Reconfigurable Systems in terms of Computer Architectures

Kees A. Vissers

Research Fellow, UC Berkeley
vissers@ieee.org
www.eecs.berkeley.edu/~vissers

July 10. 2003

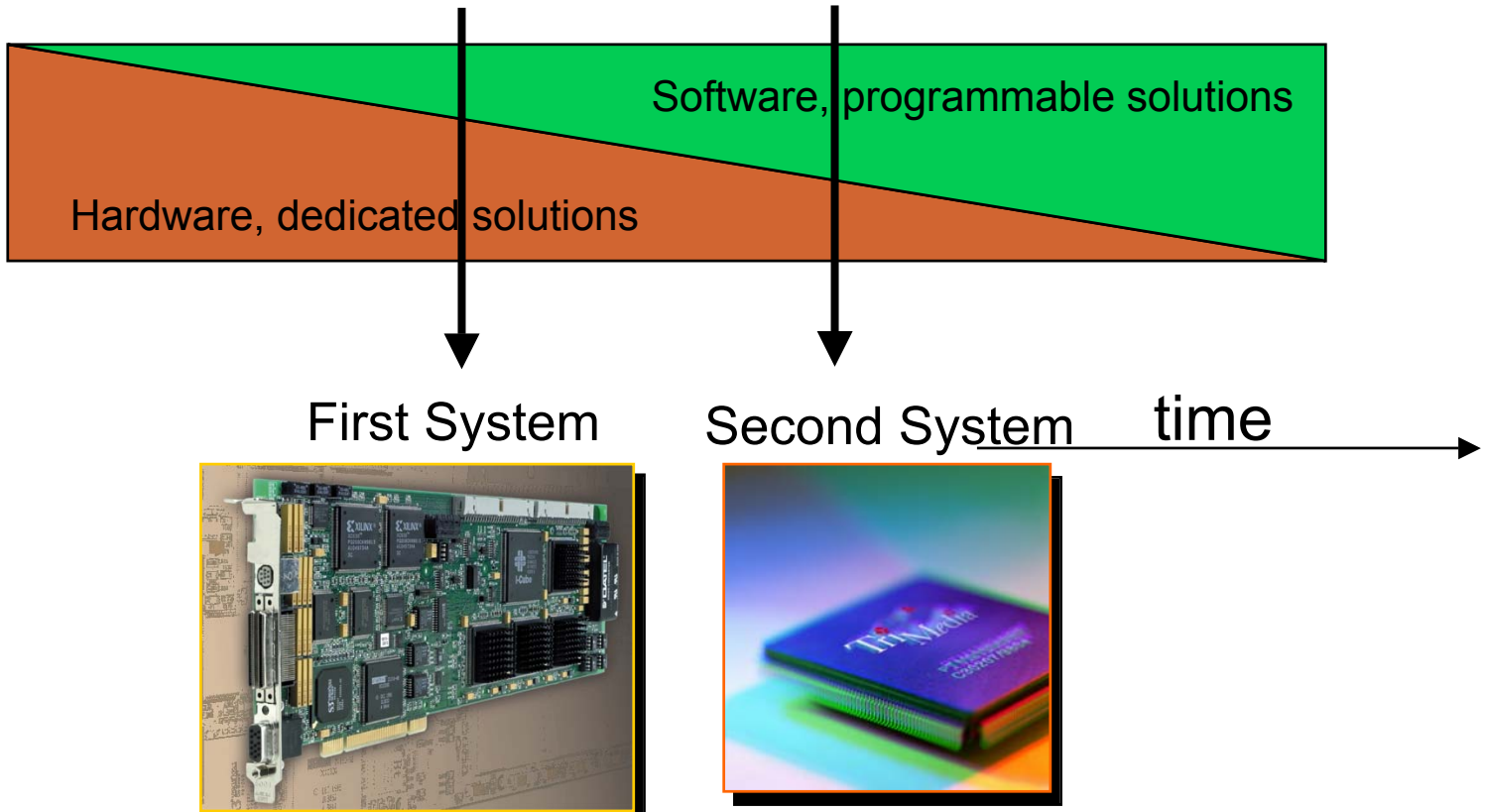
CHAMELEON SYSTEMS, INC.

Overview

- Introduction
- Video Signal Processors, Network of Alus
- Dynamic Reconfiguration
- Conclusions

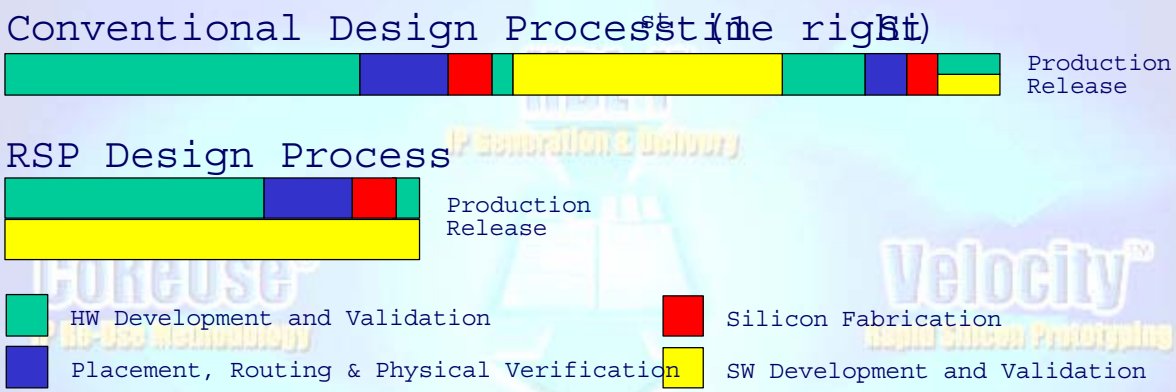
System on a chip

Silicon Technology is providing the opportunity to
add new functionality and integrate several functions
and allow more programmable/reconfigurable systems.



Theo Claasen: keynote Dac2000

Rapid silicon prototyping
Design cycle benefits



- Faster chip development
  - More than 50% total design cycle reduction
  - True HW / SW co-development
- Higher probability of first silicon success

PS - Theo Claasen DAC 2000 - 14

Let's make things better.

PHILIPS

Future: The prototype = product

- Trade-off between
  - HW/SW trade-off: Processor and reconfigurable fabric
  - Granularity issue: fine grain processing ILP, or Task level
  - bit oriented – word oriented
  - distributed on chip memory
- Large number of architectural choices:
- Which is 'the right one' for a particular application domain?

Problem definition

- Perform processing on a stream of data
- Sampling rates in the order of 100KHz to 100MHz
- per sample 1000 – 100,000 operations

Perform 10^{10}-10^{11} operations/sec, like add, multiply etc.

Take a 200MHz Alu, still need 50-500 of them,

Solution: Time multiplex and multi-processor

Note: performance is a required part of the solution!

Revisit processor architectures

- Single Risc like processor
- ILP processing: VLIW for DSP, superscalar for general purpose
- Very successful programming environment: compilers and OS.
- Multi processor: no clear winning model, suggested to move to chapter 11 in Computer Architecture, a quantitative approach from Hennessy and Patterson
- Vector processing

What are the solution options

Concurrently execute 50- 500 operations every clock cycle.

- Multiple Risc cores, e.g. ARM, MIPS etc.
- Multiple VLIW oriented DSPs, e.g TI, Starcore etc
- Build a bit oriented FPGA and synthesize everything on top of that, including processors cores, packet routing networks etc.
- Build a fabric of interconnected ALUs (coarse grained FPGA)

SoC platforms exploiting the best part for the specific application (part).

Multi processor challenges

- Programming language problem: no concurrency
- Limited extraction of ILP out of sequential program

Why can we program processors?

Model of Computation matched to the Model of the Architecture

- C/C++: single point of control, loop : Memory model, branch
- CSP (Hoare), Occam(2): Semantics of handshake in Hardware: transputer channel
- Kahn Process Networks: distributed fifo organization, semantics in hardware or software

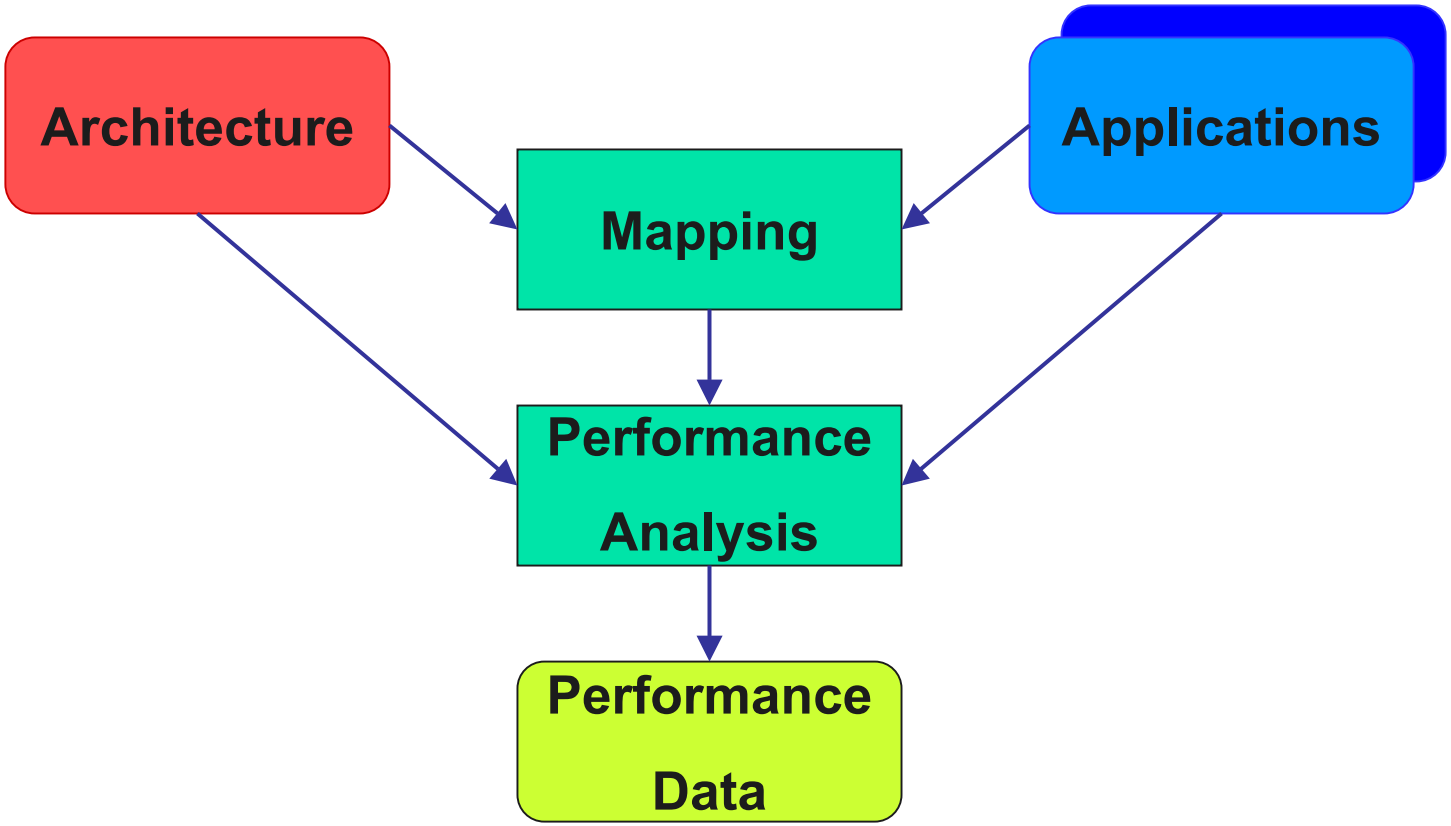
Reconfigurable programming challenges

In general 3 problems;

1. Programming a network of ALUs/ bit cells in FPGA
2. Setting up the buffers and partitioning for the network
3. Dynamic reconfiguration management

Architecture Design: Y - Chart

Benchmark driven, based on a Set of Applications



What is an instruction set processor

- C/C++, Java programming
- Program control translated to branches (most of the time)
  - for
  - if
  - case statements
- Single Program counter
- Data cache and Instruction cache
- Time-multiplex with instructions over ALUs
- Load, Store architecture, contains a Register File
- Debug with single stepping, breakpoints and register views

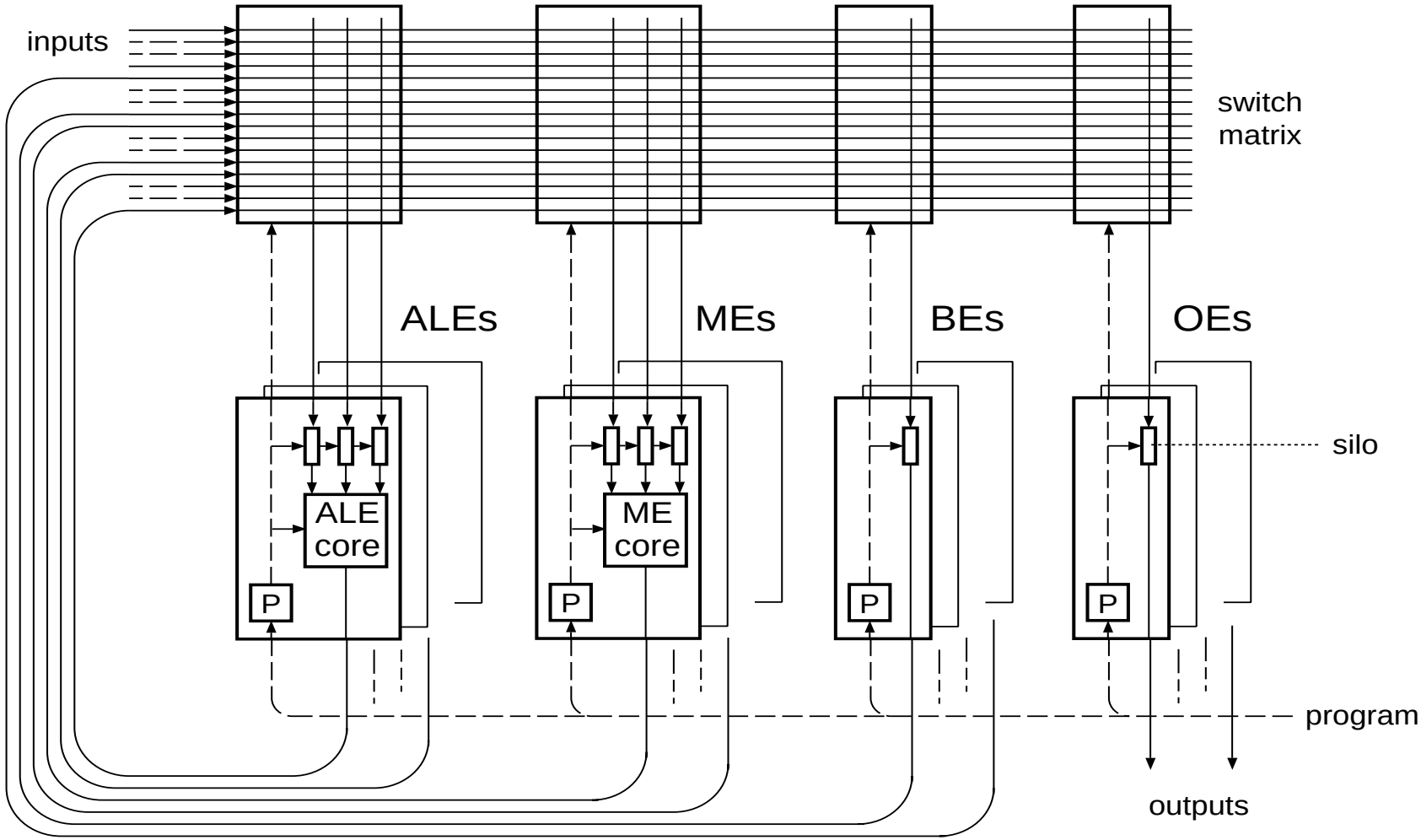
Multi-processors

- Multiple instruction set processors:
  - programmers model?
  - cache coherence?
  - granularity at the instruction level required
  - Instruction Level parallelism limited to 4-5
    - branch penalty in cycles
  - Operand routing and memory hierarchy are the cost
    - load-store instructions 30% of all instructions
    - L1 cache is half of the processor area
    - cache works poorly for stream oriented computing

The Programmability Issue

- Extract fine grain Parallelism out C/C++/ Matlab/Java
  - alias analysis and pointer problem
- Extract coarse grain Parallelism out C/C++/ Matlab/Java
  - process notion
- Start with an fine grain Parallel description
  - Hardware description languages (VHDL/Verilog)
  - Simulink environment, Signal Flow Graphs
- Start with an coarse grain Parallel description
  - CSP, occam
  - Kahn Process Networks
  - System C
  - ...

Exampe VSP architecture, 12-bit Elements



VSP1 and VSP2 Layouts

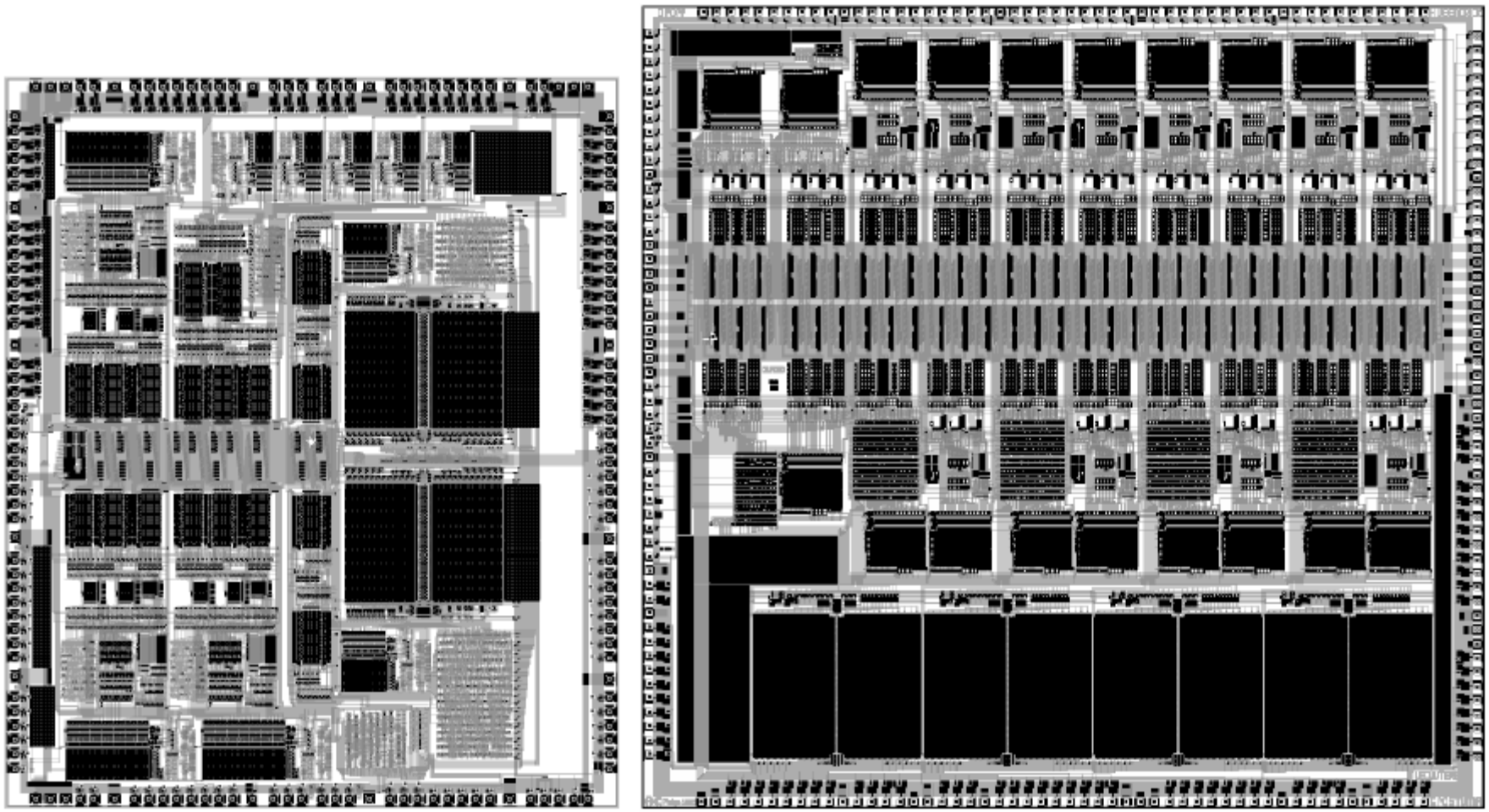
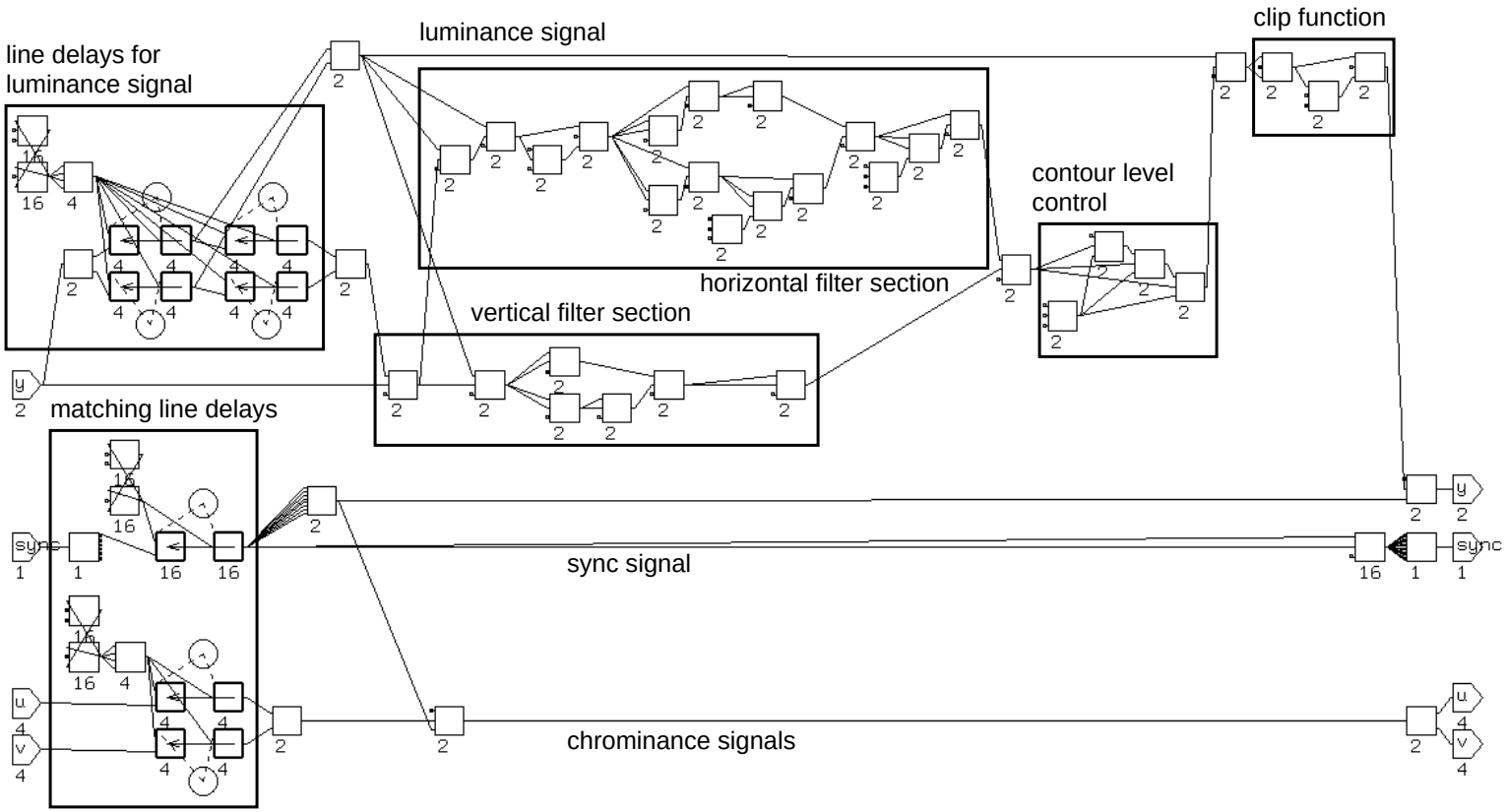


Figure 3: Layouts of VSP1 in 1.2μ CMOS and VSP2 in 0.8μ CMOS

Example programming VSP with SFG



Signal Flow Graph Mapping

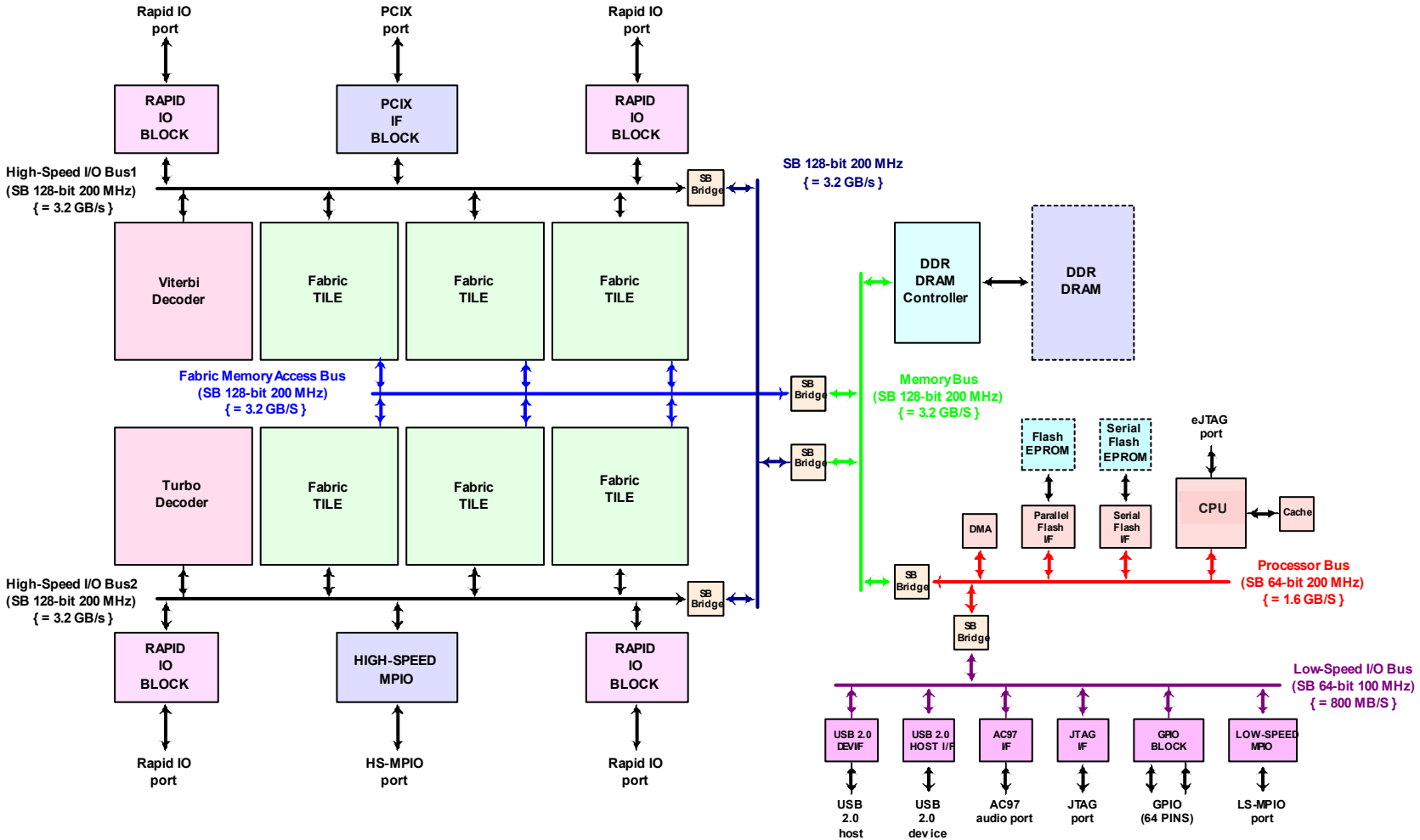
- Retiming, Delay management
- Partitioning, which part runs on which IC
- Scheduling, and processor allocation

- All for statically scheduled multi-rate signal flow graphs
- Basic operation translates to basic instruction
- explicitly programmed data memory storage
- no pointers, no branches
- inherent stream semantics
- Used for complete HDTV studio Camera Development, and next generation electronic X-ray equipment at Philips

Chameleon Second generation

- Network of interconnected ALUs (close to a VLIW)
- Explicit retiming methodology
- Programming in Simulink
- Buffer programming explicit with hardware specific choices
- Benchmark results for video and next generation cell phones, base stations etc.

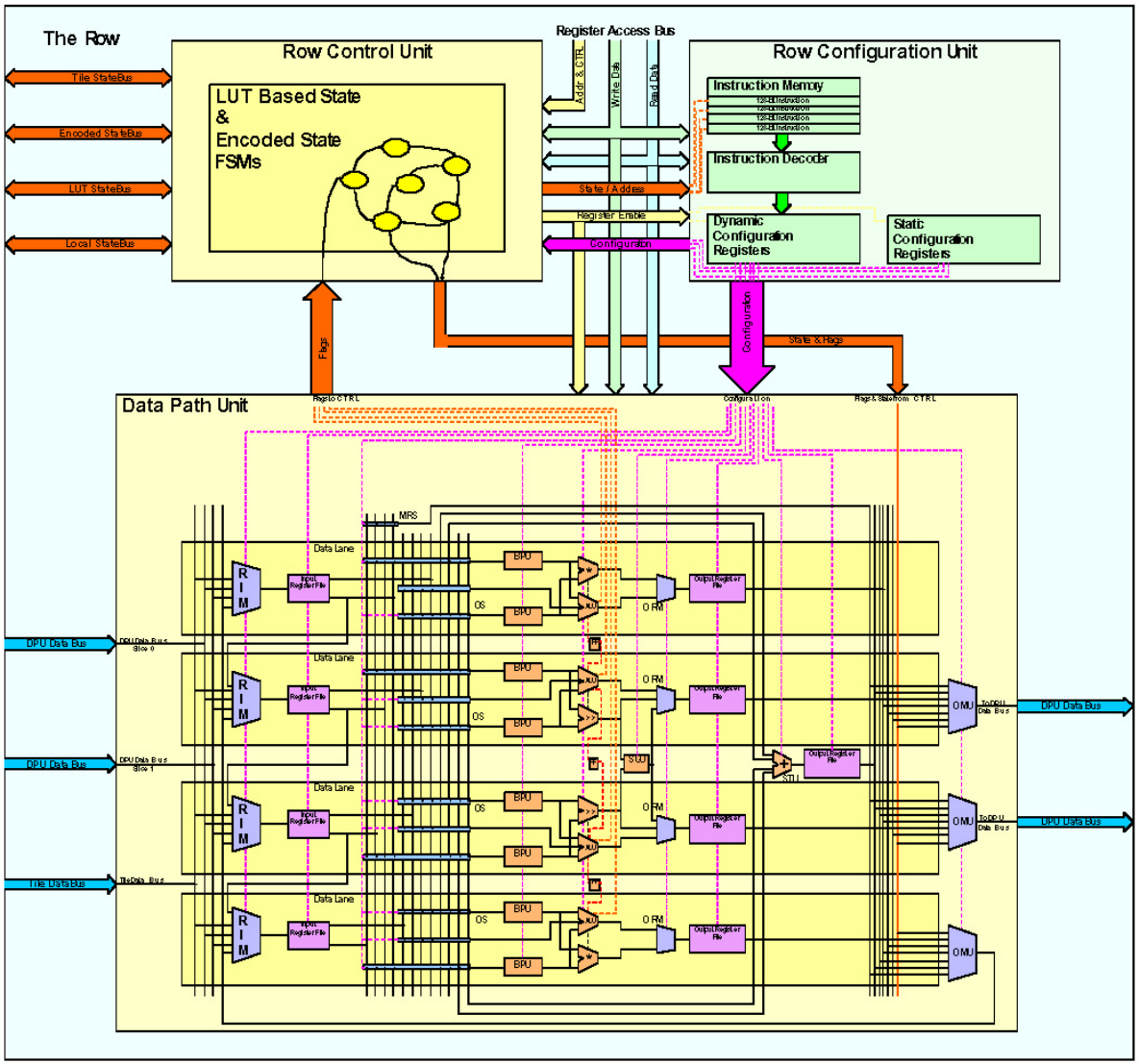
Example Chameleon 6 tile Architecture



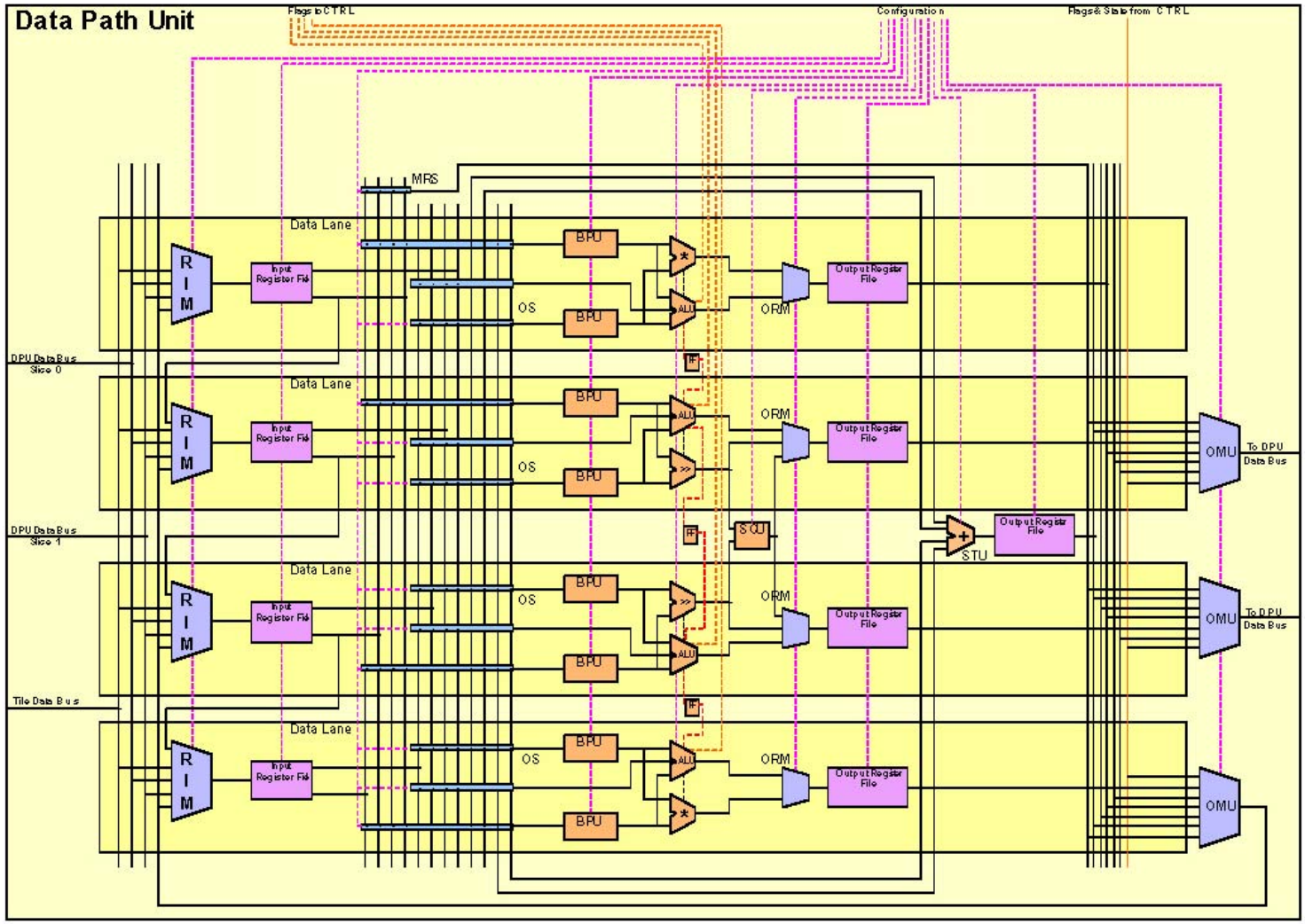
Tile Architecture

- 16 DPUs with full Interconnect
- 4 16KByte Memories
- DMA to System
- 32 Connections to nearest Neighbor

DPU datapath +control (conceptual)



DPU datapath Architecture (conceptual)

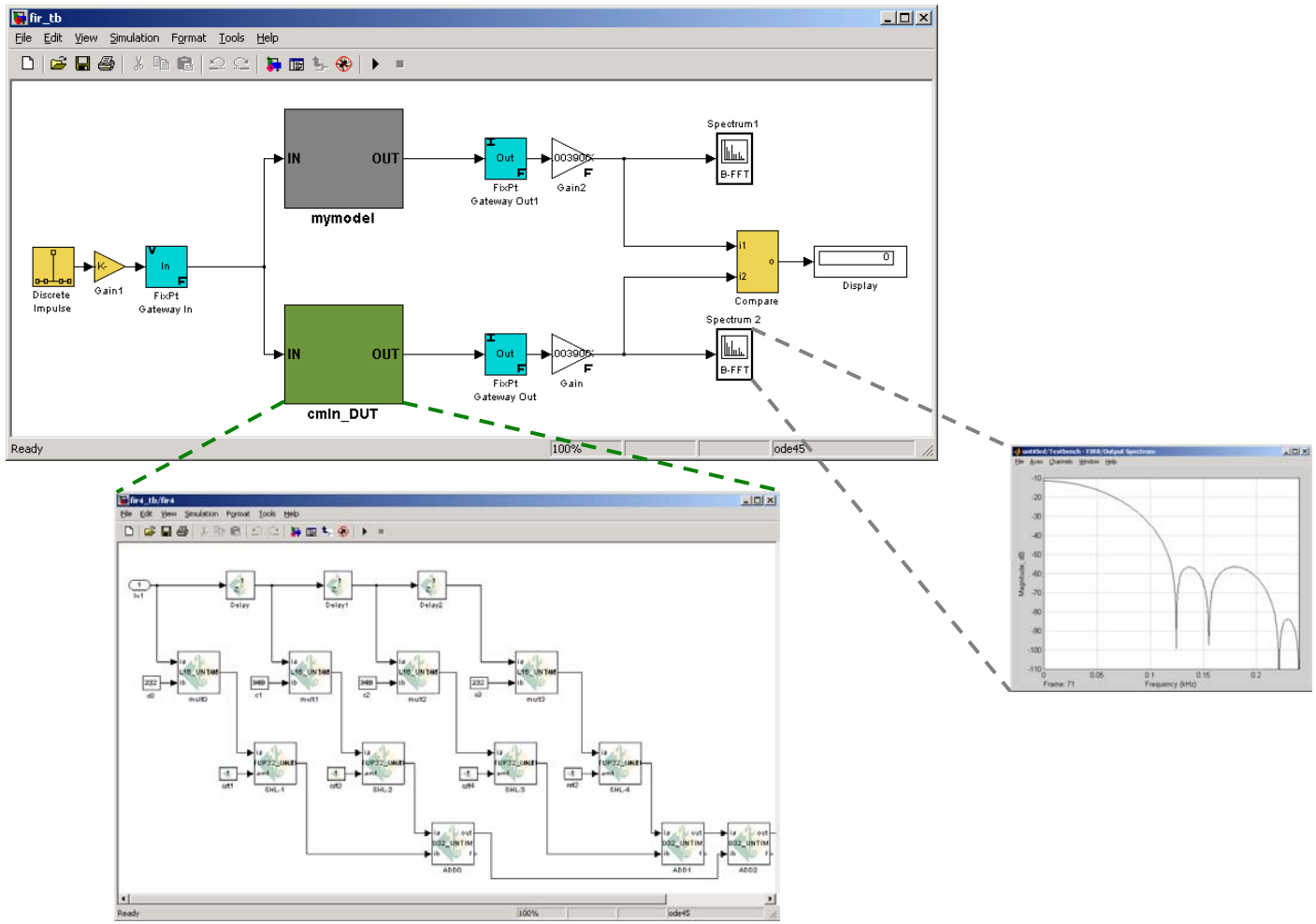


Programming the System

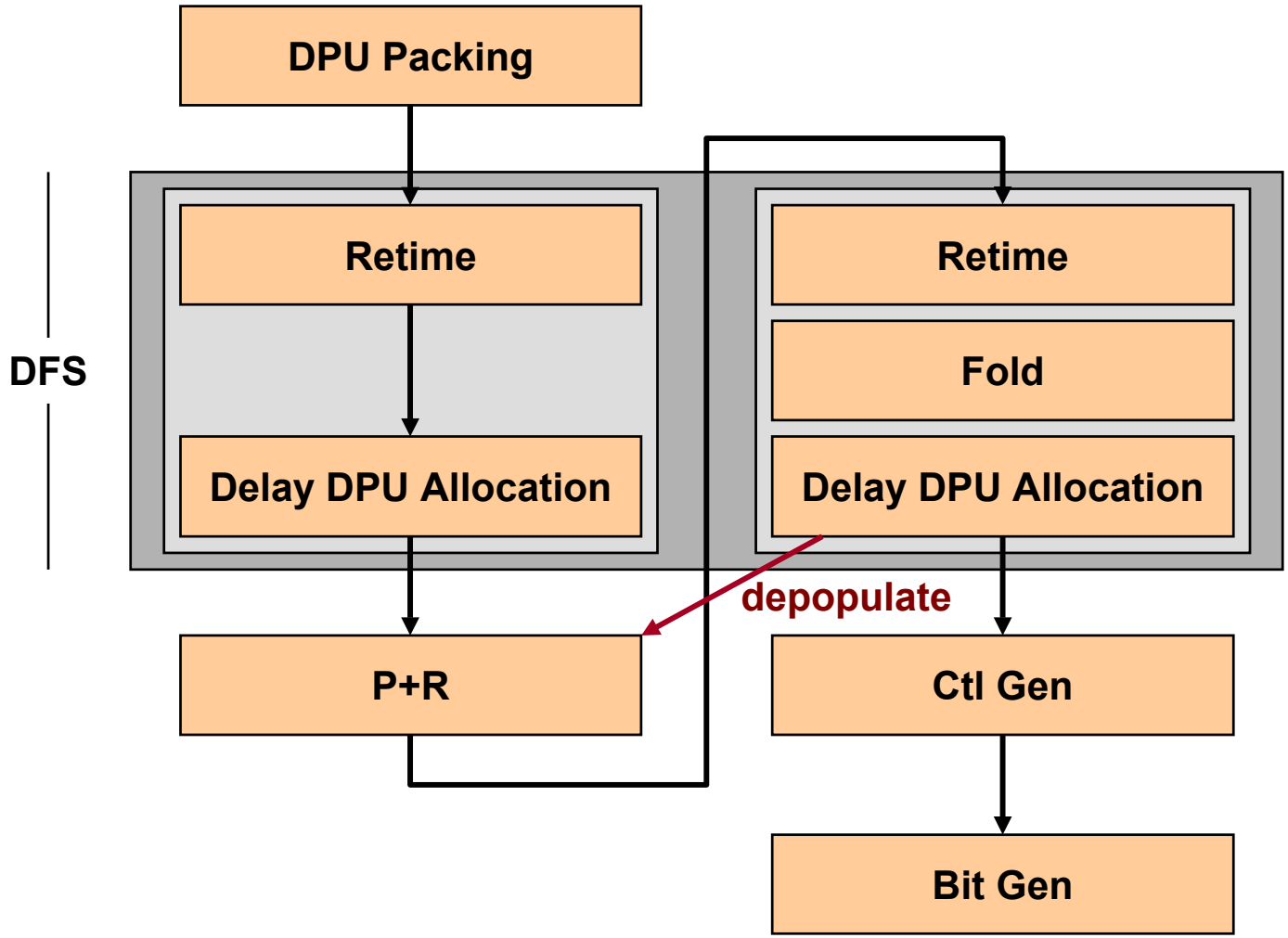
- Resource Access
  - DPU — Simulink Diagram — Untimed
  - Control — Simulink Diagram — Untimed
  - Fabric — Memory mapped
  - CPU — C code Linked via eFlow/INTs
  - Peripherals — Memory mapped/DMA
  - Memories — Memory mapped/DMA
- Programming Options
  - Enter control and dataflow in Simulink
  - C code entry with reference to dataflow kernels in fabric
    - Kernels are built with Simulink

Simulink based design entry systems

- Integrated entry and simulation environment



Detail - Scheduling and Mapping



DFS = Data Flow Scheduler

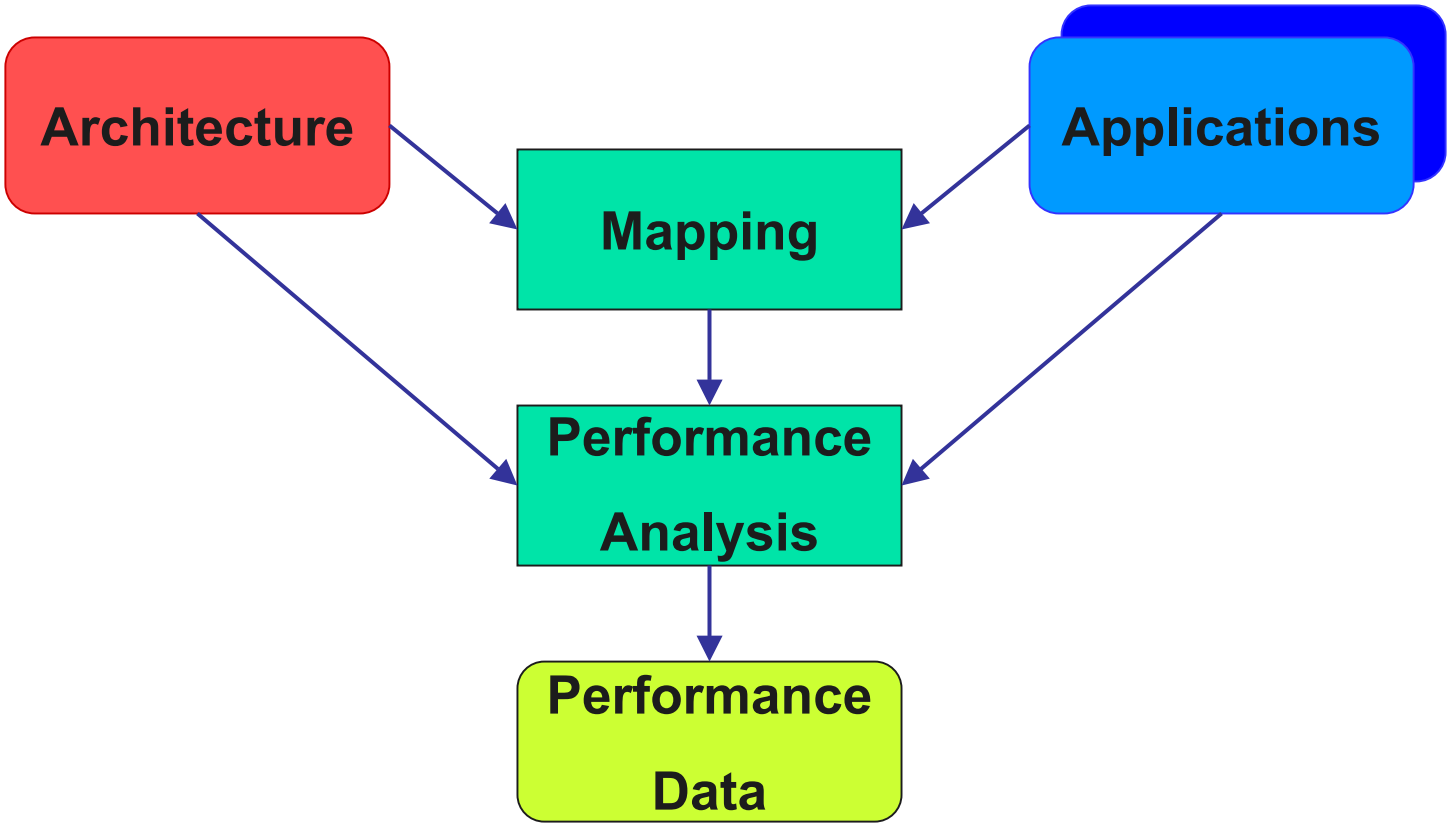
Reconfigurable programming challenges

In general 3 problems;

1. Programming a network of ALUs/ bit cell in FPGA
2. Setting up the buffers and partitioning for the network
3. Dynamic reconfiguration management

Architecture Design: Y - Chart

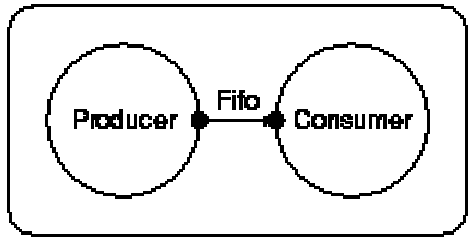
Benchmark driven, based on a Set of Applications



Recommendation: Kahn Process Networks

- Formal semantics, theory and 'proof of concepts' well established
- very well suited for stream oriented high-performance signal processing, including dynamic dataflow.
- abstraction from specific Hardware, Operating System and device drivers
- Extensive prototypes available: UC Berkeley, Caltech, Leiden University, Philips Research
- Compute result is independent of schedule
- C++ stylized industry standard proposed in SystemC 2.0
- Satisfies the goals

Producer-Consumer example (YAPI flavor)




```
#include "producer.h"
#include <iostream>
    Producer::Producer(const Id& n, Out<int>& o) :
        Process(n),
        out( id("out"), o)
    { }
    const char* Producer::type() const
    {
        return "Producer";
    }
    void Producer::main()
    {
        std::cout << "Producer started" << std::endl;

        const int n = 1000;

        write(out, n);

        for (int i=0; i<n; i++)
        {
            write(out, i);
        }
      }
```

```
#include "consumer.h"
#include <assert.h>
#include <iostream>
    Consumer::Consumer(const Id& n, In<int>& i) :
        Process(n),
        in( id("in"), i)
    { }
    const char* Consumer::type() const
    {
        return "Consumer";
    }
    void Consumer::main()
    {
        int n,j;

        std::cout << "Consumer started" << std::endl;

        read(in, n);

        for (int i=0; i<n; i++)
        {
            read(in, j);
            assert(i==j);
        }
    }
```

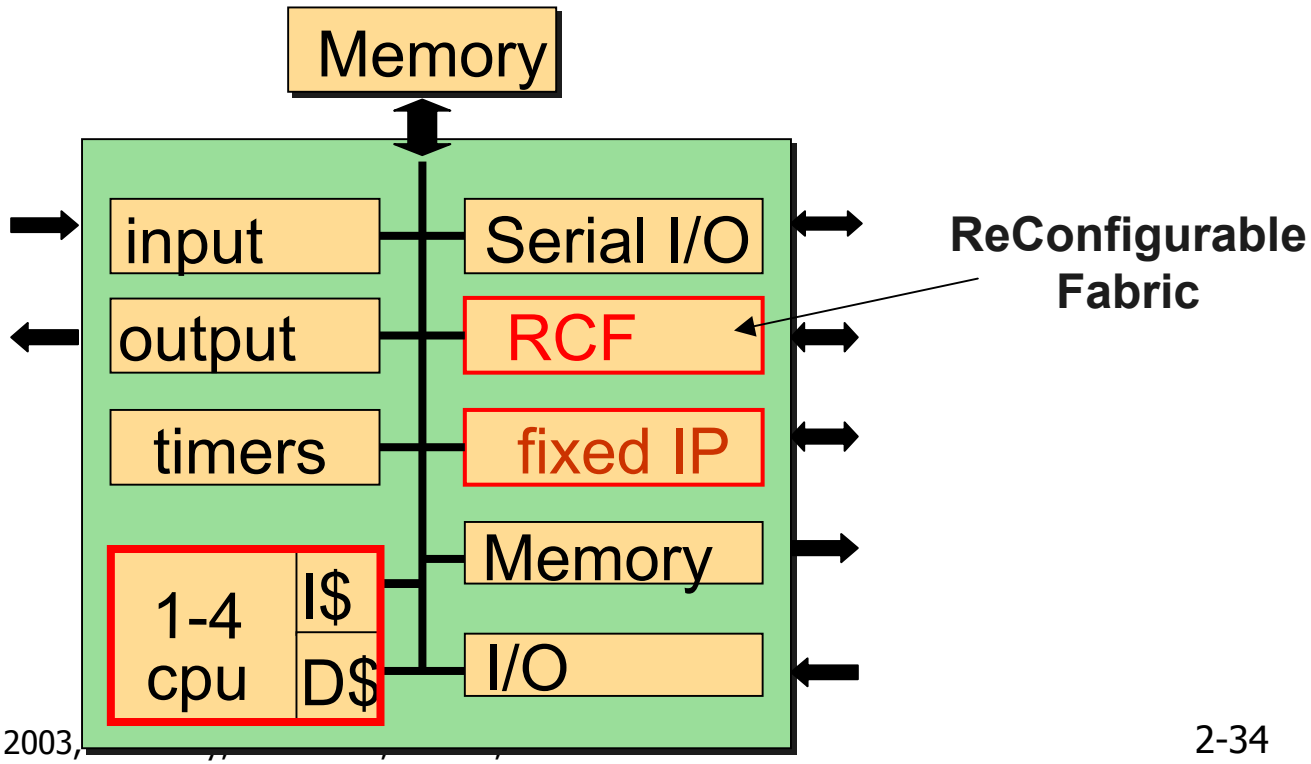
Mapping Kahn Process Networks

Solve 1,2,3

- Input language: Stylized flavors of C++
  - recommendation: SystemC dataflow proposal
- Map one or more processes onto a processor:
  - generate buffer insertion (sometimes statically determined)
  - generate schedule for switching between processes on the same processor (sometimes statically determined)
  - generate overall process for manipulation, preferably running on a general processor
  - synthesize run-time schedule or determine run-time scheduling technique, potentially using thread schedulers or RTOS or OS

New Systems

- Understand the application!
- On chip memory
- Multi processor, programmable and reconfigurable
- Power consumption of the complete IC needs to be constant
- The PROGRAMMERS view is making the difference



Programming Reconfigurable Systems

- The ONLY interesting architectures are the ones you can program/reconfigure.
- Logic synthesis for a specific architecture <> programming (FPGA problem)
- In conventional processors 2/3 of the silicon is in cache subsystems:
  - abstraction for the programmer where the instructions or the data are residing
  - conceptually a shared memory model, even cache coherent multi processors systems
- It took the DSP world more then 10 years to learn: first assembly and MAC, now RISC and VLIW and C compilers

Reconfigurable embedded systems

- SpecInt is irrelevant, EEMBC might be more relevant
- Rethink time-multiplexing:
  - Processor
    - instruction: compiler
    - task: OS
  - Reconfigurable system
    - compute graph that is statically determined
    - reconfigure dynamically: runtime support and buffer management.
- Extracting parallelism:
  - Instruction Level Parallelsim: C is the problem NOT the application
  - Task level: Wrong QUESTION!

Summary

- Reconfigurable computing has many advantages over ASIC and CPU/MPU
  - Large parallelism with no instruction overhead
  - Customizable data path size
  - Flexible (reconfigurable!)
- It is still in its infancy
  - Semantic gap between algorithms and circuits is still a major obstacle
  - Hardware platforms are only now emerging commercially that are designed for RC
  - Mappings are often architecture specific

Trends

- Think Y-chart: FIRST build a mapping environment, then ask the question is this a good architecture.
- Often ad-hoc matching of the model of computation with the model of the architecture -> formalize

- Very exciting time:
  - new tools
  - new architectures
- Reverse the world: silicon is cheap, concurrency and communication is the problem,
- New programming paradigms

Selected References

- J. Henkel, W. Najjir, F. Vahid, K. Vissers,
New Computing Platforms for Embedded Systems, Full day tutorial at 2002, Design Automation Conference.

- Andrew Mihal, Chidamber Kulkarni, Christian Sauer, Kees Vissers, Mathew Moskewicz, Mel Tsai, Niraj Shah, Scott Weber, Yujia Jin,Kurt Keutzer, Sharad Malik, *A Disciplined Approach to the Development of Architectural Platforms,* 2-12, 19, IEEE Design and Test of Computers, 2002

- M. Sima, S. Cotofana, S. Vassiliades, J.T.J. van Eijndhoven, K. Vissers, *MPEG Marcroblock Parsing and Pel Reconstruction on an FPGA-augmented Trimedia Processor*, best paper award International Conference on Computer Design (ICCD), 2001.

- J.T.J. van Eijndhoven, F.W. Sijstermans, K.A. Vissers, E.J.D. Pol, M.J.A. Tromp, P. Struik, R.H.J. Bloks, P. van der Wolf, A.D. Pimentel, H.P.E. Vranken, *Trimedia CPU64 Architecture*, International Conference on Computer Design (ICCD), 1999.

- F. Sijstermans, E.J. Pol, B. Riemens, K.A. Vissers, S. Rathnam, and G. Slavenburg. *Design Space Exploration for Future Trimedia CPUs*, ICASSP '98. 1998.

- B. Kienhuis, E. Deprettere, K.A. Vissers, and P. van der Wolf, *An Approach for Quantitative Analysis of Application-specific Dataflow Architectures.* Proceeding of 11th Int. Conference of Applications-specific Systems Architectures and Processors (ASAP '97), pp. 338-349. 1997.

- K.A. Vissers, G. Essink, P.H.J van Gerwen, P.J.M. Janssen, O. Popp, E. Riddersma, W.J.M Smits, H.J.M. Veendrick. *Architecture and Programming of Two Generations Video Signal Processors*. Journal on Microprocessing and Microprogramming . February, 1996.