SystemC/TLM and Language Standards (C++11/14)

Ralph Görgen, OFFIS

accellera SYSTEMS INITIATIVE

SYSTEMC™ EVOLUTION DAY
MAY 3, 2016 | MUNICH | GERMANY

Presentation Copyright Permission

– A non-exclusive, irrevocable, royalty-free copyright permission is granted by OFFIS e.V. 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.

C++ Language Development

- New C++ Standards C++11 and C++14

  C++ → Modern C++

- C++11
  - Many new features and improvements
- C++14
  - Slight improvements and fixes for C++11
- Both fully backwards compatible
  - (except auto storage class specifier)

Modern C++

- Improved productivity
  - auto; lambdas; range-based for; …
- Improved readability/maintainability
  - User defined literals; nullptr; inline member initialisation; …
- Improved safety
  - Strongly typed enums; delete, default, override, final, …
  - shared_ptr, unique_ptr, …
- New features
  - Variadic templates; constexpr; multi-thread + memory model; RValue reference and move; …

→ Better software quality, performance and safety

C++ Basis for SystemC

SystemC LRM (IEEE 1666-2011):

"[…]

This standard shall be used in conjunction with the following publications:

- ISO/IEC 14882:2003, Programming Languages—C++

[…]"

→ Combination of SystemC and Modern C++?

Modern C++ and SystemC?

Question:

How can …

- SystemC and TLM designers and
- Methodology library providers

… benefit from Modern C++?

Modern C++ and SystemC

- Modern C++ for modelling
- Modern C++ for standard and API extensions
- Modern C++ for PoC library implementation
- Modern C++ for synthesis

Modern C++ and SystemC

- **Modern C++ for modelling**
- Modern C++ for standard and API extensions
- Modern C++ for PoC library implementation
- Modern C++ for synthesis

Example: Auto und range-based for

```
for ( sc_core::sc_vector<
        sc_core::sc_export <
          sc_core::sc_signal_out_if<int> > >::iterator
    it = status_v_.begin();
    it < status_v_.end();
    ++it ) {  ...
```

```
for ( auto&& it : status_v_ ) {  ...
```

Example: Inline initialization and Lambda

```
SC_MODULE(adder)
{
  sc_in<uint8_t> a{"a"}, b{"b"};
  sc_out<sc_uint<9> > sum{"sum"};
  SC_CTOR(adder)
  {
    auto ph =
      sc_spawn([&](){ for(;;){ wait(a|b); sum = a + b }});
  }
};
```

Thanks to Roman Popov and David Black

Modern C++ for Modelling

- **Allow designer to use Modern C++**
- In principle no problem
- Depends on compiler used
- Limited interoperability
  - Compiler versions in EDA tools

- **Is this true?**
- **Any hidden traps?**
- **Should we force support via IEEE 1666?**

Modern C++ and SystemC

- Modern C++ for modelling
- **Modern C++ for standard and API extensions**
- Modern C++ for PoC library implementation
- Modern C++ for synthesis

Example: std::initializer_list in sc_vector

- Additional Constructor for sc_vector

```
template < typename T >
class sc_vector : //...
{
    sc_vector( std::initializer_list< T* > elements );
    // ...
```

- Initialization with list of element pointers

```
sc_core::sc_vector< my_mod_t > v =
  { new my_mod_t("ab"), new my_mod_t("cd„, 1)
  , new my_mod_t("ef", 11, 22) };
```

Example: Simplified Syntax for Thread Creation with Lambda

```
#define ASSIGN(sensitive,equation) \
  assigns.push_back(sc_spawn( \
   [&](){ for(;;){ wait(sensitive); equation }})

ASSIGN( a_sig|b_sig, y_sig = a_sig + b_sig; );
ASSIGN( clk.pos(), y_sig = a_sig + b_sig; );

// sc_method ( sensitivity_list , process_handle )
sc_method sum ( {a,b,c} , [&]() { sum = a + b + c; } };);
```

Thanks to David Black and Roman Popov

Example: Inline binding

```
#define SC_CTOR(user_module_name) \
  typedef std::function<void(user_mod_n& self)> bind_f_t; \
  typedef user_mod_n SC_CURRENT_USER_MODULE; \
  user_mod_n (sc_module_name, bind_f_t bindf= 0) \
  { if (bindf) { bindf(*this); }
#define BIND_INST(module_type) \
  [&](module_type& i)

SC_MODULE(adder_test)
{
  sc_signal<uint8_t> a{"a"}, b{"b"};
  sc_signal<sc_uint<9>> sum{"sum"};
  adder add_inst{"add_inst", BIND_INST(adder)
      { i.a(a); i.b(b); i.sum(sum); }};
...
```

Thanks to Roman Popov

Example: Move for SystemC Objects

- Implement move semantics for SystemC objects
- Allows much more freedom in handling non-copyable objects
- Requires some discussion about semantics

```
class sc_object {
  ...
  // Move COnstructor
  sc_object( sc_object&& );
  ...
```

Examples: User Defined Literals

- Literals for sc_time, …

```
sc_time operator""_ns(int t) // user defined
{ return sc_time(t,SC_NS); }

wait(10.5_ns);
```

Modern C++ for Standard and API Extensions

- Forces compiler support for C++14 for all SystemC
  - In EDA tools?
- Reference to ISO C++14 Standard in IEEE 1666?
- How to handle older compilers?
  - Opt-Out of Modern C++ features?
  - Recommend SystemC 2.3?
- Most important API extensions?
  - Proposals welcome as soon as questions above are solved
  - Discussion in LWG and Accellera Forum

Modern C++ and SystemC

- Modern C++ for modelling
- Modern C++ for standard and API extensions
- **Modern C++ for PoC library implementation**
- Modern C++ for synthesis

Modern C++ for PoC library implementation

- Allow Modern C++ for implementation of future SystemC library elements (not API relevant)?
- Refactoring of SystemC library?
  - auto, range-based for, …
  - Add overrides, final, delete, …
  - Generic smart pointers
  - Multi-threading
- Requires Modern C++ compiler support for all SystemC
  - No Opt-Out possible

Modern C++ and SystemC

- Modern C++ for modelling
- Modern C++ for standard and API extensions
- Modern C++ for PoC library implementation
- **Modern C++ for synthesis**

Modern C++ for Synthesis

- Extension of Synthesis Subset for Modern C++
- Draft available in SystemC SWG
  - sc_vector, std::array, …
  - constexpr
  - User-defined literals, binary literals

- Synthesis Subset “beyond” SystemC Standard?

Compiler Overview for C++14

- GCC:
  - Starts with GCC 4.4
  - Most C++11 features in GCC 4.7
  - Most C++14 features in GCC 4.9
    - Relaxed constexpr in GCC 5.0
- Clang
  - Many features in Clang 2.9 (e.g. auto)
  - Most features in Clang 3.1 (e.g. lambda)
  - All features in Clang 3.3
- MS Visual C++
  - Starts with MSVC 2010
  - Most features in MSVC 2013
  - Nearly all in MSVC 2015

References

- What C++011 means to SystemC?
  - David Black, NASCUG 2012
  - http://nascug.org/events/17th/black_cpp11_2_27_2012.pdf
- Problems with SystemC syntax
  - Roman Popov, Accellera Forum
  - forums.accellera.org/topic/5472-problems-with-systemc-syntax-improvment-request