Temporal assertions in SystemC

Mikhail Moiseev, Intel Corporation
Leonid Azarenkov, Intel Corporation
Ilya Klotchkov, Intel Corporation
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 assertions
  – 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, ...

• System functions: $rose, $fell, $stable, $past, ...
Temporal assertions in SystemC

• We propose to 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_TIME can be omitted
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

```c
// 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

```verilog
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

```cpp
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();
    ...
}
```
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) && arready,
           SCT_TIME(1), !resetn || arready, 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 && (!arready || !this->clocks_match) && arvalid,
           SCT_TIME(1), !resetn || arvalid, this->core_clk.pos());
Implementation details

• The assertions 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 \_\_HLS\_\_ is defined
  - Provide code in form easy to parse by an HLS tool
- \texttt{SCT\_ASSERT} in function replaced with \texttt{sct\_assert\_in\_proc\_func()} function call
- \texttt{SCT\_ASSERT} in module scope replaced with \texttt{sct\_property\_mod} type variable declaration
  - The variable name constructed with line number where \texttt{SCT\_ASSERT} placed

```cpp
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: industrial designs

<table>
<thead>
<tr>
<th>Design</th>
<th>Number of processes</th>
<th>Assertion number</th>
<th>SystemC simulation time increase</th>
<th>Verilog simulation time increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>71</td>
<td>19</td>
<td>11%</td>
<td>15%</td>
</tr>
<tr>
<td>B</td>
<td>68</td>
<td>21</td>
<td>16%</td>
<td>15%</td>
</tr>
<tr>
<td>C</td>
<td>101</td>
<td>22</td>
<td>5%</td>
<td>5%</td>
</tr>
<tr>
<td>D</td>
<td>109</td>
<td>41</td>
<td>14%</td>
<td>11%</td>
</tr>
<tr>
<td>E</td>
<td>212</td>
<td>38</td>
<td>4%</td>
<td>6%</td>
</tr>
</tbody>
</table>
Conclusion

- We demonstrate possibility to implement a subset of SVA in C++11 and will submit a proposal to SystemC language workgroup
- Proposed temporal assertions are not final solution, but a step toward that
- Assertion implementation available at
  - [https://github.com/intel/systemc-compiler](https://github.com/intel/systemc-compiler) in components/common
- SystemC-to-SystemVerilog and temporal assertions-to-SVA translation done with Intel® Compiler for SystemC*
  - [https://github.com/intel/systemc-compiler](https://github.com/intel/systemc-compiler)
Questions