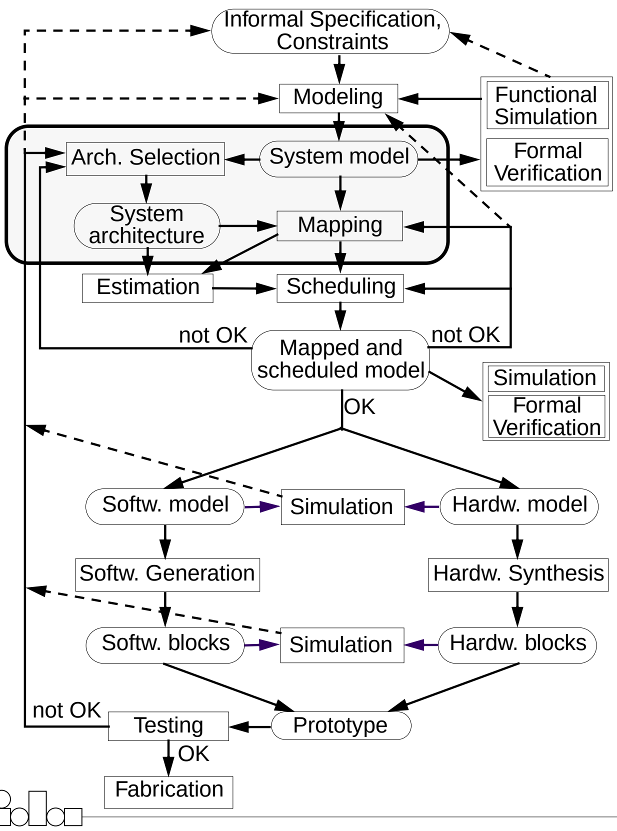
Architectures and Platforms

1. **Architecture Selection: The Basic Trade-Offs**
2. **General Purpose vs. Application-Specific Processors**
3. **Processor Specialisation**
4. **ASIP Design Flow**
5. **Specialisation of a VLIW ASIP**
6. **Tool Support for Processor Specialisation**
7. **Application Specific Platforms**
8. **IP-Based Design (Design Reuse)**
9. **Reconfigurable Systems**

Remember the Design Flow



Architecture Selection and Mapping

- Select the underlying hardware structure on which to run the modelled system.

- Map the functionality captured by the system over the components of the selected architecture.
  Functionality includes processing and communication.

Architecture Selection

General Purpose **vs.** Application Specific
- Use a general purpose, existing platform and map the application on it.
- or something in-between
- Build a customised architecture strictly optimised for the particular application.

Software **vs.** Hardware
- Use programmable processors running software.
- or both
- Use dedicated electronics
  - fixed
  - reconfigurable

Mono **vs.** Multipr.
Single **vs.** Multichip
- Monoprocessor
- Multiprocessor
  - single chip
  - multi chip

Architecture Selection (cont'd)

The trade-offs:

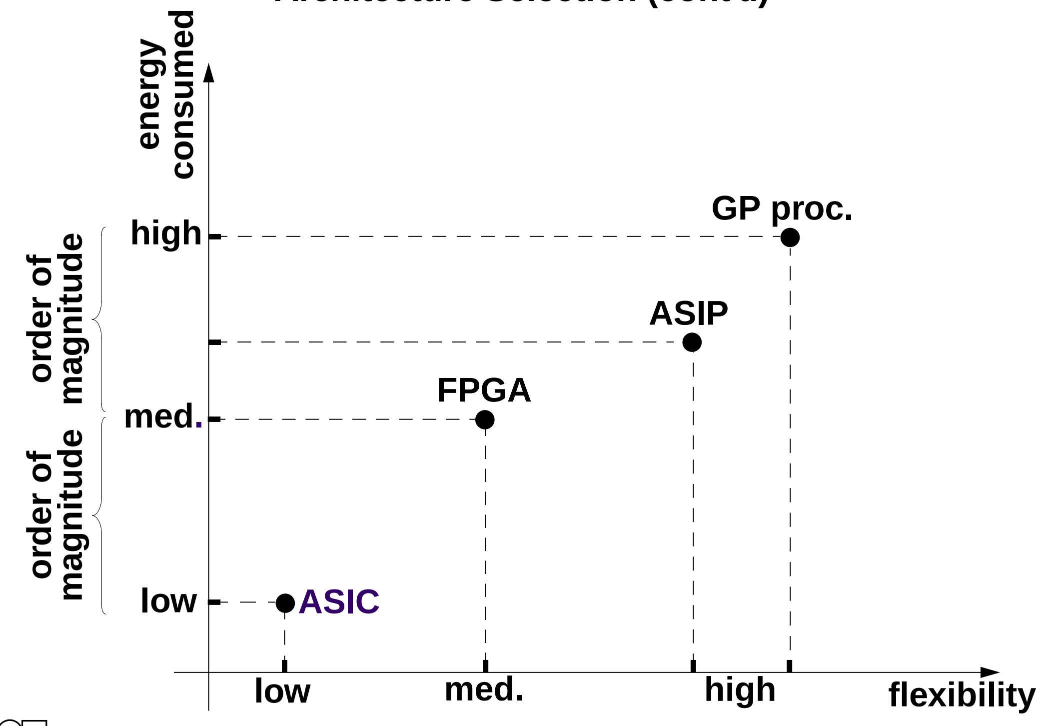
- Performance (high speed, low power consumption)

Application specific	high	Hardware	high
General purpose	low	Reconfigurable hardware	
		Software	low

- Flexibility (how easy it is to upgrade or modify)

General purpose	high	Software	high
Application specific	low	Reconfigurable hardware	
		Hardware	low

Architecture Selection (cont'd)



General Purpose vs. Application Specific Processors

☞ Both GP processors and ASIPs (application specific instruction set processors) can be RISCs, CISCs, DSPs, microcontrollers, etc.

- One could look at DSPs and microcontrollers as being specific for DSP and simple control applications respectively.
- An application specific DSP or microcontroller is, however, more specialised then *just* for DSP or control applications.

- GP processors
  - Neither instruction set nor microarchitecture or memory system are customised for a particular application or family of applications
- ASIPs
  - Instruction set, microarchitecture and/or memory system are customised for an application or family of applications.
  - What results is better performance and reduced power consumption.

What Makes an ASIP “Specific”?

What can we specialize in a processor?

☞ Instruction set (IS) specialisation

- Exclude instructions which are not used
  - reduces instruction word length (fewer bits needed for encoding);
  - keeps controller and data path simple.

- Introduce instructions, even “exotic” ones, which are specific to the application: combinations of arithmetic instructions (multiply-accumulate), small algorithms (encoding/decoding, filter), vector operations, string manipulation or string matching, pixel operations, etc.
  - reduces code size ⇒ reduced memory size, memory bandwidth, power consumption, execution time.

What Makes an ASIP “Specific”?

☞ Function unit and data path specialisation

Once an application specific IS is defined, this IS can be implemented using a more or less specific data path and more or less specific function units.

- Adaptation of word length.
- Adaptation of register number.
- Adaptation of functional units
  - Highly specialised functional units can be introduced for string matching and manipulation, pixel operation, arithmetics, and even complex units to perform certain sequences of computations (co-processors).

What Makes an ASIP “Specific”?

☞ Memory specialisation

- Number and size of memory banks.
- Number and size of access ports.
  - They both influence the degree of parallelism in memory access.
  - Having several smaller memory blocks (instead of one big) increases parallelism and speed, and reduces power consumption.
  - Sophisticated memory structures can increase cost and bandwidth requirement.
- Cache configuration:
  - separate instruction/data?
  - associativity
  - cache size
  - line size

Depends very much on the characteristics of the application and, in particular, on the properties related to locality.

Very large impact on performance and power consumption.

What Makes an ASIP “Specific”?

☞ <u>Interconnect specialization</u>

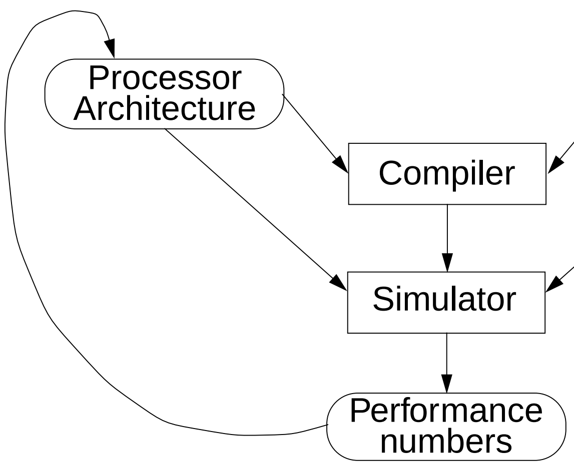
- Interconnect of functional modules and registers.
- Interconnect to memory and cache.
  - How many internal buses?
  - What kind of protocol?
  - Additional connections increase the potential of parallelism.

☞ <u>Control specialisation</u>

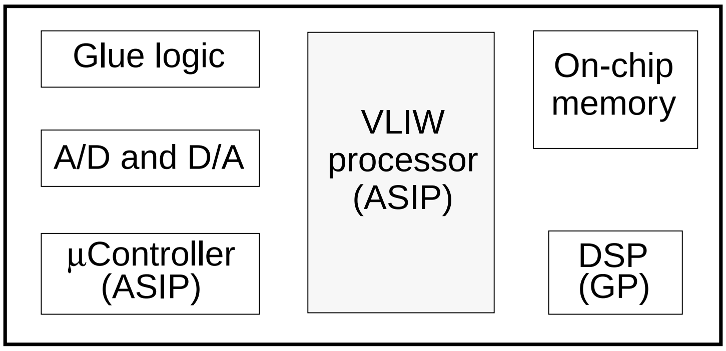
- Centralised control or distributed (globally asynchronous)?
- Pipelining?
- Out of order execution?
- Hardwired or microprogrammed?

ASIP Design Flow

(It can be seen as a part of the “big” design flow - slide 2)



A SOC for Multimedia Applications

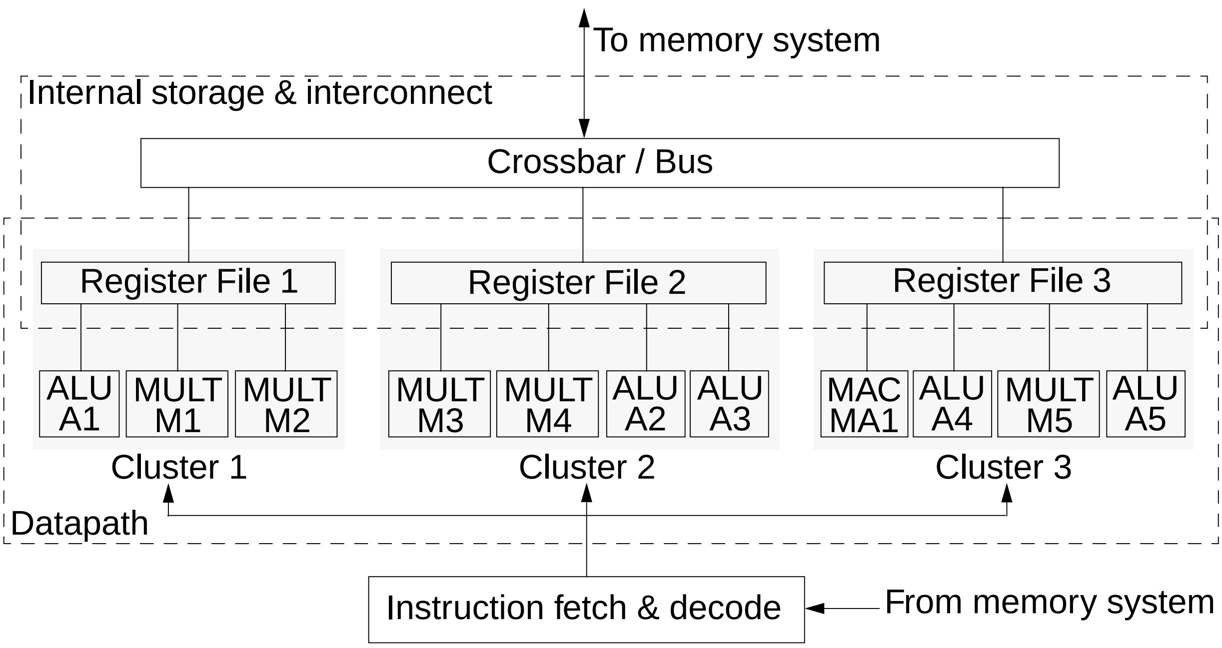


☞ This is a typical application specific *platform*. Its structure has been adapted for a family of applications.

☞ Besides GP processor cores, the platform also consists of ASIP cores which themselves are specialised.

- The application specific µController performs master control of the system and memory access control.
- The off-the-shelf (GP) DSP performs less computation intensive modem and sound codec functions.
- The VLIW ASIP performs computation intensive functions: discrete cosine and inverse discrete cosine transforms, motion estimation, etc.

Specialization of a VLIW ASIP



Specialization of a VLIW ASIP (cont’d)

That’s how an instruction word looks like:

op_1	op_2	op_3	op_4	op_5	op_6	op_7	op_8	op_9	op_{10}	op_{11}
Cluster 1			Cluster 2				Cluster 3			

Specialization of a VLIW ASIP (cont’d)

☞ Traditionally the datapath is organised as single register file shared by all functional units.

Problem: Such a centralised structure does not scale!

We increase the nr. of functional units in order to increase parallelism

$\Downarrow$

We have to increase the number of registers in the register file

$\Downarrow$

Internal storage and communication between functional units and registers becomes dominant in terms of area, delay, and power.

☞ High performance VLIW processors are limited not by arithmetic capacity but by internal bandwidth.

Specialization of a VLIW ASIP (cont'd)

A solution: *clustering*.

- Restrict the connectivity between functional units and registers, so that each functional unit can read/write from/to a subset of registers.

  ⇩

  Organise the datapath as clusters of functional units and local register files.

☞ Nothing is for free!!!
Moving data between registers belonging to different clusters takes much time and power!
You have to drastically minimise the number of such moves by:
- Carefully adapting the structure of clusters to the application.
- Using very clever compilers.

Specialization of a VLIW ASIP (cont'd)

- Instruction set specialisation: nothing special.

- Function unit and data path specialisation
  - Determine the number of clusters.
  - For each cluster determine
    - the number and type of functional units;
    - the dimension of the register file.

- Memory specialisation is extremely important because we need to stream large amounts of data to the clusters at high rate; one has to adapt the memory structure to the access characteristics of the application.
  - determine the number and size of memory banks

Specialization of a VLIW ASIP (cont'd)

- Interconnect specialization

  - Determine the interconnect structure between clusters and from clusters to memory:
    - one or several buses,
    - crossbar interconnection
    - etc.

- Control specialisation:

  That's more or less done, as we have decided for a VLIW processor.

Tool Support for Processor Specialisation

Look at the design flow on slide 12!

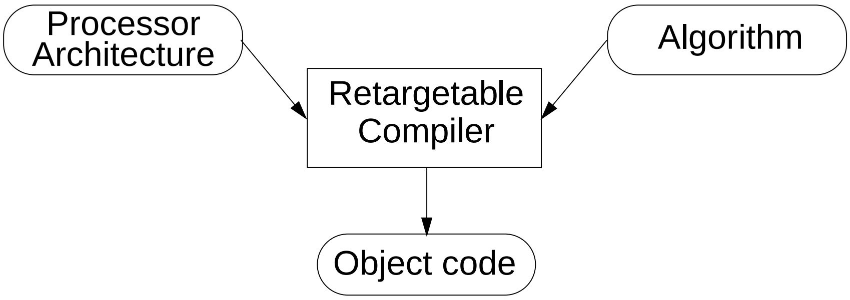
In order to be able to generate a specialised architecture you need:

- Retargetable compiler

- Configurable simulator

Retargetable Compiler

Retargetable compiler



Retargetable Compiler (cont'd)

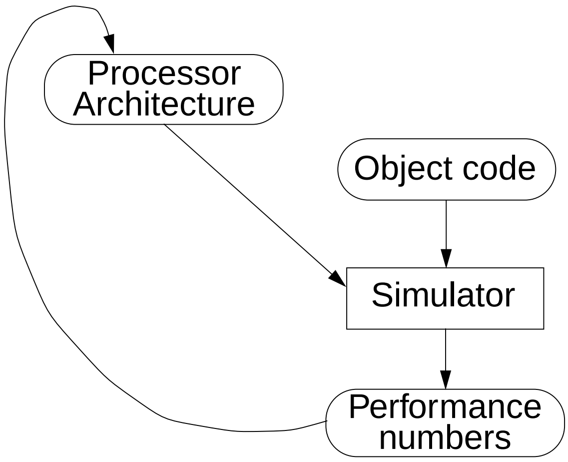
- An *automatically retargetable compiler* can be used for a range of different target architectures.

  The actual code optimization and code generation is done by the compiler, based on a description of the target processor architecture. This description is formulated in a, so called, "architecture description language".

- Having a good compiler is not only important for the processor specialisation process!

  Once you have got your specialised ASIP you need a good compiler in order to efficiently make use of it!

Configurable Simulator



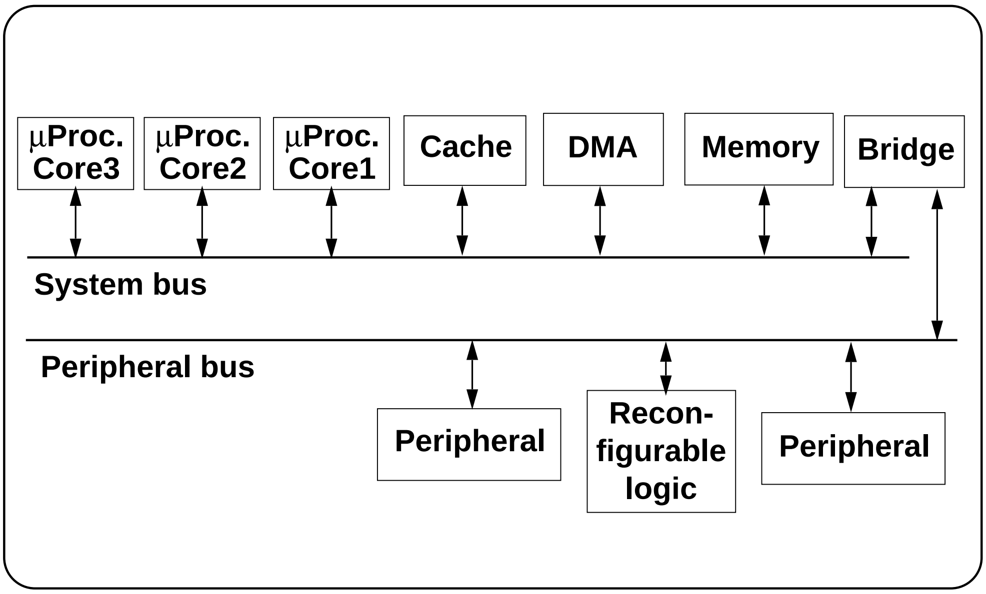
- Such a simulator can be configured for a particular architecture (based on an architecture description)
- In this context, the most important output produced by the simulator is performance numbers:
  - throughput
  - delay
  - power/energy consumption

Application Specific Platforms

☞ Not only processors but also *hardware platforms* can be specialised for classes of applications.

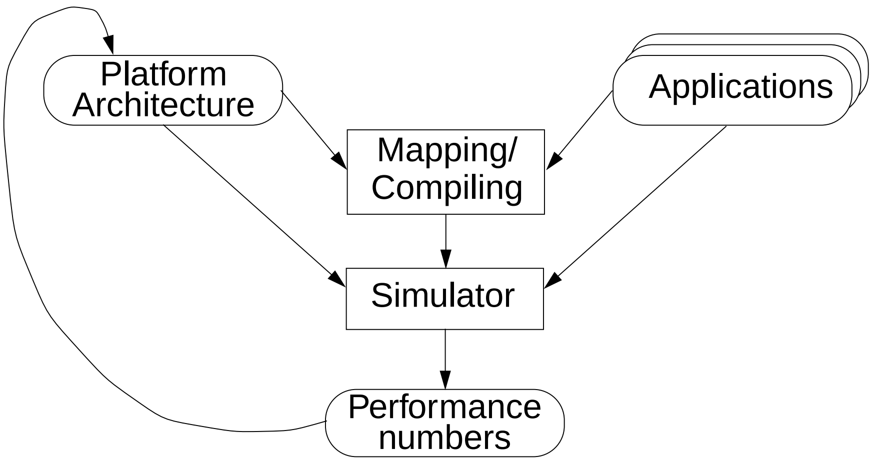
The platform will define a certain communication infrastructure (buses and protocols), certain processor cores, peripherals, accelerators commonly used in the particular application area, and basic memory structure.

Application Specific Platforms (cont'd)



Application Specific Platforms (cont'd)

Design space exploration for platform definition:

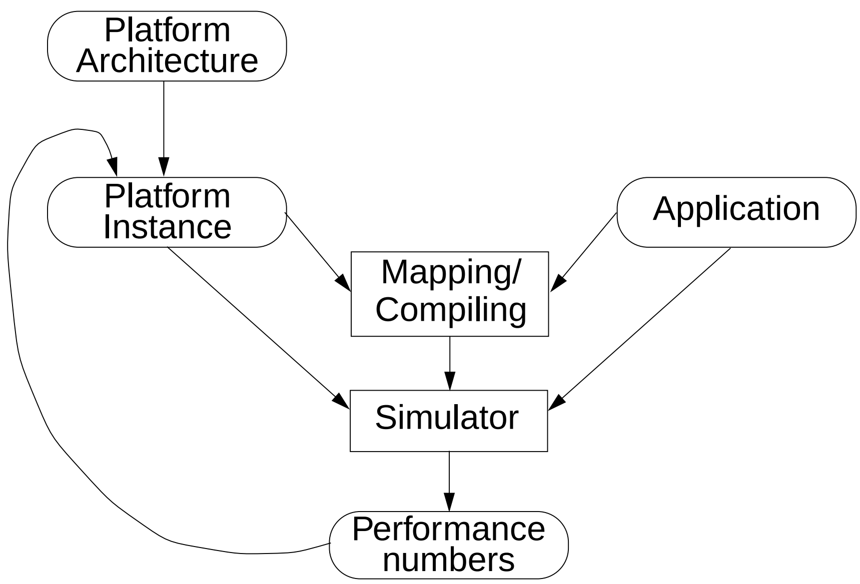


Instantiating a Platform

☞ Once we have an application, the chip to implement on will not be designed as a collection of independently developed blocks, but will be *an instance of an application specific platform*.

- The hardware platform will be refined by
  - determining memory and cache size
  - identifying the particular cores, peripherals to be used
  - adding specific ASICs, accelerators
  - determining the amount of reconfigurable logic (if needed)

Instantiating a Platform (cont'd)



System Platforms

☞ What we discussed about (see previous slides) are so called hardware platforms.

☞ The hardware platform is delivered together with a software layer: hardware platform + software layer = *system platform*.

- Software layer:
  - real-time operating system
  - device drivers
  - network protocol stack
  - compilers
- The software layer creates an abstraction of the hardware platform (an application program interface) to be seen by the application programs.

IP-Based Design (Design Reuse)

☞ The key concept in order to increase designers' productivity is *reuse*.

In order to manage the complexity of current large designs we do not start from scratch but reuse as much as possible from previous designs, or use commercially available pre-designed *IP blocks*.

IP: intellectual property.

☞ Some people call this *IP-based design*, *core-based design*, reuse techniques, etc.:

Core-based design is the process of composing a new system design by reusing existing components.

IP-Based Design (cont'd)

What are the blocks (cores) we reuse?

- interfaces, encoders/decoders, filters, memories, timers, microcontroller-cores, DSP-cores, RISC-cores, GP processor-cores.

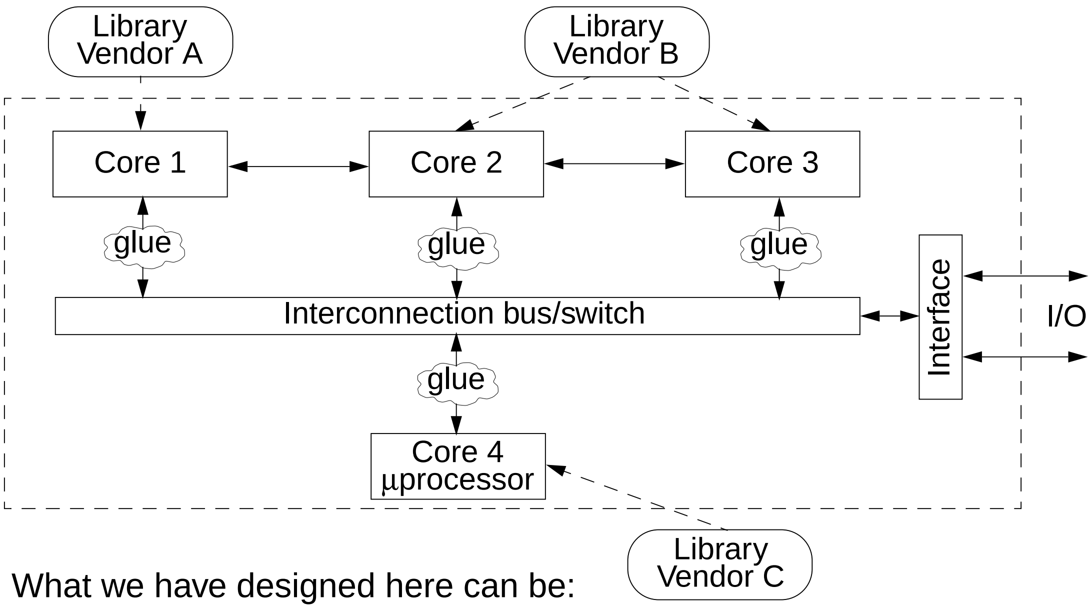
Possible(!) definition

- A *core* is a design block which is larger than a typical RTL component.

Of course:

We also reuse *software components*!

IP-Based Design (cont'd)



What we have designed here can be:

- An application specific SOC
- A platform to be further instantiated for a particular application.

Types of Cores

- Hard cores: are fully designed, placed, and routed by the supplier.

A completely validated layout with definite timing

rapid integration

low flexibility

- Firm cores: technology-mapped gate-level netlists.

less predictability

flexibility during place and route

Types of Cores (cont'd)

- Soft cores: synthesizable RTL or behavioral descriptions.

much work with integration and verification.

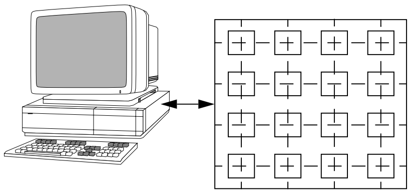
maximal flexibility

Flexibility can provide opportunities like e.g. adding application specific instructions to a processor core by modifying the behavioral description.

Reconfigurable Systems

☞ Programmable Hardware Circuits:

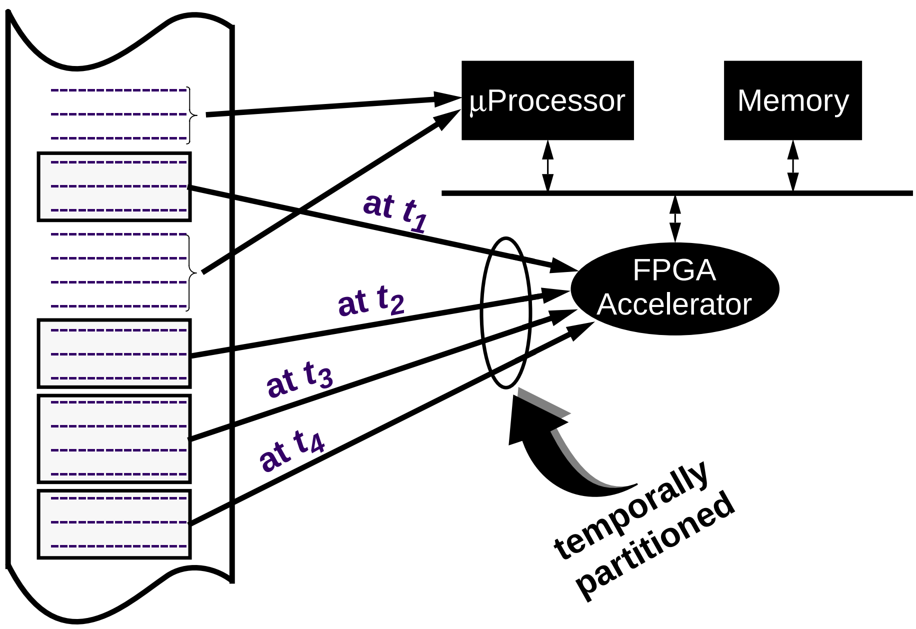
- They implement arbitrary combinational or sequential circuits and can be configured by loading a local memory that determines the interconnection among logic blocks.
- Reconfiguration can be applied an unlimited number of times.



☞ Main applications:
- Software acceleration
- Prototyping

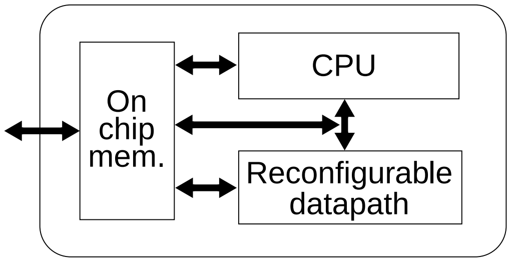
Reconfigurable Systems (cont'd)

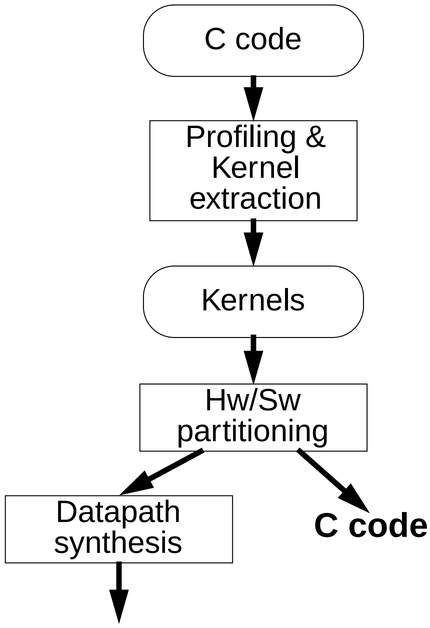
Dynamic reconfiguration: spacial and temporal partitioning



Reconfigurable Systems (cont'd)

System on Chip with dynamically reconfigurable datapath





Summary

- Architecture selection is about making trade-offs along the dimensions of speed, cost, flexibility, and power consumption.
- ASIPs are programmable processors, specialised for a particular application or for a family of applications.
- Specialisation of an ASIP concerns instruction set, function units and data path, memory system, interconnect, and control.
- Two design tools are of great importance in order to perform processor specialisation: retargetable compiler and configurable simulator.
- Not only processors can be specialised but also platforms. A Platform is specialised to execute a certain family of applications. The particular hardware to be used for a given application is a specialised instantiation of the platform.

Summary (cont'd)

- Reuse is a key technique in order to achieve high design productivity. Cores to be reused can be from interfaces and decoders to filters and processors.
- The three types of cores differ in their flexibility, predictability, and the effort needed for integration: hard, firm, and soft cores.
- Reconfigurable systems can provide good flexibility and, at the same time, many of the advantages of classical hardware implementation. They are mainly used for software acceleration and prototyping.