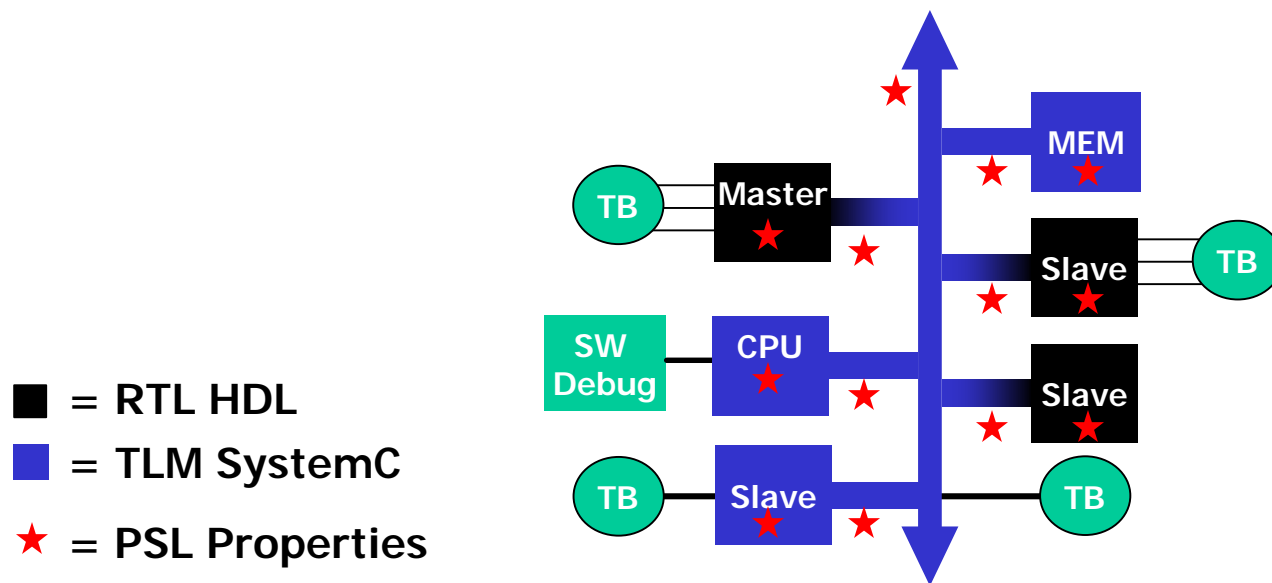


# Enabling PSL Assertions in SystemC

**Stuart Swan**  
**Senior Architect**  
**Cadence Design Systems, Inc.**  
**June 2004**

# Introduction

- Want to leverage properties across system level and RTL designs.
- Provide uniform PSL support in Verilog, VHDL, and SystemC.
- Enable top-down and bottom-up property flow.



# Approach

- Modeled on VHDL/Verilog flavors.
  - Separate PSL verification units bound to SystemC module
  - Embedded PSL within SystemC using meta-comments.
- Same PSL syntax as VHDL/Verilog enables portability.
- Full support for the PSL 1.1 LRM “simple subset” is planned.

# Example (embedded assertions)



```
// Very simple SystemC 2-bit up counter
#include "systemc.h"

SC_MODULE(ctr) {
    sc_in<bool> clk;
    sc_out<sc_lv<2> > cnt;

    void increment(); // Counter thread

    SC_CTOR(ctr) : clk("clk"), cnt("cnt"), cntReg(0)
    {
        SC_THREAD(increment);
        sensitive << clk.pos();
    }

    // psl default clock = fell(clk);
    // psl flip1: assert always (cnt[0] -> next !cnt[0]);
    // psl flip2: assert always (cnt[0] -> next[2](cnt[0]));

private:
    sc_int<2> cntReg;
};
```

# Example (property file)

- The same set of assertions can be stored in a property file like the following:

```
// Property file for PSL assertions (filename ctr.psl)

vunit flip(ctr) {

    // This vunit is bound to the SystemC module name "ctr".
    // Each instance of "ctr" will contain instances of the two
    // assertions below, as if they had been included within
    // the module definition as meta-comments.

    default clock = fell(clk);
    flip1: assert always (cnt[0] -> next !cnt[0]);
    flip2: assert always (cnt[0] -> next[2](cnt[0]));
}
```

# SystemC “flavor” of PSL

- PSL in SystemC is implemented using SystemC methods.
  - User's PSL assertion is translated into a SystemC method invoked at the appropriate clock edge (see below).
- The PSL “clock” in SystemC can be an event or event finder.
  - e.g. `clk.posedge_event()`, `clk.pos()`, or other event or event finder.
  - Unclocked assertions are sensitive to all signals referenced in the assertion.
- Syntactic sugar helps enhance portability of assertions across language boundaries:
  - Instead of `a.range(0,2)`, **use** `a[0:2]`
  - Instead of `cnt.read()[0].to_char()`, **use** `cnt[0]`

# Some Uses of PSL in SystemC



- Interface checking
  - Isolating errors close to their source.
- Protocol checking
  - Monitoring communication interactions on a bus.
- Coverage analysis
  - Transaction monitoring from assert/cover directives.
  - Future capability for reactive testbenches based on PSL coverage points.
- In all cases, the ability to reuse assertions across SystemC, Verilog, and VHDL provides many benefits.

**cādence**<sup>®</sup>

**cādence**<sup>®</sup>