Updating to Modern SystemC

David C.Black, Doulos

DOULOS

56 DESIGN AUTOMATION CONFERENCE
FROM CHIPS TO SYSTEMS – LEARN TODAY, CREATE TOMORROW

SYSTEMC™

Motivation

- Much of today's SystemC is based on the older C++ versions of 1998 or 2003.
- C++ changed considerably starting in 2011.
- Newer versions of the SystemC standard itself, including the recent CCI standard now require C++11 as a minimum.
- Older clunky looking SystemC code can now be written more intuitively and with less effort.
- All major C++ compilers support at least the 2014 standards leaving no excuses.

Main idea

- SystemC coding need not be so painful due to older C++98 limitations.
- Modern C++ includes the following features:
  - Syntax is more intuitive and fun
  - Can be written more quickly
  - Syntax provides mechanisms safety
  - Provides performance improvements
- Bottom-line: SystemC can be more productive when C++11/14 versions are adopted.

What's all the fuss about?

Simplicity, safety and performance

- Less is more
  - Less typing - easier to read, less to mistype, simplicity.
- New syntax to specify and measure intent
  - Compiler catches more problems earlier
- Mechanisms to avoid unnecessary operations
  - Faster by reducing construction/destruction
  - Basic containers avoid overhead

How has C++ changed?

The not so short list

- User-defined literals (eg. 1sec, 2kg, 8mm, 1.3v, 0.2uA)
- Compile-time functions and checks for better safety
- Uniform initialization syntax, constructor delegation
- Automatic type inference, Ranged-for loops and lambdas
- Real null pointer, scoped enumerations
- Virtual inheritance rules check signatures
- Explicit deletion and default implementation
- Containers designed for speed

C++ Standard Library Additions

- std::array
- std::chrono
- std::condition_variable
- std::forward_list
- std::functional
- std::future
- std::initializer_list
- std::memory
- std::mutex
- std::random
- std::ratio
- std::recursive
- std::regex
- std::thread
- std::tuple
- std::unordered_map
- std::unordered_set

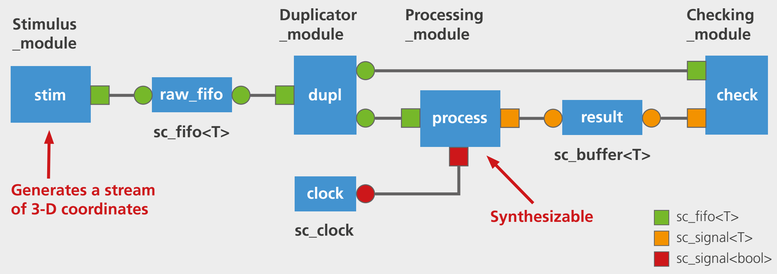
Example Code

- See block diagram to right and selected snippets
- https://github.com/dcblack/ModernSystemC.git

Summary

- C++11/14 is worth learning and applying to any new SystemC modeling efforts.
- Unavoidable - Required for CCI and future SystemC.
- A few additional features are proposed for addition to the base SystemC language.
- Deeper knowledge can be learned through:
  - Studying book/papers/presentations (will be provided)
  - Modern C++ training
  - Modern SystemC training

Block Diagram




```cpp
/**
 * @file top.hpp
 */
#ifndef TOP_HPP
#define TOP_HPP
#include "common.hpp"
#include <systemc>
#include <memory>
struct Stimulus_module;      struct Duplicator_module;
struct Processing_module;    struct Checker_module;

struct Top_module : sc_core::sc_module
{
  Top_module( sc_core::sc_module_name instance );
  ~Top_module( void );
  std::unique_ptr<Stimulus_module>   stim;
  std::unique_ptr<Duplicator_module> dupl;
  std::unique_ptr<Processing_module> process;
  std::unique_ptr<Checker_module>    check;
  std::unique_ptr<sc_core::sc_clock> clock;
  sc_core::sc_fifo<RawData_t>        raw_fifo      { "raw_fifo", FIFO_DEPTH };
  sc_core::sc_buffer<FixedPt_t>      result_buffer { "result_buffer" };
};
#endif/*TOP_HPP*/
```

```cpp
/**
 * @file  top.cpp
 * @brief Top-level interconnect implementation
 */
#include "top.hpp"
#include "stimulus.hpp"
#include "duplicator.hpp"
#include "processing.hpp"
#include "checker.hpp"
#include "sc_cxx11.hpp"
using namespace sc_core;

//------------------------------------------------------------------------------
Top_module::Top_module( sc_module_name instance ) //< Constructor
{
  /**
   * Instantiate
   */
  stim    = std::make_unique<Stimulus_module>   ( "stim"    );
  dupl    = std::make_unique<Duplicator_module> ( "dupl"    );
  process = std::make_unique<Processing_module> ( "process" );
  check   = std::make_unique<Checker_module>    ( "check"   );
  clock   = std::make_unique<sc_clock>          ( "clock", 10_ns );
  /**
   * Connect
   */
  stim->rawout_port   .bind( raw_fifo         );
  dupl->input_port    .bind( raw_fifo         );
  process->input_port .bind( dupl->out1_xport );
  process->output_port.bind( result_buffer    );
  process->clk_port   .bind( *clock           );
  check->result_port  .bind( result_buffer    );
  check->rawin_port   .bind( dupl->out2_xport );
}

//------------------------------------------------------------------------------
Top_module::~Top_module( void ) = default;
```

```cpp
/**
 * @file stimulus.hpp
 * @brief Generates stimulus to test the design
 */
#ifndef STIMULUS_HPP
#define STIMULUS_HPP
#include "common.hpp"
#include <systemc>
struct Stimulus_module : sc_core::sc_module
{
  Stimulus_module( sc_core::sc_module_name instance );
  ~Stimulus_module( void );
  sc_core::sc_fifo_out<RawData_t>  rawout_port { "rawout_port" };
private:
  void stimulus_thread( void );
};
#endif/*STIMULUS_HPP*/
```

```cpp
/**
 * @file stimulus.cpp
 */
#include "stimulus.hpp"
#include "objection.hpp"
#include "report.hpp"
#include <random>
using namespace sc_core;
namespace { char const * const MSGID{ "/Doulos/Example/Modern/Stimulus_module" }; }

//------------------------------------------------------------------------------
Stimulus_module::Stimulus_module( sc_module_name instance ) //< Constructor
{
  SC_HAS_PROCESS( Stimulus_module );
  SC_THREAD( stimulus_thread );
}

//------------------------------------------------------------------------------
Stimulus_module::~Stimulus_module( void ) = default;

//------------------------------------------------------------------------------
void Stimulus_module::stimulus_thread( void )
{
  Objection generating_data( name() );
  unsigned int seed = 1;
  std::default_random_engine generator(seed);
  std::uniform_real_distribution<double> distribution (-128.0,128.0);
  std::vector<double> directed = { 0.0, 1.0, 2.0, 3.0, 4.0, 5.0 };
  size_t samples = directed.size()*directed.size()*directed.size();
  REPORT( INFO, "Sending " << samples << " directed samples" );
  for( auto const& x : directed ) {
    for( auto const& y : directed ) {
      for( auto const& z : directed ) {
        Coordinate xyz{ x, y, z };
        rawout_port->write( xyz );
        INFO( DEBUG, "Sent " << xyz );
      }
    }
  }
  samples = 1000;
  REPORT( INFO, "Sending " << samples << " random samples" );
  for( size_t n=0; n<samples; ++n ) {
    Coordinate xyz = { distribution(generator), distribution(generator), distribution(generator) };
    rawout_port->write( xyz );
    INFO( DEBUG, "Sent " << xyz );
  }
  REPORT( INFO, "Stimulus complete" );
}
```

```cpp
/**
 * @file duplicator.hpp
 * @brief Takes a single input and copies it out to two downstream
 * connections.
 */
#ifndef DUPLICATOR_HPP
#define DUPLICATOR_HPP
#include "common.hpp"
#include <systemc>

struct Duplicator_module : sc_core::sc_module
{
  Duplicator_module( sc_core::sc_module_name instance );
  ~Duplicator_module( void ) = default;
  sc_core::sc_fifo_in <RawData_t>                          input_port { "input_port" };
  sc_core::sc_export<sc_core::sc_fifo_in_if<RawData_t>> out1_xport { "out1_xport" };
  sc_core::sc_export<sc_core::sc_fifo_in_if<RawData_t>> out2_xport { "out2_xport" };
private:
  sc_core::sc_fifo<RawData_t>                              out1_fifo  { "out1_fifo", FIFO_DEPTH };
  sc_core::sc_fifo<RawData_t>                              out2_fifo  { "out2_fifo", FIFO_DEPTH };
};
#endif/*DUPLICATOR_HPP*/
```

```cpp
/**
 * @file duplicator.cpp
 */
#include "duplicator.hpp"
#include "report.hpp"
using namespace sc_core;
namespace { char const * const MSGID{ "/Doulos/Example/Modern/Duplicator_module" }; }

//------------------------------------------------------------------------------
Duplicator_module::Duplicator_module( sc_module_name instance ) //< Constructor
{
  // Connectivity
  out1_xport.bind( out1_fifo );
  out2_xport.bind( out2_fifo );
  // Process
  auto m = this;
  sc_spawn( [m]{
      for(;;) {
        RawData_t v = m->input_port->read();
        INFO( DEBUG, "Duplicating " << v );
        m->out1_fifo.write( v );
        m->out2_fifo.write( v );
      }
    }, "duplicator_thread" );
}
```

```cpp
/**
 * @file processing.hpp
 * @brief Synthesizable module that computes the magnitude of a 3-dimensional vector.
 */
#ifndef PROCESSING_HPP
#define PROCESSING_HPP
#include "common.hpp"
struct Processing_module : sc_core::sc_module
{
  Processing_module( sc_core::sc_module_name instance );
  ~Processing_module( void );
  sc_core::sc_fifo_in <RawData_t>    input_port  { "input_port"  };
  sc_core::sc_out<FixedPt_t>         output_port { "output_port" };
  sc_core::sc_in<bool>               clk_port    { "clk_port"    };
private:
  void processing_thread( void );
};
#endif/*PROCESSING_HPP*/
```

```cpp
/**
 * @file processing.cpp
 */
#include "processing.hpp"
#include "fpsqrt.hpp"
using namespace sc_core;
namespace { char const * const MSGID{ "/Doulos/Example/Modern/Processing_module" }; }

//------------------------------------------------------------------------------
Processing_module::Processing_module( sc_module_name instance ) //< Constructor
{
  SC_HAS_PROCESS( Processing_module );
  SC_CTHREAD( processing_thread, clk_port.pos() );
}

//------------------------------------------------------------------------------
Processing_module::~Processing_module( void ) = default;

//------------------------------------------------------------------------------
void Processing_module::processing_thread( void )
{
  RawData_t v;
  FixedPt_t x, y, z, sum2, magnitude;
  wait(1);
  for(;;) {
    wait(1);
    if( input_port->nb_read( v ) ) {
      x = v.x();
      y = v.y();
      z = v.z();
      x *= x;
      y *= y;
      z *= z;
      sum2 = x + y +z;
      magnitude = FP::fpsqrt( sum2 );
      output_port->write( magnitude );
    }
  }
}
```

```cpp
/**
 * @file checker.hpp
 */
#ifndef CHECKER_HPP
#define CHECKER_HPP
#include "common.hpp"
#include <systemc>
#include <tlm>
struct Checker_module : sc_core::sc_module
{
  Checker_module( sc_core::sc_module_name instance );
  ~Checker_module( void ) = default;
  // Ports
  sc_core::sc_fifo_in<RawData_t>   rawin_port  { "rawin_port"  };
  sc_core::sc_in<FixedPt_t>        result_port { "result_port" };
private:
  // Local channels
  tlm::tlm_fifo<FixedPt_t>         expect_fifo { "expect_fifo", -2 };
  tlm::tlm_fifo<FixedPt_t>         result_fifo { "result_fifo", -2 };
  // Attributes
  bool                             anticipate_reset { false };
  size_t                           verified_count   { 0 };
  size_t                           mismatches       { 0 };
  // Processes and overrides
  void received_method( void );
  void checker_thread( void );
  void end_of_simulation( void ) override;
};
#endif/*CHECKER_HPP*/
```

```cpp
/**
 * @file checker.cpp
 */
#include "checker.hpp"
#include "objection.hpp"
#include <cmath>
#include <string>
#include "fpsqrt.hpp"
#include "report.hpp"
using namespace sc_core;
namespace { char const * const MSGID{ "/Doulos/Example/Modern/Checker_module" }; }

//------------------------------------------------------------------------------
Checker_module::Checker_module( sc_module_name instance ) //< Constructor
{
  SC_HAS_PROCESS( Checker_module );
  SC_METHOD( received_method );
    sensitive << result_port;
    dont_initialize();
  SC_THREAD( checker_thread );
}

//------------------------------------------------------------------------------
void Checker_module::received_method( void ) {
  result_fifo.put( result_port->read() );
}

//------------------------------------------------------------------------------
void Checker_module::checker_thread( void ) {
  const FixedPt_t allowable( 1.0/64.0 );
  FixedPt_t expect_value, actual_value, difference;
  REPORT( INFO, "Preparing checker" );
  for(;;) {
    // Is anything expected?
    RawData_t v = rawin_port->read();
    FixedPt_t expect_value = FP::fpsqrt( v.x()*v.x() + v.y()*v.y() + v.z()*v.z() );
    {
      Objection verifying{ name() };
      // Wait for corresponding result
      actual_value = result_fifo.get();
      // Compare result to expected
      difference = expect_value - actual_value;
      if( difference < 0 ) difference = -difference;
      if( difference > allowable ) {
        REPORT( ERROR, "Expect:" << expect_value.to_string()
              << " Actual:" << actual_value.to_string()
              << " at " << sc_time_stamp() << " FAILURE"
        );
        ++mismatches;
      }
      ++verified_count;
    }
  }
}

//------------------------------------------------------------------------------
void Checker_module::end_of_simulation( void ) {
  INFO( ALWAYS, "Verified " << std::to_string( verified_count ) << " transactions\n"
            << "- Found " << std::to_string( mismatches ) << " mismatches\n"
            << "- Objected " << Objection::total() << " times" );
}
```

Summary

- C++11/14 is worth learning and applying to any new SystemC modeling efforts.
- Unavoidable - Required for CCI and future SystemC.
- A few additional features are proposed for addition to the base SystemC language.
- Deeper knowledge can be learned through:
  - Studying book/papers/presentations (will be provided)
  - Modern C++ training
  - Modern SystemC training

www.doulos.com/systemc