SystemC Suggested Improvements Based On a Formal Flow of HLS Code

Dominik Strasser
VP Engineering
OneSpin Solutions
Presentation Copyright Permission

- A non-exclusive, irrevocable, royalty-free copyright permission is granted by OneSpin Solutions 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.
Introduction
A Path of SystemC Discovery

• OneSpin working in SystemC space for about 2 years
  – Cooperating with several large users on Formal Verification for the HLS flow

• In this time we have learned a lot about tool development and usage issues in this SystemC segment

• We would like to share some of the things we have seen and make a few suggestions
Focus on abstract hardware

- **Algorithm Components**
  - Data & control for high level synthesis

- **Configurable IP**
  - Greater configurability through synthesis

- **RTL Design**
  - RTL flow with extra testing

- **Verification Stimulus**
  - Testbenches often for data generation, etc

- **Processor Architecture**
  - Fast method for extensive processor verification

- **Virtual Platforms**
  - Software test platform
SystemC High Level Synthesis
Important SystemC Sweet Spot

- High-Level Synthesis SystemC Market
  - Possibly larger than generally realized
  - Focus:
    Image and communications DSP algorithms
  - SystemC provides benefits over HDLs
  - Appears to be an expanding area
  - Many more doing C refinement without HLS

- Companies working in this area
  - Toshiba
  - Fujitsu
  - Sony
  - NEC
  - MEE
  - Intel
  - Qualcomm
  - BlueWire
  - NXP (possibly)
  - ST (possibly)
  - ARM
  - Infineon
  - Samsung
  - Huawei
  - Broadcom
  - Canon
  - And plenty of others

C++/SystemC

Verilog/VHDL

HLS Model

- C++/SystemC Loosely Timed MicroArchitecture
- SystemC Cycle Accurate Model

High Level Synthesis (HLS)

Verilog / VHDL RTL

RTL Synthesis P&R

Device
C++/SystemC Abstraction
Range of Issues

- Proprietary library issues
  - HLS libraries required by other tools to read these designs

- SystemC hardware coding issues, e.g.
  - No undefined (X) signal state
  - Race conditions between threads

- HLS algorithm coding issues, e.g.
  - Number system verification
  - Operation hard to comprehend
OneSpin in the HLS Flow

Formal Being Used to Solve These Issues

Automated Formal Verification

SystemC/C++ Code Apps
Specialized HLS Apps

Stratus High-Level Synthesis

Verilog/VHDL RTL Model

Assertion-Based Verification

SystemC Golden Model

SVA C Asserts
C++/SystemC HLS Relies on Proprietary Libraries
Goes Against Ideals of the Standard

- HLS tools require additional language capabilities
- Currently these are being provided in proprietary libraries
  - For good reason; we’re not criticizing HLS vendors
- Synthesis subset committee not really providing answer
- For greater HLS ecosystem, this issue needs to be solved
Algorithmic Example
Varied Bit Widths in DSP Function

“Generalized” Filter implementation options
– hardware structure below or MAC working with buffer

Different bit widths due to coefficient sizes may be used to optimize design
Overflow checking for fixed floating points

- Assigning different width types may loose precision
  - Saturation mode can be set, default is to cut (possibly significant) bits
  - More confusion than with integer types, because of 2 parameters: bit width and non-fractional part
  - Non-fractional part important
  - Operations are value preserving
    - E.g. 16bit multiplied by 13 bit gives 29 bits
    - Converting unsigned to signed adds one bit
  - Truncation is done only in one operation: assignment
  - Why not place check that non-fractional part is preserved depending on the rounding mode
  - Defines could control whether overflows shall be checked, and whether they should result in an abort or just a message

- Could be extended to sc_(u)int, too
Examples

• \texttt{sc\_(u)fixed\langle W1,I1\rangle} is assigned to \texttt{sc\_(u)fixed\langle W,I\rangle}

\begin{verbatim}
sc ufixed\langle 8,8 \rangle a, b, c; * Default sc o_mode is SC_WRAP
    When using SC_WRAP, we should check for overflow
sc ufixed\langle 9,9 \rangle d;
a = b+c; => FAIL(if b+c > 2^{8})
a = sc ufixed\langle 8,8 \rangle (b+c); => NO FAIL
a = sc ufixed\langle 8,8,SC_TRN,SC_SAT \rangle (b+c); => NO FAIL
d = b+c; => NO FAIL

sc ufixed\langle 8,8,SC_TRN,SC_SAT \rangle e; * If sc o_mode is other than SC_WRAP,
    overflow checking is not required
e = b+c; => NO FAIL

sc ufixed\langle 8,8 \rangle f(b+c); => FAIL
sc ufixed\langle 8,8 \rangle g = 500; => FAIL
\end{verbatim}
Overflow when assigning to sc_ufixed

if(I<I1 && O_MODE==SC_WRAP) {
    if(a.range(W1-1,W1-(I1-I))!=0) assert(false);
}

If non-zero, overflow occurs

Bigger than 0
Overflow when assigning to `sc_fixed`

```
if(I<I1 && O_MODE==SC_WRAP) {
    if(a>=0) {
        if(a.range(W1-1,W1-(I1-I+1))!=0) assert(false);
    } else {
        if(a.range(W1-1,W1-(I1-I+1))!=sc_biguint<I1-I+1>(-1))
            assert(false);
    }
}
```
Special cases

- $I > W$ or $I_1 > W_1$
- $I < 0$ or $I_1 < 0$
  - No overflow possible, as no non-fractional part

- Singularity for `sc_fixed`, $I_1 == 1$ and $I == 0$ (only sign bit)

```cpp
sc_fixed<1,1> a; // W1=1, I1=1, a is 0 or -1
sc_fixed<1,0> b; // W=1, I=0, b is 0 or -0.5
b = a; // If a is -1, OVERFLOW occurs
```
SystemC Imposes Code Structure Can Lead to Hard-to-Verify Issues

Classical SystemC coding issues
- Lack of X state hides potential instability
- Write-write-race between threads
- Arithmetic overflow on data path
- Hard to verify algorithm versus spec
- Port-size mismatch at interface
- Array-out-of-bounds access on buffer
- Dead code and unreachable FSM states
- Packet loss on data transfer
Initialization

• SystemC has (due to it’s mother language C++) automatic initialization
  – But: Synthesis semantics specifies that initializations from module constructor shall be ignored
  – This leads to simulation-synthesis mismatches
  – Why not make this explicit by adding a template parameter to at least `sc_out`/`sc_signal` specifying that a random value should be used for initialization
  – This makes missing initialization problems visible much earlier in the verification flow which are currently difficult to catch

• Proposal:
  – Add a template parameter to `sc_out`, `sc_signal` specifying the reset behavior
  – The default for this parameter shall be the current behavior
  – Add `sc_signal_inout_if::write` registering a write
  – Add `sc_signal_inout_if::read` delivering a random value before the first write if the reset behavior is set to synthesis semantics
Arrays

• SystemC has out-of-bounds checking for bit vectors, \texttt{sc\textunderscore(u)\textunderscore int}, ...
  – But for vectors, good old C-arrays must be used
    • In C++ Software development, C-Arrays are superseded by \texttt{std\textunderscore :\textunderscore vector}
    • \texttt{std\textunderscore :\textunderscore vector} is dynamic, thus not suitable for hardware
    • bounds checking is available – either by using \texttt{.at()} or by optional debugging code

• Why no \texttt{sc\textunderscore array} class ?
  – Same interface as C-Arrays
  – (Optional) Bounds checking
  – Would make hard to find out-of-bounds errors much easier to find
  – In-line with other \texttt{sc\textunderscore *} classes doing bounds checking
  – Operator \texttt{[]} asserts on OOB
  – Function \texttt{at()} throws exception on OOB
Summary
A Path of SystemC Discovery

• HLS language issues restricting proliferation

• SystemC verification issues driving indirect, post-synthesis verification
  – Eliminates many of the advantages of HLS

• A few enhancements could make a difference
  – Proprietary code
  – Coding restrictions
  – Algorithmic design