Introduction to System-level and High-level Design

Created by
Yoshio Takamine

Updated by
System Level Design Group
Frontend Department
Renesas Design Vietnam Co.Ltd.

Abstract

System-level design and high-level design are advanced design methods which are supposed to take over from current RTL design eventually.

System-level design: Reduces cost and period for design and development of large-scale and complex SoC by preventing lots of design problems relating to specifications including performance and power consumption.

High-level design: Adopting behavioral synthesis technology, it reduces design cost by reducing number of lines for design description and by accelerating simulation speed.

In this material, this symbol indicates important points.

Purpose of this course

The following information will be transferred:

- **Some issues of LSI design, especially in following steps:**
  - Specification design
  - RTL design
  - Software design
  - System (HW&SW) evaluation
- **Tools and methodologies:**
  - SystemC
  - System Level Design
    - Modeling.
    - Virtual platform.
    - Architecture design and evaluation.
  - High Level Design
    - High level synthesis.
    - HW optimization.
- **How the issues are solved with above methodologies.**

Agenda

1. Background and Objectives
   - 1.1 Basic concepts
   - 1.2 Background
   - 1.3 Aims of System-level and High-level Design
2. System-level Design
   - 2.1 Introduction
   - 2.2 System-level Design Flow
   - 2.3 System-level Design Challenges
   - 2.4 SystemC: A System-level Design Language
   - 2.5 System-level Design Platform
   - 2.6 Another Design Platform for Performance Evaluation
   - 2.7 Summary
3. Virtual Platform for Embedded Software Development
4. High-level Design
   - 4.1 Introduction
   - 4.2 Aims of High-level Design
   - 4.3 High-level Design Flow and Tools
   - 4.4 Application Example
   - 4.5 Summary
5. Concluding Remarks

1. Background and Objectives

Basic concepts (1)

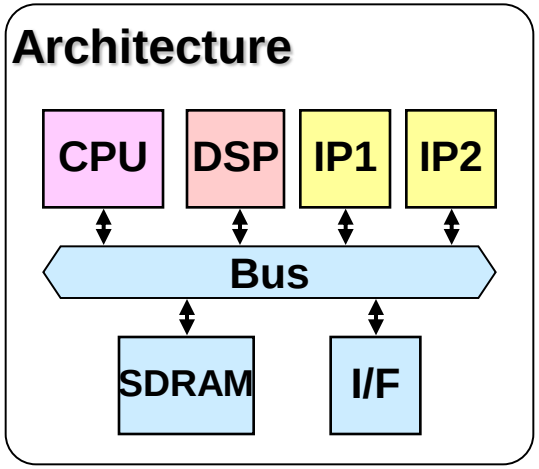
Specification

- Detailed descriptions of something to be developed (systems, LSIs, programs, components, …)
- LSI specification consists of three parts:
  - Interfaces
  - Functions or behavior
  - Design constraints

Basic concepts (2)

Architecture

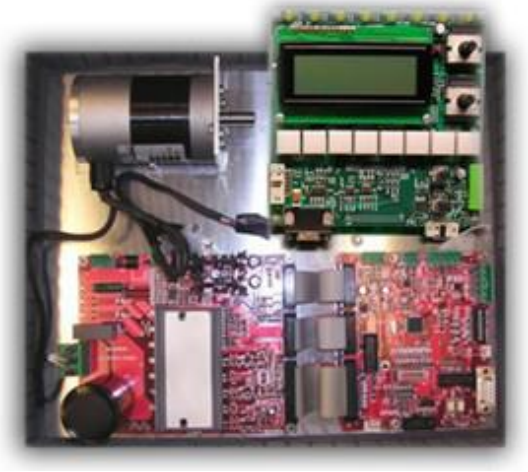
- Aspect of design that is supposed to realize given function
  - Components exist
  - Properties of the components
  - Connections between the components



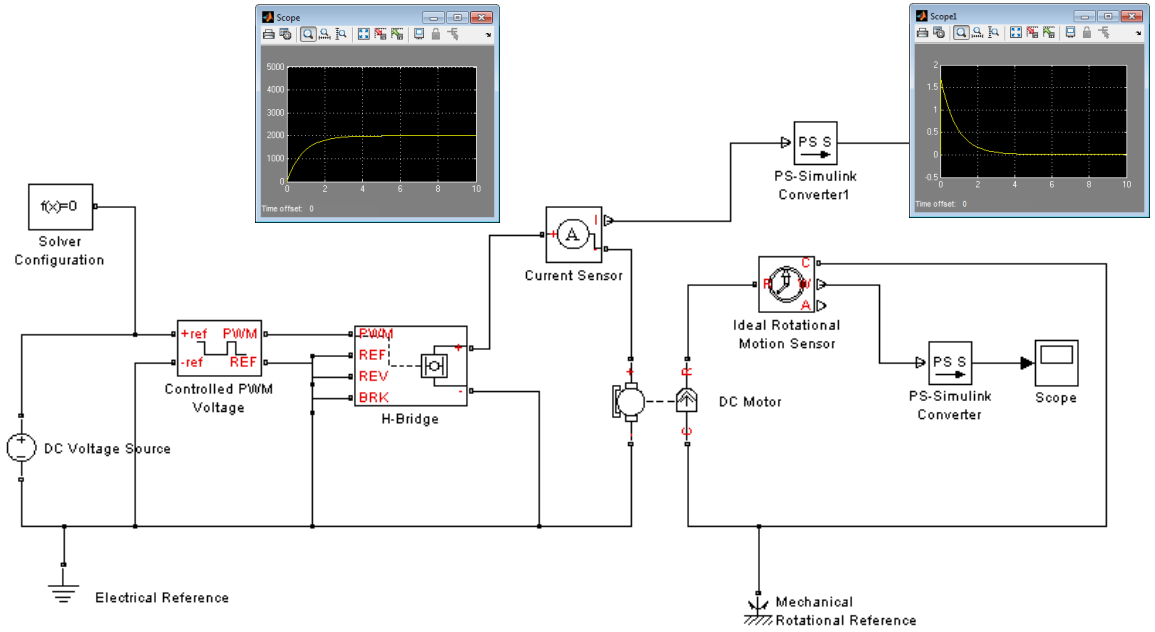
Basic concepts (3)

Model

- **Simplified or abstracted representation of a target system.**
- **Represents only necessary functions or behaviors according to the purposes of the model.**



DC motor control system



Simulation of DC motor control using models on Simulink

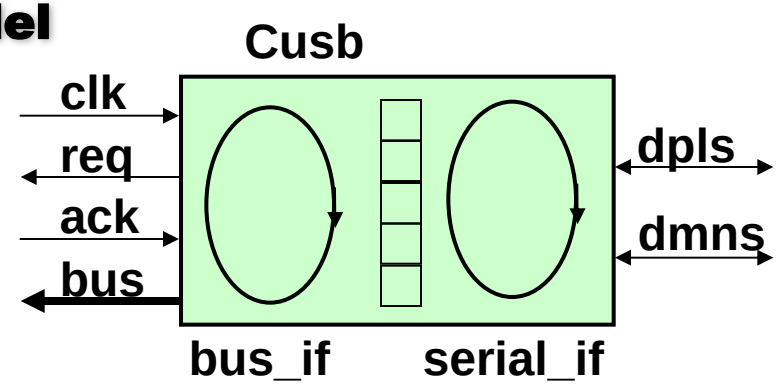
Basic concepts (4-1)

- **SystemC**
  - A library of C++ classes.
  - Easy to describe both hardware and software behavior.
  - High-speed simulation.
  - Free development environment.

Basic concepts (4-2)

An example of SystemC model

```
class Cusb : public sc_module{
public:
  sc_inout<bool>   dpls, dmns;
  sc_out  <bool>   req;
  sc_in   <bool>   clk, ack;
  sc_out  <sc_uint<32> > bus;
  Cusb(sc_module_name n) : sc_module(n)
  {
    SC_THREAD(serial_if);
    sensitive << dpls, dmns;

    SC_METHOD(bus_if);
    sensitive << clk.pos();
  ...
};

void Cusb::serial_if(){
  buffer[index++] = dpls && dmns;
  ...
}

void Cusb::bus_if(){
  if( index > 0 ){
  req = 1;
  while( ack.read() == 0 ) wait();
  ...
```



Port declaration

Specify bit width

Function call condition

Serial I/F func

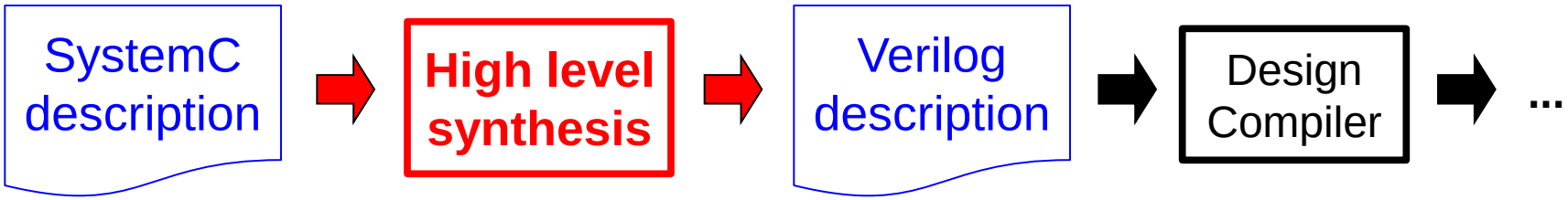
LSI bus I/F func

Parallel execution

Basic concepts (5-1)

High level synthesis

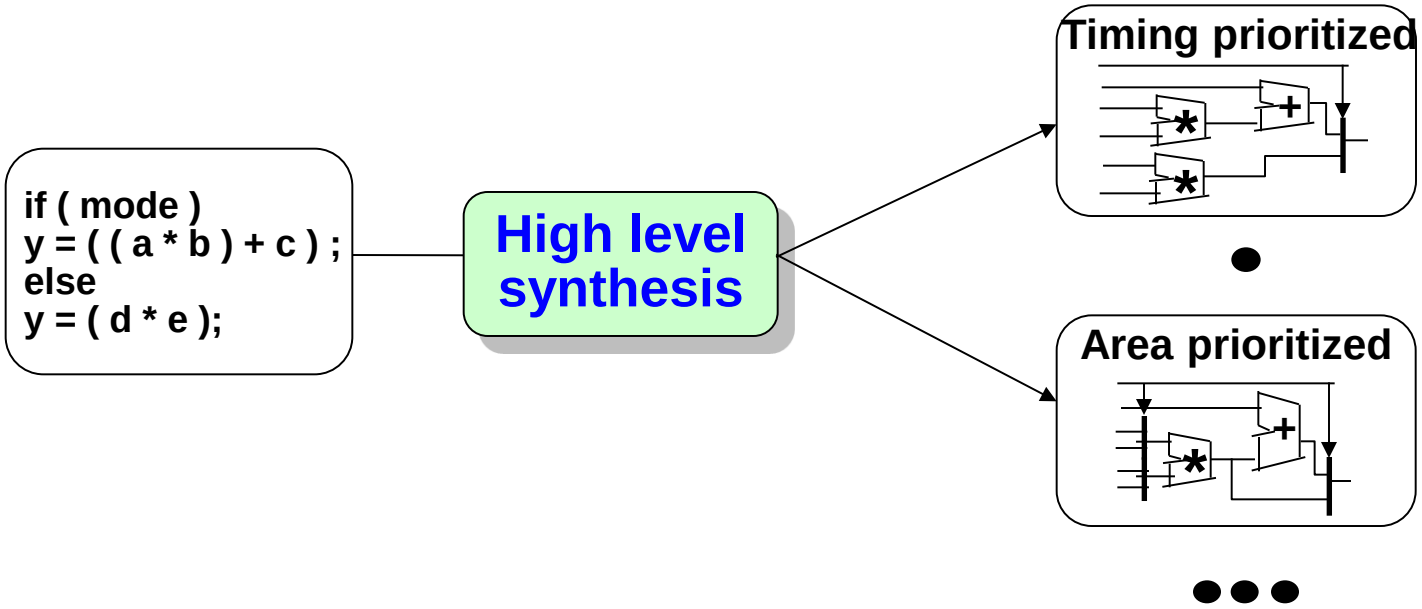
- converts C/SystemC description to synthesizable Verilog description.



Basic concepts (5-2)

High level synthesis

- **Hardware optimization**: Different HW designs can be generated from one SystemC design based on the constrains of timing and area of behavioral synthesis.

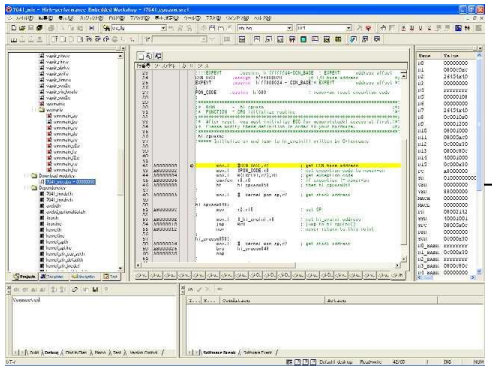


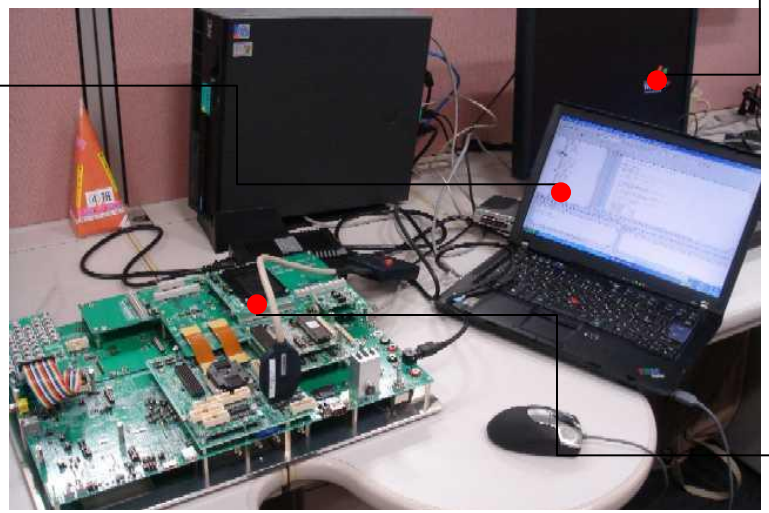
Basic concepts (6)

Software Test and Debug Environment

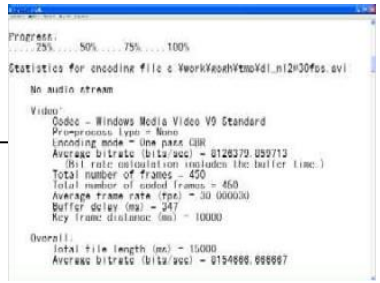
- Software is tested and debugged on the reference board with real LSIs.

Software debugger





Terminal



Display logs

LCD panel

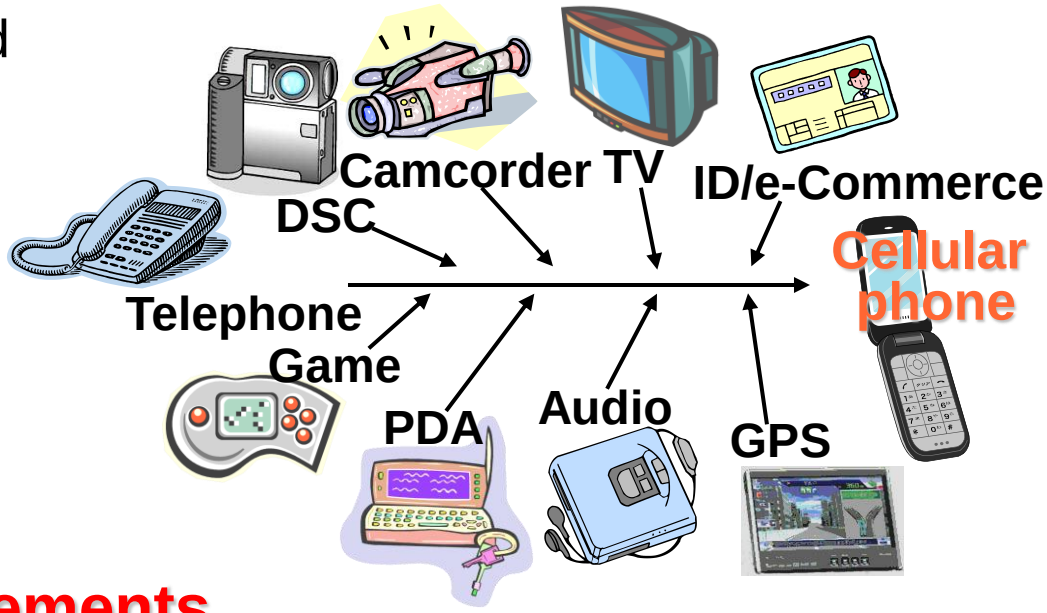
Display graphics

Background (1-1)

- **Electronic machineries are getting more and more complicated, higher and higher performance**

  << e.g. Cellular phone >>

  - New functions are added every year
  - High-performance communications, large-sized and high-quality displays, etc.



- **Difficult design requirements**

  Convergence of many functions

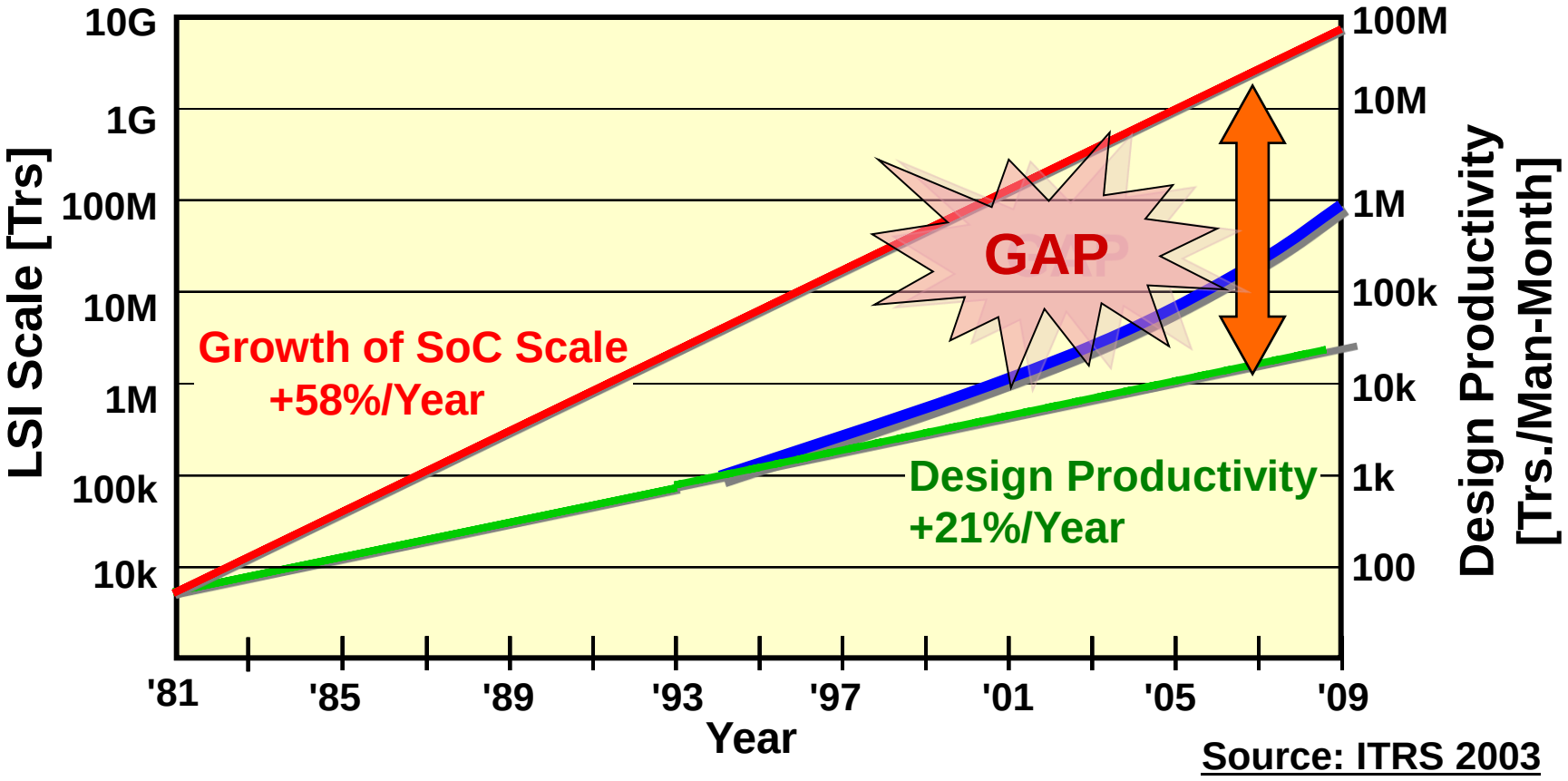
  - **High performance** and **low power consumption**
  - **Short TTM (Time-to-Market)** and **low cost** **(for both development and mass production)**

Background (2-1)

- Electronic machineries are getting higher performance
  ⇒ **Increasing** SoC's scale (1.5 ~ 1.6x per year)

However, process technology is improved **slowly**

⇒ **Big improvement of design productivity is necessary**

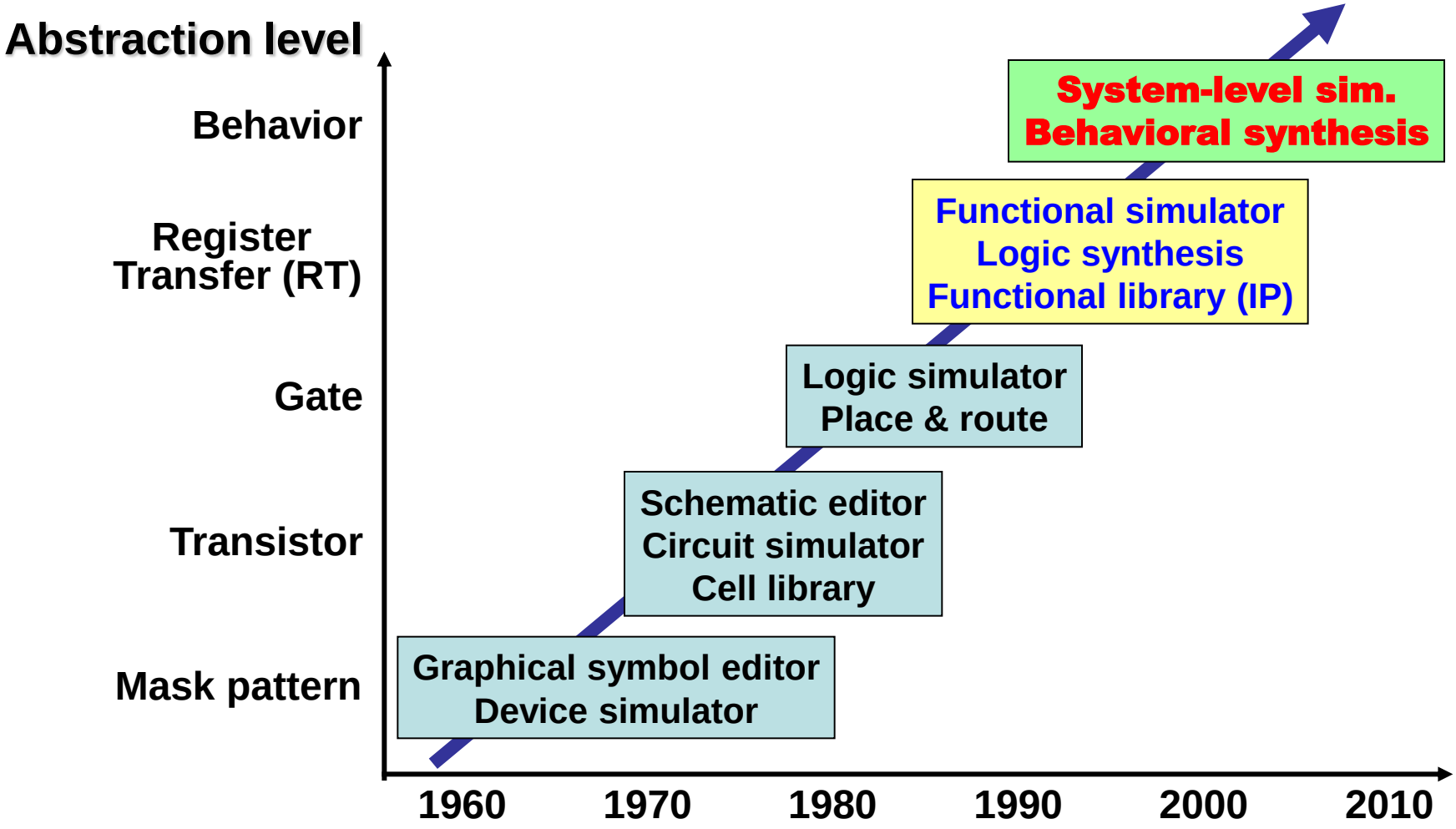


Source: ITRS 2003

Background (2-2)

- **Design methodology** change drastically **in every ten years**

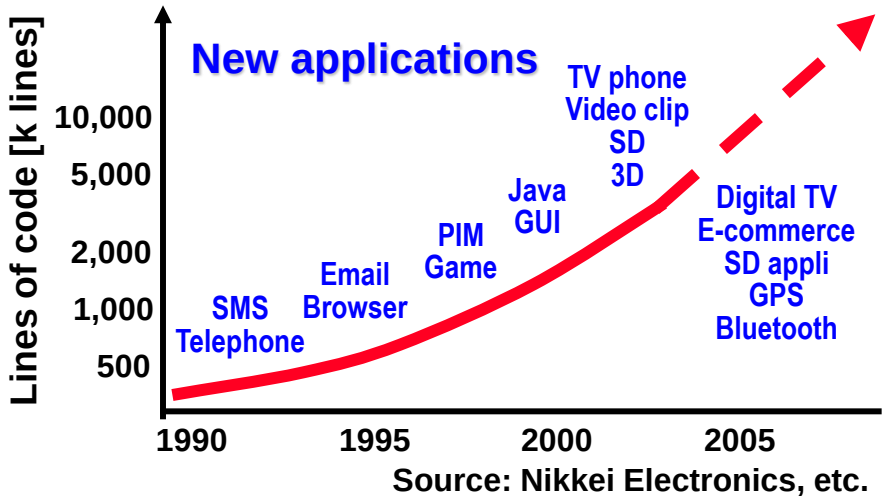
  ⇒ **Now it's time to shift to system/behavior-level**



Background (3)

- Electronic machinery is getting more complex

⇒ Scale of software development has been growing as exponential function



Source: Nikkei Electronics, etc.

- Bigger challenges than SoC development
  - Much longer period for software and system test after LSI samples ready
    - Huge number of tests
  - Serious incompatibilities between hardware and software specifications and emergence of unassumed use cases
    - Detect and debug hardware design errors simultaneously
  - If any problems relating to system specifications, performance, etc. at last of the development

⇒ Cause serious re-design or re-spin

Background (4)

- **Customer's demand for "virtual platforms" is increasing**
  - **CPU models (SH, V850, RX600, etc.)**
  - **Peripheral IP models (Timer, DMA controller, SD controller, etc.)**

◆ Virtual platform of automotive MCU:

✓MCUs become key components of cars and total amount of software gets huge
- ◆Various kinds of engines: combustion, electric, hybrid.
- ◆Many functions for safety, reducing gas and electricity, etc.

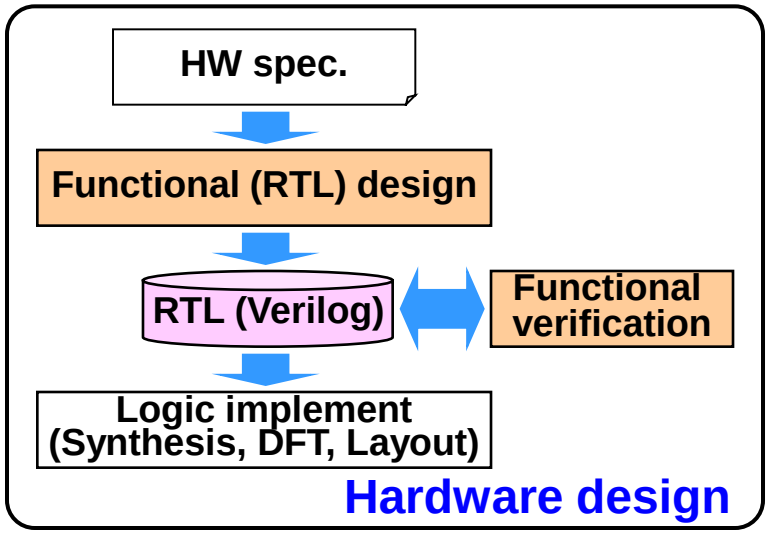
⇒ Required to start software development as early as possible and to reduce software development cost

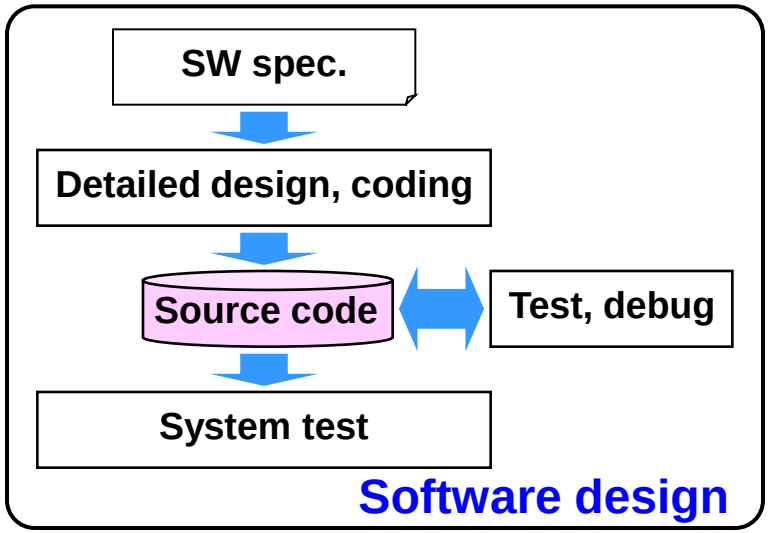
✓Model-based development becomes popular.
- ◆Instead of crashing real cars, testing on models can save huge cost
- ◆Easy to produce certain conditions for tests

⇒ Car makers ask semiconductor vendors to provide component models to build their model-based environment

What are System-level design and High-level design (1)

- Present hardware (LSI) and software design flows
  - LSI is designed at RT (Register-Transfer or functional) level in Verilog HDL, and passed to logic implementation step
  - Hardware and software specifications are developed separately and may have some incompatibilities or inconsistencies

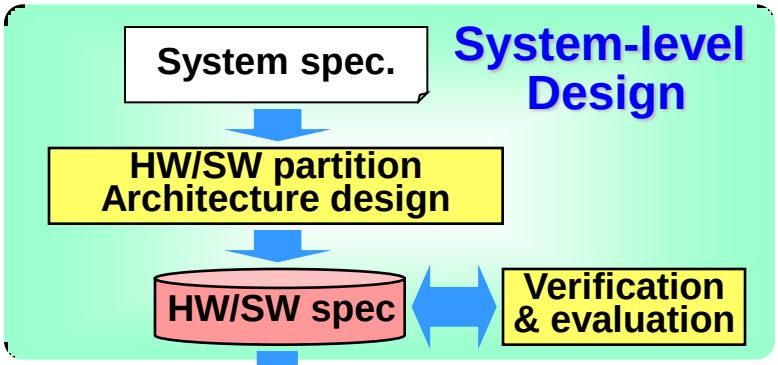




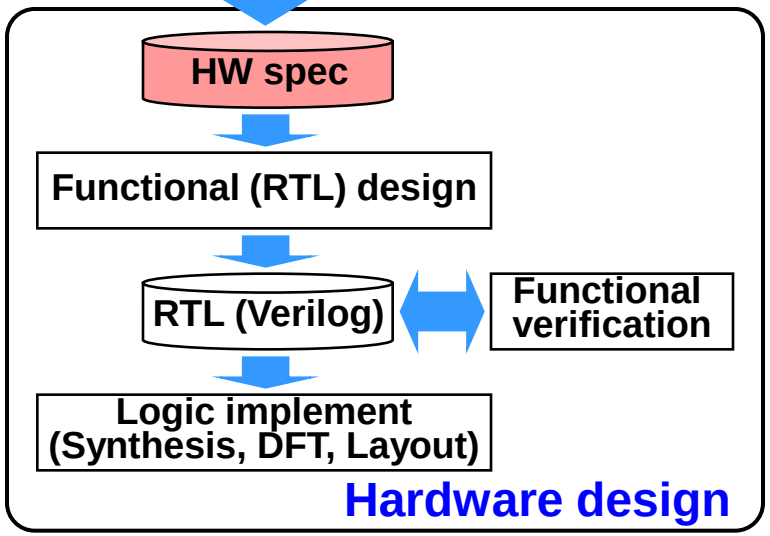
What are System-level design and High-level design (2)

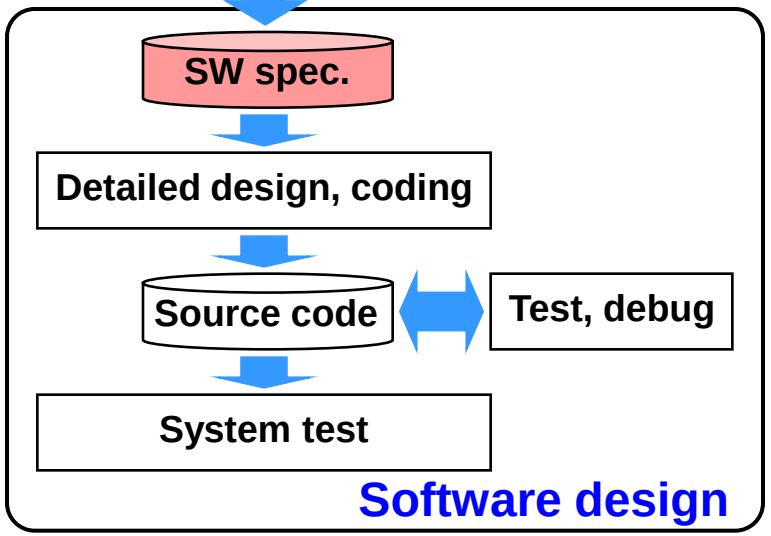
System-level design

- Develop **appropriate hardware and software specifications** according to system specification



- Purpose:
  Make following HW/SW design very efficient and reduce total development period and cost

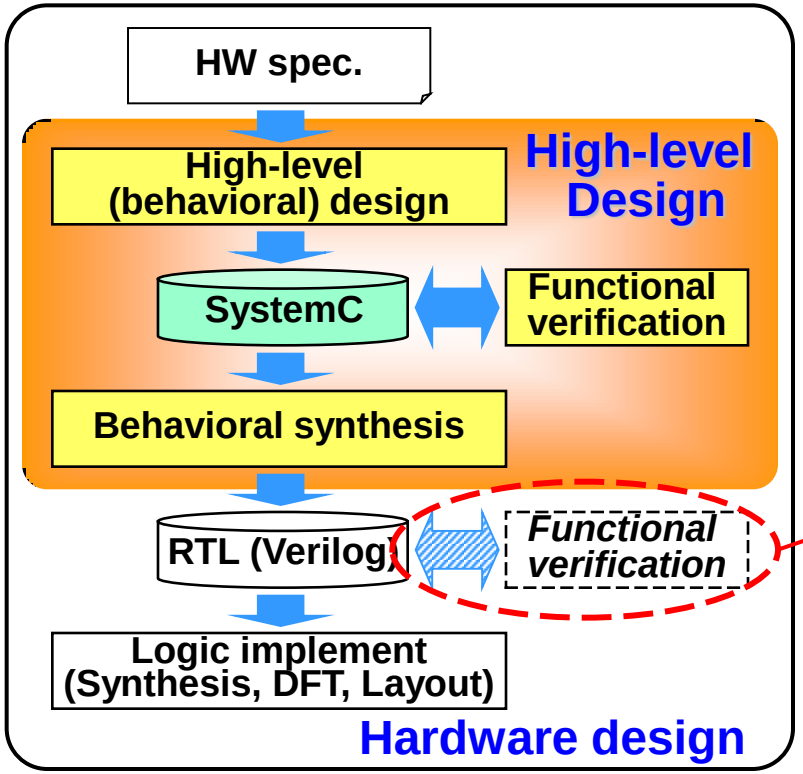




What are System-level design and High-level design (3)

High-level design

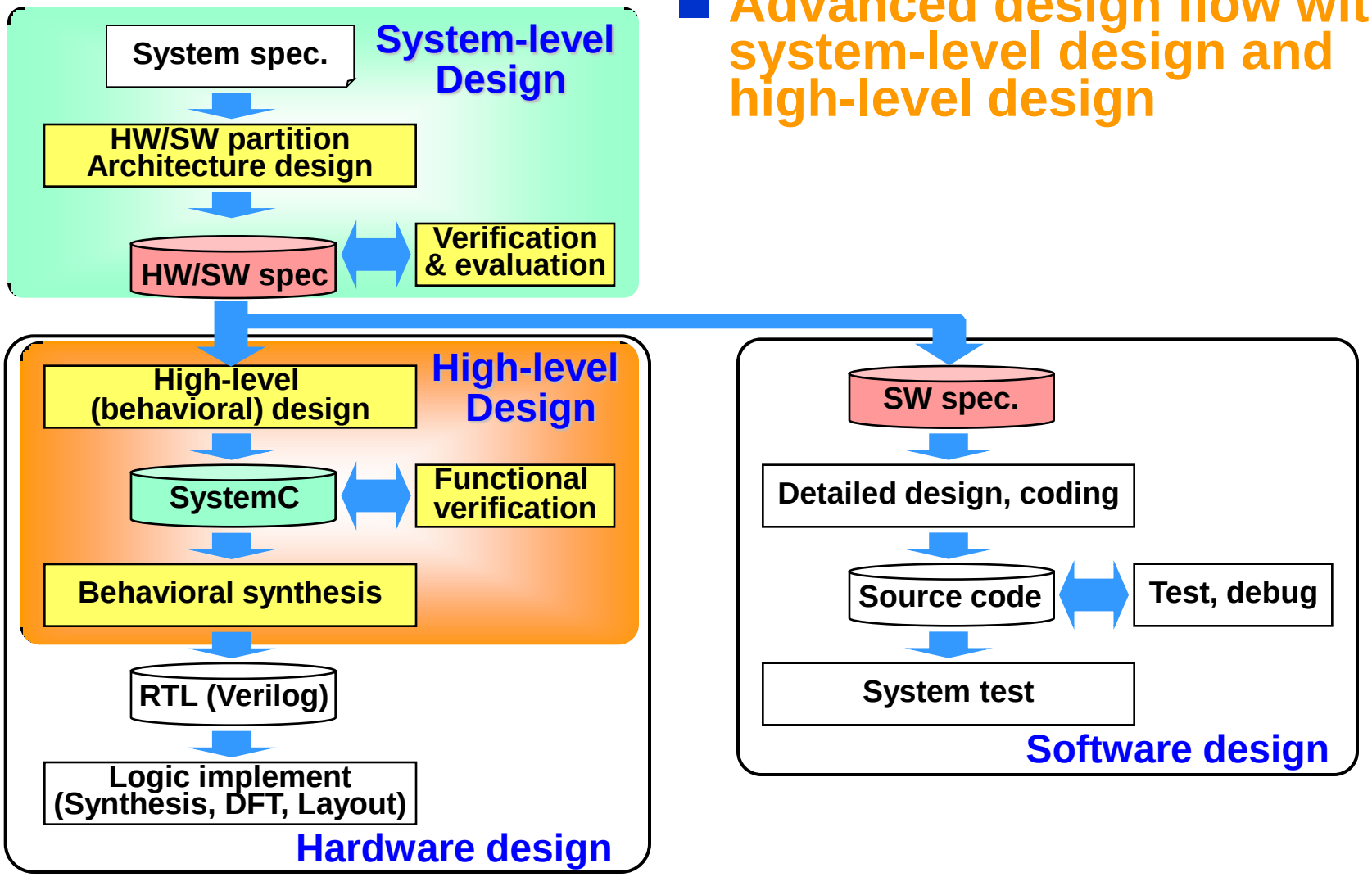
- Design hardware at higher level (i.e. behavioral) in SystemC
- RTL description (Verilog HDL) is synthesized by a tool from the SystemC description



- Purpose:
  Reduce hardware design cost by designing at abstracted level

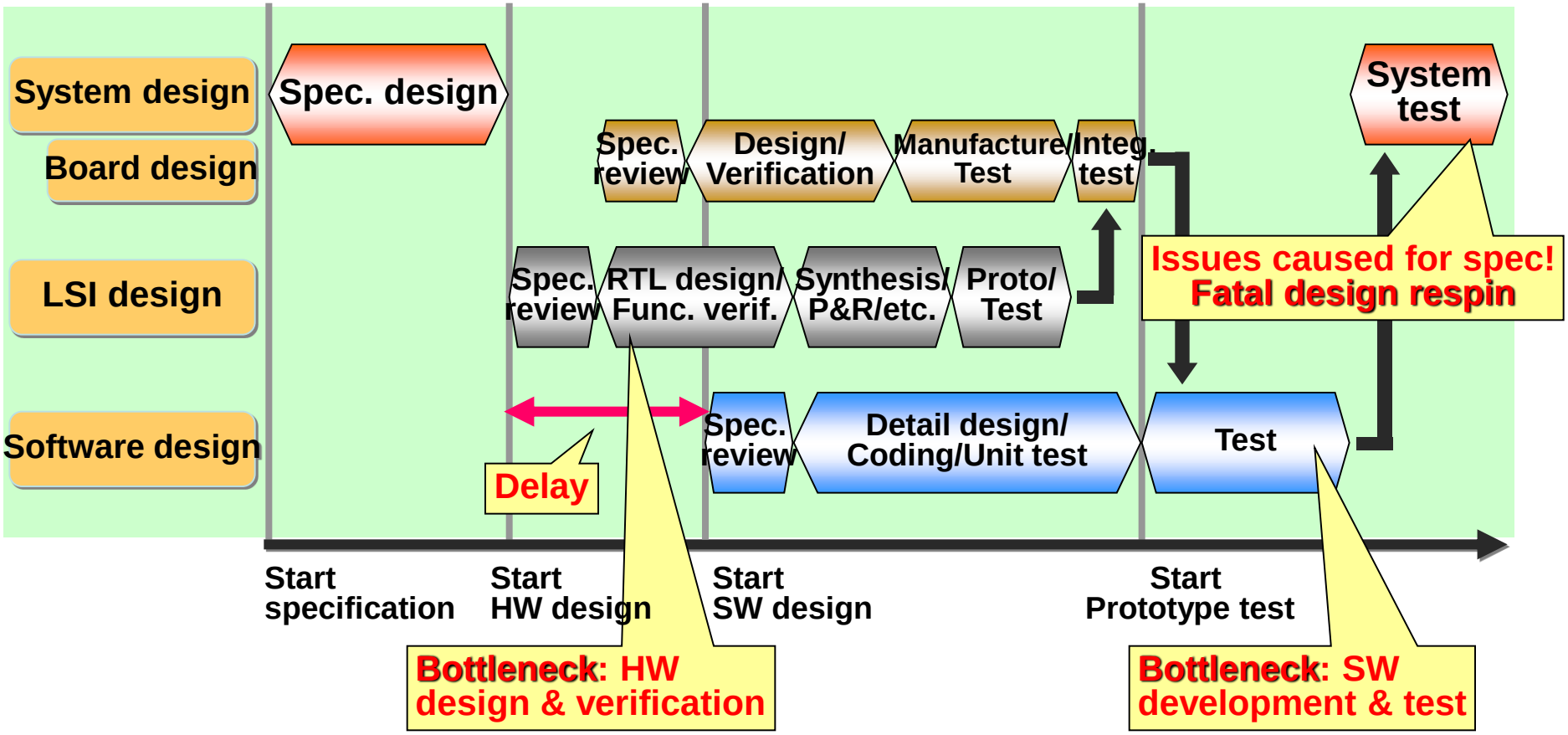
What are System-level design and High-level design (4)

- Advanced design flow with system-level design and high-level design



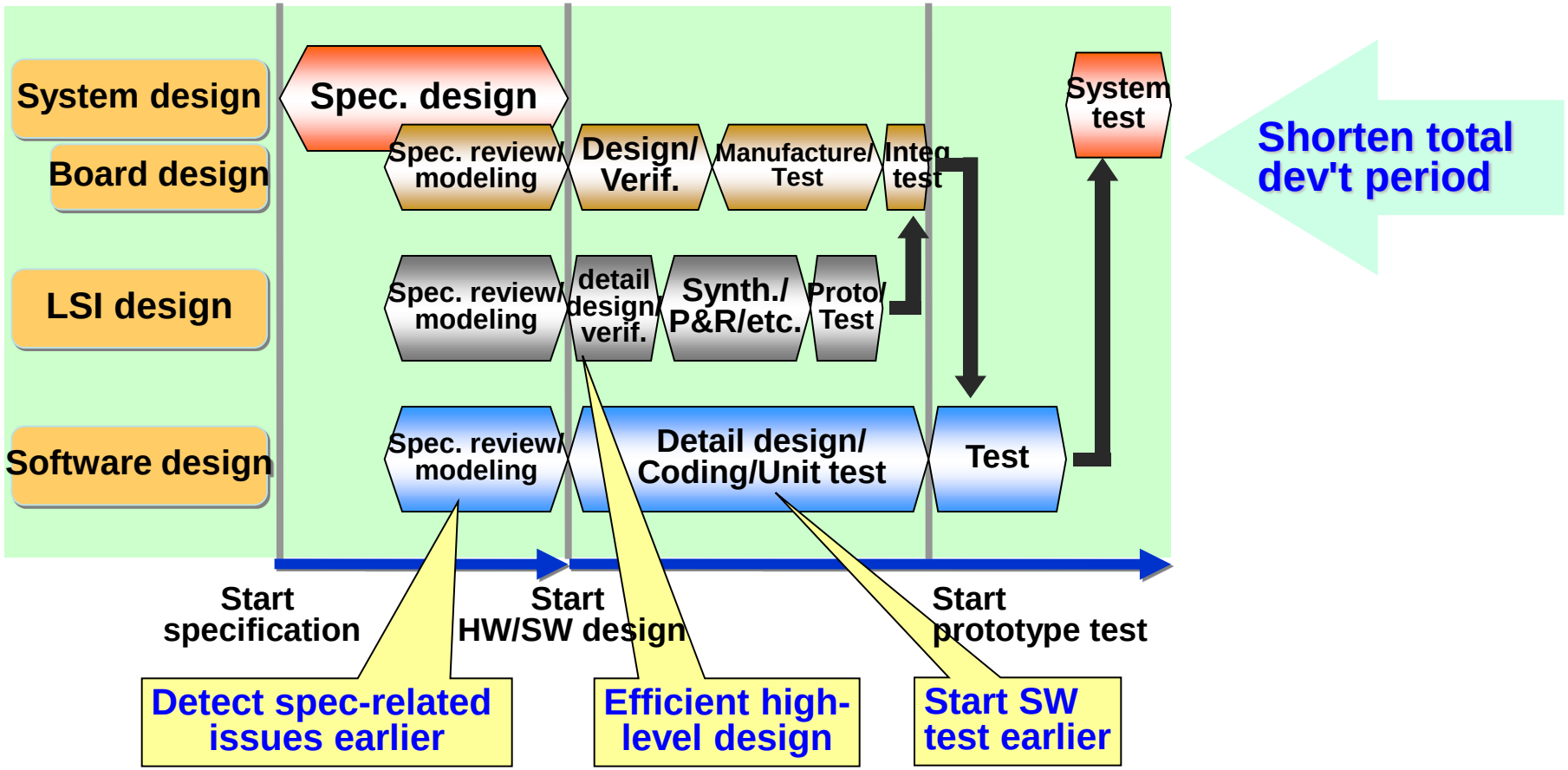
Aims of System-level design and High-level design (2-1)

- System consists of hardware (LSI, board) and software
- Most of software test and system test done at the end of development period and takes very long time

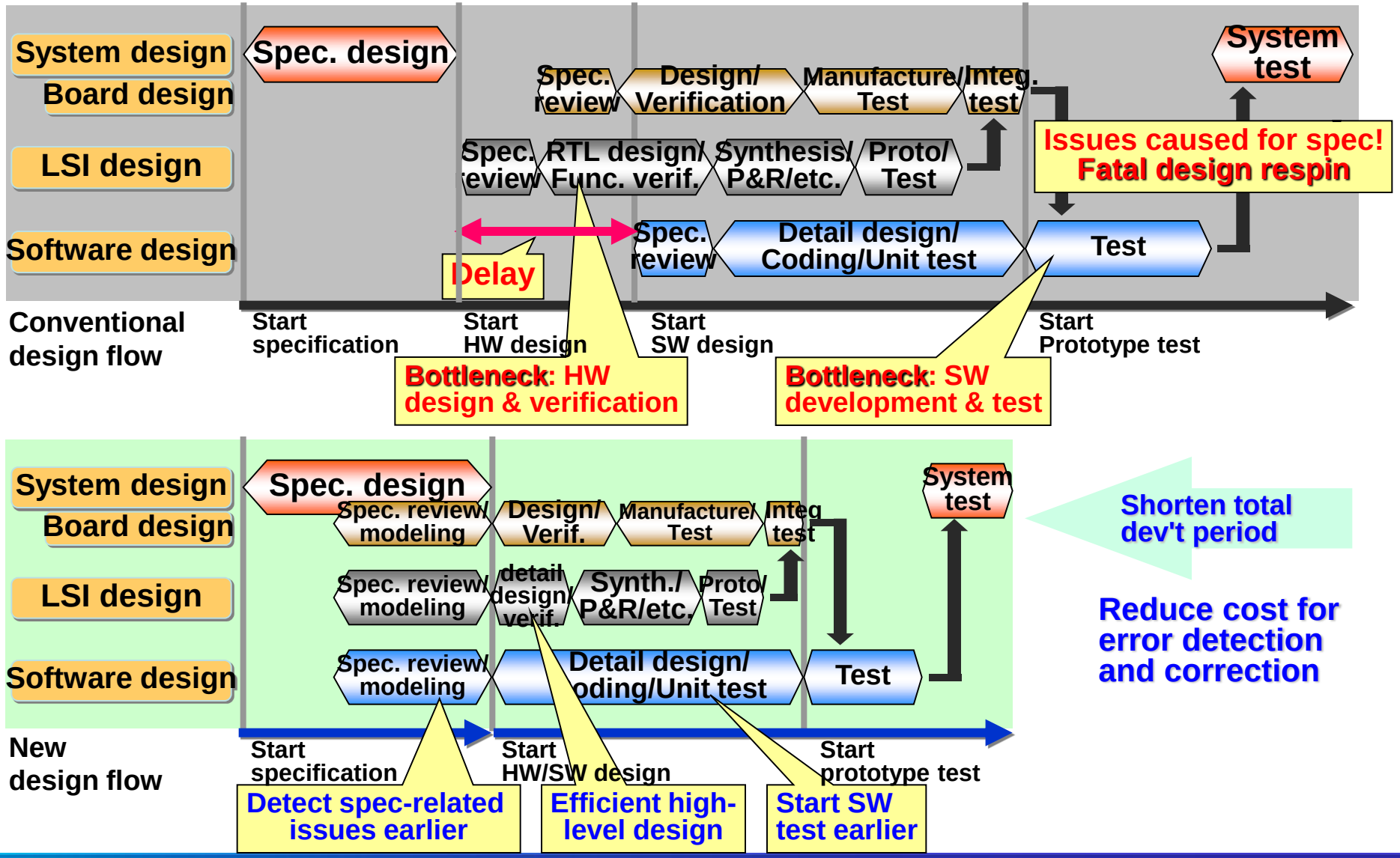


Aims of System-level design and High-level design (2-2)

- Concentrated specification design and optimization in early stage to prevent any serious redesign and respin
- Start software development just after specifications are fixed to finish it much earlier



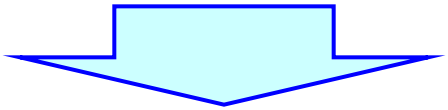
Aims of System-level design and High-level design (2-3)



2. System-level Design

Aims of System-level Design

- **Problems of system test**
  - So many tests and evaluations performed using actual LSIs
  - Take very long time and much cost for analyzing and fixing bugs
- **Carry out system test and evaluation before RTL design**
  - Evaluation and verification with effective benchmarks based on real use cases
  - This used to be carried out after sample SoCs and reference boards completion

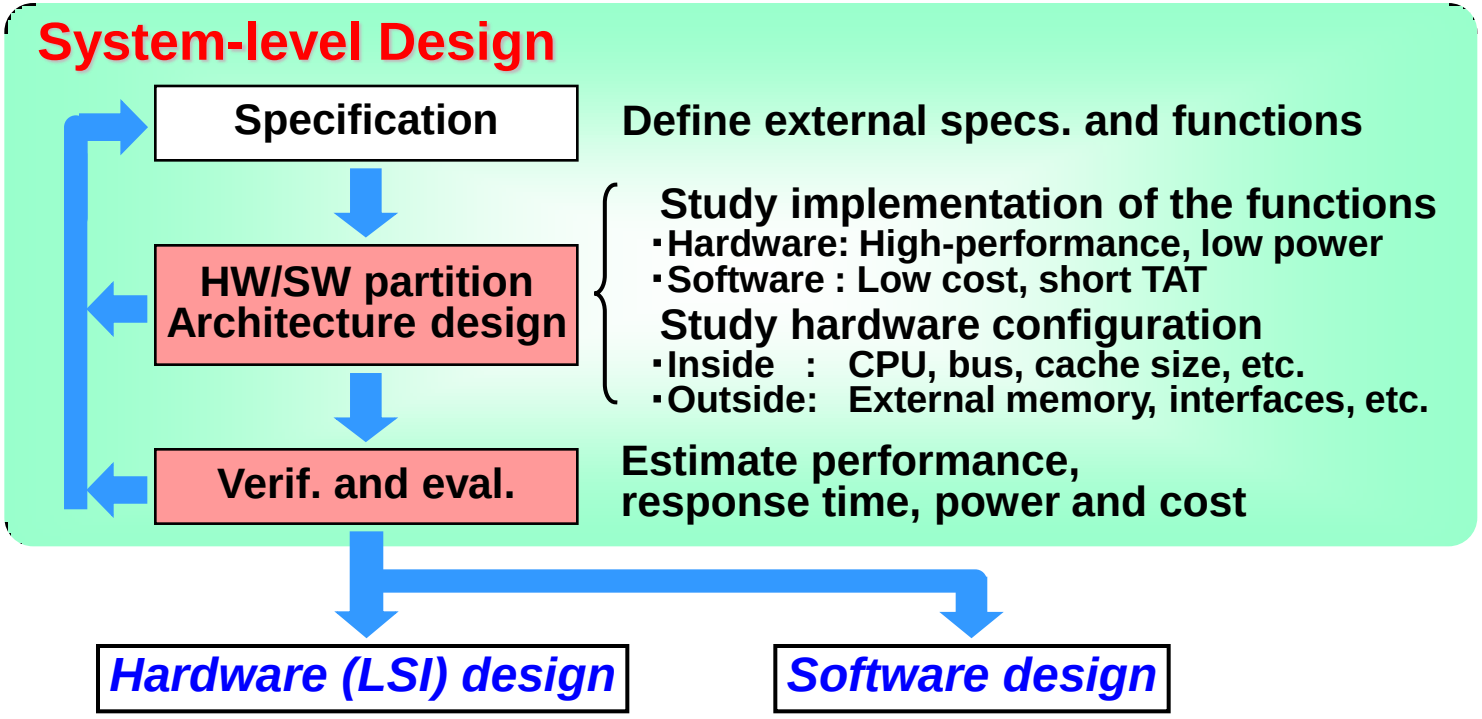


- **Architecture exploration and optimization aiming to realizing specifications**
  - Requirements including functions and performance
  - Also drive for better performance, lower power consumption and lower cost for mass production as possible
- **Detect any design errors earlier than before**

  ⇒ Reduce cost for detecting and correcting errors

System-level Design Flow (1)

- **System-level design** is to **determine system architecture** that realizes desired specifications and to **describe concrete specifications for hardware and software development**
- **Verification and quantitative evaluation** is very important for the architecture design at higher level



System-level Design Flow (2) — Studying Specifications ① —

- **What are specifications?**
  - **Detailed descriptions of something to be developed (systems, LSIs, programs, components, …)**
  - **Consist of three parts:**
    - **Interfaces**
    - **Functions or behavior**
    - **Design constraints**
  - **Interfaces:**
    - **Names, sizes and other properties of inputs, outputs, registers, etc.**
    - **Protocols for communication**
  - **Functions or behavior:**
    - **What to output as response for a certain input**
    - **How to calculate or process input data**
    - **How to deal exceptions**
    - **etc.**
  - **Design constraints:**
    - **Cost: die size or number of elements, number of pins, ...**
    - **Performance: throughput, response time, ...**
    - **Power consumption**
    - **etc.**

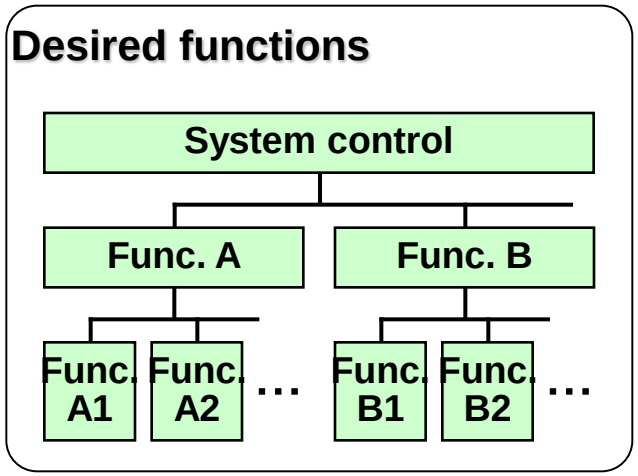
System-level Design Flow (2) — Studying Specifications ② —

Describing specifications

- Share the specifications among the persons concerned
- Requirements for the description
  - May not have any defects or any inconsistencies
  - May not have any ambiguities or any causes of misinterpretation
  - Machine-readable or executable
- Natural languages are commonly used, but it causes some errors ⇒ UML (Unified Modeling Language) becomes popular

Defining functions

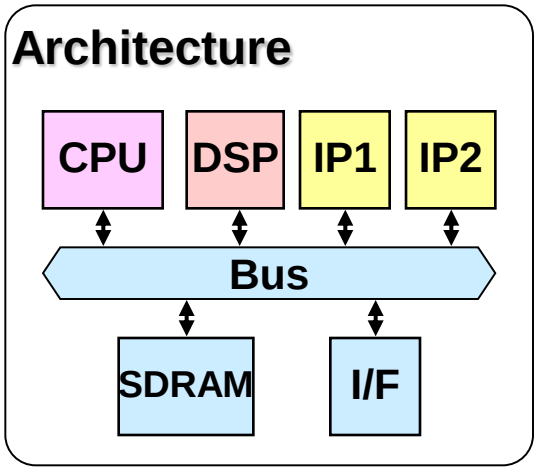
- Define functions in detail from the desired specifications
- Write executable models in C/C++
- Simulate system's behavior using the models to verify if the specifications are fulfilled
  - Focus attention on algorithms and precision for data manipulation
- Functions are represented as hierarchical manner



System-level Design Flow (3) — Architecture Design ① —

What is architecture?

- Aspect of design that is supposed to realize given function
  - Components
  - Properties of the components
  - Connections between the components



LSI architecture:

- **CPU, DSP:** Type of CPU (SH-4A, V850, RX600, ARM, ...), frequency, cache size, etc.
- **Dedicated accelerator:** Graphic processor, sound processor, etc.
- **Bus and controller:** Type of bus (SHwy, AXI, ...), band width, structure (hierarchy), arbitration scheme, etc.
- **Memory:** Size, number and bit width of ports, etc.
- **External communication interface:** Types, band width, etc.

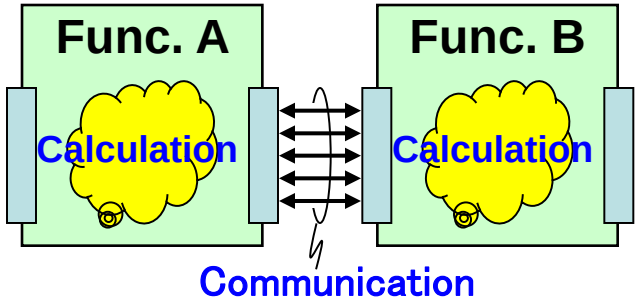
Hardware / software partition

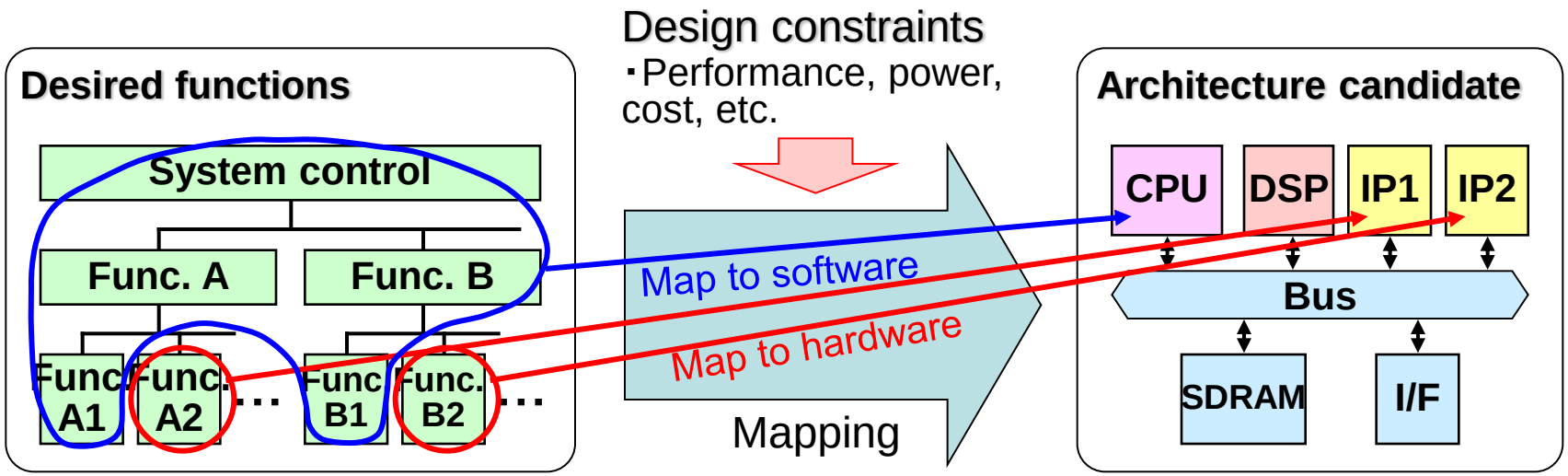
- **Hardware:** Dedicated to certain functions
- **Software:** Functions running on CPUs/DSPs

System-level Design Flow (3) — Architecture Design ② —

Architecture exploration

- Divide and merge the functions into blocks in accordance with amount of calculations and communications
- Prepare several architecture candidates
  - CPU, DSP, dedicated hardware, bus width, frequency, memory size, etc.
- Map the functional blocks on the architecture and evaluate
- Choose the best architecture





System-level Design Flow (3) — Architecture Design ③ —

How to evaluate architecture

- A conventional way
  - Basically rely on designer's experiences and intuitions
  - Sometimes use Excel and/or special purpose small C models
  - Tend toward pessimistic results which require more hardware resources
- New model-based evaluation
  - Use executable models and run appropriate benchmarks which correspond to actual use cases very well
  - Possible to derive optimal condition; sufficient and not too much
  - Possible to detect functional defect at the same time

⇒ Reduce following design and verification period

What is a model?

- Simplified or abstracted representation of a target system
- Represent only necessary functions or behaviors according to the purposes of the model

System-level Design Flow (4) — System-level evaluation ① —

- **Performance problems which can only be realized when real application and data applied**
  - **Difficult to imagine exact behavior of a complicated system;**
    - **Actual operational time of IPs, any conflicts and arbitration of requests on system buses, behavior of input / output data, …**
  - **Understanding real applications are not easy for semiconductor vendors**

- **Evaluation of power consumption with real application**
  - **It takes too long simulation time and the results might be too late if RTL design or gate-level netlist are used**

- Example for Mobile SoC
  - Motion picture recording and reproduction with high-definition camera and large size LCD
  - High-resolution and high-speed 3D graphics for entertainment
  - Long time reproduction of music and video, reception of surface digital TV, and talking with videophone

System-level Design Flow (4) — System-level evaluation ② —

- **Inconsistency of specification between IP and SoC, hardware and software**
  - Use cases that IP designers or hardware designers do not assume
  - Lack of functions for handling exceptions or extreme performance degradation by data that designers do not assume

- **Presentation of system-level performance to customers**
  - Quantitative estimation is very effective for getting customer's orders and defining SoC specifications
  - Customers (system makers) often practice such system-level evaluation by themselves

System-level Design Challenges

- **System-level design is an additional process and any evaluation takes time and cost**
  - Need to build an evaluation (simulation) environment and develop many models for it
  - Need to prepare benchmarks and modify them to run on the environment

  ⇒ **To avoid too much spending, do it appropriately**

- **"Clarification of objectives" is most important for the evaluation**
  - There are many interpretations (understandings) for "evaluation" mostly depending on the person's role
  - There are also many ways or means of evaluation depending on the objectives

  ⇒ **One evaluation way that is the best for a certain objective may not be the best nor even work for other objectives**
  - Different environments and benchmarks are necessary for each way of evaluation

  ⇒ **The cost for evaluation varies very much depending on the objective**

Clarification of Objectives

What is objective?

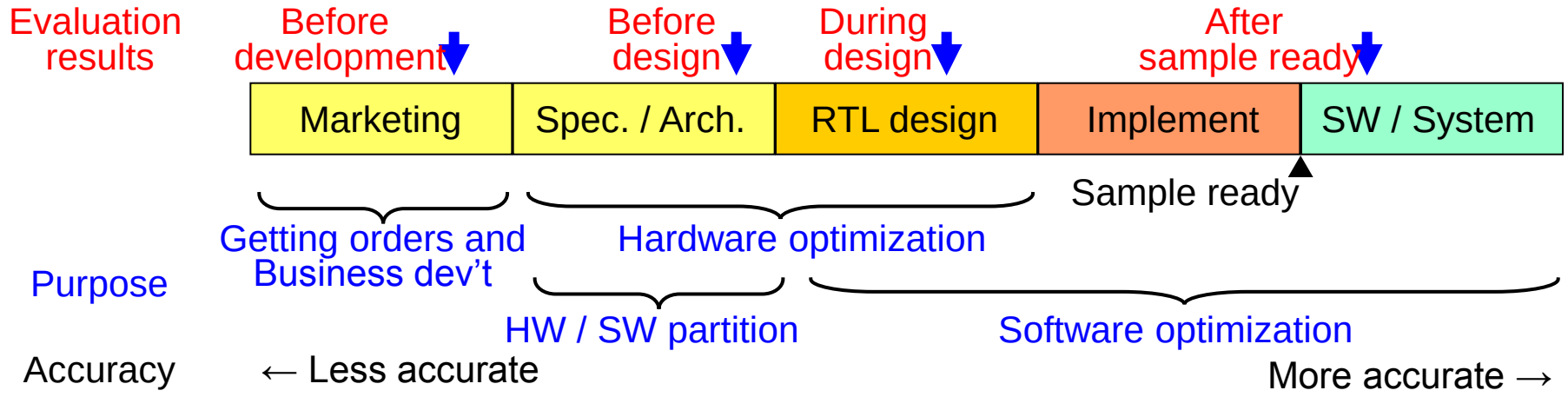
- **What** we want to know by the evaluation and **why** we want to know that, e.g.
  - Functional verification
  - Hardware and software partition (high-level architecture exploration)
  - Software optimization
  - Hardware optimization
  - Power reduction

Less timing info.

More timing info.

Power info. (obviously)

- **How accurate** the evaluation results should be
  - No timing accuracy needed?
  - 70%? or even 99%? (Almost same as RTL)
- **When** we need the evaluation results



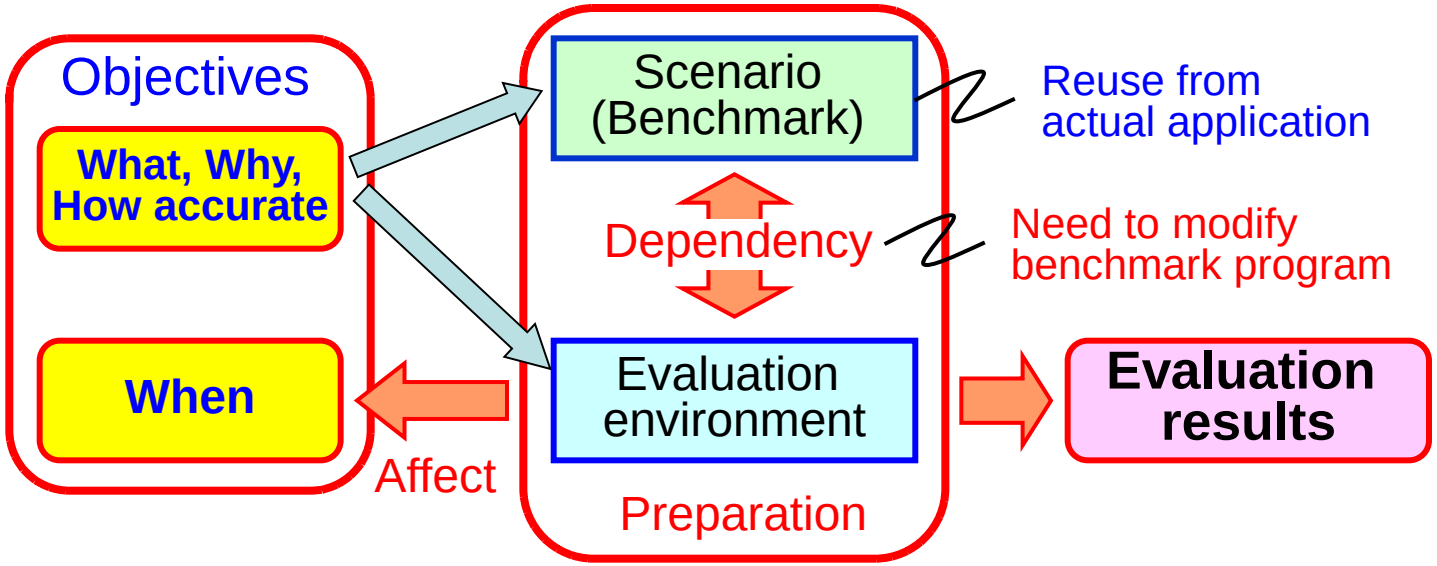
Evaluation Environment and Scenario

Objectives determine cost and time for evaluation

- **Objectives determine necessary environment and scenario**
  - **Need considerable time and cost for preparation**
  - **Difficulty for the preparation determines time and cost needed**

⇒ **Important to prepare appropriate environment and scenario according to the objectives to prevent spending too much**

- **Environment and scenario affect to each other and sometimes need to modify them so that the scenario can run on the environment**



- If environment or scenarios are not proper for the evaluation, we can't get expected results by the date, or lose opportunities

Abstraction Level of Models (1)

Abstraction level

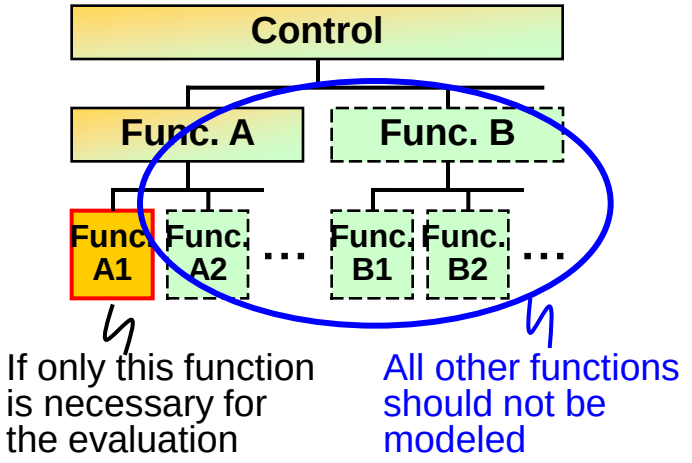
- Degree of details and accuracy for modeling in terms of functions, communications, structures, etc.
- There are various definitions and interpretation for abstraction level, and rules are necessary to use proper models

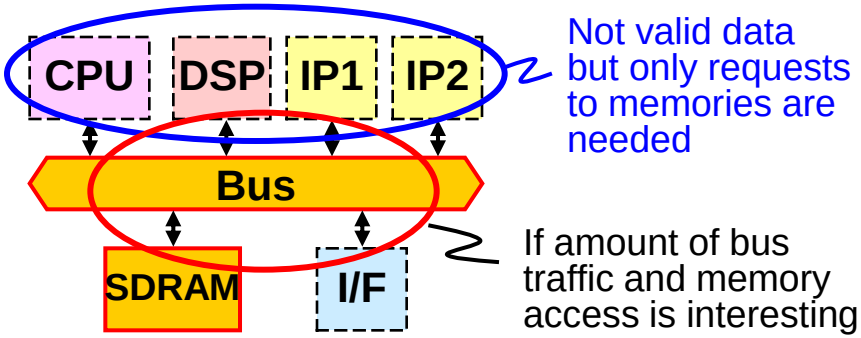
How to abstract models

- Function
- Time
- Communication

Abstraction of function

- Leave out unnecessary functions anyway
  - Error handlings are not necessary for most performance evaluation
  - Even data processing are not necessary for performance evaluation of memory access





Abstraction Level of Models (2)

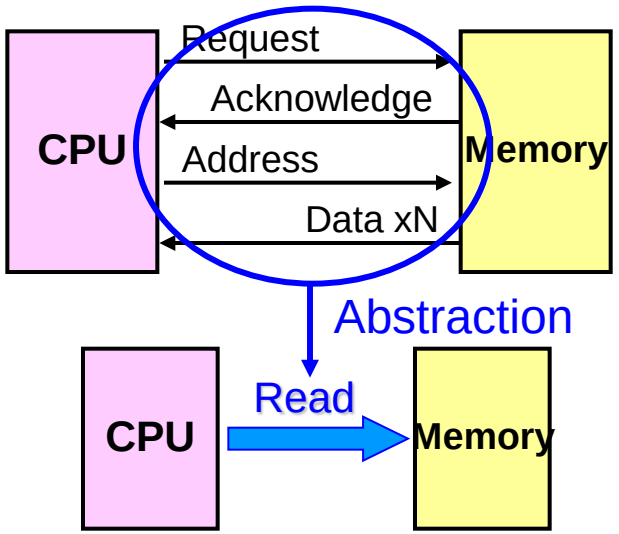
Abstraction of time

- Functional model or untimed model: no timing information
- Cycle Accurate model: same accuracy as RTL
- Approximately timed model: intermediate precision)

Abstraction of communication

- Transaction level model (TLM): Treat a sequence of operations as one transaction
  - Read, write, etc.
- Abstraction of time and communication are orthogonal
  - "Untimed transaction level models" and "approximately timed transaction level models" exist
  - Timing information is specified to each transaction (Differ from cycle accurate)

Example of read transaction



Impact to modeling by abstraction level

- Simulation speed and man-hours for modeling
  - Roughly double effort required for timed modeling compared with untimed modeling
  - More than one degree differences in simulation speed

Simulation Speed

- Is simulation speed enough for necessary evaluation?
  - Single simulation TAT (turn-around time): How long (how many seconds) necessary in real time for each benchmark
    - For example, simulation for a few seconds in real SoC takes several tens of minutes or a few hours
  - Total simulation time: How many scenarios exist for the entire evaluation
    - For example, a set of hundreds of one-hour scenarios takes several days or weeks

  ⇒ Determine necessary simulation speed

- Simulation speed is extremely affected by abstraction level of models
  - Functional simulation (no timing): several tens of MIPS
  - Cycle accurate simulation: sub (less) ~ several MIPS
  - A sole CPU model runs very fast in general, but peripheral models such as a graphic or video processor make whole simulation speed very slow
  - Much slower for mixed-level simulation with RTL models

System-level Design Language

- New design language is needed for system-level design
  - Verilog and VHDL are suitable for hardware description at RT (register-transfer) level but insufficient for higher-level description
  - Requirements for system-level / high-level description
    - Executable
      - Suitable for verification and evaluation
    - Cover various abstraction levels and representation styles;
      - RTL (hardware-aware) to behavior level
      - Functional and structural description
    - Express functions, design constraints and hardware structure
      - Algorithms, precisions, concurrency, clock, interruptions, etc.
    - Easy to write and maintain
      - Object-oriented method
- Many system-level design languages were proposed
  - Most of them were derivations from programming languages such as C/C++
    - Easy to describe algorithms and functions
    - Executable and easy to analyze and verify
  - SystemC became the most popular language

SystemC

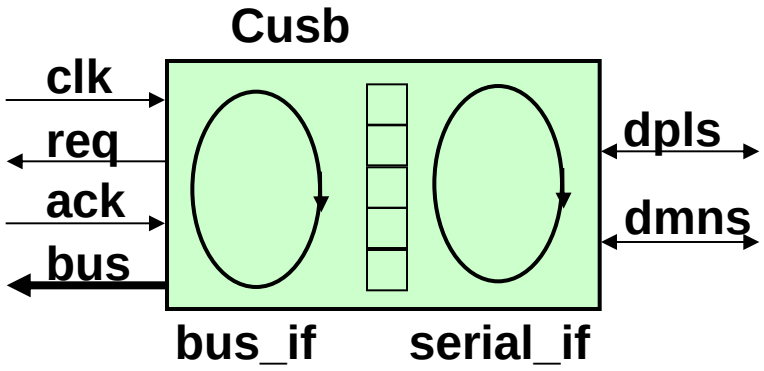
- C++ class library
- Easy to describe both hardware and software behavior
  - Some new features added for hardware description:
    - Hardware data types such as four-value type (0/1/X/Z)
    - Concurrent behavior and timing
  - Concise expression
    - Number of lines will be reduced to about 1/10 of RTL
  - Easy to adopt higher and different abstraction levels
    - Transaction level with both untimed and timed, and cycle accurate
  - Easy to adopt object oriented methods, such as inheritance
- High-speed simulation
  - Transaction level description is 10x to 100x faster than RTL
- Free development environment
  - OSCI reference simulator and GNU
- Standard language
  - OSCI (Open SystemC Initiative) defines Language References
  - Standardized as IEEE Std. 1666-2005

(IEEE: The Institute of Electrical and Electronics Engineers)

  - Many EDA vendors, including Synopsys, Cadence and Mentor Graphics, support SystemC

An example of SystemC model

```
class Cusb : public sc_module{
public:
  sc_inout<bool>   dpls, dmns;
  sc_out  <bool>   req;
  sc_in   <bool>   clk, ack;
  sc_out  <sc_uint<32> > bus;
  Cusb(sc_module_name n) : sc_module(n)
  {
    SC_THREAD(serial_if);
    sensitive << dpls, dmns;

    SC_METHOD(bus_if);
    sensitive << clk.pos();
  ...
};

void Cusb::serial_if(){
  buffer[index++] = dpls && dmns;
  ...
}

void Cusb::bus_if(){
  if( index > 0 ){
  req = 1;
  while( ack.read() == 0 ) wait();
  ...
```



Port declaration

Specify bit width

Function call condition

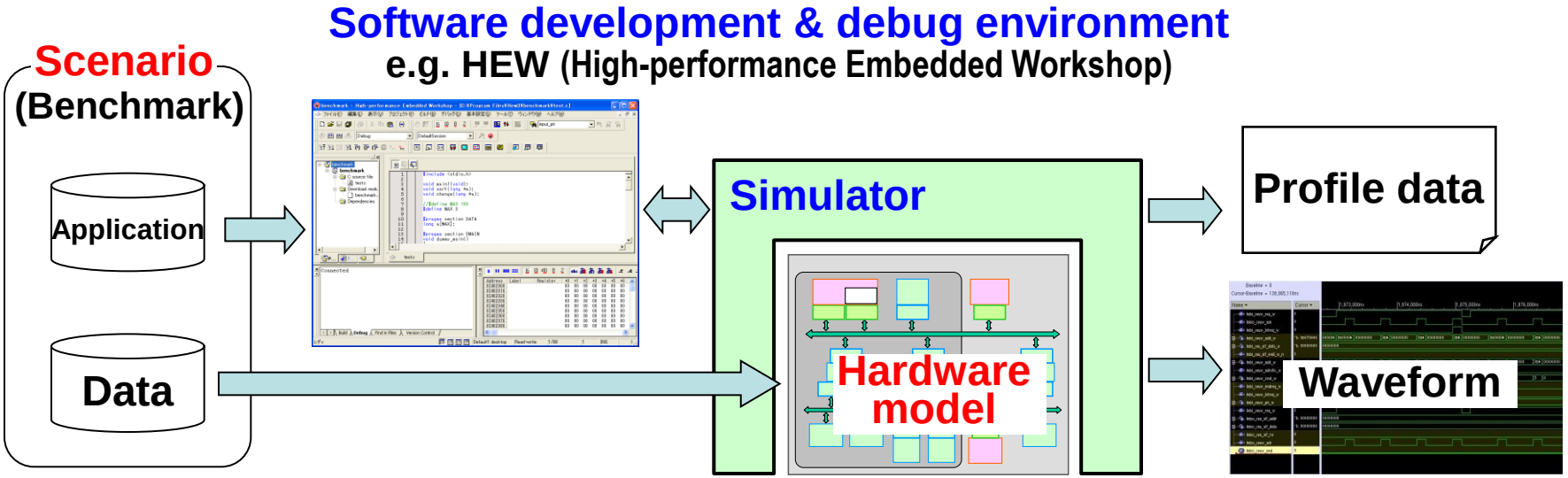
Serial I/F func

LSI bus I/F func

Parallel execution

General Structure of System-level Design Platform

- Co-simulation environment with hardware models and scenarios
  - Hardware models
    - Express functions and timing according to the evaluation objectives
    - Contain both internal components of LSI and peripheral devices
  - Scenario
    - Realistic application programs and data (represent real use cases)

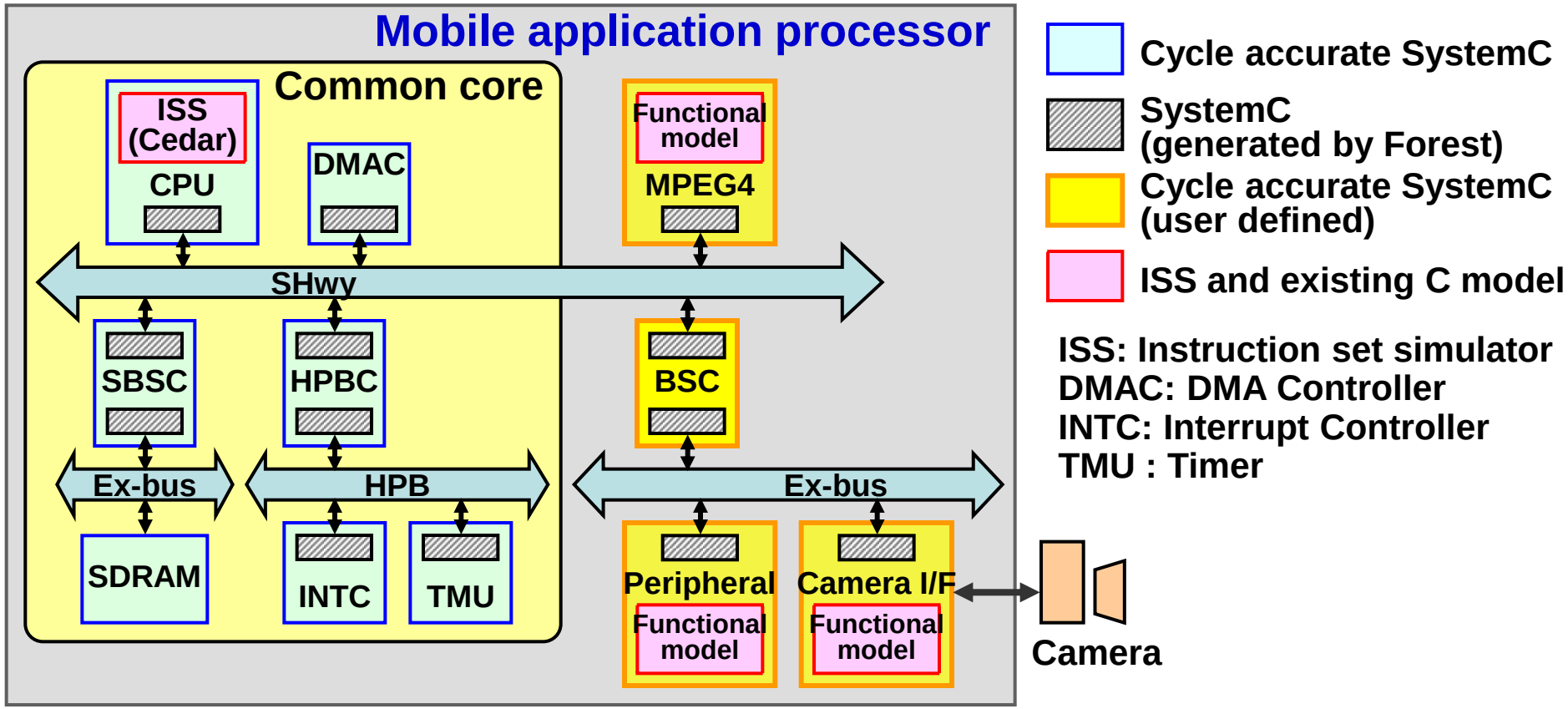


Forest: A System-level Design Platform

- **Proprietary**
  - **Independent from commercial tools**
    - **Runs on the OSCI reference simulator**
  - **Easy to develop models**
    - **Very flexible and easy to incorporate peripheral models**
    - **Automatically generate INTC model and driver software**
  - **High-speed simulation using abstracted bus I/F**
    - **Transaction-level bus I/F**
    - **Untimed models for software development**
  - **Various profile data**
    - **Easy to find and analyze performance bottlenecks**
  - **Support many types of CPUs and buses**
    - **SH-4A, SH-2A, SHwy, Kakadu, etc.**
- **Use commercial tools when necessary**
  - **For example;**
    - **Contain ARM processors and buses**
    - **Very high-speed models of CPUs are necessary**
    - **Customer's request**
  - **Synopsys (former CoWare) / Platform Architect and some other tools are popular**

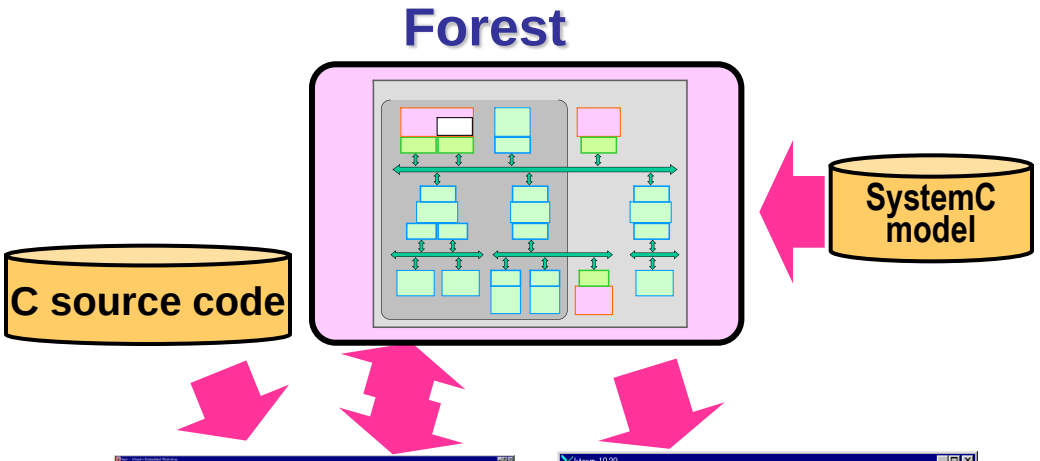
Example of System-level Simulation Environment Using Forest

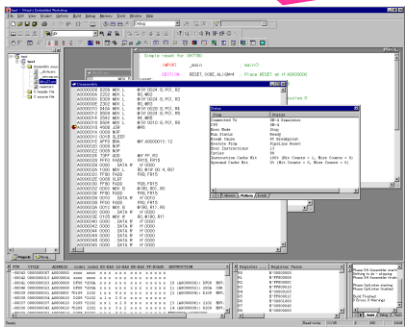
- Forest consists of ISS and common IP models including timer, interrupt controller, buses, etc.
- Provide bus I/Fs (APIs) to incorporate other necessary peripheral models written in C



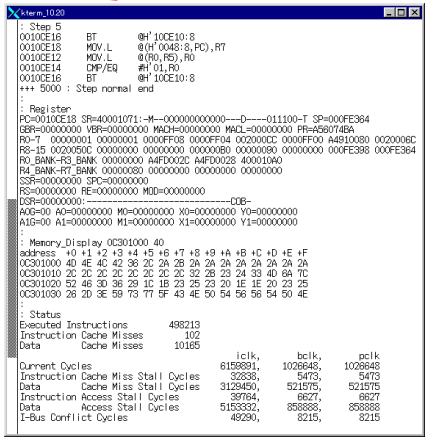
Hardware / Software Co-simulation and Performance Evaluation Flow

- Similar to common software development and optimization
- Hardware optimization or additional hardware modeling may be necessary depending on the situation





HEW



Profile data

Set performance goal

Compile

Simulate

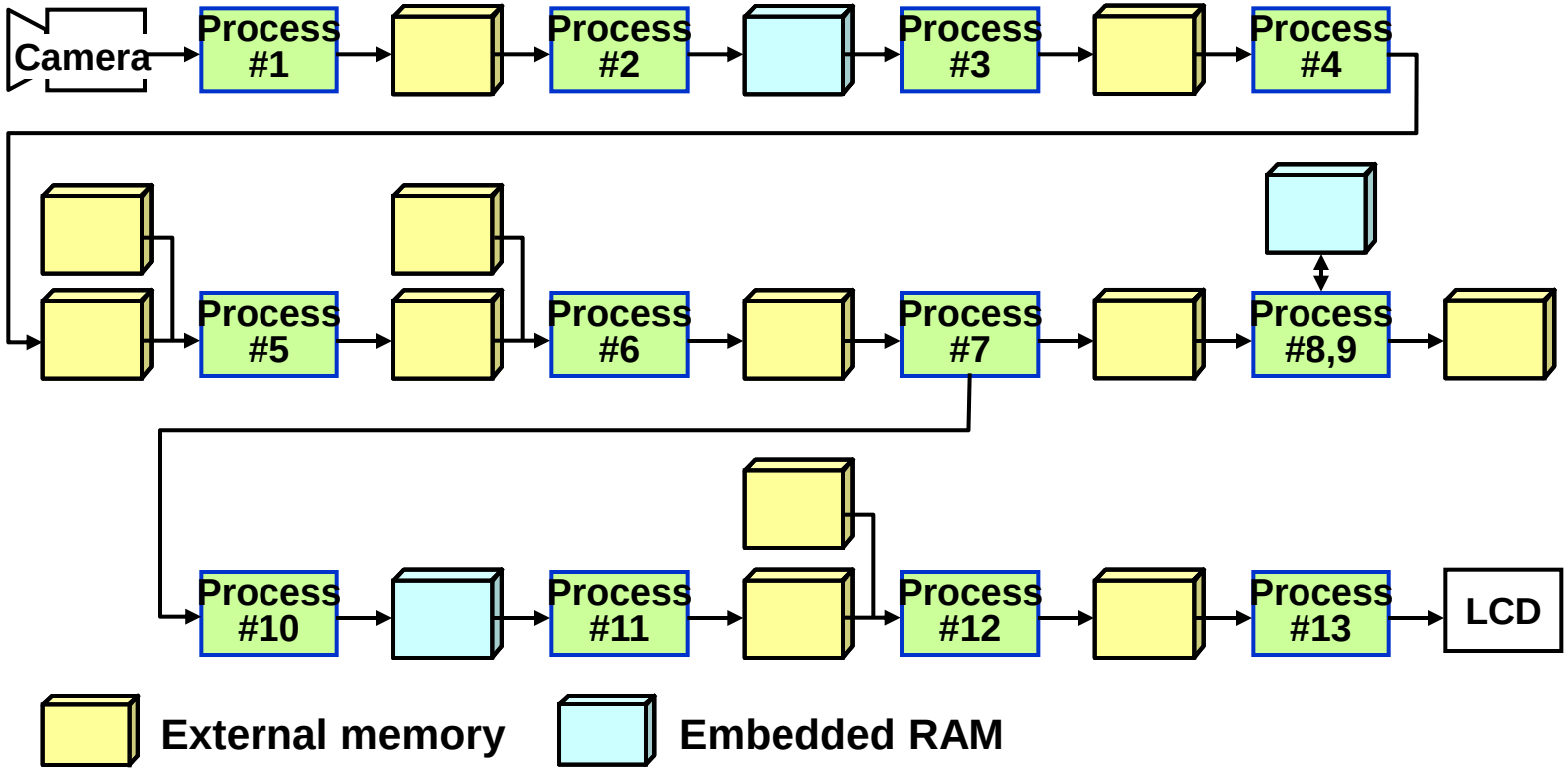
Obtain profile data

Performance analysis
(extract bottle neck)
(1) SW optimization (data struct., algorithm)
(2) HW parameter tuning
(3) HW trial for critical function

HW/SW partitioning, C modeling for HW part

Example of System-level Design (2-1)

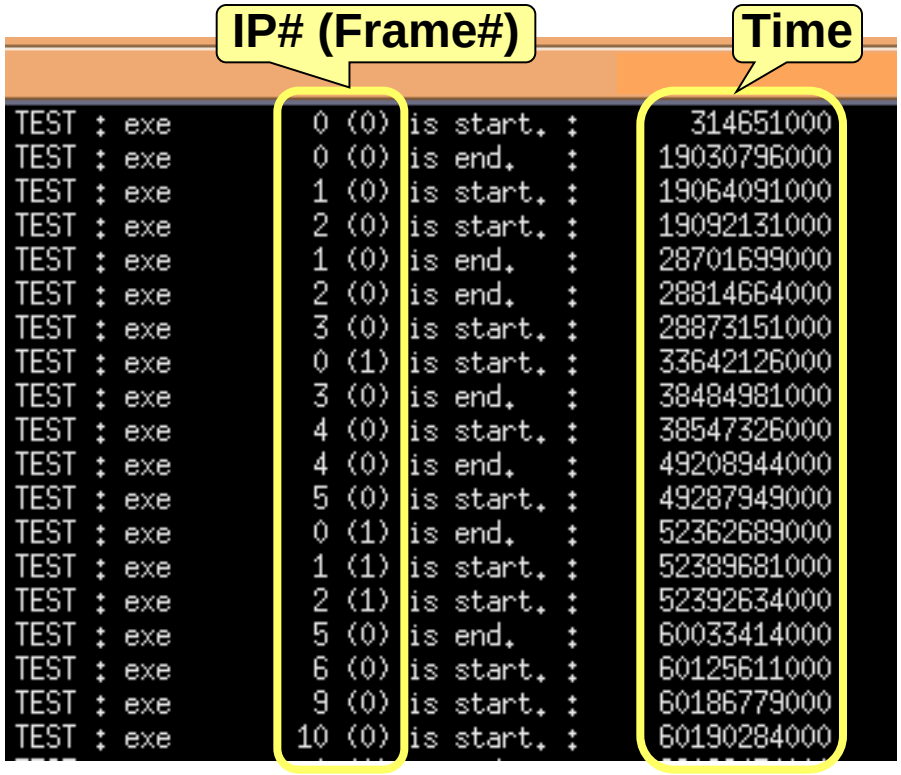
- Performance evaluation of very complicated process; motion picture recording
- It takes several days or weeks to simulate the scenario with Verilog description



Each box contains intermediate motion picture with different format and size
Processes are; enlargement, rotation, composition, conversion, etc.

Example of System-level Design (2-2)

- Performance evaluation of motion picture recording
  - Performance goal: 30 frames/sec i.e. less than 33 ms/frame
- Make graphical charts from the profiled data, then investigate performance bottlenecks or re-evaluate countermeasures for the problems if necessary

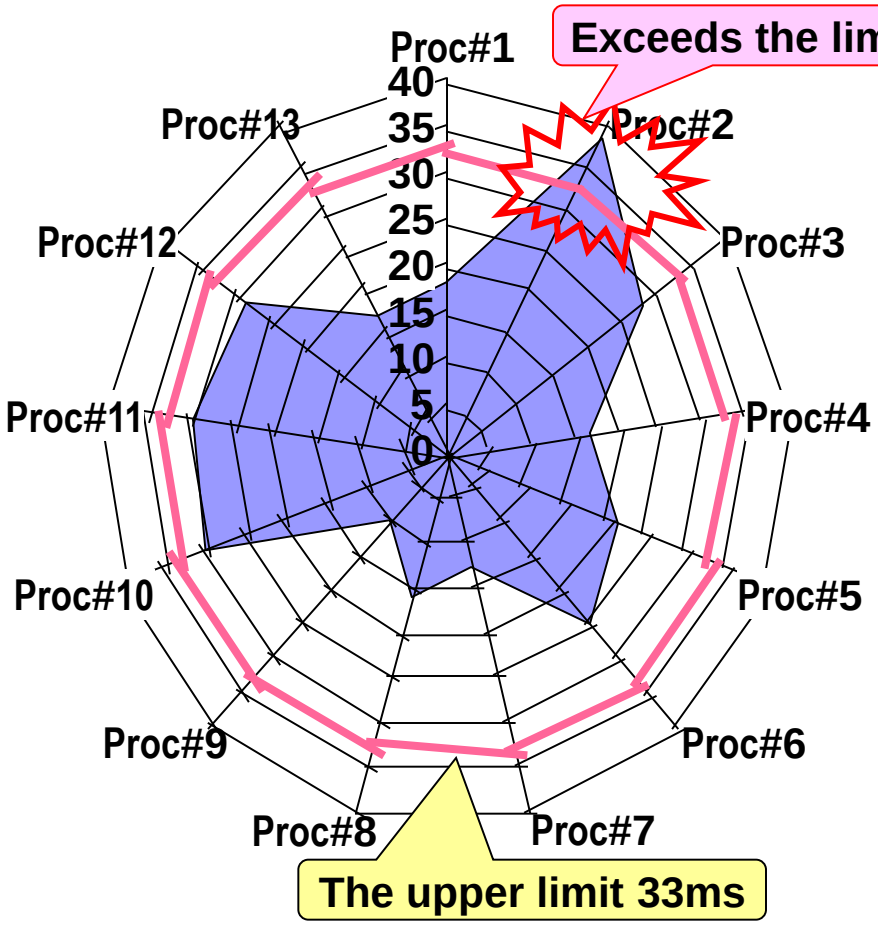


The profile data is very concise and reduces the size of disk space to 1/100

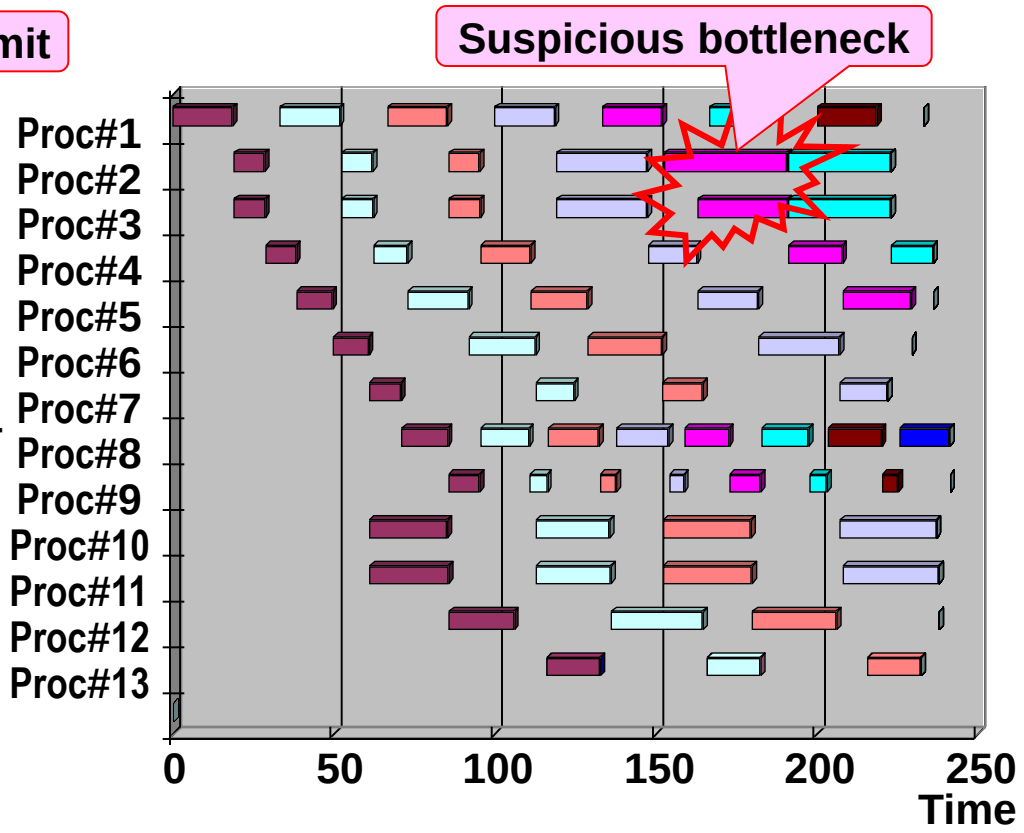
Example of System-level Design (2-3)

- Detect performance bottlenecks using graphical charts

Time required for each process



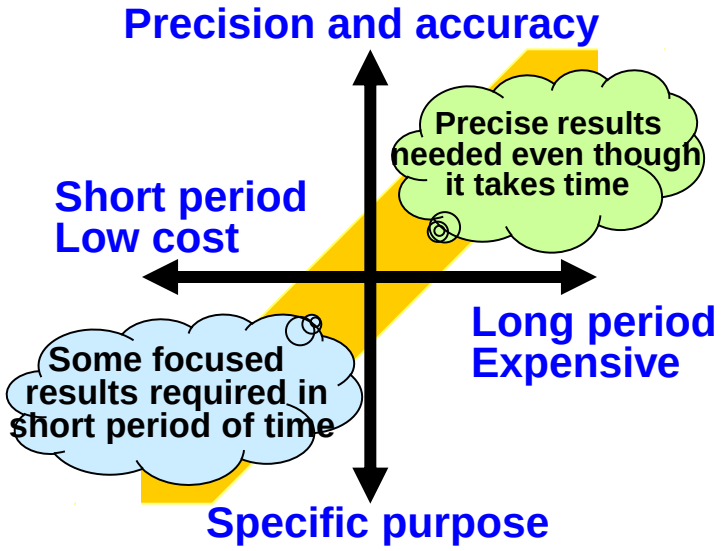
Execution time of each process

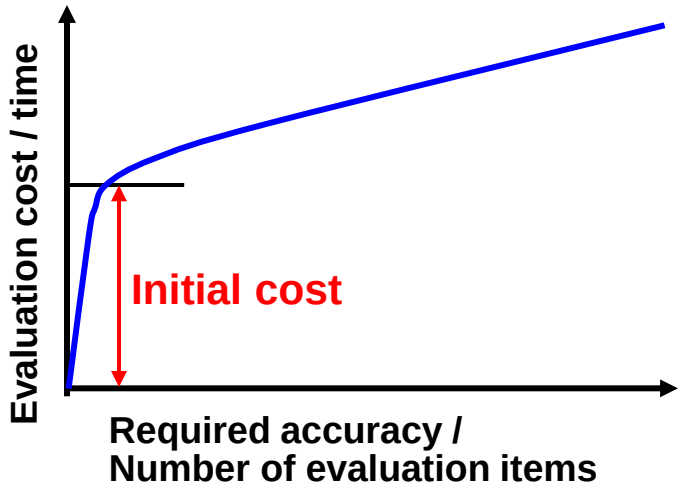


Another Design Platform for Performance Evaluation

- Problems for System-level Evaluation
  - A great quantity of cost and time required for model development
    - Accurate evaluation such as running real software on complete hardware model takes too much time and cost
- There exists a trade-off between accuracy and cost / time
- Even if only a few items are being evaluated, considerable initial cost and time necessary

- Challenge: Realize a very short TAT (turn-around-time) and low cost evaluation method
  - No model developments necessary to start evaluation
  - Take a few days or a week





Two Types of Performance Evaluation Environment

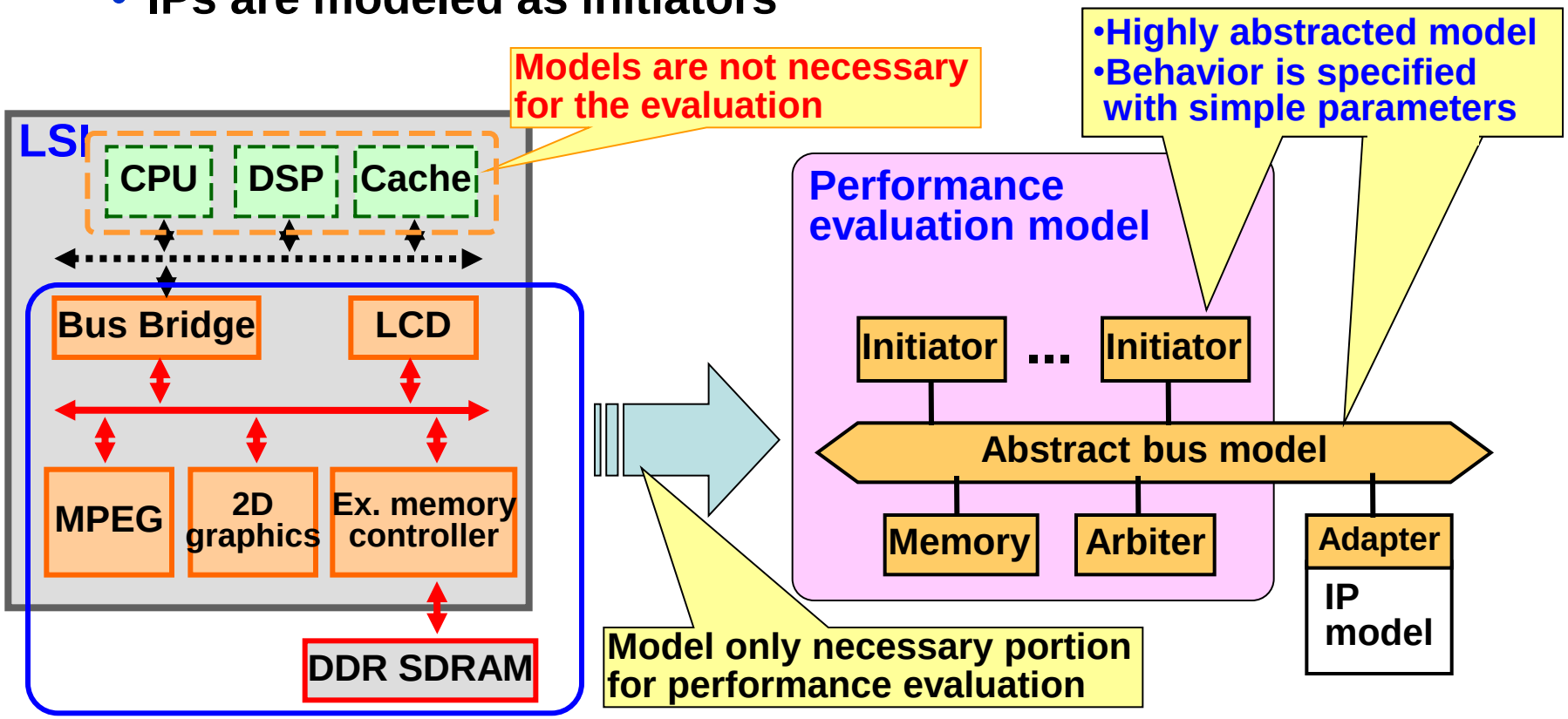
- **Two different types of evaluation environment are used in Renesas for accuracy and cost trade-off**
  - **Use proper one in accordance with the evaluation purpose**

Model	Purpose	Structure	Renesas environment
Integrated evaluation environment	•Precise performance and power evaluation with real application and data •Hardware / software co-evaluation	•ISS (instruction set simulator) •Functional timed models of buses and peripheral IPs	Forest
Directed performance evaluation model	•Performance evaluation focusing on particular bottle-necks; bus traffic and memory access •Very short TAT evaluation and minimum cost	•Transaction level bus models •Abstracted IP models (i.e. initiator models)	RESL

- New evaluation platform (RESL-X) are being developed
  - Take the place of both Forest and RESL
  - Much easier than RESL
  - Keep up with the latest bus and memory controllers

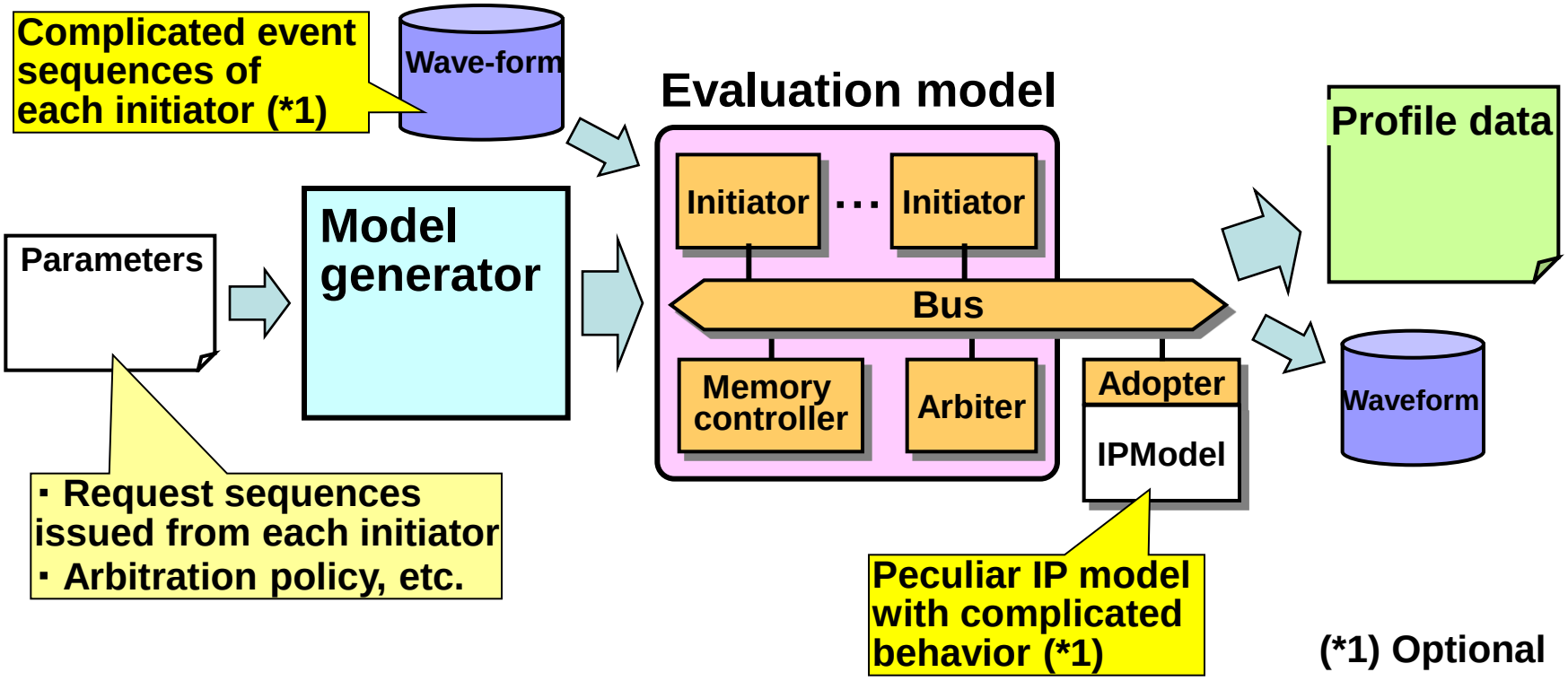
Directed Performance Evaluation Environment (RESL)

- **Focus attention on particular performance bottleneck**
  - Internal bus, external memory controller, etc.
- **Develop limited models and minimize cost and time**
  - Necessary part of LSI
  - IPs are modeled as initiators



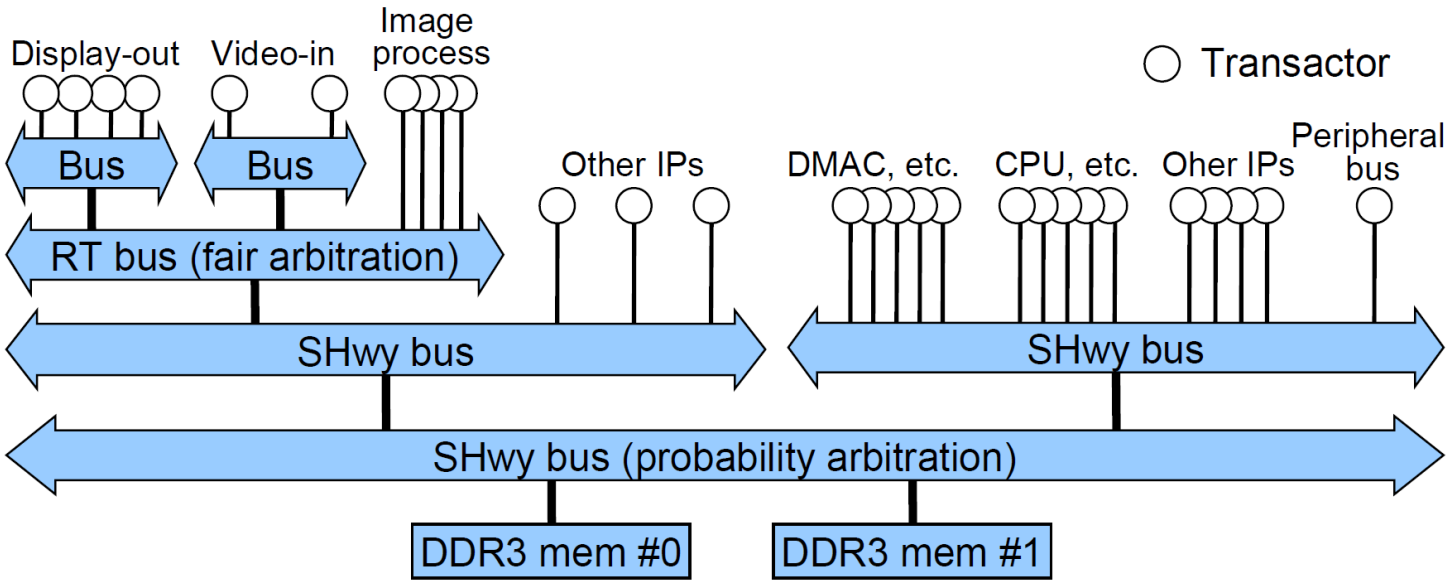
RESL Model Generator

- Performance evaluation **model is generated automatically by RESL** so that no model development required for general performance evaluation
  - **Minimize the initial cost** for evaluation
  - Start objective evaluation **in one or a few days**



Performance Evaluation of Navi SoC

- **Purpose**
  - Performance verification at worst condition (many requests at one time, always cause bank and cache miss)
- **Configuration**
  - Complex bus structure and so many peripheral IPs
  - Simple transactor models that issue any requests (read/write, etc.)
  - Precise and accurate models for buses and memory controllers
- **Results**
  - Reduce time for evaluation: 0.5~1day/case (Excel) → minutes/case



Summary

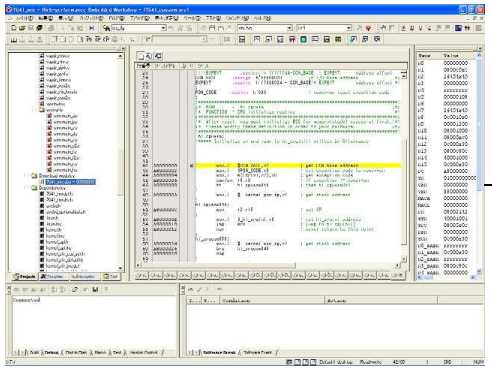
- The purpose of system-level design is to reduce total development cost for both hardware and software
  - Adopt performance evaluation and optimize system architecture
  - Detect any design problems in earlier phase of the design and prevent re-design or re-manufacturing
- During system-level design, clarifying the objectives is very important to get desired evaluation results within the time limit
  - Use appropriate abstraction level according to the objectives
- RESL is an automated environment that enables very short TAT performance evaluation
  - It is not consists of functional models of peripherals but initiator models

3. Virtual Platform for Embedded Software Development

Conventional Software Test and Debug Environment

- Reference board that is equipped with real LSIs is used

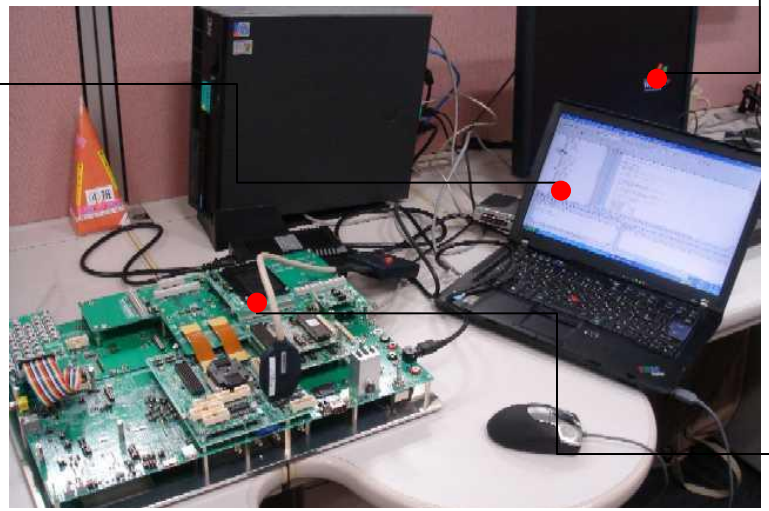
Software debugger (HEW)



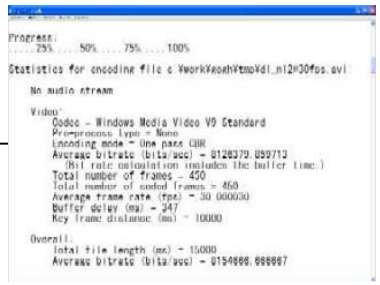
GUI
- Friendly interface
- Easy to observe

C source debug
- Easy to understand

※ HEW: High-performance Embedded Workshop (Integrated environment for SH software development)



Terminal



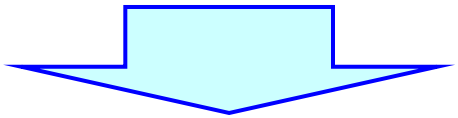
Display logs

LCD panel

Display graphics

Problems of Software Test and Debug Using Real SoC

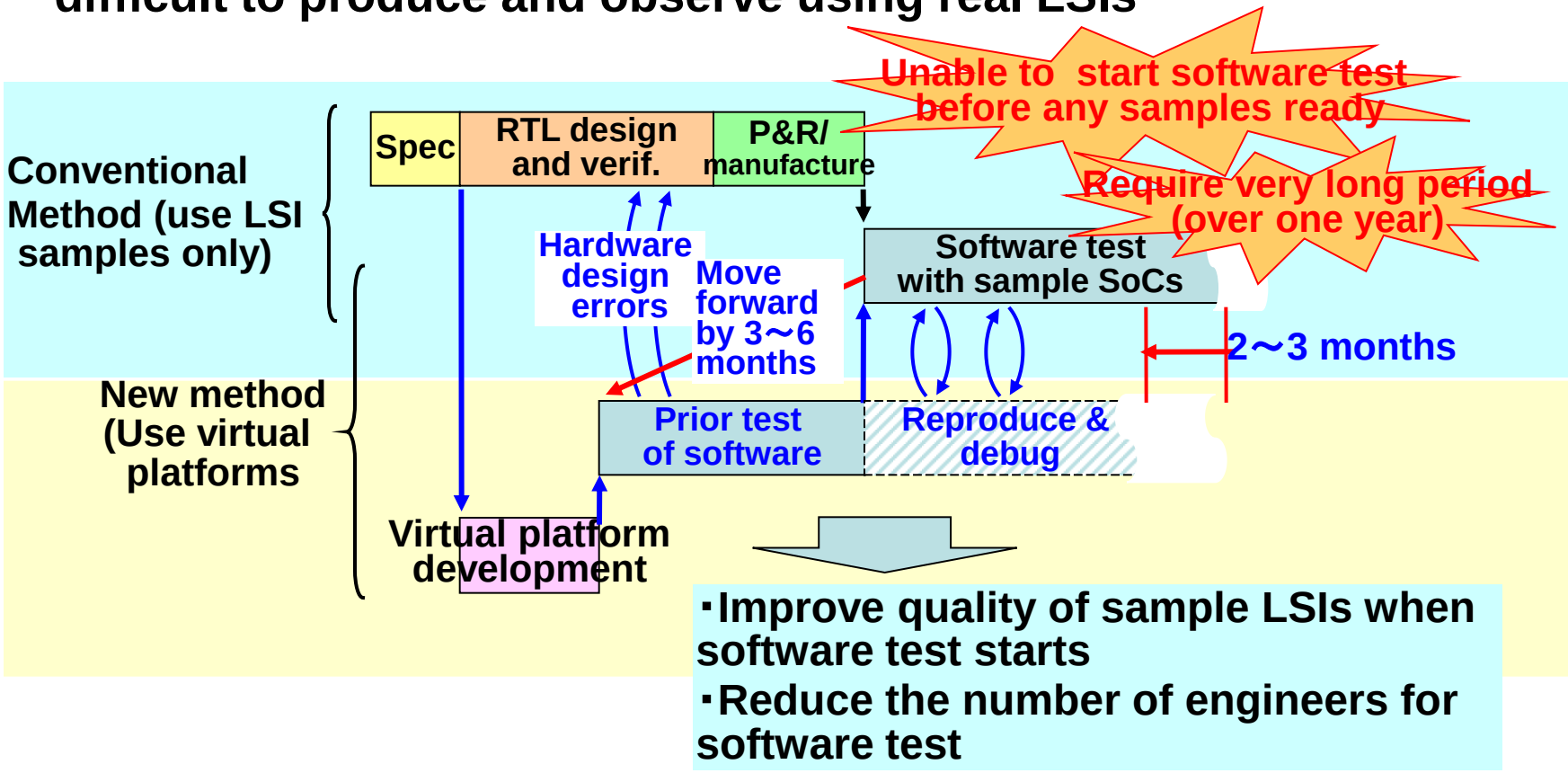
- **Unable to start test and debug before a real SoC gets ready**
  - Usually, firmware, driver, OS and middleware are tested step by step (accumulation manner) so that it takes very long time
  - Customers expect Renesas to release these software just after hardware becomes available
- **Test and debug features are not efficient enough**
  - Unable to observe signals and memories inside LSI
  - Difficult to create exceptional condition
  - Difficult to reproduce occasional abnormal phenomena



- **Virtual platform becomes popular**
  - Start software test and debug earlier than real LSIs get ready
  - Test and debug more efficiently

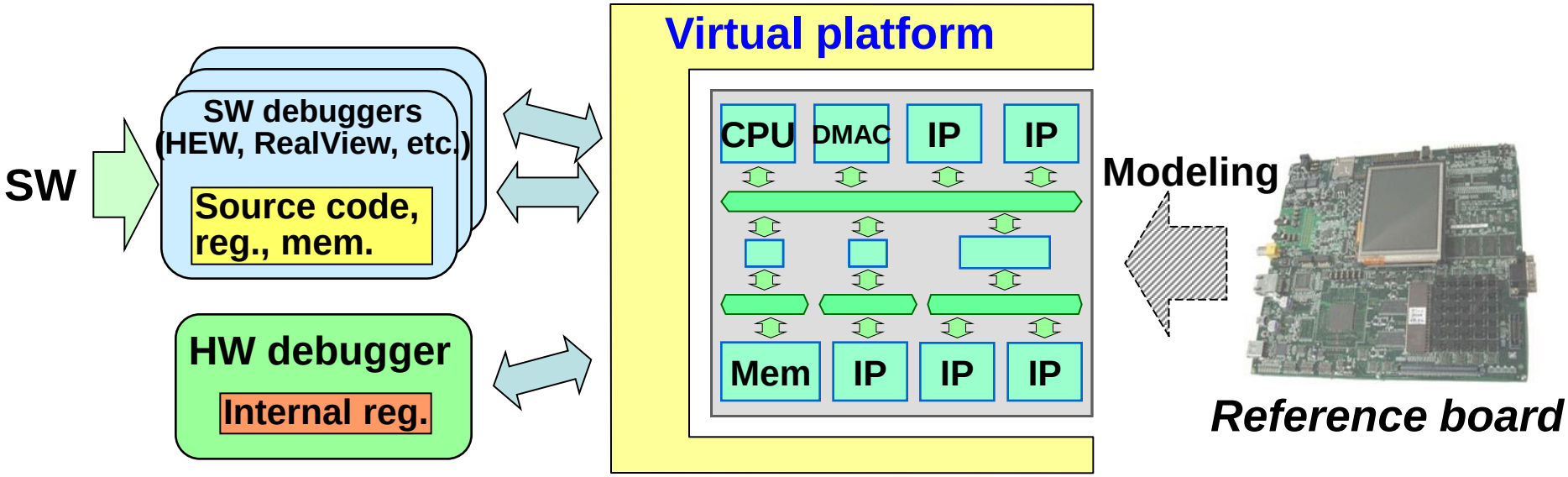
Merits of Virtual Platform for Embedded Software Development

- Start software test before LSI samples become available
- Detect specification errors and improve the quality of LSI samples by means of software / hardware co-verification
- Facilitate test and debug for exceptional conditions which are difficult to produce and observe using real LSIs



Basic Structure of Virtual Platform

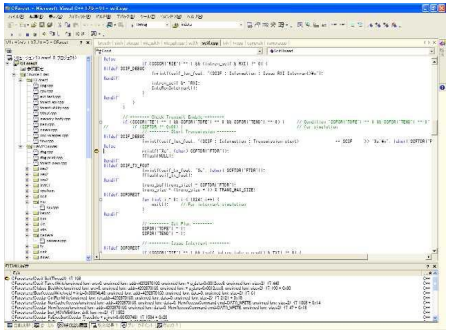
- Virtual platform consists of hardware models and software / hardware debugger
  - Hardware model
    - CPUs, peripheral IPs and external devices
    - Use either untimed models without bus for high-speed simulation or transaction-level models including bus for accurate debugging
  - Software debugger
    - Same debuggers that software engineers use with real LSIs
  - Hardware debugger
    - More efficient than ones incorporated with real LSIs



Software Test and Debug Environment Using Virtual Platform

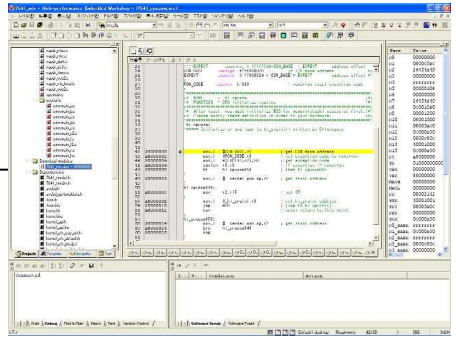
- Realize same test and debug environment on a PC just like a reference board

HW debugger (Visual C)



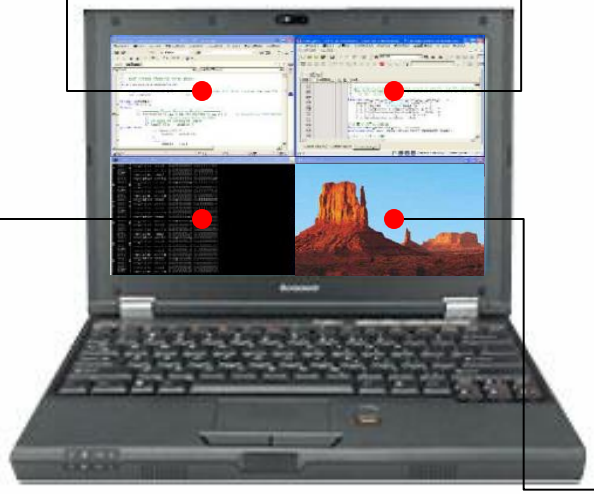
Display HW status
HW model development

SW debugger (HEW)



GUI, C source

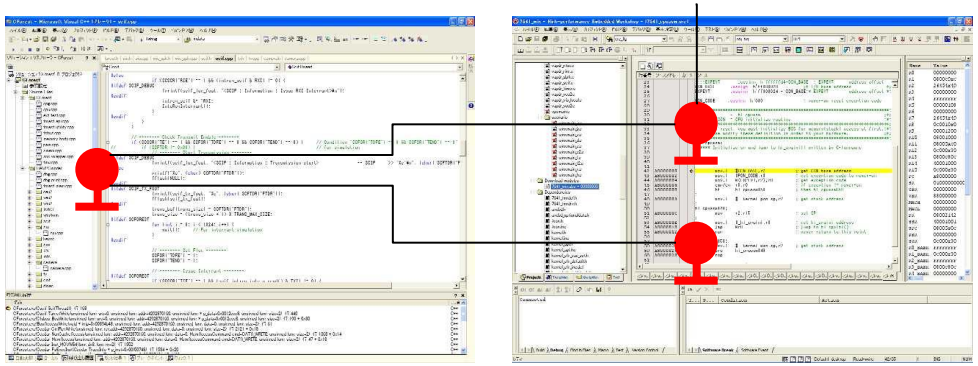
Terminal



Graphic viewer

Features of Virtual Platform

- Same functions, same environment for software test and debug as a reference board
- **High-speed simulation**
  - Around 50 MIPS (million instructions per second) for CPU core
  - Several ~ 10 MIPS including peripheral modules
  - **Around 1/10 ~ 1/100 of real hardware's speed**
- **Convenient Debug Features**
  - All signals and memories are observable
  - Execute simulation stepwise or clockwise
  - Set breakpoints on both hardware and software codes
  - **Synchronize software and hardware**, and multiple CPU cores
  - Easy to produce any **exceptional or occasional error conditions**
  - Save simulation status to restore and resume
  - Construct **complicated test environment** with bunch of terminals and base stations



Hardware debugger (Visual C++)

Software debugger (HEW, RVD, etc.)

Virtual Platform of Automotive Micro-controller

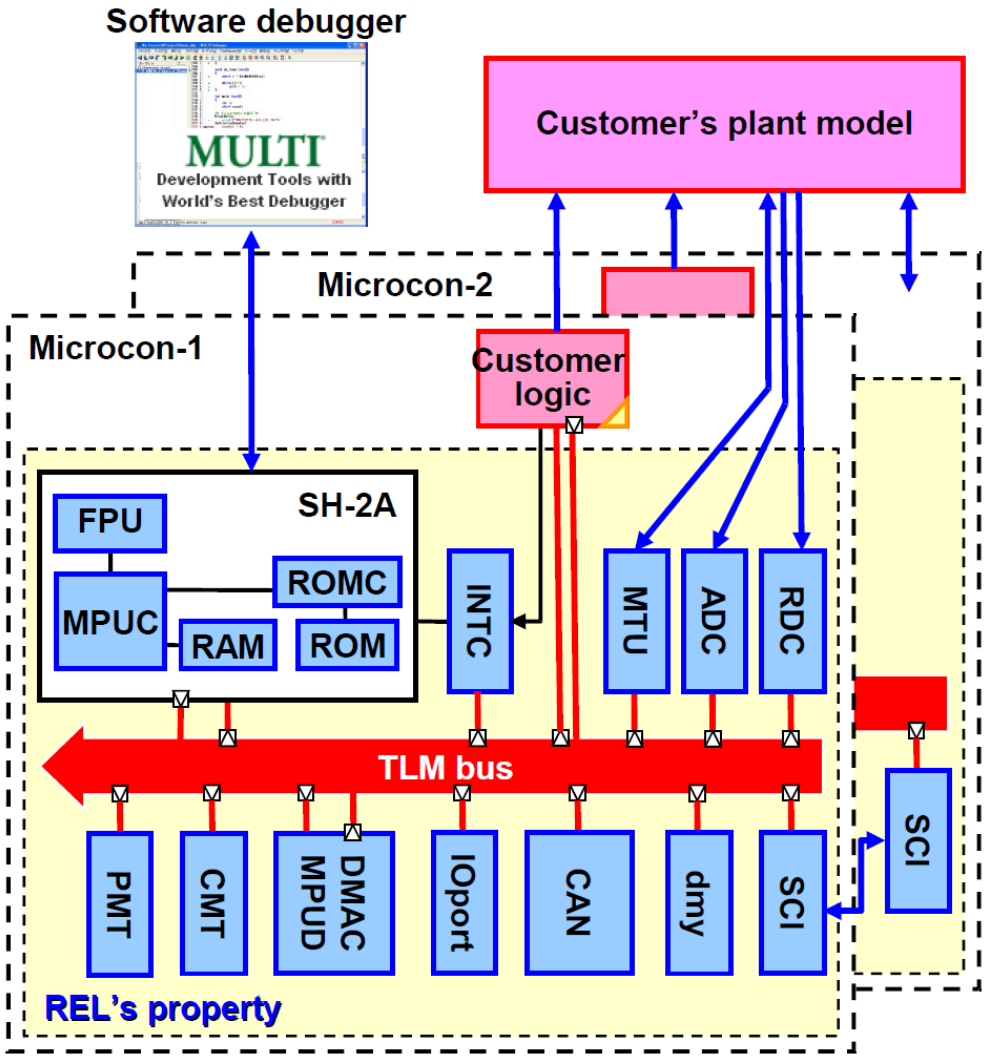
Purpose

- Early software development
- Fault injection for functional safety

Configuration

Dual SH-2A ISS

- High-speed simulation with untimed models
- Plant model I/F
- 3rd vendor's debugger



Virtual Platform of Mobile SoC

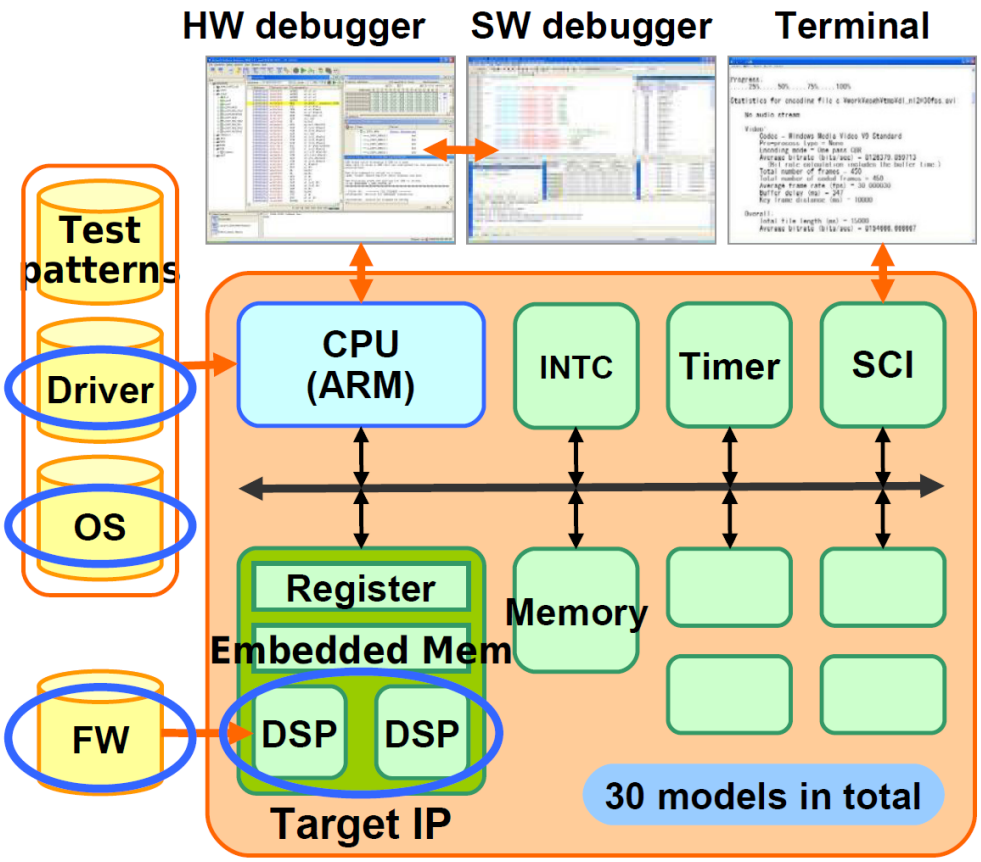
Purpose

- Develop driver software for new IP before real chip available.

=> Run DSP driver and firmware on target OS (Symbian)

Configuration

- CoWare(now Synopsis)/ Platform Architecture.
- ARM/Fast model.
- NXP/DSP model.
- TLM peripheral models made by REL.

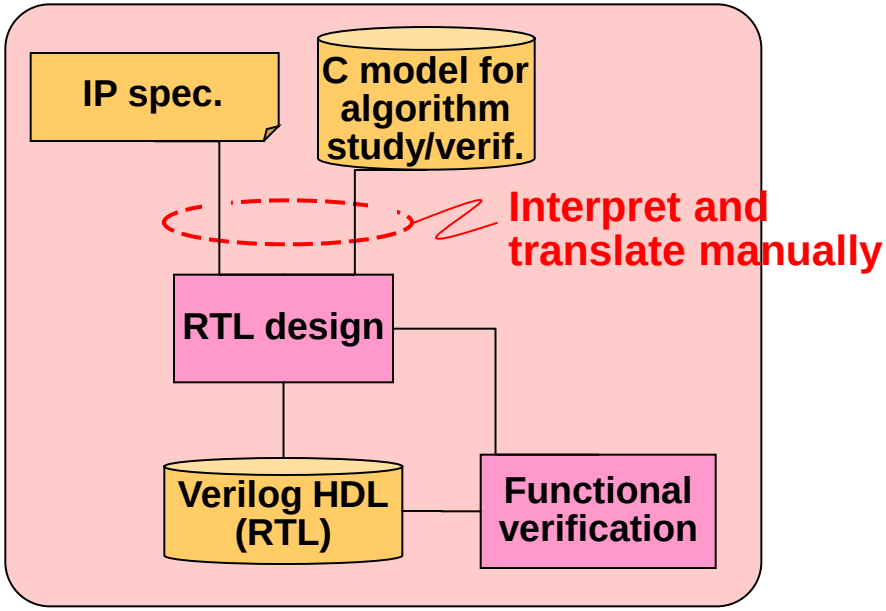


4.High-level Design

Problems of RTL Design Flow

- **Number of lines of design description increases as LSIs get larger and more complicated**
  - **Simulation becomes slower and simulation time is roughly in proportion to square of design size**
  - **Difficult to verify, debug, modify or maintain the large design**

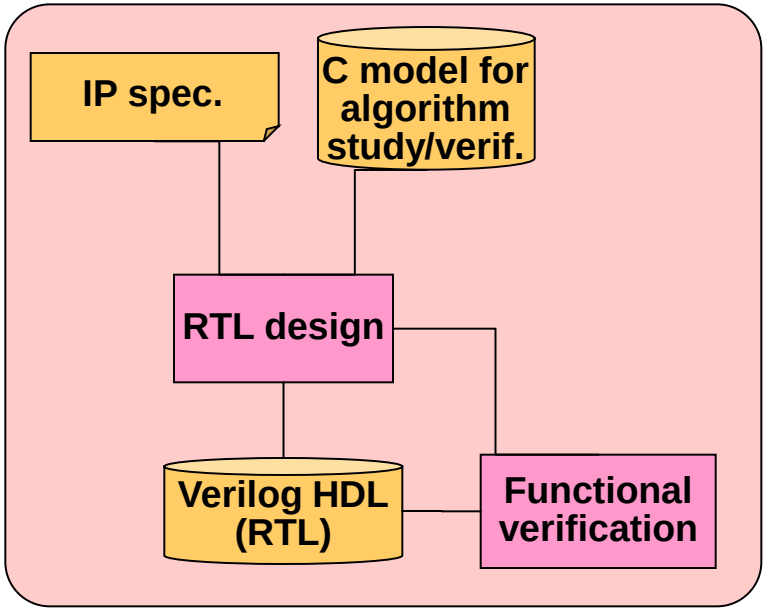
Conventional design flow



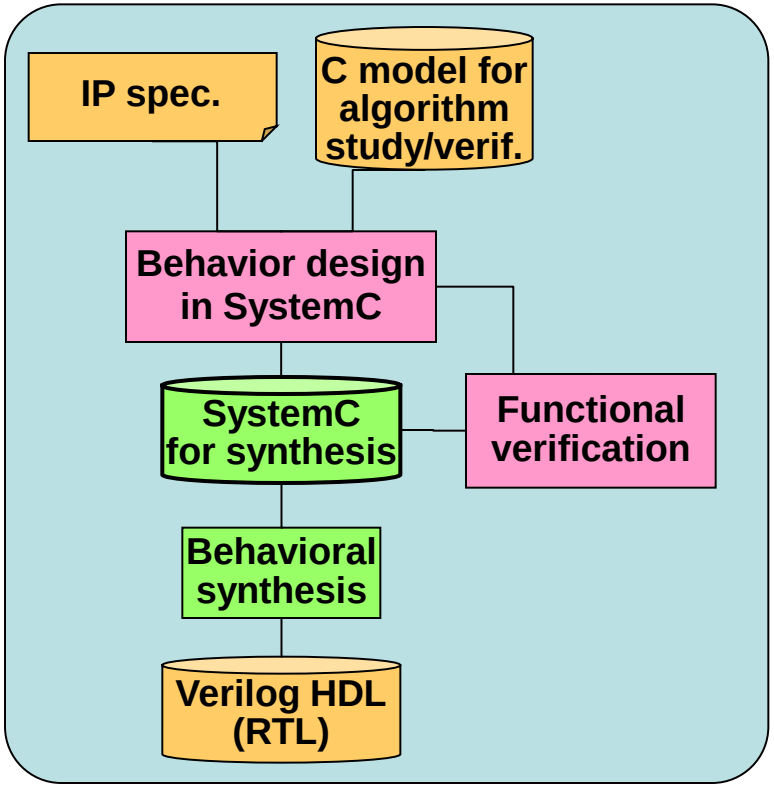
Solution with High-level Design Flow

- Design hardware at a **higher level of abstraction**
  - Use **SystemC** as a high-level design language and **reduce number of lines of design description**
  - Incorporate new **behavioral synthesis technology** and **reduce human effort to design in detail**

Conventional design flow



High-level design flow



Aims of High-level Design (1-1)

- Many codes for such as declarations and control logics are added automatically to the SystemC description in Behavioral Synthesis

SystemC

```
void dut::proc0(){
  unsigned char a_in;
  unsigned char b_in;
  unsigned char c_in;
  unsigned char d_in;
  unsigned char e_in;
  unsigned long long y_out;

  y.write(0); // reset output
  wait();

  while(1){
  a_in = a.read();
  b_in = b.read();
  c_in = c.read();
  d_in = d.read();
  e_in = e.read();

  y_out = ((a_in*b_in)+c_in)*(d_in*e_in);

  y.write(y_out);
  wait();
  }
}
```

20 lines

Added

Detailed

Verilog (RTL)

```
reg [1:0] state;
reg [1:0] state_next;
reg [15:0] mul0_in0;
reg [16:0] mul0_in1;
reg [32:0] mul_0;
reg [16:0] add_0;
reg [15:0] t1_t2;
reg [16:0] t3;
reg [32:0] y_reg;
reg [15:0] t1_t2_reg;
reg [16:0] t3_reg;
```

Internal variables

```
always @(posedge clk or negedge rst_n )
  begin
    if (!rst_n)
    begin
      state <= 2'b0;
    end
    else
    begin
      state <= state_next;
    end
  end

always @(*) begin : dut_state
    if( state == 2 ) state_next = 0;
    else state_next = state + 1;
end
```

Control logic (FSM)

```
always @(posedge clk or negedge rst_n )
  begin
    if (!rst_n)
    begin
      y <= 33'b0;
      t1_t2 <= 16'b0;
      t3 <= 17'b0;
    end
else
    begin
      y <= y_reg;
      t1_t2 <= t1_t2_reg;
      t3 <= t3_reg;
    end
end
```

Flip flops

```
always @(*) begin : dut_combinational
  case(state)
    2'b00: begin
          mul0_in0 = {8'b0,a};
          mul0_in1 = {9'b0,b};
          end
    2'b01: begin
          mul0_in0 = {8'b0,d};
          mul0_in1 = {9'b0,e};
          end
    2'b10: begin
          mul0_in0 = t1_t2;
          mul0_in1 = t3;
          end
    2'b11: begin
          mul0_in0 = t1_t2;
          mul0_in1 = t3;
          end
    default: begin
          mul0_in0 = 16'bX;
          mul0_in1 = 17'bX;
          end
  endcase
  mul_0 = mul0_in0 * mul0_in1;
  add_0 = t1_t2_reg + c;
  if( state == 0 ) t1_t2_reg = mul_0;
  else  t1_t2_reg = t1_t2;
  if( state == 1 ) t3_reg = add_0;
  else  t3_reg = t3;
  if( state == 2 ) y_reg = mul_0;
  else  y_reg = y;
end
```

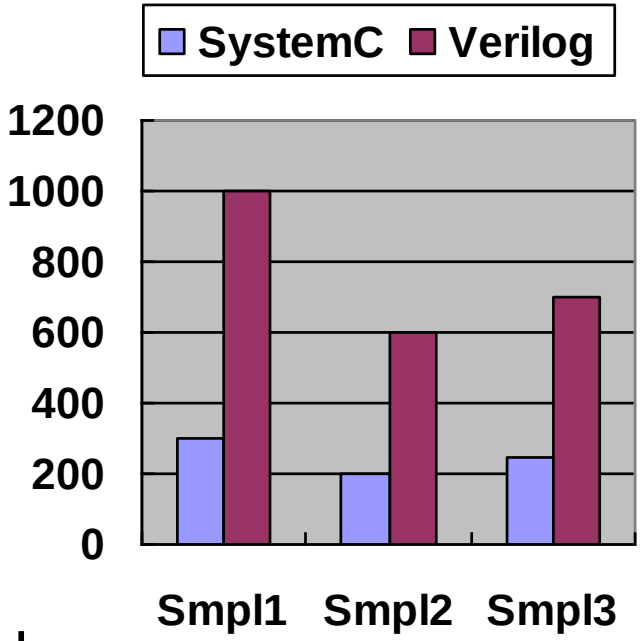
Multiplexor

73 lines

Aims of High-level Design (1-2)

- The number of lines of SystemC codes is around one third of Verilog codes for actual designs

Module	# of Gates	SystemC (lines)	Verilog (lines)	Ratio
Smpl1	50k	300	1,000	3.3
Smpl2	20k	200	600	3.0
Smpl3	5k	250	700	2.8



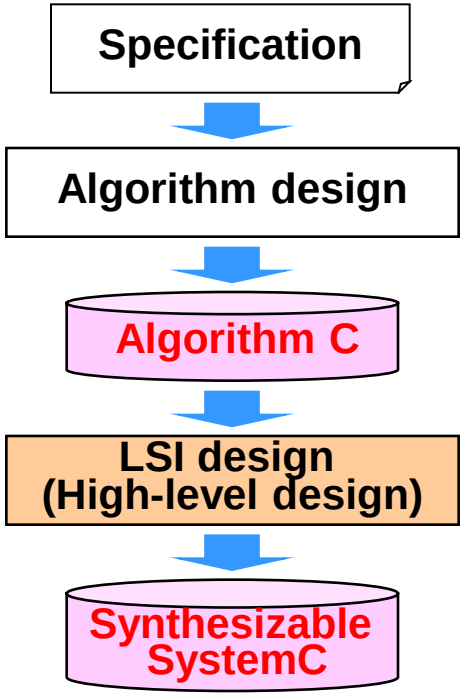
⇒ Number of design errors is reduced

⇒ Easy to correct design errors

Aims of High-level Design (2)

- **In many cases, the algorithm description of the target design, which is written in C/C++, can be used as a part of hardware design written in SystemC**

Module	# of Gates	# of lines of SystemC description	Reused C/C++ description	Reuse ratio
Smpl4	50k	340	110	32%
Smpl5	100k	7,700	5,200	68%

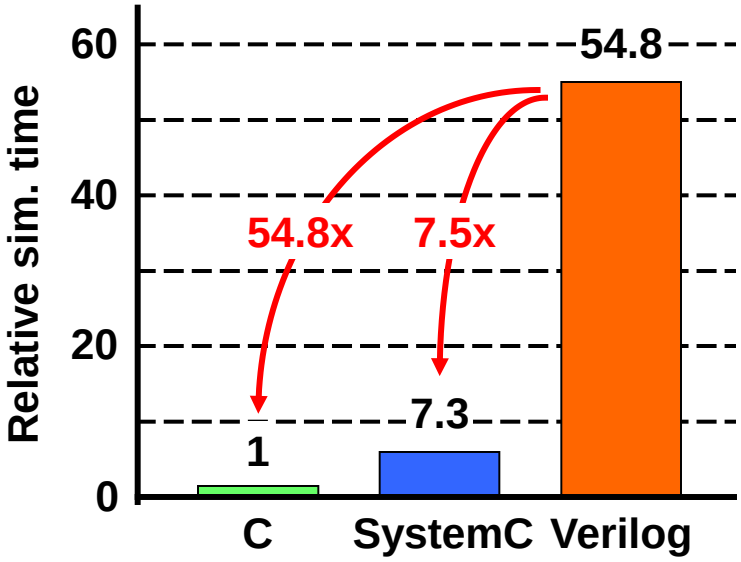


- Algorithm description can not be used directly for conventional Verilog design.

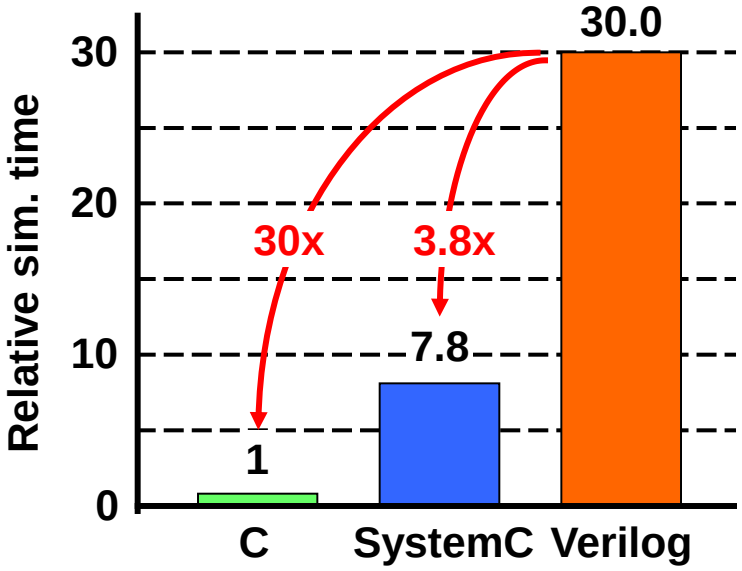
- SystemC is more suitable as larger design and for data processing modules such as motion picture encoding

Aims of High-level Design (3)

- **Simulation speed of C / SystemC description is several tens of times faster than Verilog description**



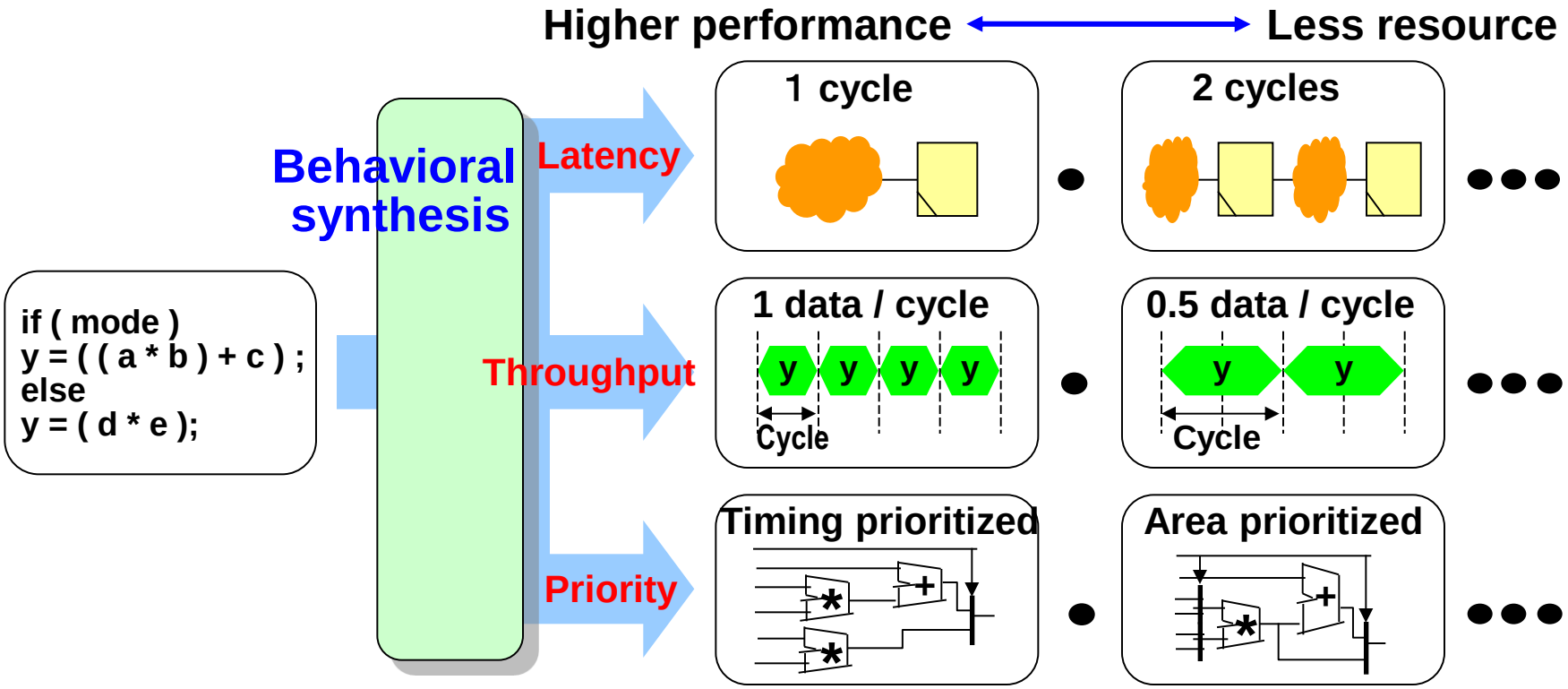
Design sample #1 (100k gates)



Design sample #2 (600k gates)

Aims of High-level Design (4-1)

- It is possible to generate **different architectures or implementations** by using behavioral synthesis
  - **Evaluate and compare many choices** of design in short period
  - **Reuse proven design** and synthesize another architecture that are functionally identical but based on **different technology and cost**, have **different frequency and performance**, and so on.



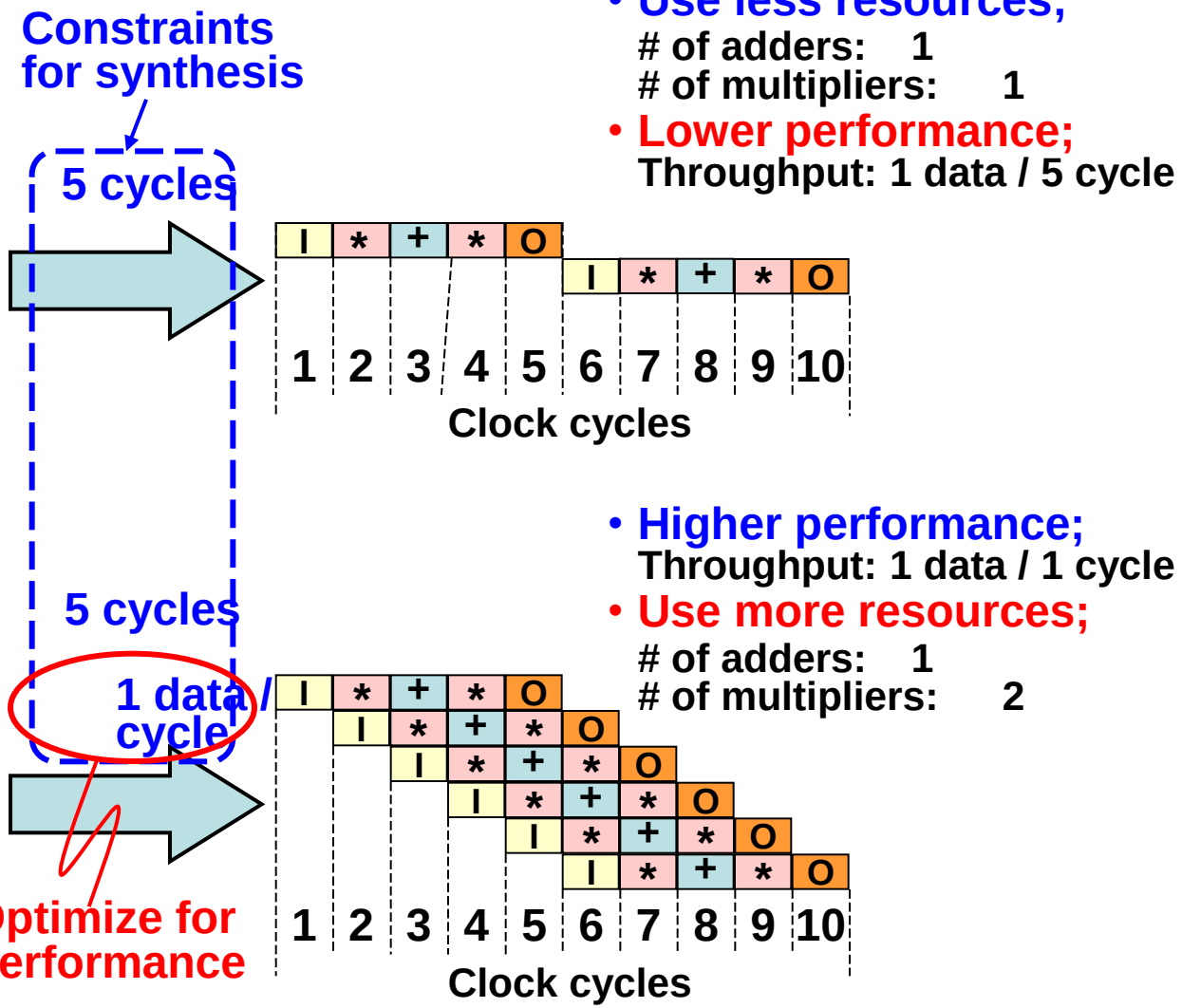
Aims of High-level Design (4-2)

Example of architecture exploration

- Sequential operation

```
while(true){
   a = a_in.read();
   b = b_in.read();
   c = c_in.read();
   d = d_in.read();
   e = ( a * b + c ) * d ;
   out.write(e);
   wait();
}
```

- Pipeline architecture



Aims of High-level Design: Summary

- **Efficient design at a higher abstraction level**
  - Design size, i.e. number of lines of design description, can be reduced to one third of RTL (in Verilog)

⇒ Number of design errors is reduced

⇒ Easy to correct design errors

  - IPs become more flexible at an abstracted level so that they can be reused more efficiently

⇒ Reduce total design effort by using such proven designs

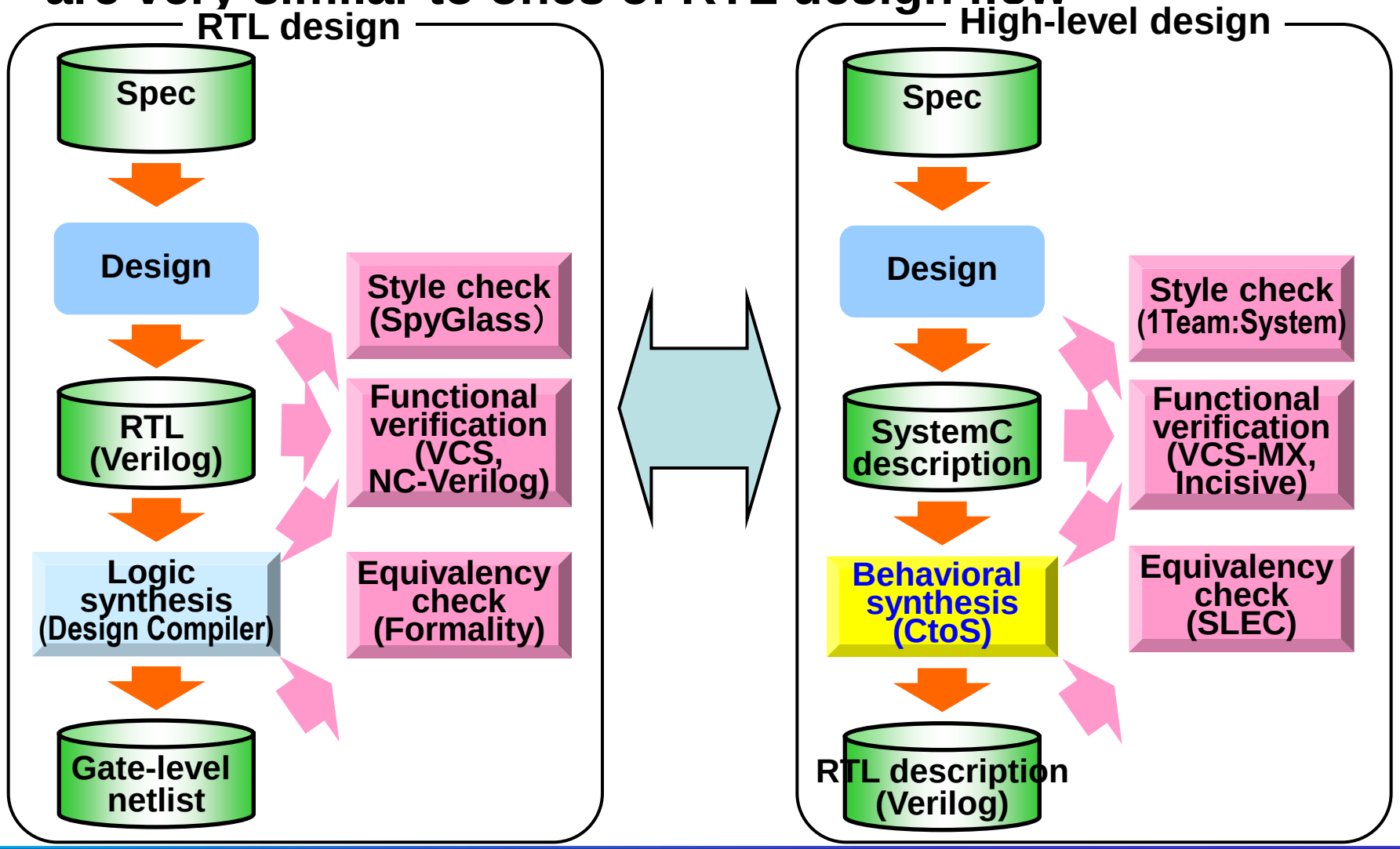
  - Simulation speed becomes about ten times faster

- **Optimize hardware architecture in short period of time**
  - Adopting efficient high-level design and automated synthesis, many design candidates can be evaluated quickly

⇒ It becomes possible to select the best architecture

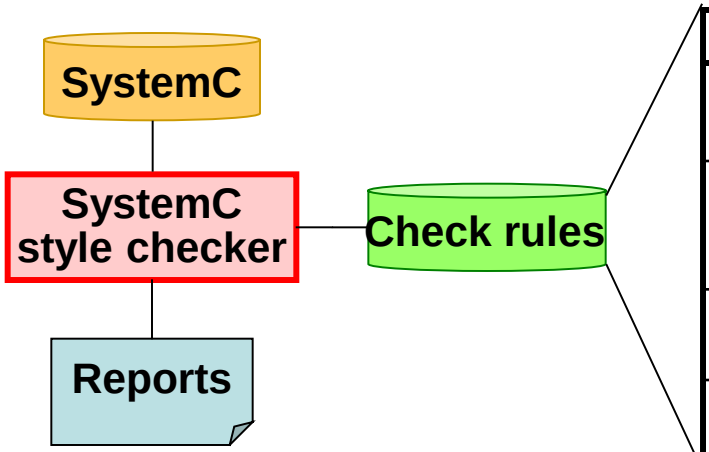
High-level Design Flow

- High-level design flow consists of design steps which are very similar to ones of RTL design flow



SystemC Style Checker

- **Statically check SystemC descriptions to remove unsuitable portion for system-level evaluation, verification and high-level synthesis very quickly**
  - **SystemC descriptions that are not supported by the high-level design flow or the tools**
  - **Error-prone descriptions in SystemC that are difficult to detect by simulation**
  - **SystemC descriptions that may cause mismatch when RTL descriptions synthesized**

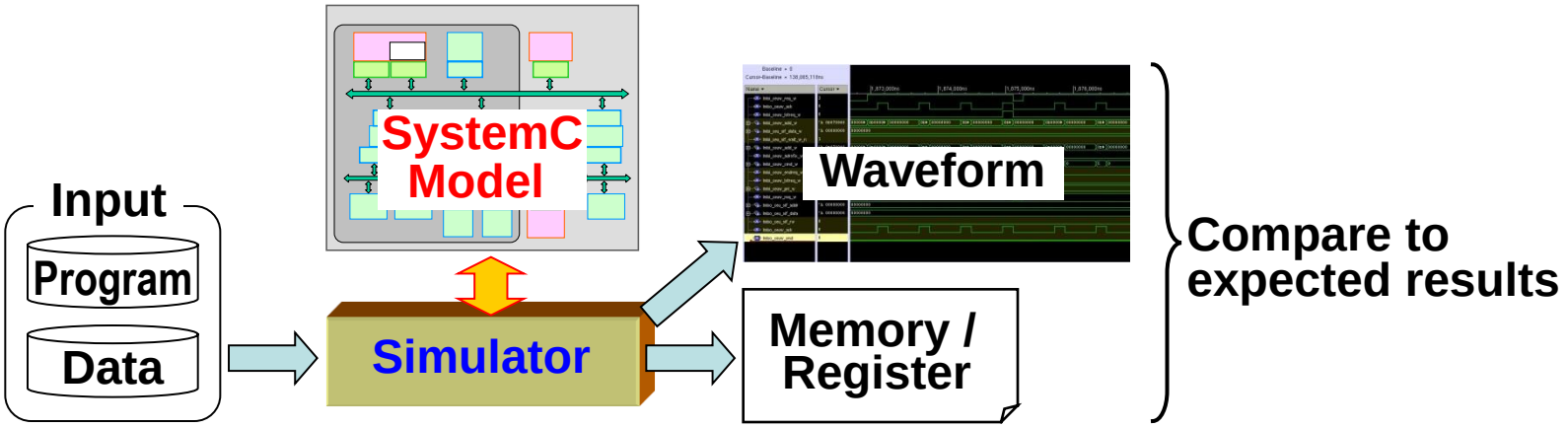


Rule ID	Rule description
C001	No redundancy, deficiency or duplication in sensitivity list of SC_METHOD
C002	No expression or function call which may change the value of any variables in condition predicate of conditional statement
⎮	⎮
C010	Not use SC_THREAD

- Atrenta's 1Team:System is available in RVC
  - Check C, C++, SystemC codes with about 700 embedded rules

SystemC Simulators

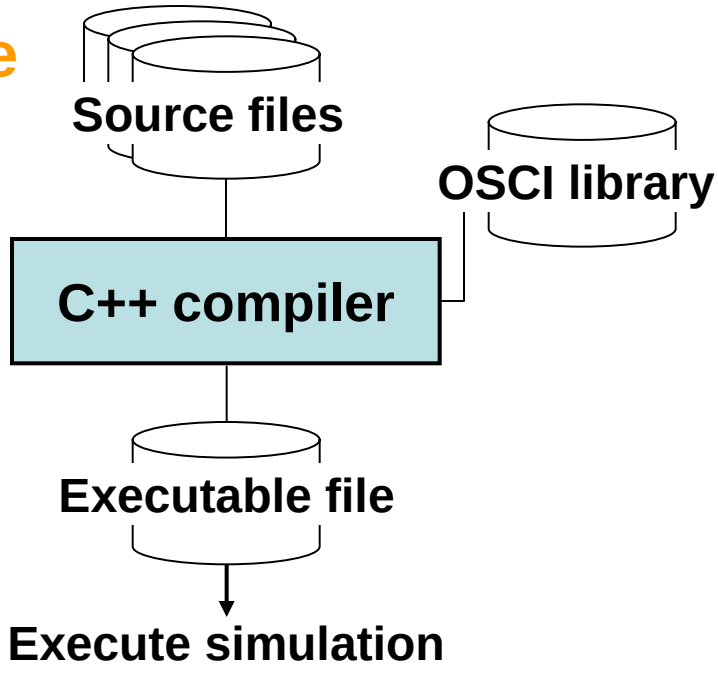
- **What is SystemC simulator**
  - **Simulate behavior of given SystemC description and other input data (including programs)**



- There are two types of SystemC simulators;
  - Free simulator: OSCI reference simulator
    - Developed by OSCI (Open SystemC Initiative) that establishes SystemC standard
  - Commercial simulators
    - Provide many useful features including debuggers, mixed language simulation, etc.
    - Synopsys's VCS-MX and Cadence's Incisive are available in RVC

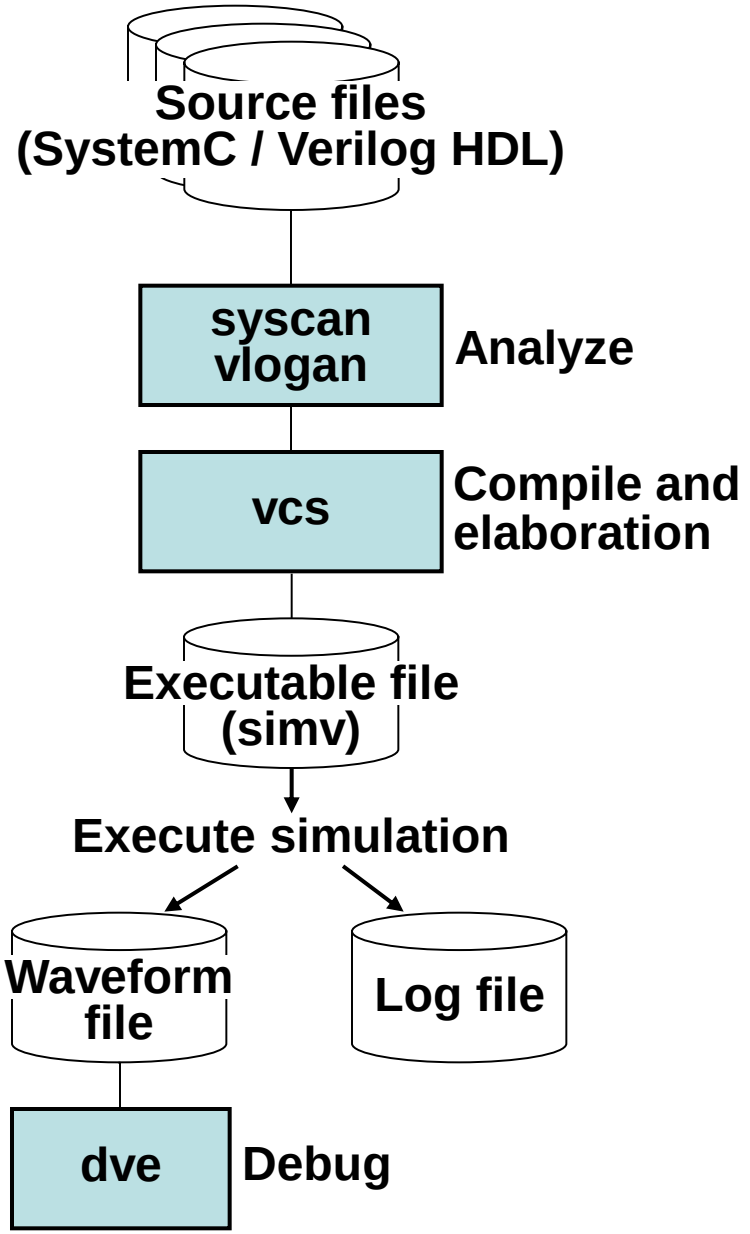
OSCI Reference Simulator

- Developed by OSCI (Open SystemC Initiative) and released with SystemC Language Reference Manual (LRM)
- How to use OSCI Reference Simulator
  - Compile the source files with a standard C++ compiler
  - At the end of compilation, link OSCI reference simulator library
  - Run the executable file that is generated by the compiler



Synopsys's VCS-MX

- Commercial simulator support many useful features:
  - Mixed language (Verilog, VHDL) simulation
  - Assertion-based verification
  - Coverage measure
  - Useful debugger with GUI



Comparison between OSCI Simulator and VCS-MX

- Choose one simulator properly according to the usage of the developed models and other conditions (OS, etc.)
- Use VCS-MX if mixed-language simulation with Verilog or VHDL is necessary

Item	OSCI simulator	VCS-MX
Supported OS	Linux and Windows	Linux only
SystemC 2.0.1/2.1/2.2	All versions supported	All versions supported
OSCI TLM2.0	Supported	Supported
Mixed-language sim.	Impossible	Possible
License fee	Free of charge	Pay license
Waveform specification	Hard	Easy
Simulation speed	Same	

Behavioral (High-level) Synthesis

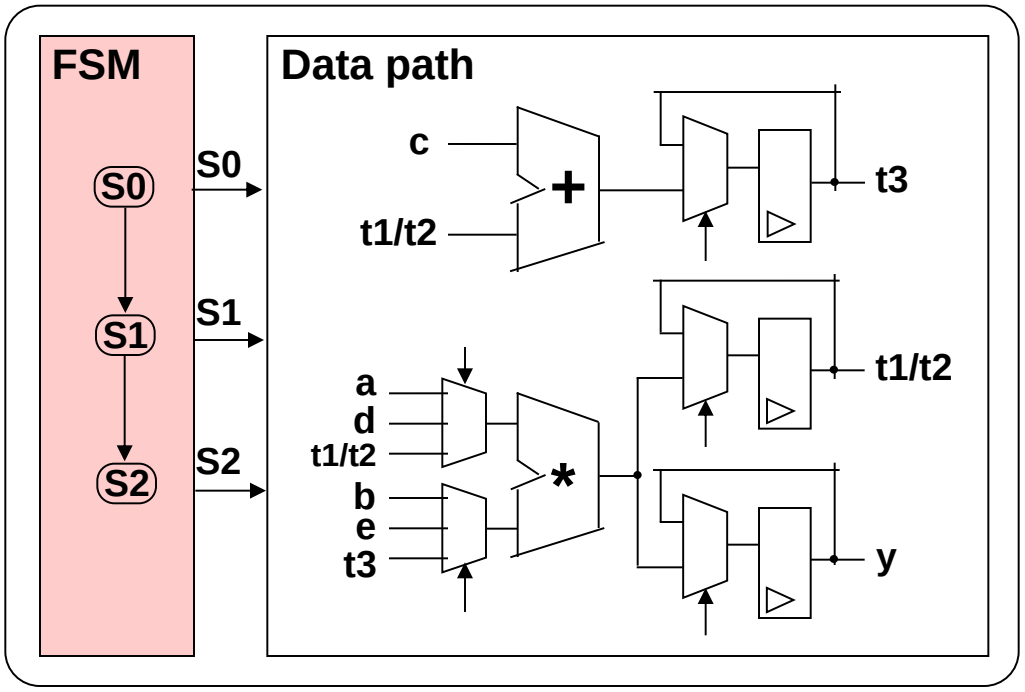
- Synthesize RTL design
  - Realizes algorithm (e.g. image processing) written in SystemC
  - Optimize in regard to frequency and circuit size
- The synthesized RTL design consists of FSM (control logic) and data path
- Cadence's CtoS (C-to-Silicon Compiler) is used in RSD and RVC

SystemC

```
void Calc::Calc_proc() {
  int t1,t2,t3,t4;

  y.write(0);
  wait();

  while(true){

      t1 = a.read() * b.read();
      wait();

      t2 = d.read() * e.read();
      t3 = t1 + c.read();
      wait();

      t4 = t2 * t3 ;
      y.write(t4);
      wait();
}
```

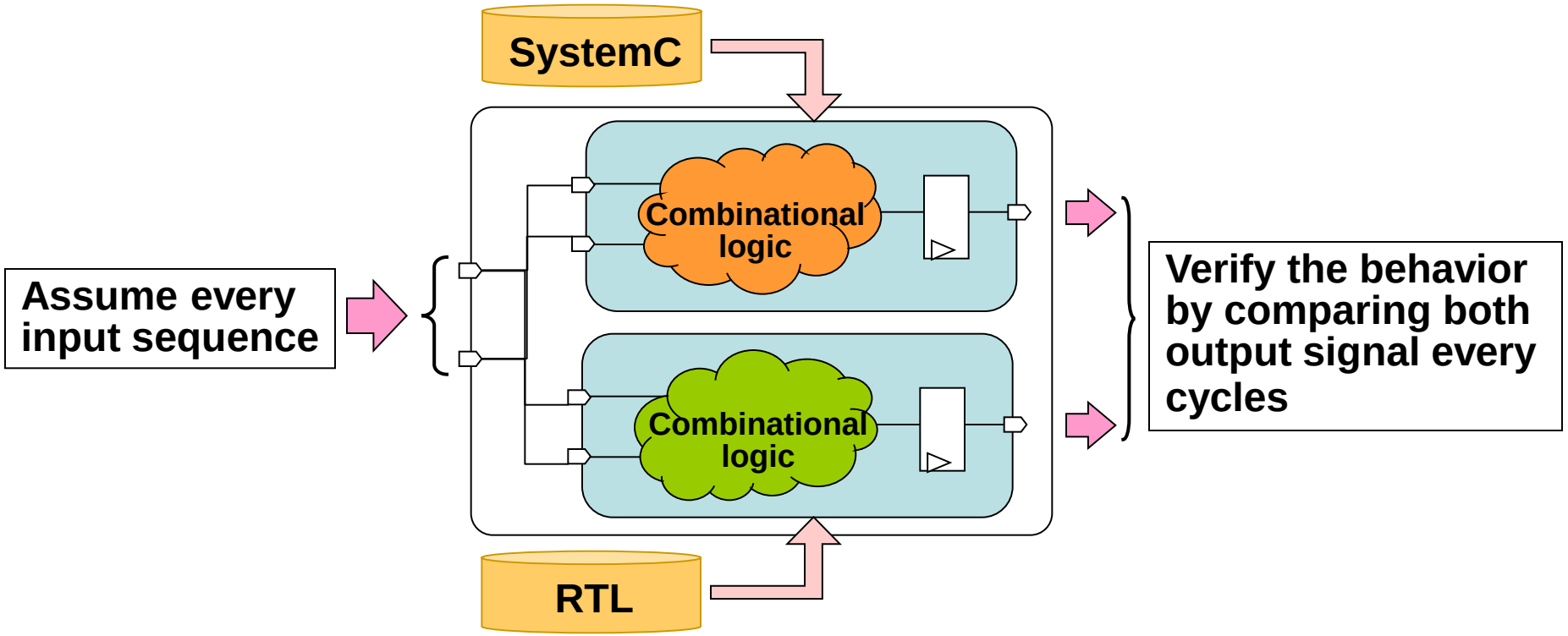
Behavioral synthesis

RTL design



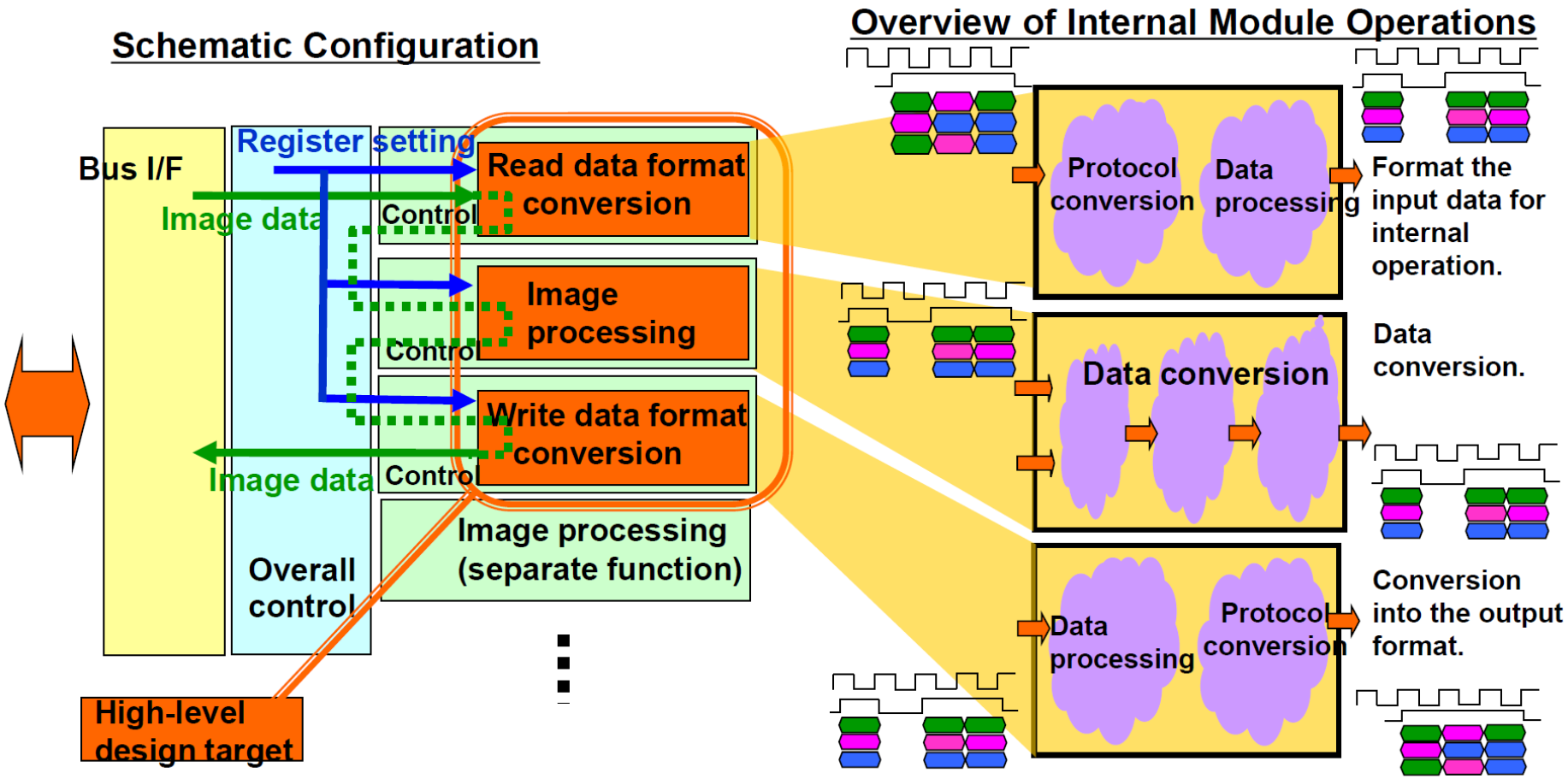
High-level Equivalency Checker

- **Formally verify functional equivalency between SystemC description and RTL description**
  - Faster than simulation
  - Available exhaustive verification
- **Calypto Design's SLEC is the only commercial tool and is used in Renesas**



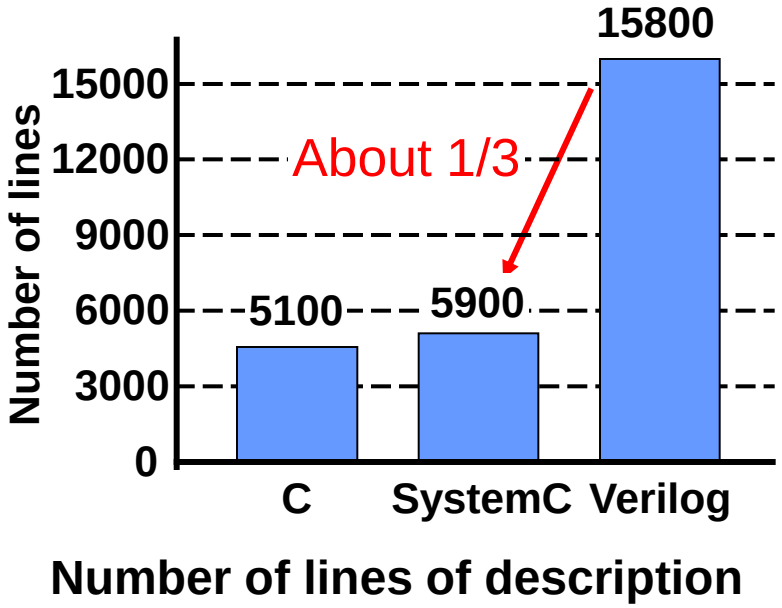
High-level Design of Image Processing IP (1)

- Verified C model (algorithm) was given, SystemC design and verification was completed in 6 man-month.

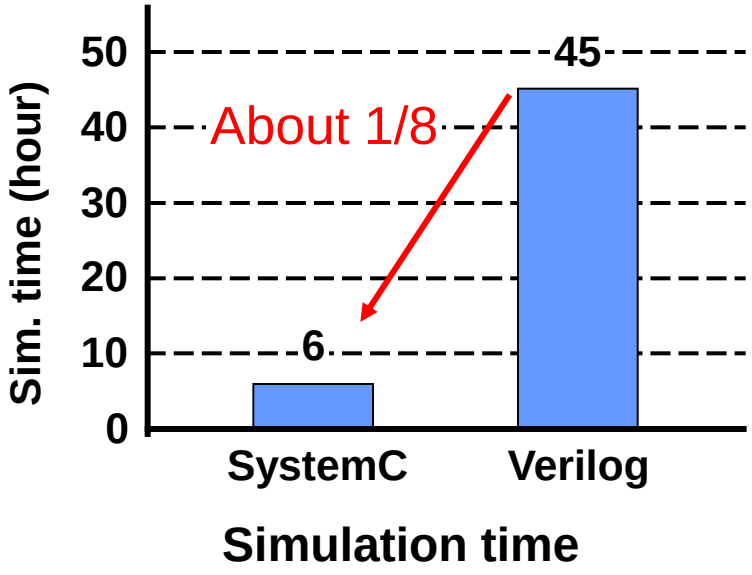


High-level Design of Image Processing IP (2)

- **SystemC model for behavioral synthesis**
  - Use original 5100-line algorithm description in C.
  - 5900 lines in SystemC (800 lines added for port definition, I/O protocol, etc.)
- **Functional verification of SystemC description**
  - Compare simulation results between SystemC and C algorithm.
  - Guarantee 100% line and branch coverage.
  - Much faster than Verilog simulation.



Number of lines of description



Simulation time

High-level Design of Image Processing IP (3)

Synthesis result

- **Area and preformance targets were achieved.**

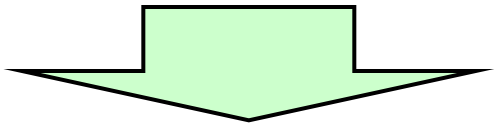
No.	Module	line of code			Ratio	Latency	Area		Freq.
		C++	SystemC	RTL	(RTL/ SystemC)		Target	Result	120Mhz/ TSMC65n
1	Read data format conversion	2.3k	2.8k	2.6k	2.6x	7	32~45kG	39kG	Met
2	Image processing	1.6k	1.9k	5.0k	2.6x	5	14~34kG	38kG	Met
3	Write data format conversion	0.8k	1.2k	3.6k	3.0x	2	30~48kG	33kG	Met

Design efficiency

- **Reduce 35% of efforts (man-month) compared with hand-crafted design.**
- **The IPs were already adopted to several SoC (Mobile, etc.) and SystemC models have been reused in another design by updating (i.e. adding functions).**

Problems of High-level Design

- Some problems exist, which prevent from spreading the high-level design technology
  - Learning new language (SystemC), lack of high-level design experiences
  - Insufficient functions and performance of design tools including high-level synthesis
    - Multi-phased clock
    - Difficult to read / modify synthesized RTL design for ECO
  - There exist lots of RTL design IPs
    - Reusing them is much easier than redesigning in SystemC
    - Effectiveness of high-level design is very limited if major part of design (LSI / IP) is RTL



- Countermeasures
  - Design guidelines and training programs are available
  - Improve design tools continuously with tool vendors and learn how to use them effectively
  - Invest strategically in translating existing RTL IPs in SystemC

Summary

- High-level design is new design methodology to reduce hardware design effort significantly
  - Design hardware at higher level of abstraction with a new design language SystemC and reduce number of lines of design description
  - Accelerate simulation by about ten times
  - Incorporate behavioral synthesis technology to avoid efforts for detailed design
  - Efficiently optimize hardware architecture in short period of time
  - Efficiently reuse existing IPs at the abstracted level when design new LSIs and IPs
- High-level synthesis tool generate RTL design supposed optimum with regard to frequency and circuit size

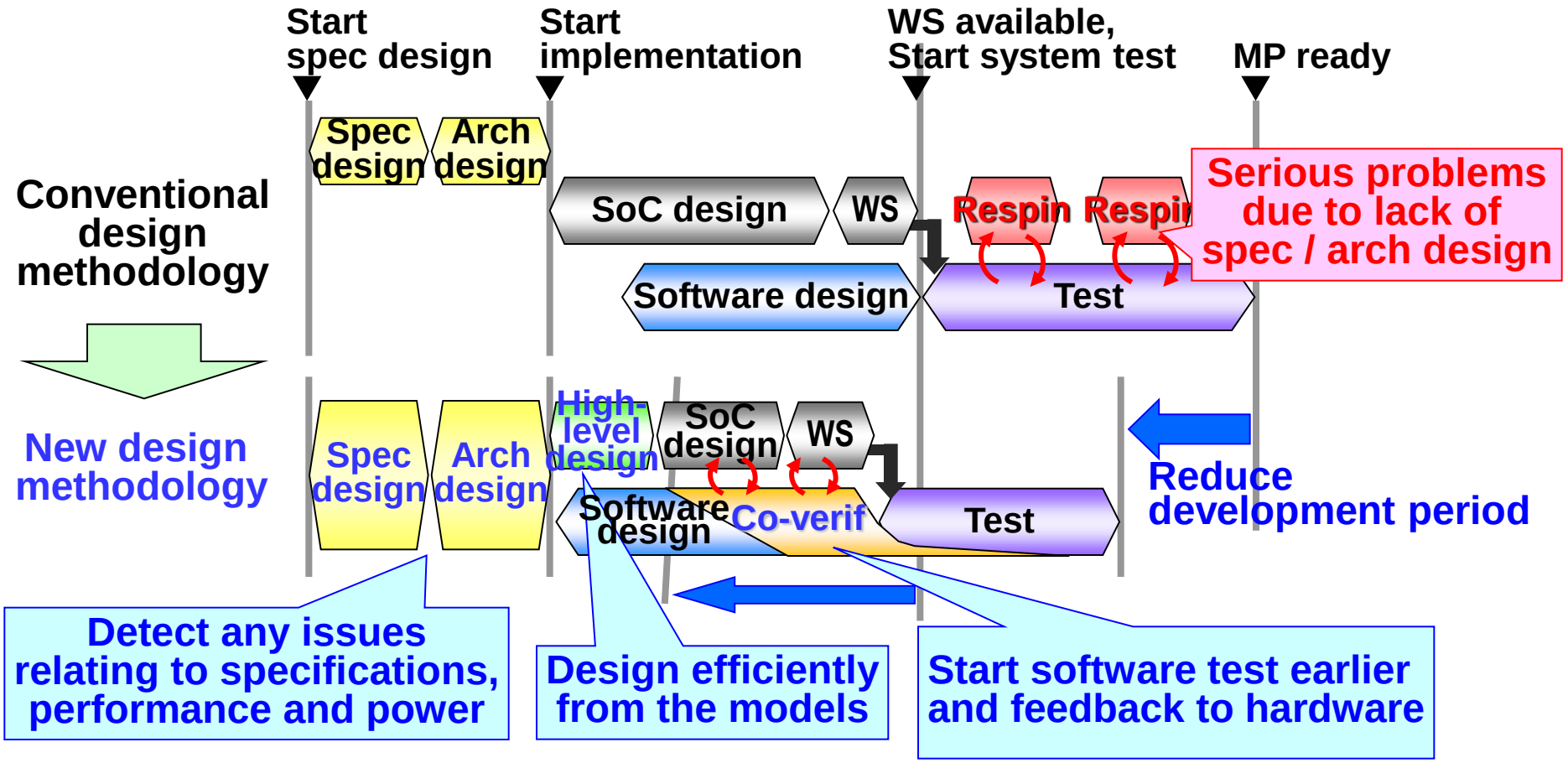
5. Concluding Remarks

Summary

- We are aiming at the next paradigm of LSI design; system-level design and high-level design
- Essence of the new design methodologies outlined in this course:
  - System-level design methodology, system-level design language SystemC, and a system-level design platforms
  - Virtual platform for software development
  - High-level design methodology, design flow and tools including behavioural synthesis
- These design methodologies are still not mature enough and there are some problems to be solved to adopt the new methodologies extensively in Renesas
  - Keep improving design flows and tools, and make design guidelines substantial by learning a lot of know-how
  - Store high-level design IPs and SystemC models even by translating existing RTL IPs

Summary: Efficient Design Methodology with System-level and High-level Design

- Successive model-based design from specification / architecture to implementation
- Pre-silicon co-design to avoid serious problems on LSI



Thank you for your attention!

Renesas Electronics Corporation