System-on-Chip with IoT Applications (EC667)

Dr. Rajib Malik

Assistant Professor, ECE Department, IIITG

A Motivational Story

Celebrating 25 years of MediaTek in 2022

- MediaTek was originally a unit of United Microelectronics Corporation (UMC) tasked with designing chipsets for home entertainment products. On May 28, 1997, the unit was spun off and incorporated. MediaTek Inc. was listed on the Taiwan Stock Exchange (TSEC) under the "2454" code on July 23, 2001.
- The company started out designing chipsets for optical drives and subsequently expanded into chip solutions for DVD players, digital TVs, mobile phones, smartphones and tablets. In general MediaTek has had a strong record of gaining market share and displacing competitors after entering new markets.
- MediaTek powers more than 2 billion devices a year – that's in 20 percent of homes and nearly 1 of every 3 mobile phones globally.

Let me ask you about your expectations!

What you will get from this course

- There are four parts to this course:

1. How to design your own SoC architecture, especially for IoT
2. What are the requirements to be followed for IoT applications
3. How to build middleware for the designed device
4. How to make your device smart and more reachable for IoT applications.

Pre-requisites to brush up before you dive in

- Digital Design, especially universal gates and sequential logic
- Embedded Systems
- VLSI Design
- Architecture and organization of computers

What we will cover and when?

1. Prerequisites; SoC basic components and why they are different from micro-processor or GPU systems; Design process flow for SoC, IoT and Cyber Physical Systems

2. SoC Design and Verification flow for ASIC and FPGA based systems

3. How to implement and verify SoC using HDL and HLLs; tools for early design space exploration.
4. Features required for IoT applications and Real Time Operating Systems for IoT

How to approach the course?

- Computer System Design – System on Chip
  by Michael J. Flynn and Wayne Luk
- Modern System-on-Chip Design on Arm
  by David J. Greaves
- Verilog HDL: A Guide to Digital Design and Synthesis
  by Samir Palnitkar
- Ask me further doubts in class or later via email (rajib@iiitg.ac.in).

What is Soc?

- A System on Chip (SoC) is an integrated circuit that consolidates all the essential components of a computer or other electronic systems onto a single chip.

- SoCs are widely used in mobile devices (smartphones, tablets), embedded systems, and Internet of Things (IoT) devices because they offer **high performance with low power consumption** in a compact form factor.

- They are designed to perform a specific function or set of functions, often with high efficiency and at a lower cost than traditional multi-chip systems.

SYSTEM ARCHITECTURE: AN OVERVIEW

The past 40 years have seen amazing advances in silicon technology and resulting increases in transistor density and performance.

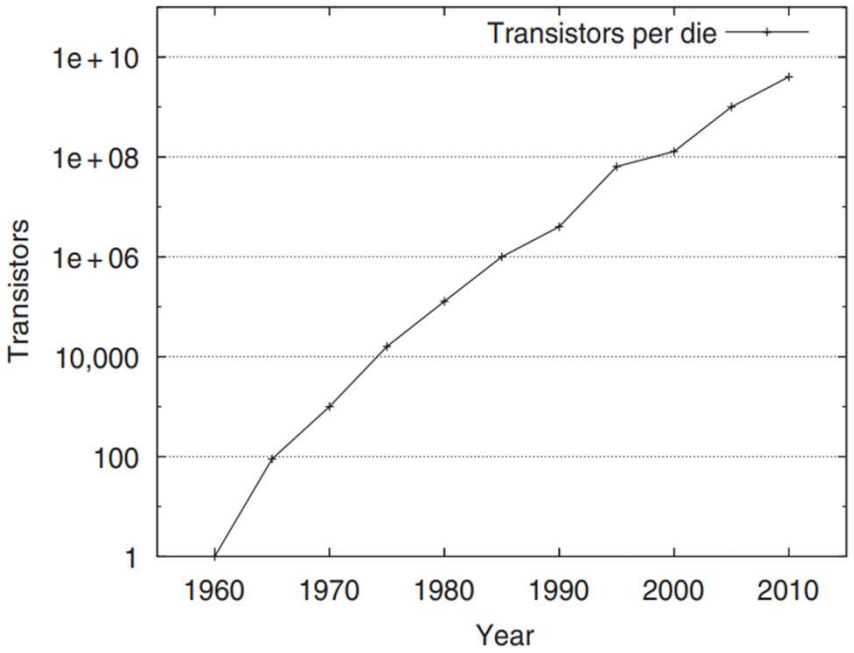


Figure 1.1 The increasing transistor density on a silicon die.

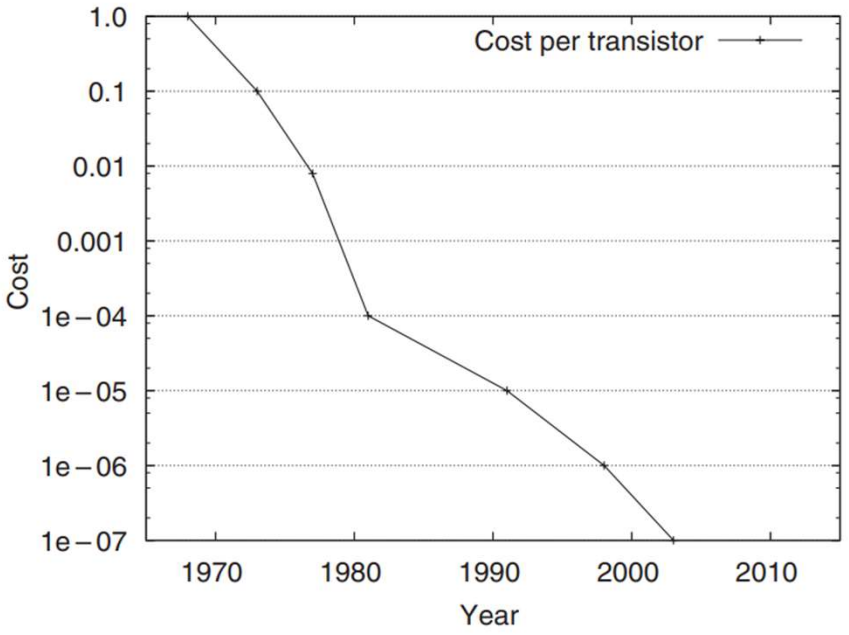


Figure 1.2 The decrease of transistor cost over the years.

SoC components

Processor (CPU): The central processing unit, which executes instructions.

Memory: Various types of memory such as RAM (for temporary storage) and ROM or Flash (for permanent storage).

Input/Output Interfaces: Ports and interfaces for connecting external devices and peripherals.

Peripherals: Additional components like graphics processing units (GPU), digital signal processors (DSP), and other specialized hardware.

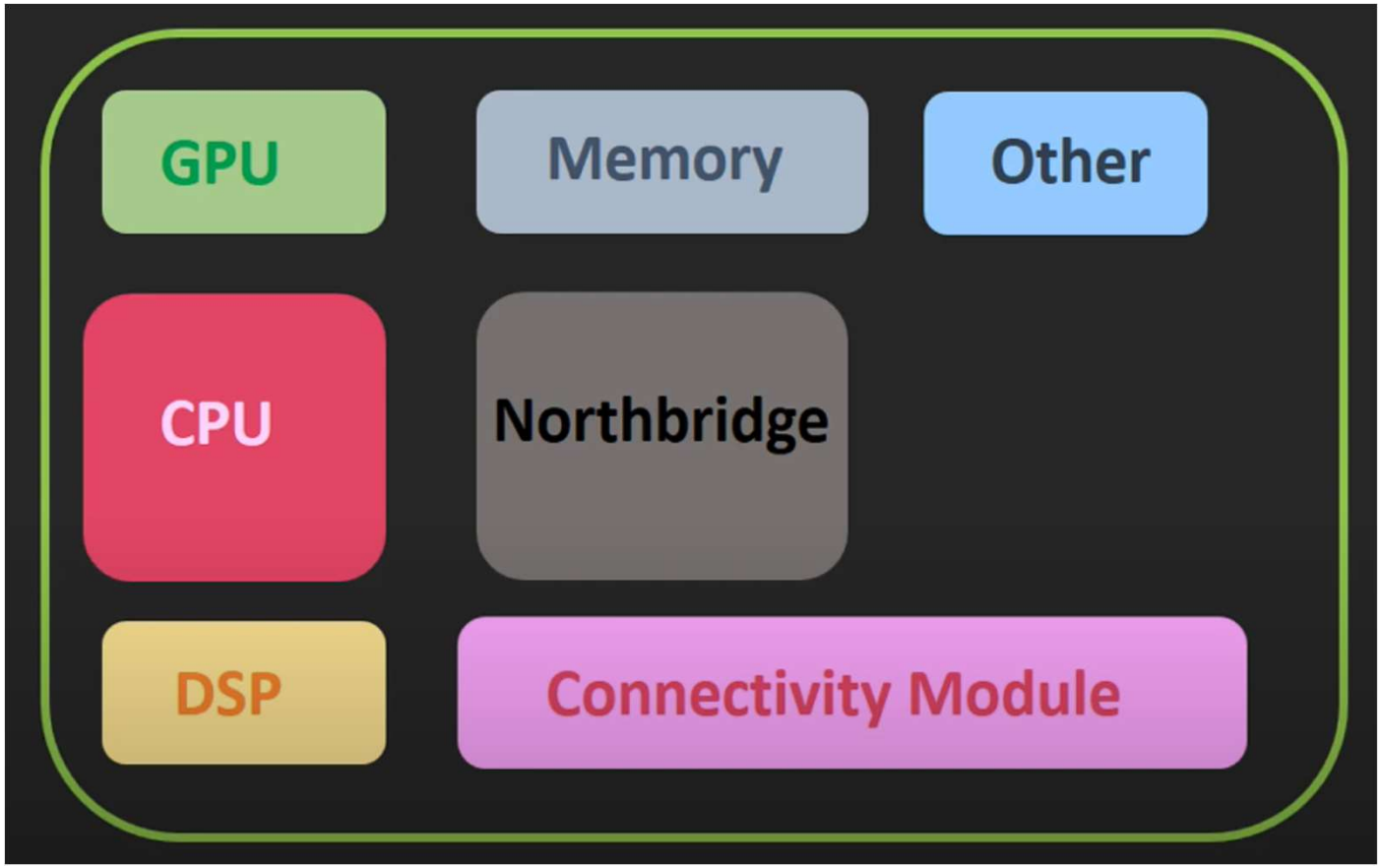
Networking: Controllers for Ethernet, Wi-Fi, Bluetooth, and other communication protocols.

Power Management Circuits: Components to manage the power usage of the SoC.

What is System-on-Chip?

- SoC: More of a System not a Chip
- In addition to IC, SoC consists of software and interconnection structure for integration. SoC may consists of all or some of the following:
- Processor/CPU cores
- On-chip interconnection (busses, network, etc.)
- Analog circuits
- Accelerators or application specific hardware modules
- ASICs Logics
- Software – OS, Application, etc.
- Firmware

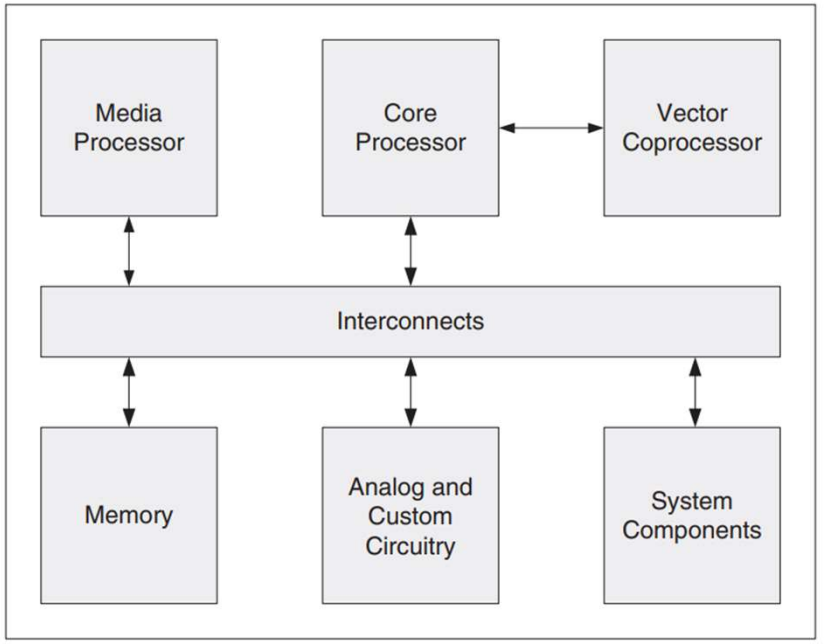
Inside System-on-Chip?



A basic SOC system model

- These include a number of heterogeneous processors interconnected to one or more memory elements with possibly an array of reconfigurable logic.

- SOC also has analog circuitry for managing sensor data and analog - to - digital conversion, or to support wireless data transmission.



Example of SoC

- As an example, an SOC for a smart phone would need to support, in addition to audio input and output capabilities for a traditional phone, Internet access functions and multimedia facilities for video communication, document processing, and entertainment such as games and movies.
- A possible configuration for the elements in previous figure would have the core processor being implemented by several ARM Cortex - A9 processors for application processing, and the media processor being implemented by a Mali - 400MP graphics processor and a Mali - VE video engine.
- The system components and custom circuitry would interface with peripherals such as the camera, the screen, and the wireless communication unit.
- The elements would be connected together by AXI (Advanced eXtensible Interface) interconnects.

HARDWARE AND SOFTWARE: PROGRAMMABILITY VERSUS PERFORMANCE

- A fundamental decision in SOC design is to choose which components in the system are to be implemented in hardware and in software.

TABLE 1.1 Benefits and Drawbacks of Software and Hardware Implementations

	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

Two big concerns in computer architecture

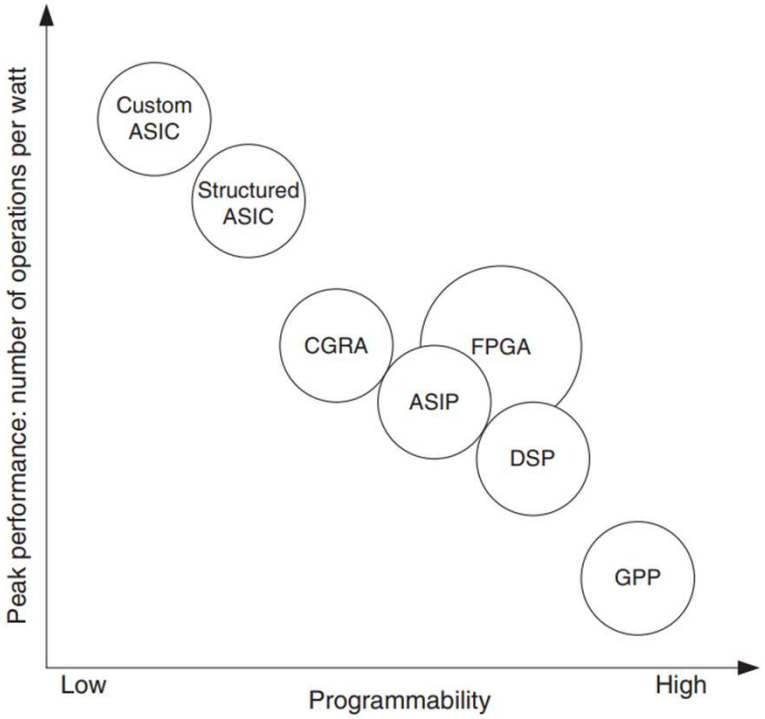
- **Programmability**
  - What range of programs can the architecture run?
  - How difficult is it to write a new program?

- **Performance**
  - Power consumption
  - Latency
  - Throughput

- A software implementation is usually executed on a general - purpose processor (GPP), which interprets instructions at run time. This architecture offers flexibility and adaptability, and provides a way of sharing resources among different applications
- The hardware implementation of the ISA(instruction set arch.) is generally slower and more power hungry than implementing the corresponding function directly in hardware without the overhead of fetching and decoding instructions
- Most software developers use high - level languages and tools that enhance productivity, such as program development environments, optimizing compilers, and performance profilers.
- In contrast, the direct implementation of applications in hardware results in custom application - specific c integrated circuits (ASICs), which often provides high performance at the **expense of programmability** — and hence flexibility, productivity, and cost.

- Given that hardware and software have complementary features, many SOC designs aim to combine the individual benefits of the two. The obvious method is to implement the performance - critical parts of the application in hardware, and the rest in software.
- Custom ASIC hardware and software on GPPs can be seen as two extremes in the technology spectrum with different trade - offs in programmability and performance;
- There are various technologies that lie between these two extremes (The two more well - known ones are application - specific instruction processors (ASIPs) and field - programmable gate arrays (FPGAs).

HARDWARE AND SOFTWARE: PROGRAMMABILITY VERSUS PERFORMANCE



HARDWARE AND SOFTWARE: PROGRAMMABILITY VERSUS PERFORMANCE

- An ASIP is a processor with an instruction set customized for a specific application or domain. Custom instructions efficiently implemented in hardware are often integrated into a base processor with a basic instruction.
- An FPGA typically contains an array of computation units, memories, and their interconnections, and all three are usually programmable in the field by application builders.
- FPGA technology often offers a good compromise: It is faster than software while being more flexible and having shorter development times than custom ASIC hardware implementations.
- Because of the growing demand for reducing the time to market and the increasing cost of chip fabrication, FPGAs are becoming more popular for implementing digital designs.

PROGRAMMABILITY VERSUS PERFORMANCE

- Coarse - Grained Reconfigurable Architecture (**CGRA**) . It contains logic blocks that process byte - wide or multiple byte - wide data, which can form building blocks of data paths.
- **Structured ASIC** . It allows application builders to customize the resources before fabrication. While it offers performance close to that of ASIC, the need for chip fabrication can be an issue.
- Digital Signal Processors (**DSPs**) . The organization and instruction set for these devices are optimized for digital signal processing applications. Like microprocessors, they have a fixed hardware architecture that cannot be reconfigured.

PROCESSOR ARCHITECTURES

- Typically, processors are characterized either by their application or by their architecture (or structure), as shown

TABLE 1.2 Processor Examples as Identified by Function

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

TABLE 1.3 Processor Examples as Identified 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

PROCESSOR ARCHITECTURES

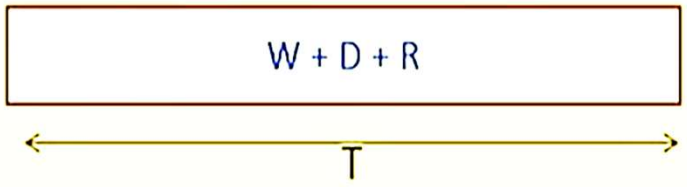
- From the programmer ’ s point of view, sequential processors execute one instruction at a time.
- However, many processors have the capability to execute several instructions concurrently in a manner that is transparent to the programmer,
- The techniques such as pipelining, multiple execution units, and multiple cores.
- Pipelining is a powerful technique that is used in almost all current processor implementations

What is Pipelining?

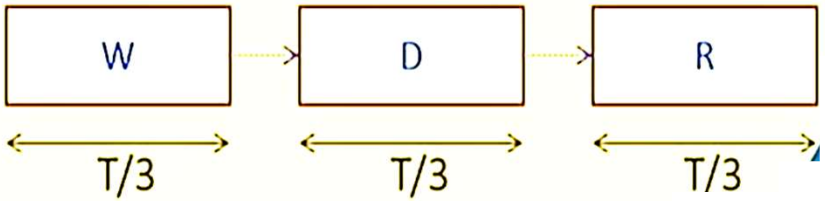
- A mechanism for overlapped execution of several input sets by partitioning some computation into a set of *k* sub-computations (or stages).
  - Very nominal increase in the cost of implementation.
  - Very significant speedup (ideally, *k*).
- Where are pipelining used in a computer system?
  - **Instruction execution**: Several instructions executed in some sequence.
  - **Arithmetic computation**: Same operation carried out on several data sets.
  - **Memory access**: Several memory accesses to consecutive locations are made.

A Real-life Example

- Suppose you have built a machine *M* that can wash (*W*), dry (*D*), and iron (*R*) clothes, one cloth at a time.
  - Total time required is *T*.
- As an alternative, we split the machine into three smaller machines M_W, M_D and M_R, which can perform the specific task only.
  - Time required by each of the smaller machines is *T/3* (say).

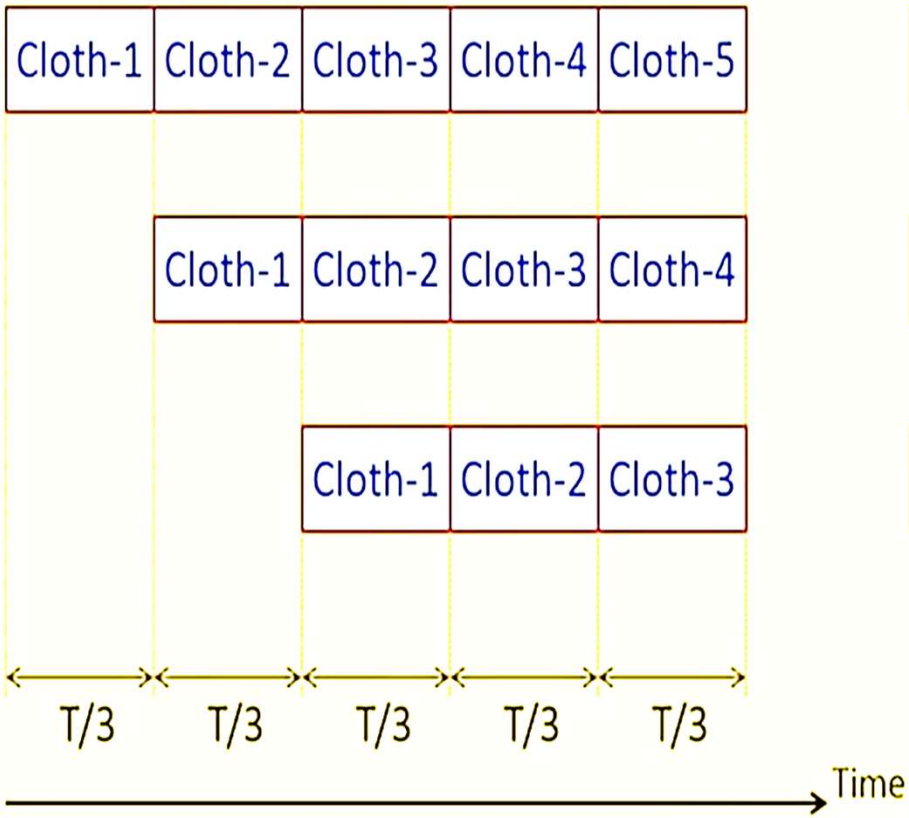


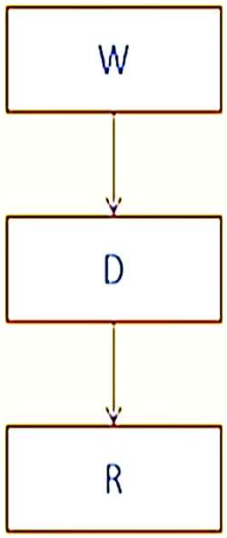
For N clothes, time $T_1 = N.T$



For N clothes, time $T_3 = (2 + N).T/3$

How does the pipeline work?





Finishing times:

- Cloth-1 – 3.T/3
- Cloth-2 – 4.T/3
- Cloth-3 – 5.T/3
- ...
- Cloth-N – (2 + N).T/3

Extending the Concept to Processor Pipeline

- The same concept can be extended to hardware pipelines.
- Suppose we want to attain *k* times speedup for some computation.
  - **Alternative 1**: Replicate the hardware *k* times → cost also goes up *k* times.
  - **Alternative 2**: Split the computation into *k* stages → very nominal cost increase.
- Need for buffering:
  - In the washing example, we need a tray between machines (W & D, and D & R) to keep the temporarily before it is accepted by the next machine.
  - Similarly in hardware pipeline, we need a *latch* between successive stages to hold the intermediate results temporarily.

Simple Sequential Processor

- Sequential processors directly implement the sequential execution model. These processors process instructions sequentially from the instruction stream.
- The next instruction is not processed until all execution for the current instruction is complete and its results have been committed.

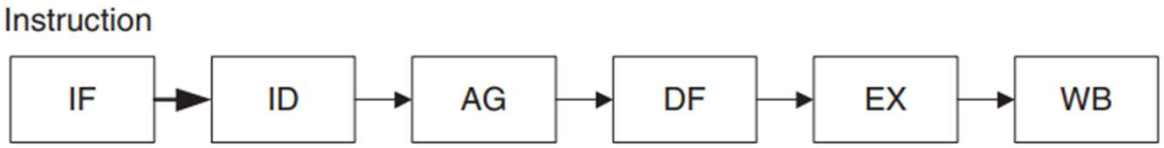
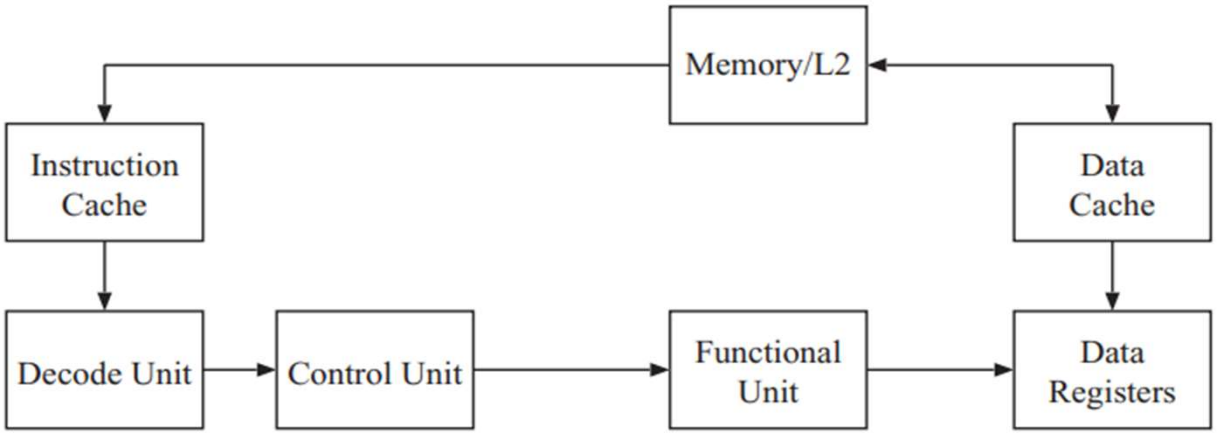


Figure 1.7 Instruction execution sequence.

1. fetching the instruction into the instruction register (IF),
2. decoding the opcode of the instruction (ID),
3. generating the address in memory of any data item residing there (AG),
4. fetching data operands into executable registers (DF),
5. executing the specified operation (EX), and
6. writing back the result to the register file (WB).

Sequential processor model

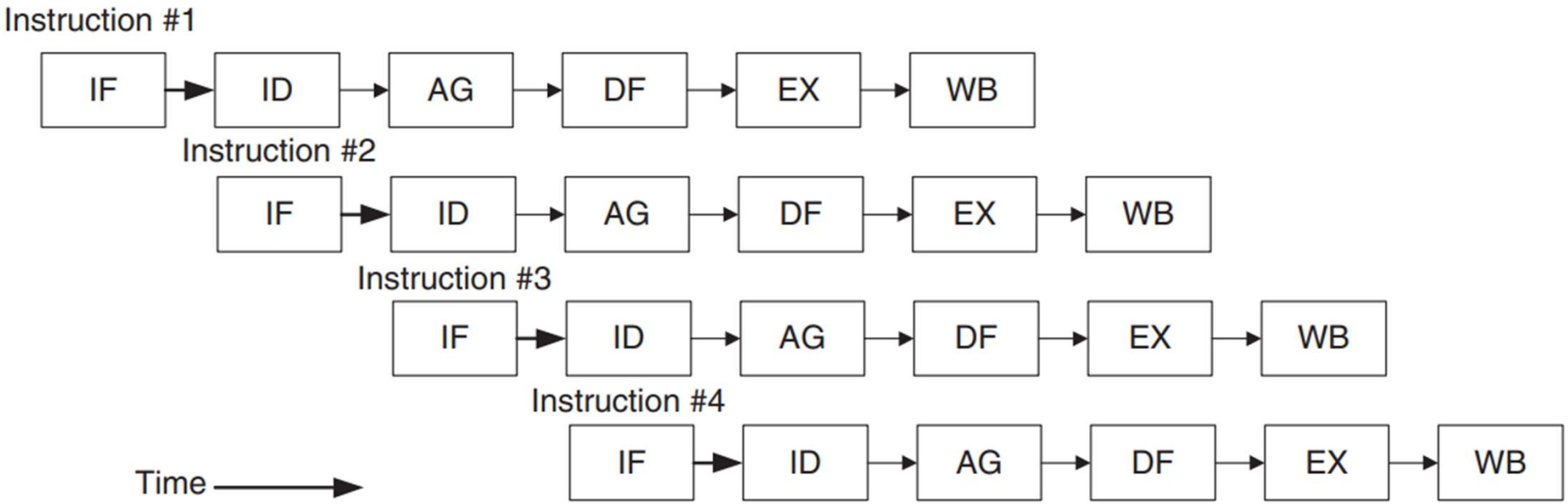
- During execution, a sequential processor executes one or more operations per clock cycle from the instruction stream.
- An instruction is a container that represents the smallest execution packet managed explicitly by the processor.
- One or more operations are contained within an instruction.
- The distinction between instructions and operations is crucial to distinguish between processor behaviors.



Drawback of Simple Sequential Processor

- Executing each instruction sequentially has significant performance drawbacks.
- A considerable amount of time is spent on overhead and not on actual execution or value-creating activities.
- Thus, the simplicity of directly implementing the sequential execution model has significant performance cost.

Instruction timing in a pipelined processor.



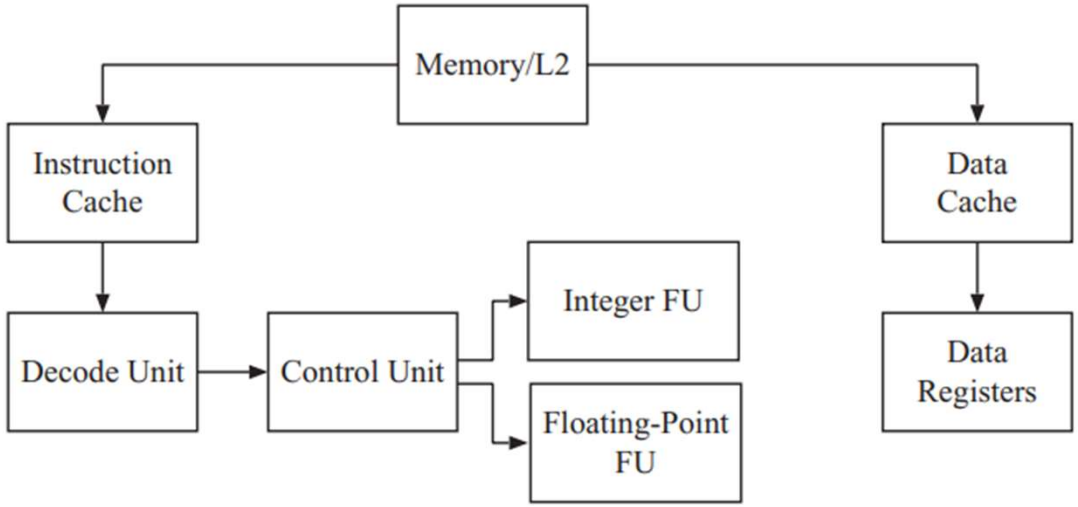
Pipelined Processor

- Pipelining is a straightforward approach to exploiting parallelism that is based on concurrently performing different phases (instruction fetch, decode, execution, etc.) of processing an instruction.

- Pipelining assumes that these phases are independent between different operations and can be overlapped — when this condition does not hold, the processor stalls the downstream phases to enforce the dependency.

Pipelined processor model.

- Requires the processor to go through all stages or phases of the pipeline whether required by a particular instruction or not calls static pipeline .
- A **dynamic pipeline** allows the bypassing of one or more pipeline stages, depending on the requirements of the instruction.
- The more complex dynamic pipelines allow instructions to complete out of (sequential) order.



ILP (instruction level parallelism).

- While pipelining does not necessarily lead to executing multiple instructions at exactly the same time, there are other techniques that do.
- These techniques may use some combination of static scheduling and dynamic analysis to perform concurrently the actual evaluation phase of several different operations, potentially yielding an execution rate of greater than one operation every cycle.
- Two architectures that exploit ILP are superscalar and VLIW (Very long instruction word) processors.

ILP (instruction level parallelism).

- A superscalar processor dynamically examines the instruction stream to determine which operations are independent and can be executed.

- A VLIW processor relies on the compiler to analyze the available operations (OP) and to schedule independent operations into wide instruction words, which then execute these operations in parallel with no further analysis.
-

Instruction timing in a pipelined ILP processor

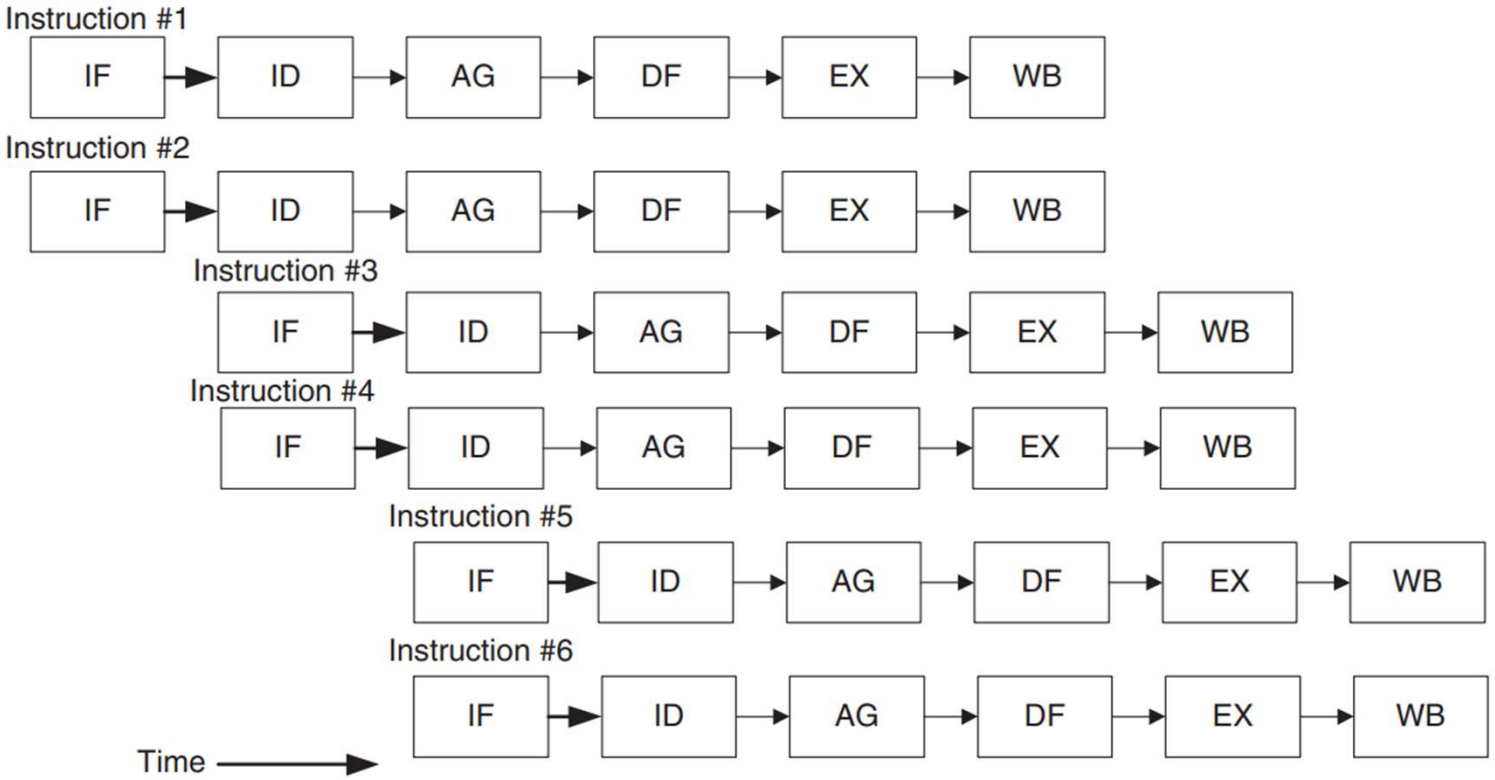
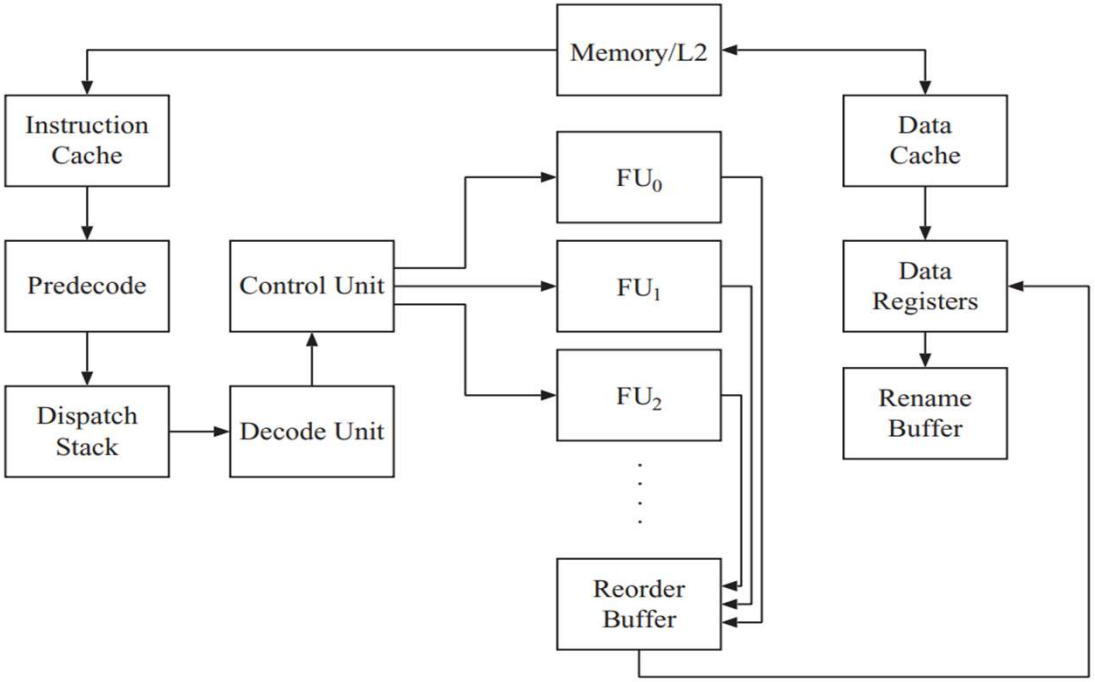


Figure shows the instruction timing of a pipelined superscalar or VLIW processor executing two instructions per cycle. In this case, all the instructions are independent so that they can be executed in parallel.

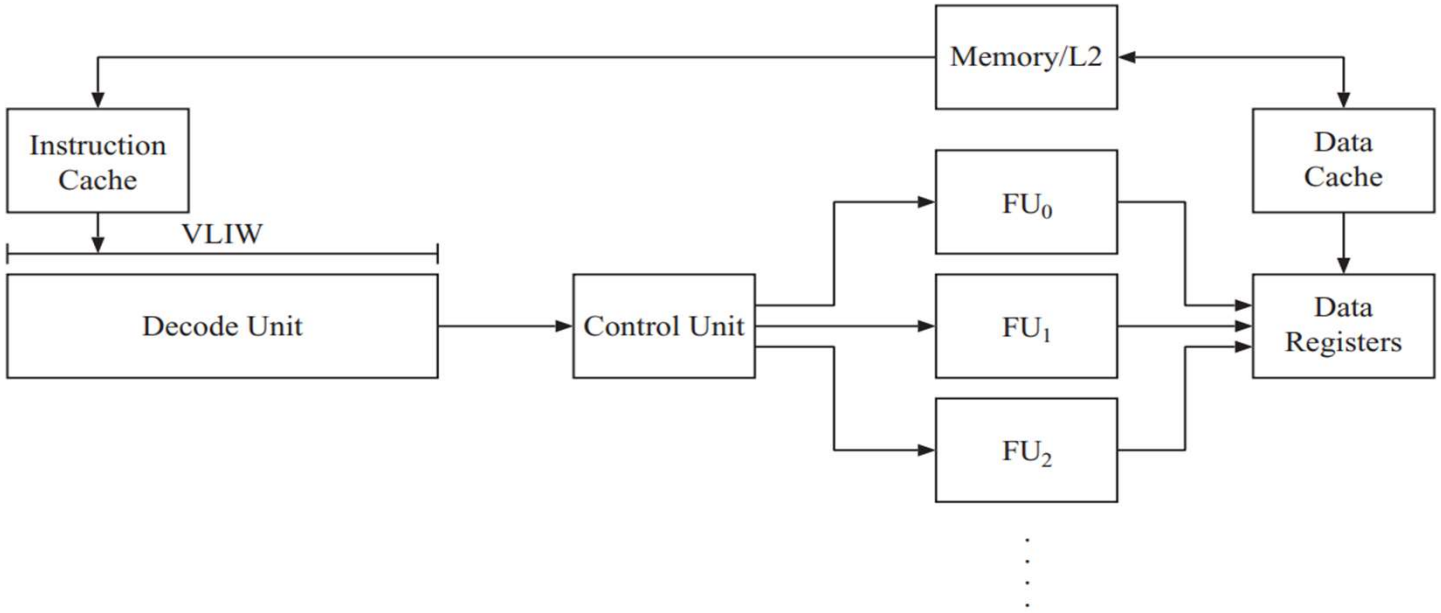
Superscalar processor model

- Dynamic pipelined processors remain limited to executing a single operation per cycle by virtue of their scalar nature.
- This limitation can be avoided with the addition of multiple functional units and a dynamic scheduler to process more than one instruction per cycle
- The most significant advantage of a superscalar processor is that processing multiple instructions per cycle is done transparently to the user.
- Compared to a dynamic pipelined processor, a superscalar processor adds a scheduling instruction window that analyzes multiple instructions from the instruction stream in each cycle.



VLIW Processors

- In contrast to dynamic analyses in hardware to determine which operations can be executed in parallel, VLIW processors (Shown below) rely on static analyses in the compiler.
- VLIW processors are thus less complex than superscalar processors and have the potential for higher performance.



VLIW processors

- VLIW processors are thus less complex than superscalar processors and have the potential for higher performance.
- A VLIW processor executes operations from statically scheduled instructions that contain multiple independent operations.
- Because the control complexity of a VLIW processor is not significantly greater than that of a scalar processor, the improved performance comes without the complexity penalties

Single instruction stream, multiple data streams (SIMD) Architecture

- The SIMD processor is a natural response to the use of certain regular data structures, such as vectors and matrices.
- From the view of an assembly - level programmer, programming SIMD architecture appears to be very similar to programming a simple processor except that some operations perform computations on aggregate data.
- The SIMD class of processor architecture includes both array and vector processor.

SIMD Processing

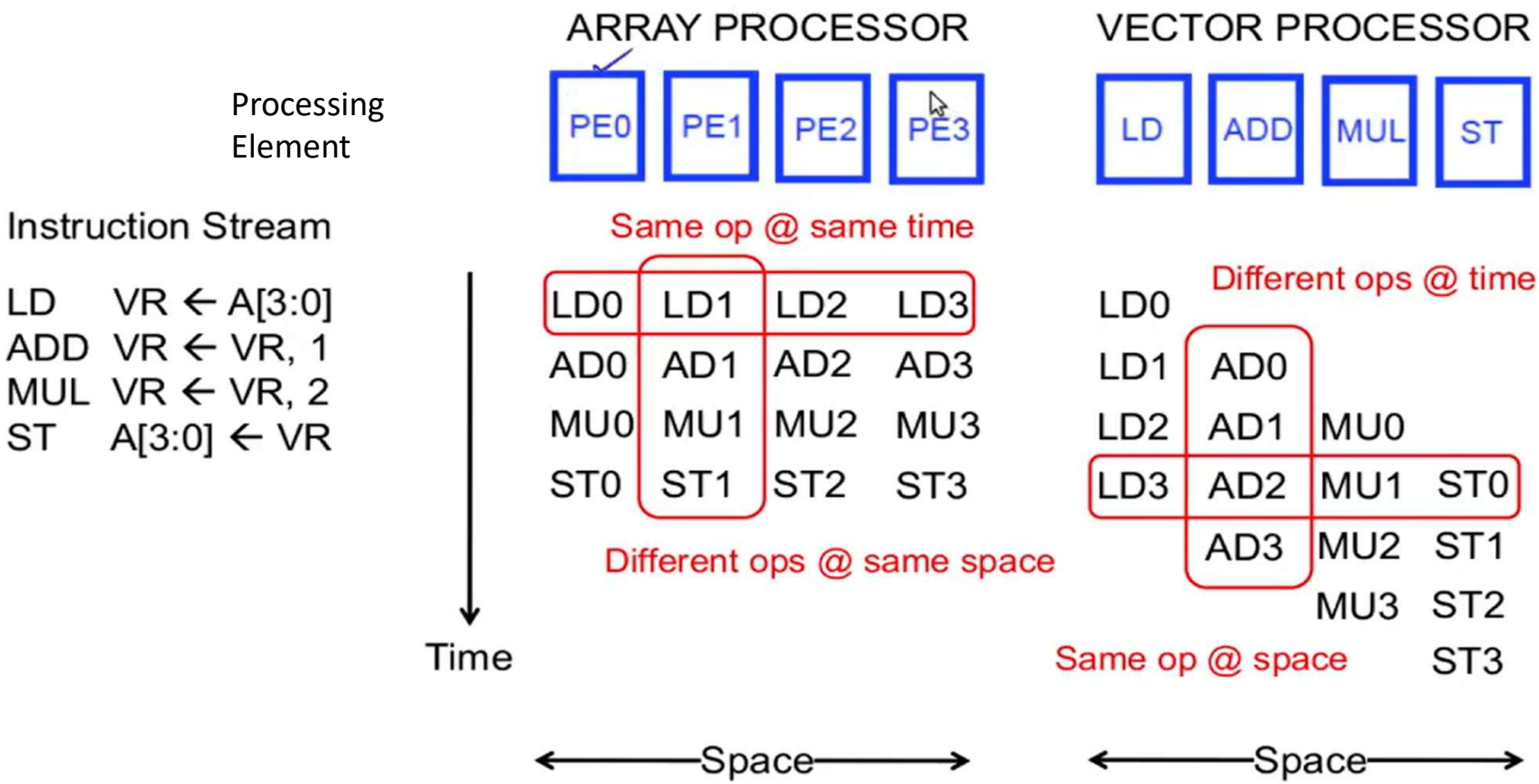
- Single instruction operates on multiple data elements
  - In time or in space
- Multiple processing elements

- Time-space duality

  - Array processor: Instruction operates on multiple data elements at the same time using different spaces

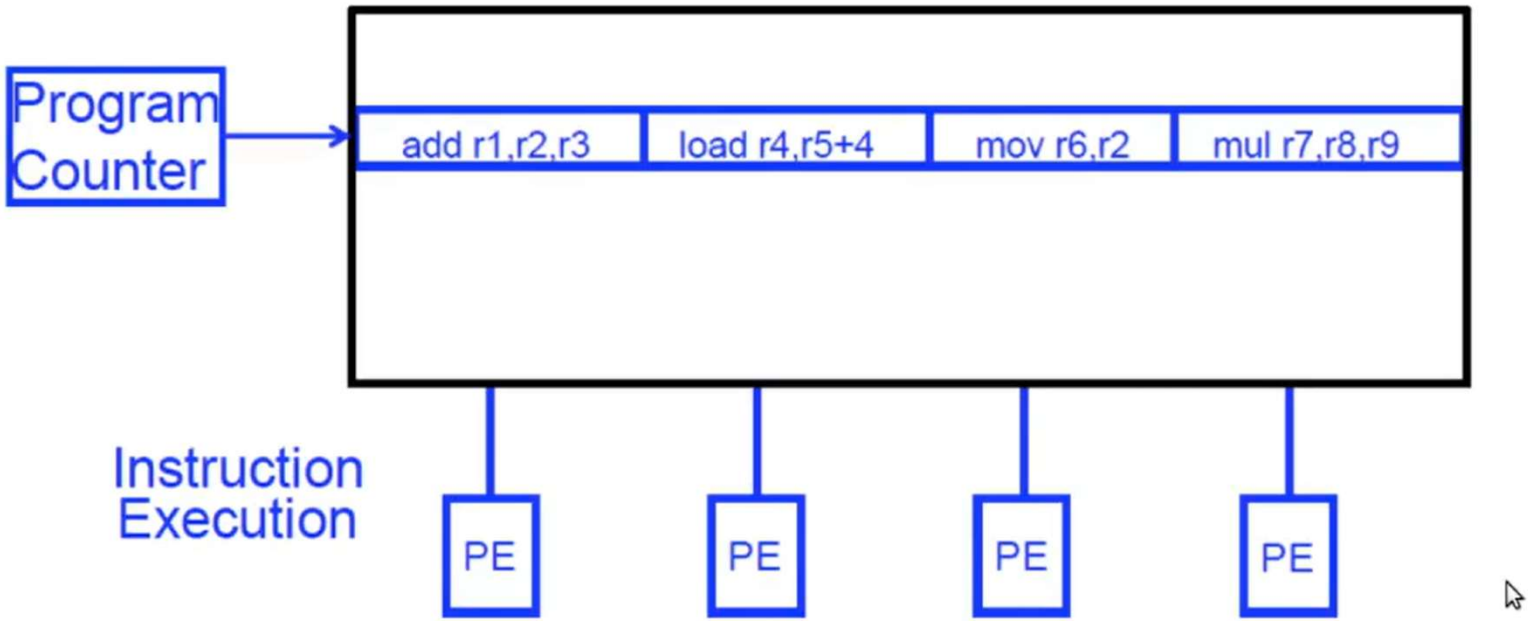
  - Vector processor: Instruction operates on multiple data elements in consecutive time steps using the same space

Array vs. Vector Processors



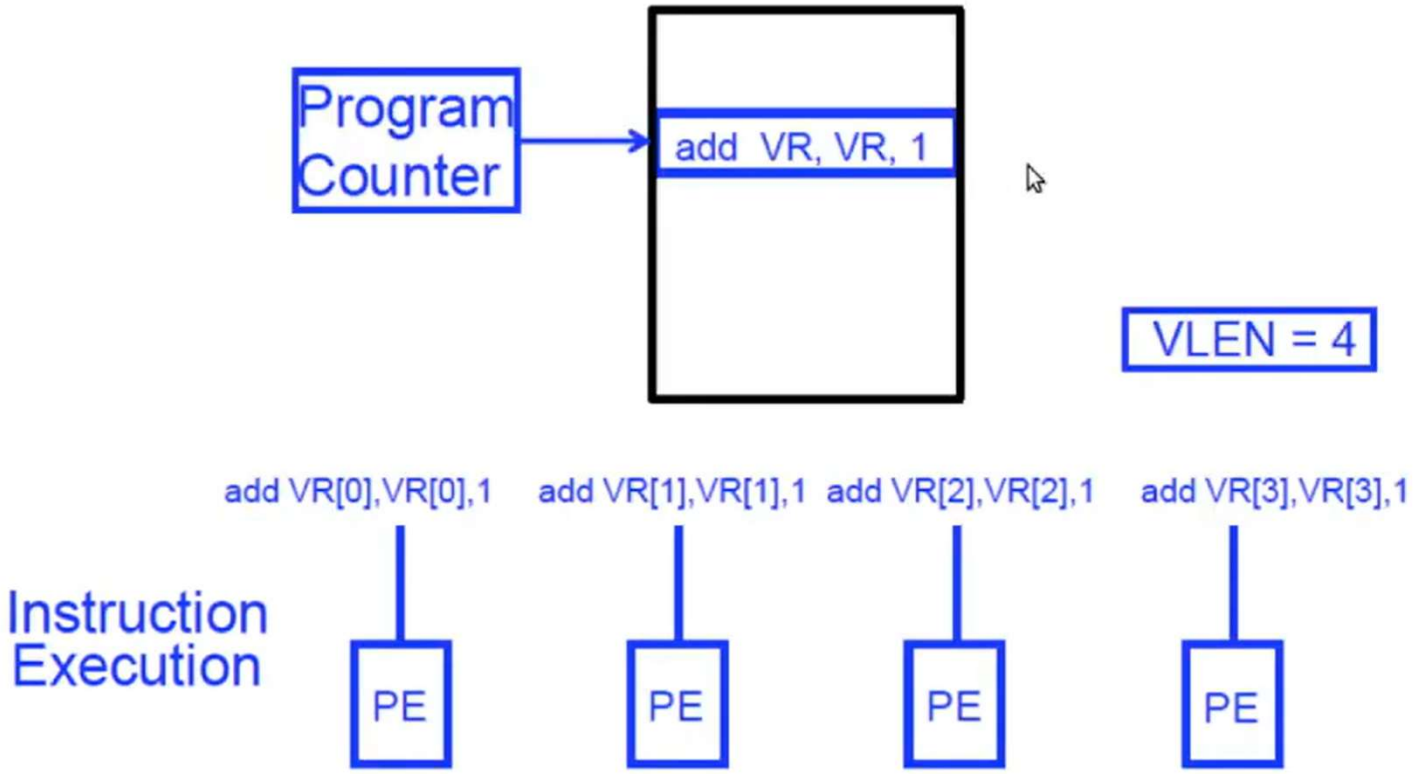
SIMD Array Processing vs. **VLIW**

- VLIW: Multiple independent operations packed together by the compiler

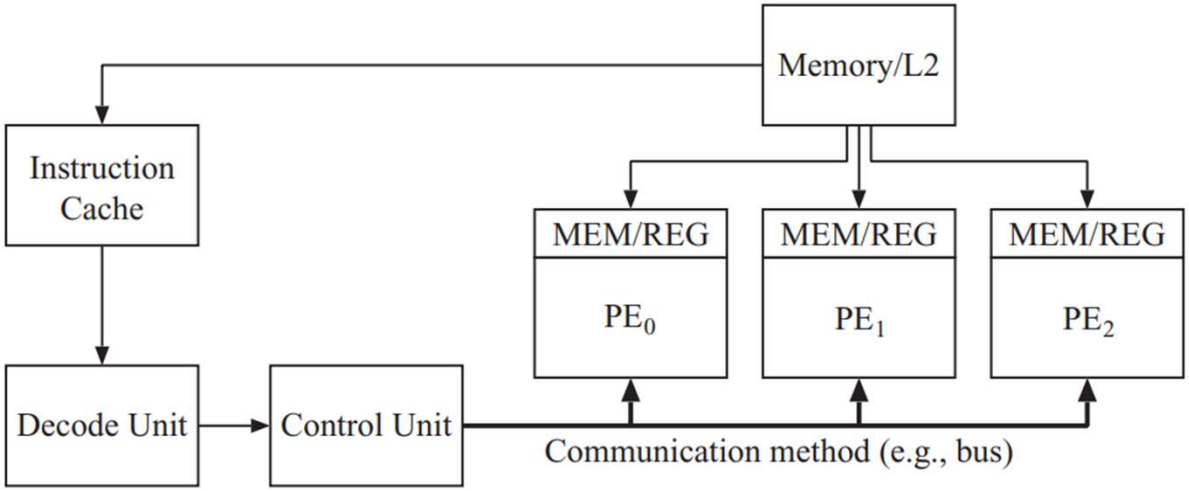


SIMD Array Processing vs. VLIW

- Array processor: Single operation on multiple (different) data elements



SIMD Architectures (Array processor model)



- An array processor consists of many interconnected processor elements, each having their own local memory space.
- A vector processor consists of a single processor that references a global memory space and has special function units that operate on vectors.
- The array processor (Figure) is a set of parallel processor elements connected via one or more networks, possibly including local and global interelement communications and control communications.
- Processor elements operate in lockstep in response to a single broadcast instruction from a control processor (SIMD)

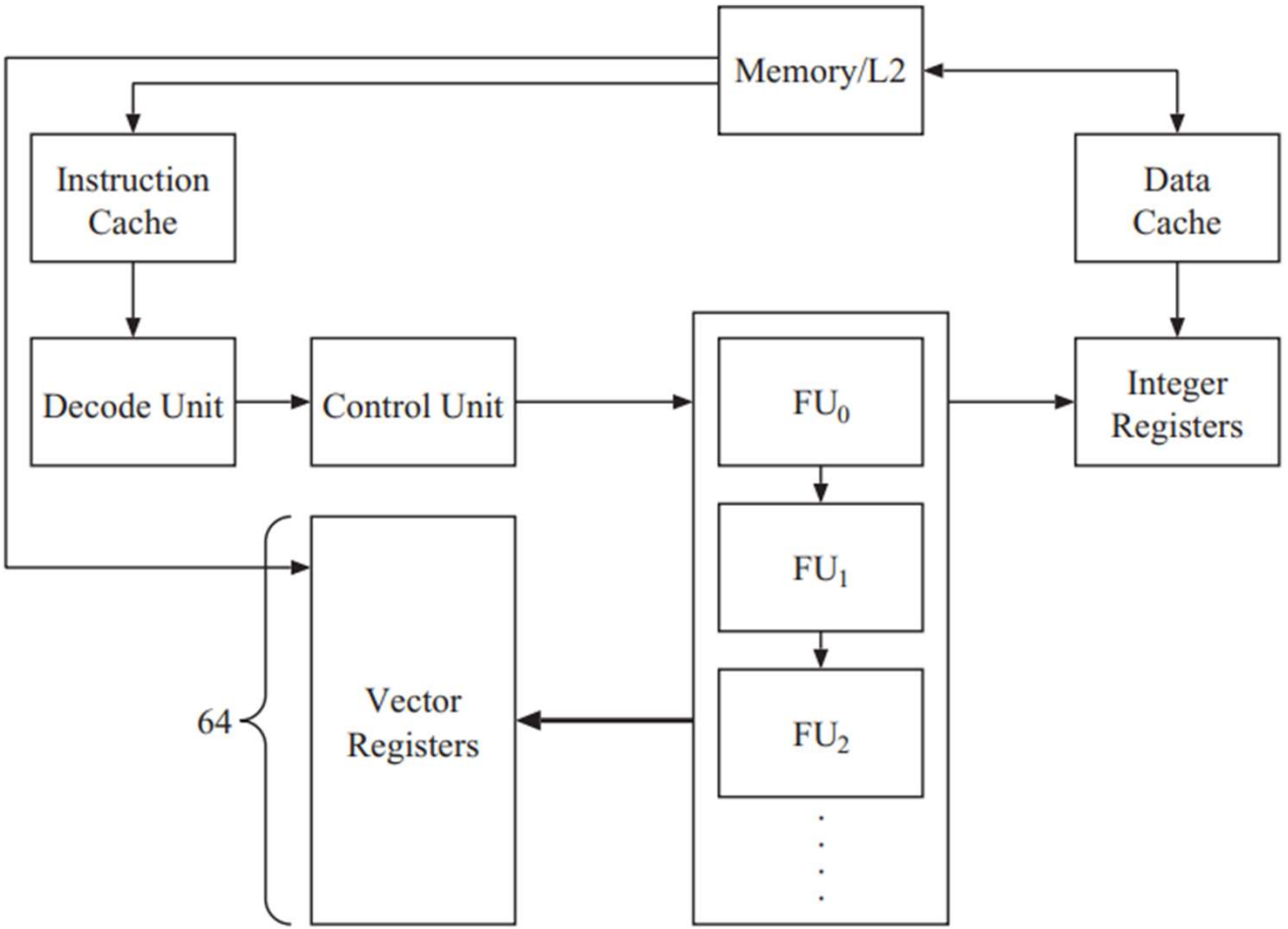
SIMD Architectures (Array processor model)

- Each processor element (PE) has its own private memory, and data are distributed across the elements in a regular fashion that is dependent on both the actual structure of the data and also the computations to be performed on the data.
- Direct access to global memory or another processor element ' s local memory is expensive, so intermediate values are propagated through the array through local interprocessor connections.
- This requires that the data be distributed carefully so that the routing required to propagate these values is simple and regular.
- It is sometimes easier to duplicate data values and computations than it is to support a complex or irregular routing of data between processor elements.

Vector processor

- A is a single processor that resembles a traditional single stream processor, except that some of the function units (and registers) operate on vectors — sequences of data values that are seemingly operated on as a single entity.
- These function units are deeply pipelined and have high clock rates.
- While the vector pipelines often have higher latencies compared with scalar function units, the rapid delivery of the input vector data elements, together with the high clock rates, results in a significant throughput.

Vector processor model



Multiprocessors

- Multiple processors can cooperatively execute to solve a single problem by using some form of interconnection for sharing results.
- Each processor executes completely independently, although most applications require some form of synchronization during execution to pass information and data between processors.
- Since the multiple processors share memory and execute separate program tasks (MIMD [multiple instruction stream, multiple data stream]), their proper implementation is significantly more complex then the array processor.
- The interconnection network in the multiprocessor passes data between processor elements and synchronizes the independent execution streams between processor elements.

Multiprocessors

- When the memory of the processor is distributed across all processors and only the local processor element has access to it, all data sharing is performed explicitly using messages, and all synchronization is handled within the message system.
- When the memory of the processor is shared across all processor elements, synchronization is more of a problem — certainly, messages can be used through the memory system to pass data and information between processor elements, but this is not necessarily the most effective use of the system.
- When communications between processor elements are performed through a shared memory address space — either global or distributed between processor elements (called distributed shared memory to distinguish it from distributed memory) — there are two significant problems that arise

Multiprocessors

- The first is maintaining memory consistency: the programmer - visible ordering effects on memory references, both within a processor element and between different processor elements. This problem is usually solved through a combination of hardware and software techniques.
- The second is cache coherency — the programmer - invisible mechanism to ensure that all processor elements see the same value for a given memory location. This problem is usually solved exclusively through hardware techniques.

Multiprocessors

- The implementation of a distributed memory machine is far easier than the implementation of a shared memory machine when memory consistency and cache coherency are taken into account.
- However, programming a distributed memory processor can be much more difficult since the applications must be written to exploit and not to be limited by the use of message passing as the only form of communication between processor elements.
- Although, shared memory processor can take advantage of whatever communications paradigm is appropriate for a given communications requirement, and can be much easier to program.

MEMORY AND ADDRESSING

- SOC applications vary significantly in memory requirements. In one case, the memory structure can be as simple as the program residing entirely in an on - chip read - only memory (ROM), with the data in on - chip RAM
- In another case, the memory system might support an elaborate operating system requiring a large off - chip memory (system on a board), with a memory management unit and cache hierarchy.

Why not simply include memory with the processor on the die?

1. It improves the accessibility of memory, improving both memory access time and bandwidth.

2. It reduces the need for large cache.

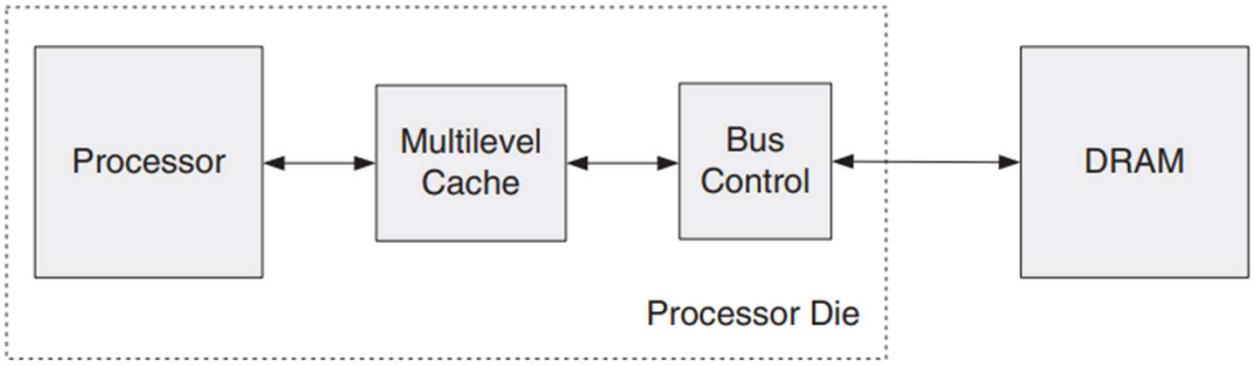
3. It improves performance for memory - intensive applications.

Limitations to include memory with the processor on the die

- The first problem is that DRAM memory process technology differs from standard microprocessor process technology, and would cause some sacrifice in achievable bit density.
- The second problem is more serious: If memory were restricted to the processor die, its size would be correspondingly limited.
- Applications that require very large real memory space would be crippled.

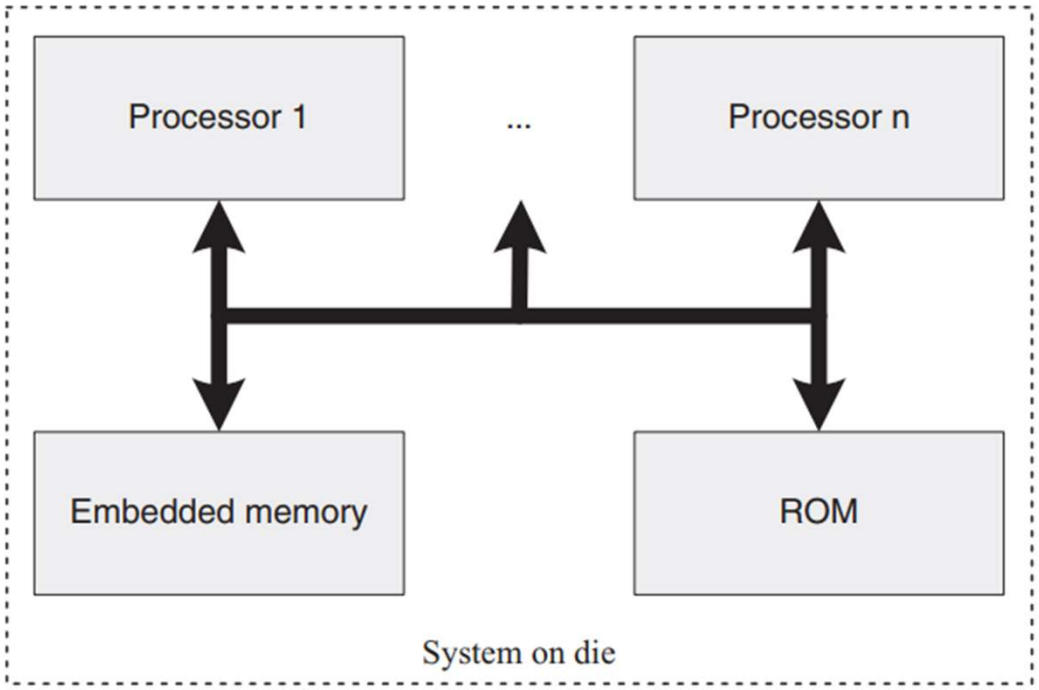
Processors with memory off - die

- Thus, the conventional processor die model has evolved (Figure) to implement multiple robust homogeneous processors sharing the higher levels of a two - or three - level cache structure with the main memory off - die, on its own multi-die module.
- From a design complexity point of view, this has the advantage of being a “ universal ” solution: One implementation fits all applications, although not necessarily equally well.



System on a chip: processors and memory.

- For specific applications, whose memory size can be bounded, we can implement an integrated memory SOC.



SOC Memory Considerations

It is important for SOC designers to consider whether to put RAM and ROM on - die or off - die.

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

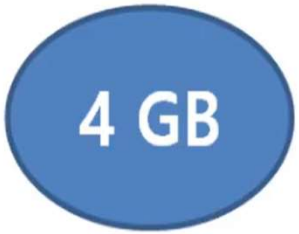
Addressing: The Architecture of Memory

- The user's view of memory primarily consists of the addressing facilities available to the programmer.
- Some of these facilities are available to the application programmer and some to the operating system programmer.
- Virtual memory enables programs requiring larger storage than the physical memory to run and allows separation of address spaces to protect unauthorized access to memory regions when executing multiple application programs.

Virtual Memory

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard disk that's set up to emulate the computer's RAM.

It is illusion to user.

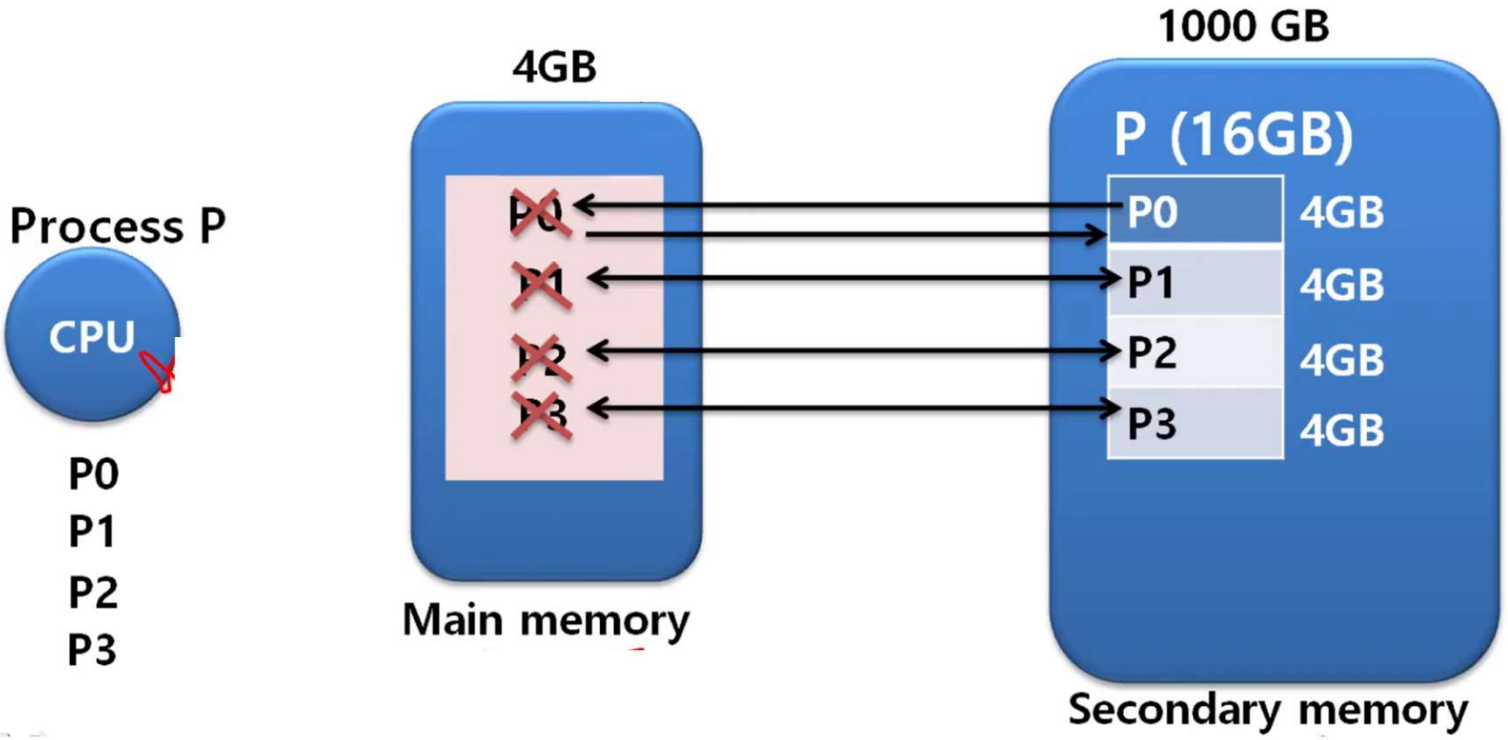


Real memory of computer

User use virtual memory

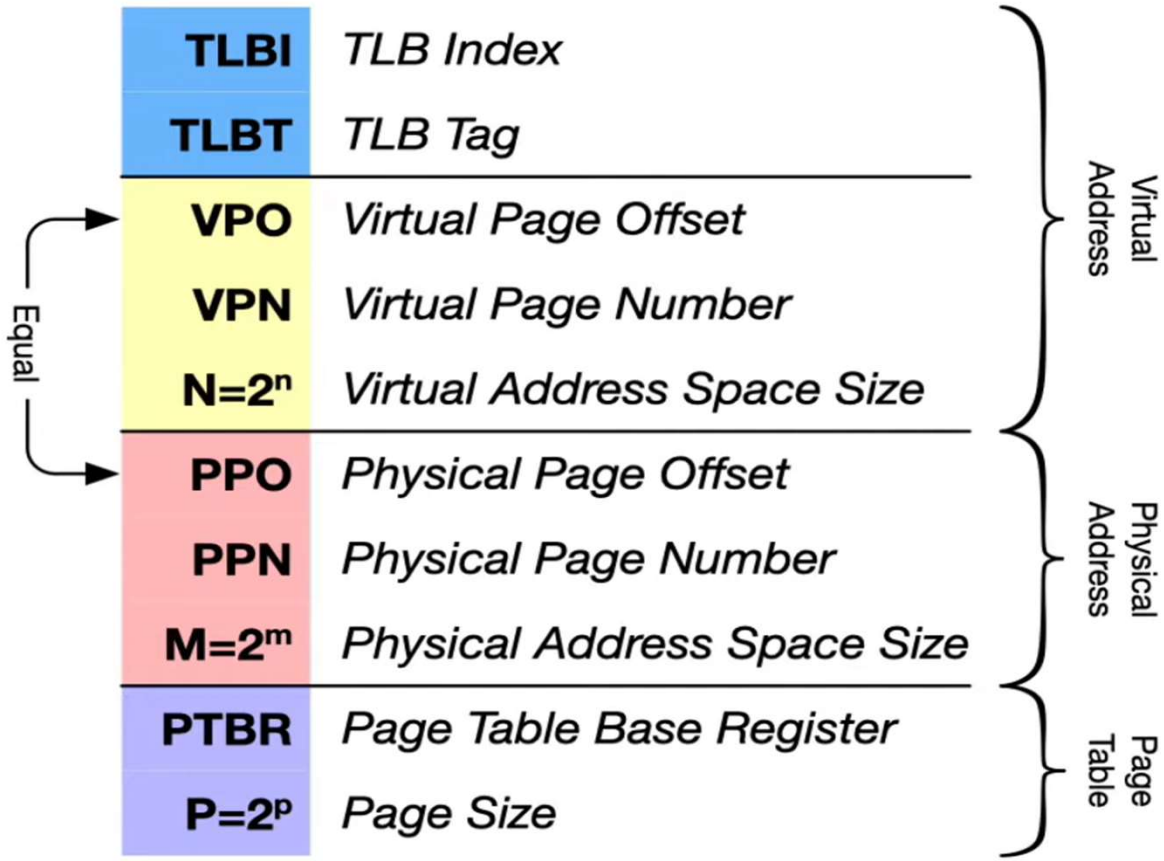
A programmer can write a program which requires more memory than the capacity of main memory such a program is executed by virtual memory techniques

Virtual Memory

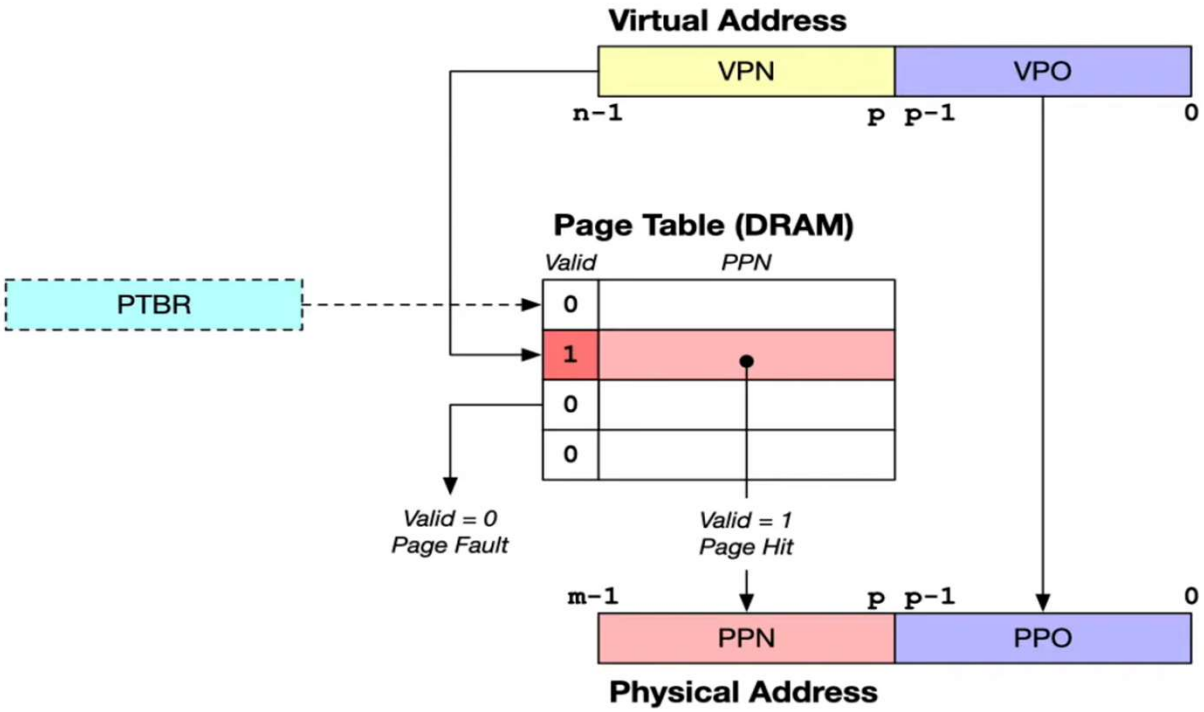


VM address translation

- Virtual memory is often supported by a memory management unit.
-
  - Virtual Address Space: $V = \{0,1,\ldots,N-1\}$
  - Physical Address Space: $P = \{0,1,\ldots,M-1\}$
  - Address Translation is a mapping
    - $M : V \rightarrow P \cup \{\varnothing\}$
    - For virtual address a:
      - $M(a) = a'$ if data at VA a is at PA $a' \in P$
      - $M(a) = \varnothing$ if data at VA a is not DRAM (invalid or on disk)


TLBI
TLB Index
TLBT
TLB Tag
VPO
Virtual Page Offset
VPN
Virtual Page Number
N=2^n
Virtual Address Space Size
PPO
Physical Page Offset
PPN
Physical Page Number
M=2^m
Physical Address Space Size
PTBR
Page Table Base Register
P=2^p
Page Size
Equal
Virtual Address
Physical Address
Page Table

Address Translation with Page Table

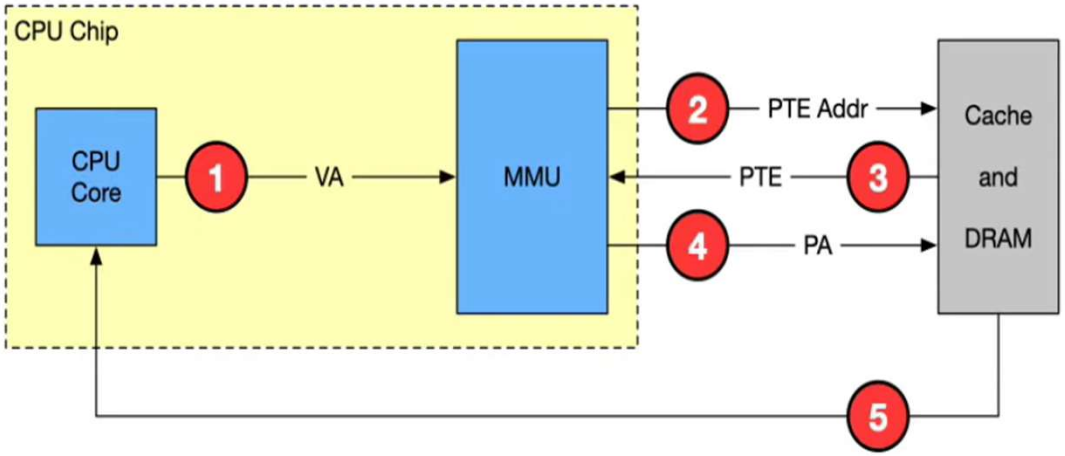


TLBI	*TLB Index*
TLBT	*TLB Tag*
VPO	*Virtual Page Offset*
VPN	*Virtual Page Number*
N=2^n	*Virtual Address Space Size*
PPO	*Physical Page Offset*
PPN	*Physical Page Number*
M=2^m	*Physical Address Space Size*
PTBR	*Page Table Base Register*
P=2^p	*Page Size*

Equal

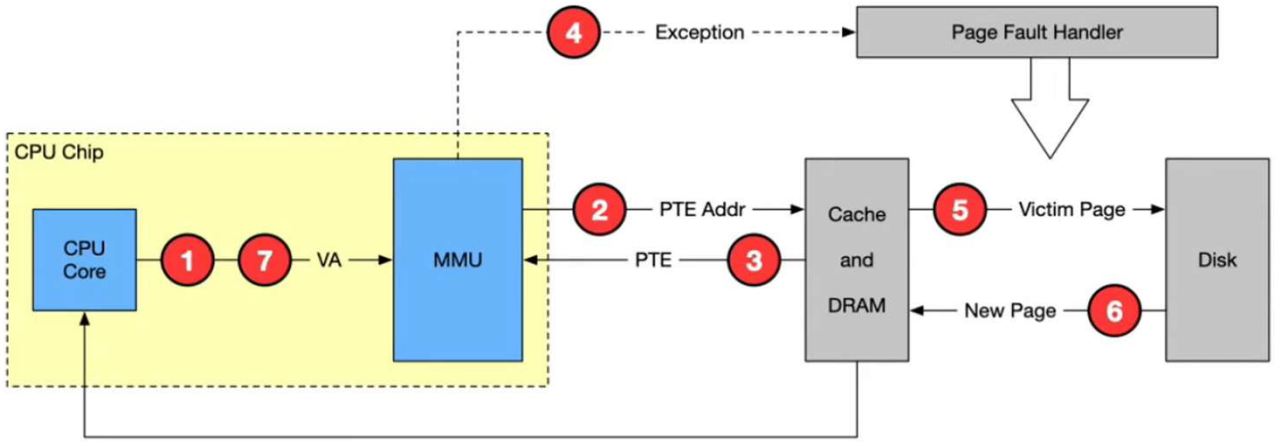
Address Translation: Page Hit

1. CPU sends VA to MMU
2. MMU requests PTE from page table
3. PTE returned from memory
4. MMU sends PA to cache/memory
5. Cache/memory returns data to CPU



Address Translation: Page Fault

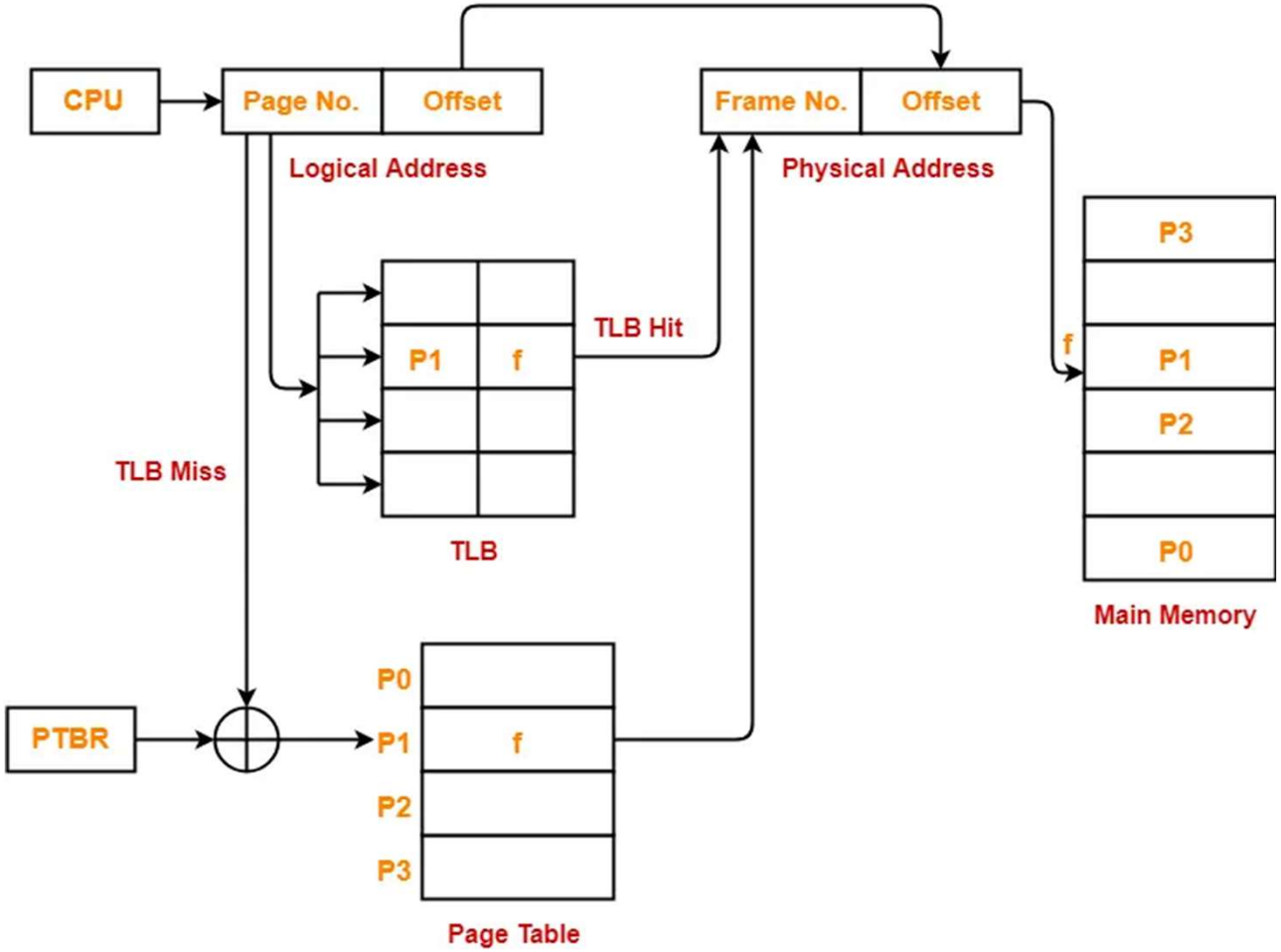
1. CPU sends VA to MMU
2. MMU requests PTE from page table
3. PTE returned from memory
4. Valid bit zero; MMU triggers page fault exception
5. Handler identifies victim; if dirty, pages it out to disk
6. Handler pages in new page, updates PTE
7. Handler returns to original process, restarts faulting instruction



Speeding up Address Translation

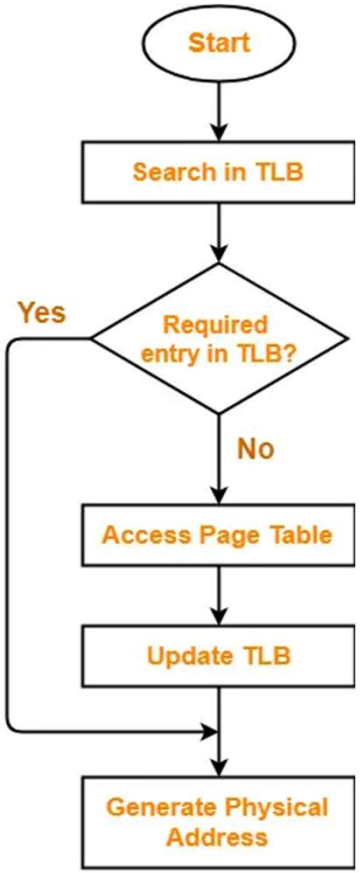
- Goal: Want extremely fast page table entry lookup
- Problem: PTEs cached in L1 like any memory word
  - PTEs may be evicted by other data references
  - PTE hit still requires a small L1 delay
- Solution: Translation Lookaside Buffer (TLB)
  - Small set-associative hardware cache in MMU
  - Maps virtual page numbers (VPNs) to physical page numbers (PPNs)
  - Contains complete page table entries for small number of pages

Translation Lookaside Buffer (TLB)



Translating Logical Address into Physical Address

Steps of translating logical address into physical address



Flowchart

A single level paging using TLB, the effective access time is given as

Effective Access Time =

Hit ratio of TLB x { Access time of TLB + Access time of main memory }

+

Miss ratio of TLB x { Access time of TLB + 2 x Access time of main memory }

TLB Access time

A paging scheme uses a Translation Look-aside Buffer (TLB). A TLB-access takes 10 ns and a main memory access takes 50 ns. What is the effective access time (in ns) if the TLB hit ratio is 90% and there is no page-fault?

A paging scheme uses a Translation Look-aside Buffer (TLB). A TLB-access takes 10 ns and a main memory access takes 50 ns. What is the effective access time (in ns) if the TLB hit ratio is 90% and there is no page-fault?

Effective access time = hit ratio * (TLB access time + Memory access time) + miss ratio * (TLB access time + Page table access time + Memory access time)

=0.9 * (10 + 50) + 0.1 (10 + 50 + 50)
= 54 + 11 = 65

SYSTEM - LEVEL INTERCONNECTION

- SOC technology typically relies on the interconnection of predesigned circuit modules (known as intellectual property (IP) blocks) to form a complete system.

- In this way, the design task is raised from a circuit level to a system level.

- Central to the system – level performance and the reliability of the finished product is dependent on the method of interconnection used.

- A well - designed interconnection scheme should have efficient communication protocols

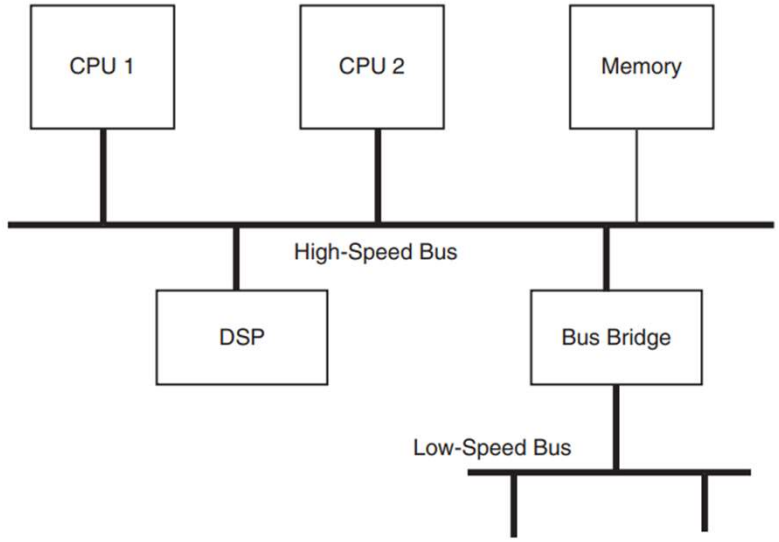
SYSTEM - LEVEL INTERCONNECTION

- It should provide efficient communication between different modules maximizing the degree of parallelism.

- SOC interconnect methods can be classified into two main approaches:
  - **buses and**
  - **network - on - chip**

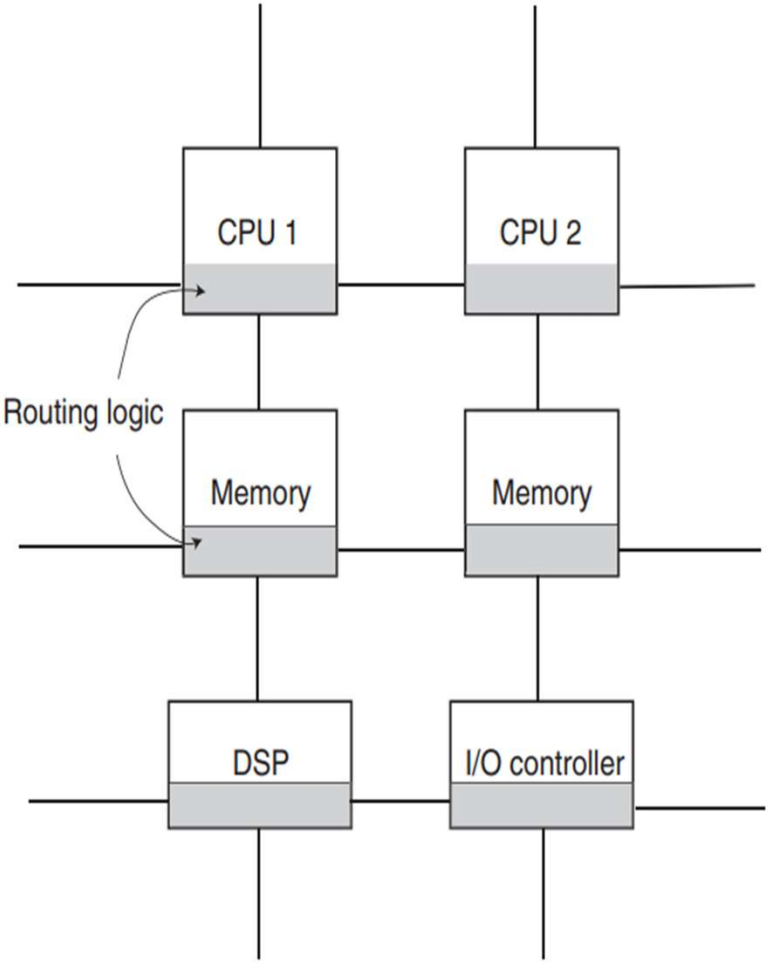
Bus - Based Approach

- With the bus - based approach, IP blocks are designed to conform to published bus standards (AMBA or CoreConnect).
- Communication between modules is achieved through the sharing of the physical connections of address, data, and control bus signals. Usually, two or more buses are employed in a system, organized in a hierarchical fashion.
- To optimize system - level performance and cost, the bus closest to the CPU has the highest bandwidth, and the bus farthest from the CPU has the lowest bandwidth.



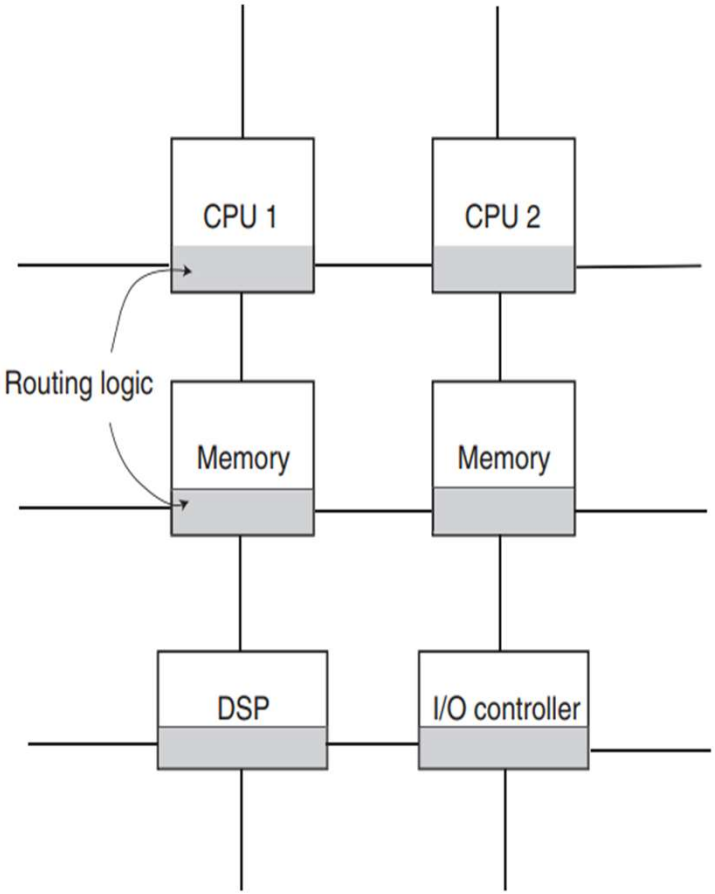
Network - on - Chip Approach

- A network - on - chip system consists of an array of switches, either dynamically switched as in a crossbar or statically switched as in a mesh.
- The crossbar approach uses asynchronous channels to connect synchronous modules that can operate at different clock frequencies.
- This approach has the advantage of higher throughput than a bus - based system while making integration of a system with multiple clock domains easier.



Network - on - Chip Approach

- This interconnect scheme is based on a two - dimensional mesh topology.
- All communications between switches are conducted through data packets, routed through the router interface circuit within each node.
- Since the interconnections between switches have a fixed distance, interconnect - related problems such as wire delay and cross talk noise are much reduced.



Interconnect Models for Different SOC Examples

SOC	Application	Interconnect Type
ClearSpeed CSX600 [59]	High Performance Computing	ClearConnect bus
NetSilicon NET +40 [184]	Networking	Custom bus
NXP LH7A404 [186]	Networking	AMBA bus
Intel PXA27x [132]	Mobile/wireless	PXBus
Matsushita i-Platform [176]	Media	Internal connect bus
Emulex InSpeed SOC320 [130]	Switching	Crossbar switch
MultiNOC [172]	Multiprocessing system	Network-on-chip

AN APPROACH FOR SOC DESIGN

Two important ideas in a design process are

- figuring out the requirements and specifications, and
- iterating through different stages of design toward an efficient and effective completion.
- **Requirements and Specifications**

- **Design Iteration –**
  - Initial Design –
  - Optimized Design

Requirements and Specifications

The system requirements are the largely externally generated criteria for the system. They may come from competition, from sales insights, from customer requests, from product profitability analysis, or from a combination. Some of the factors the designer considers in determining requirements include

- compatibility with previous designs or published standards,
- reuse of previous designs
- customer requests/complaints
- sales reports
- cost analysis
- competitive equipment analysis
- trouble reports (reliability) of previous products and competitive products.

Indeed, requirements can frequently be unrealistic: " I want it fast, I want it cheap, and I want it now! "

Design Iteration

- Design is always an iterative process.
- So, the obvious question is how to get the very first, initial design.
- This is the design that we can then iterate through and optimize according to the design criteria.

Initial Design

- This is the first design that shows promise in meeting the key requirements, while other performance and cost criteria are not considered.
- For instance, processor or memory or input/output (I/O) should be sized to meet high - priority real - time constraints.
- Promising components and their parameters are selected for better performance.
- It is simplified model of the expected computational capacity or data bandwidth capability.

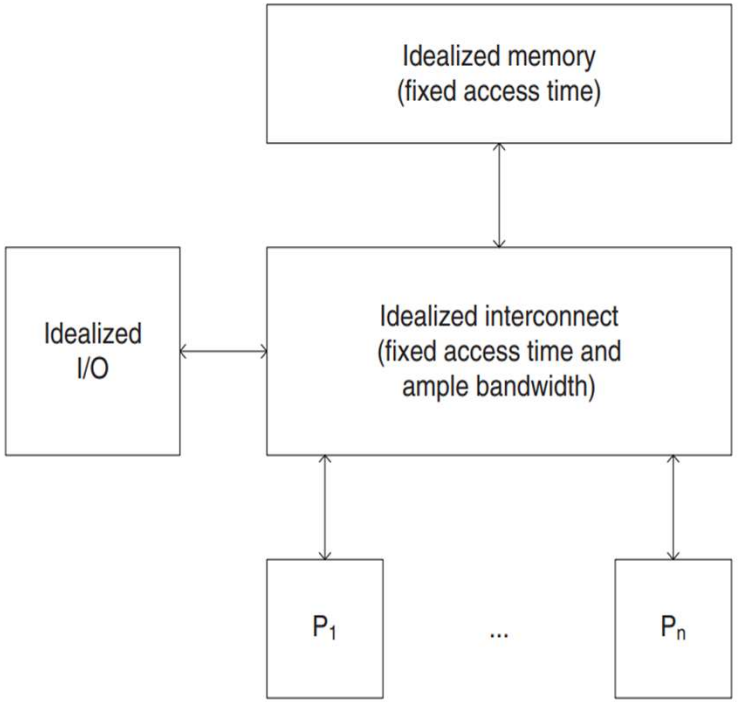


Fig: Idealized SOC components.

Steps for the SOC initial design process.

1. Understand functional, cost, and real-time requirements.
2. Identify key requirements that the design must meet.
3. Prioritize the selection of processor, memory, and interconnect components based on their impact on key requirements.
4. Evaluate whether key requirements are met.
5. If yes, then initial design is completed.
6. If no, then try a different component selection; go to step 3.

Optimized Design:

- Once the base performance (or area) requirements are met and the base functionality is ensured, then the goal is to minimize:
  - the cost (area) and/or
  - the power consumption or
  - the design effort required to complete the design
- This is the iterative step of the process.
- The first steps of this process use higher - fidelity tools (simulations, trial layouts, etc.) to ensure that the initial design actually does satisfy the design specifications and requirements.
- The later steps refine, complete, and improve the design according to the design criteria.

Design Iteration

- Usually, designs are driven by either

(1) a specific real - time requirement, after which functionality and cost become important, or

the real - time constraint is provided by I/O consideration, which the processor – memory – interconnect system must meet. The I/O system then determines the performance, and any excess capability of the remainder of the system is usually used to add functionality to the system.

(2) functionality and/ or throughput under cost – performance constraints.

the object is to improve task throughput while minimizing the cost. Throughput is limited by the most constrained component.

so the designer must fully understand the trade - offs at that point.

Design Iteration

- The basic difference between processor architecture and system architecture is that the system adds another layer of complexity, and the complexity of these systems limits the cost savings.

- Historically, the computer is a single processor plus a memory.

- As long as this is fixed (within broad tolerances), implementing that processor on one or more silicon die does not change the design complexity.

- Once die densities enable a scalar processor to fit on a chip, the complexity issue changes.

System Architecture and Complexity

As transistor densities significantly improve after this point, there are obvious processor extension strategies to improve performance:

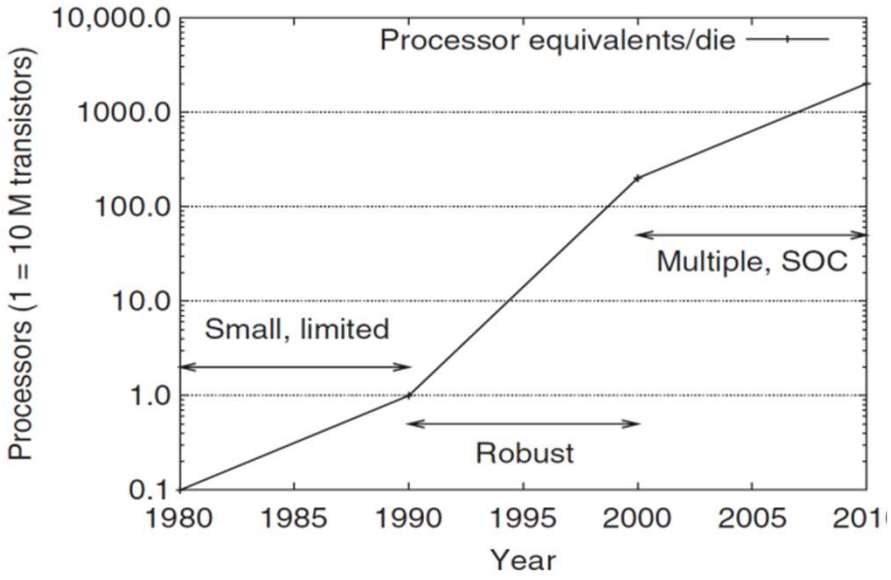
1. **Additional Cache:** Here we add cache storage and, as large caches have slower access times, a second - level cache.

2. A More Advanced Processor: We implement a superscalar or a VLIW processor that executes more than one instruction each cycle. Additionally, we can speed up the execution units that affects the critical path delay, especially the floating - point execution times.

3. Multiple Processors: Now we implement multiple (superscalar) processors and their associated multilevel caches.

System Architecture and Complexity

- Instead of the 100,000 transistor processors, our advanced processor has millions of transistors.
- The way to manage this complexity is to reuse designs.
- So, reusing several simpler processor designs implemented on a die is preferable to a new, more advanced, single processor.
- For this to work, we need a good interconnection mechanism to access the various processors and memory.



Complexity of design

System Architecture and Complexity

So, when an application is well specified, the system-on-a-chip (SOC) approach includes

1. Multiple (usually) heterogeneous processors, each specialized for specific parts of the application;

2. The main memory with (often) ROM for partial program storage;

3. A relatively simple, small (single - level) cache structure or buffering schemes associated with each processor; and

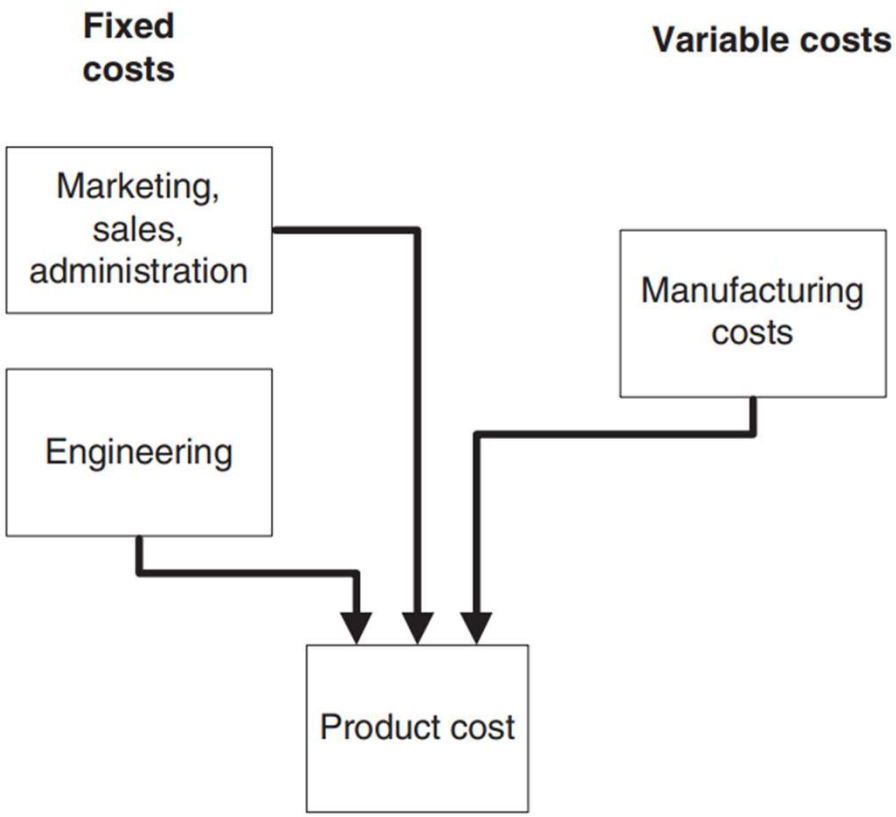
4. A bus or switching mechanism for communications.

Product Economics and Implications For SoC

Factors Affecting Product Costs

The basic cost and profitability of a product depend on many factors:

- its technical appeal,
- its cost
- the market size, and
- the effect the product has on future products.
- The issue of cost goes well beyond the product's manufacturing cost.



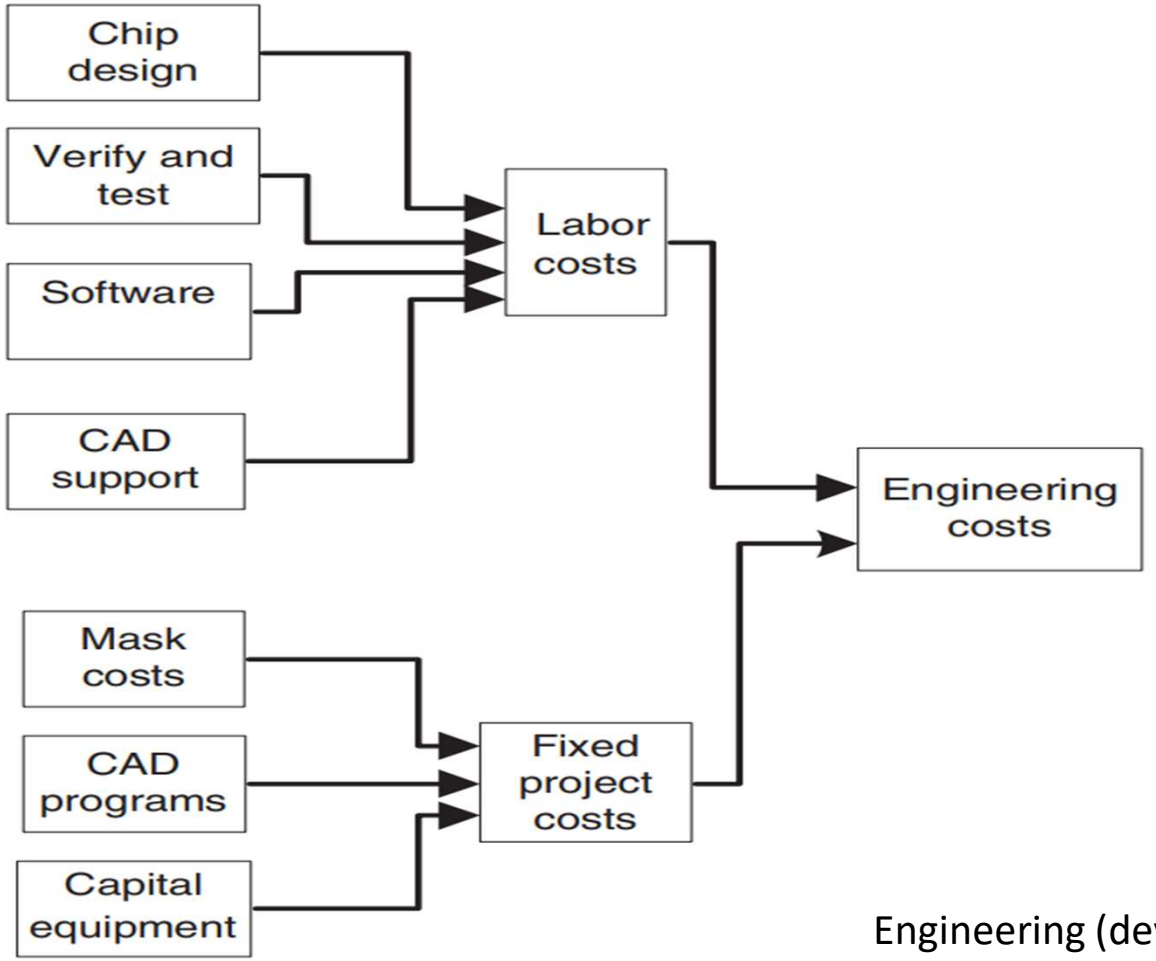
Project cost components.

Non-engineering fixed costs

Non-engineering fixed costs include

- manufacturing start - up costs,
- inventory costs,
- initial marketing and sales costs,
- administrative overhead marketing costs include obvious items such as market research,
- strategic market planning,
- pricing studies,
- competitive analysis,
- sales planning and advertising costs

Factors Affecting Engineering Costs



Engineering (development) costs

Chip Design Basics

- The trade - off between cost and performance is fundamental to any system design.
- With the advances in lithography, the transistors are getting smaller. The minimum width of the transistor gates is defined by the process technology.
- Table 2.1 refers to process technology generations in terms of nanometers;
- older generations are referred to in terms of microns (µ m).

Technology Roadmap Projections

TABLE 2.1 Technology Roadmap Projections

Year	2010	2013	2016
Technology generation (nm)	45	32	22
Wafer size, diameter (cm)	30	45	45
Defect density (per cm^2)	0.14	0.14	0.14
μP die size (cm^2)	1.9	2.6	2.6
Chip frequency (GHz)	5.9	7.3	9.2
Million transistors per square centimeter	1203	3403	6806
Max power (W) high performance	146	149	130

Design Trade - Offs

With increases in chip frequency and especially in transistor density, the designer must be able to find the best set of trade - offs in an environment of rapidly changing technology.

- In making basic design trade - offs, we have five different considerations.

Design Trade - Offs

- The first **is time** , which includes partitioning instructions into events or cycles, basic pipelining mechanisms used in speeding up the instruction execution, and cycle time as a parameter for optimizing program execution.

- Second, we discuss **area .** The cost or area occupied by a particular feature is another important aspect of the architectural trade - off.

- Third, **power consumption** affects both performance and implementation.
-
- Fourth, **reliability** comes into play to cope with deep submicron effects.

- Fifth, **configurability** provides an additional opportunity for designers to trade off recurring and nonrecurring design costs.

FIVE BIG ISSUES IN SYSTEM - ON - CHIP (SOC) DESIGN

Four of the issues are obvious.

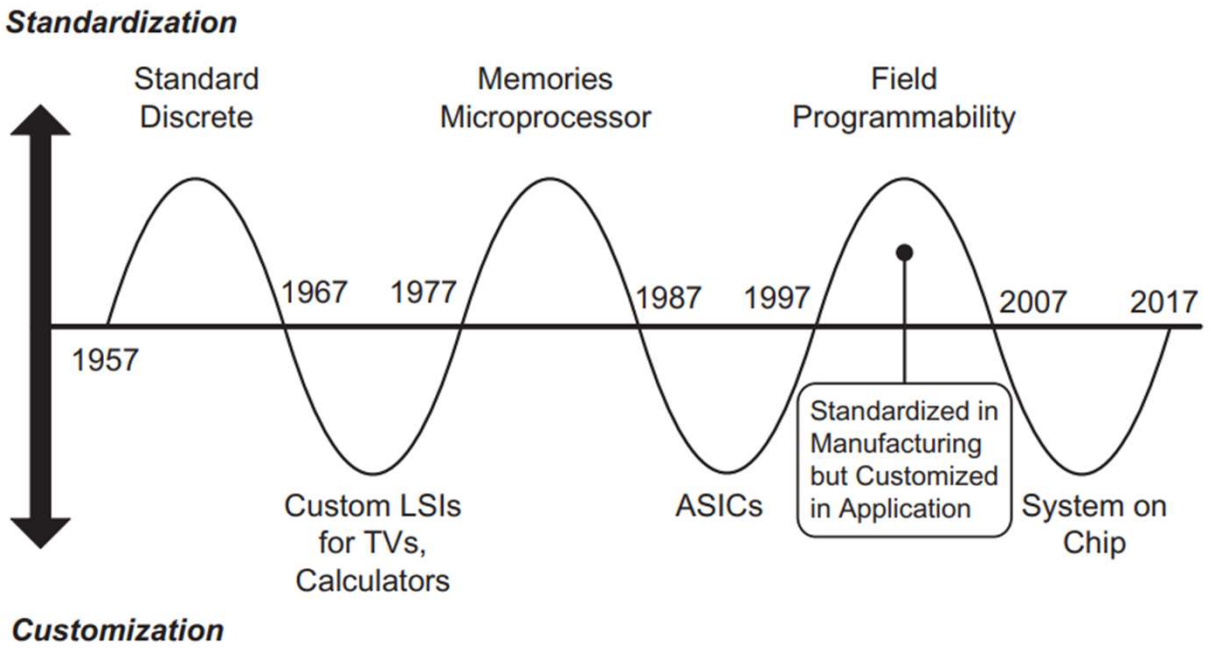
- **Die area** (manufacturing cost, The smaller each chip is, the more chips can be produced per wafer. This increases wafer utilization, reducing the cost per chip.
- **Performance** (heavily influenced by cycle time) are important basic SOC design considerations.
- **Power consumption** has also come to the fore as a design limitation.
- As technology shrinks feature sizes, **reliability** will dominate as a design consideration.
-

FIVE BIG ISSUES IN SYSTEM - ON - CHIP (SOC) DESIGN

- **Configurability,** is less obvious as an immediate design consideration.
- **Configurability** in chip design refers to the ability of a chip to be programmed, customized, or adapted after manufacturing to perform specific functions or tasks.
- Configurability enables programmability in the field and can be seen to provide features that are " standardized in manufacturing while customized in application. "

Makimoto's wave

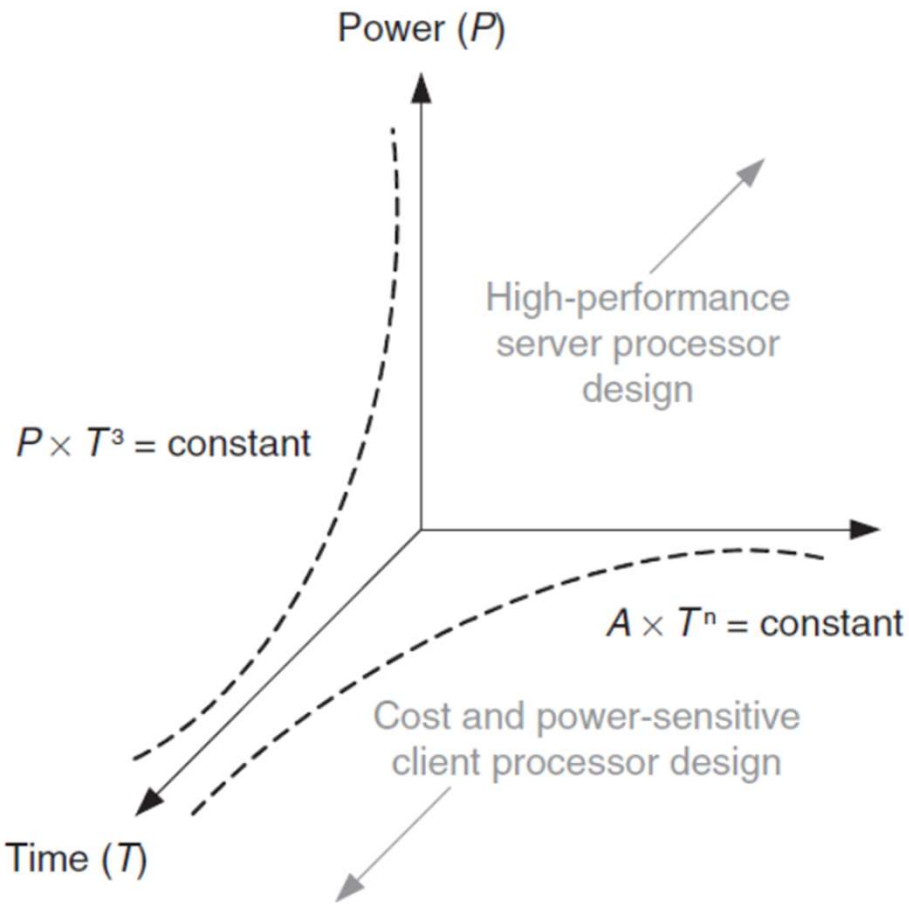
- The cyclical nature of the integrated circuit industry between standardization and customization has been observed by Makimoto and is known as Makimoto's wave.



Processor design trade - offs.

- In terms of complexity, various trade - offs are possible. For instance, area can be traded off for performance.
- Very large scale integration (VLSI) complexity theorists have shown that an $A \times T^n$ bound exists for processor designs, where n usually falls between 1 and 2.
- It is also possible to trade off time T for power with a $P \times T^3$ bound.
- Embedded and high - end processors operate in different design regions of this three - dimensional space.
- The **power and area axes** are typically optimized for embedded processors, whereas the **time axis** is typically for high - end processors.

Processor design trade - offs.



- Increase time **to complete a task**, decrease power.
- **Decrease area, decrease power consumption.**

- Decrease SoC area, possible increase the time.

Traditional AT (Area-Time) trade-offs

- For silicon technology this type of “textbook” design has been nicely formalized by theorists as “AT” bounds.

- $A\,T^n$ = constant (n is typically between 1 and 2)

- Simply stated, if a design has more area, A, available it should be able to perform a given computation is less time, T.

- doubling processing speed may increase die size by 2 to 4 times.

T P (Time-Power) trade-offs

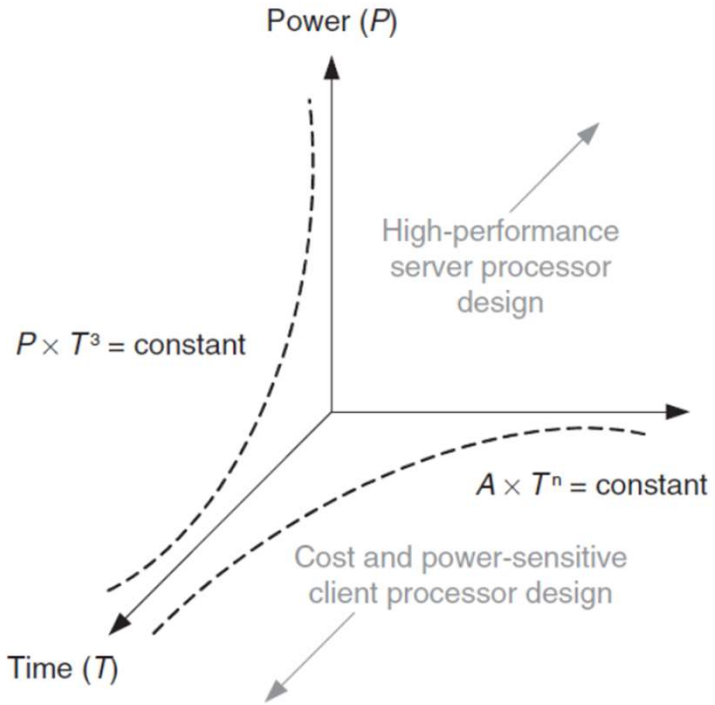
$T^3 P = \text{constant}$

- As power becomes increasingly important the cubic tradeoff between time and power ($T^3 P=k$) forces designers to use relatively cheap area to increase performance rather than expensive power required by higher clock rates.

- As designers focus attention on power the question of optimal power oriented architectures is apparent

A T P design optimization

Putting these bounds together defines a surface of optimal A T P design possibilities, as shown in figure 1.



- Increase time **to complete a task**, decrease power.
- **Decrease area, decrease power consumption.**
- Decrease SoC area, possible increase the time.

Requirements and Specifications

- The **five basic SOC trade - offs** provide a framework for analyzing **SOC requirements** so that these can be translated into specifications.
- **Cost requirements** coupled with market size can be translated into die cost and process technology.
- Requirements for **wearable and weight** put limit bounds on power or energy consumption.
- Limitations on **clock frequency,** can affect heat dissipation.
- Any one of the trade - off criteria for a particular design, have the highest priority.

Requirements and Specifications (examples)

- **High - performance** systems will **optimize time** at the **expense of cost and power.**
- **Low - cost systems** will **optimize die cost, reconfigurability, and design reuse.**
- **Wearable systems** stress **low power** (since, the power supply determines the system weight). e.g. cell phones.
- **Embedded systems** in planes and other safety - critical applications would **stress reliability**, with **performance and design lifetime** being important secondary considerations.
- **Gaming systems** would **stress cost** (specially production cost, secondarily, performance)

Cycle Time

- The **time** receives **considerable attention** from processor designers.

- It is the basic measure of performance;

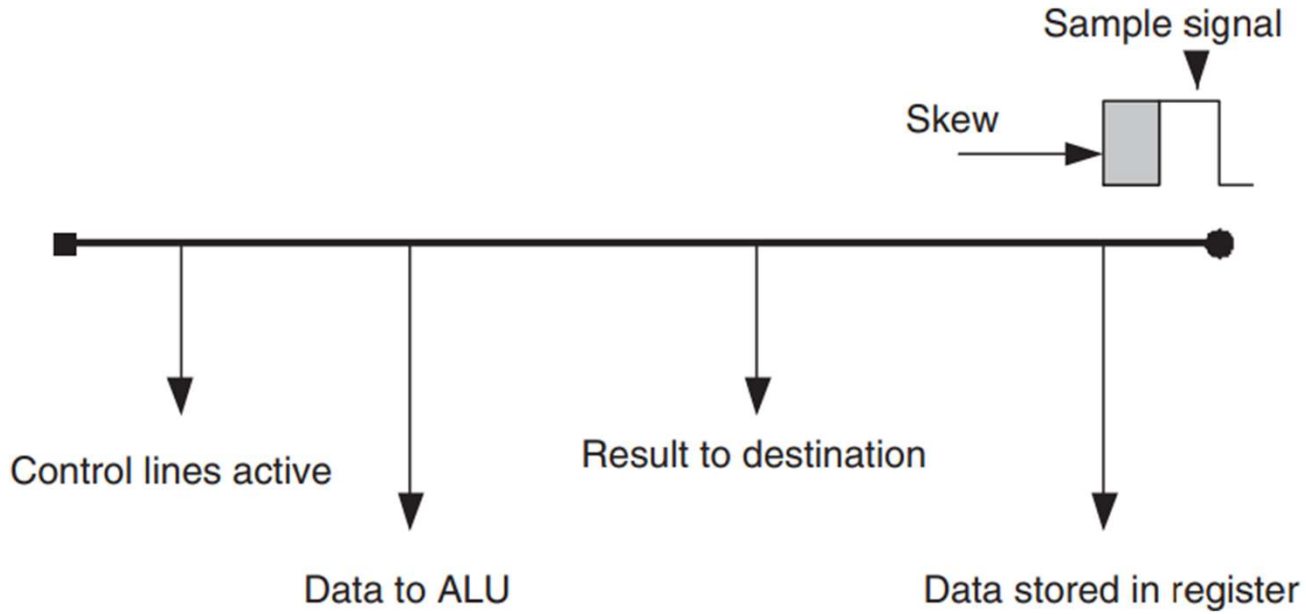
  however, breaking actions into cycles and reducing both cycle count and cycle times are important but not preferable.

- The way in which **actions are partitioned into cycles is important**.

- **A common problem is having unanticipated "extra" cycle**s required by a basic action such as a cache miss.

Defining a Cycle

- A **cycle** (of the clock) is the **basic time unit** for **processing information**.
- In a **synchronous systems**, the **clock rate is and the cycle** time is determined by finding the maximum time to a fixed value accomplish a frequent operation in the machine, such as an add or register data transfer.
- **Cycle time** must be **sufficient for data** to be stored into a specified destination register.

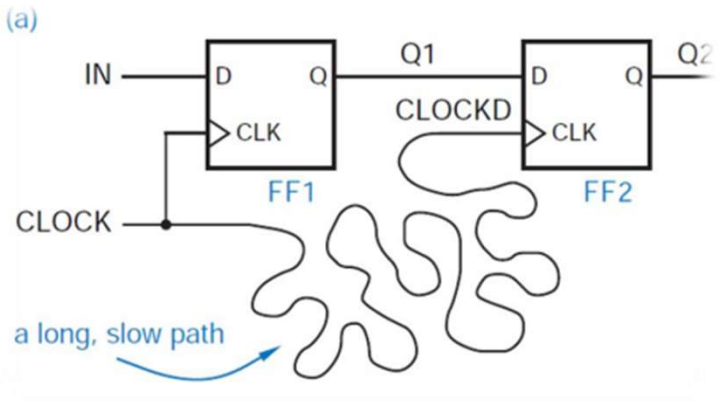


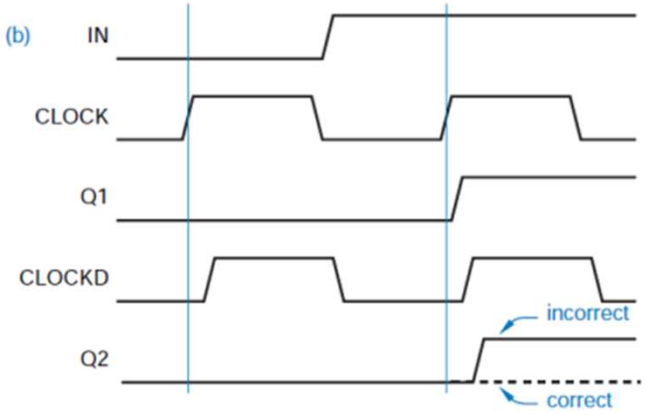
Cycle Time

- A **cycle begins** when **the instruction decoder specifies the values** for **the registers** in the system.
- These **control values** connect the **output of a specified register** to **another register** or an **adder** or similar object.
- This allows **data from source registers** to propagate through designated combinatorial logic into **the destination register.**
- Finally, **after a suitable setup time**, all registers are sampled by an edge or pulse produced by the clocking system.

Cycle Time

- In a **synchronous system:**
- The **cycle time is determined by the sum of the worst - case time** for each step or action within the cycle.
- However, the **clock itself may not arrive at the anticipated time** (due to propagation or loading effects).
- We call the maximum deviation from the expected time of clock arrival the (uncontrolled) **clock skew.**





Cycle Time

- In an **asynchronous system:**
- The **cycle time** is simply **determined by the completion of an event** or **operation.**
- A **completion signal is generated**, which then allows the next operation to begin.
- **Asynchronous design** is generally **not used within pipelined processors** because of the pipeline timing constraints.

Optimum Pipeline

- At one time, the **concept of pipelining in a processor** was treated as an **advanced processor design technique**.

- From several decades, **pipelining has been an integral part** of any processor or controller design.

- The trade - off between **cycle time and number of pipeline stages** is treated in the section on **optimum pipeline.**

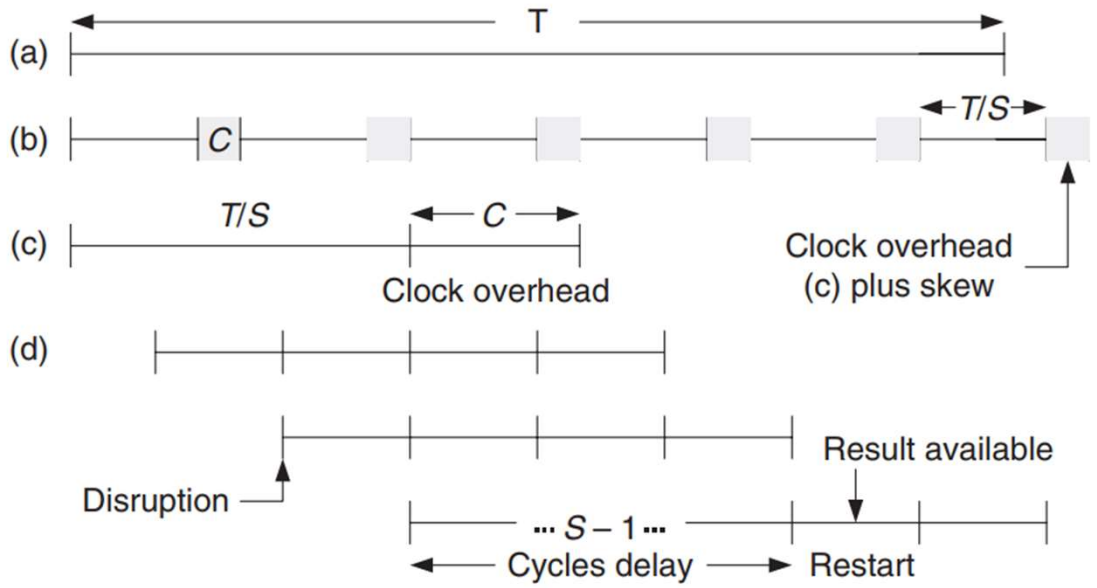
Optimum Pipeline

- A **basic optimization** for the **pipeline processor** designer is the **partitioning of the pipeline into concurrently operating segments**.

- A **large number of segments** allow a maximum speedup.
- However, each new segment carries **clocking overhead** with it, which can adversely affect performance.

- If we **ignore the problem of fitting actions into an integer number of cycles**, we can derive an **optimal cycle time, Δt,** and hence the level of **segmentation for a simple pipelined processor.**

Optimum Pipeline



(a) Unclocked instruction execution time, T .
(b) T is partitioned into S segments. Each segment requires C clocking overhead.
(c) Clocking overhead and its effect on cycle time, T / S .
(d) Effect of a pipeline disruption (or a stall in the pipeline)

Optimum Pipeline

- Total time required to execute an instruction without pipeline segments is T nanoseconds.
- Here, we need to find the optimum number of segments S to allow clocking and pipelining.
- The ideal delay through a segment is T_{seg}. **T_{seg} = T/S** = Partitioning overhead is associated with each segment.
- This clock **overhead time** C (nS), includes clock skew, setup & hold times of register.
- Now, the actual cycle time (Figure c) of the pipelined processor is the ideal cycle time T / S + overhead

$$\Delta t = \frac{T}{S} + C$$

Optimum Pipeline

- In Ideal pipelined processor, there will not be any delays, but certain delays can occur due to unexpected branches.
- Suppose, such delays (interruptions) occur with frequency b and have the effect of invalidating the (S − 1) instructions prepared to enter, or already in the pipeline (figure d) representing a " **worst - case** " disruption, d
- Considering pipeline interruption, the performance of the processor is:

$$\text{Performance} = \frac{1}{1+(S-1)b} \text{ instructions per cycle}$$

Optimum Pipeline

- The throughput (*G*) *can be calculated as*

$$G = \frac{\text{performance}}{\Delta t} \text{instructions/ns}$$

$$= \left(\frac{1}{1+(S-1)b} \right) \times \left(\frac{1}{(T/S)+C} \right)$$

If we find the *S for which*

$$\frac{dG}{dS} = 0$$

we can find S_{opt}, *the optimum number of pipeline segments*

$$S_{\text{opt}} = \sqrt{\frac{(1-b)T}{bC}}$$

Optimum Pipeline

- *The total instruction execution latency (T_{instr}) is*

$$T_{\text{instr}} = T + S \times (\text{clocking overhead})$$
$$= T + SC \quad \text{or } S(T_{\text{seg}} + \text{C})$$
$$= S\Delta t.$$

We can compute the throughput performance *G in mips*.

Suppose *T = 12.0 ns and b = 0.2, C = 0.5 ns.*
Then, S_{opt} *= 10 stages.*

Determining S_{opt} *can serve as:*
A design starting point or
As an important check on an optimized design.

Die Area and Cost

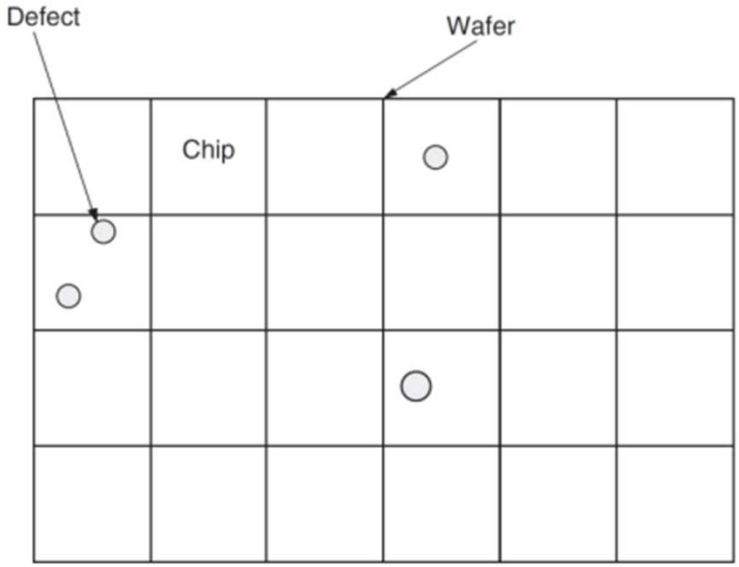
- Cycle time, machine organization, and memory configuration determine machine performance.

- Determining performance is relatively straightforward when compared to the determination of overall cost.

- A good design achieves an optimum **cost – performance trade - off** at a particular target performance. This determines the quality of a processor design

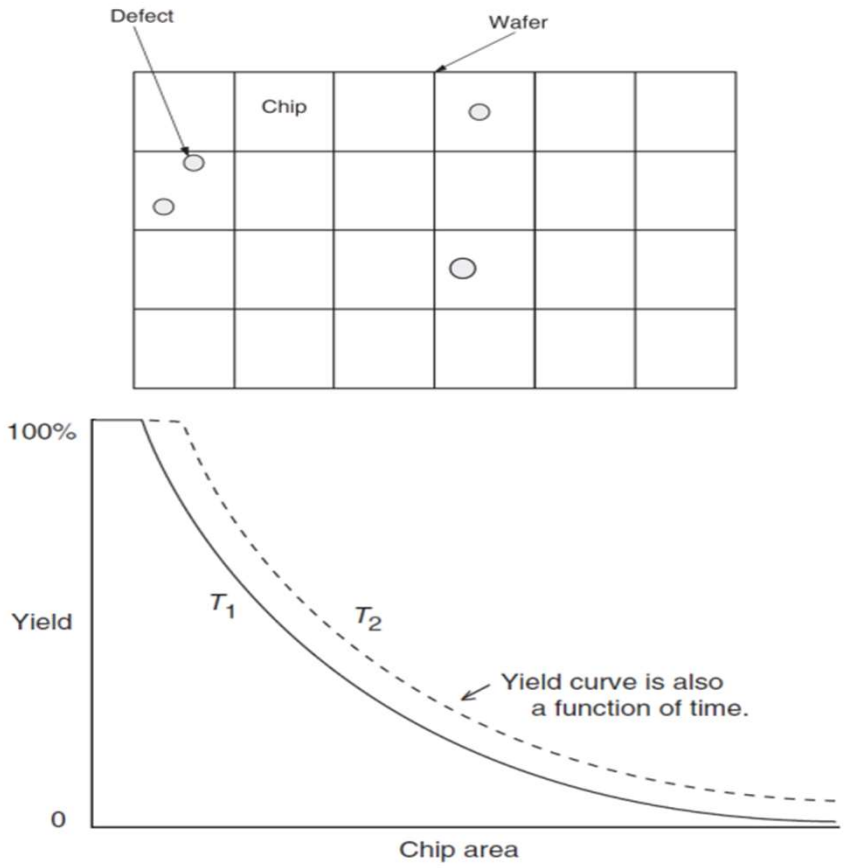
Processor Area

- SOCs usually have die sizes of about 10 – 15 mm.
- This die is produced in bulk from a larger wafer, 30 cm in diameter.
- Unfortunately, neither the silicon wafers nor processing technologies are perfect.
- **Defects randomly** occur over the wafer surface.



Processor Area

- Larger chips could readily accommodate any function that the designer might wish to include
- Large chip areas require an absence of defects over that area.
- If chips are too large for a particular processing technology, there will be little or no yield (good chips produced in a manufacturing process).
- Figure illustrates yield versus chip area.



Processor Area

$$\text{Die yield} = \text{Wafer yield} \times \left(1 + \frac{\text{Defects per unit area} \times \text{Die area}}{\alpha}\right)^{-\alpha}$$

- **Example:**

  Find the die yield for dies that are 1.5 cm on a side and 1.0 cm on a side, assuming a defect density of 0.4 per cm^2 and α is 4.

- ***Answer:***

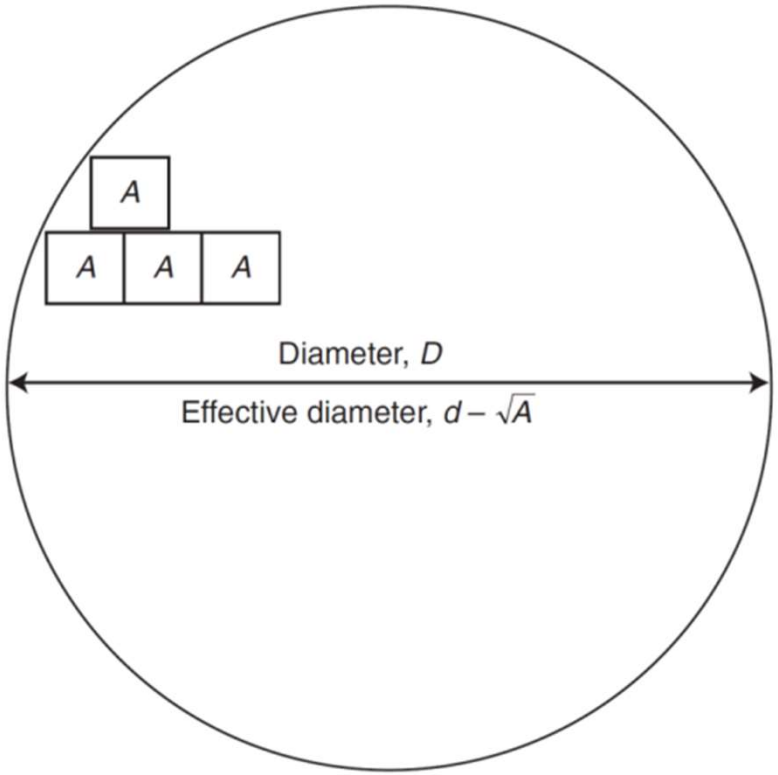
  The total die areas are 2.25 cm^2 and 1.00 cm^2 . For the larger die, the yield is

$$\text{Die yield} = \left(1 + \frac{0.4 \times 2.25}{4.0}\right)^{-4} = 0.44$$

$$\text{For the smaller die, it is Die yield} = \left(1 + \frac{0.4 \times 1.00}{4.0}\right)^{-4} = 0.68$$

That is, less than half of all the large die are good but more than two-thirds of the small die are good.

Processor Area



Number of die (of area *A*) *on a wafer of diameter d .*

Number of Dies per Wafer

- To calculate the number of dies per wafer for a given wafer diameter and die size, you can use the following formula:

$$N = \frac{\pi \times (d/2)^2}{A} - \frac{\pi \times d}{\sqrt{2A}}$$

where:

N is the number of dies per wafer,

d is the diameter of the wafer,

A is the area of a single die.

Most of the rest are removed at the upper-left etc. "corners" of the circle where the curve is at about a 45° angle to the axes. In those regions, the circumference through the die is roughly equal to the diagonal of the die. If the die's side is x then the diagonal is $x\sqrt{2}$ and the area is $S = x^2$, so we can get the diagonal from the area S by $\text{diagonal} = \sqrt{2S}$. So the number of dies removed in those regions is about one die for each $\sqrt{2S}$ length of the circle's circumference. If you look at the diagram you will see that the number of dies removed is actually higher than that, since the dies "fill in the corners" of each other.

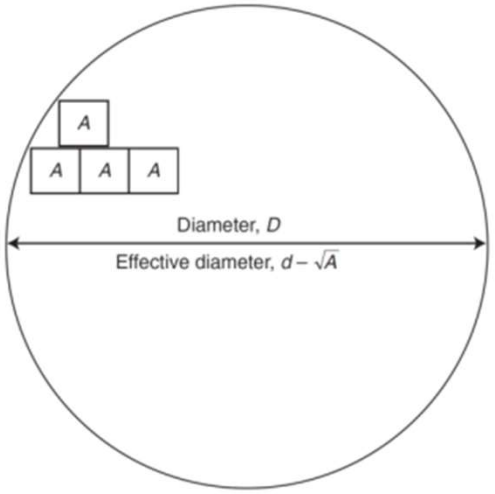
Number of Dies per Wafer

- The first term $\frac{\pi \times (d/2)^2}{A}$ gives the total number of dies that would fit into the wafer if the wafer were fully usable.
- The second term $\frac{\pi \times d}{\sqrt{2A}}$ approximates the number of dies lost due to the circular shape of the wafer (dies near the edge may not fully fit).

Number of Dies per Wafer

- Suppose a die with square aspect ratio has **area *A***. *About **N** of these dice* can be realized in a wafer of diameter ***d***:

$$N \approx \frac{\pi}{4A}\left(d - \sqrt{A}\right)^2$$



- Now suppose there are N_G *good chips and* N_D *point defects on the wafer.*

- *Even if* $N_D > N$ *, we can expect several good chips since the defects are randomly* distributed and several defects would cluster on defective chips, sparing a few goodones.

Yield Calculation

- Suppose we add a random defect to a wafer; ***(N_G / N)*** *is the* ***probability that the defect destruct a good die.***
- *If* the defect hits an bad die, it would cause no change to the number of good die.
- In other words, the change in the number of good die (***N_G***), *with* respect to the change in the number of defects (***N_D***), *is*

$$\frac{dN_G}{dN_D} = -\frac{N_G}{N}$$

$$\frac{1}{N_G} dN_G = -\frac{1}{N} dN_D$$

On Integrating and solving

$$\ln N_G = -\frac{N_D}{N} + C$$

Yield Calculation

- To evaluate C, note that when $N_G = N$, *then* $N_D = 0$*; so,* ***C must be ln (N)***.
- Then the yield is

$$\text{Yield} = \frac{N_G}{N} = e^{-N_D/N}$$

This describes a Poisson distribution of defects. If ρ_D ***is the defect density per unit area****, then*

$$N_D = \rho_D \times (\text{wafer area})$$

For large wafers ***d >> A****, the diameter of the wafer is significantly larger than* the die side and

$$\left(d - \sqrt{A}\right)^2 \approx d^2$$

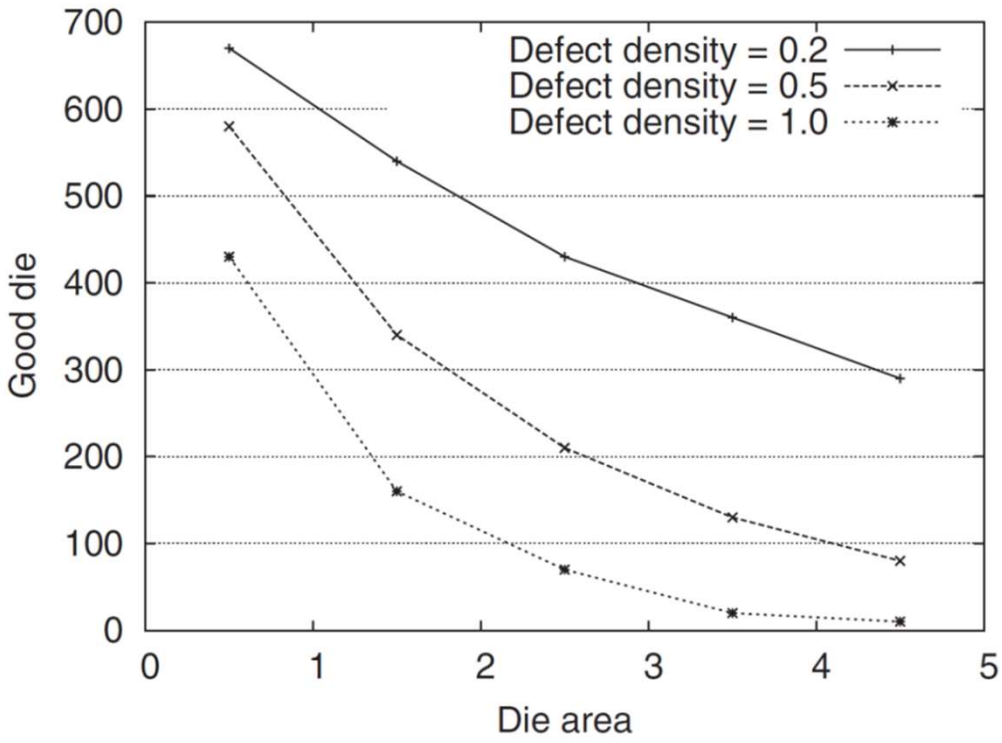
and $$\frac{N_D}{N} = \rho_D A,$$

so that $$\text{Yield} = e^{-\rho_D A}$$

Yield vs die area

$$\text{Yield} = e^{-\rho_D A}$$

- **Figure shows** the projected number of **good die as a function of die area** for several defect densities.
- Modern fab facility would have ρ_D **between 0.15 – 0.5.**
- Doubling the die area has a significant effect on yield.



Ideal and Practical Scaling

- As **feature sizes shrink** and **transistors gets smaller**, the **transistor density will improve.**

- Similarly, **transistor delay** (or gate delay) should **decrease linearly with feature size.**

- **Practical scaling is different** as wire delay, and **wire density does not scale at the same rate as transistors scale.**

- **Wire delay remains almost constant** as feature sizes shrink.

Feature Size

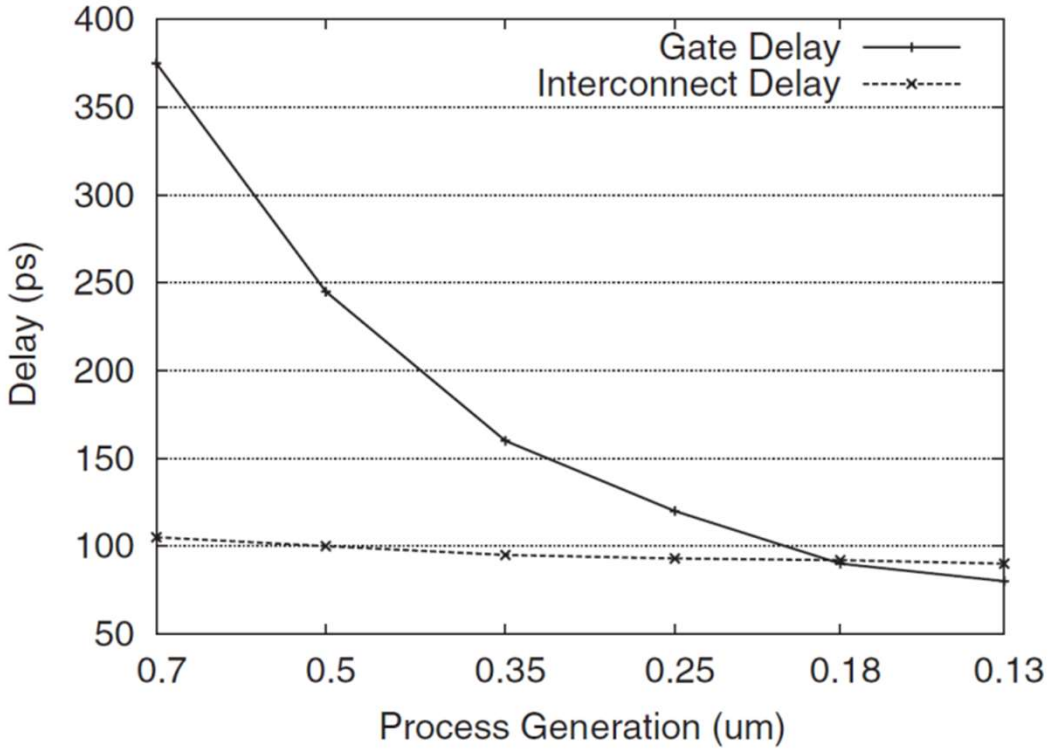
- In semiconductor manufacturing, the feature size (also called process node or technology node) refers to the smallest dimension of a transistor or other component that can be reliably manufactured on a silicon wafer.

- It is usually measured in nanometers (nm).

- Smaller feature sizes allow more transistors to be packed into the same area, which can increase the computational power and efficiency of a chip.

Ideal and Practical Scaling

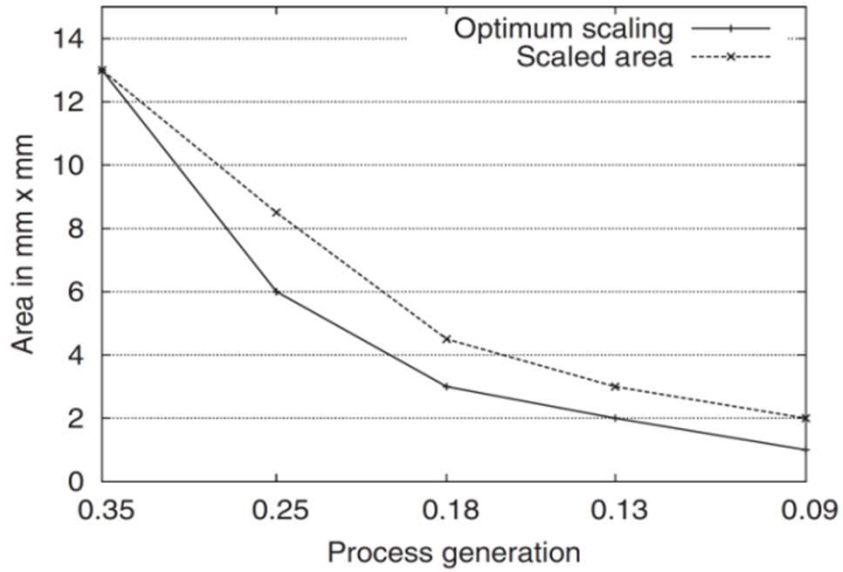
The dominance of wire delay over gate delay.



- Figure illustrates the increasing dominance of wire delay over gate delay.

Ideal and Practical Scaling

- Scaling factor of 1.5 is commonly considered more accurate.
- Major technology changes can affect scaling in a discontinuous manner.
- The simple scaling of a design might only scale as 1.5, but a new implementation taking advantage of all technology features could scale at 2.



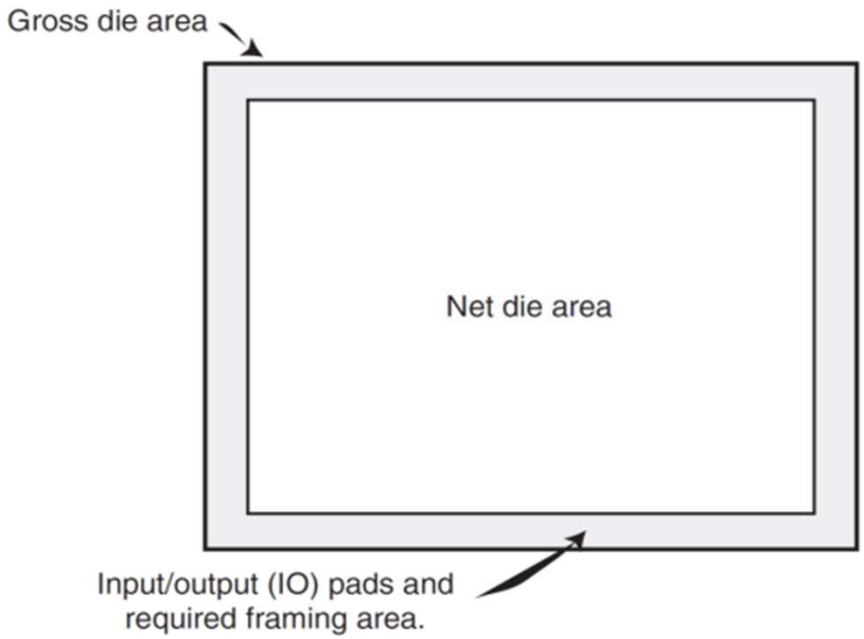
Baseline SOC Area Model:

The **key factor to design efficient system** is chip floor planning.

- Each **functional area** of the processor **must be allocated sufficient space** for its implementation.
- **Functional units that frequently communicate** must be **placed close together.**
- Sufficient room must be allocated for connection paths.
- **Baseline system** can be used to illustrate possible trade - offs in optimizing **the chip floorplan.**
- This model is based **upon empirical observations made of existing chips** and **design experience**

Baseline SOC Area Model

- ***Starting Point:*** *The design process begins with an understanding of the* parameters of the semiconductor process.
- Suppose we expect to be able to use a manufacturing process that has a **defect density of 0.2,** defect per square centimeter; for economic reasons, we target an **initial yield of about 95%:**



$$Y = e^{-\rho_D A}$$

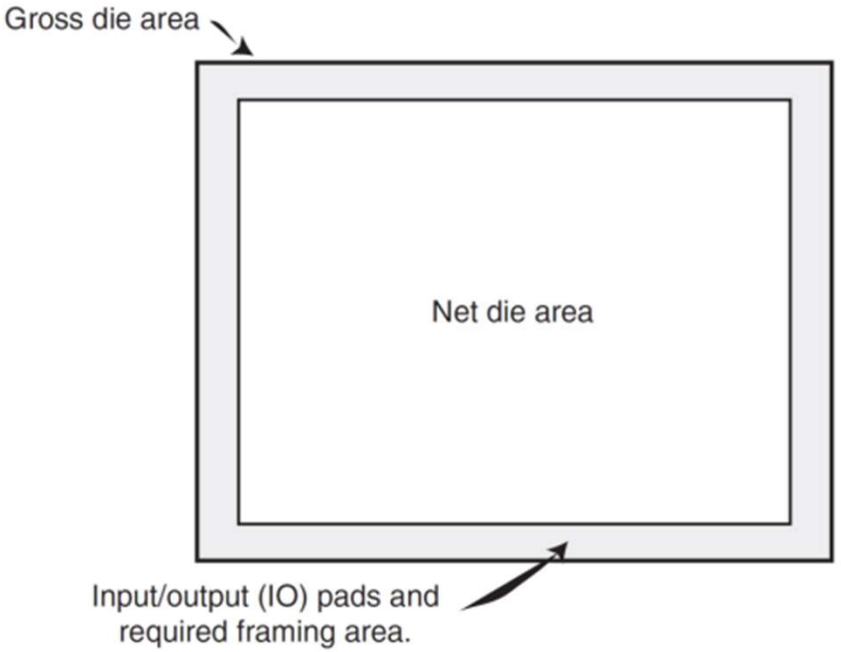
where ρ_D = *0.2 defect per square centimeter, Y = 0.95. Then*

$A = 25\ \text{mm}^2$ approximately 0.25 cm^2

Baseline SOC Area Model

- So the chip area available to us is 25 mm^2 .
- This is the total die area of the chip,
- but such things as **pads for the wire bonds** that connect the chip to the external world, **drivers** for these connections, and **power supply lines** all act to **decrease the amount of chip area** available to the designer.

- Suppose we allow **12% of the chip area to accommodate these functions** (usually around the periphery of the chip), then the **net area will be 22 mm^2**



Baseline SOC Area Model

- ***Feature Size:*** *The* ***smaller the feature size****, the* ***more logic that can be accommodated*** within a fixed area.
- At feature size, ***f = 65 nm,*** *we have about* ***5200 A*** *or area units* in **22 mm^2**

- ***The Architecture:*** *Each system has different* objectives.
- For example, assume that we need the following:
  - A **small 32 - bit core processor** with an 8 *KB I - cache and a 16 KB D - cache;*
  - Two **32 - bit vector processors**
  - Memory; **an 8 *KB I - cache and a 16 KB D - cache*** *for scalar data;*
  - A **bus control unit**;
  - Directly addressed **application memory of 128 *KB*** *; and*
  - A **shared L2 cache**.

Baseline SOC Area Model

- ***An Area Model:*** *The following is a breakdown of the area required for various* units used in the system.

Unit	Area (A)
Core processor (32^b)	100
Core cache ($24\,KB$)	96
Vector processor #1	200
Vector registers and cache #1	256 + 96
Vector processor #2	200
Vector registers and cache #2	352
Bus and bus control (50%)	See below 650
Application memory (128 KB)	512
Subtotal	2462

- ***Latches, Buses, and Interunit Control:*** *For* ***each of the functional units,*** *there* is a **certain amount of overhead** to accommodate nonspecific storage **(latches),** interunit communications **(buses)**, and interunit **control.**
- This is **allocated as 10% overhead for latches** and **40% overhead for buses, routing, clocking, and overall control.**

Baseline SOC Area Model

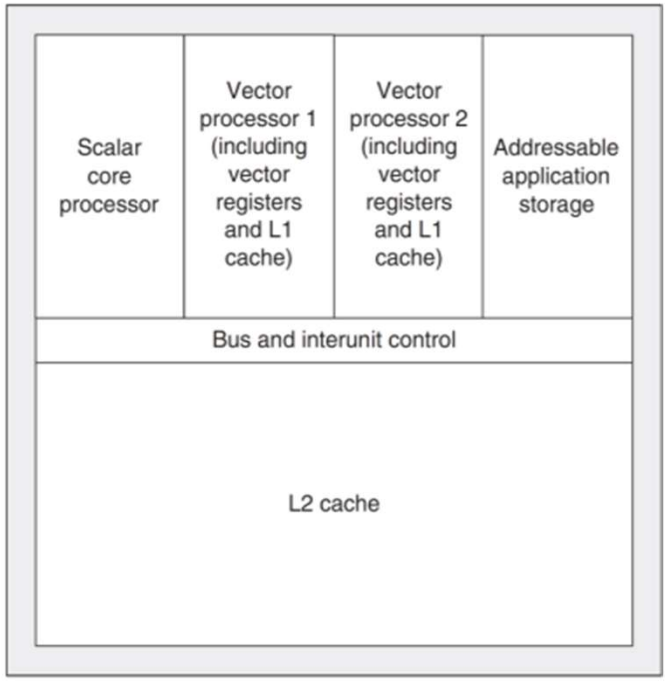
- ***Total System Area:*** *The designated* ***processor elements and storage occupy 2462 A*** *. This leaves a net of 5200 − 2462 =* ***2738 A available for cache.***

- ***Cache Area:*** *The net area available for* ***cache is 2738 A .***
- *However, bits and* pieces that may be unoccupied on the chip are not always useful to the cache designer.
- These pieces must be collected into a reasonably compact area that accommodates efficient cache designs.

Baseline SOC Area Model

- An example baseline floor plan is shown in figure.
- A summary of area design rules follow:

1. **Compute the target chip size** from the target yield and defect density.
2. **Compute the die cost** and determine whether it is satisfactory.
3. **Compute the net available area.** Allow 10 – 20% for pins, guard ring, power supplies, and so on.
4. **Determine the rbe** (register bit equivalent) **size** from the minimum feature size.
5. **Allocate the area** based on a trial system architecture until the basic system size is determined.
6. **Subtract** the basic system **size (5)** from the net available **area (3).** This is the die area available for **cache and storage**.



Register Bit Equivalent (RBE):

- This term is used to describe a unit of measurement for chip area that relates to the size of a single register bit.
- A register is a small, fast storage location in a processor used to hold data temporarily during execution.
- The RBE size quantifies how much physical space on the chip one bit of register storage occupies.

Power

- Growing demands for **wireless and portable electronic appliances** have focused much **attention on power consumption**.

- The **SIA road map** points to increasingly **higher power for microprocessor chips** because of their **higher operating frequency**, **higher overall capacitance**, and **larger size.**

- **Power scales** indirectly **with feature size** (45 nm, 32nm 22 nm etc)**.**

Some Power Operating Environments

Type	Power/Die	Source and Environment
Cooled high power	70.0 W	Plug-in, chilled
High power	10.0–50.0 W	Plug-in, fan
Low power	0.1–2.0 W	Rechargeable battery
Very low power	1.0–100.0 mW	AA batteries
Extremely low power	1.0–100.0 μW	Button battery

Battery Capacity and Duty Cycle

Type	Energy Capacity (mAh)	Duty Cycle/Lifetime	At Power
Rechargeable	10,000	50 h (10–20% duty)	400 mW–4 W
2 × AA	4000	0.5 year (10–20% duty)	1–10 mW
Button	40	5 years (always on)	1 μW

SOC Power

- At the device level, **total power dissipation (P_{total})** *has* ***two major sources:***
  - **dynamic** or **switching power** and
  - **static power** caused by leakage current:

$$P_{\text{total}} = \frac{CV^2\text{freq}}{2} + I_{\text{leakage}}V$$

Where ***C*** *is the* ***device capacitance****;*
V *is the* ***supply voltage****;*
freq *is the* ***device*** **switching frequency**; and
$I_{leakage}$ *is the* ***leakage current.***

Gate delays are roughly **proportional to** $CV / (V - V_{th})^2$, where V_{th} *is the threshold voltage of the transistors.*

SoC Power

- As **feature sizes decrease**, so **do device sizes**.
- **Smaller device sizes** result in **reduced capacitance.**
- **Decreasing the capacitance decreases** both the **dynamic power** consumption and the **gate delays.**
- As device sizes decreases, the electric field applied to them becomes destructively large in quantity.
- To **increase the device reliability,** we need to **reduce the supply voltage *V*.**

Power

- ***Reducing V** effectively **reduces** **the dynamic power consumption** but results in an **increase in the gate delays.***

- We can avoid this loss by reducing V_{th}.

- ***Reducing V_{th}** **increases** the **leakage current** and hence, **static power consumption also increases**.*

- This has an important effect on design and production; there are two device designs that must be accommodated in production:
  1. The high - speed device with low V_{th} *and high static power; and*
  2. The slower device maintaining V_{th} and low static power with *increase of circuit density* .

  In either case, we can reduce switching loss by lowering the supply voltage, V

SoC Power

Chen et al. showed that the drain current is proportional to

$$I = (V - V_{th})^{1.25},$$

- signal transition time and frequency scale with the charging current.
- So, the maximum operating frequency is also proportional to (V – Vth) $^{1.25}$ / V .
- For values of V and V_{th} of interest, this means that frequency scales with the supply voltage, V.

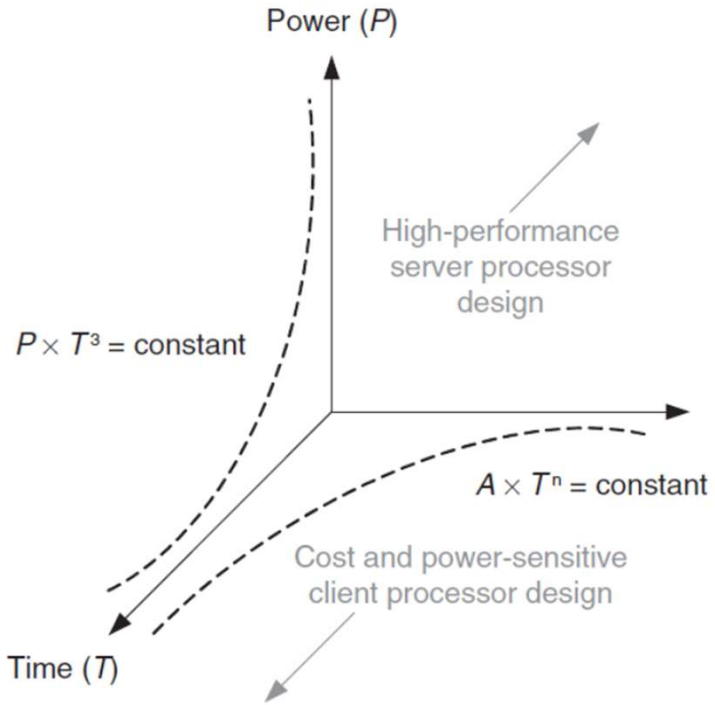
SoC Power

- Assume V_{th} is 0.6 V; suppose we reduce the supply voltage by one - half, say, from 3.0 to 1.5 V, the operating frequency is also reduced by about one - half.
- So, reducing the supply voltage by half also reduces the operating frequency by half.
- Now by the power equation (since the voltage and frequency were halved), the total power consumption is one - eighth of the original.
- Thus, if we take an existing design optimized for frequency and modify that design to operate at a lower voltage, the frequency is reduced by approximately the cube root of the original (dynamic) power:

$$\frac{\text{freq}_1}{\text{freq}_2} = \sqrt[3]{\frac{P_2}{P_1}}.$$

A T P design optimization

Putting these bounds together defines a surface of optimal A T P design possibilities, as shown in figure 1.



- Increase time **to complete a task**, decrease power.
- **Decrease area, decrease power consumption.**
- Decrease SoC area, possible increase the time.

Area – Time – Power trade - offs in processor design

Processor design trade - offs are quite different for our two general classes of processors:

1. **Workstation Processor:** These designs are oriented to high clock frequency and AC power sources (excluding laptops). Since they are not area limited as the cache occupies most die area, the designs are highly elaborated (superscalar with multithreading).

2. **Embedded Processor Used in SOC** : Processors here are generally simpler in control structure but may be quite elaborate in execution facilities (e.g., digital signal processor [DSP]). Area is a factor as is design time and power.

A Typical Processor Die Compared with a Typical SOC Die

	Processor on a Chip	SOC
Area used by storage	80% cache	50% ROM/RAM
Clock frequency	3.5 GHz	0.5 GHz
Power	≥50 W	≤10 W
Memory	≥1-GB DRAM	Mostly on-die

Reliability

- The important design dimension is reliability, (dependability or fault tolerance).
- Reliability is related to
  - -die area,
  - -clock frequency,
  - -power.
- Die area increases the amount of circuitry and the probability of a fault, but it also allows the use of error correction and detection techniques.
- Higher clock frequencies increase electrical noise and noise sensitivity.

Reliability

- Faults, if detected, can be masked by
  - error - correcting codes (ECCs),
  - instruction retry, or
  - functional reconfiguration.

- *Some definitions:*

1. A ***failure*** *is a deviation from a design specification.*
2. An ***error*** *is a failure that results in an incorrect signal value*
3. A ***fault*** *is an error that manifests itself as an incorrect logical result.*
4. A ***physical fault*** *is a failure caused by the environment, such as aging,* radiation, temperature, or temperature cycling. The probability of physical faults increases with time.
5. A ***design fault*** *is a failure caused by a design implementation that is* inconsistent with the design specification.

Reliability

- **Dealing with Manufacturing Faults:**

- The traditional way of dealing with manufacturing faults is through testing.
- As transistor density increases, the problem of testing increases even faster.
- The testable combinations increase exponentially with transistor count.

Reliability

Dealing with Manufacturing Faults:

- A technique to give testing access to interior (not accessible from the instruction set) storage cells is called scan .
- A scan chain in its simplest form consists of a separate entry and exit point from each storage cell.
- Scan allows predetermined data configurations to be entered into storage, and the output of particular configurations can be compared with known correct output configurations.

Configurability

- The **configurability of a processor** refers to its ability to be tailored or modified to suit specific requirements, workloads, or tasks.

- It is an important feature in many processor designs, particularly in embedded systems, specialized computing environments, and custom applications.

- Programmable Hardware (FPGAs & Custom Logic):
- Configurable Cores (RISC-V, ARM, etc.):
- Software-Defined Architecture:
- Dynamic Voltage and Frequency Scaling (DVFS):

Configurability

Programmable Hardware (FPGAs & Custom Logic):

- **FPGA (Field Programmable Gate Arrays):**
  - These are highly configurable processors, where logic blocks and interconnections can be reprogrammed to execute specific tasks.
  - Engineers can design custom circuits and optimize them for specialized applications, such as AI, signal processing, or cryptography.

- **ASICs (Application-Specific Integrated Circuits):**
  - These are custom-built processors that are designed to perform specific tasks efficiently, though
  - they are less flexible than FPGAs since their logic is fixed after manufacturing.

Configurability

Configurable Cores (RISC-V, ARM, etc.):

- **RISC-V:**
  - This is an open-source instruction set architecture (ISA) that allows developers to create highly customized processor designs by selecting only the features they need (e.g., floating-point units, vector processing)
  - It provides flexibility for designing low-power, high-performance processors for specific applications.

- **ARM Cores:**
  - ARM-based processors offer various configurations, with cores like Cortex-A (high-performance), Cortex-R (real-time processing), and Cortex-M (microcontrollers).
  - ARM licenses its designs, and manufacturers can customize implementations to add specific features, such as security modules or optimized cache systems.

Configurability

Software-Defined Architecture:

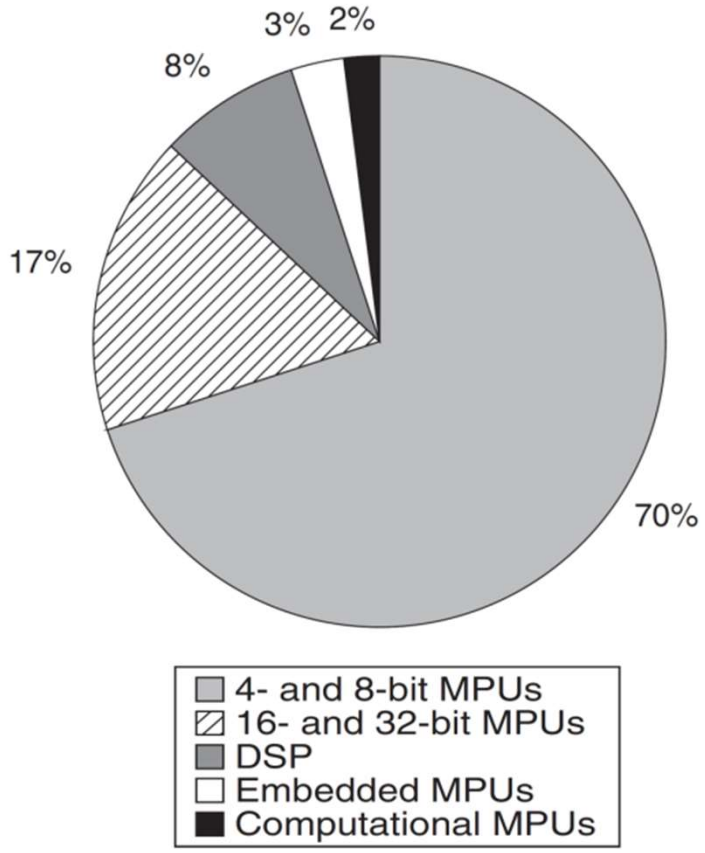
- Some processors allow for reconfiguration through software, allowing developers to optimize or reassign resources like cores, memory hierarchies, or instruction sets based on the current task.
- This is common in **data centers** and **edge computing** environments, where workloads vary dynamically.

Dynamic Voltage and Frequency Scaling (DVFS):

- This allows the processor to adjust its voltage and clock speed dynamically to balance performance and power consumption based on workload requirements.
- This type of configurability helps extend battery life in mobile devices or reduce energy usage in servers.

SOC: Processors Used

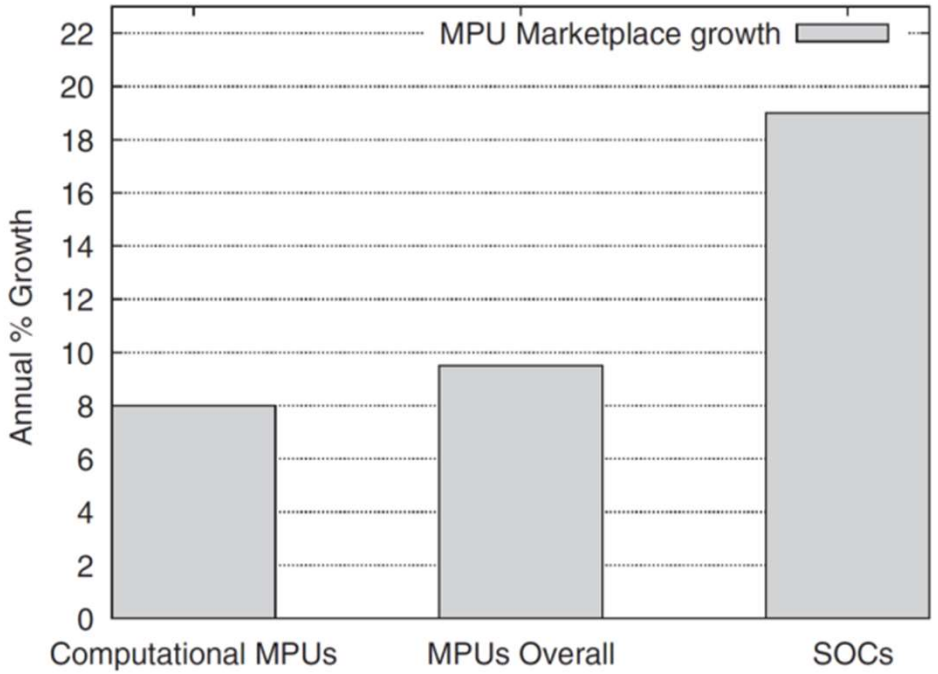
- Processors come in many types and with many intended uses.
- Much attention is focused on high - performance processors used in servers and workstations.
- Figure shows the processor production profile by annual production count



Worldwide production of microprocessors and controllers

Introduction

- Market growth, shows that the demand for SOC and larger microcontrollers is growing at almost three times that of microprocessor units.
- In SOC type applications, the processor itself is a small component occupying just a few percent of the die.
- SOC designs often use many different types of processors suiting the application.



Annual growth in demand for microprocessors and controllers

Processor Selection For SOC

- For SOC designs, the **selection of the processor** is the **most obvious task** and the **most restricted.**

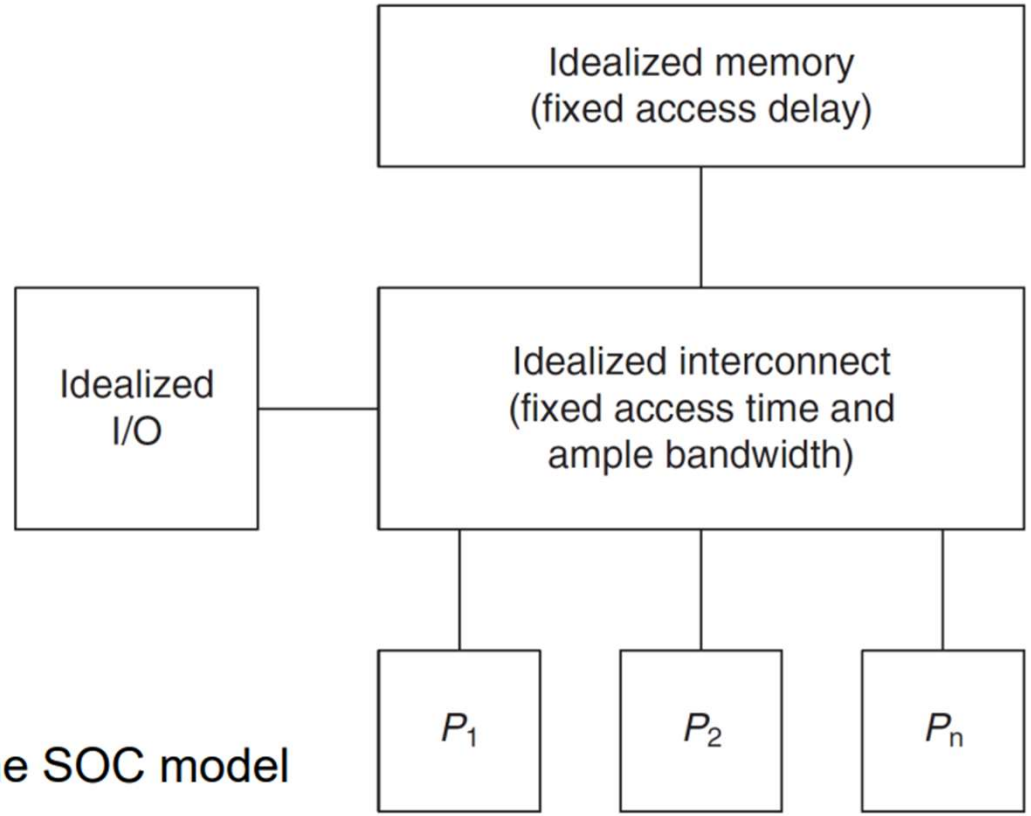
- The **processor must run a specific system software,** so at least a **core processor** (usually a general - purpose processor (GPP)) must be **selected for this function.**

- In computation - limited applications, the **system includes a processor** configured and parameterized to **meet requirements.**

Processor Selection For SOC

- In **some cases,** it may be possible **to merge these processors**, but that is usually an optimization consideration.
- **Memory and interconnect components** are considered as **simple delay elements** in **calculating processor performance**.
- These are referred to here as idealized components.

Processor Selection For SOC



Processors in the SOC model

- Figure shows the processor model used in the initial design process.

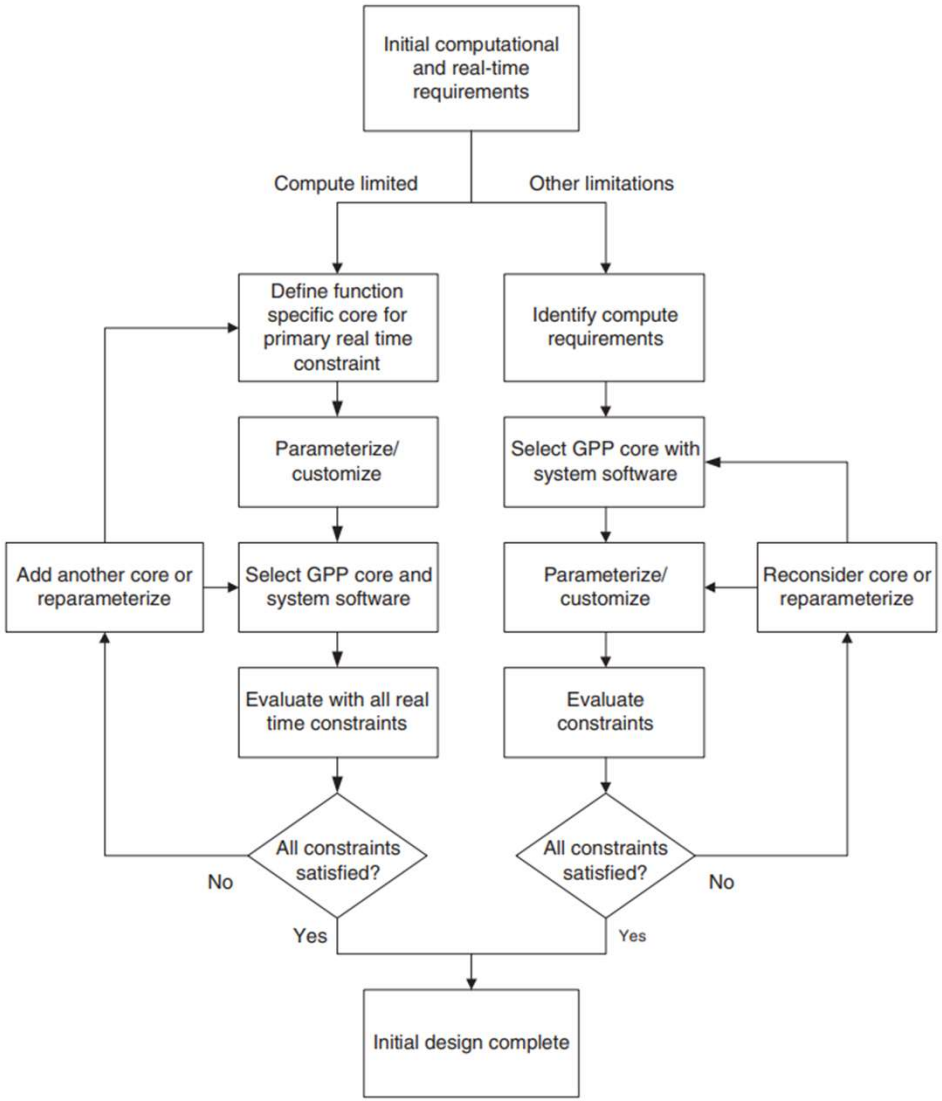
Process of processor core selection

- The process of selection is different **in the case of compute - limited selection,** as there can be a **real – time requirement** that must be met by one of the selected processors.

- The **processor selection and parameterization** should **result in an initial SOC design** that appears to fully satisfy all functional and performance requirements set out in the specifications.

decision that depends on the target application, power efficiency, performance requirements, and the scalability of the design.

Process of processor core selection



Soft Processors

- A **soft processor** is an **Intellectual Property (IP)** core that is implemented using the logic primitives of the FPGA.
- **Being soft** it has high degree of **flexibility** and **configurability.**
- **Soft processor** is a **microprocessor core** that can be **entirely implemented using logic synthesis.**
- It **can be implemented** via different **semiconductor devices containing programmable logic** (e.g., ASIC, FPGA, CPLD).

Soft processors

- **Soft processors** are not fabricated in silicon but are instead defined using hardware description languages (HDLs) like Verilog or VHDL, and

- Their architecture can be customized or modified to meet specific requirements.

Soft Processors

- **Most systems**, uses a **single soft processor.** However, a few designers may use many soft cores onto an FPGA.

- While **many people put exactly one soft microprocessor on a FPGA**. A sufficiently **large FPGA** can **hold two or more soft microprocessors,** resulting in a **multi-core processor.**

- The **number of soft processors on a single FPGA** is only **limited by the size of the FPGA.**

Soft Processors

- The term **"soft core"** refers to an **instruction processor** design in bitstream format that can be used to program a FPGA device.

- The **4 main reasons** for using such designs, despite their large area – power – time cost, are
  1. **Cost reduction** in terms of system - level integration,
  2. **Design reuse** in cases where multiple designs are really just variations on one,
  3. **Creating an exact fit** for a microcontroller/peripheral combination, and
  4. **Providing future protection** against discontinued microcontroller variants.

Processor Core Selection

Processor core selection typically revolves around two main pathways: **General Core Path** and **Compute Core Path**.

General Core Path

- The **General Core Path** is aimed at delivering balanced performance across a wide range of tasks.
- It is designed for general-purpose computing systems, like personal computers, smartphones, or embedded systems where efficiency and multitasking capabilities are essential.

- **ARM Cortex-A series** (for mobile and embedded systems)
- **Intel Core i-series** (for desktops/laptops)
- **AMD Ryzen** (for desktops/laptops)

Processor Core Selection

Compute Core Path

- The **Compute Core Path** is designed for performance-centric applications that demand heavy processing power, such as high-performance computing (HPC), data centers, AI/ML tasks, or scientific simulations.
- These cores focus on delivering maximum raw computational power.

•**NVIDIA CUDA cores** (for GPU-based parallel computing)
•**Intel Xeon** (for data centers and servers)
•**AMD EPYC** (for high-performance computing)
•**Google TPUs** (for AI/ML tasks)

Processor Core Selection (General Core Path)

- *Assume* that an **initial design had performance of 1 using 100K rbe of area**, and we would like **to have additional speed and functionality.**
- So **we double the performance** (half the *T for the processor).*
- *This **increases the** area to 400K rbe* and the **power by a factor of 8.**
- **Each rbe** is now **dissipating twice the power** as before.
- **Doubling the performance** (instruction execution rate) **doubles the number of cache misses** per unit time.

follow the AT^2 rule

Processor Core Selection (General Core Path)

- **Cache misses** significantly reduces the **realized performance**; to recover this performance, **we now need to increase the cache size.**

- The general rule **to half the miss rate**, we need to **double the cache size.**

- **If the initial cache size was also 100K rbe**, then **new design will have cache size of 600K rbe** and **probably dissipates about 10 times the power** of the initial design.

- The faster processor cache combination may provide important functionality, such as additional security checking or input/output (I/O) capability.

Processor Core Selection (Compute Core Path)

- Consider **some trade - offs for the compute - limited path.**
- Suppose the **application is generally parallelizable**, and we have several different design approaches.
- One is a **10 - stage pipelined vector processor**; the other is **multiple simpler processors.**
- The **application has performance of 1 with the vector processor** (area is 300K rbe) and **half of that performance with a single simpler processor** (area is 100K rbe).
- In order to **satisfy the real – time compute requirements**, we need to **increase the performance to 1.5**

Processor Core Selection (Compute Core Path)

- Now we must evaluate the various ways of achieving the target performance.
- **Approach 1** is to **increase the pipeline depth** and **double the number of vector pipelines**; this satisfies the performance target.
- This increases the area to 600K rbe and doubles the power, while the clock rate remains unchanged.

Processor Core Selection (Compute Core Path)

- Now we must evaluate the various ways of achieving the target performance.
- **Approach 2** is to **use an "array" of simpler interconnected processors.**
- In order **to achieve the target performance**, we **need to have at least four processors**: **three for the basic target** and **one to account for the overhead.**

Over Head

In CPU design, *overhead* refers to the additional resources or time required for tasks that don't directly contribute to computational performance but are necessary for managing or coordinating activities within the CPU. Overhead can come in many forms:

- **Instruction Overhead**
- **Memory Access Overhead**
- **Context-Switching Overhead**
- **Pipeline Overhead**
- **Parallelism Overhead**
- **Control Overhead**

Steps for Designing CPU

- **Design of instruction set [IS]**
  - Define instruction set, number of instructions
  - Define each instruction
  - Instruction Encoding (OPCODE)
- **Design of Data path**
  - Define number of functional unit required for IS
  - Decide Number of registers needed
  - Use Separate registers or register file
  - Define Flow control registers, Status registers etc.
  - Connect the functional unit and register to implement [IS]
- **Design of Control unit**
  - Program Counter (PC)
  - Instruction Register (IR)
- **Instruction Cycle**
  - Step 1 fetches an Instruction
  - Step 2 Decodes the Instruction
  - Step 3 executes the Instruction

Basic Concepts In Processor Architecture

- The **processor architecture consists** of the ***instruction set*** *of the processor.*
- While the **instruction set implies many implementation** (microarchitecture) details.
- It is the **synthesis of the physical device** limitations with **area – time – power trade - offs** to optimize specified user requirements.

Instruction Set

- The **instruction set** for most processors is **based upon a register set** to hold operands and addresses.
- The **register set size can be varied from 8 to 64 words** or more, where **each word consists of 32 – 64 bits.**
- An additional **set of floating - point registers (32 – 128 bits)** can also be used.
- A typical **instruction set specifies a program status word**, which consists of various types of **control status information, including condition codes** (CCs) set by the instruction.