SystemC (IEEE 1666)

- Library built on top of C++ initially targeted as C++ framework for RTL design (i.e. modeling how components intended to be synthesized to physical design)
- Alternative to VHDL or Verilog (does not require simulation tools)
- Incorporates an embedded simulation engine that defines model semantics to delta-based simulation (consistent with Verilog/VHDL simulation semantics)

SystemC 2.0
- Rework of SystemC with more focus on System design, while still compatible with synthesizability of SystemC 1.0 designs
- Allows specification of designs from the system level to the gate level

* Modified from DVCon
- Gabe Moretti EDN
- Modules (sc-module): Define basic component of SystemC model
  - Bundling class for computational and communication components

- Ports (sc-port, sc-in, sc-out): Represent physical connections within system model
  - sc-in and sc-out can be used to model pin-level ports
  - sc-port used with TLM

- Channels: Communication model that defines specific interfaces for accessing channel

- Interfaces: Declare a set of functions that define how components can access a specific channel that implements the interface

- sc-time: Time object used to specify time behavior

- wait(): Used to model delays and waiting for events within simulation

Examples: wait()

  wait(event) \rightarrow wait for event (sc-event or predefined event)
  wait(e1 | e2 | e3) \rightarrow wait for one of these events
  wait(200, sc_NS) \rightarrow wait for specific time
  wait(200, sc_NS, el) \rightarrow wait for 200 ns or event e1 (which ever is first)

- SystemC Threads (sc-thread): Process that is executed once at creation (e.g., start of simulation)
  - Must have at least one wait() statement
  - Typically includes an infinite loop around specified functionality

- SystemC method (sc-method): Process that executes completely each time it is activated
  - Acts as a function call called by simulation engine
  - Must not contain a wait() statement
  - Typically associated with a sensitivity list (specification) that defines what events will activate method
#include "systemc.h"

enum statetype { S0, S1, S2, S3, S4};

SC_MODULE(LaserDistMeasurer)
{
    sc_in<sc_logic> clk, rst;
    sc_in<sc_logic> B, S;
    sc_out<sc_logic> L;
    sc_out<sc_lv<16>> D;
    sc_signal<statetype> state;
    sc_signal<sc_uint<16>> Dctr;

    SC_CTOR(LaserDistMeasurer)
    {
        SC_METHOD(statemachine);
        sensitive_pos << rst << clk;
    }

    void statemachine()
    {
        if( rst.read() == SC_LOGIC_1 )
        {
            L.write(SC_LOGIC_0);
            D.write(0);
            Dctr = 0;
            state = S0; // initial state
        }
        else
        {
            switch (state) {
                case S0:
                    L.write(SC_LOGIC_0); // laser off
                    D.write(0); // clear D
                    state = S1;
                    break;

                case S1:
                    Dctr = 0; // clear count
                    if (B.read() == SC_LOGIC_1)
                        state = S2;
                    else
                        state = S1;
                    break;

                case S2:
                    L.write(SC_LOGIC_1); // laser on
                    state = S3;
                    break;

                case S3:
                    L.write(SC_LOGIC_0); // laser off
                    Dctr = Dctr.read() + 1;
                    if (S.read() == SC_LOGIC_1)
                        state = S4;
                    else
                        state = S3;
                    break;

                case S4:
                    D.write(Dctr.read()>>1); // Calculate D
                    state = S1;
                    break;
            }
        }
    }
Transaction-level Modeling (TLM)

- Modeling method that separates computation from communication, whereby communication can be defined as transactions.
- Allows independent refinement of communication and computation.
- Assists in the development of large designs from early system models to final implementation.
- Communication uses channels (separation mechanism).

*Languages for TLM: SystemC, SpecC, SystemVerilog, etc.*

**TLM Model Types**

**Unsimulated**: functionality without timing information.
- To simulate, semantics still apply, to define execution order of system components (but no real time is modeled).
- E.g., everything occurs in 0s of simulated time.

**Approximate Timed**: Provides timing details, but without relying on cycle accurate model.

**Pin/Cycle Accurate**: Provides sufficient details for cycle accurate and bit-level accurate implementation.
- Also referred to as cycle-accurate, bit accurate (CABA) model.
- Model is detailed enough to synthesize/manufacture final implementation.
System-Level Modeling Abstractions:

A: Specification/Architecture/Model
   - System-level assembly of components modeling processing elements, memories, and communication through channels
   - Approximated timed computation using simple "wait" statements

B: Component Assembly Model
   - System-level assembly of components modeling processing elements, memories, and communication through channels
   - Approximated timed computation using simple "wait" statements

C: Bus-arbitration Model
   - Communication channels are modeled using transactions in which arbitration is used to handle contention for bus
   - Approximate timed using "wait" delays for each transaction

D: Bus Functional Model
   - Communication is specified as a cycle-accurate model of physical bus signals (i.e., physical wires of bus)
   - Typically uses wrappers to map transactions at bus-arbitration model to cycle-accurate model
   - Preserves separation of computation and communication

E: Cycle-accurate Computation
   - Computational elements are modeled at pin and cycle-accurate level
   - HW processing elements modeled at RTL
   - SW modeled using cycle-accurate simulation for specific processor target

F: Implementation Model
Algorithmic and Architectural Modeling

- Defines which algorithms are needed
- How will algorithms be implemented (HW vs SW)
- Used to determine if performance is adequate
  - to processors, buses, system architecture

Virtual Software Development Platform

- Allows early development of software and analysis of performance and integration with HW platform
- Works with performance analysis of software
  - by instruction/cycle accurate simulation possible

Refinement

- Allows iterative refinement of system design as needed to add additional design details or modify design to meet requirements

Validation/Verification

- Verify that model/implementation accurately implement system requirements

Synthesis

- Automated refinement methods to explore design alternatives (design space exploration) or refine design to produce lower-level details
TLA-based System Design Flow Example

1. Requirements Definition
   → Requirements Document
   ↓ System Architecture Model Development
   → SAM
   → Transaction Level Model Development
   → TLM
   → HW Refinement
   → RTL

   SW Design and Development
   ↓ HW Verification Environment Development
   ↓ RTL to GDSII Flow