarchical Decoder
- Interface to external communication (i.e. to on-chip network)
- Typical problem of coarse-grained reconfigurable fabrics: compiler/tool-chain
  - Most kernels hand mapped
  - In the scope of the startup company, some compiler exists

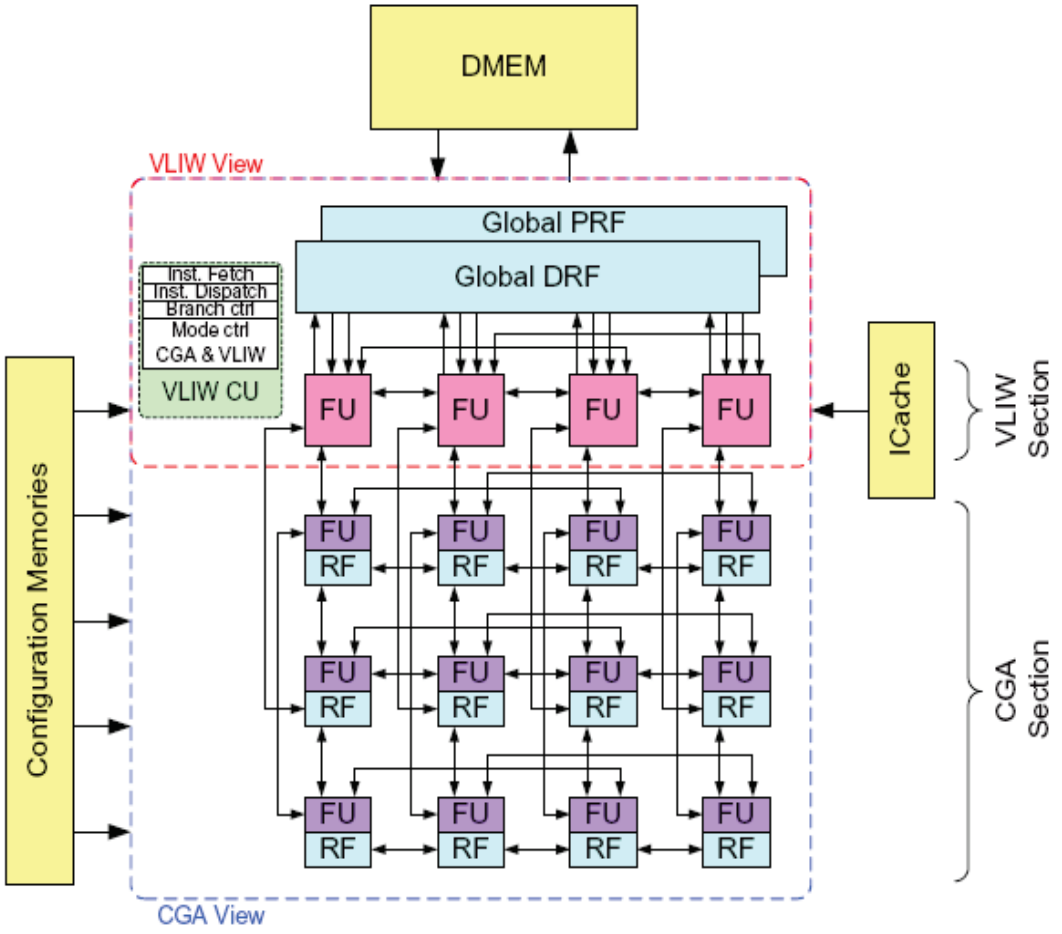
# 6.2 ADRES
## (Architecture for Dynamically Reconfigurable Embedded Systems)

# ADRES Overview

- Developed by IMEC and University of Leuven, Belgium
  - Fabricated and offered under License by IMEC
  - E.g. Toshiba is using it for their products
  - Also used for IMEC products, e.g. the Flexible-Air-Interface (FLAI) that uses 2 ADRES cores together with an ASIP, an ARM core, and further components
- Tight coupling of reconfigurable fabric with core processor
- 2D array of reconfigurable functional units
- Design-space exploration of different architectures
  - Configurable hardware template, using an XML-based architecture description language (ADL) to define communication topology, supported operations etc.
- Retargetable Compiler Framework

# ADRES Architecture

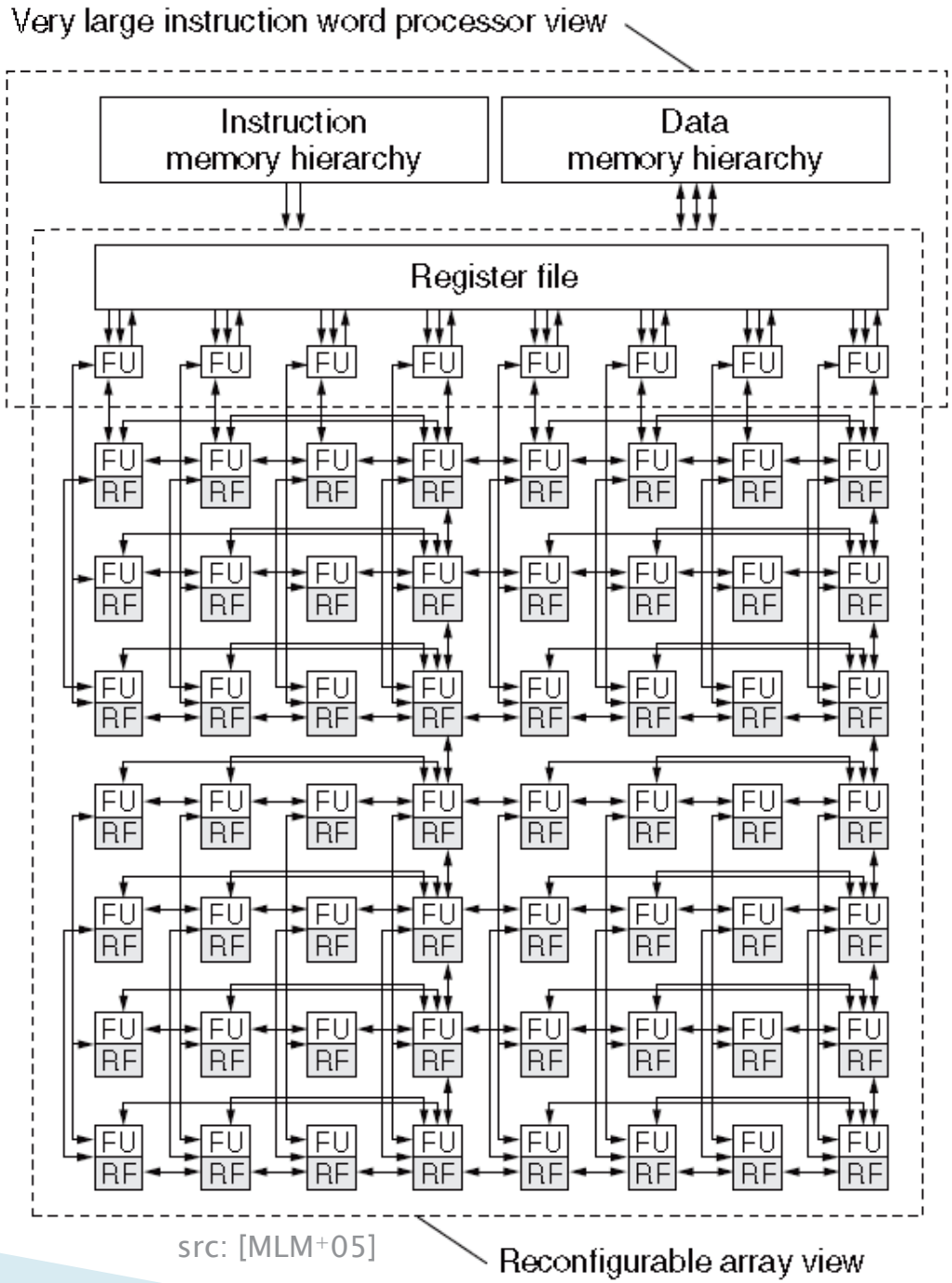
- Tightly coupled VLIW core and coarse-grained reconfigurable fabric
  - VLIW: execute sequential code and control code
  - Reconfigurable fabric: execute hot spots
  - First FU row is used to implement VLIW but is also used as part of the reconfigurable fabric
- Reconfigurable Fabric:
  - FU – Function Unit
  - RF – Register File




src: [WKMB07]

# ADRES Architecture (cont'd)

- Different instantiation of the same architecture template
- The width of the array determines the number of issue slots for the VLIW mode (can be used to adapt to different degrees of Instruction-level parallelism)
- The height is independent of the VLIW mode and only depends on the requirements of the expected SIs




src: [MLM+05]

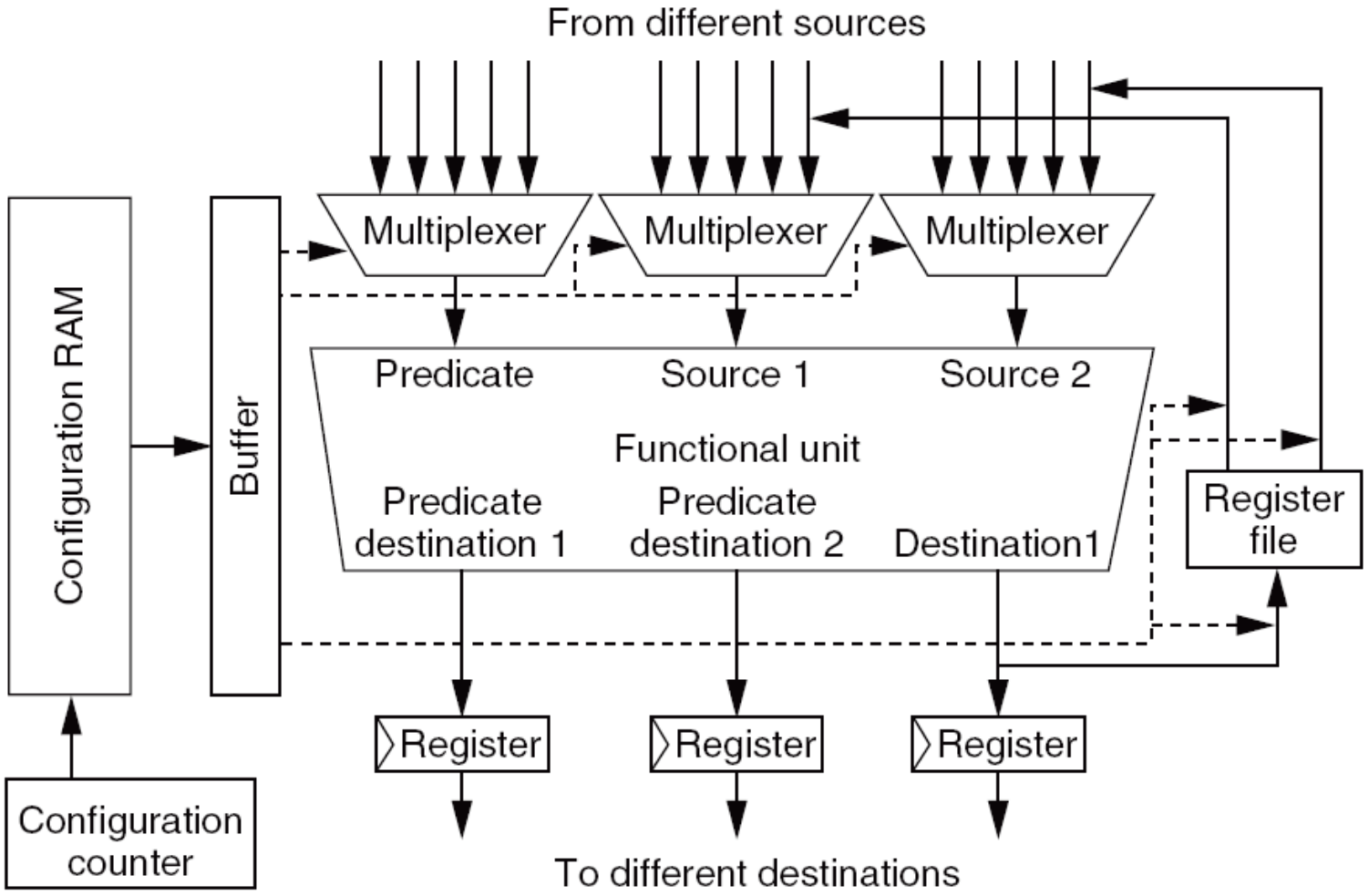
# ADRES Architecture (cont'd)

- Tight integration of VLIW mode with reconfigurable fabric
  - Reduced communication cost
  - Substantial resource sharing
  - Simplified programming model
  - Improved Performance
- Execution of VLIW mode and SI mode (executing on the reconfigurable fabric) never overlaps (e.g. first finish VLIW instruction, then start SI)
- The FUs for the VLIW mode are more powerful
  - Support branch operations
  - Connected to memory hierarchy

# Reconfigurable Cells

- The FUs that are dedicated to SI execution (i.e. excluding those for VLIW execution) are called Reconfigurable Cells (RCs)
  - They comprise of a FU and a register files (RF)
- To remove control flow inside loops that are executed by SIs, the FUs support predicated operations
- Each RC contains a small configuration RAM that stores a few configurations locally
  - Reconfiguration from this configuration RAM can be performed on a cycle-by-cycle basis

# Reconfigurable Cells (cont'd)




src: [MLM+05]

# Design-space Exploration

- Which architecture instances perform well?
  - Good performance
  - Small Area
  - Few configuration bits
  - Altogether: good trade-off
- Examine different architecture instances
  - Requires a description possibility for these instances
  - Requires a compiler to generate code for them

# Design-space Exploration (cont'd)

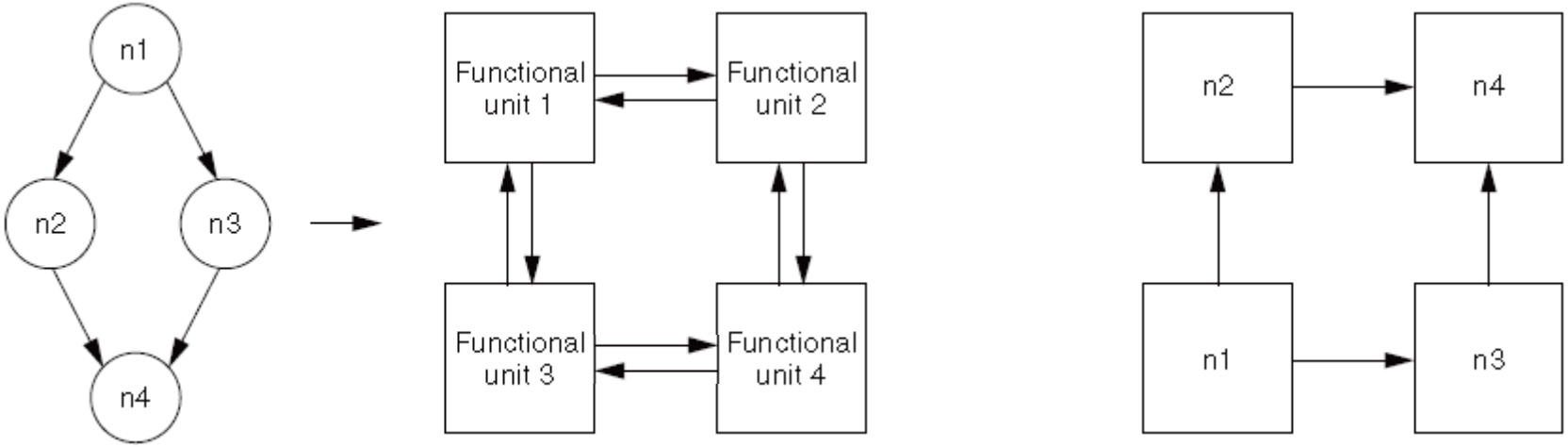
- XML-based architecture description
- DRESC: Dynamically Reconfigurable Embedded System Compiler
  - Developed together with/even before ADRES
  - Uses if-conversion and hyperblock construction
  - Supports while-loops and for-loops
    - Only innermost loops
    - Containing conditional statements
    - Not containing function calls or break/continue statements

# Design-space Exploration (cont'd)

- Compiler targets loop-level parallelism
  - Relies on so-called modulo scheduling
- Benchmark criteria
  - Instructions per Cycle (IPC)
  - Initiation Interval (II)
    - The number of cycles that have to elapse until the next iteration of the kernel loop can start execution
    - Idea: Pipeline the innermost loop
    - Additionally: Prolog and Epilog for initialization and finalization, respectively

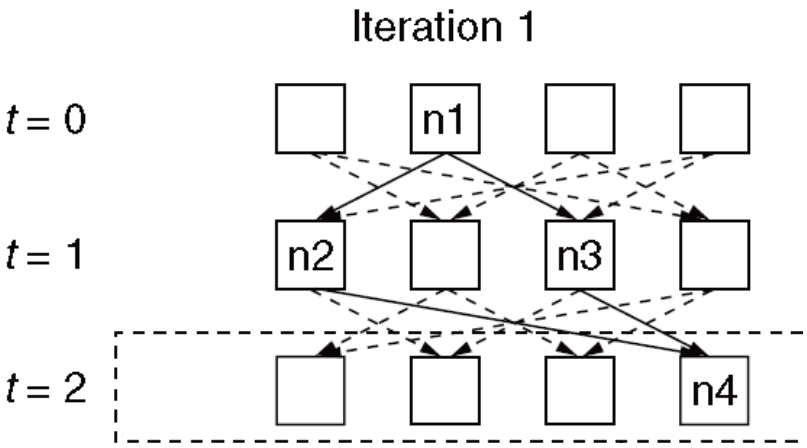
# Example for Initiation Interval

- Given a data-flow graph (i.e. the loop body) and an architecture description
- Perform a mapping of the graph to the architecture




src: [MLM+05]

# Example for Initiation Interval (cont'd)



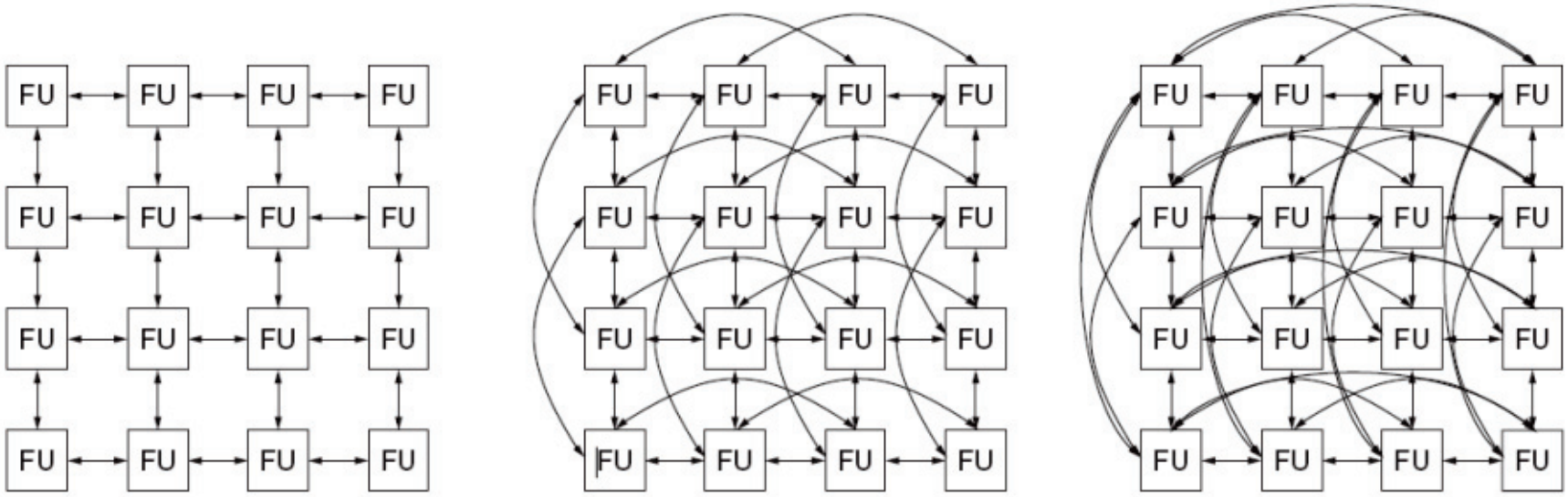

- Depending on the resource conflicts, it might not be possible to perform two subsequent iterations of the loop right after each other

src: [MLM+05]

# Design-space Exploration

- Application kernels from Multimedia and DSP domain
  - idct1, idct2: Inverse Discrete Cosine Transformation
  - get_block1,2,3: Function of the AVC Decoders
  - mimo_mmse: MIMO minimum mean square error
  - mimo_matrix: MIMO matrix calculation
  - fft: Fast Fourier Transformation
- Investigated Parameters
  - Connections
  - Functional Units
  - Memories
  - Register Files

# Connection Topology




(a) Mesh (b) Meshplus (c) MorphoSys

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

src: [MLM+05]

# Connection Topology (cont'd)

| | Mesh | | | Meshplus | | | MorphoSys | | |
|---|---|---|---|---|---|---|---|---|---|
| Kernel | II | IPC | Overuse (II – 1) | II | IPC | Overuse (II – 1) | II | IPC | Overuse (II – 1) |
| idct1 | 3 | 26.3 | 19 | 3 | 26.3 | 10 | 3 | 26.3 | 17 |
| idct2 | 4 | 32.0 | 4 | 3 | 42.7 | 68 | 3 | 42.7 | 71 |
| get_block1 | 3 | 19.0 | 4 | 3 | 19.0 | 1 | 3 | 19.0 | 1 |
| get_block2 | 3 | 28.7 | 17 | 3 | 28.7 | 5 | 3 | 28.7 | 7 |
| get_block3 | 3 | 21.7 | 2 | 2 | 32.5 | 65 | 2 | 32.5 | 61 |
| mimo_mmse | 7 | 30.4 | 1 | 6 | 35.5 | 23 | 6 | 35.5 | 13 |
| mimo_matrix | 6 | 25.8 | 3 | 5 | 31.0 | 16 | 5 | 31.0 | 23 |
| fft | 4 | 19.8 | 9 | 4 | 19.8 | 3 | 4 | 19.8 | 1 |

- Mesh performs worst
- Meshplus and Morphosys perform on a similar level (identical II)
- Note: 'Overuse' only shows how many nodes could not be scheduled when trying to reduce the reported initiation interval by one cycle

src: [MLM+05]

# Connection Topology (cont'd)

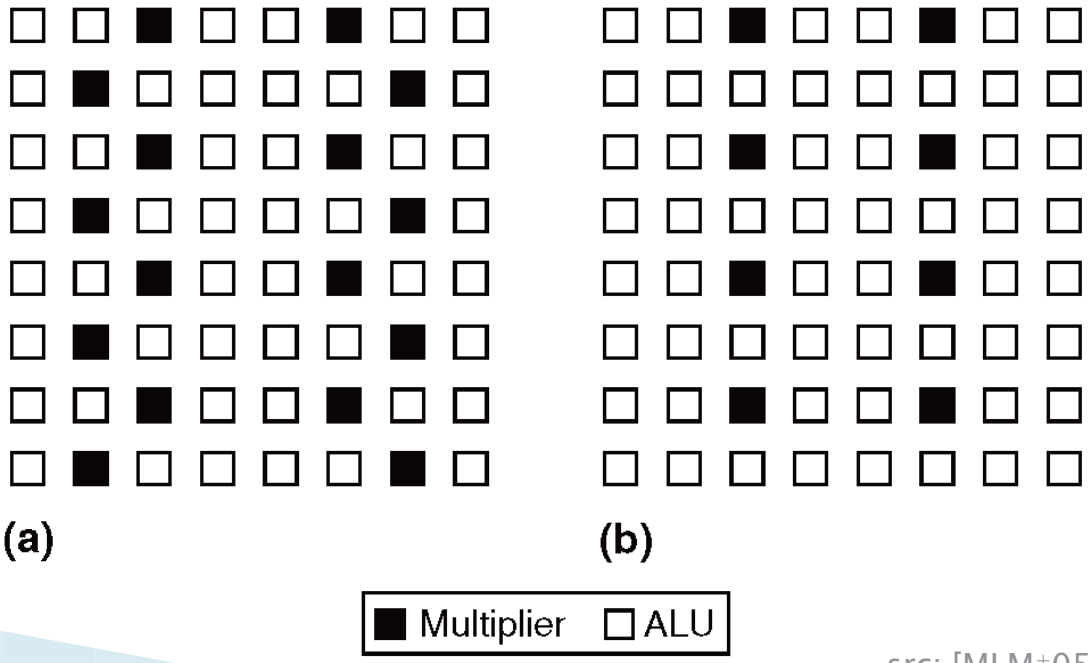
| Architecture | Total configuration bits | Multiplexer area (mm2) |
|---|---|---|
| 8mem-mesh-hete1 | 2,339 | 0.277 |
| 8mem-meshplus-hete1 | 2,415 | 0.370 |
| 8mem-morphosys-hete1 | 2,562 | 0.426 |

- The different connection topologies have a rather small impact on the required configuration bits
- But they have a large impact on the area that is required to implement the multiplexers to establish the communication
  - Synthesized for a 130 nm standard-cell library

src: [MLM+05]

# Heterogeneous Functional Units

- Two different FU types
  - Multiplier
  - ALUs: significantly cheaper (area-wise etc.)
- Not each FU needs to be able to perform a multiplication




src: [MLM+05]

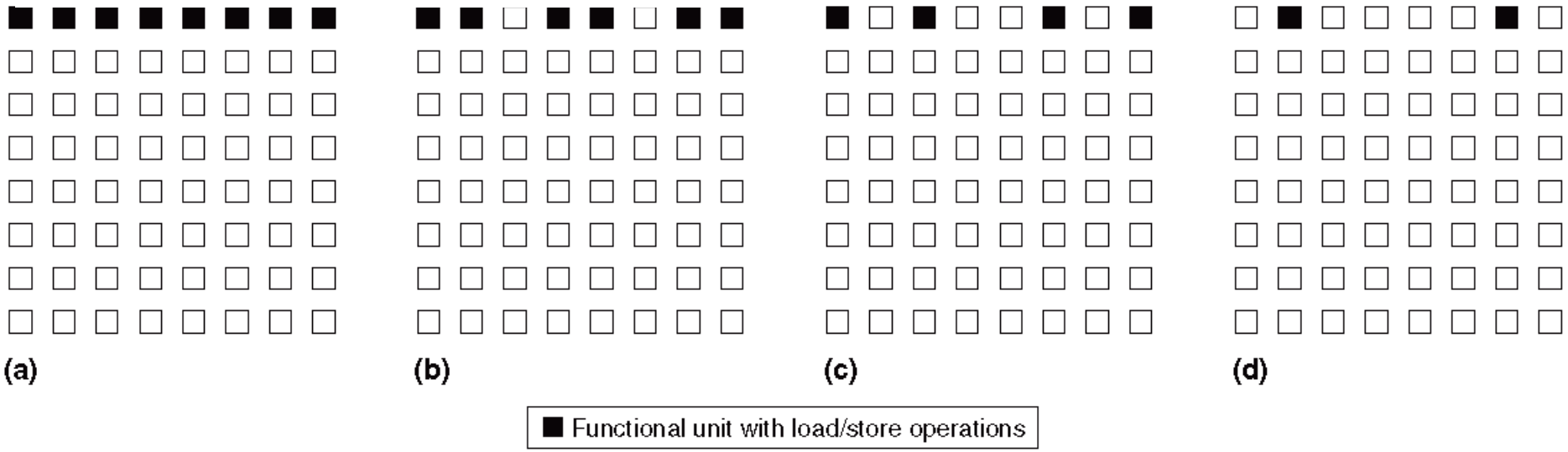
# Heterogeneous Functional Units (cont'd)

- Identical (!) results for both variants
- Rather few multiplications are performed in the benchmarked Kernels

| | Initiation interval | |
|---|---|---|
| **Kernel** | **hete1** | **hete2** |
| idct1 | 3 | 3 |
| idct2 | 3 | 3 |
| get_block1 | 3 | 3 |
| get_block2 | 3 | 3 |
| get_block3 | 2 | 2 |
| mimo_mmse | 6 | 6 |
| mimo_matrix | 5 | 5 |
| fft | 4 | 4 |

src: [MLM+05]

# Memory Ports




- Some of the VLIW FUs have access to the main memory
- The number and placement of these memory ports is to be investigated

src: [MLM+05]

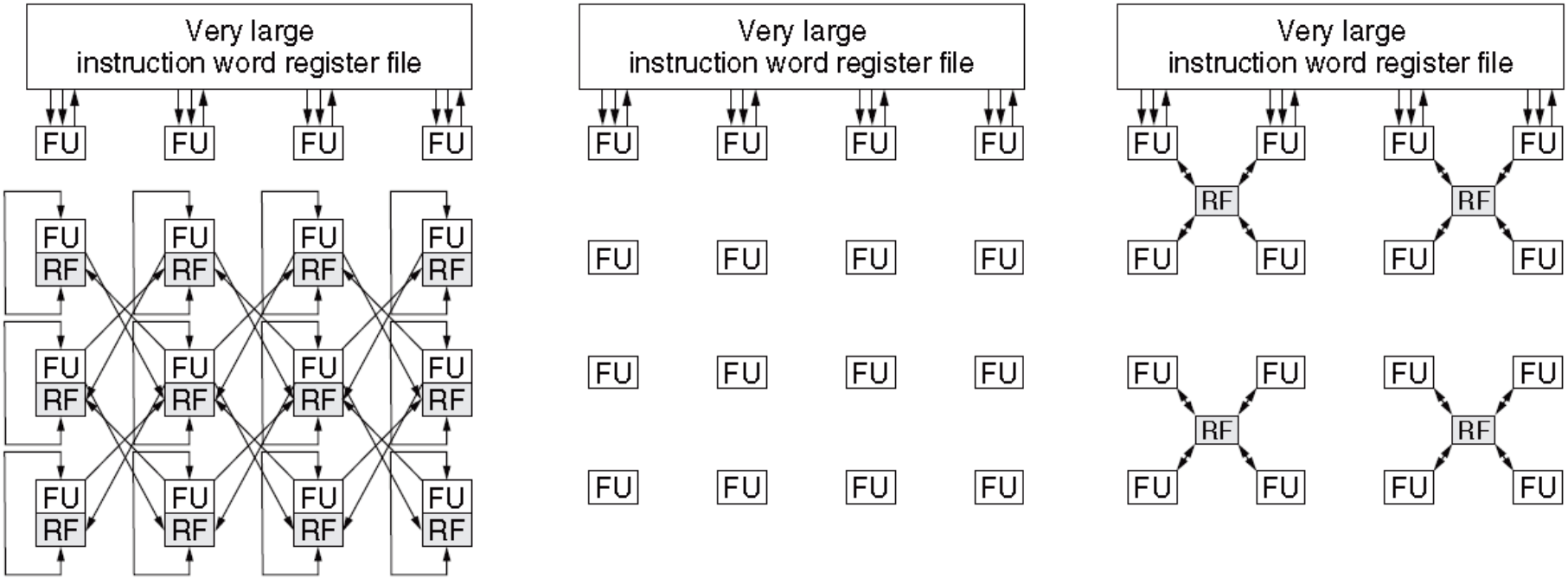
# Memory Ports (cont'd)

- The ADRES instance with 8 memory ports results in the best performance
- Large ADRES instances (i.e. large amount of FUs for the reconfigurable fabric) with rather few memory ports are inefficient

| | Initiation interval | | | |
|---|---|---|---|---|
| **Kernel** | **8mem** | **6mem** | **4mem** | **2mem** |
| idct1 | 3 | 3 | 4 | 8 |
| idct2 | 3 | 3 | 4 | 8 |
| get_block1 | 3 | 3 | 3 | 4 |
| get_block2 | 3 | 3 | 3 | 4 |
| get_block3 | 2 | 3 | 4 | 7 |
| mimo_mmse | 6 | 6 | 7 | 8 |
| mimo_matrix | 5 | 5 | 6 | 8 |
| fft | 4 | 4 | 6 | 11 |

src: [MLM+05]

# Distributed Register File




- Observation: typically only 20 % of the available register file ports are actually accessed per cycle
- Extreme design: no register files (except for VLIW FUs)
- Alternative design: some distributed register files

src: [MLM+05]

# Distributed Register File (cont'd)

| Architecture | Total configuration bits | 2R1W 32×8 RFs | Multiplexer area (mm2) |
|---|---|---|---|
| 8mem-meshplus-hete1 | 2,415 | 56 | 0.370 |
| 8mem-meshplus2-hete1 | 1,337 | 0 | 0.231 |
| 8mem-meshplus3-hete1 | 1,701 | 16 | 0.313 |

- Significant difference for the configuration data
- Also high difference for the required area
  - Note: only the area for the multiplexers is shown. Additionally, the area for the actual register files and the configuration bits needs to be considered

# Distributed Register File (cont'd)

| | 1 regs per FU | | 0 regs per FU | | 0.25 regs per FU | |
|---|---|---|---|---|---|---|
| | Meshplus | | Meshplus2 | | Meshplus3 | |
| Kernel | II | Overuse (II – 1) | II | Overuse (II – 1) | II | Overuse (II – 1) |
| idct1 | 3 | 10 | 3 | 32 | 3 | 11 |
| idct2 | 3 | 68 | 5 | 1 | 4 | 2 |
| get_block1 | 3 | 1 | 3 | 5 | 3 | 2 |
| get_block2 | 3 | 5 | 3 | 46 | 3 | 13 |
| get_block3 | 2 | 65 | 3 | 5 | 2 | 67 |
| mimo_mmse | 6 | 23 | Failed | NA | 7 | 27 |
| mimo_matrix | 5 | 16 | 7 | 15 | 6 | 1 |
| fft | 4 | 3 | 4 | 18 | 4 | 3 |

- Without register files, not all applications can be scheduled
- With distributed register files, only 3 of 7 applications have a worse II

src: [MLM+05]

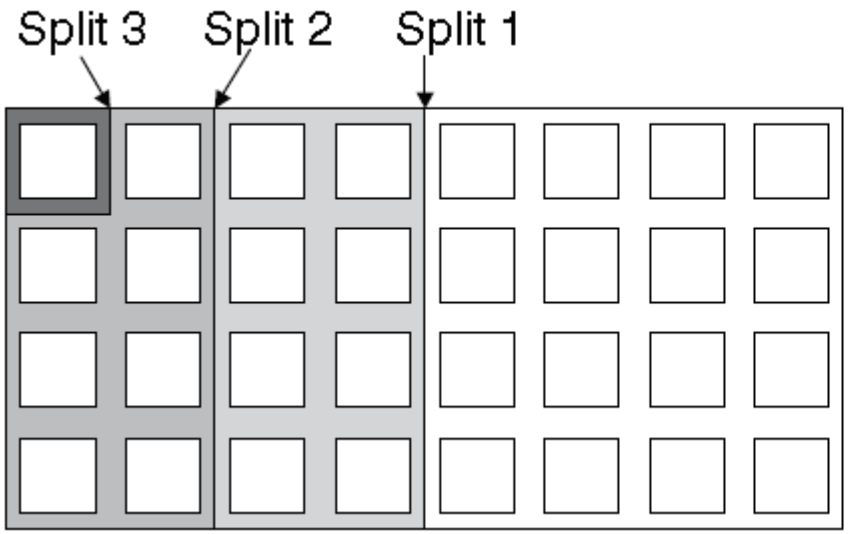
# 6.3 Multithreaded ADRES (MT-ADRES)

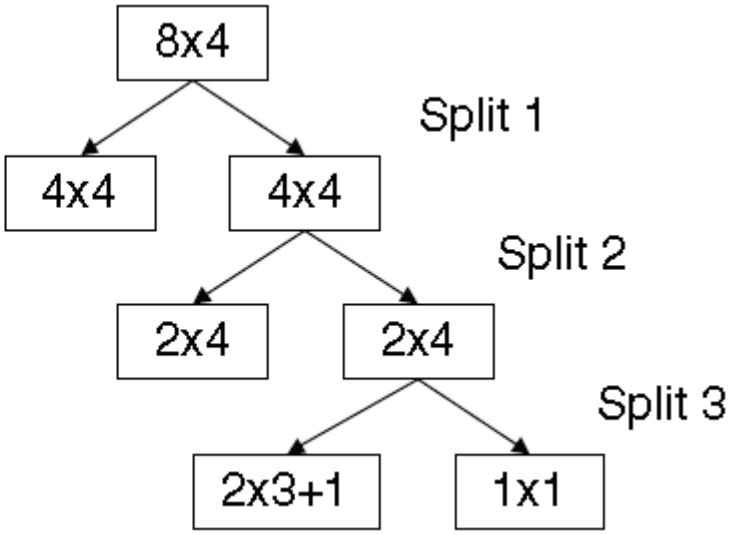
# Motivation

- Some powerful ADRES instances can execute 64 operations per cycle
- But only few applications can exploit that potential by using loop-level parallelism (**LLP**) and instruction-level parallelism (**ILP**)
- Idea: also exploit thread-level parallelism (**TLP**)
  - One multi-threaded application
  - Threads run in parallel on the ADRES instance
  - Need to share the available hardware

# Partitioning the Array

- Idea: split the array into two sub-arrays
  - Can be split again into sub-sub-arrays etc.
- Each sub*-array can be seen as a normal ADRES instance
  - Each sub*-array executes a single thread
  - The threads in the different sub*-arrays execute in parallel
- Same-sized sub-arrays may contain different resources

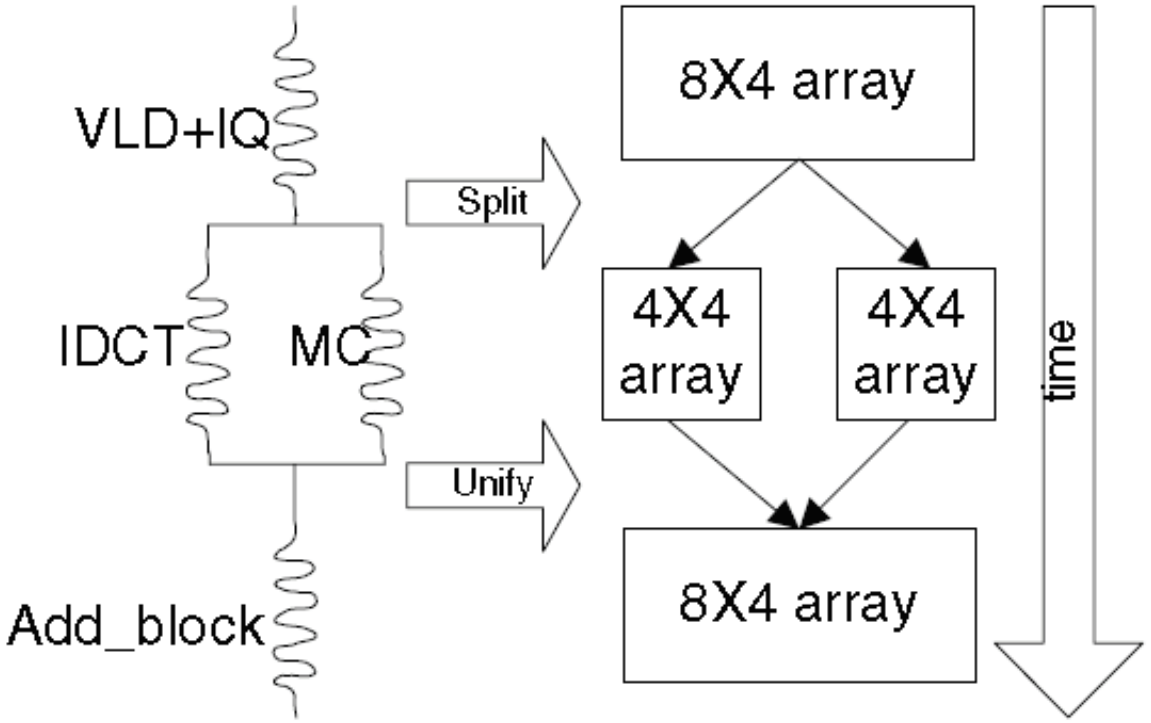



src: [WKMB07]

# Example: MPEG-2 Decoder

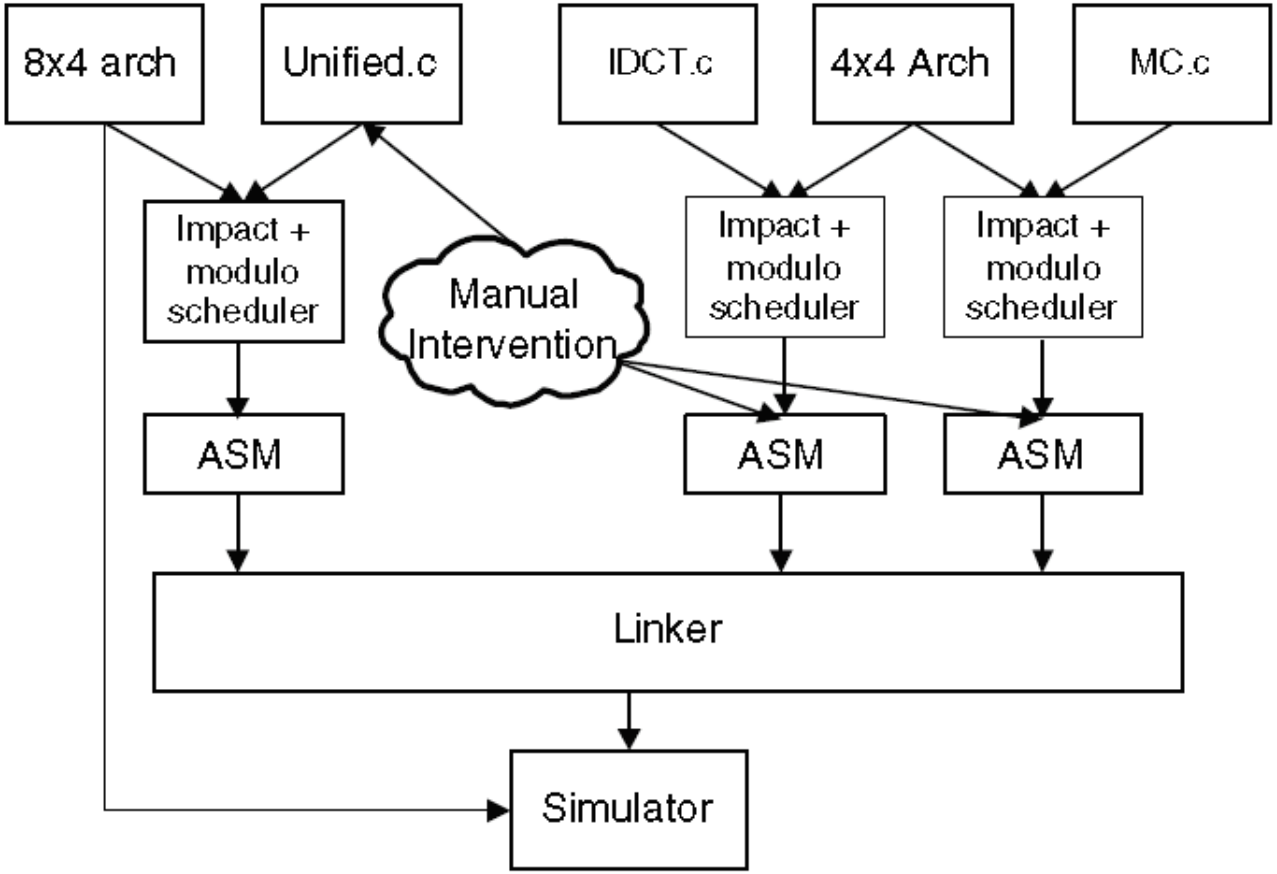
- Starting with a 8x4 array
- Splitting into two 4x4 arrays to execute 2 threads in parallel
- After their execution completed, unifying to a 8x4 array again
  - Execution stalls until all parallel threads are completed
  - Same if a 4x4 thread would be further subdivided
- Statically prepared at compile time




src: [WKMB07]

# Toolflow

- Partition the Application into multiple C-code files
  - Separate commonly used headers
- Individual threads are compiled for their particu-lar ADRES instance
- Some manual work is still required




src: [WKMB07]

# ADRES Summary

- VLIW view and reconfigurable array view
  - Sharing a 'row' of reconfigurable FUs
  - Exploiting ILP and LLP, respectively
- Array can be split into sub-arrays to exploit TLP additionally
  - Only 12-15% speedup reported for Multithreading
  - More speedup is expected for other applications
- XML-based architecture description with automatic hardware and compiler generation
  - Allows huge design-space exploration
  - Trading-off performance and area
- Developed by R&D industry (IMEC), used in own products, licensed by Samsung, Toshiba and others
  - For instance, the Samsung Reconfigurable Processor (SRP) is "a variation of ADRES processor"; src: "ULP-SRP: Ultra Low Power Samsung Reconfigurable Processor for Biomedical Applications", FPT 2012

# 6.4 PipeRench

# PipeRench Overview

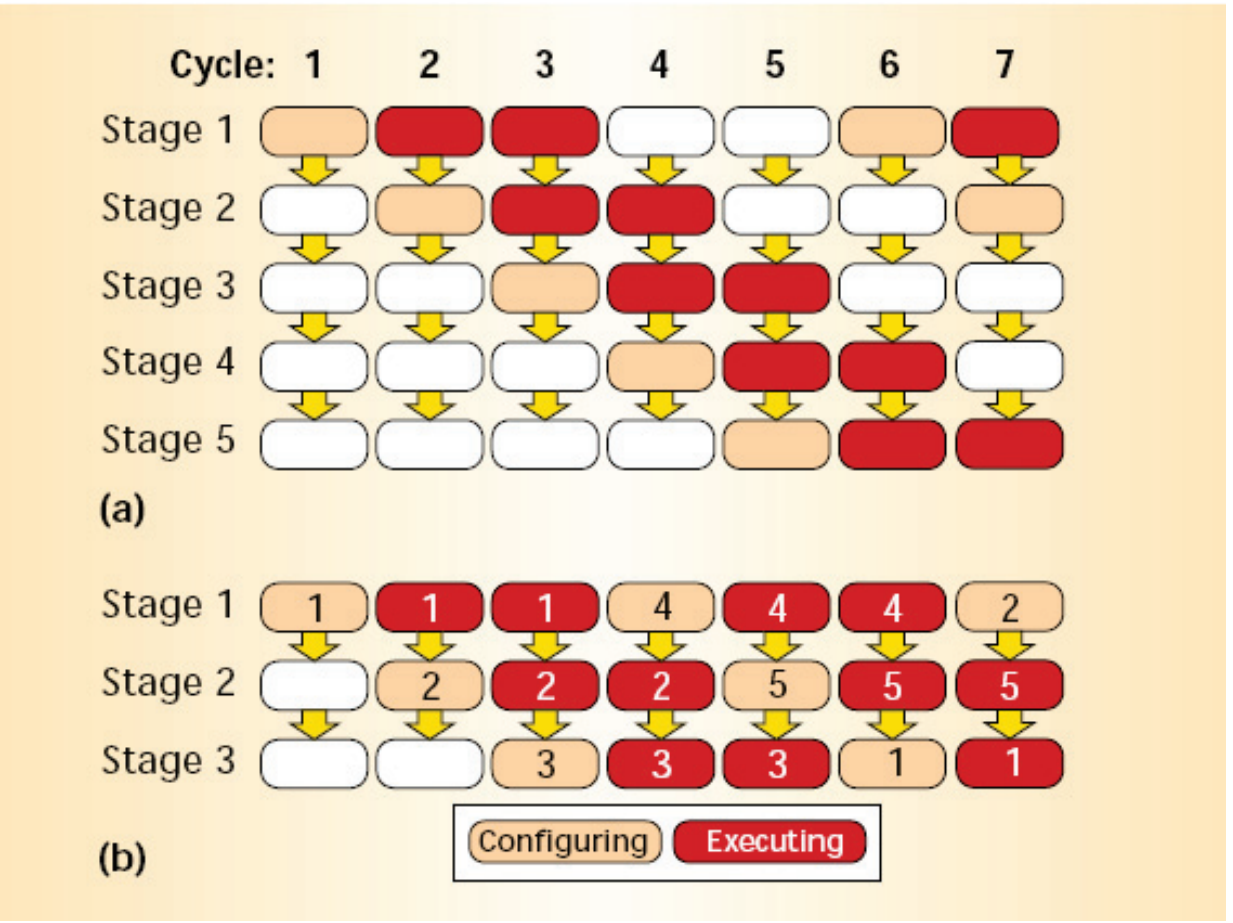
- Co-processor for data-stream applications
- Fast partial and dynamic reconfiguration
- Pipelined Reconfiguration
- Run-time scheduling of configuration and data streams
  - Maximize HW utilization
- Virtualizes the reconfigurable fabric
  - Can upgrade to larger reconfigurable fabrics without recompiling

# Pipelined Reconfiguration

- Observed problems:
  - A computation may require more hardware than is available
  - A hardware design cannot exploit the additional resources that become available in future process generations
- Solution: Pipelined reconfiguration
  - Implements a large logical configuration on a small piece of hardware
  - Breaks a single static configuration into pieces that correspond to pipeline stages in the application
  - Each pipeline stage is loaded, one per cycle, into the fabric (virtualization)
  - This makes performing the computation possible, even if the entire configuration is never present in the fabric at the same time

# Pipelined Reconfiguration (cont'd)

- Example: An SI that requires 5 pipeline stages and always processes two data packets after each other executes on a reconfigurable fabric with
  a) 5 pipeline stages
  b) 3 pipeline stages
- Constraints:
  - At most one reconf. per cycle
  - Single-cycle reconfiguration
- Note: This is just an example; obviously the 5-stage HW is not utilized efficiently



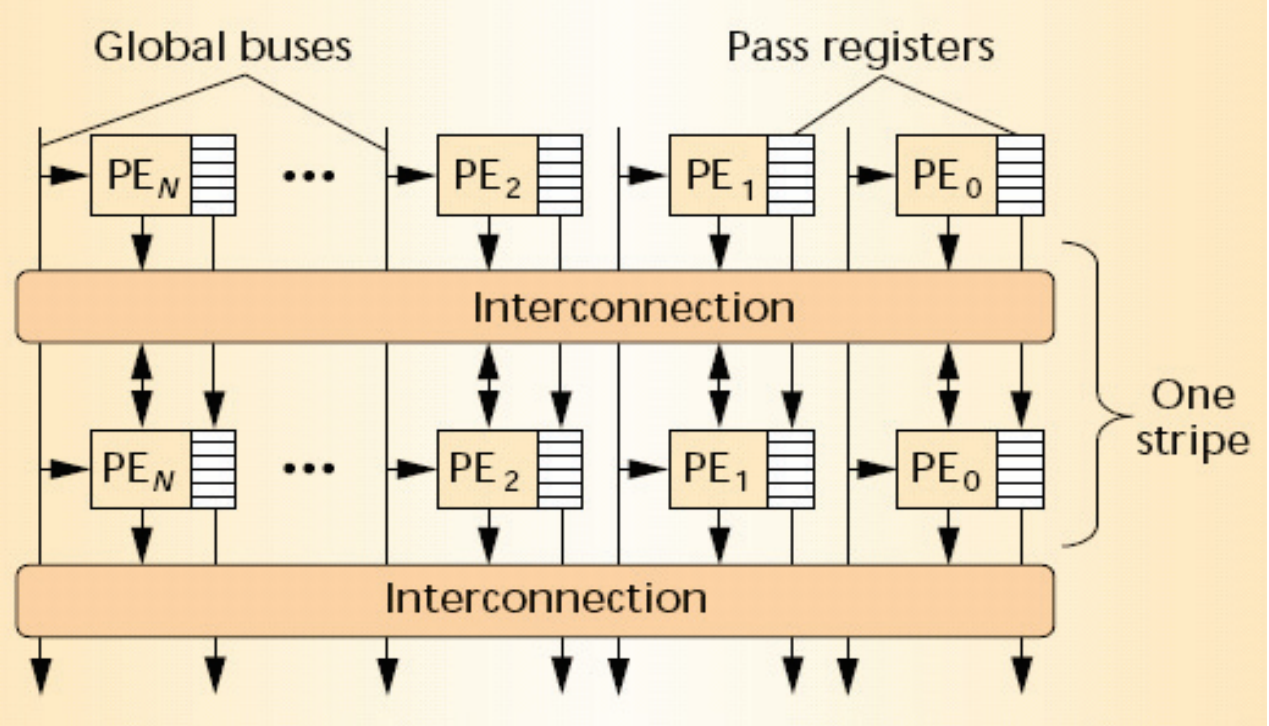

src: [GSB+00]

# Pipelined Reconfiguration (cont'd)

- Constraints for the SIs:
  - The state in any pipeline stage must be a function of only that stage's and the previous stage's current state
    - i.e., cyclic dependencies must fit within one pipeline stage
  - No communication that skips over one or more stages (only to the directly succeeding stage)
  - No communication from a stage to a previous stage
- A pipeline stage contains multiple PEs
  - PE input either from the registered outputs of previous stage or from the registered or unregistered output of other PEs of the same stage

# PipeRench Architecture

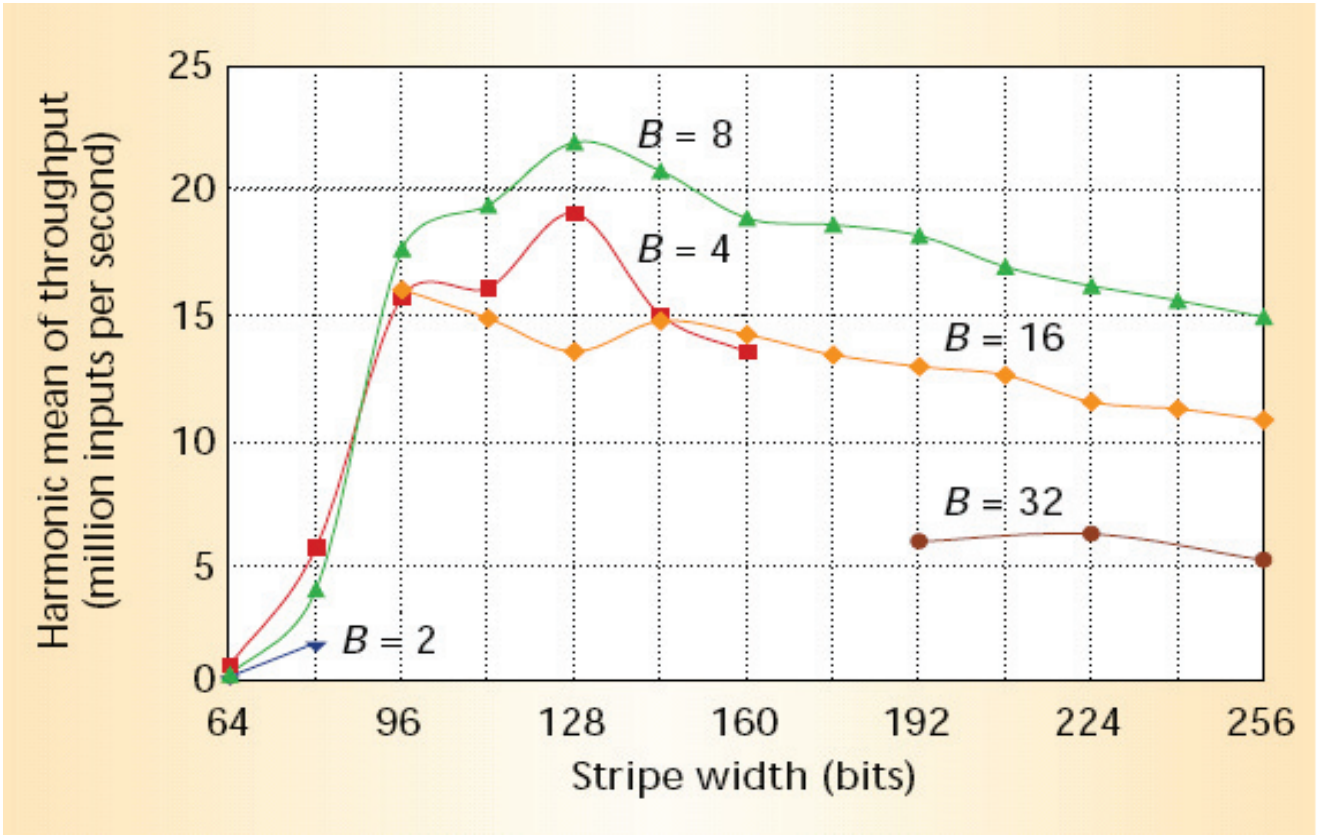
- Each PE contains an ALU, barrel shifters, extra circuitry to implement carry chains and zero-detection, and registers
- The ALU is 8-bits wide (can be extended using multiple ALUs and the carry chains
- Global Busses are used to provide SI input/output to the PEs
- Pass registers establish the communi-cation bet-ween the pipe-line stages




src: [GSB+00]

# Comparing different Bit width

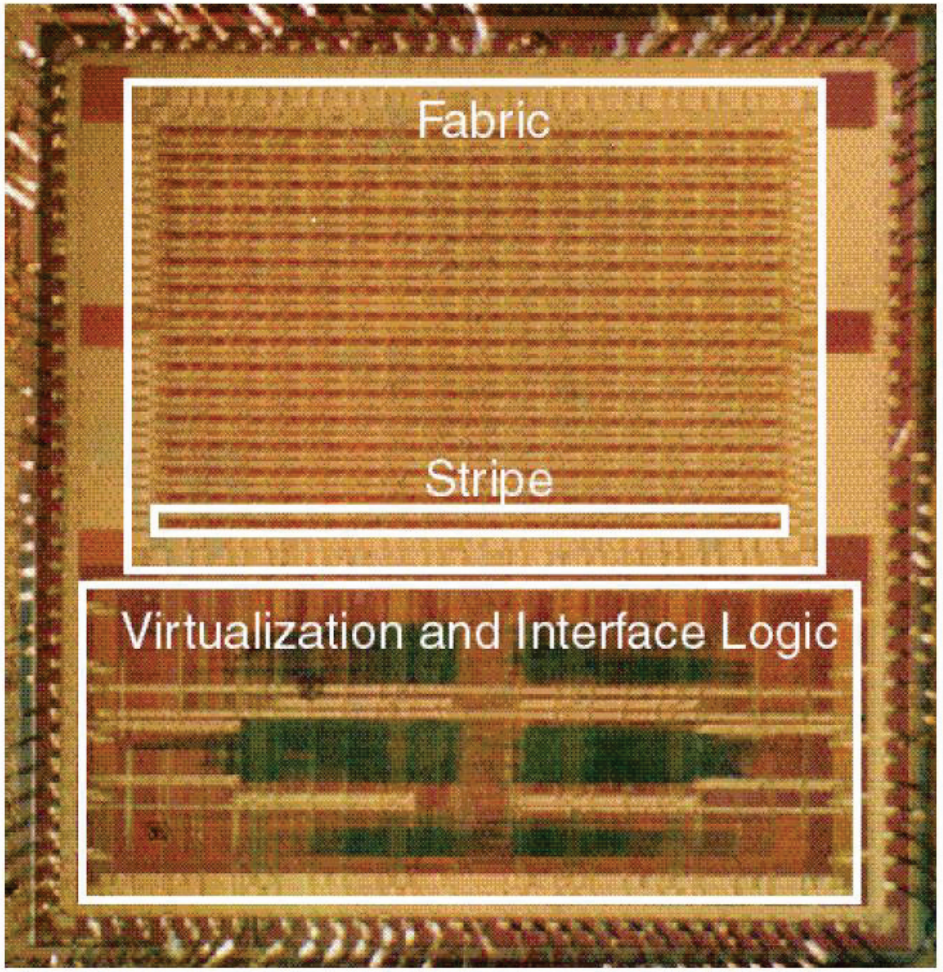
- Benchmark for 16-bit based SI operations
- 'B' denotes the bit width of the PE, ALU etc.
- 2-bit PEs are not competitive and 32-bit PEs are underused
- The higher flexibility of 8-bit PEs outperforms the 16-bit PEs




src: [GSB+00]

# Implementation

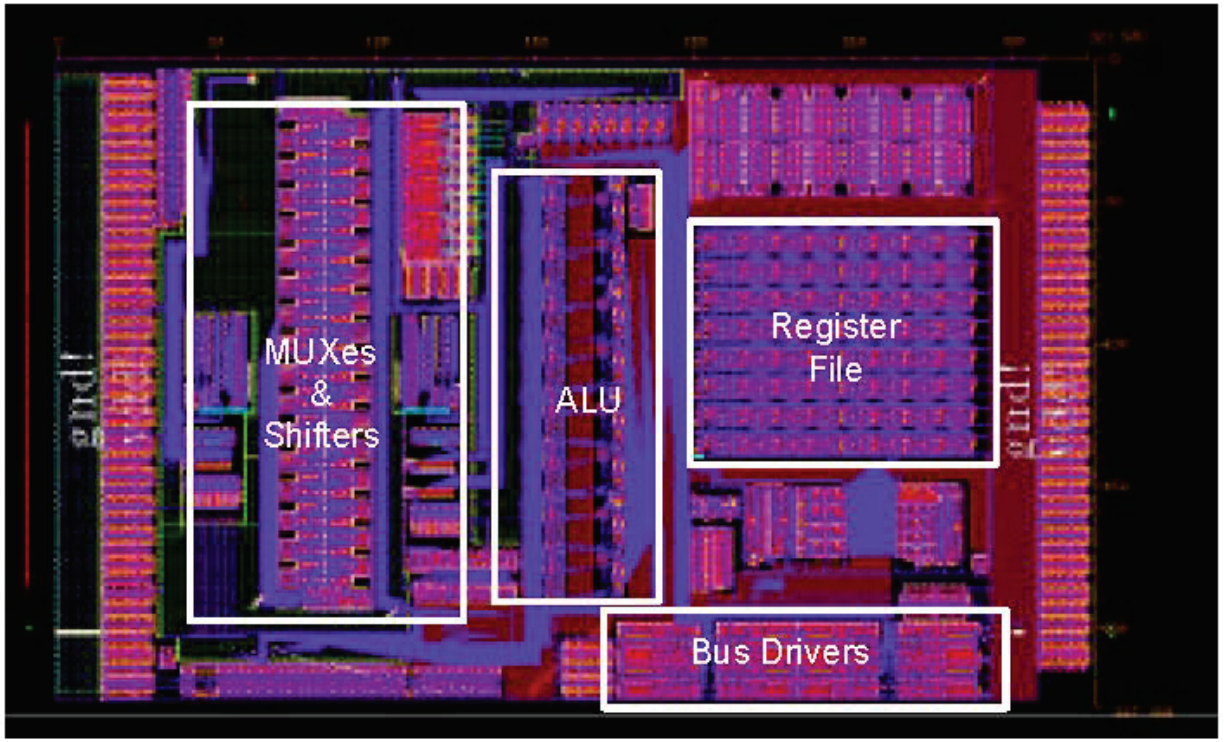
- Fabricated as ST Microelectronics six-metal layer 180 nm CMOS process
- die area is 7.3 x 7.6 mm$^2$
- 3.65 million transistors
- 16 pipeline stages
- Virtualization storage for 256 virtual stages
- 120 MHz frequency




src: [SWT+02]

# PE Implementation

- PE size: 325 µm x 225 µm
- 16 of these PEs form a pipeline stage
- Area is dominated by interconnect resources such as multiplexers and bus drivers
- The dimensions of the PE layout are dictated by the interconnect to other PEs in the stripe, and by the global busses, which run vertically over the PE cell




src: [SWT+02]

# PipeRench Summary

- Co-Processor that allows virtualization via pipelined reconfiguration
- Allows upgrading to larger PipeRench instances without recompiling
- Up to 190x faster kernels (not full applications) compared to GPP
- Created startup company Rapport (does not exist any more) that produced the KiloCore chip

# References and Sources


[HSM03] P. Heysters, G. Smit, E. Molenkamp: “A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems”, Journal of Supercomputing, vol. 26, pp. 283-308, 2003.

[H04] P. M. Heysters: “Coarse-grained reconfigurable processors – Flexibility meets efficiency," Ph.D. dissertation, Dept. Comput. Sci., Univ. Twente, Enschede, The Netherlands, 2004.

[HSM04] P. M. Heysters, G. J. M. Smit, E. Molenkamp: “Energy-Efficiency of the Montium Reconfigurable Tile Processor”, International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), pp. 38-44, 2004.

[WKMB07] K. Wu, A. Kanstein, J. Madsen, M. Berekovic: “MT-ADRES: Multithreading on Coarse-Grained Reconfigurable Architecture”, International Workshop on Applied Reconfigurable Computing (ARC), pp. 26-38, 2007.

[MLM+05] B. Mei, A. Lambrechts, J. Mignolet, D. Verkest, R. Lauwereins: “Architecture Exploration for a Reconfigurable Architecture Template”, IEEE Design & Test of Computers, vol. 22, no. 2, pp. 90-101, 2005.

[GSB+00] S.C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, R. Taylor: “PipeRench: A Reconfigurable Architecture and Compiler”, IEEE Computer, vol. 33, no. 4, pp. 70-77, 2000.

[SWT+02] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, R. R. Taylor: “PipeRench: A Virtualized Programmable Datapath in 0.18 Micron Technology”, IEEE Custom Integrated Circuits Conference, pp. 63-66, 2002.