Topic 2

SoC System Approach I

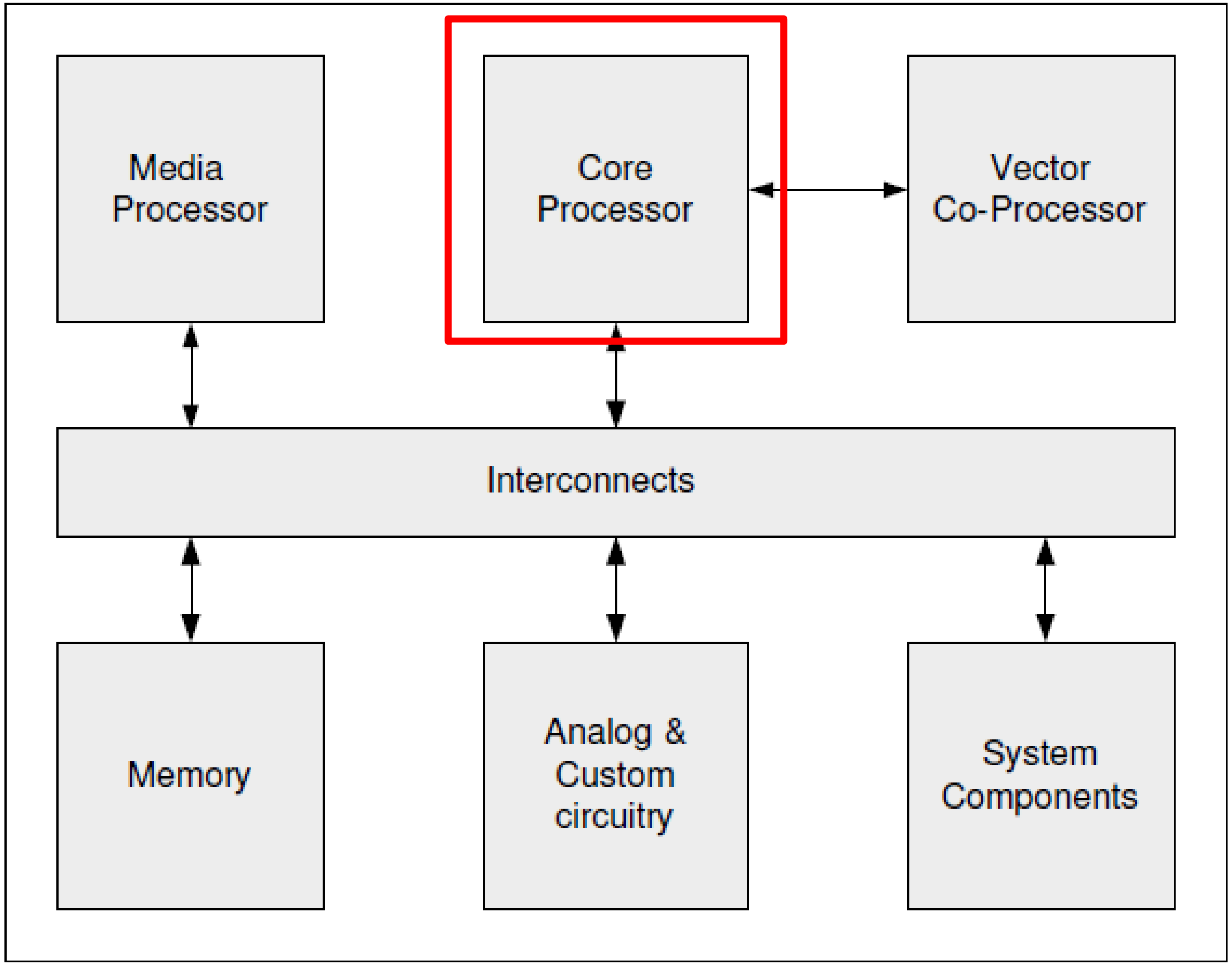
Xinfei Guo
xinfei.guo@sjtu.edu.cn

September 25th, 2024

T2 learning goals

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

Basic system-on-chip model



SoC vs. processors on chip

“What distinguishes a system on a chip from the conventional general purpose computer plus memory on a board is the specific nature of the design target. The application is assumed to be known and specified, so that the elements of the system can be selected, sized and evaluated during the design process.” – From the textbook

SoC vs processors on chip

- With lots of transistors, designs move in 2 ways:
  - Complete system on a chip
  - Multi-core processors with lots of cache

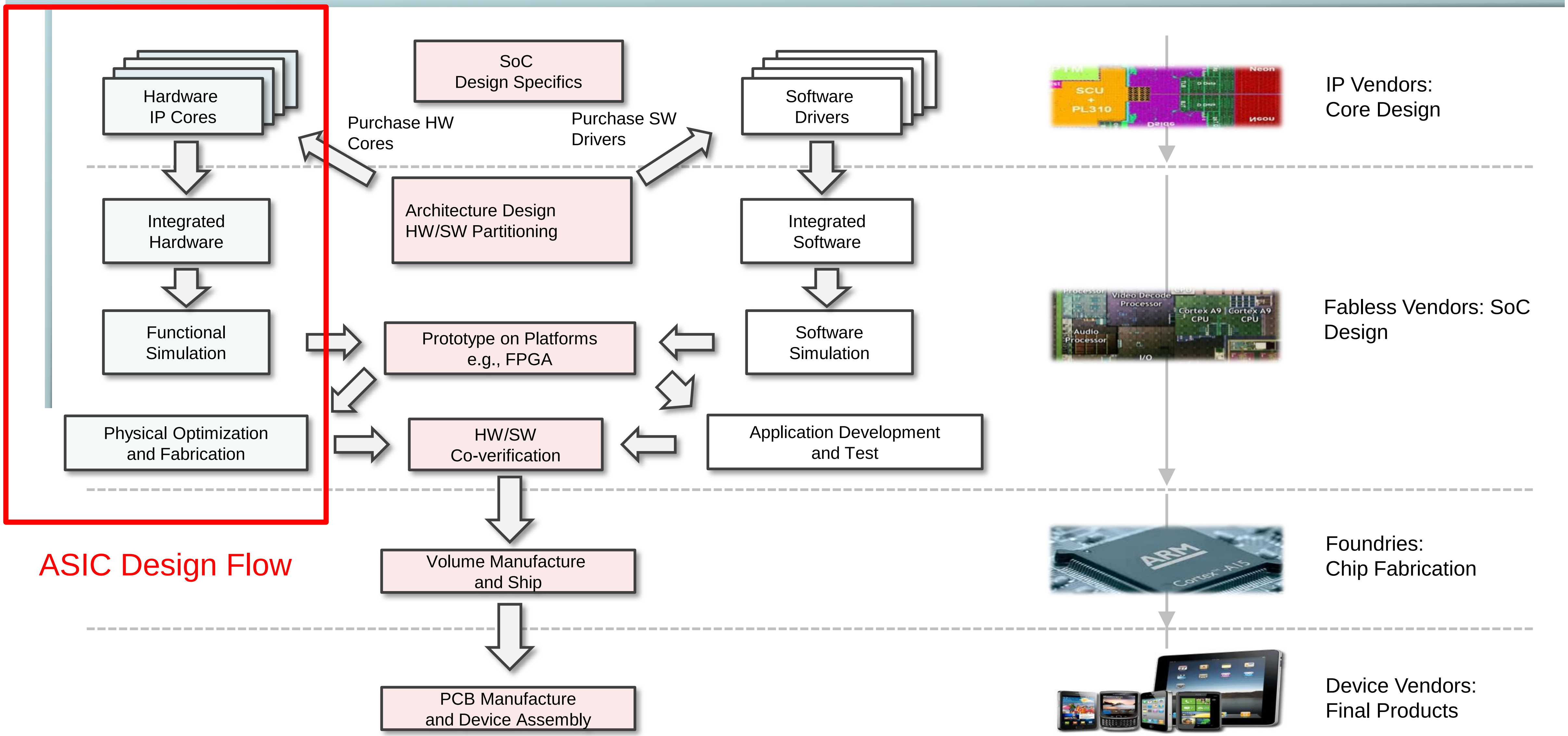
	System on chip	*Processors on chip*
processor	multiple, simple, heterogeneous	few, complex, homogeneous
cache	one level, small	2-3 levels, extensive
memory	embedded, on chip	very large, off chip
functionality	special purpose	general purpose
interconnect	wide, high bandwidth	often through cache
power, cost	both low	both high
operation	largely stand-alone	need other chips

SoC architecture and design

- System-on-chip (SOC)
  - Processors: become components in a system
- SOC covers many topics
  - Processors, cache, memory, interconnect, design tools
- Need to know
  - User view: variety of processors
  - Basic information: technology and tools
  - Processor internals: effect on performance
  - Storage: cache, embedded and external memory
  - Interconnect: buses, network-on-chip
  - Evaluation: processor, cache, memory, interconnect
  - Advanced: specialized processors, reconfiguration
  - Design productivity: system modelling, design exploration

SOC SYSTEM DESIGN FLOW OVERVIEW

SoC Design Flow Overview



Hardware IP Cores

SoC Design Specifics

Software Drivers

Purchase HW Cores

Purchase SW Drivers

Integrated Hardware

Architecture Design HW/SW Partitioning

Integrated Software

Functional Simulation

Prototype on Platforms e.g., FPGA

Software Simulation

Physical Optimization and Fabrication

HW/SW Co-verification

Application Development and Test

Volume Manufacture and Ship

PCB Manufacture and Device Assembly

ASIC Design Flow

IP Vendors: Core Design

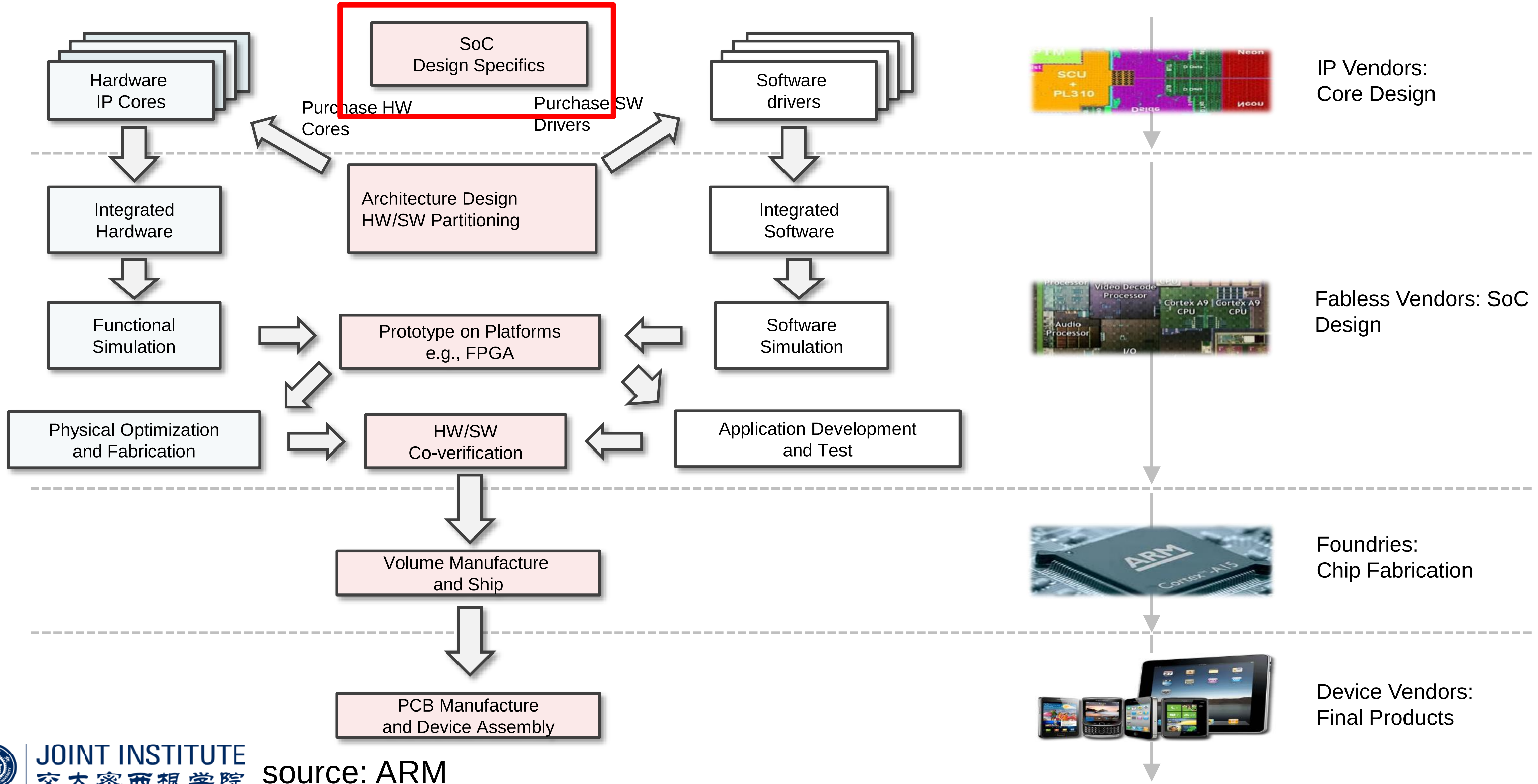
Fabless Vendors: SoC Design

Foundries: Chip Fabrication

Device Vendors: Final Products

source: ARM

SoC Design Flow



SoC
Design Specifics

Hardware
IP Cores

Purchase HW
Cores

Purchase SW
Drivers

Software
drivers

Architecture Design
HW/SW Partitioning

Integrated
Hardware

Integrated
Software

Functional
Simulation

Prototype on Platforms
e.g., FPGA

Software
Simulation

Physical Optimization
and Fabrication

HW/SW
Co-verification

Application Development
and Test

Volume Manufacture
and Ship

PCB Manufacture
and Device Assembly

IP Vendors:
Core Design

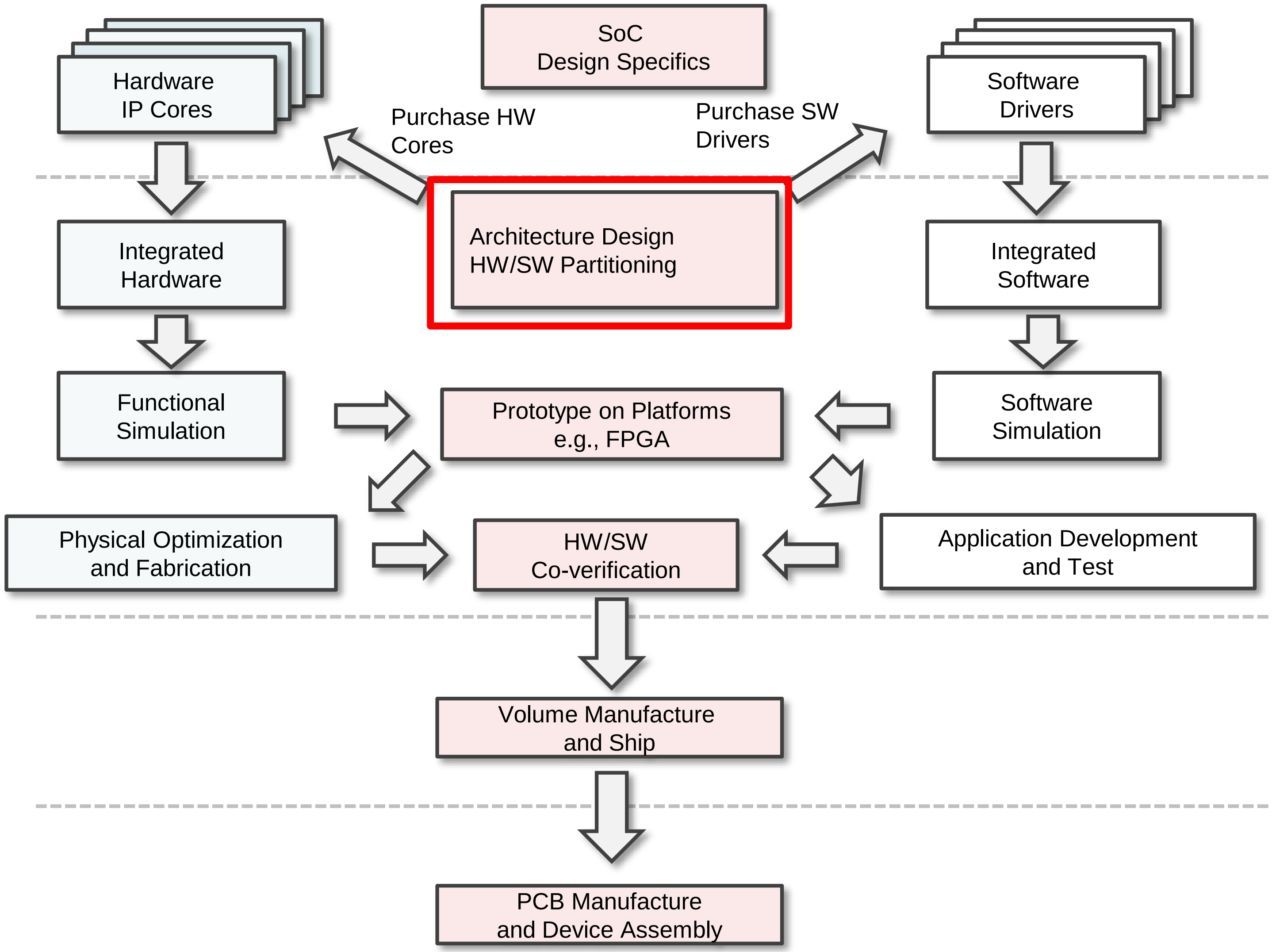
Fabless Vendors: SoC
Design

Foundries:
Chip Fabrication

Device Vendors:
Final Products

source: ARM

SoC Design Flow



SoC
Design Specifics

Hardware
IP Cores

Purchase HW
Cores

Purchase SW
Drivers

Software
Drivers

Integrated
Hardware

Architecture Design
HW/SW Partitioning

Integrated
Software

Functional
Simulation

Prototype on Platforms
e.g., FPGA

Software
Simulation

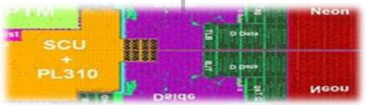
Physical Optimization
and Fabrication

HW/SW
Co-verification

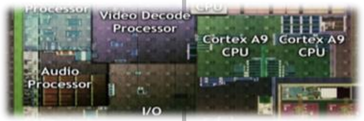
Application Development
and Test

Volume Manufacture
and Ship

PCB Manufacture
and Device Assembly



IP Vendors:
Core Design



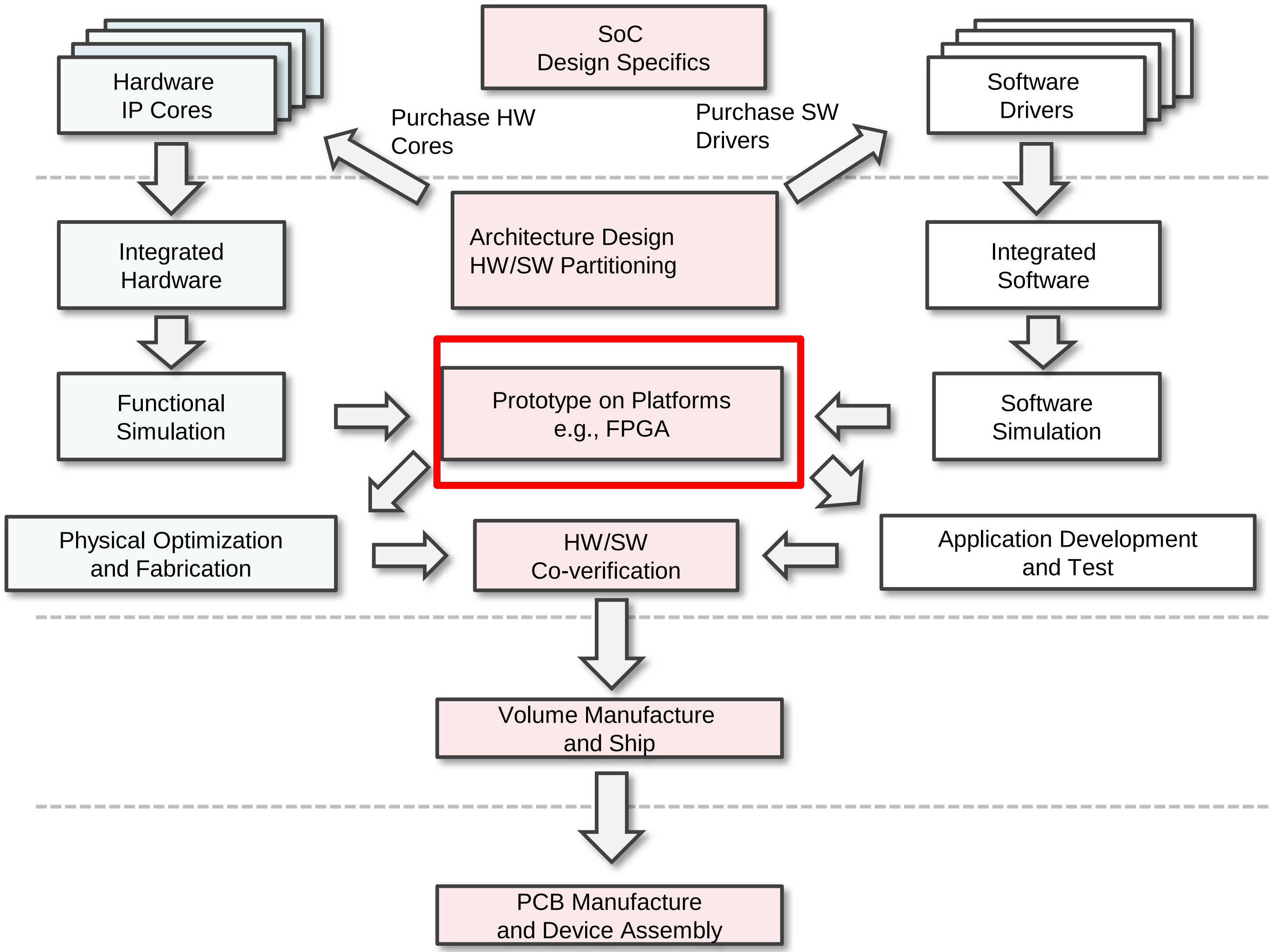
Fabless Vendors: SoC
Design

Foundries:
Chip Fabrication

Device Vendors:
Final Products

source: ARM

SoC Design Flow


SoC
Design Specifics

Hardware
IP Cores

Purchase HW
Cores

Purchase SW
Drivers

Software
Drivers

Architecture Design
HW/SW Partitioning

Integrated
Hardware

Integrated
Software

Functional
Simulation

Prototype on Platforms
e.g., FPGA

Software
Simulation

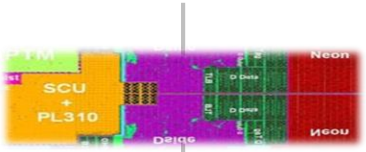
Physical Optimization
and Fabrication

HW/SW
Co-verification

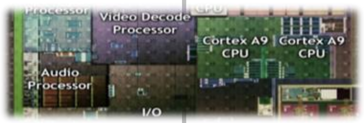
Application Development
and Test

Volume Manufacture
and Ship

PCB Manufacture
and Device Assembly



IP Vendors:
Core Design



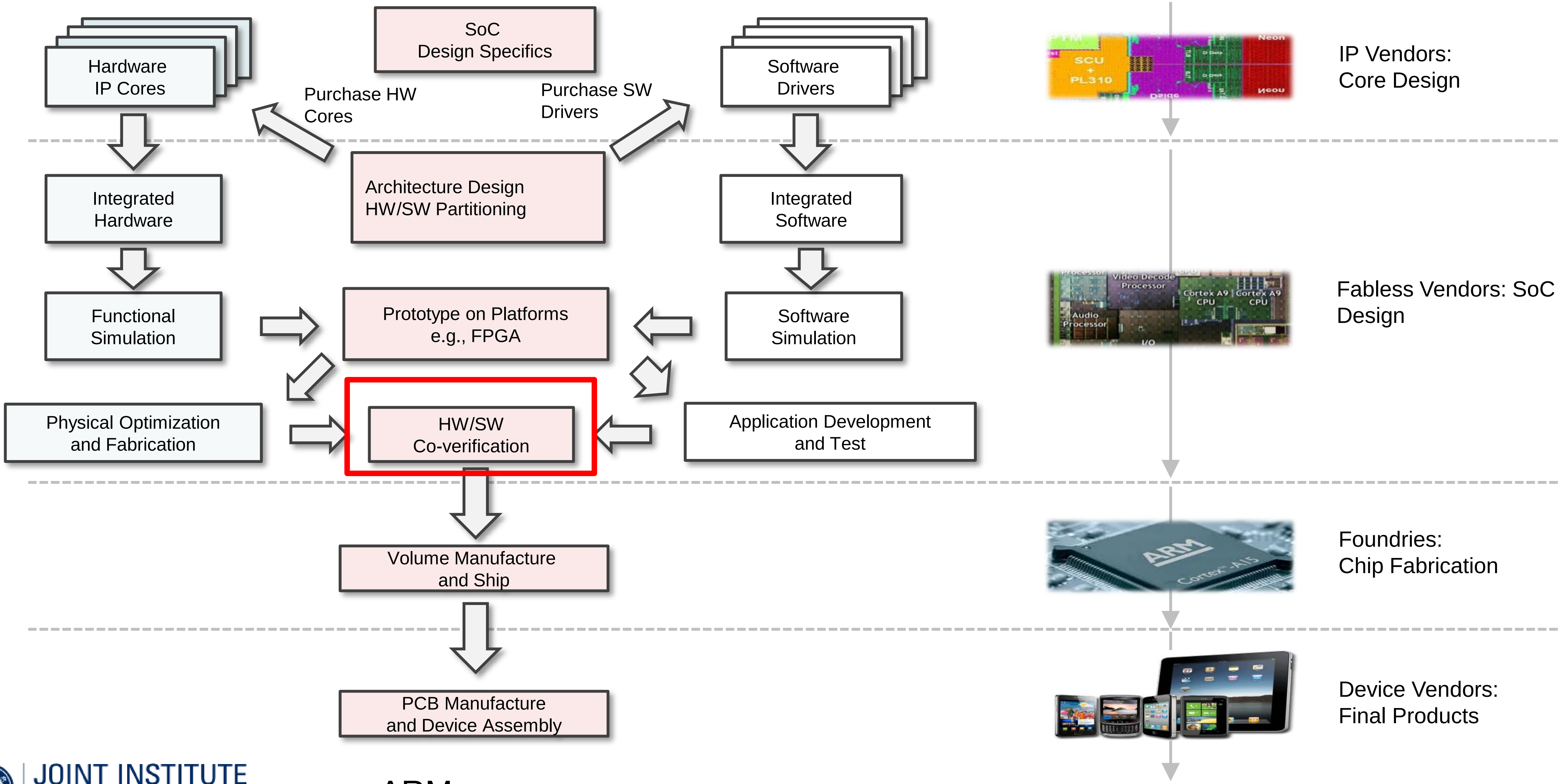
Fabless Vendors: SoC
Design

Foundries:
Chip Fabrication

Device Vendors:
Final Products

source: ARM

SoC Design Flow



SoC
Design Specifics

Hardware
IP Cores

Purchase HW
Cores

Purchase SW
Drivers

Software
Drivers

Integrated
Hardware

Architecture Design
HW/SW Partitioning

Integrated
Software

Functional
Simulation

Prototype on Platforms
e.g., FPGA

Software
Simulation

Physical Optimization
and Fabrication

HW/SW
Co-verification

Application Development
and Test

Volume Manufacture
and Ship

PCB Manufacture
and Device Assembly

IP Vendors:
Core Design

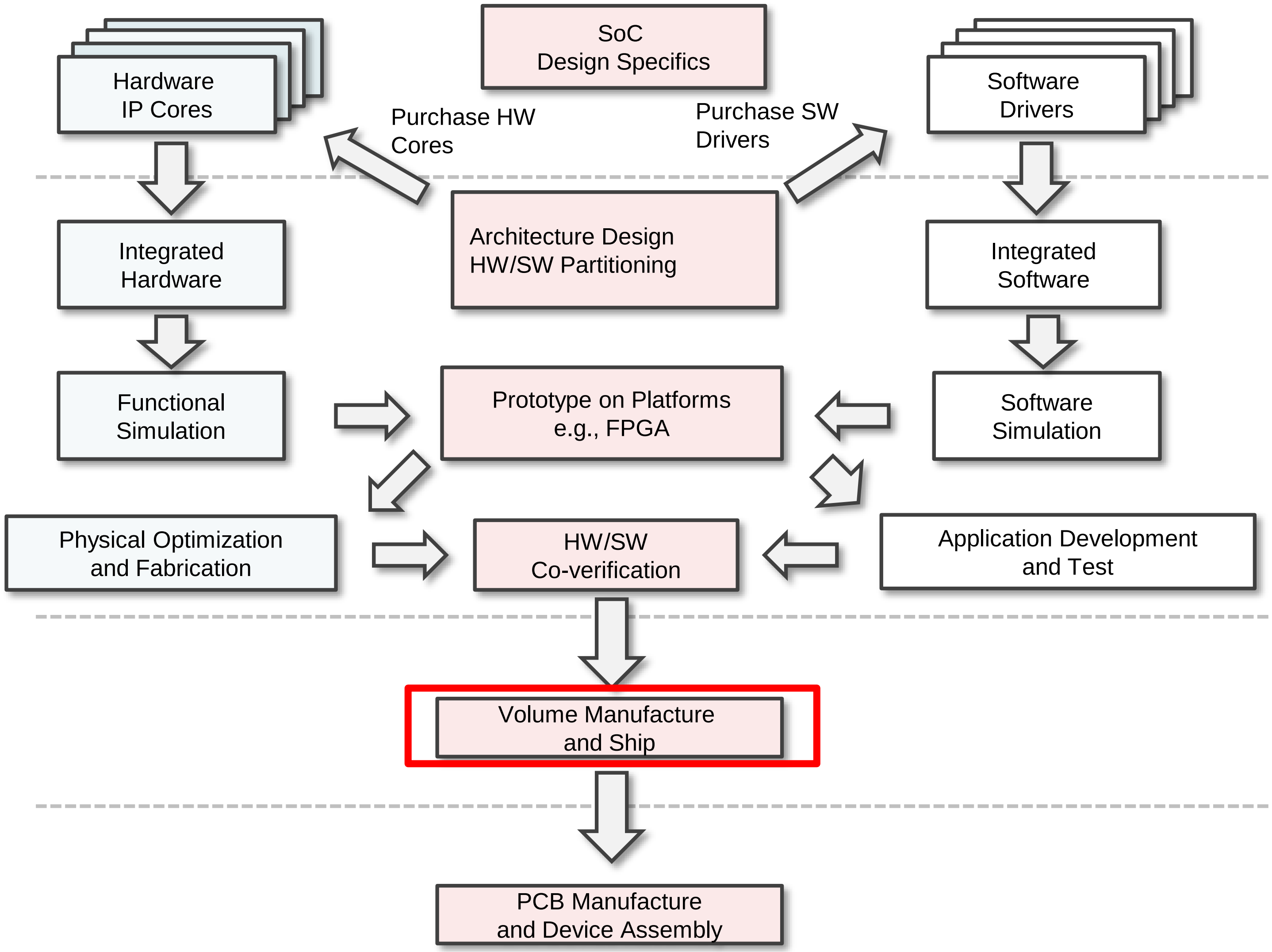
Fabless Vendors: SoC
Design

Foundries:
Chip Fabrication

Device Vendors:
Final Products

source: ARM

SoC Design Flow



SoC
Design Specifics

Hardware
IP Cores

Purchase HW
Cores

Purchase SW
Drivers

Software
Drivers

Integrated
Hardware

Architecture Design
HW/SW Partitioning

Integrated
Software

Functional
Simulation

Prototype on Platforms
e.g., FPGA

Software
Simulation

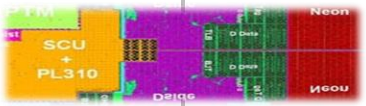
Physical Optimization
and Fabrication

HW/SW
Co-verification

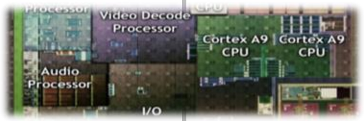
Application Development
and Test

Volume Manufacture
and Ship

PCB Manufacture
and Device Assembly



IP Vendors:
Core Design



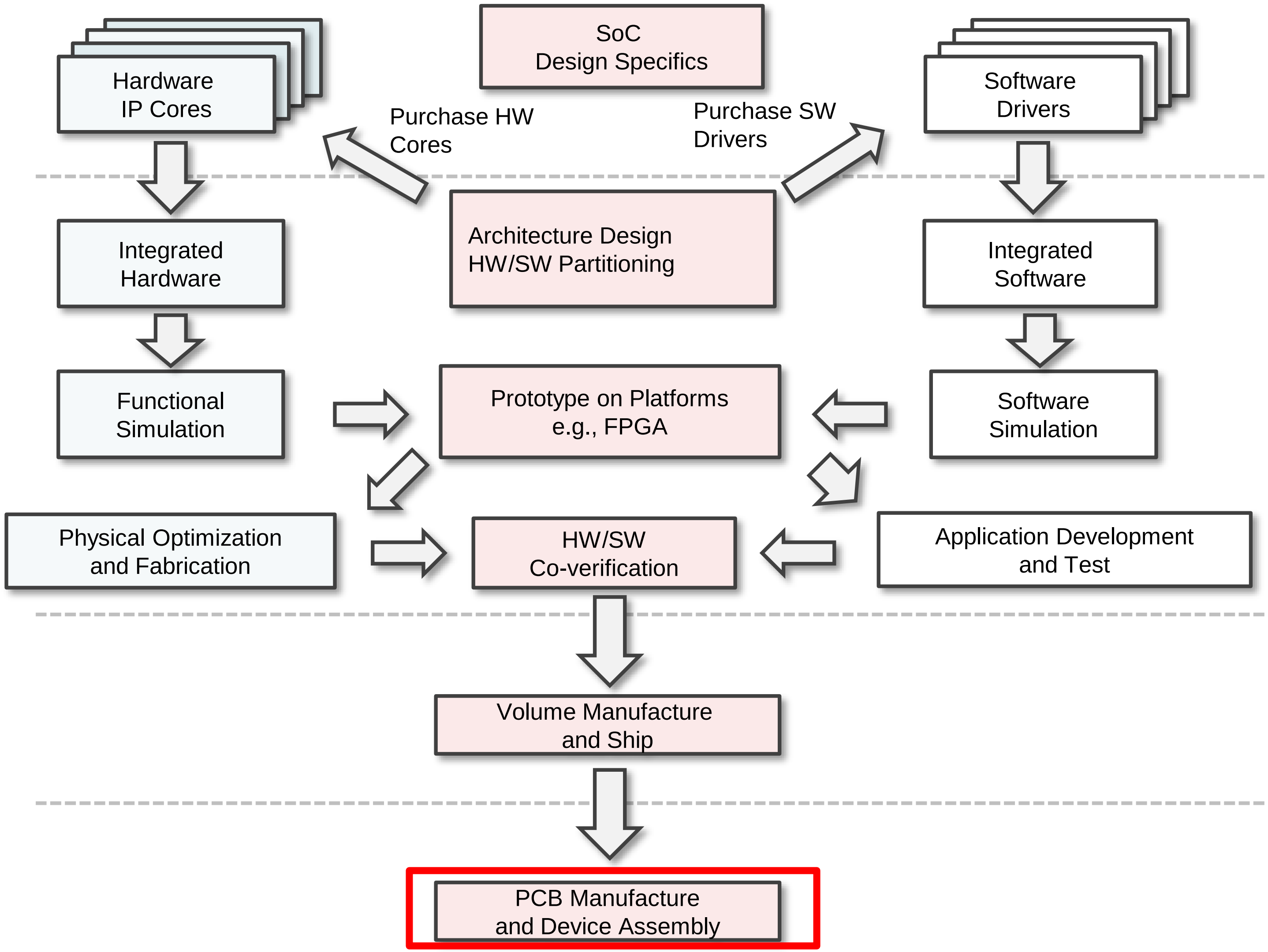
Fabless Vendors: SoC
Design

Foundries:
Chip Fabrication

Device Vendors:
Final Products

source: ARM

SoC Design Flow



SoC
Design Specifics

Hardware
IP Cores

Purchase HW
Cores

Purchase SW
Drivers

Software
Drivers

Integrated
Hardware

Architecture Design
HW/SW Partitioning

Integrated
Software

Functional
Simulation

Prototype on Platforms
e.g., FPGA

Software
Simulation

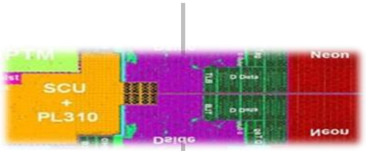
Physical Optimization
and Fabrication

HW/SW
Co-verification

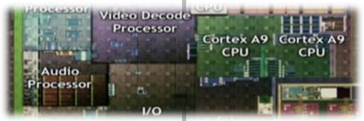
Application Development
and Test

Volume Manufacture
and Ship

PCB Manufacture
and Device Assembly



IP Vendors:
Core Design



Fabless Vendors: SoC
Design

Foundries:
Chip Fabrication

Device Vendors:
Final Products

source: ARM

Quiz

Q: Which of the following statements is incorrect?

A. An SoC is typically built using cores from different vendors.

B. Testing an SoC requires hardware/software co-simulation.

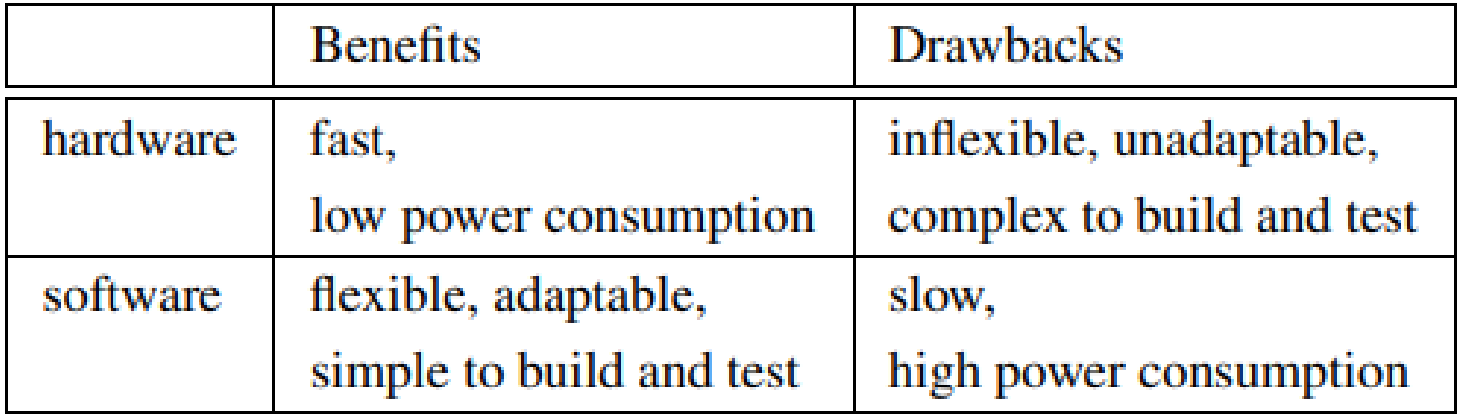
C. An SoC is normally developed for a specific application.

D. The SoC hardware components can be easily replaced if faulty.

HW/SW PARTITION

HW vs. SW tradeoffs

- Question: Which components in the SoC are to be implemented in HW and in SW?



	Benefits	Drawbacks
hardware	fast, low power consumption	inflexible, unadaptable, complex to build and test
software	flexible, adaptable, simple to build and test	slow, high power consumption

ASIC (no fetching and decoding)

General purpose CPU

Implement the performance critical parts in HW and rest in SW.

HW/SW Partitioning

- Objectives
  - Meet performance target with minimum hardware added
  - Minimize energy/power while meeting performance constraint
  - Maximum performance while keeping certain functions flexible
- Challenge: inter-dependence between the different metrics
  - Does adding application specific HW increase or decrease power?
  - How does application-specific HW impact overall system performance?

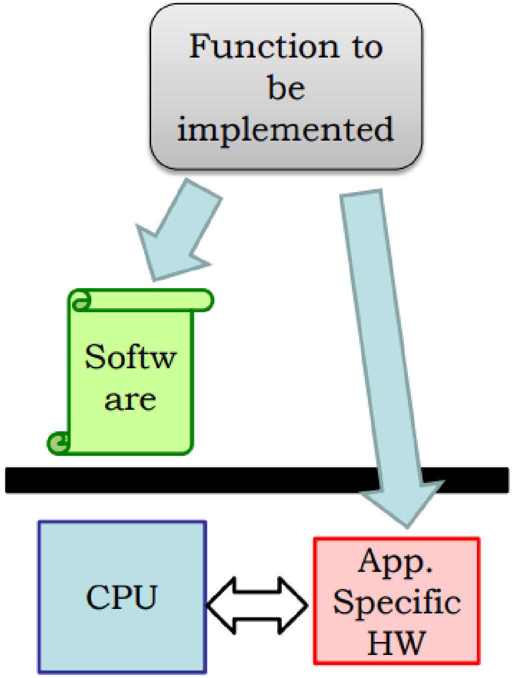
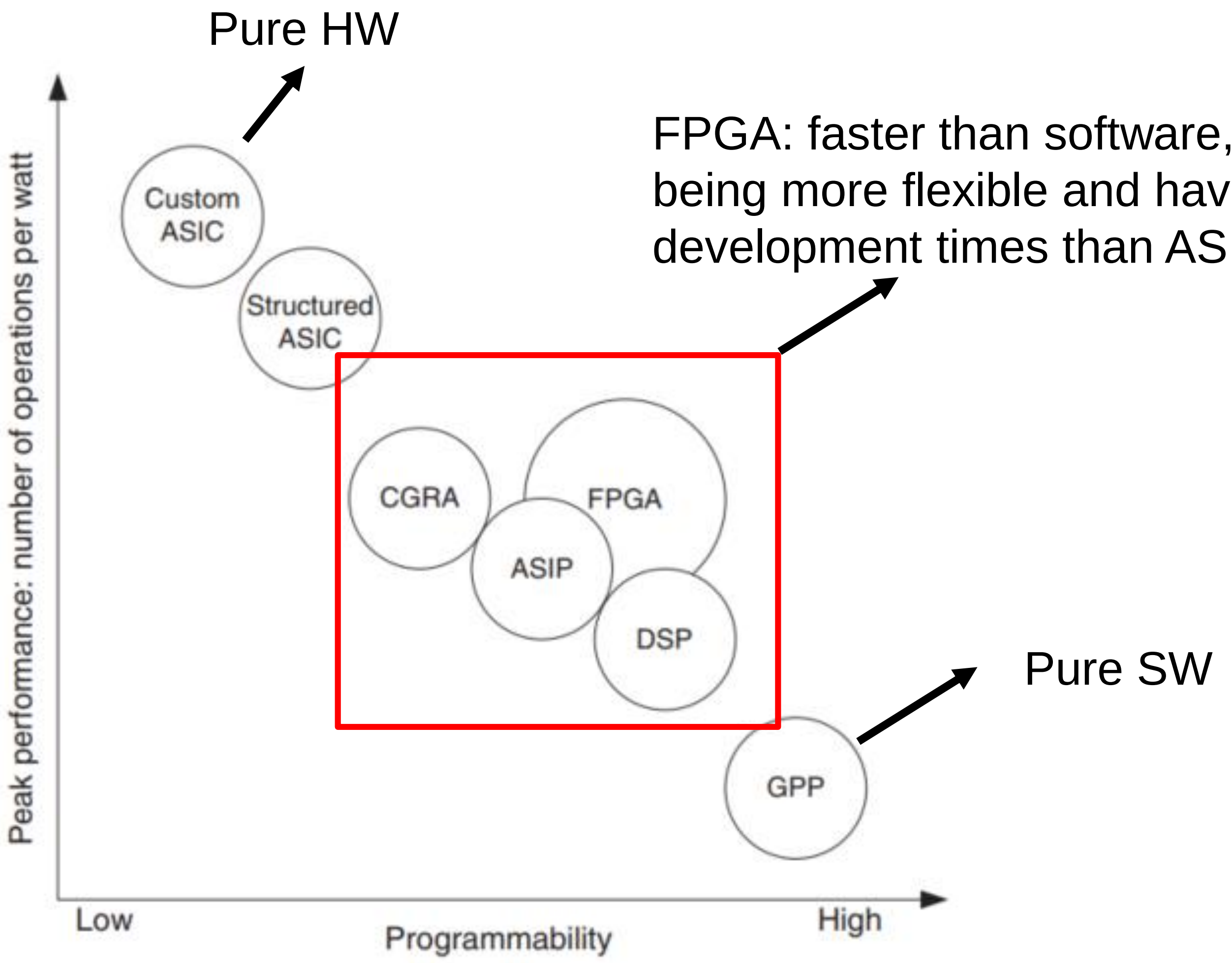


image: https://nanohub.org/courses/ECE695R/

Programmability vs. performance



Pure HW

FPGA: faster than software, while being more flexible and having shorter development times than ASIC

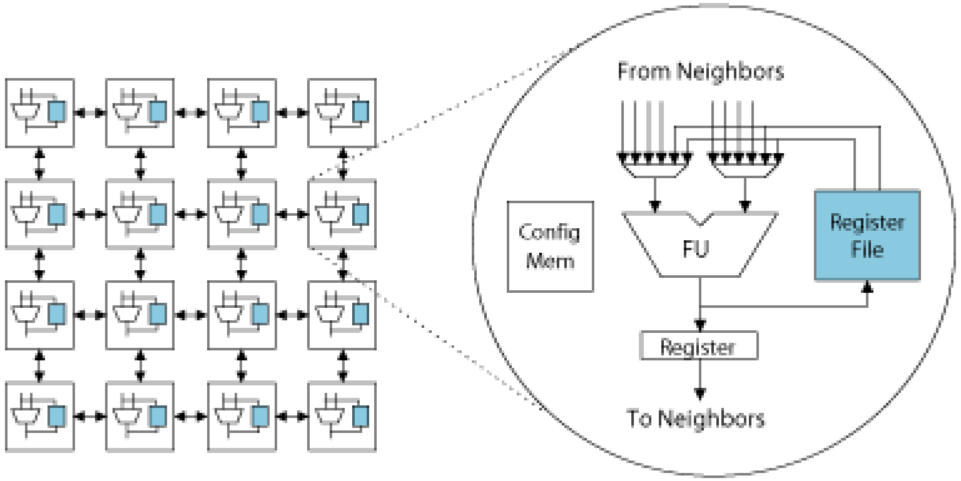
Pure SW

- FPGA - Field-Programmable gate arrays
- CGRA - Coarse Grained Reconfigurable Architecture
- ASIP - Application-Specific Instruction Processors

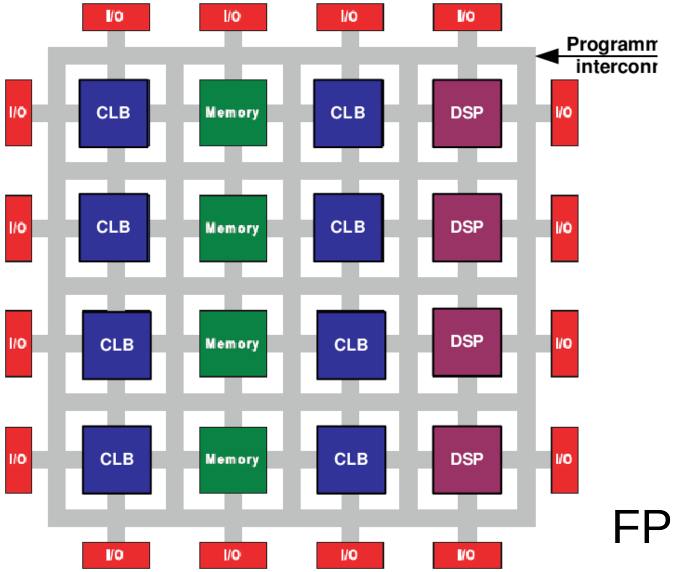
GPP: General purpose processor

CGRA vs. FPGA

- FPGAs pay a high price for **bit-level** configurability!
  - This is often not needed by applications!
- Compilation can be easier
  - Less configurability ➔ fewer choices for compiler
  - But harder to synthesize arbitrary high-level code onto a limited array



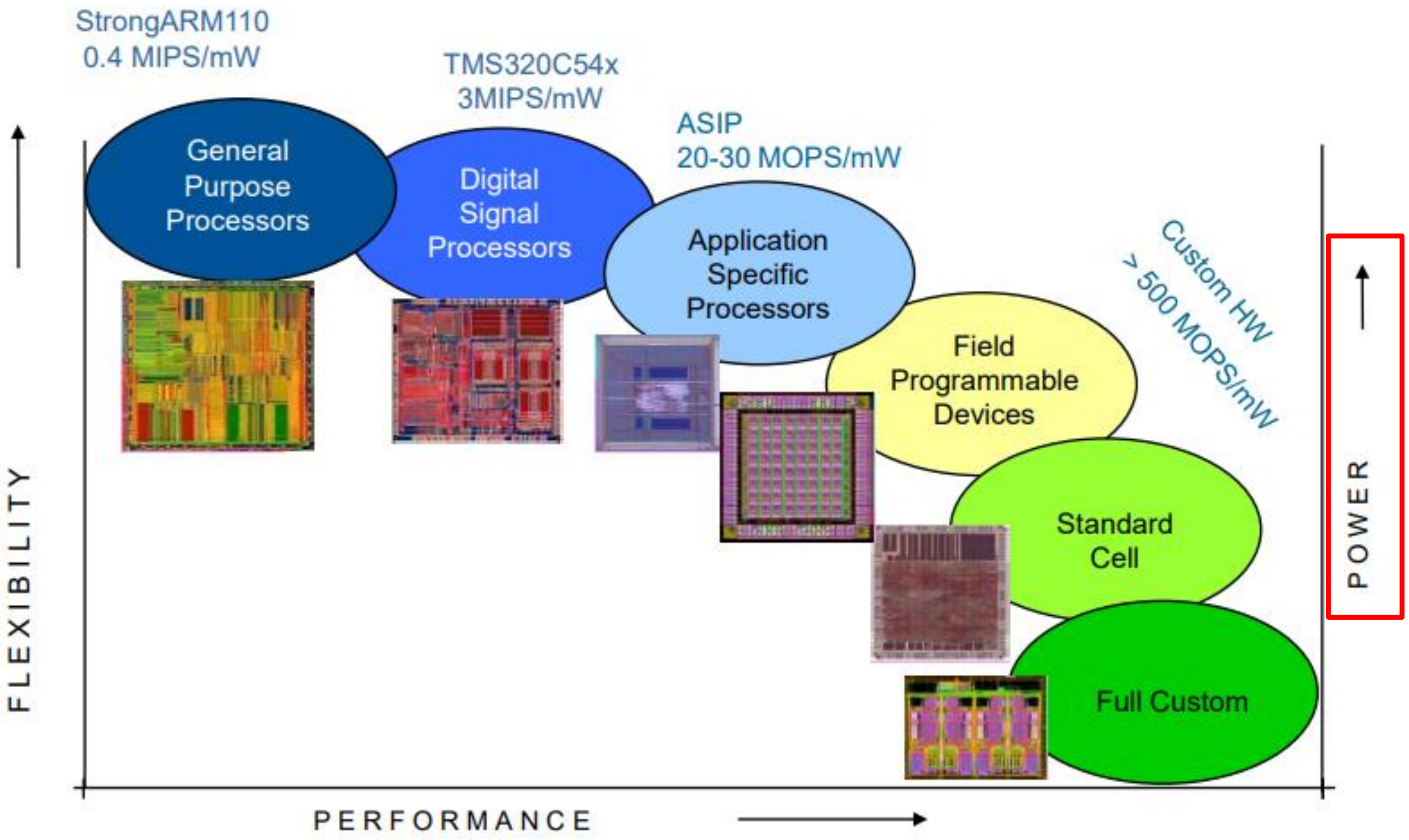
CGRA



FPGA

Image: https://www.quora.com/What-is-the-difference-between-FPGA-and-CGRA, "Field Programmable Gate Array (FPGA): From Conventional to Modern Architectures"

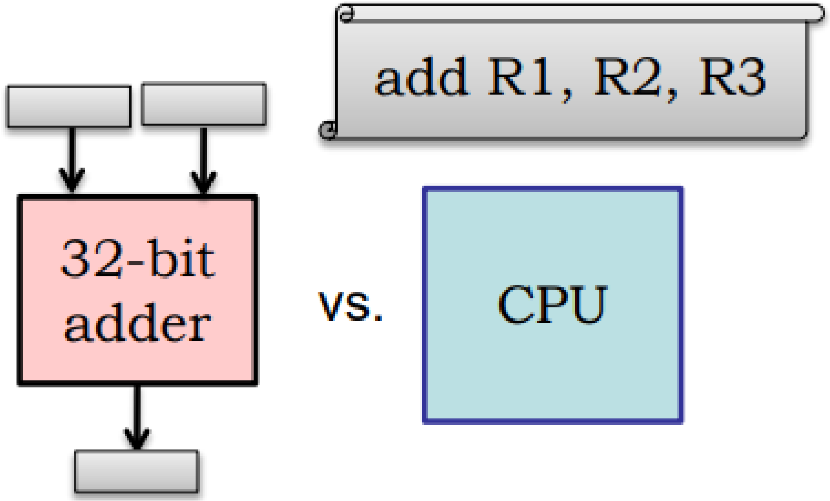
How about power?



source: https://nanohub.org/courses/ECE695R/

Why is ASIC more efficient?

- A simple example: `add` instruction
- Q: How much of the energy consumption of an add instruction in an embedded RISC processor goes into the adder itself?
  - Answer: 1-4%



source: https://nanohub.org/courses/ECE695R/

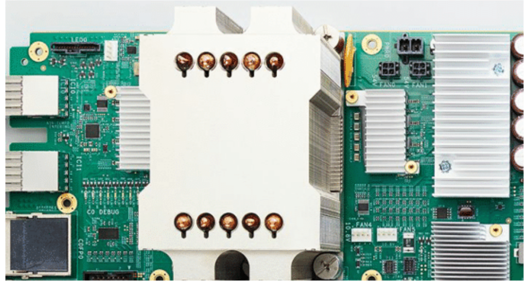
Why is ASIC more efficient?

- Abundant Parallelism
  - SW: limited by the degree of ILP that can be exploited by the processor (superscalar), or by the number of cores (multi-core), vector lanes (vector processor), etc.
  - SW: General-purpose processors are highly pipelined, and cannot exploit specific sequences of operations that can be performed in a single clock cycle
  - HW: customizing the processing elements to be as small as needed, we can utilize orders of magnitude more “processing elements” for a given power or area budget

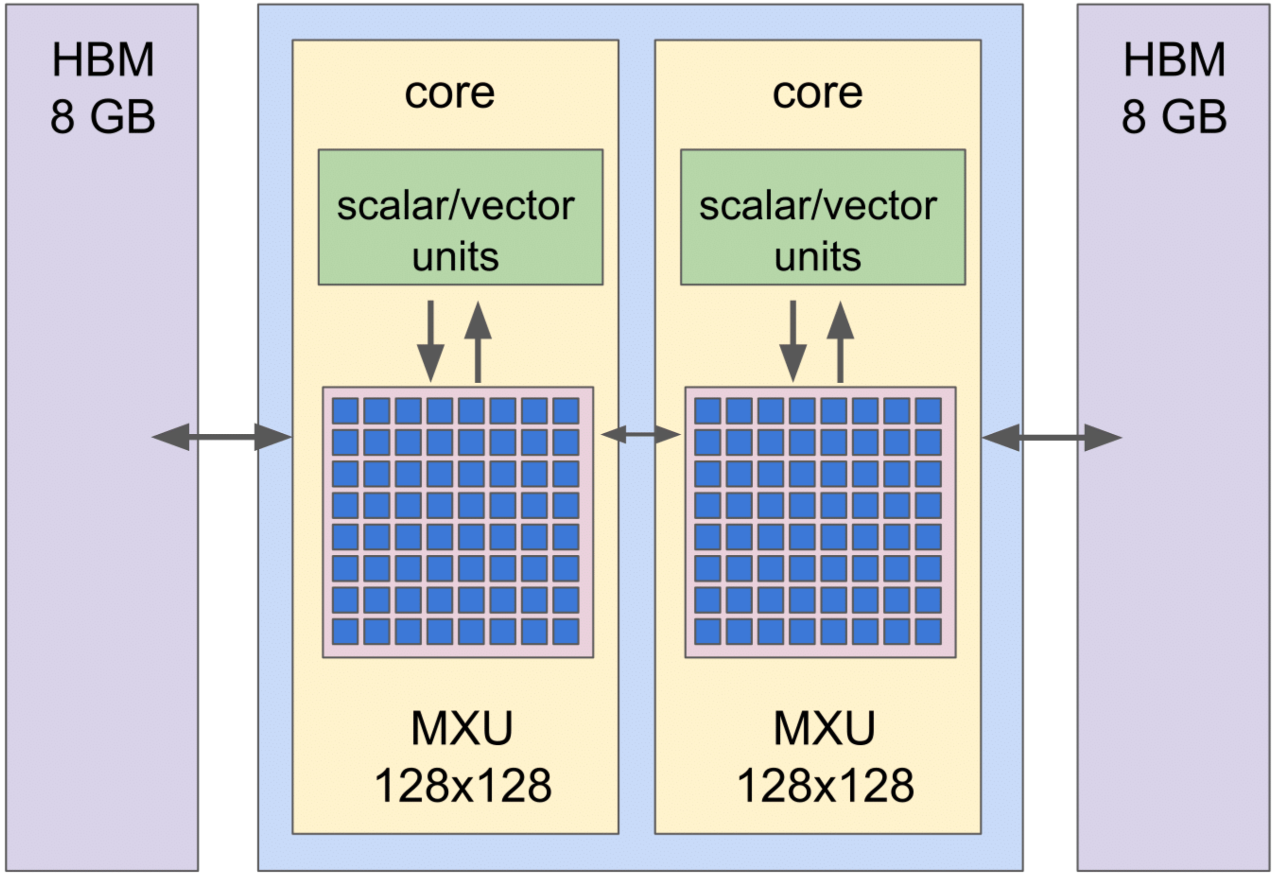
source: https://nanohub.org/courses/ECE695R/

Example: Google TPU

TPUv2 Chip



- 16 GB of HBM
- 600 GB/s mem BW
- Scalar/vector units: 32b float
- MXU: 32b float accumulation but reduced precision for multipliers
- 45 TFLOPS



source: Google

Example: Google TPU

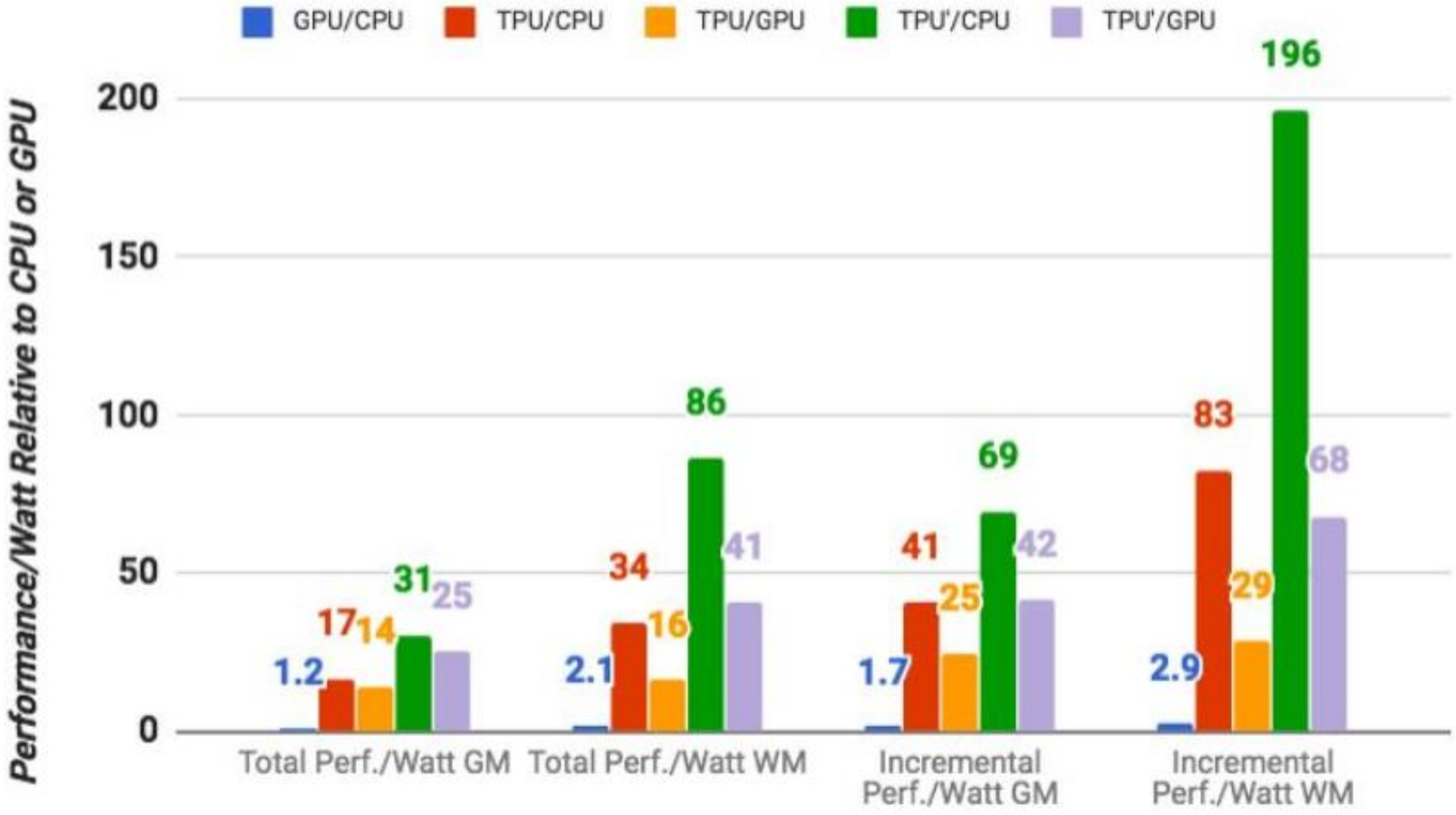


Figure 9. Relative performance/Watt (TDP) of GPU server (blue bar) and TPU server (red bar) to CPU server, and TPU server to GPU server (orange bar). TPU' is an improved TPU (Sec. 7). The green bar shows its ratio to the CPU server and the lavender bar shows its relation to the GPU server. Total includes host server power, but incremental doesn't. GM and WM are the geometric and weighted means.

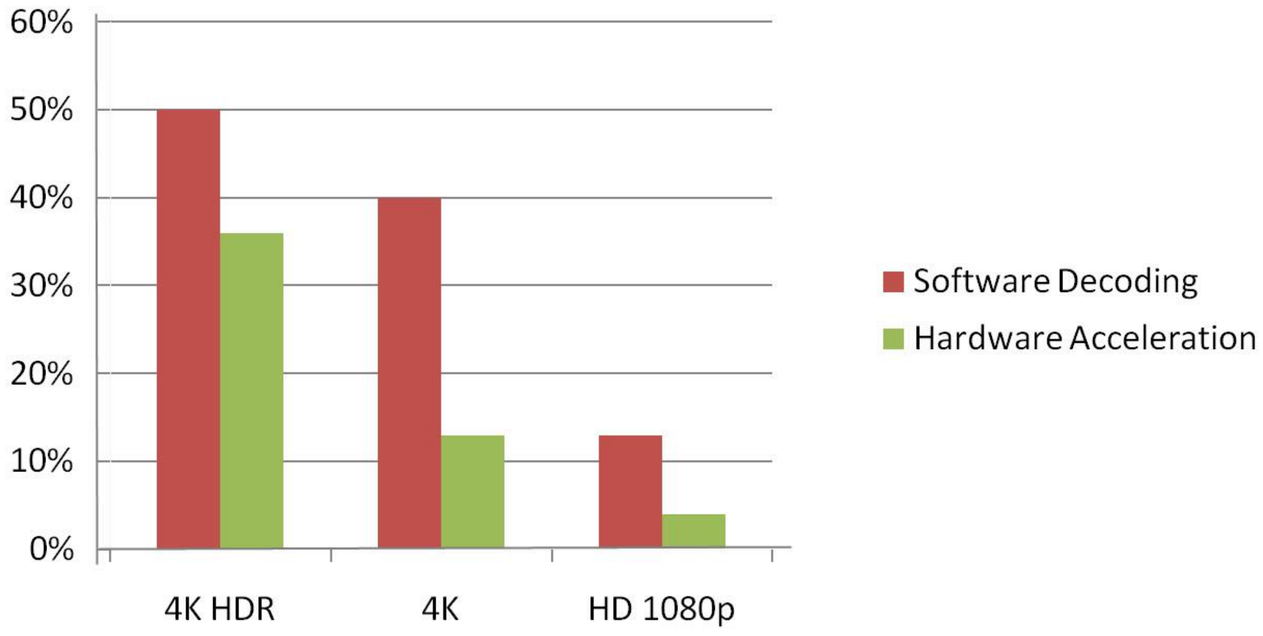
source: Google

Any other accelerated functions?

- Multimedia (Audio / Video / Image)
- Graphics (GPUs)
- Network protocol processing
- Cryptography
- Computer vision
- Recognition / Computer Vision
- Data Mining and Search
- Scientific Computation
- AI/ML
- …

Hardware Acceleration

- An example: 4K HDR Video (Sample-3) Decoding

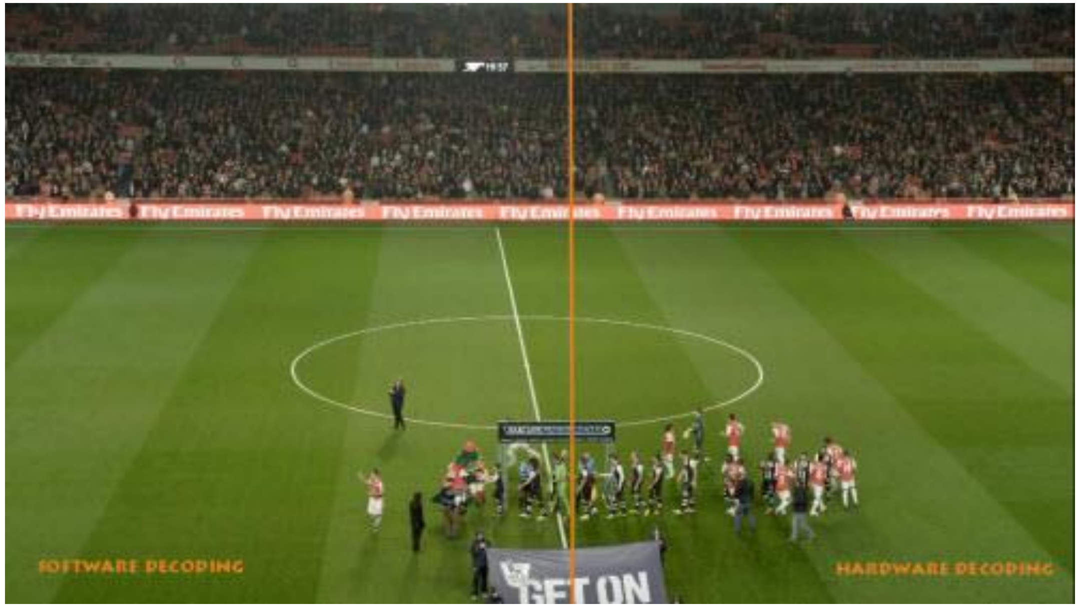


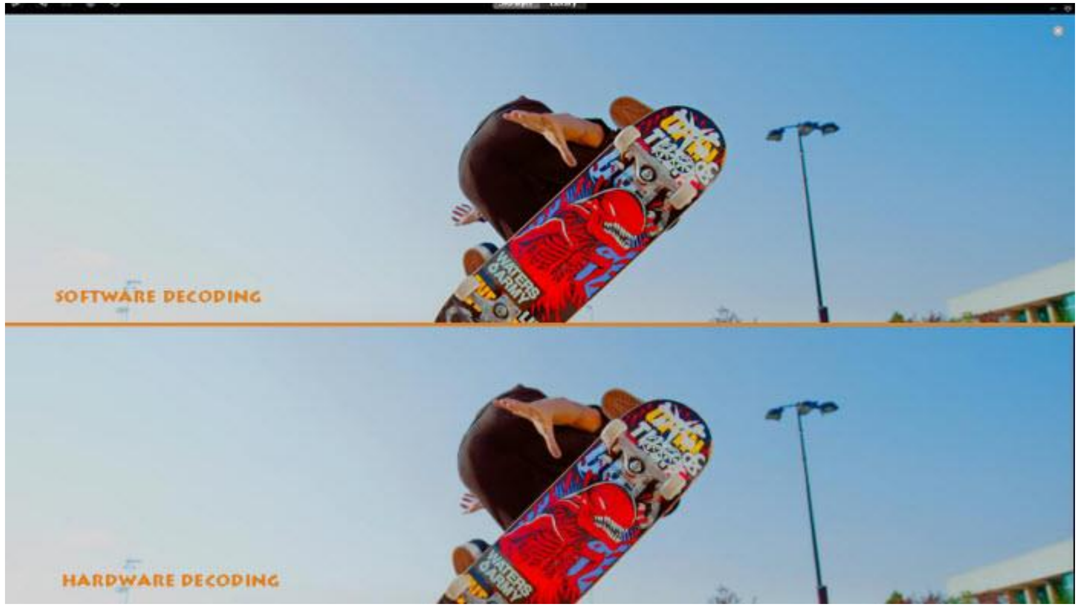
CPU Usage: Hardware Acceleration vs Software Decoding – Computer 1

source: https://www.5kplayer.com/video-music-player/paper-hardware-acceleration-vs-software-decoding.htm

Hardware Acceleration

- An example: 4K HDR Video (Sample-3) Decoding





Conclusion: hardware acceleration helps decode and render videos without occupying much CPU without scarifying quality!

source: https://www.5kplayer.com/video-music-player/paper-hardware-acceleration-vs-software-decoding.htm

A great reading

- On canvas

Understanding Sources of Inefficiency in General-Purpose Chips

Rehan Hameed[1], Wajahat Qadeer[1], Megan Wachs[1], Omid Azizi[1], Alex Solomatnikov[2], Benjamin C. Lee[1], Stephen Richardson[1], Christos Kozyrakis[1] and Mark Horowitz[1]

[1]Dept. of Electrical Engineering
Stanford University, Stanford, CA
{rhameed, wqadeer, wachs, oazizi,
bcclee, steveri, kozyraki, horowitz}@stanford.edu

[2]Hicamp Systems,
Menlo Park, CA
solomatnikov@gmail.com

ABSTRACT

Due to their high volume, general-purpose processors, and now chip multiprocessors (CMPs), are much more cost effective than ASICs, but lag significantly in terms of performance and energy efficiency. This paper explores the sources of these performance and energy overheads in general-purpose processing systems by quantifying the overheads of a 720p HD H.264 encoder running on a general-purpose CMP system. It then explores methods to eliminate these overheads by transforming the CPU into a specialized system for H.264 encoding. We evaluate the gains from customizations useful to broad classes of algorithms, such as SIMD units, as well as those specific to particular computation, such as customized storage and functional units.

The ASIC is 500x more energy efficient than our original four-processor CMP. Broadly, applicable optimizations improve performance by 10x and energy by 7x. However, the very low energy costs of actual core ops (100s fJ in 90nm) mean that over 90% of the energy used in these solutions is still "overhead". Achieving ASIC-like performance and efficiency requires algorithm-specific optimizations. For each sub-algorithm of H.264, we create a large, specialized functional unit that is capable of executing 100s of operations per instruction. This improves performance and energy by an additional 25x and the final customized CMP matches an ASIC solution's performance within 3x of its energy and within comparable area.

Categories and Subject Descriptors

C.5.4 [**Computer Systems Implementation**]: VLSI Systems – *customization, heterogeneous CMP*; C.1.3 [**Processor Architectures**]: Other Architecture Styles - *Heterogeneous (Hybrid) Systems*.

General Terms

Algorithms, Measurement, Performance, Design, Experimentation.

Keywords

ASIC, H.264, chip multiprocessor, high-performance, energy efficiency, customization, Tensilica.

ISCA'10, June 19–23, 2010, Saint-Malo, France.
Copyright 2010 ACM 978-1-4503-0053-7/10/06...$10.00.

1. INTRODUCTION

Most computing systems today are power limited, whether it is the 1W limit of a cell phone, or the 100W limit of a server. Since technology scaling no longer provides the energy savings, it once did [1], designers must turn to other techniques for continued performance improvements and tractable energy costs. One attractive option is to understand and to incorporate sources of ASIC efficiency, since general-purpose processors can be outclassed by three orders of magnitude in both performance and energy efficiency by ASIC designs [5].

The desire to achieve ASIC-like compute efficiencies with microprocessor-like application development cost is pushing designers to explore two new areas. One area aims to create CPU designs with much lower energy per instruction [6], while the other aims to create new design methodologies to reduce the cost of creating customized hardware. Examples of the latter include using higher levels of abstraction (e.g., C-to-RTL [8], [7]), and even full chip generators using extensible processors [2]. A critical first step in all of these approaches is to understand, in quantitative terms, the types and magnitudes of energy overheads in general-purpose processors. Once these are understood, it is then possible to explore ways to eliminate these overheads and assess the feasibility of creating an efficient, general-purpose machine.

This paper quantifies general-purpose overheads, exploring a series of customizations that reduce overheads to achieve ASIC-like efficiency. In particular, we consider three broad strategies: (1) techniques to exploit instruction- and data-level parallelism, such as VLIW and SIMD, (2) techniques to customize instructions by fusing complex, frequently occurring instruction sub-graphs, and (3) techniques to create application-specific data storage with fused functional units. These strategies span a range of general and domain-specific customization, incurring progressively greater design effort.

We evaluate these strategies by transforming a general-purpose, Tensilica-based, extensible CMP system into a highly efficient 720p HD H.264 encoder. We choose H.264 because it demonstrates the large energy advantage of ASIC solutions (500x) and because there exist commercial ASICs that can serve as a benchmark. Moreover, H.264 contains a variety of computational motifs, from highly data parallel algorithms (motion estimation) to control intensive ones (CABAC).

The results are striking. Starting from a 500x energy penalty, adding relatively wide (16x) SIMD execution units improves

Hameed R, Qadeer W, Wachs M, et al. Understanding sources of inefficiency in general-purpose chips, Proceedings of the 37th annual international symposium on Computer architecture. 2010: 37-47.

HW/SW Partitioning

- Differ primarily in how the custom HW is integrated with the processor
  - Co-processor (or HW accelerator)
  - Custom instruction

Co-processors / HW Accelerators

- Minimal changes to the processor itself
- Accelerator is connected to the system bus or a dedicated co-processor interface
- The instruction set and pipeline structure do not change

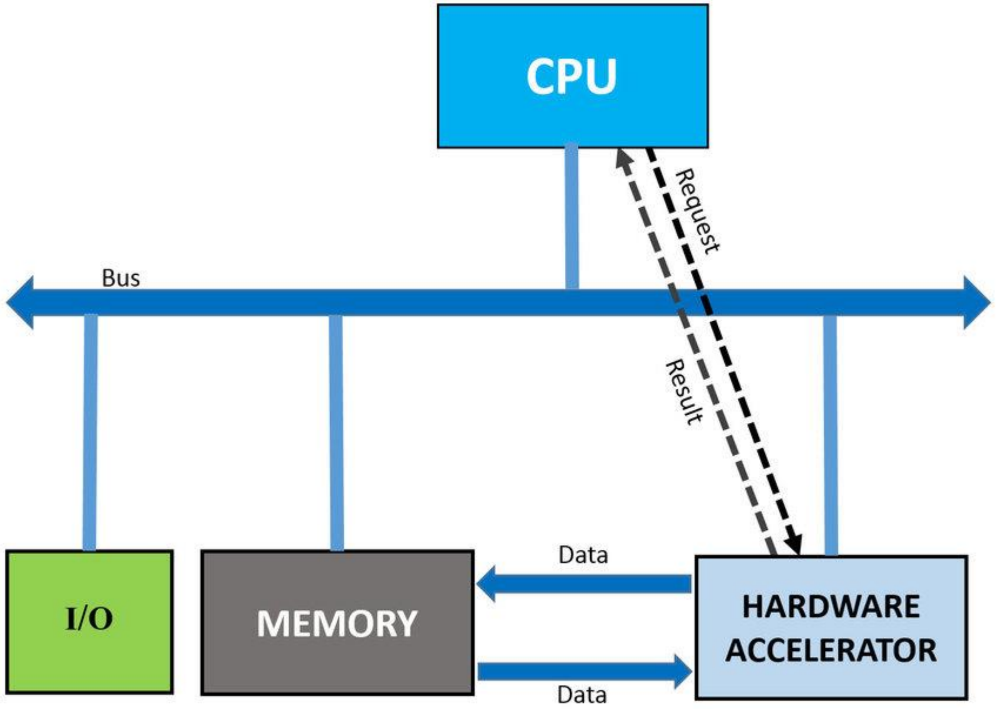
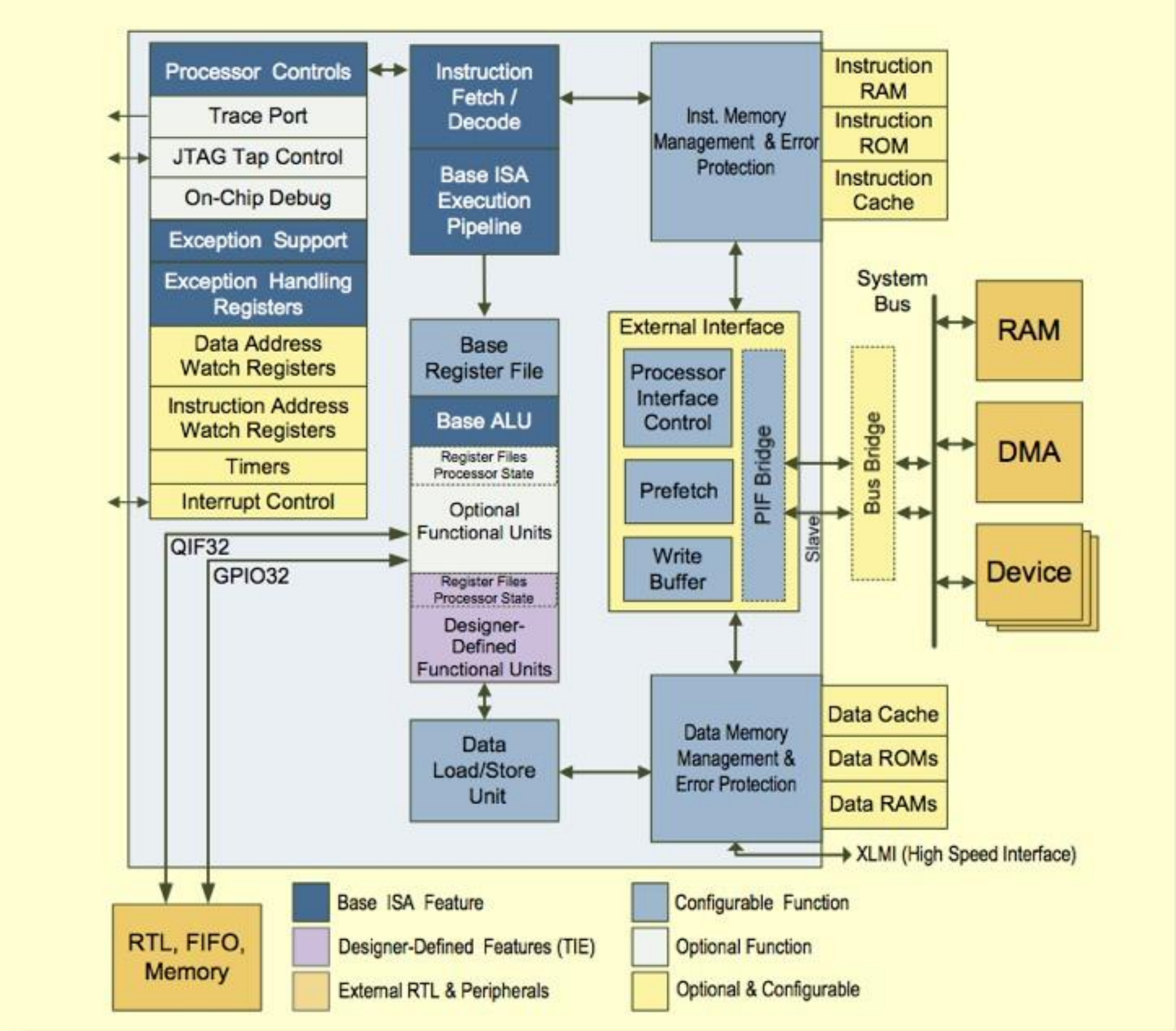


Figure: Ravi M, Sewa A, Shashidhar T G, et al. FPGA as a hardware accelerator for computation intensive maximum likelihood expectation maximization medical image reconstruction algorithm[J]. IEEE Access, 2019, 7: 111727-111735.

Custom Instruction Units

- Require fine-grained integration into the processor
  - Part of the processor’s pipeline
- Typically need to be synthesized together with the processor
- ISA changes
- More feasible with open ISA such as RISC-V

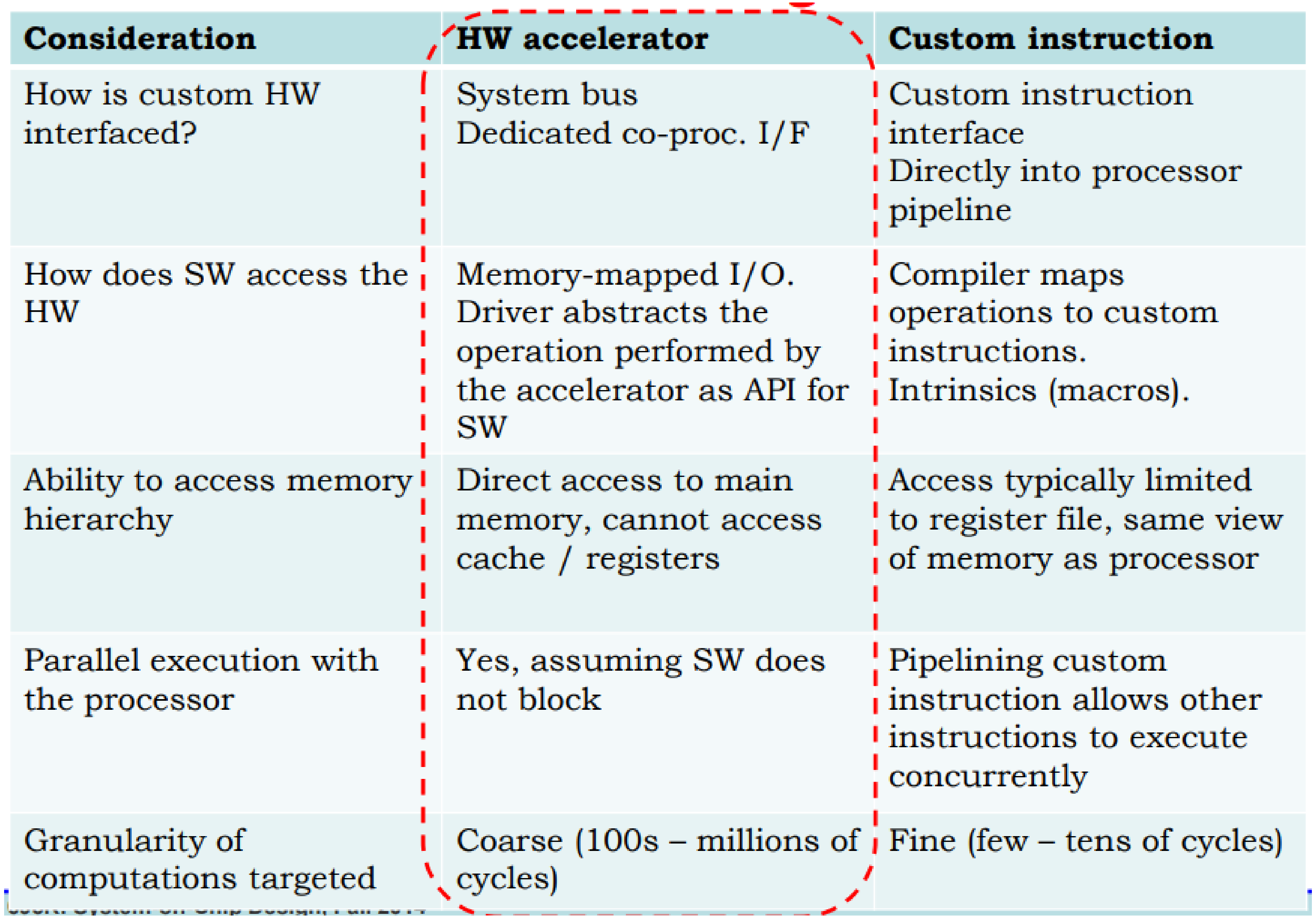


Processor Controls
Trace Port
JTAG Tap Control
On-Chip Debug
Exception Support
Exception Handling Registers
Data Address Watch Registers
Instruction Address Watch Registers
Timers
Interrupt Control
Instruction Fetch / Decode
Base ISA Execution Pipeline
Inst. Memory Management & Error Protection
Instruction RAM
Instruction ROM
Instruction Cache
Base Register File
Base ALU
Register Files Processor State
Optional Functional Units
Register Files Processor State
Designer-Defined Functional Units
External Interface
Processor Interface Control
Prefetch
Write Buffer
PIF Bridge
Slave
System Bus
Bus Bridge
RAM
DMA
Device
QIF32
GPIO32
Data Load/Store Unit
Data Memory Management & Error Protection
Data Cache
Data ROMs
Data RAMs
XLMI (High Speed Interface)
RTL, FIFO, Memory
Base ISA Feature
Designer-Defined Features (TIE)
External RTL & Peripherals
Configurable Function
Optional Function
Optional & Configurable

Image: https://www.bdti.com/InsideDSP/2013/12/11/Cadence

Cadence Xtensa processors

Which one to choose?

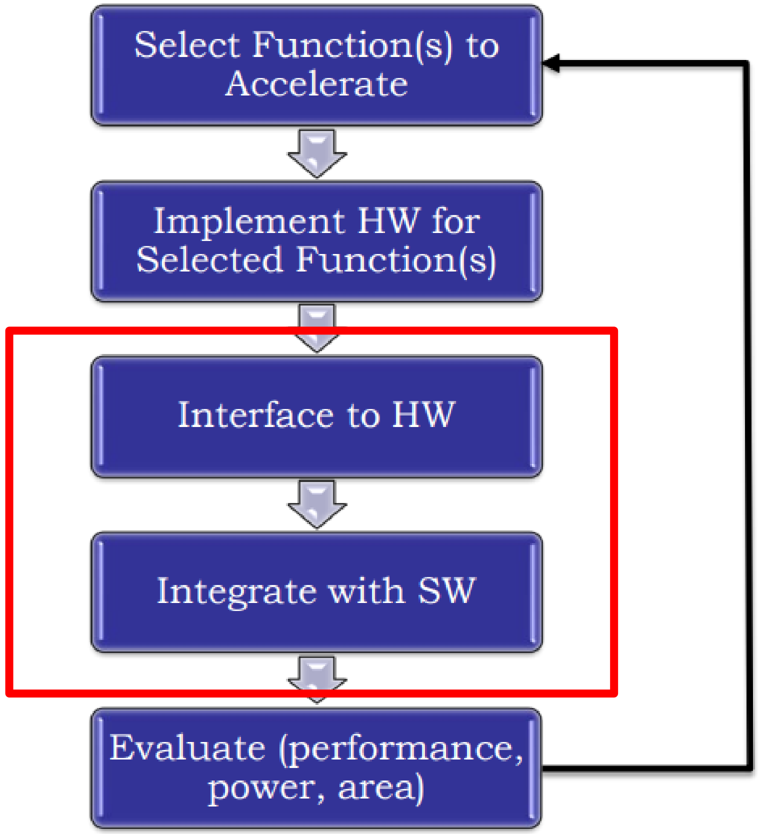


Consideration	HW accelerator	Custom instruction
How is custom HW interfaced?	System bus Dedicated co-proc. I/F	Custom instruction interface Directly into processor pipeline
How does SW access the HW	Memory-mapped I/O. Driver abstracts the operation performed by the accelerator as API for SW	Compiler maps operations to custom instructions. Intrinsics (macros).
Ability to access memory hierarchy	Direct access to main memory, cannot access cache / registers	Access typically limited to register file, same view of memory as processor
Parallel execution with the processor	Yes, assuming SW does not block	Pipelining custom instruction allows other instructions to execute concurrently
Granularity of computations targeted	Coarse (100s – millions of cycles)	Fine (few – tens of cycles)

source: https://nanohub.org/courses/ECE695R/

How to do partitioning?

- a.k.a How to decide which part to accelerate?


Select Function(s) to Accelerate
Implement HW for Selected Function(s)
Interface to HW
Integrate with SW
Evaluate (performance, power, area)

source: https://nanohub.org/courses/ECE695R/

The four "C"s in SoC Design

- Is it a Computational bottleneck?
- What is the Communication overhead?
- Is the HW implementation Cost-efficient?
- Is it a Commonly used function with a Clean interface?

Amdahl's law

$$Speedup = \frac{T_{orig}}{T_{unaccel} + T_{accelerated} + T_{communication}} \leq \frac{1}{\frac{T_{unaccel}}{T_{orig}} + \frac{T_{communication}}{T_{orig}}}$$

source: https://nanohub.org/courses/ECE695R/

HW/SW partitioning challenges

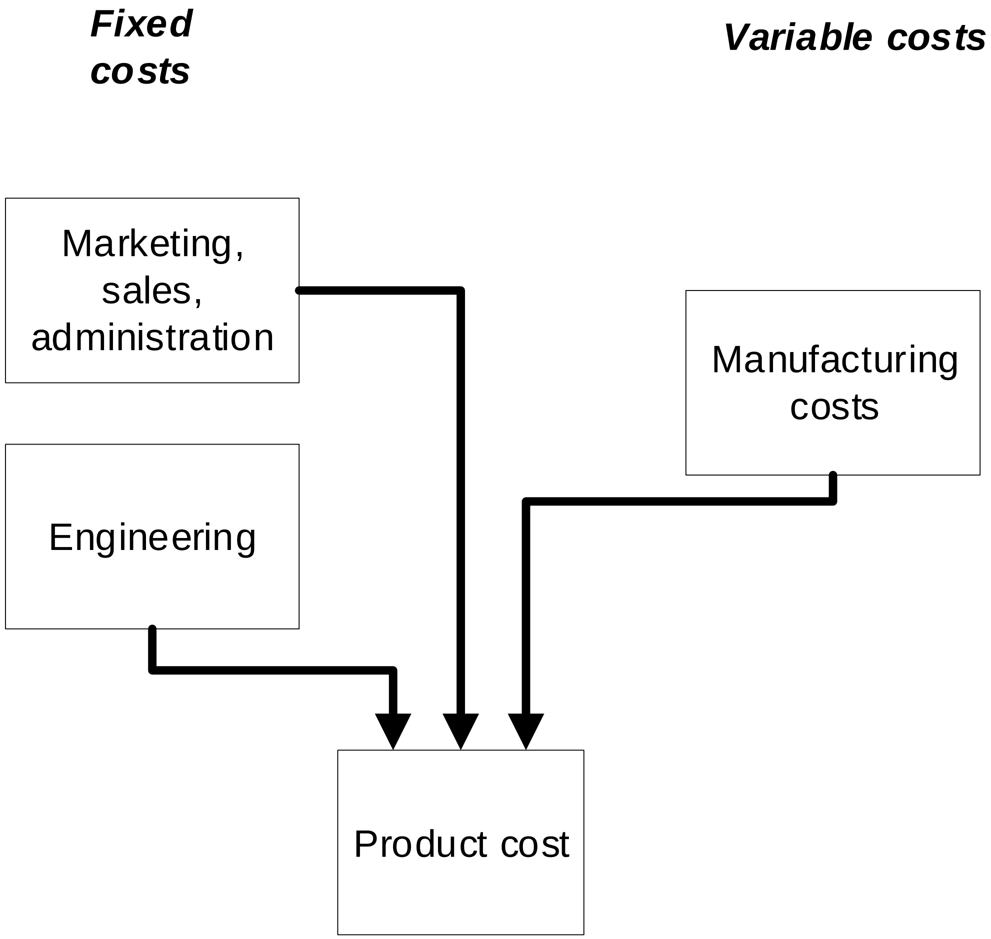
- Specification abstraction level
- Granularity
- System-component allocation
- Metrics and estimations
- Partitioning algorithms
- Objective and closeness functions
- Flow of control and designer interaction
- …

NRE COST

NRE Cost

- Non-Recurring Engineering (NRE)
  - Costs spent up front on development
    - Engineering Design Time
    - Prototypes
    - Mask costs
    - …
  - Recurring Engineering - Costs to produce each chip
  - Cost(Nchips) = Cost (NRE) + Nchips x Cost(manufacturing) /chip

Cost: product program vs engineering



Fixed costs

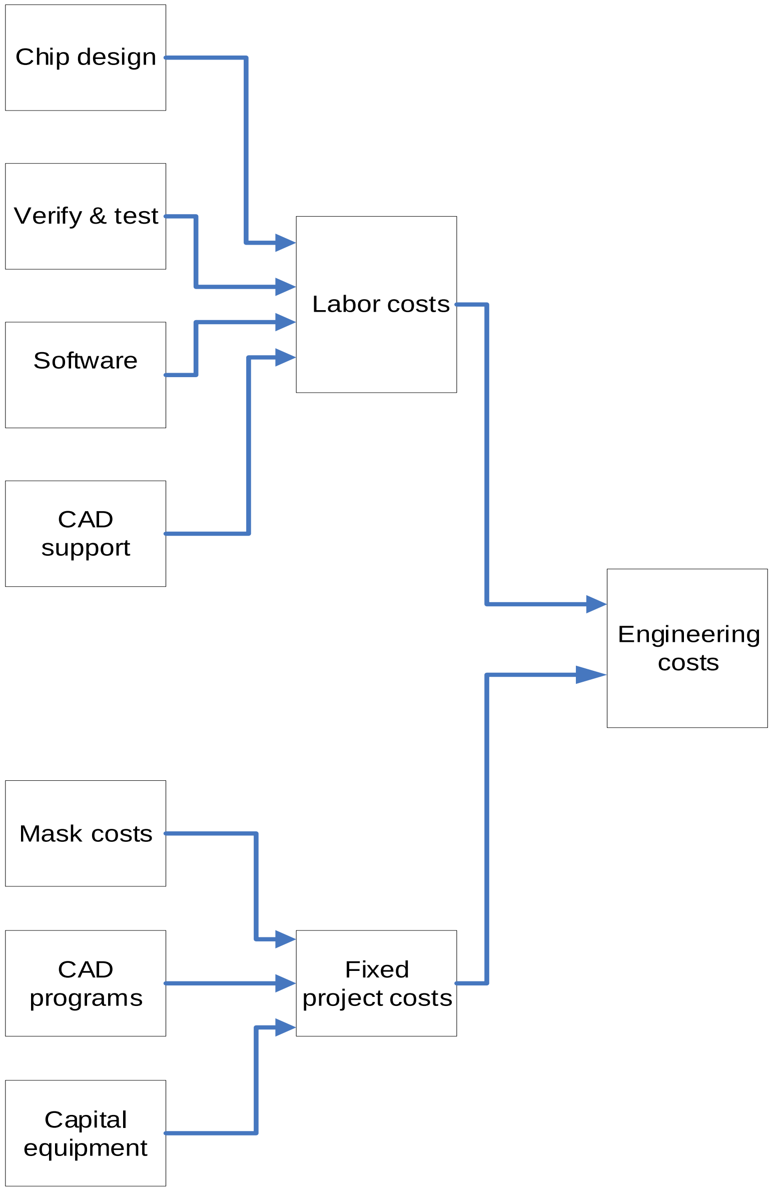
Variable costs

Marketing, sales, administration

Engineering

Manufacturing costs

Product cost



Chip design

Verify & test

Software

CAD support

Labor costs

Engineering costs

Mask costs

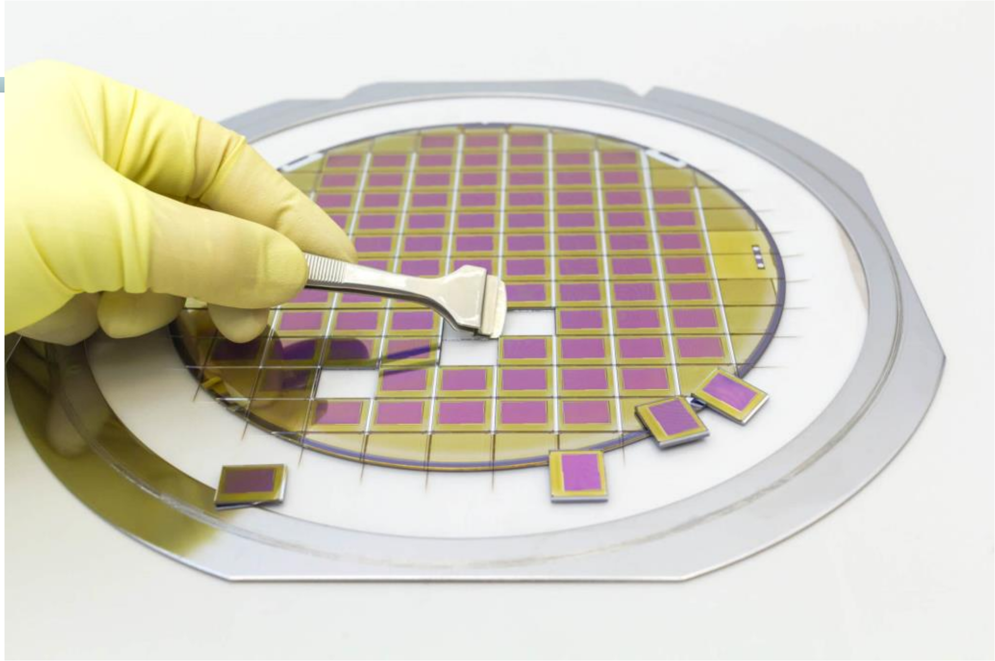
CAD programs

Capital equipment

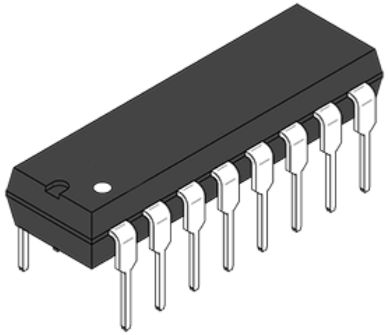
Fixed project costs

Variable (RE) Cost

- Variable costs (cost per part)
  - Wafer cost
  - Wafer processing
  - Die size (# die per wafer)
    - Size of design (# gates)
    - Technology (# gates per sq. inch)
    - % utilization of die
  - Production yield = f(defect density, die size)
  - Packaging

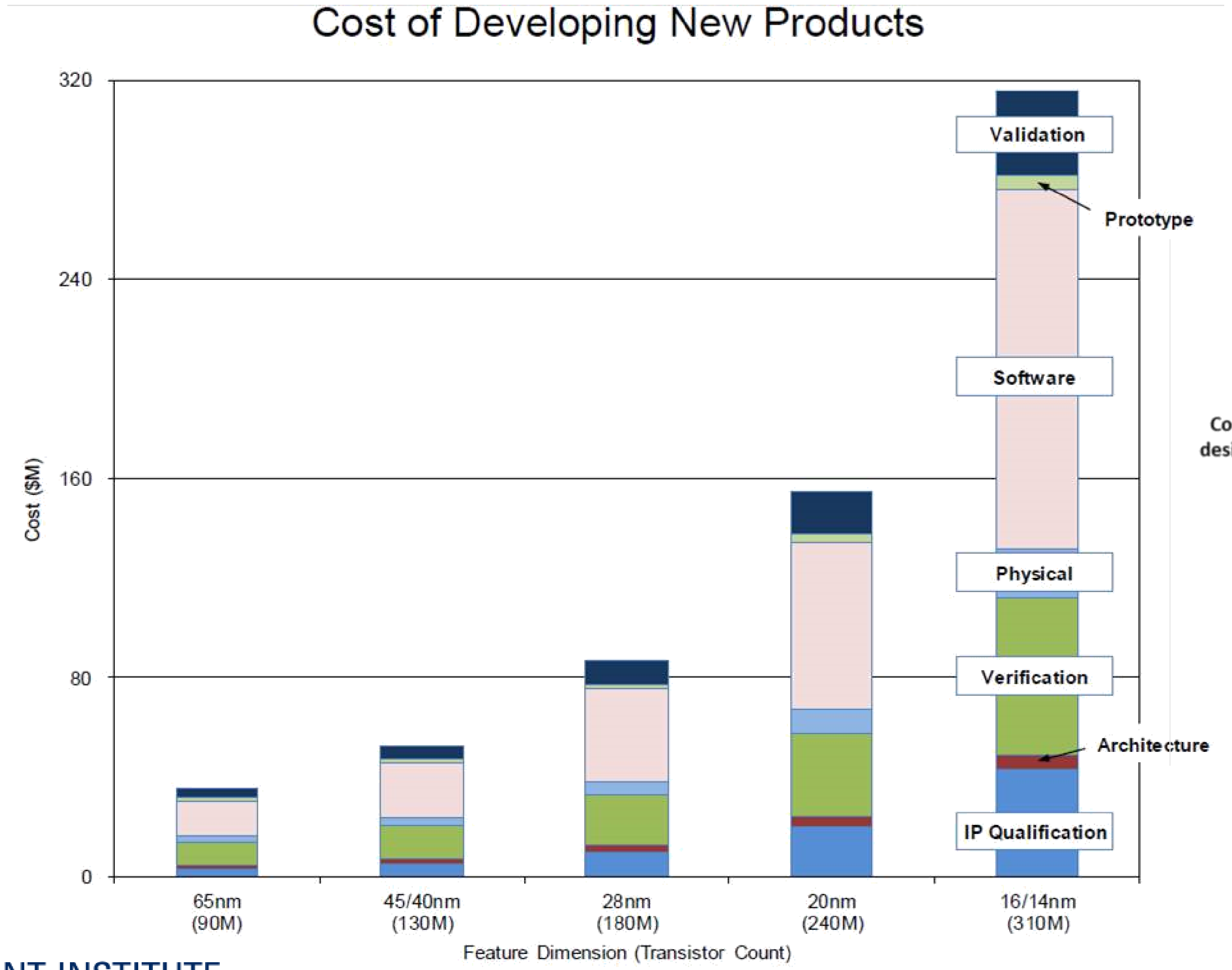


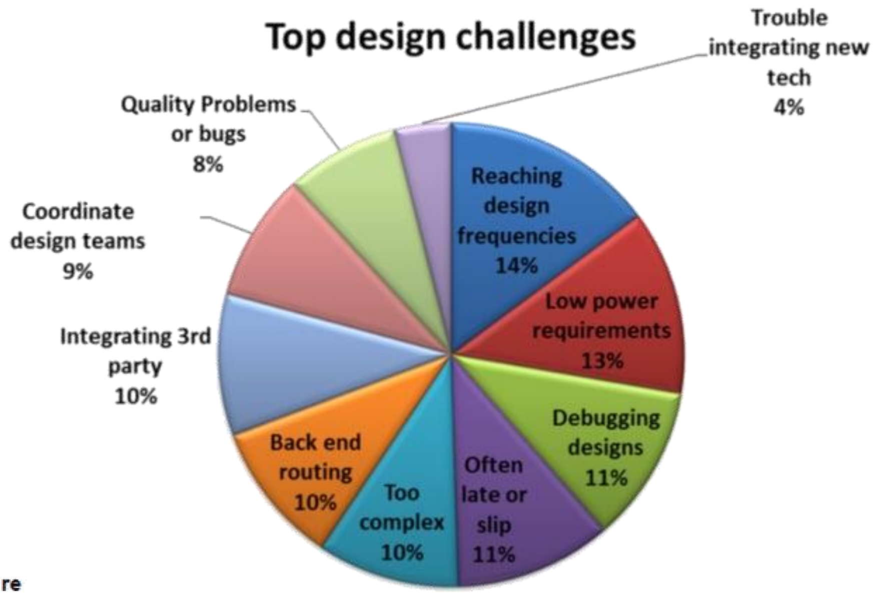
https://www.waferworld.com/post/top-causes-of-silicon-wafer-breakage



https://www.quick-pcba.com/pcb-news/ic-packaging-types.html

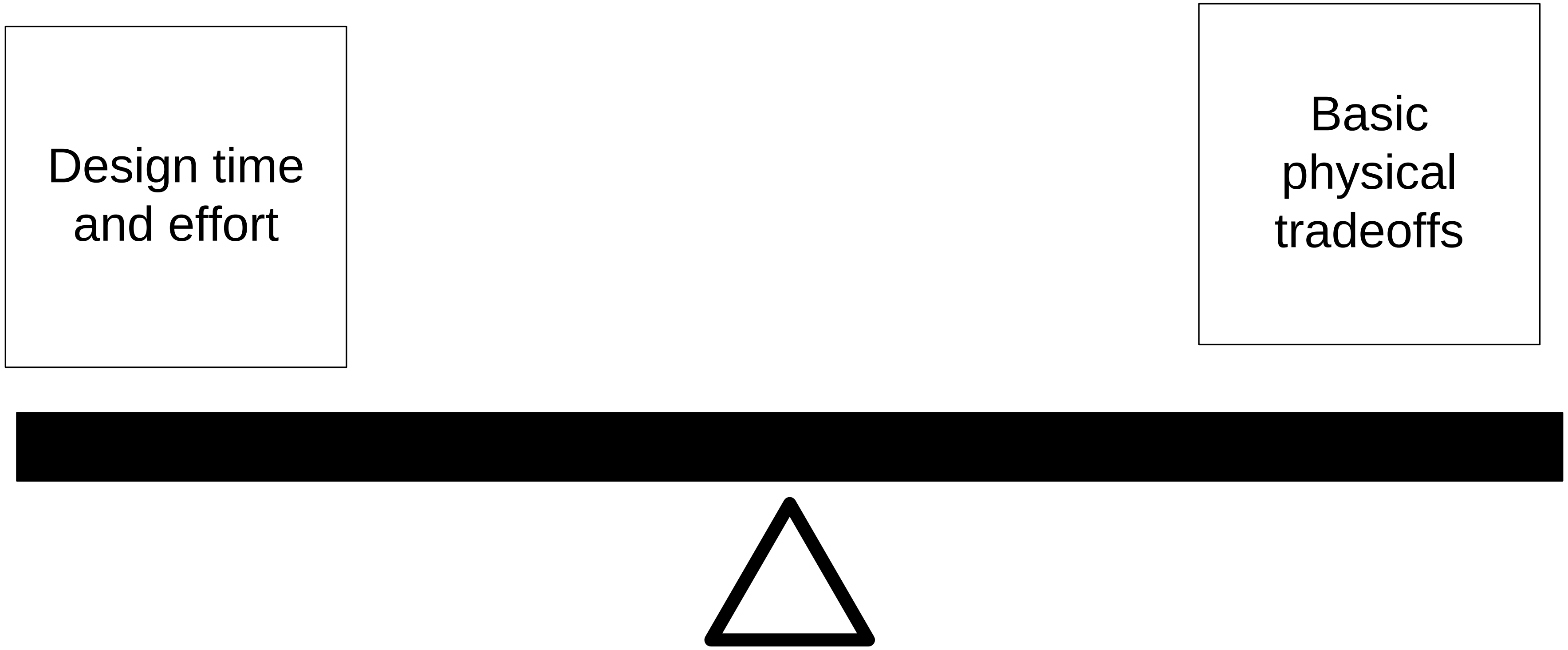
Cost is increasing!





source: https://semiengineering.com/how-much-will-that-chip-cost/

Product volume dictates design effort



Design time and effort

Basic physical tradeoffs

Balance point depends on
n, number of units

What is the solution?

- Amortize NRE with Volume

Cost(Nchips) = Cost (NRE) + Nchips x Cost(manufacturing) /chip

Cost/chip = Cost (NRE) / N(chips) + Cost(manufacturing) /chip

Market requirements

- Economics force fewer, more customizable chips
- Mask costs in the millions of dollars
- Custom IC design NRE 10s—100s of millions of dollars
- Number of unique chips must decrease
- Increase the programmability is one option

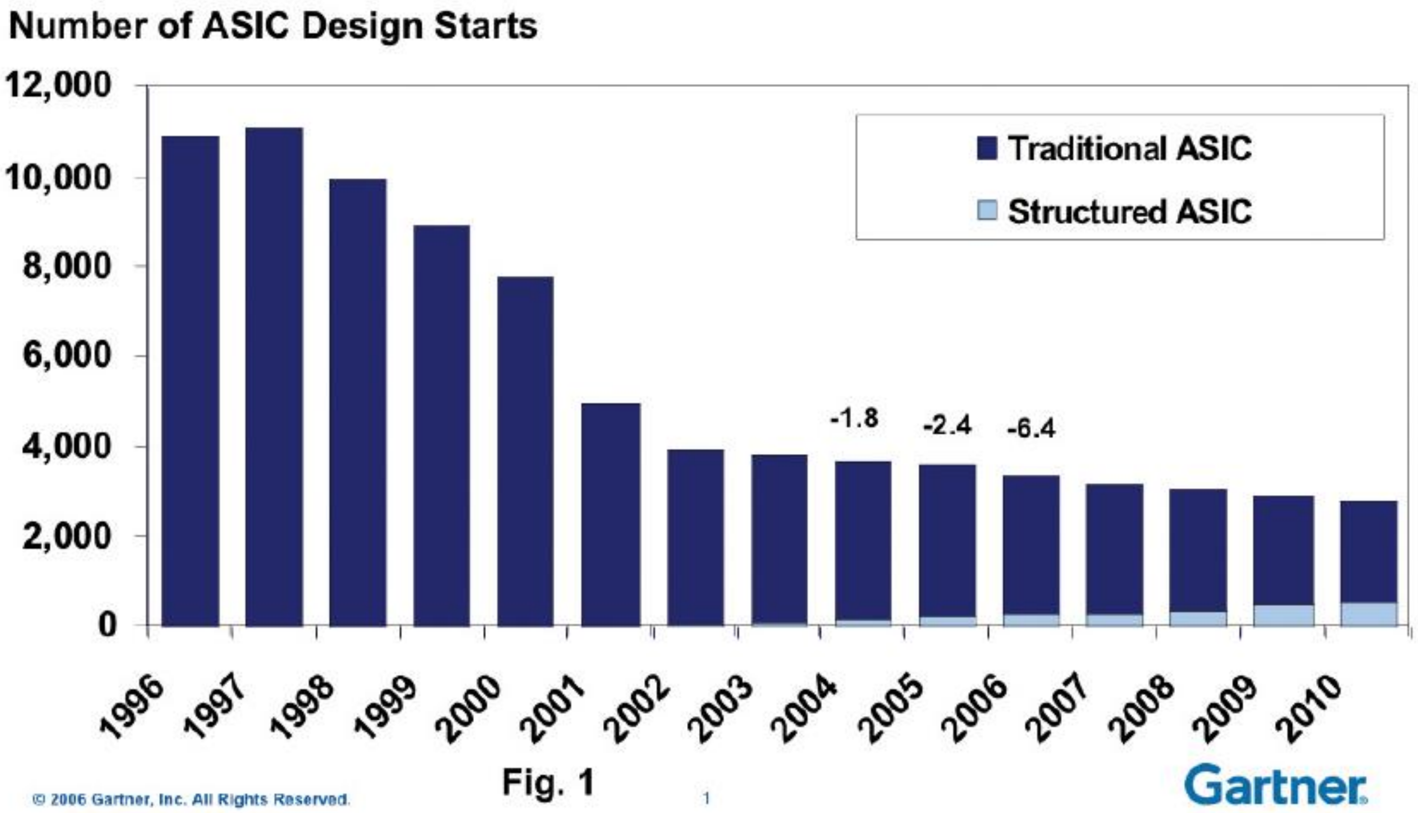
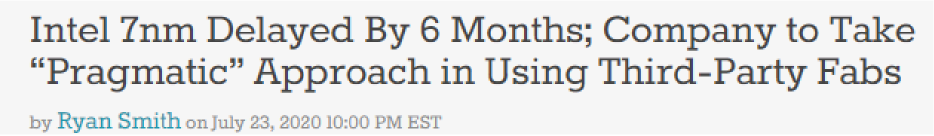


Fig. 1

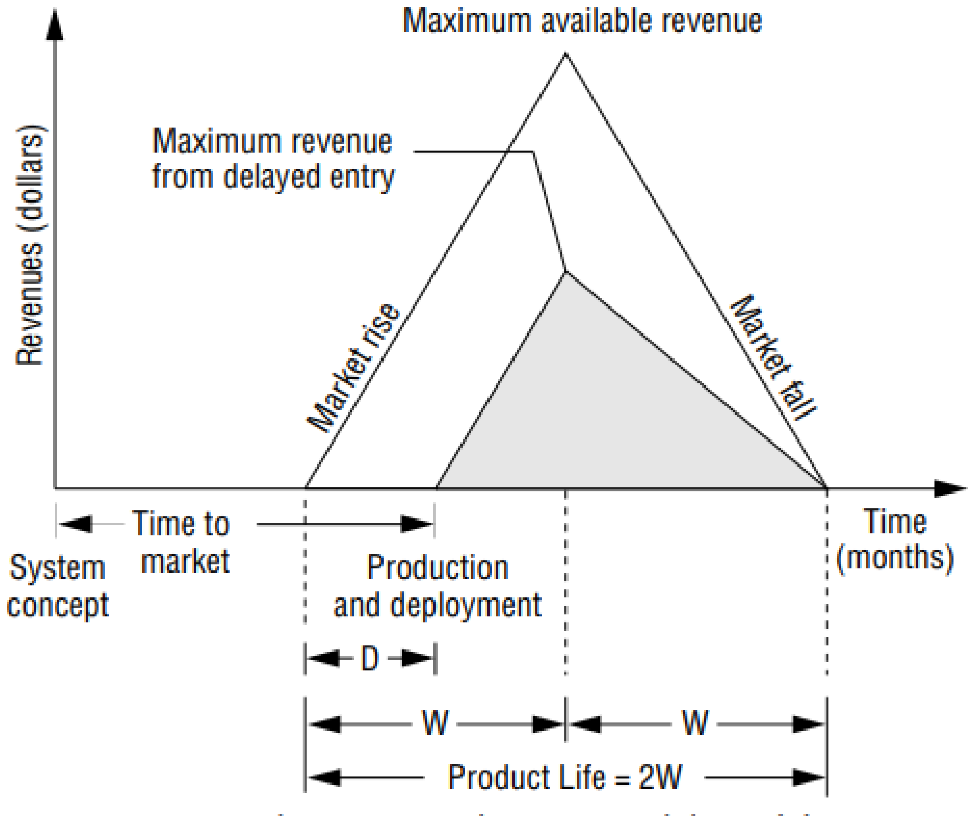
source: Upenn ESE532 Fall 2021 -- DeHon

SoC Profit Model



Intel 7nm Delayed By 6 Months; Company to Take "Pragmatic" Approach in Using Third-Party Fabs

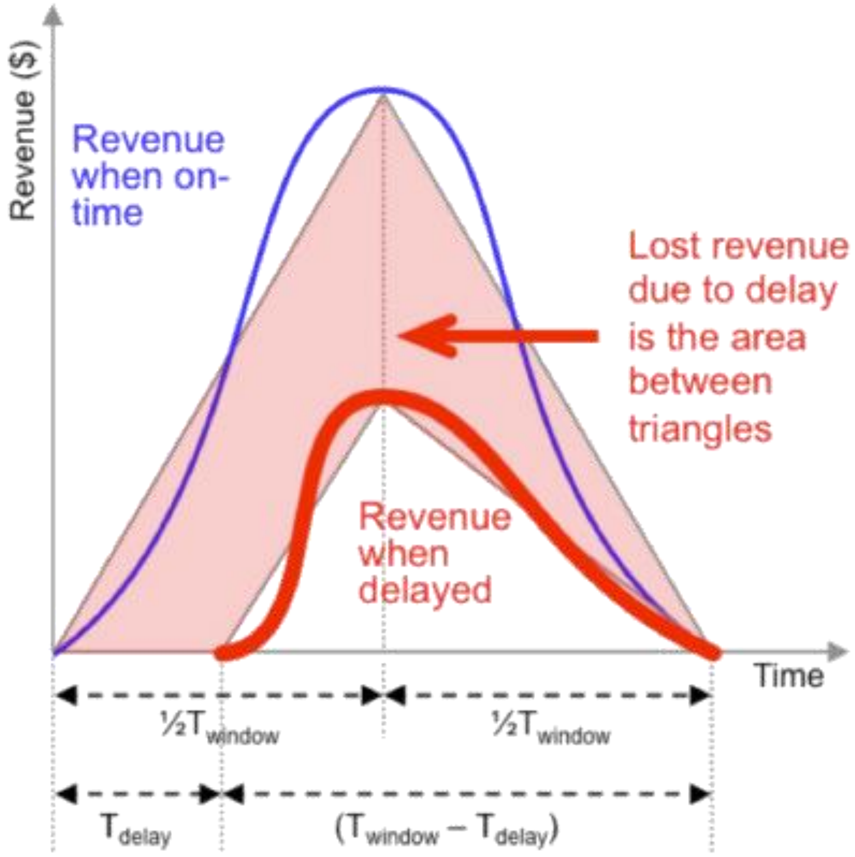
by Ryan Smith on July 23, 2020 10:00 PM EST



- Simplified revenue model
  - Product life = 2W, peak at W
  - Time of market entry defines a triangle, representing market penetration
  - Triangle area equals revenue
- Loss
  - The difference between the on-time and delayed triangle areas

Debardelaben J A, Madisetti V K, Gadient A J. Incorporating cost modeling in embedded-system design[J]. IEEE Design & Test of Computers, 1997, 14(3): 24-35. **(Available on Canvas)**

Percentage revenue loss



- On-time = ?
- Delayed = ?
- Percentage revenue =

(On-Time – Delayed)/On-Time*100% = ?

source: https://www.arteris.com/blog/bid/112221/what-does-it-cost-you-when-your-soc-is-late-to-market

Quick Summary - SoC design: key ideas

- To design and evaluate an SoC, designers need to understand:
  - its components: processors, memory, interconnect
  - applications that it targets
- SoC economics heavily dependent on:
  - costs: initial design, marginal production
  - volume: applicability, lifetime
- Reducing design complexity
  - Intellectual Property (IP)
  - reconfigurable technology

Where are we Heading?

- SoC System Approach II

Action Items

- Lab #0 due by Sunday Sept. 28th 23:59
- Watch out announcements for lab logistics, lab group assignment, etc.
- Reading Materials
  - Ch. 1.2 - 1.4,1.9, Two research papers on canvas

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. Xuan Zhang (WUSTL)
- Dr. Andreas Gerstlauer (UT Austin)
- Dr. Anand Raghunathan (Purdue)