Embedded Systems
Lecture 15: HW & SW Optimisations

Björn Franke
University of Edinburgh

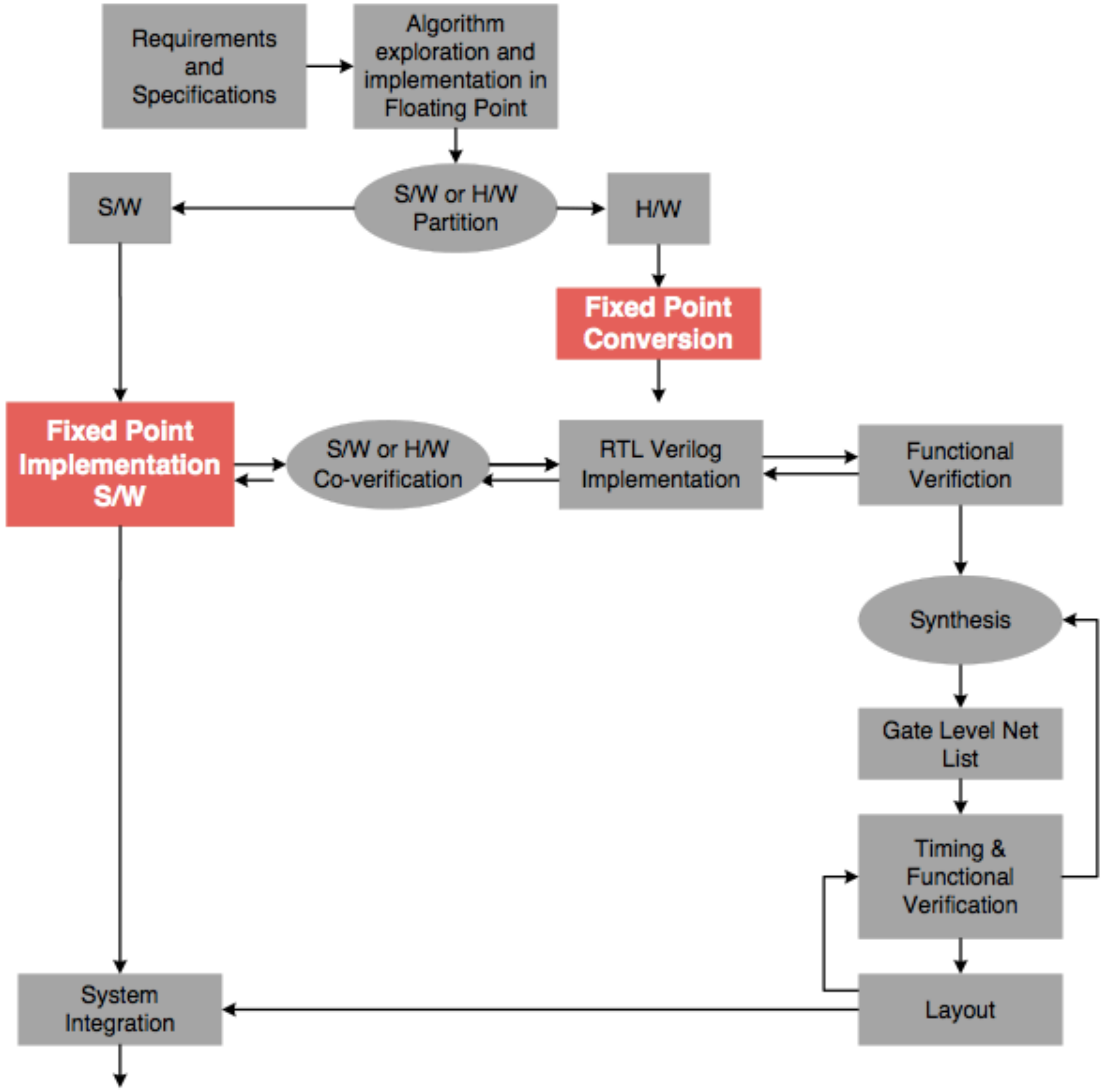
Overview

- SW Optimisations
  - Floating-Point to Fixed-Point Conversion
- HW Optimisations
  - Application-Specific Instruction Set Processors (ASIPs)
  - Application-Specific Hardware Platforms
  - IP-Based Design
  - Reconfigurable Systems

Floating & Fixed Point Arithmetic

- **Floating Point Arithmetic**
  - After each arithmetic operation numbers are normalised
  - Used where precision and dynamic range are important
  - Most algorithms are developed in FP
    - Ease of coding
  - More Cost (Area, Speed, Power)
- **Fixed Point Arithmetic**
  - Place of decimal is fixed
  - Simpler HW, low power, less silicon
  - Converting FP simulation to Fixed point simulation is time consuming
  - Multiplication doubles the number of bits: NxN multiplier produces 2N bits
  - The code is less readable, need to worry about overflow and scaling issues

System-level Design Flow and Fixed-point Arithmetic


Requirements and Specifications
Algorithm exploration and implementation in Floating Point
S/W
S/W or H/W Partition
H/W
Fixed Point Conversion
Fixed Point Implementation S/W
S/W or H/W Co-verification
RTL Verilog Implementation
Functional Verifiction
Synthesis
Gate Level Net List
Timing & Functional Verification
System Integration
Layout

Fixed-point vs Floating-point Hardware

- Algorithms are developed in floating point format using tools like Matlab

- Floating point processors and HW are expensive

- Fixed-point processors and HW are often used in embedded systems

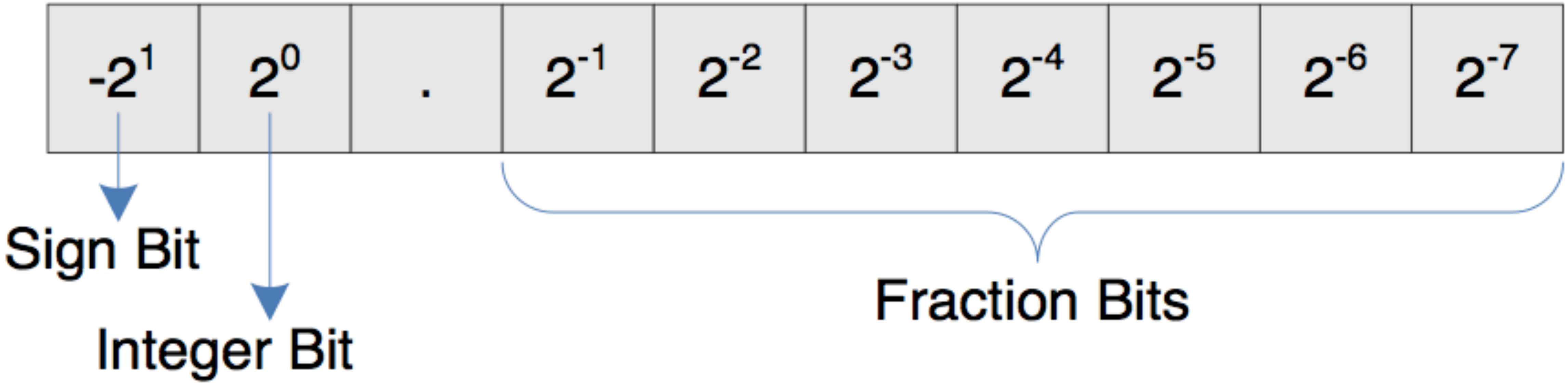
- After algorithms are designed and tested then they are converted into fixed-point implementation

- The algorithms are ported on Fixed-point processor or application specific hardware

Qn.m Format for Fixed-point Arithmetic

- Qn.m format is a fixed positional number system for representing fixed-point numbers

- A Qn.m format N-bit binary number assumes n bits to the left and m bits to the right of the binary point

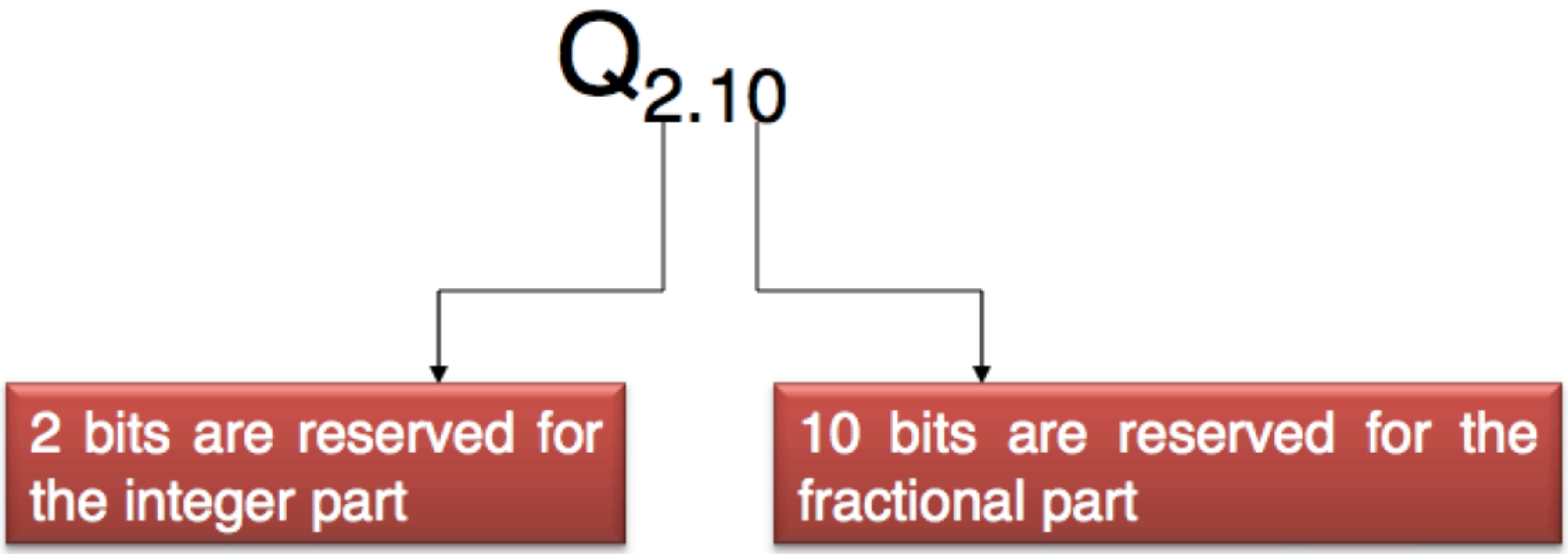


Qn.m Key Idea

In $Q_{n.m}$ format,

n entirely depends upon the range of integer

m defines the precision of the fractional part



$Q_{2.10}$

2 bits are reserved for the integer part

10 bits are reserved for the fractional part

Qn.m Positive Numbers

- The MSB is the sign bit
- For a positive fixed-point number, MSB is 0

$$b = 0b_{n-2}\ldots b_1 b_0 . b_{-1} b_{-2}\ldots b_{-m}$$

- Equivalent floating point value of the positive number is

$$b = b_{n-2}2^{n-2} + \cdots\cdots + b_1 2^1 + b_0 + b_{-1}2^{-1} + b_{-2}2^{-2} + \cdots\cdots + b_{-m}2^{-m}$$

- For Negative numbers, MSB has negative weight and equivalent value is

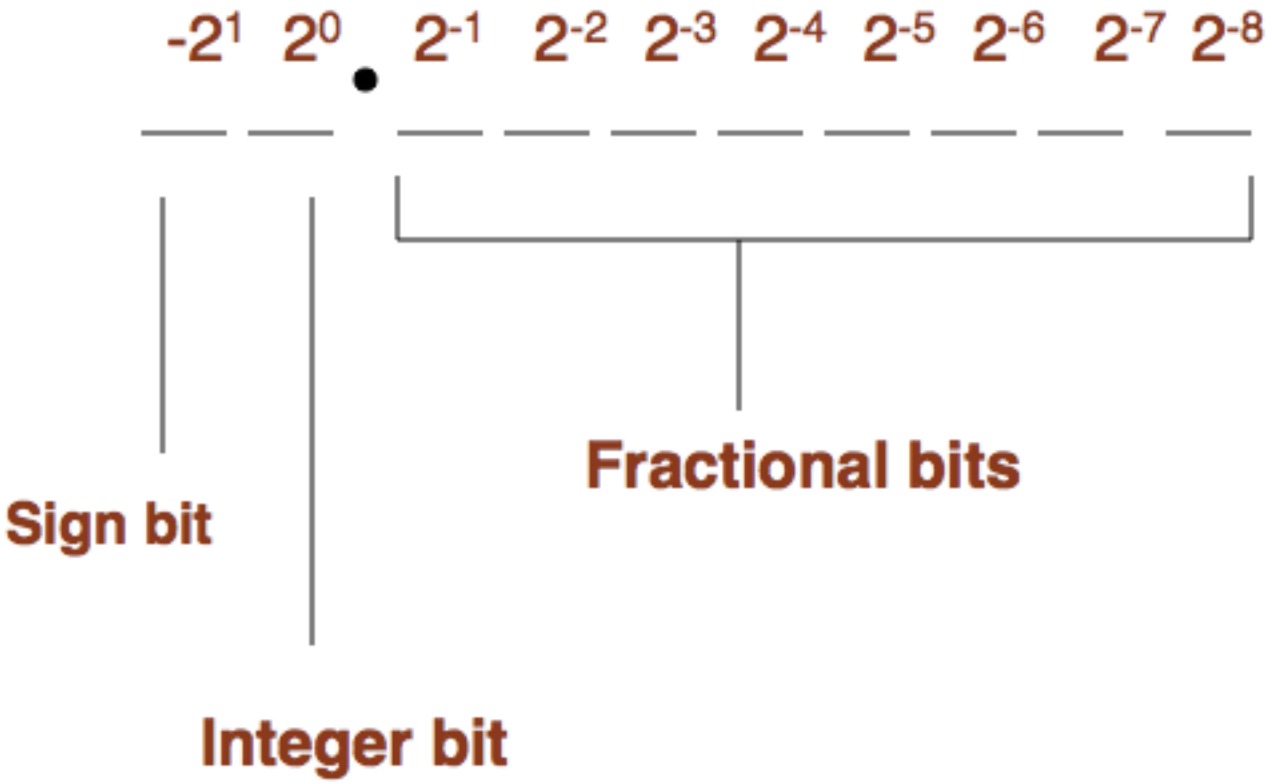
$$b = -b_{n-1}2^{n\ 1} + b_{n-2}2^{n\ 2} + \cdots\cdots + b_1 2^1 + b_0 + b_{-1}2^{\ 1} + b_{-2}2^{\ 2} + \cdots\cdots + b_{-m}2^{\ m}$$

Conversion to Qn.m

1. Define the total number. of bits to represent a Qn.m number

___ ___ ___ ___ ___ ___ ___ ___ ___ ___ (assume 10 bits)

2. Fix location of decimal based on the value of the number



-2^1 2^0 . 2^{-1} 2^{-2} 2^{-3} 2^{-4} 2^{-5} 2^{-6} 2^{-7} 2^{-8}

Sign bit

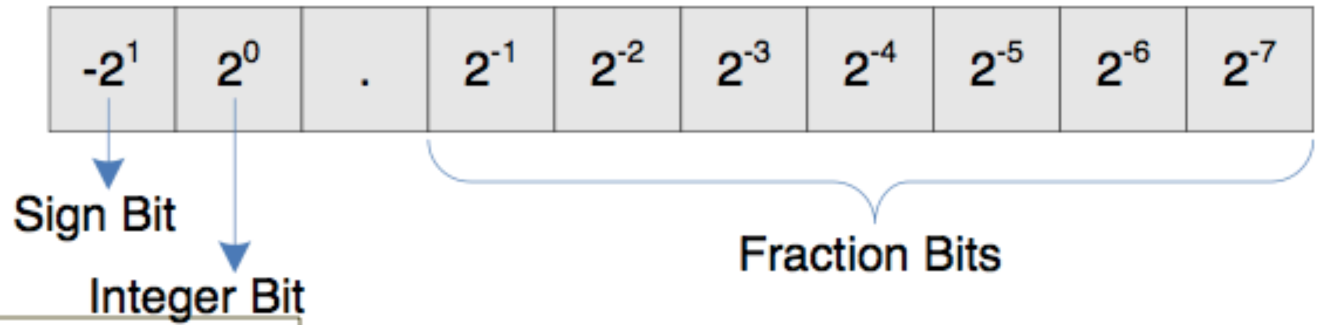
Integer bit

Fractional bits

(assume 2 bits for the integer part)

The decimal point is implied

Example



-2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}

Sign Bit

Integer Bit

Fraction Bits

01 1101 0000

In Q2.8 format, the number is :

$1 + 1/2 + 1/2^2 + 1/2^4 = 1.8125$

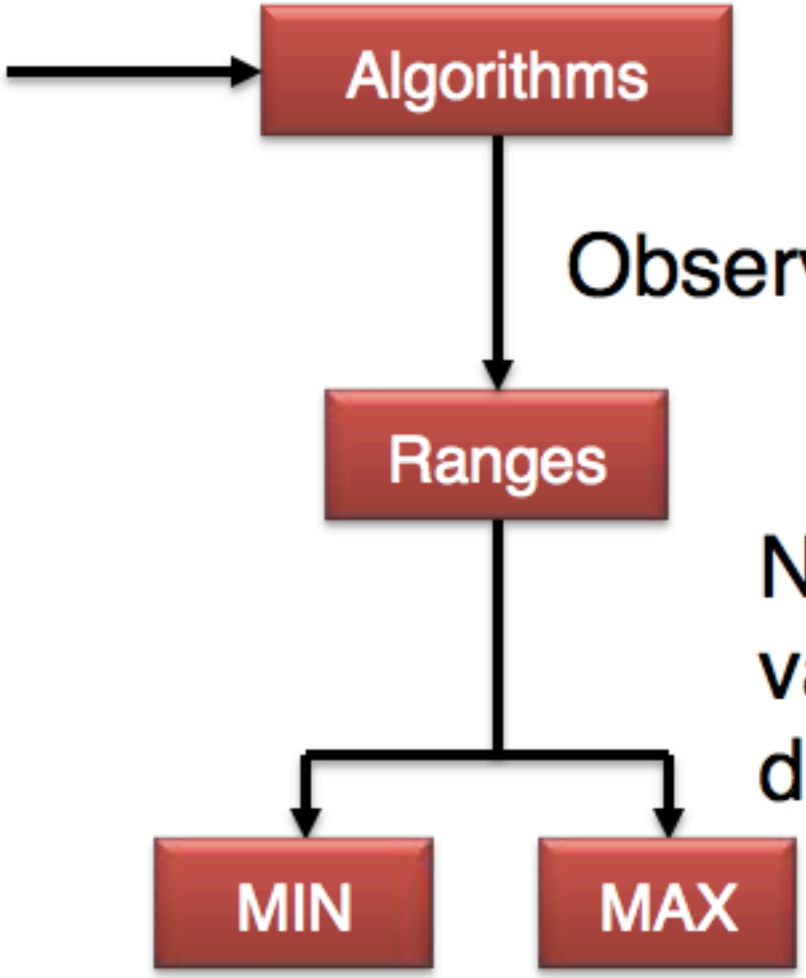
Equivalent fixed-point integer value is

0x1D0

- 2 bits for the integer and remaining 8 bits keeps the fraction part
- A 10 bit Q2.8 signed number covers -2 to +1. 9922.
- Increasing the fractional bits increases the precision.

Range Determination for Qn.m Format

Run simulations for all set of inputs



Algorithms

Observe ranges of values for all variable

Ranges

Note Min and Max values each variable takes for Qn.m format determination

MIN

MAX

Application-Specific Instruction Set Processor ASIP

- Designed for a fixed application (domain), e.g. 4G baseband processing
- Designed to accelerate heavy and most used functions
- Designed to implement the instruction set with minimum hardware cost
- Biggest challenges: silicon cost and power consumption
- Goals of ASIP design: highest performance over silicon, over power consumption, as well over the design cost
- Involves: ASIP design flow, source code profiling, architecture exploration, assembly instruction set design, design of assembly language programming tool chain, firmware design, benchmarking, and micro architecture design

What Makes an ASIP “Specific”?

- What can we specialise 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).

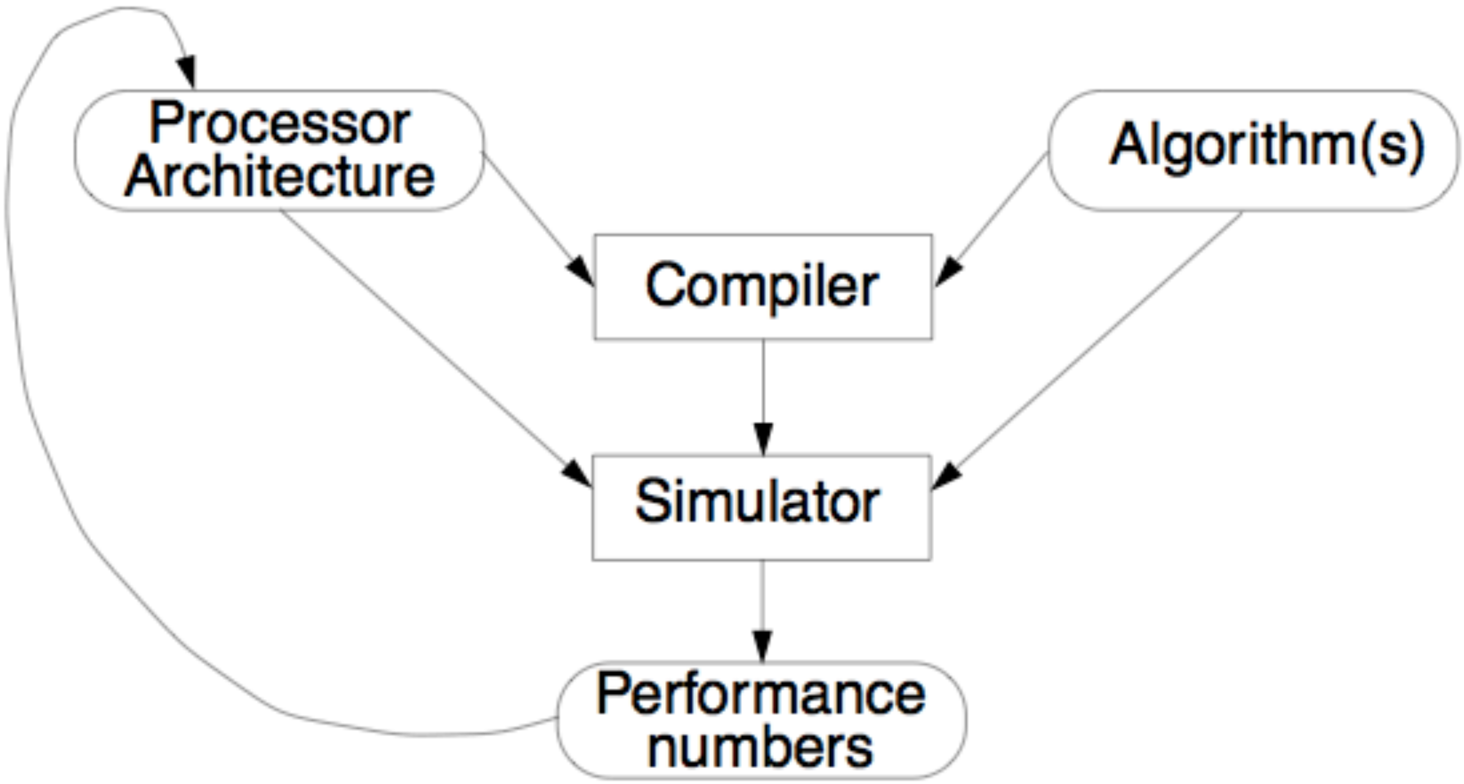
What Makes an ASIP “Specific”?

- **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.

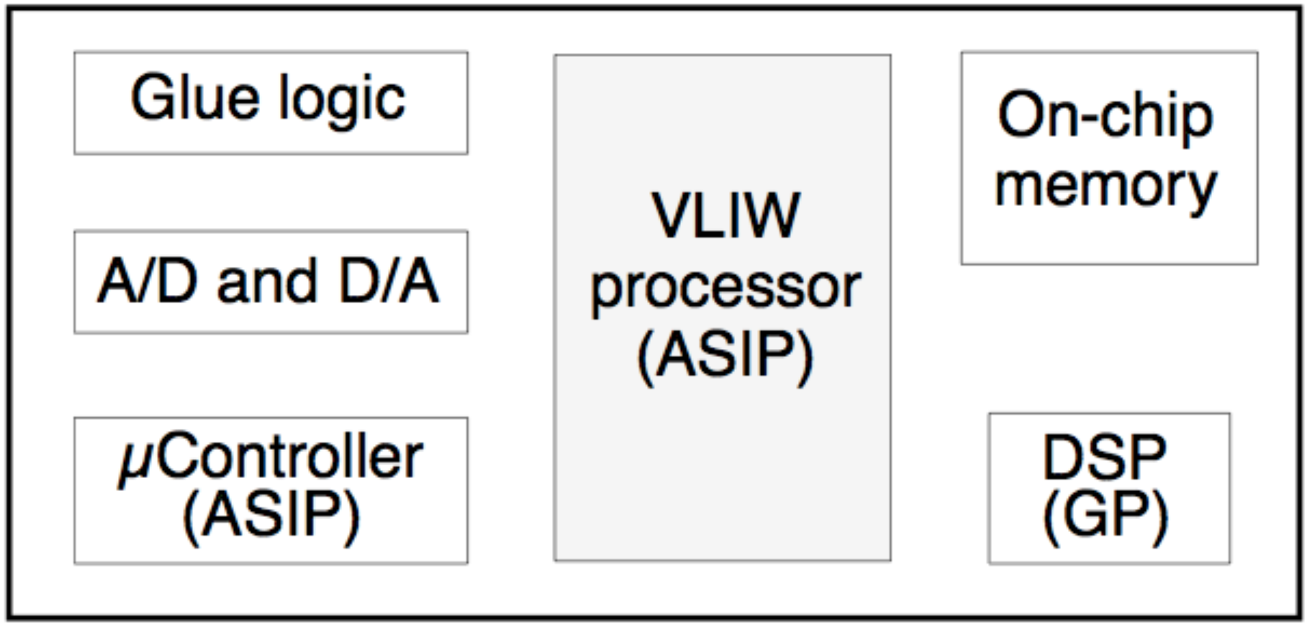
What Makes an ASIP “Specific”?

- **Interconnect specialisation**
  - 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.
- **Control specialisation**
  - Centralised control or distributed (globally asynchronous)?
  - Pipelining?
  - Out of order execution?
  - Hardwired or microprogrammed

ASIP Design Flow



Case Study: A SoC for Multimedia Applications

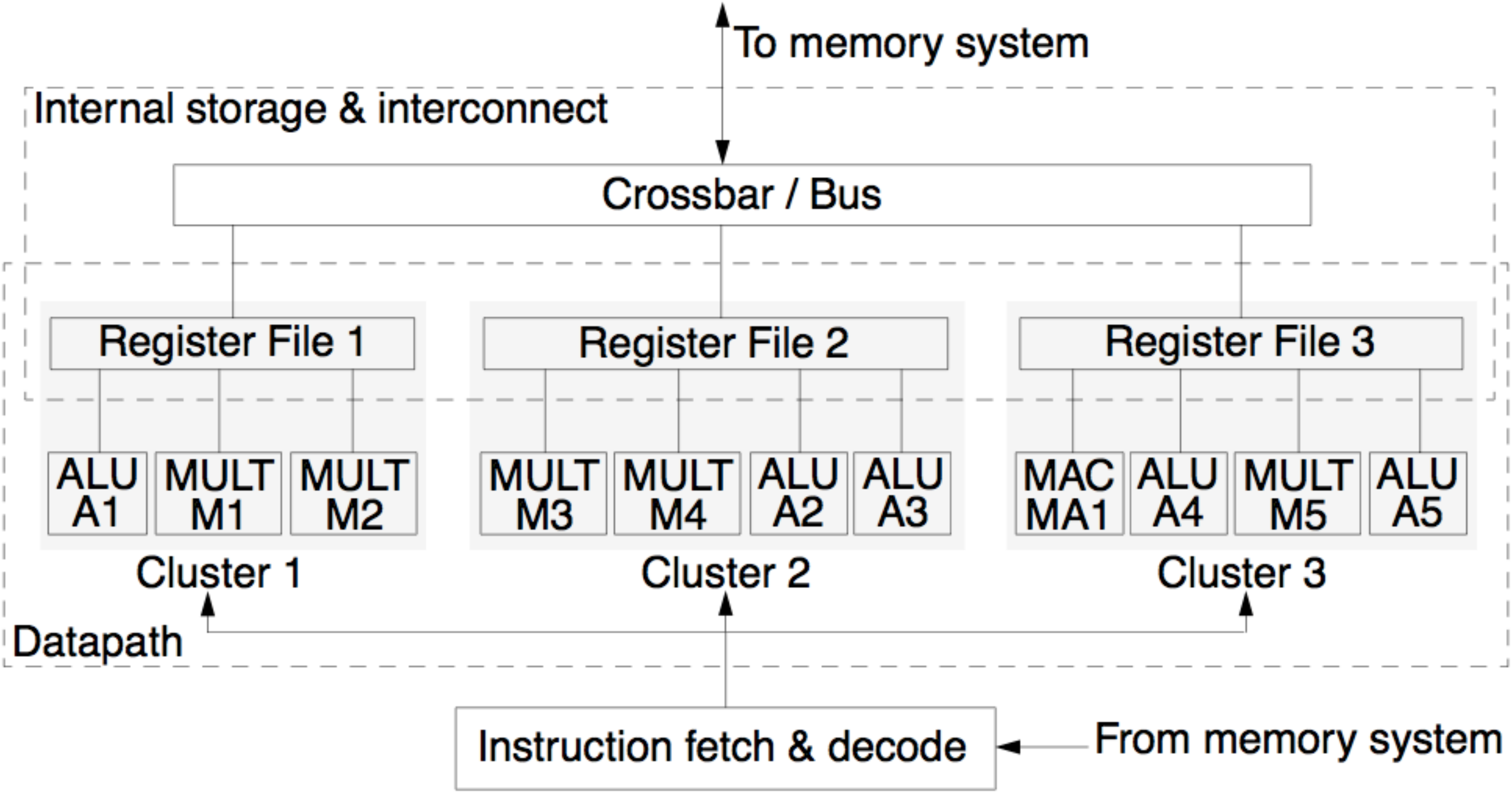


This is a typical application specific *platform*. Its structure has been adapted for a family of applications.

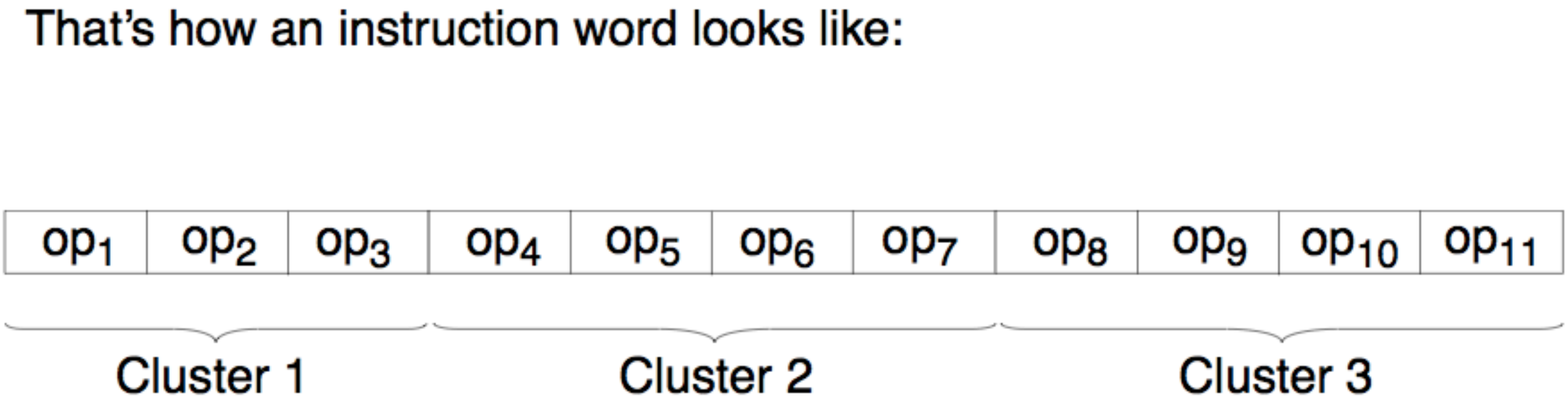
Besides GP processor cores, the platform also consists of ASIP cores which themselves are specialised.

- 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.

Specialisation of a VLIW ASIP



Specialisation of a VLIW ASIP (cont'd)



That's how an instruction word looks like:

Specialisation 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 no. of functional units in order to increase parallelism
  - We have to increase the number of registers in the register file, too
  - ☞ 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

Specialisation 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.

Specialisation of a VLIW ASIP (cont’d)

- **Instruction set specialisation**: nothing special. Maybe DSP instructions.
- **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

Specialisation of a VLIW ASIP (cont’d)

- **Interconnect specialisation**
    - 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.
    - Maybe Zero-Overhead Loops...

Tool Support for Processor Specialisation

- Design Tools for ASIP Design Flow
- In order to be able to generate a specialised architecture you need:
  - Architecture Design Tools (e.g. Synopsys Processor Designer)
  - Retargetable compiler
  - Configurable simulator

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.

- Think of TI OMAP, Exynos, ... for smartphones

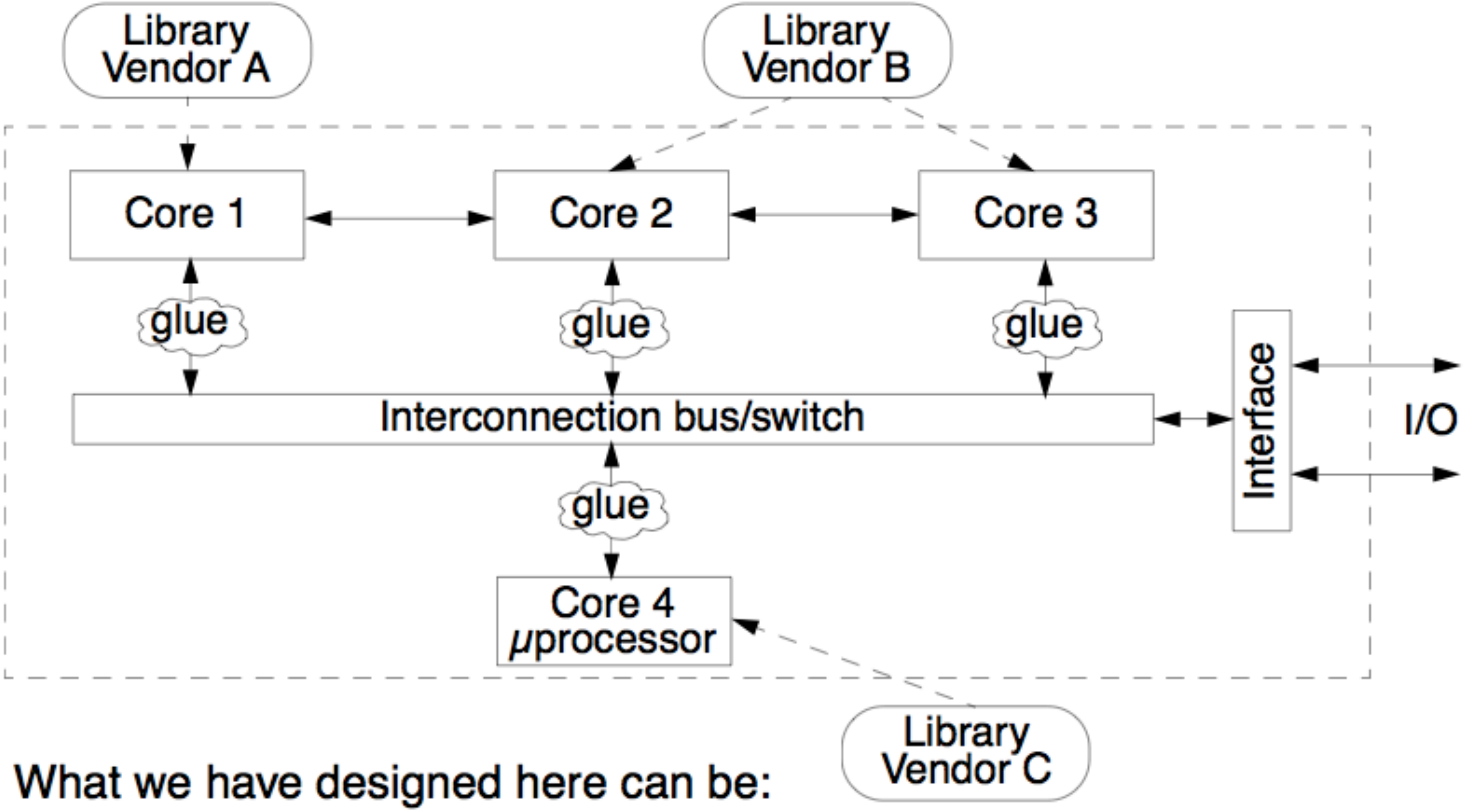
IP-Based Design (Design Reuse)

- 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
  - e.g. ARM cores, Imagination Technologies GPUs, ...
- 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)

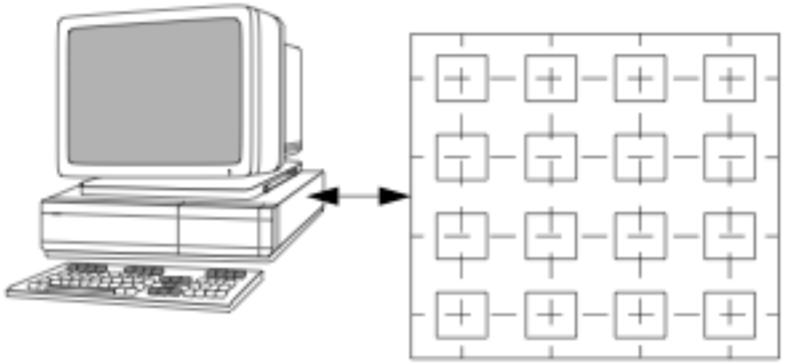


What we have designed here can be:

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

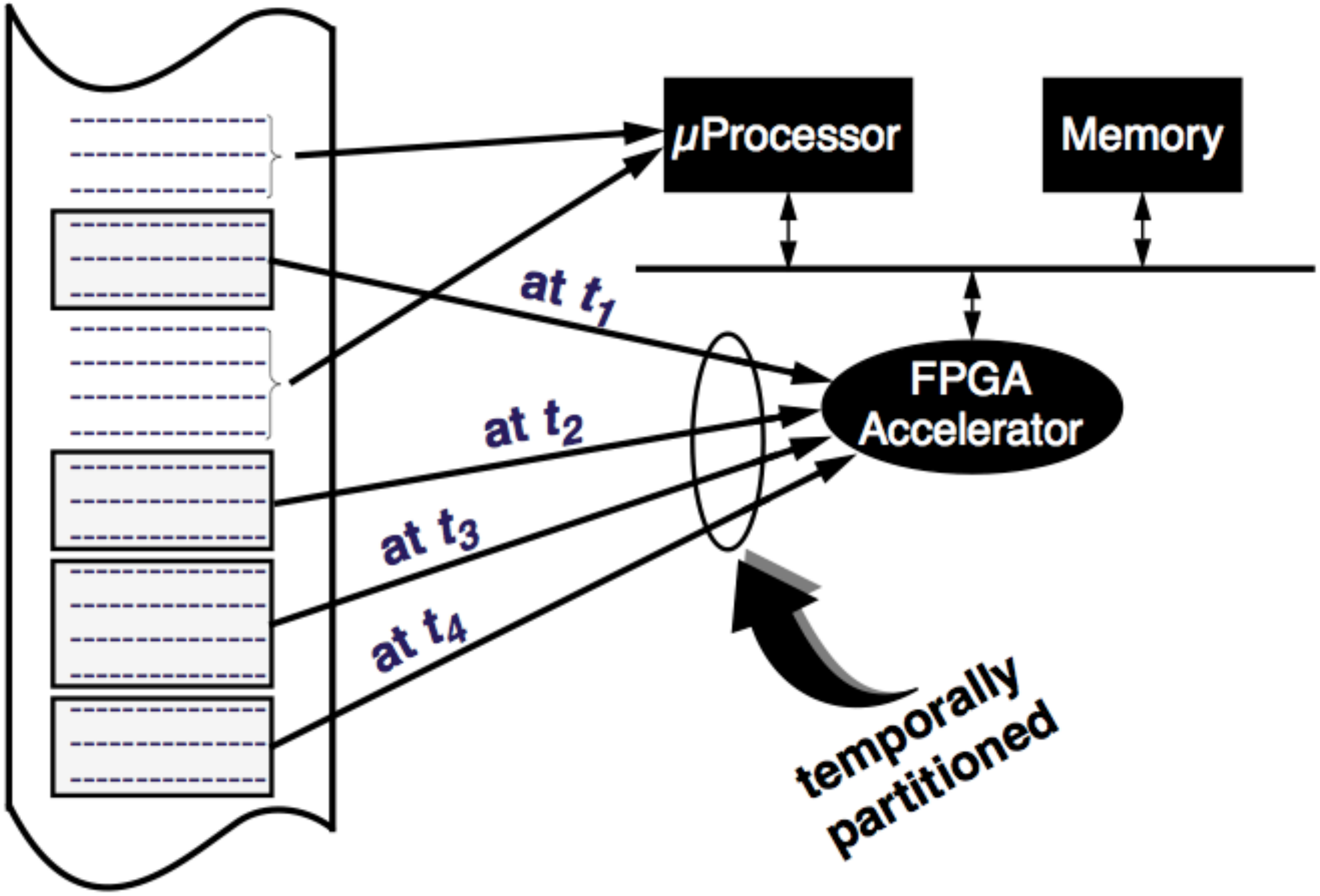
Reconfigurable Systems

- 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

Dynamic reconfiguration: spacial and temporal partitioning



Summary

- SW Optimisations
  - Floating-Point to Fixed-Point Conversion
- HW Optimisations
  - Application-Specific Instruction Set Processors (ASIPs)
  - Application-Specific Hardware Platforms
  - IP-Based Design
  - Reconfigurable Systems

Preview

- Dynamic Frequency Scaling
- Dynamic Voltage Scaling