**What is SystemC**

SystemC is an class library to realize

- hardware description

- simulation

using C++ language.

(note) SystemC follows the C++'s syntax.

It contains classes to describe hardware.

(e.g. Bit width specifiable integer data types)

- We can run its simulation using general C++ compiler.

(e.g. GCC, Visual C++)

- We can run its simulation together with Verilog design, using

VCS/NC simulator.

Notion of time

Concurrency

H/W data type : Z value for ti-state buses

The SystemC Class Library has been developed to support system level design.

The first stage, release 1.0 (presently at version 1.0.2) provides all the necessary modeling facilities to describe systems similar to those which can be described using a hardware description language, such as VHDL. Version 1.0 provides a simulation kernel, data types appropriate for fixed point arithmetic, communication channels which behave like pieces of wire (signals), and modules to break down a design into smaller parts.

**Introduction to High-level design**

**Background**

Electronic machineries are getting more and more complicated, higher and higher performance.

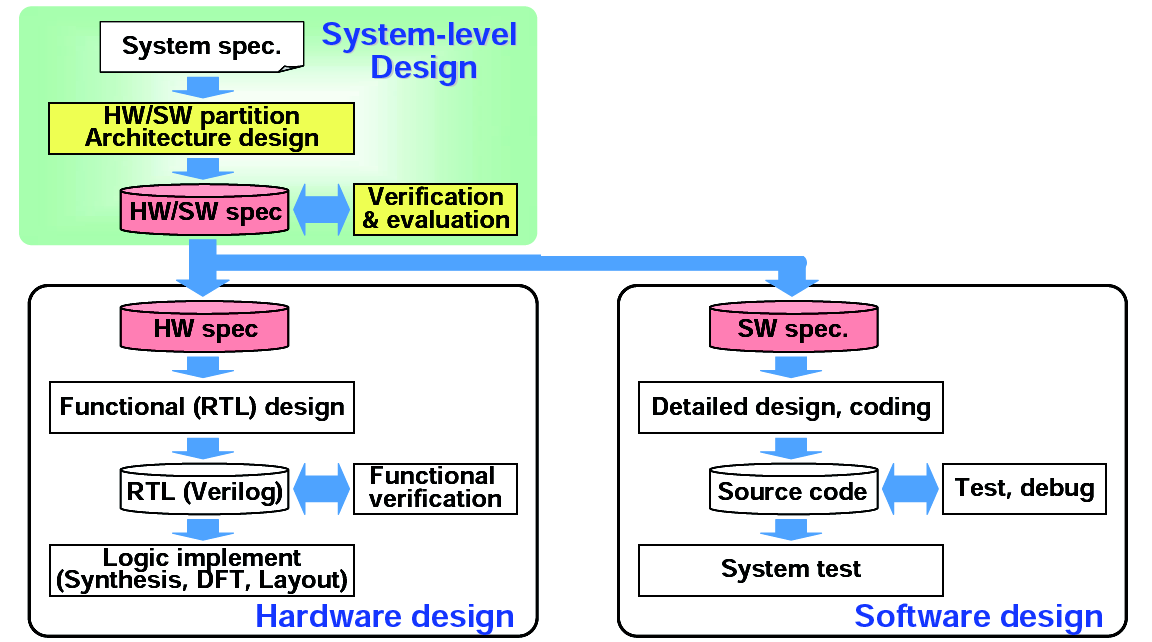
necessary to improve the design productivity.

Design methodology changed drastically in every ten years.

Now, it's time to shift to system/behaviorlevel

**System level design:** Develop appropriate hardware and software specifications according to system specification

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



9

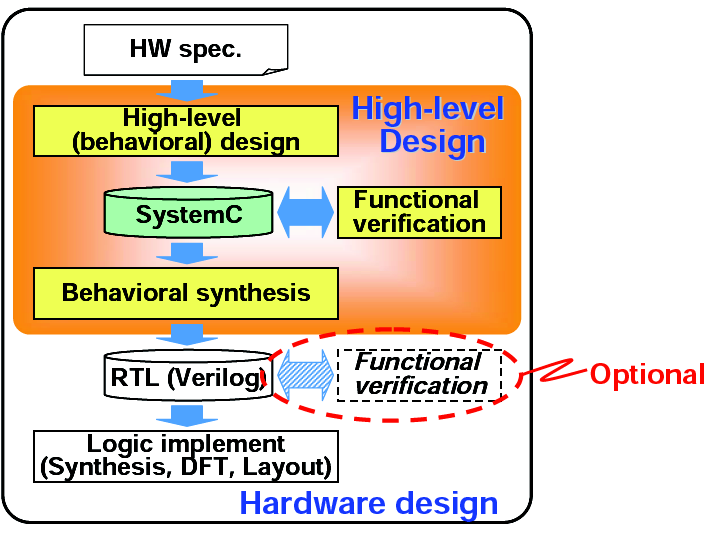
**High level design**

Design HW at higher level (i.e behavioral) in SystemC

RTL description (Verilog HDL) is synthesized by a tool from the SystemC description

Purpose

Reduce HW design cost by designing at abstract level



**Board design, LSI design, software design ?**

**Register Transfer Level (RTL) Verilog, VHDL ??**

**Problem of RTL Design Flow**

Number of lines of design description increases as LSIs get larger and more

complicated

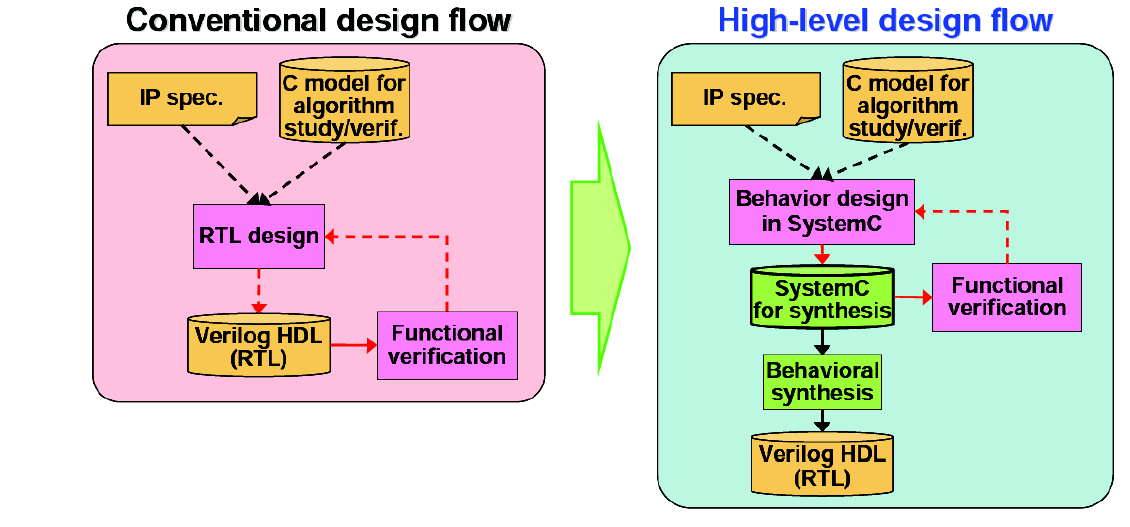
Simulation speed becomes slower and simulation time is roughly in proportion

to square of design size

Difficult to verify, debug, modify or maintain the large design

Design HW 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 man effort to design in detail

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

**Aims of High-level Design: Summary**

20

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 abstraction level so that they can be reused more efficiently

=> Reduce total design effort by using such proven designs

Optimize HW architecture in short period of time

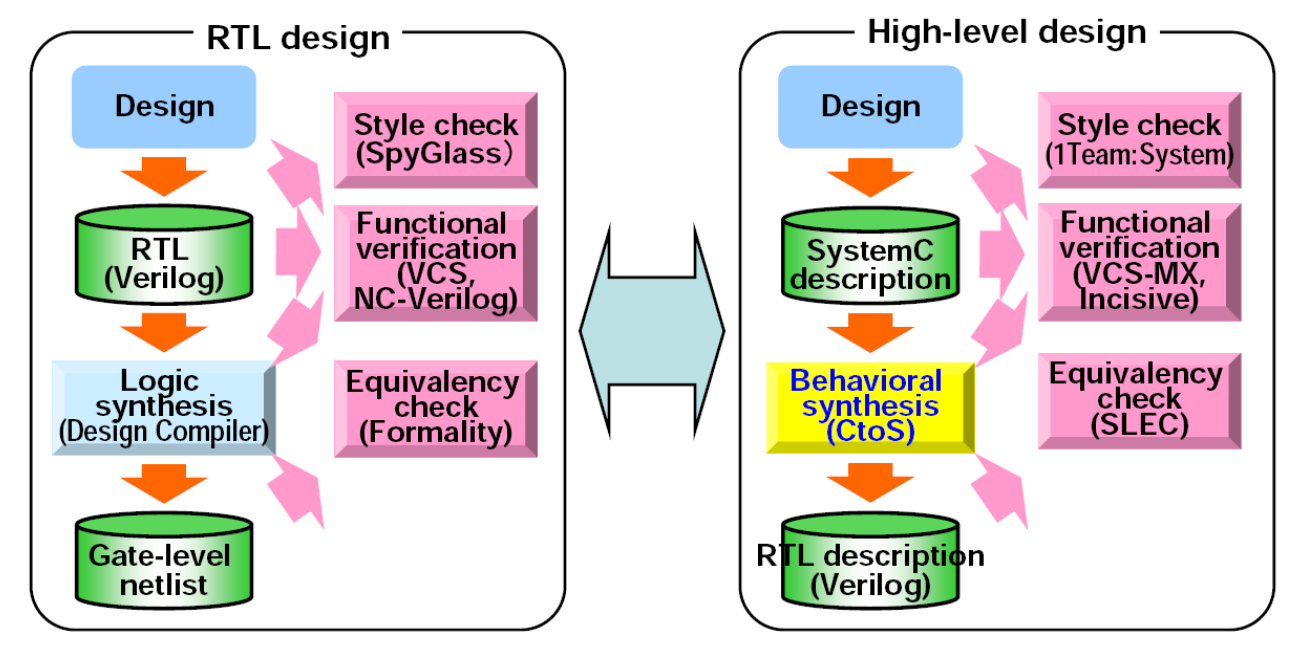
Adopting efficient high level design and automatic synthesis, many design candidates can be evaluated quickly

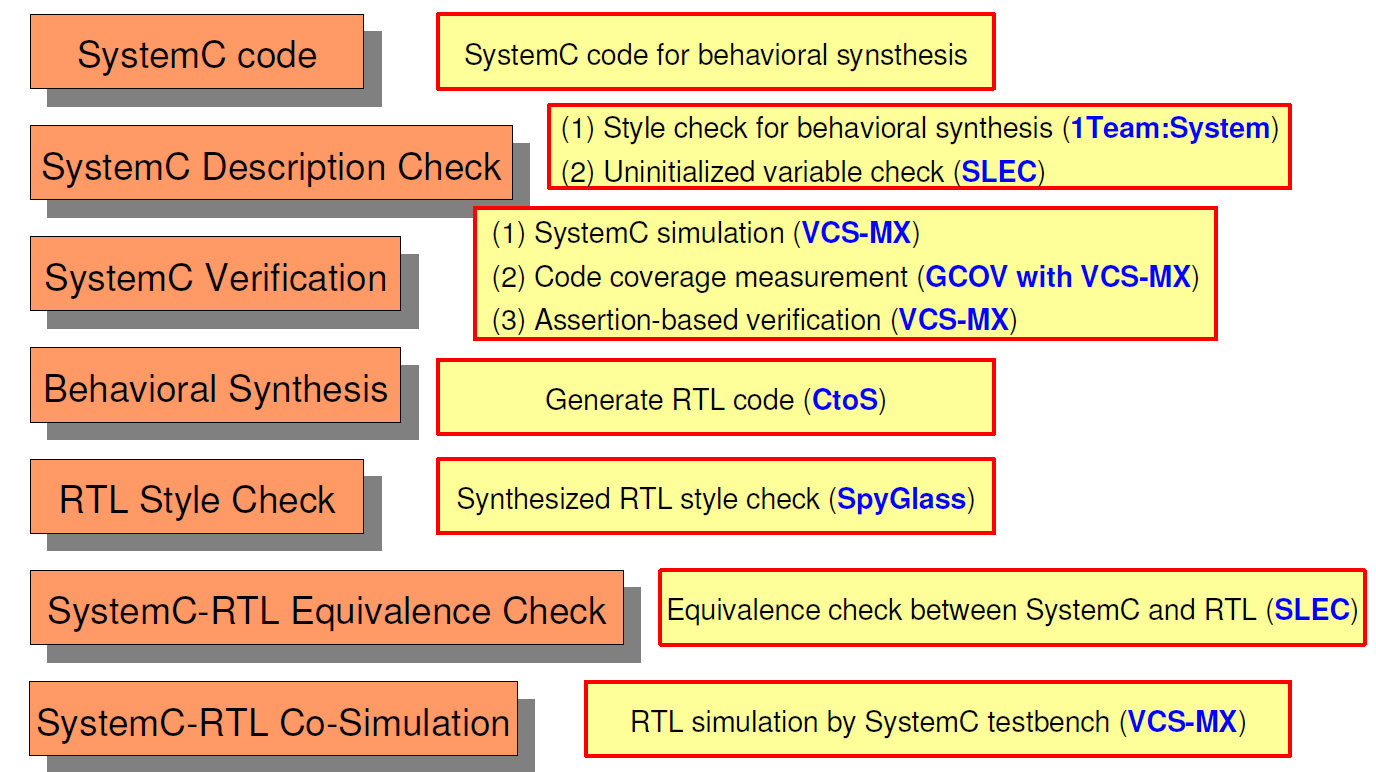
=> It becomes possible to select the best architecture

**High-level Design Flow**

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

ones of RTL design flow

****

**Design step and EDA(??) tool **

**SystemC**

**1 Introduction**

**1.1 Intent and scope**

SystemC is a set of C++ class definitions and a methodology for using these classes.

**1.2 Overview of SystemC**

This section is informative and describes in general terms a SystemC “system and how it simulates.

The SystemC library of classes and simulation kernel extend C++ to enable the modeling of systems. The extensions include providing for concurrent behavior, a notion of time sequenced operations, data types for describing hardware, structure hierarchy and simulation support.

The core language consists of an event-driven simulator as the base. It works with events and processes. The other core language elements consist of modules and ports for representing structure, while interfaces and channels are used to describe communication.

The data types are useful for hardware modeling and certain types of software programming.

The primitive channels are built-in channels that have wide use such as signals and FIFOs.

**Modules**

A Module is the basic structural building block in SystemC. It is a container class in which processes and other modules are instantiated

SC\_MODULE( *name* ) {

sc\_in *input\_port*; // (1) Port and data type

sc\_out *output\_port*;

*function*(){ // (2) Process, contrstructor and operation

*task;*

}

};

(1)

Port:

* sc\_in < data\_type > input\_nm;
* sc\_out < data\_type > input\_nm;

Data type:

* <bool> : 1-bit
* sc\_uint<N> : N-bit unsigned
* sc\_int<N> : N-bit signed
* sc\_bigunit
* sc\_biginit

(2) Process ,constructor and operation

Process

Functions executed in parallel.

Purpose: to describe structure of a circuit

void *func\_name*(void){

*reset task*

wait();

while(1){

*normal task*

wait();

}

}

Constructor

A special function.

Purpose: to specify functions

need to be treated as process.

SC\_CTOR( *name* ) {

SC\_CTHREAD( proc0 , clk.pos() );

reset\_signal\_is( reset , true );

}

Operation:

A function to do specific operation.

Purpose: to describe

operations.

int sc\_main( int sc\_argc, char\* sc\_argv[] ) {

adder adder0( “adder” );

//instance

sc\_start(); //start the program

return 0;

}

Verilog:always vs. SystemC:thread

****

****

Processes have sensitivity lists, i.e. a list of signals that cause the process to be

invoked, whenever the value of a signal in this list changes. Processes cause other

processes to execute by assigning new values to signals in the sensitivity list of the

other process.

*Method Process*

When events (value changes) occur on signals that a process is sensitive to, the process

executes. A method executes and returns control back to the simulation kernel.

When a method process is invoked, it executes until it returns. Users are strongly

recommended to not write infinite loops within a method process as control will

never be returned back to the simulator.

*Thread Processes*

Thread Process can be suspended and reactivated. The Thread Process can contain

wait() functions that suspend process execution until an event occurs on one of the

signals the process is sensitive to. An event will reactivate the thread process from

the statement the process was last suspended. The process will continue to execute

until the next wait().

The input signals that cause the process to reactivate are specified by the sensitivity

list. The sensitivity list is specified in the module constructor with the same syntax

used in the Method Process

The Thread Process is the most general process and can be used to model nearly

anything. An SC\_METHOD process to model this same design would require more

typing and be more difficult to understand and maintain. Each change of state in the

traffic light controller would have to be declared as a state in a state machine.