Assertion Checking in SystemC HLS Flows

Bob Condon

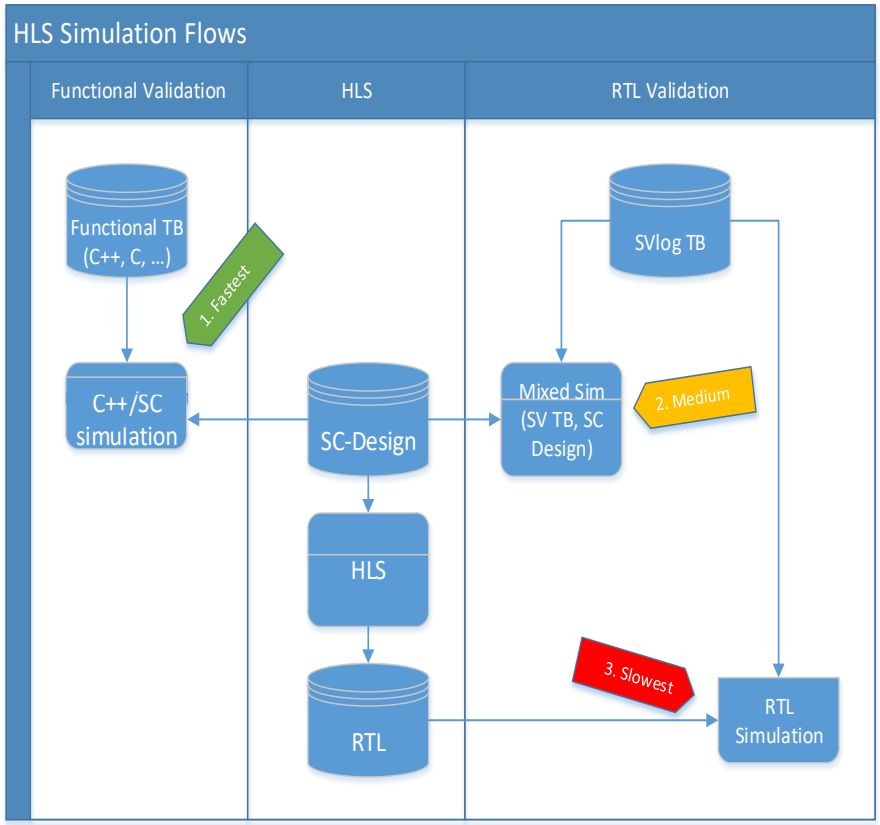
Presented by Rauf Salimi

Presentation 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 this standard.

HLS Simulation Flows

- Key benefit: faster time to bug free RTL.
  1. Fast, re-use HL TB, finds most functional bugs.
  2. Medium Effort, speed
  3. Slowest, most effort. Triage is harder.



C++/SystemC Assertion

- Used to test for conditions which must "always" be true
- Triggers program termination on failure.
- Should not contain side-effects.
- Helps specify intent
- SystemC has sc_assert()

```
enum opcode_t {ADD, SUB, MULT, DI
V};
void alu(opcode_t op, uint8_t a,
uint8_t b, uint8_t& c) {
  switch (op) {
  case ADD:  c = a + b; break;
  case SUB:  c = a - b; break;
  case MULT: c = a * b; break;
  case DIV:  assert(b != 0);
               c = a / b; break;
  default:  assert("Bad Opcode"
                && 0); break;
    }
}
```

SystemVerilog Assertions

- Immediate – Like C++ assert
  - assert (B != 0) else $error("Division by 0 attempted");
- Concurrent – specifies a "property"
  - assert property (! (rdMem && wrMem)); // signals are mutex
  - May be expressed using temporal logic
  - "Grant will be asserted no later than 1 cycle after Req asserted"
  - Assert property (@(posedge Clk) Req |-> ##[1:2] Grant);
- May be validated dynamically (by simulation) or statically (by formal tools
- Many users find writing temporal logic hard and re-use libraries of existing assertions. [See Accellera OVL]

RTL IP flows

- HLS tools support mapping a function to a pre-existing piece of RTL.
- Intended to reuse hand-optimized bit of Vlog code.
- Details depend on your HLS tool.
- User writes 2 versions of div
  - C++ and Vlog
  - User guarantees they are same

```
void DUT::proc() {
  wait();  // ^^ reset action
  while (true) {
    while (stall.read()) wait();
      bool vld = v_in.read();
      if (vld) {
        // call the vlog module
        DT a =a_i.read(), b_i.read();
        DT c = div(a, b);
       }
       v_out.write(vld);
       c_out.write(c);
       wait();
    }
}
```

Assert as RTL IP [Desired] *

- Treat each assertion in SV library as an RTL IP.
- Let IP flow instantiate the assertion in gRTL.
- * This doesn't quite work

  a_NE_b is void type

```
void DUT::proc() {
 wait();  // ^^ reset action
 while (true) {
 while (stall.read()) wait();
  bool vld = v_in.read();
  if (vld) {
   DT a =a_i.read(), b_i.read();
   a_NE_b(b, DT(0));
   DT c = (a / b);
  }
  v_out.write(vld);c_out.write(c);
  wait();
 }
}
```

Assert as RTL IP [Desired] *

```
// SystemC code
void a_NE_b (DT a, DT b) {
  #ifndef HLS
     sc_assert( a != b);  //<<< Assert in SystemC code…
  #else
  // HLS vendor-specific code
  // maps formals to RTL ports
  #endif
}
```

becomes

```
module a_ne_b (clk, rst, valid_in,   stall, a, b);
  input clk, rst, valid_in, stall;    // … becomes a module with an
  input [31:0] a, b;                  // instance of an SV assert from
  always @(posedge clk) begin         // our SV assertion library
    if (valid_in && !rst) begin
     `ASSERTC_MUST(a_ne_b, a != b,rst,
       `ERR_MSG("a_ne_b fails: a = %x, b = %x", a, b));
       end
   end
endmodule // a_ne_b
```

…but assertions have no side effects

- As far as HLS is concerned, a_NE_b can be optimized away (no write to a signal or port)

- HACK Write to extra signal
  - Make assertion return bool
  - Gang all the extra return values with XORs.
  - Has a real (but small) hardware cost

```
void DUT::proc() {
 bool p = false;  // preserve asserts
 pp.write(p); // with pseudo port
 wait();  // ^^ reset action
 while (true) {
 while (stall.read()) wait();
  bool vld = v_in.read();
  if (vld) {
   DT a =a_i.read(), b_i.read();
   p ^= a_NE_b(b, DT(0));
   DT c = (a / b);
  }
  v_out.write(vld);c_out.write(c);
  pp.write(p);
  wait();
 }
}
```

RTL IP flows [preserved]

```
bool a_NE_b (DT a, DT b) {
  #ifndef HLS
     sc_assert( a != b); //<<< Assert in SystemC code…
     return 1;           //<<< '1' will feed XOR chain
  #else
  // HLS vendor-specific code maps formals to RTL ports
  #endif
}
```

becomes

```
module a_ne_b (clk, rst, valid_in, stall, a, b, a_ne_b_out);
   input clk, rst, valid_in, stall;
   input [31:0] a, b;
   output       a_ne_b_out;
   always @(posedge clk) begin
      if (!rst && !stall && valid_in) begin
         `ASSERTC_MUST(a_ne_b, a != b, rst,
          `ERR_MSG("a_ne_b fails: a = %x, b = %x", a, b));
      end
   end
   assign  a_ne_b_out = 1'b1; //<<< '1' will feed XOR chain
endmodule // a_ne_b
```

In a Design

- Created a small assertion library (C++, Vlog) for a small bus interface block
- Assertions trigger in both SystemC and Verilog simulation.
- Triage in mixed-language simulation is significantly eased.
- For our designs, the extra hardware (the XOR chain) is almost too small to measure.
- The coding changes in the systemc code is moderate.
  - XOR chain and preserved signal.
- Somewhat labor prone
  - Library must be maintained and validated
  - Details (but not approach) is vendor specific.

Conclusion

- Assertions are useful both for validation and triage
- But – implementing today involves cost and hacks.
- We think the implementation should move to the vendor HLS tools.
  - They can automate the dead code issue.
  - They can directly translate the assertion to sv
- Future open question:
  - Can HLS tools use assertions to improve design quality.