Temporal assertions in SystemC

Mikhail Moiseev, Intel Corporation
Leonid Azarenkov, Intel Corporation
Ilya Klotchkov, Intel Corporation

Copyright Permission

- A non-exclusive, irrevocable, royalty-free copyright permission is granted by **Intel Corporation** to use this material in developing all future revisions and editions of the resulting draft and approved Accellera Systems Initiative **SystemC** standard, and in derivative works based on the standard.

Current SystemC assertions and error reporting

- sc_report_handler::report with pre-defined macros
  - SC_REPORT_FATAL (const char*, const char*)
  - SC_REPORT_ERROR (const char*, const char*)
  - SC_REPORT_WARNING (const char*, const char*)
  - SC_REPORT_INFO (const char*, const char*)
  - sc_assert(bool) which is SC_REPORT_FATAL
- C++ assert(bool)

SystemVerilog assertions (SVA)

- Immediate assertion
  - assert(expr)
  - assert(expr) else …
- Concurrent assertions
  - Run in always block
    - assert property (req |-> resp);
    - assert property (req |=> resp);
    - assert property (req |-> ##1 resp);
    - assert property (req |-> ##[1:2] resp);
  - Run in module scope with event specified
    - assert property (@(posedge clk) req |-> resp);
    - assert property (@(negedge clk) req |-> ##[1:2] resp);
- Combining sequences: and, intersect, or, …
- Local variables in property
- System functions: $rose, $fell, $stable, $past, ...

Temporal assertions in SystemC

- Extend existing SystemC assertions with temporal properties
  - Look similar to temporal SystemVerilog assertions (SVA)
- Use the assertion
  - in SystemC simulation
  - to generate equivalent SVA in high level synthesis
- Intended for hardware design, consider SystemC synthesizable subset

Temporal assertion interface

- Temporal assertions placed in
  - Module scope
    - *SCT_ASSERT (LHS, TIME, RHS, EVENT)*
    - *SCT_ASSERT (RHS, EVENT)*, short form of previous where LHS is *true* and time is *0*
  - Thread process function scope
    - *SCT_ASSERT (LHS, TIME, RHS)*
    - *SCT_ASSERT_LOOP (LHS, TIME, RHS, ITER)*

LHS – antecedent assertion expression (pre-condition)
TIME – temporal condition is specific number of cycles or time interval in cycles
RHS – consequent assertion expression, checked to be true if pre-condition was true (post-condition)
EVENT – cycle event
ITER – loop iteration counter variable(s), comma separated in arbitrary order

Temporal assertions in module scope

- Assertions in module scope avoid to clutter design function code
  - access module fields: signals, ports, …
  - require an event which is clock positive, negative or both edges

```
// A and B some expressions can be evaluated as bool
SCT_ASSERT(A, SCT_TIME(0), B, clk.neg());
SCT_ASSERT(A, SCT_TIME(1), B, clk.neg());
SCT_ASSERT(A, SCT_TIME(3,1), B, clk);
SCT_ASSERT(A, (2), B, clk);
SCT_ASSERT(A, (4,0), A, clk);
SCT_ASSERT(B, clk.pos());
```

Temporal assertions in module examples

```
// A.cpp
static const unsigned T = 3;
static const unsigned N = 4;
sc_clk_in clk{“clk”};
sc_in<bool> req{ "req"};
sc_out<bool> resp{"resp"};
sc_signal<sc_uint<8>> val{"val"};
sc_vector<sc_signal<bool>> enbl{"enbl",N};
...
SCT_ASSERT(req, (1), resp, clk.neg());
SCT_ASSERT(req, (1,2), val.read()== N, clk);
SCT_ASSERT(enbl[0], (3), enbl[1], clk);
SCT_ASSERT(req || !resp, clk.pos());
```

```
...
`ifndef SVA_OFF
A10: assert property(@(negedge clk) req |=> resp);
A11: assert property(@(clk) req |-> ##[1:2] val == 4);
A12: assert property(@(clk) enbl[0] |-> ##3, enbl[1]);
A13: assert property(@(posedge clk) 1 |-> req || !resp);
`endif // SVA_OFF
```

Temporal assertions in function

- Assertions in function scope can access local variables
- Assertions placed in reset section or after it before main loop
  - assertions after reset not executed during reset
- Special kind of assertions used inside of loops
  - intended for array/vector of modules, signals, ports or others

```
// A and B some expressions can be evaluated as bool
SCT_ASSERT(A, SCT_TIME(0), B);
SCT_ASSERT(A, SCT_TIME(1), B);
SCT_ASSERT(A, SCT_TIME(3,1), B);

for (int j = 0; j < M; ++j) {
    SCT_ASSERT_LOOP(A, SCT_TIME(1), B, j);  // A and B expressions depends on j
}
```

Temporal assertions in function examples

```
void thread_proc() {
   // Reset section
   // ...
   // Assertions in reset section
   SCT_ASSERT(req, SCT_TIME(1), ready);
   wait();
   // Assertions after reset section
   SCT_ASSERT(req, (2,3), resp);

   // Main loop
   while (true) {
      // No assertion in main loop
      wait();
}}
```

```
always_ff @(posedge clk or negedge nrst) begin
   if (~nrst) begin
      ...
   end else begin
      ...
   `ifndef SVA_OFF
     // Assertions after reset section
     sctAssertLine33:
       assert property (req |-> ##[2:3] resp);
   `endif // SVA_OFF
   end
`ifndef SVA_OFF
   // Assertions from reset section
   sctAssertLine30:
     assert property (req |=> ready);
`endif // SVA_OFF
end
```

Temporal assertions in loop examples

```
static const unsigned N = 4;
static const unsigned M = 3;
sc_vector<sc_signal<bool>> e{"e", N};
sc_vector<sc_vector<sc_signal<bool>>>
a{"a", N};  // N x M
...
void thread_proc() {
  for (int i = 0; i < N; ++i) {
   SCT_ASSERT_LOOP(e[i], (1), !e[i], i);
   for (int j = 0; j < M; ++j) {
     SCT_ASSERT_LOOP(a[i][j], (2),
                     a[i][M-j-1], i, j);
  }}
  wait();
  ...
}}
```

```
always_ff @(posedge clk or negedge nrst) begin
   ...
`ifndef SVA_OFF
    for (integer i = 0; i < 4; ++i) begin
      sctAssertLine70:
        assert property (e[i] |=> !e[i]);
    end
    for (integer i = 0; i < 4; ++i) begin
    for (integer j = 0; j < 3; ++j) begin
      sctAssertLine72:
        assert property (a[i][j] |-> ##2
                                     a[i][3-j-1]);
        end
    end
`endif // SVA_OFF
end
```

Temporal assertions in an industrial module

```
// Check no read burst interleaving by write request
SCT_ASSERT(this->core_req && !this->core_req_oper && burst_ractive && !this->burst_rlast,
           SCT_TIME(1), !this->core_req || !this->core_req_oper,
           this->core_clk.pos());

// Port ready cannot be de-asserted w/o request
SCT_ASSERT(resetn && this->core_req && (!arvalid || !this->clocks_match) && aready,
           SCT_TIME(1), !resetn || aready, this->core_clk.pos());

// Port valid cannot be de-asserted w/o response taken
SCT_ASSERT(resetn && this->core_req && (!rready || !this->clocks_match) && rvalid,
           SCT_TIME(1), !resetn || rvalid, this->core_clk.pos());

// Port valid cannot be de-asserted w/o request
SCT_ASSERT(resetn && this->core_req && (!aready || !this->clocks_match) && arvalid,
           SCT_TIME(1), !resetn || arvalid, this->core_clk.pos());
```

Implementation details

- The assertions are implemented with *SCT_ASSERT* and *SCT_ASSERT _LOOP* macros
- In SystemC simulation
  - Dynamic allocation of a *sct_property_expr* class which captures LHS and RHS as lambdas. This class *operator()* evaluates lambdas and stores pre- and post-condition traces in specified time interval, checks post-condition if pre-condition was true and reports error in case of violation
  - Create spawned method process sensitive to EVENT or current thread process event, which runs *sct_property_expr()*
  - Registration of the *sct_property_expr* instance in a static map with hash calculated for assertion string, process name and loop iteration(s). That ensures one sct_property_expr instance for an assertion in each module instance and loop iteration
- Implemented in C++11

Translation to SVA in HLS mode

- Translation to SVA if *__SC_TOOL__* is defined
  - Provide code in form easy to parse by an HLS tool
- *SCT_ASSERT* in function replaced with *sct_assert_in_proc_func()* function call
- *SCT_ASSERT* in module scope replaced with *sct_property_mod* type variable declaration
  - The variable name constructed with line number where *SCT_ASSERT* placed

```
template<class T1, class T2>
void sct_assert_in_proc_func(bool lhs, bool rhs, const char* name, T1 lo, T2 hi){}

struct sct_property_mod {
    template<class T1, class T2>
    explicit sct_property_mod(bool lhs, bool rhs, sc_event_finder& event,
                              const char* name, T1 lo, T2 hi) {} ... }
```

Evaluation results: artificial examples

Design	Number of processes	Assertions	SystemC simulation time increase
Summator	2	One assertion w/o pre-condition, single time	10%
		One assertion with pre-condition, single time	12%
		One assertion with pre-condition, time interval (1,3)*	13%
		One assertion with pre-condition, time interval (10,30)*	15%
FIFO	5	One assertion checks FIFO is not empty after push	4%
		Four assertions check main FIFO properties	17%

** time interval in number of cycles*

Evaluation results: industrial designs

Design	Number of processes	Number of assertions	SystemC simulation time increase	Verilog simulation time increase
A	71	19	11%	15%
B	68	21	16%	15%
C	101	22	5%	5%
D	109	41	14%	11%
E	212	38	4%	6%

Intel Compiler for SystemC

- All the code examples and SVA generated with Intel® Compiler for SystemC* (ICSC)
- ICSC translates cycle-accurate SystemC code to synthesizable SystemVerilog
  - SystemC synthesizable standard full support
  - Arbitrary C++ at elaboration phase, including module constructors
  - C++11/14/17 support
  - Human-readable generated Verilog code
  - CMake based project, no pragmas and other code pollution
  - Fast translation procedure, seconds to a few minutes
- ICSC open source tool, available under Apache 2.0 license
  - https://github.com/intel/systemc-compiler

Temporal assertion sources

- Assertion implementation available
  - https://github.com/intel/systemc-compiler in components/common
  - *sct_assert.h* – macros definition
  - *sct_property.h/sct_property.cpp* – sct_property_expr and sct_property_storage definition
  - *sct_assert.patch* – trivial patch for SystemC to get current process sensitivity

Conclusion

- We propose
  - Evaluate and improve the temporal assertions to finalize interface and implementation
  - Add the assertions in SystemC distribution and update IEEE1666
  - Add immediate and temporal assertions into Synthesizable subset standard