ECE4810J System-on-Chip Design

Topic 2

SoC System Approach II

Xinfei Guo
xinfei.guo@sjtu.edu.cn

September 30th, 2024

T2 learning goals

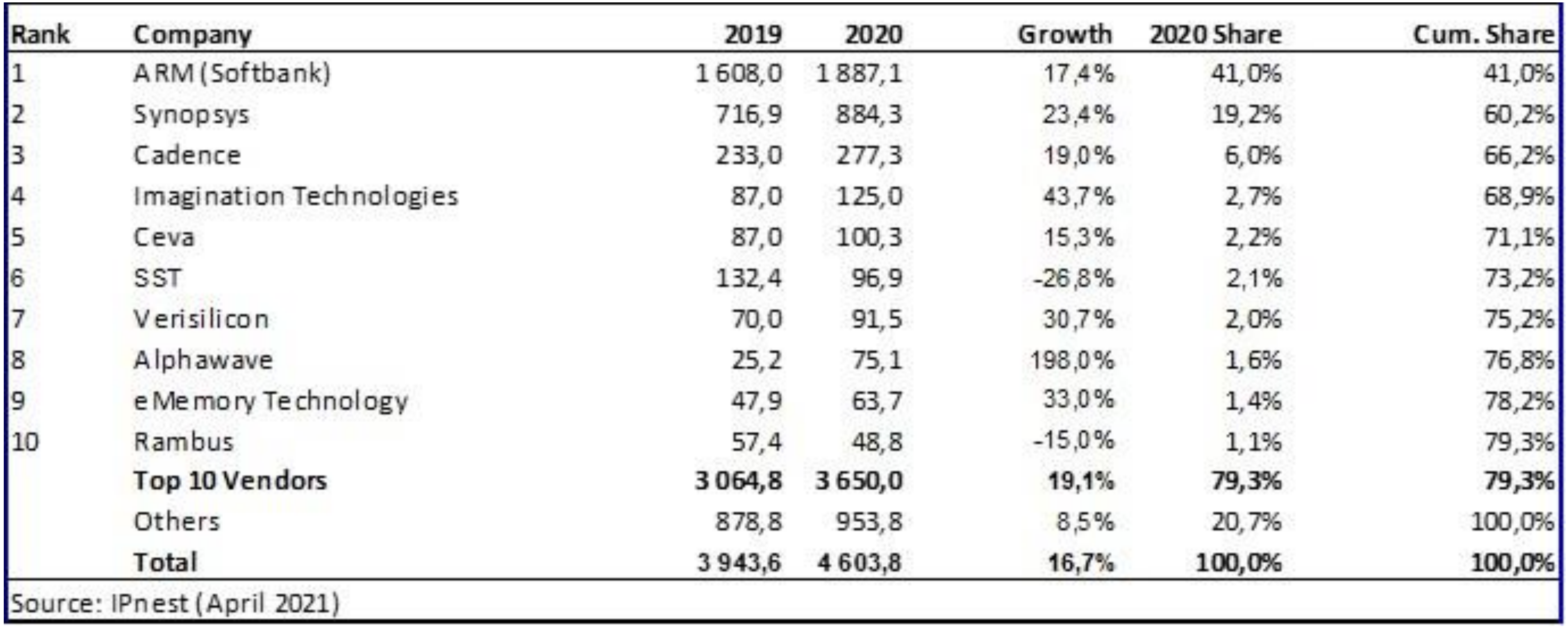
- The philosophy of designing the SoCs…
  - Section I
    - SoC Design Flow Overview
    - SW/HW partition
    - NRE cost
  - Section II
    - SoC processors
    - Memory addressing
    - Design reuse

SOC PROCESSORS

SoC Processors

- Usually a mix of special and general purpose (GP)
  - Can be proprietary design or purchased IP
- Commonly GP processor is purchased IP
  - Includes OS and compiler support
- GP processor optimized for an application
  - Additional instructions
  - Vector units

IP (intellectual property) vendors



Rank	Company	2019	2020	Growth	2020 Share	Cum. Share
1	ARM (Softbank)	1 608,0	1 887,1	17,4%	41,0%	41,0%
2	Synopsys	716,9	884,3	23,4%	19,2%	60,2%
3	Cadence	233,0	277,3	19,0%	6,0%	66,2%
4	Imagination Technologies	87,0	125,0	43,7%	2,7%	68,9%
5	Ceva	87,0	100,3	15,3%	2,2%	71,1%
6	SST	132,4	96,9	-26,8%	2,1%	73,2%
7	Verisilicon	70,0	91,5	30,7%	2,0%	75,2%
8	Alphawave	25,2	75,1	198,0%	1,6%	76,8%
9	eMemory Technology	47,9	63,7	33,0%	1,4%	78,2%
10	Rambus	57,4	48,8	-15,0%	1,1%	79,3%
	Top 10 Vendors	**3 064,8**	**3 650,0**	**19,1%**	**79,3%**	**79,3%**
	Others	878,8	953,8	8,5%	20,7%	100,0%
	Total	**3 943,6**	**4 603,8**	**16,7%**	**100,0%**	**100,0%**

Source: IPnest (April 2021)

source: https://semiwiki.com/ip/297995-design-ip-sales-grew-16-7-in-2020-best-growth-rate-ever/

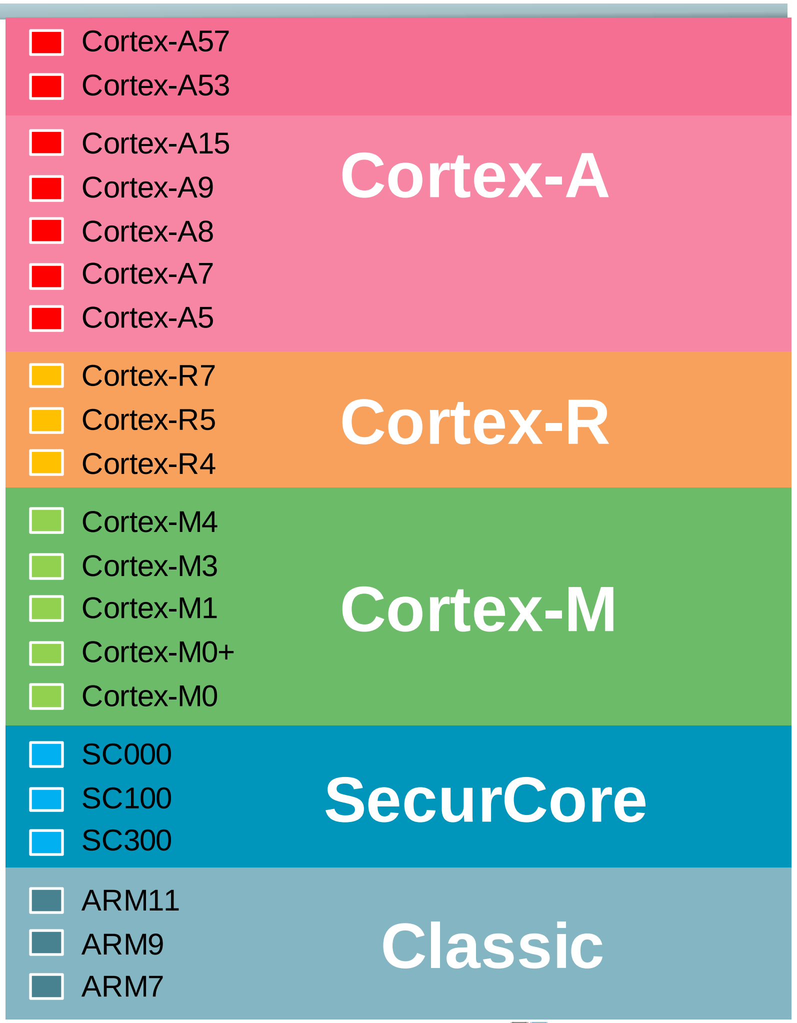
Some processors for SOCs

SOC	*Basic ISA*	*Processor description*
Freescale c600: signal processing	PowerPC	Superscalar with vector extension
ClearSpeed CSX600: general	Proprietary	Array processor with 96 processing elements
PlayStation 2: gaming	MIPS	Pipelined with 2 vector coprocessors
ARM VFP11: general	ARM	Configurable vector coprocessor
SiFIVE Freedom U540	RISC-V*	Open-source ISAs based on reduced instruction set computer principles

*https://riscv.org/exchange/cores-socs/

ARM Processor Families

- **Cortex-A** series (Application)
  - High performance processors for open Operating Systems.
  - Applications include smartphones, digital TV, smart books, home gateways.
- **Cortex-R** series (Real-time)
  - Exceptional performance for real-time applications.
  - Applications include automotive braking systems, powertrains.
- **Cortex-M** series (Microcontroller)
  - Cost-sensitive solutions for deterministic microcontroller applications.
  - Applications include microcontrollers, mixed signal devices, smart sensors, automotive body electronics and airbags.
- **SecurCore** series
  - High security applications.
- Earlier Classic processors
  - Include ARM7, ARM9, ARM11 families.



Cortex
Low-Power Leadership from ARM

source: ARM

ARM Processors and Applications

M

R

A

source: ARM

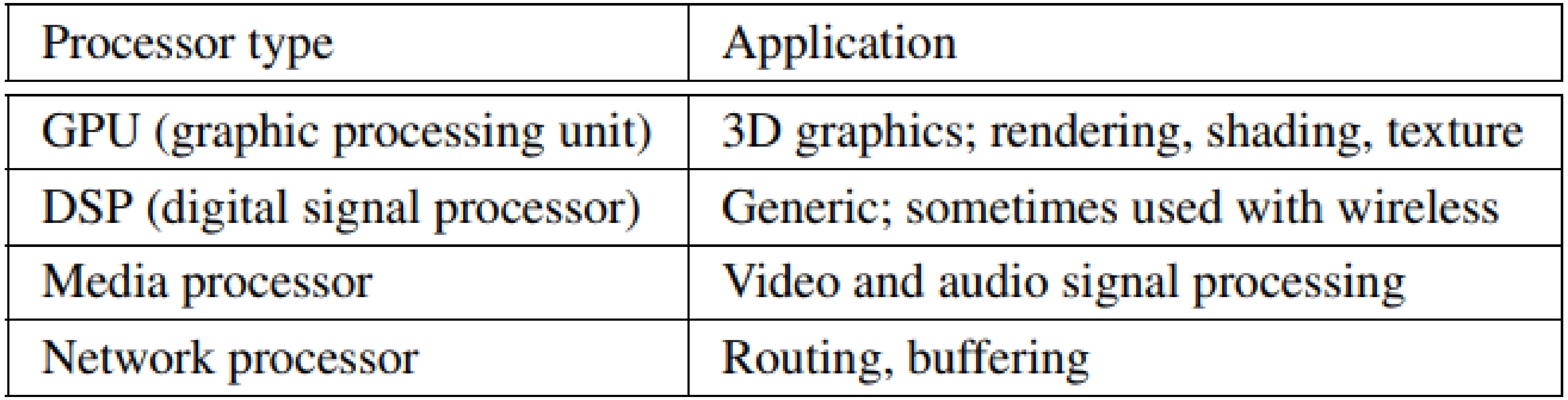
ARM's Products

- **Processors**
  - **Cortex-A, R, M, SecurCore…**
- System IP
  - CoreLink, CoreSight, AMBA Design Tools…
- Multimedia
  - Mali Graphics, Video, Display…
- Physical IP
  - Artisan Logic IP, Interface IP, Memory IP, DesignStart…
- Tools
  - Software Tools (DS-5, Keil MDK), Debug adapters, models, boards…
- Supports
  - Training, documentation, ARM Connect Community…

source: ARM

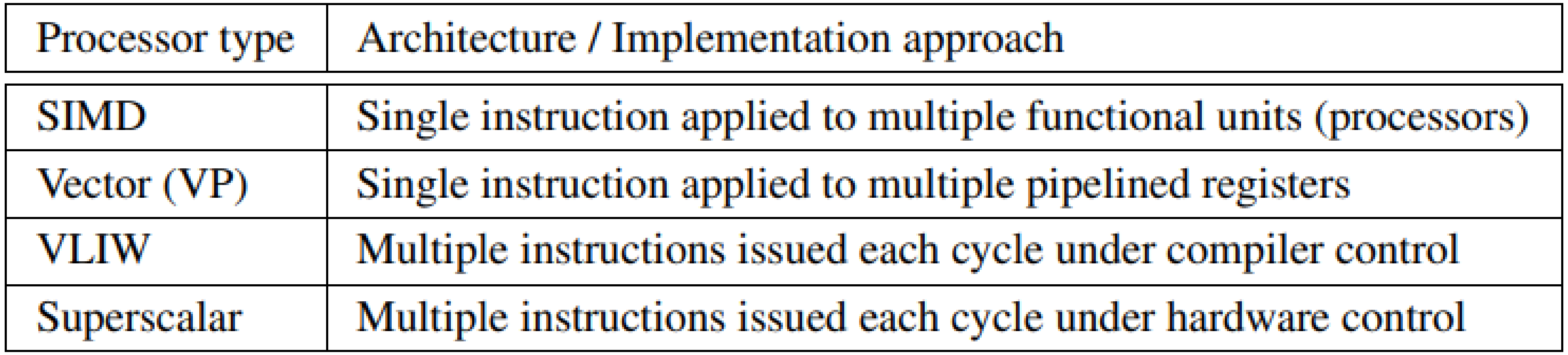
Processor Types

by function



Processor type	Application
GPU (graphic processing unit)	3D graphics; rendering, shading, texture
DSP (digital signal processor)	Generic; sometimes used with wireless
Media processor	Video and audio signal processing
Network processor	Routing, buffering

by architecture



Processor type	Architecture / Implementation approach
SIMD	Single instruction applied to multiple functional units (processors)
Vector (VP)	Single instruction applied to multiple pipelined registers
VLIW	Multiple instructions issued each cycle under compiler control
Superscalar	Multiple instructions issued each cycle under hardware control

Sequential and parallel machines

- Basic single stream processors
  - Pipelined: basic sequential
  - Superscalar: transparently concurrent
  - VLIW: compiler generated concurrency
- Multiple stream
  - Array processors
  - Vector processors
- Multiprocessors

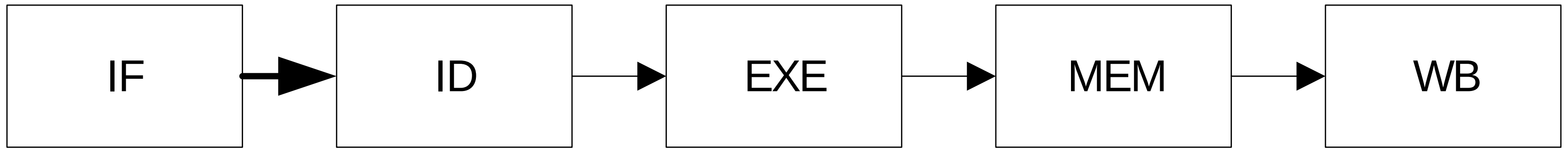
Sequential machines

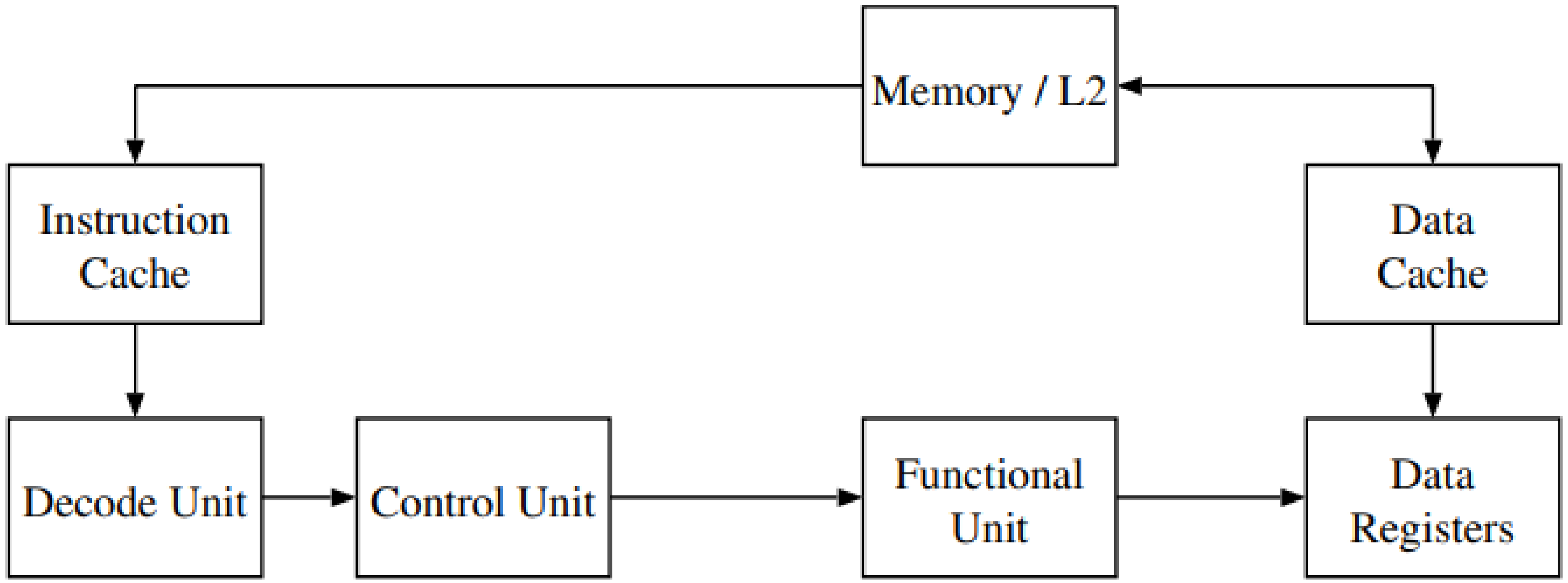
- operation
  - generally transparent to sequential programmer
  - appear as in order instruction execution
- pipeline processor
  - execution in order
  - limited to one instruction execution / cycle
- superscalar processor
  - multi instructions / cycle, managed by hardware
- VLIW
  - multi op execution / cycle, managed by compiler

Adding instructions

- additional instructions to support specialized resources
  - exception: superscalar, with hardware control
- instructions can be added to base processor for coprocessor control
  - VLIW: Very Large Instruction Word
  - Array
  - Vector

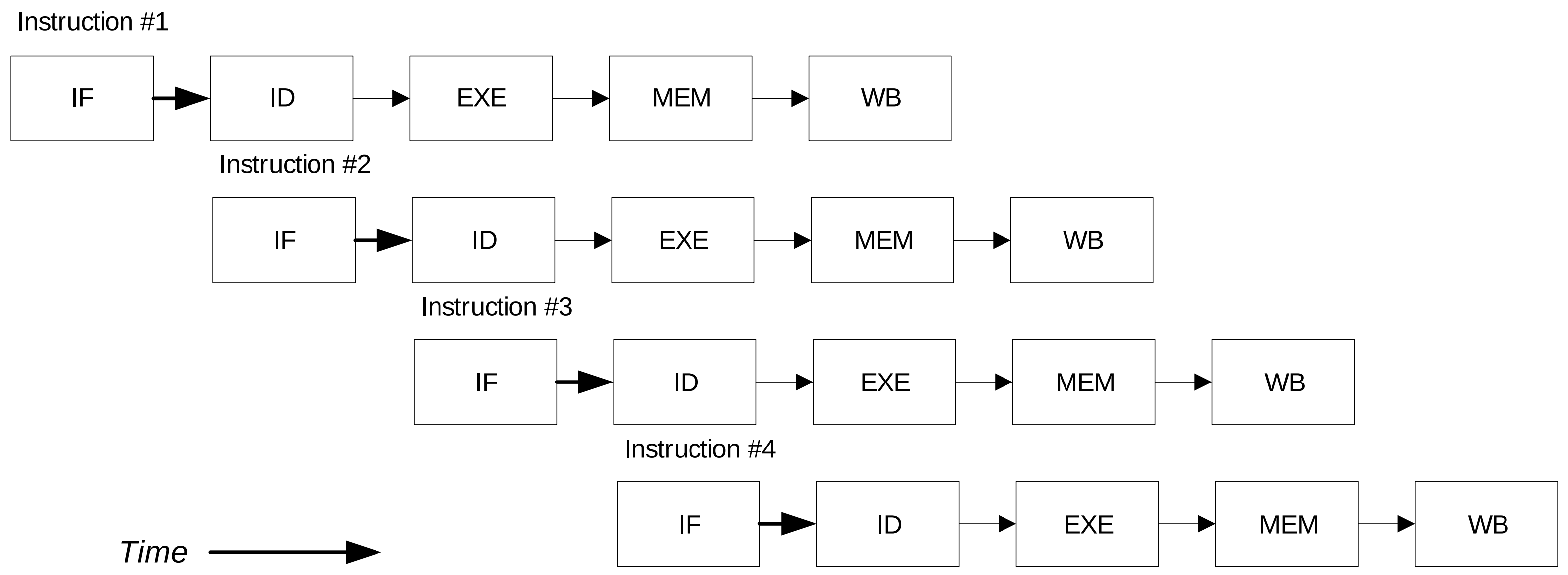
Sequential processor model





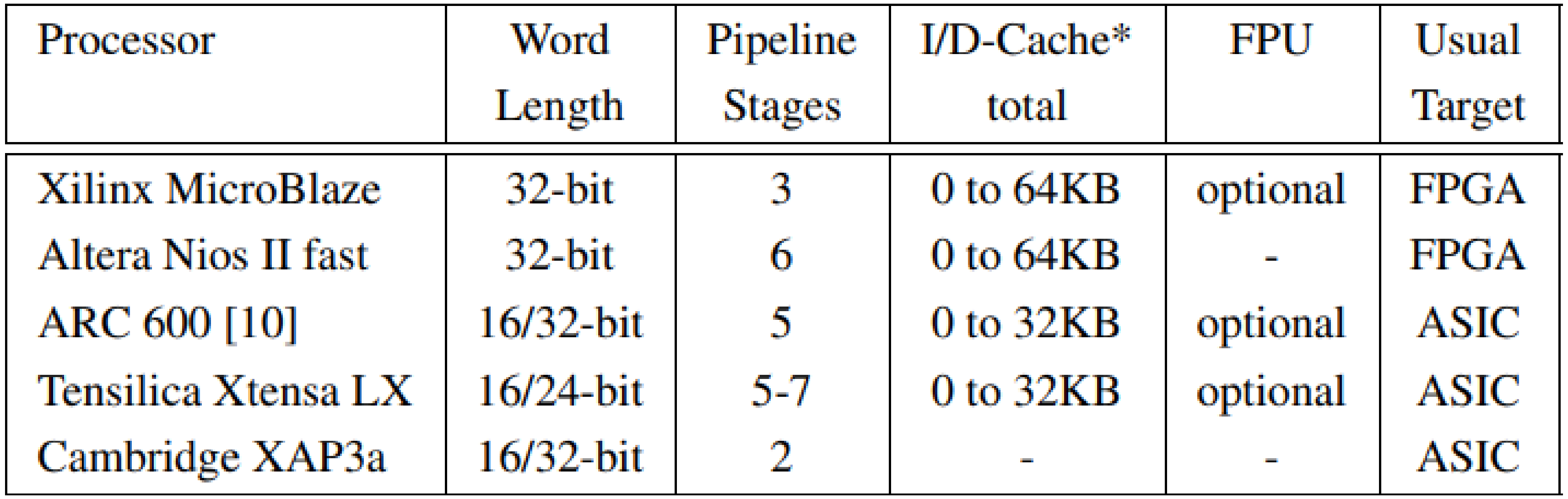
Pipelined processor

Exploiting parallelism that is based on concurrently performing different phases



- IF: Instruction Fetch
- ID: Instruction Decode
- EXE: Execution
- MEM: Memory access
- WB: Write Back

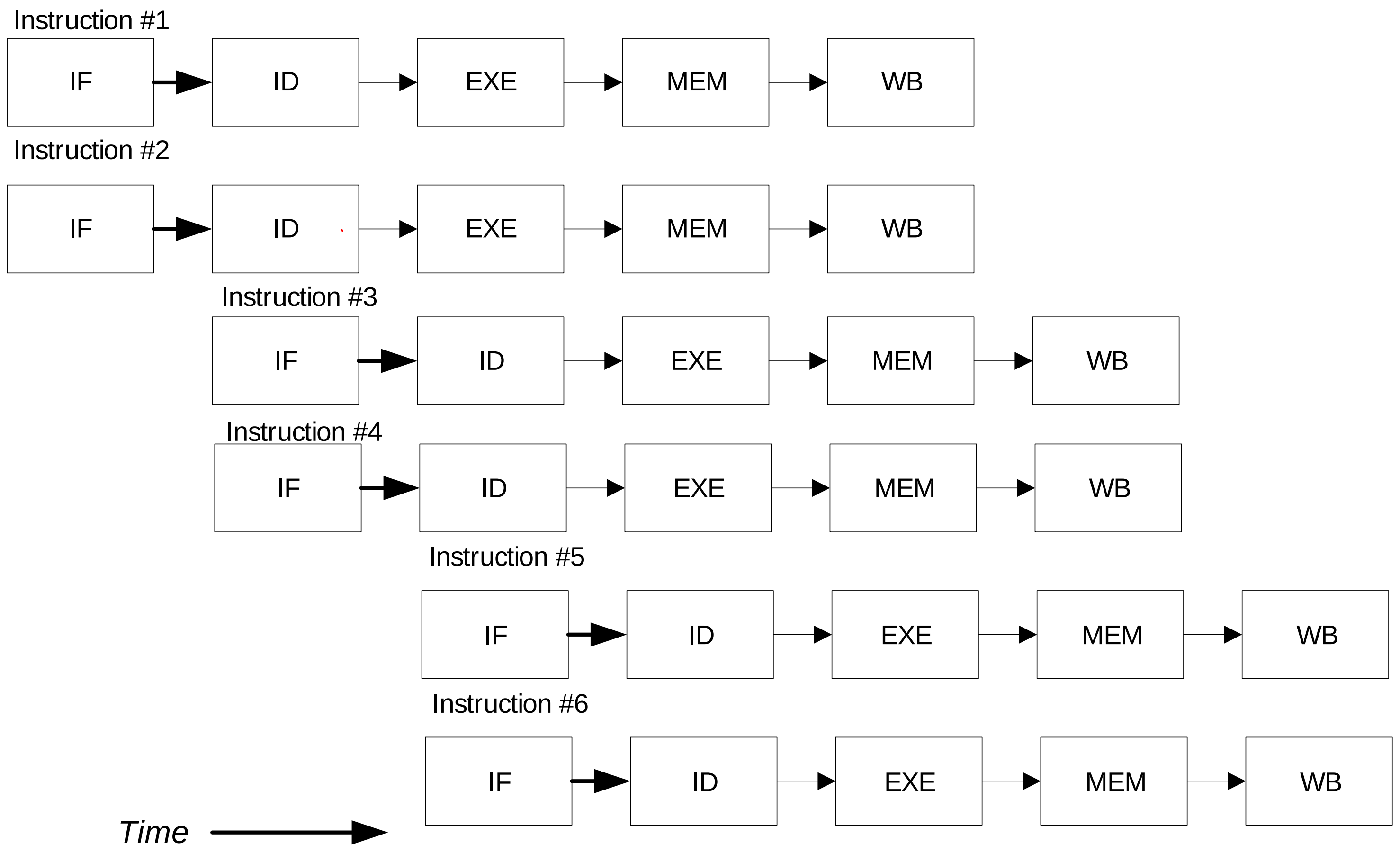
SOC examples using pipelined soft processors



Processor	Word Length	Pipeline Stages	I/D-Cache* total	FPU	Usual Target
Xilinx MicroBlaze	32-bit	3	0 to 64KB	optional	FPGA
Altera Nios II fast	32-bit	6	0 to 64KB	-	FPGA
ARC 600 [10]	16/32-bit	5	0 to 32KB	optional	ASIC
Tensilica Xtensa LX	16/24-bit	5-7	0 to 32KB	optional	ASIC
Cambridge XAP3a	16/32-bit	2	-	-	ASIC

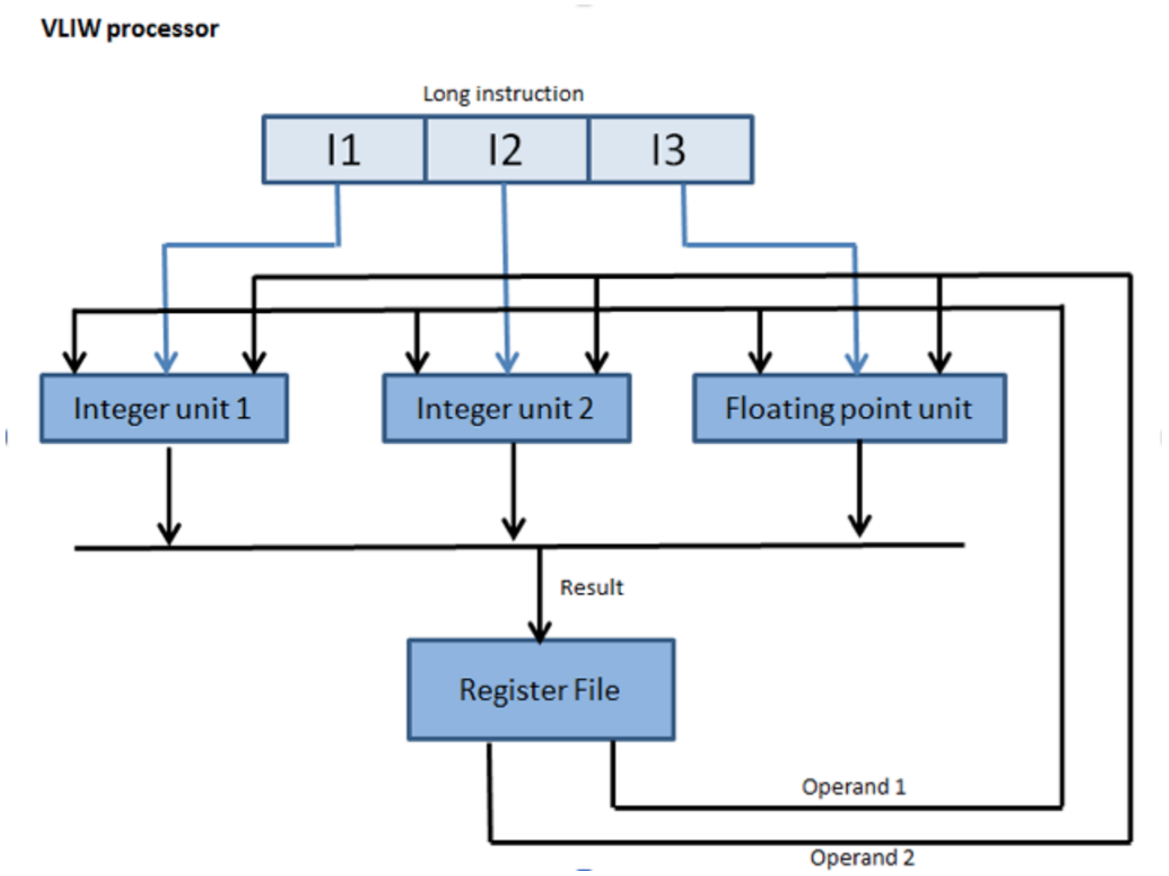
Superscalar processors

Dynamically examines the instruction stream to determine which operations are independent and can be executed.

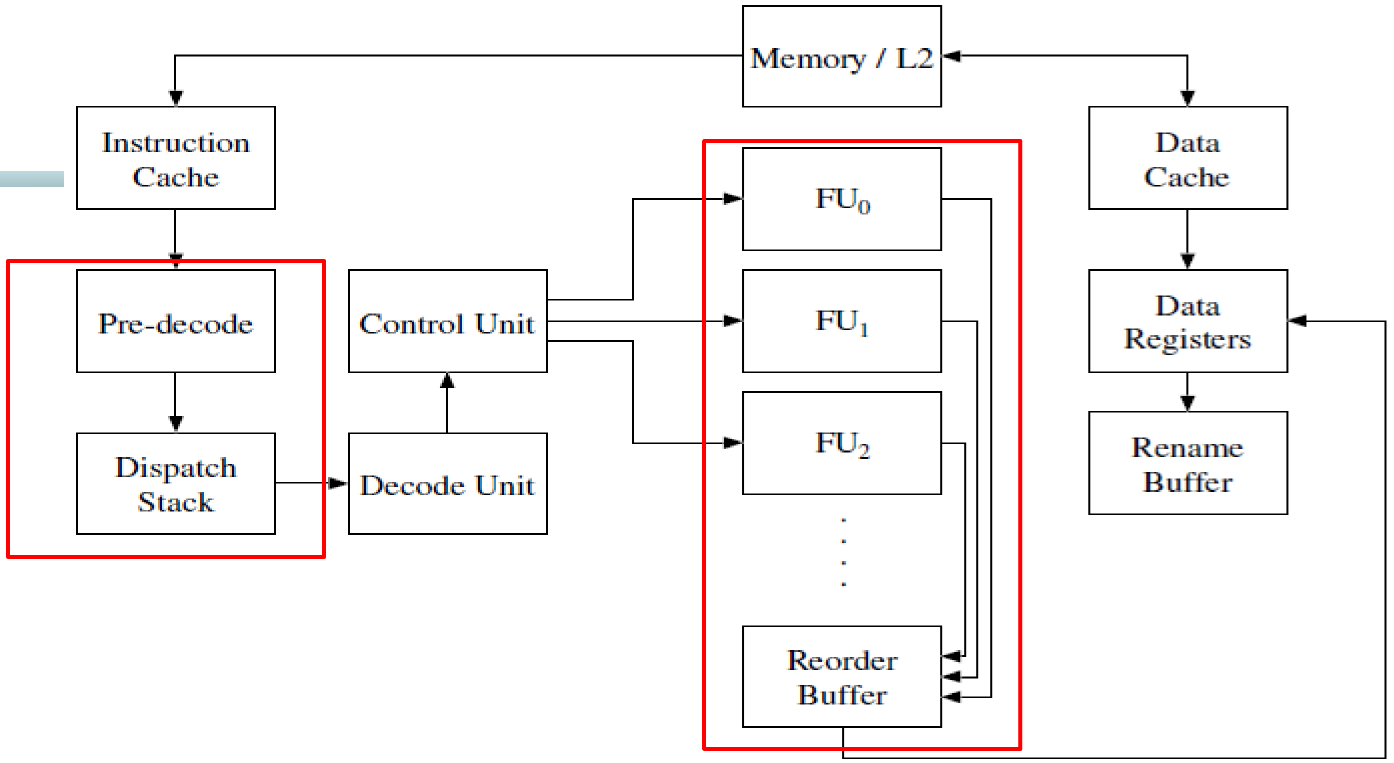


VLIW (Very Large Instruction Word)

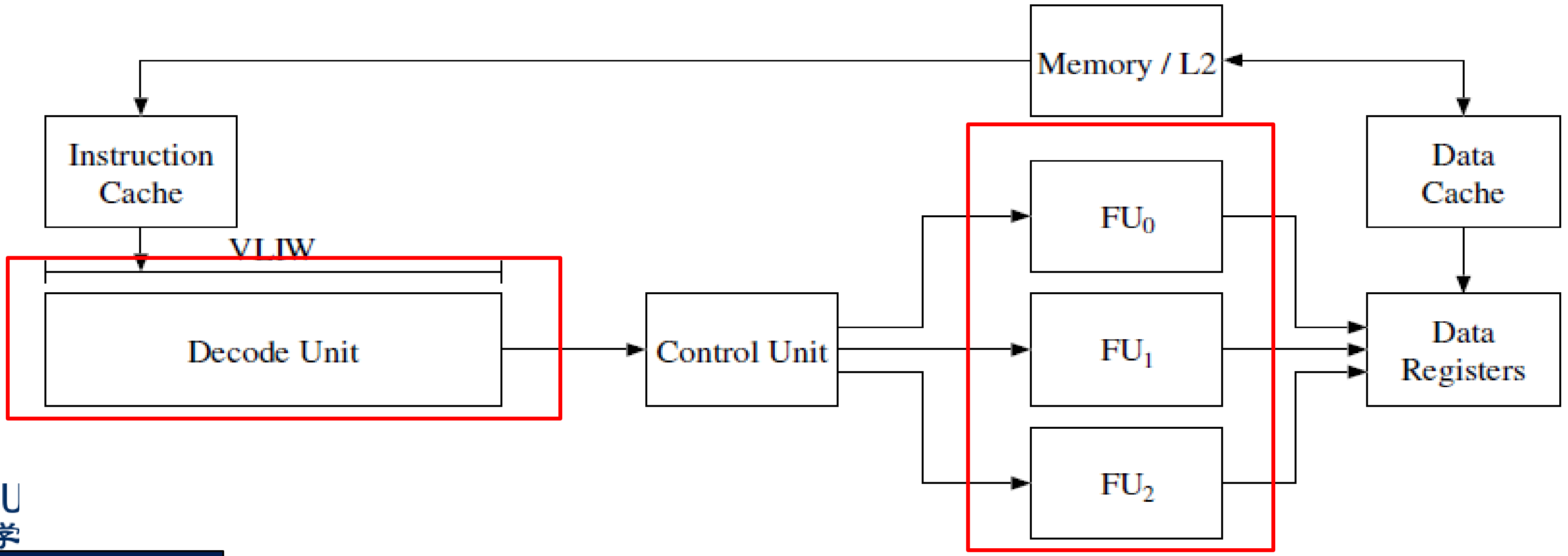
- Relies on the compiler to analyze the available operations (OP) and to schedule independent operations into wide instruction words, which then executes these operations in parallel.



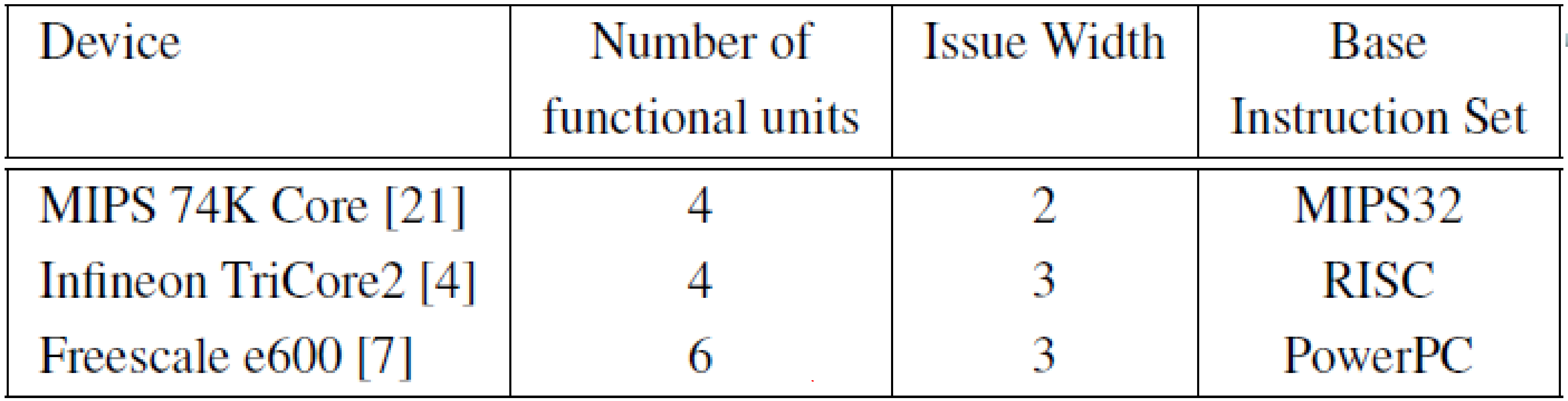
Superscalar



VLIW

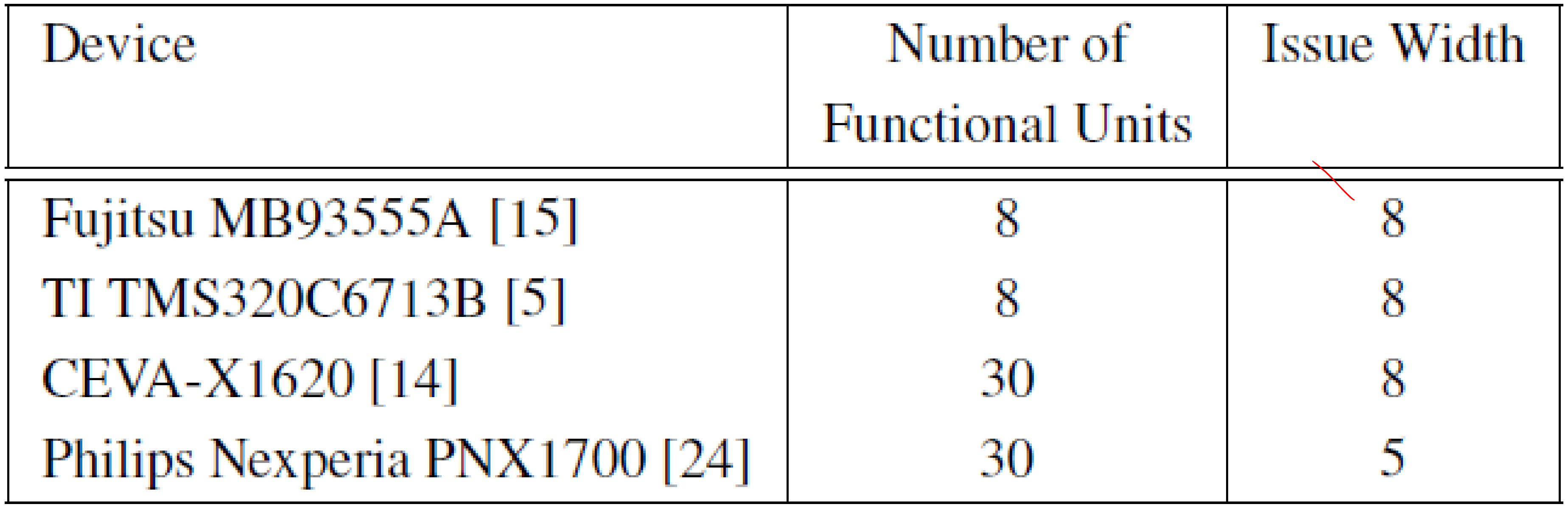


Superscalar



Device	Number of functional units	Issue Width	Base Instruction Set
MIPS 74K Core [21]	4	2	MIPS32
Infineon TriCore2 [4]	4	3	RISC
Freescale e600 [7]	6	3	PowerPC

VLIW

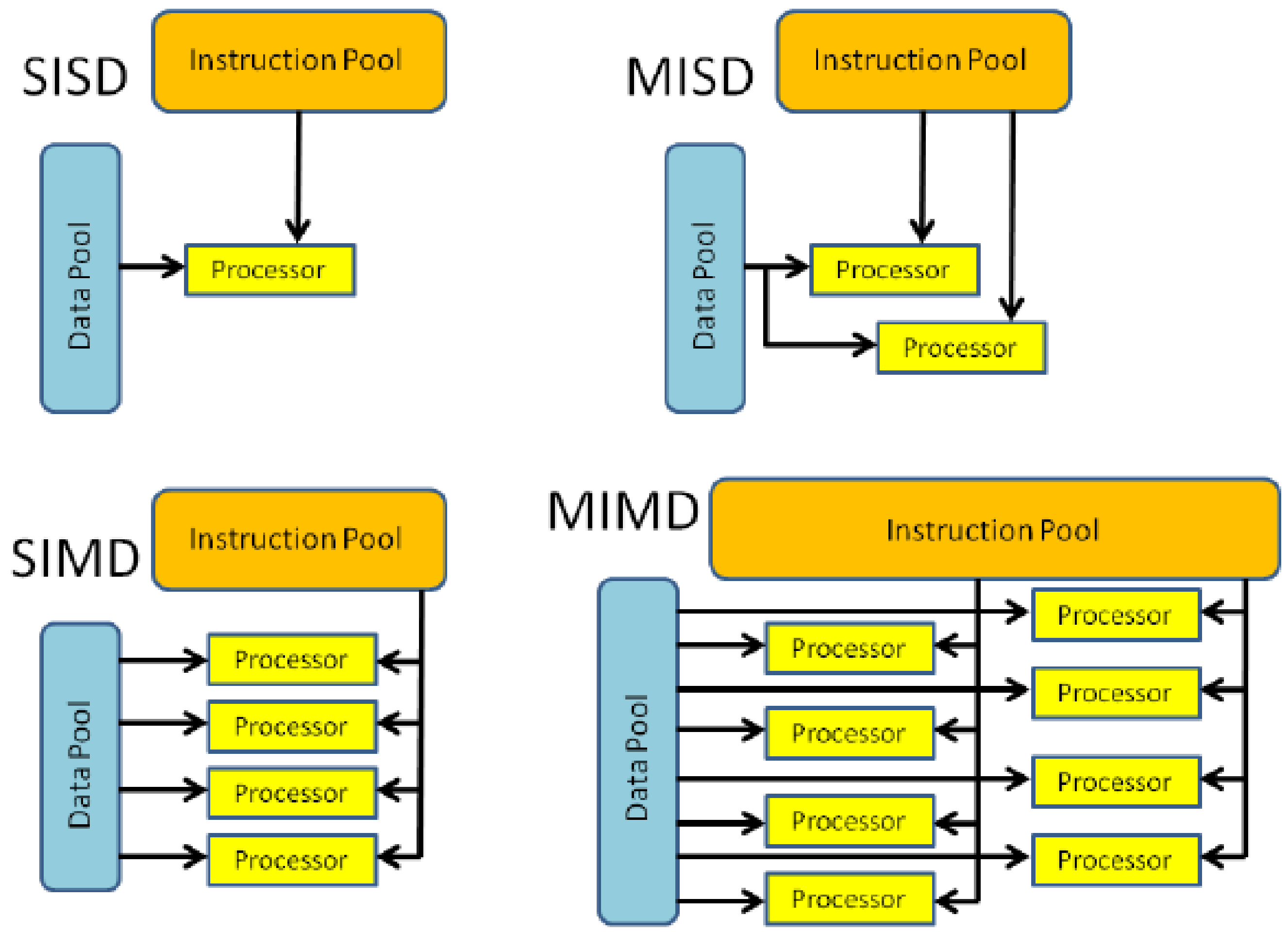


Device	Number of Functional Units	Issue Width
Fujitsu MB93555A [15]	8	8
TI TMS320C6713B [5]	8	8
CEVA-X1620 [14]	30	8
Philips Nexperia PNX1700 [24]	30	5

Parallel processors

- Execution managed by programmer
- array processors
  - single instruction stream, multiple data streams: SIMD
- vector processors
  - SIMD
- GPUs
  - SIMD
- multiprocessors
  - multiple instruction streams, multiple data streams: MIMD

Flynn's Classification

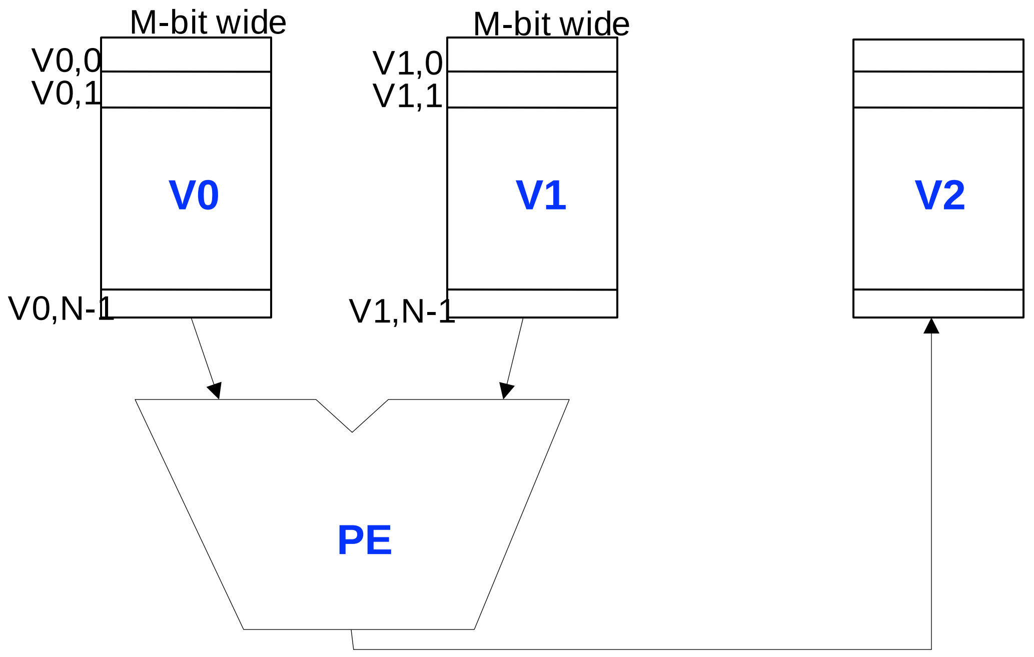


source: https://quizlet.com/229852945/flynns-taxonomy-diagram/

Vector vs. Array processor

- Array processor: Instruction operates on multiple data elements at the same time using different spaces (PEs)
- Vector processor: Instruction operates on multiple data elements in consecutive time steps using the same space (PE)
- The difference between an array processor and a vector processor is that a vector processor uses multiple vector pipelines whereas an array processor employs a number of processing elements to operate in parallel.

- Each vector data register holds N M-bit values
- Each register stores a vector



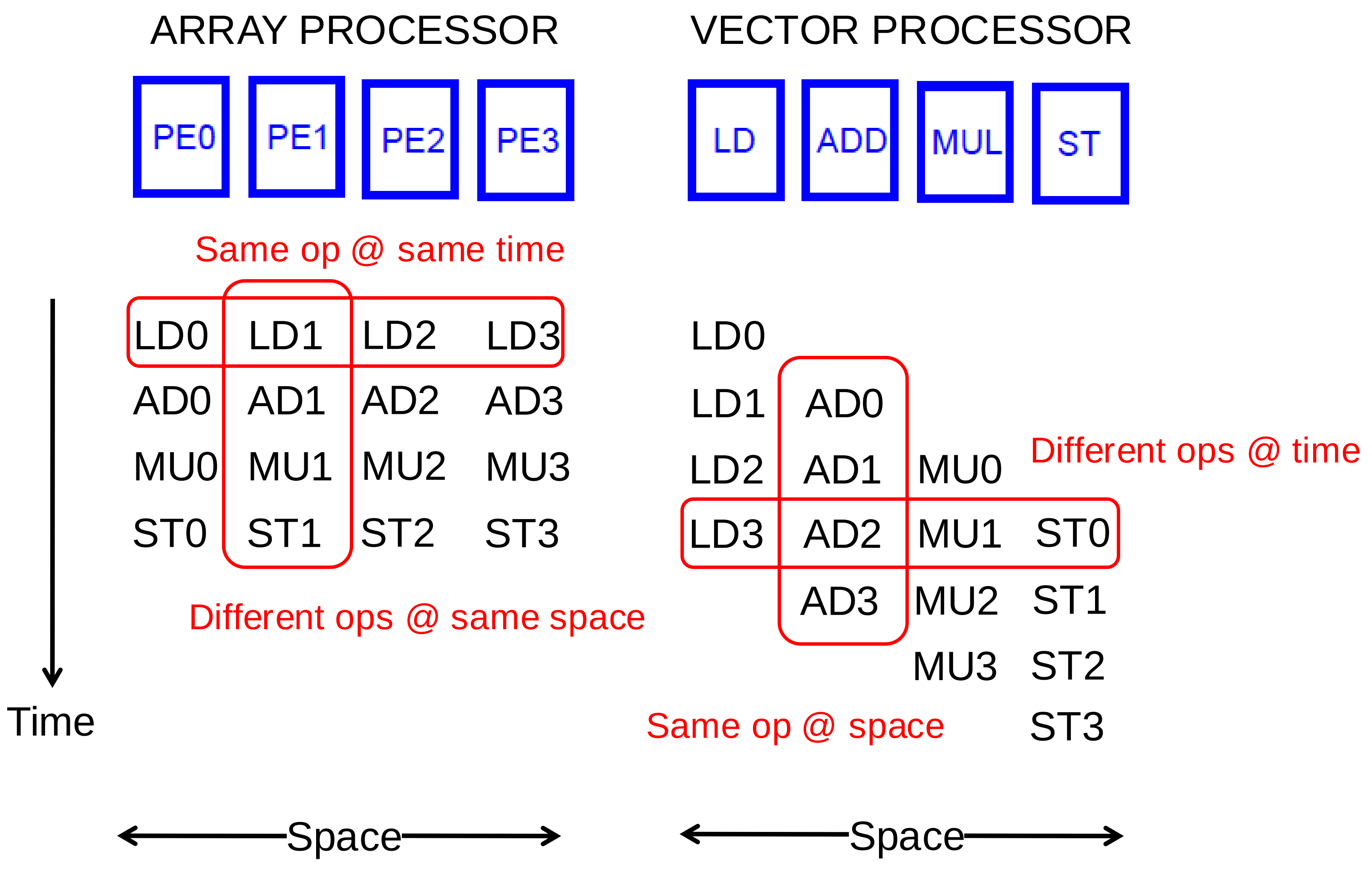
source: Prof. Onur Mutlu, Digital Design & Computer Arch., ETH Zürich

Array vs. Vector Processors

Each has four instructions

Instruction Stream

```
LD    VR ← A[3:0]
ADD   VR ← VR, 1
MUL   VR ← VR, 2
ST    A[3:0] ← VR
```



More expensive

source: Prof. Onur Mutlu, Digital Design & Computer Arch., ETH Zürich

Where to use array processor?

- Large data with regular structures
- Computations on the data which are uniformly applied to many or all elements of the data set
- Simple and regular patterns relating the computations and the data

Example:

$K(p, q) = \sum I(\mathrm{p}, \mathrm{r}).\, J(\mathrm{r}, \mathrm{q})$ for r = 0, 1, 2, …, up to (n–1)

Where p = 0, 1, 2, …, up to (n–1)

q = 0, 1, 2, …, up to (n–1)

Vector processors

- Vector: sequences of data values that are seemingly operated on as a single entity.
- Vectors explicitly loaded into special vector registers and stored back into memory
- Ability to concurrently load and store data between vector registers and memory while performing computations
  - Enabling strip-mining (It increases the temporal and spatial locality in the data cache if the data are reusable in different passes of an algorithm.)

Example1: Before Vectorization

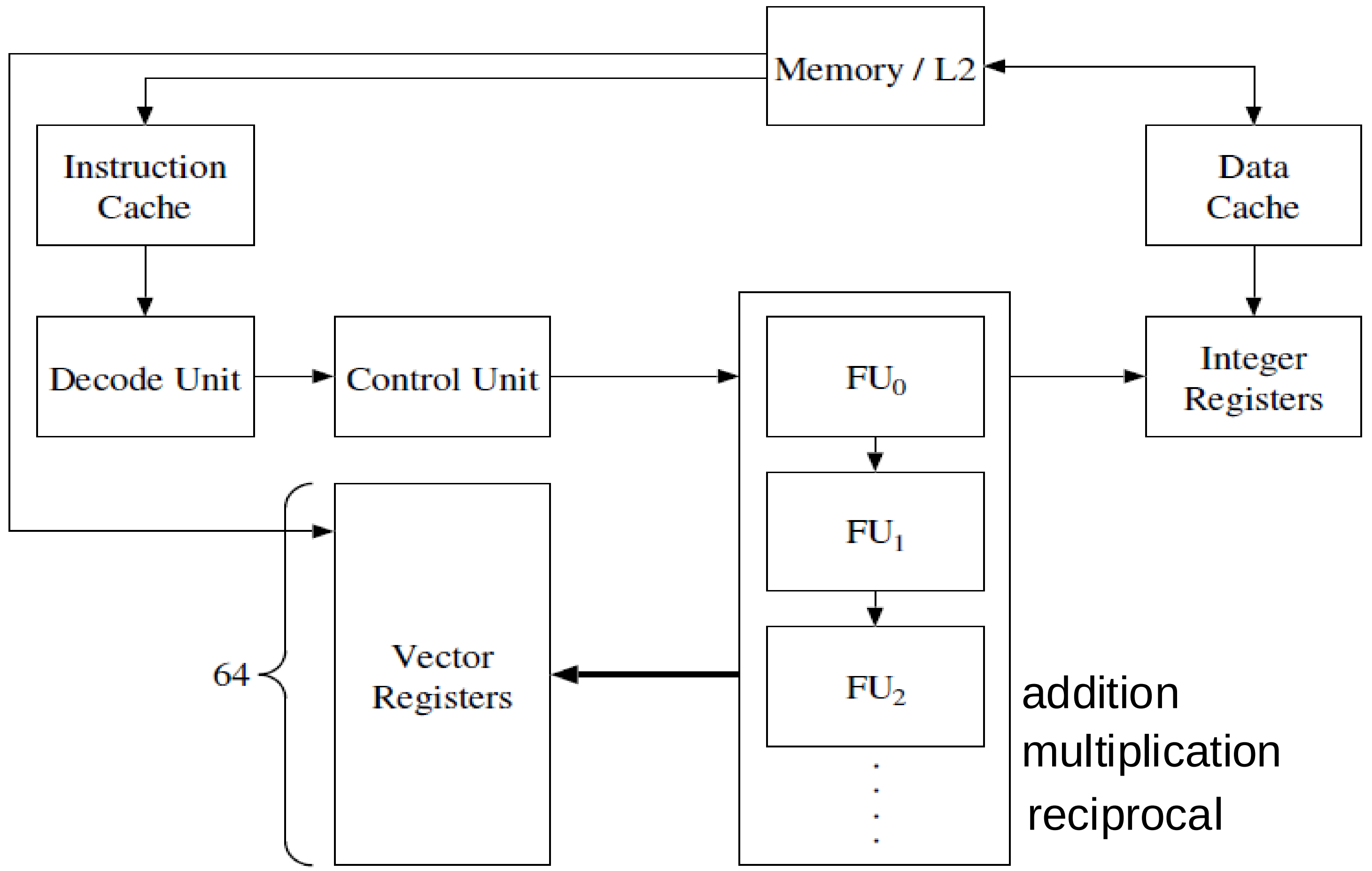
```
i = 1
do while (i<=n)
a(i) = b(i) + c(i) ! Original loop code
i = i + 1
end do
```

Example 2: After Vectorization

```
!The vectorizer generates the following two loops
i = 1
do while (i < (n - mod(n,4)))
! Vector strip-mined loop.
a(i:i+3) = b(i:i+3) + c(i:i+3)
i = i + 4
end do
do while (i <= n)
a(i) = b(i) + c(i)      !Scalar clean-up loop
i = i + 1
end do
```

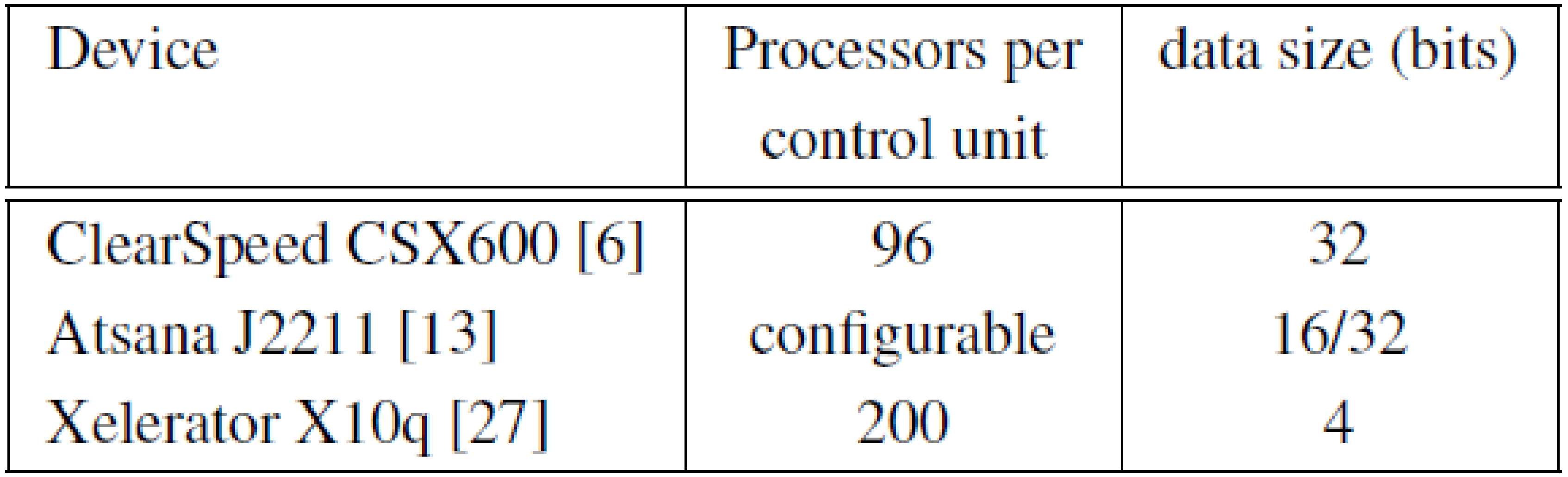
Vector processors

- vector registers, eg 8 regs x 64 words x 64 bits
- vector instructions: *VR3 <- VR2 VOP VR1*



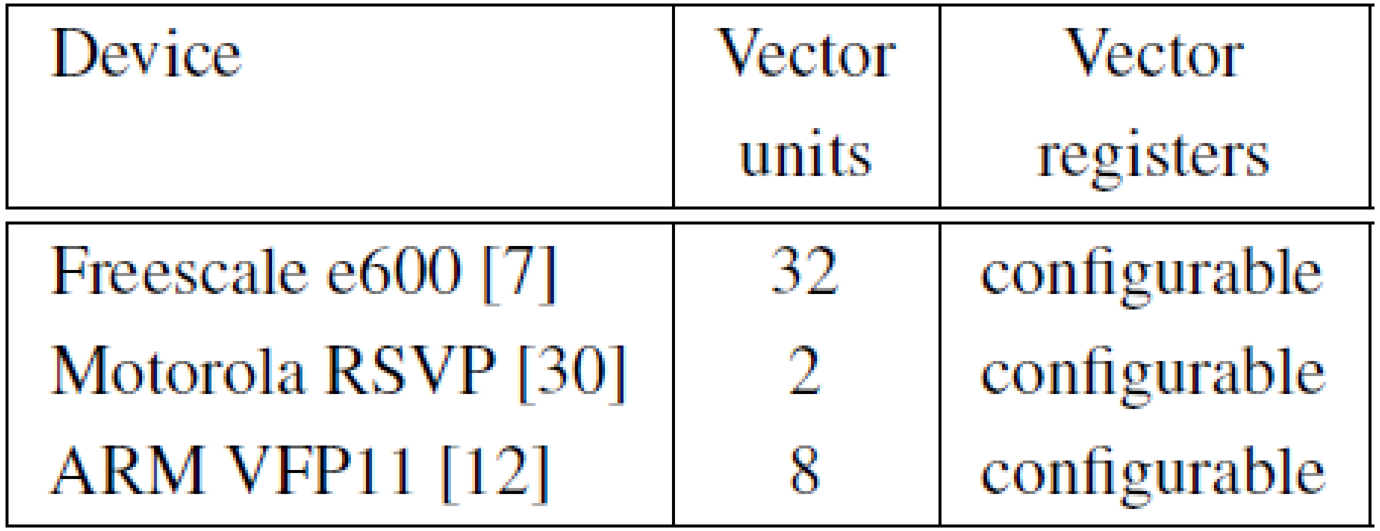
addition
multiplication
reciprocal

Array Processors



Device	Processors per control unit	data size (bits)
ClearSpeed CSX600 [6]	96	32
Atsana J2211 [13]	configurable	16/32
Xelerator X10q [27]	200	4

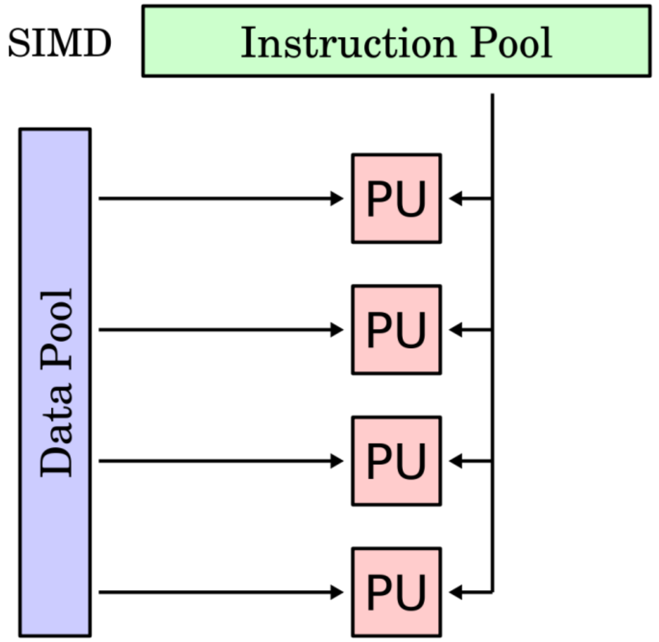
Vector Processors



Device	Vector units	Vector registers
Freescale e600 [7]	32	configurable
Motorola RSVP [30]	2	configurable
ARM VFP11 [12]	8	configurable

Graphics Processing Units

- Recall: SIMD
- A combination of both vector and array

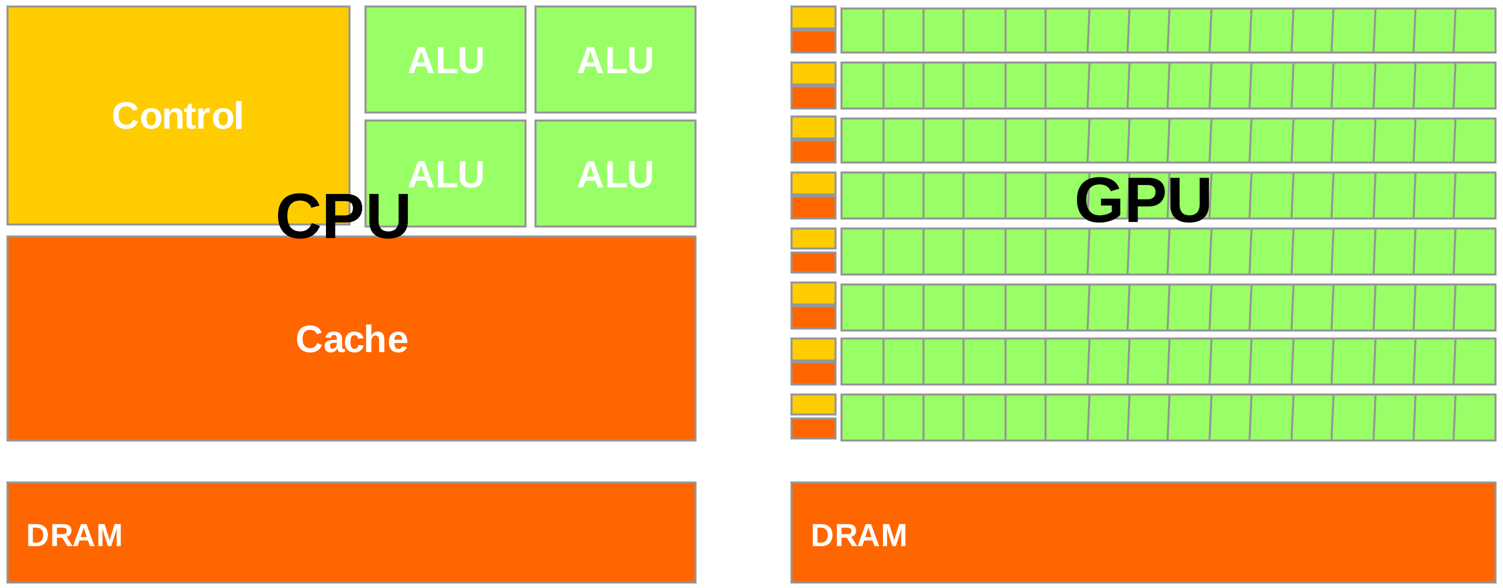


Source: Wikipedia

A processor optimized for 2D and 3D graphics, video, visual computing, and display, now designed for rapid manipulation and parallel processing of data (e.g. AI)

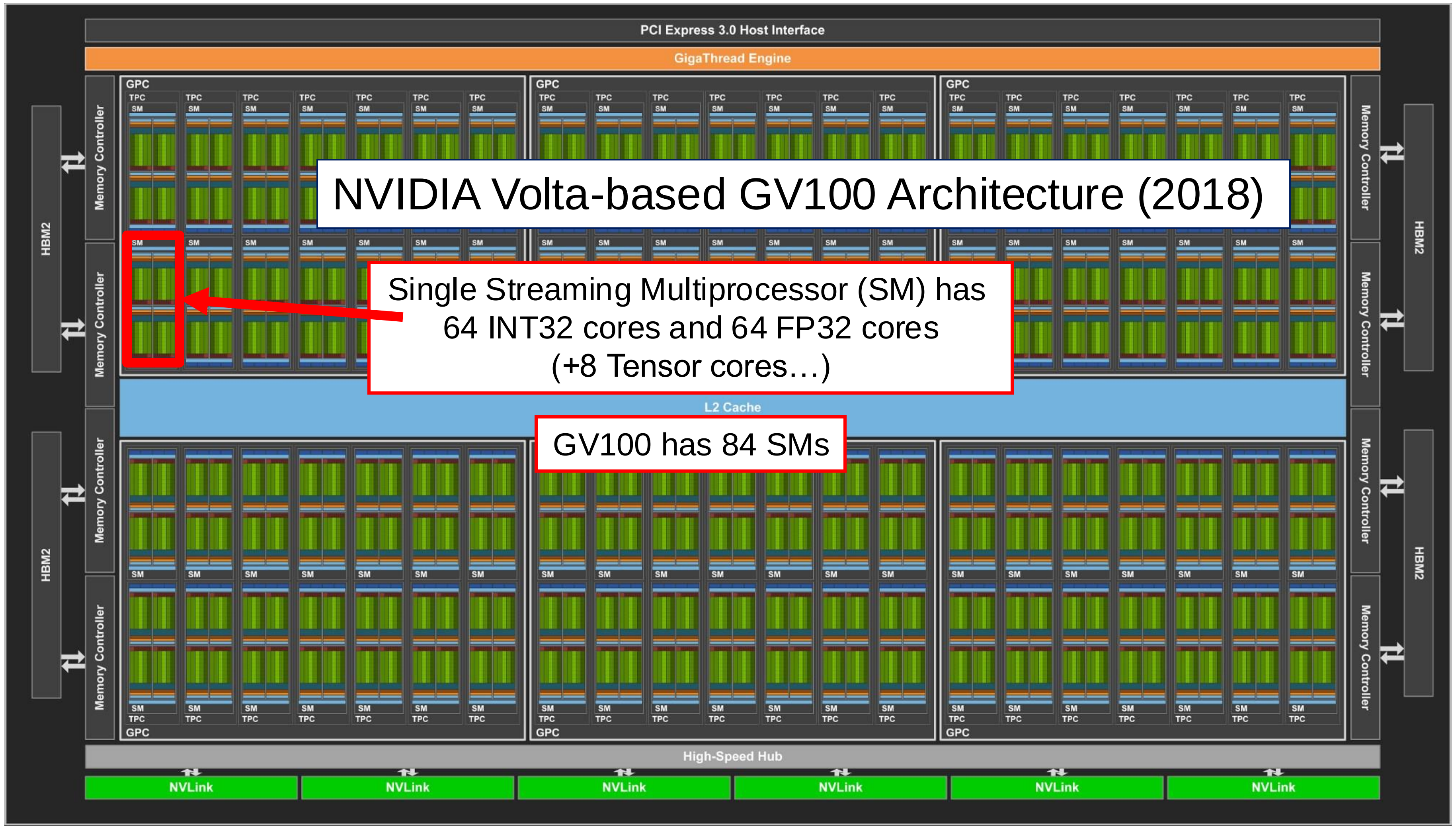
Image: Nvidia

Parallelism



© David Kirk/NVIDIA and Wen-mei W. Hwu

Parallelism



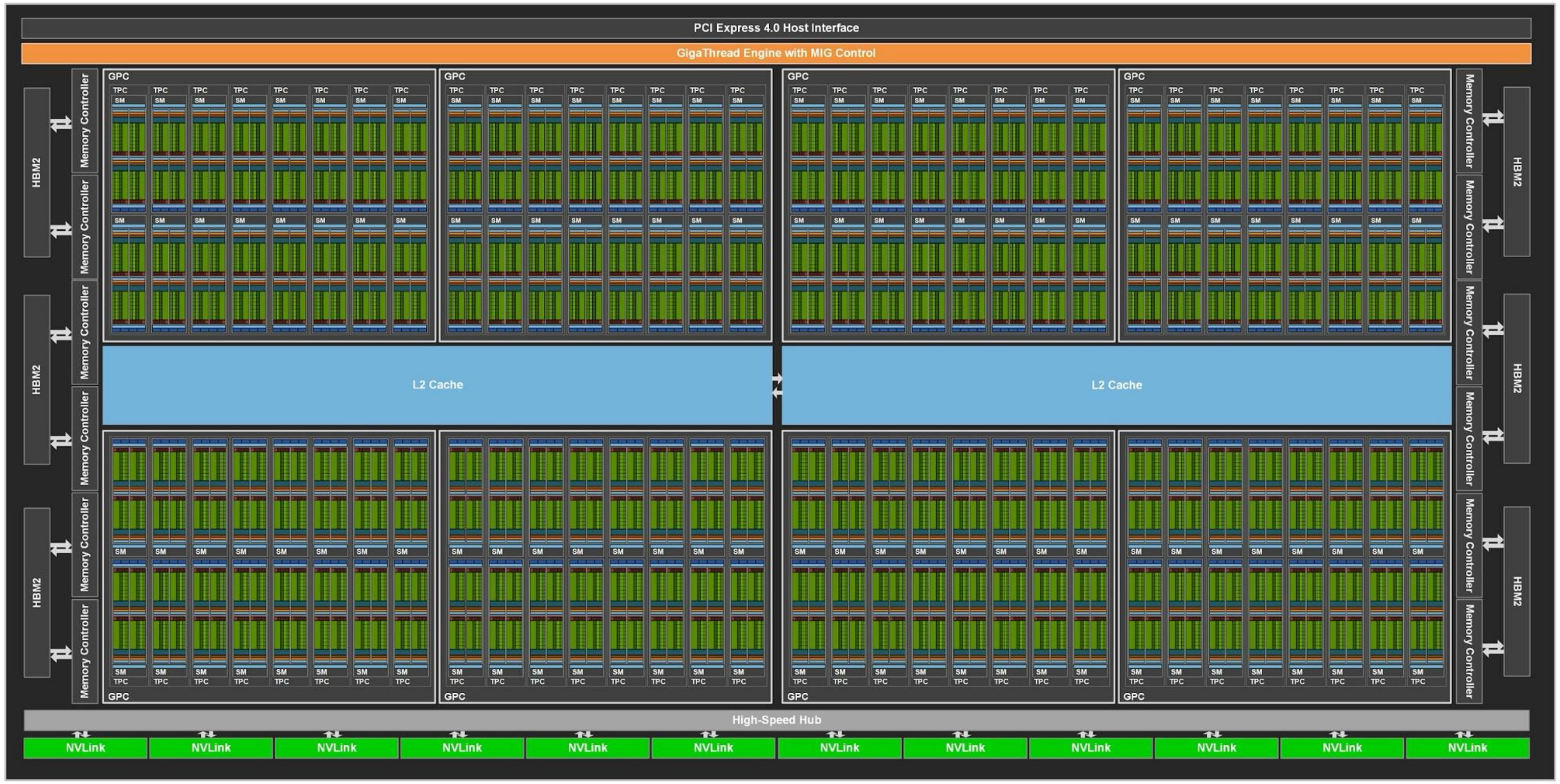
NVIDIA Volta-based GV100 Architecture (2018)

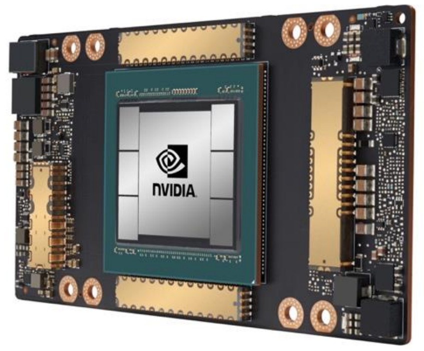
Single Streaming Multiprocessor (SM) has 64 INT32 cores and 64 FP32 cores (+8 Tensor cores…)

GV100 has 84 SMs

source: Nvidia

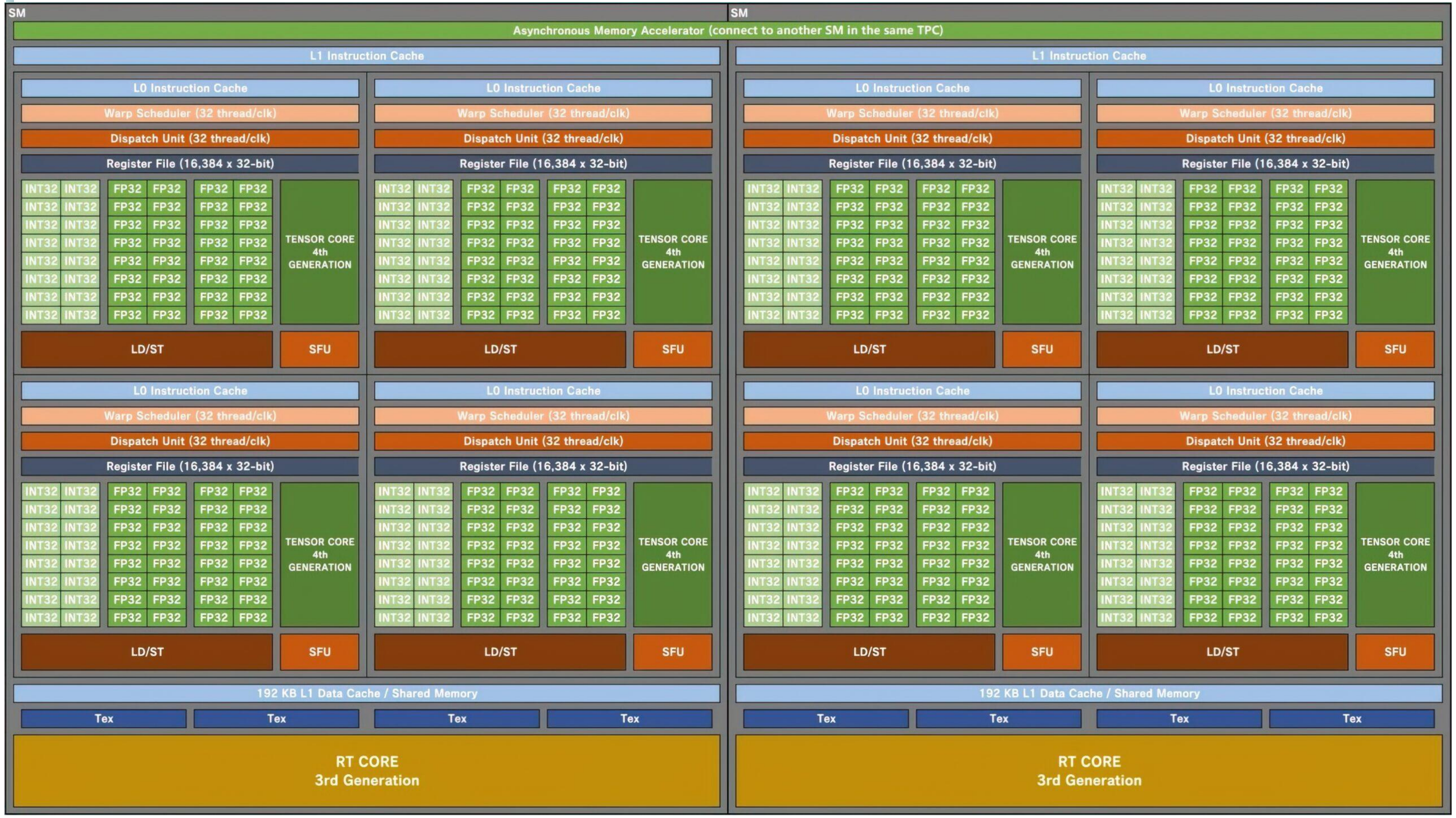
Nvidia Ampere GA100 (2020)

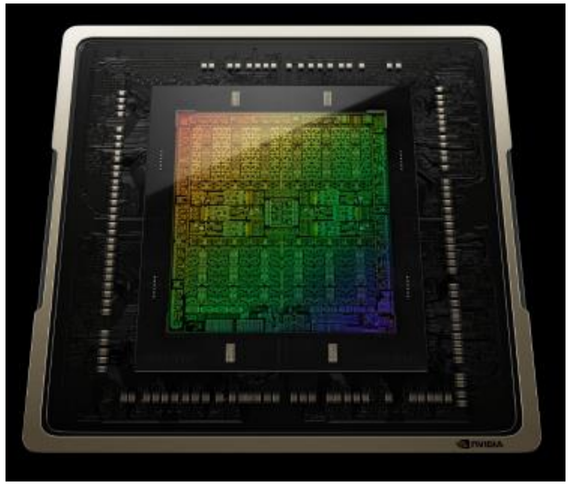

PCI Express 4.0 Host Interface
GigaThread Engine with MIG Control
GPC
TPC
SM
Memory Controller
HBM2
L2 Cache
High-Speed Hub
NVLink



source: Nvidia

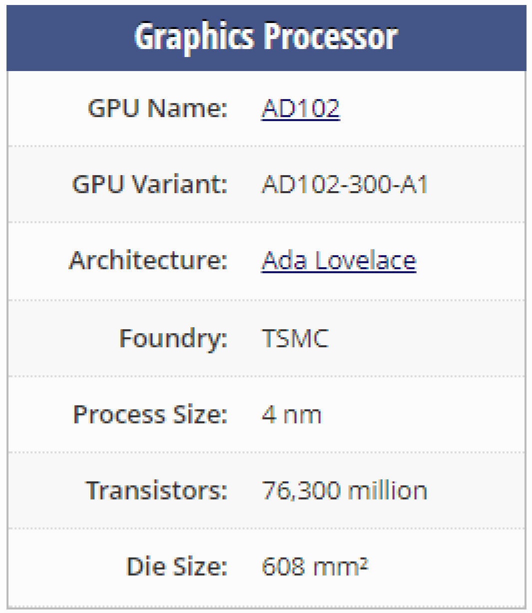
Nvidia 'Ada Lovelace' (2022)


SM
SM
Asynchronous Memory Accelerator (connect to another SM in the same TPC)
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
TENSOR CORE 4th GENERATION
LD/ST
SFU
192 KB L1 Data Cache / Shared Memory
Tex
RT CORE 3rd Generation

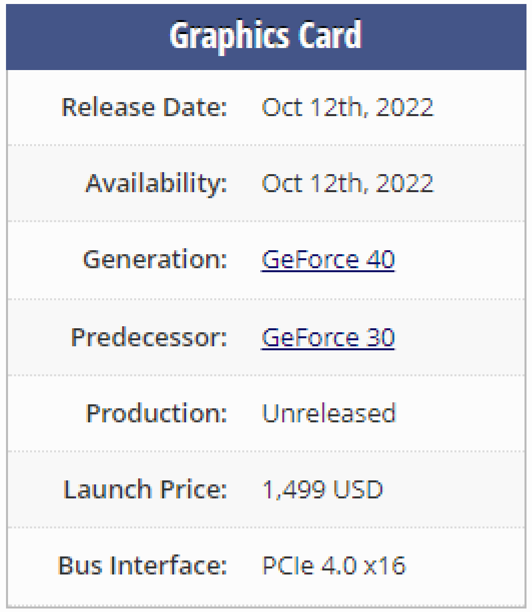


NVIDIA AD102 'Ada Lovelace' Gaming GPU 'SM' Block Diagram (Image Credits: Kopite7kimi)

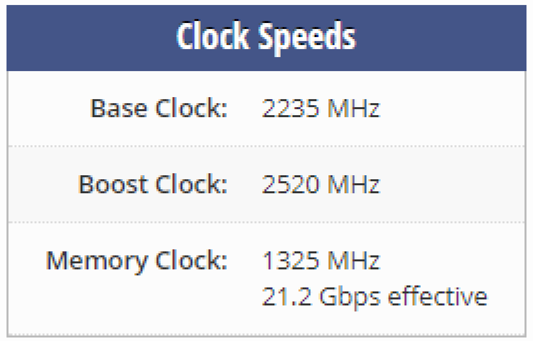
NVIDIA GeForce RTX 4090 (2022)



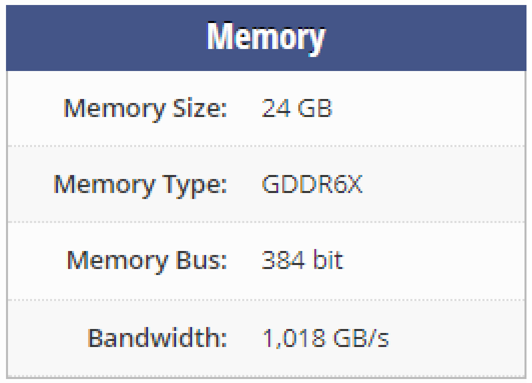
Graphics Processor	
GPU Name:	AD102
GPU Variant:	AD102-300-A1
Architecture:	Ada Lovelace
Foundry:	TSMC
Process Size:	4 nm
Transistors:	76,300 million
Die Size:	608 mm²



Graphics Card	
Release Date:	Oct 12th, 2022
Availability:	Oct 12th, 2022
Generation:	GeForce 40
Predecessor:	GeForce 30
Production:	Unreleased
Launch Price:	1,499 USD
Bus Interface:	PCIe 4.0 x16



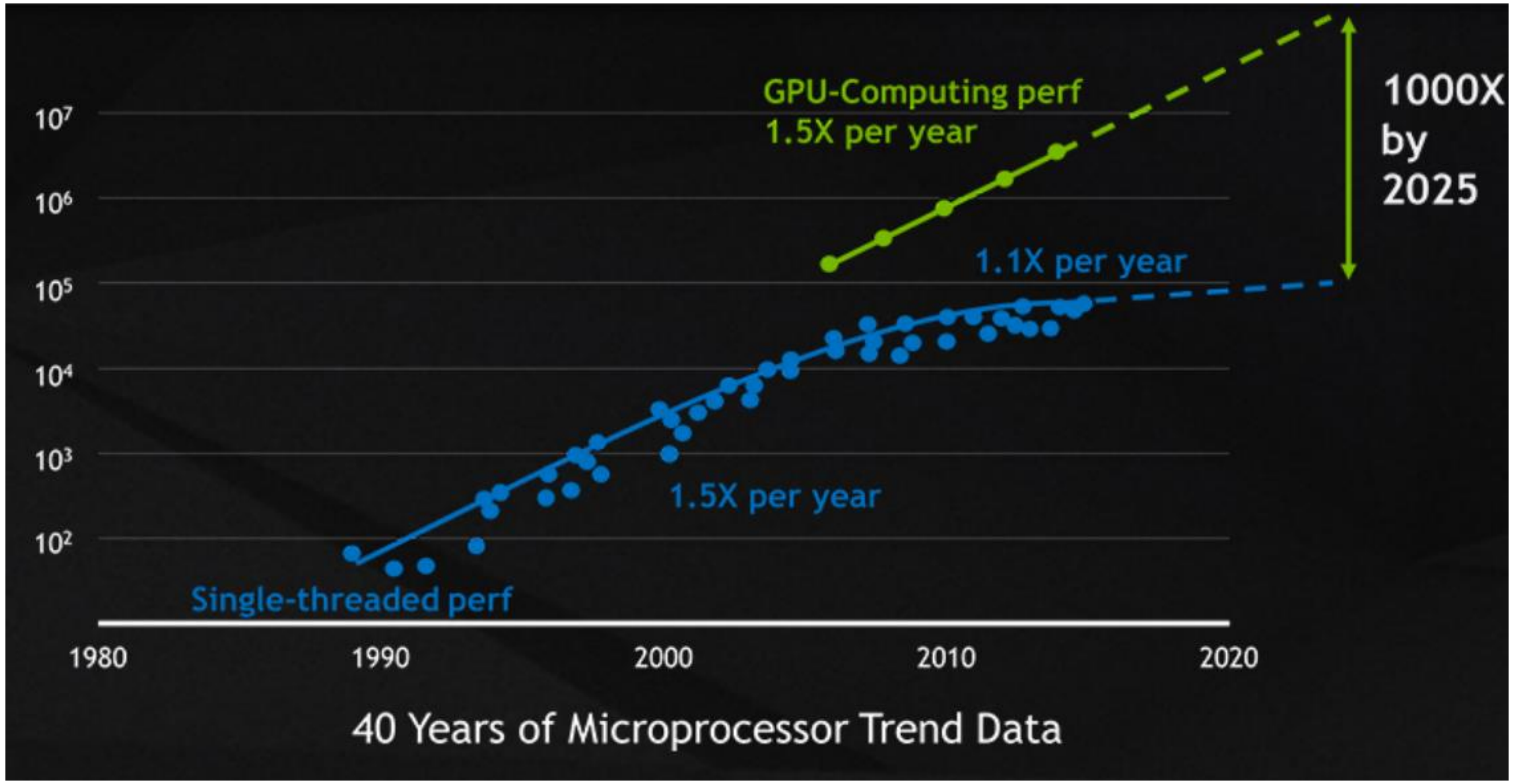
Clock Speeds	
Base Clock:	2235 MHz
Boost Clock:	2520 MHz
Memory Clock:	1325 MHz 21.2 Gbps effective



Memory	
Memory Size:	24 GB
Memory Type:	GDDR6X
Memory Bus:	384 bit
Bandwidth:	1,018 GB/s

https://www.techpowerup.com/gpu-specs/geforce-rtx-4090.c3889

CPU vs. GPU

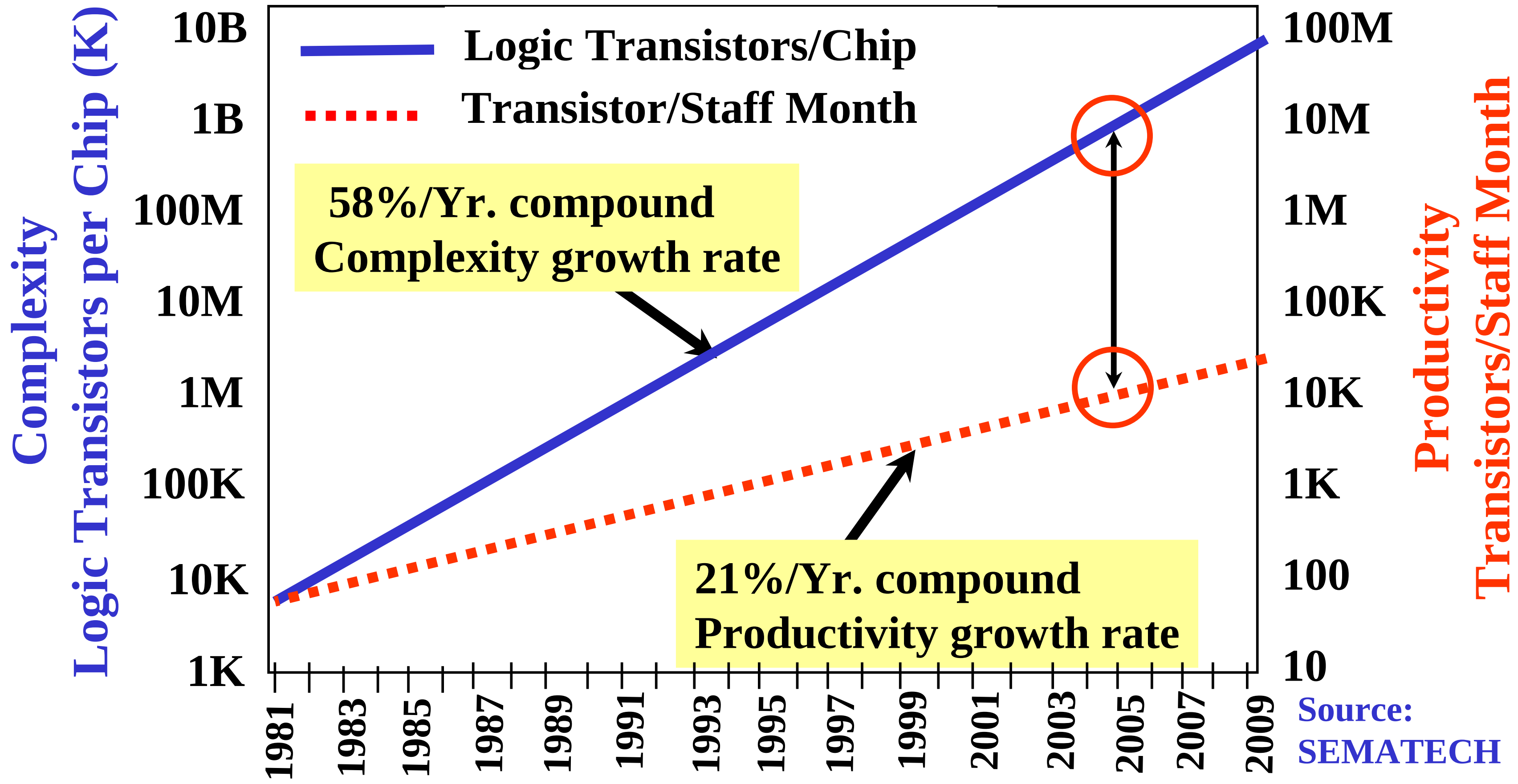


source: UCI CS295

Designer Productivity Gap

Recruiting more engineers would help to reduce gap

However, using more advanced fabrication would not help.



Source: SEMATECH

SoC designs today are complex, characterized by more and more IPs being integrated on a single chip, and a shrinking time-to-market

Memory Addressing

SoC Multiprocessors

- Each processor executes completely independently
- Share memory and execute separate program tasks (MIMD)
- synchronization is challenging
- cache coherency, memory consistency

CPU and DSP process different tasks, but share memory.

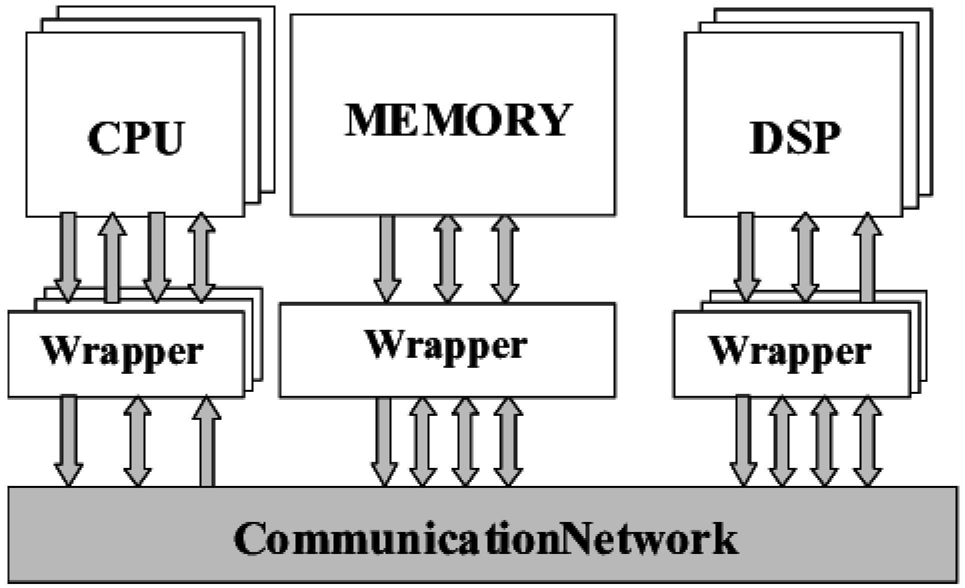
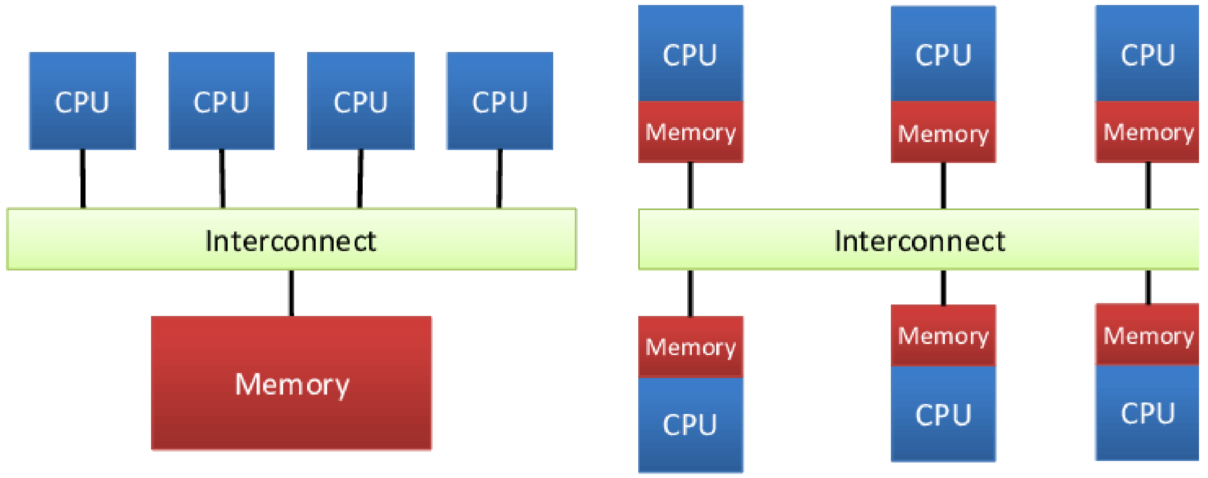


Figure: https://www.researchgate.net/figure/A-typical-Multiprocessor-SoC_fig1_2551984

MPSoC Memory Addressing Space

- Needs synchronization mechanism, to ensure consistency.

- Distributed memory – message passing
  - Easier to implement
  - Programming such processor is harder, applications need to be tailored to support the architecture
- Shared memory – communication through the memory
  - Harder to implement (due to coherence) ex. More than one CPU try to adjust same memory address.
  - Easier to program, can take advantage of any communication paradigm



(a) Shared Memory (b) Distributed Memory

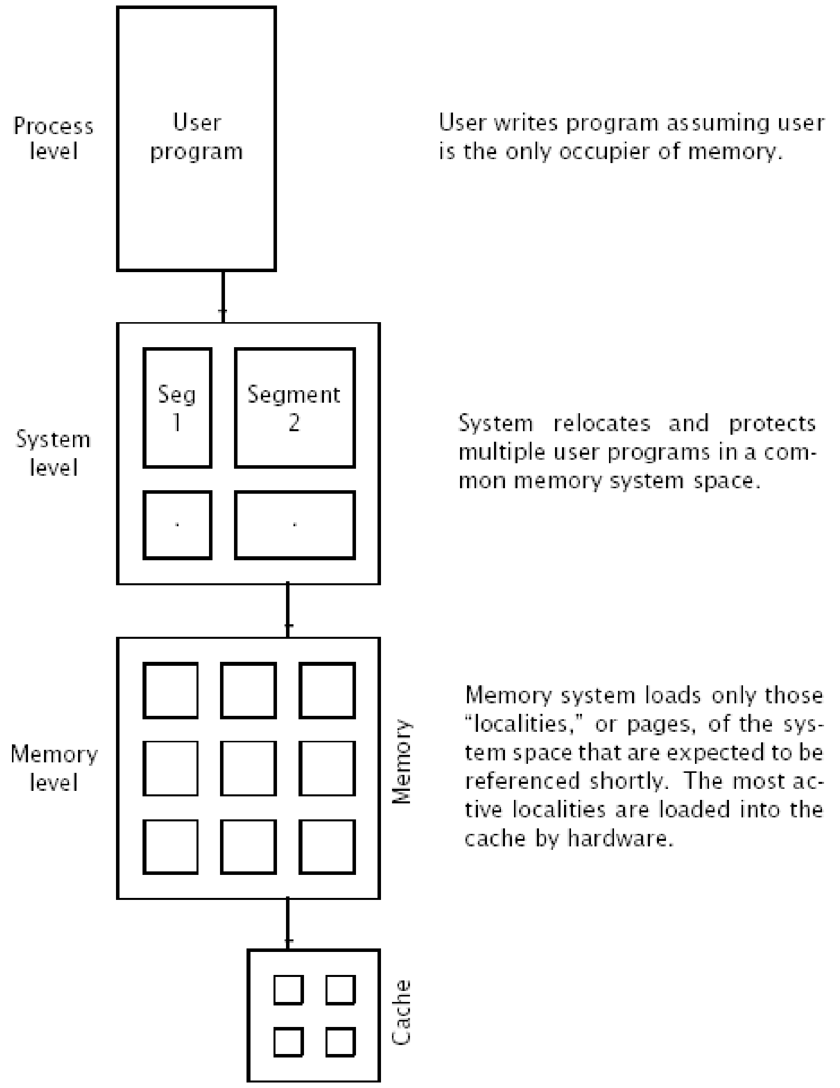
source: https://www.researchgate.net/figure/Memory-Organization-a-Shared-Memory-b-Distributed-Memory_fig1_302245489

SoC Memory and addressing

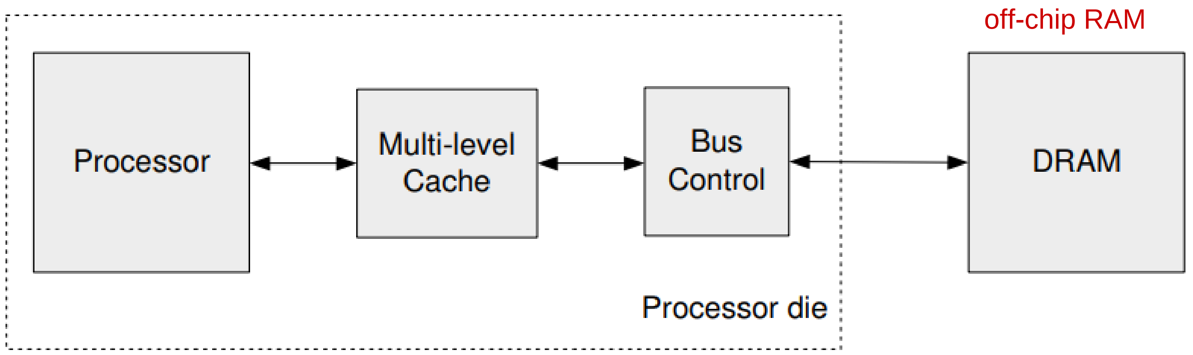
- Many SOC memory designs use simple embedded memory
  - a single level cache
  - real (rather than virtual) addressing / Normally don't use Virtual Memory as the core is small.
- SOC becomes more complex
  - their designs are expected to use more complex memory and addressing configurations

Three levels of addressing

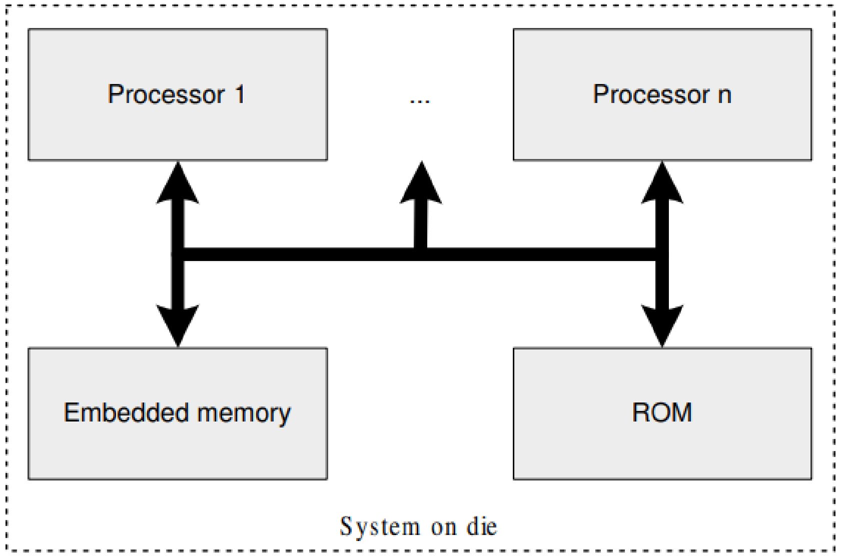
Typically, SoC involves one or two levels.


Process level
User program
User writes program assuming user is the only occupier of memory.
System level
Seg 1
Segment 2
System relocates and protects multiple user programs in a common memory system space.
Memory level
Memory
Memory system loads only those "localities," or pages, of the system space that are expected to be referenced shortly. The most active localities are loaded into the cache by hardware.
Cache

SoC Memory System



off-die memory



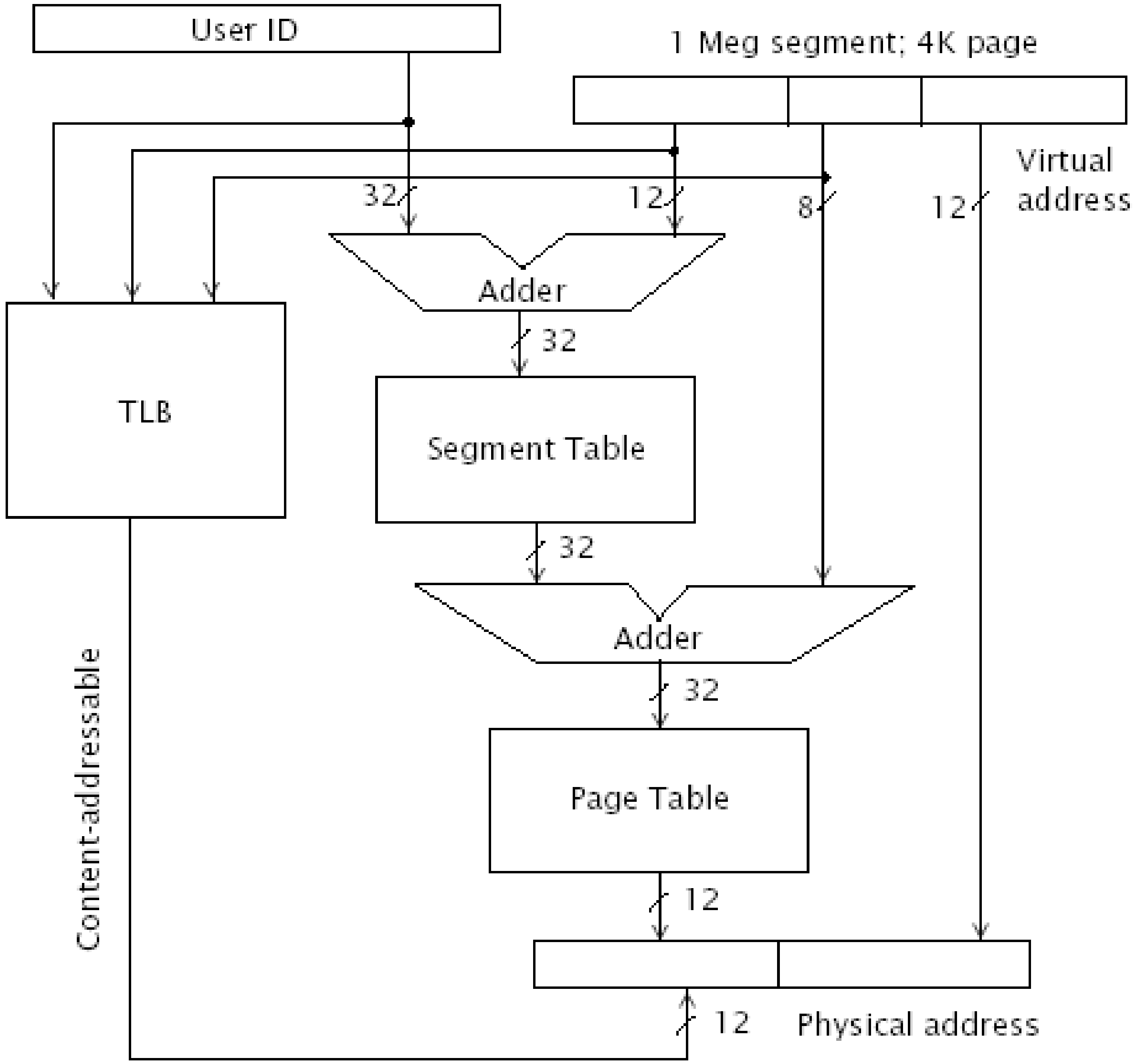
Many cases.

integrated memory SOC

User view of memory: addressing

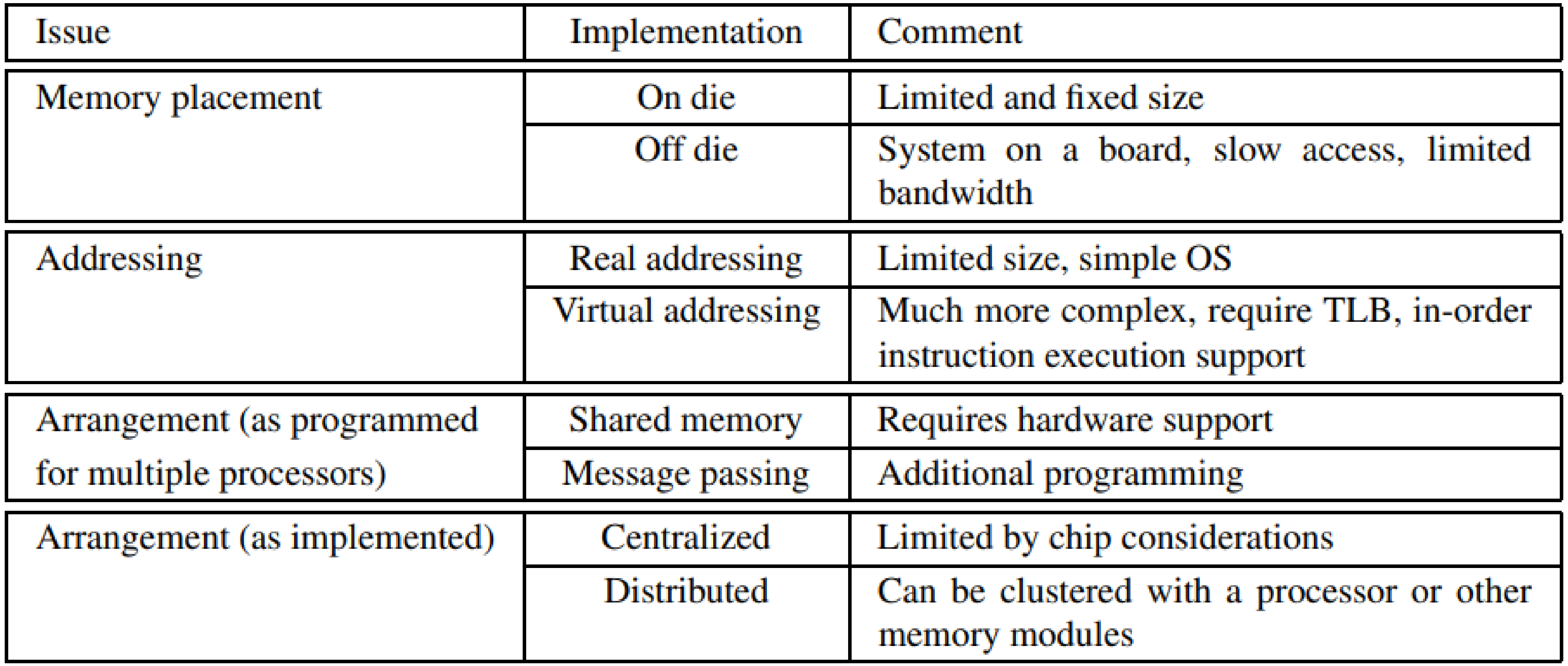
- a program: process address (offset + base + index)
  - ***virtual address:*** process address + process id
- a process: assigned a segment base and bound
  - ***system address:*** segment base + process address
- pages: active localities in main/real memory
  - virtual address: translated by table lookup to ***real address***
  - page miss: virtual pages not in page table
- **TLB** (translation look-aside buffer): recent translations
  - TLB entry: corresponding real and (virtual, id) address
- a few hashed virtual address bits address **TLB** entries
  - if virtual, id = **TLB** (virtual, id) then use translation

The TLB and The MMU



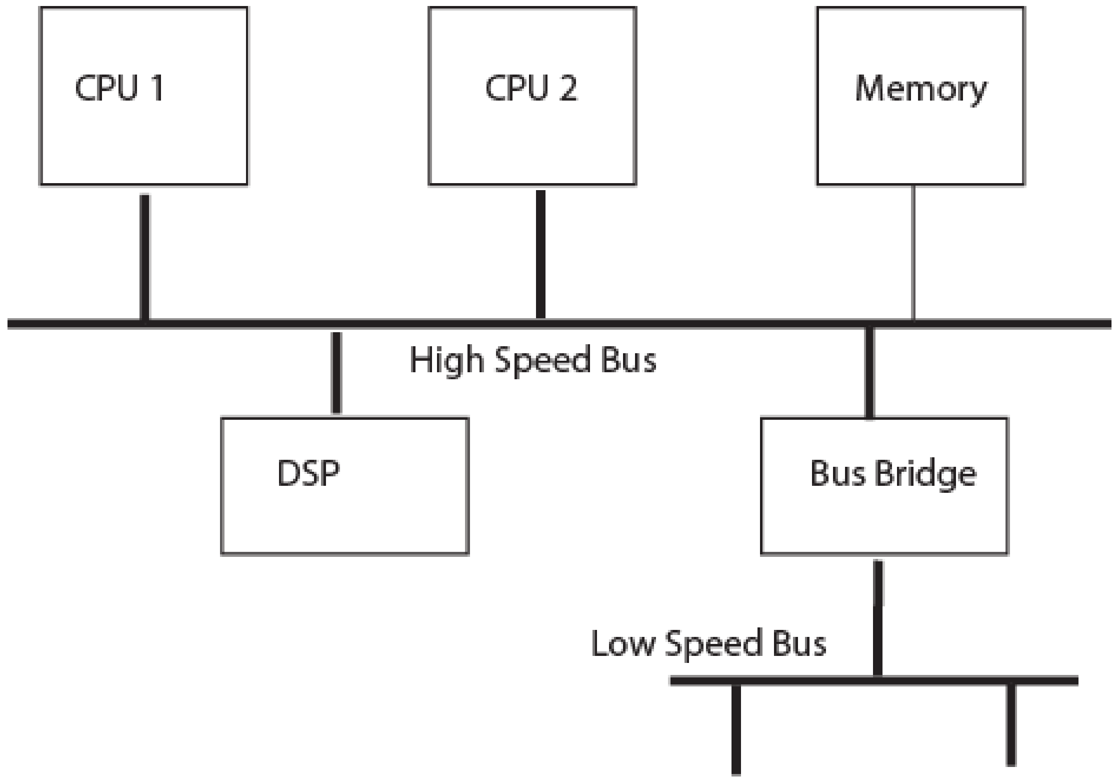
SoC memory considerations

Depends on budget we have.



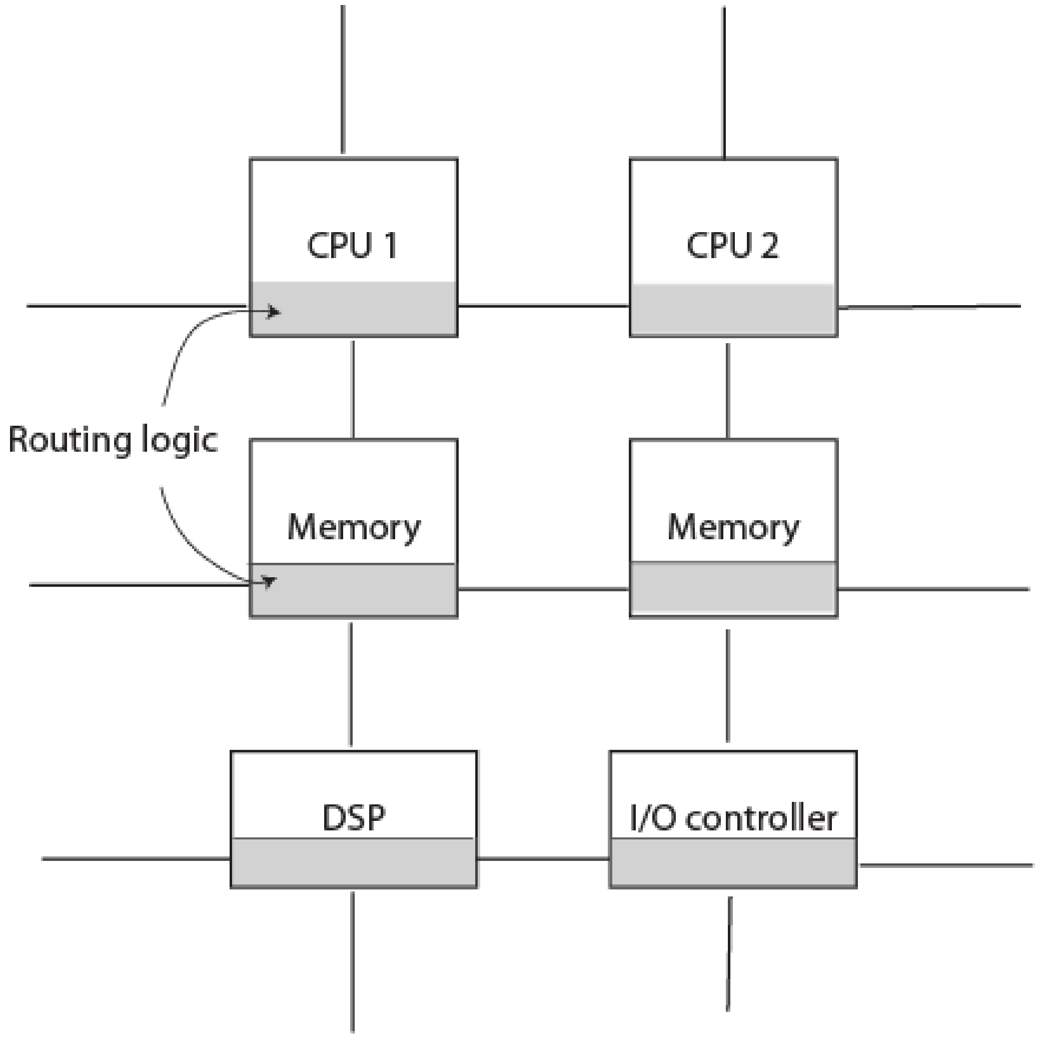
Issue	Implementation	Comment
Memory placement	On die	Limited and fixed size
	Off die	System on a board, slow access, limited bandwidth
Addressing	Real addressing	Limited size, simple OS
	Virtual addressing	Much more complex, require TLB, in-order instruction execution support
Arrangement (as programmed for multiple processors)	Shared memory	Requires hardware support
	Message passing	Additional programming
Arrangement (as implemented)	Centralized	Limited by chip considerations
	Distributed	Can be clustered with a processor or other memory modules

Bus based SOC



Network on a Chip

Memory treated as regular IP.



SOC design approach

- understand application (compiler, OS, memory and real time constrains) application requirements
- select initial die area, power, performance targets; select initial processors, memory, interconnect
- assume target processor and interconnect performance, design and evaluate memory
- evaluate and redesign processors with memory
- design interconnect to support processors and memory
- repeat and iterate to optimize

Processor optimization example

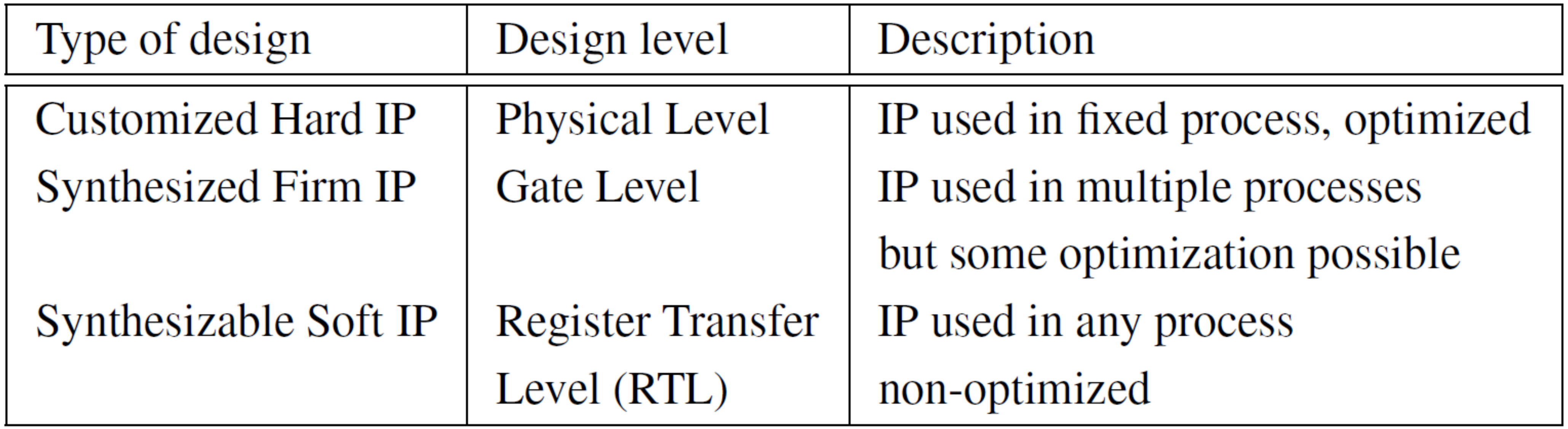
Based on application target

- given embedded ARM processor
  - in an SOC chip
- 1 IALU vs 2 IALU vs 3 IALU vs 4 IALU
  - *instructions per cycle?*
- 16k L1 instruction cache vs 32k L1 i-cache
  - *how much improvement? less power?*
- branch predictor: taken vs not-taken
  - *misprediction rate?*
- aim: explore this large design space

To figure out good design options that meet application requirements.

Design Reuse

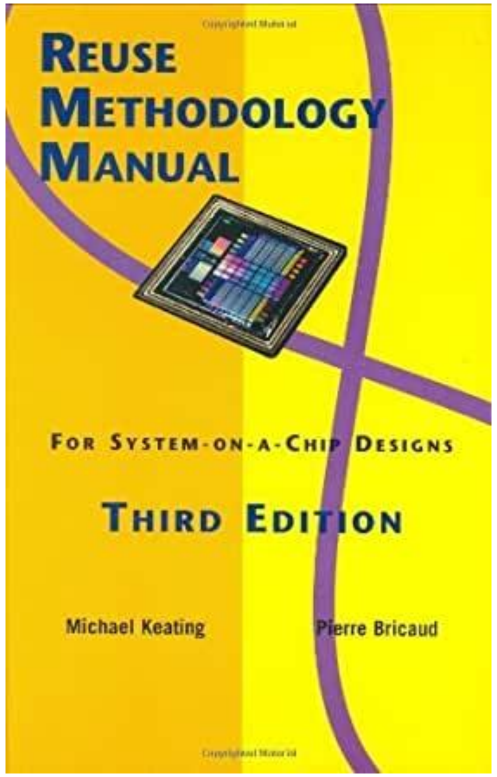
Reduce complexity: use IP



Type of design	Design level	Description
Customized Hard IP	Physical Level	IP used in fixed process, optimized
Synthesized Firm IP	Gate Level	IP used in multiple processes but some optimization possible
Synthesizable Soft IP	Register Transfer Level (RTL)	IP used in any process non-optimized

Some history

- 1999 Reuse methodology manual
- A solution practiced today for state of the art SoC design
- New business model emerges
  - IP provider / integrator
- Top IP providers
  - ARM, Cadence, Synopsys…



http://www.ime.cas.cn/icac/learning/learning_3/201907/P020190716574161163126.pdf

Design for use (first)

- To be reusable, useable first
- Being useable
  - Documentation
  - Good and clean code
  - Thorough commenting
  - Well developed verification environments
  - robust scripts
  - ecosystems
  - …

Then design for reuse

- Designed to solve a general problem
  - configurations and parameters
- Designed for use in multiple technology nodes
- Designed with standards-based interfaces
- Designed with complete verification process
- Design verified with high level of confidence
- Design with fully documented in terms of applications and limitations

IP Reuse Challenges

- Why should I design for reuse?
- Should I plan all my design for reuse?
- How to design reuse?
- Who should do it?
- When should I do it?
- ….

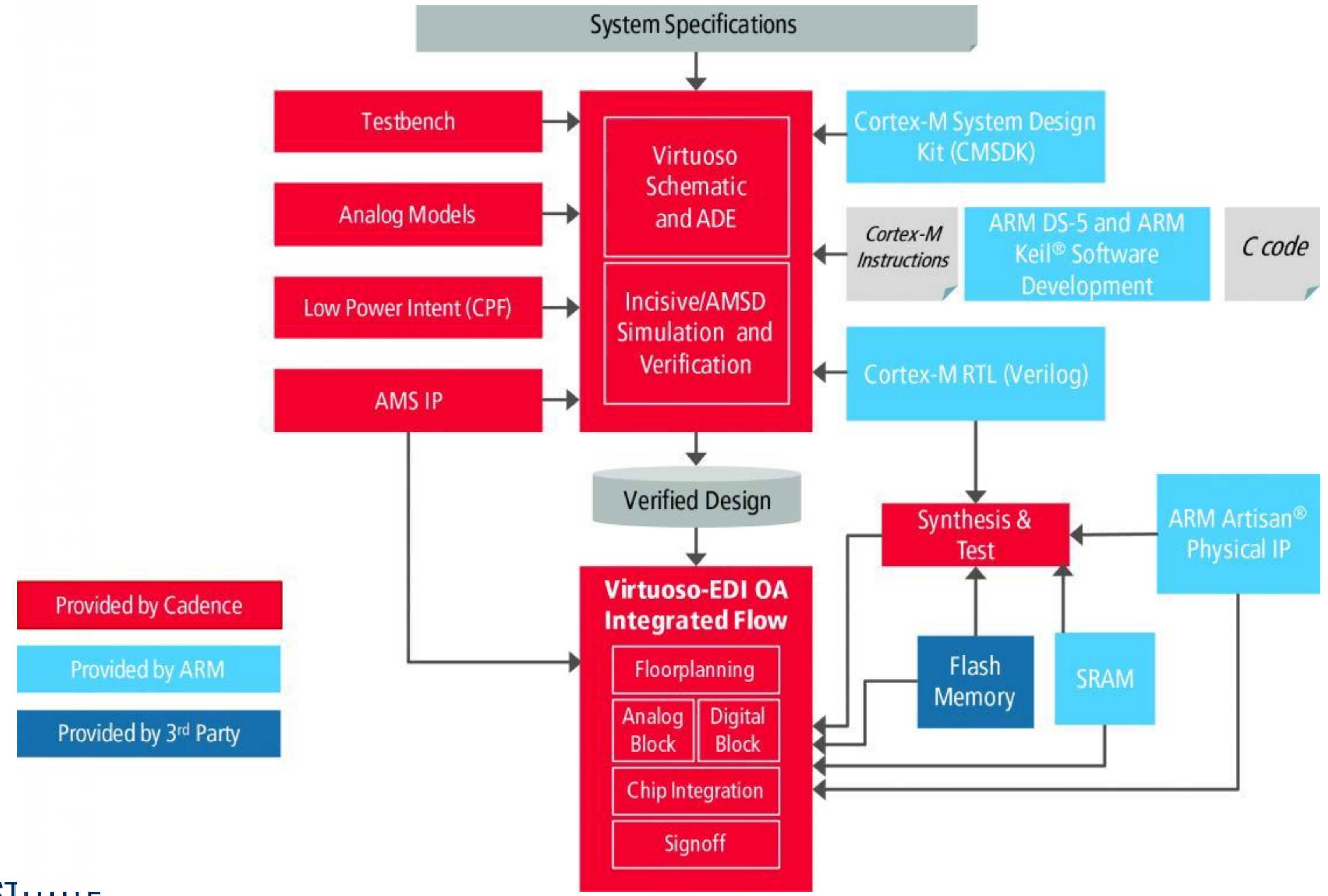
IP Reuse Challenges

- Reuse has its own cost, and it is expensive
  - learning curve, coding guidelines, documents, verification… + License fee
  - Explicit design reuse requires a dedicated team
  - 10x or an order of magnitude higher cost
- But it is rewarding...
  - 2-3x benefit for implicit reuse
  - 10x-100x productivity gain in successive design for explicit reuse
  - Coding style – a matter of habit
  - Documentation – an one-time experience
  - Verification – Any design requires verification
- Whether to adopt reuse is a managerial and cultural issue!

Paradigm Shift

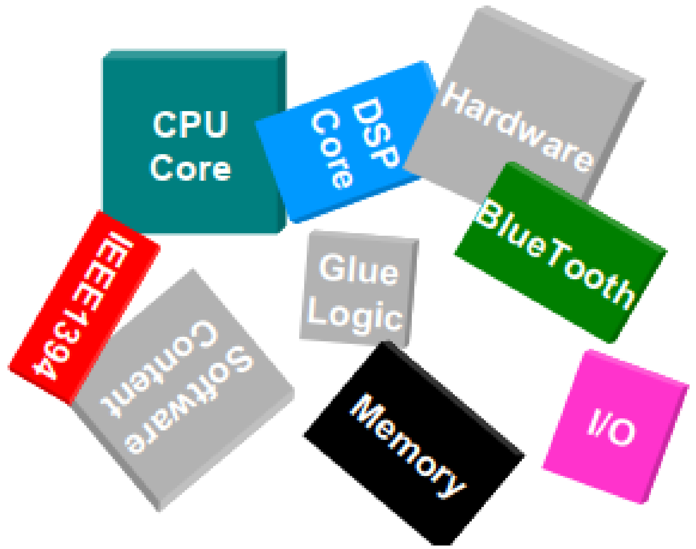
- Conventional flow
  - System/design house for RTL design, synthesis
  - ASIC vendor for physical design
- SoC design is too complicated to be handled in-house solely
  - system/design house provides H/W spec and only focuses on valued-added S/W and application
  - ASIC vendor offers IPs and integration service

Example: Design flow with IP integration

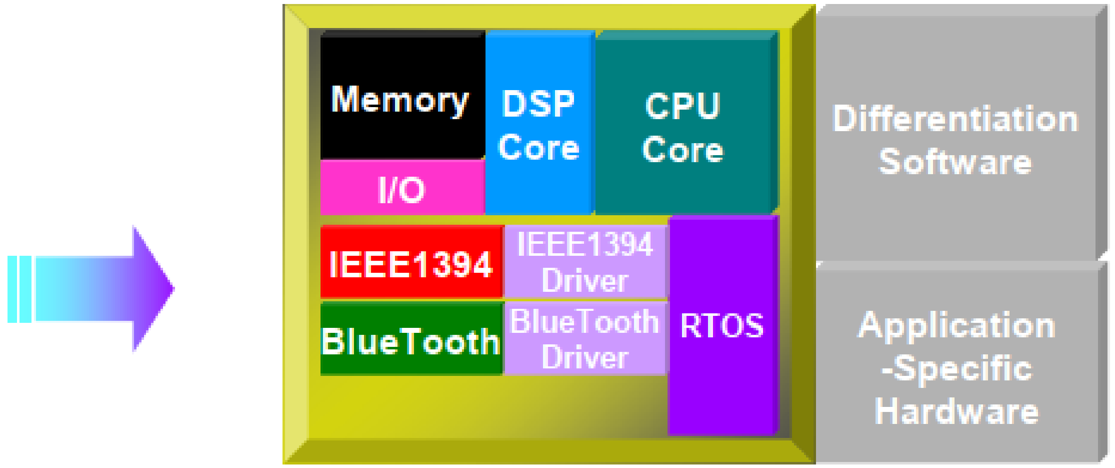


source: https://semiwiki.com/ip/arm/2945-meeting-the-challenges-of-designing-internet-of-things-socs-with-the-right-design-flow-and-ip/

Platform-based Design



Block-Based Design

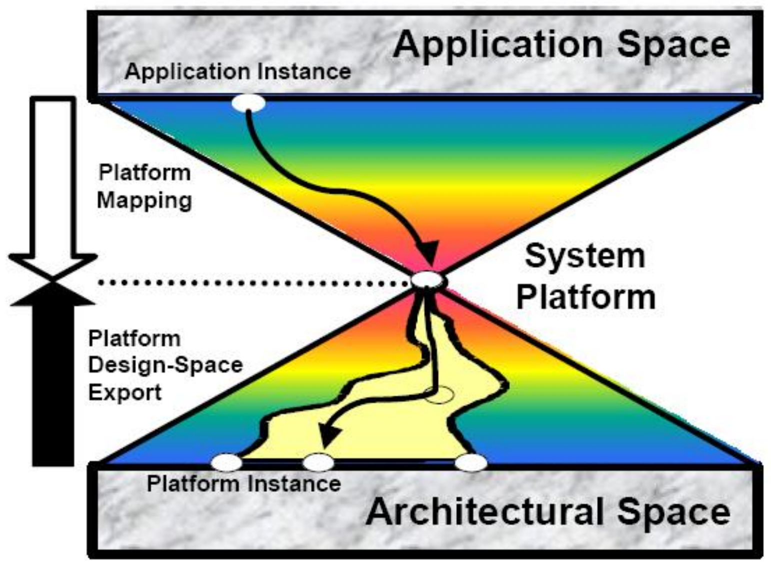


Platform-Based Design

source: http://twins.ee.nctu.edu.tw/courses/soclab_04/index.html

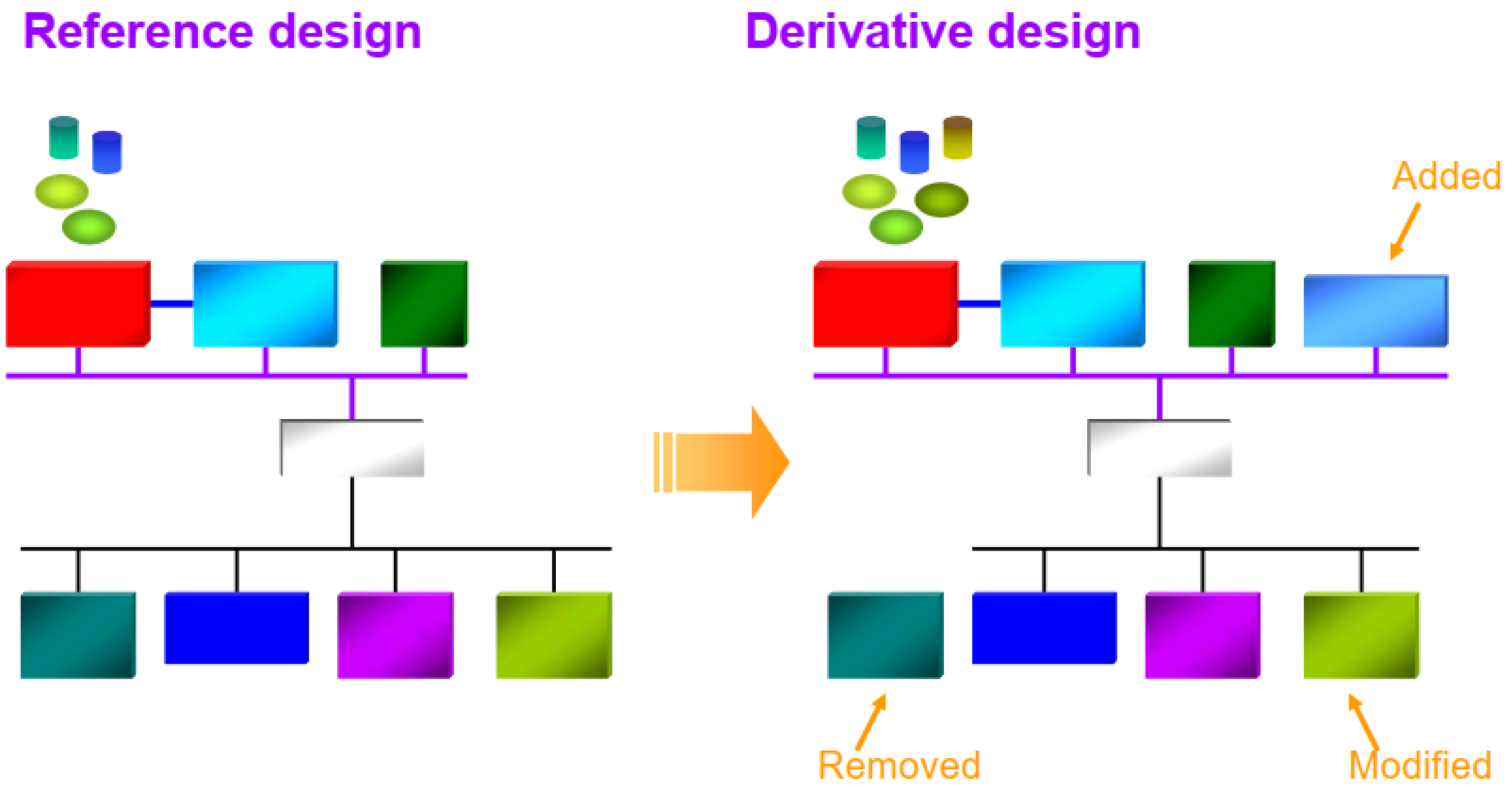
Coping with SoC Complexity

- Platform based Design
  - Design paradigm which promotes IP based design and reuse
  - A *meet-in-the-middle* approach
    - maps application to a customizable design consisting of HW, SW IPs
  - Platforms use IPs; but will soon become the IPs of the future
    - e.g. Phillips Nexperia, ARM PrimeXsys platform etc.



Decouples the application development process from the architectural implementation process.

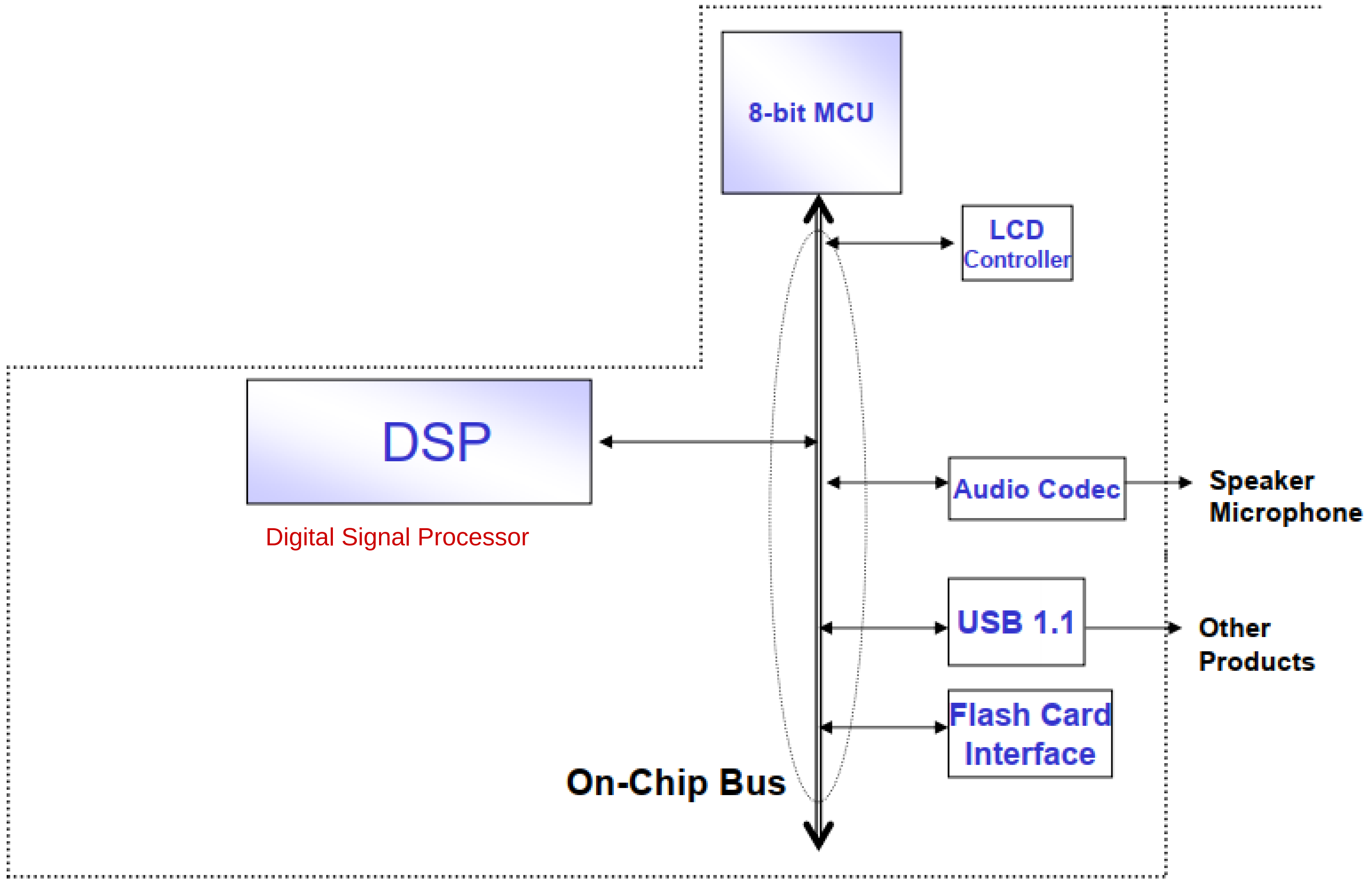
Platform-based Design



source: http://twins.ee.nctu.edu.tw/courses/soclab_04/index.html

Example: Multimedia Platform

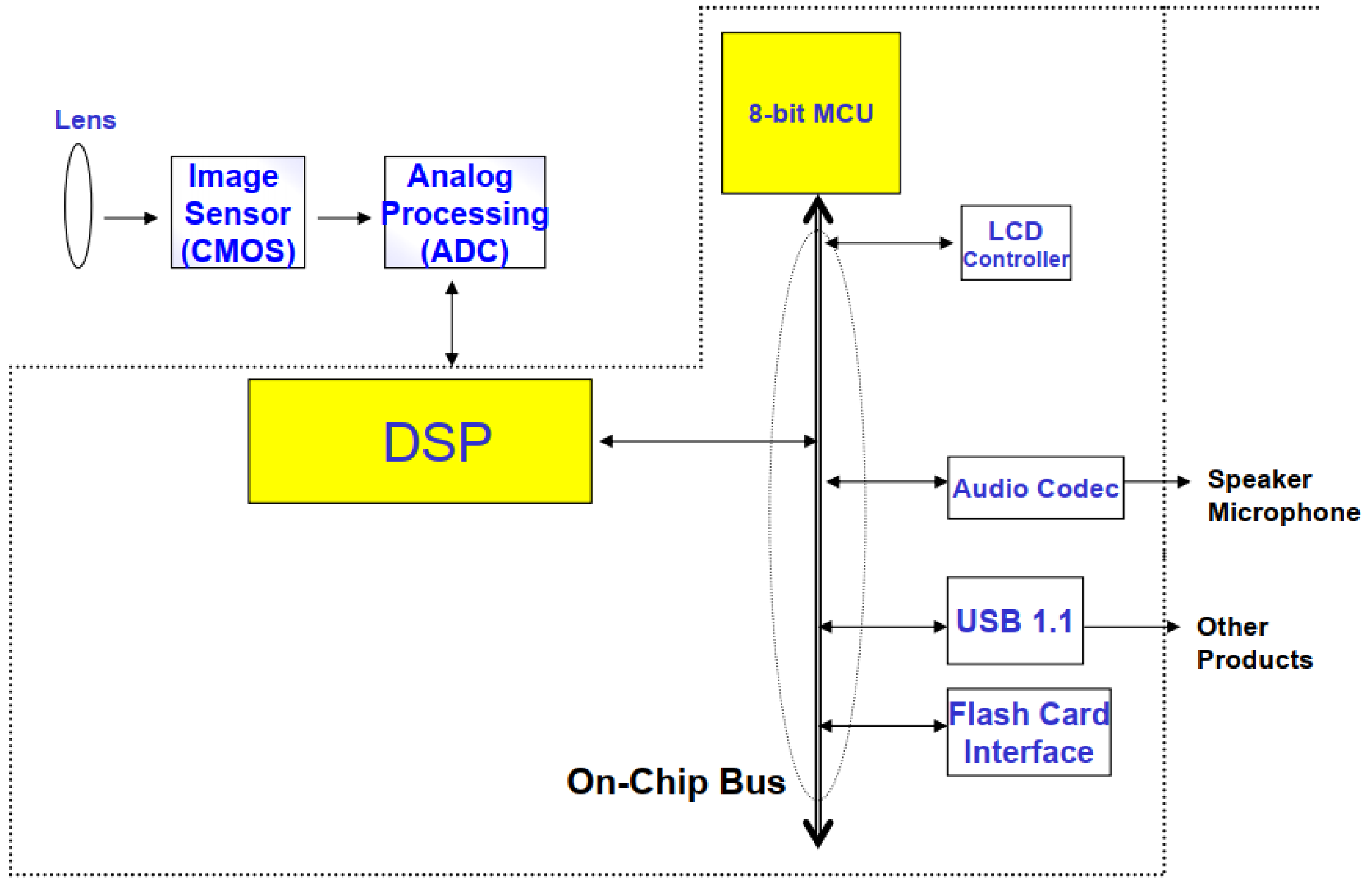
- MP3



source: http://twins.ee.nctu.edu.tw/courses/soclab_04/index.html

Example: Multimedia Platform

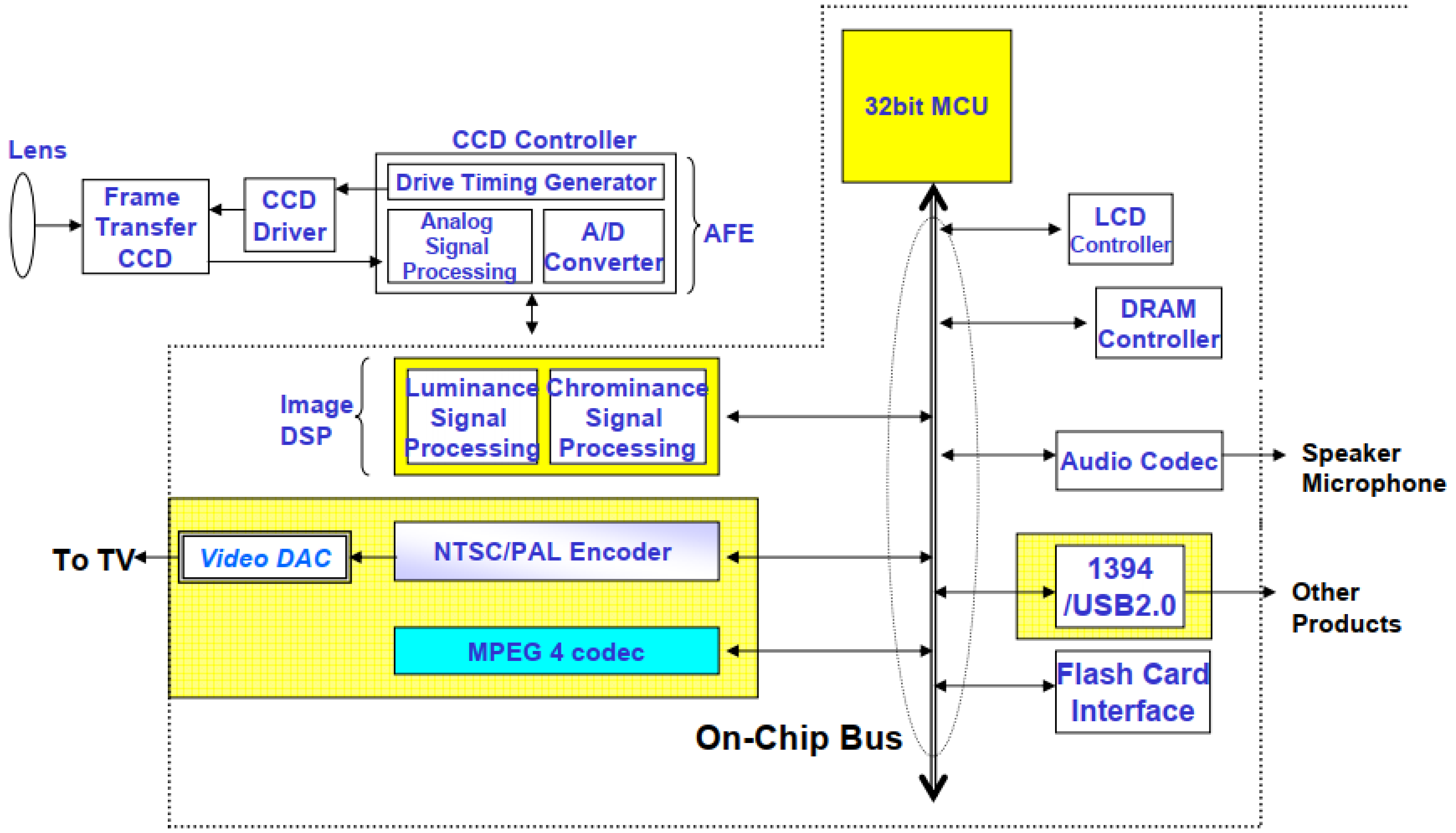
- PC Camera



source: http://twins.ee.nctu.edu.tw/courses/soclab_04/index.html

Example: Multimedia Platform

- Video Camera

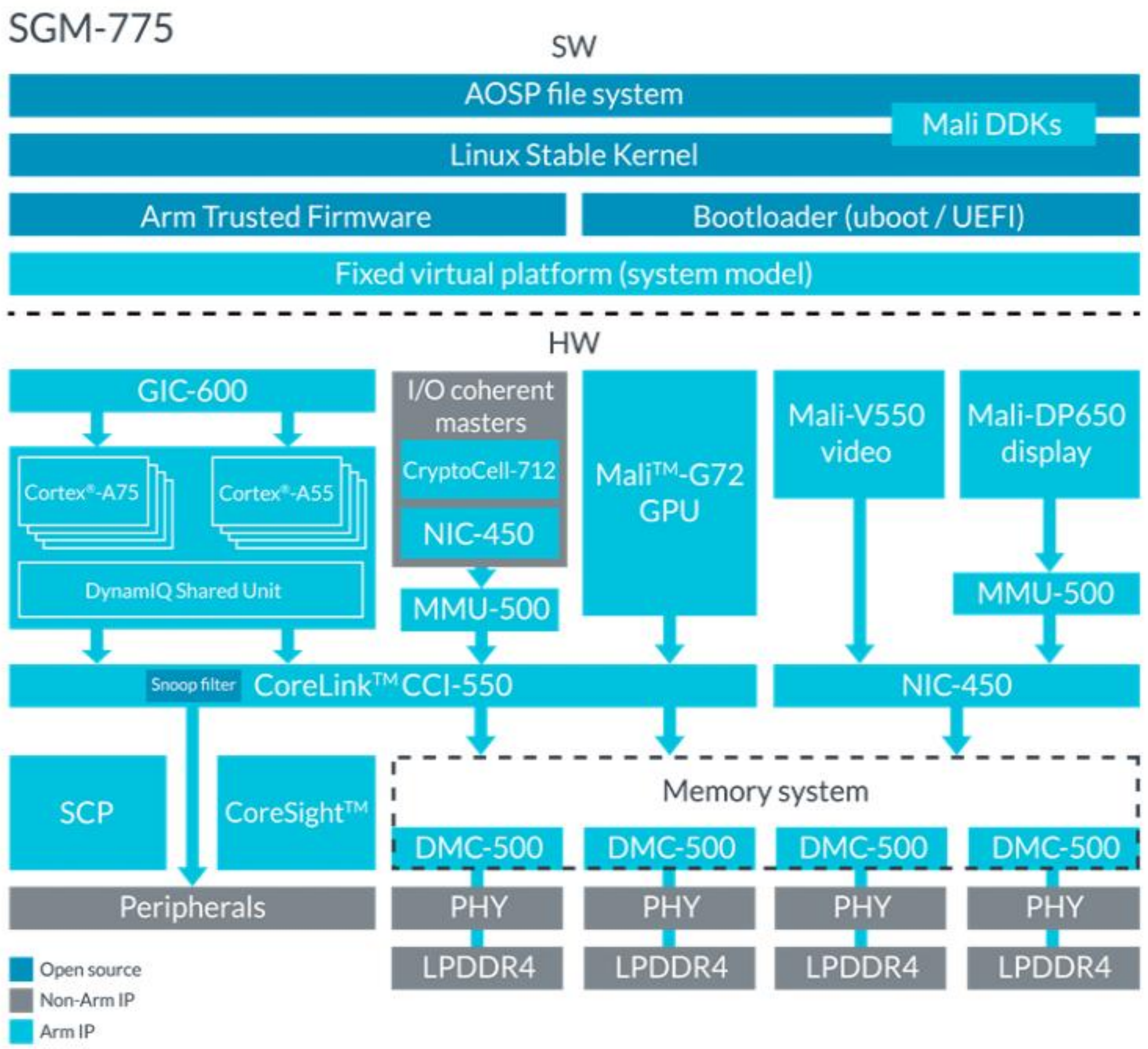


Platform-based integration

- A fully defined architecture with

  At least...
  - Bus structure
  - Clocking/power distribution
  - OS
- A collection of IP blocks
- Architecture reuse
- A result for leveraging trade-offs involving reusability, production cost and performance optimization.

Example: ARM System IP



source: ARM

Ingredients of A Platform

- Cores
  - Processor IP
  - Bus/Interconnection
  - Peripheral IP
  - Application specific IP
- Software
  - Drivers
  - Firmware
  - (Real-time) OS
  - Application software/libraries
- Validation
  - HW/SW Co-Verification
  - Compliance test suites
- Prototyping
  - HW emulation
  - FPGA based prototyping
  - Platform prototypes (i.e. dedicated prototyping devices)
  - SW prototyping

source: http://twins.ee.nctu.edu.tw/courses/soclab_04/index.html

How to build a platform?

- pick your application domain
- pick your on-chip communication architecture and structure
- pick your Star IP (e.g. CPUs)
- Memory access
- application specific HW and SW IP
- other IP blocks that are not available “wrapped” to the on-chip communication may work with IP wrappers

Pros and Cons

- Procs
  - shorten design cycle
  - sharing of pre-verified components
  - quick derivative designs based on a basic platform
  - rapid prototyping
- Cons
  - Limited creativity due to predefined platform components
  - Differentiation more difficult to achieve, need to be primarily in application software

Reduce complexity: reconfig. tech.

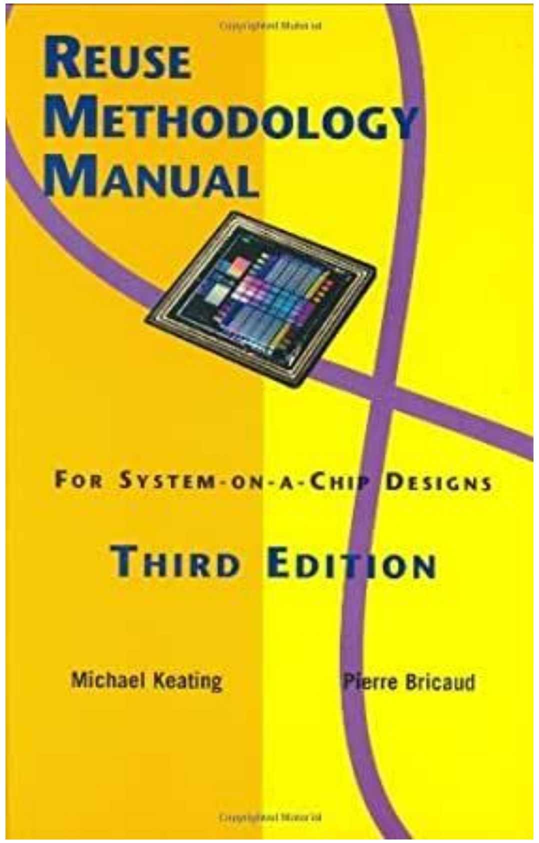
- reconfigurable technology: no fabrication costs
  - lower non-recurring engineering (NRE) costs
- reconfigurable design: faster and cheaper
  - improve time-to-market
- reconfigurability
  - in-system upgrade: improve time-in-market
  - run-time adaptation: respond to run-time conditions
  - compile-time reconfiguration: retarget accelerator
- overhead: performance, area, energy efficiency
  - less effective than ASIC (application-specific IC)

Takeaways

- SoC design requires a systematic approach, designers need to understand all components: processors, memory, interconnect, and its applications targets
- SOC economics heavily dependent on:
  - costs: initial design, marginal production
  - volume: applicability, lifetime
- reducing design complexity
  - Reuse Intellectual Property (IP)
  - reconfigurable technology

Action Items

- Lab #1 is due!
- Reading Materials
  - Ch. 1.5 – 1.9, literature on canvas



Where are we Heading?

- SoC Interconnect Architectures

Acknowledgement

Slides in this topic are inspired in part by material developed and copyright by:

- Dr. Wayne Luk (Imperial College)
- Dr. Gul N. Khan (Ryerson University)
- Dr. André DeHon (UPenn)
- Dr. Andreas Gerstlauer (UT Austin)
- Dr. Anand Raghunathan (Purdue)