LG2: Lecture Group 2: SystemC.

Topic: SystemC Overview

- LG2.1 - SC SystemC Components
- LG2.2 - SC Example (Counter)
- LG2.3 - SC SystemC Structural Netlist
- LG2.4 - SC SystemC Signals
- LG2.5 - SC Threads and Methods
- LG2.6 - SC Blocking and Eventing
- LG2.7 - SC Pre-TLM S/W Style Channels
- LG2.8 - SC SystemC Plotting and Gui
- LG2.9 - SC SystemC Synthesis

LG2.1 - SystemC Components

SystemC is a free library for C++. `www.systemc.org`

It includes (at least):

- A module system with inter-module channels,
- An eventing and threading kernel,
- Compute/commit signals as well as other forms of channel,
- A library of fixed-precision integers,
- Plotting output functions,
- A separate transactional modelling library.

Originally it served for architectural exploration.

Now being used for fast, transactional modelling.

Also now being used as an implementation language with its own synthesis tools.

Problem: hardware engineers are not C++ experts but they can be faced with complex or advanced C++ error messages when they misuse the library.

Benefit: General-purpose behavioural C code, including application code and device drivers, can all be modelled in a common language.

LG2.2 - Example (Counter)

```
SC_MODULE(mycounter)
{
   sc_in<bool> clk, reset;
   sc_out<bool> out;
   sc_int<10> sum;

   void m()
   {
      sum =  (reset) ? 0: (sum.read()+1); // Use .read() since channel contains a signal.
   }

   SC_CTOR(mycounter)
     { SC_METHOD(m);
       sensitive << clk.pos();
     }

}
```

SystemC enables a user class to be defined using the the SC_MODULE macro.

Modules inherit various attributes appropriate for an hierarchic hardware design including an instance name, a type name and channel binding capability..

LG2.3 - SystemC Structural Netlist

```
SC_MODULE(shiftreg)  // Two-bit shift register
{   sc_in  <bool>  clk, reset, din;
    sc_out <bool> dout;

    sc_signal <bool> q1_s;
    dff dff1, dff2;      // Instantiate FFs

    SC_CTOR(shiftreg) : dff1("dff1"), dff2("dff2")
    {   dff1.clk(clk);
        dff1.reset(reset);
        dff1.d(din);
        dff1.q(q1_s);

        dff2.clk(clk);
        dff2.reset(reset);
        dff2.d(q1_s);
        dff2.q(dout);
    }
};
```

SystemC channel provides general purpose interface between components.

The sc_signal channel should be used to obtain the compute/commit paradigm. Avoids non-determinacy from races.

Other channels include FIFOs and semaphores.

A user-defined channel type may instantiate user code in the form of other SystemC components

LG2.4 - SystemC Channels and Signals

All primitive channels connect modules together.

Predefined channels include buffer, fifo, signal, mutex semaphore and clock.

RTL-style compute/commit is provided by the SystemC signals.

A signal is an abstract (templated) data type that has a current and next value.

Reads are of the current value. Writes are to the next value.

```
  int nv;
  sc_out < int > data;
  sc_signal <int> mysig;
  ...
        nv += 1;
        data = nv;
        mysig = nv;
        printf("Before nv=%i, %i %i\n'', nv, data.read(), mysig.read());
        wait(10, SC_NS);
        printf("After  nv=%i, %i %i\n'', nv, data.read(), mysig.read());
  ...

Before nv=96, 95 95
After  nv=96, 96 96
```

Instantiable channel provides implementations of read, write, update and value_changed.

LG2.4c - SystemC Channels and Signals Continued

Can pass an abstract datatype along channel.

```
sc_signal <bool> mywire;

struct capsule
{ int ts_int1, ts_int2;
  bool operator== (struct ts other)
  { return (ts_int1 == other.ts_int1) && (ts_int2 == other.ts_int2); }
  ...
  ...  // Also some others
};

sc_signal <struct capsule> myast;
```

For basic types, int, sc_int, the operations are already in the library.

```
void mymethod()  { .... }
SC_METHOD(mymethod)
sensitive << mywire.pos();
```

When the scheduler blocks with no more events in the current time step, the pending new values are committed to the visible current values.

Future topic: TLM: wiring components together with methods instead of variables.

LG2.5 - Threads and Methods

The user code in an SC_MODULE is run either as an SC_THREAD or an SC_METHOD.

An SC_THREAD has a stack and is allowed to block.

An SC_METHOD is just an upcall from the event kernel and must not block.

Use SC_METHOD wherever possible, for efficiency.

Use SC_THREAD where important state must be retained in the program counter from one activation to the next or when asynchronous active behaviour is needed.

```
SC_MODULE(mydata_generator)
{
  sc_out <int> data;
  sc_out <bool> req;
  sc_in <bool> ack;

  void myloop()
  {
    while(1)
    {
      data = data.read() + 1;
      wait(10, SC_NS);
      req = 1;
      do { wait(); } while(!ack.read());
      req = 0;
      do { wait(); } while(ack.read());
    }
  }
  SC_CTOR(mydata_generator)
  {
    SC_THREAD(myloop);
  }
}
```

LG2.6 - Blocking and Eventing

A thread can block for a given amount of time using the `wait` function of SystemC. (Clashes with Posix function of that name!).

Wait on arbitrary boolean condition is harder to implement owing to the compiled nature of C++:

- C++ does not have a reflection API that enables a user's expression to be re-evaluated by the event kernel.
- Yet we still want a reasonably neat and efficient way of passing an uninterpreted function.
- Chosen answer : the delayed evaluation class.

```
waituntil(mycount.delayed() > 5 && !reset.delayed());
```

Poor user had to just insert the `delayed` keyword where needed and then ignore it when reading the code.

Now removed: use

```
do { wait(); }  while(!((mycount > 5 && !reset)));
```

LG2.7 - Pre-TLM S/W Style Channels

```
class write_if: public sc_interface
{ public:
  virtual void write(char) = 0;
  virtual void reset() = 0;
};

class read_if: public sc_interface
{ public:
  virtual void read(char) = 0;
};

class fifodevice: sc_module("fifodevice"),
public write_if, public read_if, ...
{
  void write(char) { ... }
  void reset() { ... }

  ...
}
```

```
SC_MODULE("fifo_writer")
{
  sc_port<write_if> outputport;
  sc_in <bool> clk;
  void writer()
  {
    outputport.write(random());
  }

  SC_CTOR(fifo_writer} {
     SC_METHOD(writer);
     sensitive << clk.pos();
  }
}


   fifodevice myfifo("myfifo");
   fifo_writer mywriter("mywriter");

   mywriter.outputport(myfifo);
```

Here a thread passes between modules, but modules are plumbed in Hardware/EDS netlist structural style.

LG2.8 - SystemC Plotting and Gui

We can plot to industry standard VCD files and view with gtkwave.

```
sc_trace_file *tf = sc_create_vcd_trace_file("tracefile");

// Now call:
// sc_trace(tf, <traced variable>, <string>);

sc_signal<int> a;
float b;
sc_trace(trace_file, a, "MyA");
sc_trace(trace_file, b, "MyB");

sc_start(1000, SC_NS);  // Simulate for one microsecond
sc_close_vcd_trace_file(tr);
return 0;
```

VCD can be viewed with *gtkwave*.

There are various X-windows interactive viewer libraries...

LG2.9 - SystemC Synthesis

We cannot compile general C/C++ programs to hardware. Must be finite state:

- all recursion bounded,
- all dynamic storage allocation outside of infinite loops,
- use only boolean logic and integer arithmetic,
- limited string handling,
- very limited standard library support,
- be explicit in the time/space fold/unroll.

Future topic: C-to-gates: generating hardware from software.

Products available : Catapult, SimVision, CoCentric, ... others.