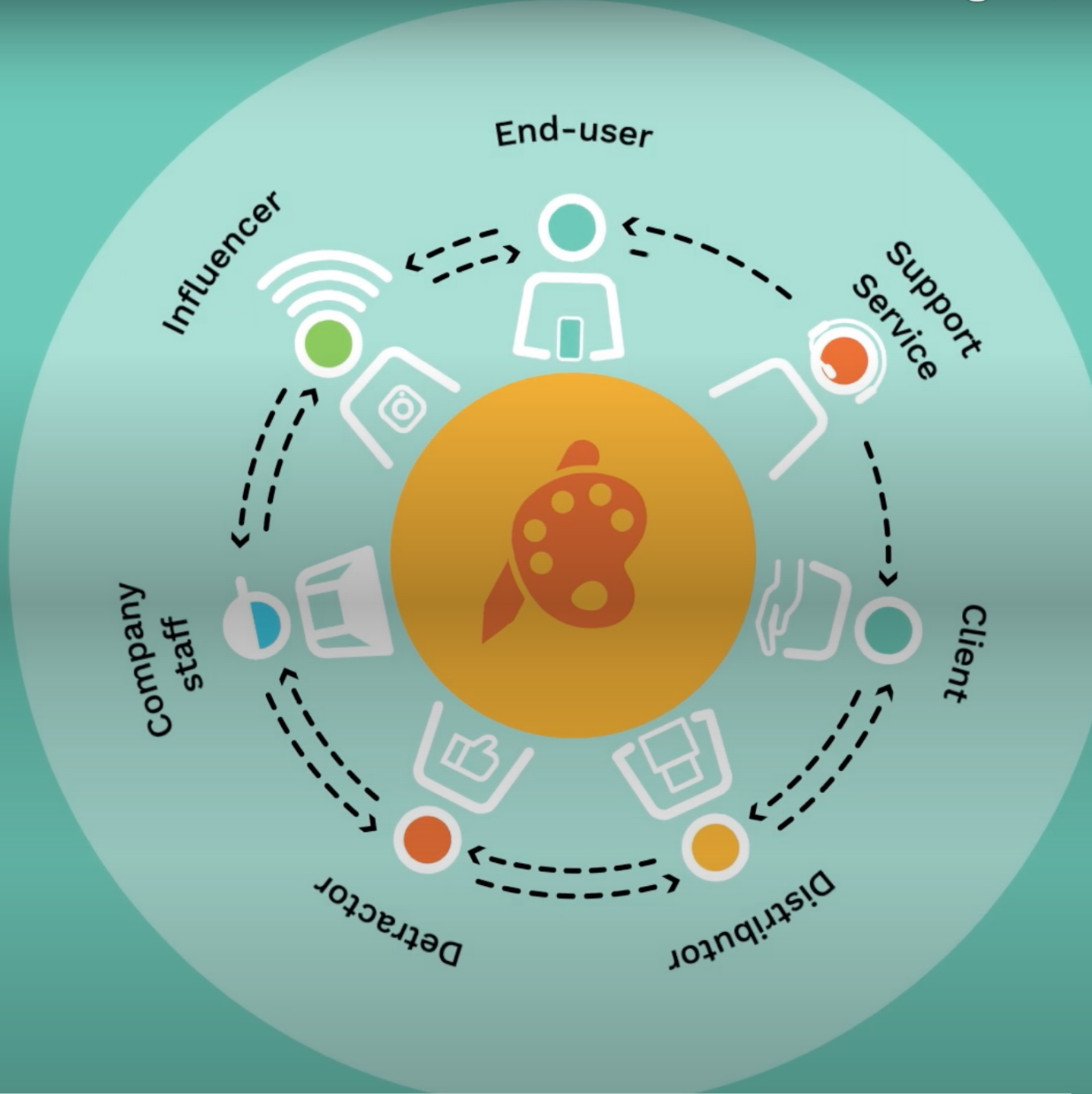
Co-Design

- All relevant stakeholders involved
- Active collaboration between users and designers



Key steps of a Co-Design process

Engage

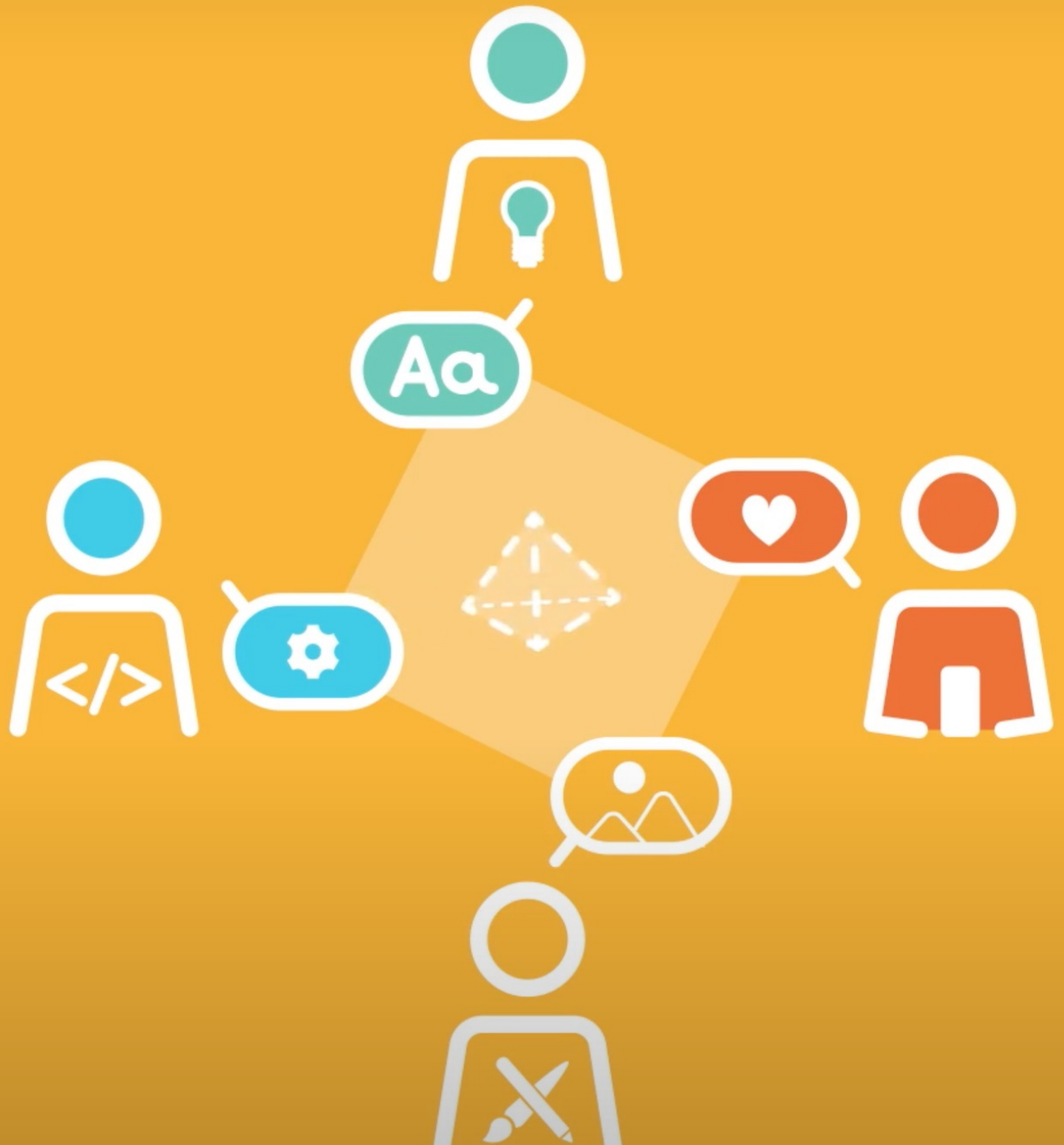
- Learn from each other
- Set the challenge

Understand

- Focus on user needs
- Gather key insights

Ideate

- Design concepts
- Build prototypes



Validate

- Present
- Test
- Evaluate

Fundamental Challenges

Computational models (model selection)

Architecture selection

Language Selection

Partitioning of System Requirements in both H/W & S/W

H/S-S/W Trade-off

Integration of Hardware and Software

ICE (In circuit emulator)

Model Selection

- Model captures and describe system characteristics.
- Model contains objects and composition rules
- It is hard to decide which model should be followed in a particular system design.
- Designers switch between a variety of models depending on the requirements.

Architecture Selection

- Model only describes the system characteristics , it does not provide the information on how the system is manufactured.
- But architecture specifies ho a system is going to be implemented in terms of components and interconnections.
- RISC,CISC,SIMD(Single Instruction Multiple Data),MIMD are common architectures.

Language Selection

- Programming language captures computational model.
- A model can be captured using multiple programming languages like C,C++,C#,JAVA which is good for s/w implementation.
- C++ is a good candidate for capturing object oriented model.
- The pre-requisite for selecting a language is that it should capture the model easily.

Partitioning system requirement

- It may be possible to implement the system requirements in either hardware or software (firmware)
- Performance, re-usability, effort etc are used for making a decision on the h/w s/w partitioning.

Computational Models

- Data Flow Graph Model(DFG)
- Conditional Data Flow Graph Model(CDFG)
- State Machine Model
- Sequential Program Model
- Concurrent Process Model

Data flow Graph Model

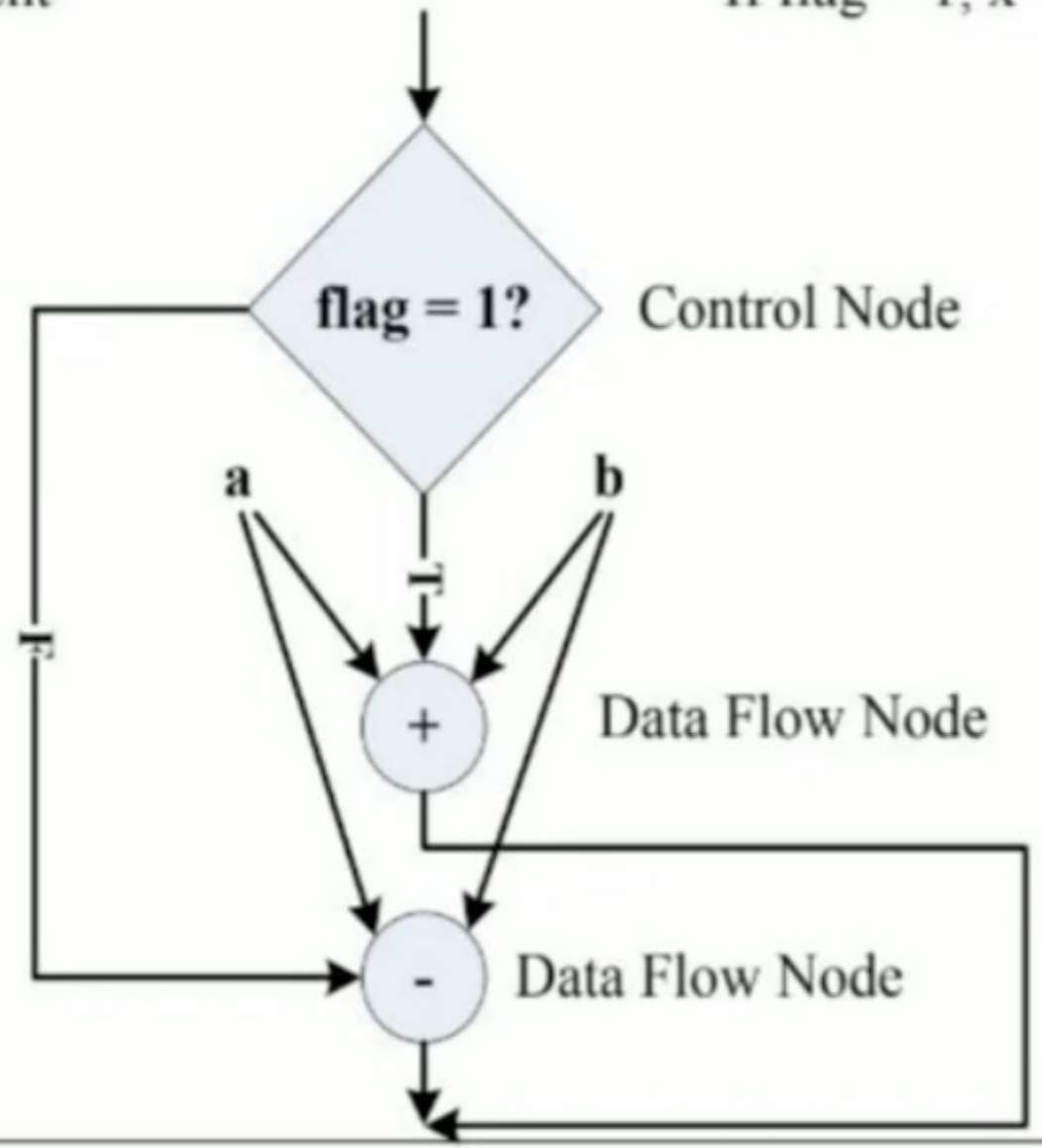
- Data Driven Model where program execution is determined by the data.
- It translates the requirements into data flow graph.
- It is a visual model where operations on the data is represented using a circle and data flow is represented using arrows
  Inward arrow to circle------INPUT DATA
  Outward arrow to circle---- OUTPUT DATA

Conditional/ Control Data Flow Model

- Contains both data operations and control operations.
- Data flow graph is elements and condtionals are decision makers.
- This model has both data nodes and decision nodes whereas DFG only has data nodes.
- Operation-----Circle Data Flow-----Arrow
  Control Node-----Diamond

Model the requirement

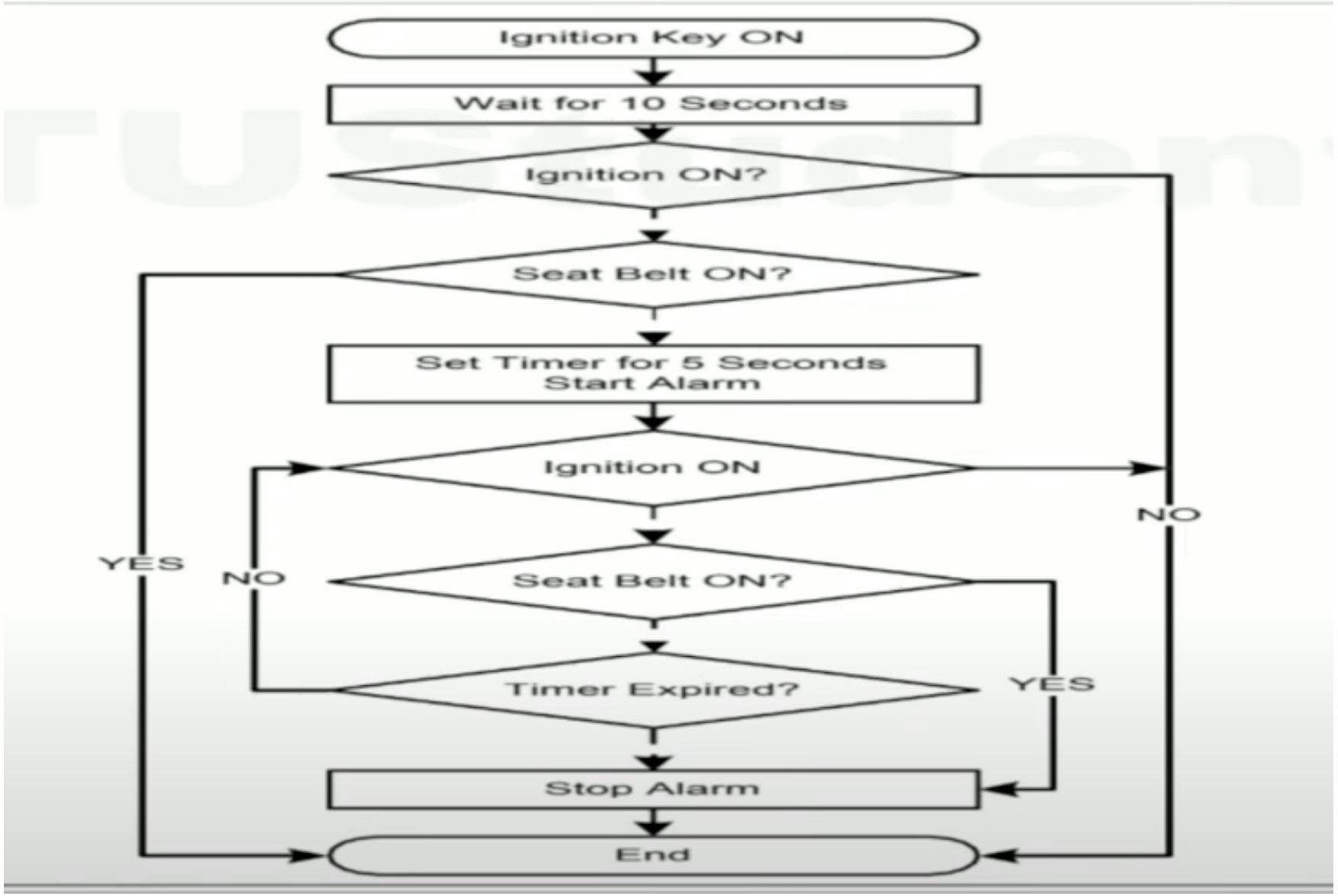
If flag = 1, x = a + b; else y = a-b;



Sequential Program Model

- The processing requirements are executed in sequence.
- Here the program instructions are iterated and executed conditionally and the data gets transformed through a series of operations.
- FSMs are good choice for sequence program model.
- Flow chart is another important tool for modeling sequencial programs.

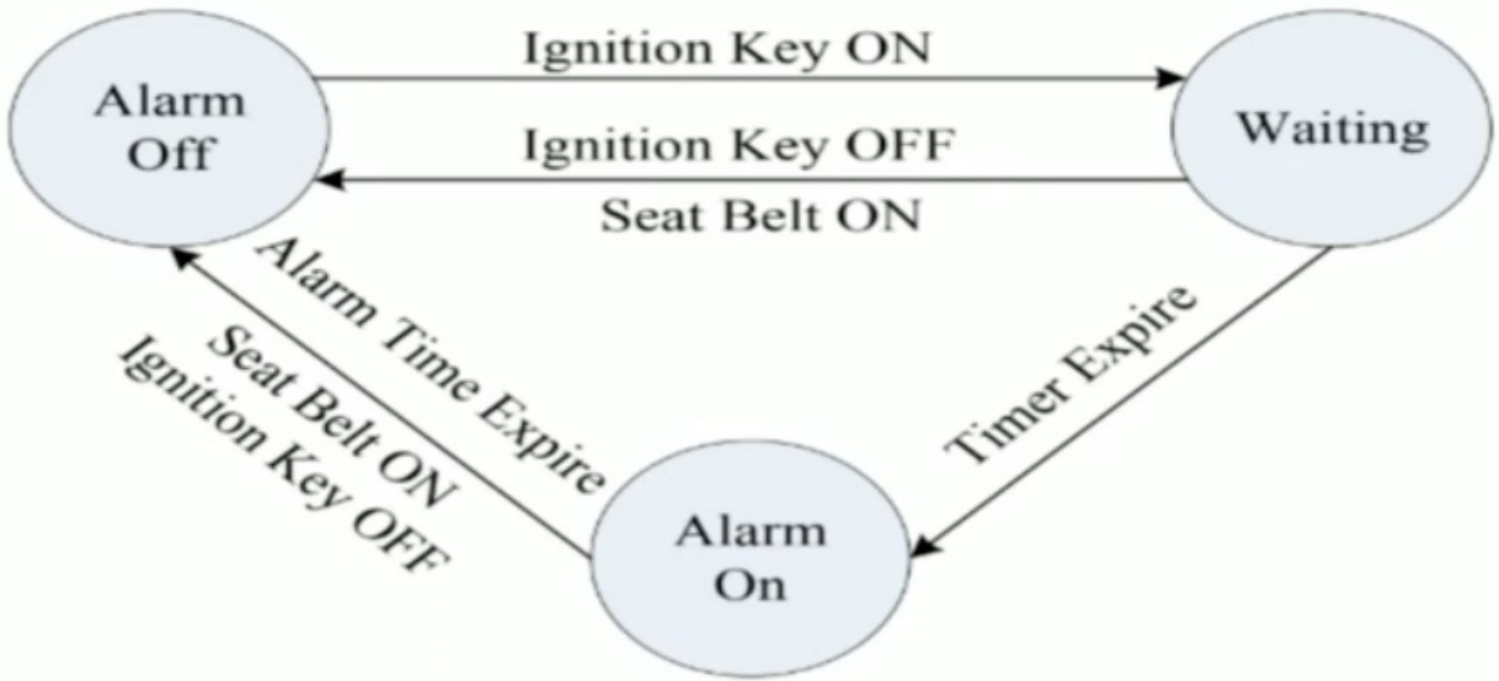
- FSM----represents states,events,transitions and actions.
- Flowchart----models the execution flow.
- Eg: Automatic Seat Belt Warning System.
- Requirements: The system should generate an alarm for 5 seconds if the ignition is ON and if the seat belt is not tightened within 10 seconds. The alarm is turned off when alarm time expires or if seatbelt is fastened or if the ignition is ON.


Ignition Key ON
Wait for 10 Seconds
Ignition ON?
Seat Belt ON?
Set Timer for 5 Seconds
Start Alarm
Ignition ON
Seat Belt ON?
Timer Expired?
Stop Alarm
End
YES
NO
NO
YES
YES

State Machine Model

- Here requirements are translated into sequence driven program.
- FSM –Finite State Machine Model
  In this model states are finite
- HCFSM---Hierarchical/Concurrent Finite State Machine Model.
- Eg: Automatic Seat Belt Warning

State Diagram for Automatic Seat Belt Warning system



Concurrent Process Model

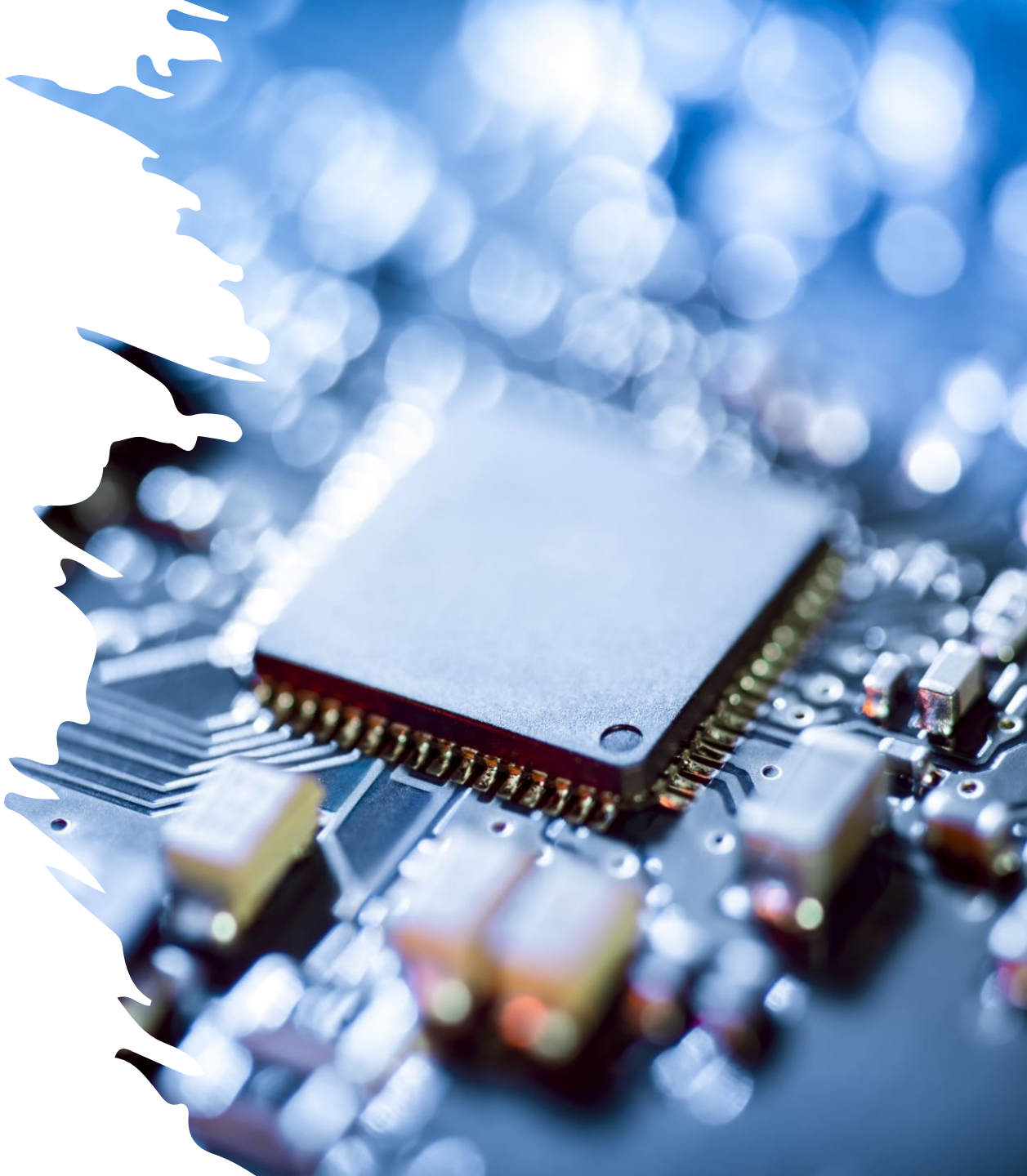
- This is the model executing tasks concurrently.
- Certain processing requirements are easier to model in concurrent processing model than conventional sequential model.
- Sequential execution leads to a single sequential execution of task which leads to poor processor utilisation whereas in this model CPU usage is done effectively.

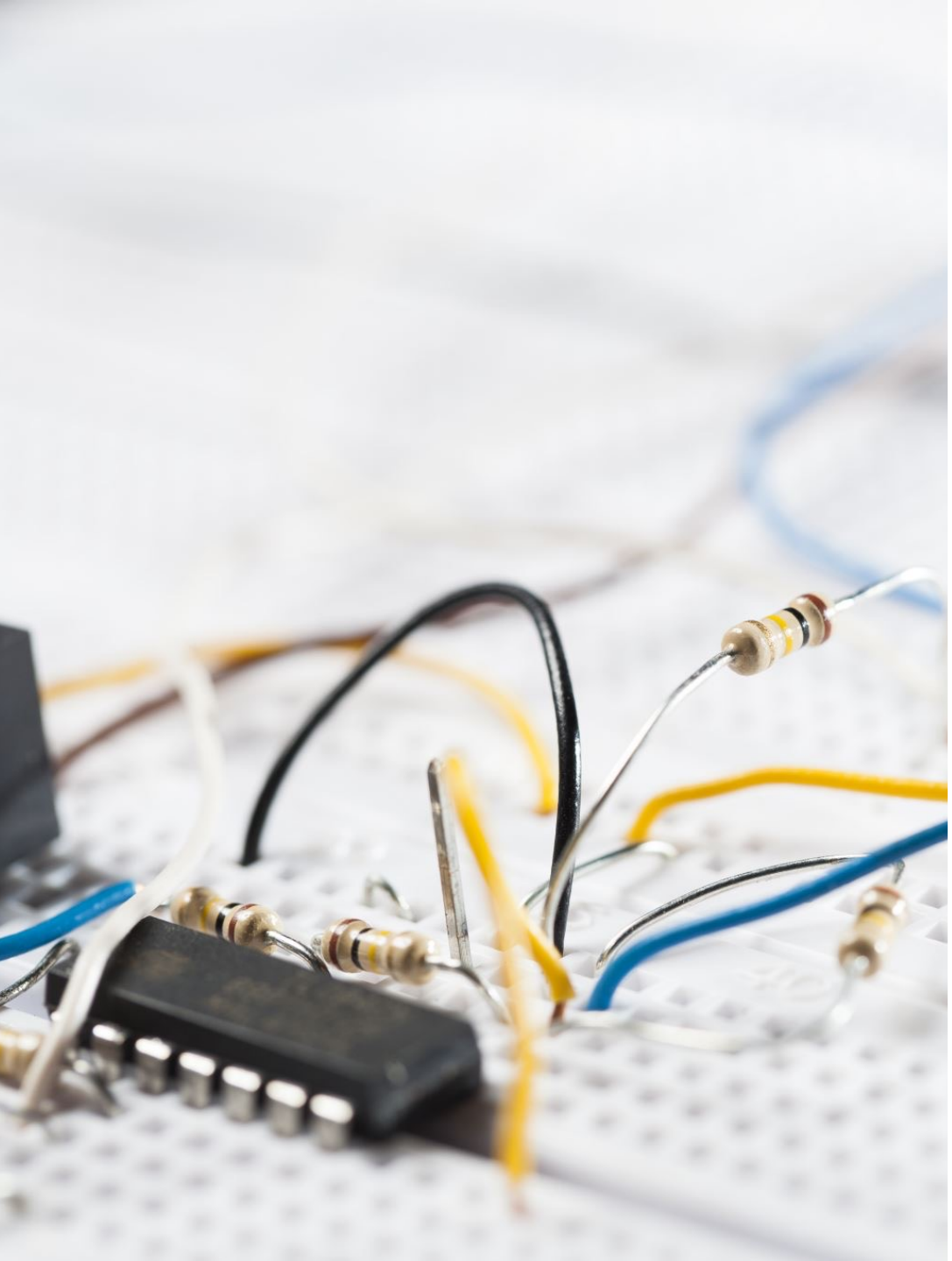
- The processing requirements are split into multiple tasks.
- The tasks are executed concurrently.

Steps in the Design Process

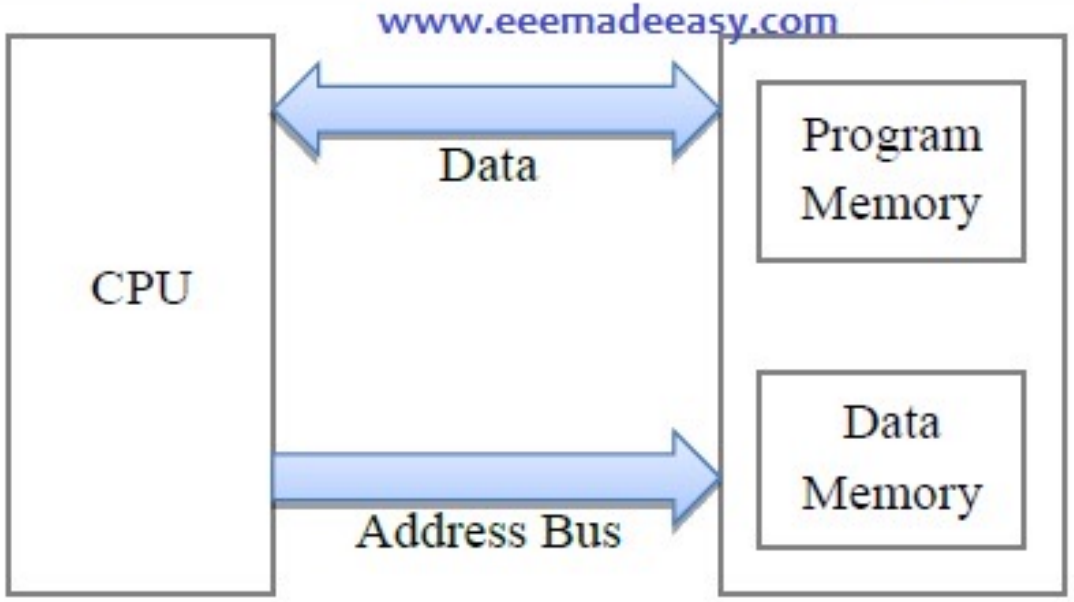
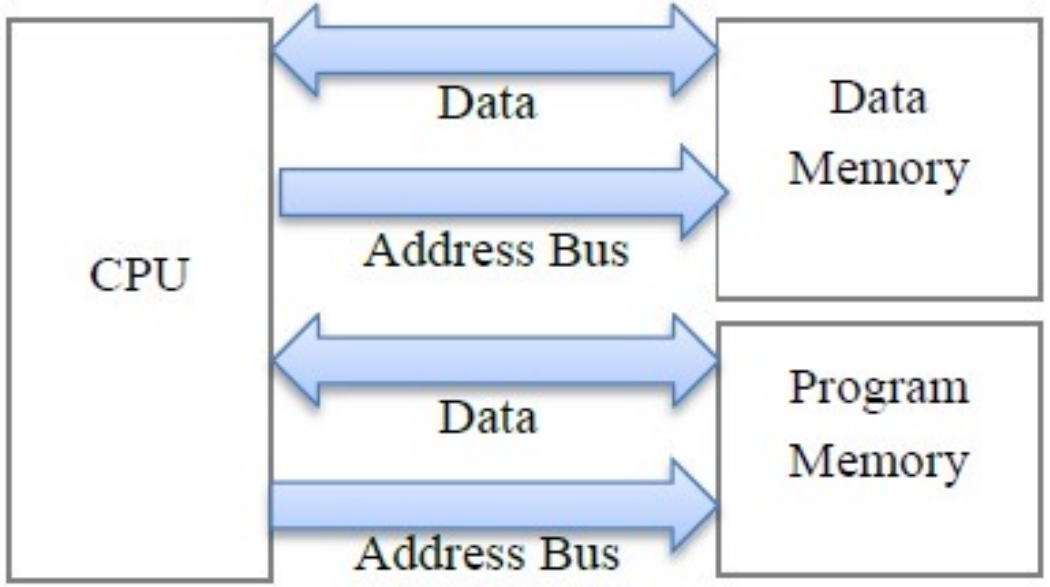
- microprocessors
- micro-controller

Micro-processor is based on von Neumann model/architecture (where program + data resides in the same memory location), it is an important part of the computer system, where external processors and peripherals are interfaced to it. It occupies more area and has more power consumption. The application of the microprocessor is personal computers.





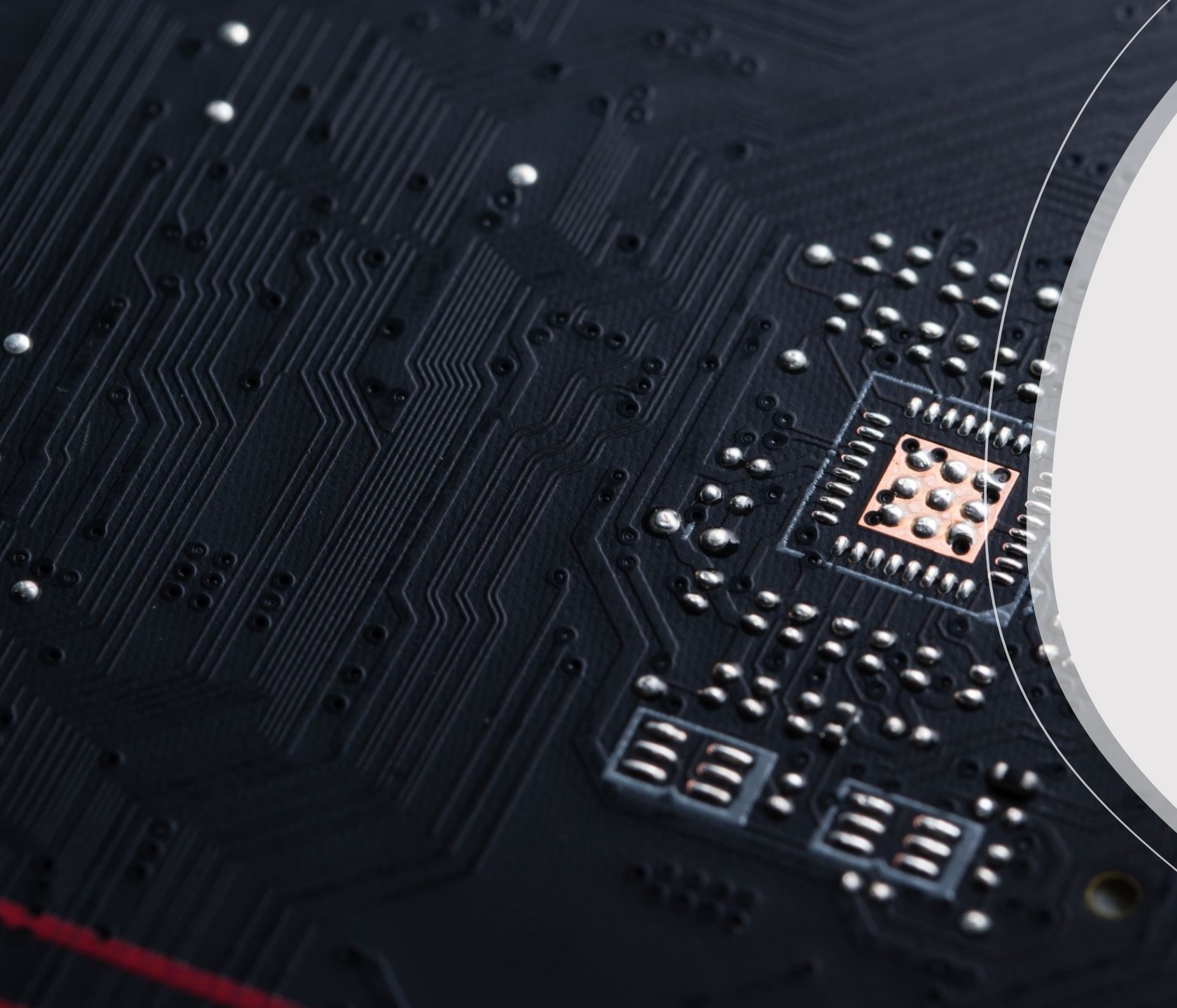
Micro-controller is based on Harvard architecture, it is an important component of an embedded system. External processor, internal memory and i/o components are interfaced with the microcontroller. It occupies less area, less power consumption.

Von-Neumann (Princeton architecture)	Harvard architecture
	

Von-Neumann Architecture	Harvard Architecture
Single memory to be shared by both code and data.	Separate memories for code and data.
Processor needs to fetch code in a separate clock cycle and data in another clock cycle. So it requires two clock cycles.	Single clock cycle is sufficient, as separate buses are used to access code and data.
Higher speed, thus less time consuming.	Slower in speed, thus more time-consuming.
Simple in design.	Complex in design.

Types of Embedded Systems

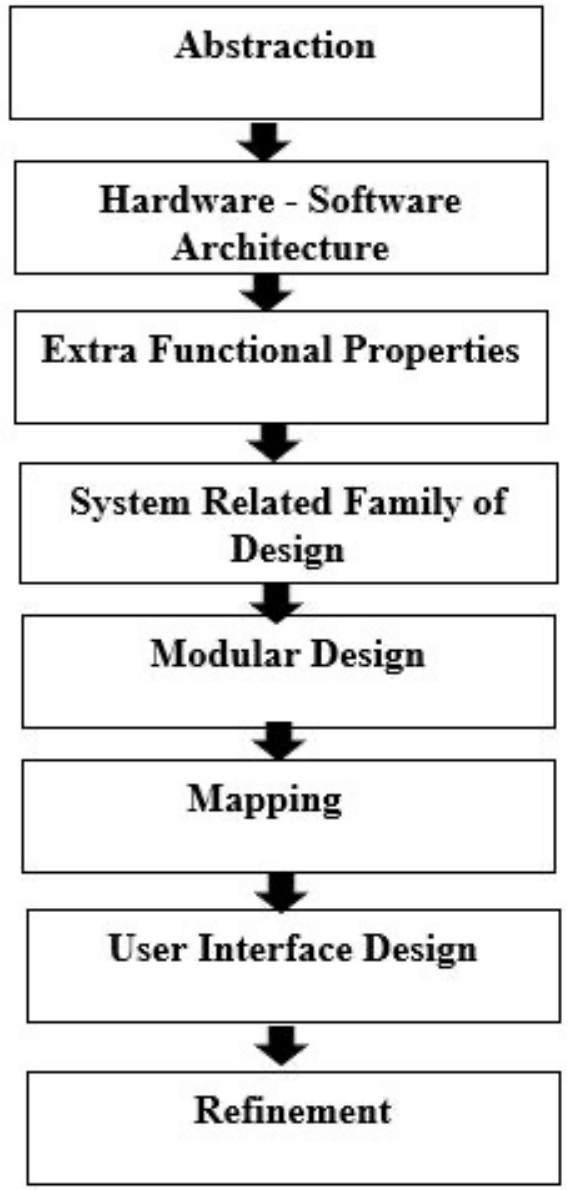
- Stand-Alone Embedded System
- Real-Time Embedded System
- Networked Appliances
- Mobile devices



Elements of Embedded Systems

- Processor
- Microprocessor
- Microcontroller
- Digital signal processor.

Steps in the Embedded System Design Process



Design Metrics / Design Parameters of an Embedded System	Function
Power Dissipation	Always maintained low
Performance	Should be high
Process Deadlines	The process/task should be completed within a specified time.
Manufacturing Cost	Should be maintained.
Engineering Cost	It is the cost for the edit-test-debug of hardware and software.
Size	Size is defined in terms of memory RAM/ROM/Flash Memory/Physical Memory.
Prototype	It is the total time taken for developing a system and testing it.
Safety	System safety should be taken like phone locking,

Embedded System Design Software Development Process Activities

Safety	System safety should be taken like phone locking, user safety like engine break down safety measure must be taken
Maintenance	Proper maintenance of the system must be taken, in order to avoid system failure.
Time to market	It is the time taken for the product/system developed to be launched into the market.

Embedded System Design Software Development Process Activities

CISC	RISC
Larger set of instructions. Easy to program	Smaller set of Instructions. Difficult to program.
Simpler design of compiler, considering larger set of instructions.	Complex design of compiler.
Many addressing modes causing complex instruction formats.	Few addressing modes, fix instruction format.
Instruction length is variable.	Instruction length varies.
Higher clock cycles per second.	Low clock cycle per second.
Emphasis is on hardware.	Emphasis is on software.
Control unit implements large instruction set using micro-program unit.	Each instruction is to be executed by hardware.
Slower execution, as instructions are to be read from memory and decoded by the decoder unit.	Faster execution, as each instruction is to be executed by hardware.
Pipelining is not possible.	Pipelining of instructions is possible, considering single clock cycle.

Embedded Software Development Process Activities

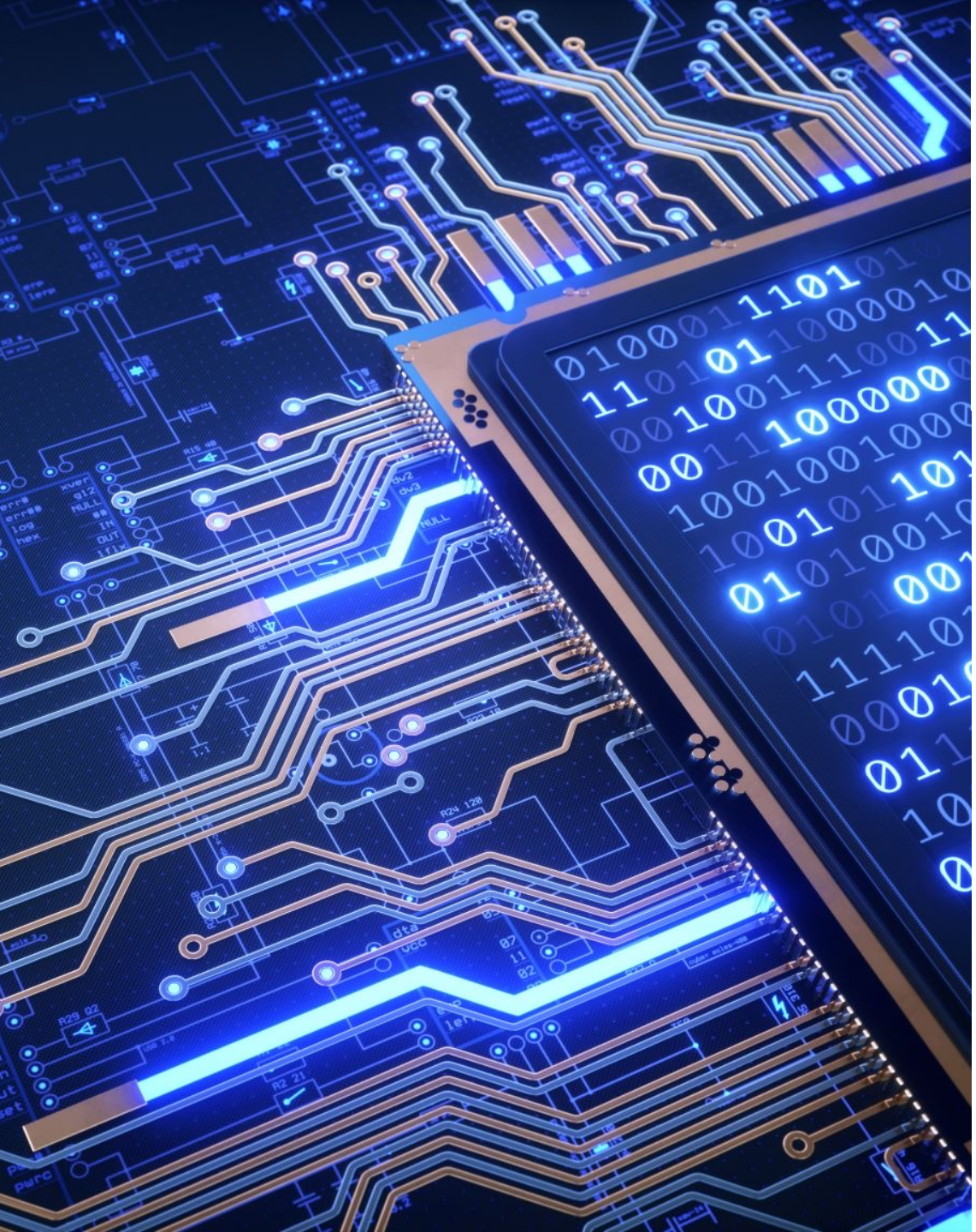
- **Specifications**
- **Architecture**
- **Components**
- **System Integration**

Processors

Program Flow Control Unit (CU)

Execution Unit (EU)

CISC and RISC

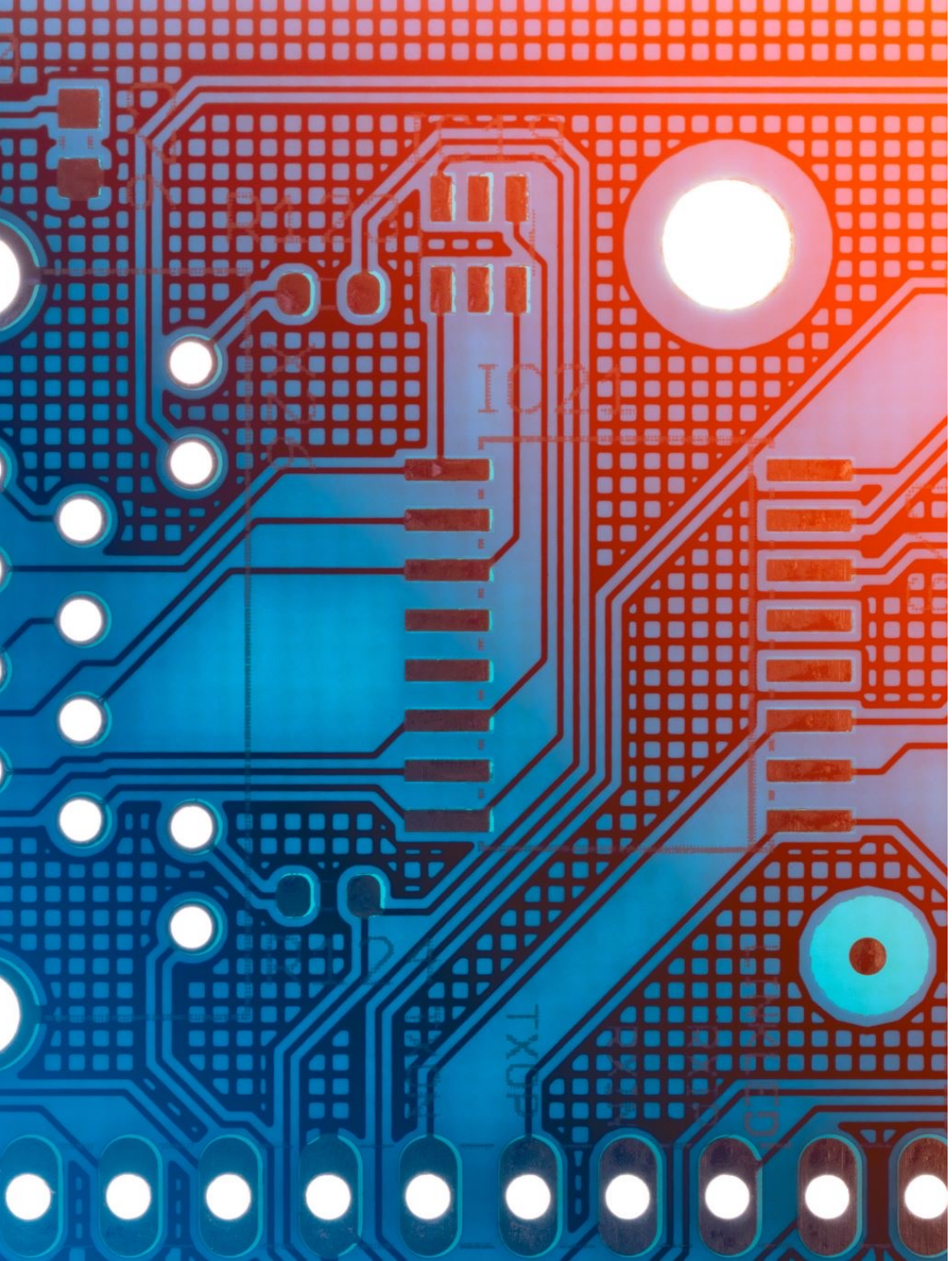


Types of Processors

- General Purpose Processor (GPP)
  - Microprocessor
  - Microcontroller
  - Embedded Processor
  - Digital Signal Processor
  - Media Processor
- Application Specific System Processor (ASSP)
- Application Specific Instruction Processors (ASIPs)
- GPP core(s) or ASIP core(s) on either an Application Specific Integrated Circuit (ASIC) or a Very Large Scale Integration (VLSI) circuit.

Digital system designs consists of hardware components and software program that execute on the hardware platform

Hardware-Software Codesign?



Microelectronics Trend

- Better Device Technology
- -Reduced in Device Sizes
- -More on chip devices> Higher density
- - higher Performances

- Higher Degree of Integration
- -increased device reliability
- -inclusion of complex designs

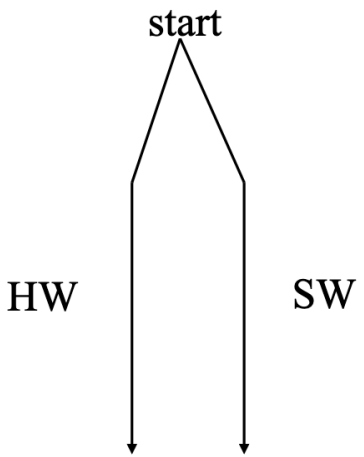
Digital System

- Judged by it’s objectives in application domain
- -Performance
- -Design and manufacturing cost
- -Ease of Programmability

Depends on both the hardware and software components

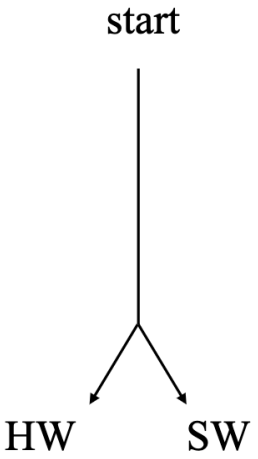
CoDesign

Traditional design flow



Designed by independent groups of experts

Concurrent (codesign) flow



Designed by Same group of experts with cooperation

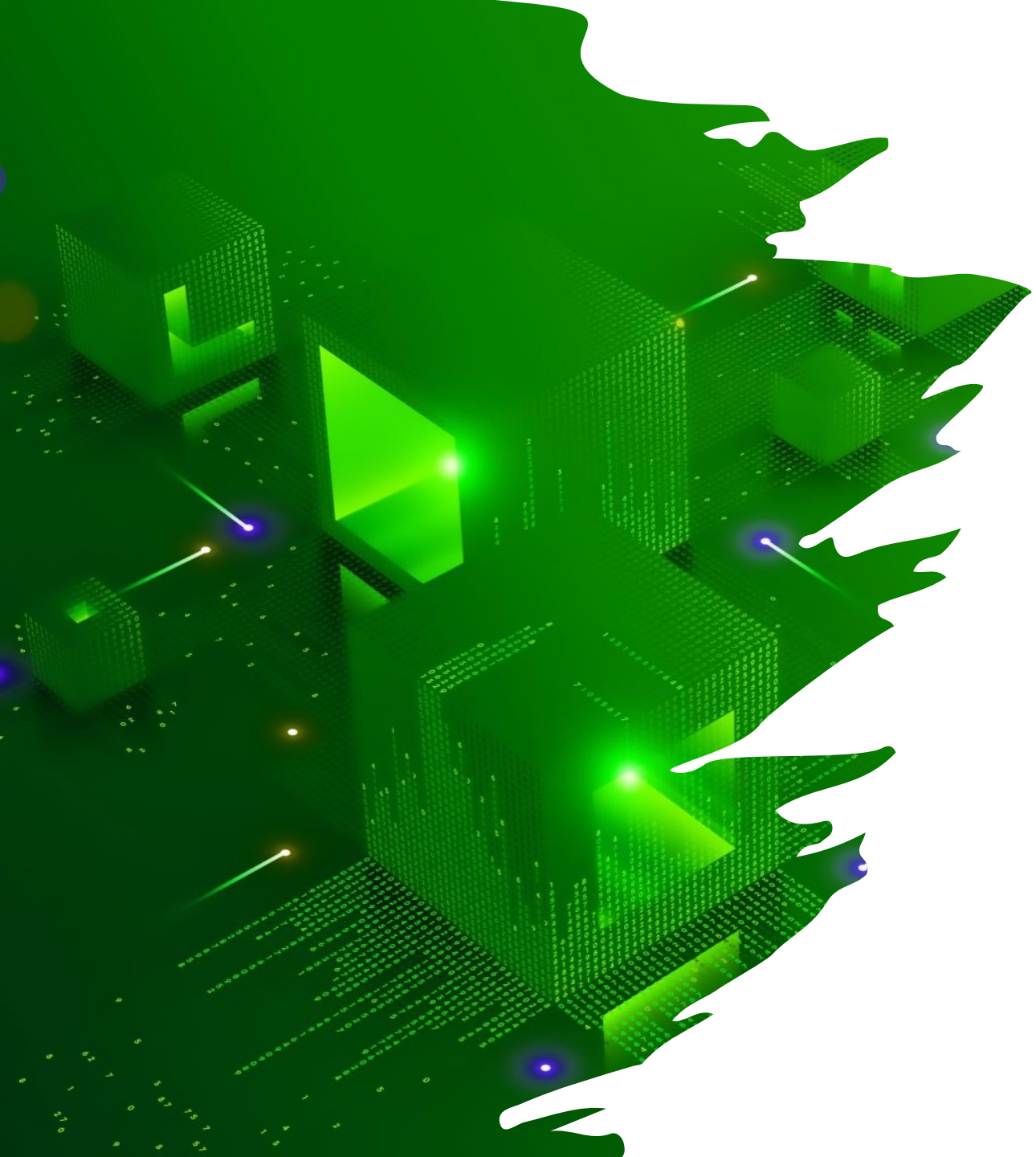
CoDesign Motivation

Trend towards smaller mask-level geometries leads to;

Higher integration and cost of fabrication

Amortize hardware design over large volume productions

* Use s/w as a means of differentiating products based on the same hardware platform



IP Cores

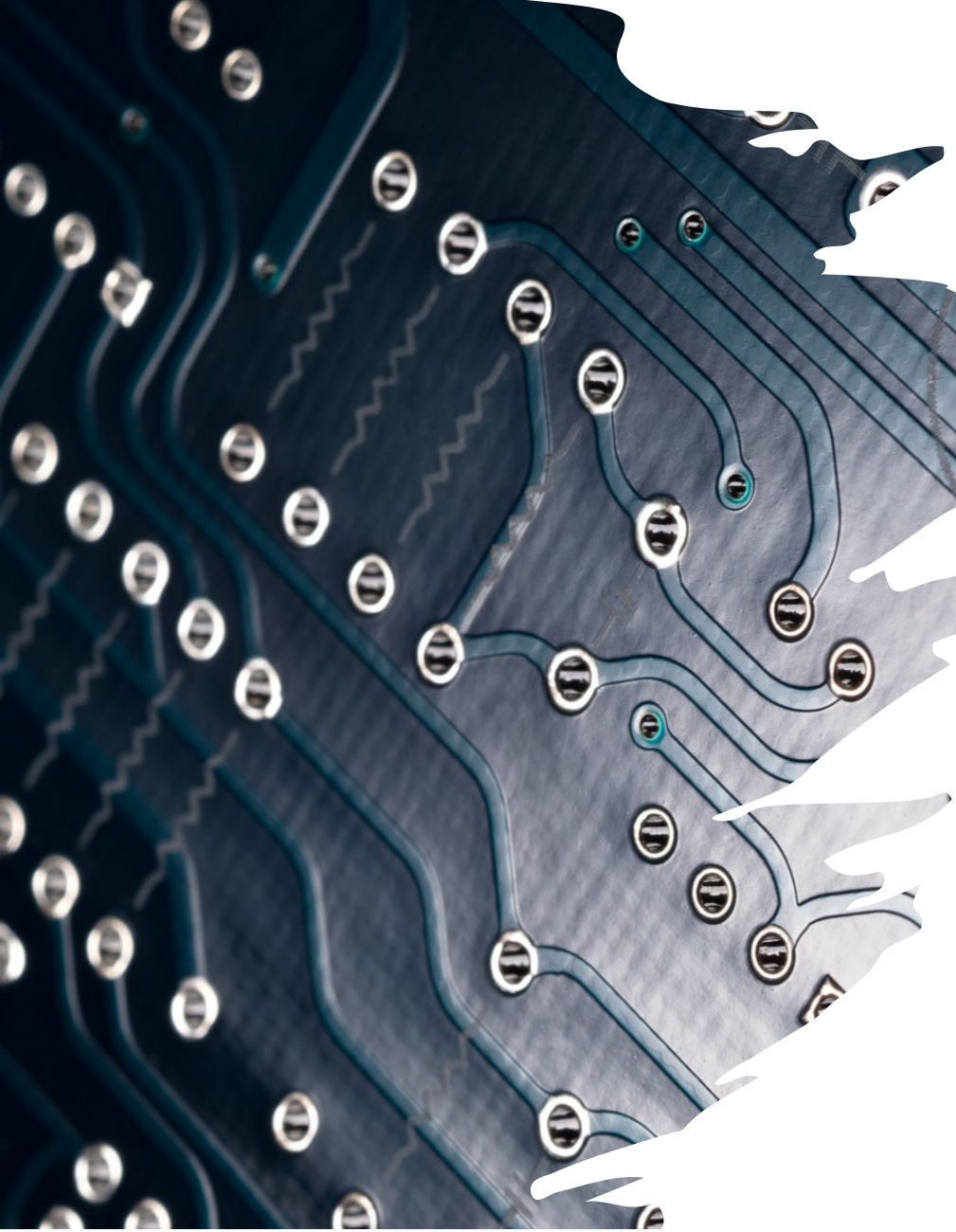
- Pre designed, pre-verified silicon circuit block, usually containing 5000 gates, that can be used in building larger application on a semiconductor chip.

- Hardware (Core)
- Software (micro cornels)

- Cores are standardized for use as system building blocks.

-Leveraging the existing software layers including OS and applications in ES.

- This results in customized VLSI Chip with better area/performance/power trade-offs and Systems on Silicon



Hardware Programmability

- Traditionally

  Hardware used to be configured at the time of manufacturing

  Software is variant at run time.

However, the FPGA (Field programmable Gate Arrays) has blurred this distinction.

FPGA circuits can be configured on the fly to implement a specific s/w function with better performances than on microprocessor.

FPGA can be reprogrammed to perform another specific function without changing the underlying hardware.

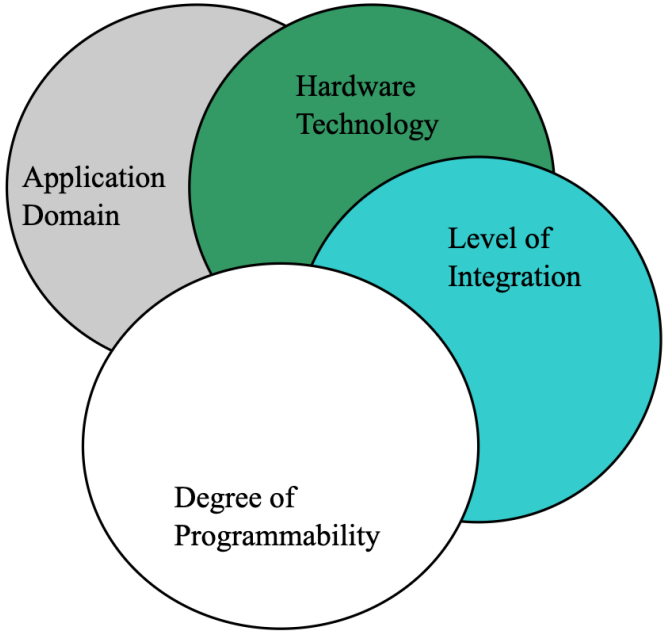
This flexibility opens new avenues in digital circuit design for various applications

Why codesign?

- Reduce time to market
- Achieve better design
  - Explore alternative designs
  - Good design can be found by balancing the HW/SW
- To meet strict design constraint
  - power, size, timing, and performance trade-offs
  - safety and reliability
  - system on chip

Distinguishing features of digital system

- Interrelated criteria for a system design



Application Domains

- General purpose computing system
  - usually self contained and with peripherals
  - Information processing systems
- Dedicated control system
  - part of the whole system, Ex: digital controller in a manufacturing plant
  - also, known as embedded systems

Embedded Systems

- Uses a computer to perform certain functions
- Conceived with specific application in mind
    - examples: dash controller in automobiles, remote controller for robots, answering machines, etc.
- User has limited access to system programming
    - system is provided with system software during manufacturing
    - not used as a computer

Degree of Programmability

Most digital systems are programmed by some software programs for functionality.

Two important issues related to programming:

- who has the access to programming?
- Level at which programming is performed.

Degree of Programmability: Accessibility

Understand the role of:

End users, application developers, system integrator and component manufacturers.

Application Developer: System to be retargetable.

System Integrator: Ensure compatibility of system components

Component Manufactures: Concerned with maximizing product reuse

Degree of Programmability

Example 1: Personal computer

End User: Limited to application level

Application Dev.: Language tools, Operating System, high-level programming environment (off the self components)

Component Manf.: Drive by bus standards, protocols etc.

Observe that: coalescing the system components due to higher chip density
Result: Few but more versatile system hardware components

Example 2.

Embedded Systems

- End user: Limited access to programming
- Most software is already provided by system integrator who could be application developer too!

Level of Programmability

- Systems can be programmed at *application, instruction and hardware* levels
- Application Level: Allows users to specify “option of functionality” using special language.
- Example: Programming VCR or automated steering control of a ship

Level of Programmability

- Instruction-level programmability
  - Most common ways with ISA processors or DSP
    - compilers are used in case of computers
    - In case of embedded systems, ISA is NOT visible

Level of programmability

- Hardware level programmability

⇓

configuring the hardware (after manufacturing) in the desired way.

Example: Microprogramming (determine the behavior of control unit by microprogram)

- Emulating another architecture by alternation of µp
- Some DSP implementations too
- Never in RISC or ISA processors

Performance and Programmability

- **General computing applications**: use of superscalar RISC architecture to improve the performance (instruction level programming)
- **Dedicated Applications**: Use of application specific designs (ASICs) for power and performance
    - Neither reusable nor cheap!

What if ASICs with embedded cores?

Performance and Programmability

- Any other solutions?

  How about replacing the standard processors by application specific processors that can be programmed at instruction level (ASIPs).

  - Better power-performance than standard processor ?
  - Worse than ASICs

Programmability and Cost:ASIPs

- Cost can typically be amortized over larger volume than on ASICs (with multiple applications using ASIPs).
- Ease to update the products and engineering changes through programming the HW,
- However, includes compiler as additional cost

Hardware Technology

- Choice of hardware to implement the design affects the performance and cost
- VLSI technology (CMOS or bipolar, scale of integration and feature size etc.) can affect the performance and cost.

Hardware Technology: FPGAs

- Performance is an order of magnitude less than corresponding non-programmable technology with comparable mask size
- For high volume production, these are more expensive than ASICs
- Power Issues?

Level of Integration

- Integration leads to reducing number of parts, which means, increased reliability, reduced power and higher performances
- But it increases the chip size (cost) and makes debugging more challenging.
- Standard components for SoC are cores, memory, sensors and actuators.

Embedded System Design Objective

- Embedded systems:

  control systems: reactive, real-time

  - ♦ function & size: micro controller to high throughput data-processor
    - requires leveraging the components and cores of microprocessors
- reliability, availability and safety are vital
  - use of formal verification to check the correctness
  - may use redundancy

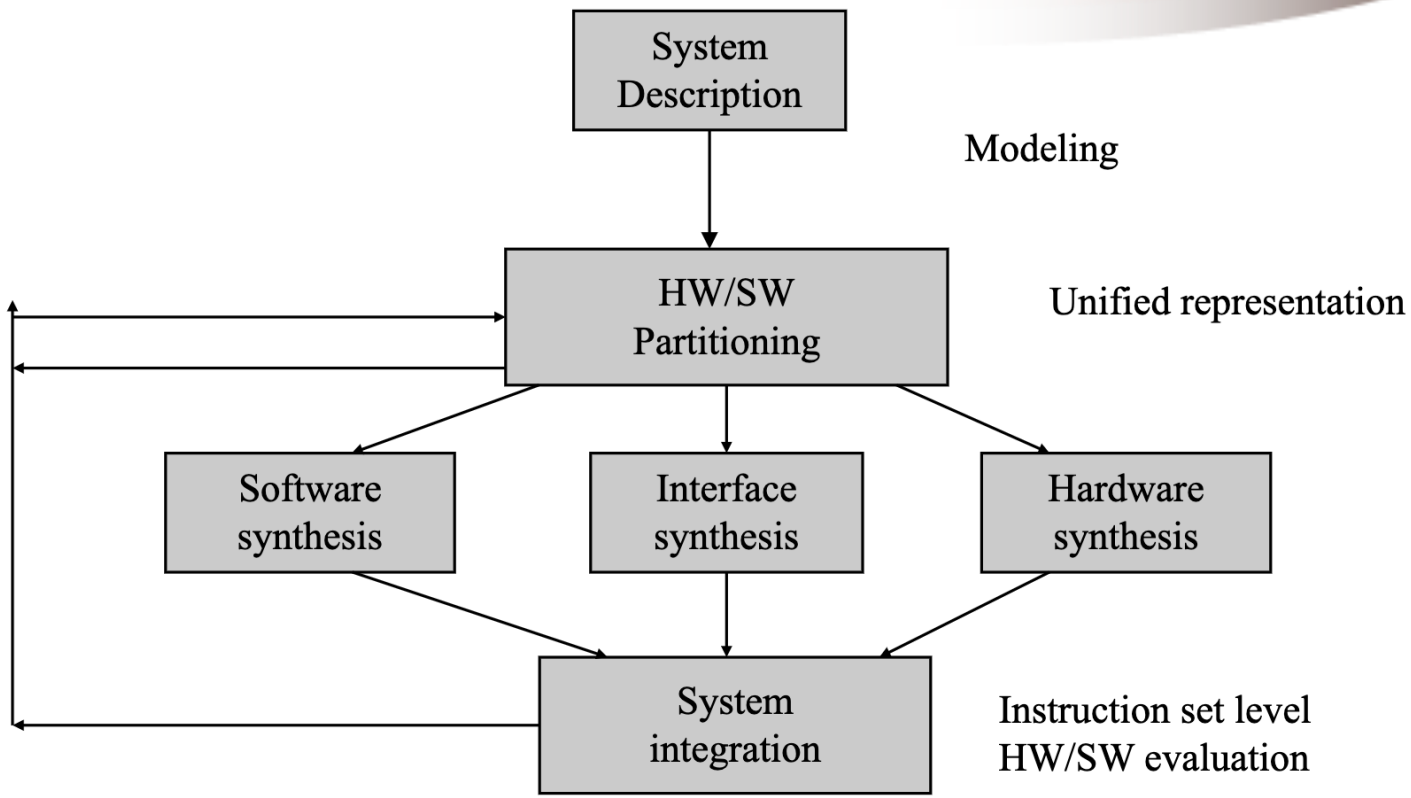
Codesign of ISA

- ISA is fundamental to digital system design
- An instruction set permits concurrent design of HW and compiler developments
- Good ISA design is critical in achieving system usability across applications
- Goal of codesign in ISP development is to optimize HW utilization by application & OS

Codesign of ISA

- For high performance in ES, selection of instruction set that matches the application is very important
    - replace the standard core by ASIP
- ASIPs are more flexible than ASICs but less than ISP
- ASIP performs better than ISP

Typical codesign process



Steps in Codesign

HW-SW system involves

- **specification**
- **modeling**
- design space exploration and partitioning
- synthesis and optimization
- **validation**

Steps in Codesign

HW-SW system involves

- **specification**
- **modeling**
- design space exploration and partitioning
- synthesis and optimization
- **validation**
- **implementation**

Steps in codesign

Specification

- List the functions of a system that describe the behavior of an abstraction clearly without ambiguity.

Modeling:

- Process of conceptualizing and refining the specifications, and producing a hardware and software model.

Modeling style

- Homogeneous: a modeling language or a graphical formalism for presentation
  - partitioning problem used by the designer
- Heterogeneous: multiple presentations
  - partitioning is specified by the models

Steps in codesign

Validation:

Process of achieving a reasonable level of confidence that the system will work as designed.

- Takes different flavors per application domain: cosimulation for performance and correctness

Steps in codesign

Implementation:

Physical realization of the hardware (through synthesis) and of executable software (through compilation).

Partitioning and Scheduling (where and when)

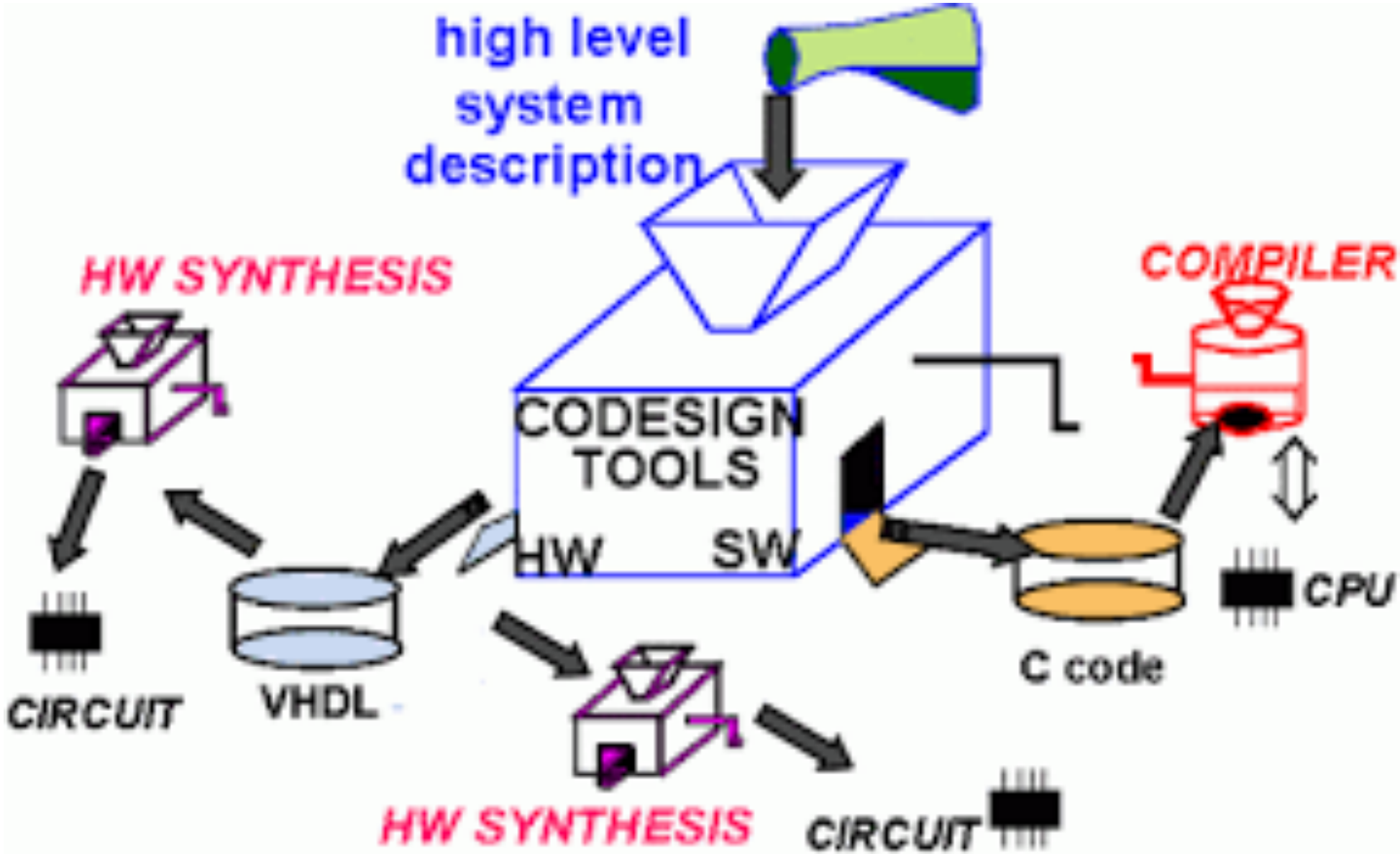
- A hardware/software *partitioning* represents a physical partition of system functionality into application-specific hardware and software.
- *Scheduling* is to assign an execution start time to each task in a set, where tasks are linked by some relations.

Summary: Research areas in codesign

- Languages
- Architectural exploration tools
- Algorithms for partitioning
- Scheduling
- SW, HW and interface Synthesis
- Verification and Testing

Hardware Software Codesign

- ***Software*** refers to the flexible part and ***Hardware*** refers to the fixed part in hardware –software co-design.
- The flexible part(software) includes C programs, configuration data, parameter settings, bitstreams, and so forth.
- We will model ***software*** as single-thread sequential programs, written in C or assembly. The choice for single-thread sequential C is simply because it matches so well to the actual execution model of a typical microprocessor.
- The fixed part(hardware) consists of programmable components such as microprocessors and coprocessors.
- We will model ***hardware*** by means of single-clock synchronous digital circuits created using word-level combinational logic and flip-flops.
- These circuits can be modeled with building blocks such as registers, adders, and multiplexers (also called register-transfer-level (RTL) modeling because the behavior of a circuit can be thought of as a sequence of transfers between registers, with logical and arithmetic operations performed on the signals during the transfers).


high level
system
description
HW SYNTHESIS
COMPILER
CODESIGN
TOOLS
HW
SW
CPU
C code
CIRCUIT
VHDL
HW SYNTHESIS
CIRCUIT

Co-design model:

Figure 1.3 shows an 8051 microcontroller and an attached coprocessor. The coprocessor is attached to the 8051 microcontroller through two 8-bit ports P0 and P1.

A C program executes on the 8051 microcontroller, and this program contains instructions to write data to these two ports. When a given, predefined value appears on port P0, the coprocessor will make a copy of the value present on port P1 into an internal register.

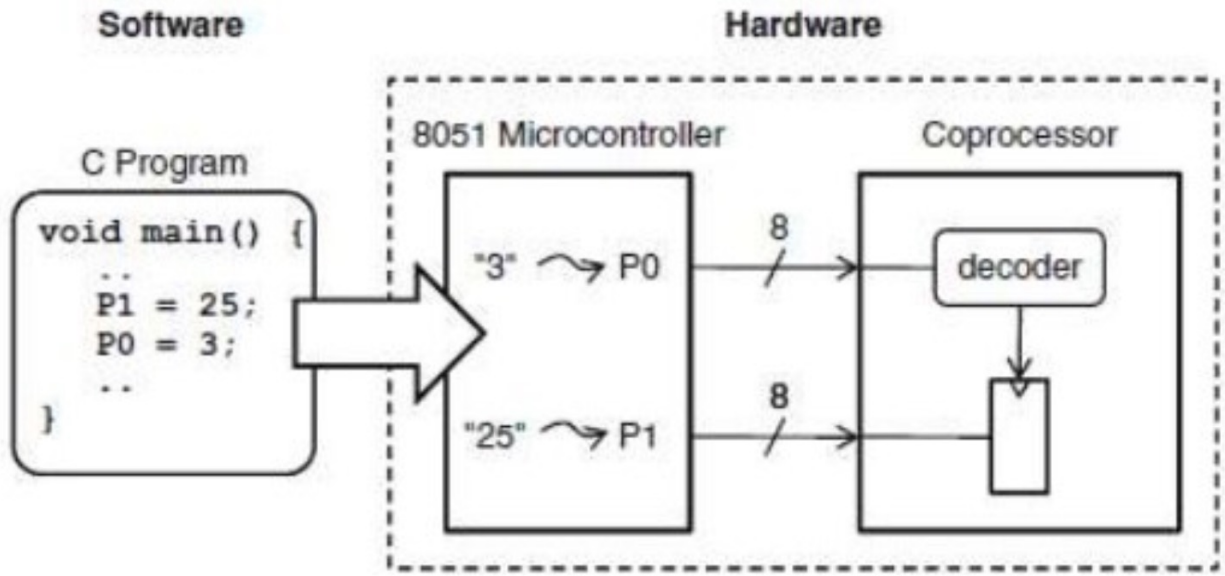


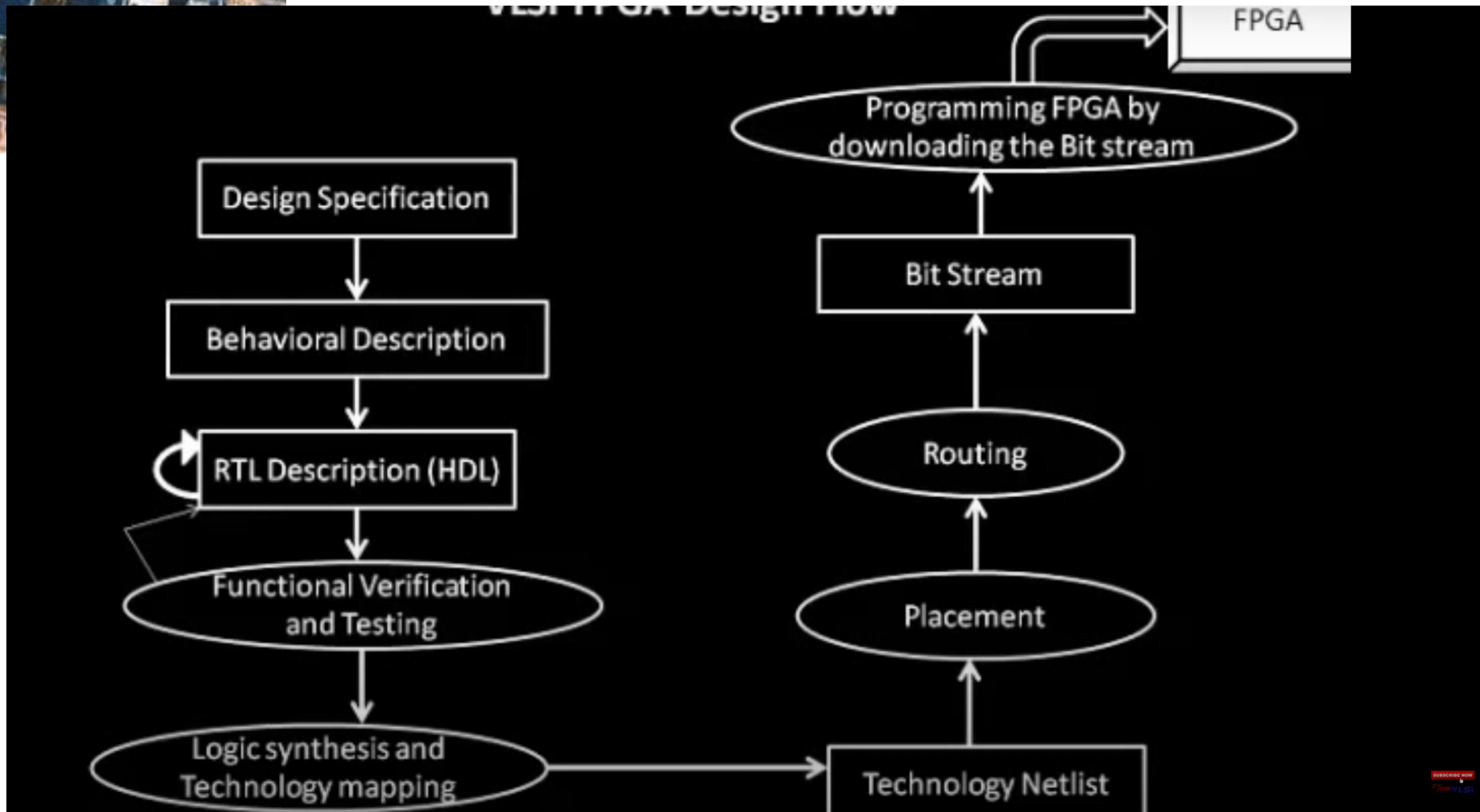
Fig. 1.3 A codesign model

This very simple design can be addressed using hardware/software codesign; it includes the design of a hardware model and the design of a C program. The model contains the 8051 processor, the coprocessor, and the connections between them. During execution, the 8051 processor will execute a software program written in C.

Definition of hardware/- software codesign: ***Hardware/Software codesign is the design of cooperating hardware components and software components in a single design effort.***

However, In reality, there are many forms of hardware and software, and the distinction between them can easily become blurred. Consider the following examples.

_A Field Programmable gate Array (FPGA) is a hardware circuit that can be reconfigured to a user-specified netlist of digital gates. The program for an FPGA is a 'bitstream', and it is used to configure the netlist topology. Writing 'software' for an FPGA really looks like hardware development – even though it is software.


FPGA
Programming FPGA by downloading the Bit stream
Design Specification
Bit Stream
Behavioral Description
Routing
RTL Description (HDL)
Functional Verification and Testing
Placement
Logic synthesis and Technology mapping
Technology Netlist

Definition of hardware/- software codesign: ***Hardware/Software codesign is the design of cooperating hardware components and software components in a single design effort.***

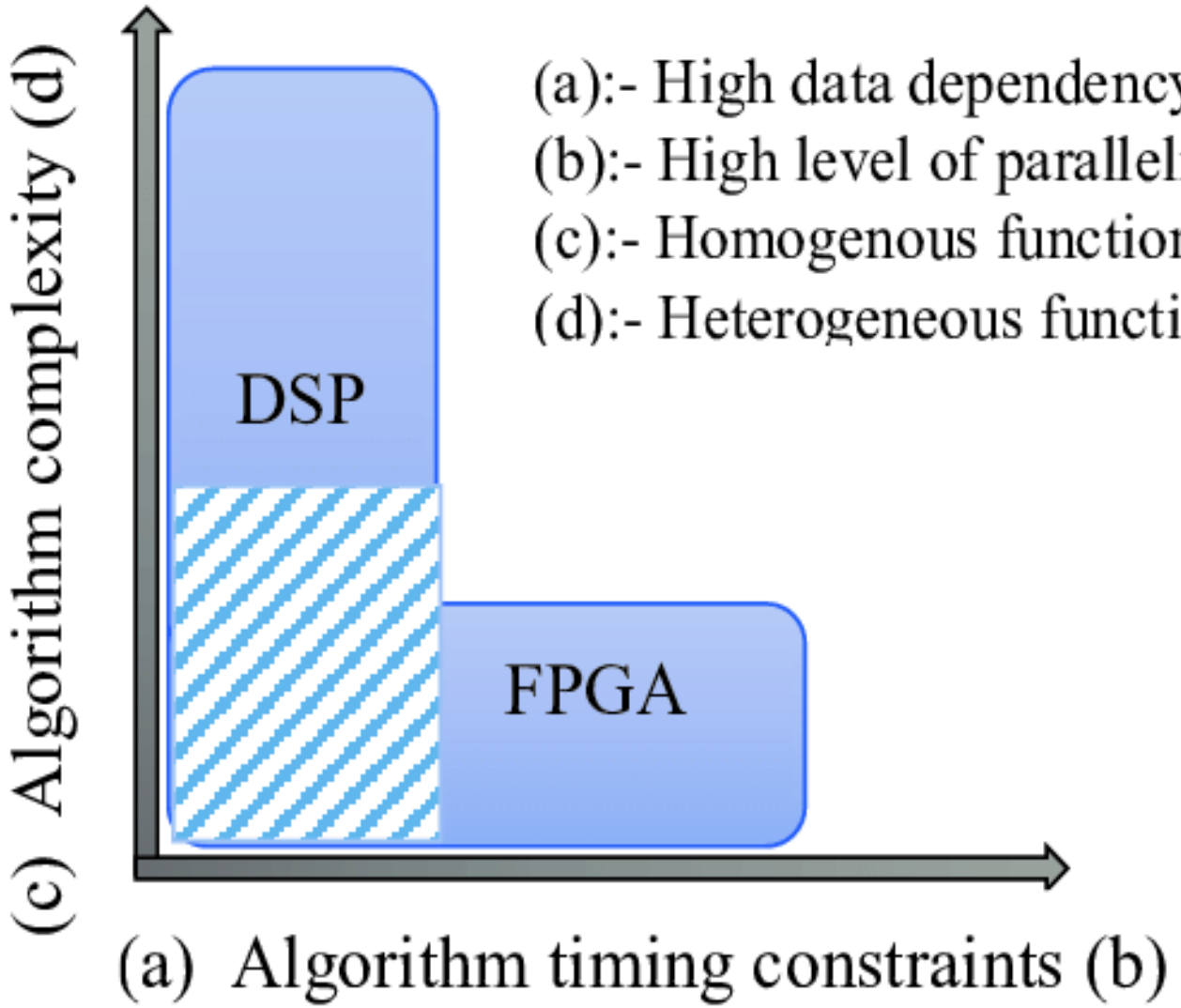
However, In reality, there are many forms of hardware and software, and the distinction between them can easily become blurred. Consider the following examples.

_A Field Programmable gate Array (FPGA) is a hardware circuit that can be reconfigured to a user-specified netlist of digital gates. The program for an FPGA is a 'bitstream', and it is used to configure the netlist topology. Writing 'software' for an FPGA really looks like hardware development – even though it is software.

_ A Digital-Signal Processor (DSP) is a processor with a specialized instruction set, optimized for signal-processing applications. Writing efficient programs for a DSP requires detailed knowledge of these specialized instructions. Very often, this means writing assembly code, or making use of a specialized software library. Hence, there is a strong connection between the efficiency of the software and the capabilities of the hardware.

Hardware/Software co-design is the partitioning and design of an application in terms of fixed and flexible components.

(a):- High data dependency
(b):- High level of parallelism of the algorithm
(c):- Homogenous functions
(d):- Heterogeneous functions



Driving Factors in Hardware-Software Co-design

_ **Performance:** The classic argument in favor of dedicated hardware design has been increased performance: more work done per clock cycle. Increased performance is obtained by reducing the flexibility of an application or by specializing the architecture that the application is mapped onto

_ **Energy Efficiency:** Almost every electronic consumer product today carries a battery (iPod, PDA, mobile phone, Bluetooth device, ..). This makes these products energy-constrained. In order to become sufficiently energy-efficient, consumer devices are implemented using a combination of embedded software and dedicated hardware components.
Thus, a well-known use of hardware–software codesign is to trade function specialization and energy-efficiency by moving (part of) the flexible software of a design into fixed hardware.

_ **Power Densities** of modern high-end processors are such that their performance can no longer be increased by making them run faster. Instead, there is a broad and fundamental shift toward parallel computer architectures. However, at this moment, there is no dominant parallel computer architecture that has shown to cover all applications.

_ **Design Complexity:** Today, it is common to integrate multiple microprocessors together with all related peripherals and hardware components on a single chip {system-on-chip (SoC)}. Modern SoC are

extremely complex. The conception of such a component is impossible without a detailed planning and design phase. Extensive simulations are required to test the design upfront, before committing to a costly implementation phase. Since software bugs are easier to address than hardware bugs, there is a tendency to increase the amount of software.

_ **Design Cost:** New chips are very expensive to design. As a result, hardware designers make chips programmable so that these chips can be reused over multiple products or product generations. The SoC is a good example of this trend.
However, 'programmability' can be found in many different forms other than embedded processors: reconfigurable systems are based on the same idea of reuse-through-reprogramming.

_ **Shrinking Design Schedules:** Each new generation of technology tends to replace the older one more quickly. In addition, each of these new technologies is exponentially more complex than the previous generation. For a design engineer, this means that each new product generation brings more work that needs to be completed in a shorter period of time. Shrinking design schedules require engineering teams to work on multiple tasks at once: hardware and software are developed concurrently. A software development team will start software development as soon as the characteristics of the hardware platform are established, even *before* an actual hardware prototype is available.

_ **Deep-Submicron Effects:** Designing new hardware from-scratch in high-end silicon processes is difficult due to second-order effects in the implementation. For example, each new generation of silicon technology has an increased variability and a decreased reliability. Programmable, flexible technologies make the hardware design process simpler, more straightforward, and easier to control. In addition, programmable technologies can be created to take the effects of variations into account.

Abstractions Commonly used by Engineers to design Hardware-Software Systems

There are five abstraction levels commonly used by computer engineers for the design of electronic hardware–software systems.

Continuous Time: At the lowest abstraction level, we describe operations as continuous actions.

- For example, electric networks can be described as systems of interacting differential equations. Solving these equations leads to an estimate for voltages and currents in these electric networks.
- This is a very detailed level, useful to analyze analog effects. However, this level of abstraction is not used to describe typical hardware–software systems.

Discrete-event: At the next abstraction level, we lump activity at discrete points in time called events. Those events can be possibly irregularly spaced.

- For example, when the inputs of a digital combinatorial circuit change, the effect of those changes will ripple from inputs to outputs, toggling nets at intermediate circuit nodes as they change.
- Discrete-event simulation is very popular to model hardware at low abstraction level. It gets rid of the differential equations and the complexity of continuous-time simulation, yet it captures all relevant information such as glitches and clock cycle edges.

- Discrete-event simulation is also used to model systems at high abstraction level, to simulate abstract event with irregular spacing in time.
- In the context of hardware–software system design however, discrete-event modeling refers to a low abstraction level.

Cycle-accurate: Single-clock synchronous hardware circuits have the important property that all interesting things happen at regularly spaced intervals, namely at the clock edge.

- A cycle-accurate model does not capture propagation delays or glitches. All activities that fall 'in between' clock edges are concentrated at the clock edge itself. As a result, activities happen either immediately (for combinatorial circuits for example), or else after an integral number of clock cycles (for sequential circuits).
- The cycle-accurate level is very important for hardware–software system modeling and very often serves as the 'golden reference' for a hardware–software implementation.

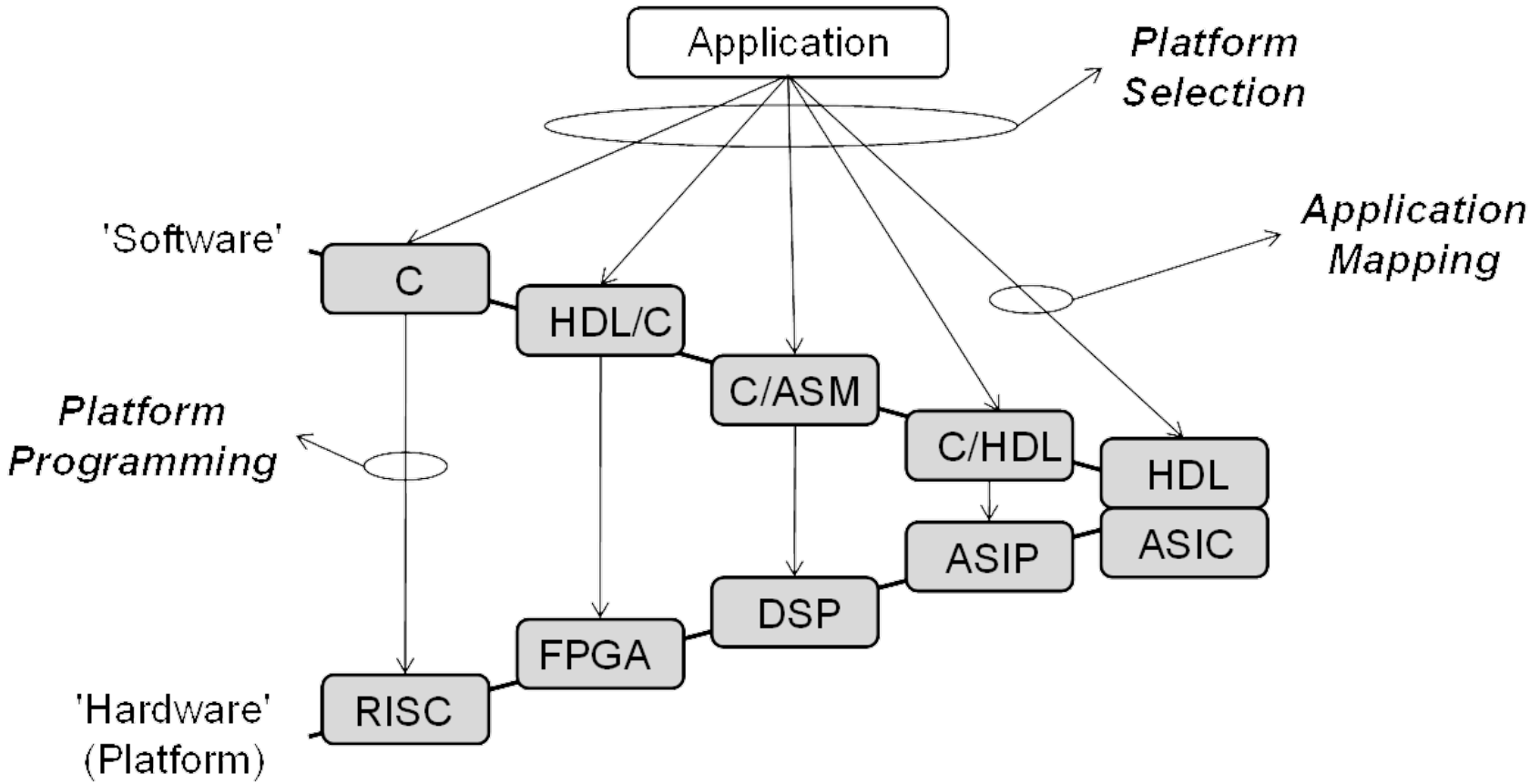
Instruction-accurate: The instruction-accurate modeling level expresses activities in steps of one microprocessor instruction. Each instruction lumps together several cycles of processing.

- RTL models are great but may be too slow for complex systems. For example, a laptop has a processor that probably clocks over 1 GHz (one billion cycles), for which lower level of abstraction is not suitable. Clearly, further abstraction can be useful to build leaner and faster models.
- Instruction-accurate simulators are used to verify complex software systems, such as complete operating systems. Instruction-accurate simulators count time in terms of an instruction count, but not a cycle count.

Transaction-accurate: In this type of model, the behavior is expressed in terms of the interactions between the components of a system. These interactions are called transactions.

- For very complex systems, even instruction-accurate models may be too slow or require too much modeling effort. For these models, yet another abstraction level is introduced: the transaction-accurate level.
- Transaction-accurate models are important in the exploratory phases of a design, where a designer is interested in defining the overall characteristics of a design without going through the effort of developing detailed models.

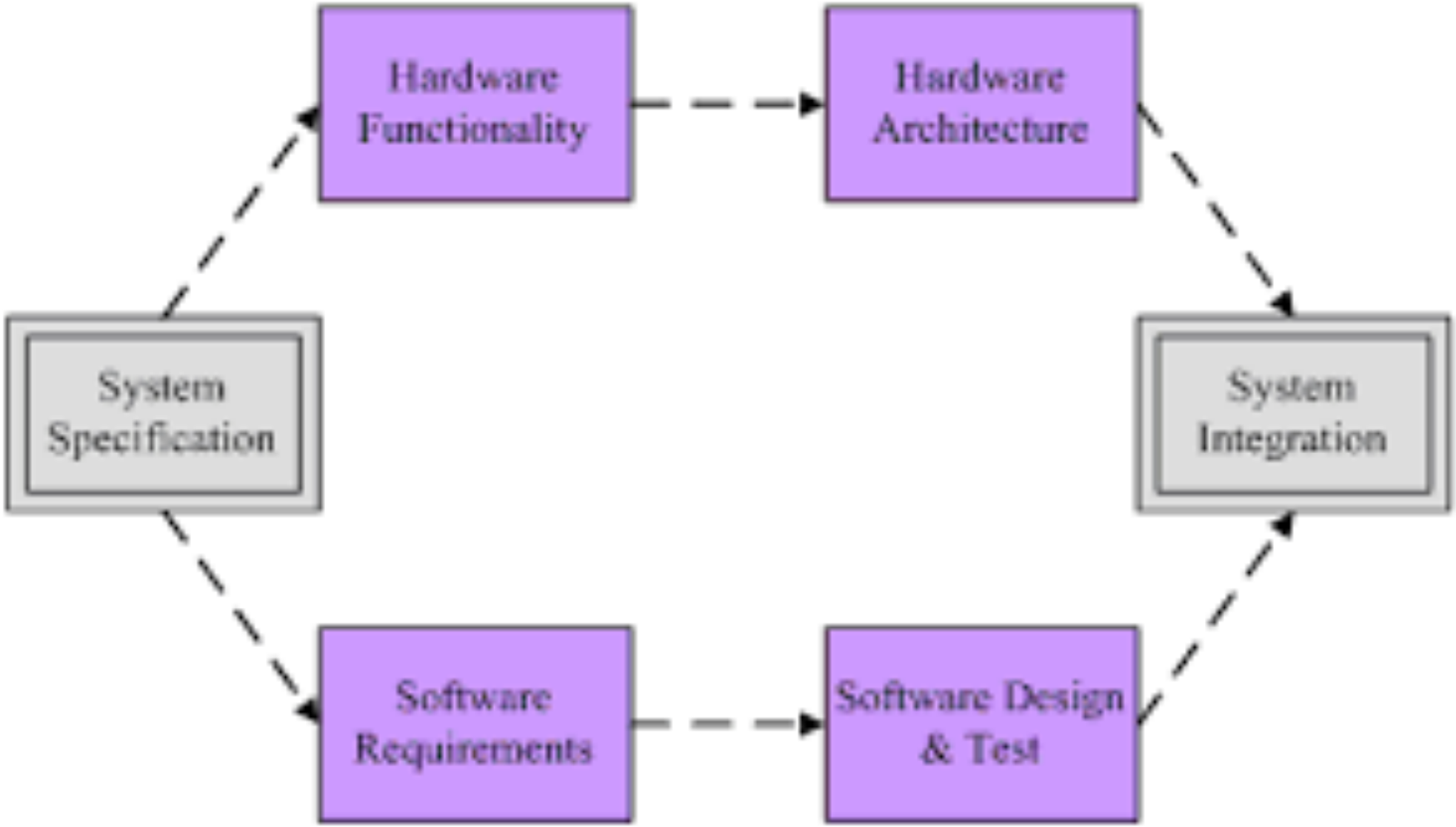
Hardware Software codesign space



Max Flexibility
Min Efficiency

Min Flexibility
Max Efficiency

General Purpose — Domain Specific — Application Specific


Hardware Functionality
Hardware Architecture
System Specification
System Integration
Software Requirements
Software Design & Test

The Platform Design Space

A specification is a description of the desired application.

A new application could be for example a novel way of encoding audio in a more economical format than current encoding methods.

The objective of the design process is to implement the application on a target platform.

Software as well as hardware have a very different meaning depending on the platform.

_ In the case of a RISC processor, software is written in C, while the hardware is a general-purpose processor.

_ In the case of a FPGA, software is written in a HDL. When the FPGA contains a soft-core processor, as discussed above, we will also write additional platform software in C.

_ A DSP uses a combination of C and assembly code for software. The hardware is a specalized processor architecture, adapted to signal processing operations.

_ An ASIP is a processor that can be specialized to a particular application domain, for example, by adding new instructions and by extending the processor datapath. The 'software' of an ASIP thus can contain C code as well as a hardware description of the processor extensions.

_ Finally, in the case of an ASIC, the application is written in HDL, which is then converted into a hardcoded netlist. In contrast to other platforms, ASICs are nonprogrammable. In an ASIC, the application and the platform have merged to a single entity.

Application Mapping

Mapping an application onto a platform means writing software for that platform, and if needed, customizing the hardware of the platform.

Flexibility means how well the platform can be adapted to different applications.

Flexibility in platforms is desired because it allows designers to make changes to the application after the platform is fabricated

Very flexible platforms, such as RISC and FPGA, are programmed with general-purpose languages. When a platform becomes more specialized, the programming tends to become more specialized as well. We visualize this by drawing the application closer to the platform. This expresses that the software becomes more specific to the hardware.

Dualism of Hardware and Software Design

Design Paradigm:

In a hardware model, circuit elements operate in parallel. Thus, by using more circuit elements, more work can be done within a single clock cycle. Software, onthe other hand, operates sequentially. By using more operations, a software program will take more time to complete. Thus, a hardware designer solves problems by *spatial decomposition*, while a software designer solves problems by *temporal decomposition*.

_ Resource Cost:

Decomposition in space, as used by a hardware designer, means that more gates are required for when a more complex design needs to be implemented.

Decomposition in time, as used by a software designer, implies that a more complex design will take more instructions to complete.

Therefore, the resource cost for hardware is circuit *area*, while the resource cost for software is execution *time*.

_ Design Constraints:

A hardware designer is constrained by the clock cycle period of a design. A software designer, on the other hand, is limited by the capabilities of the processor instruction set and the memory space available with the processor. Thus, the design constraints for hardware are in terms of a *time* budget, while the design constraints for software are fixed by the CPU.

_ Design Constraints:

A hardware designer is constrained by the clock cycle period of a design. A software designer, on the other hand, is limited by the capabilities of the processor instruction set and the memory space available with the processor. Thus, the design constraints for hardware are in terms of a *time* budget, while the design constraints for software are fixed by the CPU.

_ Flexibility:

Flexibility is the ease by which the application can be modified or adapted after the target architecture for that application is manufactured.

In software, flexibility is essentially free. In hardware on the other hand, flexibility is not trivial A hardware designer has to think carefully about such reuse: flexibility needs to be designed into the circuit by means of multiplexers and additional control signals!

Parallelism:

A dual of flexibility can be found in the ease with which parallel implementations can be created. For hardware, parallelism comes for free as part of the design paradigm.

For software, on the other hand, parallelism is a major challenge.

When a software written for a single processor is run on multiple processors, inter-processor communication and synchronization become a challenge.

_ Modeling:

In software, modeling and implementation are very close. Indeed, when a designer writes a C program, the compilation of that program for the appropriate target processor will also result in the implementation of the program!

In hardware the model and the implementation of a design are distinct concepts.

Initially, a hardware design is modeled using a HDL. Such a hardware description can be simulated, but it is not an implementation of the actual circuit. Hardware designers use a hardware *description* language, and their programs are models which are later transformed to implementation. Software designers use a software *programming* language, and their programs are an implementation by itself.

_ Reuse:

The idea of IP-reuse is that a component of a larger circuit or a program can be packaged, and later reused in the context of a different design.

In software, IP-reuse has known dramatic changes in recent years due to open source software and the proliferation of open platforms. When designing a complex program these days, designers will start from a set of standard libraries that are well-documented and available on a wide range of platforms.

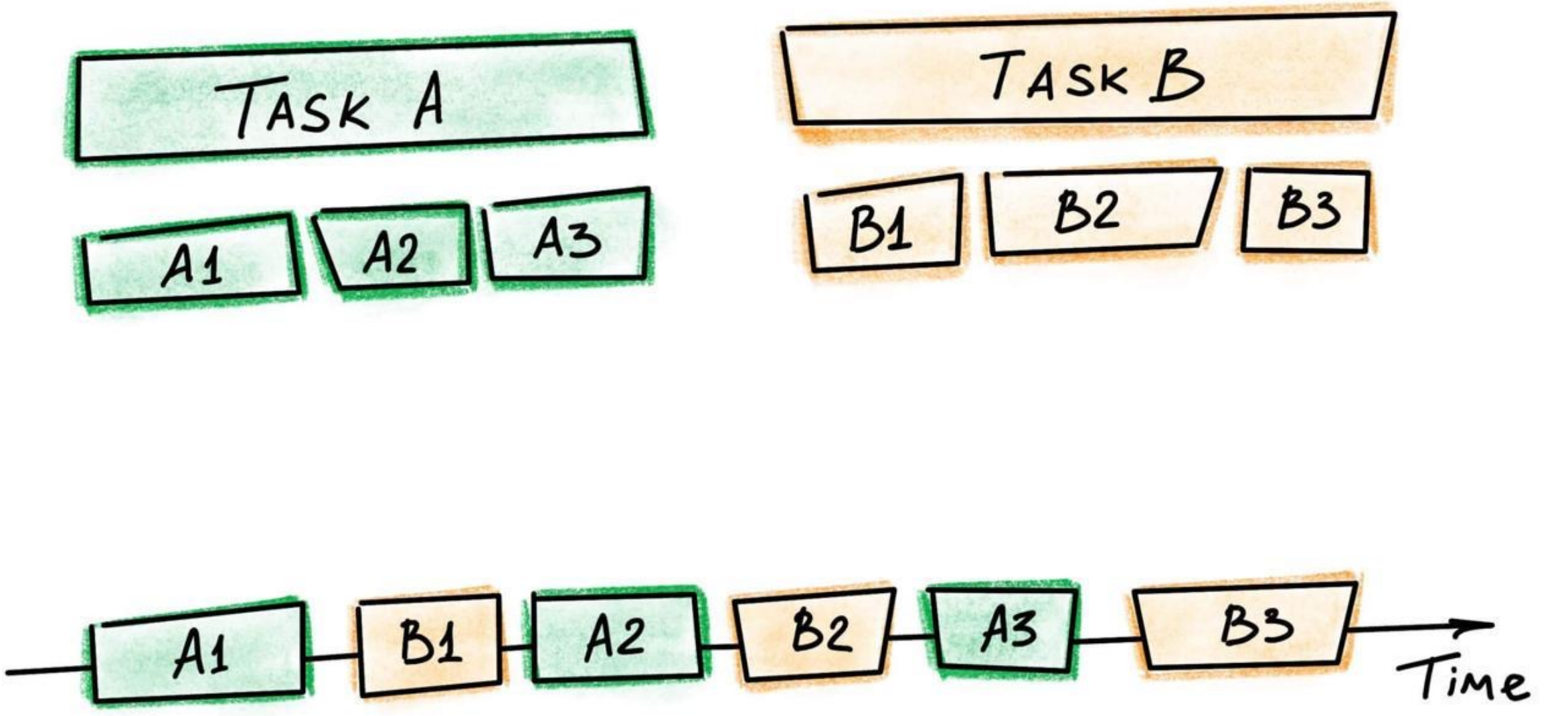
For hardware design, IP-reuse is still in its infancy. Hardware Designers are only starting to define standard exchange mechanisms.

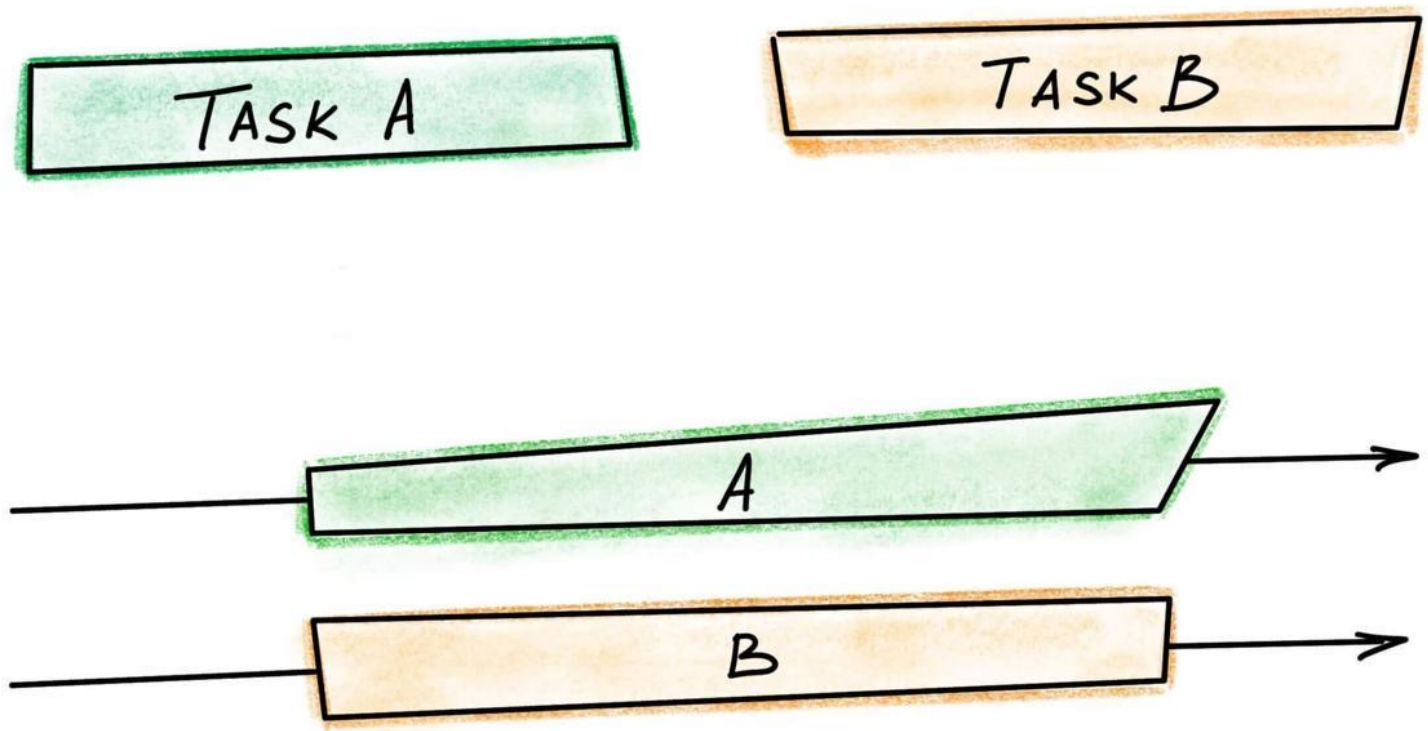
Table 1.1 The dualism of hardware and software design

	Hardware	Software
Design paradigm	Decomposition in space	Decomposition in time
Resource cost	Area (# of gates)	Time (# of instructions)
Constrained by	Time (clock cycle period)	Area (CPU instruction set)
Flexibility	Must be designed-in	Implicit
Parallelism	Implicit	Must be designed-in
Modeling	Model $\neq$ Implementation	Model $\sim$ Implementation
Reuse	Uncommon	Common

Concurrency and Parallelism

- ***Concurrency*** is the ability to execute simultaneous operations because these operations are completely independent.
- ***Parallelism*** is the ability to execute simultaneous operations because the operations can run on different processors or circuit elements. Thus, concurrency relates to an application model, while parallelism relates to the implementation of that model.
- Hardware is always parallel. Software on the other hand can be sequential, concurrent, or parallel.
- Sequential and concurrent software requires a single processor, parallel software requires multiple processors.


TASK A
A1
A2
A3
TASK B
B1
B2
B3
A1
B1
A2
B2
A3
B3
Time
@luminousmen.com



@luminousmen.com