Organic Computing

Dr. rer. nat. Christophe Bobda
Prof. Dr. Rolf Wanka
Department of Computer Science 12
Hardware-Software-Co-Design

Reconfigurable Computing Platforms

The Von Neumann Computer

- **Principle**

In 1945, the mathematician Von Neumann (VN) demonstrated in study of computation that a computer could have a
simple structure,
capable of executing any kind of program,
given a properly programmed control unit,
without the need of hardware modification

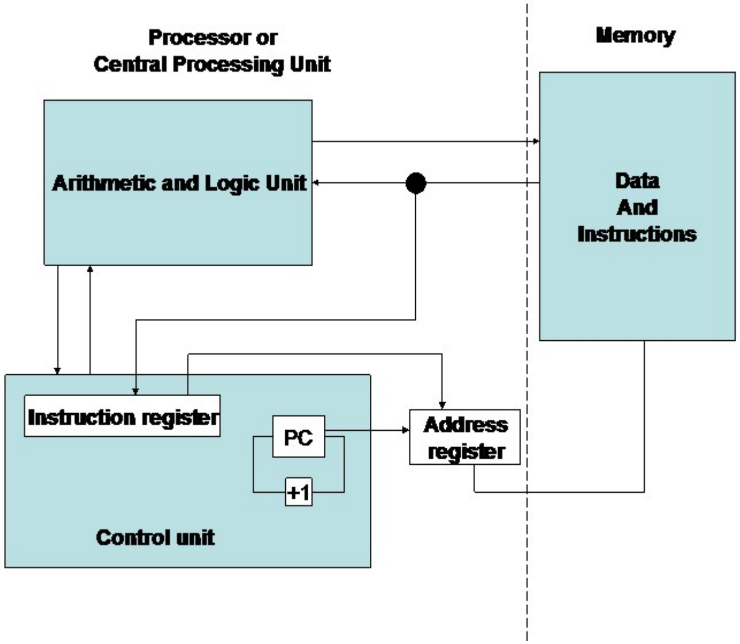
The Von Neumann Computer

- **Structure**

1. A memory for storing program and data. The memory consists of the word with the same length

3. A control unit (control path) featuring a program counter for controlling program execution

5. An arithmetic and logic unit(ALU) also called data path for program execution



The Von Neumann Computer

- **Coding**

  A program is coded as a set of instructions to be sequentially executed

- **Program execution**

  1. **Instruction Fetch (IF)**: The next instruction to be executed is fetched from the memory
  2. **Decode (D)**: The instruction is decoded to determine the operation
  3. **Read operand (R)**: The operands are read from the memory
  4. **Execute (EX)**: The required operation is executed on the ALU
  5. **Write result (W)**: The result of the operation is written back to the memory
  6. **Instruction execution in Cycle (IF, D, R, EX, W)**

The Von Neumann Computer

Advantage:

- **Flexibility**: any well coded program can be executed

Drawbacks

- **Speed efficiency**: Not efficient, due to the sequential program execution (temporal resource sharing).
- **Resource efficiency**: Only one part of the hardware resources is required for the execution of an instruction. The rest remains idle.
- **Memory access**: Memories are about 10 time slower than the processor

Drawbacks are compensated using high clock speed, pipelining, caches, instruction pre-fetching, etc.

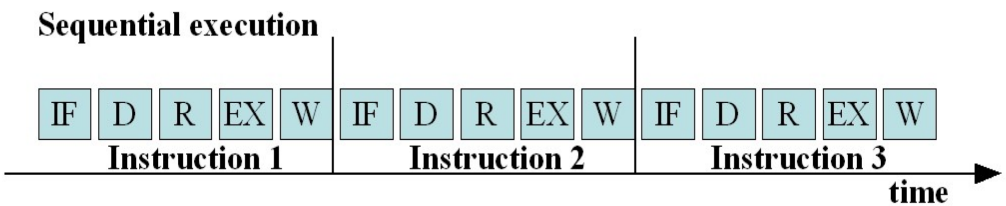
The Von Neumann Computer

Sequential execution

t_{cycle} = cycle execution time

One instruction needs $t_{instrcution} = 5*t_{cycle}$

3 instructions are executed in $15*t_{cycle}$



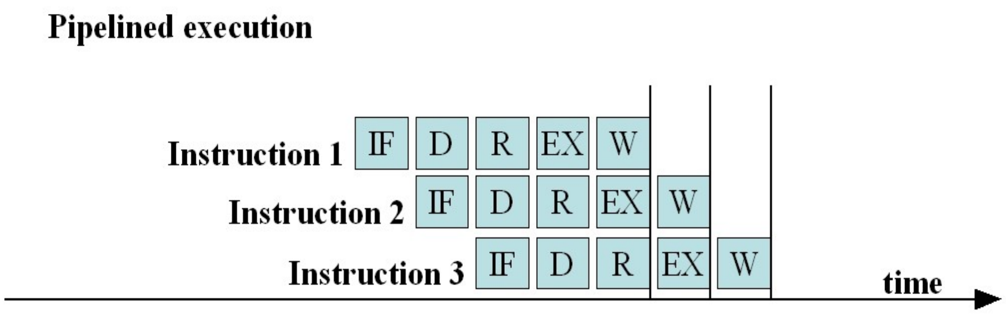
Pipelining:

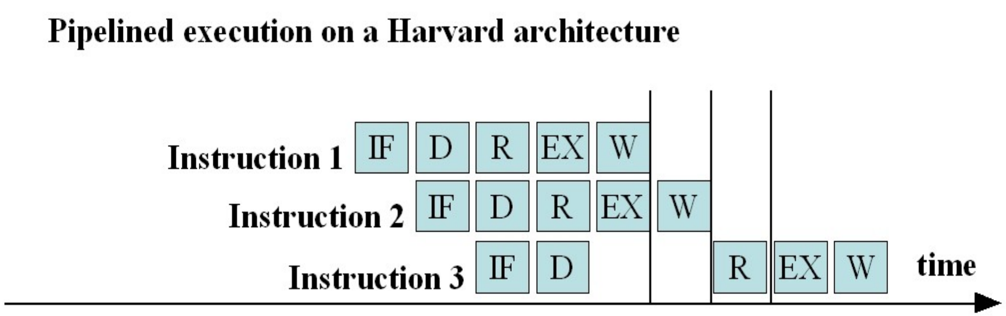
One instruction needs $t_{instrcution} = 5*t_{cycle}$

no improvement.

3 instructions need $7*t_{cycle}$ in the ideal case.

$9*t_{cycle}$ on a Harvard architecture.





- Increased throughput
- Even with pipeline and other improvement like cache, the execution remain sequential.

Domain specific processors

Goal:

Overcome the drawback of the von Neumann computer.

Optimize the Data path for a given class of applications

DSP (Digital Signal Processors) :

Signal processing applications are usually multiply accumulate (MAC) dominated.

- **The data path is optimized to execute one or many MACs in only one cycle.**
- **Instruction fetching and decoding overhead is removed**
- **Memory access is limited by directly processing the input dataflow**

Domain specific processors

DSPs:

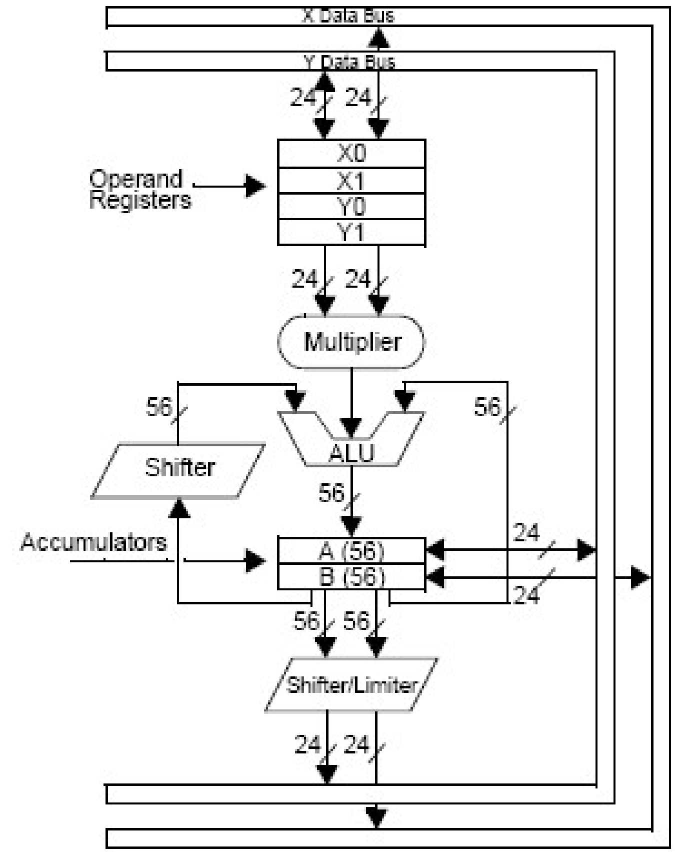
Designed for **high-performance, repetitive, numerically intensive tasks**

In one Instruction Cycle, can do:

- many MAC-operations
- many memory accesses
- special support for efficient looping

The hardware contains:

- One or more MAC-Units
- Multi-ported on-chip and off-chip memories
- Multiple on-chip busses
- Address generation unit supporting addressing modes tailored for DSP-applications



Application specific processors

Optimize the complete circuit for a given function

ASIC: Application Specific Integrated Circuit.

Optimization is done by implementing the inherent parallel structure on a chip

- The data path is optimized for only one application.
- Instruction fetching and decoding overhead is removed
- Memory access is limited by directly processing the input data flow
- Exploitation of parallel computation

Application specific processors

ASIC Example:

Implementation of a VN computer

```
if (a < b) then
{
    d = a+b;
    c = a*a;
}
else
{
    d = a+1;
    c = b-1;
}
```

At least 5 instructions

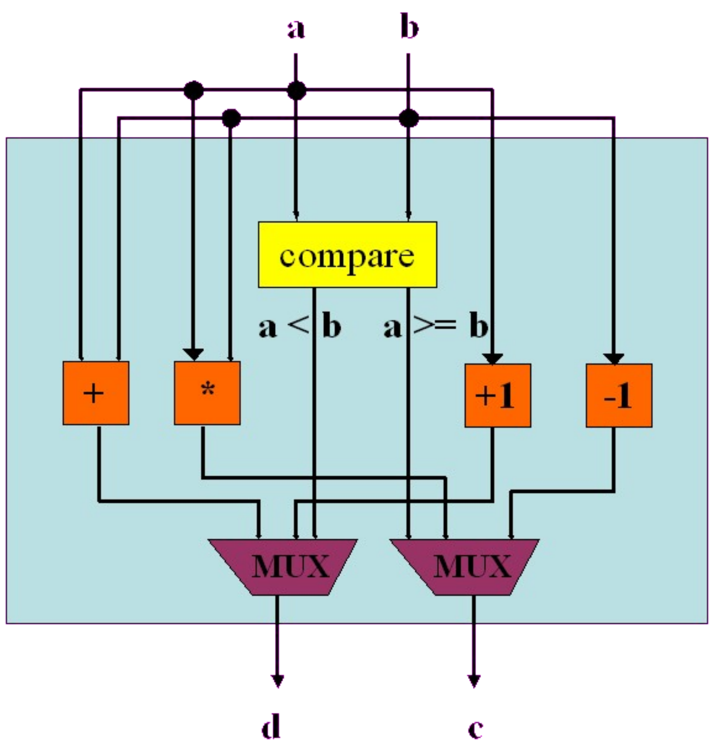
run-time >= 5*$t_{instruction}$

The VN computer needs to be clocked at least 5 time faster

ASIC implementation:

The complete execution is done in parallel in one clock cycle

run-time = t_{clock} = delay longest path from input to output



Conclusion

- **Von Neumann computer:**

  General purpose, used for any kind of function.

  - **High degree of flexibility.**

  However, high restrictions on the program coding and execution scheme

  - **the program have to adapt to the machine**

- **DSPs are Adapted for a class of applications.**
  - **Flexibility and efficiency only for a given class of applications.**

- **ASICs are**

  Tailored for one application.

  - **Very efficient in speed and resource.**

  Cannot re-adapt to a new application

  - **Not flexible**

Reconfigurable device: Goal

The Ideal device should combine:

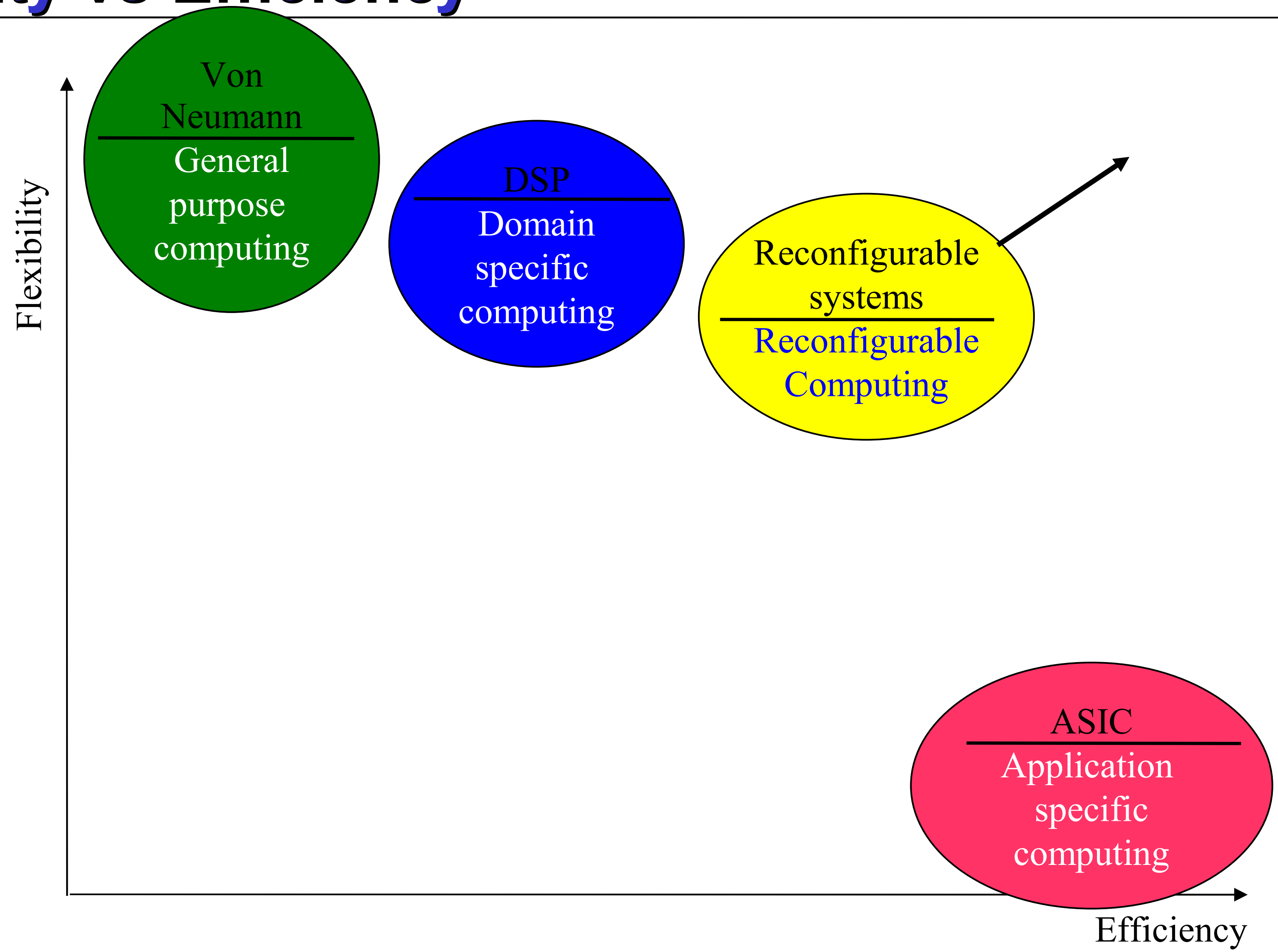
- **the flexibility of the Von Neumann computer**
- **the efficiency of ASICs**

The ideal device should be able to

- **Optimally implement an application at a given time**
- **Re-adapt to allow the optimal implementation of a new application.**

We call such a device a reconfigurable device.

Flexibility vs Efficiency



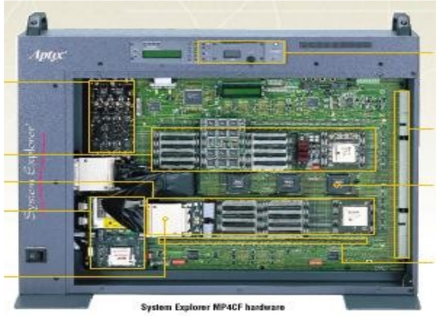
Fields of application

- Rapid prototyping
- Post fabrication customization
- Multi-modal computing tasks
- Adaptive computing systems
- Fault tolerance
- High performance parallel computing

Rapid Prototyping

Testing hardware in real conditions before fabrication

- Software simulation
  - Relatively inexpensive
  - Slow
  - Accuracy ?
- Hardware emulation
  - Hardware testing under real operation conditions
  - Fast
  - Accurate
  - Allow several iterations



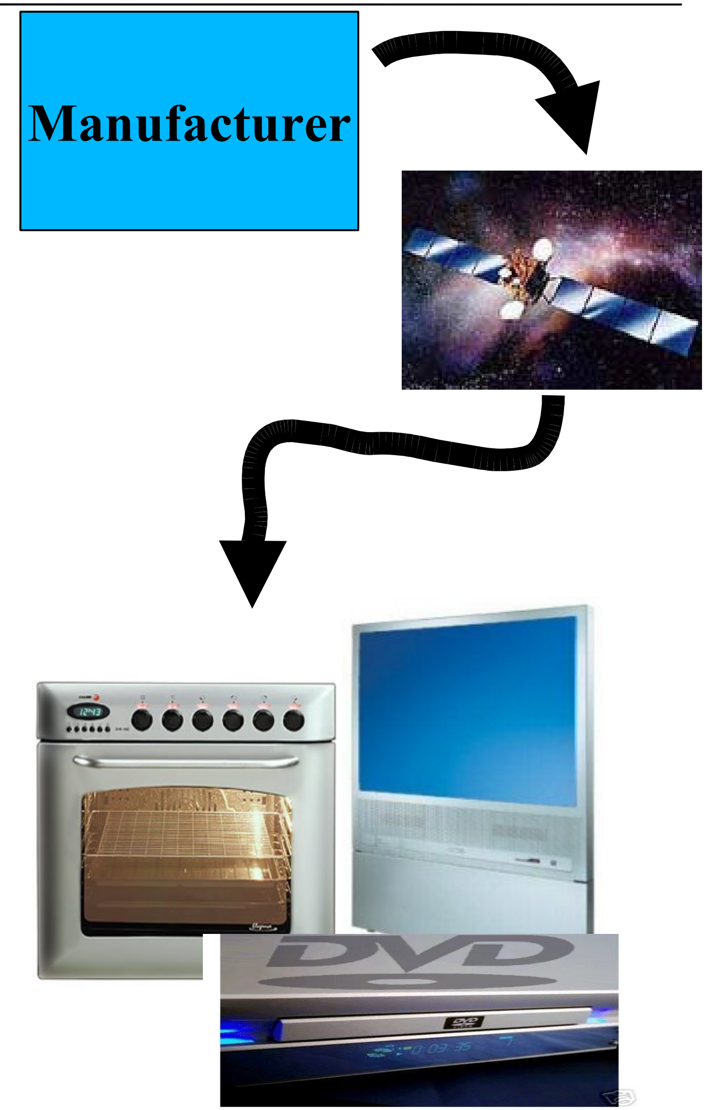
APTIX System Explorer

ITALTEL FLEXBENCH

Post fabrication customization

Time to market advantage

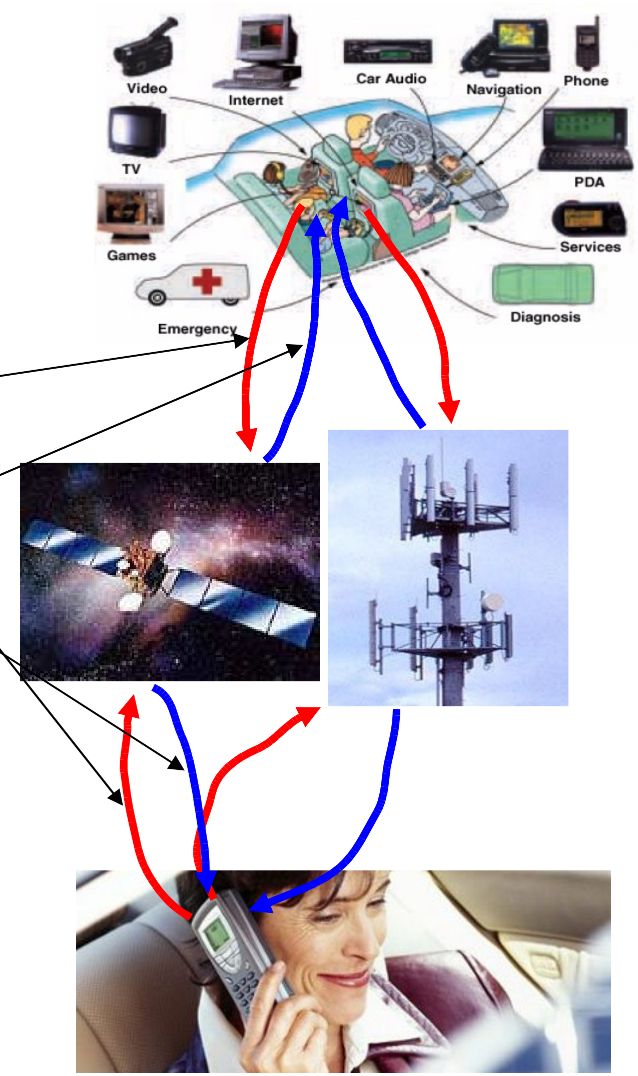
- Ship the first version of a product
- Remote upgrading with new product versions
- Remote repairing



Multi-modal computing tasks

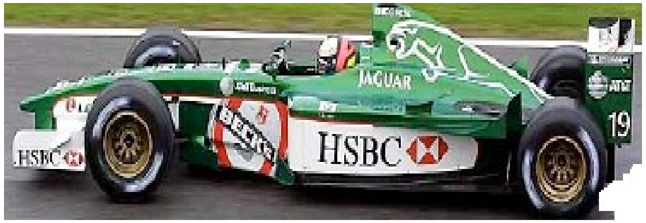
Reconfigurable vehicles, mobile phones, etc..

- Built-in Digital Camera
- Video phone service
- Games
- Internet
- Navigation system
- Emergency
- Diagnostics
- Different standard and protocols
- Monitoring
- Entertainment



Adaptive computing systems

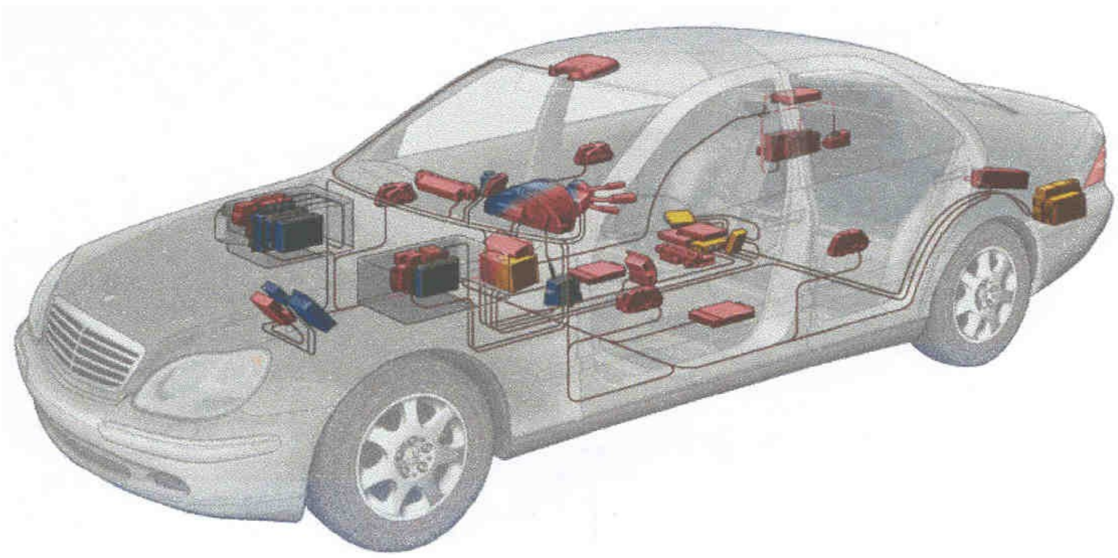
Computing systems that are able to adapt their behaviour and structure to changing operating and environmental conditions, time-varying optimization objectives, and physical constraints like changing protocols, new standards, or dynamically changing operation conditions of technical systems.

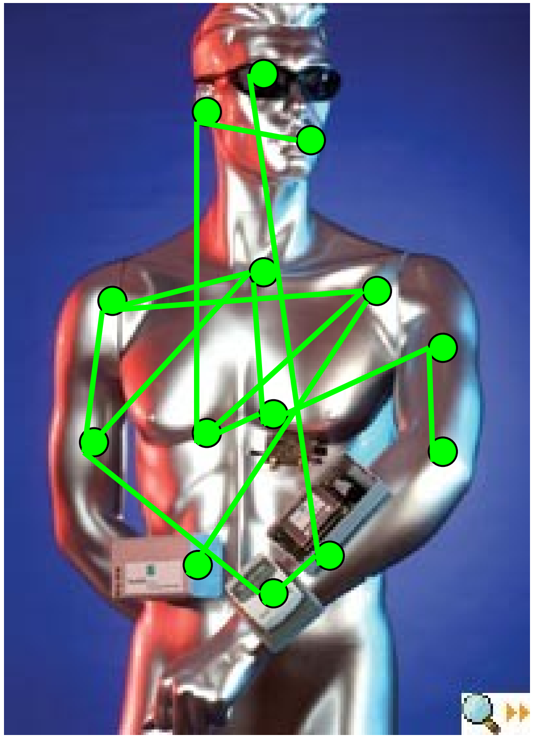


- Dynamic adaptation to environment
- Dynamic adaptation to threats (DARPA)
- Extended mission capabilities

Fault tolerance

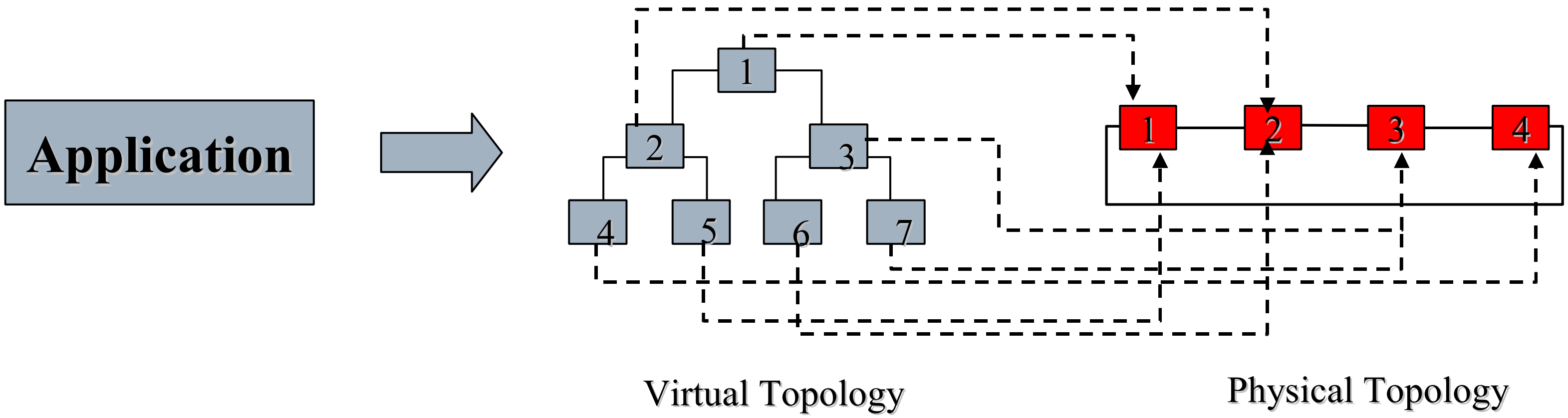
- **The RecoNet project**
  1. **Packet-oriented fault detection on communication lines**
  2. **Detections of defect nodes**
  3. **Task migration on node failure**
  4. **Load balancing computation**



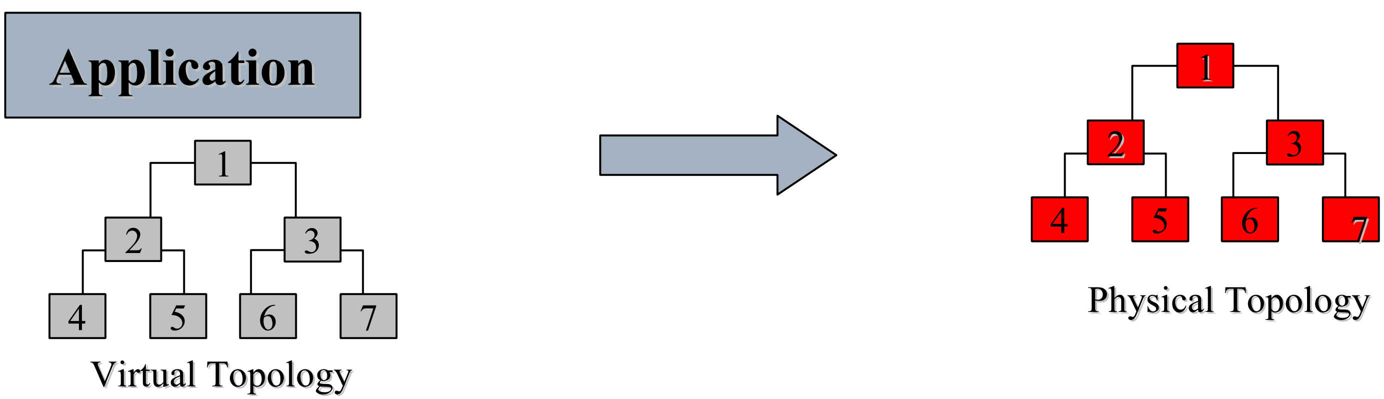


High performance parallel computing

Traditional parallel implementation flow



Exploiting reconfigurable topology



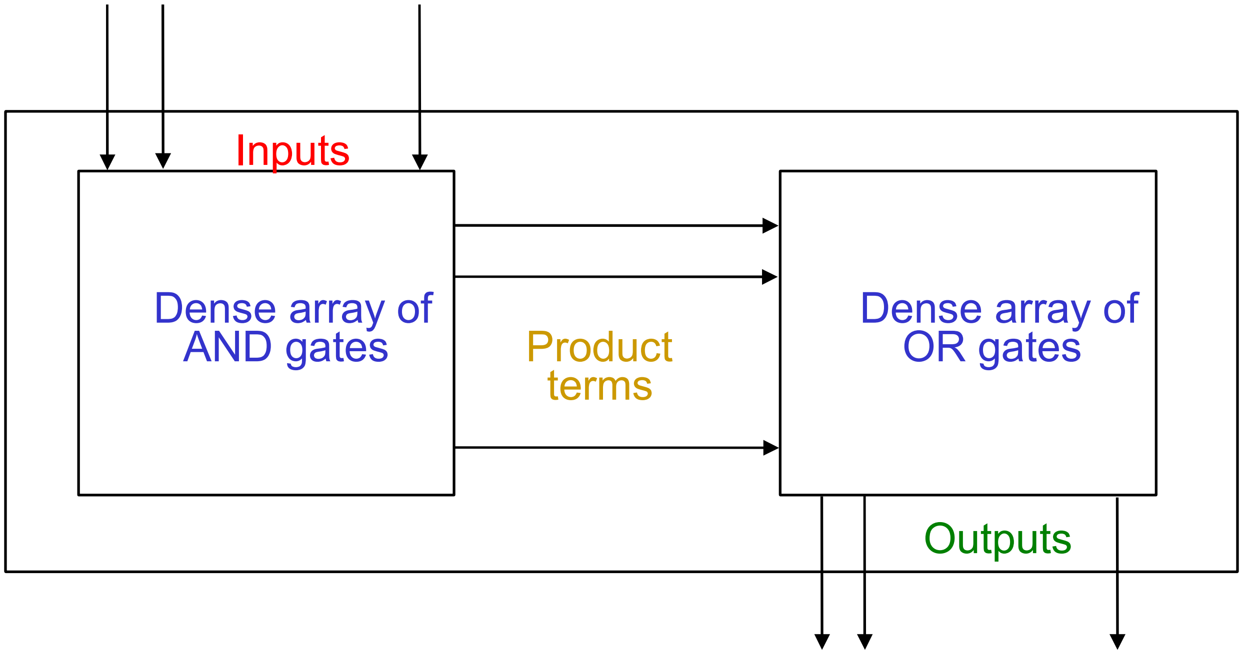
Reconfigurable architectures

Dr. rer. nat. Christophe Bobda
Lehrstuhl für Hardware-Software-Co-Design

Fine-grained reconfigurable devices

PALs and PLAs

- Pre-fabricated building block of many AND/OR gates (or NOR, NAND)
- "Personalized" by making or breaking connections among the gates



Programmable Array Block Diagram for Sum of Products Form

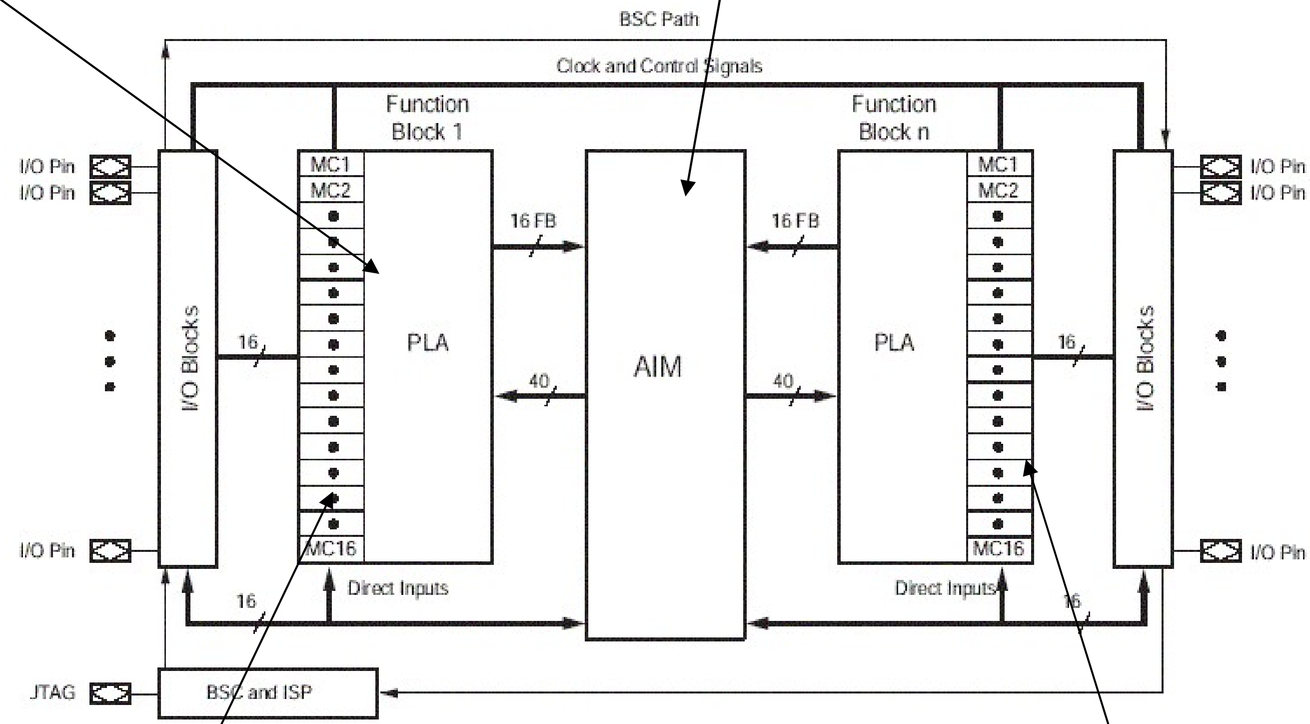
Complex Programmable Logic Devices

- Complex PLDs (CPLD) typically combine PAL combinational logic with Flip Flops
  - Organized into logic blocks connected in an interconnect matrix
  - Combinational or registered output
- Usually enough logic for simple counters, state machines, decoders, etc.
- CPLDs logic is not enough for complex operation
- FPGAs have much more logic than CPLDs
- e.g. Xilinx Coolrunner II, etc.

Xilinx Coolrunner CPLD

Function Bloc

Interconnection matrix



Macrocells for input connection

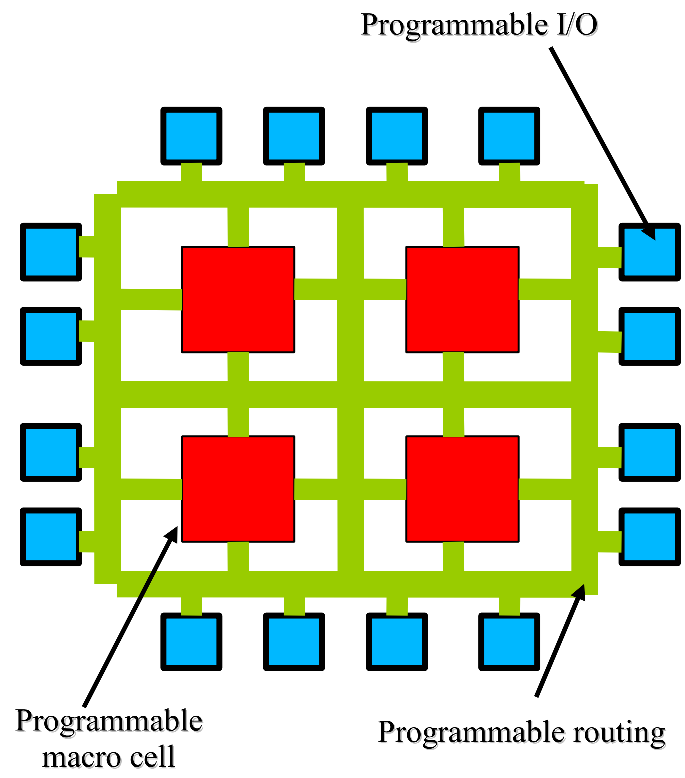
Macrocells for output connection

Field Programmable Gate Arrays (FPGAs)

Introduced in 1985 by Xilinx

Roughly seen, an FPGA consist of:

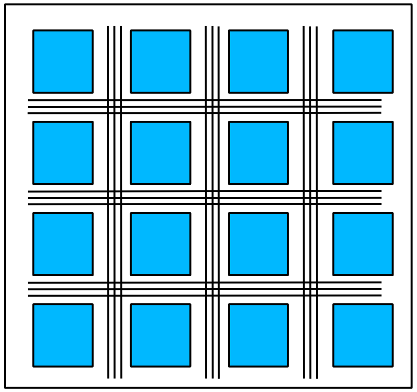
3. A set of programmable macro cells
4. A programmable interconnection network
5. Programmable input/outputs
6. Subparts of a (complex) function are implemented in macro cells which are then connected to build the complete function
7. The IO can be programmed to drive the macro cell's inputs or to be driven by the macro cell's outputs
8. Unlike traditional application-specific integrated circuit (ASIC), function is specified by the user *after* the device is manufactured
9. Physical structure and programming method is vendor dependant



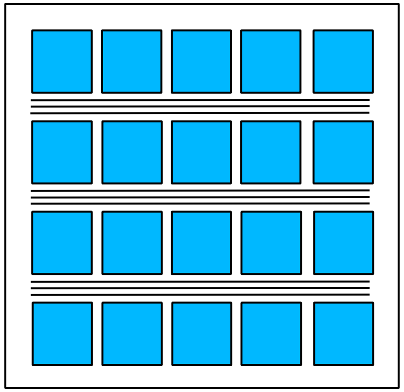
FPGA Structure

Typical organization

- Symmetrical Array
  - 2 D array of processing elements (PE) embedded in an interconnection network
  - Interconnection points at the horizontal-vertical intersection
- Row based
  - Rows of Processing elements
  - Horizontal routing via horizontal channels
  - Channels divided in segments
  - Vertical connections via dedicated vertical tracks (not on the graphic)



Symmetrical Array

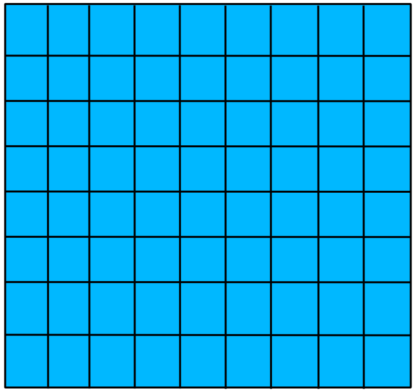


Row-based

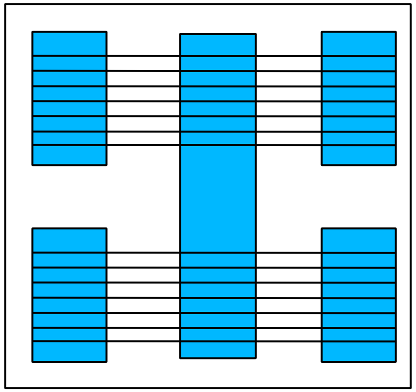
FPGA Structure

Typical organization (cont)

- Sea of gates
  - 2 D array of processing elements
  - No space left aside the PEs for routing
  - Connection is done on a separate layer on top of the cells
- Hierarchical
  - Hierarchically placed Macro cells
  - Low-level macro cells are grouped to build the higher-level's PEs



Sea of Gates

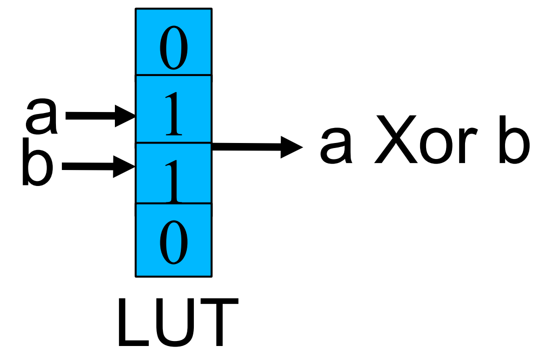


Hierarchical

FPGA Function generators

- **LUT**
  - LUT are used as function generators in SRAM-based FPGA
  - A function is implemented by writing all possible values that the function can take in the LUT
  - The inputs values are used to address the LUT and retrieve the value of the function corresponding the the input values
  - A k-inputs LUT can implement up to 2^k different functions
  - A k-input LUT has 2^k SRAM locations

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

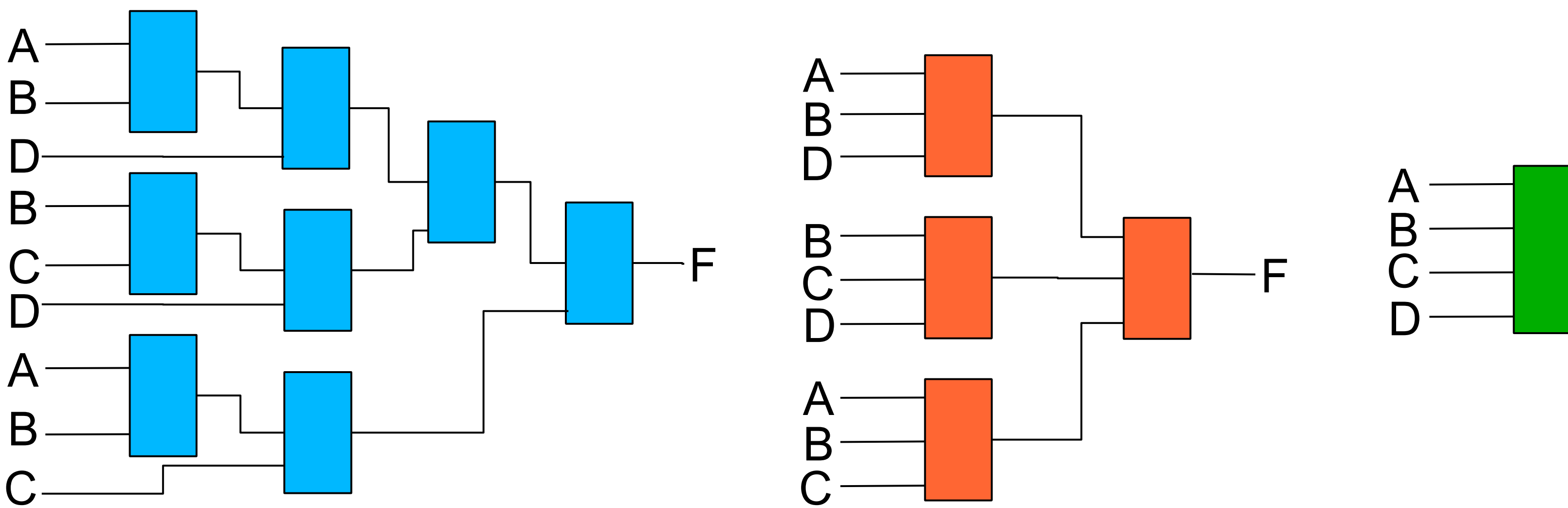


FPGA Function generators

LUT Example: Implement the function using:

$F = ABD + BC\bar{D} + \bar{A}\bar{B}\bar{C}$

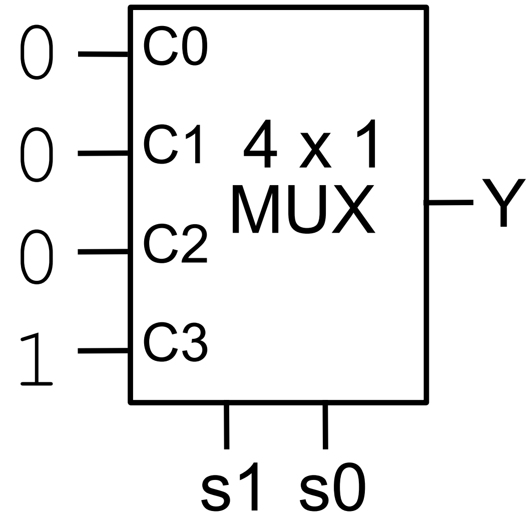
- 2-input LUTs
- 3-input LUTs
- 4-input LUTs



FPGA Function generators

Multiplexers (MUX)

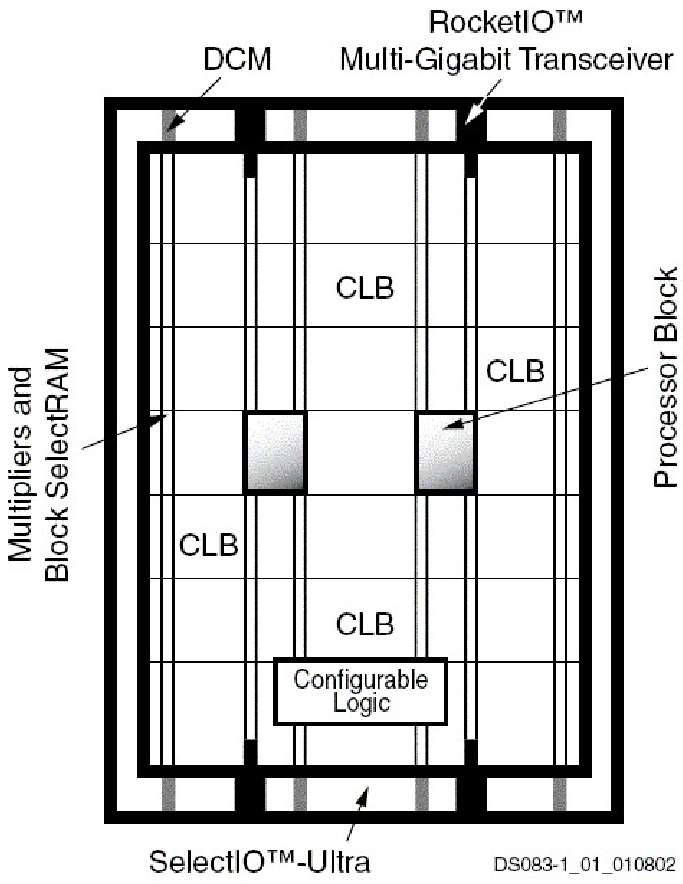
- A 2^kx1 MUX can implement up to 2^{k} different functions
- A function is implemented by writing all possible values that the function can take as constant at the MUX-Inputs
- The selector-values are used to pass the corresponding input to the MUX output
- Complex function can be decomposed and implement using many MUXes using the Shannon expansion theorem



s1	s0	Y =AND	
0	0	C0	0
0	1	C1	0
1	0	C2	0
1	1	C3	1

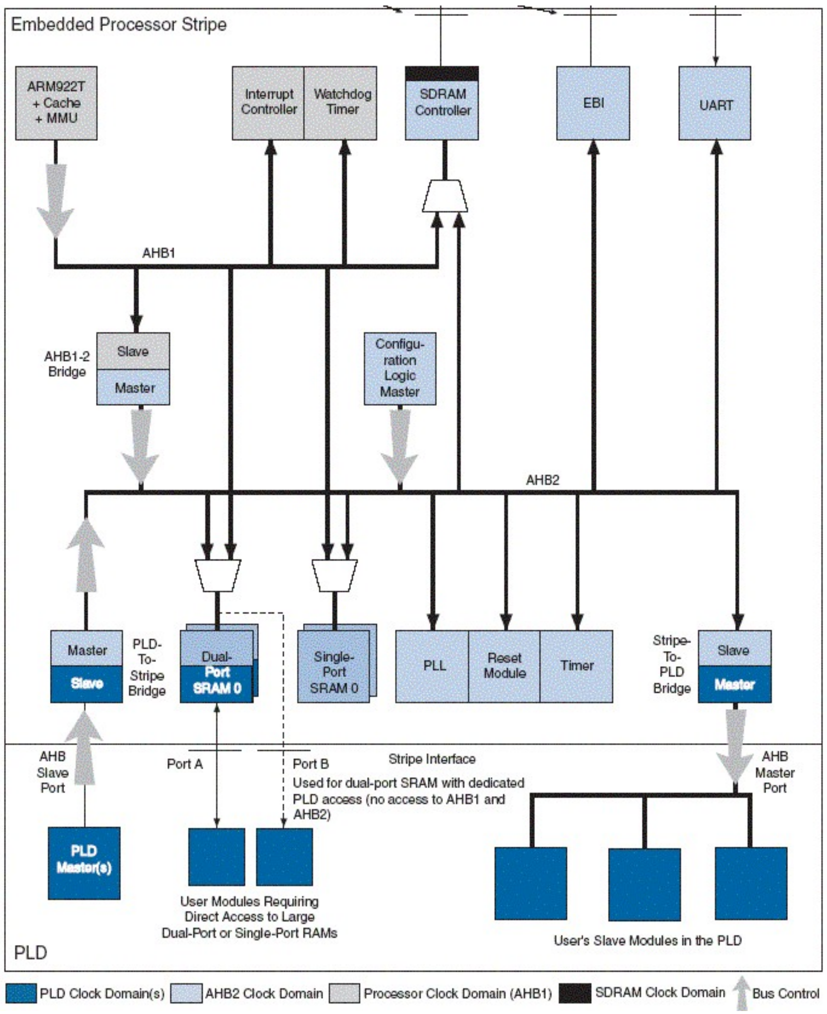
Hybrid FPGAs

1. The Xilinx VirtexII-Pro
2. Basic structure: VirtexII
3. Additional features
    1. Up to 4 hard-core embedded IBM power pc 405 RISC processors with 300+ Mhz
    2. Advanced 18bit x 18bit embedded multipliers
    3. Dual-ported RAM
    4. Embedded high speed serial RocketIO multi-gigabit transceivers



Hybrid FPGAs

1. The Altera Excalibur
2. Specific features:
   1. One ARM922T 32-bits RISC processor with 200 Mhz
   2. Embedded multipliers
   3. Internal single and dual-ported RAM and SDRAM controller
   4. Expansion bus interface for flash-RAM connection
   5. Embedded SignalTap logic analyzer



Coarse-grained reconfigurable devices

Once again: General purpose vs Special purpose

- *With the LUT as function generators, FPGA can be seen as general purpose devices.*
- *Like any general purpose device, they are flexible and "inefficient"*
- *Flexible because any n-variables Boolean function can be implemented in a n-input LUT.*
- *Inefficient since complex functions must be implemented in many LUTs at different locations.*
  - ➔ *The connections among the LUTs is done using the routing matrix wich increases the signal delays*
- *LUT implementation is usually slower than dircect „wiring"*

Once again: General purpose vs Special purpose

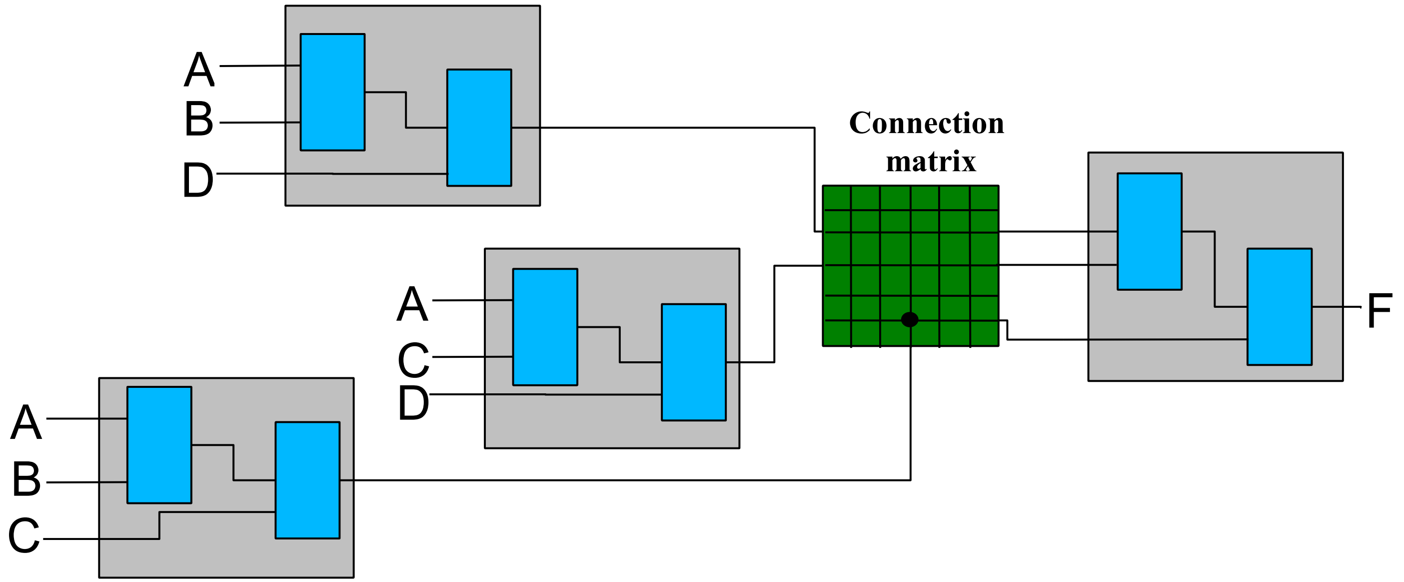
Example: ***Implement the function*** $F = ABD + AC\overline{D} + \overline{A}\,\overline{B}\,\overline{C}$

using 2-input LUTs.

LUTs are grouped in logic blocks (LB). 2 2-input LUT per LB

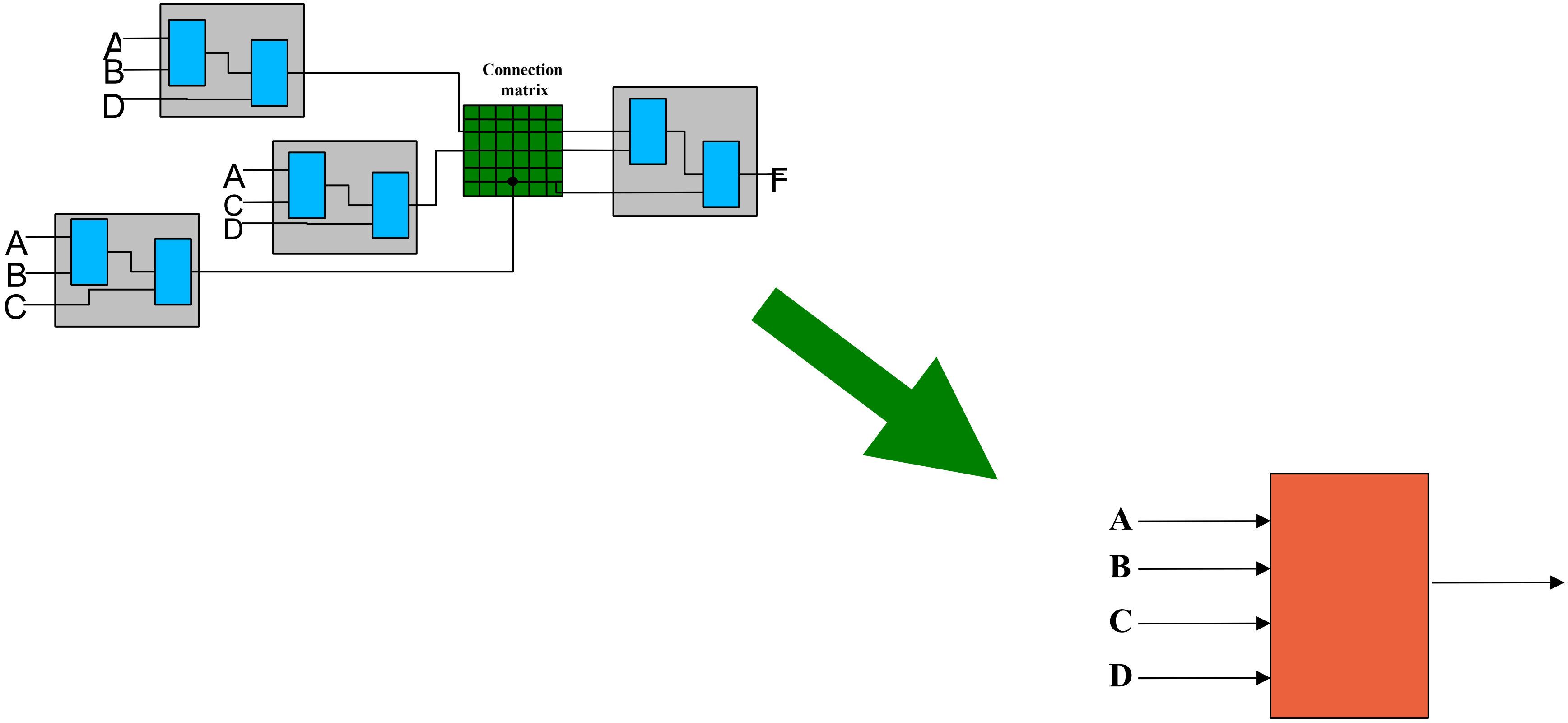
Connection inside a LB is efficient (direct)

Connection outside LBs are slow (Connection matrix)



Once again: General purpose vs Special purpose

Idea: ***Implement frequently used blocks as hard-core module in the device***



Coarse grained reconfigurable devices

- *Overcome the inefficiency of FPGAs by providing coarse grained functional units (Adder, multipliers, integrators, etc...), efficiently implemented*
- *Advantage: Very efficient in term of speed (no need for connections over connection matrice for basic operators)*
- *Advantage: Direct wiring istead of LUT implementation*
- *Usually an array of programmable and identical processing element (PE) capable of executing few operations like addition and multiplication.*
- *Depending on the manufacturer, the functional units communicate via busses or can be directly connected using programmable routing matrices*

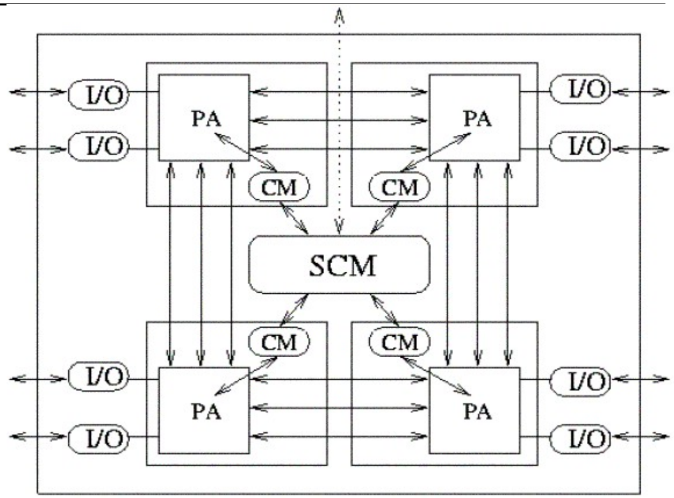
Coarse grained reconfigurable devices

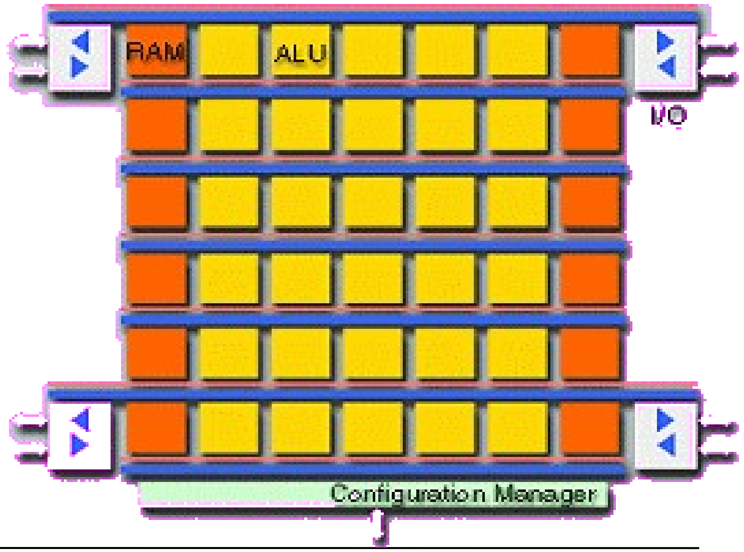
- ***Memory exist between and inside the PEs.***
- ***Several other functional units according to the manufacturer.***
- ***A PE is usually an 8-bit, 16-bit or 32-bit tiny ALU which can be configured to executed only one operation on a given period (until the next configuration)***
- ***Communication among the PEs can be either packet oriented (on busses) or point-to-point (using crossbar switches)***
- ***Since each vendor has its own implementation approach, study will be done by mean of few examples. Considered are: PACT XPP, Quicksilver ACM, NEC DRP, picoChip, IPflex DAP/DNA***

The PACT XPP – Overall structure

XPP (Extreme Processing Platform) is a hierarchical structure consisting of:

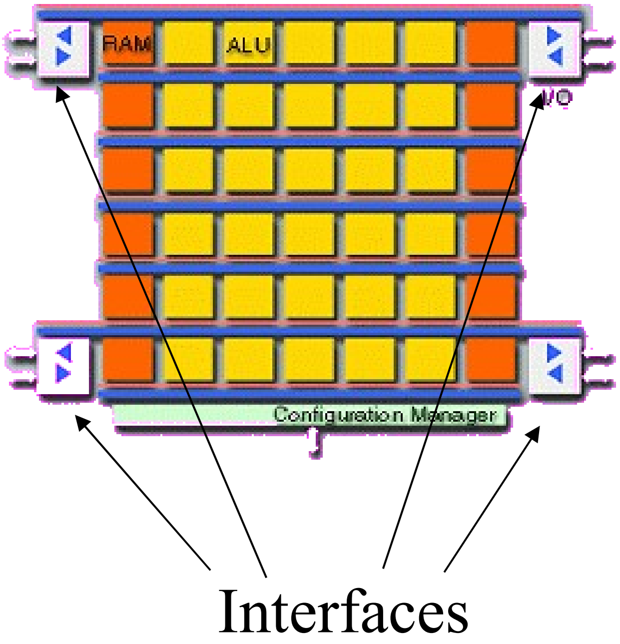
- An array of Processing Array Elements (PAE) grouped in clusters called Processing Arrays (PA)
- PAC = Processing Array Cluster (PAC) + Configuration manager (CM)
- A hierarchical configuration tree
- Local CMs manage the configuration at the PA level
- The local CMs access the local configuration memory while Supervisor CM (SCM) access external memory and supervise the whole configuration process on the device





The PACT XPP - Interface

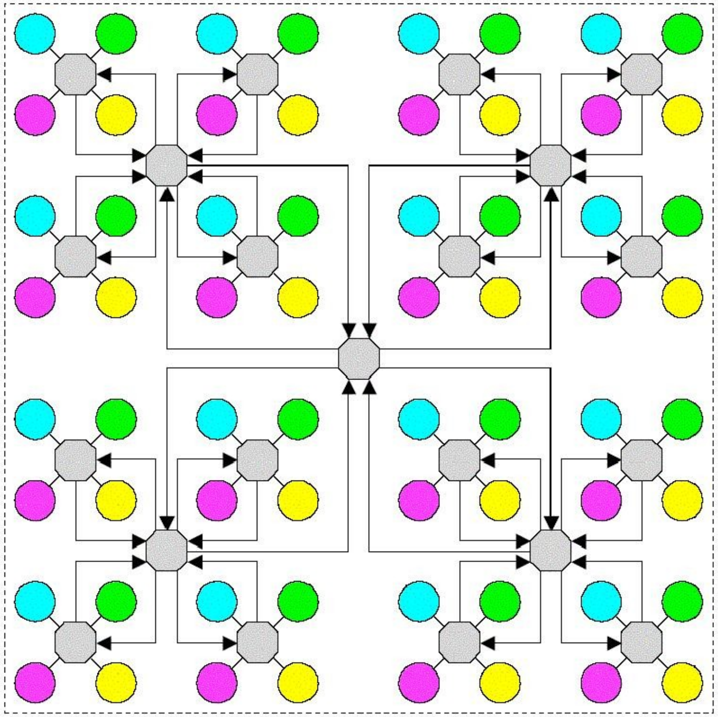
- *Interfaces are available inside the chip*
  - *Number and type of interfaces vary from device to device*
- *On the XPP42-A1:*
- *6 internal interfaces consisting of:*
  - *4 identical general purpose I/O on-chip interfaces (bottom left, upper left, upper right, and bottom right)*
  - *One configuration manager (not shown on the picture)*
  - *One JTAG (Join Test Action Group, "IEEE Standard 1149.1") Boundary scan interface or for testing purpose*



The Quicksilver ACM - Architecture

Structure: Fractal like structure

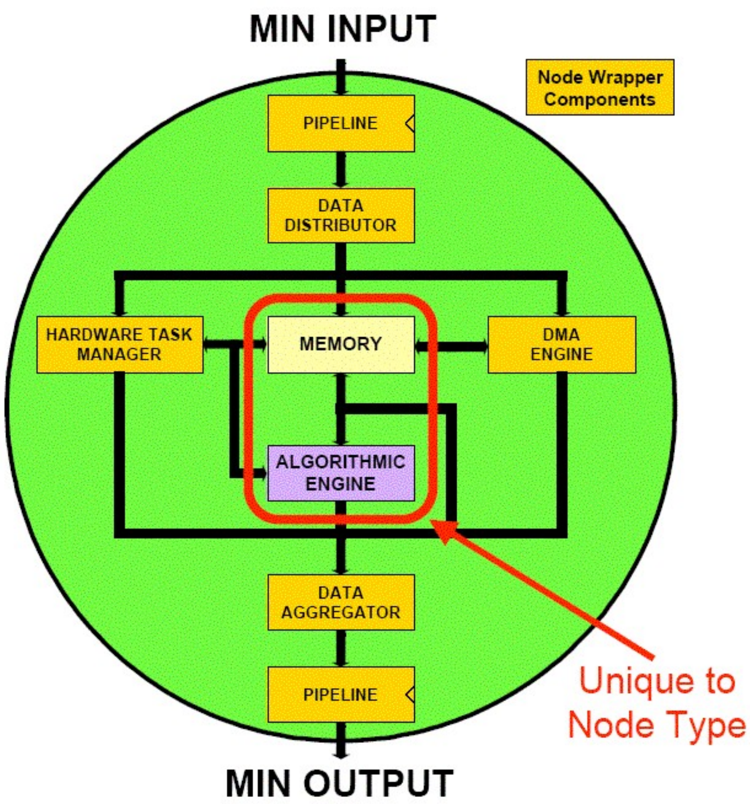
- *Hierarchically group of four nodes with full communication among the nodes*
- *4 lower level nodes are grouped in a higher level node*
- *The lowest level consist of 4 heterogeneous processing nodes*
- *The connection is done in a Matrix Interconnect Network (MIN)*
- *A system controller*
- *Various I/O*



The Quicksilver ACM – The processing node

The node wrapper Envelopes the algorithmic engine and presents an identical interface to neighbouring nodes. It features:

- *A MIN interface to support the communication among nodes via the MIN-network*
- *A hardware task manager for task management at the node level*
- *A DMA engine*
- *Dedicated I/O circuitry*
- *Memory controllers*
- *Data distributors and aggregators*

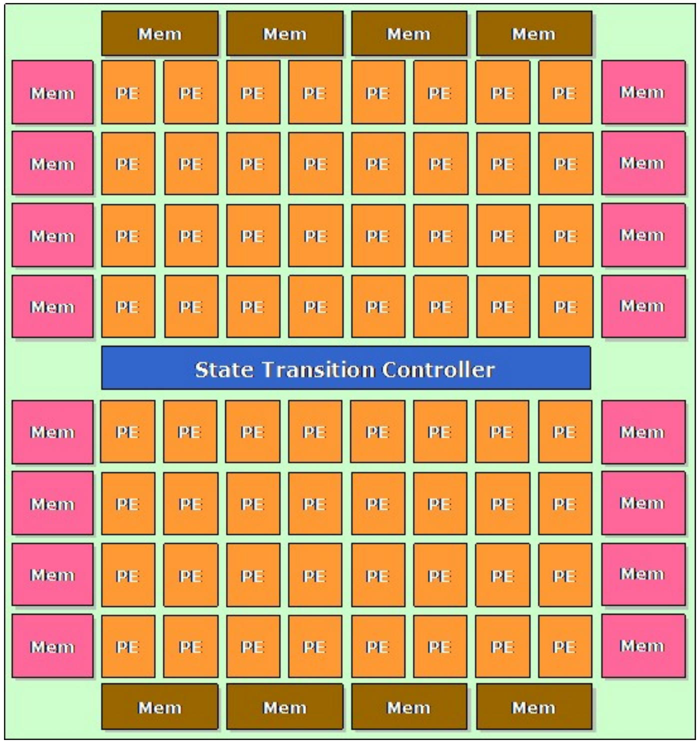


The ACM Node-Wrapper

The NEC DRP – Architecture

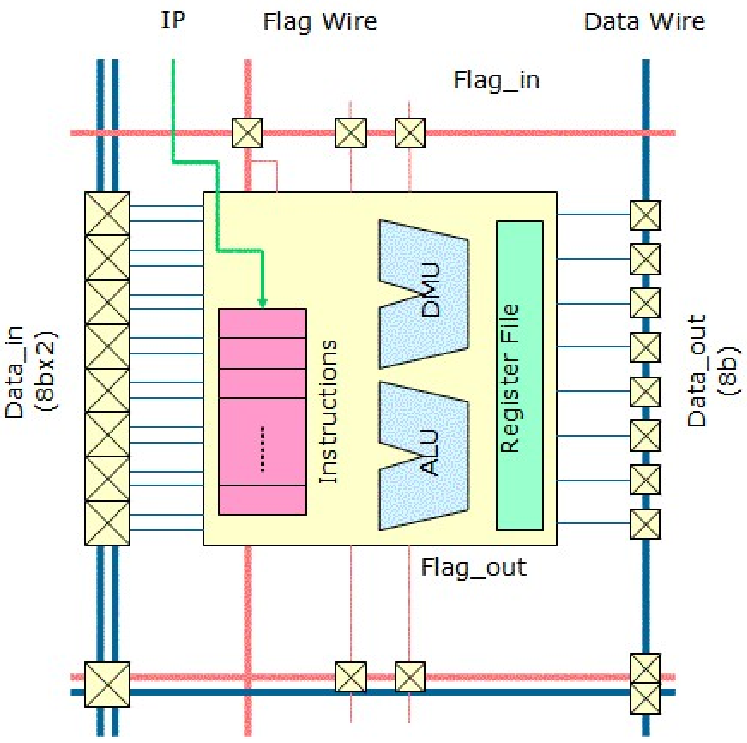
The NEC Dynamically Reconfigurable Processor (DRP) consists of:

- *A set of byte oriented processing elements (PE)*
- *A programmable interconnection network for communication among the PEs.*
- *A sequencer. Can be programmed as finite state machine (FSM) to control the reconfiguration process*
- *Memory around the device for storing configuration and computation data*
- *Various Interfaces*



The NEC DRP - The Processing Element

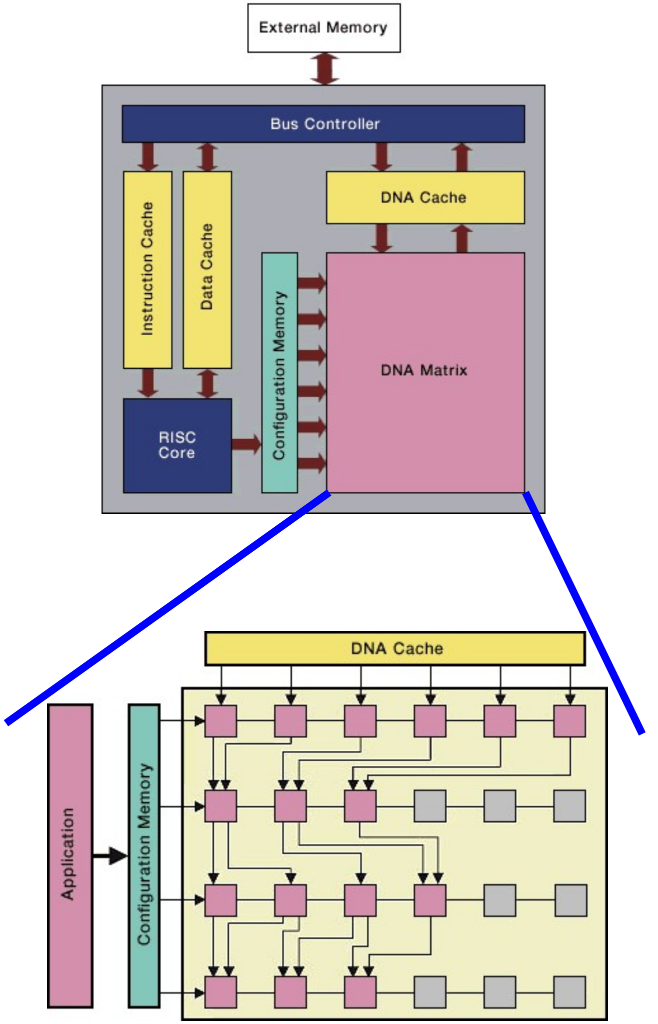
- ***ALU: ordinary byte arithmetic/logic operations***
- DMU (data management unit): handles byte select, shift, mask, constant generation, etc., as well as bit manipulations
- An instruction dictates ALU/DMU operations and inter-PE connections
- Source/destination operands can either from/to
  - ***its own register file***
  - ***other PEs (i.e., flow through)***
- Instruction pointer (IP) is provided from STC (state transition controller)



The IPflex DAP/DNA - Structure

The IPflex DAP/DNA has the structure of a System on Chip (SoC) with an embedded FPGA. It features:

- ***Integrated RISC core***
  - ***Carry some computation***
  - ***Controls the reconfiguration process***
- ***A Distributed Network Architecture (DNA) matrix (matrix of configurable operation units)***
- ***Communication over an internal bus***
- ***Different caches for data, instructions and configuration***
- ***I/O and memory Interface controllers***



The picoChip - Architecture

- *Hundreds of array elements each with versatile 16-bit processor and local data*
- *heterogeneous architecture with four types of elements optimized for different tasks (DSP or wireless function).*
- *Interface for:*
  - *SRAM*
  - *Host communication*
  - *External systems*
  - *Inter picoChip system*