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

Vorlesung im SS 2016

Reconfigurable and Adaptive Systems (RAS)

Marvin Damschen, Lars Bauer, Jörg Henkel

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

Reconfigurable and Adaptive Systems (RAS)

4. Fine-Grained Reconfigurable Processors

RAS Topic Overview

1. Introduction
2. Overview
3. Special Instructions
4. Fine-Grained Reconfigurable Processors
5. Configuration Prefetching
6. Coarse-Grained Reconfigurable Processors
7. Adaptive Reconfigurable Processors
8. Fault-tolerance by Reconfiguration

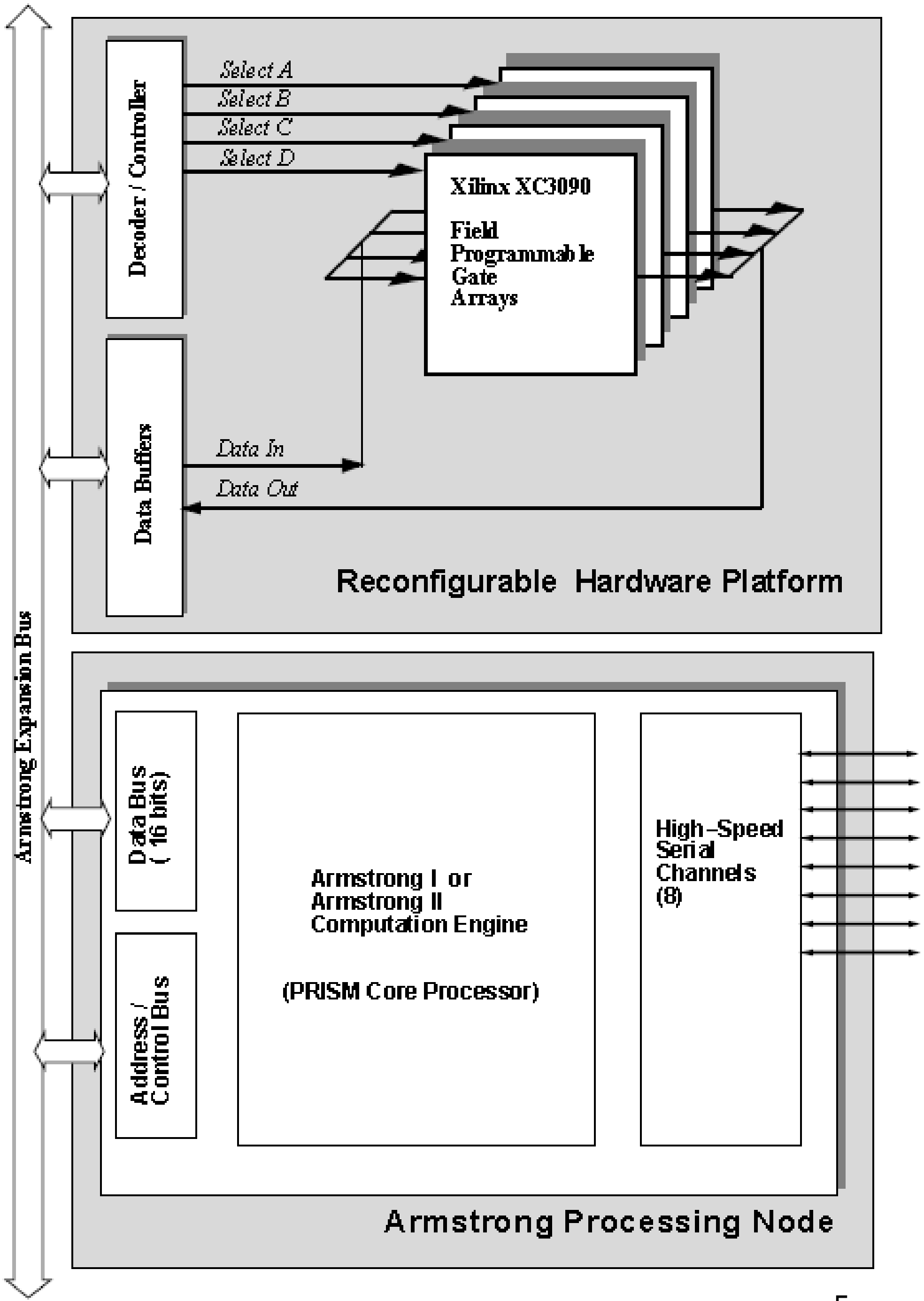
- PRISM
  - PRISM-II
- Garp
- MOLEN
- PRISC
- OneChip
  - OneChip98
- XiRISC
  - XiSystem
- New FPGA Architectures

4.1 PRISM: Processor Reconfiguration through Instruction Set Metamorphosis

PRISM Overview

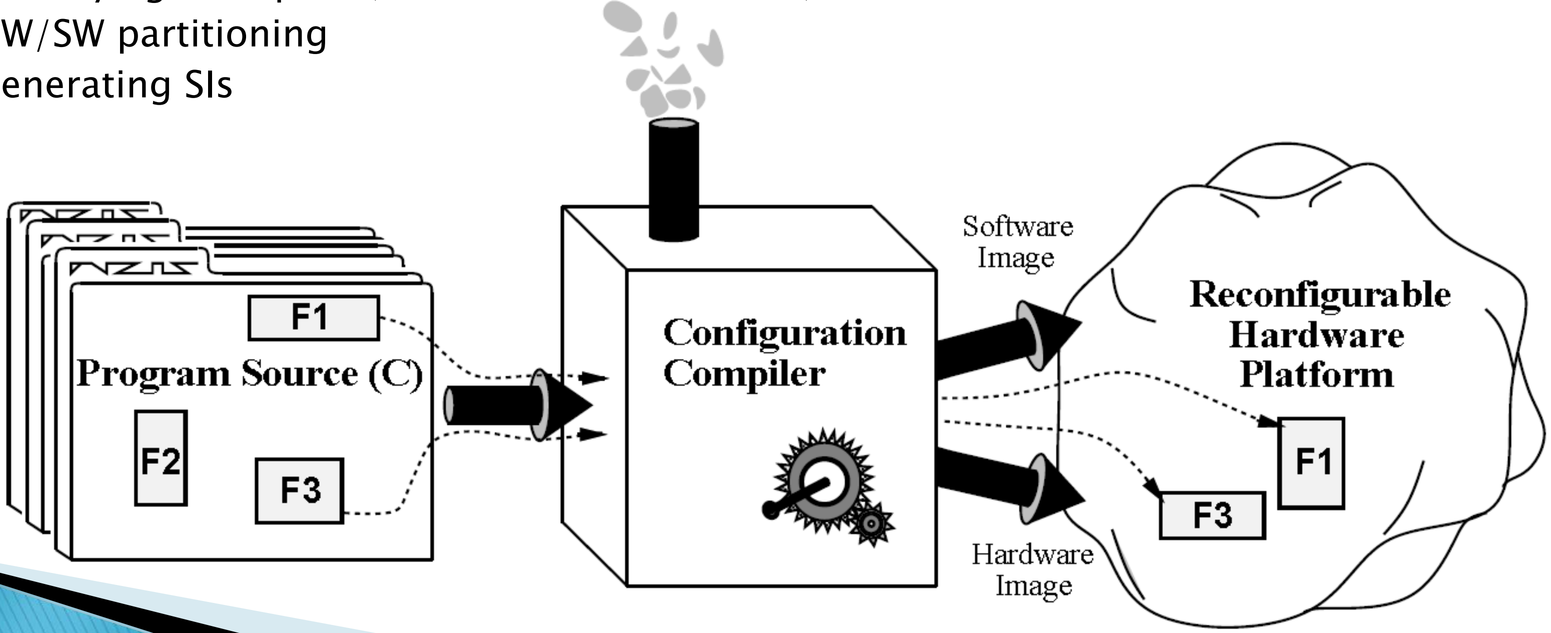
- PRISM-I system: external stand-alone processing unit
  - Two boards that are inter-connected by a 16-bit bus
  - Processor board: Motorola 68010 processor running at 10 MHz
  - Accelerator board: four Xilinx 3090 FPGAs
- Hardly run-time reconfi-gurable, i.e. it takes nearly one second to reconfigure the FPGAs

src: [WAL+93]



PRISM Tool Chain

- Observation: an adaptive micro-architecture cannot be designed by the application programmer (limited expertise)
- Solution: High Level Language compiler, so-called configuration compiler
- "The configuration compiler [...] is a special compiler that accepts a high-level language program as input, and produces both a hardware image and a software image" [WAL+93]
  - Identifying Hot spots (with manual interaction)
  - HW/SW partitioning
  - Generating SIs



src: [WAL+93]

PRISM Limitations

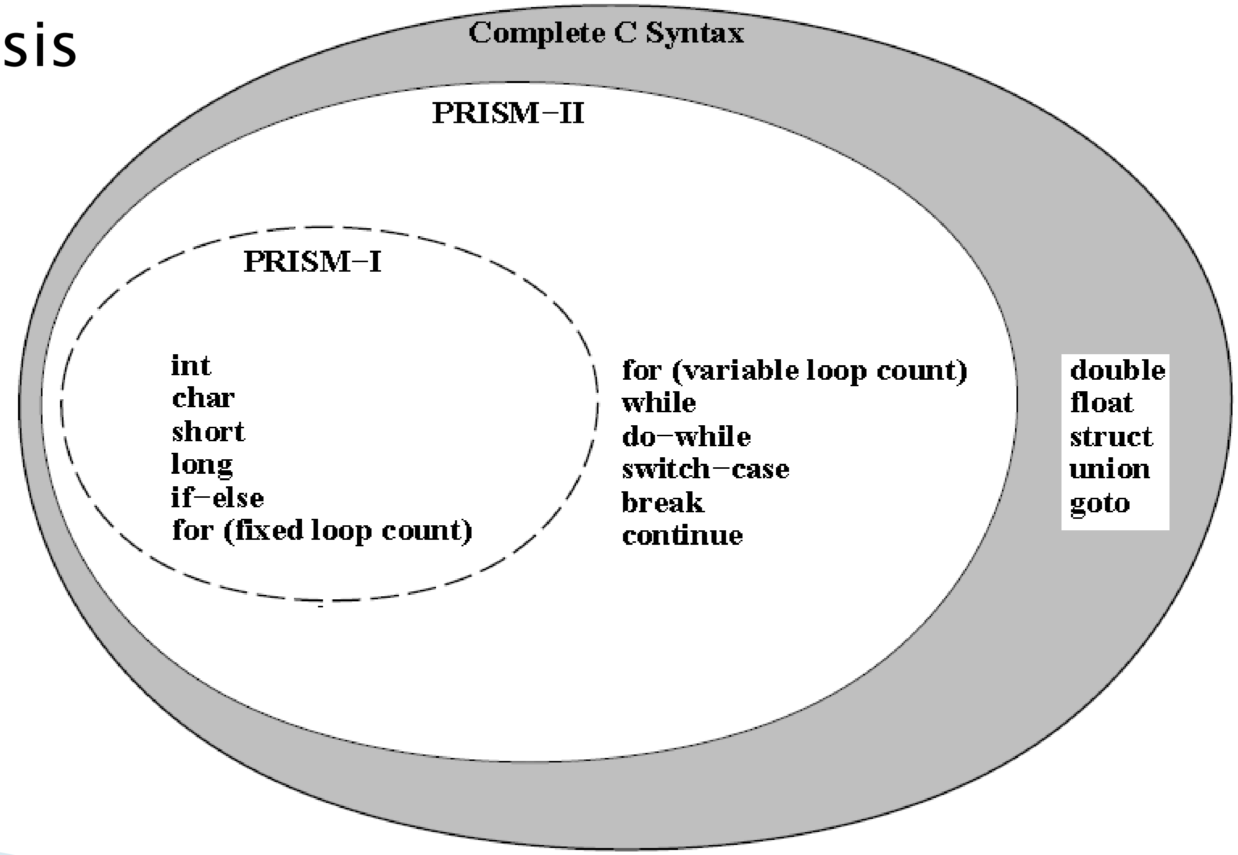
- Hardware Limitations:
  - PRISM-I is the first implementation of the PRISM concept, i.e. it is a proof-of-concept
  - Slow reconfiguration speed (nearly one second) under software control
  - FPGAs provide only a limited speed and capacity
  - Slow communication: between 45 and 75 clock cycles (at 10 MHz) to move operands to an SI and to collect the results

PRISM Limitations (cont'd)

- Tool Chain Limitations:
  - State and global variables are not supported
  - At most 32-bit input bits and 32-bit output bits respectively (may be distributed among multiple variables)
  - No support for variable loop counts (i.e. not supporting "for (i=0 to *n*)", where *n* is variable)
  - Only single-cycle SI implementations
  - Limited support for C data types (e.g. no 'float') and C constructs (e.g. no 'do-while' or 'switch-case')

4.2 PRISM-II

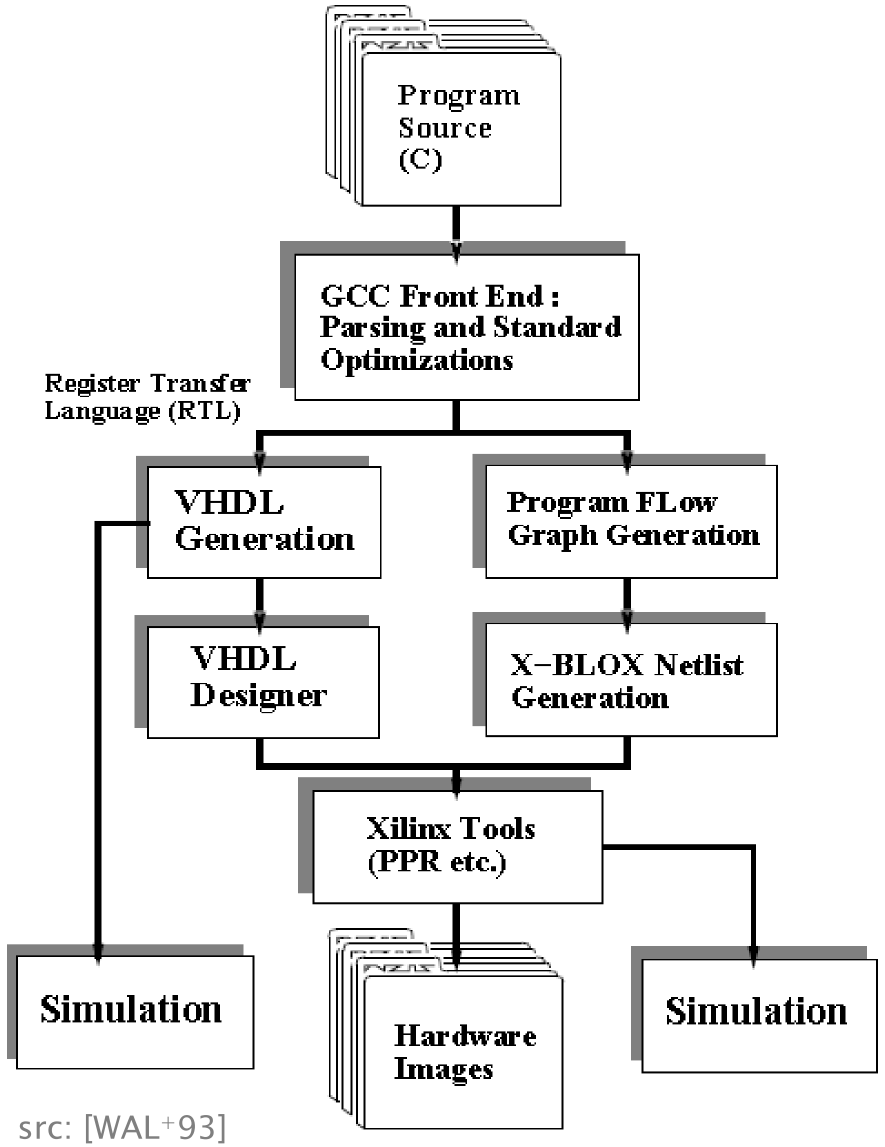
- Improved System: PRISM-II
- Supports larger parts of the C language specification
- Supports synthesis of sequential logic for execu-tion of loops with variable loop counts (i.e. unknown at compile time)



src: [WAL+93]

PRISM-II Tool Chain

- The parsing and optimization stage builds on top of GCC
  - GCC used a variation of a register transfer language at that time
- The synthesis is done using ‘VHDL Designer’ or ‘X-BLOX’

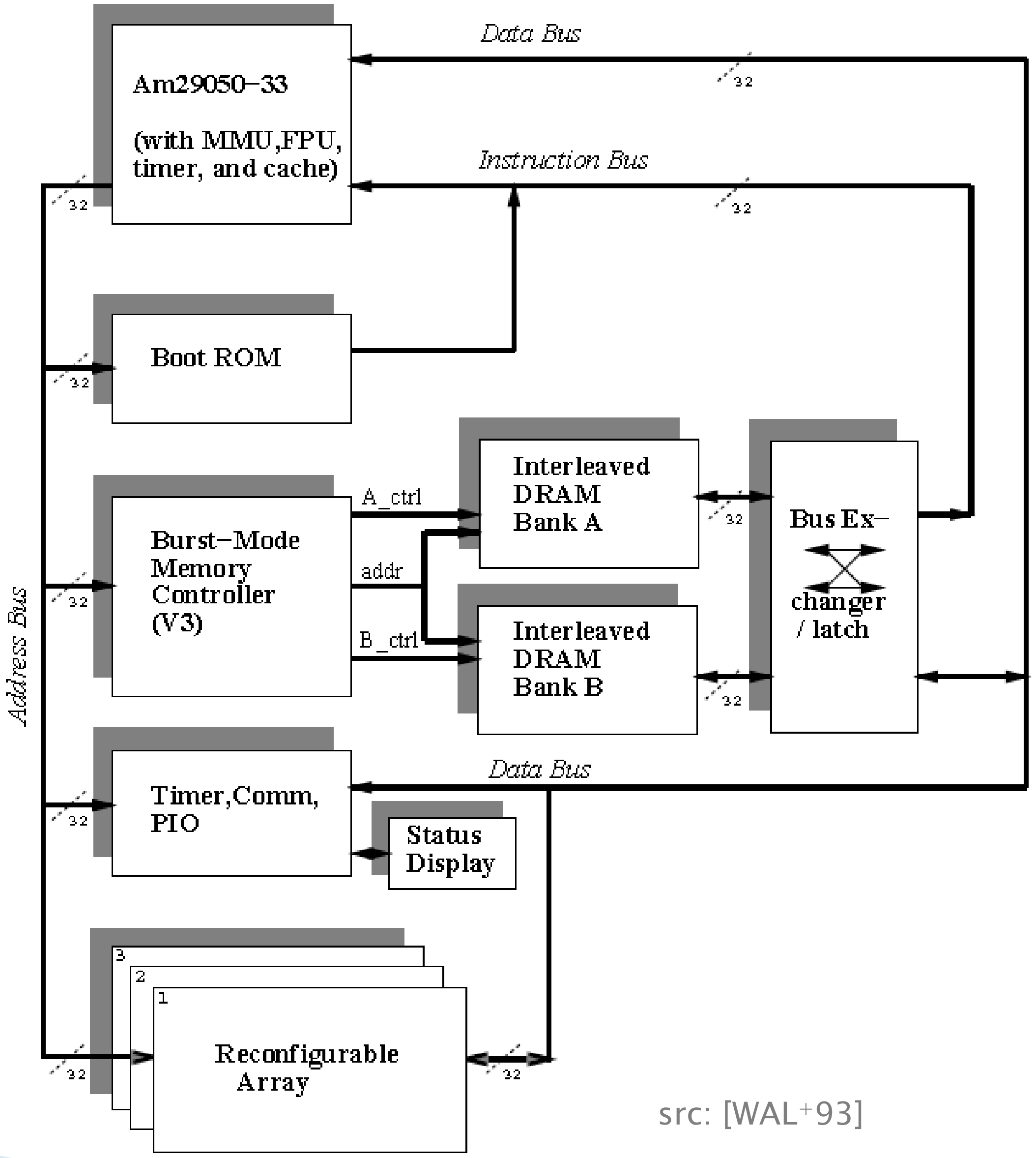


Program Source (C)
GCC Front End : Parsing and Standard Optimizations
Register Transfer Language (RTL)
VHDL Generation
Program FLow Graph Generation
VHDL Designer
X-BLOX Netlist Generation
Xilinx Tools (PPR etc.)
Simulation
Hardware Images
Simulation

src: [WAL+93]

PRISM-II Architecture

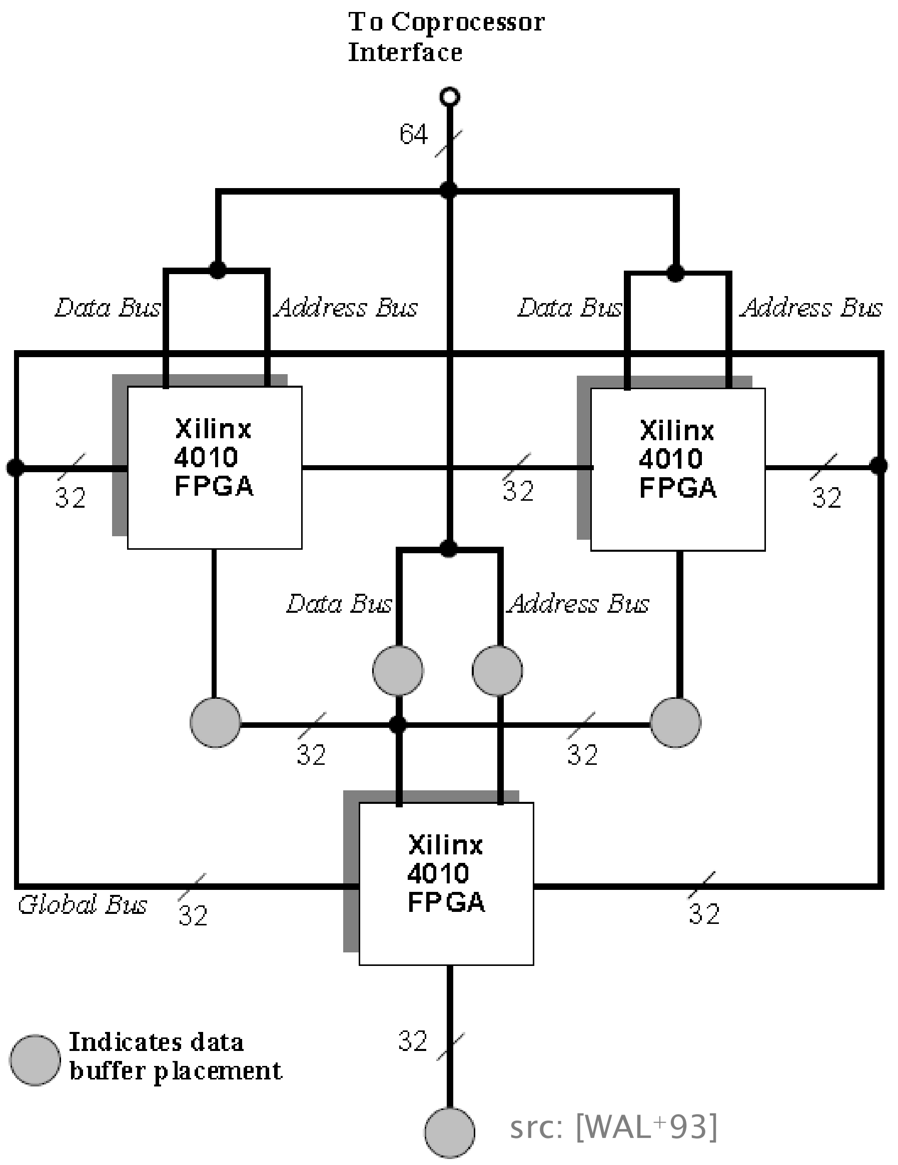
- AMD Am29050 at 33 MHz, 28 MIPS
- Coprocessor-like reconfigurable fabric
- 64-bit bus
  - Using the Address Bus and the Data Bus at the same time
  - Only 32-bit results are allowed
- Tighter coupling
  - Only 30 ns data movement cost (i.e. 1 cycle @ 33 MHz)



src: [WAL+93]

PRISM-II Architecture (cont'd)

- 3 Xilinx 4010 FPGAs
  - An SI may spread over all 3 FPGAs
- By utilizing data buffers, the FPGAs can work together or perform individual tasks
- The "Global Bus" provides control signals to be shared between FPGAs
  - used for providing global clocks
  - or transferring state information between the FPGAs
- Reported Speedup:
  - 86x for simple bit reversal
  - 10x for computing a Hamming code



src: [WAL+93]

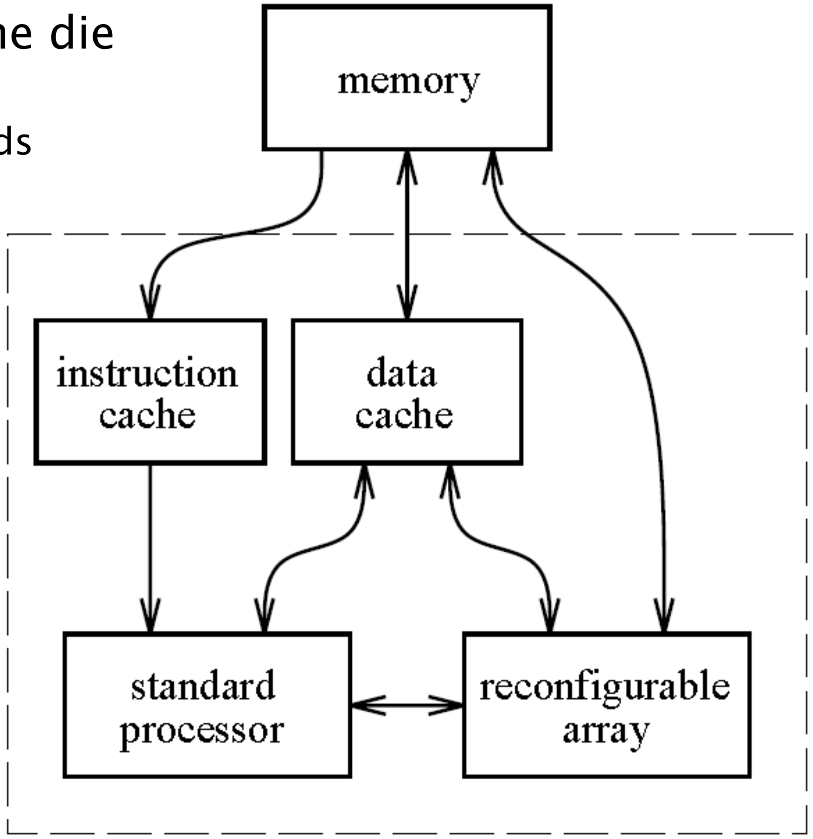
PRISM Summary

- Very early approach (1993) for a loosely coupled reconfigurable component
- PRISM-I: external Processing unit
- PRISM-II: external Coprocessor (to some degree)
- Very slow coupling
- Very slow reconfiguration time (range of seconds, not milliseconds)
- Relies on very old FPGAs (from today's perspective)
  - Multiple FPGAs are combined to obtain a reasonable amount reconfigurable fabric

4.3 Garp

Garp Overview

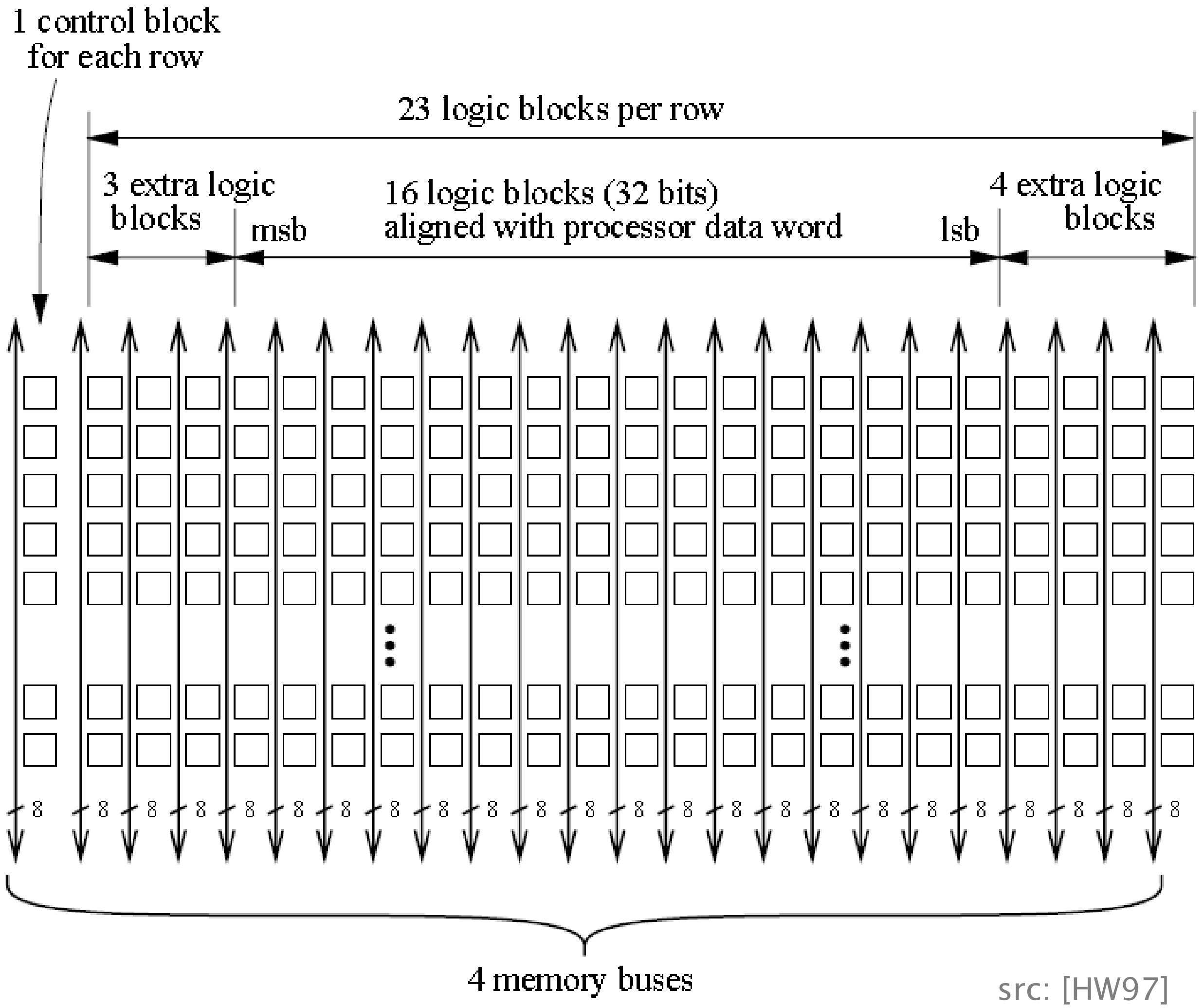
- Research effort on overcoming the limitations of reconfigurable HW
  - Reconfiguration overhead
  - Memory access from reconfigurable hardware
  - Binary compatibility of executables across version of reconfigurable hardware
- Core processor and reconf. fabric on same die
  - Core processor: a single-issue MIPS-II
  - Reconfigurable Fabric as Coprocessor, but needs some modifications in the core processor
  - However, no actual chip produced
- Core processor and reconf. fabric share the same memory hierarchy
- SW controlled run-time reconfiguration
- Reconfigurable fabric runs asynchro-nously to the core processor
  - Reconfigurable fabric: estimated 133 MHz



src: [HW97]

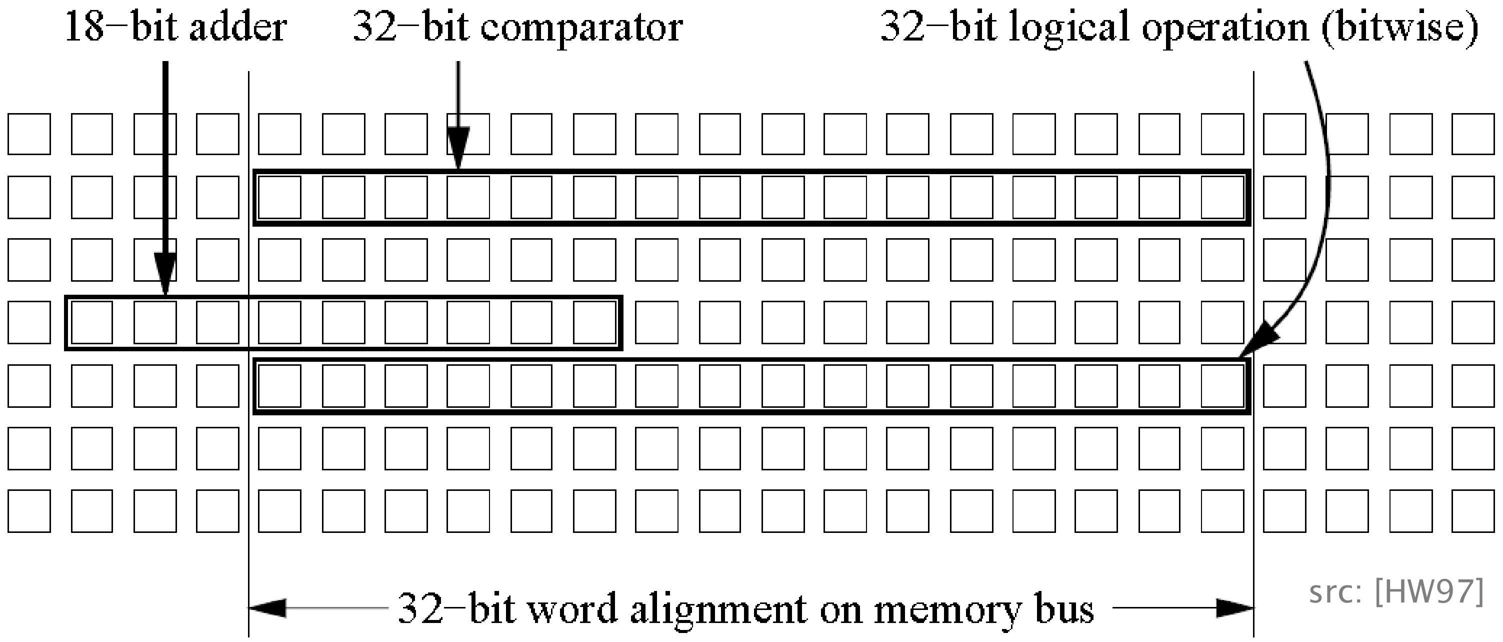
Garp Reconfigurable Fabric

- Reconf. fabric is a 2D-mesh com-posed of entities called blocks
  - Number of columns is fixed to 24 (1 control and 23 logic blocks)
  - Some special purposes blocks
  - Number of rows is implementation specific and can grow in an <u>upward-compatible</u> fashion (expected to be at least 32)


1 control block for each row
23 logic blocks per row
3 extra logic blocks
msb
16 logic blocks (32 bits) aligned with processor data word
lsb
4 extra logic blocks
8
4 memory buses

src: [HW97]

Garp Reconfigurable Fabric (cont'd)


18–bit adder
32–bit comparator
32–bit logical operation (bitwise)
32–bit word alignment on memory bus

src: [HW97]

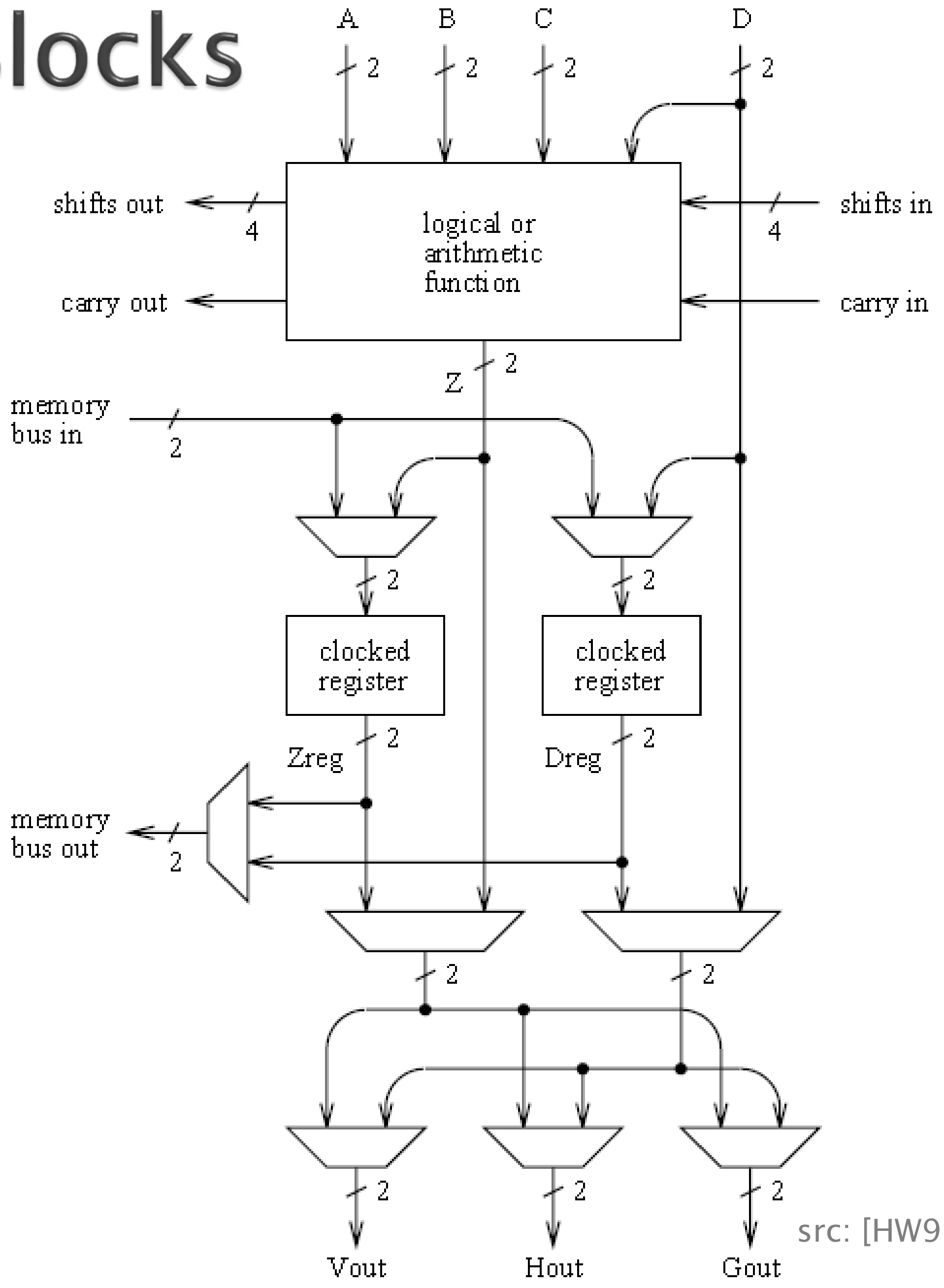
- Memory accesses can be initiated by the reconfigurable fabric, but only through the central 16 columns
- Extra blocks for overflow checking, rounding, control functions, wider data sizes etc.

Garp Reconfigurable Fabric (cont'd)

- Partially reconfiguring the reconfigurable fabric is supported
  - Basic reconfigurable unit is a row of 24 blocks, a so-called reconfigurable ALU
  - SI size is defined by #rows (→ 1D structure)
    - A row is exclusively used by at most one SI, i.e. it is not allowed that some logic blocks in a row are used for Si_i and some others in the same row are used for $Si_{j,\ j \neq i}$
  - Fabric is blocked during reconfiguration
  - Supports run-time relocation (a hardware translates from logical to physical row number)

Reconfigurable Blocks

- Each logic block takes as many as four 2-bit inputs and produces up to two 2-bit outputs
- Routing architecture:
  - 2-bit buses in horizontal and vertical columns
  - global & semi-global lines



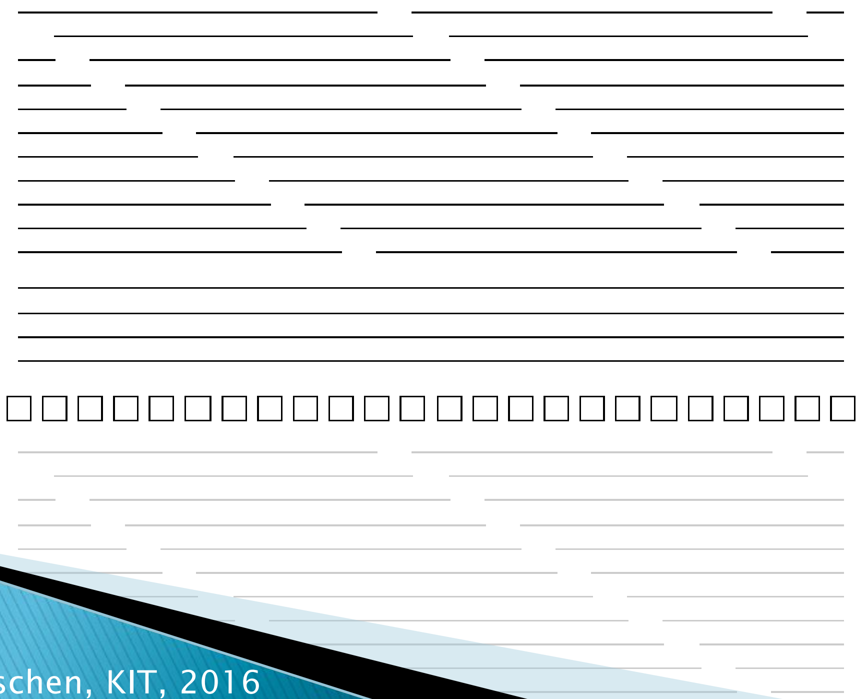
src: [HW97]

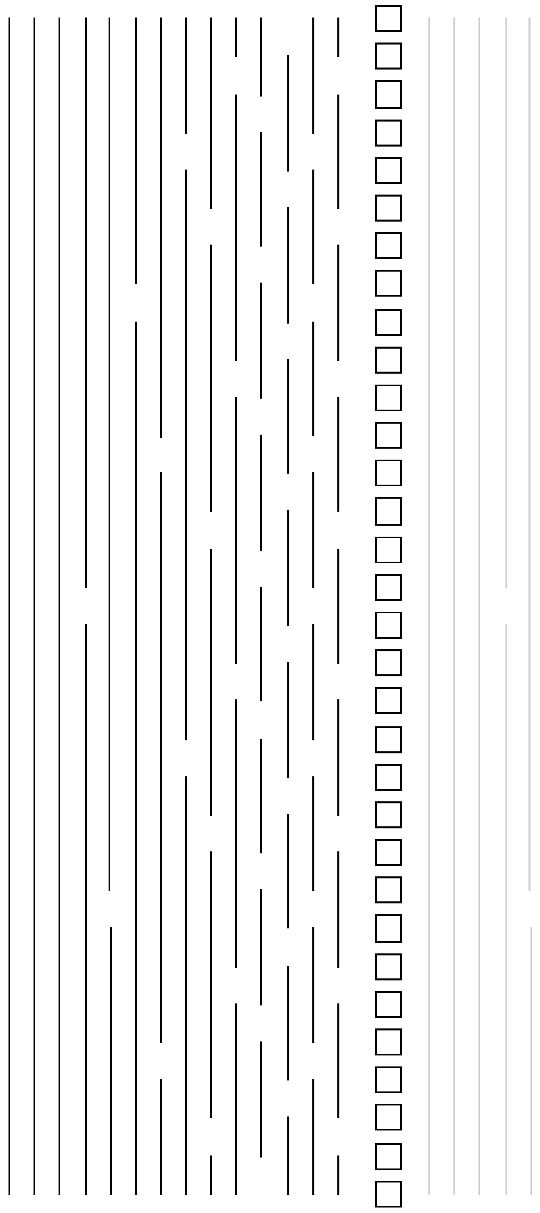
Reconfigurable Blocks (cont'd)

- Each logic block can be configured to perform
  - an arbitrary 4-input bitwise logical function,
  - a variable-length shift of up to 15 bits,
  - a 4-way select (multiplexer) function, or
  - a 3-input add/subtract/comparison function
- Garp made a first step to integrate specialized hardware blocks into a partially reconfigurable processor (not only LUTs)
  - Multi-bit adders, shifters etc. are designed with 'more hardware' than typically FPGAs at that time
- Each logic block includes four bits of data state (i.e. registers); in total 92 bits per row

Reconfigurable Routing

- The routing architecture includes 2 bit horizontal and vertical lines of different length, segmented in a non-uniform way
  - Short horizontal segments spanning 11 blocks are tailored to multi-bit shifts across a row
  - Note: the figures shows the routing for <u>one</u> row/column of logic blocks, respectively

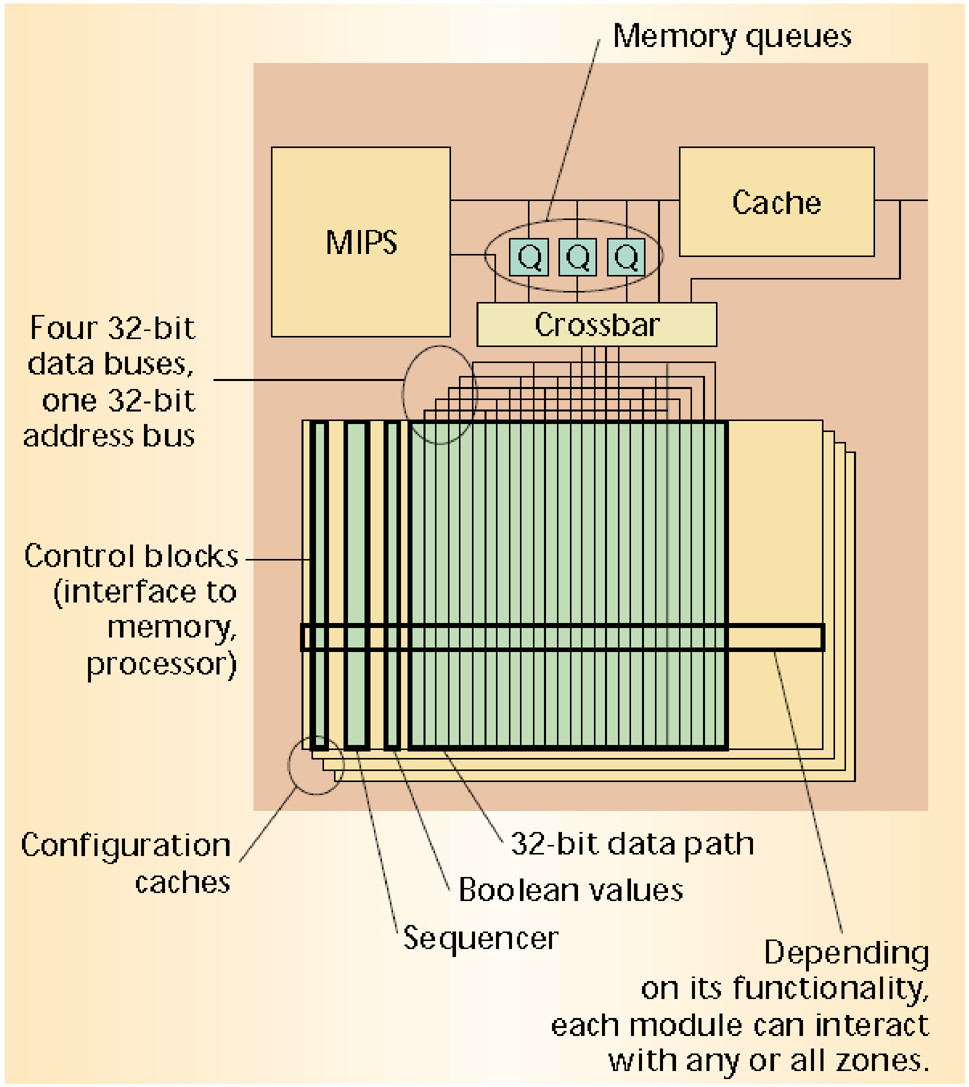




src: [HW97]

Data Access

- Data input/output
  - Up to 128 bits per cycle to/from any 4 rows in the fabric
  - Up to 64 bits per cycle from the MIPS core register file to any 2 rows
  - Up to 32 bits per cycle from any row back to the MIPS core register file
- Dedicated Queues
  - Allowing read ahead and write behind



Memory queues
MIPS
Cache
Q
Q
Q
Crossbar
Four 32-bit data buses, one 32-bit address bus
Control blocks (interface to memory, processor)
Configuration caches
32-bit data path
Boolean values
Sequencer
Depending on its functionality, each module can interact with any or all zones.

src: [CHW00]

Reconfiguration Management

- For fast reconfiguration, the reconfigurable fabric features a transparent distributed configuration cache
  - Holds the equivalent of 128 total rows of configurations
  - Distributed as 4 cached configuration rows for each physical row
  - Stores the least recently used configurations
  - Content can be pre-fetched
- Reconfiguration time from external memory is 12 external bus cycles <u>per row</u> plus some startup time
- Reconfiguration time from the integrated cache is 4 cycles (independent of the number of rows)

Reconfiguration Management (cont'd)

- Reconfiguration
  - A block requires 64 configuration bits
  - Configuring 32 rows: 8 [Bytes/block] x 24 [blocks/row] x 32 [rows] = 6144 Bytes
  - Assuming 128-bit memory access, 384 sequential accesses are required
  - Approx. 50 micro seconds (depending on the bus)
- To accelerate context switching, the Garp array does not contain large amount of embedded memory (if an SI needs some data twice, it typically has to load it twice)
- Supports virtual memory, supervisor mode, and protected execution of multiple processes
- Reported speedup (for hand-coded functions) compared to a 4-way superscalar UltraSparc 170:
  - 43x for an image median filter
  - 18.7x for DES encryption

Garp Programming

- The host has instruction set extensions (ISEs) to configure and control the reconfigurable fabric
  - Some instructions interlock (i.e. stall) until completion
  - Array execution is initialized by the number of clock cycles that shall be performed

Instruction	Interlock?	Description
`gaconf` *reg*	yes	Load (or switch to) configuration at address given by *reg*.
`mtga` *reg*, *array-row-reg*, *count*	yes	Copy *reg* value to *array-row-reg* and set array clock counter to *count*.
`mfga` *reg*, *array-row-reg*, *count*	yes	Copy *array-row-reg* value to *reg* and set array clock counter to *count*.
`gabump` *reg*	no	Increase array clock counter by value in *reg*.
`gastop` *reg*	no	Copy array clock counter to *reg* and stop array by zeroing clock counter.
`gacinv` *reg*	no	Invalidate cache copy of configuration at address given by *reg*.
`cfga` *reg*, *array-control-reg*	no	Copy value of array control register *array-control-reg* to *reg*.
`gasave` *reg*	yes	Save all array data state to memory at address given by *reg*.
`garestore` *reg*	yes	Restore previously saved data state from memory at address given by *reg*.

src: [HW97]

Garp Programming (cont'd)

- Example for loading and executing an SI:

```
add3:  la v0,config_add3    # v0 now contains pointer
                            # to 'config_add3' array
       gaconf v0            # Configure
       mtga a0,$z0          # Transfer input data
       mtga a1,$d0
       mtga a2,$d1,2        # Move 3rd input and let
                            # array exec. for 2 cycles;
       mfga v0,$z1          # Collect result
       j ra                 # Return from subroutine
```

src: [HW97]

Garp Tool Chain

- Uses the SUIF C compiler for the front-end
- Accelerates non-nested loops
- The compiler performs the following tasks:
  - Kernel identification for executing on reconfigurable hardware
  - Design of the ‘optimum’ hardware for the kernels
    - This includes module selection, placement, and routing for the kernels
  - Modification of the application to organize the interaction between processor instructions and the reconfigurable instructions

Garp Tool Chain (cont'd)

- The compiler uses a technique first developed for VLIW architectures called hyperblock scheduling
  - These transformations can increase the available instruction-level parallelism (ILP)
  - A contiguous group of basic blocks is converted into a hyperblock
    - Potentially from alternative (if-then-else) control paths
    - Control flow inside a hyperblock is converted to predicated execution
  - Optimizes for ILP across common paths
    - Often executed paths are synthesized to the reconf. fabric
    - Infeasible or rare paths are implemented on the core processor
  - The resulting reduced hyperblock is the basis for mapping
  - When execution takes an *excluded path* (i.e. not part of the synthesized logic), an *exceptional exit* occurs

Garp Tool Chain (cont'd)

- After hyperblock scheduling, interfacing instructions for the core processor are generated and the reduced hyperblock is transformed into a data flow graph (DFG)
- The proprietary 'Gamma tool' maps the DFG onto Garp using a tree covering algorithm which preserves the datapath structure and supports features like carry chains
  - Gamma first splits the DFG into subtrees and then matches subtrees with module patterns which fit in one Garp row
  - During tree covering, the modules are also placed in the array
  - After some optimizations the configuration code is generated and assembled into binary form
  - Configuration bits are included and linked as constants with ordinary C compiled programs

Garp Summary

- Attaches the reconf. fabric as Coprocessor (executes asynch.), but needs some modifications in the core processor for interfacing instructions
- Dedicated instructions for reconfig., data exchange, and SI execution
- Proposed a dedicated fine-grained reconfigurable fabric (2-bit granularity) that is optimized for run-time reconfiguration
  - Partially reconfigurable in a 1D row structure
  - Configuration relocation
  - Optimized for 32-bit operations (size of a row)
  - Only 12 external memory accesses to reconfigure a row
  - Distributed configuration cache
- Binary compatibility for smaller/larger reconfigurable fabrics
- Carefully designed data memory access, including Cache access and dedicated memory queues for streaming
- Tool chain that automatically creates configurations and interfaces
  - Based on Hyperblocks, optimization of common paths, and predicated execution

4.4 The MOLEN Polymorphic Processor

src: http://ce.et.tudelft.nl/MOLEN/

Overview

- Reconfigurable Coprocessor
- Using a one-time instruction set extension
  - Inspired by Garp
- Using a reconfigurable microcode (*ρμ*-code)
  - Difference to traditional microcode: it does not execute on fixed hardware facilities, but it operates on facilities that the *ρμ*-code itself creates (i.e. reconfigures) to operate upon
  - Microcode to control the reconfiguration
  - Microcode to control the SI execution
- Prototype uses a Virtex-II Pro FPGA
  - Using the embedded PowerPC as core processor

Overview: One-time instruction-set extension

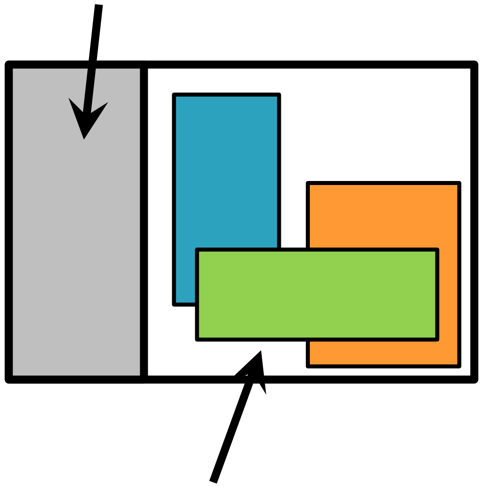
- Loading and prefetching the configurations
  - p-set (partial set), c-set (complete set), set-prefetch
- Executing and prefetching the microcode for SIs
  - execute, execute-prefetch
- Load/store instructions
  - movtx, movfx (move to/from eXchange registers)
- Synchronization instruction
  - break

One-time instruction-set extension

- p-set <address>
  - performs the configuration of common parts/ frequently used functions
- c-set <address>
  - Performs configuration of the remaining area that was not covered by p-set
  - Note: c-set is executed more often than p-set
- In case no partially reconfigurable hardware is present, the c-set instruction alone can be utilized to perform all the necessary configurations

Example:

Reconfigured once, using p-set



Reconfigured often, using c-set

One-time instruction-set extension (cont'd)

- The <address> points to a memory location that contains Microcode
  - Controlling the reconfiguration or the SI execution
  - For reconfiguration the Microcode corresponds to a bitstream (i.e. configuration data)
    - Execution is terminated when a specific end address (provided at the beginning of the Microcode) is reached
  - For SI execution the Microcode *could* correspond to a state machine that controls the execution (not further specified/explained by the authors)
    - Terminated by a special 'end_op' Microcode

One–time instruction–set extension (cont'd)

- set–prefetch <address>
  - Prefetches the Microcode that is responsible for reconfigurations into a local on–chip storage facility (the $\rho\mu$ –code unit)
  - diminish microcode loading times
- execute <address>
  - Triggers the execution of an SI
- execute–prefetch <address>
  - Prefetches the Microcode that is responsible for SI execution

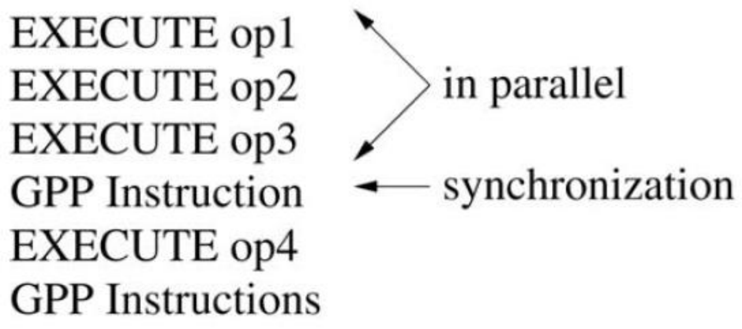
One-time instruction-set extension (cont'd)

- An exchange register file is used for explicit parameter passing
  - Size is implementation specific
  - 512 entries for their Virtex-II Pro prototype
  - The compiler performs the register allocation
- movtx XREGa Rb
  - Move the content of general-purpose register Rb to XREGa
- movfx Ra XREGb
  - Move the content of exchange register XREGb to general-purpose register Ra
- The Virtex-II Pro prototype uses the dedicated PowerPC interface to the so-called Device Control Registers (DCR) to implement movtx and movfx

One–time instruction–set extension (cont'd)

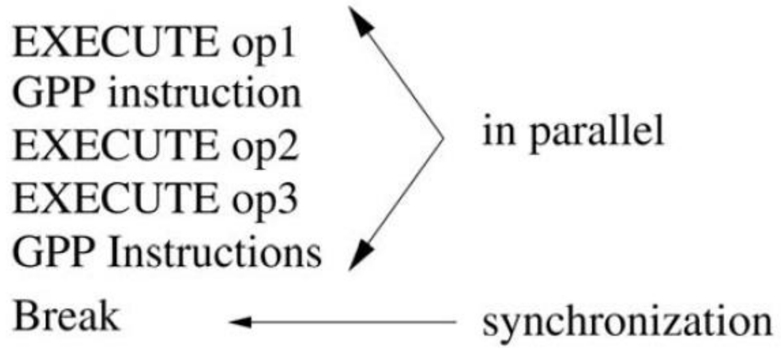
- Break
  - Utilized to facilitate the parallel execution of the reconfigurable processor and the core processor
  - Synchronization mechanism that stalls the core processor until the parallel execution of the reconfigurable processor is completed

Implicit Synchronization:



```
EXECUTE op1
EXECUTE op2
EXECUTE op3
GPP Instruction
EXECUTE op4
GPP Instructions
```

Explicit Synchronization:

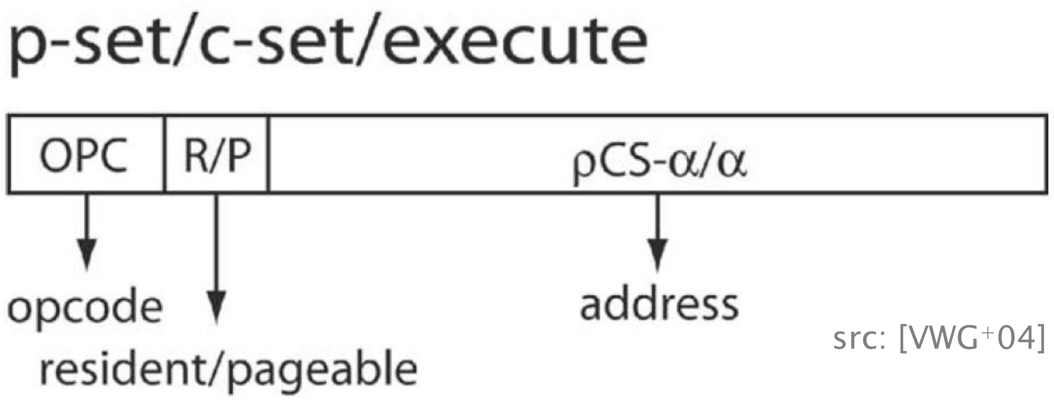


```
EXECUTE op1
GPP instruction
EXECUTE op2
EXECUTE op3
GPP Instructions
Break
```

src: [VWG+04]

One–time instruction–set extension (cont'd)

- All instructions use a simple format
  - Opcode: specifies the one–time instruction–set extensions (assures that it does not overlap with the instructions from the core processor)
  - Address: the start address of the Microcode
  - R/P bit: interpretation of the address
    - Resident: an on–chip ROM for often used Microcodes
    - Pageable: the off–chip RAM for other Microcodes



src: [VWG+04]

Instruction Set Alternatives

- The minimal ISA:
  - c-set, execute, movtx, movfx
- This is essentially the smallest set of Molen instructions needed to provide a working scenario
- By implementing the first two instructions (set/execute), any suitable SI implementation can be loaded and executed in the reconfigurable processor
  - Reconfiguration latencies can be hidden by scheduling the set instruction considerably earlier than the execute instruction
- The movtx and movfx instructions are needed to provide the input/output interface between the SI and the remaining application code

Instruction Set Alternatives (cont'd)

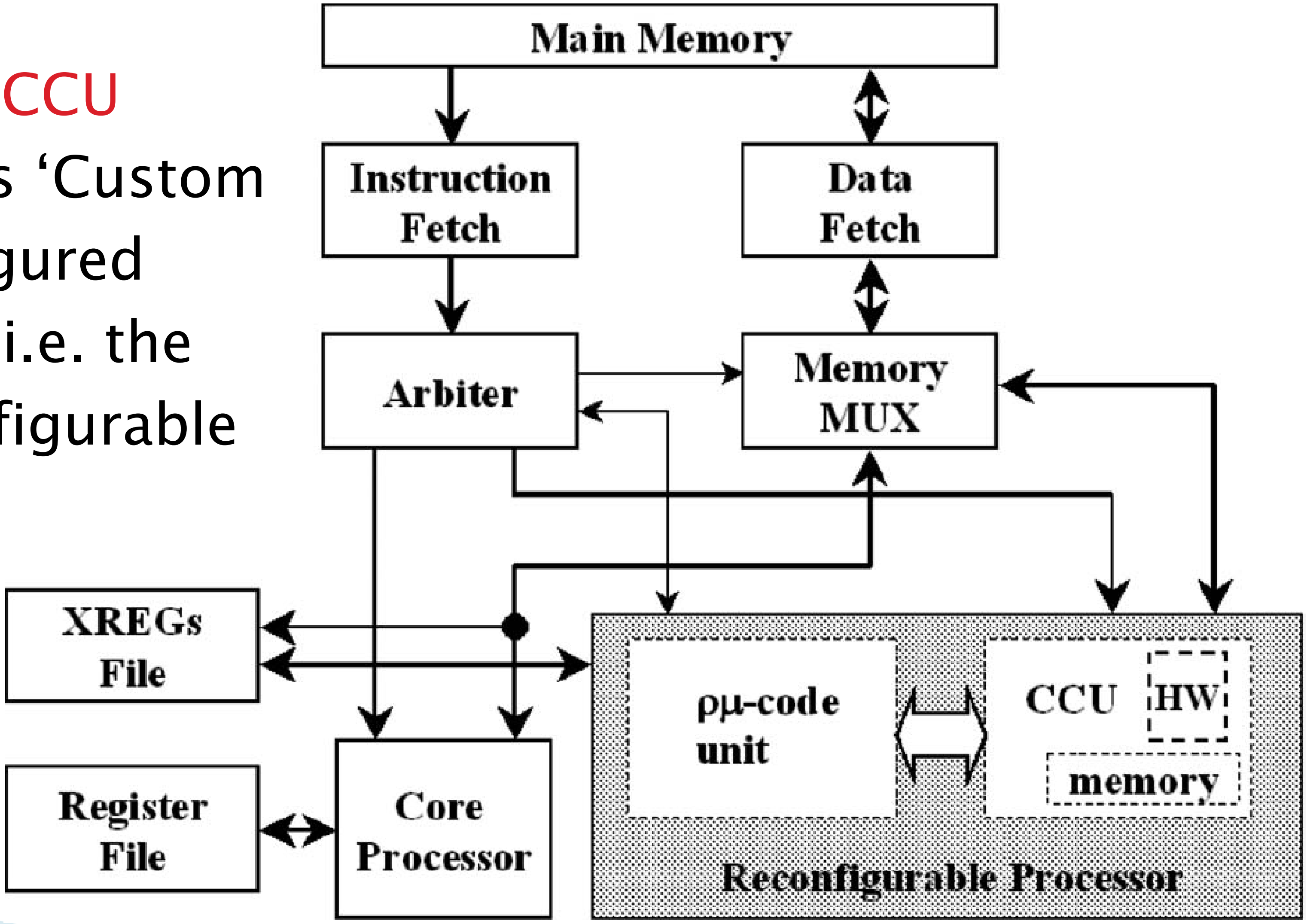
- The preferred ISA:
  - **p-set**, c-set, **set-prefetch**, execute, **execute-prefetch**, movtx, movfx
- In order to reduce reconfiguration latencies both p-set and c-set instructions are utilized
  - Then, the loading time of microcode will play an increasingly important role
  - Thus, the two prefetch instructions provide a way to diminish the microcode loading times by scheduling them well ahead of the moment that the microcode is needed
- Parallel execution is initiated by a set/execute instruction and ended by a general purpose instruction (same for minimal ISA)

Instruction Set Alternatives (cont'd)

- The complete ISA:
  - p-set, c-set, set-prefetch, execute, execute-prefetch, movtx, movfx, **break**
- Involves all ISA instructions including the break instruction
- In some applications, it might be performance-wise beneficial to execute instructions on the core processor and the reconfigurable processor in parallel
  - The break instruction provides a mechanism to synchronize the parallel execution of instructions by halting the execution of instructions following the break instruction
  - The sequence of instructions performed in parallel is initiated by an execute instruction or a set instruction

MOLEN Architecture Overview

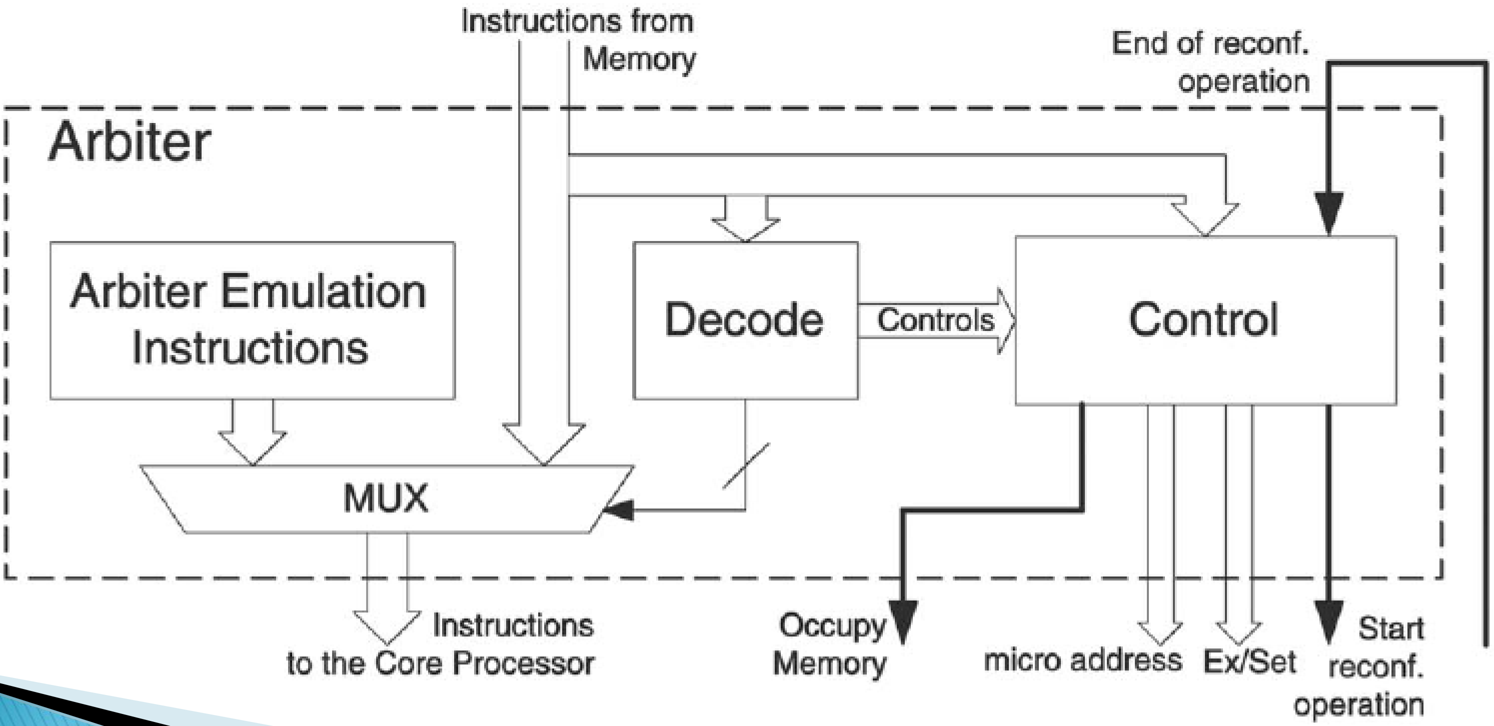
- Note, CCU means 'Custom Configured Unit', i.e. the reconfigurable fabric



src: [VWG+04]

Instruction Arbiter

- An instruction is either issued to the core processor or to the reconfigurable coprocessor (Arbiter decides)



src: [VWG+04]

Instruction Arbiter (cont'd)

- In case of a exec/set etc. instruction, control signals from the Decode block are transmitted to the Control block, which performs the following steps:
  1. Redirect the microcode location address to the $\rho\mu$–code unit
  2. Generate an internal code representing either an execute or set instruction ("Ex/Set" signal) and send it to the $\rho\mu$–code unit
  3. Initiate the operation by generating "start reconf. operation" signal to the $\rho\mu$–code unit
  4. Reserve data memory control for the Reconfigurable Processor by sending a memory occupy signal to the data memory MUX
  5. Stall the core processor (Emulation Instructions) and enter a wait state until "end of reconf. operation" arrives

Instruction Arbiter (cont’d)

- An active “end of reconf. operation” signal initiates the following actions:
  1. Data memory control is released back to the Core Processor
  2. An instruction sequence is generated to ensure proper exiting of the Core Processor from the wait state
  3. After exiting the wait state, the program execu-tion continues with the instructions following the instruction for the Reconfigurable Processor

Instruction Arbiter (cont’d)

- “Arbiter Emulation Instructions” are multiplexed to the core processor instruction bus when the actual instruction is issued to the reconfigurable processor
  - Essentially drives the processor into a wait state
- Their Virtex-II Pro prototype uses the *blr* (branch to link register) instructions to activate the wait state
- Before a set/execute/etc. instruction, the link register is initialized (using the *bl* (branch and link) instruction) to point to that instruction

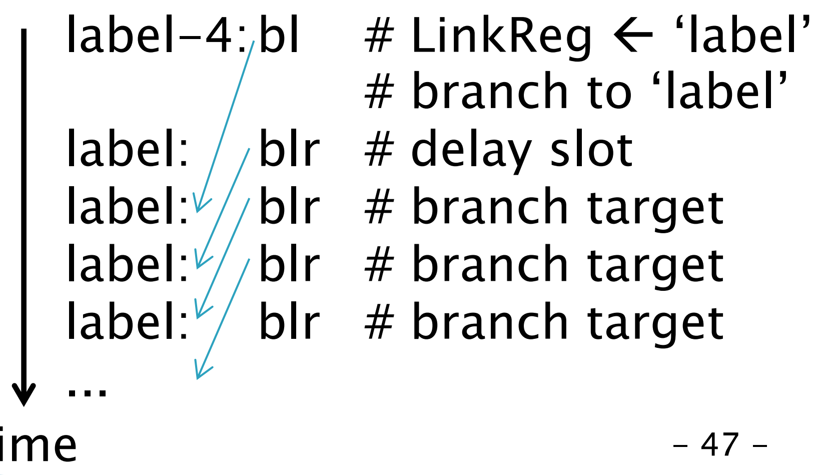
<u>Code Example:</u>

```
       bl label
label: c-set <addr>
       nop
       add
       ...
```

<u>Executed as:</u>

```
       bl label
label: blr
       nop
       add
       ...
```

<u>Control Flow during wait state</u>



```
label-4: bl    # LinkReg ← ‘label’
               # branch to ‘label’
label:   blr   # delay slot
label:   blr   # branch target
label:   blr   # branch target
label:   blr   # branch target
...
```

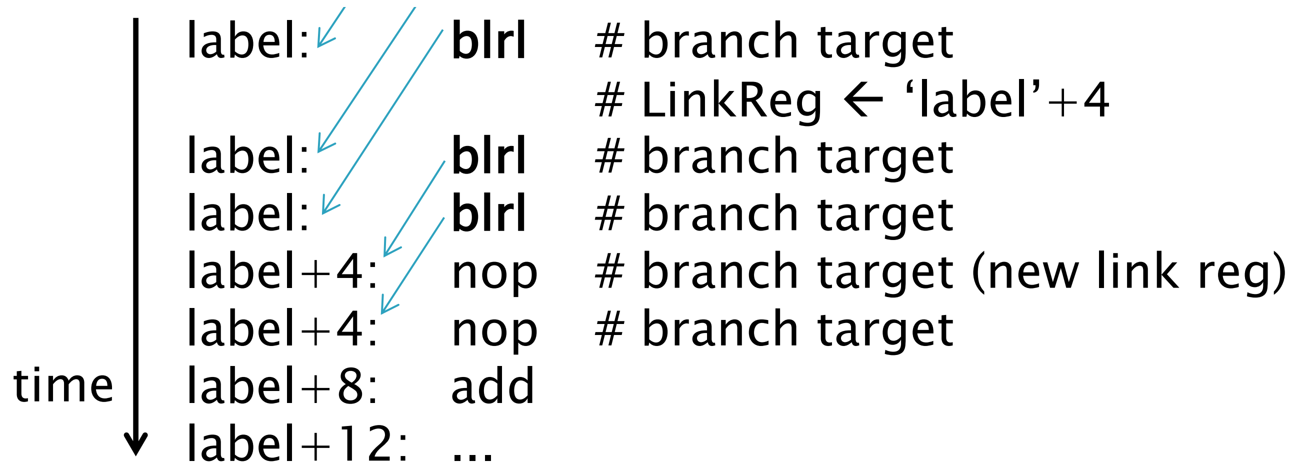
Instruction Arbiter (cont'd)

- To exit the wait state, the *blrl* (branch to link register and link) instruction is used
  - Updates the link register to point to the instruction that follows the branch

<u>Executed as:</u>

```
       bl label
label: blrl
       nop
       add
        ...
```

<u>Control Flow when exiting the wait state</u>

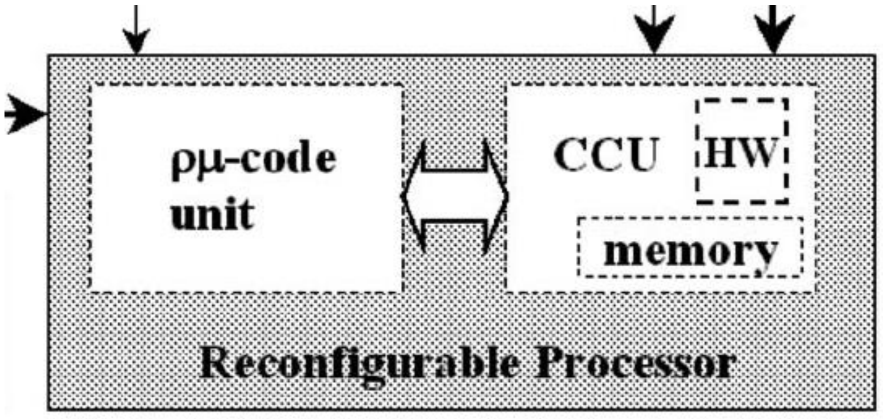


```
       label:     blrl   # branch target
                         # LinkReg ← 'label'+4
       label:     blrl   # branch target
       label:     blrl   # branch target
       label+4:   nop    # branch target (new link reg)
       label+4:   nop    # branch target
time   label+8:   add
  ↓    label+12:  ...
```

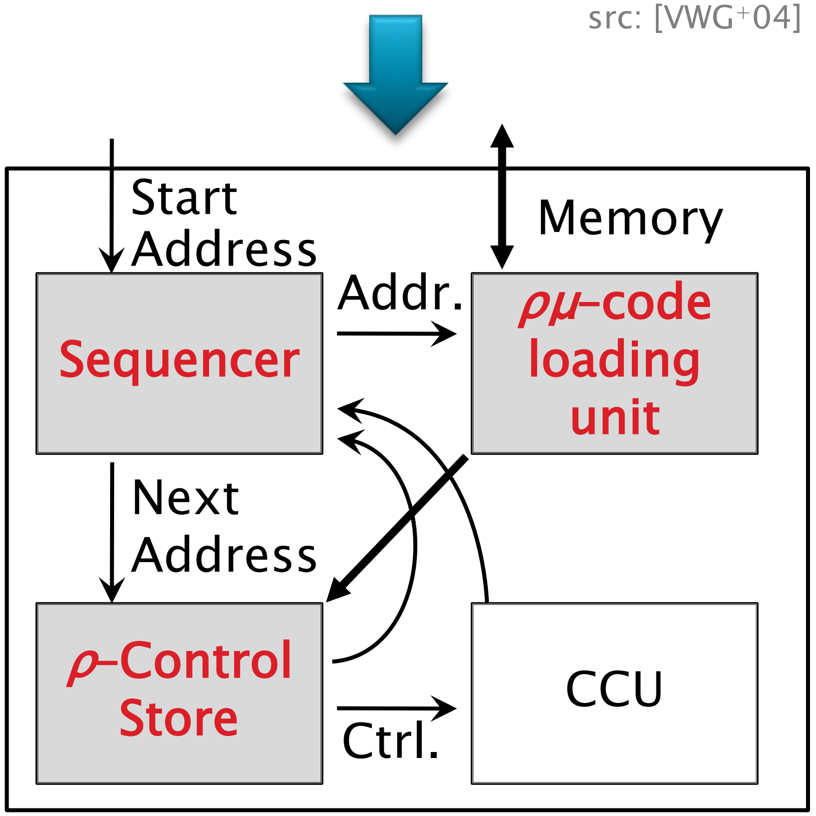
  - More care has to be taken when multiple set/exec instructions shall be executed in parallel

$\rho\mu$–code Unit

- The Sequencer mainly determines the micro-code execution sequence
- The ρ–Control Store is used as a storage facility for microcode
- The $\rho\mu$–code loading unit is responsible for loading the reconfigurable microcode from the memory

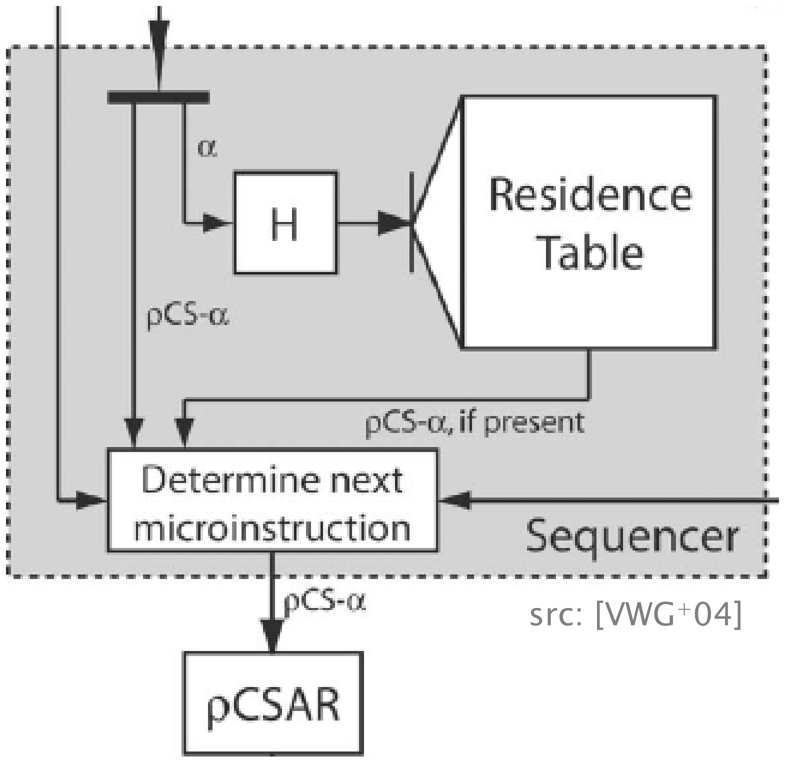


src: [VWG+04]



$\rho\mu$–code Unit (cont'd)

- The Sequencer is used to translate addresses of Microcode into *internal* address that are then sent to the ρ–Control Store Address Register (ρCSAR)
  - If the Microcode is stored in internal ROM (resident, fixed), then the address is just passed through
- The Residence Table in the Sequencer is used to translate addresses and to manage, which Microcode is available in the Control Store
  - Triggers memory access in case a required Microcode is not available
  - Uses an LRU replacement scheme to overwrite existing entries

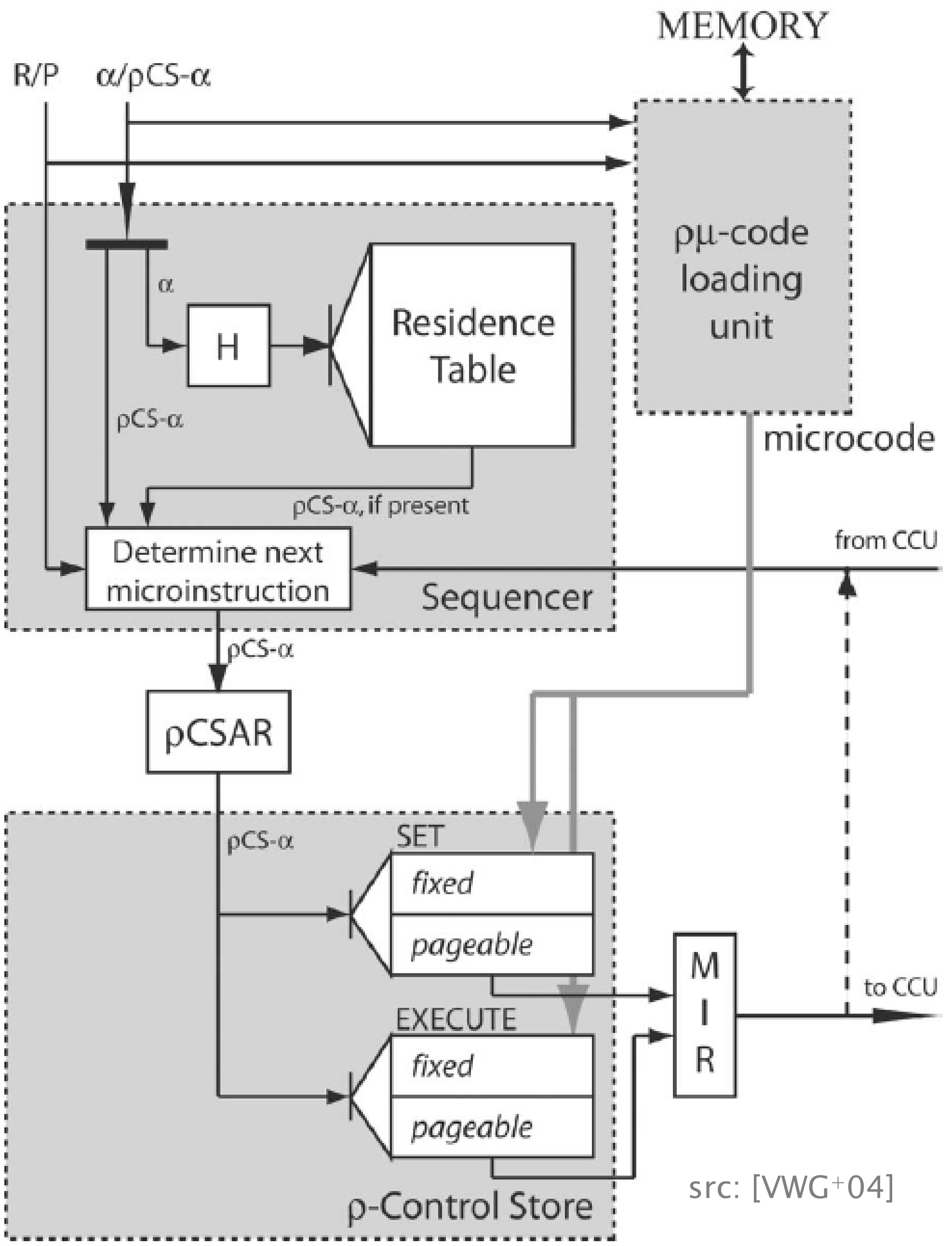


src: [VWG+04]

- The "H" block calculates a hash of the address that is used to access the residence table
- The residence table contains the information whether/where that address is placed in the Control Store

ρμ–code Unit (cont'd)

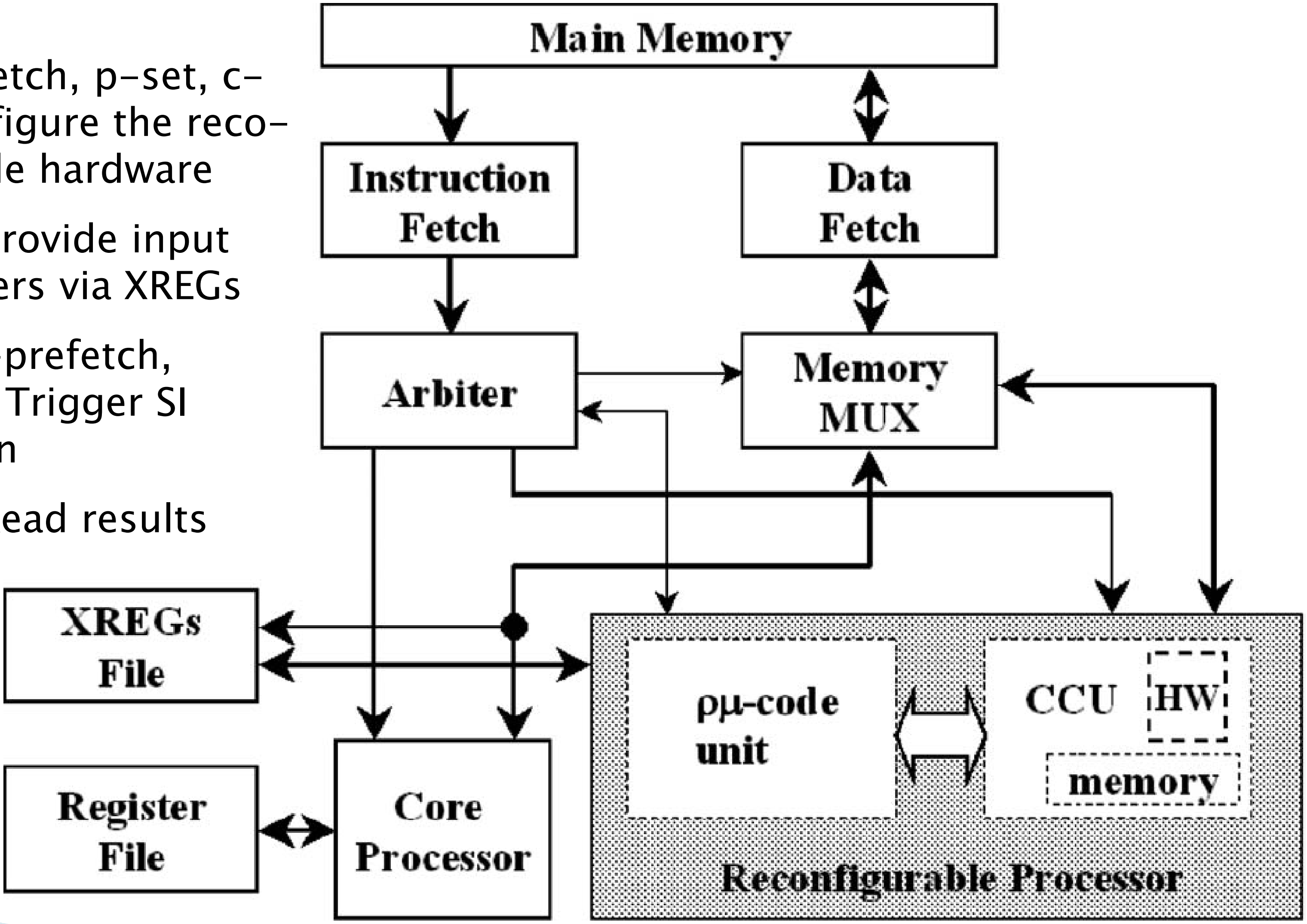
- *ρ*–Control Store contain different entries for set and execute Microcodes
  - They may differ in micro-instruction word size
  - Both contain different entries for fixed (ROM) and dynamic (RAM)
- The actual Microcode is decoded into Micro-instructions that are stored in the Microinstruction Register (MIR) to control the reconfigurable fabric
- The MIR value together with the return status of the reconfigurable fabric determine the next Microcode



src: [VWG+04]

Using MOLEN

- set-prefetch, p-set, c-set: Configure the reco-nfigurable hardware
- movtx: provide input parameters via XREGs
- execute-prefetch, execute: Trigger SI execution
- movfx: Read results back



src: [VWG+04]

Compiler Tool Chain

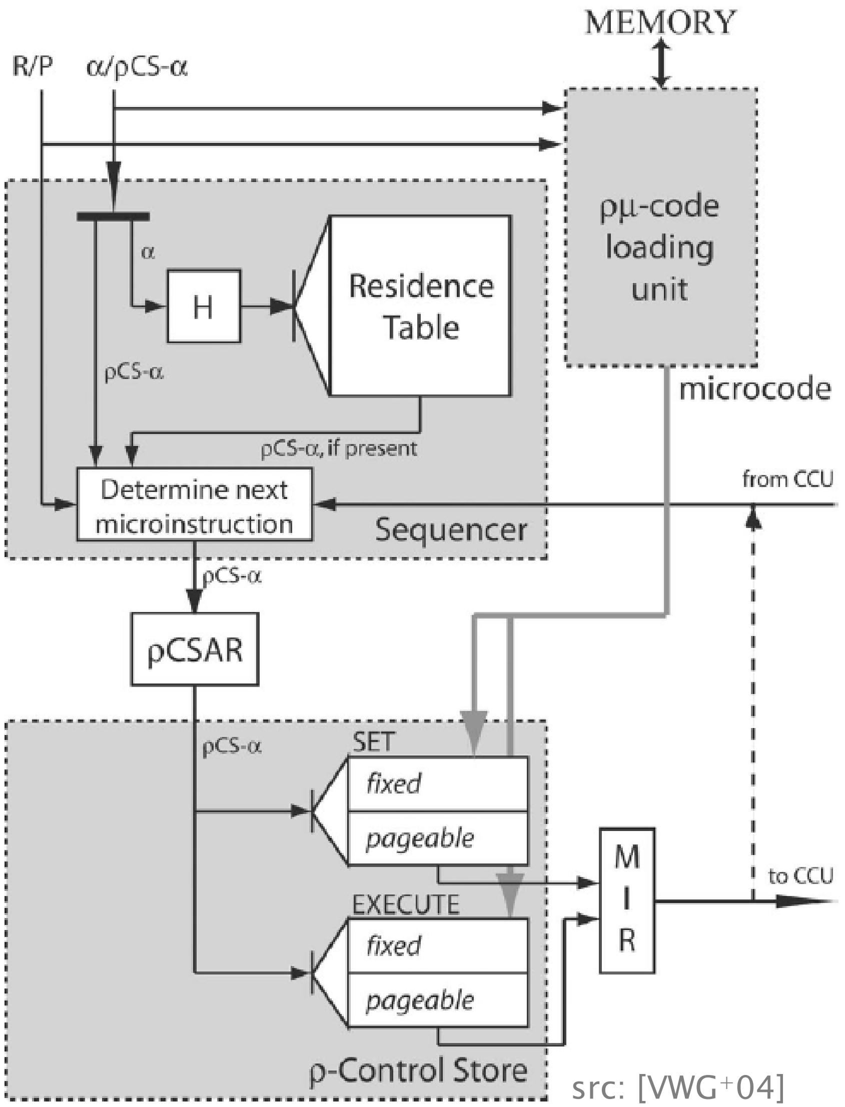
- Compiler relies on the Stanford SUIF2 Compiler Infrastructure for the front-end and for the back-end on the Harvard Machine SUIF framework
- Typically, pragmas denote a function that shall be implemented using the reconfigu-rable fabric
    - The signature of the function implicitly specifies the parameters that need to be passed

Compiler Tool Chain (cont'd)

- The following essential features for a compiler targeting custom computing machines are implemented:
  - Code identification:
    - A special pass in the SUIF front-end
    - Based on code annotation with special pragma directives
    - These function calls are marked for further modification
  - Instruction set extension:
    - Issuing set/execute instructions at the medium intermediate representation (IR) level and low IR level
  - Register file extension:
    - Register allocation algorithm allocates the XREGs in a distinct pass applied before the normal register allocation
    - Introduced in Machine SUIF at low IR level
  - Code generation:
    - Performed when translating SUIF to Machine SUIF intermediate representation

Problem: Limited Parallelism

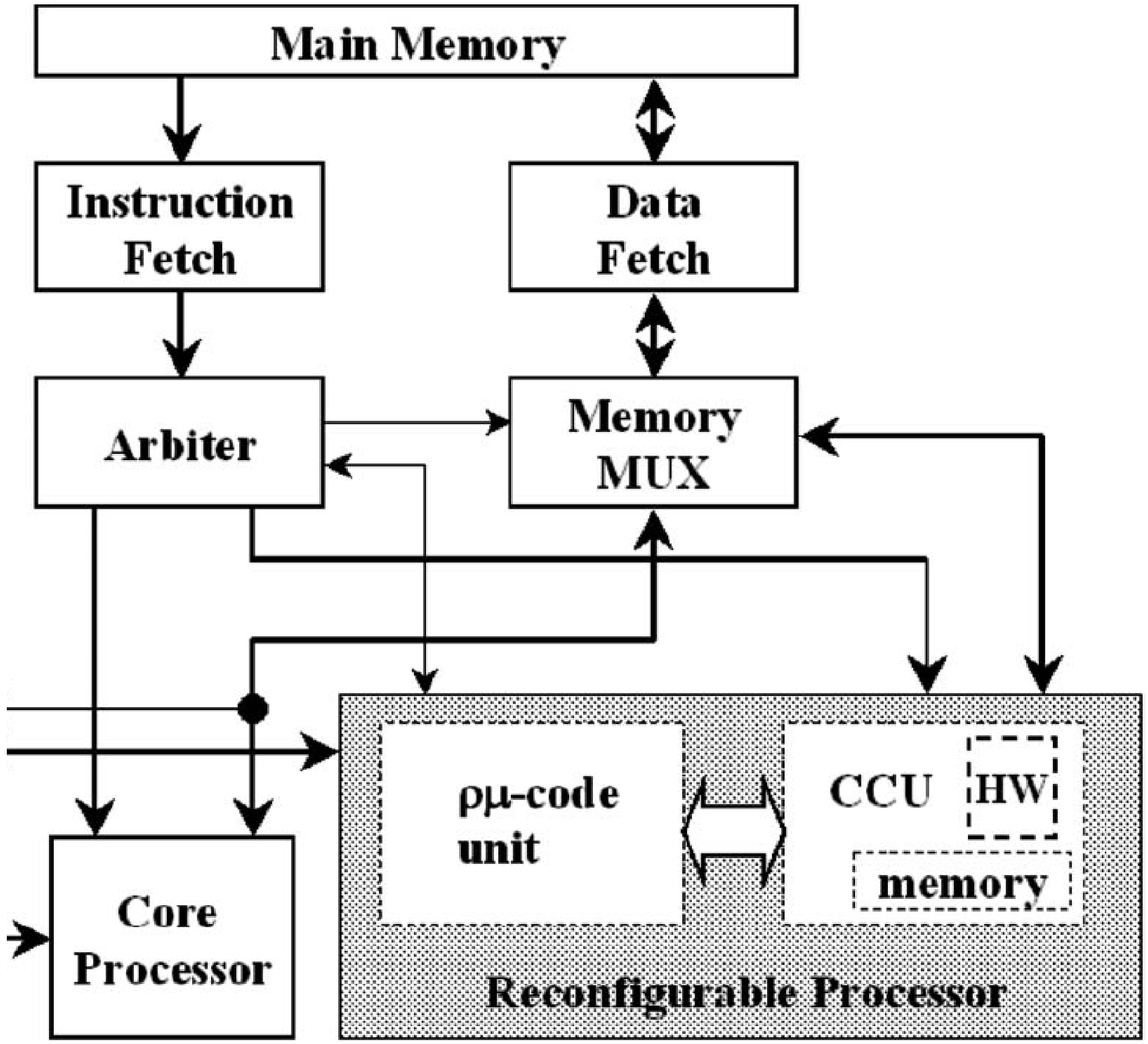
- Authors state that multiple operations targeting the reconfigurable fabric may execute in parallel
- Microcode Bottleneck
  - The Sequencer/ρ-Control Store can perform at most one operation (set or execute) at a time
  - Thus, the entire reconfigurable fabric is *stalled* during reconf.
  - And it is not possible to execute two SIs at the same time
  - 'Self-controlled' SI executions (i.e. not relying on Microcode) are mentioned but not explained any further



src: [VWG+04]

Problem: Limited Parallelism (cont'd)

- Memory Bottleneck
  - During SI exe-cution, the core CPU cannot access the main memory any more
  - Unclear, whether different SIs can access memory at the same time



src: [VWG+04]

MOLEN Summary

- Reconfigurable Coprocessor with a one-time instruction set extension (inspired by Garp)
  - Reconfiguring, Parameter passing, SI execution, synchronization
  - Different ISA alternatives (minimal, preferred, complete)
- Using a reconfigurable microcode ($\rho\mu$-code)
  - Controlling the reconfiguration
  - Controlling the SI execution
  - Sequencer, ρ-Control Store, $\rho\mu$-code Loading Unit
- Prototype uses PowerPC as core processor
  - Arbiter for the instructions (issue either to core CPU or to reconfigurable Coprocessor and send core CPU into 'wait state')
  - Multiplexer for memory access
- Compiler Tool Chain

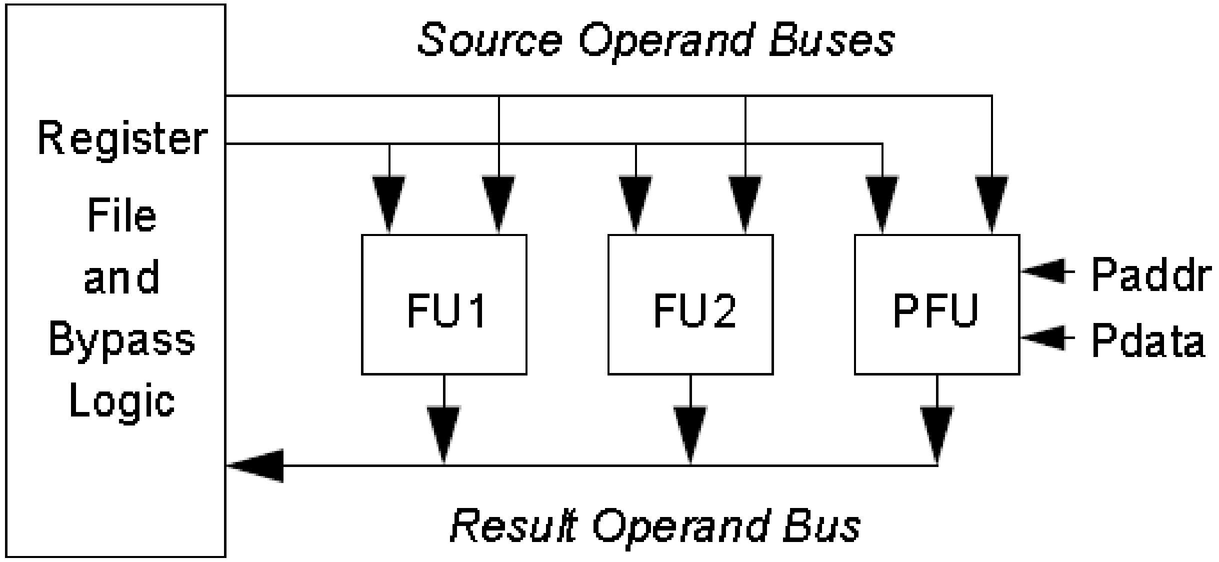
4.5 PRISC: PRogrammable Instruction Set Computer

Overview

- Tightly coupled functional unit
  - Using rather small amount of reconfigurable logic
- To some degree inspired by PRISM
  - Want to improve the communication delay, thus they propose a tight coupling
- Many other projects are (directly or indirectly) inspired by PRISC

Programmable Functions Unit

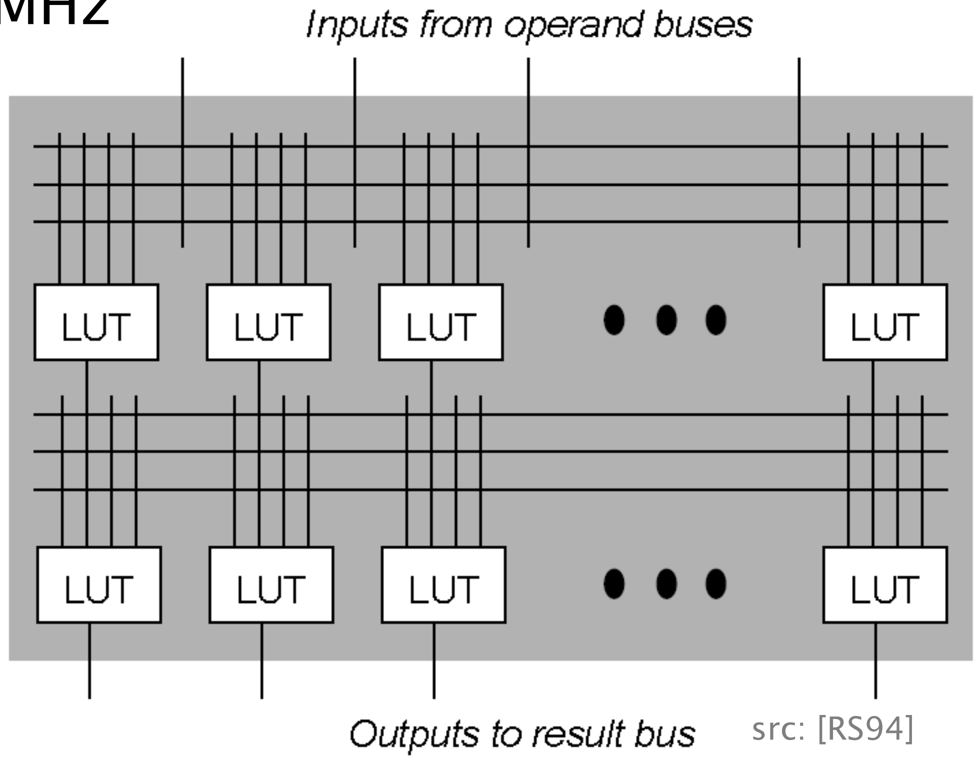
- A simple Programmable Functional Unit (PFU) that only evaluates combinational functions
  - 2 input, 1 output
- Carefully added to the microarchitecture such that is has only a minimal impact to the processor's cycle time
  - Some extra capacitive load on the source operand busses
  - Increases the size of the multiplexer for the result operand bus



src: [RS94]

Programmable Functions Unit (cont'd)

- Constraint: same delay as the delay of the already existing ALU etc.
  - Limiting the number of logic levels to bound the delay
  - Their PFU uses 3 logic layers (i.e. rows of LUTs) that should fit within a 200 MHz cycle time
  - Thus, the PFU can use the same synchro-nisation mechanisms as the other FUs
- Small area footprint (less than 1 KB on-chip SRAM)



src: [RS94]

Instruction Format

- One new instruction ‘Execute PFU’
- ‘LPnum’ defines which out of the 2048 different SI types shall execute
  - The authors state that “fewer than 200 PFU functions per application” are used [RS94]
  - Note: this is quite a lot and reflects the *small size* of the functions/PFU
  - However, the small PFU size might allow for such frequent reconfigurations

6	5	5	5	11
expfu	rs	rt	rd	LPnum

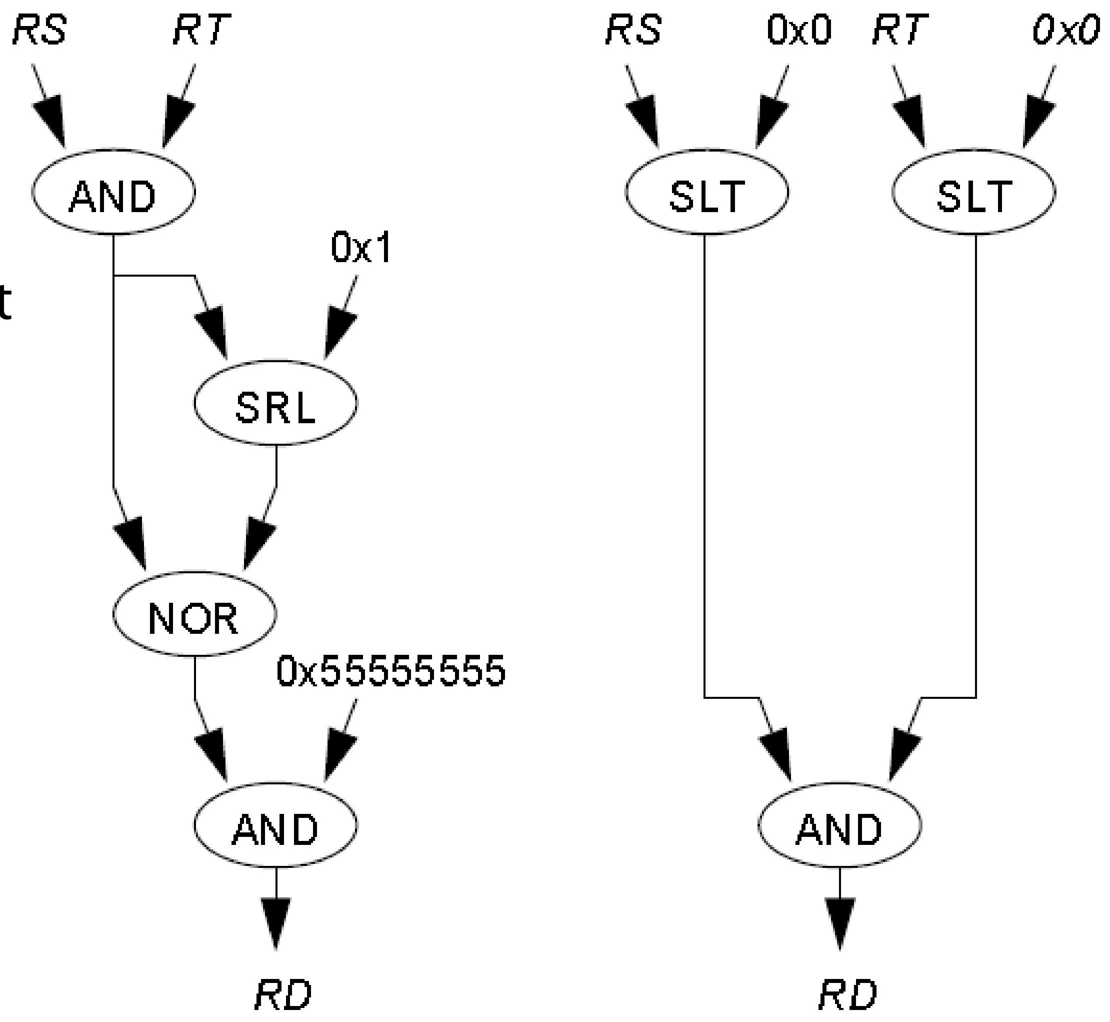
src: [RS94]

Instruction Format (cont'd)

- Each PFU is associated with a special 11-bit register that contains the SI number that is currently reconfigured into the PFU
- When an SI shall execute but is currently not reconfigured, then an exception is raised and the handler reconfigures the PFU
  - Observation by developers: Typically less than 15% of the configuration bits need to be set to '1'
  - At first, a 'hardware reset' sets all PFU configuration bits to '0'; afterwards, only the '1's are programmed
  - It takes 100-600 cycles to reconfigure 20% of the PFU memory bits

Special Instructions

- Compiler targets data dependent instructions
  - Works on control / data-flow graphs that can be implemented with logic functions
  - Supports everything except
    - memory access
    - floating point
    - wide arithmetic (not faster when executed on PFU)
    - mul/div
    - variable-length shifts



src: [RS94]

Special Instructions (cont'd)

- Complexity of an operation highly depends on its bit-width
  - Full bit-width (i.e. 32 bit) operations such as additions or multiplications are too complex for PFU resources
  - Try to identify the actually needed bit-width
- Bit-width analysis:
  - A combination of forward and backward traversals on the control/data-flow graph
  - Exploits cases where only some of the bits in a word are initialized (e.g. 'load byte', 'load immediate', 'and 0xFF') or only some of the bits are used later on
  - Algorithm iterates until no further bit-changes are found

src: [RS94]

Supported SI candidates

- PFU expression optimization: targets ‘logic’ instruction sequences
- PFU table lookup: implements truth tables
- PFU prediction optimization: targets if-then-else structures
- PFU jump optimization: targets sequences of if-then-else instructions; calculates the final branch target to reduce the number of sequential branches
- PFU loop optimization: loop unrolling to apply one of the above techniques

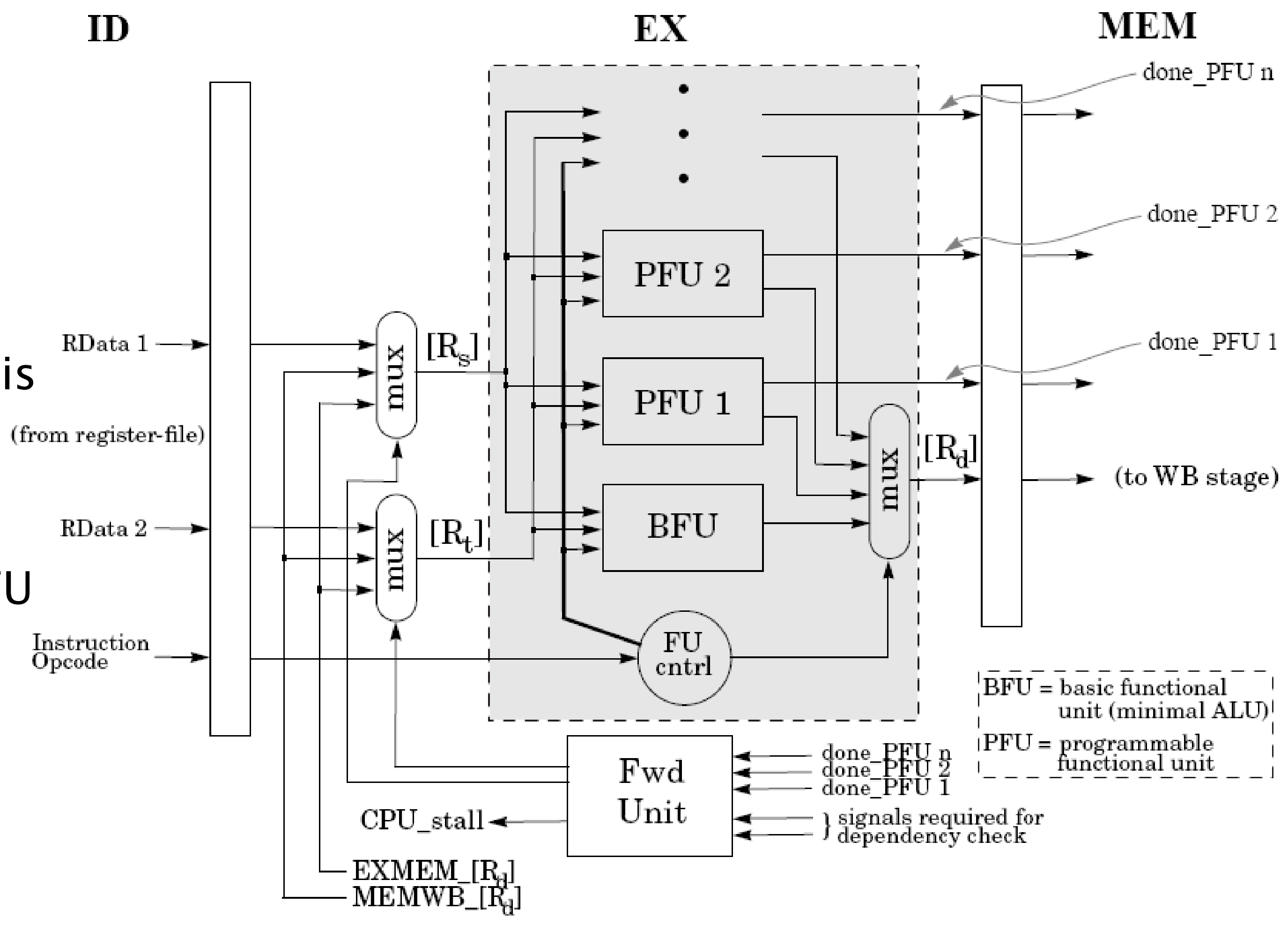
4.6 OneChip

Overview

- Inspired by PRISC
- Tightly coupled functional unit
- Using a MIPS-like core processor
- Supports more complex SIs
- Also supports I/O processing features

Programmable Functional Unit

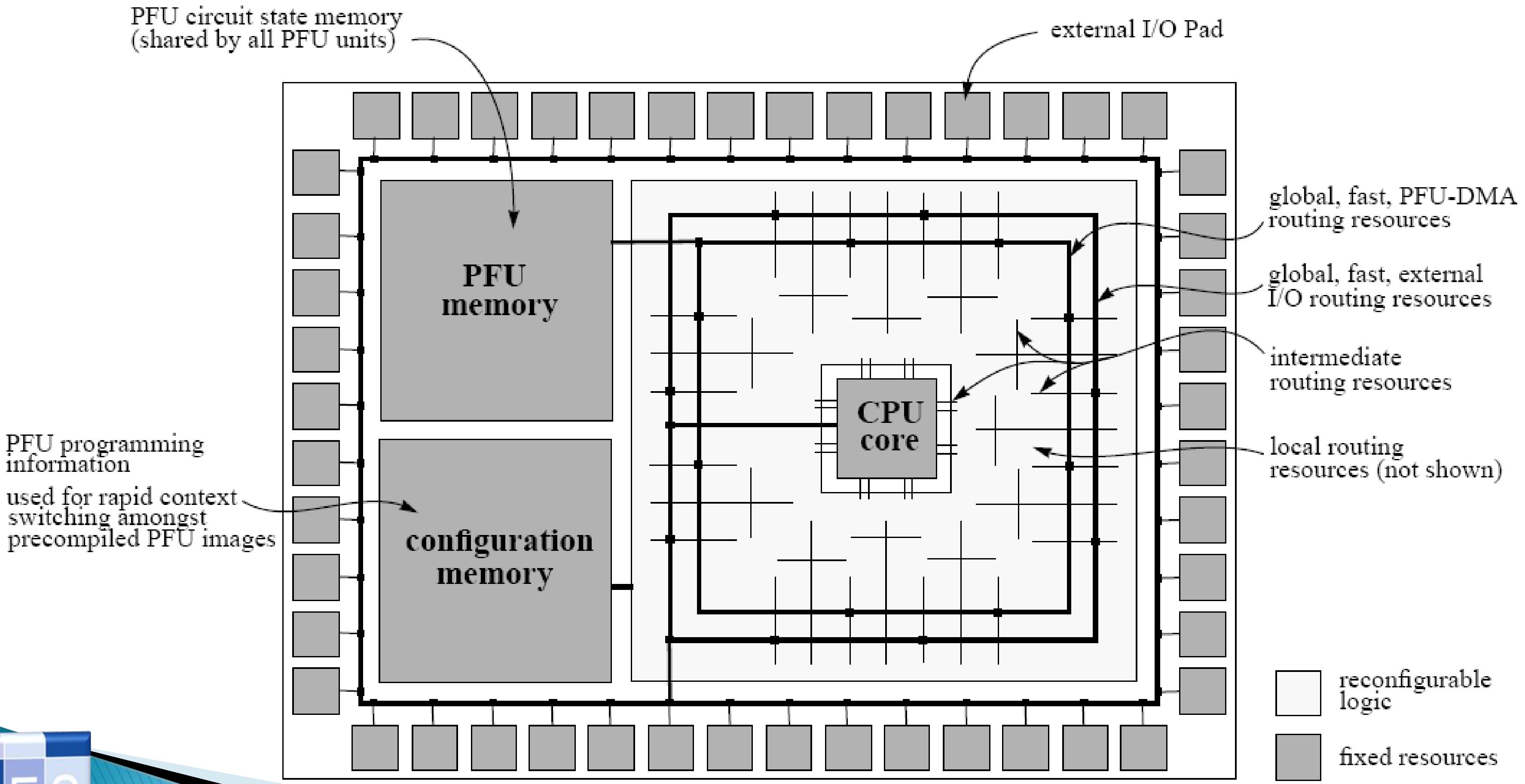
- Can reuse pipeline-internal standard components
  - Data depen-dency analysis check
  - Data fore-warding
  - Multicycle PFU latency
- Binary com-patibility to MIPS



src: [WC96]

Envisioned System

- Core CPU (estimated to be rather small) embedded into reconf. fabric
- Special PFU memory and configuration memory on the chip


PFU circuit state memory (shared by all PFU units)
external I/O Pad
global, fast, PFU-DMA routing resources
global, fast, external I/O routing resources
intermediate routing resources
local routing resources (not shown)
PFU memory
configuration memory
CPU core
PFU programming information
used for rapid context switching amongst precompiled PFU images
reconfigurable logic
fixed resources

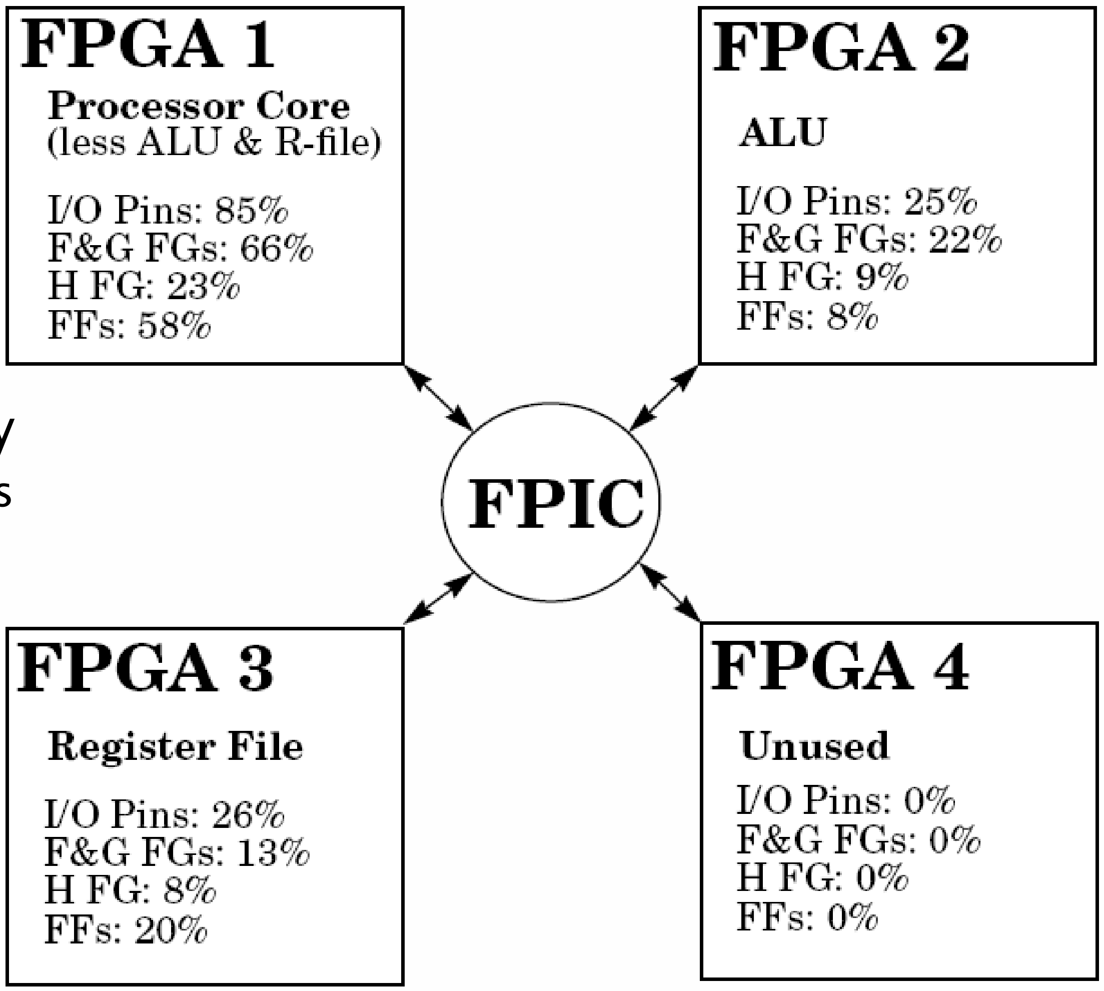
src: [WC96]

Envisioned System (cont'd)

- PFU Configuration Memory
  - The entire reconfigurable fabric is used as one PFU
  - However, the configuration memory contains the configuration data of multiple PFU configurations
  - Reconfiguration from configuration memory is fast and performed on demand (i.e. when the SI is about to execute)
- Circuit state and computational memory
  - General-purpose memory to be used by SI implementations
  - For instance to hold state variables (for multi cycle state-machine based SIs)
  - Or for temporary data storage
- Reconfigurable fabric has access to the I/O pins to implement the protocols of I/O standards, e.g. UART

Prototype

- Based on standard components
  - 4 Xilinx 4010 FPGAs
  - 2 Aptix AX1024 Field-Programmable Inter-connect Chips (FPICs)
  - 4 32Kx9 SRAMs
- Very limited functionality
  - Only 6 (out of 32) registers
  - Using time-division multiplexing to feed up to 8 signals across one physical wire
  - Results in 1.25 MHz operation frequency
  - Only configured during startup



src: [WC96]

4.7 OneChip98

Overview

- Extension of the OneChip project
- Provides memory access for PFUs
  - Note: PFU (Programmable Functional Unit) and RFU (Reconfigurable FU) are used interchangeably; both indicate an SI or the reconfigurable fabric (also called FPGA)
- Providing multiple PFUs (each with potentially multiple contexts)
- Support for superscalar execution
- Support for out-of-order execution

Motivation

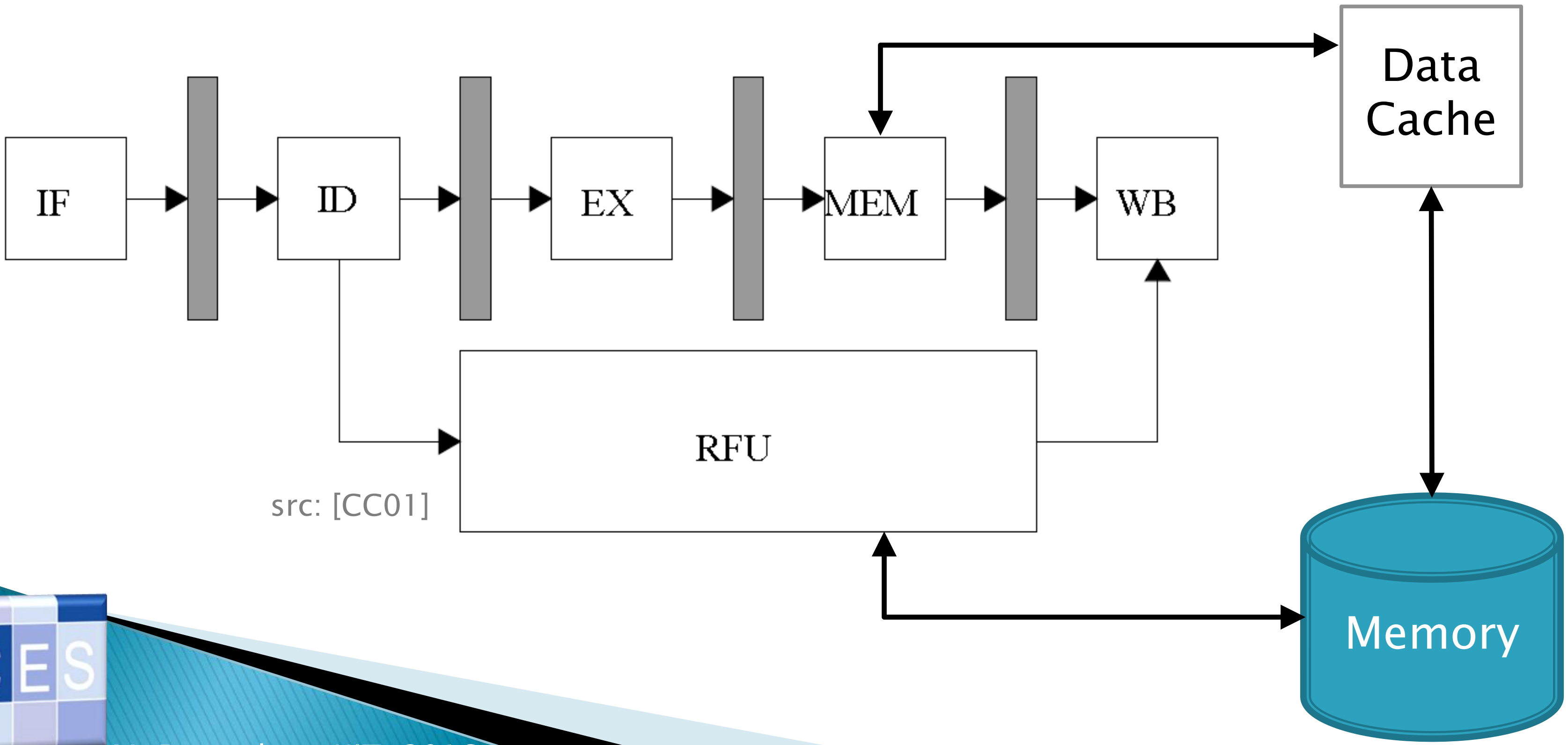
- Observation: SI execution latency is almost certainly greater than one CPU cycle
  - Either because the SI contains a state machine
  - or because the critical path of the SI is too long for the CPU frequency
- What should the pipeline do during SI execution?
  - Simple solution: stall (i.e. wait until SI completion)
  - Alternative: continue executing other instructions in parallel (like it is often done in the coprocessor approach)

Classification SuperScalar/Out-of-order

- Scalar: One operation per cycle (can be pipelined)
- SuperScalar (also called multiple issue processor): potentially multiple instructions per cycle
  - VLIW: the compiler explicitly determines, which instruc-tions shall execute in parallel (**Note:** typically not called superscalar, but somehow belongs to this category)
  - In-order SuperScalar: The issue sequence of instructions in the binary is respected, i.e. if a particular instruction cannot execute (e.g. due to a data dependency) then the instructions that follow that particular instruction are not considered for execution (even if they could)
  - Out-of-order Superscalar: Dynamic re-scheduling of the instructions; potentially executing them in a different sequence than written in the binary

Memory Inconsistency Problems

- Problems may arise when executing SIs that access memory in parallel (or even out-of-order) to normal load/store instructions



src: [CC01]

Memory Inconsistency Problems (cont'd)

Hazard Number	Hazard Type	Actions Taken
1	SI rd after CPU wr	1. Flush SI source addresses from CPU cache when SI issues 2. Prevent SI reads while CPU store instructions are pending
2	CPU rd af-ter SI wr	3. Invalidate SI destination addresses in CPU cache when SI issues 4. Prevent CPU reads from SI destination addresses until SI writes its destination block
3	SI wr after CPU rd	5. Prevent SI writes while CPU load instructions are pending
4	CPU wr af-ter SI rd	6. Prevent CPU writes to SI source addresses until SI reads its source block

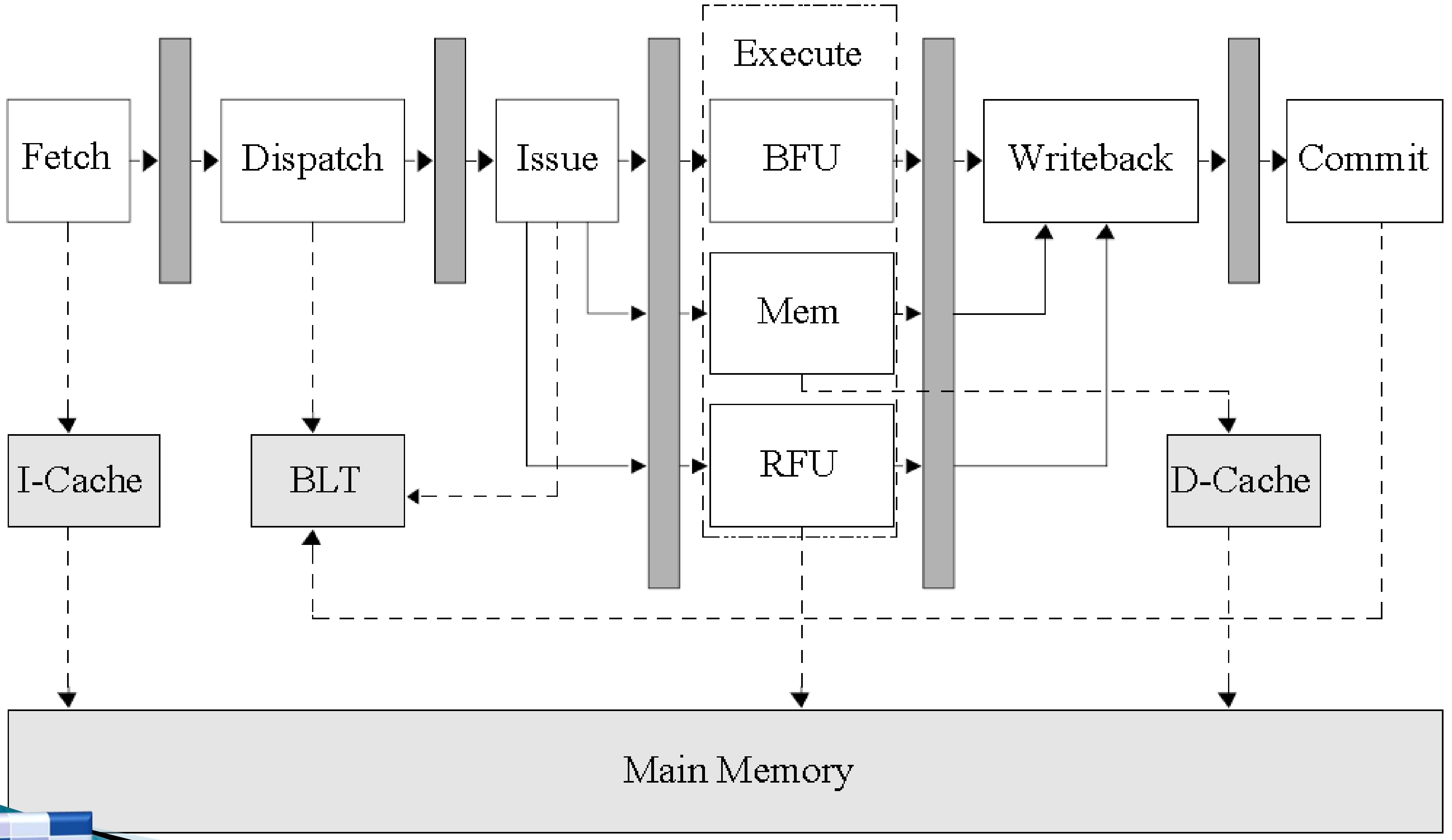
src: [CC01]

Memory Inconsistency Problems (cont'd)

Hazard Number	Hazard Type	Actions Taken
5	SI wr after CPU wr	7. Prevent SI writes while CPU store instructions are pending
6	CPU wr af-ter SI wr	8. Prevent CPU writes to SI destination addresses until SI writes its destination block
7	SI rd after SI wr	9. Prevent SI reads from locked SI destination addresses
8	SI wr after SI rd	10. Prevent SI writes to locked SI source addresses
9	SI wr after SI wr	11. Prevent SI writes to locked SI destination addresses

src: [CC01]

OneChip Out-of-order architecture



src: [CC01]

OneChip Out-of-order architecture (cont'd)

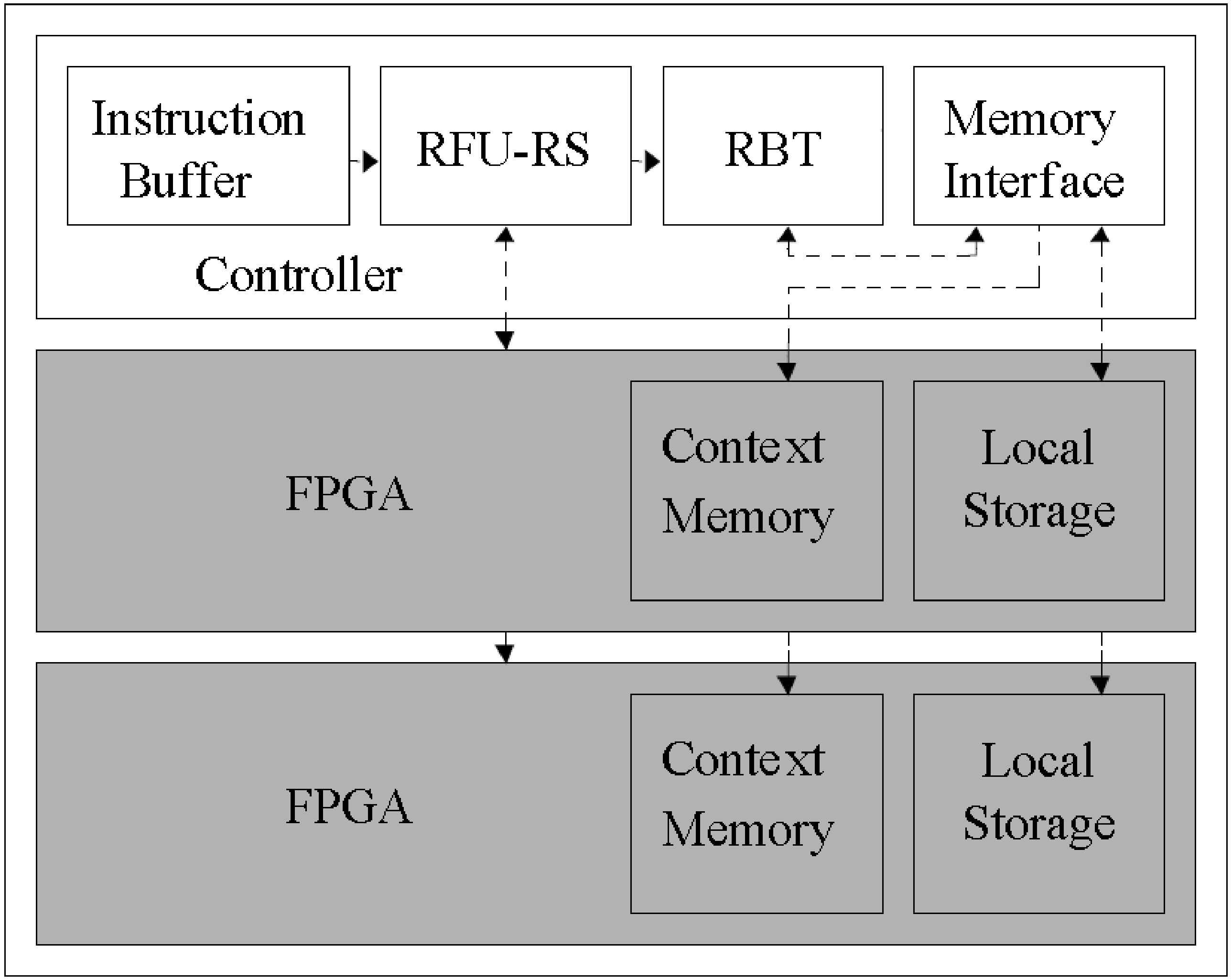
- Fetch stage: fetches instructions from I-Cache to Dispatch queue
- Dispatch stage:
  - instruction decoding
  - register renaming
  - Move instructions from dispatch queue to reservation station for core ISA (BFU), memory (Mem), or SI (RFU)
  - Add entries in the Block Lock Table (BLT, explained later) to lock memory blocks when SIs are dispatched
  - Until here, the instructions are handled in-order
- Issue stage: identifies *ready* instructions from the reservation stations (considering data dependencies, memory consistency etc.) and allow them to proceed in the pipeline
  - Performed out-of-order

OneChip Out-of-order architecture (cont'd)

- Execute stage: executes the instructions in different parallel pipelines for core ISA, memory access, and SIs
- Writeback stage:
  - Move completed operation results to a 'register update unit' (not the register file) and a 'load/store queue'
  - Scan the dependency chain of the completing instructions and wake up any dependent instructions
- Commit stage:
  - Retires instructions in-order (i.e. only Issue, Execute, and Writeback stage operate out-of-order)
  - Commits 'register update unit' data to the register file
  - Commits 'load/store queue' data to the data cache
  - Releases the resources that were used by the instructions
  - Clears BLT entries to remove SI memory locks

RFU composition

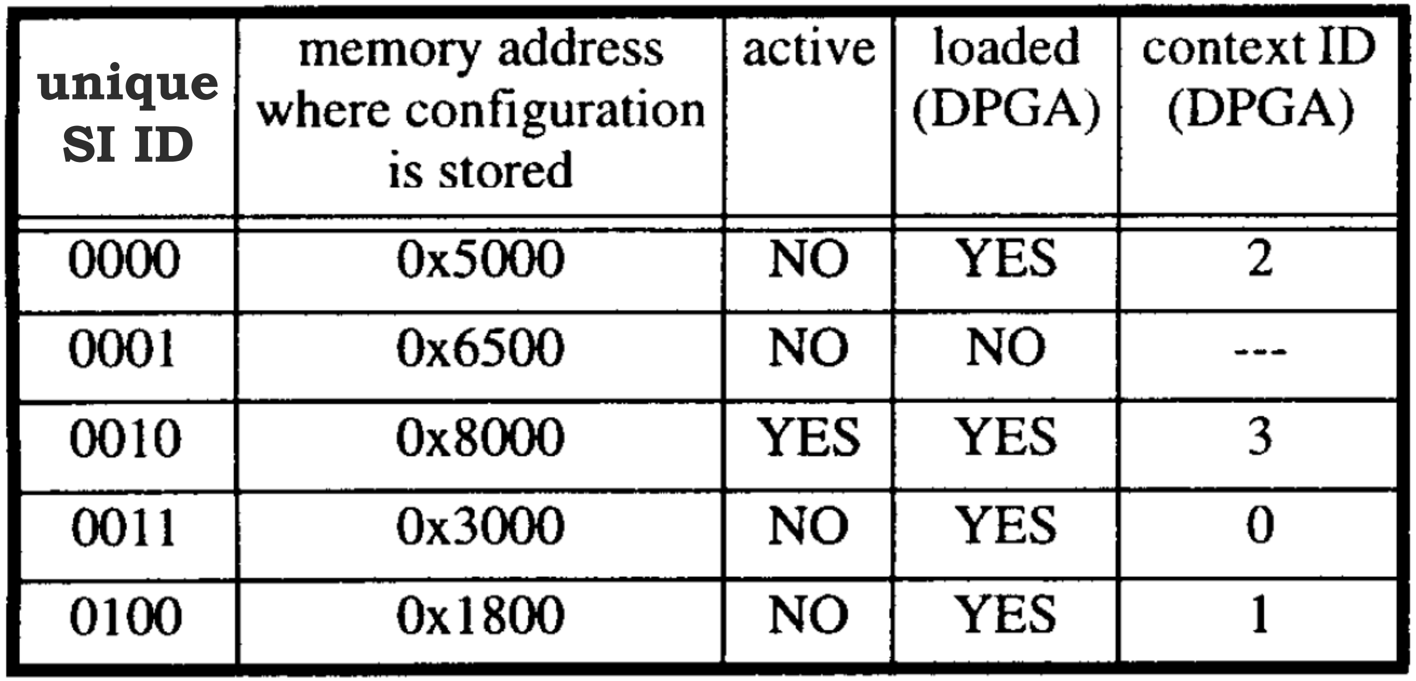
- RS: Reservation Station
- RBT: Recon-figuration Bits Table
  - Acts as con-figuration manager
- Multiple multi-context FPGAs
  - Each containing the configuration of one SI at a time
- Local Storage used for tem-porary results etc.



src: [CC01]

Reconfiguration Bits Table (RBT)

- DPGA: Dynamically Programmable Gate Array; a multi-context configuration cache for the FPGA
- Loaded denotes whether the configuration data is available in the DPGA of the FPGA (each FPGA needs such a table)
  - If yes, then the context ID shows which context it is
- Active denotes whether or not this SI is the currently active configu-ration

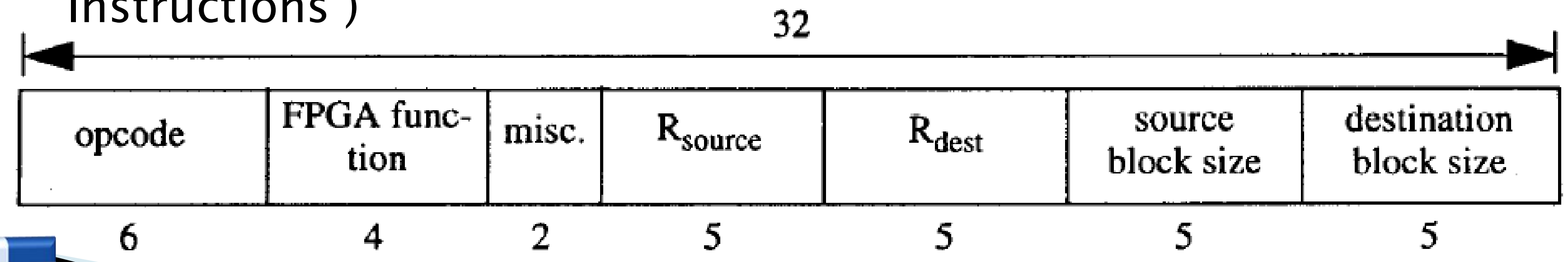


unique SI ID	memory address where configuration is stored	active	loaded (DPGA)	context ID (DPGA)
0000	0x5000	NO	YES	2
0001	0x6500	NO	NO	---
0010	0x8000	YES	YES	3
0011	0x3000	NO	YES	0
0100	0x1800	NO	YES	1

src: modified from [JC99]

Instruction Format

- ‘Opcode’ indicates that this is the instr. format for SIs
- R_{source} and R_{dest} point to registers that contain the source and destination address in data memory
  - ‘source block size’ and ‘destination block size’ indicate the amount of data that will be read and written (important for memory consistency)
- Alternative: when the amount of read data and written data is identical, one of the fields can be used to provide a third register that points to a second source address
- Two ‘FPGA functions’ are reserved for manipulating the RBT and for preloading the bitstream into an FPGA context (like ‘Helper Instructions’)



src: [JC99]

Locking data memory

- How to inform which memory the SI accesses?
- Each SI specifies which memory region will be read and which one will be written
  - Using the base address (32-bit via register)
  - And the block size (5-bit via ‘source block size’)
  - Note: with 5 bit we can distinguish 32 different block sizes
- Observation: Address space is 2^{32} bytes large
  - Idea: block size must be a power of two (2, 4, 8, …, 1G, 2G, 4G; note: ‘1’ is omitted) and the base address must be aligned on a block boundary

Locking data memory (cont'd)

- Example: given the block address (i.e. the base address of the block) below and a 'block size' of $00100_2 = 4_{10}$
  - This indicates an 'expanded' block size of $2^4 = (1<<4) = 10000_2 = 16_{10}$ bytes
  - Note: potential confusion between expectation and calculation. As block size '1' $=2^0$ is not supported, block size X is expanded to block size 2^X and reserves a block of size 2^{X+1}

Block address	0000 0000 0000 0000 0110 1010 0010 0000
Expanded block size	0000 0000 0000 0000 0000 0000 0001 0000
block mask	0000 0000 0000 0000 0000 0000 0001 1111
Masked block address	$\underbrace{\text{0000 0000 0000 0000 0110 1010 001}}_{\text{Any access to the locked region uses the same tag}}$x xxxx

src: [JC99]

Any access to the locked region uses the same tag

Block Lock Table (BLT)

- The BLT stores the information required to determine the locked memory regions

Masked Block Address	SI ID	Source / Destination
0010 100x xxxx xxxx	2	Source
0011 0110 0xxx xxxx	2	Destination
0100 00xx xxxx xxxx	1	Destination
1001 00xx xxxx xxxx	1	Source

src: [JC99]

Block Lock Table (cont'd)

- Instructions are entered to and removed from the BLT in-order
- When an SI is dispatched, its corresponding entries are added to the BLT
- When an SI commits, its entries are removed from the BLT
- The (out-of-order) issue stage probes the BLT for memory locks to determine whether an instruction is 'ready' to execute
  - Also used to flush/invalidate cache lines, depending on the hazard type

Benchmarks

- Simulated architecture
  - 4 instructions can be fetched, decoded, issued, and committed per cycle
  - 16-entry register update unit
  - 8-entry load/store queue
  - 4 integer ALUs
  - 1 integer mul/div unit
  - 2 memory ports
  - 4 floating point ALUs
  - 1 floating point mul/div unit
- Note: rather hardware-rich architecture
  - Still they obtain speedup by adding the RFU

Benchmarks (cont'd)

Legend:

All numbers are 'times', e.g. '2' means 2 times faster

- A: in-order GPP
- B: in-order OneChip
- C: out-of-order GPP
- D: out-of-order OneChip

Application	Data size	OneChip inorder (A/B)	OneChip outorder (C/D)	Outorder original (A/C)	Outorder OneChip (B/D)	Total (A/D)
JPEG encode	Small	1.37	1.34	2.29	2.25	3.08
	Medium	1.36	1.33	2.29	2.24	3.05
	Large	1.38	1.35	2.33	2.29	3.15
JPEG decode	Small	1.29	1.20	2.47	2.29	2.96
	Medium	1.29	1.19	2.52	2.34	3.01
	Large	1.25	1.16	2.53	2.35	2.93
ADPCM encode	Small	22.38	17.04	1.54	1.18	26.31
	Medium	26.25	17.85	1.62	1.10	28.94
	Large	29.92	20.57	1.56	1.07	32.02
ADPCM decode	Small	18.32	13.47	1.60	1.18	21.55
	Medium	21.79	14.81	1.62	1.10	24.02
	Large	24.43	16.27	1.61	1.07	26.13
PEGWIT encrypt	Small	1.46	1.43	2.09	2.06	3.00
	Medium	1.33	1.36	2.20	2.26	3.00
	Large	1.16	1.24	2.48	2.65	3.07
PEGWIT decrypt	Small	1.40	1.42	2.08	2.11	2.95
	Medium	1.28	1.32	2.27	2.35	3.00
	Large	1.13	1.18	2.62	2.72	3.08
MPEG2 decode	Small	4.69	5.44	2.02	2.34	11.00
	Medium	5.07	5.70	2.07	2.33	11.82
	Large	5.23	5.91	2.08	2.36	12.33
MPEG2 encode	Small	1.16	1.14	1.90	1.87	2.16
	Medium	1.30	1.26	1.86	1.81	2.34
	Large	1.28	1.24	1.87	1.81	2.32

Benchmarks (cont'd)

- Observed limited potential for execution core ISA in parallel to SIs
    - Their benchmarks never used more than one RFU hardware
    - 5 independent instructions for JPEG decoder
    - 11 independent instructions for JPEG encoder
    - The SI has a latency of 128 cycles
- Only the JPEG application benefited by using the BLT
- Recommendation of the authors: rather than using the memory consistency scheme (i.e. the BLT hardware) it should be sufficient to stall the CPU as soon as it is about to perform a memory access while an SI executes

Summary

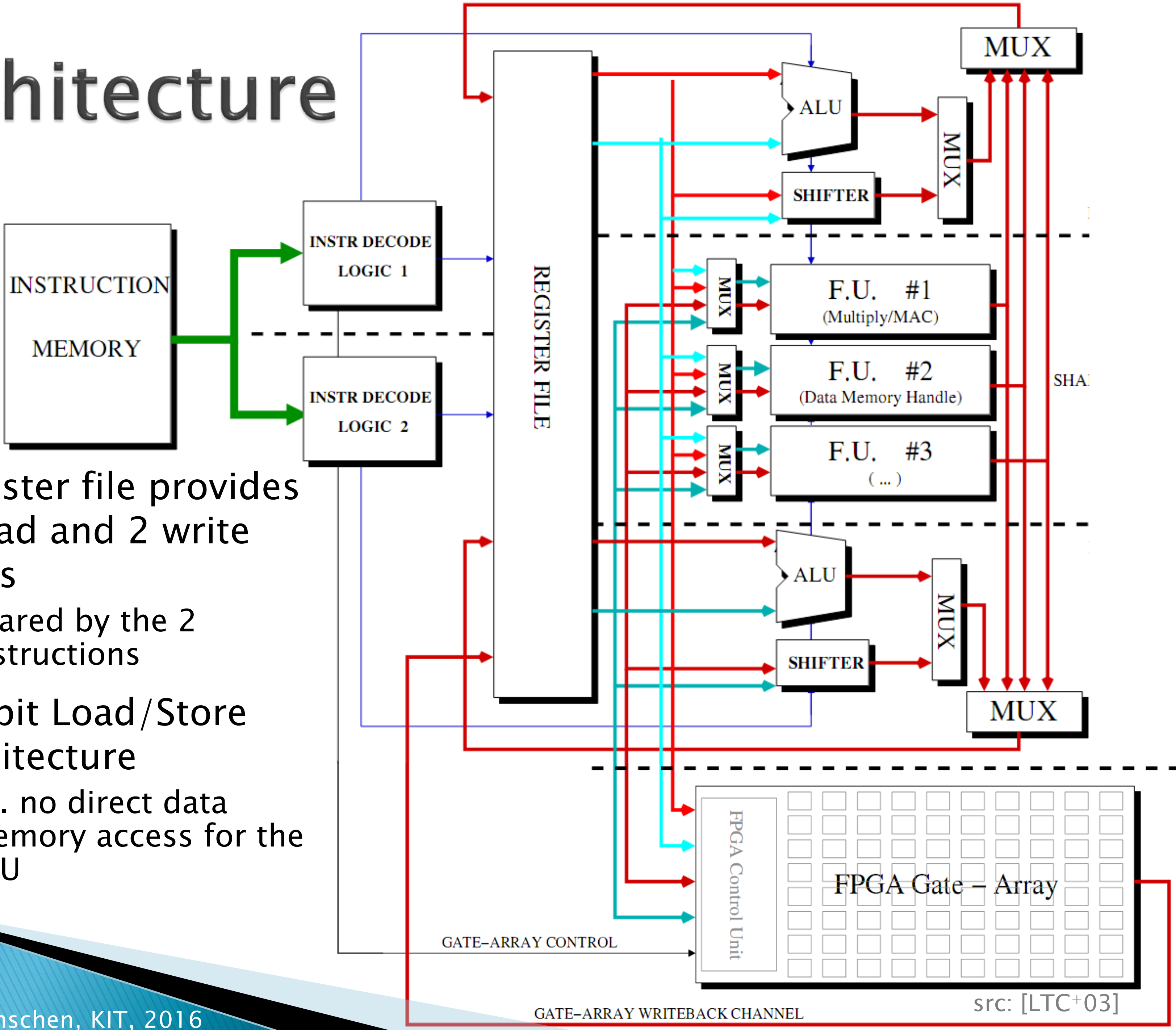
- Introduced Superscalar out-of-order execution to reconfigurable SIs
  - Automatic management to avoid memory inconsistency problems
- Reported speedup: 2x – 32x
  - MPEG-2 encoder: 2x
  - MPEG-2 decode: 11x
  - ADPCM encode: 32x
  - Comparing out-of-order issue with RFU against in-order issue (still superscalar) without RFU
  - Based on simulation
- Rather incomplete hardware prototype

4.8 XiRisc : eXtended Instruction-set RISC

Overview

- A VLIW processor, enhanced with a tightly-coupled reconfigurable functional unit (RFU)
  - Processor fetches 2 instructions per cycle that are executed concurrently
  - Using a classical 5-stage RISC pipeline
- Inspired by Garp
- Embedded in a System-on-Chip: XiSystem
  - Provides an additional eFPGA to handle I/O communication or to be used as reconfigurable Coprocessor
- Developed in collaboration with STMicroelectronics that provided an actual tape-out (i.e. chip) of the developed processor/system

Architecture



- Register file provides 4 read and 2 write ports
  - shared by the 2 instructions
- 32-bit Load/Store architecture
  - i.e. no direct data memory access for the RFU

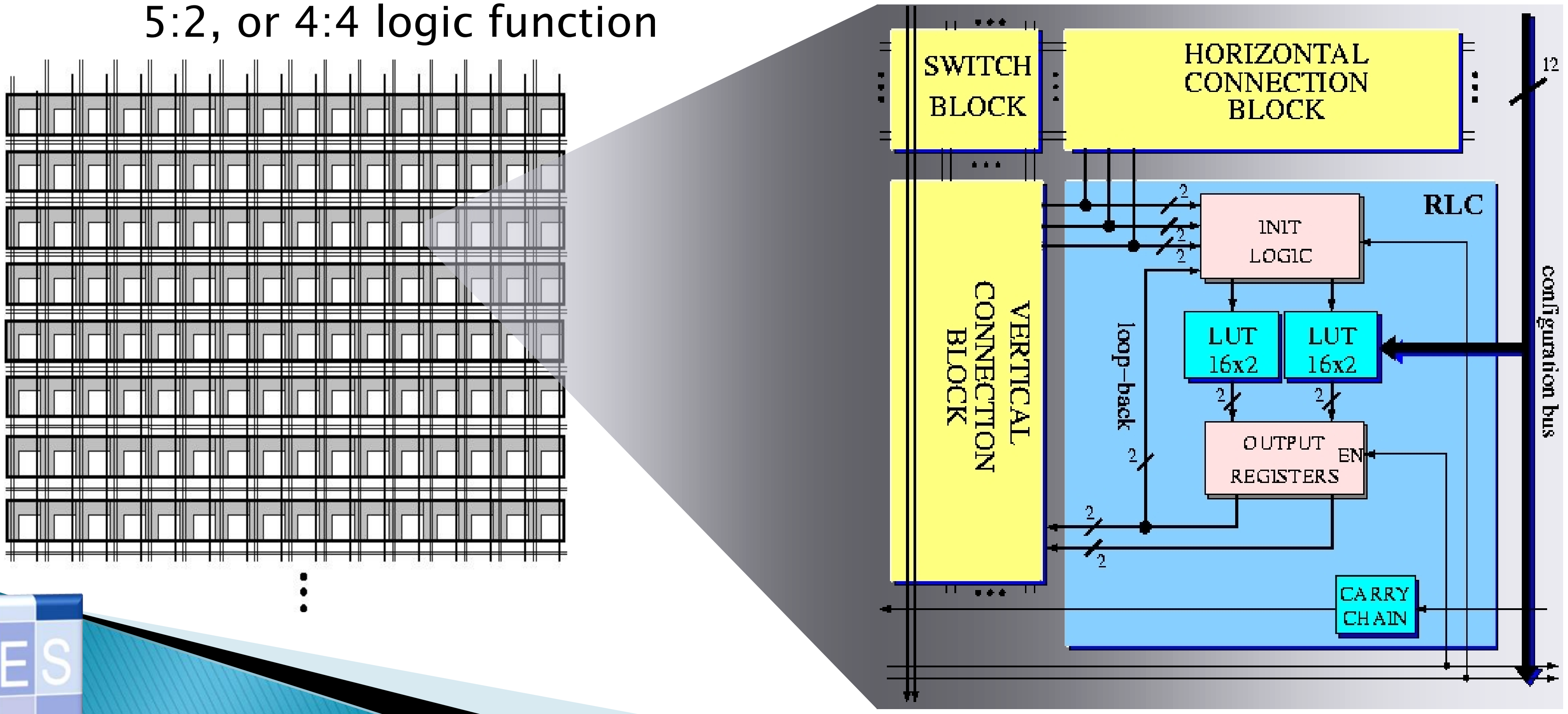
src: [LTC+03]

Architecture (cont'd)

- Fully bypassed architecture, i.e. data forwarding to reduce the effects of data dependencies
- Hardwired FUs + an additional pipelined RFU
  - Called 'Pipelined Configurable Gate Array' (*p*GA or PiCoGA)
  - Supports multi-cycle instructions
  - Can hold an internal state across several computations
  - Synchronization and consistency is realized by hardware stall logic based on a register locking mechanism (for read-after-write hazards, i.e. when the PiCoGA wants to write a register it is locked until writing is completed)

PiCoGA

- Two-dimensional array of LUT-based Reconfigurable Logic Cells (RLCs)
    - 16x24 RLC array
    - RLC contains two 4:2 LUTs that can be combined to form a 6:1, 5:2, or 4:4 logic function



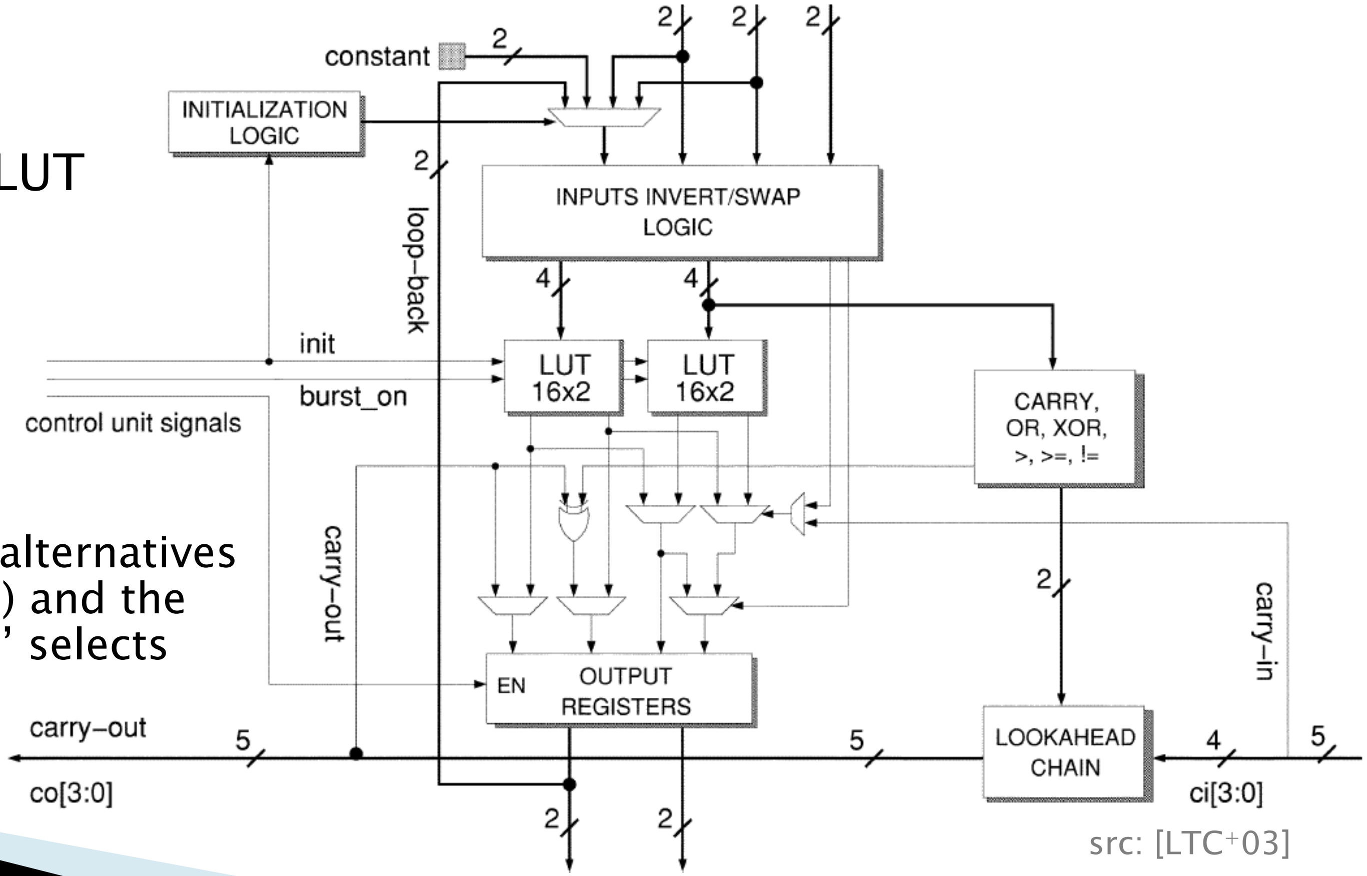
src: [LTC+03]

PiCoGA (cont'd)

- Each row implements a possible stage of a customized pipeline that executes in parallel to the normal FUs
    - 16 RLCs per row; 2-bit granularity
    - 8 horizontal channel pairs for communication within one row
    - 12 vertical channel pairs for communication between the rows
- A sequence of SIs can be processed in a pipelined way
- Up to 4x32-bit input data and up to 2x32-bit output data from/to register File

Reconfigurable Logic Cell (RLC)

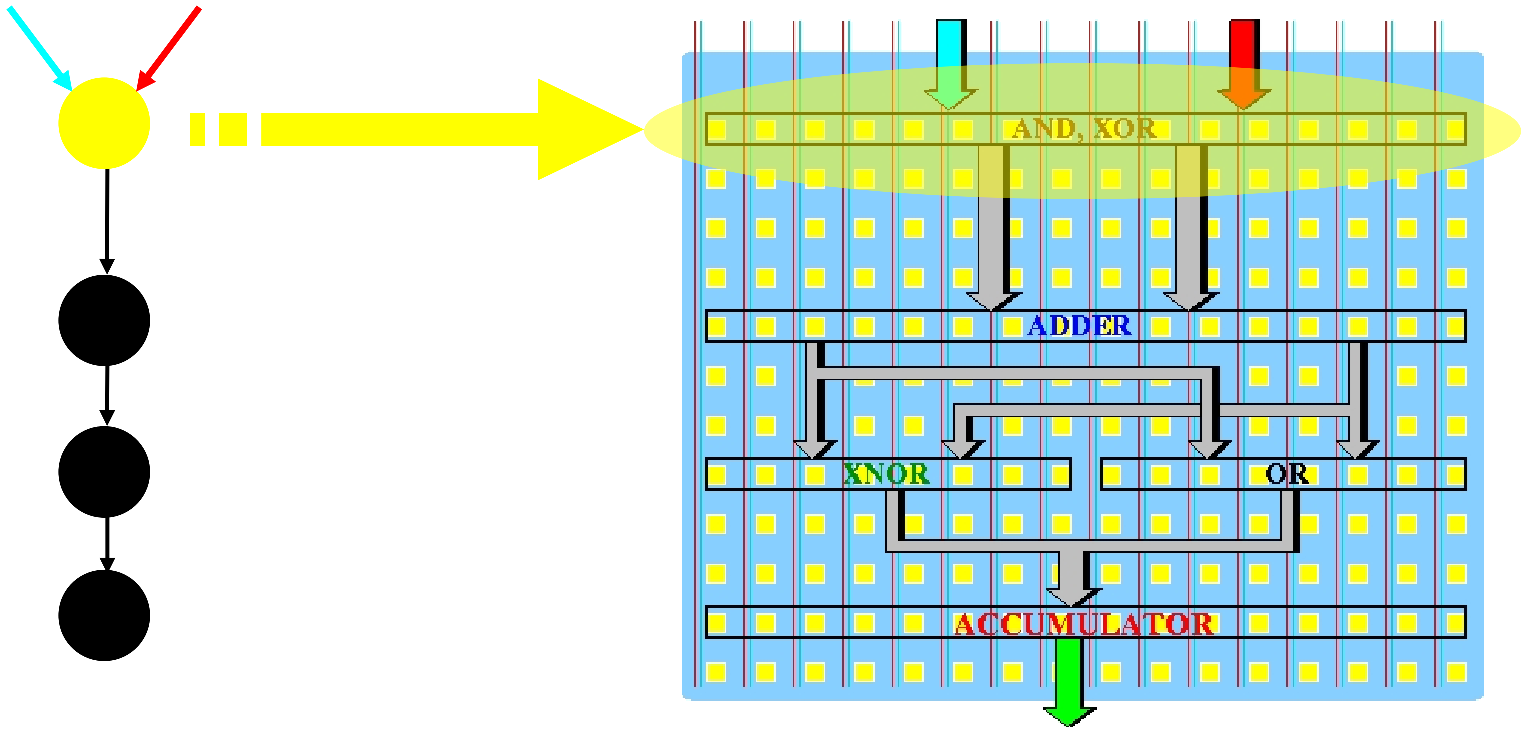
- RLC-internal loop-back to cascade the 2 LUTs or to hold a state (e.g. accumulate)
  - Constant input (selected by MUX) to initialize state
- Extra Logic for carry chain
- 1 register per LUT output
- An RLC can implement a 2-bit adder
  - The 2 LUTs compute both alternatives (carry in 0 or 1) and the actual 'carry in' selects the result)



src: [LTC+03]

Example: Pipelined SI execution with preserved state

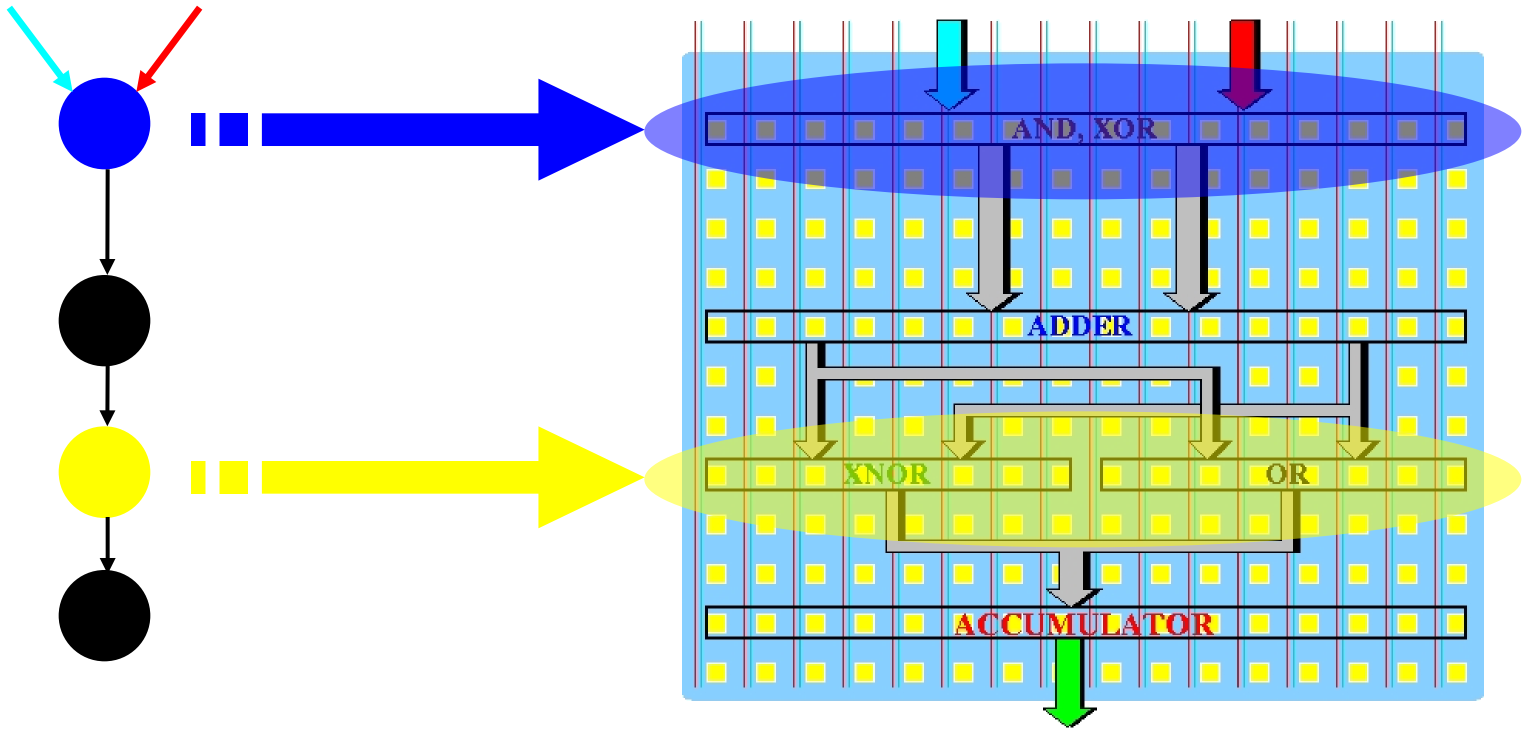
- Row elaboration is activated by an embedded control unit
- Execution enable signal for of each pipeline stage



- PiCoGA operation latency depends on the executed operation

Example: Pipelined SI execution with preserved state

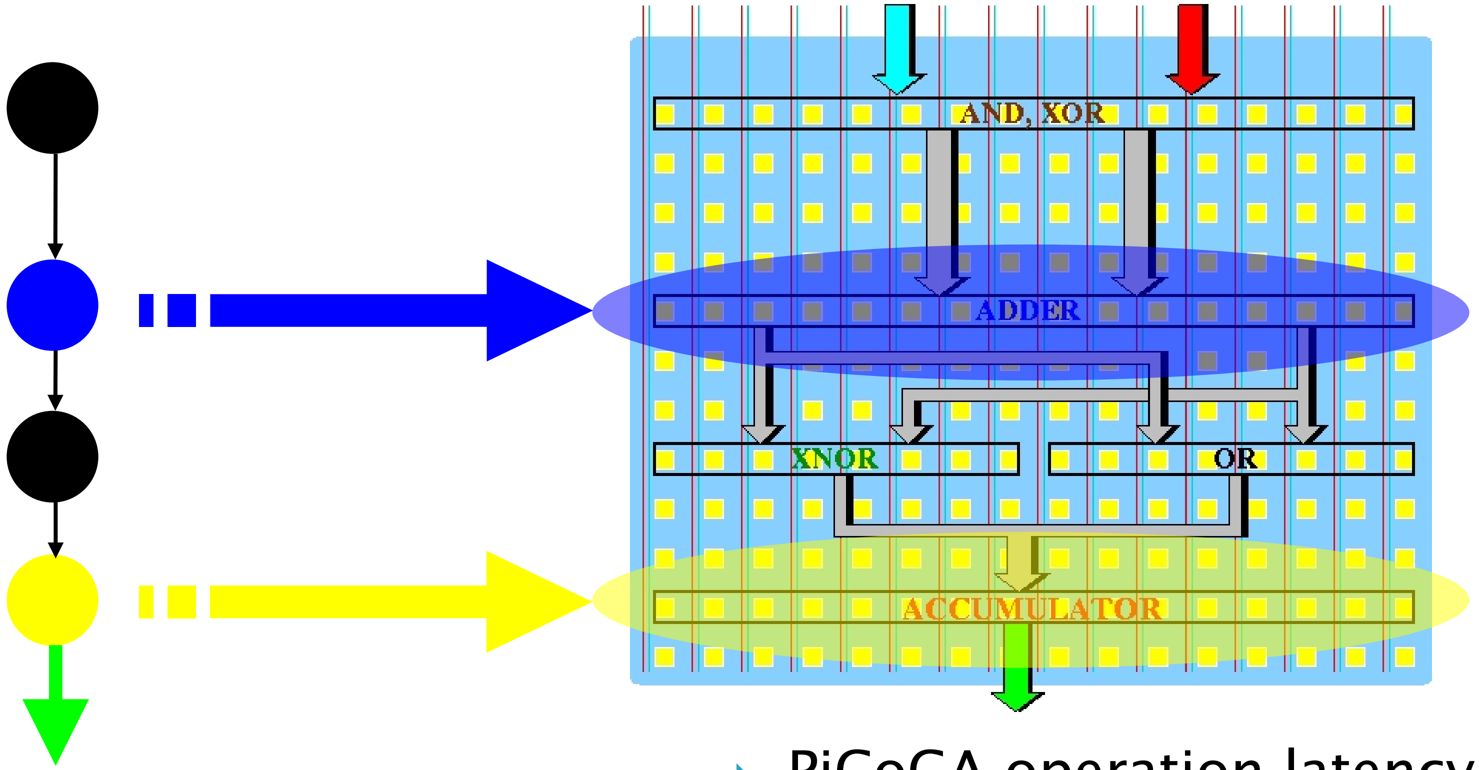
- Row elaboration is activated by an embedded control unit
- Execution enable signal for of each pipeline stage



- PiCoGA operation latency depends on the executed operation

Example: Pipelined SI execution with preserved state

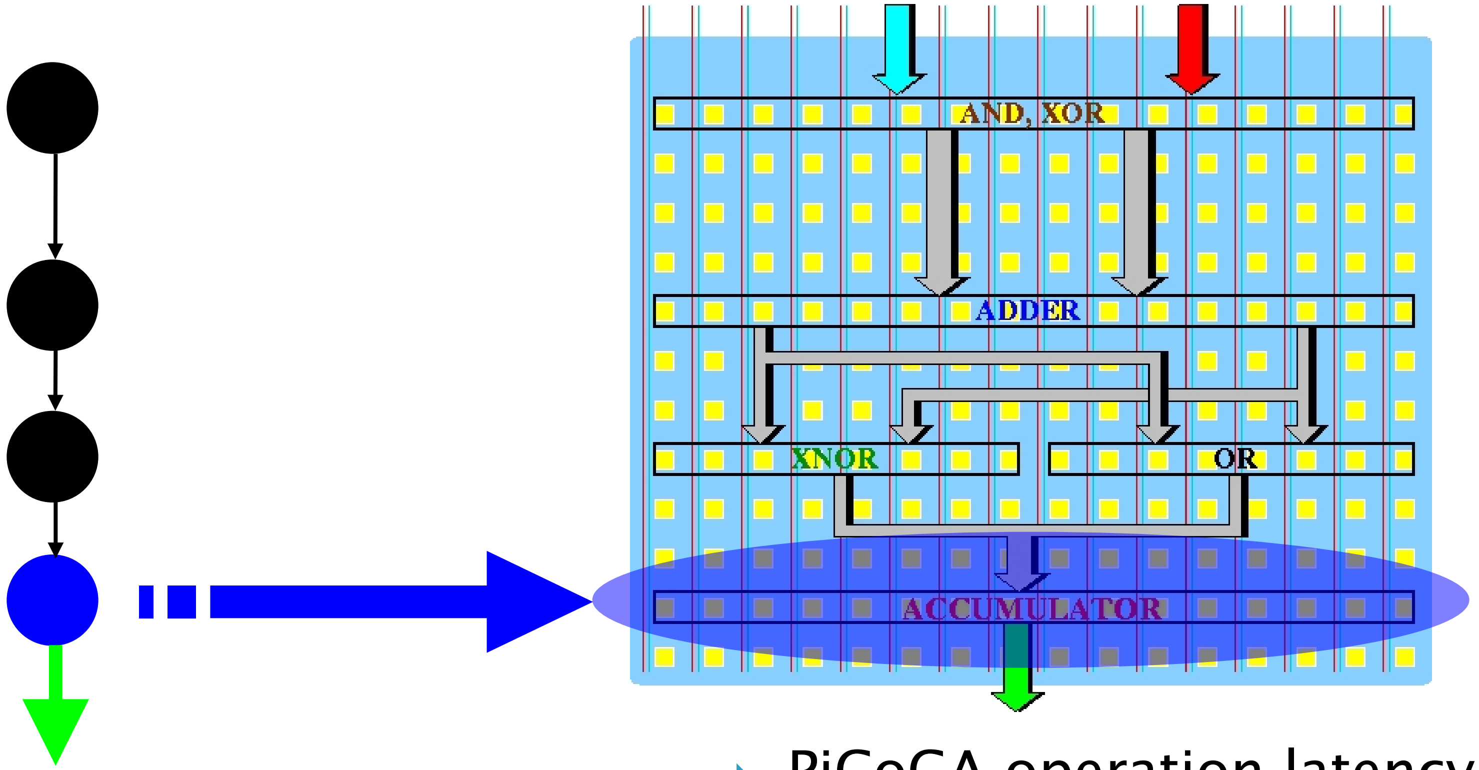
- Row elaboration is activated by an embedded control unit
- Execution enable signal for of each pipeline stage



- PiCoGA operation latency depends on the executed operation

Example: Pipelined SI execution with preserved state

- Row elaboration is activated by an embedded control unit
- Execution enable signal for of each pipeline stage



- PiCoGA operation latency depends on the executed operation

Instruction-set extension

- *p*GA-load: load a configuration into the array
- *p*GA-free: remove a configuration
- *p*GA-op: execute an SI
  - 32-bit variant that allows to execute a second instruction (VLIW) in parallel but only offers 2 source registers
  - 64-bit variant that uses both VLIW slots but therefore provides 4 source registers

pGA-load	region specification	configuration specification

32-bit pGA-op	**Source 1**	**Source 2**	**Dest 1**	**Dest 2**	operation specification

64-bit pGA-op	**Source 1**	**Source 2**	**Source 3**	**Source 4**	**Dest 1**	**Dest 2**	operation specification

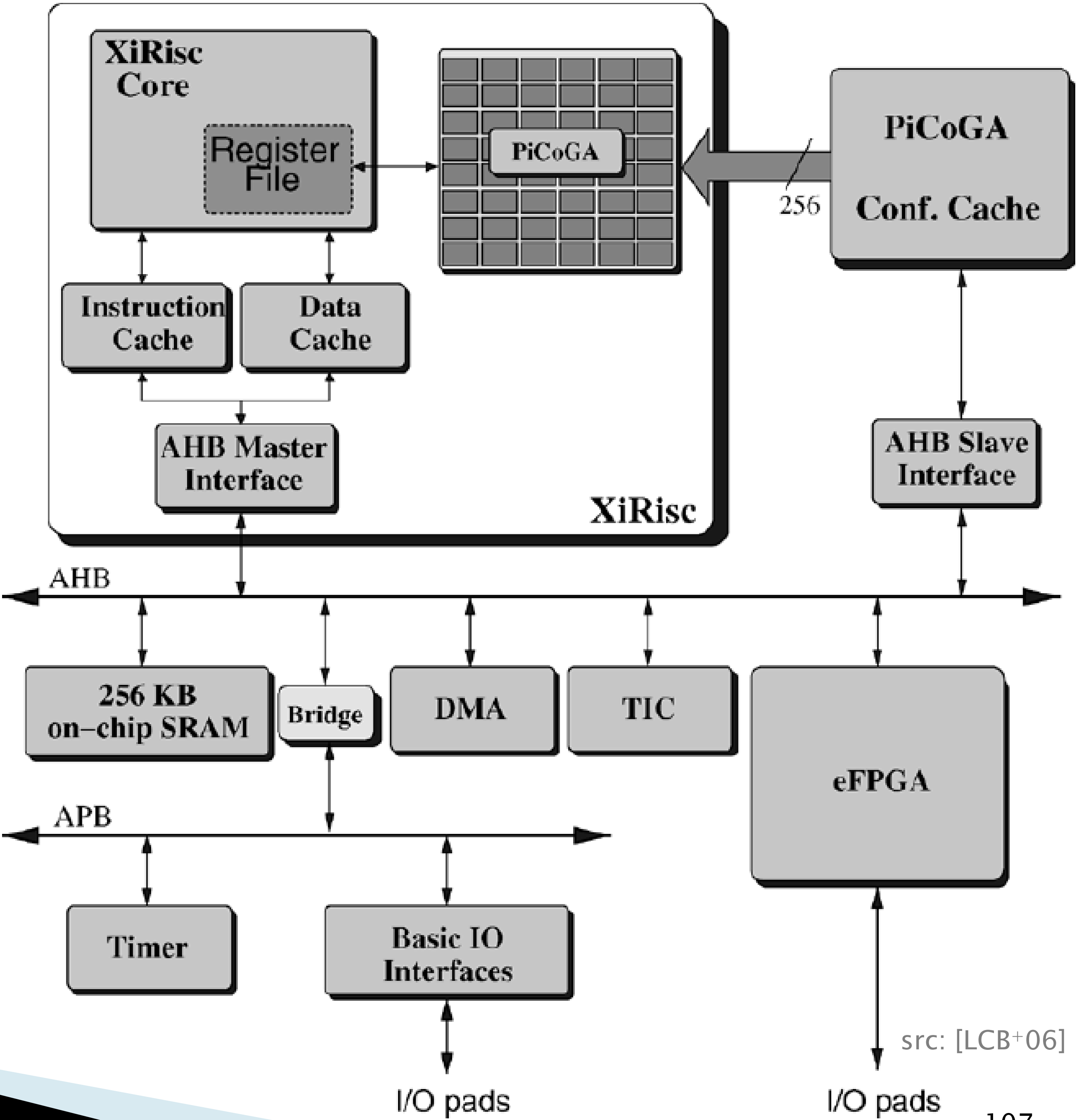
src: [CCG+03]

Configuration Caching

- Storing 4 configurations for each RLC
  - Single-cycle context switch
- Row-wise partial reconfiguration
- Interface between core CPU and reconfigurable fabric buffers the ‘configuration load’ instructions that are then performed one after the other
  - Thus, the core CPU does not need to wait (stall) for reconfiguration completion
  - SIs may execute during reconfiguration
- 192-bit bus to 2nd level on-chip configuration cache
  - 16 cycles to receive a complete configuration
  - In a later work extended to 256-bit and attached to AHB

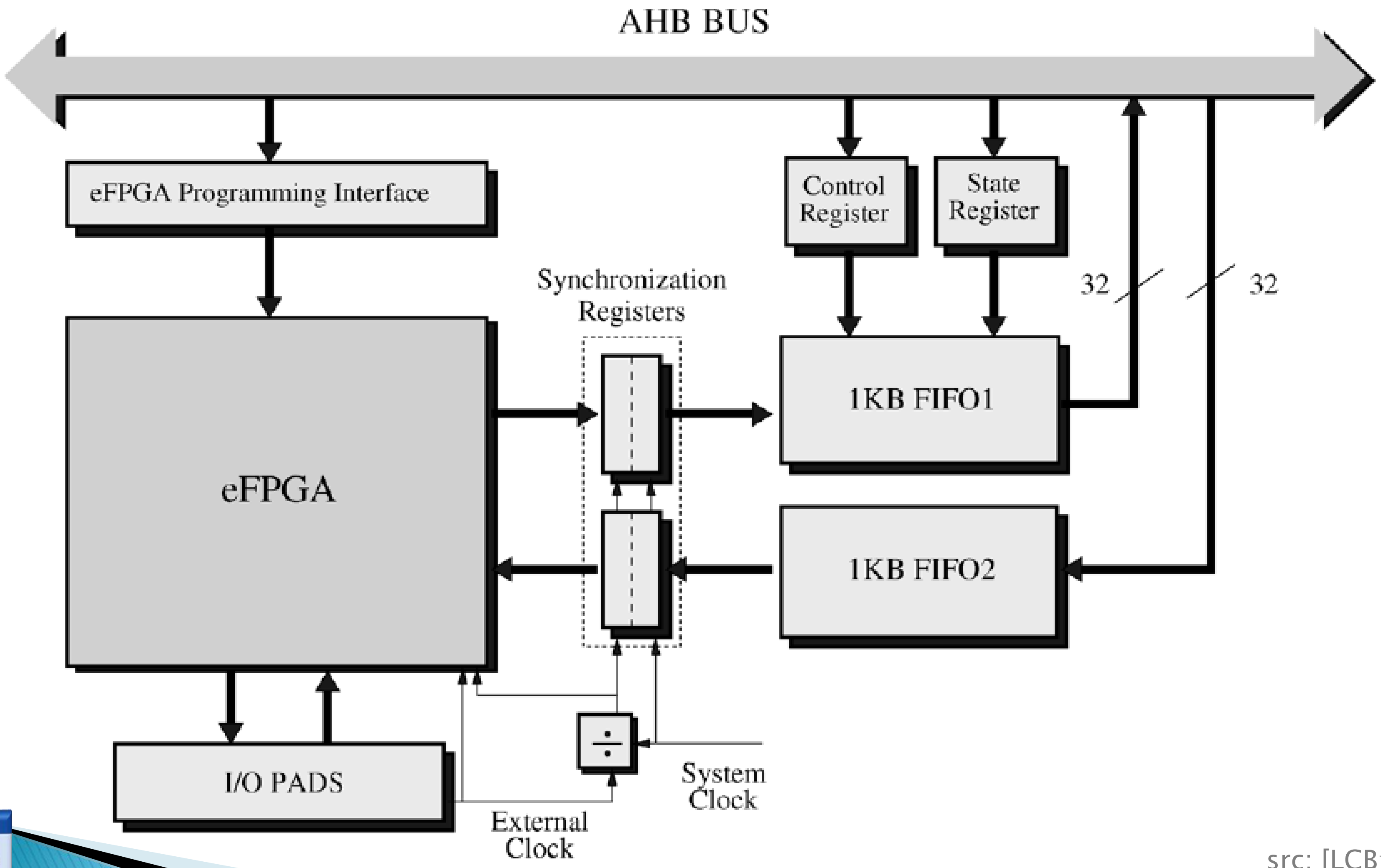
4.9 Xi-System

- Embedded FPGA (eFPGA):
  - fine-grained (1-bit granu-larity)
  - Homoge-neous
  - Single-context
  - Configurable Pull-up/Pull-down I/O pads
- TIC: standard Test Interface Controller



src: [LCB+06]

Connecting the eFPGA

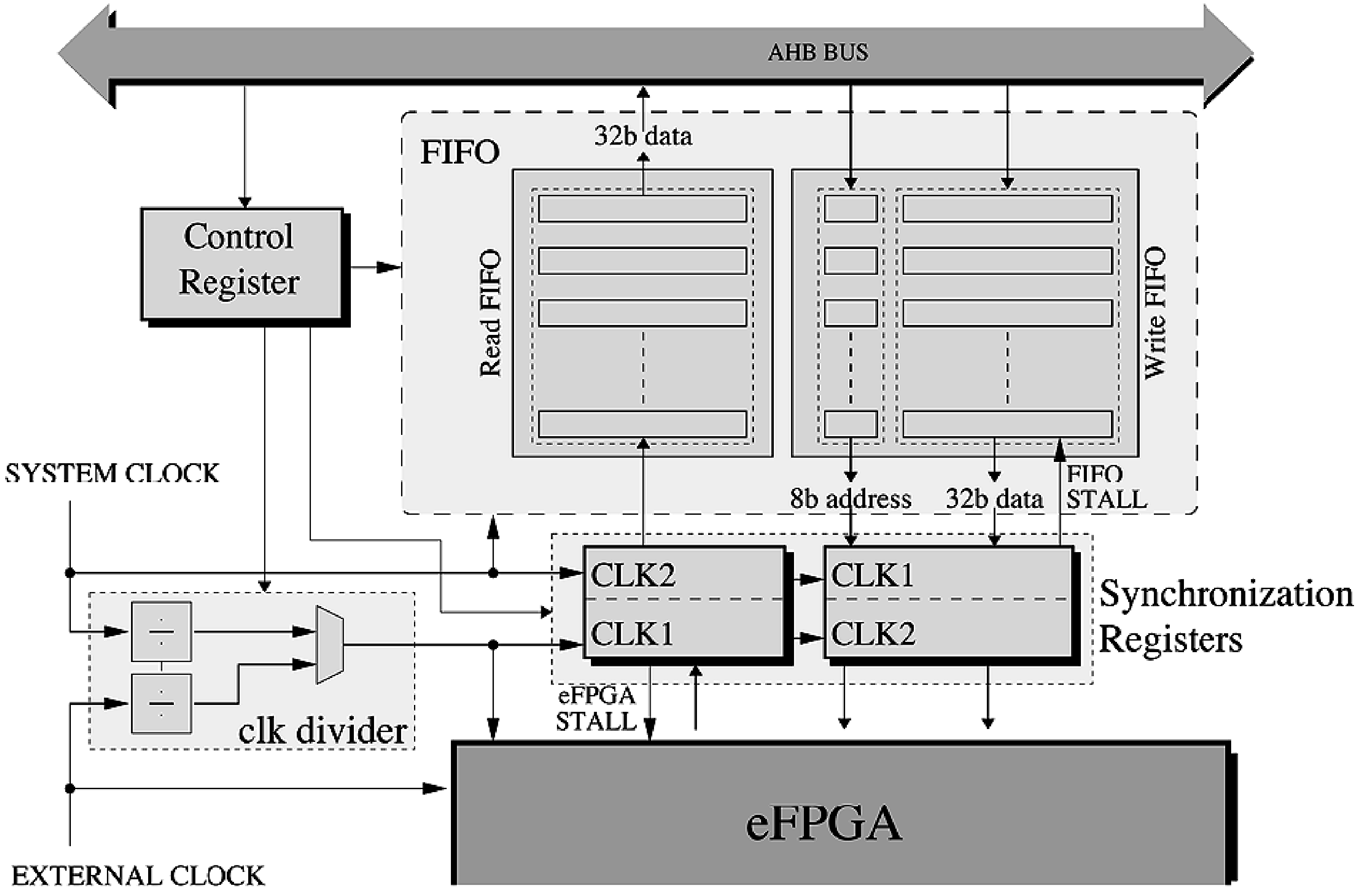


src: [LCB+06]

Connecting the eFPGA (cont'd)

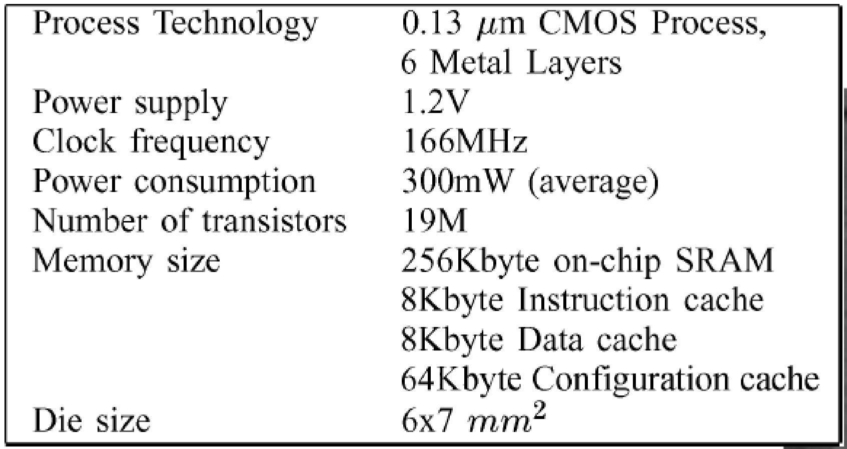
- The eFPGA is memory-mapped to 256 reserved addresses
    - i.e. an access to theses particular addresses goes to the eFPGA instead of the memory
- 2 unidirectional FIFOs with 32 32-bit locations each
    - The 'Write FIFO' (i.e. to eFPGA) additionally stores the lower 8-bits of the address to identify which memory mapped address was accessed
- The eFPGA/FIFOs can generate interrupts to control the DMA unit
    - This allows that the eFPGA can be used as autonomous data-stream Coprocessor
- The eFPGA can be clocked from different sources (system- or external clock or both) to adapt to different critical paths on the eFPGA and to different bandwidth requirements
    - For instance, the eFPGA can use a slowed-down version of the system clock (using the clock dividers) to process data in parallel, then serialize the results and use a high-frequency external clock to provide the required bandwidth

Connecting the eFPGA (cont'd)



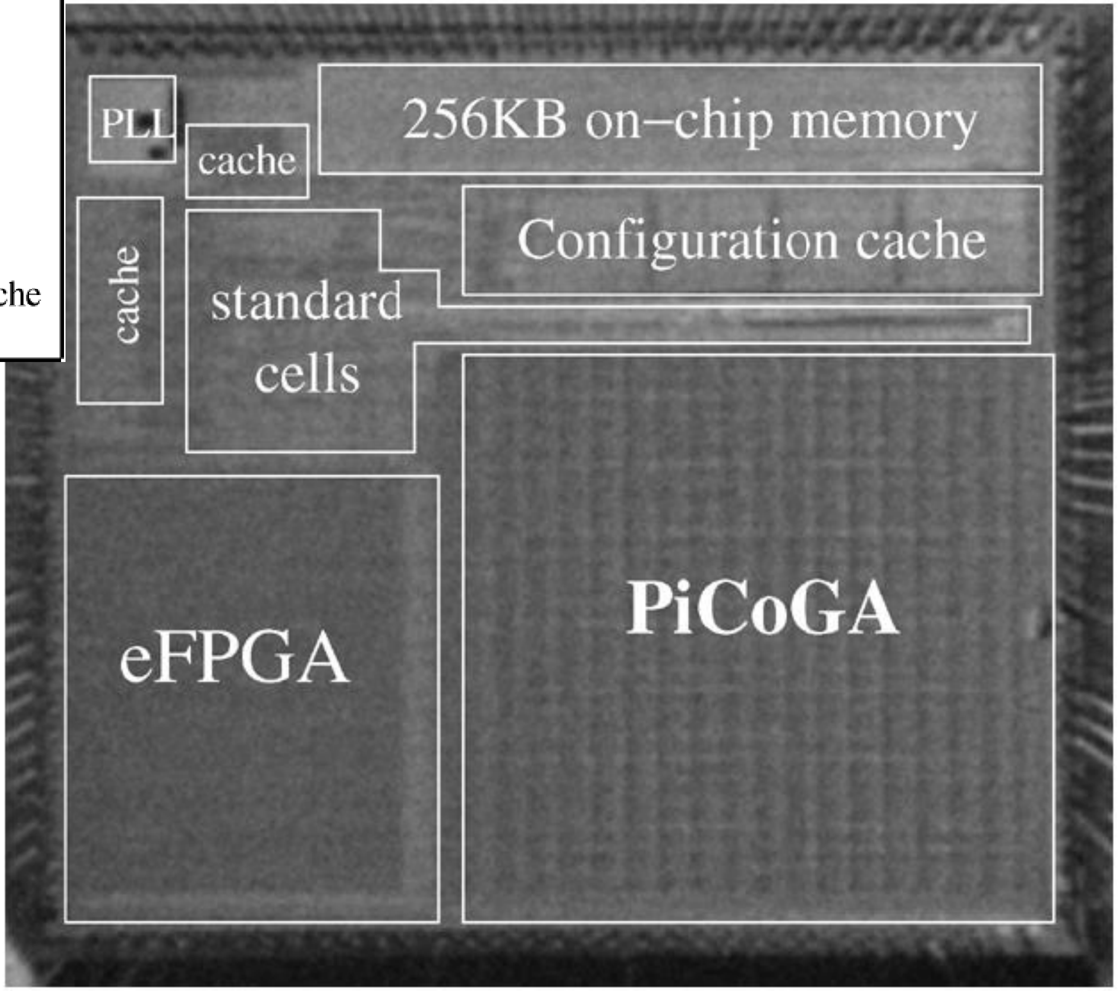
src: [LCB+06]

Fabricated Chip



Process Technology	0.13 μm CMOS Process, 6 Metal Layers
Power supply	1.2V
Clock frequency	166MHz
Power consumption	300mW (average)
Number of transistors	19M
Memory size	256Kbyte on-chip SRAM 8Kbyte Instruction cache 8Kbyte Data cache 64Kbyte Configuration cache
Die size	6x7 mm^2

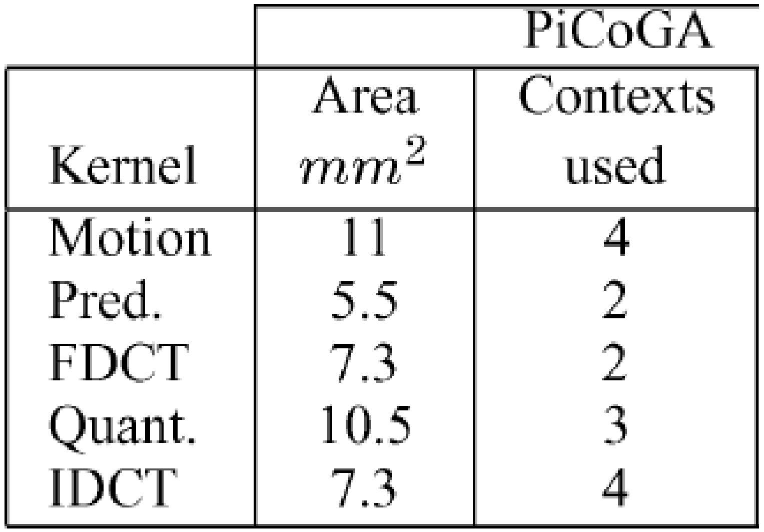
- eFPGA occupies 6 mm² for 15-Kgate capacity
- PiCoGA occupies 11 mm² for 15.4-Kgate capacity
  - Mainly due to multiple contexts



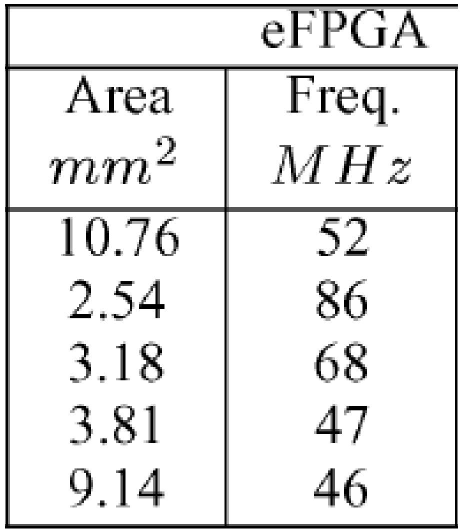
src: [LCB+06]

Comparing PiCoGA vs. eFPGA

- Using an MPEG-2 encoder application



	PiCoGA	
Kernel	Area mm^2	Contexts used
Motion	11	4
Pred.	5.5	2
FDCT	7.3	2
Quant.	10.5	3
IDCT	7.3	4



eFPGA	
Area mm^2	Freq. MHz
10.76	52
2.54	86
3.18	68
3.81	47
9.14	46

src: [LCB+06]

Comparing XiSystem with other CPUs

- Compared with 'XiRisc without PiCoGA':

Algorithm	MOPS	Speed-up	MOPS/mm^2	Energy saving
MPEG-2 enc.	568	5x	15.79	66%
MPEG-2 dec.	185	1.5x	5.13	35%
Turbodec.	1127	8.6x	31.31	84%
DES	537	13.5x	14.93	89%
IDCT	224	1.5x	6.22	41%
Motion est.	1725	14.8x	47.92	74.8%

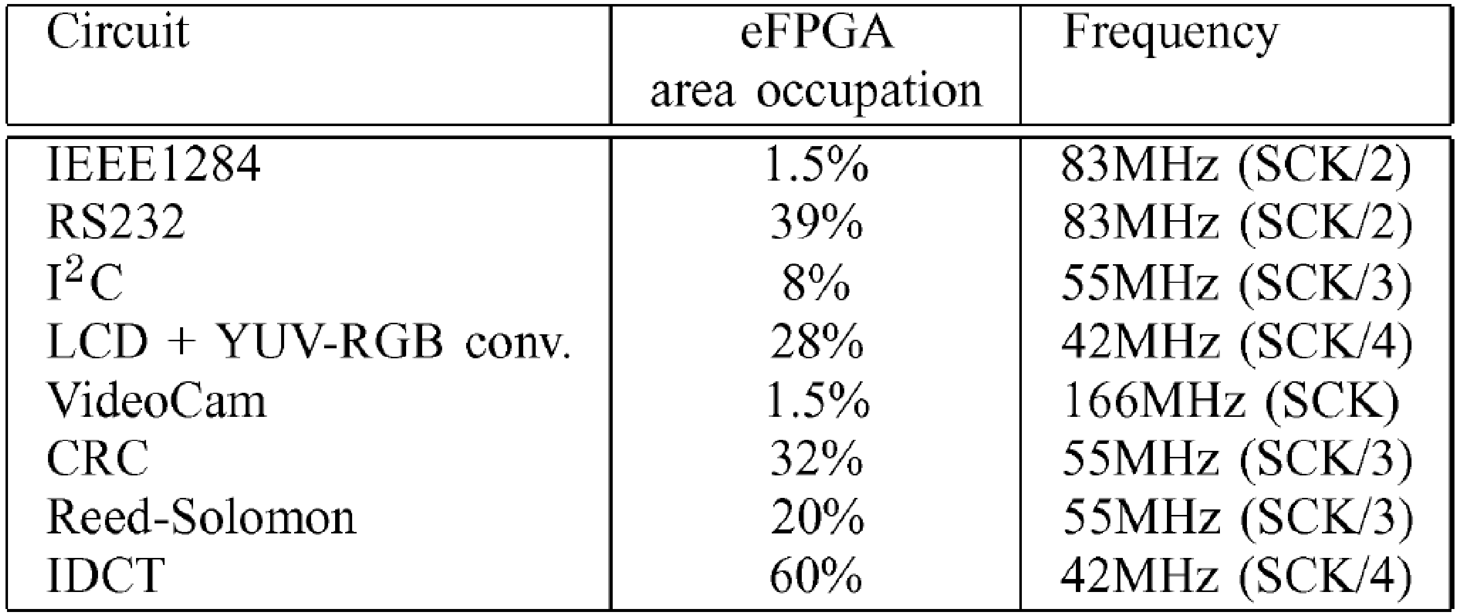
- Compared with TI C6713: VLIW running at 225 MHz, issuing up to 8 integer instructions per cycle

	XiRisc vs. TI C6713	
Algorithm	Speed-up	Energy saving
MPEG-2 encoder	1.86x	63.3%
MPEG-2 decoder	1.22x	58.4%
Turbodecoder	2.26x	76.4%
DES	6.72x	94.1%
IDCT	1.94x	73.8%
Motion estimation	2.29x	64.8%

src: [LCB+06]

Performance of the eFPGA

- Used for I/O peripherals and for Coprocessor computation



Circuit	eFPGA area occupation	Frequency
IEEE1284	1.5%	83MHz (SCK/2)
RS232	39%	83MHz (SCK/2)
I^2C	8%	55MHz (SCK/3)
LCD + YUV-RGB conv.	28%	42MHz (SCK/4)
VideoCam	1.5%	166MHz (SCK)
CRC	32%	55MHz (SCK/3)
Reed-Solomon	20%	55MHz (SCK/3)
IDCT	60%	42MHz (SCK/4)

src: [LCB+06]

Summary

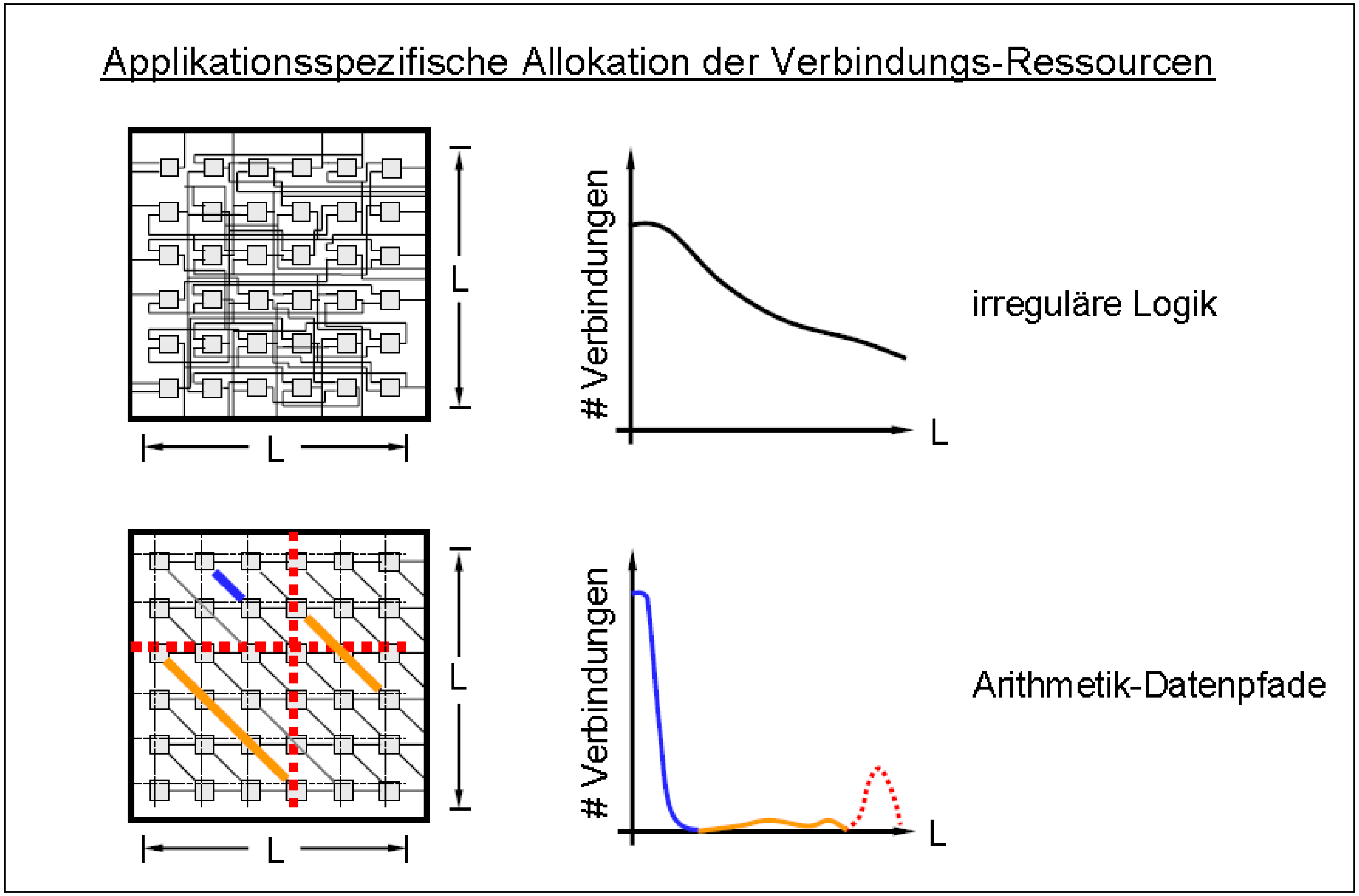
- XiRisc: VLIW RISC architecture enhanced by run-time reconfigurable function unit
- PiCoGA: pipelined, runtime configurable, row-oriented array of LUT-based cells
- Reported Speedup ranges from 1.5x to 15.8x
- Up to 89% energy consumption reduction
- Embedded in a System-on-Chip: XiSystem
- Developed in collaboration with STMicro-electronics that provided an actual tape-out of the developed chip

4.10 New FPGA Architectures

4.10.1 Domain-optimized eFPGAs

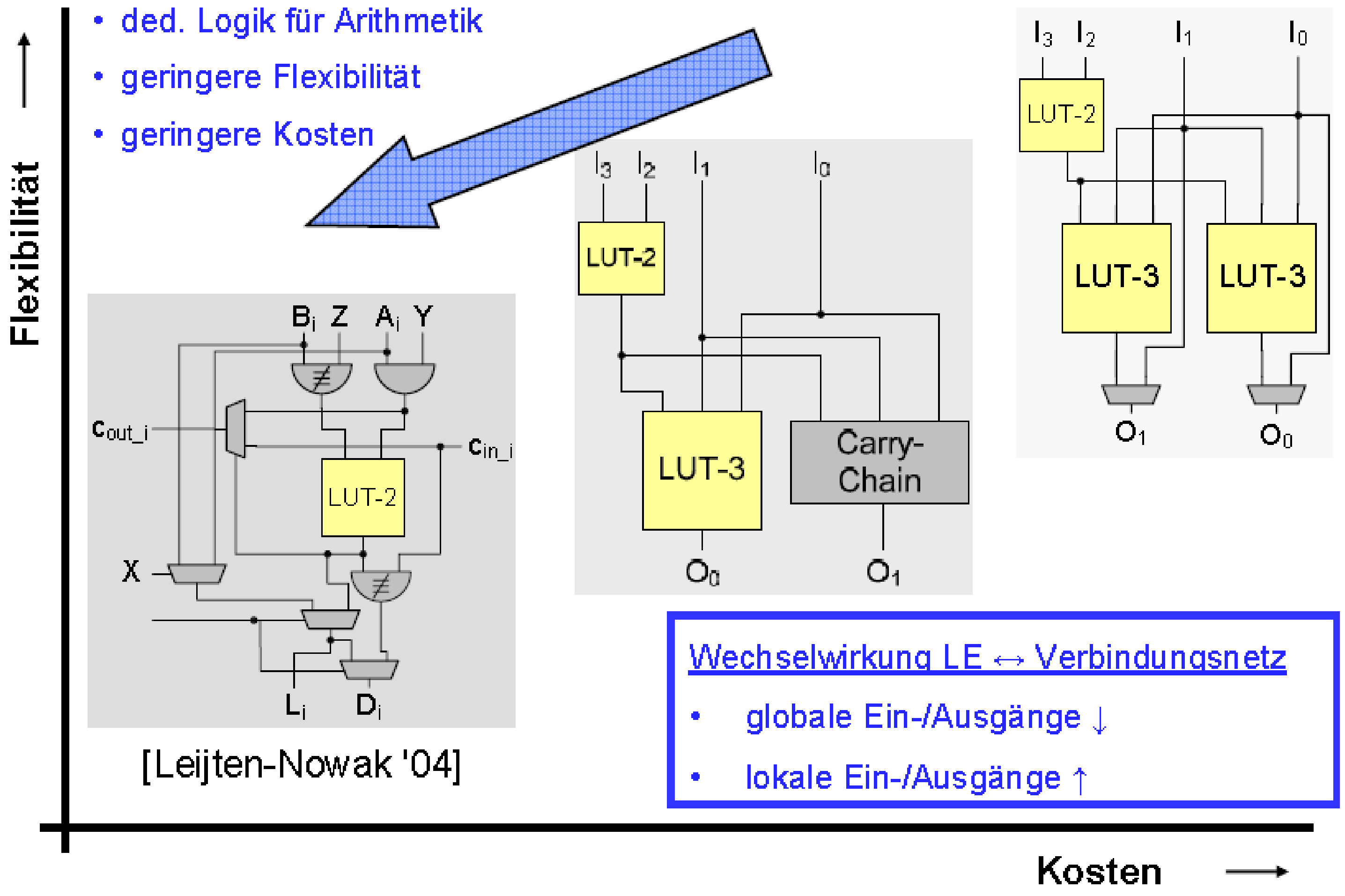
- Dedicated for a specific application or domain, e.g. arithmetic operations
- Still programmable/reconfigurable
  - but typically operating at lower efficiency when targeting a different domain
- Problem: Design-space-exploration
  - Which FPGA structure is suitable/best?
  - How can the tools (e.g. place&route) handle that structure?
- Realized by Architecture Description Languages (ADLs)
  - Automatic generation of physical layout that implements the FPGA (for ASIC)
  - Automatic generation of HDL code that describes the FPGA (for simulation)
  - Automatic generation of place&route tools targeting the FPGA

Routing Ressourcen



src: Prof. Noll, RWTH Aachen

Logic Elements



- ded. Logik für Arithmetik
- geringere Flexibilität
- geringere Kosten

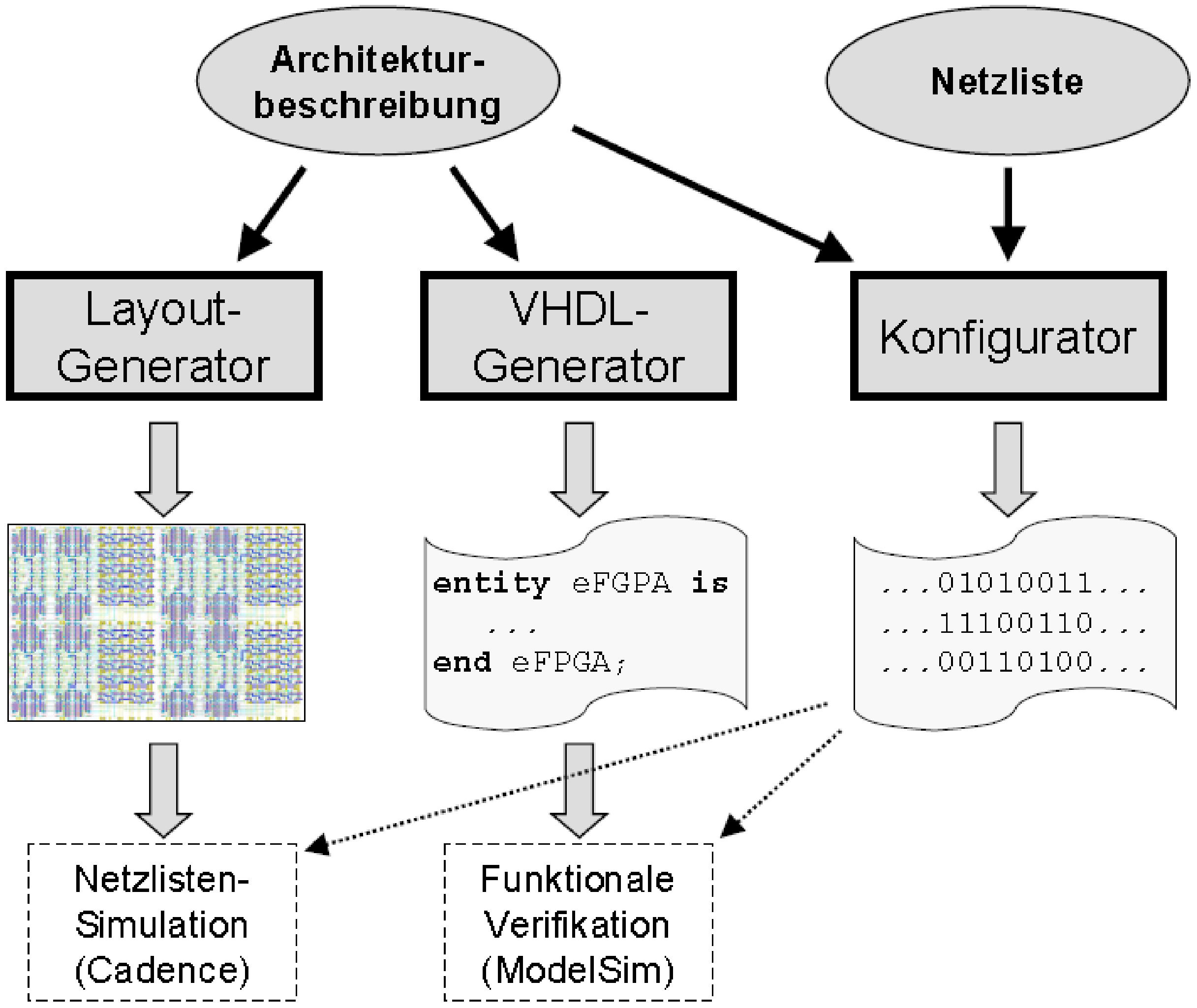
[Leijten-Nowak '04]

Wechselwirkung LE ↔ Verbindungsnetz

- globale Ein-/Ausgänge ↓
- lokale Ein-/Ausgänge ↑

src: Prof. Noll, RWTH Aachen

Overview Toolflow



src: Prof. Noll, RWTH Aachen

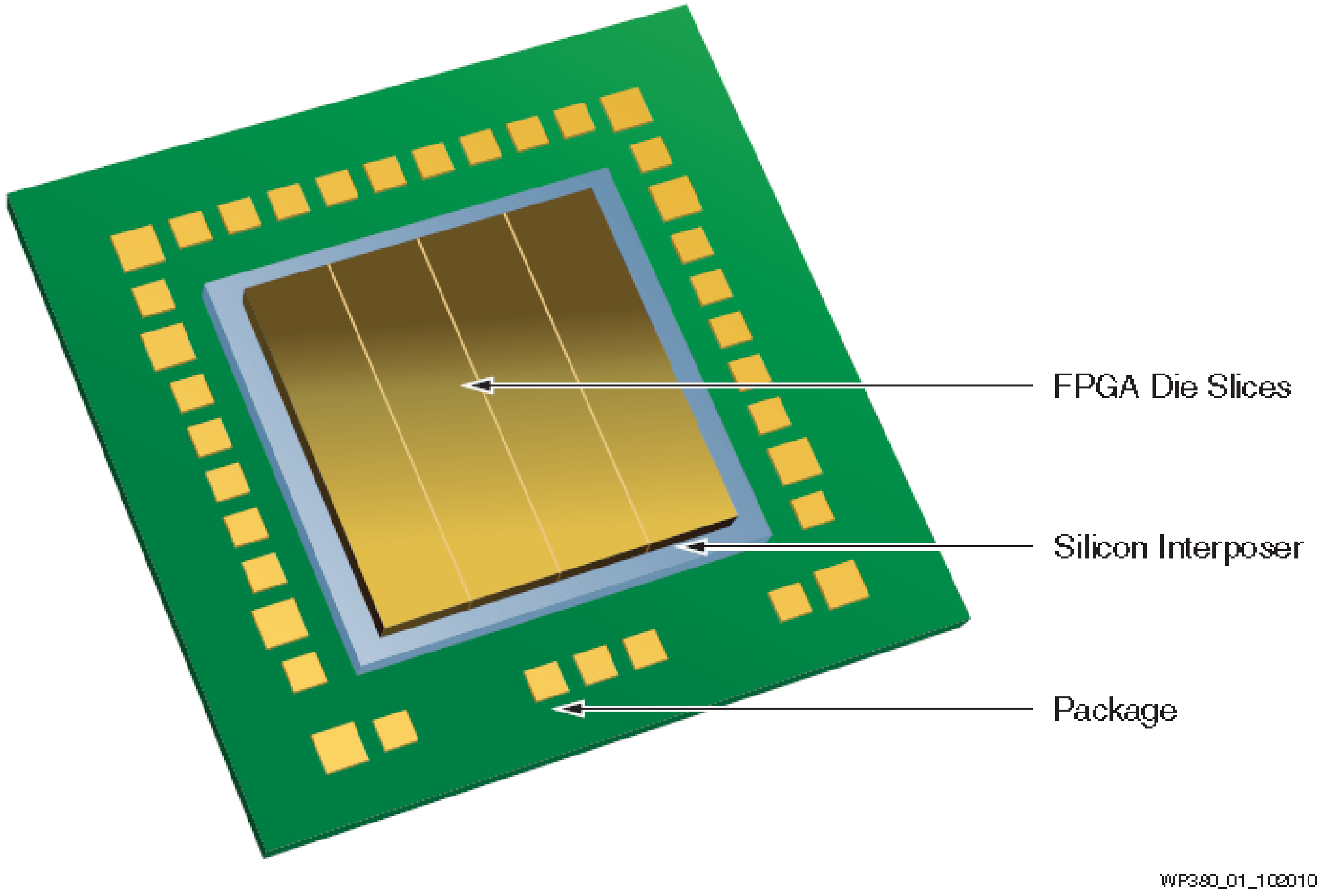
4.10.2 Xilinx Virtex-7

- Engineering Samples available since end of 2011
- 28 nm Technology
  - Manufactured by TSMC
- 6-input LUTs
- Dual 12-bit 1MSample/s ADC
  - Incl. on-chip sensors for temperature and power supply (1.0V or 0.9V)

Addressed Problems

- The ‚typical' problems, i.e. those that are improved from generation to generation
  - Power, Performance, …
- Problem: Yield
  - Especially large chips have typically yield problems
  - It takes a long time until (in comparison to smaller FPGAs) until they are available in larger quantities (or: at reasonable prices)
- Problem: Maximum size
  - So far: limited by yield
  - Workaround: connect multiple FPGAs on a PCB
  - Problem: Limited I/O Pins, Performance of Inter-FPGA connections, Distributing the clocks, larger power consumption, complicated PCB design, …
- Solution: Stacked Silicon Interposer (also called 2.5 D chips)

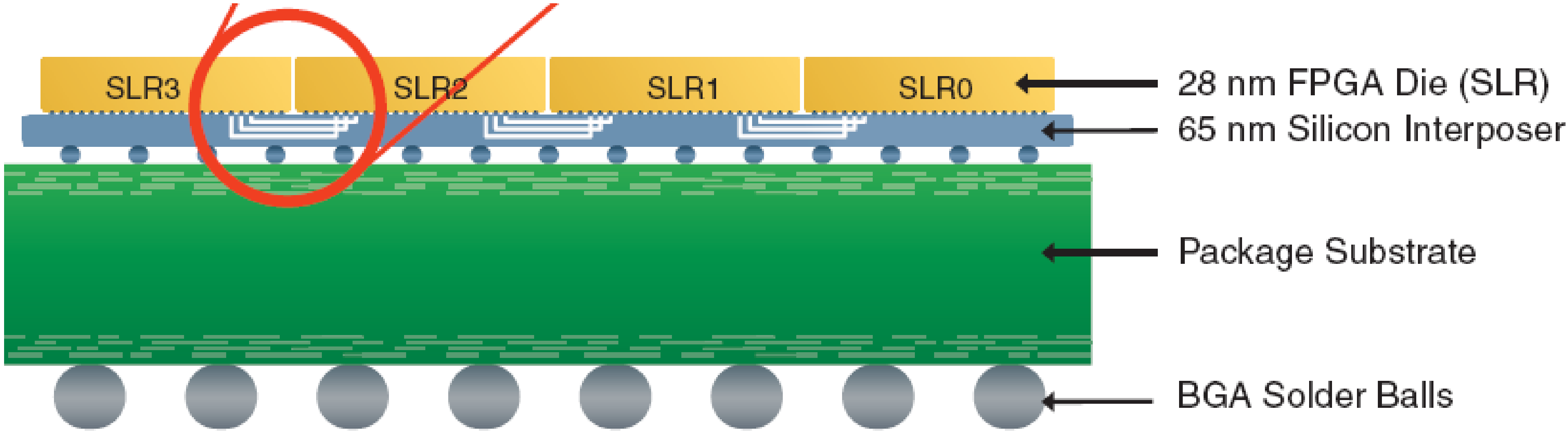
Stacked Silicon Interposer



src: Xilinx WP380 v1.0

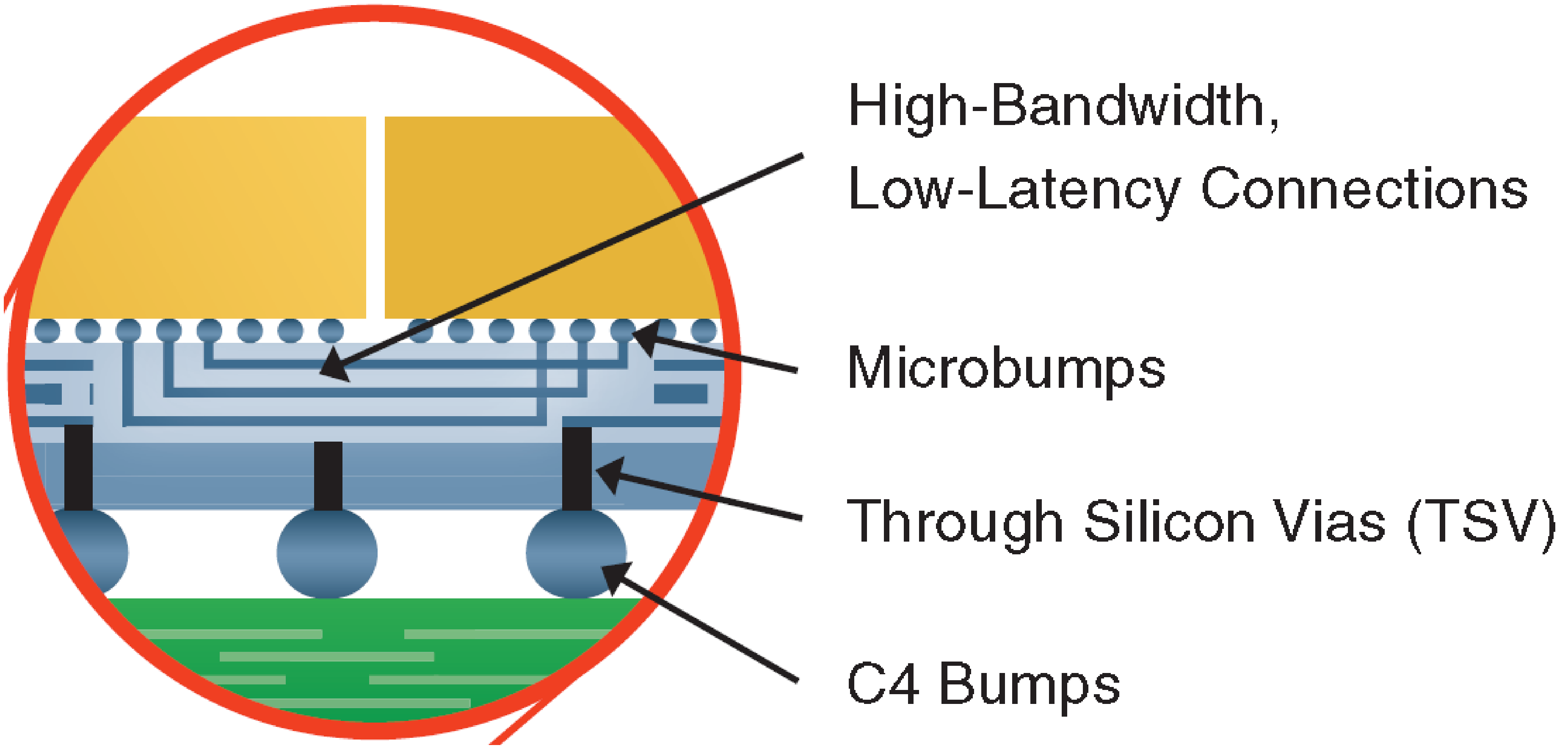
Stacked Silicon Interposer (cont'd)

SLR: Super Logic Region



src: Xilinx WP380 v1.2

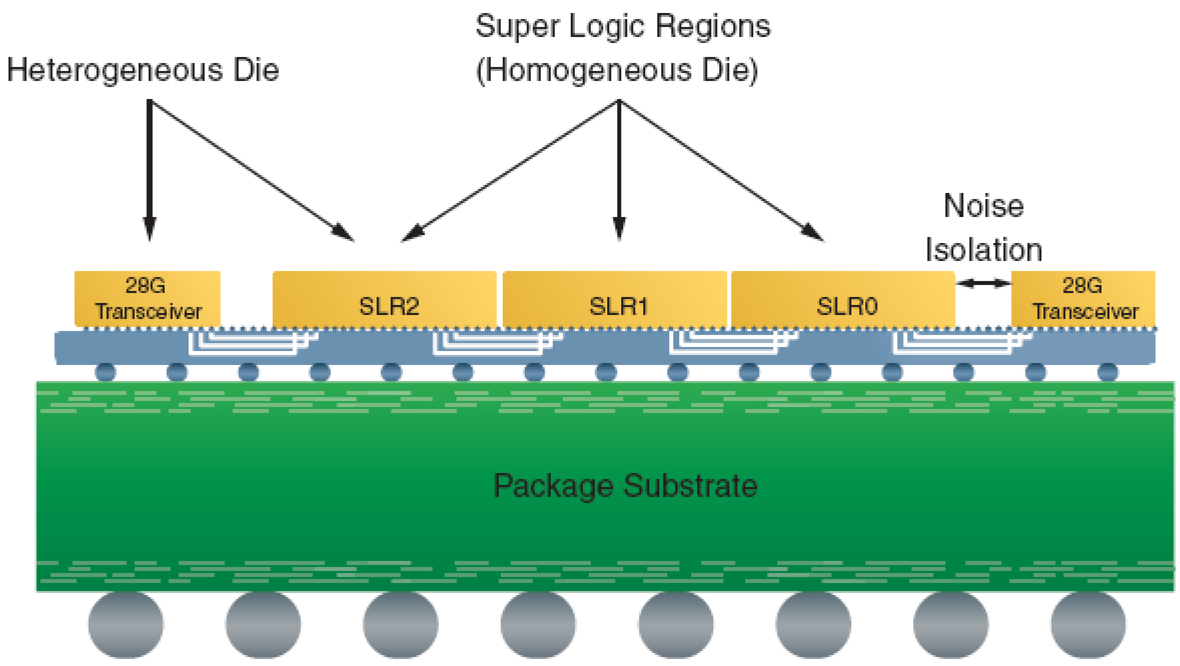
Stacked Silicon Interposer (cont'd)



src: Xilinx WP380 v1.2

Advantages

- Improves yield of 28nm FPGAs significantly
  - Disadvantage: stacking (technically complicated, but seems to work)
- It is easy to create different FPGA families and sizes by combining different FPGA fabrics on one interposer

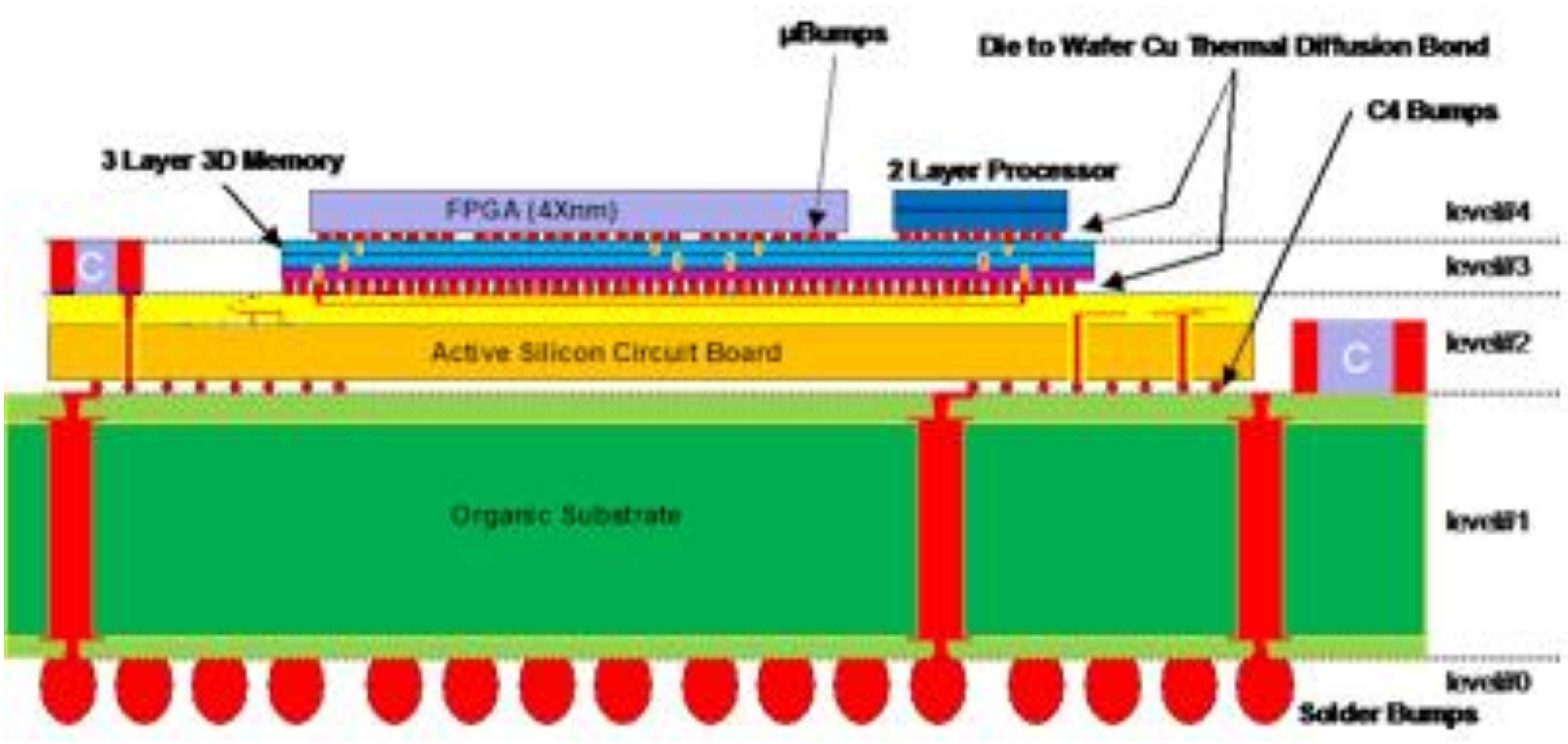


src: Xilinx WP380 v1.2

Advantages

- This technology would also allow to integrate state-of-the-art FPGAs with application-specific logic in a seamless and easy way
- E.g. combining:
  - One/Two Virtex-7 Super Logic Regions
  - One customized (reconfigurable) CPU ASIC
    On-chip DRAM for Cache/Scratchpad
- Advantages:
  - high-bandwidth low-latency connections
  - Stacked chips may be manufactured in different technologies

Next Step: ‘real’ 3D chips



src: http://www.eetimes.com/design/eda-design/4230786/Building-3D-ICs--Tool-Flow-and-Design-Software-Part-2?cid=NL_ProgrammableLogic&Ecosystem=programmable-logic

4.10.3 Xilinx Zynq

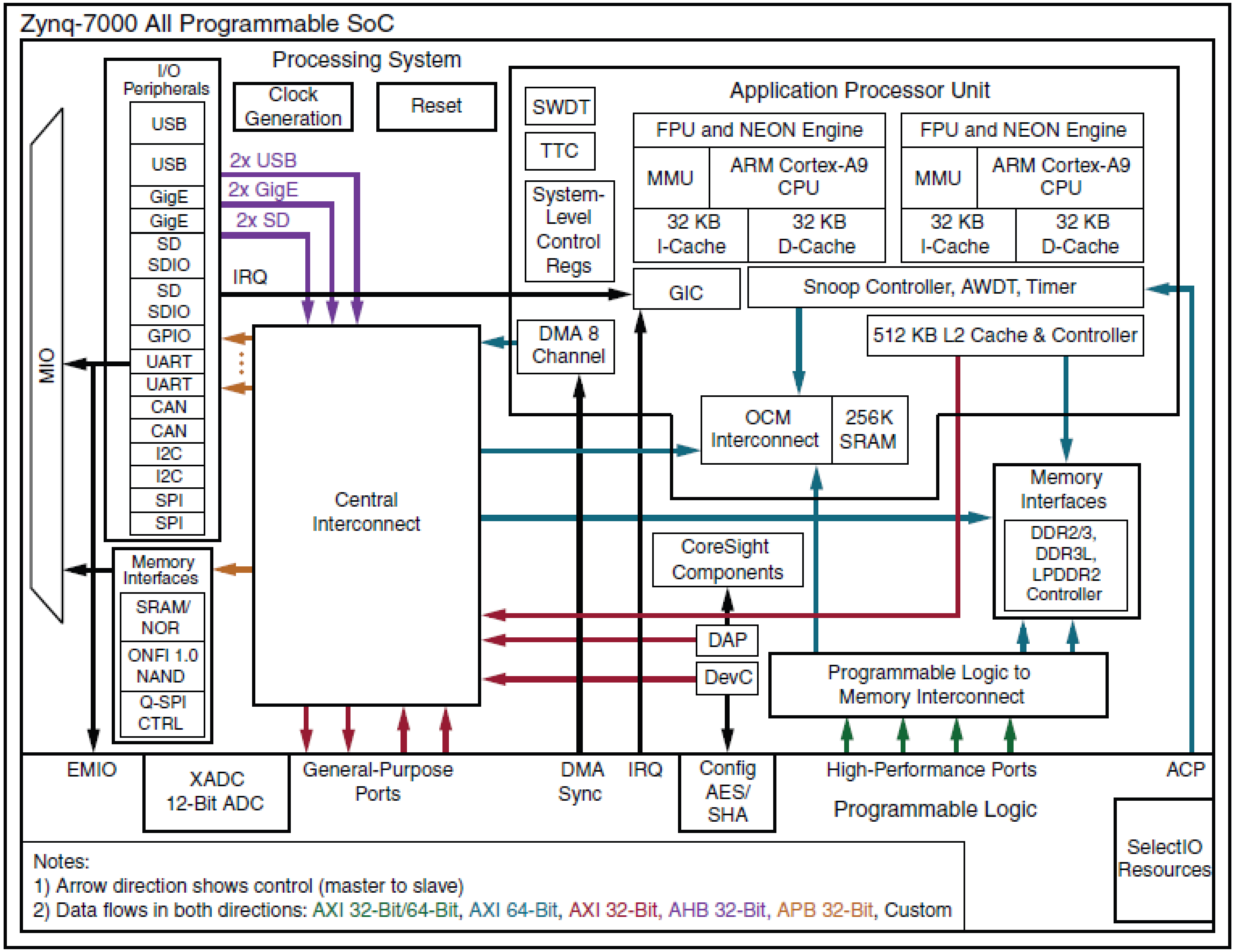
- Combination with Xilinx 7-series FPGA (called Programmable Logic – PL) and an ARM Cortex-A9 Dual-Core SoC (called Processing System – PS)
  - Smaller Zynq devices (Z-7010 and Z-7020) use the low-end Artix FPGA fabric and run the ARM at up to 800 MHz
  - Larger Zynq devices (Z-7030, Z-7045, and Z-70100) use the middle-end Kintex FPGA fabric and run the ARM at up to 1 GHz
- The ARM can be used even without configuring anything to the FPGA
  - Still, the ARM is coupled in many interesting ways to the FPGA that allow nice combinations of CPU and (reconfigurable) accelerators

Processing System

- Dual Core ARM
  - Single and double precision Vector Floating Point Unit
  - Timers, Watchdogs, Counters, and Interrupts
- Caches
  - 32 KB Level 1 4-way set-associative instruction and data caches (private per CPU)
  - 512 KB 8-way set-associative Level 2 cache (shared between the CPUs)
- On-Chip Memory
  - On-chip boot ROM
  - 256 KB on-chip RAM (OCM, aka Scratchpad)
- External Memory Interfaces
  - Multiprotocol dynamic memory controller
  - 16-bit or 32-bit interfaces to DDR3, DDR3L, DDR2, or LPDDR2
  - ECC support in 16-bit mode
  - 1GB of address space using single rank of 8-, 16-, or 32-bit-wide memories
  - Several Static memory interfaces (SRAM, NOR Flash, NAND Flash, …)
- Support for many peripherals and I/O interfaces (GigE, USB, CAN, SPI, …)

Connectivity between PS and PL

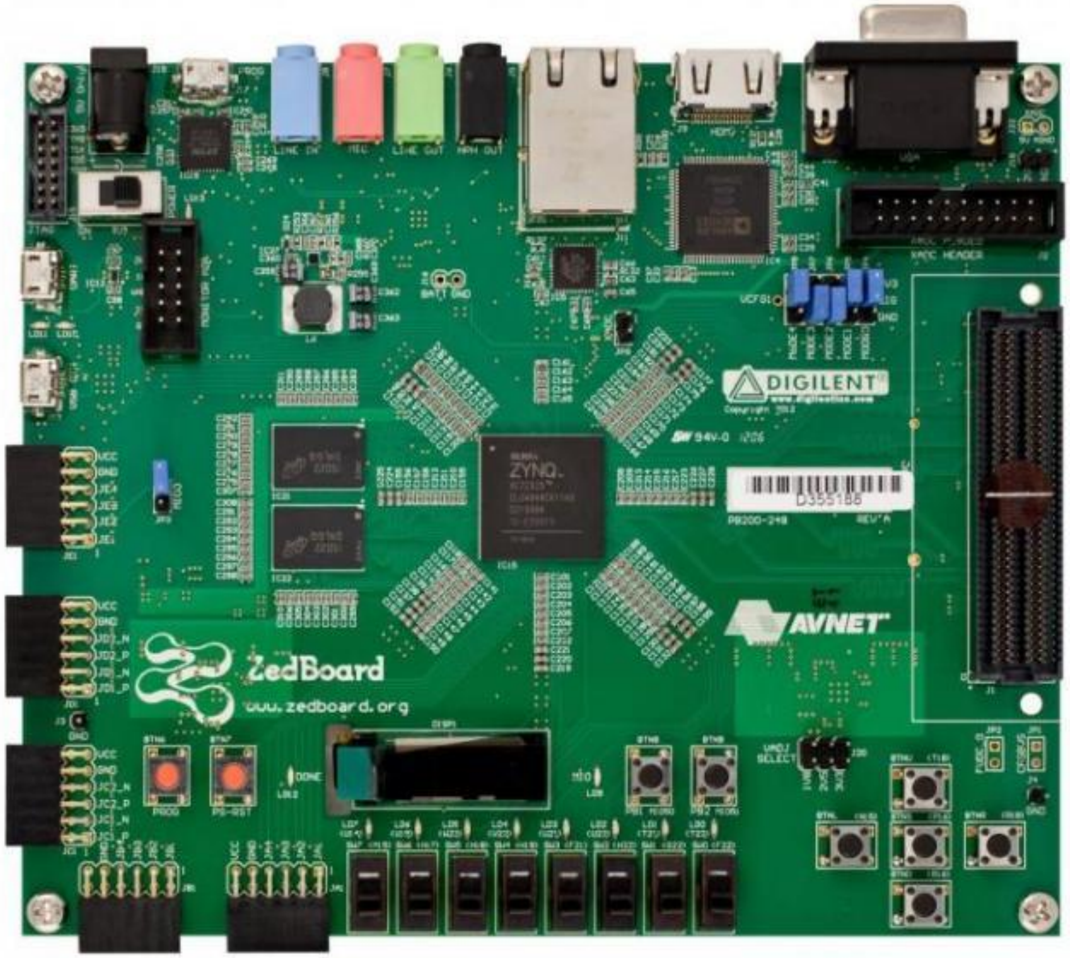
- Accelerator coherency port (ACP) interface enabling coherent accesses from PL to CPU memory space
- Dual-ported, on-chip RAM (256 KB)
  - Accessible by CPU and programmable logic (PL)
  - Designed for low latency access from the CPU
- 8-channel DMA
  - Supports multiple transfer types: memory-to-memory, memory-to-peripheral, peripheral-to-memory, and scatter-gather
  - 64-bit AXI interface, enabling high throughput DMA transfers
  - 4 channels dedicated to PL
- DDR memory controller is multi-ported and enables PS and PL to have shared access
  - One 64-bit port is dedicated for the ARM CPU(s) via the L2 cache controller and can be configured for low latency
  - Two 64-bit ports are dedicated for PL access
  - One 64-bit AXI port is shared by all other AXI masters via the central interconnect



Notes:
1) Arrow direction shows control (master to slave)
2) Data flows in both directions: AXI 32-Bit/64-Bit, AXI 64-Bit, AXI 32-Bit, AHB 32-Bit, APB 32-Bit, Custom

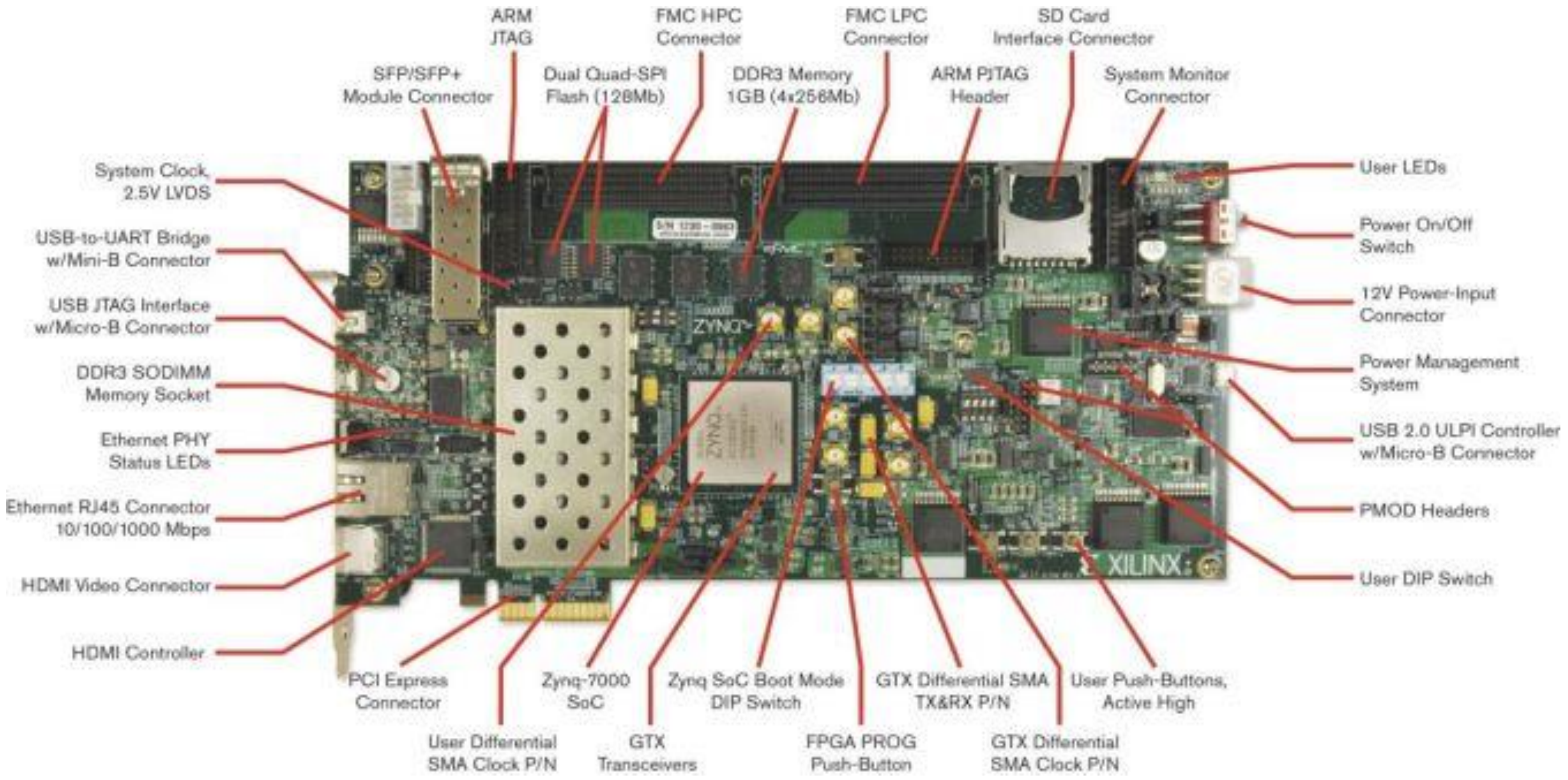
Zed Board

- Memory
  - 512 MB DDR3 memory (1066 Mbps)
  - 256 Mb Quad SPI Flash
- Connectivity
  - 10/100/1000 Ethernet
  - USB OTG (Device/Host/OTG), USB UART
- Expansion
  - FMC (Low Pin Count)
  - 5 Pmod™ headers (2x6)
- Video/Display
  - HDMI output (1080p60 + audio)
  - VGA connector
  - 128 x 32 OLED
  - User LEDs (9)
- User Inputs
  - Slide switches (8)
  - Push button switches (7)
- Audio
  - 24-bit stereo audio CODEC
  - Stereo line in/out, Headphone, Microphone input
- Analog
  - Xilinx XADC header, supports 4 analog inputs
- Price
  - 320 USD (academic)



src: http://www.zedboard.org/

SoC ZC706 Evaluation Board

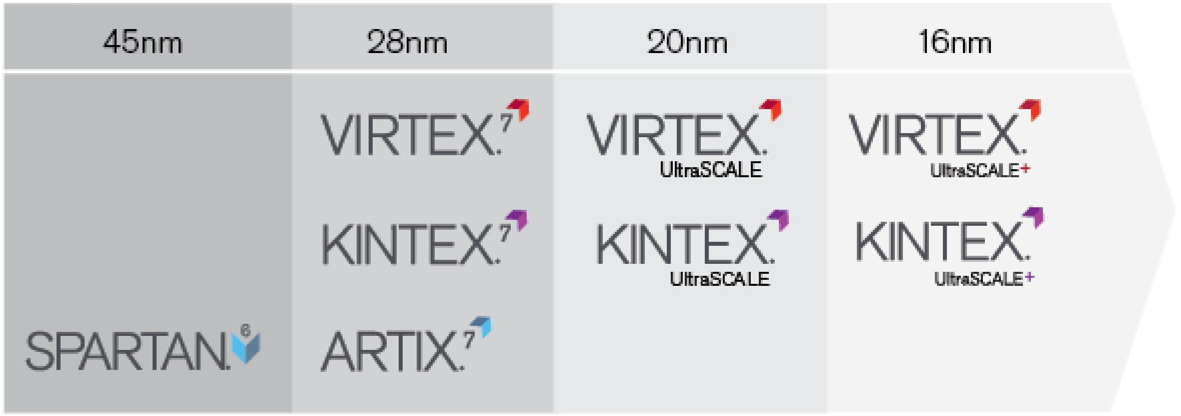

ARM JTAG
FMC HPC Connector
FMC LPC Connector
SD Card Interface Connector
SFP/SFP+ Module Connector
Dual Quad-SPI Flash (128Mb)
DDR3 Memory 1GB (4x256Mb)
ARM PJTAG Header
System Monitor Connector
System Clock, 2.5V LVDS
USB-to-UART Bridge w/Mini-B Connector
USB JTAG Interface w/Micro-B Connector
DDR3 SODIMM Memory Socket
Ethernet PHY Status LEDs
Ethernet RJ45 Connector 10/100/1000 Mbps
HDMI Video Connector
HDMI Controller
User LEDs
Power On/Off Switch
12V Power-Input Connector
Power Management System
USB 2.0 ULPI Controller w/Micro-B Connector
PMOD Headers
User DIP Switch
PCI Express Connector
Zynq-7000 SoC
Zynq SoC Boot Mode DIP Switch
GTX Differential SMA TX&RX P/N
User Push-Buttons, Active High
User Differential SMA Clock P/N
GTX Transceivers
FPGA PROG Push-Button
GTX Differential SMA Clock P/N

- Memory
  - DDR3 Component Memory 1GB (PS)
  - DDR3 SODIM Memory 1GB (PL)
  - 2X16MB Quad SPI Flash (config)
- Communication
  - PCIe Gen2x4
  - SFP+ and SMA Pairs
  - GigE RGMII Ethernet (PS)
  - USB OTG 1 (PS) - Host USB, USB UART
- Expansion Connectors
  - 2x FMC (HPC and LPC)
  - Dual Pmod (8 I/O Shared with LED's)
  - Single Pmod (4 I/O)
  - Some Buttons, Switches, LEDs
- Video/Display
  - HDMI IN and OUT 8 color RGB 4.4.4
- Price
  - 2500 USD

src: http://www.xilinx.com/

4.10.4 Xilinx UltraScale Architecture

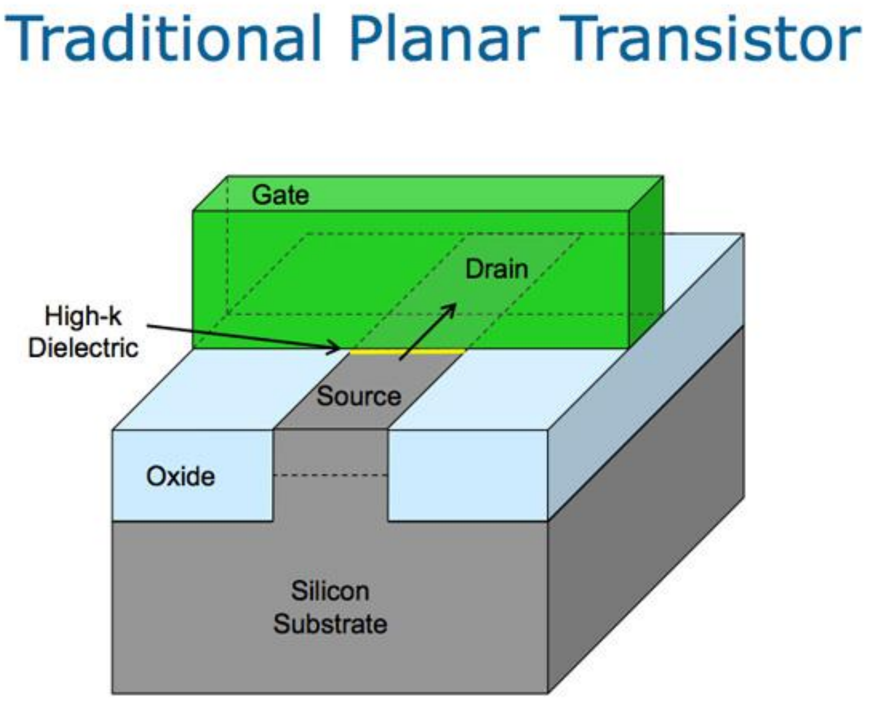
- “The UltraScale architecture was developed to scale from 20nm planar through 16nm and beyond FinFET (FF) technologies, and from monolithic through 3D ICs”
- UltraScale (20nm planar) first silicon in 2013
  - Shipping to first customers since Nov. 2013
- UltraScale+ (16nm FinFET) first silicon using TSMC 16nm FF in 2014
  - → Postponed till TSMC 16nm FF+ is available
  - Planed to ship in Q4/2015

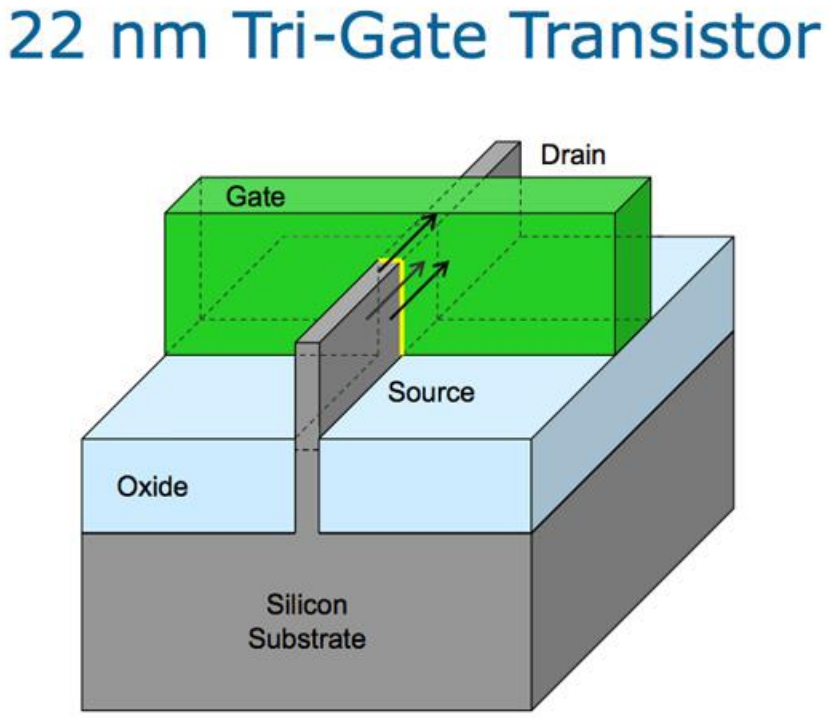


45nm	28nm	20nm	16nm
	VIRTEX 7	VIRTEX UltraSCALE	VIRTEX UltraSCALE+
	KINTEX 7	KINTEX UltraSCALE	KINTEX UltraSCALE+
SPARTAN 6	ARTIX 7		

src: http://www.xilinx.com/

FinFET (Tri-Gate)

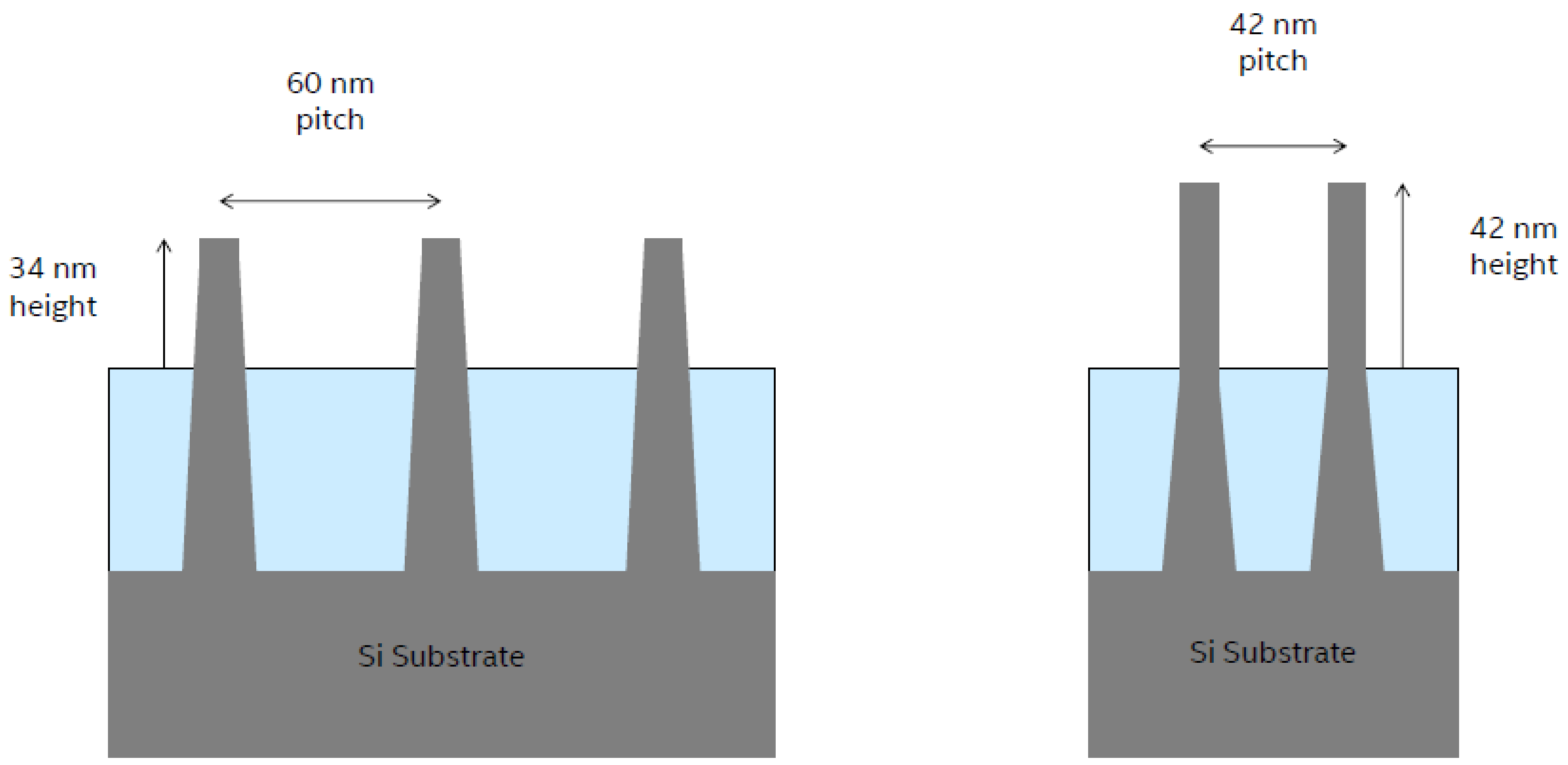




- Gate has stronger influence on channel
  - Faster transistor
  - Or: lower voltage → lower leakage
- There are often multiple fins between source and gate

src: Intel

Intel's 2nd FinFET Generation

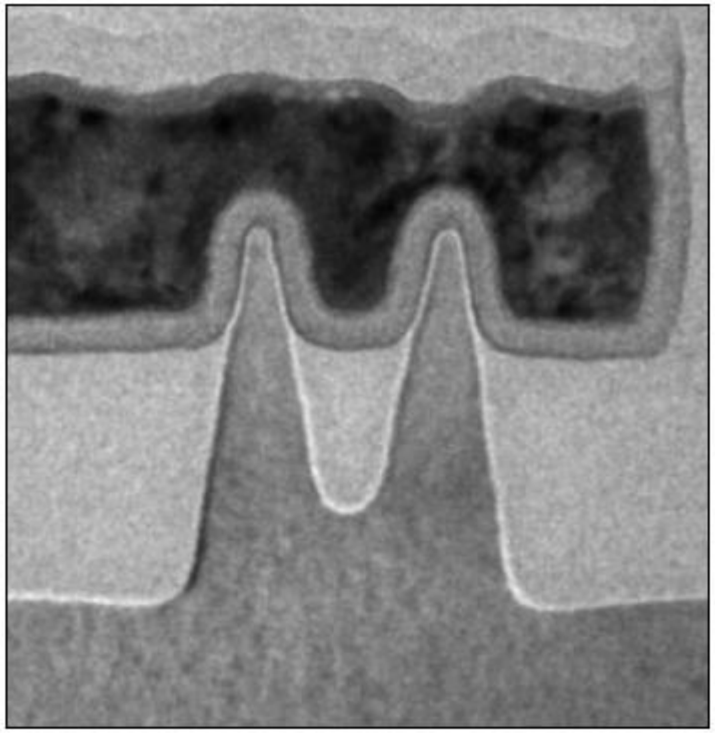


22 nm Process

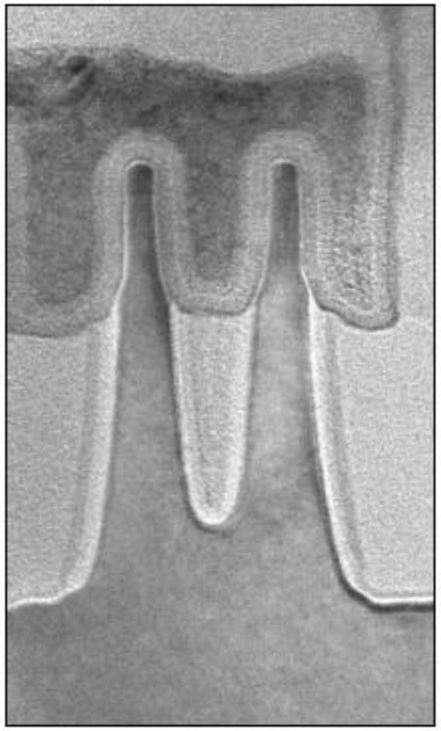
14 nm Process

src: "14 nm Technology Announcement", Intel, Mark Bohr, August 2014

Intel's 2nd FinFET Generation



22 nm 1st Generation Tri-gate Transistor

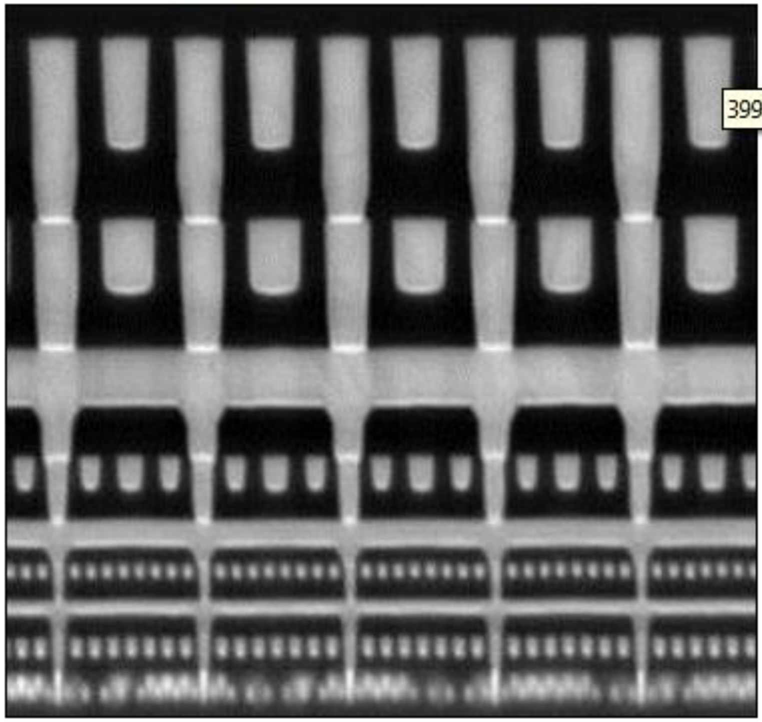


14 nm 2nd Generation Tri-gate Transistor

src: "14 nm Technology Announcement", Intel, Mark Bohr, August 2014

Interconnect

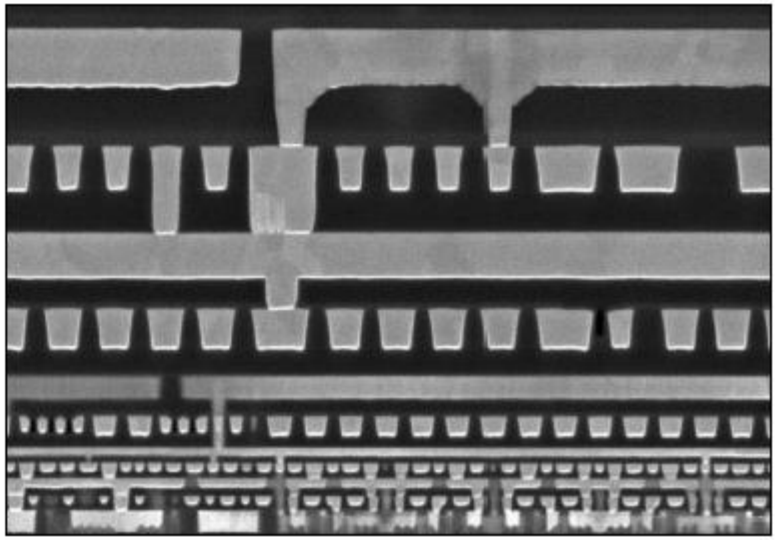
22 nm Process



80 nm minimum pitch

399_C_PubMix_01.JPG

14 nm Process



52 nm (0.65x) minimum pitch

src: "14 nm Technology Announcement",
Intel, Mark Bohr, August 2014

Back to Xilinx UltraScale: Faster, higher, further …

- Highly optimized critical paths and built-in high-speed memory, cascading to remove bottlenecks in DSP and packet processing
- Enhanced DSP slices incorporating 27x18-bit multipliers and dual adders that enable a massive jump in fixed-point and IEEE Std 754 floating-point arithmetic performance and efficiency
- Step function in inter-die bandwidth for 2nd-generation 3D IC systems integration and new 3D IC wide-memory optimized interface
- Massive I/O and memory bandwidth, including support for next generation memory interfacing with dramatic reduction in latency, optimized with multiple hardened, ASIC-class 100G Ethernet, Interlaken, and PCIe® IP cores
- Power management with a significant scope of static- and dynamic-power gating across a wide range of functional elements, yielding significant power savings

src: Xilinx Backgrounder “Introducing Xilinx UltraScale™ Architecture: Industry’s first ASIC-class all programmable architecture”

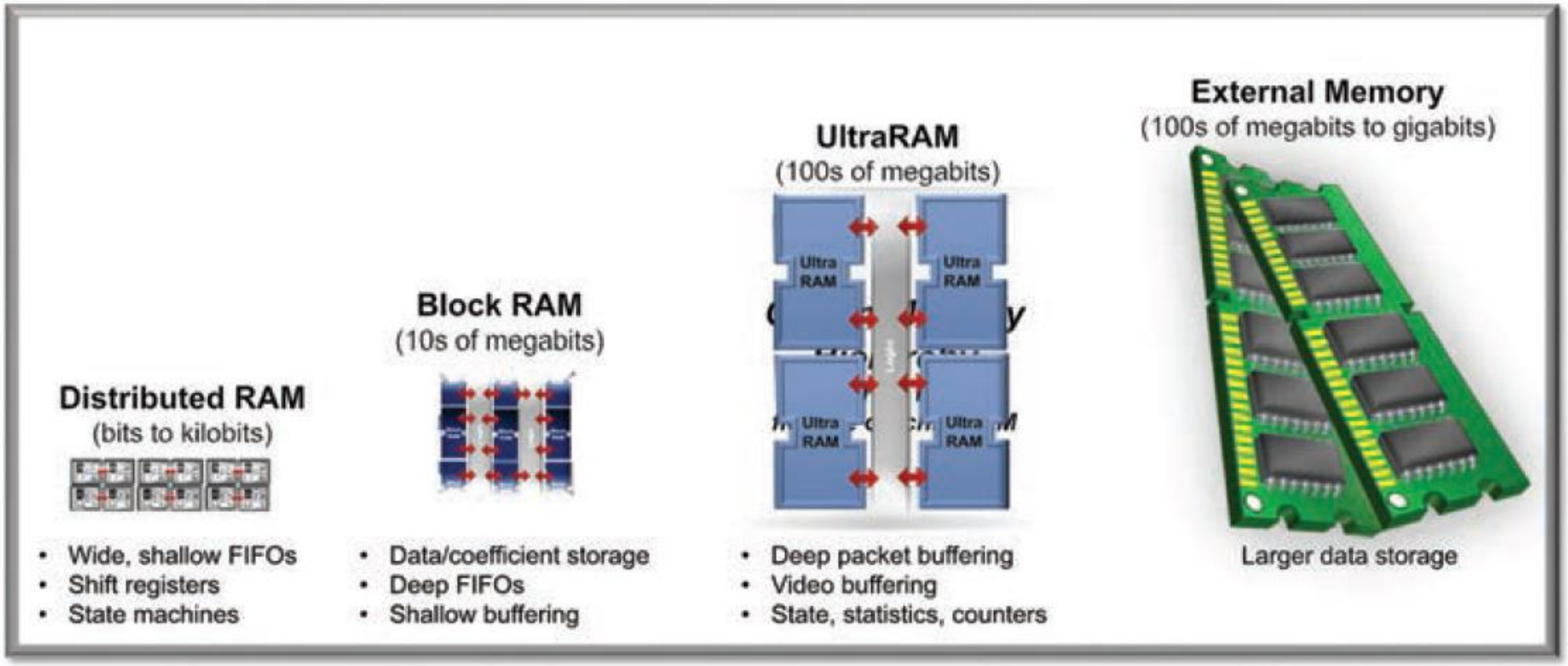
Adding more Interconnect Hardware

- “[…] interconnect is the number one bottleneck to system performance.”
- “The routing efficiencies delivered through the UltraScale Architecture essentially removes routing congestion completely. The result is simple: If the design fits, it routes. This holds true for device utilization at levels of greater than 90% with no performance degradation or increase in system latency.”

src: Xilinx Backgrounder “Introducing Xilinx UltraScale™ Architecture: Industry’s first ASIC-class all programmable architecture”

Age of ~~Ultron~~ UltraRAM

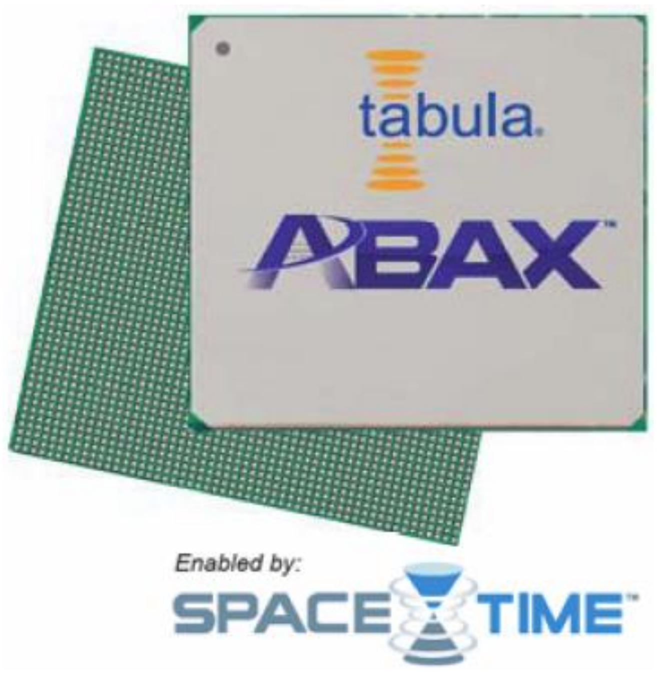
- "The largest UltraScale+ device, the VU13P, will have 432 Mbits of UltraRAM."



src: Xilinx Xcell Issue 90

4.10.5 Tabula

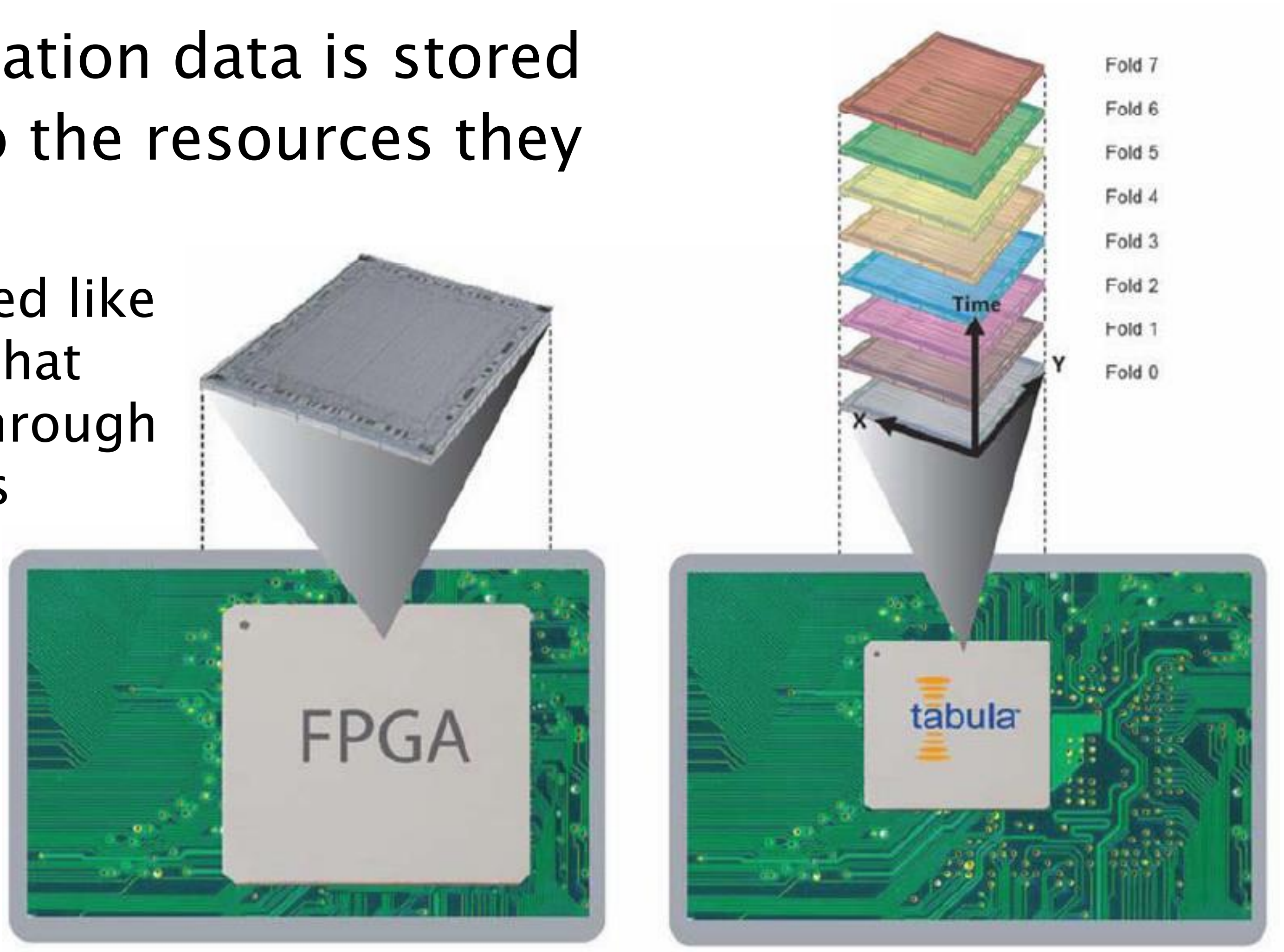
- A startup company (founded 2003) that developed a 3D Programmable Logic Device (3PLD; basically an FPGA)
  - Uses time as third dimension (rather than waiting for 3D-stacked chips to become mainstream)
- Achieved by dynamically reconfiguring logic, memory, and interconnect at multi-GHz rates
  - Executing each portion of a design in an automatically defined sequence of steps
- Spacetime compiler (i.e. synthesis, place&route tools) manages 'ultra-rapid' reconfiguration transparently



src: Tabula "ABAX Product Family Overview"

Configuration contexts: 'Folds'

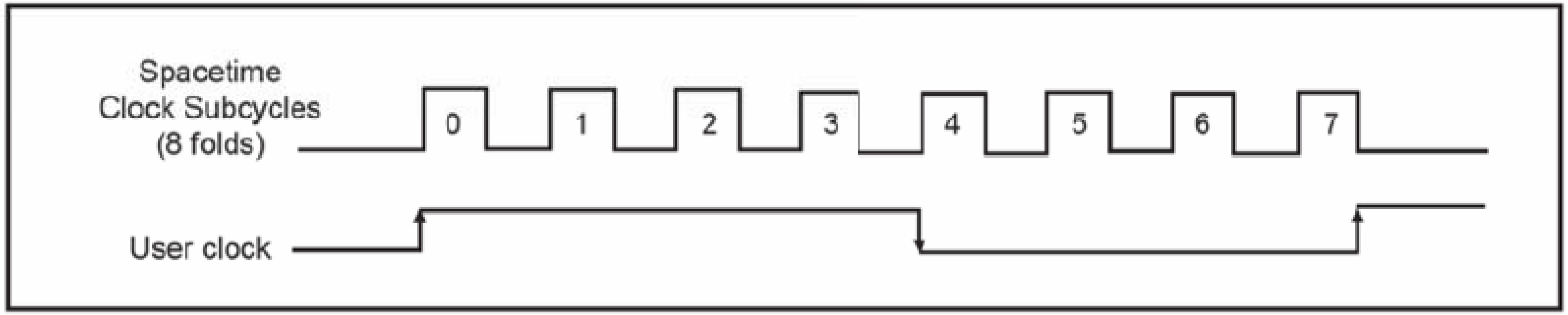
- Configuration data is stored locally to the resources they control
  - Organized like a stack that cycles through the folds



src: Tabula "Spacetime Architecture White Paper"

Spacetime clock

- The user clock is divided into sub-cycles which form the folds
  - The currently available device core operates at up to 1.6 GHz
  - The user clock depends upon the number of folds
  - E.g. 200 MHz for 8 folds (figure shows an 8-fold spacetime clock) or 400 MHz for 4 folds etc.
  - Up to 8 folds are supported



src: Tabula “Spacetime Architecture White Paper”

Data Transfer

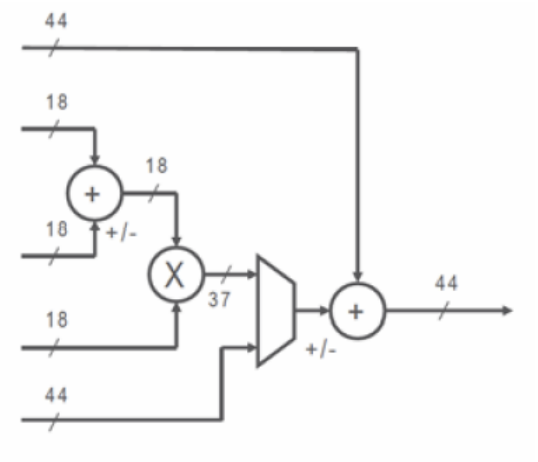
- 3D device with multiple layers (so-called ‘folds’) in which computation and signal transmission can occur
  - Each ‘fold’ performs a portion of the desired functionality and stores the result in place
- After configuring the next fold, it uses the locally stored data to perform the next portion of the function
  - → the data is not moving (at least not far), but the hardware is changing (data can stay local)
  - → lower demand for interconnect resources

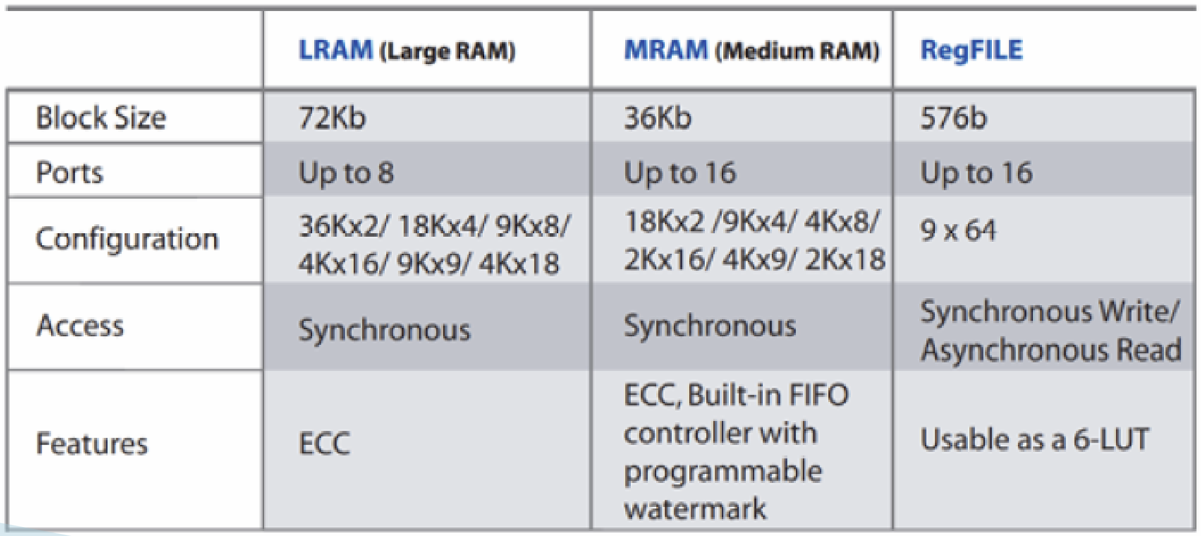
Resource reuse

- All resources can be modified and reused when going from one fold to the next
  - Memory ports, LUTs, routing, …
  - For instance, a 8-bit wide path in 8 folds delivers 64-bits per user clock cycle
  - A single-port memory appears to have 8 ports that can access arbitrary addresses
    - or a 8-fold wider memory port
    - or 8 independent memories (total capacity must not exceed capacity of the original single-port memory)

Tabula ABAX resources

- 'Spacetime' Fabric features:
  - Logic: 0.22M – 0.63M LUTs (4-input LUT equivalent), operating at 1.6 GHz
  - DSP blocks: up to 1280 1.6 GHz 18x18 multiplier/accumulators
  - Memory: 5.5 MBytes (!) @ 1.6 GHz, featuring 8 and 16 ports and built-in ECC and FIFO controller
- Manufactured using TSMC's 40 nm process





	LRAM (Large RAM)	MRAM (Medium RAM)	RegFILE
Block Size	72Kb	36Kb	576b
Ports	Up to 8	Up to 16	Up to 16
Configuration	36Kx2/ 18Kx4/ 9Kx8/ 4Kx16/ 9Kx9/ 4Kx18	18Kx2 /9Kx4/ 4Kx8/ 2Kx16/ 4Kx9/ 2Kx18	9 x 64
Access	Synchronous	Synchronous	Synchronous Write/ Asynchronous Read
Features	ECC	ECC, Built-in FIFO controller with programmable watermark	Usable as a 6-LUT

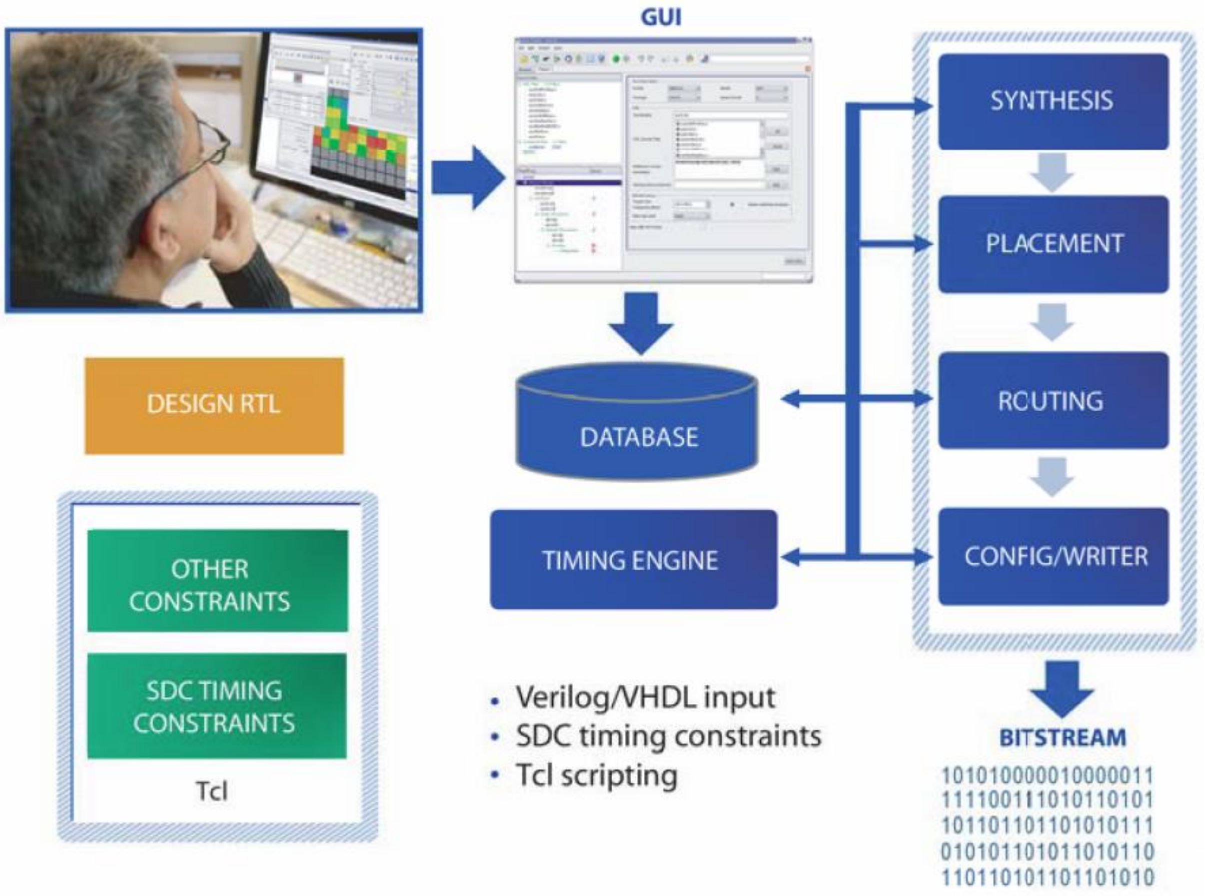
src: Tabula "ABAX Product Family Overview"

‘Marketing’ comparison with 2D FPGA

- 2.5x Logic Density (LUTs/mm^2 for 40 nm devices) and 2.0x Memory Density (due to single port memories)
  - Logic, memory, and routing resources are all re-used multiple times per user cycle
    → higher density and shorter interconnect
- 2.9x more Memory Ports (in total, i.e. over all memory ports on the device)

Spacetime Compiler 'Stylus'

- Automatically maps, places, and routes existing designs into an ABAX device
- All control of the hardware reconfiguration is automatically and invisibly managed by the Spacetime compiler

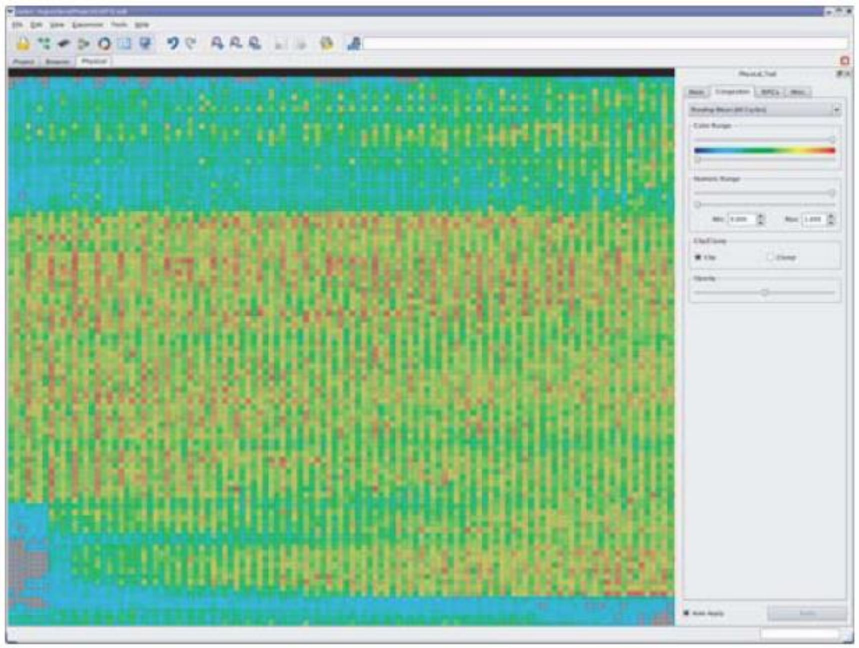


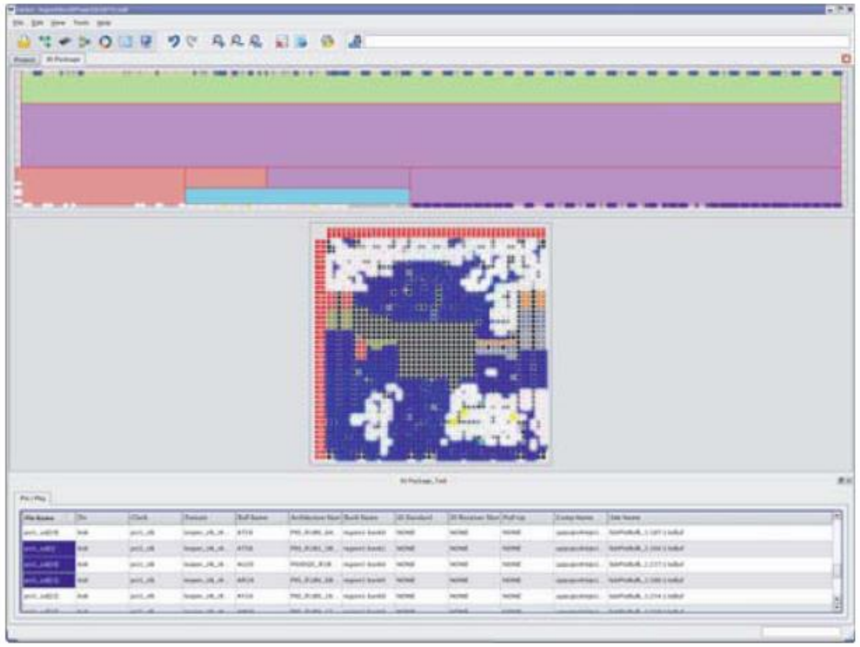
- Verilog/VHDL input
- SDC timing constraints
- Tcl scripting

src: Tabula "ABAX Product Family Overview"

Design Analysis

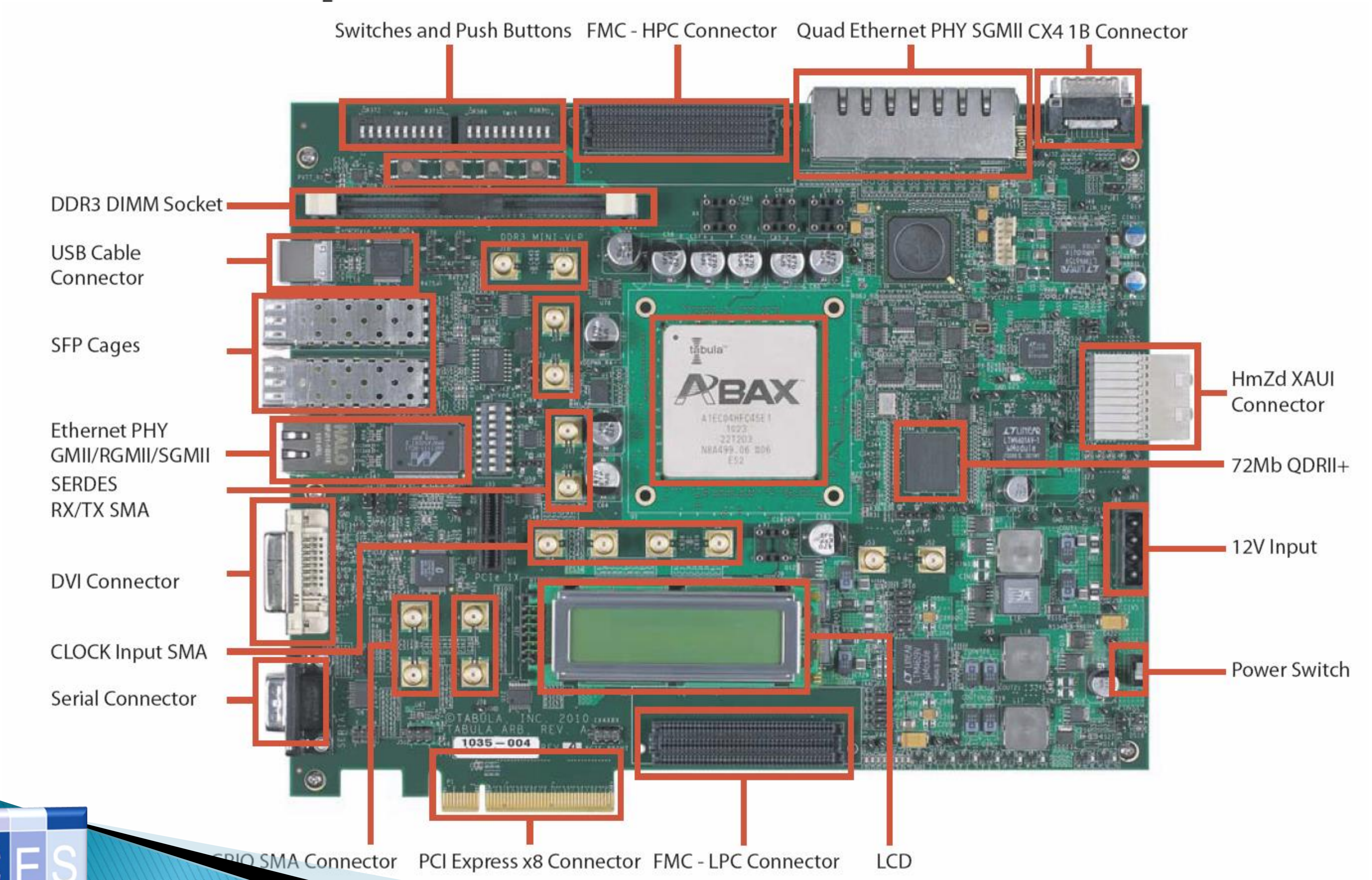
- View Placement and Routing
- Visualize timing critical paths and slack histograms
- Cross probe between HDL source, schematic, and place and route views





src: Tabula “Stylus Software Overview”

Development Kit



src: Tabula “3PLD Development Kit”

Development Kit (cont‘d)

EASY ACCESS TO ABAX TECHNOLOGY

- High density, capacity, speed and throughput
- 390k LUTs (4-input LUT equivalent)
- On-chip user RAM
  - 5.5 MBytes
  - 16-port and 8-port RAMs
- 48 SerDes, 55 Mb/s – 6.5 Gb/s with PMA/PCS built-in
  - PCI Express 1.0 and 2.0
  - XAUI, double XAUI, GigE, FC
  - Serial Rapid I/O, CPRI LV, and JC-16
  - Interlaken
- 920 Configurable parallel I/Os
  - DDR3 800 Mbps
  - LVDS 1.2 Gbps

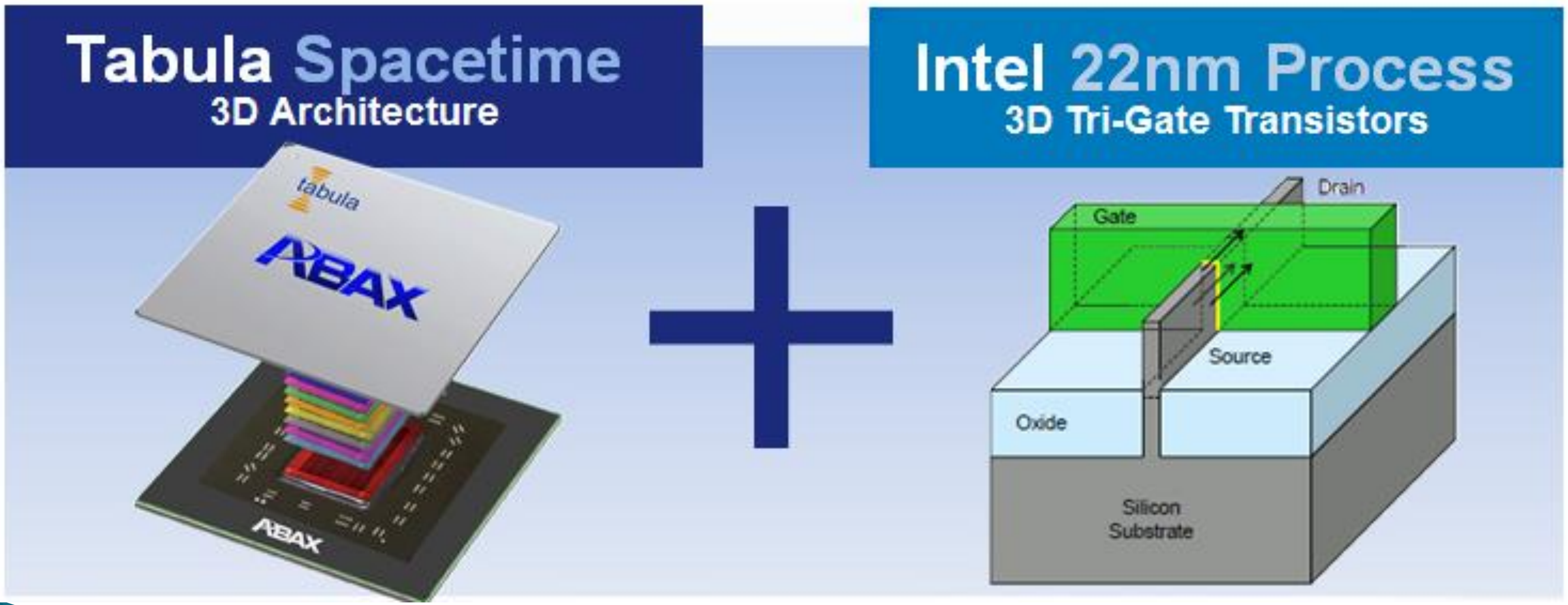
HIGH PERFORMANCE PCIe PLUG-IN CARD

- ABAX A1EC04 in 1936-pin FBGA
- Abundant off-chip memory:
  - x36 72 MBytes of QDRII
  - x72 1 GB DDR3 mini VLP DIMM
- High-speed interfaces
  - Five GigE ports (one single, one quad)
    - Single port supports SGMII, MII modes, quad port SGMII
    - XAUI backplane connectivity
    - x1 and x8 PCI Express
    - Fibre Channel, and Infiniband, DVI Support
      - Dual SFP, CX4 IB connectors
- Portfolio of third-party application specific FPGA mezzanine cards
- SMA connectors tied to device SerDes and general purpose I/O

src: Tabula “3PLD Development Kit”

New ABAX2P1 devices

- Next generation uses Intel's 22nm FinFET technology (called Tri-Gate)
  - Announced February 2012

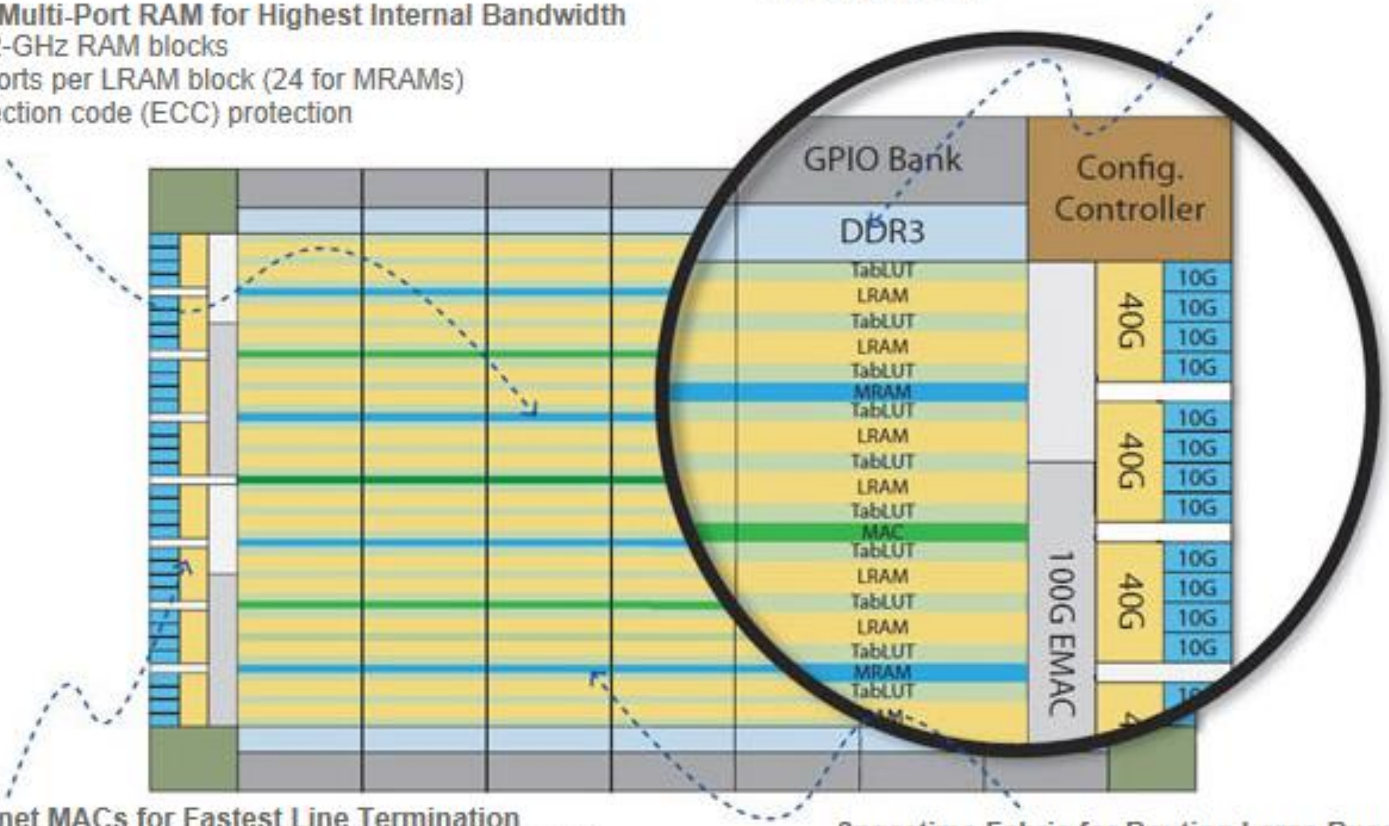


src: tabula.com

New ABAX2P1 devices (cont'd)

DDR3 Controllers for Optimal External RAM Bandwidth
- Full-speed DDR3 controllers @ 2.133 GT/s
- Up to 4 × 72-bit controllers
- Adjustable controller width from 8 to 72 bits
- Support for DDR2

Spacetime Multi-Port RAM for Highest Internal Bandwidth
- 23 MB of 2-GHz RAM blocks
- Up to 12 ports per LRAM block (24 for MRAMs)
- Error-correction code (ECC) protection



100G Ethernet MACs for Fastest Line Termination
- Up to 4 × 100G EMACs (or 16 × 40 GigE or 64 × 10 GigE)
- 64 × 14.1 Gbps transceivers
- IEEE-1588 time-stamping
- Custom preamble
- 10/40/100G forward error correction (FEC)

Spacetime Fabric for Routing Large Busses in 3D
- 2 GHz balanced fabric
- 3D fabric with up to 12 folds
- 570,000 TabLUTs
- 70,000 logic-carry blocks (LCB)

src: tabula.com

New ABAX2P1 devices (cont'd)

- 2 GHz Spacetime clock
- 12-fold architecture
- 570,000 TabLUTs
- 23 MB of RAM (up to 12/24 ports per block)

- Recent News: Company shut down in March 2015

src: tabula.com

References and Sources

[WAL+93] M. Wazlowski, L. Agarwal, T. Lee, A. Smith, E. Lam, P. Athanas, H. Silverman, S. Ghosh: "PRISM-II Compiler and Architecture", IEEE Workshop on FPGAs, 1993.

[HW97] J. R. Hauser, J. Wawrzynek: "Garp: A MIPS Processor with a Reconfigurable Coprocessor", IEEE Symposium on FPGA-Based Custom Computing Machines, pp. 24-33, 1997.

[CHW00] T. J. Callahan, J. R. Hauser, J. Wawrzynek: "The Garp Architecture and C Compiler", IEEE Computer, vol. 33, no. 4, pp. 62-69, 2000.

[VWG+04] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, E.M. Panainte: "The MOLEN Polymorphic Processor", IEEE Transactions on Computers, vol. 52, no. 11, pp. 1363-1375, 2004.

[RS94] R. Razdan, M. D. Smith: "A High-Performance Microarchitecture with Hardware-programmable Functional Units", International Symposium on Microarchitecture, pp. 172-180, 1994.

[WC96] R. D. Wittig, P. Chow: "OneChip: an FPGA processor with reconfigurable logic", IEEE Symposium on FPGAs for Custom Computing Machines, pp. 126-135, 1996. NOTE: actual screenshots taken from dissertation from Wittig (same name as paper) from 1995 due to their better visual quality

[JC99] J. A. Jacob, P. Chow: "Memory interfacing and instruction specification for reconfigurable processors", International Symposium on Field Programmable Gate Arrays, pp. 145-154, 1999.

[CC01] J. E. Carrillo, P. Chow: "The effect of reconfigurable units in superscalar processors", International Symposium on Field Programmable Gate Arrays, pp. 141-150, 2001.

[LTC+03] A. Lodi, M. Toma, F. Campi, A. Cappelli, R. Canegallo, R. Guerrieri: "A VLIW Processor with Reconfigurable Instruction Set for Embedded Application", IEEE Journal of Solid-State Circuits, vol. 38, no. 11, pp. 1876-1886, Nov. 2003.

[CCG+03] F. Campi, A. Cappelli, R. Guerrieri, A. Lodi, M. Toma, A. La Rosa, L. Lavagno, C. Passerone, R. Canegallo: "A Reconfigurable Processor Architecture and Software Development Environment for Embedded Systems", 17th International Symposium on Parallel and Distributed Processing, pp. 171.1, 2003.

[LCB+06] A. Lodi, A. Cappelli, M. Bocchi, C. Mucci, M. Innocenti, C. De Bartolomeis, L. Ciccarelli, R. Giansante, A. Deledda, F. Campi, M. Toma, R. Guerrieri: "XiSystem: A XiRisc-Based SoC with Reconfigurable IO Module", IEEE Journal of Solid-State Circuits, vol. 41, no. 1, pp. 85-96, 2006.