Embedded Systems Design

1

Embedded Systems Design

Introduction

Technical context
Vocabulary
Historical evolution and trends
Embedded system general knowledge

MCSE – EnsiCaen

2

Challenges

Ubiquitous Embedded Systems

Embedded systems are present in the automotive, aeronautics, healthcare, and connected objects industries.

Key Design Requirements

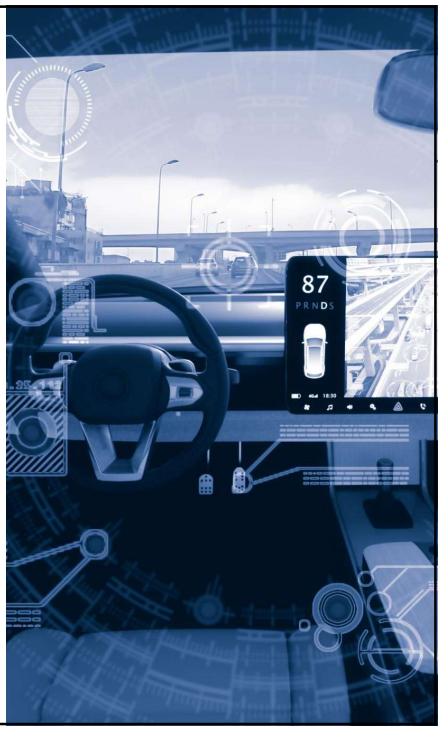
Reliability, performance, cost, and time are essential for embedded systems.

Methodological Approaches

Rigorous design using classical, model-oriented, and formal methods

Modern Trends

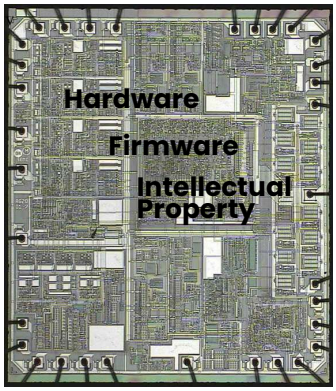
Eco-design and embedded artificial intelligence are recent developments.



3

What is the conception

HOW TO ?



Cost
Performance
Time To Market

Competencies

- **Circuit design**
- **Microprocessor system design**
- **Expertise in relevant fields (electronics, signal processing, etc.)**
- **Real-time software**
- **Design methodology**

4

Design constraints

5

Definitions 1/2

System

- An entity composed of interconnected parts arranged to fulfill one or more functions within a given environment.

Architecture

- The essential and cohesive structure of a system, expressed through its components, interfaces, relationships, behaviors, and constraints.
- Built upon a defined set of concepts.
- Represented through a collection of interrelated views (or perspectives).

Emergent Properties

- Characteristics that arise from the interactions among components, rather than from individual elements alone (greater than the sum of its parts).
- Includes properties that are observed, desired, or required.
- Encompasses constraints and properties that must be avoided.
- Strongly influenced by the environment, which may be uncertain, dynamic, or reactive

6

Definitions 2/2

Method

- A technique for solving problems, defined by a set of precise rules that ensure a correct solution.

Methodology

- A structured and consistent framework composed of models, methods, guidelines, and tools that help determine how to approach and solve a problem.

Model

- A representation of a specific, coherent aspect of reality.
- Created before making decisions or forming opinions.
- Developed to address the question that prompted its creation.
- Serves as a means to an end, not an end in itself.

7

Purpose of a model

Abstraction

- Remove unnecessary details and focus on a specific viewpoint of the system.
- Work across different levels of complexity and time scales.

Analysis

- Study the properties of the model (including property verification).
- Extrapolate findings to the real system being represented.

Communication

- Facilitate discussions and exchanges with other people.
- Enable interactions between tools.

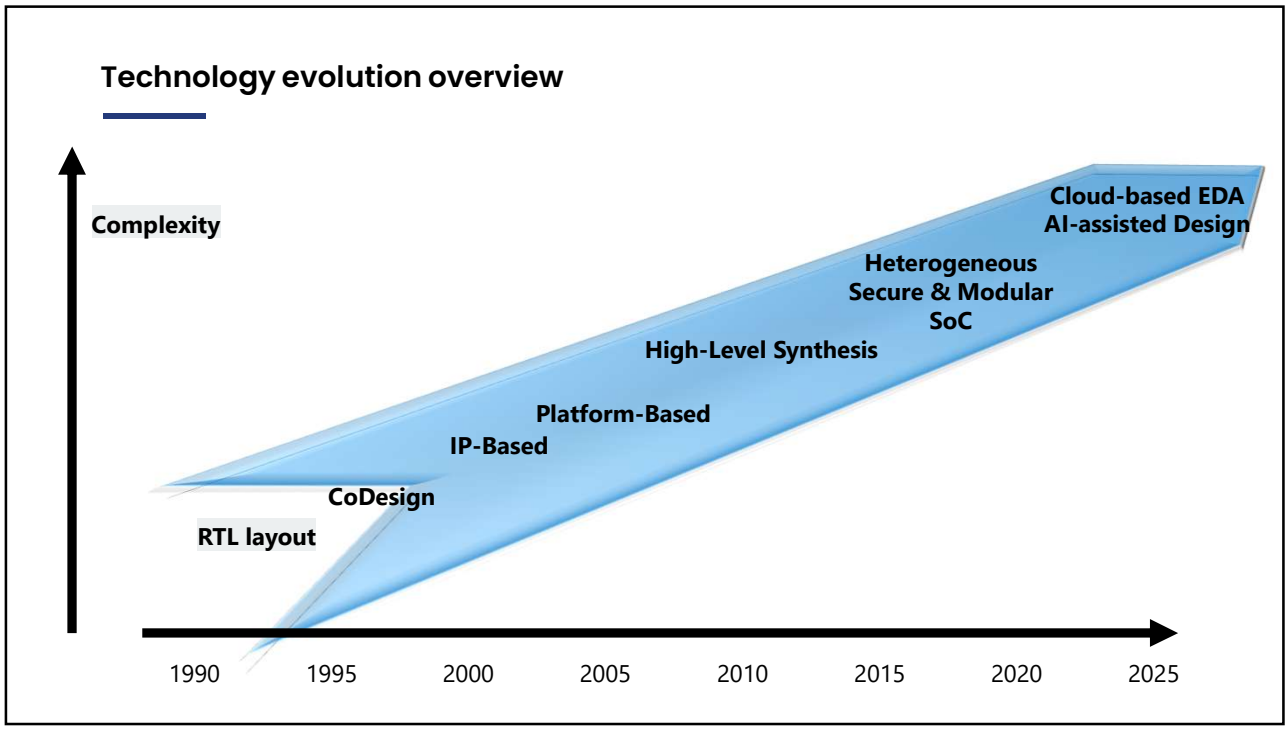
Generation / Production

- Create a representation at another level (a different model).
- Produce the actual system.

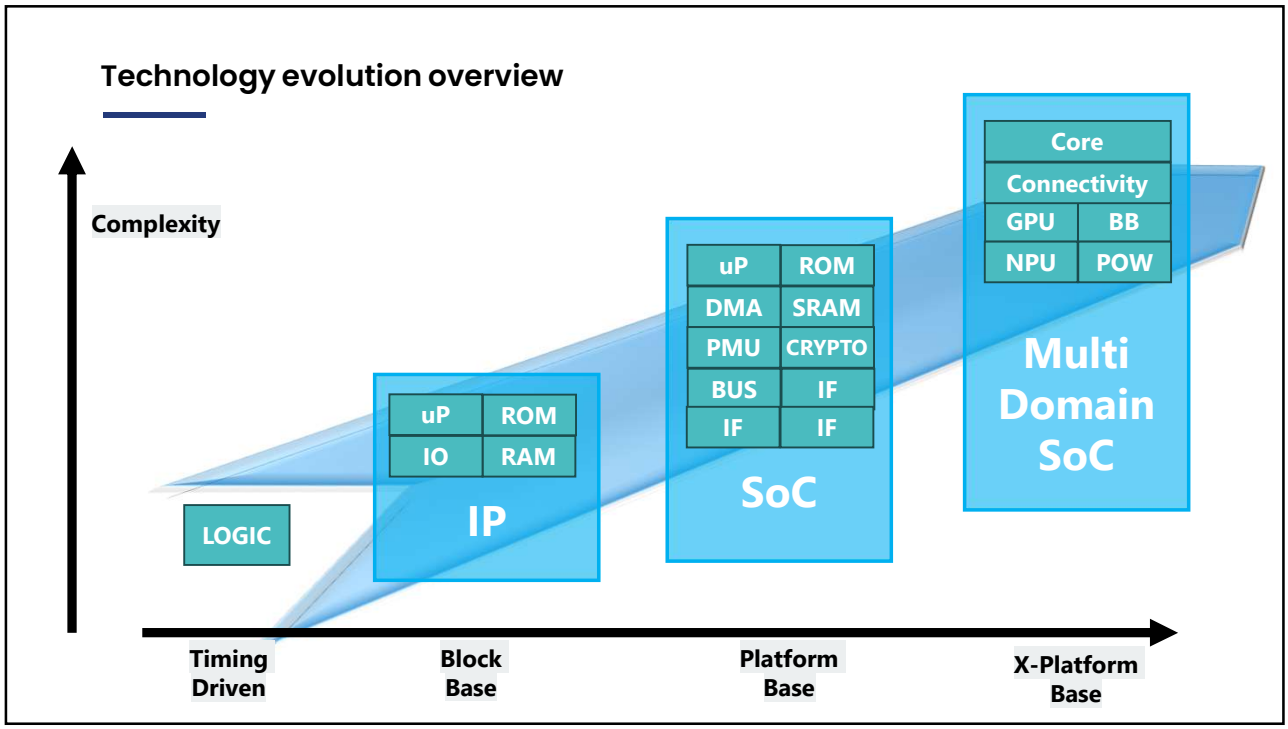
Key point :

- The model should always serve the intended objective.

8


Technology evolution overview
Complexity
Cloud-based EDA
AI-assisted Design
Heterogeneous
Secure & Modular
SoC
High-Level Synthesis
Platform-Based
IP-Based
CoDesign
RTL layout
1990
1995
2000
2005
2010
2015
2020
2025

9


Technology evolution overview
Complexity
LOGIC
uP
ROM
IO
RAM
IP
uP
ROM
DMA
SRAM
PMU
CRYPTO
BUS
IF
IF
IF
SoC
Core
Connectivity
GPU
BB
NPU
POW
Multi
Domain
SoC
Timing
Driven
Block
Base
Platform
Base
X-Platform
Base

10

Embedded Systems Design

The fundamentals of modeling

Understanding the core principles
of modeling and design

ENSI CAEN

11

Open System Interconnection reference model

- ISO : IS 7498
- CCITT : X200 (IUT-T)
- AFNOR : NF.Z.70.001

INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.200

(07/94)

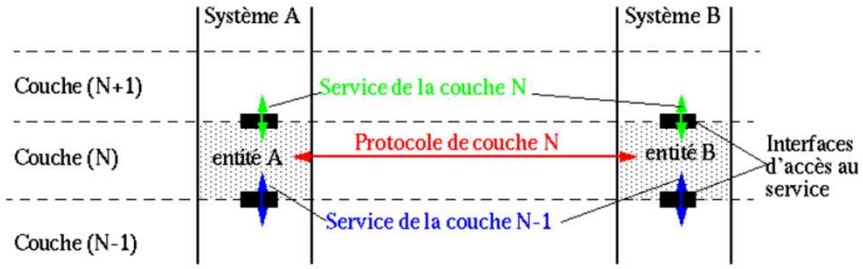
DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

OPEN SYSTEMS INTERCONNECTION – MODEL AND NOTATION

12

Notion de couche, de protocole et de service

- **Couche**: Une couche est spécialisée dans un ensemble de fonctions particulières. Elle utilise les fonctionnalités de la couche inférieure et propose ses fonctionnalités à la couche supérieure.
- **Système**: Un système est un ensemble de composants formant un tout autonome.
- **Entité**: Une entité est l'élément actif d'une couche dans un système.
- Entités **homologues** (paires): deux même couches situées dans des systèmes distants
- Le **protocole** d'une couche N définit l'ensemble des règles ainsi que les formats et la signification des objets échangés, qui régissent la communication entre les entités de la couche N.
- Le **service** d'une couche N définit l'ensemble des fonctionnalités possédées par la couche N et fournies aux entités de la couche N+1 à l'interface N/N+1.

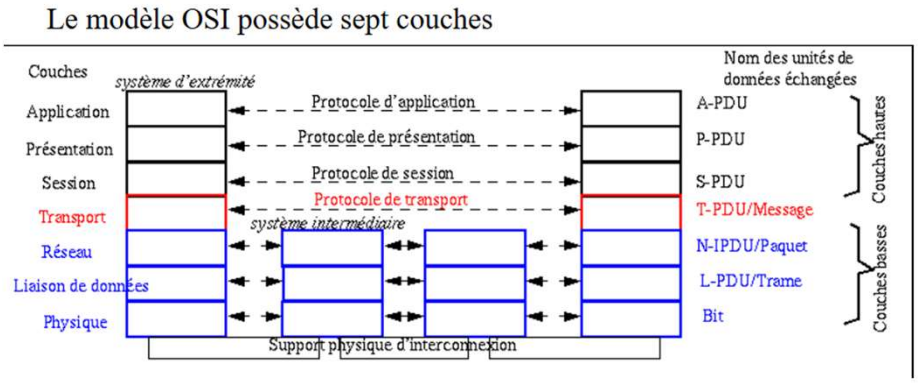


Architecture générale du modèle OSI

13

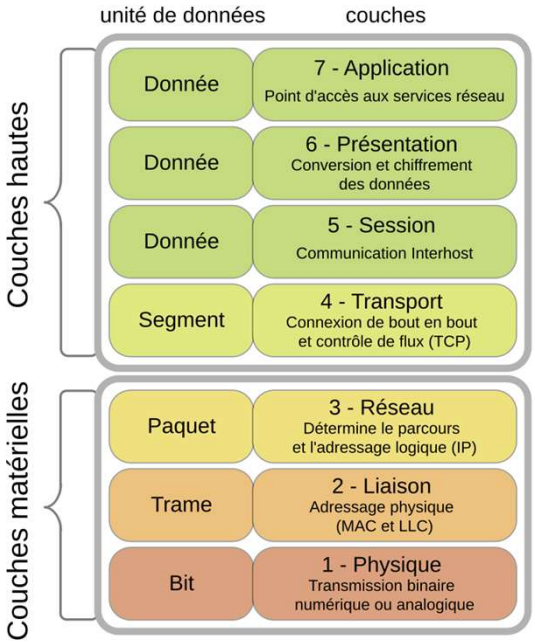
Architecture générale du modèle OSI

- L'architecture d'un réseau est définie par l'ensemble des couches et la description des protocoles et des services de chacune d'elles



14

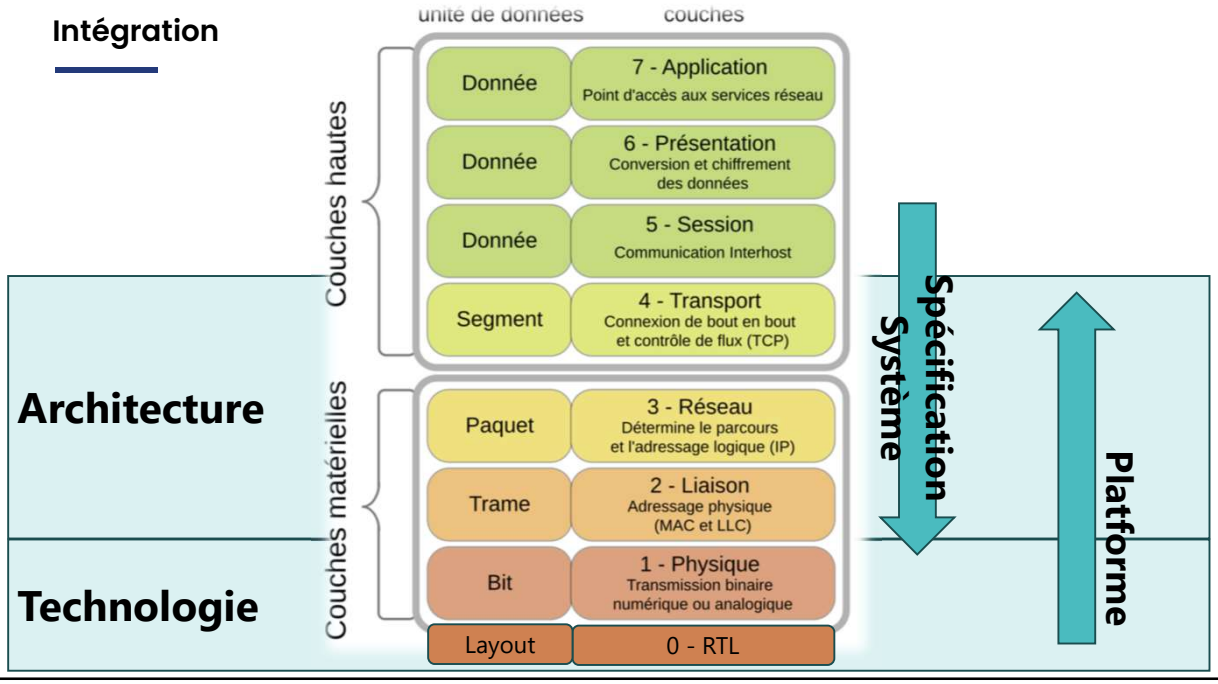
Les 7 couches



Attention : La norme stipule clairement qu'il s'agit d'un modèle de référence et par conséquent, suivant le contexte dans lequel on se trouve et les besoins de communication, certaines fonctionnalités de certaines couches peuvent ne pas être utilisées (protocoles alternatifs, classes de protocole, options, etc.).

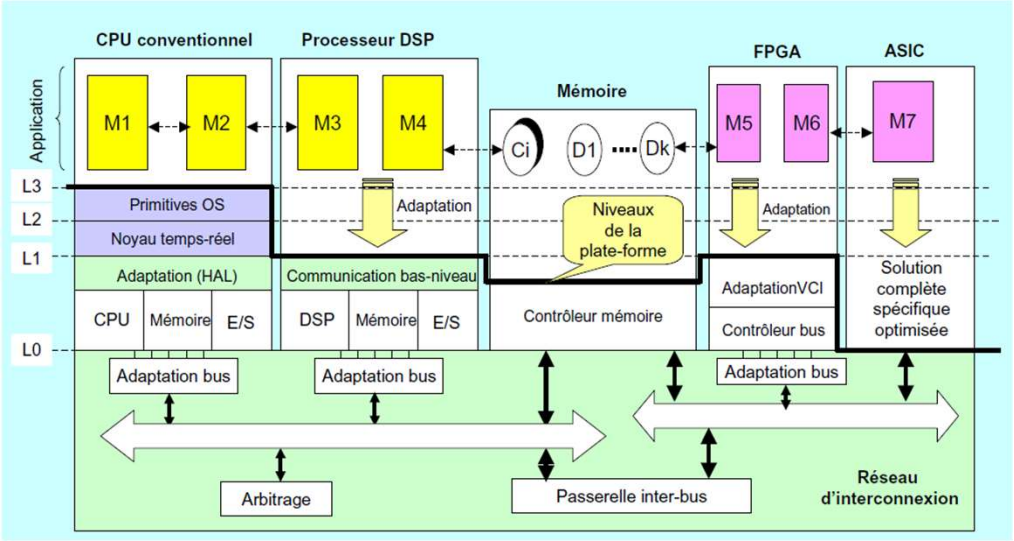
15

Intégration



16

Exemple d'une architecture hétérogène



4 Avril 2003 - Jean-Paul CALVEZ

17

Embedded Systems Design

Design flow

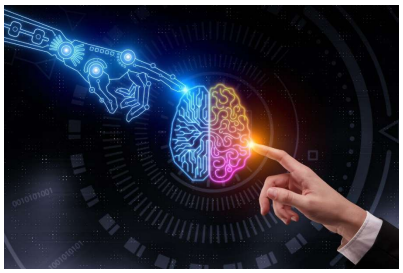
Hardware and software design flow

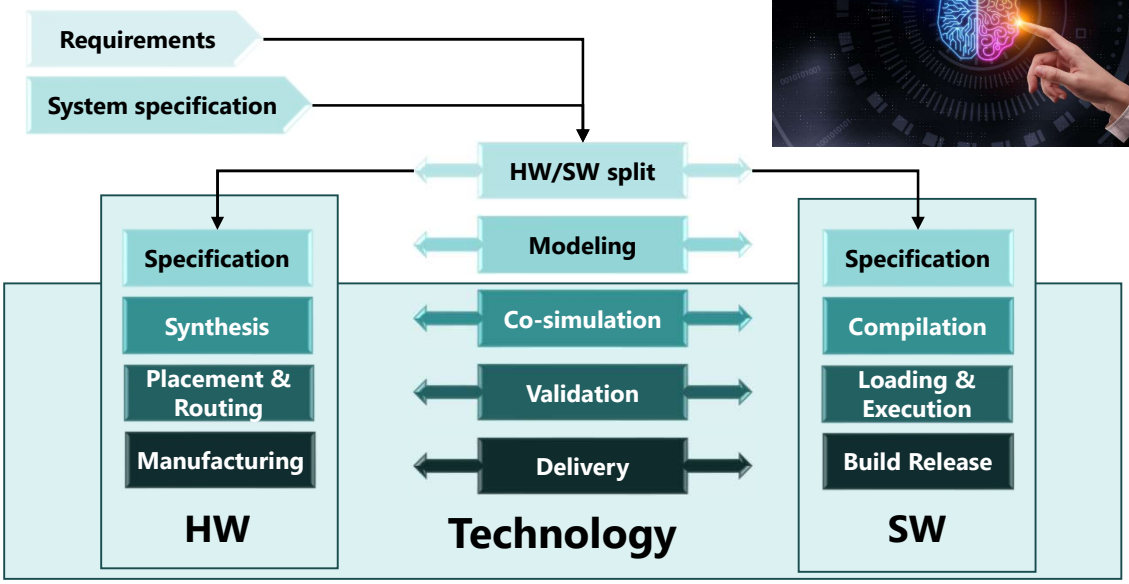
Technology background

ENSI CAEN

18

Design flow



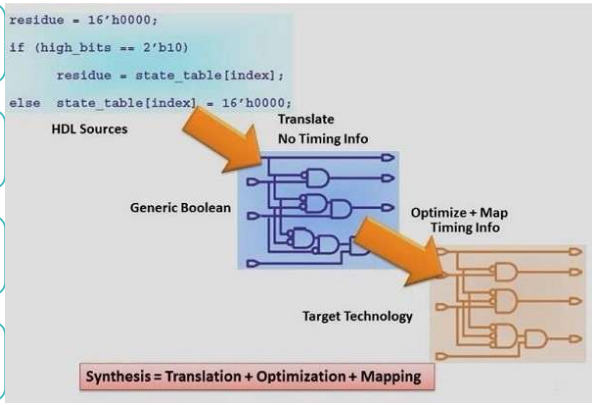


19

HW synthesis

Transform an RTL (Register Transfer Level) description into an optimized hardware implementation.

- **RTL Input**
  - VHDL / Verilog description
  - Functional correctness verified by simulation
- **Logic Optimization**
  - Boolean simplification
  - Resource sharing
- **Technology Mapping**
  - Map logic to standard cells or FPGA LUTs
- **Output**
  - Gate-level netlist
  - Ready for Place & Route



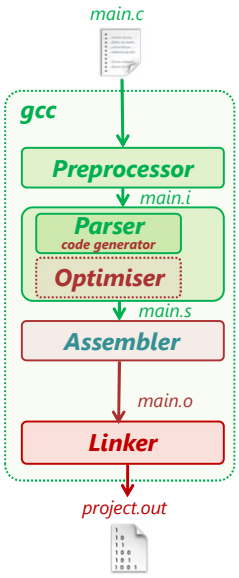
20

SW compilation

Software compilation is the process of translating source code into executable machine code.

```
if (sensorValue2 > 200) {
 if (bsensor2On == 0) {
   midiCommand(0x90, 0x42, 0x0F );
   bsensor2On = 1;
   bsensor2Off = 0;
 }
```

```
AF 31 16 3C D5 FF 0A 85 23 1D 56 EF AA 3E 89
3E 44 5A 57 69 6B 69 70 65 64 69 61 2C 20 74
65 20 66 72 65 65 20 65 6E 63 79 63 6C 6F 70
64 69 61 20 65 64 69 74 57 69 6B 69 70 65 64
61 2C 20 74 68 65 20 66 72 65 65 20 65 6E 63
63 6C 6F 70 65 64 69 61 20 74 68 61 74 20 61
79 6F 6E 65 20 63 61 6E 20 65 64 69 74 3C D6
```



21

HW placement and routing

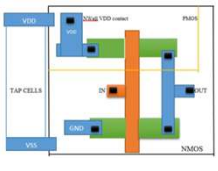
The process of physically implementing a synthesized design on a chip by placing logic elements and routing interconnections.

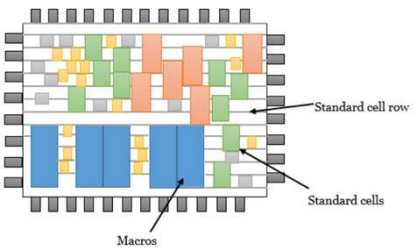
Objectives
- Meet timing constraints (setup/hold).
- Minimize chip area and power consumption.
- Ensure manufacturability and reliability.

Challenges
High design complexity.
Congestion and routing resource limitations.
Trade-offs between performance, area, and power.

Placement
- Assign physical locations to logic cells (standard cells or FPGA LUTs).
- Optimize for timing, area, and congestion.
- Consider floorplanning constraints.

Routing
- Establish physical connections between placed cells.
- Use metal layers and routing resources efficiently.
- Minimize delay and avoid signal integrity issues.



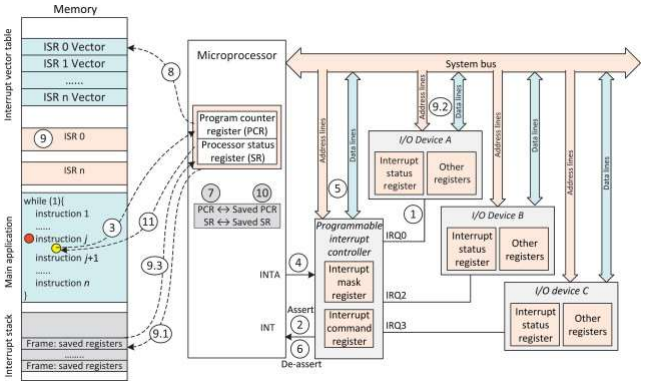


22

SW mapping, load and execution

Software load and execution is the process of loading a program into memory, setting up its execution context, and running it on the processor.

- **Loading**
  - The executable program is read from storage into main memory (RAM).
- **Vector Table:**
  - Set up interrupt and exception vectors for the processor.
- **Execution Context**
  - Prepare CPU registers and environment for program execution.
- **Boot Constraints**
  - Apply system-specific initialization rules and restrictions.
- **Execution**
  - CPU fetches, decodes, and executes instructions from memory.



23

Embedded Systems Design

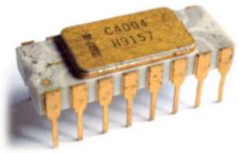
Conception evoluation

Historical chronology and evolution of methodologies for conception

ENSI CAEN

24

From 70's to 90's

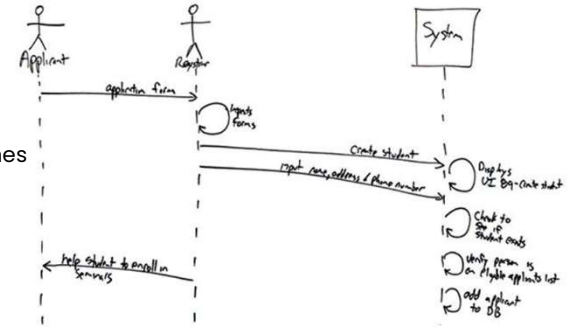


Still used

- **SADT/IDEF0**: still used for functional analysis in some industrial sectors (aerospace, defense) for documentation and communication.
- **SA/RT** (Structured Analysis for Real Time): used in educational contexts or for simple real-time systems.
- **JSD** (Jackson): very rare but sometimes present in legacy environments.
- **OOD** (Booch): its concepts have been integrated into UML and modern object-oriented approaches.

No more used

- **Structured Design** (Yourdon) and Structured Analysis (DeMarco) have been replaced by UML, SysML, and MBSE.
- **SREM, SysREM, DARTS, and SDWMC** have virtually disappeared, replaced by model-oriented approaches and integrated tools.

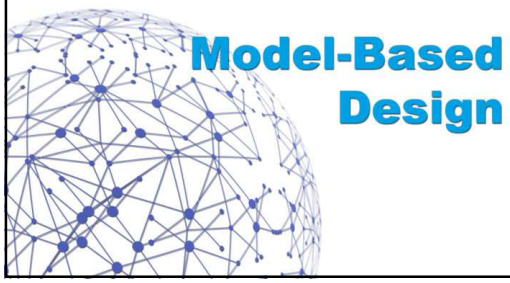


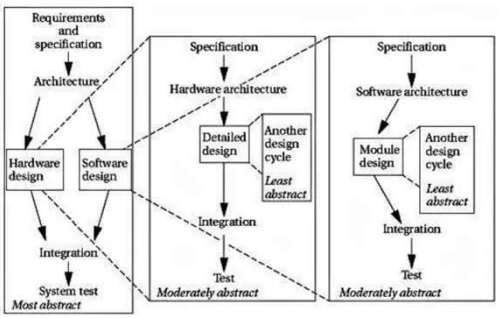
25

Modern conception methologies

Modern approaches focus on modeling, simulation, and integration:

- **UML** (Unified Modeling Language - 1997): the standard for software modeling.
- **Model-Based Design** (Simulink, SCADE - 2000): for the design and validation of real-time systems.
- **AUTOSAR** (2003): the standard for the automotive industry.
- **SysML** (Systems Modeling Language - 2006): a UML extension for embedded systems.
- **AADL** (Architecture Analysis & Design Language - 2008): for critical systems.
- **Digital Twin** (2018): for simulation and predictive maintenance.





26

Methodologies and Legacy

Sequential V-Cycle

The V-cycle ensures strict validation at each stage with good traceability of requirements.

MBSE (Model-Based Systems Engineering - 2010)

The dominant approach for complex systems (aerospace, automotive, rail).

Concurrent Engineering

This methodology promotes the parallel development of hardware and software to reduce lead times.

Hardware/Software Co-Design

Co-design integrates both aspects from the specification stage to optimize performance and resources.

27

Model-oriented methodologies and tools

Multi-Abstraction Modeling

Model-Based Systems Engineering facilitates modeling at different levels of abstraction for better understanding.

Key Modeling Tools

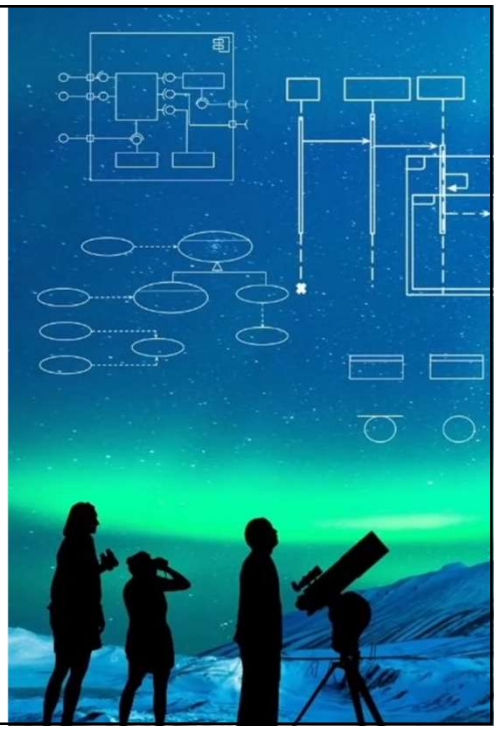
UML and SysML are essential for modeling, Architecture Analysis & Design Language for functionality, and SystemC for simulation.

Transformation Chain

Transform the Architecture Analysis & Design Language model into SystemC, then virtual prototype for dynamic validation.

Validation and Control

MATLAB/Simulink offers code generation and early validation for critical systems.



28

Recent trends

Sustainable Eco-Design

Eco-design reduces environmental impact by optimizing the life cycle of electronic products.

Embedded Artificial Intelligence

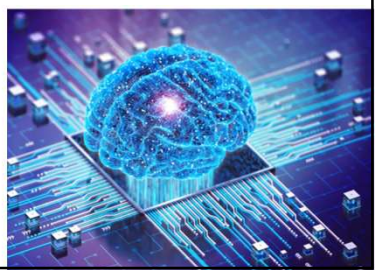
Embedded AI makes systems more autonomous while respecting energy consumption and performance.

Local Edge Computing

Edge computing processes data close to sensors, improving responsiveness and reducing central workload.

Agile and DevOps Practices

Agile and DevOps methods improve collaboration and accelerate feature delivery.



29

Methodologies and Legacy

Engineering process

V-cycle vs MBSE

ENSI CAEN

30

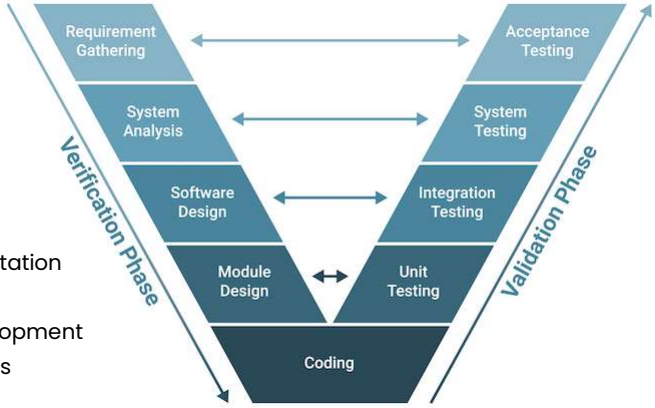
V-Model

Pro

- Improves Quality: development process
- Reduces Risks: clear roadmap
- Increases Efficiency: collaboration
- Communication: clear the requirements
- Enhances Testing: at all level
- Improves Documentation: at every stage

Cons

- Rigid: changing requirements
- Time-Consuming: planning and documentation
- Resource Intensive: no small teams
- Limited Agility: not flexibility, iterative development
- Overemphasis on Testing: delays and costs



31

MBSE: Model-Based Systems Engineering

Overview

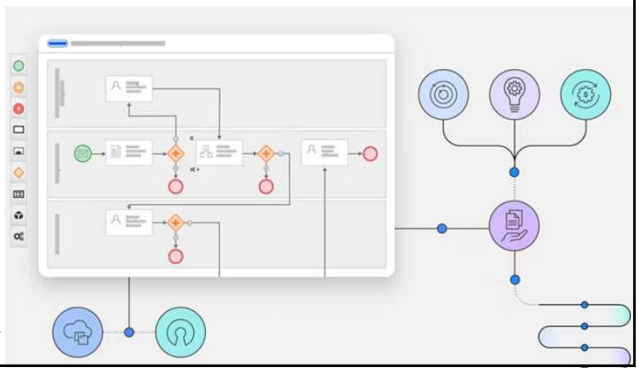
- Numerical modeling and simulation to design systems.
- Visual and interactive way to represent system components and their connections.
- This approach is particularly useful for complex systems and interfaces.

3 pillars

- The Systems Architecture Model (SAM), the single source of truth for the project.
- The engineering simulation software determines whether the SAM meets the defined requirements and functions as expected
- A centralized computing center, either cloud-based or physical, executes all the functions and stores the results.

Process

- Iterative and integrated cycle.
- The models cover requirements, architecture, behavior, and interfaces.
- Simulation and validation from the design stage.



32

Methodologies and Legacy

Concurrent Engineering

Parallel development of hardware and software

ENSI CAEN

33

Concurrent Engineering

Organization

- Hardware and Software design activities are occurring at the same time
- Increases productivity and product quality
- Errors and redesigns can be discovered early in the design process when the project is still flexible.
- Avoid costly errors during manufacturing of hardware or after

Cross-functional teams (CFT)

- Includes people from different area of the workplace that are all involved in a particular process, including manufacturing, hardware and software design, marketing, and so forth
- Designing various subsystems simultaneously
- Incremental information sharing, as soon as new information becomes available, it is shared and integrated into the design
- Integrated project management, someone is responsible for the entire project, and that responsibility is not handed-off once one aspect of the work is done.

34

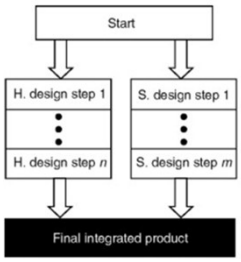
Hardware/Software Co-Design

Main goal:

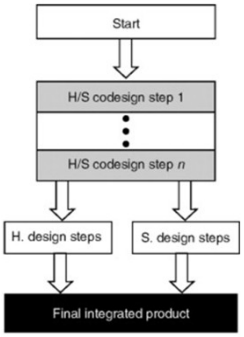
- Design of system-on-chip (SoC) or embedded cores
- Integration: General-purpose microprocessors, DSP structures, FPGA, ASIC, Memory blocks, peripherals, and interconnection buses

Traditional approach (a):

- Clear separation between hardware and software
- Independent design processes, except for compatibility standards



(A) Traditional Design Flow



(B) Hardware–Software Codesign

Modern approach (b):

- Considers the **entire system from the early design stages**
- Aims to **optimally partition tasks** between hardware and software
- Helps reduce design complexity and **optimize power consumption**

35

Methodologies and Legacy

A Generic Codesign Methodology

Overview of a general approach for designing heterogeneous hardware-software systems

An iterative approach to designing efficient hardware-software systems

ENSI CAEN

36

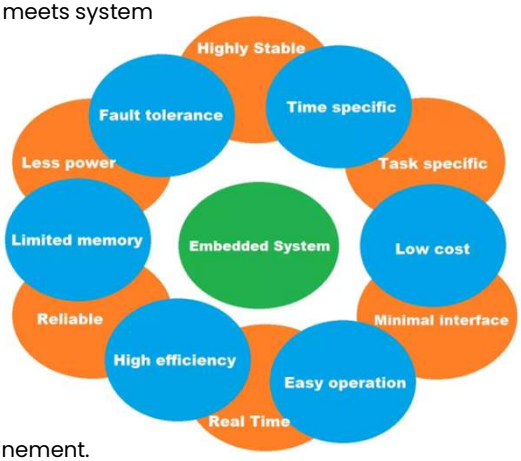
Introduction

Main goal

- Deliver an optimal hardware/software architecture that meets system specifications

Design constraints to consider

- Real-time requirements
- Performance
- Area (chip size)
- Code size
- Memory usage
- Power consumption
- Programmability



Generic methodology:

- Not tied to any specific tool or framework.

Iterative nature:

- Each step can influence others, requiring continuous refinement.

37

Step 1: Algorithm Development

Input

- High-level system specification.

Activity

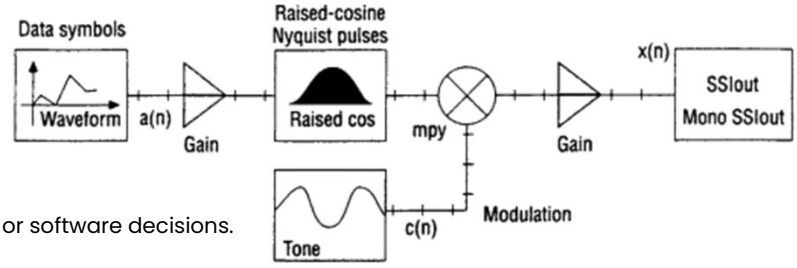
- Functional simulation at a high abstraction level, without implementation assumptions.

Goal

- Explore and validate different algorithmic approaches

Example

- Transmitter algorithm of modem design



Benefit

- Focus on core functionality before committing to hardware or software decisions.

38

Step 2: Hardware/Software Partitioning

Decision criteria

- Execution speed
- Implementation complexity
- Flexibility and reconfigurability

Software candidates

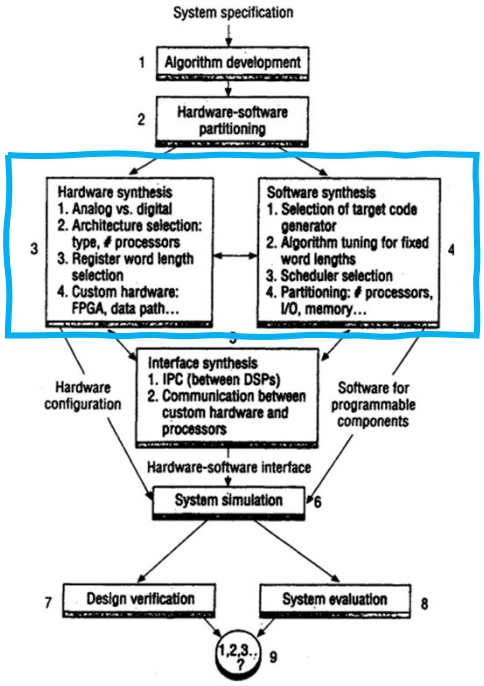
- Functions requiring programmability or standard compliance (e.g., software codec for flexible modulation schemes)

Hardware candidates

- Time-critical operations (e.g., phase detectors using CORDIC for compact VLSI)

Iterative process

- Multiple partitioning attempts may be needed to find the best trade-off.



39

Step 3: Hardware Synthesis

Key decisions:

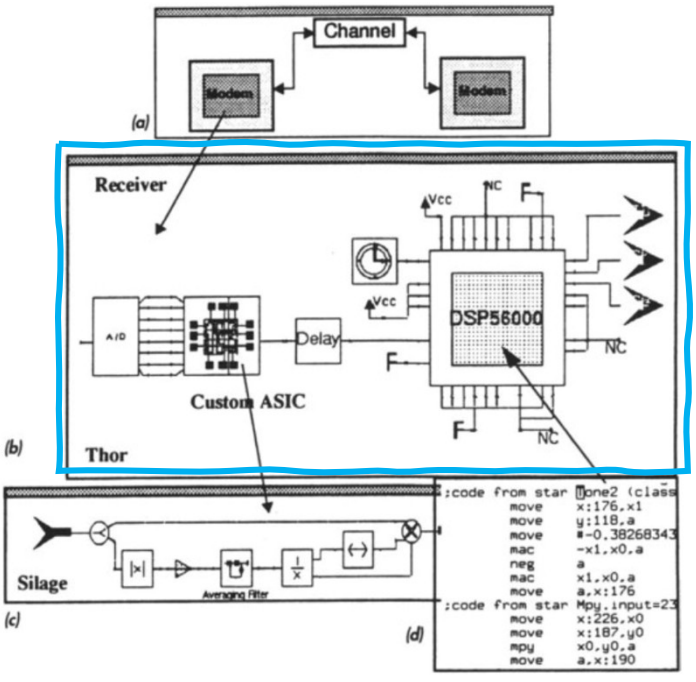
- Processor type and number
- Interconnect topology

Synthesis options:

- Custom datapath generation
- FPGA or ASIC mask generation

Optimization opportunities:

- Register word length tuning
- Area and power trade-offs
- Example: Filter structures may require fewe initially estimated



40

Step 3 bis: Digital IC design

1) Electronic System-Level Design

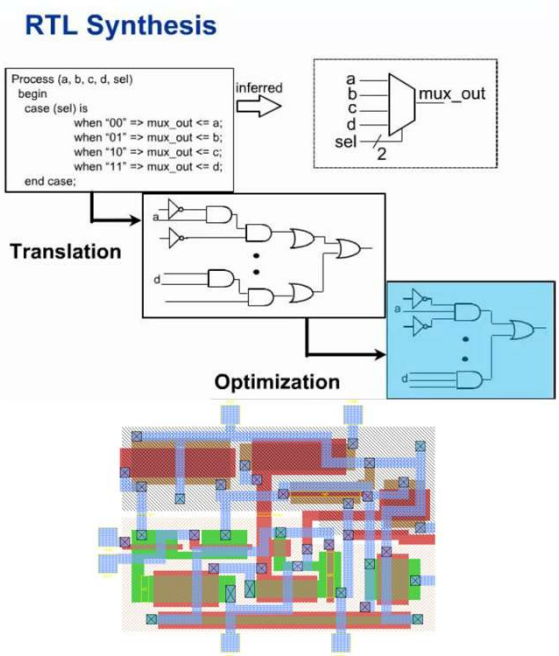
- Defines the functional specification using high-level languages and tools like C/C++, VHDL, SystemC, SystemVerilog, TLMs, Simulink, or MATLAB.

2) RTL Design

- Translates the specification into a Register Transfer Level (RTL) description, detailing digital circuit behavior and I/O connections.

3) Physical Circuit Design

- Uses RTL and a standard cell library to build the chip layout: selecting gates, placing them, and routing connections (including clock synthesis and floorplanning).



41

Step 4: Software simulation

Algorithm adaptation

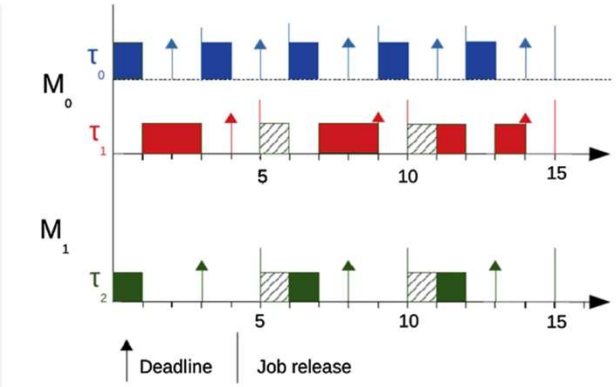
- Handle fixed-point limitations (e.g., quantization errors, limit cycles)

Tasks

- Code partitioning across processors
- Task scheduling
- Interprocessor communication code generation

Optimization goals

- Minimize communication cost
- Optimize memory bandwidth
- Balance local/global memory usage



Example. Execution of the task set with $\tau_0 = (1, 2, 3, 0)$, $\tau_1 = (2, 4, 5, 1)$, and $\tau_2 = (1, 3, 5, 1)$ allocated to a dual-core platform

42

Step 5: Interface Synthesis

Hardware side

- Add latches, FIFOs, address decoders

Software side

- Insert I/O routines and semaphore-based synchronization

Challenge

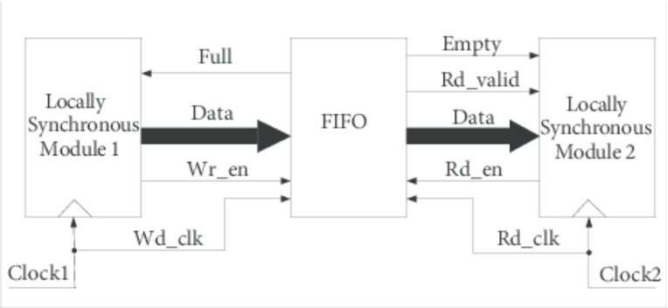
- Highly interdependent with HW and SW synthesis

Approach

- Start with a baseline design and refine iteratively

Trends

- Serialization, I3C, SPMI



Asynchronous interface based on FIFO.

43

Steps 6: Simulation, Verification, Evaluation

Heterogeneous simulation

- Simulated hardware executes the generated software
- May involve multiple simulators due to different specification languages

Verification

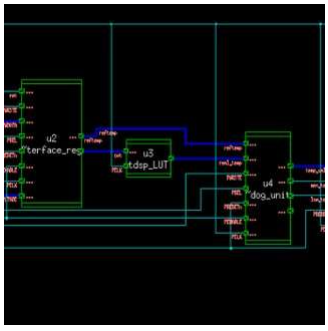
- Ensure the design meets functional and performance specs

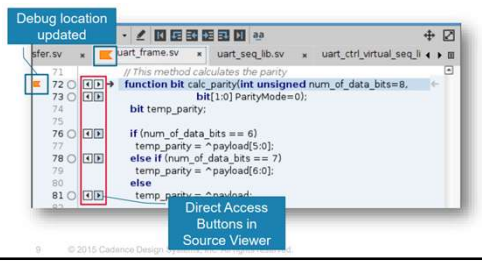
Evaluation metrics

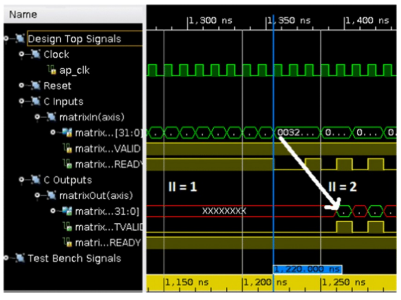
- Area, power, critical path, component/bus utilization

Outcome

- Decide whether to proceed or revise the design







44

Model-oriented methodologies and tools

Multi-Abstraction Modeling

Model-Based Systems Engineering facilitates modeling at different levels of abstraction for better understanding.

ENSI CAEN

45

Software development

Framework divides the entire software development process into four viewpoints

- **Requirements Specification**
- **Standard Technical Specification**
- **Architecture Specification**
- **Test Specification**

- **WHAT / USE CASE DRIVEN**
- **HOW / FUNCTION & MECHANISM**
- **HOW TO IMPLEMENT**
- **HOW TO TEST**

A requirement is short statement of the problem.
A specification is how to solve the problem

46

Development specification

- **Requirements Specification**
  - Context
  - Interface
  - Use Cases
  - Functional
  - Quality
  - Error Handling
  - Performance
  - Design Constraints

"WHAT"

- **Technical Specification**
  - Architecture
  - Framework
  - Communication
  - Configuration
  - Interfaces

"HOW"

- **Architecture Specification**
  - Concept
  - Logical Static View
  - Logical Dynamic View
  - Physical View: Hardware

"HOW"

47

Development specification : example of a multimedia embedded system

- **Requirements Specification**
  - Context: Automotive infotainment environment
  - Interface: Touchscreen GUI, 1280×720, gestures + buttons
  - Use Cases: Play music from USB, browse, control playback
  - Functional: Decode MP3, AAC, FLAC formats
  - Quality: Audio SNR ≥ 90 dB, no distortion
  - Error Handling: Show "Unsupported file format" message
  - Performance: Boot in ≤ 5 seconds after ignition
  - Design Constraints: Max 512 MB RAM, 1 GHz CPU
- **Technical Specification**
  - Architecture: Modular layered design
  - Framework: Embedded Linux with Yocto build system
  - Communication: CAN bus + Ethernet for data exchange
  - Configuration: XML-based configuration files
  - Interfaces: REST API for external services

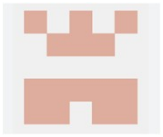
- **Architecture Specification**
  - Concept: Client-server multimedia system with modular components
  - Logical Static View: Components: UI layer, Media Engine, Network Manager, Storage Manager
  - Logical Dynamic View: Data flow: User input → Media Engine → Decoder → Output device
  - Physical View: Hardware: ARM-based SoC, 512 MB RAM, Flash storage, CAN/Ethernet interfaces

48

Key modeling tools

- **UML and SysML modeling:**
  - Tool Reference: IBM Rational Rhapsody, Enterprise Architect, MagicDraw (supports UML and SysML diagrams for system design)
- **Architecture Analysis & Design Language (AADL):**
  - Tool Reference: OSATE (Open Source AADL Tool Environment) – widely used for functional and safety-critical architecture modeling
- **SystemC for simulation:**
  - Tool Reference: Accellera SystemC Library, Synopsys Platform Architect, Cadence Virtual System Platform (for hardware/software co-simulation)
- **Transformation Chain:**
  - Tool Reference: Eclipse Modeling Framework (EMF), Papyrus, ATL (Atlas Transformation Language) – for model-to-model and model-to-code transformations






49

Unified Modeling Language (UML)

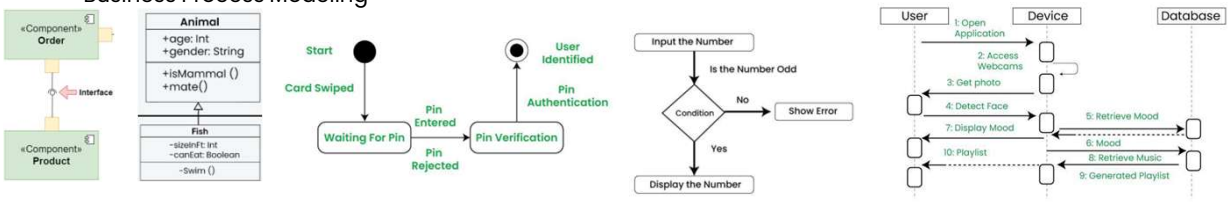
Definition

- UML is a general-purpose visual modeling language used to specify, visualize, construct, and document the artifacts of software systems.
- Purpose is to Visualize system design in engineering
- Saves time by enabling early visualization of system structure and behavior
- Standardization Published by ISO in 2005

Usage Domains

- System Architecture
- Software Engineering
- Business Process Modeling

- **Types of UML Diagrams**
- Structural diagram
  - Class diagram
  - Composite Structure Diagram
  - Object Diagram
  - Component Diagram
  - Deployment Diagram
  - Package Diagram
- Behavioral Diagrams
  - State Machine Diagrams
  - Activity Diagrams
  - Use Case Diagrams
  - Interaction Diagrams
    - Sequence Diagram
    - Communication Diagram
    - Timing Diagram



50

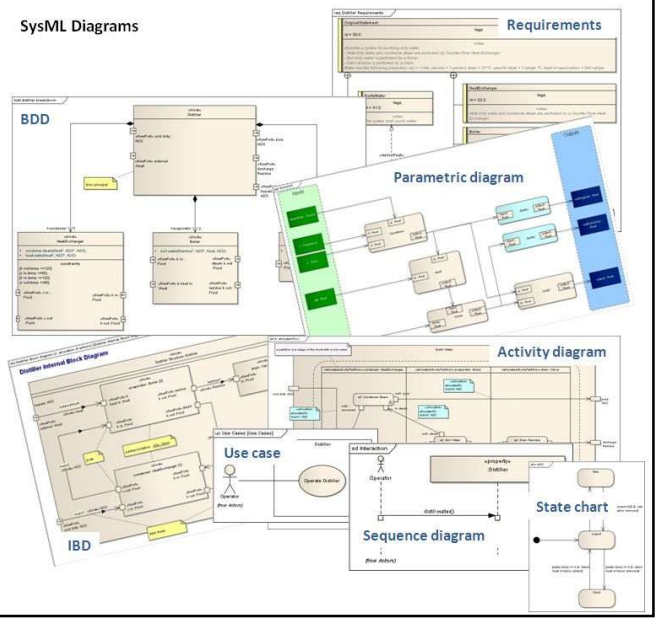
System Modeling Language (SysML)

Definition

- General-purpose modeling
- Engineering activities
- Specification, analysis, design, verification
- OpenSource license

Advantage

- Less software-centric
- Includes requirements, performances
- Small language
- Flexibles allocation tables
- IEEE-Std-1471-2000 (IEEE Recommended Practice for Architectural Description of Software Intensive Systems).



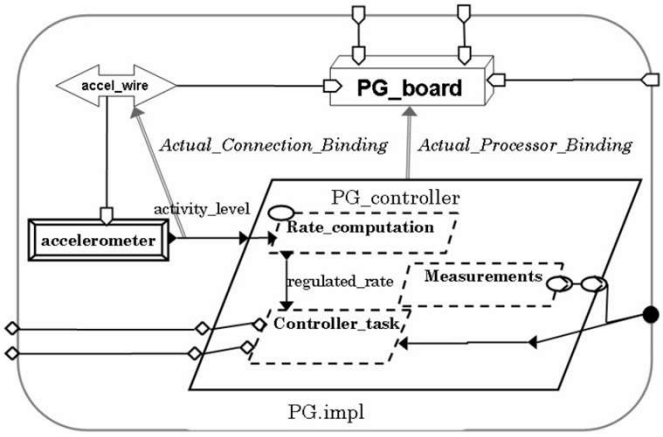
51

Architecture Analysis & Design Language (AADL)

Definition

- **Architecture** description language standard
- Model the software and hardware architecture of an **embedded**, **real-time** system.
- Can be used either as a design documentation, for analyses (such as **schedulability** and **flow control**) or for code generation (of the software portion)
- High-level architectural concepts
- Key building blocks, such as **processor**, **devices**, **threads**, and rules to assemble them

PaceMaker Design example



shows an AADL specification of the Pacemaker. It contains the following design objects: the AADL processor PG_board, which is the main computation unit of the Pacemaker; the AADL device accelerometer, which provides information about the patient's physical activities in order to regulate heartbeats; the AADL process PG_controller, which controls the heartbeat functions of the patient's activity, the configuration set provided by a practician, and of course, measures of natural beats in the patient's heart. This process component is composed of three AADL threads

52

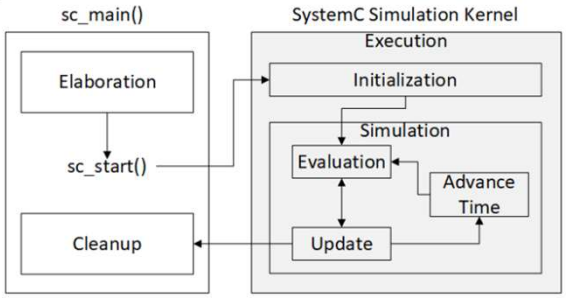
System C

System-level modeling

- Architectural exploration, performance modeling, software development, functional verification, and **high-level synthesis**.
- Set of C++ classes and macros which provide an **event-driven simulation interface**
- **Simulated real-time environment**, using signals of all the datatypes offered by C++
- Deliberately mimics the hardware description languages **VHDL** and **Verilog**
- **Features:** Modules, Ports, Signals, Exports, Processes, Channels, Interfaces, Events, Data types
- Source code is **compiled** with the SystemC library to give an executable.

Version:

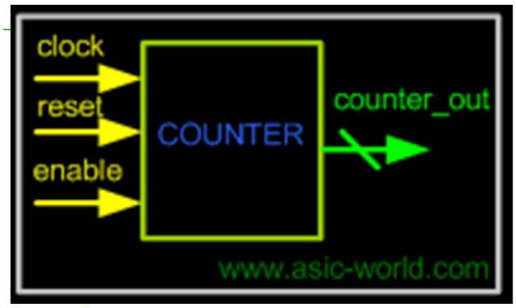
1. **Hardware-description language** features such as structural hierarchy and connectivity, clock-cycle accuracy, delta cycles, four-valued logic (0, 1, X, Z), and bus-resolution functions
2. **Communication** abstraction, **transaction-level** modeling, and **virtual-platform** modeling. It also added abstract ports, dynamic processes, and timed event notifications.



53

System C example

```
SC_MODULE (first_counter) {  // module (class) declaration
  sc_in_clk     clock ;      // Port: Clock input of the design
  sc_in<bool>   reset ;      // Port: Active high, synchronous Reset input
  sc_in<bool>   enable;      // Port: Active high enable signal
  sc_out<sc_uint<4> > counter_out; // Port: 4 bit vector output
  sc_uint<4>    count;       // local variable
  //------------Code Starts Here------------------------
  void incr_count () {       // process
    if (reset.read() == 1) {
      count =  0;
      counter_out.write(count);
    } else if (enable.read() == 1) {
      count = count + 1;
      counter_out.write(count);
    }
  }
  SC_CTOR(first_counter) {   // Constructor
    SC_METHOD(incr_count);  // register incr_count to kernel
    sensitive << reset << clock.pos(); // sensitivity list of incr_count
  }
};
```



54

Vocabulary

Methodologies & Languages

- **MBSE**: *Model-Based Systems Engin* : models instead of documents for system design.
- **UML**: *Unified Modeling Language* : Standard modeling language for software systems.
- **SysML**: *Systems Modeling Language* : Extension of UML for systems engineering.
- **AADL**: *Architecture Analysis & Design Language* : for modeling and analyzing embedded system architectures.
- **HDL**: *Hardware Description Language* : Language for *eering* – Using hardware design (e.g., VHDL, Verilog).

Tools & Technologies

- **MATLAB**: *Matrix Laboratory* : Environment for numerical computing and simulation.
- **FPGA**: *Field-Programmable Gate Array* : Reconfigurable hardware device.
- **ASIC**: *Application-Specific Integrated Circuit* : Custom-designed chip for a specific application.
- **RTOS**: *Real-Time Operating System* : OS designed for real-time constraints.

Key Concepts

- **SoC**: *System on Chip* : Complete system integrated on a single chip.
- **IoT**: *Internet of Things* : Network of connected devices.
- **Co-design**: : Joint hardware/software design approach.
- **Edge Computing**: : Processing data near the source (sensors).

55